Zynq UltraScale+ Device

Technical Reference Manual

UG1085 (v2.3.1) January 4, 2023

AMD
XILINX

Table of Contents

Chapter 1

Introduction

Introduction to the UltraScale Architecture

The Xilinx® UltraScale™ architecture enables multi-hundred gigabit-per-second levels of system performance with smart processing, while efficiently routing and processing data on-chip. UltraScale architecture-based devices address a vast spectrum of high-bandwidth, high-utilization system requirements by using industry-leading technical innovations, including next-generation routing, ASIC-like clocking, 3D-on-3D ICs, multiprocessor SoC technologies, and new power reduction features. The devices share many building blocks, providing scalability across process nodes and product families to leverage system-level investment across platforms.

All Zynq® UltraScale+ devices provide 64-bit processor scalability while combining real-time control hard engines for graphics, video, waveform, and packet processing capabilities in the programmable logic. Integrating an Arm®-based system for advanced analytics and on-chip programmable logic for task acceleration creates unlimited possibilities for applications including 5G Wireless, next generation ADAS, and Industrial Internet-of-Things.

The RFSoC devices are similar to the basic MPSoC devices with the addition of key RF subsystems for multi-band, multi-mode cellular radios and cable infrastructure (DOCSIS). The RFSoC devices combine the processing system with programmable logic located near RF-ADCs, RF-DACs, and soft-decision FEC (SD-FEC) units to enable a complete software-defined radio including direct RF sampling data converters, enabling CPRI™ and multi-gigabit Ethernet-to-RF on a single, highly programmable SoC.

Table 1-1 shows the main functional units and peripherals. For more information on the TRM, see References in Appendix A, Xilinx Documentation Navigator, and the Zynq UltraScale+ Documentation website [Ref 1].

Application Overview

Zynq UltraScale+ MPSoC is the Xilinx second-generation Zynq platform, combining a powerful processing system (PS) and user-programmable logic (PL) into the same device. The processing system features the Arm® flagship Cortex®-A53 64-bit quad-core or dual-core processor and Cortex-R5F dual-core real-time processor. In addition to the cost and integration benefits previously provided by the Zynq-7000 devices, the Zynq UltraScale+ MPSoC and RFSoC devices also provide these new features and benefits.

- Scalable PS with scaling for power and performance.
- Low-power running mode and sleep mode.
- Flexible user-programmable power and performance scaling.
- Advanced configuration system with device and user-security support.
- Extended connectivity support including PCIe®, SATA, and USB 3.0 in the PS.
- Advanced user interface(s) with GPU and DisplayPort in the PS.
- RF circuitry for with up to 16 channels of RF-ADCs and RF-DACs (RFSoC devices).
- Increased DRAM and PS-PL bandwidth.
- Improved memory traffic using Arm's advanced QoS regulators.
- Improved safety and reliability.

These new devices offer the flexibility and scalability of an FPGA, while providing the performance, power, and ease-of-use typically associated with ASICs and ASSPs. The range of the Zynq UltraScale+ family enables designers to target cost-sensitive and high-performance applications from a single platform using industry-standard tools. There are two versions of the PS; dual Cortex-A53 and quad Cortex-A53. The features of the PL vary from one device type to another. As a result, the Zynq UltraScale+ MPSoCs are able to serve a wide range of applications including:

- Automotive driver assistance, driver information, and infotainment.
- Broadcast camera.
- Industrial motor control, industrial networking, and machine vision.
- IP and smart camera.
- LTE radio and baseband.
- Medical diagnostics and imaging.
- Multifunction printers.
- Video and night vision equipment.
- Wireless radio.

- Single-chip computer.

System Block Diagram

The MPSoC and RFSoC devices consist of two major underlying sections PS and PL in two isolated power domains. PS acts as one standalone SoC and is able to boot and support all the features of the processing system shown in Figure 1-1 without powering on the PL.

The PS block has three major processing units.

- Cortex-A53 application processing unit (APU)—Arm v8 architecture-based 64-bit quad-core or dual-core multiprocessing CPU.
- Cortex-R5F real-time processing unit (RPU)—Arm v7 architecture-based 32-bit dual real-time processing unit with dedicated tightly coupled memory (TCM).
- Mali-400 graphics processing unit (GPU)—graphics processing unit with pixel and geometry processor and 64KB L2 cache (available in the EG and EV MPSoC devices).
- Video control unit (VCU)—video compression, decompression, and processing (available in the EV MPSoC devices).
- Radio frequency (RF)—up to 16 channels of RF-ADCs and RF-DACs (available in the RFSoC devices).

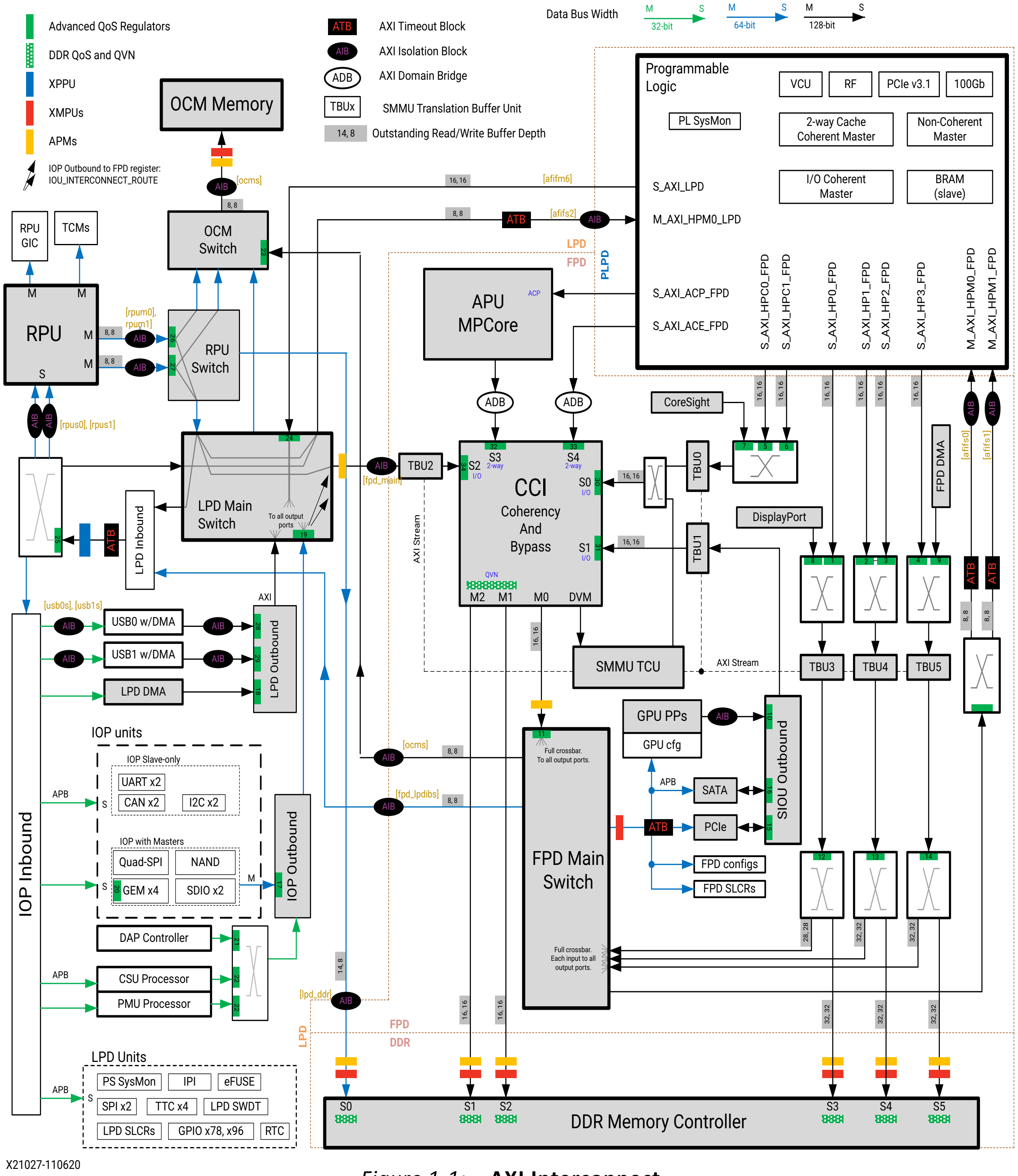


Figure 1-1: **AXI Interconnect**

Power Domains and Islands

There are four main power domains.

- Low-power domain (LPD).
- Full-power domain (FPD).
- PL power domain (PLPD).
- Battery power domain (BPD).

Each power domain can be individually isolated. The platform management unit (PMU) on the LPD facilitates the isolation of each of the power domains. Additionally, the isolation can be turned on automatically when one of the power supplies of the corresponding power domain is accidentally powered down. Since each power domain can be individually isolated, functional isolation (an important aspect of safety and security applications) is possible. See Figure 1-2.

Note: The voltages shown in Figure 1-2 are shown as a general guide. See *Zynq UltraScale+ MPSoC Data Sheet: DC and AC Switching Characteristics* (DS925) [Ref 2] for the device specifications.

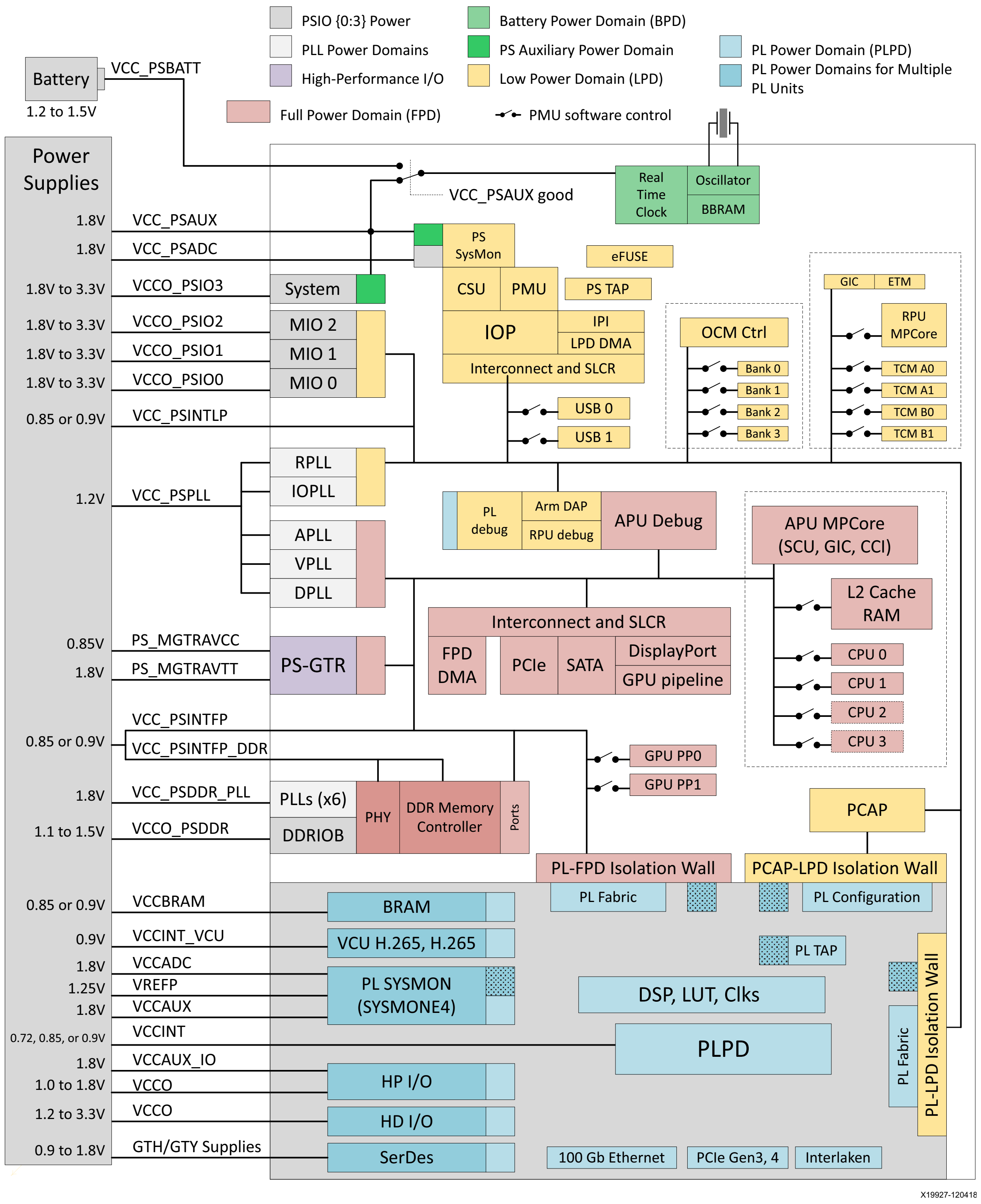


Figure 1-2: **Power Domains and Islands**

High-Speed Serial I/O

The SIOU peripherals (plus the USB controller) share four GTR transceivers in the PS. There are up to 16 GTY transceivers in the PL that are used with user-defined FPGA logic and the RF circuits in RFSoC devices.

GTR Transceivers

The four GTR transceiver channels are shared with five high-speed serial I/O peripherals; four from the SIOU in the FPD and the USB 3.0 controller based in the LPD. The controllers support the following protocols.

- PCI Express® integrated interface—PCIe™ base specification version 2.1.
- SATA 3.1 specification interface.
- DisplayPort interface—implements a DisplayPort source-only interface with video resolution up to 4k x 2k.
- USB 3.0 interface—compliant to USB 3.0 specification implementing a 5 Gb/s line rate.
- Serial GMII interface—supports a 1 Gb/s SGMII interface.

The PL includes three high-speed serial I/O peripherals. These interfaces are described in Chapter 36, PL Peripherals.

- PCI Express Integrated interface—PCIe base specification version 3.1 and 4.0.
- 100G Ethernet.
- Interlaken.

Figure 1-3 contrasts the location and I/O connectivity of all the high-speed serial I/O peripherals.

GTY Transceivers

The GTY transceivers transfer data up to 32.75 Gb/s, enabling 25G+ backplane designs with dramatically lower power per bit than previous generation transceivers.

The RFSoC devices expand the capabilities of the GTY transceivers to include higher performance PCIe and gigabit Ethernet-to-RF functionality. The transceivers support data rates for PCIe Gen3 and Gen4 (rev 0.5) in all devices. PCIe Gen4 x8 and Gen3 x16 endpoint and root port are supported in RFSoC devices.

For all devices, the transceivers support 150 Gb/s Interlaken and 100 Gb/s Ethernet (100G MAC/PCS), and enable simple, reliable support for Nx100G switch and bridge applications.

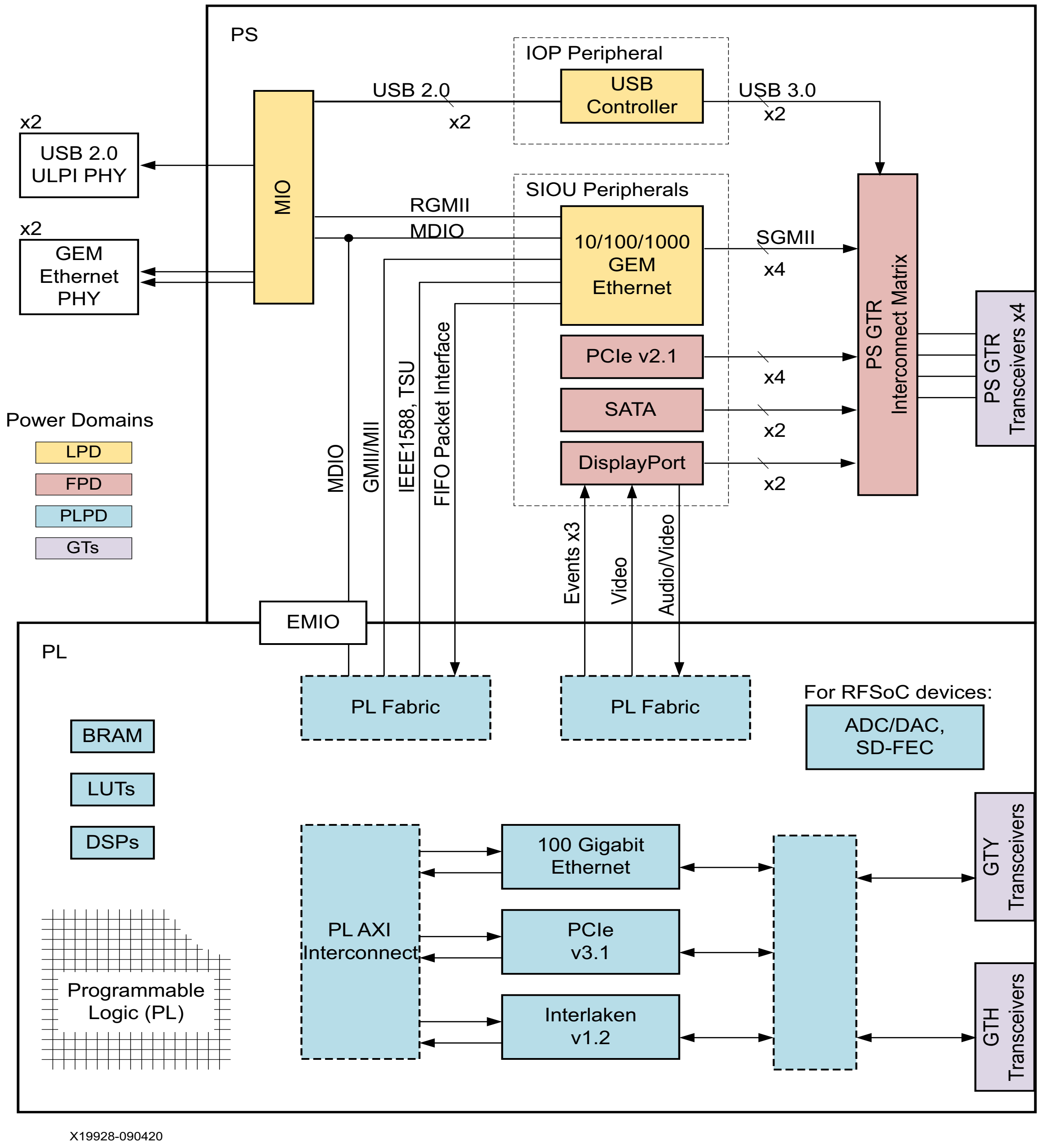


Figure 1-3: **High-Speed Serial I/O Block Diagram**

MIO and EMIO

The PS and PL can be coupled with multiple interfaces and other signals to effectively integrate user-created hardware accelerators and other functions in the PL logic that are accessible to the processors. They can also access memory resources in the processing system. The PS I/O peripherals, including the static/flash memory interfaces share a multiplexed I/O (MIO) of up to 78 MIO pins. The peripherals can also use the I/Os in the PL domain for many controllers. This is done using the extended multiplexed I/O interface (EMIO).

The I/O peripheral signal availability on MIO and EMIO is summarized in Table 2-7. The MIO pin multiplexing functionality is described in Chapter 28, Multiplexed I/O.

Platform Management and Boot

The PMU receives requests from other processors to power up and power down peripherals and other units by power sequencing nodes and islands. The PMU also enables and disables clocks and resets.

After a system reset, the PMU ROM pre-boot code initializes the system and the CSU ROM executes the first stage boot loader from the selected external boot device. The boot process configures the MPSoC platform as needed, including the PS and the PL.

After the FSBL execution starts, the CSU enters the post-configuration stage to monitor tamper signals from various sources in the system. The tamper response registers are listed at the bottom of Table 11-12.

The system includes many types of security, test, and debug features. The system can be booted either securely (boot image is either encrypted or authenticated, or encrypted and authenticated) or non-securely. Either of the following combinations can be implemented.

- Boot image is encrypted.
- Boot image is authenticated.
- Boot image is both encrypted and authenticated for the highest level of security.

The PL configuration bitstream can be applied securely or non-securely. The boot process is multi-stage and minimally includes the boot ROM and the first-stage boot loader (FSBL). Zynq UltraScale+ MPSoCs include a factory-programmed configuration security unit (CSU) ROM. The boot header determines whether the boot is secure or non-secure, performs some initialization of the system, reads the mode pins to determine the primary boot device, and loads the FSBL.

Optionally, the JTAG interface can be enabled to provide access to the PS and the PL for test and debug purposes.

Power to the PL can be optionally shut off to reduce power consumption. To further reduce power, the clocks and the specific power islands in the PS (for example, an APU power island) can be dynamically slowed down or gated off.

Functional Units and Peripherals

Table 1-1 lists and describes the main functional units and peripherals.

Table 1-1: **Functional Units and Peripherals**

Name	Description
APU MPCore	Application processing units: two or four 64-bit Cortex-A53 processors, supports four exception levels, NEON instructions, and single/double precision floating-point calculations, includes accelerator port (ACP) and AXI coherency extension (ACE), snoop-control unit (SCU), and L2 cache controller (CG devices are dual core, all others are quad).
RPU MPCore	Real-time processing units: dual 32-bit Cortex-R5F processor, Arm instruction set, dynamic branch prediction, redundant CPU logic for fault detection, 32/64/128-bit AXI interface to the PL for low-latency applications.
GPU	Graphics processing units: one geometry processor, two pixel processors, OpenGL ES 1.1 and 2.0, OpenVG 1.1, advanced anti-aliasing support (available in the EG and EV product families).
VCU	The video codec unit (VCU) provides multi-standard video encoding and decoding, including support for the high-efficiency video coding (HEVC) H.265 and advanced video coding (AVC) H.264 standards. The unit contains both encode (compress) and decode (decompress) functions, and is capable of simultaneous encode and decode. The VCU is included in the EV product family of the MPSoC devices.
RF	The RF-ADC, RF-DAC, and soft-decision FEC functions are located in the PL. These RF circuits enable software-defined radios using the direct RF sampling data converters to enable CPRI™ and gigabit Ethernet-to-RF functionality. The RF unit is included in the DR product family exclusively part of the RFSoC devices.
AMBA interconnect	AXI cache-coherent interconnect, interconnects belonging to two power domains, (central switch and low-power switch), processing system to programmable logic interface. APB buses for register access, AHB for some IOP masters.
CSU	Configuration security unit: triple redundant processor for controlling; supports secure and non-secure boot flows.
System interrupts	Processor, controller, and other system element interrupts. Inter-processor interrupts (IPI). Software generated interrupts. RPU and APU interrupts and system interrupts. Inter-processor interrupt, support for software generated interrupt and shared peripheral interrupt.
TTC	4x Triple Timer Counters: programmable 32-bit and 64-bit timers, programmable event counters.

Table 1-1: **Functional Units and Peripherals *(Cont'd)***

Name	Description
LPD and FPD DMA units	Programmable number of outstanding transfers, support for simple and scatter-gather mode, support for read-only and write-only DMA mode, descriptor prefetching, per channel flow control interface.
DDR memory controller	DDR3, DDR3L, DDR4, LPDDR4, up to two ranks, dynamic scheduling to optimize bandwidth and latency, error-correction code support in 32-bit and 64-bit mode, software programmable quality of service.
NAND memory controller	Complies with ONFI 3.1 specification, supports reset logical unit number, ODT configuration, on-die termination.
SPI controller	Full duplex operation, multi-master environment support, programmable master mode clock frequency, programmable transmission format.
Quad-SPI controller	Stacked and parallel modes, supports command queuing, supports 4/8 bit interface, 32-bit address support on AXI in DMA mode transfer.
CAN controller	Standard and extended frames, automatic retransmission on errors, four RX acceptance filters with enables, masks, and IDs.
UART controller	Programmable baud rate generator, 6/7/8 data bits, modem control signals.
I2C controller	I2C bus specification version 2.0 supported, normal and fast mode transfer, slave monitor mode.
SD/SDIO/eMMC controller	Data transfer in 1-bit or 4-bit mode, cyclic redundancy check for data and command, card insertion/removal detection.
GPIO	78 GPIO signals for device pins, 96 GPIO channels between PS and PL, programmable interrupt on individual GPIO channel.
PL peripherals	Peripherals present in the PL: • PCI Express rev 3.1 and 4.0. • Interlaken • 100G Ethernet • PL System Monitor • Video encoder/decoder (VCU is available in EV MPSoC devices). • High-speed transceivers (up to 32.75 Gb/s) • DisplayPort audio and video interface • RF I/O subsystem (RFSoC devices)
Platform management unit	System initialization during boot, management of power gating and retention states, management of sleep states, triple-redundant processor.
Clock system	Five independent system PLLs used as clock source for a few dozen clock generators for all the functional units and peripherals.
Reset system	Individual peripheral level reset generation, PS only reset.
Arm DAP controller	Access to debug access port and Arm CoreSight™ components.
Arm CoreSight debug components	Break-point and single stepping, AXI trace monitor to capture AXI transactions, CoreSight system trace macrocell (STM) captures software driven traces, CoreSight extension from the PL.
On-chip memory (OCM)	256KB RAM, very high throughput support on AXI interconnect, ECC support.
Tightly-coupled memory	Four TCM banks, each one is 64 KB.

Table 1-1: **Functional Units and Peripherals *(Cont'd)***

Name	Description
PS-GTR transceivers	Compliant with PCIe 2.0, USB 3.0, DisplayPort 1.2a, SATA 3.1, and SGMII protocols, internal PLL per lane to support multiple protocols, integrated termination resistors, BIST, and supports loopbacks as required by the supported protocols.
PCI Express rev 2.1	End Point and Root Port mode, Gen1 and Gen2 rates, MSI, MSI-X, and legacy interrupt support, AXI PCIe bridge, integrated four-channel fully-configurable DMA.
USB controller	USB 2.0/3.0 host, device, OTG, 5 Gb/s data rate, AXI master port with built-in DMA, power management feature, hibernation mode, simultaneous operation of USB 2.0 and 3.0.
SATA host controller	Compliant with the SATA 3.1 specification, supports 1.5G, 3G, and 6G line rates, compliant with the advanced host controller interface version 1.3. The controller has an embedded DMA that facilitates memory transfers.
DisplayPort interface	Source only controller with an embedded DMA controller that supports 1G or 2G transceiver lanes, supports real-time video and audio input from the PL.
Gigabit Ethernet controller and serial GMII (GEM)	IEEE Std 802.3-2008 compatible, full and half-duplex modes of operation, RGMII/SGMII interface support, MDIO interface, automatic discard frames with errors, programmable inter packet gap, full-duplex flow control. The controller has a built in DMA engine that can be used to transfer Ethernet packets from memory.
System protection units	Memory and peripheral partitioning and protection, TrustZone protection, error handling on permission violation/disallowed transactions, access control for a specific range of addresses, access control on a per-peripheral basis.

Device ID Codes

JTAG IDCODE

The device ID code uniquely identifies the major features and PS version of each device type. There are two ways to access the device ID code:

- IDCODE instruction in the PS TAP controller.
- Software readable CSU.IDCODE register.

The IDCODE read instruction is always available on the PSJTAG controller, even when it is disabled.

The software reads the same ID code as the PSJTAG interface. The CSU register set also includes the Version [PS_Version] bit field. This helps software to easily determine the version of the PS. All production devices are [PS_Version] = 3 or later. The IDCODE value depends on the device type and the minimum production revision. The device ID codes and minimum production versions are listed in Table 1-2. The functionality implemented in each device type is listed in *Zynq UltraScale+ MPSoC Product Overview* (DS891) [Ref 1]. This includes number of APU cores, the availability of the VCU, number of CLBs, number of DSPs, and other blocks in the device.

Table 1-2: **Device ID Codes and Minimum Production Revisions**

Device Name	Product Family	IDCODE[31:0][(1)]
ZU1	CG, EG	0468_8093h
ZU2	CG, EG	1471_1093h
ZU3	CG, EG	1471_0093h
ZU4	CG, EG, EV	0472_1093h
ZU5	CG, EG, EV	0472_0093h
ZU6	CG, EG	2473_9093h
ZU7	CG, EG, EV	1473_0093h
ZU9	EG	2473_8093h
ZU11	EG	0474_0093h
ZU15	EG	1475_0093h
ZU17	EG	1475_9093h
ZU19	EG	1475_8093h
ZU21	DR	147E_1093h
ZU25	DR	147E_5093h
ZU27	DR	147E_4093h
ZU28	DR	147E_0093h
ZU29	DR	147E_2093h
ZU39	DR	147E_6093h
ZU42	DR	046D_4093h
ZU43	DR	147F_D093h
ZU46	DR	147F_8093h
ZU47	DR	147F_F093h
ZU48	DR	147F_B093h
ZU49	DR	147F_E093h
ZU65	DR	046D_1093h
ZU67	DR	046D_0093h

Notes:

1. Bits [27:0] refer to the device type. Bits [31:28] are the device revision. The minimum revision value for each production-qualified device is shown in the table.

IP Revisions

Table 1-3 lists the IP revisions.

Table 1-3: **IP Revisions**

System Element	Vendor	Version
RPU Core CPUs (Cortex-R5F)	Arm	r1p3
APU Core CPUs (Cortex-A53)	Arm	r0p4-50rel0
APU Core Crypto	Arm	r0p4-00rel0
APU Core Neon	Arm	r0p4-00rel0
CCI Coherent Interconnect (CCI-400)	Arm	r1p3-00rel0
AXI Interconnect (NIC-400)	Arm	r0p2-00rel0
APU GIC Interrupts (GIC-400)	Arm	r0p1-00rel0
RPU GIC Interrupts (PL390)	Arm	r0p0-00rel2
CoreSight Debug (SoC-400)	Arm	r3p1-00rel0
AXI Interconnect QoS (QoS-400)	Arm	r0p2-00rel0
SMMU Memory Management (SMMU-500)	Arm	r2p1-00rel1
CoreSight STM (STM-500)	Arm	r0p1-00rel0
GPU Graphics (Mali-400)	Arm	r1p1-00rel2
GEM Ethernet Controllers	Cadence	r2p03
GEM Ethernet GXL	Cadence	r1p06f1
GEM Ethernet RGMII	Cadence	r1p04
I2C Controllers	Cadence	r114_f0100_final
TTC Timer/Counters	Cadence	r2
UART Controllers	Cadence	r113
SPI Controllers	Cadence	r109
LPD SWDT, FPD SWDT, CSU SWDT Units	Cadence	r1p03
DDR Memory Controller	Synopsys	2.40a-lp06
DDR Memory PHY (GDSII)	Synopsys	1.40a_patch1
USB 3.0 Controllers	Synopsys	2.90a
SD/SDIO/eMMC Controllers	Arasan	ver1p48_140929
NAND Controller (ONFI, AXI, PIO, MDMA)	Arasan	v3p9_140822
LPD DMA, FPD DMA Units	Northwest Logic	1.13
SATA Controller (Dual, AHCI, 128AXI, RAM)	CEVA	FA13

System Software

The Zynq UltraScale+ MPSoC is a complex system-on-a-chip. With the two or four high-performance 64-bit APUs, two real-time processing units (RPUs), one graphics processing unit (GPU), and other hardware peripherals, making it suitable for heterogeneous processing. There is ample supporting software to enable hardware-software co-processing and a virtual environment to derive system-level benefits.

Xilinx provides a virtual development platform, firmware code, and device drivers for all of the I/O peripherals present in the PS and PL. These device drivers are provided in source format and support bare-metal or standalone systems and Linux platforms. An example first-stage boot loader (FSBL) is also provided in source-code format. The source drivers for stand-alone and FSBL are provided as part of the Xilinx Software Development Kit (SDK). The Linux drivers are provided through the Xilinx Open Source Git repository.

More information is available in the *Zynq UltraScale+ MPSoC Software Developer's Guide* (UG1137) [Ref 3]. In addition, the Xilinx Alliance Program partners provide system software solutions for IP, middleware, and operation systems.

System Features Assigned by Software

Table 1-4 lists general purpose features that are assigned by software for specific functions.

Table 1-4: **System Features Assigned by Software**

Feature	Function
PMU global persistent general storage registers {4:7}	Four registers are used by the FSBL and other Xilinx software products: PMU_GLOBAL.PERS_GLOB_GEN_STORAGE{4:7}.
PMU global general storage registers {4:6}	Three registers are used by the FSBL and other Xilinx software products: PMU_GLOBAL.GLOBAL_GEN_STORAGE{4:6}.
PMU general purpose MIO pins	Table 6-3 provides PMU general purpose MIO pins. Pins are used to control external power supplies.
GPIO signals to reset PL instantiated logic	Four GPIO pins are used by the Vivado process or the configuration wizard (PCW) provides resets through the EMIO to PL fabric.
PMU sleep mode request	The PMU has four IPI interrupts. PMU_0 interrupt is assigned by the PMU firmware to transition the PMU to sleep mode.

Documentation

The Zynq UltraScale+ MPSoC device is divided between the PS and PL. There are several units in the PL that have special wiring connections to the PS and the PL I/O pins. The units are powered by PL voltage pins. The PL fabric can be configured using the UltraScale+ LogiCORE™ soft IP.

Table 1-5: **Document Matrix**

Document		System Architect	PCB Design	System Software	Host Software	Description/ Audience Relative to the TRM
				PMU FW, FSBL, Drivers	Linux, Other	
Technical Reference Manual	UG1085	Yes	Pin functions	PS functionality	Architecture	Architecture, functionality, and control.
Data Sheet: Overview	DS891	Start here	Overview	Overview	~	Introductions of all the system elements in the PS and PL.
MPSoC Data Sheet: DC and AC	DS925	Frequencies	AC/DC spec.	~	~	PCB designer.
RFSoC Data Sheet: DC and AC	DS926	Frequencies	AC/DC spec.	~	~	PCB designer.
PCB Design User Guide	UG583	~	Yes	~	~	PCB designer.
System Monitor User Guide	UG580	~	Analog inputs	Yes	~	Explains the core functionality of the SYSMON units.
Online Register Reference	UG1087	~	~	Yes	~	Register sets (modules) descriptions.
Software Developer Guide	UG1137	Functionality	~	Yes	Yes	System software features.
PS LogiCORE IP Product Guide	PG201	PS-PL interface	~	~	~	Integration using Vivado design tools.
Packaging and Pinouts Spec.	UG1075	~	Yes	~	~	Defines DDR to DRAM I/O connections.
Product Data Sheet: Overview	DS890	Perspective	~	Perspective	~	All UltraScale and UltraScale+ devices.
PL-based MPSoC units	Several	Functionality per device	GTR	Yes	~	Examples: VCU, PCIe, 100 Gbit.
PL-based FPGA units	Many	PL Instantiations	SelectIO	~	~	Examples: DSP, LUT, block RAM.
OS and Libraries Document Collection	UG643	Yes	~	Yes	Yes	System and application programmer.

This technical reference manual (TRM) describes the architecture and functionality of the PS and parts of the PL. The TRM is a foundation for the *Zynq UltraScale+ MPSoC Software Developer's Guide* (UG1137) [Ref 3] and other application guides. The PS control registers are defined in the *Zynq UltraScale+ MPSoC Register Reference* (UG1087) [Ref 4]. See Appendix A, Additional Resources and Legal Notices for a list of helpful documents and online resources.

Signals, Interfaces, and Pins

Introduction

The dedicated device pins and the major signals and interfaces that cross between Programmable Logic (PL) and Processing System (PS) power domains are listed in this chapter. Figure 2-1 shows the dedicated device pins, and the signals and interfaces between power domains.

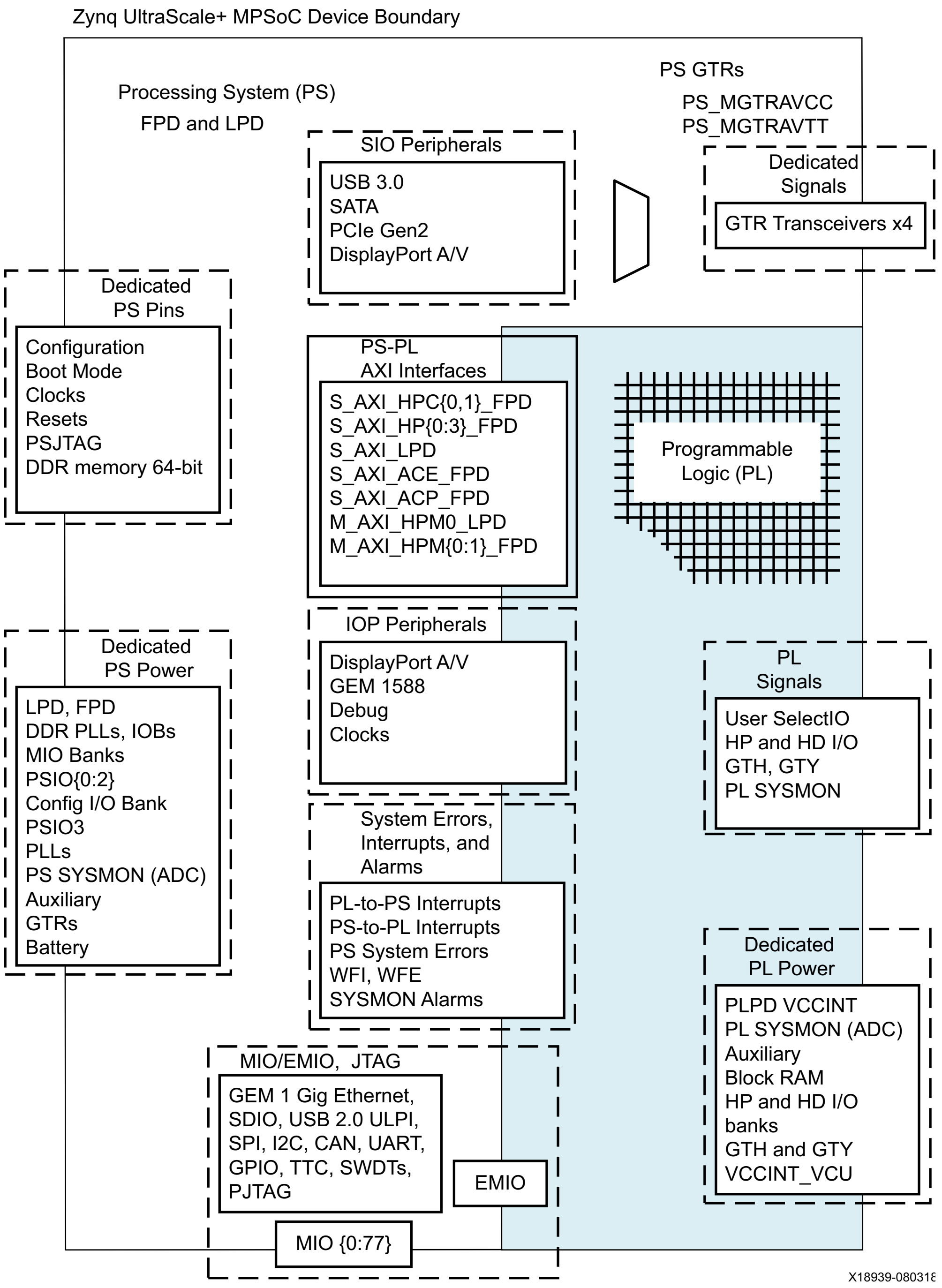


Figure 2-1: **PS Pins and Interfaces Diagram**

Dedicated Device Pins

The dedicated device pins are divided into these groups:

- Power.
- Clock, reset, and configuration.
- JTAG interfaces.
- Multiplexed I/O (MIO).
- PS GTR serial channels.
- DDR I/O (see Table 17-3 in DDR PHY Features in Chapter 17).

Power Pins

The dedicated power pins for the PS and internal logic of the PL are listed in Table 2-1. See *Zynq UltraScale+ MPSoC Data Sheet: DC and AC Switching Characteristics* (DS925) [Ref 2] for specifications.

Table 2-1: **Power Pins**

Pin Name	Description
VCC_PSINTLP	PS low-power domain (LPD) supply voltage.
VCC_PSINTFP	PS full-power domain (FPD) supply voltage.
VCC_PSAUX	PS auxiliary voltage.
VCC_PSBATT	PS battery operated voltage.
VCC_PSPLL	LPD PLLs: RPLL (RPU), IOPLL (I/O). FPD PLLs: APLL (APU), VPLL (video), DPLL (DDR controller).
VCC_PSDDR_PLL	DDR PLLs supply voltage for DDRIOB. Tie to ground.
VCC_PSINTFP_DDR	DDR memory controller supply voltage. Tie to ground.
VCCO_PSDDR	PS DDR I/O supply voltage. Tie to ground.
VCCO_PSIO[0:3]	Power supply voltage for the PS I/O banks. • VCCO_PSIO[0] is bank 500. MIO pins 0 to 25. • VCCO_PSIO[1] is bank 501. MIO pins 26 to 51. • VCCO_PSIO[2] is bank 502. MIO pins 52 to 77. • VCCO_PSIO[3] is bank 503. Mode, config, PSJTAG, error, SRST, POR, PS_REF_CLK.
VCCINT	PL power domain (PLPD) supply voltage.
VCCINT_VCU	Video codec unit supply voltage.
VCCAUX	PL auxiliary voltage.
VCCBRAM	PL block RAM supply voltage.
PS_MGTRAVCC	PS-GTR $V_{MGTAVCC}$ supply voltage.

Table 2-1: **Power Pins *(Cont'd)***

Pin Name	Description
PS_MGTRAVTT	PS GTR $V_{MGTAVTT}$ termination voltage.
VCC_PSADC	PS System Monitor analog voltage.
VCCADC	PL System Monitor analog voltage.

Clock, Reset, and Configuration Pins

The clock pins include the main PS reference clock input and the clock crystal connections to the real-time clock (RTC) in the battery power domain. The reset and configuration pins control the device and provide status information.

Table 2-2: **Clock, Reset, and Configuration Pins**

Pin Name	Direction	Type	Description
PS_REF_CLK	Input	Dedicated	System reference clock.
PS_PADI	Input	Dedicated	Crystal pad input (RTC).
PS_PADO	Output	Dedicated	Crystal pad output (RTC).
PS_POR_B	Input	Dedicated	Power-on reset signal.
POR_OVERRIDE	Input	Dedicated	POR delay override. 0 = Standard PL power-on delay time[(1)] (recommended default). 1= Faster PL power-on delay time.[(1)] Do not allow this pin to float before and during configuration. This pin must be tied to VCCINT or GND.
PS_SRST_B	Input	Dedicated	System reset commonly used during debug.
PS_MODE	Input/Output	Dedicated	4-bit boot mode pins sampled on POR deassertion.
PS_INIT_B	Input/Output	Dedicated	Indicates the PL is initialized after a power-on reset (POR). This signal should not be held Low externally to delay the PL configuration sequence because the signal level is not visible to software. However, if there is a CRC error detected when the PL bitstream is loaded PS_INIT_B will be driven low.
PS_DONE	Output	Dedicated	Indicates the PL configuration is completed. Requires an external pull-up resistor.
PS_PROG_B	Input	Dedicated	PL configuration reset signal.
PS_ERROR_OUT	Output	Dedicated	Asserted for accidental loss of power, a hardware error, or an exception in the PMU.
PS_ERROR_STATUS	Output	Dedicated	Indicates a secure lockdown state. Alternatively, it can be used by the PMU firmware to indicate system status.

Table 2-2: **Clock, Reset, and Configuration Pins *(Cont'd)***

Pin Name	Direction	Type	Description
PS_MGTREFCLK[3:0]	Input	Dedicated	Reference clock for the PS-GTR transceivers.
PUDC_B	Input	Dedicated	Pull-Up During Configuration (bar) Dedicated input pin. Active-Low input enables internal pull-up resistors on the SelectIO pins after power-up and during configuration. When PUDC_B is Low, internal pull-up resistors are enabled on each SelectIO pin. When PUDC_B is High, internal pull-up resistors are disabled on each SelectIO pin. Caution! Do not allow this pin to float before and during configuration. Must be tied High or Low. PUDC_B must be tied either directly or via a ≤ 1 k Ω resistor to VCCAUX or GND.

Notes:

1. The T_{POR} specification begins when the last of the monitored supplies (VCCINT, VCCAUX, VCCBRAM) reaches 95% of its recommended operating condition voltage.

JTAG Interfaces

There are two JTAG port interfaces: PSJTAG and PJTAG. The PSJTAG port can reach all TAP controllers on the chain. The signals are on the device pins listed in Table 2-3.

The PJTAG interface port provides exclusive access to the Arm DAP controller. The PJTAG interface signals on MIO are listed in Table 28-1.

PSJTAG is discussed in Chapter 39, System Test and Debug.

Table 2-3: **PS JTAG Interface Pins**

Pin Name	Direction	Description
PS_JTAG_TCK	Input	JTAG data clock.
PS_JTAG_TDI	Input	JTAG data input.
PS_JTAG_TDO	Output	JTAG data output.
PS_JTAG_TMS	Input	JTAG mode select.

MIO Pins

The PS uses the MIOs as described in Chapter 28, Multiplexed I/O. The MIO pins are configured by accessing registers located in the IOU_SLCR register set. The default routing for the peripheral I/O signals is through the EMIO interface to the PL fabric. The pin availability for the I/O controller is often different between routing to the MIO pins versus the EMIO interface to the PL.

Table 2-4: **MIO Pins**

Pin Name	Type	Direction	Description
PS_MIO[0:77]	Configurable pins, see Table 28-1	Input/Output	Multiplexed I/Os are configured for the IOP controllers and other interfaces: SPI, QSPI, NAND, USB 2.0 ULPI, GEM Ethernet RGMII, SDIO, UART, GPIO, MDIO, SWDT, TTC, TPIU, PJTAG.

DDR Memory Controller I/O

The DDR memory controller pins are described in Table 17-3 in Chapter 17, DDR Memory Controller.

PS GTR Serial Channel Device Pins

There are four pairs of gigabit serial device pins. These connect to the PCIe, SATA, and USB 3.0 signals from the controllers in the PS. The GTR serial channels are described in Chapter 29, PS-GTR Transceivers.

PS-PL Signals and Interfaces

The PS and PL can be tightly coupled in a heterogeneous processing system using the many signals and interfaces between the LPD and FPD in the PS and the functionality configured in the PL. The PL can also be independently isolated from the LPD and FPD regions using isolation walls. The PS-PL signals and interfaces also include other functions to configure and control the device. The PS-PL signals and interfaces include these groups:

- PS-PL Voltage Level Shifters
- Processor communications
- System error signals
- MIO-EMIO signals and interfaces
- Miscellaneous signals and interfaces
- Dedicated stream interfaces
- DisplayPort media interfaces
- Clock signals
- Timer signals
- System debug signals and interfaces

The PS-PL signal and interface names are listed in the *Zynq UltraScale+ MPSoC Processing System LogiCORE IP Product Guide* (PG201) [Ref 27].

PS-PL Voltage Level Shifters

The PS communicates with the PL using voltage level shifters. All of the signals (input and output) and interfaces between the PS and PL traverse a voltage boundary and are routed through voltage-level shifters. Some of the voltage-level shifter enables are controlled by the PL power state including the signals for the PL, the EMIO JTAGs, the PCAP interface, and other modules. The PL is treated as a separate power domain (PLPD). The AXI interfaces are isolated using isolation blocks. To enable an PS-PL AXI interface, the PS-PL isolation must be disabled by making a PMU service request using the PMU_GLOBAL[REQ_PWRUP_INT_EN] bit.

Processor Communications

Table 2-5 lists the processor communications signals. See Table 35-7 in Chapter 35, PS-PL AXI Interfaces for additional information.

Table 2-5: **Processor Communications**

Signal Name	Count	Source	Destination	Description
P2F PMU signal	32 signals	LPD	PL	GPO3 register signals to PL.(1)
F2P PMU signal	32 signals	PL	LPD	GPI3 register signals from PL.(1)
APU wake up	2 signals	PL	FPD	APU WFE and WFI event and interrupt status.
IRQ_P2F_PL_IPIx	4 channels	LPD	PL	IPI interrupts to PL targets.
IRQ_F2P_PL_IPIx	7 channels	PL	LPD	IPI interrupts to PS targets.
PL IRQs	16 signals	PL	LPD, FPD	IRQ signals from PL to GICs.
RPU CPU IRQs	4 signals	PL	LPD	FIQ, IRQ interrupts for each core.
APU CPU IRQs	8 signals	PL	FPD	FIQ, IRQ interrupts for each core.
PS System IRQs	>100 signals	PS	PL	PS generated interrupts to GICs and PL.
LPD IOP interrupts	100	LPD	PL	From peripherals to GICs and PL. See Table 13-1.
FPD IOP interrupts	64	FPD	PL	From peripherals to GICs and PL. See Table 13-1.
Events	~	LPD, FPD	PL	Events from RPU and APU.

Notes:
1. Software environments might assign meaning to the GPI and GPO signals of the PMU.

System Error Signals

Table 2-6 lists the system error signals.

Table 2-6: **System Error Signals**

Name	Count	Source	Destination	Description
System errors	49	PS	PL (and PS)	System error signals.
P2F PMU error	4	PMU	PL (and PS)	PMU output error signal.
F2P PMU error	4	PL	PMU	PMU input error signal.

MIO-EMIO Signals and Interfaces

The MIO device pins are fundamental to the I/O connections for the LPD IOP controllers. Software routes the controllers I/O signals to the MIO pins using IOU_SLCR registers. When there are not enough MIO pins for the peripheral I/O, then the EMIO can be used to connect signals to PL I/O device pins and logic within the PL. Table 2-7 lists the MIO-EMIO signals and interfaces.

RECOMMENDED: *The routing of the IOP interface I/O signals must be configured as a group. That is, the signals within an interface must not be split and routed to different MIO pin groups. For example, if the SPI 0 CLK is routed to MIO pin 40, then the other signals of the SPI 0 interface must be routed to MIO pins 41 to 45. Similarly, the signals within an IOP interface must not be split between MIO and EMIO. However, unused signals within an IOP interface do not necessarily need to be routed. Each unused MIO pin can be configured as a GPIO.*

Table 2-7: **MIO-EMIO Signals and Interfaces**

Interface	MIO Access	EMIO Access	Notes
GEM{0:3}	RGMII	GMII	MIO: 4-bit RGMII v2.0, external PHY, 250 MHz data rate. EMIO: 8-bit GMII, RGMII v2.0 (HSTL), RGMII v1.3, MII, SGMII, 1000BASE-SX, and 1000BASE-LX in PL, 125 MHz data rate.
SDIO{0, 1}	Yes	Yes	The SDIO interface performance is reduced when using the EMIO interface.
USB{0, 1}	USB 2.0 to external ULPI PHY.	No	The USB 3.0 interface is routed to a GTR channel
I2C{0, 1}	Yes	Yes	
SPI{0, 1}	Yes	Yes	The SPI interface performance is reduced when using the EMIO interface.
UART{0, 1}	Yes (RX, TX)	Yes (RX, TX, modem signals).	
CAN{0, 1}	Yes	Yes	External PHY.
GPIO Banks {0:2}	Yes (up to 78)	No	
GPIO Banks {3:5}	No	Yes (up to 96)	Input, output, and 3-state control.
Quad-SPI	Yes	No	
NAND	Yes	No	
LPD_SWDT, FPD_SWDT	Yes	Yes	Reset and output pulse.
CSU_SWDT	No	No	
TPIU Trace	Up to 16 bits	Up to 32 bits	

Miscellaneous Signals and Interfaces

Table 2-8 lists the miscellaneous signals and interfaces. For details, see Table 34-1, Table 34-2, and Table 34-3.

Table 2-8: **Miscellaneous Signals and Interfaces**

Name	Count	Source	Destination	Description
GEM FIFO	87 (x4)	GEM, PL	GEM, PL	Ethernet RX and TX FIFO packet streams.
GEM 1588	136	GEM, PL	GEM, PL	Ethernet 94-bit IEEE 1588 timestamp read by PL interface, PTP event frame interface, and timestamp clock interface.
DDR Refresh Req	2	PL	FPD	DDR memory controller external refresh request signals.
DDR Refresh Clk	1	PL	FPD	DDR memory controller refresh clock.
SEU error alarm	1	PL	CSU	Single event upset error alarm from the PL.
LPD DMA flow control	5	PL	LPD, PL clock, valids, acknowledges	See Figure 19-4.
FPD DMA flow control	5	PL	FPD, PL clock, valids, acknowledges	See Figure 19-4.

Dedicated Stream Interfaces

The GEM provides packet interface and support for the IEEE Std 1588 in the PL. The packet streaming interface (FIFO interface) from the GEM (bypassing the DMA) is available to the PL for implementation of additional functionality like packet inspection or audio-video broadcast (AVB). Additional signals for supporting the IEEE Std 1588 are also available to the PL. For details on this interface, refer to Chapter 34, GEM Ethernet.

The DisplayPort streaming interface for video and audio to/from the PL provides video and audio interfaces to the PL. It can take video/audio input from the PL and direct video/audio output to the PL. For details on this interface, refer to Chapter 33, DisplayPort Controller.

DisplayPort Media Interfaces

The DisplayPort streaming interface for video and audio to or from the PL provides video and audio interfaces to the PL. It can take video or audio input from the PL and direct video or audio output to the PL. For details on this interface, see Chapter 33, DisplayPort Controller. Table 2-9 lists the PS-PL DisplayPort media interfaces.

Table 2-9: **DisplayPort Media Interfaces**

Name	Count	Source	Destination	Description
Audio	77	PS, PL	PS, PL	One 32-bit audio input interface. One 32-bit audio output interface.
Video	154	PS, PL	PS, PL	Two 36-bit video streams to PS for overlay. One 36-bit video stream to PL display controller (e.g, HDMI, VGA, MIPI).

Clock Signals

Table 2-10 lists the clock signals.

Table 2-10: **Clock Signals**

Name	Count	Source	Destination	Description
PL_CLK{0:3}	4	LPD	PL	PS clock subsystem to PL fabric.
F2P clocks	2	PL	LPD	PS to PL auxiliary reference clocks.
RTC clock	1	LPD	LPD	RTC clock oscillator signal.

Timer Signals

Table 2-11 lists the timer signals.

Table 2-11: **Timer Signals**

Name	Count	Source	Destination	Description
TTC{0:3}_CLK	4	EMIO, MIO	LPD	Triple time counter optional clock sources.
TTC{0:3}_WAVE	4	LPD	EMIO, MIO	Triple timer counter waveform signal destinations.
WDT0_CLK	1	EMIO, MIO	LPD	LPD SWDT optional clock sources.
WDT1_CLK	1	EMIO, MIO	FPD	FPD SWDT optional clock sources.
WDT0_RST	1	LPD	GICs, EMIO, MIO	LPD SWDT reset signal destinations.
WDT1_RST	1	FPD	GICs, EMIO, MIO	FPD SWDT reset signal destinations.

System Debug Signals and Interfaces

Table 2-12 lists the system debug signals and interfaces.

Table 2-12: **System Debug Signals and Interfaces**

Name	Count	Description
CTI	48	CoreSight cross-trigger Interface.
TPIU	36	CoreSight trace-port interface.
FTM	118	Fabric trace module.
STM event		CoreSight system trace macrocell.

PS-PL AXI Interfaces

The PS-PL AXI interfaces are summarized in Table 2-13. These interfaces are described in Chapter 35, PS-PL AXI Interfaces.

Table 2-13: **PS-PL AXI Interfaces Summary**

Interface Name	Abbreviation	FIFO Interface	Master	Usage Description
S_AXI_HP{0:3}_FPD	HP{0:3}	AFI_{2:5}	PL	Non-coherent paths from PL to FPD main switch and DDR. No L2 cache allocation.
S_AXI_LPD	PL_LPD	AFI_6	PL	Non-coherent path from PL to IOP in LPD.
S_AXI_ACE_FPD	ACE	None	PL	Two-way coherent path between memory in PL and CCI.
S_AXI_ACP_FPD	ACP	None	PL	Legacy coherency. I/O coherent with L2 cache allocation.
S_AXI_HPC{0, 1}_FPD	HPC{0, 1}	AFI_{0:1}	PL	I/O coherent with CCI. No L2 cache allocation.
M_AXI_HPM{0, 1}_FPD	HPM{0, 1}	None	PS	FPD masters to PL slaves.
M_AXI_HPM0_LPD	LPD_PL	None	PS	LPD masters to PL slaves.

Application Processing Unit

Introduction

The application processing unit (APU) consists of four Cortex™-A53 MPCore processors, L2 cache, and related functionality. The Cortex-A53 MPCore processor is the most power-efficient Arm v8 processor capable of seamless support for 32-bit and 64-bit code. It makes use of a highly efficient 8-stage in-order pipeline balanced with advanced fetch and data access techniques for performance. It fits in a power and area footprint suitable for entry-level devices, and is at the same time capable of delivering high-aggregate performance in scalable enterprise systems using high core density.

Cortex-A53 MPCore Processor Features

The Cortex-A53 MPCore processor includes the following features.

- AArch32 and AArch64 execution states.
- All exception levels (EL0, EL1, EL2, and EL3) in each execution state.
- Arm v8-A architecture instruction set including advanced SIMD, VFPv4 floating-point extensions, and cryptography extensions.
- Separate 32 KB L1 caches for instruction and data.
- Two-stage (hypervisor and guest stages) memory management unit (MMU).
- CPU includes an in-order 8-stage pipeline with symmetric dual-issue of most instructions.
- 1 MB L2 cache in CCI coherency domain.
- Accelerator coherency port (ACP).
- 128-bit AXI coherency extension (ACE) master interface to CCI.
- Arm v8 debug architecture.
- Configurable endianess.
- Supports hardware virtualization that enables multiple software environments and their applications to simultaneously access the system capabilities.
- Hardware-accelerated cryptography—3-10x better software encryption performance.

- Large physical address reach enables the processor to access beyond 4 GB of physical memory.
- TrustZone technology ensures reliable implementation of security applications.

Arm v8 Architecture

The Arm v8-A is the next generation 64-bit Arm architecture. Arm v8 is backward compatible to Arm v7 (i.e., a 32-bit Arm v7 binary will run on an Arm v8 processor). Although the Arm v8 is backward compatible with the Arm v7 architecture, the Cortex-A53 MPCore is not necessarily backward compatible with Cortex-A9 architecture. This is because some of the Cortex-A9 sub-system functions (e.g., Cortex-A9 L2 control registers) were implementation specific and not part of the Arm v7 architecture.

Arm v8 supports two architecture states.

- 64-bit execution state (AArch64)
- 32-bit execution state (AArch32)

  AArch32 is compatible with Arm v7; however, it is enhanced to support some features included in AArch64 execution state (for example, load-acquire and store-release). Both execution states support advanced single-instruction multiple-data (SIMD) and floating-point extension for integer and floating-point. Also, both states support cryptography extension for the advanced encryption standard (AES) encryption/decryption, SHA1/256, and RSA/ECC.

Figure 3-1 shows the block diagram of the APU.

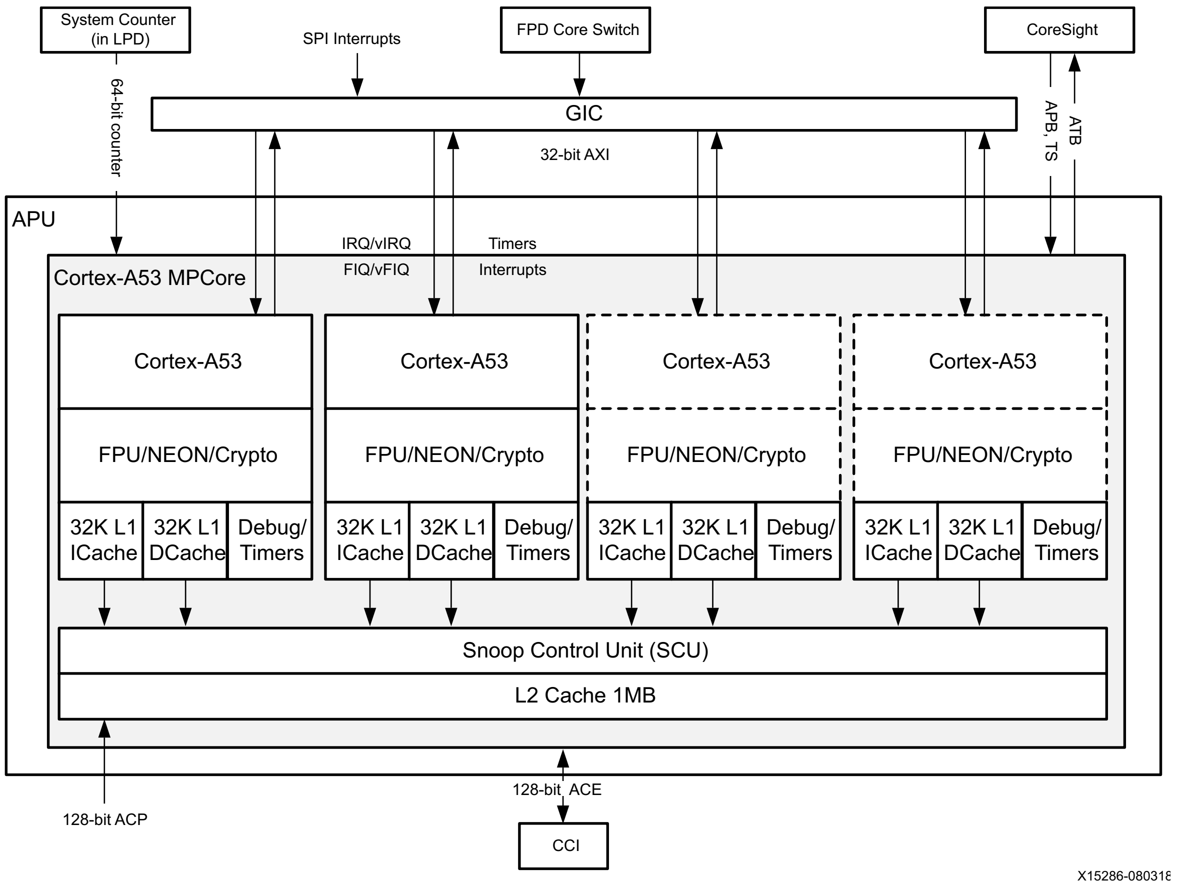


Figure 3-1: **APU Block Diagram**

The Arm v8 exception model defines exception levels EL0–EL3, where:

- EL0 has the lowest software execution privilege. Execution at EL0 is called unprivileged execution.

  Increased exception levels, from 1 to 3, indicate an increased software execution privilege.
- EL1 provides support for basic non-secure state.
- EL2 provides support for processor virtualization.
- EL3 provides support for a secure state.

The APU MPCore processor implements all the exception levels (EL0–EL3) and supports both execution states (AArch64 and AArch32) at each exception level.

When a Cortex-A53 MPCore processor is brought up in 32-bit mode using the APU_CONFIG0 [VINITHI] parameter register, its exception table cannot be relocated at run time. The V[13] bit of the system control register defines the base address of the exception vector.

See the *Zynq UltraScale+ MPSoC Software Developer's Guide* (UG1137) [Ref 3] for more information.

Figure 3-2 shows a top-level functional diagram of the Cortex-A53 MPCore processor.

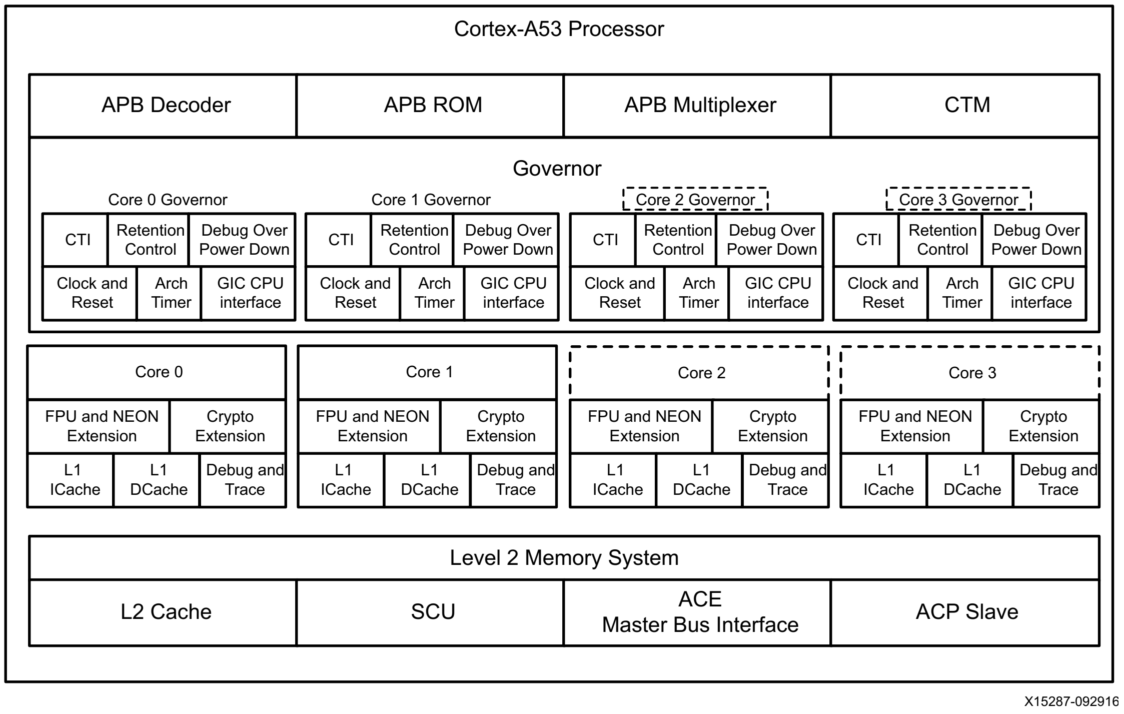


Figure 3-2: **APU Block Diagram**

Security State

An Arm v8 includes the EL3 exception level that provides the following security states, each with an associated memory address space.

- In the secure state, the processor can access both the secure memory address space and the non-secure memory address space. When executing at EL3, the processor can access all the system control resources.
- In the non-secure state, the processor can access only the non-secure memory address space and cannot access the secure system control resources.

Secure and non-secure AXI transactions are sent through the system using the TrustZone protocols.

For more information on the Arm v8 security states, see APU MPCore TrustZone Model in Chapter 16.

APU Functional Units

The following sections describe the main Cortex-A53 MPCore processor components and their functions.

- Instruction Fetch Unit
- Data Processing Unit
- Advanced SIMD and Floating-point Extension
- Cryptography Extension
- Translation Lookaside Buffer
- Data-side Memory System
- L2 Memory Subsystem
- Cache Protection
- Debug and Trace
- Generic Interrupt Controller
- Timers

Instruction Fetch Unit

The instruction fetch unit (IFU) contains the instruction cache controller and its associated linefill buffer. The Cortex-A53 MPCore instruction cache is 2-way set associative and uses virtually-indexed physically-tagged (VIPT) cache lines holding up to 16 A32 instructions, 16 32-bit T32 instructions, 16 A64 instructions, or up to 32 16-bit T32 instructions.

The IFU obtains instructions from the instruction cache or from external memory and predicts the outcome of branches in the instruction stream, and then passes the instructions to the data-processing unit (DPU) for processing.

Data Processing Unit

The data-processing unit (DPU) holds most of the program-visible processor states, such as general-purpose registers and system registers. It provides configuration and control of the memory system and its associated functionality. It decodes and executes instructions while operating on data held in the registers, in accordance with the Arm v8-A architecture. Instructions are fed to the DPU from the IFU. The DPU executes instructions that require data to be transferred to or from the memory system by interfacing to the data-cache unit (DCU), which manages all load and store operations.

Advanced SIMD and Floating-point Extension

Advanced SIMD and floating-point extension implements Arm NEON technology; a media and signal processing architecture that adds instructions targeted at audio, video, 3D graphics, image, and speech processing. Advanced SIMD instructions are available in AArch64 and AArch32 states.

Cryptography Extension

The cryptography extension supports the Arm v8 cryptography extensions. The cryptography extension adds new A64, A32, and T32 instructions to advanced SIMD that accelerate the following.

- Advanced encryption standard (AES) encryption and decryption.
- Secure-hash algorithm (SHA) functions SHA-1, SHA-224, and SHA-256.
- Finite-field arithmetic used in algorithms such as Galois/counter mode and elliptic curve cryptography.

Translation Lookaside Buffer

The translation lookaside buffer (TLB) contains the main TLB and handles all translation table walk operations for the processor. TLB entries are stored inside a 512-entry, 4-way set-associative RAM.

Data-side Memory System

The data-cache unit (DCU) consists of the following sub-blocks.

- The level 1 (L1) data-cache controller that generates the control signals for the associated embedded tag, data, and dirty RAMs, and arbitrates between the various sources requesting access to the memory resources. The data cache is 4-way set associative and uses a physically-indexed physically-tagged (PIPT) scheme for lookup that enables unambiguous address management in the system.
- The load/store pipeline that interfaces with the DPU and main TLB.
- The system controller that performs cache and TLB maintenance operations directly on the data cache and on the instruction cache through an interface with the IFU.
- An interface to receive coherency requests from the snoop-control unit (SCU).

Store Buffer

The store buffer (STB) holds store operations when they have left the load/store pipeline and are committed by the DPU. The STB can request access to the cache RAMs in the DCU, request the BIU to initiate linefills, or request the BIU to write the data out on the external write channel. External data writes are through the SCU. The STB can merge the following.

- Several store transactions into a single transaction if they are to the same 128-bit aligned address.
- Multiple writes into an AXI or CHI write burst. The STB is also used to queue maintenance operations before they are broadcast to other cores in the Cortex-A53 MPCore CPU cluster.

The Cortex-A53 MPCore L1 memory system consists of separate L1 instruction and data caches. It also consists of two levels of TLBs.

- Separate micro TLBs for both instruction and data sides.
- Unified main TLB that handles misses from micro TLBs.

Bus Interface Unit and SCU Interface

The bus interface unit (BIU) contains the SCU interface and buffers to decouple the interface from the cache and STB. The BIU interface and the SCU always operate at the processor frequency.

Snoop Control Unit

The integrated snoop-control unit (SCU) connects the APU MPCore and an accelerator coherency port (ACP) used in Zynq UltraScale+ MPSoCs. The SCU also has duplicate copies of the L1 data-cache tags for coherency support. The SCU is clocked synchronously and at the same frequency as the processors.

The SCU contains buffers that can handle direct cache-to-cache transfers between processors without having to read or write any data to the external memory system. Cache-line migration enables dirty-cache lines to be moved between processors, and there is no requirement to write back transferred cache-line data to the external memory system. The Cortex-A53 MPCore processor uses the MOESI protocol to maintain data coherency between multiple cores.

L2 Memory Subsystem

The Cortex-A53 MPCore processor's L2 memory system size is 1 MB. It contains the L2 cache pipeline and all logic required to maintain memory coherence between the cores of the cluster. It has the following features:

- An SCU that connects the cores to the external memory system through the master memory interface. The SCU maintains data-cache coherency between the APU MPCore and arbitrates L2 requests from the cores.
- The L2 cache is 16-way set-associative physically-addressed.
- The L2 cache tags are looked up in parallel with the SCU duplicate tags. If both the L2 tag and SCU duplicate tag hit, a read accesses the L2 cache in preference to snooping one of the other processors.

Cache Protection

The Cortex-A53 MPCore processor supports cache protection in the form of ECC on RAM instances in the processor using two separate protection options.

- SCU-L2 cache protection
- CPU cache protection

These options enable the Cortex-A53 MPCore processor to detect and correct a one-bit error in any RAM and detect two-bit errors in some RAMs.

Cortex-A53 MPCore RAMs are protected against single-event-upset (SEU) such that the processor system can detect and continue making progress without data corruption. Some RAMs have parity single-error detect (SED) capability, while others have ECC single-error correct, double-error detect (SECDED) capability.

Note: The L1 instruction cache is protected by parity bits. It does not implement error correction code.

The processor can make progress and remain functionally correct when there is a single-bit error in any RAM. If there are multiple single-bit errors in more than one RAM, or within different protection granules within the same RAM, then the processor also remains functionally correct. If there is a double-bit error in a single RAM within the same protection granule, then the behavior depends on the RAM.

- For RAMs with ECC capability, the error is detected and reported if the error is in a cache line containing dirty data.
- For RAMs with only parity, a double-bit error is not detected and therefore, could cause data corruption.

Interrupts upon an error event allow for the system to take the proper action, including flushing and re-loading caches, logging the error, etc. Multi-bit upsets (MBU) are avoided by proper interleaving, choice of ECC, and parity coding.

Debug and Trace

The Cortex-A53 MPCore processor supports a range of debug and trace features including the following.

- Arm v8 debug features in each core.
- ETMv4 instruction trace unit for each core.
- CoreSight™ cross-trigger interface (CTI).
- CoreSight cross-trigger matrix (CTM)
- Debug ROM.

Generic Interrupt Controller

The Cortex-A53 MPCore uses an external generic interrupt controller GIC-400 to support interrupts. It is a GICv2 implementation and provides support for hardware virtualization. For a detailed overview on GICv2 and system interrupts, refer to Chapter 13, Interrupts.

Timers

The Cortex-A53 MPCore processor implements the Arm generic timer architecture. For a detailed overview on APU timers, refer to Chapter 14, Timers and Counters.

APU Memory Management Unit

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 Arm 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 output LSBs without a lookup. Each stage walks through multiple levels of translation. Figure 3-3 and Figure 3-4 show an example block translation and a page translation, respectively.

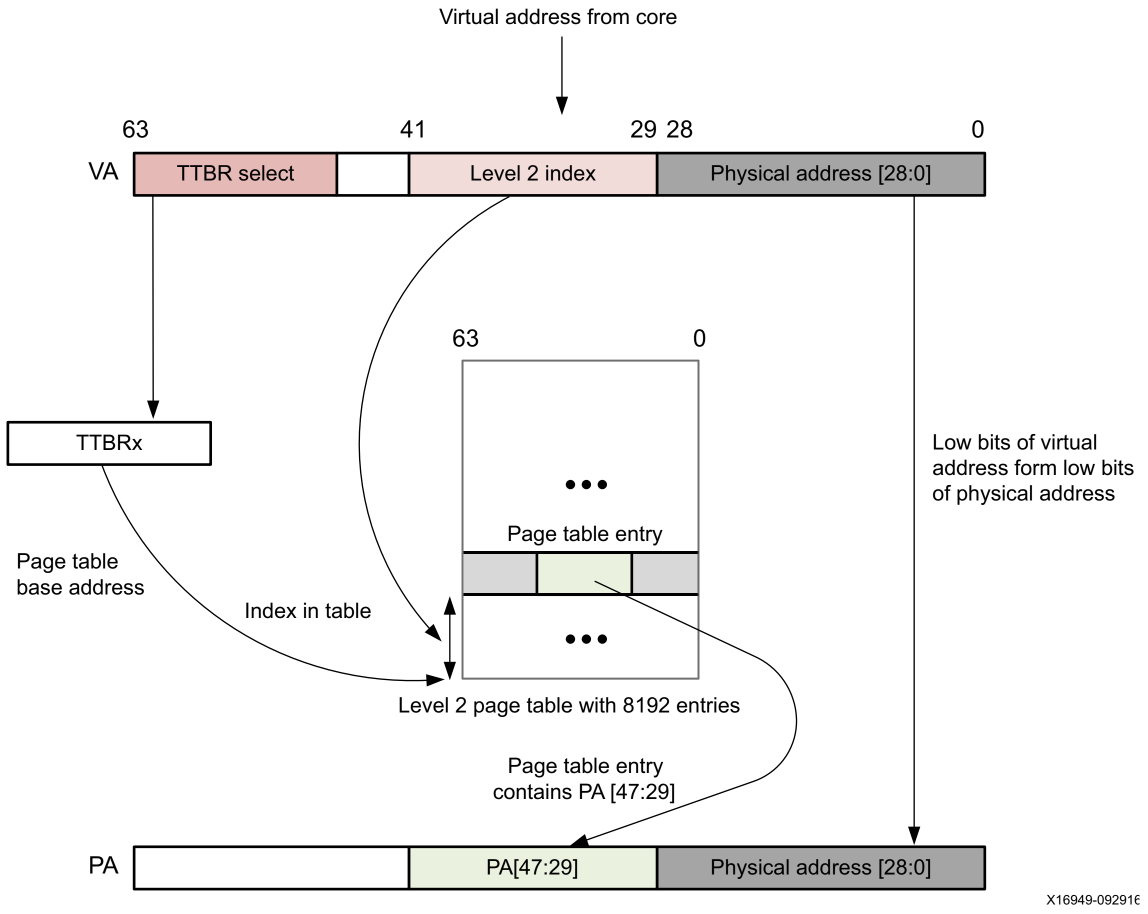


Figure 3-3: **Block Translation**

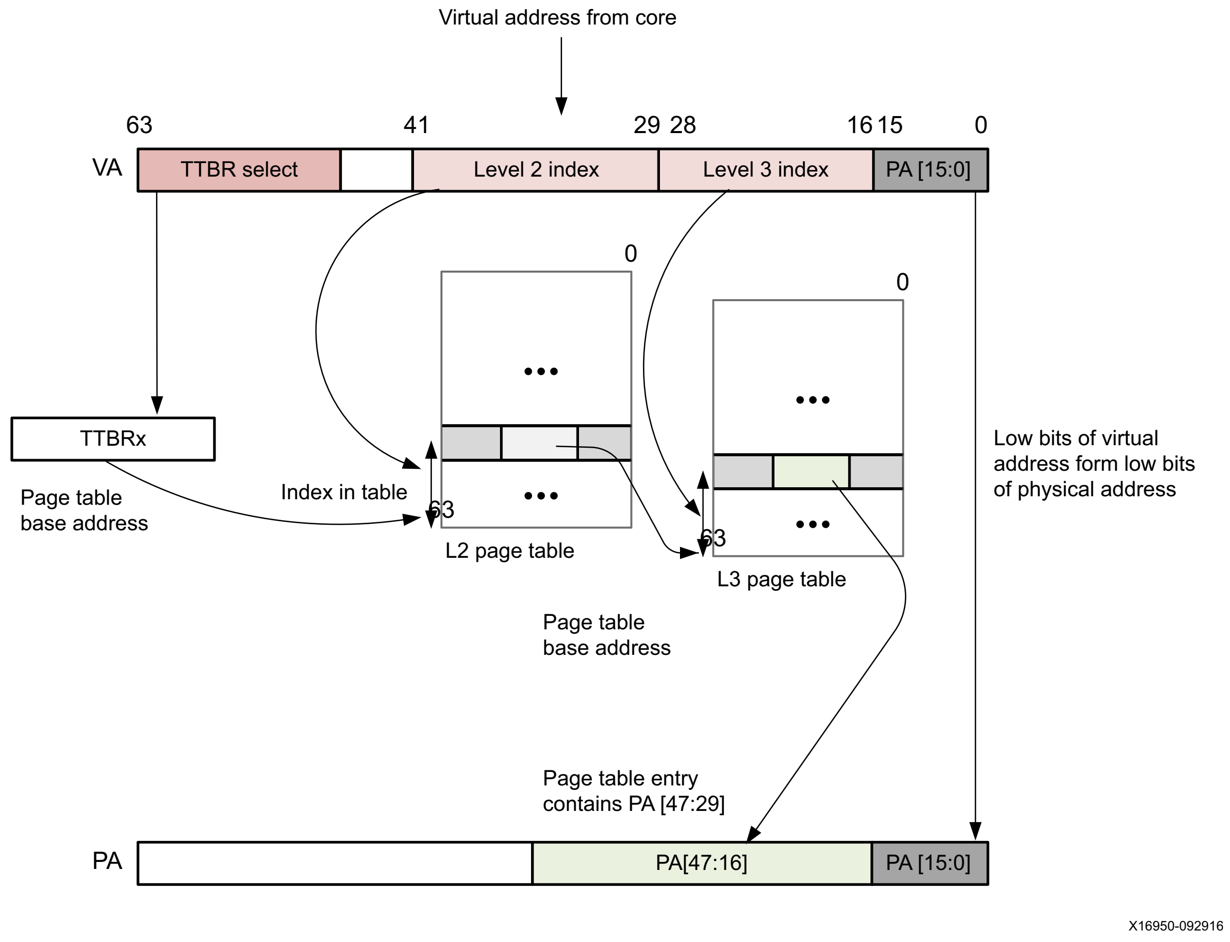


Figure 3-4: **Page Translation**

System 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 virtualization 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-A53 exception level-2 (EL2) provides processor virtualization. The Arm v8 supports virtualization extension to achieve full virtualization with near native guest operating system performance.

There are four key hardware components for virtualization.

- APU Virtualization
- Interrupt Virtualization
- Timer Virtualization
- System Memory Virtualization Using SMMU Address Translation

APU Virtualization

A processor element is in hypervisor mode when it is executing at EL2 in the AArch32 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. 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.

Figure 3-5 shows the Arm v8 execution modes discussed in this section.

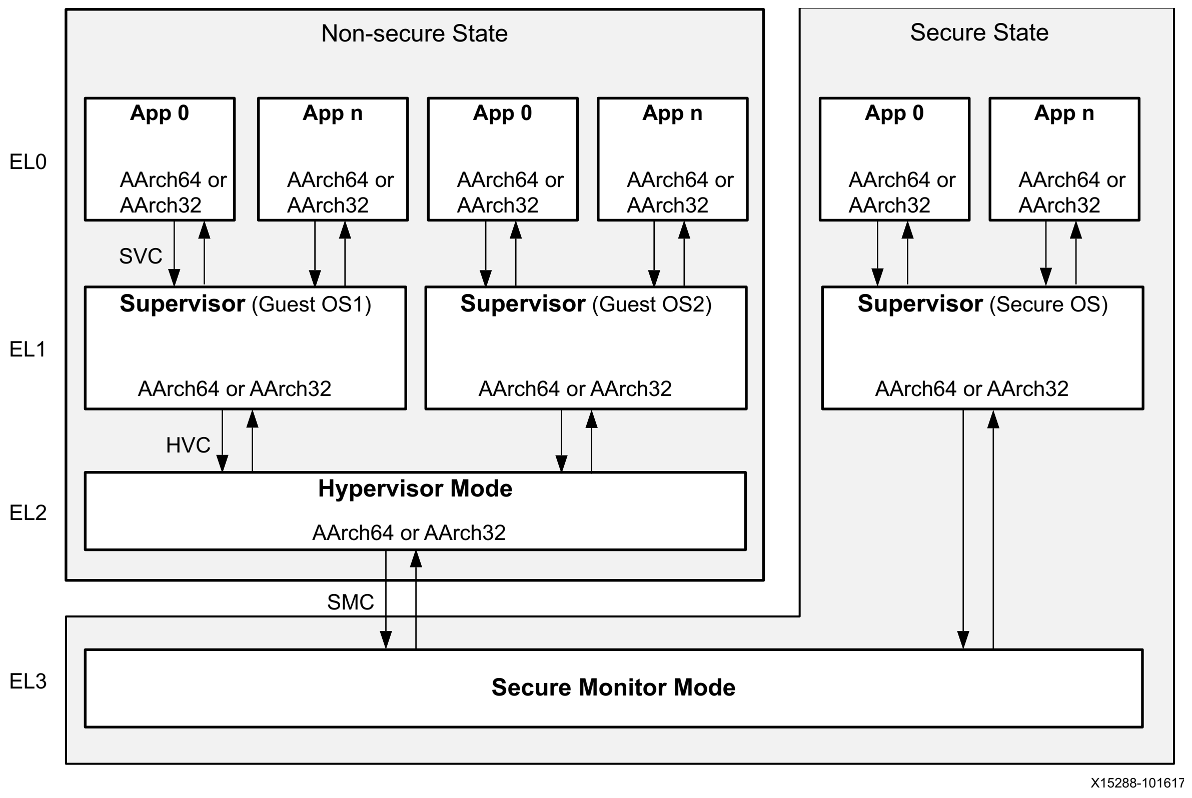


Figure 3-5: **Arm v8 Execution Modes**

Note: The following notes refer to Figure 3-5.

1. AArch64 is permitted only if EL1 is using AArch64.
2. AArch64 is permitted only if EL2 is using AArch64.

The hypervisor directly controls the allocation of the actual physical memory, thereby fulfilling its role of arbiter of the shared physical resources. This requires two stages (VA→IPA, and IPA→PA) of address translation. Figure 3-6 shows the traditional versus virtualized systems addresses in the translation stage.

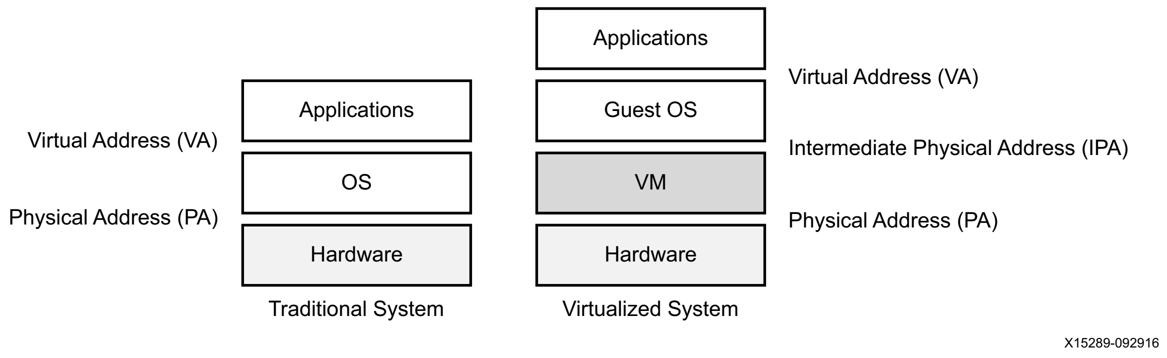


Figure 3-6: **Traditional versus Virtualized Systems Address Translation Stage**

Interrupt Virtualization

The APU GIC v2 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.

For detailed overview on the APU GIC, refer to Chapter 13, Interrupts.

Timer Virtualization

The Arm generic timers include support for timer virtualization. Generic timers provide a counter that measures (in real-time) the passing of time, and a timer for each CPU. The CPU timers are programmed to raise an interrupt to the CPU after a certain amount of time has passed, as per the counter.

Timers are likely to be used by both hypervisors and guest operating systems. However, to provide isolation and retain control, the timers used by the hypervisor cannot be directly configured and manipulated by guest operating systems. Refer to Chapter 14, Timers and Counters for further details.

System Coherency

The devices which require interaction with the CPU also share data with the CPU. However, when the CPU produces the (shared) data, the data is normally cached to improve CPU performance. Similarly, some devices have caches to improve their performance. There are two ways to share data between devices and CPUs.

- Software coherency
- Hardware coherency

In software coherency, software (as a producer) must flush CPU caches before devices can read shared data from memory. And, if the device produces the data, then software (as a consumer) must invalidate CPU caches before using the data produced by the device.

The hardware coherency (I/O coherency) can provide data coherence by having device memory requests snoop CPU caches. This speeds up data sharing significantly (by avoiding cache flush/invalidate), and simplifies software.

I/O Coherency

The Cortex-A53 MPCore processor has two options for I/O coherency.

- Accelerator coherency port (ACP) port
- Cache-coherent interconnect (CCI) ACE-Lite ports

The CCI ACE-Lite ports provide I/O coherency. The CCI ACE-Lite ports will snoop APU caches only if the request is marked coherent. All of the PS masters can be optionally configured as I/O coherent (including the RPU but excluding the FPD DMA unit). The RPU can be configured for direct DDR memory access by bypassing I/O coherency.

Full Two-way Coherency

Full coherent masters can snoop each other’s caches. Full coherency is provided through the CCI ACE-Lite ports. The Cortex-A53 MPCore supports a CCI ACE-Lite port, however, CCI ACE-Lite support must be implemented in the PL.

ACE Interface

The Zynq UltraScale+ MPSoCs interface to the cache-coherent interconnect (CCI) only supports the AXI coherency extension (ACE). ACE is an extension to the AXI protocol and provides the following enhancements. See Chapter 35, PS-PL AXI Interfaces.

- Support for hardware cache coherency.
- Barrier transactions that ensure transaction ordering.

System-level coherency enables the sharing of memory by system components without the software requirement to perform software 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.

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. Refer to the Arm® AMBA® AXI and ACE protocol specification for a detailed overview.

ACP Interface

The accelerator coherency port (ACP) is a 128-bit AXI slave interface on the snoop control unit (SCU) that provides an asynchronous cache-coherent access point directly from the PL to the APU. See Chapter 35, PS-PL AXI Interfaces.

APU Power Management

The Cortex-A53 MPCore processor provides mechanisms and support to control both dynamic and static power dissipation. The individual cores in the Cortex-A53 processor support four main levels of power management. This section describes the following.

- Power Islands
- Power Modes
- Event communication using a wait for event (WFE) or a send event (SEV) instruction. See Table 35-7.
- Communication with the platform management unit (PMU). See Chapter 6, Platform Management Unit.

Power Islands

Table 3-1 shows the power islands supported by the Cortex-A53 processor.

Table 3-1: **APU MPCore Power Islands**

Power Island	Description
CORTEXA53	Includes the SCU, the L2 cache controller, and the debug registers that are described as being in the debug domain. This domain is a part of the PS full-power domain (FPD).
PDL2	Includes the L2 cache RAM, L2 tag RAM, L2 victim RAM, and the SCU duplicate tag RAM.
PDCPU[4]	This represents core 0, core 1, core 2, and core 3. It includes the advanced SIMD and floating-point extensions, the L1 TLB, L1 cache RAMs, and debug registers.

Power Modes

The power islands can be controlled independently to give several combinations of powered-up and powered-down islands. The supported power modes in the APU MPCore are listed.

- Normal State
- Standby State
- Individual MPCore Shutdown Mode
- Cluster Shutdown Mode with System Driven L2 Flush
- Cluster shutdown the MPCore without system driven L2 flush.

Normal State

The normal mode of operation is where all of the processor functionality is available. The Cortex-A53 processor uses gated clocks and gates to disable inputs to unused functional blocks. Only the logic in use to perform an operation consumes any dynamic power.

Standby State

The following sections describe the methods to enter a standby state.

MPCore Wait for Interrupt

The *Wait for Interrupt* (WFI) feature of the Arm v8-A architecture puts the processor in a low-power state by disabling most of the clocks in the MPCore while keeping the MPCore powered up. Apart from the small dynamic power overhead on the logic used to enable the MPCore to wake up from a WFI low-power state, the power draw is reduced to only include the static leakage current variable. Software indicates that the MPCore can enter the WFI low-power state by executing the WFI instruction.

When the MPCore is executing the WFI instruction, the MPCore waits for all instructions in the MPCore to retire before entering the idle or low-power state. The WFI instruction ensures that all explicit memory accesses that occurred before the WFI instruction in the order of the program are retired. For example, the WFI instruction ensures that the following instructions receive the required data or responses from the L2 memory system.

- Load instructions
- Cache and TLB maintenance operations
- Store exclusive instructions

In addition, the WFI instruction ensures that stored instructions update the cache or are issued to the SCU.

MPCore Wait for Event

The *Wait for Event* (WFE) feature of the Arm v8-A architecture is a locking mechanism that puts the MPCore in a low-power state by disabling most of the clocks in the MPCore while keeping the MPCore powered up. Apart from the small dynamic power overhead on the logic used to enable the MPCore to wake up from the WFE low-power state, the power draw is reduced to only include the static leakage current variable.

A MPCore enters into a WFE low-power state by executing the WFE instruction. When executing the WFE instruction, the MPCore waits for all instructions in the MPCore to complete before entering the idle or low-power state.

If the event register is set, a WFE does not put the MPCore into a standby state, but the WFE clears the event register.

While the MPCore is in the WFE low-power state, the clocks in the MPCore are temporarily enabled (without causing the MPCore to exit the WFE low-power state), when any of the following events are detected.

- An L2 snoop request that must be serviced by the MPCore L1 data cache.
- A cache or TLB maintenance operation that must be serviced by the MPCore L1 instruction cache, data cache, or TLB.
- An APB access to the debug or trace registers residing in the MPCore power domain.

L2 Wait for Interrupt

When all the cores are in a WFI low-power state, the shared L2 memory system logic that is common to all the cores also enter a WFI low-power state.

Individual MPCore Shutdown Mode

In the individual MPCore shutdown mode, the PDCPU power island for an individual MPCore is shut down and all states are lost.

Use these steps to power down the MPCore.

1. Disable the data cache, by clearing the SCTLR.C bit, or the HSCTLR.C bit if in Hyp mode. This prevents more data cache allocations and causes cacheable memory attributes to change to normal, non-cacheable. Subsequent loads and stores do not access the L1 or L2 caches.
2. Clean and invalidate all data from the L1 data cache. The L2 duplicate snoop tag RAM for this MPCore is empty. This prevents any new data cache snoops or data cache maintenance operations from other MPCore in the cluster being issued to this core.
3. Disable any data coherency with other MPCores in the cluster by clearing the CPUECTLR.SMPEN bit. Clearing the SMPEN bit enables the MPCore to be taken out of coherency by preventing the MPCore from receiving cache or TLB maintenance operations broadcast by other MPCores in the cluster.
4. Execute an ISB instruction to ensure that all of the register changes from the previous steps are completed.
5. Execute a DSB SY instruction to ensure completion of all cache, TLB, and branch predictor maintenance operations issued by any MPCore in the cluster device before the SMPEN bit is cleared.
6. Execute a WFI instruction and wait until the STANDBYWFI output is asserted to indicate that the MPCore is in an idle and a low-power state.
7. Deassert DBGPWRDUP Low. This prevents any external debug access to the MPCore.
8. Activate the MPCore output clamps.
9. Remove power from the PDCPU power domain.

To power up the MPCore, apply the following sequence.

1. Assert nCPUPORESET Low. Ensure DBGPWRDUP is held Low to prevent any external debug access to the MPCore.
2. Apply power to the PDCPU power domain. Keep the state of the signals nCPUPORESET and DBGPWRDUP Low.
3. Release the MPCore output clamps.
4. Deassert the resets.
5. Set the SMPEN bit to 1 to enable snooping into the MPCore.
6. Assert DBGPWRDUP High to allow external debug access to the MPCore.

7. If required, use software to restore the state of the MPCore to its the state prior to power-down.

Cluster Shutdown Mode with System Driven L2 Flush

The cluster shutdown mode is where the PDCORTEXA53, PDL2, and PDCPU power islands are shut down and all previous states are lost. To power down the cluster, apply the following sequence.

1. Ensure that all cores are in shutdown mode, see Individual MPCore Shutdown Mode.
2. The MPCore asserts the pl_acpinact signal to idle the ACP. This is necessary to prevent ACP transactions from allocating new entries in the L2 cache during the hardware cache flush. For more information about the pl_acpinact signal, see Answer Record 70383.
3. Assert L2FLUSHREQ High.
4. Hold L2FLUSHREQ High until L2FLUSHDONE is asserted.
5. Deassert L2FLUSHREQ.
6. Assert ACINACTM. Wait until the STANDBYWFIL2 output is asserted to indicate that the L2 cache memory is idle.
7. Activate the cluster output clamps.
8. Remove power from the PDCORTEXA53 and PDL2 power domains.

The Zynq UltraScale+ MPSoC provides the ability to power off each of the four APU processors independent of the other processors. Each processor power domain includes an associated Neon core. The control for the power gating is dynamic and is handled by the power management software running on the platform management unit (PMU).

Clocks and Resets

Each of the APU cores can be independently reset. The APU MPCore reset can be triggered by the FPD system watchdog timer (FPD_SWDT) or a software register write. However, the APU is reset without gracefully terminating requests to/from the APU. The FPD system reset (FPD_SRST) is used in cases of catastrophic failure in the FPD system. The APU reset is primarily for software debug.

Programming steps for a software-generated reset:

1. Enable the reset request interrupt in the PMU. Write a 1 to one or more bits [APUx] in the PMU_GLOBAL.REQ_SWRST_INT_EN register.
2. Trigger the interrupt request. Write a 1 to one or more bits [APU{0:3}] in the REQ_SWRST_TRIG register.

The clock subsystem provides two clocks to the APU MPCore; one at the full clock rate and one at half the clock rate. The reference clock generator is described in Chapter 37, PS Clock Subsystem.

Performance Monitors

The Cortex-A53 MPCore processor includes performance monitors that implement the Arm PMUv3 architecture. The performance monitors enable gathering of various statistics on the operation of the processor and its memory system during runtime. They provide useful information about the behavior of the processor for use when debugging or profiling code. The performance monitor provides six counters. Each counter can count any of the events available in the processor.

System Registers

Table 3-2 describes the APU registers.

Table 3-2: **APU System Control Registers**

Register name	Overview
ERR_CTRL	Control register
ISR	Interrupt status register
IMR	Interrupt mask register
IEN	Interrupt enable register
IDS	Interrupt disable register
CONFIG_0	CPU core configuration
CONFIG_1	L2 configuration
RVBARADDR{0:3}{L,H}	Reset vector base address
ACE_CTRL	ACE control register
SNOOP_CTRL	Snoop control register
PWRCTL	Power control register
PWRSTAT	Power status register

IMPORTANT: *Do not perform a load/store exclusive to the device memory unless a workaround for the Arm® processor Cortex-A53 MPCore (MP030) product errata notice 829070 for APU registers is implemented. Speculative data reads might be performed to device memory.*

System Memory Virtualization Using SMMU Address Translation

The SMMU translates the virtual addresses within each operating environment into physical addresses of the system and is described in this section. The transaction protection mechanism of the SMMU is described in Chapter 16, System Protection Units.

Since the MPSoC system can support multiple operating systems with each guest OS supporting multiple application environments, the SMMU provides two stages of address translation. The first stage separates memory space for the operating systems and is managed by the hypervisor. The second stage separates application memory space within an OS and is managed by the host operating system. The programming of the address translation is coordinated with the MMUs in the MPCores and any MMUs in the PL to build the multitasking, heterogeneous system that shares one physically addressed memory subsystem.

- First-stage hardware address translation for virtualized, multiple-guest operating systems. Virtual address (VA) to intermediate physical address (IPA).
    - Hypervisor software programs the first-stage address translation unit to virtualize the addresses of bus masters other than the processors, e.g., DMA units and PL masters.
    - Associates each bus master with its intermediate virtual memory space of its OS.
- Second-stage hardware address translation for multi-application operating systems. Intermediate physical address (IPA) to physical address (PA).
    - Guest OS software programs the second-stage translation unit to manage the addressing of the memory mapped resources for each application program.
    - Associates the intermediate virtual memory address to the system’s physical address space.

The SMMU has the translation buffer and control units.

Note: The SMMU TCU uses the same master ID as that the one used for R5_0 so the security protection units cannot differentiate between the two masters. To mitigate this issue coherency can be disabled for the RPU so R5_0 is forced to access DDR through S0 port and SMMU TCU to S1 and S2 ports.

Translation Buffer Unit

The translation buffer unit (TBU) contains a translation look-aside buffer (TLB) that caches page tables maintained by the translation control unit (TCU). The SMMU implements a TBU for system masters as shown in Figure 15-1 in Chapter 15, PS Interconnect.

Translation Control Unit

The TCU controls and manages the address translation tables for the TBUs.

TBU Entry Updates

The TCU uses a private AXI stream interface to update the translation tables in the TBUs.

Figure 3-7 shows how the two address translation stages in the SMMU can be used in the system with the APU MPCore, GPU, and other masters. The location of the six TBUs is shown in Figure 15-1 in Chapter 15, PS Interconnect.

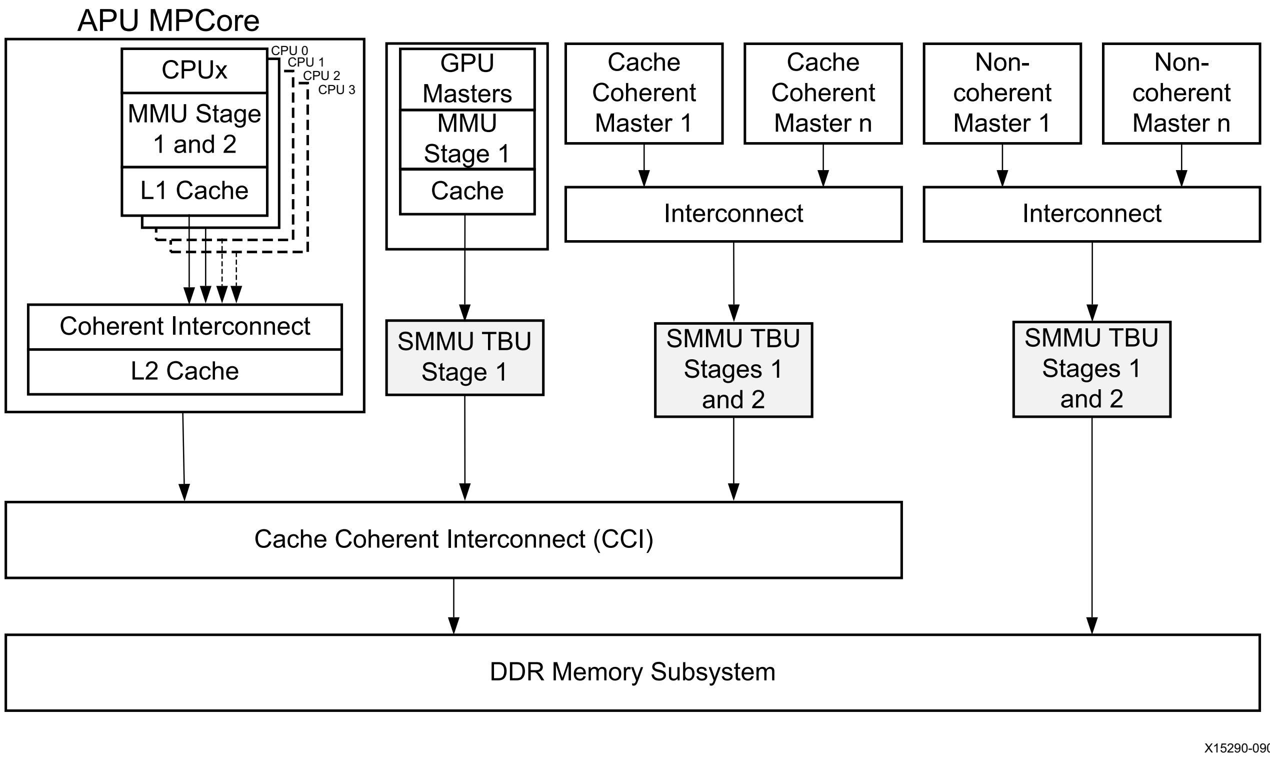


Figure 3-7: **Example of SMMU Locations in the System**

SMMU Architecture

The SMMU performs address translation of an incoming AXI address and AXI ID (mapped to *context*) to an outgoing address (PA). The Arm SMMU architecture also supports the concept of translation regimes, in which a required memory access might require two stages of address translation. The SMMU supports the following.

- Aarch32 short (32-bit) descriptor. Supports up to a 32-bit VA and 32-bit PA.
- Aarch32 long (64-bit) descriptor. Supports up to a 32-bit VA and 40-bit PA.
- Aarch64 (64-bit) descriptor. Supports up to a 49-bit VA and 48-bit PA.

Stage 1 SMMU Translation

Stage 1 translation is intended to assist the operating system, both when running natively or inside a hypervisor. Stage 1 translation works similarly to a traditional (single stage) CPU MMU. Normally, an operating system causes fragmentation of physical memory by continuously allocating and freeing memory space on the heap, both for kernel and applications. A virtualized system that includes a fragmented model between IPA and PA spaces (where multiple guest operating systems are sharing the same physical memory) is not advised because of this issue.

A typical solution, to allocate large contiguous physical memory, is to pre-allocate such buffers. This is very inefficient because the buffer is only required at runtime. Also, in a virtualized system, a pre-allocated solution requires the hypervisor to allocate any contiguous buffers to the guest operating system, which could require hypervisor modifications.

For a DMA device to operate on fragmented physical memory, a DMA scatter-gather mechanism is typically used, which increases software complexity and adds performance overhead. Also, some devices are not capable of accessing the full memory range, such as 32-bit devices in a 64-bit system. One solution is to provide a bounce buffer—an intermediate area of memory at a low address that acts as a bridge. The operating system allocates pages in an address space visible to the device and uses them as buffer pages for DMA to and from the operating system. Once the I/O completes, the content of the buffer pages is copied by the operating system into its final destination. There is significant overhead to this operation, which can be avoided with the use of SMMU. I/O virtualization can be achieved by using stage 1 (for native operating systems) and by stage 1 or 2 (for guest operating systems).

Stage 2 SMMU Translation

The SMMU stage 2 translations remove the need for the hypervisor to manage shadow translation tables, which simplifies hypervisor and improves performance. With stage 2 address translation (Figure 3-8), the SMMU enables a guest operating system to directly configure the DMA capable devices in the system.

The SMMU can also be configured to ensure that devices operating on behalf of one guest operating system are prevented from corrupting memory of another guest operating system.

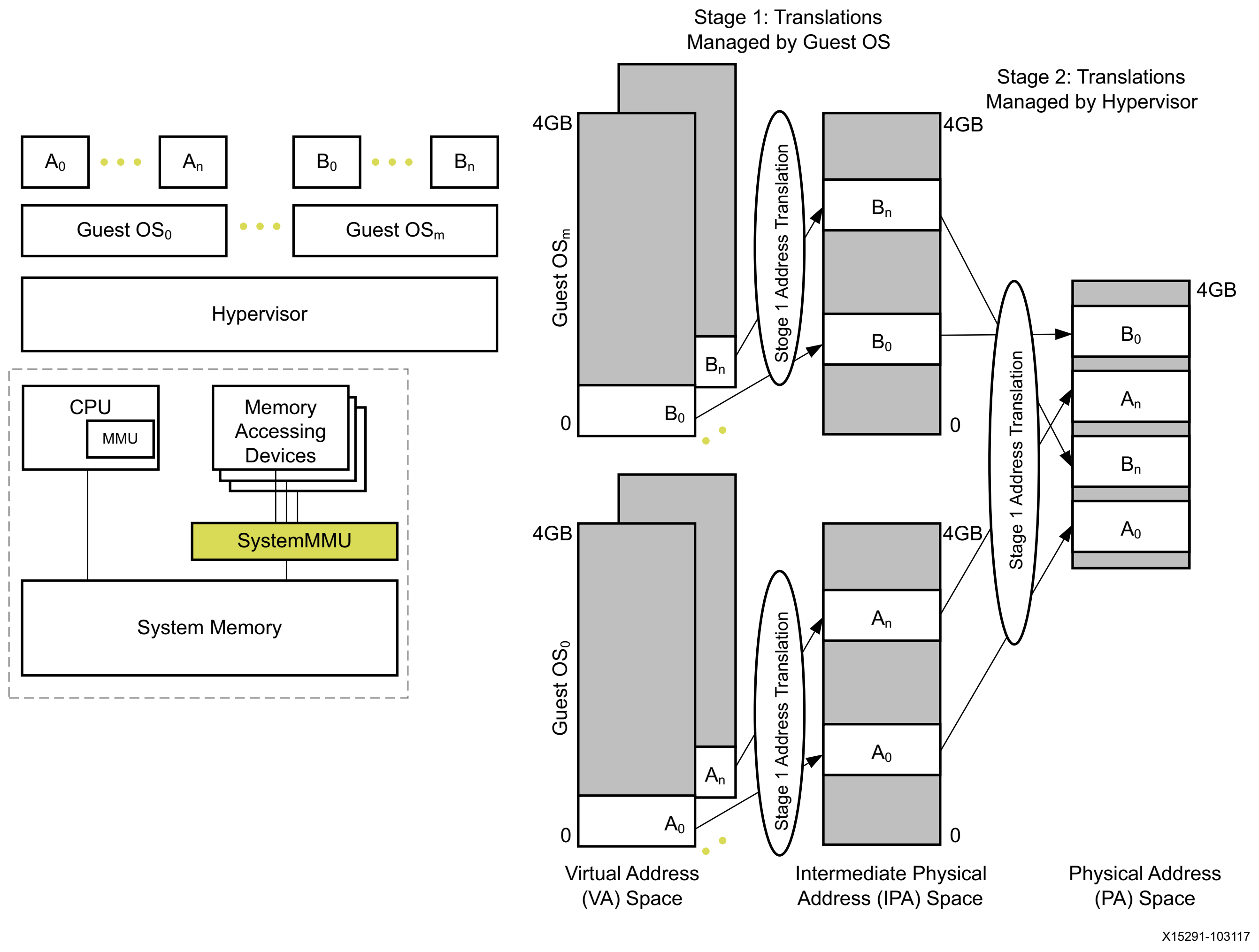

Stage 1: Translations Managed by Guest OS
Stage 2: Translations Managed by Hypervisor
A0
An
B0
Bn
Guest OS0
Guest OSm
Hypervisor
CPU
MMU
Memory Accessing Devices
SystemMMU
System Memory
4GB
0
Guest OSm
Guest OS0
Stage 1 Address Translation
Virtual Address (VA) Space
Intermediate Physical Address (IPA) Space
Physical Address (PA) Space
X15291-103117

Figure 3-8: **SMMU Stage 2 Address Translation**

Providing hardware separation between the two stages of address translation allows a clear definition of the ownership of the two different stages between the guest operating system (stage 1) and the hypervisor (stage 2). Translation faults are routed to the appropriate level of software. Management functions (TLB management, MMU enabling, register configurations) are handled at the appropriate stage of the translation process, improving performance by reducing the number of entries in the VM.

Stage 1 translations are supported for both secure and non-secure translation contexts. Stage 2 translations are only supported for non-secure translation contexts. For non-secure operations, the typical usage model for two-stage address translation is as follows.

- The non-secure operating system defines the stage 1 address translations for application and operating system level operations. The operating system does this as though it is defining mapping from VA to PA, but it is actually defining the mapping from VAs to IPA.

- The hypervisor defines the stage 2 address translation that maps the IPA to PA. It does this as part of its virtualization of one or more non-secure guest operating systems.

TLB Maintenance Operations

SMMU TLB maintenance operations (for example, TLB invalidates) can be initiated in one of the two ways.

- Accessing SMMU memory-mapped registers.
- Broadcasting TLB maintenance operations to the SMMU through the distributed virtual memory (DVM) bus. Clearing TLB entries through broadcast messages can significantly improve system performance by freeing-up TLB entries. TLB maintenance-message broadcasting is an important feature of the SMMU architecture.

SMMU Clocks and Resets

The SMMU AXI interfaces are clocked by the TOPSW_MAIN_CLK clock in the AXI interconnect for the FPD. The clock generator is described in Chapter 37, PS Clock Subsystem. The SMMU reset is in the FPD reset domain.

Chapter 4

Real-time Processing Unit

Introduction

The Zynq® UltraScale+™ MPSoC includes a pair of Cortex®-R5F processors for real-time processing based on the Cortex-R5F MP processor core from Arm®. The Cortex-R5F processor implements the Arm v7-R architecture and includes a floating-point unit that implements the Arm VFPv3 instruction set.

In the Cortex-R5F processor, interrupt latency is kept low by interrupting and restarting load-store multiple instructions. This is achieved by having a dedicated peripheral port that provides low latency access to the interrupt controller and by having tightly coupled memory ports for low latency and deterministic accesses to local RAM.

The Cortex-R5F processor is used for many safety-critical applications.

Real-time Processing Unit Features

- Integer unit implementing the Arm v7-R instruction set.
- Single and double precision FPU with VFPv3 instructions.
- Arm v7-R architecture memory protection unit (MPU).
- 64-bit master AXI3 interface for accessing memory and shared peripherals.
- 64-bit slave AXI3 interface for DMA access to the TCMs.
- Dynamic branch prediction with a global history buffer and a 4-entry return stack.
- Separate 128KB TCM memory banks with ECC protection for each TCM.
- 32KB instruction and data L1 caches with ECC protection.
- Independent Cortex-R5F processors or dual-redundant configuration.
- 32-bit master advanced eXtensible interface (AXI) peripheral interface on each processor for direct low-latency device memory type access to the interrupt controller.
- Debug APB interface to a CoreSight™ debug access port (DAP).
- Low interrupt latency and non-maskable fast interrupts.
- Performance monitoring unit.

- Exception handling and memory protection.
- ECC detection/correction on level-1 memories.
- Lock-step (redundant CPU) configuration is available to mitigate random faults in CPU registers and gates.
- Built-in self-test (BIST) to detect random faults in hardware (probably) caused by permanent failure.
- Watchdog to detect both systematic and random failures causing program flow errors.

Cortex-R5F Processor Functional Description

The Cortex-R5F processor is a mid-range CPU for use in deeply-embedded, real-time systems. It implements the Arm v7-R architecture, and includes Thumb-2 technology for optimum code density and processing throughput. The pipeline has a single arithmetic logic unit (ALU), but implements limited dual-issuing of instructions for efficient utilization of other resources such as the register file. Interrupt latency is kept low by interrupting and restarting load-store multiple instructions, and by use of a dedicated peripheral port that enables low-latency access to an interrupt controller. The processor has tightly-coupled memory (TCM) ports for low-latency and deterministic accesses to local RAM, in addition to caches for higher performance to general memory. Error checking and correction (ECC) is used on the Cortex-R5F processor ports and in Level 1 (L1) memories to provide improved reliability and address safety-critical applications. Figure 4-1 shows the system view of the real-time processing unit.

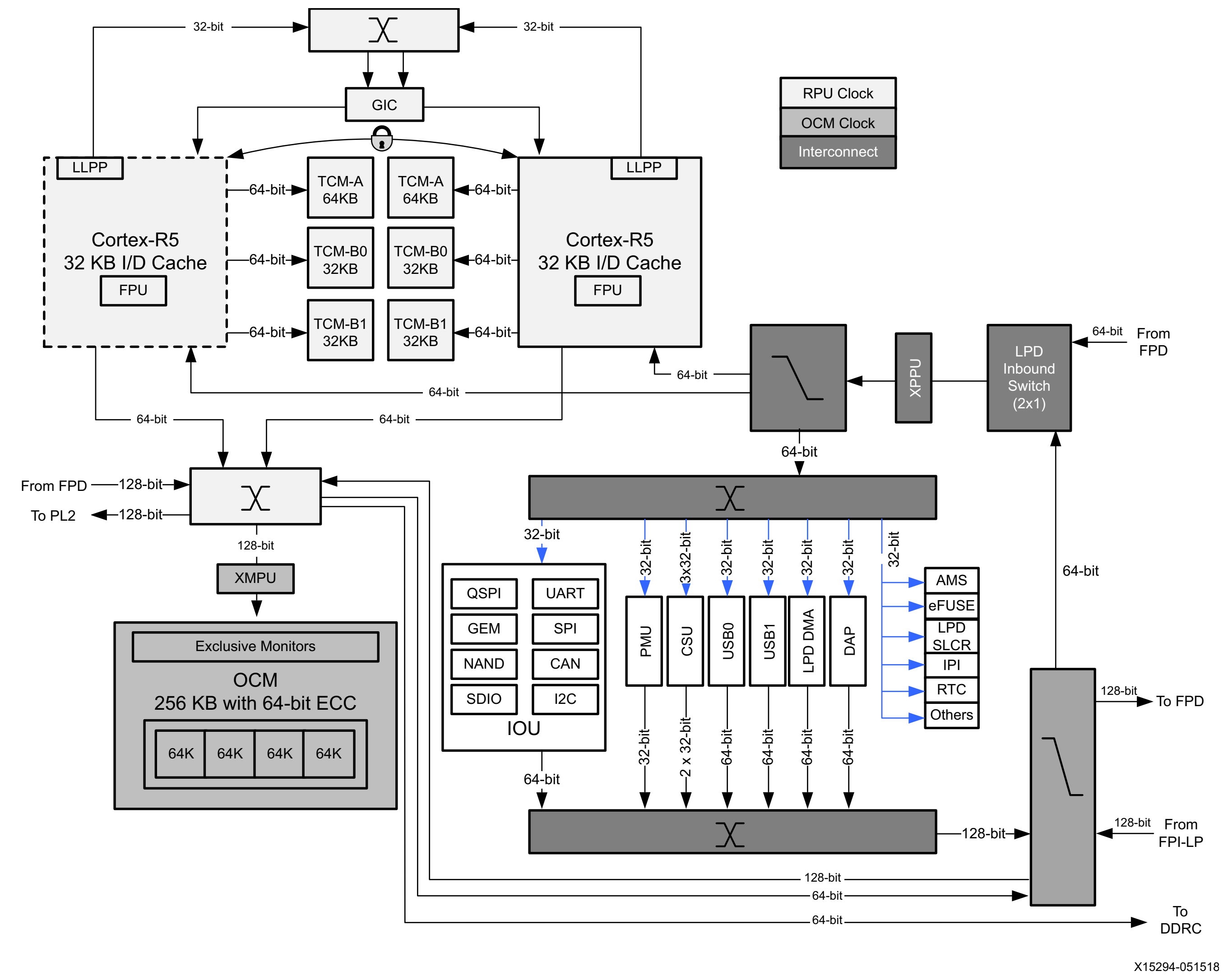


Figure 4-1: **System View of RPU**

Note: The switches have been combined in Figure 4-1 for more details refer to Figure 15-1.

RPU Pin Configuration

The following table describes the real-time processor configuration signals.

Table 4-1: **RPU Pin Configuration**

Pins	Selection	Description
VINITHIm	SLCR configurable (default 1)	Reset V-bit value. When High indicates HIVECS mode at reset.
CFGEE	SLCR configurable (default 0)	Data endianness at reset.
CFGIE	SLCR configurable (default 0)	Instruction fetch endianness.
TEINIT	SLCR configurable (default 0)	Arm or fetch at reset. 0 = Arm.
CFGNMFIm	SLCR configurable (default 0)	Non-maskable FIQ. 0 = maskable.
INITPPXm	`0x1`	AXI peripheral interface enabled at reset.
SLBTCMSBm	SLCR configurable (default 0)	B0TCM and B1TCM interleaving by addr[3].
INITRAMAm	`0x0`	Enable ATCM.
INITRAMBm	`0x1`	Enable BTCM.
ENTCM1IFm	`0x1`	Enable B1TCM interface.
LOCZRAMAm	`0x1`	When High indicates ATCM initial base address is zero.
PPXBASEm	Based on global address map	Base address of AXI peripheral interface. Must be size aligned.
PPXSIZEm	16 MB	Size of AXI peripheral interface.
PPVBASEm	Same as PPXBASEm	Base address of virtual-AXI peripheral interface.
PPVSIZEm	8 KB	Size of virtual-AXI peripheral interface.
GROUPID[3:0]	`0x1`	ID of the Cortex-R5F processor group.
DBGNOCLKSTOP	default (0)	Clock control when entering standby.
SLSPLIT	default (0)	Processor mode
SLCLAMP	default (1)	Output clamps for redundant processor.
TCM_COMB	default (1)	Combine TCMs of RPU0 and RPU1.
TCM_WAIT	default (0)	Insert wait states in TCM access.
TCM_CLK_CNTL	default (0)	TCM clock disable (all TCMs, both RPU processors).
GIC_AXPROT	default (0)	GIC access security setting. This bit is equivalent to AxPROT[1] on AXI bus.

Note: For more information on Configuration Signals, see the *ARM Cortex-R5F and Cortex-R5F Technical Reference Manual* [Ref 47].

RPU CPU Configuration

The RPU MPCore has two Cortex-R5F processors that can operate independently or in lock-step together. This section describes the CPU arrangements supported and the functionality of each arrangement.

Split/Lock

Two CPUs are included in this configuration. The processor group can operate in one of two modes.

- Split mode operates as a twin-CPU configuration. Also known as performance mode.
- Locked mode operates as a redundant CPU configuration. Also known as safety mode.

Lock-Step Operation

When the Cortex-R5F processors are configured to operate in the lock configuration, only one set of CPU interfaces are used. Because the Cortex-R5F processor only supports the static split/lock configuration, switching between these modes is only permitted right after the processor group is brought out of reset. The input signals SLCLAMP and SLSPLIT control the mode of the processor group. These signals control the multiplex and clamp logic in the locked configuration. When the Cortex-R5F processors are in the lock-step mode (Figure 4-2), 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 lock-step processor. The size of each ATCM and BTCM becomes 128 KB with BTCM interleaved accesses from the processor and AXI slave interface.*

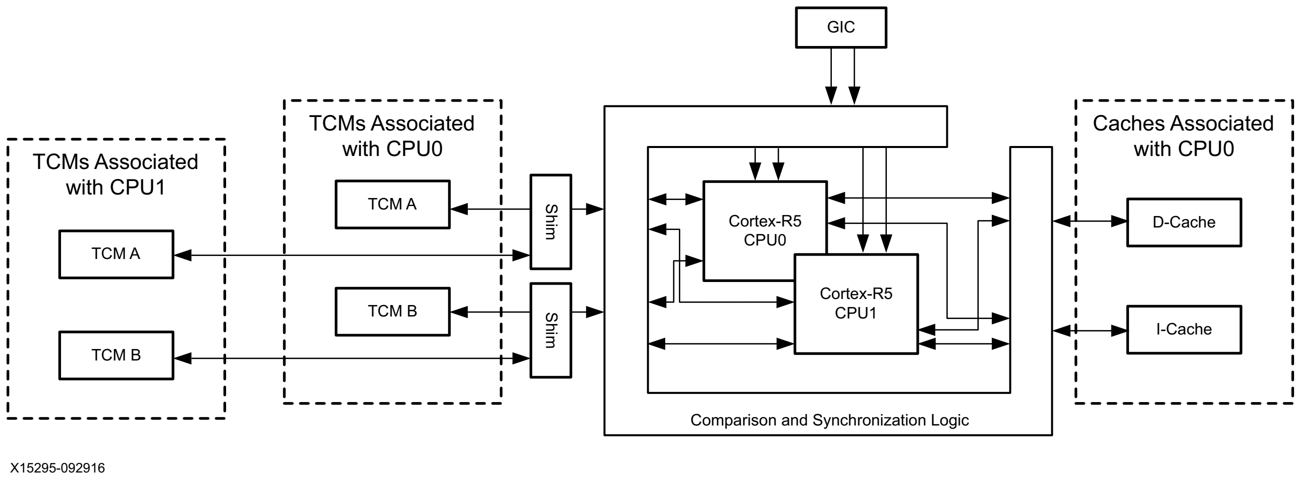

GIC
TCMs Associated with CPU1
TCM A
TCM B
TCMs Associated with CPU0
TCM A
TCM B
Shim
Shim
Cortex-R5 CPU0
Cortex-R5 CPU1
Comparison and Synchronization Logic
Caches Associated with CPU0
D-Cache
I-Cache
X15295-092916

Figure 4-2: **RPU Cortex-R5 Processor Lock-step Mode**

Error Correction and Detection

The Cortex-R5F processor supports error checking and correction (ECC) data schemes. For each aligned data set, a number of redundant code bits are computed and stored with the data. This enables the processor to detect up to two errors in the data set or its code bits, and correct any single error in the data set or its associated code bits. This is sometimes referred to as a single-error correction, double-error detection (SEC-DED) ECC scheme.

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 160 interrupt lines using this mechanism. The 160 SPIs are divided into five, 32-bit APB registers. The RPU implements an interrupt register and an interrupt mask register. The logic in Figure 4-3 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.

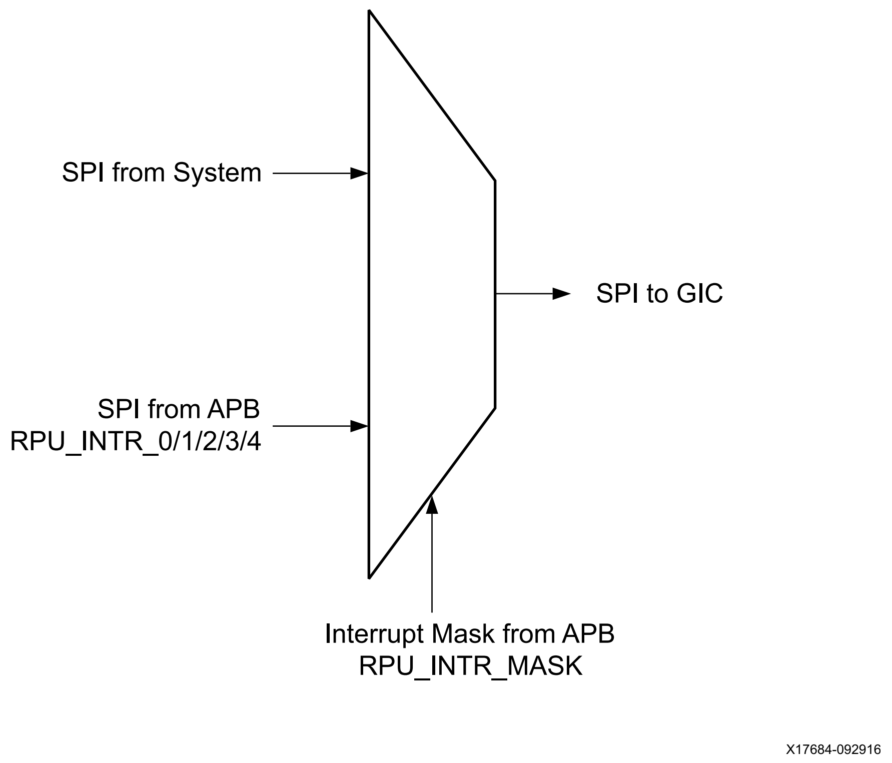


Figure 4-3: **RPU Interrupt Injection**

Table 4-2 lists the mapping of the SPI bits.

Table 4-2: **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>

Level2 AXI Interfaces

There are three distinct advanced eXtensible interfaces (AXI) to the rest of the MPSoC. The first is the AXI master interface. There is also a separate AXI peripheral interface that connects to the GIC and an AXI slave port provided to allow external masters to access ICACHE, DCACHE, and TCM RAMs. Access from the AXI slave port to the caches is only provided for use during debug. The L2 AXI interfaces enable the L1 memory system to have access to peripherals and to external memory using an AXI master and AXI slave port and the peripheral ports.

Memory Protection Unit

The memory protection unit (MPU) works with the L1 memory system to control the accesses to and from L1 cache and external memory. For a detailed description of the MPU, refer to the *Cortex-R5F Technical Reference Manual* [Ref 47].

The MPU enables you to partition memory into regions and set 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 has a maximum of 16 regions.

Using the MPU memory region programming registers you can specify the following for each region.

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

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 for 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.

Power Management

Each CPU in the Cortex-R5F processor supports three power management modes (Table 4-3) from run to shutdown, with decreasing levels of power consumption, but increasing entry and exit costs.

Table 4-3: **Power Management Modes**

Mode	CPU Clock Gated	CPU Logic Powered	TCM Memory Retention	Exit to Run Mode
Run	No	Yes	Yes	N/A
Standby	When idle	Yes	Yes	Pipeline restart.
Shutdown	Yes	No	No	Pipeline restart restore registers and configuration from memory invalidate caches and re-initialize caches and TCMs.

Exception Vector Pointers

The exception vector pointers (EVP) refer to the base-address of exception vectors (for reset, IRQ, FIQ, etc). The reset-vector starts at the base-address and subsequent vectors are on 4-byte boundaries. The Cortex-R5F processor EVPs are determined as follows.

- If the Cortex-R5F processor SCTRL.V register bit is 0, then exception vectors start from `0x0000_0000` (`LOVEC`).
- If the Cortex-R5F processor SCTRL.V register bit is 1, then exception vectors start from `0xFFFF_0000` (`HIVEC`).

The reset value of SCTRL.V is taken from the Cortex-R5F processor VINITHIm pin value, which is driven by the Zynq UltraScale+ MPSoC SLCR bit.

At system boot, the Cortex-R5F processor exception vectors (i.e., VINITHIm pin-value) default to HIVEC, which is mapped in the OCM. The FSBL (running on the Cortex-R5F processor) is expected to change the Cortex-R5F processor exception vectors by changing both the Zynq UltraScale+ MPSoC SLCR to change the value of the VINITHIm pin and the Cortex-R5F processor SCTRL.V bit to `LOVEC`. The Cortex-R5F processor exception vectors should remain at `LOVEC`.

RECOMMENDED: *Xilinx does not recommend that you change the exception vector. Changing the EVP to HIVEC will result in increased interrupt latency and jitter. Also, if the OCM is secured and the Cortex-R5F processor is non-secured, then the Cortex-R5F processor cannot access the HIVEC exception vectors in the OCM.*

System Register Overview

Table 4-4 provides an overview of the RPU system registers.

Table 4-4: **RPU Registers**

Register name	Description
RPU_GLBL_CNTL	Global control register for the RPU
RPU_GLBL_STATUS	Miscellaneous status information for the RPU
RPU_ERR_CNTL	Error response enable/disable register
RPU_RAM	Control for extra features of the RAMs
RPU_ERR_INJ	Reserved
RPU_CCF_MASK	Common cause signal mask register
RPU_INTR_0-4	RPU interrupt injection registers
RPU_INTR_MASK_0-4	RPU interrupt injection mask registers
RPU_CCF_VAL	Common cause signal value register
RPU_SAFETY_CHK	RPU safety check register
RPU_0_CFG	Configuration parameters specific to RPU0
RPU_0_STATUS	RPU0 status register
RPU_0_PWRDWN	Power-down request from the Cortex-R5F processors
RPU_0_ISR	Interrupt status register
RPU_0_IMR	Interrupt mask register
RPU_0_IEN	Interrupt enable register
RPU_0_IDS	Interrupt disable register
RPU_0_SLV_BASE	Slave base address register
RPU_0_AXI_OVER	RPU0 AXI override register
RPU_1_CFG	Configuration parameters specific to RPU1

Table 4-4: **RPU Registers *(Cont'd)***

Register name	Description
RPU_1_STATUS	RPU1 status register
RPU_1_PWRDWN	Power-down request from the Cortex-R5F processors
RPU_1_ISR	Interrupt status register
RPU_1_IMR	Interrupt mask register
RPU_1_IEN	Interrupt enable register
RPU_1_IDS	Interrupt disable register
RPU_1_SLV_BASE	Slave base address register
RPU_1_AXI_OVER	RPU1 AXI override register

Tightly Coupled Memory

Tightly-coupled memories (TCMs) are low-latency memory that provide predictable instruction execution and predictable data load/store timing. Each Cortex-R5F processor contains two 64-bit wide 64 KB memory banks on the ATCM and BTCM ports, for a total of 128 KB of memory. The division of the RAMs into two banks, and placing them on ports A and B, allows concurrent accesses to both banks by the load-store, instruction prefetch, or AXI slave ports.

The BTCM memory bank is divided into two 32 KB ranks that are connected to the BTCM-0 and BTCM-1 ports of the Cortex-R5F processors. There are two TCM interfaces that permit connection to configurable memory blocks of tightly-coupled memory (ATCM and BTCM).

- An ATCM typically holds interrupt or exception code that must be accessed at high speed, without any potential delay resulting from a cache miss.
- A BTCM typically holds a block of data for intensive processing, such as audio or video processing.

The block diagram of RPU along with the TCMs is shown in Figure 4-4.

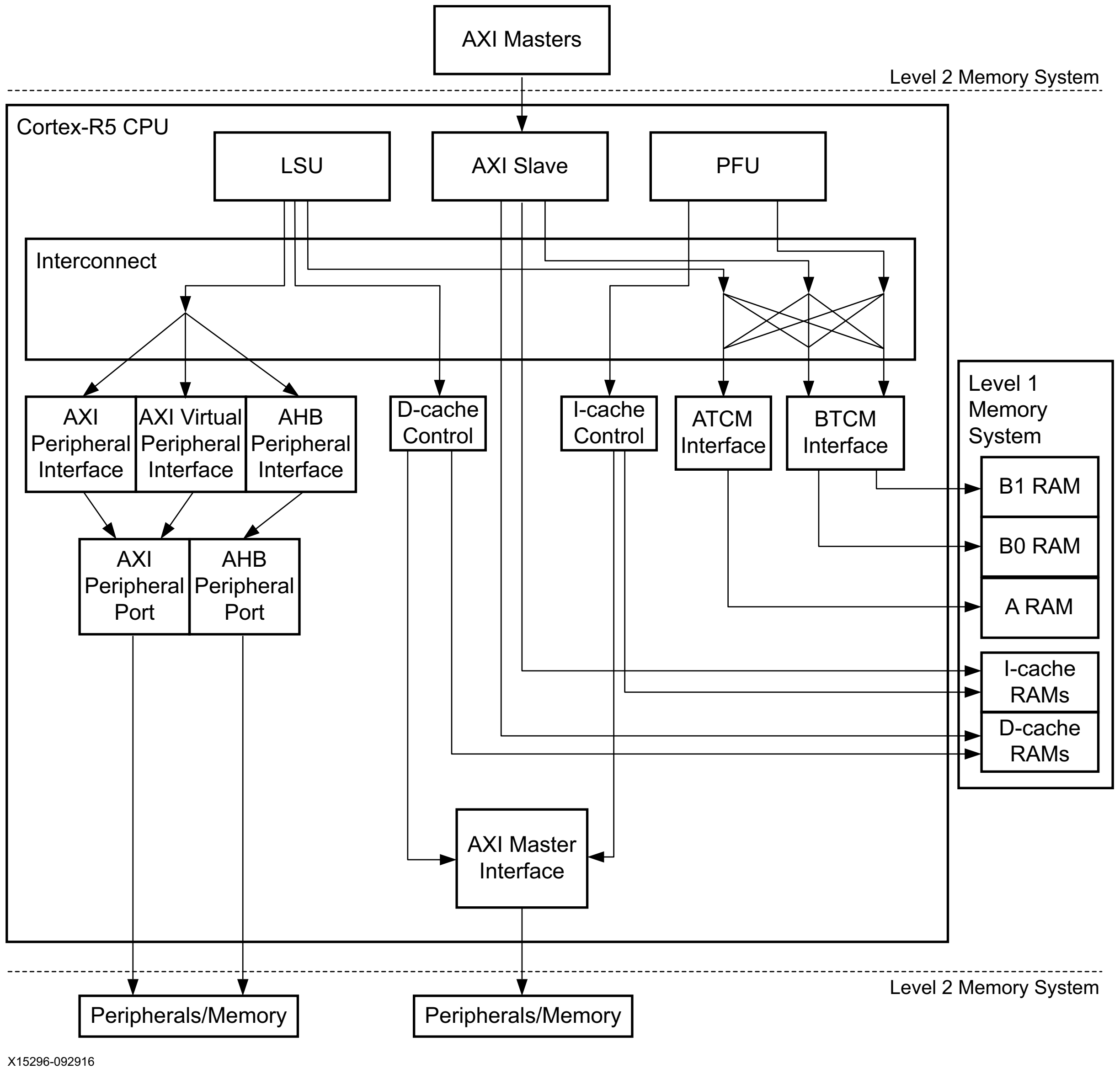


Figure 4-4: **Block Diagram of RPU with TCMs**

The entire 256 KB of TCM can be accessed by R5_0 (in lock-step mode). The PMU block controls power gating to each of the 64 KB TCM banks, through the system power and configuration state (SPCS) registers.

Tightly Coupled Memory Functional Description

The Cortex-R5F processors in the RPU block operate in normal (split) and lock-step configuration. Each of these operating modes also defines the TCM access methods. The following sections describe various TCM access methods.

Normal (Split) Operation

The 2-bank 128 KB TCM support for each Cortex-R5F processor in the split mode includes the following.

- Each TCM is 64 KB.
- One BTCM is composed of two ranks allowing interleaved accesses.
- 32-bit ECC support is available in both normal and lock-step mode.
- TCMs can be combined for a total of 256 KB (128 KB each of ATCM and BTCM) for use by R5_0 in lock-step mode.
- External TCM access from AXI slave interfaces.

Lock-step Operation

When the Cortex-R5F processors are in the lock-step mode (Figure 4-5), there should be code in the reset handler to ensure that the distributor within the GIC dispatches interrupts only to CPU0. During the lock-step operation, the TCMs that are associated with the redundant processor become available to the lock-step processor. The size of ATCM and BTCM become 128 KB each with BTCM supporting interleaved accesses from processor and AXI slave interface.

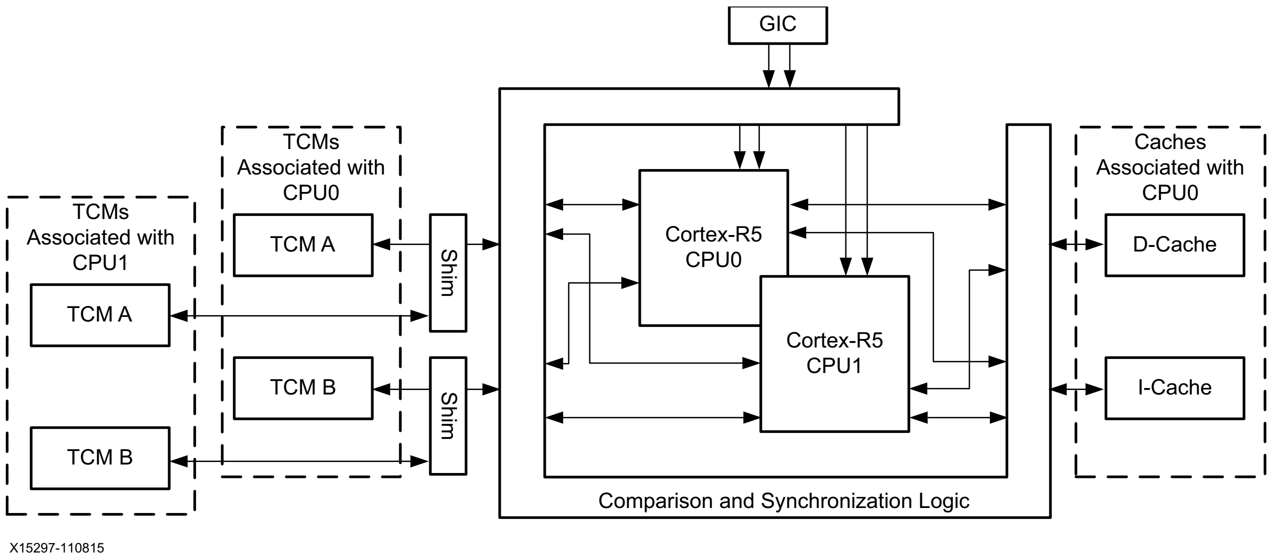

GIC
TCMs Associated with CPU1
TCM A
TCM B
TCMs Associated with CPU0
TCM A
TCM B
Shim
Shim
Cortex-R5 CPU0
Cortex-R5 CPU1
Comparison and Synchronization Logic
Caches Associated with CPU0
D-Cache
I-Cache
X15297-110815

Figure 4-5: **TCMs in Lock-step Mode**

Tightly Coupled Memory Address Map

TCMs are mapped in the local address space of each Cortex-R5F processor, but they are also mapped in the global address space for access from any master. The address maps from the RPU point of view and from the global address space are shown in Table 4-5.

Table 4-5: **TCM Address Map**

	R5_0 View (Start Address)	R5_1 View (Start Address)	Global Address View (Start Address)
Split mode			
R5_0 ATCM (64 KB)	`0x0000_0000`	N/A	`0xFFE0_0000`
R5_0 BTCM (64 KB)	`0x0002_0000`	N/A	`0xFFE2_0000`
R5_0 instruction cache (32 KB)	I-Cache	N/A	`0xFFE4_0000`
R5_0 data cache (32 KB)	D-Cache	N/A	`0xFFE5_0000`
R5_1 ATCM (64KB)	N/A	`0x0000_0000`	`0xFFE9_0000`
R5_1 BTCM (64KB)	N/A	`0x0002_0000`	`0xFFEB_0000`
R5_1 instruction cache (32 KB)	N/A	I-Cache	`0xFFEC_0000`
R5_1 data cache (32 KB)	N/A	D-Cache	`0xFFED_0000`
Lock-step mode			
R5_0 ATCM (128KB)	`0x0000_0000`	N/A	`0xFFE0_0000`
R5_0 BTCM (128KB)	`0x0002_0000`	N/A	`0xFFE2_0000`
R5_0 instruction cache (32 KB)	I-Cache	N/A	`0xFFE4_0000`
R5_0 data cache (32 KB)	D-Cache	N/A	`0xFFE5_0000`
R5_1 slave port is not accessible in lock-step mode.			

TCM Access from a Global Address Space

The following address can be routed to the Cortex-R5F processors slave port:

- If `0xFFE6_0000` > ReqAddr[31:0] ≥ `0xFFE0_0000`, then route request to R5_0
- If `0xFFEE_0000` > ReqAddr[31:0] ≥ `0xFFE9_0000`, then route request to R5_1

Figure 4-6 shows the address map views of the RPU and APU CPUs. The TCMs are mapped into a global address space that is accessible (via RPU slave port) by an APU or any other master that can access a global address space. In addition, TCMs are aliased in the local view of the RPU starting at address `0x0000-0000`.

A TCM cannot be accessed when the Cortex-R5F processor is in reset. The R5F processor must be in active or halt state to allow another master to access the TCM. The Cortex-R5F processor connection to TCM is a direct low-latency path that does not go through the SMMU. There is no protection to stop the Cortex-R5F processor from accessing the TCM.

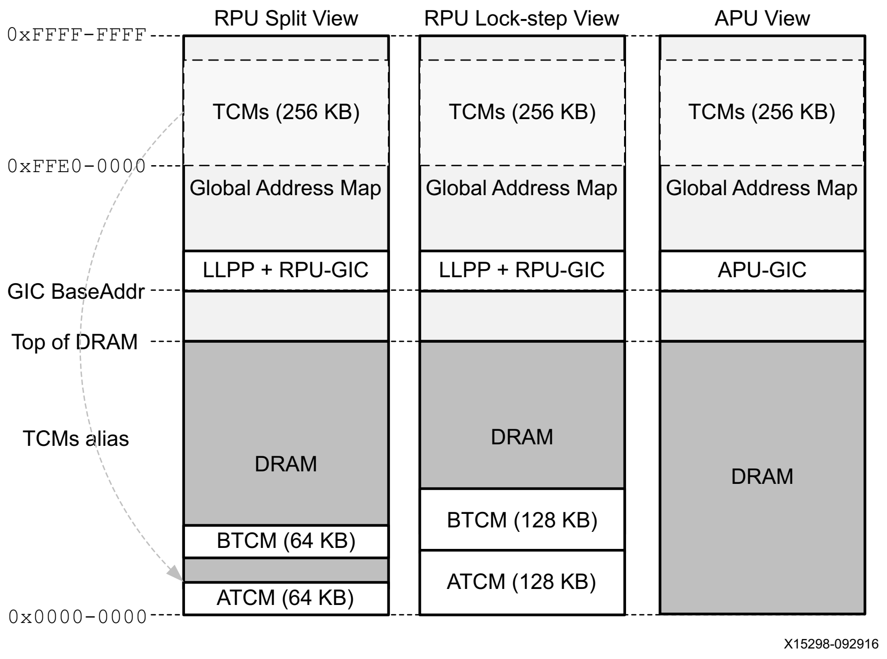


Figure 4-6: **APU and RPU CPUs TCM Address Map**

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`.

There are cases where the CCI-400 will generate transactions using the same master ID as that of the R5_0. If that region is coherent and protected by an XMPU it will generate an error, unless the R5_0 is added to the allowed master ID list of the XMPU. If access to that region by the R5_0 is not compatible with the safety or security goal of the system, the user can either run the R5F application using only R5_1 (leaving R5_0 unused) or skip the use of coherency and not add R5_0 to the XMPU allowed list.

Lock-step Sequence in Cortex-R5F Processors

The following sequence is used to enable the lock-step mode of the Cortex-R5F processors.

```
;SVC out of reset
   MOV r0,#0
   MOV r1,#0
   MOV r2,#0
   MOV r3,#0
   MOV r4,#0
   MOV r5,#0
   MOV r6,#0
   MOV r7,#0
   MOV r8,#0
   MOV r9,#0
   MOV r10,#0
   MOV r11,#0
   MOV r12,#0
   MOV r13,#0x10000
;SP - Choose a suitable stack pointer value based on your system
   MOV r14, #0;LR
;User (Sys)
   MSR CPSR_cxsf,#0x1F
   MOV r13,#0x70000
;SP - Choose a suitable stack pointer value based on your system
   MOV r14,#0;LR

;FIQ
   MSR CPSR_cxsf,#0x11
   MOV r8,#0
   MOV r9,#0
   MOV r10,#0
   MOV r11,#0
   MOV r12,#0
   MOV r13,#0x60000
;SP - Choose a suitable stack pointer value based on your system
   MOV r14,#0;LR

;IRQ
   MSR CPSR_cxsf,#0x12
   MOV r13,#0x50000
;SP - Choose a suitable stack pointer value based on your system
   MOV r14,#0;LR

;Undef
   MSR CPSR_cxsf,#0x1B
   MOV r13,#0x40000
;SP - Choose a suitable stack pointer value based on your system
   MOV r14,#0;LR

;Abort
   MSR CPSR_cxsf,#0x17
   MOV r13, #0x30000
;SP - Choose a suitable stack pointer value based on your system
   MOV r14, #0;LR

;Return to SVC
   MSR CPSR_cxsf,#0x13
```

```
FUNC(asm_init_vfp_regs)
   mov>----r1,#0
   vmov d0,r1,r1
   vmov d1,r1,r1
   vmov d2,r1,r1
   vmov d3,r1,r1
   vmov d4,r1,r1
   vmov d5,r1,r1
   vmov d6,r1,r1
   vmov d7,r1,r1
   vmov d8,r1,r1
   vmov d9,r1,r1
   vmov d10,r1,r1
   vmov d11,r1,r1
   vmov d12,r1,r1
   vmov d13,r1,r1
   vmov d14,r1,r1
   vmov d15,r1,r1
   cmp  r0,#1
   beq asm_init_vfp_regs32
   bx  lr
asm_init_vfp_regs32:
   vmov d16,r1,r1
   vmov d17,r1,r1
   vmov d18,r1,r1
   vmov d19,r1,r1
   vmov d20,r1,r1
   vmov d21,r1,r1
   vmov d22,r1,r1
   vmov d23,r1,r1
   vmov d24,r1,r1
   vmov d25,r1,r1
   vmov d26,r1,r1
   vmov d27,r1,r1
   vmov d28,r1,r1
   vmov d29,r1,r1
   vmov d30,r1,r1
   vmov d31,r1,r1
   bx  lr
```

The ECC for the cache RAMs is initialized as part of the initial invalidation after reset. The cache ECC checking must be enabled during the invalidation using the following sequence.

```
DSB
MRC p15, 0, r1, c1, c0, 1  ;Read ACTLR
ORR r1, r1, #(0x1 << 5)    ;Set Bits [5:3] = 0b101
BIC r1, r1, #(0x1 << 4)    ;to enable ECC no forced
ORR r1, r1, #(0x1 << 3)    ;write-through
MCR p15, 0, r1, c1, c0, 1  ;Write ACTLR ISB

MCR p15, 0, r0, c7, c5, 0  ;Invalidate All instruction caches
MCR p15, 0, r0, c15, c5, 0 ;Invalidate All Data caches DSB ISB
```

If you have ECC on the TCMs, then the initial accesses to the TCM locations also needs to ensure that the ECC locations are updated correctly.

Graphics Processing Unit

Introduction

The GPU is a 2D and 3D graphics subsystem based on the Arm® Mali™-400 MP2 hardware accelerator.

Note: The GPU is not supported in the Zynq UltraScale+ CG family.

Features

The GPU consists of the following components.

- One geometry processor (GP)
- Two pixel processors (PP)
- Shared 64 KB L2 cache controller (L2)
- Individual memory management units (MMU) for the GP and each PP
- 128-bit AXI master bus interface

Features achieved by the GPU components.

- OpenGL ES 1.1 and 2.0 (with software support)
- OpenVG 1.1 (with software support)
- SIMD engine features
  - 32-bit floating point arithmetic per the IEEE standard (IEEE Std 754)
  - 4-way 32-bit simultaneous instruction execution
- Vertex loader DMA unit
- High data latency tolerance
- Advanced 4x and 16x anti-aliasing
- Texture sizes of up to 4096 x 4096 pixels
- Ericsson texture compression (ETC) to reduce memory bandwidth

- Local cache to reduce memory bandwidth
- Extensive texture formats:
    - RGBA 8888, 565, 1556
    - Mono 8 and Mono 16
    - YUV format
- Automatic load balancing across the graphics shader engines
- Stage-1 virtual address translation

Power Domains

The GPU is powered by three power sources:

- Control registers, L2 cache, and geometry processor (FPD directly)
- Pixel processor 0 (FPD with power island control PP0)
- Pixel processor 1 (FPD with power island control PP1)

The GPU can operate with one, both, or none of the pixel processors. However, the programming environment might require all three system elements to be powered on. If both pixel processors are needed, the power-up sequence should be staggered to minimize current surges on the device.

Clocking Domain

The GPU runs based only on the GPU_REF_CLK clock. All interfaces, including APB and core, are clocked based on the GPU_REF_CLK clock. The values of PLL Source and clock frequency are configured using the GPU_REF_CTRL register. See Chapter 37, PS Clock Subsystem for more information on GPU_REF_CLK.

Performance

Performance values change with the operating frequency and vary by device. Operating frequency specifications are reported in the *Zynq UltraScale+ MPSoC Data Sheet: DC and AC Switching Characteristics* [Ref 2]. The following example peak performance values are for a 400 MHz operating frequency.

- Pixel fill rate: 800 Mpixel/sec
- Vertex processing rate: 40 Mvertex/sec

Graphics Processing Unit Functional Description

Figure 5-1 shows the block diagram of the GPU.

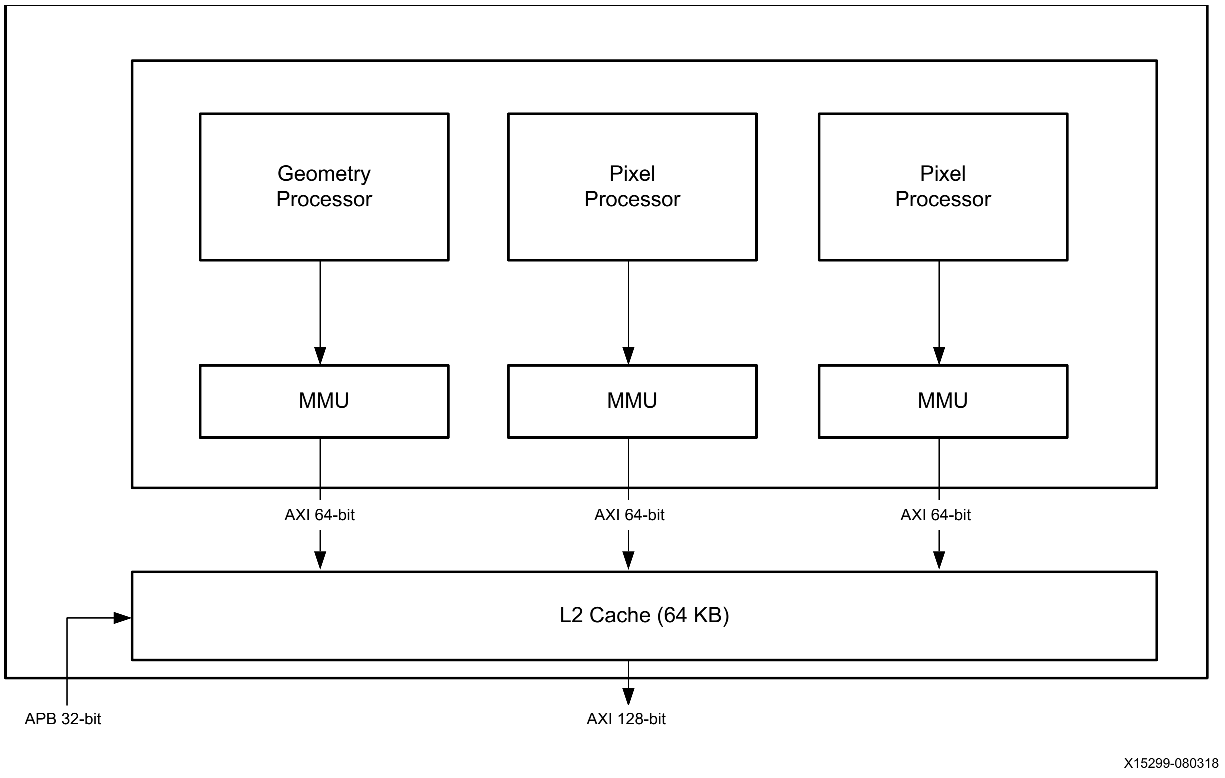


Figure 5-1: **GPU Block Diagram**

Geometry Processor

The following figure shows a top-level view of the geometry processor in the GPU.

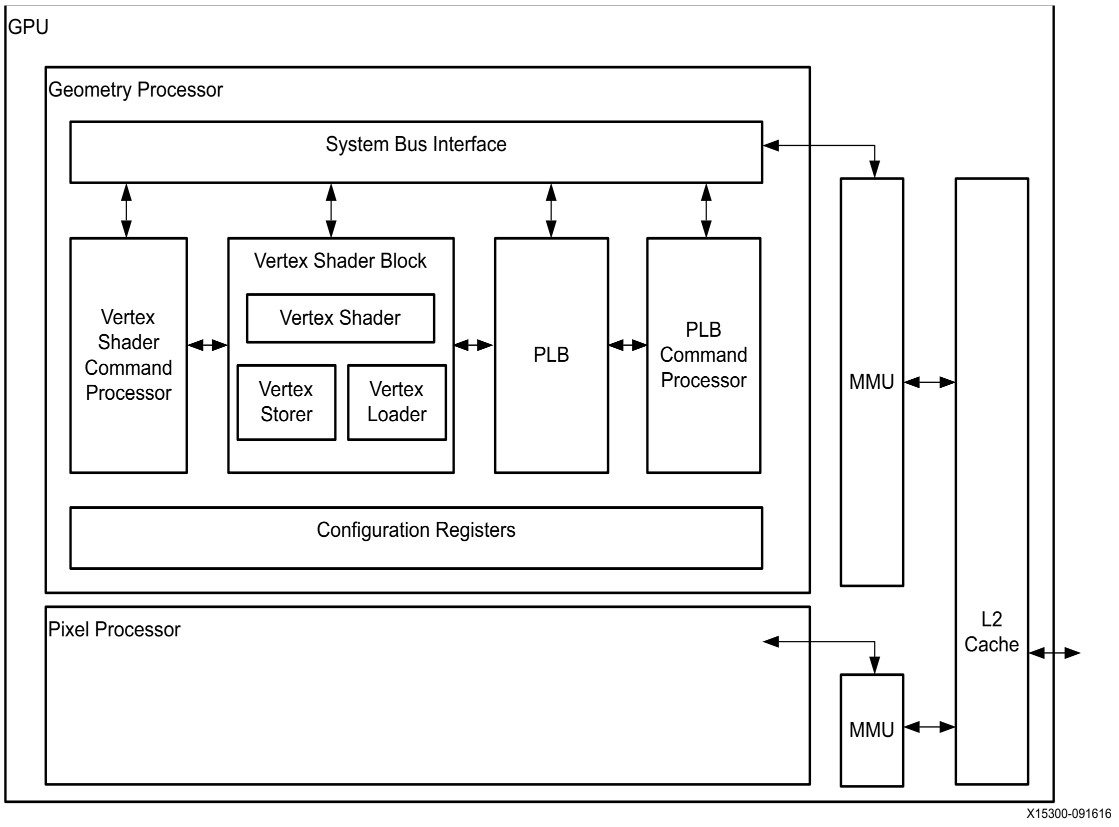


Figure 5-2: **GPU Geometry Processor Block Diagram**

The geometry processor consists of the following.

- A Vertex Shader command processor that reads and executes commands from a command list stored in memory.
- A Vertex Shader Core that loads data for processing, performs the required calculations for each vertex, stores data from output registers in memory, and then exports data to integer or floating point numbers of different sizes.
- A Polygon List Builder unit that creates lists of polygons that the pixel processor must draw.
- The polygon list builder (PLB) command processor reads and executes commands from the command list stored in memory.

Vertex Processing

The geometry processor performs the vertex processing tasks shown in Table 5-1.

Table 5-1: **Vertex Processing Tasks Performed by the Geometry Processor**

Processing Task	Description
Transform and lighting	The geometry processor scales, rotates, and positions the geometry of objects in the scene, and also calculates and assigns values to the vertices.
Primitive assembly	The PLB links vertices together to form different primitives.
Culling	This step discards polygons that must not be rendered.
Primitive list assembly	The pixel processor is a tile-based renderer. The geometry processor prepares a list of all primitives required for the pixel processor to render. For each primitive, the PLB writes a list entry for each tile that the primitive touches.
Rendering	Describes the various tasks the pixel processor performs. During rendering, the pixel processor uses the information from the polygon list to produce a final frame buffer image.

Vertex Shader

Vertex shader consists of three main stages, loader, shader, and storer.

Vertex Loader

The vertex loader is a DMA unit that loads per-vertex data for processing. It can accept data from up to 16 distinct streams, each corresponding to one of the 16 input registers. For each stream, it permits the specification of any of the following data formats.

- 1, 2, 3, or 4 values in 16-bit, 24-bit, or 32-bit floating-point formats
- 1, 2, 3, or 4 values in 8-bit, 16-bit, or 32-bit signed or unsigned fixed-point values
- 1, 2, 3, or 4 values in 8-bit, 16-bit, or 32-bit signed or unsigned normalized values

For each vertex stream, you must specify a stride in 1-byte increments. A stride is an offset between two data sets.

Vertex Shader Core

The vertex shader core (Figure 5-3) performs most of the required calculations for each vertex. The vertex shader runs a program on each vertex of a 3D scene typically performing transform and lighting (T&L). The program is limited to 512 instructions with limited flow control. The instructions of the program work on vector data that is optimized for operations in length-4 vectors and smaller vectors.

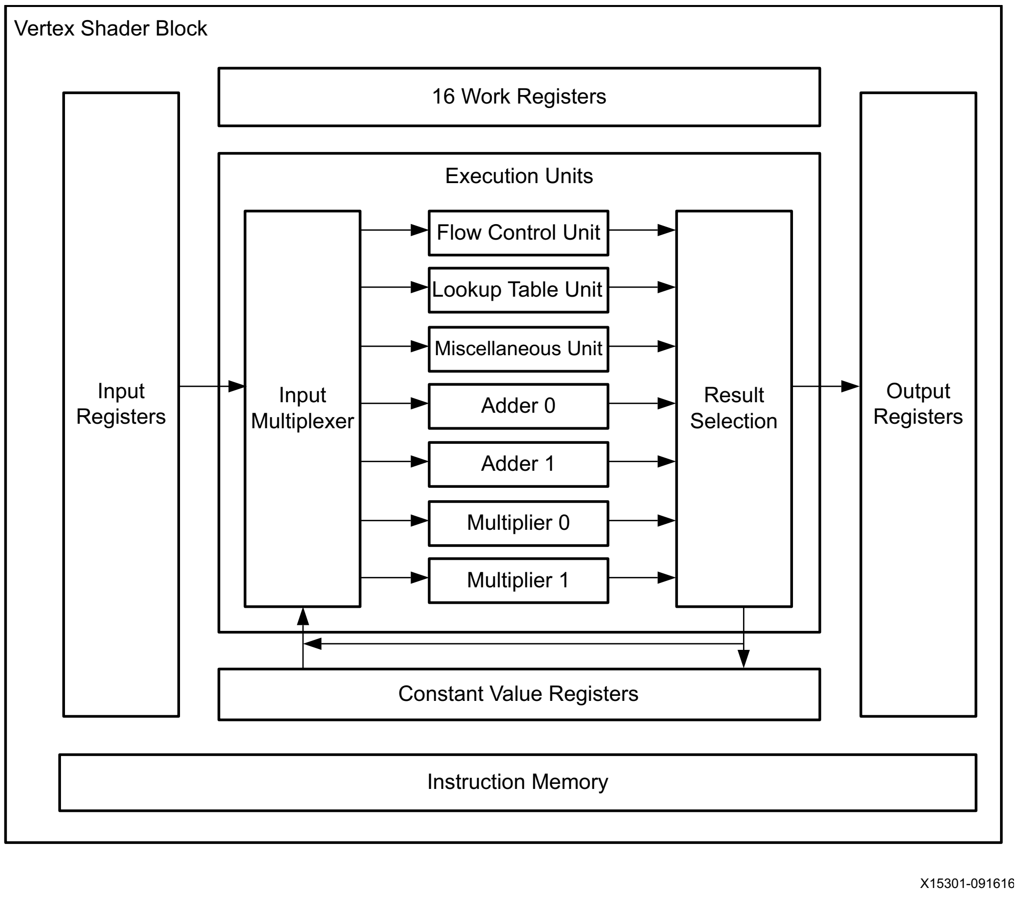


Figure 5-3: **Vertex Shader Core Block Diagram**

Vertex Storer

The vertex storer stores data from the output registers of the vertex shader to memory. The vertex storer can export data to FP24, FP16, 32-bit integer, 16-bit integer, and 8-bit integer.

See the geometry processor control register and vertex shader registers for a description of how to configure the vertex storer.

Polygon List Builder

The polygon list builder (PLB) creates lists of the polygons that the pixel processor must draw. For each polygon in a scene, the PLB decides which tiles the polygon covers, and adds the polygon to the lists that draw those tiles. The PLB only adds a polygon to lists where the polygon might have to be drawn, reducing the work involved when the pixel processor renders the scene.

The PLB also discards polygons that are certain not to be visible, based on the following criteria.

- Invalid polygons. For example, any coordinate that is non-numeric, an x or y coordinate that is infinity, or where the area is zero.
- Polygons outside the view frustum.
- Back-facing polygons as defined by the *OpenGL ES Common Profile* specification.
- Polygons outside the current scissoring box.

The PLB can handle up to 512 lists to support the tile-based rendering mode of the pixel processor efficiently; however, the default is 300. For QVGA or lower resolutions, 300 lists are normally sufficient to make one list for every tile in the scene. For higher resolutions, each list covers multiple tiles. This process is known as binning.

Each list ends with a return command. The driver creates a master tile list containing sets of commands that perform tasks such as beginning new tiles and calling polygon lists. This master tile list is input to the pixel processor polygon list reader.

The PLB requires a variable amount of memory to store the polygon lists. Memory is allocated in blocks of 128, 256, 512, or 1024 bytes, as configured in the GP_PLB_CONF_REG_PARAMS register. Further details are in the *Zynq UltraScale+ MPSoC Register Reference* (UG1087) [Ref 4].

The driver must allocate the initial memory, consisting of an array of the selected number of polygon lists. More memory is allocated automatically by the PLB from a heap area configured through the GP_PLB_CONF_REG_HEAP_START_ADDR and GP_PLB_CONF_REG_HEAP_STOP_ADDR registers. When this heap is exhausted, an interrupt is generated, and the driver must allocate more memory for the heap.

Pixel Processor

The pixel processor uses a list of primitives generated by the geometry processor to produce a final image that is displayed on the screen. There are two pixel processors. Figure 5-4 is a top-level diagram of a pixel processor.

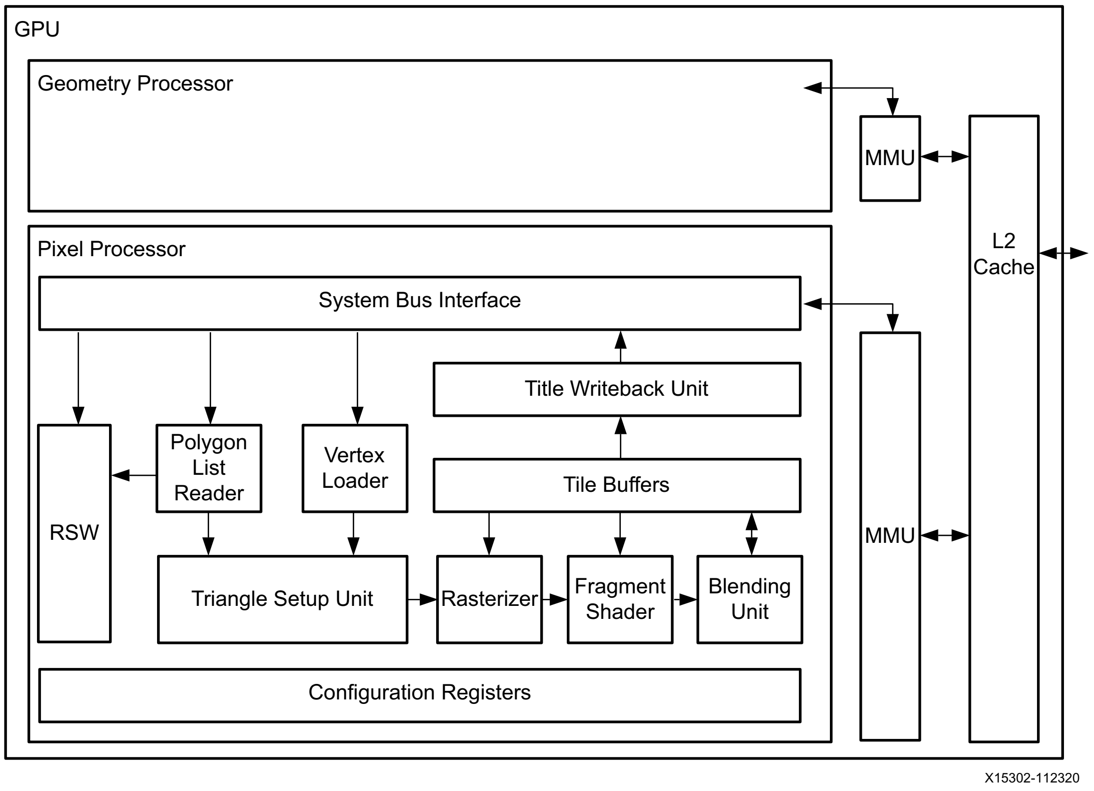


Figure 5-4: **Pixel Processor Block Diagram**

The pixel processor consists of the following.

- A polygon list reader that reads the polygon lists from main memory and executes commands from the lists.
- The render state words (RSWs) component is a data structure in main memory that contains the render state of polygons. The different pipeline stages in the renderer each reference the RSWs to determine how to process the primitives.
- The vertex loader fetches the required vertices from memory for each primitive in the polygon list.
- The triangle setup unit takes data from the vertex loader and polygon list reader and uses vertex data to compute coefficients for edge equations and varying interpolation equations.
- The rasterizer takes coefficients and equations from the triangle setup unit and uses these to divide polygons into fragments.

- The fragment shader is a programmable unit that calculates how each fragment of a primitive looks.
- The blending unit blends the calculated fragment value into the current frame buffer value at that position.
- The tile buffers take inputs from the fragment shader. The buffers perform various tests on the fragments, for example, Z tests and stencil tests. When the tile is fully rendered it is written to the frame buffer.
- The writeback unit writes the content of the tile buffer to system memory after the tile is completely rendered.

The pixel processor performs the rendering tasks shown in Table 5-2.

Table 5-2: **Pixel Processor Rendering Tasks**

Processing Task	Description
Triangle setup	To prepare the primitive for rendering by calculating various data that is required to rasterize and shade the primitive.
Rasterization	To divide the primitive into independent fragments. These are fragment-sized pieces of primitive that the shader pipeline processes. Fragments that could be visible proceed to the fragment shading stage, and fragments that are certain not to be visible are discarded.
Fragment shading	To determine how the fragment actually looks. In general, the pixel processor calculates a color for the fragment.
Blending	The fragment is blended into the frame buffer to produce the final image.
Producing the frame buffer content	After blending, the fragment becomes a fragment at a certain position in the tile buffer. If no other fragment overwrites that position, the fragment becomes a fragment in the final frame. Multi-sampling techniques to obtain better quality final images can be applied to the fragment at this stage. When the internal tile buffer is completely rendered, it is written to the frame buffer in main memory.

Pixel Processor Fragment Shader

The fragment shader is a programmable unit that calculates the appearance of each fragment of a primitive. The fragment shader program specified in the RSW for the primitive is executed for each fragment produced by the rasterizer. The fragment shader program consists of very-long instruction words (VLIW), and can use any number of functional units in a single instruction.

Figure 5-5 shows the functional units available for the fragment-shader program.

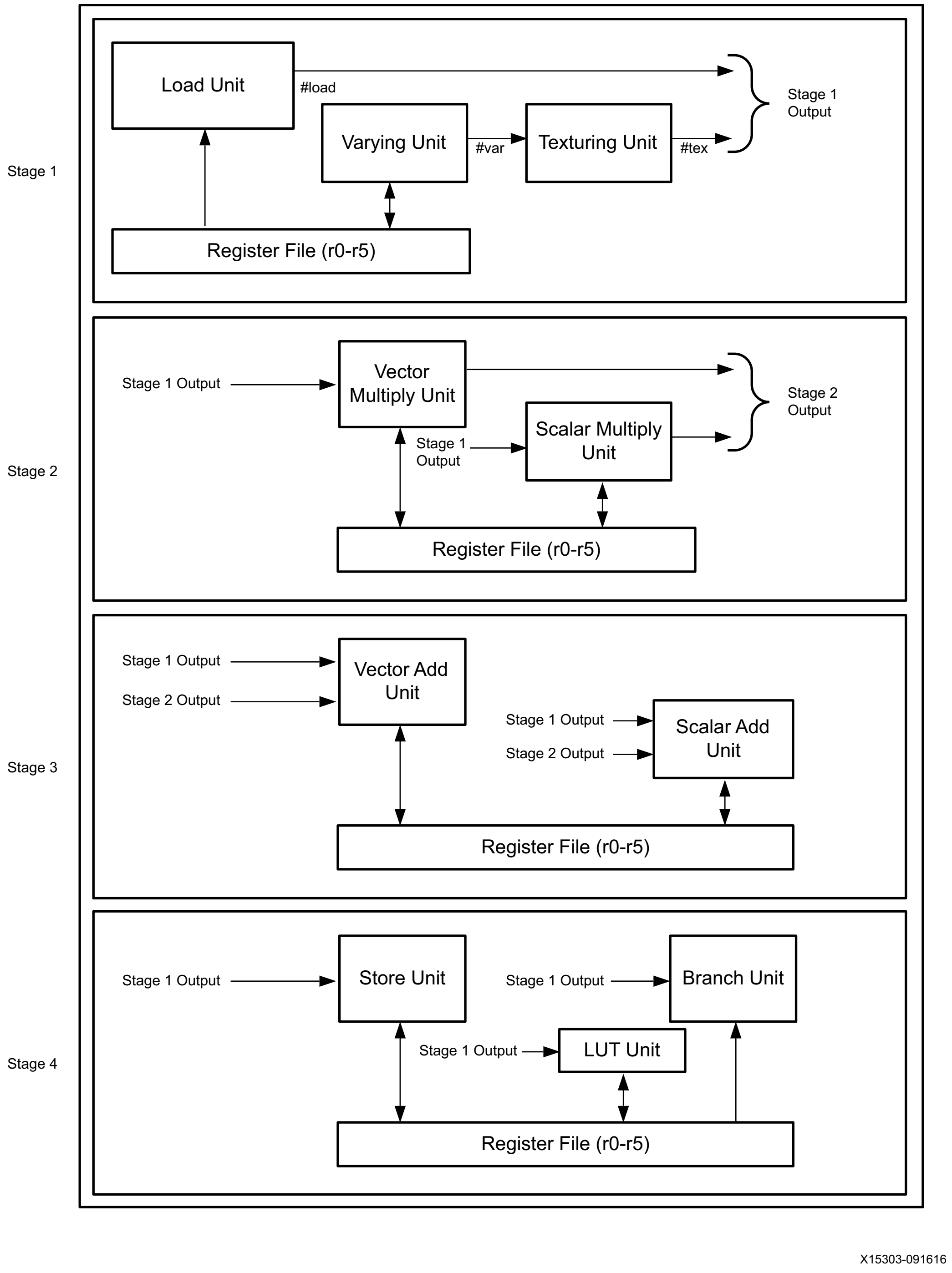


Figure 5-5: **GPU Fragment Shader Unit**

Graphics Processing Unit Level 2 Cache Controller

The advanced peripheral bus (APB) slave controls the Level 2 cache controller through various commands.

- APB slave provides an interface to enable bus masters to control the Level 2 cache. The APU MPCore or other AXI bus master can write into the cache controller. The graphics driver seamlessly supports these connections.
- Arbiter accepts memory access requests into a circulating loop. They circulate in the loop until the access router determines that they can be removed.
- Tag accessor performs a cache lookup to determine if data is in the cache.
- Access router for each read or write request matching the AXI ID and the timestamp of the current request against all other requests in the loop.
- Replay buffer that handles all request collisions of data.
- Cache tags unit holds a pipelined SRAM for the cache tags.
- Cache line fetcher draws the external data from the AXI master interface.
- Cache SRAM is the actual data store of the cache.

The Level 2 cache controller performs the tasks shown in Table 5-3.

Table 5-3: **Level 2 Cache Controller Tasks**

Controller Task	Description
Looping	Memory requests enter by the arbiter and keep circulating in a loop until the access router determines they can be taken out of the loop.
Cache tag lookup	For each request, the tag accessor performs a cache lookup to see if data is in the cache. The cache tags are kept in a single SRAM within this sub-unit. It is fully pipelined and permits one tag operation to be performed per clock cycle. A tag operation consists of a lookup, tag-write, and line-invalidate.
Access routing	The access router passes data for each request to the read buffer, write buffer or replay buffer.
Handling data collisions	The replay buffer handles all request collisions of data. These request collisions, also called bad-hits, continue to loop around the system until the access router determines when they can be taken out of the loop.
Storing cache tags	The cache tags unit holds a pipelined SRAM for the cache tags. The cache tags are stored in a single SRAM within this sub-unit.

The Level 2 cache controller is a configurable cache controller able to manage 16 KB to 16 MB of Level 2 cache RAM. It is a four-way set-associative cache controller with a pseudo-least recently used (LRU) replacement algorithm that yields great bandwidth savings in graphics operations. It supports high throughput, out of order data transactions within the AXI protocol limits, and up to 32 outstanding transactions and 64-byte bursts.

Because the Level 2 is a specialized cache controller for use with Mali GPUs, the following limitations apply.

- Write data is not cached because it does not increase memory bandwidth or performance savings.
- Writes to any cached location cause the relevant cache line to be cleared.
- Only incremental bursts are supported, there is no support for unaligned, fixed, and wrapping bursts.
- AXI slave ports are fixed at 64-bits wide. The AXI master port is configurable to be either 64 bits or 128 bits.

The main part of the cache controller is a loop of the four sub-modules.

- Arbiter
- Tag accessor
- Access router
- Replay buffer

Graphics Processing Unit Memory Management Unit

All memory accesses from the pixel processor and geometry processor use memory management units (MMUs) for access checking and translation. The GPU contains several MMUs to translate and restrict memory accesses that the pixel or geometry processors initiate. An MMU is configured by writing to control registers and uses in-memory page table structures as the basis for address translation.

The MMU divides memory into 4 KB pages, where each page can be individually configured. For each page the following parameters are specified.

- The physical memory address of the page. Known as address translation or virtual memory, this enables the processor to work using addresses that differ from the physical addresses in the memory system.
- The permitted types of accesses to that page. Each page can permit reads, writes, both, or none.

The MMU uses a two-level page table structure (Figure 5-6). The first level, the page directory consists of 1024 directory table entries (DTEs), each pointing to a page table. In the second level, the page table consists of 1024 page table entries (PTEs), each pointing to a page in memory.

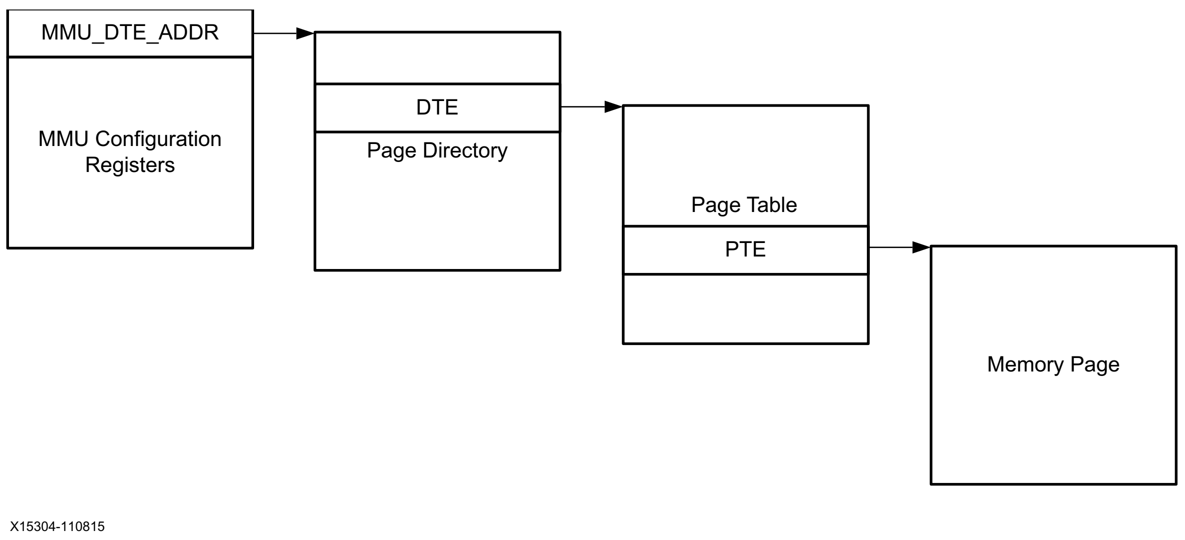


Figure 5-6: **Structure of the Two-level Page Table**

The MMU address bits are shown in Table 5-4.

Table 5-4: **MMU Address Bits**

31 ... 22	21 ... 12	11 ... 0
DTE Index	PTE Index	Page Offset

The MMU uses the following algorithm to translate an address.

1. Find the DTE at address given by MMU_DTE_ADDR + (4 x DTE index).
2. Find the PTE at address given by (page table address from DTE) + (4 x PTE index).
3. Calculate effective address as (page address from PTE) + (page offset).

Figure 5-7 shows various possible state transitions for the MMU.

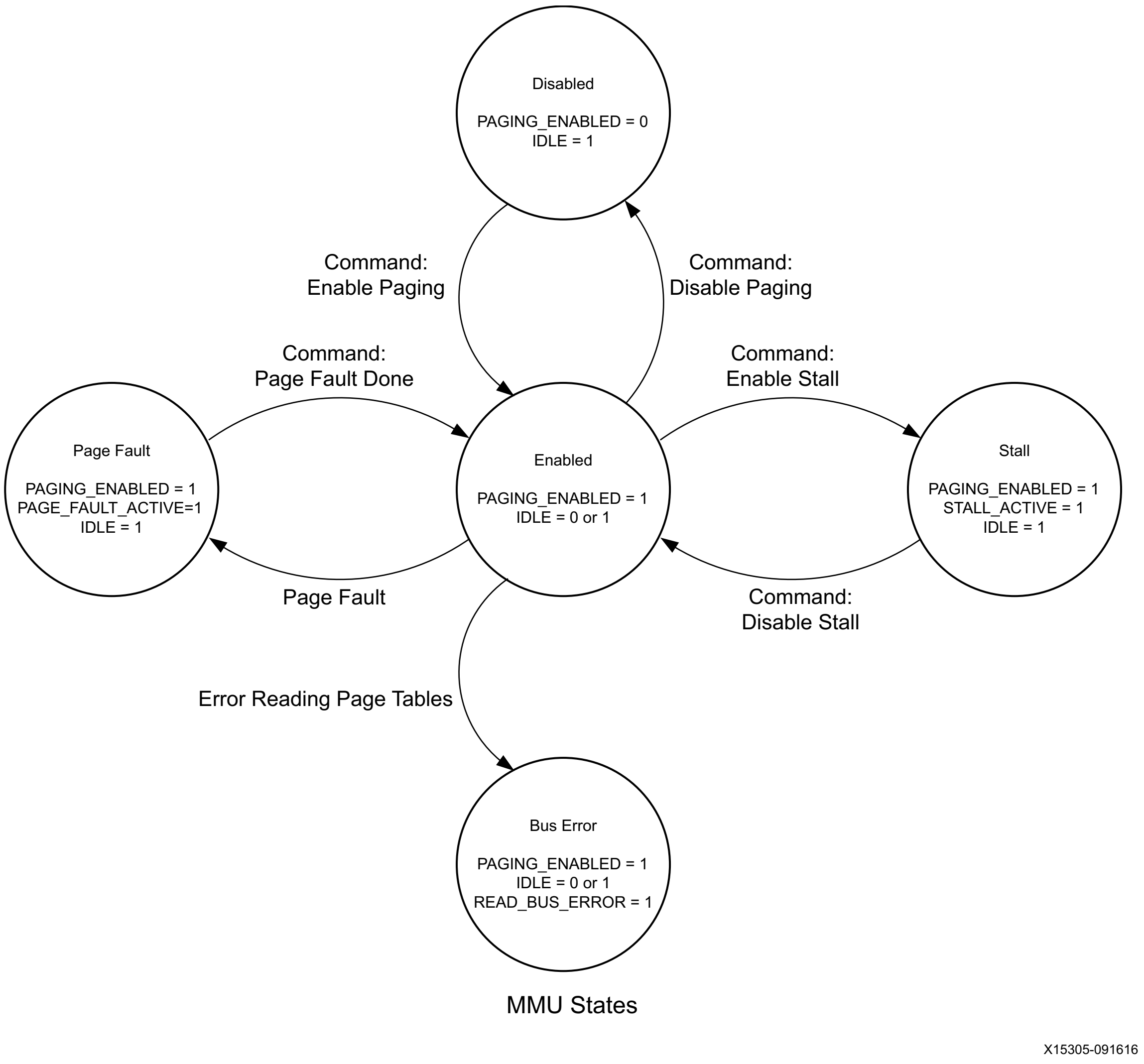


Figure 5-7: **GPU MMU State Diagram**

Graphics Processing Unit Programming Model

Power Management in GPU

The GPU acts as a slave with respect to the power management. Another processor runs the GPU device driver and responsible for managing the GPU’s overall power including the power down of the pixel processors.

All sub-blocks within the GPU (pixel processor, geometry processor, or the L2 controller) include an idle signal that is routed to the FPD_SLCR.GPU register containing the PP{0, 1} and GPU idle indicators. Before requesting the PMU to power down the GPU PP0 or PP1, the GPU device driver must check the FPD_SLCR.GPU [PPx_Idle] bits to ensure that the targeted pixel processor is idle.

The device driver then requests the PMU to power down the pixel processor by writing `1` to the [PP0] or [PP1] bit in the PMU_GLOBAL.PWR_STATE register. The pixel processor power state is indicated in the PMU_GLOBAL.PWR_STATE register.

Similarly, the device driver can initiate the power up of a GPU pixel processor by setting the bit associated with the target pixel processor in the Power_Up_Request register, which triggers the PMU to proceed with powering up the target GPU pixel processor. The request to release the reset on the GPU or its associated pixel processors must be explicitly requested by the device driver by setting the appropriate bits in the PMU Reset_Request register as explained in the PMU Reset section of Chapter 6, Platform Management Unit. Chapter 6 also has information on the Power_Down_Request and Power_Up_Request registers.

Programming the GPU

The Mali GPU can be configured using the openGLES 2.0 API. Before use, the GPU must be powered up using the PMU as described in the Operation section in Chapter 6, Platform Management Unit. Powering up uses a Xilinx driver (in the Linux environment). There are common libraries used for interacting with the driver. Those libraries are used by the OpenGLES implementation, which is then called by your specific application. Figure 5-8 describes the top level hierarchy/stack of libraries and driver for the GPU hardware. A list of documentation references follows the figure.

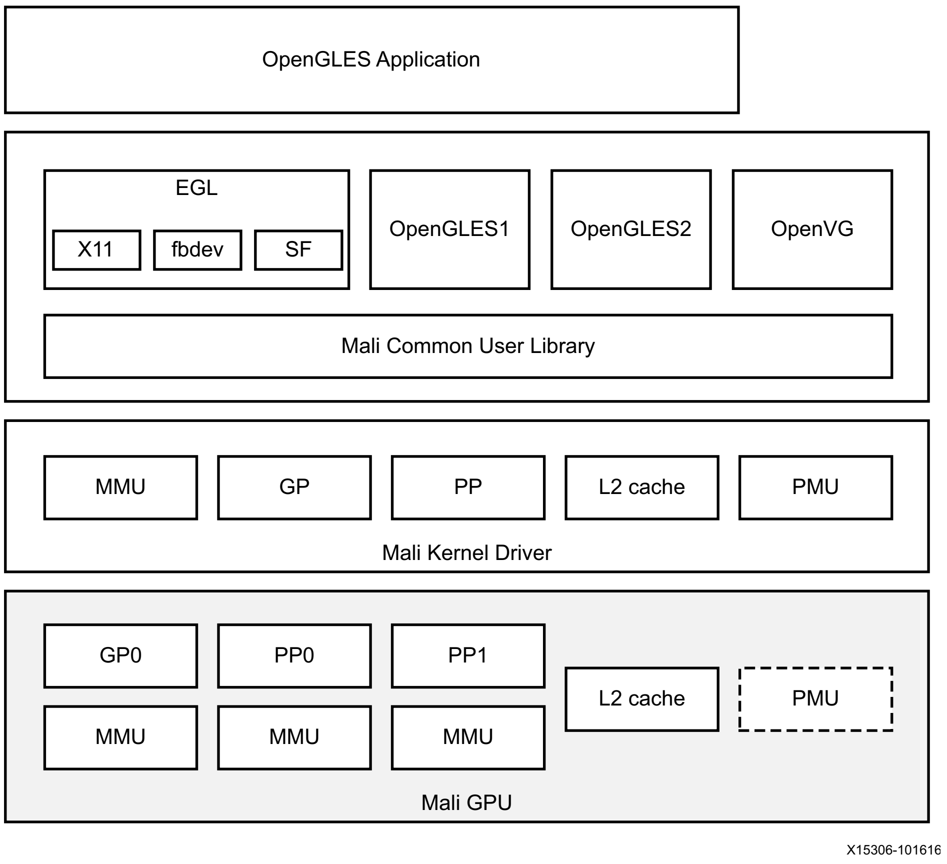


Figure 5-8: **GPU Software Stack**

The PMU section of the kernel driver calls the Xilinx PMU API instead of the Arm specified PMU controlling the API(s).

Some useful references.

- A description of the OpenGL API and how to use them is out of scope of this document.
- Details on an example and API(s) to program and use the GPU can be found at the Khronos site and the Arm Mali developer site (Arm Mali Developer's Guide and the OpenGLES 2.0 Specification).
- A simple OpenGLES 2.0 example to draw a triangle.

RECOMMENDED: *If the GPU accesses other LPD memories (such as OCM, TCM), a delayed response can occur when there is an outstanding access to other slow I/O peripherals (such as Quad SPI flash memory) from a different master. OCM and TCM memories are not recommended for GPU access.*

Note: See the *Arm Mali GPU Application Optimization Guide* [Ref 54] to optimize the application's memory performance and power utilization.

Graphics Processing Unit Register Overview

Table 5-5 is an overview of the GPU registers.

Table 5-5: **GPU Register Summary**

Register Type	Register Name	Description
Geometry Processor Control Registers	GP_CONTR_REG_VSCL_START_ADDR	GPU control register VSCL start address
	GP_CONTR_REG_VSCL_END_ADDR	GPU control register VSCL end address
	GP_CONTR_REG_PLBCL_START_ADDR	GPU control register PLBCL start address
	GP_CONTR_REG_PLBCL_END_ADDR	GPU control register PLBCL end address
	GP_CONTR_REG_PLB_ALLOC_START_ADDR	GPU control register PLB allocate start address
	GP_CONTR_REG_PLB_ALLOC_END_ADDR	GPU control register PLB allocate end address
	GP_CONTR_REG_CMD	GPU control register command
	GP_CONTR_REG_INT_RAWSTAT	GPU control register interrupt raw interrupt status
	GP_CONTR_REG_INT_CLEAR	GPU control register interrupt clear
	GP_CONTR_REG_INT_MASK	GPU control register interrupt mask
	GP_CONTR_REG_INT_STAT	GPU control register interrupt status
	GP_CONTR_REG_WRITE_BOUND_LOW	GPU control register write boundary Low
	GP_CONTR_REG_WRITE_BOUND_HIGH	GPU control register write boundary High
	GP_CONTR_REG_PERF_CNT_0_ENABLE	GPU control register performance counter 0 enable
	GP_CONTR_REG_PERF_CNT_1_ENABLE	GPU control register performance counter 1 enable
	GP_CONTR_REG_PERF_CNT_0_SRC	GPU control register performance counter 0 source
	GP_CONTR_REG_PERF_CNT_1_SRC	GPU control register performance counter 1 source
	GP_CONTR_REG_PERF_CNT_0_VAL	GPU control register performance counter 0 value
	GP_CONTR_REG_PERF_CNT_1_VAL	GPU control register performance counter 1 value
	GP_CONTR_REG_PERF_CNT_0_LIMIT	GPU control register performance counter 0 limit
	GP_CONTR_REG_PERF_CNT_1_LIMIT	GPU control register performance counter 1 limit
	GP_CONTR_REG_STATUS	GPU control register status
	GP_CONTR_REG_VERSION	GPU control register version
	GP_CONTR_REG_VSCL_INITIAL_ADDR	GPU control register VSCL initial address
	GP_CONTR_REG_PLBCL_INITIAL_ADDR	GPU control register PLBCL initial address
	GP_CONTR_REG_WRITE_BOUNDARY_ERROR_ADDR	GPU control register write error address
	GP_CONTR_REG_AXI_BUS_ERROR_STAT	GPU control AXI bus error status
	GP_CONTR_REG_WATCHDOG_DISABLE	GPU control register watchdog disable
	GP_CONTR_REG_WATCHDOG_TIMEOUT	GPU control register watchdog timeout

Table 5-5: **GPU Register Summary *(Cont'd)***

Register Type	Register Name	Description
Control Register	VERSION	Version register
	SIZE	Size register
	STATUS	Status register
	COMMAND	Command register
	CLEAR_PAGE	Clear page register
	MAX_READS	Maximum reads register
	ENABLE	Enable register
Performance Counter Register	PERFCNT_SRC0	Performance counter 0 source register
	PERFCNT_VAL0	Performance counter 0 value register
	PERFCNT_SRC1	Performance counter 1 source register
	PERFCNT_VAL1	Performance counter 1 value register
Geometry Processor MMU Control Register	GP_MMU_DTE_ADDR	MMU current page table address register
	GP_MMU_STATUS	MMU status register
	GP_MMU_COMMAND	MMU command register
	GP_MMU_PAGE_FAULT_ADDR	MMU logical address
	GP_MMU_ZAP_ONE_LINE	MMU zap-cache line register
	GP_MMU_INT_RAWSTAT	MMU raw interrupt status register
	GP_MMU_INT_CLEAR	MMU interrupt clear register
	GP_MMU_INT_MASK	MMU interrupt mask register
	GP_MMU_INT_STATUS	MMU interrupt status register
Pixel Processor MMU Control Register [x = 0, 1]	PPx_MMU_DTE_ADDR	MMU current page table address register
	PPx_MMU_STATUS	MMU status register
	PPx_MMU_COMMAND	MMU command register
	PPx_MMU_PAGE_FAULT_ADDR	MMU logical address
	PPx_MMU_ZAP_ONE_LINE	MMU zap-cache line register
	PPx_MMU_INT_RAWSTAT	MMU raw interrupt status register
	PPx_MMU_INT_CLEAR	MMU interrupt clear register
	PPx_MMU_INT_MASK	MMU interrupt mask register
	PPx_MMU_INT_STATUS	MMU interrupt status register

Table 5-5: **GPU Register Summary *(Cont'd)***

Register Type	Register Name	Description
Pixel Processor Render And Tile Buffer Control Register [x = 0, 1]	PPx_REND_LIST_ADDR	Renderer list address register
	PPx_REND_RSW_BASE	Renderer state word base address register
	PPx_REND_VERTEX_BASE	Renderer vertex base register
	PPx_FEATURE_ENABLE	Feature enable register
	PPx_Z_CLEAR_VALUE	Z clear value register
	PPx_STENCIL_CLEAR_VALUE	Stencil clear value register
	PPx_ABGR_CLEAR_VALUE_0	Alpha-blue-green-red (ABGR) clear value 0 register
	PPx_ABGR_CLEAR_VALUE_1	ABGR clear value 1 register
	PPx_ABGR_CLEAR_VALUE_2	ABGR clear value 2 register
	PPx_ABGR_CLEAR_VALUE_3	ABGR clear value 3 register
	PPx_BOUNDING_BOX_LEFT_RIGHT	Bounding box left right register
	PPx_BOUNDING_BOX_BOTTOM	Bounding box bottom register
	PPx_FS_STACK_ADDR	Fault status (FS) stack address register
	PPx_FS_STACK_SIZE_AND_INIT_VAL	Fault status (FS) stack size and initial value register
	PPx_ORIGIN_OFFSET_X	Origin offset X register
	PPx_ORIGIN_OFFSET_Y	Origin offset Y register
	PPx_SUBPIXEL_SPECIFIER	Sub-pixel specifier register
	PPx_TIEBREAK_MODE	Tie-break mode register
	PPx_PLIST_CONFIG	Polygon list format register
	PPx_SCALING_CONFIG	Scaling register
	PPx_TILEBUFFER_BITS	Tile-buffer configuration register
Write-Back Buffer Control Register [x = 0, 1] [y = 0, 1, 2]	PPx_WBy_SOURCE_SELECT	Write-back y source select register
	PPx_WBy_TARGET_ADDR	Write-back y target address register
	PPx_WBy_TARGET_PIXEL_FORMAT	Write-back y target pixel format register
	PPx_WBy_TARGET_AA_FORMAT	Write-back y target anti-aliasing format register
	PPx_WBy_TARGET_LAYOUT	Write-back y target layout
	PPx_WBy_TARGET_SCANLINE_LENGTH	Write-back y target scan-line length
	PPx_WBy_TARGET_FLAGS	Write-back y target flags register
	PPx_WBy_MRT_ENABLE	Write-back y multiple render target (MRT) enable register
	PPx_WBy_MRT_OFFSET	Write-back y MRT offset register
	PPx_WBy_GLOBAL_TEST_ENABLE	Write-back y global test enable register
	PPx_WBy_GLOBAL_TEST_REF_VALUE	Write-back y global test reference value register
	PPx_WBy_GLOBAL_TEST_CMP_FUNC	Write-back y global test compare function register

Table 5-5: **GPU Register Summary *(Cont'd)***

Register Type	Register Name	Description
Pixel Processor Misc Control Register [x = 0, 1]	PPx_VERSION	Version register
	PPx_CURRENT_REND_LIST_ADDR	Current renderer list address register
	PPx_STATUS	Pixel processor status register
	PPx_CTRL_MGMT	Control management register
	PPx_LAST_TILE_POS_START	Last tile where processing started register
	PPx_LAST_TILE_POS_END	Last tile where processing completed register
	PPx_INT_RAWSTAT	Interrupt raw status register
	PPx_INT_CLEAR	Interrupt clear register
	PPx_INT_MASK	Interrupt mask register
	PPx_INT_STATUS	Interrupt status register
	PPx_WRITE_BOUNDARY_ENABLE	Write boundary enable register
	PPx_WRITE_BOUNDARY_LOW	Write boundary Low register
	PPx_WRITE_BOUNDARY_HIGH	Write boundary High register
	PPx_WRITE_BOUNDARY_ADDRESS	Write boundary address register
	PPx_BUS_ERROR_STATUS	Bus error status register
	PPx_WATCHDOG_DISABLE	Watchdog disable register
	PPx_WATCHDOG_TIMEOUT	Watchdog time-out register
Pixel Processor's Performance Counter Registers	PPx_PERF_CNT_0_ENABLE	Performance counter 0 enable register
	PPx_PERF_CNT_0_SRC	Performance counter 0 system reset controller (SRC) register
	PPx_PERF_CNT_0_LIMIT	Performance counter 0 limit register
	PPx_PERF_CNT_0_VALUE	Performance counter 0 value register
	PPx_PERF_CNT_1_ENABLE	Performance counter 1 enable register
	PPx_PERF_CNT_1_SRC	Performance counter 1 system reset controller register
	PPx_PERF_CNT_1_LIMIT	Performance counter 1 limit register
	PPx_PERF_CNT_1_VALUE	Performance counter 1 value register
	PPx_PERFMON_CONTR	Performance monitor control register
	PPx_PERFMON_BASE	Performance monitor base address register

Chapter 6

Platform Management Unit

Introduction

The Zynq® UltraScale+™ MPSoC includes a dedicated user-programmable processor, the platform measurement unit (PMU) processor for power, error management, and execution of an optional software test library (STL) for functional safety applications.

The PMU performs the following set of tasks.

- Initialization of the system prior to boot.
- Power management.
- Software test library execution (optional).
- System error handling.

The configuration and security unit (CSU) monitors system temperature sensors.

Power Modes

There are three modes of power management operation at the PS level: battery-powered mode, low-power operation mode, and full-power operation mode.

To comply with the power domain requirements, there are separate power rails to supply the power for each domain. Figure 6-1 shows the features within the PS over the power rails.

Battery Powered Mode

To maintain critical information over the time during power off, the device provides the battery power mode. The following blocks are contained in the battery-powered domain:

- Battery-backed RAM (BBRAM) holds the key for secure configuration.
- Real-time clock (RTC) with crystal oscillator.

Low-Power Operation Mode

In the low-power operation mode, hardware blocks on the low power rail are powered up in the PS block (PMU, RPU, CSU, and the IOP). The low-power mode includes all peripherals except the SATA and display port blocks. Table 6-1 shows the IP enabled in low-power mode.

Table 6-1: **Minimum and Typical Configurations for the Low-Power Mode**

System Elements	Typical Minimum Configuration	Typical Configuration Full Optimization	Comments
Cortex-R5F	One core @ 50 MHz	Two cores @maximum data sheet frequency	Clock is gated to the unused core.
TCM configuration OCM configuration	Powered down 128KB	64 KB instruction and 64 KB data 256 KB	Power is gated off to the unused TCM banks. Power is gated off to the unused banks
Device security	Without AES	All, including AES	
Peripheral	One set of UART, I^2C, and Ethernet	All peripherals in LPS and one USB 2.0	USB can independently be powered down.
PLLs	One PLL	Two PLLs	PLLs that are not used are in the powered-down state.
SYSMON	Included	Included	Power is reduced as there are fewer supplies to be sampled.
RTC and BBRAM	Included	Included	Switched to the VCC_PSAUX rail.
PMU SOC debug	Included Standby	Included Standby	SOC debug is mostly on the FP rail. The LP section is not used.
eFuse Components outside LPD	Included Powered down	Included Powered down	
PL	Powered down	Powered down	

Full-Power Operation Mode

All domains are powered in the full-power mode, so the LPD is typically powered whenever FPD is powered. Like the low-power mode, power dissipation depends on the components that are running and their frequencies.

Note: If the FPD is needed at any point, it must be powered during the initial boot. This does not apply if the FPD is never used.

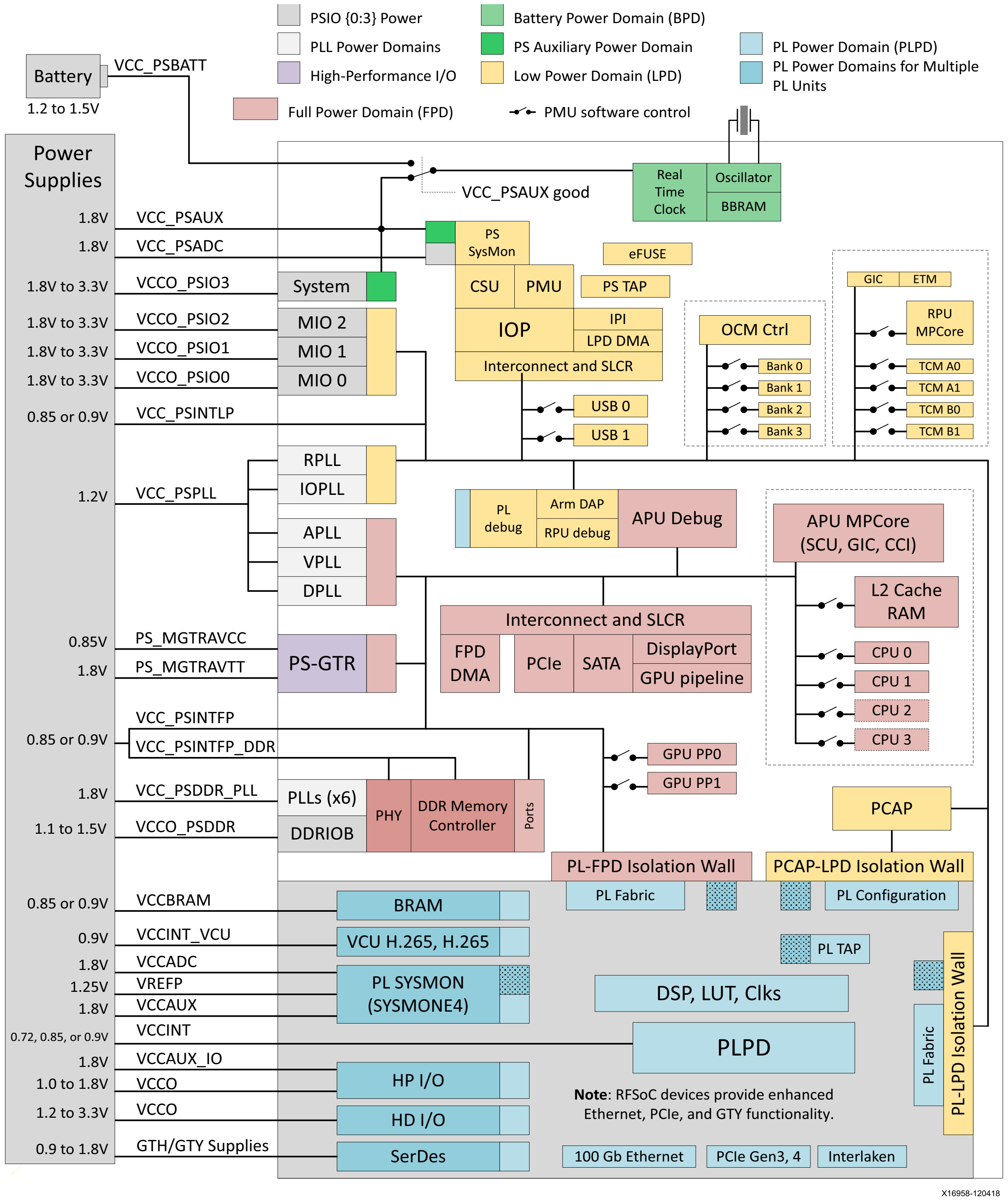


Figure 6-1: **Power Domains and Islands**

PMU System-level View

The PMU block is located within the low-power domain. Figure 6-2 shows the block diagram of the PMU. It includes the following subcomponents:

- Dedicated, fault-tolerant triple-redundant processor.
- ROM to hold PMU ROM code that includes the PMU startup sequence, routines to handle power-up or down requests, and interrupts.
- 128 KB RAM with ECC used for code and data.
- PMU local registers accessible only by the PMU.
- PMU global registers accessible by the PMU processor and also by other bus masters within the system. These include all power, isolation, and reset request registers. It also includes error capture registers and the system power state registers.
- 32-bit AXI slave interface to allow masters outside the PMU to access the PMU RAM and the global register file.
- PMU interrupt controller manages the 23 interrupts to the PMU. Four are from the inter-processor interconnect (IPI).
- GPI and GPO registers interface to the PMU, MIO, PL, and other resources within the PS for signaling to and from the PMU.
    - Six outputs and six inputs.
    - 32 GPO outputs to the PL from the PMU and 32 GPI inputs from the PL to the PMU.
    - 47 system errors to the PMU.
    - CSU error code.
    - 32 memory built-in self test (MBIST) status signals and 32 MBIST completion signals.
    - Three direct reset control signals.
    - Four AIB status signals and four AIB control signals.
    - 11 logic clear status signals.
    - DDR retention control.
    - Three programmable settings to the CSU for the PL.
- PMU MDM controller accessible using the PS TAP controller via the PSJTAG interface.

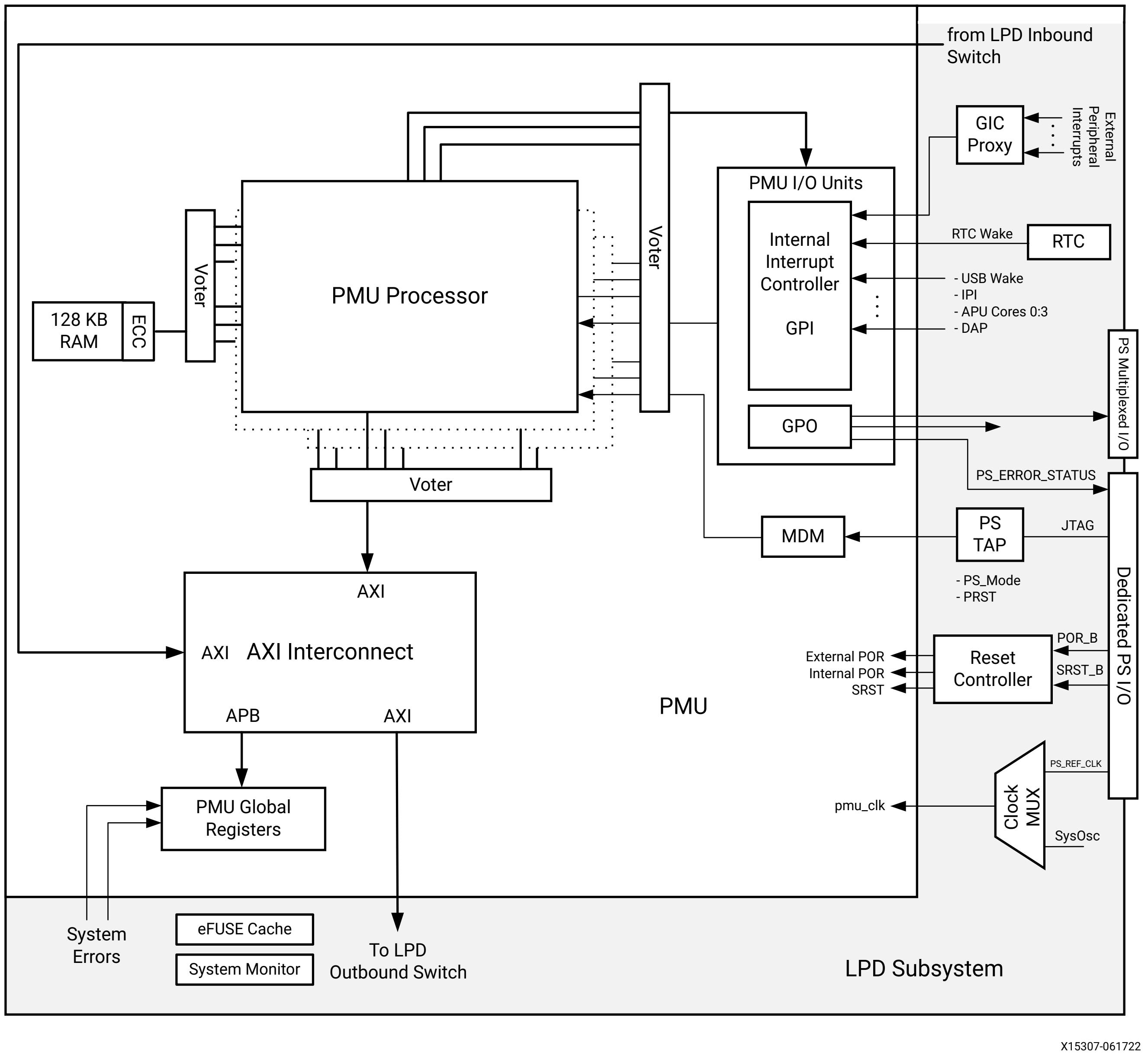


Figure 6-2: **PMU System Diagram**

Functional Description

The functionality within the PMU is outlined in this section.

- Performs the sequencing of events after POR and before CSU reset is released. These functions include the following.
  - Check the power-supply levels using the System Monitor for proper operation of the CSU and the rest of the LP domain.
  - Initialize the PLLs for the default configuration and their potential bypass.
  - Trigger and sequence the necessary scan and MBIST.
  - Capture and signal errors during this stage. Error ID can be read through JTAG.
  - Release reset to the CSU.
- Acts as a delegate to the application and real-time processors during their sleep state and initiates their power-up and restart after their wake-up request.
- Maintain the system-power state at all times.
- Handles the sequence of low-level events required for power-up, power-down, reset, memory built-in self repair (MBISR), MBIST, and scan zeroization of different blocks.
- Manages the system during the sleep mode and wake-up the system based on various triggering mechanisms.
- Includes PS-level error capture and propagation logic.

PMU Processor

The PMU processor is a triple-redundant processor without caches. The processing system provides fault tolerance by applying redundancy on the PMU and error correction (ECC) on the RAM interface. The triple redundancy and ECC corrects single errors and generates an error on multiple errors that cannot be corrected. When an error occurs with one of the PMU processors, it might not always be possible for the processor in error to properly continue operation. Thus, at some point, the PMU might require a reset for proper TMR operation.

There is a provision to allow more complex power protocol management programs to be implemented as firmware or application programs in the PMU RAM.

Note: PMU processor debug module is disabled by default on ES2 and higher versions.

Table 6-2 lists the implementation features for the PMU processor.

Table 6-2: **MicroBlaze Implementation Features**

Feature	Implementation
Pipeline	5-stage.
Interconnect standard	AXI
Endianness	Little endian.
Program counter width	32
Support for load/store exclusive	Enabled.
Fault tolerance	Enabled.
Hardware multiplier/divider/barrel shifter	Disabled/disabled/enabled.
Debug	Enabled. One of each type of break-point.
Fast interrupt	Disabled.

PMU Processor Interfaces

The PMU provides input/output signals that are grouped functionally into the following interfaces.

- 32-bit AXI master interface to the low-power domain (LPD) interconnect that allows the PMU to access other PS resources including the SLCR registers and the IPI block.
- 32-bit AXI slave interface from the LPD inbound switch to allow accesses to the PMU global registers and the PMU RAM by external processors.
- PMU clock and reset signals.
- Power control interface to all islands within the PS.
    - L2, OCM, and TCM RAMs.
    - APU_Cores [3:0].
    - Dual-core Cortex-R5F® real-time processor.
    - USB0 and USB1.
    - GPU pixel-processor (PP) PP0 and PP1.
    - Full-power and PL domain crossing bridges.

Wake interface from GPIO, RTC, APU GIC, RPU GIC, and USBs.

- Interrupt interface.
- Device reset control interface.
- Memory BIST and BISR control interface.
- Other miscellaneous interfaces including the power-supply monitor interface. Table 6-3 lists the PMU general purpose MIO pins.
- Error capture and propagation interfaces. Table 6-4 lists the error capture and propagation signals.

Table 6-3: **PMU General Purpose MIO Pins**

Register Bit Fields	Pins	Size	Direction	Clock	Clamp Value	Description
GPI1[15:10]	MIO [31:26]	6	Input	Async	`6'b0`	Inputs for external events that are available to the PMU using six MIO pins. The GPI1 register bits are listed in Table 6-6. These signals are defined by FSBL, SDK, development boards, or users.
GPO1[5:0]	MIO [37:32]	6	Output	pmu_clk		Output signals to control external power supplies and other board hardware using MIO pins. See Table 6-9 for pin assignments. GPO1[0]: used by the PMU ROM code for the FPD's VCC_PSINTFP. GPO1[1]: used by the PMU ROM code for the PL's VCCINT. GPO1[2:5]: user defined (including Xilinx reference boards and customer designs). Not used by the PMU ROM code.

Table 6-4: **Error Interface Signals To and From the PL**

Signal Name	Size	Direction	Clock	Clamp Value	Description
pmu_pl_err	4	Input	Async	`4'b0`	Generic PL errors communicated to PS.
pmu_error_to_pl	47	Output	pmu_clk		PS error communicated to the PL and JTAG.

PMU Clocking

The PMU operates on the SysOsc clock (180 MHz ± 15%) that is supplied from the internal ring-oscillator (IRO) located within the system monitor (PS SYSMON) block. The clock is gated until the POR block detects that the V_{CC_PSAUX} supply has ramped up.

SysOsc starts to oscillate as soon as the voltage is high enough for the block to function. The reset of the PMU processor is synchronous and requires a clock edge for it to take place, POR_B input must be asserted until the voltage has ramped up. This guarantees that the PMU processor GPOs, which control many hardware logic blocks within the PS, are initialized when the device is powered up.

PMU Reset

The PMU block uses both power-on reset (POR) and the system reset (SRST) inputs that are controlled by the reset block. POR clears the state of the PMU completely. All islands and power domains are powered up and all the isolations are disabled. After a POR, the PMU executes both scan and BIST clear functions on the LP and FP domains. However, the SRST will only reset the PMU processor subsystem, the PMU interconnect, and a subset of local and global registers, leaving most local and global registers in the states they were prior to the reset. When the SRST triggers the reboot of the PMU, the power state is not cleared and the power state of the PS is preserved. However, after a power-on reset, the power state is cleared by specifically clearing all RAMs and flip-flops.

PMU RAM

Much of the PMU functionality is provided by software executed by the PMU processor. The ROM memory contains instructions that provide default functionality. To extend or replace these features, or to provide new features, software can be downloaded into the PMU processor's 128 KB RAM. The PMU includes a 128 KB RAM with 32-bit ECC that is used to hold data and code. The PMU RAM is accessible both by the PMU processor and the external masters through the PMU AXI slave interface.

IMPORTANT: *Accesses by the external masters should be 32-bit wide and word-aligned.*

The PMU RAM allows only word writes, words are 4 bytes. It does not allow byte writes. If less than 4 bytes have to be written, then the 4 bytes must be read first, modified, and the entire 4 bytes must be written back.

For an external master to access the PMU RAM through the APB interface, the PMU processor must be in sleep mode. A PMU RAM access from an external master while the PMU processor is not asleep can hang the system. If the PMU processor is not put in sleep mode, it performs an instruction fetch or load/store on every clock cycle, which means that the APB never gets to access the RAM. In this case, starvation of the APB interface occurs.

The following is the order of priority to access the PMU RAM.

1. PMU processor data load/store.
2. PMU processor instruction fetch.
3. External access.

PMU ROM

PMU includes a ROM that holds the boot code for the PMU, its interrupt vectors, and the service routines that the PMU can execute (upon a request). The PMU ROM is responsible for various functions within the PMU. The following is the list of the tasks that are executed by the ROM code.

- Pre-boot tasks
  - Clean PMU RAM
  - Enable the System Monitor and check LP domain supply.
  - Configure PLLs with initial settings.
  - Trigger and sequence the necessary scan and BIST clear of PS.
  - Release reset to CSU.
- Post-boot tasks
  - Power-up and power-down domains within the PS.
  - Enable and control built-in self-repair (BISR).
  - Reset blocks when requested or as a part of the master power-ups.
- Execute firmware code upon request.

MBIST Functionality

ROM code execution initiates MBIST clear on the entire LP domain minus the PMU or on the entire FP domain. When a memory is tested or cleared using the MBIST, the rest of the system can be functioning. For most of the blocks, RAM is accessed by the MBIST and it keeps the block RAM in the reset state when the RAM is accessed by the MBIST engine. For a few blocks, such as APU core processors, RAM is accessed by the MBIST through the core functional paths that can be interfered if the block is in reset. In such cases, Arm requires a small subset of inputs to the core to be tied off to specific values during the MBIST execution.

Setting a particular bit in the MBIST_RST, MBIST_PG_EN, and MBIST_SETUP registers starts the MBIST process on that particular block. The MBIST_DONE bit is set to indicate that the process is finished. MBIST_GOOD provides the status of the process by setting either 0 (fail) or 1 (success).

There are five control and status registers:

- MBIST_RST rw
- MBIST_PG_EN rw
- MBIST_SETPU rw
- MBIST_DONE ro
- MBIST_GOOD ro

For the RAMs in:

- APU, RPU cores
- CANx, GEMx, USBx,
- GPU, PCIe, SIOU
- PS-PL AXI Interface RAMs

The MBIST units are listed by bit field in the *Zynq UltraScale+ MPSoC Register Reference* (UG1087) [Ref 4].

Scan Clear Functionality

Zeroization is a process in which zeros are shifted through all of the storage elements and then verified that the shift occurred correctly. This is achieved using MBIST and scan clear functionality. The scan clear engines can only be controlled by the PMU and CSU processors through their direct interfaces to the engines. Other processors can request the PMU through its SCAN_CLR_REQ register to start any specific scan clear engines. When a scan clear engine is started, the completion status signal from the engine transitions from 1 to 0. This signal, which is routed directly to a PMU LOGCLR_ACK register, communicates the completion status of the engine to the PMU. When a scan clear engine finishes its operation, its completion status bit toggles from 0 to 1 generating an interrupt to the PMU. The pass/fail status of the clearing operation can be checked by the bits in the PMU LOGCLR_STATUS global register that are directly driven by the pass/fail status of the engine.

The CSU only starts scan clear engines under a security lock-down scenario and there is no functional requirement for the CSU to check the pass/fail status, or the completion status, of the clearing operation.

Every power island and every power domain has a scan clear engine. The PMU and CSU blocks have separate scan clear engines even though they are not power islands. The PMU scan clear is triggered only on power-on reset and the CSU scan clear can only be triggered by the PMU.

IMPORTANT: *The scan clear has to operate on the entire power island. In this case, the power island needs to be isolated before the block is put in the scan mode to start the scan clear functionality.*

To ensure running the scan clear on the LP domain, the full LPD (minus the PMU) is in reset, the reset logic must follow these guidelines:

1. Keep reset registers off the LPD scan chain.
2. Leverage the explicit reset input to clear state in registers that have this feature (this is recommended, but not required). The explicit reset can be asserted by the scan clear request output from the PMU (scan_clear_trigger_lpd output for the LPD domain) to force the reset to stay asserted by OR'ing it with the reset. The use of explicit resets for clearing instead of using scan on these registers requires them to be applied on chains that are included in the scan test rather than in the scan clear. However, this makes the scan architecture more complex.
3. The PMU local and global registers implement self-clearing through reset and are excluded from the scan clear. This is done to prevent an unnecessary power cycle of the islands during the scan clear of the PMU. The PMU is required to be cleared only during a POR or after a security shutdown. In either case, the flip-flops on the local and global registers are excluded from clearing functions. If for any reason this is not acceptable for the security lock-down, the reset to the flip-flops with the self-clearing feature that are not cleared through scan has to be asserted after the scan clear function on the rest of the flops is completed. This guarantees that the self-clearing of the PS is not affected by a potential IR drop due to the power up of the blocks that were previously powered down.

Note: User functions that need FPD SC must power MGTRAVCC even if not using the GT.

PMU Interconnect

PMU includes a 2 × 3 interconnect which supports two AXI masters, two APB slaves, and one AXI slave. One of the masters is the 32-bit AXI master from the triple-redundant processor and the other is the low-power domain main interconnect. This AXI master is a port on its register switch allowing any master in the system to access the PMU slaves.

The two APB slaves are the PMU RAM and PMU global register file. The AXI slave is on the port routed to the LPD switch and only allows the accesses that were originated by the PMU processor to be routed to the PS slaves outside the PMU.

The PMU processor AXI master can generate a coherent transaction by setting the coherent bit in the PMU global control register. The PMU AXI master (from the LPD interconnect) always generates transactions with AWCACHE and ARCACHE equal to `4'b0001` regardless of the coherency bit. This implies that PMU requests are treated as device transactions that can be buffered.

The PMU interconnect implements TrustZone security. All accesses that are generated by the PMU are secure and only secure accesses are allowed to be routed to the PMU. The PMU interconnect will generate an error on any non-secure access to the PMU.

PMU I/O Registers

The PMU I/O registers include all the registers associated with the interrupts, GPI/GPO, and the programmable interval timers (PITs). The PMU_IOMODULE registers control the interrupt controller, GPI{0:3}, GPO{0-3}, and PIT0-PIT3. The PMU_GLOBAL registers enable the system processors to control interrupts and trigger PMU service requests.The PMU processor memory map is shown in Table 6-5.

Table 6-5: **PMU I/O Registers and Local Memory**

Memory Address	Size	Slave Interface	Accessible	AXI Interconnect
0xFFD0_0000	32 KB	PMU ROM	PMU only	Local bus
0xFFD4_0000	128 B	PMU_IOMODULE register set	PMU only	Local bus
0xFFD5_0000	1024 B	PMU_LMB_BRAM	PMU only	Local bus
0xFFD6_0000	128 B	PMU_LOCAL register set	PMU only	Local bus
0xFFD8_0000	1024 B	PMU_GLOBAL register set	System via XPPU	System bus
0xFFDC_0000	128 KB	PMU RAM memory	System via XPPU	System bus

PMU Global Registers

The global register set includes registers that are used as a means of communication between the PMU and other blocks to synchronize activities regarding power/system management and reset.

The PMU global register set is mapped at address `FFD8 0000`—`FFDB FFFF`. The registers are summarized in Table 6-16. For a bit-level description, refer to the PMU_GLOBAL section in the *Zynq UltraScale+ MPSoC Register Reference* (UG1087) [Ref 4].

PMU GPIs and GPOs

The PMU processor includes four local (only accessible by the PMU processor) GPI banks and four GPO banks. GPI0 and GPO0 are reserved for the dedicated PMU processor subsystem features (see PMU Processor), while GPI3 and GPO3 are reserved for communication with the PL. GPI1, GPI2, GPO1, and GPO2 are used for communication between the PS hardware features and the PMU.

The PMU's general-purpose I/O features include miscellaneous wake, errors, and handshaking signals. The usage of the GPIs and GPOs can be summarized as follows with all signals being active-High unless otherwise specified.

- GPI0 is used internally by the PMU processor. GPI0[31:0] shows the value of the fault-tolerance status register.
- GPI1 monitors wake-up requests. Table 6-6 describes the various GPI1 bit(s).

Table 6-6: **GPI1 Bit Descriptions**

Bit(s)	Description
GPI1[3:0]	ACPU3-ACPU0 wake from APU GIC associated with ACPU3-ACPU0.
GPI1[5:4]	R5_1 and R5_0 wake from RPU GIC associated with R5_1 and R5_0.
GPI1[7:6]	USB1 and USB0 wake.
GPI1[8]	DAP full-power domain wake-up request.
GPI1[9]	DAP RPU wake-up request.
GPI1[15:10]	General purpose wake-up and event signals from MIO (see Table 6-3). MIO[26] -> GPI1[10] MIO[27] -> GPI1[11] ... MIO[31] -> GPI1[15]
GPI1[16]	Full-power domain wake directed by the GIC proxy.
GPI1[19:17]	Reserved.
GPI1[23:20]	APU debug power-up request for ACPU3-ACPU0 APU MPCore processors 0, 1, 2, 3.
GPI1[27:24]	Reserved.
GPI1[28]	Error interrupt to PMU from error register 1.
GPI1[29]	Error interrupt to PMU from error register 2.
GPI1[30]	AXI AIB access error. A powered-down block is accessed through AXI.
GPI1[31]	APB AIB access error. A powered-down block is accessed through APB.

- GPI2 monitors power control requests. Table 6-7 describes the various GPI2 bit(s).

Table 6-7: **GPI2 Bit Descriptions**

Bit(s)	Description
GPI2[3:0]	Power-down request from APU core {3:0}.
GPI2[5:4]	Power-down request from RPU core {1:0}.
GPI2[6]	Read the state of the pcfg_por_b input from PL, which signifies that PL is properly powered up.
GPI2[7]	Reserved.
GPI2[8]	Request to reset RPU core 0 by debug.
GPI2[9]	Request to reset RPU core 1 by debug.
GPI2[15:10]	Reserved.
GPI2[16]	Warm reset request for APU core 0.
GPI2[17]	Warm reset request for APU core 1.
GPI2[18]	Warm reset request for APU core 2.
GPI2[19]	Warm reset request for APU core 3.
GPI2[20]	Warm reset request for APU core 0 by debug logic.
GPI2[21]	Warm reset request for APU core 1 by debug logic.

Table 6-7: **GPI2 Bit Descriptions *(Cont'd)***

Bit(s)	Description
GPI2[22]	Warm reset request for APU core 2 by debug logic.
GPI2[23]	Warm reset request for APU core 3 by debug logic.
GPI2[28:24]	Reserved.
GPI2[31:29]	Power rail removal alarms. [31]: Asserts when $V_{CC\_PSINTFP}$ is removed. [30]: Asserts when $V_{CC\_PSINTLP}$ is removed. [29]: Asserts when V_{CC_PSAUX} is removed.

- GPI3 monitors the GPIs from the PL.
- GPO0 is dedicated to the PMU features. Table 6-8 describes the various GPO0 bit(s).

Table 6-8: **GPO0 Bit Descriptions**

Bit(s)	Description
GPO0[0]	Used during debug to remap the 64-byte interrupt base vectors region to the RAM starting address (`0xFFD0 0000`). 0 = base vectors in ROM (default). 1 = base vectors in RAM.
GPO0[2:1]	Set PIT0 prescaler. x0 = PIT0 is a 32-bit timer with no prescaler. 01 = External prescaler. 11 = PIT1 is prescaler to PIT0.
GPO0[4:3]	Set PIT1 prescaler. x0 = PIT1 is a 32-bit timer with no prescaler. x1 = External prescaler.
GPO0[6:5]	Set PIT2 prescaler. x0 = PIT2 is a 32-bit timer with no prescaler. 01 = External prescaler. 11 = PIT3 is prescaler to PIT2.
GPO0[7]	Set PIT3 prescaler. 0 = PIT3 is a 32-bit timer with no prescaler. 1 = External prescaler.
GPO0[8]	Used to suppress the comparison of the PMU processor trace bus to not detect a trace bus mis-compare during fault injection.
GPO0[9]	Controls if the PMU processor SLEEP instruction cause a processor hardware reset during recovery from lock-step mode due to voting mode comparison.
GPO0[10]	Makes it possible to clear the value of the fault tolerance status register.
GPO0[11]	Makes it possible to reset the fault tolerance state machine.
GPO0[12]	Controls if fault tolerance state machine reset of the PMU processor is generated or not.

Table 6-8: **GPO0 Bit Descriptions *(Cont'd)***

Bit(s)	Description
GPO0[15:13]	Used to inject failures in the triple-redundant PMU processor.
GPO0[23:16]	Used as magic word #2 to reduce the risk of accidental commands controlling TMR operation being issued.
GPO0[31:24]	Used as magic word #1 to reduce the risk of accidental commands controlling TMR operation being issued.

- GPO1 is dedicated to the MIO for signaling and power-supply management. Table 6-9 lists the GPO1 register bits.

Table 6-9: **GPO1 Bit Descriptions**

Bit(s)	Description
GPO1[5:0]	These bits can drive up to six MIO outputs, their usage is described in Table 6-3.
GPO1[31: 6]	Not implemented.

- GPO2 is dedicated to the PMU-generated requests and acknowledges. Table 6-10 describes the various GPO2 bit(s).

Table 6-10: **GPO2 Bit Descriptions**

Bit(s)	Description
GPO2[5:0]	Reserved.
GPO2[6]	Used to enable a subset of signals between PL and PS after the PMU has determined that the PL is properly powered up.
GPO2[7]	PS status output from PMU to a dedicated PS general purpose I/O pad.
GPO2[8]	Acknowledge to FP wake-up request from DAP.
GPO2[9]	Acknowledge to RPU wake-up request from DAP.
GPO2[31:10]	Not implemented.

- GPO3 is dedicated to the GPOs to the PL.

PMU Programmable Interval Timers

The PMU includes four 32-bit programmable interval timers (PITs). The clock source to these timers is the fixed system oscillator (SysOsc) to the PMU. These are general-purpose timers for use as delay counters or event scheduling. The pre-scaler for the PITs can be configured through GPO0. The following are the possible pre-scaler choices for each PIT.

- PIT0: No pre-scaler, use pre-scaler value from PIT1
- PIT1: No pre-scaler
- PIT2: No pre-scaler, correctable ECC error
- PIT3: No pre-scaler

The timers are only accessible from the PMU firmware. The PMU processor's I/O module driver provides an API for these resources.

PMU Interrupts

When the PMU processor receives an interrupt, it branches to the PMU ROM. The ROM code must check the pending interrupt register within the interrupt controller in the PMU I/O module and branch to the appropriate interrupt service routine in the ROM or RAM. The priority between the pending interrupts can be enforced by the PMU firmware, and if not present, the priority is managed by the ROM. Table 6-11 lists the PMU interrupts.

Table 6-11: **PMU Interrupts**

Bit in Interrupt Pending Register	External Interrupt	Description
31	Secure lock-down request	Interrupt from CSU to initiate a secure lock down.
30	Reserved	
29	Address error interrupt	Interrupt for address errors generated during accesses to PS SLCRs or PMU global registers.
28	Power-down request	Interrupt to signal a power-down request.
27	Power-up request	Interrupt to signal a power-up request.
26	Software reset request	Interrupt to signal a software-generated reset request.
25	Hardware block RST request	Interrupt for all hardware-generated block reset requests.
24	Isolate request	Interrupt to signal an isolation request.
23	ScanClear request	Interrupt to signal a scan clear request.
22-19	IPI3-IPI0	Interrupt associated with IPI slices 3-0 to PMU.
18	RTC alarm interrupt	Interrupt from RTC to signal the alarm.
17	RTC seconds interrupt	Interrupt from RTC triggered every second.
16	Correctable ECC error	Interrupt generated when an ECC error on the PMU RAM is corrected.
15	Reserved	
14	GPI3	Interrupt generated when any input on GPI3 changes from 0 to 1.
13	GPI2	Interrupt generated when any input on GPI2 changes from 0 to 1.
12	GPI1	Interrupt generated when any input on GPI1 changes from 0 to 1.
11	GPI0	Interrupt generated when any input on GPI0 changes from 0 to 1.
10-7	Reserved	

Table 6-11: **PMU Interrupts *(Cont'd)***

Bit in Interrupt Pending Register	External Interrupt	Description
6-3	PIT3-PIT0	Programmable interval timer interrupts.
2-0	Reserved	

MIO Pin Considerations

The processing system (PS) contains three banks of 26-bit general-purpose multiplexed I/O (MIO) used by different peripherals. All the three banks can support LVCMOS18, LVCMOS25, and LVCMOS33 standards. The I/O that is used in conjunction with the PMU includes the UTMI+ low pin interface (ULPI) for one USB, six GPIs for wake and signaling, and six GPOs for power supply control and signaling. The I/O pins for power management and wake up are accessible from the GPO1 and GPI1 registers, respectively.

Among the six GPOs, the PMU ROM code uses GPO1[0] on MIO[32] to control the FPD's VCC_PSINTFP power supply and GPO1[1] on MIO[33] to control the PL's VCCINT power supply. Both pins are active high (1 is power on and 0 is power off). The other four GPO[2:5] signals can drive outputs onto the MIO[34:37] pins.

The Xilinx development boards assign functionality for the GPO[2:5] signals, but they can be re-assigned and controlled by PMU user firmware because they are not used by the PMU ROM code. The GPO signals and MIO pins are listed in Table 6-3.

PMU Error Handling and Propagation Logic

The PMU is responsible for capturing, reporting, and taking an appropriate action with respect to each error. Each system error is identified in the PMU_GLOBAL error status registers. The PMU also includes the necessary registers, logic, and interfaces for handling this functionality.

The PMU provides a collection of error input signals that route all system-level hardware errors to capture them. These errors are recorded in the error status registers 1 and 2 within the PMU and are not cleared even during a system reset or an internal POR. A captured error can only be cleared if a `1` is explicitly written to each corresponding error status bit. All errors can generate an interrupt to the PMU. This interrupt can be masked per error. The propagation of all errors to error status registers can be disabled by using the bits in the error enable registers (ERROR_EN_1 and ERROR_EN_2) global registers in the PMU.

PMU also includes registers that can capture software-generated errors. The software errors refer to the errors that occur during the execution of PMU ROM, PMU firmware, and the CSU ROM.

Similar to the hardware errors, software errors are recorded in the PMU and are cleared only by an external POR or explicitly by writing a `1` to its corresponding error status register bit. All but the software errors are recorded by the PMU during its pre-boot execution can generate an interrupt to the PMU. Similar to the hardware errors, this interrupt can be masked per error.

For each of the errors that are processed by the error handling logic, you can decide what action should be taken when the error occurs. The possible scenarios would be one or a combination of the following choices.

- Assertion of the PS_ERROR_OUT signal on the device.
- Generation of an interrupt to the PMU processor (PMU_Int).
- Generation of a system reset (SRST).
- Generation of a power-on-reset (POR).

There are four mask registers associated with each of the ERROR_STATUS registers (ERROR_STATUS_1 and ERROR_STATUS_2). These mask registers can be used to enable either POR, SRST, PMU interrupt (if firmware is installed), or signal a PS_ERROR_OUT. To set the mask, write a `1` to the appropriate bit on the ERROR_INT_EN register (ERROR_INT_EN_1 or ERROR_INT_EN_2). To clear the mask, write a `1` to the appropriate bit on the ERROR_INT_DIS register (ERROR_INT_DIS_1 or ERROR_INT_DIS_2). When selecting the option to interrupt the PMU when a specific error occurs, there should be user firmware to process the error. Otherwise, a no-firmware error will occur. The signal states can be unmasked as desired. Table 6-12 lists all possible sources of error and the corresponding reset state of the ERROR_SIG_MASK_n mask registers for the PS_ERROR_OUT device pin signal. All of the other error mask registers are set = 1 (masked).

Table 6-12: **PMU Error Sources and Reset State Masks**

System Error	ERROR_SIG_MASK_n	ERROR_STATUS Register and [Bits]	JTAG Error Register	GIC IRQ	Description
Software Errors					
CSU BootROM detected error	U	_2 [26]	[0]	~	BootROM in CSU experienced an error during boot, including bitstream authentication failure.
PMU ROM code preboot errors	U	_2 [25]	[1]	~	PMU ROM code experienced an error during the preboot process.

Table 6-12: **PMU Error Sources and Reset State Masks *(Cont'd)***

System Error	ERROR_SIG_MASK_n	ERROR_STATUS Register and [Bits]	JTAG Error Register	GIC IRQ	Description
PMU ROM code service errors	U	_2 [24]	[2]	~	PMU ROM code experienced an error processing a service request.
PMU firmware defined interrupt bits.	U	_2 [21:18]	[6:3]	~	PMU user firmware reported an error code.
FSBL detected errors				~	
Hardware Errors					
PMU hardware errors	U	_2 [17]	[7]		PMU ROM validation, TMR fault, RAM UE ECC, or register address access error.
CSU error	U	_2 [16]	[8]		CSU hardware errors. Includes CSU ROM validation error.
PMU_PB		_2 [25]			
PLL lock errors	M	_2 [12:8]	[13:9]		PMU unmasks these bits when PLL is functioning. An error is signaled when a PLL loses lock; bits are in ERROR_STATUS_2.
Generic PL errors	U	_2 [5:2]	[17:14]		Generic PL errors communicated to PS.
FPD bus timeout error	U	_2 [1]	[18]	153	OR of all timeout signals from the FPD AIB units; ABP and AXI.
LPD bus timeout error	U	_2 [0]	[19]	86	OR of all timeout signals from the LPD AIB units; ABP and AXI.
Clock monitor error	U	_1 [26]	[25]	60	Error from clock monitor logic.
FPD XMPU isolation error	U	_1 [25]	[26]	166	OR of violation signals from the FPD and DDRx XMPU protection units.

Table 6-12: **PMU Error Sources and Reset State Masks *(Cont'd)***

System Error	ERROR_SIG_MASK_n	ERROR_STATUS Register and [Bits]	JTAG Error Register	GIC IRQ	Description
LPD XMPU isolation error	U	_1 [24]	[27]	120	OR of violation signals from the OCM XMPU and the XPPU protection units.
Power supply failures detected by PS SYSMON unit	U	_1 [23:16]	[35:28]	~	[16]: VCC_PSINTLP, [17]: VCC_PSINTFP, [18]: VCC_PSAUX, [19]: VCCO_PSDDR, [20]: VCC_PSIO3, [22]: VCC_PSIO0, [21]: VCC_PSIO1, [23]: VCC_PSIO2
FPD SWDT error	U	_1 [13]	[36]	145	Timeout error from the FPD SWDT.
LPD SWDT error	U	_1 [12]	[37]	84	Timeout error from the LPD SWDT.
RPU CCF	U	_1 [9]	[38]	~	All RPU CCFS OR'ed together after RPU_CCF_MASK register.
RPU lock-step errors	M	_1 [7:6]	[40:39]	~	RPU lock-step errors from RPU MPCore.
FPD over temperature	U	_1 [5]	[41]	~	FPD temperature near APU indicates a shutdown alert from the PS SysMon unit.
LPD over temperature	U	_1 [4]	[42]	~	LPD temperature near RPU indicates a shutdown alert from the PS SysMon unit.
RPU hardware errors	U	_1 [3:2]	[44:43]	45, 44	RPU0 or RPU1 error including both correctable and uncorrectable errors.

Table 6-12: **PMU Error Sources and Reset State Masks *(Cont'd)***

System Error	ERROR_SIG_MASK_n	ERROR_STATUS Register and [Bits]	JTAG Error Register	GIC IRQ	Description
OCM uncorrectable ECC	M	_1 [1]	[45]	42	The OCM reported an uncorrectable ECC error during an OCM memory access.
DDR uncorrectable ECC	M	_1 [0]	[46]		The DDR reported an uncorrectable ECC error during a DDR memory access.

All the errors listed in Table 6-12 and the five reserved errors are also routed to the PL and are directly accessible through JTAG. In addition to these errors, the 74 bits of software errors from the PMU_PB_ERR, CSU_BR_ERR, and PMU_SERV_ERR registers are also accessible directly through JTAG. You can suppress the accessibility to these errors through JTAG permanently by blowing an eFUSE. Table 6-13 lists the assignment of errors in the JTAG status register and the error status interface to PL.

Note: The eFUSE suppresses accessibility of the errors through JTAG, but the errors are accessible internal to the device.

Table 6-13: **JTAG Error Register Description**

Error source	Bit on JTAG Error Status	Bit on Error Status to PL
CSU ROM error (same as bit 120).	0	0
PMU pre-boot error (same as bit 78).	1	1
PMU ROM service error (same as bit 99).	2	2
PMU firmware error (same as bits 103:100).	6:3	6:3
Uncorrectable PMU error. Includes ROM validation, TMR, uncorrectable RAM ECC, and local register address errors.	7	7
CSU error.	8	8
PLL lock errors [VideoPLL, DDRPLL, APUPLL, RPUPLL, IOPLL].	13:9	13:9
PL generic errors passed to PS.	17:14	17:14
Full-power subsystem time-out error.	18	18
Low-power subsystem time-out error.	19	19
Reserved errors.	24:20	24:20
Clock monitor error.	25	25

Table 6-13: **JTAG Error Register Description *(Cont'd)***

Error source	Bit on JTAG Error Status	Bit on Error Status to PL
XMPU errors [FPD XMPU, LPD XMPU].	27:26	27:26
Supply Detection Failure Errors [VCCO_PSIO_2, VCCO_PSIO_1, VCCO_PSIO_0, VCCO_PSIO_3, VCCO_PSDDR, VCC_PSAUX, VCC_PSINTFP, VCC_PSINTLP]	35:28	35:28
FPD System Watch-Dog Timer Error	36	36
LPD System Watch-Dog Timer Error	37	37
RPU CCF error	38	38
RPU Lockstep Error	40:39	40:39
FPD Temperature Shutdown Alert	41	41
LPD Temperature Shutdown Alert	42	42
RPU1 Error (Both Correctable and Uncorrectable Errors)	43	43
RPU0 Error (Both Correctable and Uncorrectable Errors)	44	44
OCM Uncorrectable ECC Error	45	45
DDR Uncorrectable ECC Error	46	46
PMU Preboot Errors (PMU_PB_ERR.PBERR_Data)	77:47	77:47
PMU Preboot Error Flag (PMU_PB_ERR.PBERR_Flag)	78	78
PMU Service Errors (PMU_SERV_ERR.SERVERR_Data)	98:79	98:79
PMU Service Error Flag (PMU_SERV_ERR.SERVERR_Flag)	99	99
PMU Firmware Error (PMU_SERV_ERR.FWERR)	103:100	103:100
CSU BootROM Errors (CSU_BR_ERR.ERR_TYPE)	119:104	119:104
CSU BootROM Errors (CSU_BR_ERR.BR_ERROR)	120	120

Operation

The PMU is responsible for handling the primary pre-boot tasks and management of the PS hardware for reliable power up/power down of system resources and system error management. Optionally, the PMU can run the Xilinx Software Test Library. The power-on-reset (POR) initiates the PMU operation which directly or indirectly releases resets to any other blocks that are expected to be powered up.

In the PS, the APU MPCore and Cortex-R5F are classified as power masters. Power masters in the system are entities that can trigger the power down or power up of all islands including themselves.

GPU pixel processors, USB, PL, and memory blocks are classified as power slaves as their power management is triggered by one of the power masters. The power masters can also be slaves because their islands can be individually powered down.

When the processors in the PS are powered down, the PMU is the sole entity in the PS that can capture a request to power up the required system and wake up the target processor.

PMU GPIs can be used as inputs for external wake signals. The ULPI and RGMII are potentially used for wakes on USB 2.0 and Ethernet, respectively. PMU GPOs are used for sending signals to power supplies and communicating errors. For a detailed description of PMU GPIs and GPOs, see PMU GPIs and GPOs.

Interacting with the PMU

User software services requests from the PMU through the PMU_GLOBAL registers generate interrupts to the PMU processor and are processed automatically in the priority set by the PMU ROM code. The requests are initiated by user software enabling the service request and subsequently asserting the associated trigger for the service. The assertion of the enabled trigger asserts an associated status flag. Once the PMU has completed the service, it clears the status flag indicating to the user software that the service has completed. If the service has experienced a failure, the PMU_SERVICE bit of the PMU_GLOBAL.ERROR_STATUS_2 register is asserted and the system responds according to the mask settings for that error event. For all software generated requests to the PMU, the above sequence is recommended for usage.

Power Down

Any master in the system can request the PMU to power down an island or domain by writing a `1` to the appropriate bits in the REQ_PWRDWN_TRIG register while the corresponding mask bit is also enabled in the REQ_PWRDWN_INT_MASK register. The PMU will be interrupted and after executing the preamble ISR to check the interrupt pending register within the I/O block, it will execute the power-down-request ISR. In the case of a simultaneous power down request, the order for processing power-down requests is that the islands are powered down before the domains. The PMU will proceed to power down an island only if there is no other request from a master to power it up.

Power Up

Any master in the system can queue a request to the PMU to power up an island or domain by writing a `1` to the appropriate bits in the REQ_PWRUP_TRIG register. If a `1` is also written to the same bit in the REQ_PWRUP_INT_MASK register, the PMU will be interrupted. After executing the preamble ISR to check the interrupt pending register within the I/O block, it will execute the power-up request ISR. The priority of the power up is enforced such that domains are powered up first, then the islands, followed by slaves, and then finally the masters.

Use Case for Power Down and Power Up by PMU

This section describes power-down and power-up using the Zynq UltraScale+ MPSoC PMU.

APU Power Down

A few methods to power-down the APU are described in this section.

Direct Power Down

The flowchart in Figure 6-3 describes how to power down using the APU. As a preparation for power down, the APU program must follow these steps.

- Disable interrupts to the core.
- Record the intention to power down the CPU in the CPUPWRDWNREQ field of the PWRCTL register in APU by writing `1` to the field that corresponds to that APU core.
- Save the state of the APU core.
- Configure the GIC or GIC proxy (if the ACPU power-down is expected to be followed by the FPD power-down) for the wake source.
- Execute a waiting for interrupt (WFI) instruction.

Because the CPUPWRDWNREQ field marks the intention of the APU core to power down, the execution of the WFI instruction not only puts the APU core in a wait state, it also causes the power-down request to propagate outside the core and inform the PMU processor by asserting the GPI2 interrupt.

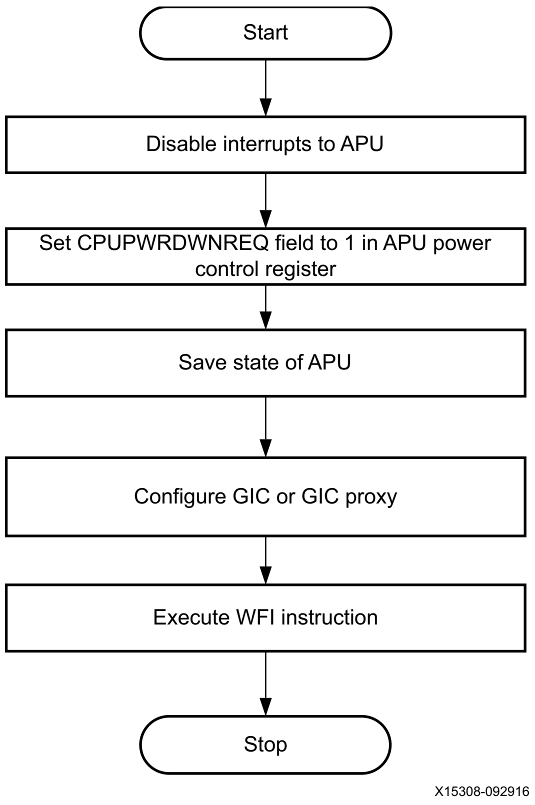

Start
Disable interrupts to APU
Set CPUPWRDWNREQ field to 1 in APU power control register
Save state of APU
Configure GIC or GIC proxy
Execute WFI instruction
Stop
X15308-092916

Figure 6-3: **APU Power Down Flowchart**

Requested Power Down

A requested power down occurs when the APU core power down is specifically requested through the REQ_PWRDWN_TRIG global registers. Setting a particular bit in the register would power down the APU. In this case, the PMU directly proceeds with powering down the APU Core. For REQ_PWRDWN_TRIG register description see the *Zynq UltraScale+ MPSoC Register Reference* (UG1087) [Ref 4].

Ensure that the appropriate bit position in the REQ_PWRDWN_STATUS global register is set to `0` to indicate that the power down request is served by the PMU.

APU Core Power Up

Unlike power down, powering up an APU core is typically requested either by another CPU through power-up request registers on the PMU or by interrupts that are associated with the peripherals on the powered-down APU core. For the latter, the interrupts for these peripherals are passed to the PMU when the ACPU is powered down. For power up, follow these steps.

- For powering up an APU core, the particular bit in the REQ_PWRUP_TRIG global register has to be set by the requesting device. For the description of REQ_PWRUP_TRIG global register, see the Register Overview section.
- If a direct power-up or wake by the GIC is associated with the APU core, the PMU follows the steps as specified by the ROM code and powers up the APU. A direct power-up refers to a power-on event triggered by an interrupt destined for the APU core, as opposed to software triggering the event by writing to the request register in the PMU_GLOBAL module.
- If a direct wake up or wake by GIC occurs after the power-up is completed, the reset to the APU core is also released automatically.
- If the power-up request is made by another processor, the same processor has to explicitly request for the reset to the APU core be released through the PMU reset-request register.
- Check if the appropriate bit position in the REQ_PWRUP_STATUS global register is set to `0` to indicate that the power up request is served by the PMU.

IMPORTANT: *After the power-up, the CPUPWRDWNREQ field of the PWRCTL register in the APU contains the value of 1 as the power status for the core that is just powered up. The CPU is expected to check the register, upon boot, to identify if this was a cold boot or a wake from sleep. Post-verification, the processor is expected to clear the bit in the CPUPWRDWNREQ field of the PWRCTL register.*

PMU Operation After a Wake-up

After receiving a wake-up trigger, the PMU can follow these three wake-up flows.

Fixed: Direct wake of a processor, will always cause the target processor to be powered up. For example, when the dual Cortex-R5F MPCores are powered down and any of the two receives an interrupt from a peripheral or a timer, the interrupt does the following.

- Route to the PMU to trigger the power up of the dual Cortex-R5F MPCores.
- Release its reset to prepare for processing of the pending interrupt.

Similarly, if an APU core is powered down while the FP domain is up, the interrupts for the APU core that was shutdown can trigger its power up followed by the release of its reset.

On-demand: Prior to requesting a power-down and entering the sleep mode, the user program can queue up the list that needs to be powered up after the wake in the PMU. The following procedure should achieve this.

1. Your program requests to power up the desired domains and islands using the REQ_PWRUP_TRIG register while masking the interrupt for those requests in the REQ_PWRUP_INT_DIS register. Even though the requests are recorded, the PMU does not actually execute them until after the wake-up.
2. Your program follows up with the normal request for power down. Because the interrupt for the power-ups were masked, the power-down routine ignores those requests and proceeds with powering down the blocks.
3. When the PMU receives a wake-up request, it checks the REQ_PWRUP_STATUS register for pending power-up requests with the interrupt being masked and proceeds with powering up those islands.
4. Similarly, if reset to any block needs to be released after the power up, your code queues up the requests to release those resets in the REQ_SWRST_TRIG register while masking their interrupts.
5. After the wake-up and its consequent power-up, the PMU releases the reset to the desired blocks.

Wake-up Code Programming: The wake up routine can be programmed into the PMU RAM and when a wake interrupt occurs the PMU executes your code which powers up all the blocks that are necessary after the wake-up.

Wake-up Through MIO

The following wake-up mechanisms can respond to any of the six GPI signals from the six MIO inputs (MIO 26 to 31) that are allocated to the PMU.

- Wake-up on external events
- Wake-up on Ethernet PHY
- Wake-up on CAN PHY

Based on the mechanism, any interrupt raised by the above interfaces, is issued to the PMU to wake up the device which has set the interface as its wake-up source.

Wake-up on USB

The USB specification defines a link-layer suspend mode in which both the USB host and the device enter a no-activity phase to save power. The decision to take the USB host into the suspend mode is determined solely by the software. Once the host enters the suspend mode, all devices connected to that host are required to enter the suspend mode within 3 ms. A USB device could not enter the suspend mode by itself; however, when the link power management (LPM) extension is supported, the USB device can request the USB host to enter the suspend mode. When the USB host enters the suspend mode, all USB devices will follow.

A USB host can exit the suspend mode either through interrupts such as timers or through a remote wake-up request by a device with special USB signal leveling. A USB device can similarly wake up through interrupts or remote wake signaling from host or additionally through host reset signaling.

When the USB is in a suspend mode, the USB ULPI link protocol provides a standard method for the PHY to power-down during a time when the D+/D- signaling is directed to the USB link. In this case, a subsection of the USB IP that is always on, detects the wake signaling and generates the wake interrupt to the PMU to proceed with powering up the USB block and the processor that is responsible for its device driver.

Wake-up on Ethernet

Wake-up by Ethernet can be performed two ways.

Wake on PHY: This wake-up procedure can be implemented using a GPI input signal routed from an MIO pin.

Wake on MAC: This wake up procedure is widely referred to as wake-on-LAN. This procedure is implemented using a special network message called a magic packet. The magic packet is a broadcast frame containing anywhere within its payload 6 bytes of all 255 (`FF FF FF FF FF FF`), followed by sixteen repetitions of the target computer's 48-bit MAC address, for a total of 102 bytes. The detection of the magic packet will generate an interrupt to the processor that is running the device driver which causes a direct wake on the processor.

Wake on Real-time Clock

This feature allows the system to wake up at a pre-determined time using the internal real-time clock (RTC). Configure the RTC to generate an interrupt when it reaches a specific time and date.

Wake through DAP

This feature wakes up a system that is in the sleep mode through the debugger. The debugger can request two possible direct power-up scenarios through DAP. One option can wake up the FP domain which includes the MPSoC debug. The other wake option initiates the power-up of the dual Cortex-R5F subsystem.

Direct Wake by the APU or Cortex-R5F

When any of the application processors or the real-time processors are powered down, if a peripheral is attempting to interrupt the powered down processor, the interrupt is routed to the PMU to trigger the power up of that specific processor.

Wake through GIC Proxy

If the power down of an application processor is in conjunction with the power down of the entire FPD, an LPD device that is associated with that processor can still trigger a direct wake to that processor by first triggering the power up of the FPD. This is accomplished by having a GIC proxy block in the LPD that can have selected peripheral interrupts routed to the PMU as an interrupt other than the direct wakes.

Upon receiving an interrupt from the GIC proxy block:

1. The PMU powers up the FPD.
2. Releases the reset to the FPD and APU.
3. Unmasks the interrupts that trigger the direct wake of that application processor.

The direct wake will take effect resulting in the power up of the application processor.

Deep-sleep Mode

The deep-sleep mode suspends the PS and waits to be woken up. The lowest power deep sleep is supported for wake sources GPI and RTC. Other sleep states are supported for wake sources of USB and Ethernet, with additional power for the wake source. Upon wake, the PS does not have to go through the boot process and the security state of the system is preserved. This reduces the restart time of the system.

The device consumes the lowest power during this mode while still maintaining its boot and security state. The PMU is placed in a sleep or suspend state waiting to be interrupted.

During the deep-sleep mode, the wake signal can be generated either through a GPI input routed from an MIO pin or by an RTC alarm.

Table 6-14 summarizes the PS configuration in deep-sleep mode.

Table 6-14: **Deep-sleep Configuration**

Configuration Type	Status	Description
Cortex-R5F	Powered down	
TCM configuration OCM configuration Device security	In retention In retention Suspended	Either TCM or OCM is powered down.
Peripheral	Suspended	Wake up peripheral logic might be active.
PLLs	Powered down	
System Monitor	Powered down	During power down, the SysOsc clock can go to 20 MHz ±50%.
RTC and BBRAM	Included	Switched to the V_{CC_PSAUX} rail.
PMU MPSoC debug	Suspended Powered down	The wake logic is active. MPSoC debug is mostly in FPD. The LPD portion is suspended.
eFUSE Components outside the LPD	Suspended Powered down	
PL internal power	Powered down	

Deep-sleep Mode Programming Model/Example

The processing system in deep-sleep mode is discussed in this section.

System Configuration prior to Sleep

System includes at least the following devices.

- The Cortex-R5F processor in the lock-step mode.
- TCM memory.
- Real-time counter.

System Configuration during Sleep

The configuration of the system during sleep is discussed in this section.

- FPD is powered off.
- RPU, USBs, and OCM are powered off.
- TCM is in retention.
- RTC alarm is set and RTC is functioning.
- PLLs are powered down.

- System Monitor is powered down.

Power Down Procedure

The power down is initiated by the Cortex-R5F MPCore. As the TCM is placed in retention, the Cortex-R5F MPCore is required to do the following (Figure 6-4).

1. Set the TCM bit in the RAM_RET_CNTRL register.
2. Set the TCM bit in the REQ_PWRDWN_TRIG register while the interrupt is masked for the TCM in the REQ_PWRDWN_INT_MASK register.
3. Set the RPU and TCM bits in the REQ_PWRUP_TRIG register while the interrupt mask bits for those fields are disabled.
4. Set the RPU bit in the REQ_SWRST_TRIG while the interrupt mask bit for it is disabled.
5. Set the alarm.
6. Disable interrupts.
7. Set the SLCR bit to request for a direct RPU power down and execute a WFI instruction.

This procedure causes an interrupt to the PMU to power down the RPU.

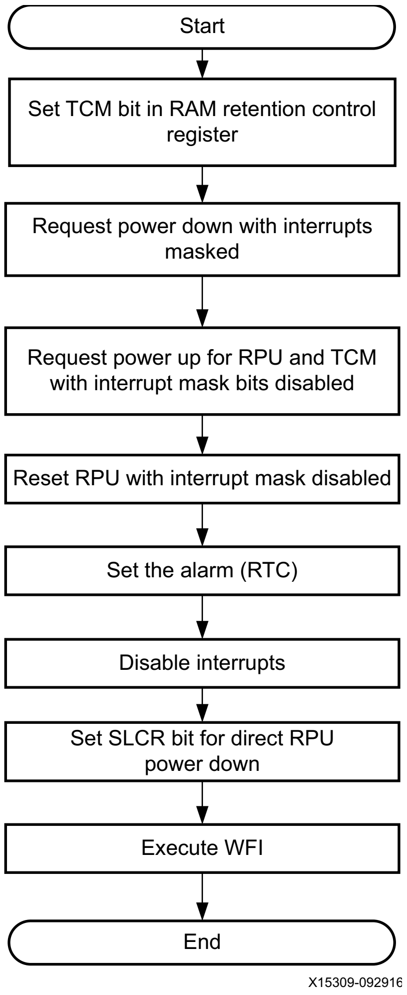

Start
Set TCM bit in RAM retention control register
Request power down with interrupts masked
Request power up for RPU and TCM with interrupt mask bits disabled
Reset RPU with interrupt mask disabled
Set the alarm (RTC)
Disable interrupts
Set SLCR bit for direct RPU power down
Execute WFI
End
X15309-092916

Figure 6-4: **Deep Sleep Power Down Flowchart**

Wake Procedure

Once the RTC alarm generates an interrupt to the PMU, the handler for the RTC wake detects if there is a firmware loaded for this purpose. If not, the handler checks whether an on-demand procedure is queued up in the PMU. Prior to the power down, the Cortex-R5F MPCore requests for the power up of the RPU and TCM while the interrupts for the power-up requests are masked. It requests the Cortex-R5F MPCore reset to be released while the interrupt for that request is masked, again. Upon waking up from the RTC, the PMU proceeds with the RPU power-up and issues the Cortex-R5F MPCore reset. Figure 6-5 shows the flowchart for wake-up from a deep sleep.

1. RPU and TCM power up requests are unmasked as a part of the RTC wake.
2. Cortex-R5F MPCore reset request is unmasked as a part of the RTC wake.
3. TCM is powered up first as a result of the follow-up TCM power-up interrupt.
4. RPU is powered up as a result of the follow-up RPU power-up interrupt.

5. Reset to the Cortex-R5F MPCore is released as a result of the follow up Cortex-R5F MPCore reset request interrupt.
6. Your code on the Cortex-R5F MPCore releases the system monitor out of the power down state.
7. The code on the Cortex-R5F MPCore clears the RTC alarm. Because the RTC has an interrupt status register, setting the alarm bit to 1 clears the interrupt.

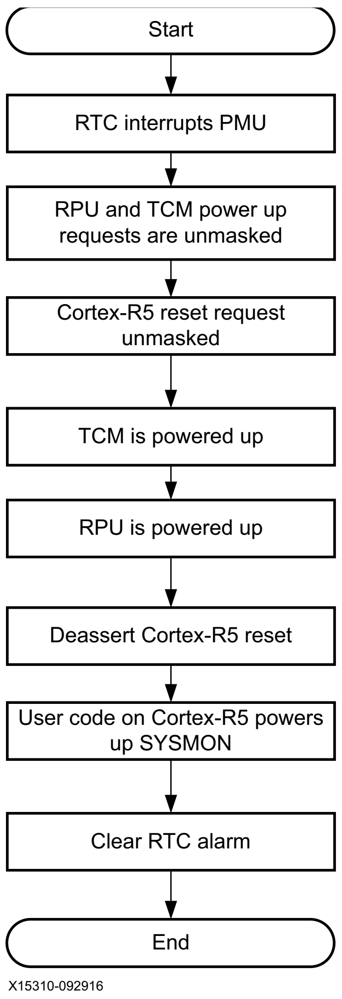

Start
RTC interrupts PMU
RPU and TCM power up requests are unmasked
Cortex-R5 reset request unmasked
TCM is powered up
RPU is powered up
Deassert Cortex-R5 reset
User code on Cortex-R5 powers up SYSMON
Clear RTC alarm
End
X15310-092916

Figure 6-5: **Wake up from Deep Sleep Flowchart**

Isolation Request

Isolation is generally used to isolate signals from a powered-up domain and a powered-down domain to prevent crowbar currents affecting the proper functioning of the blocks. Isolation ensures that the outputs of the domains are clamped to a known value. The PMU facilitates isolation of various power domains. This can be done by setting appropriate bits in the REQ_ISO_TRIG global register. For the PMU_GLOBAL.REQ_ISO_STATUS register description, see the *Zynq UltraScale+ MPSoC Register Reference* (UG1087) [Ref 4]. Three bits control domain isolation between the low-power, full-power, and PL domain. Different combinations of isolation are available. By writing to bit 0 of the REQ_ISO_TRIG register and the REQ_ISO_INT_MASK register, the full-power domain can be isolated from the low-power domain and the PL domain. By writing to bit 1 of these registers, the PS is isolated from the PL. By writing to bit 2 the PS and the PL are isolated, with the exception of the PCAP interface. Finally, to lock isolation on the full-power domain, write to bit 4.

Reset Services

This section describes the reset services. Various blocks can be reset through the REQ_SWRST_TRIG register if the interrupt for that specific reset is unmasked in the REQ_SWRST_INT_MASK register. The Table 6-15 lists the reset services.

Table 6-15: **Reset Requests**

Reset Service Block	Request Bit	Description
PL	31	Resetting the PL domain depends on your design. This service is not handled by ROM code.
FPD	30	A hard reset of the full-power domain. Transactions are not flushed.
LPD	29	The PMU firmware uses this service to reset the low-power domain. This service is not handled by ROM code.
PS_ONLY	28	Acts as an internally generated a system reset (SRST). You can perform an isolation request on the PL prior to this event and then issue this request to only SRST the PS.
Reserved	27:26	Reserved
USB1	25	Cycles the reset for USB_1 by asserting the CRL_APB.RESET_LPD_TOP. USB1_CORERESET signal and then deasserting it.
USB0	24	Cycles the reset for USB_0 by asserting the CRL_APB.RESET_LPD_TOP.USB0_CORERESET signal and then deasserting it.
GEM3	23	Cycles the reset for GEM_3 by asserting the CRL_APB.RESET_IOU0.GEM3_RESET signal and then deasserting it.
GEM2	22	Cycles the reset for GEM_2 by asserting the CRL_APB.RESET_IOU0.GEM2_RESET signal and then deasserting it.
GEM1	21	Cycles the reset for GEM_1 by asserting the CRL_APB.RESET_IOU0.GEM1_RESET signal and then deasserting it.

Table 6-15: **Reset Requests *(Cont'd)***

Reset Service Block	Request Bit	Description
GEM0	20	Cycles the reset for GEM_0 by asserting the CRL_APB.RESET_IOU0.GEM0_RESET signal and then deasserting it.
Reserved	19	Reserved
RPU	18	This service performs a sequence that resets the entire RPU and leaves the block in reset. You can request the R5_0 or R5_1 service to release the appropriate signal. The following resets signals are asserted: • PMU_GLOBAL_RESET_RPU_LS • CRL_APB.RESET_LPD_TOP.RPU_PGE_RESET • CRL_APB.RESET_LPD_TOP.R50_RESET • CRL_APB.RESET_LPD_TOP.R51_RESET The following signals release the resets. • PMU_GLOBAL.RESET_RPU_LS • CRL_APB.RESET_LPD_TOP.PRPU_PGE_RESET Prior to issuing an RPU request, the application should flush transactions to the RPU. The debug logic is not reset.
R5_1	17	Cycles the reset for the APU1 (R5_1) by asserting the CRL_APB.RESET_LPD_TOP.R51_RESET signal and then deasserting it.
R5_0	16	Cycles the reset for APU0 (R5_0) by asserting the CRL_APB.RESET_LPD_TOP.R51_RESET signal and then deasserting it.
Reserved	15:13	Reserved
Display_Port	12	Cycles the reset for the DisplayPort controller by asserting the CRL_APB.RESET_FPD_TOP.DP_RESET signal and then deasserting it.
Reserved	11	Reserved
SATA	10	Cycles the reset for the SATA controller by asserting the CRL_APB.RESET_FPD_TOP.SATA_RESET signal and then deasserting it.
PCIe	9	Cycles the reset for PCIe by asserting the CRL_APB.RESET_FPD_TOP.PCIE_RESET signal and then deasserting it.
GPU	8	This service performs a sequence that resets the entire GPU. Both pixel processors and the GPU resets are asserted and released by the following signals. • CRF_APB.RESET_FPD_TOP.GPU_RESET • CRF_APB.RESET_FPD_TOP.PP1_RESET • CRF_APB.RESET_FPD_TOP.PP0_RESET
PP1	7	Cycles the individual reset for the pixel processor by asserting the CRF_APB.RESET_FPD_TOP.GPU_PP1_RESET signal and the deasserting it.
PP0	6	Cycles the individual reset for the pixel processor by asserting the CRF_APB.RESET_FPD_TOP.GPU_PP0_RESET signal and the deasserting it.

Table 6-15: **Reset Requests *(Cont'd)***

Reset Service Block	Request Bit	Description
Reserved	5	Reserved
APU	4	This service performs a sequence that resets the entire APU and L2 and leaves them in reset until the ACPU reset service (bits 3:0) are requested while cycling the reset on the L2 and surrounding APU logic. The debug logic is not reset. The following reset signals are asserted: • CRF_APB.RESET_FPD_APU.L2_RESET • CRF_APB.RESET_FPD_APU.ACPU3_RESET • CRF_APB.RESET_FPD_APU.ACPU2_RESET • CRF_APB.RESET_FPD_APU.ACPU1_RESET • CRF_APB.RESET_FPD_APU.ACPU0_RESET The L2_RESET is released to make the L2 available.
ACPU3	3	Cycles the individual reset for the APU by asserting the CRF_APB.RESET_FPD_APU.ACPU3_RESET and the deasserting it.
ACPU2	2	Cycles the individual reset for the APU by asserting the CRF_APB.RESET_FPD_APU.ACPU2_RESET and the deasserting it.
ACPU1	1	Cycles the individual reset for the APU by asserting the CRF_APB.RESET_FPD_APU.ACPU1_RESET and the deasserting it.
ACPU0	0	Cycles the individual reset for the APU by asserting the CRF_APB.RESET_FPD_APU.ACPU0_RESET and the deasserting it.

Programming Model

Beyond the Xilinx provided firmware, the PMU can execute user programs that implement advance system monitoring and system-critical functions. Typically, PMU code loading occurs either via CSU ROM code at boot or by the first stage boot loader (FSBL). During this time, the PMU is either in an already-loaded maintenance mode or in the sleep mode. To assure that the PMU is in the sleep mode, IPI0 is used to interrupt the PMU. In response to the IPI0 interrupt, the interrupt service routine for this IPI disables interrupts and executes a sleep instruction followed by a branch to the user code being loaded in the RAM. This guarantees that the processor stays in the sleep mode and is not interrupted to execute any services until it is explicitly woken up by another master through the use of the wake-up bit in the PMU global control register. After the main processor copies the user program into the PMU RAM, the processor wake-up feature in the PMU global control register is used to direct the PMU processor into executing the newly-loaded maintenance code.

The steps required to load a user-level program and start its execution are listed here and shown in Figure 6-6.

1. Application program on another processor either APU or RPU executes IPI0 to the PMU.
2. IPI0 interrupt service routine.
3. Disables all interrupts.
4. Executes a sleep instruction. The instruction after the sleep instruction must be a branch to the address for the user code in RAM.
5. The application program loads the PMU user program into the RAM.
6. The application program writes a `1` to bit [0] of the PMU global control register to wake up the processor.
7. PMU starts executing instructions following the sleep instruction and returns to the main() function in the code.
8. PMU branches to the user code.
9. The user code clears the bit [0] in the PMU global control register and enables the interrupt.

An upper-level program can check the PMU global control register to determine the state of the firmware loading and execution.

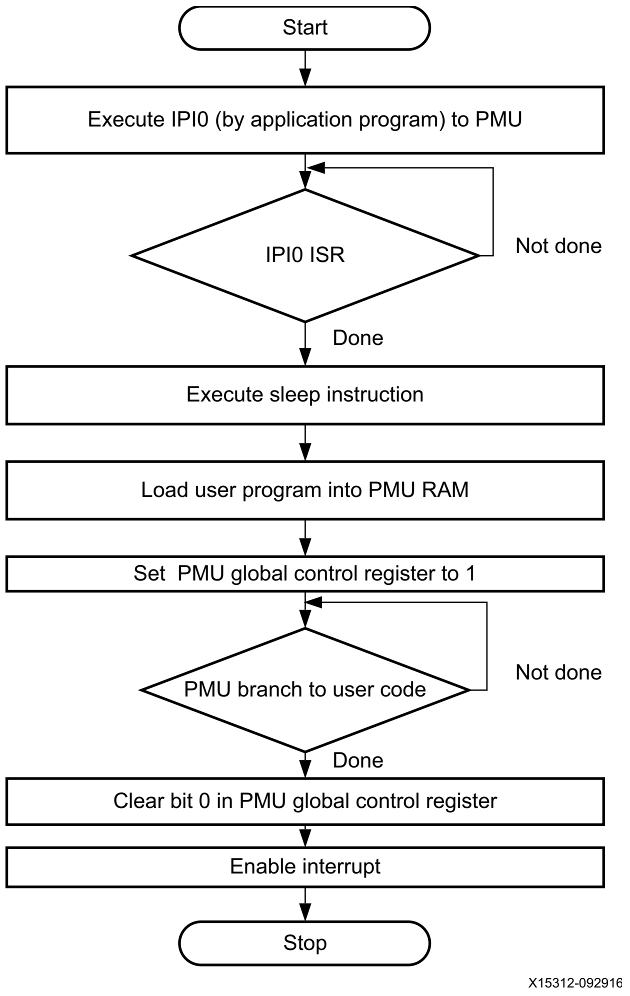

Start
Execute IPI0 (by application program) to PMU
IPI0 ISR
Not done
Done
Execute sleep instruction
Load user program into PMU RAM
Set PMU global control register to 1
PMU branch to user code
Not done
Done
Clear bit 0 in PMU global control register
Enable interrupt
Stop
X15312-092916

Figure 6-6: **PMU Programming Model**

Register Overview

The registers in Table 6-16 are in the PMU_GLOBAL module. For more information, see the *Zynq UltraScale+ MPSoC Register Reference* (UG1087) [Ref 4].

Table 6-16: **Global Registers**

Register Name	Type	Description
GLOBAL_CNTRL	Mixed	This register controls functions such as QoS for AXI read and write transactions that are generated by the PMU, or indication for firmware presence that can also be executed by other masters.
PS_CNTRL	Mixed	This register controls miscellaneous functions related to the PS that can be controlled by all masters.
APU_PWR_STATUS_INIT	Mixed	Provides a location in the PMU to hold the initialization value for the CPUPWRDWNREQ field of the APU PWRCTL register during an FPD power down. The bit associated with an ACPU is loaded by the PMU ROM code in the CPUPWRDWNREQ field of the PWRCTL register right after the routine releases the reset to the ACPU core after an FPD power up. `0` = Normal cold reset (default) `1` = Reset after a power up after a shutdown mode
ADDR_ERROR_STATUS	Mixed	Address error status register. This is a sticky register that holds the value of the interrupt until cleared by a value of `1`.
ADDR_ERROR_INT_MASK	RO	Address error mask register. This is a read-only location and can be altered through the corresponding interrupt Enable or Disable registers.
ADDR_ERROR_INT_EN	WO	Address error interrupt enable register. A write to this location will unmask the interrupt.
ADDR_ERROR_INT_DIS	WO	Address error interrupt disable register. A write of `1` to this location will mask the interrupt.
GLOBAL_GEN_STORAGE{0:6}	RW	Global general storage register that can be used by system to pass information between masters. The register is reset during system or power-on resets. These registers are used by the PMUFW, FSBL, and other Xilinx software products.
PERS_GLOB_GEN_STORAGE{0:7}	RW	Persistent global general storage register that can be used by system to pass information between masters. This register is only reset by the power-on reset and maintains its value through a system reset. Four registers are used by the FSBL and other Xilinx software products: PERS_GLOB_GEN_STORAGE{4:7}. Register is reset only by a POR reset.
DDR_CNTRL	RW	This register controls DDR I/O features that have to be driven when the FPD is powered down.

Table 6-16: **Global Registers *(Cont'd)***

Register Name	Type	Description
PWR_STATE	RO	This register provides the power-up status for all islands within the PS. (0 = powered down). Reserved bits read as zero. The register maintains its contents during a system reset.
AUX_PWR_STATE	RO	This register provides the retention state for the PS memories (1 = retention) and the power-down emulation state for the Arm processor. (1 = powered-down emulation state). The register maintains its contents during a system reset.
RAM_RET_CNTRL	Mixed	This register is used to enable retention request for the L2, OCM, and TCM RAMs. If a bit in this register is set, a power-down request of the corresponding RAM bank would guide the PMU to put the RAM in retention, instead.
PWR_SUPPLY_STATUS	RO	This register provides the status of a subset of the power supplies within the PS
REQ_PWRUP_STATUS	Mixed	If any of the bits in this register is 1, it would trigger a power-up request to the PMU. Writing a 1 to any bit will clear the request.
REQ_PWRUP_INT_MASK	RO	Power-up request interrupt mask register. This is a read-only location and can be altered through the corresponding interrupt enable or disable registers.
REQ_PWRUP_INT_EN	WO	Power-up request interrupt enable register. Writing a 1 to this location will unmask the interrupt.
REQ_PWRUP_INT_DIS	WO	Power-up request interrupt disable register. Writing a 1 to this location will mask the interrupt.
REQ_PWRUP_TRIG	WO	Power-up request trigger register. A write of 1 to this location will generate a power-up request to the PMU.
REQ_PWRDWN_STATUS	Mixed	If any of the bits in this register is 1, it would trigger a power-down request to the PMU. Writing a 1 to any bit will clear the request.
REQ_PWRDWN_INT_MASK	RO	Power-down request interrupt mask register. This is a read-only location and can be altered through the corresponding interrupt enable or disable registers.
REQ_PWRDWN_INT_EN	WO	Power-down request interrupt enable register. Writing a 1 to this location will unmask the interrupt.
REQ_PWRDWN_INT_DIS	WO	Power-down request interrupt disable register. Writing a 1 to this location will mask the interrupt.
REQ_PWRDWN_TRIG	WO	Power-down request trigger register. Writing a 1 to this location will trigger a power-down request to the PMU.
REQ_ISO_STATUS	Mixed	If any of the bits in this register is 1, it would capture an Isolation request to the PMU. Writing a 1 to any bit will clear the request.

Table 6-16: **Global Registers *(Cont'd)***

Register Name	Type	Description
REQ_ISO_INT_MASK	RO	Isolation request interrupt mask register. This is a read-only location and can be altered through the corresponding interrupt enable or disable registers.
REQ_ISO_INT_EN	WO	Isolation request interrupt enable register. A write of `1` to this location will unmask the interrupt.
REQ_ISO_INT_DIS	WO	Isolation request interrupt disable register. A write of `1` to this location will mask the interrupt.
REQ_ISO_TRIG	WO	Isolation request trigger register. A write of `1` to this location will set the corresponding isolation status register bit.
REQ_SWRST_STATUS	Mixed	If any of the bits in this register is `1`, it triggers a reset request to the PMU. Writing a `1` to any bit clears the request.
REQ_SWRST_INT_MASK	RO	Reset request interrupt mask register. This is a read-only location and can be altered through the corresponding interrupt enable or disable registers.
REQ_SWRST_INT_EN	WO	Reset request interrupt enable register. A write of `1` to this location will unmask the interrupt.
REQ_SWRST_INT_DIS	WO	Reset request interrupt disable register. A write of `1` to this location will mask the interrupt.
REQ_SWRST_TRIG	WO	Reset request trigger register. A write of `1` to this location will set the reset status register related to this interrupt.
REQ_AUX_STATUS	Mixed	If any of the service request bits in this register is `1`, it would capture an auxiliary request to the PMU. Writing a `1` to any bit will clear the request. The services for these requests need to be implemented by firmware.
REQ_AUX_INT_MASK	RO	Auxiliary service request interrupt mask register. This is a read-only location and can be altered through the corresponding interrupt enable or disable registers.
REQ_AUX_INT_EN	WO	Auxiliary service request interrupt enable register. A write of `1` to this location will unmask the interrupt.
REQ_AUX_INT_DIS	WO	Auxiliary service request interrupt disable register. A write of `1` to this location will mask the interrupt.
REQ_AUX_TRIG	WO	Auxiliary service request trigger register. A write of `1` to this location will set the corresponding auxiliary service status register bit.
LOGCLR_STATUS	RO	This register provides the status of the logic clear engines after they are run. (`0` = Fail)
CSU_BR_ERROR	Mixed	This register holds all errors related to the BootROM execution on the CSU.
MB_FAULT_STATUS	RO	This register provides the status of the redundancy logic in the triple-redundant PMU processor.

Table 6-16: **Global Registers *(Cont'd)***

Register Name	Type	Description
ERROR_STATUS_1	Mixed	Error status register 1. If the bit in this register is set to 1, it signifies an error within the system. Writing a 1 to any bit will clear the error. This register is only reset by the external power-on reset.
ERROR_INT_MASK_1	RO	Error register 1 interrupt mask register. This is a read-only location and can be altered through the corresponding interrupt enable or disable registers.
ERROR_INT_EN_1	WO	Error register 1 interrupt enable register. A write of 1 to this location will unmask the interrupt.
ERROR_INT_DIS_1	WO	Error register 1 interrupt disable register. A write of 1 to this location will mask the interrupt.
ERROR_STATUS_2	Mixed	Error status register 2. If any of the bits in this register are set to 2, it signifies an error within the system. Writing a 1 to any bit will clear the error. This register is only reset by the external power-on reset.
ERROR_INT_MASK_2	RO	Error register 2 interrupt mask register. This is a read-only location and can be altered through the corresponding interrupt enable or disable registers.
ERROR_INT_EN_2	WO	Error register 2 interrupt enable register. A write of 1 to this location will unmask the interrupt.
ERROR_INT_DIS_2	WO	Error register 2 interrupt disable register. A write of 1 to this location will mask the interrupt.
ERROR_POR_MASK_1	RO	Error register 1 power-on reset mask register. This is a read-only location and can be altered through the corresponding power-on reset enable or disable registers.
ERROR_POR_EN_1	WO	Error register 1 power-on reset enable register. A write of 1 to this location will unmask the interrupt.
ERROR_POR_DIS_1	WO	Error register 1 power-on reset disable register. A write of 1 to this location will mask the generation of power-on reset.
ERROR_POR_MASK_2	RO	Error register 2 power-on reset mask register. This is a read-only location and can be altered through the corresponding power-on reset enable or disable registers.
ERROR_POR_EN_2	WO	Error register 2 power-on reset enable register. A write of 1 to this location will unmask the generation of power-on reset.
ERROR_POR_DIS_2	WO	Error register 2 power-on reset disable register. A write of 1 to this location will mask the generation of power-on reset.
ERROR_SRST_MASK_1	RO	Error register 1 SRST mask register. This is a read-only location and can be altered through the corresponding SRST enable or disable registers.
ERROR_SRST_EN_1	WO	Error register 1 SRST enable register. A write of 1 to this location will unmask the generation of SRST.

Table 6-16: **Global Registers *(Cont'd)***

Register Name	Type	Description
ERROR_SRST_DIS_1	WO	Error register 1 SRST disable register. A write of `1` to this location will mask the generation of SRST.
ERROR_SRST_MASK_2	RO	Error register 2 SRST mask register. This is a read-only location and can be altered through the corresponding SRST enable or disable registers.
ERROR_SRST_EN_2	WO	Error register 2 SRST enable register. A write of `1` to this location will unmask the generation of SRST.
ERROR_SRST_DIS_2	WO	Error register 2 SRST disable register. A write of `1` to this location will mask the generation of SRST.
ERROR_SIG_MASK_1	RO	Error register 1 signal mask register. This is a read-only location and can be altered through the corresponding error signal enable or disable registers. This register is only reset by the external power-on reset.
ERROR_SIG_EN_1	WO	Error register 1 signal enable register. A write of `1` to this location will unmask the assertion of the PS_ERROR_OUT signal on the device.
ERROR_SIG_DIS_1	WO	Error register 1 signal disable register. A write of `1` to this location will mask the assertion of the PS_ERROR_OUT signal on the device.
ERROR_SIG_MASK_2	RO	Error register 2 signal mask register. This is a read-only location and can be altered through the corresponding error signal enable or disable registers. This register is only reset by the external power-on reset.
ERROR_SIG_EN_2	WO	Error register 2 signal enable register. A write of `1` to this location will unmask the assertion of the PS_ERROR_OUT signal on the device.
ERROR_SIG_DIS_2	WO	Error register 2 signal disable register. A write of `1` to this location will mask the assertion of the PS_ERROR_OUT signal on the device.
ERROR_EN_1	RW	Error enable register 1. If any of the bits in this register is `1`, the corresponding error is allowed to be propagated to the error handling logic.
ERROR_EN_2	RW	Error enable register 2. If any of the bits in this register is `1`, the corresponding error is allowed to be propagated to the error handling logic.
AIB_CNTRL	WO	This register is used by the PMU to request functional isolation on the AXI interfaces between the PL and PS by using the AIBs. The register maintains its contents during a system reset. AIBs are only for PS to PL isolation, handled by ISO_AIB {AXI,APB} and can respond to the PS master with a SLVERR.
AIB_STATUS	RO	This register is used by the PMU to check the status of functional isolation by the AIBs on the AXI interfaces between the PL and PS. The register maintains its contents during a system reset.

Table 6-16: **Global Registers *(Cont'd)***

Register Name	Type	Description
GLOBAL_RESET	Mixed	This register contains reset for safety-related blocks.
ROM_VALIDATION_STATUS	RO	This register holds the status of the ROM validation.
ROM_VALIDATION_DIGEST_{0:11}	RO	This register holds word {0:11} of the ROM validation digest.
SAFETY_CHK	RW	Target register for safety applications to check the integrity of interconnect data lines by periodically writing to and reading from these registers.

Table 6-17 lists the I/O registers.

Table 6-17: **I/O Registers**

Register Name	Description
IRQ_MODE	Interrupt mode register.
GPO0	I/O module miscellaneous control register (see Table 6-8).
GPO1	PMU to MIO signals.
GPO2	PMU acknowledgments (see Table 6-10).
GPO3	PMU to PL signals (GPO3).
GPI1[0]	Fault tolerance status register (GPI0).
GPI1	General purpose input register 1 (see Table 6-6).
GPI2	General purpose input register 2 (see Table 6-7).
GPI3	General purpose input from PL to PMU.
IRQ_STATUS	Interrupt status register.
IRQ_PENDING	Interrupt pending register.
IRQ_ENABLE	Interrupt enable register.
IRQ_ACK	Interrupt acknowledge register.
PIT{0:3}_PRELOAD	PIT{0:3} preload register.
PIT{0:3}_COUNTER	PIT{0:3} counter register.
PIT{0:3}_CONTROL	PIT{0:3} control register.
INSTRUCTION_INJECT_ADDR	Instruction injection address (IOModule_1.GPO1).
INSTRUCTION_INJECT	Instruction injection (IOModule_1.GPO2).

MIO Signals

Six GPI1 register bits can be driven by input signals routed through the MIO, as described in Table 6-3 and listed in Table 6-9. If these inputs are not routed through the MIO, then they are driven to 0. Six GPO1 register bits can drive output signals routed through the MIO, as described in Table 6-3 and listed in Table 6-6. All 32 GPI3 register bits are driven by PL input signals. All 32 GPO3 register bits drive PL output signals.

Real Time Clock

Introduction

The real time clock (RTC) unit maintains an accurate time base for system and application software. For high accuracy needs, the RTC also includes calibration circuitry to offset temperature and voltage fluctuations.

The RTC is powered by the VCC_PSAUX or VCC_PSBATT power supply. When the auxiliary supply is available, the RTC uses it to keep the counters active. The RTC automatically switches to the VCC_PSBATT power supply when the auxiliary supply is not available. 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.

Functional Description

RTC Operation

The RTC generates two system interrupt signals to the GICs, the GIC proxy, and the PL once every second and when its 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 among the processors.

Block Diagram

Figure 7-1 shows a system level diagram of the RTC controller. The RTC functionality is divided across three main modules.

- RTC control registers: implemented in the low-power domain (LPD); this module incorporates all of the registers associated with the RTC controller.
- RTC counters module: includes all the counters, calibration logic, and latches used to retain the programmed time and calibration in the battery-powered domain (BPD). It also includes these functions:
  - Interfaces with the crystal oscillator that also operates in the BPD.
  - Maintains the current time in seconds.
  - Contains calibration circuitry that is used to calculate one second with a maximum ppm inaccuracy by using a crystal oscillator with an arbitrary static inaccuracy.
  - Maintains a previously programmed time for read back and calibration by the software.
  - Maintains the control value used by the oscillator and power switching circuitry.
- Crystal oscillator: provides the RTC clock that is implemented with the GPIO. The power is supplied by the RTC counters module.

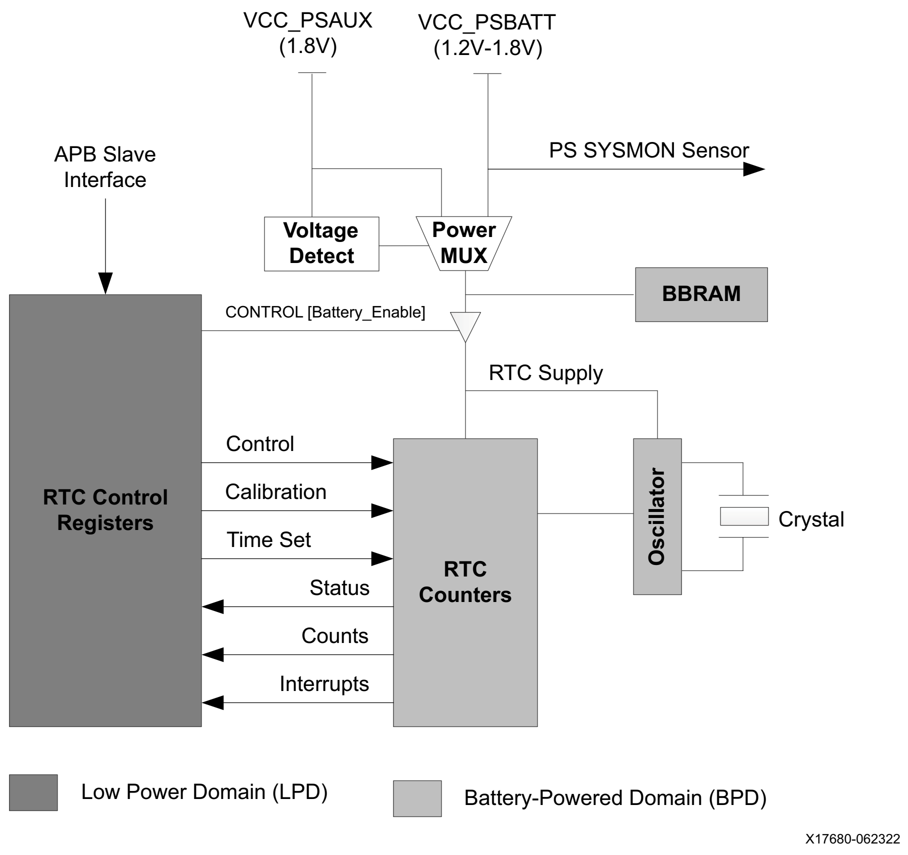


Figure 7-1: **RTC Controller System Block Diagram**

Figure 7-2 shows the functional block diagram of the RTC. The RTC controller is divided into two separate sections.

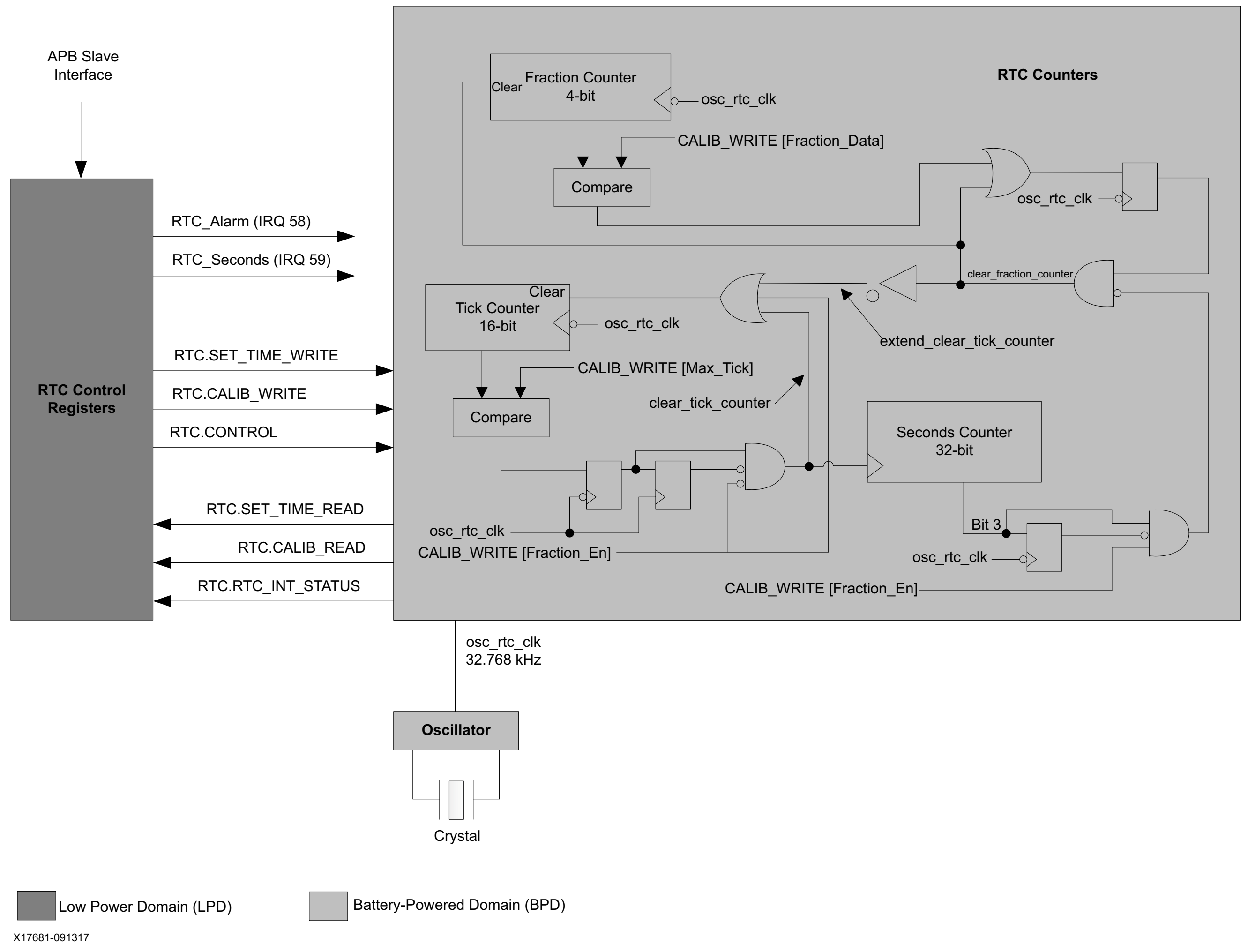


Figure 7-2: **RTC Controller Functional Block Diagram**

Interfaces and Signals

This RTC interfaces to logic in the LPD and includes the following features.

- An APB interface to access the registers within the controller and the RTC counters. This interface is clocked by the LPD_LSBUS_CLK.
- Alarm logic including the alarm register to save the alarm time (in seconds).
- Interrupt status, interrupt mask, interrupt enable, and interrupt disable registers to 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 any registers.

IMPORTANT: *The control register must be programmed every time the LPD is powered on. Otherwise, the value returned by reading the control register can be different from the actual control settings stored in the BPD.*

The SET_TIME_WRITE, CALIB_WRITE, and CURRENT_TIME registers are all implemented within the battery-powered RTC but accessed via the APB interface in the LPD.

The controller logic also includes the ALARM alarm register and alarm generation logic. Whenever the value of the seconds counter in the RTC matches the value that is explicitly loaded into the alarm register, and the alarm interrupt is enabled, the RTC_Alarm system interrupt is generated.

The RTC control registers are programmed via the APB interface in the LPD and retained in the battery-powered domain because it is required for RTC operation. The register set controls functions and is used when performing calibration functions.

Seconds Counter

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). Initially, calculate the current time through the operation system's clock device driver 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 that is used to initialize the seconds counter. After that, the seconds counter is 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 osc_rtc_clk cycle and is captured in the RTC controller's interrupt status register only on a positive-edge transition. The follow-on interrupt from the RTC counters can be used by a clock device driver to calculate the time and date.

The fractional calibration feature, if enabled, 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 calibration register's fractional calibration field.

Calibration

The clear signal used to reset the tick counter can be extended/delayed by logic that operates in conjunction with the fractional calibration value to provide fractional tick adjustment. More specifically, every time the fraction counter asserts its extend clear signal to the tick counter, the clear function to the tick counter stays asserted.

Any inaccuracy in the oscillator can be 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 a total number of ticks between zero and 16. This value is programmed in the fractional calibration segment of the calibration register. After every 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. Therefore, the tick counter increments are delayed by that 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 a 32.768 kHz crystal, 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 will generate 50 extra ticks (or off by
1-9/16 of a tick every second). By increasing the calibration value by one leaves the 9/16 of the tick. Therefore, a crystal's static +50 ppm 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 earlier 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.

By using the fractional calibration feature with a 32.768 kHz oscillator, the static inaccuracy of the RTC is bounded to ≥ 2 ppm, no matter the static inaccuracy of the oscillator. If a higher frequency crystal is used, this number is lowered. For example, by using a 62.5 kHz oscillator, the static inaccuracy is bounded to 1 ppm.

Calibration Algorithm

Assuming that the RTC is programmed at time S, then at time T the RTC is showing value R. Each of these time values is the UNIX Epoch time are represented in terms of seconds with respect to a fixed reference, which for UNIX is 00:00:00 on 1/1/1970.

If C and F are defined as:

C = Value of the calibration register (in the seconds calibration field).

F = Value of the calibration register (in the fractional calibration field).

and fractional calibration is enabled, then the actual crystal oscillator frequency is defined in Equation 7-1.

$$X_f = (R-S) \times [(C+1) + ((F+1)/16)]/(T-S) \qquad \textit{Equation 7-1}$$

where:

$C_{NEW} = \mathrm{Int}(X_f) - 1$

$F_{NEW} = \mathrm{Round}((X_f - \mathrm{Int}(X_f))\times 16) - 1$

Dynamic Oscillator Inaccuracy

The frequency characteristic of a crystal depends on the type of crystal. The frequency is normally specified by a parabolic curve centered around 25 °C. A common parabolic coefficient for a 32.768 kHz tuning fork crystal is –0.04 ppm/°C. Therefore, the crystal frequency can be represented as shown in Equation 7-2.

$$f = f_0[1 - (0.04 \text{ x } 10^{-6}) \text{ x } (T-T_0)^2] \qquad \textit{Equation 7-2}$$

For example, a clock built using a regular 32.768 kHz crystal that keeps time at room temperature loses two minutes per year at 10°C above or below room temperature and loses eight minutes per year at 20°C above or below room temperature.

The impact of temperature on the crystal oscillator can be analyzed and tabulated in advance. The example in Table 7-1 analyzes how much the crystal frequency changes with every 10°C of temperature change, and shows the change in the value to program in the calibration and fractional calibration registers. If the system has a mechanism to read the ambient temperature of the crystal, it could access this table and calibrate the RTC accordingly.

Table 7-1: **Impact of Temperature on a Crystal Oscillator**

Temperature (°C)	Frequency (Hz)	Change (PPM)	Change in Fractional Calibration	Change in Calibration
85	32,763.3	–144	5	–5
75	32,764.7	–100	12	–4
65	32,765.9	–64	14	–3
55	32,766.8	–36	13	–2
45	32,767.5	–16	8	–1
35	32,767.9	–4	14	–1
25	32,768.0	0	0	0
15	32,767.9	–4	14	–1

External Clock Crystal and Circuitry

The typical crystal used for the RTC is a 20 ppm, 32.768 kHz crystal (Figure 7-3). 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.

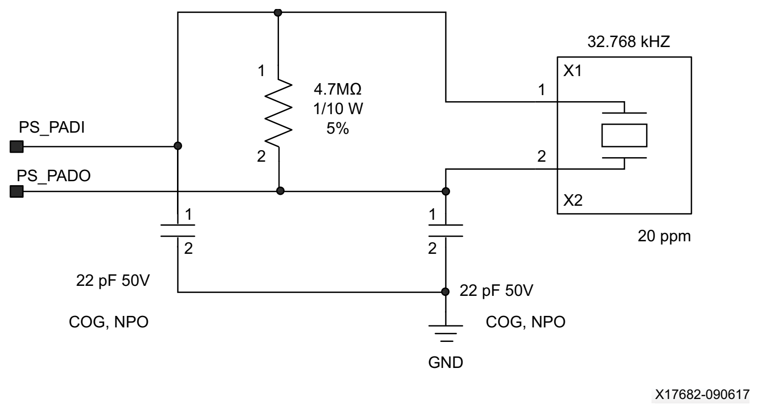


Figure 7-3: **Crystal Circuit Example**

Battery Selection

RECOMMENDED: *This section includes a few recommendations when specifying the battery and battery life.*

1. Although it is common to derating batteries by 25% from their specified capacity quoted at 25°C, a derating factor of at least 50% is recommended by Xilinx.
2. The total power in the battery-powered domain, which includes both RTC and BBRAM, is expected to be 2.5 µA at 50°C, with the I_{BATT} consumed by the BBRAM as specified in *Zynq UltraScale+ MPSoC Data Sheet: DC and AC Switching Characteristics* (DS925) [Ref 2]. Since it is not possible to power off the BBRAM, this leakage must be included when calculating battery life.
3. Power consumption in the RTC and BBRAM is dominated by leakage; therefore, using leakage as the only source of power consumption gives an accurate estimate of the battery life.
4. A leakage requirement of 2.5 µA for the battery-powered domain is specified at 50°C, which is more pessimistic than 25°C. Despite this, the same requirement is used at 25°C. This temperature (25°C) is a typical specification for battery life.
5. Battery consumption in the battery-powered domain is limited to when the PS main supplies are off (including VCC_PSAUX). Since the PS is never completely off (most of the time, although it could be in deep-sleep mode), the battery life (in years) can be divided by the percentage of the time the device is used, to get the number of years the battery should last. Embedded systems are rarely completely off and the need to turn off the device is even less in Zynq UltraScale+ MPSoCs due to the availability of the deep-sleep mode.
6. Xilinx recommends using batteries in the specified range of the VCC_PSBATT voltage (1.2V-1.5V). Using batteries with voltages higher than 1.5V requires a low dropout regulator (LDO) or voltage divider. Although LDOs and voltage dividers cause more current to be drawn from the battery even during PS power up.

Assuming an average current of 2.5 µA is required by the BPD and 50% derating on the battery, a 438 mA-hour battery is required to sustain 10 years of continuous operation (see Equation 7-3).

$$2.5\ \mu A \times 1\ mA/1000\ \mu A \times 10\ years \times 8760\ h/1\ year = 219\ mA\text{-}h/50\%\ derate = 438\ mA\text{-}h \qquad \textit{Equation 7-3}$$

Assuming a 33% system off time (using the battery), the system can operate for 10 years with one 146 mA-h battery using a 50% derating factor. Table 7-2 shows the lifetime of a battery depending upon the battery chosen to power the BPD.

Table 7-2: **Battery Lifetime for BPD (using Example Battery Types)**

Current Drawn by BPD (µA) = 2.5				Derating Factor = 50%			
Battery	**Type**	**Voltage (V)**	**Rated Capacity (mA-h)**	**Derated Capacity (mA-h)**	**% of Time Device is Powered-On (Not using battery)**	**Number of Batteries Used**	**Total Lifetime (years)**
AAA	Alkaline	1.5	1125	562.5	1%	1	26
LR1154	Alkaline	1.5	130	65	70%	1	10
SR1154	Silver oxide	1.5	185	92.5	58%	1	10
SR1131	Silver oxide	1.5	83	41.5	81%	1	10
SR1131	Silver oxide	1.5	83	41.5	62%	2	10
SR1131	Silver oxide	1.5	83	41.5	43%	3	10
SR1142	Silver oxide	1.5	125	62.5	71%	1	10
SR1142	Silver oxide	1.5	125	62.5	43%	2	10
SR754	Silver oxide	1.5	70	35	84%	1	10
SR754	Silver oxide	1.5	70	35	68%	2	10

RTC Register List

The RTC registers are mapped in a 4 KB space starting at `0xFFA6_0000`. The description and offset address for each register is listed in the following table.

Table 7-3: **RTC Register List**

Register Name	Offset	Width	Type	System Reset Value	Description
SET_TIME_WRITE	0x000	32	Write only	0	Program the RTC with the current time.
SET_TIME_READ	0x004	32	Read only	0	Read the last setting done by SET_TIME_WRITE.
CALIB_WRITE	0x008	21	Write only	0	Store the value that is used to generate one second based on the oscillator period.
CALIB_READ	0x00C	21	Read only	0	Read back the calibration value that was programmed in the RTC.
CURRENT_TIME	0x010	32	Read only	0	32-bit timer value in seconds.
ALARM(1)	0x018	32	Read/Write	0	Program the alarm value for the RTC.
RTC_INT_STATUS (1)	0x020	2	Write to clear	0	Raw interrupt status.
RTC_INT_MASK	0x024	2	Read only	11 b	Interrupt mask applied to the status.
RTC_INT_EN	0x028	2	Write only	0	Write a 1 to enable an interrupt.
RTC_INT_DIS	0x02C	2	Write only	0	Write a 1 to disable an interrupt.
ADDR_ERROR	0x030	1	Write to clear	0	Register address decode error interrupt status.
ADDR_ERROR_INT_MASK	0x034	1	Read only	1 b	Register address decode error interrupt mask.
ADDR_ERROR_INT_EN	0x038	1	Write only	0	Write a 1 to enable address decode error interrupt.
ADDR_ERROR_INT_DIS	0x03C	1	Write only	0	Write a 1 to disable address decode error interrupt.
CONTROL	0x040	32	Read/Write	0100_0000 h	Controls the battery enable, clock crystal enable, and APB address decode error.
SAFETY_CHK	0x050	32	Read/Write	0	Safety endpoint connectivity check register.

Notes:

1. Due to the sticky nature of the alarm interrupt status register, clearing the alarm interrupt status register can be done only after the second counter outruns the set alarm value.

Programming Model

The software is responsible for the following.

- Translation and storage of the second, minute, hour, day, month, and year of the current time, based on the value stored in the RTC.
- Initialization of the RTC seconds counter with the current time in seconds that is calculated with respect to a reference point that is also used to calculate the time and date, as specified in the previous bullet.
- Calibration of the RTC based on its past operation periodically, as needed.
- Calculation and storage of the alarm value in the RTC.

Programming Notes

- Program the control register every time the LPD is powered on. The value returned by reading the control register matches with the actual control settings that are stored in the battery powered domain.
- The value that is programmed through the SET_TIME_WRITE register is represented by the seconds counter when the next second is signaled by the RTC. To make the load time of this value deterministic, before writing the current time to the SET_TIME_WRITE register, the value for the calibration should be written to the CALIB_WRITE register. This clears the tick counter and forces the next second to be signaled exactly in one second. In that case, the value that is written to the SET_TIME_WRITE register must be the current time in seconds plus one.
- The value that is programmed through the SET_TIME_WRITE register is loaded in the seconds counter in one cycle (see previous programming bullet). If, for any reason, an application reads the time prior to that elapsed one second, an incorrect value could be read. In that case, after SET_TIME_WRITE register was written, a value of `FFFFh` should be written to the RTC_INT_STATUS register to clear the status of the all RTC interrupts. During a read_time function, the code should read the RTC_INT_STATUS register and if bit `0` is still `0`, it means that one second has not yet elapsed from when the RTC was set and the current time should be read from the SET_TIME_READ register; otherwise, the CURRENT_TIME register is read to report the time.
- The alarm value programmed in the RTC controller represents a specific second within the 136-year range that the RTC is operating. To set an alarm that goes off regularly at a specific time in a day, or any other regular period, the alarm interrupt service routine is expected to set the next time that the alarm is expected to go off in the alarm register.
- The calibration and set_time values are each written and read through different addresses. When a value is written to either of these registers and read back, a sync instruction must be inserted between the write and read operations to ensure that the

value read is the one that was written. Furthermore, the program should read the current time from the CURRENT_TIME register twice through back-to-back reads with a sync instruction between them. If the times match, use that value to ensure a stable value is read by the program.

Programming Sequences

init rtc

1. Write the value `0019_8231h` into the calibration register, CALIB_WRITE.
2. Set the oscillator to crystal and enable the battery switch in the control register, CONTROL.
3. Clear the interrupt status in the interrupt status register, RTC_INT_STATUS.
4. Disable all interrupts in the interrupt disable register, RTC_INT_DIS.

Set Time

1. Program the SET_TIME_WRITE register with the desired date and time value in seconds.

Programming Example – Periodic Alarm

The flowchart in Figure 7-4 shows an example of programming a periodic alarm.

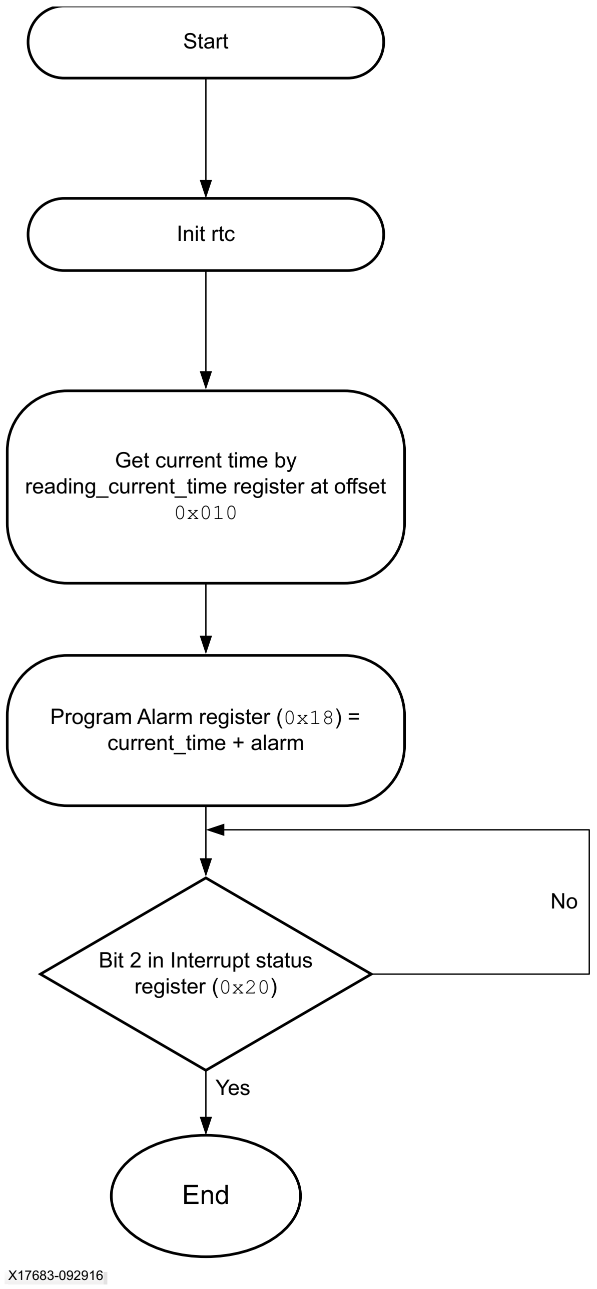

Start
Init rtc
Get current time by reading_current_time register at offset 0x010
Program Alarm register (0x18) = current_time + alarm
Bit 2 in Interrupt status register (0x20)
No
Yes
End
X17683-092916

Figure 7-4: **RTC Periodic Alarm Programming Example Flowchart**

Chapter 8

Functional Safety

Introduction

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. Two types of faults can lead to system failure and result in a violation of the functional safety goals:

- Systematic faults
- Random faults

Systematic faults arise from errors in 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. Mitigation of systematic faults is achieved by robust best practices and processes defined by safety standards.

Random faults are inherent due to silicon aging or environmental conditions, and so on. Safety standards focus on detecting and managing random faults. Some of the causes of random faults include the following.

- Permanent hardware faults, for example, stuck-at faults due to aging silicon
- Temporary hardware faults, for example, corruption of RAM data due to a single-event-upset (SEU)

The Zynq® UltraScale+™ device's LPD, FPD & PL domains were assessed and certified for use in ISO 26262:2018 and IEC 61508:2010 based applications.

The following table summarizes the ISO 26262:2018 functional safety integrity levels of ZU+ power domains:

Table 8-1: **ISO 26262:2018 Functional Safety Integrity Addressed by Zynq UltraScale+**

	Systematic Integrity	Random HW Fault Integrity
LPD	ASIL C	ASIL C
FPD	ASIL B	ASIL B
PL	ASIL B	User Responsibility

The following table summarizes the IEC61508:2010 functional safety integrity levels of ZU+ power domains:

Table 8-2: **IEC61508:2010 Functional Safety Integrity Addressed by Zynq UltraScale+**

	Systematic Integrity	Random HW Fault Integrity
LPD	SC3	SIL2 or SIL3 HFT=1 with PS & PL
PL	SC2	SIL2 or SIL3 HFT=1 with PS

In support of the functional safety application development, Xilinx provides following solution elements:

- Device assessment report and certificate
- Safety manual
- FMEDA tool and report with option to customize application of diagnostics
- Software test libraries (STL)

Apart from the Zynq UltraScale+ device related safety artifacts, Xilinx also provides certified safety solutions, including:

- Vivado Design Suite
- MicroBlaze compiler
- Platform management firmware
- Design methodologies and tools to support functional safety applications in PL

Note: Contact your Xilinx representative for more information on the Xilinx functional safety solutions and accessing the functional safety lounge.

Safety Features overview

This section presents an overview of the safety mechanisms implemented in the Zynq UltraScale+ MPSoC. See the *Zynq UltraScale+ MPSoC Safety Manual* (UG1226) available in the functional safety lounge for further details including recommendations to address ASIL/SIL requirements.

Single Point Fault Detection Measures

- ECC protection for OCM, PMU-RAM, CSU-RAM, and 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 CSU BootROM contents at every boot
- Lockstep and redundancy covers R5F
  - R5F lockstep with physical and temporal diversity
  - Redundant logic in critical control logic such as R5F lockstep checkers
- PMU and CSU implemented with redundancy
  - TMR (triple module redundant) processor cores with physical diversity
  - Triple redundant flip-flops for critical control bits such as security state
- XMPU and XPPU protect memory space
- Watch-dog timers
  - Watchdog timers provided in LPD and FPD
  - LPD watchdog timers for RPU and PMU

Common Cause Failure Measures

- System monitoring
  - Voltages monitoring
  - Temperature monitoring
  - Clock frequency monitoring
- Error management
  - Error management is handled and implemented within the PMU
  - Errors are signaled as interrupts and mirrored to PL
  - All hardware and software errors captured in ERROR_STATUS_1
  - ERROR_STATUS_2 registers are visible to the PMU, RPU, APU, and PL
- Monitoring of activation of common cause failures (CCF) by PMU
  - MBIST, SCAN, reset, power control
- Hang protection
  - Cleanup of outstanding transactions under partial reset

- Meta-stability errors
  - PS uses redundant flip-flops in selected clock crossings
- Aging errors
  - Large on-chip variation (OCV) margin to account for aging effects

Latent Fault Measures

- 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 PMU and CSU program RAMs) can be tested on demand during execution
- The functionality and status of TCM, OCM, PMU RAM, R5F lockstep, PMU, XMPU, XPPU, clocks, voltages, and temperatures can be tested and evaluated through dedicated STL's

Isolation Measures

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

Additional Measures

Safety features for the non-LPD Zynq UltraScale+ MPSoC components include:

- DDR interface supports ECC for 32-bit and 64-bit words
  - Double error detection
  - Single error correction
- ECC support for APU L2, L1-D memories
- Parity support for APU L1-I memories
- QOS management

    - QOS controls on masters
    - QOS management in PS AXI
    - QOS management in PS DDR controller
- PL or multi APU cores can provide redundant processing
- All FPD memories can be tested at full processing speed during boot by MBIST
- 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 error

System Monitors

Introduction

Each system monitor measures voltage and temperature to provide information and alarms to other parts of the system including the PMU, RPU, and APU processors. There are two instances of the SYSMONE4 architecture—the PL SYSMON advanced primitive and the PS SYSMON unit. The basic functionality of these units are the same. Table 9-1 lists the differences between the two units.

The PL SYSMON monitors the die temperature in the PL and several internal PL and PS power supply nodes. The PL SYSMON can also monitor up to 17 external analog channels. The PL SYSMON operates using the VCCAUX and VCCADC power supplies. Additional power supplies including VCCINT are required to access the PL SYSMON from the PS. The PL SYSMON can be configured by the PS using the PLSYSMON register set. The control and configuration registers of the PL SYSMON can also be accessed via JTAG, I2C, and DRP interfaces within the PL domain. The availability of each access method depends on the configuration of the PL. See Register Access via APB Slave Interface for more details.

The PS SYSMON is located in the PS LPD and depends on the VCC_PSAUX and VCC_PSADC power supplies. The PS SYSMON monitors two temperature points and several fixed voltage nodes. The PS SYSMON is controlled by the PSSYSMON and AMS register sets that can be accessed by an AXI interconnect master.

This chapter provides an introduction to the SYSMON units, including measurement points for each unit and system access interfaces.

The detailed functional aspects of the SYSMONE4 architecture are described in *UltraScale Architecture System Monitor User Guide* (UG580) [Ref 6]. This guide provides general and detailed descriptions and should be used in conjunction with this chapter. In the *UltraScale Architecture System Monitor User Guide*, the core functionality of the PS and PL SYSMON units are compatible with the SYSMONE4 architecture. The SYSMON has a dedicated interface to the PS and can interface to the PL fabric. The PS SYSMON has a simple register access path and a fixed set of analog sensor channels Otherwise, the programming and functionality are the same in both units.

Features

The SYSMONE4 architecture provides the following features.

- Voltage sensing: PS/PL internal, PS I/O banks, and PL inputs from external supplies.
- Temperature sensing channels in the LPD (RPU), FPD (APU), and the PL.
- Single channel read with alarms (measurement, minimum/maximum results since last unit reset).
- Sequencer channel reads with full control on a per channel basis.
  - Minimum/maximum result, average result, quiet (long) acquisition, and low-rate sampling.
- 16-bit conversion result with 10-bit ADC accuracy, ±1 bit.
  - Xilinx provides six LSBs to minimize quantization effects and to improve resolution.
- 1.25V ADC voltage reference (internal in PS SYSMON, internal or external in PL SYSMON).
- PL SYSMON channel input sampling.
  - Internal voltage nodes are unipolar unsigned 0 to 3V or 6V.
  - External voltage nodes can be unipolar unsigned 0 to 1V or bipolar ±0.5V range.
- PS SYSMON channel input sampling is unipolar unsigned 0 to 3V (6V for PS I/O banks).
- Two power modes: full operation and sleep.
- Register programmed via multiple hardware paths.
  - PL SYSMON: PL JTAG, TAP controller, I2C, APB DRP, and PL DRP.
  - PS SYSMON: APB interface for any AXI system master.

Unit Architectures

The PL and PS SYSMON units are functionally very similar. The differences are summarized in Table 9-1 and include conversion rates, attached sensor channels, and programming access methods. The alarms from each unit are routed to the interrupt registers in the AMS register set.

Sensor Channels

The PS SYSMON channels are all unipolar. There are several voltage nodes and two temperature points. The PL SYSMON channels are unipolar for internal nodes and either unipolar or bipolar for external nodes. The PL SYSMON unit has one local temperature point.

Alarms

Alarms occur when voltage measurements exceed an upper or lower threshold. Thermal management software uses temperature monitoring to control system cooling. The temperature channels have an optional hysteresis function that uses a lower threshold value to indicate when an alarm is deasserted to simplify the control of cooling strategies.

IMPORTANT: *The over temperature (OT) limits and associated alarm are typically set at or close to the maximum recommended operating temperature of the device. The OT alarm is generally used to trigger an immediate but controlled shutdown of the equipment before erroneous operation or permanent damage occurs.*

Block Diagrams

Figure 9-1 shows the block diagram for the PS SYSMON and the PL SYSMON.

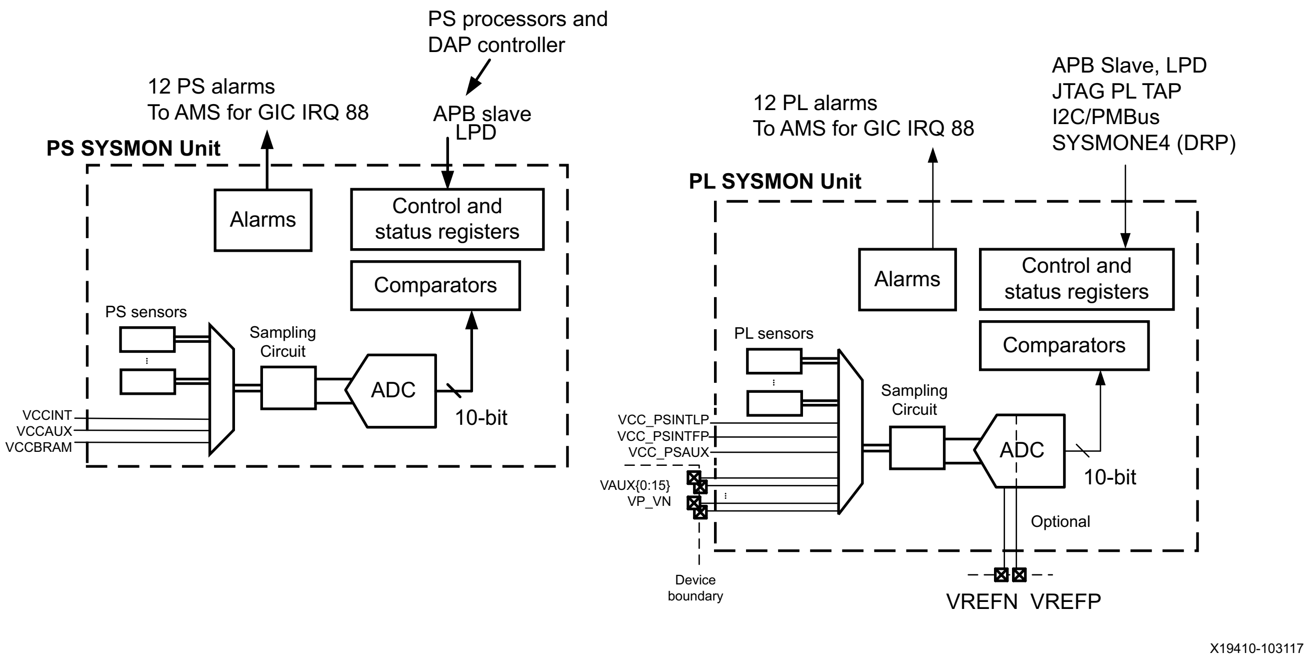


Figure 9-1: **PS SYSMON and PL SYSMON Block Diagram**

PL SYSMON

The PL SYSMON unit monitors voltage nodes within the PL, including several standard power supplies plus four user-defined voltage nodes, VUser{0:3}. The PL SYSMON can also measure up to 16 auxiliary analog inputs and the VP_VN dedicated input. Internal nodes are measured with sampling circuits that generate a 0 to 3V or 6V range. The external auxiliary inputs, VAUX{P, N}{0:15}, are routed through the analog wires in the PL to analog pins. If PL SYSMON is not instantiated, the VAUX pins are routed to the analog pins of PL bank 66. When PL SYSMON is instantiated, the bitstream must define the analog wire connections.

These channels are measured with a unipolar unsigned 0 to 1V range or a bipolar signed range of ±0.5V. The PL SYSMON also measures a few of the voltage nodes located in the PS. The PL temperature sensor diode is physically located in the PL SYSMON. The ADC voltage reference is selectable between an internal reference and the external pins VREFP and VREFN.

PS SYSMON

The PS SYSMON monitors several internal voltage nodes plus two on-chip temperature sensors. The voltage nodes are in the LPD and FPD. They include voltage nodes for internal and I/O buffers. All voltage measurements are unipolar. The internal nodes are measured with a 0 to 3V or 6V range. The PS has two temperature sensors. One is physically located in the PS SYSMON near the RPU. The second, remote sensor is located in the FPD near the APU. The ADC always uses an internally generated voltage reference.

Comparison of PS SYSMON and PL SYSMON

The notable differences between the PS SYSMON and the PL SYSMON are the programming bus interfaces, sampling rates, and analog input signal sources. The differences are listed in Table 9-1.

Table 9-1: **PS SYSMON and PL SYSMON Comparison**

Function	PS SYSMON	PL SYSMON
Sampling frequency	1 M samples per second.	200K samples per second.
Voltage reference	Internal.	Internal or external (VREFP, VREFN).
Programming interfaces	APB on AXI interconnect. Includes DAP controller via JTAG.	APB/AXI interconnect. DRP (PL configuration required). I2C/PMBus. PL JTAG controller.
Power domain	LPD.	PLPD.
Temperature sensors with OT	Temp_LPD near the RPU MPCore. Temp_FPD near the APU MPCore.	Temp_PL near the PL SYSMON unit.
On-chip supply sensors	Three PS internal voltage nodes. Three I/O voltage nodes.	Three PS internal voltage nodes. Three PL internal voltage nodes. Four PL internal VUSER nodes.
PL external sensor channels	None.	16 signal pairs; VAUXP, and VAUXN(2). One set of dedicated pins, VP and VN(2).
PL user inputs	None.	Four, full featured.
Event driven trigger	AMS.PS_SYSMON_CONTROL_STATUS	CONVST start signal input.(1)
EOS, EOC	AMS.ISR_1 [eos], [eoc] interrupts.	EOS, EOC signals to PL fabric.(1)

Table 9-1: **PS SYSMON and PL SYSMON Comparison *(Cont'd)***

Function	PS SYSMON	PL SYSMON
Reset (see Reset Sources)	POR, write to VP_VN register, AMS.PS_SYSMON_CONTROL_STATUS.	RESET pin, write to VP_VN register.

Notes:

1. This function requires the SYSMONE4 primitive to be instantiated by the bitstream for the PL. The instantiation disconnects the PL SYSMON unit from the PS, and provides a slave bus interface and the other system signals for the PL fabric.
2. The VAUXP/VAUXN, and VN/VP analog signals are connected to device pins by instantiating the SYSMONE4 primitive.

On-chip Thermal Diode

There is an on-chip thermal diode connected to device pins DXP and DXN. There are no internal connections between this thermal diode and the SYSMON units. The on-chip thermal diode can be monitored by an external device connected to the DXP and DXN device pins.

Safety Considerations

The SYSMON units contribute to the safe operation of a product. Appropriately configured, the SYSMON units can independently monitor supply voltages and die temperatures and alert the system to deviations beyond limits (thresholds) that can be defined as required.

IMPORTANT: *To ensure that the SYSMON units enhance product safety, it is important to consider the strategies deployed in response to each potential alarm so that the reactions are consistent with the safety targets of the product. For safety applications, contact Xilinx Sales for technical support.*

Set Operating Limits

By defining limits consistent with the maximum and minimum recommended values in the data sheet, alarms (and interrupts) can be generated when the device is used outside of specified limits, which could compromise operation. By defining limits at levels that allow some margin within the maximum and minimum recommended values in the data sheet, a preliminary warning can be provided for a potential issue while the device is still operating within specified limits.

Monitor Supply Voltages

Direct measurement of voltages reaching the silicon die enable the SYSMON unit to confirm the integrity of the external power supplies and the associated power distribution networks. For example, the SYSMON unit can detect voltage drops caused by the combination of high current demand and resistance in the path between the power supply and the silicon die. Similarly, the monitoring and recording of the supply voltages over a longer time (e.g., hours or years), can reveal drift in the output voltage of a power supply caused by temperature fluctuations or aging. In both cases, the supply voltages measured by the SYSMON unit can be used to fine tune the power supply controllers or report the information for further investigation (e.g., at the next scheduled service of the product).

Monitor Temperature

Each temperature sensor is associated with two sets of limits or thresholds. The regular alarm is ideally suited for controlling temperature using a cooling fan or load reduction (e.g., reducing processing, clock frequency, or shutting down parts of the device). The direct measurement of temperature can reveal the progress made by the cooling strategy and can potentially refine that strategy as a result of what is observed. The OT alarm is normally set at or close to the maximum recommended operating temperature of the device in the data sheet. The generation of an OT alarm usually implies that attempts to control temperature have failed. In this case, the system typically triggers an immediate but controlled shut down of the equipment to prevent erroneous operation or permanent damage.

Safety User Manual

For safety applications, contact Xilinx Sales for technical support.

Functional Description

This section describes the functional units and their register programming model.

Sensor Channels

All sensor channels can be sampled individually or in a sequence that loops once or multiple times. The voltage and temperature sensor channels are listed in Table 9-2 and Table 9-3.

Two Classes of Sensor Channels

There are two classes of sensor channels—basic and full-featured. The full-featured channels record minimum and maximum values and have upper and lower alarm threshold settings. The basic channels have only a measurement register.

Sensor Channel Tables

Table 9-2 and Table 9-3 characterize the register control for each of the channels in the PL and PS SYSMON units. The table headings are described here.

- Channel Name and Description

The channel name with description refers to the temperature points and voltage nodes in the system.

- Measurement, Minimum/Maximum, Alarm U/L Registers

  The offset addresses for the measurement, minimum/maximum, and upper and lower alarm threshold registers are listed in three columns.

- Sequence Channel, Low-rate, and Average Registers

  This column refers to three functions that are supported by three sets of registers. Each channel can be individually selected for the normal or low-rate sequence. Each channel can optionally accumulate an average value instead of the last measurement. The control register names and offset addresses are listed in Table 9-6.

- Alarms, Interrupts, and Errors

  The alarms are routed to the interrupt registers in the AMS register set. They are also OR'd together to generate the SYSMON IRQ signal to the RPU and APU GICs. Alarms are also routed to the PMU global registers as system errors.

- Input Sampling Circuit Type

  For most channels, the sampling circuit type is fixed (temperature or unipolar) and includes all PS SYSMON sensors.

  The VP_VN and VAUX sensor channels in the PL SYSMON unit includes programmable input sampling circuits types and long acquisition time options.

- Alternate Name

  The alternate channel names appear in other documentation and in the standalone device drivers.

PS SYSMON Sensor Channels

The PS SYSMON sensors are controlled by the PSSYSMON register set. The 3-digit offset addresses in Table 9-2 are relative to the base address `0xFFA5_0800`.

Table 9-2: **PS SYSMON Sensor Channels**

Sensor Channel Name	Description	Channel Number	Alarm No.	Register Address Offsets			Sequence Channel, Low-rate, and Average Registers	AMS Interrupt Registers	PMU_Global ERROR_STATUS_1	CSU IOMODULE ISR [(1)]	Input Circuit	Alternate Channel Name
				Measurement	Min/ Max	Alarm U/L						
Temp_LPD	Temperature for RPU MPCore.	0	0	000	080 090	140 150	0	ISR_0 [0]	~	~	Temp	PS_TEMP1
Temp_LPD_OT	LPD over temperature (OT).	~	~	~	~	14C 15C	~	ISR_1 [1]	[4]	[20]	Temp	~
VCC_PSINTLP	LPD power supply.	1	1	004	084 094	144 154	0	ISR_0 [1]	[16]	[22]	3V	SUPPLY1
VCC_PSINTFP	FPD power supply.	2	2	008	088 098	148 158	0	ISR_0 [2]	[17]	[23]	3V	SUPPLY2
VCC_PSAUX	PS auxiliary voltage.	6	3	018	08C 09C	160 170	0	ISR_0 [3]	[18]	[24]	3V	SUPPLY3
VCCO_PSDDR	I/O bank 504: DDR PHY.	13	4	034	0A0 0B0	164 174	0	ISR_0 [4]	[19]	[25]	3V	SUPPLY4
VCCO_PSIO3	I/O bank 503: boot, config, JTAG, error, SRST, POR.	14	5	038	0A4 0B4	168 178	0	ISR_0 [5]	[20]	[27]	6V	SUPPLY5
VCCO_PSIO0	I/O Bank 500: MIO[0:25].	15	6	03C	0A8 0B8	16C 17C	0	ISR_0 [6]	[21]	[26]	6V	SUPPLY6
~	OR of PS alarms in bits [6:0].	~	7	~	~	~	~	ISR_0 [7]	~	~	~	~
VCCO_PSIO1	I/O bank 501: MIO[26:51].	32	8	200	280 2A0	180 1A0	2	ISR_0 [8]	[22]	[26]	6V	SUPPLY7
VCCO_PSIO2	I/O bank 502: MIO[52:77].	33	9	204	284 2A4	184 1A4	2	ISR_0 [9]	[23]	[26]	6V	SUPPLY8
PS_MGTRAVCC	GTR SerDes I/O.	34	10	208	288 2A8	188 1A8	2	ISR_0 [10]	~	[28]	3V	SUPPLY9, VMGTAVCC

Table 9-2: **PS SYSMON Sensor Channels *(Cont'd)***

Sensor Channel Name	Description	Channel Number	Alarm No.	Register Address Offsets			Sequence Channel, Low-rate, and Average Registers	AMS Interrupt Registers	PMU_Global ERROR_STATUS_1	CSU IOMODULE ISR [(1)](#)	Input Circuit	Alternate Channel Name
				Measurement	Min/ Max	Alarm U/L						
PS_MGTRAVTT	GTR SerDes terminators.	35	11	20C	28C 2AC	18C 1AC	2	ISR_0 [11]	~	[28]	3V	SUPPLY10, VMGTAVTT
VCC_PSADC	PS SYSMON ADC circuitry.	36	12	210	290 2B0	190 1B0	2	ISR_0 [12]	~	~	3V	~
Temp_FPD	Temperature for APU MPCore.	37	13	214	294 2B4	194 1B4	2	ISR_0 [13]	~	~	Temp	T_REMOTE, Remote_Temp
Temp_FPD_OT	FPD over temperature (OT).	~	~	~	~	14C 15C	~	ISR_1 [0]	[5]	[21]	Temp	~
~	OR of PS alarms in bits [13:0].	~	15	~	~	~	~	ISR_0 [15]	~	~	~	~

Notes:

1. Three MIO banks are OR'd together for bit [26] and the two GTR supplies are OR'd together for bit [28].
2. The PSIO{1, 2} and the two GTR supplies are mapped to sensors channels {7:10} using the ANALOG_BUS register. This table shows the default mapping, see PS SYSMON Analog_Bus for more information.

PL SYSMON Sensor Channels

The PL SYSMON sensors are controlled by the PLSYSMON register set. The 3-digit offset addresses in Table 9-3 are relative to the base address `0xFFA5_0C00`.

Table 9-3: **PL SYSMON Sensor Channels**

Sensor Channel Name	Description	Channel Number	Alarm No.	Register Address Offsets			Sequence Channel, Low-rate, and Average Registers	Input Mode and Long Acquisition Time Register	AMS Interrupt Registers	Input Circuit	Alternate Channel Name
				Measurement	Min/ Max	Alarm U/L					
Temp_PL	SYSMON temperature.	0	0	000	080 090	140 150	0	~	ISR_0 [16]	Temp	PL_TEMP
Temp_PL_OT	Over temperature (OT).	~	~	~	~	14C 15C	~	~	ISR_1 [2]	Temp	~
VCCINT	PL internal voltage.	1	1	004	084 094	144 154	0	~	ISR_0 [17]	3V	SUPPLY1
VCCAUX	PL auxiliary voltage.	2	2	008	088 098	148 158	0	~	ISR_0 [18]	3V	SUPPLY2
VP_VN	Analog input pins.	3	3	00C	~	~	0	0		Uni 1V, or Bi ±0.5V	VP/VN
VREFP	ADC positive V ref.	4	4	010	~	~	0	~	ISR_0 [20]	3V	~
VREFN	ADC negative V ref.	5	5	014	~	~	0	~	ISR_0 [21]	3V	~
VCCBRAM	PL block RAM voltage node.	3	3	018	08C 09C	160 170	0	~	ISR_0 [19]	3V	SUPPLY3
~	OR of alarm bits [22:16].	~	7	~	~	~	~	~	ISR_0 [23]	~	~
VCC_PSINTLP	LPD power supply.	13	~	034	0A0 0B0	164 174	0	~	Use PS SYSMON unit.	3V	SUPPLY4
VCC_PSINTFP	FPD power supply.	14	~	038	0A4 0B4	168 178	0	~	Use PS SYSMON unit.	3V	SUPPLY5
VCC_PSAUX	PS auxiliary voltage.	15	~	03C	0A8 0B8	16C 17C	0	~	Use PS SYSMON unit.	3V	SUPPLY6

VAUXP{0:15}
VAUXN{0:15}
	Analog wires in PL fabric.(1)	16 -
		31
			~	040 to
				07C
					~	~	1	1	~	Uni 1V, or
										Bi ±0.5V
											~

Table 9-3: **PL SYSMON Sensor Channels *(Cont'd)***

Sensor Channel Name	Description	Channel Number	Alarm No.	Register Address Offsets			Sequence Channel, Low-rate, and Average Registers	Input Mode and Long Acquisition Time Register	AMS Interrupt Registers	Input Circuit	Alternate Channel Name
				Measurement	Min/ Max	Alarm U/L					
VUser0	Analog wires in PL fabric.	32	8	`200`	`280` `2A0`	`180` `1A0`	2	~	ISR_0 [24]	3V or 6V	SUPPLY7
VUser1	Analog wires in PL fabric.	33	9	`204`	`284` `2A4`	`184` `1A4`	2	~	ISR_0 [25]	3V or 6V	SUPPLY8
VUser2	Analog wires in PL fabric.	34	10	`208`	`288` `2A8`	`188` `1A8`	2	~	ISR_0 [26]	3V or 6V	SUPPLY9
VUser3	Analog wires in PL fabric.	35	11	`20C`	`28C` `2AC`	`18C` `1AC`	2	~	ISR_0 [27]	3V or 6V	SUPPLY10
~	OR of alarm bits [29:16].	~	15	~	~	~	~	~	ISR_0 [31]	~	~

Notes:

1. The auxiliary channels can be enabled by writing `0001h` to the SUPPLY2 (PL SYSMON) register prior to PL configuration. After PL configuration, the auxiliary channels can be connected to the PL analog wires using the SYSMONE4 instantiation.
2. Base addresses are used only while the PL Sysmon is not instantiated. For instantiation of base address refer to the Vivado block design.

Measurement Registers

The ADCs produce a 16-bit conversion result, and the full 16-bit result (or averaged result) is stored in the 16-bit measurement registers. The specified 10-bit accuracy corresponds to the 10 MSBs (most significant or left-most bits) in the 16-bit ADC conversion result. The unreferenced LSBs can be used to minimize quantization effects or improve resolution through averaging or filtering.

Average Measurements

Channels are individually enabled to either record the last sample, or an average of multiple samples using the SEQ_AVERAGE registers. The number of samples that are averaged are the same for all channels within a SYSMON unit that employs averaging. The average value is calculated using the last 16, 64, or 256 samples as programmed by the CONFIG_REG0 [averaging] bit field. Measurement averaging applies to full feature sensor channels (internal measurements) in both the PL and PS SYSMON units.

If a channel is selected for averaging, then its measurement register is first written when the sample count is reached for that channel, which could take 16, 64, or 256 samples of the channel. The subsequent values written to the measurement register are the average of the most recent 16, 64, or 256 samples.

When averaging is enabled, the end of sequence (EOS) event occurs after the sequencer has completed the selected number of samples. In the PL SYSMON unit, the EOS event is indicated by the assertion of the EOS signal. In the PS SYSMON unit, the EOS event sets the [eos] interrupt bit in the AMS.ISR_0 register.

When averaging is disabled, the EOS event occurs after the first pass of the sensor channels and the results are written to the measurement register and the minimum or maximum registers.

Measurement Registers in AMS

The AMS register set includes several measurement registers that are written to by the PS SYSMON unit using the single-channel mode (sequencer off). These voltage measurements are performed using the unipolar sampling circuit with a 0 to 3V range and do not have alarms or minimum/maximum registers.

The AMS measurement register names for each voltage node are listed in Table 9-4, and are accessed starting at memory location `0xFFA5_0060`.

Table 9-4: **Measurement Registers in AMS Register Set**

Voltage Node	Description	Channel Number	Seq Bit
VCC_PSPLL	System PLLs voltage.	48	0
VCC_PSBATT	Battery voltage.	51	3
VCCINT	PL internal voltage.	54	6
VCCBRAM	PL block RAM voltage.	55	7
VCCAUX	PL VAUX voltage.	56	8
VCC_PSDDR_PLL	PS DDR I/O PLLs {0:5} voltage.	57	9
VCC_DDRPHY_REF	PS DDR I/O buffer voltage.	58	10
VCC_PSINTFP_DDR	PS DDR controller voltage.	63	15

Notes:

1. PSSYSMON.SEQ_BASIC_MONITOR_CHANNEL0 register for low-rate sequence assignment.

PS SYSMON Analog_Bus

The sensor channels {7:10} are routed through four multiplexers controlled by the PSSYSMON.ANALOG_BUS register. The sensor channels connect to the following voltage nodes.

- VCCO_PSIO1
- VCCO_PSIO2
- PS_MGTRAVCC
- PS_MGTRAVTT

The recommended value of the ANALOG_BUS register is `3210h`; this value is assumed for the definition of these sensor channels in Table 9-2 and in other places. The register is programmed to `3201h` by the PMU ROM pre-boot code and should be reprogrammed by the FSBL or other software code.

Note: Once the value is reprogrammed by the FSBL it should be changed.

RECOMMENDED: *The ANALOG_BUS register should be set to* `3210h` *in the FSBL code. If the recommended value is not used, the sensor channels {7:10} are remapped and might not be compatible with the system software.*

The functionality of the ANALOG_BUS register is illustrated in Figure 9-2.

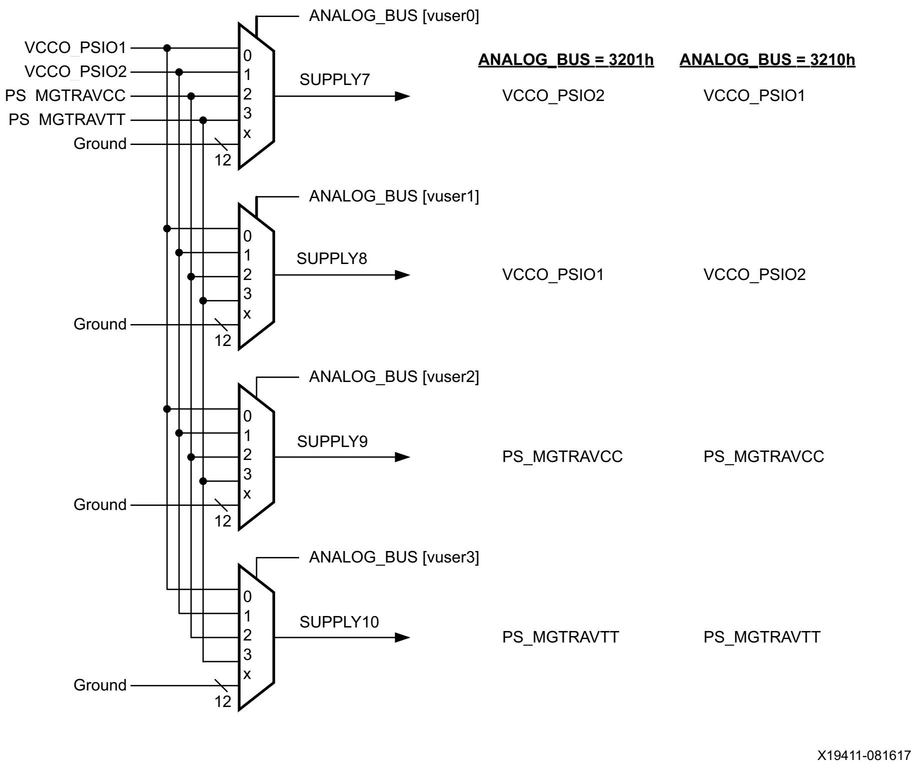


Figure 9-2: **PS SYSMON Unit Sensor Channels {7:10}**

Temperature Sensors

The temperature sensors are located in the following three areas of the chip.

- The LPD near the RPU and measured by the PS SYSMON unit.
- The FPD near the APU and measured by the PS SYSMON unit.
- The PL area near the PL SYSMON unit and measured by the PL SYSMON unit.

The ADC result is processed through a temperature sensor translation function to provide a meaningful temperature value. The temperature sensor translation function and other details of the SYSMON units are explained *UltraScale Architecture System Monitor User Guide* (UG580) [Ref 6].

Minimum and Maximum Result Registers

The minimum and maximum values are recorded for fully-featured channels. POR and system resets affects all minimum value registers set to `FFFFh` and all maximum value registers set to `0000h`.

Each new measurement is compared to the contents of its maximum and minimum registers. If the measured value exceeds the minimum/maximum extreme, the appropriate register is updated. This checking is done every time a result is written to the measurement register. The minimum and maximum result feature applies to both the PL and PS SYSMON units.

Sequencer Channel Control

Low-Rate Sampling

Channels that are less time critical can be sampled less often to free up ADC bandwidth for other channels. The SEQ_LOW_RATE_CHANNEL registers are used to select which channels are sampled at a lower rate than the others. The CONFIG_REG4 [sequence_rate] specifies how often the low-rate channels are sampled (every 4th, 16th, and 64th sequence). Selecting a channel in both the SEQ_CHANNEL and SEQ_LOW_RATE_CHANNEL registers causes the SEQ_CHANNEL to prevail, but this situation is not recommended. The low-rate sampling feature applies to both the PL and PS SYSMON units.

Long Acquisition Time

In auto-sequencer mode, the SYSMON unit waits four ADC clock cycles before sampling the analog input. In the PL SYSMON, the settling time can be extended to ten clock cycles for the external voltage measurements by setting bits in the SEQ_ACQ registers on a per channel basis. The long acquisition time extends the settling time after being selected by the analog multiplexer. The long acquisition time feature applies to the VP_VN, VUSER{0:3}, and VAUX{P, N}{0:15} channels of the PL SYSMON. The PS SYSMON does not use the long acquisition time feature.

Input Sampling Circuits

All PS SYSMON sensor channels and the internal PL SYSMON channels use a unipolar sampling circuit. The PL SYSMON external sensor channels can use unipolar or bipolar mode, selectable on a per channel basis. Operational details of the sampling circuit are discussed in the *UltraScale Architecture System Monitor User Guide* (UG580) [Ref 6]. PS SYSMON uses only unipolar mode. PL SYSMON uses unipolar mode on its internal sensor channels, and unipolar or bipolar mode on its external sensor channels.

Unipolar Mode

Unipolar mode voltage measurement is an unsigned 16-bit value representing the voltage range listed in Table 9-2 and Table 9-3.

Bipolar Mode

Bipolar mode voltage measurement is a signed, twos complement 16-bit value representing a ±0.5V sample for VP_VN and 16x VAUX channels (PL SYSMON unit).

Sensor Alarm Types

Interrupts and system errors are generated by voltage and temperature alarms. When a measurement exceeds a programmed limit, an alarm is asserted unless disabled by a CONFIG_REG register. An alarm can assert an interrupt or a system error. Several PS SYSMON alarms go to interrupt controllers in the PMU and CSU. The IRQ 88 interrupt control and status registers are in the AMS register set. The routing of the alarm signals are shown in Figure 9-4. Each fully-featured sensor channel asserts an alarm signal when a measured value exceeds a software programmed value in its associated upper or lower threshold register.

Voltage Alarms

The voltage nodes are measured and monitored for low and high conditions. The alarm is asserted when the measurement register is outside the upper or lower threshold register settings.

Normal Temperature Alarms

Each temperature channel is monitored and can generate an alarm when the corresponding normal temperature threshold is exceeded. The normal alarm signals can generate an interrupt that can be used by software to implement a thermal management scheme such as the control of a cooling fan. Each monitor has a hysteresis mode to reset the alarm. The upper and lower temperature alarm thresholds can be used to generate a tamper event in the CSU.

Upper Alarm Threshold

The PS SYSMON unit has two normal temperature upper alarm threshold registers; ALARM_UPPER_TEMP_LPD and ALARM_UPPER_TEMP_FPD. The PL SYSMON unit has one normal temperature upper alarm threshold register; ALARM_UPPER_TEMP_PL. When a temperature exceeds a threshold and the corresponding alarm is not disabled, the normal temperature alarm propagates to the system as shown in Figure 9-4.

Lower Alarm Threshold

The PS SYSMON unit has two normal temperature lower alarm threshold registers; ALARM_LOWER_TEMP_LPD and ALARM_LOWER_TEMP_FPD. The PL SYSMON unit has one normal temperature lower alarm threshold register; ALARM_LOWER_TEMP_PL.

The temperature alarm is set when the measured temperature exceeds the value in the ALARM_UPPER_TEMP_xx register. The temperature at which the alarm is cleared depends on the [threshold_mode] bit setting in the lower alarm threshold register. The alarm deasserts either immediately after the temperature drops below the upper limit, or when hysteresis is enabled, when the temperature reaches the lower limit.

All of these registers define a lower temperature threshold that can be used in one of the following ways.

- Normal Lower Threshold Mode

  In normal mode, the alarm is asserted when the temperature is below the lower threshold. When the temperature returns to between the upper and lower thresholds, the alarm is deasserted. This mode is selected when bit [0] of the lower threshold register is set to 1.

- Hysteresis Lower Threshold Mode

  In hysteresis mode, the alarm is asserted when the temperature goes above the upper threshold and deasserts when it falls below the lower threshold. The hysteresis mode is selected when bit [0] of the lower threshold register is set to 0.

Over Temperature Alarms

When the temperature point exceeds the OT upper threshold, the OT alarm is asserted. The OT alarm deasserts when the temperature falls below the OT lower threshold.

The default OT upper threshold value corresponds to a temperature of 122°C. The default value is used after reset and when the ALARM_UPPER_OT [3:0] bits = `0h`. The default lower threshold value is 67°C.

The software can define its own upper and lower OT thresholds. Setting ALARM_UPPER_OT [3:0] to `3h` overrides the default upper and lower thresholds with the temperature values defined by ALARM_UPPER_OT [15:4] and ALARM_LOWER_OT [15:4]. When overriding the default OT thresholds, ALARM_LOWER_OT [0] defines hysteresis mode when 0 or normal mode when 1.

For the PS SYSMON unit, the upper and lower OT threshold values are shared between the LPD and FPD over-temperature alarms. The register value for temperature translation algorithms is described in the *UltraScale Architecture System Monitor User Guide* (UG580) [Ref 6].

Writing `0000h` to the ALARM_UPPER_OT register restores the default upper and lower OT temperatures and hysteresis mode. In the PS SYSMON unit, the LPD and FPD sensor channels both use the same threshold register, ALARM_UPPER_OT. The OT alarm signal causes a system error that is received by the PMU, reset units, and the CSU.

Alarm Interrupt Control

An alarm generates an interrupt if the alarm generation is enabled within the SYSMON and the corresponding interrupt is enabled (unmasked). The setting and clearing of the alarm and interrupt signals are illustrated in Figure 9-3. The function is described with and without hysteresis enabled.

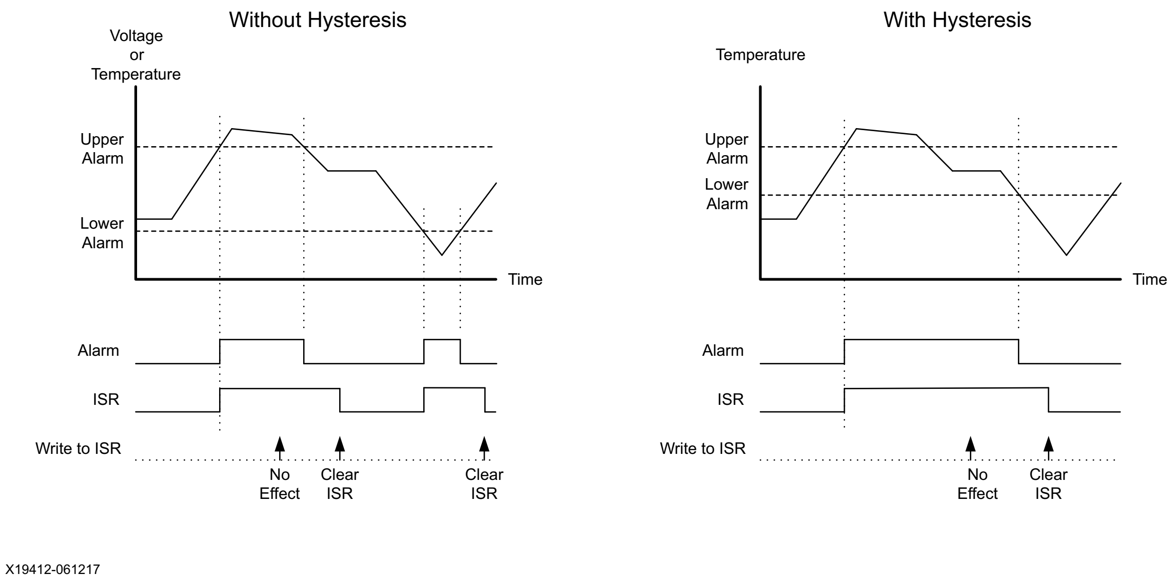


Figure 9-3: **Alarm Interrupt Function**

Alarm Signal Routing

All alarm signals from the PS SYSMON and PL SYSMON are routed to the interrupt mechanism within the PS. By default, both monitors generate alarm signals, however, individual alarm signals can be disabled by setting bits in the CONFIG_REG1 and CONFIG_REG3 registers of each monitor. The alarm signals from the PL SYSMON unit are always routed to the PS (independent of instantiating the SYSMONE4 primitive). However, the PL SYSMON alarm signals do pass through the PCAP isolation wall so they are subject to isolation (alarms appear inactive to the PS when isolation is active).

The alarm interrupts from the PS and PL SYSMON units can be observed in the ISR_{0:1} registers of the AMS. Each alarm interrupt can be individually masked by the IMR_{0:1} registers. Unmasked alarm signals assert the IRQ 88 interrupt to the RPU, APU, and proxy GICs. The routing of the SYSMON alarms to the PMU, CSU, and PL is shown in Figure 9-4. The interrupt control and status registers are described in the Sensor Alarm Types and Interrupts section.

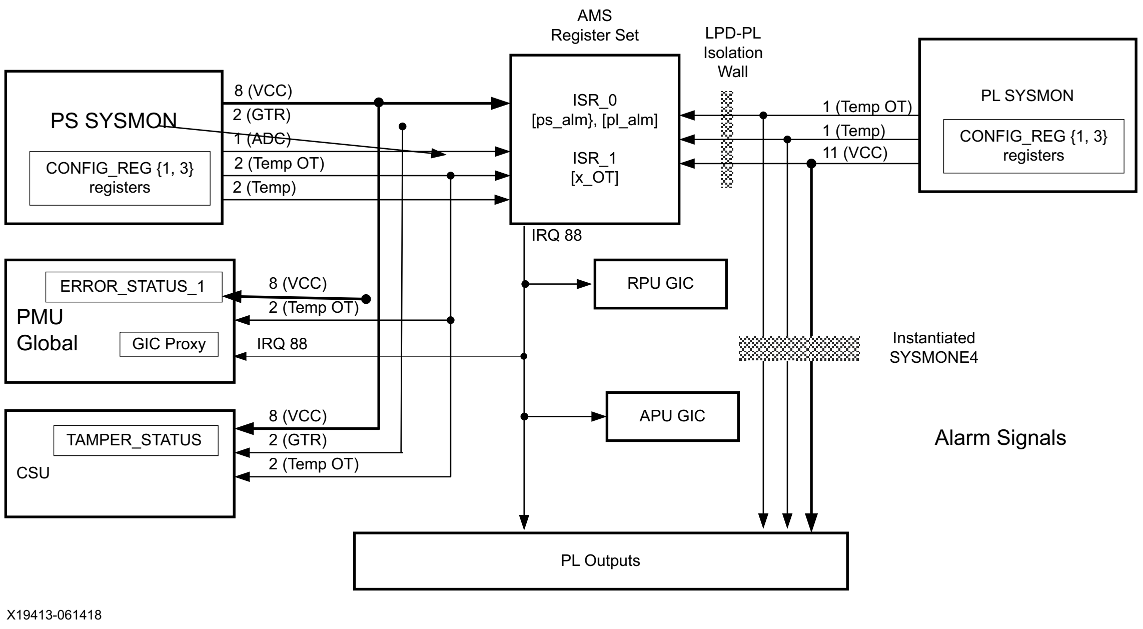


Figure 9-4: **Alarm Signal Routing**

The PMU processor receives ten alarm signals from the PS SYSMON unit that are controlled by the CONFIG_REG disables.

- LPD (RPU) OT.
- FPD (APU) OT.
- Eight power supplies out of range (see Table 9-2 for a list). ERROR_STATUS_1 [16:23].
- Two power supplies out of range (see Table 9-4 for a list).

The PL SYSMON unit can be configured to initiate an automatic shutdown procedure. For details, see the "Thermal Management" section in the *UltraScale Architecture System Monitor User Guide* (UG580) [Ref 6].

GIC IRQ 88 is an OR of the 25 alarm signals from the PS and PL SYSMON units after the CONFIG_REG alarm disables and the interrupt masking function controlled by the AMS.ISR_{0, 1} registers.

Interrupts

Each controller reports the following three events.

- End of conversion (EOC) event, useful for single channel mode.
- End of sequence (EOS) event.
- Register address decode error for AMS, PLSYSMON, and PSSYSMON register sets.

The PS SYSMON unit reports all three events as interrupt signals in AMS.ISR_1 [3, 4, 29].

The PL SYSMON unit drives the EOC and EOS signals to PL fabric outputs. The address decode error sets the AMS.ISR_1 [30] bit.

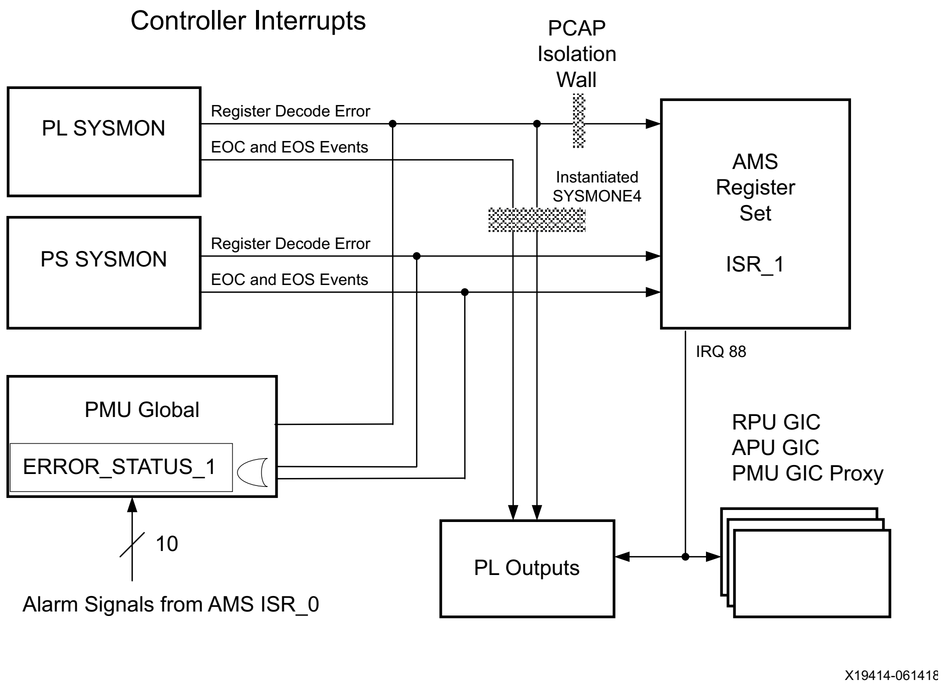


Figure 9-5: **Controller Interrupts**

End of Sequence Event

When a sequencer has completed a loop through all selected channels, it signals the end of sequence (EOS) event. The EOS signal can be generated after the normal rate sequence is done, after the low-rate sampling is done, or after either is done by programming the CONFIG_REG4 [low_rate_eos] bit. For the PL SYSMON, this can be an output on the SYSMONE4 primitive (requires it to be instantiated). For the PS SYSMON, the AMS.ISR_1 [eos] bit asserts and generates an interrupt if enabled.

End of Conversion (EOC) Event

Each time an ADC completes a measurement, it signals the end of conversion (EOC) event. For the PL SYSMON, this can be an output on the SYSMONE4 primitive (requires it to be instantiated). For the PS SYSMON, the AMS.ISR_1 [eoc] bit asserts if enabled. EOC is useful for single measurement mode.

Register Address Decode Error (APB)

If the software attempts to access a non-existent register or performs a read or write to a register that does not support that access type, the SYSMON unit detects this and sets a maskable register address decode error interrupt in ISR_1 [29:31] register bits.

Interrupt Control Registers

The SYSMON interrupt controller processes PS and PL alarms, end-of-process events, and register address decode errors.

Each SYSMON unit has configuration registers to disable alarms. If an alarm is disabled when its event occurs, the alarm signal does not assert. The interrupt controllers (AMS registers and PL signals) will not receive an IRQ. Each alarm can be disabled using a configuration register; a disabled alarm is ignored by the system.

Status/Clear

AMS.ISR_{0, 1} are sticky registers that latch a 1 when an enabled (unmasked) alarm signal is detected. These registers are read by software to determine which alarm sensor or sensors caused the interrupt. Software clears a bit by writing a 1 to it. The software must either resolve the cause of the alarm or disable the alarm (mask it) for the ISR register bit to be cleared by writing a 1.

Enable, Disable Mask

An interrupt request signal is asserted if the interrupt is not masked. The following three registers control interrupt signals.

- Write a 1 to IER_{0:1} bits to enable alarm signal (unmask). Write-only.
- Write a 1 to IDR_{0:1} bits to disable alarm signal (mask). Write-only.
- Read the IMR_{0:1} bits to determine the state of the mask (1 means masked). Read-only.

Debug Environment

In a debug environment, the DAP controller can provide a general method to generate read and write transactions on the AXI interconnect including the APB interface to the PS and PL SYSMON units. The DAP controller is part of the AXI CoreSight™ debug environment.

The JTAG interface can also access the PL SYSMON control and status registers. The multiplexing of access structures for both SYSMON units are shown in Figure 9-1.

Operating Modes

The PL and PS SYSMON units operate independently. The operating modes include single read, default sequence, and auto sequence. The SYSMON units normally operate in their default sequence mode. They can be configured for a custom sequence while in this mode. Once the custom configuration is complete, set the sequence mode in the CONFIG_REG register. The activity of the PS ADC can be monitored by reading the AMS.MON_STATUS register fields [mon_data], [channel], and [busy].

Single-channel Mode

The single-channel mode measures one sensor channel at a time.

1. Write the channel number into the CONFIG_REG0 [mux_channel] bit field and select single-channel mode in the CONFIG_REG1 [sequence_mode] bit field.

   The sensor channel numbers are listed in the sensor channel tables. All channels support long acquisition [ACQ] and the external channels support input sampling type [BU] functions (refer to the CONFIG_REG0 register).

2. Wait for the EOC interrupt and then read the associated measurement register.

Default Sequence Mode

After a reset, the SYSMON operates in default mode with a simple auto-sequencer loop. The default sequence mode is established by any SYSMON reset. The default sequence can also be programmed by writing `0000h` to the CONFIG_REG1 [SEQUENCE_MODE] bit field. In this mode, the SYSMON unit operates independently of any other control register settings, monitors the default on-chip sensors, and stores average results in the measurement registers.

In the default sequence mode, the ADC is calibrated and averaging is set to 16 samples for all sensor channels. The SYSMON also operates in default mode during device configuration or if set using the sequence bits. Table 9-5 lists the default channel sequence.

Note: The automatic alarm function is not enabled in this mode.

Table 9-5: **Default Sequence for PL and PS SYSMON Units**

Sequence Order	PL SYSMON Channels	PS SYSMON Channels
1	Calibration (low rate)	Calibration (low rate)
2	VCC_PSINTLP	VCC_PSDDR
3	VCC_PSINTFP	VCC_PSIO3
4	VCC_PSAUX	VCC_PSIO0
5	Temp_PL (low rate)	Temp_LPD (low rate)
6	VCCINT	VCC_PSINTLP
7	VCCAUX	VCC_PSINTFP
8	VCCBRAM	VCC_PSAUX

Sequencer Modes

The SYSMON units can sequence through a list of enabled channels once or continuously. The analog inputs are time-multiplexed in a fixed order and presented to the ADC input circuitry one at a time. As readings are taken, the minimum and maximum values are stored for each channel. Measurements can also be averaged over successive readings. The sequencer follows the SEQ_CHANNEL and SEQ_LOW_RATE_CHANNEL register settings in the order shown in Table 9-2 and Table 9-4.

Programming Examples

This programming section describes examples for software running on a processor. The sequences are generally applicable to JTAG and other programming ports, but with different programming methods. The programming examples include the following modes.

- Continuous loop mode
- Single pass sequence mode

Note: The PL SYSMON unit programming examples reference the PLSYSMON register set. The PS SYSMON unit programming examples reference the PSSYSMON and AMS registers sets.

Note: When accessing a SYSMON unit, be sure its registers are accessible as described in Register Access via APB Slave Interface and Figure 9-6.

Example – Continuous Loop Mode

This programming example puts the SYSMON into its default sequence mode, enables alarms, configures the sequencer channels, and selects the continuous loop sequence mode. This routine can be used for both SYSMON units. Ensure access to the register sets before attempting to access them (see Register Access via APB Slave Interface).

1. Put the SYSMON unit into its default sequence mode and enable the alarms. Write `0000h` to the CONFIG_REG{1, 3} registers.
2. PS SYSMON unit only. If the PS SYSMON unit is held in reset, then deassert reset. This causes the unit to operate in its default sequence mode and allows the software to configure the sequence registers. The PS SYSMON unit reset is controlled by AMS.PS_SYSMON_CONTROL_STATUS [reset_user]. After reset is deasserted, wait until the AMS_CTRL.PS_SYSMON_CONTROL_STATUS [startup_done] bit is set. Software might begin the startup state machine again by writing a 1 to the self clearing [startup_trigger] bit and wait again for the [startup_done] bit.
3. Select the desired full-rate sensor channels. Write to the SEQ_CHANNEL registers.
4. If the low-rate sensor channels are to be included, write to the CONFIG_REG4 [sequence_rate] and [low_rate_eos] bit fields to define the rates and select the low-rate channels using the SEQ_LOW_RATE_CHANNEL registers. Do not select channels already selected in the SEQ_CHANNEL registers.
5. PL SYSMON unit only. If the external sensor channels are used (VP_VN, VAUX), select the desired sampling circuit and acquisition length for each of these channels. Write to the appropriate SEQ_INPUT_MODE{0, 1} and SEQ_ACQ{0, 1} registers.
6. If desired, set the alarm thresholds by writing to the ALARM_*_UPPER and ALARM_*_LOWER registers. The thresholds are described in Sensor Alarm Types. The programming is done using the transfer functions described in the *UltraScale Architecture System Monitor User Guide* (UG580) [Ref 6].

7. If average measurements are desired, select the channels in the SEQ_AVERAGE registers and set the desired averaging count in the CONFIG_REG0 [averaging] bit field.
8. Change the sequence mode from default mode to continuous loop mode and keep the alarms enabled. Write `2000h` to the CONFIG_REG1 register.

Read the measurement registers for the monitored channels. If averaging is enabled, a value does not appear in the measurement register until the SYSMON unit has looped through the channel sequencer the number of times in the averaging count value. The EOS event indicates when the measurement registers have valid data.

Example – Single Pass Sequence Mode

In single pass sequence mode, the sequencer makes one pass through the sequencer channel select registers and then stops. All channels are available for single pass mode.

The following features are not applicable.

- Low-rate sequencer
- Measurement averaging

1. Put the SYSMON unit into its default sequence mode and enable the alarms. Write `0000h` to the CONFIG_REG1 and CONFIG_REG3 registers.
2. PS SYSMON unit only. If the PS SYSMON unit is held in reset, deassert reset. This causes the unit to operate in its default sequence mode and allows the software to configure the sequence registers and perform calibration. The PS SYSMON unit reset is controlled by AMS.PS_SYSMON_CONTROL_STATUS [reset_user]. After reset is deasserted, wait until the AMS_CTRL.PS_SYSMON_CONTROL_STATUS [startup_done] bit is set. Software might begin the startup state machine again by writing a 1 to the self clearing [startup_trigger] bit and wait again for the [startup_done] bit.
3. Select the desired sensor channels. Write to the SEQ_CHANNEL registers.
4. PL SYSMON unit only. If any of the VP_VN, VAUX, or VUser channels are used, (i.e., VUser channels are internal), select the desired sampling circuit and acquisition length for each of these channels. Write to the appropriate SEQ_INPUT_MODE{0, 1} and SEQ_ACQ{0, 1} registers.
5. Change the sequence mode from default mode to single pass mode and keep the alarms enabled. Write `1000h` to the CONFIG_REG1 register.

Read the measurement registers for the monitored channels. The EOS event indicates when the measurement registers have valid data. The “eos” bit in the ISR_1 register is the EOS for the PS SYSMON.

Thermal Management

Thermal management software controls the system cooling by reading the temperature measurement register (polling) and configuring the temperature alarm thresholds (interrupt-driven). As the temperature rises, or an alarm interrupt is detected, the software can turn on or increase the speed of the fan cooling the device. The software can also cool down the device by instructing the system manager to reduce computational activity or lower the clock rate to reduce power dissipation.

Normal Temperature Alarm

The normal temperature alarm is used for thermal management. The normal alarm can be cleared with or without hysteresis as described in Lower Alarm Threshold.

Critical Over-Temperature Shutdown

The second set of temperature registers are used to signal a serious OT condition that can lead to operational failures. This alarm is used to shut down the system or take other drastic action to reduce the device temperature.

OT Alarm

The OT alarm is used to signal a need for drastic action. The OT alarm can be cleared with or without hysteresis as described in Lower Alarm Threshold.

The OT alarm is operational in the default state with the OT upper threshold set to approximately 122°C. This setting can be overwritten using the upper alarm threshold register. Write the threshold value into the [15:4] bit field and write `03h` into [3:0] bits to activate this value and to activate the lower threshold value defined in the OT lower threshold register. The measured value can be converted to a temperature using the transfer function described the *UltraScale Architecture System Monitor User Guide* (UG580) [Ref 6].

Note: The PS SYSMON uses the same upper and lower OT threshold registers for both the LPD and FPD temperature channels.

Register Sets

Table 9-6 summarizes the memory-mapped control and status registers for all three register sets AMS, PSSYSMON, and PLSYSMON. Register bit details are described in the *Zynq UltraScale+ MPSoC Register Reference* (UG1087) [Ref 4].

Software access to the registers is described in Register Access via APB Slave Interface. The register map for the PL SYSMON when accessed using JTAG, I2C/PMBus or the DRP is described in the *UltraScale Architecture System Monitor User Guide* (UG580) [Ref 6].

Table 9-6 includes the registers from the AMS, PSSYSMON (PS), and PLSYSMON (PL) register sets. The register base addresses are as follows.

- AMS: 0xFFA5_0000
- PS SYSMON: 0xFFA5_0800
- PL SYSMON: 0xFFA5_0C00

Table 9-6: **Register Sets Overview**

System Address	Register Name	AMS	PS	PL	Description
SYSMON, AMS Register Set					
0xFFA5_0000	MISC_CTRL	1			Invalid register access and DRP access.
0xFFA5_0010	ISR_{0, 1}, IMR_{0, 1}, IER_{0, 1}, IDR_{0, 1}	8			Interrupt status and mask for alarms and APU register address decode errors to generate IRQ 88.
0xFFA5_0040	PS_SYSMON_CONTROL_STATUS	1			Control sequencer, reset, conversion trigger.
0xFFA5_0044	PL_SYSMON_CONTROL_STATUS	1			Indicator for PS ability to access PL SYSMON registers via APB slave interface.
0xFFA5_0050	MON_STATUS (indicators).	1			Current channel, busy, and clock health.

Table 9-6: **Register Sets Overview *(Cont'd)***

System Address	Register Name	AMS	PS	PL	Description
0xFFA5_0060 0xFFA5_006C	VCC_PSPLL and VCC_PSBATT.	2			Measurement registers.
0xFFA5_0074 - 0xFFA5_0084	VCCBRAM, VCCINT, VCCAUX, VCC_PSDDR_PLL, and VCC_PSINTFP_DDR.	5			Measurement registers.
PS SYSMON Configuration Registers, PSSYSMON Register Set					
0xFFA5_08FC	STATUS_FLAG		1		Alarm status and power indicator.
0xFFA5_0900	CONFIG_REG0		1		Single-read, averaging, and sampling modes.
0xFFA5_0904	CONFIG_REG1		1		PS alarm disables [0:6] and sequencer mode.
0xFFA5_0908	CONFIG_REG2		1		Sleep mode, ADC clock divider ratio.
0xFFA5_090C	CONFIG_REG3		1		PS alarms disables [8:13].
0xFFA5_0910	CONFIG_REG4		1		Low-rate channel skips, EOS select.
PL SYSMON Configuration Registers, PLSYSMON Register Set					
0xFFA5_0CFC	STATUS_FLAG			1	Alarm status, power indicator, VREF selection, PL JTAG access indicators.
0xFFA5_0D00	CONFIG_REG0			1	Multiplexer, single-read channel, averaging and sampling modes.
0xFFA5_0D04	CONFIG_REG1			1	PL alarm disables [0:6] and sequencer mode.
0xFFA5_0D08	CONFIG_REG2			1	Sleep mode, ADC clock divider ratio.

Table 9-6: **Register Sets Overview *(Cont'd)***

System Address	Register Name	AMS	PS	PL	Description
0xFFA5_0D10	CONFIG_REG4			1	Low-rate channel skips, EOS select, and VUser voltage range select.
PS and PL Sequencer Configuration (PS SYSMON and PL SYSMON Registers)					
0xFFA5_0xxx {920, 918} 0xFFA5_0xxx {D20, D24, D18}	PS: SEQ_CHANNEL{0, 2} PL: SEQ_CHANNEL{0, 1, 2}		2	3	Select sensor channels for the normal sequence loop. Alternate name: SEQCHSEL.
0xFFA5_0xxx {9E8, 9F0} 0xFFA5_0xxx {DE8, DEC, DF0}	PS: SEQ_LOW_RATE_CHANNEL{0, 2} PL: SEQ_LOW_RATE_CHANNEL{0, 1, 2}		2	3	Select sensor channels for the low-rate sequence loop. Alternate name: SLOWCHSEL.
0xFFA5_0xxx {928, 91C} 0xFFA5_0xxx {D28, D2C, D1C}	PS: SEQ_AVERAGE{0, 2} PL: SEQ_AVERAGE{0, 1, 2}		2	3	Enable sensor channel measurement averaging. Alternate name: SEQAVG.
0xFFA5_0Dxx	PL: SEQ_INPUT_MODE{0, 1}			2	Select input sampling circuitry, unipolar, bipolar for external voltage nodes. Alternate name: SEQINMODE.
0xFFA5_0Dxx	PL: SEQ_ACQ{0, 1}			2	Select option to extend sampling time; potentially better reading. Alternate name: SEQACQ.
PS and PL ADC Results and Thresholds (PS SYSMON and PL SYSMON Registers)					
(1)	Voltage Node Names.		~10	~10	Voltage measurements.
	TEMP_{LPD, FPD, PL}		2	1	Temperature measurements.

Table 9-6: **Register Sets Overview *(Cont'd)***

System Address	Register Name	AMS	PS	PL	Description
	MIN_xx, MAX_xx		24	22	Minimum, maximum voltage and temperature readings.
	ALARM_UPPER_xx, ALARM_LOWER_xx		24	22	Upper, lower alarm thresholds.

Notes:

1. The address offsets and names for the measurement, temperature, minimum/maximum, and upper/lower threshold registers are shown in table Table 9-2 and Table 9-3. Table 9-4 shows the measurement registers for several basic channels measured by the PS SYSMON unit.

Register Access via APB Slave Interface

The programming model for the PS and PL SYSMON units is described from a processor point of view with access to the PSSYSMON, PLSYSMON, and AMS register sets provided via a memory mapped LPD APB slave interface in the IOP. In this case, any processor connected to the AXI interconnect can potentially control the SYSMON, PMU, RPU, APU, DAP controller, and masters instantiated in the PL.

The AMS and PSSYSMON register sets are natively connected as an ABP slave interface and are protected by the XPPU protection unit. Check that the [jtag_locked] bit is "0" in the MON_STATUS register to make sure the clock is operating within the recommended range before attempting to access the PS SYSMON registers.

By contrast, the PL SYSMON unit's PLSYSMON register set has several programming interface paths that can be enabled and potentially at the same time. One of the default access paths is also to the memory mapped APB slave interface in the IOP. The other access paths to the PL SYSMON unit registers, including PL fabric and serial access, are described in Register Access via PL Fabric andSerial Channels.

The JTAG DAP controller can use the AXI interconnect to access the APB slave interfaces. For bandwidth considerations, the DAP controller via JTAG is a serial interface. The access paths to the PL unit are shown in Figure 9-6. There are several conditions and restrictions that control access to all register sets.

AMS Register Set Access

The AMS register set adapts the PS SYSMON core to the PS environment. The core's pins are attached to register bits in the AMS register set. Also, the monitor alarms are processed by the AMS interrupt registers to generate the IRQ 88 system interrupt. The AMS registers are located at `0xFFA5_0000`.

Access to the AMS register set requires the following.

- Privilege from the XPPU protection unit.
- VCC_PSINTLP, VCC_PSAUX.
- AMS.MON_STATUS [jtag_locked] bit (good AMS_REF_CLK from PS).

PSSYSMON Register Set Access

Software can access the PS SYSMON registers at `0xFFA5_0800` (PSSYSMON register set). These registers are mapped to the IOP slave ports and are protected by the XPPU. Check the [jtag_locked] bit to confirm the clock is operating correctly before attempting to access the PS SYSMON registers.

- AMS requirements.

PLSYSMON Register Set Access

The PL SYSMON unit is controlled by the PLSYSMON register set at `0xFFA5_0C00`. These registers are also protected by the XPPU and require a valid clock. In addition, the PCAP isolation wall must be disabled to access the PLSYSMON registers and control the PL SYSMON unit. The APB/AXI and DRP parallel ports provide greater bandwidth access to the SYSMON units than the serial ports.

- AMS requirements.
- AMS.MON_STATUS [jtag_locked] bit (good clock).
- APB slave interface (default mode).
    - Check the AMS.PL_SYSMON_CONTROL_STATUS [accessible] bit.)
    - No SYSMONE4 instantiation.
- PCAP isolation wall disabled.
- JTAG, I2C/PMBus arbitration.
- VCCINT (check the [PL_INIT] bit.)

PL SYSMON Register Access Arbitration

The PL SYSMON unit can be accessed using only one of the following ports.

- DRP via APB slave connected to AXI and the PS.
- DRP via PL (using the instantiated SYSMONE4 primitive).
- I2C/PMBus connected to device pins.
- JTAG PL TAP controller.

If the bitstream instantiates the SYSMONE4 primitive, then the PL design has control over the DRP interface to the PLSYSMON registers. The PS does not have access unless an alternative DRP to APB to AXI interface is established in the PL fabric and connected to a PS-PL AXI interface. The state of the native AXI interface to the PLSYSMON registers is reflected by the AMS.PL_SYSMON_CONTROL_STATUS [accessible] bit.

The PS should avoid attempts to access a PL SYSMON registers whenever a JTAG or I2C/PMBus transaction is accessing them. The APB interface assumes that it has dedicated access to the PL SYSMON registers. Simultaneous attempts to access the PL SYSMON via JTAG or I2C/PMBus can lead to unpredictable behavior of the PS. The JTAG and I2C/PMBus interfaces are available prior to PL configuration so appropriate caution and measures should be taken to avoid conflict if the PS also uses the APB interface while the device is in this state. Dedicated access to the PL SYSMON via the APB interface can be guaranteed following PL configuration with a design that does not instantiate the SYSMONE4 primitive. PL configuration disables the I2C/PMBus interface.

The JTAG interface can be disabled by generating a configuration image using the `set_property BITSTREAM.GENERAL.JTAG_SYSMON DISABLE [current_design]` option.

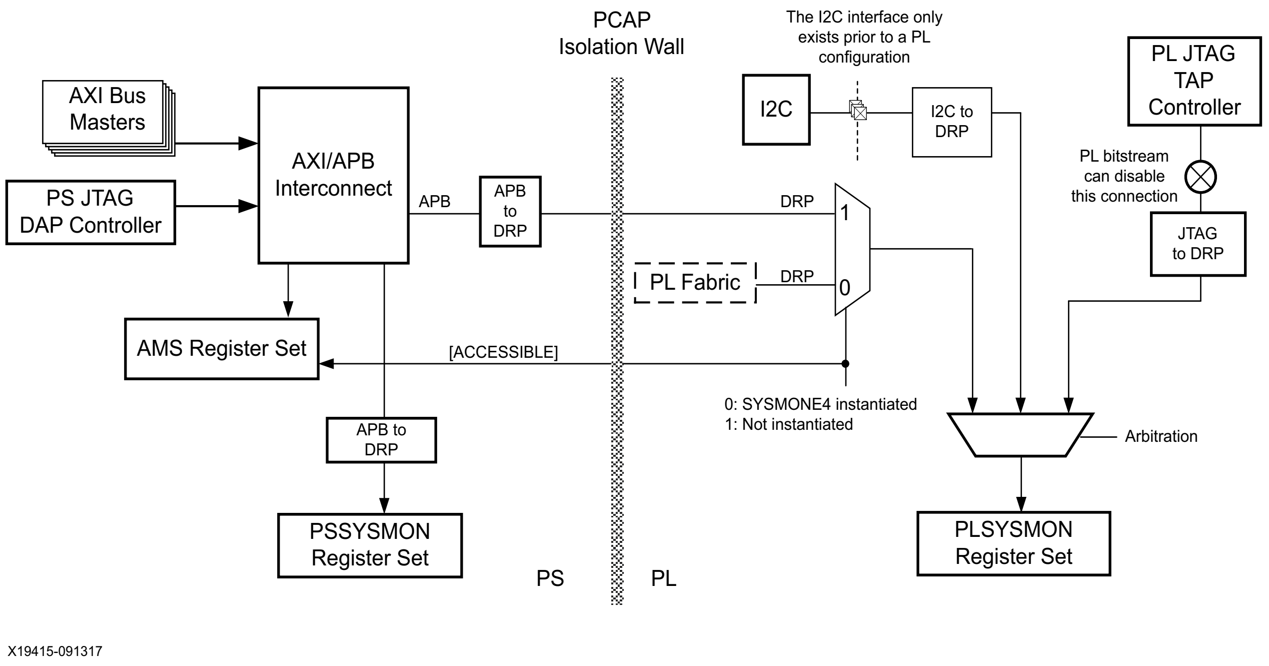


Figure 9-6: **Register Access Paths**

Register Access via PL Fabric andSerial Channels

There are several options for accessing the PL SYSMON unit registers other than the APB slave interface in the IOP. See Table 9-7.

- DRP slave interface (SYSMONE4 primitive instantiated)
- I2C/PMBus interface (package pins and SYSMONE4 primitive instantiated)
- PL TAP controller (debug environment, arbitrates, can be locked out by bitstream)

Table 9-7: **PL SYSMON Unit Register Access Interfaces**

Interface to PL SYSMON Unit	**PL is Not Configured**	**PL is Configured**	
		No SYSMONE4	**SYSMONE4 Instantiated**
APB slave interface[(1)]	Yes, but VCCINT required and disabled PCAP isolation wall.	Yes, if [accessible] and not isolated.	No.[(2)]
DRP via PL fabric	Not applicable.	No.	Yes, if connected.
I2C/PMBus	Yes.	No.	Yes, if connected.

Table 9-7: **PL SYSMON Unit Register Access Interfaces *(Cont'd)***

Interface to PL SYSMON Unit	PL is Not Configured	PL is Configured	
		No SYSMONE4	SYSMONE4 Instantiated
JTAG DAP controller	Yes.	Yes, unless disabled by bitstream.	Yes, unless disabled by bitstream.

Notes:

1. The I2C/PMBus and PL JTAG arbitrate for access. The software should not access the SYSMONs via the AXI/APB interconnect when the I2C/PMBus or JTAG interfaces are being used.
2. If the PS needs to communicate with the PL SYSMON unit when the SYSMONE4 primitive is instantiated, then the PL design must include an alternative path that interfaces to AXI PS-PL interface to the DRP interface on the instantiated SYSMONE4 primitive. Such a design would result in the PL SYSMON being treated as a user-specific hardware peripheral by the PS and the base address of the PL SYSMON registers would be within the PS-PL interface address space.
3. Refer to Answer Record 71067 to know when the APB to DRP path is to be used.

DRP Slave Interface in PL Fabric

The DRP slave interface is selected by the SYSMONE4 instantiation. This parallel interface can be adapted to an APB slave interface by a PL design.

PL TAP Controller Interface via JTAG

The JTAG interface is converted to the DRP protocol to access the PL SYSMON. The TAP controller commands that are used to access the PL SYSMON registers are described the *UltraScale Architecture System Monitor User Guide* (UG580) [Ref 6]. The PL JTAG interface is selected by the SYSMON_DRP command. This also enables the I2C serial interface.

I2C Serial Interface via Device Pins

The I2C serial interface is described in the *System Management Wizard v1.3 LogiCORE IP Product Guide (*PG185) [Ref 19]. The commands to access the SYSMON registers are described the *UltraScale Architecture System Monitor User Guide* (UG580) [Ref 6].

PM Bus

The PMBus interface uses the same signals as the I2C interface. The bus protocol is described in *UltraScale Architecture System Monitor User Guide* (UG580) [Ref 6].

System Interfaces

The SYSMON units have their own set of clocks and resets, and are controlled and monitored by several system signals.

- Clocks
- Reset sources and state
- Power
- Control signals

Clocks

The SYSMON clock is driven by an interface clock. The interface clock is divided down to generate the ADC clock using the CONFIG_REG2 [clock_divider] bit field.

- PL SYSMON clock is based on LPD_LSBUS_CLK (APB bus) or PL_DCLK (when the SYSMONE4 primitive is instantiated).
- PS SYSMON clock is based on LPD_LSBUS_CLK (APB bus).

The software can determine if the PS SYSMON clock is out of range by reading the AMS.MON_STATUS [jtag_locked] bit.

On the PL SYSMON unit, if the JTAG interface is experiencing a JTAG_Locked condition, the PL SYSMON unit is either busy transacting with another interface or the clock is out of range.

Reset Sources

The PS and PL SYSMON units have different reset methods. Not all reset methods are available all the time.

PL SYSMON

- Power on (self boot).
- PL configuration including partial reconfiguration of the PL.
- Assert reset pin on SYSMONE4 primitive.
- Write any value to the VP_VN Status Register 3.

PS SYSMON

- Internal or external POR (includes power on).
- System reset.
- Write to AMS.PS_SYSMON_CONTROL_STATUS [reset_user].

The SYSMON unit activity is normally switched between the default mode and the user-programmed sequence mode. The SYSMON can be reset if necessary. The effects of the resets are described in Reset States.

When a unit is operating in its default sequence mode, the software can configure a user-defined sequence by writing to the channel sequence and threshold registers. After the user mode is configured, write to the CONFIG_REG1 register to select it. Return to the default sequence mode to program another sequence.

Reset States

Measurement Registers

- Measurement result registers are set to `0000h`.
- Minimum result registers are set to `FFFFh`.
- Maximum result registers are set to `0000h`.

Configuration Registers

- Status and configuration.
- Sequence channel and low rate channel.
- Sequence average and acquisition time.
- Upper and lower threshold.

Table 9-8: **Reset Matrix**

SYSMON Unit	Reset Control		Default Sequence	SYSMON Measurement Registers	SYSMON Config. Registers	SYSMON Other Registers	AMS Registers	Clearing
PS and PL	Internal or External POR		Yes, except modified by PMU pre-boot ROM code.	Yes	Yes	Yes	Yes	N/A
PS: PL:	PS System Reset (SRST) PL System Reset		Yes					N/A
PS:	CRL_RST_LPD.RST_LPD_TOP	[sysmon_reset]	Yes	No	No	No	Yes	Not self clearing.
PS: PL:	AMS.PS_SYSMON_CONTROL_STATUS SYSMONE4 primitive	[startup_trigger]	Yes	Yes	Yes	Yes	No	Self cleared when done.
PS: PL:	AMS.PS_SYSMON_CONTROL_STATUS SYSMONE4 primitive	[reset_user]	No	Yes	No	No	No	Not self clearing.
PS: PL:	PSSYSMON.VP_VN PLSYSMON.VP_VN	[any register write]	No	Yes	No	No	No	Not set.
PS: PL:	PSSYSMON.CONFIG_REG1 PLSYSMON.CONFIG_REG1	[sequence_mode]	Yes	Yes	No	No	No	Stays in sequence mode.

Power

The power needs for each SYSMON are described in this section. The power for the various register access paths are described in Register Access via APB Slave Interface.

PS SYSMON Unit

- VCC_PSADC supplies power to the ADC circuitry
- VCC_PSAUX supplies power to the LPD and FPD temperature measurement sensors.
- VCC_PSINTLP supplies power to the logic and to the AMS and PSSYSMON register sets.

PL SYSMON Unit

- VCCADC supplies power to the ADC circuitry
- VCCAUX supplies power to the PL temperature measurement sensor.
- VCCINT supplies power to the logic and PLSYSMON registers.

Control and Monitor Signals

Note: The digital signals between the PL SYSMON unit and the PS are susceptible to the state of the PCAP isolation wall.

Alarms Signals

Each sensor channel can assert an alarm. The CONFIG_REG registers in each SYSMON can be configured to disable each alarm before it is routed to the PMU, CSU, PL, and the AMS interrupt registers. The alarm signals are shown in Figure 9-4. Several voltage nodes measured by the PS SYSMON unit are only accessible via the AMS register set and do not have alarms, see Table 9-4.

IRQ Interrupt

The AMS interrupt registers are programmed to enable alarm signals to generate IRQ 88 to the GICs and PL. The IRQ interrupt can also be generated when the PS SYSMON conversion or sequence is finished, or there is an address decode error on one of the three register sets. There are two interrupt register sets for the following sources.

- ISR_{0, 1} are read/write and provide the interrupt status (before the mask) corresponding with each alarm signal. Writing a “1” to a status bit clears the interrupt.
- IMR_{0, 1] are read-only and provide a mask that is used after the status and before the wide OR gate to generate IRQ 88.
- IER_{0, 1} and IDR_{0,1} are write-only to set and clear bits in the IMR registers.
- ITR_{0, 1} are write-only to enable software to trigger an individual interrupt bit in ISR.

Sequence Triggers

The trigger signal can be used to start a user programmed sequence. The trigger is set up using the [auto_convst] and [convst] controls of the PS SYSMON unit and the [convst] and [convstclk] controls of the PL SYSMON unit. The trigger signaling works differently in each SYSMON unit.

- In the PL SYSMON unit, the trigger signal is an input in the PL fabric when the SYSMONE4 primitive is instantiated.
- In the PS SYSMON unit, the trigger signal connects to the AMS.PS_SYSMON_CONTROL_STATUS register.

End-of-Conversion and End-of-Sequence Events

The EOC and EOS events are described in the Interrupts section.

System Addresses

Introduction

This chapter describes the address map of the Zynq® UltraScale+™ MPSoC that can support a single address map configuration for up to 1 TB of physical address space. The Arm v8-A architecture allows physical address configuration by software.

Global Address Map

The global address map is composed of multiple inclusive address maps, depending on the address width of the interface master. The Zynq UltraScale+ MPSoC address map is 40 bits (the physical address space is a maximum of 40 bits).

32-bit (4 GB) Address Map

To maintain compatibility with 32-bit software, a lower 4 GB address map provides aperture for all the devices. All of the peripheral address space is allocated in the lower 4 GB, with a fixed address map.

36-bit (64 GB) Address Map

The 36-bit address map is a superset of the 32-bit address map. The address space beyond 4 GB is allocated to the PL, the interface for PCIe, and the DDR controller. An additional 32 GB is allocated to the DDR controller in this region.

40-bit (1 TB) Address Map

The 40-bit address map is a superset of the 36-bit address map. The address space beyond 64 GB is allocated to the PL, the interface for PCIe, and the DDR controller.

System Address Map Interconnects

Based on the required address size, a page translation table walk can use fewer steps. For example, a 40-bit address translation (to 4 KB pages) takes four table-walker steps. A 36-bit address translation takes three table-walker steps. Thus, in a 40-bit address size system, performance is optimized by limiting the address size to 36 bits, if a 36-bit address size is sufficient for the application.

The interconnect addresses between various processing system (PS) masters to the translation buffer units (TBUs) of the system memory management unit (SMMU) are virtual addresses. The address bus (from master to SMMU) is 48 bits for the 64-bit compliant PS masters (APU, PCIe, SATA, DisplayPort, USB, GEM, SD, NAND, QSPI, and the CSU, LPD, and DMA units). The 32-bit PS masters provide a 32-bit address bus, which is zero-extended to 48 bits. The SMMU supports a 49-bit address. For PS-masters, the 49th address bit to the SMMU is zero, and the address bus from the programmable logic (PL) AXI interfaces into the PS is 49 bits.

The global system address map is shown in Figure 10-1.

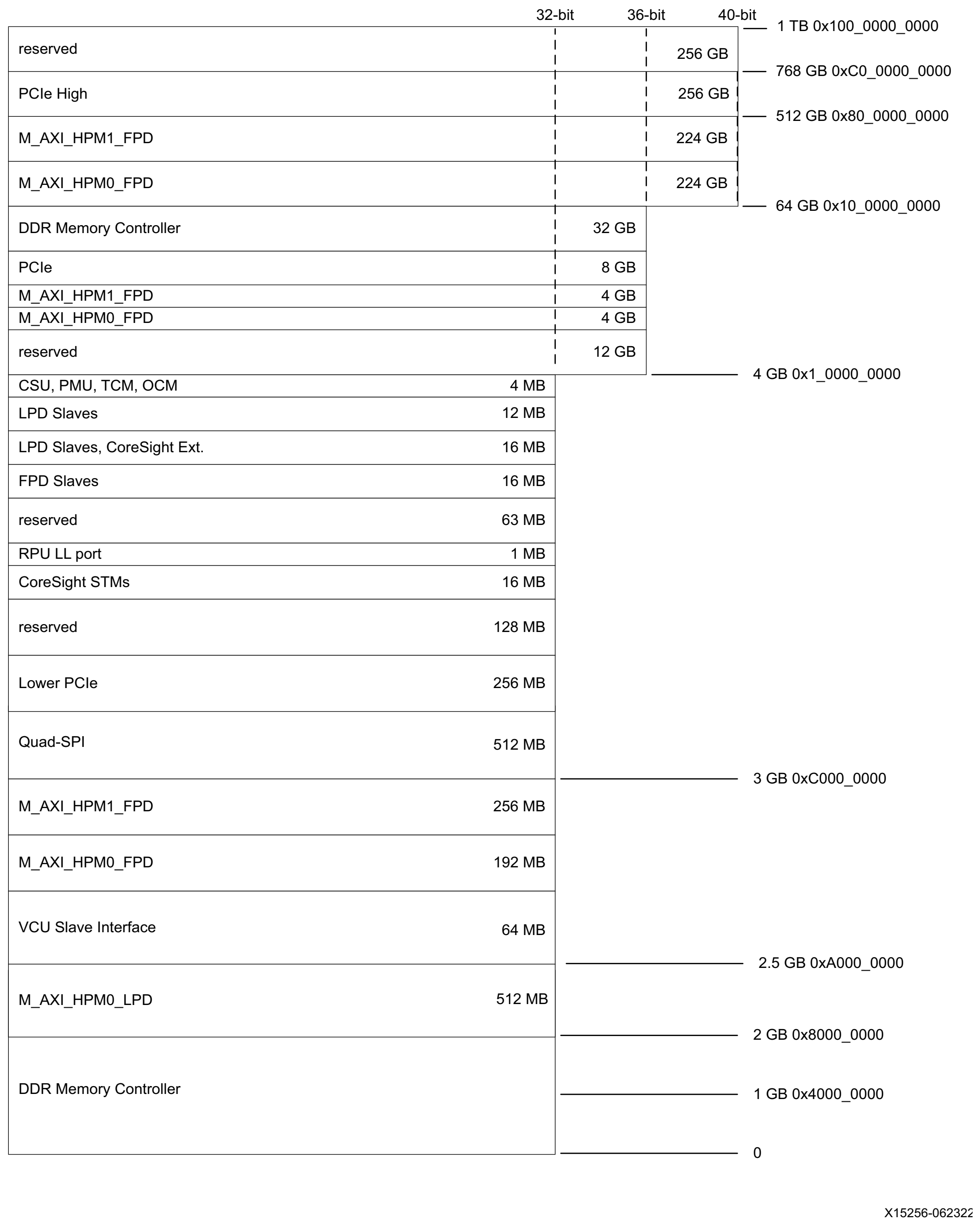


Figure 10-1: **Global System Address Map**

The SMMU supports two stage translations: Stage 1 (virtual address (VA) to intermediate physical address (IPA)), and stage 2 (IPA to physical address). The PS master virtualization target is primarily a stage 2 translation (for example, a hypervisor scenario uses only stage 2 translations). The PL can use a stage 1 and/or a stage 2 translation. For details on SMMU translation, see the SMMU Architecture section in Chapter 3.

For the stage 2 translation, the Arm v8 architecture supports a maximum of 48 bits of IPA address. For the stage 1 translation, the Arm v8 architecture supports a 49-bit maximum addressing.

System Address Map

PL AXI Interface

The AXI interface from the LPD to PL is assigned a fixed address space of 512 MB in the lower 4 GB address space. It is typically used for LPD to PL communications because it provides a low-latency path from the LPD masters like the RPU and the LPD DMA unit to the PL. The AXI interfaces from the FPD to PL are assigned multiple address ranges.

The comprehensive system-level addresses map is shown in Table 10-1.

Table 10-1: **Top-Level System Address Map**

Slave Name	Size	Start Address	End Address
DDR Low	2 GB	0x0000_0000	0x7FFF_FFFF
M_AXI_HPM0_LPD (LPD_PL)	512 MB	0x8000_0000	0x9FFF_FFFF
VCU[(1)]	64 MB	0xA000_0000	0xA3FF_FFFF
M_AXI_HPM0_FPD (HPM0) interface[(1)]	192 MB	0xA400_0000	0xAFFF_FFFF
M_AXI_HPM1_FPD (HPM1) interface	256 MB	0xB000_0000	0xBFFF_FFFF
Quad-SPI	512 MB	0xC000_0000	0xDFFF_FFFF
PCIe Low	256 MB	0xE000_0000	0xEFFF_FFFF
Reserved	128 MB	0xF000_0000	0xF7FF_FFFF
STM CoreSight	16 MB	0xF800_0000	0xF8FF_FFFF
APU GIC	1 MB	0xF900_0000	0xF90F_FFFF
Reserved	63 MB	0xF910_0000	0xFCFF_FFFF
FPD slaves	16 MB	0xFD00_0000	0xFDFF_FFFF
Upper LPD slaves	16 MB	0xFE00_0000	0xFEFF_FFFF
Lower LPD slaves	12 MB	0xFF00_0000	0xFFBF_FFFF
CSU, PMU, TCM, OCM	4 MB	0xFFC0_0000	0xFFFF_FFFF
Reserved	12 GB	0x0001_0000_0000	0x0003_FFFF_FFFF
M_AXI_HPM0_FPD (HPM0)	4 GB	0x0004_0000_0000	0x0004_FFFF_FFFF
M_AXI_HPM1_FPD (HPM1)	4 GB	0x0005_0000_0000	0x0005_FFFF_FFFF
PCIe High	8 GB	0x0006_0000_0000	0x0007_FFFF_FFFF
DDR High	32 GB	0x0008_0000_0000	0x000F_FFFF_FFFF
M_AXI_HPM0_FPD (HPM0)	224 GB	0x0010_0000_0000	0x0047_FFFF_FFFF
M_AXI_HPM1_FPD (HPM1)	224 GB	0x0048_0000_0000	0x007F_FFFF_FFFF
PCIe High	256 GB	0x0080_0000_0000	0x00BF_FFFF_FFFF

Table 10-1: **Top-Level System Address Map *(Cont'd)***

Slave Name	Size	Start Address	End Address
Reserved	256 GB	`0x00C0_0000_0000`	`0x00FF_FFFF_FFFF`

Notes:

1. The VCU is mapped by the design tools to the 64 MB address space listed in Table 10-1, but it can be configured to another address within an M_AXI_HPMx_FPD address range, if desired. If VCU is not mapped, the M_AXI_HPM0_FPD interface has a 256 MB range.

As listed in Table 10-2, the 4 MB region is further partitioned and set aside for the configuration security unit (CSU: Chapter 11), platform management unit (PMU: Chapter 6), tightly-coupled memory in RPU (RPU: Chapter 4), and on-chip memory (OCM: Chapter 18).

Table 10-2: **CSU, PMU, TCM, and OCM Address Space**

Slave Name	Size	Start Address	End Address
CSU_RAM	32 KB	`0x00FFC40000`	`0x00FFC47FFF`
CSU_ROM	128 KB	`0x00FFC00000`	`0x00FFC1FFFF`
EFUSE	64 KB	`0x00FFCC0000`	`0x00FFCCFFFF`
PMU_ROM	256 KB	`0x00FFD00000`	`0x00FFD3FFFF`
PMU_RAM	128 KB	`0x00FFDC0000`	`0x00FFDDFFFF`
OCM_RAM	256 KB	`0x00FFFC0000`	`0x00FFFFFFFF`
R5_0_ATCM_SPLIT	64 KB	`0x00FFE00000`	`0x00FFE0FFFF`
R5_0_BTCM_SPLIT	64 KB	`0x00FFE20000`	`0x00FFE2FFFF`
R5_0_ICACHE	64 KB	`0x00FFE40000`	`0x00FFE4FFFF`
R5_0_DCACHE	64 KB	`0x00FFE50000`	`0x00FFE5FFFF`
R5_1_ATCM_SPLIT	64 KB	`0x00FFE90000`	`0x00FFE9FFFF`
R5_1_BTCM_SPLIT	64 KB	`0x00FFEB0000`	`0x00FFEBFFFF`
R5_1_ICACHE	64 KB	`0x00FFEC0000`	`0x00FFECFFFF`
R5_1_DCACHE	64 KB	`0x00FFED0000`	`0x00FFEDFFFF`
R5_0_ATCM_LSTEP	128 KB	`0x00FFE00000`	`0x00FFE1FFFF`
R5_0_BTCM_LSTEP	128 KB	`0x00FFE20000`	`0x00FFE3FFFF`

The reserved address regions are listed in Table 10-3.

Table 10-3: **Reserved Addresses**

Address Range	Notes
0xF000_0000 to 0xF7FF_FFFF	128 MB reserved
0xF910_0000 to 0xFCFF_FFFF	63 MB reserved

System Address Register Overview

The registers for system-level control, private bus, PS I/O peripherals, and miscellaneous PS functions are listed in this section.

System-level Control Registers

The system-level control register sets are used to control the PS behavior. The detailed descriptions for each register is available in the *Zynq UltraScale+ MPSoC Register Reference* (UG1087) [Ref 4]. A summary of the registers with their base addresses is shown in Table 10-4. Several register sets always require a secure access. All registers are accessed via the XPPU, which can set the access requirements for secure, read/write, and by master. For more information on *System Software Mutexes, see System Software Mutexes on Zynq UltraScale* [Ref 59].

Table 10-4: **System-level Register Sets**

Base Address	Name	Secure Access	Description
0xFD1A_0000	CRF_APB	XMPU	FPD clock and reset control.
0xFD5C_0000	APU	XMPU	APU control. See Table 3-2, page 62.
0xFD61_0000	FPD_SLCR	XMPU	Global SLCR for full-power domain (FPD).
0xFD69_0000	FPD_SLCR_SECURE	Yes	Global SLCR for FPD TrustZone settings for PCIe, SATA, and other protocols.
0xFF18_0000	IOU_SLCR	XPPU	IOU SLCR for MIO pin configuration.
0xFF24_0000	IOU_SECURE_SLCR	Yes	IOU SLCR for AXI read/write protection configuration.
0xFF26_0000	IOU_SCNTRS	Yes	Always system timestamp generator.
0xFF41_0000	LPD_SLCR	XPPU	SLCR for the low-power domain (LPD).
0xFF4B_0000	LPD_SLCR_SECURE	Yes	SLCR for LPD TrustZone configuration.
0xFF5E_0000	CRL_APB	XPPU	LPD clock and reset control.
0xFF9A_0000	RPU	XPPU	RPU control.
0xFD6E_0000	CCI_GPV	Yes	CCI_GPV (CCI400, parameters)
0xFD70_0000	FPD_GPV	Yes	FPD_GPV (parameters)

Private CPU Registers

There are separate private CPU registers for the RPUs and APUs to program the interrupt controllers. The addresses are shown in Table 10-5. The APU_GIC is located on the AXI interconnect and can be made exclusively accessible to the APU by using the FPD_XMPU protection unit.

Table 10-5: **CPU Private Registers**

Register Base Address	Description
`0xF900_0000` to `0xF900_1FFF`	GIC distributor.
`0xF900_2000` to `0xF900_2FFF`	GICC interface.
`0xFD6E_0000` to `0xFD6E_FFFF`	CCI_GPV (CCI400, parameters)
`0xFD70_0000` to `0xFD7F_FFFF`	FPD_GPV (parameters)
`0xFE00_0000` to `0xFE0F_FFFF`	IOU_GPV (parameters)
`0xFE10_0000` to `0xFE1F_FFFF`	LPD_GPV (parameters
`0xFD80_0000` to `0xFDFF_FFFF`	SMMU_GPV (SMMU500, parameters)

Note: The generic CPU timer, L2 cache, and SCU (etc.) in the APU can only be accessed through co-processor instructions, they are not memory mapped.

PS I/O Peripherals Registers

The I/O peripheral registers are accessed through the 32-bit APB bus. The base addresses for both the low-power domain and the and full-power domain peripherals are listed in Table 10-6 and Table 10-7.

Table 10-6: **I/O Peripherals Register Map (LPD)**

Base Address	Description
`0xFF00_0000, 0xFF01_0000`	UART0, UART1
`0xFF02_0000, xFF03_0000`	I2C0, I2C1
`0xFF04_0000, 0xFF05_0000`	SPI0, SPI1
`0xFF06_0000, 0xFF07_0000`	CAN0, CAN1
`0xFF0A_0000`	GPIO
`0xFF0B_0000, 0xFF0C_0000, 0xFF0D_0000, 0xFF0E_0000`	GEM0, GEM1, GEM2, GEM3
`0xFF0F_0000`	QSPI
`0xFF10_0000`	NAND(1)(2)
`0xFF16_0000, 0xFF17_0000`	SD0, SD1
`0xFF99_0000`	IPI message buffer memory; see Table 13-3.
`0xFF9D_0000, 0xFF9E_0000`	USB0, USB1

Table 10-6: **I/O Peripherals Register Map (LPD) *(Cont'd)***

Base Address	Description
`0xFFA5_0000`, `0xFFA5_0800`, `0xFFA5_0C00`(3)	System monitor register sets (AMS, PSSYSMON, PLSYSMON)
`0xFFCB_0000`	CSU_SWDT, system watchdog timer (csu_pmu_wdt).

Notes:

1. NAND cannot be accessed through AXI as a linear mode peripheral.
2. AXI address cannot be directly translated to NAND memory address.
3. The default address for the PL SYSMON register set is `0xFFA5_0C00`, but can be changed by instantiating the SYSMONE4 LogiCORE and mapping it to an M_AXI_HPMx_FPD or M_AXI_HPM0_LPD interface to the PL.

Table 10-7: **I/O Peripheral Register Map (FPD)**

Base Address	Description
`0xFD0C_0000`	SATA registers (HBA, vendor, port-0/1 control)
`0xFD0E_0000`	AXI PCIe bridge
`0xFD0E_0800`	AXI PCIe ingress {0:7}
`0xFD0E_0C00`	AXI PCIe egress {0:7}
`0xFD0F_0000`	AXI PCIe DMA {0:7}
`0xFD3D_0000`	SIOU slave access ports
`0xFD40_0000`	PS GTR transceivers
`0xFD48_0000`	PCIe attributes
`0xFD4A_0000`	DisplayPort controller
`0xFD4B_0000`	GPU
`0xFD4C_0000`	DisplayPort DMA

PS System Registers

Registers not covered in the previous sections are listed in Table 10-8.

Table 10-8: **PS System Register Map (LPD)**

Base Address	Description
`0xFF30_0000`	Inter-processor interrupts (IPI)
`0xFF11_0000`, `0xFF12_0000`, `0xFF13_0000`, `0xFF14_0000`	TTC0, TTC1, TTC2, TTC3
`0xFF15_0000`	LPD_SWDT, system watchdog timer (swdt0)
`0xFF98_0000`	XPPU (Xilinx peripheral protection unit)
`0xFF9C_0000`	XPPU_Sink
`0xFF9B_0000`	PL_LPD (S_AXI_LPD)
`0xFFA0_0000`	Arm for OCM interconnect
`0xFFA1_0000`	Arm for LPD to FPD interconnect

Table 10-8: **PS System Register Map (LPD)** ***(Cont'd)***

Base Address	Description
0xFFA6_0000	Real-time clock (RTC)
0xFFA7_0000	OCM_XMPU
0xFFA8_0000	LPD_DMA channels {0:7}
0xFFC8_0000	CSU_DMA
0xFFCA_0000	Configuration and security unit (CSU)
0xFFCD_0000	Battery-backed RAM (BBRAM) control and data

Table 10-9: **PS System Register Map (FPD)**

Base Address	Description
0xFD00_0000	DDR_XMPU{0:5}
0xFD07_0000	DDR controller
0xFD08_0000	DDR PHY
0xFD09_0000	DDR QoS control
0xFD0B_0000	Arm for DDR
0xFD36_0000	HPC0 (S_AXI_HPC0_FPD)
0xFD37_0000	HPC1 (S_AXI_HPC1_FPD)
0xFD38_0000	HP0 (S_AXI_HP0_FPD)
0xFD39_0000	HP1 (S_AXI_HP1_FPD)
0xFD3A_0000	HP2 (S_AXI_HP2_FPD)
0xFD3B_0000	HP3 (S_AXI_HP3_FPD)
0xFD49_0000	Arm for CCI
0xFD4D_0000	FPD_SWDT, system watchdog timer (swdt1)
0xFD50_0000	FPD_DMA channels {0:7}
0xFD5D_0000	FPD_XMPU
0xFD4F_0000	XMPU_Sink (FPD)
0xFD5E_0000	CCI_REG register set wrapper: debug enables
0xFD5F_0000	SMMU_REG (interrupts, power, and unit control)
0xFD6E_0000	CCI_GPV (CCI400, parameters)
0xFD70_0000	FPD_GPV (parameters)
0xFD80_0000	SMMU_GPV (SMMU500, parameters)
0xFE00_0000	IOU_GPV (parameters)
0xFE10_0000	LPD_GPV (parameters)

Boot and Configuration

Introduction

The system boot-up process is managed and carried out by the platform management unit (PMU) and configuration security unit (CSU).

The boot-up process consists of three functional stages.

- Pre-configuration stage

  The pre-configuration stage is controlled by the platform management unit that executes PMU ROM code to setup the system. The PMU handles all reset and wake-up processes. Power-on reset is used to reset the CSU and PMU because they are responsible for debug, system, and software reset. There are other reset methods such as SRST and SLCR.

- Configuration stage

  In the configuration stage, the BootROM (part of the CSU ROM code) interprets the boot header to configure the system and load the processing system's (PS) first-stage boot loader (FSBL) code into the on-chip RAM (OCM) in both secure and non-secure boot modes. The boot head defines many boot parameters including the security mode and the processor MPCore to execute the FSBL. The boot header parameters are listed in Table 11-4. During boot, the CSU also loads the PMU user firmware (PMU FW) into the PMU RAM to provide platform management services in conjunction with the PMU ROM. The PMU FW must be present in most systems for the Xilinx-based FSBL and system software.

- Post-configuration stage

  After a FSBL execution starts, the CSU ROM code enters the post-configuration stage, which is responsible for system tamper response. The CSU hardware provides ongoing hardware support to authenticate files, configure the PL via PCAP, store and manage secure keys, and decrypt files.

Boot Flow

The PMU performs a number of mandatory and optional security operations, including the following.

- Optional function: zeroize low power domain (LPD) registers. When the LPD_SC eFUSEs are programmed, the PMU zeroizes all registers in the LPD.
- Optional function: zeroize full power domain (FPD) registers. When the FPD_SC eFUSEs are programmed, the PMU zeroizes all registers in the FPD.
- Zeroize PMU RAM: the PMU RAM has zeros written to it and read back to confirm the write was successful.
- Zeroize the PMU processor's TLB memory.
- Voltage checks: the PMU checks the supply voltage of the LPD, AUX, and dedicated I/O to confirm that the voltages are within specification.
- Zeroize memories: the PMU zeroizes memories located in the CSU, LPD, and FPDs.

Once these security operations are complete, the PMU sends the CSU immutable ROM code through the SHA-3/384 engine and compares the calculated cryptographic checksum to the golden copy stored in the device. If the cryptographic checksums enabled in the bif file match, the integrity of the CSU ROM is validated and the reset to the CSU is released.

The PMU is responsible for handling the primary pre-boot tasks and management of the PS for reliable power up/power down of system resources. The power-on reset (POR) initiates the PMU operation which directly or indirectly releases resets to any other blocks that are expected to be powered up. In this paradigm, the PMU requires ROM code to hold the initial power-up sequence. The PMU is running even after the boot-up process and is responsible for handling various system resets. It is also used while changing the power state of the system (like power-up, sleep, and wake-up).

During initial boot, the PMU is brought out of reset by the POR, which is then followed by PMU ROM execution. The following describes the sequence of operations done by the PMU processor by executing PMU ROM pre-boot code after a POR reset.

1. Initialize the PS SYSMON unit and the PLL required for boot.
2. Clear the PMU RAM and CSU RAM (external POR only).
3. Validate the PLL locks.
4. Validate the LPD, AUX, and I/O supply ranges using the PS SYSMON unit.
5. Clear the low-power and full-power domains.
6. If there is no error in the previous steps, the PMU releases the CSU reset and enters the PMU service mode. If not, generate and flag a boot error.

Note: When PMUFW is not used PMU goes to sleep state after boot-up.

When the CSU reset is released, it performs following sequence.

1. Initialize OCM.
2. Determines the boot mode by reading the boot mode register from the captured boot mode state PMU FW, at the POR.
3. The CSU continues by loading the FSBL in OCM for execution by either the RPU or the APU. The CSU then loads the PMU user firmware (PMU FW) into the PMU RAM for execution by the PMU firmware.

   The PMU FW provides platform management services in conjunction with the PMU ROM code. The PMU FW is required in most systems and must be present for the Xilinx-based FSBL and system software. The PMU is described in Chapter 6, Platform Management Unit.

The CSU is the central configuration processor that manages secure and non-secure system-level configuration. Triple redundancy and built-in ECC (in the embedded processor and surrounding logic) is for system reliability and strong SEU resilience. The CSU also contains the key management unit, crypto accelerators, and the PS/PL programming interface.

The CSU is composed of two main blocks:

- A triple-redundant secure processor. It contains the triple-redundant embedded processor(s), associated ROM, a small private RAM for security sensitive data storage, and the necessary control/status registers required to support all secure operations.
- A crypto interface contains AES-GCM, a key vault for key storage, DMA, SHA3, RSA, and the processor configuration-access port (PCAP) interface.

Boot Modes

The BootROM can boot the system from Quad-SPI, SD, eMMC, USB 2.0 controller 0, or NAND external boot devices.

Note: The flash memory devices for boot are listed in Answer Record 65463. For SD and eMMC devices, the JEDEC interface specified in Chapter 26, SD/SDIO/eMMC Controller is supported. For Quad-SPI and NAND, specific devices are tested and supported.

IMPORTANT: *If you use NAND as the primary boot device, only use NAND devices from a vendor that guarantees screening for zero data corruption on the first parameter page.*

Table 11-1 describes various boot modes. All modes can be non-secure. All modes can be secure and signed except PS JTAG and PJTAG.

Table 11-1: **Boot Modes**

Boot Mode	Mode Pins [3:0]	Pin Location	CSU Mode	Description
PS JTAG	0000	JTAG	Slave	PSJTAG interface, PS dedicated pins.
Quad-SPI (24b)	0001	MIO[12:0]	Master	24-bit addressing (QSPI24).
Quad-SPI (32b)	0010	MIO[12:0]	Master	32-bit addressing (QSPI32).
SD0 (2.0)	0011	MIO[25:21, 16:13]	Master	SD 2.0.
NAND	0100	MIO[25:09]	Master	Requires 8-bit data bus width.
SD1 (2.0)	0101	MIO[51:43]	Master	SD 2.0.
eMMC (1.8V)	0110	MIO[22:13]	Master	eMMC version 4.5 at 1.8V.
USB0 (2.0)	0111	MIO[52:63]	Slave	USB 2.0 only.
PJTAG (MIO #0)	1000	MIO[29:26]	Slave	PJTAG connection 0 option.
PJTAG (MIO #1)	1001	MIO[15:12]	Slave	PJTAG connection 1 option.
SD1 LS (3.0)	1110	MIO[51:39]	Master	SD 3.0 with a required SD 3.0 compliant voltage level shifter.

Quad-SPI (24b/32b): The BootROM code can boot Quad-SPI using 24- or 32-bit addressing using the configurations shown in Table 24-1.

The QSPI boot mode size limit and image search limit are listed in Table 11-2. Image search for multi-boot is supported in this boot mode. The QSPI boot mode also supports x1, x2 and x4 read modes for single Quad-SPI memory and x8 for a dual QSPI. This is the only boot mode that supports execute-in-place (XIP).

Table 11-2 shows the boot modes supporting image search along with the search offset limit.

Table 11-2: **Boot Image Search Limits**

Boot Mode	Search Offset Limit
QSPI: 24-bit single	16 MB
QSPI: 24-bit dual parallel	32 MB
QSPI: 32 bit	256 MB
QSPI: 32-bit dual parallel	512 MB
NAND	128 MB
SD/eMMC	8,191 files
USB	1 file

RECOMMENDED: *Xilinx recommends using the QSPI 32-bit boot mode for flash sizes larger than 16 MB and when the flash supports 32-bit addressing.*

RECOMMENDED: *Xilinx recommends that verifying the QSPI commands supported by a specific flash memory. The CSU ROM supports the QSPI commands listed in Table 11-3.*

Table 11-3: **QSPI Command Codes**

Quad-SPI Data Interface	Read Mode	Command Code
24-bit single	Normal read	`0x03`
24-bit dual	Output fast read	`0x3B`
24-bit quad	Output fast read	`0x6B`
32-bit boot	Normal read	`0x13`
32-bit dual	Output fast read	`0x3C`
32-bit quad	Output fast read	`0x6C`

NAND: The NAND boot mode only supports 8-bit widths for reading the boot images. Image search for multi-boot is supported. Boot mode image search limits are listed in Table 11-2.

SD0/SD1: These boot modes support FAT 16/32 file systems for reading the boot images. Image search for multi-boot is supported. The maximum number of files that can be searched as part of an image search for multi-boot are 8,191. The SD supported version is 2.0, which only supports 3.3V for the I/Os and up to 4 bits of data interface.

SD1(LS): The SD1-LS boot mode is the same as SD0/SD1 with additional support of the SD 3.0 (with an SD 3.0 compliant voltage level shifter).

eMMC(18): This boot mode is the same as the SD boot mode except it only supports 1.8V for the I/Os and up to 8 bits of data interface. The eMMC mode is used for eMMC interfacing and the SD0/1 mode is used for SD card only. The application must switch to the high-speed modes.

TIP: *For SD and eMMC boot modes, the boot image file should be at the root of first partition of the SD card (not inside any directory).*

USB0: The USB boot mode configures USB controller 0 into device mode and uses the Device Firmware Upgrade (DFU) protocol to communicate with an attached host. See the "Boot Sequence for USB Boot Mode" section in *Zynq UltraScale+ MPSoC: Embedded Design Tutorial* (UG1209) [Ref 17] for more information.

The USB host contains the FSBL boot image (e.g., boot.bin) that is loaded into OCM memory for the CSU BootROM code and an all encompassing boot image file (e.g., boota53_all.bin) that is loaded into DDR memory.

The size of these files are limited by the size of the OCM and DDR memories. The USB boot mode does not support multi-boot, image fallback, or XIP.

Note: USB Timeout Condition - When the Zynq UltraScale+ MPSoC powers up in the USB boot mode, the USB host can download the boot image to the Zynq UltraScale+ MPSoC memory through the USB interface in DFU protocol. However during a time out of approximately five minutes in CSU ROM code, no image is downloaded and the host will not able to locate the DFU device with DFU utility.

Golden Image Search

The BootROM can search for a valid boot header to load and run a boot image. To validate a boot header, the BootROM 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 boot header and the rest of the boot image (including the FSBL) are loaded into the RPU or APU memory for further processing.

Boot images can be located every 32 KB in the boot memory device, which allows for more than one boot image to be in the memory device.

If an image header is invalid, the BootROM increments the image header address register by 32 KB and tries again. The boot image search mechanism is only available for the Quad-SPI, NAND, SD, and eMMC boot modes.

If a boot header is valid, but the FSBL determines the boot image is corrupt, the FSBL can recover by writing the location of another boot header into the CSU.csu_multi_boot register and issuing a system reset (not a POR). Figure 11-1 illustrates the image search mechanism.

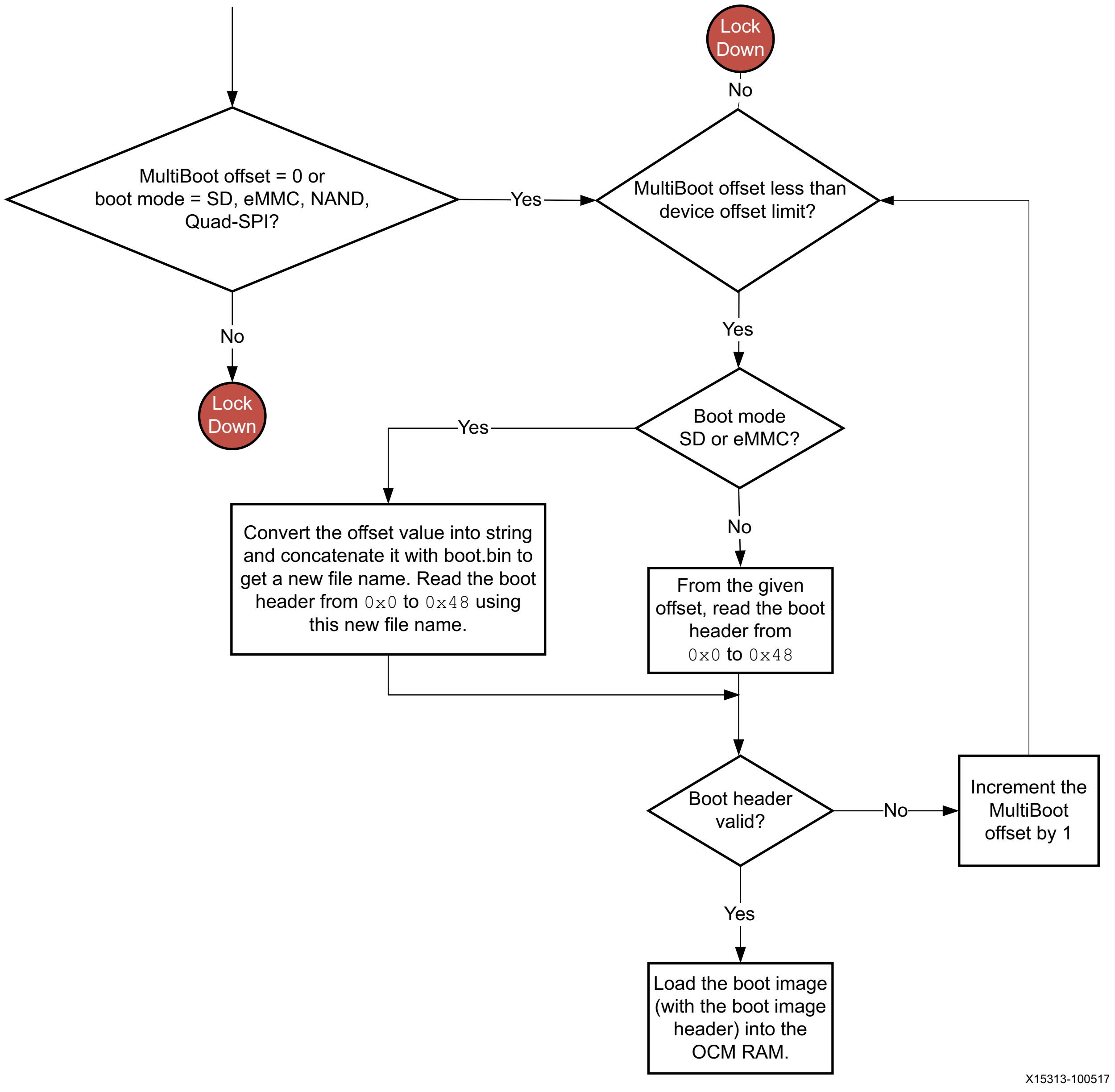

Lock Down
No
MultiBoot offset = 0 or boot mode = SD, eMMC, NAND, Quad-SPI?
Yes
MultiBoot offset less than device offset limit?
No
Lock Down
Yes
Boot mode SD or eMMC?
Yes
Convert the offset value into string and concatenate it with boot.bin to get a new file name. Read the boot header from 0x0 to 0x48 using this new file name.
No
From the given offset, read the boot header from 0x0 to 0x48
Boot header valid?
No
Increment the MultiBoot offset by 1
Yes
Load the boot image (with the boot image header) into the OCM RAM.
X15313-100517

Figure 11-1: **Image Search Flowchart**

Fallback

Using the FSBL, a fallback boot image can be loaded by loading the address of another boot header into the CSU.csu_multi_boot register and issuing a system reset (not a POR).

After the system reset, the boot header is fetched from the address location equal to the value of csu_multi_boot register times 32,768.

If the fallback boot header is invalid, the CSU continues normally with its boot image search function if the boot device supports image search. The BootROM header is described in Table 11-4.

Fallback and MultiBoot Flow In the Zynq UltraScale+ MPSoC device, the CSU bootROM supports MultiBoot and fallback boot image search where the configuration security unit CSU ROM or bootROM searches through the boot device looking for a valid image to load.

The sequence is as follows:

- BootROM searches for a valid image identification string (as image ID) at offsets of 32 KB in the flash.
- After finding a valid identification value, validates the checksum for the header.
- If the checksum is valid, the bootROM loads the image.

This allows for more than one image in the flash. In MultiBoot:

- CSU ROM or FSBL or the user application must initiate the boot image search to choose a different image from which to boot.
- To initiate this image search, CSU ROM or FSBL updates the MultiBoot offset to point to the intended boot image and generates a soft reset by writing into the CRL_APB register.

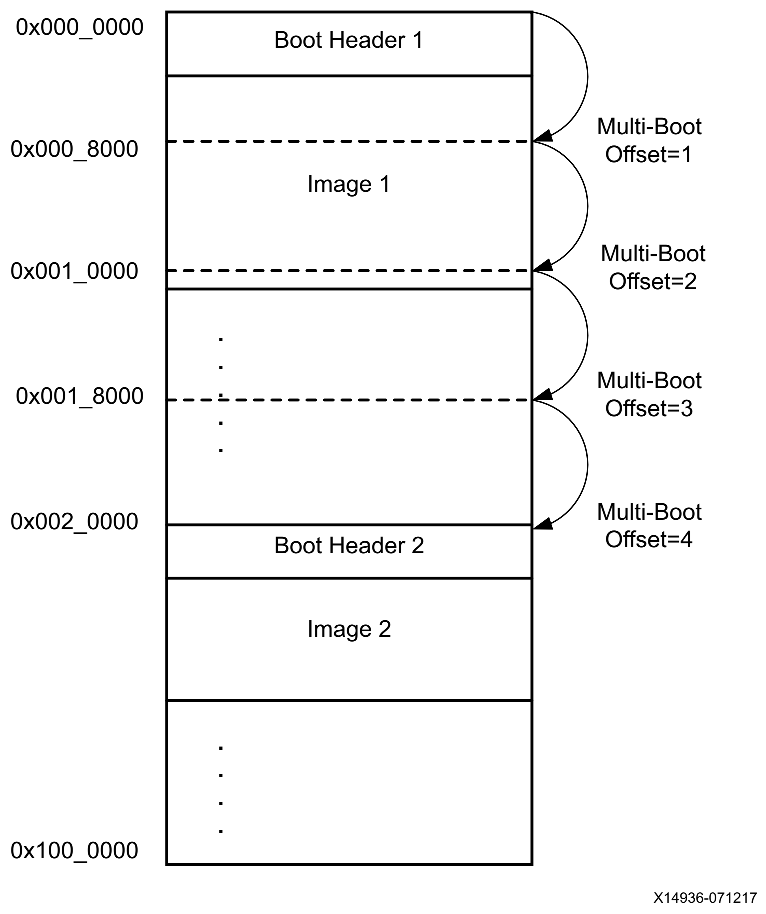


Figure 11-2: **MultiBoot Flow**

Note: The same flow is applicable to both Secure and non-secure boot methods.

In the example fallback boot flow figure, the following sequence occurs:

- The CSU bootROM loads the boot image found at 0x000_0000.
- If this image is found to be corrupted or the decryption and authentication fails, CSU bootROM increments the MultiBoot offset by 1 and searches for a valid boot image at 0x000_8000 (32 KB offset).
- If the CSU bootROM does not find the valid identification value, it increments the MultiBoot offset by 1 again and searches for a valid boot image at the next 32 KB aligned address.
- The CSU bootROM repeats this until a valid boot image is found or the image search limit is reached. In this example flow, the next image is shown at 0x002_0000 corresponding to a MultiBoot offset value of 4.

- I In Figure 11-2, the MutiBoot offset is updated to 4 by FSBL/CSU-ROM to load the second image at the address 0x002_0000. When the MultiBoot offset is updated, soft reset the system.

Table 11-2 shows the MultiBoot image search range for different booting devices.

Boot Image Format

Because the CSU ROM supports the MultiBoot option, there can be more than one boot image in a boot device. The boot image consists of a boot header and partitions for different images along with a partition header. Figure 11-3 shows the simplest form of a boot image with only a mandatory image partition (FSBL) with associated mandatory headers. A detailed secure image format is illustrated in Table 12-17.

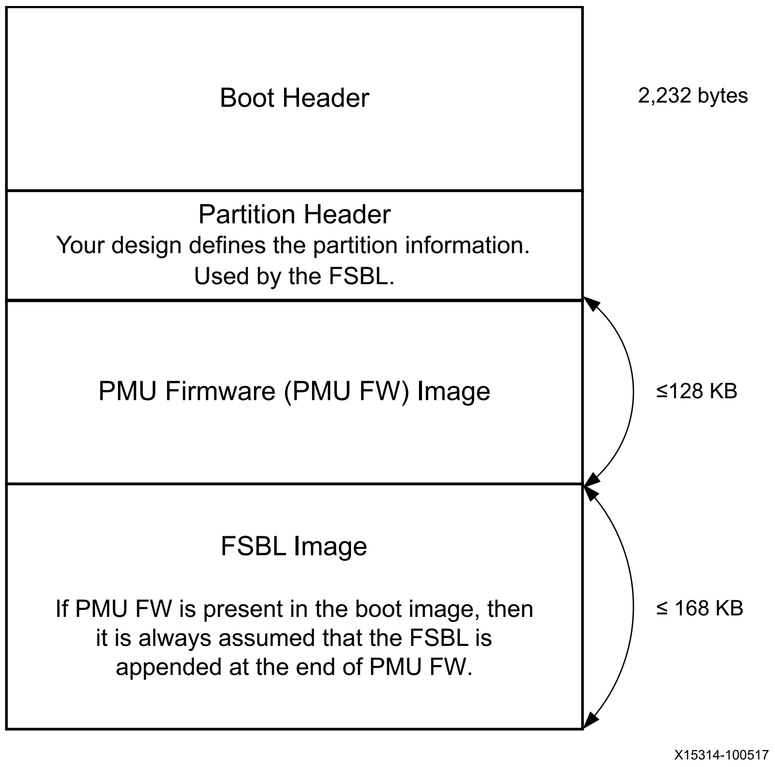

Boot Header
2,232 bytes
Partition Header
Your design defines the partition information.
Used by the FSBL.
PMU Firmware (PMU FW) Image
≤128 KB
FSBL Image
If PMU FW is present in the boot image, then it is always assumed that the FSBL is appended at the end of PMU FW.
≤ 168 KB
X15314-100517

Figure 11-3: **Boot Image Format with FSBL and PMU Firmware**

With secure boot, the authentication certificate follows the FSBL image. Both the boot header and partition header are always in plain text.

The boot header format is shown in Table 11-4. This is a plain-text header associated with each boot image that indicates various characteristics, attributes (Table 11-5), and other details about that boot image.

Table 11-4: **Boot Header Format**

Offset	Description	Details
`0x000 - 0x01C`	Reserved for interrupts	This field is used in case of XIP boot mode when the default `0x01F` interrupt vectors are changed in the LQSPI address space.
`0x020`	Width detection	Quad-SPI width description.
`0x024`	Image identification	Boot image identification string.
`0x028`	Encryption status	This field is used to identify the AES key source. `0000_0000h`: Unencrypted. `3A5C_3C5Ah`: Red key in BBRAM. `A35C_7CA5h`: Obfuscated key in boot header. `A35C_7C53h`: Black key in boot header. `A5C3_C5A3h`: Red key in eFUSE. `A5C3_C5A5h`: Black key in eFUSE (PUF key). `A5C3_C5A7h`: eFUSE (Gray key). `A3A5_C3C5h`: User key. ***Note:*** The user key is only used with single partition boot images.
`0x02C`	FSBL execution address	FSBL execution start address.
`0x030`	Source offset	PMU FW and FSBL source start address.
`0x034`	PMU FW image length	PMU FW original image length.
`0x038`	Total PMU FW image length	PMU FW total image length. This includes the complete PMU firmware image block size, AES key, AES IV, and GCM tag (in case of an encrypted image). This field size must be ≤128 KB.
`0x03C`	FSBL image length	FSBL original image length.
`0x040`	Total FSBL image length	Total FSBL image length.
`0x044`	Image attributes	Image attributes are described in Table 11-5.
`0x048`	Header checksum	Header checksum from `0x20` to `0x44`.
`0x04C–0x068`	Obfuscated key	256-bit obfuscated key. Only valid when `0x028` (encryption status) is `A35C_7CA5h`.
`0x06C`	Reserved	
`0x070–0x09C`	FSBL/User defined	How to use the FSBL/user defined areas is explained in the *Zynq UltraScale+ MPSoC Software Developer's Guide* (UG1137) [Ref 3].
`0x0A0–0x0A8`	Secure header initialization vector	Initialization vector for a secure header for both PMU FW and FSBL.
`0x0AC–0x0B4`	Obfuscated or black key initialization vector	Initialization vector used when decrypting the obfuscated key.
`0x0B8–0x8B4`	Register initialization	Store register write pairs for system register initialization.
`0x8B8-0xEC0`	PUF helper data	Store the PUF helper data. The helper data is used only when the image attribute PUF HD location = `0x3`.

I/O Configuration Detection

The BootROM can detect the intended I/O width of the Quad-SPI interface using the width detection parameter value (0xAA995566) and the image identification parameter value (0x584C4E58) in a 8- bit parallel configuration.

4-bit I/O Detection

During the Quad-SPI boot process, the BootROM configures the controller with 4-bit I/O. This configuration includes a single device and the dual 4-bit stacked case. The BootROM reads the first (or only) Quad-SPI device in x1 mode and reads the width detection parameter in the BootROM Header. If the width detection parameter is equal to 0xAA995566, then the BootROM assumes it found a valid header that is requesting a 4-bit I/O configuration. It might be one device or it might be a dual 4-bit stacked configuration. In the latter case, the second device is always ignored by the BootROM, but it might be accessed by user code. After reading the width detection parameter in x1 mode, the BootROM attempts to read the parameter in x4 mode. If x4 mode fails, it tries x2 mode. After this, the BootROM uses the widest supported I/O bus width to access the Quad-SPI device.

8-bit I/O Detection

The BootROM also looks for the dual device, 8-bit parallel configuration. In this case, the BootROM only reads the even bits of the BootROM header because it is only accessing the first device and the header is split across both devices. The BootROM forms a 32-bit word that includes the even bits of the width detection (0x20) and image identification (0x24) parameter values. When the BootROM detects this condition, it assumes the system uses the 8-bit parallel configuration and programs the controller for the x8 operating mode. This mode is used for the rest of the boot process.

Table 11-5: **Image Attributes Offset Definition**

Field Name	Bit Offset	Width	Default Value	Description
Reserved	31:16	16	`0x0`	
Bhdr RSA	15:14	2	`0x0`	`0x3`: If the RSA_EN eFUSEs are not programmed, RSA authentication of the boot image is done, excluding verification of PPK hash and SPK ID. If the RSA_EN eFUSEs are programmed, then an error is generated. All others: RSA authentication is decided based on the RSA_EN eFUSEs
SHA2 select	13:12	2	`0x0`	`0x3`: While doing RSA authentication, SHA2 is used in place of SHA3.(1) All others: SHA3 is used while doing RSA authentication.

Table 11-5: **Image Attributes Offset Definition *(Cont'd)***

Field Name	Bit Offset	Width	Default Value	Description
CPU select	11:10	2	`0x0`	`0x0`: Cortex-R5F single (split mode). `0x1`: Cortex-A53 single 32-bit. `0x2`: Cortex-A53 single 64-bit. `0x3`: Cortex-R5F dual (lock-step mode).
Hashing select	9:8	2	`0x0`	`0x0`, `0x1`: No integrity check. `0x2`: SHA2 is used as a hash function to do boot image integrity check.(1) `0x3`: SHA3 is used as a hash function to do boot image integrity check. ***Note:*** This option should not be selected if authentication (RSA) is used. If the RSA_EN eFUSEs are programmed and this option is set, an error occurs.
PUF HD location	7:6	2	`0x0`	`0x3`: PUF HD is part of the boot header. All others: Means PUF HD is in eFUSE.
Authenticate only	5:4	2	`0x0`	`0x3`: Boot image is only RSA signed, do not decrypt the image even if `0x28` offset is non-zero. All others: Means if `0x28` is non-zero, then decrypt the boot image.
OP key	3:2	2	`0x0`	`0x3`: Secure header contains operational key for block 0 decryption. All others: Means that the root device key is used for block 0 decryption.
Reserved	1:0	2	`0x0`	

Notes:

1. Xilinx recommends using SHA3 only. SHA2 will be deprecated in 2019.1.

Functional Units

Figure 12-1 describes the CSU. The CSU consists of processor, security blocks, CSU DMA, secure stream switch, and PCAP. The CSU processor and security blocks are described in Chapter 12, Security.

Secure Stream Switch

The secure-stream switch (SSS) allows data movement between multiple sources and destinations. During boot, the secure-stream switch is exclusively controlled by the CSU. After boot, any system master can control the configuration of the secure-stream switch. Table 11-6 lists the possible connections in the secure stream switch.

The JTAG PS TAP controller is accessible via the dedicated PS pins. The AXI DMA is in the CSU.

Table 11-6: **Secure Stream Switch**

		Destinations				
		AXI DMA	**JTAG**	**AES-GCM**	**PCAP**	**SHA**
Sources	AXI DMA	X		X	X	X
	JTAG	X			X	
	AES-GCM	X			X	
	PCAP	X	X			
	ROM					X

The secure-stream switch is configured using a single SSS configuration register (csu_sss_cfg). Some common configurations for the secure stream switch are listed in Table 11-7.

Table 11-7: **Secure Stream Switch Configurations**

Secure Stream Switch Setup	**Description**	**CSU_SSS_CFG Setting**
DMA to DMA	DMA loopback.	`0x00000050`
DMA to PCAP	PL configuration.	`0x00000005`
DMA to AES, AES to DMA	Secure PS configuration.	`0x000005A0`
DMA to AES, AES to PCAP	Secure PL configuration.	`0x0000050A`
DMA to DMA, DMA to SHA	PS image load with simultaneous SHA calculation.	`0x00005050`

CSU DMA

The CSU DMA allows the CSU to move data efficiently between the memory and the CSU stream peripherals (AES, SHA, PCAP), using the secure stream switch. The CSU DMA can access the OCM, TCM, and DDR memory. The CSU DMA is a two-channel, simple DMA, allowing separate control of the SRC (read) channel and DST (write) channel with a 128 x 32-bit data FIFO for each channel. The DMA is effectively able to transfer data.

- From the PS-side to the secure stream switch (SSS) side (SRC DMA only).
- From the SSS-side to the PS-side (DST DMA only).
- Simultaneously from the PS-side to the SSS-side and from the SSS-side to the PS-side.

The APB interface allows for control and monitoring of the CSU DMA module's functions. A single interrupt output port is sent to the CSU. It is combined with other interrupt sources before being sent out to the interrupt controller on a single interrupt pin.

Two clocks are provided, one for the main CSU DMA operation and one for the APB interface. Along with these clocks, there are two-reset inputs. These reset pins are synchronized to the respective clock domains by the CSU before sending them to the CSU DMA.

The DMA interfaces with a secure-stream switch through two sets of handshake signals; one for the DMA SRC (memory to stream) direction and the other for the DMA DST (stream-to-memory) direction.

The DMA has a DST_FIFO that is sized to hold a minimum of one PL configuration frame. Although overflow is not anticipated, an interrupt register (FIFO_OVERFLOW) is provided in cases where an overflow occurs.

Loopback Mode

Loopback is implemented in the secure-stream switch hardware. It is not internal to the CSU DMA. The CSUDMA.CSUDMA_DST_CTRL [SSS_FIFOTHRESH] bit field controls the level of the DST FIFO to result in asserting the data_out_fifo_level_hit signal on the DST interface. This can be used to flow control data between the SRC and DST FIFOs in loopback mode. If loopback mode is used, where SRC data is looped around in the secure-stream switch and presented to the DST channel, the software should always start the DST channel before starting the SRC channel. This ensures that the DST channel is always ready once the first piece of data present at its secure-stream switch interface. Refer to the Programming the CSU DMA section for details on the CSU DMA programming sequence.

PL Configuration

The processor configuration access port (PCAP) is used to configure the programmable logic (PL) from the PS. The PCAP is the only interface used to configure the PL during normal operating conditions. The PCAP bus is 32 bits wide. During debug, the JTAG interface can be used to configure the PL. The PS is connected to the PCAP through the secure-stream switch. Bitstream data can be sent to the PL using either the CSU DMA or the AES path.

PCAP Isolation Wall Control

Software should disable the PS-PL isolation wall before the PL is configured with its bitstream.

CSU BootROM Error Codes

Any error from the CSU while in the boot process is recorded in the PMU_GLOBAL.CSU_BR_ERR register. This register also determines the boot success or failure. On failure or error condition, the 16-bit error code is recorded in the CSU_BR_ERR register.

A configuration BootROM error code is 8 bits long. This means that with an allocation of 16 bits, two error codes are stored. Error bits [15-8] correspond to the first image error code and error bits [7-0] indicate the most recent image error code (Table 11-8).

Table 11-8 describes the configuration BootROM(CBR) error codes.

Table 11-8: **BootROM Error Bits**

15 8	7 0
First Image Error	Last Image Error

Table 11-9 describes the configuration bootRAM (CBR) error codes.

Table 11-9: **BootROM Error Codes**

Error Code	Description	Solution
`0x10`	Secure processor voting has failed during boot.	Ensure LPD power supply is correct and power on reset (POR) the chip.
`0x11`	Secure processor is unable to power up the OCM.	Ensure LPD power supply is correct and POR the chip.
`0x12`	An error occurred while initializing the OCM with `0xDEADBEEF` value.	Ensure LPD power supply is correct and POR the chip.
`0x14`	eFUSE is not properly loaded by hardware. There is a parity error in eFUSE values.	Ensure LPD power supply is correct and (POR) the chip.
`0x15`	The TBITs in eFUSE are not properly written. For a successful boot, TBITS in eFUSE should have either all zeros or a `1010b` pattern.	Check the TBITS values and POR the chip.
`0x16`	DMA transfer timeout error during the OCM initialization.	Ensure LPD power supply is correct and (POR) the chip.
`0x17`	eFUSE controller is unable to load the eFUSE values to the cache registers.	Ensure LPD power supply is correct and (POR) the chip.
`0x20`	eFUSE RSA bits read from eFUSE has a mismatch.	
`0x23`(1)	Error occurred during QSPI 24 boot mode initialization.	Check that the Quad-SPI device is properly connected to the QSPI MIO pins. Ensure the width detection word is set equal to the data pattern 0xAA995566 and that the image identification word has x584C4E58, 'XLNX'. Ensure that the device content is not blank in single device applications.
`0x24`	Error occurred during QSPI 32 boot mode initialization.	Check that the Quad-SPI device is properly connected to the QSPI MIO pins. Ensure the width detection word is set equal to the data pattern 0xAA995566 and that the image identification word has x584C4E58, 'XLNX'. Ensure that the device content is not blank.

Table 11-9: **BootROM Error Codes *(Cont'd)***

Error Code	Description	Solution
`0x25`	Error occurred during NAND boot mode initialization.	Check that the NAND device is properly connected to the NAND MIO pins. Ensure the width detection word is set equal to the data pattern 0xAA995566 and that the image identification word has x584C4E58, 'XLNX'. Ensure that the device content is not blank.
`0x26`	Error occurred during SD boot mode initialization.	Check that the SD device is properly connected to the SD MIO pins. Ensure SD card is formatted properly with FAT32/FAT16 file system.
`0x27`	Error occurred during eMMC boot mode initialization.	Check that the eMMC device is properly connected to the eMMC MIO pins. Ensure that the MMC card is formatted properly with the FAT32/FAT16 file system.
`0x2A`	Invalid boot mode is selected in the boot mode setting.	Ensure the boot mode pins values are valid.
`0x30`	Boot header does not have an XLNX string.	Ensure that the Image Identification word has x584C4E58, 'XLNX' in the Boot header.
`0x31`	Boot header checksum is wrong or boot header fields are not length aligned.	Ensure that the boot header checksum is correctly calculated and written to flash device. Also, ensure all the length fields are word aligned.
`0x32`	Boot header encryption status value is not valid. Key selected is not a valid key source.	Ensure that the key source value in Encryption Status field is valid.
`0x33`	Boot header attributes value is not valid. Reserved fields in image attributes are not zero.	Ensure that the reserved field is ZERO in the Image attributes.
`0x34`	Either of the boot header PMU firmware length and total PMU firmware length fields are not valid.	Ensure the PMU firmware length fields are in valid range. Ensure PMU firmware length is more than the total PMU firmware length.
`0x35`	Either of the boot header FSBL and total FSBL length fields are not valid.	Ensure the FSBL length fields are in valid range. Ensure the FSBL length is more than the total FSBL length.
`0x36`	Selected does not support the XIP mode.	Ensure the boot mode pins are set QSPI. Only non-secure images are allowed in XIP mode. Ensure image is secure.
`0x37`	FSBL execution address is not in the OCM address range.	Ensure the FSBL execution address in the OCM 256K region.
`0x38`	Source offset is not valid. It is beyond the flash image search limit.	Check if the offset in the flash is beyond the flash image search limit. The offset is calculated from the multi boot register value and source offset field in boot header.

Table 11-9: **BootROM Error Codes *(Cont'd)***

Error Code	Description	Solution
`0x3A`	Authentication only is selected, but no key source is selected (for selecting the device key source) or authentication only is selected, but no authentication is selected in eFUSE or the boot header.	To use the authentication only feature, the encryption key shall be selected so that ROM can unlock that key. Authentication of the image is mandatory. Enable eFUSE or boot header authentication.
`0x3B`	Reading failed from the selected boot device.	Ensure the boot device is properly connected and right boot mode is selected.
`0x3D`	Selected CPU is disabled in the eFUSE.	Ensure A53 CPU is enabled for the chip when A53-0 is selected as a hand off CPU.
`0x3E`	Time out occurred while calculating the PPK hash.	Ensure power supply are proper and POR the chip.
`0x3F`(2)	Boot with boot header helper data is not allowed until eFUSE helper data is invalidated.	To use the PUF helper data it is necessary to set the SYN_INVALID bit in eFUSE.
`0x40`	Boot header and eFUSE RSA are enabled at the same time, which is not allowed.	The boot header and eFUSE RSA should not be enabled at the same time. Ensure the RSA eFUSE is not blown when boot header RSA is selected.
`0x41`	Selected PPK value in boot header is not valid.	ROM supports 2 PPK keys. Ensure the proper key value is used in the Boot header.
`0x42`	Selected PPK is revoked.	Ensure the selected PPK is not revoked in eFUSE.
`0x43`	All PPK in the device are revoked.	All PPK present inside the chip are revoked. Authentication cannot be performed on this chip.
`0x44`	Mismatch in the PPK hash calculated from Boot header and PPK hash in eFUSE	Ensure the correct PPK is burned inside the eFUSEs. Ensure the PPK keys used to create boot image match the RSA public key that is present inside the chip.
`0x45`	SPK signature verification is failed	Ensure the SPK signature is created with the PPK that is present in the authentication certificate. Ensure the SPK ID present in the boot header is same as eFUSE SPK ID.
`0x46`	Selected SPK ID is not matching with the eFUSE SPK ID.	Ensure the SPK ID present in the boot header is same as the eFUSE SPK ID.
`0x47`	Boot header signature is failed.	Ensure the boot header signature is created with the SPK that is present in the authentication certificate.
`0x48`	Selected boot mode does not support the golden image search.	Golden Image search is supported only by QSPI, NAND, and SD boot modes. For all other boot modes, the multi boot register value should be zero.

Table 11-9: **BootROM Error Codes *(Cont'd)***

Error Code	Description	Solution
`0x49`	No image found in QSPI after searching the allowed address range.	ROM reached end of the image search limit for a QSPI device and no other good image is found. Ensure image in the QSPI is valid.
`0x4A`	No image found in NAND after searching the allowed address range.	ROM reached end of the image search limit for a NAND device and no other good image is found. Ensure the image in the NAND is valid.
`0x4B`	No image found in the SD/eMMC after searching the allowed number of files.	ROM reached end of the file limit for a SD/eMMC device and no other good image is found. Ensure the boot file is valid in the SD/eMMC FAT file system.
`0x4D`	Time out error while calculating the SPK SHA hash.	Ensure the LPD power supply is correct and POR the chip.
`0x4E`	Time out error while calculating the boot header SHA hash.	Ensure the LPD power supply is correct and POR the chip.
`0x50`	Mismatch while writing to the secure registers.	Ensure the LPD power supply is correct and POR the chip.
`0x51`	Changing the state of the device from secure to non-secure is not allowed.	After POR, if first image is secure then subsequent images are secure.
`0x52`	Changing the key source is not allowed while in the secure state.	Ensure the key source is the same across multiple images used for boot in POR and SRST.
`0x53`	Changing the state from non-secure to secure is not allowed.	Ensure the key source is the same across multiple images used for boot in POR and SRST.
`0x54`	BBRAM key is disabled in eFUSE but the key source selected is BBRAM.	Ensure the BBRAM_DIS eFUSE bit is not set when BBRAM is selected as a key source.
`0x55`	Only encrypted boots with the eFUSE key source are allowed.	When ENC_ONLY eFUSE bit is set, no other key sources are allowed apart from eFUSE.
`0x60`	One of the register addresses in the boot header is not allowed.	Ensure the register addresses in the boot header are in the valid address range allowed by ROM.
`0x61`	Copying from selected boot device failed after register initialization.	Ensure the frequencies or any modifications done to boot devices through register initialization are correct.
`0x62`	Boot header read after register initialization is mismatched with the original boot header.	Ensure the frequencies or any modifications done to boot devices through register initialization are correct. Data read after register initialization does not match the previous values.

Table 11-9: **BootROM Error Codes *(Cont'd)***

Error Code	Description	Solution
`0x70`	Error occurred while copying the PMU FW.	Ensure the boot device is connected correctly. Ensure the frequencies or any modifications done to boot devices through register initialization are correct.
`0x71`	Error occurred while copying the FSBL.	Ensure the boot device is connected correctly. Ensure the frequencies or any modifications done to boot devices through register initialization are correct.
`0x72`	Time out occurred while loading the key.	Ensure LPD power supply is correct and POR the chip.
`0x73`	Time out occurred while using the CSU DMA in image processing for AES/SHA.	Ensure LPD power supply is correct and POR the chip.
`0x74`	Time out occurred for the PMU to go to sleep.	
`0x75`	Time out occurred while calculating the SHA during boot image signature verification.	Ensure LPD power supply is correct and POR the chip.
`0x76`	Time out occurred while calculating the SHA for boot image during the integrity check.	Ensure LPD power supply is correct and POR the chip.
`0x78`	Boot image signature mismatch occurred.	Ensure the boot image signature is created with the SPK that is present in authentication certificate.
`0x79`	Error occurred while decrypting the PMU firmware.	Ensure same keys are used as present in the chip for encrypting the Boot image. Ensure the right key source is used for encrypting the image.
`0x7A`	Error occurred while decrypting the FSBL.	Ensure the same keys are used as present in the chip for encrypting the Boot image. Ensure right key source is used for encrypting the image.
`0x7B`	Mismatch in the hash while checking for the boot image integrity.	Ensure the SHA3 is used while creating the boot image integrity.
`0x80`	Unable to power up the selected CPU.	Ensure the power rail is ON for the selected CPU. FPD in case of A53.
`0x81`	Unable to wake up the PMU after loading the PMU firmware.	Ensure the PMU Firmware is running properly and set FW_IS_PRESENT bit after completing initialization.
`0x90`	Tamper event that is detected while in post boot. Every tamper event is stored in an error register with `0x90` + Index. Index is the tamper event ID according to the CSU tamper register.	This is tamper detection by ROM during post boot. Ensure all voltages are in range. Ensure JTAG is not toggled when it is disabled.
`0xA0`	Error when selected boot mode does not support fallback.	Ensure no error is present in the first image during USB boot mode.
`0xB0`	Error when exceptions occurs while booting.	Ensure the LPD power supply is proper and POR the chip.

Table 11-9: **BootROM Error Codes *(Cont'd)***

Error Code	Description	Solution
`0xB1`	Error when exceptions occurs while in post boot.	Ensure the LPD power supply is proper and POR the chip.

Notes:

1. To use QSPI32 boot mode for flash sizes greater than 16 MB.
2. Once the SYN_INVALID bit is set in eFUSE, the system does not allow to boot with eFUSE PUF helper data.

PL Bitstream

The PL bitstream contains configuration data for the device's Programmable Logic (PL). This configuration data can be loaded and read back from the PS using the PCAP interface. The FSBL can handle loading of the initial configuration of the PL using a bitstream stored in the boot image. Alternatively, the PL bitstream can also be loaded at a later time by application code, including using software such as U-Boot or Linux. The XilFPGA software library provides an API that facilitates the loading and read back of configuration data to and from the PL. More information on the XilFPGA library can be found in the *Zynq UltraScale+ MPSoC Software Developer's Guide* (UG1137) [Ref 3]. The PL bitstream length and composition depend on the device. Table 11-1 lists the attributes and values of the bitstream for device type. Although bitstream options such as compression (BITSTREAM.GENERAL.COMPRESS) can alter the required bitstream length and Since compressed bitstreams can change in size between design iterations, adequate memory should be reserved for the full uncompressed bitstream.

Table 11-10: **PL Bitstream Length**

Device	Configuration Bitstream Length (bits)	Minimum Configuration Flash Memory Size (Mb)	Configuration Frames	Frame Length in Words	Configuration Array Size in Words	Configuration Overhead in Words
ZU1	23,748,480	32	7980	93	742,140	515
ZU2	44,549,344	64	14,964	93	1,391,652	515
ZU3	44,549,344	64	14,964	93	1,391,652	515
ZU4	61,269,888	64	20,956	93	1,948,939	515
ZU5	61,269,888	64	20,956	93	1,948,939	515
ZU6	212,086,240	256	71,260	93	6,627,180	515
ZU7	154,488,736	256	51,906	93	4,827,258	515
ZU9	212,086,240	256	71,260	93	6,627,180	515
ZU11	188,647,264	256	63,384	93	5,894,712	515
ZU15	229,605,952	256	77,147	93	7,174,671	515
ZU17	290,744,896	512	97,691	93	9,085,263	515
ZU19	290,744,896	512	97,691	93	9,085,263	515

Table 11-10: **PL Bitstream Length *(Cont'd)***

Device	Configuration Bitstream Length (bits)	Minimum Configuration Flash Memory Size (Mb)	Configuration Frames	Frame Length in Words	Configuration Array Size in Words	Configuration Overhead in Words
ZU21	275,498,848	512	92,568	93	8,608,824	515
ZU25	275,498,848	512	92,568	93	8,608,824	515
ZU27	275,498,848	512	92,568	93	8,608,824	515
ZU28	275,498,848	512	92,568	93	8,608,824	515
ZU29	275,498,848	512	92,568	93	8,608,824	515
ZU39	275,498,848	512	92,568	93	8,608,824	515
ZU42	166,864,320	256	56070	93	5,214,510	515
ZU43	275,498,848	512	92,568	93	8,608,824	515
ZU46	275,498,848	512	92,568	93	8,608,824	515
ZU47	275,498,848	512	92,568	93	8,608,824	515
ZU48	275,498,848	512	92,568	93	8,608,824	515
ZU49	275,498,848	512	92,568	93	8,608,824	515
ZU65	166,864,320	256	56070	93.	5,214,510	515
ZU67	166,864,320	256	56070	93	5,214,510	515

Note: Compressed bitstreams can be also encrypted and authenticated.

The allowed register accesses depend on the boot mode and are listed in Table 11-11.

Table 11-11: **Register Access Range and Boot Mode**

Control Register	Ranges	Secure Boot Mode
ACPU_GIC	0xF9000000 to 0xF900FFFC	Yes
SATA	0xFD070000 to 0xFD0C00FC	Yes
PCIE	0xFD0E0000 to 0xFD0EFFFC	Yes
CRF_APB	0xFD1A0000 to 0xFD1A001C	Yes
CRF_APB	0xFD1A0048 to 0xFD1A00F8	Yes
AFIFM_DP	0xFD360000 to 0xFD4AFFFC	Yes
APU	0xFD5C0000 to 0xFD5CFFFC	Yes
CCI_REG	0xFD5E0000 to 0xFD5EFFFC	Yes
FPD_SLCR	0xFD610000 to 0xFD61FFFC	Yes
FPD_GPV	0xFD6E0000 to 0xFD70FFFC	Yes
IOU_GPV	0xFE000000 to 0xFE10FFFC	Yes
LPD_GPV	0xFE000000 to 0xFE10FFFC	Yes
DPROM	0xFE800000 to 0xFF05FFFC	Yes

Table 11-11: **Register Access Range and Boot Mode *(Cont'd)***

Control Register	Ranges	Secure Boot Mode
SPI	0xFE800000 to 0xFF05FFFC	Yes
GPIO	0xFF0A0000 to 0xFF0AFFFC	Yes
QSPI	0xFF0F0000 to 0xFF0F01FC	Yes
NAND 0	0xFF100000 to 0xFF100020	Yes
NAND 1	0xFF100028 to 0xFF10004C	Yes
NAND 2	0xFF10005C to 0xFF10006C	Yes
TTC1 to TTC4	0xFF110000 to 0xFF14FFFC	Yes
SDIO 0	0xFF160004 to 0xFF160054	Yes
SDIO 0	0xFF160060 to 0xFF160100	Yes
SDIO1	0xFF170004 to 0xFF170054	Yes
SDIO1	0xFF170060 to 0xFF170100	Yes
IOU_SLCR	0xFF180000 to 0xFF18FFFC	Yes
LPD_SLCR	0xFF250000 to 0xFF41FFFC	Yes
CRL_APB	0xFF5E0000 to 0xFF5E009C	Yes
CRL_APB	0xFF5E00A4 to 0xFF5E01DC	Yes
RPU - AFIFM	0xFF9A0000 to 0xFF9BFFFC	Yes
APM – RTC	0xFFA00000 to 0xFFA6FFFC	Yes
MBISTJTAG	0xFFCF0000 to 0xFFCFFFFC	Yes

Register Overview

Table 11-12: **CSU Register Summary**

Register Type	Register Name	Description
Configuration security unit control	csu_status	CSU status.
	csu_ctrl	CSU control.
	csu_sss_cfg	CSU secure stream switch configuration.
	csu_dma_reset	CSU DMA reset.
	csu_multi_boot	MultiBoot address.
	csu_tamper_trig	CSU secure lockdown.
	csu_ft_status	CSU fault tolerant status.
	csu_isr	CSU interrupt status.
	csu_imr	CSU interrupt mask.
	csu_ier	CSU interrupt enable.
	csu_idr	CSU interrupt disable.
	jtag_chain_status	JTAG chain configuration status.
	jtag_sec	JTAG security.
	jtag_dap_cfg	DAP configuration.
	idcode	Device IDCODE.
	version	PS version.
ROM SHA digest	csu_rom_digest_{0:11}	CSU ROM SHA-3 digest 0 to 11.
AES control	aes_status	AES status.
	aes_key_src	AES key source.
	aes_key_load	AES key load.
	aes_start_msg	AES start message.
	aes_reset	AES reset.
	aes_key_clear	AES key clear.
	aes_kup_wr	AES KUP write control.
	aes_kup_{0:7}	AES key update 0 to 7.
	aes_iv_{0:3}	AES initialization vector 0 to 3.
SHA control	sha_start	SHA start message.
	sha_reset	SHA reset.
	sha_done	SHA done.
	sha_digest_{0:11}	SHA digest 0 to 11.

Table 11-12: **CSU Register Summary *(Cont'd)***

Register Type	Register Name	Description
PCAP control	pcap_prog	PCAP PROGRAM_B control.
	pcap_rdwr	PCAP read/write control.
	pcap_ctrl	PCAP control.
	pcap_reset	PCAP reset.
	pcap_status	PCAP status.
Tamper response	tamper_status	Tamper response status.
	csu_tamper_{0:12}	CSU tamper response 0 to 12.

Table 11-13: **CSU DMA Register Summary**

Register Type	Register Name	Description
CSU DMA source	csudma_src_addr	Source memory address (LSBs) for DMA memory → stream data transfer.
	csudma_src_size	DMA transfer payload for DMA memory → stream data transfer.
	csudma_src_sts	General source DMA status.
	csudma_src_ctrl	General source DMA control register 1.
	csudma_src_crc	Source DMA pseudo CRC.
	csudma_src_i_sts	Source DMA interrupt status.
	csudma_src_i_en	Source DMA interrupt enable.
	csudma_src_i_dis	Source DMA interrupt disable.
	csudma_src_i_mask	Source DMA interrupt mask.
	csudma_src_ctrl2	General source DMA control register 2.
	csudma_src_addr_msb	Source memory address (MSBs) for DMA memory → stream data transfer.

Table 11-13: **CSU DMA Register Summary *(Cont'd)***

Register Type	Register Name	Description
CSU DMA destination	csudma_dst_addr	Destination memory address (LSBs) for DMA stream → memory data transfer.
	csudma_dst_size	DMA transfer payload for DMA stream → memory data transfer.
	csudma_dst_sts	General destination DMA status.
	csudma_dst_ctrl	General destination DMA control.
	csudma_dst_i_sts	Destination DMA interrupt status.
	csudma_dst_i_en	Destination DMA interrupt enable.
	csudma_dst_i_dis	Destination DMA interrupt disable.
	csudma_dst_i_mask	Destination DMA interrupt mask.
	csudma_dst_ctrl2	General Destination DMA control register 2.
	csudma_dst_addr_msb	Destination memory address (MSBs) for DMA stream → memory data transfer.
Safety	csudma_safety_chk	Safety endpoint connectivity check register

Configuration Programming Model

Load the PL Bitstream

After executing CSU ROM code, the CSU hands off the control to the first-stage boot loader (FSBL). The FSBL uses the PCAP interface to configure the PL with the bitstream. Use the following steps to load the PL bitstream.

1. Initialize PCAP Interface.
2. Write a Bitstream Through the PCAP.
3. Wait for the PL Done Status.

The following section explains each of these steps.

Initialize PCAP Interface

1. Take the PCAP out of reset. Write 1 to the csu.pcap_reset[reset] bit.
2. Configure the PCAP in write mode.
    a. Select the PCAP mode. Write 1 to the csu.pcap_ctrl[pcap_pr] bit.
    b. Select write mode. Write 0 to the csu.pcap_rdwr[pcap_rdwr_b] bit.

3. Power up the PL, if needed. Read the csu.pcap_status [pl_gpwrdwn_b]. If Off, then trigger a request to the PMU to power up the PL using the pmu_global.req_pwrup_trig [PL] bit.
4. Reset the PL.
    a. Assert the PL reset. Write 0 to the csu.pcap_prog [pcfg_prog_b] bit.
    b. Wait for at least 250 ns.
    c. Deassert the PL reset. Write 1 to the csu.pcap_prog [pcfg_prog_b] bit.

Write a Bitstream Through the PCAP

1. Set the secure stream switch configuration to receive from DMA source: Set the csu.csu_sss_cfg[pcap_sss] to `0x5`.
2. Configure and set the CSU_DMA to establish channel and transfer. Use the following for CSU DMA programming (see the Programming the CSU DMA section for details).
    a. Channel type is DMA_SRC.
    b. Source address is the address of the bitstream.
    c. Size is bitstream size in words.
3. Wait for the CSU DMA operation to finish. on the source channel (see the Programming the CSU DMA section for details).
4. Clear the CSU_DMA interrupts and acknowledge the transfer is completed: Set the csudma.csudma_src_i_sts[done] bit.
5. Wait for PCAP done: Poll while the csu.pcap_status[pcap_wr_idle] bit is cleared.

Wait for the PL Done Status

Wait for the PL done status before doing anything else. This indicates the bitstream is programmed properly, as described in the following steps.

1. Wait for the PL done status: Poll while the csu.pcap_status[pl_done] bit is clear.
2. Once it is done, reset the PCAP interface: Set the csu.pcap_reset[reset] bit.

Programming the CSU DMA

During execution of the CSU ROM code, the CSU uses the CSU DMA for boot-image transfer. The FSBL also uses the CSU DMA for PL programming (through PCAP) and also for image transfers. The CSU DMA can be used after bringing it out of reset, followed by programing the appropriate transfer channel. To bring the CSU DMA out of reset, clear the csu.csu_dma_reset[reset] bit, as described in the following steps.

Trigger a CSU DMA Transfer

A CSU DMA transfer is triggered after writing the size value for a DMA source channel. In the case of PL programming, there is only source channel. In the case of loopback, a DMA destination channel is configured first and then the source channel is configured.

The following steps are used to initiate a CSU DMA transfer.

1. Decide the channel type to be configured and set the address appropriately.
   a. To configure the source channel, set the source address:
      - Set the csudma.csudma_src_addr[addr] = <LSB 30-bit source address (ignore the last 2 bits)>.
      - Set the csudma.csudma_src_addr_msb[addr_msb] = <MSB 16-bit source address>.

   b. Else, set the destination address.
      - Set the csudma.csudma_dst_addr[addr] = <LSB 30-bit destination address (ignore last 2 bits)>.
      - Set the csudma.csudma_src_addr_msb[addr_msb] = <MSB 16-bit destination address>
2. Configure the source/destination size.
   a. To configure the source channel:

      Set the csudma.csudma_src_size[size] = <size of source buffer>.

   b. Else,

      Set csudma.csudma_dst_size[size] = <size of destination buffer>

Wait for CSU DMA Done

1. CSU DMA done can be verified by polling the done bit of the status register.
   a. To poll the source channel:

      Poll while the csudma.csudma_src_i_sts[done] is not set.

   b. Else,

      Poll while the csudma.csudma_dst_i_sts[done] is not set.

DMA done can be acknowledge by clearing the same bit of the status register.

Figure 11-4 describes a CSU DMA transfer.

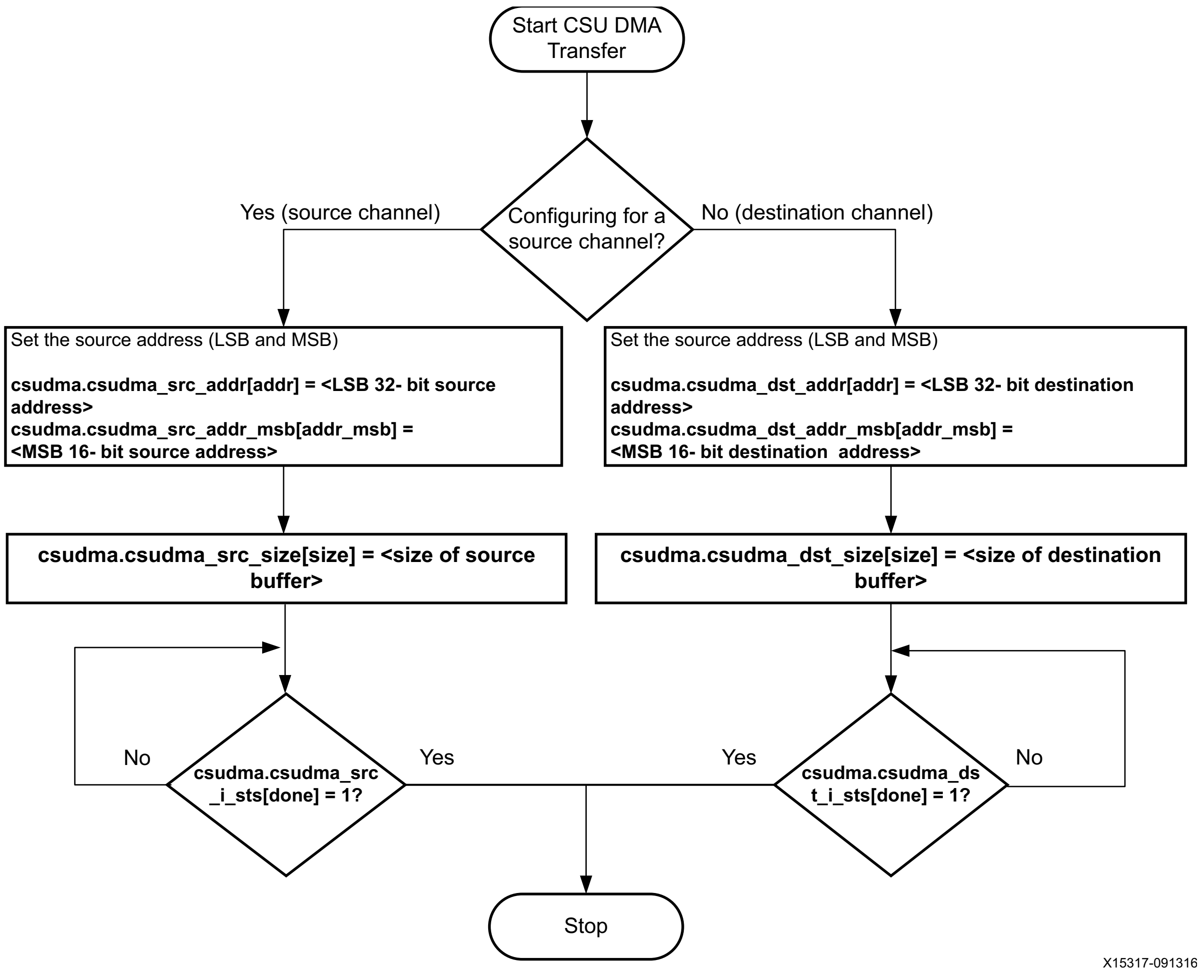

Start CSU DMA Transfer
Configuring for a source channel?
Yes (source channel)
No (destination channel)
Set the source address (LSB and MSB)
csudma.csudma_src_addr[addr] = <LSB 32- bit source address>
csudma.csudma_src_addr_msb[addr_msb] = <MSB 16- bit source address>
Set the source address (LSB and MSB)
csudma.csudma_dst_addr[addr] = <LSB 32- bit destination address>
csudma.csudma_dst_addr_msb[addr_msb] = <MSB 16- bit destination address>
csudma.csudma_src_size[size] = <size of source buffer>
csudma.csudma_dst_size[size] = <size of destination buffer>
No
csudma.csudma_src_i_sts[done] = 1?
Yes
Yes
csudma.csudma_dst_i_sts[done] = 1?
No
Stop
X15317-091316

Figure 11-4: **CSU DMA Transfer**

Security

Introduction

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 Zynq UltraScale+ MPSoC provides features to help secure applications running on the SoC. These features include the following.

- Encryption and authentication of configuration files.
- Hardened crypto-accelerators for use by the user application.
- Secure methods of storing cryptographic keys.
- Methods for detecting and responding to tamper events.

The sections in this chapter describe these features and their use.

The hardware provides many features to detect security intrusions (see *Developing Tamper-Resistant Designs with Zynq UltraScale+ Devices*, XAPP1323 [Ref 32]). This document provides guidance and practical examples to help protect the user IP and sensitive data within a system. This protection (in the form of tamper resistance) needs to be effective before, during, and after the device has been securely booted with a software image or configured with a programmable logic (PL) bitstream. 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 (i.e., initial memory contents and initial state). It also includes external data that is dynamically brought in and out of the device during post-boot normal operation.

Device and Data Security

Configuration Security Unit (CSU) Introduction

At the center of the device security is the configuration security unit (CSU). The CSU is composed of two main blocks as shown in Figure 12-1. On the left is the secure processor block (SPB) that contains a triple redundant processor for controlling boot operation. It also contains an associated ROM, a small private RAM, the physically unclonable function (PUF), and the necessary control/status registers required to support all secure operations. The component on the right is the crypto interface block (CIB) and contains the AES-GCM, DMA, SHA-3, RSA, and PCAP interfaces.

Runtime access to the CSU can be controlled via the Xilinx peripheral protection unit (XPPU). The CSU has a number of responsibilities, including the following.

- Secure boot.
- Tamper monitoring and response.
- Secure key storage and management.
- Cryptographic hardware acceleration.

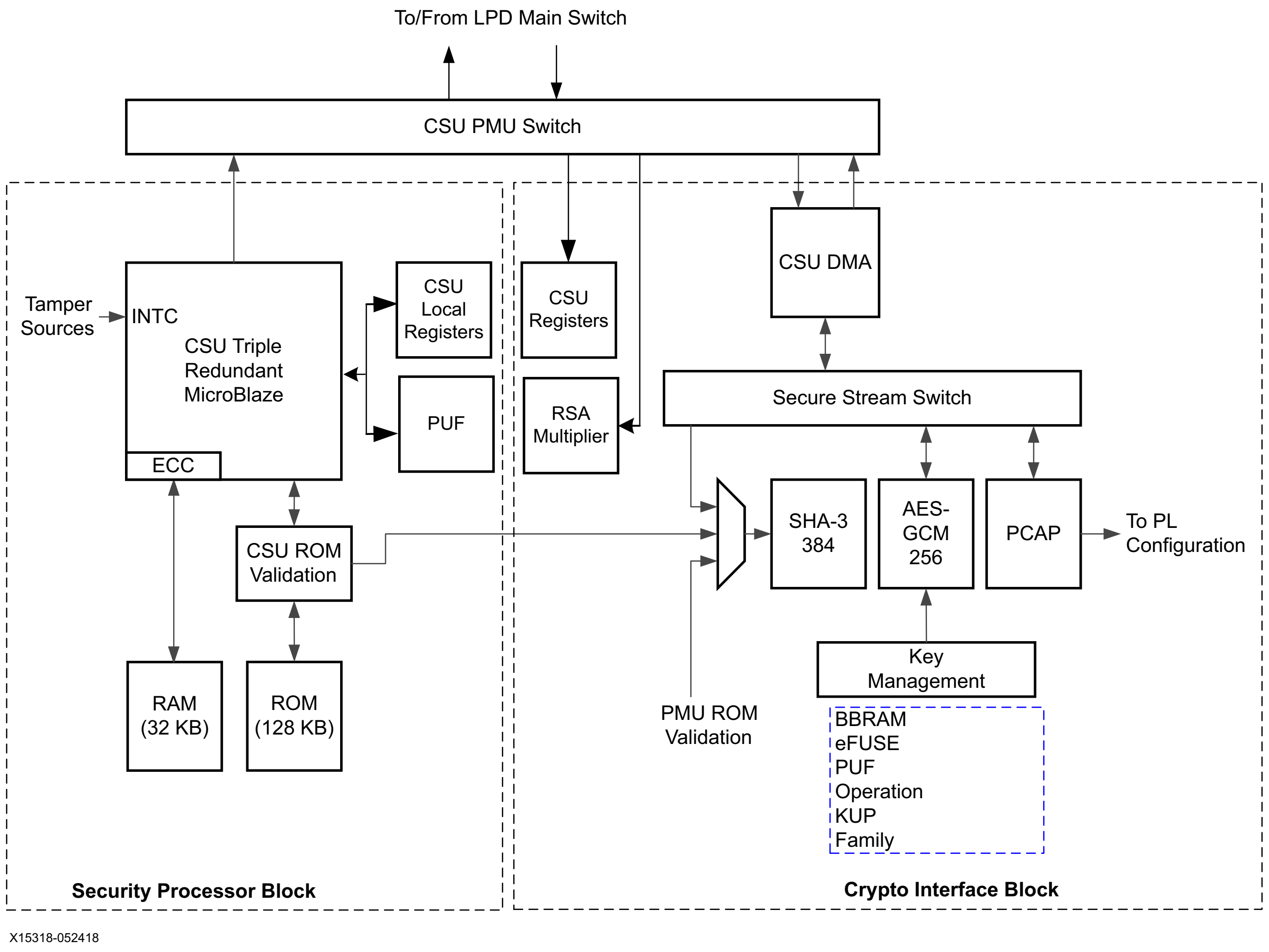


Figure 12-1: **Configuration Security Unit Block Diagram**

Secure Processor Block

The triple-redundant CSU processor provides a highly reliable and robust processing unit for secure boot. The 128 KB CSU ROM is used to store the secure immutable ROM code program. The ROM code passes an integrity check using the SHA-3 prior to being executed. The 32 KB CSU RAM is used as a local secure data storage, and also includes ECC.

The features of the secure processor block are listed here.

- Triple redundant MicroBlaze.
    - Not user accessible.
    - Operates through first error and halts on second error.
- Internal, uninterruptible clock source.
- Dedicated internal RAM protected by ECC.

- Dedicated internal boot ROM protected by SHA-3 integrity check.
- PUF for generation of a device-unique encryption key.

Crypto Interface Block

The features of the CIB include the following.

- Secure stream switch for managing data exchange with cryptographic cores.
- SHA-3/384 hardened core.
- AES-GCM-256 hardened core.
- RSA exponential multiplier accelerator hardened core.
- Secure key management including BBRAM and eFUSE key storage.
- Processor configuration access port (PCAP).

In secure configurations, the RSA and SHA-3/384 are used to authenticate the image and the AES-GCM is used to decrypt the image. During boot, the CIB and SPB run on the internal clock oscillator. After boot, the CIB clock can be sourced from a faster PLL clock to increase the performance of the user-accessible crypto blocks.

Data is moved into and out from the CIB using a direct memory access controller (CSU_DMA) and the secure stream switch (SSS). The Secure Stream Switch in Chapter 11 outlines the options for data movement. See Secure Stream Switch and CSU DMA in Chapter 11, Boot and Configuration for more information on DMA between cryptographic accelerators and memory. The CIB also contains key vaults and key management functionality for keys used during boot, as well as post boot for cryptographic acceleration.

Access to the PL is provided via the PCAP interface.   See PL Configuration in Chapter 11 for more information. Table 11-12 lists CSU registers for performing cryptographic functions, as well as other CSU security critical functionality.

CSU Resets

The different secure blocks of the CSU are reset by writing to the registers in Table 12-1. Write `1` to assert reset, write `0` to deassert reset.

Table 12-1: **CSU Reset Registers**

Component	Reset Register Name
AES-GCM	aes_reset
PCAP	pcap_reset
SHA-3/384	sha_reset

Tamper Monitoring and Response

The primary function of the CSU SPB post-boot is to monitor the system for a tamper event. Table 12-2 lists the twelve different monitoring functions that can be configured.

- The PS system monitor (SYSMON unit) triggering limits for voltage and temperature alarms are user defined and configured.
    - The csu_tamper_4 and csu_tamper_5 registers generate an over and under temperature alarm when the PS SYSMON unit "threshold mode" is set to 1.
- The PL SEU alarm is a runtime health check of the programmable logic.
- Activity on the external PSJTAG interface pins can be detected from within the device and reported on the JTAG toggle detect alarm.
- The CSU can act as a centralized tamper monitor and response hub for a system.
- Single external tamper detect signal through MIO.

The csu_tamper_x registers are write to clear (WTC) so that once a tamper is detected, the tamper alarm can be cleared by writing to the corresponding register.

Table 12-2: **Tamper and Control Registers Channels**

Register	Event Source
csu_tamper_12	PS SYSMON voltage alarm for PS GTR (VTT and VCC are both monitored).
csu_tamper_11	PS SYSMON voltage alarm for PSIO bank 3.
csu_tamper_10	PS SYSMON voltage alarm for PSIO bank 0/1/2 (all three banks).
csu_tamper_9	PS SYSMON voltage alarm for VCC_PSINTFP_DDR.
csu_tamper_8	PS SYSMON voltage alarm for VCC_PSAUX.
csu_tamper_7	PS SYSMON voltage alarm for VCC_PSINTFP.
csu_tamper_6	PS SYSMON voltage alarm for VCC_PSINTLP.
csu_tamper_5	PS SYSMON upper and lower temperature alarms for FPD.
csu_tamper_4	PS SYSMON upper and lower temperature alarms for LPD.
csu_tamper_3	PL single event upset (SEU) error.
csu_tamper_2	JTAG toggle detect.(1)
csu_tamper_1	Input signal via MIO pin.(2)
csu_tamper_0	CSU register.

Notes:

1. The tamper event is caused by toggling the TDI or TMS input signals on the dedicated JTAG pins. The PJTAG interface signals on the MIO are not monitored. The JTAG toggle detect system interrupt is persistent and cannot be cleared until a power-on reset (POR) is done. If this response is chosen, the interrupt must be disabled (masked) after detection to prevent an endless interrupt loop.
2. Assert the MIO tamper input (tamper 1) High until the Tamper Response occurs as configured by the csu_tamper_1 register (Table 12-4). If the system is reset using the PS_RESET_B, then de-assert the MIO tamper signal before releasing PS_RESET_B.

The external tamper detect signals on MIO are listed in Table 12-3.

Table 12-3: **External Tamper Detect Signal on MIO**

CSU Signal	MIO Pins	I/O	Default Input Value to Controller
ext_tamper	18,19,20,21,22,23,24,25,26,31,32,33	I	0

After a tamper event occurs, how the CSU responds is user configurable. Table 12-4 indicates which bit in the tamper response registers to set to obtain a specific tamper response for each tamper event. Multiple tamper response bits can be set for each tamper event. When more than one response bit [3:0] is set, the highest MSB that is set determines the tamper response. If bit [4] and one of the bits [3:0] are set, the BBRAM key is erased and the CSU generates the response associated with the MSB. For example, if bits 1, 2, and 4 are set, the BBRAM key is erased and secure lockdown occurs (no reset).

Table 12-4: **Tamper Monitor and Response Bits**

Bit [4:0]	Response
1 xxxx	Erase the BBRAM key and the response based on the MSB of bits [3:0], if any are set.(1) (3)
x 1xxx	Secure lockdown and 3-states all I/O pins including MIO, PS dedicated, and PL.(2)
x 01xx	Secure lockdown.
x 001x	System reset.
x 0001	System interrupt (GIC IRQ# 117).

Notes:

1. For example, if bit 4, 3, and 2 are all set, the tamper event erases the BBRAM, generates a secure lockdown, and 3-states on all I/Os.
2. The CSU hardware 3-states the PL I/Os and the CSU ROM code writes 1s to the MIO_MST_TRI {0:2} registers.
3. Bit 4 is set for all CSU_TAMPER registers except for the CSU_TAMPER_0 register. For the CSU_TAMPER_0 register, BBRAM is cleared using Bit 5.

The registers are readable but can only be set on write accesses. Specifically, once a specific tamper response is selected for a given tamper event, the bit selecting that response cannot be cleared except by a POR. This prevents incorrect or rogue software from accidentally decreasing the tamper response penalty. Tamper responses can only be added.

Lockdown

Non-Secure Lockdown

Non-secure lockdown is initiated by the CSU ROM when a lockdown event occurs in a non-secure boot mode.

Secure Lockdown

Secure lockdown is a device state that occurs when:

- A tamper event occurs when tamper monitor and response bits 2 or 3 are set for a given tamper event source.
- A failure occurs during secure boot. Very early in the secure boot process all failures will result in secure lockdown. Once image loading has started, failures will only result in secure lockdown if the SEC_LK_eFUSE is programmed (see Figure 12-7).

Secure lockdowns are processed by the CSU ROM. The CSU ROM performs the following steps during a secure lockdown:

1. Tri-state the MIOs.
2. Zeroize the AES keys and reset the AES-GCM core.
3. Reset the APUs.
4. Reset the RPUs.
5. Disable the SRST pin.
6. Enable LPD/FPD isolation.
7. Enable the JTAG security gates (if not already enabled).
8. Toggle PROG_B to PL (this will clear whatever configuration is in the PL).
9. Instruct PMU to perform its lockdown.
10. PMU runs MBIST on the LPD, FPD and PMU.
11. PMU waits for PL housecleaning to complete.
12. PMU puts all blocks in reset.
13. PMU runs SCAN clear on the LPD and FPD if the LPD_SC and FPD_SC eFUSEs are programmed.
14. Secure Lockdown complete is asserted.
15. Optional (disabled by eFUSE)
    a. PMU set bootmode to JTAG.
    b. (optional) PMU triggers internal POR.
    c. (optional) PS reboots, enabling the BSCAN capabilities. See Figure 12-7 for more details.

Emulating a Tamper Event

During system design and test, a tamper event can be emulated to ensure the system is functioning correctly. The csu_tamper_trig register, combined with the csu_tamper_0 register, provides a mechanism for testing tamper responses. An example of emulating a tamper response is as follows.

1. Write to the csu_tamper_trig register.

2. The associated tamper response in csu_tamper_0 is executed.
3. The csu_tamper_trig register self-clears.

Staged Response to a Tamper Event

Systems might require multiple responses to a tamper event. The csu_tamper_trig register, combined with the csu_tamper_0 register, provides a way to have a two-staged response to a tamper event. An example of building a staged response is as follows.

1. Set bit 0 in csu_tamper_6 (i.e., generate an IRQ when VCCINT_LPD is out of range).
2. Set bit 2 in csu_tamper_0 (i.e., enter secure lockdown).
3. Tamper event occurs. VCCINT_LPD goes out of range.
4. The csu_tamper_6 causes an IRQ to be set.
5. User software responds to IRQ and clears the tamper.
6. User software performs some additional action, such as logging or zeroing of configuration or data.
7. User software writes to csu_tamper_trig register.
8. Csu_tamper_0 response is executed. The device goes into secure lockdown.

The tamper events can be securely and permanently logged for later analysis. Logging can be done within the device through a user eFUSE.

Key Management

The AES crypto engine has access to a diverse set of key sources. Non-volatile key sources include eFUSEs, BBRAM, a PUF key encryption key (KEK), and a family key. These keys maintain their values even when the device is powered-down. Volatile key sources include an operational key and a key update register key.

The device key source selection is exclusively done by the CSU ROM based on the authenticated boot image header. A device key can be from any of the following sources (see Figure 12-2).

- BBRAM
- Boot
- eFUSE
- Family
- Operational
- PUF KEK

Table 12-5: **Types of Keys**

Key Name	Description
BBRAM	The BBRAM key is stored in plain text form in a 256-bit RAM array.
Boot	The boot key register holds the decrypted key while the key is in use.
eFUSE	The eFUSE key is stored in eFUSEs. It can be either plain text, obfuscated (i.e., encrypted with the family key), or encrypted with the PUF KEK.
Family	The family key is a constant AES key value hard-coded into the devices. The same key is used across all devices in the Zynq UltraScale+ MPSoC family. This key is only used by the CSU ROM to decrypt an obfuscated key. The decrypted obfuscated key is used to decrypt the boot images. The obfuscated key can be stored in either eFUSE or the authenticated boot header. Because the family key is the same across all devices, the term obfuscated is used rather than encrypted to reflect the relative strength of the security mechanism.
Operational	The operational (OP) key is obtained by decrypting the secure header using a plain text key obtained from the other device key sources. For secure boot, this key is optional. Use of the OP key is specified in the boot header and minimizes the use of the device key, thus limiting its exposure.
PUF KEK	The PUF KEK is a key-encryption key that is generated by the PUF.
Key update register (KUP)	User provided key source. After boot, a user selected key can be used with the hardened AES accelerator.

In addition to the BBRAM and eFUSE key storage locations, the Zynq UltraScale+ MPSoC also allows for the device key to be stored externally in the boot flash. This key can be stored in its obfuscated form (i.e., encrypted with the family key) or in its black form (i.e., encrypted with the PUF KEK).

A device key (a key used to boot the device) is selected by the CSU ROM based on the authenticated boot header or the ENC_ONLY eFUSE setting. To use the device key post boot, the following conditions must be met.

- The device key is available post boot if the initial configuration files are encrypted or if the authentication only option is selected. See Hardware Root of Trust Only Boot (Auth_Only Option) for more information.
- The device key used during boot must be used for all image partitions. The key source used for partition decryption cannot be changed until the next POR. For example, it is not possible to encrypt some partitions with a BBRAM key and others with an eFUSE key. It is also not possible for some partitions to use the operational key and other partitions to not use the operational key.
- The device key can be changed to the PUF KEK if all of the conditions in the section are fulfilled. See Secure Non-Volatile Storage for more information.

Using only the device key post boot is not restricted. A user key can also be loaded into the KUP. The aes_key_src register can be used to select between the device key and the key update key. Figure 12-2 shows the key selection process and the protections in place.

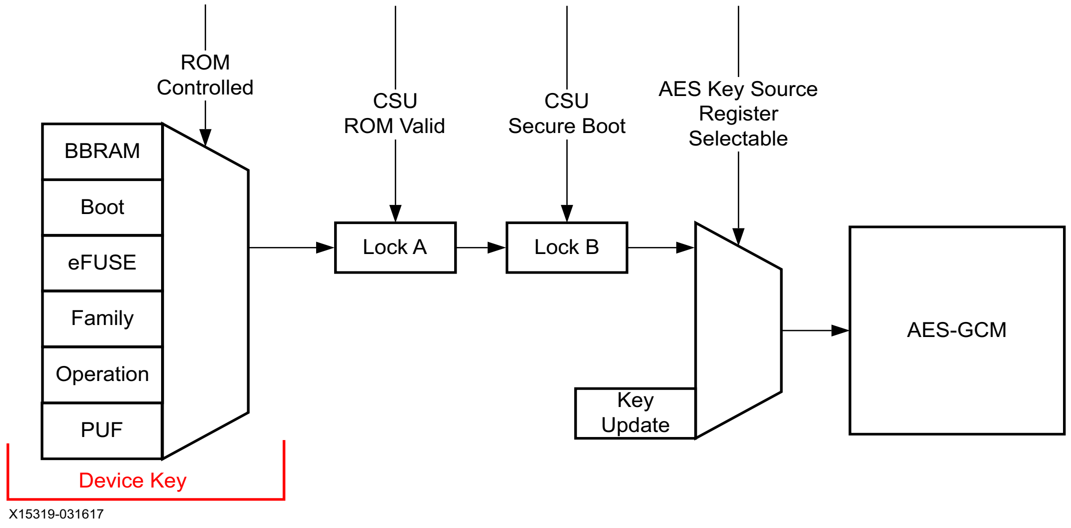


Figure 12-2: **Key Selection**

Battery-Backed RAM

The BBRAM module 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 PS_VCCAUX supply, the BBRAM is powered by the PS_VCCAUX supply. When the PS_VCCAUX supply is switched off, the device automatically switches the BBRAM power over to PS_VCCBATT. The key stored in BBRAM can only be stored in its unencrypted form (i.e., red). It cannot be obfuscated (family) or encrypted (black). The BBRAM can also be cleared, which is valuable as a tamper response.

BBRAM Programming

The BBRAM key memory space is 288-bits. The BBRAM can be programmed by system software running on an RPU or APU processor, or via the PJTAG interface on MIO that connects to the Arm DAP controller and becomes an AXI bus master. The BBRAM block diagram is shown in Figure 12-3. The BBRAM and eFUSE programming details are described in the *Programming BBRAM and eFUSEs Application Note* (XAPP1319) [Ref 20].

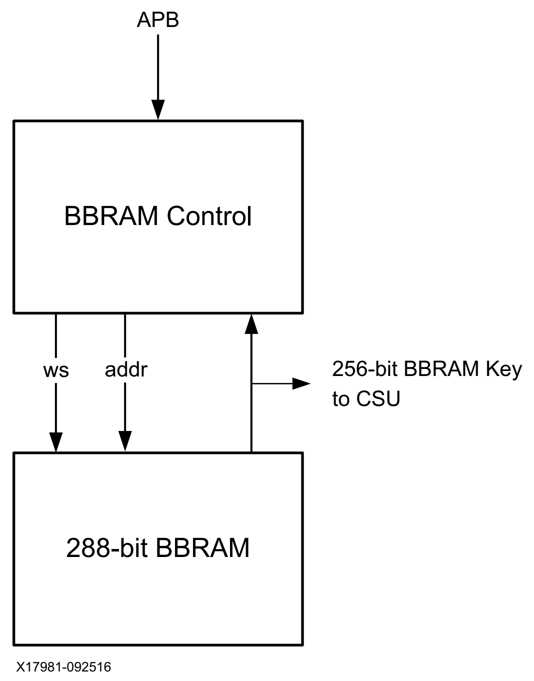


Figure 12-3: **BBRAM Programming Interface**

BBRAM Readback Protections

In previous generations of Xilinx devices, the AES key stored in battery-backed RAM (known as BBR) could be read out for validation. The BBR had a protocol mechanism in which the key was erased prior to being able to program and verify the key. Although these protocol protection mechanisms still exist, the readback path for the key has been removed. The Zynq UltraScale+ MPSoC does not allow read back of the AES key in its BBRAM. 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 value. The device then provides a pass or fail result.

BBRAM Zeroization

The AES key in BBRAM can be erased using an active write to 0's controlled by an internal zeroization signal. A status bit is provided to confirm that the key is all 0's.

BBRAM Key Agility

The AES key in BBRAM can be securely updated from within the device while the device is in operation. Once the key is updated, subsequent boots of the device will use the new key.

eFUSE

The eFUSE array contains a block of 256 eFUSEs that can provide a key to the AES-256 crypto 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), obfuscated form (gray), or encrypted form (black).

eFUSE Programming

The eFUSEs can be programmed by system software running on an RPU or APU processor or via the PJTAG interface (on MIO) that connects to the Arm DAP controller and becomes an AXI bus master. In both cases, the eFUSE programming registers are accessed.

The XilSKey macro library provides a convenient structure to program the eFUSEs. For details on eFUSE programming, see the *Programming BBRAM and eFUSEs Application Note* (XAPP1319) [Ref 20].

eFUSE Readback Protections

In previous generations of Xilinx devices, the key stored in the eFUSEs could be read out for validation. There were options to close the readback path by blowing additional eFUSEs. This readback path has been removed. The Zynq UltraScale+ MPSoC does not allow read back. 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.

eFUSE Zeroization

Although the eFUSE key can by "oneized" by blowing all the eFUSEs from inside the device, it is unclear how much value this provides from a security point of view. Care must be taken to ensure that the key cannot be observed using a simple power analysis (SPA) attack during the blowing of the key bits. It is also possible for an adversary to manipulate external voltages and clocks to compromise successful eFUSE programming.

Key Update Register

The key update register is used during boot to support the key rolling feature, where the different AES key must be loaded multiple times. After boot, any key can be loaded into this register via APB by software running on the PS. A 256-bit KUP key is stored in the eight AES key update registers.

Operational Key

The OP key is a register that holds the key decrypted from the secure header of the boot image. See Minimizing Use of the AES Boot Key (OP Key Option) for more details.

Storing Keys in Obfuscated Form

As shown in Figure 12-4, the user key is encrypted with the family key, which is embedded in the metal layers of the device. This family key is the same for all devices in the Zynq

UltraScale+ MPSoC family. The result is referred to as the *obfuscated key*. The term obfuscated is used instead of encrypted to reflect the relative strength of the security mechanism. The obfuscated key can reside in either the authenticated boot header or eFUSEs. During boot, the CSU ROM takes the obfuscated key, decrypts it with the family key, and then uses the resulting user key to decrypt the boot images.

The Xilinx development tools (bootgen) can be used to create a boot image with the obfuscated key. The family key is not distributed with the Xilinx development tools. To receive the family key, contact secure.solutions@xilinx.com. For more information on generating boot images with the obfuscated key, see "Chapter 8: Security Features" in the *Zynq UltraScale+ MPSoC Software Developer's Guide* (UG1137) [Ref 3].

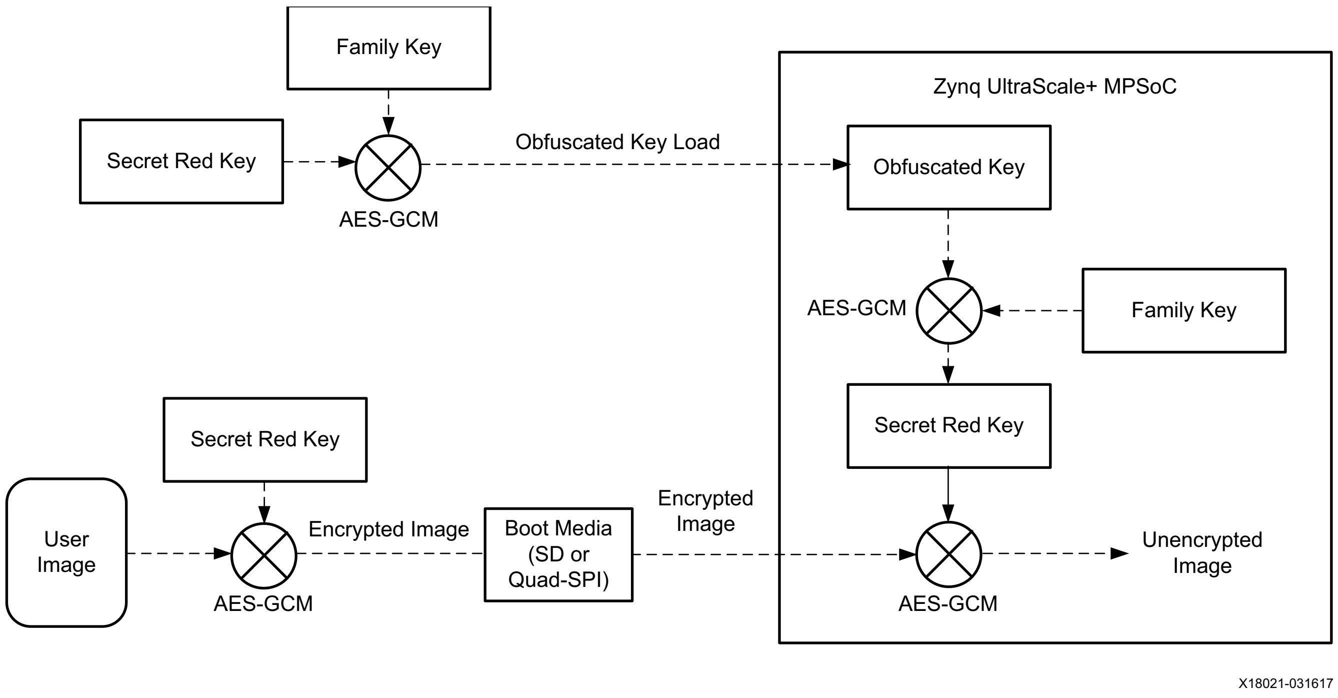


Figure 12-4: **Obfuscated Key**

Storing Keys in Encrypted Form (Black)

The black key storage solution, as shown in Figure 12-5, uses a cryptographically strong KEK generated from a 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 the following 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 (cannot be read by the user).

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. PUFs are tiny circuits that exploit these chip-unique variations to generate unique keys. The type of PUF used to generate the KEK is also an important consideration. The Zynq UltraScale+ MPSoC PUF uses an asymmetric technology (i.e., a ring oscillator based type PUF licensed from Verayo), which 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.

IMPORTANT: *PUF regeneration can only be performed when authentication is enabled. The PUF is disabled in the encrypt-only secure boot mode.*

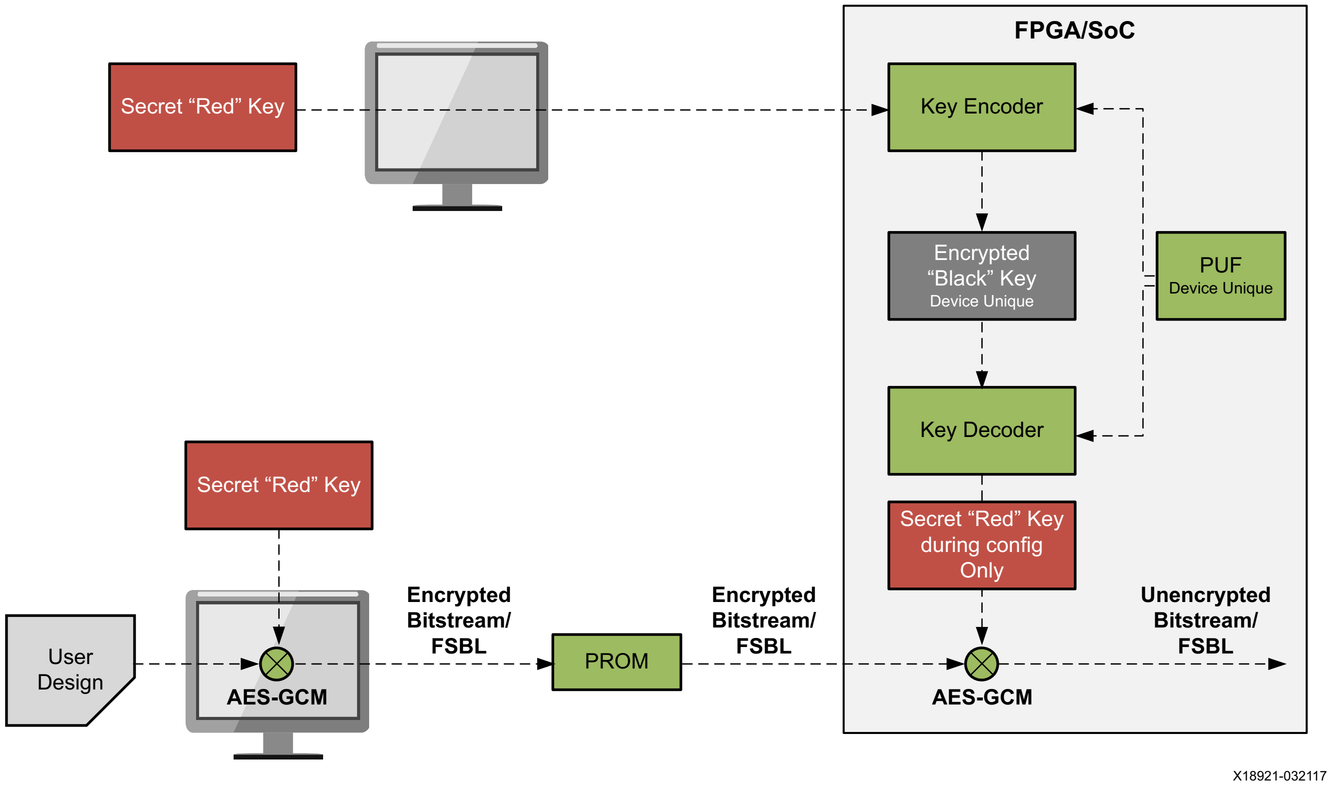

FPGA/SoC
Secret "Red" Key
Key Encoder
Encrypted "Black" Key Device Unique
PUF Device Unique
Key Decoder
Secret "Red" Key
Secret "Red" Key during config Only
User Design
AES-GCM
Encrypted Bitstream/ FSBL
PROM
Encrypted Bitstream/ FSBL
AES-GCM
Unencrypted Bitstream/ FSBL
X18921-032117

Figure 12-5: **Black Key Storage**

PUF Helper Data

The PUF uses approximately 4 Kb of helper data to help the PUF recreate the original KEK value over the complete guaranteed operating temperature and voltage range over the life of the part. The helper data consists of a Syndrome value, an Aux value, and a Chash value (see Table 12-6). The helper data can either be stored in eFUSEs or in the boot image.

Table 12-6: **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. • If the CHASH is not programmed, then BH black key can be used so long as (EITHER bh_auth or rsa_en) is used. • If the CHASH is programmed, then the eFUSE black key can be used so long as (EITHER bh_auth or rsa_en is used) AND the efuse syndrome data has not been invalidated. • If the CHASH is programmed, then the BH black key can be used so long as (EITHER bh_auth or rsa_en) is used AND the efuse syndrome data has been invalidated.

PUF Operations

Access to the PUF is restricted by the CSU. The CSU offers the PUF as a CSU service. The PUF can be accessed through the CSU registers. The CSU supports the user commands listed in Table 12-7.

Table 12-7: **CSU User Commands**

Command	Description
Registration	Create a new KEK and associated helper data (first time).
Re-registration	Create a new KEK and associated new helper data.
Reuse	Encrypt/decrypt with the existing KEK and associated helper data (valid for eFUSE helper data only).

Figure 12-6 shows a block diagram of how the PUF is connected inside the CSU.

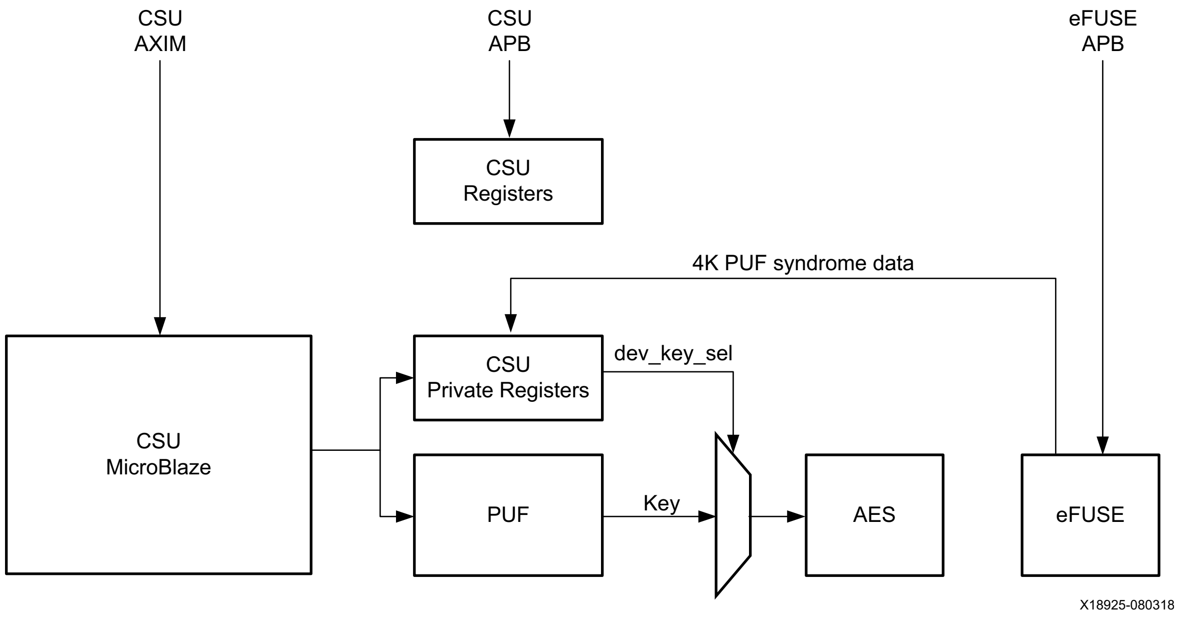


Figure 12-6: **Block Diagram of PUF Connection in CSU**

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. The registration software can then use the KEK to encrypt the user key and program the eFUSEs. Alternatively, the encrypted user key can be output for inclusion into a boot image. The registration software also programs the helper data into the eFUSEs. Alternatively, 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 CSU bootROM examines the authenticated boot image header. 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 CSU then initializes the PUF, loads the helper data, and regenerates the KEK. This process is called regeneration. Once the KEK is regenerated, the CSU bootROM can use it to decrypt the user key, which is then used to decrypt the rest of the boot image.

PUF Control eFUSEs

The eFUSEs listed in Table 12-8 control additional PUF behaviors.

Table 12-8: **PUF Control eFUSEs**

eFUSE Name	Description
REG_DIS	Disables registrations of the PUF.
SYN_INVALID	Invalidates the helper data contained in the eFUSEs.
SYN_LOCK	Prevents modification of the helper data contained in the eFUSEs.

PUF Characterization, Testing, and Ordering

The *Zynq UltraScale+ MPSoC PUF Characterization Report* (RPT236) is a Xilinx proprietary document that covers additional characterization testing performed on the PUF. This characterization covers the PUF's stability (i.e., ability to accurately regenerate the KEK) over the voltage, temperature, and aging. It also covers characterization of the entropy, or security strength, of the KEK. Contact your local Xilinx FAE or sales person for details on how to obtain a copy of the report.

Special ordering codes are required for devices where additional manufacturing tests have been performed on the PUF to help ensure entropy (i.e., key strength).

Key Management Summary

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), obfuscated with the family key, or encrypted with the PUF KEK (black). These options are described in Table 12-9.

Table 12-9: **Boot Image Keys**

Features	BBRAM	eFUSE	Boot Image
Programming method	Internal via software External via JTAG	Internal via software External via JTAG PUF registration software	Bootgen Bootgen + PUF Registration software
Program verification	CRC32 Only	CRC32 Only	N/A
Key state during storage	Red	Red, black, or obfuscated	Black or obfuscated
In-use protections	Temporary storage in registers, not RAM. Transferred in parallel, not serial. Boot: DPA counter measures and zeroization after use.		

Protecting Test Interfaces

JTAG Interface Protections

On power-up, the default boot state is secure, and the JTAG interface only accepts a limited set of commands. These commands are listed here.

- IDCODE
- HIGHZ_I/O (applies only to PS I/O)
- JTAG_STATUS
- PS_ERROR_STATUS

- BYPASS

The device can boot up in secure and non-secure mode. The two secure boot modes are hardware root of trust and encrypt only.

Figure 12-7 shows the JTAG capabilities throughout the secure and non-secure boot process. For non-secure boots, once the boot is complete, either successfully or unsuccessfully, the full suite of JTAG commands are enabled.

For secure boots, if the boot is completed successfully, the authenticated software is capable of enabling the additional JTAG commands. Otherwise, only the IDCODE, HIGHZ_IO, BYPASS, JTAG_STATUS, and PS_ERROR_STATUS commands are available. Since the PS_ERROR_STATUS pin is driven by GPO of the PMU, when the PMU is reset the PS_ERROR_STATUS will be cleared. However, at the end of the secure lockdown, if the option to reboot into JTAG for boundary scan debug is on, then the PMU will get reset and the ERROR_STATUS pin will be deserted. It should be noted that this reset event doesn’t clear the JTAG_ERROR_STATUS register, which can be read via the JTAG_ERROR_STATUS instruction on the JTAG TAP. In the event of a failed secure boot, the JTAG capabilities are dependent on how the device was provisioned.

- Programming the SEC_LK eFUSE forces every failed secure boot to enter secure lockdown.
- In the event that SEC_LK is not programmed:
  - User integration and test is supported via commanding authentication and encryption through the boot header. See Integration and Test Support (BH RSA Option) for more details. In the event of a failed secure boot, JTAG is enabled.
  - For fielded systems, where authentication or encryption is forced upon every boot, the device enables the BSCAN capabilities only to support continuity testing. In this state, internal memory and registers are zeroized, and both the A53s and the R5s are held in reset.

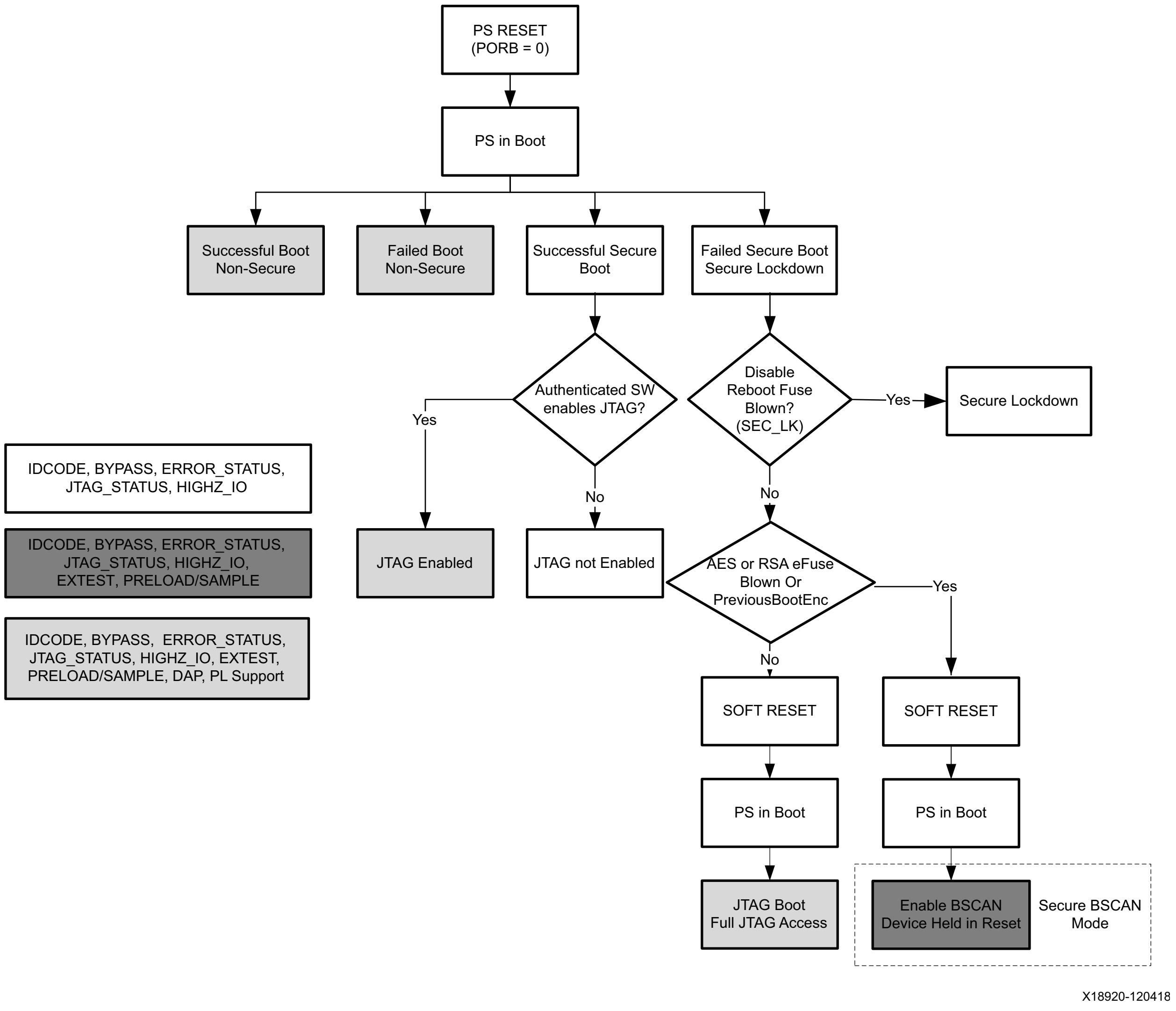

PS RESET
(PORB = 0)
PS in Boot
Successful Boot Non-Secure
Failed Boot Non-Secure
Successful Secure Boot
Failed Secure Boot Secure Lockdown
Authenticated SW enables JTAG?
Yes
No
Disable Reboot Fuse Blown? (SEC_LK)
Yes
No
Secure Lockdown
IDCODE, BYPASS, ERROR_STATUS, JTAG_STATUS, HIGHZ_IO
IDCODE, BYPASS, ERROR_STATUS, JTAG_STATUS, HIGHZ_IO, EXTEST, PRELOAD/SAMPLE
IDCODE, BYPASS, ERROR_STATUS, JTAG_STATUS, HIGHZ_IO, EXTEST, PRELOAD/SAMPLE, DAP, PL Support
JTAG Enabled
JTAG not Enabled
AES or RSA eFuse Blown Or PreviousBootEnc
Yes
No
SOFT RESET
SOFT RESET
PS in Boot
PS in Boot
JTAG Boot Full JTAG Access
Enable BSCAN Device Held in Reset
Secure BSCAN Mode
X18920-120418

Figure 12-7: **JTAG Interface Protections**

In addition to disabling specific JTAG commands, specific JTAG sites are disabled by default on power-on by software-controlled security gates. Triple redundancy is used to maintain the state of these security gates. The location of these gates is shown in Figure 39-1 in Chapter 39, System Test and Debug.

Finally, there is an eFUSE that completely disables the JTAG interface in all situations. Only BYPASS and IDCODE are allowed when the JTAG_DIS eFUSE is programmed.

PL Clearing

The CSU contains the PCAP interface. The PCAP interface can be used to monitor the configuration memory's health in the PL. The PCAP CSU.pcap_prog [pcfg_prog_b] register bit can be used to erase the configuration memory in the PL and the CSU.pcap_status can be used to actively verify the contents have been erased. This provides a means of using PL configuration memory clearing as a tamper response.

A POR or soft reset, by default, clears the PL. There are applications where independence is needed between the PS and PL. To enable these applications, the ability to gate the reset/reprogramming of the PL is added through the PROG_GATE circuit. The PROG_GATE circuit can be controlled by the PMU_GLOBAL. PS_CNTRL.PROG_GATE and PROG_ENABLE bits as listed in Table 12-10

Table 12-10: **PROG_GATE Circuit Control**

Prog_Enable	Prog_Gate	Description
0	0	Previous control maintained (This is the reset/power on state. The PROG_GATE circuit powers on with the PS able to reset the PL).
0	1	pcfg_prog_b is blocked – PS reset does not reset the PL.
1	0	pcfg_prog_b is not blocked – PS reset does reset the PL.
1	1	Invalid condition.

After a successful configuration, SW can write to this register and configure the PROG_GATE circuit so that a soft reset to the PS does not clear the PL.

The behavior can be changed by programming any one of the three PROG_GATE[2:0] eFUSEs. These eFUSEs override the PROG_GATE circuit and force the PL to always be cleared upon a PS reset. The PROG_GATE[2:0] eFUSEs can be observed in the SEC_CTRL register in the eFUSE registers.

Device DNA Identifiers

Each device has a unique 96-bit DNA identifier number to improve security. No two devices have the same DNA. A DNA identifier number exists in the PL and the PS.

Xilinx recommends using the PL-based DNA identifier for secure applications that depend on an unchangeable and unique device identifier. There is a PS-based DNA identifier, but it is possible that one or more of its bits could be changed.

Note: The PL and PS DNA identifiers might not be exactly the same as shipped by Xilinx.

The PL DNA identifier cannot be changed (all PL DNA bits are read-only). Table 12-11 lists the device DNA identifiers.

Table 12-11: **Device DNA Identifiers**

Identifier	Length	Read-only?	Read Access Methods
PL DNA	96 bits	Yes, always.	Method 1: instantiate the PL DNA_PORTE2 primitive Method 2: connect to the JTAG PL TAP controller and use the FUSE_DNA instruction ***Note:*** For more details, see the *UltraScale Architecture Configuration User Guide* (UG570) [Ref 33]. ***Note:*** The Vivado Hardware Manager displays the PL DNA value.
PS DNA	96 bits	No, not all bits.	Access the read-only EFUSE.DNA_x registers at addresses: • DNA_0: 0xFFCC_100C • DNA_1: 0xFFCC_1010 • DNA_2: 0xFFCC_1014 ***Note:*** The SDK API, XilSKey_ZynqMp_EfusePs_ReadDna, returns the PS DNA value.

The Xilinx 2D bar code that is printed on the top of each device also includes the PL DNA identifier.

Error Output Disable

Many secure applications have requirements to disable error notifications to the outside world. These applications want to disable the PS_ERROR_OUT and PS_ERROR_STATUS signals. The registers that control these signals are described in Table 6-16 in Chapter 6, Platform Management Unit. The user software must be loaded and executing to control these registers.

The ERR_DIS eFUSE permanently disables reading of the PS_ERROR_STATUS register from the external JTAG chain.

Cryptographic Acceleration

AES-GCM

The AES-GCM core has a 32-bit word-based data interface with support for a 256-bit key. The AES-GCM mode supports encryption and decryption, multiple key sources, and built-in message integrity check.

Note: The AES engine operates on a 32-bit boundary.

The AES-GCM-256 core allows for the following key sources.

- BBRAM key
- eFUSE device key

- Operation key (OPKEY)
- Key update register (KUP)
- Family key (for obfuscated key storage)
- PUF key-encryption-key (KEK) (for black key storage)

Initialization Vector Register

The four initialization vector (IV) registers combined create a larger 128 bit value. This 128-bit values contains two separate fields. The first field resides in the first three AES IV registers (aes_iv_0, aes_iv_1, and aes_iv_2) and contains the 96-bit AES-GCM initialization vector (IV). The 96-bit AES-GCM IV is specified by the AES-GCM standard and initializes the counts used in this AES mode. The fourth register (aes_iv_3) contains the decrypt length count (DLC). The DLC specifies the data size of the next block. The DLC is used when the key rolling feature is enabled in the boot image. Table 12-12 shows the IV vector format.

Table 12-12: **Initialization Vector Format**

127 32 0

GCM IV	DLC (Decrypt length count)
96-bit random value	Next block data size (key rolling)

Programming AES-GCM Engine

The XilSecure library provides APIs to access the AES-GCM core. For more information, see the AES-GCM chapter in the *Xilinx Standalone Library Documentation: OS and Libraries Document Collection* (UG643) [Ref 16].

SHA-3/384

The SHA hardware accelerator included in the Zynq UltraScale+ MPSoC implements the SHA-3 algorithm and produces a 384-bit digest. It is used together with the RSA accelerator to provide image authentication. It is also used to perform an integrity check of the CSU and PMU ROMs prior to execution. The SHA-3 block generates a 384-bit digest value. If a design requires a 256-bit digest, use the least significant 256 bits of the digest (see *Recommendation for Applications Using Approved Hash Algorithms NIST Special Publication 800-107* [Ref 56]).

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* [Ref 57] for padding requirements. SHA3-384 padding should be M || 01 || 10 * 1.

Programming SHA-3 Engine

The XilSecure library provides APIs to access the SHA core. See the SHA-3 chapter in the *Xilinx Standalone Library Documentation: OS and Libraries Document Collection* (UG643) [Ref 23].

RSA Accelerator

The Zynq UltraScale+ MPSoC includes an RSA accelerator for public and private key operations. The RSA accelerator supports the following features.

- Implements a modular exponentiation engine.
- Support for R*R mod M pre-calculation.
- Support for multiple RSA key sizes including 2048, 3072, and 4096. Only the key size of 4096 is supported during boot. For all key sizes supported, see the *Xilinx Standalone Library Documentation: OS and Libraries Document Collection* (UG643) [Ref 23].
- Implements efficient processing of a short public exponent.

Programming the RSA Engine

The XilSecure library provides APIs to access the RSA accelerator. See the RSA chapter in the *Xilinx Standalone Library Documentation: OS and Libraries Document Collection* (UG643) [Ref 23].

Information regarding the performance of cryptographic acceleration, within two different software architectures, is provided in *Accelerating Cryptographic Performance on the Zynq UltraScale+ MPSoC* (WP512) [Ref 37].

Secure Non-Volatile Storage

In addition to storing the user key in encrypted form, the PUF can also be used to encrypt/decrypt data to store in external memory. This use case provides a secure non-volatile solution. In cases where the PUF helper data is stored in eFUSEs and RSA authentication is enabled, the regeneration process can be used by the user's application software to regenerate the KEK. This KEK can then be used to encrypt data, such as additional user keys, using the device unique KEK. This encrypted user data can then be stored off-chip or in the user eFUSEs and decrypted using the same process at a later time. See the *External Secure Storage Using the PUF Application Note* (XAPP1333) [Ref 34].

Note: When the PUF is used in this manner, it becomes the device key and the device key selection cannot be changed back to the BBRAM or eFUSE key without a power on reset. The user can still choose between the Key Update Register and the PUF (see Figure 12-2.)

Security Related eFUSEs

PS eFUSEs

An eFUSE is a small, one-time programmable, non-volatile memory element. The eFUSE arrays store various types of important information. The definition of each bit of an eFUSE is represented in the eFUSE map shown in Table 12-13. The device caches the eFUSE values into registers so that reading the eFUSE value means reading the eFUSE *cache* and not the physical device eFUSEs. Loading the eFUSE cache occurs during the pre-boot phase, via a register command (EFUSE.EFUSE_CACHE_LOAD) or automatically when the XilSKey library is used. Reading is done from the eFUSE registers at 0xFFCC0000 (see the *Zynq UltraScale+ MPSoC Register Reference* (UG1087) [Ref 4]).

Because readback is not available on the AES key, a CRC check has been built in to validate that the AES key eFUSE has been programmed correctly. Before the CRC check can be performed on a newly programmed eFUSE, the eFUSE cache must be reloaded.

eFUSEs can be programmed using the Xilinx XilSKey library. Inputs are provided in the application header file `xilskey_efuseps_zynqmp_input.h`. The corresponding macro names are listed in Table 12-13. For PUF usage, input is provided via the `xilskey_puf_registration.h` file. For more information on XilSKey library usage, see the Xilinx library documentation.

For details on how to program eFUSEs, see *Programming BBRAM and eFUSEs Application Note* (XAPP1319) [Ref 20].

Table 12-13: **Zynq UltraScale+ MPSoC Security eFUSEs**

Size	Name	Description	XilSKey Name: XSK_EFUSEPS_
32	USER_{0:7}	256 user defined eFUSEs: ***Note:*** In the `input.h` file (see text), write data in the XSK_EFUSEPS_USER{0:7}_FUSES macro and execute the write by setting the XSK_EFUSEPS_USER{0:7}_FUSE macro = True.	USER{0:7}_FUSE
1	USER_WRLK	8 user-defined eFUSE locks. USER_WRLK columns: 0: Locks USER_0, 1: Locks USER_1, ... 7: Locks USER_7, ***Note:*** Each eFUSE permanently locks the entire corresponding user-defined USER_{0:7} eFUSE row so it cannot be changed.	USER_WRLK_{0:7}
1	LBIST_EN	Enables logic BIST to run during boot.	LBIST_EN
3	LPD_SC	Enables zeroization of registers in low power domain (LBD) during boot. ***Note:*** Any of the eFUSE programmed will perform zeroization. Xilinx recommends programming all of them.	LPD_SC_EN
3	FPD_SC	Enables zeroization of registers in full power domain (FBD) during boot. ***Note:*** MGTs must be powered to perform zeroization of the FPD. ***Note:*** Any of the eFUSE programmed will perform zeroization. Xilinx recommends programming all of them.	FPD_SC_EN

Table 12-13: **Zynq UltraScale+ MPSoC Security eFUSEs *(Cont'd)***

Size	Name	Description	XilSKey Name: XSK_EFUSEPS_
3	PBR_BOOT_ ERROR	When programmed, boot is halted on any PMU error.	PBR_BOOT_ERR
32	CHASH	PUF helper data	N/A - handled by PUF registration software directly.
24	AUX	PUF helper data: ECC vector	N/A - handled by PUF registration software directly.
1	SYN_INVLD	Invalidates PUF helper data stored in eFUSEs.	XSK_PUF_SYN_INVALID
1	SYN_LOCK	Locks PUF helper data from future programming.	XSK_PUF_SYN_WRLK
1	REG_DIS	Disables PUF registration.	XSK_PUF_REGISTER_DISABLE
1	AES_RD	Disables the AES key CRC integrity check for eFUSE key storage.	AES_RD_LOCK
1	AES_WR	Locks AES key from future programming.	AES_WR_LOCK
1	ENC_ONLY(1)(2)	When programmed, all partitions are required to be encrypted. Xilinx recommends using this only if security is required and the hardware root of trust (RSA_EN) is not used.	ENC_ONLY
1	BBRAM_DIS	Disables the use of the AES key stored in BBRAM.	BBRAM_DISABLE
1	ERR_DIS	Prohibits error messages from being read via JTAG (ERROR_STATUS register). ***Note:*** The error is still readable from inside the device.	ERR_DISABLE
1	JTAG_DIS(1)	Disables JTAG. IDCODE and BYPASS are the only allowed commands.	JTAG_DISABLE
1	DFT_DIS(1)	Disables design for test (DFT) boot mode.	DFT_DISABLE
3	PROG_GATE	When programmed, these fuses prohibit the PROG_GATE feature from being engaged. If any of these are programmed, the PL is always reset when the PS is reset. ***Note:*** Only one eFUSE needs to be programed to prohibit the PROG_GATE feature from being engaged. Xilinx recommends programming all three.	PROG_GATE_DISABLE
1	SEC_LK	When programmed, the device does not enable BSCAN capability while in secure lockdown.	SECURE_LOCK
15	RSA_EN(1)(2)	When any one of the eFUSEs is programmed, every boot must be authenticated using RSA. Xilinx recommends programming all 15 eFUSEs.	RSA_ENABLE
1	PPK0_WR	Primary public key write lock. When programmed, this prohibits future programming of PPK0.	PPK0_WR_LOCK
2	PPK0_INVLD	When either of the eFUSEs are programmed, PPK0 is revocated. Xilinx recommends programming both eFUSEs when revocating PPK0.	PPK0_INVLD
1	PPK1 WR	Primary public key write lock. When programmed this prohibits future programming of PPK1.	PPK1_WR_LOCK
2	PPK1_INVLD	When either of the eFUSEs are programmed, PPK1 is revocated. Xilinx recommends programming both eFUSEs when revocating PPK1.	PPK1_INVLD
32	SPK_ID	Secondary public key ID. ***Note:*** Write the SPK ID bits into the XSK_EFUSEPS_SPK_ID eFUSE array and set XSK_EFUSEPS_SPKID = True.	SPK_ID

Table 12-13: **Zynq UltraScale+ MPSoC Security eFUSEs *(Cont'd)***

Size	Name	Description	XilSKey Name: XSK_EFUSEPS_
256	AES	User AES key ***Note:*** Write data in the XSK_EFUSEPS_AES_KEY macro and execute the write by setting the XSK_EFUSEPS_WRITE_AES_KEY macro = True.	AES_KEY
384	PPK0	User primary public key0 HASH ***Note:*** Write data in the XSK_EFUSEPS_PPK0_HASH macro. To program 256 bits, use the LSBs and set XSK_EFUSEPS_PPK0_IS_SHA3 = False. To program 384 bits, set XSK_EFUSEPS_PPK0_IS_SHA3 = True. Execute the write by setting the XSK_EFUSEPS_WRITE_PPK0_HASH macro = True.	PPK0_HASH
384	PPK1	User primary public key1 HASH ***Note:*** Write data in the XSK_EFUSEPS_PPK1_HASH macro. To program 256 bits, use the LSBs and set XSK_EFUSEPS_PPK1_IS_SHA3 = False. To program 384 bits, set XSK_EFUSEPS_PPK1_IS_SHA3 = True. Execute the write by setting the XSK_EFUSEPS_WRITE_PPK1_HASH macro = True.	PPK1_HASH
N/A	PUF_HD	Syndrome of PUF HD. These eFUSEs are programmed using Xilinx provided software, Xilskey	N/A - handled by PUF registration software directly.

Note:

1. **IMPORTANT.** Programming any of the noted eFUSE settings preclude Xilinx test access. Consequently, Xilinx does not accept return material authorization (RMA) requests.
2. When the ENC_ONLY or RSA_EN eFUSE is blown, the JTAG boot mode is no longer available. If this was the only mechanism used to program the boot flash, a secondary means should be employed. Xilinx recommends some other form of in-system flash programming and not relying on booting the device successfully to update the flash contents.

PL eFUSEs

In addition to the eFUSEs available in the PS, there are 128 eFUSEs available within the PL. Any value can be programmed into these eFUSEs, which are available via the FUSE_USER_128 register. A description of the eFUSEs is provided in the *UltraScale Architecture Configuration User Guide* (UG570) [Ref 33]. The programming and reading of the eFUSEs is described in the *Internal Programming of BBRAM and eFUSEs Application Note* (XAPP1283) [Ref 35].

Secure Boot

Secure Boot Introduction

The Zynq UltraScale+ device supports two secure boot modes: hardware root of trust or encrypt only. The hardware root of trust uses asymmetric authentication with optional encryption to provide confidentiality, integrity, and authentication of the boot and configuration files. An alternative to the hardware root of trust is the encrypt only secure boot, which is a boot mechanism that does not utilize asymmetric authentication but requires that all configuration loaded must be encrypted and authenticated using AES-GCM.

Secure Boot Summary

There are a number of functional blocks involved in the secure boot process, including the following.

- Dedicated hardware state machines
- Platform management unit (PMU)
- Configuration and security unit (CSU)

The high level boot flow summary is shown in Figure 12-8.

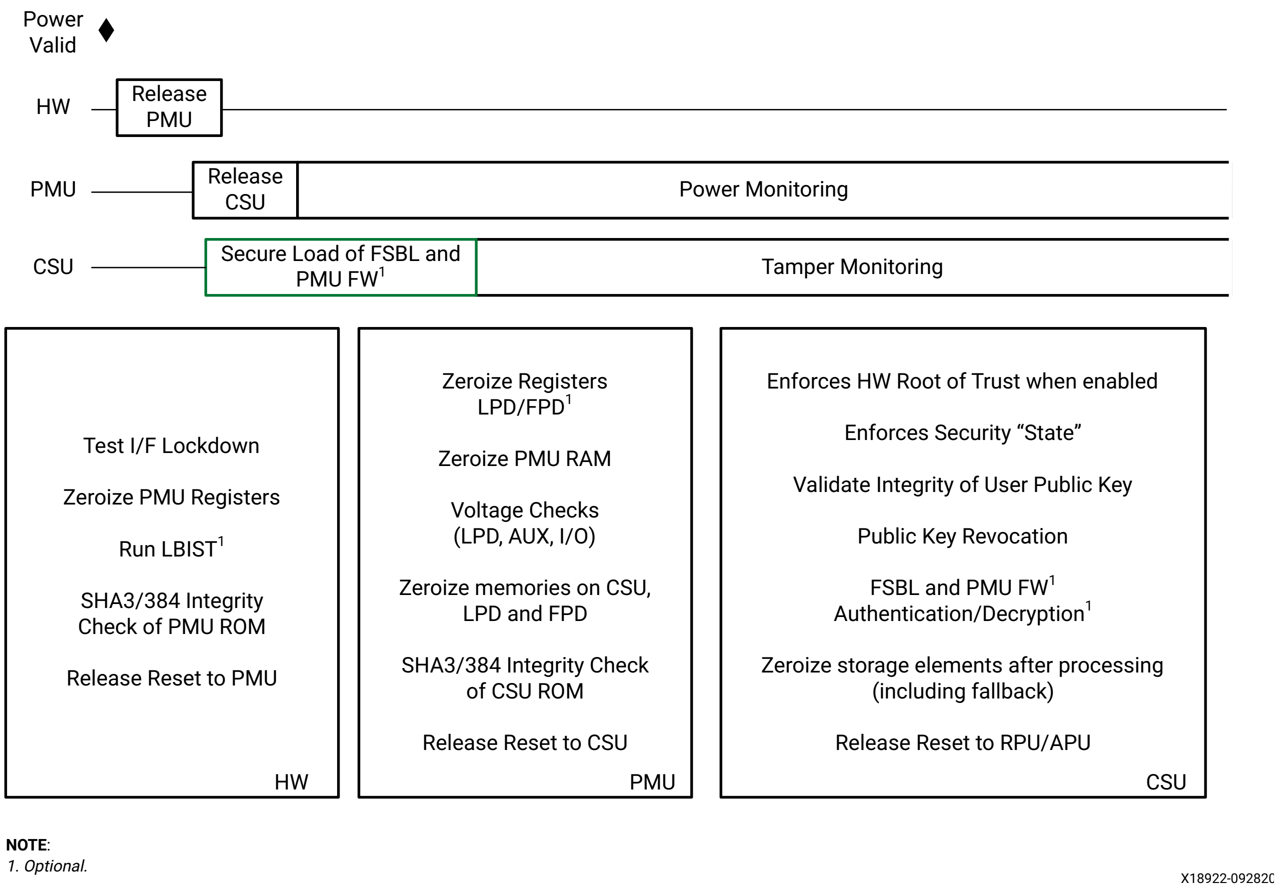

Power Valid
HW
Release PMU
PMU
Release CSU
Power Monitoring
CSU
Secure Load of FSBL and PMU FW[1]
Tamper Monitoring
Test I/F Lockdown
Zeroize PMU Registers
Run LBIST[1]
SHA3/384 Integrity Check of PMU ROM
Release Reset to PMU
HW
Zeroize Registers LPD/FPD[1]
Zeroize PMU RAM
Voltage Checks (LPD, AUX, I/O)
Zeroize memories on CSU, LPD and FPD
SHA3/384 Integrity Check of CSU ROM
Release Reset to CSU
PMU
Enforces HW Root of Trust when enabled
Enforces Security "State"
Validate Integrity of User Public Key
Public Key Revocation
FSBL and PMU FW[1] Authentication/Decryption[1]
Zeroize storage elements after processing (including fallback)
Release Reset to RPU/APU
CSU
NOTE:
1. Optional.
X18922-092820

Figure 12-8: **High-Level Boot Flow**

Once power is valid to the device, the dedicated hardware state machines perform a series of mandatory and optional tasks. The device includes test logic used by the developer for device verification and test. The test interfaces power up in a known secure state. The registers in the PMU are zeroized, which means zeros are written to them, and the zeros are readback to confirm they were written correctly. Optionally, a logic built in self test (LBIST) can be performed during boot. This option is enabled by programming the LBIST_EN eFUSE. LBIST is commonly used in functional safety applications, see Chapter 8, Functional Safety for more details on what circuits of the device are covered via LBIST.

Note: Extra boot time is required when running LBIST.

Finally, the dedicated hardware sends the PMU immutable ROM code through the SHA-3/384 engine and compares the calculated cryptographic checksum to the golden copy stored in the device. If the cryptographic checksums match, the integrity of the PMU ROM is validated and the reset to the PMU is released. If any of these tasks fail, an error flag is set in the JTAG error status register (readable through JTAG). To prevent the error message from being readable through the JTAG error status register, the ERR_DIS eFUSE can be programmed.

The PMU performs a number of mandatory and optional security operations as listed in Table 12-14. See Chapter 6, Platform Management Unit for more information.

Table 12-14: **PMU Security Operations**

Security Operation	Description	Optional?
Zeroize low power domain (LPD) registers	When the LPD_SC eFUSE is programmed, the PMU zeroizes all registers in the LPD.	Yes
Zeroize full power domain (FPD) registers	When the FPD_SC eFUSE is programmed, the PMU zeroizes all registers in the FPD. ***Note:*** The MGTs must be powered during full-power domain zeroization.	Yes
Zeroize PMU RAM	The PMU RAM has zeros written to it, and read back to confirm the write is successful.	No
Voltage checks	The PMU checks the supply voltage of the LPD, AUX, and dedicated I/O to confirm that the voltages are within specifications.	No
Zeroize memories	The PMU zeroizes memories located in the CSU, LPD, and FPD.	No

Once these security operations are complete, the PMU sends the CSU immutable ROM code through the SHA-3/384 engine and compares the calculated cryptographic checksum to the golden copy stored in the device. If the cryptographic checksums match, the integrity of the CSU ROM is validated and the reset to the CSU is released. If any of these tasks fail, an error flag is set in the JTAG error status register (readable through JTAG). The error message can be prevented from being read through the JTAG error status register by programming the ERR_DIS eFUSE. In the event of a PMU error, the default operation of the device is to continue the boot process and release the reset to the CSU. Once the design comes online, it can read the status of all the error messages from inside the device and determine whether or not to continue to operate. To make the device automatically go into lockdown when an error occurs during the boot process, the PBR_BOOT_ERROR eFUSE can be programmed.

The CSU is the center of the secure boot process. It enforces the hardware root of trust or encrypt only secure boot steps when they are enabled. The CSU also maintains the security state of the device by prohibiting the transition from a secure state to an unsecure state, or from an unsecure state to a secure state without a full POR. Once the FSBL (and, if applicable, the PMUFW) has been loaded securely, the CSU zeroizes the storage elements of the cryptographic engines and releases the reset to the specified processing unit (APU or RPU).

Hardware Root Of Trust Secure Boot Details

The Zynq UltraScale+ MPSoC hardware root of trust is based on the RSA-4096 asymmetric authentication algorithm in conjunction with SHA-3/384. There are two key pairs used in the Zynq UltraScale+ MPSoC, and consequently two public key types: the primary public key (PPK) and the secondary public key (SPK). Table 12-15 lists the characteristics of each public key type.

Table 12-15: **Public Keys**

Public Key	Number	Location	Revocation	Notes
Primary (PPK)	2	External memory and hash in eFUSEs.	Can be revoked.	*Only* used to authenticate SPK and authentication header.
Secondary (SPK)	Up to 256	Boot image.	Can be revoked.	Signed by PPK. Used to authenticate everything else.

There are two PPKs; the full public key is stored in external memory (e.g., flash) and a SHA-3/384 hash of the public key is stored in eFUSEs on the device. The CSU, during the boot process, validates the integrity of the public key stored in external memory using the hash stored in eFUSEs. The PPKs can be revoked. The main purpose of the PPK is to authenticate the SPK.

There are 32 SPKs available for the bootloader (FSBL) and up to 256 SPKs available for all other partitions depending on which SPK revocation method is used (standard or enhanced). The SPK is delivered via the authenticated boot image, and is consequently protected against modification. The SPKs can also be revoked and are used to authenticate everything else.

There are a number of considerations when utilizing the hardware root of trust capabilities. These are discussed in detail in Device Provisioning, Boot Operation, and Key Revocation.

Device Provisioning

Before the device can boot with the root of trust, a minimum amount of user information must be programmed or provisioned into the device. At a minimum, the hardware root of trust must be enabled and a hash of the user public key must be programmed into the device. Figure 12-9 shows the critical eFUSEs that must be programmed.

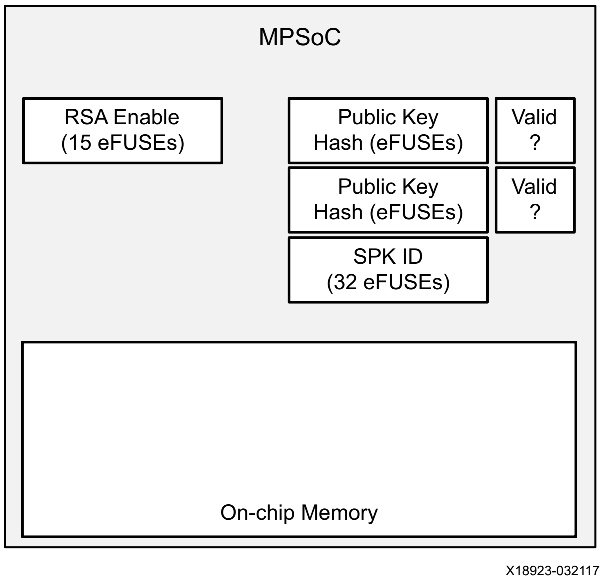


Figure 12-9: **Device Provisioning**

The generation of the primary and secondary key pairs is a user decision. Utilizing Xilinx tools, a hash of the each of the PPKs is obtained and programmed into the eFUSE locations on the device. If desired, the secondary public key identification (ID) can be programmed to a non-zero value.

IMPORTANT: *The Zynq Ultrascale+ MPSoC supports two PPKs. Both PPK hash values shall be programmed before fielding a system.*

Finally, the hardware root of trust must be enabled by programming the fifteen (15) RSA enable eFUSEs. While programming, any one of the fifteen forces every boot to be authenticated. It is recommended that all 15 are programmed. The enable eFUSEs are implemented redundantly as a countermeasure against advanced physical modification attacks such as those using a focused ion beam (FIB).

Boot Operation

Figure 12-10 shows the top-level hardware root of trust boot flow used to authenticate, and optionally decrypt, the FSBL.

Note: PMUFW is optional.

The process starts by determining which PPK to use and then validating the PPK integrity. Since the public key is stored in the boot image in external memory, it be must assumed that an adversary could tamper with it. Consequently, the CSU reads the public key from external memory, calculates its cryptographic checksum using the SHA-3/384 engine, and then compares it to the value stored in eFUSEs. If they match, the integrity of the public key has been validated and the boot can continue.

The secondary public key, and its associated ID, are then read, stored in on-chip memory (OCM), and authenticated using the PPK. Once the SPK and SPK ID have been authenticated, the CSU checks the ID that was bound to the SPK in the boot image to the ID that is stored in eFUSEs. If the IDs match, the SPK is valid and the boot can continue.

The SPK is then used to verify the authenticity of the entirety of the boot image. The CSU authenticates the FSBL, and optionally the PMUFW, while in internal memory. If encrypted, the CSU also performs the decryption.

Note: Encrypting the configuration files is optional.

IMPORTANT: *The CSU processes the FSBL and PMUFW as two separation partitions. Consequently, if the FSBL and PMUFW are encrypted, the AES key and IV are reused, which is a violation of the standard. If the FSBL and PMUFW must both be encrypted, the PMUFW must be loaded by the FSBL, and not the CSU.*

At this stage, control is handed over to the user and the user is responsible for maintaining the chain of trust. The remaining secure boot process is configurable by the user. An example of a hardware root of trust secure boot process is shown in Figure 12-10.

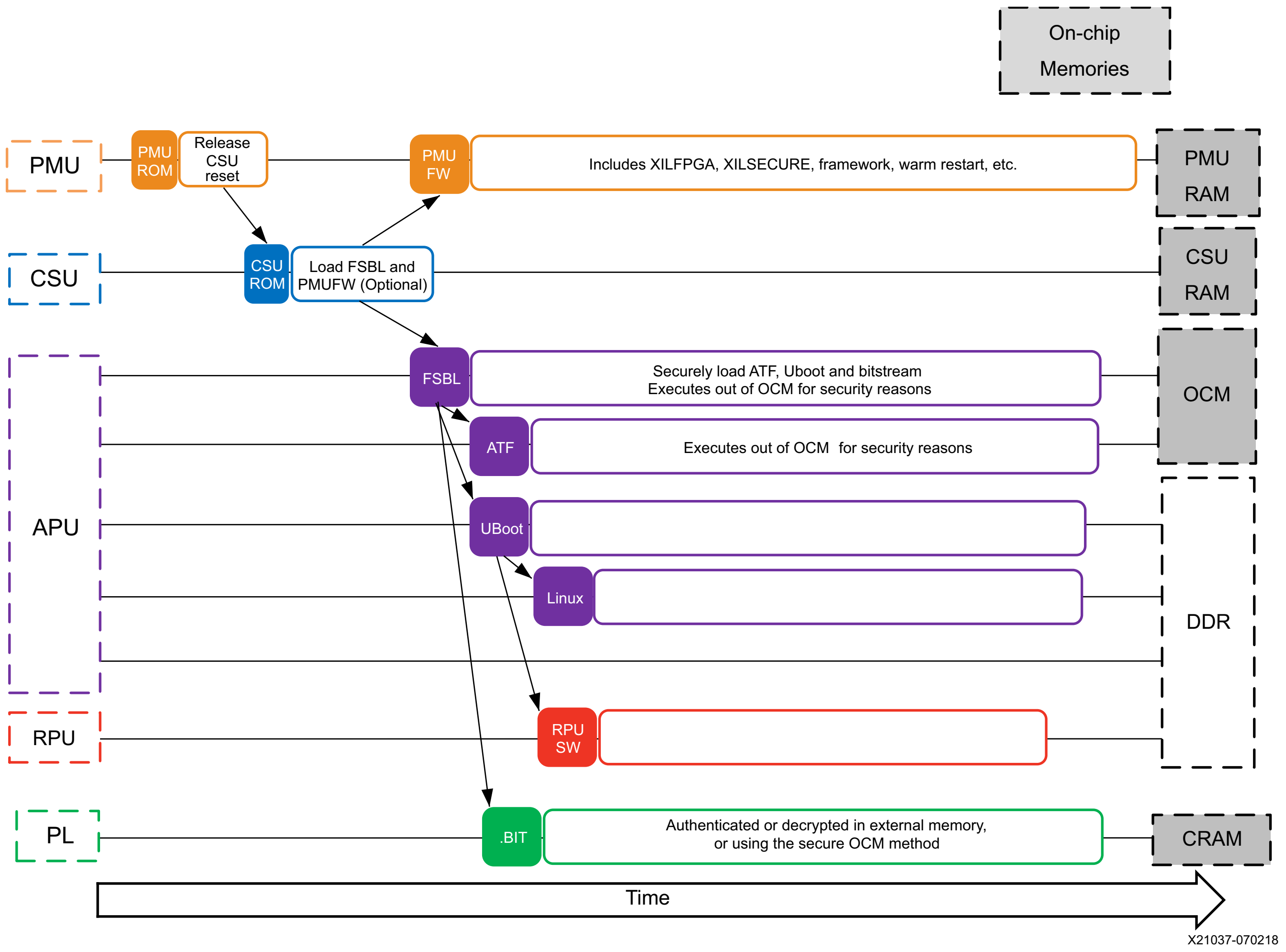

On-chip Memories
PMU
PMU ROM
Release CSU reset
PMU FW
Includes XILFPGA, XILSECURE, framework, warm restart, etc.
PMU RAM
CSU
CSU ROM
Load FSBL and PMUFW (Optional)
CSU RAM
APU
FSBL
Securely load ATF, Uboot and bitstream
Executes out of OCM for security reasons
OCM
ATF
Executes out of OCM for security reasons
UBoot
Linux
DDR
RPU
RPU SW
PL
.BIT
Authenticated or decrypted in external memory, or using the secure OCM method
CRAM
Time
X21037-070218

Figure 12-10: **Hardware Root of Trust Secure Boot Example**

In this example, the FSBL is responsible for securely loading Arm trusted firmware (ATF), U-Boot, and the PL bitstream, all of which can be considered individual partitions and authenticated or encrypted separately. The FSBL executing at EL3 is responsible for all of the security checks (i.e., PPK integrity check and PPK and SPK revocation checks), as well as the actual authentication or decryption of the partitions. The hardware accelerators are used by the FSBL to authenticate or decrypt each partition.

ATF is loaded into OCM and authenticated or decrypted. U-Boot is authenticated or decrypted in external memory because it is too large for internal memory. Bitstreams are always first loaded from Flash into DDR, for systems that have DDR, regardless of security settings. In DDR-less systems, the bitstream remains in the Flash for the authentication, decryption and loading steps. If the bitstream is authenticated, then the bitstream is first validated using the RSA algorithm. If authentication passes, it is then sent from its source location (DDR or Flash) to the PCAP or to the AES decryptor and then onto PCAP. As this causes the bitstream to be read twice, once for authentication and a second time for decryption/loading, additional steps using internal OCM memory as a temporary buffer are taken to ensure the bitstream is not modified between these two reads. For more details on this secure OCM method of loading bitstreams, see the "Bitstream Authentication Using External Memory" section in the *Zynq UltraScale+ MPSoC Software Developer's Guide* (UG1137) [Ref 3]. If any of these partitions fail authentication or decryption, the FSBL sets the multi-boot register and initiates a soft reset. The boot process starts over with the CSU looking for a valid boot image in memory. If a valid boot image is not found, the device goes into secure lockdown and requires a POR to exit.

To complete the secure boot process, Linux and the software to be executed on the RPU (RPU SW) must be securely loaded by U-Boot. Linux and the RPU SW might be part of the boot.bin or they might be a single partition image that is resident in a different physical memory. In either case, U-Boot does not perform the authentication or decryption but rather calls the XilSecure library, which was securely loaded as part of the PMUFW. The XilSecure library executes out of internal PMU RAM, performs all of the security checks (i.e., PPK integrity check and PPK and SPK revocation checks), and uses the CSU accelerators to do the authentication or decryption. In the event of an authentication or decryption failure, the XilSecure library passes the failure status to U-Boot.

System Configuration

Systems with external DRAM

The HWRoT secure boot can be achieved differently in systems with external DRAM based on specific requirements and whether the external DRAM is considered secure.

Note: 2019.1 development tools, or subsequent releases, are used.

- Non-bitstream partitions are authenticated and/or decrypted by the FSBL or XilSecure. In both cases, the external DRAM, which is the final destination, is considered secure.
    - The FSBL will copy the partition data from external non-volatile memory to the specified DRAM address and then authenticate and/or decrypt in place.
    - XilSecure, when called, will authenticate and/or decrypt at the destination DRAM address. The partition must be copied into external DRAM before calling XilSecure.
- Bitstream partitions can be loaded by the FSBL or XilFPGA.
    - The bitstream partition can be authenticated and/or decrypted in external DRAM by XilSecure and then loaded, in plain-text form, using XilFPGA. In this scenario, the external DRAM is assumed secure since authentication and decryption occurs in external DRAM.
    - The bitstream partition can be authenticated by XilFPGA while in external DRAM. Once authentication is complete, XilFPGA will read the partition into the device where it is decrypted by the AES engine and then loaded into the programmable logic. Since authentication is performed in external DRAM, the external DRAM is assumed to be secure.

    - The FSBL or XilFPGA can be used to authenticate and decrypt the bitstream using the Secure OCM method. This method does not require the DRAM to be secure.

Systems without external DRAM

Secure boot in systems without external DRAM is supported when the FSBL is used to load the bitstream and non-bitstream partitions.

- Non-bitstream partitions can only be loaded by the FSBL. The FSBL will copy the partition data from external non-volatile memory to the internal memory location and then authenticate and/or decrypt in place. XilSecure does not support loading a non-bitstream directly from external non-volatile memory.
- Bitstream partitions can only be loaded by the FSBL. The FSBL utilizes the Secure OCM method to load the bitstream. XilFPGA does not support loading a bitstream directly from external non-volatile memory.

Secure boot in systems with external DRAM can be achieved differently based on specific requirements and whether the external DRAM is considered secure.

Note: 2019.1 development tools, or subsequent releases, are used.

- Non-bitstream partitions are decrypted by the FSBL or XilSecure. In both cases, the external DRAM, which is the final destination, is considered secure.
    - The FSBL will copy the partition data from external non-volatile memory to the specified DRAM address and then decrypt in place.
    - XilSecure, when called, will decrypt at the destination DRAM address. The partition must be loaded into external DRAM before calling XilSecure.
- Bitstream partitions can be loaded in multiple ways in systems with external DRAM.
    - The bitstream partition could be decrypted in external DRAM by XilSecure and then loaded, in plain-text form, using XilFPGA. In this scenario, the external DRAM is assumed secure.
    - XilFPGA can be called to read the bitstream partition into the device where it is decrypted by the AES engine and then loaded into the programmable logic. The partition must be loaded into external DRAM before calling XilFPGA. Since the decryption is performed internal to the device, this method does not require the DRAM to be secure.

Secure boot in systems without external DRAM is supported when FSBL is used to load the bitstream and non-bitstream partitions.

- Non-bitstream partitions (e.g. application software) can only be loaded by the FSBL. The FSBL will copy the partition data from external non-volatile memory to the internal memory location and then decrypt in place. XilSecure does not support loading a non-bitstream directly from external non-volatile memory.

- Bitstream partitions can only be loaded by the FSBL. The FSBL will read the partition data from external non-volatile memory and then send it for decryption and load into the configuration memory. XilFPGA does not support loading a bitstream directly from external non-volatile memory.

DPA Resistance

DPA resistance is achieved by authentication before decryption and by key rolling. Authentication before decryption, using the RSA algorithm, prevents an adversary from acquiring additional data per key by substituting their own data for the data contained in the boot image. Key rolling limits the amount of data encrypted on any given key. The amount of data encrypted by a key is configurable by the user.

Rolling Keys

The AES-GCM accelerator supports the rolling keys feature, where the entire encrypted image is represented in terms of smaller AES encrypted messages. Each message is encrypted using its own unique key. The initial key is stored at the key source on the device (e.g., BBRAM or eFUSE), while keys for each successive message are encrypted (wrapped) in the prior message. During boot, all partitions can be decrypted through key rolling. In Figure 12-11, “IV” illustrates the decryption flow and image format for the PMU firmware and FSBL. The same format is used for other partitions.

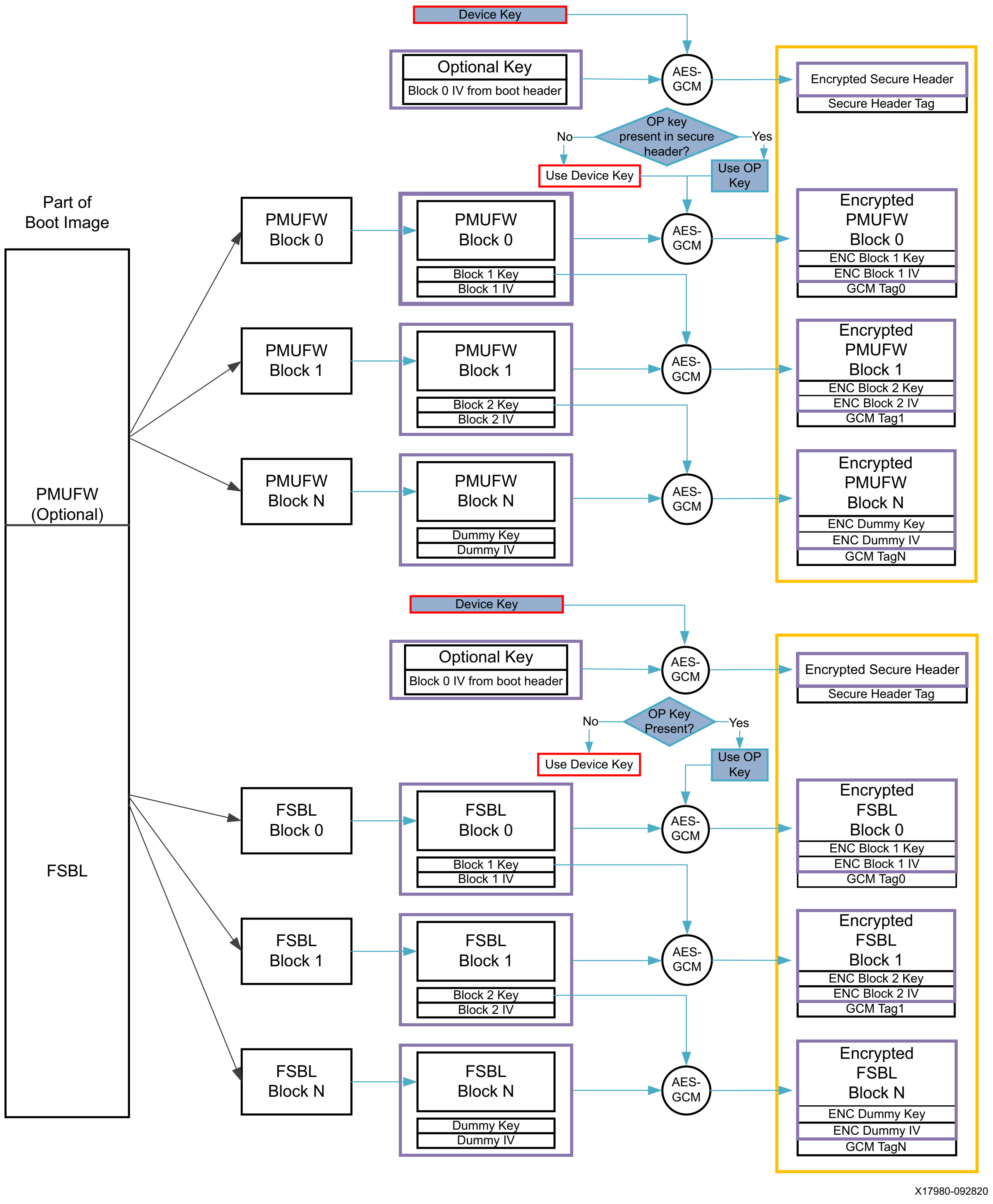


Figure 12-11: **Key Rolling**

Integration and Test Support (BH RSA Option)

Developing secure systems is always a challenge due to the limited, or non-existent, integration and test capabilities that exist once secure features are enabled. To assist integration and test efforts, the ability to command a hardware root of trust via the configuration file is provided. The BH RSA option is set in bootgen. This commands the device to boot using the root of trust without having to program the eFUSEs that force authentication. Authenticated or unauthenticated boots can now be performed during the integration and test phase. The functionality that is *not* performed in this mode include the following.

- Does not validate the integrity of the PPK (this would require eFUSEs to be programmed).
- Does not validate the SPK ID (this would require eFUSEs to be programmed).

Clearly this mode should *not* be used in a fielded system since this portion of the configuration file is *not* authenticated and could easily be modified by an adversary. If the BH RSA option is set *and* the RSA_EN eFUSEs are programmed to force authentication, the device will go back to boot and continue to search for a valid boot image. If a valid boot image is not found then the device will enter secure lockdown.

Hardware Root of Trust Only Boot (Auth_Only Option)

The CSU automatically locks out the AES key, stored in either BBRAM or eFUSEs, as a key source to the AES engine if the FSBL is not encrypted. This prevents using the BBRAM or eFUSE as the key source to the AES engine during run-time applications.

Note: A user key can still be used by loading it into the key update register (KUP).

Systems that choose not to encrypt the FSBL and employ only the hardware root of trust boot mechanism can still use the AES key, post-boot, if the Auth-Only option is set.

After a hardware root of trust boot, to leverage the AES cryptographic accelerator *and* use the key stored in either the BBRAM or eFUSE as a potential key source, the Auth Only option must be selected in bootgen.

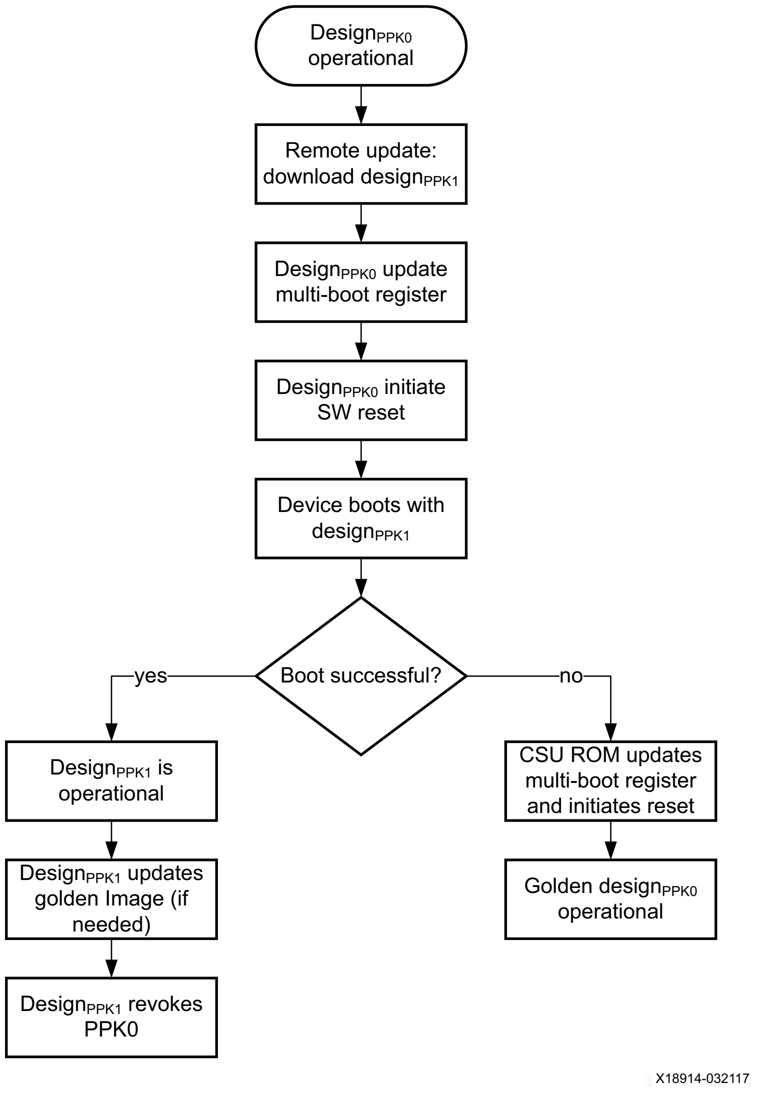
Note: This option is part of the configuration file that is authenticated.

Key Revocation

Key revocation is an integral part of any public key system. Whether keys are being changed and revoked due to good key management practices or in the unfortunate case where a private key is compromised, the ability to revoke a key is a necessary function. This section describes how to revoke both the PPK and SPK, as well as how to use the revocation as a permanent and temporary penalty.

PPK Revocation

There are two PPKs in Zynq UltraScale+ MPSoC. Each PPK has a set of invalid bits, (PPK0_INVLD and PPK1_INVLD) implemented as eFUSEs, that can be programmed to permanently revoke the PPK from use. If either of these eFUSEs is programmed, the PPK is revoked. Figure 12-12 shows a notional and proposed method to perform a remote update forced by the revocation of the PPK.



Design$_{PPK0}$ operational
Remote update: download design$_{PPK1}$
Design$_{PPK0}$ update multi-boot register
Design$_{PPK0}$ initiate SW reset
Device boots with design$_{PPK1}$
Boot successful?
yes
no
Design$_{PPK1}$ is operational
Design$_{PPK1}$ updates golden Image (if needed)
Design$_{PPK1}$ revokes PPK0
CSU ROM updates multi-boot register and initiates reset
Golden design$_{PPK0}$ operational
X18914-032117

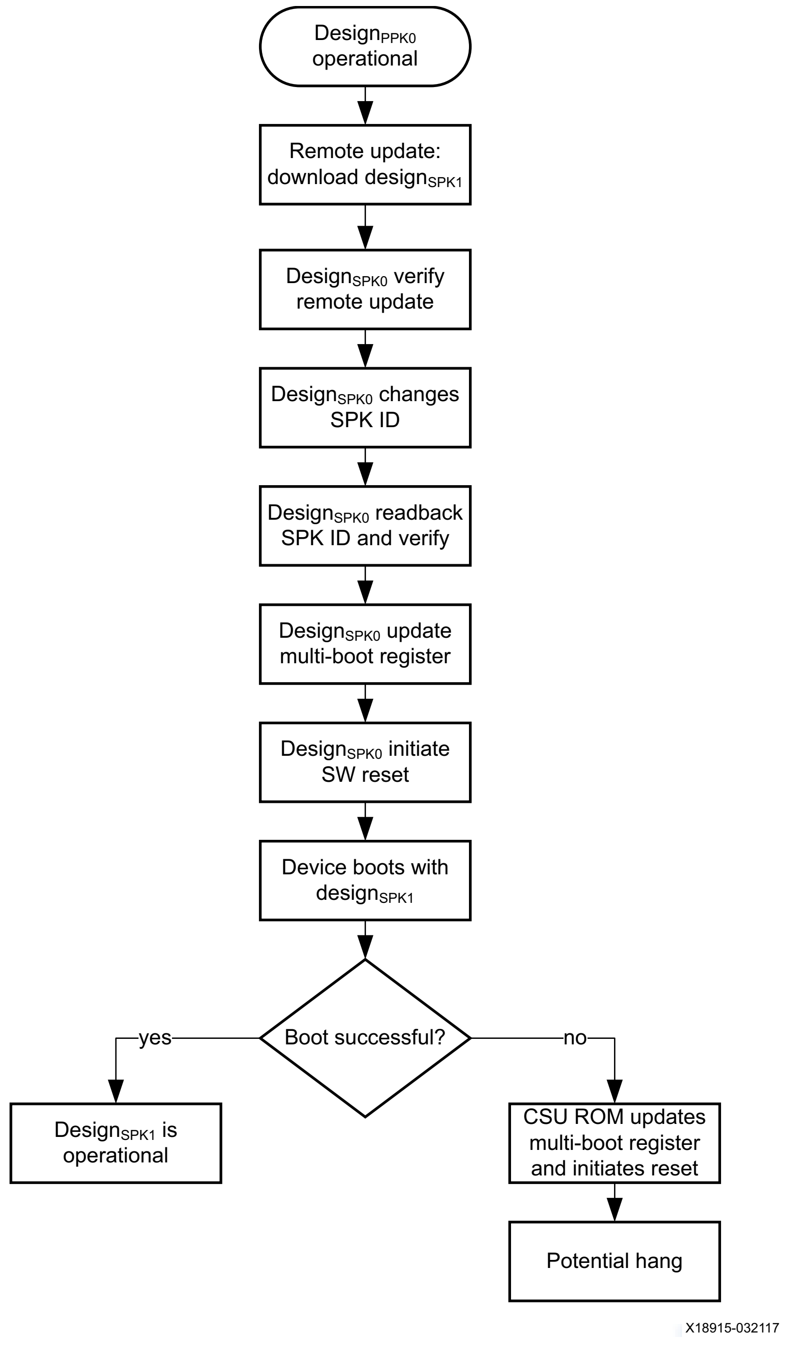
Figure 12-12: **PPK Revocation**

In the notional system, it is assumed that resident in external memory is the design authenticated with PPK0 (Design$_{PPK0}$) and a golden image, also signed with PPK0 (Golden Design$_{PPK0}$). Some applications choose to use a golden image as a backup. The golden image is not full-featured, but provides basic diagnostic and communication in the event of a failed boot of the primary image. Again, this is a representative system used to describe the process of updating a system in the event of a primary key revocation, and not a requirement.

The initial design, Design$_{PPK0}$, is notified when a remote update is being performed (in many cases the design itself is responsible for supporting the remote update). Design$_{PPK0}$ writes to the multi-boot register and then initiates a reset. Design$_{PPK1}$ is booted, and if successful, begins operation. Design$_{PPK1}$ should update the golden image (if necessary) and then program the eFUSEs to revoke PPK0. In the event of a failed boot, the CSU updates the multi-boot register and initiates a reset. As the golden image is stored at a higher address in external memory, it is ultimately loaded and communication is established. For more information on the golden image, see Golden Image Search in Chapter 11.

Standard SPK Revocation

Revocation of the SPK is very different than the PPK since the SPK, and its associated ID, are delivered to the Zynq UltraScale+ MPSoC as part of the programming image and authenticated with the PPK. To revoke an SPK, change the SPK ID implemented as eFUSEs inside of the device. If a device boots with an old SPK and SPK ID, the CSU recognizes that the IDs do not match and keeps the device from booting. Figure 12-13 shows a notional and proposed method to perform a remote update forced by the revocation of the SPK. There are two very important steps that are performed to avoid any complications in the revocation process.



Design$_{PPK0}$ operational
Remote update: download design$_{SPK1}$
Design$_{SPK0}$ verify remote update
Design$_{SPK0}$ changes SPK ID
Design$_{SPK0}$ readback SPK ID and verify
Design$_{SPK0}$ update multi-boot register
Design$_{SPK0}$ initiate SW reset
Device boots with design$_{SPK1}$
Boot successful?
yes
no
Design$_{SPK1}$ is operational
CSU ROM updates multi-boot register and initiates reset
Potential hang
X18915-032117

Figure 12-13: **SPK Revocation**

Once the new design, Design$_{SPK1}$, is loaded into external memory, Design$_{SPK0}$ verifies the integrity of Design$_{SPK1}$. The method to verify the integrity of Design$_{SPK1}$ is a user decision. One recommendation is to send a hash of Design$_{SPK1}$ with the remote update. Design$_{SPK0}$ could then read back Design$_{SPK1}$ from external memory, calculate its hash, and then compare it to what was delivered. Design$_{SPK0}$ now revokes the SPK by changing the programming of a new SPK ID into the SPK ID eFUSEs. Design$_{SPK0}$ reads back the SPK ID to confirm it was programmed correctly. Once these verification steps have been performed, Design$_{SPK0}$ can now update the multi-boot register and initiate a software reset for the system to boot using Design$_{SPK1}$.

Enhanced SPK Revocation

An alternative method of SPK revocation, called enhanced SPK revocation, utilizes the user eFUSEs (USER_{0:7}) in addition to the SPK ID eFUSEs. This approach provides these advantages over the standard SPK revocation method:

- An increase in the number of revocations – up to 256
- Allows each partition to have its own SPK, which allows one partition to be invalidated without invalidating all partitions
- Reduces the risk of failure during an upgrade process

IMPORTANT: *The enhanced SPK revocation is not applicable for the FSBL loaded by the CSU ROM. The standard SPK revocation is used on the FSBL. Everything else that is securely loaded during the boot process can use the enhanced SPK method.*

The enhanced SPK revocation uses the user eFUSEs, rather than SPK ID eFUSEs to determine if an SPK has been revoked. Since the user eFUSEs are a general purpose resource, it is important to allocate as many eFUSEs as are required in the architecture to avoid conflict.

Note: All user eFUSEs do not have to be used. As many eFUSEs as needed can be allocated.

When using enhanced SPK revocation, the user eFUSE represents which SPK has been revoked, thereby allowing many SPKs to be valid at one time. Revocation of the SPK occurs when the specific user eFUSE has been programmed. The authenticated boot image specifies which revocation method is employed and the FSBL, XilSecure, and XilFPGA libraries check the appropriate eFUSEs (user or SPK ID). A user specifies which revocation method to use, on a partition by partition basis, by selecting either user-efuse or spk-efuse for the spk_select option in the BIF file. For more details, see the Bootgen User Guide (UG1283) [Ref 36].

Figure 12-14 below compares and contrasts the standard and enhanced SPK revocation methods.

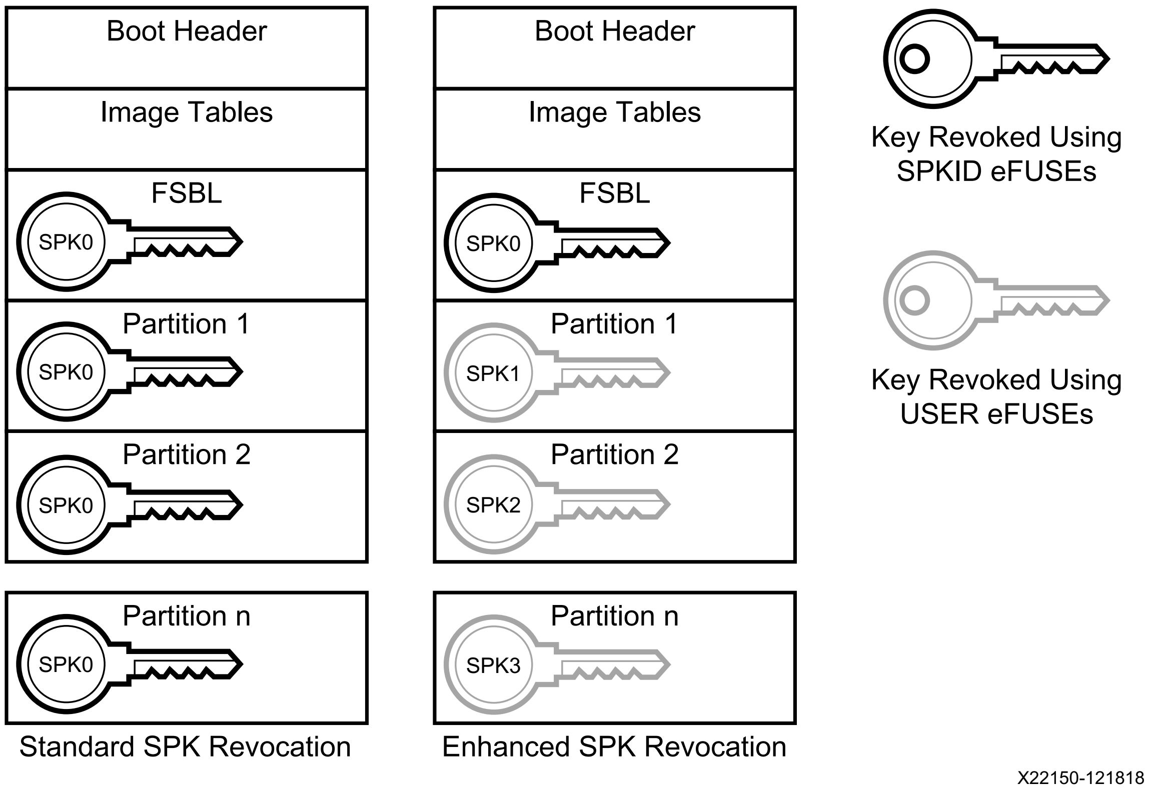


Figure 12-14: **Key Revocation in Boot Image**

Revocation as a Tamper Penalty

Key revocation has a valuable dual use role. Revocation can be used to inflict a penalty when a tamper event is detected. Programming both PPK invalid bits makes the device permanently inoperable (also known as a brick). While in some applications *bricking* the device is valuable, in other applications a temporary disabling is desired. In this situation, the SPK ID can be modified as a result of a tamper condition. This keeps the device from booting until the authorized user creates a new boot image with the correct SPK and SPK ID. In this scenario, the penalty is temporary until a new boot image is loaded, either remotely or when a system is returned to a depot.

Encrypt Only Secure Boot Details

The Zynq UltraScale+ MPSoC hardware encrypt only secure boot is based solely on the confidentiality and symmetric authentication provided by AES-GCM. The encrypt only secure boot mode is enabled by programming the ENC_ONLY eFUSE. This eFUSE forces the device to decrypt every partition using the AES-GCM key stored in the eFUSEs.

The CSU, FSBL, XilSecure, and XilFPGA libraries loaded by the CSU decrypts every partition (e.g., FSBL, PMUFW, ATF, bitstream, U-Boot, etc.) loaded. The CSU ROM reads the ENC_ONLY eFUSE, sees that it is programmed and then automatically decrypts the FSBL. Since the CSU processes the FSBL and PMUFW as two separation partitions, the AES key and IV are reused if the PMUFW is part of the bootloader. The reuse of the AES key and IV is a violation of the AES standard. Consequently, the PMUFW must be loaded by the FSBL, and not the CSU, in the encrypt only secure boot mode. Once the FSBL has been decrypted and authenticated (using symmetric authentication provided by AES-GCM) in internal memory, the CSU releases the reset to the specified processing unit (APU or RPU). At this stage, control is handed over to the user and the user is responsible for maintaining the established security. The remaining secure boot process is configurable by the user. An example of an encrypt only secure boot process is shown in Figure 12-15.

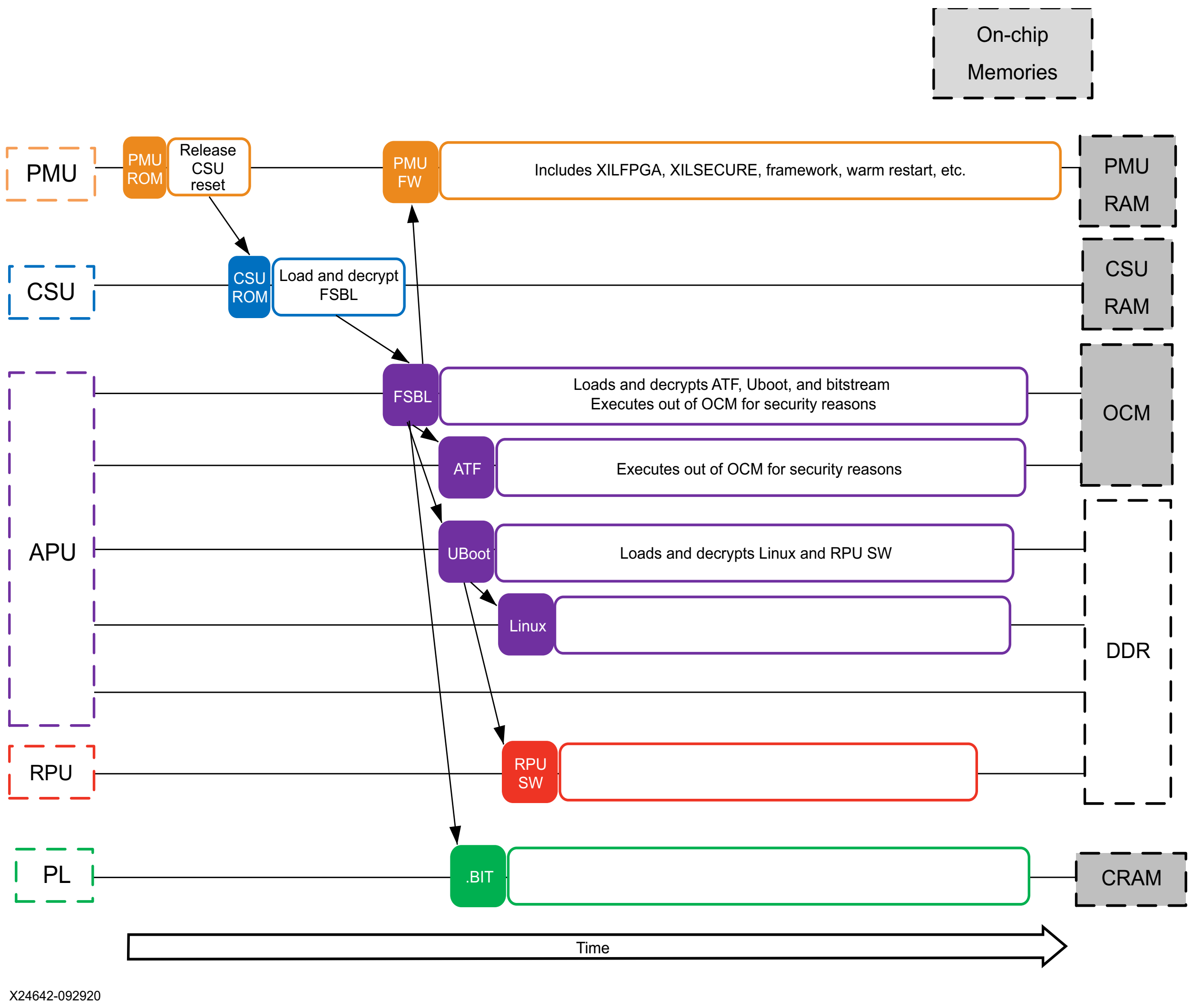

On-chip Memories
PMU
PMU ROM
Release CSU reset
PMU FW
Includes XILFPGA, XILSECURE, framework, warm restart, etc.
PMU RAM
CSU
CSU ROM
Load and decrypt FSBL
CSU RAM
FSBL
Loads and decrypts ATF, Uboot, and bitstream
Executes out of OCM for security reasons
OCM
ATF
Executes out of OCM for security reasons
APU
UBoot
Loads and decrypts Linux and RPU SW
Linux
DDR
RPU
RPU SW
PL
.BIT
CRAM
Time
X24642-092920

Figure 12-15: **Encrypt Only Secure Boot Example**

In this example, the FSBL decrypts the Arm trusted firmware (ATF), U-Boot, and the PL bitstream, which are all individual partitions. The FSBL executing at EL3 and using the

AES-GCM accelerator decrypts each partition using the device key stored in either eFUSE or BBRAM. However, the eFUSE key *must* be used when you use ENC_ONLY.

IMPORTANT: *If the bitstream is encrypted with the device key, the key stored in eFUSE or BBRAM, the bitstream must be decrypted and loaded by the FSBL. If the bitstream is encrypted with a user-provided key, U-Boot and Linux can command the bitstream load and decryption via the XILFPGA library. For more details, see the Loading Bitstreams section.*

IMPORTANT: *The PUF is disabled and not supported for use in the encrypt only secure boot mode.*

If any of these partitions fail the decryption GCM-tag check, the FSBL sets the multi-boot register and initiates a soft reset. The boot process starts over with the CSU looking for a valid boot image in memory. If a valid boot image is not found, the device goes into secure lockdown and requires a POR to exit.

To complete the secure boot process, Linux and the RPU SW must be decrypted and loaded by U-Boot. Linux and the RPU SW can be part of the boot.bin or they can be a single partition image that is resident in a different physical memory. In either case, U-Boot does not perform the decryption but rather calls the XilSecure library, which was securely loaded as part of the PMUFW. XilSecure executes out of internal PMU RAM and uses the AES-GCM accelerator to perform the decryption.

Note: Partitions that are decrypted by U-Boot must be encrypted with a user-provided key and not the device key.The user-provided key is identified via the XilSecure API.

If there is a decryption failure, while using the GCM-tag check, XilSecure passes the failure status to U-Boot.

There are two important considerations of the encrypt only secure boot mode that may require you to provide system-level protections. First, the partitions are not authenticated before decryption. The symmetric authentication occurs at the end of the decryption cycle. This means that the device is subject to a DPA random-data attack. Hence, you should provide system-level protections if the DPA attack vector is a concern. For more information, see DPA Resistance. Second, the boot and partition headers are not authenticated. Without authentication of these headers, anyone with access to the boot image, can modify the control fields resulting in incorrect secure boot behavior. One such example, is modification of the destination execution address. This address represents the start instruction address for a loaded partition. Anyone with access to the boot image could modify the address, causing the device to jump to an arbitrary memory location to modify or bypass the secure boot process. Hence, you should provide system-level protections if the lack of authentication of the boot and partition headers is a concern.

Loading Bitstreams

In the example secure boot processes described previously, the bitstream was loaded by the FSBL. The bitstream can also be loaded, authenticated, and/or decrypted by U-Boot or

Linux. In this scenario, U-Boot or Linux calls the XilFPGA library, which was securely loaded as part of the PMUFW to perform the security operations. XilFPGA executes out of internal PMU RAM, performs all of the security checks, and uses the CSU accelerators to do the authentication and/or decryption.

In the hardware root of trust secure boot mode, bitstreams can be authenticated, or authenticated and decrypted with either the device key or a user provided key. In the encrypt only secure boot mode, the bitstream is decrypted using the eFUSE device key loaded by the FSBL. Bitstream authentication and decryption is supported for the FSBL, standalone XilFPGA drivers, U-Boot and Linux using either device keys or user keys for both full bitstreams and partial reconfiguration bitstreams.

Secure Boot Image Format

The secure boot image format is shown in Figure 12-16.

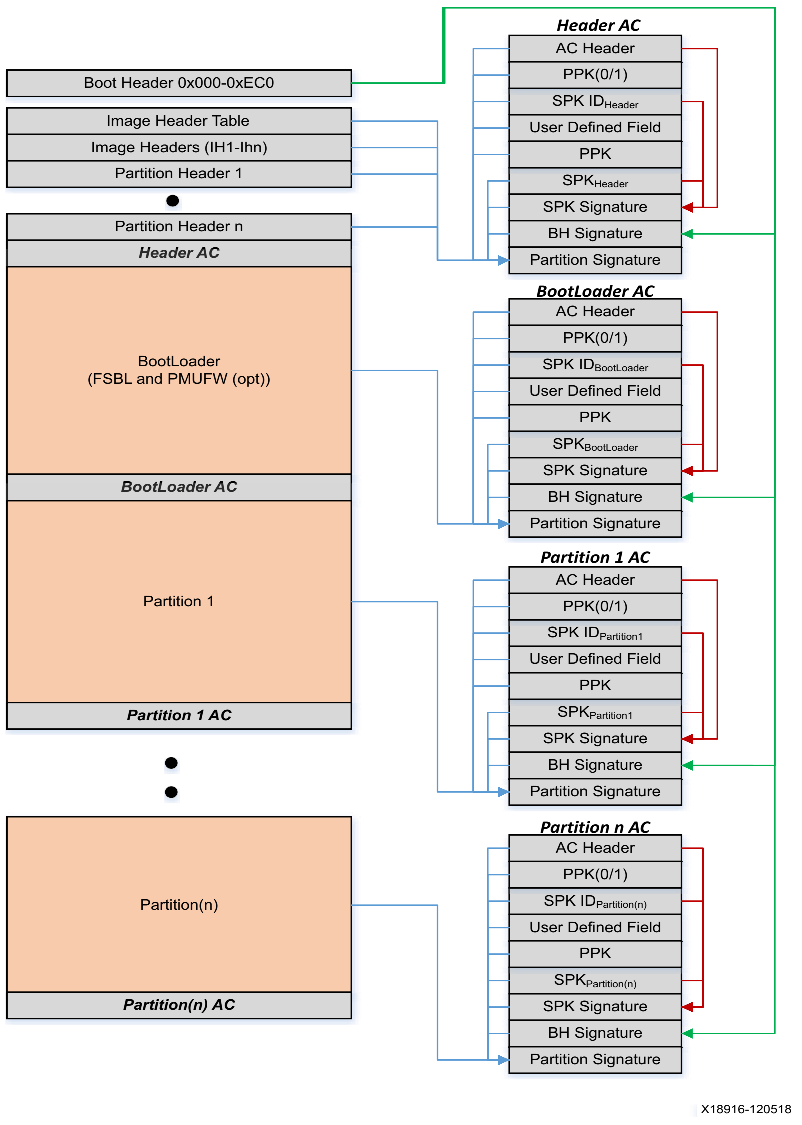


Figure 12-16: **Secure Boot Image**

There are multiple authentication certificates (AC) within a boot image. The authentication certificates include:

- Header AC: authentication certificate for the image header table and partition headers.
- Bootloader AC: authentication certificate for the bootloader (FSBL and optionally the PMUFW).
- Partition AC: authentication certificate for each partition in the image.

The equations for each signature within an AC (SPK, boot header, and partition) are listed here.

- SPK signature – the 512 bytes of the SPK signature is generated by this calculation:

    - SPK signature = RSA(PSK, padding || SHA(SPK+ auth_header))
- Boot header signature – the 512 bytes of the boot header signature is generated by this calculation:
    - Boot header signature = RSA(SSK, padding || SHA(boot header))
- Partition signature – the 512 bytes of the partition signature is generated by this calculation:
    - Partition signature = RSA(SSK, padding || SHA(Partition + authentication certificate))

Table 12-16 provides a summary of which asymmetric private key is used, and which SHA padding is used, for each signature within an AC

Table 12-16: **Authentication Certificates Signatures**

AC	Signature	SHA Padding Used	Private Key Used
Header AC	SPK signature	Keccak if standard key revocation; NIST if enhanced key revocation	PSK
	BH signature	Keccak	$\text{SSK}_{\text{Header}}$
	Header signature	NIST	$\text{SSK}_{\text{Header}}$
BootLoader AC	SPK signature	Keccak	PSK
	BH signature	Keccak	$\text{SSK}_{\text{BootLoader}}$
	BootLoader signature	Keccak	$\text{SSK}_{\text{BootLoader}}$
Partition AC	SPK signature	Keccak if standard key revocation; NIST if enhanced key revocation	PSK
	BH signature	Keccak	$\text{SSK}_{\text{Partition}}$
	Partition signature	NIST	$\text{SSK}_{\text{Partition}}$

Each part of the AC is described in the “Authentication Certificate” and “Authentication Certificate Header” sections in Chapter 16 of the *Zynq UltraScale+ MPSoC Software Developer’s Guide* (UG1137) [Ref 3].

Table 12-17 summarizes the encryption and authentication attributes of each portion of the secure boot image.

Table 12-17: **Secure Boot Image Encryption and Authentication**

Boot Image Block	Encrypted	Authenticated[(1)]	Notes
Boot header	No	Yes - signed with user secondary secret key (SSK)	Described in Table 11-4 and Table 11-5 of this TRM. A signature of the BH is provided in each AC.
Image header table	No	Yes - signed with user SSK	Described in the "Image Header Table" section of the *Zynq UltraScale+ MPSoC Software Developer's Guide* (UG1137) [Ref 3].
Image headers	No	Yes - signed with user SSK	Not currently used.
Partition headers	No	Yes - signed with user SSK	Described in the "Partition Header Tables" section of the *Zynq UltraScale+ MPSoC Software Developer's Guide* (UG1137) [Ref 3]. There is one partition header for each partition within the boot image.
FSBL secure header	Dependent on secure boot mode[(2)]	Yes - signed with user SSK	This is part of the FSBL that minimizes the use of the device key. The FSBL secure header contains the key and IV used to decrypt the FSBL. See *Bootgen User Guide* (UG1283) [Ref 36] for more details on Secure Header use. Only included when the OP key option is chosen. See Minimizing Use of the AES Boot Key (OP Key Option).
FSBL	Dependent on secure boot mode[(2)]	Yes - signed with user SSK	
PMUFW secure header	Dependent on secure boot mode[(2)]	Yes - signed with user SSK	This is part of the PMUFW and minimizes the use of the device key. The PMUFW Secure Header contains the key and IV used to decrypt the PMUFW. See *Bootgen User Guide* (UG1283) [Ref 36] for more details on Secure Header use.
PMUFW	Dependent on secure boot mode[(2)]	Yes - signed with user SSK	The PMUFW can be included as part of the bootloader and consequently loaded by the CSU. Alternatively, it can be its own partition.
Partition secure header	Dependent on secure boot mode[(2)]	Yes - signed with user SSK	This is part of the partition that minimizes the use of the other device key. The Partition secure header contains the key and IV used to decrypt the partition. See *Bootgen User Guide* (UG1283) [Ref 36] for more details on Secure Header use.

Table 12-17: **Secure Boot Image Encryption and Authentication *(Cont'd)***

Boot Image Block	Encrypted	Authenticated[(1)]	Notes
Partition	Dependent on secure boot mode[(2)]	Yes - signed with user SSK	

Notes:

1. In hardware root of trust secure boot mode.
2. Required for encrypt only secure boot mode and optional for hardware root of trust secure boot mode.

Boot Options

Minimizing Use of the AES Boot Key (OP Key Option)

Good key management practices includes minimizing the use of secret or private keys. This can be accomplished using the OP key option enabled in bootgen. When enabled, the encrypted secure header in the FSBL will contain nothing more than the OP key, which is user specified, and the initialization vector (IV) needed for the first block of the configuration file. The result is that the AES key stored on the device, in either the BBRAM or eFUSEs, is used for only 384 bits, which significantly limits its exposure to side channel attacks. Figure 12-17 explains how the OP key is used to minimize the use of the AES device key and integrate into the key rolling technique described in DPA Resistance.

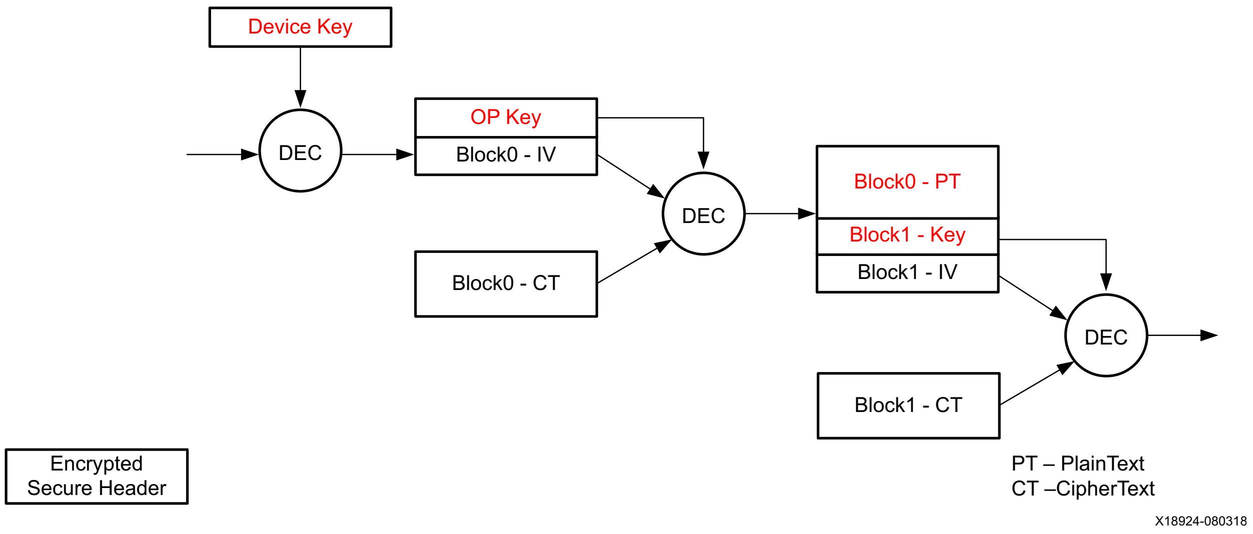


Figure 12-17: **OP Key**

The device key is used to decrypt the secure header which results in the OP key and the IV of the first block of the FSBL. The first encrypted block of the FSBL (shown as Block0-CT) is then decrypted using the IV and OP key. The result is the decrypted version of the first FSBL block (shown as Block0-PT) and the key and IV needed to decrypt the next block. This process continues until the entire FSBL is decrypted. Note that this process is entirely transparent. The bootgen option and the value of the OP key are user specified, but the rest is handled automatically by the tools and silicon.

Once the OP key is used, it becomes the device key and the device key selection cannot be changed back to the BBRAM or eFUSE key without a POR. The user can still choose between the KUP and the OP key (see Figure 12-2). When bootgen creates a single boot image with multiple encrypted partitions, it automatically encrypts the partitions with the OP key. However, if a customer chooses to create multiple boot images with encrypted partitions, the customer must provide bootgen the OP key value as the AES key so the partitions are encrypted correctly.

Protect Device Key in Development Environment with OP Key

The OP key has an added benefit in that it can be used to protect the device key in a development environment where some team members are responsible for managing the device key and other team members are not.

For example, Team A (Secure Team) and Team B (Not Secure Team) work collaboratively to build an encrypted image without sharing the secret red key. Team A manages the secret red key. Team B builds encrypted images for development and test but does not have access to the secret red key. Team A encrypts the boot loader with the device key (using the OP key option) and delivers the encrypted bootloader to Team B. Team B encrypts all the other partitions using the OP key. Team B takes the encrypted partitions they created and the encrypted boot loader from Team A and uses bootgen to combine everything into a single boot.bin. For more details, see "Using OP Key to Protect the Device Key in a Development Environment" in Chapter 8 of the *Zynq UltraScale+ MPSoC Software Developer's Guide* UG1137 [Ref 3].

Chapter 13

Interrupts

Introduction

Interrupts are pervasive within and between processors in the PS and PL. The system interrupts communicate status, events, requests, and errors within the heterogeneous processing system.

The platform management unit (PMU) and configuration security unit (CSU) have local interrupt controllers. The PMU interrupt controller is described in Chapter 6, Platform Management Unit. It includes the CPU and external interrupt controllers. The CSU interrupt controller is a closed system managed by the CSR ROM code.

The RPU uses the Arm PL-390 generic interrupt controller that is compliant to the GICv1 architecture specification. The APU MPCore uses the Arm GIC-400 generic interrupt controller and is compliant to the GICv2 architecture specification. The GIC manages the software-generated interrupts (SGI), each CPU’s private peripheral interrupts (PPI), and the shared peripheral interrupts (SPI).

The PMU uses the GIC proxy interrupts when the RPU and APU cannot service an interrupt because the processor is powered down. The GIC proxy is a Xilinx architecture for the PMU external interrupt controller and is controlled by the PMU.

The register documentation for the three system interrupt controllers is listed here.

- RPU GIC: PrimeCell Generic Interrupt Controller (PL390), DDI 0416B, r0p0.
- APU GIC: CoreLink GIC-400 Generic Interrupt Controller, DDI 0471B, r0p1.
- GIC Proxy: *Zynq UltraScale+ MPSoC Register Reference* (UG1087) [Ref 4], LPD_SLCR register set.

There are 148 system interrupts that connect to each GIC, the GIC proxy interrupt structure, and the PL fabric. The system interrupts are normally handled by the RPU or APU MPCores. The user firmware in the PMU can process system interrupts in the absence of an RPU or APU. The CSU does not connect to the system interrupts.

Note: An inter-processor interrupt (IPI) channel is associated with a processor target to allow other processors in the heterogeneous processing system to send it messages and receive a response in return. The IPI target processor receives an interrupt from another processor and accesses the message buffer with a prearrange communications protocol. The IPI channels can target the system processors: APU MPCore, RPU core 0, RPU core 1, four channels to processor(s) in the PL, and four private channels to the PMU.

Note: The PMU has four IPI interrupts. PMU_0 interrupt is assigned by the PMU firmware to transition the PMU to sleep mode.

GIC Features

Both GICs have many similar features:

- Multiprocessor environment for the MPCore.
- Arbitrate system interrupts to the CPU cores.
- Software generated interrupt:
  - Mechanism for one CPU to interrupt another CPU within an MPCore.
- Private peripheral interrupts IRQ/FIQ from PL.
- Shared peripheral interrupt:
  - Manage system interrupts from system elements.

RPU-specific GIC Features

The RPU GIC has some unique features:

- GICv1 programming model.
- Security extensions.

APU-specific GIC Features

The APU GIC includes most of the same features as in the RPU GIC with the addition of security and virtualization:

- GICv2 programming model.
- Security extensions.
- Virtualization extensions.
- Interrupt groups:
  - Group 0 interrupts cause either IRQ or IFQ signaling.
  - Group 1 interrupts use IRQ signaling.
  - Unified scheme to handle priority.
  - Optional register lockdown on some group 0 interrupts.
- CPU private registers have restricted accessed on the AXI interconnect.

GIC Proxy Interrupts

The GIC proxy manages all the system interrupts connected to the GIC SPI interrupts. These system interrupts set bits in the GICP{0:4}_IRQ_STATUS registers. After the masking registers, the bit in each register is OR'ed together to set bits in another status register that is OR'ed together to generate a single interrupt signal to the PMU external interrupt controller.

The GIC proxy interrupts are used by the PMU in a fall-back mode to handle system interrupts that cannot be managed by the application processors.

System Interrupts

The system interrupts are generated by many system elements and broadcast to the GICs, the PMU via the GIC proxy (GICPx_IRQ registers), and to output signals in the PL. The system interrupts are listed in Table 13-1. The table lists the IRQ numbers for the RPU and APU interrupt controllers, as well as the GIC proxy bit assignments.

Table 13-1: **System Interrupts**

IRQ Name	IRQ Number (GIC)	IDCICR[4]	Bits	Required Type	GICPx_IRQ Bits (GIC Proxy)	Description
RPU0_Perf_Mon	40	2	[17:16]	High level	GICP0 [8]	RPU0 performance monitor (ARM_PMU)[1].
RPU1_Perf_Mon	41	2	[19:18]	High level	GICP0 [9]	RPU1 performance monitor (ARM_PMU)[1].
OCM	42	2	[21:20]	High level	GICP0 [10]	OCM CE and UE ECC errors.
LPD_APB	43	2	[23:22]	High level	GICP0 [11]	OR of all APB slave interface errors in LPD.
RPU0_ECC	44	2	[25:24]	High level	GICP0 [12]	RPU0 errors combined: FPU, memory ECC, and APB access.
RPU1_ECC	45	2	[27:26]	High level	GICP0 [13]	RPU1 CE errors combined: FPU, memory, ECC, and APB access.
NAND	46	2	[29:28]	High level	GICP0 [14]	NAND memory controller.
QSPI	47	2	[31:30]	High level	GICP0 [15]	Quad-SPI controller.
GPIO	48	3	[1:0]	High level	GICP0 [16]	GPIO controller.
I2C0	49	3	[3:2]	High level	GICP0 [17]	I2C0 controller.

Table 13-1: **System Interrupts *(Cont'd)***

IRQ Name	IRQ Number (GIC)	IDCICR[4]	Bits	Required Type	GICPx_IRQ Bits (GIC Proxy)	Description
I2C1	50	3	[5:4]	High level	GICP0 [18]	I2C1 controller.
SPI0	51	3	[7:6]	High level	GICP0 [19]	SPI0 controller.
SPI1	52	3	[9:8]	High level	GICP0 [20]	SPI1 controller.
UART0	53	3	[11:10]	High level	GICP0 [21]	UART 0 controller.
UART1	54	3	[13:12]	High level	GICP0 [22]	UART 1 controller.
CAN0	55	3	[15:14]	High level	GICP0 [23]	CAN 0 controller.
CAN1	56	3	[17:16]	High level	GICP0 [24]	CAN 1 controller.
LPD_APM	57	3	[19:18]	High level	GICP0 [25]	OR of the LPD and OCM APM interrupts.
RTC_Alarm	58	3	[21:20]	High level	GICP0 [26]	RTC alarm interrupt.
RTC_Seconds	59	3	[23:22]	High level	GICP0 [27]	RTC seconds interrupt.
ClkMon	60	3	[25:24]	High level	GICP0 [28]	Clock monitor in LPD.
IPI_Ch7	61	3	[27:26]	High level	GICP0 [29]	IPIs targeting channel 7.
IPI_Ch8	62	3	[29:28]	High level	GICP0 [30]	IPIs targeting channel 8.
IPI_Ch9	63	3	[31:30]	High level	GICP0 [31]	IPIs targeting channel 9.
IPI_Ch10	64	4	[1:0]	High level	GICP1 [0]	IPIs targeting channel 10.
IPI_Ch1	65	4	[3:2]	High level	GICP1 [1]	IPIs targeting channel 1.
IPI_Ch2	66	4	[5:4]	High level	GICP1 [2]	IPIs targeting channel 2.
IPI_Ch0	67	4	[7:6]	High level	GICP1 [3]	IPIs targeting channel 0.
TTC0	68:70	4	[13:8]	High level	GICP1 [4:6]	Triple-timer counter 0.
TTC1	71:73	4	[19:14]	High level	GICP1 [7:9]	Triple-timer counter 1.
TTC2	74:76	4	[25:20]	High level	GICP1 [10:12]	Triple-timer counter 2.
TTC3	77:79	4	[31:26]	High level	GICP1 [13:15]	Triple-timer counter 3.
SDIO0	80	5	[1:0]	High level	GICP1 [16]	SDIO 0 controller.
SDIO1	81	5	[3:2]	High level	GICP1 [17]	SDIO 1 controller.
SDIO0_Wakeup	82	5	[5:4]	High level	GICP1 [18]	SDIO 0 wake-up interrupt.
SDIO1_Wakeup	83	5	[7:6]	High level	GICP1 [19]	SDIO 1 wake-up interrupt.

Table 13-1: **System Interrupts *(Cont'd)***

IRQ Name	IRQ Number (GIC)	IDCICR[(4)]	Bits	Required Type	GICPx_IRQ Bits (GIC Proxy)	Description
LPD_SWDT	84	5	[9:8]	High level	GICP1 [20]	LPD watchdog timer (wdt0). Edge sensitive trigger.[(2)]
CSU_SWDT	85	5	[11:10]	High level	GICP1 [21]	CSU and PMU watchdog timer. Edge sensitive trigger.[(2)]
LPD_ATB	86	5	I [13:12]	High level	GICP1 [22]	OR of all ATB timeout errors in LPD.
AIB	87	5	[15:14]	High level	GICP1 [23]	OR of all AIB errors on AXI and APB.
SysMon	88	5	[17:16]	High level	GICP1 [24]	OR of all system monitor interrupts.
GEM0	89	5	[19:18]	High level	GICP1 [25]	Ethernet 0 controller.
GEM0_Wakeup	90	5	[21:20]	High level	GICP1 [26]	Ethernet 0 wake-up interrupt.
GEM1	91	5	[23:22]	High level	GICP1 [27]	Ethernet 1 controller.
GEM1_Wakeup	92	5	[25:24]	High level	GICP1 [28]	Ethernet 1 wake-up interrupt.
GEM2	93	5	[27:26]	High level	GICP1 [29]	Ethernet 2 interrupt.
GEM2_Wakeup	94	5	[29:28]	High level	GICP1 [30]	Ethernet 2 wake-up interrupt.
GEM3	95	5	[31:30]	High level	GICP1 [31]	Ethernet 3 controller.
GEM3_Wakeup	96	6	[1:0]	High level	GICP2 [0]	Ethernet 3 wake-up interrupt.
USB0_Endpoint	97:100	6	[9:2]	High level	GICP2 [1:4]	USB 0 bulk transfer, isochronous transfer, controller interrupt, control transfer.
USB0_OTG	101	6	[11:10]	High level	GICP2 [5]	USB 0 OTG mode.
USB1_Endpoint	102:105	6	[19:12]	High level	GICP2 [6:9]	USB 1 bulk transfer, isochronous transfer, controller interrupt, control transfer.
USB1_OTG	106	6	[21:20]	High level	GICP2 [10]	USB 1 OTG mode.
USB0_Wakeup	107	6	[23:22]	High level	GICP2 [11]	USB 0 controller to wake-up PMU.
USB1_Wakeup	108	6	[25:24]	High level	GICP2 [12]	USB 1 controller to wake-up PMU.
LPD_DMA	109:116	6, 7	[31:26], [9:0]	High level	GICP2 [13:20]	Eight LPD DMA channels 0 to 7.

Table 13-1: **System Interrupts *(Cont'd)***

IRQ Name	IRQ Number (GIC)	IDCICR[4]	Bits	Required Type	GICPx_IRQ Bits (GIC Proxy)	Description
CSU	117	7	[11:10]	High level	GICP2 [21]	Configuration and security unit.
CSU_DMA	118	7	[13:12]	High level	GICP2 [22]	CSU DMA controller.
eFuse	119	7	[15:14]	High level	GICP2 [23]	eFuse interrupt.
LPD_XMPU_XPPU	120	7	[17:16]	High level	GICP2 [24]	OCM XMPU and XPPU protection units in LPD.
PL_PS_Group0	121:128	7, 8	[31:18], [1:0]	Rising edge/high level	GICP2 [25:31] GICP3[0]	PL to PS interrupt signals 0 to 7.[3]
Reserved	129:135	8	[15:2]	High level	GICP3 [1:7]	Seven reserved interrupts.
PL_PS_Group1	136:143	8	[31:16]	Rising edge/high level	GICP3 [8:15]	PL to PS interrupt signals 8 to 15.[3]
DDR	144	9	[1:0]	High level	GICP3 [16]	DDR memory controller.
FPD_SWDT	145	9	[3:2]	High level	GICP3 [17]	FPD system watchdog timer (wdt1). Edge sensitive trigger.[2]
PCIe_MSI0	146	9	[5:4]	High level	GICP3 [18]	PCIe MSI vectors 0 to 31.
PCIe_MSI1	147	9	[7:6]	High level	GICP3 [19]	PCIe MSI vectors 32 to 63.
PCIe_INTx	148	9	[9:8]	High level	GICP3 [20]	PCIe legacy: OR of INT A, B, C, and D interrupts.
PCIe_DMA	149	9	[11:10]	High level	GICP3 [21]	PCIe_DMA controller.
PCIe_MSC	150	9	[13:12]	High level	GICP3 [22]	PCIe_MSC controller.
DisplayPort	151	9	[15:14]	High level	GICP3 [23]	DisplayPort controller.
FPD_APB	152	9	[17:16]	High level	GICP3 [24]	OR of all APB slave interface errors in FPD.
FPD_ATB	153	9	[19:18]	High level	GICP3 [25]	OR of all ATB timeout errors in FPD.
DPDMA	154	9	[21:20]	High level	GICP3 [26]	DisplayPort DMA controller.
FPD_APM	155	9	[23:22]	High level	GICP3 [27]	OR of the CCI and DDR APM interrupts.
FPD_DMA	156:163	9, 10	31:24], [7:0]	High level	GICP3 [28:31] GICP4 [0:3]	Eight FPD DMA channels 0 to 7.

Table 13-1: **System Interrupts *(Cont'd)***

IRQ Name	IRQ Number (GIC)	IDCICR[4]	Bits	Required Type	GICPx_IRQ Bits (GIC Proxy)	Description
GPU	164	10	[9:8]	High level	GICP4 [4]	OR of all GPU interrupts.
SATA	165	10	[11:10]	High level	GICP4 [5]	SATA controller.
FPD_XMPU	166	10	[13:12]	High level	GICP4 [6]	FPD and memory XMPU protection units in FPD.
APU_VCPUMNT	167:170	10	[21:14]	High level	GICP4 [7:10]	Virtual processor interface maintenance.
CPU_CTI	171:174	10	[29:22]	High level	GICP4 [11:14]	CoreSight cross-trigger interface.
APU{0:3}_Perf_Mon	175:178	10, 11	[31:30], [5:0]	High level	GICP4 [15:18]	APU{0:3} performance monitors (ARM_PMU)[1].
APU{0:3}_Comm	179:182	11	[13:6]	High level	GICP4 [19:22]	Communications from APU cores 0 to 3.
L2_Cache	183	11	[15:14]	High level	GICP4 [23]	L2 cache uncorrectable ECC error.
APU_ExtError	184	11	[17:16]	High level	GICP4 [24]	APU AXI transaction with write error response.
APU_RegError	185	11	[19:18]	High level	GICP4 [25]	APU register access address decode error.
CCI	186	11	[21:20]	High level	GICP4 [26]	Cache coherent interconnect unit.
SMMU	187	11	[23:22]	High level	GICP4 [27]	System memory management unit.

Notes:

1. The ARM_PMU is a performance monitor unit developed by Arm and is different from the platform management unit (PMU) developed by Xilinx.
2. The system watchdog timers produce an interrupt pulse of at least four clock periods, which are programmable using the x_SWDT.MODE [IRQLN] bit field. The four clock pulse length is sufficiently long enough for normal situations. The GIC interrupt controllers must be programmed for edge sensitivity.
3. The minimum interrupt pulse width for detection is four clock periods of the GIC, which is normally a 100 MHz clock resulting in a minimum 40 ns pulse width. The signal synchronizers might detect a shorter pulse, but it is not guaranteed. Glitches should be avoided.
4. The configuration registers provided by the GIC distributor are mentioned under IDCICR as defined by Arm. For more details refer "Distributor Register Description" in Chapter 3 of the ARM Primecell Generic Interrupt Controller PL390 Technical Reference Manual r0p0 document number DDI0416B.

GIC Interrupt System Architecture

The system interrupt architecture includes GIC interrupt controllers for both MPCores, the GIC proxy interrupt unit for the PMU, and the IPI interrupts for system-level processor communications.

Interrupt Block Diagram

Figure 13-1 shows the block diagram of the processor interrupts. The shared peripheral interrupts are generated from various subsystems that include the I/O peripherals in the PS and logic in the PL. The PCIe MSI are handled by the controller for PCIe by decoding the MSI into a bit-vector and then asserting a sideband interrupt. To guarantee PCIe ordering, the controller for PCIe must wait for the completion of previously outstanding (inbound) writes before asserted an MSI interrupt. Also, the controller for PCIe must ensure that the MSI buffer, which holds the MSI information after asserting interrupt to CPU, does not end up stalling the PCIe inbound traffic (which can cause deadlock).

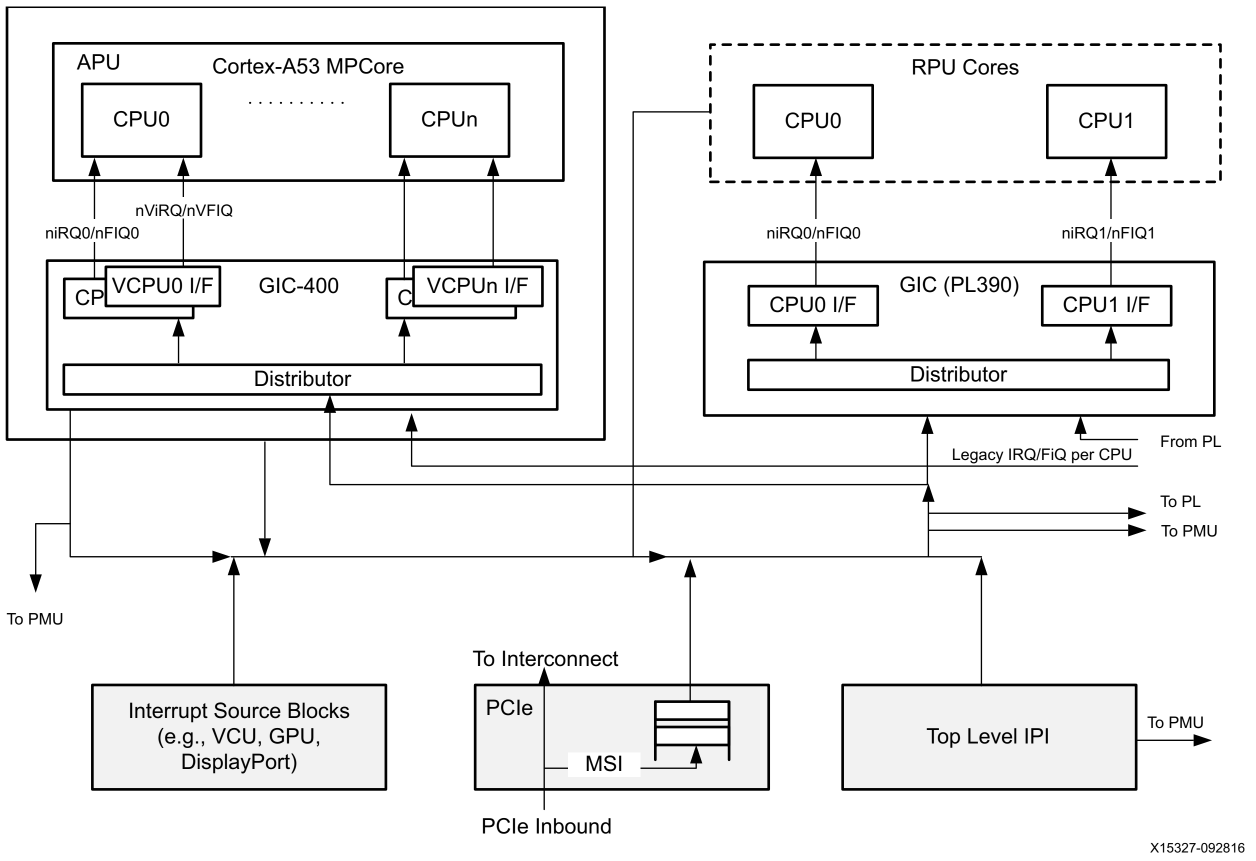


Figure 13-1: **GIC Interrupts Block Diagram**

RPU GIC Interrupt Controller

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

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

The APU MPCores processors access the RPU_GIC interrupt controller (Figure 13-2) through their peripheral interface. 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. Because of this, 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.

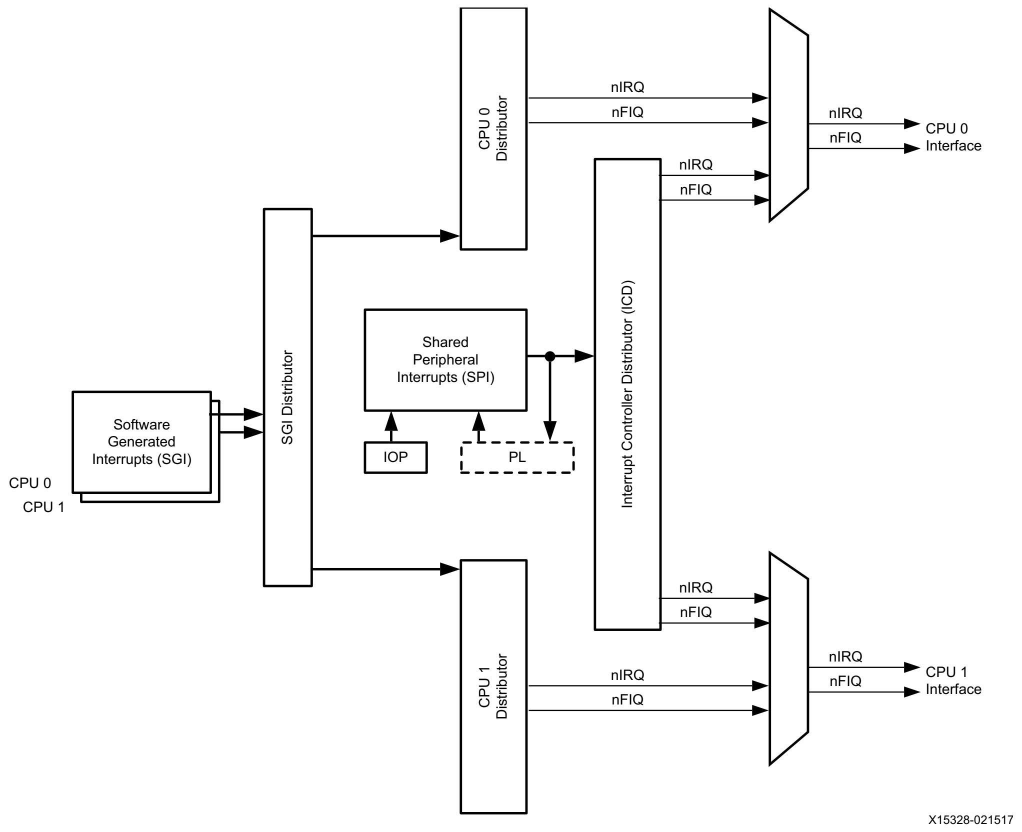


Figure 13-2: **RPU Interrupt Controller Block Diagram**

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 CPU(s). 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 approximately 160 shared peripheral interrupts (SPIs) 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.

SPI Interrupt Sensitivity

The shared peripheral interrupts (SPI) can be targeted to any number of 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 IRQ (121) through IRQ(128) and IRQ (136) through IRQ(143), which are the interrupts from the PL, all interrupt sensitivity types are fixed by the requesting sources and cannot be changed. The GIC must be programmed to accommodate this. The BootROM does not program these registers; therefore 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 CPU_2x3x periods. This requirement applies to any IRQ-F2P[n] (from PL) with a rising-edge sensitivity type. See Answer Record 69390 for more information.

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 and PPI 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 to the CPU interfaces the highest pending interrupt. It receives back the information that the interrupt is acknowledged and can now change the status of the corresponding interrupt. Only the CPU that acknowledges the interrupt can end that interrupt.

APU GIC Interrupt Controller

The APU uses an external GICv2 controller as a central resource to support and manage interrupts. There are peripheral interrupts, software generated interrupts, and virtual interrupts.

Peripheral Interrupts

Peripheral interrupts are asserted by a signal to the GIC. The GIC architecture defines the following types of peripheral interrupts.

- Private peripheral interrupt (PPI) is a peripheral interrupt that is specific to a single processor.
- Shared peripheral interrupt (SPI) is a peripheral interrupt that the distributor can route to any of a specified combination of processors. These are wired interrupts coming from various sources to the GIC.

Each peripheral interrupt is either edge-triggered or level-sensitive.

Software-generated Interrupts

Software-generated interrupts (SGIs) are generated by software writing to a GICD_SGIR register in the GIC. The system uses SGIs for inter-processor communication.

An SGI has edge-triggered properties. The software triggering of the interrupt is equivalent to the edge transition of the interrupt request signal.

Virtualization Extensions

GIC virtualization extensions are used when an virtual SGI occurs. Management registers in the GIC virtualization extensions enable the requesting processor to be reported to the guest OS, as required by the GIC specifications. By writing to the management registers in the GIC virtualization extensions, a hypervisor can generate a virtual interrupt that appears to a virtual machine as an SGI.

Virtual Interrupt

A virtual interrupt targets a virtual machine running on a processor and is typically signaled to the processor by the connected virtual CPU interface.

APU Interrupt Partitioning

This section covers the partitioning of the GICv2.

The distributor block performs interrupt prioritization and distribution to the CPU interface blocks that connect to the processors in the system.

Each CPU interface block performs priority masking and preemption handling for a connected processor in the system.

The GIC virtualization extensions add a virtual CPU interface for each processor in the system. Each virtual CPU interface is partitioned into the following blocks.

- Virtual interface control: The main component of the virtual interface control block is the GIC virtual interface control registers. These registers include a list of active and pending virtual interrupts for the current virtual machine on the connected processor. Typically, these registers are managed by the hypervisor that is running on that processor.
- Virtual CPU interface: Each virtual CPU interface block provides physical signaling of virtual interrupts to the connected processor. The Arm processor virtualization extensions signal these interrupts to the current virtual machine on that processor. The GIC virtual CPU interface registers, accessed by the virtual machine, provide interrupt control and status information for the virtual interrupts.

APU Interrupt Grouping and Virtualization

A virtual machine running on a processor communicates with a virtual CPU interface on the GICv2 (Figure 13-3). The virtual machine receives virtual interrupts from this interface, and cannot distinguish these interrupts from physical interrupts.

A hypervisor handles all IRQs, translating those destined for a virtual machine into virtual interrupts, and, in conjunction with the GIC, manages the virtual interrupts and the associated physical interrupts. It also uses the GIC virtual interface control registers to manage the virtual CPU interface. As part of this control, the hypervisor updates the List registers that are a subset of the GIC virtual interface control registers. In this way the hypervisor and GIC together provide a virtual distributor that appears to a virtual machine as the physical GIC distributor.

The GIC virtual CPU interface signals virtual interrupts to the virtual machine, subject to the normal GIC handling and prioritization rules.

- Secure software assigns the following.
  - Secure interrupts to group 0, signaled to the processor as FIQs
  - Non-secure interrupts to group 1, signaled to the processor as IRQs.
- A hypervisor is used for the following.
  - Implements a virtual distributor, using features of the virtualization extension on the GIC. This virtual distributor can virtualize IRQ interrupts from the GIC as virtual IRQ and virtual FIQ interrupts, which it routes to an appropriate virtual machine.
  - Routes physical IRQs to hypervisor mode, so they can be serviced by the virtual distributor.

When the GIC signals an IRQ to the processor, the interrupt is routed to hypervisor mode. The hypervisor determines whether the interrupt is for itself or for a guest OS. If it is for a guest OS it determines the following.

- The specific guest OS that must handle the interrupt.
- Whether that guest OS has configured the interrupt as an FIQ or as an IRQ
- The interrupt priority, based on the priority configuration by the target guest OS.

If the interrupt targets the current guest OS, the hypervisor updates the list registers, to add the interrupt to the list of pending interrupts for the current virtual machine.

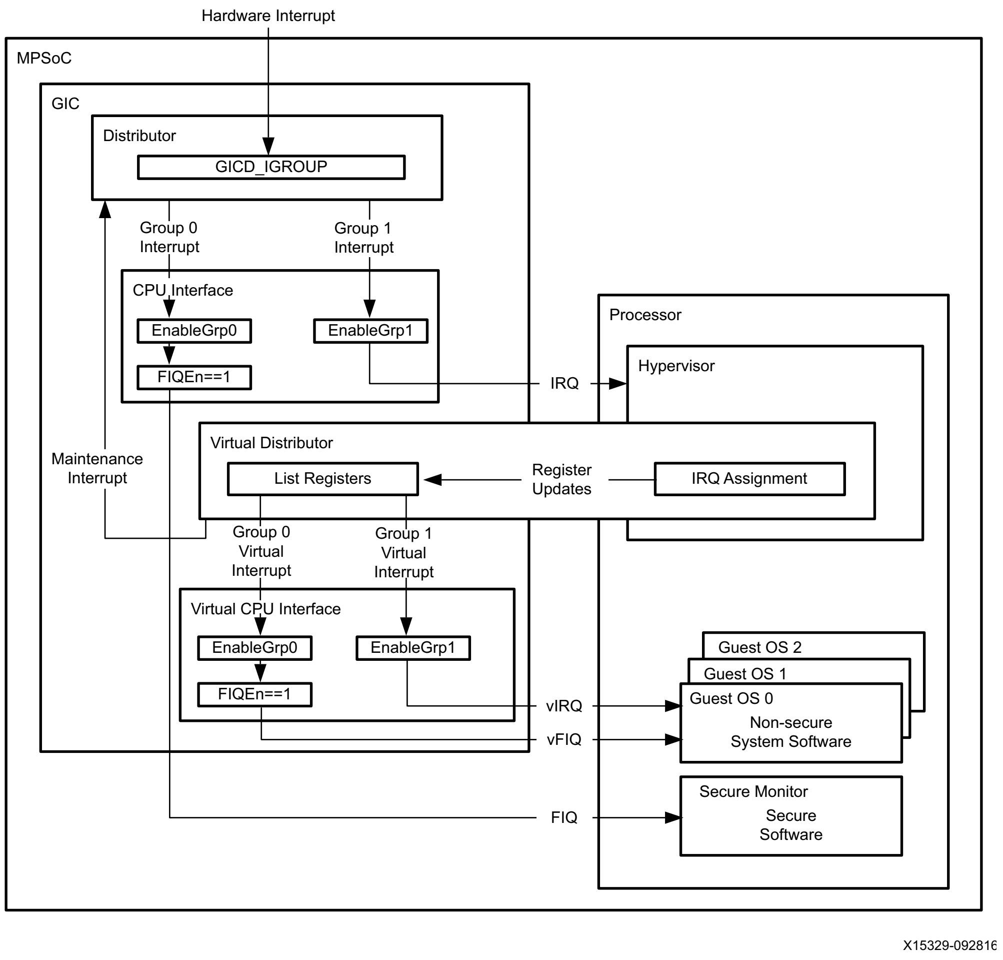


Figure 13-3: **APU with Interrupt Virtualization Block Diagram**

Note: The APU GIC is physically located on the AXI interconnect as an FPD slave, but should only be accessed by the APU MPCore. The FPD main interconnect switch can restrict access to the APU GIC. However, a PL master can access the APU GIC through the S_AXI_ACP_FPD interface and cannot be stopped by the FPD switch. The XMPU can be configured to block the S_AXI_ACP_FPD interface from accessing the APU GIC.

IPI Interrupts and Message Buffers

The heterogeneous multiprocessor system uses the inter-processor interrupt (IPI) structure to exchange short interrupt-driven messages between processors in the system. The IPI architecture allows the passing of messages across the system without the complications of autonomous read-write transactions and polling inefficiency.

- Four channels assigned to target the PMU.
- Seven channels can be assigned to target RPU core 0, RPU core 1, the APU MPCore, four processors in the PL, and four channels to the PMU (in addition to the dedicated channels).
- Register access is restricted to a processor by the XPPU protection unit.

Note: The IPI channel registers can be owned by any of the masters except the interrupts for the PMU channels are only routed to the PMU.

Processor communications include both an IPI interrupt structure and memory buffers to exchange short private 32B messages between eight IPI agents — the PMU, RPU, APU, and PL processors. Access to the interrupt registers and message buffers is protected by the XPPU to give exclusive access to the AXI transactions of the agents.

In a typical situation, the sender writes a 32-byte request message and generates an interrupt to the receiver. The receiver can write a response message and clear the interrupt that is observed by the sender. The communications process uses both the IPI interrupt structure and the message buffers. There are eleven interrupt channels and eight sets of message buffers.

- The interrupt channels are as follows.
    - Seven interrupts default to APU MPCore, RPU0, RPU1, and PL {0:3}, but can be reprogrammed to any processor because they are distributed to all four system interrupt controllers.
    - Four interrupts are hardwired to the PMU interrupt controller, IPI channels {3:6}.

- Message buffers provide exclusive communications between each sender and each receiver.
    - Seven sets of assignable message buffers.
    - One set of buffers dedicated to the PMU.
    - Each set has eight request and eight response buffers (16 buffers per set, 128 total buffers).

The PMU special considerations are as follows.

- Four sets of IPI interrupt registers for one processor.
- The PMU IPI 0 interrupt instructs the PMU to enter sleep mode.
- One set of message buffers are used for all four PMU interrupts.

The IPI interrupts and message buffers are independent hardware functions that are associated by software programming. There are default owners and an implied association between the interrupt registers and message buffers. Only the PMU interrupts are fixed in hardware.

The sender can post multiple interrupt requests and have different communication protocols with each target. The assignment and use of the non-PMU interrupts and the entire message passing architecture can be programmed as needed by the system architecture. The reset default conditions and software conventions in the SDK define a starting state for the system.

Interrupt Architecture

The interrupt architecture includes eleven 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). Access to each set of interrupt registers is protected by eight of the 64 KB apertures in the XPPU. Only eight apertures are needed because the four PMU interrupt registers are all within one 64 KB address space.

To send an interrupt, the sender writes a 1 to the bit in its trigger register that corresponds to the receiving master. The receiver sees the interrupt in its status register, ISR, in the bit field that corresponds to the sender. The sender can observe the state of the interrupts that it triggered to the receivers using its observation register (OBS). The receiver agent processes interrupts in a normal manner. The registers and signal routings are shown in Figure 13-4.

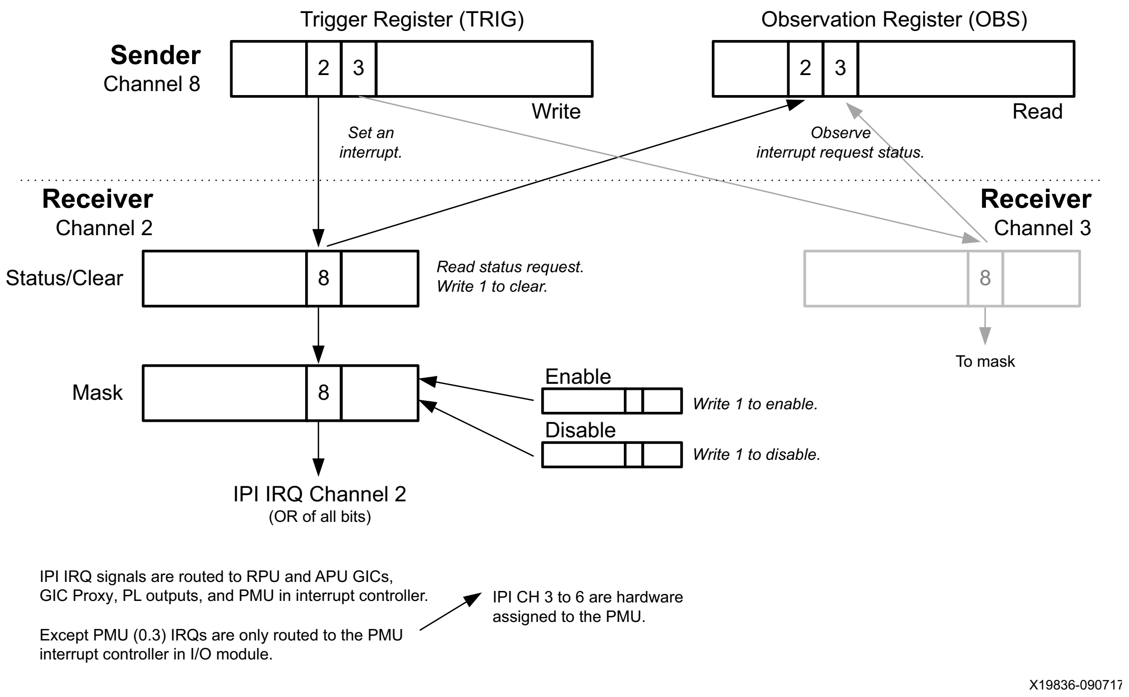

Trigger Register (TRIG)
Observation Register (OBS)
Sender
Channel 8
2
3
Write
Read
Set an interrupt.
Observe interrupt request status.
Receiver
Channel 2
Receiver
Channel 3
Status/Clear
8
Read status request.
Write 1 to clear.
To mask
Mask
Enable
Write 1 to enable.
Disable
Write 1 to disable.
IPI IRQ Channel 2
(OR of all bits)
IPI IRQ signals are routed to RPU and APU GICs, GIC Proxy, PL outputs, and PMU in interrupt controller.
Except PMU (0.3) IRQs are only routed to the PMU interrupt controller in I/O module.
IPI CH 3 to 6 are hardware assigned to the PMU.
X19836-090717

Figure 13-4: **Sender-Receiver Interrupt Functions**

Interrupt Register Descriptions

Each processor is assigned to a set of six IPI registers divided into sending and receiving interrupts. The IPI interrupt register functionality is provided in Table 13-2.

Table 13-2: **IPI Interrupt Register Functionality**

Channel Activity	Register Name	Acronym	Bit Writes		Bit Reads	
			Write a 1	**Write a 0**	**Read a 1**	**Read a 0**
Send interrupt	Trigger	TRIG	Assert interrupt	Ignored	Write only	
	Observation	OBS	Read only		Interrupt request asserted.	Interrupt request not asserted.
Receive interrupt	Status and clear	ISR	Clear bit	Ignored	Interrupt request asserted.	Interrupt request not asserted.
	Mask	IMR	Read only		IRQ not generated if status bit is asserted.	IRQ generated if status bit is asserted.
	Mask enable	IER	Set IMR = 1	Ignored	Write only	
	Mask disable	IDR	Set IMR = 0	Ignored	Write only	

Interrupt Register Channels

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 eleven sets of interrupt registers for use by any processor, except IPI channels {3:6}, which are hardwired for the PMU. The default and hardwired channel assignments are shown in Figure 13-5. Default channel assignments are defined in the Xilinx software and supported by the master IDs configured in the XPPU after reset.

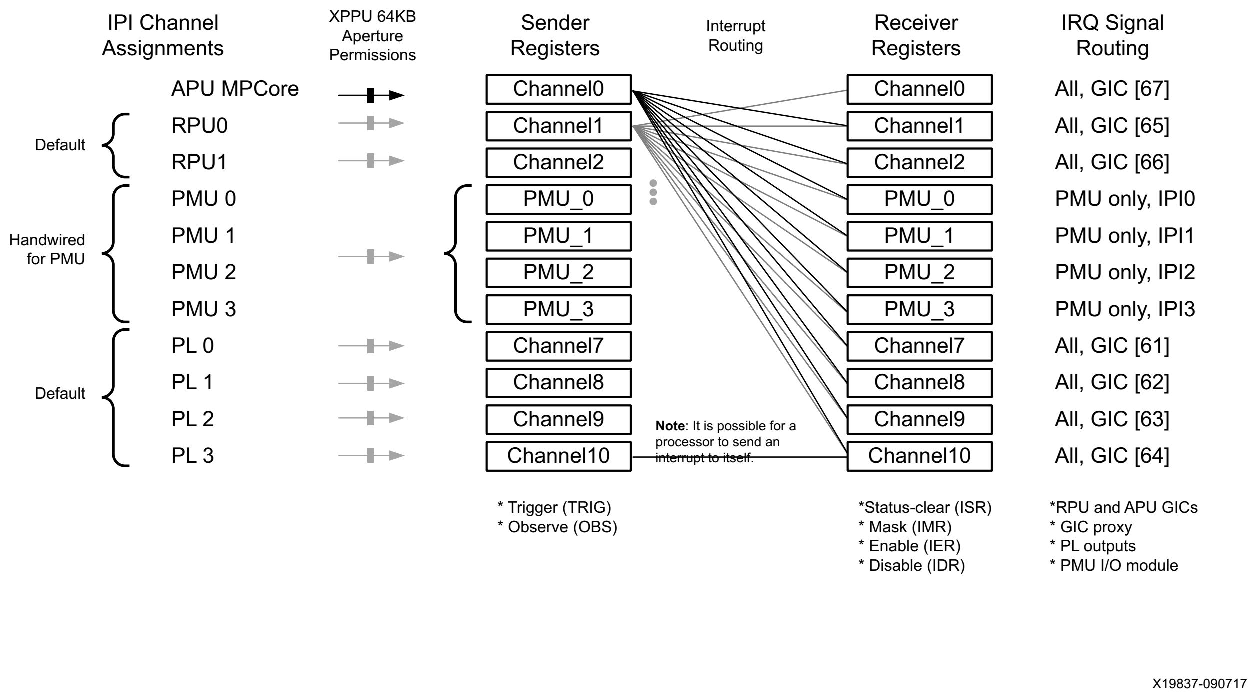


Figure 13-5: **IPI Interrupt Channel Architecture**

The non-PMU IRQ system interrupts are bused to four places, as follows.

Note: It is the responsibility of the individual masters to mask any unwanted IPIs in their own GIC.

- RPU GIC uses the GICv1.0 architecture and is controlled by the RPU.
- APU GIC uses the GICv2.0 architecture and is controlled by the APU.
- GIC proxy is a Xilinx architecture for the PMU external interrupt controller and is controlled by the PMU.
- PL outputs include four signals from the PS to the PL.

The PMU IRQ signals are only routed to the PMU.

Message Passing Architecture

The messaging system connects eight agents together in a mesh configuration. The PL is represented by four agents, and the PMU is one agent. The message passing between agents can be done exclusively between the sender and receiver using all 128 of the 32B permission apertures in the XPPU.

To support message passing, the software in the two processors must pre-define the format of the request and response message buffers. The buffer content does not affect the hardware. The use of a message buffer is optional. Figure 13-6 shows the IPI message passing architecture.

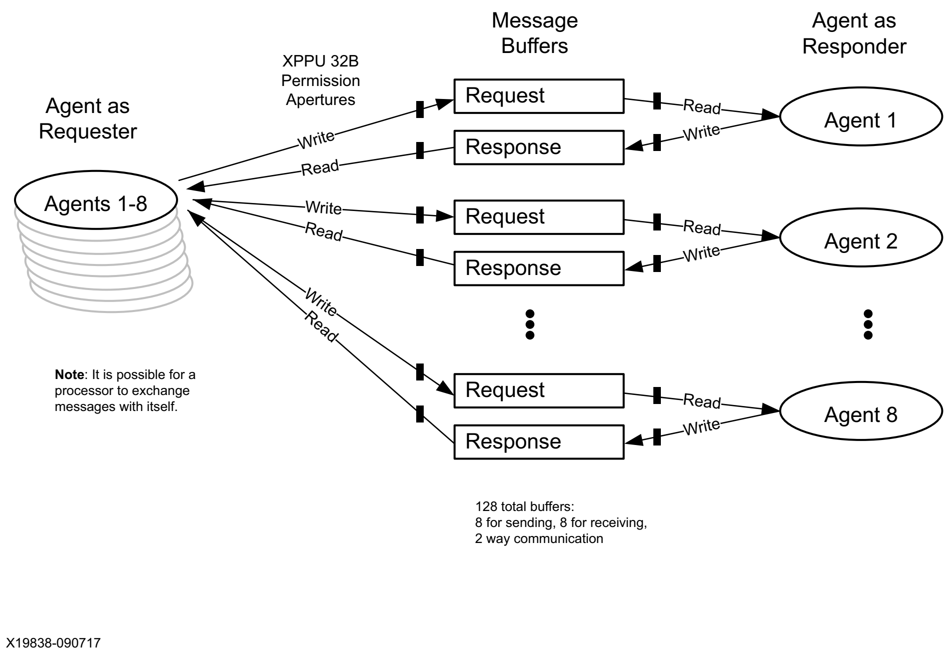


Figure 13-6: **IPI Message Passing Architecture**

Register and Buffer Summary

The IPI interrupt channels and message buffers are pre-defined and software associated as described in Table 13-3.

Note: The software might reassign the interr/upt channels and message buffers except for the PMU interrupts.

Table 13-3: **IPI Channel and Message Buffer Default Associations**

Channel Number	Default Owner	IPI Interrupt Registers			IPI Message Buffers		
		Name	Base Address	XPPU 64 KB Aperture	SI Agent Number	Base Address	XPPU 32B Apertures
Channel 0	APU	Channel0	0xFF30_0000	048	1	0xFF99_0400	256 - 271
Channel 1	RPU0	Channel1	0xFF31_0000	049	2	0xFF99-0000	272 - 287
Channel 2	RPU1	Channel2	0xFF32_0000	050	3	0xFF99_0200	288 - 303
Channel 3	PMU[1]	PMU_0[2]	0xFF33_0000	051	8	0xFF99_0E00	368 - 383
Channel 4		PMU_1	0xFF33_1000				
Channel 5		PMU_2	0xFF33_2000				
Channel 6		PMU_3	0xFF33_3000				
Channel 7	PL 0	Channel7	0xFF34_0000	052	4	0xFF99_0600	304 - 319
Channel 8	PL 1	Channel8	0xFF35_0000	053	5	0xFF99_0800	320 - 335
Channel 9	PL 2	Channel9	0xFF36_0000	054	6	0xFF99_0A00	336 - 351
Channel 10	PL 3	Channel10	0xFF37_0000	055	7	0xFF99_0C00	352 - 367

Notes:

1. The PMU interrupts are hardwired because the PMU IRQ signals only go to the PMU interrupt.
2. The PMI IPI0 interrupt causes the PMU to enter sleep mode.

Programming

The communication channels between processors must be coordinated with an agreed upon protocol and message format.

Generate an Interrupt

To generate an interrupt, the sender writes a 1 to a bit in its trigger (TRIG) register that corresponds to the target receiver. It can verify that a bit is set in the receiver's status register by reading its own OBS register. However, it cannot determine if the interrupt is enabled to generate the IRQ interrupt signal.

Determine the Source of Interrupt

A processing unit reads its interrupt status (ISR) and mask (IMR) registers to determine the source that caused the IRQ interrupt. Once serviced, the ISR can be cleared by writing the data that was read from this register. The bits that were set are cleared while preserving any bits that got set after the read took place, which helps to eliminate missed interrupts.

Send an IPI Communication

This section describes how to send an IPI communication.

1. Write a 32B request into the appropriate message buffer.
2. Write a 1 in the target receiver bit of its interrupt trigger register.
3. Optionally, verify that the interrupt is posted by reading its observation register.
4. Determine that the interrupt has been processed with *one* of the following steps.
    a. Poll the observation register until the status bit is cleared indicating that the receiver has processed the interrupt.
    b. Receive an IPI interrupt from the sender.

The method to indicate when an interrupt has been processed must be pre-arranged between the sender and receiver. The format of the message buffers must also be pre-arranged.

Receive an IPI Communication

This section describes how to receive an IPI communication.

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 interrupt controller by accessing GIC registers.
    b. Poll the status register for bits being set.
2. When an interrupt is received, optionally write a 32B response into the appropriate message buffer.
3. Signal to the sender that the interrupt has been processed with *one* of the following steps.
    a. Clear the status register.
    b. Issue an IPI interrupt back to the sender.

Interrupt Registers

There are several sets of interrupt registers and IPI message buffers that are memory mapped.

- RPU GIC: Arm PL390 with GICv1 interrupt architecture.
- APU GIC: Arm GIC400 with GICv2 interrupt architecture.
- GIC proxy system interrupt controller and Xilinx PMU interrupt architecture.
- IPI interrupts and Xilinx processor communications architecture.
- IPI message buffers, 32B x 128 buffers starting at address 0xFF99_0000.

The interrupt register sets are summarized in Table 13-5.

GIC Proxy Interrupts

The GIC proxy interrupts are used by the PMU when the RPU and APU cannot service an interrupt because the processor is powered-down.

The GIC proxy interrupts are listed in Table 13-4 and are controlled by the five sets of interrupt registers in the LPD_GIC_PROXY register set: GICP{0:4}_IRQ_{TRIGGER, STATUS, MASK, ENABLE, DISABLE}. The mask register bits are applied to the status register bits.

Interrupt Status Register

The bits in the GIC proxy status registers GICP{0:4}_IRQ_STATUS are sticky and remain asserted after the source of the interrupt has deasserted its signal. The minimum interrupt pulse width for detection is four clock periods of the GIC proxy unit, which is normally a 100 MHz clock resulting in a minimum 40 ns pulse width. A shorter pulse width might also be detected. The status register bits are cleared by writing a 1 to them. The status register shows the interrupt state before the mask is applied. This register can be polled to determine if the event occurred or did not occur, irrespective of the state of the associated mask bit. Software acknowledges the interrupt by clearing this register.

Interrupt Mask Register (IMR_REG)

The mask register is read-only. When a bit reads as a 1, it means an active interrupt from the status register is masked and it does not propagate to the GICP_PMU_IRQ_STATUS register. The default (reset) state is 1, implying all interrupts are masked.

Interrupt Enable and Interrupt Disable Registers

There are separate write-only registers for enabling (GICPx_IRQ_ENABLE) and disabling (GICPx_IRQ_DISABLE) a particular interrupt. This allows enabling/disabling on any single interrupt without the need for a read-modify-write register operation.

Interrupts to PMU

The state of the GIC proxy interrupts after the interrupt mask are OR'ed together on a per register basis to set bits in the LPD_SLCR.GICP_PMU_IRQ_STATUS register. For example, if any unmasked interrupt in the LPD_GIC_PROXY.GICP0_IRQ_STATUS register is active, then the LPD_SLCR.GICP_PMU_IRQ_STATUS [src0] bit is set by the interrupt hardware.

The PMU can read the GICP_PMU_IRQ_{STATUS, MASK} registers to determine which GIC proxy register allowed the interrupt to propagate. Finally, the GIC proxy status and mask registers that were determined to propagate the interrupt can be read to determine which system element caused the interrupt.

CPU Private Peripheral Interrupts

The functionality of the RPU PPIs are described by the GICv1 architecture specification. This is a subset of the APU PPI functionality that is described by the GICv2 specification.

Each CPU connects to a private set of peripheral interrupts. The list for the RPU is a subset of the APU. The sensitivity type (edge or level) for PPIs are fixed and cannot be changed.

RPU Private Interrupts

The ICDICFR1 register is read-only, since it specifies the sensitivity types of all five PPIs.

The fast interrupt (FIQ) signal and the interrupt (IRQ) signal from the PL are inverted and then sent to the interrupt controller. Consequently, they are active High at the PS-PL interface, although the ICDICFR1 register reflects them as active Low level.

APU Private Interrupts

Each APU core has a private set of peripheral interrupts routed from the CPU itself and the PL. They are listed in Table 13-4.

Table 13-4: **APU Private Peripheral Interrupts**

Name	Interrupt ID	Description
Virtual maintenance interrupt	25	Configurable event generated by virtual CPU interface to indicate a situation that might require hypervisor action.
Hypervisor timer	26	Physical timer event in hypervisor mode, PPI5 (CNTHP IRQ).
Virtual timer	27	Virtual timer generated event, PPI4 (CNTV IRQ).
Legacy FIQ signal	28	FIQ signal from the PL.
Secure physical timer	29	Secure physical timer event, PPI1 (CNTPS IRQ).
Non-secure physical timer	30	Non-secure physical timer event, PPI2 (CNTPNS IRQ).
Legacy IRQ signal	31	IRQ signal from the PL.

GIC Address Map

The APU GIC's base address is configured by the APU MPCore pins (PERIPHBASE). The RPU GIC's base address is aligned to the Cortex-R5F MPCore's low-latency peripheral port (LLPP) base-address.

The GIC-400 uses eight pages of 4 KB memory-mapped address-space. However, to support a 64 KB page size (as required by SBSA v2), the GIC-400 address needs to be mapped such that pages are 64 KB. For this, the AXI address is mapped to a GIC slave interface as described in this equation.

AddressGIC400[14:0] = {AddressAXI[18:16], AddressAXI[11:0]}

Register Overview

There are several interrupt register sets as shown in Table 13-5.

Table 13-5: **Interrupt Register Overview**

Starting Address	Register Set	Count	Description
RPU - Private CPU Bus for RPU MPCore			
0xF900_0000	PL390.enable	1	Interrupt control register (ICDICR).
0xF900_0080	PL390.sgi_security_if_n	1	SGI interrupt security register (ICDISR).
0xF900_0084	PL390.spi_security	5	SPI interrupt security register (ICDISR).
0xF900_0104	PL390.spi_enable_set	5	SPI enable set register (ICDISER).
0xF900_0184	PL390.spi_enable_clr	5	SPI interrupt clear-enable registers (ICDICER).
0xF900_0200	PL390.sgi_pending_set_if_n	1	SGI interrupt set-pending registers (ICDISPR).
0xF900_0204	PL390.spi_pending_set	5	SPI interrupt set-pending registers (ICDISPR).
0xF900_0280	PL390.sgi_pending_clr_if_n	1	SGI pending clear register (ICDICPR).
0xF900_0284	PL390.spi_pending_clr	5	SPI pending clear register (ICDICPR).
0xF900_0300	PL390.sgi_active_if_n	1	SGI active bit registers (ICDABR).
0xF900_0304	PL390.spi_active	5	SPI active bit registers (ICDABR).
0xF900_0400	PL390.priority_sgi_if_n	16	SGI interrupt priority registers (ICDIPR).
0xF900_0420	PL390.priority_spi	160	SPI interrupt priority registers (ICDIPR).
0xF900_0820	PL390.targets_spi	160	SPI target register interrupt (ICDIPTR).
0xF900_0C08	PL390.spi_config	5	SPI interrupt configuration register Interrupt (ICDICR).
APU - AXI Interconnect with Access Restricted to APU MPCore			
0xF901_0000	GIC400.GICD	180	Display controller.
0xF902_0000	GIC400.GICC	15	CPU interface.
0xF904_0000	GIC400.GICH	99	Hypervisor.
0xF906_0000	GIC400.GICV	14	Virtual machine.
GIC Proxy - LPD Slave			
0xFF41_8000	LPD_GIC_PROXY.GICP{0:5}_ {STATUS, MASK, ENABLE, DISPLAY, TRIGGER}	30	System interrupt control registers.
0xFF41_80A0	LPD_GIC_PROXY.GICP_IRQ_ {STATUS, MASK, ENABLE, DISPLAY, TRIGGER}	5	OR'ed interrupts control registers.

Table 13-5: **Interrupt Register Overview *(Cont'd)***

Starting Address	Register Set	Count	Description
IPI - LPD Slave			

0xFF30_0000	IPI.CH0_{TRIG, OBS, ISR, IMR, IER, IDR}
	IPI.CH1_{TRIG, OBS, ISR, IMR, IER, IDR}
	IPI.CH2_{TRIG, OBS, ISR, IMR, IER, IDR}
	IPI.PMU_0_{TRIG, OBS, ISR, IMR, IER, IDR}
	IPI.PMU_1_{TRIG, OBS, ISR, IMR, IER, IDR}
	IPI.PMU_2_{TRIG, OBS, ISR, IMR, IER, IDR}
	IPI.PMU_3_{TRIG, OBS, ISR, IMR, IER, IDR}
	IPI.CH7_{TRIG, OBS, ISR, IMR, IER, IDR}
	IPI.CH8_{TRIG, OBS, ISR, IMR, IER, IDR}
	IPI.CH9_{TRIG, OBS, ISR, IMR, IER, IDR}
	IPI.CH10_{TRIG, OBS, ISR, IMR, IER, IDR}
		60	Inter-processor interrupts:
			trigger, observation, status and clear, mask, mask enable, mask disable.

	IPI_CTRL	1	SLVERR bus error enable control.
	IPI_{ISR, IMR, IER, IDR}	4	SLVERR interrupt status, mask, mask enable/disable.
0xFF38_0030	SAFETY_CHK	1	Safety endpoint connectivity check register, no effect on IPI operations.

Programming Examples

- For Programming of the GICV1, refer to the Arm® Generic Interrupt Controller Architecture version 1.0.
- For Programming of the GICV2, refer to the Arm® Generic Interrupt Controller Architecture version 2.0.

Clearing Pending Interrupts from the APU GICv2

The GICv2 gets reset based on reset of the interconnect and does not have a soft reset bit. These steps ensure that all pending interrupts are cleared after the CPU comes back up from a reset:

1. Write the value FFFF_FFFFh into the GICD_ICENABLERx register.
2. Write the value FFFF_FFFFh into the GICD_ICPENDRx   register.
3. Write the value FFFF_FFFFh into the GICD_ICACTIVERx register.
4. Write the value FFFF_FFFFh into the GICD_CPENDSGIRx register.

The following is sample FSBL code for clearing the pending interrupts from the APU GICv2.

```
XFsbl_Printf (DEBUG_GENERAL, "Clear pending interrupts from APU GIC\n\r");
for (i = 0; i < 6; i++) {
      Xil_Out32 (GICD_BASEADDR + 0x180 + 4*i, 0xffffffff); // GICD_ICENABLERx (x= 0 to 5)
      Xil_Out32 (GICD_BASEADDR + 0x280 + 4*i, 0xffffffff); // GICD_ICPENDRx (x= 0 to 5)
      Xil_Out32 (GICD_BASEADDR + 0x380 + 4*i, 0xffffffff); // GICD_ICACTIVERx (x= 0 to 5)
}

  for (i = 0; i < 4; i++) {
      Xil_Out32 (GICD_BASEADDR + 0xF10 + 4*i, 0xffffffff); // GICD_CPENDSGIRx (x= 0 to 3)
}
```

Programming Model IPI

This section describes programming the interrupts.

Example: Initiate an IPI

1. Initiator software writes a request message to memory.
2. Initiator writes a 1 to its Trigger register for the target processor.
3. Initiator may pole its Observation register or wait for an IPI response interrupt from the target.
4. After the target has indicated it has responded to the interrupt, the initiator can read the response message in memory. This is in a pre-defined format to a pre-defined memory location.

Example: Receive an IPI

1. The target must enable interrupts to receive an IRQ to its interrupt controller. This is done using the Mask register.
2. The target IRQ handler reads its status (ISR) and mask (IMR) registers to determine the identity of the initiator.
3. The target reads the Request Message written by the initiator. It processes it and provides a Response Message. The messages are in a pre-determined format in a pre-determined place in memory.
4. The target clears the IRQ in the IPI and optionally sends an IPI response to the target.

Enable the Interrupt

To enable the interrupt, write a `1` to the bit corresponding to the processing unit whose interrupt needs to be enabled in the *_IER register.

Disable the Interrupt

To disable the interrupt, write a `1` to the bit corresponding to the processing unit whose interrupt needs to be disabled in the *_IDR register.

Timers and Counters

Introduction

The PS has many different types of timers and counters.

- APU MPCore AArch64 timers:
    - APU MPCore global timer (system private).
    - APU core private timers (physical private, virtual private).
- Triple-timer counter:
    - Four triple-timer counter (TCC) units in the LPD.
- System watchdog timers:
    - FPD_SWDT: system watchdog timer on the FPD interconnect (swdt1).
    - LPD_SWDT: system watchdog timer on the LPD interconnect (swdt0).
    - CSU_SWDT: system watchdog timer on the CSU/PMU interconnect.

System Block Diagram

Figure 14-1 shows the system timers in the PS.

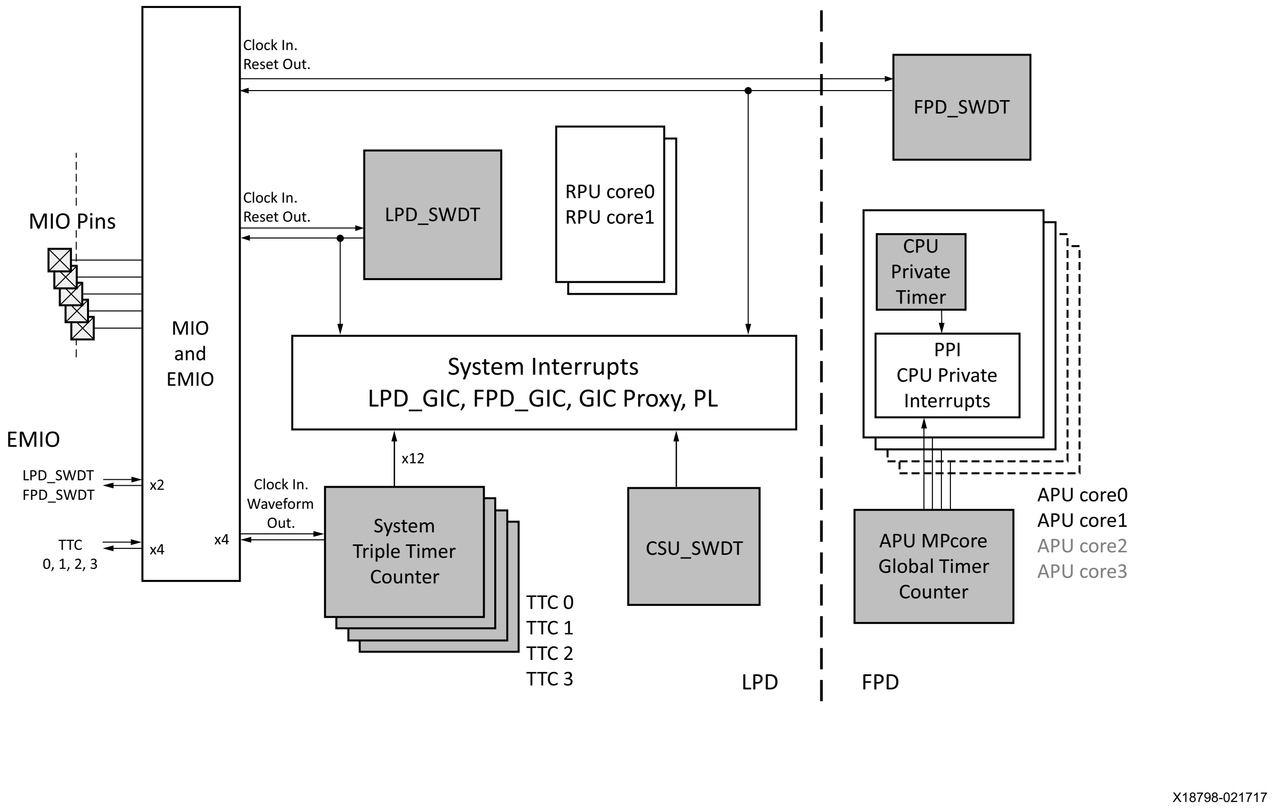


Figure 14-1: **Timers System Block Diagram**

APU MPCore System Counter

The system timer is documented in the Cortex-A53 MPCore Technical Reference Manual. This counter is sometimes referred to as the global counter. The counter is controlled by IOU_SCNTRS register set. The clock controlled by CRL_APB.TIMESTAMP_REF_CTRL register; Vivado PCW [TIMESTAMP] setting.

Features

- 64-bit counter is private to the APU MPCore.
- Auto-incrementing feature.
- 64-bit comparator can assert a private interrupt.

Software can access the CNTFRQ register to read or modify the clock frequency of the system counter. Each Cortex-A53 MPCore has a counter input that can capture each increment of the system counter.

Typically, initializing and reading the system counter frequency includes setting the system counter frequency using the system control 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.

Software can read the CNTFRQ register to determine the current system counter frequency in these states and modes.

- Non-secure EL2 mode.
- Secure and non-secure EL1 modes.
- When CNTKCTL.EL0PCTEN is set to 1, secure and non-secure EL0 modes.

Applications

Event Streams

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.

An event stream is configured by these selections.

- Selecting which bit from the bottom 32-bits of a counter triggers the event. This 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.

Programming

Generic Timer Programming

Memory-mapped controls of the system counter are accessible only through the memory-mapped interface to the system counter.

These controls are listed.

- Enabling and disabling the counter.
  CNTCR, 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 such accesses.
- Changing the operating mode to change the update frequency and increment value.
  CNTCR, counter control register FCREQ, bits [17:8]: 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.

Register Overview

The MPCore timers are defined by the AArch64 architecture specification. Table 14-1 provides an overview of the AArch32 registers.

Table 14-1: **AArch32 Register Overview**

Function	Control Register
Timer frequency.	CNTFRQ
Kernel control.	CNTKCTL
Hypervisor control.	CNTHCTL
Virtual offset.	CNTVOFF

Register Access

The system counter control and status registers are accessible to all APU cores using their CPU private register space.

APU Core Private Physical and Virtual Timers

The system timer is documented in the Cortex-R5F or Cortex-A53 MPCore TRMs [Ref 46] [Ref 48]. The clock is controlled by the CRL_APB.DBG_TSTMP_CTRL register Vivado PCW [DBG_TSTMP] setting.

System Timer

The System timer can be exclusively configured for A53 and R5F by reading and writing to IOU_SCNTR and IOU_SCNTRS registers residing in the LPD_IOU domain. These registers can be accessed by any master. Both registers IOU_SCNTR and IOU_SCNTRS map to the same physical timer (IOU_SCNTR is read-only). The clock is controlled by the CRL_APB.DBG_TSTMP_CTRL register Vivado PCW [DBG_TSTMP] setting.

For more information refer to the Cortex-R5F or Cortex-A53 MPCore TRMs [Ref 46] [Ref 47].

Table 14-2: **System Timer Registers**

Register Name	Address	Access Type	Description
IOU_SCNTR	0xFF250000	Read/Write	System Timestamp Generator
IOU_SCNTRS	0xFF260000	Read/Write	System Timestamp Generator- Secure

Features

- 64-bit counter is private to each CPU core.

- Same PPI interrupt number for each APU core.
- Extensions to the timer to AArch64:
  - Non-secure EL1 physical timer.
  - Secure E1 physical timer.
  - Non-secure EL2 physical timer.
  - Virtual timer.

Physical Timer

Physical Counter

Each APU core includes a physical counter that contains the count value of the system counter. 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. The timer is clocked at ½ the APU clock frequency. This logic generates a tick after N clock pulses, where N is defined as:

N = (½ APU clock frequency)/100 MHz.

100 MHz is a configurable clock that goes to the TSGEN module. TSGEN is the timestamp generator in the Coresight™ debug module in the APU and runs between 200 MHz and 400 MHz. 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 DBG_TSTMP_CTRL register.

Accessing the Physical Counter

Software with sufficient privilege can read CNTPCT using a 64-bit system control register read.

Virtual Timer

Virtual Counter

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 control register read.

Register Access

Accessing the Timer Registers

For each timer, all timer registers have the same access permissions.

EL1 Physical Timer

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 following describes the EL1 physical timer.

- 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.

Virtual Timer

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.

EL2 Physical Timer

Accessible from non-secure hypervisor mode, and from the secure monitor mode when SCR_EL3.NS is set to 1.

Register Overview

The following table provides an overview of the APU core private timers.

Table 14-3: **APU Core Private Timers (AArch64)**

Function	Physical Timer	Virtual Timer	Physical Secure Timer	Hypervisor Physical Timer
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		

Triple-timer Counters

The four triple-timer counter (TTC) units are located in the LPD region and each unit has three similar counters. The TTCs can generate periodic interrupts or can be used to count the widths of signal pulses from an MIO pin or from the PL. All three counters must have the same security status because they share a single APB bus.

TTC Counter Features

- 32-bit APB programming interface.
- A selectable clock input.
  - Internal PS bus clock (LPD_LSBUS_CLK)
  - Internal clock (from PL)
  - External clock (from MIO)
- Support for three independent 32-bit timer/counters.
- Support for a 16-bit prescaler for the clock.
- Three system interrupts, one for each timer counter.
- Interrupt on overflow, counter match programmable values.
- Increment and decrement counting.
- Generates a waveform output (for example, PWM) through the MIO and to the PL.

TTC Block Diagram

Figure 14-2 is a block diagram of the TTC. The clock-in and wave-out multiplexing for the timer/clock 0 is controlled by the slcr.MIO_PIN_xx registers. If no selection is made in these registers, then the default becomes the EMIO interface.

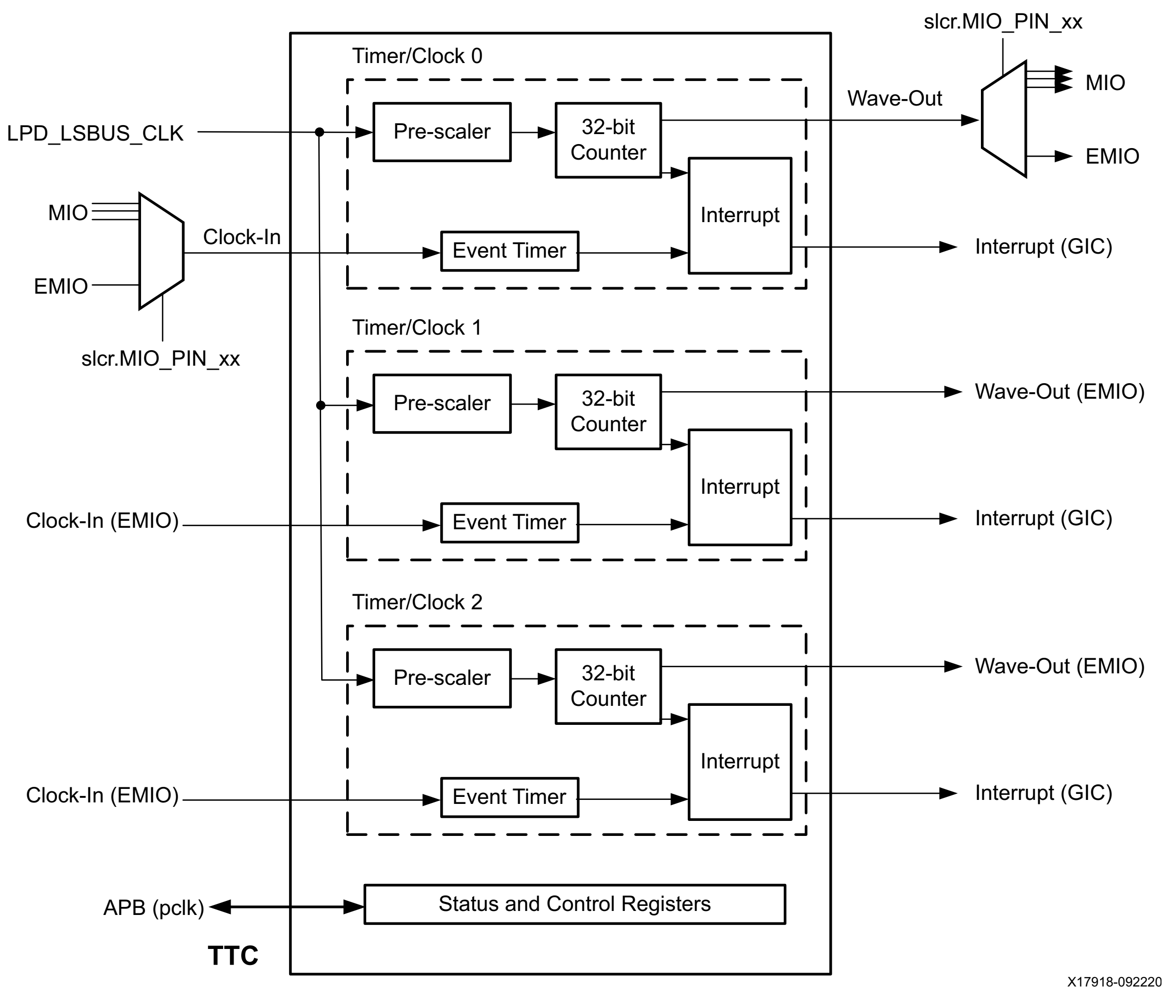


Figure 14-2: **TTC Block Diagram**

TTC Functional Description

Each prescaler module can be independently programmed to use the LPD_LSBUS_CLK, or an external clock from the MIO or the PL. For an external clock, the SLCR registers determine the exact pinout via the MIO or from the PL. The selected clock is then divided down from two to `0xFFFF_FFFF` before being applied to the counter. 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.

The interrupt module combines interrupts of various types: counter interval, counter matches, counter overflow, and event timer overflow. Each type can be individually enabled.

Initialization

On initialization, the counters are set to these configurations.

- Overflow mode.
- Internal clock selected.
- Counter disabled.
- All interrupts disabled.
- Event timer disabled.
- Output waveforms disabled.

Prescaler

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 system clock or an external clock (ext_clk). The selected clock is then divided down to provide the count clock; division can be from ÷2 to ÷65536.

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.

Interrupt Module

Three interrupt signals are available for use at the system level, one from each timer counter. An interrupt occurs when a bit in the interrupt enable register and the corresponding bit in the interrupt detect register are both set. The resulting ANDed outputs are then ORed to generate the system interrupt signal. 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.

Modes of Operation

Each of the timer counter modules can operate in one of four modes, and register matching can also be programmed for each of these modes.

- Interval timing, increment count.
- Interval timing, decrement count.
- Overflow detection, increment count.
- Overflow detection, decrement count.

Interval 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.

Overflow 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 `0xFFFF_FFFF` it overflows to zero, 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 `0xFFFF_FFFF` and counting down is restarted.

Event Control Timer Operation

The event control timer operates by having an internal 32-bit counter clocked by the LPD_LSBUS_CLK 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 register (TTC.Event_Control_Timer_{0:3}) controls 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).
    - When `0`: Overflow causes [E_En] to be `0` (see the [E_En] bit description).
    - When `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 LPD_LSBUS_CLK. If overflow occurs, the event register is not updated and maintains the old value.

Register Overview

Table 14-1 provides an overview of the AArch32 register. Table 14-14 lists the system watchdog timer registers and Table 14-5 lists the TTC registers.

Table 14-4: **Watchdog Timers**

Watchdog Timers	LPD_SWDT	FPD_SWDT	CSU_SWDT
Clock select	IOU_SLCR.WDT_CLK_SEL [SELECT]	FPD_SLCR.WDT_CLK_SEL [SELECT]	LPD_SLCR.CSUPMU_WDT_CLK_SEL [SELECT]
Reset input	RST_CTRL_LPD.RST_LPD_TOP [lpd_swdt_reset]	RST_CTRL_FPD.RST_FPD_TOP [swdt_reset]	RST_CTRL_LPD.RST_LPD_IOU2 [swdt_reset]
Mode select	SWDT.MODE [WDEN], [RSTEN], [IRQEN], [RSTLN], [IRQLN], [ZKEY]	WDT.MODE [WDEN], [RSTEN], [IRQEN], [RSTLN], [IRQLN], [ZKEY]	CSU_WDT.MODE [WDEN], [RSTEN], [IRQEN], [RSTLN], [IRQLN], [ZKEY]
Control	SWDT.CONTROL [CLKSEL], [CRV], [CKEY]	WDT.CONTROL [CLKSEL], [CRV], [CKEY]	CSU_WDT.CONTROL [CLKSEL], [CRV], [CKEY]
Restart	SWDT.RESTART [RSTKEY]	WDT.RESTART [RSTKEY]	CSU_WDT.RESTART [RSTKEY]
Status	SWDT.STATUS [WDZ]	WDT.STATUS [WDZ]	CSU_WDT.STATUS [WDZ]
Reset output on MIO pins	IOU_SLCR.MIO_PIN_xx	IOU_SLCR.MIO_PIN_xx	~
System error status	PMU_GLOBAL.ERROR_STATUS_1 [LPD_SWDT]	PMU_GLOBAL.ERROR_STATUS_1 [FPD_SWDT]	~
GIC proxy interrupt status	LPD_GIC_PROXY.GICP1_IRQ_STATUS [20]	LPD_GIC_PROXY.GICP3_IRQ_STATUS [17]	LPD_GIC_PROXY.GICP1_IRQ_STATUS [21]

Table 14-5: **TTC Registers**

Name	Description
Clock_Control_{1:3}	Clock control register.
Counter_Control_{1:3}	Operational mode and reset.
Counter_Value_{1:3}	Current counter value.
Interval_Counter_{1:3}	Interval value.
Match_{1:3}_Counter_{1:3}	Match value.
Interrupt_Register_{1:3}	Counter 1 to 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 Programming Examples

- Initialization
- Set options
- Prescalar
- setup timer
- Setup ticker
- Stop timer

TTC Programming

The programming steps for the TTC are listed in Table 14-6 through Table 14-11. Figure 14-4 shows the TTC flowchart.

Table 14-6: **TTC Initialization**

Task	Register	Register Field	Register Offset	Bits	Value
Check if timer counter is started	counter_control	DIS	0x0C	0	Read operation
Ensure timer counter has not started					
Write reset value to the counter control register	counter_control	All	0x0C	31:0	0x21 (hex)
Reset clock control	clock_control	All	0x00	31:0	0x00 (hex)
Reset interval count value	interval_counter	All	0x24	31:0	0x00 (hex)
Reset match-1 value	match_1_counter	All	0x3C	31:0	0x00 (hex)
Reset match-2 value	match_2_counter	All	0x48	31:0	0x00 (hex)
Reset IER	interrupt_enable	All	0x60	31:0	0x00 (hex)
Reset ISR	interrupt_register	All	0x54	31:0	0x00 (hex)
Reset counter	counter_control	RST	0x0C	4	1b'1

Table 14-7: **TTC Set Options**

Task	Register	Register Field	Register Offset	Bits	Value
External clock set option	clock_control	C_Src	0x00	5	1b'1
External clock deselect option	clock_control	C_Src	0x00	5	1b'0
Negative edge clock selection	clock_control	Ex_E	0x00	6	1b'1
Negative edge clock deselect	clock_control	Ex_E	0x00	6	1b'0
Interval mode select	counter_control	INT	0x0C	1	1b'1
Interval mode deselect	counter_control	INT	0x0C	1	1b'0
Decrement counter	counter_control	DEC	0x0C	2	1b'1
Decrement counter deselect	counter_control	DEC	0x0C	2	1b'0
Select match mode	counter_control	Match	0x0C	3	1b'1
Deselect match mode	counter_control	Match	0x0C	3	1b'0
Disable waveform output	counter_control	Wave_en	0x0C	5	1b'1
Enable waveform output	counter_control	Wave_en	0x0C	5	1b'0
Select waveform polarity	counter_control	Wave_pol	0x0C	6	1b'0
Select waveform polarity	counter_control	Wave_pol	0x0C	6	1b'0

Table 14-8: **TTC Set Prescaler**

Task	Register	Register Field	Register Offset	Bits	Value
Clear prescaler control bits	clock_control	PS_V | PS_En	`0x00`	4:0	`5b'00000`
Write the value only if prescaler value is less than 16	clock_control	PS_V | PS_En	`0x00`	4:0	Prescaler value to be written

Table 14-9: **Setup Timer**

Task	Register	Register Field	Register Offset	Bits	Value
Stop timer	counter_control	DIS	`0x0C`	0	`1b'1`
Initialize the device. Refer to TTC Initialization.					
Set required options. Refer to TTC Set Options.					
Calculate interval and prescaler.					
Setup interval	interval_counter	All	`0x24`	31:0	Interval value calculated in previous step
Set prescaler value calculated in previous step. Refer to TTC Set Prescaler.					

Table 14-10: **Setup Ticker**

Task	Register	Register Field	Register Offset	Bits	Value
Setup timer. Refer to Setup Timer.					
Register the ticker handler with the GIC.					
Enable TTC interrupts in the GIC.					
Enable interval interrupt	interval_counter	Interval	`0x60`	0	`1b'1`
Start timer	counter_control	DIS	`0x0C`	0	`1b'0`

Table 14-11: **TTC Stop Timer**

Task	Register	Register Field	Register Offset	Bits	Value
Stop timer	counter_control	DIS	`0x0C`	0	`1b'1`

System Watchdog Timers

There are three system watchdog timer (SWDT) units in the PS. They are all based on the Arm system watchdog timer architecture. One major difference between the timers is the system interface signals.

The clock source for the LPD and FPD watchdog timers can come from one of three sources. The CSU_SWDT can source its clock from either the local bus or directly from the PS_REF_CLK pin.

A watchdog timer is used to detect and recover from system malfunctions. The watchdog timer can be used to prevent system lockup; for example, when software becomes trapped in a deadlock. In normal operation, an interrupt handler running on a processor restarts the watchdog timer at regular intervals before the timer counts down to zero. In cases where the timer does reach zero and the watchdog is enabled, one or a combination of the following signals is generated: a system reset, an interrupt, or an external signal. The watchdog timeout period and the duration of any output signals are programmable.

There are three watchdog timers in the system. Each timer has the same programming model and similar control registers.

- LPD_SWDT: uses the SWDT register set and is sometimes referred to as swdt0.
- FPD_SWDT: uses the WDT register set and is sometimes referred to as swdt1.
- CSU_SWDT: uses the CSU_WDT register set.

The LPD watchdog timer, LPD_SWDT, protects the RPU MPCore and its interconnect. The FPD watchdog timer, FPD_SWDT, protects the APU MPCore and its interconnect. The third watchdog timer, CSU_SWDT, protects the CSU and PMU interconnects. It also includes a logic built-in self-test (LBIST) to promote operating safety.

The APU SWDT can be used to reset the APU or the FPD. The RPU SWDT can be used to reset the RPU or the LPD.

- An internal 24-bit counter.
- Variable timeout period, from 1 ms to 30 seconds using a 100 MHz clock.
- Programmable reset period.

Figure 14-3 is a block diagram of the watchdog timer.

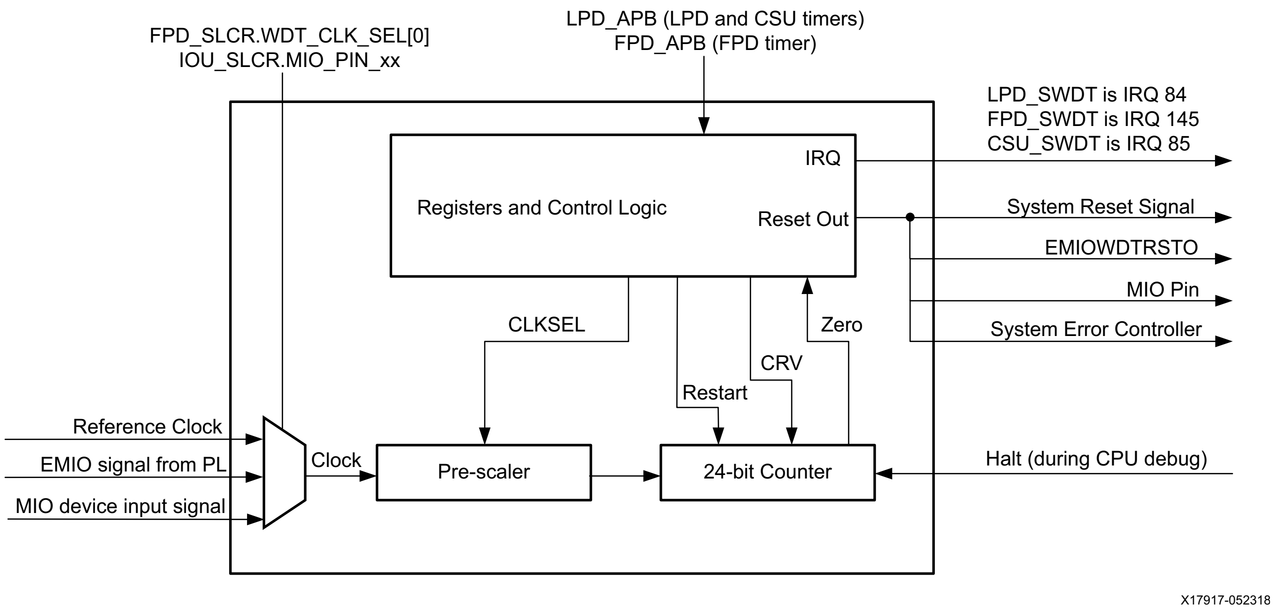


Figure 14-3: **SWDT Block Diagram**

Figure 14-3 notes:

- Clock selects: FPD_SLCR.WDT_CLK_SEL [select] and FPD_SLCR.WDT_CLK_SEL [select].
- MIO pin selects: IOU_SLCR.MIO_PIN_x registers.
- Program the clock prescaler and restart values: {SWDT, WDT, SU_WDT}.CONTROL [CLKSEL], [CRV].
- A restart signal causes the 24-bit counter to reload the [CRV] value and restart counting.
- A halt signal causes the counter to halt during CPU debug (same behavior as the APU SWDT).

  The halt conditions are as follows:

  - Clock selects: FPD_SLCR.WDT_CLK_SEL [select] and FPD_SLCR.WDT_CLK_SEL [select].
  - LPD system WDT– halted by RPU only; either core in the debug can halt it.
  - LPD CSU WDT– halted by RPU only; either core in the debug can halt it.
  - FPD System WDT– halted by APU only; any core in the debug can halt it.

SWDT Functional Description

The control logic block has an APB interface connected to the system interconnect. Writes to the MODE and CONTROL registers require a key. The mode register requires the zKEY and the controller register requires the cKEY.

- The zero mode register controls the behavior of the watchdog timer when its internal 24-bit counter reaches zero. Upon receiving a zero signal, the control logic block (if both mode bits [WDEN] and [IRQEN] are set) asserts the interrupt output signal for MODE IRQLN clock cycles, and (if [WDEN] is set) also asserts the reset output signals for approximately one clock cycle. The 24-bit counter then stays at zero until it is restarted.
- The counter control register sets the timeout period by setting reload values in CONTROL[CLKSET] and [CRV] bits to control the prescaler and the 24-bit counter.
- The restart register is used to restart the counting process. Writing to this register with a matched key causes the prescaler and the 24-bit counter to reload the values from the CRV signals.
- The status register shows whether the 24-bit counter reaches zero. Regardless of the [WDEN] bit in the zero mode register, the 24-bit counter keeps counting down to zero when it is not zero and the selected clock source is present. Once the 24-bit counter reaches zero, the [WDZ] bit of the status register is set and remains set until the 24-bit counter is restarted.
- The prescaler block divides down the selected clock input. The [CLKSEL] bit is sampled at every rising clock edge.
- The internal 24-bit counter counts down to zero and stays at zero until it is restarted. While the counter is at zero, the zero output signal is High.

Interrupt to RPU and APU GIC Interrupt Controllers

The pulse length of four clock cycles (SWDT.MODE[IRQLN] = `2'b00`) from the watchdog timer is sufficient for the interrupt controller to capture the interrupt using rising-edge sensitivity.

Watchdog Enabled on Reset

The purpose of this watchdog is to prevent system lockup if the software becomes trapped in a deadlock. The watchdog is therefore enabled on reset as the software lockup could occur immediately after the reset is removed.

CPU Debug

An input cpu_debug is provided by the SWDT. It is possible to stop the CPU and analyze the content of system register and memory. To enable diagnosis of system problems during prototype commissioning, connect the signal that stops the CPU to the SWDT input cpu_debug. This suspends the SWDT and it will not time out on a CPU that is stopped for diagnostic purposes.

SWDT I/O Control and Configuration Register Sets

The system watchdog timer configuration registers are listed in Table 14-12.

Table 14-12: **SWDT I/O Control and Configuration Register Sets**

	Name	Internal	External	MIO	EMIO	Register Control
Clock Input						
	LPD_SWDT	LPD_LSBUS_CLK	~	Yes	Yes	IOU_SLCR.WDT_CLK_SEL.
	FPD_SWDT	TOPSW_LSBUS_CLK	~	Yes	Yes	FPD_SLCR.WDT_CLK_SEL.
	CSU_SWDT		PS_REF_CLK	No	No	LPD_SLCR.CSUPMU_WDT_CLK_SEL.
Reset Output						
	LPD_SWDT	IRQ [84]	~ [1]	Yes	Yes	Always IRQ and EMIO. IOU_SLCR.MIO_PIN_xx.
	FPD_SWDT	IRQ [145]	~ [1]	Yes	Yes	Always IRQ and EMIO. IOU_SLCR.MIO_PIN_xx.
	CSU_SWDT	IRQ [85]	~ [2]	No	No	Always IRQ.
Configuration Register Sets						
	LPD_SWDT	~	~	~	~	SWDT register set.
	FPD_SWDT	~	~	~	~	WDT register set.
	CSU_SWDT	~	~	~	~	CSU_WDT register set.

Notes:
1. The LPD and FPD system watchdog timers can cause a system lockdown to affect the PS_ERROR_STATUS signal.

SWDT Register Overview

The registers for the three SWDT units are summarized in Table 14-13.

Table 14-13: **SWDT Register Overview**

	LPD_SWDT	FPD_SWDT	CSU_SWDT
Clock select	IOU_SLCR WDT_CLK_SEL [SELECT]	FPD_SLCR WDT_CLK_SEL [SELECT]	LPD_SLCR CSUPMU_WDT_CLK_SEL [SELECT]
Reset input	RST_CTRL_LPD RST_LPD_TOP [lpd_swdt_reset]	RST_CTRL_FPD RST_FPD_TOP [swdt_reset]	RST_CTRL_LPD RST_LPD_IOU2 [swdt_reset]
Mode select	SWDT.MODE [WDEN], [RSTEN], [IRQEN], [RSTLN], [IRQLN], [ZKEY]	WDT.MODE [WDEN], [RSTEN], [IRQEN], [RSTLN], [IRQLN], [ZKEY]	CSU_WDT.MODE [WDEN], [RSTEN], [IRQEN], [RSTLN], [IRQLN], [ZKEY]
Control	SWDT.CONTROL [CLKSEL], [CRV], [CKEY]	WDT.CONTROL [CLKSEL], [CRV], [CKEY]	CSU_WDT.CONTROL [CLKSEL], [CRV], [CKEY]
Restart	SWDT.RESTART [RSTKEY]	WDT.RESTART [RSTKEY]	CSU_WDT.RESTART [RSTKEY]
Status	SWDT.STATUS [WDZ]	WDT.STATUS [WDZ]	CSU_WDT.STATUS [WDZ]
Reset output on MIO pins	IOU_SLCR.MIO_PIN_xx	IOU_SLCR.MIO_PIN_xx	~
System error status	PMU_GLOBAL ERROR_STATUS_1 [LPD_SWDT]	PMU_GLOBAL ERROR_STATUS_1 [FPD_SWDT]	~
GIC proxy interrupt status	LPD_GIC_PROXY GICP1_IRQ_STATUS [20]	LPD_GIC_PROXY GICP3_IRQ_STATUS [17]	LPD_GIC_PROXY GICP1_IRQ_STATUS [21]

SWDT Register Overview

Table 14-14 is an overview of the SWDT registers, SWDT, WDT, and CSU_WDT register sets.

Table 14-14: **SWDT Register Overview**

Offset	Name	Access	Bits	Description
`0x00`	Watchdog zero mode register state on reset: `0x1C2`	Read/Write	0	WDEN: Watchdog enable. If set, the watchdog is enabled and can generate enabled signals.
			1	RSTEN: Reset enable. If set, the watchdog issues an internal reset when the counter reaches zero, if WDEN = `1`.
			2	IRQEN: Interrupt request enable. If set, the watchdog issues an interrupt request when the counter reaches zero, if WDEN = `1`.
			3	Reserved.
			6:4	RSTLN: Reset length, 2 to 256 PCLK cycles.
			8:7	IRQLN: Interrupt request length, 4 to 32 PCLK cycles.
			11:9	EXLN: External signal length, 8 to 2048 PCLK cycles.
		Write only	23:12	ZKEY: Zero access key. Writes to the zero mode register are only valid if this field is `0xABC`.
`0x04`	Counter control register state on reset: `0b111100`	Read/Write	1:0	CLKSEL: Counter clock prescale, from PCLK/8 to PCLK/4096.
			13:2	CRV: Counter restart value. The counter is restarted with `0xNFFF`, where N is the value of this field.
		Write only	25:14	CKEY: Counter access key. Writes to the control register are only valid if this field is `0x248`.
`0x08`	Restart register	Write only	15:0	RSTKEY: Restart key. The watchdog is restarted if this field is set to `0x1999`.
`0x0C`	Status register state on reset: `0x00`	Read only	0	WDZ: Watchdog zero. This bit is set when the counter reaches zero.

SWDT Programming Sequence

Programming Model

The watchdog timers are only reset by a power-on reset and not by a system reset. This ensures that the timer is not reset by its own reset output.

Enable Sequence

1. SWDT is reset by a power-on reset.
2. Disable the timer by clearing the WDEN bit. Write `AB_C000h` to the mode register. This disables the timer and sends the correct [ZKEY] bit field of `12'h0ABC`. The other bits can be 0 for now.
3. Initialize the counter control register. For example, writing `0x0923C` to the control register sets the divide by eight prescalar and the counter restart value to its maximum.The [CKEY] value in bits 25:14 must be `12'h0248`.
4. Enable the timer. For example, writing `0xABC1C5` to the mode register. Bit 0, [WDEN] enables the timer. Bit 1, [RSTEN] deasserts reset. Bit 2, [IRQEN] enables interrupts. Always write 0 to bit 3. Also, IRQLN and RSTLN must be greater than or equal to the specified minimum values.

SWDT Programming Examples

- Timer start/stop/restart
- Timer expiry
- Enable/disable signal output
- Set/get control values
- Self test
- Example

Watchdog Timer Programming

The programming steps for the watchdog timer are listed in Table 14-15 through Table 14-25. Table 14-26 lists an example of programming the watchdog timer interrupt.

Table 14-15: **Watchdog Timer Start**

Task	Register	Register Field	Register Offset	Bits	Value
Enable watchdog timer.	MODE	WDEN	`0x00`	0	`1`
Program zero access key.	MODE	ZKEY	`0x00`	23:12	`12'h0ABC`

Table 14-16: **Watchdog Timer Stop**

Task	Register	Register Field	Register Offset	Bits	Value
Disable watchdog timer.	MODE	WDEN	`0x00`	0	`0`
Program zero access key.	MODE	ZKEY	`0x00`	23:12	`12'h0ABC`

Table 14-17: **Watchdog Timer Restart**

Task	Register	Register Field	Register Offset	Bits	Value
Restart watchdog timer.	RESTART	RSTKEY	`0x08`	31:0	`1999h`

Table 14-18: **Check Watchdog Timer Expiry**

Task	Register	Register Field	Register Offset	Bits	Value
Read status register.	STATUS	WDZ	`0x0C`	0	Read operation
Wait until status register WDZ field is set. It is set when the watchdog reaches a zero count.					

Table 14-19: **Watchdog Timer Enable Signal Output**

Task	Register	Register Field	Register Offset	Bits	Value
To enable reset signal.					
Enable reset.	MODE	RSTEN	`0x00`	1	`1`
To enable IRQ signal.					
Enable IRQ.	MODE	IRQEN	`0x00`	2	`1`
Program zero access key.	MODE	ZKEY	`0x00`	23:12	`12'h0ABC`

Table 14-20: **Watchdog Timer Disable Signal Output**

Task	Register	Register Field	Register Offset	Bits	Value
If the reset signal to be disabled.					
Disable reset.	MODE	RSTEN	`0x00`	1	`0`
If IRQ signal to be disabled.					
Disable IRQ.	MODE	IRQEN	`0x00`	2	`0`
Program access value.	MODE	ZKEY	`0x00`	23:12	`12'h0ABC`

Table 14-21: **Watchdog Timer Set Control Value**

Task	Register	Register Field	Register Offset	Bits	Value
To set clock prescale value.					
Read clock prescale value.	CONTROL	CLK_SEL	`0x04`	1:0	Value to be written
To set counter reset value.					
Read counter reset value.	CONTROL	CRV	`0x04`	13:2	Value to be set
Program new zero access key.	CONTROL	CKEY	`0x04`	25:14	`12'h0248`

Table 14-22: **Watchdog Timer Get Control Value**

Task	Register	Register Field	Register Offset	Bits	Value
To read clock prescale value.					
Read clock prescale value.	CONTROL	CLK_SEL	`0x04`	1:0	Read operation
To read counter reset value.					
Read counter reset value.	CONTROL	CRV	`0x04`	13:2	Read operation

Table 14-23: **Watchdog Timer Self Test**

Task	Register	Register Field	Register Offset	Bits	Value
Read zero mode register.	MODE	All	`0x00`	31:0	Read
Select the number of clock cycles that the internal system reset is held active after it is invoked.					
Write back reset length.	MODE	All	`0x00`	31:0	Mode | RSTLN
Read back zero mode register.	MODE	All	`0x00`	6:4	Read
Write to the zero mode register is only valid if zero access key (ZKEY) is set to `0xABC`.					
Write with key value.	MODE	All	`0x00`	31:0	Mode | RSTLN| ZKEY
Read back zero mode register.	MODE	All	`0x00`	31:0	Read
Read ZKEY and compare with `0xABC`. If it is matching, hardware test passed. Otherwise, hardware test failed and hardware locking feature is functional.					
Program original register value and return success.	MODE	All	`0x00`	31:0	Mode | ZKEY

Table 14-24: **Watchdog Timer Setup Interrupts**

Task	Register	Register Field	Register Offset	Bits	Value
Initialize generic interrupt controller (GIC) controller.					
Set GIC priority trigger type.					
Register GIC interrupt handler.					
Connect GIC to the snoop control unit (SCU) watchdog timer interrupt handler.					
Enable GIC interrupt.					

Table 14-25: **Watchdog Timer Interrupt Handler**

Task	Register	Register Field	Register Offset	Bits	Value
Notify the application by setting a global variable.					

Table 14-26: **Watchdog Timer Interrupt Example**

Task	Register	Register Field	Register Offset	Bits	Value
Perform self test. Refer to Watchdog Timer Self Test.					
Set the initial counter restart to the smallest value. Refer to Watchdog Timer Set Control Value.					
Set the initial divider ratio at the smallest value. Refer to Watchdog Timer Set Control Value.					
Disable the reset output. Refer to Watchdog Timer Disable Signal Output.					
Start watchdog timer device. Refer to Watchdog Timer Start.					
Restart watchdog timer. Refer to Watchdog Timer Restart.					
Check if watchdog timer is expired. Refer to Check Watchdog Timer Expiry.					
Stop watchdog timer. Refer to Watchdog Timer Stop.					
Set up interrupt system. Refer to Watchdog Timer Setup Interrupts.					
Enable IRQ output. Refer to Watchdog Timer Enable Signal Output.					
Start watchdog timer device. Refer to Watchdog Timer Start.					
Restart watchdog timer. Refer to Watchdog Timer Restart.					
Wait till watchdog timer IRQ handler notification. Refer to Watchdog Timer Interrupt Handler.					
If no notification is received,					
Disable interrupts and return failure.					
Else, if test passed,					
Restart watchdog timer. Refer to Watchdog Timer Restart.					
Verify that the watchdog timer does not time out when restarted all the time.					
Restart watchdog timer. Refer to Watchdog Timer Restart.					
If more time has passed than it took for it to expire when not restarted in the previous test, then stop the timer.					
Check if watchdog timer is expired. Refer to Check Watchdog Timer Expiry.					
If no notification from interrupt handler, disable interrupts and return success.					

Watchdog Timer Flowcharts

Figure 14-4 shows the watchdog timer example flowchart for timer configuration and timer expiration.

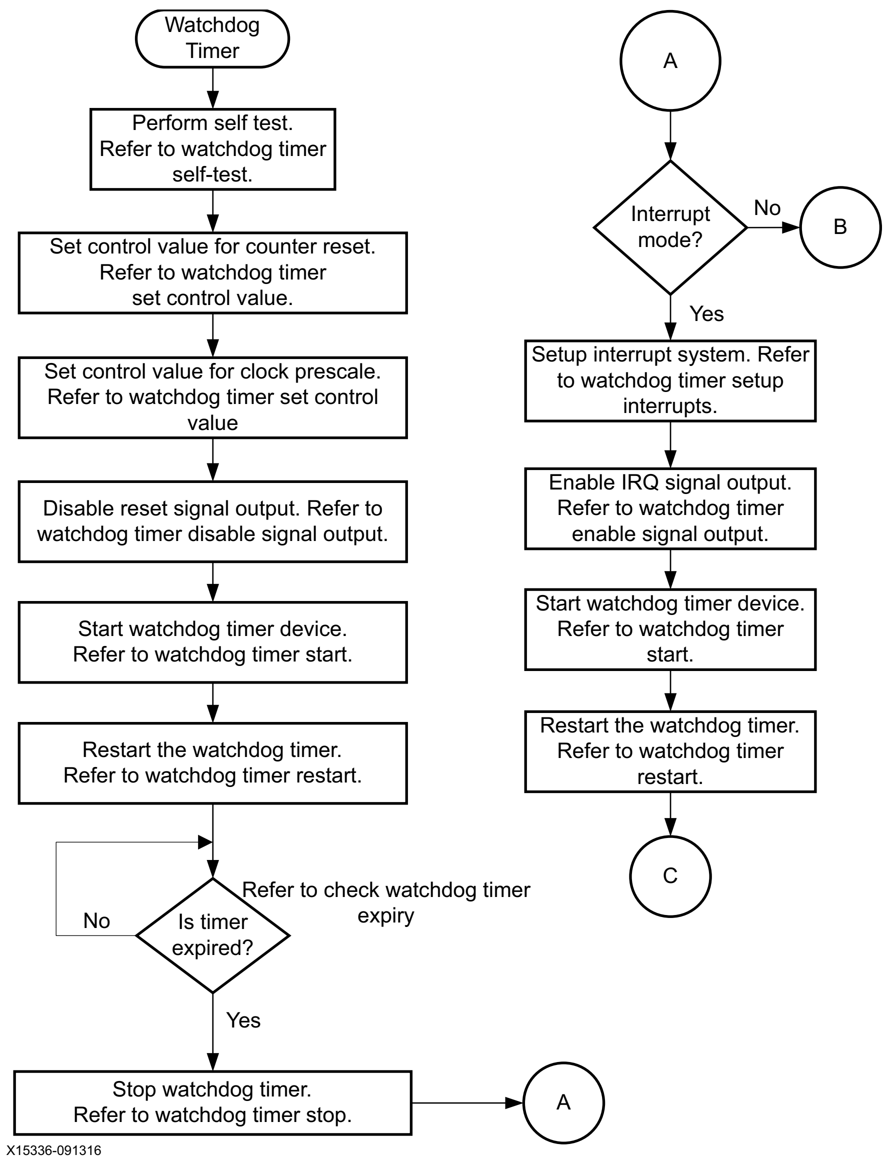

Watchdog Timer
Perform self test. Refer to watchdog timer self-test.
Set control value for counter reset. Refer to watchdog timer set control value.
Set control value for clock prescale. Refer to watchdog timer set control value
Disable reset signal output. Refer to watchdog timer disable signal output.
Start watchdog timer device. Refer to watchdog timer start.
Restart the watchdog timer. Refer to watchdog timer restart.
Refer to check watchdog timer expiry
Is timer expired?
No
Yes
Stop watchdog timer. Refer to watchdog timer stop.
A
A
Interrupt mode?
No
B
Yes
Setup interrupt system. Refer to watchdog timer setup interrupts.
Enable IRQ signal output. Refer to watchdog timer enable signal output.
Start watchdog timer device. Refer to watchdog timer start.
Restart the watchdog timer. Refer to watchdog timer restart.
C
X15336-091316

Figure 14-4: **Watchdog Timer Flowchart**

Figure 14-5 shows the watchdog timer timeout test flowchart.

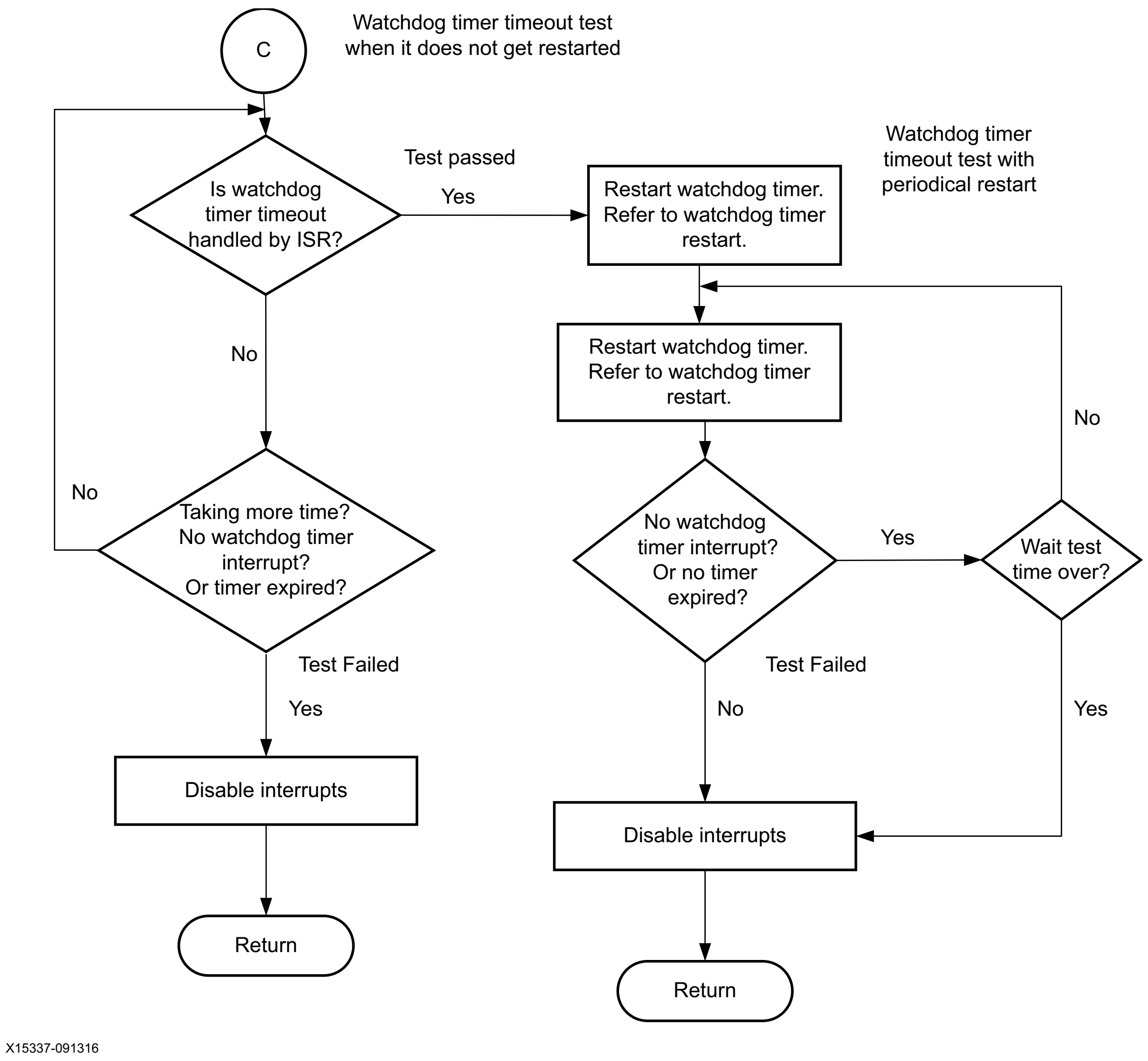

C
Watchdog timer timeout test when it does not get restarted
Is watchdog timer timeout handled by ISR?
Test passed
Yes
No
Restart watchdog timer. Refer to watchdog timer restart.
Watchdog timer timeout test with periodical restart
Restart watchdog timer. Refer to watchdog timer restart.
No
No
Taking more time? No watchdog timer interrupt? Or timer expired?
No watchdog timer interrupt? Or no timer expired?
Yes
Wait test time over?
Test Failed
Yes
Test Failed
No
Yes
Disable interrupts
Disable interrupts
Return
Return
X15337-091316

Figure 14-5: **Watchdog Timer Timeout Test**

Figure 14-6 shows the watchdog timer polled mode example flowchart.

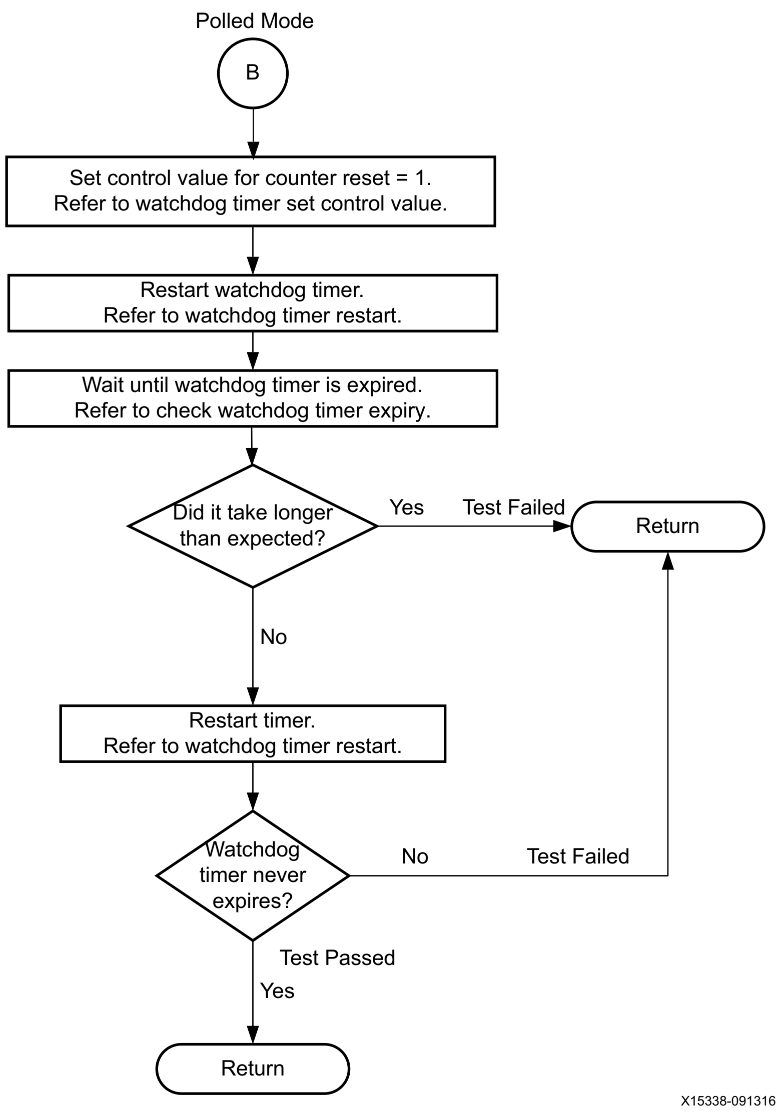

Polled Mode
B
Set control value for counter reset = 1.
Refer to watchdog timer set control value.
Restart watchdog timer.
Refer to watchdog timer restart.
Wait until watchdog timer is expired.
Refer to check watchdog timer expiry.
Did it take longer than expected?
Yes
Test Failed
Return
No
Restart timer.
Refer to watchdog timer restart.
Watchdog timer never expires?
No
Test Failed
Test Passed
Yes
Return
X15338-091316

Figure 14-6: **Watchdog Timer Polled Mode Flowchart**

MIO - EMIO Signals

Timer I/O signals are listed in the following table. There are four triple timer counters (TTC0 to TTC3) in the system. Each TTC has three sets of interface signals: clock in and wave out for counter/timers 0, 1, and 2. For each triple timer counter, the signals for counter/timer 0 can be routed to the MIO using the MIO_PIN_xx registers. If the clock in or wave out signal is not selected by the MIO_PIN_xx register, then the signal is routed to EMIO by default. The signals for counter/timers 1 and 2 are only available through the EMIO.

Table 14-27: **MIO – EMIO Signals**

TTC	PS I/O Name	Index[1]	I/O	MIO Pins	EMIO Signals	Controller Default Input Value
TTC0	ttc0_clk_in	0	I	6,14,22,30,38,46,54,62,70	emio_ttc0_clk_i[2:0]	0
	ttc0_wave_out	1	O	7,15,23,31,39,47,55,63,71	emio_ttc0_wave_o[2:0]	-
TTC1	ttc1_clk_in	0	I	4,12,20,28,36,44,52,60,68	emio_ttc1_clk_i[2:0]	0
	ttc1_wave_out	1	O	5,13,21,29,37,45,53,61,69	emio_ttc1_wave_o[2:0]	-
TTC2	ttc2_clk_in	0	I	2,10,18,26,34,42,50,58,66	emio_ttc2_clk_i[2:0]	0
	ttc2_wave_out	1	O	3,11,19,27,35,43,51,59,67	emio_ttc2_wave_o[2:0]	-
TTC3	ttc3_clk_in	0	I	0,8,16,24,32,40,48,56,64	emio_ttc3_clk_i[2:0]	0
	ttc3_wave_out	1	O	1,9,17,25,33,41,49,57,65	emio_ttc3_wave_o[2:0]	-

Notes:

1. The index numbers are listed in Table 28-1.

System watchdog timer I/O signals are listed in Table 14-28.

Table 14-28: **System Watchdog Timer I/O Signals**

SWDT	PS I/O Name	Index[1]	I/O	MIO Pins	EMIO Signals	Controller
SWDT0 (LPD_SWDT)	wdt0_clk_i	0	I	6,10,18,22,30,34,42,46,50, 62,66,70,74	emio_wdt0_clk_i	0
	wdt0_rst_o[2]	1	O	7,11,19,23,31,35,43,47,51, 63,67,71,75	emio_wdt0_rst_o [2]	~
SWDT1 (FPD_SWDT)	wdt1_clk_i	0	I	4,8,16,20,24,32,36,44,48,5 6,64,68,72	emio_wdt1_clk_i	0
	wdt1_rst_o[2]	1	O	5,9,17,21,25,33,37,45,49,5 7,65,69,73	emio_wdt1_rst_o [2]	~

Notes:

1. The index numbers are listed in Table 28-1.
2. wdt0_rst_o and wdt1_rst_o are active high.

PS Interconnect

Introduction

The interconnect located within the processing system (PS) comprises multiple switches to connect system resources using the advanced eXtensible interface (AXI) point-to-point channels for communicating addresses, data, and response transactions between master and slave clients. This Arm® AMBA 4.0 interconnect implements a full array of the interconnect communications capabilities and overlays for QoS, debug, and test monitoring.

Features

The interconnect is based on the AXI high-performance datapath switches.

- Interconnect switches based on the Arm NIC-400.
- Cache coherent interconnect (CCI-400).
- System memory management unit (SMMU) enabling use of virtual addresses.
- Separate interconnects in two power domains: full-power domain (FPD) and low-power domain (LPD).
- QoS support for better prioritizing AXI transactions.
- AXI performance monitors (APM) gather transaction metrics.
- AXI timeout block (ATB) that works as a watchdog timer for interconnect hang.
- AXI isolation block (AIB) module that is responsible for the functionally that isolates the AXI/APB master from the slave in preparation for powering down an AXI/APB master or slave.
- Interfaces between the processing system (PS) and programmable logic (PL) with the following.
    - S_AXI_HPC[0:1]_FPD and S_AXI_HP[0:3]_FPD: High-performance AXI slave ports that are accessed by AXI masters in the PL.
    - M_AXI_HPM0/1_FPD: Low-latency AXI master ports for accessing AXI slaves in the PL.

- S_AXI_ACE_FPD: Two way AXI coherency extension slave port that can be accessed by AXI masters in the PL.
- S_AXI_ACP_FPD: Cache-coherent accelerator coherency slave port that can be accessed by AXI masters in the PL.
- S_AXI_LPD: Low-power domain AXI slave ports that are accessed by AXI masters in the PL.
- M_AXI_HPM0_LPD: Low-power domain AXI master port for accessing AXI slaves in the PL.

Block Diagram

The top-level interconnect architecture is described in Figure 15-1. The LPD interconnect that is associated with the real-time processing unit (RPU) and the FPD interconnect that is associated with the application processing unit (APU) are shown in Figure 4-1.

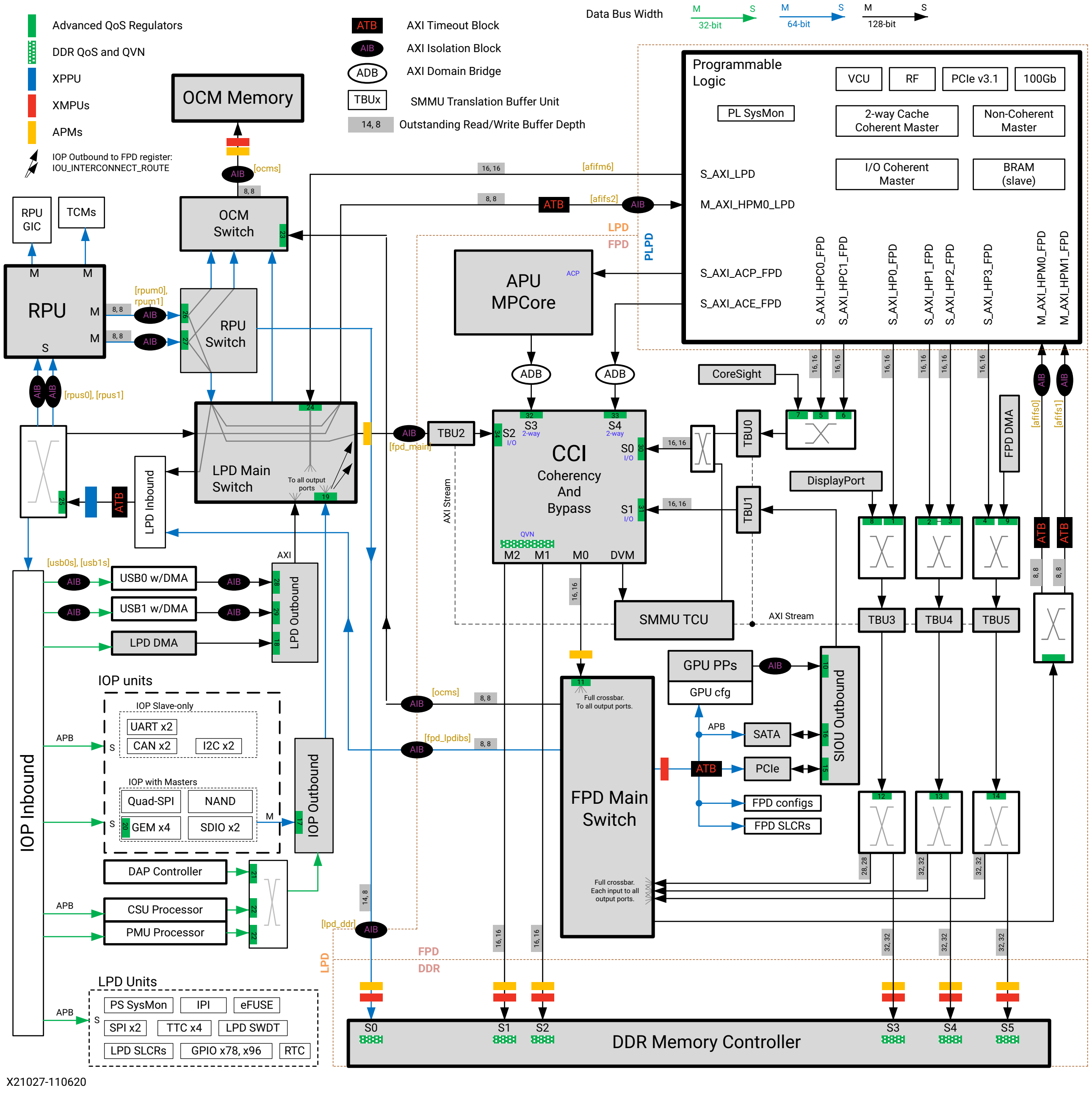


Figure 15-1: **PS Interconnect**

FPD Main Switch

The 128-bit FPD main switch is one of the switches in top-level interconnect that connects the FPD masters to the LPD slaves (including the OCM and TCM). The switch provides a direct path to the OCM (bypassing the LPD interconnect) to minimize latency and improve throughput from the FPD to the OCM. In addition, the switch provides a separate (narrow) path to access the LPD peripheral registers by the FPD masters.

Cache Coherent Interconnect

The cache-coherent interconnect (CCI) combines parts of the interconnect and coherency functions into a single block. It provides two ACE slave ports (for full coherency), three ACE-Lite slaves (for I/O coherency), two ACE-Lite master ports (for DDR), and one ACE-Lite master port for non-DDR memory-mapped accesses. It also provides the distributed virtual memory (DVM) message interface to the system memory management unit (SMMU). Figure 15-1 shows the CCI port connections. CCI registers are globally mapped and can be accessed from the LPD.

Full Coherency

Full (both-way) coherent masters can snoop each other's caches. For fine-grain data sharing between the APU and PL, a system can have cache implemented in the PL. The APU can snoop PL caches, and the PL can snoop APU caches.

I/O Coherency

The I/O (one-way) coherent masters can snoop APU caches through the CCI ACE-Lite slave ports, thus avoiding the need for software to providing coherency by flushing APU caches (when APU data is shared with I/O masters).

All of the PS masters, including the RPU but excluding the full-power DMA controller (FPD DMA), DisplayPort, and S_AXI_HP{0:3}_FPD PS masters, can be optionally configured as I/O coherent. For more information on I/O Coherency, see Zynq UltraScale MPSoC Cache Coherency [Ref 58].

ACP Coherency

The PL masters can also snoop APU caches through the APU accelerator coherency port (ACP). The ACP accesses can be used to (read or write) allocate into L2 cache. However, the ACP supports restricted transactions. See Chapter 35, PS-PL AXI Interfaces.

Interconnect Submodules

The interconnect has following sub-modules.

- Xilinx memory protection unit (XMPU): FPD, OCM, and DDR.

- Xilinx peripheral protection unit (XPPU).
- System memory management unit (SMMU).
- AXI timeout block (ATB) that works as a watchdog timer for interconnect hang.
- AXI and APB isolation block (AIB) units that are responsible for functionally isolating the AXI/APB master from the slave in preparation for powering down an AXI/APB master or slave.
- PS-PL AXI interfaces.
- AXI performance monitor.

Xilinx Memory Protection Unit

The Xilinx memory protection unit (XMPU) provides memory partitioning and TrustZone protection for memory and FPD slaves. The XMPU can be configured to isolate a master or a given set of masters to a programmable set of address ranges. The XMPU is further described in Chapter 16, System Protection Units.

Xilinx Peripheral Protection Unit

The Xilinx peripheral protection unit (XPPU) provides LPD peripheral isolation and IPI protection. The XPPU can be configured to permit one or more masters to access an LPD peripheral. The XPPU is further described in Chapter 16, System Protection Units.

System Memory Management Unit

The system memory management unit (SMMU) provides protection services of slaves and address translation for I/O masters to identify more than its actual addressing capability. In 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. The SMMU consists of the translation control unit (TCU) and multiple translation buffer units (TBUs). The protection functions are described in Chapter 16, System Protection Units. The translation functions are described in SMMU Architecture in Chapter 3.

AXI Timeout Block

There is an AXI timeout block in the interconnect to ensure that the interconnect does not hang because of a non-responding slave. This block keeps track of AXI transactions and times out when the slave does not respond within a specific time. It responds to the master with a response. This completes the AXI transaction and prevents the master from hanging forever while waiting for the response from the slave.

AXI and APB Isolation Block

Interconnect has AXI and APB isolation block (AIB) units that are responsible for functionally isolating the AXI/APB master from the slave in preparation for an AXI/APB master or slave to be powered down. The AIB manages AXI and APB interfaces during the isolation process resulting in a graceful transition to a power-down state. The AIB is transparent and offers zero latency during normal transactions. When isolation is requested, the AIB blocks all new transactions generated by a master until all the outstanding interactions are completed by the slave, then isolates the slave by responding to all new transactions on behalf of the slave.

Quality of Service Block

The quality of service (QoS) has a set of features that allow the regulating of memory traffic to meet the needs of memory client devices.

- Traffic regulating mechanism by using the NIC-400 QoS block.
- QoS latency is managed by the DDR memory controller. See Chapter 17, DDR Memory Controller.
- Deep buffer at source to increase latency tolerance.

PS-PL AXI Interfaces

There are several types of PS-to-PL and PL-to-PS AXI interfaces. They are described in Chapter 35, PS-PL AXI Interfaces.

IOP Bus Masters

The bus requests from the peripheral masters will be routed to DDR memory (non-coherent) or through the CCI (coherent). The route for the eight IOP masters are individually selected by the IOU_INTERCONNECT_ROUTE register.

ATB Timeout Description

There is an AXI timeout block in the interconnect to ensure that the interconnect does not hang because of a non-responding slave. This block keeps track of AXI transactions and times out when the slave does not respond within a specific time. It responds to the master with a response. This completes the AXI transaction and prevents the master from hanging forever while waiting for the response from the slave. Figure 15-2 describes the top-level architecture of the AXI timeout block.

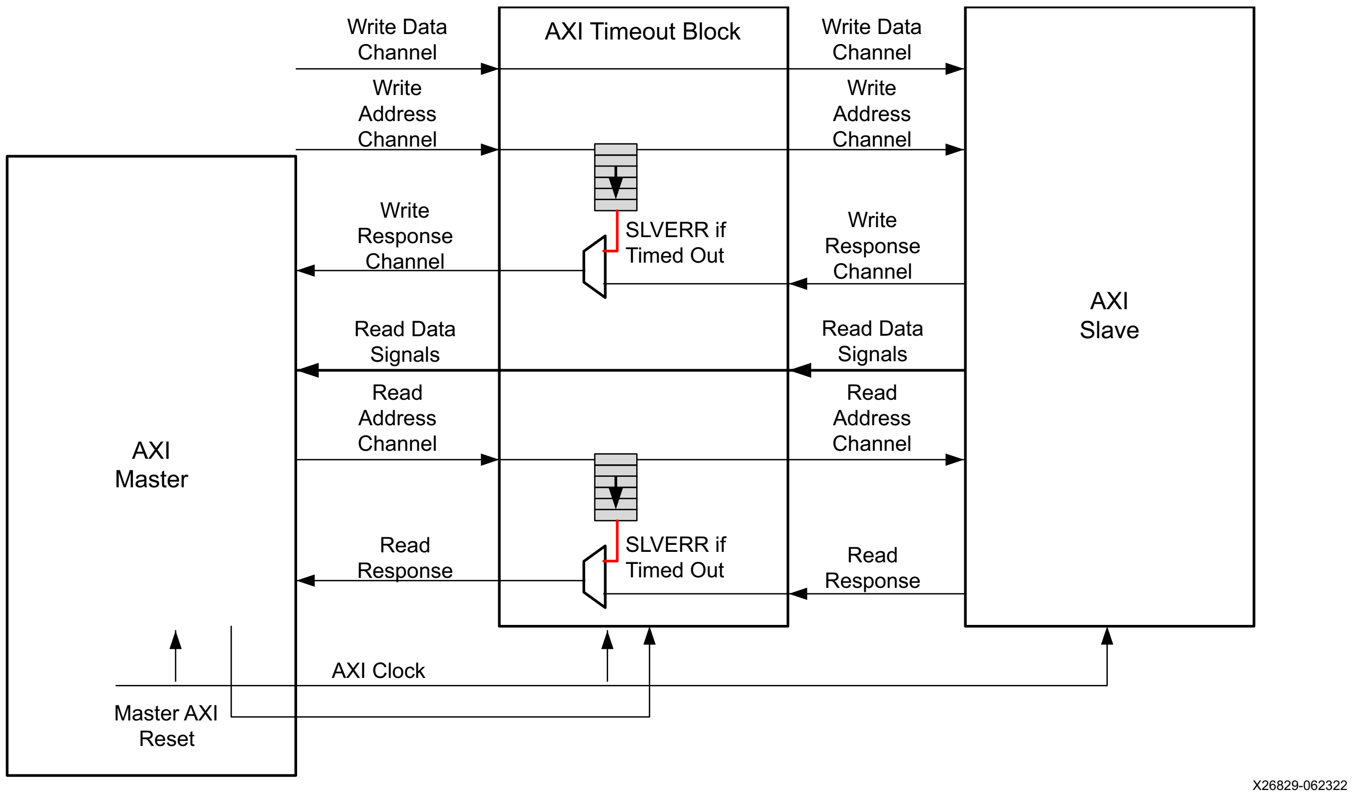


Figure 15-2: **AXI Timeout Block Architecture**

The AXI timeout block instances in the interconnect are shown in Table 15-1. These blocks in the LPD and FPD domains derive the timeout value from the ATB_PRESCALE register, which is present in the LPD_SLCR and FPDSLCR register sets, respectively.

Instances

Table 15-1: **AXI Timeout Block Instances**

Instance Number	Domain	Master Device	Slave Device
1	LPD	LPD main interconnect	M_AXI_HPM0_LPD
2	LPD	LPD inbound interconnect	Core switch
3	FPD	Core switch	PCIe and GPU
4	FPD	FPD main interconnect	M_AXI_HPM0_FPD
5	FPD	FPD main interconnect	M_AXI_HPM1_FPD

Programming

Use the following steps to enable an AXI Timeout Block. The specific registers are either part of the LPD_SLCR or FPD_SLCR module in *Zynq UltraScale+ MPSoC Register Reference* UG1087 [Ref 4].

1. Enable the ATB to send responses in the `ATB_RESP_EN` register.
2. Configure the `ATB_RESP_TYPE` register to generate a SLVERR for a timed-out AXI transaction.
3. Set the timeout value by writing to the `ATB_PRESCALE.value` register. The formula for calculating the timeout is 65536 * APB Clock Period * (ATB_PRESCALE.value + 1), where APB Clock Period is either LPD_LSBUS (LPD) or TOPSW_LSBUS (FPD).
4. If required, enable ATB interrupts by configuring `ERR_ATB_IER`.
5. Enable timeouts by writing a 1 to `ATB_PRESCALE.enable`.

To disable the ATB, clear `ATB_CMD_STORE_EN`. This prevents the ATB from tracking read and write transactions.

Note: The ATB is only able to track a limited number of transactions (<16) before hanging the requesting master. After the first indication of a timeout, it is recommended that the system treat it as a critical error that requires restart.

RECOMMENDED: *Xilinx recommends configuring these registers during boot time. To change the ATB configuration during run-time, the user must complete all outstanding AXI transactions and idle all AXI masters using the path.*

AXI Performance Monitor

The programmable AXI performance monitors (APM) collect real-time transaction metrics at multiple points on the PS AXI interconnect to help system software profile real-time activity. This section provides an overview of the APM with specific functionality of the PS-based implementation.

Features

The APM provides several features for system software to profile the PS AXI interconnect traffic.

- Clock counter for real-time profiling by system software.
- Event Counter accumulates AXI events; the counters can be set, read by system software, and used to analyze and enhance the system performance.
- Cross-probe trigger between event counter and event logging.

Implementation

The APMs are based on the Xilinx AXI Performance Monitor available as a LogiCORE IP in the PL fabric. The APM functionality is defined in *AXI Performance Monitor LogiCORE IP Product Guide* (PG037) [Ref 22]. The PS-based APMs implement the advanced mode without error logging or the AXI Stream features.

Note: While the APMs are based on the Xilinx AXI Performance Monitor IP, there are a few registers with minor differences.

PS Instances

There are four APM units in on the PS AXI interconnect and are characterized in Table 15-2. Each APM has one slot as listed in Table 15-2, except the DDR_APM has six slots corresponding to the six DDR memory controller ports.

Table 15-2: **APM Units**

Unit Name	Number of Counters	Power Domain	Clock	Register Set	Location
DDR_APM	10	FPD	TOPSW_LSBUS_CLK	APM_DDR	Six Xilinx AXI port interface (XPI) data ports on the DDR memory controller.
CCI_APM	8	FPD	TOPSW_LSBUS_CLK	APM_CCI_INTC	AXI channel from the CCI to the main switch.

Table 15-2: **APM Units *(Cont'd)***

Unit Name	Number of Counters	Power Domain	Clock	Register Set	Location
OCM_APM	8	LPD	LPD_LSBUS_CLK	APM_INTC_OCM	AXI channel from the OCM switch to the OCM memory.
LPD_APM	8	LPD	LPD_LSBUS_CLK	APM_LPD_FPD	AXI channel from the LPD switch to the FPD main switch.

The following table shows which PS clock increments the Global Clock Count Register for each instance of the APM.

Table 15-3: **GCCR Clock per APM Instance**

Unit Name	GCCR Clock
DDR_APM	DDR_REF_CLK
CCI_APM	DDR_REF_CLK
OCM_APM	CPU_R5_CLK
LPD_APM	LPD_SWITCH_CLK

Event Metric List

There are several types of events to capture. Table 15-4 lists the metrics measured by each APM. The metric selection is done in the MSR_x register.

Table 15-4: **APM Event Metric List**

Selection [SEL]	Metric	Description
0	Write Transaction Count	Number of write transactions by/to a particular master/slave. Count increments for every write address acceptance on the interface.
1	Read Transaction Count	Number of read transactions by/to a particular master/slave. Count increments for every read address acceptance on the interface.
2	Write Byte Count	Number of bytes written by/to a particular master/slave.
3	Read Byte Count	Number of bytes read from/by a particular slave/master.
4	Write Beat Count	Number of beats written by/to a particular master/slave.
5	Total Read Latency	Used with Num_Rd_Reqs (Read Transaction Count) to compute the Average Read Latency. This metric is for the selected ID transactions.
6	Total Write Latency	Used with Num_Wr_Reqs (Write Transaction Count) to determine the Average Write Latency. This metric is for the selected ID transactions.
7	Slv_Wr_Idle_Cnt	Number of idle cycles caused by the slave during a Write transaction.

Table 15-4: **APM Event Metric List *(Cont'd)***

Selection [SEL]	Metric	Description
8	Mst_Rd_Idle_Cnt	Number of idle cycles caused by the master during a read transaction.
9	Num_BValids	Number of BValids given by a slave to the master. This count helps in checking the responses against the number of requests given.
10	Num_WLasts	Number of WLasts given by the master. This count should exactly match the number of requests given by the master. This helps in debugging of the system.
11	Num_RLasts	Number of RLasts given by the slave to the master. This count helps in checking the responses against requests. This count should exactly match the number of requests given by the master.
12	Minimum Write Latency	Minimum write latency number. The default minimum write latency provided by the core is `0xFFFFFFFF`.
13	Maximum Write Latency	Maximum write latency number.
14	Minimum Read Latency	Minimum Read Latency number. The default minimum read latency provided by the core is `0xFFFFFFFF`.
15	Maximum Read Latency	Maximum Read latency number.

Register Overview

There are four similar sets of APM registers.

- APM_CCI_INTC
- APM_INTC_OCM
- APM_LPD_FPD
- APM_DDR

The APM registers are summarized in Table 15-5.

Table 15-5: **APM Register Overview**

APM_CCI_INTR, APM_INTC_OCM, APM_LPD_FPD	APM_DDR	Description
CR		Control.
RIDR, RIDMR, WIDR, WIDMR		Read and Write ID filter and mask.
FECR		Flag enables.
SWDR		Software-written data.
GCCR_H, GCCR_L		Global Clock Counter, high and low.
SIR, SICR, SISR		Sample Interval configuration and control.

Table 15-5: **APM Register Overview *(Cont'd)***

APM_CCI_INTR, APM_INTC_OCM, APM_LPD_FPD	APM_DDR	Description
GIER, IER, ISR		Interrupt enable and status.
MSR_{0,1}	MSR_{0:2}	Metric select. Slot select for APM_DDR only.
IR_{0:7}	IR_{0:9}	Increment value.
RR_{0:7}	RR_{0:9}	Range, high and low limits.
MCR_{0:7}	MCR_{0:9}	Metric count.

Synchronization mechanism is recommended before accessing registers of the same APM by multiple masters simultaneously. For example, when APU0 and RPU0 are trying to access APM_INTC_OCM registers simultaneously, both APU0 and RPU0 should use synchronization mechanism.

All APM instances exhibit a common behavior where a read to any register results in four reads inside of the APM module. The returned read data is the result of the first read. The only side effect to over-reading is if SICR.MET_CNT_RST is set to 1'b1. In this case, the metric counters are reset every time the sample register is read and a few samples are lost.

Programming Example - Read Byte Count on DDR Port 3

This simple example lists the steps to program the APM_DDR to count the number of bytes read from the DDR memory by the DisplayPort controller. The DisplayPort controller accesses DDR memory using XPI port (slot) 3 as shown in Figure 15-1. Additional programming examples are in the *AXI Performance Monitor LogiCORE IP Product Guide* (PG037) [Ref 22].

Note: The DDR XPI port 3 is shared by the DisplayPort and the S_AXI_HP0_FPD interface from the PL. There is no way for the APM to select between the AXI traffic from the PL and the DisplayPort controller. So, this example will also count the bytes read by this PL AXI interface, if it is active.

APM metric counter 7 is used for this example. All programming registers are in the APM_DDR register set.

1. Configure metric counter 7 to the XPI port (slot) 3. Write `011b` to the MSR_1 [MET_CT7_SLOT] bit field.
2. Select the read-byte count metric. Write `011b` to the MSR_1 [MET_CT7_SEL] bit field.
3. Enable the metric counter. Write 1 to the CR [MET_CNT_EN] bit.
4. Configure the sample interval time. Write `32'h000` to the SIR [SMPL_INTRVL_SIR] bit field.
5. Load the sample interval time value into the APM counter. Write 1 to the SICR [LOAD] bit.

6. Disable the down counter. Write 0 to the SICR [ENABLE] bit.
7. Reset and enable the down counter. Write `9'h101` to SICR: [MET_CNT_RST] = 1, [LOAD] = 0, and [ENABLE] = 1.
8. Get the byte count from metric counter 7. Read MCR_7 [MET_CT].

Programming Example – Metric Counter

This example lists the steps used to program the DDR APM to count the number of bytes read from the DDR memory by the DisplayPort controller. The DisplayPort controller accesses DDR memory using the XPI port (slot) 3 as shown in Figure 15-1.

APM metric counter 7 is used for this example. All programming registers are in the APMDDR register set.

1. Configure metric counter 7 to the XPI port (slot) 3. Write `011b` to the MSR_1 [MET_CT7_SLOT] bit field.
2. Select the read-byte count metric. Write `011b` to the MSR_1 [MET_CT7_SEL] bit field.
3. Enable the metric counter. Write 1 to the CR [MET_CNT_EN] bit.
4. Set the sample interval time. Write `32'h000` to SIR [SMPL_INTRVL_SIR] bit field.
5. Load the sample interval time value into the APM counter. Write 1 to the SICR [LOAD] bit.
6. Disable the down counter. Write 0 to SICR [ENABLE] bit.
7. Reset and enable the down counter. Write `9'h101` to SICR: [MET_CNT_RST] = 1, [LOAD] = 0, [ENABLE] = 1.
8. Get the byte count from metric counter 7. Read MCR_7 [MET_CT].

Quality of Service

The interconnect is built using the Arm NIC400 IP. Figure 15-1 shows the high-level block diagram of the interconnect switch hierarchy. There are six independent AXI ports on the DDR controller. In some cases, traffic classes are physically separated on the interconnect using different paths.

The AXI interconnect supports all the AXI4 signals. For some AXI masters, the interconnect provides registers for programming the value of the ArQoS and AwQoS bits.

The PL AXI masters include the following options.

- Static QoS: For programming the value of the AxQoS bits using the AFIFM.RDQoS registers.
- Dynamic QoS: The PL master can drive the QoS bits on a per transaction basis.

The NIC400 IP (interconnect) uses the following AxQoS bits for arbitration.

- AxQoS[3:0] is used to indicate the priority of the request. An `0xF` is the highest priority and an `0x0` is the lowest priority.
- In the event that more than one requester has the same AxQoS priority value, the NIC400 reverts to a least recently granted arbitration scheme to break the tie.

AXI Traffic Types

Each AXI transaction carries a traffic type based on its need to be serviced. For example, high-priority traffic carries the low latency (LL) declaration. Video and audio traffic are isochronous and must be serviced in a timely matter to avoid system degradation. The three types of AXI traffic are described in this section.

Low Latency (High Priority) Masters

For some masters, read latency is key to meeting performance requirements. In the PS, the three key low-latency masters are the APU, RPU, and SMMU. Without low-latency access to memory, the CPU spends most of the time in idle waiting for data to either be fetched from or stored to external memory space.

High Throughput (Best Effort) Masters

These masters can tolerate longer latency but they must have very high throughput to achieve an architectural goal. The typical examples are the GPU and PL. Due to the nature of these devices, they could issue a data request long before it is used to effectively cancel out latency. However, the interconnect must be able to accept multiple outstanding requests at the same time.

Isochronous (Video and Audio Class) Masters

This category of masters can tolerate longer latency in typical conditions. However, there is a critical moment (maximum latency) that data must be available without causing system breakdown. The key requirement is a guaranteed maximum latency. The typical examples requiring these masters are video encoders, camera sensors, or display devices.

QoS Subsystems

The four major components of the QoS are listed.

- AXI QoS-400 Regulators on AXI Interconnect.

- AXI and APB Timeout Units (ATB).
- AXI QoS Virtual Network Channels (QVN network).
- DDR QoS Controller in DDR memory controller.

Each one of these components implements different pieces of QoS support, but together the individual pieces form the complete QoS System solution. Of the four components, the DDR controller has the most sophisticated QoS features, but the other three components are essential to guarantee the overall required system performance.

QoS Regulator

The QoS-400 regulator provides advanced features to manage content flow through the LPD and FPD AXI interconnect. In Figure 15-1, the QoS Regulators are marked numerically. Table 15-6 table lists the QoS Regulators and the corresponding numeric representation.

Table 15-6: **QoS Regulators Mapping**

QoS Regulator	Port
afifm0m_intfpd_*	1
afifm1m_intfpd_*	2
afifm2m_intfpd_*	3
afifm3m_intfpd_*	4
afifm4m_intfpd_*	5
afifm5m_intfpd_*	6
coresightm_intfpd_ib_*	7
dp_intfpd_ib_*	8
gdma_intfpd_ib_*	9
gpu_intfpd_ib_*	10
intfpdcci_intfpdmain_ib_*	11
intfpdsmmutbu3_intfpdmain_*	12
intfpdsmmutbu4_intfpdmain_*	13
intfpdsmmutbu5_intfpdmain_*	14
pciem_intfpd_ib_*	15
satam_intfpd_ib_*	16
iopinbound_iopoutbound*	17
admam_intlpd_ib_*	18
Iopoutbound_lpd main*	19

Table 15-6: **QoS Regulators Mapping *(Cont'd)***

QoS Regulator	Port
gem0m_intiou_*	20
gem1m_intiou_*	
gem2m_intiou_*	
gem3m_intiou_*	
dap_intlpd_ib_*	21
intcsupmu_intlpd_ib_*	22
intfpd_intlpdocm_*	23
afifm6m_intlpd_ib_*	24
intlpdinbound_intlpdmain_*	25
rpum0_intlpd_*	26
rpum1_intlpd_*	27
usb0m_intlpd_ib_*	28
usb1m_intlpd_ib_	29
Qos_Control_Register_S0	30
Qos_Control_Register_S1	31
Qos_Control_Register_S3	32
Qos_Control_Register_S4	33
Qos_Control_Register_S2	34

Outstanding Command Issuing Control

The QoS-400 can be programmed to limit the maximum number of outstanding transactions possible at any one time.

Command Issue Rate Control

The QoS-400 can be programmed to limit the command issue rate.

The QoS-400 module is instantiated in almost all AXI masters in the system, Table 15-7 lists the exceptions.

Table 15-7: **AXI Masters Without QoS-400**

AXI Master	Rationale
DPDMA Controller	Classified as a video class master.
CSU	Issuing capability is one.
PMU	Issuing capability is one.

QoS Controller

This section does not attempt to describe the full details of the QoS controller operation, but describes the high-level functions as they relate to the system QoS.

One of the issues with isochronous traffic passing through the interconnect is that the timeout associated with the transaction only starts to run once the transaction enters the DDR controller. If a transaction was trapped just outside of the DDR controller, behind another transaction (perhaps due to the system being very heavily utilized), that transactions' timer would not be running. When the transaction eventually enters the DDR controller, it starts its timer running, but does not account for the elapsed time it has waited, while sitting just outside the controller. The result is that the timeout is inaccurate and fails to meet the needs of the programmed isochronous maximum latency.

One of the aims of the QoS controller is to ensure that there is space available in the memory controller CAMs for isochronous traffic at all times. It achieves this goal by monitoring the CAM levels and throttling the XPI port.

QoS Virtual Networks in CCI-400

Figure 15-3 shows how the traffic from the low-latency and best-effort masters goes to port 1 and port 2 of the DDR memory controller through the CCI-400. The CCI-400 has three master ports, two are connected to the DDR memory controller. To avoid head-of-line blocking (HOLB) on the DDR memory controller ports, the CCI-400 does not mix the best-effort traffic with low-latency traffic. This is possible because the CCI-400 allows access to the complete DDR memory through both master ports.

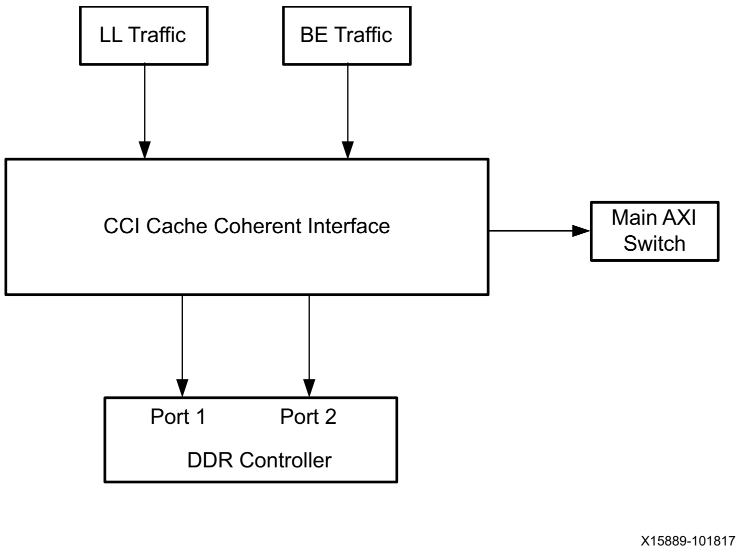


Figure 15-3: **Low Latency and Best Effort Paths through the CCI-400 to the DDR Controller**

When both low-latency and best-effort masters try to read from the same DDR memory region that is allocated to one of the master ports of the CCI-400, a mix of best-effort and low-latency traffic can be present on that particular port of the DDR memory controller. This mix can lead to head-of-line blocking on that port. To avoid a HOLB issue, the QoS virtual networks (QVN) feature is used inside the CCI-400. The QVN-aware slave is implemented to talk with the CCI-400 QVN enabled master ports.

As shown in Figure 15-4, there are two queues per port structure in the QoS controller. The red queue is mapped to low-latency traffic and the blue queue is mapped to best-effort traffic based on the AxQoS signal values. The QVN logic implementation details are listed.

- The low-latency and best-effort credit counters indicate the depth of the queue.
- When there is space available in the queue and a master virtual network requests a token, then grant the token and reduce the credit counter.
- Once the transaction is in queue, it waits until the DDR memory controller port arbitration accepts the read command.
- Once the arbitration accepts the read command, the respective pop signal is asserted.
- Assertion of the low-latency/best-effort queue command pop signal increases the respective credit counter.
- After the previous steps, the logic stops issuing a QVN token when the respective read command queue is full.
- The Push signal indicates that the master is requesting a token on the virtual channel.
- Push_Enable indicates that there is room available in the queue and a token can be issued.

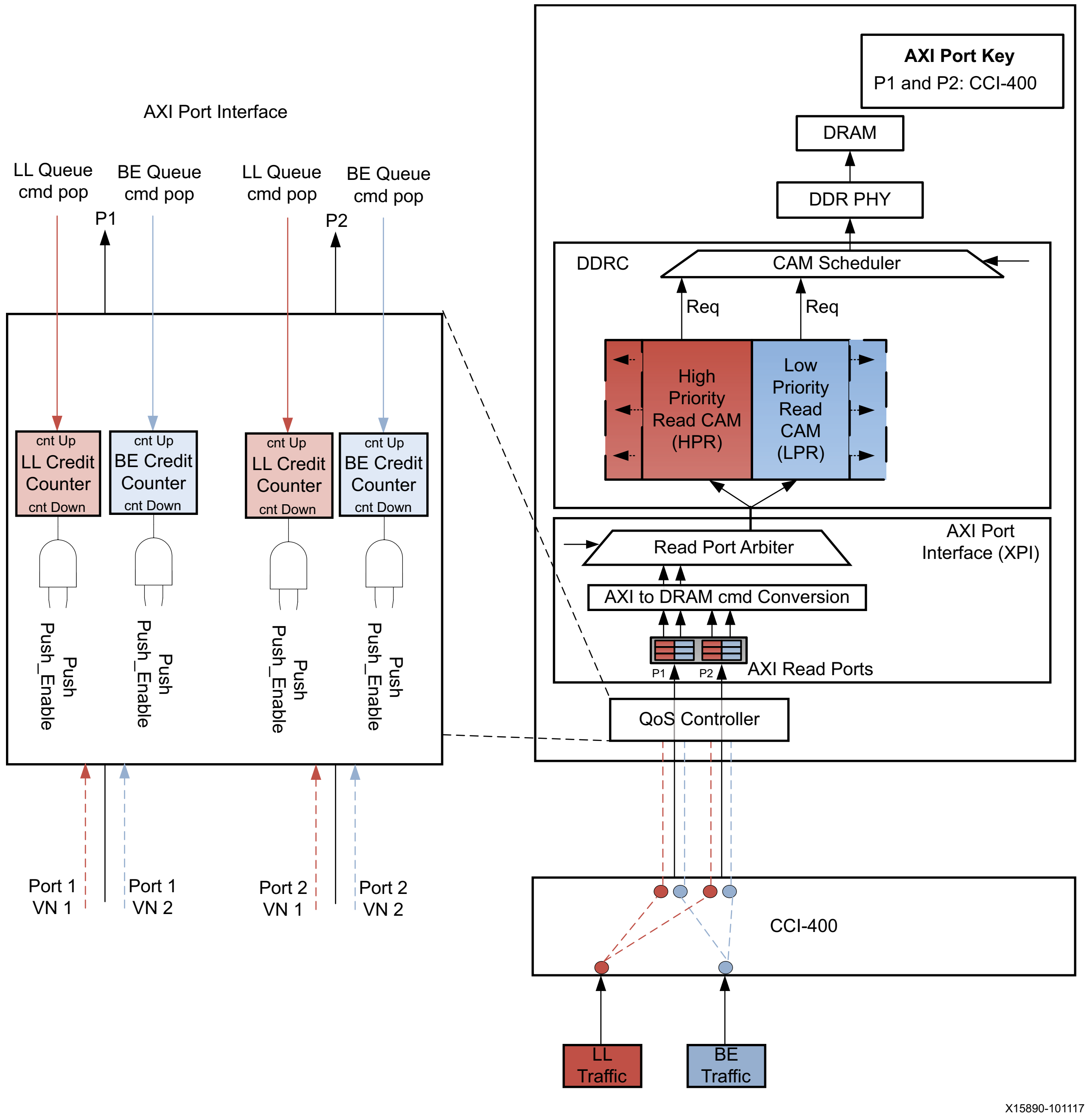


Figure 15-4: **End-to-End Path with QVN Enabled**

The QVN issues credit from the credit counter (this counter size is the same as the XPI queue depth) to the master when there is space in the XPI queue. Once credit is issued, the credit counter is decremented. When the pop signal is asserted, the transaction is popped from the XPI and the credit counter is incremented to return the credit.

DDR Controller QoS

For information about the QoS features and system limitations of the DDR controller, see Chapter 17, DDR Memory Controller.

Interconnect Register Overview

Table 15-8 is an overview of the interconnect registers.

Table 15-8: **Interconnect Registers**

Register Name	Description
CCI400 Register Set	
Control_Override_Register	Control override register.
Speculation_Control_Register	Speculation control register.
Secure_Access_Register	Secure access register.
Status_Register	Status register.
Imprecise_Error_Register	Imprecise error register.
Performance_Monitor_Control_Register	Performance monitor control register.
Snoop_Control_Register_{0:4}	Snoop control for CCI Slave interface {0:4}.
Shareable_Override_Register_S{0:2}	Shareable override for CCI Slave interface {0:2}.
Read_Qos_Override_Register_{0:4}	Read QoS override for CCI Slave interface {0:4}.
Write_Qos_Override_Register_{0:4}	Write QoS override for CCI Slave interface {0:4}.
Qos_Control_Register_{0:4}	QoS control for CCI Slave interface {0:4}.
Max_OT_Register_{0:2}	Maximum outstanding for CCI Slave interface {0:2}.
Target_Latency_Register_{0:4}	Target latency for CCI Slave interface {0:4}.
Latency_Regulation_Register_{0:4}	Latency regulation for CCI Slave interface {0:4}.
Qos_Range_Register_{0:4}	QoS range for CCI Slave interface {0:4}.
Cycle_Counter	Cycle counter.
Cycle_Counter_Control	Cycle counter control.
Cycle_Count_Overflow	Cycle count overflow.
ESR{0:3}	Event Interface and Number {0:3}.
Event_Counter{0:3}	Event counter {0:3}.
Event_Counter{0:3}_Control	Event counter {0:3} control.
Event_Counter{0:3}_Overflow	Event counter {0:3} overflow.
CCI_REG Register Set	
MISC_CTRL	Controls for the register block.

Table 15-8: **Interconnect Registers *(Cont'd)***

Register Name	Description
{ISR, IMR, IER, IDR}_0	CCI interrupt registers for address error decode, error response, and event counter overflows.
CCI_MISC_CTRL	Miscellaneous control register.
FPD_SLCR LPD_SLCR Register Sets	
ATB_PRESCALE	Prescale value for ATB timeout (AXI and APB).
ATB_CMD_STORE_EN	ATB timeout enable.
ATB_RESP_{EN, TYPE}	ATB timeout response enable and type.
ERR_ATB_{ISR, IMR, IER, IDR}	ATB error interrupts.
LPD_SLCR Register Set	
ERR_AIBAXI_{ISR, IMR, IER, IDR}	AIB AXI error interrupts.
ERR_AIBAPB_{ISR, IMR, IER, IDR}	AIB APB error.
ISO_AIBAXI_REQ	Request AXI isolation.
ISO_AIBAXI_TYPE	`1`: AIB sends SLVERR response. `0`: No response is sent.
ISO_AIBAXI_ACK	Isolation acknowledgment.
ISO_AIBAPB_REQ	Request APB isolation.
ISO_AIBAPB_TYPE	`1`: AIB sends SLVERR response. `0`: No response is sent.
ISO_AIBAPB_ACK	Isolation acknowledgment.

Chapter 16

System Protection Units

Introduction

The AXI interconnect has several system features that protect the system 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, or unintentional design.

The Arm TrustZone technology tags the security level of each AXI transaction. The Xilinx peripheral protection unit (XPPU) and the Xilinx memory protection unit (XMPU) verify that a system master is explicitly allowed to access an address. The system memory management unit (SMMU) has two sections: two-stage address translation and access protection. The two-stage address translation creates partitions to support multiple host operating systems with exclusive access to their assigned peripherals and other system elements. This functionality is described in Chapter 3, Application Processing Unit. The SMMU access protection functionality is similar to the XMPU; they work together on the AXI interconnect to support safety and security applications.

Typical AXI masters include DMA units (LPD, FPD, PL, and SIOU peripherals), RPU and APU MPCores, and PL masters accessing the system via the PS-PL AXI interfaces. The PS slaves include control and status registers and memory (DDR, OCM). Slaves in the PL must be protected by logic configured in the PL fabric.

The system protection functionality includes several features.

- System protection starts with masters that generate AXI transaction requests:
  - Master ID (unique for each AXI master).
  - Address (physical, intermediate physical, or virtual).
  - TrustZone secure or non-secure (NS).
  - Read or write.
- Transaction security state can be modified by a translation buffer unit (TBU) of the SMMU (translation function).

- Secure slaves are protected against non-secure transactions by several mechanisms:
    - XPPU protection unit protects IOP slave ports, SIOU slave ports, and Quad-SPI memory.
    - Multiple XMPU protection units protect DDR and OCM memory, and FPD slaves.
    - SMMU with multiple TBUs control accesses by processors in the PS and PL.
    - Hardware configured register sets that always require a secure transaction.
- Write-protected registers limit access by errant code.
- Local processor registers are only accessible by a single processor.
- Isolation walls provide a hard separation between power domains (and islands). See Chapter 6, Platform Management Unit.

The Xilinx protection units, SMMU translation buffer units with protection mechanisms, and system masters and slaves are shown in Figure 16-1.

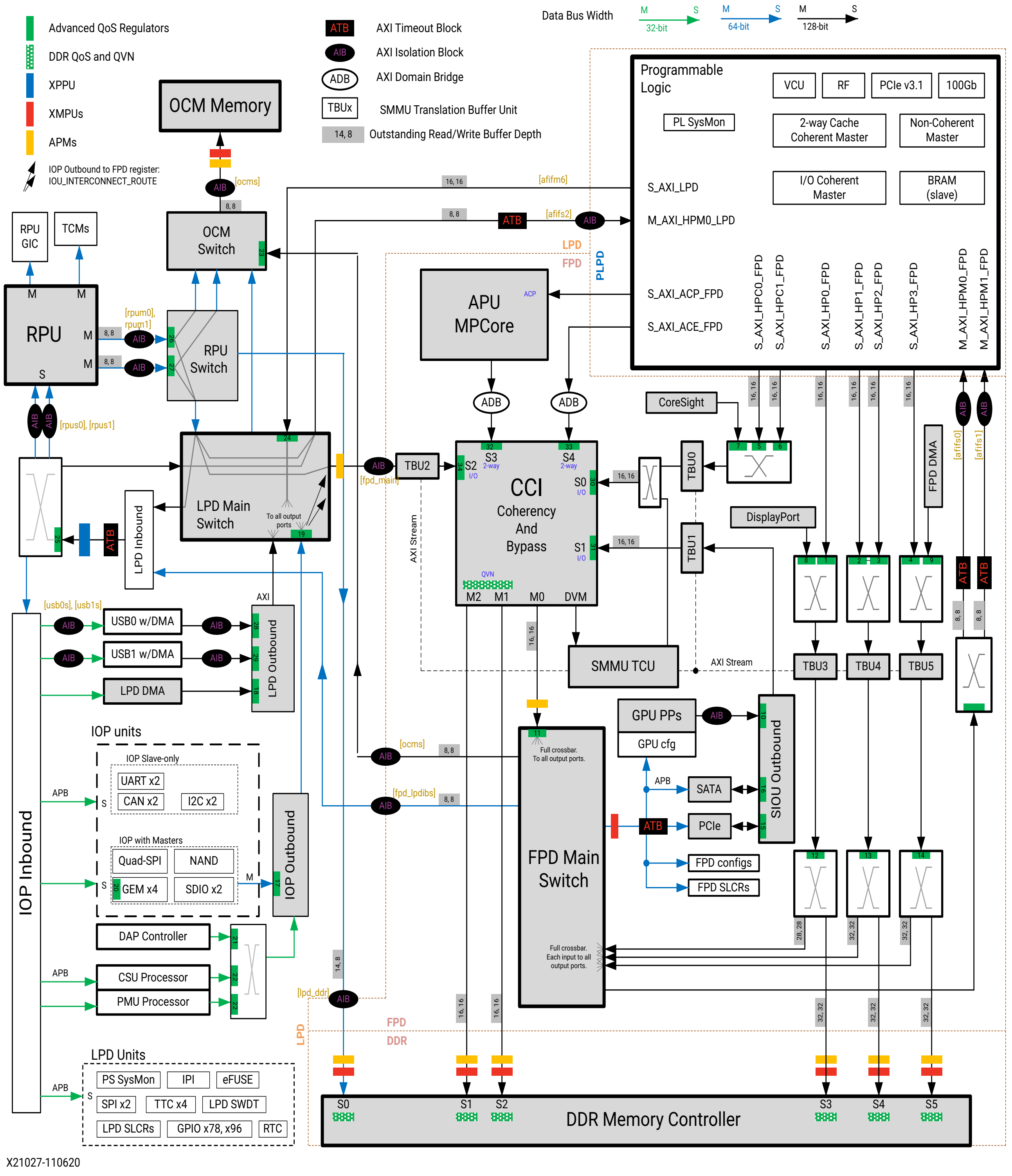


Figure 16-1: **System Protection Units**

Secured Register Sets

Some control register sets always require a secure transaction. All other register sets are protected by the XPPU protection unit except the SATA, PCIe, and GPU register sets are protected by the FPD_XMPU.

- XPPU protected (majority of registers).
- XMPU protected (SIOU controller registers).
- Hardware protected (always secure registers).

The security protections for the register sets are listed with their addresses in Table 10-4.

Write-Protected Registers

Several register sets include a write protection mechanism to avoid inadvertent register writes. The register write protection mechanism is not a security function. The register write protection feature can be used in an open development environment to block errant code from accessing important system-level functions. The write-protected registers are listed in the Write-Protected Registers Table section.

Processor-only Accessible Registers

There are several register sets that are only accessible to a processor (e.g., PMU, CSU, RPU, and APU):

- PMU address map (PMU_LOCAL_REG, PMU_IOMODULE, PMU_LMB_BRAM).
- CSU address map.
- RPU address map (RPU GIC registers).
- APU address map (PPI interrupts).

TrustZone Security

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 the TrustZone section.

SMMU Protection

The SMMU offers isolation services in addition to its address translation features. The two-stage address translation for I/O devices can also affect the transaction context. The SMMU also provides transaction filtering to isolate the transactions masters.

To offer isolation and memory protection, the SMMU restricts device access of DMA-capable I/O to a configured physical address space.

The protection features are described in the SMMU Protection on CCI Slave Ports section. The address translation functions are described in the System Memory Virtualization Using SMMU Address Translation section in Chapter 3, Application Processing Unit.

XPPU and XMPU Protection Units

The AXI interconnect has two types of protection units to monitor bus transactions on an AXI channel. Each unit determines if the type of transaction and its master are allowed to access the memory location. The XMPU appears on several AXI channels to protect the DDR system memory, OCM memory, and the SIOU address space. There is one XPPU to protect the IPI buffers, control and status registers on the IOP inbound switch, other non-DDR memory, and the Quad-SPI memory space.

The protection units are shown in Figure 16-1 as one blue and several small red rectangles.

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 Quad-SPI controller could also require protection and be dedicated for use by the RPU. To accomplish these requirements the following is required:

- The RPU generates secure transactions.
- The XMPU protects the region of the OCM for the RPU and makes the rest available for use by other masters.
- The XPPU protects the UART controller and Quad-SPI controller for use by the RPU.

Similarly, to protect access to the PL through the PS PCAP interface, the XPPU can be programmed to protect the CSU subsystem and DMA register sets.

Terminology

Table 16-1 summarizes the system protection units terminology.

Table 16-1: **System Protection Units Terminology**

	Descriptions
TrustZone	Allows and maintains isolation between the secure and non-secure processes within the same system.
SMMU	System Memory Management Unit includes one translation cache unit (TCU) and six translation buffer units (TBU). 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.
XMPU	Xilinx Memory Protection Units (8 units). Provides memory partitioning and TrustZone protection for memory and FPD slaves.
XPPU	Xilinx Peripheral Protection Unit (1 unit). Provides LPD peripheral isolation and IPI protection.
ATB	AXI Timeout Block. Prevents AXI masters from lockup if the slave does not respond within a programmed time. The ATB generates a response back to the master if the transaction times out. This term should not be confused with the CoreSight advanced trace bus (ATB) unit.
AIB	AXI Isolation Block. Provides functional isolating between the AXI master and the slave in preparation for an AXI master or slave to be powered down. The AIB is useful in a power management use case where, to save power, the FPD can be powered down when not needed.

Note: In this chapter, the term secure implies the TrustZone classification. Secure is not meant to imply secure boot, encryption, or authentication.

TrustZone

TrustZone technology is a software-controlled, hardware-enforced system for separating secure and non-secure AXI transactions. Devices and peripherals 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. 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 sideband signals.

Since 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.

Architecture

The PS AXI interconnect supports a 40-bit physical address, where the 41st bit (=AxPROT[1]) indicates secure or non-secure access. 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 (ROS) and TEEs 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.
- DDR and OCM memory can include secure and non-secure regions:
  - Programmable on a per region basis (1 MB for DDR, 4 KB for OCM).

    - Configurable using the 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 Table

The security profile for master and slaves are listed in Table 16-2.

Table 16-2: **TrustZone Profile**

PS Entity	Slave Port	Master Port	Notes
APU			
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_REG control registers	Both (internal)	~	Cache coherent interconnect (CCI) control registers can be configured to be secure or non-secure.
CCI GPV	Secure	~	Can be programmed to have a non-secure access to all the CCI 400 registers.
SMMU			
TCU APB	Secure	~	SMMU_REG
TBU AXI	Both	Both	Programmable.
XPPU, XMPU			
APB interface	Secure	~	XPPU, XMPU_{DDR, FPD, OCM} registers
AXI interface	Both	Both	Programmable.

Table 16-2: **TrustZone Profile *(Cont'd)***

PS Entity	Slave Port	Master Port	Notes
FPD and LPD DMA Units			
DMA channels	SLCR configurable	SLCR configurable	Programmable on a per channel basis.
RPU			
RPU R5_0/1	~	SLCR configurable	
RPU TCMs	XPPU configurable	~	External AXI slave port.
LPD Peripherals and Slaves			
Secure SLCR	Secure	~	See Table 10-4.
CSU	Secure	Secure	
PMU	Secure	Secure	
eFUSE/BPD/PS_SYSMON	Secure	~	Fuses, battery power unit, PS SYSMON unit.
CoreSight	Secure	Secure	
IOP peripherals	XPPU configurable	SLCR configurable	I2C, GPIO, SPI, GEM Ethernet, SDIO, CAN, USB, UART, Quad-SPI, and NAND.
LP slave interfaces on APB	XPPU configurable	~	Potential secure slaves: reset-controller.
TTC{0:3}	Configurable	~	
{LPD, FPD, CSU}_SWDT	TBD	~	
FPD Peripherals and Slaves (FPD_GPV) can be configured to be secure.			
Secure SLCR	Secure	~	
GPU/SATA/DP/PCIe	XPPU configurable	SLCR configurable	
FP slaves APB	XPPU configurable	~	Potential secure slaves: reset-controller and PCIe.
DDR System Memories and OCM			
OCM	XMPU configurable	~	Secure/non-secure per region with 4 KB granularity.
DDR DRAM	XMPU configurable	~	Secure/non-secure per region with 1 MB granularity.

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.

Table 16-3: **CCI Registers**

Module Name	Registers	Description
CCI_REG	MISC_CTRL	Controls for the register block
	ISR_0	Interrupt Status Register
	IMR_0	Interrupt Mask Register
	IER_0	Interrupt Enable Register
	IDR_0	Interrupt Disable Register
	CCI_MISC_CTRL	Misc. Control Register
CCI_GPV (CCI 400)	Control_Override_Register	Additional control register that provides a fail-safe override for some CCI-400 functions.
	Speculation_Control_Register	Disables speculative fetches for a master interface or for traffic through a specific slave interface.
	Secure_Access_Register	Secure_Access_Control, Enable non-secure access to CCI-400 registers
	Status_Register	Safely enables and disables snooping
	Imprecise_Error_Register	Records the CCI-400 interfaces that receive an error that is not signaled precisely.
	Performance_Monitor_Control_Register	Controls the performance monitor.
	Snoop_Control_Register_S0/S1/S2/S3/S4	One Snoop Control Register exists for each slave interface.
	Shareable_Override_Register_S0/S1/S2/S3	Overrides shareability of normal transactions
	Read_Qos_OverCCride_Register_S0/S1/S2/S3/S4	Contains override values for **ARQOS**, with a register for each slave interface.
	Write_Qos_Override_Register_S0/S1/S2/S3/S4	Contains override values for **AWQOS**, with a register for each slave interface.
	Qos_Control_Register_S0/S1/S2/S3/S4	Controls the regulators that are enabled on the slave interfaces.
	Max_OT_Register_S0/S1/S2	Determine how many outstanding transactions are permitted when the OT regulator is enabled for each ACE-Lite slave interface.
	Target_Latency_Register_S0/S1/S2/S3/S4	Determine the target latency, in cycles, for the regulation of reads and writes.
	Latency_Regulation_Register_S0/S1/S2/S3/S4	Latency regulation value, **AWQOS** or **ARQOS**, scale factor coded for powers of 2 in the range 2-5-2-12, to match a 16-bit integrator.

Table 16-3: **CCI Registers *(Cont'd)***

Module Name	Registers	Description
CCI_GPV (CCI 400) *(Cont'd)*	Qos_Range_Register_S0/S1/S2/S3/S4	Enables you to program the minimum and maximum values for the **ARQOS** and **AWQOS** signals that the QV regulators generate.
	Cycle_Counter	The cycle counter counts either every CCI-400 clock cycle depending on the **PMCR** bit.
	Cycle_Counter_Control	Enable or disable the cycle and event counters.
	Cycle_Count_Overflow	Detects for an overflow of the event counter.
	Event_Select_Register_0/1/2/3 Selects the event.	
	ESR0/1/2/3	
	Event_Counter0/1/2/3	
	Event_Counter0/1/2/3_Control	
	Event_Counter0/1/2/3_Overflow	
	Event_Counter_0/1/2/3	Indicates the number of events occur.
	Event_Counter_0/1/2/3_Control	Enables or disables the event counter.
	Event_Counter_0/1/2/3_Overflow	Detects for overflow of the event counter.

TrustZone System-level Control Registers

The system-level control registers (SLCR) contains the LPD_SLCR_SECURE and FPD_SLCR_SECURE register sets with the TrustZone control registers.

Note: These SLCR registers are always secure, which means that the reads and writes are always done with a secure AXI transaction.

These registers include security controls:

- Peripherals and RPU security controls.
- FPGA advanced eXtensible interface (AXI) master ports security control.

Register Write Protection Lock

Further protection is provided by using a secure configuration register lock. Once set, it prevents all further write accesses to the security register subset of the SLCR, regardless of its security status, until a power-on reset is detected.

PL TrustZone Extension

The AXI TrustZone signals extend into the PL, allowing users to build trusted master and slave devices within the PL. The secure (encrypted) bitstream is designed for the PL to be as secure as any other secure element in the PS. The security state of the PL to PS AXI interface masters is controlled by the PL.

DDR TrustZone Protection

All of the transactions going to the DDR memory port interfaces provide TrustZone security protection by six XMPUs.

APU MPCore TrustZone Model

The TrustZone technology allows and maintains isolation between secure and non-secure processes within the same system. A secure mode can access both secure and non-secure *worlds*, but a non-secure mode can only access a non-secure *world*. The Arm technical reference manual contains further implementation details. Figure 16-2 shows the Arm v8 modes.

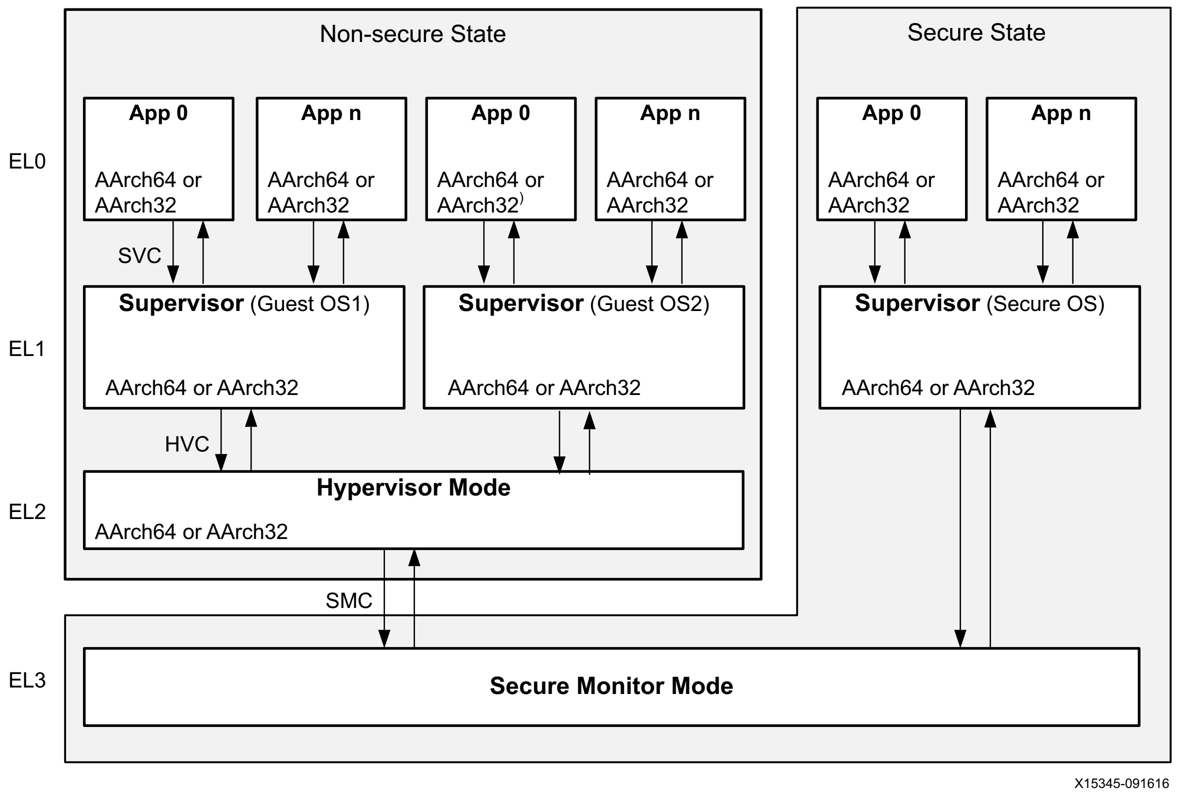


Figure 16-2: **Arm v8 Modes**

Notes relevant to Figure 16-2.

- AArch64 is permitted only if EL1 is using AArch64.
- AArch64 is permitted only if EL2 is using AArch64.
- EL3 is the most secure exception level.
- SVC instruction generates a supervisor call. It is normally used to request privileged operations.
- HVC instruction causes a hypervisor call exception and processor mode changes to the hypervisor.
- Secure monitor call (SMC) is used to enter the secure monitor mode.

SMMU Protection on CCI Slave Ports

The system memory management unit (SMMU) is described in SMMU Architecture in Chapter 3. The system protection aspects of the SMMU are emphasized in this section. The SMMU can be described as a hardware assist to provide address translation and isolation to the attached AXI masters. SMMU protection is discussed in both a native (non-virtualized) and a virtualized scenario.

Address Translation Isolation (Native, Non-Virtualized Scenario)

The SMMU provides address translation for an I/O device to identify more than its actual addressing capability. In absence of memory isolation, I/O devices may be able to 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.

As an example, consider the AXI interfaces from programmable logic to the PS that passes 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.

Guest Domain Isolation (Virtualized Scenario)

As described in Chapter 3, 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 [Ref 50].

TBU Instances

There are six TBUs supported by the SMMU TCU. These are listed in Table 16-4 with their system masters.

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. Bits [9:0] of the stream ID are the master ID defined in Table 16-13. Bits [14:10] are the TBU number the master transaction passes through. For example, for the GEM0 PS master, its master ID is `10'h074` and its TBU number is `5'h02`. Concatenating these fields gives GEM0 a stream ID of `15'h0874`.

Table 16-4: **System Masters**

SMMU Unit	System Masters
TBU0	S_AXI_HPC{0, 1}_FPD SMMU TCU CoreSight
TBU1	SIOU peripheral's DMA units
TBU2	LPD
TBU3	S_AXI_HP0_FPD DisplayPort
TBU4	S_AXI_HP{1, 2}_FPD
TBU5	S_AXI_HP3_FPD FPD DMA

XMPU Protection of Slaves

The XMPU is a region-based memory protection unit. This section describes the XMPU in detail, including configuration and functionality.

The XMPU interface consists of the following features:

- Slave AXI port to receive a transaction.
- Master AXI port with poison output.
- APB slave for programming the control registers.
- Interrupt for AXI and register access violations.
- AXI clock (same for master and slave ports) and APB clock for register programming.
- Lock register - once set, the lock is only resettable by a POR reset.
- Memory partitioned and protected to isolate a master or a given set of masters to a programmable set of address ranges.

- Six DDR XMPUs provide 1 MB memory apertures.
- FPD and OCM XPMUs provide 4 KB memory apertures.
- TrustZone protection for ports going into the DDR memory controller is provided using secure/non-secure bits in the register for masters that cannot drive the AxPROT[1] bit.
- AXI transaction permission violation interrupt.
- APB slave interface address decode error interrupt.

Architecture

The system block diagram (Figure 16-1) shows the AXI interfaces and APB bus structures connected to the XPPU and eight XMPUs.

The poison by attribute method allows the AXI transaction to continue to the memory with an option to set the [POISON] attribute that is received by the memory unit. The poison by address method redirects the AXI transaction to its sink unit.

XMPU 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 possibilities, 1 MB and 4 KB, depending on the XMPU unit. For the XMPU configured with the 1 MB region alignment, the start address of each region is 1 MB aligned. Similarly, for the OCM_XMPU configured with the 4 KB alignment, the start address is 4 KB aligned.

When a memory space has overlapping regions the higher region number has higher priority (for example, region 0 has the lowest priority). An overlapping region is defined as a region where both the MID and address match. An address match with different MIDs is not considered to be an overlapping region. Each region can be independently enabled or disabled. If a region is disabled, it is not used for 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 (allow or poison) as specified in the XMPU control register. There are two ways to poison a request: forward the transaction with a poison attribute or poison (replace) the upper address bits and then forward the transaction.

Poison Attribute Signals

An AXI request can be poisoned by adding poison attribute signals on AxUSER and then passing the poisoned request to a connected AXI slave. When the destination slave (for example, a DDR memory interface port) receives a poisoned request, it handles the request with a write-ignore/read-all-zero (WI/RAZ) response, and, optionally generates a DECERR. Only the DDR_XMPU and OCM_XMPU support poison attribute signals.

Poison Address

An AXI request can be poisoned by changing the address to a preprogrammed poison address. A 4 KB poisoned address aperture is defined as an AXI slave, which is an XMPU sink. The interconnect routes the 4 KB poisoned address to this sink, which responds to poisoned transactions similar to an undefined register space. This results in either a data abort or an interrupt to the processor.

Region Checking Operation

An incoming read or write request on an AXI port is checked against each XMPU region as described in this section.

TIP: *When a memory space has overlapping regions, the higher region number has higher priority (for example, region 0 has the lowest priority). An overlapping region is defined as a region where both the MID and address match. An address match with different MIDs is not considered to be an overlapping region. 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 Equation 16-1 is satisfied (along with security and read/write checks), the transaction is allowed. In Equation 16-1, these are [10-bit parameters] in the Rxx_MASTER region register:

$$[ID]\ \&\ [MASK] == AXI\_MasterID\ \&\ [MASK] \qquad \text{Equation 16-1}$$

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 XMPU 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 XMPU Error Handling.

See *Isolation Methods in Zynq UltraScale+ MPSoCs* (XAPP1320) [Ref 38] for more information.

Instances

Table 16-5 lists the system protection units.

Table 16-5: **System Protection Units**

Protection Unit	System Slave	System Masters	Control Registers
DDR XMPU0	DDR Port 0	RPU	0xFD00_0000
DDR XMPU1	DDR Port 1	CCI (APU, S_AXI_ACE_FPD, TCU)	0xFD01_0000
DDR XMPU2	DDR Port 2	CCI (APU, S_AXI_ACE_FPD, TCU)	0xFD02_0000
DDR XMPU3	DDR Port 3	S_AXI_HP0_FPD, DisplayPort	0xFD03_0000
DDR XMPU4	DDR Port 4	S_AXI_HP{1,2}_FPD	0xFD04_0000
DDR XMPU5	DDR Port 5	S_AXI_HP3_FPD, FPD DMA	0xFD05_0000
FPD XMPU	SIOU, APU GIC, SMMU TCU, FPD XMPU, others(1)	FPD main switch to SIOU	0xFD5D_0000
OCM XMPU	OCM	LPD main switch to OCM	0xFFA7_0000

Notes:
1. See Table 16-6 for more information.

Note: The DDR_XMPU{1, 2} protection units are located on two parallel AXI channels between the CCI and the DDR memory controller. They can be configured identically, or differently depending on how these channels are used. The QVN virtual network controller in the CCI works with the DDR memory controller for optimal system performance of these two memory paths for bulk, isochronous, and low latency transactions.

Table 16-6 lists the peripherals secured by FPD_XMPU.

Table 16-6: **Peripherals Secured by FPD_XMPU**

Memories/Peripherals	Protection Unit	Address
ACPU_GIC	FPD_XMPU	0xF900_0000
AFI 0/1/2/3/4/5	FPD_XMPU	0xFD36_0000 0xFD37_0000 0xFD38_0000 0xFD39_0000 0xFD3A_0000 0xFD3B_0000
APM 0/5	FPD_XMPU	0xFD0B_0000 0xFD49_0000
APU	FPD_XMPU	0xFD5C_0000
CCI_GPV	FPD_XMPU	0xFD6E_0000
CCI_REG	FPD_XMPU	0xFD5E_0000
CRF_APB	FPD_XMPU	0xFD1A_0000
DDDR_CTRL	FPD_XMPU	0xFD07_0000
DDR_PHY	FPD_XMPU	0xFD08_0000
DDR_QOS_CTRL	FPD_XMPU	0x FD09_0000
DDR_XMPU0/1/2/3/4/5_CFG	FPD_XMPU	0xFD00_0000 0xFD01_0000 0xFD02_0000 0xFD03_0000 0xFD04_0000 0xFD05_0000
DISPLAY PORT	FPD_XMPU	0xFD4A_0000
DPDMA	FPD_XMPU	0xFD4C_0000
FPD_DMA_CH0/1/2/3/4/5/6/7	FPD_XMPU	0xFD50_0000 0xFD51_0000 0xFD52_0000 0xFD53_0000 0xFD54_0000 0xFD55_0000 0xFD56_0000 0xFD57_0000
FPD_GPV	FPD_XMPU	0xFD70_0000
FPD_SLCR	FPD_XMPU	0xFD61_0000
FPD_SLCR_SECURE	FPD_XMPU	0xFD69_0000

Table 16-6: **Peripherals Secured by FPD_XMPU *(Cont'd)***

Memories/Peripherals	Protection Unit	Address
FPD_XMPU_CFG	FPD_XMPU	0xFD5D_0000
FPD_XMPU_SINK	FPD_XMPU	0xFD4F_0000
GPU	FPD_XMPU	0xFD4B_0000
PCIE_LOW	FPD_XMPU	0xE000_0000
PCIE_HIGH_1/2	FPD_XMPU	0x6000_0000 0x8000_0000
PCIE_MAIN	FPD_XMPU	0xFD0E_0000
PCIE_ATTRIB	FPD_XMPU	0xFD48_0000
PCIE_DMA	FPD_XMPU	0xFD0F_0000
RCPU_GIC	FPD_XMPU	0xF900_0000
SATA	FPD_XMPU	0xFD0C_0000
SERDES	FPD_XMPU	0xFD40_0000
SIOU	FPD_XMPU	0xFD3D_0000
SMMU	FPD_XMPU	0xFD80_0000
SMMU_REG	FPD_XMPU	0xFD5F_0000
SWDT1	FPD_XMPU	0xFD4D_0000

Error Handling

XMPU Error Handling

Errors can occur from security violations. The errors can be due to read or write transactions. When an error occurs, the XMPU poisons the request, records the address and master ID of the first transaction that failed the check, flags the violation, and, optionally generates 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. For simultaneous read and write errors, only the write error is recorded.

IMPORTANT: *The following is required for the various XMPU instances if used in the Zynq UltraScale+ MPSoC to function properly. When using a Xilinx delivered tool flow, they are already setup by the first-stage boot loader (FSBL).*

For the XMPU instances, DDR, and OCM memories, a poison attribute is used by programming the XMPU CTRL [PoisonCfg] bit to 0 and the XMPU POISON [ATTRIB] bit to 1.

When a violation occurs the base address is recorded and the AxUser [10] poison bit is set for the AXI transaction. For write transactions, the memory controller masks the write data and can assert a DECERR and/or interrupt. For read transactions, the memory controller returns zeros and can assert DECERR and/or interrupt.

The responses by DDR Memory Controller are configured by DDRC.POISONCFG register. The DECERR response by the OCM Memory Controller is controlled by the OCM_ERR_CTRL [PZ_ERR_RES] bit.

For the XMPU instance in the FPD interconnect, a poison address is used by programming the XMPU CTRL [PoisonCfg] bit to 1 and the POISON [ATTRIB] bit to 0. The poison address is fixed in the XMPU read-only POISON [BASE] bit field to point to XMPU_SINK at 0xFD4F_0000. When a violation occurs, the incoming base address is recorded and a new outgoing base [BASE] is applied to the AXI transaction. This addresses the FPD_XMPU_SINK unit where the offset address is recorded, a PSLVERR is returned and an interrupt is generated.

Configuration

Only secure masters can read/write the XMPU. It should only be configured once at boot time. Once it is configured at boot time, 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, then 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. If you program an XMPU, program all its settings. This ensures only the programmed transactions will go through.*

Alignment and Poison Configuration

The recommended bit settings for the XMPU alignment and poisoning configurations are listed in Table 16-7 with poison attribute and base settings. The DDR and OCM memory controllers expect attribute poisoning when the AXI transaction is disallowed by an XMPU. The FPD XMPU must poison the transaction so it is steered to the XMPU sink unit.

Table 16-7: **XMPU Configuration Table**

Register Bit Field Bit Field Type Bit Field Meaning	CTRL [AlignCfg] Read-only 1 = 1 MB 0 = 4 KB	CTRL [PoisonCfg] R/W, reset value 0 = attribute 1 = address	POISON [ATTRIB] R/W, reset value AxUser bit	POISON [BASE] Read-only Address bits [31:12]
6 XMPUs on DDR	1	0	0	~
1 XMPU on OCM	0	0	0	~
1 XMPU on FPD	0	1	~	`20'h FD4F0`

Block Diagram

The AXI transaction data paths for the XMPUs are shown in Figure 16-3.

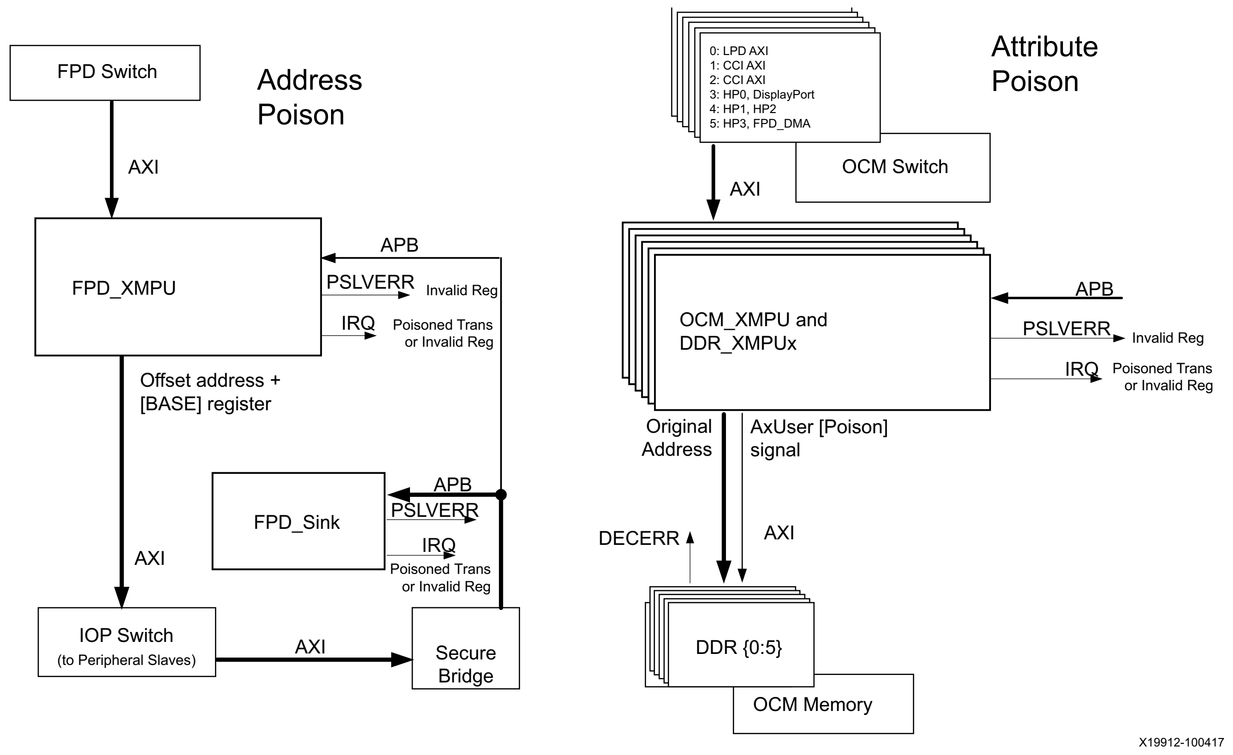


Figure 16-3: **XMPU Poison Methods Block Diagram**

XMPU Register Set Overview

Each XMPU protection unit is controlled by its own register set. The XMPU registers are listed in Table 16-8. The base addresses for each XMPU is listed in Figure 16-4.

Table 16-8: **XMPU Register Summary**

Address	Register Names	Number of Registers	Description
XMPU Control and Status			
0x0000	CTRL	1	Default read/write, poison, and alignment configuration.
0x0004+	ERR_STATUS1, ERR_STATUS2	2	Poison address and master ID value (FPD_XMPU), or Poison attribute and base address (DDR_XMPU and OCM_XMPU).
0x000C	POISON	1	Base address of XPPU sink.
0x0010+	ISR, IMR, IEN, IDS	4	Interrupt controls: address decode error, transaction violations.
XMPU Regional Controls			
0x0100+	R{00:15}_START	16	Region starting base address.
0x0104+	R{00:15}_END	16	Region ending base address.
0x0108+	R{00:15}_MASTER	16	Region master IDs.
0x010C+	R{00:15}_CONFIG	16	Region profile: enable, read/write allowed, secure level, relaxed checking.
XMPU Sink for FPD XMPU			
0xFD4F_FF00	ERR_STATUS	1	R/W type and offset address access violations.
0xFD4F_FFEC	ERR_CTRL	1	PSLVERR signaling enable.
0xFD4F_FF10+	ISR, IMR, IER, IDR	4	Interrupt controls: register address decode error.

XPPU Protection of Slaves

The XPPU is used to protect LPD peripheral and control registers (SLCR) along with message buffers (IPI) rather than memory (DDR, OCM). Controlled access to these registers helps achieve security, safety, and operating system isolation. The XPPU is located in the LPD to protect the IOP from erroneous read and write transactions. The XPPU is shown in the IOP interconnect in Figure 16-1.

Two data structures are used by the XPPU to control access.

- The master ID list (part of the register set shown in Figure 16-4), is partially user programmable to allow the enumeration of the masters that are allowed to access peripherals. The list defines a pool of potential masters. Out of 20 master IDs to be programmed in the list, the first eight master ID entries on the list are predefined and the rest can be defined and allocated by user software. The master ID list should be initialized before the XPPU is enabled.
- The aperture permission list defines the set of accessible address apertures (where apertures refer to the peripheral IP address space) and identifies the masters that can access each aperture. The XPPU includes 400 apertures. As shown in Figure 16-4, a RAM is used to store the permission settings set up by the software. This RAM is on the system address map and is accessible like regular software programmable registers.

Note: The XPPU must be programmed once before being used and should only be enabled when there are no transactions going through it to avoid misbehavior. Transactions can be generated from several sources including another system master or a CSU event.

The master ID list and the aperture permission list provides access control for all peripheral apertures. The apertures can be made accessible or hidden from any master ID.

Features

- Provides access control for a specified set of address apertures on a per-master basis.
- Provides a means of controlling access on a per-peripheral or a per-message buffer basis.
- Supports up to 20 simultaneous sets of masters.
- Several sets of programmable apertures:
  - 128 x 32B for IPI message buffers.
  - 256 x 64 KB for peripheral slave ports.
  - 16 x 1 MB for peripheral slave ports.
  - Single 512 MB for Quad-SPI memory controller.
- AXI transaction permission violation interrupt.

- APB slave interface address decode error interrupt.

When an AXI transaction is not permitted to proceed, the protection unit sets the poison user bit and forwards the transaction to the XPPU_SINK where an interrupt is generated and the unit responds with or without a ready or SLVERR.

The XPPU is a look-up based peripheral protection unit. The XPPU is for protecting peripherals, message buffers (for inter-processor interrupts and communications), and Quad-SPI flash memory. The XPPU also has a mechanism to protect the access of its own programming registers.

In comparison with the XMPU, the XPPU uses finer grained address matching and provides many more address apertures to suit the different needs of peripherals, IPI, and Quad-SPI flash memory.

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.
- APB slave for programming the XPPU.
- Level-sensitive, asynchronous interrupt output.
- AXI clock (same for master and slave ports).
- APB bus clock for programming registers.

Instances

The locations of the various system protection modules in the PS are shown in Figure 16-1 as XMPU, XPPU, and TBUx. The ATB timeout and AIB isolation units are also shown in Figure 16-1.

The one instance of the XPPU is in the LPD. The XMPU is placed in the following locations.

- Six instances on the DDRC with a 1 MB address alignment.
- One instance on the OCM interconnect with a 4 KB address alignment.
- One instance on the FPD interconnect with a 4 KB address alignment.

Note: The XMPU instances near the DDR memory have a 1 MB region alignment. The XMPU instance for the OCM and the FPD peripherals has a 4 KB alignment. These configurations are fixed and cannot be changed.

XPPU Operation

For every read and write transaction, the XPPU determines if the transaction is allowed to proceed with fine grain control of specific memory addresses. If the transaction is allowed, it proceeds normally. If the transaction is not allowed, it invalidates the transaction by address *poisoning*. When an address is poisoned, the transaction is sent to the XPPU_Sink unit.

An AXI transaction request is allowed to access the memory range defined by an APERPERM_xxx register if three 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.*

A block-level diagram summarizing the XPPU operation is shown in Figure 16-4.

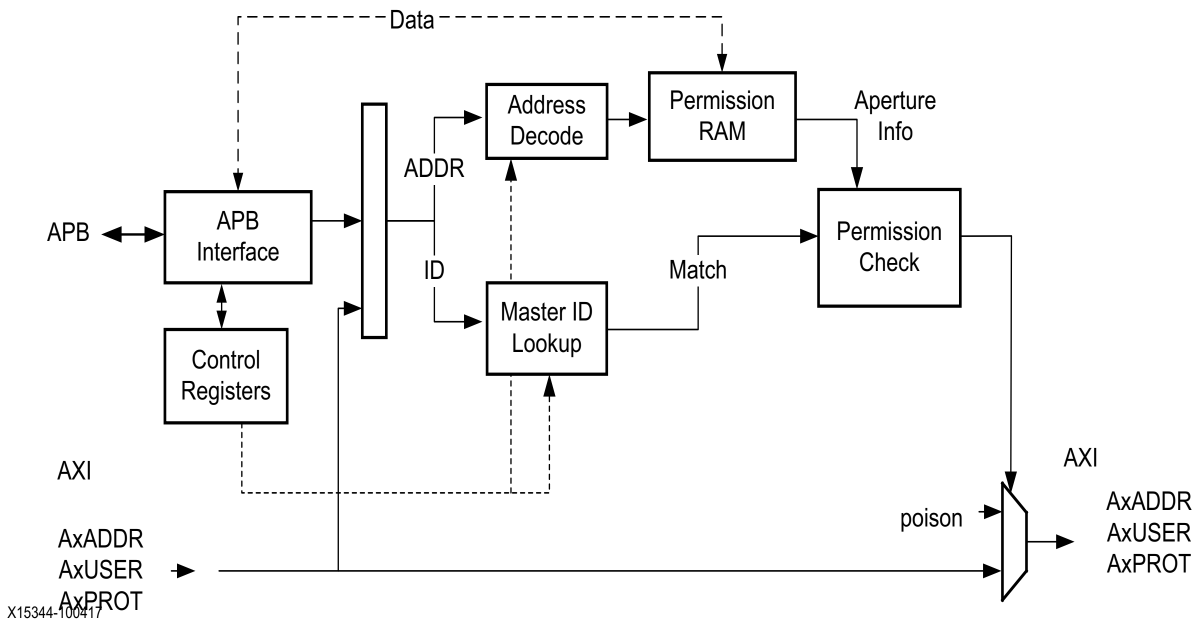


Figure 16-4: **XPPU Functional Block Diagram**

Master ID List

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 nth 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 Table 16-9.

Table 16-9: **Master ID List Entry**

Name	Bit Field	Bitfield	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 16-11) 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.

Aperture Permission List

Table 16-10 shows the four sets of apertures and the address protected for each aperture.

Table 16-10: **XPPU Address Table**

Aperture Size	Number of Supported Apertures	Aperture Number	Protected Memory Address Range	Aperture Configuration Register Address
32B	128	256	`0xFF99_0000 - 0xFF99_001F`	`0xFF98_1400`
		257	`0xFF99_0020 - 0xFF99_003F`	`0xFF98_1404`
		...	...	
		383	`0xFF99_0FE0 - 0xFF99_0FFF`	`0xFF98_15FC`
64 KB	256	000	`0xFF00_0000 - 0xFF00_FFFF`	`0xFF98_1000`
		001	`0xFF01_0000 - 0xFF01_FFFF`	`0xFF98_1004`
		...	...	
		255	`0xFFFF_0000 - 0xFFFF_FFFF`	`0xFF98_13FC`
1 MB	16	384	`0xFE00_0000 - 0xFE0F_FFFF`	`0xFF98_1600`
		385	`0xFE10_0000 - 0xFF1F_FFFF`	`0xFF98_1604`
		...	....	
		399	`0xFEF0_0000 - 0xFEFF_FFFF`	`0xFF98_163C`
512 MB	1	400	`0xC000_0000 - 0xDFFF_FFFF`	`0xFF98_1640`

IMPORTANT: *The 32B IPI buffers can be accessed only with burst length = 1 and burst size = 4B or 8B transactions. The 32B aperture is used to access IPI message buffers. The 64 KB aperture is for peripherals. For example,* `0xFF00_0000` *is the address for UART0 registers for which the aperture permission list entry is at address* `0xFF98_1000`.

The overlapping address range between IPI buffers and 64 KB peripheral registers is resolved as follows:

- For the range `0xFF99_0000` – `0xFF99_0FFF` (the first 4 KB), the 32B APL entry takes precedence over the 64 KB APL entries. The 64 KB APL is ignored here.
- For the range `0xFF99_1000` – `0xFF99_FFFF` (the remaining 60 KB), the PSLVERR signal is returned because no registers exist. This is independent of the 64 KB aperture allowing access to this area.

Entry Format

Aperture Permission List

The XPPU aperture register structure enumerates the permission settings on each protected peripheral, message buffer, and the Quad-SPI flash memory. Each APERPERM_{000:400} register entry contains the information listed in Table 16-11.

Table 16-11: **Aperture Permissions Register Format**

Field Name	Bitfield	Description
PERMISSION	[19:0]	Master ID profile permission. Each of the 20 [PERMISSION] bits correspond to the MASTER_ID{19:0} 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]	`1` = Secure or non-secure transactions are allowed. `0` = Only 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 Figure 16-5.

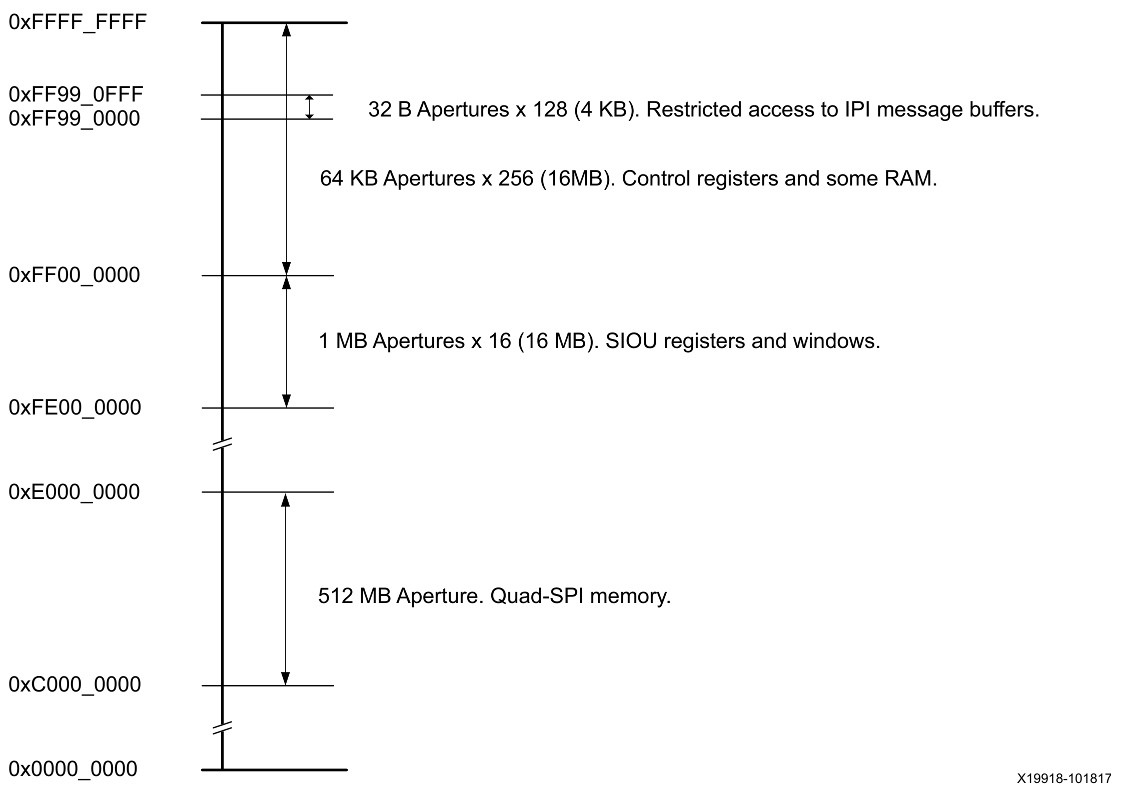

0xFFFF_FFFF
0xFF99_0FFF
0xFF99_0000
32 B Apertures x 128 (4 KB). Restricted access to IPI message buffers.
64 KB Apertures x 256 (16MB). Control registers and some RAM.
0xFF00_0000
1 MB Apertures x 16 (16 MB). SIOU registers and windows.
0xFE00_0000
0xE000_0000
512 MB Aperture. Quad-SPI memory.
0xC000_0000
0x0000_0000
X19918-101817

Figure 16-5: **XPPU Aperture Memory Map**

Permission Checking

Permission checking is performed using the AXI master ID and TZ security settings of the AXI 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.

Transaction_OK = (MATCH & 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, then the transaction is allowed.

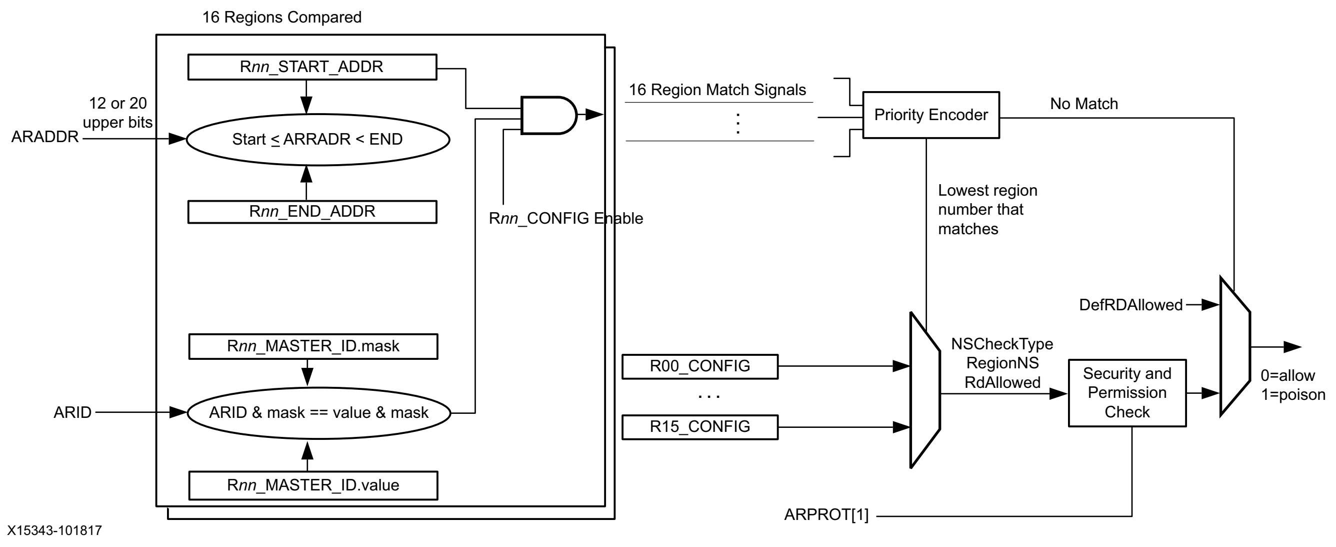


Figure 16-6: **XPPU Functional Block Diagram**

Error Handling

Table 16-12 lists the possible errors that can be encountered by the XPPU and how they are handled.

Table 16-12: **Error Handling in XPPU**(1)

Error	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 is poisoned 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 is poisoned. 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.

Sync and Async Abort

To understand the sync and async abort, consider the example application running in the APU domain that generates two types of interrupts: sync and error aborts. The actual error type depends on the corresponding transaction type (read or write) and each must have their own handler. The requirements for two types of errors comes from the Arm architecture itself, not the Xilinx-specific implementation. Handling these errors is up to the developer and the requirements of the system being developed. See *Isolation Methods in Zynq UltraScale+ MPSoCs* (XAPP1320) [Ref 38] for more information.

Transaction Poisoning

Transaction poisoning can be accomplished by the following.

- Force the outgoing AxADDR [48:32] to zeros.
- Replace the incoming AxADDR[31:12] with LPD_XPPU.POISON[BASE].
- Keep the incoming AxADDR[11:0] intact.

On the system address map, a 4 KB size sink module at address {`17'b0`, LPD_XPPU.POISON[BASE] `12'b0`} is present, which takes a poisoned transaction, returns an error response, and optionally records the lower 12 bits of the transaction address. This can cause either a data abort or an interrupt to the processor.

Note: The poisoned reads return value of all zeros and poisoned writes are ignored. To make the system aware of transaction poisoning, generation of a slave error response can be enabled using the XPPU_SINK.ERR_CTRL[PSLVERR] bit.

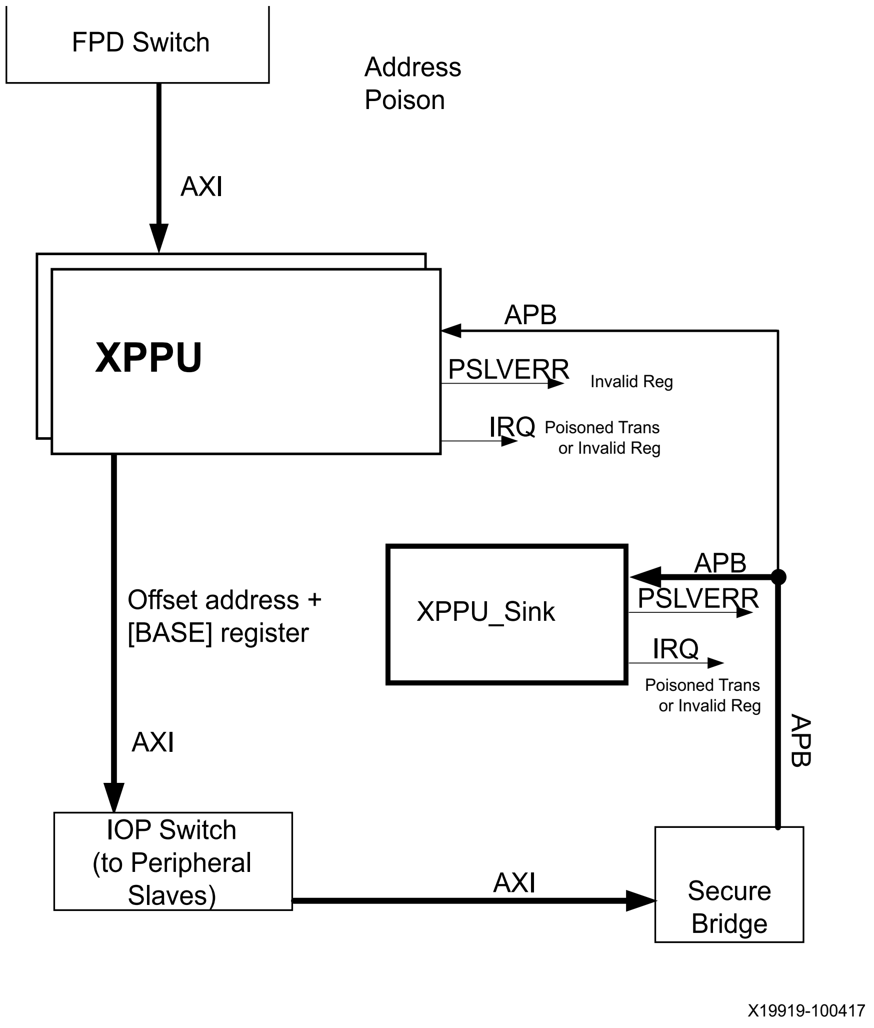

FPD Switch
Address Poison
AXI
XPPU
APB
PSLVERR
Invalid Reg
IRQ
Poisoned Trans or Invalid Reg
Offset address + [BASE] register
XPPU_Sink
APB
PSLVERR
IRQ
Poisoned Trans or Invalid Reg
APB
AXI
IOP Switch (to Peripheral Slaves)
AXI
Secure Bridge
X19919-100417

Figure 16-7: **XPPU Address Poison Block Diagram**

XPPU Self-Protection

The following measures are used to protect the XPPU itself.

- The master ID list and the aperture permission list can only be written and read by secure masters.
- The master ID list and the aperture permission list can be locked from further writes (by secure masters) through the same protection mechanism as for other peripherals (that is, setting appropriate entries in the master ID list and aperture permission list).

RECOMMENDED: *Xilinx recommends setting up the XPPU one time during boot by the FSBL or other secure agent. Programming the XPPU requires access through the XPPU.*

The XPPU programming interface uses the XPPU itself for protection, an access destined to a location in the XPPU (including master ID and aperture registers) *visits* the XPPU twice.

- The first *visit* comes from AXI, and passes through the protection logic to the AXI interconnect and on to the APB bridge.
- The second *visit* comes from the APB, and arrives at the addressed location, assuming its first visit passed the XPPU permission check.

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 Equation 16-2 is satisfied (along with [TRUSTZONE] and [PERMISSION] checks in the APERPERM_xxx register), then the transaction is allowed. In Equation 16-2, these are [10-bit parameters] in the MASTER_IDxx register:

$$[MID] \ \& \ [MIDM] == AXI\_MasterID \ \& \ [MIDM] \qquad \text{Equation 16-2}$$

Master IDs List

The PS interconnect assigns the master ID bits and transfers these bits on the AxUSER bits of the associated AXI transaction. For masters that support multiple channels or sources, a portion of the master ID bits are derived from the AXI ID (AWID/ARID) of the associated AXI transaction. This allows the user to enforce system-level protection on a per channel, per processor, or per PL IP basis. In the context of the SMMU, the description of a master ID and a stream ID have the same meaning.

Note: The PS interconnect assigns the master ID bits and transfers these bits on the AxUSER bits of the associated AXI transaction.

Table 16-13: **Master IDs List**

Master Device	Master ID [9:0]
RPU0	0000, 00, AXI ID[3:0]
RPU1	0000, 01, AXI ID[3:0]
PMU processor	0001, 00, 0000
CSU processor	0001, 01, 0000
CSU DMA	0001, 01, 0001
USB0	0001, 10, 0000
USB1	0001, 10, 0001
DAP APB control	0001, 10, 0010
LPD DMA	0001, 10, 1xxx CH{0:7}
SD0	0001, 11, 0000
SD1	0001, 11, 0001
NAND	0001, 11, 0010
QSPI	0001, 11, 0011
GEM0	0001, 11, 0100
GEM1	0001, 11, 0101
GEM2	0001, 11, 0110
GEM3	0001, 11, 0111
SMMU TCU	0000, 00, 0000
CCI-400	0000, 00, 0000
APU	APU 0010, AXI ID [5:0]
SATA	0011, 00, 000x DMA{0, 1}
GPU	0011, 00, 0100
DAP AXI CoreSight	0011, 00, 0101
PCIe	0011, 01, 0000
DisplayPort DMA	0011, 10, 0xxx DMA{0:5}
FPD DMA	0011, 10, 1xxx CH{0:7}
S_AXI_HPC0_FPD (HPC0)	1000, AXI ID [5:0] from PL
S_AXI_HPC1_FPD (HPC1)	1001, AXI ID [5:0] from PL
S_AXI_HP0_FPD (HP0)	1010, AXI ID [5:0] from PL
S_AXI_HP1_FPD (HP1)	1011, AXI ID [5:0] from PL
S_AXI_HP2_FPD (HP2)	1100, AXI ID [5:0] from PL
S_AXI_HP3_FPD (HP3)	1101, AXI ID [5:0] from PL
S_AXI_LPD (PL_LPD)	1110, AXI ID [5:0] from PL

Table 16-13: **Master IDs List *(Cont'd)***

Master Device	Master ID [9:0]
S_AXI_ACE_FPD (ACE)	1111, AXI ID [5:0] from PL

Notes:

1. Each R5 CPU can be identified by its Master ID. However, there is only one Master ID for the A53 cluster of CPUs.

PS-PL AXI Interfaces

The PL is required to drive all 10 bits of the ACE port's master ID. The AXI_USER is the master ID for ACE.

AXI_USER[9:0] = {`4'b1111`, AXI_ID[5:0]}

The upper four bits are required to be driven High by the logic in the PL.

XPPU Register Set Overview

Table 16-14: **XPPU Register Summary**

Start Address	Register Names	Number of Registers	Description
XPPU Control and Status			
0xFF98_0000	CTRL	1	Permission and parity error checking enables.
0xFF98_0004+	ERR_STATUS{1, 2}	2	Poisoned address and Master ID value.
0xFF98_000C	POISON	1	Base address of XPPU sink.
0xFF98_0010+	ISR, IMR, IEN, IDS	4	Interrupt controls: register address decode error, transaction violations, parity errors.
0xFF98_003C	M_MASTER_IDS	1	Number of Master IDs configured.
0xFF98_0040+	M_APERTURE_{32 B, 64 KB, 1 MB, 512 MB}	4	Apertures for IPI, IOP CSRs, Memory, and Quad-SPI.
0xFF98_0050+	BASE_{32 B, 64 KB, 1 MB, 512 MB}	4	Base address for each aperture start address (read-only).
XPPU Aperture Controls			
0xFF98_0100+	MASTER_ID{00:19}	20	Master ID profiles.
0xFF98_1000 - 0xFF98_13FF	APERPERM_{000:255}	256	IOP, 64-KB pages.
0xFF98_1400 - 0xFF98_15FF	APERPERM_{256:383}	128	IPI, 32-B pages.
0xFF98_1600 - 0xFF98_163F	APERPERM_{384:399}	16	IOP memory, 64-KB pages.
0xFF98_1640	APERPERM_400	1	Quad-SPI memory, 512 MB.
XPPU Sink Control and Status			
0xFF9C_FF00	ERR_STATUS	1	R/W type and Offset address access violations.

Table 16-14: **XPPU Register Summary *(Cont'd)***

Start Address	Register Names	Number of Registers	Description
0xFF9C_FFEC	ERR_CTRL	1	PSLVERR signaling enable.
0xFF9C_FF10+	ISR, IMR, IER, IDR	4	Interrupt controls: register address decode error.

Lock Unused Memory Attribute

Enabling this option restricts the use of all memory locations (memory and peripherals) not explicitly defined by the isolation setup. In a secure system this would typically be checked, however, there are operations in a system that are not always clear and could be impacted by this parameter. For example, when loading authenticated and/or encrypted images after boot (a partial bit file), the system must access the CSU, eFuse, and potentially BBRAM register space. If these are not explicitly added to the secure subsystem performing this action, it will be blocked by the lock unused memory option. For more details see *Isolation Methods in ZynqUltraScale+MPSoCs* (XAPP1320) [Ref 38].

Programming Example

This programming example includes the XPPU and XMPU programming steps for the RPU and APU to permit secure read and write access to several system elements in the following system configuration:

- The APU is the master of these system elements:
    - DDR memory space: DDR XMPU, first 1 GB of memory.
    - GEM0 control registers: XPPU aperture.
    - SATA AHCI registers: FPD XMPU, a 64 KB region.
- The RPU is the master for these system elements:
    - OCM memory space: OCM XMPU, first 64 KB of memory.
    - I2C0 control registers: XPPU aperture.

Note: The memory regions can be configured for read-only/write-only or non-secure access based configuration parameters.

Note: The Vivado Design Suite generates first stage bootloader (FSBL) code to program the XMPU and XPPU based on the design defined by the processor configuration wizard (PCW).

The XMPU and XPPU register sets are listed in Table 16-8 and Table 16-14.

Use Cases

This section describes how each protection unit is used for this programming example.

XMPU DDR Protection Units: The APU transactions are routed through the CCI and to access the DDR memory via the DDR XMPU{1, 2} protection units. Other system masters can also use this path, but they are blocked by the protection units using AXI Master ID filtering. In this example system, only an APU core is allowed to access the DDR memory controller.

The DDR XMPU{0, 3, 4, 5} units are disabled; the RPU, PL masters, and others are not permitted to access DDR memory.

XMPU FPD Protection Unit: APU transactions are routed through the FPD XMPU protection unit to reach the SATA AHCI memory-mapped registers in the SIOU.

XMPU OCM Protection Unit: RPU transactions are routed through the OCM XMPU protection unit to reach the OCM memory.

XPPU Protection Unit: The XPPU protection unit permits access to the I2C and GEM registers.

Program the DDR XMPUs

There are six DDR, one FPD, and one OCM memory protection units. Four of the DDR XMPUs are not used in this programming example (all of the other XMPUs are used).

Disable the unused DDR XMPU units by setting the [DefWrAllowed] and [DefRdAllowed] bit fields to “not allowed” and leave the region registers R{00:15}_{START, END, MASTER, CONFIG} in their reset state.

- Write `0h` to the DDR_XMPUx_CFG.CTRL registers (units 0, 3, 4, and 5).

Program two DDR XMPU units (1, 2} for the two parallel AXI channels from the CCI to the DDR memory controller. For this example, they are programmed in the same manner.

1. Disallow default accesses for all regions. Write `8h` to the DDR_XMPUx_CFG.CTRL registers.
2. Program a set of region configuration registers for secure reads and writes to the first GB of DDR memory by any of the APU cores.
   a. Write `0007h` to the DDR_XMPUx_CFG.R00_CONFIG register for ports 1 and 2.
   b. Write `0000h` to the DDR_XMPUx_CFG.R00_START register for ports 1 and 2.
   c. Write `03FFh` to the DDR_XMPUx_CFG.R00_END register. The memory region for the DDR XMPU units is 1-MB aligned so bits [19:0] are always `0h` and bits [39:20] are programmed. The resulting end address is 0x0_3FF0_0000 plus the block size. The result is 0x0_3FFF_FFFF for a total of 1 GB.

d. Write `00C0_0080h` to the DDR_XMPUx_CFG.R00_MASTER register. To allow only the APU cores to access the DDR memory, the [MASK] bit field is set to `C0h` and the [ID] bit field is set to `80h`. See Table 16-9 for the list of Master ID numbers and equation Equation 16-1 for the comparison testing done by the controller.

Note: Additional, high-order Master ID bits could be tested, but that is unnecessary for these DDR protection units because no other master with similar ID bits has access to the DDR XMPU{1, 2} units.

Note: To enable additional masters to access the DDR memory region via the CCI, including the DMA units in the GEM and PCIe controllers, then program additional sets of region registers using their Master IDs and the desired memory range.

Program the FPD XMPU

Program the FPD XMPU so that it only allows the APU to access the SATA AHCI registers with secure reads and writes in a 64-KB memory region.

1. Disallow default accesses for all regions. Write 0h to the FPD_XMPU_CFG.CTRL register.
2. Program a set of region registers.
   a. Write 0007h to the FPD_XMPU_CFG.R00_CONFIG register. If strict secure/non-secure checking is desired, write 0017h instead.
   b. Write `0F_D0C0h` to the FPD_XMPU_CFG.R00_START register. The memory region for the FPD XMPU unit is 4-KB aligned so bits [11:0] are always `0h` and address bits [31:12] are programmed. The resulting start address is 0xFD0C_0000; the start of the OCM memory.
   c. Write `0F_D0CFh` to the FPD_XMPU_CFG.R00_END register. The end address is 0xFD0C_F000 plus the last block. The result is 0xFD0C_FFFF.
   d. Write `02C0_0080h` to the FPD_XMPU_CFG.R00_MASTER register. To allow only the APU cores to access the SATA AHCI registers, the [MASK] bit field is set to `2C0h` and the [ID] bit field is set to `080h`. Refer to Table 16-9 for the list of Master ID numbers and Equation 16-1 for the comparison testing done by the controller.

Note: These [ID] and [MASK] bit field settings are more selective than the DDR XMPU settings because the DDR XMPUs have additional AXI masters with access to the AXI channels protected by these protection units.

Program the OCM XMPU

Program the OCM XMPU so that it allows the RPU to access the first 64 KB of the OCM memory region with secure read and write transactions.

1. Disallow default accesses for all regions. Write 0h to the OCM_XMPU_CFG.CTRL register.
2. Program a set of region registers.
   a. Write `0007h` to the OCM_XMPU_CFG.R00_CONFIG register. If strict secure/non-secure checking is desired, write `0017h` instead.

b. Write `0F_FFC0h` to the OCM_XMPU_CFG.R00_START register. The memory region for the FPD XMPU unit is 4-KB aligned so bits [11:0] are always `0h` and address bits [31:12] are programmed. The resulting start address is 0xFFFC_0000; the start of the OCM memory.

c. Write 0F_FFCFh to the OCM_XMPU_CFG.R00_END register. The end address is 0xFD0C_F000 plus the last block; the result is 0xFD0C_FFFF.

d. Write `02C0_0080h` to the OCM_XMPU_CFG.R00_MASTER register. To allow only the APU cores to access the SATA AHCI registers, the [MASK] bit field is set to `2C0h` and the [ID] bit field is set to `080h`. Refer to Table 16-9 for the list of Master ID numbers and Equation 16-1 for the comparison testing done by the controller.

Program the XPPU

This example configures two AXI masters and two 64 KB apertures in the XPPU.

- RPU0 permitted to access the I2C0 controller.
- APU permitted to access the GEM registers.

This is a two-step process that establishes two masters using the MASTER_IDxx registers and two 64-KB aperture registers.

Configure two masters: the master ID [MID] and mask [MIDM] bit fields identify the master using Equation 16-1.

1. Configure the APU as master 0. Write 02C0_0080 to the MASTER_ID00 register.
2. Configure the RPU0 as master 1. Write 02C0_0000 to the MASTER_ID01 register.

Note: The first eight MASTER_IDxx registers are predefined by reset, but they can be overwritten and configured for any master.

Program two 64-KB Apertures: one each for access to the GEM0 and I2C0 registers.

The GEM0 registers are mapped to 0xFF0B_0000. Access is controlled by the APERPERM_011 register at 0xFF98_102C.

- Configure aperture 11 for APU access. Write `0_0001h` (enable bit for master 0) to the APERPERM_011 [PERMISSION] bit field.

The I2C0 registers are mapped to 0xFF02_0000. Access is controlled by the APERPERM_002 register at 0xFF98_1008.

- Configure aperture 2 for RPU0 access. Write 0_0002h (enable bit for master 1) to the APERPERM_002 [PERMISSION] bit field.

Note: The aperture registers include two other fields. The [TRUSTZONE] bit would be set to 0 in this example to ensure only secure transactions are allowed and the [PARITY] bit needs to be calculated and written to ensure data integrity.

Enable XPPU permission and parity checking: the XPPU control register includes permission and parity checking enables.

1. Enable the XPPU unit. Write a 1 to the CTRL [ENABLE] bit.
2. Optionally enable parity for the master configuration registers {00:19}. Write a 1 to the CTRL [MID_PARITY_EN] bit.
3. Optionally enable parity for the aperture registers {000:400}. Write a 1 to the CTRL [APER_PARITY_EN] bit.

Note: Parity errors are signaled by the ISR [MID_PARITY] and [APER_PARITY] status bits.

Write-Protected Registers Table

CRF APB Registers

The CRF_APB write-protected registers are listed Table 16-15. The protection is controlled by the CRF_APB.CRF_WPROT [active] bit.

Table 16-15: **Write-protected Registers, CRF_APB**

Registers	Count	Registers	Count	Registers	Count
acpu_ctrl	1	dll_ref_ctrl	1	qspi_ref_ctrl	1
adma_ref_ctrl (LPD_DMA)	1	dp_audio_ref_ctrl	1	reset_ctrl	1
ams_ref_ctrl	1	dp_stc_ref_ctrl	1	reset_reason	1
apll_{cfg,ctrl}	2	dp_video_ref_ctrl	1	rpll_{cfg,ctrl}	2
apll_frac_cfg	1	dpll_cfg	1	rpll_frac_cfg	1
apll_to_lpd_ctrl	1	dpll_frac_cfg	1	rpll_to_fpd_ctrl	1
bank3_ctrl{0:5}	6	dpll_to_lpd_ctrl	1	rst_ddr_ss	1
bank3_status	1	gdma_ref_ctrl (FPD_DMA)	1	rst_fpd_apu	1
blockonly_rst	1	gem_tsu_ref_ctrl	1	rst_fpd_top	1
boot_mode_{por,user}	2	gem{0:3}_ref_ctrl	4	rst_lpd_dbg	1
can{0,1}_ref_ctrl	2	gpu_ref_ctrl	1	rst_lpd_iou{0,2}	2
chkr{0:7}_clka_lower	8	i2c{0,1}_ref_ctrl	2	rst_lpd_top	1
chkr{0:7}_clka_upper	8	acpu_ctrl	1	sata_ref_ctrl	1
chkr{0:7}_clkb_cnt	8	iopll_{cfg,ctrl}	2	sdio{0,1}_ref_ctrl	2
chkr{0:7}_ctrl	8	iopll_frac_cfg	1	spi{0,1}_ref_ctrl	2
clkmon_{disable,enable}	2	iopll_to_fpd_ctrl	1	timestamp_ref_ctrl	1
clkmon_{mask,status}	2	iou_switch_ctrl	1	topsw_lsbus_ctrl	1
clkmon_trigger	1	lpd_lsbus_ctrl	1	topsw_main_ctrl	1

Table 16-15: **Write-protected Registers, CRF_APB *(Cont'd)***

Registers	Count	Registers	Count	Registers	Count
cpu_r5_ctrl	1	lpd_switch_ctrl	1	uart{0,1}_ref_ctrl	2
csu_pll_ctrl	1	nand_ref_ctrl	1	usb{0,1}_bus_ref_ctrl	2
dbg_fpd_ctrl	1	pcap_ctrl	1	usb3_dual_ref_ctrl	1
dbg_lpd_ctrl	1	pcie_ref_ctrl	1	vpll_{cfg,ctrl}	2
dbg_trace_ctrl	1	pl{0:3}_ref_ctrl	4	vpll_frac_cfg	1
dbg_tstmp_ctrl	1	pl{0:3}_thr_ctrl	4	vpll_to_lpd_ctrl	1
ddr_ctrl	1	pll_status	1		

Other Write-Protected Registers

The remaining write-protected registers are listed in Table 16-16. The protection is controlled by the lock bit shown in Table 16-16.

Table 16-16: **Write-Protected Registers, Others**

Register Set	Registers	Count	Lock Register Bit
DDR_XMPU{0:5}_CFG	ctrl, poison, err_status{1,2}	24	LOCK [RegWrDis] All registers except interrupt registers.
	r{00:15}_config, master	192	
	r{00:15}_start, end	192	
FPD_XMPU_CFG	ctrl, poison, err_status{1,2}	4	LOCK [RegWrDis] All registers except interrupt registers.
	r{00:15}_config, master	32	
	r{00:15}_start, end	32	
OCM_XMPU_CFG	ctrl, poison, err_status{1,2}	4	LOCK [RegWrDis] All registers except interrupt registers.
	r{00:15}_config, master	32	
	r{00:15}_start, end	32	
VCU_SLCR	alg_{dec,enc}_core_ctrl	2	CRL_WPROT [ACTIVE] Write lock for several registers.
	alg_{dec,enc}_mcu_ctrl	2	
	alg_vcu_axi_ctrl, pll_status	2	
	vcu_pll_{cfg,ctrl}, vcu_pll_frac_cfg	2	
EFUSE	efuse_cache_load	1	WR_LOCK [LOCK] Write lock for several registers.
	efuse_{rd, pgm}_addr	2	
	cfg, tpgm, trd	3	
	tsu_h_{cs, ps, ps_cs}	3	

Security and Safety Errors

Errors in a system can be classified into security or safety errors. An error could be safety critical and/or security critical.

Security Error

The result of a security critical error can expose security assets. When this type of error occurs, the system needs to be locked down. Some examples include PS SYSMON alarms for voltage or temperature, and an indication of a single-event upset (SEU).

Safety Error

A correctable or catastrophic error is categorized as a safety error. As a result of a catastrophic safety error, the system needs to be reset to a safe state.

A safety-compliant system is required to detect and react to an error in less than 10 ms. It is also required to put itself into a safe state as a result of an error.

A safe state is one where the following occurs.

- Error manager is informed.
- Failure is isolated.
- If possible, high-level software (safety operating system) gets an indication of the error.
- Indicate error to outside world.
- Store error source and context for diagnostic purpose.

The hardware error status is sent to the platform management unit (PMU) as interrupts. Based on the error source, the user-programmable software on the PMU should determine the type of reset (PS-only reset, full-power domain reset, or RPU reset).

For some of the errors (e.g., a power failure that is both security and safety related) the action can be configured in the configuration security unit (CSU) for a security lockdown. However, you should only configure one or the other (depending upon the specific system requirement). Various enable registers for power-on reset (POR), system reset, and PS error are provided in the PMU_GLOBAL register set, which can be configured to trigger the respective action.

The PMU is responsible for capturing all errors within the device, reporting these errors to the outside world, and taking the appropriate action with respect to each error. The PMU includes the necessary registers, logic, and interfaces for handling these functions.

Refer to Chapter 6, Platform Management Unit for further details on error handling and reporting.

AIB Isolation Functionality

The AMBA interconnect has AXI and APB isolation blocks (AIBs) that are responsible for functionally isolating the AXI/APB master from the slave in preparation for an AXI/APB master or slave to be powered down. The AIB manages AXI and APB interfaces during the isolation process resulting in a graceful transition to a power-down state.

The AIB is transparent and offers zero latency during normal transactions. When isolation is commanded by the PMU, the AIB does not propagate new transactions; it will respond to the master with an SLVERR or ignore the transaction and cause the interconnect to timeout. The AIB response is selected by the lpd_slcr_aib.ISO_AIB{AXI, APB}_TYPE registers. When the isolation command is received, the AIB will allow all posted transactions to complete. When this is done, the AIB will assert an interrupt to indicate that the channel is quiescent and ready for an orderly isolation.

Instances

The AIB instances in the interconnect are shown in Table 16-17.

Table 16-17: **AIB Instances**

Instance Number	Domain	Master Device	Slave Device
1	LPD	RPU0	LPD interconnect
2	LPD	RPU1	LPD interconnect
3	LPD	RPU through LPD interconnect	DDR through FPD interconnect
4	LPD	LPD interconnect	FPD interconnect (all other FPD slaves)
5	LPD	LPD interconnect	RPU0
6	LPD	LPD interconnect	RPU1
7	LPD	LPD interconnect	USB0
8	LPD	LPD interconnect	USB1
9	LPD	LPD interconnect	OCM
10	LPD	LPD interconnect	M_AXI_HPM0_LPD
11	FPD	FPD interconnect	LPD interconnect (except OCM)
12	FPD	FPD interconnect	LPD interconnect (only to OCM)
13	FPD	FPD interconnect	M_AXI_HPM0_FPD
14	FPD	FPD interconnect	M_AXI_HPM1_FPD
15	FPD	FPD interconnect	GPU

Programming

An AXI/APB isolation block programming model when using AXI as a slave is listed in the following steps.

1. Enable the slave response by configuring the ISO_AIBAXI_TYPE (`0xFF413038`) register for a specific slave.

RECOMMENDED: *Xilinx recommends configuring this register during boot time and not changing it later.*

2. Request for the isolation by writing into the ISO_AIBAXI_REQ (`0xFF413030`) register depending on the slave to be isolated.
3. Wait for the acknowledgment by polling the ISO_AIBAXI_ACK (`0xFF413040`) register.

RECOMMENDED: *Xilinx recommends holding the request until there is an acknowledgment.*

DDR Memory Controller

Introduction

Figure 17-1 shows the Zynq® UltraScale+™ MPSoC DDR subsystem placement. It connects to rest of the MPSoC through six AXI data interfaces and one AXI control interface. One of the data paths is connected to the real-time processing unit (RPU) and two to the cache coherent interconnect (CCI-400). Others are multiplexed across the DisplayPort controller, FPD DMA, and the programming logic (PL). Of the six interfaces, five are 128-bits wide and the sixth interface (tied to the RPU) is 64-bits wide.

The DDR subsystem supports DDR3, DDR3L, LPDDR3, DDR4, and LPDDR4. It can accept read and write requests from six application host ports that are connected to the controller using AXI bus interfaces. These requests are queued internally and scheduled for access to DRAM devices. The memory controller issues commands on the DDR PHY interface (DFI) interface to the PHY module that reads and writes data from DRAM.

System Memories

The processor-addressable memories are shown in Figure 17-1 and include the following:

- External DDR DRAM memory.
- Internal OCM memory (LPD), see Chapter 18, On-chip Memory.
- RPU tightly-coupled memory (TCM), see Chapter 4, Real-time Processing Unit.
- PL block UltraRAM memories, see *UltraScale Architecture Memory Resources User Guide* (UG573) [Ref 10].

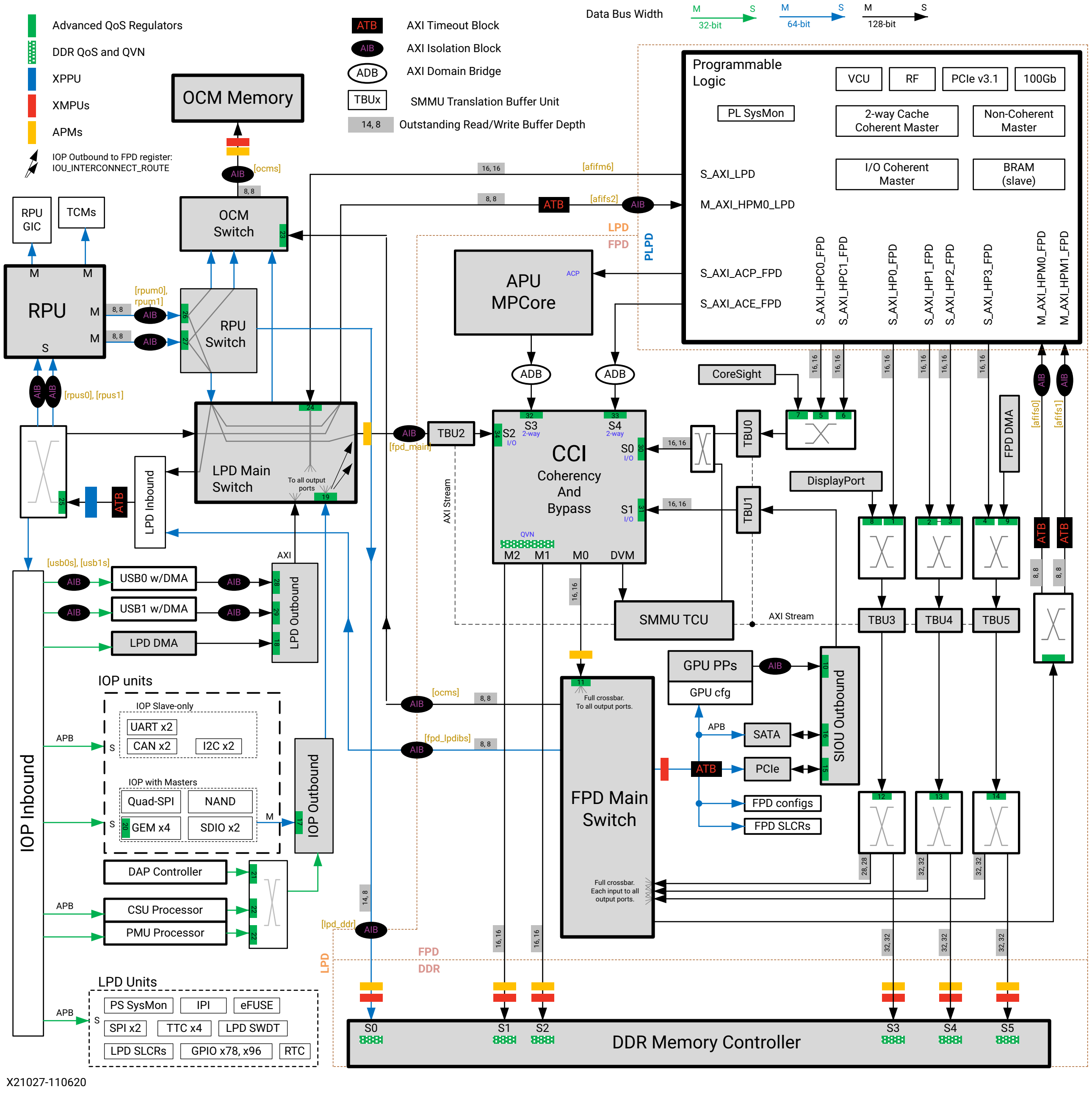


Figure 17-1: **System Memories**

Features

- DDR3, DDR3L, LPDDR3, DDR4, and LPDDR4.
- Support Dynamic DDR configuration of key timings parameter values by reading the SPD on SODIMM/UDIMM/RDIMM parts.
- Dual-rank configurations.
- Dynamic scheduling to optimize bandwidth and latency.
- 64 read and 64 write buffers in fully associative content addressable memories (CAMs).
- Error correction code (ECC) support in 32-bit and 64-bit mode, 2-bit error detection and 1-bit error correction. No ECC support for LPDDR3.
- Programmable quality of service (QoS):
  - Video, isochronous: reads and writes.
  - Low latency: reads.
  - Best effort: reads and writes.
- Delayed writes for optimum performance on SDRAM data bus.
- Out-of-order execution of commands for enhanced SDRAM efficiency.
- Automatic DRAM low-power modes:
  - Entry and exit events based on memory traffic.
  - Power-down, clock-stop, and self-refresh.
  - Clock-stop not supported with RDIMMs.
  - No automatic low-power support for LPDDR3 and LPDDR4.
- Explicit SDRAM mode register updates under software control.
- Highly efficient read-modify-write transactions when byte enables are used with ECC enabled.
- Automatic logging of both correctable and uncorrectable errors.
- Ability to poison the write data by adding uncorrectable errors, for use in testing ECC error handling (ECC poison).
- Responsive to XMPU poisoned AXI transaction.
- Enable 2tCK command timing (2T timing) on the DDR3/DDR4 command/address/control bus signals. This feature can be used as a workaround to address signal quality problems on the CAC bus caused by sub-optimal board layout or power quality issues.

DDR PHY Features

- Complete PHY initialization, training, and control.
- Automatic differential data strobe (DQS) gate training.
- Delay line calibration and voltage threshold (VT) compensation.
- Automatic write leveling.
- Automatic read and write data bit deskew and eye centering.
- Automatic address/command bit deskew and eye centering for LPDDR3.
- Automatic bit deskew and eye centering for LPDDR4.
- Enhanced power saving support.
- PHY control and configuration registers.
- Compatible with the DFI 4.0 PHY interface standard.

DDR Memory Types, Densities, and Data Widths

The DDR memory controller is able to connect to devices under the conditions listed in Table 17-1.

Table 17-1: **DDR Memory Controller Conditions**

Parameter	Value	Notes
Maximum total memory density (GB)	34	This is the maximum supported density.
Total data width (bits)	16, 32, 64	16 and 64-bit LPDDR4 are not supported. 16-bit is supported for DDR4 only.
Component memory density (Gb per die)	0.5, 1, 2, 4, 6, 8, 12, 16	3, 6, 12 and 16Gb single-channel LPDDR4 are not supported. 6, 12, 24 and 32Gb dual-channel LPDDR4 are not supported.
Component data width (bits)	8, 16, 32	4-bit devices not supported. Byte-mode LPDDR4 devices not supported.
Number of ranks	2	
Number of row address bits	17	Limited by the memory controller.
Number of bank address bits	3	
Bank group	2	
MEMC_FREQ_RATIO	2	DDR PHY to controller clock ratio (2:1).

Note: 3DS DDR4 is not supported in PS.

Table 17-2 lists some memory configuration examples. The memory configuration speeds are listed in the *Zynq UltraScale+ MPSoC Data Sheet: DC and AC Switching Characteristics* (DS925) [Ref 2].

Table 17-2: **Example Memory Configurations**

Technology	Configuration	Number of Components	Total Width	Component Density	Capacity	Rank
DDR3 (with ECC)	x8	9	72	4 Gb/8 Gb	4 GB/8 GB	1 and 2
DDR3 (with ECC)	x16	5	72	2 Gb		1
DDR3L (with ECC)	x8	9	72	4 Gb/8 Gb	4 GB/8 GB	1 and 2
DDR3L (with ECC)	x16	5	72	2 Gb		1
LPDDR3	x32	2	64	4Gb		1
DDR4	x8	8	64	8 Gb/16 Gb	8 GB/16 GB	1 and 2
DDR4 (with ECC)	x16	5	72	8 Gb		1
LPDDR4	x32	1	32	8 Gb		1
LPDDR4	x32 with DDP	2	32	16 Gb		2
LPDDR4 (with ECC)	x32	2	40	8 Gb		1

Note: Table 17-2 lists just some of the possible memory configurations. Other configurations are possible.

DDR DRAM Pins

The DDR I/O pins are located on bank 504 and can have a 16-bit, 32-bit or 64-bit data path to the DRAMs depending on the device type. Bytes 0 to 1 correspond to 16-bit data, bytes 0 to 3 correspond to 32-bit data, and bytes 0 to 7 correspond to 64-bit data. Byte 8 refers to the ECC bits. The pins are summarized in Table 17-3. See *Zynq UltraScale+ MPSoC Packaging and Pinout User Guide* (UG1075) [Ref 7] for pin assignments. The pin swap guidelines are described in Answer Record 67330. See *UltraScale Architecture PCB Design User Guide* (UG583)[Ref 15] for clamshell functionality.

Table 17-3: **DDR Pins**

Pin Name	Direction	Description
PS_DDR_DQ	Input/Output	DRAM data.
PS_DDR_DQS_P	Input/Output	DRAM differential data strobe positive.
PS_DDR_DQS_N	Input/Output	DRAM differential data strobe negative.
PS_DDR_ALERT_N	Input	DRAM alert signal.
PS_DDR_ACT_N	Output	DRAM activation command.
PS_DDR_A	Output	DRAM row and column address.
PS_DDR_BA	Output	DRAM bank address.
PS_DDR_BG	Output	DRAM bank group.

Table 17-3: **DDR Pins *(Cont'd)***

Pin Name	Direction	Description
PS_DDR_CK_N	Output	DRAM differential clock negative.
PS_DDR_CK	Output	DRAM differential clock positive.
PS_DDR_CKE	Output	DRAM clock enable.
PS_DDR_CS	Output	DRAM chip select.
PS_DDR_DM	Output	DRAM data mask.
PS_DDR_ODT	Output	DRAM termination control.
PS_DDR_PARITY	Output	DRAM parity signal.
PS_DDR_RAM_RST_N	Output	DRAM reset signal, active Low.
PS_DDR_ZQ	Input/Output	ZQ calibration signal.

Power and Reset

The DDR memory controller is powered by the VCC_PSINTFP_DDR pins. These pins must be connected to the VCC_PSINTFP power pins and the FPD power supply. The DDR memory controller can only be reset along with the FPD using the PMU_GLOBAL.GLOBAL_RESET [FPD_RST] reset bit.

Note: Once the initial calibration sequence in psu_init.c is completed Xilinx-AMD does not recommend using RST_DDR_SS to reset the DDR subsystem.

System Block Diagram

The DDR subsystem (Figure 17-2) consists of six instances of the Xilinx memory protection unit (DDR_XMPU), AXI to APB bridge, AXI performance monitor, DDR controller and DDR PHY.

The XMPU protection unit prevents unauthorized access to restricted areas of the DDR. Both read and write accesses are restricted. The AXI performance monitor provides AXI throughput and latency for all six input ports of the DDR controller. The AXI performance monitor can be accessed by software.

After the request goes through the performance monitors and protection units, the DDR controller establishes a priority for each read, write, and high priority read request based on many factors. The requests are arbitrated and presented to the DDR PHY, and several stages of buffering occurs.

The PHY processes read/write requests from the controller and translates them into specific signals within the timing constraints of the target DDR memory. Signals from the controller are used by the PHY to produce internal signals that connect to the pins through the digital

PHYs. The DDR pins connect directly to the DDR device or devices through the PCB signal traces.

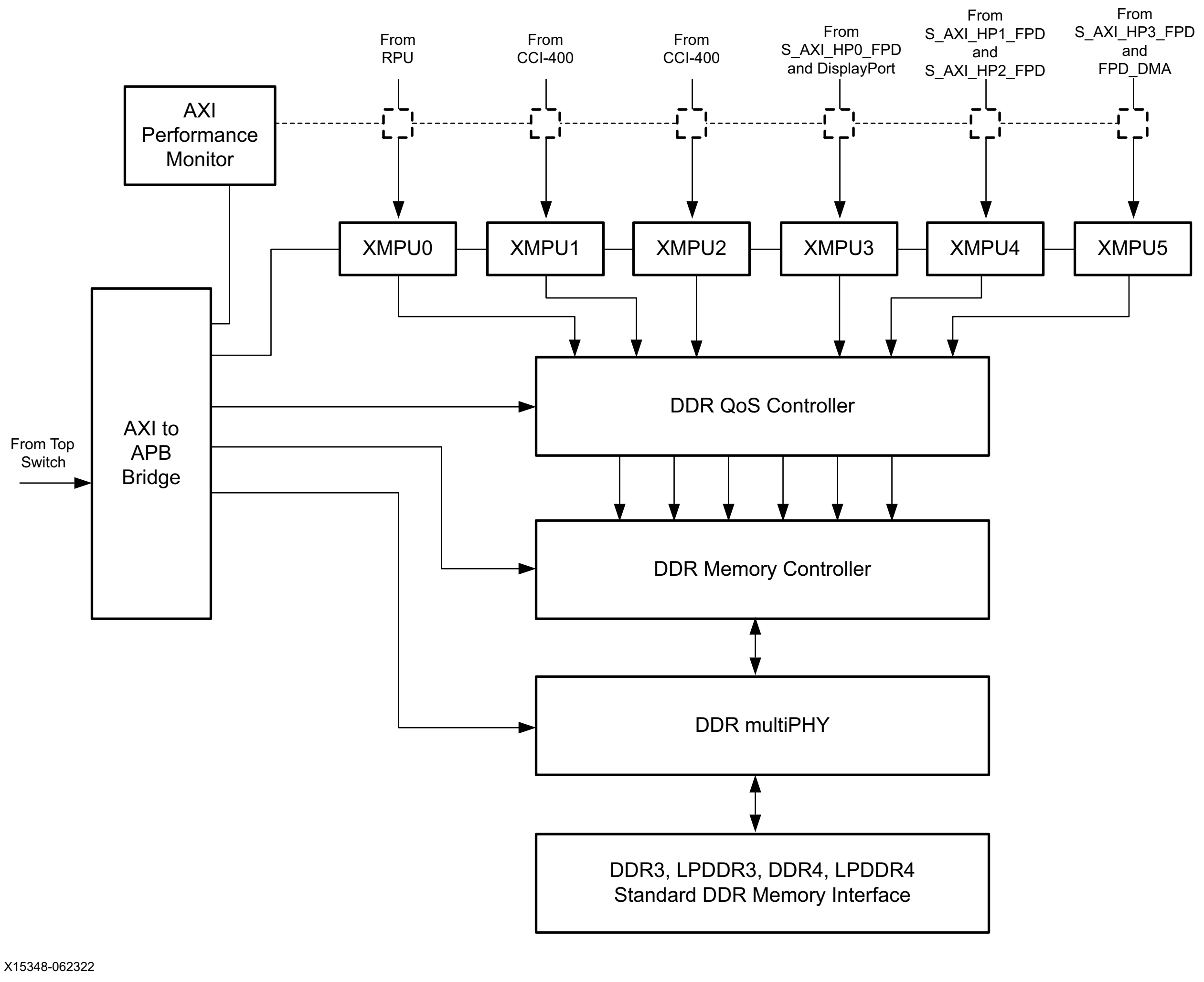


Figure 17-2: **DDR Subsystem Block Diagram**

Xilinx Memory Protection Unit

The XMPU is a region-based memory protection unit. In this chapter, an AXI port interface is referred to as an AXI port. An incoming read or write request on an AXI port in one of the XMPUs is checked against each XMPU region. Any read or write transactions to the DDR regions undergo predefined checks and only when they pass these checks are the transactions allowed. Read and write permissions are independently checked. If the check fails, then the transaction is handled as described in the *XMPU Error Handling* section in the System Protection Units in Chapter 16.

The addresses and master IDs are used for checks. If the address and ID range checks are true, and if the memory region is configured as secure, then only a secure request can access this region. If the transaction is non-secure and the region is configured as secure, the check fails, and the transaction is handled as described in the *XMPU Error Handling* section. If the region is configured as secure, the region's read/write permissions determine if reads or writes are allowed.

If a read (or write) security check passes but the permission check fails, then the XMPU poisons the request, records the address and master ID of the first transaction that failed the check, flags a read (or write) permission violation, and optionally generates an interrupt. For more details, refer to the *XMPU Error Handling* section.

DDR QoS Controller

The DDR memory controller implements the top-level QoS policy for the six ports it supports for DDR access. The QoS is a priority based scheme, where each master in a system can assign a priority value to a transaction request where a servicing node with a choice of more than one transaction selects the transaction with the higher QoS value to process first. The system-level QoS implements two major objectives: Prevention of Head-of-Line Blocking and Traffic Classes.

Prevention of Head-of-Line Blocking

Head-of-line blocking (HOLB) can occur when two or more different traffic classes share the same physical channel. A typical example is when a low-priority request cannot make progress and a high priority request is blocked behind it. HOLB is prevented with the following guidelines.

- Have two ports between the CCI-400 and the DDR controller carry two QoS virtual network channels (QVN) per physical channel.
- On all other DDR controller ports, assign only one class of traffic to each port.

Traffic Classes

The Zynq UltraScale+ MPSoC broadly defines three types of traffic classes.

- Video/isochronous traffic class is a real-time, fixed bandwidth, fixed maximum latency class. Typically, a long latency and low priority is acceptable, but the latency must be bounded in all cases and never exceed a predefined maximum value.
- Low latency (LL) traffic class is a low-latency, high-priority (HPR) class. It is typically assigned the highest memory access priority and can only be surpassed by a video class transaction that has exceeded its threshold maximum latency.
- Best effort (BE) traffic is a high-latency, low-priority (LPR) class used for all other traffic types. This class of traffic is typically assigned the lowest memory access priority.
- DDR reads are prioritized in one of these three classes: best effort (BER), video/isochronous (VPR), and low latency (LL). DDR writes are prioritized in one of these two classes: best effort (BEW) and video/isochronous (VPW).

Further details are described in the Read and Write Priorities section of this chapter.

The QoS controller ensures that there is space available in the DDR controller content addressable memory (CAM) for video class traffic at all times. It achieves this by continuously monitoring the CAM levels and throttling the XPI data port arbiter requests for non-video class traffic when preprogrammed CAM thresholds are exceeded. When the CAM level drops below the predefined threshold value, it sends a control signal to the DDRC to throttle down all BE requests. When the CAM reaches a programmed threshold level, the QoS controller masks the corresponding AXI port/direction from requesting to the port arbiter (PA) inside the DDRC using the pa_rmask and pa_wmask signals. Figure 17-3 shows the functional block diagram of the QoS controller.

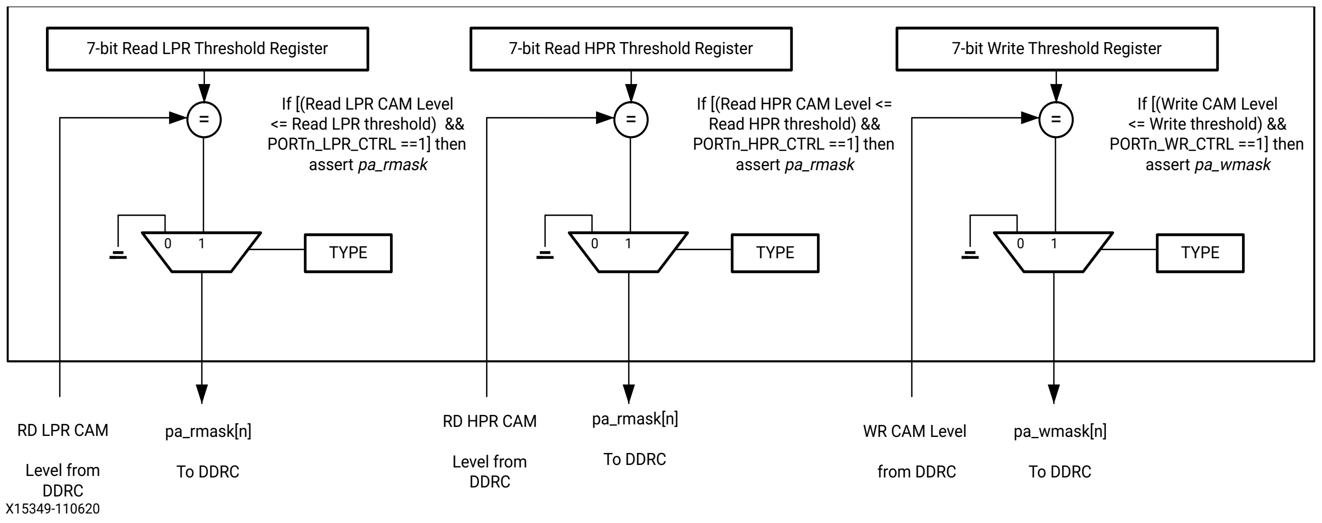

7-bit Read LPR Threshold Register
7-bit Read HPR Threshold Register
7-bit Write Threshold Register
If [(Read LPR CAM Level <= Read LPR threshold) && PORTn_LPR_CTRL ==1] then assert *pa_rmask*
If [(Read HPR CAM Level <= Read HPR threshold) && PORTn_HPR_CTRL ==1] then assert *pa_rmask*
If [(Write CAM Level <= Write threshold) && PORTn_WR_CTRL ==1] then assert *pa_wmask*
=
0
1
TYPE
RD LPR CAM Level from DDRC
pa_rmask[n] To DDRC
RD HPR CAM Level from DDRC
pa_rmask[n] To DDRC
WR CAM Level from DDRC
pa_wmask[n] To DDRC
X15349-110620

Figure 17-3: **QoS Controller Functional Block Diagram**

The QoS controller has a defined set of software-programmable registers per port.

Type Register

The DDR_QOS_CTRL.PORT_TYPE register (2 bits) specifies the traffic class for each of the six ports.

Table 17-4: **Type Register**

Bit Field Value	Description
00	Best effort traffic class.
01	Low-latency traffic class.
10	Video/isochronous traffic class.
11	Reserved

Control Registers

The QoS control register (QOS_CTRL) enables and disables controller operation independently per port for write queue, LPR, and HPR. Each port uses the following control bits that you set (where n equals the number of ports).

Table 17-5: **Control Registers**

Bit Field Name	Description
PORTn_LPR_CTRL	QoS function for low-priority reads (read LPR) channel enable/disable.
PORTn_HPR_CTRL	QoS function for high-priority reads (read HPR) channel enable/disable.
PORTn_WR_CTRL	QoS function for write channel enable/disable.

Threshold Registers

Table 17-6: **Threshold Registers**

Bit Field Name	Description
RD_LPR_THRSLD	Read LPR threshold.
RD_HPR_THRSLD	Read HPR threshold.
WR_THRSLD	Write threshold.

When the DDRC CAM levels reaches their threshold, the QoS controller applies the following throttling rules.

Table 17-7: **Throttling Rules**

Rule	Description
Read channel throttle rules	If ((slots available in read LPR CAM ≤ read LPR threshold) && PORTn_LPR_CTRL == 1), then assert pa_rmark to DDRC for all best effort ports.
	If ((slots available in read HPR CAM ≤ read HPR threshold) && PORTn_HPR_CTRL == 1), then assert pa_rmark to DDRC for all low-latency ports
Write channel throttle rule	If ((slots available in write CAM ≤ write CAM threshold) && PORTn_WR_CTRL == 1), then assert pa_wmask to DDRC for all best-effort ports.

Interrupt Sources

Several events in the DDR memory controller can assert IRQ 144. The interrupts from the following sources are managed by the DDR_QOS_CTRL.QOS_IRQ_{STATUS, MASK} registers.

- Correctable ECC error [DDR_ECC_CORERR].
- Uncorrectable ECC error.
- DFI initialization complete [DFI_INIT_COMP].
- DFI parity error due to mode register set (MRS) [DFI_ALT_ERR_FTL].
- DFI parity error counter reaches to its maximum count [DFI_ALT_ERR_MAX].

- DFI parity or CRC error detected on the DFI interface [DFI_ALT_ERR].
- Performance counter interrupt when register copy is done [PC_COPY_DONE].
- Invalid DDR memory controller register access [INV_APB].

All interrupts from the list, except for *DFI initialization complete*, must be cleared in both the QoS controller and DDR controller by writing to their respective clear registers. The steps that are required for clearing interrupts are listed.

1. Clear the interrupt in the DDR controller by writing to the respective clear register.
2. Clear the interrupt in the QoS controller by writing to the DDR_QOS_CTRL.qos_irq_status register.

DDR Subsystem Overview

The DDR subsystem (Figure 17-4) is divided into two major blocks, the DDR memory controller and the DDR PHY and I/O, and includes the DRAM memory device(s).

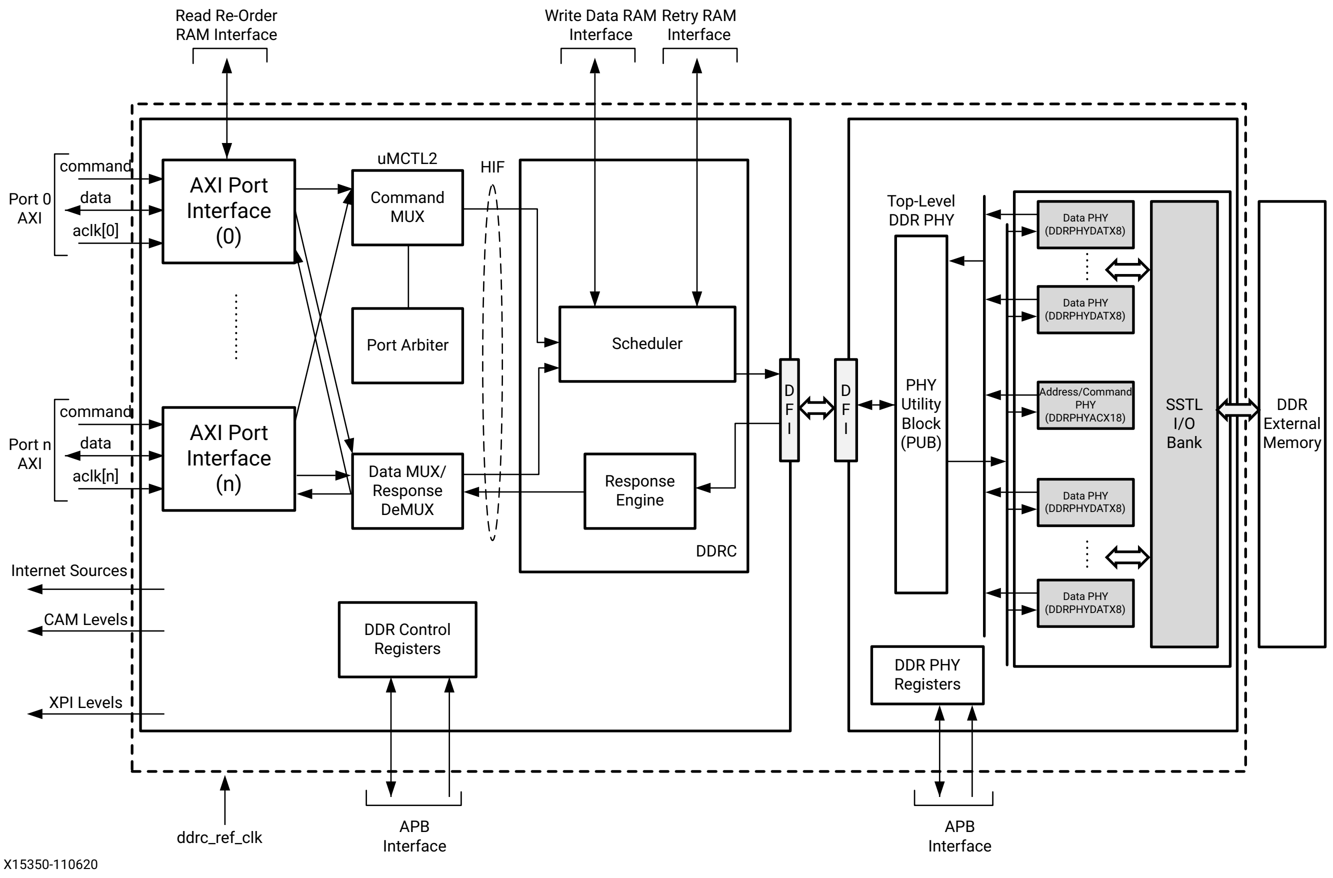


Figure 17-4: **DDR Subsystem Block Diagram**

The DDR memory controller consists of four major blocks: an AXI port interface, a port arbiter, a DDR controller, and the APB register block. The AXI port interface (XPI) interfaces the AXI application port to the memory controller. It converts AXI burst into read and write requests that are forwarded to the port arbiter. The port arbiter block arbitrates command requests from multiple AXI port interfaces and ensures maximum memory bus efficiency. The DDR controller (DDRC) block contains a logical content addressable memory (CAM) that holds information on the commands. The CAM is used by the scheduling algorithms to optimally schedule commands to be sent to the PHY, based on priority, bank/rank status and DDR timing constraints.

AXI Port Interface

The AXI port interface (XPI) provides the interface to the application ports. It provides bus protocol handling, data buffering and reordering for read data, data bus size conversion (upsizing or downsizing), and memory burst address alignment.

The XPI interfaces the AXI application port to the DDR memory controller and performs the following main functions.

- Read address generation.
- Write address generation.
- Write data generation.
- Read data and response generation.
- Write response generation.

The XPI converts AXI bursts into DRAM read and write requests that are forwarded to the port arbiter (PA). In the opposite direction, the XPI converts the responses from the DDRC into appropriate AXI responses. All AXI ports are configured synchronous to the memory controller clock.

Read Address Channel

The AXI read address channel has the following features.

- The read transaction can be of any length up to 16.
- The burst types supported are incremental and wrapping. Fixed burst is not supported.
- The burst start address can be unaligned to the AXI data width boundaries.
- The size of the burst can be less than the full width of the AXI data bus (also known as sub-sized transfers).

All read requests are stored in read-address queue (RAQ). Generation of new read requests are based on alignment (derived from AXI address and size), burst lengths (derived from AXI length and memory burst length), and burst type (derived from AXI incremental or wrapping). Each AXI burst is divided into packets of length equal to the memory burst length (BL4, BL8, or BL16). In case of an unaligned burst, the first read request is unaligned and the remaining read requests are aligned. In general, realignment to a memory burst boundary potentially causes some data beats to be discarded (affecting bandwidth) and potentially introduces additional latency on the read data and response channel. The XPI handles the generation of the token that is used by the DDRC for identifying the read command and corresponding data.

Write Address Channel

Write address queue (WAQ) is used to store all the addresses for write requests from a given port. There is a single queue for all AXI IDs from a given port. The write-address channel behavior is similar to the read-address channel. Write address and read address channels are independent and the ordering between the write and read requests might not be preserved. To preserve the sequence, a higher-level protocol needs to wait for a read/write response before sending the next transaction. Transactions across the ports are independent and can be issued in any order. The write command is not forwarded to the DDRC until the write data is collected and the strobes are evaluated.

Read Data and Response Channel

The XPI handles the common response interface to the DDRC to process the read data from the memory. AXI read data and response channel has a single-data storage queue, the read data queue (RDQ). Data from different IDs are stored in the same queue and are returned in the order of read-address acceptance. The controller provides an OKAY response for each read, except for exclusive read transactions. A SLVERR response can be returned for read transactions (both normal and exclusive) in the following cases.

- ECC uncorrected error detected at the DFI.
- Invalid LPDDR3 row address.
- Transaction is poisoned.

The read data can be returned from the DDRC in a different order from the order that the read commands are forwarded from the XPI. This is due to the re-ordering of read commands in the DDRC to maximize SDRAM bandwidth. A read reorder buffer is implemented in each port to reorder the read data for that port to the same order as the order of the AXI read commands. The read reorder buffer SRAM holds the same number of entries as the read CAM and each entry holds the read data corresponding to a DDR command. The AXI protocol allows the read data for transactions of different IDs to be interleaved. To reduce potential delays where read data for one ID is blocked waiting for data associated with another ID, the read reorder buffer is organized in number of virtual channels. The read reorder virtual channel (RRVC) is a mechanism to allow independent read data reordering between multiple groups of AXI IDs.

Write Data Channel

The AXI write data channel has a data storage queue. This queue is used as a registering layer between the AXI domain and DDR controller. At the output of the write-data queue (WDQ), some beats of a write data can be masked depending on alignment, burst size, and burst length. DDR write data from each port is forwarded to the DDRC, which forwards the data to the DDR.

Write Response Channel

The write response is generated once the last beat of write data, for a given AXI burst, is accepted by the DDRC. The write response generation makes use of the result of the exclusive access monitor. For a write transaction, the response is always returned as OKAY. For an exclusive write transaction, the response can be returned as OKAY or EXOKAY. A SLVERR response can be returned in the following cases.

- Invalid LPDDR3 row address.
- On-chip parity address or data error.
- Transaction is poisoned.

Exclusive Access

All exclusive read transactions have an EXOKAY response (except in the case of an ECC uncorrectable error). Successful exclusive write accesses have an EXOKAY response, unsuccessful exclusive write accesses return an OKAY response. If an exclusive write fails, the data mask for the exclusive write is forced Low and the data is not written. The DDRC monitors one address per transaction ID for exclusivity. Therefore, if a master does not complete the write portion of an exclusive operation, a subsequent exclusive read to the same ID changes the address that is being monitored for exclusivity. Once an exclusive access monitor for a given address is enabled, all write transactions are monitored for violation, regardless of the originating port. The violation check operates across the ports. The exclusive access monitor compares the exclusive write transaction address, size, length, ID, and port number against the exclusive read transaction address, size, length, ID, and port number, and only accepts an exclusive write when these parameters match. Otherwise, the exclusive write is considered as a fail. An exclusive access monitor is present in the DDRC. Zynq UltraScale+ devices support eight exclusive access address monitors for six ports.

XMPU Poisoned Transaction

A sideband signal, AxPOISON, renders an AXI transaction (read or write) invalid and causes an error response (AXI SLVERR). If a write is poisoned, all of its strobes are deasserted, making the write effectively transparent to the memory. If a read is poisoned, the command is issued to the DDR memory and all of the read data beats are overridden and returned as all zeros in conjunction with error response.

Port Arbiter

The port arbiter (PA) block provides latency sensitive, priority-based arbitration between the DRAM commands issued by the XPIs (by the ports). The PA block arbitrates command requests from six AXI ports to the host interface (HIF) of the DDR controller (DDRC). The port arbiter is comprised of multiple tiers of arbitration stages which include the following.

Read/Write Arbitration

The main goal of the read/write arbitration is to combine reads and writes together as long as the selected direction has available credits and the timeout has not occurred for any port of the opposite direction. If all conditions are equal, reads are prioritized over writes. Minimizing direction switches improves memory bus efficiency.

For example while executing reads, stay on the reads as long as there is a timed-out read port or expired video/isochronous priority reads (VPR) with available credit, else switch to writes if there is a timed-out write port or expired video/isochronous priority writes (VPW) with available credit. Otherwise, switch to writes when there is no read-credit left and there is a pending write with available credit.

While executing writes, stay on the writes as long as there is a timed-out write port or expired-VPW with available credit, else switch to reads if there is a timed-out read port or expired-VPR with available credit. Otherwise, switch to reads if there is an HPR read port with available credit, else switch to reads when there is no write-credit left and there is a pending read with available credit.

The timeouts are implemented using aging counters implemented per port, per direction that count down the time when a port is requesting but not granted. The timeout condition occurs when a port aging counter becomes 0 and the port becomes the highest priority requester (priority 0) to the port arbiter. The PCFGR [rd_port_priority] and PCFGW [wr_port_priority] register bits determine the initial value of the counters. The aging feature and the timeout are enabled by the PCFGR [rd_port_aging_en] and PCFGW [wr_port_aging_en] register bits.

Read and Write Priorities

The read channel of a port can be set to operate as high priority reads (HPR), low priority reads (LPR), or video/isochronous priority reads (VPR). The write channel of a port can be set as best-effort writes (BEW) or video/isochronous priority writes (VPW). The PA gives higher priority to an HPR read port than an LPR/VPR read port. VPRs that are not expired are treated as LPRs from the arbiter point of view. If a VPR transaction expires in the XPI, it has higher priority than HPRs or writes.

BEW and VPW have the same initial priority. VPWs that are not expired, are treated as BEWs from the arbiter point of view. If a VPW transaction expires in the XPI, it has higher priority than BEWs. When multiple VPR/VPW commands expire simultaneously, the PA executes them in round-robin order to the DDRC.

Port Command Priority

The next tier of arbitration policy is the multiple-level port command priorities. The priorities are per command and can dynamically change for a given port based on AXI AxQoS signals (arqos/awqos). This tier of arbitration has a lower-level priority than the timeout tier. In addition, for reads, the port priority tier has lower priority than the HPR/LPR-VPR tier.

Round-Robin Arbitration

After passing all tiers of arbitration, a tie is resolved by the final round-robin arbitration stage. The round-robin pointer starts from a port that has the lowest port index. After a grant, the pointer is moved to the first active requester after the one that just received the grant.

Port Arbiter Masking

When the mask bit for an AXI interface to the QoS controller is set, it masks the request from the corresponding AXI port/direction to the port arbiter (PA). An external QoS controller controls the port arbitration by throttling certain XPI ports based on traffic class, queue status, and other dynamic criteria. The port masking is managed by the QoS setting in the tools and the various QoS policies followed by the DDRC.

DDR Controller Address Map

The DDR controller (DDRC) block performs the scheduling and SDRAM command generation. It holds information on the commands, and then based on the scheduling algorithms optimally schedule commands to be sent to the PHY-based on priority, bank/rank status, and DDR timing constraints.

Address Map

A master that wants to access the DDR provides memory read and write requests using a system address. The system address is the command address of a transaction as presented on one of the XPI data ports. The address mapper block within the DDR controller converts this system address to a physical address. It maps the system address to the SDRAM rank, bank group (for DDR4), bank, row, and column addresses.

The controller supports the definition of up to two disjoint memory regions mapping to the SDRAM consecutive addresses. The system address region specification is the same for all data ports. An error response is not generated by the controller for addresses falling outside the specified address regions. The same address translation is applied from one base address to the next base address. The base addresses must be specified and they cannot overlap. It is assumed that an outside agent can generate address decode errors if errors happen to occur. An example is shown in Figure 17-5.

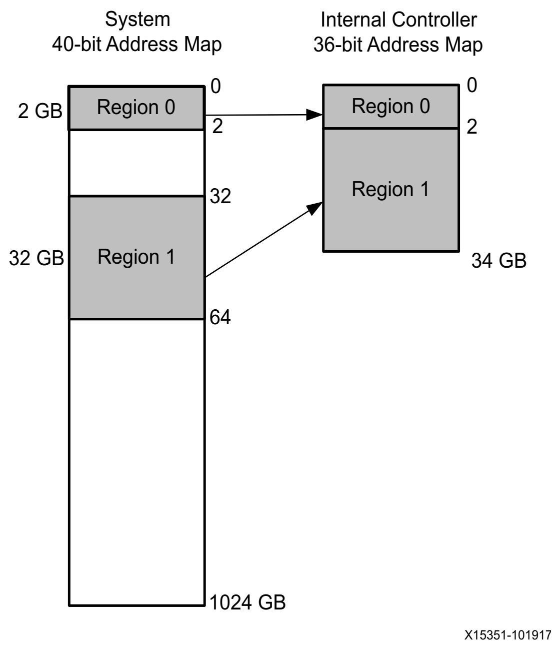


Figure 17-5: **Address Region Mapping Example**

SDRAM Address Mapping

The DDRC is responsible for mapping system addresses used by the PS and PL AXI masters to the SDRAM row, bank, and column addresses. Optimizing the mapping to specific data access patterns allows increased SDRAM utilization by reducing page and row change overhead. Many combinations of address remapping are not available however, the bank-row-column and row-bank-column configurations are achievable.

The first part of the mapping is the conversion of a system address to an AXI byte address. The DDRC maps the disjointed address regions into internal consecutive addresses. The second part of this mapping is conversion of AXI byte address to HIF word address. This is performed in the XPI block. The last part is the conversion of the HIF word address to the SDRAM address. A flexible address mapper maps the HIF word address to the SDRAM rank/bank/bank group/row/column address. This address mapper is located within the DDRC.

The address mapper maps HIF word addresses to SDRAM addresses by selecting the HIF address bit that maps to every applicable SDRAM address bit. The available address space is only accessible when no two SDRAM address bits are determined by the same HIF address bit. The registers ADDRMAPx (x = 0 to 11) are used to program the address mapper.

Each SDRAM address bit has an associated register vector to determine its source. The associated HIF address bit is determined by adding the internal base of the ADDRMAPx (x = 0 to 11) register to the programmed value for that register, as described in Equation 17-1.

[HIF address bit number] = [internal base] + [register value] *Equation 17-1*

For example, an ADDRMAP3.addrmap_col_b7 register internal base is 7. When a full data bus is in use, column bit 7 is determined by Equation 17-2.

[Internal base (i.e., 7)] + [register value] *Equation 17-2*

If the ADDRMAP3.addrmap_col_b7 register is programmed to 2, then the HIF address bit is as shown in Equation 17-3.

[AXI address bit number]= 7 + 2 = 9 *Equation 17-3*

The result is that the column address bit 7 that is sent to the SDRAM is mapped to an HIF address bit of *_ADDR[9].

In the half bus-width mode, all the column bits shift up by one bit. In this case, the ADDRMAP3.addrmap_col_b6 register determines the mapping of the SDRAM column address bit 7. In the quarter bus-width mode (only supported for DDRA), all of the column bits shift up by two bits.

Address Collision Handling

The DDRC can execute transactions out-of-order while ensuring that all transactions appear as if they are executed in the order that they are received. Every transaction that requires a response from the DDRC arrives with a token number which is provided back to the Zynq UltraScale+ MPSoC as part of the response. Because the DDRC queues transactions prior to execution, it is possible that multiple transactions to the same SDRAM address can arrive before the first transaction to that address is issued. To enforce ordering of accesses to the same address, the DDRC uses the following algorithm.

- New read colliding with queued read causes no problems. The two reads can end up being executed out-of-order.
- New write colliding with queued write.
    - If a write combine is enabled, the DDRC overwrites the data for the old write with the one from the new write and only performs one write transaction (write combine).
    - If the write combine is disabled, the DDRC holds the new write transaction in a temporary buffer, applies flow control to prevent more transactions from arriving, and flushes the internal queue holding the colliding transaction until that transaction is serviced. Once completed, the DDRC accepts the new transaction.
- New read (or write) colliding with queued write (or read) respectively. In this case, the DDRC performs the following sequence.
    a. Holds the new transactions in a temporary buffer.
    b. Applies flow control back at the AXI port interface to prevent more transactions from arriving.

c. Flushes the internal queue holding the colliding transaction until that transaction is serviced.

d. Accepts the new transaction and removes flow control.

- A new read colliding with both read and write can happen when a read collides with a read-modify-write (RMW) command. In this case, the reads are flushed until the read collision is cleared, then the writes are flushed
- A new write colliding with both read and write can happen when a write collides with a RMW command. In this case, the new write is held in a temporary buffer until the read is completed. Then, it is combined with the queued write (if write combine is enabled).
- In a new RMW colliding with queued write case, the new RMW is stored in a temporary buffer until the queued write is completed.

Error Correcting Code

The error correcting code (ECC) block consists of an encoder and a decoder that can detect and correct single-bit errors, and detect double-bit errors for configurations where the DRAM data width is configured to be 32 or 64 bits. The syndrome bits are calculated on a 32-bit or 64-bit basis based on DRAM data width selection. The ECC block support only covers the data bus and is not applicable for address and command bus. ECC is not supported for LPDDR3.

The ECC block has these features:

- Hamming code based ECC calculations
- Single-bit error detection and correction
- Double-bit error detection
- Error counter for single-bit and double-bit errors
- Supports error injection (single-bit and double-bit errors) for testing and debugging
- Interrupt generation on error

Figure 17-6 illustrates the interface between the DDR subsystem and DRAM memories with ECC enabled.

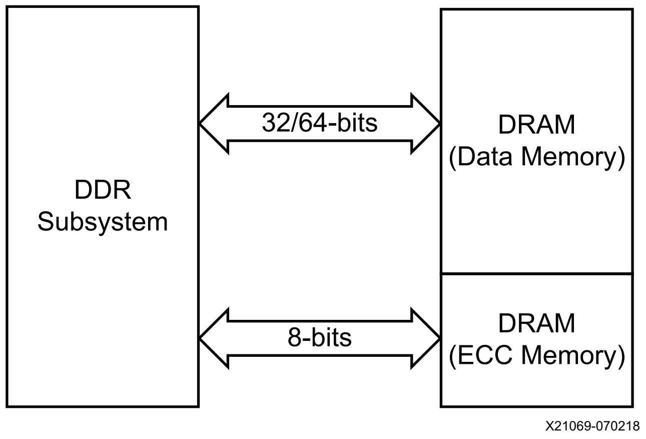


Figure 17-6: **DDR Interface with ECC Enabled**

The ECC feature can be enabled or disabled by programming the ECCCFG0.ecc_mode register. If ECCCFG0 [ecc_mode] = 100, the ECC is enabled and the controller performs the following functions.

- On writes, the syndrome is calculated across DRAM data width, and the resulting ECC code is written as an additional byte along with the data as shown in Figure 17-6. This additional ECC byte is always written to the uppermost byte (byte 8 for both 32-bit and 64-bit DRAM).
- On reads, the DRAM data bus including the ECC byte is read from the DRAM and is then decoded. A check is performed to verify that the ECC byte is as expected, based on the data in DRAM data bus. If it is correct, the data is sent to the processing system as normal. If it is not correct it executes steps as described in ECC Error Behavior.
- On read-modify-write operations, first a read is performed. The read data is then combined with the write data, making use of the write mask received to over-write certain bytes of the data that are read. The ECC is then calculated on the resulting data, and the write is performed.

Note: Avoid streaming high priority, non 64-bit write transactions to memory when ECC is enabled. The read-modify-write sequences negatively impact memory controller performance and might cause unrelated high throughput video traffic to be starved.

Note: Enabling ECC on the DDR can diminish overall available memory bandwidth or increase access latency under certain conditions. System designers should carefully consider the trade offs between enabling ECC and bandwidth/latency requirements of other components in the system, when there is a possibility of a significant amount of partial (sub-64 bit) writes to memory. See Answer Record 67651 for more information.

ECC Initialization

When the ECC mode is enabled, a write operation computes and stores an ECC code along with the data, and a read operation reads and checks the data against the stored ECC code. Consequently, it is possible to receive ECC errors when reading uninitialized memory locations. To avoid this problem, all memory locations must be written before being read.

Writing to the entire DDR DRAM through the CPU can be time intensive. It might be worthwhile to use a DMA device to generate larger bursts to the DDR controller initialization and offload the CPU.

If the ECC mode is selected in the Vivado tools, the first-stage boot loader (FSBL) initializes the DDR DRAM to a known value.

ECC Error Behavior

The controller performs these steps when it detects correctable ECC errors:

1. Sends the corrected data to the PS core as part of the read data.
2. Writes the address, syndrome bits, and data mask bits to ECC registers in the DDRC register set.
3. Performs a RMW operation to correct the data present in the DRAM (only if ECC scrubbing is enabled (ECCCFG0.dis_scrub = 0). This RMW operation is invisible to the core. Only one scrub RMW command can be outstanding in the controller at any time. No scrub is performed on single-bit ECC errors that occur while the controller is processing another scrub RMW.
4. Sets the [DDRECC_CORERR] interrupt bit in the DDR_QOS_CTRL.QOS_IRQ_STATUS register.

The controller performs these steps when it detects uncorrectable ECC errors:

1. Sends the data with the error back to the AXI interconnect as the read data.
2. Writes the address and syndrome bits to ECC registers in the DDRC register set.
3. Generates an error response SLVERR on the AXI interface. If L2 cache is disabled, CPU receives the SLVERR response directly which can cause a Data Abort exception. If L2 cache is enabled, L2 cache reports the SLVERR by issuing an interrupt to CPU.
4. Sets the [DDRECC_UNCRERR] interrupt bit in the DDR_QOS_CTRL.QOS_IRQ_STATUS register.

Data Mask During ECC Mode

When ECC is enabled, the memory controller generates a simple write to the DRAMs when the data is 64 bits wide and aligned to a 64-bit boundary. Otherwise, the controller performs a more time-consuming read-modify-write operation on the DRAM. In this case, the controller first fetches the read data from the DRAM and merges it with the write data from the AXI interconnect and generates the ECC bits. The controller then writes whole words with ECC to the DRAMs.

Note: If a stream of partial writes are performed by an AXI port interface with high priority, it can have a major negative impact on the ability of other ports to access memory. For example, this can cause an isochronous video stream to drop data.

Encoding for Corrected Bit Number

Table 17-8 provides the encoding used for the status register field DDRC.ECCSTAT [corrected_bit_num] which indicates the bit that is corrected.

Table 17-8: **Encoding for DDRC.ECCSTAT [corrected_bit_num]**

Value on DDRC.ECCSTAT[corrected_bit_num]	Bit that has Error	
	DRAM Bus Width = 64	DRAM Bus Width = 32
0	64 (ecc[0])	64 (ecc[0])
1	65 (ecc[1])	65 (ecc[1])
2	66 (ecc[2])	66 (ecc[2])
3	0	0
4	67 (ecc[3])	67 (ecc[3])
5	1	1
6	2	2
7	3	3
8	68 (ecc[4])	68 (ecc[4])
9	4	4
10	5	5
...	...	...
15	10	10
16	69 (ecc[5])	69 (ecc[5])
17	11	11
18	12	12
...	...	...
21	15	15
22	16	16
...	...	...
31	25	25
32	70 (ecc[6])	70 (ecc[6])
33	26	26
34	27	27
...	...	...
38	31	31
39	32	NA
...	...	...
63	56	NA
64	71 (ecc[7])	71 (ecc[7])

Table 17-8: **Encoding for DDRC.ECCSTAT [corrected_bit_num] *(Cont'd)***

Value on DDRC.ECCSTAT[corrected_bit_num]	Bit that has Error	
	DRAM Bus Width = 64	DRAM Bus Width = 32
65	57	NA
66	58	NA
...	...	...
71	63	NA

ECC Programming Model

This section describes the ECC programming requirements.

Note: These configurations are in addition to the regular DDR initialization programming. Initialization of the entire DDR space before reading any data from it is recommended to prevent ECC error generation as a result of accessing uninitialized areas of memory. See ECC Initialization for more details.

Monitoring ECC Status

1. Read the ECC error counter register (DDRC.ECCERRCNT) to see the number of correctable and uncorrectable ECC errors detected.
2. If ECCERRCNT is non-zero, read the ECC status register (DDRC.ECCSTAT), which provides the bit number corrected by single-bit ECC error, single-bit error indicators, and double-bit error indicators. See Encoding for Corrected Bit Number for the encoding used for the status register field DDRC.ECCSTAT [corrected_bit_num].
3. Read the DDRC.{ECCCADDR0, ECCCADDR1} register to see the bank/row/column information of the corrected ECC error.
4. Read the DDRC.{ECCUADDR0, ECCUADDR1} register to see the bank/row/column information of the uncorrected ECC error.

ECC Poisoning

1. Program DDRC.ECCCFG1 [data_poison_en] to `1'b1`, which introduces the ECC errors on writes to the address specified by the DDRC.ECCPOISONADDR {0, 1} registers.
2. Selects the correctable data poisoning or uncorrectable data poisoning. Program DDRC.ECCCFG1 [data_poison_bit] bit (`1'b0` - 2-bit (uncorrectable) data poisoning, `1'b0` - 1-bit (correctable) data poisoning).
3. Set the address to be poisoned in the DDRC.ECCPOISONADDR {0, 1} registers.

   Note: ECCPOISONADDR0[11:0], ecc_poison_col, must be burst aligned. In 64-bit bus width mode, ecc_poison_col[2:0] must be set to 0. In 32-bit bus width mode, ecc_poison_col[3:0] must be set to 0.
4. Write to the poison address. Subsequent reads to the same 1-2 DRAM burst length of addresses are detected.

A sample test program tests the ECC correctable/uncorrectable error detection by inserting error bits into DDR memory is described in the *Zynq UltraScale+ MPSoC – 64-bit DDR Access with ECC* technical article [Ref 25].

ECCSTAT Register DDRC for Encoding of ECC Corrected Bit Number

Figure 17-7 shows the DDR ECC error bit location.

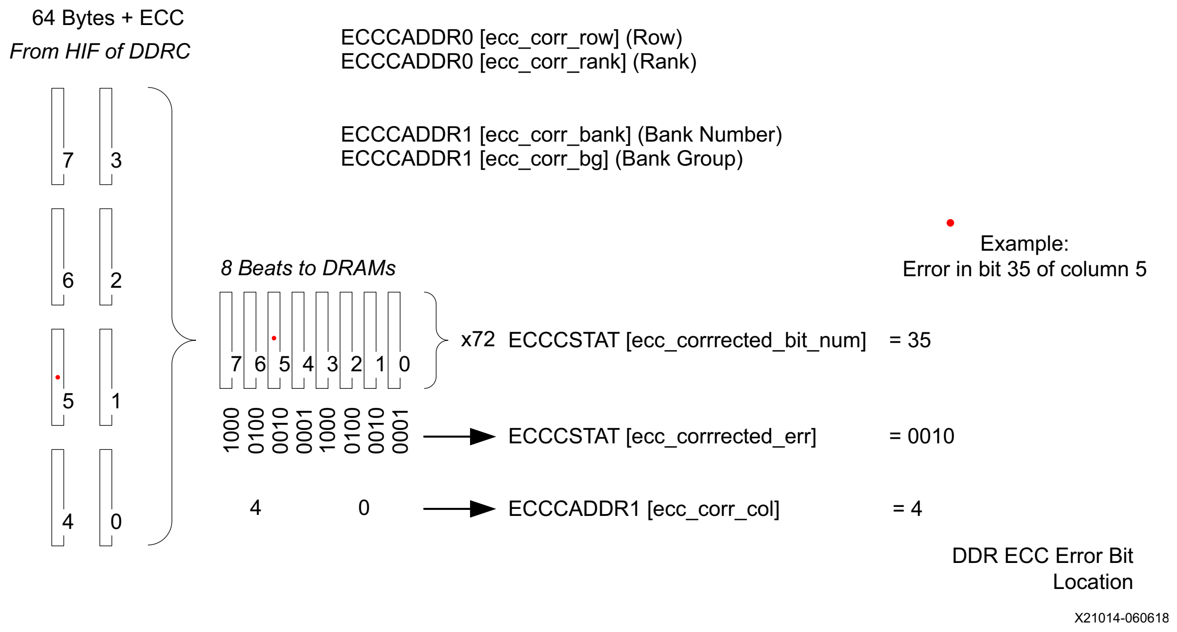

64 Bytes + ECC
From HIF of DDRC
7
3
6
2
5
1
4
0
ECCCADDR0 [ecc_corr_row] (Row)
ECCCADDR0 [ecc_corr_rank] (Rank)
ECCCADDR1 [ecc_corr_bank] (Bank Number)
ECCCADDR1 [ecc_corr_bg] (Bank Group)
Example:
Error in bit 35 of column 5
8 Beats to DRAMs
7 6 5 4 3 2 1 0
x72 ECCCSTAT [ecc_corrrected_bit_num] = 35
1000 0100 0010 0001 1000 0100 0010 0001
ECCCSTAT [ecc_corrrected_err] = 0010
4 0
ECCCADDR1 [ecc_corr_col] = 4
DDR ECC Error Bit Location
X21014-060618

Figure 17-7: **DDR ECC Error Bit Location**

Functional Description

The DDR PHY (DDRP) provides the interface between the DDR controller and the I/O pads. It handles all issues associated with command launch, write data launch, and read data capture.

DDR PHY PLL Control

The six DDR PLLs provide fast, accurate clocking to the I/O buffers interfacing to the DRAMs. The DDR_REF_CLK provides the source clock to the PLLs, which are controlled by six sets of registers. The data and ECC registers can be updated individually or as a group using the broadcast set of registers, PLLCR{0:5}. The PLL control architecture is shown in Figure 17-8.

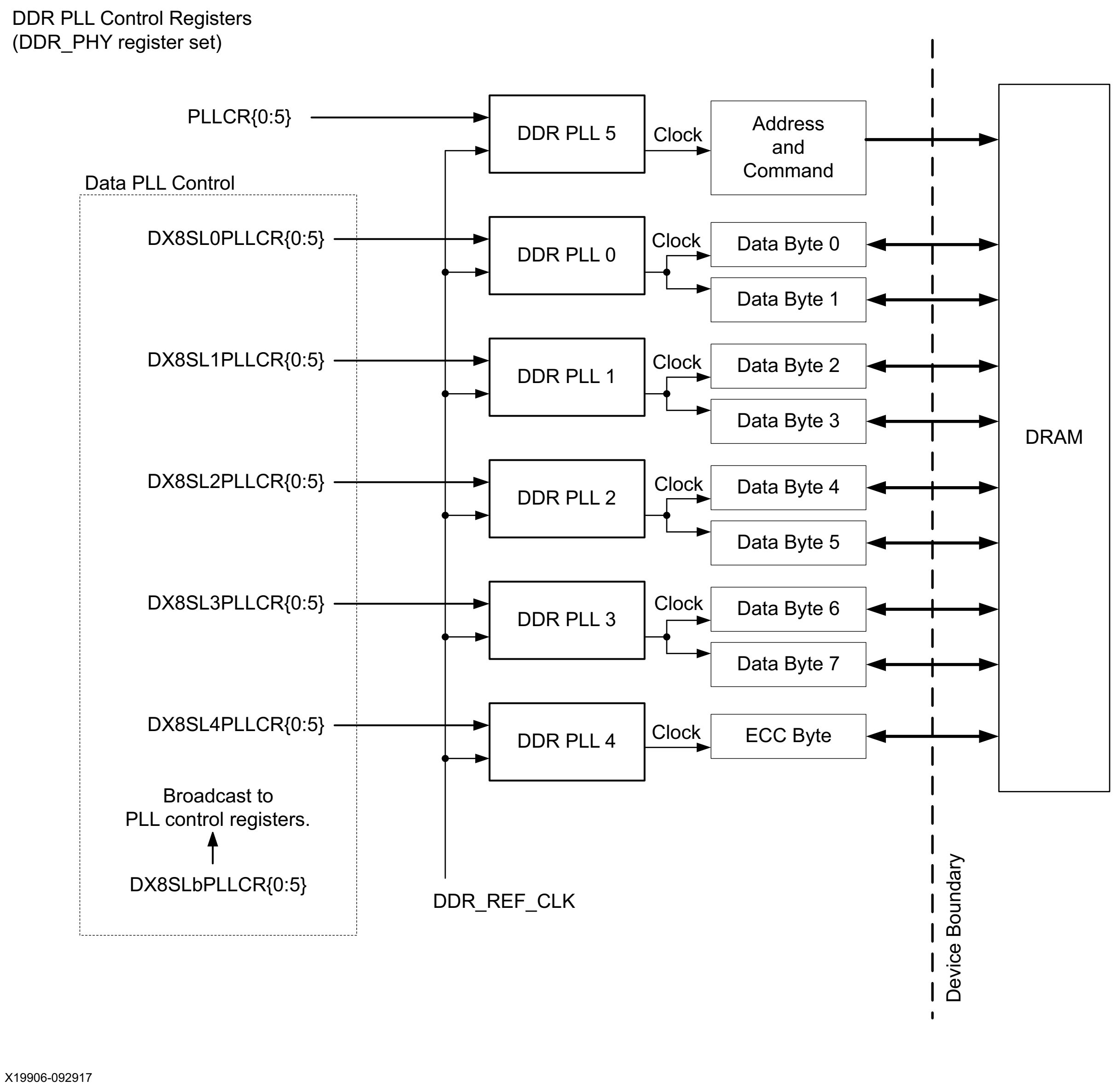


Figure 17-8: **DDR PHY PLL Control Architecture**

The DDR controller PHY consists of the following units.

PHY Utility Block

The PHY utility block (PUB) controls various features of the PHY such as initialization, DQS gate training, delay line calibration and VT compensation, write leveling, and programmable configuration controls. It also provides a DFI interface to the PHY. The PUB includes configuration registers in the DDR_PHY register set.

PHY Description

The PHY has four types of I/O buffers: address/command, data (8-bit blocks), clocking, and SSTL (configurable).

- DDRPHYAC
  The DDR SDRAM address/command PHY (DDRPHYAC) provides an address and command interface to the external SDRAM memories. A memory interface would typically contain a single address/command PHY.
- DDRPHYDATX8
  The DDR SDRAM data PHY (DDRPHYDATX8) provides data interface to one byte of an external SDRAM memory.
- SSTL I/O library
  The stub-series terminated logic (SSTL) I/O library includes process, voltage, and temperature (PVT)-compensated, on-die termination (ODT), and output impedance.

Controller Initialization

The controller initialization sequence has the following phases.

- PHY initialization
- DRAM initialization
- Data training

Figure 17-9 and Figure 17-10 show high-level illustrations of the initialization sequence of the PHY.

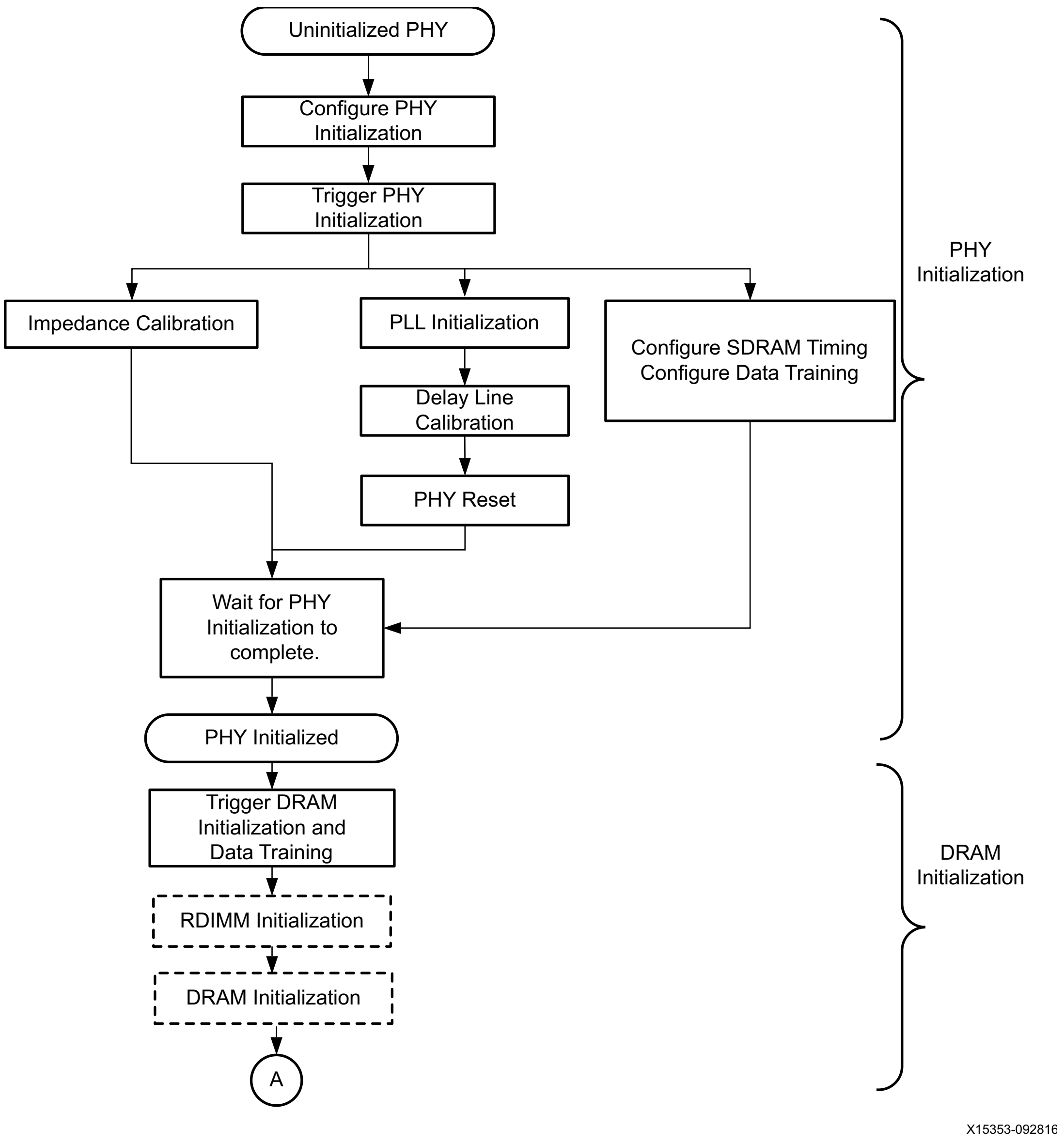


Figure 17-9: **PHY Initialization Sequence**

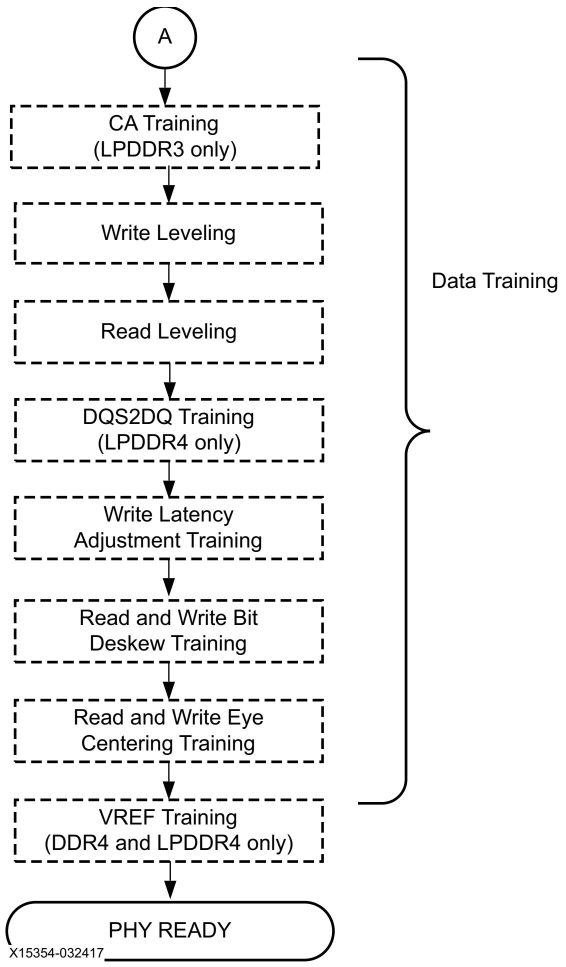

A
CA Training (LPDDR3 only)
Write Leveling
Read Leveling
DQS2DQ Training (LPDDR4 only)
Write Latency Adjustment Training
Read and Write Bit Deskew Training
Read and Write Eye Centering Training
VREF Training (DDR4 and LPDDR4 only)
PHY READY
Data Training
X15354-032417

Figure 17-10: **PHY Initialization Sequence (*Continued*)**

Impedance calibration failures can be caused by an open or short on the PS_DDR_ZQ pin. Double check to ensure PS_DDR_ZQ is connected to GND with a 240Ω resistor. There should be separate 240Ω resistors at the FPGA and the DRAM.

PHY Initialization

After deassertion of reset, the PHY is uninitialized. PHY initialization is comprised of initializing the PHY PLL(s), running the initial impedance calibration, and running delay-line calibration. These functions can all be triggered at the same time by writing PIR = `x0000_0033`. The initial impedance calibration can be run in parallel with the PLL initialization and subsequent delay line calibration.

DRAM Initialization

The DDR controller performs DRAM initialization. The DDR_PHY PIR must be programmed with PIR = `0004_0001` to transfer control of the DFI interface from the PUB to the DDR controller for DRAM initialization.

Data Training

After RDIMM initialization and SDRAM initialization, the PIR can be programmed to run one or more of the training steps.

The following training steps can be triggered by writing to the appropriate PIR register bit.

1. CA training (LPDDR3 only).
2. Write leveling.
3. Read leveling.
4. DQS2DQ training (LPDDR4 only).
5. Write latency adjust training.
6. Read data bit deskew training.
7. Write data bit deskew training.
8. Read data eye training.
9. Write data eye training.
10. V_{REF} training (DDR4 and LPDDR4).

Dynamic DDR Configuration

The controller is able to be configured for different memory settings at runtime while the DDR controller is in reset. Typically, this is used in DIMM topologies by reading the DRAM configuration from the DIMM SPD EEPROM via a I2C peripheral.

See Answer Record 75768 for more information.

Programming Topics

PHY General Status Register

After initializing the DRAM interface, basic status information is captured in PHY General Status Register 0 (PGSR0). This register indicates whether various initialization and data training steps were completed, and whether any high-level errors or warnings were flagged for any of the steps. Table 17-9 defines the fields within PGSR0.

Table 17-9: **PHY General Status Register 0 (PGSR0)**

Bits	Name	Description	Address
[0]	IDONE	Initialization done: if set, indicates that the DDR system initialization has completed. This bit is set after all the selected initialization routines have completed.	`0xFD080030`
[1]	PLDONE	PLL lock done: if set, indicates that PLL locking has completed.	`0xFD080030`
[2]	DCDONE	Digital delay line (DDL) calibration done: if set, indicates that DDL calibration has completed.	`0xFD080030`
[3]	ZCDONE	Impedance calibration done: if set, indicates that impedance calibration has completed.	`0xFD080030`

Table 17-9: **PHY General Status Register 0 (PGSR0)** ***(Cont'd)***

Bits	Name	Description	Address
[4]	DIDONE	DRAM initialization done: if set, indicates that DRAM initialization has completed.	`0xFD080030`
[5]	WLDONE	Write leveling done: if set, indicates that write leveling has completed.	`0xFD080030`
[6]	QSGDONE	DQS gate training done: if set, indicates that read leveling (DQS Gate Training) has completed.	`0xFD080030`
[7]	WLADONE	Write leveling adjustment done: if set, indicates that write leveling adjustment has completed.	`0xFD080030`
[8]	RDDONE	Read bit deskew done: if set, indicates that read bit deskew has completed.	`0xFD080030`
[9]	WDDONE	Write bit deskew done: if set, indicates that write bit deskew has completed.	`0xFD080030`
[10]	REDONE	Read eye training done: if set, indicates that read eye training has completed.	`0xFD080030`
[11]	WEDONE	Write eye training done: if set, indicates that write eye training has completed.	`0xFD080030`
[12]	CADONE	CA training done: if set, indicates that LPDDR3 CA training has completed.	`0xFD080030`
[14]	VDONE	VREF training done: if set, indicates that DRAM and host VREF training has completed. DDR4 and LPDDR4 only.	`0xFD080030`
[15]	DQS2DQDONE	Write DQS2DQ training done. if set, indicates that write DQS2DQ training has completed. LPDDR4 only.	`0xFD080030`
[18]	DQS2DQERR	Write DQS2DQ training error: if set, indicates that there is an error in DQS2DQ training.	`0xFD080030`
[19]	VERR	VREF training error: if set, indicates that there is an error in VREF training.	`0xFD080030`
[20]	ZCERR	Impedance calibration error: if set, indicates that there is an error in impedance calibration.	`0xFD080030`
[21]	WLERR	Write leveling error: if set, indicates that there is an error in write leveling.	`0xFD080030`
[22]	QSGERR	DQS gate training error: if set, indicates that there is an error in read leveling (DQS Gate Training).	`0xFD080030`
[23]	WLAERR	Write leveling adjustment error: if set, indicates that there is an error in write leveling adjustment.	`0xFD080030`
[24]	RDERR	Read bit deskew error: if set, indicates that there is an error in read bit deskew.	`0xFD080030`
[25]	WDERR	Write bit deskew error: if set, indicates that there is an error in write bit deskew.	`0xFD080030`
[26]	REERR	Read eye training error: if set, indicates that there is an error in read eye training.	`0xFD080030`

Table 17-9: **PHY General Status Register 0 (PGSR0) *(Cont'd)***

Bits	Name	Description	Address
[27]	WEERR	Write eye training error: if set, indicates that there is an error in write eye training.	`0xFD080030`
[28]	CAERR	CA training error: if set, indicates that there is an error in LPDDR3 CA training.	`0xFD080030`
[29]	CAWRN	CA training warning: if set, indicates that there is a warning in LPDDR3 CA training.	`0xFD080030`
[31]	APLOCK	AC PLL lock: if set, indicates that the AC PLL has locked.	`0xFD080030`

By studying the contents of PGSR0, it is possible to identify any errors or warnings that occurred during initialization and training. Additional registers can be checked for more information related to any errors or warnings. See subsequent sections in this chapter for a more detailed description of each initialization and training step, as well as where to look for more debugging information.

Impedance Calibration

The PHY includes calibration I/O cells and finite state machine logic to automatically compensate output drive strength and on-die termination strength, adjusting for variations in process, voltage, and temperature. The Impedance Control Status Registers (ZQnSR) provide additional debugging information. ZQ0SR shows the results of calibration for address, command, and control I/Os. ZQ1SR shows the results of calibration for data, strobe, and mask I/Os. Table 17-10 lists the fields within ZQnSR.

Impedance calibration failures can be caused by an open or short on the PS_DDR_ZQ pin. Double check to ensure PS_DDR_ZQ is connected to GND with a 240Ω resistor. There should be separate 240Ω resistors at the FPGA and the DRAM.

Table 17-10: **Impedance Control Status Register (ZQnSR)**

Bits	Name	Description	Address

[1:0]	ZPD	Output impedance pull-down calibration status. Valid status encodings are:
		`2b00` = Completed with no errors
		`2b01` = Overflow error
		`2b10` = Underflow error
		`2b11` = Calibration in progress
			`0xFD08069C`
			and
			`0xFD0806BC`

[3:2]	ZPU	Output impedance pull-up calibration status. Valid status encodings are:
		`2b00` = Completed with no errors
		`2b01` = Overflow error
		`2b10` = Underflow error
		`2b11` = Calibration in progress
			`0xFD08069C`
			and
			`0xFD0806BC`

Table 17-10: **Impedance Control Status Register (ZQnSR)** ***(Cont'd)***

Bits	Name	Description	Address

[5:4]	OPD	On-die termination (ODT) pull-down calibration status. Valid status encodings are:
		`2b00` = Completed with no errors
		`2b01` = Overflow error
		`2b10` = Underflow error
		`2b11` = Calibration in progress
			`0xFD08069C`
			and
			`0xFD0806BC`

[7:6]	OPU	On-die termination (ODT) pull-up calibration status. Valid status encodings are:
		`2b00` = Completed with no errors
		`2b01` = Overflow error
		`2b10` = Underflow error
		`2b11` = Calibration in progress
			`0xFD08069C`
			and
			`0xFD0806BC`

[8]	ZERR	Impedance calibration error: if set, indicates that there was an error during impedance calibration.	`0xFD08069C`
			and
			`0xFD0806BC`

[9]	ZDONE	Impedance calibration done: indicates that the first round of impedance calibration has completed. Any time impedance calibration is restarted, this bit goes back to 0 until all segments are recalibrated, following which this bit returns to 1.	`0xFD08069C`
			and
			`0xFD0806BC`

[10]	PU_DRV_SAT	Pull-up drive strength code saturated due to drive strength adjustment setting in ZQnPR register. Is non-zero only in LPDDR4 mode. If this is set to `1'b1`, the adjustment factor or ZPROG setting for the corresponding segment needs to be scaled.	`0xFD08069C`
			and
			`0xFD0806BC`

[11]	PD_DRV_SAT	Pull-down drive strength code saturated due to drive strength adjustment setting in ZQnPR register. Is non-zero only in DDR4 mode. If this is set to `1'b1`, the adjustment factor or ZPROG setting for the corresponding segment needs to be scaled.	`0xFD08069C`
			and
			`0xFD0806BC`

[12]	PU_ODT_SAT	Pull-up termination strength code saturated due to drive strength adjustment setting in ZQnPR register. Is non-zero only in DDR4 mode. If this is set to `1'b1`, the adjustment factor or ZPROG setting for the corresponding segment needs to be scaled.	`0xFD08069C`
			and
			`0xFD0806BC`

[13]	PD_ODT_SAT	Pull-down termination strength code saturated due to drive strength adjustment setting in ZQnPR register. Is non-zero only in LPDDR4 mode. If this is set to `1'b1`, the adjustment factor or ZPROG setting for the corresponding segment needs to be scaled.	`0xFD08069C`
			and
			`0xFD0806BC`

PLL Initialization

After triggering reset, the PHY waits for the PLLs to lock before any further initialization task that uses a high-speed (controller) clock can commence. The PLL initialization completion status is indicated by the PGSR0.P.LDONE bit. The lock status of individual PLLs is indicated by the bits in Table 17-11.

Table 17-11: **PLL Lock Status Bits**

Register	Bits	Name	Description	Address
PGSR0	[31]	APLOCK	AC PLL lock: if set, indicates that the AC PLL has locked.	`0xFD080030`
DX0GSR0	[16]	DPLOCK	DATX8 PLL lock: if set, indicates that the DATX8 PLL controlling bytes 0 and 1 has locked.	`0xFD0807E0`
DX2GSR0	[16]	DPLOCK	DATX8 PLL lock: if set, indicates that the DATX8 PLL controlling bytes 2 and 3 has locked.	`0xFD0809E0`
DX4GSR0	[16]	DPLOCK	DATX8 PLL lock: if set, indicates that the DATX8 PLL controlling bytes 4 and 5 has locked.	`0xFD080BE0`
DX6GSR0	[16]	DPLOCK	DATX8 PLL lock: if set, indicates that the DATX8 PLL controlling bytes 6 and 7 has locked.	`0xFD080DE0`
DX8GSR0	[16]	DPLOCK	DATX8 PLL lock: if set, indicates that the DATX8 PLL controlling byte 8 has locked.	`0xFD080FE0`

If any PLLs fail to lock, check the integrity of the Vcc_psddr_pll supply. See the *UltraScale Architecture PCB Design User Guide* (UG583) [Ref 15] to ensure that the guidelines for the Vcc_psddr_pll supply have been followed. Check that the correct memory interface device frequency has been entered in the Zynq UltraScale+ MPSoC DDR configuration page in the Vivado design tools. This number must be set no lower than 166 MHz.

Delay Line Calibration

After the PLLs have locked, the PHY executes delay line calibration before any further initialization task that uses a high-speed (controller) clock can commence.

Each master delay line has to be calibrated for the SDRAM clock period. This is done by measuring the number of delay line steps that are required to produce a delay equal to the DDR clock period. Each master delay line is calibrated independently. Delay line calibration is normally done as part of the PHY initialization sequence.

Once all delay lines have been calibrated, the calibration done status is asserted through a status register bit, PGSR0.DCDONE. The results of the calibration are available in the registers listed in Table 17-12.

Table 17-12: **Master Delay Line Registers**

Register	Bits	Name	Description	Address
ACMDLR0	[8:0]	IPRD	Initial period: initial period measured by the master delay line calibration for AC.	`0xFD0805A0`
ACMDLR0	[24:16]	TPRD	Target period: target period measured by the master delay line calibration for AC. This changes with voltage and temperature.	`0xFD0805A0`
DX0MDLR0	[8:0]	IPRD	Initial period: initial period measured by the master delay line calibration for byte 0.	`0xFD0807A0`
DX0MDLR0	[24:16]	TPRD	Target period: target period measured by the master delay line calibration for byte 0. This changes with voltage and temperature.	`0xFD0807A0`
DX1MDLR0	[8:0]	IPRD	Initial period: initial period measured by the master delay line calibration for byte 1.	`0xFD0808A0`
DX1MDLR0	[24:16]	TPRD	Target period: target period measured by the master delay line calibration for byte 1. This changes with voltage and temperature.	`0xFD0808A0`
DX2MDLR0	[8:0]	IPRD	Initial period: initial period measured by the master delay line calibration for byte 2.	`0xFD0809A0`
DX2MDLR0	[24:16]	TPRD	Target period: target period measured by the master delay line calibration for byte 2. This changes with voltage and temperature.	`0xFD0809A0`
DX3MDLR0	[8:0]	IPRD	Initial period: initial period measured by the master delay line calibration for byte 3.	`0xFD080AA0`
DX3MDLR0	[24:16]	TPRD	Target period: target period measured by the master delay line calibration for byte 3. This changes with voltage and temperature.	`0xFD080AA0`
DX4MDLR0	[8:0]	IPRD	Initial period: initial period measured by the master delay line calibration for byte 4.	`0xFD080BA0`
DX4MDLR0	[24:16]	TPRD	Target period: target period measured by the master delay line calibration for byte 4. This changes with voltage and temperature.	`0xFD080BA0`
DX5MDLR0	[8:0]	IPRD	Initial period: initial period measured by the master delay line calibration for byte 5.	`0xFD080CA0`
DX5MDLR0	[24:16]	TPRD	Target period: target period measured by the master delay line calibration for byte 5. This changes with voltage and temperature.	`0xFD080CA0`
DX6MDLR0	[8:0]	IPRD	Initial period: initial period measured by the master delay line calibration for byte 6.	`0xFD080DA0`
DX6MDLR0	[24:16]	TPRD	Target period: target period measured by the master delay line calibration for byte 6. This changes with voltage and temperature.	`0xFD080DA0`
DX7MDLR0	[8:0]	IPRD	Initial period: initial period measured by the master delay line calibration for byte 7.	`0xFD080EA0`

Table 17-12: **Master Delay Line Registers *(Cont'd)***

Register	Bits	Name	Description	Address
DX7MDLR0	[24:16]	TPRD	Target period: target period measured by the master delay line calibration for byte 7. This changes with voltage and temperature.	`0xFD080EA0`
DX8MDLR0	[8:0]	IPRD	Initial period: initial period measured by the master delay line calibration for byte 8.	`0xFD080FA0`
DX8MDLR0	[24:16]	TPRD	Target period: target period measured by the master delay line calibration for byte 8. This changes with voltage and temperature.	`0xFD080FA0`

DRAM Initialization

After the PHY PLLs have been initialized and the delay lines and PHY I/Os have been calibrated, the interface is ready for initializing the DRAMs. The DRAM must be correctly initialized before further training of the PHY can be executed. The controller has built-in logic for performing DRAM initialization that is applicable to all DRAM types supported by the controller. The built-in initialization completion is indicated through the PGSR.IDONE register bit.

CA Training (LPDDR3 Only)

CA training is a feature of LPDDR3 memory used for optimizing the setup and hold times of the CA bus relative to the memory clock. CA training is a special mode of operation in the memory enabled through mode register writes. In this mode, the value of the CA bus captured by the memory during assertion of the chip select (cs_n) is reflected back on the DQ bus. The rising edge CA values are returned on even DQ bits and falling edge CA values are returned on odd DQ bits. Since minimum DQ width is 16, only 8 CA bits can be trained in a session. Consequently, two sessions are required for complete CA training. Completion of CA training is signaled by PGSR0.CADONE. High-level error and warning flags are PGSR0.CAERR and PGSR0.CAWRN, respectively.

CA training deskews the CA bits by adjusting bit delay line (BDL) delays on all the CA bits. The results of this deskew are visible in the AC bit delay line registers, as listed in Table 17-13.

Table 17-13: **AC Bit Delay Line Registers**

Register	Bits	Name	Description	Address
ACBDLR1	[5:0]	ACTBD	Delay select for the BDL on ACTN. In LPDDR3 mode, with address copy enabled, this is connected to CA_B[9].	`0xFD080544`
ACBDLR2	[5:0]	BA0BD	Delay select for the BDL on BA[0]. In LPDDR3 mode, with address copy enabled, this is connected to CA_B[6].	`0xFD080548`

Table 17-13: **AC Bit Delay Line Registers *(Cont'd)***

Register	Bits	Name	Description	Address
ACBDLR2	[13:8]	BA1BD	Delay select for the BDL on BA[1]. In LPDDR3 mode, with address copy enabled, this is connected to CA_B[7].	`0xFD080548`
ACBDLR2	[21:16]	BG0BD	Delay select for the BDL on BG[0]. In LPDDR3 mode, with address copy enabled, this is connected to CA_B[8]	`0xFD080548`
ACBDLR6	[5:0]	A00BD	Delay select for the BDL on address A[0].	`0xFD080558`
ACBDLR6	[13:8]	A01BD	Delay select for the BDL on address A[1].	`0xFD080558`
ACBDLR6	[21:16]	A02BD	Delay select for the BDL on address A[2].	`0xFD080558`
ACBDLR6	[29:24]	A03BD	Delay select for the BDL on address A[3].	`0xFD080558`
ACBDLR7	[5:0]	A04BD	Delay select for the BDL on address A[4].	`0xFD08055C`
ACBDLR7	[13:8]	A05BD	Delay select for the BDL on address A[5].	`0xFD08055C`
ACBDLR7	[21:16]	A06BD	Delay select for the BDL on address A[6].	`0xFD08055C`
ACBDLR7	[29:24]	A07BD	Delay select for the BDL on address A[7].	`0xFD08055C`
ACBDLR8	[5:0]	A08BD	Delay select for the BDL on address A[8].	`0xFD080560`
ACBDLR8	[13:8]	A09BD	Delay select for the BDL on address A[9].	`0xFD080560`
ACBDLR8	[21:16]	A10BD	Delay select for the BDL on address A[10]. In LPDDR3 mode, with address copy enabled, this is connected to CA_B[0].	`0xFD080560`
ACBDLR8	[29:24]	A11BD	Delay select for the BDL on address A[11]. In LPDDR3 mode, with address copy enabled, this is connected to CA_B[1].	`0xFD080560`
ACBDLR9	[5:0]	A12BD	Delay select for the BDL on address A[12]. In LPDDR3 mode, with address copy enabled, this is connected to CA_B[2].	`0xFD080564`
ACBDLR9	[13:8]	A13BD	Delay select for the BDL on address A[13]. In LPDDR3 mode, with address copy enabled, this is connected to CA_B[3].	`0xFD080564`
ACBDLR9	[21:16]	A14BD	Delay select for the BDL on address A[14]. In LPDDR3 mode, with address copy enabled, this is connected to CA_B[4].	`0xFD080564`
ACBDLR9	[29:24]	A15BD	Delay select for the BDL on address A[15]. In LPDDR3 mode, with address copy enabled, this is connected to CA_B[5].	`0xFD080564`

CA training also adjusts a locally calibrated delay line (LCDL) to center the clock within the CA bits. The results of this training are listed in Table 17-14.

Table 17-14: **AC Local Calibrated Delay Line Register (ACLCDLR)**

Bits	Name	Description	Address
[8:0]	ACD	Address/command delay for AC Macro 0: Delay select for the address/command (ACD) LCDL.	`0xFD080584`
[24:16]	ACD1	Address/command delay for AC Macro 1: delay select for the address/command (ACD) LCDL.	`0xFD080584`

Write Leveling

For signal integrity reasons, clock, address, and control signals in multiple SDRAM systems must be routed sequentially from one SDRAM to the next. This is called fly-by topology and helps to reduce the number of stubs and their length. The write data and strobe signals can, however, be routed with equal delay to each SDRAM. The fly-by topology can cause skew between the clock and the data strobe, making it difficult for the controller to maintain tDQSS, tDSS, and tDSH specification. Write leveling is used to compensate for this skew by aligning the clock with the data strobe at each SDRAM.

The PHY uses the write leveling feature, and feedback from the SDRAM, to adjust the DQS_t - DQS_c to CK_t - CK_c relationship. Write leveling has adjustable delay settings on DQS_t - DQS_c to align the rising edge of DQS_t - DQS_c with that of the clock at the DRAM pin. The DRAM asynchronously feeds back CK_t - CK_c (sampled with the rising edge of DQS_t - DQS_c) through the DQ bus. Writing leveling repeatedly delays DQS_t - DQS_c until a transition from 0 to 1 is detected. The DQS_t - DQS_c delay established through write leveling confirms the tDQSS specification.

The completion of write leveling is signaled by PGSR0[WLDONE] PGSR0[WLERR] indicates that an error occurred during write leveling. More detailed status information is listed in Table 17-15.

Table 17-15: **Write Leveling Status Information in DATX8 (DXnGSR0)**

Register	Bits	Name	Description	Address
DX0GSR0	[5]	WLDONE	Write leveling done: if set, indicates that the DATX8 has completed write leveling for byte 0.	`0xFD0807E0`
DX1GSR0	[5]	WLDONE	As described above, but for byte 1.	`0xFD0808E0`
DX2GSR0	[5]	WLDONE	As described above, but for byte 2.	`0xFD0809E0`
DX3GSR0	[5]	WLDONE	As described above, but for byte 3.	`0xFD080AE0`
DX4GSR0	[5]	WLDONE	As described above, but for byte 4.	`0xFD080BE0`
DX5GSR0	[5]	WLDONE	As described above, but for byte 5.	`0xFD080CE0`
DX6GSR0	[5]	WLDONE	As described above, but for byte 6.	`0xFD080DE0`
DX7GSR0	[5]	WLDONE	As described above, but for byte 7.	`0xFD080EE0`
DX8GSR0	[5]	WLDONE	As described above, but for byte 8.	`0xFD080FE0`
DX0GSR0	[6]	WLERR	Write leveling error: if set, indicates that there is a write leveling error in the DATX8 for byte 0.	`0xFD0807E0`

Table 17-15: **Write Leveling Status Information in DATX8 (DXnGSR0) *(Cont'd)***

Register	Bits	Name	Description	Address
DX1GSR0	[6]	WLERR	As described above, but for byte 1.	`0xFD0808E0`
DX2GSR0	[6]	WLERR	As described above, but for byte 2.	`0xFD0809E0`
DX3GSR0	[6]	WLERR	As described above, but for byte 3.	`0xFD080AE0`
DX4GSR0	[6]	WLERR	As described above, but for byte 4.	`0xFD080BE0`
DX5GSR0	[6]	WLERR	As described above, but for byte 5.	`0xFD080CE0`
DX6GSR0	[6]	WLERR	As described above, but for byte 6.	`0xFD080DE0`
DX7GSR0	[6]	WLERR	As described above, but for byte 7.	`0xFD080EE0`
DX8GSR0	[6]	WLERR	As described above, but for byte 8.	`0xFD080FE0`

Additional write leveling debugging information is listed in Table 17-16.

Table 17-16: **Write Leveling Debug Registers**

Register	Bits	Name	Description	Address
DX0GSR0	[15:7]	WLPRD	Write leveling period: returns the DDR clock period measured by the write leveling LCDL during calibration of byte 0. The measured period is used to generate the control of the write leveling pipeline, which is a function of the write-leveling delay and the clock period. This value is PVT compensated.	`0xFD0807E0`
DX1GSR0	[15:7]	WLPRD	As described above, but for byte 1.	`0xFD0808E0`
DX2GSR0	[15:7]	WLPRD	As described above, but for byte 2.	`0xFD0809E0`
DX3GSR0	[15:7]	WLPRD	As described above, but for byte 3.	`0xFD080AE0`
DX4GSR0	[15:7]	WLPRD	As described above, but for byte 4.	`0xFD080BE0`
DX5GSR0	[15:7]	WLPRD	As described above, but for byte 5.	`0xFD080CE0`
DX6GSR0	[15:7]	WLPRD	As described above, but for byte 6.	`0xFD080DE0`
DX7GSR0	[15:7]	WLPRD	As described above, but for byte 7.	`0xFD080EE0`
DX8GSR0	[15:7]	WLPRD	As described above, but for byte 8.	`0xFD080FE0`
DX0LCDLR0	[8:0]	WLD	Write leveling delay: delay select for the write leveling (WL) LCDL for byte 0.	`0xFD080780`
DX1LCDLR0	[8:0]	WLD	As described above, but for byte 1.	`0xFD080880`
DX2LCDLR0	[8:0]	WLD	As described above, but for byte 2.	`0xFD080980`
DX3LCDLR0	[8:0]	WLD	As described above, but for byte 3.	`0xFD080A80`
DX4LCDLR0	[8:0]	WLD	As described above, but for byte 4.	`0xFD080B80`
DX5LCDLR0	[8:0]	WLD	As described above, but for byte 5.	`0xFD080C80`
DX6LCDLR0	[8:0]	WLD	As described above, but for byte 6.	`0xFD080D80`
DX7LCDLR0	[8:0]	WLD	As described above, but for byte 7.	`0xFD080E80`
DX8LCDLR0	[8:0]	WLD	As described above, but for byte 8.	`0xFD080F80`

Table 17-16: **Write Leveling Debug Registers *(Cont'd)***

Register	Bits	Name	Description	Address
DX0LCDLR1	[8:0]	WDQD	Write data delay: delay select for the write data (WDQ) LCDL for byte 0.	`0xFD080784`
DX1LCDLR1	[8:0]	WDQD	As described above, but for byte 1.	`0xFD080884`
DX2LCDLR1	[8:0]	WDQD	As described above, but for byte 2.	`0xFD080984`
DX3LCDLR1	[8:0]	WDQD	As described above, but for byte 3.	`0xFD080A84`
DX4LCDLR1	[8:0]	WDQD	As described above, but for byte 4.	`0xFD080B84`
DX5LCDLR1	[8:0]	WDQD	As described above, but for byte 5.	`0xFD080C84`
DX6LCDLR1	[8:0]	WDQD	As described above, but for byte 6.	`0xFD080D84`
DX7LCDLR1	[8:0]	WDQD	As described above, but for byte 7.	`0xFD080E84`
DX8LCDLR1	[8:0]	WDQD	As described above, but for byte 8.	`0xFD080F84`
DX0GTR0	[19:16]	WLSL	Write leveling system latency: used to adjust the write latency of byte 0 after write leveling. Valid values: `0000`: Write latency = WL-1 DRAM clock period `0001`: Write latency = WL-0.5 DRAM clock period `0010`: Write latency = WL `0011`: Write latency = WL+0.5 DRAM clock period `0100`: Write latency = WL+1 DRAM clock period `0101`: Write latency = WL+1.5 DRAM clock period `0110`: Write latency = WL+2 DRAM clock period `0111`: Write latency = WL+2.5 DRAM clock period `1000`: Write latency = WL+3 DRAM clock period `1001`: Write latency = WL+3.5 DRAM clock period `1010`: Write latency = WL + 4 DRAM clock period `1011` - `1111`: RESERVED Write DQS are pipelined according to the table above. ***Note:*** Write data carries additional pipeline delay according to WDQSL.	`0xFD0807C0`
DX1GTR0	[19:16]	WLSL	As described above, but for byte 1.	`0xFD0808C0`
DX2GTR0	[19:16]	WLSL	As described above, but for byte 2.	`0xFD0809C0`
DX3GTR0	[19:16]	WLSL	As described above, but for byte 3.	`0xFD080AC0`
DX4GTR0	[19:16]	WLSL	As described above, but for byte 4.	`0xFD080BC0`
DX5GTR0	[19:16]	WLSL	As described above, but for byte 5.	`0xFD080CC0`
DX6GTR0	[19:16]	WLSL	As described above, but for byte 6.	`0xFD080DC0`
DX7GTR0	[19:16]	WLSL	As described above, but for byte 7.	`0xFD080EC0`
DX8GTR0	[19:16]	WLSL	As described above, but for byte 8.	`0xFD080FC0`

Table 17-16: **Write Leveling Debug Registers *(Cont'd)***

Register	Bits	Name	Description	Address
DX0GTR0	[26:24]	WDQSL	DQ write path latency pipeline for byte 0: write data is pipelined by (WLSL + WDQSL). Total write data pipeline is: [Write leveling system latency] + WDQSL/2 DRAM clock periods.	`0xFD0807C0`
DX1GTR0	[26:24]	WDQSL	Same as above, for byte 1.	`0xFD0808C0`
DX2GTR0	[26:24]	WDQSL	Same as above, for byte 2.	`0xFD0809C0`
DX3GTR0	[26:24]	WDQSL	Same as above, for byte 3.	`0xFD080AC0`
DX4GTR0	[26:24]	WDQSL	Same as above, for byte 4.	`0xFD080BC0`
DX5GTR0	[26:24]	WDQSL	Same as above, for byte 5.	`0xFD080CC0`
DX6GTR0	[26:24]	WDQSL	Same as above, for byte 6.	`0xFD080DC0`
DX7GTR0	[26:24]	WDQSL	Same as above, for byte 7.	`0xFD080EC0`
DX8GTR0	[26:24]	WDQSL	Same as above, for byte 8.	`0xFD080FC0`

Read Leveling

The read DQS strobes from the DRAM are ordinarily gated by the PHY to suppress noise and correctly capture read data. The precise alignment of the gate to the read data is a prerequisite for proper reads. Since delays, such as board trace lengths in the read path, are often imprecisely known, it is necessary to train the gate for a particular system. The PHY features a built-in read DQS strobe gate training unit that might be triggered as part of the initialization process.

Read leveling is an algorithm that works with the edge of the DQS. Gate and a delayed (by a few LCDL taps) gate sample the DQS signal. Gate starts from (a position of delay equal to zero) until the first edge of the DQS is found between the two sampling edges of the gate and delayed gate. Final position of the gate is found by adding a programmable (delay) offset to this value.

The completion of read leveling is signaled by PGSR0.QSGDONE. PGSR0.QSGERR indicates that an error occurred during read leveling. Errors are flagged in the DATX8 Rank Status register 1, as listed in Table 17-17.

Table 17-17: **DATX8 Rank Status Register 1 (DXnRSR1)**

Register	Bits	Name	Description	Address
DX0RSR1	[1:0]	RDLVLERR	Read leveling error: if set, indicates that there is an error in read leveling training of byte 0. One bit for each of the up to two ranks.	`0xFD0807D4`
DX1RSR1	[1:0]	RDLVLERR	Same as above, for byte 1.	`0xFD0808D4`
DX2RSR1	[1:0]	RDLVLERR	Same as above, for byte 2.	`0xFD0809D4`
DX3RSR1	[1:0]	RDLVLERR	Same as above, for byte 3.	`0xFD080AD4`

Table 17-17: **DATX8 Rank Status Register 1 (DXnRSR1)** ***(Cont'd)***

Register	Bits	Name	Description	Address
DX4RSR1	[1:0]	RDLVLERR	Same as above, for byte 4.	`0xFD080BD4`
DX5RSR1	[1:0]	RDLVLERR	Same as above, for byte 5.	`0xFD080CD4`
DX6RSR1	[1:0]	RDLVLERR	Same as above, for byte 6.	`0xFD080DD4`
DX7RSR1	[1:0]	RDLVLERR	Same as above, for byte 7.	`0xFD080ED4`
DX8RSR1	[1:0]	RDLVLERR	Same as above, for ECC byte.	`0xFD080FD4`

Additional read leveling debugging information is listed in Table 17-18.

Table 17-18: **Read Leveling Debug Registers**

Register	Bits	Name	Description	Address
DX0GSR0	[25:17]	GDQSPRD	Read DQS gating period: returns the DDR clock period measured by the read DQS gating LCDL during calibration of byte 0. This value is PVT compensated.	`0xFD0807E0`
DX1GSR0	[25:17]	GDQSPRD	Same as above, for byte 1.	`0xFD0808E0`
DX2GSR0	[25:17]	GDQSPRD	Same as above, for byte 2.	`0xFD0809E0`
DX3GSR0	[25:17]	GDQSPRD	Same as above, for byte 3.	`0xFD080AE0`
DX4GSR0	[25:17]	GDQSPRD	Same as above, for byte 4.	`0xFD080BE0`
DX5GSR0	[25:17]	GDQSPRD	Same as above, for byte 5.	`0xFD080CE0`
DX6GSR0	[25:17]	GDQSPRD	Same as above, for byte 6.	`0xFD080DE0`
DX7GSR0	[25:17]	GDQSPRD	Same as above, for byte 7.	`0xFD080EE0`
DX8GSR0	[25:17]	GDQSPRD	Same as above, for byte 8.	`0xFD080FE0`
DX0GTR0	[4:0]	DGSL	DQS gating system latency: this is used to increase the number of clock cycles need to expect valid DDR read data for byte 0. This is used to compensate for board delays and other system delays. Power-up default is `0x00` (i.e., no extra clock cycles required). Valid values are 0 to 18 and each increment adds a half SDRAM CK period.	`0xFD0807C0`
DX1GTR0	[4:0]	DGSL	Same as above, for byte 1.	`0xFD0808C0`
DX2GTR0	[4:0]	DGSL	Same as above, for byte 2.	`0xFD0809C0`
DX3GTR0	[4:0]	DGSL	Same as above, for byte 3.	`0xFD080AC0`
DX4GTR0	[4:0]	DGSL	Same as above, for byte 4.	`0xFD080BC0`
DX5GTR0	[4:0]	DGSL	Same as above, for byte 5.	`0xFD080CC0`
DX6GTR0	[4:0]	DGSL	Same as above, for byte 6.	`0xFD080DC0`
DX7GTR0	[4:0]	DGSL	Same as above, for byte 7.	`0xFD080EC0`
DX8GTR0	[4:0]	DGSL	Same as above, for byte 8.	`0xFD080FC0`
DX0LCDLR2	[8:0]	DQSGD	DQS gating delay: delay select for the DQS gating (DQSG) LCDL for byte 0.	`0xFD080788`
DX1LCDLR2	[8:0]	DQSGD	Same as above, for byte 1.	`0xFD080888`
DX2LCDLR2	[8:0]	DQSGD	Same as above, for byte 2.	`0xFD080988`
DX3LCDLR2	[8:0]	DQSGD	Same as above, for byte 3.	`0xFD080A88`
DX4LCDLR2	[8:0]	DQSGD	Same as above, for byte 4.	`0xFD080B88`
DX5LCDLR2	[8:0]	DQSGD	Same as above, for byte 5.	`0xFD080C88`
DX6LCDLR2	[8:0]	DQSGD	Same as above, for byte 6.	`0xFD080D88`

Table 17-18: **Read Leveling Debug Registers *(Cont'd)***

Register	Bits	Name	Description	Address
DX7LCDLR2	[8:0]	DQSGD	Same as above, for byte 7.	`0xFD080E88`
DX8LCDLR2	[8:0]	DQSGD	Same as above, for byte 8.	`0xFD080F88`

Write DQS2DQ Training (LPDDR4 only)

LPDDR4 memory devices use an unmatched DQS-DQ path to enable high-speed performance and save power. As a result, the DQS strobe is trained to arrive at the DQ latch center-aligned with the data eye. The DQ receiver latches the data present on the DQ bus when DQS reaches the latch. DQS2DQ training is accomplished by delaying the DQ signals relative to DQS such that the data eye arrives at the receiver latch centered on the DQS transition. DQS to DQ training is referred to as write training in the JEDEC® standard and write DQ training in the DFI standard.

DQS2DQ training completion is signaled by the PGSR0.DQS2DQDONE bit. If errors are encountered during training, PGSR0.DQS2DQERR is set. Per byte error flags are visible in DXnGSR2.DQS2DQERR, as listed in Table 17-19.

Table 17-19: **DQS2DQ Training Error Flags**

Register	Bits	Name	Description	Address
DX0GSR2	[15:12]	DQS2DQERR	Write DQS2DQ training error: if set, indicates that the DATX8 has encountered an error during execution of the write DQS2DQ training of byte 0. Each 2 bits indicate an error on one rank. (e.g. bits [13:12] indicate an error on rank 0) Status encoding is: `2'b00`: No error `2'b01`: oscillator results are all 0s `2'b10`: oscillator results are all 1s `1'b11`: oscillator results read timeout	`0xFD0807E8`
DX1GSR2	[15:12]	DQS2DQERR	Same as above, for byte 1.	`0xFD0808E8`
DX2GSR2	[15:12]	DQS2DQERR	Same as above, for byte 2.	`0xFD0809E8`
DX3GSR2	[15:12]	DQS2DQERR	Same as above, for byte 3.	`0xFD080AE8`
DX8GSR2	[15:12]	DQS2DQERR	Same as above, for ECC byte.	`0xFD080FE8`

Additional debugging info is available in the DXnLCDLR1 and DXnGTR0 registers, as listed in Table 17-20.

Table 17-20: **Write DQS2DQ Training Debug Registers**

Register	Bits	Name	Description	Address
DX0LCDLR1	[8:0]	WDQD	Write data delay: delay select for the write data (WDQ) LCDL for byte 0.	`0xFD080784`
DX1LCDLR1	[8:0]	WDQD	As described above, but for byte 1.	`0xFD080884`
DX2LCDLR1	[8:0]	WDQD	As described above, but for byte 2.	`0xFD080984`
DX3LCDLR1	[8:0]	WDQD	As described above, but for byte 3.	`0xFD080A84`
DX8LCDLR1	[8:0]	WDQD	As described above, but for ECC byte.	`0xFD080F84`
DX0GTR0	[26:24]	WDQSL	DQ write path latency pipeline for byte 0: Write data is pipelined by (WLSL + WDQSL). Total write data pipeline is: [Write leveling system latency] + WDQSL/2 DRAM clock periods.	`0xFD0807C0`
DX1GTR0	[26:24]	WDQSL	Same as above, for byte 1.	`0xFD0808C0`
DX2GTR0	[26:24]	WDQSL	Same as above, for byte 2.	`0xFD0809C0`
DX3GTR0	[26:24]	WDQSL	Same as above, for byte 3.	`0xFD080AC0`
DX8GTR0	[26:24]	WDQSL	Same as above, for ECC byte.	`0xFD080FC0`

Write Latency Adjustment

After write leveling, the strobe is aligned to the clock at each SDRAM, but it is not known if the strobe is aligned to the correct clock edge. To clear up this ambiguity, a second level of write leveling is used to determine if extra pipeline stages need to be added in the write path due to the write leveling or the board delays.

The write latency adjustment writes a fixed-pattern back-to-back sequence of two BL16s, appended with extra DQS pulses at the end of the last BL16 to obtain a sufficiently long pattern so that nine, previously ambiguous, system write latency situations can be uniquely distinguished. The algorithm writes this data using the minimal DFI pipeline depth.

The distinction is performed by counting the number of one beats in odd and even DQ lines. After determining the write latency, a second sequence of writes and reads are issued to validate the computed latency adjustment setting. For a multi-rank system, this sequence is repeated for each rank.

If an error is detected, the PGSR0.WLAERR field is set. Warnings and errors are flagged in DXnRSR2.WLAWN and DXnRSR3.WLAERR, respectively. See Table 17-21

Table 17-21: **DATX8 Rank Status Registers 2 and 3 (DXnRSR2 and DXnRSR3)**

Register	Bits	Name	Description	Address
DX0RSR2	[1:0]	WLAWN	Write latency adjustment "DQS off on some DQ lines". Warning: One bit per rank indicates that, for that rank, the WLA algorithm found some DQ lines where the read data sequence did not match the expected comparison signatures for byte 0.	`0xFD0807D8`
DX1RSR2	[1:0]	WLAWN	Same as above, for byte 1.	`0xFD0808D8`
DX2RSR2	[1:0]	WLAWN	Same as above, for byte 2.	`0xFD0809D8`
DX3RSR2	[1:0]	WLAWN	Same as above, for byte 3.	`0xFD080AD8`
DX4RSR2	[1:0]	WLAWN	Same as above, for byte 4.	`0xFD080BD8`
DX5RSR2	[1:0]	WLAWN	Same as above, for byte 5.	`0xFD080CD8`
DX6RSR2	[1:0]	WLAWN	Same as above, for byte 6.	`0xFD080DD8`
DX7RSR2	[1:0]	WLAWN	Same as above, for byte 7.	`0xFD080ED8`
DX8RSR2	[1:0]	WLAWN	Same as above, for byte 8.	`0xFD080FD8`
DX0RSR3	[1:0]	WLAERR	Write latency adjustment error: indicates, for each of the system ranks, that an error occurred in the WLA algorithm for byte 0.	`0xFD0807DC`
DX1RSR3	[1:0]	WLAERR	Same as above, for byte 1.	`0xFD0808DC`
DX2RSR3	[1:0]	WLAERR	Same as above, for byte 2.	`0xFD0809DC`
DX3RSR3	[1:0]	WLAERR	Same as above, for byte 3.	`0xFD080ADC`
DX4RSR3	[1:0]	WLAERR	Same as above, for byte 4.	`0xFD080BDC`
DX5RSR3	[1:0]	WLAERR	Same as above, for byte 5.	`0xFD080CDC`
DX6RSR3	[1:0]	WLAERR	Same as above, for byte 6.	`0xFD080DDC`
DX7RSR3	[1:0]	WLAERR	Same as above, for byte 7.	`0xFD080EDC`
DX8RSR3	[1:0]	WLAERR	Same as above, for byte 8.	`0xFD080FDC`

The write latency adjustment changes the same outputs controlled by write leveling. See Table 17-16 for more debugging information.

Data Eye Training

As bit rates increase to 2133 Mb/s and beyond, maintaining timing margins in the DDR interfaces becomes more difficult. The PHY solution includes delay lines to compensate for per-bit skew due to factors such as PHY to I/O routing skews, package skews, and PCB skew.

The PHY contains automatic training sequences to perform read and write deskew, which align the data bits to the DQ bit with the longest delay using bit delay lines (BDL). After performing bit deskew, the read and write eye centering training is executed to place the strobe in the center of the eye defined by the bits in the respective byte.

During read or write eye training each individual byte lane has a register DXnGSR2 that contains error and warning status flags for each of the eye training algorithms.

Error conditions are fatal and the PHY will immediately terminate data training. Within the DXnGSR2 register, a bit field named ESTAT contains an error status code. This error status code identifies the sub-step where the failure occurred and the algorithm descriptions provide the conditions for the error and the associated error status code.

A warning status generally indicates that either the right or left edges of the data eye could not be detected. This can occur for a variety of reasons but this is more likely to occur during write bit deskew or write eye centering. When this warning occurs, the algorithm has assumed that the edge of the eye has been detected when it has exhausted the available DDL resources. This can result in a skewed center positioning of the DQS/DQS# within the data eye.

Read bit deskew, write bit deskew, read eye training, and write eye training are the data eye training steps.

Read Bit Deskew

The read bit deskew algorithm is performed in parallel for all byte lanes and requires write and read access to memory. The goal of the PHY read bit deskew algorithm is to align a 0-to-1 transition on each of the data bits in the read path. An initial pattern is written into memory, read back, and then evaluated. Then per-bit delay lines are used to align all the data bits to each other. After deskewing, another read is executed to confirm data integrity.

Read bit deskew completion is signaled by the PGSR0.RDDONE bit. The high-level error flag is PGSR0.RDERR. Additional debugging information is listed in Table 17-22 and Table 17-23.

Table 17-22: **DATX8 General Status Register 2 (DXnGSR2)**

Register	Bits	Name	Description	Address
DX0GSR2	[0]	RDERR	Read bit deskew error: if set, indicates that the DATX8 has encountered an error during execution of the read bit deskew training of byte 0.	`0xFD0807E8`
DX1GSR2	[0]	RDERR	Same as above, for byte 1.	`0xFD0808E8`
DX2GSR2	[0]	RDERR	Same as above, for byte 2.	`0xFD0809E8`
DX3GSR2	[0]	RDERR	Same as above, for byte 3.	`0xFD080AE8`
DX4GSR2	[0]	RDERR	Same as above, for byte 4.	`0xFD080BE8`
DX5GSR2	[0]	RDERR	Same as above, for byte 5.	`0xFD080CE8`
DX6GSR2	[0]	RDERR	Same as above, for byte 6.	`0xFD080DE8`
DX7GSR2	[0]	RDERR	Same as above, for byte 7.	`0xFD080EE8`
DX8GSR2	[0]	RDERR	Same as above, for byte 8.	`0xFD080FE8`
DX0GSR2	[1]	RDWN	Read bit deskew warning: if set, indicates that the DATX8 has encountered a warning during execution of the read bit deskew training of byte 0.	`0xFD0807E8`
DX1GSR2	[1]	RDWN	Same as above, for byte 1.	`0xFD0808E8`
DX2GSR2	[1]	RDWN	Same as above, for byte 2.	`0xFD0809E8`
DX3GSR2	[1]	RDWN	Same as above, for byte 3.	`0xFD080AE8`
DX4GSR2	[1]	RDWN	Same as above, for byte 4.	`0xFD080BE8`
DX5GSR2	[1]	RDWN	Same as above, for byte 5.	`0xFD080CE8`
DX6GSR2	[1]	RDWN	Same as above, for byte 6.	`0xFD080DE8`
DX7GSR2	[1]	RDWN	Same as above, for byte 7.	`0xFD080EE8`
DX8GSR2	[1]	RDWN	Same as above, for byte 8.	`0xFD080FE8`
DX0GSR2	[11:8]	ESTAT	Error status: if an error occurred for byte 0 as indicated by RDERR, the error status code can provide additional information regarding when the error occurred during the algorithm execution.	`0xFD0807E8`
DX1GSR2	[11:8]	ESTAT	Same as above, for byte 1.	`0xFD0808E8`

Table 17-22: **DATX8 General Status Register 2 (DXnGSR2)** ***(Cont'd)***

Register	Bits	Name	Description	Address
DX2GSR2	[11:8]	ESTAT	Same as above, for byte 2.	0xFD0809E8
DX3GSR2	[11:8]	ESTAT	Same as above, for byte 3.	0xFD080AE8
DX4GSR2	[11:8]	ESTAT	Same as above, for byte 4.	0xFD080BE8
DX5GSR2	[11:8]	ESTAT	Same as above, for byte 5.	0xFD080CE8
DX6GSR2	[11:8]	ESTAT	Same as above, for byte 6.	0xFD080DE8
DX7GSR2	[11:8]	ESTAT	Same as above, for byte 7.	0xFD080EE8
DX8GSR2	[11:8]	ESTAT	Same as above, for byte 8.	0xFD080FE8

Table 17-23: **Read Bit Deskew Error Indications**

PGSR0.RDERR	DXnGSR2.RDERR	DXnGSR2.ESTAT	PGSR0. RDDONE	Error Condition
1	1	0000	1	Initial read data is skewed by more than three beats of data prior to any deskew.
1	1	0001	1	Read DQS/DQS# is too early relative to data, and during deskew, DQS/DQS# LCDL is at maximum value and any read DQ BDL is at minimum value.
1	1	0010	1	While searching for left edge of read data eye, DQS/DQS# LCDL is at the minimum value and any read DQ BDL is at the maximum value.
1	1	0101	1	While searching for right edge of read data eye, DQS/DQS# LCDL is at the maximum value and any read DQ BDL is at the minimum value.
1	1	0111	1	Read data miscompare after read bit deskew.

The results of read bit deskew can be viewed in the DXnBDLR3, DXnBDLR4, and DXnBDLR5 registers, as listed in Table 17-24.

Table 17-24: **Read Bit Deskew Results Registers**

Register	Bits	Name	Description	Address
DXnBDLR3	[5:0]	DQ0RBD	DQ0 read bit delay: delay select for the BDL on DQ0 read path.	`FD080750, FD080850, FD080950, FD080A50, FD080B50, FD080C50, FD080D50, FD080E50, FD080F50`
DXnBDLR3	[13:8]	DQ1RBD	DQ1 read bit delay: delay select for the BDL on DQ1 read path.	`FD080750, FD080850, FD080950, FD080A50, FD080B50, FD080C50, FD080D50, FD080E50, FD080F50`
DXnBDLR3	[21:16]	DQ2RBD	DQ2 read bit delay: delay select for the BDL on DQ2 read path.	`FD080750, FD080850, FD080950, FD080A50, FD080B50, FD080C50, FD080D50, FD080E50, FD080F50`
DXnBDLR3	[29:24]	DQ3RBD	DQ3 read bit delay: delay select for the BDL on DQ3 read path.	`FD080750, FD080850, FD080950, FD080A50, FD080B50, FD080C50, FD080D50, FD080E50, FD080F50`
DXnBDLR4	[5:0]	DQ4RBD	DQ4 read bit delay: delay select for the BDL on DQ4 read path.	`FD080754, FD080854, FD080954, FD080A54, FD080B54, FD080C54, FD080D54, FD080E54, FD080F54`
DXnBDLR4	[13:8]	DQ5RBD	DQ5 read bit delay: delay select for the BDL on DQ5 read path.	`FD080754, FD080854, FD080954, FD080A54, FD080B54, FD080C54, FD080D54, FD080E54, FD080F54`
DXnBDLR4	[21:16]	DQ6RBD	DQ6 read bit delay: delay select for the BDL on DQ6 read path.	`FD080754, FD080854, FD080954, FD080A54, FD080B54, FD080C54, FD080D54, FD080E54, FD080F54`
DXnBDLR4	[29:24]	DQ7RBD	DQ7 read bit delay: delay select for the BDL on DQ7 read path.	`FD080754, FD080854, FD080954, FD080A54, FD080B54, FD080C54, FD080D54, FD080E54, FD080F54`
DXnBDLR5	[5:0]	DMRBD	DM read bit delay: delay select for the BDL on DM read path.	`FD080758, FD080858, FD080958, FD080A58, FD080B58, FD080C58, FD080D58, FD080E58, FD080F58`

Write Bit Deskew

The write bit deskew algorithm is performed in parallel for all byte lanes and requires write and read access to memory. The goal of the PHY write bit deskew algorithm is to align a 0-to-1 transition on each of the data bits in the write path. An initial pattern is written into memory, read back, and then evaluated. Then per-bit delay lines are used to align all the data bits to each other. After deskewing, another read is executed to confirm data integrity.

Write bit deskew completion is signaled by the PGSR0.WDDONE bit. The high-level error flag is PGSR0.WDERR. Additional debugging information is listed in Table 17-25 and Table 17-26.

Table 17-25: **DATX8 General Status Register (DXnGSR2)**

Register	Bits	Name	Description	Address
DX0GSR2	[2]	WDERR	Write bit deskew error: if set, indicates that the DATX8 has encountered an error during execution of the write bit deskew training of byte 0.	`FD0807E8`
DX1GSR2	[2]	WDERR	Same as above, for byte 1.	`FD0808E8`
DX2GSR2	[2]	WDERR	Same as above, for byte 2.	`FD0809E8`
DX3GSR2	[2]	WDERR	Same as above, for byte 3.	`FD080AE8`
DX4GSR2	[2]	WDERR	Same as above, for byte 4.	`FD080BE8`
DX5GSR2	[2]	WDERR	Same as above, for byte 5.	`FD080CE8`
DX6GSR2	[2]	WDERR	Same as above, for byte 6.	`FD080DE8`
DX7GSR2	[2]	WDERR	Same as above, for byte 7.	`FD080EE8`
DX8GSR2	[2]	WDERR	Same as above, for byte 8.	`FD080FE8`
DX0GSR2	[3]	WDWN	Write bit deskew warning: if set, indicates that the DATX8 has encountered a warning during execution of the write bit deskew training of byte 0.	`FD0807E8`
DX1GSR2	[3]	WDWN	Same as above, for byte 1.	`FD0808E8`
DX2GSR2	[3]	WDWN	Same as above, for byte 2.	`FD0809E8`
DX3GSR2	[3]	WDWN	Same as above, for byte 3.	`FD080AE8`
DX4GSR2	[3]	WDWN	Same as above, for byte 4.	`FD080BE8`
DX5GSR2	[3]	WDWN	Same as above, for byte 5.	`FD080CE8`
DX6GSR2	[3]	WDWN	Same as above, for byte 6.	`FD080DE8`
DX7GSR2	[3]	WDWN	Same as above, for byte 7.	`FD080EE8`
DX8GSR2	[3]	WDWN	Same as above, for byte 8.	`FD080FE8`
DX0GSR2	[11:8]	ESTAT	Error status: If an error occurred for byte 0 as indicated by WDERR, the error status code can provide additional information regarding when the error occurred during the algorithm execution.	`FD0807E8`

Table 17-25: **DATX8 General Status Register (DXnGSR2) *(Cont'd)***

Register	Bits	Name	Description	Address
DX1GSR2	[11:8]	ESTAT	Same as above, for byte 1.	FD0808E8
DX2GSR2	[11:8]	ESTAT	Same as above, for byte 2.	FD0809E8
DX3GSR2	[11:8]	ESTAT	Same as above, for byte 3.	FD080AE8
DX4GSR2	[11:8]	ESTAT	Same as above, for byte 4.	FD080BE8
DX5GSR2	[11:8]	ESTAT	Same as above, for byte 5.	FD080CE8
DX6GSR2	[11:8]	ESTAT	Same as above, for byte 6.	FD080DE8
DX7GSR2	[11:8]	ESTAT	Same as above, for byte 7.	FD080EE8
DX8GSR2	[11:8]	ESTAT	Same as above, for byte 8.	FD080FE8

Table 17-26: **Write Bit Deskew Error Indications**

PGSR0.WDERR	DXnGSR2.WDERR	DXnGSR2.ESTAT	PGSR0. WDDONE	Error Condition
1	1	0000	1	Initial write data is skewed by more than three beats of data prior to any deskew.
1	1	0001	1	Write DQS/DQS# is too early relative to data, and during deskew, DQ LCDL is at minimum value and any write DQ BDL is at minimum value.
1	1	0010	1	While searching for left edge of write data eye, DQ LCDL is at the maximum value and any write DQ BDL is at the maximum value.
1	1	1100	1	Read data miscompare after write bit deskew.

The results of write bit deskew can viewed in the DXnBDLR0, DXnBDLR1, and DXnBDLR2 registers, as listed in Table 17-27.

Table 17-27: **Write Bit Deskew Error Indications**

Register	Bits	Name	Description	Address
DXnBDLR0	[5:0]	DQ0WBD	DQ0 write bit delay: delay select for the BDL on DQ0 write path.	FD080740, FD080840, FD080940, FD080A40, FD080B40, FD080C40, FD080D40, FD080E40, FD080F40
DXnBDLR0	[13:8]	DQ1WBD	DQ1 write bit delay: delay select for the BDL on DQ1 write path.	FD080740, FD080840, FD080940, FD080A40, FD080B40, FD080C40, FD080D40, FD080E40, FD080F40
DXnBDLR0	[21:16]	DQ2WBD	DQ2 write bit delay: delay select for the BDL on DQ2 write path.	FD080740, FD080840, FD080940, FD080A40, FD080B40, FD080C40, FD080D40, FD080E40, FD080F40
DXnBDLR0	[29:24]	DQ3WBD	DQ3 write bit delay: delay select for the BDL on DQ3 write path.	FD080740, FD080840, FD080940, FD080A40, FD080B40, FD080C40, FD080D40, FD080E40, FD080F40
DXnBDLR1	[5:0]	DQ4WBD	DQ4 write bit delay: delay select for the BDL on DQ4 write path.	FD080744, FD080844, FD080944, FD080A44, FD080B44, FD080C44, FD080D44, FD080E44, FD080F44
DXnBDLR1	[13:8]	DQ5WBD	DQ5 write bit delay: delay select for the BDL on DQ5 write path.	FD080744, FD080844, FD080944, FD080A44, FD080B44, FD080C44, FD080D44, FD080E44, FD080F44
DXnBDLR1	[21:16]	DQ6WBD	DQ6 write bit delay: delay select for the BDL on DQ6 write path.	FD080744, FD080844, FD080944, FD080A44, FD080B44, FD080C44, FD080D44, FD080E44, FD080F44
DXnBDLR1	[29:24]	DQ7WBD	DQ7 write bit delay: delay select for the BDL on DQ7 write path.	FD080744, FD080844, FD080944, FD080A44, FD080B44, FD080C44, FD080D44, FD080E44, FD080F44
DXnBDLR2	[5:0]	DMWBD	DM write bit delay: delay select for the BDL on DM write path.	FD080748, FD080848, FD080948, FD080A48, FD080B48, FD080C48, FD080D48, FD080E48, FD080F48

Read Eye Centering

The read eye centering algorithm is performed in parallel for all byte lanes and requires write and read access to memory. The goal of the PHY read eye centering algorithm is to center the strobe within the data eye in each byte in the read path. An initial pattern is written into memory, read back, and then evaluated. Then read DQS/DQS# is moved to find the left and right edges of the read eye, and the optimal position is calculated. After centering, another read is executed to confirm data integrity.

Read eye centering completion is signaled by the PGSR0.REDONE bit. The high-level error flag is PGSR0.REERR. Additional debugging information is listed in Table 17-28 and Table 17-28.

Table 17-28: **DATX8 General Status Register 2 (DXnGSR2)**

Register	Bits	Name	Description	Address
DX0GSR2	[4]	REERR	Read eye centering error: if set, indicates that the DATX8 has encountered an error during execution of the read eye centering training of byte 0.	`FD0807E8`
DX1GSR2	[4]	REERR	Same as above, for byte 1.	`FD0808E8`
DX2GSR2	[4]	REERR	Same as above, for byte 2.	`FD0809E8`
DX3GSR2	[4]	REERR	Same as above, for byte 3.	`FD080AE8`
DX4GSR2	[4]	REERR	Same as above, for byte 4.	`FD080BE8`
DX5GSR2	[4]	REERR	Same as above, for byte 5.	`FD080CE8`
DX6GSR2	[4]	REERR	Same as above, for byte 6.	`FD080DE8`
DX7GSR2	[4]	REERR	Same as above, for byte 7.	`FD080EE8`
DX8GSR2	[4]	REERR	Same as above, for byte 8.	`FD080FE8`
DX0GSR2	[5]	REWN	Read eye centering warning: if set, indicates that the DATX8 has encountered a warning during execution of the read eye centering training of byte 0.	`FD0807E8`
DX1GSR2	[5]	REWN	Same as above, for byte 1.	`FD0808E8`
DX2GSR2	[5]	REWN	Same as above, for byte 2.	`FD0809E8`
DX3GSR2	[5]	REWN	Same as above, for byte 3.	`FD080AE8`
DX4GSR2	[5]	REWN	Same as above, for byte 4.	`FD080BE8`
DX5GSR2	[5]	REWN	Same as above, for byte 5.	`FD080CE8`
DX6GSR2	[5]	REWN	Same as above, for byte 6.	`FD080DE8`
DX7GSR2	[5]	REWN	Same as above, for byte 7.	`FD080EE8`
DX8GSR2	[5]	REWN	Same as above, for byte 8.	`FD080FE8`

Table 17-28: **DATX8 General Status Register 2 (DXnGSR2) *(Cont'd)***

Register	Bits	Name	Description	Address
DX0GSR2	[11:8]	ESTAT	Error status: If an error occurred for byte 0 as indicated by REERR, the error status code can provide additional information regarding when the error occurred during the algorithm execution.	`FD0807E8`
DX1GSR2	[11:8]	ESTAT	Same as above, for byte 1.	`FD0808E8`
DX2GSR2	[11:8]	ESTAT	Same as above, for byte 2.	`FD0809E8`
DX3GSR2	[11:8]	ESTAT	Same as above, for byte 3.	`FD080AE8`
DX4GSR2	[11:8]	ESTAT	Same as above, for byte 4.	`FD080BE8`
DX5GSR2	[11:8]	ESTAT	Same as above, for byte 5.	`FD080CE8`
DX6GSR2	[11:8]	ESTAT	Same as above, for byte 6.	`FD080DE8`
DX7GSR2	[11:8]	ESTAT	Same as above, for byte 7.	`FD080EE8`
DX8GSR2	[11:8]	ESTAT	Same as above, for byte 8.	`FD080FE8`

Table 17-29: **Read Eye Centering Error Indications**

PGSR0.REERR	DXnGSR2.REERR	DXnGSR2.ESTAT	PGSR0. REDONE	Error Condition
1	1	`0000`	1	Initial read data miscompare before centering.
1	1	`0101`	1	Read data miscompare after read eye centering.

The results of the read eye centering can be viewed in the DXnLCDLR3 and DXnLCDLR4 registers, as listed in Table 17-30.

Table 17-30: **Read Eye Centering Results Registers**

Register	Bits	Name	Description	Address
DXnLCDLR3	[8:0]	RDQSD	Read DQS delay: delay select for the read DQS (RDAS) LCDL for each byte.	`FD08078C`, `FD08088C`, `FD08098C`, `FD080A8C`, `FD080B8C`, `FD080C8C`, `FD080D8C`, `FD080E8C`, `FD080F8C`
DXnLCDLR4	[8:0]	RDQSND	Read DQSN delay: delay select for the read DQSN (RDQSN) LCDL for each byte.	`FD080790`, `FD080890`, `FD080990`, `FD080A90`, `FD080B90`, `FD080C90`, `FD080D90`, `FD080E90`, `FD080F90`

Write Eye Centering

The write eye centering algorithm is performed in parallel for all byte lanes and requires write and read access to memory. The goal of the PHY write eye centering algorithm is to center the strobe within the data eye in each byte in the write path. An initial pattern is written into memory, read back, and then evaluated. Then write DQ is moved to find the left and right edges of the write eye, and the optimal position is calculated. After centering, another read is executed to confirm data integrity.

Write eye centering completion is signaled by the PGSR0.WEDONE bit. The high-level error flag is PGSR0.WEERR. Additional debugging information is available in DXnGSR2 as listed in Table 17-31 and Table 17-32.

Table 17-31: **DATX8 General Status Register 2 (DXnGSR2)**

Register	Bits	Name	Description	Address
DX0GSR2	[6]	WEERR	Write eye centering error: if set, indicates that the DATX8 has encountered an error during execution of the write eye centering training of byte 0.	FD0807E8
DX1GSR2	[6]	WEERR	Same as above, for byte 1.	FD0808E8
DX2GSR2	[6]	WEERR	Same as above, for byte 2.	FD0809E8
DX3GSR2	[6]	WEERR	Same as above, for byte 3.	FD080AE8
DX4GSR2	[6]	WEERR	Same as above, for byte 4.	FD080BE8
DX5GSR2	[6]	WEERR	Same as above, for byte 5.	FD080CE8
DX6GSR2	[6]	WEERR	Same as above, for byte 6.	FD080DE8
DX7GSR2	[6]	WEERR	Same as above, for byte 7.	FD080EE8
DX8GSR2	[6]	WEERR	Same as above, for byte 8.	FD080FE8
DX0GSR2	[7]	WEWN	Write eye centering warning: if set, indicates that the DATX8 has encountered a warning during execution of the write eye centering training of byte 0.	FD0807E8
DX1GSR2	[7]	WEWN	Same as above, for byte 1.	FD0808E8
DX2GSR2	[7]	WEWN	Same as above, for byte 2.	FD0809E8
DX3GSR2	[7]	WEWN	Same as above, for byte 3.	FD080AE8
DX4GSR2	[7]	WEWN	Same as above, for byte 4.	FD080BE8
DX5GSR2	[7]	WEWN	Same as above, for byte 5.	FD080CE8
DX6GSR2	[7]	WEWN	Same as above, for byte 6.	FD080DE8
DX7GSR2	[7]	WEWN	Same as above, for byte 7.	FD080EE8
DX8GSR2	[7]	WEWN	Same as above, for byte 8.	FD080FE8
DX0GSR2	[11:8]	ESTAT	Error Status: If an error occurred for byte 0 as indicated by WEERR, the error status code can provide additional information on when the error occurred during the algorithm execution.	FD0807E8

Table 17-31: **DATX8 General Status Register 2 (DXnGSR2) *(Cont'd)***

Register	Bits	Name	Description	Address
DX1GSR2	[11:8]	ESTAT	Same as above, for byte 1.	FD0808E8
DX2GSR2	[11:8]	ESTAT	Same as above, for byte 2.	FD0809E8
DX3GSR2	[11:8]	ESTAT	Same as above, for byte 3.	FD080AE8
DX4GSR2	[11:8]	ESTAT	Same as above, for byte 4.	FD080BE8
DX5GSR2	[11:8]	ESTAT	Same as above, for byte 5.	FD080CE8
DX6GSR2	[11:8]	ESTAT	Same as above, for byte 6.	FD080DE8
DX7GSR2	[11:8]	ESTAT	Same as above, for byte 7.	FD080EE8
DX8GSR2	[11:8]	ESTAT	Same as above, for byte 8.	FD080FE8

Table 17-32: **Write Eye Centering Error Indications**

PGSR0.WEERR	DXnGSR2.WEERR	DXnGSR2.ESTAT	PGSR0. WEDONE	Error Condition
1	1	0000	1	Initial read data miscompare before centering
1	1	0101	1	Read data miscompare after write eye centering.

The results of Write Eye Centering can be viewed in the DXnGTR0 and DXnLCDLR1 registers listed in Table 17-33.

Table 17-33: **Write Eye Centering Results Registers**

Register	Bits	Name	Description	Address
DXnGTR0	[26:24]	WDQSL	DQ write path latency pipeline: Write data is pipelined by (WLSL + WDQSL). Total write data pipeline is: [Write leveling system latency] + WDQSL/2 DRAM clock periods. This value is adjusted by LPDDR4 tDQS2DQ training and write eye centering. Any update in DXnLCDLR1.WDQD updates this field after 20 ctl_clk clock cycles. Reading this field shows the number of pipelines (UI delays) written into the DXnLCDLR1.WDQD field. Ensure this field is never overwritten by software. Writing into this field changes (corrupts) the total write DQ delay written into the DXnLCDLR1.WDQD field.	`FD0807C0, FD0808C0, FD0809C0, FD080AC0, FD080BC0, FD080CC0, FD080DC0, FD080EC0, FD080FC0`
DXnLCDLR1	[8:0]	WDQD	Write data delay: delay select for the write data (WDQ) LCDL for each byte. The WDQ LCDL register is automatically updated after DDL calibration (by Tck/4) and after write leveling when write leveling is performed. Total delay should be written into this field. It overrides the delay set by hardware. Delay written in this field is converted to following two elements after 20 ctl_clk clock cycles: 1. Number of UI delays (pipelines) added to write dq path that can be read from DxnGTR0.WDQSL field. 2. The remainder of the delay that is the number of LCDL tap delays (written delay - DxnGTR0.WDQSL * one UI period). It is smaller than one UI and is available to read in this field. Reading this field returns the delay in item 2. This field should be programmed only after running calibration.	`FD080784, FD080884, FD080984, FD080A84, FD080B84, FD080C84, FD080D84, FD080E84, FD080F84`

VREF Training (DDR4 and LPDDR4 only)

The write and read eyes should be as wide as possible to provide a stable and robust memory access. The eye position depends upon LCDL, as well as VREF values. The write and read data eye training is used to find out the best eye position by changing LCDL values with an initial calculated and programmed VREF setting.

VREF training is used to determine a range of VREF values where memory interface (write and read) is stable and then determine an optimum write and read eye position.

These types of VREF training are supported:

- DRAM VREF training: this training is used to optimize the write eye by sweeping DRAM VrefDQ values inside memory.
- Host VREF training: this training is used to optimize the read eye by sweeping the PHY I/O's VREF setting.

VREF training completion is signaled by the PGSR0.VDONE bit. If errors are encountered during training, PGSR0.VERR is set. Per byte error flags are visible in DXnGSR3, as listed in Table 17-34.

Table 17-34: **DATX8 General Status Register 3 (DXnGSR3)**

Register	Bits	Name	Description	Address
DX0GSR3	[9:8]	HVERR	Host VREF training error: indicates if set that there is an error in VREF Training of byte 0. Each bit indicates an error for one rank.	`FD0807EC`
DX1GSR3	[9:8]	HVERR	Same as above, for byte 1.	`FD0808EC`
DX2GSR3	[9:8]	HVERR	Same as above, for byte 2.	`FD0809EC`
DX3GSR3	[9:8]	HVERR	Same as above, for byte 3.	`FD080AEC`
DX4GSR3	[9:8]	HVERR	Same as above, for byte 4.	`FD080BEC`
DX5GSR3	[9:8]	HVERR	Same as above, for byte 5.	`FD080CEC`
DX6GSR3	[9:8]	HVERR	Same as above, for byte 6.	`FD080DEC`
DX7GSR3	[9:8]	HVERR	Same as above, for byte 7.	`FD080EEC`
DX8GSR3	[9:8]	HVERR	Same as above, for byte 8.	`FD080FEC`
DX0GSR3	[17:16]	DVERR	DRAM VREF training error: indicates if set that there is an error in VREF Training of byte 0. Each bit indicates an error for one rank.	`FD0807EC`
DX1GSR3	[17:16]	DVERR	Same as above, for byte 1.	`FD0808EC`
DX2GSR3	[17:16]	DVERR	Same as above, for byte 2.	`FD0809EC`
DX3GSR3	[17:16]	DVERR	Same as above, for byte 3.	`FD080AEC`
DX4GSR3	[17:16]	DVERR	Same as above, for byte 4.	`FD080BEC`

Table 17-34: **DATX8 General Status Register 3 (DXnGSR3)** ***(Cont'd)***

Register	Bits	Name	Description	Address
DX5GSR3	[17:16]	DVERR	Same as above, for byte 5.	`FD080CEC`
DX6GSR3	[17:16]	DVERR	Same as above, for byte 6.	`FD080DEC`
DX7GSR3	[17:16]	DVERR	Same as above, for byte 7.	`FD080EEC`
DX8GSR3	[17:16]	DVERR	Same as above, for byte 8.	`FD080FEC`

DX0GSR3	[26:24]	ESTAT	VREF training error status code: indicates which phase of error check failed. Valid status encodings are:
			ESTAT[0] = Initial VREF check failed.
			ESTAT[1] = Final check for DRAM VREF failed.
			ESTAT[2] = Final check for Host VREF failed.
				`FD0807EC`

DX1GSR3	[26:24]	ESTAT	Same as above, for byte 1.	`FD0808EC`
DX2GSR3	[26:24]	ESTAT	Same as above, for byte 2.	`FD0809EC`
DX3GSR3	[26:24]	ESTAT	Same as above, for byte 3.	`FD080AEC`
DX4GSR3	[26:24]	ESTAT	Same as above, for byte 4.	`FD080BEC`
DX5GSR3	[26:24]	ESTAT	Same as above, for byte 5.	`FD080CEC`
DX6GSR3	[26:24]	ESTAT	Same as above, for byte 6.	`FD080DEC`
DX7GSR3	[26:24]	ESTAT	Same as above, for byte 7.	`FD080EEC`

Register Overview

DDR QoS Control Registers

Table 17-35: **DDR QoS Control Registers**

Register Name	Register Description
PORT_TYPE	Set port type register.
QOS_CTRL	Set port type register.
RD_HPR_THRSLD	Set value for read high-priority read (HPR) CAM threshold.
RD_LPR_THRSLD	Set value for read low-priority read (LPR) CAM threshold.
WR_THRSLD	Set value for write CAM threshold.
ZQCS_CTRL0	ZQCS control register 0.
ZQCS_CTRL1	ZQCS control register 1.
ZQCS_STATUS	ZQCS status register.

Table 17-35: **DDR QoS Control Registers *(Cont'd)***

Register Name	Register Description
DDRC_EXT_REFRESH	DDRC external refresh control register.
DDRC_EXT_REFRESH_RANK0_REQ	
DDRC_EXT_REFRESH_RANK1_REQ	
QOS_IRQ_STATUS	Interrupt status register for intrN. This is a sticky register that holds the value of the interrupt until cleared with a value of `1`.
QOS_IRQ_MASK	Interrupt mask register for intrN. This is a read-only location and can be atomically altered by either the IDR or the IER.
QOS_IRQ_ENABLE	Interrupt enable register. A write of zero to this location unmasks the interrupt. (IMR: `0`)
QOS_IRQ_DISABLE	Interrupt disable register. A write of one to this location masks the interrupt. (IMR: `1`)
DDRC_URGENT	DDRC urgent sideband signal control register.
DDRC_QVN_CTRL	DDRC QVN control register.
DDRC_MRR_STATUS	DDRC MRR register status.
DDRC_MRR_DATA{0:11}	DDRC MRR register data {0:11}.
DDR_CLK_CTRL	DDR subsystem clock control.

DDR Controller Registers

Table 17-36: **DDR Controller Registers**

Register Name	Register Description
MSTR(2)	Master register.
STAT	Operating mode status register.
MRCTRL{0:2}	Mode register read/write control register {0:2}
MRSTAT	Mode register read/write status register.
DERATEEN	Temperature derate enable register.
DERATEINT(4)	Temperature derate interval register.
PWRCTL	Low-power control register.
PWRTMG	Low-power timing register.
HWLPCTL(3)	Hardware low-power control register.
RFSHCTL0	Refresh control register 0.
RFSHCTL1	Refresh control register 1.
RFSHCTL3(2)	Refresh control register 3.
RFSHTMG	Refresh timing register.
ECCCFG0	ECC configuration register 0.

Table 17-36: **DDR Controller Registers *(Cont'd)***

Register Name	Register Description
ECCCFG1[3]	ECC configuration register 1.
ECCSTAT	ECC status register.
ECCCLR	ECC clear register.
ECCERRCNT	ECC error counter register.
ECCCADDR{0:1}	ECC corrected error address register {0:1}.
ECCCSYN{0:2}	ECC corrected syndrome register {0:2}.
ECCBITMASK{0:2}	ECC corrected data bit mask register {0:2}.
ECCUADDR{0:1}	ECC uncorrected error address register {0:1}.
ECCUSYN{0:2}	ECC uncorrected syndrome register {0:2}.
ECCPOISONADDR{0:1}	ECC data poisoning address register{0:1}.
CRCPARCTL{0:2}	CRC parity control register {0:2}.
CRCPARSTAT	CRC parity status register.
INIT{0:7}[1][2][3]	SDRAM initialization registers {0:7}.
DIMMCTL	DIMM control register.
RANKCTL	Rank control register.
DRAMTMG{0:14}[1][2][4]	SDRAM timing registers {0:14}.
ZQCTL{0:2}[2][4]	ZQ control register {0:2}.
ZQSTAT	ZQ status register.
DFITMG{0:1}[1][2][3][4]	DFI timing register {0:1}.
DFILPCFG{0:1}	DFI low-power configuration register {0:1}.
DFIUPD{0:2}[3]	DFI update register {0:2}.
DFIMISC[3]	DFI miscellaneous control register.
DFITMG2[4]	DFI timing register 2.
DBICTL[1]	DM/DBI control register.
ADDRMAP{0:11}	Address map registers {0:11}.
ODTCFG[1][4]	ODT configuration register.
ODTMAP	ODT/rank map register.
SCHED[3]	Scheduler control register.
SCHED1	Scheduler control register 1.
PERFHPR1[3]	High-priority read CAM register 1.
PERFLPR1[3]	Low-priority read CAM register 1.
PERFWR1[3]	Write CAM register 1.
PERFVPR1	Video/isochronous priority read CAM register 1.
PERFVPW1	Video/isochronous priority write CAM register 1.
DQMAP{0:5}	DQ map registers {0:5}.

Table 17-36: **DDR Controller Registers *(Cont'd)***

Register Name	Register Description
DBG{0:1}	Debug register {0:1}.
DBGCAM	CAM debug register.
DBGCMD	Command debug register.
DBGSTAT	Status debug register.
SWCTL	Software register programming control enable.
SWSTA	Software register programming control status.
POISONCFG	AXI poison configuration register.
POISONSTAT	AXI poison status register.
PSTAT	Port status register.
PCCFG	Port common configuration register.
PCFGR_{0:5}	Port {0:5} configuration read register.
PCFGW_{0:5}	Port {0:5} configuration write register.
PCTRL_{0:5}	Port {0:5} control register.
PCFGQOS0_{0:5}(3)	Port {0:5} read QoS configuration register 0.
PCFGQOS1_{0:5}(3)	Port {0:5} read QoS configuration register 1.
PCFGWQOS0_{0:5}(3)	Port {0:5} write QoS configuration register 0.
PCFGWQOS1_{0:5}(3)	Port {0:5} write QoS configuration register 1.
SARBASE0	SAR base address register n.
SARSIZE0	SAR size register n.
SARBASE1	SAR base address register n.
SARSIZE1	SAR size register n.
DERATEINT_SHADOW	Temperature derate interval shadow register.
RFSHCTL0_SHADOW	Refresh control shadow register 0.
RFSHTMG_SHADOW	Refresh timing shadow register.
INIT3_SHADOW	SDRAM initialization shadow register 3.
INIT4_SHADOW	SDRAM initialization shadow register 4.
INIT6_SHADOW	SDRAM initialization shadow register 6.
INIT7_SHADOW	SDRAM initialization shadow register 7.
DRAMTMG{0:14}_SHADOW	SDRAM timing shadow registers {0:14}.
ZQCTL0_SHADOW	ZQ control shadow register 0.
DFITMG0_SHADOW	DFI timing shadow register 0.
DFITMG1_SHADOW	DFI timing shadow register 1.
DFITMG2_SHADOW	DFI timing shadow register 2.

Table 17-36: **DDR Controller Registers *(Cont'd)***

Register Name	Register Description
ODTCFG_SHADOW	ODT configuration shadow register.

Notes:

1. Quasi dynamic registers group 1.
2. Quasi dynamic registers group 2.
3. Quasi dynamic registers group 3.
4. Quasi dynamic registers group 4.
5. For detailed description, see the *Zynq UltraScale+ MPSoC Register Reference* (UG1087) [Ref 4].

DDRPHY Registers

Table 17-37: **DDRPHY Registers**

Register Type	Register Name	Register Description
Configuration register	PGCR{0:7}	PHY general configuration registers {0:7}.
	DXCCR	DATX8 common configuration register.
	DSGCR	DDR system general configuration register.
	ODTCR	ODT configuration register.
	DCR	DRAM configuration register.
	RDIMMGCR0	RDIMM general configuration register 0.
	RDIMMGCR1	RDIMM general configuration register 1.
	RDIMMGCR2	RDIMM general configuration register 2.
	DTCR0	Data training configuration register 0.
	DTCR1	Data training configuration register 1.
	DCUGCR	DCU general configuration register.
	RIOCR{0:5}	Rank I/O configuration registers {0:5}.
	ACIOCR{0:5}	AC I/O configuration registers {0:5}.
	DX{0:8}GCR0	DATX8 {0:8} general configuration registers {0:8}.
	DX{0:8}GCR1	DATX8 {0:8} general configuration register 1.
	DX{0:8}GCR2	DATX8 {0:8} general configuration register 2.
	DX{0:8}GCR3	DATX8 {0:8} general configuration register 3.
	DX{0:8}GCR4	DATX8 {0:8} general configuration register 4.
	DX{0:8}GCR5	DATX8 {0:8} general configuration register 5.
	DX{0:8}GCR6	DATX8 {0:8} general configuration register 6.
	DX8SL0IOCR	DATX8 0-1 I/O configuration register.
	DX8SL1IOCR	DATX8 2-3 I/O configuration register.
	DX8SL2IOCR	DATX8 4-5 I/O configuration register.
	DX8SL3IOCR	DATX8 6-7 I/O configuration register.
	DX8SL4IOCR	DATX8 0-1 I/O configuration register.
	DX8SLbIOCR	DATX8 0-8 I/O configuration register.

Table 17-37: **DDRPHY Registers *(Cont'd)***

Register Type	Register Name	Register Description
Status Register	PGSR0	PHY general status register 0.
	PGSR1	PHY general status register 1.
	PGSR02	PHY general status register 2.
	DCUSR00	DCU status register 0.
	DCUSR01	DCU status register 1.
	ZQ0SR	ZQ n impedance control status register.
	ZQ1SR	ZQ n impedance control status register.
	DX{0:8}RSR1	DATX8 {0:8} rank status register 1.
	DX{0:8}RSR2	DATX8 {0:8} rank status register 2.
	DX{0:8}RSR3	DATX8 {0:8} rank status register 3.
	DX{0:8}GSR0	DATX8 {0:8} general status register 0.
	DX{0:8}GSR1	DATX8 {0:8} general status register 1.
	DX{0:8}GSR2	DATX8 {0:8} general status register 2.
	DX{0:8}GSR3	DATX8 {0:8} general status register 3.
Line Register	ACBDLR{0:9}	AC bit delay line registers {0:9}.
	ACBDLR15	AC bit delay line register 15.
	ACBDLR16	AC bit delay line register 16.
	ACLCDLR	AC local calibrated delay line register.
	ACMDLR0	AC master delay line register 0.
	ACMDLR1	AC master delay line register 1.
	DX{0:8}BDLR0	DATX8 {0:8} bit delay line register 0.
	DX{0:8}BDLR1	DATX8 {0:8} bit delay line register 1.
	DX{0:8}BDLR2	DATX8 {0:8} bit delay line register 2.
	DX{0:8}BDLR3	DATX8 {0:8} bit delay line register 3.
	DX{0:8}BDLR4	DATX8 {0:8} bit delay line register 4.
	DX{0:8}BDLR5	DATX8 {0:8} bit delay line register 5.
	DX{0:8}BDLR6	DATX8 {0:8} bit delay line register 6.
	DX{0:8}LCDLR0	DATX8 {0:8} local calibrated delay line register 0.
	DX{0:8}LCDLR1	DATX8 {0:8} local calibrated delay line register 1.
	DX{0:8}LCDLR2	DATX8 {0:8} local calibrated delay line register 2.
	DX{0:8}LCDLR3	DATX8 {0:8} local calibrated delay line register 3.
	DX{0:8}LCDLR4	DATX8 {0:8} local calibrated delay line register 4.
	DX{0:8}LCDLR5	DATX8 {0:8} local calibrated delay line register 5.
	DX{0:8}MDLR0	DATX8 {0:8} master delay line register 0.
	DX{0:8}MDLR1	DATX8 {0:8} master delay line register 1.

Table 17-37: **DDRPHY Registers *(Cont'd)***

Register Type	Register Name	Register Description
Control Register	PLLCR{0:5}	Address/control PLL controls {0:5}.
	AACR	Anti-aging control register.
	RDIMMCR0	RDIMM control register 0.
	RDIMMCR1	RDIMM control register 1.
	RDIMMCR2	RDIMM control register 2.
	RDIMMCR3	RDIMM control register 3.
	RDIMMCR4	RDIMM control register 4.
	IOVCR0	I/O VREF control register 0.
	IOVCR1	I/O VREF control register 1
	VTCR0	VREF training control register 0.
	VTCR1	VREF training control register 1.
	ZQCR	ZQ impedance control register.
	DX8SL0OSC	DATX8 0-1 oscillator, delay-line test, PHY FIFO and high-speed reset, loopback, and gated clock control register.
	DX8SL0PLLCR0	DAXT8 0-1 PLL control register 0.
	DX8SL0PLLCR1	DAXT8 0-1 PLL control register 1 (Type B PLL only).
	DX8SL0PLLCR2	DAXT8 0-1 PLL control register 2 (Type B PLL only).
	DX8SL0PLLCR3	DAXT8 0-1 PLL control register 3 (Type B PLL only).
	DX8SL0PLLCR4	DAXT8 0-1 PLL control register 4 (Type B PLL only).
	DX8SL0PLLCR5	DAXT8 0-1 PLL control register 5 (Type B PLL only).
	DX8SL0DQSCTL	DATX8 0-1 DQS control register.
	DX8SL0TRNCTL	DATX8 0-1 training control register.
	DX8SL0DDLCTL	DATX8 0-1 DDL control register.
	DX8SL0DXCTL1	DATX8 0-1 DX control register 1.
	DX8SL0DXCTL2	DATX8 0-1 DX control register 2.

Table 17-37: **DDRPHY Registers *(Cont'd)***

Register Type	Register Name	Register Description
Control Register (*Cont'd*)	DX8SL1OSC	DATX8 0-1 oscillator, delay-line test, PHY FIFO and high-speed reset, loopback, and gated clock control register.
	DX8SL1PLLCR0	DAXT8 0-1 PLL control register 0.
	DX8SL1PLLCR1	DAXT8 0-1 PLL control register 1 (Type B PLL only).
	DX8SL1PLLCR2	DAXT8 0-1 PLL control register 2 (Type B PLL only).
	DX8SL1PLLCR3	DAXT8 0-1 PLL control register 3 (Type B PLL only).
	DX8SL1PLLCR4	DAXT8 0-1 PLL control register 4 (Type B PLL only).
	DX8SL1PLLCR5	DAXT8 0-1 PLL control register 5 (Type B PLL only).
	DX8SL1DQSCTL	DATX8 0-1 DQS control register.
	DX8SL1TRNCTL	DATX8 0-1 training control register.
	DX8SL1DDLCTL	DATX8 0-1 DDL control register.
	DX8SL1DXCTL1	DATX8 0-1 DX control register 1.
	DX8SL1DXCTL2	DATX8 0-1 DX control register 2.
	DX8SL2OSC	DATX8 0-1 oscillator, delay-line test, PHY FIFO and high-speed reset, loopback, and gated clock control register.
	DX8SL2PLLCR0	DAXT8 0-1 PLL control register 0.
	DX8SL2PLLCR1	DAXT8 0-1 PLL control register 1 (Type B PLL only).
	DX8SL2PLLCR2	DAXT8 0-1 PLL control register 2 (Type B PLL only).
	DX8SL2PLLCR3	DAXT8 0-1 PLL control register 3 (Type B PLL only).
	DX8SL2PLLCR4	DAXT8 0-1 PLL control register 4 (Type B PLL only).
	DX8SL2PLLCR5	DAXT8 0-1 PLL control register 5 (Type B PLL only).
	DX8SL2DQSCTL	DATX8 0-1 DQS control register.
	DX8SL2TRNCTL	DATX8 0-1 training control register.
	DX8SL2DDLCTL	DATX8 0-1 DDL control register.
	DX8SL2DXCTL1	DATX8 0-1 DX control register 1
	DX8SL2DXCTL2	DATX8 0-1 DX control register 2
	DX8SL3OSC	DATX8 0-1 oscillator, delay-line test, PHY FIFO and high-speed reset, loopback, and gated clock control register.
	DX8SL3PLLCR0	DAXT8 0-1 PLL control register 0.
	DX8SL3PLLCR1	DAXT8 0-1 PLL control register 1 (Type B PLL only).
	DX8SL3PLLCR2	DAXT8 0-1 PLL control register 2 (Type B PLL only)
	DX8SL3PLLCR3	DAXT8 0-1 PLL control register 3 (Type B PLL only)
	DX8SL3PLLCR4	DAXT8 0-1 PLL control register 4 (Type B PLL only)
	DX8SL3PLLCR5	DAXT8 0-1 PLL control register 5 (Type B PLL only)
	DX8SL3DQSCTL	DATX8 0-1 DQS control register.
	DX8SL3TRNCTL	DATX8 0-1 training control register.

Table 17-37: **DDRPHY Registers *(Cont'd)***

Register Type	Register Name	Register Description
Control Register (*Cont'd*)	DX8SL3DDLCTL	DATX8 0-1 DDL control register.
	DX8SL3DXCTL1	DATX8 0-1 DX control register 1.
	DX8SL3DXCTL2	DATX8 0-1 DX control register 2.
	DX8SL4OSC	DATX8 0-1 oscillator, delay-line test, PHY FIFO and high-speed reset, loopback, and gated clock control register.
	DX8SL4PLLCR0	DAXT8 0-1 PLL control register 0.
	DX8SL4PLLCR1	DAXT8 0-1 PLL control register 1 (Type B PLL only).
	DX8SL4PLLCR2	DAXT8 0-1 PLL control register 2 (Type B PLL only).
	DX8SL4PLLCR3	DAXT8 0-1 PLL control register 3 (Type B PLL only).
	DX8SL4PLLCR4	DAXT8 0-1 PLL control register 4 (Type B PLL only).
	DX8SL4PLLCR5	DAXT8 0-1 PLL control register 5 (Type B PLL only).
	DX8SL4DQSCTL	DATX8 0-1 DQS control register.
	DX8SL4TRNCTL	DATX8 0-1 training control register.
	DX8SL4DDLCTL	DATX8 0-1 DDL control register.
	DX8SL4DXCTL1	DATX8 0-1 DX control register 1.
	DX8SL4DXCTL2	DATX8 0-1 DX control register 2.
	DX8SLbOSC	DATX8 0-1 oscillator, delay-line test, PHY FIFO and high-speed reset, loopback, and gated clock control register.
	DX8SLbPLLCR0	DAXT8 0-1 PLL control register 0.
	DX8SLbPLLCR1	DAXT8 0-1 PLL control register 1 (Type B PLL only).
	DX8SLbPLLCR2	DAXT8 0-1 PLL control register 2 (Type B PLL only).
	DX8SLbPLLCR3	DAXT8 0-1 PLL control register 3 (Type B PLL only).
	DX8SLbPLLCR4	DAXT8 0-1 PLL control register 4 (Type B PLL only).
	DX8SLbPLLCR5	DAXT8 0-1 PLL control register 5 (Type B PLL only).
	DX8SLbDQSCTL	DATX8 0-1 DQS control register.
	DX8SLbTRNCTL	DATX8 0-1 training control register.
	DX8SLbDDLCTL	DATX8 0-1 DDL control register.
	DX8SLbDXCTL1	DATX8 0-1 DX control register 1.
	DX8SLbDXCTL2	DATX8 0-1 DX control register 2.
Identification Register	RIDR	Revision identification register.
Initialization Register	PIR	PHY initialization register.

Table 17-37: **DDRPHY Registers *(Cont'd)***

Register Type	Register Name	Register Description
Timing Register	PTR0	PHY timing register 0.
	PTR1	PHY timing register 1.
	PTR2	PHY timing register 2.
	PTR3	PHY timing register 3.
	PTR4	PHY timing register 4.
	PTR5	PHY timing register 5.
	PTR6	PHY timing register 6.
	DX0GTR0	DATX8 n general timing register 0.
	DX1GTR0	DATX8 n general timing register 0.
	DX2GTR0	DATX8 n general timing register 0.
	DX3GTR0	DATX8 n general timing register 0.
	DX4GTR0	DATX8 n general timing register 0.
	DX5GTR0	DATX8 n general timing register 0.
	DX6GTR0	DATX8 n general timing register 0.
	DX7GTR0	DATX8 n general timing register 0.
	DX8GTR0	DATX8 n general timing register 0.
Parameters Register	DTPR0	DRAM timing parameters register 0.
	DTPR1	DRAM timing parameters register 1.
	DTPR2	DRAM timing parameters register 2.
	DTPR3	DRAM timing parameters register 3.
	DTPR4	DRAM timing parameters register 4.
	DTPR5	DRAM timing parameters register 5.
	DTPR6	DRAM timing parameters register 6.
	DCUTPR	DCU timing parameters register.
Purpose Register	GPR0	General purpose register 0.
	GPR1	General purpose register 1.
Command Register	SCHCR0	Scheduler command register 0.
	SCHCR1	Scheduler command register 1.

Table 17-37: **DDRPHY Registers *(Cont'd)***

Register Type	Register Name	Register Description
Mode Register	MR0	LPDDR4 mode register 0.
	MR1	LPDDR4 mode register 1.
	MR2	LPDDR4 mode register 2.
	MR3	LPDDR4 mode register 3.
	MR4	DDR4 mode register 4.
	MR5	DDR4 mode register 5.
	MR6	DDR4 mode register 6.
	MR7	DDR4 mode register 7.
	MR11	LPDDR4 mode register 11.
	MR12	LPDDR4 mode register 12.
	MR13	LPDDR4 mode register 13.
	MR14	LPDDR4 mode register 14.
	MR22	LPDDR4 mode register 22.
Address Register	DTAR0	Data training address register 0.
	DTAR1	Data training address register 1.
	DTAR2	Data training address register 2.
	DCUAR	DCU address register.
	BISTAR0	BIST address register 0.
	BISTAR1	BIST address register 1.
	BISTAR2	BIST address register 2.
	BISTAR3	BIST address register 3.
	BISTAR4	BIST address register 4.
Data Register	DTDR0	Data training data register 0.
	DTDR1	Data training data register 1.
	DTEDR0	Data training eye data register 0.
	DTEDR1	Data training eye data register 1.
	DTEDR2	Data training eye data register 2.
	VTDR	VREF training data register.
	DCUDR	DCU data register.
	ZQ0DR0	ZQ n impedance control data register 0.
	ZQ0DR1	ZQ n impedance control data register 1.
	ZQ0OR0	ZQ n impedance control override data register 0.
	ZQ0OR1	ZQ n impedance control override data register 1.

Table 17-37: **DDRPHY Registers *(Cont'd)***

Register Type	Register Name	Register Description
Data Register (*Cont'd*)	ZQ1DR0	ZQ n impedance control data register 0.
	ZQ1DR1	ZQ n impedance control data register 1.
	ZQ1OR0	ZQ n impedance control override data register 0.
	ZQ1OR1	ZQ n impedance control override data register 1.
Training register	CATR0	CA training register 0.
	CATR1	CA training register 1.
Drift register	DQSDR0	DQS drift register 0.
	DQSDR1	DQS drift register 1.
	DQSDR2	DQS drift register 2.
Run register	DCURR	DCU run register.
Loop register	DCULR	DCU loop register.
ID Register	RANKIDR	Rank ID register.
Program Register	ZQ0PR0	ZQ n impedance control program register 0.
	ZQ0PR1	ZQ n impedance control program register 1.
	ZQ1PR0	ZQ n impedance control program register 0.
	ZQ1PR1	ZQ n impedance control program register 1.

Programming Model

This section contains the programming models for various operations.

Programming Modes

This section outlines the circumstances under which the DDRC registers can be written. Most registers are initialized when the DDRC core is in reset (core_ddrc_rstn = 0) and should not need to be changed afterwards. The exceptions are listed in the following sections. The core_ddrc_core_clk should be brought up and running before the DDRC core is brought out of reset (core_ddrc_rstn is deasserted).

The DDRC register programming modes are described in the *Zynq UltraScale+ MPSoC Register Reference* UG1087 [Ref 4]. In UG1087, registers are described as static, dynamic, dynamic - refresh related, or quasi dynamic.

Dynamic Registers

The dynamic registers can be written at any time during the operation of the DDRC.

Dynamic - Refresh Related Registers

The refresh related registers are dynamic, however, to update them perform the following:

- Change the refresh associated register as desired.
- After the changed register is stable, toggle the RFSHCTL3.refresh_update_level signal.

The SDRAM controller recognizes the refresh_update_level signal change and updates all refresh-related register values accordingly. This mechanism is needed to avoid sampling errors in the target clock domain, as well as to allow the controller to provide special handling (such as issuing an additional refresh and resetting the refresh timer if needed) when a refresh-related timing register has changed.

The refresh related registers are dynamic, except RFSHCTL3.refresh_mode, which can only be programmed during the initialization or when the controller is in self-refresh mode. At initialization, the RFSHCTL3.refresh_mode must be set to match the refresh mode field of MR3, written to SDRAM via INIT4.emr3. When updating this register in self-refresh mode, the corresponding MR3 command is sent automatically after SRX. In this case, the INIT4.emr3 should be modified as well because the value written to the SDRAM via the MR3 command is taken from INIT4.emr3, and not from RFSHCTL3.refresh_mode.

Quasi Dynamic Registers

In addition to the dynamic registers, the following categories of registers can be written after reset:

- Group 1: registers that can be written when no Read/Write traffic is present at the DFI.
- Group 2: registers that can be written in self-refresh, DPD, and MPSM modes.
- Group 3: registers that can be written when the controller is empty.
- Group 4: registers that can be written depending on MSTR.frequency_mode and the MSTR2.target_frequency.

Each category requires specific conditions for the registers to be programmed. Once the programming conditions are met, the SWCTL.sw_done register must be programmed to `1’b0` to enable the software programming.

Once the programming is completed, the SWCTL.sw_done must be set to `1’b1` and the SWSTAT.sw_done_ack must be read as `1’b1` to ensure that the quasi dynamic registers are propagated correctly to the destination clocks.

Traffic must be enabled again depending on the register category as described in the following sections.

Group 1: Registers that can be written when no read/write traffic is present at the DFI

By setting the DBG1.dis_dq register and polling DBGCAM.wr_data_pipeline_empty and DBGCAM.rd_data_pipeline_empty1, it is possible to prevent any read or write traffic from being sent on the DFI. Also, if DDR4 retry is enabled by CRCPARCTRL1.crc_parity_retry_enable, poll CRCPARSTAT.cmd_in_err_window until it is equal to 0. If software intervention is enabled by CRCPARCTL1.alert_wait_for_sw, also monitor CRCPARSTAT.dfi_alert_err_int and CRCPARSTAT.dfi_alert_err_fatl_int during the polling. If one or more of them are asserted before polling is done, retry procedure must be completed prior to the subsequent steps. In this mode, it is safe to write to the group 1 registers.

Re-enable the traffic by writing DBG1.dis_dq to `1'b0`. To make sure the correct value is propagated, registers DBGCAM.wr_data_pipeline_empty and DBGCAM.rd_data_pipe-line_empty must be polled at least twice after DBG1.dis_dq is set to 1.

Group 2: Registers that can be written in self-refresh, DPD, and MPSM modes

When the DDRC has entered self-refresh mode via software (PWRCTL.selfref_sw), the DFI bus is idle until software exits self-refresh. The same is true in deep power-down (DPD) for LPDDR3, and maximum power saving mode (MPSM) for DDR4.

Note: For self-refresh, ensure that self-refresh is not caused by "Automatic Self-refresh only" by checking the STAT.operating_mode = `3'b011` and STAT.selfref_type = `2'b10`. If DDR4 retry is enabled by CRCPARCTRL1.crc_parity_retry_enable and software intervention is enabled by CRCPARCTL1.alert_wait_for_sw, also monitor CRCPARSTAT.dfi_alert_err_int and CRCPARSTAT.dfi_alert_err_fatl_int during the polling STAT.selfref_type. If one or more of them are asserted before polling is done, retry procedure must be completed prior to the subsequent steps.

In this section, references to self-refresh mean self-refresh (non-LPDDR4), or SR-Powerdown (LPDDR4).

- For MPSM, ensure STAT.operating_mode = `3'b110` is the case before changing any of the registers listed below (see explanation below). If DDR4 retry is enabled by CRCPARCTRL1.crc_parity_retry_enable, poll CRCPARSTAT.cmd_in_err_window until it equals 0. If software intervention is enabled by CRCPARCTL1.alert_wait_for_sw, also monitor CRCPARSTAT.dfi_alert_err_int and CRCPARSTAT.dfi_alert_err_fatl_int during the polling. If one or more of them are asserted before polling, retry procedure must be completed prior to the subsequent steps.
- For DPD, ensure STAT.operating_mode = `3'b110` is the case before changing any of the registers listed below (see the explanation below).

- STAT.operating_mode = `3'b1xx` for DPD/MPSM, but entry/exit and in mode itself can be differentiated as follows:
  - operating_mode = `3'b101` — DPD/MPSM entry is occurring.
  - operating_mode = `3'b110` — DPD/MPSM mode is reached.
  - operating_mode = `3'b111` — DPD/MPSM exit is occurring.

In this mode, it is safe to write the group 2 registers.

Re-enable the traffic by writing PWRCTL.selfref_sw to `1'b0`.

Group 3: Registers that can be written when controller is empty

For multi-port configurations, PCTRL_n.port_en is used to enable or disable the input traffic per port.

The controller idleness can be polled first from PSTAT register (wr_port_busy_n and rd_port_busy_n bit fields) and should read as PSTAT==`32'b0` (not busy).

The DDRC CAM/pipeline empty status must be polled ((DBGCAM.dbg_wr_q_empty== `1'b1`) && (DBGCAM.dbg_rd_q_empty== `1'b1`) && (DBGCAM.wr_data_pipeline_empty== `1'b11`) && (DBGCAM.rd_data_pipeline_empty== `1'b1`)). Also, if the DDR4 retry is enabled by the CRCPARCTRL1.crc_parity_retry_enable, poll CRCPARSTAT.cmd_in_err_window until it is equal to 0.

If software intervention is enabled by CRCPARCTL1.alert_wait_for_sw, monitor CRCPARSTAT.dfi_alert_err_int and CRCPARSTAT.dfi_alert_err_fatl_int during the polling. If one or more of them are asserted before polling is finished, retry, because the procedure must be completed prior to the subsequent steps. In this mode, it is safe to write the group 3 registers. Enable the traffic by writing `1'b1` to PCTRL_n.port_en.

Group 4: Registers that can be written depending on MSTR.frequency_mode

When MSTR.frequency_mode = 0, it is safe to write to group 4 registers in the *_SHADOW registers. When MSTR.frequency_mode = 1, it is safe to write to group 4 registers in the non-*_SHADOW registers.

Power Saving Features

The DDR memory controller supports various methods to save power within the system in different modes: Precharge power-down, self-refresh, deep power-down, maximum power saving, and disabling clock to the SDRAM using PWRCTL.en_dfi_dram_clk_disable.

In multi-rank systems, these power-saving modes cannot be applied on a per-rank basis. If applied, they are always applied globally. When enabled, the controller automatically enters and exits precharge power-down mode based on a programmable idle timeout period. Self-refresh can be entered/exited using the following approaches.

- Based on a programmable idle timeout period (similar to precharge power-down idle timeout).
- Software controlled.
- Hardware low-power interface(s).

Deep power-down (DPD) and maximum power saving mode (MPSM) entry and exit are explicitly controlled by you. In addition, the clock to the SDRAM can be disabled by setting the PWRCTL.en_dfi_dram_clk_disable bit.

This can be done in the following modes:

- Self-refresh.
- Self-refresh power down (LPDDR4 only).
- Power-down.
- Deep power-down.
- Maximum power saving mode.

IMPORTANT: *Do not enable more than one of the following power-saving modes simultaneously.*

- Deep power-down.
- Maximum power saving mode.

You can enable any combination of power-down and self-refresh modes simultaneously.

- Power-down: PWRCTL[powerdown_en]=1.
- Automatic self-refresh: PWRCTL[selfref_en]=1.
- Software self-refresh: PWRCTL[selfref_sw=1.

Enabling the assertion of the PWRCTL[en_dfi_dram_clk_disable] bit is valid in combination with any of the power-saving modes.

Automatic Low Power Modes

The automatic low power modes include power-down, clock-stop, and self-refresh. The modes supported for each memory type are listed in Table 17-38. The controller can automatically switch in and out of these modes based on the memory traffic.

Table 17-38: **Low Power Feature Support**

Memory Type	Low-power Features
DDR 3 and 4 UDIMM	Power-down, clock-stop, self-refresh.
DDR 3 and 4 RDIMM	Power-down, self-refresh.
LPDDR 3 and 4	No automatic low-power.

Precharge Power Down

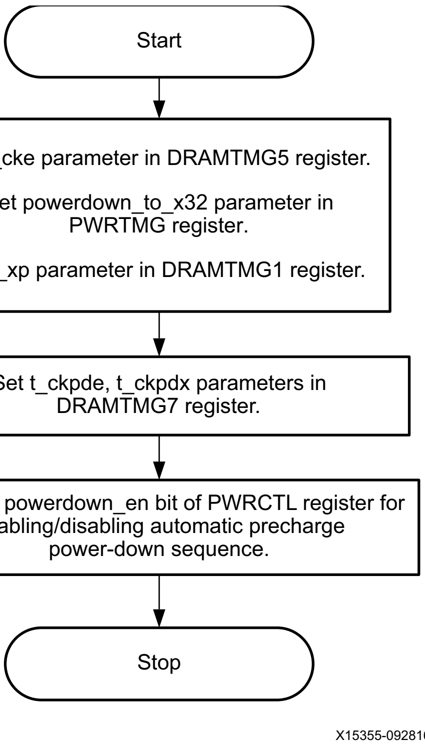

Start
Set t_cke parameter in DRAMTMG5 register.
Set powerdown_to_x32 parameter in PWRTMG register.
Set t_xp parameter in DRAMTMG1 register.
Set t_ckpde, t_ckpdx parameters in DRAMTMG7 register.
Set/reset powerdown_en bit of PWRCTL register for enabling/disabling automatic precharge power-down sequence.
Stop
X15355-092816

Figure 17-11: **Precharge Power Down Flowchart**

Deep Power-Down

Note: This power saving mode is applicable for LPDDR3 devices only.

Entering Deep Power-down

By setting the PWRCTL.deeppowerdown_en bit, the SDRAM device can be put into deep power-down mode if all of these conditions are true:

- The period specified by PWRTMG.powerdown_to_x32 has passed while the DDRC is idle (except for issuing refreshes).
- PWRCTL.selfref_sw = 0.
- PWRCTL.selfref_en = 0.
- If HWLPCTL.hw_lp_en = 1, DPD is entered only when the hardware low power interface has completed a self-refresh exit. (This can be checked by observing STAT.operating_mode and STAT.selfref_type).
- If HWLPCTL.hw_lp_exit_idle_en = 1, DPD is entered only when all bits of cactive_in_ddrc = 0.

Entering deep power-down includes these steps:

1. If there is a self-refresh exit previously, wait for at least one refresh command (or eight per-bank refresh commands if LPDDR3 per-bank refresh is enabled) to all active ranks. Auto-refresh logic must be enabled, or refresh should be issued using direct software requests of refresh command via DBGCMD.rank*_refresh.
2. Precharging (closing) all open pages. Pages are closed one at a time in no specified order.
3. Waiting for tRP (row precharge) idle period.
4. Issuing the command to enter deep power-down. For multi-rank systems, all chip-selects are asserted so that all ranks enter deep power-down simultaneously. The deep power-down entry commands are CKE=0, CSN=0, CA0=1, CA1=1, and CA2=0.
5. This step occurs only if the DFI low power interface for deep power-down is enabled (DFILPCFG0.dfi_lp_en_dpd). It attempts an entry to low power mode via DFI low power interface with dfi_lp_wakeup set by DFILPCFG0.dfi_lp_wakeup_dpd. The low power entry attempt is delayed with DFITMG0.dfi_t_ctrl_delay + DRAMTMG6.t_ckdpde clock cycles, this is needed to satisfy SDRAM timings related to disabling clocks when the PHY is programmed to gate the clock, to save maximum power.

If the DDRC receives a read or write request from the SoC core during step 1 or step 3, the deep power-down entry is immediately aborted. The same is true if PWRCTL.deep_powerdown_en is driven to 0 during step 1 or step 3. Once the deep power-down entry command is issued, proper deep power-down exit is required, as described in the following section.

Note: Contents of SDRAM might be lost upon entry into deep power-down mode.

Exiting Deep Power-down

Once the DDRC puts the DDR SDRAM device in deep power-down mode, the DDRC automatically exits deep power-down and repeats the initialization sequence when PWRCTL.deeppowerdown_en is reset to 0. An exit from DFI low power mode is performed prior to exiting deep power-down (this occurs only if DFI low power mode entry during deep power-down entry is successful). DFI low power mode is exited after the wakeup time specified by DFILPCFG0.dfi_lp_wakeup_dpd, but not earlier than DFITMG1.dfi_t_dram_clk_enable + DRAMTMG6.t_ckdpdx clock cycles.

Exiting deep power-down involves these steps when SDRAM initialization is performed by the PHY (INIT0.skip_dram_init = `2'b01` or `2'b11`):

1. To prevent the uMCTL2 asserting dfi_cke before the SDRAM initialization is complete, it is necessary to set INIT0.skip_dram_init = `2'b11` before clearing PWRCTL.deeppowerdown_en
2. If step 1 is performed, to ensure that controller updates do not occur when INIT0.skip_dram_init is changed back to `2'b01` (which could make DFI bus active when dfi_ctrlupd_req), it is necessary to set DFIUPD0.dis_auto_ctrlupd and DBG1.dis_hif and to stop sending software controller updates before clearing PWRCTL.deeppowerdown_en.
3. Clear DFIMISC.dfi_init_complete_en = 0 register, before clearing PWRCTL.deeppowerdown_en to ensure that the DDRC waits until the PHY completes its initialization.
4. Reset PWRCTL.deeppowerdown_en to 0 and poll STAT.operating mode to detect when the DDRC exits from DPD and then start the SDRAM initialization by setting the PUB_PIR register.
5. Once PHY Init is started and PIR is programmed, set back the old value of skip_dram_init, if it was updated as described in step 1.
6. Poll the relevant PUB's PGSR register to detect when the PUB Initialization is complete.
7. Change back the DFIUPD0.dis_auto_ctrlupd and DBG1.dis_hif values and/or restart sending software controller updates, if they were disabled as described in step 2.
8. Set DFIMISC.dfi_init_complete_en = 1 to allow the DDRC's state machine to exit the initialization state.

Self Refresh

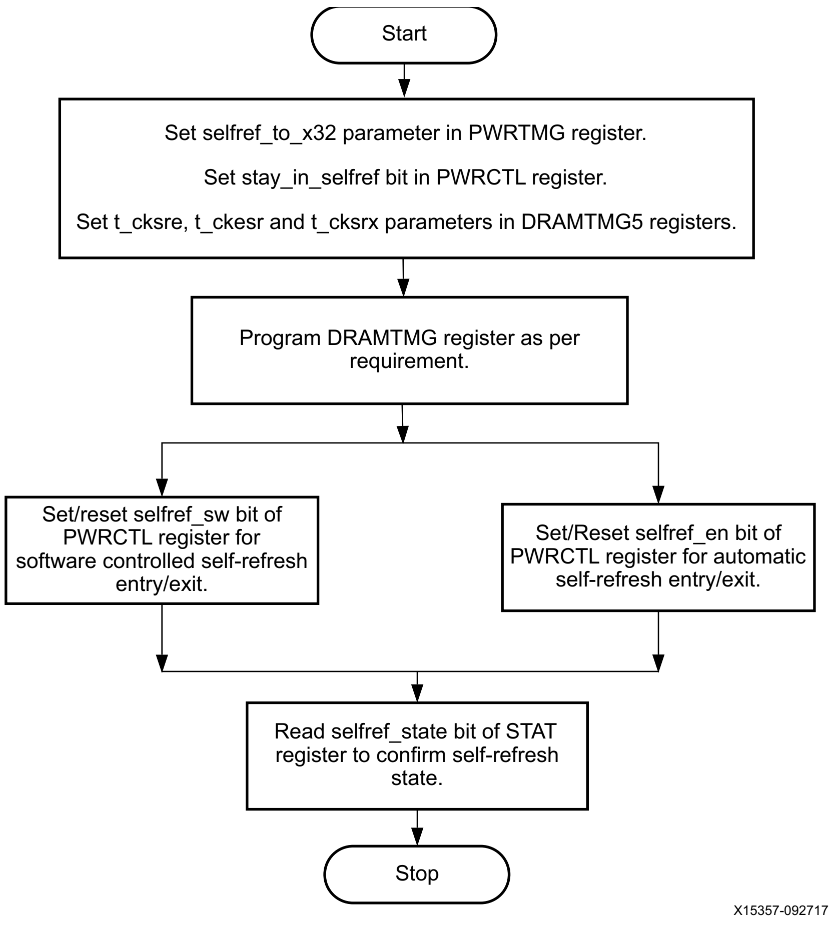

Start
Set selfref_to_x32 parameter in PWRTMG register.
Set stay_in_selfref bit in PWRCTL register.
Set t_cksre, t_ckesr and t_cksrx parameters in DRAMTMG5 registers.
Program DRAMTMG register as per requirement.
Set/reset selfref_sw bit of PWRCTL register for software controlled self-refresh entry/exit.
Set/Reset selfref_en bit of PWRCTL register for automatic self-refresh entry/exit.
Read selfref_state bit of STAT register to confirm self-refresh state.
Stop
X15357-092717

Figure 17-12: **Self Refresh Flowchart**

Maximum Power Saving

Note: Maximum power saving mode is applicable for DDR4 devices only.

Entering Maximum Power Saving Mode

By setting the PWRCTL.mpsm_en bit, you can put the DDR4 devices into maximum power saving mode, if all of these conditions are true:

- The DDRC is idle (except for issuing refreshes).
- PWRCTL.selfref_sw = 0.
- PWRCTL.selfref_en = 0.
- If HWLPCTL.hw_lp_en = 1, MPSM is entered only when the hardware low power interface has completed a self-refresh exit. (This can be checked by observing STAT.operating_mode and STAT.selfref_type).

- If HWLPCTL.hw_lp_exit_idle_en=1, MPSM is entered only when all bits of cactive_in_ddrc = 0.

If CA parity is enabled, the DDRC disables CA parity before entering maximum power saving mode, and enables the CA parity after exiting maximum power saving mode. Note that the DDRC uses the setting of INIT6.mr5[2:0] to determine whether to perform this disabling/enabling of CA parity, and uses the entire INIT6.mr5[15:0] for the automatic MRS commands. Consequently, if any part of the SDRAM's MR5 is updated by software, it is also the responsibility of the software to update INIT6.mr5 so that it is aligned to the SDRAM's MR5, if it is intended to enter MPSM.

If CAL mode is enabled, the DDRC disables CAL mode before entering maximum power saving mode, and enables CAL mode after exiting maximum power saving mode.

If geardown is enabled, the user must disable geardown by using self-refresh, before setting the PWRCTL.mpsm_en, as follows:

1. Put SDRAM in self-refresh mode by setting PWRCTL.selfref_sw to 1 and polling STAT.operating_mode.
2. Disable geardown mode by setting MSTR.geardown_mode to 0.
3. Wake SDRAM up from self-refresh by setting PWRCTL.selfref_sw to 0 and polling STAT.operating_mode (geardown is disabled).

The DDRC does not disable or enable geardown before entering or after exiting MPSM.

Entering maximum power saving mode includes the following steps:

1. If there is a self-refresh exit previously, wait for at least one refresh command to all active ranks. Auto-refresh logic must be enabled, or refresh should be issued using direct software requests of refresh command via DBGCMD.rank*_refresh.
2. Precharging (closing) all open pages. Pages are closed one-at-a-time (not in a specified order).
3. Waiting for tRP (row precharge) idle period.
4. Issuing the MRS command to enter maximum power saving mode. For multi-rank systems, MRS commands should be sent to all ranks. This occurs either simultaneously, to even and odd ranks separately, or to each rank separately, depending on the value of registers DIMMCTL.dimm_output_inv_en, DIMMCTL.dimm_addr_mirr_en, and DIMMCTL.dimm_stagger_cs_en.
5. This step occurs only if DFI low power interface for maximum power saving mode is enabled (DFILPCFG1.dfi_lp_en_mpsm). It attempts an entry to low power mode via DFI low power interface with dfi_lp_wakeup set by DFILPCFG1.dfi_lp_wakeup_mpsm. The low power entry attempt is delayed with DFITMG0.dfi_t_ctrl_delay + DRAMTMG11.t_ckmpe clock cycles, this is needed to satisfy SDRAM timings related to disabling clocks when the PHY is programmed to gate the clock, to save maximum power.

If the DDRC receives a read or write request from the SoC core during step 1 or step 2, the maximum power saving mode entry is immediately aborted. The same is true if PWRCTL.mpsm_en is driven to 0 during step 1 or step 2. Once the maximum power saving mode entry command is issued, proper maximum power saving mode exit is required as described in the next section.

Exiting Maximum Power Saving Mode

Once the DDRC puts the DDR SDRAM device in maximum power saving mode, the DDRC automatically exits maximum power saving mode when PWRCTL.mpsm_en is reset to 0. An exit from DFI low power mode is performed prior to exiting the maximum power saving mode (occurs only if DFI low power mode entry during maximum power saving mode is successful). DFI low power mode is exited after the wakeup time specified by DFILPCFG0.dfi_lp_wakeup_mpsm, but not earlier than DFITMG1.dfi_t_dram_clk_enable + DRAMTMG5.t_cksrx clock cycles (tCKMPX value is the same as tCKSRX).

After exiting maximum power saving mode, geardown should be enabled back by using self-refresh, if it was disabled before MPSM entry.

1. After setting PWRCTL_.mpsm_en to 0, put SDRAM in self-refresh mode by setting PWRCTL.selfref_sw to 1 and polling STAT.operating_mode.
2. Enable back geardown mode by setting MSTR.geardown_mode back to 1.
3. Wake SDRAM up from self-refresh by setting PWRCTL.selfref_sw to 0 and polling STAT.operating_mode (geardown is enabled immediately after self-refresh exit)

Asserting PWRCTL.en_dfi_dram_clk_disable to Disable the Clocks to DRAM

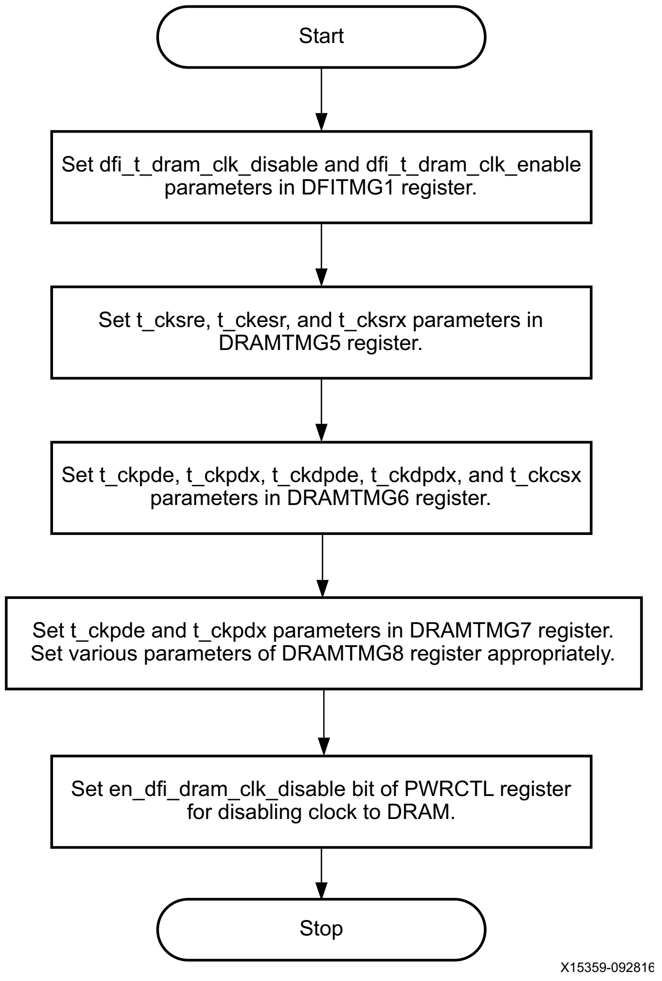

Start
Set dfi_t_dram_clk_disable and dfi_t_dram_clk_enable parameters in DFITMG1 register.
Set t_cksre, t_ckesr, and t_cksrx parameters in DRAMTMG5 register.
Set t_ckpde, t_ckpdx, t_ckdpde, t_ckdpdx, and t_ckcsx parameters in DRAMTMG6 register.
Set t_ckpde and t_ckpdx parameters in DRAMTMG7 register. Set various parameters of DRAMTMG8 register appropriately.
Set en_dfi_dram_clk_disable bit of PWRCTL register for disabling clock to DRAM.
Stop
X15359-092816

Figure 17-13: **Asserting the PWRCTL.en_dfi_dram_clk_disable bit to Disable the Clocks to DRAM Flowchart**

DDR Initialization

PHY Initialization

After deasserting the reset, the PHY is uninitialized. PHY initialization is comprised of initializing the PHY PLL(s), running the initial impedance calibration, and running delay line calibration. These functions can all be triggered at the same time by writing PIR = `x0004_0073`. The initial impedance calibration can be run in parallel with the PLL initialization and subsequent delay line calibration.

DRAM Initialization

The DDR PHY has an embedded state machines that performs DRAM initialization based on the DRAM type programmed into the PHY registers.

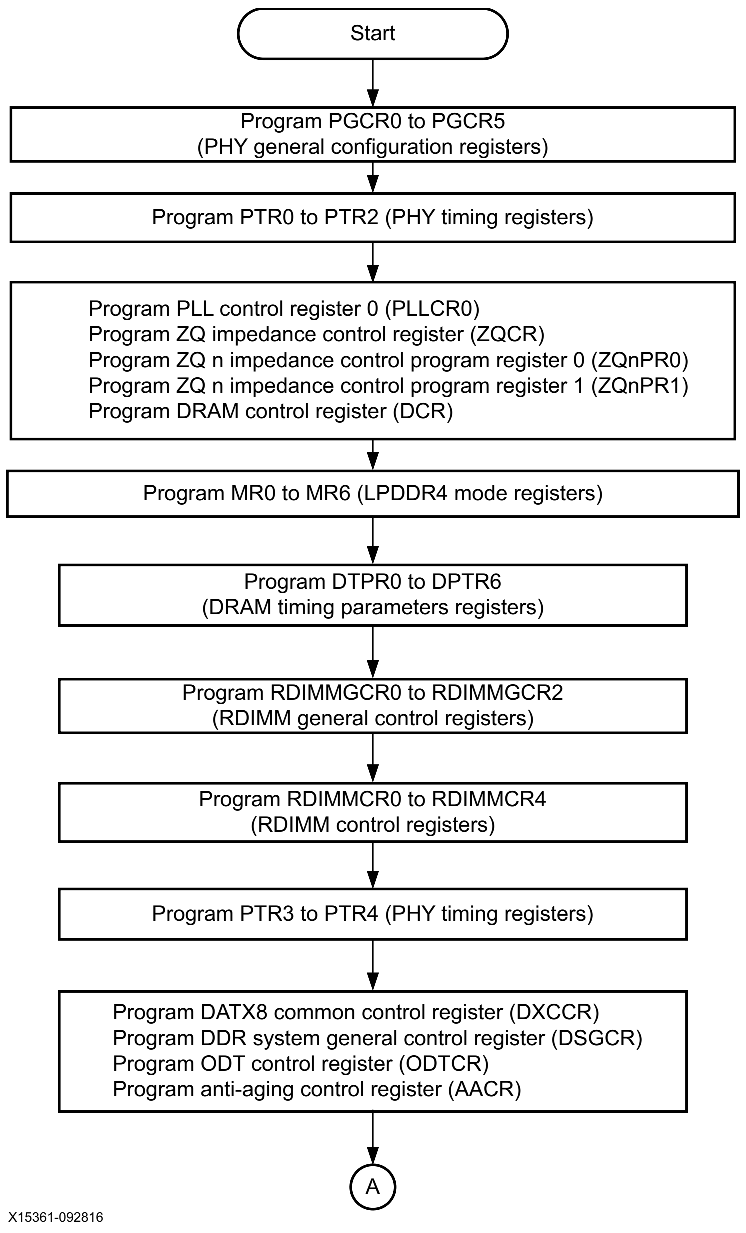
To trigger DRAM initialization using the PHY the PIR = `x0000_0081` or PIR = `x0010_0081` when RDIMM is required. Alternatively, you can have the DDR controller perform DRAM initialization. To do this, the PIR must be programmed with PIR = `0004_0001` to transfer control of the DFI interface from the PUB to the DDR controller.

Data Training

After the PHY and SDRAM are successfully initialized, the PHY is trained for optimum operating timing margins. This includes CA training (LPDDR3 only), write leveling, the training of the DQS gating during reads, write latency adjustment, bit deskew, and the training of the read and write data eyes. Table 17-39 lists the various training options. Figure 17-14 through Figure 17-17 shows the flowcharts.

Table 17-39: **Data Training**

CA training	This feature of the LPDDR3 memory is used for optimizing the setup and hold times of the CA bus relative to the memory clock.
Write leveling	This training is used to compensate for skew by aligning the clock with the data strobe at each SDRAM.
DQS gate training	This training executes a series of reads sweeping the read DQS gate over possible gating positions to discover an appropriate placement that results in successful read operations.
Write DQS2DQ training	LPDDR4 memory devices use an unmatched DQS-DQ path to enable high-speed performance and save power. As a result, the DQS strobe is trained to arrive at the DQ latch center-aligned with the data eye. The DQ receiver latches the data present on the DQ bus when DQS reaches the latch, and DQS2DQ training is accomplished by delaying the DQ signals relative to DQS such that the data eye arrives at the receiver latch centered on the DQS transition.
Write latency adjustment	This is second level of write leveling to find if extra pipeline stages need to be been added in the write path due to the write leveling and/or the board fly-by delays. After determining the write latency, a second sequence of writes and reads are issued to validate the computed latency adjustment setting.
Data eye training (read, write)	This training is used at greater than 2133 Mb/s rates to compensate for per-bit skew due to factors such as PHY to I/O routing skews, package skews, PCB skew, etc. The PHY performs automatic training sequences for read and write deskew, which aligns the data bits to the DQ bit with the longest delay using a bit delay line (BDL). After performing bit deskew, the read and write eye centering training is executed to place the strobe in the center of the eye defined by the bits in the respective byte.
V_{REF} training	Write and read eyes should be as wide as possible to provide a stable and robust memory access. The eye position depends upon LCDL, as well as VREF values. The write and read data eye training is used to find out the best eye position by changing LCDL values with an initial calculated and programmed VREF setting.



Start
Program PGCR0 to PGCR5 (PHY general configuration registers)
Program PTR0 to PTR2 (PHY timing registers)
Program PLL control register 0 (PLLCR0)
Program ZQ impedance control register (ZQCR)
Program ZQ n impedance control program register 0 (ZQnPR0)
Program ZQ n impedance control program register 1 (ZQnPR1)
Program DRAM control register (DCR)
Program MR0 to MR6 (LPDDR4 mode registers)
Program DTPR0 to DPTR6 (DRAM timing parameters registers)
Program RDIMMGCR0 to RDIMMGCR2 (RDIMM general control registers)
Program RDIMMCR0 to RDIMMCR4 (RDIMM control registers)
Program PTR3 to PTR4 (PHY timing registers)
Program DATX8 common control register (DXCCR)
Program DDR system general control register (DSGCR)
Program ODT control register (ODTCR)
Program anti-aging control register (AACR)
A
X15361-092816

Figure 17-14: **Data Training Flowchart 1**

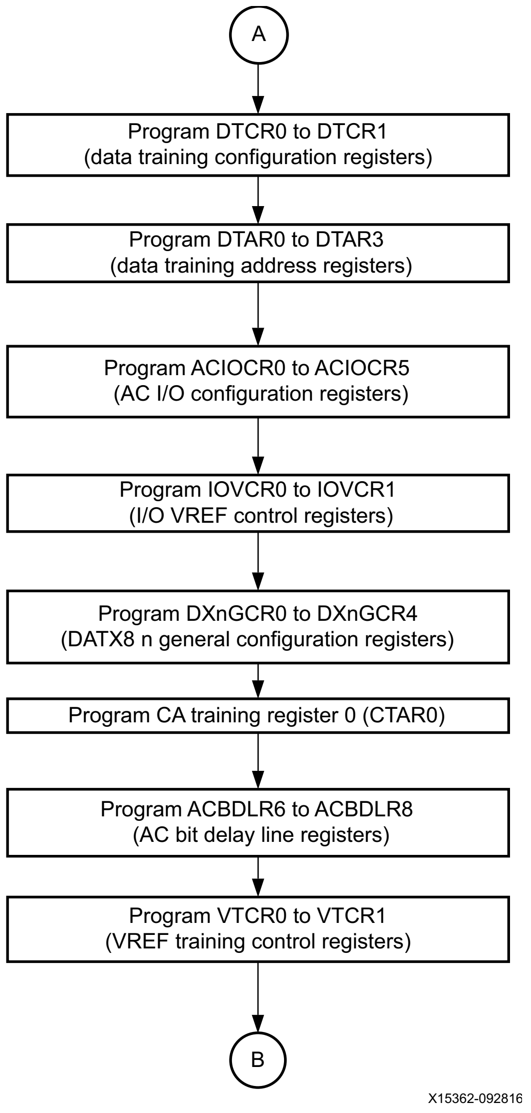

A
Program DTCR0 to DTCR1
(data training configuration registers)
Program DTAR0 to DTAR3
(data training address registers)
Program ACIOCR0 to ACIOCR5
(AC I/O configuration registers)
Program IOVCR0 to IOVCR1
(I/O VREF control registers)
Program DXnGCR0 to DXnGCR4
(DATX8 n general configuration registers)
Program CA training register 0 (CTAR0)
Program ACBDLR6 to ACBDLR8
(AC bit delay line registers)
Program VTCR0 to VTCR1
(VREF training control registers)
B
X15362-092816

Figure 17-15: **Data Training Flowchart 2**

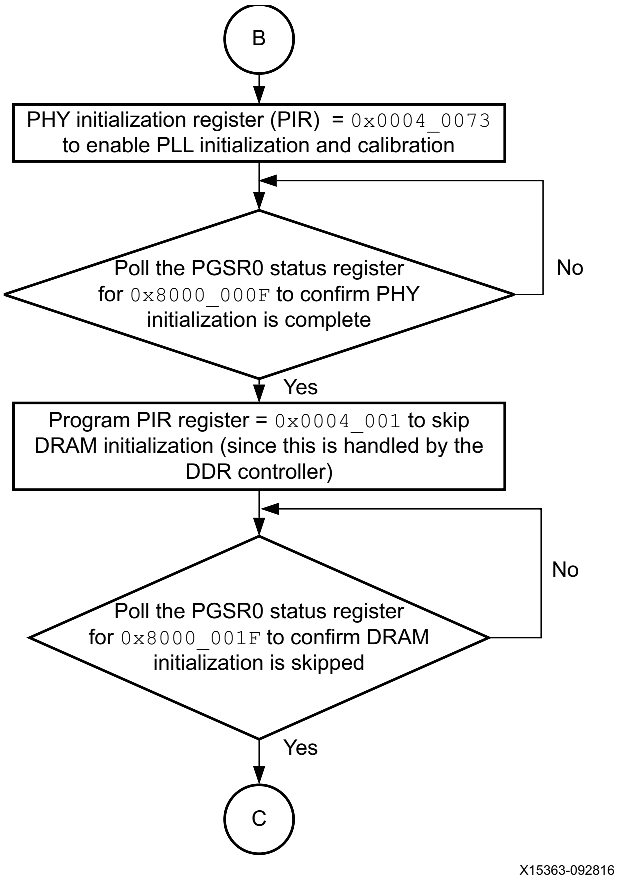

B
PHY initialization register (PIR) = 0x0004_0073 to enable PLL initialization and calibration
Poll the PGSR0 status register for 0x8000_000F to confirm PHY initialization is complete
No
Yes
Program PIR register = 0x0004_001 to skip DRAM initialization (since this is handled by the DDR controller)
Poll the PGSR0 status register for 0x8000_001F to confirm DRAM initialization is skipped
No
Yes
C
X15363-092816

Figure 17-16: **Data Training Flowchart 3**

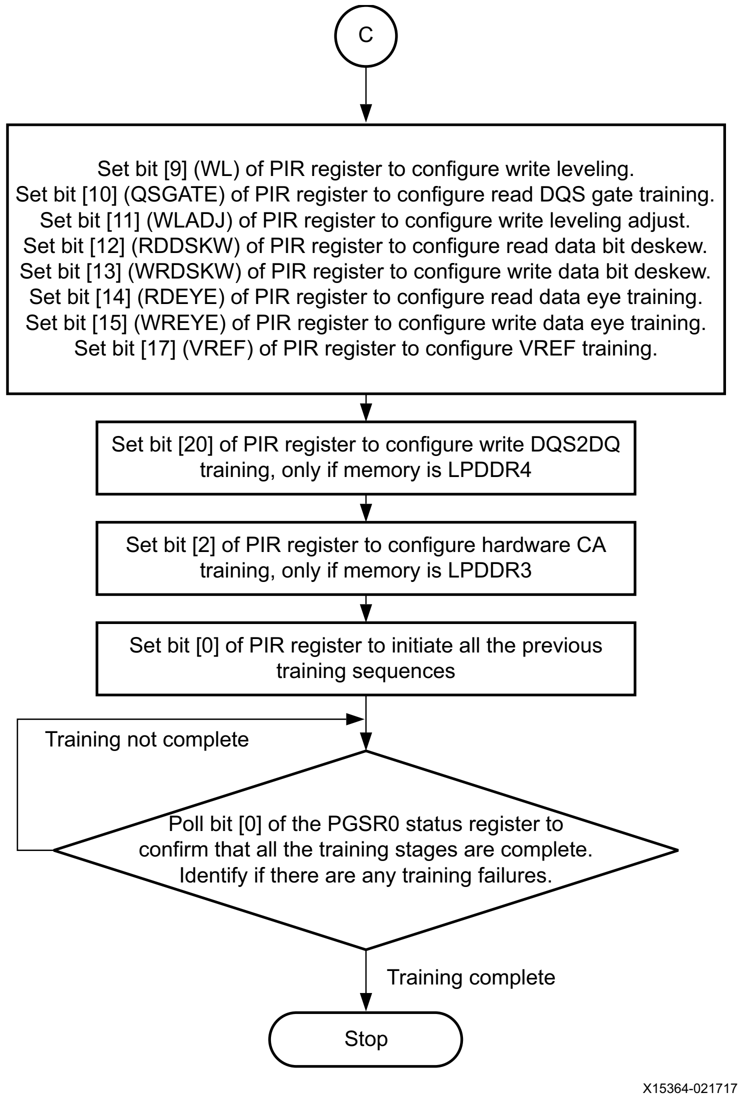

C
Set bit [9] (WL) of PIR register to configure write leveling.
Set bit [10] (QSGATE) of PIR register to configure read DQS gate training.
Set bit [11] (WLADJ) of PIR register to configure write leveling adjust.
Set bit [12] (RDDSKW) of PIR register to configure read data bit deskew.
Set bit [13] (WRDSKW) of PIR register to configure write data bit deskew.
Set bit [14] (RDEYE) of PIR register to configure read data eye training.
Set bit [15] (WREYE) of PIR register to configure write data eye training.
Set bit [17] (VREF) of PIR register to configure VREF training.
Set bit [20] of PIR register to configure write DQS2DQ training, only if memory is LPDDR4
Set bit [2] of PIR register to configure hardware CA training, only if memory is LPDDR3
Set bit [0] of PIR register to initiate all the previous training sequences
Training not complete
Poll bit [0] of the PGSR0 status register to confirm that all the training stages are complete. Identify if there are any training failures.
Training complete
Stop
X15364-021717

Figure 17-17: **Data Training Flowchart 4**

Reading DRAM Configuration Mode Registers

This section describes how to perform mode register reads and writes via software. Mode register reads (MRR) are applicable only to LPDDR2/LPDDR3/LPDDR4, and are used to read configuration and status data from mode registers in the SDRAM. Mode register writes (MRW or MRS) are applicable to all supported DDR protocols, and are used to write configuration data to mode registers in the SDRAM.

For DDR4, the PS DDR also supports Multi-purpose register (MPR) reads and writes.

Mode Register Accesses

The basic sequence is as follows:

1. Poll MRSTAT.mr_wr_busy until it is 0. This checks that there is no outstanding MR transaction. No writes should be performed to MRCTRL0 and MRCTRL1 if MRSTAT.mr_wr_busy = 1.
2. Write MRCTRL0.mr_type = 1 (for read), and MRCTRL0.mr_rank = 0x1 or 0x2 (depending on which rank you want to read).
3. Write MRCTRL1[15:8] to the address of the mode register to be read.
4. In a separate APB transaction, write MRCTRL0.mr_wr to 1. This bit is self-clearing, and triggers the MR transaction. The DDRC then asserts the MRSTAT.mr_wr_busy while it performs the MR transaction to SDRAM, and no further accesses can be initiated until it is deasserted.
5. Read MRCTRL0.mr_wr to make sure it has been cleared back to 0.
6. Read MRSTAT.mr_wr_busy to make sure the MRR has completed.
7. Read DDR_QOS_CTRL.DDRC_MRR_STATUS to look for bit 0 = 1 and bits 3:1 greater than 0.
8. Read DDR_QOS_CTRL.DDRC_MRR_DATA0 to see the results of the MRR.
9. Read DDR_QOS_CTRL.DDRC_MRR_DATA11 to reset the MRR read FIFO RD pointer.
10. Repeat to read other registers.

For example, the sequence to read MR8 is:

```
configparams force-mem-accesses 1

#Check DDRC.MRSTAT.mr_wr_busy == 0
mrd 0xfd070018

#Write DDRC.MRCTRL0.mr_rank and mr_type to indicate read from rank 0.
mwr 0xfd070010 0x11

#Write DDRC.MRCTRL1[15:8] to the MR address to be read, in this case 8.
mwr 0xfd070014 0x800

#Write DDRC.MRCTRL0.mr_rank and mr_type to indicate read from rank 0, this time setting
#bit 31 = 1 to initiate the MRR.
mwr 0xfd070010 0x80000011

#Read DDRC.MRCTRL0 to look for bit 31 to have been cleared
mrd 0xfd070010

#Check DDRC.MRSTAT.mr_wr_busy == 0
mrd 0xfd070018

#Check DDR_QOS_CTRL.DDRC_MRR_STATUS to look for 0x3 or higher
mrd 0xfd090518

#Read DDR_QOS_CTRL.DDRC_MRR_DATA0 to see the results of the MRR
mrd 0xfd09051c

#Read DDR_QOS_CTRL.DDRC_MRR_DATA11 to reset the MRR read FIFO RD pointer.
mrd 0xfd090548
```

Multi-Purpose Register (DDR4 Only)

This section describes how MPR reads and MPR writes are performed. The DDR4 SDRAMs contain four 8-bit programmable MPRs that can be used for DQ training, CA parity log, MRS readout, or for vendor specific purposes. The registers can be accessed when the SDRAM is in MPR mode and MRCTRL0.mpr_en is set to 1.

For an MPR write, the SoC core must perform the following steps:

1. Issue MRS command to SDRAM MR3 to put the SDRAM into MPR mode. The register MRCTRL0.mpr_en must be set to 0. The MPR page selection, MR3[1:0] must also be selected at this time.
2. Wait until MRSTAT.mr_wr_busy is 0. Write MRCTRL1.mr_data, where MRCTRL1.mr_data[7:0] = <MPR data>.
3. Write MRCTRL0, where MRCTRL0.mr_addr = MPR Location, MRCTRL0.mr_type = write, MRCTRL0.mr_wr = 1 and MRCTRL0.mpr_en = 1. This causes the DDRC to issue the MPR Write.
4. Issue MRS command to SDRAM MR3 to exit the SDRAM from MPR mode. The register MRCTRL0.mpr_en must be set to 0.

For an MPR Read, the SoC core must perform the following steps:

1. If retry is enabled by CRCPARCTRL1.crc_parity_retry_enable = 1, disable reads and writes from being issued on the DFI by setting DBG1.dis_dq = 1 and polling DBGCAM.wr_data_pipeline_empty and DBGCAM.rd_data_pipeline_empty, to ensure that all outstanding commands have been sent on the DFI. Poll CRCPARSTAT.cmd_in_err_window until it equals 0, to ensure that no parity error has occurred. If software intervention is enabled by CRCPARCTL1.alert_wait_for_sw, also monitor CRCPARSTAT.dfi_alert_err_int and CRCPARSTAT.dfi_alert_err_fatl_int during the polling CRCPARSTAT.cmd_in_err_window. If one or more of them are asserted before the polling is done, retry procedure must be completed prior to the subsequent steps.

   Note: If software performs MPR read during software intervention time of retry to read parity/CRC error log from SDRAM, do not poll CRCPARSTAT.cmd_in_err_window.

2. Issue MRS command to SDRAM MR3 to put the SDRAM into MPR mode. The register MRCTRL0.mpr_en must be set to 0. The MPR page selection, MR3[1:0] and read format, MR3[12:11] must also be selected at this time. A typical sequence is:

```
#Read value in DDRC.INIT4.emr3
set curval "0x[string range [mrd -force 0xfd0700E0] end-8 end]"
set current_emr3 [expr {$curval & 0x0000FFFF}]

#Clear bits 0, 1, 2, 11, and 12
set wrval [expr {$current_emr3 & 0x0000E7F8}]

#Set MPR page [1:0], MPR mode [2], and Read Format [12:11]
#Read Format: 00=serial, 01=parallel, 10=staggered
#Page 0 is parallel, others must be serial
set page 0x2
set mpr_mode [expr {1 << 2}]
set read_format [expr {00 << 11}]
set wrval [expr {$wrval + $page + $mpr_mode + $read_format}]

#Set MRCTRL0 bits to write to rank0 of MR3
mwr 0xFD070010 0x00003010

#Set MRCTRL1 to write data
mwr 0xFD070014 $wrval

#Set MRCTRL0 bit 31 to trigger the write to MR3
mwr 0xFD070010 0x80003010
```

3. Wait until MRSTAT.mr_wr_busy is 0. Write MRCTRL1.mr_data, where MRCTRL1.mr_data[1:0] = 00.

   ```
   #Read MRSTAT.mr_wr_busy
   mrd 0xFD070018

   #Set MRCTRL1.mr_data = 0
   mwr 0xFD070014 0x00000000
   ```

4. Write MRCTRL0, where MRCTRL0.mr_addr = MPR Location, MRCTRL0.mr_type = read, MRCTRL0.mr_wr = 1, and MRCTRL0.mpr_en = 1. This causes the DDRC to issue the MPR Read.

   ```
   #Set MPR location [15:12], MR type [0] = 1 (read), MR rank [4], and
   MPR Enable [1] = 1 (MPR)
   set mpr_location [expr {0x3 << 12}]
   set mr_type 1
   set mr_rank [expr {1 << 4}]
   set mpr_enable [expr {0 << 1}]
   set mrctrl0_val [expr {$mpr_location + $mr_type + $mr_rank +
   $mpr_enable}]

   #Set MRCTRL0
   mwr 0xFD070010 $mrctrl0_val

   #Set MRCTRL0 bit 31 to trigger the write to MR3
   set mr_wr [expr {1 << 31}]
   set mrctrl0_val [expr {$mrctrl0_val + $mr_wr}]
   mwr 0xFD070010 $mrctrl0_val
   ```

   The mode register contents are available on DDR_QOS_CTRL.DDRC_MRR_DATA[11:0], qualified by DDR_QOS_CTRL.DDRC_MRR_STATUS.VALID.

   DDR_QOS_CTRL.DDRC_MRR_STATUS[0] == 1 indicates that valid data is available in the read FIFO.

   DDR_QOS_CTRL.DDRC_MRR_STATUS[3:1] indicates how many valid data entries are available in the FIFO.

   Each FIFO entry is 288 bits, representing a burst of four 72-bit values.

   DDRC_MRR_DATA0 = UI0 of bytes 3:0

   DDRC_MRR_DATA1 = UI0 of bytes 7:4

   DDRC_MRR_DATA2 = UI0 of the ECC byte

   This is the MSB for serial mode reads.

This pattern repeats four times:

DDRC_MRR_DATA3 = UI1 of bytes 3:0

up to:

DDRC_MRR_DATA11 = UI3 of the ECC byte

Reading DDRC_MRR_DATA11 pops the FIFO and makes the next 288 bits available in DDRC_MRR_DATA[11:0].

When all the data has been read from the FIFO, it is safe to move to the next step.

5. Issue MRS command to SDRAM MR3 to exit the SDRAM from MPR mode. The register MRCTRL0.mpr_en must be set to 0.
6. If reads/writes have been disabled in step 1, re-enable reads and writes on the DFI by setting DBG1.dis_dq = 0.

On-chip Memory

Introduction

The on-chip memory (OCM) contains 256 KB of RAM. It supports a 128-bit AXI slave interface port. The OCM has eight exclusive access monitors that can simultaneously monitor up to eight exclusive access transactions.

The OCM supports AXI read and write throughput for RAM access by implementing a double-width memory (256 bits) to maximize the read and write bandwidth. Maximum bandwidth is achieved when the read/write accesses are a multiple of 256 bits with 256-bit aligned addresses. The OCM memory unit implements a read-modify-write operation to accommodate writes that are not 256 bits or unaligned to an 8 byte boundary.

Arbitration between the read and write channels of the OCM switch ports is performed within the OCM.

Accesses to the OCM must pass through the OCM protection unit, OCM_XMPU. The entire 256 KB of RAM is divided into 64 blocks (4 KB each) and assigned security attributes independently.

Features

- OCM RAM size is 256 KB.

  Note: The OCM clock is the same as the cpu_r5_clk.
- Clock frequency up to 600 MHz.
- Ensures low memory access latency for the RPU MCore AXI accesses.
- Exclusive access support, implements eight exclusive access monitors.
- AXI port exclusive access.
- Round-robin arbitration.
- 64-bit ECC with error injection scheme to check ECC errors.
- Four island power down and data retention support.
- Memory protection and TrustZone support.

On-chip Memory Functional Description

The OCM memory is mainly composed of four memory banks. The OCM also contains arbitration, framing, ECC, and interrupt logic in addition to the RAM arrays. The OCM architecture diagram is shown in Figure 18-1.

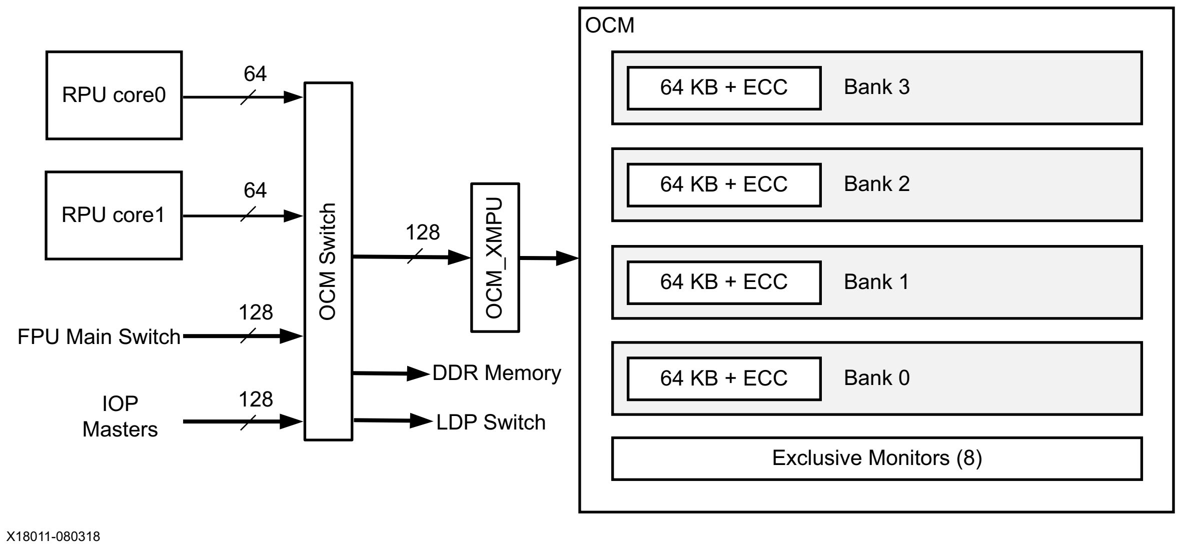


Figure 18-1: **OCM Memory Architecture**

Address Mapping

The address range assigned to the OCM memory exists in the higher 256 KB of the 32-bit address map. This cannot be modified.

Mapping Summary

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 PMU. The mapping summary is listed in Table 18-1.

Table 18-1: **OCM Mapping Summary**

Address Range	Size	Memory Bank
`FFFC_0000—FFFC_FFFF`	64 KB	0
`FFFD_0000—FFFD_FFFF`	64 KB	1
`FFFE_0000—FFFE_FFFF`	64 KB	2
`FFFF_0000—FFFF_FFFF`	64 KB	3

64-bit ECC Support

The OCM supports 64-bit wide ECC functionality to detect multi-bit errors and recover from a single-bit memory fault. The syndrome bits are calculated on a 64-bit basis. For every 64 bits processed by the ECC, both write channels generate and append eight additional syndrome bits. This adds 32 syndrome bits per memory location (256 bit).

On a write, if all bytes are being written, the ECC is generated (on a per 64-bit basis) and written into the ECC RAM along with the write-data that is written into the data RAM. If one or more bytes are not written (the byte enables are disabled), then the data RAM is first read, and the read data is corrected and merged with the write data. The merged write data is written with all bytes enabled. If the read (of the read-modify-write operation) results in an uncorrectable error, then the write is not performed and an AXI SLVERR is generated.

If a correctable or uncorrectable error is detected during a read, then the read address is captured in the ocm.OCM_{CE, UE}_FFA registers, depending on the type of error. For a correctable error, an optional interrupt is generated. If the ECC error status register (OCM_ISR) is not cleared by software, any further error information is not recorded.

1-bit or 2-bit errors per 64-bit (ECC WORD) can be injected based on the memory-mapped register value (64 + 8 bits) that is XOR-ed with the written data and syndrome.

Low Power Operation

The OCM memory implements four different banks with the two MSBs of the address determining the bank that is accessed. Each bank is implemented in a separate power-gated island that is controlled by the PMU. The PMU can also configure the RAMs in a retention state, in addition to the complete powered-down state. In the case of an access to a bank that is powered down or is in retention, the OCM generates an address decode error.

On-chip Memory Register Overview

The OCM implementation register is provided in Table 18-2. Further details are in the *Zynq UltraScale+ MPSoC Register Reference* (UG1087) [Ref 4].

Table 18-2: **OCM Register Overview**

Type	Register Name	Description
Error control	OCM_ERR_CTRL	Enable/disable a error response.
	OCM_ECC_CTRL	Enable/disable ECC. Detect only capability can be enabled only when ECC is enabled.
Interrupt	OCM_ISR OCM_IMR OCM_IEN OCM_IDS	Interrupt control and status registers.
Correctable error registers	OCM_CE_FFA OCM_CE_FFD0 OCM_CE_FFD1 OCM_CE_FFD2 OCM_CE_FFD3 OCM_CE_FFE	Correctable error-related information pertaining to various data banks can be found in these registers.
Uncorrectable error registers	OCM_UE_FFA OCM_UE_FFD0 OCM_UE_FFD1 OCM_UE_FFD2 OCM_UE_FFD3 OCM_UE_FFE	Uncorrectable error-related information pertaining to various data banks can be found in these registers.
Fault injection	OCM_FI_D0 OCM_FI_D1 OCM_FI_D2 OCM_FI_D3 OCM_FI_SY OCM_FI_CNTR	Fault injection data registers are used to inject faults in any of the 64 Kb data banks.
Information	OCM_IMP	Provides information regarding the amount of OCM memory currently implemented on the device.
Miscellaneous	OCM_CLR_EXE	Clear exclusive access monitors.
	OCM_RMW_UE_FFA	Read-modify-write uncorrectable error log.
	OCM_SAFETY_CHK	Safety endpoint connectivity check register.
	OCM_PRDY_DBG	Debug register.

On-chip Memory Programming Model

The flowchart in Figure 18-2 shows the details regarding error/fault injection and detection in OCM memory.

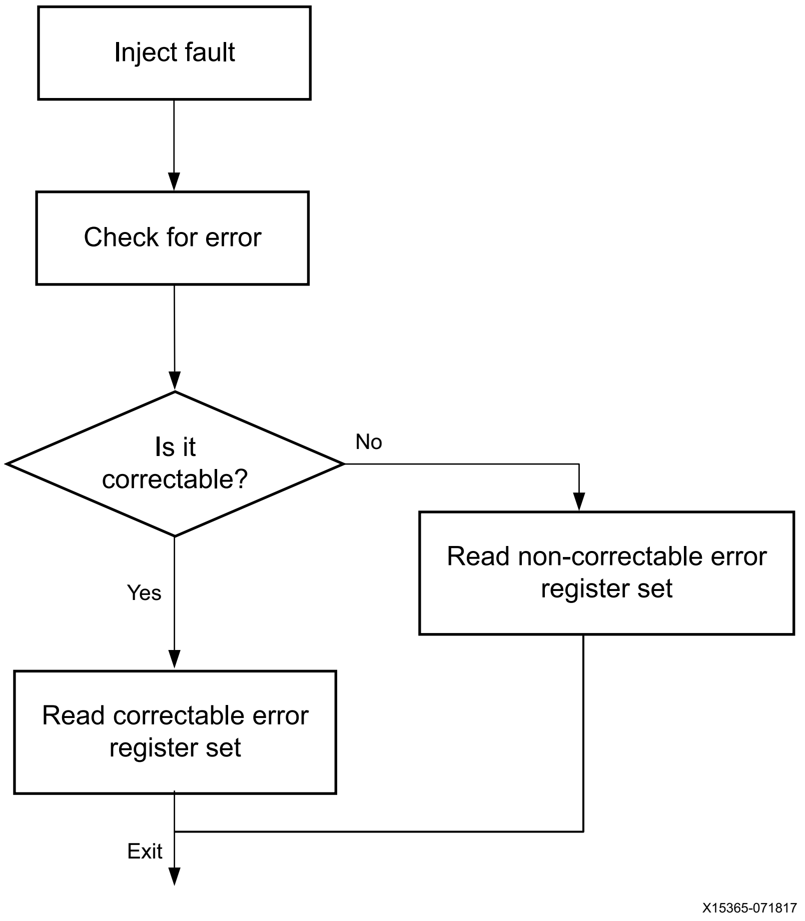

Inject fault
Check for error
Is it correctable?
No
Yes
Read non-correctable error register set
Read correctable error register set
Exit
X15365-071817

Figure 18-2: **OCM Error Checking Flowchart**

Inject Fault

1. Enable error response by setting the third bit of the ocm.OCM_ERR_CTRL register.
2. Enable ECC by setting the zeroth bit of the ocm.OCM_ECC_CTRL register.
3. To only detect single bit errors, set the first bit of the ocm.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.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.OCM_FI_CNTR register.
6. A fault can be injected into the syndrome bits using the ocm.OCM_FI_SY register. Faults in the data words can be injected using the ocm.OCM_FI_D{0:3} registers.
7. Interrupts can be enabled for different errors by setting the required bits of the ocm.OCM_IEN register.
8. Unwanted interrupts can be disabled by setting the required bits of the ocm.OCM_IDS register.
9. Reading the ocm.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.OCM_IEN and ocm.OCM_IDS registers.

Check for Error

If errors occur due to a fault injection or other reasons, an interrupt is generated. The ocm.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.OCM_ISR register for information on whether the error is correctable or uncorrectable.

Read Correctable Error Register Set

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.OCM_CE_FFA register.
2. Retrieve ECC syndrome bits of corrected error. Read ocm.OCM_CE_FFE [SYNDROME] bit field.
3. Retrieve corrected data. Read the four data words using the ocm.OCM_CE_FFD{0:3} registers.

Read Uncorrectable Error Register Set

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.OCM_UE_FFA register.
2. Retrieve ECC syndrome bits of uncorrected error. Read ocm.OCM_UE_FFE [SYNDROME] bit field.
3. Retrieve uncorrected data. Read the four data words using the ocm.OCM_UE_FFD{0:3} registers.

DMA Controller

Introduction

The general purpose direct memory access (DMA) controller supports memory to memory and memory to I/O buffer transfers. There are two instances of the general purpose DMA controller: LPD DMA and FPD DMA. The LPD DMA can have I/O coherent access. The FPD DMA transfers are never hardware coherent with CCI. This chapter refers to the FPD and LPD DMA controllers collectively as DMA. The 8-channel LPD and FPD DMA controllers are architecturally identical except for coherency, command buffer size, and data width.

- FPD DMA: 128-bit AXI data interface, 4 KB command buffer, non-coherent with CCI.
- LPD DMA: 64-bit AXI data interface, 2 KB command buffer, I/O coherent with CCI.

Features

- AXI-4 interface, burst length is limited to 16 to provide AXI-3 compatibility.
- Source (SRC) and destination (DST) payloads can start and end at any alignment. The DMA takes care of 4 KB boundary crossing.
- Over fetching can be enabled/disabled per channel.
- Each channel can be programmed as secure or non-secure.
- Programmable number of outstanding transactions per channel on the source side. Up to 32 outstanding transactions supported on each AXI channel.
- Periodic transaction scheduling. Period can be independently programmed per channel.
- Simple register-based DMA and scatter gather (SG) DMA modes. Hybrid descriptor support in SG DMA mode.
- Read-only DMA mode.
    - Read data is discarded in this mode.
    - Read-only feature is only supported in simple DMA mode.

Note: For memory to I/O buffer transfers, the peripheral should support DMA transfers.

- Write-only DMA mode.
  - Data specified in the control register is written to DST address locations, no read command is issued.
  - Write-only feature is only supported in simple DMA mode.
- Common buffer is automatically shared among all enabled DMA channels.
- Support for DMA START, STOP, and PAUSE features.
- Interrupt accounting support.
- Descriptor prefetch support to maximize DMA efficiency. 100% efficiency with 128-bit aligned SRC and DST payloads.
- Support for error recovery.
- INCR and FIXED type burst supported. Fixed bursts are only supported in simple DMA mode.
- Independent AXI burst length is supported on both the SRC and DST sides.
- Flow control on a per channel basis via the PL EMIO interface.

DMA Controller Functional Description

The Zynq UltraScale+ MPSoC DMA (Figure 19-1) is a general-purpose DMA for memory-to-memory, memory-to-I/O, I/O-to-memory, and I/O-to-I/O transfers.

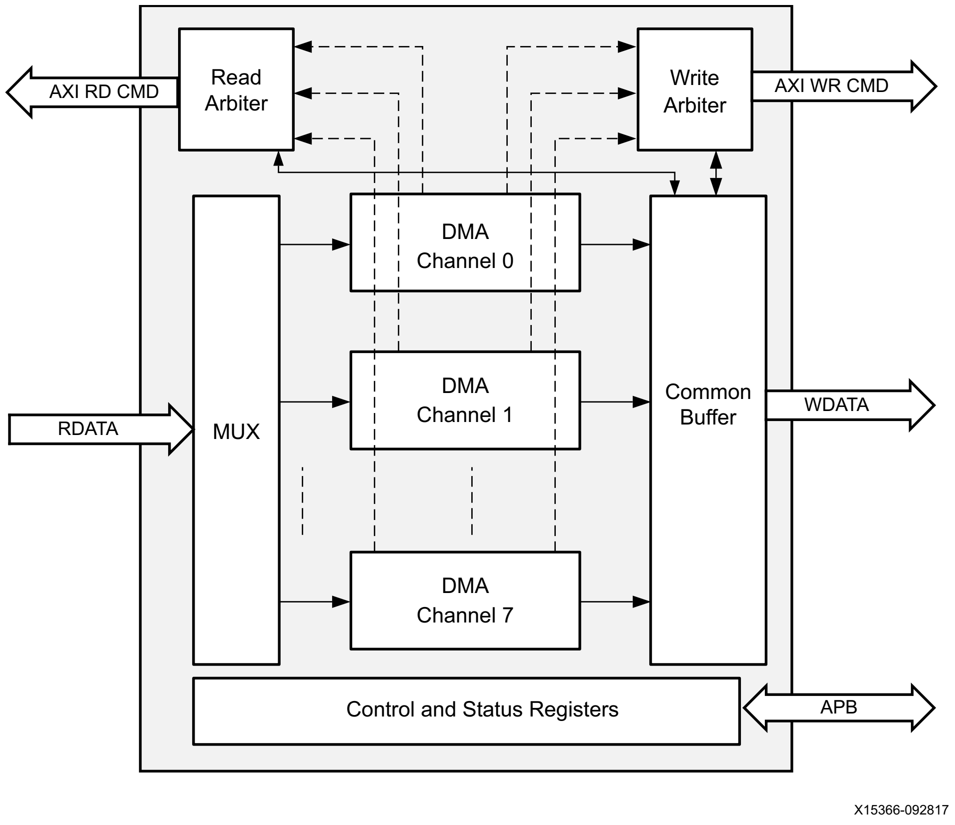


Figure 19-1: **DMA Block Diagram**

The DMA acts as an AXI-4 master. Each channel can be independently enabled or disabled at any time. DMA supports pause functionality per-channel, which allows software to pause the channel using a descriptor and allows software to program new sets of descriptors. Software can resume the channel once it has programmed a new set of descriptors.

The DMA implements a common buffer that is sized to allow the DMA to utilize the full AXI 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 buffer per channel. Each channel uses the available buffer on a first come first serve basis. Buffer utilization of each channel can be controlled by programming issuing capability of each channel and rate control. DMA supports two modes of operation, simple register-based DMA or 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, overfetch of data is not allowed on the read channel. For these slaves, software can disable overfetch. Software can independently enable/disable overfetch on each DMA channel. Over-fetch disable can significantly impact DMA efficiency (depends on payload alignment). Xilinx advises only using this feature if it is required by an AXI slave.

DMA Architecture

The major functional blocks are:

- Common buffer
- Arbiter
    - AXI read channel
    - AXI write channel
- DMA engine channel(s)

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 utilization of full AXI bandwidth. The size of the common buffer is 4 KB for FPD DMA and 2 KB for LPD DMA.

- Shares the full buffer space between enabled channels. When only one channel is enabled, it can use the full buffer memory space.
- Does not utilize/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.

AXI Read Arbiter

Each DMA channel has two AXI read interfaces. One interface is used for reading data buffers and the other interface is used for reading buffer descriptors. The DMA implements round-robin arbitration. Arbitration is never granted to any request if the common buffer does not have enough space. This way 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 Write Arbiter

The DMA channels share an AXI write channel. The features of the write arbiter are listed.

- Round-robin arbitration
- Common buffer flush in the event of an error

DMA Channel

The DMA channel is responsible for the bulk of the DMA operation and management.

DMA Data Flow

This section outlines the DMA model, modes, and the buffer descriptor (BD) format.

DMA Model

The LPD and FPD DMA controllers each have 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 Figure 19-2 illustrates the source and destination side scatter-gather mode buffer descriptor arrays. The buffer descriptors 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.

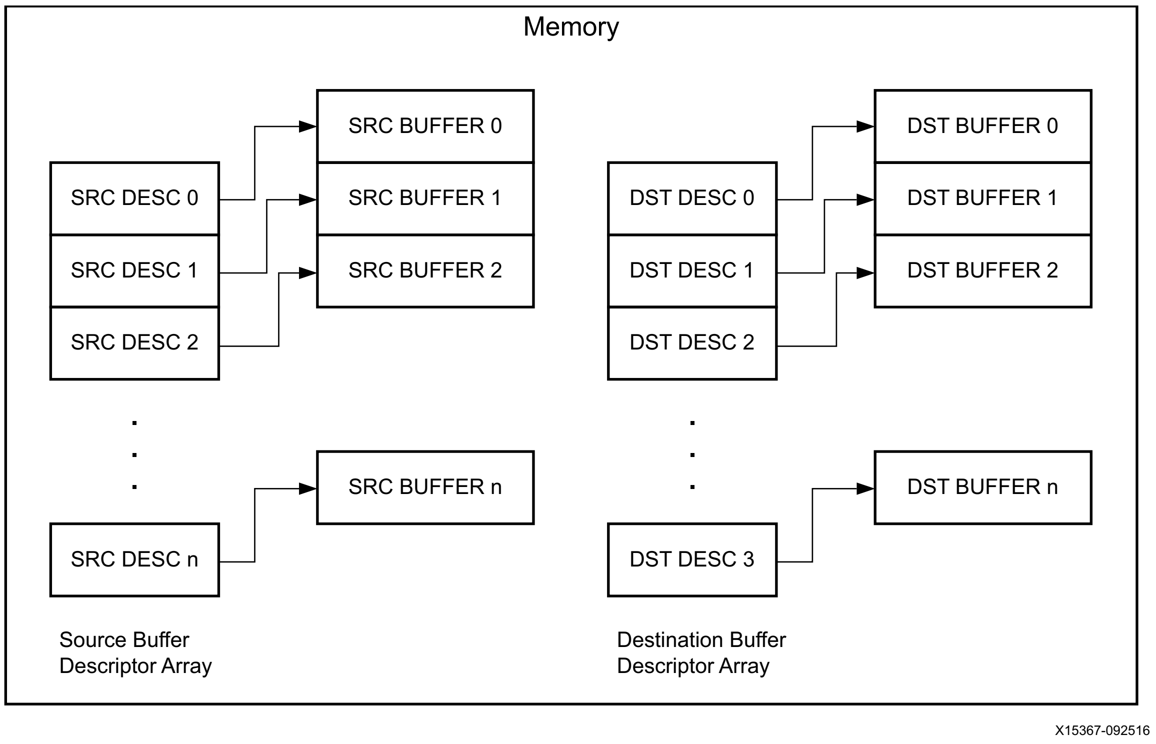


Figure 19-2: **SRC and DST Descriptors Pointing to Data Buffers**

DMA Modes

Each DMA channel can be independently programmed in one of the following DMA modes as described in this section.

Simple DMA Mode

Simple DMA mode is also known as single-command mode because the DMA performs a data transfer upon receiving a command in a single command 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 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.

Scatter Gather DMA Mode

In scatter gather mode, DMA transfer parameters are specified in memory (descriptors) and parameters in APB registers are ignored. Software programs SRC and DST descriptors programs registers to point to the start of these descriptors in memory and enables a channel. Upon receiving a channel enable, the DMA channel fetches SRC and DST descriptors from memory and uses these parameters to perform the actual data transfer. The descriptors must be configured before enabling a channel.

Descriptor 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, or a linked list of BDs, or a mixed mode wherein 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 wherein 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. Figure 19-3 shows supported descriptor modes.

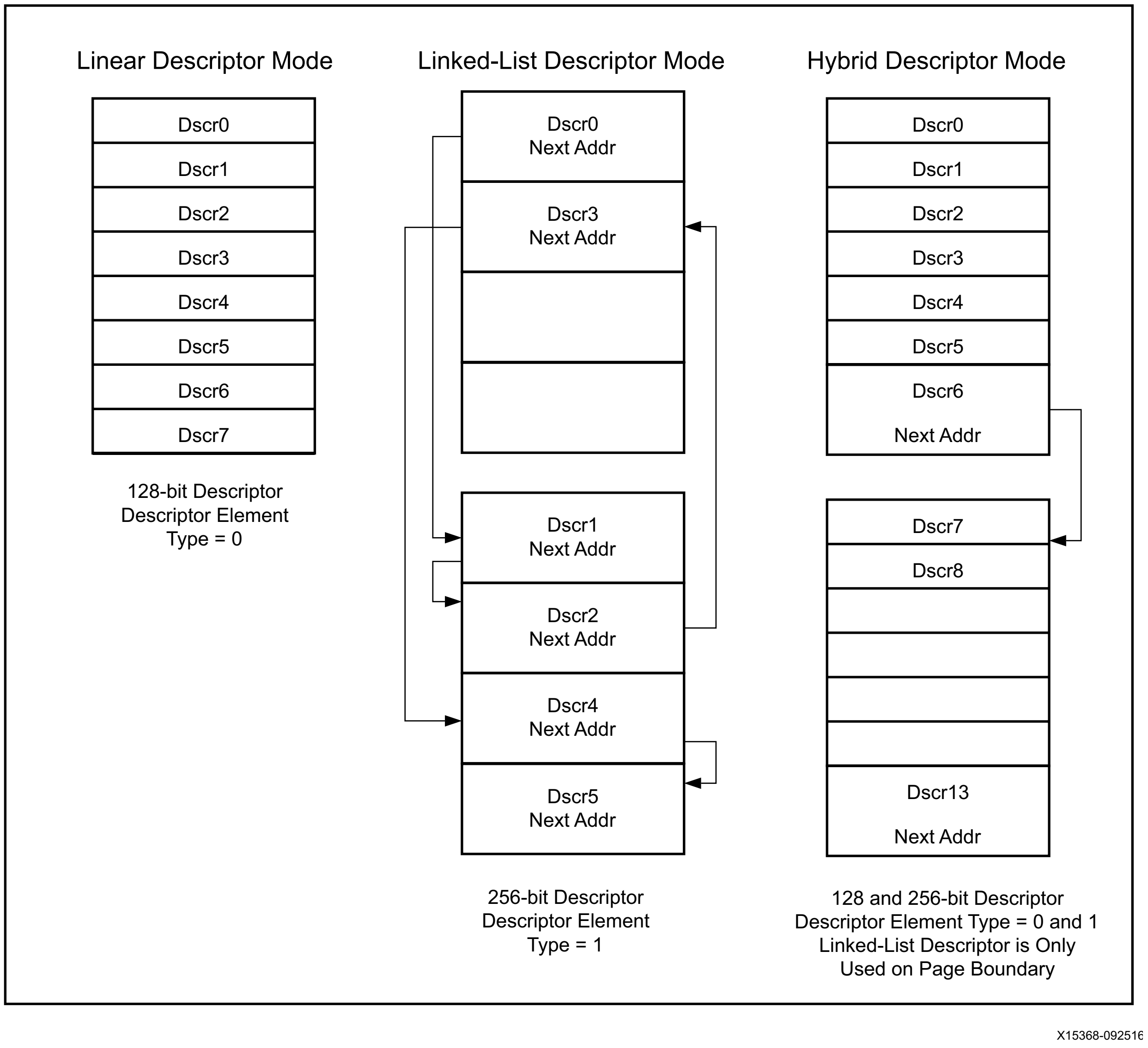


Figure 19-3: **DMA Supported Descriptor Mode Use-cases in Scatter Gather Mode**

Linear Descriptor Use Case

In the linear descriptor use case mode, BDs are stored in a linear array. In Figure 19-3, 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. Further details are shown in Table 19-1.

- Each descriptor is 128 bits wide
- Each descriptor must be 128-bit aligned
- The descriptor element type is always `0` (in linear descriptor mode).

Table 19-1: **Buffer Descriptor Format 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).

Further details are shown in Table 19-2.

- 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 requires two AXI reads.

Table 19-2: **Buffer Descriptor Format in Linked-list 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

Table 19-2: **Buffer Descriptor Format in Linked-list Descriptor Mode *(Cont'd)***

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 19-3) 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 DMA never prefetches across the 4 KB boundary.
- The circular descriptor should always have at least one link-list element.
- Descriptors are not updated back to the memory. For instance, once 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.

Buffer Descriptor Format

The buffer descriptor (BD) format used in scatter gather (SG) mode is shown in Table 19-3. 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 SG mode.

Table 19-3: **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
3	CNTL	4	[0]	Coherency: `0`: AXI transactions generated to process the descriptor payload are marked non-coherent. `1`: AXI transactions generated to process the descriptor payload are marked coherent. ***Note:*** This bit has no effect for the FPD DMA controller.
			[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. Once 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. Once 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.

Table 19-3: **Buffer Descriptor Format *(Cont'd)***

Word Number	Field Name	Size (bytes)	Bits	Description
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`.
7	Reserved	4	[31:0]	Reserved This field exists only if the DSCR element type is set as `1`.

DMA Performance Requirements

The DMA provides 100% efficiency in the following scenario.

- Read and write descriptor payload are 128-bit aligned (in scatter-gather mode)
- SRC and DST descriptors are 256-bit aligned
- SRC and DST payload is >4 KB

100% efficiency is achieved when there is no back pressure on the read and write AXI channels and DMA fully utilizes AXI read and write channels.

DMA Interrupt Accounting

The DMA channel does not update descriptors in memory. The feedback when the descriptor is done is provided by a SRC/DST done interrupt along with an interrupt accountings counter (control register). The following interrupt accounting scheme is implemented on each DMA channel, on both the SRC and DST sides.

- The software can selectively request a completion interrupt on descriptors. Once a descriptor is processed, the DMA increments the interrupt accountings counter. The definition of a descriptor done is different on the SRC and DST sides.
- A SRC descriptor done interrupt is generated once 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 once 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). Implementing 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), then the interrupt accounting counter overflow interrupt is generated two clock cycles after the DMA done interrupt due to asynchronous boundary crossing logic.

DMA 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 required. In the case where 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 Figure 19-4 uses a source descriptor size of 8190 bytes (with a start address at `0x000_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 documented.

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)

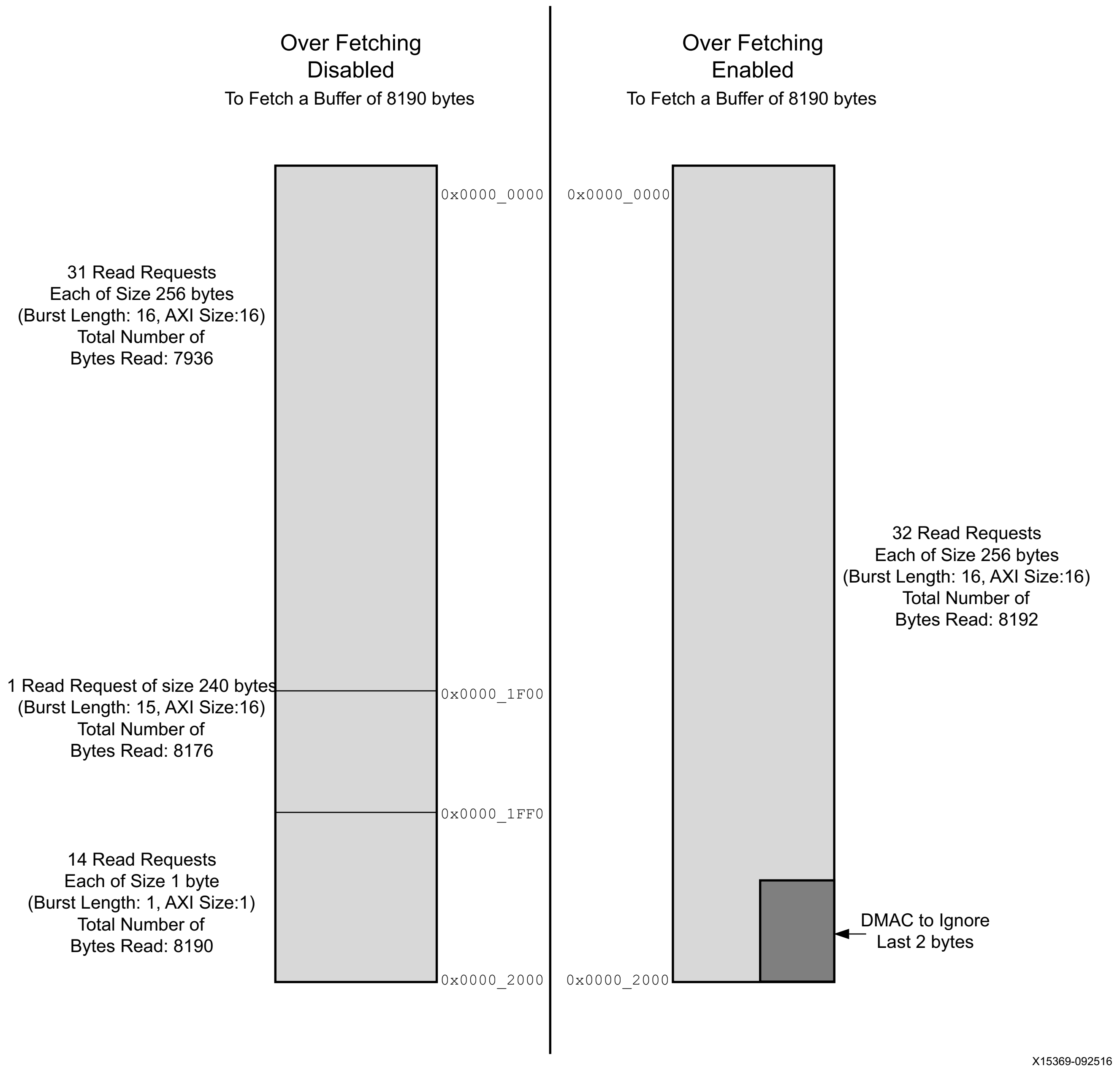

Over Fetching Disabled
To Fetch a Buffer of 8190 bytes
Over Fetching Enabled
To Fetch a Buffer of 8190 bytes
0x0000_0000
0x0000_0000
31 Read Requests Each of Size 256 bytes (Burst Length: 16, AXI Size:16) Total Number of Bytes Read: 7936
32 Read Requests Each of Size 256 bytes (Burst Length: 16, AXI Size:16) Total Number of Bytes Read: 8192
1 Read Request of size 240 bytes (Burst Length: 15, AXI Size:16) Total Number of Bytes Read: 8176
0x0000_1F00
0x0000_1FF0
14 Read Requests Each of Size 1 byte (Burst Length: 1, AXI Size:1) Total Number of Bytes Read: 8190
DMAC to Ignore Last 2 bytes
0x0000_2000
0x0000_2000
X15369-092516

Figure 19-4: **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.*

DMA 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 in each LPD and FPD DMA unit).

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.

Arbitration has no queue of its own and does not get full. Instead, it relies on the AXI queues. Bandwidth maximization depends on parameters such as transaction, length, and alignment.

Rate control remains in effect until disabled. Therefore, when any new transactions begin, the previously programmed rate controls take effect (unless disabled).

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. Software configures what accesses are flow controlled by the FCI (read/write).

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 DST (write)). Table 19-4 lists the FCI signals.

IMPORTANT: *The maximum number of credits accepted are 32.*

Table 19-4: **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
dma2pl_cack	Credit acknowledgment: • Credits are accumulated when both pl2dma_cvld and dma2pl_cack are High (TRUE). • Each FCI can accumulate up to 32 credits. • If the FCI is not enabled, the credits are flushed.
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 TVLD and TACK are TRUE.

The timing diagram for the flow control interface is as shown in Figure 19-5.

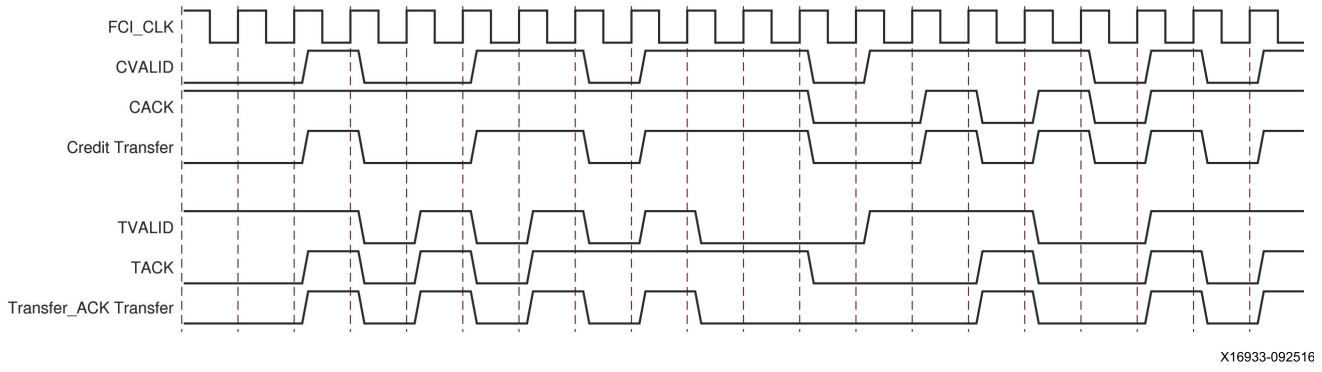


Figure 19-5: **FCI Flow Control Interface**

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.

FCI Considerations

- FCI is always disabled.
- FCI is configured to flow control the SRC (read) side upon reset.
- When FCI is enabled, both the SRC and DST sides use ARLEN for all AXI transactions.
- Software configures the FCI interface 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 will stop issuing write commands, if the PL slave does not provide a TACK in response to a 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 PROG_CELL_CNT of the ZDMA_CH_FCI register in that channel. The DMA channel stops issuing data read commands once 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.

Once 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. Channel will not issue a new transaction if credit is not available. 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 flushed until the channel is enabled. Once 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. Once done with error recovery, the channel clears both the FCI_EN and channel EN flags. Once it clears the FCI_EN, the DMA channel flushes 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. Software can calculate and provide the total number of valid transactions expected to complete the current DMA transaction to PL slave. PL slave can use a transaction valid to find where a DMA channel is in a current DMA transaction.

DMA Controller Register Overview

Table 19-5 is an overview of the DMA controller registers.

Table 19-5: **DMA Controller Registers**

Register Name	Description
ZDMA_ERR_CTRL	Enable/disable an error response.
ZDMA_CH_ISR	Interrupt status register for intrN. This is a sticky register that holds the value of the interrupt until cleared by a value of `1`.
ZDMA_CH_IMR	Interrupt mask register for intrN. This is a read-only location and can be automatically altered by either the IDS or the IEN.
ZDMA_CH_IEN	Interrupt enable register. A write of `1` to this location will unmask the interrupt. (IMR: 0).
ZDMA_CH_IDS	Interrupt disable register. A write of `1` to this location will mask the interrupt. (IMR: 1).
ZDMA_CH_CTRL0	Channel control register 0.
ZDMA_CH_CTRL1	Channel control register 1.
ZDMA_CH_FCI	Channel flow control register.
ZDMA_CH_STATUS	Channel status register.
ZDMA_CH_DATA_ATTR	Channel DATA AXI parameter register.
ZDMA_CH_DSCR_ATTR	Channel DSCR AXI parameter register.
ZDMA_CH_SRC_DSCR_WORD0	SRC DSCR word 0.
ZDMA_CH_SRC_DSCR_WORD1	SRC DSCR word 1.
ZDMA_CH_SRC_DSCR_WORD2	SRC DSCR word 2.
ZDMA_CH_SRC_DSCR_WORD3	SRC DSCR word 3.
ZDMA_CH_DST_DSCR_WORD0	DST DSCR word 0.
ZDMA_CH_DST_DSCR_WORD1	DST DSCR word 1.
ZDMA_CH_DST_DSCR_WORD2	DST DSCR word 2.
ZDMA_CH_DST_DSCR_WORD3	DST DSCR word 3.
ZDMA_CH_WR_ONLY_WORD0	Write-only data word 0.
ZDMA_CH_WR_ONLY_WORD1	Write-only data word 1.
ZDMA_CH_WR_ONLY_WORD2	Write-only data word 2.
ZDMA_CH_WR_ONLY_WORD3	Write-only data word 3.
ZDMA_CH_SRC_START_LSB	SRC DSCR start address LSB register.
ZDMA_CH_SRC_START_MSB	SRC DSCR start address MSB register.
ZDMA_CH_DST_START_LSB	DST DSCR start address LSB register.
ZDMA_CH_DST_START_MSB	DST DSCR start address MSB register.
ZDMA_CH_RATE_CTRL	Rate control count register.

Table 19-5: **DMA Controller Registers *(Cont'd)***

Register Name	Description
ZDMA_CH_IRQ_SRC_ACCT	SRC interrupt account count register.
ZDMA_CH_IRQ_DST_ACCT	DST interrupt account count register.
ZDMA_CH_CTRL2	DMA control register 2.

DMA Programming for Data Transfer

This section describes the steps and the register configuration necessary to perform DMA transfers using various modes supported by the DMA.

Simple Mode Programming

Step 1

Wait until the DMA is in an idle state by reading the STATE field of ZDMA_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 POINT_TYPE (bit 6) of the ZDMA_CH_CTRL0 register is 0.
- Program the data source buffer address LSB into register ZDMA_CH_SRC_DSCR_WORD0.
- Program the data source buffer address MSB into register ZDMA_CH_SRC_DSCR_WORD1.

Step 3

- Program the data destination buffer address LSB into register ZDMA_CH_DST_DSCR_WORD0.
- Program the data destination buffer address MSB into register ZDMA_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 ZDMA_CH_SRC_DSCR_WORD2 register.
- Program the destination data transaction size into the ZDMA_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 ZDMA_CH_DST_DSCR_WORD3 and/or ZDMA_CH_SRC_DSCR_WORD3 registers.

Step 6

In cases where the source and destination buffer are allocated as cache coherent or are flushed, there is no need to set COHRNT. Otherwise, if the source and destination buffer are not allocated as cache coherent or have not been flushed, set COHRNT in the ZDMA_CH_SRC_DSCR_WORD3 and ZDMA_CH_DST_DSCR_WORD3 registers, respectively. The COHRNT bit is valid only in case of LPD DMA. The FPD DMA does not support coherency.

Step 7

Enable the DMA channel to perform DMA transfers by setting the EN bit of ZDMA_CH_CTRL2. After enabling DMA, check for possible error conditions as described in Error Conditions.

Scatter Gather Mode Programming

DMA supports three different use cases under scatter gather mode.

- Linear
- Linked list
- Hybrid

Linear 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 Figure 19-6 captures the main steps.

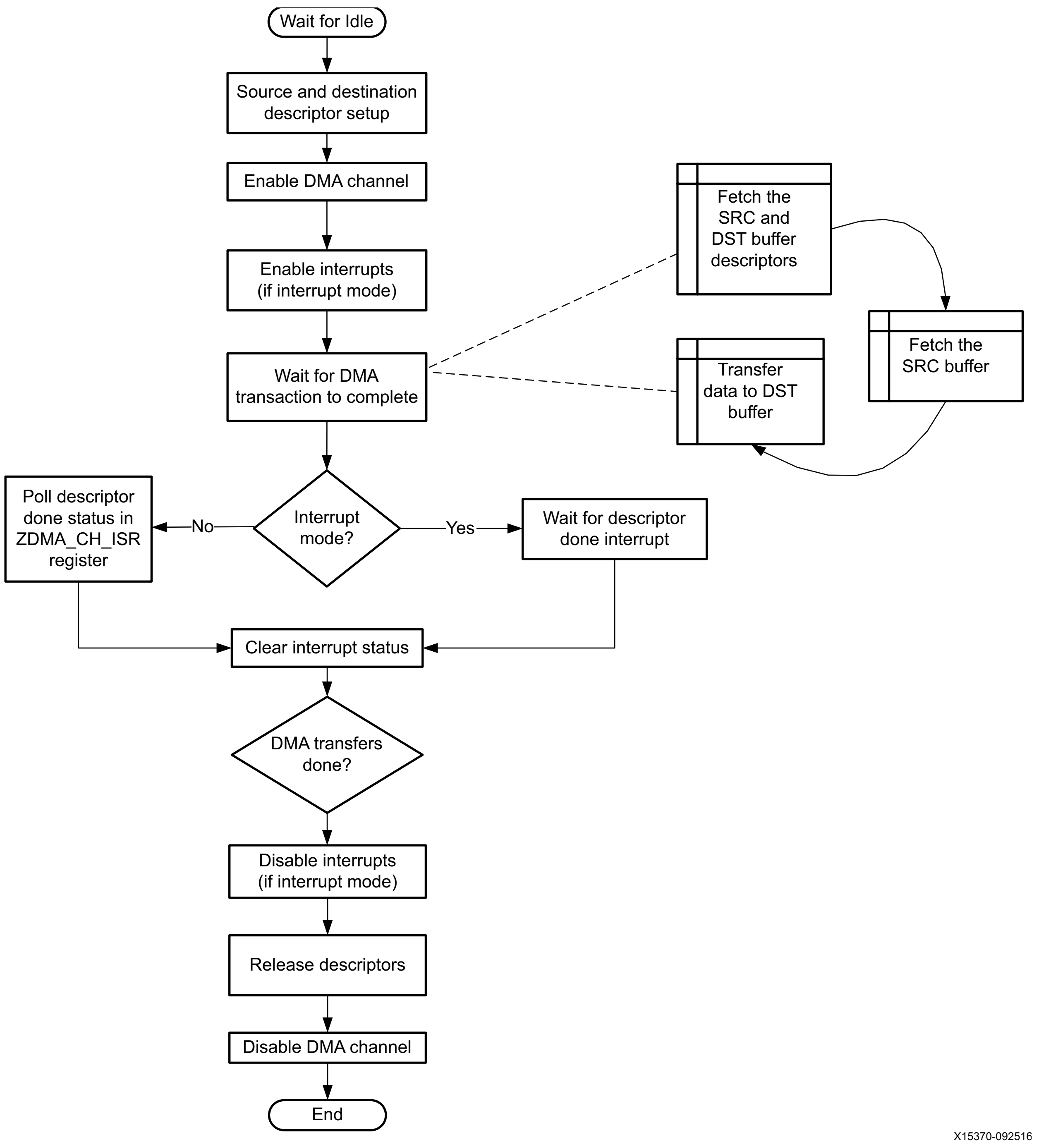

Wait for Idle
Source and destination descriptor setup
Enable DMA channel
Enable interrupts (if interrupt mode)
Wait for DMA transaction to complete
Fetch the SRC and DST buffer descriptors
Fetch the SRC buffer
Transfer data to DST buffer
Interrupt mode?
No
Yes
Poll descriptor done status in ZDMA_CH_ISR register
Wait for descriptor done interrupt
Clear interrupt status
DMA transfers done?
Disable interrupts (if interrupt mode)
Release descriptors
Disable DMA channel
End
X15370-092516

Figure 19-6: **Linear Mode DMA Programming Model**

Step 1

Ensure DMA is not in a busy state by reading the STATE field of the ZDMA_CH_STATUS and ensuring it is not `10`. 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

a. Ensure that the bit POINT_TYPE (bit 6) of the ZDMA_CH_CTRL0 is set to `1` for scatter-gather mode of operation.

b. Allocate the source buffer descriptor array in memory. Ensure that the buffer descriptor start address is 128-bit aligned (for the LPD DMA, 64-bit aligned is acceptable). In case any buffer descriptors are not allocated as cache coherent or the buffer descriptors are not flushed prior to enabling the DMA channel, set the AXCOHRNT field to `1`. The address of the first buffer descriptor in an array is written to ZDMA_CH_SRC_START_LSB and ZDMA_CH_SRC_START_MSB.

c. Allocate the destination buffer descriptor array in memory. Ensure that the buffer descriptors start address is 128-bit aligned (for the LPD DMA, 64-bit aligned is acceptable). In case any buffer descriptors are not allocated as cache coherent or the buffer descriptors are not flushed prior to enabling the DMA channel, set the AXCOHRNT field to `1`. The address of the first buffer descriptor in an array is written to ZDMA_CH_DST_START_LSB and ZDMA_CH_DST_START_MSB. The AXCOHRNT bit is valid only in the case of LPD DMA. The FPD DMA does not support coherency at buffer descriptor or buffer level.

TIP: *The buffer descriptors can also be pre-allocated during initialization time and can be reused for each DMA data transfer. In this case, Step 2 can be skipped.*

Step 3

a. Program each source data fragment to successively transfer into the allocated source buffer descriptors. The ADDR LSB and ADDR MSB fields are programmed.

b. Program the size of each source data fragment to transfer into the respective source buffer descriptor. The SIZE field is programmed.

c. Set the coherency bit if the source data buffer is not flushed or is not allocated as cache coherent prior to enabling the DMA channel for data transfer. The coherency bit is valid only in the case of LPD DMA. The FPD DMA does not support coherency at buffer descriptor or buffer level.

d. Ensure that the DSCR element type is `0`.

e. Set the INTR field if an interrupt is required after data is read for transfer. Typically, this can be set for the buffer descriptor corresponding to the last source data fragment. Setting the last source descriptor for interrupt reduces the number of interrupts received.

f. 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: *You can set `01` for pause after completing the descriptor if you want the DMA in a paused state after completing the final-buffer descriptor. The steps to come out of a paused state into a enabled/disabled state are described in Channel Paused.*

Step 4

a. Program each destination buffer fragment to successively transfer into the allocated buffer descriptors. The ADDR LSB and ADDR MSB fields are programmed.

b. Program the size of each destination data fragment to transfer into the respective destination buffer descriptor. The SIZE field is programmed.

c. Set the coherency bit if the source data buffer is not invalidated or is not allocated as cache coherent prior to enabling the DMA channel for data transfer. The coherency bit is valid only in the case of LPD DMA. The FPD DMA does not support coherency at buffer descriptor or buffer level.

d. Ensure that the DSCR element type is `0`.

e. 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 corresponding to the last destination data fragment. Setting the last destination descriptor for interrupt reduces the number of interrupts received.

f. 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: *You can set `01` for pause after completing the descriptor if you want the DMA in a paused state after completing the final-buffer descriptor. The steps to come out of a paused state into a enabled/disabled state are described in Channel Paused, page 554.*

Step 5

Enable the DMA channel by writing into the control register ZDMA_CH_CTRL2. This initiates the data DMA transfer.

Step 6

Upon transfer completion, the DMA channel provides interrupt(s) to the processor depending upon how the INTR field of the buffer descriptor(s) are set.

Software can use ZDMA_CH_IRQ_DST_ACCT and/or 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 DMA transfers. Upon updating the ZDMA_CH_IRQ_DST_ACCT and/or ZDMA_CH_IRQ_SRC_ACCT with an equal count, software can infer that the DMA data transfer is complete. Software should count only the descriptors that are enabled for interrupts. For for information on handling interrupts, refer to Interrupt Handling.

Step 7

After the DMA transfers are done, disable the DMA channel. Refer to Channel Disabled for more information.

Linked List Mode Use Case

To facilitate DMA data transfers using the linked list mode, the following steps are necessary. The linked-list mode can be used when software cannot find a contiguous set of memory that can accommodate all the buffer descriptors necessary (source and destination) as an array.

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`. In case DMA is in the PAUSE state, follow the steps to bring it out of the PAUSE state as described in Channel Paused.

Step 2:

a. Ensure the POINT_TYPE bit in ZDMA_CH_CTRL0 is set to `1`.

b. Allocate the source buffer descriptor objects in memory. Ensure that the buffer descriptor object address is 256-bit aligned. In case any buffer descriptors are not allocated as cache coherent or the buffer descriptors are not flushed prior to enabling DMA channel, set the AXCOHRNT field to `1`. The address of the first buffer descriptor in a list is written to ZDMA_CH_SRC_START_LSB and ZDMA_CH_SRC_START_MSB.

c. Allocate the destination buffer descriptor objects in memory. Ensure that the buffer descriptor object address is 256-bit aligned. In case any buffer descriptors are not allocated as cache coherent or the buffer descriptors are not flushed prior to enabling the DMA channel, set the AXCOHRNT field to `1`. The address of the first buffer descriptor in a list is written to ZDMA_CH_DST_START_LSB and ZDMA_CH_DST_START_MSB. The AXCOHRNT bit is valid only in case of LPD DMA. The FPD DMA does not support coherency at buffer descriptor or buffer level.

TIP: *The buffer descriptors can also be pre-allocated during initialization time.*

Step 3:

For each allocated source buffer descriptor object, program the following.

a. Program the source data fragment to transfer into the source buffer descriptor object. The ADDR LSB and ADDR MSB fields are programmed.

b. Program the size of each source data fragment to transfer into the source buffer descriptor object. The SIZE field is programmed.

c. Set the coherency bit if the source data buffer is not flushed or is not allocated as cache coherent. The coherency bit is valid only in the case of LPD DMA. The FPD DMA does not support coherency at buffer descriptor or buffer level.

d. Set the DSCR element type to `1`.

e. 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.

f. 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: *You can set `01` for pause after completing the descriptor if you want the DMA in a paused state after completing the final-buffer descriptor. The steps to come out of a paused state into a enabled/disabled state are described in Channel Paused, page 554.*

g. 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.

a. Program the destination data fragment to transfer into the destination buffer descriptor object. The ADDR LSB and ADDR MSB fields are programmed.

b. Program the size of each destination data fragment to transfer into each respective destination buffer descriptor. The SIZE field is programmed.

c. Set the coherency bit if the destination data buffer is not flushed or is not allocated as cache coherent. The coherency bit is valid only in the case of LPD DMA. The FPD DMA does not set support coherency at buffer descriptor or buffer level.

d. Set the DSCR element type to `1`.

e. Setting the last source descriptor for interrupt reduces the number of interrupts received.et 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.

f. 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: *You can set `01` for pause after completing the descriptor if you want the DMA in a paused state after completing the final-buffer descriptor. The steps to come out of a paused state into a enabled/disabled state are described in Channel Paused, page 554.*

g. 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 interrupt(s) to the processor depending upon how the INTR field of the buffer descriptor(s) are set. For for information on handling interrupts, refer to Interrupt Handling.

Software can use ZDMA_CH_IRQ_DST_ACCT and/or 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 of the ZDMA_CH_IRQ_DST_ACCT and/or 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. Refer to Channel Disabled for more information.

Interrupt Handling

Interrupt handling is done by following these steps.

1. Read the status from the ZDMA_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. In case any other error bit is set, set the channel as IDLE in the software context.
5. Clear the interrupt status from the ZDMA_CH_ISR register by writing back value read in step 1.

DMA Programming Model for FCI

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. 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).
- ZDMA_CH_{DATA, DSCR}_ATTR attribute 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, ZDMA_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 ZDMA_CH_FCI [EN] bit is set. The DMA channel clears ZDMA_CH_FCI [EN] once 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 your applications follows.

- 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

FCI 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.

DMA 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. Once 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.

DMA 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 you do not flow control the read/SRC when the FCI is configured to the flow control DST, the channel can issue multiple read transactions and utilize the entire common buffer. This will starve other channels. To resolve this issue, software configures the maximum number of entries used by the DMA channel. Once the DMA channel exceeds the programmed value, it will 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 do 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.

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 and how to go into a particular state.

- Disabled
- Enabled
- Paused

Channel Enabled

The software can enable one or more channels at any time using the following enable sequence.

- Setup channel mode (simple or scatter gather mode).
- Set the ZDMA_CH_{DATA, DSCR}_ATTR attribute registers.
- Setup DMA mode.
    - Simple mode, program the DSCR registers.
    - In scatter gather mode, program the DSCR in memory and program the DSCR start address register.
- 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 the following reasons.

1. 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.

2. DMA channel is in simple DMA mode and transfer is done.
    a. Once a channel is done transferring the data indicated into the SRC/DST DSCR register, the channel goes into a disable state.
    b. For subsequent transfers, the software must enable the channel.

3. Software can put any paused channel into a disable state.
    a. 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.
4. Any error detected on an AXI channel/descriptor programming puts the DMA channel into a disable state.

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. Once 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.

1. 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.

2. 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.

Security

The DMA allows software to mark each channel secure/non-secure by programming the LPD_SLCR_SECURE.slcr_adma and FPD_SLCR_SECURE.slcr_gdma registers for the RPU and APU DMA units, respectively. 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. They can access only non-secure regions.

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 Programing Error

The DMA assumes that software programs registers and descriptors as expected. In case of wrong software programming, a DMA channel does not take any action for error recovery and DMA channel behavior is unpredictable.

Total Transferred Bytes Overflow

Each DMA channel maintains a counter that provides status information about the number of bytes transferred to a destination. This counter is updated only when a DMA channel receives a BRESP write transaction. This counter always indicates the correct counter (even under an AXI error condition). This counter value is reflected on the APB register. The total transferred byte counter can be cleared by software by writing to the total transferred byte register.

In the case where the total transferred byte count overflows, a DMA channel indicates this as an interrupt. This is non-fatal error and does not affect the functionality of the DMA channel.

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.

Chapter 20

CAN Controller

Introduction

There are two CAN controllers in the PS. Each one is independently configured and controlled. Defining the CAN protocol is outside the scope of this document, and knowledge of the specifications is assumed.

Features

- Compatible with the ISO 11898-1, CAN 2.0A, and CAN 2.0B standards.
- Standard (11-bit identifier) and extended (29-bit identifier) frames.
- Transmit message FIFO (TXFIFO) with a depth of 64 messages.
- Transmit prioritization through one high-priority transmit buffer (TXHPB)
- Watermark interrupts for TXFIFO and RXFIFO.
- Automatic re-transmission on errors or arbitration loss in normal mode.
- Receive message FIFO (RXFIFO) with a depth of 64 messages.
- Four RX acceptance filters with enables, masks, and IDs.
- Loopback and snoop modes for diagnostic applications
- Sleep mode with automatic wake-up
- Maskable error and status interrupts
- 16-bit time stamping for receive messages
- Readable RX/TX error counters

Functional Description

The CAN controllers are controlled by the CAN0 and CAN1 registers sets (for example, the CAN0.MSR register). The system viewpoint of the CAN controller is shown in Figure 20-1.

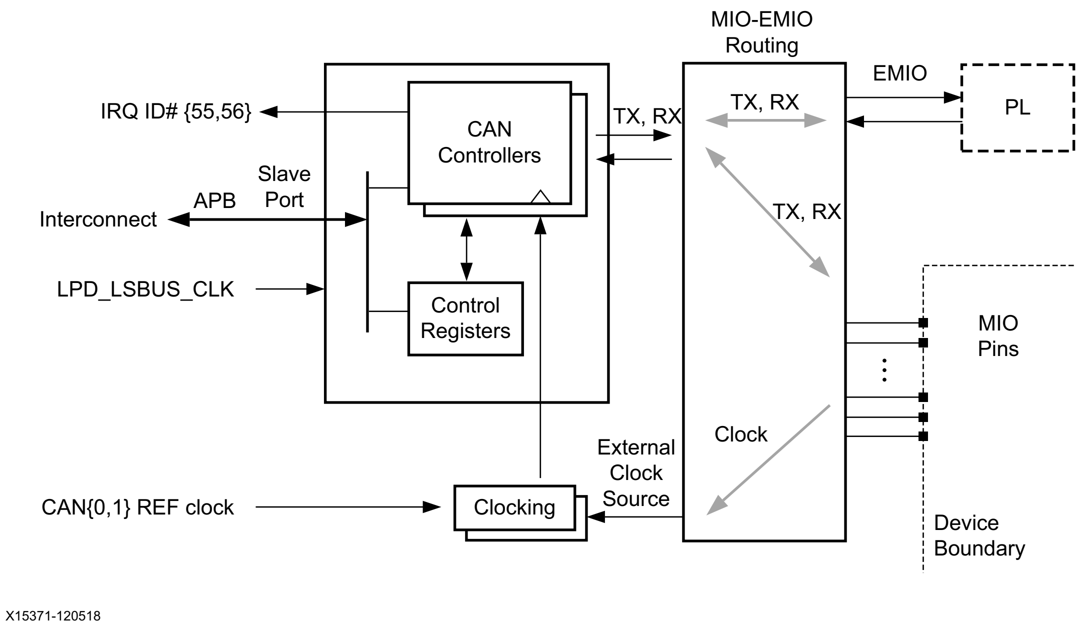


Figure 20-1: **CAN Controller System Viewpoint**

Block Diagram

The high-level architecture of the CAN controller is shown in Figure 20-2. The sub-modules are described in subsequent sections.

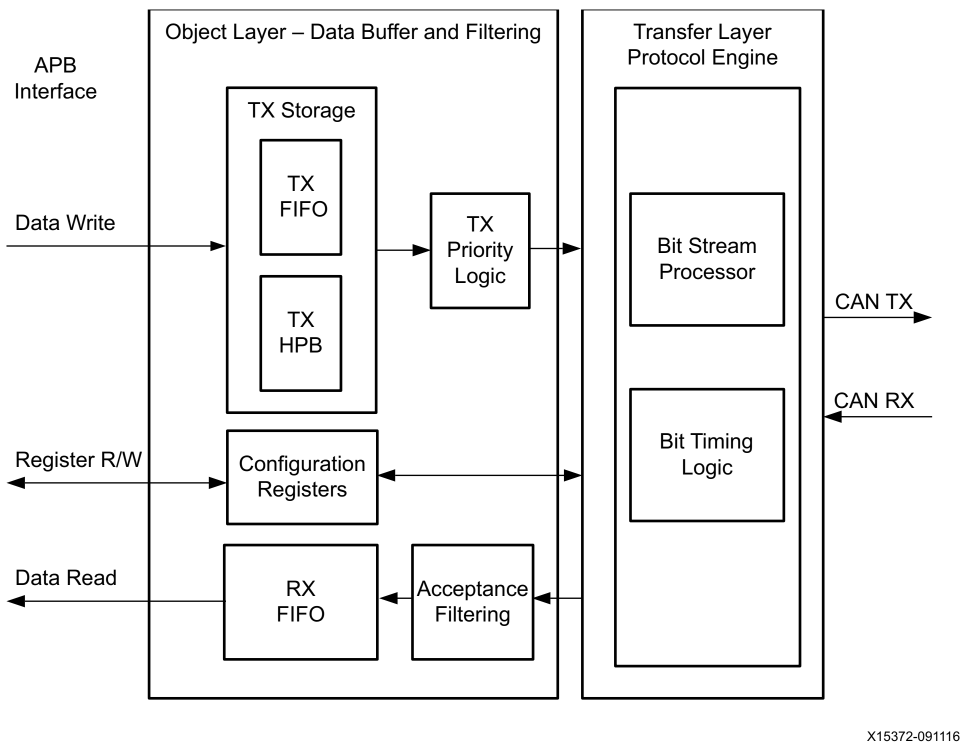


Figure 20-2: **CAN Controller Block Diagram**

Clocks

The CAN controller operates with two clocks: the bus LPD_LSBUS_CLK for register configuration and the CAN transceiver reference clock CAN{0, 1}_REF_CLK.

The LPD_LSBUS_CLK clock is used to clock all APB register logic using only the positive edge.

The CAN reference clock is used to generate an over-sampling clock based on the desired baud rate. The frequency of this clock depends on the required baud rate accuracy and must keep within the range of the baud-rate divider logic. It uses only the positive edge of this clock. The signal produced by the baud-rate division of this clock (the quantum clock) is used as an enable for bit-timing logic clocked by the CAN_REF_CLK, it is not used as a clock directly. The frequency of CAN_REF_CLK must be chosen such that a suitable accuracy is achieved for the required baud rate as specified in the ISO 11898-1, CAN 2.0A, and CAN 2.0B standards.

The CAN_REF_CLK is divided by a programmable baud-rate prescaler (BRPR) to generate the quantum clock as shown in Equation 20-1.

$$f_{quantum\_clk} = \frac{f_{CAN\_REF\_CLK}}{BRP + 1} \qquad \text{Equation 20-1}$$

Additionally, the number of quantum clocks within each phase of the CAN frame can be configured through the BTR register. The controller and I/O interface are driven by the reference clock (CAN_REF_CLK). The controller's interconnect also requires an APB interface clock. The APB interconnect clock always comes from the PS clock subsystem.

The reference clock normally comes from the PS clock subsystem, but it can alternatively be driven by an external clock source through any available MIO pin. The reference clock is used by the protocol engine, the baud-rate generator, and the datapath. The controllers share the same reference clock frequency from the PS clock subsystem. If the reference clock is from an MIO pin, then the frequencies can be different.

LPD_LSBUS_CLK Clock

The LPD_LSBUS_CLK clock runs asynchronous to the CAN reference clock.

Reference Clock

CAN_REF_CLK is normally sourced from the PS clock subsystem, but it can alternatively be driven by an external clock source through an MIO pin. Internally, the PS has three PLLs and two clock divider pairs. The clock source choice, PS clock subsystem or external MIO pin, is controlled by the IOU_SLCR.CAN_MIO_CTRL register.

The CAN reference clock frequencies are controlled by CRL_APB.CAN0_REF_CTRL and CRL_APB.CAN1_REF_CTRL registers.

Example: Configure and Route Internal Clock for Reference Clock

Configure the clock and disable MIO path. Assume the PLL is operating at 1000 MHz and the required CAN reference clock is 24 MHz (23.8095 MHz).

1. Program the clock subsystem. Write `0x0030_0E03` to the CRL_APB.CAN0_REF_CTRL register.
   a. Enable both CAN reference clocks.
   b. Divide the I/O PLL clock by 42 (`0x02A`): [DIVISOR0] = `0x0E` and [DIVISOR1] = `0x03` used by both controllers.
2. Disable the MIO path. Write `0x0000_0000` to the IOU_SLCR.CAN_MIO_CTRL register to select the clock from the internal clock subsystem/PLL for both controllers.

Programming Example – Assign MIO Pin as CAN Reference Clock Input

This example assigns MIO pin 45 to the clock input reference for the CAN controller. Configure MIO pin 45 for the external CAN reference clock. These steps refer to the IOU_SLCR register set.

1. Route the reference clock. Write `32h'0000_1200` to the MIO_PIN_45 register.
2. Disable the output driver. Write a 1 to the MIO_MST_TRI1 [PIN_45_TRI] bit.
3. Select CMOS input (not Schmitt). Write 0 to BANK1_CTRL3 [20] bit.
4. Select the internal pull-up resister. Write 1 to BANK1_CTRL4 [20] bit.
5. Enable the internal pull-up resister. Write 1 to BANK1_CTRL5 [20] bit.

The output controls, [drive0], [drive1], and [slow_fast_slew_n] do not need programming, but it is recommended to select minimum drive and slow slew rate. The voltage applied to PSIO bank 1 can be read using the BANK1_STATUS [0] bit.

Configure the reference clock at the controller. These steps refer to the IOU_SLCR.CAN_MIO_CTRL register.

6. Select the clock path to use MIO pin 45. Write 1 to the [CAN_REF_SEL] bit.
7. Select MIO Pin 45. Write `2Dh` (`45d`) to the [CAN_MUX] bit.
8. Choose an active reference clock edge for RX using the [CAN_RXIN_REG] bit.

Resets

There are two resets associated with the CAN. The effects for each reset type are summarized in Table 20-1.

Table 20-1: **CAN Reset Effects**

Name	APB Interface	RX and TX FIFOs	Protocol Engine	Control and Status Registers	Acceptance Filters (ID and Mask)
Local CAN reset can.SRR[SRST]	Yes	Yes	Yes	Yes	No
PS reset subsystem: CRL_APB.RST_LPD_IOU2 [CAN_RESET]	Yes	Yes	Yes	Yes	No

Example: Reset using Local CAN Reset

Write to the local CAN reset register. Write a `1` to can.SRR[SRST] bit field. This bit is self-clearing.

Example: Reset using Reset Subsystem

Write to the SLCR reset register for CAN. Write a `1` then a `0` to the CRL_APB.RST_LPD_IOU2 [CAN_RESET] bit field.

Configuration Registers

The CAN controller configuration register defines the configuration registers. This module allows for read and write access to the registers through the APB interface. An overview of the CAN controller registers is shown Register Overview.

Transmit and Receive Messages

A separate storage buffers exist for transmit (TXFIFO) and receive (RXFIFO) messages through a FIFO structure. Each buffer can store up to 64 messages. Once a message is written into the TXFIFO, the total delay to transmit it over the CAN bus is
2 x (TX driver delay + propagation delay + RX driver delay).

TX High Priority Buffer

Each controller also has a transfer high-priority buffer (TXHPB) that provides storage for one transmit message. Messages written on this buffer have maximum transmit priority. They are queued for transmission immediately after the current transmission is complete, preempting any message in the TXFIFO.

Acceptance Filters

Acceptance filters sort incoming messages with the user-defined acceptance mask and ID registers to determine whether to store messages in the RXFIFO, or to acknowledge and discard them. Messages passed through acceptance filters are stored in the RXFIFO.

Controller Modes

The CAN controller supports the following modes of operation.

- Configuration Mode
- Normal Mode
- Sleep Mode
- Loopback Mode (Diagnostics)
- Snoop Mode (Diagnostics)

Configuration Mode

The CAN controller enters the configuration mode when any of the following actions are performed, regardless of the operation mode.

- Writing a `0` to the [CEN] bit in the SRR register.
- Writing a `1` to the [SRST] bit in the SRR register. The controller enters the configuration mode immediately following the software reset.
- Driving a `0` on the reset input. The controller continues to be in reset as long as reset is `0`. The controller enters configuration mode after reset is negated to `1`.

Normal Mode

Normal mode transmits and receives messages on the TX and RX I/O signals as defined by the Bosch and IEEE specifications.

Sleep Mode

Sleep mode can be used to save a small amount of power during idle times. When in sleep mode, the controller can transition to normal mode or configuration mode. Sleep mode includes the following actions.

- When another node transmits a message, the controller receives the message and exits sleep mode.
- When there is a new TX request, the controller switches to normal mode and the services the request.
- An interrupt can be generated when the controller enters sleep mode.
- An interrupt can be generated when the controller wakes up.

Sleep mode is exited by the hardware when there is CAN bus activity or a request in either the TXFIFO or TXHPB. When the controller exits sleep mode, can.MSR[SLEEP] is set to `0` by the hardware and an interrupt can be generated.

The CAN controller enters sleep mode from configuration mode when the [LBACK] bit in the [MSR] is `0`, the [SLEEP] bit in the MSR register is `1`, and the [CEN] bit in the SRR register is `1`. The CAN controller enters sleep mode only when there are no pending transmission requests from either the TXFIFO or the TX high-priority buffer.

The CAN controller enters sleep mode from normal mode only when the [SLEEP] bit is `1`, the CAN bus is idle, and there are no pending transmission requests from either the TXFIFO or the TX high-priority buffer (TXHPB).

When another node transmits a message, the CAN controller receives the transmitted message and exits sleep mode. When the controller is in sleep mode, if there are new transmission requests from either the TXFIFO or the TXHPB, these requests are serviced, and the CAN controller exits sleep mode. Interrupts are generated when the CAN controller enters sleep mode or wakes up from sleep mode. From sleep mode, the CAN controller can enter either the configuration or normal modes.

Loopback Mode (Diagnostics)

Loopback mode is used for diagnostic purposes. When in loopback mode, the controller must only be programmed to enter configuration mode or issue a reset. In loop back mode, the following actions occur.

- The controller transmits a recessive bitstream onto the CAN_PHY_TX bus signal.
- TX messages are internally looped back to the RX line and are acknowledged.
- TX messages are not sent on the CAN_PHY_TX bus signal. The controller receives all the messages that it transmits.
- The controller does not receive any messages transmitted by other CAN nodes.

Snoop Mode (Diagnostics)

Snoop mode is used for diagnostic purposes. When in snoop mode, the controller must only be programmed to enter configuration mode or be held in reset. In snoop mode the following actions occur.

- The controller transmits a recessive bitstream onto the CAN bus.
- The controller does not participate in normal bus communication.
- The controller receives messages that are transmitted by other CAN nodes.
- 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`.

Mode Transitions

The supported mode transitions are shown in Figure 20-3. The transitions are primarily controlled by the resets, the [CEN] bit, the MSR register settings, and a hardware wake-up mechanism.

To enter normal mode from configuration mode the following steps must occur.

- Clear can.MSR[LBACK, SNOOP, SLEEP] = `0`.
- Set can.SRR[CEN] = `1`.

To enter sleep mode from normal mode (interrupt generated), set can.MSR[SLEEP] = `1`.

Events that cause the controller to exit sleep mode (interrupt generated) include the following steps.

- RX signal activity (hardware sets can.MSR[SLEEP] = 0).
- TXFIFO or TXHPB activity (hardware sets can.MSR[SLEEP] = 0).
- Software writes 0 to can.MSR[SLEEP].

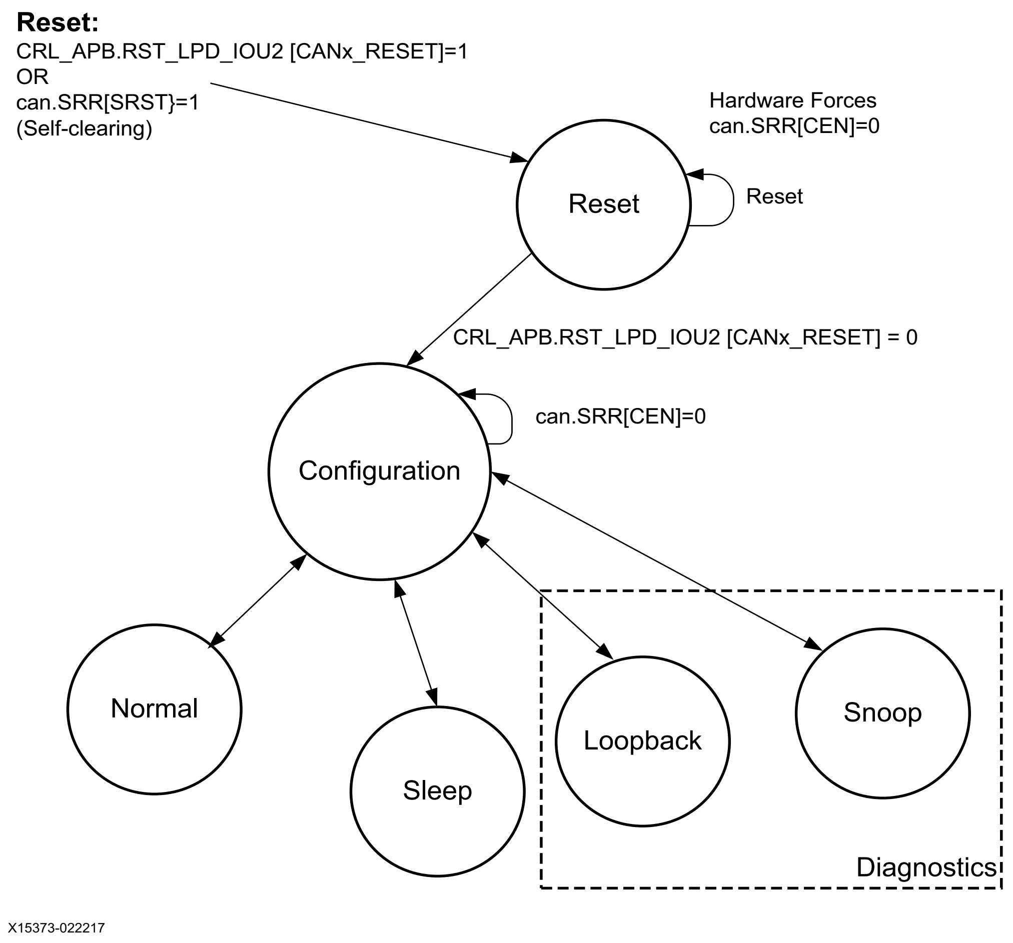


Figure 20-3: **CAN Operating Mode Transitions, Mode Settings**

Table 20-2 defines the CAN controller modes of operation and corresponding control and status bits.

Table 20-2: **CAN Controller Modes of Operation**

[CAN_RESET] bit	Software Reset Register (can.SRR)		Mode Select Register (MSR) (Read/Write bits)			Status Register (SR) (Read Only bits)					Operational Mode
	SRST (CAN Reset)	CEN (CAN Enable)	LBACK	SLEEP	SNOOP	CONFIG	LBACK	SLEEP	NORMAL	SNOOP	
1	X	X	X	X	X	1	0	0	0	0	Reset
0	1	X	X	X	X	1	0	0	0	0	Reset
0	0	0	X	X	X	1	0	0	0	0	Configuration
0	0	1	1	X	X	0	1	0	0	0	Loop back
0	0	1	0	1	0	0	0	1	0	0	Sleep
0	0	1	0	0	1	0	0	0	1	1	Snoop
0	0	1	0	0	0	0	0	0	1	0	Normal

Message Format

The same message format is used for RXFIFO, TXFIFO, and TXHPB. Each message includes four words (16 bytes). Software must read and write all four words regardless of the actual number of data bytes and valid fields in the message.

The message words, fields, and structure are shown in Table 20-3.

Table 20-3: **CAN Message Format**

Message Word Registers Description	Bits	Bit Field Name	Frame Types Data	Remote	Frame Size Standard	Extended	Default Value
Identifier:							
{RXFIFO, TXFIFO, TXHPB}_ID							
Remote transmission request	0	[RTR]	= 0	= 1	NA, = 0	Applies	0
Extended message frame ID	18:1	[IDL]	Valid	Valid	NA	Applies	0
Identifier extension for frame size	19	[IDE]	Valid	Valid	= 0	= 1	0
Substitute remote transmission request	20	[SRRRTR]	= 0	= 1	Applies	NA, = 1	0
Standard message frame ID	31:21	[IDH]	Valid	Valid	Applies	Applies	0
Data Length Code (DLC):							
{RXFIFO, TXFIFO, TXHPB}_DLC							
Data length code, 0 to 8 bytes	31:28	[DLC]	Valid	Valid	Valid	Valid	0
Reserved	27:0	~	~	~	~	~	0
Timestamp (RXFIFO only)	15:0	[RXT]					

Table 20-3: **CAN Message Format *(Cont'd)***

Message Word Registers Description	Bits	Bit Field Name	Frame Types Data	Remote	Frame Size Standard	Extended	Default Value
Data Word 1:							
{RXFIFO, TXFIFO, TXHPB}_DATA1							
Data Byte 0	31:24	[DB0]	Valid	Valid	Valid	Valid	0
Data Byte 1	23:16	[DB1]	Valid	Valid	Valid	Valid	0
Data Byte 2	15:8	[DB2]	Valid	Valid	Valid	Valid	0
Data Byte 3	7:0	[DB3]	Valid	Valid	Valid	Valid	0
Data Word 2:							
{RXFIFO, TXFIFO, TXHPB}_DATA2							
Data Byte 4	31:24	[DB4]	Valid	Valid	Valid	Valid	0
Data Byte 5	23:16	[DB5]	Valid	Valid	Valid	Valid	0
Data Byte 6	15:8	[DB6]	Valid	Valid	Valid	Valid	0
Data Byte 7	7:0	[DB7]	Valid	Valid	Valid	Valid	0

Bit Field Details

Writes

If a bit field or data byte is not required, then write zeros.

Reads

Data starts at byte 0 and continues for the number of counts in DLC.

Message Buffering

This section describes the RX and TX message buffers.

RX Messages

The RXFIFO can store up to 64 RX CAN messages that are received and optionally filtered. RX messages that pass any of the acceptance filters are stored in the RXFIFO. When no acceptance filter is selected, all received messages are stored in the RXFIFO. The software reads these messages as described in Read Messages from RXFIFO.

A timestamp is added to each successfully stored RX message. A free running 16-bit counter is clocked using CAN_REF_CLK. The rules for time stamping an RX message are as follows.

- The counter rolls over. No status bit indicates that a roll-over condition occurred. At certain bit rates, the choice of the reference clock frequency is constrained by the roll-over clock.
- The timestamp included when an RX message is successfully collected. The sampling of the counter takes place at the last bit of EOF.
- The counter is cleared when can.SSR [CEN] = 0 or when software writes a `1` to the can.TCR register.

Software must read all four registers of an RX message in the RXFIFO, regardless of how many data bytes are in the message. The first word is read using the RXFIFO_ID register and contains the received message standard and extended IDs, [IDH], and [IDL], respectively. The second word is read using the RXFIFO_DLC register and contains the 16-bit timestamp and data length code [DLC] field. The third and fourth words contain data word 1 (RXFIFO_DATA1) and data word 2 (RXFIFO_DATA2) registers.

Writes to the RXFIFO registers are ignored. Read data from an empty RXFIFO are invalid and might generate an interrupt.

The messages in the RXFIFO are retained even if the CAN controller enters the bus-off state or configuration mode.

TX Messages

The controller has a configurable TXFIFO that software can use buffer up to 64 TX CAN messages. The controller also has a high priority transmit buffer (TXHPB), with storage for one message. When a higher priority message needs to be sent, software writes the message to the high priority transmit buffer when it is available. The message in the TXHPB has higher priority over messages in the TXFIFO.

When arbitration loss or errors occur during the transmission of a message, the controller tries to retransmit the message. No subsequent message, even a newer, high-priority message is transmitted until the original message is transmitted without errors or arbitration loss.

The controller transmits the message starting with bit 31 of the IDR word. After the identifier word is transmitted, the TXFIFO_DLC word is transmitted. This is followed by the data bytes in this order: DB0, DB1, ... DB7. The MSB of a data byte is transmitted first.

The status bit, can.ISR[TXOK] is set = 1 after the controller successfully transmits a message from either the TXFIFO or TXHPB.

The messages in the TXFIFO and TXHPB are retained even if the CAN controller enters bus-off state or configuration mode.

The message format is described in Message Format.

Reads from RXFIFO

All 16 bytes must be read from the RXFIFO to receive the complete message.

- The first word read (4 bytes) returns the identifier of the received message RXFIFO_DLC register.
- The second read returns the 16-bit receive time stamp and data length code [DLC] field of the received message RXFIFO_DLC register.
- The third read returns data word 1 RXFIFO_DATA1 register.
- The fourth read returns data word 2 RXFIFO_DATA2 register.

A free running 16-bit counter provides a time stamp relative to the time the message was successfully received.

All four words must be read for each message, even if the message contains less than eight data bytes. Write transactions to the RXFIFO are ignored. Reads from an empty RXFIFO return invalid data and generates an RX underflow interrupt.

RX and TX Error Counters

When an RX or TX error occurs, the associated error counters in the protocol engine (see Protocol Engine) are incremented. The two error counters are 8 bits wide and are read using the read-only can.ECR register, bit fields [REC] and [TEC]. The RX and TX counters are reset when any the these situations occur.

- After a `1` is written to can.SRR[SRST] field = 1. This bit write is self-clearing.
- Anytime can.SRR[CEN] = 0 (configuration mode).
- When the controller enters the bus-off state.

Interrupts

Each CAN controller has a single interrupt signal to the generic interrupt controller (GIC). CAN 0 connects to IRQ ID#55 and CAN 1 connects to ID #56. The source of an interrupt can be grouped into one of the following.

- TXFIFO and TXHPB
- RXFIFO
- Message passing and arbitration
- Sleep mode and bus-off state

Enable and disable interrupts by using the can.IER register. Check the raw status of the interrupt using the can.ISR register. Clear interrupts by writing a `1` to the can.ICR register.

Some interrupt sources have an additional method to clear the interrupt as shown in Table 20-4.

List of Interrupts

All of the CAN interrupts are sticky. CAN status and interrupts are identified in Table 20-4. All bits are cleared by writing to the ICR register. Some bits can be cleared by writing a 0 to the can.SRR [CEN] bit field.

Table 20-4: **List of CAN Status and Interrupts**

Name	Bit Number	Additional Methods to Clear Interrupt	Usage
Arbitration lost	0	Write 0 to can.SRR[CEN]	Arbitration lost during message transmission
Message TX	1	Write 0 to can.SRR[CEN]	Message transmission successful
TXFIFO full	2	None	Read to determine if more TX messages can be written to the TXFIFO.
TXHPB full	3	None	Read to determine if more TX messages can be written to the TXHPB.
Message RX	4	Write 0 to can.SRR[CEN]	New message received in RXFIFO
RXFIFO underflow	5	None	Programming error, message read from RXFIFO when no messages were there.
RXFIFO overflow	6	Write 0 to can.SRR[CEN]	RX FIFO was full and RX message(s) likely lost.
RXFIFO not empty	7	None	One or more RX messages can be read.
Message error	8	Write 0 to can.SRR[CEN]	Any of the five errors in the error status register, ESR.
Bus-off state	9	Write 0 to can.SRR[CEN]	Bit is asserted when controller enters bus-off state
Enter sleep mode	10	Write 0 to can.SRR[CEN]	Bit is asserted when controller enters sleep state
Exit sleep mode	11	Write 0 to can.SRR[CEN]	Controller wakes up and enters normal or configuration mode.
RXFIFO watermark	12	None	Operational threshold indicates RXFIFO is above watermark setting.
TXFIFO watermark	13	None	Operational threshold indicates TXFIFO has more room than watermark setting.
TXFIFO empty	14	None	TXFIFO empty indicator.

RXFIFO and TXFIFO Interrupts

The FIFO watermark levels and all the FIFO interrupts are illustrated in Figure 20-4.

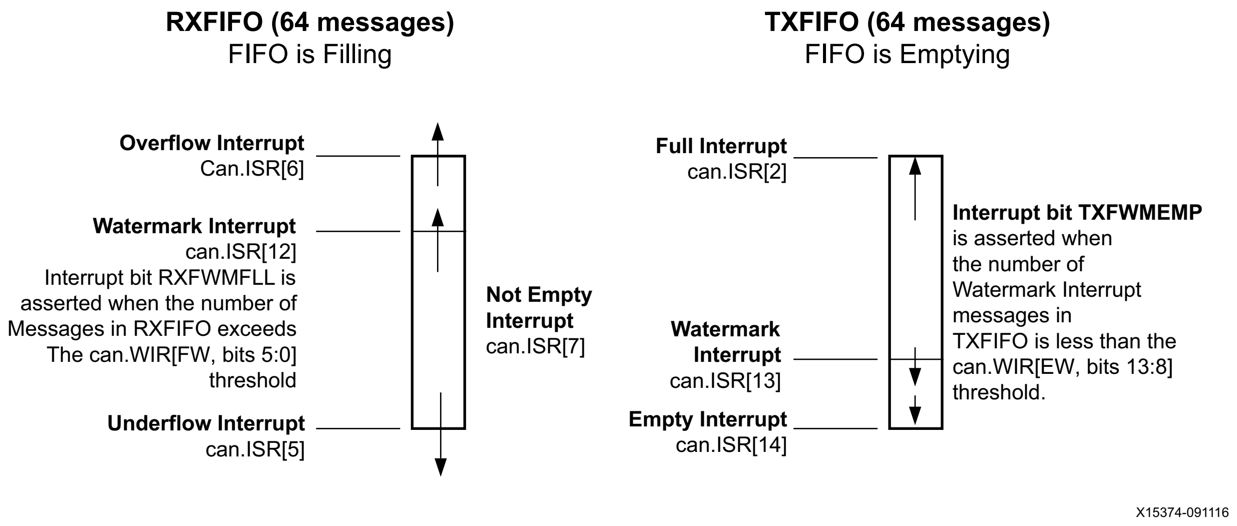


Figure 20-4: **CAN RXFIFO and TXFIFO Watermark Interrupts**

Example: Program RXFIFO Watermark Interrupt (12)

The following steps can be used to setup and control the RXFIFO watermark interrupt. See Figure 20-4. The watermark status and control interrupts are described in the Protocol Engine section.

1. Disable the RXFIFO watermark interrupt. Write a `0` to can.IER[12].
2. Program the RXFIFO full watermark level. Write to can.WIR[FW].
3. Clear the RXFIFO watermark interrupt. Write a `1` to can.ICR[12].
4. Read the RXFIFO watermark status. Read can.ISR[12].
5. Enable the RXFIFO watermark interrupt. Write a `1` to can.IER[12].

Example: Program TXFIFO Watermark Interrupt (13)

The following steps can be used to setup and control the TXFIFO watermark interrupt. See Figure 20-4. The watermark status and control interrupts are described in the Protocol Engine section.

1. Disable the TXFIFO watermark interrupt. Write a `0` to can.IER[13].
2. Program the TXFIFO empty watermark level. Write to can.WIR[EW].
3. Clear the TXFIFO watermark interrupt. Write a `1` to can.ICR[13].
4. Read the TXFIFO watermark status. Read can.ISR[13].
5. Enable the TXFIFO watermark interrupt. Write a `1` to can.IER[13].

Example: Program TXFIFO Empty Interrupt (14)

The following steps can be used to control the TXFIFO empty interrupt:

1. Disable the TXFIFO empty interrupt. Write a `1` to can.IER[14].
2. Clear the TXFIFO empty interrupt. Write a `1` to can.ICR[14].
3. Enable the TXFIFO empty interrupt. Write a `1` to can.IER[14].
4. Read the TXFIFO empty status. Read can.ISR[14]. It indicates the status (whether TXFIFO is empty).

RX Message Filtering

To filter RX messages, configure and enable up to four acceptance filters with acceptance mask, and ID registers to determine whether to store messages in the RXFIFO or to acknowledge and discard them.

Acceptance filtering is performed by the following 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, then the message is stored in the RXFIFO.
5. Acceptance filtering is processed by each of the defined filters. If the incoming identifier passes through any acceptance filter, then the message is stored in the RXFIFO.

Acceptance Filter Enable

The acceptance filter register (AFR) defines the acceptance filters usage. It includes four enable bits that correspond to the four acceptance filters. Each acceptance filter ID register (AFIR) and acceptance filter mask register (AFMR) pair is associated with a use acceptance filter (UAF) bit.

- When the UAF bit is `1`, the corresponding acceptance filter pair is used for acceptance filtering.
- When the UAF bit is `0`, the corresponding acceptance filter pair is not used for acceptance filtering.

To modify an acceptance filter pair in normal mode, the corresponding UAF bit in this register must first be set to `0`. After the acceptance filter is modified, the corresponding UAF bit must be set to `1` for the filter to be enabled.

The UAF bits in the can.AFR register enable the RX acceptance filters.

- If all UAF bits are set to `0`, then all received messages are stored in the RXFIFO.
- If the UAF bits are changed from a `1` to `0` during reception of a CAN message, the message will not be stored in the RXFIFO.

If any of the enabled filters (up to four) satisfy the following equation, then the RX message is stored in the RXFIFO.

If (AFMR and Message_ID) == (AFMR and AFIR) then capture message

Each acceptance filter is independently enabled. The filters are selected by the can.AFR register.

- Set can.AFR[UAF4] = `1` to enable AFMR4 and AFID4.
- Set can.AFR[UAF3] = `1` to enable AFMR3 and AFID3.
- Set can.AFR[UAF2] = `1` to enable AFMR2 and AFID2.
- Set can.AFR[UAF1] = `1` to enable AFMR1 and AFID1.

If all can.AFR[UAFx] bits are set to `0`, then all received messages are stored in the RXFIFO. The UAF bits are sampled by the controller at the start of an incoming message.

Acceptance Filter Mask Register

The acceptance filter mask registers (AFMR) contain mask bits used for acceptance filtering. The incoming message identifier portion of a message frame is compared with the message identifier stored in the acceptance filter ID register. The mask bits define the identifier bits that are stored in the acceptance filter ID register and are compared to the incoming message identifier.

There are four AFMRs. These registers are stored in a memory. Reads from AFMRs return Xs if the memory is not initialized. Asserting a software reset or hardware reset does not clear register contents. These registers can be read from and written to. These registers are written to only when the corresponding UAF bits in the can.AFR register are `0` and the [ACFBSY] bit in the can.SR register is `0`.

Acceptance Filter Identifier

The acceptance filter ID registers (AFIR) contain identifier bits, which are used for acceptance filtering. There are four read/write acceptance filter ID registers. These registers should only be written when the corresponding UAF bits in the SR register are `0` and the [ACFBSY] bit in the SR register is `0`.

TIP: *Proper programming of the TXFIFO_ID and RXFIF_ID [IDE] bits for standard and extended frames must be followed. Setting the [AIIDE] bit in AMIR register to `0` implies that there is only a standard frame ID check.*

Example: Program Acceptance Filter

Each acceptance filter has its own mask (can.AFMR{1,2,3,4}) and ID register (can.AFIR{1,2,3,4}).

1. Disable acceptance filters. Write a `0` to the can.AFR register.
2. Wait for the filter to not be busy. Poll the can.SR[ACFBSY] bit for a `0`.
3. Write a filter mask and ID. Write to a pair of AFMR and AFIR registers (see examples in Program the AFMR and AFIR Registers section).
4. Write additional filter masks and IDs. Go to step 2.
5. Enable one or more filters. To enable all filters, write `32'h0F` to the can.AFR register.

Program the AFMR and AFIR Registers

The valid AFMR and AFIR register bit fields for sending TX messages to the controller are summarized in Table 20-5. These fields are described in the Message Format section.

Table 20-5: **CAN Message Acceptance Mask and Identifier Register Bit Fields**

AFMR{1:4} Registers AFIR{1:4} Registers	[AMRTR] [AIRTR]	[AMIDL] [AIIDL]	[AMIDE] [AIIDE]	[AMSRR] [AISRR]	[AMIDH] [AIIDH]
Standard frame	Set = 0	Set = 0	Valid	Valid	Valid
Extended frame	Valid	Valid	Valid	Valid	Valid

In the AFMR mask registers, enable (unmask) the compare functions for each field for the incoming RX message by writing a `1` to the bit field. In the AFIR registers, write the values that are to be compared to the incoming TX message.

Example: Program the AFMR and AFIR for Standard Frames

This example sets up the acceptance filter for standard frames. The frame ID number is shown as `55Eh`, but could be set to a specific value based upon your application.

1. Configure the filter mask for standard frames. Write `FFF8_0000h` to the can.AFMR register.
   a. Enable the compare for the standard message ID, [AMIDE] = `1`.
   b. Compare all bits in the standard message ID, [AMIDH] = `7FFh`.
   c. Enable the compare for substitute remote transmission request, [AMSRR] = `1`.
   d. Zero-out the extended frame bits, [AMIDL, AMRTR] = `0`.
2. Configure the filter ID for standard frames. Write `ABC0_0000h` to the can.AFIR register.
   a. Select the standard frame message mode, [AIIDE] = `0`.
   b. Program the standard message ID, [AIIDH] = `55Eh`.
   c. Disable substitute remote transmission request, [AISRR] = `0`.
   d. Zero-out extended frame bits, [AIIDL, AIRTR] = `0`.

Example: Program the AFMR and AFIR for Extended Frames

This example setups up the acceptance filter for extended frames. The frame ID number is shown as `55Eh`, but could be set to a specific value based upon your application.

1. Configure the filter mask for extended frames. Write `FFFF_FFFFh` to the can.AFMR register.
    a. Enable the substitute remote transmission request mask for frame, [AMSRR] = `1`.
    b. Compare all bits in the compare for the standard message ID, [AMIDH] = `7FFh`.
    c. Enable the extended frame, [AIIDE] = `1`.
    d. Extended ID, [AIIDL] = `3_FFFFh`.
    e. Remote transmission request bit for extended frame, [AIRTR] = `1`.
2. Configure the filter ID for extended frames. Write `ABDF_9BDEh` to the can.AFIR register.
    a. Standard ID, [AIIDH] = `55Eh`.
    b. Remote transmission request bit for standard frame, [AISRR] = `1`.
    c. Select standard/extended frame, [AIIDE] = `1`.
    d. Extended ID, [AIIDL] = `3_CDEFh`.
    e. Remote transmission request bit for extended frame, [AIRTR] = `0`.

Protocol Engine

The CAN protocol engine consists primarily of the bit timing logic (BTL) and the bitstream processor (BSP) modules. Figure 20-5 shows a block diagram of the CAN protocol engine.

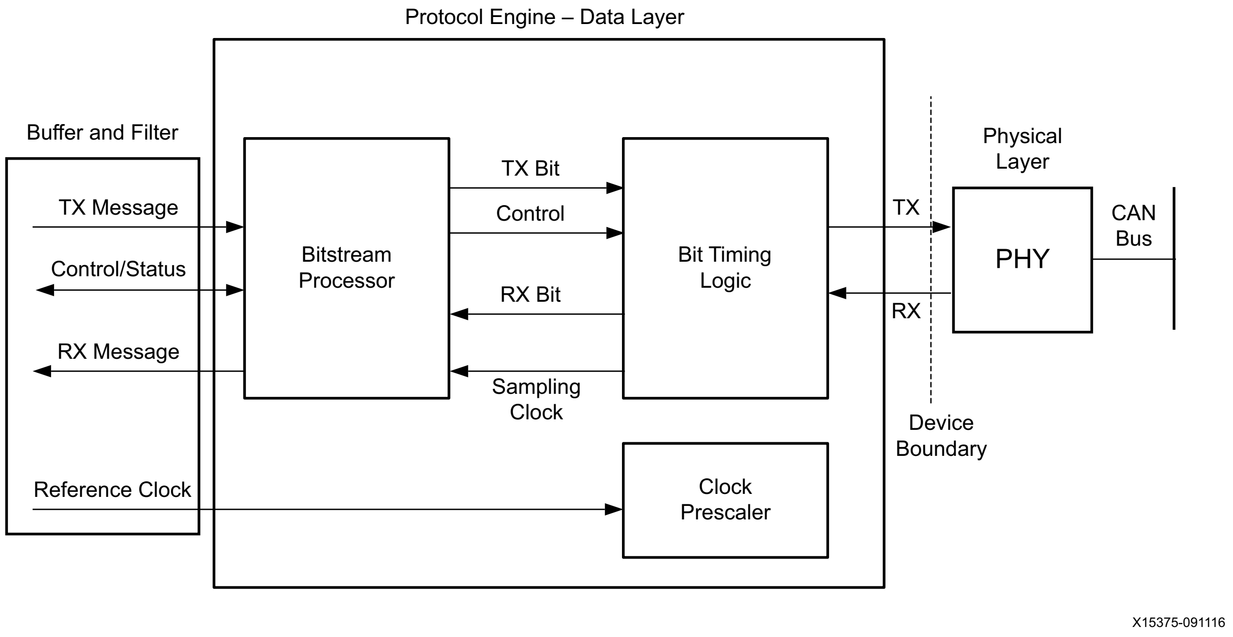


Figure 20-5: **CAN Controller Protocol Engine**

RX/TX Bit Timing Logic

The primary functions of the bit timing logic (BTL) module include the following.

- Generate the RX sampling clock for the bitstream processor (BSP).
- Synchronize the CAN controller to CAN traffic on the bus.
- Sample the bus and extracting the data stream from the bus during reception.
- Insert the transmit bitstream onto the bus during transmission.

The nominal length of the bit time clock period is based on the CAN_REF_CLK clock frequency, the baud rate generator divider (can.BRPR register), and the segment lengths (can.BTR register).

The bit timing logic module manages the re-synchronization function for CAN using the sync width parameter in the can.BTR[SJW] bit field. The CAN bit timing is shown in Figure 20-6.

The sync segment count always equals one time quanta period. The TS 1 and TS 2 period counts are programmable using the can.BTR[TS1, TS2] bit fields. These registers are written when the controller is in configuration mode. The width of the propagation segment (PROP_SEG) must be less than the actual propagation delay.

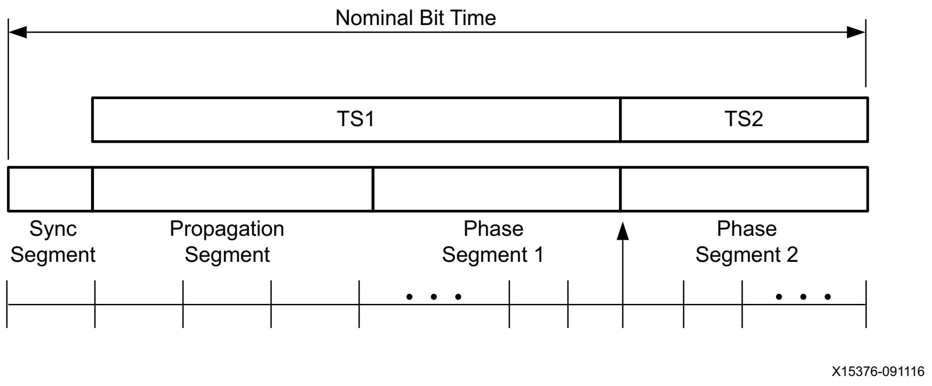


Figure 20-6: **CAN Bit Time**

Time Quanta Clock

The time quanta clock (TQ_CLK) is derived from the controller reference clock (CAN_REF_CLK) divided by the baud rate prescaler (BRP).

$t_{TQ\_CLK} = t_{CAN\_REF\_CLK} * (can.BRPR[BRP] + 1)\ freqT_{Q\_CLK}$
$= freq_{CAN\_REF\_CLK} / (can.BRPR[BRP] + 1)$

$t_{SYNC\_SEGMENT} = 1 * t_{TQ\_CLK}\ t_{TIME\_SEGMENT1}$
$= t_{TQ\_CLK} * (can.BPR[TS1] + 1)\ t_{TIME\_SEGMENT2}$
$= t_{TQ\_CLK} * (can.BPR[TS2] + 1)\ t_{BIT\_RATE}$
$= t_{SYNC\_SEGMENT} + t_{TIME\_SEGMENT1} + t_{TIME\_SEGMENT2}\ freq_{BIT\_RATE}$
$= freq_{CAN\_REF\_CLK} / ((can.BRPR[BRP] + 1) * (3 + can.BTR[TS1] + can.BTR[TS2]))$

TIP: *A given bit-rate can be achieved with several bit-time configurations, but values should be selected after careful consideration of oscillator tolerances and CAN propagation delays. For more information on CAN bit-time register settings, refer to the CAN 2.0A, CAN 2.0B, and ISO 11898-1 specifications.*

Bitstream Processor

The bitstream processor (BSP) module performs several functions while sending and receiving CAN messages. The BSP obtains a message for transmission from either the TXFIFO or the TXHPB and performs the following functions before passing the bitstream to the BTL.

- Serializing the message.
- Inserting Stuff bits, CRC bits, and other protocol defined fields during transmission.

During transmission the BSP simultaneously monitors RX data and performs bus arbitration tasks. It then transmits the complete frame when arbitration is won, and retrying when arbitration is lost.

During reception the BSP removes Stuff bits, CRC bits, and other protocol fields from the received bitstream. The BSP state machine also analyzes bus traffic during transmission and reception for form, CRC, ACK, Stuff, and bit violations. The state machine then performs error signaling and error confinement tasks. The CAN controller does not voluntarily generate overload frames but does respond to overload flags detected on the bus.

This module determines the error state of the CAN controller: error active, error passive, or bus-off. When TX or RX errors are observed on the bus, the BSP updates the transmit and receive error counters according to the rules defined in the CAN 2.0A, CAN 2.0B, and ISO 11898-1 standards. Based on the values of these counters, the error state of the CAN controller is updated by the BSP.

CAN0-to-CAN1 Connection

The I/O signals of the two CAN controllers in the PS can be connected together. In this mode, the RX signal of one CAN controller is connected to the TX signal of the other controller. These connections are enabled using the IOU_SLCR.MIO_LOOPBACK [CAN0_LOOP_CAN1] bit.

I/O Interface

This section discusses the MIO and EMIO signals.

MIO Programming

Each set of controller RX/TX signals is connected to either MIO pins or the EMIO interface.

Programming Example – Assign MIO Pin to CAN RX Input

This example assigns MIO pin 46 to the CAN RX controller signal. These steps refer to the IOU_SLCR register set.

1. Route the reference clock. Write `32h'0000_1221` to the MIO_PIN_46 register.
2. Disable output driver. Write a 1 to the MIO_MST_TRI1 [PIN_46_TRI] bit.
3. Select CMOS input (not Schmitt). Write 0 to BANK1_CTRL3 [21] bit.
4. Select the internal pull-up resister. Write 1 to BANK1_CTRL4 [21] bit.
5. Enable the internal pull-up resister. Write 1 to BANK1_CTRL5 [21] bit.

The I/O buffer output controls, [drive0], [drive1], and [slow_fast_slew_n] do not need programming, but it is recommended to select minimum drive and slow slew rate. The voltage applied to PSIO bank 1 can be read using the BANK1_STATUS [0] bit.

Programming Example – Assign MIO Pin to CAN TX Output

This example assigns MIO pin 47 to the CAN TX controller signal. These steps refer to the IOU_SLCR register set.

1. Route the reference clock. Write `32h'0000_1220` to the MIO_PIN_47 register.
2. Enable output driver. Write a 0 to the MIO_MST_TRI1 [PIN_47_TRI] bit.
3. Select slow slew rate output. Write 1 to the BANK1_CTRL6 [22] bit.
4. Choose an output drive strength. Write to the BANK1_CTRL0 [22] and BANK1_CTRL1 [22] bits.

The I/O buffer input control, [schmit_cmos_n], does not need programming, but it is recommended to select CMOS. The voltage applied to PSIO bank 1 can be read using the BANK1_STATUS [0] bit.

MIO-EMIO Signals

The CAN I/O signals are identified in Table 20-6.

Table 20-6: **CAN MIO Pins and EMIO Signals**

CAN Interface	Default Controller Input Value	MIO Pins		EMIO Signals	
		Numbers	I/O	Name	I/O
CAN 0 RX	0	2, 6, 10, 14, 18, 22, 26, 30, 34, 38, 42, 46, 50, 54, 58, 62, 66, 70, 74	I	EMIOCAN0PHYRX	I
CAN 0 TX	–	3, 7, 11, 15, 19, 23, 27, 31, 35, 39, 43, 47, 51, 55, 59, 63, 67, 71, 75	O	EMIOCAN0PHYTX	O
CAN 0 CLK	–	Any MIO pin	I	–	–
CAN 1 RX	0	1, 5, 9, 13, 17, 21, 25, 29, 33, 37, 41, 45, 49, 53, 57, 61, 65, 69, 73, 77	I	EMIOCAN1PHYRX	I
CAN 1 TX	–	0, 4, 8, 12, 16, 20, 24, 28, 32, 36, 40, 44, 48, 52, 56, 60, 64, 68, 72, 76	O	EMIOCAN1PHYTX	O
CAN 1 CLK	–	Any MIO pin	I	–	–

Register Overview

The control and status registers are listed in Table 20-7. Each of these registers is 32-bits wide. Any read operations to reserved bits or bits that are not used return a `0`. Write a `0` to reserved bits and unused bit fields. Writes to reserved locations are ignored.

Table 20-7: **CAN Register Overview**

Type	Register Names (CAN Registers, except where noted)	Description
Configuration and control	SRR, MSR, BRPR, BTR, ECR, TCR	Enable/disable and reset the controller. Setup baud rate and timing. Clear timestamp counter.
Interrupt processing	ISR, IER, ICR, WIR	Enable/disable the interrupt detection, mark interrupt sent to the interrupt controller, read raw interrupt status.
Status	ECR, ESR, SR	Inform about the status of the controller.
Transmit FIFO	TXFIFO_ID, TXFIFO_DLC, TXFIFO_DATA1, TXFIFO_DATA2	Write message to be transmitted.
Transmit high-priority buffer	TXHPB_ID, TXHPB_DLC, TXHPB_DATA1, TXHPB_DATA2	Store one high priority transmit message.
Receive FIFO	RXFIFO_ID, RXFIFO_DLC, RXFIFO_DATA1, RXFIFO_DATA2	Read received message.
Acceptance filter	AFR, AFMR[4:1], AFIR[4:1]	Configure and control the four acceptance filters.

Programming Model

Flowchart

Figure 20-7 shows the programming flowchart. Table 20-8 through Table 20-10 list the CAN controller modes.

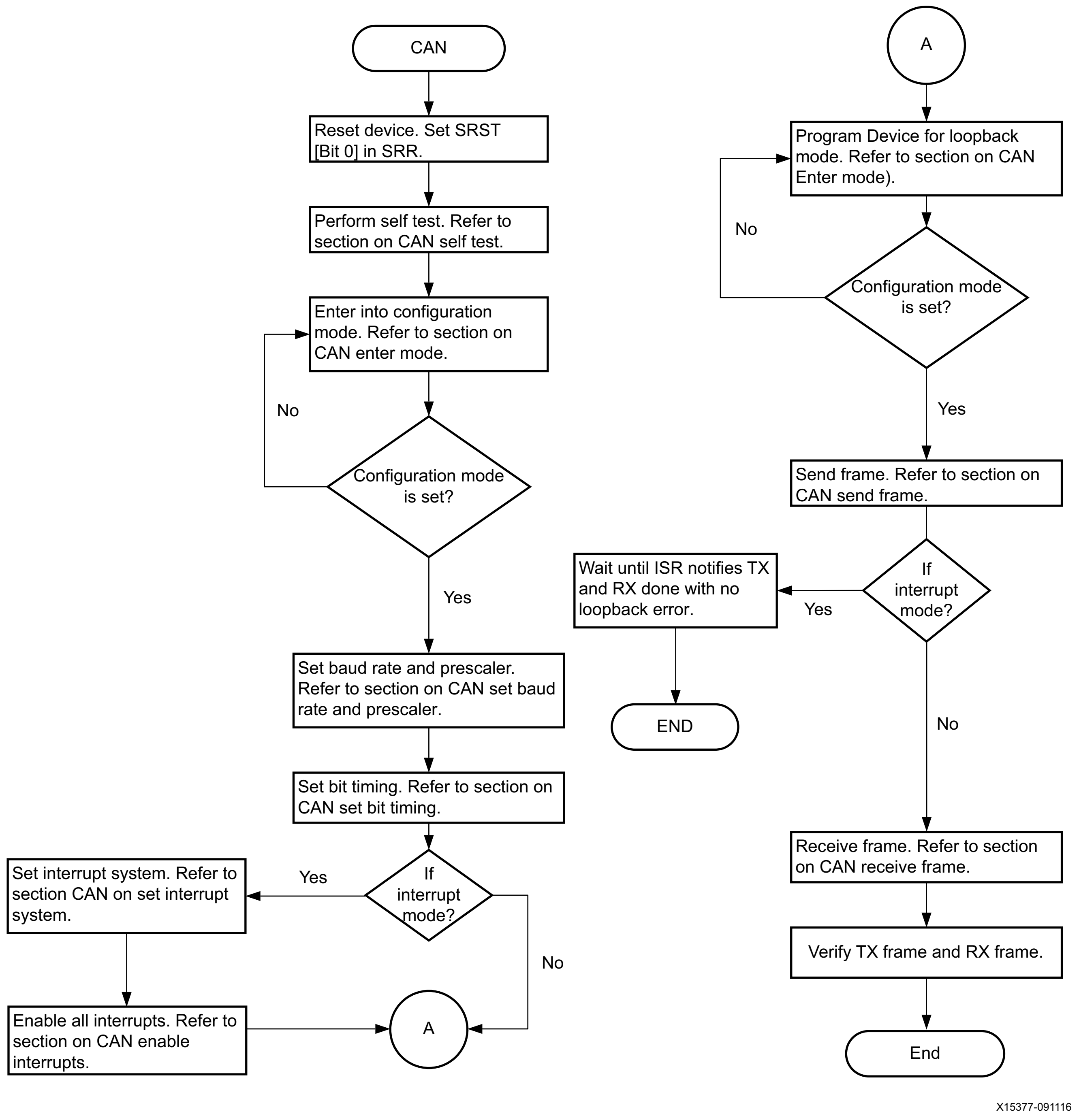

CAN
Reset device. Set SRST [Bit 0] in SRR.
Perform self test. Refer to section on CAN self test.
Enter into configuration mode. Refer to section on CAN enter mode.
No
Configuration mode is set?
Yes
Set baud rate and prescaler. Refer to section on CAN set baud rate and prescaler.
Set bit timing. Refer to section on CAN set bit timing.
Yes
If interrupt mode?
No
Set interrupt system. Refer to section CAN on set interrupt system.
Enable all interrupts. Refer to section on CAN enable interrupts.
A
A
Program Device for loopback mode. Refer to section on CAN Enter mode).
No
Configuration mode is set?
Yes
Send frame. Refer to section on CAN send frame.
If interrupt mode?
Yes
Wait until ISR notifies TX and RX done with no loopback error.
END
No
Receive frame. Refer to section on CAN receive frame.
Verify TX frame and RX frame.
End
X15377-091116

Figure 20-7: **CAN Controller Flowchart**

Table 20-8: **CAN Get Mode**

Task	Register	Register Field	Register Offset	Bits	Value (Binary)
If CONFIG (bit 0) bit is set, the device is in configuration mode.					
If NORMAL (bit 3) bit is set, then if the SNOOP (bit 12) bit is set, then the device is in snoop mode, else device is in normal mode.					
When none of these (CONFIG, NORMAL, or SNOOP) bits are set, device is in loopback mode.					
Read status register	SR	All	`0x18`	31:0	Read operation

Notes:

1. If CONFIG (bit 0) bit is set, device is in configuration mode.
2. If NORMAL bit (bit 3) is set, then if SNOOP bit (bit 12) is set, then device is in snoop mode, else device is in normal mode.
3. When none of above bits are set, device is in loopback mode.

Table 20-9: **CAN Set Baud Rate and Prescaler**

Task	Register	Register Field	Register Offset	Bits	Value (Binary)
Get the current mode of the device to confirm that the device is in configure mode. Refer to the CAN Get Mode.					
Program baud rate value.	BRPR	BRP	`0x08`	7:0	`7b'00101001`

Table 20-10: **CAN Set Bit Timing**

Task	Register	Register Field	Register Offset	Bits	Value (Binary)
Get the current mode of the device to confirm the device is in configure mode. Refer to the CAN Get Mode.					
Program the baud rate value.	BTR	SJW | TS2 | TS1	`0x0C`	8:0	`8b110101111`

Table 20-11: **CAN Enter Mode**

Task	Register	Register Field	Register Offset	Bits	Value (Binary)
To get the current mode of to device. Refer to the CAN Get Mode.					
If current mode is normal mode, and requested mode is sleep mode, then follow these normal mode to sleep mode steps.					
Select sleep mode and return	MSR	SLEEP	`0x04`	0	`1b'1`
If current mode is sleep mode, and requested mode is normal mode, then follow these sleep mode too normal mode steps.					
Select normal mode and return.	MSR	SLEEP | LBACK | SNOOP	`0x04`	2:0	`3b'000`
If the mode transition is not any of the two cases above, CAN must enter configuration mode before switching into the target operation mode.					
Set configuration mode	SRR	CEN | SRST	`0x00`	1:0	`2b'00`
Check the device mode. Refer to the CAN Get Mode. If the device is not entered into configuration mode, return. If entered, then follow these steps to set the requested mode.					
To set sleep mode					
Select sleep mode.	MSR	SLEEP	`0x04`	0	`1b'1`
Enable CAN.	SRR	CEN	`0x00`	1	`1b'1`

Table 20-11: **CAN Enter Mode *(Cont'd)***

Task	Register	Register Field	Register Offset	Bits	Value (Binary)
To set normal mode					
Select normal mode.	MSR	SLEEP | LBACK | SNOOP	`0x04`	2:0	`3b'000`
Enable CAN	SRR	CEN	`0x00`	1	`1b'1`
To set loopback mode					
Select sleep mode.	MSR	LBACK	`0x04`	1	`1b'1`
Enable CAN	SRR	CEN	`0x00`	1	`1b'1`
To set snoop mode					
Select snoop mode.	MSR	SNOOP	`0x04`	2	`1b'1`
Enable CAN	SRR	CEN	`0x00`	1	`1b'1`

Table 20-12: **Check CAN FIFO is Full**

Task	Register	Register Field	Register Offset	Bits	Value (Binary)
Read status register.	SR	TXFLL	`0X18`	10	READ operation
If TXFLL is set, then the FIFO is full. Else, the FIFO is not full.					

Table 20-13: **CAN Frame Send**

Task	Register	Register Field	Register Offset	Bits	Value (Binary)
Check if the CAN FIFO is full to make sure there is room in the FIFO. Refer to Check CAN FIFO is Full.					
Program TXFIFO_ID.	TXFIFO_ID	IDH | SRRRTR | IDE | IDL | RTR	`0x30`	31:0	`0x20000000 (hex)`
Program TXFIFO_DLC.	TXFIFO_DLC	DLC	`0x34`	31:28	`4b'1000`
Program TXFIFO_DATA1.	TXFIFO_DATA1	DB0 | DB1 | DB2 | DB3	`0x38`	31:0	Data
Program TXFIFO_DATA2.	TXFIFO_DATA2	DB4 | DB5| DB6 | DB7	`0x3c`	31:0	Data

Table 20-14: **CAN Check RX Empty**

Task	Register	Register Field	Register Offset	Bits	Value (Binary)
Read ISR.	ISR	RXNEMP	`0x1C`	7	Read operation
If the RXNEMP bit is set, then the RX is not empty. Else, the RX FIFO is empty.					

Table 20-15: **CAN Receive Frame**

Task	Register	Register Field	Register Offset	Bits	Value (Binary)
Check RX empty to make sure data is present (refer CAN Check RX empty)					
Read RXFIFO_ID.	RXFIFO_ID	IDH | SRRRTR | IDE | IDL | RTR	`0x50`	31:0	Read
Read RXFIFO_DLC.	RXFIFO_DLC	DLC | RXT	`0x54`	31:0	Read
Read RXFIFO_DATA1.	RXFIFO_DATA1	DB0 | DB1 | DB2 | DB3	`0x58`	31:0	Read
Read RXFIFO_DATA2.	RXFIFO_DATA2	DB4 | DB5| DB6 | DB7	`0x5C`	31:0	Read
Clear RXNEMP bit.	ISR	RXNEMP	`0x1C`	7	`1'b0`

Table 20-16: **CAN Setup Interrupt System**

Task	Register	Register Field	Register Offset	Bits	Value (Binary)
Initialize GIC. Refer to the GIC section.					
Register GIC interrupt handler. Refer to the GIC section.					
Register CAN interrupt handler with the GIC.					
Enable GIC. Refer to the GIC section.					
Enable processor interrupts.					

Table 20-17: **CAN Interrupt Handler**

Task	Register	Register Field	Register Offset	Bits	Value (Binary)
Read ISR (status).	ISR	All	`0x1C`	14:0	Read
Get enabled interrupts list (pendingintr and status).	IER	All	`0x20`	14:0	Read
Clear all interrupts.	ICR	All	`0x24`	14:0	pendingintr
If error interrupt is set (bit CERROR), notify application the error interrupt has been set.					
Read error status (esr_status).	ESR	`0x14`	All	4:0	Read
Clear error status.	ESR	`0x14`	esr_status	4:0	esr_status
If the bus off interrupt is set (BSOFF bit), return from interrupt.					
If water mark full OR RXNEMP interrupts set, receive frame. Refer to CAN Receive Frame.					
If TXOK interrupt is set notify application that TX is ok.					

Programming Guide Overview

The controller has several operating modes and different ways to receive and transmit messages. The low-level functions were described in Functional Description. The system-level operations are described in Clocks. All the controller registers are listed in Table 20-7. Further details are in the *Zynq UltraScale+ MPSoC Register Reference* (UG1087) [Ref 4].

Configuration Mode State

The CAN controller enters configuration mode, irrespective of the operation mode, when any of the following actions are performed.

- Writing a `0` to the CEN bit in the SRR register.
- Writing a `1` to the SRST bit in the SRR register. The controller enters configuration mode immediately following the software reset.
- Driving a `1` on the reset input controlled through the SLCR. The controller continues to be in reset as long as reset = `1`. The controller enters configuration mode after reset is negated to `0`.

In configuration mode the following apply.

- The CAN controller loses synchronization with the CAN bus and drives a constant recessive bit on the bus line.
- The error count register (ECR) is reset.
- The error status register (ESR) is reset.
- The bit timing register (BTR) and baud-rate prescaler register (BRPR) can be modified.
- The CAN controller does not receive any new messages.
- The CAN controller does not transmit any messages. Messages in the TXFIFO and the TXHPB are appended. These packets are sent when normal operation is resumed.
- Reads from the RXFIFO can be performed.
- Writes to the TXFIFO and TXHPB can be performed (provided the snoop bit is not set).
- Interrupt status register bits ARBLST, TXOK, RXOK, RXOFLW, ERROR, BSOFF, SLP, and WKUP can be cleared.
- Interrupt status register bits RXNEMP and RXUFLW can be set due to RXFIFO read operations.
- Interrupt status register bits TXBFLL and TXFLL and status register bits TXBFLL and TXFLL can be set due to write operations to the TXHPB and TXFIFO, respectively.
- Interrupts are generated if the corresponding bits in the interrupt enable register (IER) are `1`.
- All configuration registers are accessible.

When in configuration mode, the CAN controller stays in this mode until the CEN bit in the SRR register is set to `1`. After the CEN bit is set to `1`, the CAN controller waits for a sequence of `11` recessive bits before exiting configuration mode.

The CAN controller enters normal, loopback, snoop, or sleep modes from configuration mode, depending on the LBACK, snoop, and sleep bits in the MSR register.

Start-up Controller

The controller can operate in normal, sleep, snoop and loopback modes. Refer to Figure 20-3 for supported transitions. The controller clocks and configuration bits are programmed on startup. The operating mode is then selected and enabled.

Example: Start-up Sequence

1. Configure clocks. Refer to the Clocks section.
2. Configure TX/RX signals. Refer to the MIO Programming section.
3. Wait for configuration mode. Read can.SR[CONFIG] until it equals `1`.
4. Reset the controller. The controller comes up in configuration mode. Refer to the Resets section.
5. Program the bit-sampling clock. Refer to the RX/TX Bit Timing Logic section.
6. Program the interrupts, as needed. Refer to the Interrupts section.
7. Program the acceptance filters. Refer to the RX Message Filtering section.
8. Select operating mode. Normal, sleep, snoop, or loopback. Refer to the Change Operating Mode section.
9. Enable the controller. Write a `1` to can.SRR[CEN].

Change Operating Mode

This section contains programming examples to change the operating mode.

Example: Normal to Sleep Mode

Sleep mode is entered from the normal mode when the following conditions are met.

1. Select sleep mode. Write a `1` to can.MSR[SLEEP].
2. Wait for the CAN bus to go idle.
3. Wait for all the TXFIFO and TXHPB messages to be transmitted.

In normal mode, can.MSR[LBACK] = 0 and can.SSR[CEN] = 1. Also, can.MSR[SNOOP] = don't care.

Example: Configuration to Sleep Mode

Sleep mode is entered from the configuration mode when the following conditions are met.

1. Select sleep mode. Write a `1` to can.MSR[SLEEP] and write a `0` to can.MSR[LBACK].
2. Enable the controller. Write a `1` to can.SSR[CEN].
3. Wait for the TXFIFO or TXHPB to empty.

In configuration mode, can.MSR[SNOOP] = don't care.

Sleep mode is exited when I/O bus activity is detected or when software writes a message to either the TXFIFO or the TXHPB. When the controller exits sleep mode, can.MSR[SLEEP] is set to `0` and an interrupt is generated by the controller.

Write Messages to TXFIFO

With either option, can.SR[TXFLL] can be polled before writing a message.

All messages written to the TXFIFO should follow the format defined in Message Format.

Example: Write Message to TXFIFO Using Polling Method

1. Poll the TXFIFO status. Read can.SR[TXFLL] for a `0` and can.SR[TXFEMP] for a `1`, and then write the message into the TXFIFO.
2. Write the message to TXFIFO. Write to all four data registers (can.TXFIFO_ID, can.TXFIFO_DLC, can.TXFIFO_DATA1, and can.TXFIFO_DATA2).

Example: Write Message to TXFIFO Using Interrupt Method

In interrupt mode, writes can continue until can.ISR[TXFLL] generates an interrupt.

Messages can be continuously written to the TXFIFO until the TXFIFO is full. When the TXFIFO is full, the can.ISR[TXFLL] and can.SR[TXFLL] are set to `1`. When the TXFIFO is empty, can.ISR[TXFEMP] is set to `1`.

Write Messages to TXHPB

All messages written to the TXHPB use the polling method. The format should follow the Message Format section.

Example: Write Message to TXHPB

1. Poll the TXHPB status. Read can.SR[TXBFLL] until it equals `0` and then write the message into the TXHPB.
2. Write the message to TXHPB. Write to all four data registers (can.TXHPB_ID, can.TXHPB_DLC, can.TXHPB_DATA1, and can.TXHPB_DATA2).

Read Messages from RXFIFO

Whenever a new message is received and put into the RXFIFO, the can.ISR[RXNEMP] and can.ISR[RXOK] bits are set to `1`. If the RXFIFO is empty when the message is read, then the can.ISR[RXUFLW] is also set to `1`.

Example: Read Message from RXFIFO Using Polling Method

1. Poll the RXFIFO status. Read the can.ISR[RXOK] or can.ISR[RXNEMP] register until a message is received. Proceed to step 2 when a bit is set.
2. Read the message from the RXFIFO. Read all four of the registers (can.RXFIFO_ID, can.RXFIFO_DLC, can.RXFIFO_DATA1, and can.RXFIFO_DATA2).
3. Determine if more messages are in the RXFIFO. Read can.ISR[RXNEMP].

Example: Read Message from RXFIFO Using Interrupt Method

The can.ISR[RXOK] and/or can.ISR[RXNEMP] bit fields can generate the interrupt.

1. Program the RXFIFO watermark level interrupt. Write to can.WIR[FW] to set the watermark can.ISR[RXFWMFLL] interrupt.
2. Proceed to step 3 when an interrupt is received.
3. Wait until a message is received. Read can.ISR[RXOK] or can.ISR[RXFWMFLL].
4. Read the message from the RXFIFO. Read all four of the registers (can.RXFIFO_ID, can.RXFIFO_DLC, can.RXFIFO_DATA1, and can.RXFIFO_DATA2).
5. Determine if RXFIFO is not empty. Read can.ISR[RXNEMP].
6. Repeat until the RXFIFO is empty.
7. Clear the interrupt.

UART Controller

Introduction

The UART controller is a full-duplex asynchronous receiver and transmitter that supports a wide range of programmable baud rates and I/O signal formats. The controller can accommodate automatic parity generation and multi-master detection mode. The PS UART interface specifications (RX/TX baud rate and clock frequency) are listed in the *Zynq UltraScale+ MPSoC Data Sheet: DC and AC Switching Characteristics* (DS925) [Ref 2].

The UART operations are controlled by the configuration and mode registers. The state of the FIFOs, modem signals, and other controller functions are read using the status, interrupt status, and modem status registers.

The controller is structured with separate RX and TX data paths. Each path includes a 64-byte FIFO. The controller serializes and deserializes data in the TX and RX FIFOs and includes a mode switch to support various loopback configurations for the RxD and TxD signals. The FIFO interrupt status bits support polling or interrupt driven handler. Software reads and writes data bytes using the RX and TX data port registers.

When using the UART in a modem-like application, the modem control module detects and generates the modem handshake signals and also controls the receiver and transmitter paths according to the handshaking protocol.

Features

- Programmable baud rate generator
- Configurable receive and transmit FIFOs, with byte, two-byte or four-byte APB access mechanisms
- 6, 7, or 8 data bits
- 1, 1.5, or 2 stop bits
- Odd, even, space, mark, or no parity
- Parity, framing, and overflow error detection
- Line break generation and detection
- Automatic echo, local loopback, and remote loopback channel modes

- Interrupt generation
- Modem control signals: CTS, RTS, DSR, DTR, RI, and DCD
- UART has two clocks. The advanced peripheral bus (APB) clocks up to 100 MHz. The uart_ref_clock ranges from 1 MHz to 100 MHz.

UART Controller Functional Description

UART Controller Block Diagram

The block diagram for the UART controller is shown in Figure 21-1.

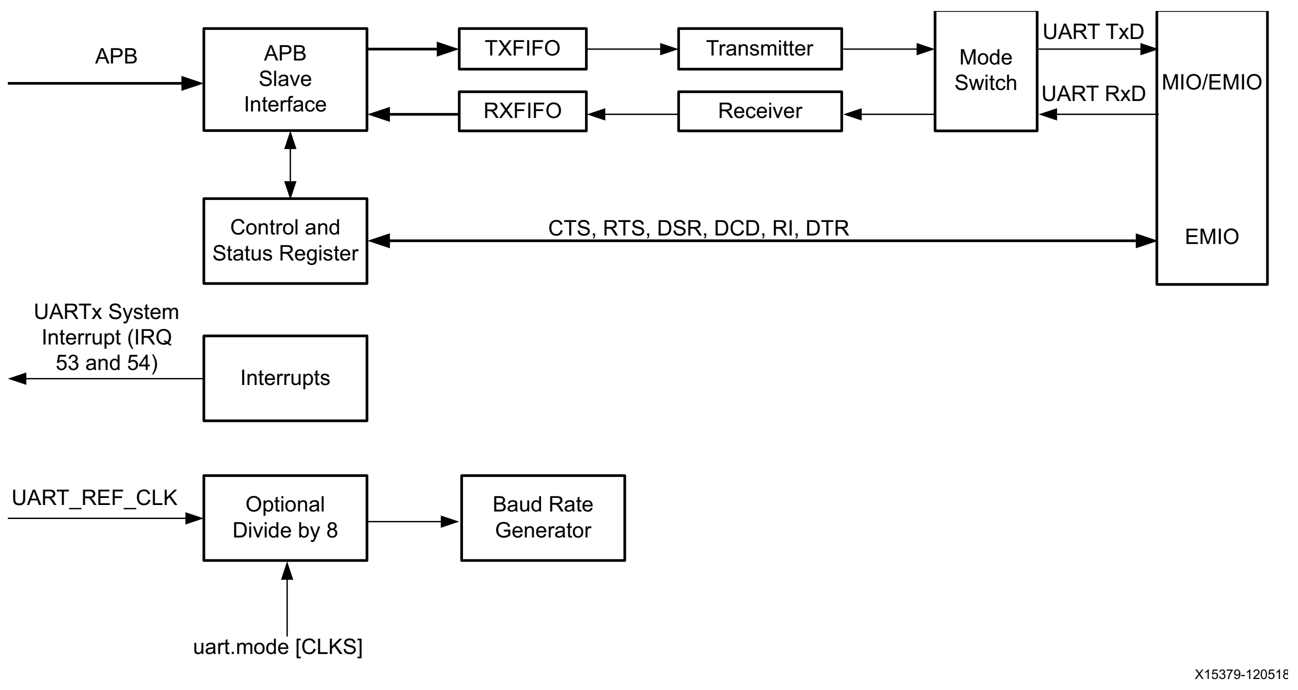


Figure 21-1: **UART Controller**

Control Logic

The control logic contains the control register and the mode register that are used to select the various operating modes of the UART.

The control register enables, disables, and issues soft resets to the receiver and transmitter modules. In addition, it restarts the receiver timeout period and controls the transmitter break logic. The receive line break detection must be implemented in Software. It is indicated by a frame error followed by one or more zero bytes in the RXFIFO.

The mode register selects the clock used by the baud rate generator. It also selects the bit length, parity bit, and stop bit used by the transmitted and received data. In addition, it selects the mode of operation of the UART, switching between normal UART mode, automatic echo, local loopback, or remote loopback, as required.

Baud Rate Generator

The baud rate generator furnishes the bit period clock, or baud rate clock, for both the receiver and the transmitter. The baud rate clock is implemented by distributing the base clock UART_REF_CLK and a single cycle clock enable to achieve the effect of clocking at the appropriate frequency division. The effective logic for the baud rate generation is shown in Figure 21-2.

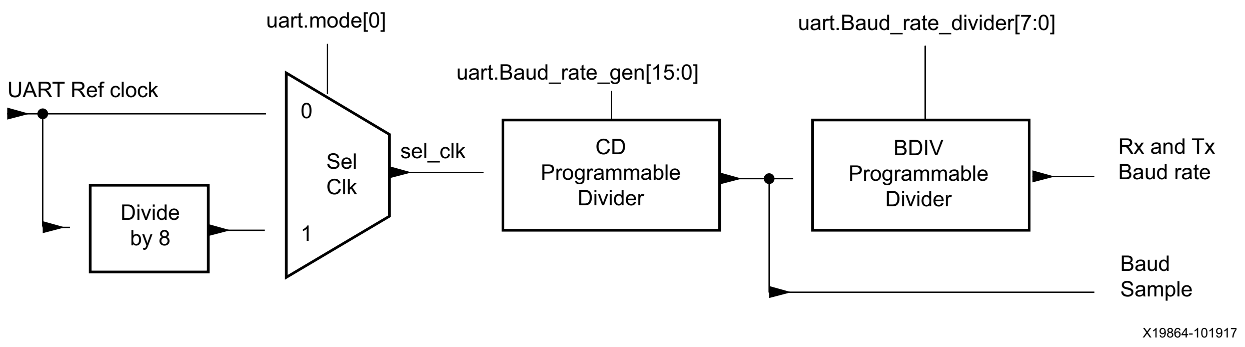


Figure 21-2: **UART Board Rate Generator**

The baud rate generator can use either the master clock signal, UART_REF_CLK, or the master clock divided by eight, UART_REF_CLK/8. The clock signal used is selected according to the value of the CLKS bit in the Mode register (uart.mode). The resulting selected clock is termed sel_clk in the following description.

The sel_clk clock is divided down to generate three other clocks: baud_sample, baud_tx_rate, and baud_rx_rate. The baud_tx_rate is the target baud rate used for transmitting data. The baud_rx_rate is nominally at the same rate, but gets resynchronized to the incoming received data. The baud_sample runs at a multiple ([BDIV] + 1) of baud_rx_rate and baud_tx_rate and is used to over-sample the received data.

The sel_clk clock frequency is divided by the CD field value in the Baud Rate Generator register to generate the baud_sample clock enable. This register can be programmed with a value between 1 and 65535.

The baud_sample clock is divided by [BDIV] plus 1. BDIV is a programmable field in the Baud Rate Divider register and can be programmed with a value between 4 and 255. It has a reset value of 15, inferring a default ratio of 16 baud_sample clocks per baud_tx_clock / baud_rx_rate.

The frequency of the baud_sample clock enable is shown in Equation 21-1.

$$baud\_sample = \frac{sel\_clk}{CD} \qquad \text{Equation 21-1}$$

The frequency of the baud_rx_rate and baud_tx_rate clock enables is show in Equation Equation 21-2.

$$baud\_rate = \frac{sel\_clk}{CD \times (BDIV + 1)} \qquad \text{Equation 21-2}$$

IMPORTANT: *It is essential to disable the transmitter and receiver before writing to the Baud Rate Generator register (uart.Baud_rate_gen), or the baud rate divider register (uart.Baud_rate_divider). A soft reset must be issued to both the transmitter and receiver before they are re-enabled.*

Some examples of the relationship between the UART_REF_CLK clock, baud rate, clock divisors (CD and BDIV), and the rate of error are shown in Table 21-1. The highlighted entry shows the default reset values for CD and BDIV. For these examples, a system clock rate of UART_REF_CLK = 50 MHz and UART_REF_CLK/8 = 6.25 MHz is assumed. The frequency of the UART reference clock can be changed to get a more accurate Baud rate frequency, refer to Chapter 37, PS Clock Subsystem for details to program the UART_REF_CLK.

Table 21-1: **UART Parameter Value Examples**

Clock	Baud Rate	Calculated CD	Actual CD	BDIV	Actual Baud Rate	Error (BPS)	% Error
UART_REF_CLK	600	10416.667	10417	7	599.980	0.020	-0.003
UART_REF_CLK /8	9,600	81.380	81	7	9,645.061	45.061	0.469
UART_REF_CLK	9,600	651.041	651	7	9,600.614	0.614	0.006
UART_REF_CLK	28,800	347.222	347	4	28,818.44	18.44	0.064
UART_REF_CLK	115,200	62.004	62	6	115,207.37	7.373	0.0064
UART_REF_CLK	230,400	31.002	31	6	230,414.75	14.75	0.006
UART_REF_CLK	460,800	27.127	9	11	462,962.96	2,162.96	0.469
UART_REF_CLK	921,600	9.042	9	5	925,925.92	4,325.93	0.469

Transmit FIFO

The transmit FIFO (TxFIFO) stores data written from the APB interface until it is removed by the transmit module and loaded into its shift register. The TxFIFO’s maximum data width is eight bits. Data is loaded into the TxFIFO by writing to the TxFIFO register.

When data is loaded into the TxFIFO, the TxFIFO empty flag is cleared and remains in this Low state until the last word in the TxFIFO has been removed and loaded into the transmitter shift register. This means that host software has another full serial word time until the next data is needed, allowing it to react to the empty flag being set and write another word in the TxFIFO without loss in transmission time.

The TxFIFO full interrupt status (TXFULL) indicates that the TxFIFO is completely full and prevents any further data from being loaded into the TxFIFO. If another APB write to the TxFIFO is performed, an overflow is triggered and the write data is not loaded into the TxFIFO. The transmit FIFO nearly full flag (TNFUL) indicates that there is not enough free space in the FIFO for one more write of the programmed size, as controlled by the WSIZE bits of the Mode register.

The TxFIFO nearly-full flag (TNFUL) indicates that there is only byte free in the TxFIFO.

A threshold trigger (TTRIG) can be setup on the TxFIFO fill level. The Transmitter Trigger register can be used to setup this value, such that the trigger is set when the TxFIFO fill level reaches this programmed value.

Transmitter Data Stream

The transmit module removes parallel data from the TxFIFO and loads it into the transmitter shift register so that it can be serialized.

The transmit module shifts out the start bit, data bits, parity bit, and stop bits as a serial data stream. Data is transmitted, least significant bit first, on the falling edge of the transmit baud clock enable (baud_tx_rate). A typical transmitted data stream is illustrated in Figure 21-3.

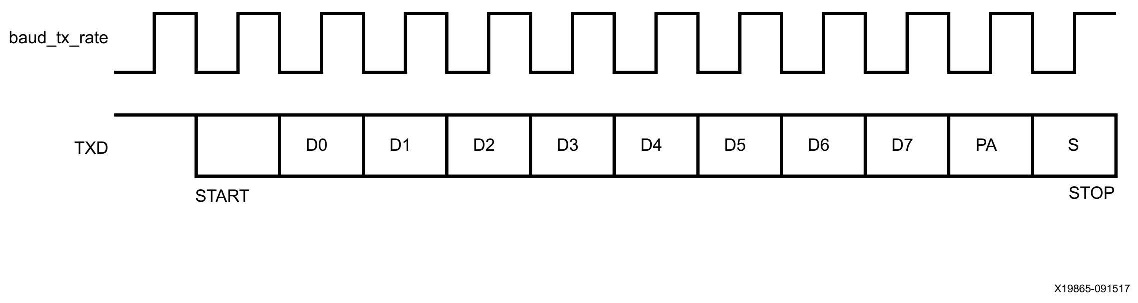


Figure 21-3: **Transmitted Data Stream**

The uart.mode[CHRL] register bit selects the character length, in terms of the number of data bits. The uart.mode[NBSTOP] register bit selects the number of stop bits to transmit.

Receiver FIFO

The RxFIFO stores data that is received by the receiver serial shift register. The RxFIFO's maximum data width is eight bits.

When data is loaded into the RxFIFO, the RxFIFO empty flag is cleared and this state remains Low until all data in the RxFIFO has been transferred through the APB interface. Reading from an empty RxFIFO returns zero.

The RxFIFO full status (Chnl_int_sts [RXFULL] and Channel_sts [RXFULL] bits) indicates that the RxFIFO is full and prevents any further data from being loaded into the RxFIFO. When a space becomes available in the RxFIFO, any character stored in the receiver will be loaded.

A threshold trigger (RTRIG) can be setup on the RxFIFO fill level. The Receiver Trigger Level register (Rcvr_FIFO_trigger_level) can be used to setup this value, such that the trigger is set when the RxFIFO fill level transitions this programmed value. The Range is 1 to 63.

Receiver Data Capture

The UART continuously over-samples the UARTx_RxD signal using UART_REF_CLK and the clock enable (baud_sample). When the samples detect a transition to a Low level, it can indicate the beginning of a start bit. When the UART senses a Low level at the UART_RxD input, it waits for a count of half of BDIV baud rate clock cycles, and then samples three more times. If all three bits still indicate a Low level, the receiver considers this to be a valid start bit, as illustrated in Figure 21-4 for the default BDIV of 15.

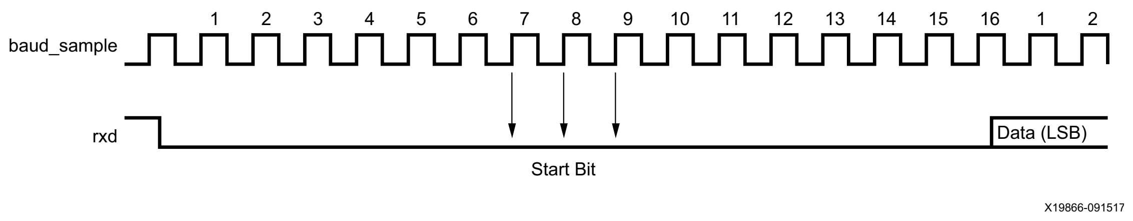


Figure 21-4: **Default BDIV Receiver Data Stream**

When a valid start bit is identified, the receiver baud rate clock enable (baud_rx_rate) is re-synchronized so that further sampling of the incoming UART RxD signal occurs around the theoretical mid-point of each bit, as illustrated in Figure 21-5.

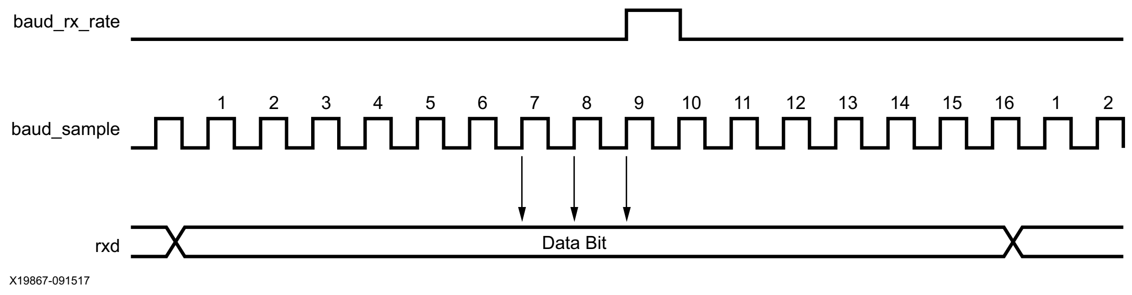


Figure 21-5: **Re-synchronized Receiver Data Stream**

When the re-synchronized baud_rx_rate is High, the last three sampled bits are compared. The logic value is determined by majority voting; two samples having the same value define the value of the data bit. When the value of a serial data bit has been determined, it is shifted to the receive shift register. When a complete character has been assembled, the contents of the register are then pushed to the RxFIFO.

Receiver Parity Error

Each time a character is received, the receiver calculates the parity of the received data bits in accordance with the uart.mode [PAR] bit field. It then compares the result with the received parity bit. If a difference is detected, the parity error bit is set = `1`, uart.Chnl_int_sts [PARITY]. An interrupt is generated, if enabled.

Receiver Framing Error

When the receiver fails to receive a valid stop bit at the end of a frame, the frame error bit is set = `1`, uart.Chnl_int_sts [FRAMING]. An interrupt is generated, if enabled.

Receiver Overflow Error

When a character is received, the controller checks to see if the RxFIFO has room. If it does, then the character is written into the RxFIFO. If the RxFIFO is full, then the controller waits. If a subsequent start bit on RxD is detected and the RxFIFO is still full, then data is lost and the controller sets the Rx overflow interrupt bit, uart.Chnl_int_sts [OVER] = `1`. An interrupt is generated, if enabled.

Receiver Timeout Mechanism

The receiver timeout mechanism enables the receiver to detect an inactive RxD signal (a persistent High level). The timeout period is programmed by writing to the uart.Rcvr_timeout [RTO] bit field. The timeout mechanism uses a 10-bit decrementing counter. The counter is reloaded and starts counting down whenever a new start bit is received on the RxD signal, or whenever software writes a `1` to uart.Control [TORST] (regardless of the previous [TORST] value).

If no start bit or reset timeout occurs for 1,023 bit periods, a timeout occurs. The Receiver timeout error bit [TOUT] will be set in the interrupt status register, and the [TORST] bit in the Control register should be written with a `1` to restart the timeout counter, which loads the newly programmed timeout value.

The upper 8 bits of the counter are reloaded from the value in the [RTO] bit field and the lower 2 bits are initialized to zero. The counter is clocked by the UART bit clock. As an example, if [RTO] = `0xFF`, then the timeout period is 1,023 bit clocks (256 x 4 minus 1). If `0` is written into the [RTO] bit, the timeout mechanism is disabled.

When the decrementing counter reaches 0, the receiver timeout occurs and the controller sets the timeout interrupt status bit uart.Chnl_int_sts [TOUT] = `1`. If the interrupt is enabled (uart.Intrpt_mask [TOUT] = `1`), then the IRQ signal to the PS interrupt controller is asserted.

Whenever the timeout interrupt occurs, it is cleared with a write back of `1` to the Chnl_int_sts [TOUT] bit. Software must set uart.Control [TORST] = `1` to generate further receive timeout interrupts.

I/O Mode Switch

The mode switch controls the routing of the RxD and TxD signals within the controller as shown in Figure 21-6. The loopback using the mode switch occurs regardless of the MIO-EMIO routing of the UARTx TxD/RxD I/O signals. There are four operating modes as shown in Figure 21-6. The mode is controlled by the uart.mode [CHMODE] register bit field: normal, automatic echo, local loopback and remote loopback.

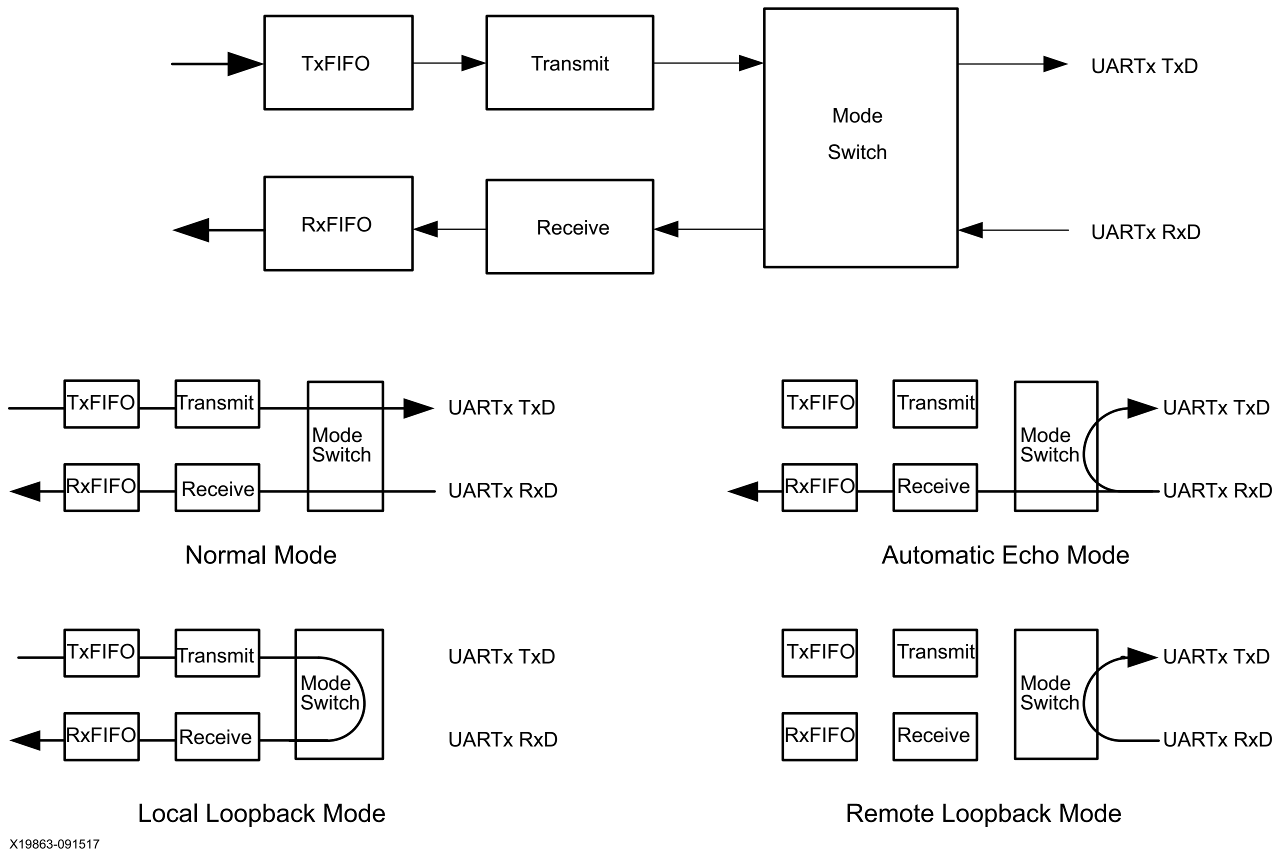


Figure 21-6: **UART Mode Switch for TxD and RxD**

Normal Mode

Normal mode is used for standard UART operations.

Automatic Echo Mode

Echo mode receives data on RxD and the mode switch routes the data to both the receiver and the TxD pin. Data from the transmitter cannot be sent out from the controller.

Local Loopback Mode

Local loopback mode does not connect to the RxD or TxD pins. Instead, the transmitted data is looped back around to the receiver.

Remote Loopback Mode

Remote loopback mode connects the RxD signal to the TxD signal. In this mode, the controller cannot send anything on TxD and the controller does not receive anything on RxD.

UART0-to-UART1 Connection

The I/O signals of the two UART controllers in the PS can be connected together. In this mode, the RxD and CTS input signals from one controller are connected to the TxD and RTS output signals of the other UART controller by setting the iou_slcr.MIO_LOOPBACK [UA0_LOOP_UA1] bit = `1`. The other flow control signals are not connected. This UART-to-UART connection occurs regardless of the MIO-EMIO programming.

Status and Interrupts

Interrupt and Status Registers

There are two status registers that can be read by software. Both show raw status. The Chnl_int_sts register can be read for status and generate an interrupt. The Channel_sts register can only be read for status.

The Chnl_int_sts register is sticky; once a bit is set, the bit stays set until software clears it. Write a `1` to clear a bit. This register is bit-wise AND'ed with the Intrpt_mask mask register. If any of the bit-wise AND functions have a result = `1`, then the UART interrupt is asserted to the PS interrupt controller.

- **Channel_sts**: Read-only raw status. Writes are ignored.

The various FIFO and system indicators are routed to the uart.Channel_sts register and/or the uart.Chnl_int_sts register as shown in Figure 21-7.

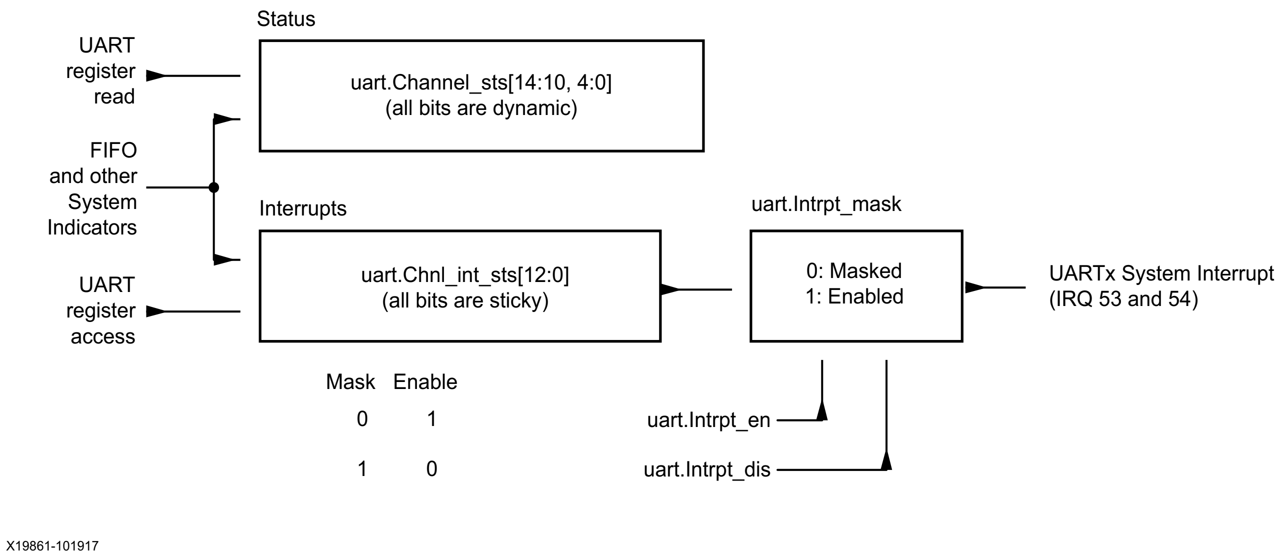

Status
UART register read
uart.Channel_sts[14:10, 4:0] (all bits are dynamic)
FIFO and other System Indicators
Interrupts
uart.Intrpt_mask
UART register access
uart.Chnl_int_sts[12:0] (all bits are sticky)
0: Masked
1: Enabled
UARTx System Interrupt (IRQ 53 and 54)
Mask Enable
0 1
1 0
uart.Intrpt_en
uart.Intrpt_dis
X19861-101917

Figure 21-7: **Interrupts and Status Signals**

The interrupt registers and bit fields are summarized in Table 21-2.

Table 21-2: **UART Interrupt Status Bits**

Interrupt Register Names and Bit Assignments														
14	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
uart.Intrpt_en uart.Intrpt_dis uart.Intrpt_mask uart.Chnl_int_sts														
x	RBRK	TOVR	TNFUL	TTRIG	DMS	TOUT	PARITY	FRAMING	OVER	TXFUL	TXEMPTY	RXFULL	RXEMPTY	RXOVR
uart.Channel_sts														
TNFUL	TTRIG	FLOWDEL	TACTIVE	RACTIVE	X	X	X	X	X	TXFUL	TXEMPTY	RXFULL	RXEMPTY	RXOVR

Interrupt Mask Register

Intrpt_mask is a read-only interrupt mask/enable register that is used to mask individual raw interrupts in the Chnl_int_sts register:

- If the mask bit = `0`, the interrupt is masked.
- If the mask bit = `1`, the interrupt is enabled.

This mask is controlled by the write-only Intrpt_en and Intrpt_dis registers. Each associated enable/disable interrupt bit should be set mutually exclusive (e.g., to enable an interrupt, write `1` to Intrpt_en[x] and write `0` to Intrpt_dis[x]).

Channel Status

These status bits are in the Channel_sts register.

- **TACTIVE**: Transmitter state machine active status. If in an active state, the transmitter is currently shifting out a character.
- **RACTIVE**: Receiver state machine active status. If in an active state, the receiver is has detected a start bit and is currently shifting in a character.
- **FLOWDEL**: Receiver flow delay trigger continuous status. The FLOWDEL status bit is used to monitor the RxFIFO level in comparison with the flow delay trigger level.

Non-FIFO Interrupts

These interrupt status bits are in the Chnl_int_sts register.

- **TOUT**: Receiver Timeout Error interrupt status. This event is triggered whenever the receiver timeout counter has expired due to a long idle condition.
- **PARITY**: Receiver Parity Error interrupt status. This event is triggered whenever the received parity bit does not match the expected value.
- **FRAMING**: Receiver Framing Error interrupt status. This event is triggered whenever the receiver fails to detect a valid stop bit. See Receiver Data Capture.
- **DMS**: indicates a change of logic level on the DCD, DSR, RI or CTS modem flow control signals. This includes High-to-Low and Low-to-High logic transitions on any of these signals.

FIFO Interrupts

The status bits for the FIFO interrupts listed in Table 21-2 are illustrated in Figure 21-8. These interrupt status bits are in the Channel Status (uart.Channel_sts) and Channel Interrupt Status (uart.Chnl_int_sts) registers.

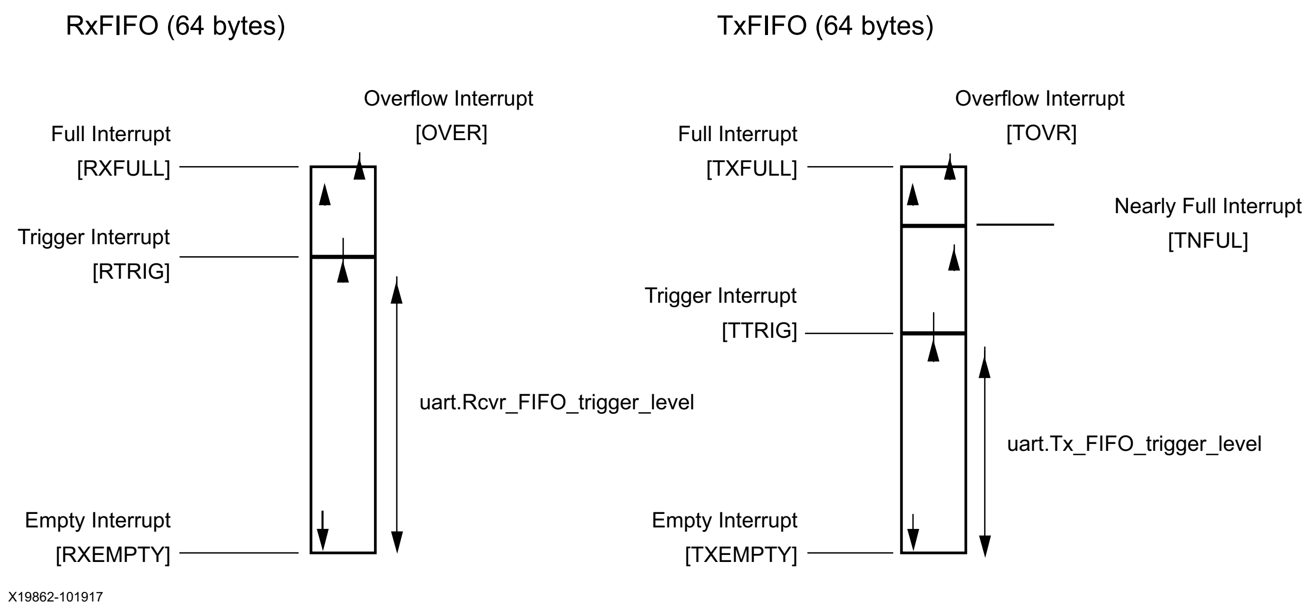


Figure 21-8: **UART RxFIFO and TxFIFO Interrupt**

The FIFO trigger levels are controlled by these bit fields:

- uart.Rcvr_FIFO_trigger_level[RTRIG], a 6-bit field
- uart.Tx_FIFO_trigger_level[TTRIG], a 6-bit field

Modem Control

The modem control module facilitates the control of communication between a modem and the UART. It contains the Modem Status register, the Modem Control register, the DMSI bit in interrupt status register, and FLOWDEL in the channel status register. This event is triggered whenever the CTS, DSR, RIX, or DCD in the modem status register are being set.

The read-only Modem Status register is used to read the values of the clear to send (CTS), data carrier detect (DCD), data set ready, (DSR) and ring indicator (RI) modem inputs. It also reports changes in any of these inputs and indicates whether automatic flow control mode is currently enabled. The bits in the Modem Status register are cleared by writing a `1` to the particular bit.

The read/write only Modem Control register is used to set the data terminal ready (DTR) and request to send (RTS) outputs, and to enable the Automatic Flow Control Mode register.

By default, the automatic flow control mode is disabled, meaning that the modem inputs and outputs work completely under software control. When the automatic flow control mode is enabled by setting the FCM bit in the Modem Control register, the UART transmission and reception status is automatically controlled using the modem handshake inputs and outputs.

In automatic flow control mode the request to send output is asserted and deasserted based on the current fill level of the receiver FIFO, which results in the far-end transmitter pausing transmission and preventing an overflow of the UART receiver FIFO. The FDEL field in the Flow Delay register (Flow_delay) is used to setup a trigger level on the Receiver FIFO which causes the deassertion of the request to send. It remains Low until the FIFO level has dropped to below four less than FDEL.

Additionally in automatic flow control mode, the UART only transmits while the clear to send input is asserted. When the clear to send is deasserted, the UART pauses transmission at the next character boundary.

If flow control is selected as automatic, then Flow Delay register must be programmed in order to have a control on the inflow of data, which is done by deasserting RTS signal. The value corresponds to the RxFIFO level at which RTS signal will be deasserted. It will be reasserted when the RxFIFO level drops to four below the value programmed in the Flow Delay register.

The uart.Channel_sts [FLOWDEL] register bit is used to monitor the RxFIFO level in comparison with the flow delay trigger level. The [FLOWDEL] bit is set whenever the RxFIFO level is greater than or equal to trigger the level programmed in the Flow Delay register.

The trigger level programmed in the Flow Delay register has no dependency on the Rx Trigger Level register. This is to only control the inflow of data using the RTS modem signal.

The CPU will be interrupted by receive data only on receipt of an Rx Trigger interrupt. Data is retrieved based on the trigger level programmed in the Rx Trigger Level register.

UART Controller Register Overview

An overview of the UART registers is shown in Table 21-3.

Table 21-3: **UART Registers**

Type	Register Names	Description
Configuration	Control Mode Baud_rate_gen Baud_rate_divider	Configure mode and baud rate.
Interrupt processing	Intrpt_en Intrpt_dis Intrpt_mask Chnl_int_sts Channel_sts	Enable/disable interrupt mask, channel interrupt status, channel status.
RX and TX data	TX_RX_FIFO0	Read data received. Write data to be transmitted.
Receiver	Rcvr_timeout Rcvr_FIFO_trigger_level	Configure receiver timeout and RXFIFO trigger level value.
Transmitter	Tx_FIFO_trigger_level	Configure TXFIFO trigger level value.
Modem	Modem_ctrl Modem_sts Flow_delay	Configure modem-like application.

Clocks

The controller and I/O interface are driven by the reference clock (UART{0, 1}_REF_CTRL). The controller's interconnect also requires an APB interface clock (LSBUS clock). Both of these clocks always come from the PS clock subsystem.

LSBUS Clock

See Chapter 37, PS Clock Subsystem for more information. The LSBUS Clock runs asynchronous to the UART reference clock.

Reference Clock

The reference clock is generated based on the generic clocking diagram shown in Figure 37-4. The input clock source can be selected based on the crl_apb.UART{0,1}_REF_CTRL [srcsel] bits, where the source can be from the RPLL, IOPLL, or DPLL. The crl_apb.UART{0,1}_REF_CTRL [divisor0] register selects the 6-bit programmable divider 0. The crl_apb.UART{0,1}_REF_CTRL [divisor1] register selects the 6-bit programmable divider 1.The crl_apb.UART{0,1}_REF_CTRL [clkact] bit selects whether the clock should be gated or enabled.

Resets

The controller reset bits are generated by the PS, see Chapter 38, Reset System.

MIO – EMIO Signals

The UART I/O signals are identified in Table 21-4. The MIO pins and any restrictions based on device versions are shown in Table 28-1 in Chapter 28, Multiplexed I/O.

Table 21-4: **UART MIO Pins and EMIO Signals**

UART Interface Signal	Default Controller Input Value	MIO Pins		EMIO Signals	
		Numbers	I/O	Name	I/O
UART 0 Transmit	~	3, 7, 11, 15, 19, 23, 27, 31, 35, 39, 43, 47, 51, 55, 59, 63, 67, 71, 75	O	EMIOUART0TX	O
UART 0 Receive		2, 6, 10, 14, 18, 22, 26, 30, 34, 38, 42, 46, 50, 54, 58, 62, 66, 70, 74	I	EMIOUART0RX	I
UART 0 Clear to Send		~	~	EMIOUART0CTSN	I
UART 0 Ready to Send	~	~	~	EMIOUART0RTSN	O
UART 0 Data Set Ready		~	~	EMIOUART0DSRN	I
UART 0 Data Carrier Detect		~	~	EMIOUART0DCDN	I
UART 0 Ring Indicator		~	~	EMIOUART0RIN	I
UART 0 Data Terminal Ready	~	~	~	EMIOUART0DTRN	O
UART 1 Transmit	~	0, 4, 8, 12, 16, 20, 24, 28, 32, 36, 40, 44, 48, 52, 56, 60, 64, 68, 72	O	EMIOUART1TX	O
UART 1 Receive		1, 5, 9, 13, 17, 21, 25, 29, 33, 37, 41, 45, 49, 53, 57, 61, 65, 69, 73	I	EMIOUART1RX	I
UART 1 Clear to Send		~	~	EMIOUART1CTSN	I
UART 1 Ready to Send	~	~	~	EMIOUART1RTSN	O
UART 1 Data Set Ready		~	~	EMIOUART1DSRN	I
UART 1 Data Carrier Detect		~	~	EMIOUART1DCDN	I
UART 1 Ring Indicator		~	~	EMIOUART1RIN	I
UART 1 Data Terminal Ready	~	~	~	EMIOUART1DTRN	O

UART Controller Programming Model

The flow diagram for the UART controller programming sequence is shown in Figure 21-9.

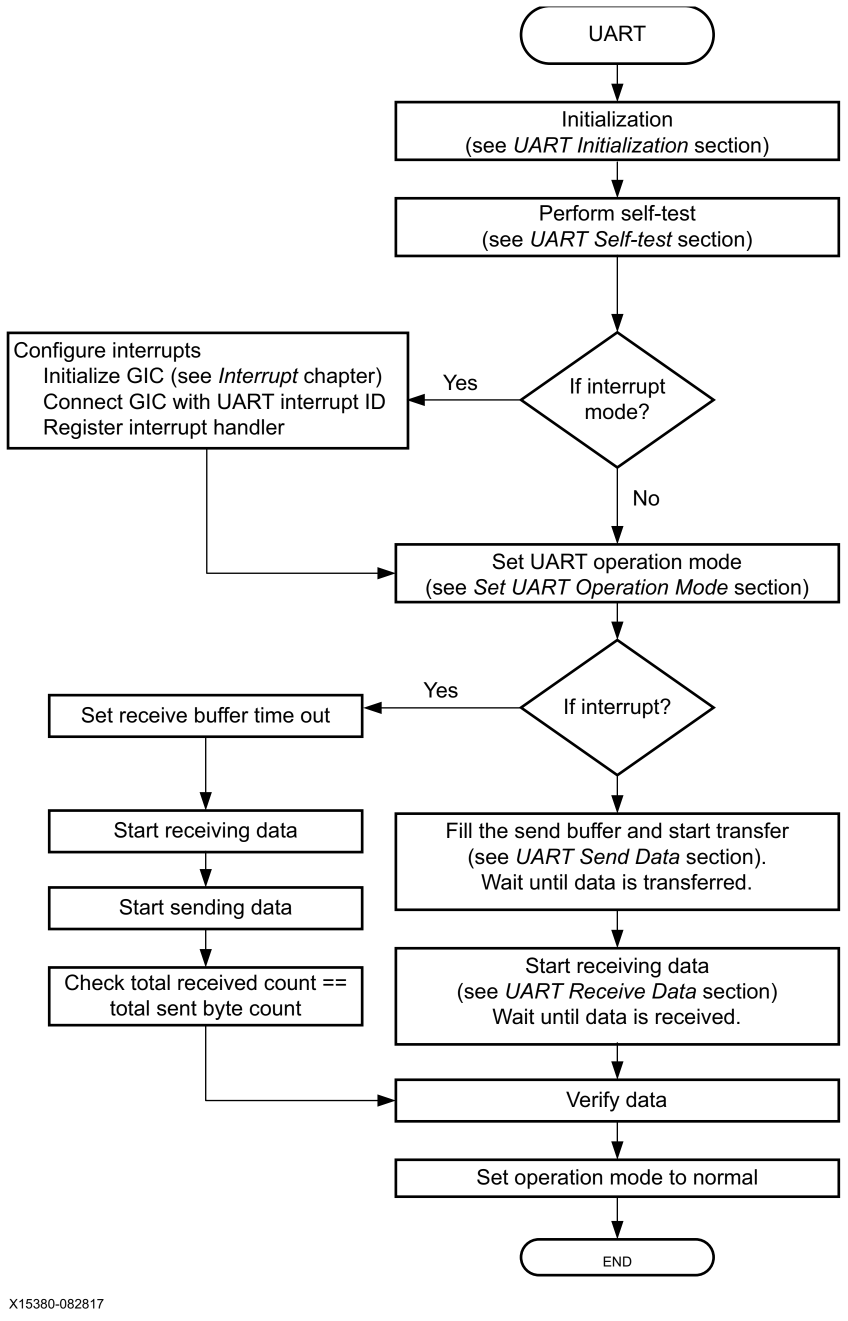

UART
Initialization
(see *UART Initialization* section)
Perform self-test
(see *UART Self-test* section)
If interrupt mode?
Yes
Configure interrupts
Initialize GIC (see *Interrupt* chapter)
Connect GIC with UART interrupt ID
Register interrupt handler
No
Set UART operation mode
(see *Set UART Operation Mode* section)
If interrupt?
Yes
Set receive buffer time out
Start receiving data
Start sending data
Check total received count == total sent byte count
Fill the send buffer and start transfer
(see *UART Send Data* section).
Wait until data is transferred.
Start receiving data
(see *UART Receive Data* section)
Wait until data is received.
Verify data
Set operation mode to normal
END
X15380-082817

Figure 21-9: **UART Controller Flowchart**

UART Controller Programming

The programming steps/tasks for the UART controller are listed in Table 21-5 through Table 21-9.

- UART Configuration
- UART Self Test
- UART Set Operating Mode
- UART Send Data
- UART Receive Data

Table 21-5: **UART Configuration**

Task	Register	Register Field	Register Offset	Bits	Note
Set the default baud rate 115200 b/s.					
Fastest possible input clock data rate is 25 MHz/2, ensure requested data rate does not exceed the limit.					
Read bit 0 in mode register (base + `0x04`) to check if clock/8 option is set.					
Check if input clock is divided by 8.	Mode register	Clk_sel	`0x004`	0	Read operation
If clk_sel bit is set, divide input clock value by 8 to calculate baud rate.					
Calculate baud rate generator value (Best_BRGR).					
Calculate baud rate divider value (Best_BAUDDIV).					
Disable UART	Configuration register	RX_DIS | TX_DIS	`0x000`	5 and 3	`28h`
Clear bit 5:2 in base + `0x00`.					
Set bit 5 and bit 3 in base + `0x00`.					
Write baud rate generator value	Baud generator register	All	`0x018`	31:0	Best_BRGR
Write baud rate divider value	Baud divider register	All	`0x0034`	31:0	Best_BAUDDIV
Reset TX and RX	Configuration register	TX_RST | RX_RST	`0x00`	1:0	`11b`
Enable UART	Configuration register	RX_DIS | TX_DIS	`0x000`	5 and 3	`0`
Clear bit 5:2 in base + `0x00`.					
Set bit 2 and bit 4 in base + `0x00`.					
Clear bits 1, 2, 3, 4, 5, and 7 in base + `0x04`. Set bit 5 in base + `0x04`.					
Set mode to 8-bit, 1 stop, and no parity	Mode register	CHAR_LEN_8BIT | PARITY_NONE | STOPMODE_1	`0X004`	7, 5:1	Reg = `20h`

Table 21-5: **UART Configuration *(Cont'd)***

Task	Register	Register Field	Register Offset	Bits	Note
Write value to set RXFIFO trigger 8 bytes.	RX_WM	All	`0x0020`	31:0	`8h`
Write RX time-out value.	Timeout register	All	`0x001C`	31:0	`1h`
Write values to disable all interrupts.	Interrupt disable	All	`0x00C`	12:0	`1FFFh`

Table 21-6: **UART Self Test**

Task	Register	Register Field	Register Offset	Bits	Note
Save interrupt mask register contents.	Interrupt mask register	All	`0x0010`	31:0	Read operation
Disable all interrupts.	Interrupt disable	All	`0x00C`	12:0	`1FFFh`
Save mode register contents.	Mode register	All	`0x004`	31:0	Read operation
Enable local loopback.	Mode register	CH_mode L_LOOP	`0x004`	9:8	`10b`
Sending data refer section UART send data.					
Wait until RXFIFO empty flag cleared	Status register	RX_EMPTY	`0x02c`	2	Read and check
Wait until RXFIFO empty flag is set.					
Receive data, refer to UART receive data section. Repeat previous three steps until all bytes (sent and received) are transferred.					
Verify all data received (both send and receive buffers).					
Restore (write back) interrupt mask value saved in first step.	Interrupt enable register	All	`0x008`	31:0	Value read in the first task of this UART self-test procedure.
Restore mode register contents.	Mode register	All	`0x004`	31:0	Value read in the third task of this UART self-test procedure.

Table 21-7: **UART Set Operating Mode**

Task	Register	Register Field	Register Offset	Bits	Note
Clear bits 9 and 8 in base + `0x04`. Then, set bits 9:8.					
For normal mode (0).	Mode register	CHMODE	`0x004`	9:8	`00b`
For auto mode (1).	Mode register	CHMODE	`0x004`	9:8	`01b`
For local loop back mode (2).	Mode register	CHMODE	`0x004`	9:8	`10b`
For remote loop back mode (3).	Mode register	CHMODE	`0x004`	9:8	`11b`

Table 21-8: **UART Send Data**

Task	Register	Register Field	Register Offset	Bits	Note
Set bits 8 and 7 in base+`0x0C` to disable interrupts.					
Disable interrupts.	Interrupt disable register	TX_EMPTY | TX_FULL	`0x0C`	4:3	`11b`
Check bit number 8 in base+`0x2C`.					
Check if TX_FULL bit is set, if TXFIFO is full, send nothing.	Status register	TXFULL	`0x02C`	4	Read
If TX_FULL is not set, fill the remaining bytes.	TXFIFO register	All	`0x0030`	31:00	Data to be sent
Perform previous two tasks until all bytes transferred.					
Read interrupt mask register	Interrupt mask register	RX_FULL | RX_EMPTY | RX_OVR_FLW	`0x0010`	2:0	Read
If any of RXFIFO full or RXFIFO empty or RX overflow interrupts are set, enable TX empty interrupt by setting bit 3 in base + `0x008`.					
If any bit set from previous operation, write TXEMPTY bit to 1.	Interrupt enable register	TX_EMPTY	`0x8`	3	`1b`
Wait until the transfer is over by monitoring bit 11 (transfer active) and bit 7 (TX empty) in base + `0x2C`.					

Table 21-9: **UART Receive Data**

Task	Register	Register Field	Register Offset	Bits	Note
Save interrupt mask register (base+`0x10`) offset contents.					
Disable interrupts.	Interrupt disable register	All	`0x0C`	4:3	`1FFFh`
Wait until all data received by checking the bit 1 in status register (base+`0x002C`) to check the RXFIFO is empty.					
Check RX_EMPTY flag.	Status register	RX_EMPTY	`0x002C`	1	Read operation
Receive data by reading FIFO register.	FIFO register	All	`0x0030`	31:0	Read data operation
Do the previous two operations until RX_EMPTY is not set and bytes yet to be sent.					
Restore the interrupt mask register.	Interrupt enable register	All	`0x008`	31:0	Value read in first step.

Chapter 22

I2C Controllers

Introduction

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 modes. In master mode, a transfer can only be initiated by the processor writing the slave address into the I2C address register. The processor is notified of any available received data by a data interrupt or a transfer complete interrupt. If the hold bit is set, the I2C interface holds the clock signal (SCL) Low after the data is transmitted to support slow processor service. The master can be programmed to use both normal addressing and extended addressing modes. The extended addressing mode is only supported in master mode.

In slave monitor mode, the I2C interface 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.

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, [TO] 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, a [TO] interrupt bit is generated to avoid stall conditions.

I2C Controller Features

There are two I2C controllers in the LPD IOP section of the PS.

- I2C bus specification version 2.
- 16-byte FIFO.
- Programmable normal and fast bus data rates.
- Multi-master support.
- Master mode
    - Read and write transfers
    - Seven and 10-bit addressing.
    - Clock stretching by allowing hold for slow processor service.
    - [TO] interrupt bit to avoid stall condition.
    - Repeated start.
    - Slave monitor mode
- Slave mode
    - Transmit and receive.
    - Fully programmable slave response address.
    - [HOLD] bit helps to prevent the overflow condition.
    - [TO] bit helps interrupt flag to avoid stall condition.
    - Clock stretching helps to delay communication if data is not readily available.
- Software can poll for status or function as interrupt-driven device.
- Programmable interrupt generation.

Functional Description

System Block Diagram

The system viewpoint diagram for the I2C module is shown in Figure 22-1.

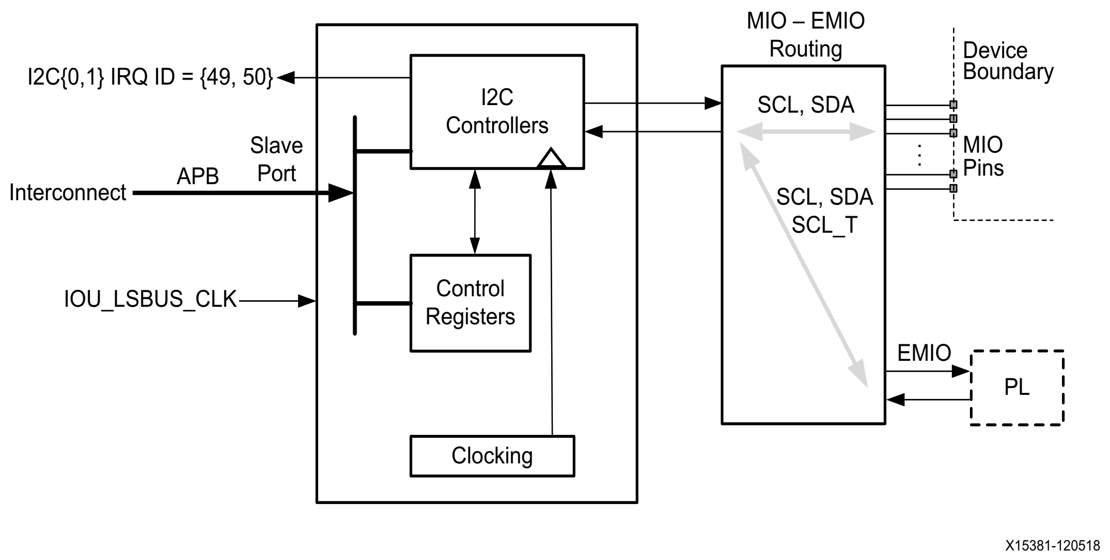


Figure 22-1: **I2C System Block Diagram**

I2C Module Block Diagram

The block diagram of I2C module is shown in Figure 22-2.

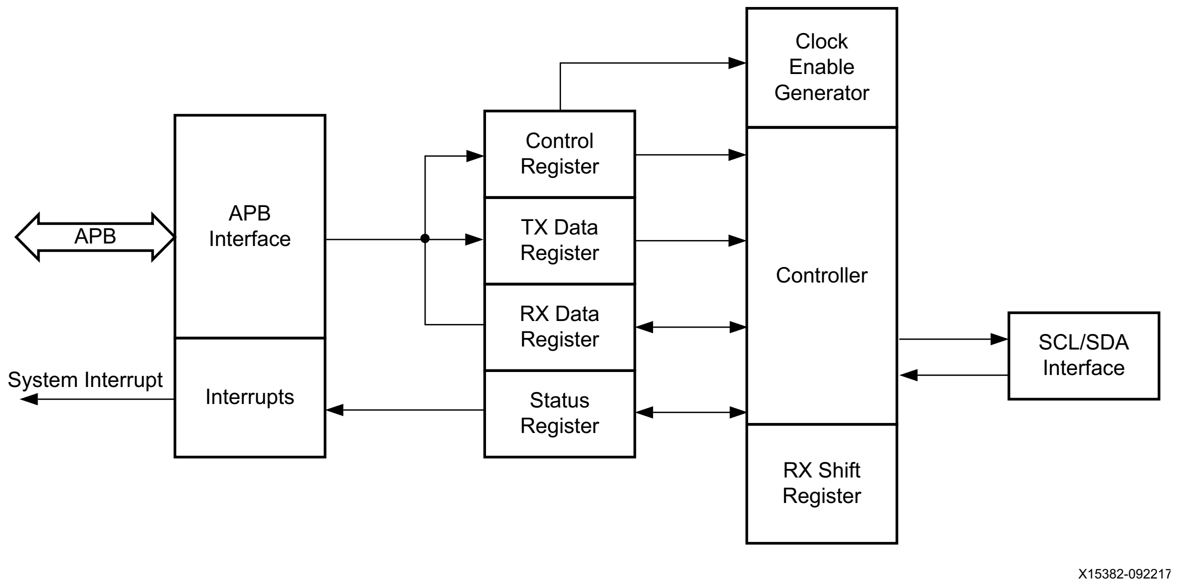


Figure 22-2: **I2C Block Diagram**

I2C Master Mode

The master is always the device that drives the SCL clock signal. The slaves are the devices that respond to the master. There can be multiple slaves on the I2C bus however, there is normally only one master. It is possible to have multiple masters. To select master mode, set Control [MS] bit = 1.

Slave Monitoring

The slave monitoring option is available in the master mode by setting Control [SLVMON] bit = 1. To monitor a specific slave on the bus, wait until the ACK is received or a timeout occurs.

10-bit Addressing Mode

The I2C controller supports the 10-bit addressing mode. The extended addressing mode is only supported in master mode. When the controller uses 10-bit addressing, it enables the mode and writes the 10-bit address:

1. Set I2C{0,1}.Control_Reg[NEA] = 0.
2. Write 10-bit address to the I2C Address register (I2C{0,1}.I2C_Address).

Note: In the Linux flow, a 10-bit address needs to be passed to the I2C core layer through the device tree or through the IOCTL.

I2C Slave Mode

When configured in slave mode, the I2C controller can only respond 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. To configure an I2C controller as a slave, set Control [MS] bit = 0.

Glitch Filter

The I2C bus specification specifies that 50 ns glitches should be removed from the clock and data signals. The I2C controller provides a digital glitch filter for filtering glitches on the SDA and SCL inputs. The filter is built using a shift register and the filter length is specified in terms of APB interface clock cycles (LPD_LSBUS_CLK). The glitch filter control register Glitch_Filter is used to set the length of the glitch filter shift register. The appropriate value written into the Glitch_Filter register allows for the removal of 50 ns glitches. Consequently, the value written into the Glitch_filter_reg.GF register should be equal to the number of APB clock cycles that gives a total length of 50 ns. The default value is five. If the length of the glitch filter shift register is set to zero, then the glitch filter is bypassed.

I/O Signals

Table 22-1 lists the I2C interface signals by MIO pin number.

Table 22-1: **I2C Interface Pins and Signals**

I2C Interface	MIO Pins		EMIO Signals	
	Number	**I/O**	**Name**	**I/O**
I2C 0 SCL	2, 6, 10, 14, 18, 22, 26, 30, 34, 38, 42, 46, 50, 54, 58, 62, 66, 70, 74	I/O	EMIOI2C0SCLI	I
			EMIOI2C0SCLO	O
			MIOI2C0SCLT	O
I2C 0 SDA	3, 7, 11, 15, 19, 23, 27, 31, 35, 39, 43, 47, 51, 55, 59, 63, 67, 71, 75	I/O	EMIOI2C0SDAI	I
			EMIOI2C0SDAO	O
			EMIOI2C0SDAT	O
I2C 1 SCL	0, 4, 8, 12, 16, 20, 24, 28, 32, 36, 40, 44, 48, 52, 56, 60, 64, 68, 72, 76	I/O	EMIOI2C1SCLI	I
			EMIOI2C1SCLO	O
			MIOI2C1SCLT	O

Table 22-1: **I2C Interface Pins and Signals *(Cont'd)***

I2C Interface	MIO Pins		EMIO Signals	
	Number	**I/O**	**Name**	**I/O**
I2C 1 SDA	1, 5, 9, 13, 17, 21, 25, 29, 33, 37, 41, 45, 49, 53, 57, 61, 65, 69, 73, 77	I/O	EMIOI2C1SDAI	I
			EMIOI2C1SDAO	O
			EMIOI2C1SDAT	O

I2C0-to-I2C1 Loopback Connection

The I/O signals of the two I2C controllers in the PS are connected together when the iou_slcr.MIO_LOOPBACK [I2C0_LOOP_I2C1] bit is set = 1. In this mode, the serial clocks are connected together and the serial data signals are connected together.

Register Overview

An overview of the I2C registers is provided in Table 22-2.

Table 22-2: **I2C Register Overview**

Register Type	Register Name	Description
Configuration	Control_Reg	Configure the operating mode.
Data	Address Data Transfer_Size Slave_Mon_Pause Time_Out Status_Reg Glitch_Filter	Transfer data and monitor status.
Interrupt processing	Interrupt_Status Intrpt_Mask Intrpt_Enable Intrpt_Disable	Enable or disable interrupt detection, mask interrupt set to the interrupt controller, read raw interrupt status.

For individual register descriptions, refer to the *Zynq UltraScale+ MPSoC Register Reference* (UG1087) [Ref 4].

Interrupt Mask Register

Each bit in the interrupt mask register (IMR) corresponds to a bit in the interrupt status register (ISR). If bit [i] in the interrupt mask register is set, the corresponding bit in the interrupt status register is ignored. Otherwise, an interrupt is generated whenever bit [i] in the interrupt status register is set. Bits in the IMR mask register are set through a write to the interrupt disable register and are cleared by a write to the interrupt enable register. All mask bits are set (interrupts disabled) after reset. The interrupt mask register has the same bit field order as the interrupt status register.

Interrupt Enable Register

The interrupt enable register (IER) has the same bit field order as the interrupt status register. Setting a bit in the interrupt enable register clears the corresponding mask bit in the interrupt mask register, effectively enabling a corresponding interrupt to be generated.

Programming Model

The following steps are used to program and use the I2C controller.

Reset Controller

To reset the I2C controller using bits from the CRL_APB.RST_LPD_IOU2 register, which is a software control register.

- Assert the I2C 0 controller reset: write a 1 to the CRL_APB.RST_LPD_IOU2 [i2c0_reset] bit.
- Deassert the I2C 0 controller reset: write a 0 to the CRL_APB.RST_LPD_IOU2 [i2c0_reset] bit.

Similarly, the [i2c1_reset] bit of the register controls the reset for I2C 1 controller.

Configure I/O Signal Routing

The I2C SCL and SDA signals can be routed to one of many sets of MIO pins or to the EMIO interface. All of the I2C signals are listed in Table 22-1. The MIO pins are configured by accessing registers located in the IOU_SLCR register set.

Configure Clocks

The controller, I/O interface, and the APB interconnect are driven by the LPD_LSBUS_CLK clock. This clock is generated from the PS. The clock signal can be derived from any of the PLLs as described in Chapter 37, PS Clock Subsystem.

Controller Configuration

I2C transfer parameters are programmed using the Control register.

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. Write the slave address. Write the address to the Address register.
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 done 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.

I2C Controller Programming Sequence

The flow diagram for the I2C controller programming sequence is shown in Figure 22-3 and Figure 22-4.

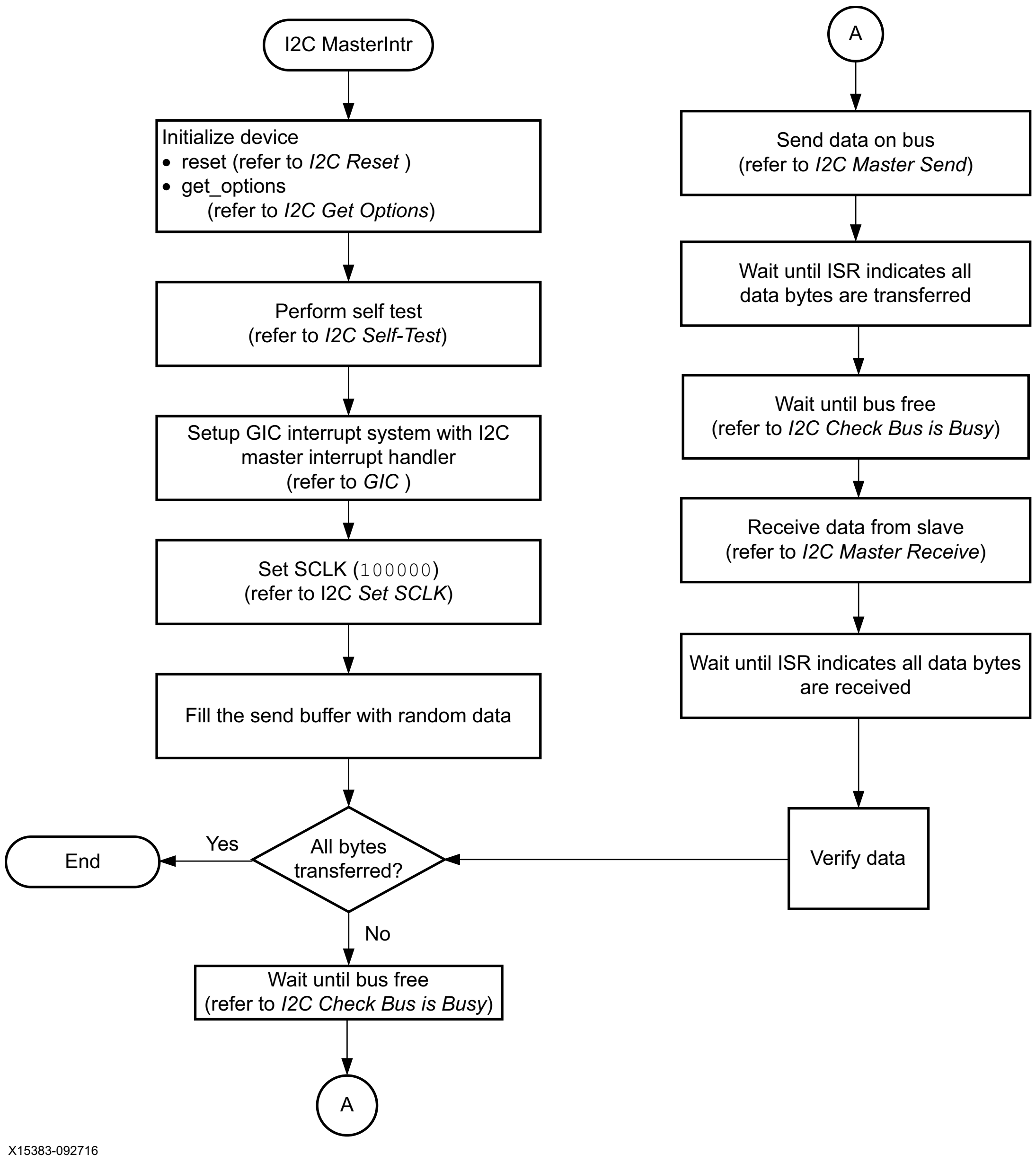

I2C MasterIntr
Initialize device
• reset (refer to I2C Reset)
• get_options (refer to I2C Get Options)
Perform self test (refer to I2C Self-Test)
Setup GIC interrupt system with I2C master interrupt handler (refer to GIC)
Set SCLK (100000) (refer to I2C Set SCLK)
Fill the send buffer with random data
All bytes transferred?
Yes
End
No
Wait until bus free (refer to I2C Check Bus is Busy)
A
A
Send data on bus (refer to I2C Master Send)
Wait until ISR indicates all data bytes are transferred
Wait until bus free (refer to I2C Check Bus is Busy)
Receive data from slave (refer to I2C Master Receive)
Wait until ISR indicates all data bytes are received
Verify data
X15383-092716

Figure 22-3: **I2C Master Interrupt Example Flowchart**

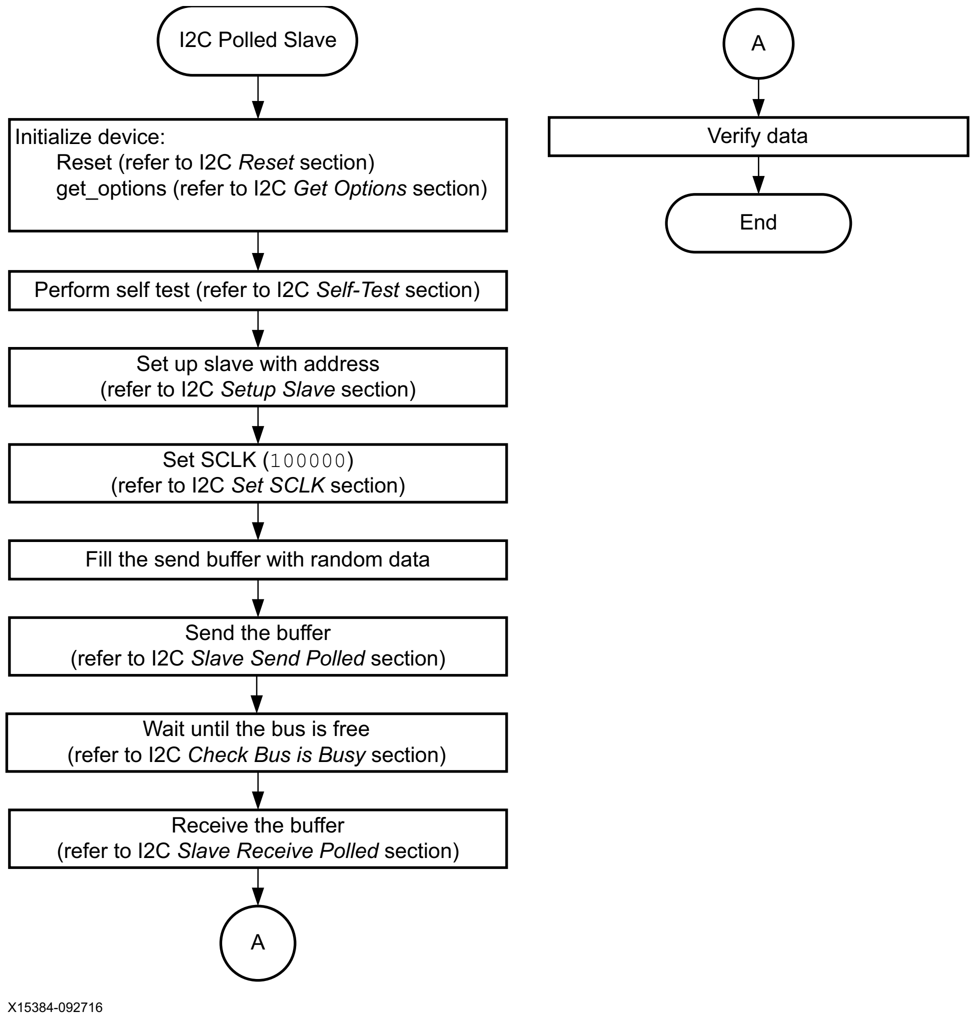

I2C Polled Slave
Initialize device:
Reset (refer to I2C *Reset* section)
get_options (refer to I2C *Get Options* section)
Perform self test (refer to I2C *Self-Test* section)
Set up slave with address (refer to I2C *Setup Slave* section)
Set SCLK (100000) (refer to I2C *Set SCLK* section)
Fill the send buffer with random data
Send the buffer (refer to I2C *Slave Send Polled* section)
Wait until the bus is free (refer to I2C *Check Bus is Busy* section)
Receive the buffer (refer to I2C *Slave Receive Polled* section)
A
A
Verify data
End
X15384-092716

Figure 22-4: **I2C Slave Polled Example Flowchart**

I2C Controller Programming Steps

The programming steps for the I2C controller are listed in Table 22-3 through Table 22-28.

Table 22-3: **I2C Reset**

Task	Register	Register Field	Bits	Notes
Abort start				
Save interrupt mask register	IMR, 0x20	All	9:0	Read operation
Disable interrupts	IDR, 0x28	All	9:0	Write `2FFh`
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`

Table 22-4: **I2C Get Options**

Task	Register	Register Field	Bits	Notes
Read control register	Control, 0x00	All	15:0	Read operation

Table 22-5: **I2C Check Bus is Busy**

Task	Register	Register Field	Bits	Notes
Read bus active state	Status, 0x04	BA	8	Read operation
If set bus is busy, else bus is free				

Table 22-6: **I2C Transmit FIFO Fill**

Task	Register	Register Field	Bits	Notes
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. Refer to I2C Send Byte.				

Table 22-7: **I2C Send Byte**

Task	Register	Register Field	Bits	Notes
Write byte into data register	Data, 0x0C	DATA	7:0	Write data

Table 22-8: **I2C Reset Hardware**

Task	Register	Register Field	Bits	Notes
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`

Table 22-9: **I2C Setup Master**

Task	Register	Register Field	Bits	Notes
Read control register	Control, 0x00	All	15:0	Read operation
If [HOLD] is set = 1, then check if bus is busy (refer to I2C 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`
For transmitter role				
Enable master transmitter	Control, 0x00	RW	0	`0`
Disable all interrupts	IDR, 0x28	All	9:0	`2FFh`

Table 22-10: **I2C Master Send**

Task	Register	Register Field	Bits	Notes
Set repeated start if data is more than FIFO depth				
Set hold bit	Control, 0x00	HOLD	4	`1`
Setup master for transmitter role (refer to I2C Setup Master)				
Transmit FIFO full (refer to I2C Transmit FIFO Fill)				
Program transfer address	Address, 0x08	ADD	9:0	Address
Enable interrupts	IER, 0x24	ARB_LOST, NACK, COMP	9, 2, and 0	`205h`

Table 22-11: **I2C Master Receive**

Task	Register	Register Field	Bits	Notes
Set repeated start if data is more than FIFO depth				
Set hold bit	Control, 0x00	HOLD	4	`1`
Setup master for receiver role (refer to I2C Setup Master)				
Program transfer address	Address, 0x08	ADD	9:0	Write address
Setup transfer size	Transfer_Size, 0x14	Transfer_Size	7:0	Required transfer size
Enable interrupts	IER, 0x24	ARB_LOST, RX_OVF, NACK, COMP	9, 5, 2, 1 and 0	`227h`

Table 22-12: **I2C Master Send Polled**

Task	Register	Register Field	Bits	Notes
Set repeated start if data is more than FIFO DEPTH				
Set hold bit	Control, 0x00	HOLD	4	1
Setup master for transmitter role (refer to I2C 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 (refer to I2C 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 (refer to I2C 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

Table 22-13: **I2C Master Receive Polled**

Task	Register	Register Field	Bits	Notes
Set repeated start if data is more than FIFO DEPTH				
Set hold bit	Control, 0x00	HOLD	4	1
Setup master for receiver role (refer to I2C 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 from hardware reported				
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				
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				

Table 22-14: **I2C Enable Slave Monitor**

Task	Register	Register Field	Bits	Notes
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

Table 22-15: **I2C Disable Slave Monitor**

Task	Register	Register Field	Bits	Notes
Disable slave monitor mode	Control, 0x00	SLVMON	5	0
Disable slave monitor interrupt	IER, 0x24	SLV_RDY	4	0

Table 22-16: **I2C Master Send Data**

Task	Register	Register Field	Bits	Notes
Transmit first FIFO full of data (refer to I2C Transmit FIFO Fill)				
Set repeated start bit if requested	Control, 0x00	HOLD	4	1

Table 22-17: **I2C Master Interrupt Handler**

Task	Register	Register Field	Bits	Notes
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 (refer to I2C 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 22-17: **I2C Master Interrupt Handler *(Cont'd)***

Task	Register	Register Field	Bits	Notes
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`

Table 22-18: **I2C Setup Slave**

Task	Register	Register Field	Bits	Notes
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	Intrpt_disable_reg0	All	9:0	`2FFh`
Transfer address	Address, 0x08	ADD	9:0	Address

Table 22-19: **I2C Slave Send**

Task	Register	Register Field	Bits	Notes
Enable interrupts	IER, 0x24	TX_OVF, TO, NACK, DATA, COMP	6, 3, 2, 1, and 0	`4Fh`

Table 22-20: **I2C Slave Receive**

Task	Register	Register Field	Bits	Notes
Enable interrupts	IER, 0x24	RX_UNF, RX_OVF, TO, NACK, DATA, COMP	7, 5, 3, 2, 1, and 0	`AFh`

Table 22-21: **I2C Slave Send Polled**

Task	Register	Register Field	Bits	Notes
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 (refer to I2C 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

Table 22-22: **I2C Slave Receive Polled**

Task	Register	Register Field	Bits	Notes
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
Read status register	Status, 0x04	All	8:0	Read operation

Table 22-22: **I2C Slave Receive Polled *(Cont'd)***

Task	Register	Register Field	Bits	Notes
Loop-3 ended				
Loop-1 ended				

Table 22-23: **I2C Receive Data**

Task	Register	Register Field	Bits	Notes
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				

Table 22-24: **I2C Slave Interrupt Handler**

Task	Register	Register Field	Bits	Notes
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) • This means master wants to do more data transfers. • Also check for completion of transfer, signal upper layer if done				
For sending transmit FIFO fill (refer to I2C Transmit FIFO Fill)				
Else receive slave data (refer to I2C Slave Receive data)				

Table 22-25: **I2C Set and Clear Options**

Task	Control Register Field (offset 0x00)	Bits	Set Options
For 7-bit address option	NEA	2	1
For 10-bit address option	NEA	2	0
Slave monitor option	SLVMON	5	1
For repeated start option	HOLD	4	1
			Clear Options
For 7-bit address option	NEA	2	0
For 10-bit address option	NEA	2	1
Slave monitor option	SLVMON	5	0
For repeated start option	HOLD	4	0

Table 22-26: **I2C Set SCLK**

Task	Register	Register Field	Bits	Notes
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 with in range FSCL >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

Table 22-27: **I2C Get CLK**

Task	Register	Register Field	Bits	Notes
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)).				

Note: The actual frequency of SCL output is exactly 22 times slower than the frequency of clock_enable. The frequency of clock_enable signal is defined by the frequency of pclk input and the values of divisor_a and divisor_b.

Table 22-28: **I2C Self-Test**

Task	Register	Register Field	Bits	Notes
All the I2C registers should be in their default state.				
Read control register (CR)	Control, 0x00	All	15:0	Read operation
Read interrupt mask register (imr)	IMR, 0x20	All	9:0	Read operation
If (CR != `0`) OR if (imr != `0x2FF`) stop here.				
Perform reset (refer to I2C 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 written value. If not same test failed. Else passed.				
Reset slave monitor register	Slave_Mon_Pause, 0x18	Pause	3:0	`0h`

SPI Controller

Introduction

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, slave mode, or multi-master mode. There are two instances of an SPI controller. Both the controllers are identical and independently controlled by software drivers. They can be operated simultaneously. The following discussions are applicable to both instances of the controller.

Features

- Full-duplex operation offers simultaneous receive and transmit.
- Master or slave SPI modes of operation.
- Four wire bus: data RX, data TX, clock, and select.
- Supports multi-master environment: Identifies an error condition if more than one master detected.
- Memory-mapped APB interface.
- Buffered operation with separate transmit and receive FIFOs: The APB can read from the RXFIFO and write to the TXFIFO.
- In master mode, the SPI clock can be generated from one of three separate clock sources.
- Programmable master-mode clock frequencies.
- Serial clock with programmable polarity.
- Programmable transmission format.
- FIFO levels available through DUT outputs, or through software accessible registers.
- FIFO level status can be polled by software or can be interrupt driven.
- Programmable interrupt generation.

Functional Description

Figure 23-1 shows the SPI controller block diagram.

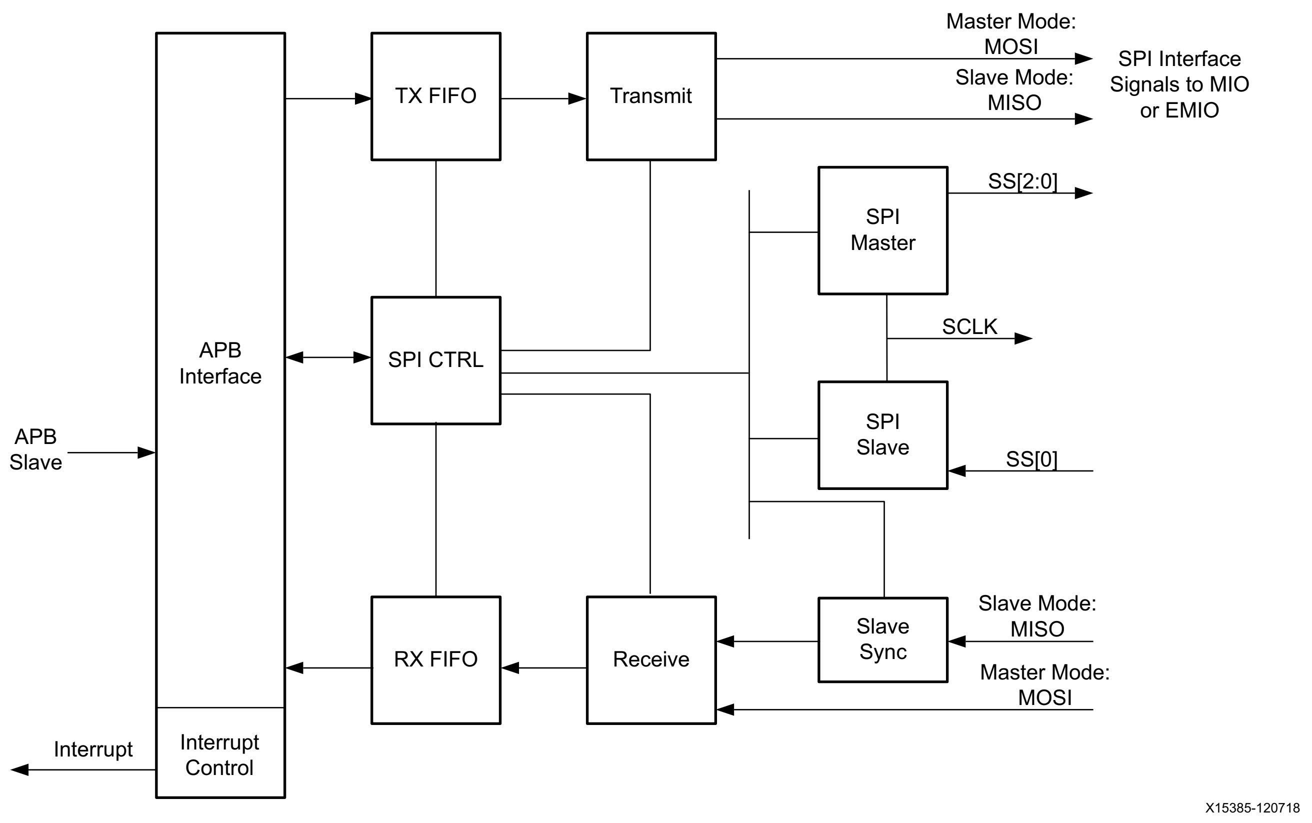


Figure 23-1: **SPI Controller Block Diagram**

FIFOs

The RX and TX FIFOs are each 128-bytes deep. Software reads and writes these FIFOs using the register mapped data-port registers. The FIFOs bridge two clock domains; the APB interface (LPD_LSBUS_CLK) and the controller's SPI_REF_CLK. Software writes to the TXFIFO in the APB clock domain and the controller reads the TXFIFO in the SPI_Ref_Clk domain. The controller fills the RXFIFO in the SPI_Ref_Clk domain and software reads the RXFIFO in the APB clock domain.

RXFIFO

If the controller attempts to push data into a full RXFIFO, then the content is lost and the sticky overflow flag is set. No data is added to a full RXFIFO. Software writes a 1 to the bit to clear the [RX_OVERFLOW] bit.

TXFIFO

If software attempts to write data into a full TXFIFO, then the write is ignored. No data is added to a full TXFIFO. The [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`.

Clocks

The SPI controller receives two clock inputs from the PS clock subsystem and, in slave mode, the SCLK clock from the attached SPI master.

- SPI_REF_CLK clock operates the controller and the baud-rate divider for the SCLK in master mode.
- LPD_LSBUS_CLK clock operates the APB slave interface for register access. Refer to Answer Record 73356.

These clocks run asynchronous to each other. Clock generation is described in Chapter 37, PS Clock Subsystem. The clock frequency specifications are defined in the data sheet.

Master Mode SCLK

In master mode, the I/O signals are clocked by the controller generated SCLK that is derived from the SPI_REF_CLK using the baud-rate divider 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,... 256). The slave select input pin must be driven synchronously with respect to the SCLK input.

Slave Mode SCLK

In slave mode, the controller samples the MOSI and SS I/O signals and drives the MISO signal using the SCLK from the attached master. The input signals are synchronized to the SPI_REF_CLK and processed by the controller.

Note: The SPI_REF_CLK frequency should be at least 2x the SCLK frequency for the controller to properly detect the start of the word transfer on the SPI bus.

Resets

The controller is reset by the PS reset subsystem individually or by a system or POR reset. See Chapter 38, Reset System for more information.

SPI Controller Modes of Operation

The SPI controller operates in three modes:

- Master mode
- Multi-master mode
- Slave mode

In multi-master mode, the controller's output signals are 3-stated when the controller is not active and can detect contention errors when enabled. The outputs are 3-stated immediately by resetting the SPI enable bit. An interrupt status register indicates a mode fault.

In slave mode, the controller receives the serial clock from the master device and uses the SPI_REF_CLK to synchronize data capture. 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.

Master Mode

In master mode, the SPI I/O interface can transmit data to a slave or initiate a transfer to receive data from a slave. In this mode, the controller drives the serial clock and slave selects with an option to support the SPI's multi-master mode. The serial clock is derived from the PS clock subsystem.

The controller 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, it is possible to add a 3-to-8 decoder 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 Mode

For multi-master mode, 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 done, the software disables the controller. The controller cannot be selected by an external master when the controller is in master mode.

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, it sets the spi.ISR [MODE_FAIL] interrupt status bit and clears the spi.Enable [SPI_EN] control bit. The software can receive the [MODE_FAIL] interrupt so it can abort the transfer, reset the controller, and re-send the transfer.

SPI Data Transfers

The SPI controller follows a specific series of operations to initiate and control the data transfers on the SPI bus. The following subsections detail 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. In addition, the slave select assertion/deassertion can be done by the controller hardware or from 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 spi.Config [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

The following procedure 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 spi.Config [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 [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.

IMPORTANT: *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 any of the following design selections.*

- Ensure that the external master does not initiate data transfer until at least ten SPI_REF_CLK cycles are complete after slave mode is enabled.
- Ensure that slave mode is enabled before the master is enabled.
- Ensure that the slave select input signal is not active when the slave is enabled.

MIO-EMIO Signals

MIO Signals

The SPI I/O interface signals available on the MIO interface are listed in Table 23-1. If the I/O signals are not routed to a set of MIO pins (MIO_PIN_xx register programming), then the EMIO interface input signals are enabled.

Table 23-1: **SPI MIO Pins**

SPI Interface I/O	All Modes			Slave or Master Mode	Master Mode	
	Clock	MOSI	MISO	SS 0	SS 1	SS 2
Signal Type	I/O	I/O	I/O	I/O	O	O
Index[1]	5	0	1	2	3	4
Controller Default Input Value	0	0	0	1	-	-
SPI 0, choice 1	0	5	4	3	2	1
SPI 0, choice 2	12	17	16	15	14	13
SPI 0, choice 3	26	31	30	29	28	27
SPI 0, choice 4	38	43	42	41	40	39
SPI 0, choice 5	52	57	56	55	54	53
SPI 0, choice 6	64	69	68	67	66	65
SPI 1, choice 1	6	11	10	9	8	7
SPI 1, choice 2	22	23	18	21	20	19

Table 23-1: **SPI MIO Pins *(Cont'd)***

SPI Interface I/O	All Modes			Slave or Master Mode	Master Mode	
	Clock	MOSI	MISO	SS 0	SS 1	SS 2
SPI 1, choice 3	32	37	36	35	34	33
SPI 1, choice 4	44	49	48	47	46	45
SPI 1, choice 5	58	63	62	61	60	59
SPI 1, choice 6	70	75	74	73	72	71

Notes:

1. The index numbers are listed in Table 28-1.

EMIO Signals

The SPI I/O interface signals available on the EMIO interface are identified in Table 23-2.

Table 23-2: **SPI EMIO Signals**

SPI Interface	Default Input	EMIO Signals		
		Input Name (I)	Output Name (O)	3-state Name (O)
SPI 0 Clock	0	spi0_sclk_i	spi0_sclk_o	spi0_sclk_t
SPI 0 MOSI	0	spi0_s_i	spi0_m_o	spi0_mo_t
SPI 0 MISO	0	spi0_m_i	spi0_s_o	spi0_so_t
SPI 0 Slave Select 0	1	spi0_ss_i_n	spi0_ss_o_n	-
SPI 0 Slave Select 1	~	-	spi0_ss1_o_n	-
SPI 0 Slave Select 2	~	-	spi0_ss2_o_n	-
SPI 0 SS 3-state	~	-	-	spi0_ss_n_t
SPI 1 Clock	0	spi1_sclk_i	spi1_sclk_o	spi1_sclk_t
SPI 1 MOSI	0	spi1_s_i	spi1_m_o	spi1_mo_t
SPI 1 MISO	0	spi1_m_i	spi1_s_o	spi1_so_t
SPI 1 Slave Select 0	1	spi1_ss_i_n	spi1_ss_o_n	-
SPI 1 Slave Select 1	~	-	spi1_ss1_o_n	-
SPI 1 Slave Select 2	~	-	spi1_ss2_o_n	-
SPI 1 SS 3-state	~	-	-	spi1_ss_n_t

SPI0-to-SPI1 Loopback Connection

The I/O signals of the two SPI controller in the PS are connected together when the iou_slcr.MIO_LOOPBACK [SPI0_LOOP_SPI1] bit is set = 1. In this mode, 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.

Register Overview

Table 23-3 summarizes the SPI controller registers.

Table 23-3: **SPI Controller Registers**

Type	Register Name	Description
Controller configuration	Config	Configuration
Controller enable	Enable	SPI controller enable
Interrupt	ISR	Interrupt status (RX full, not empty and TX full, not full)
	IER	Interrupt enable
	IDR	Interrupt disable
	IMR	Interrupt mask/enable
FIFO thresholds	TX_thres	TXFIFO threshold level for not full
	RX_thres	RXFIFO Threshold level for not empty
Master mode	Delay	Slave select delays and separation counts in master mode
FIFO data ports	Tx_data	Transmit data (TXFIFO)
	Rx_data	Receive data (RXFIFO)
Slave mode	Slave_Idle_count	Slave idle count detects inactive SCLK

Programming Model

The flow diagram for the SPI programming sequence is shown in Figure 23-2.

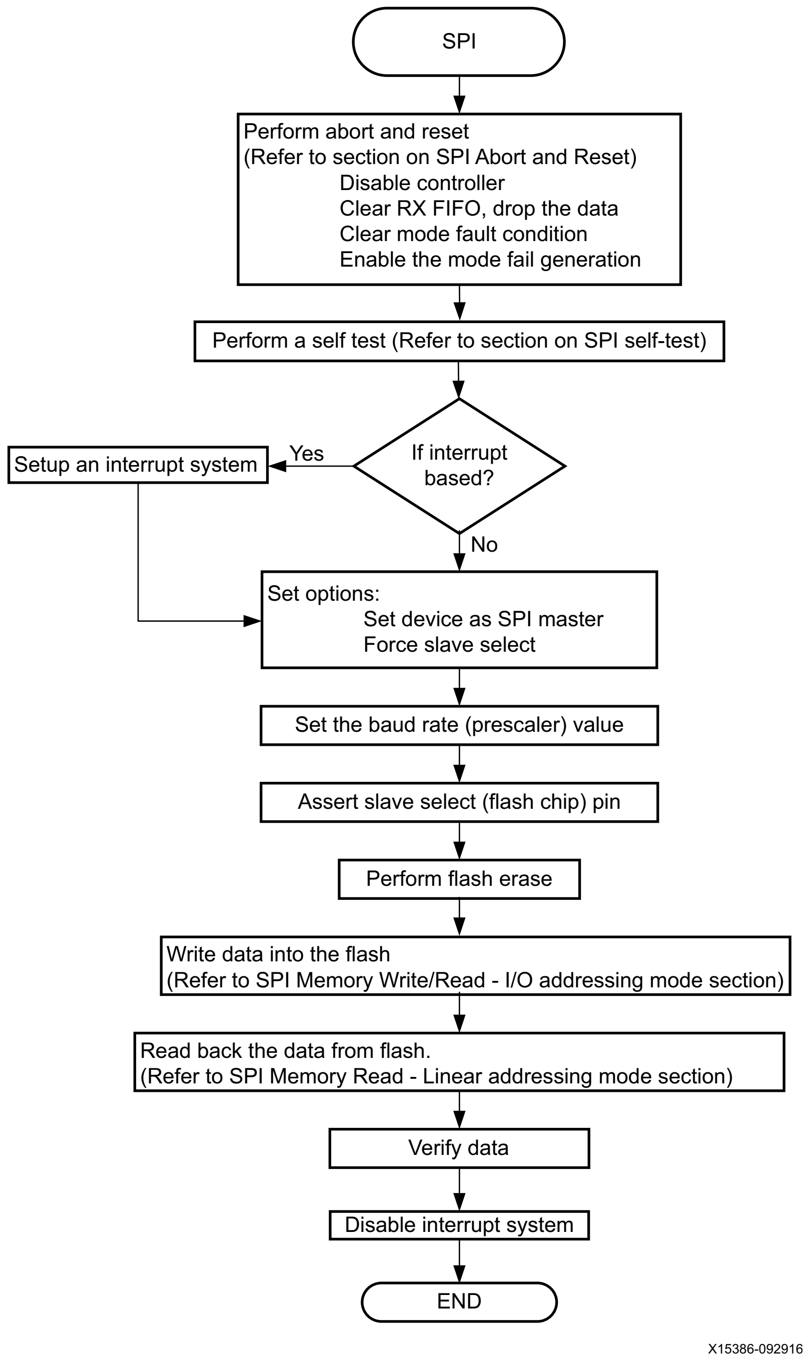

SPI
Perform abort and reset
(Refer to section on SPI Abort and Reset)
Disable controller
Clear RX FIFO, drop the data
Clear mode fault condition
Enable the mode fail generation
Perform a self test (Refer to section on SPI self-test)
If interrupt based?
Yes
Setup an interrupt system
No
Set options:
Set device as SPI master
Force slave select
Set the baud rate (prescaler) value
Assert slave select (flash chip) pin
Perform flash erase
Write data into the flash
(Refer to SPI Memory Write/Read - I/O addressing mode section)
Read back the data from flash.
(Refer to SPI Memory Read - Linear addressing mode section)
Verify data
Disable interrupt system
END
X15386-092916

Figure 23-2: **SPI Programming Sequence Flowchart**

The programming steps to perform various operations on the SPI controller are listed in Table 23-4 through Table 23-6.

Table 23-4: **SPI Abort and Reset**

Task	Register	Register Field	Register Offset	Bits	Note
Disable SPI device	Enable	SPI_EN	`0x14`	0	`0`
Check RX FIFO empty	ISR	RX_FIFO_not_empty	`0x04`	4	Read
Read RX buffer	Rx_data	RX_FIFO_data	`0x20`	31:0	Read
Clear mode fault	ISR	MODE_FAIL	`0x04`	1	`1`
Enable mode fail generation	Config	Modefail_gen_en	`0x00`	17	`1`

Table 23-5: **SPI Self Test**

Task	Register	Register Field	Register Offset	Bits	Note
Abort and reset	Refer to SPI Abort and Reset.				
Check SPI registers default state	Config	Modefail_gen_en	`0x00`	17	Read
TX FIFO reset state check	ISR	TX_FIFO_not_full	`0x04`	2	Read operation
Writing known values	Delay	d_nss, d_btwn, d_after, d_int	`0x18`	31:24, 23:16, 15:8, and 7:0	Register write `5AA5_AA55h`
Read back delay register and verify	Delay	d_nss, d_btwn, d_after, d_int	`0x18`	31:24, 23:16, 15:8, and 7:0	Read operation
Reset delay register	Delay	d_nss, d_btwn, d_after, d_int	`0x18`	31:24, 23:16, 15:8, and 7:0	Register write `0h`
Abort and reset	Refer to SPI Abort and Reset.				

Table 23-6: **SPI Setup Interrupt System**

Task
Initialize GIC.
Register GIC interrupt handler.
Register SPI interrupt handler with the GIC.
Enable GIC.
Enable processor interrupts.

Table 23-7: **SPI Set Options**

Task	Register	Register Field	Register Offset	Bits	Note
Selecting Options					
Select master I/O mode	Config	MODE_SEL	`0x00`	0	`1`
Select clock polarity, CPOL	Config	CLK_POL	`0x00`	1	`1`
Select clock phase, CPHA	Config	CLK_PH	`0x00`	2	`1`
Select external multiplexer mode	Config	PERI_SEL	`0x00`	9	`0`
Select manual CS mode	Config	Manual_CS	`0x00`	14	`1`
Select manual start	Config	Man_start_en	`0x00`	15	`1`
Read CPOL and CPHA status	Config	CLK_POL, CLK_PH	`0x00`	1, 2	Read
If CPOL and CPHA status is different from requested					
Disable the controller	Enable	SPI_EN	`0x14`	0	`0`
End if					
Write the selected options	Config	MODE_SEL, CLK_POL, CLK_PH,PERI_SEL, CS,Manual_CS Man_start_en	`0x00`	0, 1, 2, 9, 14, and 15	
If CPOL and CPHA status is different from requested					
Enable the device	Enable	SPI_EN	`0x14`	0	`1`
End if					

Notes:

1. See Answer Record 73588 for more information.

Table 23-8: **SPI Flash Erase**

Task
If chip erase needs to be performed, follow these next steps.
Prepare the write buffer with a write enable command (`0x06`).
Perform the transfer (refer to SPI Memory Write/Read Command Issue).
Prepare the write buffer with the read status command (`0x05`).
Perform the transfer (refer to SPI Memory Write/Read Command Issue).
Wait until read status becomes `0x01`.
Prepare the write buffer with a write status command (`0x01`).
Perform the transfer (refer to SPI Memory Write/Read Command Issue).
Prepare the write buffer with the read status command (`0x05`).
Perform the transfer (refer to SPI Memory Write/Read Command Issue).
Wait until read status becomes `0x01`.
Prepare the write buffer with a write enable command (`0x06`).
Perform the transfer (refer to SPI Memory Write/Read Command Issue).
Prepare the write buffer with the read status command (`0x05`).
Perform the transfer (refer to SPI Memory Write/Read Command Issue).
Prepare the write buffer with a chip erase command (`0x60`).
Perform the transfer (refer to SPI Memory Write/Read Command Issue).
Prepare the write buffer with the read status command (`0x05`).
Perform the transfer (refer to SPI Memory Write/Read Command Issue).
Wait until read status becomes `0x01`.

Table 23-9: **SPI Flash Write**

Task
If chip erase needs to be performed, follow these next steps.
Prepare the write buffer with a write enable command (`0x06`).
Perform the transfer (refer to SPI Memory Write/Read Command Issue).
Prepare the write buffer with a write command (`0x02`), address and data pointers.
Perform the transfer (refer to SPI Memory Write/Read Command Issue).
Prepare the write buffer with the read status command (`0x05`).
Perform the transfer (refer to SPI Memory Write/Read Command Issue).
Wait until the read status becomes `0x01`.
Prepare the write buffer with the write disable command (`0x04`).
Perform the transfer (refer to SPI Memory Write/Read Command Issue).
Wait until the read status becomes `0x01`.
Perform the previous steps until all bytes are transferred.

Table 23-10: **SPI Flash Read**

Task
If flash read needs to be performed, follow these next steps.
Fill the write buffer with a read command (`0x03`) and an address to be read.
Perform the transfer (refer to SPI Memory Write/Read Command Issue).
Wait until all bytes are received.

Table 23-11: **SPI Set Slave Select**

Task	Register	Register Field	Register Offset	Bits	Note
Read the status of Decode slave select	Config	PERI_SEL	`0x00`	9	Read operation
If decode slave select is set					
Set the slave select value	Config	CS	`0x00`	13:10	`001`
Else					
Clear the slave select value	Config	CS	`0x00`	13:10	`0`

Table 23-12: **SPI Memory Write/Read Command Issue**

Task	Register	Register Field	Register Offset	Bits	Note
If manual slave select is configured					
Initialize slave select value (0, 1, 2)	Config	CS	`0x00`	13:10	`0`
End if					
Enable the device	Enable	SPI_EN	`0x14`	0	`1`
Clear all the interrupts	ISR	RX_OVERFLOW, MODE_FAIL, RX_FIFO_not_empty, RX_FIFO_full	`0x04`	0, 1, 4, and 5	Register write `43h`
Send data	TXD	TXD	`0x1C`	31:0	data
Perform the previous steps until all of the bytes are transferred					
Enable interrupts	IER	RX_OVERFLOW, MODE_FAIL, TX_FIFO_not_full, RX_FIFO_full	`0x08`	0,1,2 and 5	Register write `27h`
Check if master mode enabled	Config	MODE_SEL	`0x00`	0	Read operation
Check if manual start option enabled	Config	Man_start_en	`0x00`	15	Read operation
If both the previous options are set					

Table 23-12: **SPI Memory Write/Read Command Issue *(Cont'd)***

Task	Register	Register Field	Register Offset	Bits	Note
Start transfer manually	Config	Man_start_com	0x00	16	1
Wait until transfer is over					

Table 23-13: **SPI Interrupt Handler**

Task	Register	Register Field	Register Offset	Bits	Note
Read interrupt status	ISR	All	0x04	31:0	Read operation
Clear interrupts	ISR	RX_OVERFLOW, MODE_FAIL, TX_FIFO_underflow	0x04	0, 1, and 6	Register write 43h
Disable TX FIFO full interrupt	Intr_dis_reg	TX_FIFO_full	0x0C	3	1
If a mode fault interrupt occurs					
Perform abort (refer to SPI Abort and Reset)					
Return from interrupt					
If a TX FIFO full interrupt occurs					
Receive data	Rx_data	RXD	0x20	31:0	Read operation
Perform previous step until all bytes are received.					
Fill TX FIFO	Tx_data	TXD	0x1C	7:0	Data
Perform previous steps until all remaining bytes are transferred.					
If all bytes transferred (if 1)					
Disable interrupts	IDR	RX_OVERFLOW, MODE_FAIL, TX_FIFO_not_full, RX_FIFO_full	0x0C	0, 1, 2, and 5	Register write 27h
If manual slave select is configured (if 2)					
Disable slave manually	Config	CS	0x00	10, 11, 12, and 13	1111b
End if (if 2)					
Disable device	Enable	SPI_EN	0x14	0	0
If 1 over					
Else bytes remain to be transferred					
Enable TX FIFO not full interrupt	IER	TX_FIFO_not_full	0x08	2	1
Check if master mode enabled	Config	MODE_SEL	0x00	0	Read operation
Check if manual start option enabled	Config	Man_start_en	0x00	15	Read operation

Table 23-13: **SPI Interrupt Handler *(Cont'd)***

Task	Register	Register Field	Register Offset	Bits	Note
If both the previous options are set					
Start transfer manually	Config	Man_start_com	0x00	16	1
Else over					
End if					
If RX overflow interrupt occurs					
If manual slave select is configured					
Disable slave manually	Config	CS	0x00	10, 11, 12, and 13	1111b
End if					
RX overflow handling is over					
If a TX underflow interrupt occurs					
If a manual slave select is configured					
Disable slave manually	Config	CS	0x00	10, 11, 12, and 13	1111b
End if					
TX underflow handling is over					
Return from interrupt					

Chapter 24

Quad-SPI Controllers

Introduction

The IOP has two Quad-SPI controllers with different functional features and I/O interfacing capabilities. They share the same APB slave interface and I/O signals to the multiplexed I/O (MIO) pins. Only one controller can be enabled at a time. The Quad-SPI controllers access multi-bit flash memory devices for high throughput and low pin-count applications.

The legacy Quad-SPI controller (LQSPI) provides a linear addressable memory space on the Quad-SPI AXI slave interface. It supports execute-in-place (XIP) for booting and application software for some configurations. The generic Quad-SPI controller (GQSPI) provides I/O, DMA, and SPI mode interfacing. Boot and XIP are not supported in the GQSPI. The I/O interface configurations are summarized in Table 24-1 for the legacy and generic Quad-SPI controllers. See Answer Record 65463 for Xilinx tested and supported QSPI devices.

Only one controller can be selected at a time. Switching between controllers is explained in System Control.

Table 24-1: **Quad-SPI I/O Configurations**

I/O Type	Device Count	Slave Selects	Data Signals	Figure	Controller Type	Boot[1] (GQSPI)	XIP[2] (LQSPI)
Single SS 4-bit	1	1	4	Figure 24-5	LQSPI and GQSPI	Yes	Yes
Dual SS stacked	2	2	4	Figure 24-7	LQSPI and GQSPI	Yes[3]	Yes[3]
Dual SS parallel[4]	2	2	8	Figure 24-6	LQSPI and GQSPI	Yes	No

Notes:
1. The CSU BootROM uses the GQSPI controller for system boot. The Width Detection parameter in the boot header selects between 4- and 8-bit I/O. If the XIP FSBL is selected (FSBL length = 0 in the boot header), the BootROM switches to the LQSPI controller before handing-off the system to the FSBL code.
2. XIP requires linear addressing and is only supported with the LQSPI controller.
3. The dual SS stacked configuration supports boot and XIP with the image only in the lower device.
4. The data ordering used by the GQSPI and LQSPI controllers are different for the dual SS parallel configuration 8-bit I/O.

Legacy Quad-SPI Controller Mode

The legacy Quad-SPI controller operates only in linear address modes. The legacy Quad-SPI controller can interface to one or two flash devices. To minimize pin count, two devices can be connected in parallel for 8-bit performance, or in a stacked, 4-bit arrangement.

Linear Address Mode

The linear address mode uses a subset of device operations to eliminate the software overhead to read the flash memory. Linear address mode issues commands to the flash memory and controls the flow of data from the flash memory bus to the AXI interface. The controller responds to memory requests on the AXI interface as if the flash memory were a ROM memory.

Generic Quad-SPI Controller Modes

The generic Quad-SPI controller meets the requirements for generic low-level access by the software. The controller supports generic and future command sequences and future NOR/NAND flash devices. Due to the generic nature of the Quad-SPI controller, software can generate any command sequence in any mode. The Quad-SPI controller supports all features in SPI, dual-SPI, and Quad-SPI modes. The generic Quad-SPI controller operates in three modes, the I/O, DMA, and SPI modes.

The generic Quad-SPI controller can interface to one or two flash devices. To minimize pin count, two devices can be connected in parallel for 8-bit performance, or in a stacked, 4-bit arrangement.

I/O Mode

In generic I/O mode, the software interacts closely with the flash device protocol. The software writes the flash commands in the generic FIFO and data into the TXFIFO. The software reads the RXD register that contains the data received from the flash device. The generic Quad-SPI controller removes the software overhead that occurs when filling the TXFIFO in I/O mode.

DMA Mode

In generic DMA mode, the internal DMA module transfers data from the flash device to the system memory. This mode avoids using the processor for reading the flash data beat-by-beat and removes the software overhead that occurs when the TXFIFO is filled with data read from the flash device.

SPI Mode

In SPI mode, the generic Quad-SPI controller can be used as a standard SPI controller.

Architecture Overview

Figure 24-1 shows the dual Quad-SPI controller. The controller consists of a legacy linear Quad-SPI controller and the generic Quad-SPI controller. Register control (generic_qspi_sel) set to `1` selects the generic Quad-SPI controller. The shaded units in Figure 24-1 are used by both the legacy and generic controllers. The receive capture logic with delay line is shared between both the controllers.

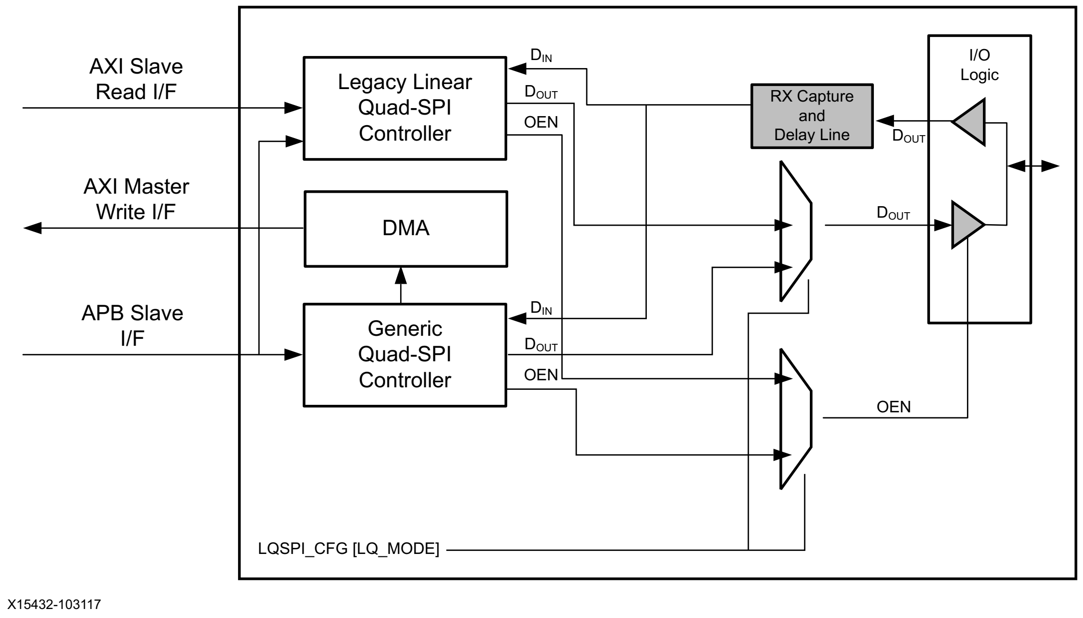


Figure 24-1: **Quad-SPI Dual Controllers Block Diagram**

System Control

The selected controller must be inactive before switching to the other controller, changing clock rates, or reconfiguring the I/O protocol.

Controller Selection

One controller is selected at a time using the LQSPI_CFG [LQ_MODE] bit. The generic controller is selected by setting the bit = 0 and the legacy linear controller is selected by setting the bit = 1. The active controller must be quiescent before switching from one controller to the other.

Legacy Controller to Generic Quad-SPI Controller

1. Wait until all pending transfers from the legacy controller are completed.
2. Disable the legacy linear controller. Set LQSPI_En [SPI_EN] = 0.
3. Configure the generic Quad-SPI controller.
4. Select the generic controller. Set LQSPI_CFG [LQ_MODE] = 0.
5. Enable the generic controller. Set GQSPI_En [GQSPI_EN] =1.
6. Use the generic controller.

Generic Quad-SPI Controller to Legacy Controller

1. Wait until all pending transfers from the generic controller are completed.
2. Disable the generic controller. Set GQSPI_En [GQSPI_EN] = 0.
3. Configure the legacy linear Quad-SPI controller.
4. Select the legacy linear Quad-SPI controller. Set LQSPI_CFG [LQ_MODE] = 1.
5. Enable the legacy linear controller. Set LQSPI_En [SPI_EN] = 1.
6. Use the legacy linear controller.

Clock Polarity, Phase, and Baud Rate Reconfiguration

To reconfigure the clock polarity, phase, or baud-rate divisor values, the controller must be disabled before configuration.

Dynamic Mode and Baud Rate Change Limitations

The generic Quad-SPI controller requires a reset (CRL_APB.RST_LPD_IOU2[qspi_reset]) to simultaneously switch both the baud-rate divisor and the dual-parallel mode. For example, when operating in the single or stacked mode and accessing the lower flash with a baud rate of 4 and switching to the dual-parallel or stacked dual-parallel mode with a baud rate of 2, a reset is required.

Reference Clock Change Limitations

The Quad-SPI controller requires a reset (CRL_APB.RST_LPD_IOU2 [qspi_reset]) when the reference clock is changed.

Clocks and Resets

The Quad-SPI controller core and I/O interface are driven by the reference clock, QSPI_REF_CLK. The controller's master and slave AXI interfaces are driven by the IOU_SWITCH_CLK, and the APB slave programming interface is clocked by the LPD_LSBUS_CLK clock. All of these clocks are generated by the PS clock subsystem. These three clocks run asynchronous to each other.

See the Interconnect Clock Generators section in Chapter 37, PS Clock Subsystem for more information.

Reference Clock and Quad-SPI Interface Clocks

The reference clock is generated by the PS clock subsystem using the circuit shown in Figure 37-4. The input clock source can be selected based on the crl_apb.QSPI_REF_CTRL [srcsel] bits, where the source can be from the RPLL, IOPLL, or DPLL. The crl_apb.QSPI_REF_CTRL [divisor0] register selects the 6-bit programmable divider 0. The crl_apb.QSPI_REF_CTRL [divisor1] register selects the 6-bit programmable divider 1. The crl_apb.QSPI_REF_CTRL [clkact] bit selects whether the clock should be gated or enabled.

To generate the Quad-SPI interface clock, the reference clock is divided down by 2, 4, 8, 16, 32, 64, 128, or 256 using the qspi.Config [BAUD_RATE_DIV] bit field. See Answer Record 69831 for how to generate the QSPI interface clock.

Quad-SPI Feedback Clock

The Quad SPI interface has an optional feedback clock pin named clk_for_lpbk. The clk_for_lpbk pin is not used for loopback mode. The internal clock is used for loopback mode. The loopback mode is used with the high-speed Quad SPI timing mode, where the memory interface clock needs to be greater than 40 MHz. The feedback signal is received from the internal input from the I/O. So, MIO pin 6 should be programmed and allowed to toggle freely.

Based on the tap delay value programmed, the internal clock is delayed and used for capturing the data. See Quad-SPI Tap Delay Values for programming the tap values for different operation frequencies.This pin (MIO 6) is not driven from outside and should be left floating in QSPI clock feedback mode. In QSPI non-clock feedback mode, the pin is not used by the QSPI, so it can be used as a peripheral I/O (GPIO, CAN, I2C, and so on).

Resets

The controller reset bits are generated by the PS. For more information, see Chapter 38, Reset System.

Generic Quad-SPI Controller

Controller Features

- Low-level (generic) access
- 3, 4, 6,...n-byte addresses
- SPI NAND flash devices
- Command queuing (generic FIFO depth is 32)
- 4- or 8-bit interface
- Two chip-select lines
- 4-bit bidirectional I/O signals
- x1, x2, and x4 read/write
- 44-bit address space on AXI in DMA mode
- Byte stripe when two data buses are connected
- Single system interrupt for controller/DMA interrupt status (IRQ 47)
- Single transfer rate (STR) mode
- 64-word RXFIFO, 64-word TXFIFO

Block Diagram

Figure 24-2 shows a block diagram of the generic Quad-SPI controller.

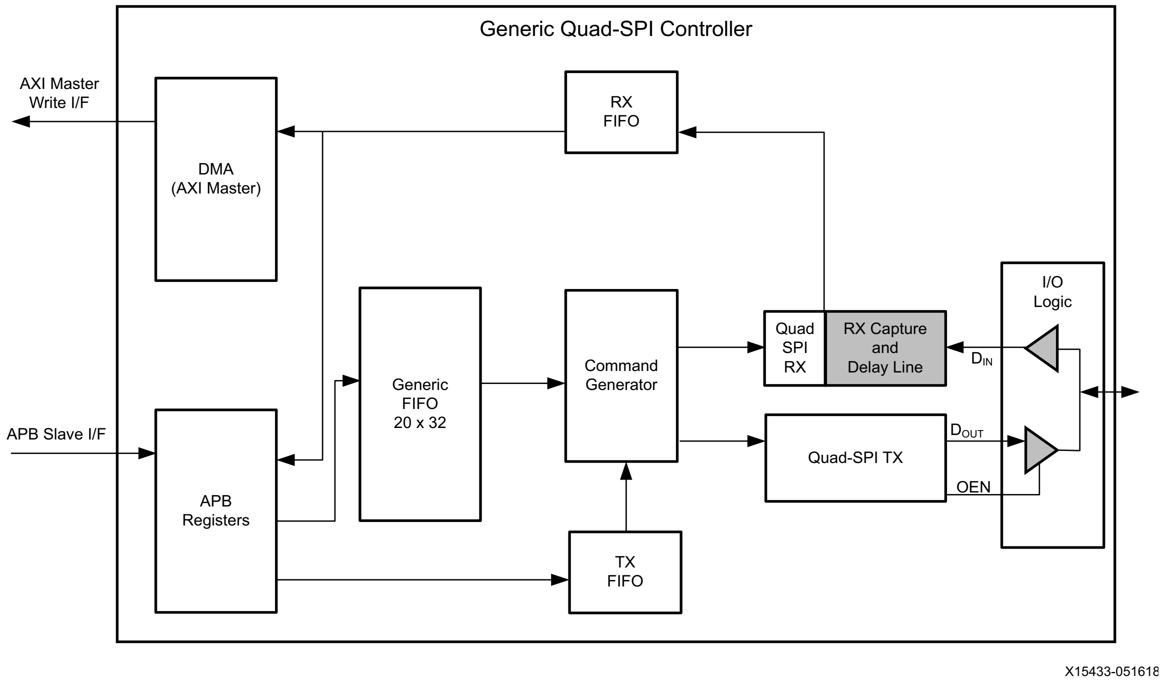


Figure 24-2: **Generic Quad-SPI Controller Block Diagram**

The following interfaces are used by the generic Quad-SPI controller.

- The APB slave read/write interface is used to read/write the registers and also to write the TX FIFO and generic FIFO data.
- The AXI master write interface is used to issue DMA write requests on the AXI interface. The data read from flash memory is written into the RXFIFO. The data from the RXFIFO is transferred into external memory (for example, DDR) using this interface. The AXI address bus is 44 bits wide and the data is 32 bits wide.

DMA–AXI Master

The DMA unit generates AXI requests for the RXFIFO. It is a master on the AXI interface. The DMA module uses the AXI write channel to initiate AXI write requests that write RXFIFO data into the external memory (for example, DDR). This 32-bit AXI master interface allows access to the PS slaves via the top-level interconnect. An APB interface is provided for control and monitoring of the DMA write-channel module's functions. There is a single interrupt output that is sent to the DMA logic where it is combined with other controller interrupt sources to become a single system interrupt. The DMA controller does not support unaligned data transfers. All the data transfers are word aligned.

The DMA memory transactions will be routed to the CCI.

SPI Interface Logic

The SPI interface logic sends the data and commands to the SPI electrical interface. The SPI interface has two sets of chip select, clock (SCLK), and data ports. Depending on the data bus select of the generic FIFO, both data buses/both SCLK signals or a single data bus/single SCLK is activated.

Register Set

The register set contains the control, status, and interrupt registers. These registers are memory mapped with an APB interface.

APB Interface

The 32-bit APB interface accesses the generic FIFO, TXFIFO, RXFIFO, and also the register set.

Command Generator

The SPI Command generator will generate the SPI command after reading the command FIFO and TX Data FIFOs. The command generator module decodes each field of the command FIFO and accordingly generates the relevant SPI request.

RXFIFO

The RXFIFO is used to hold the receive data. The data received from the SPI interface is written into the RXFIFO. When during data transfer the RXFIFO is full, the SCLK is not toggled (the CS remains asserted) and no Quad-SPI data is lost. The Quad-SPI RX block ensures that the RXFIFO does not overflow.

Generic Command FIFO—20-bit Width and 32-bit Depth

The generic FIFO contains information related to the SPI requests. The FIFO is 32 bits deep and 20 bits wide. Details of each field are described in Table 24-2 and Table 24-3. Because the generic FIFO is 32 bits deep, it can hold more than one SPI/flash request. The number of SPI commands that can be issued to the SPI device per request depends on the sequence of SPI device commands needed for that request. The generic FIFO is accessed by the APB interface. Each entry in the generic FIFO requires one APB write request.

Table 24-2: **Generic FIFO Fields**

Reserved	Poll	Stripe	Receive	Transmit	Databus Select	cs_upper	cs_lower	SPI Mode	Exponent	data _xfer	immediate _data
31:20	19	18	17	16	15:14	13	12	11:10	9	8	7:0

Table 24-3: **Generic FIFO Details**

Field	Bits	Description
Reserved	31:20	Reserved.
Poll	19	This bit is applicable when receive is enabled. `1'b0`: Once. `1'b1`: Poll. When set to `1'b1`, the generic Quad-SPI controller keeps reading the data until it matches the received data with the POLL_DATA field data from the poll register, depending on the configured masking in poll register.
Stripe	18	`1'b0`: Do not Stripe. Mirror the same data on the lower and upper data buses. `1'b1`: Stripe data across the lower and upper data buses. Only byte stripe is supported. The lower data bus uses even bytes, i.e., byte 0, 2, 4 ..., of a data word. The upper data bus uses odd bytes, i.e., byte 1, 3, 5, ..., of a data word. Stripe is applicable when the data bus select is `2'b11`, and both upper and lower data buses are active. Stripe = `1'b0` is not applicable when receive = `1'b1`.
Receive	17	`1'b0`: Discard RX data. `1'b1`: Capture/receive data. When [receive, transmit, data_xfer] = [0,0,1], it represents a dummy cycle.
Transmit	16	`1'b0`: Write dummies/Zero pump `1'b1`: Transmit When [receive, transmit, data_xfer] = [0,0,1], it represents a dummy cycle.

Table 24-3: **Generic FIFO Details *(Cont'd)***

Field	Bits	Description
Data bus select	15:14	`2'b00`: No bus. `2'b01`: Lower bus select. `2'b10`: Upper bus select. `2'b11`: Both lower and upper buses. The number of data bus bits depends on the SPI mode. SPI mode: the data bus is 1 bit wide. Dual-SPI mode: the data bus is 2 bits wide. Quad-SPI mode: the data bus is 4 bits wide. The lower clock (QSPI0_SCLK sclk_out) is driven when the lower bus is selected. The upper clock (QSPI1_SCLK) is driven when the upper bus is selected.
cs_upper	13(1)	`1'b0`: Do not drive the upper chip select. `1'b1`: Drive the upper chip select.
cs_lower	12(1)	`1'b0`: Do not drive the lower chip select. `1'b1`: Drive the lower chip select.
SPI mode	11:10	`2'b00`: Reserved. `2'b01`: SPI. `2'b10`: Dual-SPI. `2'b11`: Quad-SPI.
Exponent	9	0: Absolute. 1: Exponent. When data_xfer = 1, this field is used. When data_xfer = 1, and because the immediate_data is 8 bits, the maximum data to be transmitted/received is 2^8 = 256 bits. To transmit/receive more than 256 bits of data, use the exponent bit. For example, when reading 1G bytes from the SPI flash, the generic FIFO fields uses the following contents. data_xfer = `1'b1` exponent = `1'b1` immediate_data = `8'h1E` (decimal 30). The number of data bytes = 2^{30} = 1G bytes.

Notes:

1. When both bits {13:12} are set to 1, it indicates the configuration for the dual parallel mode. Both of these bits are set by the driver.

Table 24-3: **Generic FIFO Details *(Cont'd)***

Field	Bits	Description
data_xfer	8	`1'b0`: The immediate_data is used as immediate. `1'b1`: The immediate_data is used as the number of data bytes to be sent/received.
immediate_data	7:0	When data_xfer = `1'b0`, and transmit is a non zero, these bits are sent on the SPI interface. When data_xfer = `1'b1`, and transmit is non zero, this field specifies the number of data bytes sent from the TXFIFO. When data_xfer = `1'b1`, and receive is non zero, this field specifies the number of data bytes to read into the RXFIFO. The maximum number of data bytes allowed are 2^{28} in DMA and PIO mode. When data_xfer = `1'b1`, RX = `1'b0` and TX = `1'b0`, this field specifies the number of dummy SCLK cycles sent on the SPI interface. When data_xfer = `1'b0`, RX = `1'b0`, data bus select is non zero, TX = `1'b0`, and cs_lower/cs_upper is non zero, this field specifies the CS setup time using the number of reference clock cycles. When data_xfer = `1'b0`, RX = `1'b0`, data bus select is non zero, TX = `1'b0`, and cs_lower/cs_upper is zero, this field specifies the CS hold time using the number of reference clock cycles.

Generic Quad-SPI Commands

Table 24-4 lists the fields used for programming the generic FIFO for different SPI commands when the generic Quad-SPI controller is used for SPI mode commands and only the lower data bus is used. When needed, the other fields (poll, stripe, and exponent) can be programmed.

Table 24-4: **Generic Quad-SPI Controller: SPI Mode Commands**

SPI Command	Receive	Transmit	Data Bus Select	cs_upper	cs_lower	SPI Mode	data _xfer	immediate _data	Description
CS assert	1'b0	1'b0	2'b01	1'b0	1'b1	2'b01	1'b0	8'h04 (CS setup Time)	Assert lower chip select (CS). The immediate_data field specifies the value of the CS setup time (t_{CSS}). Because the value is 8'h04, the CS setup time is four QSPI_REF_CLK cycles. In SPI mode, the receive and transmit are not considered. The data bus select and cs_upper and cs_lower should be valid values.
read	1'b1	1'b0	2'b01	1'b0	1'b1	2'b01	1'b1	8'h34 (52 read data bytes)	Read lower data bus. The immediate_data field specifies the number of data bytes read. Because the value is 8'h34, the number of data bytes received is 52 bytes.
write immediate	1'b0	1'b1	2'b01	1'b0	1'b1	2'b01	1'b0	8'h64	Write lower data bus immediate. The immediate_data field specifies the data to be written. Because the value is 8'h64, the data sent on the SPI is 8'h64.
write	1'b0	1'b1	2'b01	1'b0	1'b1	2'b01	1'b1	8'h64 (100 write data bytes)	Write lower data bus. The immediate_data field specifies the number of write data bytes. Because the value is 8'h64, the number of data bytes written is 100 bytes.

Table 24-4: **Generic Quad-SPI Controller: SPI Mode Commands *(Cont'd)***

SPI Command	Receive	Transmit	Data Bus Select	cs_upper	cs_lower	SPI Mode	data _xfer	immediate _data	Description
read_write immediate	`1'b1`	`1'b1`	`2'b01`	`1'b0`	`1'b1`	`2'b01`	`1'b0`	`8'h64`	Read and write lower data bus immediate. The immediate_data field specifies the data to be written. Because the value is `8'h64`, the data sent on the SPI is `8'h64`. When receive is set to `1`, one data byte is received and stored in the RXFIFO.
read_write	`1'b1`	`1'b1`	`2'b01`	`1'b0`	`1'b1`	`2'b01`	`1'b1`	`8'h64` (100 read and write data bytes)	The read and write lower data bus. The immediate_data field specifies the number of write and read data bytes. Because the value is `8'h64`, the number of data bytes received is 100 bytes and transmitted is also 100 bytes.

Table 24-4: **Generic Quad-SPI Controller: SPI Mode Commands *(Cont'd)***

SPI Command	Receive	Transmit	Data Bus Select	cs_upper	cs_lower	SPI Mode	data _xfer	immediate _data	Description
dummy	`1'b0`	`1'b0`	`2'b01`	`1'b0`	`1'b1`	`2'b01`	`1'b1`	`8'h06` (six dummy SCLK cycles)	Inserts dummy cycles on lower data bus. The immediate_data field specifies the number of dummy SCLK cycles. Because the value is `8'h06`, the number of dummy cycles is six. Data bus select, data_xfer and cs_upper, and cs_lower should be valid values. The data bus select value during the dummy phase should be same as the data phase.
CS deassert	`1'b0`	`1'b0`	`2'b01`	`1'b0`	`1'b0`	`2'b01`	`1'b0`	`8'h06`	Deassert the lower chip select. The immediate_data field specifies the value of the chip select hold time (t_{CSH}). Because the value is `8'h06`, the chip select hold time is five reference clock cycles (one less than the specified value (6-1 = 5)). SPI mode, receive, and transmit are not considered. The data bus select should be valid. The minimum immediate_data field should be four.

Generic Controller I/O Wiring Diagrams

There are four possible I/O wiring diagrams for the generic controller as listed in Table 24-1:

- Single slave select, 4-bit (Figure 24-5).
- Dual slave select, stacked (Figure 24-7).
- Dual slave select, parallel (Figure 24-6).

Legacy Quad-SPI Controller

The legacy Quad-SPI controller (LQSPI) provides a linear addressable memory space on the AXI slave interface.

Features

- 32-bit AXI interface for linear address mode transfers.
- Programmable bus protocol for flash memories from Micron and Spansion.
- Scalable performance: 1x, 2x, 4x, and 8x I/O widths.
- Flexible I/O:
  - Single slave select 4-bit I/O flash interface mode.
  - Dual slave select 8-bit parallel I/O flash interface mode.
  - Dual slave select 4-bit stacked I/O flash interface mode.
- Supports up to 512 MB addresses.
- 63-word RXFIFO, 63-word TXFIFO.
- Linear address mode (executable read accesses):
  - Memory reads and writes are interpreted by the controller.
  - AXI port buffers up to four read requests.
  - AXI incrementing and wrapping address functions.
- Auto filling of TXFIFO with zeros.
- Single transfer rate (STR) mode.

System-level View

The Quad-SPI flash controller is part of the I/O peripheral (IOP) and connects to external SPI flash memory through the multiplexed I/O (MIO) as shown in Figure 24-3. The controller supports one or two memories.

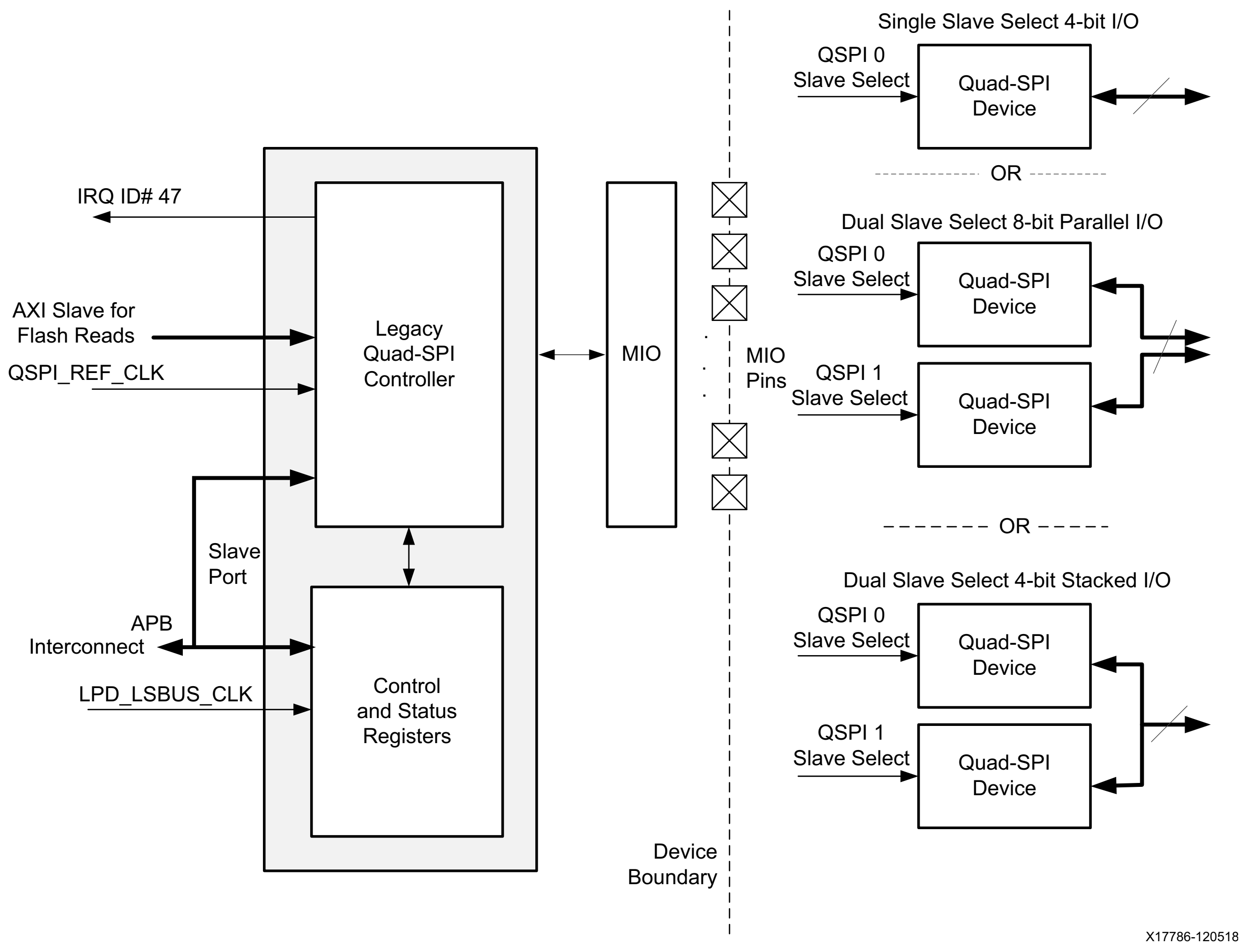


Figure 24-3: **Legacy Quad-SPI Controller System-level View**

Address Map and Device Matching For Linear Address Mode

When a single device is used, the address map for direct memory reads starts from `0xC000_0000` and increases to a maximum of `0xCFFF_FFFF` (256 MB). The address map for a two-device system depends on the memory device and I/O configuration. In a two-device system, the Quad-SPI devices must be from the same vendor and have the same protocol.

The 8-bit parallel I/O configuration also requires that the devices have the same capacity. The address map for the parallel I/O configuration starts from `0xC000_0000` and increases to the address of the combined memory capacities, up to a maximum of `0xDFFF_FFFF` (512 MB).

In the 4-bit stacked I/O configuration, the devices can have different capacities, but must have the same protocol. When two different size devices used a 2048 Mb device on the lower address. In this mode, the Quad-SPI 0 device starts at `0xC000_0000` and increases to a maximum of `0xCFFF_FFFF` (256 MB). The Quad-SPI 1 device starts at `0xD000_0000` and increases to a maximum of `0xDFFF_FFFF` (another 256 MB). If the first device is smaller than 256 MB, then there will be a memory space hole between the two devices.

Figure 24-4 shows a block diagram of a linear address mode.

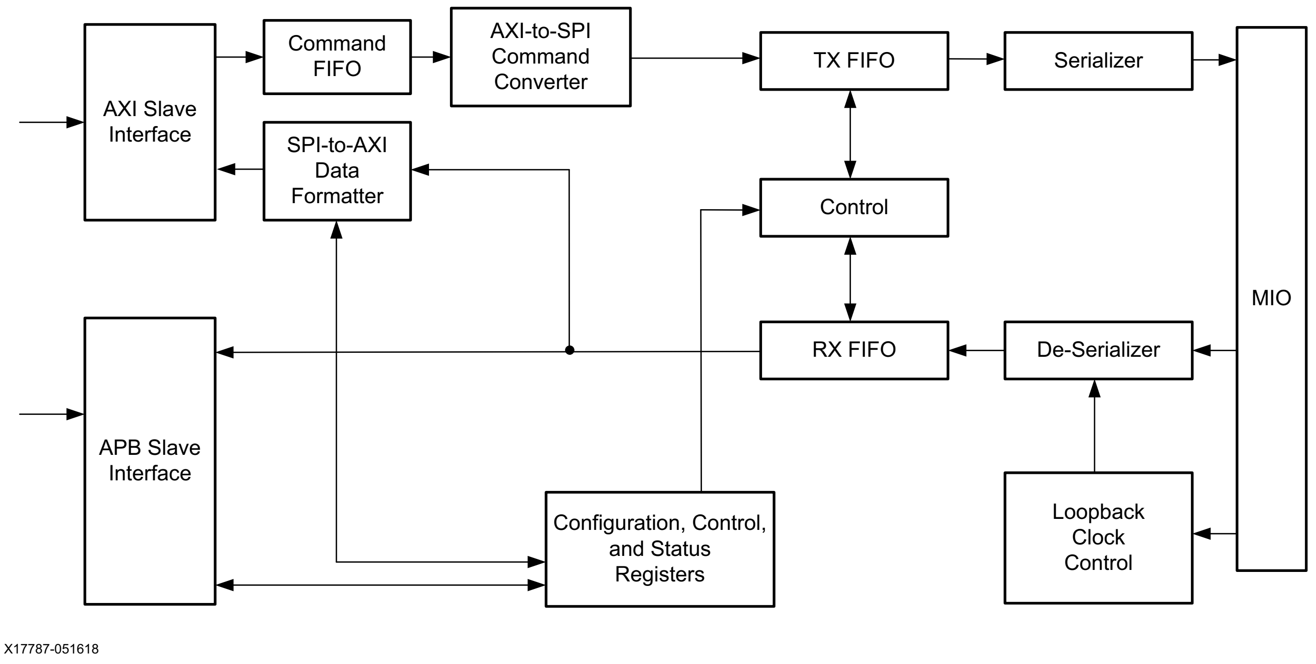


Figure 24-4: **Legacy Linear Controller Block Diagram**

Legacy Quad-SPI Operating Restrictions

When a single device is used, it must be connected to QSPI 0. When two devices are used, both devices must be identical (same vendor and same protocol sequencing).

Legacy Quad-SPI Functional Description

The legacy Quad-SPI flash controller can only operate in linear address mode. For reads, the controller supports single, dual, and quad modes in linear address mode.

Legacy Quad-SPI Linear Address Mode

The controller has a 32-bit AXI slave interface to support linear address mapping for read operations. When a master issues an AXI read command through this port, the Quad-SPI controller generates commands to load the corresponding memory data and send it back through the AXI interface.

In linear address mode, the flash memory subsystem is similar to a typical read-only memory with an AXI interface that supports a command pipeline depth of four. By reducing the amount of software overhead, the linear address mode improves the overall throughput of the read memory. From a software perspective, there is no perceived difference between accessing the legacy Quad-SPI memory subsystem and that of other ROMs, except for the potentially longer latency.

A transfer to linear address mode occurs when the qspi.LQSPI_CFG [LQ_MODE] bit is set to 1. Before entering into linear address mode, both the TXFIFO and RXFIFO must be empty. Once the qspi.LQSPI_CFG [LQ_MODE] bit is set, the FIFOs automatically control the legacy Quad-SPI module and I/O access to TXD and RXD are undefined.

In linear address mode, the CS pins are automatically controlled by the QSPI controller. Before a transition into legacy Quad-SPI linear address mode, both of the qspi.Config [Man_start_en] and qspi.Config[PCS] must be zero.

A simplified block diagram of the controller showing the linear and I/O portions is shown in Figure 24-4.

Linear Address Mode AXI Interface Operation

Only AXI read commands are supported by the linear address mode. All valid write addresses and write data are acknowledged immediately but are ignored, that is, no corresponding programming (write) of the flash memory is carried out. All AXI writes generate an external AXI slave error (SLVERR) on the write response channel.

Both increment or wrapping address burst reads are supported. Fixed address bursts are not supported and cause an SLVERR response. Therefore, the only recognized ARBURST[1:0] value is either `2'b01` or `2'b10`. All read accesses must be word-aligned and the data width must be 32 bits (no narrow burst transfers are allowed).

Table 24-5 lists the read address channel signals from a master that are ignored by the interface.

Table 24-5: **Ignored AXI Read Address Channel Signals**

Signal	Value
ARADDR[1:0]	Ignored, assumed to be 0 (always assumed to be word aligned).
ARSIZE[2:0]	Ignored, always a 32-bit interface.
ARLOCK[1:0]	Ignored
ARCACHE[3:0]	Ignored
ARPROT[2:0]	Ignored

The AXI slave interface provides a read acceptance capability of four to accept up to four outstanding AXI read commands.

Legacy Quad-SPI AXI Read Command Processing

AXI read-burst commands are translated into SPI flash read instructions that are sent to the Quad-SPI controller TXFIFO. The controller transmit logic retrieves the read instruction from the TXFIFO and passes them to the SPI flash memory device according to the SPI protocol.

A 64-deep FIFO is used to provide read data buffering to hold up to four burst of 16 data. Since the RXFIFO starts receiving data as soon as the chip-select signal is active, the linear address module removes any incoming data that corresponds to the instruction code, the address, and the dummy cycles, and responses to the AXI read instruction with valid data.

Legacy Quad-SPI AXI Interface Configuration and Read Modes

AXI read-burst transfers are translated into SPI flash read instructions that are sent to the Quad-SPI controller TXFIFO. The controller transmit logic retrieves the read instructions from the TXFIFO and passes them to the SPI flash memory device according to the SPI protocol.

The SPI read command is used in linear address mode by writing to the qspi.LQSPI_CFG [INST_CODE]. The supported read command codes and the recommended configuration register settings (qspi.LQSPI_CFG) are listed in Table 24-6. The optimal register values for Quad-SPI boot performance using a 33 MHz PS_REF_CLK are shown in Table 24-6. These Quad-SPI registers can be programmed in non-secure mode using the register initialization feature in the BootROM header which to speeds the loading of the FSBL/user code. A faster PS_REF_CLK requires adjusting the clock dividers.

The choice of operating mode depends on the capabilities of the Quad-SPI device. For the fastest performance, the I/O fast read modes use 4-bit parallel transfers for address and data. The quad output fast read uses 4-bit parallel transfers for data only. These are still faster than a serial-bit mode.

Table 24-6: **Quad-SPI Device Configuration Register Values**

Instruction Code	LQSPI_CFG	LQSPI_CFG	COMMAND Register	
	Single	Dual	Micron	Winbond/Spansion
03h	0x80000203	0xe0000203	0x00002000	0x00002000
0Bh	0x8000020b	0xe000020b	0x00002820	0x00002820
3Bh	0x8000023b	0xe000023b	0x00002820	0x00002820
6Bh	0x8000026b	0xe000026b	0x00002820	0x00002820
BBh	0x800002bb	0xe00002bb	0x00001c20	0x00001810
EBh	0x800002eb	0xe00002eb	0x00001828	0x00001418
13h	0x88000213	0xe8000213	0x00002800	0x00002800
0Ch	0x8800020c	0xe800020c	0x00003020	0x00003020
3Ch	0x8800023c	0xe800023c	0x00003020	0x00003020
6Ch	0x8800026c	0xe800026c	0x00003020	0x00003020
BCh	0x880002bc	0xe80002bc	0x00002020	0x00001C10
ECh	0x880002ec	0xe80002ec	0x00001a28	0x00001618

Legacy Quad-SPI Controller Unsupported Devices

There are devices that implement custom 4-bit wide SPI-like interfaces for flash memory access. Other Quad-SPI devices offer an option to switch operation to a custom 4-bit interface, through a non-volatile configuration bit. These interfaces operate differently from the devices supported by the dual controller. These flash memory devices operate in 4-bit mode during the instruction phase, as well as the address and data phases. This requires the Quad-SPI flash controller to power up in 4-bit mode and remain in that mode permanently (or until otherwise configured, if the option is available). The dual controller does not offer support for these custom interfaces.

4-byte Address Support

The 4-byte address commands supported by the legacy Quad-SPI controller are listed in Table 24-7. The legacy Quad-SPI controller supports the following 4-byte address commands. The number of data lanes (DQ pins) for sending the instruction, opcode, and data and receiving the data are listed in table. The legacy Quad-SPI controller does not support instructions where the flash device requires using a different number of data lanes for the same instruction code.

Table 24-7: **4-byte Address Support**

Command Number	Instruction Code	Address Bytes	Opcode Lanes	Address Lanes	Data Lanes	Command Type
1	`8'h13`	4 bytes	1	1	1	4-byte address read. Single lane for opcode, address, and data.
2	`8'h3C`	4 bytes	1	1	2	4-byte address fast read, dual output. Single lane for opcode, address, and two lanes for data.
3	`8'h6C`	4 bytes	1	1	4	4-byte address fast read, quad output. Single lane for opcode, address, and four lanes for data.
4	`8'hBC`	4 bytes	1	2	2	4-byte address fast read, dual I/O. Two lanes for opcode, address, and two lanes for data.
5	`8'hEC`	4 bytes	1	4	4	4-byte address fast read, quad I/O. Single lane for opcode, four lanes for address and four lanes for data.
6	`8'h0C`	4 bytes	1	1	1	4-byte fast. Single lane for opcode, address, and data.

3-Byte Address Support

The legacy Quad-SPI controller supports the following 3-byte address commands. The number of data lanes (DQ pins) for sending the instruction, opcode, and data and receiving the data listed in Table 24-7. The legacy Quad-SPI controller does not support instructions where the flash device requires using a different number of data lanes for the same instruction code.

Table 24-8: **3-Byte Address Support**

Command Number	Instruction Code	Address Bytes	Data Lanes Used for Opcode	Data Lanes Used for Address	Data Lanes Used for Data	Command Type
1	`8'h05`	3 bytes	1	–	1	Read status register
2	`8'h03`	3 bytes	1	1	1	Read normal
3	`8'h0B`	3 bytes	1	1	1	Read fast
4	`8'h3B`	3 bytes	1	1	2	Read dual output
5	`8'h6B`	3 bytes	1	1	4	Read quad output
6	`8'hBB`	3 bytes	1	2	2	Read dual I/O
7	`8'hEB`	3 bytes	1	4	4	Read quad I/O

Legacy Linear Addressing

The legacy Quad-SPI has 128 MB of allocated system memory and requires a 27-bit address [26 down to 0] to decode the address space. Register bit 27 of a linear Quad-SPI configuration register is added to enable the 4-byte address capability. When enabled, a 32-bit address is formed by concatenating the lower 27 bits received on the AXI address bus with five zeros.

When two flash devices are cascaded and the 4-byte address feature enabled, then bit 26 is used to select the flash. Setting bit 26 of the AXI address bus selects the upper flash by using the lower 26 bits of the AXI address bus and appending the MSB 6 bits with zeros to form a 4-byte address to the flash. Setting bit 26 to zero selects the lower flash by using the lower 26 bits of the AXI address bus and appending the MSB 6 bits with zeros to form a 4-byte address to the flash.

When two flash devices are cascaded and the 4-byte address feature is not enabled, then bit 26 is used to select the flash. Setting bit 26 of the AXI address bus selects the upper flash and the lower 24 bits of the AXI address bus are used to address the flash. To select the lower flash, bit 26 is set to zero and the lower 24 bits of the AXI address bus are used to address the flash.

When two flash devices are connected in parallel and the 4-byte address feature is enabled, then to access the flash pad bit 26 down to bit 1 with seven zeros. Bit 0 is discarded because the memory content is shared across the memories.

When two flash devices are connected in parallel and the 4-byte address feature is disabled, then to access the flash use bit 24 down to bit 1. Bit 0 is discarded because the memory content is shared across the memories.

When a single flash is connected and the 4-byte address feature is enabled, then to access the flash pad bit 25 down to bit 0 of the AXI address bus with six zeros.

When a single flash is connected and the 4-byte address feature is disabled, then to access the flash use bits 23 down to bit 0 of the AXI address.

Programming Requirements for Linear Mode

In linear mode, for both the 3-byte and 4-byte address, set the DUMMY_CYCLE_EN register bit to `1'b1`. For the read data bytes `0x03` command, set the DUMMY_CYCLE_EN register to `1'b1` and program the DUMMY_CYCLES in the COMMAND register to zeros.

Legacy Quad-SPI I/O Interface

The I/O signals are available through the MIO pins. The Quad-SPI controller supports up to two SPI flash memories in either a shared or separate bus configuration. The controller supports operation in the following configurations.

- Quad-SPI single slave select 4-bit I/O.
- Quad-SPI dual slave select 8-bit parallel I/O.
- Quad-SPI dual slave select 4-bit stacked I/O.

IMPORTANT: *QSPI0 should always be present when using the Quad-SPI memory subsystem. QSPI1 is optional and is only required for a two-memory arrangement. Therefore, QSPI1 cannot be used alone.*

Legacy Quad-SPI Single Slave Select 4-bit I/O

Figure 24-5 shows a block diagram of the 4-bit flash memory interface connected to the controller configuration.

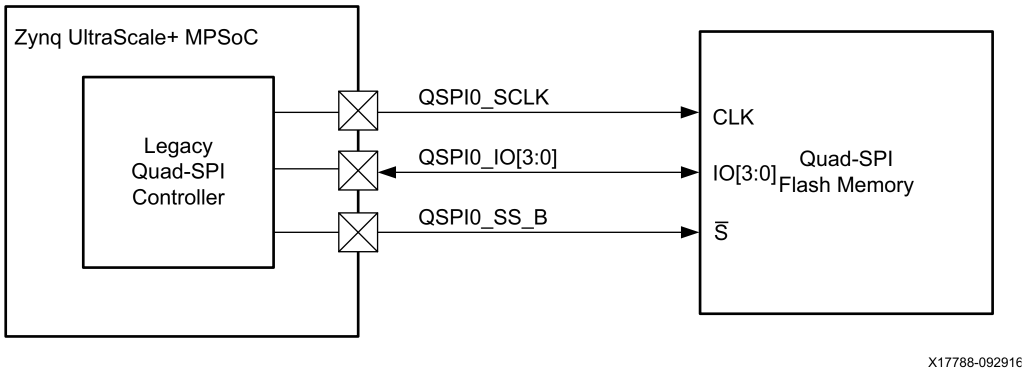


Figure 24-5: **Legacy Quad-SPI Single Slave Select 4-bit I/O**

Legacy Quad-SPI Dual Slave Select 8-bit Parallel I/O

The controller supports up to two SPI flash memories operating in parallel, as shown in Figure 24-6. This configuration increase the maximum addressable SPI flash memory from 16 MB (24-bit address) to 32 MB (25-bit address). In this configuration, the device level XIP mode is not supported.

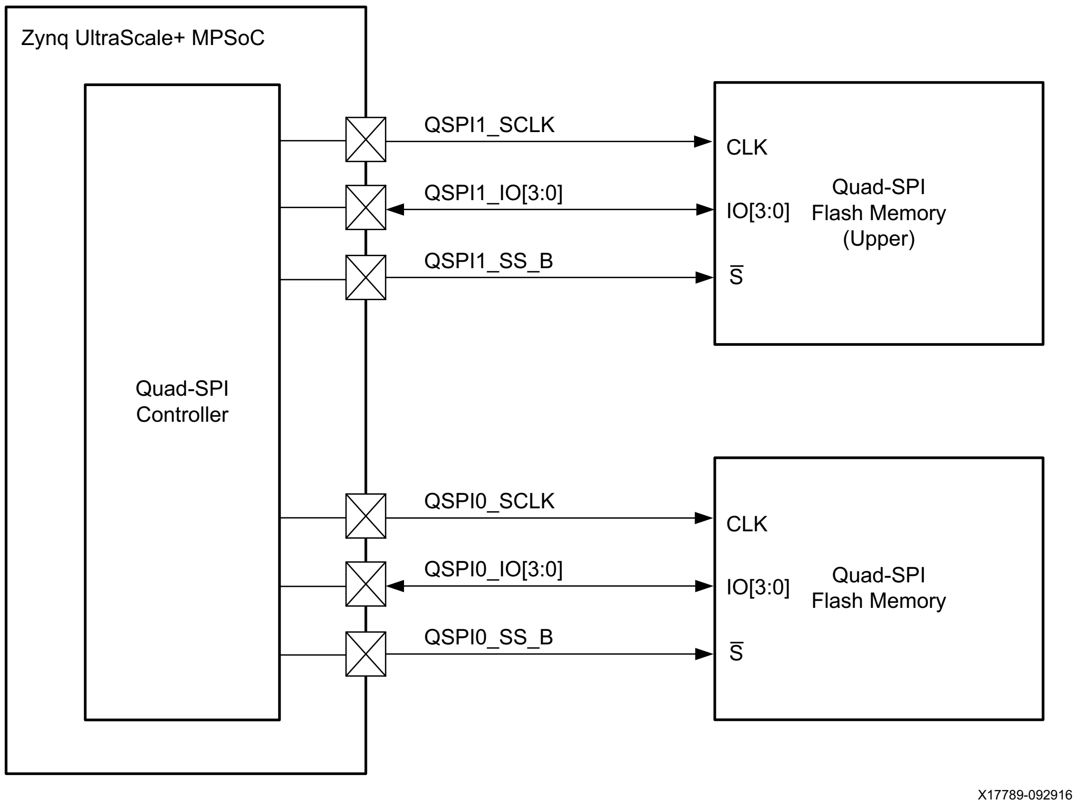


Figure 24-6: **Legacy Quad-SPI Dual Slave Select 8-bit Parallel I/O**

For 8-bit parallel configuration, the even bits of the data words are located in the lower memory and the odd bits of data are located in the upper memory. The Quad-SPI controller manages the data in linear mode. The controller reads from the two Quad-SPI devices and ORs (OR operation) the status information from both devices before writing the status data in the RXFIFO. Figure 24-9 shows the data bit arrangement of a 32-bit data word for an 8-bit parallel configuration. Table 12-8 shows the Quad-SPI commands in dual Quad-SPI parallel mode.

Table 24-9: **Quad-SPI Dual Slave Select 8-bit Parallel I/O Data Management**

Single Device																															
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
byte 0								byte 1								byte 2								byte 3							

Dual Devices Lower Memory															
6	4	2	0	14	12	10	8	22	20	18	16	30	28	26	24

Dual Devices Upper Memory															
7	5	3	1	15	13	11	9	23	21	19	17	31	29	27	25
byte 0				byte 1				byte 2				byte 3			

Table 24-10: **Quad-SPI Command List for Dual Quad-SPI Parallel Mode**

Command	Dual Parallel Quad-SPI Controller
Sector Erase	The Quad-SPI controller sends and erase command to both devices. A 64 KB erase operation erases each device, which effectively erases a combined 128 KB from both memories.
Read ID	The received data is taken from the lower flash bus and places it in RXD. There is no need to combine data. The upper and lower flash devices must be identical when using the parallel flash mode.
Read	The even and odd data bits are read from both devices and are interleaved as shown in Table 24-9.
RDSR	The work-in-progress (WIP) bit from both devices are OR'ed together to form the LSB of the data read. The other seven bits come from the lower bus.

In 8-bit parallel configuration, the total addressable memory size is 512 MB.This requires a 29-bit address. All accesses to memory must be word aligned and have double-byte resolution. In linear mode, the Quad-SPI controller divides the AXI address by two and sends the divided address to the Quad-SPI device.

Note: In a dual parallel configuration of two flash devices, when GQSPI is used to write to flash, and LQSPI in linear mode is used to read data from flash, the data read does not match with the data write. This data mismatch is caused by the difference in the handling of data by the two controllers in dual parallel configuration. GQSPI writes are "byte striped" and LQSPI reads are "bit interleaved". The application software should handle this interoperability between LQSPI and GQSPI in a dual parallel configuration.

Legacy Quad-SPI Dual Slave Select 4-bit Stacked I/O

To reduce the I/O pin count, the controller also supports up to two SPI flash memories in a shared bus configuration, as shown in Figure 24-7. This configuration increase the maximum addressable SPI flash memory from 256 MB (28-bit address) to 512 MB (29-bit address), but the throughput remains the same as in single memory mode. In this 4-bit stacked I/O configuration, the device level XIP mode (read instruction codes of `BBh` and `EBh`) is not supported.

The lower SPI flash memory should always be connected when using the linear Quad-SPI memory subsystem. The upper flash memory is optional. The total address space is 512 MB with a 29-bit address. In linear address mode, the AXI address bit 28 determines the upper or lower memory page. All of the commands are executed by the device selected by address bit 28.

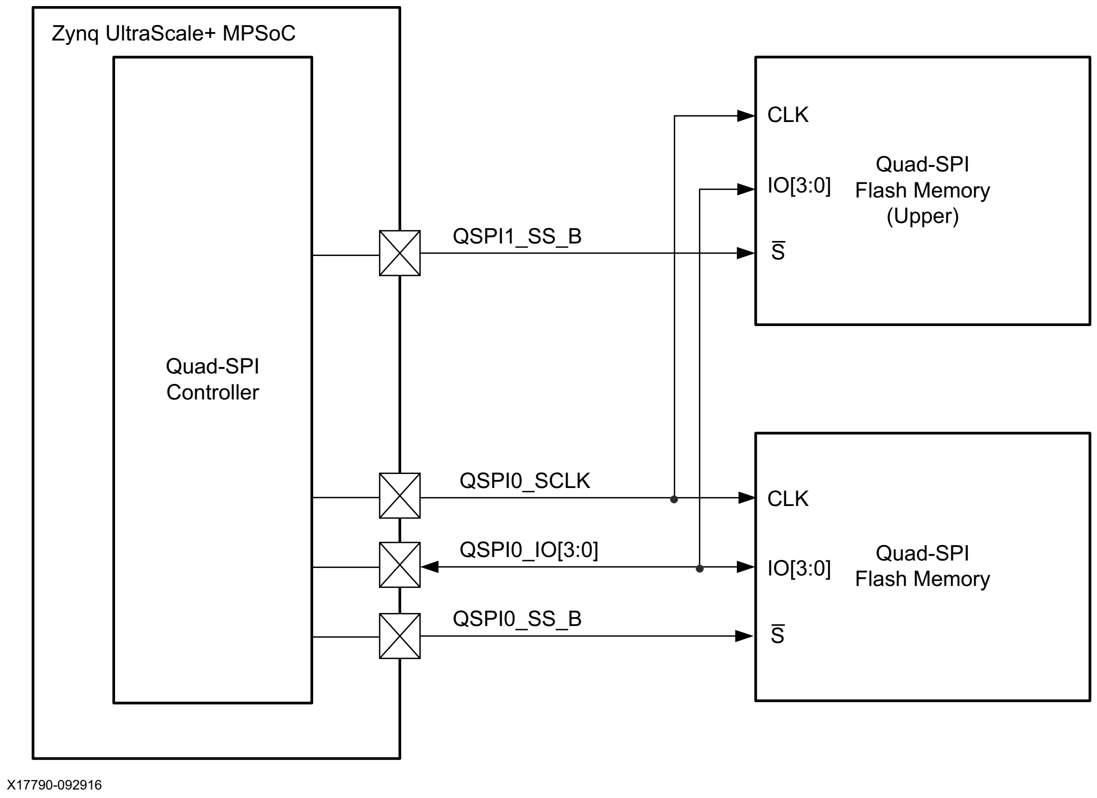


Figure 24-7: **Quad-SPI Dual Slave Select 4-bit Stacked I/O**

Register Overview

The registers for the legacy controller are not shared by the generic controller because there are few overlapped registers. However, all the DMA related registers are shared between the legacy controller and the generic controller. The register set for the dual controller is located at 0xFF0F_0000.

Table 24-11 lists the Quad-SPI controller registers.

Table 24-11: **Quad-SPI Register Summary**

Register Name	Register Offset	Width	Type	Reset Value	Description
Config	0x00000000	32	Mixed	0x80000000	Quad-SPI configuration register.
ISR	0x00000004	32	Mixed	0x00000104	Quad-SPI interrupt status register.
IER	0x00000008	32	Mixed	0x00000000	Interrupt enable register.
IDR	0x0000000C	32	Mixed	0x00000000	Interrupt disable register.
IMR	0x00000010	32	RO	0x00000000	Interrupt unmask register.
Enable	0x00000014	32	Mixed	0x00000000	Quad-SPI enable register.
Delay	0x00000018	32	RW	0x00000000	Delay register.
TXD0	0x0000001C	32	WO	0x00000000	Transmit data register. Keyhole addresses for the transmit data FIFO. See also TXD1-3.
Rx_data	0x00000020	32	RO	0x00000000	Receive data register.
Slave_Idle_count	0x00000024	32	Mixed	0x000000FF	Slave idle count register.
Tx_thres	0x00000028	32	RW	0x00000001	TXFIFO threshold register.
Rx_thres	0x0000002C	32	RW	0x00000001	RXFIFO threshold register.
GPIO	0x00000030	32	RW	0x00000001	General purpose inputs and outputs register for the Quad-SPI controller.
LPBK_DLY_ADJ	0x00000038	32	RW	0x00000033	Loopback master clock delay adjustment register.
TXD1	0x00000080	32	WO	0x00000000	Transmit data register. Keyhole addresses for the transmit data FIFO.
TXD2	0x00000084	32	WO	0x00000000	Transmit data register. Keyhole addresses for the transmit data FIFO.
TXD3	0x00000088	32	WO	0x00000000	Transmit data register. Keyhole addresses for the transmit data FIFO.
LQSPI_CFG	0x000000A0	32	RW	0x000002EB	Configuration register specifically for the linear Quad-SPI controller.
LQSPI_STS	0x000000A4	9	RO	0x00000000	Status register specifically for the linear Quad-SPI controller.

Table 24-11: **Quad-SPI Register Summary *(Cont'd)***

Register Name	Register Offset	Width	Type	Reset Value	Description
COMMAND	`0x000000C0`	32	Mixed	`0x00000000`	Command control register. This register needs to be programmed for every request.
TRANSFER_SIZE	`0x000000C4`	32	Mixed	`0x00000000`	Transfer size register.
DUMMY_CYCLE_EN	`0x000000C8`	32	Mixed	`0x00000000`	Dummy cycles enable register.
MOD_ID	`0x000000FC`	32	RW	`0x01090101`	Module identification register.
GQSPI_CFG	`0x00000100`	32	Mixed	`0x00000000`	Generic Quad-SPI configuration register.
GQSPI_ISR	`0x00000104`	32	Mixed	`0x00000B84`	Generic Quad-SPI interrupt status register.
GQSPI_IER	`0x00000108`	32	Mixed	`0x00000000`	Generic Quad-SPI interrupt enable register.
GQSPI_IDR	`0x0000010C`	32	Mixed	`0x00000000`	Generic Quad-SPI interrupt disable register.
GQSPI_IMR	`0x00000110`	32	Mixed	`0x00000FBE`	Generic Quad-SPI interrupt unmask register.
GQSPI_En	`0x00000114`	32	Mixed	`0x00000000`	Generic Quad-SPI enable register.
GQSPI_TXD	`0x0000011C`	32	WO	`0x00000000`	Generic Quad-SPI TX data register. Keyhole addresses for the transmit data FIFO.
GQSPI_RXD	`0x00000120`	32	RO	`0x00000000`	Generic Quad-SPI RX data register.
GQSPI_TX_THRESH	`0x00000128`	32	Mixed	`0x00000001`	Generic Quad-SPI TXFIFO Threshold Level register.
GQSPI_RX_THRESH	`0x0000012C`	32	Mixed	`0x00000001`	Generic Quad-SPI RXFIFO threshold level register.
GQSPI_GPIO	`0x00000130`	32	Mixed	`0x00000001`	Generic Quad-SPI GPIO for write protect register.
GQSPI_LPBK_DLY_ADJ	`0x00000138`	32	Mixed	`0x00000033`	Generic Quad-SPI loopback clock delay adjustment register.
GQSPI_GEN_FIFO	`0x00000140`	32	Mixed	`0x00000000`	Generic Quad-SPI generic FIFO data register. Keyhole addresses for the generic FIFO.
GQSPI_SEL	`0x00000144`	32	Mixed	`0x00000000`	Generic Quad-SPI select register.
GQSPI_FIFO_CTRL	`0x0000014C`	32	Mixed	`0x00000000`	Generic Quad-SPI FIFO control register.
GQSPI_GF_THRESH	`0x00000150`	32	Mixed	`0x00000010`	Generic Quad-SPI generic FIFO threshold level register.
GQSPI_POLL_CFG	`0x00000154`	32	Mixed	`0x00000000`	Generic Quad-SPI poll configuration register
GQSPI_P_TIMEOUT	`0x00000158`	32	RW	`0x00000000`	Generic Quad-SPI poll timeout register.

Table 24-11: **Quad-SPI Register Summary *(Cont'd)***

Register Name	Register Offset	Width	Type	Reset Value	Description
GQSPI_XFER_STS	0x0000015C	32	RO	0x00000000	Generic Quad-SPI transfer status register.
GQSPI_GF_SNAPSHOT	0x00000160	32	Mixed	0x00000000	Generic Quad-SPI generic FIFO snap shot register.
GQSPI_RX_COPY	0x00000164	32	Mixed	0x00000000	Generic Quad-SPI receive data copy register.
QSPI_DATA_DLY_ADJ	0x000001F8	32	RW	0x00000000	Quad-SPI RX data delay register.
GQSPI_MOD_ID	0x000001FC	32	RW	0x010A0000	Generic Quad-SPI module identification register.
QSPIDMA_DST_ADDR	0x00000800	32	Mixed	0x00000000	Destination memory address for DMA stream→memory data transfer.
QSPIDMA_DST_SIZE	0x00000804	32	Mixed	0x00000000	DMA transfer payload for DMA stream→memory data transfer.
QSPIDMA_DST_STS	0x00000808	32	Mixed	0x00000000	General DST DMA status.
QSPIDMA_DST_CTRL	0x0000080C	32	RW	0x803FFA00	General DST DMA control.
QSPIDMA_DST_I_STS	0x00000814	32	Mixed	0x00000000	DST DMA interrupt status register.
QSPIDMA_DST_I_EN	0x00000818	32	Mixed	0x00000000	DST DMA interrupt enable.
QSPIDMA_DST_I_DIS	0x0000081C	32	Mixed	0x00000000	DST DMA interrupt disable.
QSPIDMA_DST_I_MASK	0x00000820	32	Mixed	0x000000FE	DST DMA interrupt mask.
QSPIDMA_DST_CTRL2	0x00000824	32	Mixed	0x081BFFF8	General DST DMA control register 2.
QSPIDMA_DST_ADDR_MSB	0x00000828	32	Mixed	0x00000000	Destination memory address (MSBs) for DMA stream→memory data transfer.

Quad-SPI Tap Delay Values

The recommended clock and data tap delay values should be programmed based upon the frequency of operation. Refer to the *Zynq UltraScale+ MPSoC Data Sheet: DC and AC Switching Characteristics* (DS925) [Ref 2] and Answer Record 72797 for information on I/O timing.

At 40 MHz, the Quad-SPI controller should be in non-loopback mode with the clock and data tap delays bypassed. The register settings are shown in Table 24-12. These are default values and are applicable for both generic and legacy modes.

Table 24-12: **Quad-SPI Controller at 40 MHz Tap Delay Value**

Register Bit	Value	Description
IOU_SLCR.IOU_TAPDLY_BYPASS [LQSPI_RX]	1	Bypass clock tap delay.
QSPI.LPBK_DLY_ADJ [USE_LPBK]	0	Disable clock loopback mode.
QSPI.LPBK_DLY_ADJ [DLY1]	00	Clock tap delay 1.
QSPI.LPBK_DLY_ADJ [DLY0]	000	Clock tap delay 0.
QSPI.QSPI_DATA_DLY_ADJ [USE_DATA_DLY]	0	Data tap delay enable.
QSPI.QSPI_DATA_DLY_ADJ [DATA_DLY_ADJ]	000	Data tap delay.

At 100 MHz, the Quad-SPI controller should be in clock loopback mode with the clock tap delay bypassed, but the data tap delay enabled. The register settings are shown in Table 24-13. These values are applicable for both generic and legacy modes.

Table 24-13: **Quad-SPI Controller at 100 MHz Tap Delay Value**

Register Bit	Value	Description
IOU_SLCR.IOU_TAPDLY_BYPASS [LQSPI_RX]	1	Bypass clock tap delay.
GQSPI_LPBK_DLY_ADJ [USE_LPBK]	1	Enable clock loopback mode.
GQSPI_LPBK_DLY_ADJ [DLY1]	00	Clock tap delay 1.
GQSPI_LPBK_DLY_ADJ [DLY0]	000	Clock tap delay 0.
QSPI_DATA_DLY_ADJ [USE_DATA_DLY]	1	Data tap delay enable.
QSPI_DATA_DLY_ADJ [DATA_DLY_ADJ]	010	Data tap delay (three taps).

Note: The taps mentioned here refer to non-PVT compensated delay elements. Programmable delay elements are provided on incoming data and on sampling clock to delay data or sampling clock signals. The value of these taps should not be changed dynamically.

At 150 MHz, only the generic controller can be used. The generic controller should be in clock loopback mode and the clock tap delay enabled, but the data tap delay disabled. The register settings are shown in Table 24-14.

Table 24-14: **Generic Quad-SPI Controller at 150 MHz Tap Delay Values**

Register Bit	Value	Description
IOU_SLCR.IOU_TAPDLY_BYPASS [LQSPI_RX]	`0`	Enable clock tap delay.
GQSPI_LPBK_DLY_ADJ[USE_LPBK]	`1`	Enable clock loopback mode.
GQSPI_LPBK_DLY_ADJ [DLY1]	`00`	Clock tap delay 1.
GQSPI_LPBK_DLY_ADJ [DLY0]	`000`	Clock tap delay 0.
QSPI_DATA_DLY_ADJ [USE_DATA_DLY]	`0`	Data tap delay disable.
QSPI_DATA_DLY_ADJ [DATA_DLY_ADJ]	`000`	Data tap delay.

Note: The legacy Quad-SPI controller does not support 150 MHz frequency.

Programming and Usage Considerations

The generic Quad-SPI controller supports two operating modes: I/O mode and the DMA mode.

In I/O mode, which supports all types of memory operations, the SPI memory instructions are sent to the generic FIFO at `0x00000140` and the program data is sent to a fixed offset address of `0x0000011C`. The read data or status is retrieved from a fixed offset address of `0x00000120`, `0x0000010C`. The software is responsible for providing the SPI instruction and handling data formatting, and alignment. The generic Quad-SPI controller is responsible for managing the low-level signaling.

DMA Mode Configuration Sequence

1. DMA configuration
   - Program the QSPIDMA_DST_ADDR with the destination location (word aligned).
   - For memories greater than 32 address bits, the QSPIDMA_DST_ADDR_MSB must be configured.
   - Program the QSPIDMA_DST_SIZE with the number of words to transfer (word aligned).
   - Program the QSPIDMA_DST_CTRL and QSPIDMA_DST_CTRL2 as required.

2. Quad-SPI I/O mode configuration
    - Configure the MODE_EN bits to `2'b10` in the GQSPI_CFG register.
    - Program the generic FIFO for writing the command, flash memory address, dummy cycles, and transfer size.

In all the modes listed, one or two SPI memories can be used, but the lower memory should always be present in dual-parallel mode. Configure the two memory devices to use separate data buses to double both throughput and storage size or a common shared data bus to reduce pin count with double storage size.

By default, the Quad-SPI memory subsystem comes up in I/O mode to allow users to configure the flash memory or to carry out different type of memory operations.

Transfer Size Limitations

The RXFIFO, TXFIFO, and generic command FIFO are 32-bit wide FIFOs, and all transfers must be a multiple of 4-bytes (i.e., 4, 8, 12, 16, etc.).

For the TXFIFO, a transfer of a non-multiple of 4 bytes results in the subsequent transfer to pop out from the start of the next word, and not from the next byte. This means that trying to transmit any number of bytes in a word drains the entire 4 bytes of that word completely, but the data transfer will only be the bytes requested. For example, when 5 words (20 bytes) are loaded into the TXFIFO and a request is made to transmit 10 bytes, this results in a 10 byte transmission. However, as the number of bytes requested is not word aligned, the TXFIFO pops out the entire 4 bytes of the last word, which drains off 12 bytes instead of 10 bytes. As a result, the remaining number of bytes in the FIFO is 8 bytes (2 words).

Generic Quad-SPI Controller Programming

The flow diagram for the generic Quad-SPI programming sequence is shown in Figure 24-8.

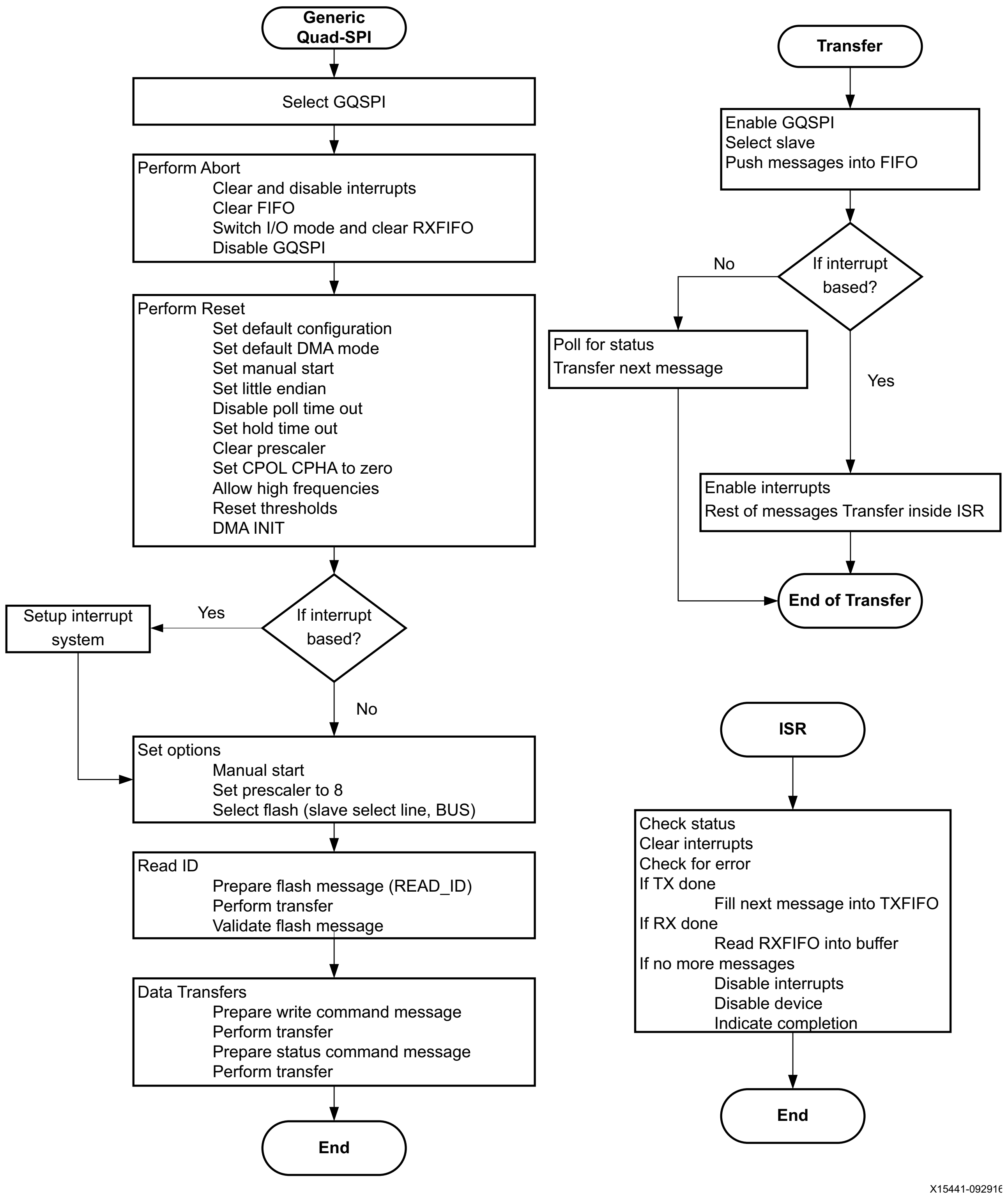


Figure 24-8: **Generic Quad-SPI Programming Flowchart**

Generic FIFO Programming

Programming SPI Modes

Table 24-15 lists the lower data bus pins driven by the generic Quad-SPI controller as per the SPI mode, receive, and transmit of the lower data bus.

Table 24-15: **SPI Modes in Generic Quad-SPI Controller (Lower Data Bus is Active)**

SPI Mode	Mode Description	Data Bus Select	Receive	Transmit	Lower Data Bus [3:0] I/O			
`2'b01`	SPI transmit.	`2'b01`	`1'b0`	`1'b1`	Not used	Not used	Not used	O
`2'b01`	SPI receive.	`2'b01`	`1'b1`	`1'b0`	Not used	Not used	I	Not used
`2'b01`	SPI transmit and receive.	`2'b01`	`1'b1`	`1'b1`	Not used	Not used	I	O
`2'b10`	Dual-SPI transmit.	`2'b01`	`1'b0`	`1'b1`	Not used	Not used	O	O
`2'b10`	Dual-SPI receive.	`2'b01`	`1'b1`	`1'b0`	Not used	Not used	I	I
`2'b11`	Quad-SPI transmit.	`2'b01`	`1'b0`	`1'b1`	O	O	O	O
`2'b11`	Quad-SPI receive.	`2'b01`	`1'b1`	`1'b0`	I	I	I	I

Programming Data Transfer Length and Usage of Exponent

This section describes some examples on programming different data transfer lengths. Only the length related fields are mentioned.

When 128 bytes are read/written from the SPI flash, the generic FIFO configuration uses the values in Table 24-16.

Table 24-16: **Generic FIFO Configuration for 128 Bytes**

Description	immediate_data	data_xfer	exponent
128 bytes	`8'h80`	`1'b1`	`1'b0`

When 1,000 bytes are read/written from the SPI flash, the generic FIFO configuration uses the values described by the options listed in Table 24-17.

Table 24-17: **Generic FIFO Configuration for 1,000 Bytes**

	Description	immediate_data	data_xfer	exponent
Option 1	512 bytes using exponent	`8'h09`	`1'b1`	`1'b1`
	256 bytes	`8'h08`	`1'b1`	`1'b1`
	Remaining 232 bytes	`8'hE8`	`1'b1`	`1'b0`
Option 2	256 bytes	`8'h08`	`1'b1`	`1'b1`
	256 bytes	`8'h08`	`1'b1`	`1'b1`
	256 bytes	`8'h08`	`1'b1`	`1'b1`
	Remaining 232 bytes	`8'hE8`	`1'b1`	`1'b0`

When 1G bytes are read/written from the SPI flash, the generic FIFO configuration uses the values in Table 24-18.

Table 24-18: **Generic FIFO Configuration for 1G Bytes**

Description	immediate_data	data_xfer	exponent
1G bytes	`8'h1E`	`1'b1`	`1'b1`

When 64 bytes are read/written from the SPI flash, the generic FIFO configuration uses the values in Table 24-19.

Table 24-19: **Generic FIFO Configuration for 64 Bytes**

Description	immediate_data	data_xfer	exponent
64 bytes	`8'h40`	`1'b1`	`1'b0`

Programming Poll

The poll bit of the generic FIFO is used to continuously read the status of SPI device until it matches with the value in the POLL_DATA field of the poll register. The data read from the SPI device is written into the RXFIFO.

When two flash devices are used, the Quad-SPI controller does not execute the next command until the data from both flash devices matches with the value in the POLL_DATA of the poll register, which is determined by the mask bits.

The poll bit is useful when checking the status of a flash device. For example, when a page program is issued to a flash device, the software polls the status, to check if write is completed. The polling requires multiple read requests to status register. By setting the poll bit 1, the generic Quad-SPI controller continuously reads the data and checks the expected status bits.

Use Case: Check Success of Page Program/Erase

When a page program/erase command is issued to a flash device, the software needs to poll the status of the operation. This requires multiple read requests to the status register. By setting the poll bit 1, the generic Quad-SPI controller continuously reads the data and checks the expected status bits. This mechanism avoids having the software/processor issue multiple read requests and is controlled on the MPSoC by the generic Quad-SPI controller.

When the poll bit is set the useful fields are listed.

- Generic FIFO field—SPI mode.
- Generic FIFO field—receive.
- Generic FIFO field—data bus select.
- Poll register field—POLL_DATA value.
- Poll register field—enable lower data bus mask.
- Poll register field—enable upper data bus mask.
- Poll register field—data bus mask, the same value is used for both upper and lower devices.

In the case of a polling operation when one data bus of a generic Quad-SPI controller is active, in single mode or stacked mode, only one data bus is active and the value of the data bus select is either `2'b01` or `2'b10`. In this case, only one device is connected to the generic Quad-SPI controller. Hence, the data on the connected device is captured and compared against the POLL_DATA field of the poll register.

In the case of a polling operation when two data buses of a generic Quad-SPI controller are active, in dual-parallel mode, there are two devices connected to the generic Quad-SPI controller. When the value of receive = `1'b1`, the possible values of the data bus select are `2'b01`, `2'b10`, or `2'b11`.

- When receive = `1'b1` and the data bus select is `2'b01`, the data in the lower device is captured and compared against the immediate_data field depending on the poll mask register values.
- When receive = `1'b1` and the data bus select is `2'b10`, the data in the upper device is captured and compared against the POLL_DATA field depending on the poll mask register value.
- When receive = `1'b1` and the data bus select is `2'b11`, the data in both the lower and upper devices is captured simultaneously and compared independently against the POLL_DATA field of the poll register. The generic Quad-SPI controller reads the data from both data buses. It compares the data against value that is configured in the POLL_DATA field of the poll register, also considering the poll mask value. The controller waits until the data matches. Once the data matches, the controller goes to the next entry (if any) of the generic FIFO.

Terminating Poll

The poll operation termination is controlled using the POLL_TIMEOUT register and the EN_POLL_TIMEOUT field of the generic Quad-SPI configuration register.

When the EN_POLL_TIMEOUT field is set to 0, the generic Quad-SPI controller polls indefinitely until it matches with the value programmed in the POLL_DATA field of the poll register. In this case, the poll operation is only terminated when the received data matches with the value of the POLL_DATA field. This depends on the poll mask value, if enabled.

When the EN_POLL_TIMEOUT field is set to 1, the value of the POLL_TIMEOUT register is used. The generic Quad-SPI controller increments an internal counter and keeps polling for the number of reference clock cycles configured in the POLL_TIMEOUT register. When the internal counter expires, the generic Quad-SPI controller generates a Poll_Timeout_Int interrupt.

When both upper and lower data buses are active, the interrupt indicates if the captured data of any one or both of the devices captured data does not matched with the POLL_DATA field of the poll register. This depends on the poll mask value, if enabled.

When only one data bus is active, in dual-parallel mode, the poll counter expires when either the lower or upper data is not as expected in the POLL_DATA field of the poll register. This depends on the poll mask value, if enabled.

Programming Stripe

The stripe bit of a generic FIFO is used when both lower and upper data buses are active. When both cs_lower and cs_upper are set to 1, program the options listed in Table 24-20.

Table 24-20: **Transmit and Receive Generic FIFO Stripe Bit**

Description	immediate_data	data_xfer	Stripe	Transmit	Receive	Data Bus Select
Transmit immediate_data field to both flash devices when stripe is `1'b0`.						
Generic FIFO fields	`8'hEB`	`1'b0`	`1'b0`	`1'b1`	`1'b0`	`2'b11`
Transmit even bytes of TXFIFO data to lower device and odd bytes of TXFIFO to upper device when stripe is `1'b1`.						
Generic FIFO fields	`8'h64`	`1'b1`	`1'b1`	`1'b1`	`1'b0`	`2'b11`
Transmit TXFIFO data to both devices (not commonly used) when stripe is `1'b0`.						
Generic FIFO fields	`8'h64`	`1'b1`	`1'b0`	`1'b1`	`1'b0`	`2'b11`
Receive RXFIFO (Stripe = 1'b0 is not applicable when receive is set to 1.						
Generic FIFO fields	`8'h64`	`1'b1`	`1'b1`	`1'b0`	`1'b1`	`2'b11`

Transferring Odd Bytes

The generic Quad-SPI 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.

Modes of Operation

Generic Quad-SPI Controller in 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.

Generic Quad-SPI Controller in DMA Mode

For DMA 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'b10` of the GQSPI_CFG register.
3. Write the command, address, dummies in the generic FIFO using the read request.
4. The generic Quad-SPI controller sends the command as programmed in the generic FIFO and reads the data into the RXFIFO.
5. The DMA controller issues DMA requests using the AXI master interface and sends the RXFIFO 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.

Flash Commands

NOR Flash Commands

To send different flash commands to the flash devices, the generic FIFO must be precisely programmed. The following examples describe sample SPI flash commands.

Page Read Command

The generic FIFO contents for the read command are listed in Table 24-21.

Table 24-21: **Generic FIFO Contents for Read Command**

Description	Reserved	Poll	Stripe	Receive	Transmit	Data Bus Select	CS_ Upper	CS_ Lower	SPI Mode	Exponent	Data _xfer	Immediate _Data
	31:20	19	18	17	16	15:14	13	12	11:10	9	8	7:0
Start driving chip select (CS). Setup time is four QSPI_REF_CLK cycles.	12'd0	1'b0	1'b0	1'b0	1'b0	2'b01	1'b0	1'b1	2'b01	1'b0	1'b0	8'h04
Send opcode 03 for page read. Start driving chip select and clock.	12'd0	1'b0	1'b0	1'b0	1'b1	2'b01	1'b0	1'b1	2'b01	1'b0	1'b0	8'h03
Send first address byte 10.	12'd0	1'b0	1'b0	1'b0	1'b1	2'b01	1'b0	1'b1	2'b01	1'b0	1'b0	8'h10
Send second address byte 20.	12'd0	1'b0	1'b0	1'b0	1'b1	2'b01	1'b0	1'b1	2'b01	1'b0	1'b0	8'h20
Send third address byte 30.	12'd0	1'b0	1'b0	1'b0	1'b1	2'b01	1'b0	1'b1	2'b01	1'b0	1'b0	8'h30
Read 100 bytes.	12'd0	1'b0	1'b0	1'b1	1'b0	2'b01	1'b0	1'b1	2'b01	1'b0	1'b1	8'h64
Stop CS/SCLK, chip-select is deasserted. CS hold time is three reference clock cycles (optional).	12'd0	1'b0	1'b0	1'b0	1'b0	2'b01	1'b0	1'b0	2'b01	1'b0	1'b0	8'h04

Quad I/O Read Command

The generic FIFO contents for the quad I/O read command are listed in Table 24-22.

Table 24-22: **Generic FIFO Contents for Quad I/O Read Command**

Description	Reserved	Poll	Stripe	Receive	Transmit	Data Bus Select	CS_ Upper	CS_ Lower	SPI Mode	Exponent	Data _xfer	Immediate _Data
	31:20	19	18	17	16	15:14	13	12	11:10	9	8	7:0
Start driving chip select (CS). Setup time is four reference clock cycles.	12'd0	1'b0	1'b0	1'b0	1'b0	2'b01	1'b0	1'b1	2'b01	1'b0	1'b0	8'h04
Send opcode EB for page read in SPI mode.	12'd0	1'b0	1'b0	1'b0	1'b1	2'b01	1'b0	1'b1	2'b01	1'b0	1'b0	8'hEB
Send first address byte 10 in quad mode.	12'd0	1'b0	1'b0	1'b0	1'b1	2'b01	1'b0	1'b1	2'b11	1'b0	1'b0	8'h10
Send second address byte 20 in quad mode.	12'd0	1'b0	1'b0	1'b0	1'b1	2'b01	1'b0	1'b1	2'b11	1'b0	1'b0	8'h20
Send third address byte 30 in quad mode.	12'd0	1'b0	1'b0	1'b0	1'b1	2'b01	1'b0	1'b1	2'b11	1'b0	1'b0	8'h30
Send dummy cycle.	12'd0	1'b0	1'b0	1'b0	1'b1	2'b01	1'b0	1'b1	2'b11	1'b0	1'b0	8'hA0
Send four dummy cycles.	12'd0	1'b0	1'b0	1'b0	1'b0	2'b01	1'b0	1'b1	2'b11	1'b0	1'b1	8'h04
Read 100 bytes in quad mode.	12'd0	1'b0	1'b0	1'b1	1'b0	2'b01	1'b0	1'b1	2'b11	1'b0	1'b1	8'h64
Stop CS/SCLK, chip select is deasserted (optional).	12'd0	1'b0	1'b0	1'b0	1'b0	2'b01	1'b0	1'b0	2'b11	1'b0	1'b0	8'h00

Quad Page Program Command

The generic FIFO contents for the quad page program command are listed in Table 24-23.

Table 24-23: **Generic FIFO Contents for Quad Page Program Command**

Description	Reserved	Poll	Stripe	Receive	Transmit	Data Bus Select	CS_ Upper	CS_ Lower	SPI Mode	Exponent	Data _xfer	Immediate _Data
	31:20	19	18	17	16	17:16	13	12	11:10	9	8	7:0
Start driving chip select (CS). Setup time is four reference clock cycles.	`12'd0`	`1'b0`	`1'b0`	`1'b0`	`1'b0`	`2'b01`	`1'b0`	`1'b1`	`2'b01`	`1'b0`	`1'b0`	`8'h04`
Send opcode 02 for page program in SPI mode.	`12'd0`	`1'b0`	`1'b0`	`1'b0`	`1'b1`	`2'b01`	`1'b0`	`1'b1`	`2'b01`	`1'b0`	`1'b0`	`8'h02`
Send first address byte 10 in SPI mode.	`12'd0`	`1'b0`	`1'b0`	`1'b0`	`1'b1`	`2'b01`	`1'b0`	`1'b1`	`2'b01`	`1'b0`	`1'b0`	`8'h10`
Send second address byte 20 in SPI mode.	`12'd0`	`1'b0`	`1'b0`	`1'b0`	`1'b1`	`2'b01`	`1'b0`	`1'b1`	`2'b01`	`1'b0`	`1'b0`	`8'h20`
Send third address byte 30 in SPI mode.	`12'd0`	`1'b0`	`1'b0`	`1'b0`	`1'b1`	`2'b01`	`1'b0`	`1'b1`	`2'b01`	`1'b0`	`1'b0`	`8'h30`
Send fourth address byte 40 in SPI mode.	`12'd0`	`1'b0`	`1'b0`	`1'b0`	`1'b1`	`2'b01`	`1'b0`	`1'b1`	`2'b01`	`1'b0`	`1'b0`	`8'h40`
Write 512 bytes in quad mode. Use exponent bit as $2^9 = 512$.	`12'd0`	`1'b0`	`1'b0`	`1'b0`	`1'b1`	`2'b01`	`1'b0`	`1'b1`	`2'b11`	`1'b1`	`1'b1`	`8'h09`
Stop CS/SCLK, chip select is deasserted.	`12'd0`	`1'b0`	`1'b0`	`1'b0`	`1'b0`	`2'b01`	`1'b0`	`1'b0`	`2'b11`	`1'b0`	`1'b0`	`8'h00`

Two SPI Flash Memories with Separate Buses (Dual Parallel)

The generic Quad-SPI controller supports up to two SPI flash memories operating in parallel as shown in Figure 24-9. Unlike the legacy Quad-SPI (LQSPI) controller, the chip select is driven independently to the upper and lower flash memory devices. The selection of the lower/upper memory is controlled by using the data bus select field (Table 24-4). With this approach, commands and data can be transmitted and received from both devices, or only upper or only lower flash memory device. In this configuration, the device level XIP mode is not supported.

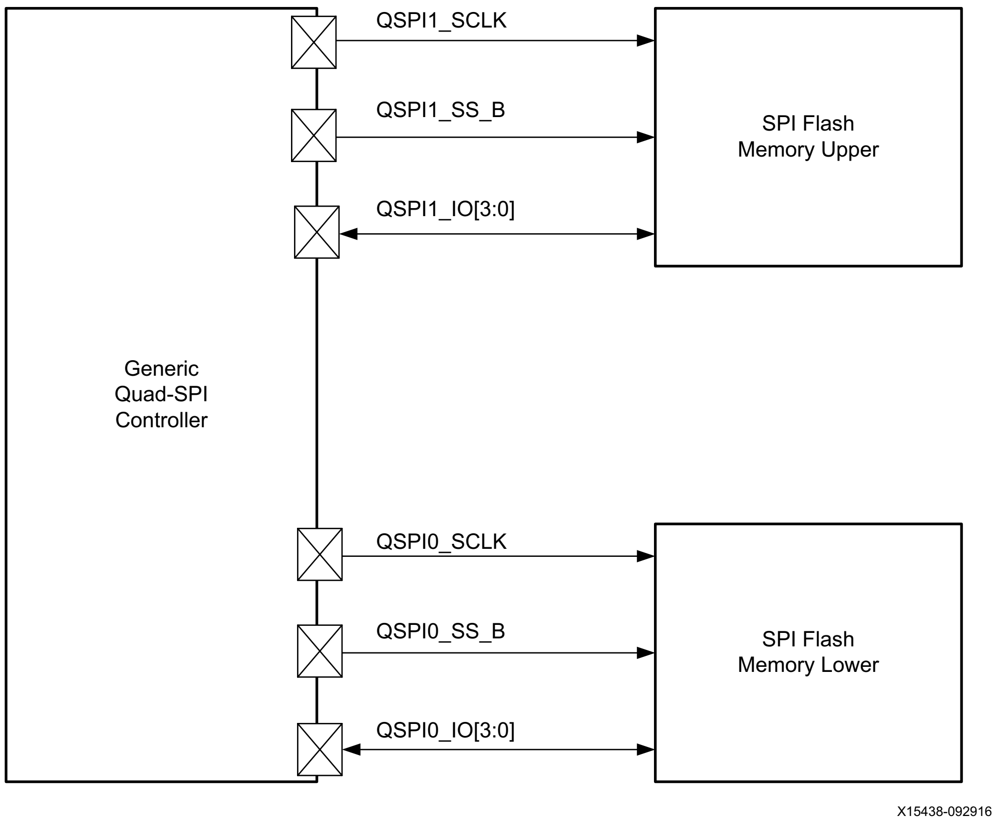


Figure 24-9: **Dual Parallel Mode**

Data Arrangement

When the stripe field of a generic FIFO is set, the lower data bus uses even bytes, i.e., byte 0, 2, 4 ..., of a data word, and the upper data bus uses odd bytes, i.e., byte 1, 3, 5, ..., of a data word.

Two SPI Flash Memories with a Shared Bus (Stacked)

To reduce I/O pin count, the generic Quad-SPI controller also supports two SPI flash memories in a shared bus arrangement. Figure 24-10 shows an example of a lower data bus connected to both flash devices. It is also possible to connect the upper data bus to both flash devices. The lower or upper memory selection is controlled by the data bus select field.

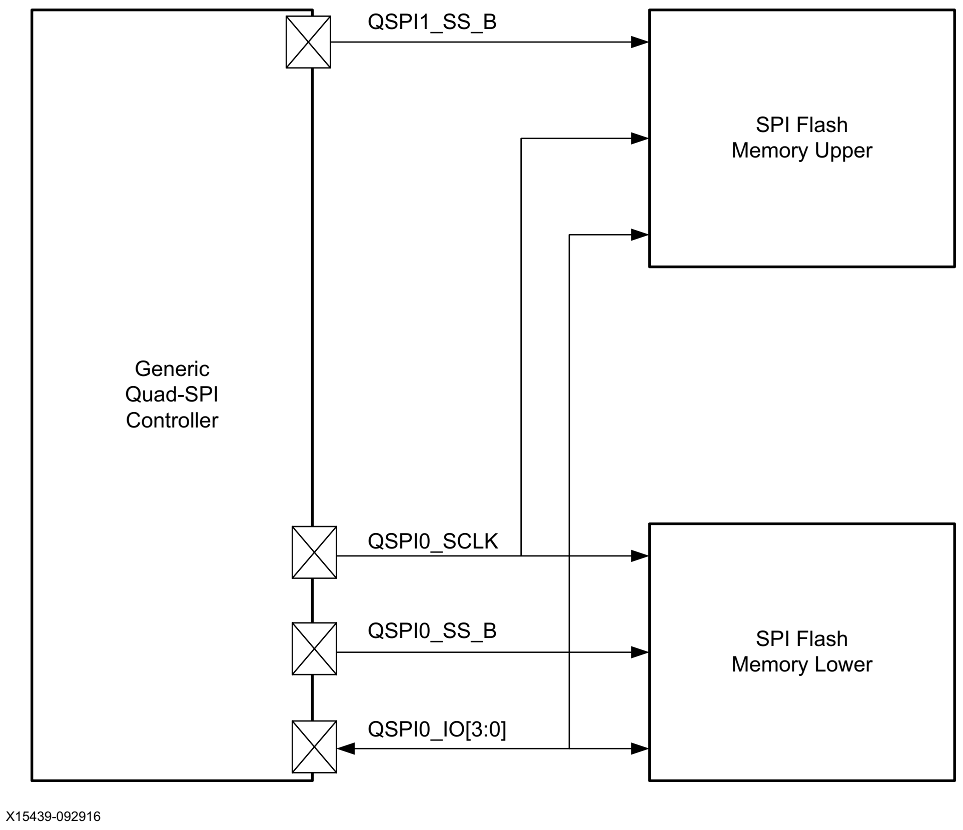


Figure 24-10: **Stacked Mode**

Write Protect

The write protect output signal is controlled by the QSPI.GPIO [WP_N] bit. Write protect is often connected on bit [2] of a 4-bit quad-SPI device bus. The write protect signal is driven Low for most flash devices, therefore the reset value is High (write protection deasserted).

In SPI and dual-SPI modes, the write protect signal from the general-purpose I/O register is connected to the WPB pin through the wpn_mo2 output for connection to the write protect control input on the flash device.

In quad mode, the write protect signal is connected as MIO2 and is driven by the controller.

When a write protect operation is not used, the write protect pin is driven to a 1 as expected by the SPI devices. Hence, an internal pull-up resistor is needed by the SPI device.

Controller Hold Signal

The hold signal is not supported by the generic Quad-SPI controller. In dual- or single-bit modes, the hold signal is tied to 1. In quad mode, the hold signal is driven by the controller. When a hold operation is not used, the hold pin is driven High as expected by the SPI devices.

Controller Interrupt

The generic Quad-SPI controller has a single interrupt output signal. The dma_irq signal from the DMA module is ORed with the internal interrupt (gqspi_int_irq) in the generic Quad-SPI controller, as shown in Figure 24-11. The generic Quad-SPI interrupt register does not show the status of the DMA interrupt signal.

The following steps outline the software programming model for a DMA interrupt.

1. Enable the DMA IRQ bits[0..7] of qspidma_dst_i_en, the DMA interrupt enable register.
2. The interrupt is set due to the DMA. For example, DMA done.
3. The generic Quad-SPI controller interrupt is asserted as the DMA module asserts dma_irq signal that is ORed with the generic Quad-SPI internal interrupt signal.
4. The software reads the generic Quad-SPI interrupt status register. The generic Quad-SPI interrupt status register bits are not set because this model is for the DMA interrupt.
5. The software reads the DMA interrupt status register, the DMA done bit is set.
6. The software writes a 1 to clear the done bit of the DMA interrupt status register.
7. The dma_irq signal is deasserted and the generic Quad-SPI interrupt is immediately deasserted.

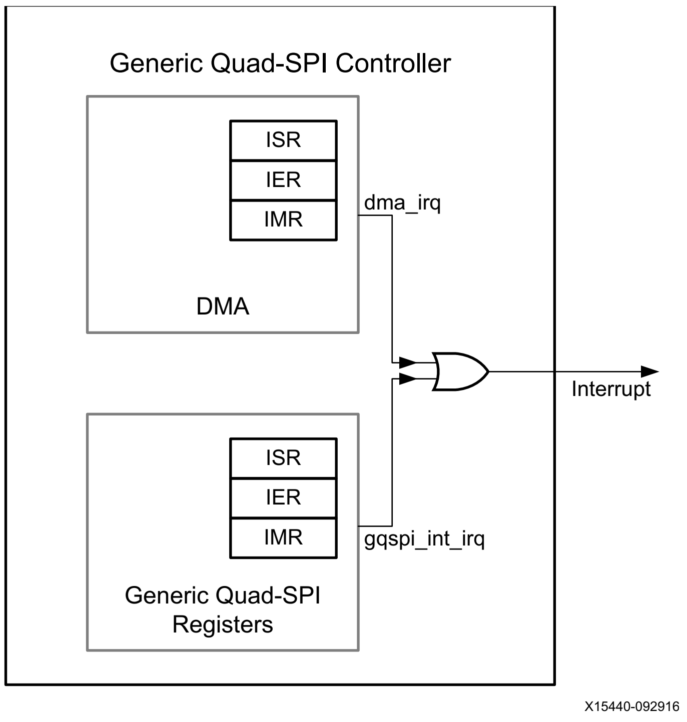


Figure 24-11: **Interrupt Mechanism**

Programming Examples

The programming examples for the generic Quad-SPI controller are listed in Table 24-24 through Table 24-31.

- Generic Quad-SPI Initialization and Reset
- Generic Quad-SPI Abort
- Generic Quad-SPI Set Options
- Generic Quad-SPI Interrupt Transfer
- Generic Quad-SPI Polled Transfer
- Generic Quad-SPI Flash Read ID
- Generic Quad-SPI Flash Erase
- Generic Quad-SPI Flash Write

Table 24-24: **Generic Quad-SPI Initialization and Reset**

Task	Register	Register Field	Register Offset	Bits	Value
Select generic Quad-SPI	GQSPI_SEL	generic_qspi_sel	`0x144`	0	`1`
Call: Quad-SPI PSU abort					
Configure SPI	GQSPI_CFG	MODE_EN | GEN_FIFO_START_MODE | ENDIAN | EN_POLL_TIMEOUT | WP_HOLD | BAUD_RATE_DIV | CLK_PH | CLK_POL	`0x100`	31:0	`A008_0000h`
Allow high frequencies	GQSPI_LPBK_DLY_ADJ	USE_LPBK	`0x138`	5	`1`
Reset thresholds	GQSPI_TX_THRESH	Level_TX_FIFO	`0x128`	5:0	`00_0001b`
Reset thresholds	GQSPI_RX_THRESH	Level_RX_FIFO	`0x12C`	5:0	`00_0001b`
Reset thresholds	GQSPI_GF_THRESH	Level_GF_FIFO	`0x150`	4:0	`1_0000b`
DMA initialize	QSPIDMA_DST_CTRL	FIFO_LVL_HIT_THRESH | APB_ERR_RESP | ENDIANNESS | AXI_BRST_TYPE | TIMEOUT_VAL | FIFO_THRESH | PAUSE_STRM | PAUSE_MEM	`0x80C`	31:0	`403F_FA00h`

Table 24-25: **Generic Quad-SPI Abort**

Task	Register	Register Field	Register Offset	Bits	Value
Read interrupt status and save	GQSPI_ISR	All bits	`0x104`	31:0	Read operation
Clear poll timeout counter interrupt	GQSPI_ISR	Poll_Time_Expire	`0x104`	1	`1`
Disable interrupts	QSPIDMA_DST_I_STS	FIFO_OVERFLOW | INVALID_APB | THRESH_HIT | TIMEOUT_MEM | TIMEOUT_STRM | AXI_BRESP_ERR | DONE	`0x814`	7:1	WTC, read and write back the same value.
Clear the transfer count interrupt	QSPIDMA_DST_STS	DONE_CNT	`0x808`	15:13	`111b`
Disable interrupts	GQSPI_IDR	RX_FIFO_EMPTY | Gen_FIFO_full | Gen_FIFO_not_full | TX_FIFO_EMPTY | Gen_FIFO_Empty | RX_FIFO_full | RX_FIFO_not_empty | TX_FIFO_full | TX_FIFO_not_full | Poll_Time_Expire	`0x10C`	11:0	`FBEh`
Disable interrupts	QSPIDMA_DST_I_DIS	FIFO_OVERFLOW | INVALID_APB | THRESH_HIT |TIMEOUT_MEM | TIMEOUT_STRM | AXI_BRESP_ERR | DONE	`0x81C`	7:1	`7'h7F`
Clear FIFO interrupt: Read RXFIFO Empty status	GQSPI_ISR	RX_FIFO_Empty	`0x104`	11	Read operation
If read RXFIFO empty status == TRUE					
Reset FIFO	GQSPI_FIFO_CTRL	RST_TX_FIFO | RST_GEN_FIFO	`0x14C`	1:0	`11b`
ENDIF					
If read RXFIFO empty status == FALSE					
Switch to I/O mode	GQSPI_CFG	MODE_EN	`0x100`	31:30	`00`
Clear RX_FIFO	GQSPI_FIFO_CTRL	RST_RX_FIFO	`0x14C`	2	`1`
ENDIF					
If DMA read mode == TRUE					
Enable DMA	GQSPI_CFG	MODE_EN	`0x100`	31:30	`10b`
ENDIF					
Disable device	GQSPI_En	GQSPI_EN	`0x114`	0	`0`
END: Quad-SPI PSU abort					

Table 24-26: **Generic Quad-SPI Set Options**

Task	Register	Register Field	Register Offset	Bits	Value
For clock active-Low option	GQSPI_CFG	CLK_POL	`0x100`	1	`1`
For clock phase option	GQSPI_CFG	CLK_PH	`0x100`	2	`1`
For star manual mode option	GQSPI_CFG	GEN_FIFO_START_MODE	`0x100`	29	`1`

Table 24-27: **Generic Quad-SPI Interrupt Transfer**

Task	Register	Register Field	Register Offset	Bits	Value
Enable device	GQSPI_En	GQSPI_EN	`0x114`	0	`1`
Select slave by writing appropriate values to GQSPI_GEN_FIFO.					
Select bus width, stripe configuration values in to GQSPI_GEN_FIFO.					
If byte count is less than 8, follow this next step.					
Do the transfer in I/O mode	GQSPI_CFG	MODE_EN	`0x100`	31:30	`00b`
If transmission, fill the write buffers and follow these next steps.					
Enable TX	GQSPI_GEN_FIFO	TX	`0x140`	16	`1`
Select data transfer	GQSPI_GEN_FIFO	Data transfer	`0x140`	8	`1`
Write data	GQSPI_TXD	TX_DATA	`0x11C`	31:0	Write buffer address
If reception, fill the write buffers and follow these next steps.					
Disable TX	GQSPI_GEN_FIFO	TX	`0x140`	16	`0`
Enable RX	GQSPI_GEN_FIFO	RX	`0x140`	17	`1`
Select data transfer	GQSPI_GEN_FIFO	Data transfer	`0x140`	8	`1`
If DMA is requested, follow these next steps.					
Write destination address	QSPIDMA_DST_ADDR	ADDR	`0x800`	31:2	Destination address
Write address MSB	QSPIDMA_DST_ADDR_MSB	ADDR_MSB	`0x828`	11:0	MSB of adder
Write size of DMA transfer	QSPIDMA_DST_SIZE	SIZE	`0x804`	28:2	Size of data
For dummy transfer, follow these next steps.					
Disable TX	GQSPI_GEN_FIFO	TX	`0x140`	16	`0`
Disable RX	GQSPI_GEN_FIFO	RX	`0x140`	17	`0`
Select data transfer	GQSPI_GEN_FIFO	Data transfer	`0x140`	8	`1`
For both dummy and transfer, follow these next steps.					
Enable TX	GQSPI_GEN_FIFO	TX	`0x140`	16	`1`
Enable RX	GQSPI_GEN_FIFO	RX	`0x140`	17	`1`
Select data transfer	GQSPI_GEN_FIFO	Data transfer	`0x140`	8	`1`

Table 24-27: **Generic Quad-SPI Interrupt Transfer *(Cont'd)***

Task	Register	Register Field	Register Offset	Bits	Value
Write data	GQSPI_TXD	TX_DATA	`0x11C`	31:0	Write buffer address
If DMA read mode enabled, follow these next steps.					
Write destination address	QSPIDMA_DST_ADDR	ADDR	`0x800`	31:2	Destination address
Write address MSB	QSPIDMA_DST_ADDR_MSB	ADDR_MSB	`0x828`	11:0	MSB of adder
Write size of DMA transfer	QSPIDMA_DST_SIZE	SIZE	`0x804`	28:2	Size of data
If byte count is less than 256, follow these next steps.					
Write size of DMA transfer	QSPIDMA_DST_SIZE	SIZE	`0x804`	28:2	Size of data
Write IMM data count	GQSPI_GEN_FIFO	IMM	`0x140`	7:0	Data count
Else:					
Write exponent entries until all bytes over	GQSPI_GEN_FIFO	EXP	`0x140`	9	Exponent entry count
Write immediate entries left	GQSPI_GEN_FIFO	IMM	`0x140`	7:0	Data count
If I/O mode selected, follow this next step.					
Write dummy entry	GQSPI_GEN_FIFO	GEN_DATA	`0x140`	19:0	`0`
If manual start mode enabled, follow these next steps.					
Manual start	GQSPI_CFG	GEN_FIFO_START_MODE	`0x100`	29	`1`
Enable interrupts	GQSPI_IER	TX_FIFO_not_full | TX_FIFO_EMPTY | RX_FIFO_not_empty | Gen_FIFO_Empty | RX_FIFO_EMPTY	`0x108`	11, 7, 4, 8, and 2	`1`
If read mode DMA enabled, follow this next step.					
Clear done bit	QSPIDMA_DST_I_EN	Done	`0x818`	2	`1`

Table 24-28: **Generic Quad-SPI Polled Transfer**

Task	Register	Register Field	Register Offset	Bits	Value
Enable device	GQSPI_En	GQSPI_EN	`0x114`	0	`1`
Select slave by writing appropriate values to GQSPI_GEN_FIFO.					
Select bus width, stripe configuration values in to GQSPI_GEN_FIFO.					
If byte count is less than 8, the follow these next steps.					
Do the transfer in I/O mode	GQSPI_CFG	MODE_EN	`0x100`	31:30	`00b`
If transmission, fill the write buffers and follow these next steps.					
Enable TX	GQSPI_GEN_FIFO	TX	`0x140`	16	`1`
Select data transfer	GQSPI_GEN_FIFO	Data transfer	`0x140`	8	`1`
Write data	GQSPI_TXD	TX_DATA	`0x11C`	31:0	Write buffer address
If reception, fill the write buffers and follow these next steps.					
Disable TX	GQSPI_GEN_FIFO	TX	`0x140`	16	`0`
Enable RX	GQSPI_GEN_FIFO	RX	`0x140`	17	`1`
Select data transfer	GQSPI_GEN_FIFO	Data transfer	`0x140`	8	`1`
If DMA is requested, then follow these next steps.					
Write destination address	QSPIDMA_DST_ADDR	ADDR	`0x800`	31:2	Destination address
Write address MSB	QSPIDMA_DST_ADDR_MSB	ADDR_MSB	`0x828`	11:0	MSB of adder
Write size of DMA transfer	QSPIDMA_DST_SIZE	SIZE	`0x804`	28:2	Size of data
For dummy transfer follow these next steps.					
Disable TX	GQSPI_GEN_FIFO	TX	`0x140`	16	`0`
Disable RX	GQSPI_GEN_FIFO	RX	`0x140`	17	`0`
Select data transfer	GQSPI_GEN_FIFO	Data transfer	`0x140`	8	`1`
For both dummy and transfer follow these next steps.					
Enable TX	GQSPI_GEN_FIFO	TX	`0x140`	16	`1`
Enable RX	GQSPI_GEN_FIFO	RX	`0x140`	17	`1`
Select data transfer	GQSPI_GEN_FIFO	Data Xfer	`0x140`	8	`1`
Write data	GQSPI_TXD	TX_DATA	`0x11C`	31:0	Write buffer address
If DMA read mode enabled, follow these next steps.					
Write destination address	QSPIDMA_DST_ADDR	ADDR	`0x800`	31:2	Destination address
Write address MSB	QSPIDMA_DST_ADDR_MSB	ADDR_MSB	`0x828`	11:0	MSB of address
Write size of DMA transfer	QSPIDMA_DST_SIZE	SIZE	`0x804`	28:2	Size of data

Table 24-28: **Generic Quad-SPI Polled Transfer *(Cont'd)***

Task	Register	Register Field	Register Offset	Bits	Value
If byte count is less than 256, follow these next steps.					
Write size of DMA transfer	QSPIDMA_DST_SIZE	SIZE	0x804	28:2	Size of data
Write IMM data count	GQSPI_GEN_FIFO	IMM	0x140	7:0	Data count
Else:					
Write exponent entries until all bytes over	GQSPI_GEN_FIFO	EXP	0x140	9	Exponent entry count
Write immediate entries left	GQSPI_GEN_FIFO	IMM	0x140	7:0	Data count
If I/O mode is selected.					
Write dummy entry	GQSPI_GEN_FIFO	GEN_DATA	0x140	19:0	0
If manual start mode enabled, follow these next steps.					
Manual start	GQSPI_CFG	GEN_FIFO_START_MODE	0x100	29	1
If more data is left for transfer, follow these next steps.					
Read ISR	GQSPI_ISR	All	0x104	31:0	Read operation
If TX_FIFO_not_full bit is set, follow these next steps.					
Write data	GQSPI_TXD	TX_DATA	0x11C	31:0	Write buffer address
If read mode DMA selected and RX buffer is not null, follow these next steps.					
Read DMA ISR	QSPIDMA_DST_I_STS	All	0x114	7:1	Read operation
If done bit is set until the TX buffer is empty and the FIFO empty bits are cleared, follow these next steps.					
Write back the read value to DMA ISR	QSPIDMA_DST_I_STS	All	0x114	7:1	The value read in the previous operation
Read using I/O mode	GQSPI_CFG	MODE_EN	0x100	31:30	00b
If read mode DMA is not selected and the RX buffer is not null, follow these next steps.					
Read RX threshold offset	GQSPI_RX_THRESH	Level_RX_FIFO	0x12C	5:0	Read
If the RX buffer is not empty, follow these next steps.					
Read remaining bytes	GQSPI_RXD	RX_DATA	0x120	31:0	Read
Read over					
Deselect slave by writing appropriate values to GQSPI_GEN_FIFO.					

Table 24-28: **Generic Quad-SPI Polled Transfer *(Cont'd)***

Task	Register	Register Field	Register Offset	Bits	Value
If manual start mode enabled, follow these next steps.					
Manual start	GQSPI_CFG	GEN_FIFO_START_MODE	`0x100`	29	`1`
Wait until the FIFO empty flag is false.					
Disable device	GQSPI_En	GQSPI_EN	`0x114`	0	`0`

Table 24-29: **Generic Quad-SPI Flash Read ID**

Task	Register	Register Field	Register Offset	Bits	Value
Fill the write buffer with a READ_ID command (`9Fh`) and all required parameters like SPI bus width, mode.					
If polled mode, perform steps mentioned in generic Quad-SPI polled transfer (Table 24-28).					
If interrupt mode perform steps mentioned in generic Quad-SPI interrupt transfer (Table 24-27).					

Table 24-30: **Generic Quad-SPI Flash Erase**

Task
Select slave by writing appropriate values to GQSPI_GEN_FIFO.
For bulk erase:
Fill the write buffer with a write enable command (`06h`).
Perform a generic Quad-SPI polled transfer for polled mode.
Perform a generic Quad-SPI interrupt transfer for interrupt mode.
Fill the write buffer with the bulk erase command (`C7h`).
Perform a generic Quad-SPI polled transfer for polled mode.
Perform a generic Quad-SPI interrupt transfer for interrupt mode.
Perform the following steps until the flash status becomes `1`.
Fill the write buffer with the read status command (`05h`).
Perform a generic Quad-SPI polled transfer for polled mode.
Perform a generic Quad-SPI interrupt transfer for interrupt mode.
For die erase:
Fill the write buffer with the write enable command (`06h`).
Perform a generic Quad-SPI polled transfer for polled mode.
Perform a generic Quad-SPI interrupt transfer for interrupt mode.
Fill the write buffer with the die erase command (`C4h`).
Perform a generic Quad-SPI polled transfer for polled mode.
Perform a generic Quad-SPI interrupt transfer for interrupt mode.
Perform the following steps until flash status becomes `1`.
Fill the write buffer with the read status command (`05h`).

Table 24-30: **Generic Quad-SPI Flash Erase *(Cont'd)***

Task
Perform a generic Quad-SPI polled transfer for polled mode.
Perform a generic Quad-SPI interrupt transfer for interrupt mode.

Table 24-31: **Generic Quad-SPI Flash Write**

Task
Select slave by writing the appropriate values to GQSPI_GEN_FIFO.
For flash write:
Fill the write buffer with the write enable command (`06h`).
Perform a generic Quad-SPI polled transfer for polled mode.
Perform a generic Quad-SPI interrupt transfer for interrupt mode.
Fill the write buffer with a write command (`2` or `12h`).
Perform a generic Quad-SPI polled transfer for polled mode.
Perform a generic Quad-SPI interrupt transfer for interrupt mode.
Perform the following steps until flash status becomes `1`.
Fill the write buffer with the read status command (`05h`).
Perform a generic Quad-SPI polled transfer for polled mode.
Perform a generic Quad-SPI interrupt transfer for interrupt mode.
Select slave by writing the appropriate values to GQSPI_GEN_FIFO.
For flash read:
Fill the write buffer with a write command (`6Bh` or `6Ch`).
Perform a generic Quad-SPI polled transfer for polled mode.
Perform a generic Quad-SPI interrupt transfer for interrupt mode.

Legacy Quad-SPI Controller Programming

The legacy Quad-SPI controller is programmed to provide a linearly addressable memory space for reads and writes done by system masters with access provided via the XPPU protection unit.

Linear Addressing Mode (Memory Reads)

The sequence of operations for data reads in linear addressing mode is as follows:

1. **Set manual start enable to auto mode**. Set qspi.Config[Man_start_en] = 0.
2. **Assert the chip select**. Set qspi.Config[PCS] = 0.
3. **Program the configuration register for linear addressing mode**. The supported read command codes and the recommended configuration register settings (qspi.LQSPI_CFG) are listed in Table 24-6.
4. **Enable the controller**. Set qspi.Enable[SPI_EN] = 1.
5. **Read data from the linear address memory region**. The memory range depends on the size and number of devices. The range is from `0xC000_0000` to `0xDFFF_FFFF`.
6. **Disable the controller**. Set qspi.Enable[SPI_EN] = 0.
7. **De-assert chip select**. Set qspi.Config[PCS] = 1.

MIO Signals

The Quad-SPI flash memory signals are routed through the MIO multiplexer to the MIO device pins. The sides of the dual controller port can be individually enabled or can operate together as an 8-bit I/O interface. The Quad-SPI flash memory signals are routed to the MIO pins as listed in Table 24-32.

Table 24-32: **Quad-SPI Flash Interface Signals**

Quad-SPI Flash Memory Interface			MIO Pin				Controller Default Input Value
Data Mode			Quad-SPI0 (lower)	Quad-SPI1 (upper)	I/O	Name	
1-Bit Data	2-Bit Data	4-Bit Data					
Chip select			5	7	O	SS_b	~
Serial clock			0	12	O	SCLK	~
Optional feedback clock			6		O	LPBK_CLK	~
MOSI	I/O 0	I/O 0	4	8	I/O	IO[0]	0
MISO	I/O 1	I/O 1	1	9	I/O	IO[1]	0
Write protect		I/O 2	2	10	I/O	IO[2]	0
Hold		I/O 3	3	11	I/O	IO[3]	0

NAND Memory Controller

Introduction

This chapter describes the architecture and features of the Zynq® UltraScale+™ MPSoC NAND controller. Defining the NAND protocol is outside the scope of this document, and knowledge of the specifications is assumed.

Features

- ONFI Specification 3.1.
- Up to a 512 Gb device.
- 8-bit I/O width with two chip enable.
- SDR and NV-DDR data interfaces.
- Boot mode support.
- Multi-LUN/DIE operations.
- Full access to spare area.
- Supports SLC flash memory with ECC algorithms:
  - Hamming code with 1-bit error correction and 2-bit error detection.
  - Bose-Chaudhuri-Hocquenghem (BCH) code with 4-bit, 8-bit, 12-bit, and 24-bit error correction.

The NAND flash controller configuration and operational registers are programmed via its AXI slave interface. The block supports the open NAND flash interface working group (ONFI) standards 1.0, 2.0, 2.1, 2.2, 2.3, 3.0, and 3.1. The controller handles all the command, address, and data sequences, manages all the hardware protocols, and allows access NAND flash memory simply by reading or writing into the operational registers. The NAND DMA controller accesses system memory using its AXI master interface.

See Answer Record 65463 for Xilinx tested and supported NAND devices.

Functional Description

Figure 25-1 shows the NAND flash AXI functional block diagram.

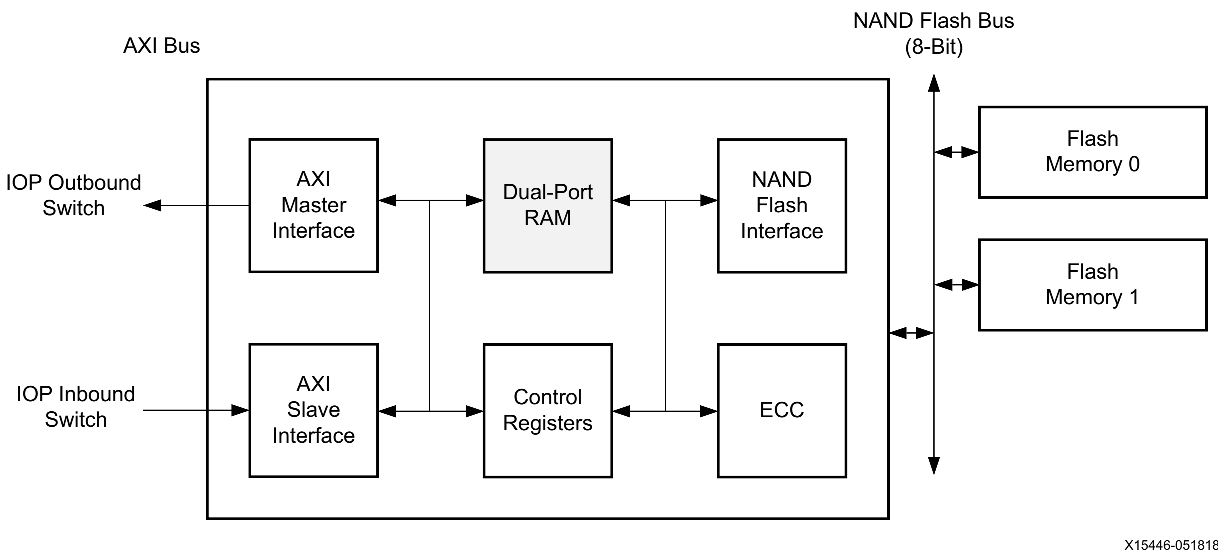


Figure 25-1: **NAND Flash AXI Functional Block Diagram**

NAND Flash Interface

The NAND flash interface handles all the command, address, and data sequences, and manages all the hardware protocols for ONFI 1.0, 2.0, 2.1, 2.2, 2.3, 3.0, and 3.1, and provides an 8-bit interface to the flash memories. The interface supports a maximum of 512 Gb of NAND flash memory. SDR and NV-DDR data interfaces are supported. Timing modes (0-5) are supported for both SDR and NV-DDR.

Dual-port RAM

The dual-port RAM block has handshake logic to communicate with the AXI interface and on the other side communicate with the flash interface. The typical RAM size is 256 x 32 to support block sizes of 512 bytes. The FIFO depth is configurable.

ECC

The ECC module provides error detection and correction support for single-level cell (SLC) flash memory. ECC supports Hamming code with 1-bit error correction, 2-bit error detection, and BCH code with 4-bit, 8-bit, 12-bit, and 24-bit error detection.

RECOMMENDED: *Skip the blank check operation, proceed with writing the data, and verify the written data by reading it back. This is because the NAND controller does not update the ECC to a spare area for erase commands. ECC failures can occur when trying to read erased data.*

Control Registers

The host processor controls the configuration and operation of the NAND flash controller through the control registers. Configuration includes the set up time (T_{CCS}, T_{DQSQ}, T_{DS}), memory configuration (address, page size, packet size, packet count), and timing modes (SDR and NV-DDR). The control registers also provide operating status such as busy and data-ready signals.

AXI Interface

The AXI interface provides the system bus interface.

AXI Master Interface

The AXI master interface transfers boot code from the NAND flash memory to the system memory during system power-up. The NAND flash controller acts as a master during a memory DMA mode of transaction. When the AXI master interface places control signals on the AXI bus depends upon the FIFO status. During a write transaction, the AXI master interface reads data from system memory and stores it in the FIFO. During a read transaction, the AXI master interface reads data from the FIFO and writes data into system memory. The AXI master interface asserts a DMA interrupt when the DMA buffer boundary is reached.

The DMA memory transactions will be routed to the CCI.

AXI Slave Interface

The AXI slave block contains the operational registers. A processor connecting to the custom interface can control the operation of the NAND flash controller through the NAND flash control registers. The flash memory read/write operations can be performed through the NAND flash interface.

Address Aliasing

The NAND controller checks only for address bits [31:15] and [7:0], and ignores the address bits [14:8]. Consequently, addresses assigned to the NAND controller will alias to the 256B register address space.

Register Overview

Table 25-1 lists the NAND memory controller registers and the sequence of NAND flash devices.

Table 25-1: **Register Address Mapping for NAND Flash Controller**

Register Type	Register Name	Description
Packet control register	Packet_Register	Packet register allows control of packet count and size.
Memory bank register	Memory_Address_Register1	32-bit argument points to the flash memory area.
Memory bank register	Memory_Address_Register2	Memory address register 2.
NAND command register	Command_Register	Command register to configure DMA transfer and page size.
NAND program register	Program_Register	Program register.
Interrupt register	Interrupt_Status_Enable_Register	Interrupt status enable register.
Interrupt register	Interrupt_Signal_Enable_Register	Interrupt signal enable register.
Interrupt register	Interrupt_Status_Register	Interrupt status register.
Status register	Ready_Busy	Ready busy register.
System address register	DMA_system_address1_register	DMA system address register.
Status register	Flash_Status_Register	Flash status.

Table 25-1: **Register Address Mapping for NAND Flash Controller *(Cont'd)***

Register Type	Register Name	Description
NAND timing information register	Timing_Register	Sets timing for the NV-DDR mode. When operating in NV-DDR mode, the data might be sampled incorrectly within a FIFO in the controller leading to a data comparison error. The dqs_in phase can be shifted by 1-tap (around 500 ps) when meta-stability is observed using the tap delay register (Timing_Register[dqs_buff_sel_in]) configuration.
Data port register	Buffer_Data_Port_Register	NAND flash internal buffer access register.
ECC register	ECC_Register	ECC register.
ECC register	ECC_Error_Count_Register	ECC error count register.
ECC register	ECC_Spare_Command_Register	ECC spare command register.
Error status register	Error_count_1bit_register	Error count 1-bit register.
Error status register	Error_count_2bit_register	Error count 2-bit register.
Error status register	Error_count_3bit_register	Error count 3-bit register.
Error status register	Error_count_4bit_register	Error count 4-bit register.
System address register	DMA_system_address0_register	DMA system address register.
DMA register	DMA_buffer_boundary_register	DMA buffer boundary register.
CPU release register	CPU_Release_Register	CPU release register.
Error status register	Error_count_5bit_register	Error count 5-bit register.
Error status register	Error_count_6bit_register	Error count 6-bit register.
Error status register	Error_count_7bit_register	Error count 7-bit register.
Error status register	Error_count_8bit_register	Error count 8-bit register.
NAND data interface register	Data_interface_register	Sets SDR mode and NV-DDR mode.

Clocks and Resets

The controller and I/O interface are driven by the reference clock (NAND_REF_CTRL). The controller's interconnect also requires an APB interface clock (LSBUS clock). Both of these clocks always come from the PS clock subsystem.

LSBUS Clock

The LSBUS clock runs asynchronous to the NAND reference clock.

Reference Clock

The reference clock can be generated based on the generic clocking diagram shown in Figure 37-4.

The input clock source can be selected based on the crl_apb. NAND_REF_CTRL [srcsel] bits, where the source can be from the RPLL, IOPLL, or DPLL. The crl_apb. NAND_REF_CTRL [divisor0] register selects the 6-bit programmable divider 0. The crl_apb. NAND_REF_CTRL [divisor1] register selects the 6-bit programmable divider 1. The crl_apb. NAND_REF_CTRL [clkact] bit selects whether the clock should be gated or enabled.

Resets

The controller reset bits are generated by the PS, Chapter 38, Reset System.

I/O Signal Pins

The NAND flash memory signals are routed to the MIO pins as listed in Table 25-2.

Table 25-2: **NAND Interface Signals**

NAND Signals	MIO Pins		I/O	Signal Name	Default Value
	Option 1	Option 2			
Chip enable 1	9	26	O	NFC_CE[1]	-
Ready/busy 0	10	27	I	NFC_RB_n[0]	0
Ready/busy 1	11	28	I	NFC_RB_n[1]	0
Data strobe	12	32	I/O	NFC_DQS_OUT	-
Chip enable 0	13	13	O	NFC_CE[0]	-
Command latch enable	14	14	O	NFC_CLE	-

Table 25-2: **NAND Interface Signals *(Cont'd)***

NAND Signals	MIO Pins		I/O	Signal Name	Default Value
	Option 1	Option 2			
Address latch enable	15	15	O	NFC_ALE	-
Data/address/CMD 0	16	16	I/O	NFC_DQ_OUT[0]	0
Data/address/CMD 1	17	17	I/O	NFC_DQ_OUT[1]	0
Data/address/CMD 2	18	18	I/O	NFC_DQ_OUT[2]	0
Data/address/CMD 3	19	19	I/O	NFC_DQ_OUT[3]	0
Data/address/CMD 4	20	20	I/O	NFC_DQ_OUT[4]	0
Data/address/CMD 5	21	21	I/O	NFC_DQ_OUT[5]	0
Write enable	22	22	O	NFC_WE_B	-
Data/address/CMD 6	23	23	I/O	NFC_DQ_OUT[6]	0
Data/address/CMD 7	24	24	I/O	NFC_DQ_OUT[7]	0
Read enable	25	25	O	NFC_RE_n	-

Figure 25-2 shows a block diagram of a single NAND flash memory connected to the NAND controller.

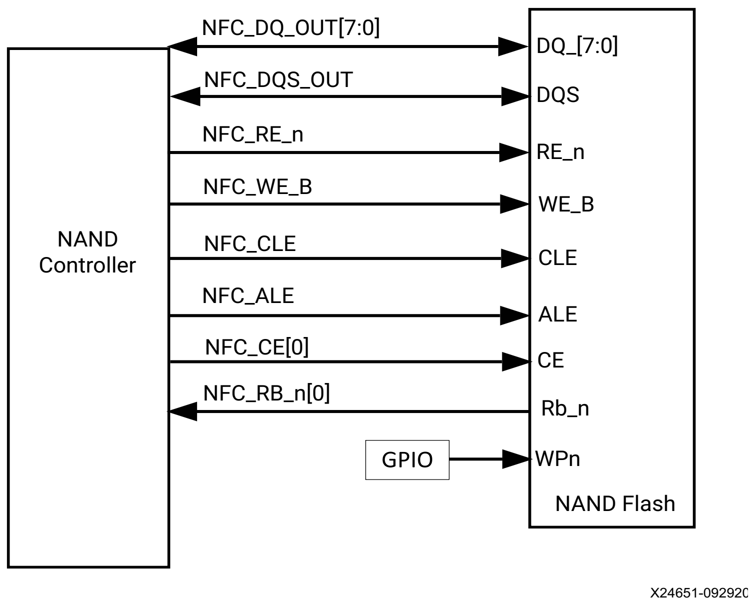


Figure 25-2: **Single NAND Device Wiring Diagram**

Figure 25-3 shows a block diagram of two NAND flash memories connected to the NAND controller.

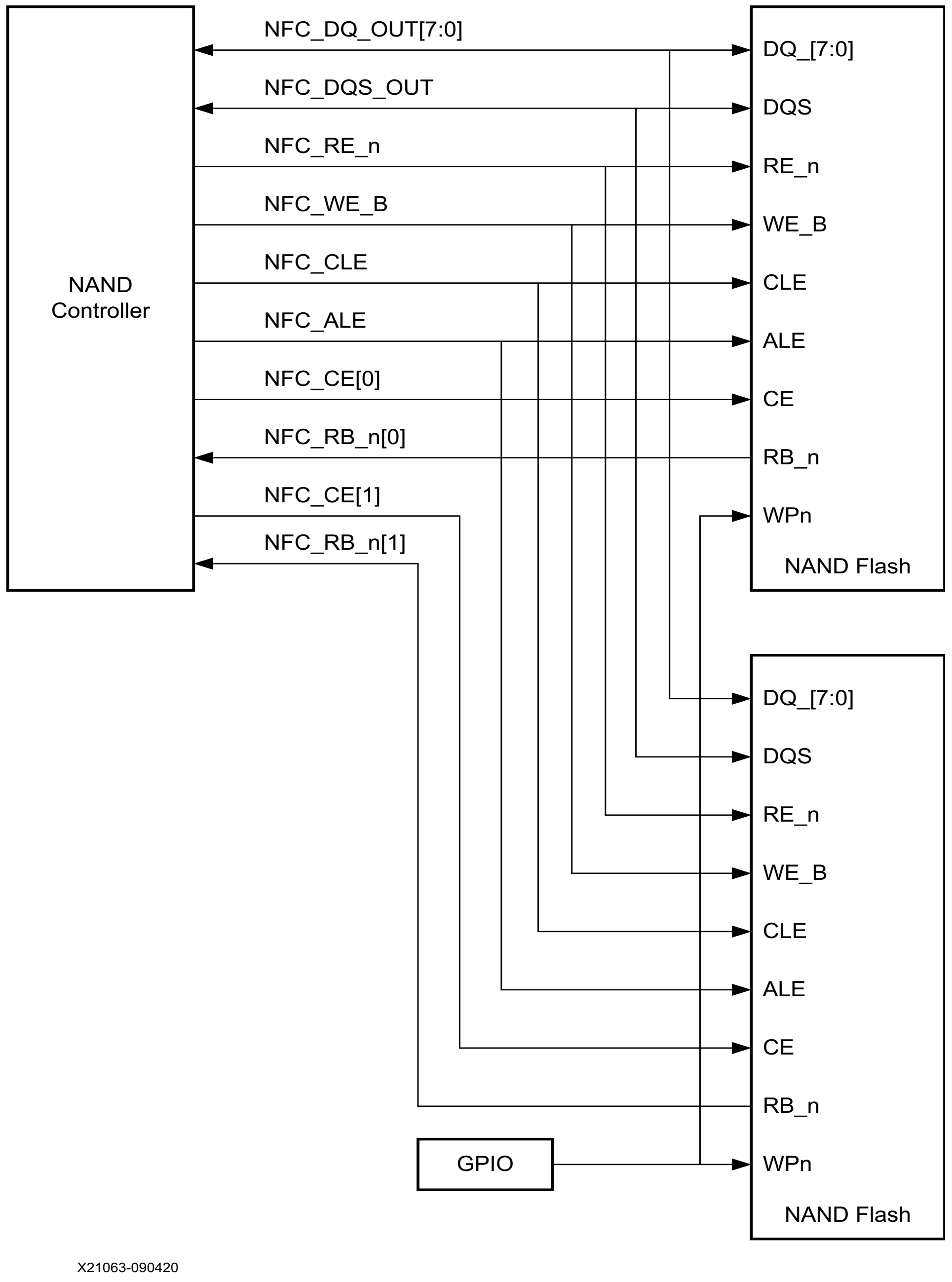


Figure 25-3: **Two NAND Flash Device Wiring Diagram**

Programming Model

Flash Initialization

Table 25-3: **Flash Initialization Program Steps**

Task	Register	Register Field	Register Offset	Bits	Value (Binary)
Clear data interface register	Data_interface_register	All	`0x06C`	10:0	`0x00`
Clear DMA buffer boundary register	DMA_buffer_boundary_register	All	`0x054`	3:0	`0x00`
Perform the following operations for all targets.					
Reset the device (see Reset the Target Device (ONFI Reset)).					
Read ONFI ID (see Read ONFI ID).					
Verify ONFI ID.					
Read mandatory parameter pages (three are mandatory) and perform the following three steps three times.					
If first parameter: • Read parameter page (see Read ONFI Parameters Page). • Else change read column (see Change Read Column). • Check CRC.					
If first target, fill the geometry.					

Reset the Target Device (ONFI Reset)

Table 25-4: **Reset Target Device Program Steps**

Task	Register	Register Field	Register Offset	Bits	Value (Binary)
Enable transfer complete interrupt.	Interrupt_Status_Enable_Register	trans_comp_sts_en	`0x014`	2	`1b'1`
Program command register with reset command (`0xFF`), no ECC, and no DMA.	Command_Register	All	`0x0C`	31:0	`0x0000FF00`
TRAINING: *Select the device.*	Memory_Address_Register2	Chip_Select	`0x08`	31:30	Targets chip select value.
Set reset.	Program_Register	Reset	`0x10`	8	`1b'1`
Poll for transfer complete event.	Interrupt_Status_Register	trans_comp_reg	`0x1C`	2	Wait until transfer is completed or wait time is over.
Clear the transmit complete interrupt after transfer completed.	Interrupt_Status_Enable_Register	trans_comp_sts_en	`0x014`	2	`1b'0`
Clear the transmit complete flag after transfer completed.	Interrupt_Status_Register	trans_comp_reg	`0x1C`	2	`1b'1`

Read ONFI ID

Table 25-5: **Read ONFI ID Program Steps**

Task	Register	Register Field	Register Offset	Bits	Value (Binary)
Enable buffer read ready interrupt.	Interrupt_Status_Enable_Register	buff_rd_rdy_sts_en	0x014	1	1b'1
Program ONFI read ID command (0x90) with no ECC, no DMA, and one address cycles.	Command_Register	All	0x0C	31:0	0x01000090
Program column, page, and block address (next two steps).					
Program memory address register 1.	Memory_Address_Register1	All	0x04	31:0	• Program block address in bits 31:25. • Program page address in bits 22:16. • Program column address in bits 12:0.
Program memory address register 2.	Memory_Address_Register2	All	0x008	31:0	Write required values for memory address.
Select the device.	Memory_Address_Register2	Chip_Select	0x08	31:30	Targets chip select value.
Select packet size and count.	Packet_Register	Packet_count | packet_size	0x00	23:0	Required packet size and count.
Set read ID program register.	Program_Register	Read_ID	0x10	6	1b'1
Poll for buffer read ready event.	Interrupt_Status_Register	buff_rd_rdy_reg	0x1C	1	Wait until bit is set or wait time is over.
Enable the transmit complete interrupt after transfer completed.	Interrupt_Status_Enable_Register	trans_comp_sts_en	0x014	2	1b'1
Clear buffer read ready interrupt.	Interrupt_Status_Register	buff_rd_rdy_reg	0x1C	1	1b'1

Table 25-5: **Read ONFI ID Program Steps *(Cont'd)***

Task	Register	Register Field	Register Offset	Bits	Value (Binary)
Read packet data.	Buffer_Data_Port_Register	Data_Port_Register	`0x030`	31:0	Read until all data is received.
Poll for transfer complete event.	Interrupt_Status_Register	trans_comp_reg	`0x1C`	2	Wait until transfer is completed or wait time is over.
Clear the transmit complete interrupt after transfer completed.	Interrupt_Status_Enable_Register	trans_comp_sts_en	`0x014`	2	`1b'0`
Clear the transmit complete flag after transfer completed.	Interrupt_Status_Register	trans_comp_reg	`0x1C`	2	`1b'1`

Read ONFI Parameters Page

Table 25-6: **Read ONFI Parameters Page**

Task	Register	Register Field	Register Offset	Bits	Value (Binary)
Enable buffer read ready interrupt.	Interrupt_Status_Enable_Register	buff_rd_rdy_sts_en	`0x014`	1	`1b'1`
Program read parameter page command (`0xEC`) with no ECC, no DMA, and one address cycle.	Command_Register	All	`0x0C`	31:0	`0x010000EC`
Program column, page, and block address (next two steps).					
Program memory address register 1.	Memory_Address_Register1	All	`0x04`	31:0	• Program block address in bits 31:25. • Program page address in bits 22:16. • Program column address in 12:0 bits.

Table 25-6: **Read ONFI Parameters Page *(Cont'd)***

Task	Register	Register Field	Register Offset	Bits	Value (Binary)
Program memory address register 2.	Memory_Address_Register2	All	`0x008`	31:0	Write required values for memory address.
Select the device.	Memory_Address_Register2	Chip_Select	`0x08`	31:30	Targets chip select value.
Select packet size and count (256).	Packet_Register	Packet_count | packet_size	`0x00`	23:0	Required packet size and count.
Set read parameter page in program register.	Program_Register	Read_Parameter_Page	`0x10`	7	`1b'1`
Poll for buffer read ready event.	Interrupt_Status_Register	buff_rd_rdy_reg	`0x1C`	1	Wait until bit is set or wait time over.
Enable the transmit complete interrupt after transfer completed.	Interrupt_Status_Enable_Register	trans_comp_sts_en	`0x014`	2	`1b'1`
Clear buffer read ready interrupt.	Interrupt_Status_Register	buff_rd_rdy_reg	`0x1C`	1	`1b'1`
Read packet data.	Buffer_Data_Port_Register	Data_Port_Register	`0x030`	31:0	Read until all data received.
Poll for transfer complete event.	Interrupt_Status_Register	trans_comp_reg	`0x1C`	2	Wait until transfer is completed or wait time is over.
Clear the transmit complete interrupt after transfer completed.	Interrupt_Status_Enable_Register	trans_comp_sts_en	`0x014`	2	`1b'0`
Clear the transmit complete flag after transfer completed.	Interrupt_Status_Register	trans_comp_reg	`0x1C`	2	`1b'1`

Change Read Column

Table 25-7: **Change Read Column**

Task	Register	Register Field	Register Offset	Bits	Value (Binary)
If DMA is enabled, enable DMA boundary interrupt.	Interrupt_Status_Enable_Register	dma_int_sts_en | trans_comp_sts_en	`0x014`	6 and 2	`0x44 (hex)`
Else enable buffer read ready interrupt.	Interrupt_Status_Enable_Register	buff_rd_rdy_sts_en	`0x014`	1	`1b'1`
Program change read column command (`0x05`) with no ECC, DMA, and address cycles.	Command_Register	All	`0x0C`	31:0	Program `0x05` with required DMA mode and address cycles.
Set page size.	Command_Register	page_size	`0x0C`	25:23	`3'd0`: 512B `3'd1`: 2 KB `3'd2`: 4 KB `3'd3`: 8 KB `3'd4`: 16 KB `3'd5`: 1 KB `0x16` bit flash support 6-7 - RES
Program column, page, and block address (next two steps).					
Program memory address register 1.	Memory_Address_Register1	All	`0x04`	31:0	• Program block address in bits 31:25. • Program page address in bits 22:16 • Program column address in bits 12:0.
Program memory address register 2.	Memory_Address_Register2	All	`0x008`	31:0	Write required values for memory address.

Table 25-7: **Change Read Column *(Cont'd)***

Task	Register	Register Field	Register Offset	Bits	Value (Binary)
Select packet size and count.	Packet_Register	Packet_count | packet_size	`0x00`	23:0	Required packet size and count.
If DMA enabled, program DMA system address and buffer boundary (following three steps).					
Invalidate the data cache.					
For 64-bit architecture, program higher address word.	DMA_system_address1_register	DMA_system_ address1_register	`0x024`	31:0	Program higher address word.
Program lower address word.	DMA_system_address0_register	DMA_system_ address0_register	`0x50`	31:0	Program lower address word.
Select the device.	Memory_Address_Register2	Chip_Select	`0x08`	31:30	Targets chip select value.
Set read command in program register.	Program_Register	Read	`0x10`	0	`1b'1`
For non-DMA mode, perform following steps until all packets received (next six steps).					
Poll for buffer read ready event.	Interrupt_Status_Register	buff_rd_rdy_reg	`0x1C`	1	Wait until bit is set or wait time is over.
If buffer read ready events are equal to packet count, then enable the transmit complete interrupt after transfer completed.	Interrupt_Status_Enable_Register	trans_comp_sts_en	`0x014`	2	`1b'1`
Else, clear buffer read ready interrupt in status enable register.	Interrupt_Status_Register	buff_rd_rdy_reg	`0x14`	1	`1b'0`
Clear buffer read ready interrupt.	Interrupt_Status_Register	buff_rd_rdy_reg	`0x1C`	1	`1b'1`
Read packet data.	Buffer_Data_Port_Register	Data_Port_Register	`0x030`	31:0	Read until all data received.

Table 25-7: **Change Read Column *(Cont'd)***

Task	Register	Register Field	Register Offset	Bits	Value (Binary)
If buffer read ready events are less than packet count, then enable buffer read ready interrupts and start next iteration, else break the loop here.	Interrupt_Status_Enable_Register	buff_rd_rdy_sts_en	`0x014`	1	`1b'1`
Poll for transfer complete event.	Interrupt_Status_Register	trans_comp_reg	`0x1C`	2	Wait until transfer is completed or wait time is over.
Clear the transmit complete interrupt after transfer completed.	Interrupt_Status_Enable_Register	trans_comp_sts_en	`0x014`	2	`1b'0`
Clear the transmit complete flag after transfer completed.	Interrupt_Status_Register	trans_comp_reg	`0x1C`	2	`1b'1`

XNandPsu_SetEccAddrSize

Table 25-8: **XNandPsu_SetEccAddrSize**

Task	Register	Register Field	Register Offset	Bits	Value (Binary)
Calculate and write ECC_Addr, ECC_Size and Slc_Mlc values and program into ECC register.	ECC_Register	All	`0x034`	27:0	Refer to the register set definitions.
Program BCH mode in memory address register 2.	Memory_Address_Register2	nfc_bch_mode	`0x008`	27:25	`3'b001`: 12-bit ECC `3'b010`: 8-bit ECC `3'b011`: 4-bit ECC `3'b100`: 24-bit ECC

Erase Block

Table 25-9: **Erase Block**

Task	Register	Register Field	Register Offset	Bits	Value (Binary)
Enable transfer complete interrupt.	Interrupt_Status_Enable_Register	trans_comp_sts_en	0x014	2	1b'1
Program command for block erase (0xD060).	Command_Register	All	0x0C	31:0	Program 0xD060 with required DMA mode and address cycles.
Program column, page, and block address (next two steps).					
Program memory address register 1.	Memory_Address_Register1	All	0x04	31:0	• Program block address in bits 31:25. • Program page address in bits 22:16. • Program column address in bits 12:0.
Program memory address register 2.	Memory_Address_Register2	All	0x008	31:0	Write required values for memory address.
Select the device.	Memory_Address_Register2	Chip_Select	0x08	31:0	Targets chip select value.
Set block erase in program register.	Program_Register	Block_Erase	0x10	2	1b'1
Poll for transfer complete event.	Interrupt_Status_Register	trans_comp_reg	0x1C	2	Wait until transfer is completed or wait time is over.
Clear the transmit complete interrupt after transfer completed.	Interrupt_Status_Enable_Register	trans_comp_sts_en	0x014	2	1b'0
Clear the transmit complete flag after transfer completed.	Interrupt_Status_Register	trans_comp_reg	0x1C	2	1b'1

Read Status

Table 25-10: **Read Status**

Task	Register	Register Field	Register Offset	Bits	Value (Binary)
Enable transfer complete interrupt.	Interrupt_Status_Enable_Register	trans_comp_sts_en	`0x014`	2	`1b'1`
Program command for read status (`0x70`).	Command_Register	All	`0x0C`	31:0	`0x00000070`
Select the device.	Memory_Address_Register2	Chip_Select	`0x08`	31:30	Targets chip select value.
Program packet size and packet count.	Packet_Register	Packet_count | packet_size	`0x00`	23:0	Required packet size and count.
Set status in program register.	Program_Register	Read_Status	`0x10`	3	`1b'1`
Poll for transfer complete event.	Interrupt_Status_Register	trans_comp_reg	`0x1C`	2	Wait until transfer is completed or wait time is over.
Clear the transmit complete interrupt after transfer completed.	Interrupt_Status_Enable_Register	trans_comp_sts_en	`0x014`	2	`1b'0`
Clear the transmit complete flag after transfer completed.	Interrupt_Status_Register	trans_comp_reg	`0x1C`	2	`1b'1`
Read flash status register.	Flash_Status_Register	Flash_Status	`0x28`	15:0	Read operation.

Program Page

Table 25-11: **Program Page**

Task	Register	Register Field	Register Offset	Bits	Value (Binary)
Program command for page programming (`0x1080`) with ECC, DMA enabled.	Command_Register	All	`0x0C`	31:0	Program the command with required address cycles.
If DMA is enabled, enable DMA boundary interrupt.	Interrupt_Status_Enable_Register	dma_int_sts_en | trans_comp_sts_en	`0x014`	6 and 2	`0x44` (hex)
Else enable buffer write ready interrupt.	Interrupt_Status_Enable_Register	buff_wr_rdy_sts_en	`0x014`	0	`1b'1`
Set page size.	Command_Register	page_size	`0x0C`	25:23	`3'd0`: 512B `3'd1`: 2 KB `3'd2`: 4 KB `3'd3`: 8 KB `3'd4`: 16 KB `3'd5`: 1 KB `0x16` bit flash support 6-7 - RES
Select packet size and count.	Packet_Register	Packet_count | packet_size	`0x00`	23:0	Required packet size and count.
If DMA enabled, program DMA system address and buffer boundary (following three steps).					
Invalidate the data cache.					
For 64-bit architecture, program higher address word.	DMA_system_address1_register	DMA_system_ address1_register	`0x024`	31:0	Program higher address word.
Program lower address word.	DMA_system_address0_register	DMA_system_ address0_register	`0x50`	31:0	Program lower address word.

Table 25-11: **Program Page *(Cont'd)***

Task	Register	Register Field	Register Offset	Bits	Value (Binary)
Program column, page, and block address (next two steps).					
Program memory address register 1.	Memory_Address_Register1	All	`0x04`	31:0	• Program block address in bits 31:25. • Program page address in bits 22:16. • Program column address in bits 12:0.
Program memory address register 2.	Memory_Address_Register2	All	`0x008`	31:0	Write required values for memory address.
Select the device.	Memory_Address_Register2	Chip_Select	`0x08`	31:30	Targets chip select value.
Set ECC spare command (`0X85`) if hardware ECC enabled.	ECC_Spare_Command_Register	Number_of_ECC_and_Spare_Address_cycles | ECC_Spare_cmd	`0x3c`	30:0	`0X85` for spare command and required address cycles.
Set page program in program register.	Program_Register	Page_Program	`0x10`	4	`1b'1`
For non-DMA mode, perform following steps until all packets received (next six steps).					
Poll for buffer write ready event.	Interrupt_Status_Register	buff_wr_rdy_reg	`0x1C`	0	Wait until bit is set or wait time is over.
If buffer write ready events are equal to packet count, then enable the transmit complete interrupt after transfer completed.	Interrupt_Status_Enable_Register	trans_comp_sts_en	`0x014`	2	`1b'1`
Else, clear buffer write ready interrupt in status enable register.	Interrupt_Status_Register	buff_wr_rdy_sts_en	`0x14`	0	`1b'0`
Clear buffer write ready interrupt.	Interrupt_Status_Register	buff_wr_rdy_reg	`0x1C`	0	`1b'1`

Table 25-11: **Program Page *(Cont'd)***

Task	Register	Register Field	Register Offset	Bits	Value (Binary)
Write packet data.	Buffer_Data_Port_Register	Data_Port_Register	`0x030`	31:0	Write until all data over.
If buffer write ready events are less than packet count, then enable buffer write ready interrupts and start next iteration else break the loop here.	Interrupt_Status_Enable_Register	buff_wr_rdy_sts_en	`0x014`	0	`1b'1`
Poll for transfer complete event.	Interrupt_Status_Register	trans_comp_reg	`0x1C`	2	Wait until transfer is completed or wait time is over.
Clear the transmit complete interrupt after transfer completed.	Interrupt_Status_Enable_Register	trans_comp_sts_en	`0x014`	2	`1b'0`
Clear the transmit complete flag after transfer completed.	Interrupt_Status_Register	trans_comp_reg	`0x1C`	2	`1b'1`

Read Page

Table 25-12: **Read Page**

Task	Register	Register Field	Register Offset	Bits	Value (Binary)
Program command for read page (`0x3000`) with ECC, DMA enabled.	Command_Register	All	`0x0C`	31:0	Program the command with required address cycles.
If DMA is enabled, enable DMA boundary interrupt.	Interrupt_Status_Enable_Register	dma_int_sts_en | trans_comp_sts_en	`0x014`	6 and 2	`0x44` (hex)
Else enable buffer read ready interrupt.	Interrupt_Status_Enable_Register	buff_rd_rdy_sts_en	`0x014`	1	`1b'1`
Enable single bit error and multi-bit error if hardware ECC is enabled.	Interrupt_Status_Enable_Register	err_intrpt_sts_en | mul_bit_err_sts_en	`0x014`	4:3	`2b'3`

Table 25-12: **Read Page *(Cont'd)***

Task	Register	Register Field	Register Offset	Bits	Value (Binary)
Set page size.	Command_Register	page_size	`0x0C`	25:23	`3'd0` - 512B `3'd1` - 2 KB `3'd2` - 4 KB `3'd3` - 8 KB `3'd4` - 16 KB `3'd5` - 1 KB `0x16` bit flash support 6-7 - RES
Program column, page, and block address (next two steps).					
Program memory address register 1.	Memory_Address_Register1	All	`0x04`	31:0	• Program block address in bits 31:25. • Program page address in bits 22:16. • Program column address in bits 12:0.
Program memory address register 2.	Memory_Address_Register2	All	`0x008`	31:0	Write required values for memory address.
Select packet size and count.	Packet_Register	Packet_count | packet_size	`0x00`	23:0	Required packet size and count.
If DMA enabled, program DMA system address and buffer boundary (following three steps).					
Invalidate the data cache.					
For 64-bit architecture, program higher address word.	DMA_system_address1_register	DMA_system_address1 _register	`0x024`	31:0	Program higher address word.
Program lower address word.	DMA_system_address0_register	DMA_system_address0 _register	`0x50`	31:0	Program lower address word.
Select the device.	Memory_Address_Register2	Chip_Select	`0x08`	31:30	Targets chip select value.

Table 25-12: **Read Page *(Cont'd)***

Task	Register	Register Field	Register Offset	Bits	Value (Binary)
Set ECC spare command (0X85) if hardware ECC enabled.	ECC_Spare_Command_Register	Number_of_ECC_and_Spare_Address_cycles | ECC_Spare_cmd	0x3c	30:0	0X85 for spare command and required address cycles.
Set page program in program register.	Program_Register	Read	0x10	0	1b'1
For non-DMA Mode, perform following steps until all packets received (next six steps).					
Poll for buffer read ready event.	Interrupt_Status_Register	buff_rd_rdy_reg	0x1C	1	Wait until bit is set or wait time is over.
If buffer read ready events are equal to packet count, then enable the transmit complete interrupt after transfer completed.	Interrupt_Status_Enable_Register	trans_comp_sts_en	0x014	2	1b'1
Else, clear buffer read ready interrupt in status enable register.	Interrupt_Status_Register	buff_rd_rdy_sts_en	0x14	1	1b'0
Clear buffer read ready interrupt.	Interrupt_Status_Register	buff_rd_rdy_reg	0x1C	1	1b'1
Read packet data.	Buffer_Data_Port_Register	Data_Port_Register	0x030	31:0	Read until all data received.
If buffer read ready events are less than packet count, then enable buffer read ready interrupt and start next iteration else break the loop here.	Interrupt_Status_Enable_Register	buff_rd_rdy_sts_en	0x014	1	1b'1
Poll for transfer complete event.	Interrupt_Status_Register	trans_comp_reg	0x1C	2	Wait until transfer is completed or wait time is over.

Table 25-12: **Read Page *(Cont'd)***

Task	Register	Register Field	Register Offset	Bits	Value (Binary)
Clear the transmit complete interrupt after transfer completed.	Interrupt_Status_Enable_Register	trans_comp_sts_en	`0x014`	2	`1b'0`
Clear the transmit complete flag after transfer completed.	Interrupt_Status_Register	trans_comp_reg	`0x1C`	2	`1b'1`
If hardware ECC mode is enabled, check for ECC errors.					
Hamming multi-bit errors					
Read interrupt status.	Interrupt_Status_Register	mul_bit_err_reg	`0x1C`	3	Read
If multi-bit error bit set, clear the status.	Interrupt_Status_Register	mul_bit_err_reg	`0x1C`	3	`1b'1`
Read ECC error count.	ECC_Error_Count_Register	Page_bound_Err_count	`0x38`	16:8	Read
Hamming single-bit or BCH errors					
Read interrupt status.	Interrupt_Status_Register	err_intrpt_reg	`0x1C`	4	Read
If multi-bit error bit set, clear the status.	Interrupt_Status_Register	err_intrpt_reg	`0x1C`	4	`1b'1`
Read ECC error count.	ECC_Error_Count_Register	Page_bound_Err_count	`0x38`	16:8	Read

Change Timing Mode for SDR and NV-DDR

Table 25-13: **Change Timing Mode for SDR and NV-DDR**

Task	Register	Register Field	Register Offset	Bits	Value (Binary)
If the interface is NV-DDR, program the ONFI set feature with the data interface and timing values for all targets.					
If the interface is SDR: • Change clock frequency with SDR CLK 100 MHz. • Update the new data interface and timing mode values in the data interface register.					
Reset all targets (refer to Reset the Target Device (ONFI Reset))					
Set feature with new modes (ONFI Set Feature)					

ONFI Set Feature

Table 25-14: **ONFI Set Feature**

Task	Register	Register Field	Register Offset	Bits	Value (Binary)
Mask all bits in interrupt status enable register.	Interrupt_Status_Enable_Register	ALL	`0x14`	All	`0x00000000`
Enable buffer write ready interrupt.	Interrupt_Status_Enable_Register	Bit 0	`0x14`	1	`1b'1`
Write command into command register.	Command_Register	Number_of_Address_cycles |	`0x0c`	All	`0x110000EF` (HEX)
Program page, column, and block address.	Memory_Address_Register1	All	`0x04`	All	Memory address
	Memory_Address_Register2	All	`0x08`	All	Memory address and modes
Program packet size and packet count.	Packet_Register	All	`0x00`	All	Packet count and size
Set read parameter page.	Program_Register	Set_Features	`0x10`	10	`1b'1`
Write feature to NAND memory.					

Chapter 26

SD/SDIO/eMMC Controller

Introduction

The two SD controllers have the same feature set and can be operated independently. The controller communicates with SDIO devices, SD memory cards, and eMMC cards with up to eight data lines. In SD mode, data transfers in 1-bit and 4-bit modes. In eMMC mode, data transfers in 1-bit, 4-bit, and 8-bit modes. The interface can be routed through the MIO multiplexer to the MIO pins or through the EMIO to the SelectIO pin in the PL.

The controller is accessed by the APU and RPU via the AXI bus. The controller also includes a DMA unit with an internal FIFO to meet throughput requirements.

Features

The controller is compatible with:

- 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 3.0.
- eMMC specification version 4.51.
- MMC specification version 4.51.

System/Host Interfaces

- AXI master, DMA interface.
- AXI slave, PIO data transfers.
- APB slave, register accesses.

SD/SDIO Card Interface

- Operating mode with maximum clock rate:
  - Standard mode (19 MHz)
  - High-speed mode (50 MHz)
  - SDR12 (25 MHz)
  - SDR25 (50 MHz)
  - SDR50 (100 MHz)
  - SDR104 (200 MHz)
  - DDR50 mode (50 MHz)
- Up to 800 Mb/s data rate using four parallel data lines (SDR104 mode).
- Cyclic redundancy check CRC7 for command and CRC16 for data integrity.
- Variable-length data transfers.
- Performs read wait control, suspend/resume operation SDIO card.
- Designed to work with I/O cards, read-only cards, and read/write cards.
- Supports read wait control, suspend/resume operation.

eMMC Card Interface

- Operating mode with maximum clock rate:
  - Standard mode (25 MHz)
  - High-speed SDR mode (50 MHz)
  - High-Speed DDR mode (50 MHz)
  - HS200 mode (200 MHz)
- Up to 1660 Mb/s data rate using 8-bit parallel data lines (HS200 mode).
- Cyclic redundancy check CRC7 for command and CRC16 for data integrity.
- Card detection (insertion/removal).

FIFO Buffer

- Handle the FIFO overrun and underrun condition by stopping the SD clock.

Speed Modes

The SD card speed modes are listed in Table 26-1. The eMMC speed modes are listed in Table 26-2.

Table 26-1: **SD Card Speed Modes**[1]

SD Speed Mode	Data Rate	Bus Width	SD_CLK Frequency in MHz	Throughput in MB/s	SD Interface Voltage	I/O Interface Support
Default speed[2]	Single	1, 4	25	12.5	3.3V	MIO and EMIO
High speed	Single	1, 4	50	25	3.3V	MIO
SDR12	Single	4	25	12.5	1.8V	MIO and EMIO
SDR25	Single	4	50	25	1.8V	MIO
DDR50	Double	4	50	50	1.8V	MIO
SDR50	Single	4	100	50	1.8V	MIO
SDR104	Single	4	200	100	1.8V	MIO

Notes:

1. SD line selection is based on the SD 2.0 or 3.0 compliance. Refer to the *Zynq UltraScale+ MPSoC Processing System Product Guide* (PG201) [Ref 5].
2. In the SD default speed mode with level shifter, the maximum frequency (F_{MAX}) is 19 MHz.

Table 26-2: **eMMC Speed Modes**

eMMC Speed Mode	Data Rate	Bus Width	Frequency in MHz	Throughput in MB/s	Voltage	MIO/EMIO Support
Legacy MMC speed[1][2]	Single	1, 4, 8	25	25	3.3V and 1.8V	MIO and EMIO
High-speed SDR	Single	4, 8	50	50	3.3V and 1.8V	MIO
High-speed DDR	Double	4, 8	50	100	3.3V and 1.8V	MIO
HS200	Single	4, 8	200	200	1.8V	MIO

Notes:

1. Legacy MMC speed relates to default eMMC speed.
2. The default eMMC boots in legacy MMC speed mode only. Your application must switch to the high-speed modes.

Functional Description

Figure 26-1 shows the SD/SDIO/eMMC controller block diagram.

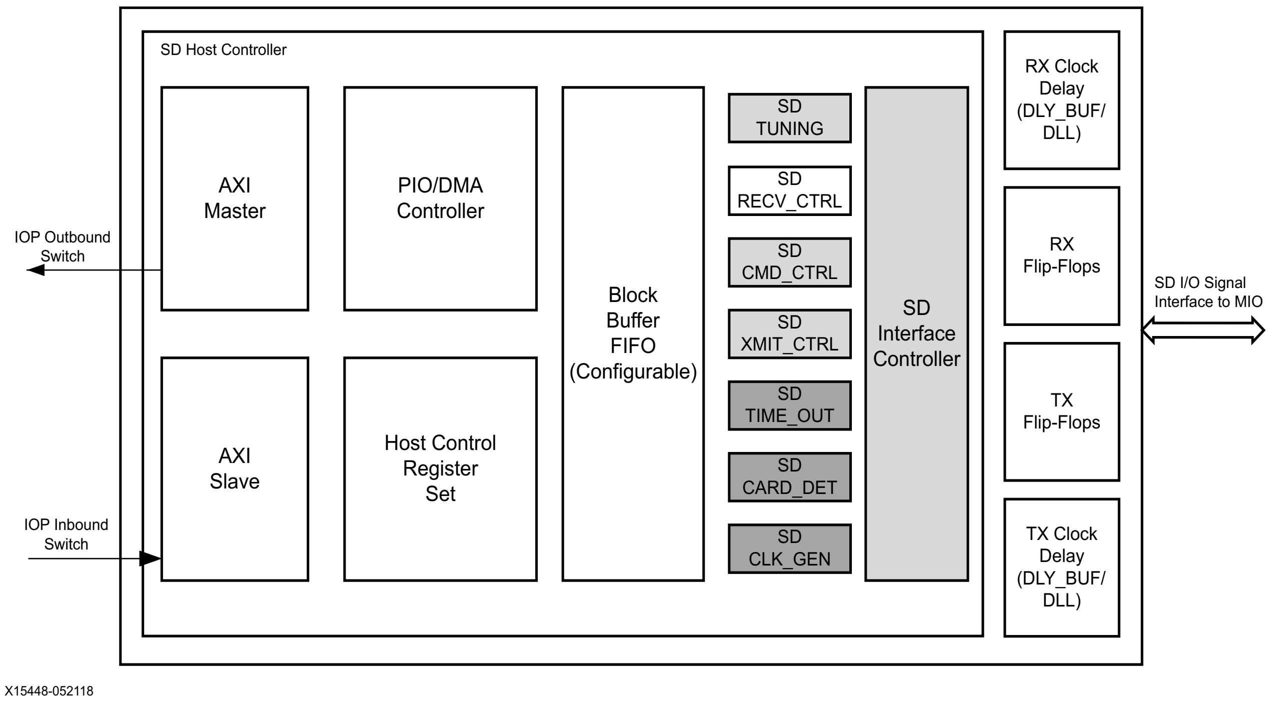


Figure 26-1: **SD/SDIO/eMMC Controller Block Diagram**

Host Interface (Master/Slave)

The host controller interfaces to the system bus using the AXI master and slave interface.

The slave bus is used to access the registers inside the host controller. Also, when operating in PIO mode, the driver can access the SD data port register through this interface. This is the PIO method in which the host driver transfers data using the buffer data port register. The slave bus supports only single transfer access (no burst support). Also, in the case of the AXI interface, the slave bus supports only one outstanding read/write transaction.

The master bus is used by the DMA controller (when using DMA or ADMA2 modes). The DMA controller uses the master DMA interfaces to transfer data between the internal buffer and the system memory and vice versa. The DMA controller also uses the master interface to fetch the descriptors while operating in ADMA2 mode.

Register Set

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 SD/SDIO controller registers are programmed by the processor through the AXI slave interface. Interrupts are generated to the host processor based on the values set in the interrupt status register and interrupt enable registers.

The registers are listed in Table 26-12.

PIO/DMA Controller

The PIO/DMA controller implements the SDMA and ADMA2 engines as defined in the SD host controller specification and maintains the block transfer counts for PIO operation. It 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. The DMA controller implements a separate DMA for SDMA operation and separate DMA for the ADMA2 operation. In addition, it implements a host transaction generator that generates controls for the host master interface.

The DMA memory transactions will be routed to the CCI.

Block Buffer

The dual-port block buffer (read/write on both ports) stores block data during SD transfers. The block buffer size implemented in the design is 4KB. The block buffer uses a circular buffer architecture. One side of the block buffer is interfaced with the DMA controller and operates at the host clock rate while the other side of the block buffer interfaces with the SD control logic and operates at the SD clock rate. During a write transaction (data transferred from the Arm APU/RPU to the SD 3.0/SDIO 3.0 card), the data is fetched from the system memory and is stored in the block buffer. When a block of data is available, the SD control logic transfers it onto the SD interface. The DMA controller continues to fetch additional blocks of data when the block buffer has space. During a read transaction (data transferred from an SD 3.0/SDIO 3.0 card to the APU/RPU), the data from the SD 3.0/SDIO 3.0 card is written into the block buffer, and at the end when the CRC of the block is valid, the data is committed. When a block of data is available, the DMA controller transfers this data to the system memory. The SD control logic meanwhile receives the next block of data, provided that there is space in the block buffer. If the controller cannot accept any data from an SD 3.0/SDIO 3.0/eMMC 4.51 card, then it will issue a read wait (if the card supports a read wait mechanism) to stop the data transfer from the card or by stopping the clock.

Note: When the block buffer size is twice the block size, the block buffer behaves as a ping-pong buffer.

Card Detect

The SD card detect logic monitors the SD_CD# pin for card insertion/removal events. It implements debouncing logic to filter false transitions on the SD_CD# pin. The card insertion and removal events are reported to the SD host register set from which the interrupt is eventually generated.

Figure 26-2 shows the SDHC card detection.

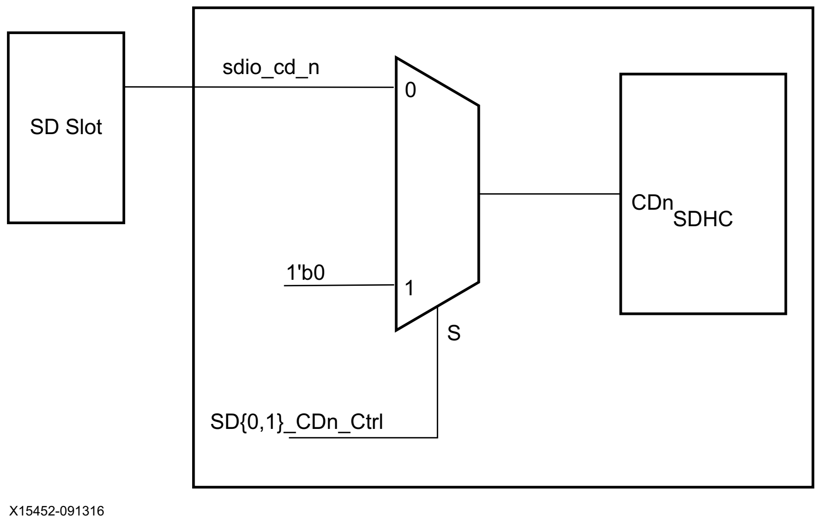


Figure 26-2: **Card Detection in SDHC**

- The SD host controller card detection uses the SD host control register card detect signal bit as the selection bit.
- If the SD control register card detection bit is `1'b1`, then the card is inserted during boot time or an eMMC.
- If the SD control register card detection bit is `1'b0`, then the SD slot interface is used to identify the insertion and removal of the card using the MIO pin.

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.

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.

SD Transmit Control

The SD 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.

SD Receive Control

The SD 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.

Clocks and Resets

Resets

The controller reset bits are generated by the PS, see Chapter 38, Reset System.

Clocking Overview

The SD clock generator generates the SD clock from the reference clock (sdio_ref_clk) based on the controls programmed in the clock control register (SDIO.reg_clockcontrol). These include the clock divide value, SD clock enable, etc. The outputs are the SD_CLK and the SD_CARD clock. The SD_CLK is used by the most of the SD control logic (SD Command Control, SD transmit control, SD tuning block and block buffer). The SD_CARD clock is the same as SD_CLK, except that this is available only when the SD clock enable (SDIO.reg_clockcontrol [clkctrl_sdclkena]) bit is set and is connected to the SD_CLK pin on the SD interface. Figure 26-3 shows the clocking architecture.

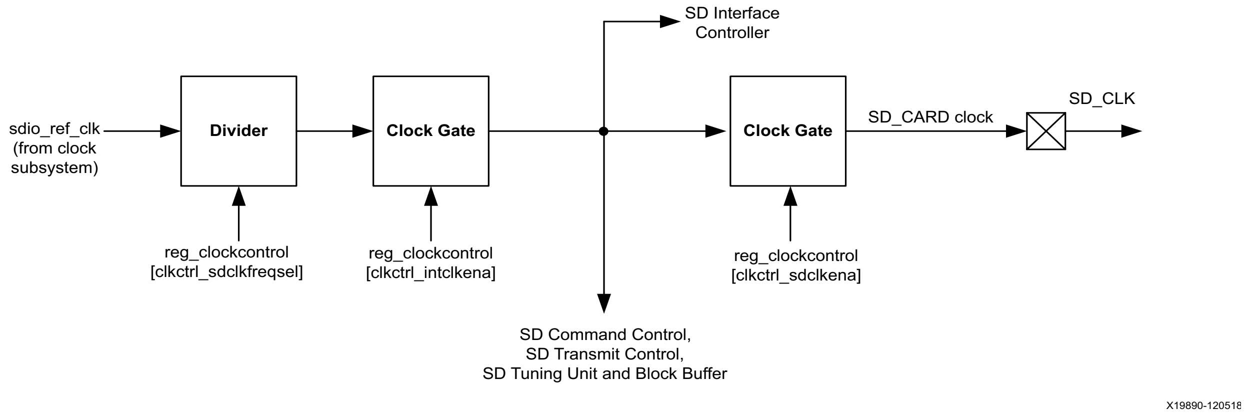

SD Interface Controller
sdio_ref_clk (from clock subsystem)
Divider
Clock Gate
Clock Gate
SD_CARD clock
SD_CLK
reg_clockcontrol [clkctrl_sdclkfreqsel]
reg_clockcontrol [clkctrl_intclkena]
reg_clockcontrol [clkctrl_sdclkena]
SD Command Control, SD Transmit Control, SD Tuning Unit and Block Buffer
X19890-120518

Figure 26-3: **Clocking Architecture**

The host controller supports both full speed and high speed cards. For the high speed card, the host controller should clock out the data at the rising edge of the SDIO clock. For the full speed card, the host controller should clock out the data at the falling edge of the SDIO clock.

The host bus interface (AXI Master/Slave), the host control register set, and the PIO/DMA controller operate on the AXI interface clock.

Reference Clock

The reference clock is generated in the PS clock subsystem. The input clock source can be selected based on the crl_apb.SDIO{0,1}_REF_CTRL[srcsel] bits, where the source can be from the RPLL, IOPLL, or DPLL. The crl_apb.SDIO{0,1}_REF_CTRL[divisor0] register selects the

6-bit programmable divider 0. The crl_apb.SDIO{0,1}_REF_CTRL[divisor1] register selects the 6-bit programmable divider 1.The crl_apb.SDIO{0,1}_REF_CTRL[clkact] bit selects whether the clock should be gated or enabled.

Tuning Unit

The SD PS controller supports auto-tuning for SDR-104,SDR-50 and HS200. Whereas, manual tuning is supported for HSD/SDR25/DDR50 modes and eMMC high-speed SDR/DDR modes in order to find the valid rx clock delay. The tuning unit generates the delay controls to the external delay controller. The tuning unit receives the 64-byte tuning word (SD mode) or 128-byte tuning word (eMMC mode) and maintains a tuning vector to determine the optimal delay. The tuning unit can be configured with the number of supported delay taps (180 maximum). Using this, the tuning unit performs tuning and selects the optimal tap point for the receive clock.

Note: Programming the delay taps non-sequentially might lead to the failing of the auto-tuning. See Answer Record 65676 for the work-around sequence.

Interface Controller

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 from the RXFLOPS unit are latched on the rx_clk (looped back or tuned clock) and output to the receive control unit for further processing.

RX Clock Delay Unit

The RX clock delay unit is used to support receive clock tuning to center align the receive data to the receive clock. There are two modes for delaying the receive clock. The first one is the automatic tuning of the receive clock when operating in SDR104 modes in SD 3.0 or eMMC 4.51 in SDR50 modes. The second one is under manual controls to offset for post-silicon board delays. The manual control is implemented for high-speed mode and SDR25/SDR50/DDR50 modes using the corectrl_itapdlysel and corectrl_itapdlyen signals.

The maximum number of tap delays (phases of the clock) is 180, but the useful number of tap delays is greatly reduced as the clock frequency goes up. A typical design uses four to eight tap delay selections.

The preferred method uses the looped back SD_CLK (rxclk_in) to generate multiple phases of the clock. In the case of a DLL-based approach, this looped back clock is not ideal because the clock itself can dynamically be stopped by the host controller to pause the data reception from the SD/eMMC card. Because the DLL takes longer times to lock the clock, a continuous clock is needed. Because the SD_CLK is a gated version of the internal sd_clk, Xilinx recommends using the sd_clk as the input to the RXCLK delay unit.

TXCLK Delay Unit

The CMD and DAT outputs need to be delayed with regard to the output SD_CLK signal to meet the hold time requirements in various modes of operation. The outgoing SD clock is delayed. The delayed clock is used to flip-flop the CMD/DAT lines and this output is used to drive the SD interface. The SD_CLK output itself is not delayed.

Controller Clocking

The controller supports a range of clock frequencies including the popular 25, 50, 100, and 200 MHz.

Frequencies above 25 MHz automatically switch to use the DLL clock generator, however the minimum DLL clock frequency is 33 MHz. Controller frequencies between 25 and 33 MHz are not supported.

Non-DLL Clock Mode

The non-DLL clock mode is automatically selected by the controller when the clock frequency is 25 MHz or less. The frequency is controlled by the IOU_SLCR.SD_CONFIG_REG1 [SDx_BASECLK] and SDIO.reg_clockcontrol [clkctrl_sdclkfreqsel] bit fields as shown in Table 26-3.

Table 26-3: **Non-DLL Mode Frequencies**

[SDx_BASECLK] (MHz)	[clkctrl_sdclkfreqsel][1]	Actual BASECLK Divider Value[2]	SD Output Frequency[3] (MHz)
200	4	8	25
	5	10	20
	6	12	16.67
100	2	4	25
	3	6	16.67
	4	8	12.5
	5	10	10
50	1	2	25
	2	4	12.5
	3	6	8.3
	4	8	6.26
	5	10	5
25	1	2	12.5
	2	4	6.125
	3	6	4.12
	4	8	3.12
	5	10	2.5

Notes:

1. The [clkctrl_sdclkfreqsel] bit field must not be set to 0.
2. See the description of SDIO.reg_clockcontrol register in the *Zynq UltraScale+ MPSoC Register Reference* (UG1087) [Ref 4].
3. Maximum clock frequencies are specified in the *Zynq UltraScale+ MPSoC Data Sheet: DC and AC Switching Characteristics* (DS925) [Ref 2] data sheet.

DLL Clock Mode

The DLL input frequency to the SD controller comes from the PLL output of the IOPLL or RPLL (not from the VCO output). The SD controller DLL input frequency can be changed by the IOPLL or RPLL in the CRL_APB.DLL_REF_CTRL register. The DLL mode is automatically selected by the SD controller when the SD output clock frequency is more than 25 MHz; however, the minimum DLL clock frequency is 33 MHz. The DLL mode supported SD reference clocks are 50 MHz, 100 MHz, and 200 MHz. The reference clock values are updated in the IOU_SLCR.SD_CONFIG_REG1 base clock. In the DLL mode of operation, the tap delays in the IOU_SLCR.{SD_ITAPDLY and OTAPDLY} registers must be programmed.

The input tap delay programming sequence is as follows using the IOU_SLCR register set.

1. A DLL reset is issued.
2. The sdx_itapchgwin is set (to gate any glitches on the line).
3. The sd0_itapdlyena is enabled and the tap delay values are programmed.
4. The sdx_itapchgwin is cleared after setting the input tap delay.
5. The DLL reset is released. Wait for the DLL to lock.

The output tap delay programming sequence is as follows:

1. A DLL reset is issued.
2. The tap values are programmed.
3. The DLL reset is released. Wait for the DLL to lock.

Note: Refer to the programming sequence in the SD Change Bus Speed section for the DLL reference clock setting.

Transmit CMD/DAT Delay

The TX CMD/DAT delay is used to delay the CMD/DAT lines to avoid a hold time violation in the card due to board layout timing issues. In some cases, the board layout might not be optimal and the CMD/DAT lines might have hold time violations on the card because the SD_CLK and CMD/DAT are source synchronous for the card.

One option is to use delay buffers on each of the CMD and eight DAT lines to provide enough hold margin and use these delayed outputs to the SD interface. This approach has the disadvantage of using nine delay lines and also not having the same delay on each of the CMD/DAT lines. Another option is to delay the internal sd_clk that is being sent out on the clock line (as SD_CLK), and use this delayed clock to flop out the CMD/DAT lines. This approach has the advantage of using only one single DLL/delay on the clock line and provides uniform output delays across the CMD/DAT lines.

The TX clock delay unit uses the delay buffers or a DLL to generate a phase-shifted clock. Using the variable delay output, you can program the tap delay using the external ports. To configure the OTAPDLYSEL, refer to the *Program Sequence for DLL TAP Delay* in Table 26-27. The following are bit fields in SD{0,1}_OTAPDLYSEL.

- sd{0,1}_otapdlysel[5:0]: Used to select the optimum delay from 8 to 45 tap delay lines.

The TXFLOPS unit implements the final stage registers using this delayed clock. The TXFLOPS unit also implements two sets of registers for each of the CMD/DAT lines (one for the positive edge output and another one for the negative edge output).

In the case of DDR, both the flip-flops are used and in SDR mode, only the positive edge is used (DS mode uses the negative edge outputs) when operating in default DS. The outputs are driven on the falling edge of the clock so that the card can have enough setup/hold time when latching the data. In this case, the output tap delay control is not necessary and should be disabled.

When operating in HS modes and other SDR modes, the output data is driven on the rising edge of the clock. The same clock is also output to the card (SD interface). Based on the post-silicon board layout, the card might see hold time violations for the CMD/DAT lines. To avoid this, the output tap delay lines can be programmed under user control.

Receive Clock Tap Delay

The RX clock delay is used for tuning/delaying the receive clock so as to align the clock in the center of the data window. This is used in both auto tuning (in SDR50 and SDR104 mode) and optional manual tuning (for high-speed modes such as DDR).

During read operation, the host controller acts as a receiver, and the data might not be exactly aligned with respect to the clock. The clock signal can be delayed either by auto tuning or manual tuning so that the clock is center aligned to receive data.

For SDR104 mode and for SDR50 mode, automatic tuning is performed and the external controls are cleared i.e., sd{0,1}_itapdlyena is reset. The host controller has an algorithm to correctly find the center of the eye for better timing. The tuning procedure selects one phase of the clock (rxclk_in) for each iteration. At the end of tuning, the right phase of the clock is selected that is in the center of the data.

In other modes (such as DDR50), the manual tuning of the SD_CLK (rxclk_in) can be performed using the external controls.

- sd{0,1}_itapdlysel [7:0]: Used to select the optimum delay from 30 to 180 tap delay lines.
- sd{0,1}_itapdlyen: Used to enable the input tap delay.

The clock delay can be imposed by either using tap delay or a DLL that generates multiple phases of the clock. The maximum number of phases (tap delay) supported is 180, even though the typical number of phases (tap delay) is four or eight.

Figure 26-4 shows the usage of the DLL for the TX CMD/DAT delay and the RX clock tap delay.

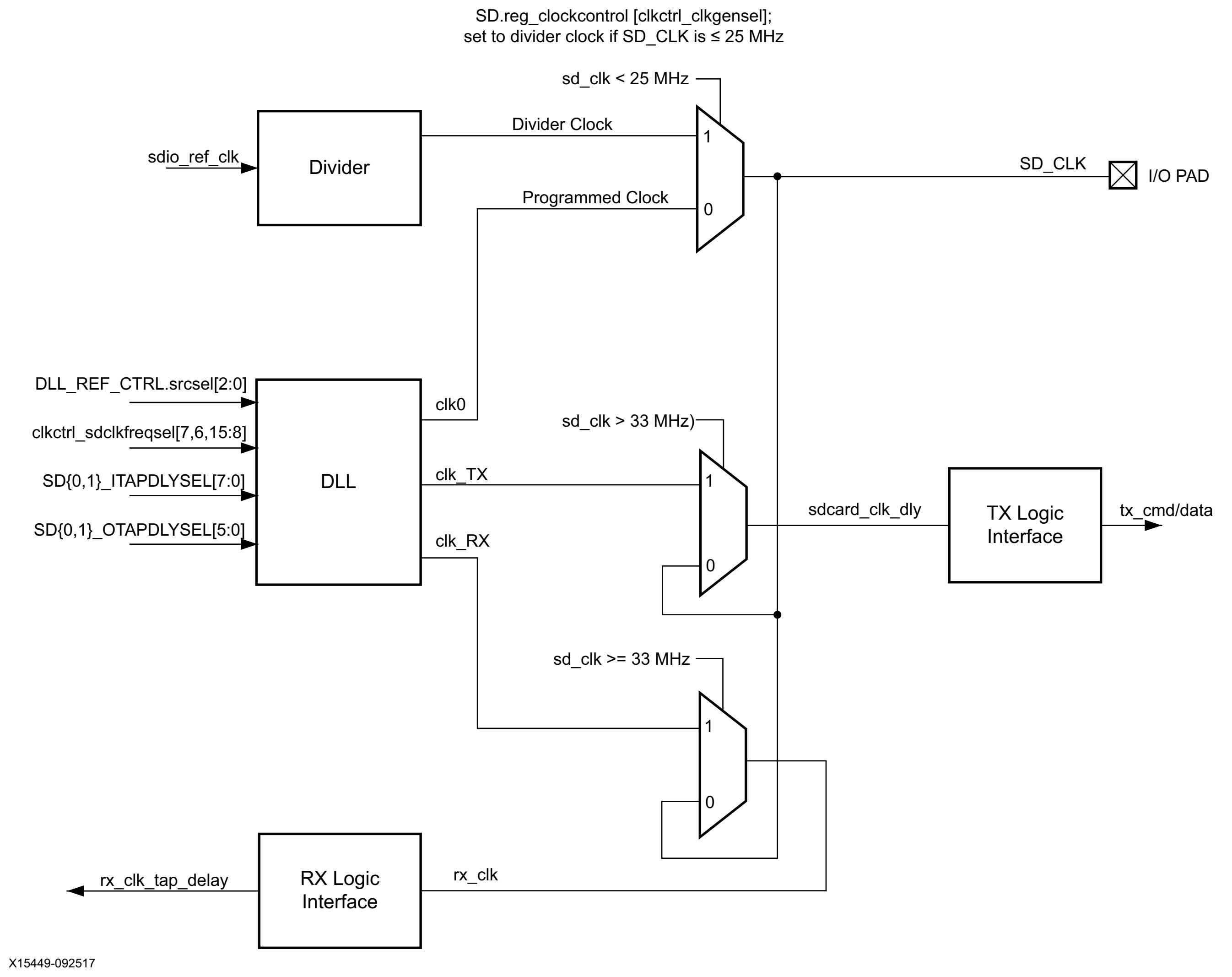


Figure 26-4: **SD Clock Control Using Existing DLL**

The block diagram (Figure 26-4) is further described in this section.

- The DLL has the DLL_REF_CTRL.srcsel[2:0], clkctrl_sdclkfreqsel[7, 6,15:8], SD{0,1}_ITAPDLYSEL[7:0], and SD{0,1}_OTAPDLYSEL[5:0] register controls.
- DLL clock can be generated based on the DLL_REF_CTRL.srcsel[2:0] and DLL divisor value. This DLL divisor internally selects based on the clkctrl_sdclkfreqsel[7, 6,15:8] value from the reg_clockcontrol register of SDI{0,1}. For example, the DLL is derived from the IOPLL (Table 26-4).

Table 26-4: **DLL Mode Supported Clocks**

SD{0,1}_BASECLK (MHz)	DLL (MHz) IOPLL or RPLL	clkctrl_sdclkfreqsel	Actual DLL Divider Value	SD Output Frequency (MHz)
200	1500	0	7.5	200
	1500	1	15	100
	1500	2	30	50
	1500	3	45	33.33
100	1500	0	15	100
	1500	1	30	50
50	1500	0	30	50

Note: For more information, see Xilinx Answer 71825.

- A CLK_TX is generated based on the SD{0,1}_OTAPDLYSEL[5:0] register and multiplexed with the feedback clock for SD_CLK value > 33 MHz as the select line. If SD_CLK is greater than 25 MHz, the CLK_TX is selected as the transmission clock for the TX CMD/DATA delay. The CLK_TX is shifted by up to a fully divided clock cycle from CLK_0 in increments of 1/DIV. The TX clock delay can be calculated (Equation 26-1) using the OTAP delay, the DLL divisor, and a clock period.

  Delay = (OTAPDLYSEL[5:0] x Clock Period) x (1/DLL_DIV) *Equation 26-1*

  For example, calculate the TX clock delay with the following values.

  OTAPDLYSEL[5:0] = 4

  Clock period = 5 ns (1/DLL_REF_CLK)

  DLL_DIV = 7.5

  TX clock delay = (4 x 5) x (1/7.5) = 2.667 ns

- A CLK_RX is generated based on the SD{0,1}_ITAPDLYSEL[7:0] register and multiplexed with the feedback clock for SD_CLK value ≥ 33 MHz as the select line. If SD_CLK is ≥ 33 MHz, the CLK_RX is selected as the receive clock for the RX clock delay unit.The CLK_RX is shifted by up to a fully divided clock cycle from CLK_0 in increments of 1/(4 x DIV). The RX clock delay is calculated (Equation 26-2) using the ITAP delay, DLL divisor, and a clock period.

  Delay = (ITAPDLYSEL[7:0] x Clock Period) x (1/4 x DLL_DIV) *Equation 26-2*

For example, calculate the RX clock delay with the following values.

ITAPDLYSEL[7:0] = 4

Clock period = 5 ns (1/DLL_REF_CLK)

DLL_DIV = 7.5

RX clock delay = (4 x 5) x (1/(4 x 7.5)) = 0.667 ns

SD Tap Delay Settings

The [SD0_ITAPCHGWIN] and [SD1_ITAPCHGWIN] bits in the IOU_SLCR.SD_ITAPDLY register are used to gate the output of the tap delay lines to avoid glitches being propagated into the controller. This signal should be asserted a few clocks before the corectrl_itapdlysel changes and stay asserted for a few clocks afterwards.

For SDIO{0, 1}, use the IOU_SLCR register set and the tap delay values in Table 26-5 through Table 26-10.

Table 26-5: **SD104/eMMC200 Mode**

Register Bit	SDIO0 Bank 0	SDIO{0, 1} Bank 1	SDIO{0, 1} Bank 2	Description
SD_ITAPDLY[SDx_ITAPDLYENA]	`1'b1`	`1'b1`	`1'b1`	SLCR I tap delay enable (RX).
SD_ITAPDLY[SDx_ITAPDLYSEL]	N/A[(1)]	N/A[(1)]	N/A[(1)]	RX tap delay values.
SD_OTAPDLY[SDx_OTAPDLYSEL]	`6'b000011`	`6'b000011`	`6'b000010`	TX tap delay values.

Notes:

1. The (N/A) value is calculated from auto-tuning by the SDIO controller. You can program this value with trial and error.

Table 26-6: **SD50 Mode**

Register Bit	SDIO0 Bank 0	SDIO{0, 1} Bank 1	SDIO{0, 1} Bank 2	Description
SD_ITAPDLY[SDx_ITAPDLYENA]	1'b1	1'b1	1'b1	SLCR I tap delay enable (RX).
SD_ITAPDLY[SDx_ITAPDLYSEL]	8'b00010100	8'b00010100	8'b00010100	RX tap delay values.
SD_OTAPDLY[SDx_OTAPDLYSEL]	6'b000011	6'b000011	6'b000011	TX tap delay values.

Table 26-7: **SD DDR Mode**

Register Bit	SDIO0 Bank 0	SDIO{0, 1} Bank 1	SDIO{0, 1} Bank 2	Description
SD_ITAPDLY[SDx_ITAPDLYENA]	`1'b1`	`1'b1`	`1'b1`	SLCR I tap delay enable (RX).
SD_ITAPDLY[SDx_ITAPDLYSEL]	`8'b00111101`	`8'b00111101`	`8'b00111101`	RX tap delay values.
SD_OTAPDLY[SDx_OTAPDLYSEL]	`6'b000100`	`6'b000100`	`6'b000100`	TX tap delay values.

Table 26-8: **eMMC DDR Mode**

Register Bit	SDIO0 Bank 0	SDIO{0, 1} Bank 1	SDIO{0, 1} Bank 2	Description
SD_ITAPDLY[SDx_ITAPDLYENA]	`1'b1`	`1'b1`	`1'b1`	SLCR I tap delay enable (RX).
SD_ITAPDLY[SDx_ITAPDLYSEL]	`8'b00010010`	`8'b00010010`	`8'b00010010`	RX tap delay values.
SD_OTAPDLY[SDx_OTAPDLYSEL]	`6'b000110`	`6'b000110`	`6'b000110`	TX tap delay values.

Table 26-9: **SD HSD Mode**

Register Bit	SDIO0 Bank 0	SDIO{0, 1} Bank 1	SDIO{0, 1} Bank 2	Description
SD_ITAPDLY[SDx_ITAPDLYENA]	`1'b1`	`1'b1`	`1'b1`	SLCR I tap delay enable (RX).
SD_ITAPDLY[SDx_ITAPDLYSEL]	`8'b00010101`	`8'b00010101`	`8'b00010101`	RX tap delay values.
SD_OTAPDLY[SDx_OTAPDLYSEL]	`6'b000101`	`6'b000101`	`6'b000101`	TX tap delay values.

Table 26-10: **eMMC HSD Mode**

Register Bit	SDIO0 Bank 0	SDIO{0, 1} Bank 1	SDIO{0, 1} Bank 2	Description
SD_ITAPDLY[SDx_ITAPDLYENA]	`1'b1`	`1'b1`	`1'b1`	SLCR I tap delay enable (RX).
SD_ITAPDLY[SDx_ITAPDLYSEL]	`8'b00010101`	`8'b00010101`	`8'b00010101`	RX tap delay values.
SD_OTAPDLY[SDx_OTAPDLYSEL]	`6'b000110`	`6'b000110`	`6'b000110`	TX tap delay values.

SD Interface Voltage Translation

SD initialization and booting of the PS is completed at 3.3V in high-speed mode. After initialization, the SD interface (command, data, and clock) operates at 1.8V to support the ultra high-speed (UHS) SD cards. The highest speed modes are only supported when the bank voltage is 1.8V.

Dynamic switching from 3.3V to 1.8V is required on the host and the SD interface. Figure 26-5 shows the SD voltage switching sequence diagram.

The voltage translation function is implemented by an external voltage level translator as shown in Figure 26-6.

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 (Table 26-1). The SEL pin is automatically driven by the controller if configured in SD3.0.

IMPORTANT: *Voltage level shifters are only used for SD cards. SD interface voltage translation is only applicable to SD UHS cards.*

Note: The SDx_DIR0 and SDx_DIR1 direction signals might not be required for all devices. For additional information, refer to the Versal ACAP PCB Design User Guide (UG863) and verify with device vendor

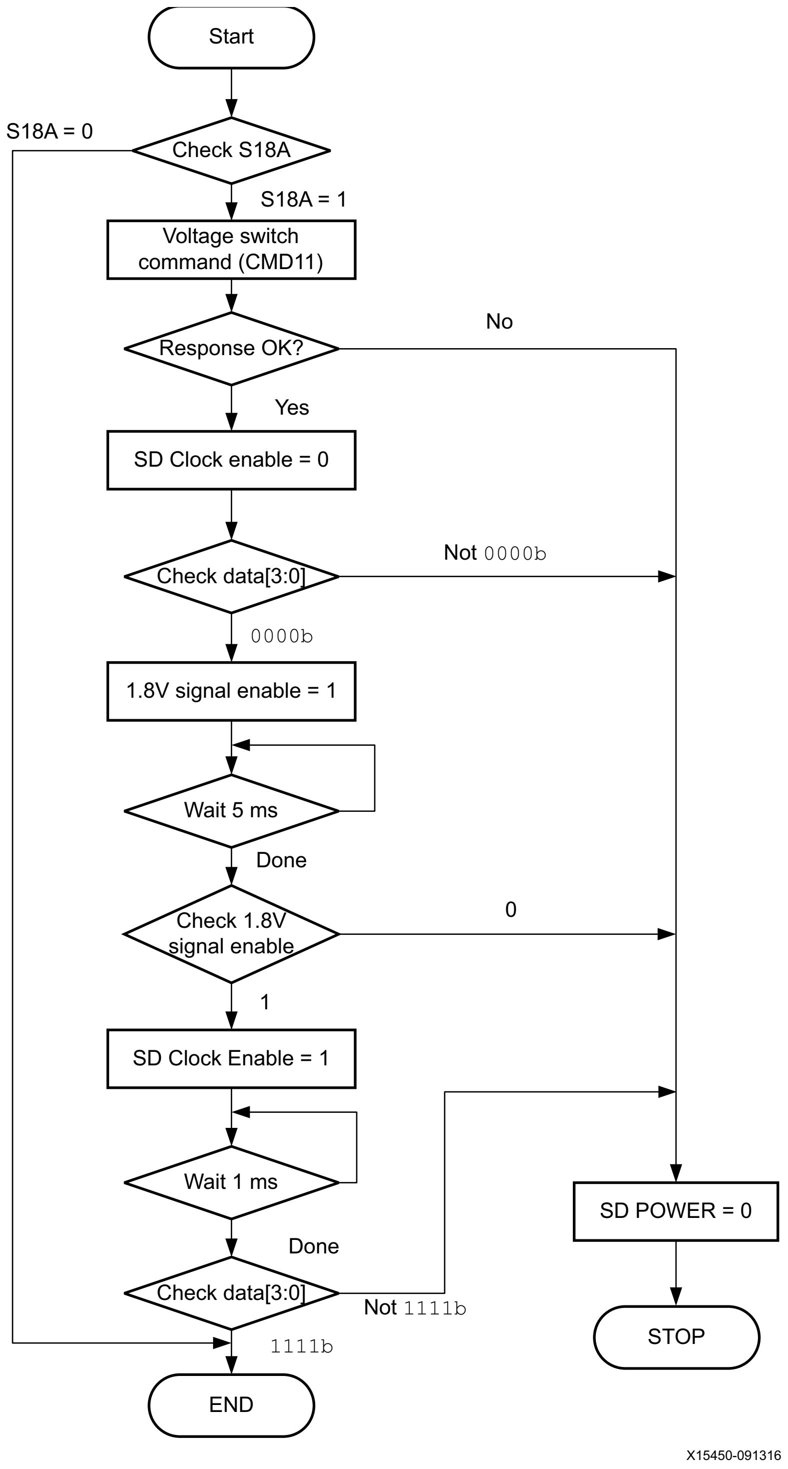


Figure 26-5: **SD Voltage Switching Sequence Diagram**

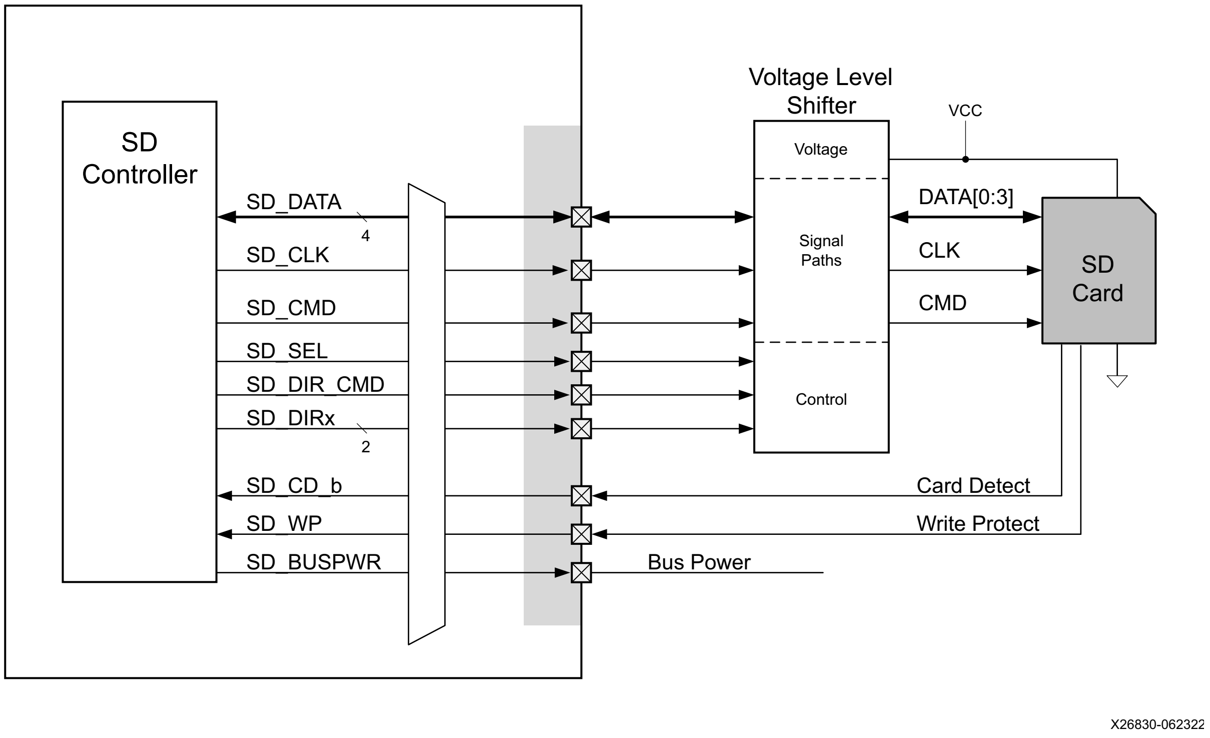


Figure 26-6: **External Voltage Translator**

I/O Signals

MIO-EMIO Signals

The SDIO media interface signals are independently routed to the MIO pins or to a set of EMIO interface signals, see Table 26-11. MIO is discussed in Chapter 28, Multiplexed I/O.

Table 26-11: **SDIO Interface Signals**

SDIO Interface	MIO Pins		EMIO Signals		Default Controller Input Value
	Number	I/O	Name	I/O	
SDIO 0 clock	22,38,64	I/O	emio_sdio0_fb_clk_in	I	0
			emio_sdio0_clkout	O	~
SDIO 0 command	21,40,66	I/O	emio_sdio0_cmdin	I	0
			emio_sdio0_cmdout	O	~
			emio_sdio0_cmdena	O	~
SDIO 0 data 0	13,41,67	I/O	emio_sdio0_datain[0]	I	0
			emio_sdio0_dataout[0]	O	~
			emio_sdio0_dataena[0]	O	~

Table 26-11: **SDIO Interface Signals *(Cont'd)***

SDIO Interface	MIO Pins		EMIO Signals		Default Controller Input Value
	Number	I/O	Name	I/O	
SDIO 0 data 1	14,42,68	I/O	emio_sdio0_datain[1]	I	0
			emio_sdio0_dataout[1]	O	~
			emio_sdio0_dataena[1]	O	~
SDIO 0 data 2	15,43,69	I/O	emio_sdio0_datain[2]	I	0
			emio_sdio0_dataout[2]	O	~
			emio_sdio0_dataena[2]	O	~
SDIO 0 data 3	16,44,70	I/O	emio_sdio0_datain[3]	I	0
			emio_sdio0_dataout[3]	O	~
			emio_sdio0_dataena[3]	O	~
SDIO 0 data 4	17,45,71	I/O	emio_sdio0_datain[4]	I	0
			emio_sdio0_dataout[4]	O	~
			emio_sdio0_dataena[4]	O	~
SDIO 0 data 5	18,46,72	I/O	emio_sdio0_datain[5]	I	0
			emio_sdio0_dataout[5]	O	~
			emio_sdio0_dataena[5]	O	~
SDIO 0 data 6	19,47,73	I/O	emio_sdio0_datain[6]	I	0
			emio_sdio0_dataout[6]	O	~
			emio_sdio0_dataena[6]	O	~
SDIO 0 data 7	20,48,74	I/O	emio_sdio0_datain[7]	I	0
			emio_sdio0_dataout[7]	O	~
			emio_sdio0_dataena[7]	O	~
SDIO 0 card detect	24,39,65	I	emio_sdio0_cd_n	I	
SDIO 0 write protect	25,50,76	I	emio_sdio0_wp	I	
SDIO 0 power control	23,49,75	O	emio_sdio0_buspower	O	~
SDIO 0 LED control	~	~	emio_sdio0_ledcontrol	O	~
SDIO 0 bus voltage	~	~	emio_sdio0_bus_volt[2:0]	O	~
SDIO 1 clock	51	I/O	emio_sdio1_fb_clk_in	I	0
			emio_sdio1_clkout	O	~
SDIO 1 command	50	I/O	emio_sdio1_cmdin	I	0
			emio_sdio1_cmdout	O	~
			emio_sdio1_cmdena	O	~

Table 26-11: **SDIO Interface Signals *(Cont'd)***

SDIO Interface	MIO Pins		EMIO Signals		Default Controller Input Value
	Number	I/O	Name	I/O	
SDIO 1 data 0	46, 71	I/O	emio_sdio1_datain[0]	I	0
			emio_sdio1_dataout[0]	O	~
			emio_sdio1_dataena[0]	O	~
SDIO 1 data 1	47,72	I/O	emio_sdio1_datain[1]	I	0
			emio_sdio1_dataout[1]	O	~
			emio_sdio1_dataena[1]	O	~
SDIO 1 data 2	48,73	I/O	emio_sdio1_datain[2]	I	0
			emio_sdio1_dataout[2]	O	~
			emio_sdio1_dataena[2]	O	~
SDIO 1 data 3	49,74	I/O	emio_sdio1_datain[3]	I	0
			emio_sdio1_dataout[3]	O	~
			emio_sdio1_dataena[3]	O	~
SDIO 1 data 4	39	I/O	emio_sdio1_datain[4]	I	0
			emio_sdio1_dataout[4]	O	~
			emio_sdio1_dataena[4]	O	~
SDIO 1 data 5	40	I/O	emio_sdio1_datain[5]	I	0
			emio_sdio1_dataout[5]	O	~
			emio_sdio1_dataena[5]	O	~
SDIO 1 data 6	41	I/O	emio_sdio1_datain[6]	I	0
			emio_sdio1_dataout[6]	O	~
			emio_sdio1_dataena[6]	O	~
SDIO 1 data 7	42	I/O	emio_sdio1_datain[7]	I	0
			emio_sdio1_dataout[7]	O	~
			emio_sdio1_dataena[7]	O	~
SDIO 1 card detect	45,77	I	emio_sdio1_cd_n	I	
SDIO 1 write protect	44,69	I	emio_sdio1_wp	I	
SDIO 1 power control	43,70	O	emio_sdio1_buspower	O	~
SDIO 1 LED control	~	~	emio_sdio1_ledcontrol	O	~
SDIO 1 bus voltage	~	~	emio_sdio1_bus_volt[2:0]	O	~

Note: SDIO 0/1 power control & emio_sdio0/1_buspower signal is a an output signal. This can be used enable or disable to an external power device to control VDD supply which powers the SDIO level shifter. For more information, refer to *UltraScale Architecture PCB Design User Guide* (UG583) [Ref 15] for more information for PCB recommendations.

Register Overview

The SD controller registers are listed in Table 26-12.

Table 26-12: **SD Controller Register Overview**

Register Type	Register Name	Address	Width	Type	Description
Command Generation	reg_sdmasysaddrlo	0x0000	16	Read/Write	This register contains the lower 16-bit of the physical system memory address used for DMA transfers or the second argument for the auto CMD23.
	reg_sdmasysaddrhi	0x0002	16	Read/Write	This register contains the higher 16-bits of the physical system memory address used for DMA transfers or the second argument for the auto CMD23.
	reg_blocksize	0x0004	16	Read/Write	This register is used to configure the number of bytes in a data block.
	reg_blockcount	0x0006	16	Read/Write	This register is used to configure the number of data blocks.
	reg_argument1lo	0x0008	16	Read/Write	This register contains the lower bits of the SD command argument.
	reg_argument1hi	0x000A	16	Read/Write	This register contains the higher bits of the SD command argument.
	reg_transfermode	0x000C	16	Read/Write	This register is used to control the operations of data transfers.
	reg_command	0x000E	16	Read/Write	This register is used to program the command for the host controller.
Response	reg_response{0:7}	0x0010 to 0x001E	16	Read only	Store responses from SD cards.
Buffer	reg_dataport	0x0020	32	Read/Write	This register is used to access the internal buffer.

Table 26-12: **SD Controller Register Overview *(Cont'd)***

Register Type	Register Name	Address	Width	Type	Description
Host controller 1	reg_presentstate	`0x0024`	32	Read only	The host driver can get the status of the host controller from this 32-bit read-only register.
	reg_hostcontrol1	`0x0028`	8	Read/Write	This register is used to program DMA modes, LED control, data transfer width, high-speed enable, card detect test level, and signal selection.
	reg_powercontrol	`0x0029`	8	Read/Write	This register is used to program the SD bus power and voltage level.
	reg_blockgapcontrol	`0x002A`	8	Mixed	This register is used to program the block gap request, read wait control, and interrupt at block gap.
	reg_wakeupcontrol	`0x002B`	8	Read/Write	This register is used to program the wakeup functionality.
	reg_clockcontrol	`0x002C`	16	Mixed	This register is used to program the clock frequency select, generator select, clock enable, and internal clock state fields.
	reg_timeoutcontrol	`0x002E`	8	Read/Write	The register sets the data timeout counter value.
	reg_softwarereset	`0x002F`	8	Clear on Write CLRONWR	This register is used to program the software reset for data, command, and for all.
Interrupt controls	reg_normalintrsts	`0x0030`	16	Mixed	This register gives the status of all the interrupts.
	reg_errorintrsts	`0x0032`	16	Write to Clear (WTC)	This register gives the status of the error interrupts.
	reg_normalintrstsena	`0x0034`	16	Mixed	This register is used to enable the normal interrupt status register fields.
	reg_errorintrstsena	`0x0036`	16	Read/Write	This register is used to enable the error interrupt status register fields.
	reg_normalintrsigena	`0x0038`	16	Mixed	This register is used to enable the normal interrupt signal register.
	reg_errorintrsigena	`0x003A`	16	Mixed	This register is used to enable the error interrupt signal register.
	reg_autocmderrsts	`0x003C`	16	Read only	This register is used to indicate CMD12 response error of auto CMD12 and CMD23 response error of auto CMD 23.

Table 26-12: **SD Controller Register Overview *(Cont'd)***

Register Type	Register Name	Address	Width	Type	Description
Host controller 2	reg_hostcontrol2	`0x003E`	16	Mixed	This register is used to program UHS select mode, UHS select mode, execute tuning, sampling clock select, asynchronous interrupt enable, and preset value enable.
Capabilities	reg_capabilities	`0x0040`	64	Read only	This register provides the host driver with information specific to the host controller implementation.
	reg_maxcurrentcap	`0x0048`	64	Read only	This register indicates maximum current capability for each voltage.
Force event	reg_ForceEventfor AUTOCMDErrorStatus	`0x0050`	16	Write only	This register is not physically implemented, rather it is an address where auto CMD error status register can be written.
	reg_forceeventforerrintsts	`0x0052`	16	Mixed	This register is not physically implemented, rather it is an address where the error interrupt status register can be written.
ADMA controller	reg_admaerrsts	`0x0054`	8	Read only	When the ADMA error interrupt occurs, this register holds the ADMA state in the ADMA error states field and the ADMA system address holds the address around the error descriptor.
	reg_admasysaddr0	`0x0058`	16	Read/Write	This register contains the physical address used for ADMA data transfer.
	reg_admasysaddr1	`0x005A`	16	Read/Write	This register contains the physical address used for ADMA data transfer.
	reg_admasysaddr2	`0x005C`	16	Read/Write	This register contains the physical address used for ADMA data transfer.
	reg_admasysaddr3	`0x005E`	16	Read/Write	This register contains the physical address used for ADMA data transfer.

Table 26-12: **SD Controller Register Overview *(Cont'd)***

Register Type	Register Name	Address	Width	Type	Description
Preset values	reg_presetvalue0	`0x0060`	16	Read only	This register is used to read the SD_CLK frequency select value, clock generator select value, and driver strength select value.
	reg_presetvalue1	`0x0062`	16	Read only	This register is used to read the SD_CLK frequency select value, clock generator select value, and driver strength select value for default speed.
	reg_presetvalue2	`0x0064`	16	Read only	This register is used to read the SD_CLK frequency select value, clock generator select value, and driver strength select value for high speed.
	reg_presetvalue3	`0x0066`	16	Read only	This register is used to read the SD_CLK frequency select value, clock generator select value, and driver strength select value for SDR12.
	reg_presetvalue4	`0x0068`	16	Read only	This register is used to read the SD_CLK frequency select value, clock generator select value, and driver strength select value for SDR25.
	reg_presetvalue5	`0x006A`	16	Read only	This register is used to read the SD_CLK frequency select value, clock generator select value, and driver strength select value for SDR50.
	reg_presetvalue6	`0x006C`	16	Read only	This register is used to read the SD_CLK frequency select value, clock generator select value, and driver strength select value for SDR104.
	reg_presetvalue7	`0x006E`	16	Read only	This register is used to read the SD_CLK frequency select value, clock generator select value, and driver strength select value for DDR50.
General control and status	reg_boottimeoutcnt	`0x0070`	32	Read/Write	This is used to program the boot timeout value counter.
	reg_slotintrsts	`0x00FC`	16	Read only	This register is used to read the interrupt signal for each slot.
	reg_hostcontrollerver	`0x00FE`	16	Read only	This register is used to read the vendor version number and the specification version number.

SD Command Generation

The registers to generate SD commands are listed in Table 26-13.

Table 26-13: **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
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

Table 26-13 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.

Programming Examples

This section shows examples of various data transfer protocols for a host controller.

DMA Data Transaction

DMA Read Transfer

On receiving the response end bit from the card for the write command (data flowing from host to card), the SD host controller acts as the master and requests the system/host bus. After receiving the grant, the host controller starts reading a block of data from the system memory and fills the first FIFO. Whenever a block of data is ready, the transmitter starts sending the data in the SD bus.

While transmitting the data in the SD bus, the host controller requests the bus to fill the second block in the second FIFO. Ping-pong FIFOs are used to increase the throughput. Similarly, the host controller reads a block of data from the system memory whenever a FIFO is empty. This continues until all the blocks are read from the system memory. The transfer complete interrupt is only set after transferring all the blocks of data to the card.

DMA Write Transfer

The block of data received from the card (data flowing from card to host) is stored in the first half of the FIFO. Whenever a block of data is ready, the SD host controller acts as the master and requests the system/host bus. After receiving the grant, the host controller starts writing a block of data into the system memory from the first FIFO. While transmitting the data into system memory, the host controller receives the second block of data and stores it in the second FIFO. Similarly, the host controller writes a block of data into the system memory whenever data is ready. This continues until all the blocks are transferred to the system memory. The transfer complete interrupt is only set after transferring all the blocks of data to the system memory.

TIP: *The host controller receives a block of data from the card only when it has room to store a block of data in a FIFO. When both the FIFOs are full, the host controller stops the data coming from the card through a read wait mechanism (if the card supports a read wait mechanism) or by stopping the clock.*

SD Configuration

Table 26-14: **SD Configuration**

Task	SD{0, 1} Registers	Register Field	Register Offset	Bits	Value
Reset the controller.	reg_softwarereset	swreset_for_all	`0x2f`	0	`1b'1`
Wait until device has been reset.	reg_softwarereset	swreset_for_all	`0x2f`	0	Read operation
Read and save controller capabilities(1).	reg_capabilities	ALL	`0x40`	63:0	Read operation
Select voltage 3.3V and enable bus power.	reg_powercontrol	pwrctrl_sdbusvoltage | pwrctrl_sdbuspower	`0x29`	3:1,0	`4b'1111`
Change the clock frequency to 400 KHz (see Table 26-15).					
Select 32-bit ADMA2 mode.	reg_hostcontrol1	hostctrl1_dmaselect	`0x28`	4:3	`2b'10`
Enable all interrupt status except card interrupt initially.	reg_normalintrstsena	ALL	`0x34`	15:0	`0xFEFF`
Enable error interrupts.	reg_errorintrstsena	ALL	`0x36`	12:0	Write `3FF h`
Disable all interrupt signals.	reg_normalintrsigena	ALL	`0x38`	15:0	Write `0000h`
Disable all error signals.	reg_errorintrsigena	ALL	`0x40`	12:0	Write `000h`

Table 26-14: **SD Configuration *(Cont'd)***

Task	SD{0, 1} Registers	Register Field	Register Offset	Bits	Value
Transfer mode register: default value. DMA enabled, block count enabled, data direction card to host (read).	reg_transfermode	xfermode_dmaenable | xfermode_blkcntena | xfermode_dataxferdir	`0x0C`	4, 1, and 0	Write 1 to all bits
Set block size to 512 by default.	reg_blocksize	xfer_blocksize	`0x04`	11:0	Write `200h`

Notes:

1. the re-tuning interval in the SDIO capabilities register is determined by the IOU_SLCR.SD_CONFIG_REG3 [SD0_RETUNETMR] and [SD1_RETUNETMR] fields. The default value for this register is set to `0x8`, which enables auto refresh at 128s. The software driver must ensure programming this register to the appropriate value based on your specific application requirements.

Table 26-15: **SD Clock Frequency Change**

Task	SD{0, 1} Registers	Register Field	Register Offset	Bits	Value
Disable clock.	reg_clockcontrol	clkctrl_intclkena and clkctrl_sdclkena	`0x2C`	2 and 0	Write 0
Set clock divisor	reg_clockcontrol	clkctrl_sdclkfreqsel and clkctrl_intclkena	`0x2C`	15:7 and 0	Write 1 to bit 0 and divisor value to bits 15:7.
Wait for 1 or 2 microseconds					
Wait until internal clock stabilized.	reg_clockcontrol	sdhcclkgen_intclkstable_dsync	`0x2C`	1	Read until set
Enable SD clock.	reg_clockcontrol	clkctrl_sdclkena	`0x2C`	2	Write 1

Notes:

1. The internal clock enable signal (clkctrl_intclkena) needs to cleared for at least one SD_CLK cycle whenever clock frequency is changed.

SD Card Initialize

Table 26-16: **SD Card Initialize**

Task	SD{0, 1} Registers	Register Field	Register Offset	Bits	Value
Check the present state register to make sure the card is inserted and detected by the host controller.	reg_presentstate	sdhccarddet_inserted_dsync	`0x24`	16	Read
74 clock delay after card is powered up, before the first command.					
Send CMD0 to card with no response expected (see Table 26-17).					
Send CMD8 to card with response (`0x1AA` for supply voltage 2.7–3.6V and AA pattern) expected (see Table 26-17).					
Read response 0 for CMD8 and decide card version.	reg_response0	command_response	`0x10`	15:0	Read
Send ACMD41 while card is still busy with power up.					
Send CMD55 (see Table 26-17).					
Send ACMD41 `0x40300000`: host high capacity support and 3.3V window (see Table 26-17).					
Read response 0 for response with card capacity.	reg_response0	command_response	`0x10`	15:0	Read
Perform above three steps until OCR ready bit (31st) is set.					
Send CMD2 for card ID (see Table 26-17).					
Read card specific data in response.	reg_response0	command_response	`0x10`	15:0	Read
Read card specific data in response.	reg_response1	command_response	`0x12`	15:0	Read
Read card specific data in response.	reg_response2	command_response	`0x14`	15:0	Read
Read card specific data in response.	reg_response3	command_response	`0x16`	15:0	Read
Send CMD3 and read response until relative card address (upper 16 bits of response) received.	reg_response0	command_response	`0x10`	15:0	Read
Send CMD9 with relative address received.					
Read card specific data in response.	reg_response0	command_response	`0x10`	15:0	Read
Read card specific data in response.	reg_response1	command_response	`0x12`	15:0	Read
Read card specific data in response.	reg_response2	command_response	`0x14`	15:0	Read
Read card specific data in response.	reg_response3	command_response	`0x16`	15:0	Read

SD CMD Transfer

Table 26-17: **SD CMD Transfer**

Task	SD{0, 1} Registers	Register Field	Register Offset	Bits	Value
Check the command inhibit to make sure no other command transfer is in progress.	reg_presentstate	presentstate_inhibitcmd	`0x24`	0	Read
Write block count register.	reg_blockcount	xfer_blockcount	`0x06`	15:0	Block count
Write timeout.	reg_timeoutcontrol	timeout_ctrvalue	`0x2E`	3:0	`0x0E`
Write argument register.	reg_argument1lo	command_argument1	`0x08`	15:0	Argument
Clear all normal status interrupts.	reg_normalintrsts	ALL	`0x30`	15:0	`0xFFFF`
Clear all error status interrupts.	reg_errorintrsts	All	`0x36`	12:0	`0xF3FF`
Frame the command.					
Check for data inhibit in case of command using DAT lines.	reg_presentstate	presentstate_inhibitdat	`0x24`	1	Read
Write command.	reg_command	ALL	`0x0E`	13:0	Command
Polling for response while command complete bit set.	reg_normalintrsts	normalintrsts_cmdcomplete	`0x30`	0	Read until set
Clear error bits if error interrupt bit sets from previous operation.	reg_errorintrsts	ALL	`0x32`	15:0	`0xF3FF`
Clear command complete bit.	reg_normalintrsts	normalintrsts_cmdcomplete	`0x30`	0	`1b'1`

SD Set Block Size

Table 26-18: **SD Set Block Size**

Task	SD{0, 1} Registers	Register Field	Register Offset	Bits	Value
Check the present state register to make sure the bus is free.	reg_presentstate	presentstate_inhibitcmd, presentstate_inhibitdat, sdhcdmactrl_wrxferactive, sdhcdmactrl_rdxferactive	`0x24`	9:8 and 1:0	Read
Send block write command (CMD16) (see Table 26-17).					
Read card-specific data in response.	reg_response0	command_response	`0x10`	15:0	Read
Set the block size.	reg_blocksize	xfer_blocksize	`0x04`	11:0	`0x200`

Setup ADMA2 Descriptor Table

Table 26-19: **Setup ADMA2 Descriptor Table**

Task	SD{0, 1} Registers	Register Field	Register Offset	Bits	Value
Read block size and calculate total descriptor lines.	reg_blocksize	xfer_blocksize	`0x04`	11:0	Read operation
Prepare descriptor table.					
Program descriptor table address to ADMA2 address register.	reg_admasysaddr0	adma_sysaddress0	`0x58`	15:0	Descriptor table address

SD Read Polled

Table 26-20: **SD Read Polled**

Task	SD{0, 1} Registers	Register Field	Register Offset	Bits	Value
Check the present state register to make sure the card is present.	reg_presentstate	sdhccarddet_inserted_dsync	`0x24`	16	Read
If not already set, set block size to 512 (see Table 26-18).					
Set up ADMA2: slave select setup ADMA2 descriptor table.					
Set up mode register with Auto CMD12 enable, block count enable, data transfer direction, DMA enable, and multi/single block select.	reg_transfermode	ALL	`0x0C`	5:0	`0x37`
Send block read command (CMD18) (see Table 26-17).					

Table 26-20: **SD Read Polled *(Cont'd)***

Task	SD{0, 1} Registers	Register Field	Register Offset	Bits	Value
Check for transfer completed.	reg_normalintrsts	reg_errorintrsts	`0x30`	15	Read operation
Clear the interrupts (if any).	reg_normalintrsts	ALL	`0x30`	15:0	`0xF3FF`
Check transfer complete and clear if transfer is completed.	reg_normalintrsts	normalintrsts_xfercomplete	`0x30`	1	`1b'1`
Read response 0.	reg_response0	command_response	`0x10`	15:0	Read

SD Write Polled

Table 26-21: **SD Write Polled**

Task	SD{0, 1} Registers	Register Field	Register Offset	Bits	Value
Check the present state register to make sure the card is present.	reg_presentstate	sdhccarddet_inserted_dsync	`0x24`	16	Read
If not already set, set block size to 512 (see Table 26-18).					
Set up ADMA2 (see Table 26-19).					
Set up mode register with auto CMD12 enable, block count enable, data transfer direction, DMA enable, and multi/single block select.	reg_transfermode	ALL	`0x0C`	5:0	`0x37`
Send block read command (CMD18) (see Table 26-17)					
Check for transfer completed.	reg_normalintrsts	reg_errorintrsts	`0x30`	15	Read operation
Clear the interrupts (if any).	reg_normalintrsts	ALL	`0x30`	15:0	`0xF3FF`
Check transfer complete and clear if transfer is completed.	reg_normalintrsts	normalintrsts_xfercomplete	`0x30`	1	`1b'1`

SD Select Card

Table 26-22: **SD Select Card**

Task	SD{0, 1} Registers	Register Field	Register Offset	Bits	Value
Send card select command CMD7 (see Table 26-17).					
Read response 0.	reg_response0	command_response	`0x10`	15:0	Read
Set default block size (512) (see Table 26-18).					

eMMC Card Initialize

Table 26-23: **eMMC Card Initialize**

Task	SD{0, 1} Registers	Register Field	Register Offset	Bits	Value
Check the present state register to make sure the card is present.	reg_presentstate	sdhccarddet_inserted_dsync	0x24	16	Read
74 clock delay after card is powered up, before the first command.					
Send CMD0 to card with no response expected (see Table 26-17).					
Send CMD1 while card is still busy with power up (perform the following two steps).					
Send command (CMD1) with options to host high-capacity support and high-voltage window (see Table 26-17).					
Read response 0.	reg_response0	command_response	0x10	15:0	Read
Send CMD2 for CARD ID (see Table 26-17).					
Send CMD3. Save relative card address in response 0.					
Read card specific data in response.	reg_response0	command_response	0x10	15:0	Read
Read card specific data in response.	reg_response1	command_response	0x12	15:0	Read
Read card specific data in response.	reg_response2	command_response	0x14	15:0	Read
Read card specific data in response.	reg_response3	command_response	0x16	15:0	Read
Send CMD9 with relative card address saved in CMD3 response.					
Read card specific data in response.	reg_response0	command_response	0x10	15:0	Read
Read card specific data in response.	reg_response1	command_response	0x12	15:0	Read
Read card specific data in response.	reg_response2	command_response	0x14	15:0	Read
Read card specific data in response.	reg_response3	command_response	0x16	15:0	Read

SD Get Bus Width

Table 26-24: **SD Get Bus Width**

Task	SD{0, 1} Registers	Register Field	Register Offset	Bits	Value
Send block write command (CMD55) (see Table 26-17).					
Set block size to desired value.	reg_blocksize	xfer_blocksize	0x04	11:0	Block size value
Set up ADMA2 descriptor table.					
Set transfer mode with data direction and DMA enable.	reg_transfermode	xfermode_dmaenable | xfermode_dataxferdir	0x0C	4 and 1	0x11
Data cache invalidate range.					
Send ACMD51 with desired block count.					
Check for transfer completed.	reg_normalintrsts	reg_errorintrsts	0x30	15	Read operation

Table 26-24: **SD Get Bus Width *(Cont'd)***

Task	SD{0, 1} Registers	Register Field	Register Offset	Bits	Value
Clear the interrupts (if any).	reg_normalintrsts	ALL	`0x30`	15:0	`0xF3FF`
Check transfer complete and clear if transfer is completed.	reg_normalintrsts	normalintrsts_xfercomplete	`0x30`	1	`1b'1`
Read response 0.	reg_response0	command_response	`0x10`	15:0	Read

SD Change Bus Width

Table 26-25: **SD Change Bus Width**

Task	SD{0, 1} Registers	Register Field	Register Offset	Bits	Value
For SD Card					
Send block write command (CMD55) (see Table 26-17) if not defined as eMMC card.					
Send CMD6 command with 4-bit data bus width selected.					
Set bus width in host control register.	reg_hostcontrol1	hostctrl1_datawidth	`0x28`	1	`1b'1`
Read card specific data in response.	reg_response0	command_response	`0x10`	15:0	Read
For eMMC					
Send ACMD6 command with 4-bit data bus width selected.					
Wait for 2 ms.					
Set bus width in host control register.	reg_hostcontrol1	hostctrl1_datawidth	`0x28`	1	`1b'1`
Read card specific data in response.	reg_response0	command_response	`0x10`	15:0	Read

SD Get Bus Speed

Table 26-26: **SD Get Bus Speed**

Task	SD{0, 1} Registers	Register Field	Register Offset	Bits	Value
Set block size to desired value.	reg_blocksize	xfer_blocksize	`0x04`	11:0	Block size value
Set up ADMA2 descriptor table (see Table 26-19).					
Set transfer mode with data direction and DMA enable.	reg_transfermode	xfermode_dmaenable | xfermode_dataxferdir	`0x0C`	4 and 1	`0x11`
Data cache invalidate range.					
Send CMD6.					

Table 26-26: **SD Get Bus Speed *(Cont'd)***

Task	SD{0, 1} Registers	Register Field	Register Offset	Bits	Value
Check for transfer completed.	reg_normalintrsts	reg_errorintrsts	0x30	15	Read operation
Clear the interrupts (if any).	reg_normalintrsts	ALL	0x30	15:0	0xF3FF
Check transfer complete and clear if transfer is completed.	reg_normalintrsts	normalintrsts_xfercomplete	0x30	1	1b'1
Read response 0.	reg_response0	command_response	0x10	15:0	Read

SD Change Bus Speed

Table 26-27: **SD Change Bus Speed**

Task	SD{0, 1} and IOU_SLCR Registers	Register Field	Register Offset	Bits	Value
For SD card					
Set block size to desired value.	reg_blocksize	xfer_blocksize	0x04	11:0	Block size value
Set up ADMA descriptor table (see Table 26-19).					
Set transfer mode with data direction and DMA enable.	reg_transfermode	xfermode_dmaenable | xfermode_dataxferdir	0x0C	4 and 1	0x11
Data cache (dcache) invalidate range.					
Send CMD6.					
Check for transfer completed.	reg_normalintrsts	All	0x30	15	Read operation
Clear the error interrupts (if any).	reg_errorintrsts	All	0x30	15:0	0xF3FF
Check transfer complete and clear if transfer is completed.	reg_normalintrsts	normalintrsts_xfercomplete	0x30	1	1b'1
Change clock frequency to 50 MHz (see Table 26-15).					
Enable high speed.	reg_hostcontrol1	hostctrl1_highspeedena	0x28	2	1b'1
Read response 0.	reg_response0	command_response	0x10	15:0	Read
For eMMC card					
Send CMD6 with eMMC high-speed argument and wait 2 ms.					
Change clock frequency to 52 MHz.					
Enable high speed.	reg_hostcontrol1	hostctrl1_highspeedena	0x28	2	1b'1

Table 26-27: **SD Change Bus Speed *(Cont'd)***

Task	SD{0, 1} and IOU_SLCR Registers	Register Field	Register Offset	Bits	Value
Program sequence to execute the tuning for high-speed cards					
Set block size value.	reg_blocksize	xfer_blocksize	`0x04`	11:0	Block size value
Set the mode to data transfer.	reg_transfermode	xfermode_dataxferdir	`0x0C`	4	`0x10`
Set the execute tuning mode.	reg_hostcontrol2	hostctrl2_executetuning	`0x3E`	6	`0x40`
SD card: send CMD19 or eMMC card: send CMD 21.					
Check if the reg_hostcontrol2 for tuning mask bit is not set.					
Repeat the process until the tuning mask bit is not set.					
Check the sampling clock select is set.	reg_hostcontrol2	hostctrl2_ samplingclkselect	`0x3E`	8	`0x80`
Set the desired clock frequency using Table 26-15, if sampling clock selection is set.					
Program sequence for DLL tap delay for SD0 and SD1 controllers for high-speed cards.					
Set the DLL reset value.	SD_DLL_CTRL	SD{0,1}_DLL_RST	`0x0`	2 and 18	`0x4`
Set the ITAPCHGWIN to gate the glitches in the line.	SD_ITAPDLY	SD{0,1}_ITAPCHGWIN	`0x0`	9 and 25	`0x200`
Set the ITAPDLYENA.	SD_ITAPDLY	SD{0,1}_ITAPDLYENA	`0x0`	8 and 24	`0x100`
Unset the ITAPCHGWIN.	SD_ITAPDLY	SD{0,1}_ITAPCHGWIN	`0x0`	9 and 25	Clear bit 9
Set the taps for the desired clock value.	SD_OTAPDLYSEL	SD{0,1}_OTAPDLYSEL	`0x0`	5:0 and 21:16	Desired value based on the clock.
Release DLL reset.	SD_DLL_CTRL	SD{0,1}_DLL_RST	`0x0`	2 and 18	Clear bit 2.
Set the taps for the desired clock value.	SD_OTAPDLYSEL	SD1_OTAPDLYSEL	`0x0`	21:16	Desired value based on the clock.
Unset the DLL reset value.	SD_DLL_CTRL	SD1_DLL_RST	`0x0`	18	Unset the bit 2.

SD Send Pullup Command

Table 26-28: **SD Send Pullup Command**

Task	Register	Register Field	Register Offset	Bits	Value
Send CMD55.					
Send ACMD42.					

Get eMMC EXT CSD

Table 26-29: **Get eMMC EXT CSD**

Task	Register	Register Field	Register Offset	Bits	Value
Set block size to desired value.	reg_blocksize	xfer_blocksize	`0x04`	11:0	Block size value
Set up ADMA2 descriptor table (see Table 26-19).					
Data cache invalidate range.					
Set transfer mode with data direction and DMA enable.	reg_transfermode	xfermode_dmaenable | xfermode_dataxferdir	`0x0C`	4 and 1	`0x11`

Resetting the DLL

Table 26-30: **Resetting the DLL**

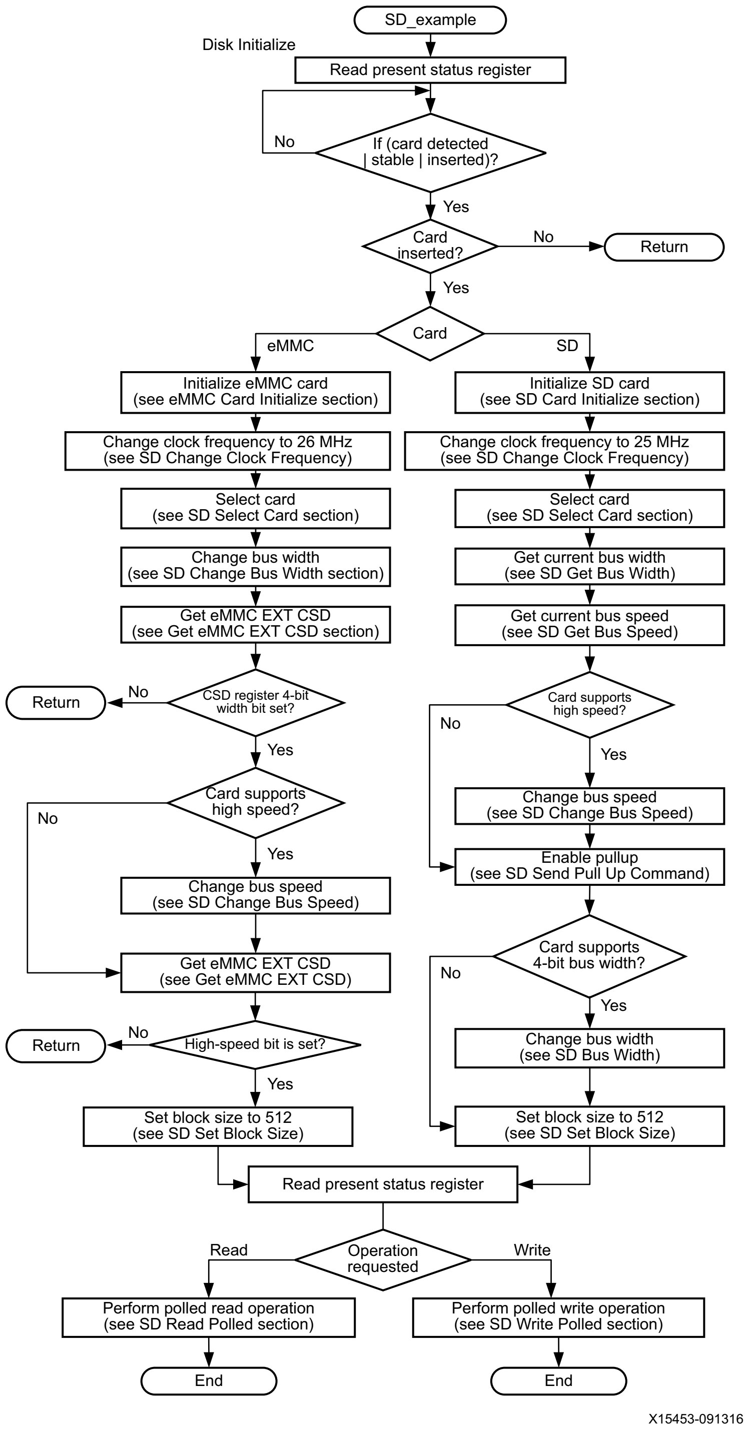
Task	Register	Register Field	Register Offset	Bits	Value
Disable clock.	reg_clockcontrol	clkctrl_sdclkena	0x2C	2	Clear bit 2
Set the DLL reset value.	SD_DLL_CTRL	SD{0,1}_DLL_RST	0x358	2 and 18	Set bit 2 and 18
Wait for 1 or 2 microseconds.					
Release DLL from reset.	SD_DLL_CTRL	SD{0,1}_DLL_RST	0x358	2 and 18	Clear bit 2 and 18
Wait until internal clock to stabilize.	reg_clockcontrol	sdhcclkgen_intclkstable_dsync	0x2C	1	Read
Enable SD clock.	reg_clockcontrol	clkctrl_sdclkena	0x2C	2	Set bit 2

Manual Tuning

Table 26-31: **Manual Tuning**

Task	Register	Register Field	Register Offset	Bits	Value
Disable clock.	reg_clockcontrol	clkctrl_sdclkena	`0x2C`	2	Clear bit 2
Gate the glitches in the line.	SD_ITAPDLY	SD{0,1}_ITAPCHGWIN	`0x314`	9 and 25	Set bit 9 and 25
Enable Rx tap delay clock.	SD_ITAPDLY	SD{0,1}_ITAPDLYENA	`0x314`	8 and 24	Set bit 8 and 24
Select number of taps for DLL.	SD_ITAPDLY	SD{0,1}_ITAPDLYSEL	`0x314`	7:0 and 23:16	Desired value based on the clock
Unset the ITAPCHGWIN.	SD_ITAPDLY	SD{0,1}_ITAPCHGWIN	`0x314`	9 and 25	Clear bit 9 and 25
Set the taps for the desired clock value.	SD_OTAPDLYSEL	SD{0,1}_OTAPDLYSEL	`0x318`	5:0 and 21:16	Desired value based on the clock.
Wait for few cycles for the taps to get synchronized.					
Set the DLL reset value.	SD_DLL_CTRL	SD{0,1}_DLL_RST	`0x358`	2 and 18	Set bit 2 and 18
Wait for few cycles					
Release DLL from reset.	SD_DLL_CTRL	SD{0,1}_DLL_RST	`0x358`	2 and 18	Clear bit 2 and 18
Wait until internal clock to stabilize.	reg_clockcontrol	sdhcclkgen_intclkstable_dsync	`0x2C`	1	Read
Enable SD clock.	reg_clockcontrol	clkctrl_sdclkena	`0x2C`	2	Set bit 2

SD/eMMC Example Flow Diagram



SD_example
Disk Initialize
Read present status register
No
If (card detected | stable | inserted)?
Yes
Card inserted?
No
Return
Yes
Card
eMMC
SD
Initialize eMMC card (see eMMC Card Initialize section)
Initialize SD card (see SD Card Initialize section)
Change clock frequency to 26 MHz (see SD Change Clock Frequency)
Change clock frequency to 25 MHz (see SD Change Clock Frequency)
Select card (see SD Select Card section)
Select card (see SD Select Card section)
Change bus width (see SD Change Bus Width section)
Get current bus width (see SD Get Bus Width)
Get eMMC EXT CSD (see Get eMMC EXT CSD section)
Get current bus speed (see SD Get Bus Speed)
Return
No
CSD register 4-bit width bit set?
Yes
Card supports high speed?
No
Yes
No
Card supports high speed?
Yes
Change bus speed (see SD Change Bus Speed)
Change bus speed (see SD Change Bus Speed)
Enable pullup (see SD Send Pull Up Command)
Get eMMC EXT CSD (see Get eMMC EXT CSD)
Card supports 4-bit bus width?
No
Yes
Return
No
High-speed bit is set?
Yes
Change bus width (see SD Bus Width)
Set block size to 512 (see SD Set Block Size)
Set block size to 512 (see SD Set Block Size)
Read present status register
Read
Operation requested
Write
Perform polled read operation (see SD Read Polled section)
Perform polled write operation (see SD Write Polled section)
End
End
X15453-091316

Figure 26-7: **SD/eMMC Example Flow Diagram**

Sequence Flowchart for Using DMA

Figure 26-8 is a flowchart for using DMA.

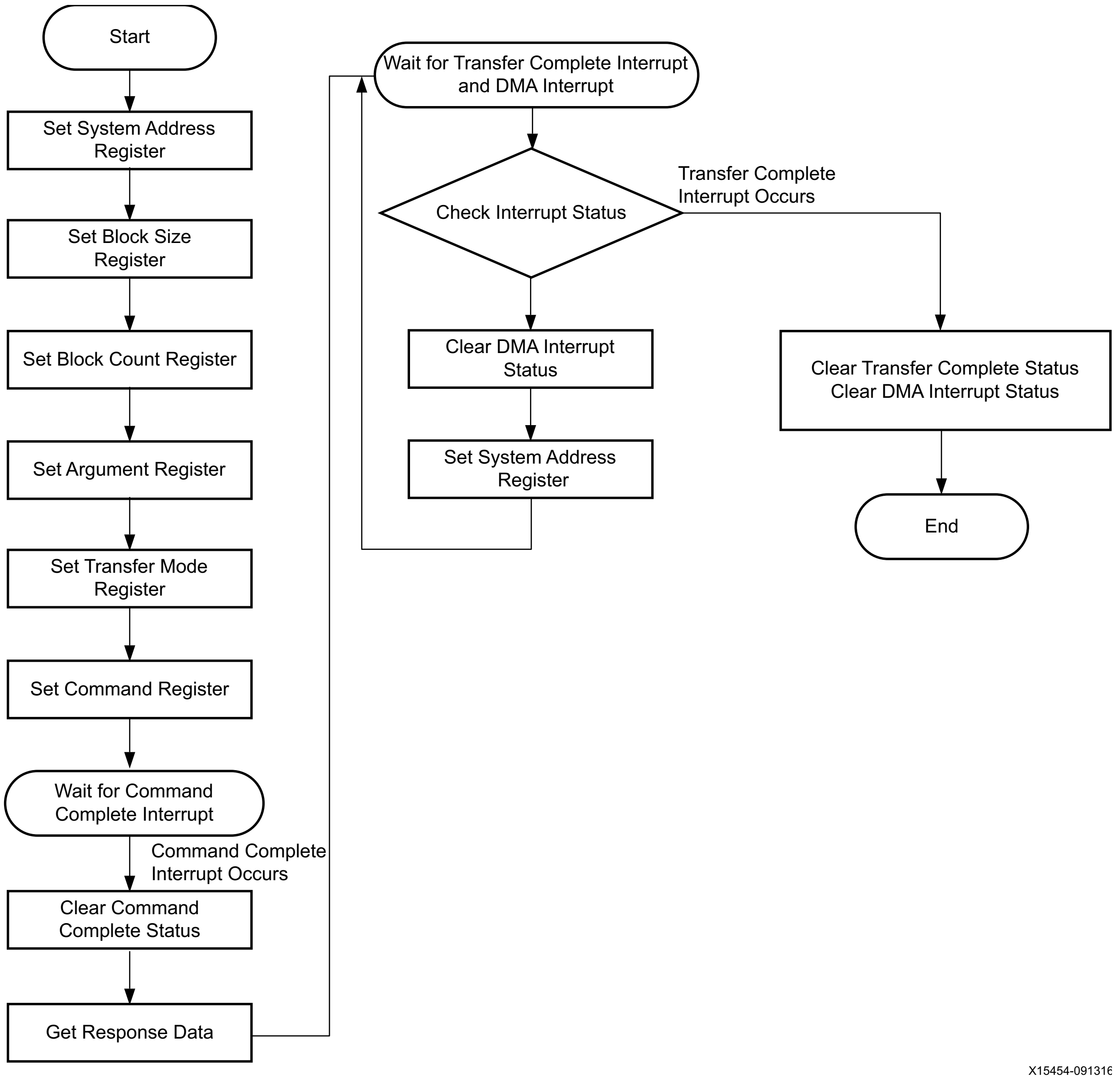

Start
Set System Address Register
Set Block Size Register
Set Block Count Register
Set Argument Register
Set Transfer Mode Register
Set Command Register
Wait for Command Complete Interrupt
Command Complete Interrupt Occurs
Clear Command Complete Status
Get Response Data
Wait for Transfer Complete Interrupt and DMA Interrupt
Check Interrupt Status
Transfer Complete Interrupt Occurs
Clear DMA Interrupt Status
Set System Address Register
Clear Transfer Complete Status
Clear DMA Interrupt Status
End
X15454-091316

Figure 26-8: **DMA Data Transaction Flowchart**

Non-DMA Data Transaction

Steps for a Non-DMA Data Transaction

1. Set the value corresponding to the executed data byte length of one block to the block size register.
2. Set the value corresponding to the executed data block count to the block count register.
3. Set the value corresponding to the issued command to the argument register.
4. Set the value to multi or single block select and block count enable.
5. Set the value corresponding to the issued command to the data transfer direction, auto CMD12 enable, and DMA enable.
6. Set the value corresponding to the issued command in the command register. When writing the upper byte of the command register, the SD command is issued.
7. Wait for the command complete interrupt.
8. Write a 1 to the command complete in the normal interrupt status register to clear this bit.
9. Read the response register and get the necessary information in accordance with the issued command.
10. When this sequence is used for writing to a card, go to step 11. When reading from a card, go to step 15.
11. Wait for a buffer write ready interrupt.

Non-DMA Write Transfer

On receiving the buffer write ready interrupt, the Arm processor acts as a master and starts transferring the data through the buffer data port register (FIFO_1). The transmitter starts sending the data in the SD bus when a block of data is ready in FIFO_1. While transmitting the data in the SD bus, the buffer write ready interrupt is sent to the Arm processor for the second block of data. The Arm processor acts as a master and starts sending the second block of data through the buffer data port register to FIFO_2. The buffer write ready interrupt is only asserted when a FIFO is empty to receive a block of data.

12. Write a 1 to the buffer write ready in the normal interrupt status register to clear this bit.
13. Write the block data (in accordance with the number of bytes specified in step 1) to the buffer data port register.
14. Repeat until all the blocks are sent and then go to step 19.

Non-DMA Read Transfer

The buffer read ready interrupt is asserted whenever a block of data is ready in one of the FIFOs. On receiving the buffer read ready interrupt, the Arm processor acts as a master and starts reading the data through the buffer data port register (FIFO_1). The receiver starts reading the data from the SD bus only when a FIFO is empty and can receive a block of data. When both the FIFOs are full, the host controller stops the data coming from the card through a read wait mechanism (if the card supports a read wait mechanism) or by stopping the clock.

Wait for Buffer Read Ready Interrupt

15. Write a 1 to the buffer read ready in the normal interrupt status register to clear this bit.
16. Read the block data (in accordance with the number of bytes specified in step 1) from the buffer data port register.
17. Repeat until all blocks are received and then go to step 18.
18. If this sequence is for a single or multiple block transfer, go to step 19. For an infinite block transfer, go to step 21.
19. Wait for a transfer complete interrupt.
20. Write a 1 to the transfer complete in the normal interrupt status register to clear this bit.
21. Perform the sequence for abort transaction.

Note: Step 1 and step 2 can be executed at same time. Step 4 and step 5 can also be executed at same time.

IMPORTANT: *During the process of auto tuning, the software driver must ignore the reg_presentstate (SDIO) register's [sdhcdmactrl_piobufrdena] field before sending CMD19 or CMD21.*

Sequence Flowchart for Not Using DMA

Figure 26-9 is a flowchart for not using DMA.

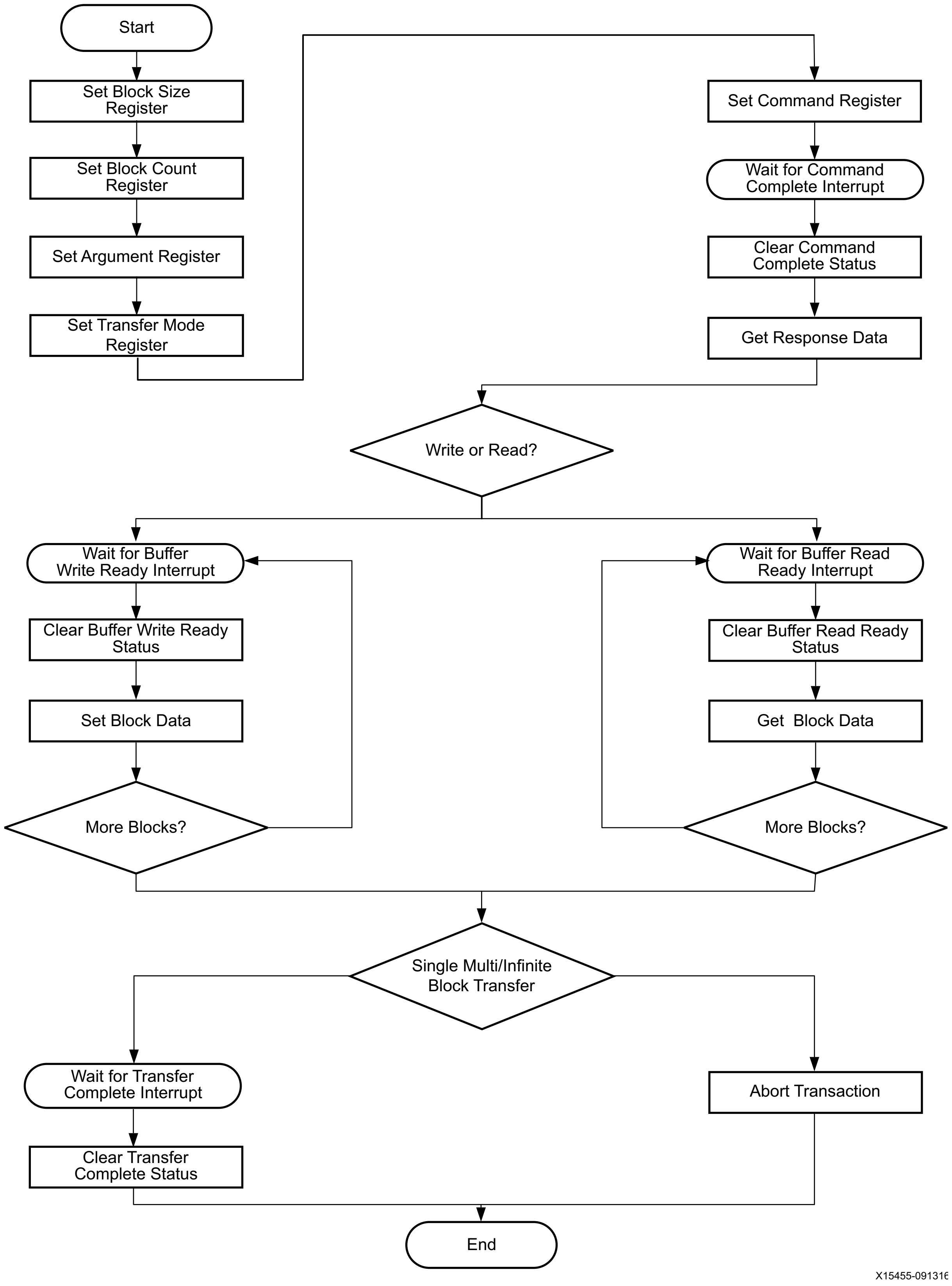

Start
Set Block Size Register
Set Block Count Register
Set Argument Register
Set Transfer Mode Register
Set Command Register
Wait for Command Complete Interrupt
Clear Command Complete Status
Get Response Data
Write or Read?
Wait for Buffer Write Ready Interrupt
Clear Buffer Write Ready Status
Set Block Data
More Blocks?
Wait for Buffer Read Ready Interrupt
Clear Buffer Read Ready Status
Get Block Data
More Blocks?
Single Multi/Infinite Block Transfer
Wait for Transfer Complete Interrupt
Clear Transfer Complete Status
Abort Transaction
End
X15455-091316

Figure 26-9: **Non-DMA Data Transaction Flowchart**

Chapter 27

General Purpose I/O

Introduction

The general purpose I/O (GPIO) is a collection of input/output signals available to software applications. The GPIO consists of the MIO with 78 pins, and the extended multiplexed I/O interface (EMIO) with 288 signals that are divided into 96 inputs from the programmable logic (PL) and 192 outputs to the PL. The GPIO is organized into six banks of registers that group related interface signals.

Each GPIO channel is independently and dynamically programmed as input, output, or interrupt sensing. Software applications can read all GPIO values within a bank using a single load instruction, or write data to one or more GPIOs using a single store instruction. The GPIO control and status registers are memory mapped beginning at base address `0xFF0A_0000` and are protected by the XPPU.

Features

Key features of the GPIO peripheral are summarized as follows:

- 78 GPIO interfaces to the device pins.
    - Routed through the MIO multiplexer.
    - Programmable I/O drive strength, slew rate, and 3-state control.
- 96 GPIO interfaces to the PL (four allocated by software to reset PL logic).
    - Routed through the EMIO interface.
    - Data inputs.
    - Data outputs.
    - Output enables.
- I/O interface is organized into six banks (3 MIO and 3 EMIO).
- Interface control registers are grouped by bank {0:5}.
- Input values are read using the six DATA_RO_x registers.
- Two types of data ports for writing:

    - Full bank write using the DATA_x registers.
    - Split bank maskable write using the MASK_DATA_x_{LWS, MWS} register pairs.
- The function of each GPIO can be dynamically programmed on an individual or group basis.
- Enable, bit or bank data write, output enable and direction controls.
- Programmable interrupts on individual GPIO basis.
    - Status read of raw and masked interrupt
    - Selectable sensitivity: Level-sensitive (High or Low) or edge-sensitive (positive, negative, or both).

SDK and Hardware Design

The Xilinx software and hardware design tools assign functionality to GPIO channels. For example, the tools define four GPIO [92:95] channels routed to the EMIO for resets to user-defined logic in the PL.

Functional Description

Figure 27-1 shows the block diagram of the GPIO. The GPIO is divided into six banks.

- Bank 0: 26-bit bank controlling MIO pins [0:25].
- Bank 1: 26-bit bank controlling MIO pins [26:51].
- Bank 2: 26-bit bank controlling MIO pins [52:77].

  Note: Bank 0 to bank 2 are 26 bits each.
- Bank 3: 32-bit bank controlling EMIO signal sets [0:31].
- Bank 4: 32-bit bank controlling EMIO signal sets [32:63].
- Bank 5: 32-bit bank controlling EMIO signal sets [64:95].

Note: Up to four outputs for GPIO[92:95] can act as reset signals to user-defined logic in the PL. The number of GPIO EMIO signals depends on the number of PL fabric resets selected in the Vivado PS configuration wizard (PCW). For example, if one reset is selected, GPIO[95] is assigned as a reset signal. If two are selected, then GPIO[95:94] are assigned.

The GPIO is controlled by software through a series of memory-mapped registers. The control for each bank is the same, although there are minor differences between the MIO and EMIO banks due to their differing functionality.

The EMIO interface is not connected to MIO pins. 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.

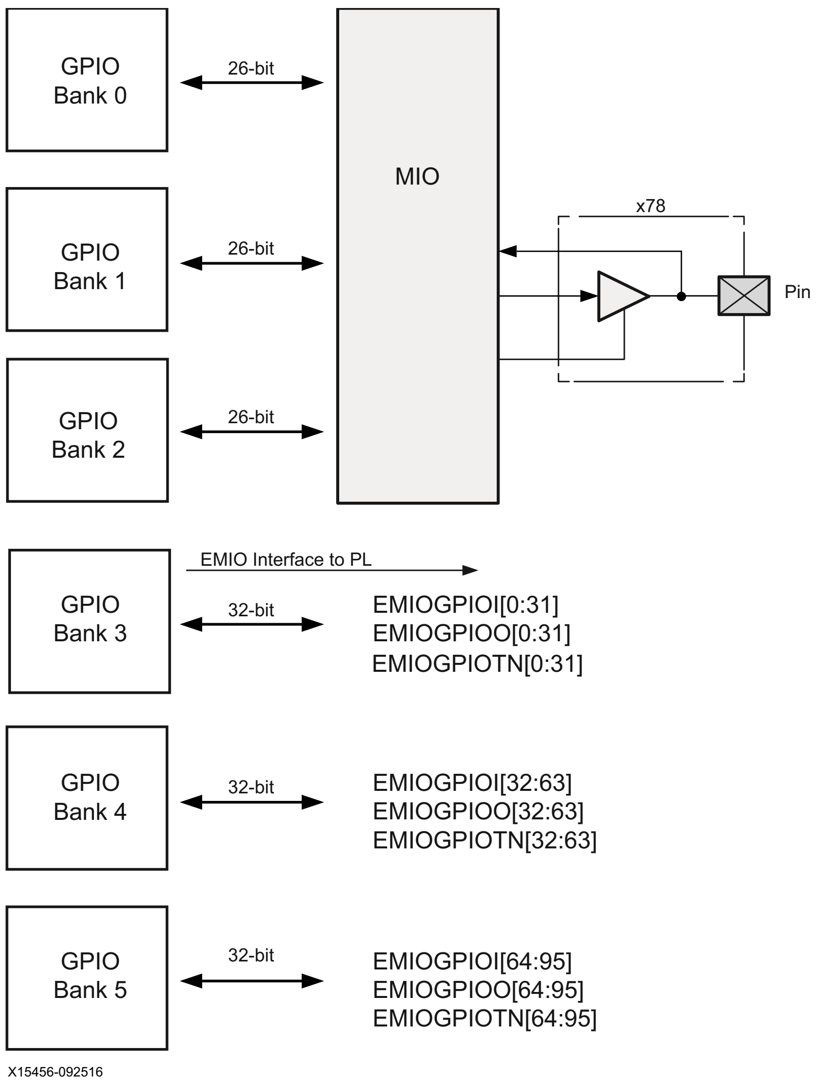


Figure 27-1: **GPIO Block Diagram**

MIO Pin Configuration

Banks 0 to 2 of the GPIO peripheral are routed to device pins through the MIO. All MIO pin configuration registers in Table 28-2 use the IOU_SLCR register set.

Basic GPIO Functions

The main function of the GPIO peripheral is to provide direct access to device pins from withing the PS and to allow software designers to drive pins through application software in a highly flexible manner. This is useful for a wide variety of system applications to control external hardware components and implement general purpose interfaces between devices. The GPIO peripheral also provides an interrupt capability through an event detection unit. The basic functions of a bank of GPIO pins are illustrated in Figure 27-2.

GPIO Channel Architecture

The GPIO channels for the MIO and EMIO are very similar. For the MIO, the input, output, and 3-state signals connect to the I/O buffer. For the EMIO, all three signals (two output and one input) are available to the PL fabric.

Device Pin Channels

GPIO banks 0, 1, and 2 connect to device pins through the MIO (see Figure 27-2).

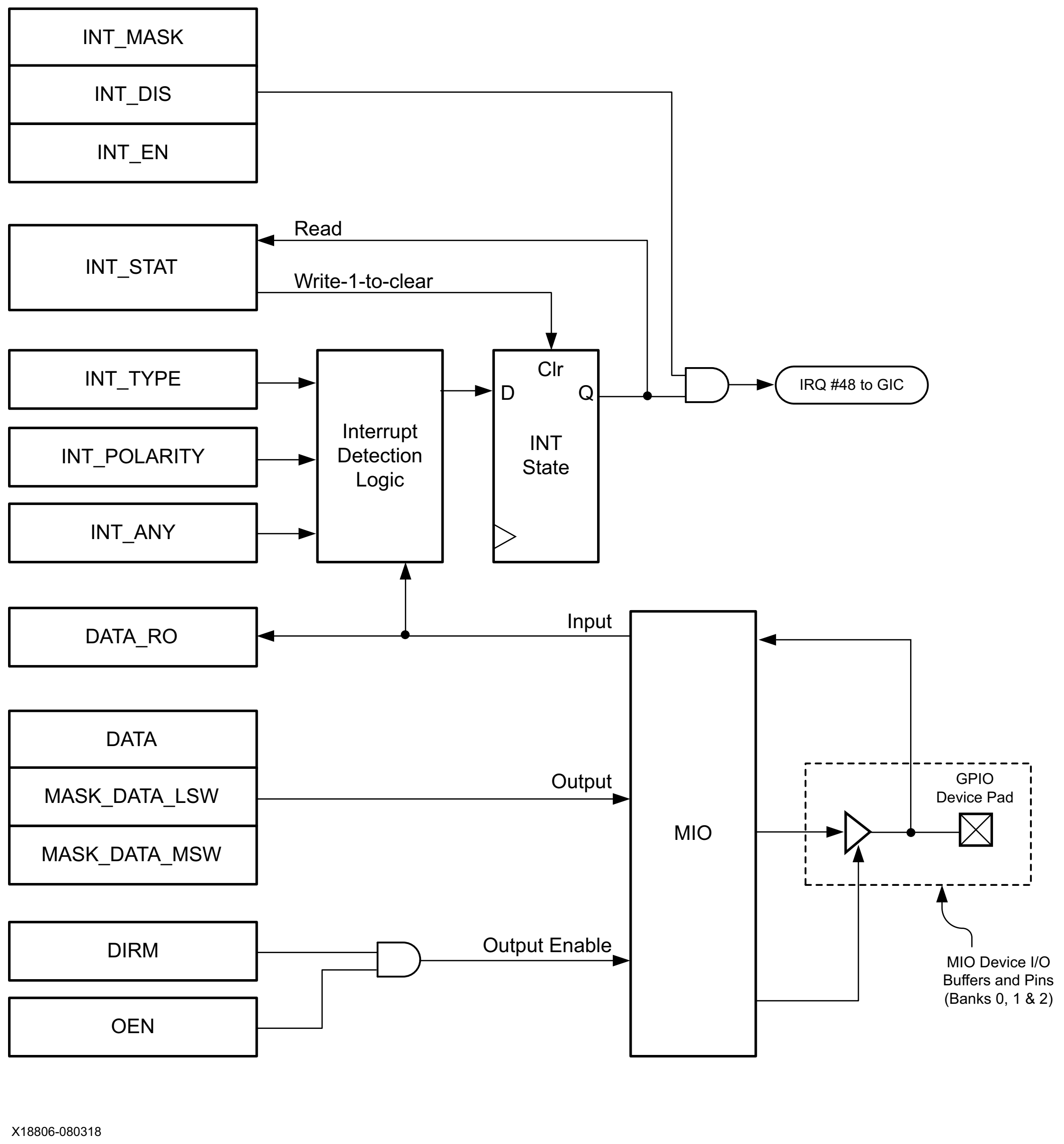


Figure 27-2: **GPIO Channel**

Software configures the GPIO as either an output or input. The DATA_RO register always returns the state of the GPIO pin regardless of whether the GPIO is set to input (OE signal false) or output (OE signal true). To generate an output waveform, software repeatedly writes to one or more GPIOs (usually using the MASK_DATA register).

Applications might need to switch more than one GPIO at the same time (less a small amount of inherent skew time between two I/O buffers). In this case, all of the GPIOs that need to be switched simultaneously must be from the same 16-bit half-bank (i.e., either the most-significant 16 bits or the least-significant 16 bits) of GPIOs to enable the MASK_DATA register to write to them in one store instruction.

GPIO bank control (for banks 0, 1, and 2) is summarized as follows:

- **DATA_RO:** This register enables software to observe the value on the device pin. If the GPIO signal is configured as an output, then this would normally reflect the value being driven on the output. Writes to this register are ignored.

  Note: If the MIO is not configured to enable this pin as a GPIO pin, then DATA_RO is unpredictable because software cannot observe values on non-GPIO pins through the GPIO registers.

- **DATA:** 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}; it 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 upper16 bits of the bank.
- **DIRM:** Direction Mode. This controls whether the I/O pin is acting as an input or an output. Since the input logic is always enabled, this effectively enables/disables the output driver. When DIRM[x]==0, the output driver is disabled.
- **OEN:** Output Enable. When the I/O is configured as an output, this controls whether the output is enabled or not. When the output is disabled, the pin is 3-stated. When OEN[x]==0, the output driver is disabled.

  Note: If MIO TRI_ENABLE is set to 1, enabling 3-state and disabling the driver, then OEN is ignored and the output is 3-stated.

MIO Signals

This section describes the operation of GPIO bank 0, bank 1, and bank 2.

Input Mode

In input mode, the pin values are passed through to the corresponding register location after meta-stability protection (GPIO inputs are considered asynchronous). The pin values are available through two different paths. There is a dedicated path as well as a path through the register. 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.

Output Mode

In output mode, the pin values are driven by the corresponding register location. The direction control must be set to a `1` and the output enable set to a `1` for the output to be passed through the pad driven by MIO. The direction control and output enable can be controlled separately. The direction control can be used to disable input values being passed to the registers or APB write bus values being applied to input registers. The output enable can be used separately to control whether an output value is passed or not passed to the pin. The actual I/O pad direction control (gpio.OEN_{0:5}) is the logical combination of both these signals, the output enable value is masked when the direction mode is set to input.

In the output mode, when the output enable is active, the output pin value can be read from either the APB read only location or the APB read/write location. When the output enable is inactive, the pin is an input pin, and the value is available at the read only location. The register value that drives the inactive 3-state buffer can be read from the read/write location. The GPIO output and OEN signals are asserted and de-asserted asynchronously to all PL clocks.

EMIO Signals

This section describes the operation of GPIO bank 3, bank 4, and bank 5. The register interface for the EMIO banks is the same as for the MIO banks. The EMIO interface differences are explained in this section.

The inputs come from the PL and are unrelated to the output values or the OEN (gpio.OEN_{0:5}) register. They can be read from the DATA_0_R0 register when their bit in the DIRM register is set to 0 (making it an input). The outputs are not 3-state capable and are not affected by the OEN register. The output value is programmed using the DATA, MASK_DATA_LSW, and MASK_DATA_MSW registers. DIRM must be set to 1 (making it an output). For more details on these registers, refer to Table 27-2.

Note: Similar to MIO, there is no PL clock associated with the GPIO EMIO signals and should be considered asynchronous to PL logic.

Interrupt Function

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, then 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 in order 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 27-1: **GPIO Interrupt Trigger Settings**

Type	gpio.INT_TYPE_0	gpio.INT_POLARITY_0	gpio.INT_ANY_0
Rising edge-sensitive	1	1	0
Falling edge-sensitive	1	0	0
Both rising- and falling edge-sensitive	1	X	1
Level sensitive, asserted High	0	1	X
Level sensitive, asserted Low	0	0	X

System Interfaces

The controller clocks and resets are described in this section. All of the interrupts generated in the GPIO controller are routed to IRQ 48. The GPIO I/O signals can be routed to either the MIO or EMIO.

Clock

The controller operates on the rising edge of the LPD_LSBUS_CLK clock that is used for both the APB interface as well as all other GPIO logic.

Reset

The controller uses a single reset signal associated with the LPD_LSBUS_CLK interface clock.

Register Overview

An overview of the GPIO registers is shown in Table 27-2.

Table 27-2: **GPIO Register Overview**

Function	Register Name	Description	Type
Data reads and data writes	gpio.MASK_DATA_{0:5}_LSW gpio.MASK_DATA_{0:5}_MSW	Bit masked data output writes.	Mixed
	gpio.DATA_{0:5}	Output data	R/W
	gpio.DATA_{0:5}_RO	Input data	RO
I/O buffer control	gpio.DIRM_{0:5}	Direction	R/W
	gpio.OEN_{0:5}	Output enable	R/W
Interrupt controls	gpio.INT_MASK_{0:5}	Interrupt mask	RO
	gpio.INT_EN_{0:5}	Interrupt enable	WO
	gpio.INT_DIS_{0:5}	Interrupt disable	WO
	gpio.INT_STAT_{0:5}	Interrupt status	WTC
	gpio.INT_TYPE_{0:5}	Interrupt type	RW
	gpio.INT_POLARITY_{0:5}	Interrupt polarity	RW
	gpio.INT_ANY_{0:5}	Interrupt any	RW

MIO Signals

All GPIO I/O pins routed through the MIO are listed in Table 27-3.

Table 27-3: **GPIO Interface Signals via MIO Pins**

MIO Pins					
GPIO 0	**I/O**	**GPIO 1**	**I/O**	**GPIO 2**	**I/O**
0	I/O	26	I/O	52	I/O
1	I/O	27	I/O	53	I/O
2	I/O	28	I/O	54	I/O
3	I/O	29	I/O	55	I/O
4	I/O	30	I/O	56	I/O
5	I/O	31	I/O	57	I/O
6	I/O	32	I/O	58	I/O
7	I/O	33	I/O	59	I/O
8	I/O	34	I/O	60	I/O

Table 27-3: **GPIO Interface Signals via MIO Pins *(Cont'd)***

MIO Pins					
GPIO 0	**I/O**	**GPIO 1**	**I/O**	**GPIO 2**	**I/O**
9	I/O	35	I/O	61	I/O
10	I/O	36	I/O	62	I/O
11	I/O	37	I/O	63	I/O
12	I/O	38	I/O	64	I/O
13	I/O	39	I/O	65	I/O
14	I/O	40	I/O	66	I/O
15	I/O	41	I/O	67	I/O
16	I/O	42	I/O	68	I/O
17	I/O	43	I/O	69	I/O
18	I/O	44	I/O	70	I/O
19	I/O	45	I/O	71	I/O
20	I/O	46	I/O	72	I/O
21	I/O	47	I/O	73	I/O
22	I/O	48	I/O	74	I/O
23	I/O	49	I/O	75	I/O
24	I/O	50	I/O	76	I/O
25	I/O	51	I/O	77	I/O

Programming Model

This section discusses GPIO programming models. The example flow in Figure 27-3 is programming a GPIO interrupt. In this case, Bank0 of the GPIO is configured to latch the switch input from the board and generate an interrupt when there is a status change on the switch. Bank 1 is configured as output to display LED with the corresponding Bank 0 interrupt.

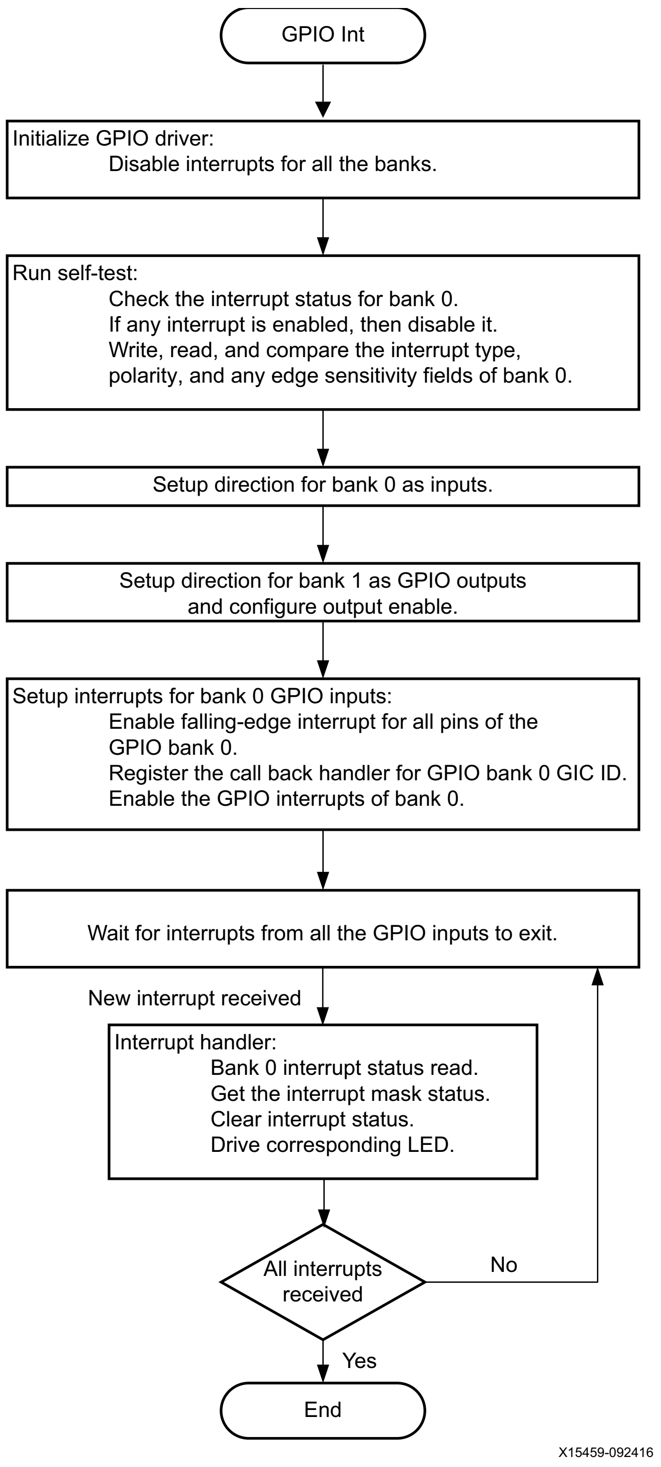

GPIO Int
Initialize GPIO driver:
Disable interrupts for all the banks.
Run self-test:
Check the interrupt status for bank 0.
If any interrupt is enabled, then disable it.
Write, read, and compare the interrupt type, polarity, and any edge sensitivity fields of bank 0.
Setup direction for bank 0 as inputs.
Setup direction for bank 1 as GPIO outputs and configure output enable.
Setup interrupts for bank 0 GPIO inputs:
Enable falling-edge interrupt for all pins of the GPIO bank 0.
Register the call back handler for GPIO bank 0 GIC ID.
Enable the GPIO interrupts of bank 0.
Wait for interrupts from all the GPIO inputs to exit.
New interrupt received
Interrupt handler:
Bank 0 interrupt status read.
Get the interrupt mask status.
Clear interrupt status.
Drive corresponding LED.
All interrupts received
No
Yes
End
X15459-092416

Figure 27-3: **GPIO Interrupt Programming Flow**

The register level configuration details are listed in Table 27-4 through Table 27-11 for the GPIO interrupt programming flow shown in Figure 27-3.

Initialize the GPIO Driver

Table 27-4 shows the registers used to initialize the GPIO driver and mask the interrupts for all the GPIO banks.

Table 27-4: **Registers Used to Initialize the GPIO Driver**

Task	Register Name	Bit Field	Register Offset	Bits	Value
Disable interrupts for GPIO bank 0	gpio.INT_DIS_0	INT_DISABLE_0	0x214	31:0	FFFF_FFFFh
Disable interrupts for GPIO bank 1	gpio.INT_DIS_1	INT_DISABLE_1	0x254	31:0	FFFF_FFFFh
Disable interrupts for GPIO bank 2	gpio.INT_DIS_2	INT_DISABLE_2	0x294	31:0	FFFF_FFFFh
Disable interrupts for GPIO bank 3	gpio.INT_DIS_3	INT_DISABLE_3	0x2D4	31:0	FFFF_FFFFh
Disable interrupts for GPIO bank 4	gpio.INT_DIS_4	INT_DISABLE_4	0x314	31:0	FFFF_FFFFh
Disable interrupts for GPIO bank 5	gpio.INT_DIS_5	INT_DISABLE_5	0x354	31:0	FFFF_FFFFh

Run Self-Test on the GPIO

Table 27-5 and Table 27-6 describe the flow to run self-test on the GPIO.

In Table 27-5, the tasks outline the flow to check the interrupt status for bank 0. If any interrupt is enabled, then disable the particular interrupt (bank 0).

Table 27-5: **Check the Interrupt Status and Disable Interrupts**

Task	Register Name	Register Field	Register Offset	Bits	Value
Get the interrupt mask status	gpio.INT_MASK_0	INT_MASK_0	0x20C	31:0	Read value
Disable interrupts for GPIO bank 0	gpio.INT_DIS_0	INT_DISABLE_0	0x214	31:0	Read value of previous step

The tasks in Table 27-6 outline the flow used to write, read, and compare the interrupt type, polarity, and any edge sensitivity fields of bank 0.

Table 27-6: **Read, Write, and Compare Interrupt Types**

Task	Register Name	Register Field	Register Offset	Bits	Value
Write interrupt type	gpio.INT_TYPE_0	INT_TYPE_0	0x21C	31:0	Test Value
Write interrupt polarity	gpio.INT_POLARITY_0	INT_POL_0	0x220	31:0	Test Value
Write interrupt any edge sensitivity	gpio.INT_ANY_0	INT_ON_ANY_0	0x224	31:0	Test Value
Read interrupt type	gpio.INT_TYPE_0	INT_TYPE_0	0x21C	31:0	
Read interrupt polarity	gpio.INT_POLARITY_0	INT_POL_0	0x220	31:0	
Read interrupt any edge sensitivity	gpio.INT_ANY_0	INT_ON_ANY_0	0x224	31:0	

Setup Direction for Bank 0 as Inputs

Table 27-7: **Setup the Direction for Bank 0 as Inputs**

Task	Register Name	Register Field	Register Offset	Bits	Value
Set bank 0 as input GPIO	gpio.DIRM_0	DIRECTION_0	0x204	31:0	0000_0000h

Setup Direction for Bank 1 as GPIO Outputs and Configure Output Enable

Table 27-8: **Setup Direction for Bank 1 as GPIO Outputs and Configure Output Enable**

Task	Register Name	Register Field	Register Offset	Bits	Value
Set Bank1 as output GPIO	gpio.DIRM_1	DIRECTION_1	0x244	31:0	FFFF_FFFFh
Set BANK1 output Enable	gpio.OEN_1	OP_ENABLE_1	0x248	31:0	FFFF_FFFFh

Setup Interrupts for Bank 0 GPIO Inputs

The tasks in Table 27-9 outline the flow used to enable a falling-edge interrupt for all pins of the GPIO bank 0.

Table 27-9: **Setup Interrupts for Bank 0 GPIO Inputs**

Task	Register Name	Register Field	Register Offset	Bits	Value
Write interrupt type	gpio.INT_TYPE_0	INT_TYPE_0	0x21C	31:0	0000_0000h
Write interrupt polarity	gpio.INT_POLARITY_0	INT_POL_0	0x220	31:0	0000_0000h
Write interrupt any edge sensitivity	gpio.INT_ANY_0	INT_ON_ANY_0	0x224	31:0	0000_0000h

The task in Table 27-10 outlines the flow used to register the call back handler for the GPIO Bank 0 GIC ID.

Table 27-10: **Enable the GPIO Interrupts of Bank 0**

Task	Register Name	Register Field	Register Offset	Bits	Value
Enable interrupts for GPIO bank 0	gpio.INT_EN_0	INT_ENABLE_0	0x210	31:0	FFFF_FFFFh

Wait for Interrupts from all the GPIO Inputs to Exit

The tasks in Table 27-10 outline the flow used for interrupt handling and to drive a GPIO write to glow LED.

Table 27-11: **Wait for Interrupts from all the GPIO Inputs to Exit**

Task	Register Name	Register Field	Register Offset	Bits	Value
Bank 0 interrupt status read	gpio.INT_STAT_0	INT_STATUS_0	`0x218`	31:0	
Get the interrupt mask status	gpio.INT_MASK_0	INT_MASK_0	`0x20C`	31:0	
Clear interrupt status	gpio.INT_STAT_0	INT_STATUS_0	`0x218`	31:0	Write a `1` to clear the particular interrupt.
Drive the corresponding LEDs	gpio.DATA_1	DATA_1	`0x44`	31:0	Accumulated interrupt status.

Chapter 28

Multiplexed I/O

Introduction

The features and functional description of the multiplexed I/Os (MIOs) are described in this chapter including the MIO signal routing, bank-level mapping, and pin assignment considerations for efficient use of the available MIO pins.

The basic MIO function is to multiplex access from the processing system (PS) peripheral interface pins to the appropriate peripheral interfaces, as defined in the configuration registers. An additional function is to control access from the extended multiplexed I/O interface (EMIO) block to the input signals of the peripheral interfaces, for instance, where there is a receive path. The MIO module allows you to configure the PS pin-out as required. Seventy-eight (78) of the general purpose I/Os (GPIO) are used as MIOs. They are configured by accessing the MIO control registers (detailed in this chapter) and are located in the system-level control, IOU_SLCR register set.

The 78 MIO signals are divided into three banks, and each bank includes 26 device pins. Each bank (500, 501, and 502) has its own power pins, VCCO_PSIO{0:2} for the hardware interface. The I/O logic and interface to the system are in the LPD power domain. The voltage signaling level, 1.8 or 3.3V, can be determined by reading the IOU_SLCR.bank{0:2} registers.

The boot device is assigned to a specific set of MIO pins (see Table 11-1). These assignments can drive the decisions on bank assignments for interfacing with other hardware.

Overview of the Blocks Function

The MIO module can be described as a wide multiplexer/de-multiplexer, routing a number of different peripheral interfaces to a limited number of external pins under software configuration. A number of different interfaces are routed to and from the pins by the MIO, with varying timing requirements. Therefore, a priority structure based on maximum toggle rates must be implemented to place high-speed signal interfaces (such as gigabit Ethernet RGMII or USB ULPI) closer to the pin in the multiplexer tree structure.

Control of the functionality associated with each pin is through the MIO section of the IOU_SLCR system-level control registers. Output control signals are generated from these register settings. These signals are used either directly as multiplexer selects or indirectly through multiplexer select remapping functions. There are multiple port mapping options available for peripherals (e.g., 12 for CAN and I2C) where the interface to the peripheral can be constructed using any of the following.

- Mapping of ports from a single group.
- Mapping of ports from different groups.
- A mix of PS pins and PL pins through the EMIO interface.

PS and PL Pins

The MIO is fundamental to the I/O peripheral connections due to the limited number of MIO pins (Figure 28-1). Software programs the routing of the I/O signals to the MIO pins. The I/O peripheral signals can also be routed to the PL (including PL device pins) through the EMIO interface. This is used to gain access to more device pins (PL pins) and to allow an I/O peripheral controller to interface to internal logic in the PL.

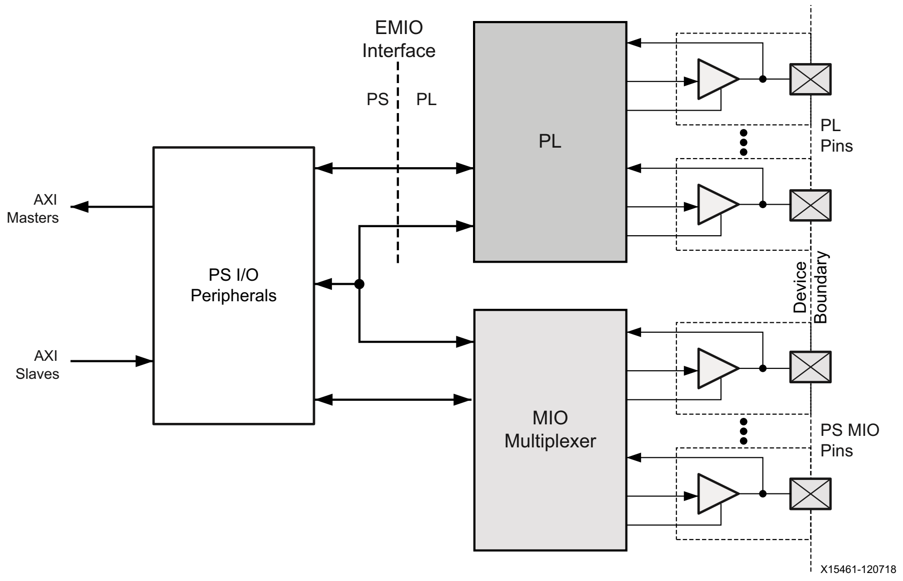


Figure 28-1: **MIO-EMIO Wiring Diagram**

Output Multiplexer

The output multiplexer example in Figure 28-2 shows a single bit cell of an output multiplexer. The l3_output_* signal name is used to denote any of the range of low-speed peripherals, where the ordering is not significant. To illustrate the general multiplexing structure, other interfaces are identified without specifying a particular signal. Interfaces of similar speed can be swapped at each multiplexer level. For instance, the fast trace-port interface can be used where neither the ULPI nor RGMII PHY interfaces are used.

Figure 28-2 shows the default multiplexer structure for the output and enable multiplexer. For most pins, only one of the high-speed interfaces (RGMII or ULPI or trace) is present. Similarly, for many signals generated or consumed by peripherals, there is no corresponding 3-state enable under the implemented protocol for its external interface. For example, because RGMII does not use 3-state enables, the diagram includes them to illustrate the concept of the output enable shadowing the output signal.

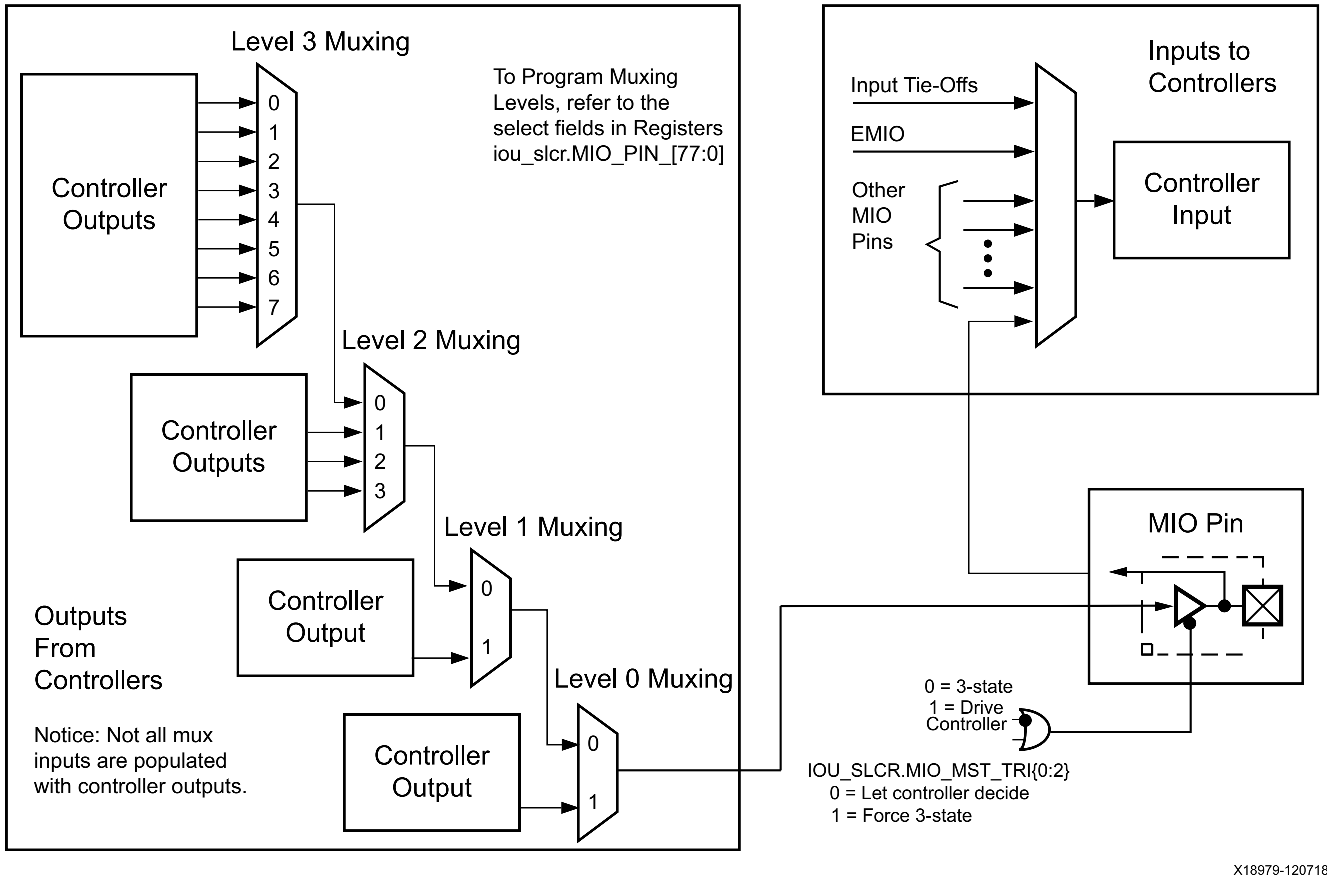


Figure 28-2: **MIO Multiplexing Stages and 3-State Output Control**

Master 3-state Enables

As shown in Figure 28-2, each pin has a master 3-state enable that overrides any interface specific output enable provided by the peripherals. The master enable is logically combined with the interface specific output enable signals (if provided) currently selected by the output enable multiplexer tree to produce a single output enable for connection to the I/O cell.

Access to the master enable control registers is on a bit-by-bit basis as the pins are configured or in parallel by accessing two 32-bit registers.

Default Logic Levels

The inputs to the I/O peripherals are driven with default values when another source is not routed to either the MIO or the EMIO. If an input is routed to EMIO, but the PL is powered down, then the same default value is driven to the I/O peripheral (see Figure 28-3.)

For MIO-only signals, the default signal input is driven when the MIO multiplexer does not route the signal to an MIO pin.

For MIO-EMIO signals, the default signal input is driven when the MIO multiplexer does not route the signal to an MIO pin (the signal defaults to the EMIO interface) and when the signal is programmed to be routed through the EMIO, but the PL either does not drive the signal (not configured) or is not able to drive it (powered down).

The default input signal logic levels are designed to be benign to the I/O peripheral. As a precaution, the related peripheral core should also be disabled when not in use. The logic levels are shown in the signal tables in each chapter for each I/O peripheral.

When PS_POR_B is asserted Low, the PS GPIO outputs connected to EMIO are forced and held High.

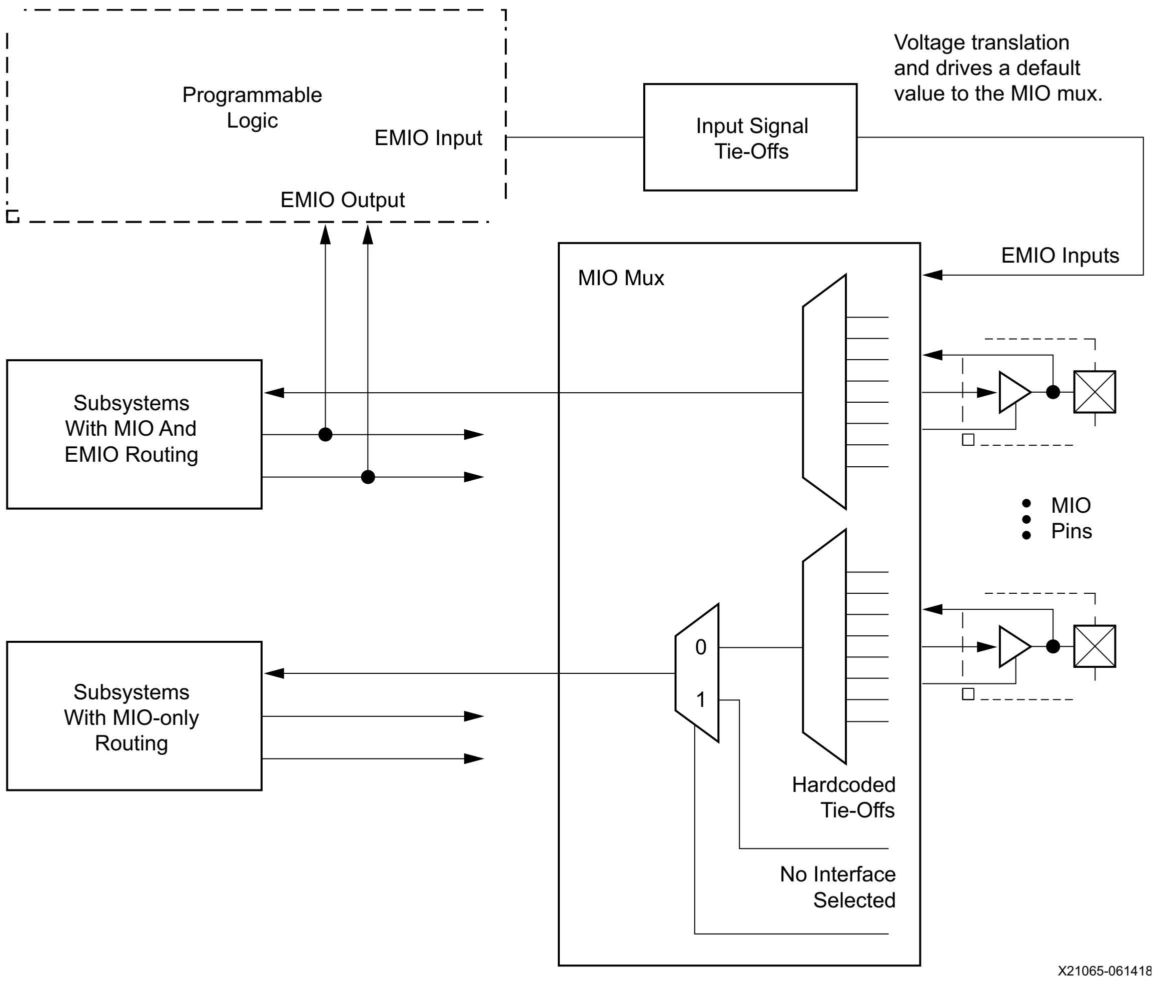


Figure 28-3: **Non-selected Controller Inputs**

MIO Pin Assignment Considerations

IMPORTANT: *There are several important MIO pin assignment considerations. The MIO-at-a-Glance table and these pin assignment considerations are helpful for pin planning. There are individual MIO signal tables for each controller/unit that uses the MIO pins.*

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.

I/O Buffer Output Enable Control

The output enable for each MIO I/O buffer is controlled by a combination of the setting of the three-state override control bit, the selected signal type (input-only or not), and the state of the peripheral controller. The three-state override bit can be controlled from either of two places: the iou_slcr.MIO_PIN_xx register bit or the iou_slcr.MIO_MST_TRIx register bits. These bits control the same flip-flop to help control the three-state signal of the I/O buffer. The I/O buffer output is enabled when the three-state override control bit equals `0` and either the signal is an output-only or the I/O peripheral is driving a signal that is configured as I/O.

Boot from SD Card

The BootROM expects the SD card to be connected to MIO pins 13 through 25 for SD0 and MIO pins 39 through 51 for SD1.

eMMC Mapping

The SD1/eMMC can only operate in 4-bit mode when it is mapped to MIO bank 3.

Quad-SPI Interface

The lower memory Quad-SPI interface (QSPI_0) must be used when using the Quad-SPI memory subsystem. The upper interface (QSPI_1) is optional and is only used for a two-memory arrangement (parallel or stacked). Do not use the Quad-SPI 1 interface alone.

Drive Strength

After power up, the default I/O setting of the MIO banks 0, 1, and 2 is 8 mA.

MIO Table at a Glance

For pin planning, see Table 28-1. MIO signals are also listed in each controller chapter along with their function, direction, and presence in EMIO.

Table 28-1: **MIO Interfaces**

Interface	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
gem0																										
gem1																										
gem2																										
gem3																										
gem_tsu																										
qspi[2]	4	1	2	3	0	5	12	6	8	9	10	11	7													
nand										2	13	14	4	1	3	0	5	6	7	8	9	10	16	11	12	15
pcie																										
usb0																										
usb1																										
pmu																										
sd0[1]														4	5	6	7	8	9	10	11	3	2	13	1	0
sd1[1]																										
CSU tamper																			0	0	0	0	0	0	0	0
DisplayPort aux																										
gpio0	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
gpio1																										
gpio2																										
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
i2c0			0	1			0	1			0	1			0	1			0	1			0	1		

Interface	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
gem0	0	1	2	3	4	5	6	7	8	9	10	11														
gem1													0	1	2	3	4	5	6	7	8	9	10	11		
gem2																										
gem3																										
gem_tsu																									0	0
qspi[2]																										
nand	2	13	14				4																			
pcie				0	0	0		0	0	0	0	0														
usb0																										
usb1																										
pmu	0	1	2	3	4	5	6	7	8	9	10	11														
sd0[1]													2	1	3	4	5	6	7	8	9	10	11	13	0	
sd1[1]														8	9	10	11	13	0	1	4	5	6	7	3	2
CSU tamper	0					0	0	0																		
DisplayPort aux		0	1	2	3				0	1	2	3														
gpio0																										
gpio1	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
gpio2																										
can0	0	1			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		
i2c0	0	1			0	1			0	1			0	1			0	1			0	1			0	1

Interface	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77
gem0																										
gem1																										
gem2	0	1	2	3	4	5	6	7	8	9	10	11														
gem3													0	1	2	3	4	5	6	7	8	9	10	11		
gem_tsu																										
qspi[2]																										
nand																										
pcie																										
usb0	0	1	6	3	4	5	2	7	8	9	10	11														
usb1													0	1	6	3	4	5	2	7	8	9	10	11		
pmu																										
sd0[1]													2	1	3	4	5	6	7	8	9	10	11	13	0	
sd1[1]																		0	13	4	5	6	7	3	2	1
CSU tamper																										
DisplayPort aux																										
gpio0																										
gpio1																										
gpio2	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
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
i2c0			0	1			0	1			0	1			0	1			0	1			0	1		

Table 28-1: **MIO Interfaces *(Cont'd)***

Interface	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
i2c1	0	1			0	1			0	1			0	1			0	1			0	1			0	1
pjtag	3	0	1	2									3	0	1	2										
lpd_swdt							0	1			0	1							0	1			0	1		
fpd_swdt					0	1			0	1							0	1			0	1			0	1
spi0	5	4	3	2	1	0							5	4	3	2	1	0								
spi1							5	4	3	2	1	0							1	4	3	2	5	0		
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							0	1
mdio{0:3}																										
ua0			0	1			0	1			0	1			0	1			0	1			0	1		
ua1	1	0			1	0			1	0			1	0			1	0			1	0			1	0
trace	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17								

Interface	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
i2c1			0	1			0	1			0	1			0	1			0	1			0	1		
pjtag	3	0	1	2									3	0	1	2										
lpd_swdt					0	1			0	1							0	1			0	1			0	1
fpd_swdt							0	1			0	1							0	1			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		
ttc0					0	1							0	1							0	1				
ttc1			0	1							0	1							0	1						
ttc2	0	1							0	1							0	1							0	1
ttc3							0	1							0	1							0	1		
mdio{0:3}																									0	1
ua0	0	1			0	1			0	1			0	1			0	1			0	1			0	1
ua1			1	0			1	0			1	0			1	0			1	0			1	0		
trace	6	7	8	9	10	11	12	13	14	15	16	17	0	1	2	3	4	5								

Interface	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77
i2c1	0	1			0	1			0	1			0	1			0	1			0	1			0	1
pjtag	3	0	1	2			3	0	1	2																
lpd_swdt											0	1			0	1			0	1			0	1		
fpd_swdt					0	1							0	1			0	1			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		
ttc0			0	1							0	1							0	1						
ttc1	0	1							0	1							0	1								
ttc2							0	1							0	1										
ttc3					0	1							0	1												
mdio{0:3}																									0	1
ua0			0	1			0	1			0	1			0	1			0	1			0	1		
ua1	1	0			1	0			1	0			1	0			1	0			1	0				
trace	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17								

Notes:

1. SD0/1 peripheral pins can also be configured as eMMC 0/1, respectively. The difference between SD and eMMC configuration is as follows.
    - The **Card Detect** and **Write Protect** signals are only available in SD mode.
    - The BUS_POW pin in SD mode is treated as a reset pin in eMMC mode.
    - In SD mode, data transfers in 1-bit and 4-bit modes. In eMMC mode, data transfers in 1-bit, 4-bit, and 8-bit modes.
    - If the SD interface is configured for SD 3.0, the signals SEL, DIR_CMD, DIR_0, and DIR_1_3 are mapped to sdio{0,1}_data_out [4], [5], [6], and [7], respectively.
2. In Quad-SPI loopback mode, leave the clk_for_lpbk signal floating. In Quad-SPI non-loopback mode, the clk_for_lpbk signal is not used by the Quad-SPI and can be used as a peripheral I/O (such as GPIO, CAN, or I2C).

Register Overview

Some MIO pins are programmed by the PMU ROM pre-boot. Some might also be programmed by the PMU user firmware, the CSU for the boot device, the FSBL, or other low-level code. The affected registers are listed in Table 28-2. All MIO registers use the IOU_SLCR register set and can be programmed in any order.

Table 28-2: **MIO Control Registers**

Description	Register Name	Type
Route I/O signals of IOP peripherals to MIO pins {0:77}.	MIO_PIN_{0:77}	R/W
Disable 3-state output buffers on MIO pins {0:77}.	MIO_MST_TRI{0:2}	R/W
Select input type (CMOS or Schmitt with hysteresis).	BANK{0:2}_CTRL3	R/W
Select internal pull-up or pull-down.	BANK{0:2}_CTRL4	R/W
Enable or disable internal resister.	BANK{0:2}_CTRL5	R/W
Select slew rate output (fast or slow).	BANK{0:2}_CTRL6	R/W
Select output drive strength (2 bits; 2, 4, 8, and 12 mA).	BANK{0:2}_CTRL{0, 1}	R /W
Read the voltage applied to PSIO bank.	BANK{0:2}_STATUS	R
Enable loopback function with MIO for SPI, UART, CAN, and I2C I/O interfaces.	MIO_LOOPBACK	R/W

Note: Setting MIO_MST_TRIx [PIN_xx_TRI] to `0` enables the GPIO to control the 3-state mode of the I/O. If the 3-state control is set to `1` in the MIO, then the output driver will be set to 3-state regardless of the GPIO settings.

Programming Model

Typically, the MIO configuration code is generated as part of the FSBL from the hardware project. An SDK export of a Vivado Design Suite project carries the PCW configuration information for the MIO pins. The SDK tools process the MIO configuration during FSBL creation.

I2C Interface Programming Example

The MIO can be configured to route the I2C interface signals to MIO pins 2 and 3. To route the I2C SCL signal to MIO pin 2, write `'h40` to the IOU_SLCR.MIO_PIN_2 register. To route the I2C SDA signal to MIO pin 3, write `'h40` to the IOU_SLCR.MIO_PIN_3 register.

The MIO pins and their signal names for each interface are listed in Table 28-3.

Table 28-3: **MIO Interfaces**

	Interface Type																																						
	gem0	gem1	gem2	gem 3	gem_ tsu	qspi[2]	nand	pcie	usb0	usb1	pmu	sd0[1]	sd1[1]	test_ scan	csu	dpaux	gpio0	gpio1	gpio2	can0	can1	i2c0	i2c1	mdio0	pjtag	lpd_ swdt	fpd_ swdt	mdio1	spi0	spi1	mdio2	ttc0	ttc1	ttc2	ttc3	mdio3	ua0	ua1	trace
Size	12	12	12	12	1	13	17	1	12	12	12	13	13	38	1	4	26	26	26	2	2	2	2	2	4	2	2	2	6	6	2	2	2	2	2	2	2	2	18
Pin																																							
0						sclk_out 4								io[0] 0			io[0] 0				phy_tx 1		scl 0		tck 3				sclk 5						clk 0			txd 1	clk 0
1						io[1] 1								io[1] 1			io[1] 1				phy_rx 0		sda 1		tdi 0				n_ss_ out[2] 4						wave_ out 1			rxd 0	clk 1
2						io[2] 2								io[2] 2			io[2] 2			phy_rx 0		scl 0			tdo 1				n_ss_ out[1] 3					clk 0			rxd 0		dq_0 2
3						io[3] 3								io[3] 3			io[3] 3			phy_tx 1		sda 1			tms 2				n_ss_ out[0] 2					wave_ out 1			txd 1		dq_1 3
4						si_mio[0] 0								io[4] 4			io[4] 4				phy_tx 1		scl 0				clk_in 0		miso 1				clk 0					txd 1	dq_2 4
5						n_ss_ou t 5								io[5] 5			io[5] 5				phy_rx 0		sda 1				rst_out 1		mosi 0				wave_ out 1					rxd 0	dq_3 5
6						clk_for_l pbk 12								io[6] 6			io[6] 6			phy_rx 0		scl 0				clk_in 0				sclk 5		clk 0					rxd 0		dq_4 6
7						n_ss_ou t_ upper 6								io[7] 7			io[7] 7			phy_tx 1		sda 1				rst_out 1				n_ss_ out[2] 4		wave_ out 1					txd 1		dq_5 7
8						upper _io[0] 8								io[8] 8			io[8] 8				phy_tx 1		scl 0				clk_in 0			n_ss_ out[1] 3					clk 0			txd 1	dq_6 8
9						upper _io[1] 9	ce[1] 2							io[9] 9			io[9] 9				phy_rx 0		sda 1				rst_out 1			n_ss_ out[0] 2					wave_ out 1			rxd 0	dq_7 9
10						upper _io[2] 10	rb_n[0] 13							io[10] 10			io[10] 10			phy_rx 0		scl 0				clk_in 0				miso 1				clk 0			rxd 0		dq_8 10
11						upper _io[3] 11	rb_n[1] 14							io[11] 11			io[11] 11			phy_tx 1		sda 1				rst_out 1				mosi 0				wave_ out 1			txd 1		dq_9 11
12						sclk_ out_ upper 7	dqs 4							io[12] 12			io[12] 12				phy_tx 1		scl 0		tck 3				sclk 5				clk 0					txd 1	dq_10 12

The MIO pins and their signal names for each interface are listed in Table 28-3.

Table 28-3: **MIO Interfaces *(Cont'd)***

	Interface Type																																						
	gem0	gem1	gem2	gem 3	gem_ tsu	qspi[2]	nand	pcie	usb0	usb1	pmu	sd0[1]	sd1[1]	test_ scan	csu	dpaux	gpio0	gpio1	gpio2	can0	can1	i2c0	i2c1	mdio0	pjtag	lpd_ swdt	fpd_ swdt	mdio1	spi0	spi1	mdio2	ttc0	ttc1	ttc2	ttc3	mdio3	ua0	ua1	trace
13							ce[0] 1					data_io [0] 4		io[13] 13			io[13] 13				phy_rx 0		sda 1		tdi 0				n_ss_ out[2] 4				wave_ out 1					rxd 0	dq_11 13
14							cle 3					data_io [1] 5		io[14] 14			io[14] 14			phy_rx 0		scl 0			tdo 1				n_ss_ out[1] 3			clk 0					rxd 0		dq_12 14
15							ale 0					data_io [2] 6		io[15] 15			io[15] 15			phy_tx 1		sda 1			tms 2				n_ss_ out[0] 2			wave_ out 1					txd 1		dq_13 15
16							dq[0] 5					data_io [3] 7		io[16] 16			io[16] 16				phy_tx 1		scl 0				clk_in 0		miso 1						clk 0			txd 1	dq_14 16
17							dq[1] 6					data_io [4] 8		io[17] 17			io[17] 17				phy_rx 0		sda 1				rst_out 1		mosi 0						wave_ out 1			rxd 0	dq_15 17
18							dq[2] 7					data_io [5] 9		io[18] 18	ext_ tamper 0		io[18] 18			phy_rx 0		scl 0				clk_in 0				miso 1				clk 0			rxd 0		
19							dq[3] 8					data_io [6] 10		io[19] 19	ext_ tamper 0		io[19] 19			phy_tx 1		sda 1				rst_out 1				n_ss_ out[2] 4				wave_ out 1			txd 1		
20							dq[4] 9					data_io [7] 11		io[20] 20	ext_ tamper 0		io[20] 20				phy_tx 1		scl 0				clk_in 0			n_ss_ out[1] 3			clk 0					txd 1	
21							dq[5] 10					cmd_io 3		io[21] 21	ext_ tamper 0		io[21] 21				phy_rx 0		sda 1				rst_out 1			n_ss_ out[0] 2			wave_ out 1					rxd 0	
22							we_b 16					clk_out 2		io[22] 22	ext_ tamper 0		io[22] 22			phy_rx 0		scl 0				clk_in 0				sclk 5		clk 0					rxd 0		
23							dq[6] 11					bus_po w 13		io[23] 23	ext_ tamper 0		io[23] 23			phy_tx 1		sda 1				rst_out 1				mosi 0		wave_ out 1					txd 1		
24							dq[7] 12					cd_n 1		io[24] 24	ext_ tamper 0		io[24] 24				phy_tx 1		scl 0				clk_in 0								clk 0			txd 1	
25							re_n 15					wp 0		io[25] 25	ext_ tamper 0		io[25] 25				phy_rx 0		sda 1				rst_out 1								wave_ out 1			rxd 0	
26	rgmii_ tx_clk 0				gem_ tsu_clk 0		ce[1] 2				gpi[0] 0			io[26] 26	ext_ tamper 0			io[0] 0		phy_rx 0		scl 0			tck 3				sclk 5					clk 0			rxd 0		dq_4 6
27	rgmii_ txd[0] 1						rb_n[0] 13				gpi[1] 1			io[27] 27		data_ out 0		io[1] 1		phy_tx 1		sda 1			tdi 0				n_ss_ out[2] 4					wave_ out 1			txd 1		dq_5 7

The MIO pins and their signal names for each interface are listed in Table 28-3.

Table 28-3: **MIO Interfaces *(Cont'd)***

	Interface Type																																						
	gem0	gem1	gem2	gem3	gem_tsu	qspi[2]	nand	pcie	usb0	usb1	pmu	sd0[1]	sd1[1]	test_scan	csu	dpaux	gpio0	gpio1	gpio2	can0	can1	i2c0	i2c1	mdio0	pjtag	lpd_swdt	fpd_swdt	mdio1	spi0	spi1	mdio2	ttc0	ttc1	ttc2	ttc3	mdio3	ua0	ua1	trace
28	rgmii_txd[1] 2						rb_n[1] 14				gpi[2] 2			io[28] 28		hot_plug_detect 1		io[2] 2			phy_tx 1		scl 0		tdo 1				n_ss_out[1] 3				clk 0					txd 1	dq_6 8
29	rgmii_txd[2] 3							reset_n 0			gpi[3] 3			io[29] 29		data_oe 2		io[3] 3			phy_rx 0		sda 1		tms 2				n_ss_out[0] 2				wave_out 1					rxd 0	dq_7 9
30	rgmii_txd[3] 4							reset_n 0			gpi[4] 4			io[30] 30		data_in 3		io[4] 4		phy_rx 0		scl 0				clk_in 0			miso 1			clk 0					rxd 0		dq_8 10
31	rgmii_tx_ctl 5							reset_n 0			gpi[5] 5			io[31] 31	ext_tamper 0			io[5] 5		phy_tx 1		sda 1				rst_out 1			mosi 0			wave_out 1					txd 1		dq_9 11
32	rgmii_rx_clk 6						dqs 4				gpo[0] 6			io[32] 32	ext_tamper 0			io[6] 6			phy_tx 1		scl 0				clk_in 0			sclk 5					clk 0			txd 1	dq_10 12
33	rgmii_rxd[0] 7							reset_n 0			gpo[1] 7			io[33] 33	ext_tamper 4			io[7] 7			phy_rx 0		sda 1				rst_out 1			n_ss_out[2] 4					wave_out 1			rxd 0	dq_11 13
34	rgmii_rxd[1] 8							reset_n 0			gpo[2] 8			io[34] 34		data_out 0		io[8] 8		phy_rx 0		scl 0				clk_in 0				n_ss_out[1] 3				clk 0			rxd 0		dq_12 14
35	rgmii_rxd[2] 9							reset_n 0			gpo[3] 9			io[35] 35		hot_plug_detect 1		io[9] 9		phy_tx 1		sda 1				rst_out 1				n_ss_out[0] 2				wave_out 1			txd 1		dq_13 15
36	rgmii_rxd[3] 10							reset_n 0			gpo[4] 10			io[36] 36		data_oe 2		io[10] 10			phy_tx 1		scl 0				clk_in 0			miso 1			clk 0					txd 1	dq_14 16
37	rgmii_rx_ctl 11							reset_n 0			gpo[5] 11			io[37] 37		data_in 3		io[11] 11			phy_rx 0		sda 1				rst_out 1			mosi 0			wave_out 1					rxd 0	dq_15 17
38		rgmii_tx_clk 0										clk_out 2						io[12] 12		phy_rx 0		scl 0			tck 3				sclk 5			clk 0					rxd 0		clk 0
39		rgmii_txd[0] 1										cd_n 1	data_io[4] 8					io[13] 13		phy_tx 1		sda 1			tdi 0				n_ss_out[2] 4			wave_out 1					txd 1		clk 1
40		rgmii_txd[1] 2										cmd_io 3	data_io[5] 9					io[14] 14			phy_tx 1		scl 0		tdo 1				n_ss_out[1] 3						clk 0			txd 1	dq_0 2
41		rgmii_txd[2] 3										data_io[0] 4	data_io[6] 10					io[15] 15			phy_rx 0		sda 1		tms 2				n_ss_out[0] 2						wave_out 1			rxd 0	dq_1 3

The MIO pins and their signal names for each interface are listed in Table 28-3.

Table 28-3: **MIO Interfaces *(Cont'd)***

	Interface Type																																					
	gem0	gem1	gem2	gem 3	gem_ tsu	qspi(2)	nand	pcie	usb0	usb1	pmu	sd0(1)	sd1(1)	test_ scan	csu	dpaux	gpio0	gpio1	gpio2	can0	can1	i2c0	i2c1	mdio0	pjtag	lpd_ swdt	fpd_ swdt	mdio1	spi0	spi1	mdio2	ttc0	ttc1	ttc2	ttc3	mdio3	ua0	ua1	trace
42		rgmii_txd[3] 4										data_io[1] 5	data_io[7] 11					io[16] 16		phy_rx 0		scl 0				clk_in 0			miso 1					clk 0			rxd 0		dq_2 4
43		rgmii_tx_ctl 5										data_io[2] 6	bus_pow 13					io[17] 17		phy_tx 1		sda 1				rst_out 1			mosi 0					wave_out 1			txd 1		dq_3 5
44		rgmii_rx_clk 6										data_io[3] 7	wp 0					io[18] 18			phy_tx 1		scl 0				clk_in 0			sclk 5			clk 0					txd 1	
45		rgmii_rxd[0] 7										data_io[4] 8	cd_n 1					io[19] 19			phy_rx 0		sda 1				rst_out 1			n_ss_out[2] 4			wave_out 1					rxd 0	
46		rgmii_rxd[1] 8										data_io[5] 9	data_io[0] 4					io[20] 20		phy_rx 0		scl 0				clk_in 0				n_ss_out[1] 3		clk 0					rxd 0		
47		rgmii_rxd[2] 9										data_io[6] 10	data_io[1] 5					io[21] 21		phy_tx 1		sda 1				rst_out 1				n_ss_out[0] 2		wave_out 1					txd 1		
48		rgmii_rxd[3] 10										data_io[7] 11	data_io[2] 6					io[22] 22			phy_tx 1		scl 0				clk_in 0			miso 1					clk 0			txd 1	
49		rgmii_rx_ctl 11										bus_pow 13	data_io[3] 7					io[23] 23			phy_rx 0		sda 1				rst_out 1			mosi 0					wave_out 1			rxd 0	
50					gem_tsu_clk 0							wp 0	cmd_io 3					io[24] 24		phy_rx 0		scl 0				clk_in 0		gem1_mdc 0						clk 0			rxd 0		
51					gem_tsu_clk 0								clk_out 2					io[25] 25		phy_tx 1		sda 1				rst_out 1		gem1_mdio 1						wave_out 1			txd 1		
52			rgmii_tx_clk 0						ulpi_clk_in 0										io[0] 0		phy_tx 1		scl 0		tck 3				sclk 5				clk 0					txd 1	clk 0
53			rgmii_txd[0] 1						ulpi_dir 1										io[1] 1		phy_rx 0		sda 1		tdi 0				n_ss_out[2] 4				wave_out 1					rxd 0	clk 1
54			rgmii_txd[1] 2						ulpi_rx_data[2] 6										io[2] 2	phy_rx 0		scl 0			tdo 1				n_ss_out[1] 3			clk 0					rxd 0		dq_0 2
55			rgmii_txd[2] 3						ulpi_nxt 3										io[3] 3	phy_tx 1		sda 1			tms 2				n_ss_out[0] 2			wave_out 1					txd 1		dq_1 3

The MIO pins and their signal names for each interface are listed in Table 28-3.

Table 28-3: **MIO Interfaces *(Cont'd)***

	Interface Type																																					
	gem0	gem1	gem2	gem3	gem_tsu	qspi[2]	nand	pcie	usb0	usb1	pmu	sd0[1]	sd1[1]	test_scan	csu	dpaux	gpio0	gpio1	gpio2	can0	can1	i2c0	i2c1	mdio0	pjtag	lpd_swdt	fpd_swdt	mdio1	spi0	spi1	mdio2	ttc0	ttc1	ttc2	ttc3	mdio3	ua0	ua1	trace
56			rgmii_txd[3] 4						ulpi_rx_data[0] 4										io[4] 4		phy_tx 1		scl 0				clk_in 0		miso 1						clk 0			txd 1	dq_2 4
57			rgmii_tx_ctl 5						ulpi_rx_data[1] 5										io[5] 5		phy_rx 0		sda 1				rst_out 1		mosi 0						wave_out 1			rxd 0	dq_3 5
58			rgmii_rx_clk 6						ulpi_stp 2										io[6] 6	phy_rx 0		scl 0			tck 3					sclk 5				clk 0			rxd 0		dq_4 6
59			rgmii_rxd[0] 7						ulpi_rx_data[3] 7										io[7] 7	phy_tx 1		sda 1			tdi 0					n_ss_out[2] 4				wave_out 1			txd 1		dq_5 7
60			rgmii_rxd[1] 8						ulpi_rx_data[4] 8										io[8] 8		phy_tx 1		scl 0		tdo 1					n_ss_out[1] 3			clk 0					txd 1	dq_6 8
61			rgmii_rxd[2] 9						ulpi_rx_data[5] 9										io[9] 9		phy_rx 0		sda 1		tms 2					n_ss_out[0] 2			wave_out 1					rxd 0	dq_7 9
62			rgmii_rxd[3] 10						ulpi_rx_data[6] 10										io[10] 10	phy_rx 0		scl 0				clk_in 0				miso 1		clk 0					rxd 0		dq_8 10
63			rgmii_rx_ctl 11						ulpi_rx_data[7] 11										io[11] 11	phy_tx 1		sda 1				rst_out 1				mosi 0		wave_out 1					txd 1		dq_9 11
64				rgmii_tx_clk 0						ulpi_clk_in 0		clk_out 2							io[12] 12		phy_tx 1		scl 0				clk_in 0		sclk 5						clk 0			txd 1	dq_10 12
65				rgmii_txd[0] 1						ulpi_dir 1		cd_n 1							io[13] 13		phy_rx 0		sda 1				rst_out 1		n_ss_out[2] 4						wave_out 1			rxd 0	dq_11 13
66				rgmii_txd[1] 2						ulpi_rx_data[2] 6		cmd_io 3							io[14] 14	phy_rx 0		scl 0				clk_in 0			n_ss_out[1] 3					clk 0			rxd 0		dq_12 14
67				rgmii_txd[2] 3						ulpi_nxt 3		data_io[0] 4							io[15] 15	phy_tx 1		sda 1				rst_out 1			n_ss_out[0] 2					wave_out 1			txd 1		dq_13 15
68				rgmii_txd[3] 4						ulpi_rx_data[0] 4		data_io[1] 5							io[16] 16		phy_tx 1		scl 0				clk_in 0		miso 1				clk 0					txd 1	dq_14 16

The MIO pins and their signal names for each interface are listed in Table 28-3.

Table 28-3: **MIO Interfaces *(Cont'd)***

	Interface Type																																						
	gem0	**gem1**	**gem2**	**gem 3**	**gem_ tsu**	**qspi[2]**	**nand**	**pcie**	**usb0**	**usb1**	**pmu**	**sd0[1]**	**sd1[1]**	**test_ scan**	**csu**	**dpaux**	**gpio0**	**gpio1**	**gpio2**	**can0**	**can1**	**i2c0**	**i2c1**	**mdio0**	**pjtag**	**lpd_ swdt**	**fpd_ swdt**	**mdio1**	**spi0**	**spi1**	**mdio2**	**ttc0**	**ttc1**	**ttc2**	**ttc3**	**mdio3**	**ua0**	**ua1**	**trace**
69				rgmii_tx_ctl 5						ulpi_rx_data[1] 5		data_io[2] 6	wp 0						io[17] 17		phy_rx 0		sda 1				rst_out 1		mosi 0				wave_out 1					rxd 0	dq_15 17
70				rgmii_rx_clk 6						ulpi_stp 2		data_io[3] 7	bus_pow 13						io[18] 18	phy_rx 0		scl 0				clk_in 0				sclk 5		clk 0					rxd 0		
71				rgmii_rxd[0] 7						ulpi_rx_data[3] 7		data_io[4] 8	data_io[0] 4						io[19] 19	phy_tx 1		sda 1				rst_out 1				n_ss_out[2] 4		wave_out 1					txd 1		
72				rgmii_rxd[1] 8						ulpi_rx_data[4] 8		data_io[5] 9	data_io[1] 5						io[20] 20		phy_tx 1		scl 0				clk_in 0			n_ss_out[1] 3								txd 1	
73				rgmii_rxd[2] 9						ulpi_rx_data[5] 9		data_io[6] 10	data_io[2] 6						io[21] 21		phy_rx 0		sda 1				rst_out 1			n_ss_out[0] 2								rxd 0	
74				rgmii_rxd[3] 10						ulpi_rx_data[6] 10		data_io[7] 11	data_io[3] 7						io[22] 22	phy_rx 0		scl 0				clk_in 0				miso 1							rxd 0		
75				rgmii_rx_ctl 11						ulpi_rx_data[7] 11		bus_pow 13	cmd_io 3						io[23] 23	phy_tx 1		sda 1				rst_out 1				mosi 0							txd 1		
76												wp 0	clk_out 2						io[24] 24		phy_tx 1		scl 0	gem0_mdc 0				gem1_mdc 0			gem2_mdc 0					gem3_mdc 0			
77													cd_n 1						io[25] 25		phy_rx 0		sda 1	gem0_mdio 1				gem1_mdio 1			gem2_mdio 1					gem3_mdio 1			

Notes:

1. SD0/1 peripheral pins can also be configured as eMMC 0/1, respectively. The difference between SD and eMMC configuration is as follows.
   - The **Card Detect** and **Write Protect** signals are only available in SD mode.
   - The BUS_POW pin in SD mode is treated as a reset pin in eMMC mode.
   - In SD mode, data transfers in 1-bit and 4-bit modes. In eMMC mode, data transfers in 1-bit, 4-bit, and 8-bit modes.
   - If the SD interface is configured for SD 3.0, the signals SEL, DIR_CMD, DIR_0, and DIR_1_3 are mapped to data_io[4], data_io[5], data_io[6], and data_io[7], respectively.
2. In Quad-SPI loopback mode, leave the clk_for_lpbk signal floating. In Quad-SPI non-loopback mode, the clk_for_lpbk signal is not used by the Quad-SPI and can be used as a peripheral I/O (such as GPIO, CAN, or I2C).

PS-GTR Transceivers

Introduction

The multi-gigabit GTR transceivers provide I/O for high-speed communication links between the media access controllers (MACs) of the peripherals in the serial input output unit (SIOU) and their link partners outside the device. The four programmable transceivers support the sublayer protocols with data rates up to 6 Gb/s.

The PS-GTR transceivers provide the only I/O path for the PCIe v2.0, USB3.0, DisplayPort (transmitter only), GEM Ethernet, and SATA controllers. The interface's interconnect matrix (ICM) connects up to four MAC I/O signals from the controllers to the physical coding sublayer (PCS) and the physical media attachment (PMA) units in the transceiver interfaces.

The PCS provides 8B/10B encoding and decoding, elastic buffer, and buffer management logic such as comma detection and byte and word alignment.

The PMA provides one PLL per lane with the ability to share reference clocks, transmitter de-emphasis, receiver continuous time linear equalizer, SSC support, out-of-band signaling, and LFPS/Beacon signaling for USB3.0/PCIe v2.0 designs.

The PS-GTR transceivers are controlled by the SERDES register set that are exclusively programmed and managed by the Vivado PS configuration wizard (PCW). This chapter explains the detailed functionality of the GTR transceivers and system functions. The system block diagram for the SIOU and GTR transceivers are shown in Figure 29-1.

Note: The high-speed serial I/O controllers in the SIOU are exclusive and separate from the high-speed serial I/O peripherals in the PL that include 100 Gb Ethernet (x4 CAUI-4), and PCIe Gen3 (up to x16). See Chapter 36, PL Peripherals for further information.

The high-speed serial I/O controllers connected to the GTRs are shown in Figure 1-1, page 14. The controller connections to the GTRs are shown in Figure 1-3, page 18.

Features

Functionality

- Memory mapped configuration, and control registers.
- PS-GTR transceiver registers are exclusively programmed through PCW.
- Independent PS-GTR protocol support per lane (programmable through PCW).
- D+/D- lane polarity inversion for flexible board integration.
- SSC support.
- Elastic buffer management.
- 8b10b support for USB3.0 and PCIe v2.0 only. For other protocols, 8b10b support is in the MAC IP.

Clocking

- Internal PLL per lane.
- Different reference clock inputs per lane, with the ability to share reference clocks between lanes (programmed through PCW).

Power

- PS-GTR requires two analog supplies: PS_MGTRAVTT (1.8V nominal value), and PS_MGTRAVCC (0.850V nominal value).

PCIe v2.0 PHY Protocol

- Gen 1 and Gen 2.
- Lane-to-lane deskew for multi-lane PCIe design.
- Beacon signaling.
- Supports integrated RX termination resistors.

USB3.0 PHY Protocol

- Integrated RX termination resisters.
- LFPS signaling.

DisplayPort 1.2a PHY Protocol (Transmitter only)

- Reduced bit rate (RBR), 1.62 Gb/s.
- High bit rate (HBR), 2.7 Gb/s.
- HBR2, 5.4 Gb/s.
- Supports integrated RX termination resistors.

Gigabit Ethernet PHY Interfaces

- SGMII.
- 1000BASE-SX.
- 1000BASE-LX.
- Supports integrated RX termination resistors.

SATA v3.1 PHY Protocol

- Generation 1, 1.5 Gb/s.
- Generation 2, 3.0 Gb/s.
- Generation 3, 6.0 Gb/s.
- Out-of-band (OOB) signaling.
- Supports integrated RX termination resistors.

Note: The protocols other than the ones listed are not supported.

Functional Description

Figure 29-1 provides a top-level overview of the PS-GTR block and its interface with other components. Figure 29-1 shows four PS-GTR transceivers that are shared among various controllers. For information on the DisplayPort, SATA, PCIe, Ethernet, and USB blocks, see:

Chapter 30, PCI Express Controller

Chapter 31, USB Controller

Chapter 32, SATA Controller

Chapter 33, DisplayPort Controller

Chapter 34, GEM Ethernet

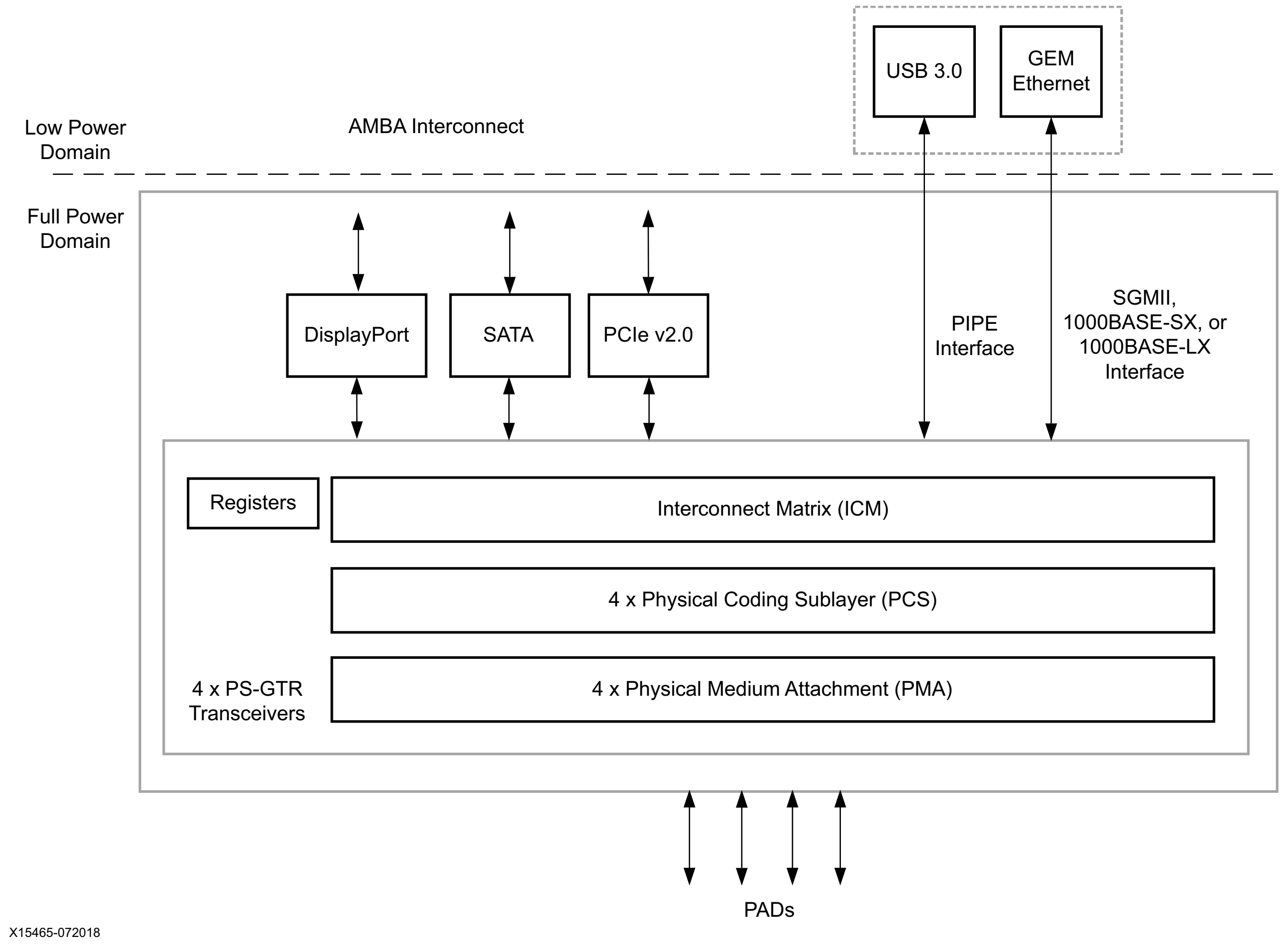


Figure 29-1: **SIOU Block**

Interconnect Matrix

The interconnect matrix (ICM) implements connectivity between the media access controllers (MACs) and the physical coding sublayer (PCS). The ICM is automatically programmed by the *Processing System Configuration Wizard* (PCW). Table 29-1 shows the connectivity implemented by the ICM between PS-GTR transceivers and the available MACs.

Table 29-1: **Interconnect Matrix**

Controller	PHY Lane 0	PHY Lane 1	PHY Lane 2	PHY Lane 3
PCIe v2.0	PCIe.0	PCIe.1	PCIe.2	PCIe.3
SATA	SATA.0	SATA.1	SATA.0	SATA.1
USB0 3.0	USB0	USB0	USB0	
USB1 3.0				USB1
DisplayPort	DP.1	DP.0	DP.1	DP.0
GEM0(1)	GEM0			
GEM1(1)		GEM1		
GEM2(1)			GEM2	
GEM3(1)				GEM3

Notes:
1. The GEM Ethernet interface to the GTRs includes SGMII, 1000BASE-SX, and 1000BASE-LX protocols.

PS-GTR transceivers can be broadly divided into the following blocks.

- Physical Coding Sublayer
- Reference Clock Network
- GTR reference clock PS_MGTREFCLKP/N should be stable before releasing the PS_POR_B, just like PS_REF_CLK.

Physical Coding Sublayer

The physical coding sublayer (PCS) consists of transmit and receive paths. It also includes common logic that is required for both receive and transmit paths, clock generator logic, and reset generator/synchronization logic.

Transmit Path

Figure 29-2 shows a block diagram of the transmit path.

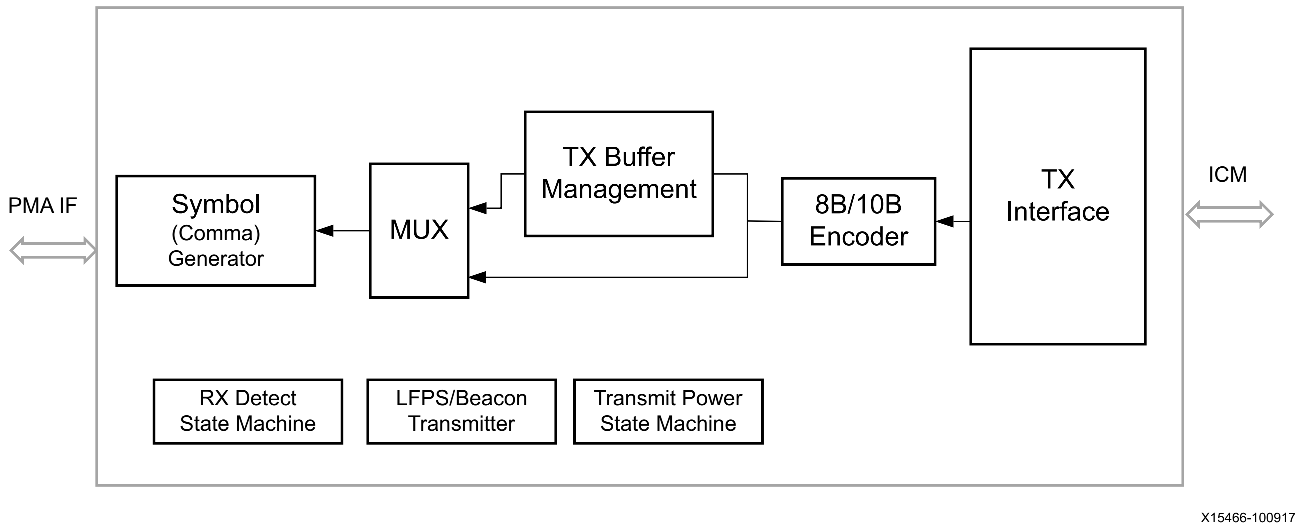


Figure 29-2: **PCS Transmit Path**

The PCS transmit block has the following features.

- Transmit buffer management.
- 8B/10B encoder for PCIe v2.0, DisplayPort, and USB3.0 only. Other protocols contain the encoder in the MAC IP.
- Transmit power state finite state machine (FSM).
- Low-frequency periodic signal (LFPS)/beacon transmitter (USB 3.0 and PCIe).
- Symbol (comma) generator for PCIe v2.0 and USB3.0 only. Other protocols contain the generator in their MAC IP.

Receive Path

Figure 29-3 shows a block diagram of the receive path.

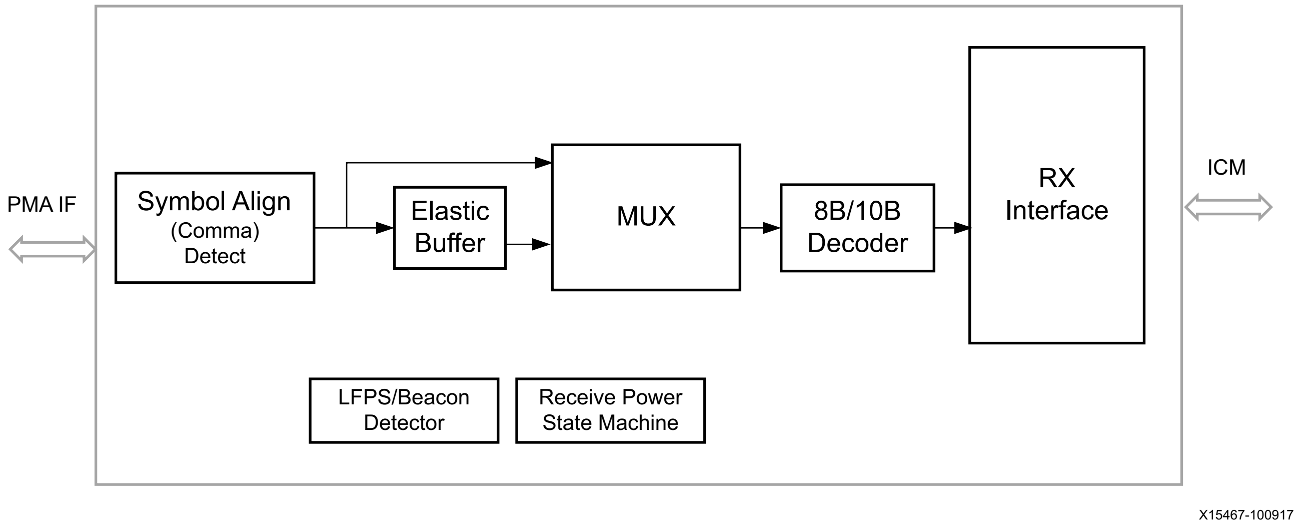


Figure 29-3: **PCS Receive Path**

The PCS receive block has the following features.

- Symbol (comma) alignment for USB3.0 and PCIe v2.0 only. For other protocols, symbol alignment happens in the MAC IP.
- Elasticity buffer management.
- 8B/10B decoder for USB3.0 and PCIe v2.0. Other protocols contain the decoder in the MAC IP.
- Receive power state FSM.
- LFPS/beacon detector (USB 3.0 and PCIe v2.0).

Reference Clock Network

The reference clock network architecture has four lanes (PS_MGTREFCLK0, PS_MGTREFCLK1, PS_MGTREFCLK2, and PS_MGTREFCLK3 also referred as Ref Clk [0, 1, 2, 3 in GUI) and supports multiple protocols at each lane independently. The reference clock frequencies required to support various protocols are listed in Table 29-2. Each lane can be programmed through the PCW under *Clock Configurations* to have its own reference clock, or share a reference clock from another lane. For more information regarding the reference clock frequencies, refer to the *Zynq UltraScale+ MPSoC Data Sheet: DC and AC Switching Characteristics* (DS925) [Ref 2].

Table 29-2: **Reference Clock per Protocol**

Protocol	Reference Clock Frequency (MHz)
PCIe v2.0 (multi-lane) Only the common clock architecture is supported.	100.0 MHz
SATA (multi-core)	125.0 MHz, 150.0 MHz
USB 3.0	26.0 MHz, 52.0 MHz, 100.0 MHz
DisplayPort (harmonic of 27.0 MHz)	27.0 MHz, 108.0 MHz, 135.0 MHz
GEM SGMII, 1000BASE-SX, or 1000BASE-LX	125.0 MHz

Note: GTR reference clock PS_MGTREFCLKP/N should be stable before releasing the PS_POR_B, just like PS_REF_CLK.

Physical Medium Attachment Sublayer

The physical medium attachment (PMA) sublayer is based on the following architecture.

- Programmable TX driver
- Clock data recovery (CDR)
- Serializer/deserializer (SerDes)
- I/O buffers

PLL Lock Status

Each of the four PS-GTR lanes contain its own PLL clock circuit. The input for each clock circuit is individually selected from one of several clock sources using the PCW. The PLL status can be read using the L{0:3}_PLL_STATUS_READ_1 [pll_lock_status_read] register bit 4. The PLL generates a wide number of frequencies. The required GTR clock frequencies for each protocol are listed in Table 29-2.

Note: The PLL lock status bit is valid only after GTR reset (GTR reset is performed by a controller reset). After the lock bit is set, it stays set until another reset occurs (i.e., during normal operation, if the PLL lock is lost, the lock status bit will not update).

PMA Transmitter

Figure 29-4 shows a block diagram of the PMA transmitter.

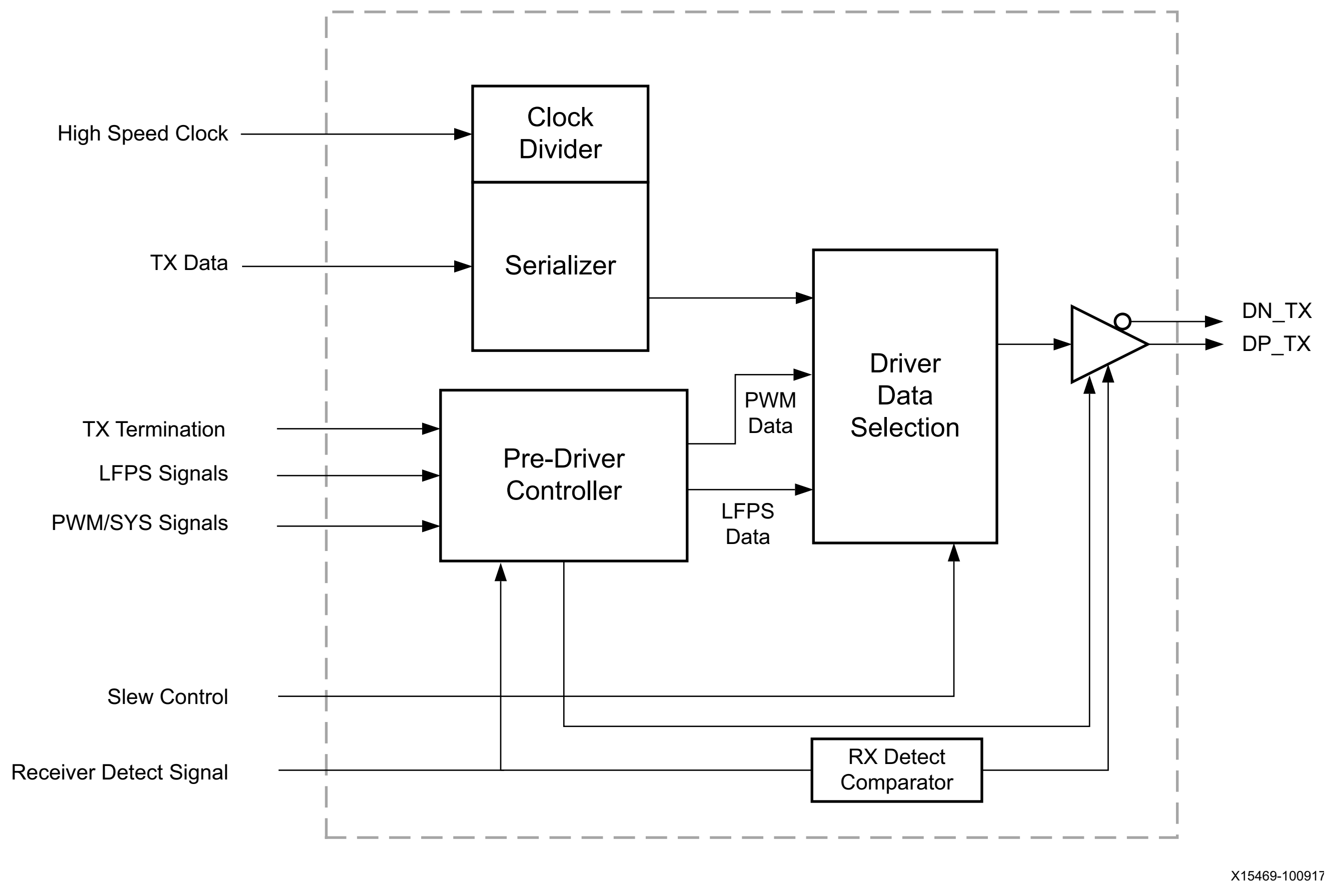


Figure 29-4: **PMA Transmitter Block Diagram**

Serializer and Clock Divider

The serializer and clock divider module are clocked with the high-speed half-rate clocks from the PLL through the PS-GTR transceiver clock and reset distribution block. The parallel data is loaded after the load signal is active. When not being loaded, the data is serially shifted out to the voltage mode driver. The clock divider block, implemented with the serializer, generates all the required clocks for serialization.

TX Polarity Control

When the TXP and TXN differential I/O signals are swapped on the PCB, the differential data transmitted by the PS-GTR transceiver TX is reversed. One solution is to invert the parallel data before serialization and transmission on the differential pair. The TX polarity control can be accessed using the SERDES.L{0:3}_TX_ANA_TM_13 [3:2] register bits. To enable TX polarity reversal, set these two bits High so TXP is negative and TXN is positive.

Data Selection Multiplexer, Predriver, and Voltage Mode Driver

The PMA transmitter uses a voltage mode driver supporting normal swing, low swing, and low-power low-swing mode. The predriver controller puts the driver in the correct states and selects the correct analog components. This module is tightly coupled with the function of the analog voltage mode driver and high-speed serial data selection multiplexer. The allowed states of the analog transmitters are as follows.

- Normal swing high-speed data transmission with and without allowed de-emphasis level for supported protocols.
- Low swing mode high-speed data transmission with and without allowed de-emphasis level for supported protocols.
- Electrical idle which holds the common mode at line.
- Putting the line in a high-Z state.
- LFPS transmission.
- Receiver detection state.

The data selection multiplexer selects data from various sources including serial data during LFPS. The low-power driver mode performs receiver detection through the predriver by selecting the appropriate data at the data selection multiplexer input and controlling the driver switches.

TX Configurable Driver

IMPORTANT: *This section outlines advanced features of the PS-GTR transceiver. Xilinx strongly advices using the default settings in the PCW.*

The PS-GTR transceiver TX driver is a high-speed voltage-mode differential output buffer. To maximize signal integrity, it includes the following features.

- Differential voltage control
- Transmit de-emphasis control

Each controller modifies the differential output voltage and de-emphasis to default values based on the protocol selected. Table 29-3 shows these default values on a per protocol basis.

Note: Xilinx recommends using the default TX differential output voltages and de-emphasis settings.

Table 29-3: **Default Differential Output Voltage and De-Emphasis by Protocol**

Protocol	Default Differential Output Voltage (Lx_TX_ANA_TM_16)	Default De-Emphasis (Lx_TX_ANA_TM_18)
USB3	850 mV	–3.5 dB
PCIe Gen. 1	850 mV	–3.5 dB
PCIe Gen. 2	850 mV	–3.5 dB (default) controller might modify this value to –6 dB during link-up.
GEM SGMII or 1000BASE-SX/LX	850 mV	0 dB
DisplayPort RBR, HBR, and HBR2	Controller modifies value during link-up.	Controller modifies value during link-up.
SATA Gen. 1, 2, and 3	425 mV	0 dB

Table 29-4 defines the TX configurable driver attributes.

Table 29-4: **TX Configurable Driver Attributes**

Register name	Address offset	Bit	Value and description
TX Swing			
L0_TX_ANA_TM_15 L1_TX_ANA_TM_15 L2_TX_ANA_TM_15 L3_TX_ANA_TM_15	Lane 0: `0x0003C` Lane 1: `0x0403C` Lane 2: `0x0803C` Lane 3: `0x0C03C`	6	Enable TX full/low swing setting. `1'b0`: Default set by the PCW. `1'b1`: TX swing defined by L0_TX_ANA_TM_15[7], L1_TX_ANA_TM_15[7], L2_TX_ANA_TM_15[7], or L3_TX_ANA_TM_15[7].
		7	`1'b0`: Full swing (>0.8V) – default value. `1'b1`: Low swing (>0.4)
TX Margin			
L0_TX_ANA_TM_16 L1_TX_ANA_TM_16 L2_TX_ANA_TM_16 L3_TX_ANA_TM_16	Lane 0: `0x00040` Lane 1: `0x04040` Lane 2: `0x08040` Lane 3: `0x0C040`	0	Enable TX driver swing control. `1'b0`: Default set by the PCW. `1'b1`: TX differential output defined by L0_TX_ANA_TM_16[3:1], L1_TX_ANA_TM_16[3:1], L2_TX_ANA_TM_16[3:1], or L3_TX_ANA_TM_16[3:1].
		3:1	If full swing: `000` → 0.85V – default value. `001` → 0.85V `010` → 0.6375V `011` → 0.53125V `100` → 0.425V `101` → 0.31875V `110` → 0.2656V `111` → 0.213V If low swing: `000` → 0.478V – default value. `001` → 0.478V `010` → 0.372V `011` → 0.2656V `100` → 0.159V `101` → 0.10625V `110` → 0.053V `111` → Reserved

Table 29-4: **TX Configurable Driver Attributes *(Cont'd)***

Register name	Address offset	Bit	Value and description
TX De-emphasis			
L0_TX_ANA_TM_18 L1_TX_ANA_TM_18 L2_TX_ANA_TM_18 L3_TX_ANA_TM_18	Lane 0: `0x00048` Lane 1: `0x04048` Lane 2: `0x08048` Lane 3: `0x0C048`	7:0	`8'b0000_0000` → –6.0 dB de-emphasis `8'b0000_0001` → –3.5 dB de-emphasis `8'b0000_0010` → –0.0 dB de-emphasis – default value. Other settings not supported `1'b0`: Default set by the PCW. `1'b1`: TX de-emphasis value set by L0_TX_ANA_TM[7:0], L1_TX_ANA_TM[7:0], L2_TX_ANA_TM[7:0], or L3_TX_ANA_TM[7:0].
L0_TX_ANA_TM_118 L1_TX_ANA_TM_118 L2_TX_ANA_TM_118 L3_TX_ANA_TM_118	Lane 0: `0x001D8` Lane 1: `0x041D8` Lane 2: `0x081D8` Lane 3: `0xC1D8`	0	Force TX swing de-emphasis

Electrical Idle

There are two different kinds of electrical idle: one during sub-low power mode, and another during high-speed low-latency mode. During high-speed mode, the predriver controller controls the driver to short the lines (TXP and TXN) where they instantly change to the common-mode voltage specified by the protocols. In sub-low power mode, a low-power circuit holds the line. The power-consuming driver is in shut-off mode during the electrical idle.

Spread-Spectrum Clocking Transmitter Support

By default, spread-spectrum clocking (SSC) is generated by the PS-GTR transmitter for USB 3.0, SATA Gen1, SATA Gen2, SATA Gen 3, and DisplayPort. SSC settings are done by the PCW. For PCIe, SSC generation in the transmitter is turned off because the protocol specification requires SSC to be supported by the reference clock (PS_MGTREFCLK).

PMA Receiver

The PMA receiver consists of the PMA receiver analog implementation and receiver control module. The RX analog PMA interfaces with the PCS through the control module. The control module also provides power state information and isolation in certain cases. Figure 29-5 shows a block diagram of the PMA receiver.

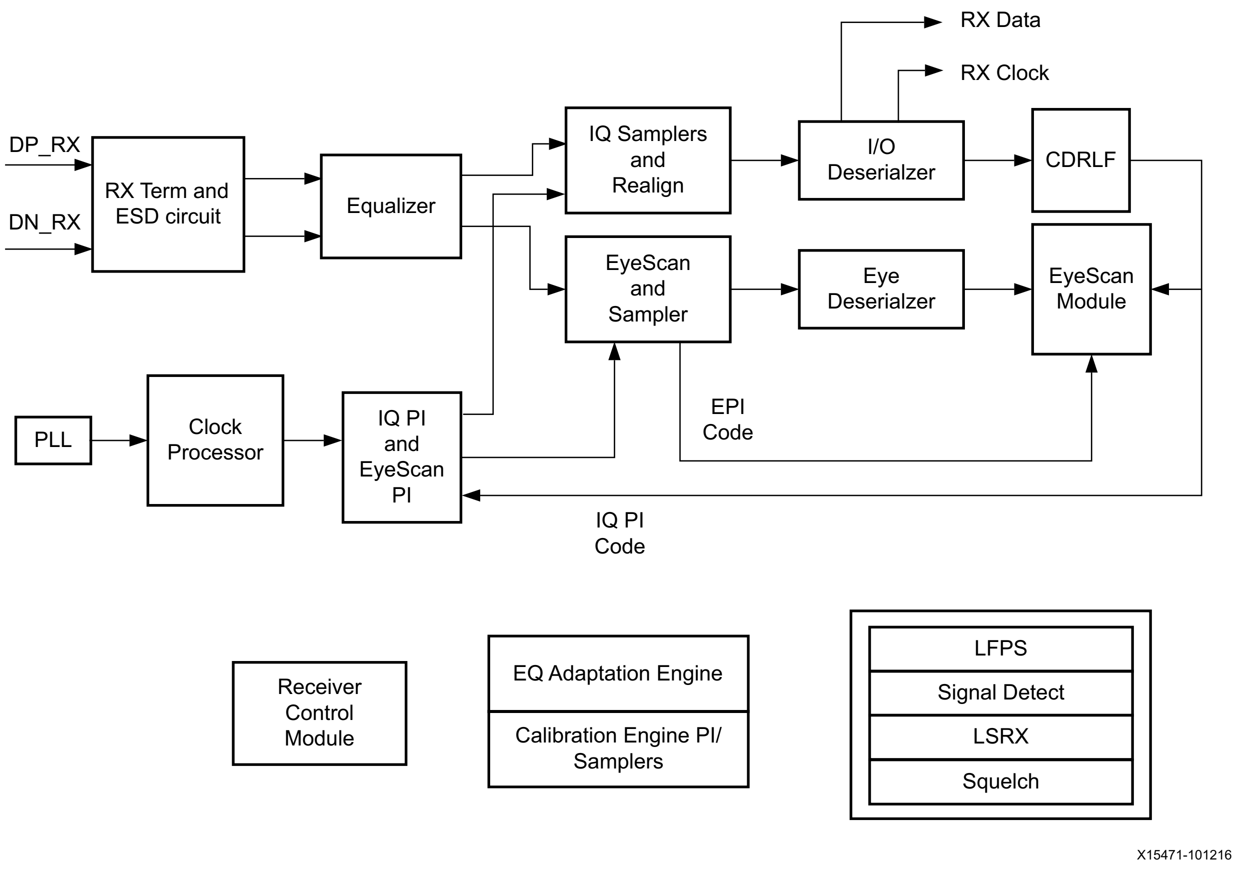


Figure 29-5: **PMA Receiver Block Diagram**

> ***Note:*** GTR calibration is done by the processor configuration wizard settings in Vivado software. From 2020.1 software version this calibration code is placed in the psu_init source code. See Answer Record 72992 for more information.

Receiver Equalizer

Equalization is implemented as a continuous time linear equalizer (CTLE).

Spread-Spectrum Clocking Receiver

The PS-GTR receiver supports 5000 ppm down-spread, spread-spectrum clocking (SSC) modulated at rate of 30-33 KHz.

Sampler and Realign

There are four samplers that operate on four phases of a half-rate clock. Because the four phases are quadrature phases, they are collectively called the IQ path. The sampling clock is the recovered clock that is the output of the IQ-phase interpolator (PI). The samplers operate on a full CMOS-level clock, sample the low-swing data received from the equalizer, and output CMOS-level data. Local current-mode logic (CML) to CMOS converters convert the recovered clock phases coming from the PI in CML levels to CMOS levels for sampling.

The outputs of the four samplers are finally realigned to one phase of the recovered clock, inside the realign block, before being sent to the digital loop filter. The on-chip EyeScan, that measures the horizontal eye opening known as an E-sampler, operates at half the rate of clocks coming from the EyeScan PI. For the vertical EyeScan (opening), the EyeScan digital-to-analog converter (DAC) is used to control the sampling point in E-samplers in the Y-direction. Sampler offsets are independently calibrated out for these samplers using an offset calibration scheme.

Clock Processor

A full-rate CMOS-level clock is distributed from the PLL module to the receiver. The PLL clock goes into the clock processor module. This module divides down the PLL full-rate clock as per the programmed division factor and provides two differential phases of this divided clock as output. Because the receiver front end operates at half rate, the division factor is always programmed to give two half-rate clocks. Thus, for a division factor of two, the outputs are 0° and 180° phases of a half-rate CMOS clock.

Phase Interpolator

The PI receives the 0° and 180° phases of the half-rate CMOS-level clocks from the clock processor. The PI provides four quadrature phases of the CML-level half-rate clock that are phase shifted as compared to the input clocks as dictated by the PI code coming from the clock and data recovery loop filter (CDRLF). The PI can shift the recovered clocks with a resolution of UI/32. The samplers use these clocks to sample the receive data over a span of two UI. Table 29-5 describes these clocks. Each clock is 90° out of phase with the next clock.

Table 29-5: **CMOS-level Clocks**

Clock	Clock Usage
I clock	Samples the data in the middle of the first data eye.
Q clock	Samples the data at the edge between the first and second data eyes.
I-bar clock	Samples the data in the middle of the second data eye.
Q-bar clock	Samples the data at the edge at the end of the second data eye.

The CDRLF uses the feedback path to control the PI where the phase of the clocks are lined up where they are expected to be, relative to the incoming serial data stream. Figure 29-6 shows the recovered clock relationship to the incoming data after CDR lock.

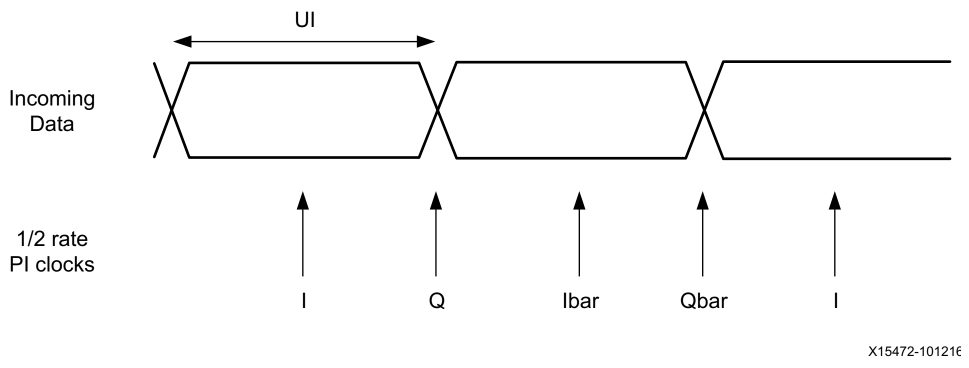


Figure 29-6: **PI Clock Relationship to Data Post CDR Lock**

For on-chip EyeScan, a replica PI takes in the PI codes from the EyeScan module. The EyeScan PI codes are offset by a particular value from the main IQ PI codes to do a horizontal EyeScan. Thus, the recovered EyeScan clocks are phase shifted from the main recovered IQ clocks by an offset defined by the EyeScan module. This, in combination with the EyeScan samplers, enables 2D EyeScan.

RX Polarity Control

If the RXP and RXN differential I/O signals are swapped on the PCB, then the differential data transmitted by the PS-GTR transceiver TX are reversed. The PS-GTR receiver allows inversion to be done on parallel bytes in the PCS after the SIPO to offset reversed polarity on the differential pair. RX polarity control can be accessed using the SERDES.L{0:3}_TM_MISC1 [7] register bit. To enable RX polarity reversal, set bit 7 High so RXP is negative and RXN is positive.

CDRLF, Deserializer, and PI Controller

The digital CDR loop filter takes in the main recovered clock and the four data samples from the IQ samplers and generates the IQ PI codes as output. This loop filter, in combination with the samplers and PI, forms a proportional and integral negative feedback loop to align the I phase of the recovered clock to the center point of the received data bits. The deserializer converts the half-rate data from the samplers to 10-bit symbol data, which is then given to the receiver control module for further processing before it is passed to the PCS.

The PI controller converts the output PI codes from the CDRLF to thermometric format as accepted by the analog PI. The CDRLF, deserializer, and PI controller constitute a digital module that is synthesized and implemented using the digital flow tools. This block runs at a clock speed of 3 GHz.

EyeScan Module

The EyeScan module implements a function that moves the sample point for data anywhere in the data eye and measures the number of bit errors at that point. To accomplish this, the EyeScan module uses three other digital modules in addition to an analog phase interpolator and a deserializer. These digital modules are the CDRLF and PI controller, and the eye plot PI controller.

The eye plot PI controller controls the eye-phase interpolator (E PI), under the direction of the EyeScan module. The E PI tracks to the location of the primary PI when no additional ups and downs are requested by the EyeScan module. The EyeScan module can request additional ups and downs be performed on the E PI to create a fixed offset between the E PI and primary PI. The deserializer uses the E PI to sample receive eye data (E data) at times that are at different phases to the normal sampled I and Q data. An up request causes the clock phase delay to increase (moving the clock phase later in time), and a down request causes the clock phase delay to decrease (moving the clock phase earlier in time). The EyeScan module uses this system to measure the bit-error rate at any point in the received data eye.

Note: For more information on EyeScan Module, see Xilinx Answer 67295.

Sideband Receive Path

The blocks covered in this section are supporting blocks as part of various protocols or test features. They are not part of the regular datapath.

Signal Detect

The signal-detect block is used to detect an exit from electrical idle in the PCIe protocol. In the SATA protocol, the signal detect block is used to detect out of band (OOB) signaling. This block compares the magnitude of differential signals on the PS_MGTRRX pins with a specified reference and provides an output. This output is asserted in the presence of a differential signal greater than 175 mV_{PPD} (PCIe) or 200 mV_{PPD} (SATA) and deasserted in the presence of any signal less than 65 mV_{PPD} (PCIe) or 75 mV_{PPD} (SATA). The reference of comparison is programmable. The output is asserted during the reception of valid high-speed data but it can glitch because of high-frequency components in the data. A programmable digital filter is implemented to filter out such unwanted glitches.

LFPS Detect

The LFPS block is used to detect low frequency periodic signaling (LFPS) in the USB 3.0 protocol. It compares the magnitude of differential signals on the PS_MGTRRX pins with a specified reference and provides an output to the PCS. This output is asserted in the presence of an LFPS signal greater than 300 mV_{PPD} and deasserted in the presence of any signal less than 100 mV_{PPD}. The reference of comparison is programmable from 100 to 200 mV_{PPD}. The output is deasserted during reception of valid super-speed (SS) data but it can glitch because of high-frequency components in the data. A programmable digital filter is implemented in PCS to filter out such unwanted glitches.

Register Overview

This section lists the applicable PS-GTR transceiver interface registers.

PS-GTR Registers

The PS-GTR registers are listed in the *Zynq UltraScale+ MPSoC Register Reference* (UG1087) [Ref 4] and programmed by the Vivado Processor Configuration Wizard (PCW).

Configuration Program

The PS-GTR transceiver is exclusively programmed using the processor configuration wizard (PCW).

Under PCW, the user can program the reference clocking scheme for each PS-GTR lane, as well as the protocols used per lane. All other behavior of the PS-GTR transceiver is exclusively handled by the respective MAC IP.

PCI Express Controller

Introduction

The Zynq® UltraScale+™ MPSoC provides a controller for the integrated block for PCI Express® v2.1 compliant, AXI-PCIe bridge, and DMA modules. The AXI-PCIe bridge provides high-performance bridging between PCIe and A/XI. The block diagram of the controller for PCIe is shown in Figure 30-1.

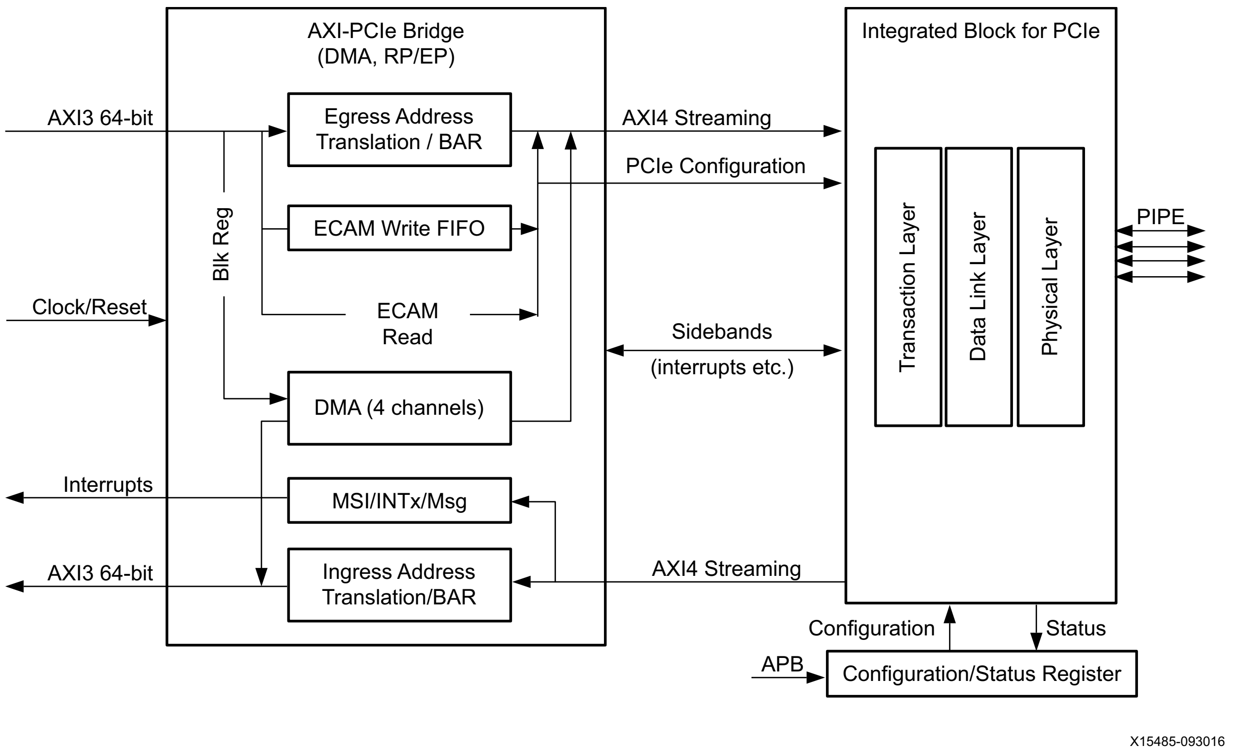


Figure 30-1: **Block Diagram of the Controller for PCIe**

The controller for PCIe supports both Endpoint and Root Port modes of operations. As shown in Figure 30-1, the controller comprises two sub-modules.

- The AXI-PCIe bridge provides AXI to PCIe protocol translation and vice-versa, ingress/egress address translation, DMA, and Root Port/Endpoint (RP/EP) mode specific services.
- The integrated block for PCIe interfaces to the AXI-PCIe bridge on one side and the PS-GTR transceivers on the other. It performs link negotiation, error detection and recovery, and many other PCIe protocol specific functions. This block cannot be directly accessed.

Features

This section provides a summary of features supported by the controller for PCIe.

- Endpoint or Root Port mode of operation
- Support for Gen1 (2.5 GT/s) or Gen2 (5.0 GT/s) link rates.
- Support for single x1, x2, or x4 link.
- Endpoint mode supports MSI-X interrupts in addition to MSI and legacy.
- Support for advanced error reporting capability.
- AXI-PCIe bridge supports:
  - 64-bit AXI3 compliant AXI master and AXI slave interfaces operating at a 250 MHz clock.
  - MSI-X table and PBA implementation at predefined location for Endpoint mode.
  - Eight fully-configurable address translation apertures in each direction (egress—AXI to PCIe and ingress—PCIe to AXI).
  - Generation of configuration transactions through the enhanced configuration access mechanism (ECAM) and messages by the AXI CPU in Root Port mode.
  - Receive interrupt controller aggregates and presents legacy and MSI interrupts from PCIe to the AXI CPU in Root Port mode.
- Receive PCIe message FIFO for Root Port.
- Four-channel fully-configurable DMA engine.
  - Each DMA channel controllable from PCIe CPU, AXI CPU, or both.
  - Separate source and destination scatter-gather queues with the option to have separate status scatter-gather queues.

Note: I/O space is not supported. Link rate change in root mode is not supported.

Functional Description

This section provides an overview of the clock and reset scheme for the controller for PCIe followed by a functional description of the integrated block for PCI Express and the AXI-PCIe bridge with the DMA controller.

Clock Scheme

The controller for PCIe operates in multiple clock domains. Figure 30-2 shows the clock domains. The pipe_clk and user_clk are derived from the PS-GTR transceiver interface provided 250 MHz clock.

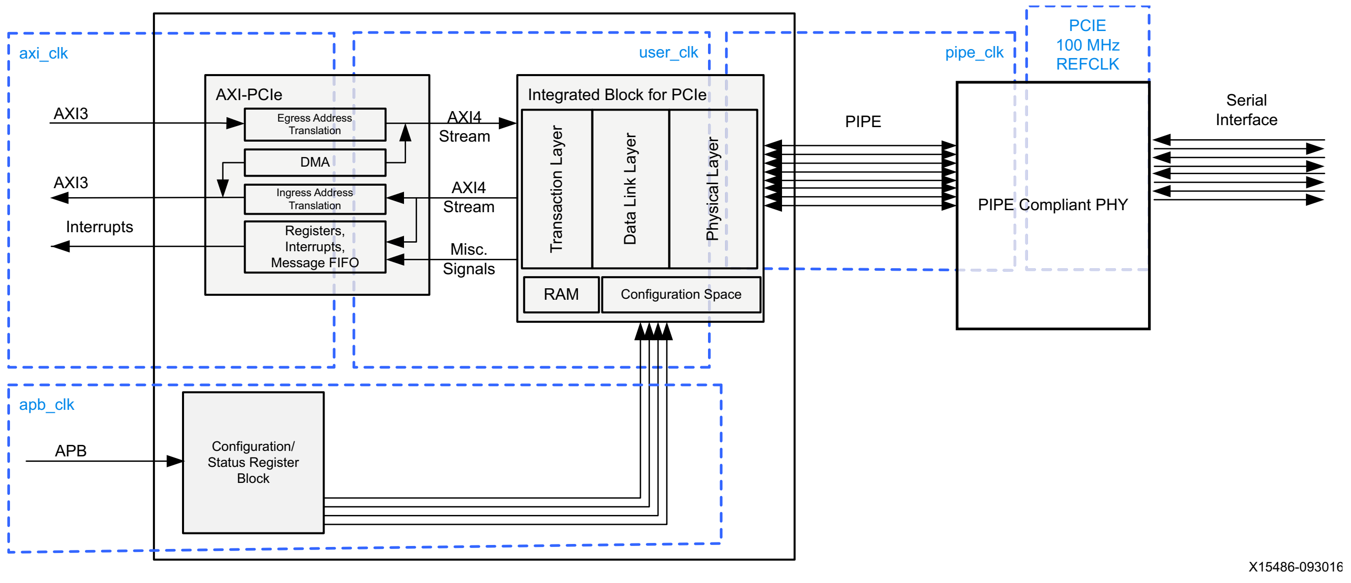


Figure 30-2: **Controller for PCIe Clock Domains**

Table 30-1 provides a description of the clocks.

Table 30-1: **Clock Description**

Clock	Description
PCIe 100 MHz reference clock	The PCIe protocol specifies a 100 MHz clock with spread spectrum. This clock is used as a reference clock to the PS-GTR transceiver interface, which is part of the PHY. The PS-GTR transceiver interface generates a 250 MHz clock from this reference clock for the parallel datapath. This clock comes from external interface (typically on-board clock source in the case of Root Port mode, and sourced by a host system via the PCIe slot in the case of Endpoint mode). Only common clock mode is supported for reference clock.
pipe_clk	The PS-GTR transceiver interface provided clock is converted to a 125 MHz clock for a Gen1 link and a 250 MHz clock for a Gen2 link.
user_clk	This is a 250 MHz clock derived from a PS-GTR transceiver interface provided clock. The link and transaction layers of the integrated block for PCIe operate in this clock domain. The AXI-PCIe bridge interfaces to the integrated block for PCIe in the user_clk domain.
apb_clk	The APB interface and its associated register block and supporting logic runs in the apb_clk domain. This clock domain is independent of all other domains in terms of frequency and phase and is derived from the PLL on the PS.
axi_clk	The AXI interfaces of the AXI-PCIe bridge run in the axi_clk domain. This domain is independent of all other domains in terms of frequency and phase. To keep up with the x4 Gen2 throughput requirements, this clock needs to be at least 250 MHz. This clock is derived from the PLL on the PS and can be programmed to a lower frequency for lower performance PCIe configurations through the CRF_APB.PCIE_REF_CNTRL registers. For non-x4 Gen2 configurations, the 125 MHz is sufficient to achieve the best performance.

Reset Scheme

The reset scheme of the controller for PCIe is shown in Figure 30-3.

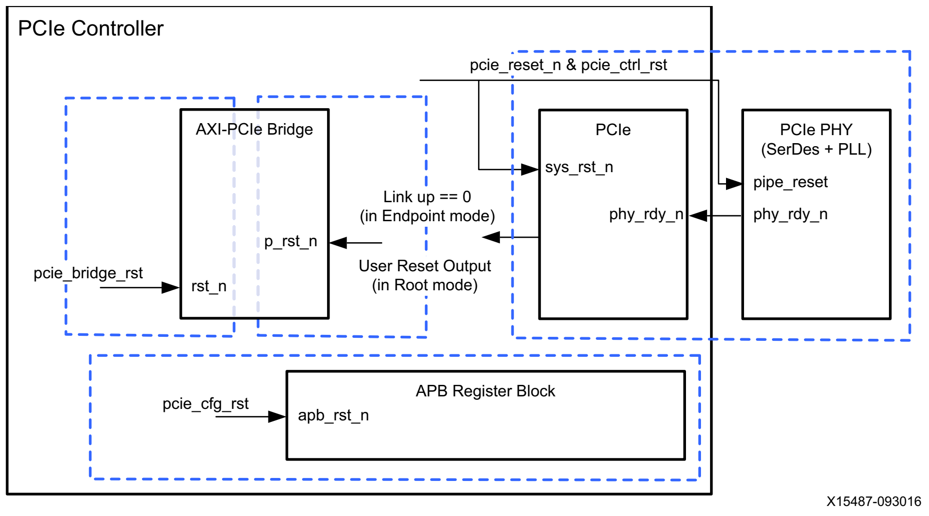


Figure 30-3: **Controller for PCIe Reset Scheme**

Table 30-2 provides a description of the resets.

Table 30-2: **Reset Description**

Reset	Description
pcie_reset_n	This is the PCIe protocol reset. In Endpoint mode, this reset is controlled by the host device, and the Endpoint designated MIO pin can be used as an input for this reset. In Root Port mode, this reset is controlled by the software outside the PCIe block, and the MIO pin can be configured as an output to drive the reset. When the MIO pin is not allocated to the PCIe, this signal is driven High to allow the PCIe block to come out of reset under local software control (pcie_ctrl_rst_n).
pcie_cfg_rst_n	This resets the register block that holds the attribute configuration of the controller for PCIe.
pcie_ctrl_rst_n	The reset pcie_reset_n is controlled by the host. It is possible that this reset is released before the configuration of the PCIe core is completed, thereby causing the controller for PCIe to come out of reset prematurely. This reset allows the software to override the externally controlled pcie_reset_n. Software is required to release this reset only after the integrated block for PCIe attribute programming and the PS-GTR transceiver interface programming is complete.
pcie_bridge_rst_n	The AXI interfaces of the AXI-PCIe bridge have a separate clock and reset domain. The reset pcie_bridge_rst_n controls this domain. This reset can be released once the AXI clock domain is stable. This domain does not reset due to a link down to allow the AXI domain (APU or RPU) to (if needed) access the bridge configuration registers.

TIP: *The reset to AXI-PCIe bridge is determined by the mode of operation i.e., whether Root Port or Endpoint as shown in Figure 30-3.*

Integrated Block for PCI Express

The integrated block for PCIe complies with the *PCI Express base specification, rev. 2.1* and consists of the physical, data link, and transaction layers. The protocol uses packets to exchange information between layers. Packets are formed in the transaction and data link layers to carry information from the transmitting component to the receiving component. Information is added to the transmitted packet that is required to handle the packet at specific layers.

The functions of the protocol layers include the following.

- Generating and processing of TLPs.
- Flow-control management.
- Initialization and power management functions.
- Data protection.
- Error checking and retry functions.
- Physical link interface initialization.
- Maintenance and status tracking.

The integrated block for PCI Express supports up to 4-lane 2.5 GT/s and 5.0 GT/s PCI Express Endpoint and Root Port configurations.

The integrated block for PCIe provides PIPE interface for connection to the gigabit transceivers. The PIPE interface runs at 250 MHz in Gen2 (5 Gb/s, per lane, per direction) mode or 125 MHz in Gen1 (2.5 Gb/s, per lane, per direction) mode.

The PS-GTR transceivers in the processing system (PS) are used for serialization/deserialization (SerDes) purposes. The high-speed transceivers are used through the multiplexer switch and are shared with other blocks (such as DisplayPort, SATA, USB, and GEM) in the PS.

Further details on transceivers are available in Chapter 29, PS-GTR Transceivers.

IMPORTANT: *The AXI streaming and sideband signals between the AXI-PCIe bridge and the integrated block for PCI Express are not directly accessible. Every PCIe transfer initiated in the AXI domain passes through the AXI-PCIe bridge.*

PCI Express uses a credit-based flow control mechanism. The integrated block for PCIe can be programmed to advertise credit information based on the buffering used. This is set by default to optimal values (based on the RAMs used in the implementation). The values are available in the PCIE_ATTRIB register space for various flow control credit options.

IMPORTANT: *In both Endpoint and Root Port modes, the integrated block for PCIe advertises infinite completions; finite completions are not supported.*

IMPORTANT: *When integrated block for PCIe is enabled in root port mode, enabling coherency features through CC-400 using AxACACHE overrides in AXI-PCIe Bridge registers and enabling SMMU are not supported.*

Configuration Control (APB Interface)

The attributes for the integrated block for PCIe (Endpoint or Root Port mode) are configured through the programmable configuration and status registers (CSR) accessible through the APB interface. APB interface uses the apb_clk, which is asynchronous to the other clocks. It is a 32-bit wide address and 32-bit wide data bus interface.

The integrated block for PCIe attributes are used to set up the mode of operation (Root Port or Endpoint), the list of capabilities and address pointers and so on. A detailed list of these attributes is available in the PCIE_ATTRIB register set in the *Zynq UltraScale+ MPSoC Register Reference* (UG1087) [Ref 4].

Power Management

The PCIe protocol specification indicates four low-power states: L0s, L1, L2, and L3, with L0s having the least recovery latency (shallow-power state) and L3 having the maximum recovery latency as it involves possible power supply turn-off (deep-power state). L0 is the normal working link state. In addition to the L0 state, the integrated block for PCIe supports the L1 (low power) state. Entry into L1 from L0 needs to be initiated by the software.

Note: The integrated block for PCIe does not support active state power management (ASPM). ASPM L0s support is optional per the *PCI-SIG ASPM Optionality ECN* [Ref 55].

Programmed Power Management

To achieve considerable power savings on the PCI Express hierarchy tree, the core supports these link states of the programmed power management (PPM).

- L0: Active state, data exchange state.
- L1: Higher latency, lower power standby state.

The PPM protocol is initiated by the downstream component/upstream port.

- PPM L0 state

  The L0 state represents normal operation and is transparent to your logic. The core reaches the L0 state after a successful initialization and training of the PCI Express link as per the protocol.

- PPM L1 state

  The following steps outline the transition of the core to the PPM L1 state.

  a. The transition to a lower power PPM L1 state is always initiated by an upstream device by programming the PCI Express device power state to non-D0 (in the PM capability in configuration register space). The current device power state can be read through APB registers.

  b. The integrated block for PCIe stops accepting any further transactions. Any pending transactions are accepted fully and completed later.

  c. The integrated block for PCIe exchanges appropriate power management data link layer packets (DLLPs) with its link partner to successfully transition the link to a lower power PPM L1 state.

  d. All transactions are stalled for the duration of time when the device power state is non-D0.

TIP: *After identifying the device power state as non-D0, the software logic can initiate a request through the cfg_pm_wake to the upstream link partner to configure the device back to the D0 power state. If the upstream link partner has not configured the device to allow the generation of PM_PME messages (cfg_pmcsr_pme_en = 0), the assertion of cfg_pm_wake is ignored by the core. See the Zynq UltraScale+ MPSoC Register Reference (UG1087) [Ref 4].*

AXI-PCI Express Bridge

The AXI-PCIe bridge is a protocol converter between the AXI3 and PCIe domains and also provides optional DMA capability.

When a remote master issues a transaction over PCIe link, it appears as an AXI transaction on the AXI master port. When a local master (in the PS) issues an AXI transaction on the slave port, it goes onto the PCIe link based on address translation apertures set as either memory or configuration TLP.

The bridge supports non-contiguous and zero-byte enable in compliance with the PCIe specification. For the AXI master the following is true.

- On AXI, a 1DW or 2DW write from the PCIe domain received with a non-contiguous byte enable is completed as a series of 1 byte writes only for the enabled bytes.
- PCIe zero-byte writes are propagated over AXI as writes with no byte enables asserted.
- PCIe reads with non-contiguous byte enables are converted to AXI reads reading all bytes. Disabled bytes are read and data is provided as part of the completion data. AXI memory, which can be the target of such non-contiguous reads, should be prefetchable and should not have any read side-effects.
- PCIe zero-byte reads are issued as a single byte read in the AXI domain and provided as completion data. This provides the desired write-flushing mechanism.
- Writes use the same AXI ID (m_awid, m_wid, m_bid) for all write transactions regardless of the source; hence, these should be completed in order on AXI.
- Read and write response timeouts are configurable through registers in the bridge.
    - Reads initiated by the master AXI that do not complete within the specified timeout period are assumed to never complete and result in a completer abort response on the PCIe link.
    - Writes initiated by the master AXI that do not complete within the specified timeout period are assumed to never complete and are terminated.

For the AXI slave the following is true.

- An AXI write with non-contiguous byte enable is completed with as many contiguous byte-enable write transactions as necessary to write all enabled bytes in the PCIe domain and then is provided with an aggregated write response.

- Transactions that cannot be forwarded to the PCIe due to PCIe specific reasons (such as the PCIe data link layer is down, PCIe domain is in reset, or when bus master enable = 0 for Endpoint applications) are dropped and completed on AXI with a DECERR status.
- AXI slave interface initiated reads, I/O writes, and configuration writes that fail to complete after a timeout duration are assumed to never complete and are terminated with a SLVERR response to the AXI. When the AXI clock is 250 MHz, the duration of the timeout is 50 ms. The timeout has a linear relation with the AXI clock, for example, the timeout is 100 ms if the AXI clock is 125 MHz. When the integrated block for PCIe completion timeout disable attribute is set to one, the timeout is disabled.

Accessing Bridge Internal Registers

Internal bridge registers are accessed through the AXI slave using a bridge register translation. Various registers like the DMA registers, MSI-X table, and pending-bit array are accessed through their respective translations. The various translation registers are listed in the *Zynq UltraScale+ MPSoC Register Reference* (UG1087) [Ref 4].

The MSI-X table and PBA are applicable only for Endpoint mode of operation and the corresponding registers are implemented in the AXI-PCIe bridge at predefined offsets.

The bridge register translation on exit from reset is configured to accept all AXI transactions as bridge register access. As part of the Zynq UltraScale+ MPSoC initialization, one of the actions should be to reconfigure the bridge register translation into a specific (small) window to enable other address translations like DMA registers or ECAM. Refer to the Bridge Initialization section of the programming model for further details.

IMPORTANT: *The recommended address values, as defined in the Zynq UltraScale+ MPSoC Register Reference (UG1087) [Ref 4], should be used for various apertures. All access to the bridge registers should be one DWORD (32 bits) from the AXI domain.*

Figure 30-4 shows the AXI and PCIe domain access of various registers in the AXI-PCIe bridge.

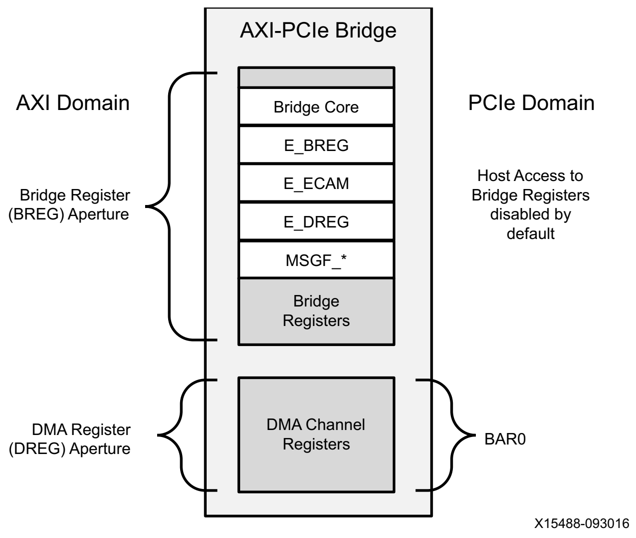


Figure 30-4: **AXI-PCIe Bridge Register**

AXI Domain

- Bridge registers are accessed through the AXI slave bridge register translation.
- DMA channel registers are accessed through the AXI slave DMA register translation.

Integrated Block for PCIe Domain

Bridge and DMA registers are accessed over the integrated block for PCIe at fixed offsets in the PCIe BAR region. The cfg_dma_reg_bar is zero (by default) making all BAR0 transactions consumable by the bridge.

- PCIe access to bridge registers is disabled (by default) (cfg_disable_pcie_bridge_reg_access).
- PCIe access to DMA channel registers is controlled through cfg_disable_pcie_dma_reg_access. This access is enabled by default.

The registers in the bridge are prefetchable. The BAR targeting these registers (BAR0 by default) can be marked prefetchable.

Address Translation

The bridge provides eight fully-configurable address apertures to support address translation both for ingress (from PCIe to AXI) and egress (from AXI to PCIe) transactions.

- In an AXI master, up to eight ingress translation regions can be set up. Translation is done for the PCIe TLPs that are not decoded as MSI or MSI-X interrupts or internal DMA transactions.
- In an AXI slave, up to eight translation regions can be set up. Translation is done for AXI transactions destined for PCIe and not PCIe ECAM or any other internal bridge register access.

IMPORTANT: *For egress translations, it is important to limit the AXI domain address to the following ranges per the System Address Map in Chapter 10.*

- 256 MB region starting at `0xE000_0000`.
- 8 GB region starting at `0x6_0000_0000`.
- 256 GB region starting at `0x80_0000_0000`.

Only when AXI transactions target these ranges are they routed to the controller for PCIe for further translation by the bridge.

In the following discussions, the term *tran* refers to ingress/egress translation. For example, tran_size refers to translation size and a tran_src_base refers to ingress/egress_src_base.

A translation is hit when the following occurs.

- Translation is enabled (tran_enable == 1).
- The tran_src_base[63:(12+tran_size)] == source address [63:(12+tran_size)].

On a hit, the upper source address bits are replaced with destination base address bits before forwarding the transaction to the destination.

Destination address = {tran_dst_base[63:(12+tran_size)] source address[12+tran_size]}.

If a translation is marked invalid (tran_invalid == 1), the transaction is not forwarded to destination and is handled as error.

- For egress, DECERR response is returned on AXI.
- For ingress, it is handled as an unsupported request on the PCIe.

If translation is valid (tran_invalid==0) and security_enable==1 then the following occurs.

- For ingress, ARPROT/AWPROT on AXI is assigned value from tz_at_ingr[i] associated with the translation.

- For egress, if ARPROT/AWPROT from AXI matches the security level of tz_at_egr[i] associated with the translation then transaction is forwarded to PCIe. Otherwise, it is discarded with SLVERR response on AXI.

IMPORTANT: *The security values for translation (tz_at_ingr/egr) are programmed at boot time as part of the SLCR_PCIE register under the FPD_SLCR_SECURE register set in the Zynq UltraScale+ MPSoC Register Reference (UG1087) [Ref 4].*

The following sequence provides an example for ingress address translation.

1. Consider host assigns PCIe BAR2 = `0xFFA0_0000`; 1MB size.
2. Ingress source base = `0xFFA0_0000`; destination base = `0x44A0_0000`; aperture size = 64 KB
3. Incoming PCIe memory transaction hitting BAR2 at `0xFFA0_xyzw` translates to address `0x44A0_xyzw` on AXI master port.

Note: The source/destination address programmed should be aligned to the translation aperture size. For a 64 KB aperture size, the lower 16 bits of the source/destination addresses must be zeros.

If multiple translation hits occur, the translation with the lowest index (lowest translation register address offset for the ingress/egress direction) is used for the transaction.

When operating as an Endpoint, the PCIe BARs are setup by the host PC during enumeration and ingress translations required for PCIe to AXI translations are set up by the AXI CPU.

IMPORTANT: *The bridge registers are accessible only through the AXI interface and not over PCIe by default. Host CPU access to bridge registers is enabled by writing to the bridge register (cfg_disable_pcie_bridge_reg_access bit in the AXI_PCIE_MAIN.cfg_pcie_rx0 register) through AXI.*

When a transaction fails to hit all translations, the subtractive decode (if enabled) and the transaction is forwarded without translation. This is controlled by the AXIPCIE_MAIN.I_ISUB_CONTROL register for ingress translations and the AXIPCIE_MAIN.E_ESUB_CONTROL register for egress translations.

Enhanced Configuration Access Mechanism

The bridge implements ECAM to translate AXI read or write transactions to PCIe configuration read or write TLPs. ECAM maps a portion of the AXI memory address space to the PCI Express configuration transactions. A write transaction targeting this region is converted into a PCI Express configuration write transaction and a read transaction targeting this region is converted into a PCI Express configuration read transaction.

The ECAM region is hit when the following occurs.

- ECAM is enabled (ecam_enable == 1).
- The ecam_base[63:(12+ecam_size)] == AXI address[63:(12+ecam_size)].

On a hit, the lower AXI address bits are mapped into the PCI Express configuration transaction as listed in Table 30-3.

Table 30-3: **AXI Address to PCIe Configuration TLP Mapping**

AXI Address Bits[1]	PCIe Configuration TLP Field	Notes
AXI address [27:20]	Bus number [7:0].	If ecam_size is set less than 256 MB, then the upper bus number bits that are not controlled by the AXI address are set to 0.
AXI address [19:12]	AXI address[19:15] = Device Number[4:0].	For PCI Express devices, implementing an alternative routing ID (ARI). AXI Address[19:12] = Function Number[7:0].
AXI address [14:12]	Function number [2:0].	
AXI address [11:2]	Configuration register DWORD address [11:2].	

Notes:

1. AXI address[1:0] along with AXI transaction size are used to compute the transaction byte enables.

ECAM transactions are not permitted to cross a DWORD address boundary. If a transaction hit to the ECAM region crosses a DWORD address boundary or times out, the transaction is aborted with SLVERR.

Note: The bridge generates SLVERR for ECAM transactions when the link is down. Software is required to check for link up status before sending ECAM transactions. The exception to this is during access of the local root configuration space (bus number = 0) when the PCIe controller is used as the Root Port.

Generation of Type-0 or Type-1 Configuration Transactions

Type-0 or type-1 configuration transactions are generated when operating as Root Port to enumerate the PCIe hierarchy. The following summarizes when a type-0 or a type-1 configuration transaction is generated. The bus, device and function number terminology used in the following description is extracted from the incoming AXI address hitting the ECAM aperture.

- When the bus number in the ECAM address == PCIe core bus number.
    - For device number = 0 and function number = 0, an internal configuration access is generated for the integrated block for PCIe.
    - If either device number or function number is non-zero, transaction is ended with a DECERR.

- If the target bus number in an ECAM address == secondary bus number programmed through CSR module.
  - For device number = 0, a type-0 configuration TLP is transmitted.
  - For non-zero device number, the transaction is ended with DECERR.
- When an ECAM address targets a bus number that is different from the other options, a type-1 configuration TLP is transmitted.

Configuration Request Retry Status

In cases where an Endpoint is not ready to respond, a configuration request retry status (CRS) response is issued to incoming PCIe configuration requests.

Note: In this section, an Endpoint refers to the Endpoints that would be connected to a Zynq UltraScale+ MPSoC operating as a Root.

The PCIe Root Port CRS software visibility is controlled by the PCIE_ATTRIB.ATTR_79 bit [5]. If the CRS software visibility is enabled, then reads targeting DW0 in configuration space (Device ID: Vendor ID) result in, the AXI response OKAY with AXI data = `0xFFFF0001` (this special data means that the device has issued a CRS status).

If the CRS software visibility is not enabled, the AXI-PCIe bridge continues to retry the transaction until a status other than CRS is returned. However the transaction will be aborted with DECERR if it fails to succeed after being attempted for >1 second (the longest time period that a PCIe device is permitted to return CRS status).

Root Port Received Interrupt and Message Controller

A received interrupt and message controller collects interrupts and messages received from the PCIe hierarchy. Interrupt reception is applicable only to Root Port mode.

The following interrupt outputs are provided and connected to the AXI CPU (PS generic interrupt controller (GIC) in this case).

- Two interrupt ports for MSI.
  - Configurable address range, support for 64 vectors.
  - Each interrupt output provides interrupt for 32 vectors.
- One interrupt port for legacy interrupt.
- One interrupt port for DMA.
- One interrupt port for miscellaneous.

A 128-word deep message FIFO is implemented to hold messages and optionally MSI interrupts (based on msii_status_enable). Legacy interrupts, DMA channel interrupts, and optionally MSI are recorded into interrupt status registers. The FIFO level is indicated in the AXI_PCIE_MAIN.MSGF_RX_FIFO_LEVEL register. When this register is not zero, there are received interrupts or messages pending. The oldest received interrupt or message contents are available by reading the AXI_PCIE_MAIN.MSGF_RX_FIFO_TYPE, AXI_PCIE_MAIN.MSG, AXI_PCIE_MAIN.ADDRESS_LO, AXI_PCIE_MAIN.ADDRESS_HI, or AXI_PCIE_MAIN.DATA registers. When finished reading the current interrupt or message, the current element is removed from the FIFO by writing to the AXI_PCIE_MAIN.MSGF_RX_FIFO_POP register.

Each status register has a corresponding mask register; only when mask register bit entry = 1 and corresponding status register bit = 1, then an interrupt output is generated to the AXI CPU. For example, legacy interrupts provide AXI_PCIE_MAIN.MSGF_LEG_MASK and AXI_PCIE_MAIN.MSGF_LEG_STATUS registers. Only when MSGF_LEG_MASK[i] = 1 and MSGF_LEG_STATUS[i] = 1, is an interrupt output generated to the AXI-CPU. All four legacy interrupts are ORed together to generate one output interrupt.

Interrupts

Interrupt generation capability is provided to both the PCIe bus and the system interrupt controllers (see Chapter 13, Interrupts). This sections describes the various interrupts.

PCIe Bus Interface Interrupts

As an Endpoint, the controller supports, legacy, MSI (multi-vector up to four) and MSI-X (up to four vectors) interrupt generation. These interrupts (when enabled) are generated by DMA transactions due to the completion of a DMA transfer or due to an error event. The mask for these events are enabled in the AXIPCIE_DMA*.DMA_CHANNEL_PCIE_INTERRUPT_CONTROL register. A coalesce count option is also provided for DMA completion events so that the frequency of interrupts can be controlled.

A software controlled interrupt is provided (per DMA channel) and can be asserted without enabling the DMA channel. Four scratchpad registers (per DMA channel) are also provided. These can be asserted by writing to the AXIPCIE_DMA*.DMA_CHANNEL_PCIE_INTERRUPT_ASSERT [pcie_software_interrupt] register. All interrupts require enabling of the AXIPCIE_DMA*.DMA_CHANNEL_PCIE_INTERRUPT_CONTROL[interrupt_mask] bit.

When in Endpoint mode, the bridge optionally generates interrupts when cfg_pcie_int_axi_pcie_n = 0. When MSI-X is enabled, the bridge implements an MSI-X table and PBA at fixed offset with regards to cfg_dma_reg_bar. Each DMA channel in the bridge uses one MSI-X vector for interrupts (for example, i^{th} MSI-X table entry is used for i^{th} DMA channel interrupt generation. Any miscellaneous interrupt uses the MSI-X table's 0^{th} entry to generate MSI-X interrupt upstream.

Note: As an Endpoint, when legacy interrupts are used, only INTA is supported.

IMPORTANT: *As a Root Port, if an Endpoint sends non-compliant MSI TLP, it will be dropped. It is required for the first byte-enable field in the MSI TLP to be equal to all ones.*

System Interrupts

System interrupts can be generated when the controller for PCIe is used either as a Root Port or as an Endpoint. The five PCIe system interrupts (MSI0, MSI1, INT{A, B, C, D}, DMA, and MSC) are listed in Table 13-1.

As an Endpoint, the following interrupts can be generated to the system controller.

- DMA interrupts due to completion or an error when enabled; these are generated when the DMA operation is enabled.
- Since the PCIe protocol does not support interrupts downstream, the host software can create an interrupt in the AXI domain.
- Host software controlled interrupts provided per DMA channel which can be used for handshake purpose. Note that PCIe protocol doesn't support interrupts downstream so this provides a means of host (Root Port) interrupting processor on MPSoC Endpoint. The interrupts are asserted by writing to the AXIPCIE_DMA*.DMA_CHANNEL_AXI_INTERRUPT_ASSERT [axi_software_interrupt] register.

All interrupts require an enabled AXIPCIE_DMA*.DMA_CHANNEL_AXI_INTERRUPT_CONTROL [interrupt_enable] bit. Additionally, for AXI domain interrupts that are provided per DMA channel, the AXIPCIE_MAIN.MSGF_DMA_MASK bits for each DMA channel should be set.

Transaction Handling

This section provides an overall summary of PCIe↔AXI transaction handling and mapping.

Ingress Transactions

Figure 30-5 is a flowchart for ingress transaction handling.

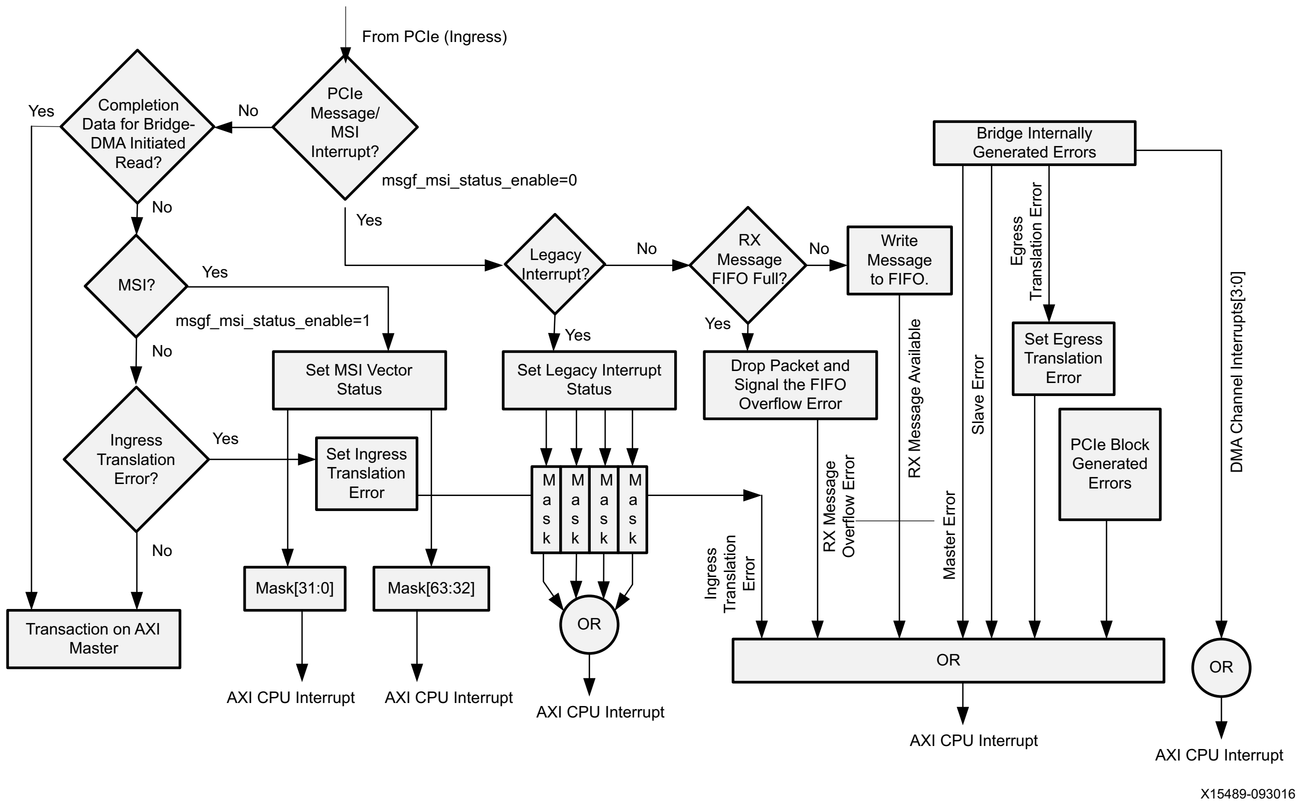

From PCIe (Ingress)
PCIe Message/ MSI Interrupt?
No
Completion Data for Bridge-DMA Initiated Read?
Yes
No
msgf_msi_status_enable=0
Yes
MSI?
Yes
msgf_msi_status_enable=1
No
Legacy Interrupt?
No
RX Message FIFO Full?
No
Write Message to FIFO.
Yes
Yes
Set MSI Vector Status
Set Legacy Interrupt Status
Drop Packet and Signal the FIFO Overflow Error
Ingress Translation Error?
Yes
Set Ingress Translation Error
No
Mask
Mask[31:0]
Mask[63:32]
OR
Transaction on AXI Master
AXI CPU Interrupt
AXI CPU Interrupt
AXI CPU Interrupt
Bridge Internally Generated Errors
Egress Translation Error
Set Egress Translation Error
PCIe Block Generated Errors
RX Message Available
Slave Error
Master Error
RX Message Overflow Error
Ingress Translation Error
DMA Channel Interrupts[3:0]
OR
AXI CPU Interrupt
OR
AXI CPU Interrupt
X15489-093016

Figure 30-5: **Ingress Transaction Handling**

PCIe to AXI Map

Table 30-4 summarizes the PCIe-AXI transaction mapping.

Table 30-4: **Ingress Transaction Map**

PCIe Transaction	AXI Transaction	Map Conditions
Memory read TLP	AXI read on AXI master port.	Translated address if ingress translation is hit. If translation is not hit, and subtractive decode is enabled, forward to AXI without translation. Otherwise, unsupported request.
Memory write TLP	AXI write on AXI master port.	Translated address if ingress translation is hit. Otherwise, the same address (no translation) on subtractive decode.
Configuration TLP	–	Handled internally by the integrated block for PCIe.
Successful Cpl (CfgWr response)	AXI response OKAY.	
Cpl with unsupported request (CfgWr response)	AXI response DEC_ERR.	
Cpl with unsupported request (CfgRd response)	AXI response DEC_ERR if cfg_rd_ur_is_ur_ok1s_n = 1. AXI response OKAY with data as all 1's when cfg_rd_ur_is_ur_ok1s_n = 0.	
Cpl with completer abort	AXI response SLVERR.	
Cpl with CRS	AXI response OKAY with data `0xFFFF0001` when CRS software visibility is enabled. AXI response DECERR after 1s retry when CRS software visibility is not enabled.	

Egress Transactions

Figure 30-6 is a flowchart for egress transaction handling.

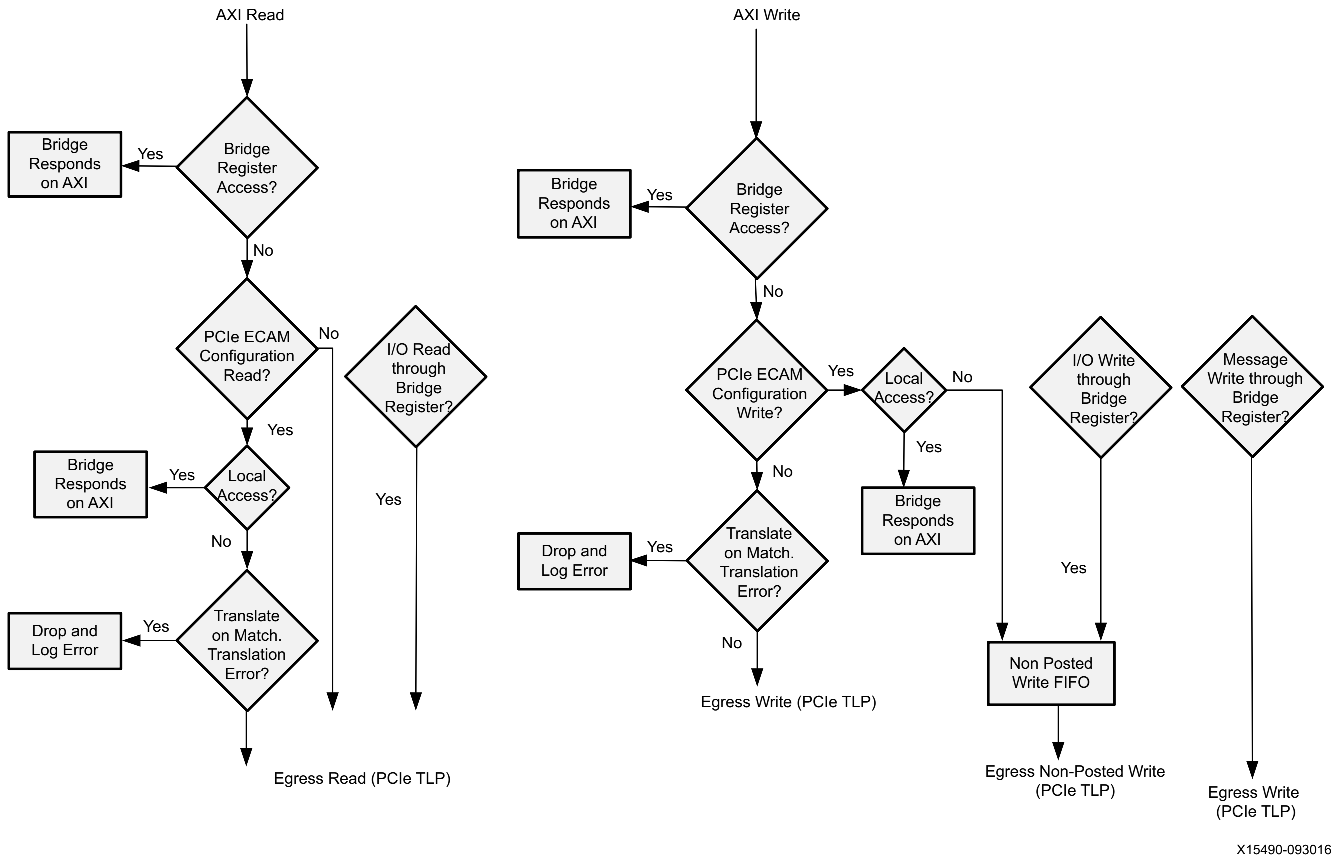


Figure 30-6: **Egress Transaction Handling**

ECAM write and I/O write transactions are non-posted in the PCIe domain. Non-posted transactions must not be allowed to stall posted transactions to avoid deadlock conditions. These non-posted writes require arbitration for a PCIe tag and completion handling resources managed by the bridge's reorder queue. These non-posted writes are not queued in reorder queue and instead are queued into an additional non-posted write FIFO.

AXI-PCIe Transaction Mapping

Table 30-5 summarizes AXI transaction mapping to PCIe domain.

Table 30-5: **Egress Transaction Map**

AXI Transaction	PCIe Transaction	Notes
AXI read transaction	Local bridge register read if BREG aperture is hit. DMA register read if DREG aperture is hit. Configuration read TLP if ECAM aperture is hit. Memory read TLP if no other aperture is hit.	For memory read TLP, the address is translated if the egress translation is hit. Otherwise, it remains the same.
AXI write transaction	Local bridge register write if BREG aperture is hit. DMA register write if DREG aperture is hit. Configuration write TLP if ECAM aperture is hit. Memory write TLP if no other aperture is hit.	For memory write TLP, the address is translated if the egress translation is hit. Otherwise, it remains the same.

To generate messages, the AXI-PCIe bridge provides registers. Refer to the AXI_PCIE_MAIN.TX_PCIE_MSG_* registers for details on generating these types of transactions in the *Zynq UltraScale+ MPSoC Register Reference* (UG1087) [Ref 4].

Note: Special handling is required by software for memory write transactions with ECRC errors. The PCI Express specification mandates that the ECRC errors be captured and signaled to the software. The error could be in the payload or in the header. If the payload is corrupted, a known location could receive incorrect data. If the address is corrupted, the transaction could end up at a completely incorrect slave. Software is required to read the header from the AER registers in the PCIe configuration space and take corrective action because, by the time software receives notification of such an event, the write transaction with the ECRC error could already be executed.

Endpoint Compliance

When doing PCIECV for PCISIG compliance, the Endpoint drivers on a host system are not installed. Likewise, when using the Zynq UltraScale+ MPSoC as an Endpoint for a PCIECV test, any driver accessing AXI-PCIe bridge registers running on the Zynq UltraScale+ MPSoC (APU or RPU clusters) should not be installed.

If the driver running on the Zynq UltraScale+ MPSoC accesses the AXI-PCIe bridge registers, it can cause the transaction pending bit (in PCIe configuration space) to be set, which would cause a PCIECV compliance failure.

Security Features

The AXI master is capable of generating transactions with TrustZone secure and non-secure classification. The TrustZone classification of each address translation, as well as each DMA channel, is configurable from the FPD SLCR SECURE register block.

The AXI master provides security ports, namely awprot[1] and arprot[1], are driven differently depending on the transaction source. The DMA transaction source is assigned a security level by the FPD_SLCR_SECURE.slcr_pcie[24:21] bits. Each bit corresponds to one DMA channel. The integrated block for PCIe on ingress translation hit, takes the security level provided by FPD_SLCR_SECURE.slcr_pcie[20:13].

Security levels are defined for other translation apertures in FPD_SLCR_SECURE.slcr_pcie register.

Note: The AXI-PCIe bridge does not implement a store and forward FIFO to drop a memory write packet that has an ECRC error in it. This type of memory write is eventually executed by the PS—either as a PCIe write to the bridge registers or as an AXI write transaction to an AXI slave internal to the PS, depending on the address in the header of the packet. ECRC errors are captured in AER capability. Xilinx recommends managing these packets in software on an individual basis.

TIP: *The default values represented on the Zynq UltraScale+ MPSoC Register Reference (UG1087) [Ref 4] for PCIE_ATTRIB registers are preset defaults. These values can be different depending upon the configuration used by the Processing System Configuration Wizard (PCW) in the zynq_ultra_ps_e. For configuration options, refer to the Zynq UltraScale+ MPSoC Processing System Product Guide (PG201) [Ref 5].*

DMA

The controller for PCIe contains a high-performance 4-channel direct memory access (DMA) engine. Each channel can be programmed for either transmit or receive DMA operation. Each channel can be controlled from both the PCIe or AXI domains. The DMA supports separate source and destination scatter-gather queues. The DMA hardware is responsible for merging the source and destination information for data movement. The scatter-gather elements can be located in either PCIe or AXI memory.

Each DMA channel implements 128 bytes of DMA registers. DMA channel registers are accessed through AXI slave when AXI transaction hits the DMA register translation. The channel registers are at (DREG + `0x0`) for first channel, (DREG + `0x80`) for second channel and so on. DMA channel registers are accessed through PCI Express through the BAR associated with DMA channel registers (cfg_dma_reg_bar). This access is by default BAR0. DMA channel registers are at (BAR0 + `0x00`) for the first channel and (BAR0 + `0x80`) for the second channel and so on.

Table 30-6: **DMA Channel Address Map**

DMA Channel	AXI Address	PCIe Address
0	DREG_BASE	cfg_dma_reg_bar[1]
1	DREG_BASE + `0x80`	cfg_dma_reg_bar + `0x80`
2	DREG_BASE + `0x100`	cfg_dma_reg_bar + `0x100`
3	DREG_BASE + `0x180`	cfg_dma_reg_bar + `0x180`

Notes:

1. By default, cfg_dma_reg_bar is BAR0.

Suffice DMA Descriptors

This section provides the details of the DMA descriptors. The scatter-gather queues supported by DMA are listed.

- SRC-Q provides data buffer source information and corresponding STAS-Q to indicate completion of SRC-Q processing by the DMA
- DST-Q provides destination buffer information and corresponding STAD-Q which indicates DST-Q processing completion by DMA
- Source and destination scatter-gather queues describe the fragmentation of the source and the destination memory. The queues are independent. The DMA channel merges the information from the queues to perform DMA operations based on the fragmentation of each queue.

The Q elements layout is shown in Figure 30-7.

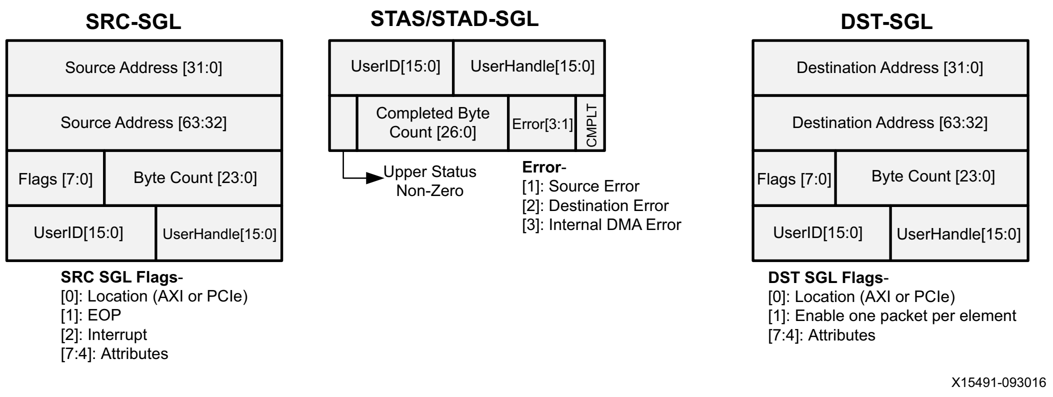


Figure 30-7: **DMA SGL-Q Format**

The source and destination scatter-gather Q elements are 128 bits wide. The corresponding status scatter-gather Q elements can be chosen to be either 32-bit or 64-bit.

The description of various format fields are listed in Table 30-7, Table 30-8, and Table 30-9.

Table 30-7: **SRC-Q Element Descriptions**

Field Name	Location	Description
Source address	[63:0]	Source address.
Byte count	[87:64]	Byte count, a value of 0 implies 2^{24} bytes.
Flags[7:0]	[95:88]	[7:4] DMA data read attribute If source is AXI m_arcache[3:0] = [7:4] If source is PCIe, PCIe attr[2:0] = [6:4] [3] Reserved [2] Interrupt A value of `1`, generates an interrupt when the status-Q is written with a DMA completion status and valid when EOP = `1`. The interrupt is generated in either PCIe or AXI or both directions based on the DMA channel interrupt register configuration. A value of `0`, does not generate an interrupt. [1] EOP A value of `1`, end of packet, status Q is updated when EOP is transferred to DMA destination. A value of `0`, not end of packet, packet can span multiple Q elements. [0] Location A value of `1`, DMA data source is AXI A value of `0`, DMA data source is PCIe
UserHandle[15:0]	[111:96]	The UserHandle is copied from SRC SGL with EOP = `1` to corresponding STAS-Q elements in the UserHandle field. It provides a means to associate SRC-Q to STAS-Q elements.
UserID[15:0]	[127:112]	UserID is copied from SRC SGL with EOP = `1` to corresponding STAS and STAD-Q element's UserID field. It provides a means to transfer user specific data from source to destination.

Table 30-8: **DST-Q Element Descriptions**

Field Name	Location	Description
Destination address	[63:0]	Destination address.
Byte count	[87:64]	Byte count. Value of 0 implies 2^{24} bytes.
Flags[7:0]	[95:88]	[7:4] DMA data write attribute. If destination is AXI m_awcache[3:0] = [7:4]. If destination is PCIe, PCIe Attr[2:0] = [6:4]. [3:2] Reserved [1] Enable one packet per destination SGL. A value of `1`, skip to next destination SGL on EOP. A value of `0`, pack packets back-to-back in destination SGL. [0] Location A value of `1`, DMA data destination is AXI. A value of `0`, DMA data destination is PCIe.
UserHandle[15:0]	[111:96]	UserHandle is copied from the final DST SGL element used in packet transfer to corresponding STAD-Q element's UserHandle field. It provides a means to associate DST-Q elements to STAD-Q elements.
Reserved	[127:112]	Reserved

Table 30-9: **Status Q Element Descriptions**

Field Name	Location	Description
UserID[15:0]	[63:48]	UserID copied from SRC SGL element with EOP=1
UserHandle[15:0]	[47:32]	For STAS-Q, this is copied from the SRC-Q element with EOP = `1`. For STAD-Q, this is copied from final DST-Q element used for packet transfer. This provides software with a means to associate SRC/DST-Q with STAS/STAD elements.
Upper status is non-zero	[31]	For 64-bit status elements this bit is `1` when [63:32] = 0. For 32-bit status elements, this bit always reads 0.
Completed byte count	[30:4]	Completed byte count. Range is 0 to (2^{27}–1).
Internal error	[3]	Internal error during DMA operation.
Destination error	[2]	Destination error during DMA operation.
Source error	[1]	Source error during DMA operation.
Completed	[0]	Status Q element completion indication.

IMPORTANT: *Both UserHandle and UserID cannot be zero at the same time when using 64-bit status queues and checking the "upper status is non-zero" bit for a status queue update by the DMA. When not using the UserID field, keep the UserID to a fixed non-zero signature value so that the UserHandle can start from the value of zero.*

Status-Q elements contain information for a single packet transfer. The upper status is a non-zero bit that provides a method to ensure that CPUs read the correct information in case of 32-bit atomic operations.

DMA errors are indicated in status-Q elements for packets that are able to be completed. In error situations where packets cannot complete, DMA errors are indicated in channel's error status registers.

These Qs can be resident either in host memory or AXI memory. Q elements are required to be in contiguous location for DMA to fetch multiple SRC/DST-Q elements in a burst fetch. The software driver sets up the Q elements in contiguous location and DMA takes care of wrap-around of Q. Every DMA channel has the following registers pertaining to each Q.

- Q_PTR: Indicates the starting address of the Q.
- Q_SIZE: The number of SGL elements in Q.
- Q_LIMIT: An index of the first element still owned by the software; DMA hardware wraps around to start element location when Q_LIMIT is equal to Q_SIZE.

Figure 30-8 shows a DMA SGL-Q operation.

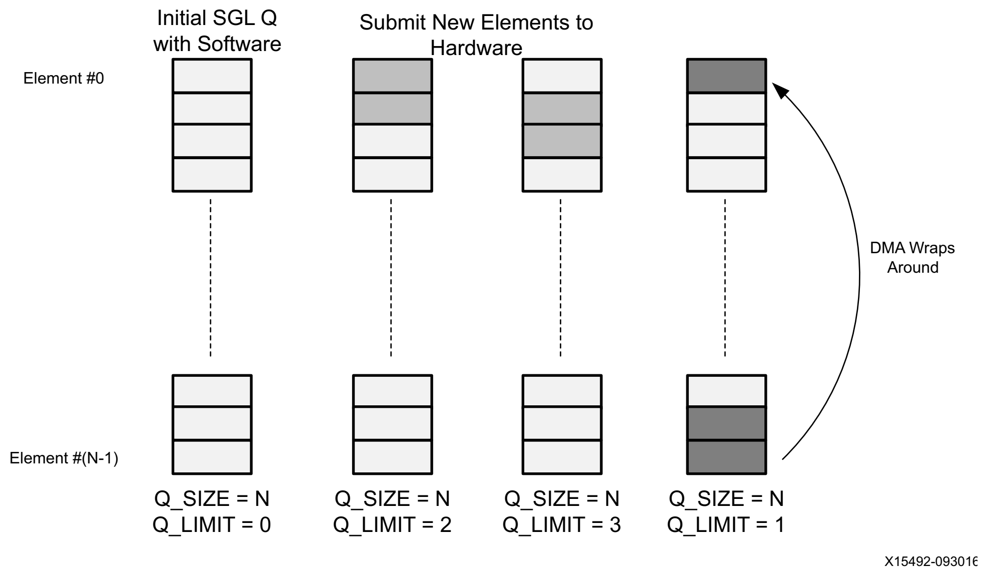


Figure 30-8: **DMA SGL-Q Operation Summary**

Status Updates

The status elements are updated only on EOP and not every SGL element. This section describes the status updates and suggests use of UserHandle field to associate multiple SRC or DST-Q elements to a single STAS or STAD-Q element update.

TIP: *The DMA does not use the UserHandle or UserID fields. To utilize the UserHandle field, you can implement the same relationships entirely in software layer and use the UserHandle and UserID fields in your custom environment.*

Relationship between SRC-Q and STAS-Q

As shown in Figure 30-9, packet-0 spans across three SRC-Q elements; the third element indicates EOP=1 with UserHandle=2. On EOP, DMA updates STAS-Q with UserHandle = 2, which corresponds to the handle value in the SRC-Q element with EOP = 1. Similarly, packet-1 spans two elements and in STAS-Q the updated handle value corresponds to EOP = 1 element. This UserHandle mechanism allows software to associate number of SRC-Q elements corresponding to a STAS-Q update.

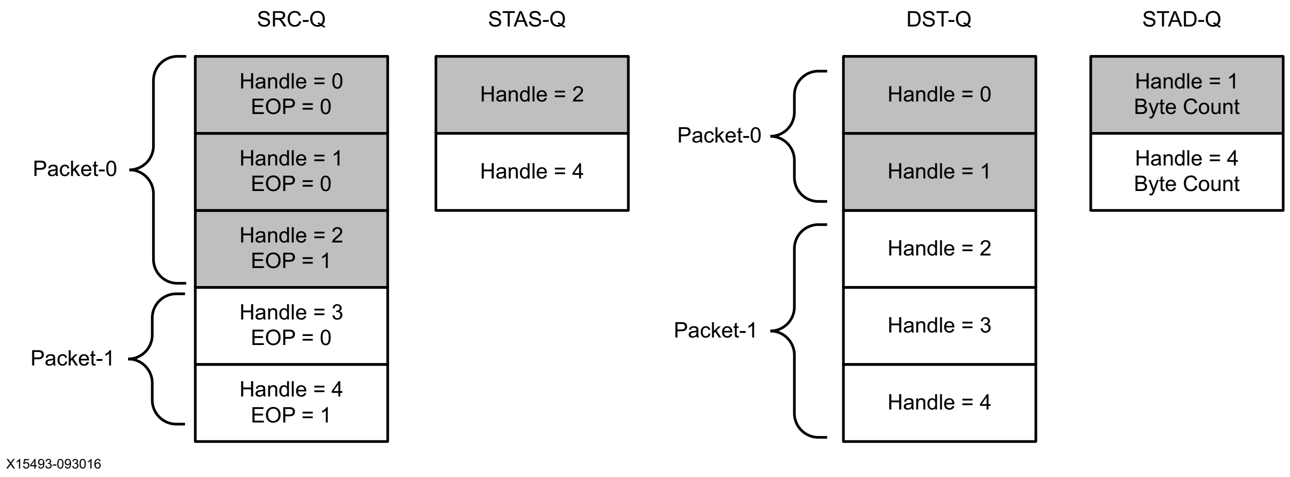


Figure 30-9: **UserHandle Usage**

Relationship between DST-Q and STAD-Q

Software sets up DST-Q elements with predefined UserHandle values and points to empty buffers. For example, in Figure 30-9 a packet-0 spans across two DST-Q elements; one STAD-Q element is updated with the handle value of the last DST-Q element used by the packet and corresponding packet length. Software maintains the number of DST-Q elements used (buffers used and appropriate buffer fragment pointers) for a particular status completion.

The DMA provides programmable options for the following.

- Number of status Qs: The DMA can be operated in 3-Q mode or 4-Q mode. In 3-Q mode, only a single status-Q is used for both the SRC and DST queues.
- 32-bit or 64-bit status Q: DMA provides an option to use the 32-bit status Q or 64-bit status Q. The upper 32 bits of the status Q are unavailable when the 32-bit status Q mode of operation is programmed.

Note: In rare circumstances, the status-Q might not be updated with a completion status when an interrupt is received. These issues can be resolved by reading the DMA completion interrupt status register twice before reading the status-Q.

DMA Channel Flow Control

Each DMA channel contains three internal per-channel first in, first out (FIFO) buffers.

1. One FIFO caches up to eight source SGL elements.
   a. The buffer is filled by reading the source SGL queue, that is made available as the SRC_Q_LIMIT register is updated.
   b. The buffer is emptied when DMA transactions, that fully satisfy the size of the current source SGL element, are created.
2. One FIFO caches up to eight destination SGL elements.
   a. The buffer is filled by reading the destination SGL queue that is made available as the DST_Q_LIMIT register is updated.
   b. The buffer is emptied when DMA transactions, that fully satisfy the size of the current destination SGL element, are created.
3. One FIFO caches up to four DMA completion status.
   a. The buffer is filled when DMA transfers with SRC SGL EOP == 1 completes.
   b. The buffer is emptied when DMA completion status is written to the DMA completion status (STAS/SATD) queues.

DMA channels arbitrate amongst other DMA channels and bridge functions, using a round-robin arbitration scheme, to carry out a DMA transfer. The DMA transaction size arbitrated is up to 512 bytes for x1 to x4 PCI express lanes.

The source SGL element, and the destination SGL element are available in the internal FIFO cache. The DMA completion status FIFO has at least one element available to receive DMA completion status. When a DMA source SGL with EOP == 1 completes, the DMA completion status is written into the per channel DMA completion status internal FIFO until it can be written to the external STAS/STAD DMA completion status queues.

DMA Error Detection

The following describes scenarios resulting in DMA errors.

- Errors occurred while reading the SGL from the SRC or DST QUEUE or while reading the DMA data.
    - Errors detected and reported by PCIe read.
        - Unsupported request or completer abort status returned for read.
        - ECC/parity or ECRC error detected by the integrated block for PCIe.
        - Poisoned TLP (EP bit set in a received TLP that contained a data payload).
    - Errors detected and reported by AXI read.
        - AXI error response returned for a read (read response is not OKAY).
        - No response received for read (completion timeout).
        - Internal RAM ECC error while processing SGL or DMA data.
- Errors occurred while writing DMA completion status to the status queue or while writing DMA data.
    - Error response returned for STA(S/D) QUEUE or DMA write.
        - PCIe does not provide a response for memory writes.
        - AXI error response returned for a write (write response is not OKAY).
    - Internal ECC error while processing STA(S/D) QUEUE or DMA write.
- Out of range LIMIT pointer detected while DMA is enabled.
    - SRC_LIMIT >= SRC_SIZE.
    - DST_LIMIT >= DST_SIZE.
    - STA_LIMIT >= STA_SIZE.
- Out of range NEXT pointer detected while DMA is enabled.
    - SRC_NEXT >= SRC_SIZE.
    - DST_NEXT >= DST_SIZE.
    - STA_NEXT >= STA_SIZE.
- Invalid queue SIZE detected while DMA is enabled.
    - SRC_SIZE < 2.
    - DST_SIZE < 2.
    - STA_SIZE < 2.

DMA Error Handling

The DMA core associates the DMA errors with the source DMA channel and reports the errors on a per DMA channel basis.

When any DMA error occurs (as listed in DMA Error Detection), it is handled by the DMA channel as follows.

1. DMA enable register value is set to 0 (if 1).
   - No new DMA operations are scheduled.
   - DMA operations, started while the DMA was still enabled, continue for completion.

   Note: After the error is detected, expect continued DMA activity for a short period of time for the already in process transactions to complete.
2. PCIe DMA error and AXI DMA error registers are set to 1 to log the error.
3. A PCIe and AXI interrupt event is scheduled and reflected in the PCIe interrupt status and AXI interrupt status. Error interrupts are handled the same way as regular DMA interrupts.
   - The same interrupt vector is used as for regular DMA completion interrupts.
   - The interrupt is generated only when interrupts are enabled. Software can read the PCIe DMA error and/or the AXI DMA error to determine if the interrupt is generated due to an error.

When a DMA transaction fully completes, and (normally) a status queue element is written, then any errors detected during the DMA are reported in the status queue element written during DMA completion. This is a common scenario for small DMA operations. For larger DMA, because DMA operations are halted as soon as possible after detecting a DMA error, the DMA transaction with the error does not complete and the status queue element is not written.

When an error occurs, the software can continue operation without performing a system reset (depending on the severity of the error). In such cases, the DMA channel must be reset before reusing it again. For more details on how to reset a channel, refer to the Programming Topics.

DMA Operation

This section describes two modes of DMA operation.

- Dual CPU control, where a single DMA channel is managed by both the host CPU as well as the AXI CPU. This requires a software device driver to be running on both the host CPU and the Arm CPU on the Zynq UltraScale+ MPSoC.
- Single CPU control, where a single DMA channel is only managed by the host CPU.

IMPORTANT: *The SGL elements contain fields which determine direction of data flow. A single DMA channel cannot be tasked with elements to simultaneously support DMA transfers with multiple directions of data flow. For example, in the case of SRC-SGL, if the first element indicates PCIe as a source of data, the subsequent element cannot indicate AXI as source of data. Before changing dataflow directions on the same DMA channel, all DMA transfers for the prior data flow direction must be fully completed.*

Dual-CPU Control

The DMA supports multi-CPU DMA operation, that is, each DMA channel can be managed by both host CPU as well as an AXI CPU.

The dataflow for this mode is described in this section. The example assumes that the Zynq UltraScale+ MPSoC’s integrated block for PCIe is an Endpoint.

System to Card (Host Memory to EP Memory)

1. The host software sets up and manages SRC and STAS Q elements in host memory; Arm software (driver on the Zynq UltraScale+ MPSoC) sets up and manages DST and STAD Q elements in AXI memory.
2. The source buffer lies in PCIe memory and destination buffer in AXI memory.
3. DMA channel's registers are programmed by both host CPU and AXI CPU (registers corresponding to SRC/STAS Qs by host and DST/STAD Qs by AXI CPU).
4. On DMA channel enable, the SRC elements are fetched over PCIe and DST elements over AXI.
5. Source buffer pointed by SRC-Q is fetched over PCIe and made available (AXI write transaction on AXI master port) to the destination address (provided by DST-Q) on AXI.
6. On completion of the operation the STAS-Q is updated in host memory and the STAD-Q is updated in AXI memory.

Card to System Flow (EP Memory to Host Memory)

1. Host software sets up and manages the DST and STAD Q elements in host memory; Arm software (driver on the Zynq UltraScale+ MPSoC) sets up and manages the SRC and STAS Q elements in AXI memory.
2. The source buffer lies in the AXI memory and destination buffer in PCIe memory.
3. DMA channel's registers are programmed by both host CPU and AXI CPU. Registers corresponding to SRC/STAS Qs by the AXI CPU, and DST/STAD Qs by the host.
4. On DMA channel enable, SRC elements are fetched over AXI (read transactions) and DST elements over PCIe.

5. The source buffer pointed by SRC-Q is fetched over AXI (read transaction) and made available to the destination address (provided by DST-Q) as memory write TLP over PCIe.
6. On completion of the operation, the STAS-Q is updated in AXI memory and the STAD-Q is updated in host memory.

Single CPU Control

In this mode, host software controls all the Qs for a DMA channel. The AXI domain address for buffer transfers needs to be made known to host software in advance.

System to Card Flow (Host memory to EP)

1. Software sets up the SRC-Q with a buffer address in the host and an appropriate buffer size in host memory.
2. Software sets up the DST-Q with a buffer address in the AXI domain and an appropriate buffer size in host memory.
3. Software sets up the STAS-Q and STAD-Q in host memory.
4. On enabling the DMA, the DMA fetches SRC and DST elements over PCIe.
5. The DMA fetches the buffer pointed to by SRC elements (upstream memory read) and provides it on the AXI interface (as AXI write transaction) targeting the AXI address provided in DST-Q.
6. On completion of DMA transfer (encountering EOP), STAS-Q and STAD-Q are updated in host memory.

Card to System Flow (EP to Host Memory)

1. Software sets up the SRC-Q with a buffer address pointing to the AXI domain and the appropriate buffer size in host memory.
2. Software sets up the DST-Q with a buffer address in the host and the appropriate buffer size.
3. Software sets up STAS-Q and STAD-Q in host memory.
4. On enabling the DMA, the DMA fetches the SRC and DST elements over PCIe.
5. The DMA fetches the buffer pointed to by the SRC elements over AXI (through AXI read transaction) and writes it into the address provided in DST-Q in host memory (upstream memory write).
6. On completion of the DMA transfer (encountering EOP), STAS-Q and STAD-Q are updated in host memory.

Each DMA channel provides scratchpad and doorbell registers. The doorbell register is useful to raise interrupts from PCIe to AXI domains as downstream interrupts are not supported in PCIe. The scratchpad registers can be used for communication in the case of a host CPU interrupting an AXI CPU.

IMPORTANT:
Xilinx strongly recommends using the integrated DMA controller in the PS PCIe to exercise PCIe traffic.

Deadlock situations can occur when the PS PCIe shares the path between the CCI and the FPD Main Switch with an external master also targeting the PS PCIe interface. Refer to Figure 15-1 *to see this path.*

When the Zynq UltraScale+ MPSoC is used as an Endpoint, external DMA like FPD DMA or PL DMA IP connected to S_AXI_HP[0:3]_FPD can be used to exercise PCIe traffic. This is because they to not route traffic through the CCI to the FPD.

Do not use PL DMA IP connected to S_AXI_HPC[0:1], S_AXI_LPD, or any other PS masters like LPD DMA to exercise PCIe traffic because these masters use the shared path between the CCI and the FPD Main Switch.

When the Zynq UltraScale+ MPSoC is used as a Root Port, Xilinx recommends that PCIe link partners (Endpoints) access only the PS-DDR. They should not access any other memory like OCM or PL memory on the Root Port. It is also recommended that the GPU (if enabled) should not access Programmable Logic, because the share path between the CCI and the FPD Main Switch can result in a deadlock situation.

For more information, see Xilinx Answer 72341.

I/O Signals

MIO Signals

The PCIe Root Port mode and Endpoint mode reset signals are routed to specific MIO pins as listed in Table 30-10.

Table 30-10: **PCIe Reset Signals on MIO**

PCIe Reset	MIO Pins	I/O	Default Input Value to Controller
Rootport reset output (use GPIO controller)	0 ... 77	O	~
Endpoint reset input	29,30,31,33,34,35,36,37	I	0

Register Overview

This section provides an overview of the registers in the PCI Express controller and DMA.

Table 30-11 lists the bridge core registers.

Table 30-12 lists ingress address translation registers.

Table 30-13 lists the egress address translation registers.

Table 30-14 lists the DMA channel control and status registers.

Bridge Core Registers

The bridge core registers program the bridge features and apertures for various access regions. Table 30-11 summarizes the bridge core registers.

Table 30-11: **Bridge Core Registers**

Register Name	Description
BRIDGE_CORE_CFG_PCIE_RX0	PCI Express receive access and BAR configuration.
BRIDGE_CORE_CFG_PCIE_RX1	PCI Express receive transaction attribute handling.
BRIDGE_CORE_CFG_AXI_MASTER	AXI master maximum payload size configuration.
BRIDGE_CORE_CFG_PCIE_TX	PCI Express transmit cut through configuration.
BRIDGE_CORE_CFG_INTERRUPT	PCI Express core interrupt routing configuration.
BRIDGE_CORE_CFG_RAM_DISABLE0	ECC RAM 1-bit error correction enable/disable (designs with ECC support only).
BRIDGE_CORE_CFG_RAM_DISABLE1	ECC RAM 2-bit error handling enable/disable (designs with ECC support only).
BRIDGE_CORE_CFG_PCIE_RELAXED_ORDER	PCI Express receive completion ordering configuration.
BRIDGE_CORE_CFG_PCIE_RX_MSG_FILTER	PCI Express receive message filtering configuration.
BRIDGE_CORE_CFG_RQ_REQ_ORDER	PCI Express and AXI read reorder queue completion ordering configuration.
BRIDGE_CORE_CFG_PCIE_CREDIT	PCI Express transmit completion header and data credit metering configuration.
BRIDGE_CORE_CFG_AXI_M_W_TICK_COUNT	AXI master write completion timeout configuration.
BRIDGE_CORE_CFG_AXI_M_R_TICK_COUNT	AXI master read completion timeout configuration.
BRIDGE_CORE_CFG_CRS_RPL_TICK_COUNT	PCIe configuration write/read request CRS replay timeout configuration.
E_BREG_CAPABILITIES	Egress bridge register translation: capabilities.
E_BREG_STATUS	Egress bridge register translation: status.
E_BREG_CONTROL	Egress bridge register translation: control.

Table 30-11: **Bridge Core Registers *(Cont'd)***

Register Name	Description
E_BREG_BASE_LO	Egress bridge register translation: source address Low.
E_BREG_BASE_HI	Egress bridge register translation: source address High.
E_ECAM_CAPABILITIES	Egress ECAM translation: capabilities.
E_ECAM_STATUS	Egress ECAM translation: status.
E_ECAM_CONTROL	Egress ECAM translation: control.
E_ECAM_BASE_LO	Egress ECAM translation: source address Low.
E_ECAM_BASE_HI	Egress ECAM translation: source address High.
E_MSXT_CAPABILITIES	Egress MSI-X table translation: capabilities.
E_MSXT_STATUS	Egress MSI-X table translation: status.
E_MSXT_CONTROL	Egress MSI-X table translation: control.
E_MSXT_BASE_LO	Egress MSI-X table translation: source address Low.
E_MSXT_BASE_HI	Egress MSI-X table translation: source address High.
E_MSXP_CAPABILITIES	Egress MSI-X PBA translation: capabilities.
E_MSXP_STATUS	Egress MSI-X PBA translation: status.
E_MSXP_CONTROL	Egress MSI-X PBA translation: control.
E_MSXP_BASE_LO	Egress MSI-X PBA translation: source address Low.
E_MSXP_BASE_HI	Egress MSI-X PBA translation: source address High.
E_DREG_CAPABILITIES	Egress DMA register translation: capabilities.
E_DREG_STATUS	Egress DMA register translation: status.
E_DREG_CONTROL	Egress DMA register translation: control.
E_DREG_BASE_LO	Egress DMA register translation: source address Low.
E_DREG_BASE_HI	Egress DMA register translation: source address High.
E_ESUB_CAPABILITIES	Egress subtractive decode translation: capabilities.
E_ESUB_STATUS	Egress subtractive decode translation: status.
E_ESUB_CONTROL	Egress subtractive decode translation: control.
I_MSII_CAPABILITIES	Ingress PCI Express received MSI interrupt translation: capabilities.
I_MSII_CONTROL	Ingress PCI Express received MSI interrupt translation: control.
I_MSII_BASE_LO	Ingress PCI Express received MSI interrupt translation: source address Low.
I_MSII_BASE_HI	Ingress PCI Express received MSI interrupt translation: source address High.
I_MSIX_CAPABILITIES	Ingress PCI Express received MSI-X interrupt translation: capabilities.
I_MSIX_CONTROL	Ingress PCI Express received MSI-X interrupt translation: control.
I_MSIX_BASE_LO	Ingress PCI Express received MSI-X interrupt translation: source address Low.

Table 30-11: **Bridge Core Registers *(Cont'd)***

Register Name	Description
I_MSIX_BASE_HI	Ingress PCI Express received MSI-X interrupt translation: source address High.
I_ISUB_CAPABILITIES	Ingress subtractive decode translation: capabilities.
I_ISUB_STATUS	Ingress subtractive decode translation: status.
I_ISUB_CONTROL	Ingress subtractive decode translation: control.
MSGF_MISC_STATUS	Received interrupt and message controller: miscellaneous interrupt status.
MSGF_MISC_MASK	Received interrupt and message controller: miscellaneous interrupt mask.
MSGF_MISC_SLAVE_ID	Slave error AXI ID.
MSGF_MISC_MASTER_ID	Master error AXI ID.
MSGF_MISC_INGRESS_ID	Ingress error AXI ID.
MSGF_MISC_EGRESS_ID	Egress error AXI ID.
MSGF_LEG_STATUS	Legacy interrupt status.
MSGF_LEG_MASK	Legacy interrupt mask.
MSGF_MSI_STATUS_LO	MSI interrupt status.
MSGF_MSI_STATUS_HI	MSI interrupt status.
MSGF_MSI_MASK_LO	MSI interrupt mask.
MSGF_MSI_MASK_HI	MSI interrupt mask.
MSGF_DMA_STATUS	DMA interrupt status.
MSGF_DMA_MASK	DMA interrupt mask.
MSGF_RX_FIFO_LEVEL	Received interrupt and message FIFO: level.
MSGF_RX_FIFO_POP	Received interrupt and message FIFO: pop element.
MSGF_RX_FIFO_TYPE	Received interrupt and message FIFO: message/interrupt type.
MSGF_RX_FIFO_MSG	Received message header.
MSGF_RX_FIFO_ADDRESS_LO	Received message/interrupt address.
MSGF_RX_FIFO_ADDRESS_HI	Received message/interrupt address.
MSGF_RX_FIFO_DATA	Received message/interrupt data payload.
TX_PCIE_IO_EXECUTE	PCIe I/O write/read request execution.
TX_PCIE_MSG_EXECUTE	PCIe message request execution.
TX_PCIE_MSG_CONTROL	PCIe message request execution: control.
TX_PCIE_MSG_SPECIFIC_LO	PCIe message request execution: message specific.
TX_PCIE_MSG_SPECIFIC_HI	PCIe message request execution: message specific.
TX_PCIE_MSG_DATA	PCIe message request execution: message data payload.

Address Translation Registers

There are eight address translation apertures in each direction namely ingress (PCIe to AXI) and egress (AXI to PCIe). Each aperture defines registers for translation that are listed in Table 30-12 and Table 30-13. The first translation aperture starts at offset `0x00`, the next one at `0x20`, and the subsequent one at `0x40`, and so on.

Table 30-12: **Ingress Address Translation Registers**

Register Name	Description
TRAN_INGRESS_CAPABILITIES	Ingress AXI translation: capabilities.
TRAN_INGRESS_STATUS	Ingress AXI translation: status.
TRAN_INGRESS_CONTROL	Ingress AXI translation: control.
TRAN_INGRESS_SRC_BASE_LO	Ingress AXI translation: source address Low.
TRAN_INGRESS_SRC_BASE_HI	Ingress AXI translation: source address High.
TRAN_INGRESS_DST_BASE_LO	Ingress AXI translation: destination address Low.
TRAN_INGRESS_DST_BASE_HI	Ingress AXI translation: destination address High.

Table 30-13: **Egress Address Translation Registers**

Register Name	Description
TRAN_EGRESS_CAPABILITIES	Egress AXI translation: capabilities.
TRAN_EGRESS_STATUS	Egress AXI translation: status.
TRAN_EGRESS_CONTROL	Egress AXI translation: control.
TRAN_EGRESS_SRC_BASE_LO	Egress AXI translation: source address Low.
TRAN_EGRESS_SRC_BASE_HI	Egress AXI translation: source address High.
TRAN_EGRESS_DST_BASE_LO	Egress AXI translation: destination address Low.
TRAN_EGRESS_DST_BASE_HI	Egress AXI translation: destination address High.

DMA Channel Control and Status Registers

There are four DMA channels at offsets `0x00` for channel-0, `0x80` for channel-1, `0x100` for channel-2, and `0x180` for channel-3. Each channel has its own control and status register set (Table 30-14).

Table 30-14: **DMA Channel Control and Status Registers**

Register Name	Description
DMA_CHANNEL_SRC_Q_PTR_LO	Source queue base address Low.
DMA_CHANNEL_SRC_Q_PTR_HI	Source queue base address High.
DMA_CHANNEL_SRC_Q_SIZE	Source queue size.
DMA_CHANNEL_SRC_Q_LIMIT	Source queue limit pointer.
DMA_CHANNEL_DST_Q_PTR_LO	Destination queue base address Low.

Table 30-14: **DMA Channel Control and Status Registers *(Cont'd)***

Register Name	Description
DMA_CHANNEL_DST_Q_PTR_HI	Destination queue base address High.
DMA_CHANNEL_DST_Q_SIZE	Destination queue size.
DMA_CHANNEL_DST_Q_LIMIT	Destination queue limit pointer.
DMA_CHANNEL_STAS_Q_PTR_LO	Source status queue base address Low.
DMA_CHANNEL_STAS_Q_PTR_HI	Source status queue base address High.
DMA_CHANNEL_STAS_Q_SIZE	Source status queue size.
DMA_CHANNEL_STAS_Q_LIMIT	Source status queue limit pointer.
DMA_CHANNEL_STAD_Q_PTR_LO	Destination status queue base address Low.
DMA_CHANNEL_STAD_Q_PTR_HI	Destination status queue base address High.
DMA_CHANNEL_STAD_Q_SIZE	Destination status queue size.
DMA_CHANNEL_STAD_Q_LIMIT	Destination status queue limit pointer.
DMA_CHANNEL_SRC_Q_NEXT	Source queue next pointer.
DMA_CHANNEL_DST_Q_NEXT	Destination queue next pointer.
DMA_CHANNEL_STAS_Q_NEXT	Source status queue next pointer.
DMA_CHANNEL_STAD_Q_NEXT	Destination status write only to initialize the DMA channel.
DMA_CHANNEL_SCRATCH0	Scratchpad register.
DMA_CHANNEL_SCRATCH1	Scratchpad register.
DMA_CHANNEL_SCRATCH2	Scratchpad register.
DMA_CHANNEL_SCRATCH3	Scratchpad register.
DMA_CHANNEL_PCIE_INTERRUPT_CONTROL	PCI Express interrupt control.
DMA_CHANNEL_PCIE_INTERRUPT_STATUS	PCI Express interrupt status.
DMA_CHANNEL_AXI_INTERRUPT_CONTROL	PCI Express interrupt control.
DMA_CHANNEL_AXI_INTERRUPT_STATUS	AXI interrupt status.
DMA_CHANNEL_PCIE_INTERRUPT_ASSERT	PCI Express interrupt assertion.
DMA_CHANNEL_AXI_INTERRUPT_ASSERT	AXI interrupt assertion.
DMA_CHANNEL_DMA_CONTROL	DMA channel control.
DMA_CHANNEL_DMA_STATUS	DMA channel status.

Programming Topics

This section summarizes programming of the PCI Express controller for Endpoint and Root Port mode operations.

Programming the PS-GTR Transceiver

The following steps are used to program the PS-GTR transceiver interface to support the PCI Express protocol. For more information on the PS-GTR transceiver interface, refer to Chapter 29, PS-GTR Transceivers.

1. Assign SerDes lanes to the PCIe PHY that presents a PIPE interface to the controller for PCIe.
2. Program SERDES.ICM_CFG0 and SERDES.ICM_CFG1 to support the PCIe protocol lanes as per the requirement (Table 30-15).

Table 30-15: **PS-GTR Multiplexer Configuration for PCI Express Lanes**

PCI Express Lane Configuration	Registers to program	Comments
x1	SERDES.ICM_CFG0[L0_icm_cfg] = `1`	Other 3 lanes (L1, L2, L3) can be used by other protocols like SATA, DisplayPort, USB, GEM.

x2	SERDES.ICM_CFG0[L0_icm_cfg] = `1`
	SERDES.ICM_CFG0[L1_icm_cfg] = `1`
		Other 2 lanes (L2, L3) can be used by other protocols like SATA, DisplayPort, USB, GEM.

x4	SERDES.ICM_CFG0 = `0x0011`
	SERDES.ICM_CFG1 = `0x0011`
		All lanes are assigned to PCIe protocol.

3. Set the PLL reference clock to 100 MHz: SERDES.PLL_REF_SEL0 = `0x0D`

Programming Reset Pin

Program the MIO registers in the IOU_SLCR module to configure the PCIe reset pin. For the Endpoint port, the PCIe reset pin is configured as an input. For the Root Port, it is configured as an output.

- For the Endpoint port, use one of MIO_PIN_[29,30,31,33,34,35,36,37] from the IOU_SLCR module as PCIe reset input (based on board layout). The input reset signal is listed in table Table 30-10.
- For the Root Port, use any GPIO to map the reset output, which is driven by software.

Programming Controller

The following steps describe the sequence of operations to program the PCI Express controller.

1. Program the CRF_APB.RST_FPD_TOP register to release pcie_cfg_rst, gt_rst, and pcie_bridge_reset.
2. Program the CRF_APB.PCIE_REF_CTRL register to activate the clock. The minimum required values for 250 MHz are set as default divisor values. Frequencies that are less than 250 MHz can have performance implications.
3. Program the integrated block for PCIe to Endpoint or Root Port role using the APB interface. The default values in registers are for an Endpoint mode of operation.
4. For the Root Port mode operation, program the following.
    a. Set the BAR and the memory base/limit registers to defaults for the Root Port, as documented in the *Zynq UltraScale+ MPSoC Register Reference* (UG1087) [Ref 4].

       PCIE_ATTRIB.ATTR_7= `0x0`
       PCIE_ATTRIB.ATTR_8= `0x0`
       PCIE_ATTRIB.ATTR_9= `0x0`
       PCIE_ATTRIB.ATTR_10= `0x0`
       PCIE_ATTRIB.ATTR_11= `0xFFFF`
       PCIE_ATTRIB.ATTR_12= `0xFF`
       PCIE_ATTRIB.ATTR_13= `0x0`
       PCIE_ATTRIB.ATTR_15= `0xFFF0`
       PCIE_ATTRIB.ATTR_16= `0xFFF0`
       PCIE_ATTRIB.ATTR_17= `0xFFF1`
       PCIE_ATTRIB.ATTR_18= `0xFFF1`

       PCIE_ATTRIB.ATTR_101 [ATTR_DISABLE_BAR_FILTERING] = `0x1` (this setting is specific to RP mode).

       Note: For EP mode, BAR settings are dependent on user selection such as size, prefetchable or not, etc.

b. Change the class code.

   PCIE_ATTRIB.ATTR_24 [ATTR_CLASS_CODE] = `0x400`
   PCIE_ATTRIB.ATTR_25 [ATTR_CLASS_CODE] = `0x6`

c. Change the header to type-1.

   PCIE_ATTRIB.ATTR_34 [ATTR_HEADER_TYPE] = `0x1`
   PCIE_ATTRIB.ATTR_100 [ATTR_UPSTREAM_FACING] = `0x0`

d. Change the device port type to Root Port.

   PCIE_ATTRIB.ATTRIB_50 [ATTR_PCIE_CAP_DEVICE_PORT_TYPE] = `0x4`

e. Change the Next pointer for PM capability to point to PCIe capability.

   PCIE_ATTRIB.ATTRIB_53 [ATTR_PM_CAP_NEXTPTR] = `0x60`

f. Disable the MSI capability.

   PCIE_ATTRIB.ATTRIB_41 = `0x0`

g. Enable the routing of various message TLPs to the bridge from the integrated block for PCIe.

   PCIE_ATTRIB.ATTRIB_101 [ATTR_ENABLE_MSG_ROUTE] = `0x7FF`

h. Set the credits to defaults, as documented in the register database.

   PCIE_ATTRIB.ATTR_105 [ATTR_VC0_TOTAL_CREDITS_CD] = `0xCD`
   PCIE_ATTRIB.ATTR_106 [ATTR_VC0_TOTAL_CREDITS_CH] = `0x24`
   PCIE_ATTRIB.ATTR_106 [ATTR_VC0_TOTAL_CREDITS_NPH] = `0xC`
   PCIE_ATTRIB.ATTR_107 [ATTR_VC0_TOTAL_CREDITS_NPD] = `0x18`
   PCIE_ATTRIB.ATTR_108 [ATTR_VC0_TOTAL_CREDITS_PD] = `0xB5`
   PCIE_ATTRIB.ATTR_109 [ATTR_VC0_TOTAL_CREDITS_PH] = `0x20`

i. CRS SW visibility is specific to RP mode.

   PCIE_ATTRIB.ATTR_79 [ATTR_ROOT_CAP_CRS_SW_VISIBILITY] = `0x1`

5. Program CRF_APB.RST_FPD_TOP to release pcie_ctrl_rst.

At this point, the controller is ready to initiate link training with a link partner, if pcie_reset_n (PERST#) is released by the board or the host.

Note: When using the Xilinx delivered tool flow, the attributes for Endpoint or Root Port mode operation are set by the first-stage boot loader.

Bridge Initialization

Bridge initialization is performed by the software driver running on the PS processor.

Note: When using 32-bit addressing mode, Xilinx recommends using 256 MB space for PCIe (starting at address `0xE000_0000`). For Root Port, 16 MB of this space can be used for ECAM and the rest of the 240 MB can be used for BARs for Endpoints.
If you require more than 256 MB space for PCIe, then use a higher addressing mode (40-bit or 44-bit).

1. Program the registers in the AXI-PCIe bridge with the following information.

Note: After power on, these registers can be accessed using address `0xFD0E_0000` as documented in *the Zynq UltraScale+ MPSoC Register Reference (UG1087) [Ref 4]*. Access to registers is possible only by using the respective aperture addresses programmed after completing the bridge initialization and enabling the bridge translation.

2. Setup the AXI-PCIe bridge register apertures for bridge internal registers, ECAM aperture, and DMA register access.
   a. Map the bridge register aperture.

      AXIPCIE_MAIN.E_BREG_CONTROL[breg_size] = 4KB - 32KB
      AXIPCIE_MAIN.E_BREG_BASE_LO = `0xFD0E0000`
      AXIPCIE_MAIN.E_BREG_BASE_HI = `0x00000000`

   b. Map the ECAM space.

      AXIPCIE_MAIN.E_ECAM_CONTROL[ecam_size] = 4 KB (when Endpoint); 16 MB (when Root Port)
      AXIPCIE_MAIN.E_ECAM_BASE_LO = `0xE0000000`
      AXIPCIE_MAIN.E_ECAM_BASE_HI = `0x00000000`
      AXIPCIE_MAIN.E_ECAM_CONTROL[ecam_enable] = `1`

   Note: In the Root Port mode, this address can also be mapped to 40/44-bit space based on your requirement. In such cases care should be taken to program both ECAM_BASE_HI and ECAM_BASE_LO addresses appropriately.

   c. Map DMA register aperture

      AXIPCIE_MAIN.E_DREG_CONTROL[dma_size] = 0B
      AXIPCIE_MAIN.E_DREG_BASE_LO = `0xFD0F0000`
      AXIPCIE_MAIN.E_DREG_BASE_HI = `0x00000000`

   Note: Enable DMA register access, if Endpoint application is going to exercise DMA AXIPCIE_MAIN.E_DREG_CONTROL[dma_enable] = 1 (only for Endpoint mode).

3. For the Root Port mode.

   a. Setup ingress MSI aperture.

      AXIPCIE_MAIN.I_MSII_BASE_LO = `0xFE440000`
      AXIPCIE_MAIN.I_MSII_CONTROL[i_msii_enable] = `1`

   Note: An address assigned by the software driver can also be used.

   b. Disable DMA register access from Endpoint as there is no BAR in Root mapped to local registers.

      AXIPCIE_MAIN.BRIDGE_CORE_CFG_PCIE_RX0[cfg_dma_reg_bar] = `7` (disabled)
      AXIPCIE_MAIN.BRIDGE_CORE_CFG_PCIE_RX0[cfg_disable_pcie_dma_reg_access] = `1`

   c. Allow all upstream transactions (memory read, write) to access the AXI interface without any translation by enabling ingress subtractive decode.

      AXIPCIE_MAIN.I_ISUB_CONTROL = `0x01`

4. Enable translation apertures in the bridge for access by the AXI processor (i.e., after this access to bridge registers, the access is possible only by the use of the address programmed in AXIPCIE_MAIN.{E_BREG_BASE_HI, E_BREG_BASE_LO} and the DMA registers can be accessed by use of address programmed in AXIPCIE_MAIN.{E_DREG_BASE_HI, E_DREG_BASE_LO} and so on.

   AXIPCIE_MAIN.E_BREG_CONTROL[breg_enable] = `1`
   AXIPCIE_MAIN.E_BREG_CONTROL[breg_enable_force] = `0`

5. If using the Root Port mode, release pcie_reset_n from the MIO/GPIO programming registers. For Endpoint mode, wait for pcie_reset_n to be released by the host.

After the release of reset, link training is done with the peer and a link is established.

In the Endpoint mode, the host programs the PCIe configuration space registers inside the integrated block for PCIe.

Programmed I/O Transfers

This section describes setting up programmed I/O (PIO) transfers (both ingress and egress) in Endpoint mode.

Ingress Transfers

Ingress refers to PCIe in the AXI direction. For PIO transfers from the host system to be accepted by the Endpoint, they have to hit the Endpoint BAR.

Since the host system is unaware of the AXI domain address on the Endpoint, an ingress translation is setup to map incoming BAR-hit transactions to AXI transactions. Before setting up the ingress translation aperture, the Endpoint software performs a handshake with the software driver running on the host system.

This example demonstrates a handshake using software interrupts and the scratchpad in DMA registers. A typical flow is illustrated in Figure 30-10.

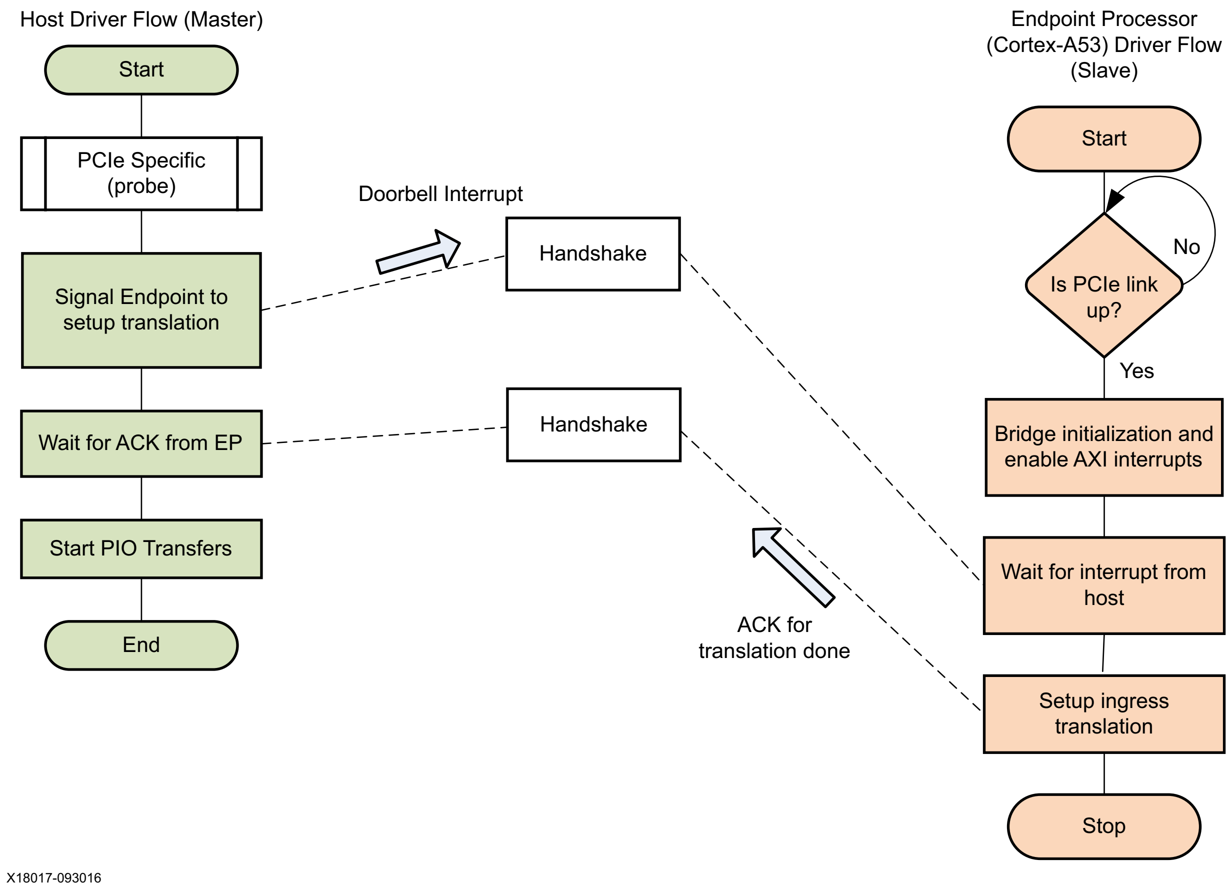

Host Driver Flow (Master)
Start
PCIe Specific (probe)
Signal Endpoint to setup translation
Wait for ACK from EP
Start PIO Transfers
End
Doorbell Interrupt
Handshake
Handshake
ACK for translation done
Endpoint Processor (Cortex-A53) Driver Flow (Slave)
Start
Is PCIe link up?
No
Yes
Bridge initialization and enable AXI interrupts
Wait for interrupt from host
Setup ingress translation
Stop
X18017-093016

Figure 30-10: **Ingress PIO Transfers Flow Chart**

Driver on a Zynq UltraScale+ MPSoC Endpoint

1. Checks for PCIe link up before proceeding ahead with any other initialization.
2. Performs bridge initialization as described in Bridge Initialization.
3. Enables AXI interrupts by programming:

   AXIPCIE_DMA0.DMA_CHANNEL_AXI_INTERRUPT_CONTROL[interrupt enable] = 1

   AXIPCIE_MAIN.MSGF_DMA_MASK = `0x1`

   This example only uses the DMA channel 0 interrupt.

4. Wait for interrupt from host before setting up ingress translation aperture.
5. Once the interrupt (AXIPCIE_DMA0.DMA_CHANNEL_AXI_INTERRUPT_STATUS[software_int]) is received:
   a. Read BAR2 or BAR4 via the ECAM aperture to obtain the BAR value host programmed in the Endpoint configuration space (BAR0 is dedicated to the bridge).
   b. Setup the ingress source address to the BAR address and destination address to the desired destination address; program the ingress aperture size and enable the translation.

      Note: The handshake can also be implemented using a poll mode where the driver polls for a predefined signature value in the scratchpad registers of the DMA channel. Instead of reading the BAR address via ECAM and programming, the host system driver can pass the address and size via scratchpad registers. There are different ways of implementing this and one options is described. The memory enable bit in the command register in the PCIe configuration space can also be used to validate the BAR assignment.
6. Once translation is setup and to inform the host system, raise an interrupt to the host (AXIPCIE_DMA0.DMA_CHANNEL_PCIE_INTERRUPT_ASSERT[pcie_software_interrupt]) or write a signature value to the scratchpad register.

Driver on Host System

1. Does required a PCIe device specific probe?
2. A doorbell interrupt to the Endpoint is raised though the AXIPCIE_DMA0.DMA_CHANNEL_AXI_INTERRUPT_ASSERT[axi_software_interrupt] register and wait for an acknowledgment from the Endpoint.
3. On reception of an acknowledgment (interrupt or polling based), PIO transfers to write to a BAR mapped space are started and read back for verification.

Egress Transfers

Egress refers to the AXI to PCIe direction. Typically, these transfers are achieved using the DMA however, egress transactions provide a way for an Endpoint to drive 4-byte transfers into the host system memory without using a DMA. This requires bus mastering to be enabled for the Endpoint. Additionally, egress translation aperture should be setup. For this, host system driver allocates memory and physical address of this memory is communicated to the Endpoint, which becomes the egress translation destination address. For egress source address, the address must fall within the PCIe address range as defined in Chapter 10, System Addresses. This is 256 MB in the 4G address space and 8 GB and 256 GB for higher address widths. A typical flow is illustrated in Figure 30-11.

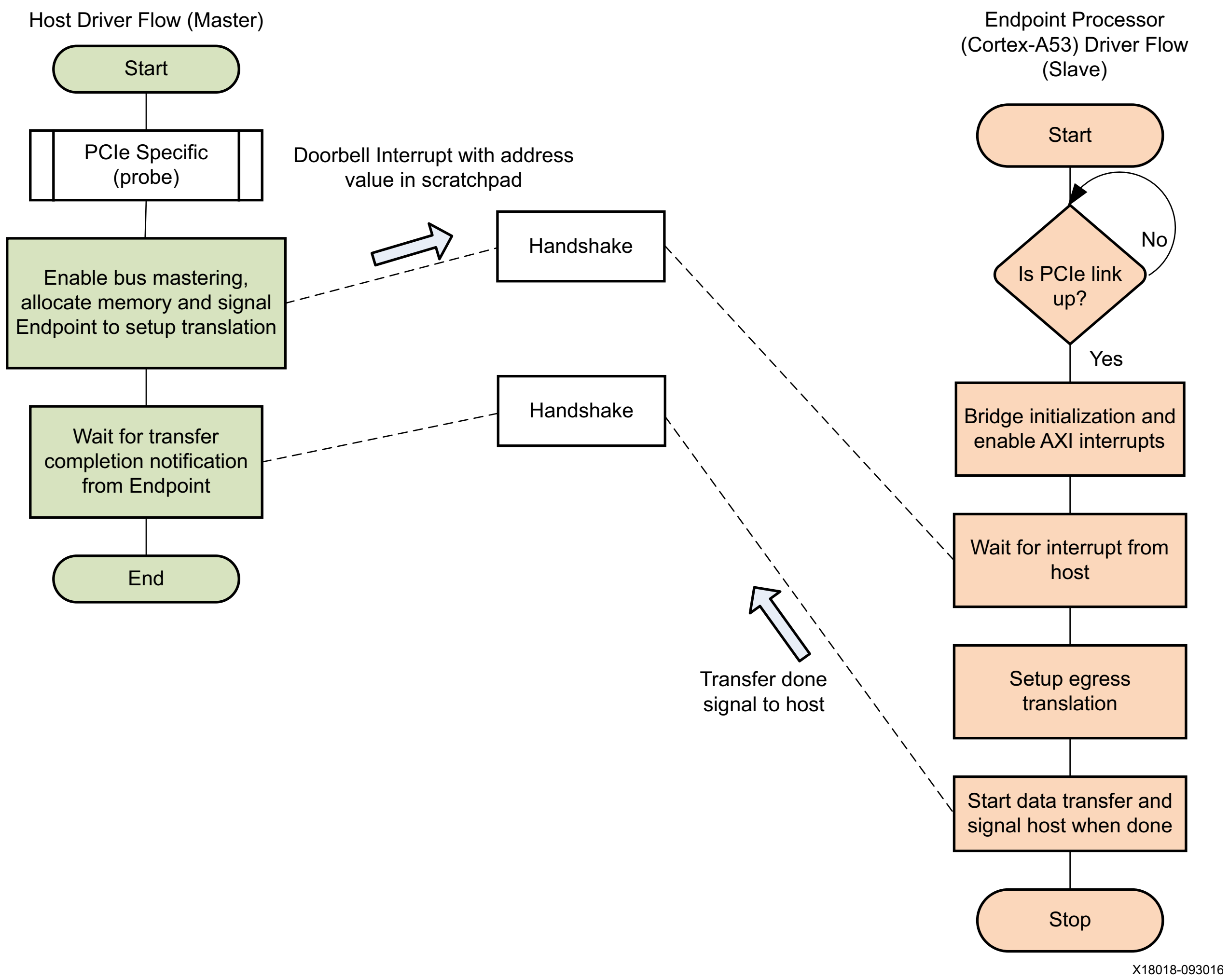

Host Driver Flow (Master)
Start
PCIe Specific (probe)
Enable bus mastering, allocate memory and signal Endpoint to setup translation
Wait for transfer completion notification from Endpoint
End
Doorbell Interrupt with address value in scratchpad
Handshake
Handshake
Transfer done signal to host
Endpoint Processor (Cortex-A53) Driver Flow (Slave)
Start
Is PCIe link up?
No
Yes
Bridge initialization and enable AXI interrupts
Wait for interrupt from host
Setup egress translation
Start data transfer and signal host when done
Stop
X18018-093016

Figure 30-11: **Egress Transfer Flow Chart**

Egress Host Driver

1. Does the host driver require a PCIe device specific probe?
2. Enables bus mastering for the Endpoint and allocates the memory region to accept data from the Endpoint.
3. Conveys the physical address of the memory to the Endpoint through the scratchpad registers using doorbell interrupts.
4. Waits for a transfer completion signal from the Endpoint to consume data.

Egress Endpoint Driver

1. Checks for PCIe link up before proceeding ahead with any other initialization.
2. Performs bridge initialization as described in Bridge Initialization.
3. Enables AXI interrupts by programming:

   AXIPCIE_DMA0.DMA_CHANNEL_AXI_INTERRUPT_CONTROL[interrupt enable] = 1

   AXIPCIE_MAIN.MSGF_DMA_MASK = `0x1`

   This example only uses the DMA channel 0 interrupt.
4. Wait for interrupt from host before setting up ingress translation aperture.
5. Once the interrupt (AXIPCIE_DMA0.DMA_CHANNEL_AXI_INTERRUPT_STATUS[software_int]) is received:
   a. Read the scratchpad to obtain the host system memory address allocated for egress transfers.
   b. Setup the egress source address to AXI address (falling in the PCIe address domain) and destination address to the received host system memory address; program the egress aperture size and enable the translation.

Perform data transfers and signal the host system on completion to further process transferred by host system software.

Endpoint Mode DMA Operation

This section describes the DMA operation when the Zynq UltraScale+ MPSoC controller for PCIe is used as an Endpoint. The dual CPU mode is outlined, where a single DMA channel is managed by both source and sink processors. The AXI CPU and PCIe CPU can each become source or sink based on the direction of data transfer.

The DMA Operation section describes the dataflow for this mode. Once the core is configured for Endpoint mode of operation, the DMA activity can be divided into three phases.

- Initialization Phase: initializes the DMA channel, sets up the descriptor elements.
- DMA Phase: the DMA operation phase.
- Exit Phase: the DMA transaction completes and the allocated resources are released in this phase.

Handshake between Host and AXI-CPU Driver

In the example in Figure 30-12, the driver in the host is considered to be the master. It can enable/disable/reset a DMA channel. Hence, a handshake is required between the host software and AXI-CPU software to indicate DMA reset/channel enable/disable events. The scratch pad registers available per DMA channel can be used for communication of a handshake event with signature values written to convey the meaning. This is typically based on a predefined messaging protocol between the two software drivers.

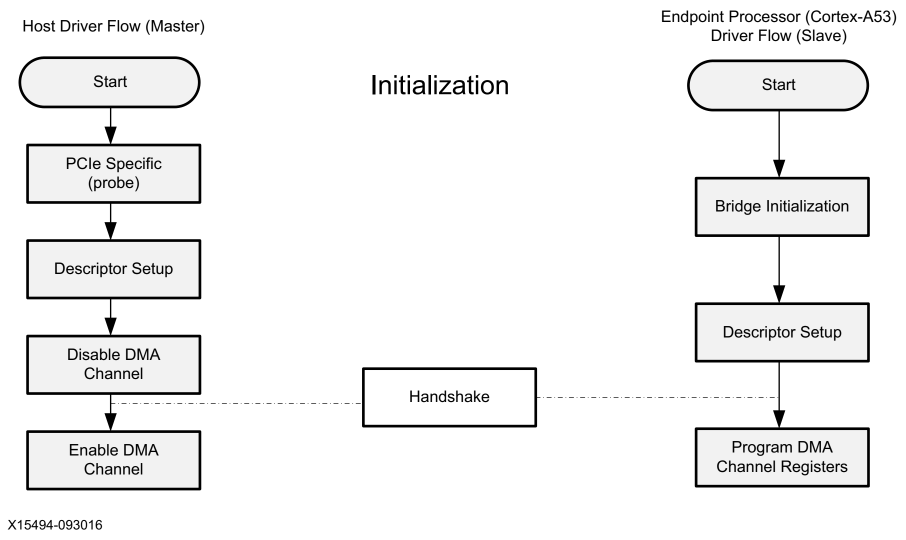


Figure 30-12: **Initialization Phase**

Descriptor Setup

The SRC and DST SGL elements need to be setup based on the required transfer flow.

For example, for the system to card transfers (host to AXI-CPU).

- Setup appropriate flags in the SRC element on the host (buffer fetch direction PCIe and interrupt if required on the EOP element).
- Setup appropriate flags in the DST element on the AXI-CPU (buffer write direction AXI and back-to-back packing of data if needed).
- Setup appropriate flags for elements in a card-to-system (C2S) transfer.

Sequence for Enabling DMA Channel

The following DMA channel enable sequence is recommended. The process can be completed in any order, provided the DMA enable value is set to `1` in the end.

Note: Prior to enabling the DMA channel, the source scatter-gather queue, the destination scatter-gather queue, the DMA source completion status queue, and the DMA destination completion status queue must be initialized.

1. Verify that the DMA channel is idle.

   Read the DMA running and verify it reads `0x0`. If a non-0, follow the instructions provided in Disabling an Active DMA Channel.

2. Initialize the queue base address and attributes.
   a. Write SRC_Q_PTR_LO and SRC_Q_PTR_HI with the base address of the queue.
   b. Write DST_Q_PTR_LO and DST_Q_PTR_HI with the base address of the queue.
   c. Write STAS_Q_PTR_LO and STAS_Q_PTR_HI with the base address of the queue.
   d. Write STAD_Q_PTR_LO and STAD_Q_PTR_HI with the base address of the queue.
3. Initialize the queue size.
   a. Write SRC_Q_SIZE to the size of the queue.
   b. Write DST_Q_SIZE to the size of the queue.
   c. Write STAS_Q_SIZE to the size of the queue.
   d. Write STAD_Q_SIZE to the size of the queue.

4. Initialize queue Next pointers to the beginning of the queue.
   a. Write SRC_Q_NEXT = `0x0`.
   b. Write DST_Q_NEXT = `0x0`.
   c. Write STAS_Q_NEXT = `0x0`.
   d. Write STAD_Q_NEXT = `0x0`.
5. Initialize scatter-gather queues to the empty condition (no DMA operations to execute).
   a. Write SRC_Q_LIMIT = `0x0`.
   b. Write DST_Q_LIMIT = `0x0`.
6. Initialize status queues to the fully available condition (all status queue elements except one, which is needed to preserve software flow control, are available).
   a. STAS_Q_LIMIT set to (STAS_Q_SIZE-1).
   b. STAD_Q_LIMIT set to (STAD_Q_SIZE-1).
7. Initialize all STAS and STAD queue elements to `0x0`.

   DMA completion status queue elements must be initialized to `0x0` so that the software can specify the completion of status elements. Status elements return a non-0 value when complete.

8. Optionally, initialize all the SRC and DST scatter-gather queue elements.

   The initialization of source and destination SGL elements is solely at the discretion of software. Source and destination SGL elements will not be fetched until they have been filled in with DMA transaction instructions and the associated queue's LIMIT pointer advanced to give these elements to the DMA channel to execute.

9. Write DMA enable = 1 to enable the DMA channel.

Optionally, for interrupt mode, the following registers are programmed to enable the interrupts.

- AXIPCIE_DMA.DMA_CHANNEL_PCIE_INTERRUPT_CONTROL for enabling interrupts in the PCIe domain.
- AXIPCIE_DMA.DMA_CHANNEL_AXI_INTERRUPT_CONTROL for enabling interrupts in the AXI domain.

Interrupts for various events can be enabled, SGL completion (EOP) or error. Additionally, coalesce count can be set to control the frequency of interrupt generation.

IMPORTANT: *DMA channel MSI-X and MSI interrupts are signaled using the MSI-X/MSI interrupt vector corresponding to their DMA channel number. For example, DMA channel[0] interrupts are signaled on MSI-X/MSI vector 0, DMA channel[1] interrupts are signaled on MSI-X/MSI vector 1, etc.*

DMA Operation

DMA channel executes the DMA transactions by writing the source SGL and the destination SGL into the SRC/DST SGL queues and incrementing the SRC/DST_Q_LIMIT registers (Figure 30-13).

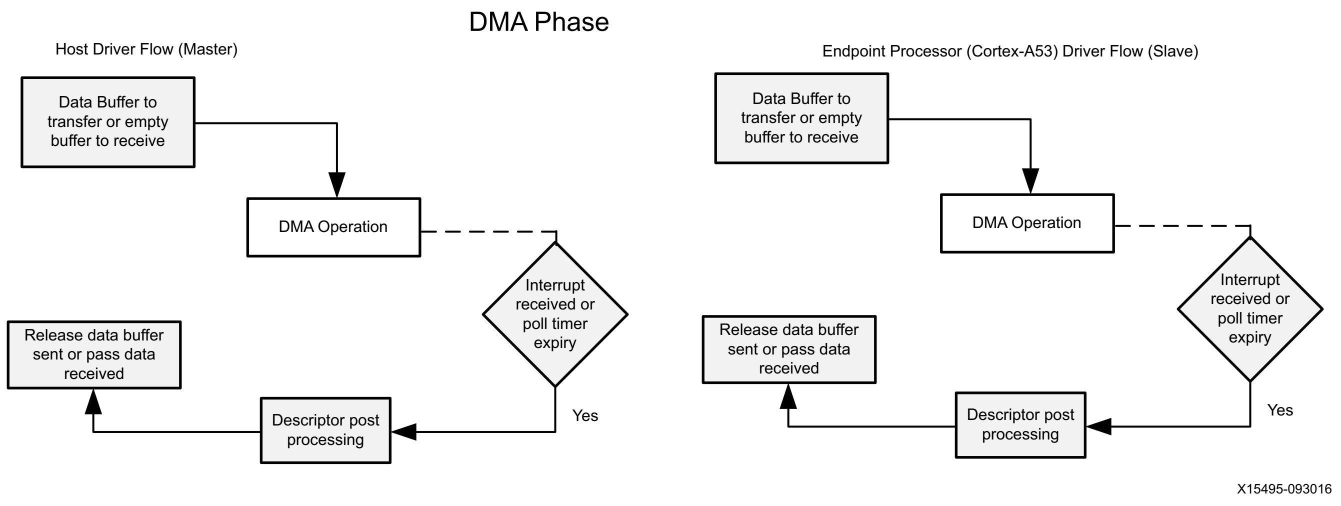

DMA Phase
Host Driver Flow (Master)
Data Buffer to transfer or empty buffer to receive
DMA Operation
Interrupt received or poll timer expiry
Yes
Descriptor post processing
Release data buffer sent or pass data received
Endpoint Processor (Cortex-A53) Driver Flow (Slave)
Data Buffer to transfer or empty buffer to receive
DMA Operation
Interrupt received or poll timer expiry
Yes
Descriptor post processing
Release data buffer sent or pass data received
X15495-093016

Figure 30-13: **DMA Phase**

Note: Queue management registers *_PTR_LO, *_PTR_HI, *_SIZE, and *_NEXT must not be written when the DMA channel is enabled. It is permissible to modify only the SRC/DST/STAS/STAD_Q_LIMIT queue registers, while the DMA channel is enabled. Increment these registers to provide additional elements for the DMA channel to execute.

IMPORTANT: *The minimum queue size must be large enough to hold at least one full DMA transaction of maximum size. DMA completion status queues are only written when a source SGL element is completed that had its EOP flag == 1 (end of a DMA transaction). If the queue is too small to be able to place all of the SGL for a single DMA transaction in the queue, then the SGL with EOP == 1 is not added to the queue and the DMA operation will not complete. In such a case, the software is not able to free queue elements and no new SGL can be given to the DMA channel unless the queue elements are freed.*

A queue size of N has N queue elements: [0],[1],...,[N-1]. For example, a queue size of 2 has [0] and [1] elements.

The queue wraps at the N-1 element. For example, for a queue size of 2 the wrap occurs as: [0], [1], [0],...

RECOMMENDED: *The DMA queues are intended to be initially setup and reused for multiple DMA operations. The DMA queues are designed to enable highly overlapped transactions. Software can setup new DMA operations in the queue while the DMA channel is executing operations that the software placed in the queue earlier.*

The DMA queues can also be setup for each DMA transaction, although this method provides lower performance because the queues must be reconfigured between DMA transactions. DMA queues can only be reconfigured when the DMA channel is disabled. The steps to setup the DMA queues are listed.

1. Wait for all outstanding DMA transactions for the channel to complete.
2. Disable the DMA channel.
3. Reconfigure the queues.
4. Re-enable the DMA channel.

Descriptor Post-processing

When a DMA transaction completes (software reads the current DMA completion status queue element and reads status == complete), software processes the resulting DMA data and recycles the source/destination SGL queue elements associated with the transfer as well as the associated DMA source/destination completion status queue elements.

Note: Status queue elements must be written to `0x0` when recycled because software uses a read of non-0 for a status queue element as indication that a DMA transfer is completed. Recycled queue elements are reused when the queue LIMIT pointer wraps back to the position of the recycled elements.

Disabling an Active DMA Channel

The preferred process to disable an operational DMA channel (Figure 30-14) is as follows.

1. Software stops adding new DMA transfers to the source and destination SGL queues. The last source and destination SGL queue elements given to the DMA channel to execute (via SRC/DST_Q_LIMIT registers) should be fully consumed by the final DMA transfer.
2. Software waits for all outstanding DMA operations to complete and processes the DMA completion status from the DMA completion status queue. Software implements a timeout in the event that the DMA operations are never complete. This can occur if software is not provided with the matching source and destination SGL elements that can be fully consumed.
3. Software writes DMA_Enable == 0 to the DMA channel. This disables the DMA channel.
4. Software reads the DMA_Running in the DMA channel. If DMA_Running == 0, then the DMA channel is finished with all the outstanding transactions. The software can optionally skip to step 6.
5. If DMA_Running == 1, then the DMA channel could not have finished all the outstanding transactions. One or more of its internal source SGL, destination SGL, or DMA completion status queues is not empty. Software writes DMA_Reset == 1, waits $\geq$256 nS, and writes DMA_Reset == 0. Setting DMA_Reset == 1 flushes the internal DMA source SGL, destination SGL, and DMA completion status FIFOs.

6. The DMA channel is now ready for reuse.

You can use these steps when you use a single CPU mode, where a host driver manages the DMA channel. The driver on Arm will only be responsible for bridge initialization.

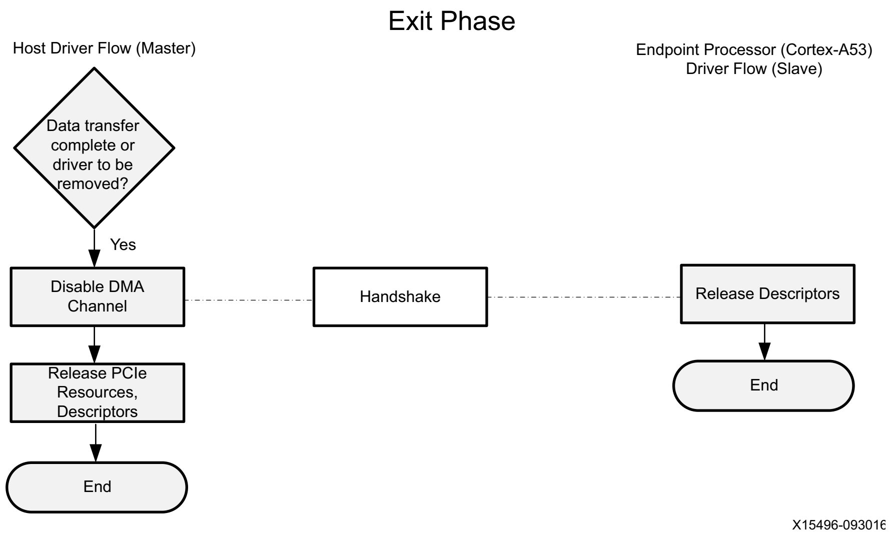

Exit Phase
Host Driver Flow (Master)
Endpoint Processor (Cortex-A53) Driver Flow (Slave)
Data transfer complete or driver to be removed?
Yes
Disable DMA Channel
Handshake
Release Descriptors
Release PCIe Resources, Descriptors
End
End
X15496-093016

Figure 30-14: **Exit Phase**

USB Controller

Introduction

The USB 3.0 controller consists of two independent dual-role device (DRD) controllers. Both can be individually configured to work as host or device at any given time. The USB 3.0 DRD controller provides an eXtensible host controller interface (xHCI) to the system software through the advanced eXtensible interface (AXI) slave interface. An internal DMA engine is present in the controller and it utilizes the AXI master interface to transfer data. The three dual-port RAM configurations implement the RX data FIFO, TX data FIFO, and descriptor/register cache. The AXI master port and the protocol Layers access the different RAMs through the buffer management unit.

USB 2.0/3.0 Controller Details

Each instance of the controller supports the configurations in Table 31-1.

Table 31-1: **USB Controller Configurations**

Configuration	PHY Interface	Super Speed	High Speed	Full Speed	Low Speed
USB 2.0 host	ULPI		Yes	Yes	Yes
USB 2.0 device	ULPI		Yes	Yes	Yes
USB 2.0 OTG	ULPI		Yes	Yes	Yes
USB 3.0 host (xHCI)	PIPE3	Yes	Yes	Yes	Yes
USB 3.0 device	PIPE3	Yes	Yes	Yes	

USB Controller Features

- Two USB 2.0/3.0 controllers
- Supports a 5.0 Gb/s data rate
- Supports host and device modes
- Supports on-the-go (OTG) host/device selection for USB 2.0 (only)
- Provides simultaneous operation of the USB 2.0 and USB 3.0 interfaces (only in host 3.0 mode).
- 64-bit AXI master port with built-in DMA
- AXI port for register programming
- Power management features: hibernation mode
- Support for 48-bit address space
- Supports 12 endpoints (six out and six in)

Note: When the USB controller is used in a 3.0 configuration, USB 2.0 mode must also be enabled in the MPSoC processor configuration window (PCW). This is necessary because the DC voltage bus (VBUS) valid signal from the ULPI interface PHY is used. Consequently, it is mandatory to enable the USB 2.0 mode though the USB 3.0 mode is required irrespective of the host, device, or OTG modes.

PHY Loopback

The USB 3.0 host and device modes support PHY loopback. When the host/device with PHY connects to a tester during the polling state, the tester sends a TS2 ordered set with the loopback bit enabled. The host/device changes the link state from polling to loopback and asserts USB3_{0:1}_XHCI.TxDetRxLoopback.

Next, the tester sends TX data to the host/device, and the host/device PHY decodes the data and sends it back. There is no programming involved. When the tester finishes loopback testing and is ready to exit loopback mode, it performs a U2 or loopback (LFPS handshake) exit. See the USB 3.0 specification for more information.

Figure 31-1 shows a high-level diagram of the Zynq UltraScale+ MPSoC USB 3.0 block.

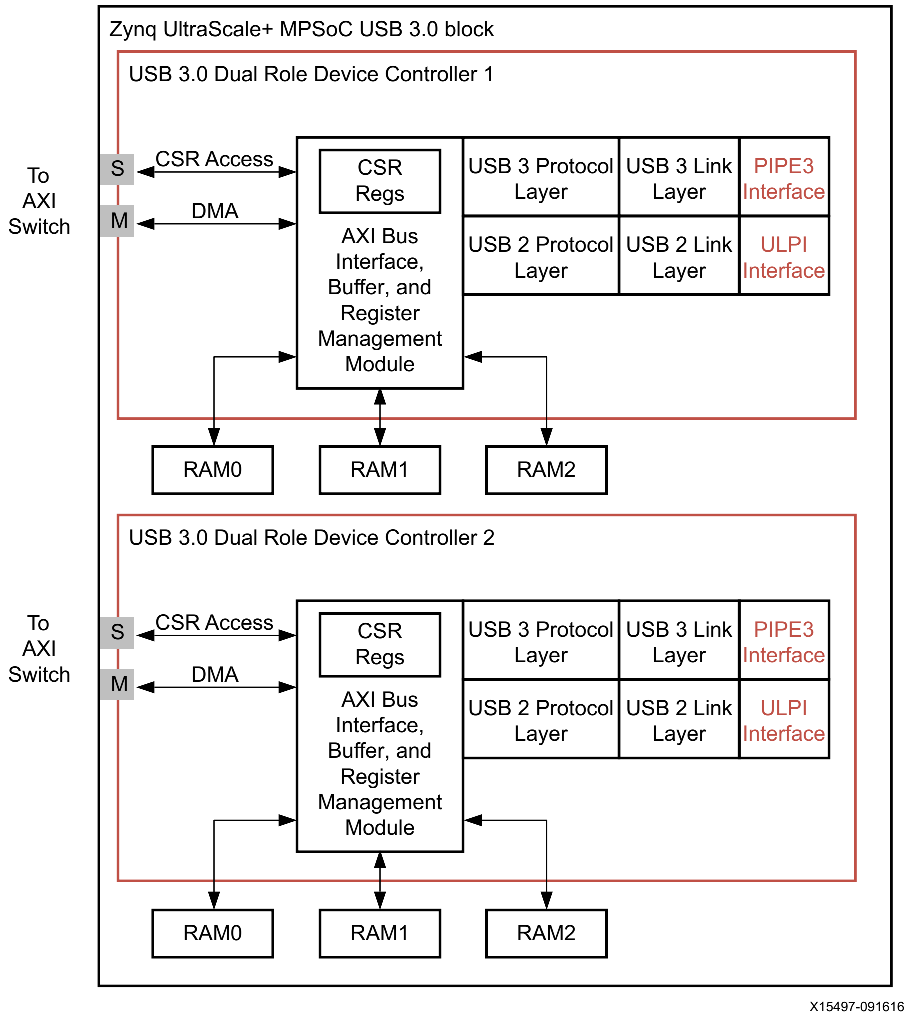


Figure 31-1: **Zynq UltraScale+ MPSoC USB 3.0 Block Diagram**

The controller can be visualized as software and embedded blocks. The embedded partition consists of USB 3.0 host/device and the associated PHY interfaces. Software drivers for host or device that connect the controller to the USB peripheral stack are available from either third-party or open source vendors.

The DC voltage bus (VBUS) can be controlled using PL signals only in non-OTG mode.

- U2dsport_vbus_ctrl for USB 2.0
- U3dsport_vbus_ctrl for USB 3.0

These signals are only used for non-OTG mode. There is no VBUS control port signal in USB 2.0 OTG mode with a ULPI interface. The VBUS control signal comes from the command.

Note: Zynq UltraScale+ USB controller does not generate an external USB2.0 ULPI PHY RESET (ULPI_PHY_RESETB) signal. With Processor configuration wizard (PCW), signals can be generated with a reset logic. Please refer for more details in the Zynq UltraScale+ MPSoC Processing System Product Guide (PG201) for details.

Data Flow

To operate the USB controller in various modes, 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 USB 3.0 controller. The data structures support 32-bit or 64-bit memory buffer address space.

The basic data structures are shown in Table 31-2.

Table 31-2: **Basic Data Structures**

Data Structure	Max Size (Bytes)	Boundary	Alignment (Bytes)
Context Data Structures			
Device context	2048	PAGESIZE	64
Device context data structure contains slot context and endpoint contexts (up to 32). An array of device contexts is prepared and maintained by the xHCI embedded block 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 utilized by the xHCI scratchpad mechanism.			
Slot context	64	PAGESIZE	32
The slot context data structure defines 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.			
Endpoint context	64	PAGESIZE	32
The endpoint context data structure defines information that applies to a specific endpoint			
Stream context	16	PAGESIZE	16
This data structure defines information that applies to a specific stream associated with an endpoint.			
Input context	132	PAGESIZE	64
The input context data structure specifies the 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
The input control context data structure defines which 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
The port bandwidth context data structure is used to provide system 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.			

Table 31-2: **Basic Data Structures *(Cont'd)***

Data Structure	Max Size (Bytes)	Boundary	Alignment (Bytes)
Ring Data Structures			
Transfer ring segments	64 KB	64 KB	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 KB	64 KB	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 KB	64 KB	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 KB	None	64
This is a table of event ring segments.			
Scratchpad buffers	PAGESIZE	PAGESIZE	Page
A scratchpad buffer is a PAGESIZE block of system memory located on a PAGESIZE boundary. Each of these buffers allocated from system memory for storing internal state.			

Data Structure Network

The data structures in the xHCI are linked as shown in Figure 31-2.

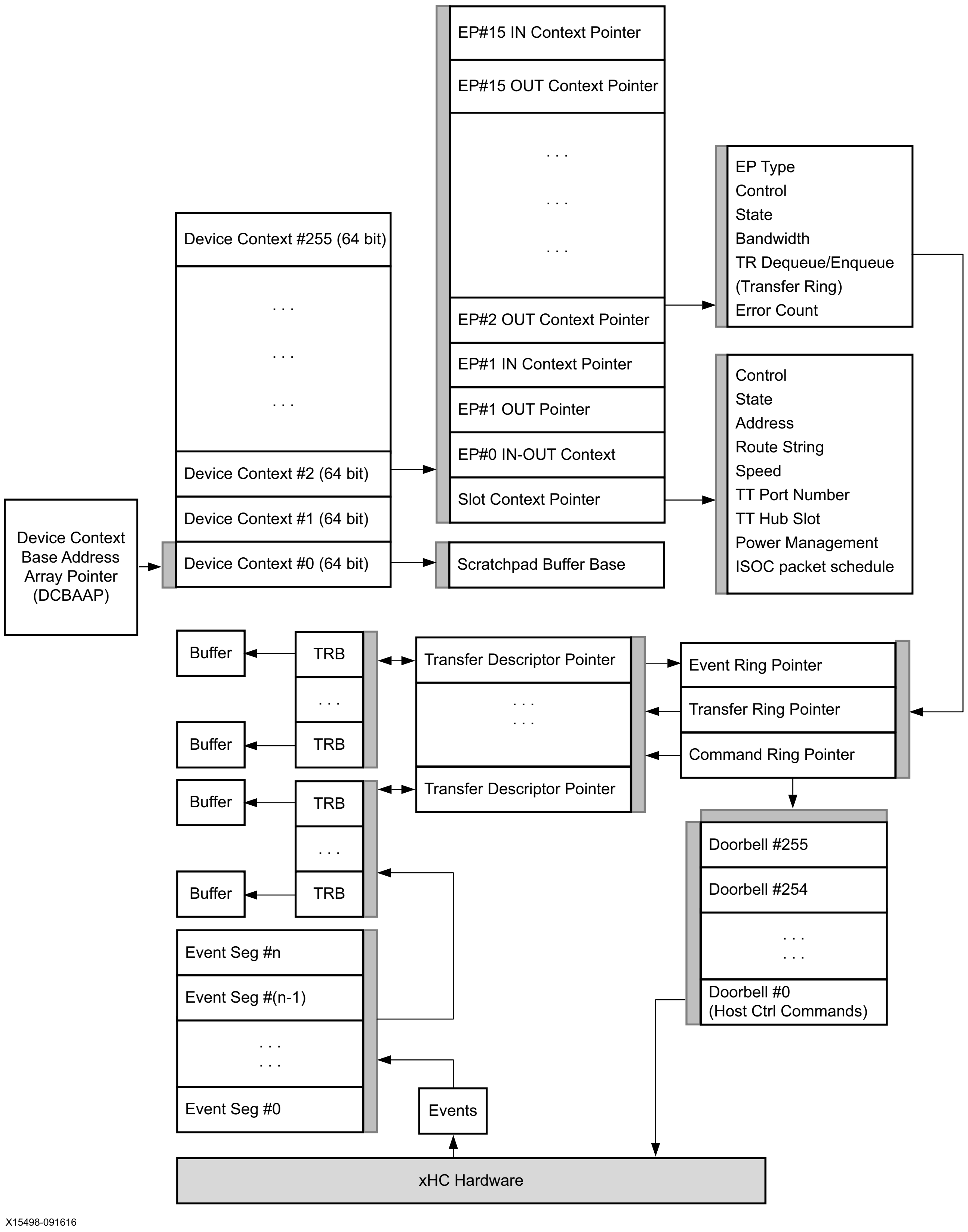


Figure 31-2: **Network of Software Data Structures**

Data Structure Network

The data structures are detailed in this section.

Device Context Data Structure

Figure 31-3 refers to the slot context data structure where each instance of this structure represents a device connected to the host.

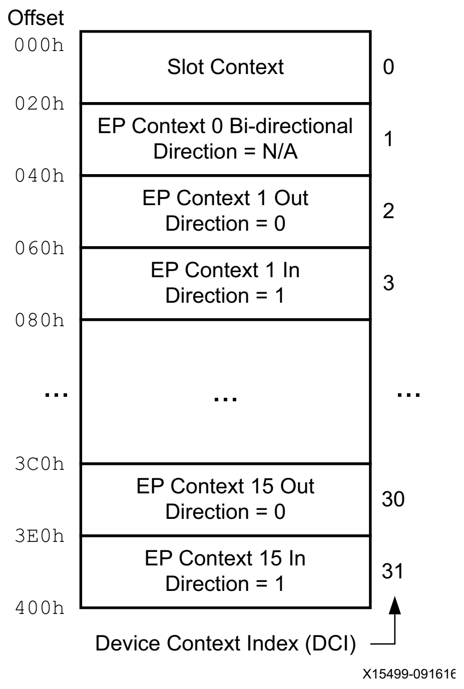

Offset
000h
020h
040h
060h
080h
...
3C0h
3E0h
400h
Slot Context
EP Context 0 Bi-directional Direction = N/A
EP Context 1 Out Direction = 0
EP Context 1 In Direction = 1
...
EP Context 15 Out Direction = 0
EP Context 15 In Direction = 1
0
1
2
3
...
30
31
Device Context Index (DCI)
X15499-091616

Figure 31-3: **Device Context Data Structure**

Slot Context Data Structure and State Diagram

Figure 31-4 and Table 31-3 refer to the slot context state transfers and how a host treats each state of the device connected to the bus.

Table 31-3: **Slot Context Data Structure**

31 27	26	25	24	23 22	21 20	19 18	17 16	15 8	7 0	
Context Entries	Hub	MTT	RsvdZ	Speed		Route String				03-00H
Number of Ports				Root Hub Port Number				Maximum Exit Latency		07-04H
Interrupter Target					RsvdZ		TT	TT Port Number	TT Hub Slot ID	0B-08H
Slot State	RsvdZ								USB Device Address	0F-0CH
xHCI Reserved (Rsvd0)										13-10H
xHCI Reserved (Rsvd0)										17-14H
xHCI Reserved (Rsvd0)										1B-18H
xHCI Reserved (Rsvd0)										1F-1CH

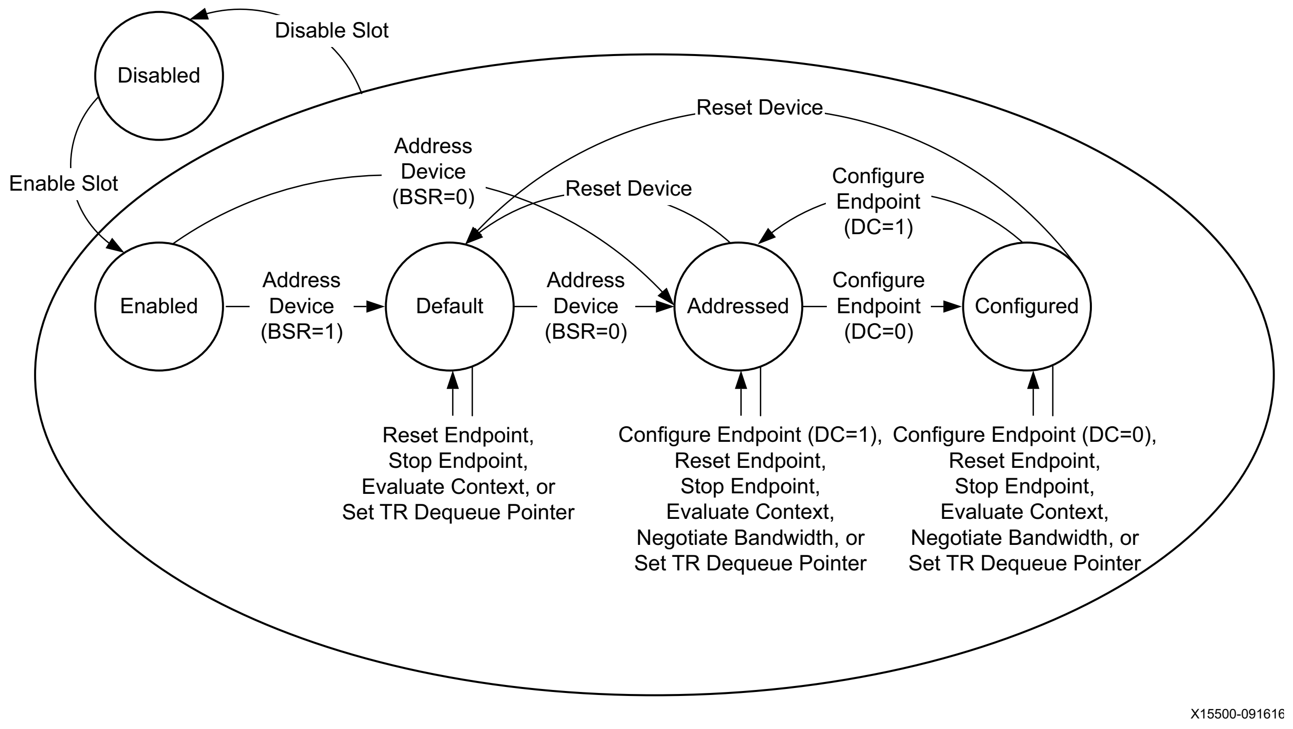

Disabled
Disable Slot
Enable Slot
Reset Device
Address Device (BSR=0)
Reset Device
Configure Endpoint (DC=1)
Enabled
Address Device (BSR=1)
Default
Address Device (BSR=0)
Addressed
Configure Endpoint (DC=0)
Configured
Reset Endpoint, Stop Endpoint, Evaluate Context, or Set TR Dequeue Pointer
Configure Endpoint (DC=1), Reset Endpoint, Stop Endpoint, Evaluate Context, Negotiate Bandwidth, or Set TR Dequeue Pointer
Configure Endpoint (DC=0), Reset Endpoint, Stop Endpoint, Evaluate Context, Negotiate Bandwidth, or Set TR Dequeue Pointer
X15500-091616

Figure 31-4: **Slot Context State Machine**

Endpoint Context Data Structure and State Diagram

Table 31-4 refers to the Endpoint data structure that contains information for each Endpoint in a device. Figure 31-5 represents the Endpoint state machine and error handling.

Table 31-4: **Endpoint (EP) Data Structure**

31 24	23 16	15	14 10	9 8	7	6	5 4	3	2 1	0	
Maximum ESIT Payload High	Interval	LSA	MaxPStreams	Multi	RsvdZ				EP State		03-00H
Maximum Packet Size		Maximum Burst Size			HID	RsvdZ	EP Type		CERR	RsvdZ	07-04H
Transfer Ring Dequeue Pointer Low								RsvdZ		DCS	0B-08H
Transfer Ring Dequeue Pointer High											0F-0CH
Maximum ESIT Payload Low		Average TRB Length									13-10H
xHCI Reserved (Rsvd0)											17-14H
xHCI Reserved (Rsvd0)											1B-18H
xHCI Reserved (Rsvd0)											1F-1CH

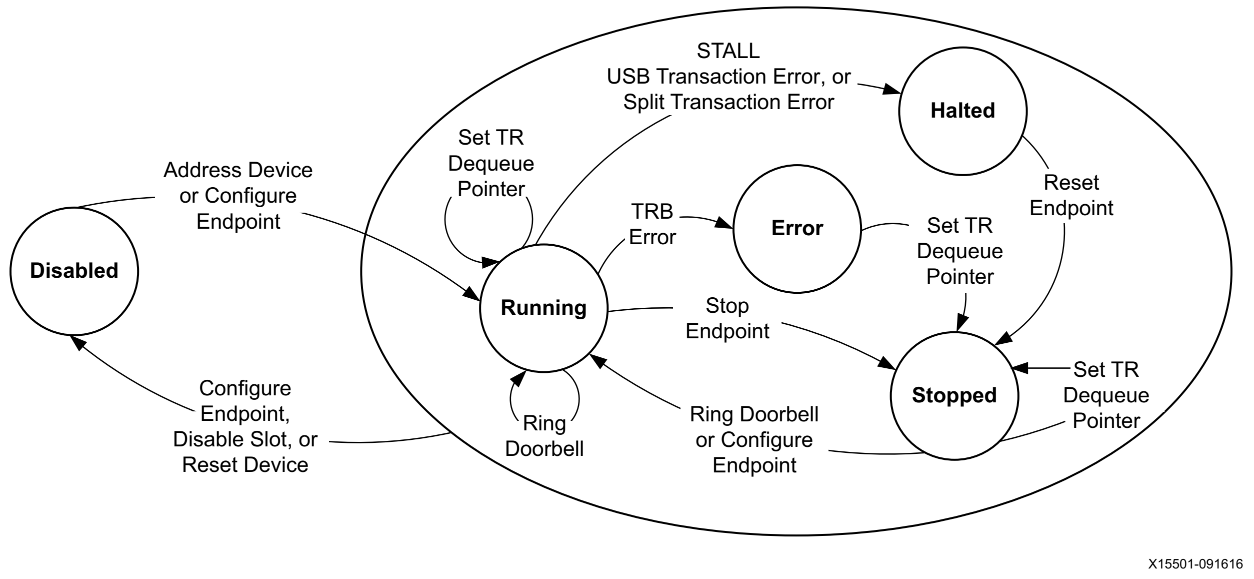


Figure 31-5: **Endpoint Structure State Machine**

The interface consists of transfer request buffers (TRBs) that are managed in TRB rings. All transfer types (ISOC, interrupt, Control, Bulk, also command, events) use the same basic TRB structure. TRBs also support scatter/gather operations. Any transfer (command, data, event) between the software and the controller are moved as blocks of data. These blocks are called transfer request blocks (also TRBs). Based on the type of data movement, the specification defines various TRBs.

Figure 31-6 represents the formation of transfer rings from an atomic buffer level.

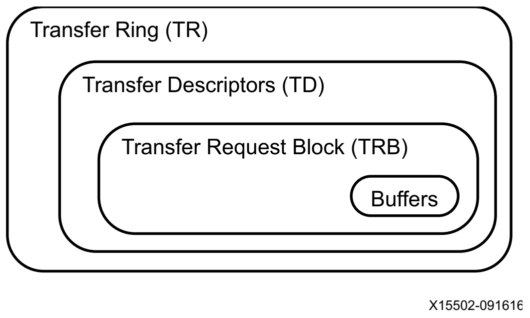


Figure 31-6: **Formation of Transfer Rings**

The Table 31-5 to Table 31-34 represent the various TRBs.

Transfer TRBs

Normal TRB

Table 31-5: **Normal TRB Data Structure**

31 22	21 17	16	15 10	9	8 7	6	5	4	3	2	1	0	
Data Buffer Pointer Low													03-00H
Data Buffer Pointer High													07-04H
Interrupter Target	TD Size	TRB Transfer Length											0B-08H
RsvdZ			TRB Type	BEI	RsvdZ	IDT	IOC	CH	NS	ISP	ENT	C	0F-0CH

Control TRB: Setup Stage

Table 31-6: **Control TRB Setup Stage Data Structure**

31 22	21 18	17	16	15 10	9 8	7	6	5	4 1	0	
wValue				bRequest		bRequestType					03-00H
wLength				wIndex							07-04H
Interrupter Target	RsvdZ		TRB Transfer Length								0B-08H
RsvdZ		TRT		TRB Type	RsvdZ		IDT	IOC	RsvdZ	C	0F-0CH

Control TRB: Data Stage

Table 31-7: **Control TRB Data Stage Data Structure**

31 22	17	16	15 10	9 7	6	5	4	3	2	1	0	
Data Buffer Low												03-00H
Data Buffer High												07-04H
Interrupter Target	TD Size	TRB Transfer Length										0B-08H
RsvdZ		DIR	TRB Type	RsvdZ	IDT	IOC	CH	NS	ISP	ENT	C	0F-0CH

Control TRB: Status Stage

Table 31-8: **Control TRB Data Status Data Structure**

31 22	21 17	16	15 10	9 6	5	4	3 2	1	0	
RsvdZ										03-00H
RsvdZ										07-04H
Interrupter Target	RsvdZ									0B-08H
RsvdZ		DIR	TRB Type	RsvdZ	IOC	CH	RsvdZ	ENT	C	0F-0CH

ISOC TRB

Table 31-9: **ISOC TRB Data Structure**

31	30 29 22	21 20	19 17	16	15 10	9	8 7	6	5	4	3	2	1	0	
Data Buffer Pointer Low															03-00H
Data Buffer Pointer High															07-04H
Interrupter Target		TD Size		TRB Transfer Length											0B-08H
SIA	Frame ID		TLBPC		TRB Type	BEI	TBC	IDT	IOC	CH	NS	ISP	ENT	C	0F-0CH

NoOp TRB

Table 31-10: **NoOp TRB Data Structure**

31 22	21 17 16	15 10	9 6	5	4	3 2	1	0	
RsvdZ									03-00H
RsvdZ									07-04H
Interrupter Target	RsvdZ								0B-08H
RsvdZ		TRB Type	RsvdZ	IOC	CH	RsvdZ	ENT	C	0F-0CH

Event TRBs

Transfer Event TRB

Table 31-11: **Event TRB Data Structure**

31 24	23 20 16	15 10	9 3	2	1	0	
TRB Pointer Low							03-00H
TRB Pointer High							07-04H
Completion Code	TRB Transfer Length						0B-08H
Slot ID	VF ID	TRB Type	RsvdZ	ED	RsvdZ	C	0F-0CH

Command Completion Event TRB

Table 31-12: **Command Completion TRB Data Structure**

31 24	23 17 16	15 10	9 4	3 1	0	
Command TRB Pointer Low				RsvdZ		03-00H
Command TRB Pointer High						07-04H
Completion Code	Command Completion Parameter					0B-08H
Slot ID	VF ID	TRB Type	RsvdZ		C	0F-0CH

Port Status Change Event TRB

Table 31-13: **Port Status TRB Data Structure**

31 24	23 16	15 10	9 1	0	
Port ID	RsvdZ				03-00H
RsvdZ					07-04H
Completion Code	RsvdZ				0B-08H
RsvdZ		TRB Type	RsvdZ	C	0F-0CH

Bandwidth Request Event TRB

Table 31-14: **Bandwidth Request TRB Data Structure**

31 24	23 16	15 10	9 1	0	
RsvdZ					03-00H
RsvdZ					07-04H
Completion Code	RsvdZ				0B-08H
Slot ID	RsvdZ	TRB Type	RsvdZ	C	0F-0CH

Doorbell Event TRB

Table 31-15: **Doorbell TRB Data Structure**

31 – 24	23 – 16	15 – 9	9 – 5	4 – 1	0	
RsvdZ				DB Reason		03-00H
RsvdZ						07-04H
Completion Code	RsvdZ					0B-08H
Slot ID	VF ID	TRB Type	RsvdZ		C	0F-0CH

Host Controller Event TRB

Table 31-16: **Host Controller Event TRB Data Structure**

31 – 24	23 – 16	15 – 10	9 – 1	0	
RsvdZ					03-00H
RsvdZ					07-04H
Completion Code	RsvdZ				0B-08H
RsvdZ		TRB Type	RsvdZ	C	0F-0CH

Device Notification Event TRB

Table 31-17: **Device Notification TRB Data Structure**

31 – 24	23 – 16	15 – 10	9 – 8	7 – 4	3 – 1	0	
Device notification Data Low				Notification Type	RsvdZ		03-00H
Device notification Data High							07-04H
Completion Code	RsvdZ						0B-08H
Slot ID	RsvdZ	TRB Type	RsvdZ			C	0F-0CH

MFINDEX Wrap Event TRB

Table 31-18: **MFINDEX TRB Data Structure**

31 – 24	23 – 16	15 – 10	9 – 1	0	
RsvdZ					03-00H
RsvdZ					07-04H
Completion Code	RsvdZ				0B-08H
RsvdZ		TRB Type	RsvdZ	C	0F-0CH

Command TRB

NoOp Command TRB

Table 31-19: **NoOp Command TRB Data Structure**

31 16	15 10	9 1	0	
RsvdZ				03-00H
RsvdZ				07-04H
RsvdZ				0B-08H
RsvdZ	TRB Type	RsvdZ	C	0F-0CH

Enable Slot Command TRB

Table 31-20: **Enable Slot TRB Data Structure**

31 21	20 16	15 10	9 1	0	
RsvdZ					03-00H
RsvdZ					07-04H
RsvdZ					0B-08H
RsvdZ	Slot Type	TRB Type	RsvdZ	C	0F-0CH

Disable Slot Command TRB

Table 31-21: **Disable Slot TRB Data Structure**

31 24	23 16	15 10	9 1	0	
RsvdZ					03-00H
RsvdZ					07-04H
RsvdZ					0B-08H
Slot ID	RsvdZ	TRB Type	RsvdZ	C	0F-0CH

Address Device Command TRB

Table 31-22: **Address Device TRB Data Structure**

31 24	23 16	15 10	9	8 4	3 1	0	
Input Context Pointer Low					RsvdZ		03-00H
Input Context Pointer High							07-04H
RsvdZ							0B-08H
Slot ID	RsvdZ	TRB Type	BSR	RsvdZ		C	0F-0CH

Configure Endpoint Command TRB

Table 31-23: **Configure Endpoint TRB Data Structure**

31 24	23 16	15 10	9	8 4	3 1	0	
Input Context Pointer Low					RsvdZ		03-00H
Input Context Pointer High							07-04H
RsvdZ							0B-08H
Slot ID	RsvdZ	TRB Type	DC	RsvdZ		C	0F-0CH

Evaluate Context Command TRB

Table 31-24: **Evaluate Context TRB Data Structure**

31 24	23 16	15 10	9 4	3 1	0	
Input Context Pointer Low				RsvdZ		03-00H
Input Context Pointer High						07-04H
RsvdZ						0B-08H
Slot ID	RsvdZ	TRB Type	RsvdZ		C	0F-0CH

Reset Endpoint Command TRB

Table 31-25: **Reset Endpoint TRB Data Structure**

31 24	23 21	20 16	15 10	9	8 1	0	
RsvdZ							03-00H
RsvdZ							07-04H
RsvdZ							0B-08H
Slot ID	RsvdZ	Endpoint ID	TRB Type	TSP	RsvdZ	C	0F-0CH

Stop Endpoint Command TRB

Table 31-26: **Stop Endpoint TRB Data Structure**

31 24	23	22 21	20 16	15 10	9 1	0	
RsvdZ							03-00H
RsvdZ							07-04H
RsvdZ							0B-08H
Slot ID	SP	RsvdZ	Endpoint ID	TRB Type	RsvdZ	C	0F-0CH

Set TR Dequeue Pointer Command TRB

Table 31-27: **Set TR Dequeue Pointer TRB Data Structure**

31 24	23 21	20 16	15 10	9 4	3 1	0	
New TR Dequeue Pointer Low					SCT	DCS	03-00H
New TR Dequeue Pointer High							07-04H
Stream ID			RsvdZ				0B-08H
Slot ID	RsvdZ	Endpoint ID	TRB Type	RsvdZ		C	0F-0CH

Reset Device Command TRB

Table 31-28: **Reset Device TRB Data Structure**

31 24	23 16	15 10	9 1	0	
RsvdZ					03-00H
RsvdZ					07-04H
RsvdZ					0B-08H
Slot ID	RsvdZ	TRB Type	RsvdZ	C	0F-0CH

Force Event Command TRB

Table 31-29: **Force Event TRB Data Structure**

31 24	23 22	21 16	15 10	9 4	3 1	0	
Event TRB Pointer Low					RsvdZ		03-00H
Event TRB Pointer High							07-04H
VF Interrupter Target		RsvdZ					0B-08H
RsvdZ	VF ID		TRB Type	RsvdZ		C	0F-0CH

Negotiate Bandwidth Command TRB

The negotiate bandwidth command TRB uses the same format as the disable slot command Table 31-21, with the exception that the TRB type field is set to the negotiate bandwidth command TRB type ID, and the slot ID is set to the ID of the slot that requires the bandwidth negotiation.

Set Latency Tolerance Command TRB

Table 31-30: **Set Latency Tolerance TRB Data Structure**

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	
RsvdZ					03-00H
RsvdZ					07-04H
RsvdZ					0B-08H
RsvdZ	Best Effort Latency Tolerance Value (BELT)	TRB Type	RsvdZ	C	0F-0CH

Get Port Bandwidth Command TRB

Table 31-31: **Get Port Bandwidth TRB Data Structure**

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

Port Bandwidth Context Pointer Low				RsvdZ		03-00H
Port Bandwidth Context Pointer High						07-04H
RsvdZ						0B-08H
Hub Slot ID	RsvdZ	Dev Speed	TRB Type	RsvdZ	C	0F-0CH

Force Header Command TRB

Table 31-32: **Force Header TRB Data Structure**

31 24 23 16 15 10 9 5 4 1 0

Header Information Low				Type	03-00H
Header Information Mid					07-04H
Header Information High					0B-08H
Root Hub Port Number	RsvdZ	TRB Type	RsvdZ	C	0F-0CH

Other TRBs

Link TRB

Table 31-33: **Link TRB Data Structure**

31 22 21 17 16 15 10 9 6 5 4 3 2 1 0

Ring Segment Pointer Low							RsvdZ		03-00H
Ring Segment Pointer High									07-04H
Interrupter Target	RsvdZ								0B-08H
RsvdZ	TRB Type	RsvdZ	IOC	CH	RsvdZ	TC	C		0F-0CH

Event Data TRB

Table 31-34: **Event Data TRB Data Structure**

31 21 17 16 15 10 9 8 6 5 4 3 2 1 0

Event Data Low									03-00H
Event Data High									07-04H
Interrupter Target	RsvdZ								0B-08H
RsvdZ	TRB Type = Translate	BEI	RsvdZ	IOC	CH	RsvdZ	ENT	C	0F-0CH

Programming Guide

Initial Commands to USB Controller

There is only one command ring that is used for issuing xHCI specific commands or commands related to device slots.

The command ring control register is defined in the operational register space. All xHCI commands are issued by placing the desired command TRB(s) on the command ring, then ringing the xHCI command doorbell register, that is writing the host controller command code to the DB target field of doorbell register 0.

All commands result in the generation of a command completion event TRB on the event ring.

Host Mode Initialization

After the system boots, the following steps are executed in the system software.

1. The host controller is enumerated, assigned a base address for the xHCI register space.
2. The system software sets the frame length adjustment (FLADJ) register to a system-specific value.
3. Initialize the system I/O memory maps.
4. Wait until the controller not ready (CNR) flag in the USBSTS is `0` before writing any xHCI operational or run time registers.
5. Program the maximum device slots enabled (MaxSlotsEn) field in the configuration register to enable the device slots that system software is going to use.
6. Program the device context base address array pointer (DCBAAP) register with a 64-bit address pointing to where the device context base address array is located.
7. Define the command ring dequeue pointer by programming the command ring control register with a 64-bit address pointing to the starting address of the first TRB of the command ring.
8. Initialize interrupts
9. Initialize each active interrupter by with the following steps.
    a. Defining the event ring.
        - Allocate and initialize the event ring segment(s)
        - Allocate the event ring segment table (ERST). Initialize the ERST table entries to point to and to define the size (in TRBs) of the respective event ring segment.

- Program the interrupter event ring segment table size (ERSTSZ) register with the number of segments described by the event ring segment table.
- Program the interrupter event ring dequeue pointer (ERDP) register with the starting address of the first segment described by the event ring segment table.
- Program the interrupter event ring segment table base address (ERSTBA) register with a 64-bit address pointer to where the event ring segment table is located.

  Note: Writing the ERSTBA enables the event ring.

b. Define the interrupts.

- Enable system bus interrupt generation by writing a `1` to the interrupter enable (INTE) flag of the USBCMD register.
- Enable the interrupter by writing a `1` to the interrupt enable (IE) field of the management register.
- Write the USBCMD to turn the host controller ON by setting the run/stop (R/S) bit to `1`. This operation allows the xHCI to begin accepting doorbell references.

c. The host controller is now up and running.

10. The root hub ports begin reporting device connects. System software can begin enumerating devices. USB2 (LS/FS/HS) devices require the port reset process to advance the port to the enabled state. Once USB2 ports are enabled, the port is active with start of files (SOFs) occurring on the port.

Device Detection, Enumeration

The USB device initialization process is the same, whether the device attached to the port is a function or a hub. Once the pipes associated with an external hub are set up, the hub driver enumerates the devices attached to the external hub's ports using standard hub class command sequences.

1. When the xHCI detects a device attach, it sets the current connect status (CCS) and connect status change (CSC) flags to `1`. If the assertion of CSC results in a `0` to `1` transition of the port status change event generation, the xHCI generates a port status change event.
2. Upon receipt of a port status change event the system software evaluates the port ID field to determine the port that generated the event.
3. System software then reads the PORTSC register of the port that generated the event. CSC = `1` if the event was due to attach (CCS = `1`) or detach (CCS = `0`).
4. A USB2 protocol port requires software to reset the port, to advance the port to the enabled state, and a USB device from the powered state to the default state. After an attach event, the PED and PR flags is `0` and the PLS field is `7` (polling) in the PORTSC register.

5. System software enables the port by resetting the port (writing a `1` to the PORTSC PR bit) then waiting for a port status change event due to the assertion of the port reset change (PRC) flag.
6. The completion of the port reset causes the PORTSC register PRC and PED flags to be set (`1`), the PR flag to be cleared (`0`), and the PLS field to be U0 (`0`). If the assertion of PRC results in a `0` to `1` transition of PSCEG, the xHCI generates a port status change event as a result of the transition of PRC. The reset operation sets the USB2 device into the default state, preparing it for a SET_ADDRESS request.
7. After the port successfully reaches the enabled state, the system software obtains a device slot for the newly attached device using an enable slot command.
8. After successfully obtaining a device slot, the system software initializes the data structures associated with the slot.
9. Once the slots related data structures are initialized, the system software uses an address device command to assign an address to the device and enable its default control endpoint.
10. For full-speed devices, the system software should initially read the first 8 bytes of the USB device descriptor to retrieve the value of the bMaxPacketSize0 field and determine the actual maximum packet size for the default control endpoint, by issuing a USB GET_DESCRIPTOR request to the device, update the default control endpoint context with the actual maximum packet size, and inform the xHCI of the context change.
11. Software then issues an evaluate context command with add context bit [1] (A1) set to `1` to inform the xHCI of the change to the default control endpoint's maximum packet size parameter.
12. Now that the default control endpoint is fully operational.
    a. System software can read the complete USB device descriptor and possibly the configuration descriptors to be able to hand the device off to the appropriate class driver(s).
    b. The class driver can then configure the device slot using a configure endpoint command, and configure the USB device itself by issuing a USB SET_CONFIGURATION request through the devices' default control endpoint.
    c. The successful completion of both operations is required to advance the state of the USB device from *Address* to *Configured* and the xHCI device slot from *Address* to *Configured*.
    d. If required, the system software can configure alternate interfaces.
13. The pipe interfaces to the USB device are now fully operational.

Device Detach

- When the device is detached from a root hub port, the PORTSC current connection status (CCS) bit is cleared to `0` and the connect status change (CSC) bit is set to `1`.

- If a 0 to 1 transition of PSCEG occurs, the xHCI reports the change through a port status change event.
- After the detection of detach, system software disables the device slot associated with the port by issuing a disable slot command for the affected slot.

Device Programming

The following sequence describes register programming for initialization of the xHCI controller as a USB 2.0 device.

1. In register DCTL, set the CSftRst field to 1 and wait for a read to return 0. This resets the device.
2. In registers GSBUSCFG0/1, leave the default values if the correct power-on values were selected during coreConsultant configuration.
3. This step is only required to enable threshold. In register GTXTHRCFG/ GRXTHRCFG, leave the default values (if the correct power-on values were selected during coreConsultant configuration).
4. In register GSNPSID, the software must read the Synopsys ID register to find the device version and configure the driver for any version-specific features.
5. Optionally, in register GUID, the software can program the user ID GUID read/write register.
6. In register GUSB2PHYCFG, program the following PHY configuration fields: USBTrdTim, FSIntf, PHYIf, TOUTCal, or leave the default values (if the correct power-on values were selected during coreConsultant configuration).

IMPORTANT: *The PHY must not be enabled for auto-resume in device mode. The field GUSB2PHYCFG[15] (ULPIAutoRes) must be written with 0 during the power-on initialization in case the reset value is 1.*

7. In register GUSB3PIPECTL, program the following PHY configuration fields: DatWidth, PrtOpDir, or leave the default values (if the correct power-on values were selected during coreConsultant configuration).
8. In register GTXFIFOSIZn, write these registers to allocate prefetch buffers for each TX endpoint. Unless the packet sizes of the endpoints are application specific, it is recommended to use the default value.
9. In register GRXFIFOSIZ0, write this register to allocate the receive buffer for all endpoints. Unless the packet sizes of the endpoints are application-specific, it is recommended to use the default value.
10. In registers GEVNTADRn/ GEVNTSIZn/ GEVNTCOUNTn, depending on the number of interrupts allocated, program the event buffer address and size registers to point to the

event buffer locations in system memory, the sizes of the buffers, and unmask the interrupt.

IMPORTANT: *USB operation stops if the event buffer memory is insufficient, because the block stops receiving/transmitting packets.*

11. In register GCTL, program this register to override scaledown, RAM clock select, and clock gating parameters.
12. In register DCFG, program device speed and periodic frame interval.
13. In register DEVTEN, at a minimum, enable USB reset, connection done, and USB/link state change events.
14. In register DEPCMD0, issue a DEPSTARTCFG command with DEPCMD0.XferRscIdx set to `0` and CmdIOC set to `0` to initialize the transfer resource allocation. Poll CmdAct for completion.
15. In registers DEPCMD0/ DEPCMD1, issue a DEPCFG command for physical endpoints 0 and 1 with the following characteristics, and poll CmdAct for completions.
    - USB endpoint number = 0 or 1 (for physical endpoint 0 or 1)
    - FIFONum = 0
    - XferNRdyEn and XferCmplEn = 1
    - Maximum packet size = 512
    - Burst size = 0
    - EPType = `2’b00` (Control)
16. In registers DEPCMD0/ DEPCMD1, issue a DEPXFERCFG command for physical endpoints 0 and 1 with DEPCMDPAR0_0/1 set to `1`, and poll CmdAct for completions.
17. In register DEPCMD0, prepare a buffer for a setup packet, initialize a setup TRB, and issue a DEPSTRTXFER command for physical endpoint 0, pointing to the setup TRB. Poll CmdAct for completion.

    Note: The block attempts to fetch the setup TRB through the master interface after this command completes.
18. In register DALEPENA, enable physical endpoints 0 and 1 by writing `0x3` to this register.
19. In register DCTL, set DCTL.RunStop to `1` to allow the device to attach to the host. The device now is ready to receive start-of-file (SOF) packets, respond to control transfers on control endpoint 0, and generate events.

Register Overview

The registers are classified into four groups. These registers are commonly used for various modes of operations (USB 2.0 host, device, and OTG, and USB 3.0 host and device).

The base addresses of the controllers are listed.

For inst0: FE20xxxx

For inst1: FE30xxxx

The register address map is shown in Table 31-35. Further definition of these registers are listed in Table 31-36 through Table 31-40.

- Global registers (Table 31-36)
- Device registers (Table 31-37)
- OTG and battery charger registers (Table 31-38)
- xHCI host registers (Table 31-39)
- Groups of register map (Table 31-40)
- MIO Pins (Table 31-41)

Table 31-35: **Register Address Map**

Address Index	Register Type
`0x0_0000` to `0x0_7FFF` • 0 to CAPLENGTH – 1 • CAPLENGTH to RTSOFF – 1 • RTSOFF to DBOFF – 1 • DBOFF to (xECP*4 – 1) • (xECP*4) to `0x0_7FFF`	xHCI registers • xHCI capability registers • Host controller operational registers • Controller run time registers • Doorbell registers • xHCI extended capabilities
`0x0_C100` to `0x0_C6FF`	Global registers
`0x0_C700` to `0x0_CBFF`	Device registers
`0x0_CC00` to `0x0_CCFF`	OTG and battery charger registers
`0x0CCD00` to `0x0_CFFF`	Unused
`0x4_0000` to `0x7_FFFF`	Internal RAM#0 debug access (256 KB)
`0x8_0000` to `0xB_FFFF`	Internal RAM#1 debug access (256 KB)
`0xC_0000` to `0xF_FFFF`	Internal RAM#2 debug access (256 KB)

Table 31-36: **Global Registers**

Register Name	Address	Width	Type	Reset Value	Description
GSBUSCFG0	0x0000C100	32	Mixed	0x0000000E	Global MPSoC bus configuration register 0
GSBUSCFG1	0x0000C104	32	Mixed	0x00000300	Global MPSoC bus configuration register 1
GTXTHRCFG	0x0000C108	32	Mixed	0x00000000	Global TX threshold control register
GRXTHRCFG	0x0000C10C	32	Mixed	0x00000000	Global RX threshold control register
GCTL	0x0000C110	32	Read/Write	0x00593004	Global common register
GPMSTS	0x0000C114	32	Mixed	0x00000000	Global power management status register
GSTS	0x0000C118	32	Read only	0x3E800000	Global status register
GUCTL1	0x0000C11C	32	Read/Write	0x0000018A	Global user control register 1
GSNPSID	0x0000C120	32	Read only	0x5533260A	Global Synopsys ID register
GGPIO	0x0000C124	32	Mixed	0x00000000	Global general purpose I/O register
GUID	0x0000C128	32	Read/Write	0x12345678	Global user ID register
GUCTL	0x0000C12C	32	Read/Write	0x0F808010	Global user control register
GBUSERRADDRLO	0x0000C130	32	Read only	0x00000000	Register GBUSERRADDRLO
GBUSERRADDRHI	0x0000C134	32	Read only	0x00000000	Register GBUSERRADDRHI
GPRTBIMAPLO	0x0000C138	32	Mixed	0x00000000	Register R
GPRTBIMAPHI	0x0000C13C	32	Read only	0x00000000	Register R
GHWPARAMS0	0x0000C140	32	Read only	0x4020404A	Global hardware parameters register 0
GHWPARAMS1	0x0000C144	32	Read only	0x8262493B	Global hardware parameters register 1
GHWPARAMS2	0x0000C148	32	Read only	0x12345678	Global hardware parameters register 2
GHWPARAMS3	0x0000C14C	32	Read only	0x0618C089	Global hardware parameters register 3
GHWPARAMS4	0x0000C150	32	Read only	0x47822004	Global hardware parameters register 4
GHWPARAMS5	0x0000C154	32	Read only	0x04204108	Global hardware parameters register 5
GHWPARAMS6	0x0000C158	32	Read only	0x07BAAC20	Global hardware parameters register 6
GHWPARAMS7	0x0000C15C	32	Read only	0x030807D6	Global hardware parameters register 7
GDBGFIFOSPACE	0x0000C160	32	Mixed	0x00420000	Global debug queue/FIFO apace available register
GDBGLTSSM	0x0000C164	32	Read only	0x01010442	Global debug LTSSM register
GDBGLNMCC	0x0000C168	32	Read only	0x00000000	Global debug LNMCC register

Table 31-36: **Global Registers *(Cont'd)***

Register Name	Address	Width	Type	Reset Value	Description
GDBGBMU	0x0000C16C	32	Read only	0x00000000	Global debug BMU register
GDBGLSPMUX_HST	0x0000C170	32	Mixed	0x003F0000	Internal global debug LSP MUX register
GDBGLSP	0x0000C174	32	Read only	0x00000000	Global debug LSP register
GDBGEPINFO0	0x0000C178	32	Read only	0x00000000	Global debug endpoint information register 0
GDBGEPINFO1	0x0000C17C	32	Read only	0x00800000	Global debug endpoint information register 1
GPRTBIMAP_HSLO	0x0000C180	32	Mixed	0x00000000	High-speed port to bus instance mapping
GPRTBIMAP_HSHI	0x0000C184	32	Read only	0x00000000	High-speed port to bus instance mapping
GPRTBIMAP_FSLO	0x0000C188	32	Mixed	0x00000000	Register full-speed port to bus instance mapping
GPRTBIMAP_FSHI	0x0000C18C	32	Read only	0x00000000	Register full-speed port to bus instance mapping
Reserved_94	0x0000C194	32	Read/Write	0x00000000	Reserved register
Reserved_98	0x0000C198	32	Read/Write	0x00000000	Reserved register
GUSB2PHYCFG	0x0000C200	32	Mixed	0x00002410	Register Rs
GUSB2I2CCTL	0x0000C240	32	Read only	0x00000000	Reserved register
GUSB2PHYACC_ULPI	0x0000C280	32	Read only	x	Register PHYACC_ULPI
GUSB3PIPECTL	0x0000C2C0	32	Mixed	0x010C0002	Register GUSB3PIPECTL
GTXFIFOSIZ0	0x0000C300	32	Read/Write	0x00000042	Register GTXFIFOSIZ 0
GTXFIFOSIZ1	0x0000C304	32	Read/Write	0x00420184	Register GTXFIFOSIZ 1
GTXFIFOSIZ2	0x0000C308	32	Read/Write	0x01C60184	Register GTXFIFOSIZ 2
GTXFIFOSIZ3	0x0000C30C	32	Read/Write	0x034A0184	Register GTXFIFOSIZ 3
GTXFIFOSIZ4	0x0000C310	32	Read/Write	0x04CE0184	Register GTXFIFOSIZ 4
GTXFIFOSIZ5	0x0000C314	32	Read/Write	0x06520184	Register GTXFIFOSIZ 5
GRXFIFOSIZ0	0x0000C380	32	Read/Write	0x00000185	Register
GRXFIFOSIZ1	0x0000C384	32	Read/Write	0x01850000	Register
GRXFIFOSIZ2	0x0000C388	32	Read/Write	0x01850000	Register
GEVNTADRLO_0	0x0000C400	32	Read/Write	0x00000000	Register GEVNTADRLO instance 0 of an array of four.
GEVNTADRHI_0	0x0000C404	32	Read/Write	0x00000000	Register GEVNTADRHI[0] instance 0 of an array of four.
GEVNTSIZ_0	0x0000C408	32	Mixed	0x00000000	Register instance 0 of an array of four.

Table 31-36: **Global Registers *(Cont'd)***

Register Name	Address	Width	Type	Reset Value	Description
GEVNTCOUNT_0	0x0000C40C	32	Mixed	0x00000000	Register instance 0 of an array of four.
GEVNTADRLO_1	0x0000C410	32	Read/Write	0x00000000	Register GEVNTADRLO instance 1 of an array of four.
GEVNTADRHI_1	0x0000C414	32	Read/Write	0x00000000	Register GEVNTADRHI[0] instance 1 of an array of four.
GEVNTSIZ_1	0x0000C418	32	Mixed	0x00000000	Register instance 1 of an array of four.
GEVNTCOUNT_1	0x0000C41C	32	Mixed	0x00000000	Register instance 1 of an array of four.
GEVNTADRLO_2	0x0000C420	32	Read/Write	0x00000000	Register GEVNTADRLO instance 2 of an array of four.
GEVNTADRHI_2	0x0000C424	32	Read/Write	0x00000000	Register GEVNTADRHI[0] instance 2 of an array of four.
GEVNTSIZ_2	0x0000C428	32	Mixed	0x00000000	Register instance 2 of an array of four.
GEVNTCOUNT_2	0x0000C42C	32	Mixed	0x00000000	Register instance 2 of an array of four.
GEVNTADRLO_3	0x0000C430	32	Read/Write	0x00000000	Register GEVNTADRLO instance 3 of an array of four.
GEVNTADRHI_3	0x0000C434	32	Read/Write	0x00000000	Register GEVNTADRHI[0] instance 3 of an array of four.
GEVNTSIZ_3	0x0000C438	32	Mixed	0x00000000	Register instance 3 of an array of four.
GEVNTCOUNT_3	0x0000C43C	32	Mixed	0x00000000	Register instance 3 of an array of four.
GHWPARAMS8	0x0000C600	32	Read only	0x000007BA	Global hardware parameters register 8
GTXFIFOPRIDEV	0x0000C610	32	Mixed	0x00000000	Global device TX FIFO DMA priority register
GTXFIFOPRIHST	0x0000C618	32	Mixed	x	Global host TX FIFO DMA priority register
GRXFIFOPRIHST	0x0000C61C	32	Mixed	x	Global host RX FIFO DMA priority register
GFIFOPRIDBC	0x0000C620	32	Read/Write	x	Global host debug capability DMA priority register
GDMAHLRATIO	0x0000C624	32	Mixed	0x00000808	Global host FIFO DMA High-Low priority ratio register
GFLADJ	0x0000C630	32	Read/Write	0x0F83F020	Global frame length adjustment register

Table 31-37: **Device Registers**

Register Name	Address	Width	Type	Reset Value	Description
DCFG	0x0000C700	32	Read/Write	0x00080804	Device configuration register
DCTL	0x0000C704	32	Mixed	0x00000000	Device control register
DEVTEN	0x0000C708	32	Mixed	x	Device event enable register
DSTS	0x0000C70C	32	Mixed	0x00520004	Device status register
DGCMDPAR	0x0000C710	32	Read/Write	0x00000000	Device generic command parameter register
DGCMD	0x0000C714	32	Mixed	0x00000000	Device generic command register
DALEPENA	0x0000C720	32	Read/Write	0x00000000	Device active USB endpoint enable register
Rsvd0	0x0000C780	32	Read Only	0x00000000	Register reserved
Rsvd1	0x0000C784	32	Read Only	0x00000000	Register reserved
Rsvd2	0x0000C788	32	Read Only	0x00000000	Register reserved
Rsvd3	0x0000C78C	32	Read Only	0x00000000	Register reserved
Rsvd4	0x0000C790	32	Read Only	0x00000000	Register reserved
Rsvd5	0x0000C794	32	Read Only	0x00000000	Register reserved
Rsvd6	0x0000C798	32	Read Only	0x00000000	Register reserved
Rsvd7	0x0000C79C	32	Read Only	0x00000000	Register reserved
Rsvd8	0x0000C7A0	32	Read Only	0x00000000	Register reserved
Rsvd9	0x0000C7A4	32	Read Only	0x00000000	Register reserved
Rsvd10	0x0000C7A8	32	Read Only	0x00000000	Register reserved
Rsvd11	0x0000C7AC	32	Read Only	0x00000000	Register reserved
Rsvd12	0x0000C7B0	32	Read Only	0x00000000	Register reserved
Rsvd13	0x0000C7B4	32	Read Only	0x00000000	Register reserved
Rsvd14	0x0000C7B8	32	Read Only	0x00000000	Register reserved
Rsvd15	0x0000C7BC	32	Read Only	0x00000000	Register reserved
Rsvd16	0x0000C7C0	32	Read Only	0x00000000	Register reserved
Rsvd17	0x0000C7C4	32	Read Only	0x00000000	Register reserved
Rsvd18	0x0000C7C8	32	Read Only	0x00000000	Register reserved
Rsvd19	0x0000C7CC	32	Read Only	0x00000000	Register reserved
Rsvd20	0x0000C7D0	32	Read Only	0x00000000	Register reserved
Rsvd21	0x0000C7D4	32	Read Only	0x00000000	Register reserved
Rsvd22	0x0000C7D8	32	Read Only	0x00000000	Register reserved
Rsvd23	0x0000C7DC	32	Read Only	0x00000000	Register reserved
Rsvd24	0x0000C7E0	32	Read Only	0x00000000	Register reserved
Rsvd25	0x0000C7E4	32	Read Only	0x00000000	Register reserved

Table 31-37: **Device Registers *(Cont'd)***

Register Name	Address	Width	Type	Reset Value	Description
Rsvd26	0x0000C7E8	32	Read Only	0x00000000	Register reserved
Rsvd27	0x0000C7EC	32	Read Only	0x00000000	Register reserved
Rsvd28	0x0000C7F0	32	Read Only	0x00000000	Register reserved
Rsvd29	0x0000C7F4	32	Read Only	0x00000000	Register reserved
Rsvd30	0x0000C7F8	32	Read Only	0x00000000	Register reserved
Rsvd31	0x0000C7FC	32	Read Only	0x00000000	Register reserved
DEPCMDPAR2_0	0x0000C800	32	Read/Write	0x00000000	Register DEPCMDPAR2 instance 0 of an array of 32.
DEPCMDPAR1_0	0x0000C804	32	Read/Write	0x00000000	Register DEPCMDPAR1[0] instance 0 of an array of 32.
DEPCMDPAR0_0	0x0000C808	32	Read/Write	0x00000000	Register DEPCMDPAR0[0] instance 0 of an array of 32.
DEPCMD_0	0x0000C80C	32	Read/Write	0x00000000	Register R instance 0 of an array of 32.
DEPCMDPAR2_1	0x0000C810	32	Read/Write	0x00000000	Register DEPCMDPAR2 instance 1 of an array of 32.
DEPCMDPAR1_1	0x0000C814	32	Read/Write	0x00000000	Register DEPCMDPAR1[0] instance 1 of an array of 32.
DEPCMDPAR0_1	0x0000C818	32	Read/Write	0x00000000	Register DEPCMDPAR0[0] instance 1 of an array of 32.
DEPCMD_1	0x0000C81C	32	Read/Write	0x00000000	Register R instance 1 of an array of 32.
DEPCMDPAR2_2	0x0000C820	32	Read/Write	0x00000000	Register DEPCMDPAR2 instance 2 of an array of 32.
DEPCMDPAR1_2	0x0000C824	32	Read/Write	0x00000000	Register DEPCMDPAR1[0] instance 2 of an array of 32.
DEPCMDPAR0_2	0x0000C828	32	Read/Write	0x00000000	Register DEPCMDPAR0[0] instance 2 of an array of 32.
DEPCMD_2	0x0000C82C	32	Read/Write	0x00000000	Register R instance 2 of an array of 32.
DEPCMDPAR2_3	0x0000C830	32	Read/Write	0x00000000	Register DEPCMDPAR2 instance 3 of an array of 32.
DEPCMDPAR1_3	0x0000C834	32	Read/Write	0x00000000	Register DEPCMDPAR1[0] instance 3 of an array of 32.
DEPCMDPAR0_3	0x0000C838	32	Read/Write	0x00000000	Register DEPCMDPAR0[0] instance 3 of an array of 32.
DEPCMD_3	0x0000C83C	32	Read/Write	0x00000000	Register R instance 3 of an array of 32.
DEPCMDPAR2_4	0x0000C840	32	Read/Write	0x00000000	Register DEPCMDPAR2 instance 4 of an array of 32.
DEPCMDPAR1_4	0x0000C844	32	Read/Write	0x00000000	Register DEPCMDPAR1[0] instance 4 of an array of 32.
DEPCMDPAR0_4	0x0000C848	32	Read/Write	0x00000000	Register DEPCMDPAR0[0] instance 4 of an array of 32.

Table 31-37: **Device Registers *(Cont'd)***

Register Name	Address	Width	Type	Reset Value	Description
DEPCMD_4	0x0000C84C	32	Read/Write	0x00000000	Register R instance 4 of an array of 32.
DEPCMDPAR2_5	0x0000C850	32	Read/Write	0x00000000	Register DEPCMDPAR2 instance 5 of an array of 32.
DEPCMDPAR1_5	0x0000C854	32	Read/Write	0x00000000	Register DEPCMDPAR1[0] instance 5 of an array of 32.
DEPCMDPAR0_5	0x0000C858	32	Read/Write	0x00000000	Register DEPCMDPAR0[0] instance 5 of an array of 32.
DEPCMD_5	0x0000C85C	32	Read/Write	0x00000000	Register R instance 5 of an array of 32.
DEPCMDPAR2_6	0x0000C860	32	Read/Write	0x00000000	Register DEPCMDPAR2 instance 6 of an array of 32.
DEPCMDPAR1_6	0x0000C864	32	Read/Write	0x00000000	Register DEPCMDPAR1[0] instance 6 of an array of 32.
DEPCMDPAR0_6	0x0000C868	32	Read/Write	0x00000000	Register DEPCMDPAR0[0] instance 6 of an array of 32.
DEPCMD_6	0x0000C86C	32	Read/Write	0x00000000	Register R instance 6 of an array of 32.
DEPCMDPAR2_7	0x0000C870	32	Read/Write	0x00000000	Register DEPCMDPAR2 instance 7 of an array of 32.
DEPCMDPAR1_7	0x0000C874	32	Read/Write	0x00000000	Register DEPCMDPAR1[0] instance 7 of an array of 32.
DEPCMDPAR0_7	0x0000C878	32	Read/Write	0x00000000	Register DEPCMDPAR0[0] instance 7 of an array of 32.
DEPCMD_7	0x0000C87C	32	Read/Write	0x00000000	Register R instance 7 of an array of 32.
DEPCMDPAR2_8	0x0000C880	32	Read/Write	0x00000000	Register DEPCMDPAR2 instance 8 of an array of 32.
DEPCMDPAR1_8	0x0000C884	32	Read/Write	0x00000000	Register DEPCMDPAR1[0] instance 8 of an array of 32.
DEPCMDPAR0_8	0x0000C888	32	Read/Write	0x00000000	Register DEPCMDPAR0[0] instance 8 of an array of 32.
DEPCMD_8	0x0000C88C	32	Read/Write	0x00000000	Register R instance 8 of an array of 32.
DEPCMDPAR2_9	0x0000C890	32	Read/Write	0x00000000	Register DEPCMDPAR2 instance 9 of an array of 32.
DEPCMDPAR1_9	0x0000C894	32	Read/Write	0x00000000	Register DEPCMDPAR1[0] instance 9 of an array of 32.
DEPCMDPAR0_9	0x0000C898	32	Read/Write	0x00000000	Register DEPCMDPAR0[0] instance 9 of an array of 32.
DEPCMD_9	0x0000C89C	32	Read/Write	0x00000000	Register R instance 9 of an array of 32.
DEPCMDPAR2_10	0x0000C8A0	32	Read/Write	0x00000000	Register DEPCMDPAR2 instance 10 of an array of 32.
DEPCMDPAR1_10	0x0000C8A4	32	Read/Write	0x00000000	Register DEPCMDPAR1[0] instance 10 of an array of 32.

Table 31-37: **Device Registers *(Cont'd)***

Register Name	Address	Width	Type	Reset Value	Description
DEPCMDPAR0_10	0x0000C8A8	32	Read/Write	0x00000000	Register DEPCMDPAR0[0] instance 10 of an array of 32.
DEPCMD_10	0x0000C8AC	32	Read/Write	0x00000000	Register R instance 10 of an array of 32.
DEPCMDPAR2_11	0x0000C8B0	32	Read/Write	0x00000000	Register DEPCMDPAR2 instance 11 of an array of 32.
DEPCMDPAR1_11	0x0000C8B4	32	Read/Write	0x00000000	Register DEPCMDPAR1[0] instance 11 of an array of 32.
DEPCMDPAR0_11	0x0000C8B8	32	Read/Write	0x00000000	Register DEPCMDPAR0[0] instance 11 of an array of 32.
DEPCMD_11	0x0000C8BC	32	Read/Write	0x00000000	Register R instance 11 of an array of 32.
DEPCMDPAR2_12	0x0000C8C0	32	Read/Write	0x00000000	Register DEPCMDPAR2 instance 12 of an array of 32.
DEPCMDPAR1_12	0x0000C8C4	32	Read/Write	0x00000000	Register DEPCMDPAR1[0] instance 12 of an array of 32.
DEPCMDPAR0_12	0x0000C8C8	32	Read/Write	0x00000000	Register DEPCMDPAR0[0] instance 12 of an array of 32.
DEPCMD_12	0x0000C8CC	32	Read/Write	0x00000000	Register R instance 12 of an array of 32.
DEPCMDPAR2_13	0x0000C8D0	32	Read/Write	0x00000000	Register DEPCMDPAR2 instance 13 of an array of 32.
DEPCMDPAR1_13	0x0000C8D4	32	Read/Write	0x00000000	Register DEPCMDPAR1[0] instance 13 of an array of 32.
DEPCMDPAR0_13	0x0000C8D8	32	Read/Write	0x00000000	Register DEPCMDPAR0[0] instance 13 of an array of 32.
DEPCMD_13	0x0000C8DC	32	Read/Write	0x00000000	Register R instance 13 of an array of 32.
DEPCMDPAR2_14	0x0000C8E0	32	Read/Write	0x00000000	Register DEPCMDPAR2 instance 14 of an array of 32.
DEPCMDPAR1_14	0x0000C8E4	32	Read/Write	0x00000000	Register DEPCMDPAR1[0] instance 14 of an array of 32.
DEPCMDPAR0_14	0x0000C8E8	32	Read/Write	0x00000000	Register DEPCMDPAR0[0] instance 14 of an array of 32.
DEPCMD_14	0x0000C8EC	32	Read/Write	0x00000000	Register R instance 14 of an array of 32.
DEPCMDPAR2_15	0x0000C8F0	32	Read/Write	0x00000000	Register DEPCMDPAR2 instance 15 of an array of 32.
DEPCMDPAR1_15	0x0000C8F4	32	Read/Write	0x00000000	Register DEPCMDPAR1[0] instance 15 of an array of 32.
DEPCMDPAR0_15	0x0000C8F8	32	Read/Write	0x00000000	Register DEPCMDPAR0[0] instance 15 of an array of 32.
DEPCMD_15	0x0000C8FC	32	Read/Write	0x00000000	Register R instance 15 of an array of 32.
DEPCMDPAR2_16	0x0000C900	32	Read/Write	0x00000000	Register DEPCMDPAR2 instance 16 of an array of 32.

Table 31-37: **Device Registers *(Cont'd)***

Register Name	Address	Width	Type	Reset Value	Description
DEPCMDPAR1_16	0x0000C904	32	Read/Write	0x00000000	Register DEPCMDPAR1[0] instance 16 of an array of 32.
DEPCMDPAR0_16	0x0000C908	32	Read/Write	0x00000000	Register DEPCMDPAR0[0] instance 16 of an array of 32.
DEPCMD_16	0x0000C90C	32	Read/Write	0x00000000	Register R instance 16 of an array of 32.
DEPCMDPAR2_17	0x0000C910	32	Read/Write	0x00000000	Register DEPCMDPAR2 instance 17 of an array of 32.
DEPCMDPAR1_17	0x0000C914	32	Read/Write	0x00000000	Register DEPCMDPAR1[0] instance 17 of an array of 32.
DEPCMDPAR0_17	0x0000C918	32	Read/Write	0x00000000	Register DEPCMDPAR0[0] instance 17 of an array of 32.
DEPCMD_17	0x0000C91C	32	Read/Write	0x00000000	Register R instance 17 of an array of 32.
DEPCMDPAR2_18	0x0000C920	32	Read/Write	0x00000000	Register DEPCMDPAR2 instance 18 of an array of 32.
DEPCMDPAR1_18	0x0000C924	32	Read/Write	0x00000000	Register DEPCMDPAR1[0] instance 18 of an array of 32.
DEPCMDPAR0_18	0x0000C928	32	Read/Write	0x00000000	Register DEPCMDPAR0[0] instance 18 of an array of 32.
DEPCMD_18	0x0000C92C	32	Read/Write	0x00000000	Register R instance 18 of an array of 32.
DEPCMDPAR2_19	0x0000C930	32	Read/Write	0x00000000	Register DEPCMDPAR2 instance 19 of an array of 32.
DEPCMDPAR1_19	0x0000C934	32	Read/Write	0x00000000	Register DEPCMDPAR1[0] instance 19 of an array of 32.
DEPCMDPAR0_19	0x0000C938	32	Read/Write	0x00000000	Register DEPCMDPAR0[0] instance 19 of an array of 32.
DEPCMD_19	0x0000C93C	32	Read/Write	0x00000000	Register R instance 19 of an array of 32.
DEPCMDPAR2_20	0x0000C940	32	Read/Write	0x00000000	Register DEPCMDPAR2 instance 20 of an array of 32.
DEPCMDPAR1_20	0x0000C944	32	Read/Write	0x00000000	Register DEPCMDPAR1[0] instance 20 of an array of 32.
DEPCMDPAR0_20	0x0000C948	32	Read/Write	0x00000000	Register DEPCMDPAR0[0] instance 20 of an array of 32.
DEPCMD_20	0x0000C94C	32	Read/Write	0x00000000	Register R instance 20 of an array of 32.
DEPCMDPAR2_21	0x0000C950	32	Read/Write	0x00000000	Register DEPCMDPAR2 instance 21 of an array of 32.
DEPCMDPAR1_21	0x0000C954	32	Read/Write	0x00000000	Register DEPCMDPAR1[0] instance 21 of an array of 32.
DEPCMDPAR0_21	0x0000C958	32	Read/Write	0x00000000	Register DEPCMDPAR0[0] instance 21 of an array of 32.
DEPCMD_21	0x0000C95C	32	Read/Write	0x00000000	Register R instance 21 of an array of 32.

Table 31-37: **Device Registers *(Cont'd)***

Register Name	Address	Width	Type	Reset Value	Description
DEPCMDPAR2_22	0x0000C960	32	Read/Write	0x00000000	Register DEPCMDPAR2 instance 22 of an array of 32.
DEPCMDPAR1_22	0x0000C964	32	Read/Write	0x00000000	Register DEPCMDPAR1[0] instance 22 of an array of 32.
DEPCMDPAR0_22	0x0000C968	32	Read/Write	0x00000000	Register DEPCMDPAR0[0] instance 22 of an array of 32.
DEPCMD_22	0x0000C96C	32	Read/Write	0x00000000	Register R instance 22 of an array of 32.
DEPCMDPAR2_23	0x0000C970	32	Read/Write	0x00000000	Register DEPCMDPAR2 instance 23 of an array of 32.
DEPCMDPAR1_23	0x0000C974	32	Read/Write	0x00000000	Register DEPCMDPAR1[0] instance 23 of an array of 32.
DEPCMDPAR0_23	0x0000C978	32	Read/Write	0x00000000	Register DEPCMDPAR0[0] instance 23 of an array of 32.
DEPCMD_23	0x0000C97C	32	Read/Write	0x00000000	Register R instance 23 of an array of 32.
DEPCMDPAR2_24	0x0000C980	32	Read/Write	0x00000000	Register DEPCMDPAR2 instance 24 of an array of 32.
DEPCMDPAR1_24	0x0000C984	32	Read/Write	0x00000000	Register DEPCMDPAR1[0] instance 24 of an array of 32.
DEPCMDPAR0_24	0x0000C988	32	Read/Write	0x00000000	Register DEPCMDPAR0[0] instance 24 of an array of 32.
DEPCMD_24	0x0000C98C	32	Read/Write	0x00000000	Register R instance 24 of an array of 32.
DEPCMDPAR2_25	0x0000C990	32	Read/Write	0x00000000	Register DEPCMDPAR2 instance 25 of an array of 32.
DEPCMDPAR1_25	0x0000C994	32	Read/Write	0x00000000	Register DEPCMDPAR1[0] instance 25 of an array of 32.
DEPCMDPAR0_25	0x0000C998	32	Read/Write	0x00000000	Register DEPCMDPAR0[0] instance 25 of an array of 32.
DEPCMD_25	0x0000C99C	32	Read/Write	0x00000000	Register R instance 25 of an array of 32.
DEPCMDPAR2_26	0x0000C9A0	32	Read/Write	0x00000000	Register DEPCMDPAR2 instance 26 of an array of 32.
DEPCMDPAR1_26	0x0000C9A4	32	Read/Write	0x00000000	Register DEPCMDPAR1[0] instance 26 of an array of 32.
DEPCMDPAR0_26	0x0000C9A8	32	Read/Write	0x00000000	Register DEPCMDPAR0[0] instance 26 of an array of 32.
DEPCMD_26	0x0000C9AC	32	Read/Write	0x00000000	Register R instance 26 of an array of 32.
DEPCMDPAR2_27	0x0000C9B0	32	Read/Write	0x00000000	Register DEPCMDPAR2 instance 27 of an array of 32.
DEPCMDPAR1_27	0x0000C9B4	32	Read/Write	0x00000000	Register DEPCMDPAR1[0] instance 27 of an array of 32.
DEPCMDPAR0_27	0x0000C9B8	32	Read/Write	0x00000000	Register DEPCMDPAR0[0] instance 27 of an array of 32.

Table 31-37: **Device Registers *(Cont'd)***

Register Name	Address	Width	Type	Reset Value	Description
DEPCMD_27	0x0000C9BC	32	Read/Write	0x00000000	Register R instance 27 of an array of 32.
DEPCMDPAR2_28	0x0000C9C0	32	Read/Write	0x00000000	Register DEPCMDPAR2 instance 28 of an array of 32.
DEPCMDPAR1_28	0x0000C9C4	32	Read/Write	0x00000000	Register DEPCMDPAR1[0] instance 28 of an array of 32.
DEPCMDPAR0_28	0x0000C9C8	32	Read/Write	0x00000000	Register DEPCMDPAR0[0] instance 28 of an array of 32.
DEPCMD_28	0x0000C9CC	32	Read/Write	0x00000000	Register R instance 28 of an array of 32.
DEPCMDPAR2_29	0x0000C9D0	32	Read/Write	0x00000000	Register DEPCMDPAR2 instance 29 of an array of 32.
DEPCMDPAR1_29	0x0000C9D4	32	Read/Write	0x00000000	Register DEPCMDPAR1[0] instance 29 of an array of 32.
DEPCMDPAR0_29	0x0000C9D8	32	Read/Write	0x00000000	Register DEPCMDPAR0[0] instance 29 of an array of 32.
DEPCMD_29	0x0000C9DC	32	Read/Write	0x00000000	Register R instance 29 of an array of 32.
DEPCMDPAR2_30	0x0000C9E0	32	Read/Write	0x00000000	Register DEPCMDPAR2 instance 30 of an array of 32.
DEPCMDPAR1_30	0x0000C9E4	32	Read/Write	0x00000000	Register DEPCMDPAR1[0] instance 30 of an array of 32.
DEPCMDPAR0_30	0x0000C9E8	32	Read/Write	0x00000000	Register DEPCMDPAR0[0] instance 30 of an array of 32.
DEPCMD_30	0x0000C9EC	32	Read/Write	0x00000000	Register R instance 30 of an array of 32.
DEPCMDPAR2_31	0x0000C9F0	32	Read/Write	0x00000000	Register DEPCMDPAR2 instance 31 of an array of 32.
DEPCMDPAR1_31	0x0000C9F4	32	Read/Write	0x00000000	Register DEPCMDPAR1[0] instance 31 of an array of 32.
DEPCMDPAR0_31	0x0000C9F8	32	Read/Write	0x00000000	Register DEPCMDPAR0[0] instance 31 of an array of 32.
DEPCMD_31	0x0000C9FC	32	Read/Write	0x00000000	Register R instance 31 of an array of 32

Table 31-38: **OTG and Battery Charger Registers**

Register Name	Address	Width	Type	Reset Value	Description
OCFG	0x0000CC00	32	Mixed	0x00000000	OTG configuration register
OCTL	0x0000CC04	32	Mixed	0x00000040	OTG control register
OEVT	0x0000CC08	32	Mixed	0x00000000	OTG events register
OEVTEN	0x0000CC0C	32	Mixed	0x00000000	OTG events enable register
OSTS	0x0000CC10	32	Read only	0x00000819	OTG status register
ADPCFG	0x0000CC20	32	Mixed	0x00000000	ADP configuration register
ADPCTL	0x0000CC24	32	Mixed	0x00000000	ADP control register

Table 31-38: **OTG and Battery Charger Registers *(Cont'd)***

Register Name	Address	Width	Type	Reset Value	Description
ADPEVT	`0x0000CC28`	32	Mixed	`0x00000000`	ADP event register
ADPEVTEN	`0x0000CC2C`	32	Mixed	`0x00000000`	ADP event enable register

Table 31-39: **xHCI Host Registers**

Register Name	Address	Width	Type	Reset Value	Description
CAPLENGTH	`0x00000000`	32	Read only	`0x01000020`	Capability registers Length
HCSPARAMS1	`0x00000004`	32	Read only	`0x02000440`	Host controller structural parameters 1
HCSPARAMS2	`0x00000008`	32	Read only	`0x0C0000F1`	Host controller structural parameters 2
HCSPARAMS3	`0x0000000C`	32	Read only	`0x07FF000A`	Structural parameters 3 register
HCCPARAMS	`0x00000010`	32	Read only	`0x0238F665`	Capability parameters register
DBOFF	`0x00000014`	32	Read only	`0x000004E0`	Doorbell offset register
RTSOFF	`0x00000018`	32	Read only	`0x00000440`	Run time register space offset register
Rsvd_HC	`0x0000001C`	32	Read only	`0x00000000`	Register Rsvd_HC
USBCMD	`0x00000020`	32	Mixed	`0x00000000`	USB command register
USBSTS	`0x00000024`	32	Mixed	`0x00000001`	USB status register bit definitions.
PAGESIZE	`0x00000028`	32	Read only	`0x00000001`	Page size register bit definitions. This register is used by software to enable or disable the reporting of the reception of specific USB device notification transaction packets. A notification enable (Nx, where x = 0 to 15) flag is defined for each of the 16 possible device notification types. If a flag is set for a specific notification type, a device notification event is generated when the respective notification packet is received. After reset, all notifications are disabled.
DNCTRL	`0x00000034`	32	Mixed	`0x00000000`	Device notification register bit definitions.
CRCR_LO	`0x00000038`	32	Mixed	`0x00000000`	Register CRCR_LO
CRCR_HI	`0x0000003C`	32	Read/Write	`0x00000000`	Register CRCR_HI
DCBAAP_LO	`0x00000050`	32	Mixed	`0x00000000`	Register DCBAAP_LO
DCBAAP_HI	`0x00000054`	32	Read/Write	`0x00000000`	Register DCBAAP_HI
CONFIG	`0x00000058`	32	Mixed	`0x00000000`	Configure register bit definitions. 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 31-39: **xHCI Host Registers *(Cont'd)***

Register Name	Address	Width	Type	Reset Value	Description
PORTSC_0	0x00000420	32	Mixed	0x000002A0	Port status and control register bit definitions. Instance 0 of an array of two.
PORTPMSC_0	0x00000424	32	Mixed	0x00000000	USB3 port power management status and control register bit definitions. 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). Instance 0 of an array of two.
PORTLI_0	0x00000428	32	Read only	0x00000000	Port link information register instance 0 of an array of two.
PORTHLPMC_0	0x0000042C	32	Mixed	0x00000000	USB2 port LPM control register bit definitions. Instance 0 of an array of two.
PORTSC_1	0x00000430	32	Mixed	0x000002A0	Port status and control register bit definitions. Instance 1 of an array of two.
PORTPMSC_1	0x00000434	32	Mixed	0x00000000	USB3 port power management status and control register bit definitions. 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). Instance 1 of an array of two.
PORTLI_1	0x00000438	32	Read only	0x00000000	Port link information register instance 1 of an array of two.
PORTHLPMC_1	0x0000043C	32	Mixed	0x00000000	USB2 port hardware LPM control register bit definitions. Instance 1 of an array of two.
MFINDEX	0x00000440	32	Read only	0x00000000	Microframe index register bit definitions.
RsvdZ	0x00000444	32	Read only	0x00000000	Register RsvdZ
IMAN_0	0x00000460	32	Mixed	0x00000000	Interrupter management register bit definitions. Instance 0 of an array of four.

Table 31-39: **xHCI Host Registers *(Cont'd)***

Register Name	Address	Width	Type	Reset Value	Description
IMOD_0	`0x00000464`	32	Read/Write	`0x00000FA0`	Interrupter moderation 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: Instance 0 of an array of four.
ERSTSZ_0	`0x00000468`	32	Mixed	`0x00000000`	Event ring segment table size register bit definitions. Instance 0 of an array of four.
RsvdP_0	`0x0000046C`	32	Read only	`0x00000000`	Register RsvdP instance 0 of an array of four.
ERSTBA_LO_0	`0x00000470`	32	Mixed	`0x00000000`	Register ERSTBA_LO instance 0 of an array of four.
ERSTBA_HI_0	`0x00000474`	32	Read/Write	`0x00000000`	Register ERSTBA_HI instance 0 of an array of four.
ERDP_LO_0	`0x00000478`	32	Read/Write	`0x00000000`	Register ERDP_LO instance 0 of an array of four.
ERDP_HI_0	`0x0000047C`	32	Read/Write	`0x00000000`	Register ERDP_HI instance 0 of an array of four.
IMAN_1	`0x00000480`	32	Mixed	`0x00000000`	Interrupter management register bit definitions. Instance 1 of an array of four.
IMOD_1	`0x00000484`	32	Read/Write	`0x00000FA0`	Interrupter moderation 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: Instance 1 of an array of four.
ERSTSZ_1	`0x00000488`	32	Mixed	`0x00000000`	Event ring segment table size register bit definitions. Instance 1 of an array of four.

Table 31-39: **xHCI Host Registers *(Cont'd)***

Register Name	Address	Width	Type	Reset Value	Description
RsvdP_1	`0x0000048C`	32	Read only	`0x00000000`	Register RsvdP instance 1 of an array of four.
ERSTBA_LO_1	`0x00000490`	32	Mixed	`0x00000000`	Register ERSTBA_LO instance 1 of an array of four.
ERSTBA_HI_1	`0x00000494`	32	Read/Write	`0x00000000`	Register ERSTBA_HI instance 1 of an array of four.
ERDP_LO_1	`0x00000498`	32	Read/Write	`0x00000000`	Register ERDP_LO instance 1 of an array of four.
ERDP_HI_1	`0x0000049C`	32	Read/Write	`0x00000000`	Register ERDP_HI instance 1 of an array of four.
IMAN_2	`0x000004A0`	32	Mixed	`0x00000000`	Interrupter management register bit definitions. Instance 2 of an array of four.
IMOD_2	`0x000004A4`	32	Read/Write	`0x00000FA0`	Interrupter moderation 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: Instance 2 of an array of four.
ERSTSZ_2	`0x000004A8`	32	Mixed	`0x00000000`	Event ring segment table size register bit definitions. Instance 2 of an array of four.
RsvdP_2	`0x000004AC`	32	Read only	`0x00000000`	Register RsvdP instance 2 of an array of four.
ERSTBA_LO_2	`0x000004B0`	32	Mixed	`0x00000000`	Register ERSTBA_LO instance 2 of an array of four.
ERSTBA_HI_2	`0x000004B4`	32	Read/Write	`0x00000000`	Register ERSTBA_HI instance 2 of an array of four.
ERDP_LO_2	`0x000004B8`	32	Read/Write	`0x00000000`	Register ERDP_LO instance 2 of an array of four.
ERDP_HI_2	`0x000004BC`	32	Read/Write	`0x00000000`	Register ERDP_HI instance 2 of an array of four.
IMAN_3	`0x000004C0`	32	Mixed	`0x00000000`	Interrupter management register bit definitions. Instance 3 of an array of four.

Table 31-39: **xHCI Host Registers *(Cont'd)***

Register Name	Address	Width	Type	Reset Value	Description
IMOD_3	`0x000004C4`	32	Read/Write	`0x00000FA0`	Interrupter moderation 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: Instance 3 of an array of four.
ERSTSZ_3	`0x000004C8`	32	Mixed	`0x00000000`	Event ring segment table size register bit definitions. Instance 3 of an array of four.
RsvdP_3	`0x000004CC`	32	Read only	`0x00000000`	Register RsvdP instance 3 of an array of four.
ERSTBA_LO_3	`0x000004D0`	32	Mixed	`0x00000000`	Register ERSTBA_LO instance 3 of an array of four.
ERSTBA_HI_3	`0x000004D4`	32	Read/Write	`0x00000000`	Register ERSTBA_HI instance 3 of an array of four.
ERDP_LO_3	`0x000004D8`	32	Read/Write	`0x00000000`	Register ERDP_LO instance 3 of an array of four.
ERDP_HI_3	`0x000004DC`	32	Read/Write	`0x00000000`	Register ERDP_HI instance 3 of an array of four.
DB0	`0x000004E0`	32	Mixed	`0x00000000`	Doorbell register bit field definitions.
DB1	`0x000004E4`	32	Mixed	`0x00000000`	Doorbell register bit field definitions.
DB2	`0x000004E8`	32	Mixed	`0x00000000`	Doorbell register bit field definitions.
DB3	`0x000004EC`	32	Mixed	`0x00000000`	Doorbell register bit field definitions.
DB4	`0x000004F0`	32	Mixed	`0x00000000`	Doorbell register bit field definitions.
DB5	`0x000004F4`	32	Mixed	`0x00000000`	Doorbell register bit field definitions.
DB6	`0x000004F8`	32	Mixed	`0x00000000`	Doorbell register bit field definitions.
DB7	`0x000004FC`	32	Mixed	`0x00000000`	Doorbell register bit field definitions.
DB8	`0x00000500`	32	Mixed	`0x00000000`	Doorbell register bit field definitions.
DB9	`0x00000504`	32	Mixed	`0x00000000`	Doorbell register bit field definitions.
DB10	`0x00000508`	32	Mixed	`0x00000000`	Doorbell register bit field definitions.
DB11	`0x0000050C`	32	Mixed	`0x00000000`	Doorbell register bit field definitions.
DB12	`0x00000510`	32	Mixed	`0x00000000`	Doorbell register bit field definitions.
DB13	`0x00000514`	32	Mixed	`0x00000000`	Doorbell register bit field definitions.
DB14	`0x00000518`	32	Mixed	`0x00000000`	Doorbell register bit field definitions.

Table 31-39: **xHCI Host Registers *(Cont'd)***

Register Name	Address	Width	Type	Reset Value	Description
DB15	0x0000051C	32	Mixed	0x00000000	Doorbell register bit field definitions.
DB16	0x00000520	32	Mixed	0x00000000	Doorbell register bit field definitions.
DB17	0x00000524	32	Mixed	0x00000000	Doorbell register bit field definitions.
DB18	0x00000528	32	Mixed	0x00000000	Doorbell register bit field definitions.
DB19	0x0000052C	32	Mixed	0x00000000	Doorbell register bit field definitions.
DB20	0x00000530	32	Mixed	0x00000000	Doorbell register bit field definitions.
DB21	0x00000534	32	Mixed	0x00000000	Doorbell register bit field definitions.
DB22	0x00000538	32	Mixed	0x00000000	Doorbell register bit field definitions.
DB23	0x0000053C	32	Mixed	0x00000000	Doorbell register bit field definitions.
DB24	0x00000540	32	Mixed	0x00000000	Doorbell register bit field definitions.
DB25	0x00000544	32	Mixed	0x00000000	Doorbell register bit field definitions.
DB26	0x00000548	32	Mixed	0x00000000	Doorbell register bit field definitions.
DB27	0x0000054C	32	Mixed	0x00000000	Doorbell register bit field definitions.
DB28	0x00000550	32	Mixed	0x00000000	Doorbell register bit field definitions.
DB29	0x00000554	32	Mixed	0x00000000	Doorbell register bit field definitions.
DB30	0x00000558	32	Mixed	0x00000000	Doorbell register bit field definitions.
DB31	0x0000055C	32	Mixed	0x00000000	Doorbell register bit field definitions.
DB32	0x00000560	32	Mixed	0x00000000	Doorbell register bit field definitions.
DB33	0x00000564	32	Mixed	0x00000000	Doorbell register bit field definitions.
DB34	0x00000568	32	Mixed	0x00000000	Doorbell register bit field definitions.
DB35	0x0000056C	32	Mixed	0x00000000	Doorbell register bit field definitions.
DB36	0x00000570	32	Mixed	0x00000000	Doorbell register bit field definitions.
DB37	0x00000574	32	Mixed	0x00000000	Doorbell register bit field definitions.
DB38	0x00000578	32	Mixed	0x00000000	Doorbell register bit field definitions.
DB39	0x0000057C	32	Mixed	0x00000000	Doorbell register bit field definitions.
DB40	0x00000580	32	Mixed	0x00000000	Doorbell register bit field definitions.
DB41	0x00000584	32	Mixed	0x00000000	Doorbell register bit field definitions.
DB42	0x00000588	32	Mixed	0x00000000	Doorbell register bit field definitions.
DB43	0x0000058C	32	Mixed	0x00000000	Doorbell register bit field definitions.
DB44	0x00000590	32	Mixed	0x00000000	Doorbell register bit field definitions.
DB45	0x00000594	32	Mixed	0x00000000	Doorbell register bit field definitions.
DB46	0x00000598	32	Mixed	0x00000000	Doorbell register bit field definitions.
DB47	0x0000059C	32	Mixed	0x00000000	Doorbell register bit field definitions.
DB48	0x000005A0	32	Mixed	0x00000000	Doorbell register bit field definitions.
DB49	0x000005A4	32	Mixed	0x00000000	Doorbell register bit field definitions.

Table 31-39: **xHCI Host Registers *(Cont'd)***

Register Name	Address	Width	Type	Reset Value	Description
DB50	0x000005A8	32	Mixed	0x00000000	Doorbell register bit field definitions.
DB51	0x000005AC	32	Mixed	0x00000000	Doorbell register bit field definitions.
DB52	0x000005B0	32	Mixed	0x00000000	Doorbell register bit field definitions.
DB53	0x000005B4	32	Mixed	0x00000000	Doorbell register bit field definitions.
DB54	0x000005B8	32	Mixed	0x00000000	Doorbell register bit field definitions.
DB55	0x000005BC	32	Mixed	0x00000000	Doorbell register bit field definitions.
DB56	0x000005C0	32	Mixed	0x00000000	Doorbell register bit field definitions.
DB57	0x000005C4	32	Mixed	0x00000000	Doorbell register bit field definitions.
DB58	0x000005C8	32	Mixed	0x00000000	Doorbell register bit field definitions.
DB59	0x000005CC	32	Mixed	0x00000000	Doorbell register bit field definitions.
DB60	0x000005D0	32	Mixed	0x00000000	Doorbell register bit field definitions.
DB61	0x000005D4	32	Mixed	0x00000000	Doorbell register bit field definitions.
DB62	0x000005D8	32	Mixed	0x00000000	Doorbell register bit field definitions.
DB63	0x000005DC	32	Mixed	0x00000000	Doorbell register bit field definitions.
USBLEGSUP	0x000008E0	32	Mixed	0x00000401	Register USBLEGSUP
USBLEGCTLSTS	0x000008E4	32	Mixed	0x00000000	Register USBLEGCTLSTS
SUPTPRT2_DW0	0x000008F0	32	Read only	0x02000402	Register SUPTPRT2_DW0
SUPTPRT2_DW1	0x000008F4	32	Read only	0x20425355	Register SUPTPRT2_DW1
SUPTPRT2_DW2	0x000008F8	32	Read only	0x00180101	xHCI supported protocol capability, data word 2
SUPTPRT2_DW3	0x000008FC	32	Read only	0x00000000	Register SUPTPRT2_DW3
SUPTPRT3_DW0	0x00000900	32	Read only	0x03000002	Register SUPTPRT3_DW0
SUPTPRT3_DW1	0x00000904	32	Read only	0x20425355	Register SUPTPRT3_DW1
SUPTPRT3_DW2	0x00000908	32	Read only	0x00000102	Register SUPTPRT3_DW2
SUPTPRT3_DW3	0x0000090C	32	Read only	0x00000000	Register SUPTPRT3_DW3
DCID	0x00000910	32	Read only	0x000F000A	Register DCID
DCDB	0x00000914	32	Mixed	0x00000000	Register DCDB
DCERSTSZ	0x00000918	32	Mixed	0x00000000	Register DCERSTSZ
DCERSTBA_LO	0x00000920	32	Mixed	0x00000000	Register DCERSTBA_LO
DCERSTBA_HI	0x00000924	32	Read/Write	0x00000000	Register DCERSTBA_HI
DCERDP_LO	0x00000928	32	Read/Write	0x00000000	Register DCERDP_LO
DCERDP_HI	0x0000092C	32	Read/Write	0x00000000	Register DCERDP_HI
DCCTRL	0x00000930	32	Mixed	0x000F0000	Register DCCTRL
DCST	0x00000934	32	Read only	0x00000000	Register DCST
DCPORTSC	0x00000938	32	Mixed	0x00000000	Register DCPORTSC

Table 31-39: **xHCI Host Registers *(Cont'd)***

Register Name	Address	Width	Type	Reset Value	Description
DCCP_LO	0x00000940	32	Mixed	0x00000000	Register DCCP_LO
DCCP_HI	0x00000944	32	Read/Write	0x00000000	Register DCCP_HI
DCDDI1	0x00000948	32	Read/Write	0x00000000	Register DCDDI1
DCDDI2	0x0000094C	32	Read/Write	0x00000000	Register DCDDI2

The groups of register map is shown in the following table.

Table 31-40: **Groups of Register Map**

Register	Description
0x04E0	Doorbell registers (default DBOFF = 0x4E0)
0x0460	Interrupt registers
0x0440	Run time register set (default RTSOFF = 0x440)
0x0420	Port register set
0x0020	xHCI operational registers (default CAPLENGTH = 0x20)
0x0000	xHCI base address
0x0000_0000	Capability registers length (CAPLENGTH)
0x0000_0014	Doorbell offset register (DBOFF)
0x0000_0018	Runtime register space offset register (RTSOFF)
0x0000_04E0 - 0x0000_05DC	Doorbell register
0x0000_C100 - 0x0000_C630	Global registers
0x0000_C700 - 0x0000_C8BC	Device registers
0x0000_CC00 - 0x0000_CC10	OTG registers

Table 31-41: **USB MIO Pins**

Signal Name	MIO Pin/ net name in schematic	ULPI PIN	Description
USB-0 MIO Connections			
ULPI0_CLK	MIO52_ ULPI0_CLK	1	Reference clock
ULPI0_DATA0	MIO56_ ULPI0_DATA0	3	Data bit 0
ULPI0_DATA1	MIO57_ ULPI0_DATA1	4	Data bit 1
ULPI0_DATA2	MIO54_ ULPI0_DATA2	5	Data bit 2
UPLI0_DATA3	MIO59_ ULPI0_DATA3	6	Data bit 3
ULPI0_DATA4	MIO60_ ULPI0_DATA4	7	Data bit 4
ULPI0_DATA5	MIO61_ ULPI0_DATA5	9	Data bit 5
ULPI0_DATA6	MIO62_ ULPI0_DATA6	10	Data bit 6
ULPI0_DATA7	MIO63_ ULPI0_DATA7	13	Data bit 7
ULPI0_DIR	MIO53_ ULPI0_DIR	31	Data bus direction
ULPI0_NXT	MIO55_ ULPI0_NXT	2	Send next data byte
ULPI0_STP	MIO58_ ULPI0_STP	29	Stop transmission, last data byte
ULPI0_RST_N	PS_MODE1	27	Reset to USB PHY

USB_SSTX_P	GT0_ USB0_SSTX_P		Terminated and connected to USB Connector
USB_SSTX_N	GT0_ USB0_SSTX_N		Terminated and connected to USB Connector
USB_SSRX_P	GT0_ USB0_SSRX_P		
USB_SSRX_N	GT0_ USB0_SSRX_N		
USB-1 MIO Connections			
ULPI1_CLK	MIO64_ ULPI1_CLK	1	Reference clock
ULPI1_DATA0	MIO68_ULPI1_DATA0	3	Data bit 0
ULPI1_DATA1	MIO69_ ULPI1_DATA1	4	Data bit 1
ULPI1_DATA2	MIO66_ULPI1_DATA2	5	Data bit 2
UPLI1_DATA3	MIO71_ULPI1_DATA3	6	Data bit 3
ULPI1_DATA4	MIO72_ ULPI1_DATA4	7	Data bit 4
ULPI1_DATA5	MIO73_ ULPI1_DATA5	9	Data bit 5
ULPI1_DATA6	MIO74 _ULPI1_DATA6	10	Data bit 6
ULPI1_DATA7	MIO75_ULPI1_DATA7	13	Data bit 7
ULPI1_DIR	MIO65_ULPI1_DIR	31	Data bus direction
ULPI1_NXT	MIO67_ ULPI1_NXT	2	Send next data byte
ULPI1_STP	MIO70_ ULPI1_STP	29	Stop transmission, last data byte
ULPI1_RST_N	PS_MODE2	27	Reset to USB PHY
USB_SSTX_P	GT3_ USB1_SSTX_P		Terminated and connected to USB Connector
USB_SSTX_N	GT3_ USB1_SSTX_N		Terminated and connected to USB Connector
USB_SSRX_P	GT3_ USB1_SSTX_P		
USB_SSRX_N	GT3_ USB1_SSTX_N		

Chapter 32

SATA Controller

Introduction

The serial ATA (SATA) protocol was designed to replace the old parallel ATA (or IDE) interface used mainly for storage devices. SATA uses the ATA/ATAPI command-set, but uses serial communication over the differential wire pairs at rates of 1.5, 3.0, or 6.0 Gb/sec corresponding to SATA generation 1, generation 2 or generation 3. The serial data is 8B/10B encoded which ensures sufficient transition in the data pattern to ensure DC balancing and enables the clock data recovery circuit to extract the clock from the incoming data pattern.

The SATA controller is a high-performance dual-port SATA host controller with an AHCI compliant command layer which supports advanced features such as native command queuing and frame information structure (FIS) based switching for systems employing port multipliers.

Features

- SATA host port supporting two external devices.
- Designed to be compliant with SATA 3.1 specifications.
- Compliant with the advanced host controller interface (AHCI) version 1.3.
- Supports 1.5, 3.0, and 6.0 Gb/s data rates.
- 64-bit AXI master port with a built-in DMA with 40/44-bit addressing.
- Configuration done through register programming of these register sets:
    - SATA_AHCI_HBA
    - SATA_AHCI_VENDOR
    - SATA_AHCI_PORTCNTRL
    - SERDES (PS-GTR)
    - FPD_GPV (AXI Interconnect)
    - SIOU (clock)
- Power management features: support partial and slumber modes.

- Supports hot-plug detect feature.

IMPORTANT: *The current characterization of SATA Gen3 return loss shows a marginal violation of the specification in the band of frequencies between 800MHz-1GHz.*

Functional Description

The SATA host controller is responsible for implementing the physical, link, transport and command layer functions as described in SATA rev 3.1 specification for a two port host device.

System Viewpoint

Figure 32-1 shows the SATA host controller system.

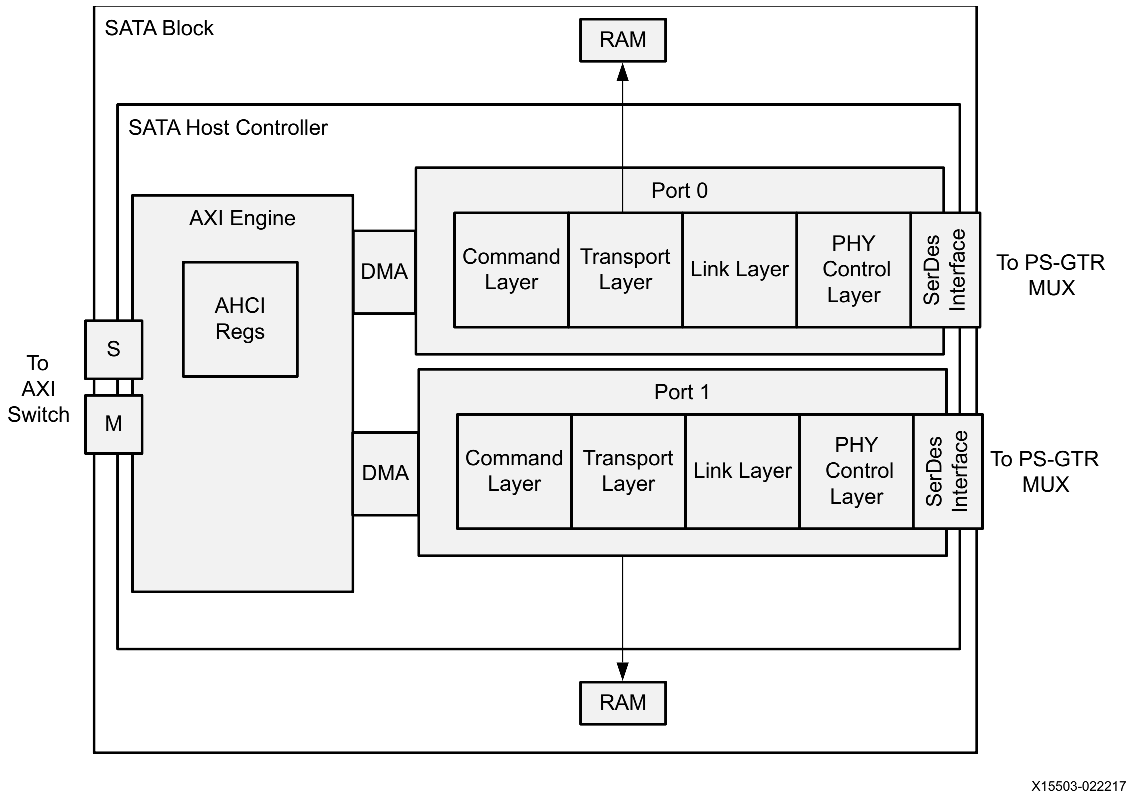


Figure 32-1: **SATA System Block Diagram**

Description

The SATA controller has the following primary interfaces.

- The AXI slave interface is a 64-bit AXI slave interface with a 12-bit AXI ID and a 46-bit AXI address. This interface has a maximum burst length of one. A burst with a burst size more than one results in an AXI error. This restriction does not affect performance during normal operation.
- The interrupt interface is used to signal interrupts to the host CPU. The interrupt controller has one bit assigned to the SATA block that is connected to the GIC. The interrupt signal routed to the GIC is a WIRE-OR output of the PORT0 interrupt, the PORT1 interrupt, and the command coalescing channel interrupt. In the interrupt separate mode, each interrupt output from PORT0, PORT1, and the command coalescing blocks are routed to the GIC. Refer to Table 13-4 for the interrupt ID of the SATA host controller block. Software must not enable the interrupt separation mode because it is not supported on the device.
- The AXI master interface is used by the block to perform DMA operations for moving data between the host memory (example DDR) and SATA device (example hard drive). This interface has ability to initiate burst between 1 and 16 cycles. The master interface has 4-bit AXI ID buses that can take on one of four (programmable) ID values that are out of reset default to 0, 1, 2, and 3. The IDs are configured through the SATA_AHCI_VENDOR.PAXIC port register. The block can issue up to 16 read and write transactions through the AXI master interface. Out of reset, the maximum number of outstanding transactions (issuing ability) defaults to four reads and four writes, and it can be changed through the port AXI CFG register. This port only generates incremental bursts with lengths of 1, 4, 8, or 16 beats. The burst length is selected by the block based on FIFO fill levels and is not controllable by user. The AxCACHE bits can be controlled through the AXI cache control register. The AxPROT bits are controllable through the FPD_SLCR_SECURE register set as well as through the security (TrustZone) bus.

The SATA controller performs 8B/10B encoding and decoding functionality and uses 20-bit parallel interface to the PS-GTR block. The following sections describe each of the layers implemented as part of the SATA protocol stack.

Command Layer

The operation of the command layer is defined by the AHCI specification. The SATA controller implements the following functionality in addition to what is required by the AHCI specification.

Local Port Context Management

When the AHCI controller is connected to a port multiplier supporting FIS-based switching a local context store can be enabled to avoid the process of lookup of the related memory addressing for data transactions. This feature facilitates quick context switching and allows the AHCI to operate with multiple devices in a seamless manner. Each context stores information about the memory address and SATA block address of any executing command on the first four ports of a port multiplier.

Vendor Specific BIST Operation

As part of the host self-diagnostic operation, a vendor specific BIST mode is supported. This mode, in conjunction with SIOU's serial loop-back, allow for the test of the host controller operation. When programmed, the host fetches a command from the memory loop that manages the FIS through the transport and link layers and then posts the payload to the receive FIS area for checking. The mode exercises the following paths.

- DMA controller FIS transmission.
- Command layer FIS transmission.
- Transport layer TX FIFO FIS transmission.
- Link layer FIS transmission.
- PHY modes.
- Link layer FIS reception.
- Transport layer RXFIFO and FIS reception.
- Command layer FIS reception.
- Host DMA controller FIS reception.

When the host controller indicates the command is complete, the software examines the contexts of the command descriptors statistics field. If the pre-programmed values are present, the test has passed.

The command list structure is shown in Figure 32-2. To support the vendor BIST operation, the command header structure is modified. The reserved bit 11 of DW0 is now designated as VBIST, setting this bit indicates the associated command used as the payload for a vendor BIST operation.

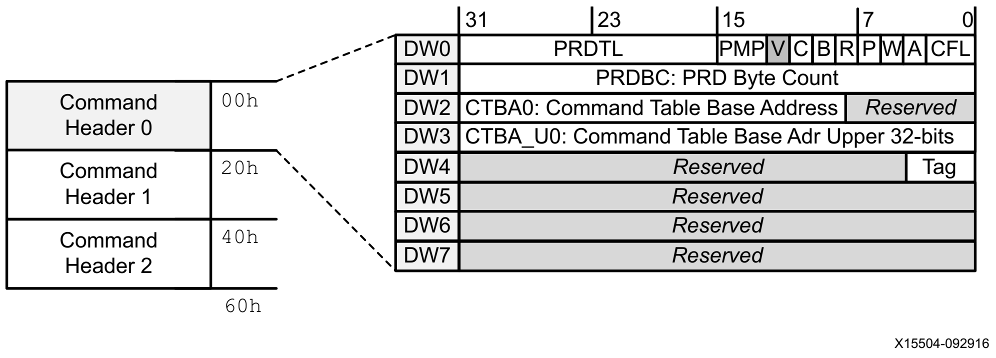


Figure 32-2: **Command List Structure**

Transport Layer

The function of the SATA transport layer is to interface between the command and link layers in the transmission and reception of the frame information structures (FIS).

On transmit, the transport layer frames the FIS placed into the TX FIFO. The FISs are framed based on a programmed length for non-data FIS and or a configurable length for data FIS. When the transport layer is instructed to send a non-data FIS, it employs a retry policy until the far end signals acceptance of the transmitted FIS.

On reception, the transport layer de-frames the FIS and places them into the RX FIFO. When a FIS is received, the transport layer informs the command layer.

For a non-data FIS the FIS is considered received when the EOF is signaled by the link layer and the FIS is received with a good CRC.

For a short vendor-specific FIS, the FIS is considered as a non-data FIS. For longer vendor-specific FIS, the FIS reception is signaled when the RX FIFO reaches its watermark.

For a data FIS, the FIS is considered received when the first double word (header) is written into the FIFO.

The transport layer is responsible for crossing the clock domain between the transport layer txDouble word and rxDouble word clocks and command layer clock domain. The receive FIFO is written to on the transport layer receive double-word clock with data contained in the FIS sent by the link layer. Once the data is stable at the output of the receive FIFO, on the command layer clock domain, the command layer can take the data. If the command layer is not ready to accept the data, the data builds up in the receive FIFO. When the receive FIFO exceeds its threshold, the transport layer stalls the link layer, which sends HOLD primitives to the far end to stall it. This threshold takes into consideration the latency involved in getting the far end to stop transmitting the data. This threshold is programmable to allow for the use of high latency repeaters or re-timers in between the host and device.

The transmit FIFO is written to on the command layer clock, with data to be sent in the FIS transferred by the DMA controller. Once the data is stable at the output of the transmit FIFO on the transmit double word clock domain, the link layer can take the data. If the transmit FIFO cannot supply data to the link layer, the transport layer stalls the link layer, which sends HOLD primitives to the far end to stall it.

Link Layer

The function of the SATA link layer is to interface between the transport and PHY control layers in the transmission and reception of frames and primitives. The link layer utilizes the two unidirectional links provided by the SATA interface to maintain coordinated communication between the host and the device. Payload data can only be transmitted in one direction at a time.

The link layer can work at SATA generation 1 (1.5 Gb/s), generation 2 (3.0 Gb/s) and generation 3 (6.0 Gb/s) speeds. For 1.5 Gb/s operation it must be clocked with a 37.5 MHz clock derived from the receive side of the SATA PHY, for 3.0 Gb/s operation it is clocked with a similarly derived 75 MHz clock and for 6.0 Gb/s operation, this becomes 150 MHz.

The data flow for the transmit side is shown in Figure 32-3. The data flow for the receive side is shown in Figure 32-4.

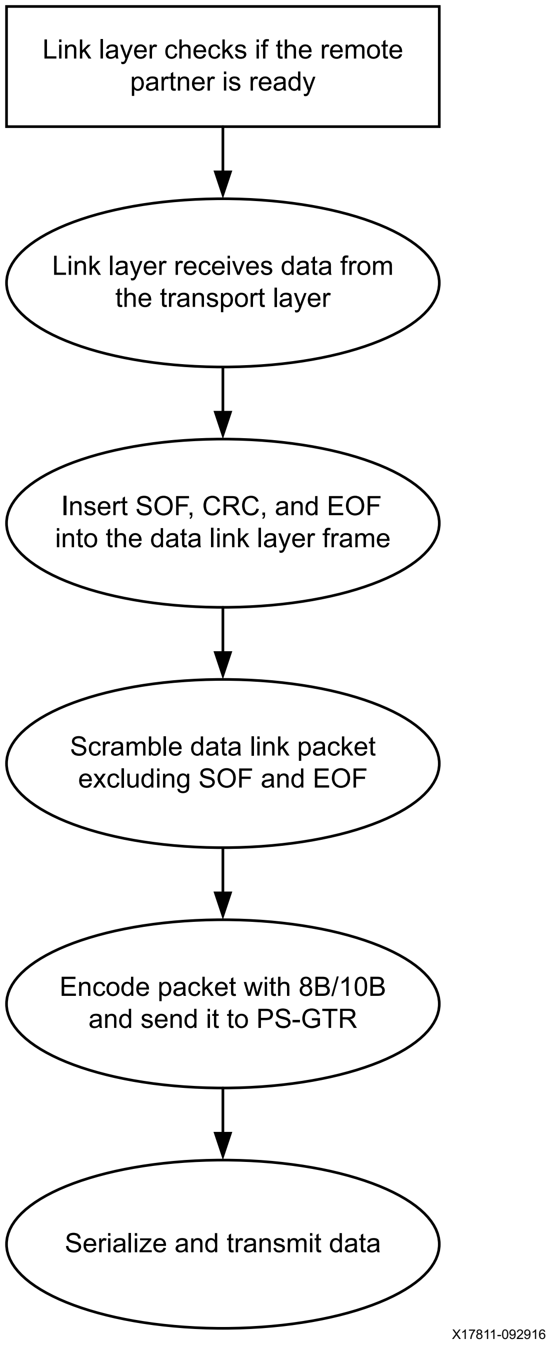

Link layer checks if the remote partner is ready
Link layer receives data from the transport layer
Insert SOF, CRC, and EOF into the data link layer frame
Scramble data link packet excluding SOF and EOF
Encode packet with 8B/10B and send it to PS-GTR
Serialize and transmit data
X17811-092916

Figure 32-3: **Transmit Side Data Flow**

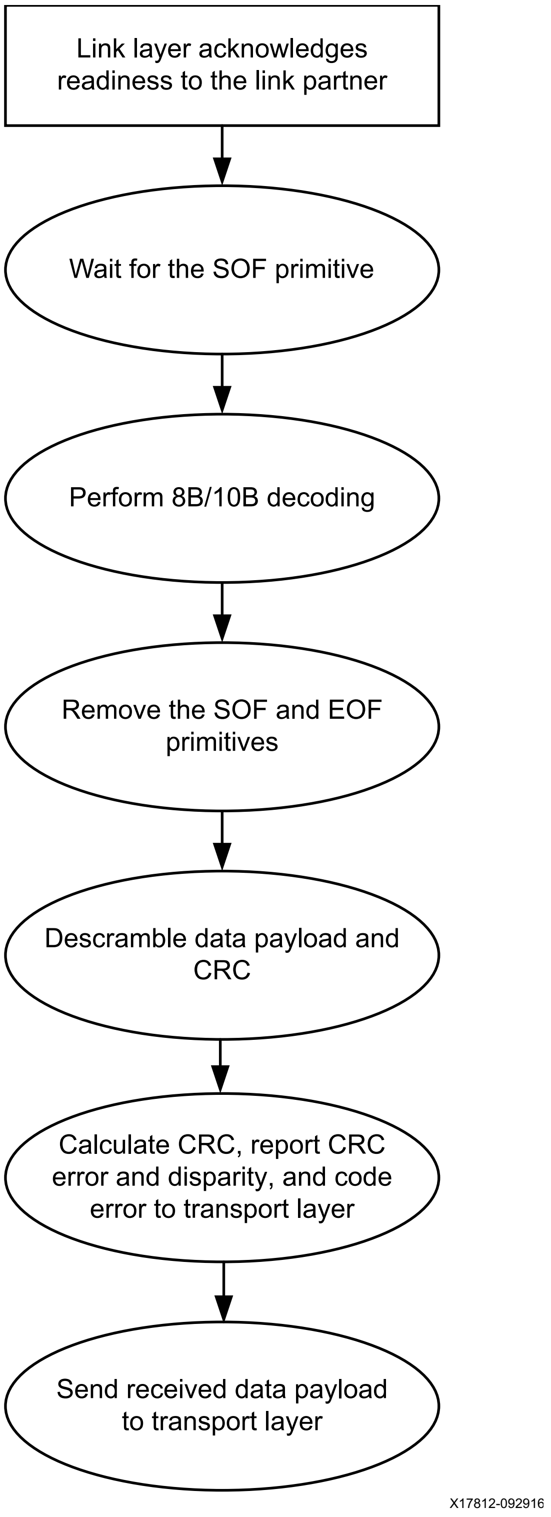

Link layer acknowledges readiness to the link partner
Wait for the SOF primitive
Perform 8B/10B decoding
Remove the SOF and EOF primitives
Descramble data payload and CRC
Calculate CRC, report CRC error and disparity, and code error to transport layer
Send received data payload to transport layer
X17812-092916

Figure 32-4: **Receive Side Data Flow**

The link layer also partakes in flow control between the local and remote ends. The layer supports flow control actions based on the local FIFO status (which is located in the transport layer), or in response to receiving flow control messages from the remote end.

The transmit side of the link layer is also responsible for inserting a pair of ALIGN primitives every 254 double words, or more frequently as you can program the frequency.

PHY Control Layer

The PHY control layer operates between the PS-GTR and link layers. The main functions of the PHY control layer are listed.

- Data path operation.
- RX data path.
- TX data path.
- PHY initialization state machine.
- Out-of-band processing.
- Speed negotiation.

On receive, the PHY control layer converts the encoded 20-bit parallel data from the PS-GTR to a 32-bit double word, which it presents to the link layer. The PHY control layer aligns the control word of the SATA primitive to the lowest word position of the double word. The PHY control contains an 8B/10B decoder function and decodes the incoming data into data, control/data and code, or disparity error. The PHY controller sends data, control/data and code, or disparity error to the PS-GTR.

On transmit, the PHY control layer takes in the 32-bit transmit data from the link layer and converts the data into encoded 20-bit parallel data to the PS-GTR. The control/data bit from the link layer (which is always assumed to be associated with the lowest byte position of the transmit double word) is also passed onto the PS-GTR with the appropriate word. The PHY control layer takes the transmit word clock output by the PS-GTR and converts it to a double word transmit clock which it sends to the link layer.

TrustZone Support

The SATA block is capable of enforcing TrustZone security scheme on both AXI interfaces.

AXI Master Port Security Features

The AXI Master port is capable of driving the AWPROT and ARPROT bits with a programmable value controlled from the TrustZone configuration register, FPD_SLCR_SECURE.slcr_sata. When the [tz_en] bit is set to 1, the TrustZone security for the slave port is determined by the [tz_axidma{0, 1}] bits. A value of 1 indicates TrustZone is enabled for the AHCI interface master port. A value of 0 indicates TrustZone is disabled for the master port.

The value of AWPROT can be independently controlled for the following transfers.

- Status FIS transfers.
- Intermediate data burst of a data transfer.
- Final data burst of a data transfer.

The value of ARPROT can be independently controlled for the following transfers.

- Posting PRD read-address to memory controller.
- Posting header read address to the memory controller
- Posting command FIS read address to memory controller
- Posting data burst read address to the memory controller

AXI Slave Port Security Features

The security level of the slave port is controlled from the TrustZone register configuration (fpd_slcr_secure.SLCR_SATA). A value of `1` in the [tz_en] bit means that the TrustZone function is enabled for the AXI slave port. When [tz_en] is set to 1, the TrustZone security for the slave port is determined by [tz_axis] bit. A value of 1 in [tz_axis] indicates TrustZone is enabled for the AHCI interface slave port. A value of 0 indicates TrustZone is disabled for the slave port.

If the AXI slave port is given a secure status by the security controller (source of the TrustZone bus) and an AXI access targets the slave port with a non-secure security level, an AXI slave error is reported and a maskable interrupt is signaled.

SATA Clocking and Reset

The AXI interface clock can be configured using the crf_apb.SATA_REF_CTRL register. For more details on AXI interface clocking, refer to Chapter 37, PS Clock Subsystem.

The clocks used between the SATA host controller and PS-GTR transceiver are derived from the reference clock used in the serial input output unit (SIOU). For more details, refer to the PS-GTR Transceivers in Chapter 29.

Follow these steps when generating the AXI interface clock for the SATA controller.

1. To avoid a performance impact when configuring for SATA generation 2 and generation 3, choose a frequency around 200 MHz (lower than 250 MHz).
2. For other configurations, choose a frequency near 100 MHz.

The block level reset to the SATA block is controlled by the crf_apb.RST_FPD_TOP[sata_reset] bit.

Register Overview

The SATA host controller implements control and configuration registers in the vendor specific space starting offset `0x0A0`. Table 32-1 summarizes the registers that have been implemented in the SATA host bus adapter (SATA_AHCI_HBA register set).

Table 32-1: **SATA Host Bus Adapter Memory Registers**

Register Type	Register Name	Address	Description
SATA Host Bus Adapter	CAP	`0xFD0C0000`	HBA capabilities.
	GHC	`0xFD0C0004`	Global HBA control.
	IS	`0xFD0C0008`	Interrupt status.
	PI	`0xFD0C000C`	Ports implemented.
	VS	`0xFD0C0010`	AHCI version.
	CCC_CTL	`0xFD0C0014`	Command completion coalescing control.
	CCC_PORTS	`0xFD0C0018`	Command completion coalescing ports.
	EM_LOC	`0xFD0C001C`	Enclosure management location.
	EM_CTL	`0xFD0C0020`	Enclosure management control.
	CAP2	`0xFD0C0024`	HBA capabilities extended.
	BOHC	`0xFD0C0028`	BIOS/OS handoff control and status.

Table 32-2 shows SATA AHCI ports 0 and 1 control registers (SATA_AHCI_PORTCNTRL register set).

Table 32-2: **SATA AHCI Ports 0 and 1 Registers**

Register Type	Register Name	Address	Description
SATA AHCI Port 0	PxCLB	`0xFD0C0100` `0xFD0C0180`	Port 0 and 1 command list base address.
	PxCLBU	`0xFD0C0104` `0xFD0C0184`	Port 0 and 1 command list base address upper 32 bits.
	PxFB	`0xFD0C0108` `0xFD0C0188`	Port 0 and 1 FIS base address.
	PxFBU	`0xFD0C010C` `0xFD0C018C`	Port 0 and 1 FIS base address upper 32 bits.
	PxIS	`0xFD0C0110` `0xFD0C0190`	Port 0 and 1 interrupt status.
	PxIE	`0xFD0C0114` `0xFD0C0194`	Port 0 and 1 interrupt enable.
	PxCMD	`0xFD0C0118` `0xFD0C0198`	Port 0 and 1 command and status.
	PxTFD	`0xFD0C0120` `0xFD0C01A0`	Port 0 and 1 task file data.
	PxSIG	`0xFD0C0124` `0xFD0C01A4`	Port 0 and 1 signature.
	PxSSTS	`0xFD0C0128` `0xFD0C01A8`	Ports 0 and 1 serial ATA status (SCR0: Sstatus).
	PxSCTL	`0xFD0C012C` `0xFD0C01AC`	Ports 0 and 1 serial ATA control (SCR2: SControl).
	PxSERR	`0xFD0C0130` `0xFD0C01B0`	Ports 0 and 1 serial ATA error (SCR1: SError) and diagnostics.
	PxSACT	`0xFD0C0134` `0xFD0C01B4`	Ports 0 and 1 serial ATA active (SCR3: SActive).
	PxCI	`0xFD0C0138` `0xFD0C01B8`	Ports 0 and 1 command issue.
	PxSNTF	`0xFD0C013C` `0xFD0C01BC`	Ports 0 and 1 serial ATA notification (SCR4: SNotification).
	PxFBS	`0xFD0C0140` `0xFD0C01C0`	Ports 0 and 1 FIS-based switching control.
	PxDEVSLP	`0xFD0C0144` `0xFD0C01C4`	Ports 0 and 1 device sleep.
	PxBERR	`0xFD0C0170` `0xFD0C01F0`	Ports 0 and 1 BIST error.
	PxCMDS	`0xFD0C0174` `0xFD0C01F4`	Ports 0 and 1 command status error.

Table 32-3 shows the vendor-specific registers of the SATA controller (SATA_AHCI_VENDOR register set).

Table 32-3: **Vendor Specific Registers**

Register Type	Register Name	Address	Description
Vendor specific registers	PCTRL	`0xFD0C00A0`	Port PS-GTR control.
	PCFG	`0xFD0C00A4`	Port configuration. Dual-lane port select, timer scalars, interrupt separation.
	PPCFG	`0xFD0C00A8`	Port PHY configuration: control layer.
	PP2C	`0xFD0C00AC`	Port PHY configuration 2 (Phy2Cfg): OOB timing for COMMINIT.
	PP3C	`0xFD0C00B0`	Port PHY Configuration 3 (Phy3CFg): OOB timing for the COMMINIT.
	PP4C	`0xFD0C00B4`	Port PHY Configuration 4: burst timing in COM.
	PP5C	`0xFD0C00B8`	Port PHY Configuration 5: retry, interval time.
	AXICC	`0xFD0C00BC`	AXI cache control.
	PAXIC	`0xFD0C00C0`	AXI configuration.
	AXIPC	`0xFD0C00C4`	AXI PROT control.
	PTC	`0xFD0C00C8`	Port transfer configuration (TransCfg): transport layer.
	PTS	`0xFD0C00CC`	Port transport layer status (TransStatus).
	PLC	`0xFD0C00D0`	Port link-layer configuration 0 (LinkCfg).
	PLC1	`0xFD0C00D4`	Port link-layer configuration 1 (LinkCfg1).
	PLC2	`0xFD0C00D8`	Port link-layer configuration 2 (LinkCfg2).
	PLS	`0xFD0C00DC`	Port link-layer status 0.
	PLS1	`0xFD0C00E0`	Port link-layer status 1.
	PCMDC	`0xFD0C00E4`	Port command configuration.
	PPCS	`0xFD0C00E8`	Port Phy status: PhyControlStatus.
	AMS	`0xFD0C00EC`	AXI master status.
	TCR	`0xFD0C00F0`	Timer control.

Programming Considerations

This section defines the programming flow for SATA controller. The following figure shows the flow diagram.

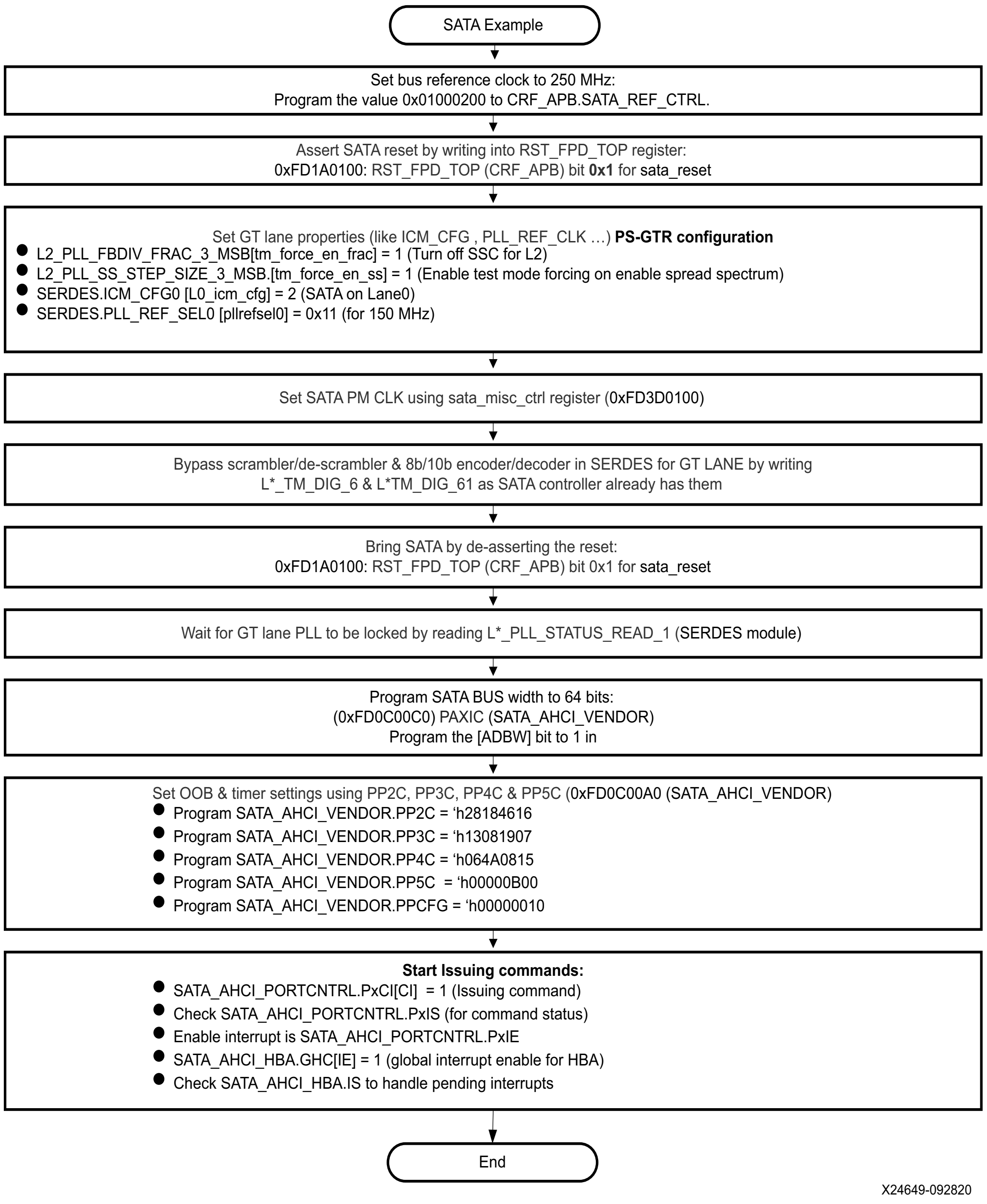

SATA Example
Set bus reference clock to 250 MHz:
Program the value 0x01000200 to CRF_APB.SATA_REF_CTRL.
Assert SATA reset by writing into RST_FPD_TOP register:
0xFD1A0100: RST_FPD_TOP (CRF_APB) bit **0x1** for sata_reset
Set GT lane properties (like ICM_CFG , PLL_REF_CLK …) **PS-GTR configuration**
● L2_PLL_FBDIV_FRAC_3_MSB[tm_force_en_frac] = 1 (Turn off SSC for L2)
● L2_PLL_SS_STEP_SIZE_3_MSB.[tm_force_en_ss] = 1 (Enable test mode forcing on enable spread spectrum)
● SERDES.ICM_CFG0 [L0_icm_cfg] = 2 (SATA on Lane0)
● SERDES.PLL_REF_SEL0 [pllrefsel0] = 0x11 (for 150 MHz)
Set SATA PM CLK using sata_misc_ctrl register (0xFD3D0100)
Bypass scrambler/de-scrambler & 8b/10b encoder/decoder in SERDES for GT LANE by writing
L*_TM_DIG_6 & L*TM_DIG_61 as SATA controller already has them
Bring SATA by de-asserting the reset:
0xFD1A0100: RST_FPD_TOP (CRF_APB) bit 0x1 for sata_reset
Wait for GT lane PLL to be locked by reading L*_PLL_STATUS_READ_1 (SERDES module)
Program SATA BUS width to 64 bits:
(0xFD0C00C0) PAXIC (SATA_AHCI_VENDOR)
Program the [ADBW] bit to 1 in
Set OOB & timer settings using PP2C, PP3C, PP4C & PP5C (0xFD0C00A0 (SATA_AHCI_VENDOR)
● Program SATA_AHCI_VENDOR.PP2C = 'h28184616
● Program SATA_AHCI_VENDOR.PP3C = 'h13081907
● Program SATA_AHCI_VENDOR.PP4C = 'h064A0815
● Program SATA_AHCI_VENDOR.PP5C = 'h00000B00
● Program SATA_AHCI_VENDOR.PPCFG = 'h00000010
Start Issuing commands:
● SATA_AHCI_PORTCNTRL.PxCI[CI] = 1 (Issuing command)
● Check SATA_AHCI_PORTCNTRL.PxIS (for command status)
● Enable interrupt is SATA_AHCI_PORTCNTRL.PxIE
● SATA_AHCI_HBA.GHC[IE] = 1 (global interrupt enable for HBA)
● Check SATA_AHCI_HBA.IS to handle pending interrupts
End
X24649-092820

Figure 32-5: **SATA Controller Programming Flow**

The following sections define the individual steps that need to be carried out.

SATA Clock Programming

Table 32-4: **Program SATA Clock**

Task	CRF_APB Register Set	Bit Field	Register Offset	Bits	Value
Program for 250 MHz IOPLL source	SATA_REF_CTRL	[CLKACT], [DIVISOR0], [SRCSEL]	`0xA0`	24 | 13:8 | 2:0	`0x010200`

SATA AXI Bus Configuration

Table 32-5: **Configure SATA AXI Bus**

Task	SATA_AHCI_VENDOR Register Set	Bit Field	Register Offset	Bits	Value
Select 64-bit bus width	PAXIC	[ADBW]	`0xC0`	1:0	`2b'01`

PS-GTR Configuration

Table 32-6: **Configure PS-GTR**

Task	SERDES Register Set	Bit Field	Register Offset	Bits	Value
SSC turn off for L0	L0_PLL_FBDIV_FRAC_3_MSB	[tm_force_en_frac]	`0x2360`	6	`1b'1`
SSC turn off for L0	L0_PLL_SS_STEP_SIZE_3_MSB	[tm_force_en_ss]	`0x237C`	7	`1b'1`
SSC turn off for L1	L1_PLL_FBDIV_FRAC_3_MSB	[tm_force_en_frac]	`0x6360`	6	`1b'1`
SSC turn off for L1	L1_PLL_SS_STEP_SIZE_3_MSB	[tm_force_en_ss]	`0x637C`	7	`1b'1`
SSC turn off for L2	L2_PLL_FBDIV_FRAC_3_MSB	[tm_force_en_frac]	`0xA360`	6	`1b'1`
SSC turn off for L2	L2_PLL_SS_STEP_SIZE_3_MSB	[tm_force_en_ss]	`0xA37C`	7	`1b'1`
SSC turn off for L3	L3_PLL_FBDIV_FRAC_3_MSB	[tm_force_en_frac]	`0xE360`	6	`1b'1`
SSC turn off for L3	L3_PLL_SS_STEP_SIZE_3_MSB	[tm_force_en_ss]	`0xE37C`	7	`1b'1`
For PORT0					
Select lane0 for SATA0	ICM_CFG0	[L0_icm_cfg]	`0x0010`	2:0	`3b'010`
PM clock frequency selection for 150 MHz	PLL_REF_SEL0	[pllrefsel0]	`0x0000`	4:0	`0x11`
PM clock frequency selection for 300 MHz			`0x0000`	4:0	`0x01`

PHY Configuration

Table 32-7: **PHY Configuration**

Task	Register	Bit Field	Register Offset	Bits	Value
To select port 0	SATA_AHCI_VENDOR.PCFG	[PAD]	`0x00A4`	5:0	`5b'00010`
PHY control OOB timing for the COMINIT parameters	SATA_AHCI_VENDOR.PP2C	ALL	`0X00AC`	31:0	`'h2818_4616`
PHY control OOB timing for the COMWAKE parameters	SATA_AHCI_VENDOR.PP3C	ALL	`0x00B0`	31:0	`'h1308_1907`
PHY control burst timing for the COM parameters	SATA_AHCI_VENDOR.PP4C	ALL	`0x00B4`	31:0	`'h064A_0815`
PHY control retry Interval timing	SATA_AHCI_VENDOR.PP5C	[RCT]	`0X00b8`	31:20	`'hB00`
Set host target speed	SATA_AHCI_PORTCNTRL.PxSCTL	[IPM], [SPD]	`0x012C`	11:4	`'h33`
Clear errors	SATA_AHCI_PORTCNTRL.PxSERR	ALL	`0x0130`	31:0	`'hFFFF_FFFF`

AHCI SATA Configuration

Table 32-8: **AHCI SATA Configuration**

Task	Register	Bit Field	Register Offset	Bits	Value
Program command list base address	SATA_AHCI_PORTCNTRL.PxCLB	[CLB]	`0x0100`	31:10	Address of CLB data structure
Program FIS base address	SATA_AHCI_PORTCNTRL.PxFB	[FB]	`0x0108`	31:8	Address of FIS data structure
Enable FIS receive	SATA_AHCI_PORTCNTRL.PxCMD	[FRE]	`0x0118`	4	`'b1`
Wait until [CR] (bit 15) bit set in register SATA_AHCI_PORTCNTRL.PxCMD to make sure no command list is running.					
Start command processing	SATA_AHCI_PORTCNTRL.PxCMD	[ST]	`0x0118`	0	`b'1`

Issuing Command

Table 32-9: **Issuing Command**

Task	SATA_AHCI_PORTCNTRL Register Set	Register Field	Register Offset	Bits	Value
Check if the command slot is free	PxCI	[CI]	`0x0138`	31:0	Read operation
Wait until the required command slot bit becomes 0.					
Set the slot bits	PxCI	[CI]	`0x0138`	31:0	Write `1'b1` (for command slot0)
Check if the command slot is free	PxCI	[CI]	`0x0138`	31:0	Read operation
Wait until the required command slot bit becomes 0 (to ensure the completion of the command).					

Basic Steps When Building a Command

When software builds a command for the HBA to execute, it first finds an empty command slot by reading the PxCI and PxSACT registers for the port. An empty command slot has its respective bit cleared to `0` in both the PxCI and PxSACT registers. After a free slot (pFreeSlot notation), is found:

- Software builds a command frame information structure (FIS) in system memory at location PxCLB[CH(pFreeSlot)]:CFIS with the command type.
- If it is an ATAPI command, the ACMD field is filled in with the ATAPI command.
- Software builds a command header at PxCLB[CH(pFreeSlot)] with:
  - PRDTL containing the number of entries in the PRD table.
  - CFL set to the length of the command in the CFIS area.
  - A bit set if it is an ATAPI command.
  - W (Write) bit set if data is going to the device.
  - P (Prefetch) bit optionally set.
  - If a port multiplier is attached, the PMP field is set to the correct port multiplier port.
- If it is a queued command, software first sets PxSACT [DS (pFreeSlot)]. Software should only write new bits to set to `1`; the previous register content of PxSACT should not be rewritten in the register write.
- Software sets PxCI [CI (pFreeSlot)] to indicate to the HBA that a command is active. Software should only write new bits to set to `1`; the previous register content of PxCI should not be rewritten in the register write.

Command FIS (CFIS)

This is a software constructed FIS. For data transfer operations, this is the H2D Register FIS format as specified in below sections. The HBA sets PxTFD [STS_BSY], and then sends this structure to the attached port. If a port multiplier is attached, this field must have the port multiplier port number in the FIS itself – it should not be added by the HBA. Valid CFIS lengths are 2 to 16 Dwords and must be in Dword granularity.

FIS Types

The following sections define the structure of each individual FIS.

FIS Type Values

The value for the FIS type fields of all FISes has been selected to provide additional robustness. In minimally buffered operations that might not buffer a complete FIS, the state machines might begin acting on the received FIS type value prior to the ending CRC having been checked.

Because the FIS type value might be acted upon prior to the integrity of the complete FIS being checked against its ending CRC, the FIS type field values have been selected to maximize the Hamming distance between them.

FIS type value assignments are listed in Table 32-10.

Table 32-10: **FIS Type Value Assignments**

Type Field Value	Type Field Value Description
`27h`	Register FIS: Host to device.
`34h`	Register FIS: Device to host.
`39h`	DMA activate FIS: Device to host.
`41h`	DMA setup FIS: Bidirectional.
`46h`	Data FIS: Bidirectional.
`58h`	BIST activate FIS: Bidirectional.
`5Fh`	PIO setup FIS: Device to host.
`A1h`	Set device bits FIS: Device to host.
`A6h`	Reserved.
`B8h`	Reserved.
`BFh`	Reserved.
`C7h`	Vendor specific.
`D4h`	Vendor specific.
`D9h`	Reserved.

Chapter 33

DisplayPort Controller

Introduction

The DisplayPort controller implements a flexible display and audio pipeline architecture. The DisplayPort controller can source data from memory (non-live input) or the (live input) programmable logic (PL). The DisplayPort processes data, and sends it out through the DisplayPort source-only controller block to external display devices or to the PL (live output). The DisplayPort pipeline consists of the DisplayPort direct memory access (DMA) for fetching data from memory, a centralized buffer manager, a display rendering block, an audio mixer block, and the DisplayPort source controller, along with the PS-GTR block. The DisplayPort pipeline supports an ultra-high definition (UHD) aggregate video bandwidth of 30 Hz.

The DisplayPort DMA controller (DPDMA) supports up to six input channels as non-live input. Video/graphics, and audio streams can be sourced from the PL as live streams. The video processing stage involves mixing video and graphics streams, color space conversion, and chroma sub-sampling. The audio processing stage involves mixing two audio streams and volume control. The output of the audio/video processing pipeline can be output to the DisplayPort source controller or optionally be routed to the PL as live output.

Table 33-1 describes several DisplayPort usage scenarios. It assumes that the functions listed in the table can be enabled or disabled in software. When enabled, a function is *used*. When disabled, a function is *bypassed*. Although desirable, the ability to dynamically switch between *used* and *bypass* without causing a video artifact is not required.

Table 33-1: **DisplayPort Usage**

	Usage	Video Chroma Upsampling	Video Color Space Conversion	Graphics Color Space Conversion	Alpha Blend	Blended Video and Graphics Color Space Conversion	Blended Chroma Down-sampling	Notes
1	V: YUV444 G: RGBA8888 TX: RGB	Bypass	Use	Bypass	Use	Bypass	Bypass	
2	V: YUV422 G: RGBA8888 TX: RGB	Use	Use	Bypass	Use	Bypass	Bypass	
3	V: YUV444 G: RGBA8888 TX: YUV444	Bypass	Use	Bypass	Use	Use	Bypass	
4	V: YUV444 G: RGBA8888 TX: YUV422	Bypass	Use	Bypass	Use	Use	Use	
5	V: YUV422 G: none TX: YUV422	Bypass	Bypass	X	Use	Bypass	Bypass	Output must be equal to the input (bit-exact).
6	V: YUV422 G: none TX: YUV422	Use	Use	X	Use	Use	Use	This mode allows dynamic addition and removal of graphics.
7	V: none G: YUV TX: YUV422	X	X	Bypass	Use	Bypass	Use	Graphics can be YUV.
8	V: none G: RGB TX: YUV422	X	X	Bypass	Use	Use	Use	
9	V: YUV422 G: YUV TX: YUV422	Use	Use	Use	Use	Use	Use	Graphics can be YUV.

Notes:

1. Chroma upsampling is not required in the graphics processing pipeline.

Features

- Based on the VESA DisplayPort v1.2a source-only specification.
- Video support for the following:
    - Resolution up to 4K x 2K at 30Fps.
    - Y-only, YCbCr444, YCbCr422, YCbCr420, and RGB video formats.
    - 6, 8, 10, or 12 bits per color components.
    - Progressive video.
    - A 36-bit native video input interface to capture live video.
    - Non-live video from frame buffers using local DPDMA.
- Graphics features:
    - Non-live graphics from the frame buffer in DDR memory.
    - 36-bit native video interface along with an 8-bit alpha channel to capture live graphics.
    - 2-plane, on-the-fly rendering of video and graphics.
    - Chroma upsampling and chroma downsampling.
    - Color space conversion from YCbCr to RGB and vice versa.
    - Video blending.
    - Chroma keying.
- Audio features:
    - Two audio channel with up to 24-bit sampling size.
    - Maximum sample rate of 48 KHz.
    - Live 24-bit audio sampling from the PL.
    - Non-live 16-bit audio from the frame buffer in DDR memory.
- Audio mixer and volume control.
    - Mixing of two audio streams of the same sampling rate and channel count.
    - Provides gain control for audio streams.
- Streaming A/V output to the PL via an AXI interface.
- Includes a system time clock (STC) that is compliant with the ISO/IEC 13818-1 standard. Provides time stamping of the A/V presentation unit.
- Video timing controller used for non-live video.
- Built-in test pattern generator.

- Dedicated video PLL in the FPD with an optional alternate reference clock input.
- Glitch-free start-stop behavior.

The following features are not supported.

- Interlacing.
- Two or more partial graphics overlay (OSD) regions over video.
- Multi-stream transport.
- 5.1 or 7.1 channel audio.
- Audio mixing of different sample rates. The live audio interface does not support sample rate conversion.
- FAUX channel support.
- No back-pressure support on the PS to PL audio interface.
    - Information frame is not supported.
    - No user selectable option for audio metadata for the live audio input.

Note: See Answer Record 68671 for information on Xilinx tested monitors.

System Viewpoint

Figure 33-1 shows the video, graphics, and audio processing pipeline stages in the DisplayPort controller block.

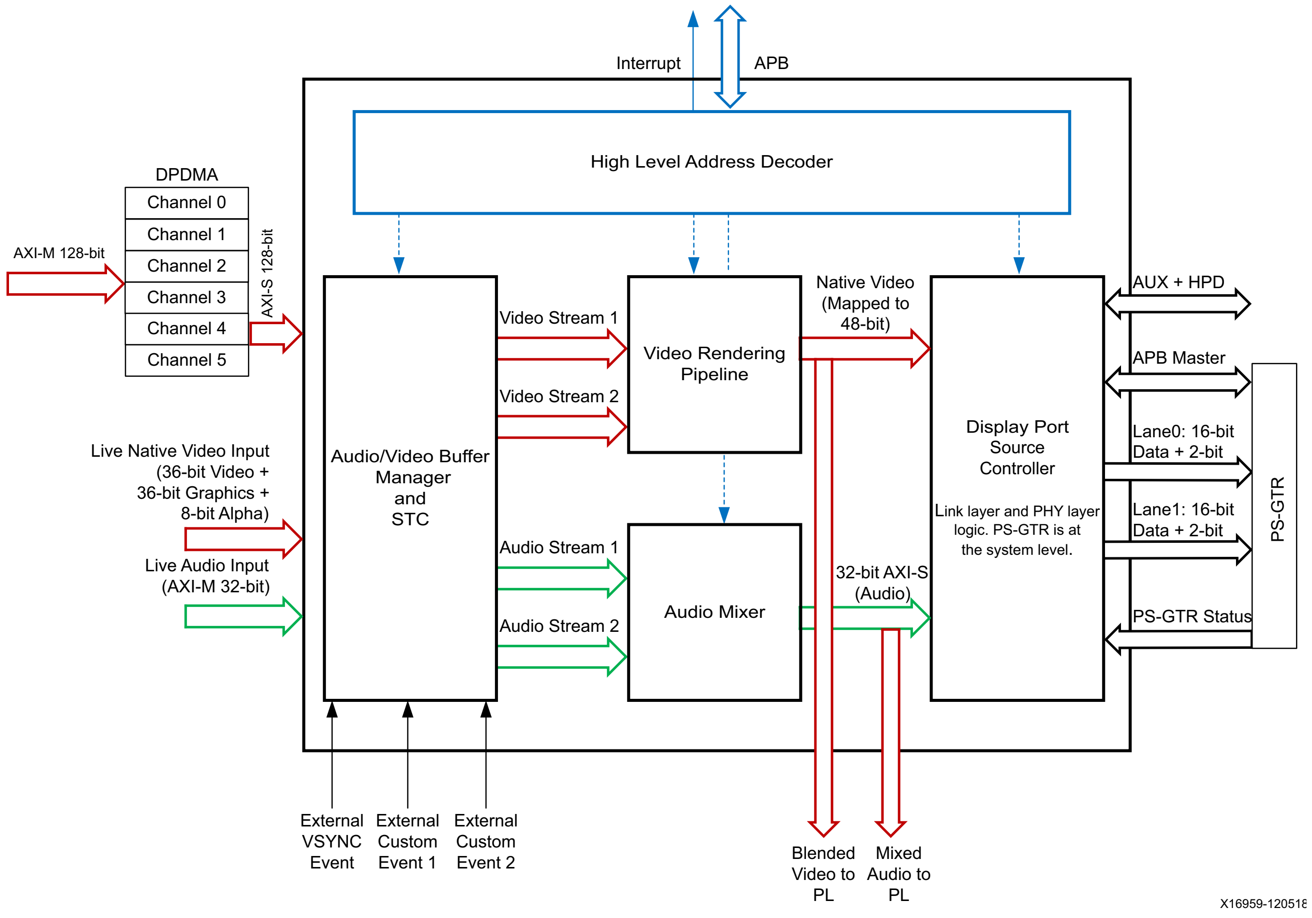


Figure 33-1: **Data Flow in the DisplayPort Controller**

Functional Description

This section describes the following functions.

- Video/Graphics
- Audio
- DisplayPort DMA
- DisplayPort Controller Clocking

Video/Graphics

Figure 33-2 shows an overview of the system.

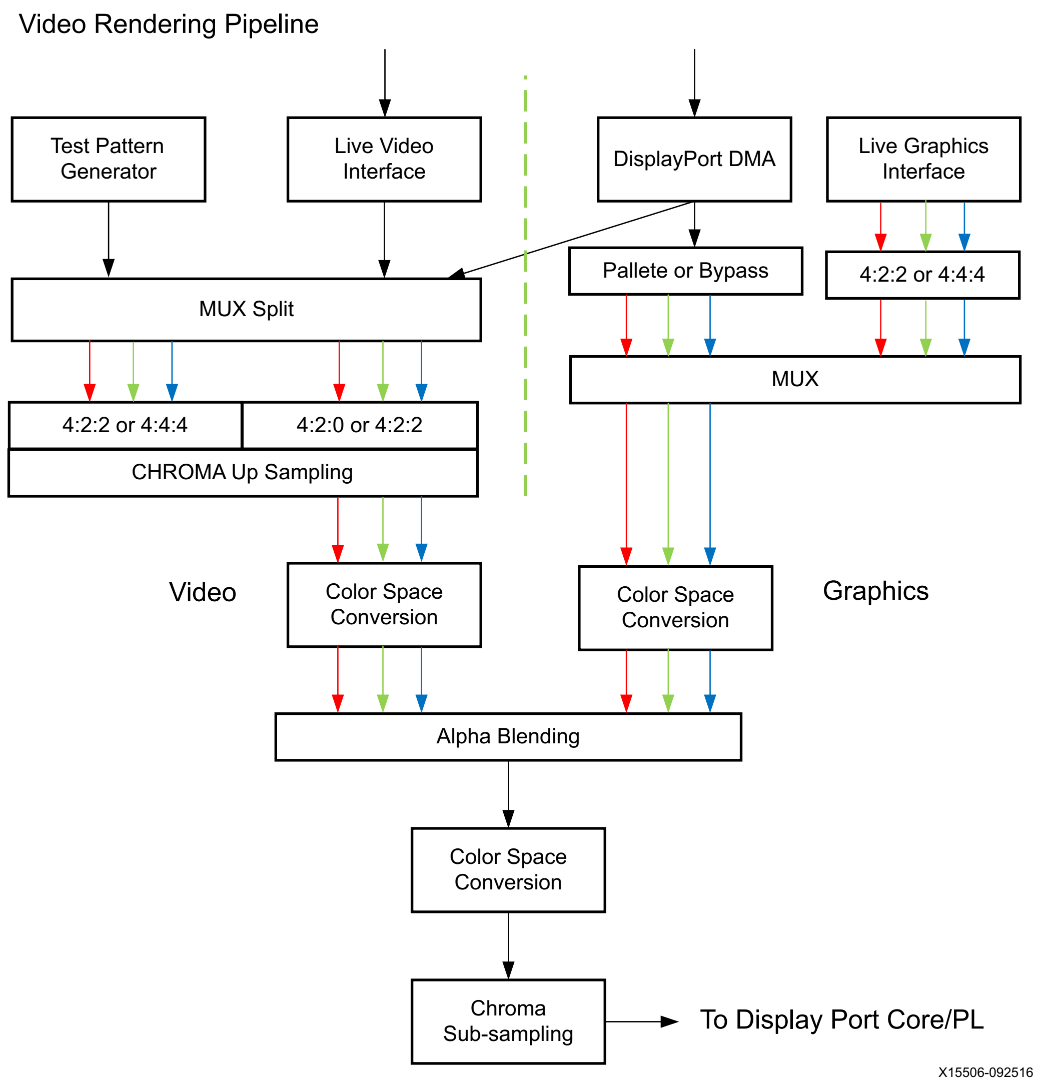


Figure 33-2: **DisplayPort Controller Video Rendering Pipeline Block Diagram**

Video Input Stage

Non-live Video/Graphics Input

When video/graphics data is sourced from memory using the DPDMA, the input stream is called a non-live input. The DPDMA has six input channels that are capable of fetching data from memory. Refer to the DisplayPort DMA section for more details about handling non-live input from memory.

Live Video/Graphics Input

In the live input example, video and graphics data can be sourced from the PL. The video and graphics frame synchronization signals are input to the live input interface. The video timing can be controlled either from the PS or from the PL. When a live video interface is used, video timing signals can be generated internally by using the VTC block in the PS or the video timing generator block in the PL. This is more clear when the live video interface are segregated into input and output. For live video input, (for example, an HDMI input), the video timing must be generated in the PL. However for live video output, video timing signals can be generated in both PS (VTC) and PL. Refer to the Live Video Output section for more details about the live input stream.

Audio/Video Buffer Manager

The A/V buffer manager manages audio/video data from memory and from the PL layer. Data from memory is considered *non-live* and data from the PL is considered *live*. Data from memory is written into channel buffers using the AXI4 stream interface. The maximum burst allowed is 256 bytes for video channels and 64 bytes on audio channels.

The DisplayPort controller is capable of presenting the following.

- Live video/audio stream
- Non-live video/audio
- A mix/blend of live and non-live video/audio

Live Presentation Mode

In live presentation mode, A/V data is received through the PL interface and the A/V timing is used to drive the DisplayPort controller. There are two live inputs: video and live graphics from the PL, which can be mixed together using alpha blending or chroma keying.

Non-Live Presentation Mode

In non-live presentation mode, A/V data is fetched from memory through the AXI master port using the local DMA controller (DPDMA). Because the data is not timed, A/V timing is locally generated in the DisplayPort controller using the internal A/V timing generator. The DPDMA is driven to fetch data so as to ensure continuous A/V data flow with no underflow due to memory latency fluctuations (a suitably sized FIFO is included). Two non-live inputs are supported: video and graphics, which can be mixed together using alpha blending or chroma keying.

To assist in A/V synchronization, the A/V presentation time must be captured as a timestamp relative to a system time clock, and associated with A/V presentation units (e.g. video frames and audio buffers), and provided to software, for example, by storing the timestamp in the DPDMA descriptor.

Mixed Presentation Mode

In mixed presentation mode, both live A/V and non-live A/V data is blended/mixed together and presented using the live A/V timing. The internal audio mixer provides audio mixing only (audio sample rate and other attributes of the two sources must be identical). The internal video/graphics alpha blender is used to blend the two video streams. In the mixed presentation mode, the following stream requirements must be fulfilled.

- The resolution of the video/graphics frames going to the mixer must be the same.
- The input streams to the audio mixer requires the audio sampling frequency to be the same between two streams.

Video Rendering Pipeline

The video rendering pipeline performs image blending, chroma upsampling and pixel scaling. It has two inputs (before blending) and one output (after blending). The two input paths are not identical. One input is used for video and the other is used for graphics. The graphics path has a color palette and does not have chroma upsampling block (converts 4:2:2 to 4:4:4), so graphics must be in the 4:4:4 format. The video path has a 4:2:0 to 4:2:2 converter, a test pattern generator, and a chroma upsampling block.

Chroma Re-sampling

Chroma sub-sampling converts the video to 4:2:2 format by horizontally sub-sampling the Cb and Cr components by a factor of 2. This is a simple DSP sample rate conversion operation, in which the new sample rate is exactly half the old sample rate. Up-sampling is the process of converting 4:2:2 format back to 4:4:4 by over-sampling the chroma components by a factor of 2.

The video rendering pipeline contains an up-scaling color depth scaling block at the input and down-scaling color depth with dithering at the output. Depending on the configuration, the pixel scaling block can scale a lower bit-per-color (BPC) value to a higher BPC value. The pixel descaling block can convert a higher BPC to a lower BPC at the output of the video blender block. The pixel scaling block converts low-resolution pixels to high-resolution pixels after multiplying the input padded pixels by a scale factor. Pixel descaling is done at the video blender output. Dithering reduces the contouring artifacts that occur at low pixel resolutions. The dithering operation consists of the following.

- Add a dithering value
- Saturate
- Truncate to the desired size

The video blender (Figure 33-3) takes two native video stream inputs and outputs blended pixels. The blending operation converts two input streams into RGB. The final result is converted to proper format per user-supplied programming. The final output video is also forwarded to the PL layer.

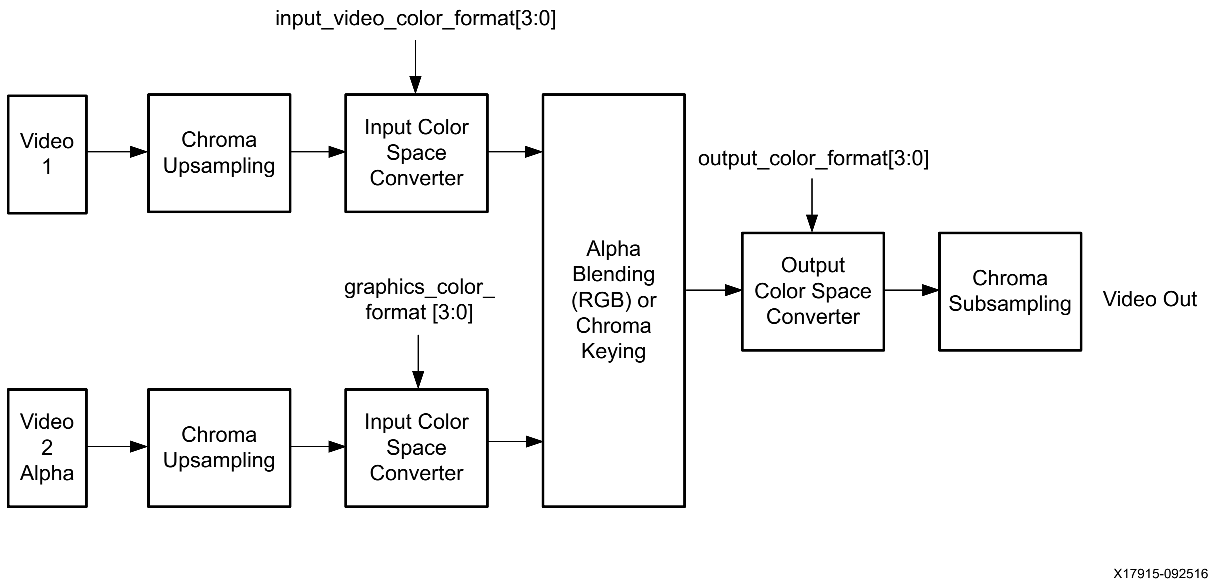

input_video_color_format[3:0]
Video 1
Chroma Upsampling
Input Color Space Converter
Alpha Blending (RGB) or Chroma Keying
output_color_format[3:0]
Output Color Space Converter
Chroma Subsampling
Video Out
graphics_color_ format [3:0]
Video 2 Alpha
Chroma Upsampling
Input Color Space Converter
X17915-092516

Figure 33-3: **Video Blender Block Diagram**

Alpha Blending

Video Blending is defined for two RGB video streams. One of these streams will be graphics that have an alpha value along with RGB stream. The alpha value available with the graphics stream will define the transparency of the graphics. Alpha value defined for blending function is always 8-bit. 1-bit alpha and 4-bit alpha are also supported, but these are scaled to 8-bits before they are used for alpha blending.

Chroma Keying

A chroma-keying operation for two video streams is supported with a programmable master select, a programmable color select with a programmable range, and an enable for chroma keying. If chroma keying is selected, video blending is bypassed.

The programmable options supported include the following.

- A programmable color as key with a range minimum to maximum.
- A select to enable chroma keying.
- Master stream select.

Video/Graphics Output Stage

DisplayPort Source Controller

The DisplayPort source controller is responsible for managing the link and physical layer functionality. The controller packs audio/video data into transfer units or micro packets and sends them over the main link. The link rate and lane counts can be selected based on the application bandwidth requirements.

The DisplayPort v1.2 protocol supports up to four lanes at a 5.4G line rate and includes audio support. The DisplayPort controller only supports up to two lanes at a 5.4G line rate. It does not support multi-stream transport or other optional features.

The source core is partitioned into three blocks.

- **Main link**: Provides for the delivery of the primary video stream.
- **Secondary link**: Integrates the delivery of audio information into the main link blanking period.
- **AUX channel**: Establishes the dedicated source-to-sink communication channel.

Figure 33-4 shows the blocks of the DisplayPort source controller.

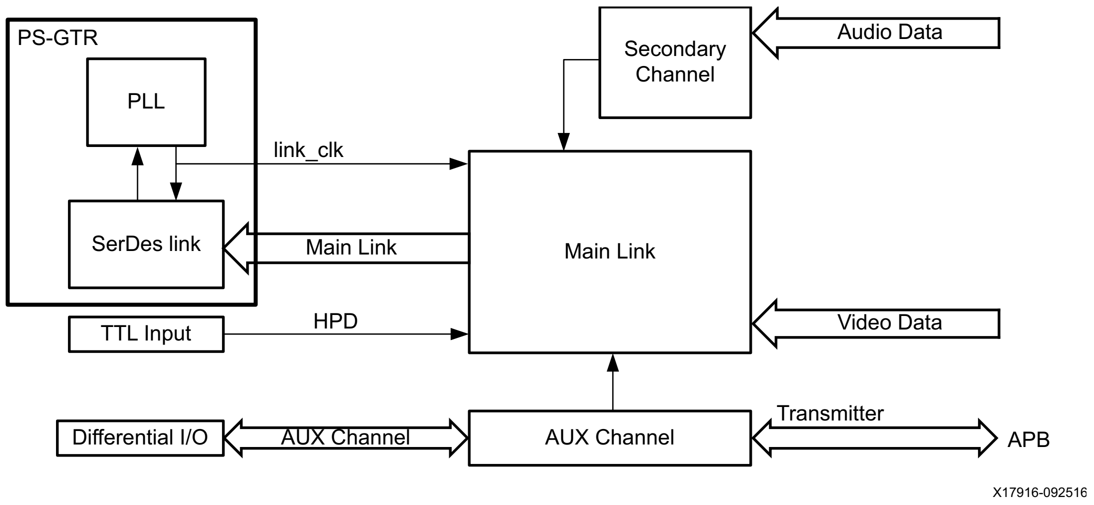


Figure 33-4: **DisplayPort Source Controller**

Live Video Output

The output of the video rendering pipeline can optionally be routed to the PL through the live video output interface. Refer to the Live Video Interface section for more information about the live video output.

Live Video Interface

The live video input interface comprises these features.

- A 36-bit native video interface is referred to as the live video input in this manual.
- A second 36-bit native video interface is referred to as the live graphics input and it has a corresponding 8-bit alpha channel.

- Both the video and graphics inputs are expected to be of same resolution and have the same video timing.
- The live graphics and live video can be of different bits per component (BPC) or video formats.
- Y-only, RGB, YUV444, YUV 422 formats are supported on live video and graphics inputs.
- A bits per component (BPC) of 6/8/10/12 is supported on live video/graphics inputs.

The live video output interface comprises these features.

- A blended 36-bit video output.
- BPC = 12 (always). To use only 8 BPC, truncate the four least significant bits (LSBs).
- Supports the RGB, YUV 444, YUV 422, and Y-Only video formats.
- When the output format is Y-only, the values on the Cb/Cr can be ignored.

Table 33-2 shows the PS-PL interface signals for live video interface.

Table 33-2: **PS-PL Signals for the Live Video Interface**

Signal Name	Type	Initial Value	Description
dp_live_video_in_vsync	Input	–	Video VSYNC.
dp_live_video_in_hsync	Input	–	Video HSYNC.
dp_live_video_in_de	Input	–	Video data enable.
dp_live_video_in_pixel1 [35:0]	Input	–	Video pixel data 1. Video data from the PL.
dp_live_gfx_pixel1_in[35:0]	Input	–	Graphics pixel input: Graphics data from the PL.
dp_live_gfx_alpha_in[7:0]	Input	–	Alpha corresponding to graphics input from the PL.
dp_video_in_clk	Input	–	Live video pixel clock for the live video and graphics I/O interfaces.
dp_video_out_pixel1 [35:0]	Output	0	Blended output, video pixel 1. Always 12 BPC.
dp_video_out_vsync	Output	0	Blended output, video VSYNC.
dp_live_video_de_out	Output	0	Blended output, video data enable.
dp_video_out_hsync	Output	0	Blended output, video HSYNC.
dp_external_custom_event1	Input	-	External user-defined event trigger to capture timestamps. Rising edge is detected as event trigger.
dp_external_custom_event2	Input	-	External user-defined event trigger to capture timestamps. Rising edge is detected as event trigger.
dp_external_vsync_event	Input	-	External VSYNC event trigger to capture timestamps. Rising edge is detected as event trigger.
dp_video_ref_clk	Output	-	27 MHz STC reference clock. This clock is provided from PS.

Notes:

1. The PS-PL interface signals for the live video/graphics interface are synchronized with the dp_video_in_clk signal.

Figure 33-5 shows the live video timing on the PS-PL interface.

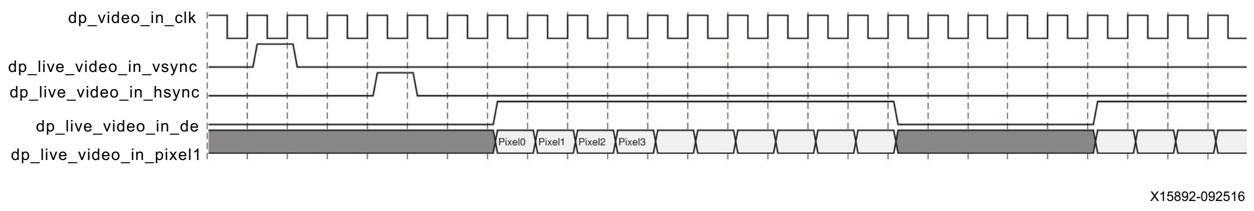


Figure 33-5: **Live Video Timing on the PS-PL Interface**

TIP: *The PS-PL interface signals are synchronized to the* `dp_video_in_clk` *signal.*

Video Timing Generation

When using the live video interface, the video timing signals can be generated internally by using the VTC block in the PS or the video timing generator block in the PL. The VTC block in the PS accepts the PL live video clock (dp_video_in_clk) as an input and can generate HSYNC and VSYNC signals. The VTC accepts clock inputs from two sources.

- Live video clock input (dp_video_in_clk) from the PL.
- Video clock generated from the video PLL.

When using a live video input from an external interface (for example, an HDMI input), the video timing must be generated in the PL. For a live video output, the VTC block in the PS can be used to generate the video timing signals.

When using the video PLL for generating the reference clock for the DisplayPort controller, the programming flow is the same as the other PS PLLs. The video PLL can accept input from any of the available reference clock inputs (PS_REF_CLK, ALT_REF_CLK, AUX_REF_CLK, or VIDEO_CLK). It requires the helper data (mentioned in Chapter 37, PS Clock Subsystem) to program the appropriate values of the PLL attributes.

High Level Address Decoder

The high-level address decoder module takes care of decoding the addresses targeted for the blocks described in the Audio/Video Buffer Manager and the DisplayPort Source Controller sections.

Video Formats

The pixel data are stored in a packed format in memory. The pixel unpacker block reads samples based on the specified format and presents it as single pixel-per-clock data to the next processing block in the DisplayPort controller block. The supported frame buffer color formats for video and graphics are described in the following tables.

Live Video Format

The live video is expected to be in the format shown in Table 33-3.

Table 33-3: **Live Video Format**

Format	BPC/BPP	R	G	B	Cr	Y	Cb	Cr/Cb	Y
RGB	6/18	[35:30]	[23:18]	[11:6]					
RGB	8/24	[35:28]	[23:16]	[11:4]					
RGB	10/30	[35:26]	[23:14]	[11:2]					
RGB	12/36	[35:24]	[23:12]	[11:0]					
YCbCr444	6/18				[35:30]	[23:18]	[11:6]		
YCbCr444	8/24				[35:28]	[23:16]	[11:4]		
YCbCr444	10/30				[35:26]	[23:14]	[11:2]		
YCbCr444	12/36				[35:24]	[23:12]	[11:0]		
YCbCr422	8/16							[35:28]	[23:16]
YCbCr422	10/20							[35:26]	[23:14]
YCbCr422	12/24							[35:24]	[23:12]
YONLY	8/8								[35:28]
YONLY	10/10								[35:26]
YONLY	12/12								[35:24]

For live video YCbCr 422, the first pixel can have Cb or Cr. There is a programmable option to select whether Cb or Cr is received as the first pixel.

Video Packer Format

Table 33-4 shows the video packer format. The dp.AV_BUF_FORMAT[NL_VID_FORMAT] register determines the input video format that can be fetched from memory. The request interval depends on the burst length that is programmed in the dp.AV_CHBUF0[BURST_LEN] and signifies the time interval between each sample of the packed video format.

Table 33-4: **Video Packer Format**

Color Format	dp.AV_BUF_FORMAT [NL_VID_FORMAT]	Format Description	BPP	Number of pixels in a beat	Request Interval
Video					
Cb-Y0-Cr-Y1	0	Interleaved 422	16	8	8 x BL
Cr-Y0-Cb-Y1	1	Interleaved 422	16	8	8 x BL

Table 33-4: **Video Packer Format *(Cont'd)***

Color Format	dp.AV_BUF_FORMAT [NL_VID_FORMAT]	Format Description	BPP	Number of pixels in a beat	Request Interval
Y0-Cr-Y1-Cb	2	Interleaved 422	16	8	8 x BL
Y0-Cb-Y1-Cr	3	Interleaved 422	16	8	8 x BL
YV16 (planar)	4	Planar 422	16	16 pixels from Buffer 0; 32 from Buffer 1 and 2.	16 x BL for channel 0 32 x BL for channel 1 and 2
YV24 (planar)	5	Planar 444	24	16 from 1 beats from each buffer.	16 x BL on all 3 buffers
YV16ci (semi-planar)	6	Semi-planar 422	16	16, from 2 buffers	16 x BL on 2 buffers
Monochrome (Y-only)	7	Monochrome Cb/Cr at the output of the unpacker = 2048 (signed zero)	8	16	16 x BL
YV16ci2 (semi-planar)	8	Semi-planar 422 with Cb/Cr swapped	16	16, from 2 buffers	16 x BL on all 3 buffers
YUV444	9	Interleaved 444	24	16, from 3 beats	BL = 1: 5, 5, 6 BL = 2: 10, 11, 11 BL = 4: 21, 21, 22 BL = 8: 42, 43, 43 BL = 16: 85, 85, 86
RGB888	10	Interleaved 444	24	16, from 3 beats	BL = 1: 5, 5, 6 BL = 2: 10, 11, 11 BL = 4: 21, 21, 22 BL = 8: 42, 43, 43 BL = 16: 85, 85, 86
RGBA8880#	11	Interleaved 4440	32	4	4 x BL
RGB888_10BPC	12	Interleaved 444	30	4 pixels per beat. Ignore 2 bits in each 32 bits.	4 x BL
YUV444_10BPC	13	Interleaved 444	30	4 pixels per beat. Ignore 2 bits in each 32 bits.	4 x BL

Table 33-4: **Video Packer Format *(Cont'd)***

Color Format	dp.AV_BUF_FORMAT [NL_VID_FORMAT]	Format Description	BPP	Number of pixels in a beat	Request Interval
YV16ci2_10BPC (planar)	14	Semi-planar 422 with Cb/Cr swapped	20	12 pixels per beat. Ignore MSB 8 bits.	12 x BL
YV16ci_10BPC (planar)	15	Semi-planar 422	20	12 pixels per beat. Ignore MSB 8 bits.	
YV16_10BPC (planar)	16	Planar 422	20	12 pixels per beat from Y buffer. 24 pixels per beat from each of Cb/Cr buffers. Ignore MSB 8 bits.	12 x BL for Y buffer 24 x BL for Cb and Cr buffers
YV24_10BPC (planar)	17	Planar 444	30	12 pixels per beat. Ignore MSB 8 bits	12 x BL
Monochrome_ 10BPC	18	Monochrome	10	12 pixels per beat Ignore MSB 8 bits	12 x BL
YV16_420 (planar)	19	Planar 420	16	16 from Y buffer 32 from Cb/Cr buffers	16 x BL for Y buffer 32 x BL for Cb and Cr buffers (based on vertical filter requirements)
YV16CI_420 (semi-planar)	20	Semi-planar 420	16	16, from 2 buffers	16 x BL for buffer 0 and 1 (based on vertical filter requirements)
YV16CI2_420	21	Semi-planar 420 with Cb/Cr swapped	16	16, from 2 buffers (with Cb/Cr swapped)	16 x BL for buffer 0 and 1 (based on vertical filter requirements)
YV16_420_ 10BPC (planar)	22	Planar 420	20	12 from Y buffer 24 from Cb/Cr buffers (together)	12 x BL for buffer 0 and 24 x BL for buffer 1 and 2 (based on vertical filter requirements)
YV16CI_420_ 10BPC (semi-planar)	23	Semi-planar 420	20	12, from 2 buffers	12 x BL for buffer 0 and 1 (based on vertical filter requirements)
YV16CI2_420_ 10BPC	24	Semi-planar 420 with Cb/Cr swapped	20	12, from 2 buffers (with Cb/Cr swapped)	16 x BL for buffer 0 and 1 (based on vertical filter requirements)

Graphics Packer Format

Table 33-5 shows the graphics packer format. The dp.AV_BUF_FORMAT[NL_GRAPHICS_FORMAT] register determines the input video format that can be fetched from memory. The request interval depends on the burst length that is programmed in the dp.AV_CHBUF0[BURST_LEN] and signifies the time interval between each sample of the packed video format.

Table 33-5: **Graphics Packer Format**

Color Format	dp.AV_BUF_FORMAT [NL_GRAPHX_FORMAT]	Description	Num Pixels in Beat	Request Interval
Graphics				
RGBA8888	0		4	4 x BL
ABGR8888	1		4	4 x BL

RGB888	2		For 3 beats, 16 pixels	BL = 1: 5, 5, 6
				BL = 2: 10, 11, 11
				BL = 4: 21, 21, 22
				BL = 8: 42, 43, 43
				BL = 16: 85, 85, 86

BGR888	3		For 3 beats, 16 pixels	BL = 1: 5, 5, 6
				BL = 2: 10, 11, 11
				BL = 4: 21, 21, 22
				BL = 8: 42, 43, 43
				BL = 16: 85, 85, 86

RGBA5551	4		8	8 x BL
RGBA4444	5		8	8 x BL
RGB565	6		8	8 x BL
8BPP	7		16 pixel addresses	16 x BL
4BPP	8		32 pixel addresses	32 x BL
2BPP	9		64 pixel addresses	64 x BL
1BPP	10		128 pixel addresses	128 x BL

YUV444	11	Unpacking same as YUV444 video format	For 3 beats, 16 pixels	BL = 1: 5, 5, 6
				BL = 2: 10, 11, 11
				BL = 4: 21, 21, 22
				BL = 8: 42, 43, 43
				BL = 16: 85, 85, 86

RGB8880 looks just like RGBA8888, i.e., eight bits per color, three colors, alpha unused, so pixels are 32-bit aligned. Data can be either RGB or YUV. For video, alpha is never used.

Supported Video Formats

The Figure 33-6 to Figure 33-13 show the supported video formats. The data on the left side is from the memory and pixel unpacker formats the data as pixel data (as is shown in the right side). The buffers are 128-bit organized. These figures show how the pixels are mapped in the lower 8-bytes (0 to 63). The mapping of the upper 8-bytes is the same as the lower bytes. The interface between the A/V buffer manager and the video blender is 48 bits.

It supports 16 bits/component. The video path works on 12 bits/component, the extra bits are dead bits.

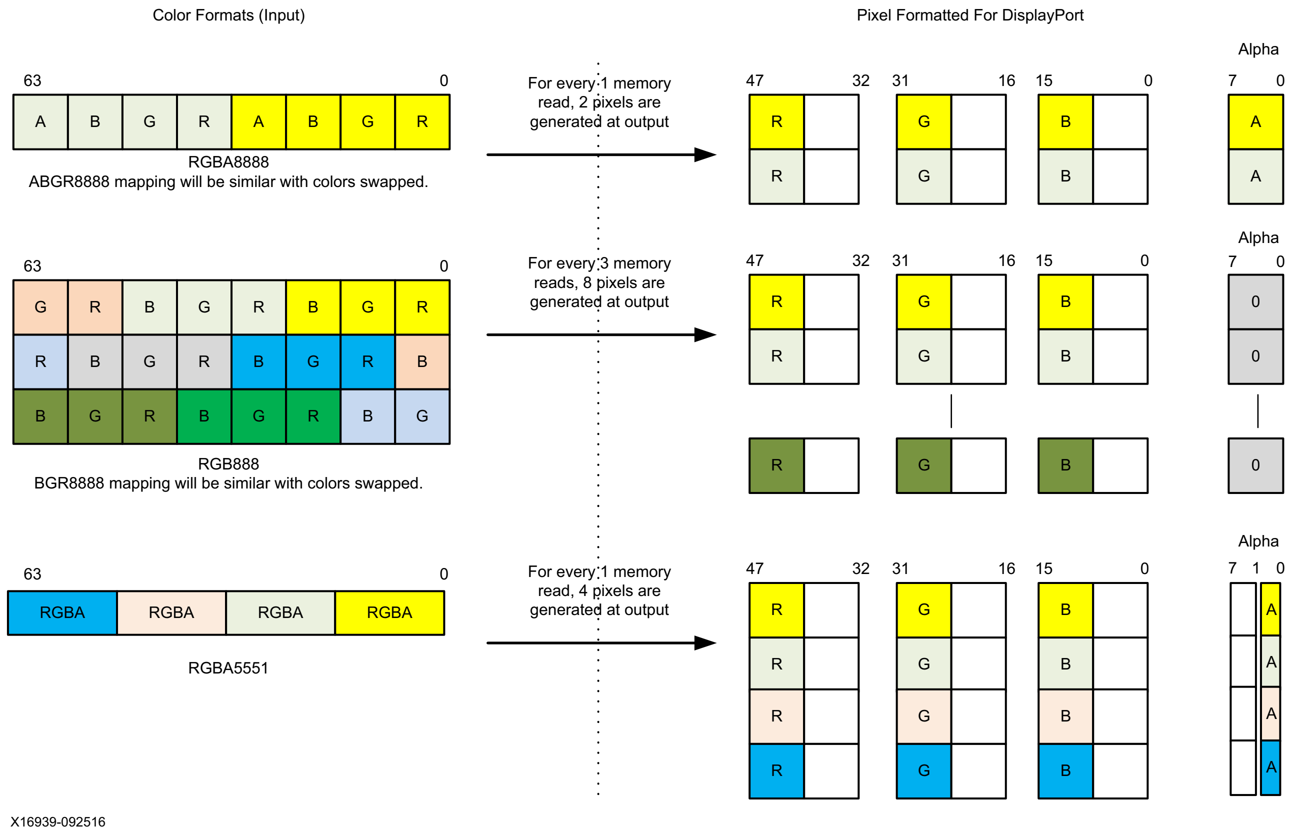

Color Formats (Input)
Pixel Formatted For DisplayPort
63
0
A B G R A B G R
RGBA8888
ABGR8888 mapping will be similar with colors swapped.
For every 1 memory read, 2 pixels are generated at output
47 32 31 16 15 0
Alpha
7 0
R G B A
G R B G R B G R
R B G R B G R B
B G R B G R B G
RGB888
BGR8888 mapping will be similar with colors swapped.
For every 3 memory reads, 8 pixels are generated at output
RGBA RGBA RGBA RGBA
RGBA5551
For every 1 memory read, 4 pixels are generated at output
7 1 0
X16939-092516

Figure 33-6: **RBG Video Format**

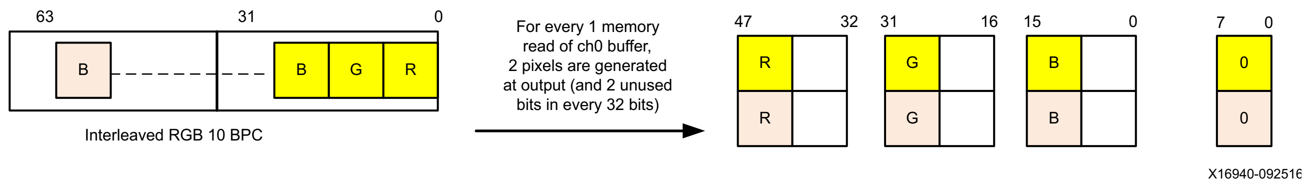
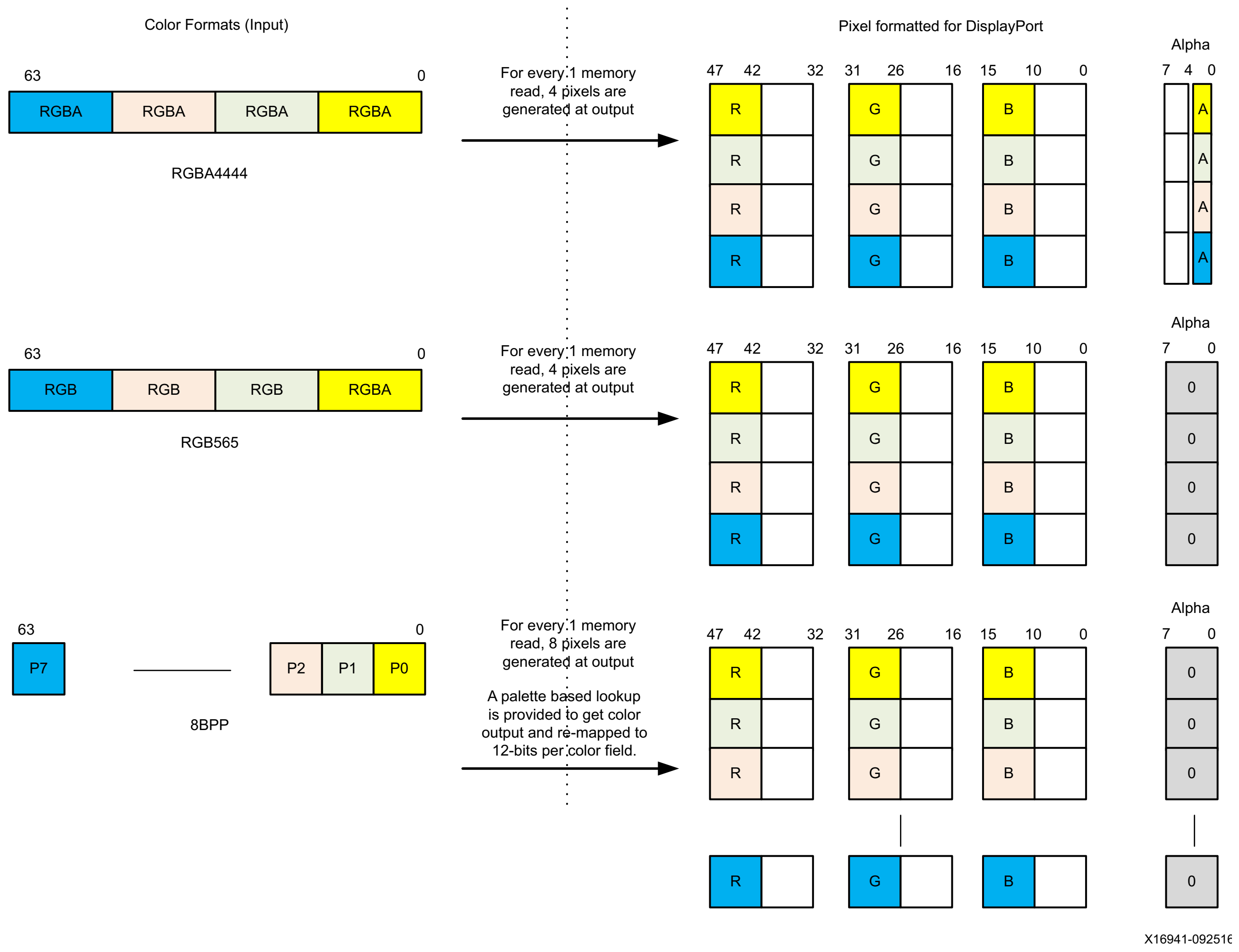

63
31
0
B
B G R
Interleaved RGB 10 BPC
For every 1 memory read of ch0 buffer, 2 pixels are generated at output (and 2 unused bits in every 32 bits)
47 32 31 16 15 0
7 0
R G B 0
X16940-092516

Figure 33-7: **Interleaved RBG Video Format**



Color Formats (Input)
Pixel formatted for DisplayPort
63
0
RGBA
RGBA
RGBA
RGBA
RGBA4444
For every 1 memory read, 4 pixels are generated at output
47 42 32 31 26 16 15 10 0
R
G
B
Alpha
7 4 0
A
RGB
RGB
RGB
RGBA
RGB565
P7
P2
P1
P0
8BPP
For every 1 memory read, 8 pixels are generated at output
A palette based lookup is provided to get color output and re-mapped to 12-bits per color field.
X16941-092516

Figure 33-8: **RGB444 Video Format**

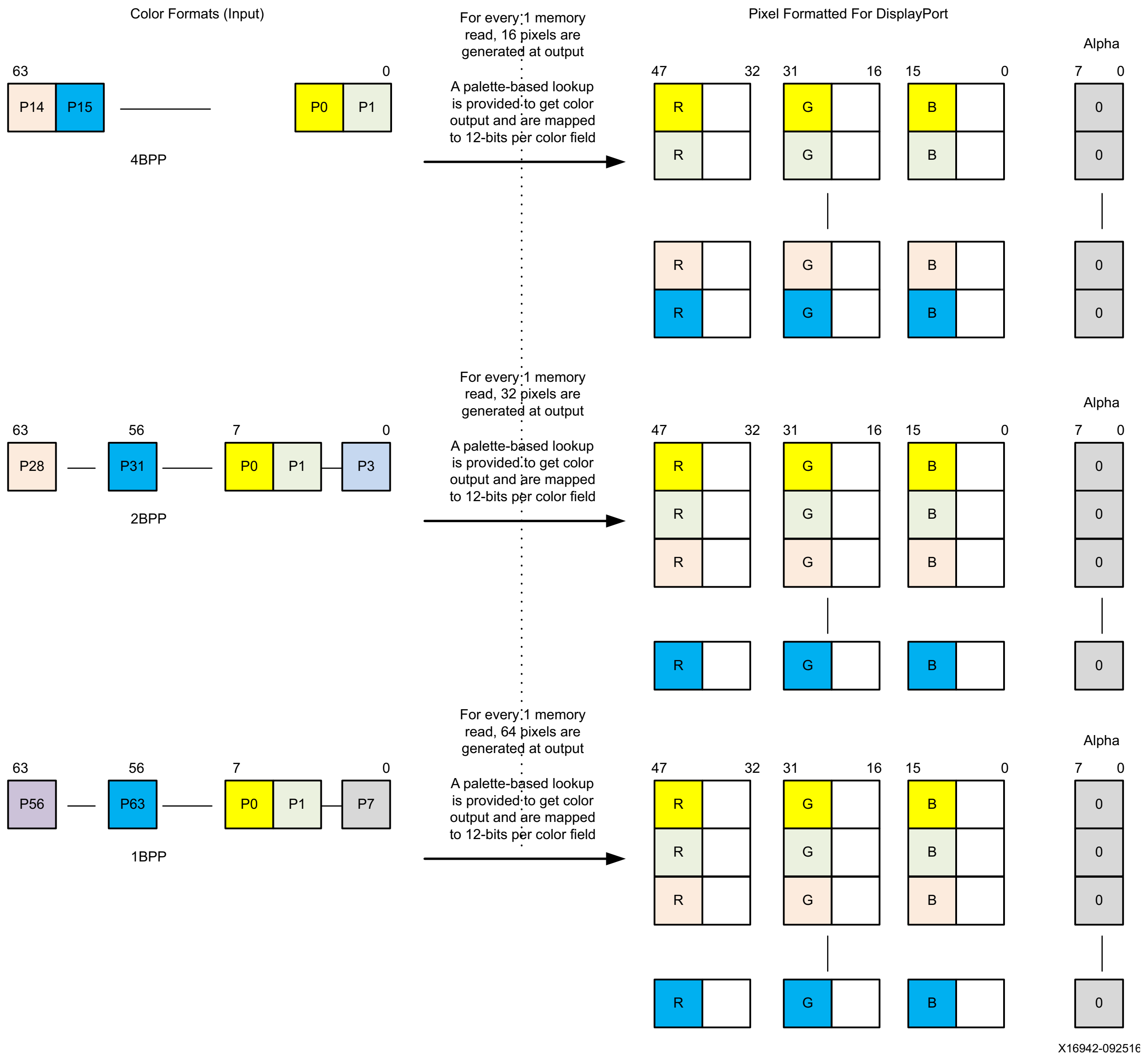
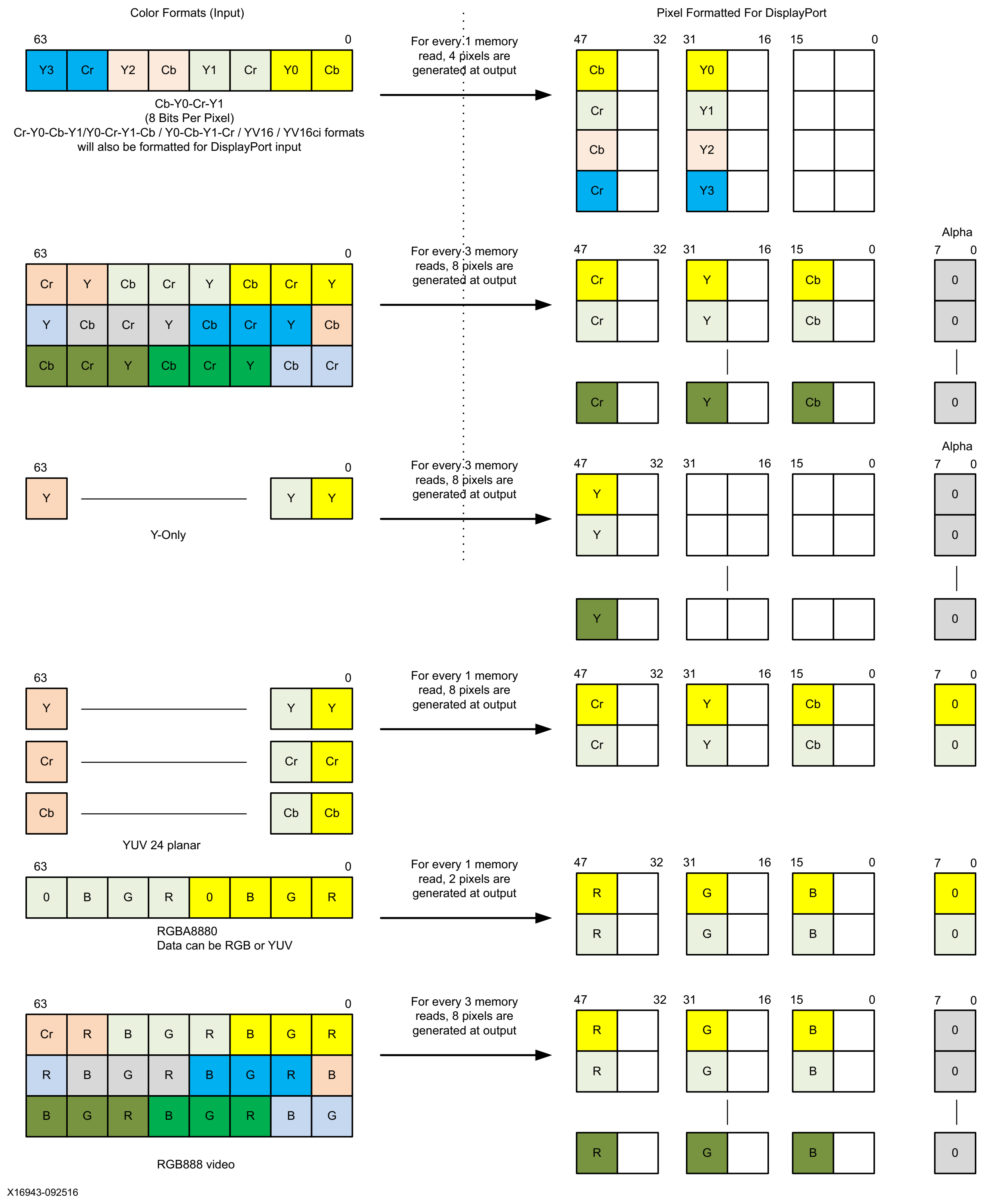

Color Formats (Input)
Pixel Formatted For DisplayPort
For every 1 memory read, 16 pixels are generated at output
A palette-based lookup is provided to get color output and are mapped to 12-bits per color field
4BPP
63
0
P14
P15
P0
P1
47
32
31
16
15
0
Alpha
7
0
R
G
B
For every 1 memory read, 32 pixels are generated at output
2BPP
56
7
P28
P31
P3
For every 1 memory read, 64 pixels are generated at output
1BPP
P56
P63
P7
X16942-092516

Figure 33-9: **BPP Video Format**



Color Formats (Input)
Pixel Formatted For DisplayPort
63
0
Y3 Cr Y2 Cb Y1 Cr Y0 Cb
Cb-Y0-Cr-Y1
(8 Bits Per Pixel)
Cr-Y0-Cb-Y1/Y0-Cr-Y1-Cb / Y0-Cb-Y1-Cr / YV16 / YV16ci formats will also be formatted for DisplayPort input
For every 1 memory read, 4 pixels are generated at output
47 32 31 16 15 0
Cb Cr Cb Cr
Y0 Y1 Y2 Y3
For every 3 memory reads, 8 pixels are generated at output
Alpha
7 0
Cr Y Cb Cr Y Cb Cr Y
Y Cb Cr Y Cb Cr Y Cb
Cb Cr Y Cb Cr Y Cb Cr
Y-Only
For every 1 memory read, 8 pixels are generated at output
YUV 24 planar
0 B G R 0 B G R
RGBA8880
Data can be RGB or YUV
For every 1 memory read, 2 pixels are generated at output
R G B
Cr R B G R B G R
R B G R B G R B
B G R B G R B G
RGB888 video
X16943-092516

Figure 33-10: **YUV Video Format**

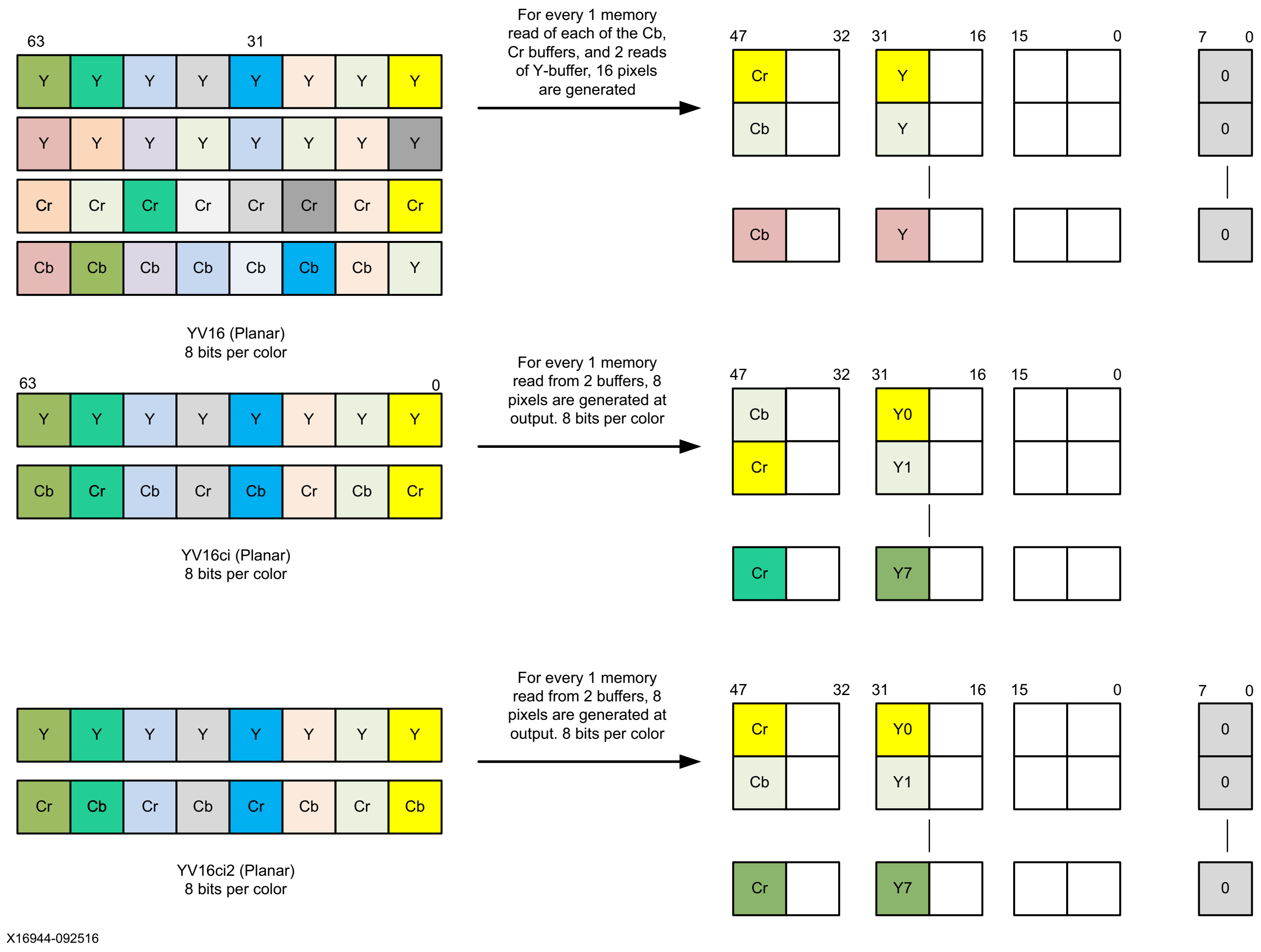

63
31
Y
Cr
Cb
For every 1 memory read of each of the Cb, Cr buffers, and 2 reads of Y-buffer, 16 pixels are generated
47
32
16
15
0
7
YV16 (Planar)
8 bits per color
For every 1 memory read from 2 buffers, 8 pixels are generated at output. 8 bits per color
Y0
Y1
Y7
YV16ci (Planar)
8 bits per color
YV16ci2 (Planar)
8 bits per color
X16944-092516

Figure 33-11: **YV Video Format**

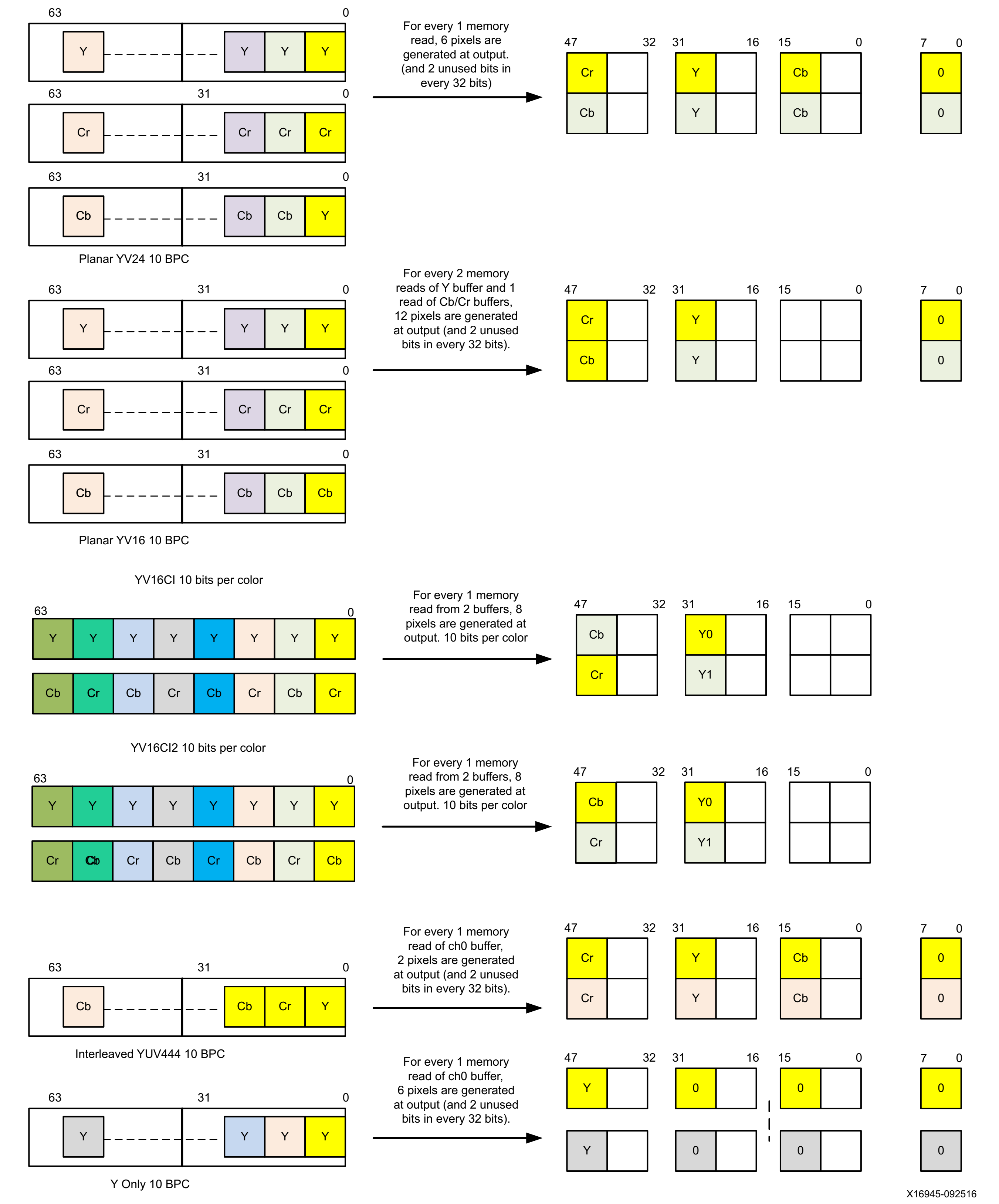
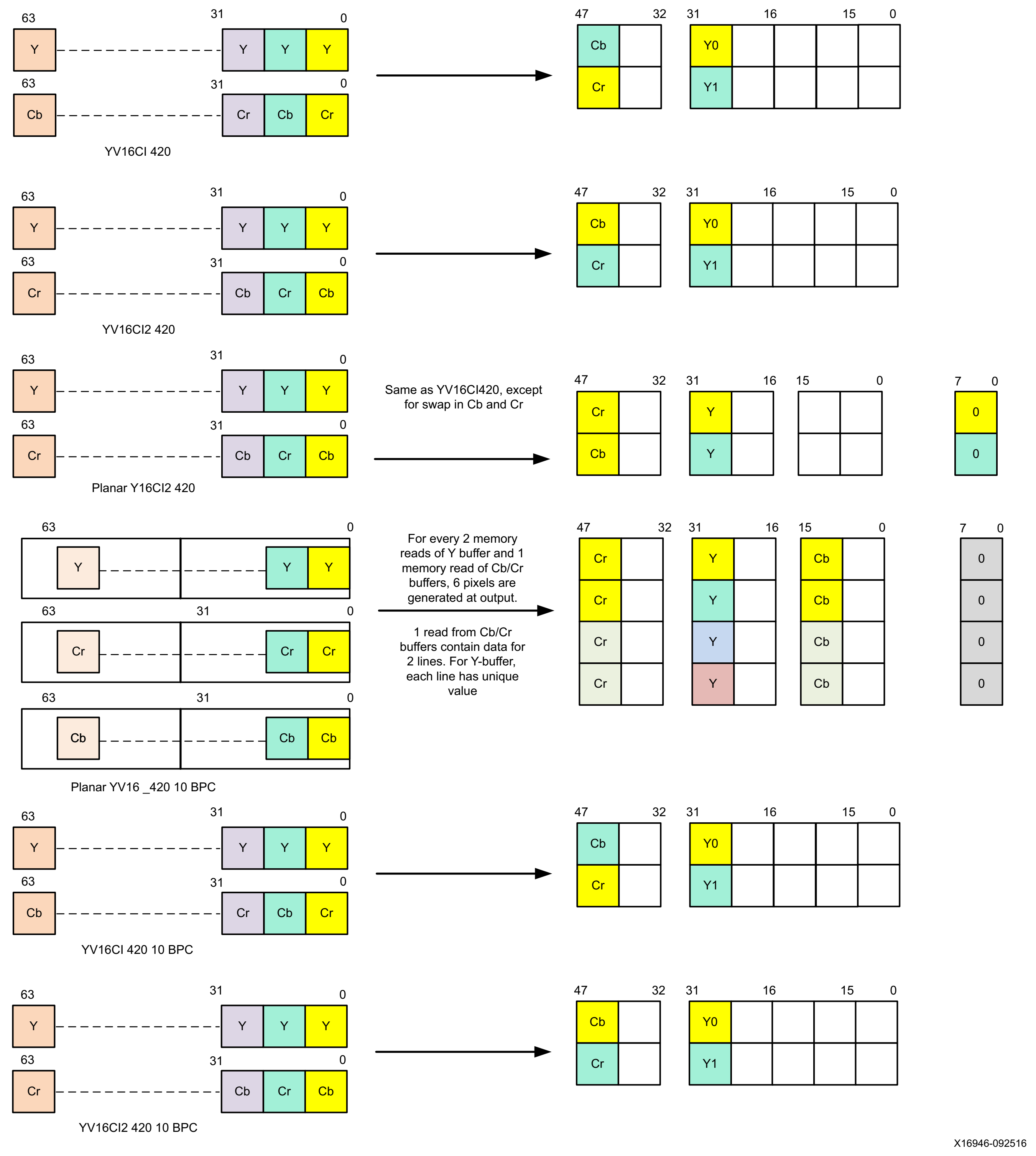

63
0
Y
Y
Y
Y
For every 1 memory read, 6 pixels are generated at output. (and 2 unused bits in every 32 bits)
63
31
0
Cr
Cr
Cr
Cr
63
31
0
Cb
Cb
Cb
Y
Planar YV24 10 BPC
47
32
31
16
15
0
7
0
Cr
Cb
Y
Y
Cb
Cb
0
0
63
31
0
Y
Y
Y
Y
For every 2 memory reads of Y buffer and 1 read of Cb/Cr buffers, 12 pixels are generated at output (and 2 unused bits in every 32 bits).
63
31
0
Cr
Cr
Cr
Cr
63
31
0
Cb
Cb
Cb
Cb
Planar YV16 10 BPC
Cr
Cb
Y
Y
0
0
YV16CI 10 bits per color
63
0
Y Y Y Y Y Y Y Y
Cb Cr Cb Cr Cb Cr Cb Cr
For every 1 memory read from 2 buffers, 8 pixels are generated at output. 10 bits per color
Cb
Cr
Y0
Y1
YV16CI2 10 bits per color
63
0
Y Y Y Y Y Y Y Y
Cr Cb Cr Cb Cr Cb Cr Cb
For every 1 memory read from 2 buffers, 8 pixels are generated at output. 10 bits per color
Cb
Cr
Y0
Y1
63
31
0
Cb
Cb
Cr
Y
Interleaved YUV444 10 BPC
For every 1 memory read of ch0 buffer, 2 pixels are generated at output (and 2 unused bits in every 32 bits).
Cr
Cr
Y
Y
Cb
Cb
0
0
63
31
0
Y
Y
Y
Y
Y Only 10 BPC
For every 1 memory read of ch0 buffer, 6 pixels are generated at output (and 2 unused bits in every 32 bits).
Y
Y
0
0
0
0
0
0
X16945-092516

Figure 33-12: **YV24 Video Format**



63
31
0
Y
Y
Y
Y
Cb
Cr
Cb
Cr
YV16CI 420
47
32
16
15
Cb
Cr
Y0
Y1
YV16CI2 420
Cr
Planar Y16CI2 420
Same as YV16CI420, except for swap in Cb and Cr
7
For every 2 memory reads of Y buffer and 1 memory read of Cb/Cr buffers, 6 pixels are generated at output.
1 read from Cb/Cr buffers contain data for 2 lines. For Y-buffer, each line has unique value
Planar YV16 _420 10 BPC
YV16CI 420 10 BPC
YV16CI2 420 10 BPC
X16946-092516

Figure 33-13: **Planar Video Format**

Audio

The DisplayPort controller supports a non-live audio channel from memory and a live audio channel from the PL. The audio mixer block is capable of mixing the two audio channels based on predefined gain settings. The output of the mixer can either be sourced to the DisplayPort source-only controller or to the PL.

Audio Input Stage

The audio can be sourced from memory or from the PL using a dedicated audio channel to the DisplayPort controller. The following sections describe each of the interfaces for sourcing audio.

Audio Non-live Input

Non-live audio input can be sourced from memory using the DPDMA. This interface supports two non-live audio channels capable of fetching audio samples from memory. Refer to the DisplayPort DMA section for further details.

Audio Live Input

Live audio can be sourced from PL. The A/V buffer manager handles multiplexing between live and non-live audio input and provides two audio streams to audio mixer block. For more details of live audio interface, please refer to PS-PL audio interface section.

Audio Processing Stage

The audio processing stage involves mixing two audio streams based on a predefined gain setting. The processing pipeline contains volume control circuitry to control the volume of the audio output.

Audio Mixer

Audio mixing uses two audio streams with the same audio sample rates. The audio mixer block does not perform any upsampling or downsampling. Figure 33-14 shows the block diagram of the audio mixer block.

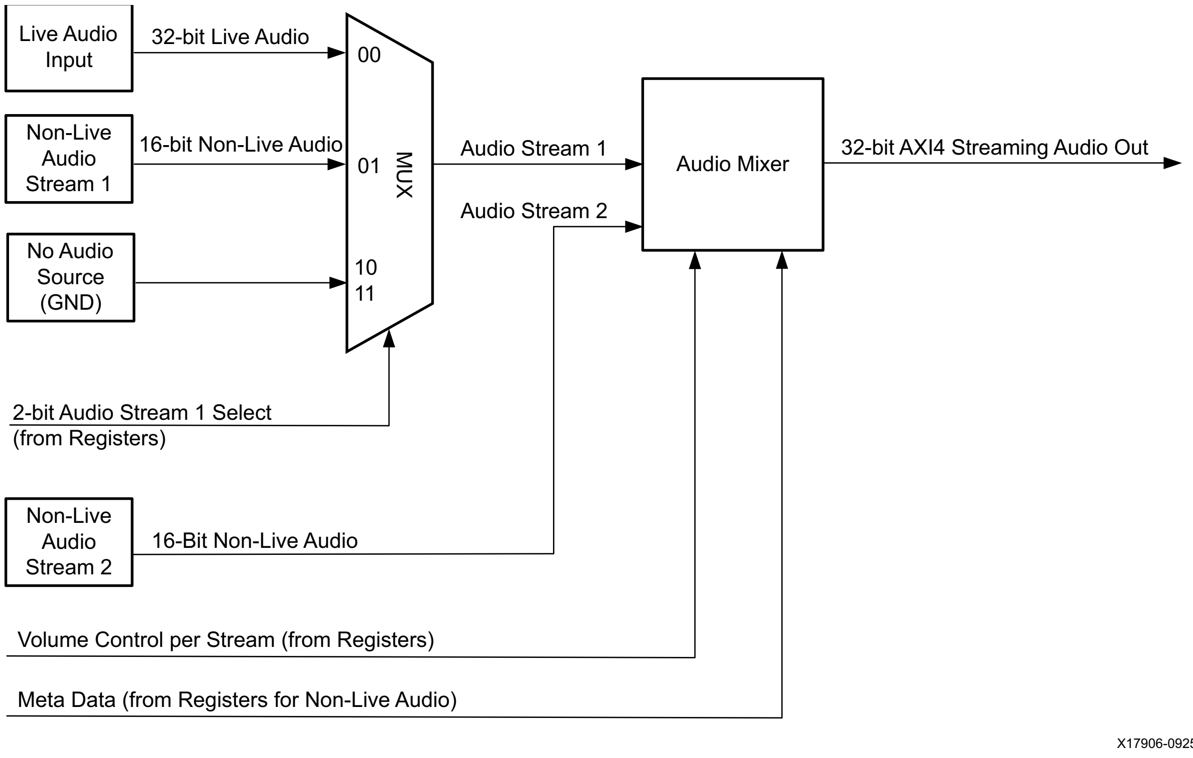


Figure 33-14: **Audio Mixer Block Diagram**

The mixing is implemented using additive logic on 24-bit audio samples as shown in Figure 33-14. Each audio stream is multiplied with a corresponding 16-bit volume control and then added.

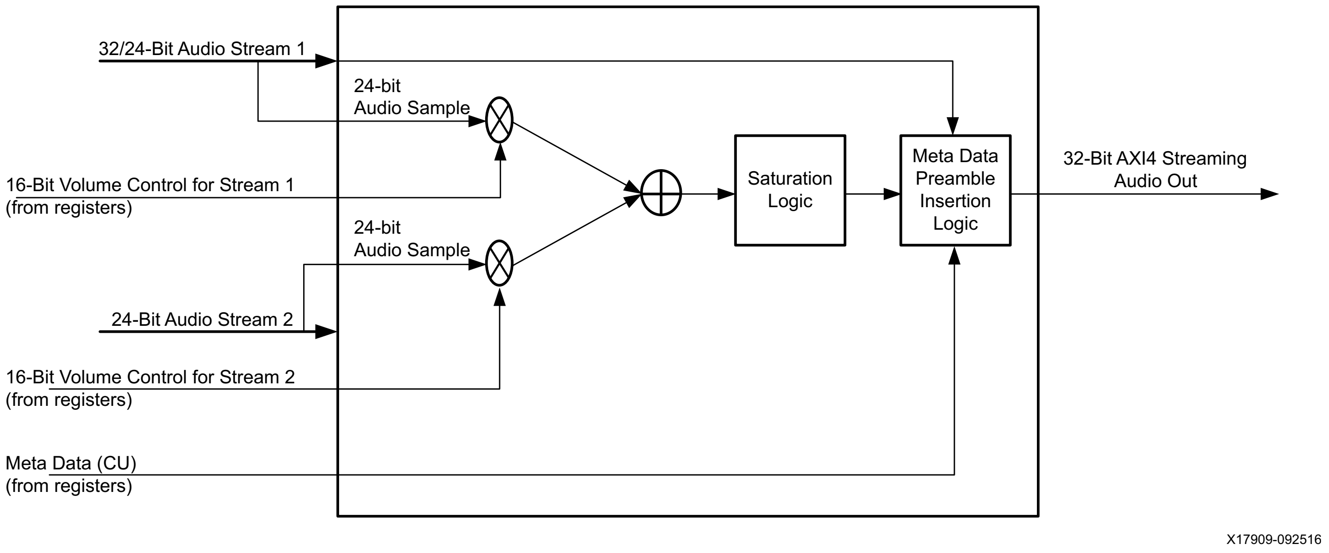

32/24-Bit Audio Stream 1
24-bit Audio Sample
16-Bit Volume Control for Stream 1 (from registers)
24-bit Audio Sample
24-Bit Audio Stream 2
16-Bit Volume Control for Stream 2 (from registers)
Meta Data (CU) (from registers)
Saturation Logic
Meta Data Preamble Insertion Logic
32-Bit AXI4 Streaming Audio Out
X17909-092516

Figure 33-15: **Volume Control Block Diagram**

The mixed audio in AXI-S format is forwarded to the DisplayPort source controller and the PL. A small holding buffer that supports AXI-S is used to handshake with the PL. The audio channel does not accept any back pressure from the PL interface.

Audio Output Stage

Audio Output Stage from the DisplayPort Source Controller

The output of the pipeline stage is provided to the DisplayPort source controller. The DisplayPort source controller inserts audio packets in the audio slots and transmits the audio to the receiving device.

Audio Live Output

Optionally, the output of the audio mixer can be output to the PL. Refer to the PS-PL Audio Interface section for details about the live audio output interface.

PS-PL Audio Interface

On the output interface, the data is at audio frequency, each sample separated by $1/f_s$ where f_s = sampling frequency. Back pressure from PL to PS is not supported.

Table 33-6 shows the PS-PL interface signals for live audio interface. The audio_in group of signals is called the S_AXIS_AUDIO interface. The audio_out group of signals is called the M_AXIS_MIXED_AUDIO interface.

Table 33-6: **PS-PL Signals for the Live Audio Interface**

Signal Name	Type	Initial Value	Description
dp_s_axis_live_audio_tdata[31:0]	Input	–	Streaming data input. [3:0] Preamble code (PR). `4'b0001` → Subframe 1, start of audio block. `4'b0010` → Subframe 1. `4'b0011` → Subframe 2.
			[27:4] Audio sample word.
			[28] Validity bit (V).
			[29] User bit (U).
			[30] Channel status (C).
			[31] Parity (P).
dp_s_axis_live_audio_tid	Input	–	Audio channel ID.
dp_s_axis_live_audio_tvalid	Input	–	Valid indicator for audio data from master.
dp_s_axis_live_audio_tready	Output	0	Ready indicator from DisplayPort controller.
dp_s_axis_audio_aclk	Input	–	Clock for AXI slave audio data input.

Table 33-6: **PS-PL Signals for the Live Audio Interface *(Cont'd)***

Signal Name	Type	Initial Value	Description
dp_m_axis_mixed_audio_tdata[31:0]	Output	0	Streaming data input. [3:0] Preamble code (PR). `4'b0001` → Subframe 1, start of audio block. `4'b0010` → Subframe 1. `4'b0011` → Subframe 2.
			[27:4] Audio sample word.
			[28] Validity bit (V).
			[29] User bit (U).
			[30] Channel status (C).
			[31] Parity (P).
dp_m_axis_mixed_audio_tid	Output	0	Audio channel ID.
dp_m_axis_mixed_audio_tvalid	Output	0	Valid indicator for audio data from master.
dp_m_axis_mixed_audio_tready	Input	–	Ready indicator from PL

Notes:

1. The live audio interface needs to be synchronized with the dp_s_axis_audio_aclk signal.

Non-Live Audio Format

A 16-bit audio format is supported by the non-live channel and samples are packed in a 128-bit AXI bus. The mapping for the lower 64-bits, upper 64-bits are shown below. There is an option to swap left and right channels.

16-bit audio sample 4 (right channel sample)	16-bit audio sample 3 (left channel sample)	16-bit audio sample 2 (right channel sample)	16-bit audio sample 1 (left channel sample)

TIP: *To swap left and right channels using the SW bit for both graphics and non-live audio. See the register description: dp.AV_BUFFER_AUDIO_CH_CONFIG.*

The maximum burst size supported on an audio channel is 4.

Live Audio Format

Audio data is written into memory in an AES3 standard format. The mapping for a 32-bit sample is shown below.

31	30	29	28	27:4	3:0
Parity (P)	Channel Status (C)	User Bit (U)	Validity (V)	Audio Sample Word	Preamble Code: • `4'b0001`: Subframe1 start of audio block. • `4'b0010`: Subframe1 • `4'b0011`: Subframe2

The metadata (such as P, C, U, V and preamble code) is embedded along with the live data input. For non-live data, registers are provided to input the metadata. The audio stream selector picks two streams per your selection and forwards them to the output.

Audio Metadata

When live audio is selected on stream1, irrespective of stream2, the channel status (C), user bit (U), and validity (V) metadata are considered from thelive data input. Because 16-bit non-live audio does not carry the channel status (C) metadata, user bit (U) bits are used from the registers. The parity bit (P) is internally calculated and inserted in the audio sample as audio mixing can change the incoming data parity. The use-cases in Table 33-7 are supported for metadata insertion.

Table 33-7: **Audio Metadata Use Cases**

Stream1	Stream2	Metadata
Live	OFF	Live
Live	Non-Live	Live
Non-Live	OFF	Register
Non-Live	Non-Live	Register
OFF	Non-Live	Register

DisplayPort DMA

The DisplayPort controller source system supports multiple video and audio channels which are used to get video/audio data from system DDR memory. These are known as non-live video/audio. To facilitate the data transfer from DDR to the DisplayPort controller, a DisplayPort DMA (DPDMA) block is included in the DisplayPort subsystem to handle six channels; three video channels, one graphics channel, and two audio channels. The DPDMA fetches the frame buffer data from the DDR and hands it over to the audio video buffer (AV buffer) inside the DisplayPort controller. The DPDMA uses an AXI stream interface with the DisplayPort controller, while it is connected to the DDR through the AXI interconnect in the PS. An AXI3 128-bit master interface is used by the DPDMA to connect with the PS interconnect.

Each DPDMA channel has a configuration register for the QoS value. By default, the input of the DisplayPort controller from the PL should have the QoS to set the video traffic class. Otherwise, small latency may choke the DMA and end up cutting the display. Refer to *Zynq UltraScale+ Devices Register Reference* (UG1087) [Ref 4].

The DPDMA supports the following features:

- Support for simultaneous read and write transactions.
- Six independent channels.
- Multiple outstanding transactions per channel.
- Fixed interval transaction scheduling.
- Simple memory buffer and 2-D buffer formats with line stride (for video).
- Memory-based descriptor task linked list with wrap option (a circular list of buffers).
- Support for autonomous operation with a circular task list.
- Each descriptor (per channel) provides programmable values to be programmed by system software.
- Support for line/buffer size that is not an integer multiple of the AXI burst size/length.
- Support for the option to set/clear buffer done flag in the descriptor.
- Support for the option to store a timestamp in the descriptor.
- Support for optional interrupt generation at the end of each task.

DPDMA does not support the following features:

- Varying burst length for each channel.
- A CRC option on the DPDMA.
- Redundant pixel formats for video and graphics.

Figure 33-16 shows the DPDMA block in the Zynq UltraScale+ MPSoC. The following section describes the descriptor structure of DPDMA.

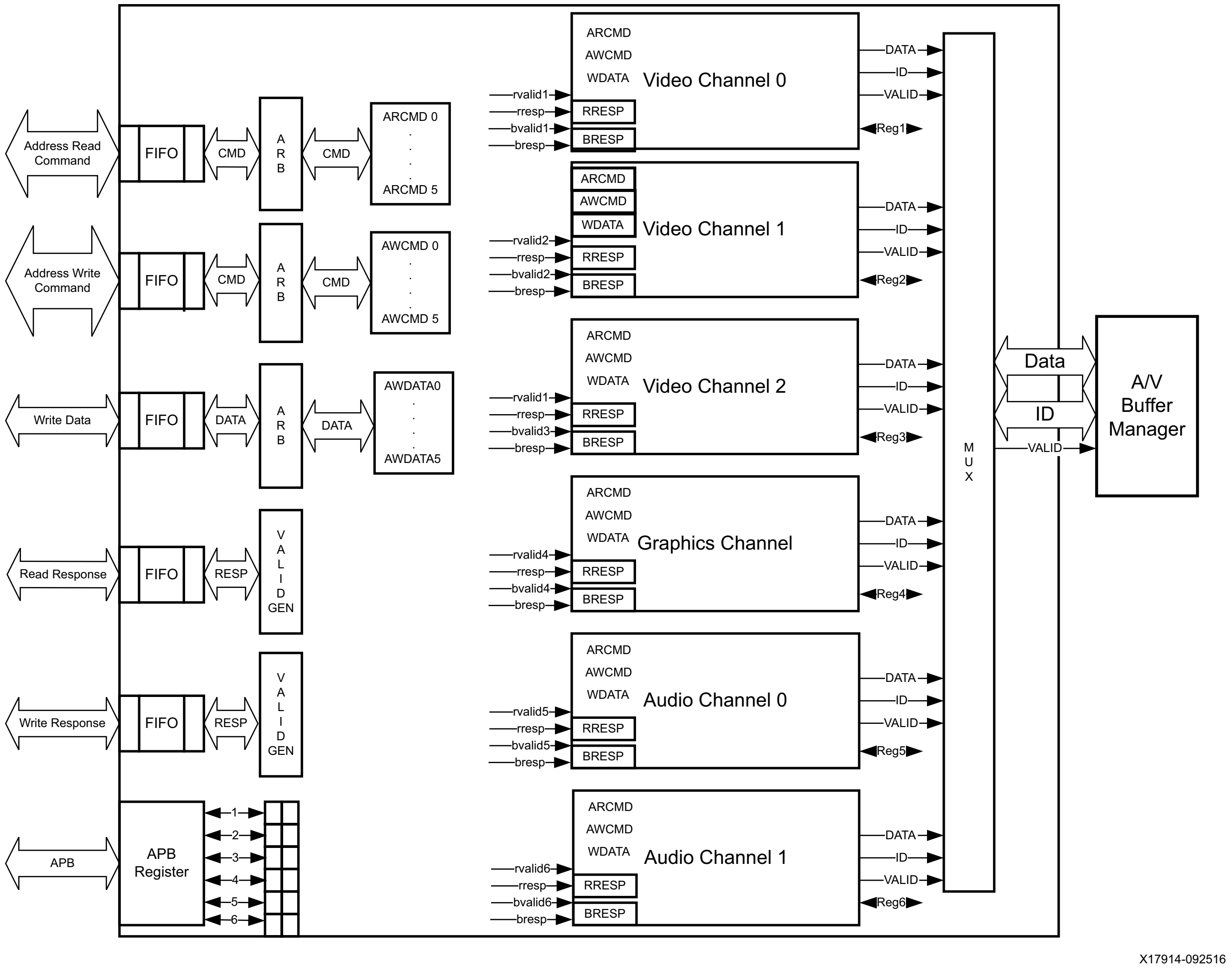


Figure 33-16: **DPDMA Architectural Blocks**

The DPDMA block acts as an AXI master in the full-power domain (FPD) and has a 128-bit AXI master port. This block is primarily used for fetching descriptor data and descriptor updates. The DPDMA also implements the advanced peripheral bus (APB) port for register access. Upon a data fetch request from software, the DPDMA initiates read transfers from memory. This data is provided to the A/V buffer manager through the 128-bit data port.

The DPDMA block receives information regarding VSYNC, HSYNC, and active video time from the DisplayPort subsystem. On video channels, the DPDMA can read 256 bytes of data every burst (128 x 16) when the burst size is programmed as 16. The DisplayPort subsystem takes 32 pixel clocks to consume this data (4B/pixel video format). The DisplayPort requests data every 32 pixel clocks. The DPDMA block fetches 256 bytes of data on every AXI transaction and receives information regarding stride and line size from the descriptor field.

Based on line start, line end, frame start, and frame end signals, the DPDMA reads the frame buffer from memory.

The DPDMA supports two descriptor payload formats, contiguous payload and fragmented payload. The contiguous payload format is efficient for bare-metal applications where large chunks of contiguous memory are available. On Linux systems, large chunks of contiguous memory allocation are difficult to obtain. To support display applications on Linux systems, the DPDMA implements a fragmented payload mode on the descriptor. It supports payload sizes as small as 4KB. The descriptor format is explained in more details in a later section.

The DPDMA uses a descriptor-based architecture. This allows software to divide frame buffers into data sets that are small as 4 KB. Software can maintain a circular chain of descriptors per channel. The DPDMA goes through the chain and provides data to the DisplayPort subsystem.

The DPDMA puts the following restrictions on size and alignment of the descriptor.

- Descriptor and data payload must start at a 256-byte aligned address.
    - With this requirement, the DPDMA does not have to deal with a 4K crossing of an AXI burst.
    - The DPDMA generates a fixed number of transactions every fetch request.
- The data payload must end at a line boundary or frame boundary.
    - The payload cannot end within the line.
    - This restriction is necessary for QoS to work efficiently. By doing this, the DPDMA ensures that it does not fetch a descriptor during active video time.

The DPDMA supports two descriptor formats to alleviate these restrictions.

Descriptor Fields

This section outlines the descriptor fields (Table 33-8).

TIP: *If a descriptor update is required, the DPDMA only updates word 4 and 5. The other words are not updated by the DPDMA.*

PREAMBLE Field

DPDMA checks the validity of the descriptor by comparing the preamble with a predefined preamble value (`0xA5`). If there is an error in the preamble, the DPDMA goes to an invalid location to read the descriptor. This should result in a preamble mismatch. The DPDMA generates an interrupt to indicate this error.

Table 33-8: **DPDMA Descriptor Fields**

Word Number	Field Name	Size (Bytes)	Bits	Description
0	control	4	[7:0]	Descriptor (8).
			[8]	Enable completion interrupt (1).
			[9]	Enable descriptor update (1).
			[10]	Ignore done (1).
			[11]	AXI burst type INCR or FIXED (1).
			[15:12]	AXCACHE (4).
			[17:16]	AXPROT bits (2).
			[18]	Mode = descriptor mode. 0 = contiguous 1 = fragmented
			[19]	Last descriptor (1).
			[20]	Enable CRC (1).
			[21]	Last descriptor of frame (1).
			[31:22]	Reserved (3)
1	DSCR_ID	4	[15:0]	Descriptor ID (16)
			[31:16]	Reserved(16)
2	XFER_SIZE	4	[31:0]	Indicates transfer size in both modes (in bytes) (32).
3	LINE_SIZE_STRIDE	4	[17:0]	Horizontal resolution (line size) (18).
			[31:18]	Stride (14).
4	Timestamp LSB	4	[31:0]	If enabled, the DPDMA stores the LSB of the timestamp here (32).
5	Timestamp MSB	4	[9:0]	If enabled, the DPDMA stores the MSB of the timestamp here (10).
			[30:10]	Reserved (21).
			[31]	Status/done (1).
6	ADDR_EXT	4	[15:0]	Next descriptor extension (16).
			[31:16]	SRC address extension (16).
7	NEXT_DESR	4	[31:0]	Address of the next descriptor (32).
8	SRC_ADDR	4	[31:0]	Source address (32).
9	ADDR_EXT_23	4	[15:0]	Address extension for SRC Addr2 (16).
			[31:16]	Address extension for SRC Addr3 (16).
10	ADDR_EXT_45	4	[15:0]	Address extension for SRC Addr4 (16).
			[31:16]	Address extension for SRC Addr5 (16).
11	SRC_ADDR2	4	[31:0]	Source address of 2nd page (32).
12	SRC_ADDR3	4	[31:0]	Source address of 3rd page (32).
13	SRC_ADDR4	4	[31:0]	Source address of 4th page (32).

Table 33-8: **DPDMA Descriptor Fields *(Cont'd)***

Word Number	Field Name	Size (Bytes)	Bits	Description
14	SRC_ADDR5	4	[31:0]	Source address of 5th page (32).
15	CRC	4	[31:0]	Reserved (32).

EN_DSCR_DONE_INTR Field

If this bit is set, the DPDMA generates an interrupt to indicate that the processing of the current descriptor is complete. If the descriptor update is enabled, the DPDMA updates (writes back) the descriptor and waits for the BRESP (write response) to generate an interrupt. This ensures the coherency of the IRQ generation.

In case the DSCR update is not requested, it generates an interrupt after it receives all outstanding transaction responses, after the descriptor is processed.

EN_DSCR_UP Field

The DPDMA updates the descriptor by writing status and timestamp information back to DDR memory. If this bit is not set, the DPDMA does not update the descriptor.

IGNR_DONE Field

If this bit is set, the DPDMA ignores the done bit and processes the descriptor even when the done bit is set (Table 33-9).

Table 33-9: **IGNR_DONE Descriptor Field**

IGNR_DONE	DONE	Action
1	x	DPDMA processes the descriptor.
0	0	DPDMA processes the descriptor.
0	1	DPDMA does not process the descriptor and raises an interrupt to indicate that it read the descriptor with DONE set.

BURST_TYPE Field

- `0`: DPDMA uses the INCR type burst for a data read.
- `1`: DPDMA uses the FIXED type burst for a data read.

ARCACHE Field

The DPDMA uses these bits during a data read. The DSCR read transaction gets ARCACHE bits from the DPDMA APB register.

ARPROT Field

The DPDMA uses these bits to generate the ARPROT [2:0] bit during AXI command generation for a data read (Table 33-10).

Table 33-10: **ARPROT Descriptor Field**

AXI ARPROT[2:0]	Value
0	ARPROT[0] from the descriptor.
1	TZ_SLCR_DPDMA
2	ARPROT[1] from the descriptor.

For more information on the previous values, see the AXI3 specification.

MODE Field

The DPDMA supports two modes of operation (controlled by the mode bit (27) in the descriptor).

- The contiguous mode is supported for systems where a large set of contiguous memory is available. The transfer size must be an integer multiple of the frame or line (the descriptor payload must end at a line or frame boundary). Software can choose to have a whole frame descriptor payload. The DPDMA uses the stride information along with the horizontal line resolution to determine the end of line and start of the next line.

  For a pixel resolution of 4-bytes, it can take up to 20 KB to store a single line in memory. It is difficult to assign 20 KB of contiguous memory in the Linux environment. This creates a support requirement for the fragmented descriptor mode.

  - A contiguous descriptor cannot store more than a single frames worth of data.
  - The transfer size should be an integer multiple of the line size.
- In fragmented mode, the maximum resolution line supported on the DisplayPort subsystem is 20 KB. Under a Linux system, the smallest resolution of memory available is 4 KB (MMU resolution). This mode is only used if the line size is more than 4 KB. Software can divide a single line into multiple fragments to store the single line data payload in a non-contiguous space.

  In fragmented mode, each descriptor is divided in up to five fragments. Each fragment can store up to 4 KB of data. Software can divide line payload (20 KB) in five sub-payloads. Because it is possible to start and end a data payload on a non-4 KB boundary, software can use up to five fragments to store a whole line, and the DPDMA determines an end of fragment and an end of line.

  - The fragment descriptor transfer size must be the same as the line size.
    - The fragmented descriptor only holds one lines worth of data.
    - This is used if a line size is more than 4k and software cannot allocate a contiguous space.
  - All fragmented addresses must be 256-byte aligned.

The example in Table 33-11 uses a line size of 10 KB with the largest set of contiguous memory available set at 4 KB. The start of the line data pay load is not 4K aligned, (start address is `x0000_FF00`) The source address for each fragment must be 256-byte aligned. It is acceptable to use less than five fragments, the DPDMA knows this from the line size (horizontal resolution) information provided in the descriptor.

Table 33-11: **Fragmented Descriptor Example**

Fragment Address	Actual Address	Size
SRC_ADDR	0000_AF00	256-bytes
SRC_ADDR2	0000_C000	4096
SRC_ADDR3	0000_D000	4096
SRC_ADDR4	0000_F000	1552
SRC_ADDR5	N/A	

LAST_DSCR Field

- `1`: The current descriptor is the last descriptor in the chain. The next address is not valid and the DPDMA should stop operation.
- `0`: After the DPDMA is done processing the current descriptor, the DPDMA fetches the next descriptor from the NEXT ADDR.

LAST DSCR OF FRAME Field

If this bit is set by software on the descriptor, it indicates that this is the last descriptor of the frame. After the DPDMA is done processing this descriptor, it fetches the first descriptor of the next frame.

EN_CRC_CHK Field

- `1`: CRC information stored at the end of the descriptor is valid. The DPDMA should only process the descriptor if the CRC is valid.
- `0`: CRC information is invalid and the DPDMA should not check the CRC.

DONE Field

- `1`:
    - Read From DPDMA. If this bit is set upon a read, the DPDMA considers this as an error condition (if the IGNR_DONE bit is not set) and generates an interrupt. This indicates that software fell behind the DPDMA and the DPDMA reached end of the chain unwillingly
    - Write From DPDMA. The DPDMA writes a `1` to this bit after it is done processing the descriptor, if the descriptor update is requested.
    - Software uses the LAST_DSCR to indicate an end of the task.
- `0`: This bit is `0` when software writes to the descriptor. If the descriptor update is requested, the DPDMA updates the DONE and time stamp information.

TIME_STAMP_LSB and TIME_STAMP_MSB Fields

The DPDMA updates the 42-bit time stamp information after it is done processing the descriptor. This functionally can be enabled by EN_DSCR_UP. The time stamp value is captured when the DPDMA starts processing the descriptor.

XFER_SIZE Field

This field indicates the total payload size in bytes.

- The contiguous mode valid transfer size can be following one or more lines (must be integer multiple of line) or the size of one frame. A frame example would use a 128 x 32 image resolution with pixel resolution 4-bytes, it should be `5'd16384`.
- The fragmented mode must indicate line size in bytes, XFER_SIZE must be same as LINE_SIZE.

LINE_SIZE Field

An 18-bit field indicates the size of the line in bytes.

STRIDE Field

A 14-bit field indicates the stride value in a 16-byte resolution. This field is only used in contiguous mode when the transfer size is larger than the line size. It is not used in fragmented mode, as it is always line wide. The stride value must be 256-byte aligned.

ADDR_EXT Field

The 16-bit address extension is used for the NEXT_ADDR_EXT and SRC_ADDR_EXT addresses. This field is used with the NEXT DSCR and SRC ADDR field to generate 48-bit address.

IMPORTANT: *The following addresses must be 256-byte aligned.*

- The DPDMA uses the NEXT_ADDR address for the next descriptor fetch if the LAST_DSCR is not set.
- Start address (SRC_ADD) of the data payload.
- The 16-bit address extension (ADDR_EXT_23) is for the SRC_ADDR2_EXT and SRC_ADDR3_EXT addresses. This field is used with the following fields to generate a 48-bit address.
- The 16-bit address extension (ADDR_EXT_45) is for the SRC_ADDR4_EXT and SRC_ADDR5_EXT addresses. This field is used with the following fields to generate a 48-bit address.

Descriptor Identifier Fields

Software generates unique 16-bit IDs for each descriptor. This information can be used by software to track the location of the DPDMA. Hardware does not check if ID values are unique. The DPDMA provides a descriptor ID of the current descriptor under process in the APB register. By reading this register, software can determine the location in the DPDMA channel within a descriptor chain.

CRC Field

The CRC is calculated using a 128-bit sum. This field is only valid if EN_CRC_CHK is set. Software generates the CRC and stores it, along with DSCR. When the CRC check is enabled, the DPDMA uses the CRC field to verify the data integrity. To calculate the CRC, the following steps are used.

1. Initialize CRC descriptor field to zero.
2. The descriptor size is 512 bits. The CRC is calculated using 32-bit addition of 16, 32-bit words.
3. Any carry generated during an addition is not used.
4. CRC = word[0] + word[1] + ... + word[15].

The received descriptor is checked against the CRC using following scheme.

1. Calculated CRC = word[0] + word[1] + ... + word[14].
2. Word[15] == calculated CRC.

The DSCR registers reflect the current states from hardware. When the DMA channel is running these multiple registers keep updating (some of these might have partial updates), whereas some registers may still hold the previous status. In order to avoid partial reads:

1. The channel can be paused and read, however, this is not reasonable as it takes time to pause and it disturbs the current streaming.

2. The channel can be read at specific timing. Once a new descriptor is scheduled there's an interrupt (sync) and the DMA channel operates on the descriptor. The descriptor states stay the same for that period and they can be read in the beginning of the period upon the interrupt. It's unlikely to read partially, but the inner descriptor states such as current payload address will keep on changing within a descriptor period.

DisplayPort Controller Clocking

The DisplayPort controller operates in different clock domains. Table 33-12 summarizes the clocks.

Table 33-12: **DisplayPort Controller Clock Domains**

Interface/Block Name	Clock(s)
DPDMA to A/V manager	AXI4 memory mapped clock
Live video and graphics	Live video clock
Live audio	Live audio clock
A/V manager	Video master clock, APB clock, audio clock
Video rendering pipe	Video master clock, APB clock
Audio mixer	Audio clock, APB clock
DisplayPort source controller	Video master clock, audio clock, link layer clock.
Live video output	Live video clock
Live audio output	Live audio clock

Figure 33-17 shows the clock domains used in the controller.

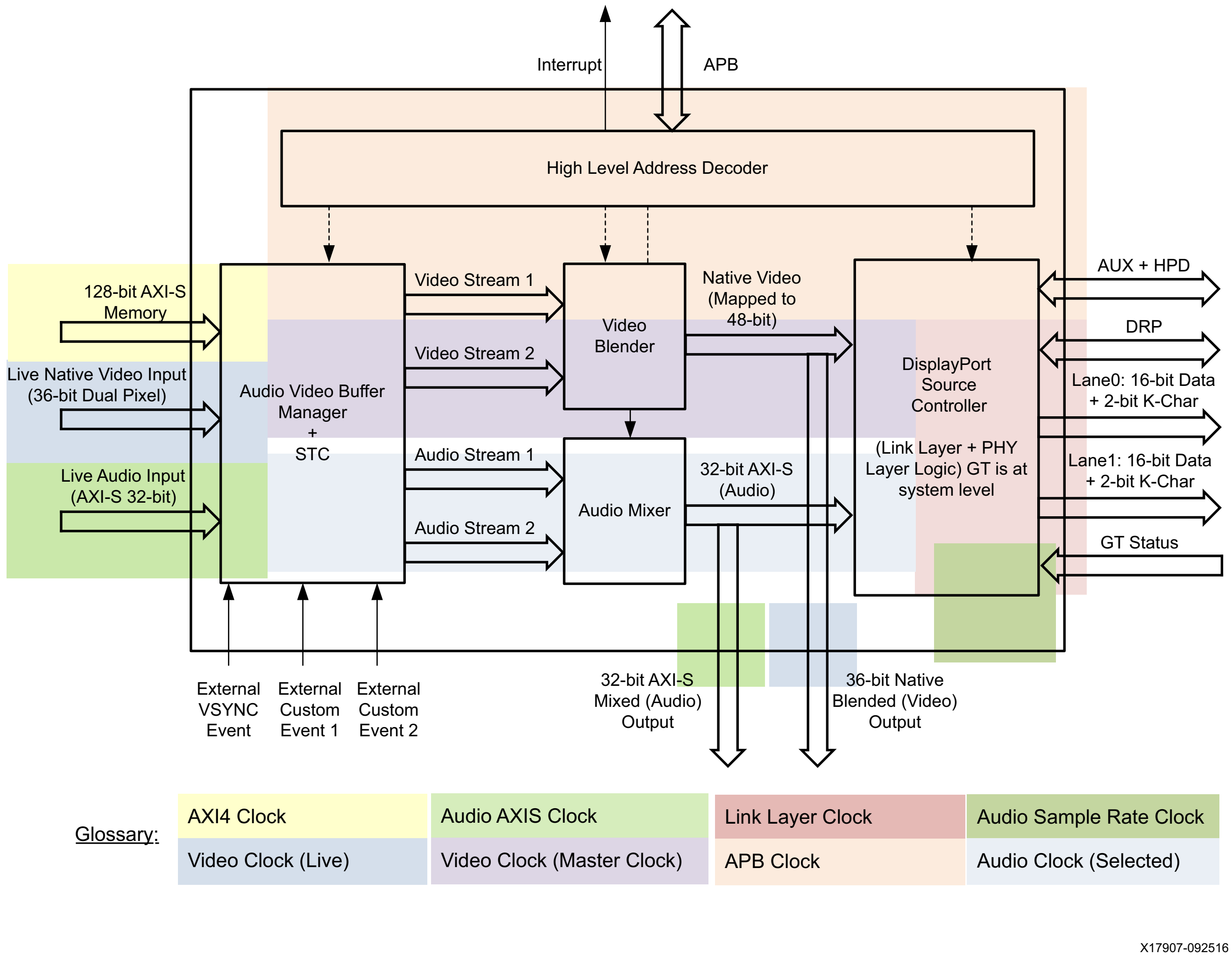

Interrupt
APB
High Level Address Decoder
128-bit AXI-S Memory
Live Native Video Input (36-bit Dual Pixel)
Live Audio Input (AXI-S 32-bit)
Audio Video Buffer Manager + STC
Video Stream 1
Video Stream 2
Audio Stream 1
Audio Stream 2
Video Blender
Audio Mixer
Native Video (Mapped to 48-bit)
32-bit AXI-S (Audio)
DisplayPort Source Controller
(Link Layer + PHY Layer Logic) GT is at system level
AUX + HPD
DRP
Lane0: 16-bit Data + 2-bit K-Char
Lane1: 16-bit Data + 2-bit K-Char
GT Status
External VSYNC Event
External Custom Event 1
External Custom Event 2
32-bit AXI-S Mixed (Audio) Output
36-bit Native Blended (Video) Output
Glossary:
AXI4 Clock
Audio AXIS Clock
Link Layer Clock
Audio Sample Rate Clock
Video Clock (Live)
Video Clock (Master Clock)
APB Clock
Audio Clock (Selected)
X17907-092516

Figure 33-17: **Block Level Clocking**

PS-PL Clocking Interface

Figure 33-18 shows the PL live video clocking of the DisplayPort controller.

RECOMMENDED: *When using live clock input from the PL, the VTC clock source selection must be selected as live input from the PL. Since the live interface also interfaces with the video processing pipeline, the same live clock is used by the rest of the pipeline in the PS.*

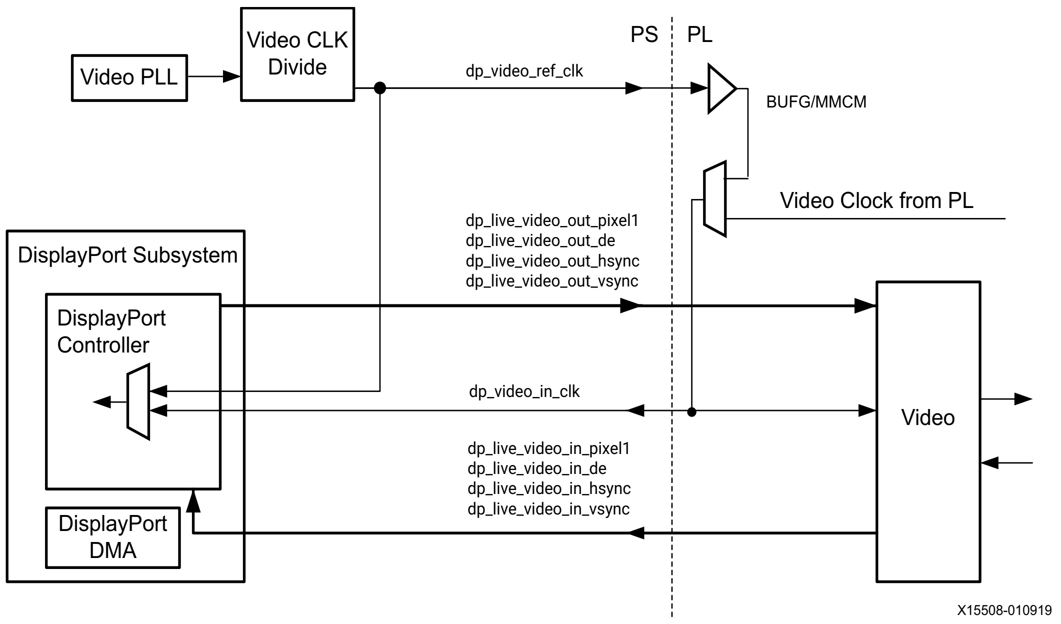


Figure 33-18: **Clocking for Live Video**

Note: While using the live video interface, the user must select VPLL as the input reference clock for the pixel clock or the live clock input from PL.

Figure 33-19 shows the details of the live audio clocking interface.

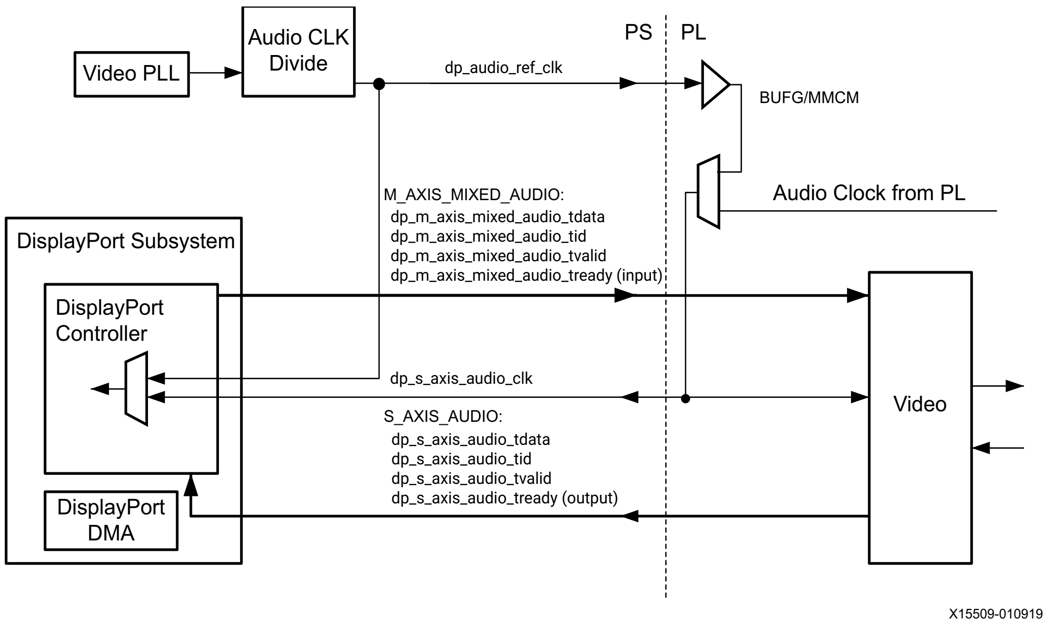


Figure 33-19: **Clocking for Live Audio**

Register Overview

Table 33-13 lists the DisplayPort configuration registers (DB register set).

Table 33-13: **DisplayPort Configuration Registers**

Register Type	Register Name	Description
DisplayPort configuration (cont'd)[1]	DP_LINK_BW_SET	Sets the value of the main link bandwidth for the sink device.
	DP_LANE_COUNT_SET	To set the lane count.
	DP_ENHANCED_FRAME_EN	To enable enhanced framing.
	DP_TRAINING_PATTERN_SET	To force training pattern.
	DP_LINK_QUAL_PATTERN_SET	To transmit the link quality pattern.
	DP_SCRAMBLING_DISABLE	
	DP_DOWNSPREAD_CTRL	For down-spreading control.
	DP_SOFTWARE_RESET	Soft reset of DisplayPort controller.
	DP_COMP_PATTERN_80BIT_1	32 bits of a 80-bit custom pattern that is used for the LINK quality test. These bits are valid when bit 2 of DP_LINK_QUAL_PATTERN_SET register is set to a `1`.
	DP_COMP_PATTERN_80BIT_2	32 bits of a 80-bit custom pattern that is used for the LINK quality test. These bits are valid when bit 2 of DP_LINK_QUAL_PATTERN_SET register is set to a `1`.
	DP_COMP_PATTERN_80BIT_3	32 bits of a 80-bit custom pattern that is used for the LINK quality test. These bits are valid when bit 2 of DP_LINK_QUAL_PATTERN_SET register is set to a `1`.
	DP_TRANSMITTER_ENABLE	Enable the basic operations of the transmitter.
	DP_MAIN_STREAM_ENABLE	Enable the transmission of main link video information.
	DP_FORCE_SCRAMBLER_RESET	Reads from this register always return `0x0`.
	DP_VERSION_REGISTER	DisplayPort controller version register.
	DP_CORE_ID	Returns the unique identification code of the DisplayPort controller and the current revision level.
	DP_AUX_COMMAND_REGISTER	
	DP_AUX_WRITE_FIFO	FIFO containing up to 16 bytes of write data for the current AUX channel command.
	DP_AUX_ADDRESS	Specifies the address for the current AUX channel command.

Table 33-13: **DisplayPort Configuration Registers *(Cont'd)***

Register Type	Register Name	Description
DisplayPort configuration (cont'd)[(1)]	DP_AUX_CLOCK_DIVIDER	Contains the clock divider value for generating the internal 1 MHz clock from the APB host interface clock. The clock divider register provides integer division only and does not support fractional APB clock rates. For example, set to 75 for a 75 MHz APB clock.
	DP_TX_USER_FIFO_OVERFLOW	Indicates an overflow in the user FIFO. The event can occur if the video rate does not match the transfer unit size programming.
	DP_INTERRUPT_SIGNAL_STATE	Contains the raw signal values for the conditions that can cause an interrupt.
	DP_AUX_REPLY_DATA	Maps to the internal FIFO which contains up to 16 bytes of information received during the AUX channel reply. Reply data is read from the FIFO starting with byte 0. The number of bytes in the FIFO corresponds to the number of bytes requested.
	DP_AUX_REPLY_CODE	Reply code received from the most recent AUX channel request. The AUX reply code corresponds to the code from the DisplayPort specification.
	DP_AUX_REPLY_COUNT	Provides an internal counter of the number of AUX reply transactions received on the AUX channel. Writing to this register clears the count.
	DP_REPLY_DATA_COUNT	Returns the total number of data bytes actually received during a transaction. This register does not use the length byte of the transaction header.
	DP_REPLY_STATUS	AUX transaction read-only status register.
	DP_HPD_DURATION	Duration of the HPD pulse in microseconds.
	DP_MAIN_STREAM_HTOTAL	Specifies the total number of clocks in the horizontal framing period for the main stream video signal.
	DP_MAIN_STREAM_VTOTAL	Provides the total number of lines in the main stream video frame.
	DP_MAIN_STREAM_HSWIDTH	Sets the width of the horizontal sync pulse.
	DP_MAIN_STREAM_VSWIDTH	Sets the width of the vertical sync pulse.
	DP_MAIN_STREAM_HRES	Horizontal resolution of the main stream video source.
	DP_MAIN_STREAM_VRES	Vertical resolution of the main stream video source.
	DP_MAIN_STREAM_HSTART	Number of clocks between the leading edge of the horizontal sync and the start of active data.

Table 33-13: **DisplayPort Configuration Registers *(Cont'd)***

Register Type	Register Name	Description
DisplayPort configuration (cont'd)[(1)]	DP_MAIN_STREAM_VSTART	Number of lines between the leading edge of the vertical sync and the first line of active data.
	DP_MAIN_STREAM_MISC0	Miscellaneous stream attributes. Implements the attribute information contained in the DisplayPort MISC0 register described in section 2.2.4 of the DisplayPort standard.
	DP_MAIN_STREAM_MISC1	Miscellaneous stream attributes. Implements the attribute information contained in the DisplayPort MISC1 register described in section 2.2.4 of the DisplayPort standard.
	DP_MAIN_STREAM_M_VID	M value for the video stream as computed by the source. If synchronous clocking mode is used, this register must be written with the M value.
	DP_MSA_TRANSFER_UNIT_SIZE	Sets the size of a transfer unit in the framing logic On reset, transfer size is set to 64.
	DP_MAIN_STREAM_N_VID	N value for the video stream as computed by the source. If synchronous clocking mode is used, this register must be written with the N value.
	DP_USER_PIX_WIDTH	User pixel width size.
	DP_USER_DATA_COUNT_PER_LANE	This register is used to translate the number of pixels per line to the native internal 16-bit datapath.
	DP_MIN_BYTES_PER_TU	Programs source to use the minimum number of bytes per transfer unit. The calculation should be done based on the DisplayPort specification.
	DP_FRAC_BYTES_PER_TU	Calculating minimum bytes per transfer unit is often not a whole number. This register is used to hold the fractional component.
	DP_INIT_WAIT	This register defines the number of initial wait cycles at the start of a new line by the framing logic. This allows enough data to be buffered in the input FIFO.
	DP_PHY_RESET	Reset the transmitter PHY.
	DP_PHY_VOLTAGE_DIFF_LANE_0	Controls the differential voltage swing for lane 0 of the DisplayPort link.
	DP_PHY_VOLTAGE_DIFF_LANE_1	Controls the differential voltage swing for lane 1 of the DisplayPort link.
	DP_TRANSMIT_PRBS7	Enable the pseudo-random bit sequence 7 pattern transmission for link quality assessment.
	DP_PHY_CLOCK_SELECT	Instructs the PHY PLL to generate the proper clock frequency for the required link rate.
	DP_TX_PHY_POWER_DOWN	Control PHY power down.
	DP_PHY_PRECURSOR_LANE_0	Set the pre-cursor level (post cursor 1 for PS-GTR) for lane 0 of the DisplayPort link.

Table 33-13: **DisplayPort Configuration Registers *(Cont'd)***

Register Type	Register Name	Description
DisplayPort configuration (cont'd)[(1)]	DP_PHY_PRECURSOR_LANE_1	Set the pre-cursor level for lane 1 of the DisplayPort link.
	DP_PHY_POSTCURSOR_LANE_0	Set the post-cursor level (post cursor 2) for lane 0 of the DisplayPort link.
	DP_PHY_POSTCURSOR_LANE_1	Set the post-cursor level (post cursor 2) for lane 1 of the DisplayPort link.
	DP_PHY_STATUS	Provides the current status from the PHY.
	DP_TX_AUDIO_CONTROL	Enables audio stream packets in main link and provides buffer control.
	DP_TX_AUDIO_CHANNELS	Used to input active channel count. Transmitter collects audio samples based on this information.
	DP_TX_AUDIO_INFO_DATA{0:7}	Words formatted as per CEA 861-C info frame.
	DP_TX_M_AUD	M value of audio stream as computed by the transmitter.
	DP_TX_N_AUD	N value of audio stream as computed by the transmitter.
	DP_TX_AUDIO_EXT_DATA0	Word formatted as per the extension packet described in the protocol specification. Extended packet is fixed to 32-byte length. The controller has buffer space for only one extended packet.
	DP_TX_AUDIO_EXT_DATA1	2nd word of the 9 words of the extended packet.
	DP_TX_AUDIO_EXT_DATA2	3rd word of the 9 words of the extended packet.
	DP_TX_AUDIO_EXT_DATA3	4th word of the 9 words of the extended packet.
	DP_TX_AUDIO_EXT_DATA4	5th word of the 9 words of the extended packet.
	DP_TX_AUDIO_EXT_DATA5	6th word of the 9 words of the extended packet.
	DP_TX_AUDIO_EXT_DATA6	7th word of the 9 words of the extended packet.
	DP_TX_AUDIO_EXT_DATA7	8th word of the 9 words of the extended packet.
	DP_TX_AUDIO_EXT_DATA8	9th word of the 9 words of the extended packet.
	DP_INT_STATUS	Interrupt status register for intrN. This is a sticky register that holds the value of the interrupt until cleared by a value of `1`.
	DP_INT_MASK	Interrupt mask register for intrN. This is a read-only location and can be atomically altered by either the IDR or the IER.
	DP_INT_EN	Interrupt enable register (IER). A write of to this location unmasks the interrupt. (IMR: 0)
	DP_INT_DS	Interrupt disable register (IDR). A write of one to this location masks the interrupt. (IMR: 1)
	V_BLEND_BG_CLR_0	Sets background color of the layers.
	V_BLEND_BG_CLR_1	Sets background color of the layers.

Table 33-13: **DisplayPort Configuration Registers *(Cont'd)***

Register Type	Register Name	Description
DisplayPort configuration (cont'd)(1)	V_BLEND_BG_CLR_2	Sets background color of the layers.
	V_BLEND_SET_GLOBAL_ALPHA_REG	To set the global alpha register.
	V_BLEND_OUTPUT_VID_FORMAT	
	V_BLEND_LAYER0_CONTROL	Layer 0 is always video pixel.
	V_BLEND_LAYER1_CONTROL	Layer 1 is always graphics.
	V_BLEND_RGB2YCBCR_COEFF{0:8}	Coefficient values from matrix. A total of 9 values are needed to form a 3x3 matrix. The value is scaled by 2^{12} and stored in a 15-bit signed format, (1-bit reserved). 12 bits out of the 15 represent a fractional value and 2 bits for the decimal value and one signed bit. The order of programming values is from v0 - v8.
	V_BLEND_IN1CSC_COEFF{0:8}	
	V_BLEND_LUMA_IN1CSC_OFFSET	Offset values for Y before and after matrix multiplication for input color space conversion.
	V_BLEND_CR_IN1CSC_OFFSET	Offset values for CR before and after matrix multiplication for input color space conversion.
	V_BLEND_CB_IN1CSC_OFFSET	Offset values for CB before and after matrix multiplication for input color space conversion.
	V_BLEND_LUMA_OUTCSC_OFFSET	Offset values for Y before and after matrix multiplication for output color space conversion.
	V_BLEND_CR_OUTCSC_OFFSET	Offset values for CR before and after matrix multiplication for output color space conversion.
	V_BLEND_CB_OUTCSC_OFFSET	Offset values for color CB before and after matrix multiplication for output color space conversion.
	V_BLEND_IN2CSC_COEFF0	Coefficient values from matrix. A total of 9 values are needed to form a 3x3 matrix. The value is scaled by 2^{12} and stored in 15-bit signed format (1-bit is reserved). The order of programming values is same as described in V_BLEND_RGB2YCBCR_COEFF.
	V_BLEND_IN2CSC_COEFF1	
	V_BLEND_IN2CSC_COEFF2	
	V_BLEND_IN2CSC_COEFF3	
	V_BLEND_IN2CSC_COEFF4	
	V_BLEND_IN2CSC_COEFF5	
	V_BLEND_IN2CSC_COEFF6	
	V_BLEND_IN2CSC_COEFF7	
	V_BLEND_IN2CSC_COEFF8	
	V_BLEND_LUMA_IN2CSC_OFFSET	Offset values for Y before and after matrix multiplication for input color space conversion.
	V_BLEND_CR_IN2CSC_OFFSET	Offset values for CR before and after matrix multiplication for input color space conversion.
	V_BLEND_CB_IN2CSC_OFFSET	Offset values for CB before and after matrix multiplication for input color space conversion.

Table 33-13: **DisplayPort Configuration Registers *(Cont'd)***

Register Type	Register Name	Description
DisplayPort configuration (cont'd)[(1)]	V_BLEND_CHROMA_KEY_ENABLE	
	V_BLEND_CHROMA_KEY_COMP1	[11:0] B component of the key minimum value. [27:16] B component of the key maximum value.
	V_BLEND_CHROMA_KEY_COMP2	
	V_BLEND_CHROMA_KEY_COMP3	
	AV_BUF_FORMAT	This register should be programmed based on the video/graphics packing format in memory. DisplayPort unpacker works based on this.
	AV_BUF_NON_LIVE_LATENCY	The memory fetch latency. This parameter is used to offset the early VTC. The maximum latency supported is 412.
	AV_CHBUF0	Channel enable, flush, and burst length to be programmed based on video formats. Each channel can be programmed with independent burst length. Channel 0 must be always enabled for any video mode. Channel 1 and 2 should be enabled for planar modes. Channel 3 for graphics. Channel 4 and 5 for audio modes.
	AV_CHBUF1	
	AV_CHBUF2	
	AV_CHBUF3	
	AV_CHBUF4	
	AV_CHBUF5	
	AV_BUF_STC_CONTROL	
	AV_BUF_STC_INIT_VALUE0	
	AV_BUF_STC_INIT_VALUE1	
	AV_BUF_STC_ADJ	A write to this register triggers STC adjust.
	AV_BUF_STC_VIDEO_VSYNC_TS_REG0	STC TS with VSYNC event.
	AV_BUF_STC_VIDEO_VSYNC_TS_REG1	STC TS with VSYNC event.
	AV_BUF_STC_EXT_VSYNC_TS_REG0	STC TS with VSYNC event.
	AV_BUF_STC_EXT_VSYNC_TS_REG1	STC TS with VSYNC event.
	AV_BUF_STC_CUSTOM_EVENT_TS_REG0	STC TS with custom event 1.
	AV_BUF_STC_CUSTOM_EVENT_TS_REG1	STC TS with custom event 1.
	AV_BUF_STC_CUSTOM_EVENT2_TS_REG0	STC TS with custom event 2 (can be audio TS).
	AV_BUF_STC_CUSTOM_EVENT2_TS_REG1	STC TS with custom event 2 (can be audio TS).
	AV_BUF_STC_SNAPSHOT0	
	AV_BUF_STC_SNAPSHOT1	
	AV_BUF_OUTPUT_AUDIO_VIDEO_SELECT	Select the buffer manager outputs.
	AV_BUF_HCOUNT_VCOUNT_INT0	If the early VTC timing values (VCOUNT and HCOUNT) match the values programmed in this register and the corresponding interrupt mask is enabled, then an interrupt is generated.
	AV_BUF_HCOUNT_VCOUNT_INT1	

Table 33-13: **DisplayPort Configuration Registers *(Cont'd)***

Register Type	Register Name	Description
DisplayPort configuration (cont'd)[(1)]	AV_BUF_DITHER_CONFIG	This register is used for configuring dither functions.
	DITHER_CONFIG_SEED0	To set seed for LFSR0.
	DITHER_CONFIG_SEED1	
	DITHER_CONFIG_SEED2	
	DITHER_CONFIG_MAX	To set the maximum output value on video pixel (at the blender output towards the DisplayPort)
	DITHER_CONFIG_MIN	To set the minimum output value on video pixel (at the blender output towards the DisplayPort)
	PATTERN_GEN_SELECT	
	AUD_PATTERN_SELECT1	
	AUD_PATTERN_SELECT2	
	AV_BUF_AUD_VID_CLK_SOURCE	When live video from the PL is absent, then the internal clock is a video pipeline clock. If the live video is present, then clock from PL is the video pipe line clock. Similarly, for the audio you can select from either the PS or PL clock.
	AV_BUF_SRST_REG	
	AV_BUF_AUDIO_RDY_INTERVAL	Debug register.
	AV_BUF_AUDIO_CH_CONFIG	
	AV_BUF_GRAPHICS_COMP0_SCALE_FACTOR	Scaling factor for graphics for component #. For 4 bits, scale factor is 16/15 x 2^{16} = `0x11111` For 5 bits, scale factor is 32/31 x 2^{16} = `0x10842` For 6 bits, scale factor is 64/63 x 2^{16} = `0x10410`. For 8 bits, scale factor is 256/255 x 2^{16} = `0x10101` For 10 bits, scale factor is 1024/1023 x 2^{16} = `0x10040` For BPC = 12, no scaling is done. This register is unused and can be default.
	AV_BUF_GRAPHICS_COMP1_SCALE_FACTOR	
	AV_BUF_GRAPHICS_COMP2_SCALE_FACTOR	
	AV_BUF_VIDEO_COMP0_SCALE_FACTOR	
	AV_BUF_VIDEO_COMP1_SCALE_FACTOR	
	AV_BUF_VIDEO_COMP2_SCALE_FACTOR	
	AV_BUF_LIVE_VIDEO_COMP0_SF	
	AV_BUF_LIVE_VIDEO_COMP1_SF	
	AV_BUF_LIVE_VIDEO_COMP2_SF	
	AV_BUF_LIVE_VID_CONFIG	Programmable option to configure Cb or Cr first, when YUV422 mode is enabled.

Table 33-13: **DisplayPort Configuration Registers *(Cont'd)***

Register Type	Register Name	Description
DisplayPort configuration (cont'd)(1)	AV_BUF_LIVE_GFX_COMP0_SF	Scaling factor for live graphics color comp#. For 4 bits, scale factor is 16/15 x 2^{16} = `0x11111` For 5 bits, scale factor is 32/31 x 2^{16} = `0x10842` For 6 bits, scale factor is 64/63 x 2^{16} = `0x10410`. For 8 bits, scale factor is 256/255 x 2^{16} = `0x10101` For 10 bits, scale factor is 1024/1023 x 2^{16} = `0x10040` For BPC = 12, no scaling is done. This register is unused and can be default.
	AV_BUF_LIVE_GFX_COMP1_SF	
	AV_BUF_LIVE_GFX_COMP2_SF	
	AV_BUF_LIVE_GFX_CONFIG	Programmable option to configure Cb or Cr first, when YUV422 mode is enabled.
	AUDIO_MIXER_VOLUME_CONTROL	Setting value to 8192 means no volume change (1.0 scaling factor).
	AUDIO_MIXER_META_DATA	
	AUD_CH_STATUS_REG0	Audio channel status bits 31 to 0.
	AUD_CH_STATUS_REG1	Audio channel status bits 63 to 32.
	AUD_CH_STATUS_REG2	Audio channel status bits 95 to 64.
	AUD_CH_STATUS_REG3	Audio Channel status bits 127 to 96.
	AUD_CH_STATUS_REG4	Audio Channel status bits 159 to 128.
	AUD_CH_STATUS_REG5	Audio Channel status bits 191 to 160.
	AUD_CH_A_DATA_REG0	User data bits 31 to 0.
	AUD_CH_A_DATA_REG1	User data bits 63 to 32.
	AUD_CH_A_DATA_REG2	User data bits 95 to 64.
	AUD_CH_A_DATA_REG3	User data bits 127 to 96.
	AUD_CH_A_DATA_REG4	User data bits 159 to 128.
	AUD_CH_A_DATA_REG5	User data bits 191 to 160.
	AUD_CH_B_DATA_REG0	User data bits 31 to 0.
	AUD_CH_B_DATA_REG1	User data bits 63 to 32.
	AUD_CH_B_DATA_REG2	User data bits 95 to 64.
	AUD_CH_B_DATA_REG3	User data bits 127 to 96.
	AUD_CH_B_DATA_REG4	User data bits 159 to 128.
	AUD_CH_B_DATA_REG5	User data bits 191 to 160.
	PATGEN_CRC_R	16-bit CRC calculated on the first component of video output from the internal test pattern generator.

Table 33-13: **DisplayPort Configuration Registers *(Cont'd)***

Register Type	Register Name	Description
DisplayPort configuration (cont'd)[1]	PATGEN_CRC_G	16-bit CRC calculated on the second component of video output from the internal test pattern generator.
	PATGEN_CRC_B	16-bit CRC calculated on the third component of video output from the internal test pattern generator.

1. Refer to *Zynq UltraScale+ MPSoC Register Reference* (UG1087) [Ref 4] for more information.

Table 33-14 summarizes the DisplayPort DMA egisters (DPDMA register set).

Table 33-14: **DisplayPort DMA Registers**

Register Type	Register Name	Description
Error response	DPDMA_ERR_CTRL	Enable/disable a error response.
Interrupts	DPDMA_ISR	Interrupt status register for intrN. This is a sticky register that holds the value of the interrupt until cleared by a value of `1`.
	DPDMA_IMR	Interrupt mask register for intrN. This is a read-only location and can be atomically altered by either the IDR or the IER.
	DPDMA_IEN	Interrupt enable register. A write of `1` to this location unmasks the interrupt. (IMR: 0)
	DPDMA_IDS	Interrupt disable register. A write of `1` one to this location masks the interrupt. (IMR: 1)
	DPDMA_EISR	Interrupt status register for intrN. This is a sticky register that holds the value of the interrupt until cleared by a value of `1`.
	DPDMA_EIMR	Interrupt mask register for intrN. This is a read-only location and can be atomically altered by either the IDR or the IER.
	DPDMA_EIEN	Interrupt enable register. A write of `1` to this location unmasks the interrupt. (IMR: 0)
	DPDMA_EIDS	Interrupt disable register. A write of one to this location masks the interrupt. (IMR: 1)

Table 33-14: **DisplayPort DMA Registers *(Cont'd)***

Register Type	Register Name	Description
DMA control and status	DPDMA_CNTL	DPDMA global control register, holds fields which control all six channels.
	DPDMA_GBL	Global control register provides control to start or redirect any channel.
	DPDMA_ALC0_CNTL	Global control register provides control to start or redirect any channel.
	DPDMA_ALC0_STATUS	Status register.
	DPDMA_ALC0_MAX	ALC0 maximum latency register.
	DPDMA_ALC0_MIN	ALC0 minimum latency register.
	DPDMA_ALC0_ACC	ALC0 accumulated transaction latency register.
	DPDMA_ALC0_ACC_TRAN	ALC0 accumulated transaction count register.
	DPDMA_ALC1_CNTL	Global control register provides control to start or redirect any channel.
	DPDMA_ALC1_STATUS	Status register.
	DPDMA_ALC1_MAX	ALC1 maximum latency register.
	DPDMA_ALC1_MIN	ALC1 minimum latency register.
	DPDMA_ALC1_ACC	ALC1 accumulated transaction latency register.
	DPDMA_ALC1_ACC_TRAN	ALC1 accumulated transaction count register.
DMA channels	DPDMA_CH{0:5}_DSCR_STRT_ADDRE	Channel x descriptor; start extended address (Hi).
	DPDMA_CH{0:5}_DSCR_STRT_ADDR	Channel x descriptor; start address (Lo).
	DPDMA_CH{0:5}_DSCR_NEXT_ADDRE	Channel x descriptor; next extended address (Hi).
	DPDMA_CH{0:5}_DSCR_NEXT_ADDR	Channel x descriptor; next address (Lo).
	DPDMA_CH{0:5}_PYLD_CUR_ADDRE	Channel x descriptor; current payload extended address (Hi).
	DPDMA_CH{0:5}_PYLD_CUR_ADDR	Channel x descriptor; current payload address (Lo).
	DPDMA_CH{0:5}_CNTL	Channel x control.
	DPDMA_CH{0:5}_STATUS	Channel x status.
	DPDMA_CH{0:5}_VDO	Channel x video parameter.
	DPDMA_CH{0:5}_PYLD_SZ	Channel x descriptor; current payload size.
	DPDMA_CH{0:5}_DSCR_ID	Channel x descriptor; current 16-bit ID.

Programming Considerations

Note: The DisplayPort controller can sometimes report underflow and overflow status for the same frame. This scenario can occur when the DDR memory responds to a requested burst slowly. Both underflow and overflow flags can be set in the dp.DP_INT_STATUS register.

Source Controller Setup and Initialization

This section lists the procedural tasks required to achieve link communication. See Table 33-15.

Source Controller Setup

1. Place the PHY into reset. The PS-GTR reset bit in the PHY_reset [bit 1 of DP_PHY_RESET] bit should be set to `1`.

   DP_PHY_RESET = `0x01`

2. Disable the transmitter.

   DP_TRANSMITTER_ENABLE = `0x00`

3. Set the clock divider by programming the dp.DP_AUX_CLOCK_DIVIDER[clk_div] register.
4. Set DisplayPort clock speed. Program the dp.DP_PHY_CLOCK_SELECT[sel] register with the desired link speed.
5. Bring the PHY out of reset. Write 0 to the DP.DP_PHY_RESET [GT_RESET] bit.

   dp.DP_PHY_RESET = `0x00`

6. Wait for the PHY reset done and PLL lock.

   DP_PHY_STATUS bits [1:0] = `2'b11` and DP_PHY_STATUS bit [4] = `1'b1`

7. Enable the transmitter.

   DP_TRANSMITTER_ENABLE = `0x01`

8. (Optional) Turn on the interrupt mask for the HPD.

   INTERRUPT_MASK = `0x00`

Table 33-15: **Source Controller Setup and Initialization**

Task	DP Register Set	Bit Field	Register Offset	Bits	Value
Reset PHY.	DP_PHY_RESET	GT_RESET	`0x200`	1	1
Disable transmitter.	DP_TRANSMITTER_ENABLE	TX_EN	`0x0080`	0	`1b'0`
Set the clock divider.	DP_AUX_CLOCK_DIVIDER	AUX_SIGNAL_WIDTH_FILTER | CLK_DIV	`0x010C`	15:0	Refer to the register for the value.
Set DisplayPort clock.	DP_PHY_CLOCK_SELECT	SEL	`0x0234`	2:0	`0x05 =` 5.40 Gb/s link `0x03 =` 2.70 Gb/s link `0x01 =` 1.62 Gb/s link
Bring the PHY out of reset.	DP_PHY_RESET	GT_RESET	`0x200`	1	`1'b0`
Wait for reset done by checking DP_PHY_STATUS register.					
Check reset done.	DP_PHY_STATUS	RESET_LANES_0_1	`0x0280`	1:0	`2b'11` indicates reset done for lane 0 and lane 1.
Check PLL locked.	DP_PHY_STATUS	RESET_LANES_0_1	`0x0280`	4	`1b'1` indicates PLL is locked.
Enable transmitter.	DP_TRANSMITTER_ENABLE	TX_EN	`0x0080`	0	`1b'1`

Note: At this point, the source controller is initialized and ready to use. The link policy maker should be monitoring the status of HPD and taking appropriate action for connect/disconnect events or HPD interrupt pulses.

Although #DP_PHY_RESET has two bits (GT_RESET and PLL_RESET), use GT_RESET during source controller setup.

To change the PS-GTR link rate dynamically (Table 33-16):

1. Disable the transmitter.

   TRANSMITTER_ENABLE = `0x00`

2. Set DisplayPort clock speed.

   PHY_CLOCK_SELECT = desired link speed

3. Wait for the PHY rate change done and PLL lock.

   a. PHY_STATUS bits [3:0] = `4'b1111` (for two lanes) or PHY_STATUS bits [3:0] = `4'b0101` (for one lane)

   b. PHY_STATUS bits [4] = `1'b1`

4. Enable the transmitter.

   TRANSMITTER_ENABLE = `0x01`

Table 33-16: **PS-GTR Link Rate**

Task	Register	Register Field	Register Offset	Bits	Value
Disable transmitter.	DP_TRANSMITTER_ENABLE	TX_EN	`0x0080`	0	`1b'0`
Set DisplayPort clock.	DP_PHY_CLOCK_SELECT	SEL	`0x0234`	2:0	`0x05` = 5.40 Gb/s link `0x03` = 2.70 Gb/s link `0x01` = 1.62 Gb/s link
Wait for the PHY rate change done and PLL lock by checking DP_PHY_STATUS register.					
Check reset done.	DP_PHY_STATUS	RESET_LANES_0_1	`0x0280`	3:0	`4'b1111` (for 2 lanes) `4'b0101` (for 1 lane)
Check PLL locked.	DP_PHY_STATUS	RESET_LANES_0_1	`0x0280`	4	`1b'1` indicates PLL has been locked.
Enable transmitter.	DP_TRANSMITTER_ENABLE	TX_EN	`0x0080`	0	`1b'1`

Note: If the current PS-GTR rate is 1.62 Gb/s and you try to set it again to 1.62 Gb/s, the DP.DP_PHY_STATUS [RATE_CHANGE_DONE_0_1] bits will not be set. The PS-GTR reset doesn't need to be explicitly issued from the software during a dynamic link rate change, it is handled inside the PS-GTR. Therefore, unless the phy configuration changes the link rate from existing value to the desired value, the RATE_CHANGE_DONE_0_1bits won't be set.

Upon HPD Assertion

1. Read the DPCD capabilities fields out of the sink device (`0x00000`–`0x0000B`) through the AUX channel. See Table 33-17.

2. Determine values for lane count, link speed, enhanced framing mode, downspread control, and main link channel code based on each link partners' capability and needs.

3. Write the configuration parameters to the link configuration field (`0x00100`–`0x00101`) of the DPCD through the AUX channel.

   Note: Some sink devices' DPCD capability fields are unreliable. Many source devices start with the maximum transmitter capabilities and scale back as necessary to find a configuration the sink device can handle. This could be an advisable strategy instead of relying on DPCD values.

4. Equivalently, write the appropriate values to the source controller's local configuration space.

   a. LANE_COUNT_SET

b. LINK_BW_SET

c. ENHANCED_FRAME_EN

d. PHY_CLOCK_SELECT

Table 33-17: **HPD Assertion**

Task	Register	Register Field	Register Offset	Bits	Value

To set lane count.	DP_LANE_COUNT_SET	LANE_CNT	`0x004`	4:0	Possible values:
					`5b'00001`
					`5b'00010`

To set the value of the main link bandwidth for the sink device.	DP_LINK_BW_SET	BW	`0x000`	7:0	`0x06` = 1.62 Gb/s `0x0A` = 2.7 Gb/s `0x14` = 5.4 Gb/s
To enable enhanced framing.	DP_ENHANCED_FRAME_EN	ENH_FRAMING_EN	`0x008`	0	Set to `1` by the source to enable the enhanced framing symbol sequence.
Select PHY clock.	DP_PHY_CLOCK_SELECT	SEL	`0x0234`	2:0	`0x05` = 5.40 Gb/s link `0x03` = 2.70 Gb/s link `0x01` = 1.62 Gb/s link

Training Pattern 1 Procedure (Clock Recovery)

1. Turn off scrambling and set training pattern 1 in the source through direct register writes. See Table 33-18.
    - SCRAMBLING_DISABLE = `0x01`
    - TRAINING_PATTERN_SET = `0x01`
2. Turn off scrambling and set training pattern 1 in the sink DPCD (`0x00102`–`0x00106`) through the AUX channel.
3. Wait 100 μs before reading status registers for all active lanes (`0x00202`–`0x00203`) through the AUX channel.
4. If clock recovery failed, check for voltage swing or pre-emphasis level increase requests (`0x00206`–`0x00207`) and react accordingly. Run this loop up to five times. If after five iterations this has not succeeded, reduce the link speed, if at a high speed and try again. If already at a low speed, training fails.

Table 33-18: **Clock Recovery**

Task	Register	Register Field	Register Offset	Bits	Value
Scrambling disable.	DP_SCRAMBLING_DISABLE	SCR_DIS	`0x014`	0	`1b'1`
To force training pattern.	DP_TRAINING_PATTERN_SET	TP_SET	`0x00C`	1:0	`00` = Training off. `01` = Training pattern 1, used for clock recovery. `10` = Training pattern 2, used for channel equalization. `11` = Training pattern 3, used for channel equalization for controllers with DisplayPort v1.2.

Training Pattern 2 Procedure (Symbol Recovery, Interlane Alignment)

1. Turn off scrambling and set training pattern 2 in the source through direct register writes. See Table 33-19.
    - SCRAMBLING_DISABLE = `0x01`
    - TRAINING_PATTERN_SET = `0x02`
2. Turn off scrambling and set training pattern 2 in the sink DPCD (`0x00102`–`0x00106`) through the AUX channel.
3. Wait 400 µs and then read status registers for all active lanes (`0x00202`–`0x00203`) through the AUX channel.
4. Check the channel equalization, symbol lock, and interlane alignment status bits for all active lanes (`0x00204`) through the AUX channel.
5. If any of these bits are not set, check for voltage swing or pre-emphasis level increase requests (`0x00206`–`0x00207`) and react accordingly.
6. Run this loop up to five times. If after five iterations this has not succeeded, reduce the link speed if at a high speed and return to the instructions for training pattern 1. If already at a low speed, training fails.
7. Signal the end of training by enabling scrambling and setting training pattern to 0x00 in the sink device (`0x00102`) through the AUX channel.
8. On the source side, re-enable scrambling and turn off training.
    - TRAINING_PATTERN_SET = `0x00`
    - SCRAMBLING_DISABLE = `0x00`

At this point, training has completed.

Note: Training pattern 3 replaces training pattern 2 for 5.4 G link rate devices. See the DisplayPort v1.2 specification for details.

Table 33-19: **Symbol Recovery, Interlane Alignment**

Task	Register	Register Field	Register Offset	Bits	Value
Scrambling disable	DP_SCRAMBLING_DISABLE	SCR_DIS	`0x014`	`0`	`1b'1`
To force training pattern	DP_TRAINING_PATTERN_SET	TP_SET	`0x00C`	`1:0`	`00` = Training off. `01` = Training pattern 1, used for clock recovery. `10` = Training pattern 2, used for channel equalization. `11` = Training pattern 3, used for channel equalization for controllers with DisplayPort v1.2.

Enabling Main Link Video

The main link video should not be enabled until a proper video source has been provided to the source controller. Typically, the source device wants to read the EDID from the attached sink device to determine its capabilities, most importantly its preferred resolution and other resolutions that it supports should the preferred mode not be available. After a resolution has been determined, set the main stream attributes in the source controller (`0x180`–`0x1B0`). Enable the main stream (`0x084`) only when a reliable video source is available.

Note: The scrambler/descrambler must be reset after enabling the main link video. Before starting to transmit video, the source must initialize the scrambler and the link partner's descrambler. This is done by forcing a scrambler reset (`0x0C0`) before the main link is enabled.

Note: The TRANSFER UNIT size of the DisplayPort transmit controller can be set to 32/64 or any even number in between. This is system dependent (on RX buffer capabilities).

Accessing the Link Partner

The DisplayPort controller is configured through the APB host interface. The host processor interface uses the DisplayPort AUX channel to read the register space of the attached sink device and determines the capabilities of the link. Accessing DPCD and EDID information from the sink is done by writing and reading from register space `0x100` through `0x144`.

Before any AUX channel operation can be completed, you must first set the proper clock divide value in `0x10C`. This must be done only one time after a reset. The value held in this register should be equal to the frequency of apb_clk. So, if apb_clk runs at 135 MHz, the value of this register should be 135 (`'h87`). This register is required to apply a proper divide function for the AUX channel sample clock, which must operate at 1 MHz. The act of writing to the AUX_COMMAND initiates the AUX event. After an AUX request transaction is started, the host should not write to any of the control registers until the REPLY_RECEIVED bit is set to a `1`, indicating that the sink has returned a response.

Audio Management

This section contains the procedural tasks required to achieve audio communication.

Programming the DisplayPort Source

1. Disable audio by writing a `0x00` to the TX_AUDIO_CONTROL register. The disable bit also flushes the buffers in DisplayPort source and sets MUTE bit in VB-ID.
2. Write the audio information frame. Based on your requirements, this could be optional for some systems. The audio information frame consists of eight writes.
3. Write channel count to the TX_AUDIO_CHANNELS register (the value is actual count minus 1).
4. If the system is using synchronous clocking, then write the MAUD and NAUD values into the TX_AUDIO_MAUD and TX_AUDIO_NAUD registers.
5. Enable audio by writing a `0x01` to the TX_AUDIO_CONTROL register.

Reprogramming Source Audio

1. Wait a few ms (~1-2 ms) so that the DisplayPort source can complete any pending secondary transmission.
2. Disable the audio in the DisplayPort TX.
3. Wait until the video/audio clock is recovered and stable.
4. Enable the audio in the DisplayPort TX.
5. Wait for some time (1 ms).
6. Start audio transfer.

Info Packet Management

The controller provides an option to program a single information packet. The packet is transmitted to the sink after per video frame or 8192 cycles.

To change an information packet during transmission, follow these steps.

1. Disable audio, because new information packet means a new audio configuration. The disable audio also flushes the internal audio buffers.
2. Follow steps in Programming the DisplayPort Source.

Extension Packet Management

A single packet buffer is provided for the extension packet. If the extension packet is available in the buffer, the packet is transmitted as soon as there is availability in the secondary channel. The packet length is fixed to eight words (32 bytes).

Use these steps to write an extended packet in the DisplayPort source controller.

1. Write nine words (as required) into the TX_AUDIO_EXT_DATA buffer.
2. Wait for the EXT_PKT_TXD interrupt.
3. Write the new packet (follow step 1).

AUX Write Transaction

An AUX write transaction (Figure 33-20) is initiated by setting up the AUX_ADDRESS, and writing the data to the AUX_WRITE_FIFO followed by a write to the AUX_COMMAND register with the code `0x08`. Writing the command register begins the AUX channel transaction. The host should wait until either a reply received event or reply time-out event is detected. These events are detected by reading INTERRUPT_STATUS registers (either in ISR or polling mode). When the reply is detected, the host should read the AUX_REPLY_CODE register and look for the code `0x00` indicating that the AUX channel has successfully acknowledged the transaction.

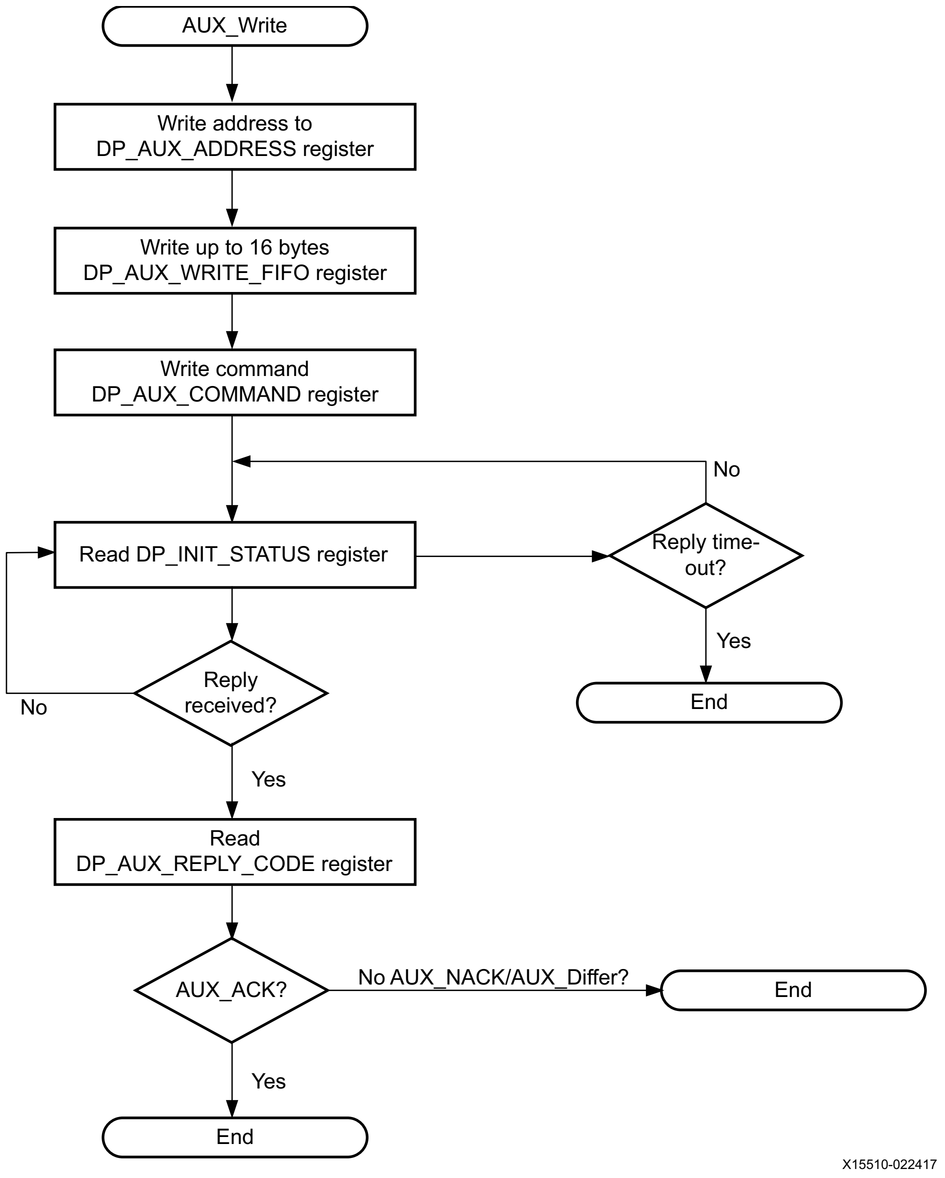

AUX_Write
Write address to DP_AUX_ADDRESS register
Write up to 16 bytes DP_AUX_WRITE_FIFO register
Write command DP_AUX_COMMAND register
No
Read DP_INIT_STATUS register
Reply time-out?
Yes
End
Reply received?
No
Yes
Read DP_AUX_REPLY_CODE register
AUX_ACK?
No AUX_NACK/AUX_Differ?
End
Yes
End
X15510-022417

Figure 33-20: **AUX Write Transaction**

AUX Read Transaction

The AUX read transaction (Figure 33-21) is prepared by writing the transaction address to the AUX_ADDRESS register. After it is set, the command and the number of bytes to read are written to the AUX_COMMAND register. After initiating the transfer, the host should wait for an interrupt or poll the INTERRUPT_STATUS register to determine when a reply is received. When the REPLY_RECEIVED signal is detected, the host can read the requested data bytes from the AUX_REPLY_DATA register. This register provides a single address interface to a byte FIFO which is 16 elements deep. Reading from this register automatically advances the internal read pointers for the next access.

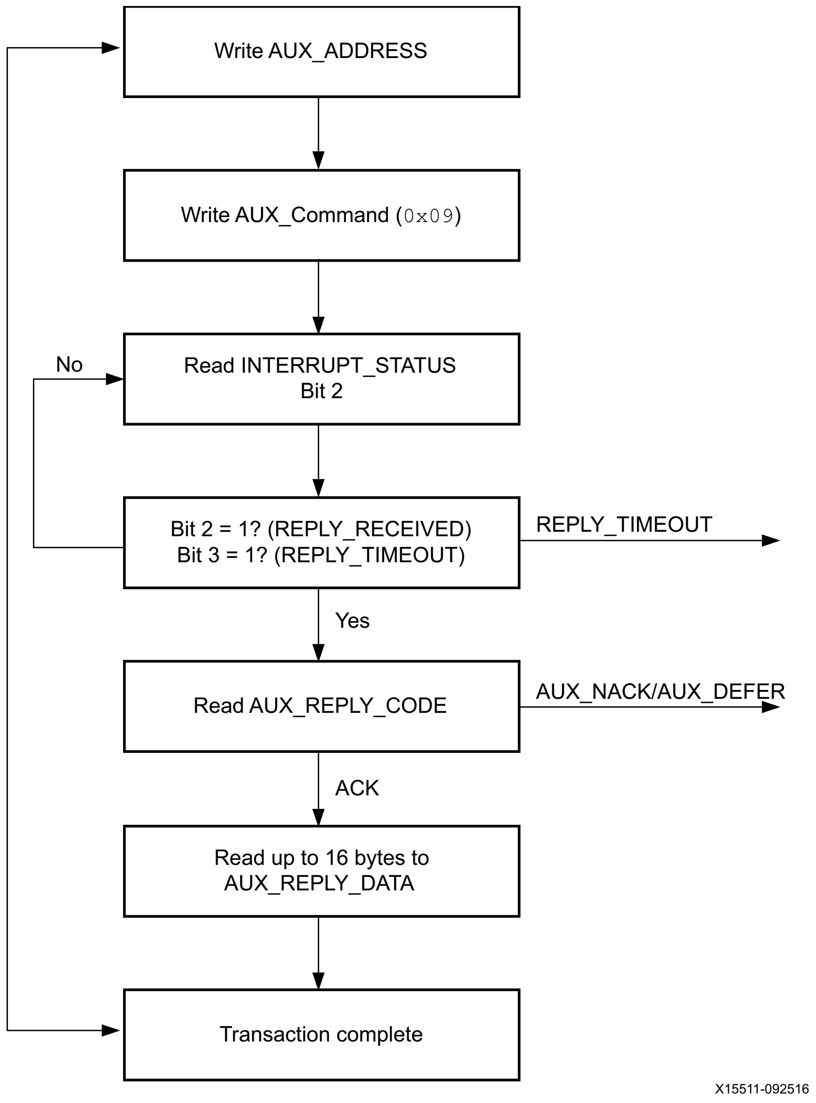


Figure 33-21: **AUX Read Transaction**

Commanded I2C Transactions

The controller supports a special AUX channel command intended to make I2C over AUX transactions faster and easier to perform. In this case, the host bypasses the external I2C master/slave interface and initiates the command by directly writing to the register set. The sequence for performing these transactions is exactly the same as a native AUX channel transaction with a change to the command written to the AUX_COMMAND register. The supported I2C commands are summarized in Table 33-20.

Table 33-20: **I2C Commands**

Aux_Command[11:8]	Command
`0x0`	I2C write
`0x4`	I2C write middle of transaction (MOT)
`0x1`	I2C read
`0x5`	I2C read MOT
`0x6`	I2C write status with MOT
`0x2`	I2C write status

By using a combination of these commands, the host can emulate an I2C transaction. The flow of commanded I2C transactions is shown in Figure 33-22.

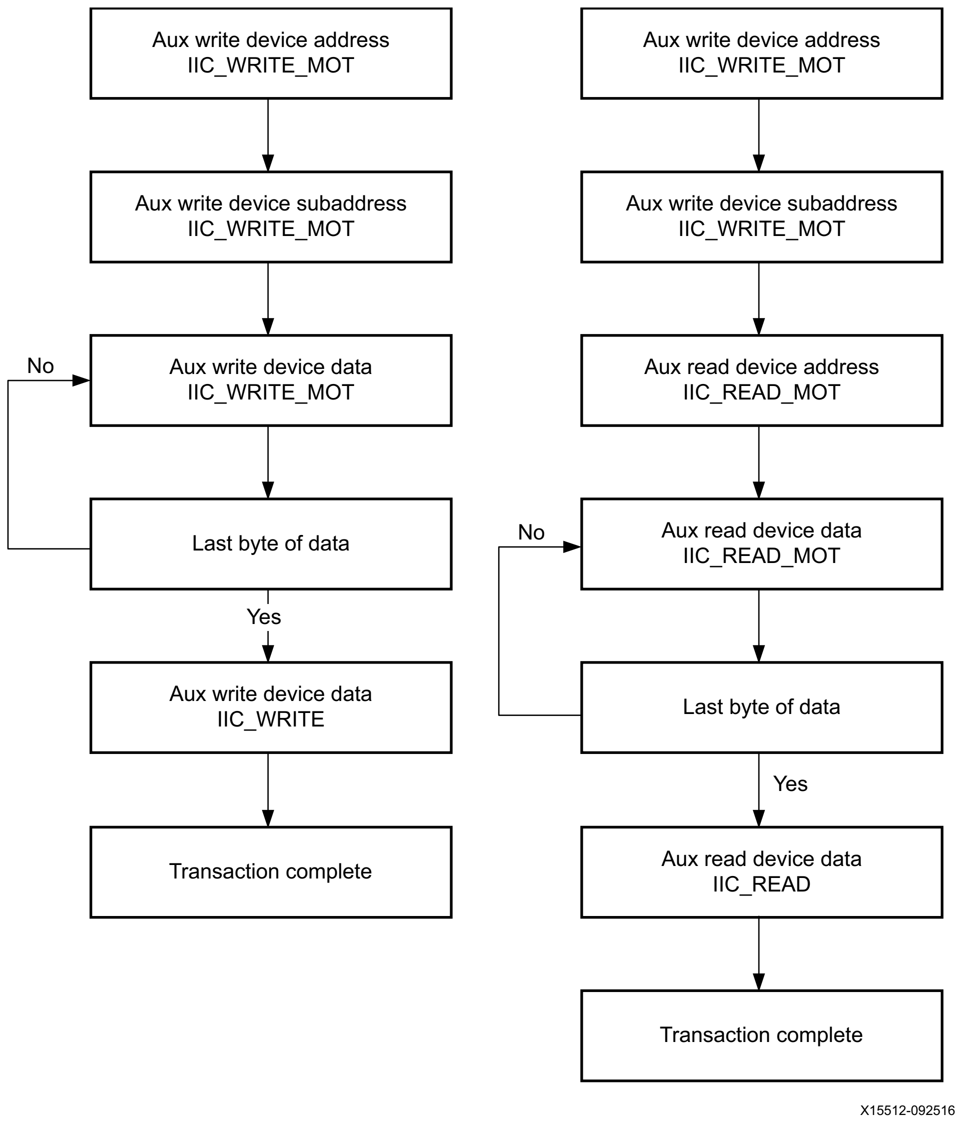


Figure 33-22: **Commanded I2C Transactions**

Because I2C transactions can be significantly slower than AUX channel transactions, the host should be prepared to receive multiple AUX_DEFER reply codes during the execution of the state machines.

The AUX-I2C commands are as follows.

- MOT definition.
    - Middle of transaction bit in the command field.
    - This controls the stop condition on the I2C slave.

- For a transaction with MOT set to 1, the I2C bus is not stopped, but left to remain in the previous state.
- For a transaction with MOT set to 0, the I2C bus is forced to idle at the end of the current command or in special abort cases.

- Partial ACK.
    - For I2C write transactions, the sink controller can respond with a partial ACK (ACK response followed by the number of bytes written to I2C slave).

Special AUX commands include the following.

- Write address only and read address only commands do not have any length field transmitted over the AUX channel. The intent of these commands are as follows.
    - Send address and RD/WR information to I2C slave. No data is transferred.
    - End previously active transaction, either normally or through an abort.

The address-only write and read commands are generated from the source by using bit [12] of the command register with command as I2C WRITE/READ.

The write status command does not have any length information. The intent of the command is to identify the number of bytes of data that have been written to an I2C slave when a partial ACK or defer response is received by the source on a AUX-I2C write. The write status command is generated from the source by using bit [12] of the command register with command as I2C WRITE STATUS.

The generation of AUX transactions is described in Table 33-21.

Table 33-21: **AUX Transactions**

Transaction	AUX Transaction	IIC Transaction	Usage	Sequence
Write address only with MOT = 1.	START -> CMD -> ADDRESS -> STOP	START -> DEVICE_ADDR -> WR -> ACK/NACK	Setup I2C slave for write to address defined.	Write AUX address register (`0x108`) with device address. Issue command to transmit transaction by writing into AUX command register (`0x100`). Bit [12] must be set to 1.
Read address only with MOT = 1.	START -> CMD -> ADDRESS -> STOP	START -> DEVICE_ADDR -> RD -> ACK/NACK	Setup I2C slave for read to address defined.	Write AUX address register with device address. Issue command to transmit transaction by writing into AUX command register. Bit [12] must be set to 1.

Table 33-21: **AUX Transactions *(Cont'd)***

Transaction	AUX Transaction	IIC Transaction	Usage	Sequence
Write/Read address only with MOT = 0.	START -> ADDRESS -> STOP	STOP	To stop the I2C slave, used as abort or normal stop.	Write AUX address register (`0x108`) with device address. Issue command to transmit transaction by writing into AUX command register (`0x100`). Bit [12] must be set to 1.
Write with MOT = 1.	START -> CMD -> ADDRESS -> LENGTH -> D0 to DN -> STOP	I2C bus is IDLE or new device address. START -> START/RS -> DEVICE_ADDR -> WR -> ACK/NACK -> DATA0 -> ACK/NACK to DATAN -> ACK/NACK I2C bus is in write state and the same device address DATA0 -> ACK/NACK to DATAN -> ACK/NACK	Setup I2C slave write data.	Write AUX address register (`0x108`) with device address. Write the data to be transmitted into AUX write FIFO register (`0x104`). Issue write command and data length to transmit transaction by writing into AUX command register (`0x100`). Bits [3:0] represent length field.

Table 33-21: **AUX Transactions *(Cont'd)***

Transaction	AUX Transaction	IIC Transaction	Usage	Sequence
Write with MOT = 0.	START -> CMD -> ADDRESS -> LENGTH -> D0 to DN -> STOP	I2C bus is IDLE or different I2C device address. START -> START/RS -> DEVICE_ADDR -> WR -> ACK/NACK -> DATA0 -> ACK/NACK to DATAN -> ACK/NACK -> STOP I2C bus is in write state and the same I2C device address. DATA0 -> ACK/NACK to DATAN -> ACK/NACK -> STOP	Setup I2C slave write data and stop the I2C bus after the current transaction.	Write the AUX address register (`0x108`) with device address. Write the data to be transmitted into AUX write FIFO register (`0x104`). Issue write command and data length to transmit transaction by writing into AUX command register (`0x100`). Bits [3:0] represent length field.
Read with MOT = 1.	START -> CMD -> ADDRESS -> LENGTH -> STOP	I2C bus is IDLE or different I2C device address. START -> START/RS -> DEVICE_ADDR -> RD -> ACK/NACK -> DATA0 -> ACK/NACK to DATAN -> ACK/NACK I2C bus is in write state and the same I2C device address. DATA0 -> ACK/NACK to DATAN -> ACK/NACK	Setup I2C slave read data.	Write AUX address register (`0x108`) with device address. Issue read command and data length to transmit transaction by writing into AUX command register (`0x100`). Bits [3:0] represent the length field.

Table 33-21: **AUX Transactions *(Cont'd)***

Transaction	AUX Transaction	IIC Transaction	Usage	Sequence
Read with MOT = 0.	START -> CMD -> ADDRESS -> LENGTH -> D0 to DN -> STOP	I2C bus is IDLE or different I2C device address. START -> START/RS -> DEVICE_ADDR -> RD -> ACK/NACK -> DATA0 -> ACK/NACK to DATAN -> ACK/NACK -> STOP I2C bus is in write state and the same I2C device address. DATA0 -> ACK/NACK to DATAN -> ACK/NACK -> STOP	Setup I2C slave read data and stop the I2C bus after the current transaction.	Write AUX address register (`0x108`) with device address. Issue read command and data length to transmit transaction by writing into AUX command register (`0x100`). Bits [3:0] represent the length field.
Write status with MOT = 1.	START -> CMD -> ADDRESS -> STOP	No transaction	Status of previous write command that was deferred or partially ACKED.	Write AUX address register (`0x108`) with device address. Issue status update command to transmit transaction by writing into AUX command register (`0x100`) Bit [12] must be set to 1.
Write status with MOT = 0.	START -> CMD -> ADDRESS -> STOP	Forces a STOP and the end of write burst.	Status of previous write command that was deferred or partially ACKED. MOT = 0 ensures the bus returns to IDLE at the end of the burst.	Write AUX address register (`0x108`) with device address. Issue status update command to transmit transaction by writing into AUX command register (`0x100`). Bit [12] must be set to 1.

Handling I2C Read Defers/Timeout

The sink controller could issue a DEFER response for a burst read to I2C. The following are the actions that can be taken by the source controller.

- Issue the same command (previously issued read, with same device address and length) and wait for response. The sink controller on completion of the read from I2C (after multiple defers) should respond with read data.
- Abort the current read using the following.
  - Read to a different I2C slave.
  - Write command.
  - Address-only Read or write with MOT = 0.

Handling I2C Write Partial ACK

The sink could issue a partial ACK response for a burst write to I2C. The following are the actions that can be taken by the source controller.

- Use the write status command to poll the transfers happening to the I2C. On successful completion, the sink should issue an NACK response to these requests while intermediate ones will get partial ACK.
- Issue the same command (previously issued with the same device address, length and data) and wait for response. On completion of the write to I2C (after multiple NACK deferments), the sink controller should respond with an ACK.
- Abort the current write using the following.
  - Write to a different I2C slave.
  - Read command.
  - Address-only read or write with MOT = 0.

Handling I2C Write Defer/Timeout

The sink controller could issue a defer response for a burst write to I2C. The following are the actions that can be taken by the source controller.

- Use the Write status command to poll the transfers happening to the I2C. On successful completion, the sink controller should issue an ACK response to these request while intermediate ones will get a partial ACK.
- Issue the same command (previously issued with the same device address, length and data) and wait for response. The sink controller on completion of the write to I2C (after multiple NACK deferments) should respond with an ACK.

- Abort the current write using the following.
  - Write to a different I2C slave.
  - Read command.
  - Address only read or write with MOT = 0.

Setting Up a DisplayPort System

The following steps are needed for a proper operation.

- Reset the complete system for initialization.
- Program the DisplayPort transmit with video resolution and lane/link rate configuration.
- Program the video blender to select the proper input and output formats.
- Program the AV buffer manager as per stream requirements.
- Program the audio mixer to select proper volume gain and metadata information.
- Enable the DisplayPort transmitter as described in the step in previous sections.

AV Buffer Manager Sequence

The following steps are to be followed in the same order, to initialize the AV buffer manager.

For video from buffer manager.

1. Program the video/graphics mode using the DP.AV_BUF_FORMAT register. This register applies only when using non-live modes.
2. Program the video clock source (`0xB120`).
3. Depending on the mode, enable the corresponding buffers in the below sequence (registers `0xB010` to `0xB024`).
   a. Flush the buffer.
   b. Enable and program burst length.
4. Configure the AV buffer outputs (`0xB070`). This should only be done after channel enable (can be after any delay). This is because the register is gating planar and interleaved buffers for them all to start from the same VSYNC boundary. This applies to video planar buffers and does not apply to video+graphics buffer.
5. Program the scaling factors for the corresponding stream. For example, if the output from the buffer is live+graphics, program the live scaling factor (`0xB218` to `0xB220`) and the graphics scaling factor (`0xB200` to `0xB208`).
6. When changing the clock source on video/audio (register `0xB120`), issue a soft reset (`0xB124`).

For audio from the buffer manager.

1. Update the ready interval for audio.
2. Enable the audio channel buffers.

AV Buffer Manager Programming Options

1. Each channel can be programmed with an independent burst length.
2. Channel 0: must be always enabled for any video mode.
3. Channel 1 and 2 should be enabled for planar modes.
4. Channel 3 for graphics.
5. Channel 4 and 5 for audio modes.

Key Points to Note in Programming

1. Changing the burst length (BL) in real time is not supported. It is expected that, after a burst length is chosen it is kept unchanged. If you must change the BL, then the previous programming sequence must be followed. The same applies to changing video resolution and pixel format.
2. The suggested programming sequence (for DisplayPort and DPDMA together).
   a. Program the DisplayPort source controller (see Source Controller Setup and Initialization).
   b. Program the AV buffer manager.
   c. Program the DPDMA.
3. Follow this sequence for disabling the DPDMA channel.
   a. Disable DPDMA (Table 33-22).
   b. Disable channel and output stream in AV buffer manager.
   c. Disable DisplayPort source controller.
   d. This is a requirement from DPDMA. The DisplayPort controller is agnostic whether DPDMA is enabled first.

4. When blending is enabled, make sure to program the RGB_MODE bit of layer 0 and layer 1 control registers (`0xA018` and `0xA01C`). If the blender output is RGB data, this bit must be set to `1`. For YUV, this bit should be `0`. Whenever palette graphics are used, this bit must be set to `1` (even if the palette is filled with YUV).

Table 33-22: **Disable DPDMA**

Task	Register	Register Field	Register Offset	Bits	Value
Pause required DPDMA channel (CHx).	DPDMA_CHx_CNTL	PAUSE	`0x0218` (CH0)	1	`1b'1`
Wait until the DPDMA transfers over on the channel.	DPDMA_ISR	no_ostand_tran0	`0x04`	6	Read and write to clear
Clear DPDMA enable.	DPDMA_CHx_CNTL	EN	`0x218` (CH0)	0	`1b'0`
Clear pause required DPDMA channel (CHx).	DPDMA_CHx_CNTL	PAUSE	`0x0218` (CH0)	1	`1b'0`

Retrigger

If the conventional descriptor update is used to feed the frame information to the DPDMA, then the DPDMA takes one or two clock cycles to process the new frame based on when the descriptor was updated by the software.

To solve this problem, DPDMA has a feature that allows the software to redirect or retrigger the DPDMA at any frame boundary.

DPDMA has a retrigger bit per channel that allows the software to redirect one or more channels. The flow of operations is as follows.

- The software triggers the channel.
- The channel waits for the first VSYNC to fetch the descriptor from the DSCR START ADDR register.
- After the DPDMA channel is done processing the descriptor, it uses the NEXT ADDR (from the descriptor) to fetch the next descriptor.
- The software can make the DPDMA channel loop on the same descriptor by giving the NEXT ADDR the same as the current descriptor address and setting an ignore done flag in the descriptor.
- After the GPU is done rendering a new frame, the software comes and writes the start address, which points to the descriptor that holds a new frame and the software also writes the retrigger bit.

- The DPDMA channel knows where the end of frame is (the descriptor flag that indicates the current descriptor is the last descriptor of the frame.) The DPDMA uses the start address to fetch the next descriptor if the software has provided a retrigger during the current frame. If the software has not provided a retrigger, the DPDMA channel fetches the next descriptor from the NEXT ADDR specified in the current descriptor.

MIO-EMIO Signals

The DisplayPort Aux interface signals are independently routed to the MIO pins or to a set of EMIO interface signals as listed in Table 33-23.

Table 33-23: **DPAUX Interface Signals**

DPAUX Interface	Index[2]	MIO Pins			EMIO Signals	
		Option 1	Option 2	I/O	Name	I/O
DPAUX_DATA_OUT	0	27	34	O	dp_aux_data_out	O
DPAUX_HPD	1	28	35	I	dp_hot_plug_detect	I
DPAUX_DATA_OE	2	29	36	O	dp_aux_data_oe_n	O
DPAUX_DATA_IN	3	30	37	I	dp_aux_data_in	I

Notes:

1. The polarity of DPAUX_DATA_OE is inverted between MIO and EMIO.
2. The index numbers are listed in Table 28-1.

GEM Ethernet

Introduction

The gigabit Ethernet controller (GEM) implements a 10/100/1000 Mb/s Ethernet MAC that is compatible with the IEEE Standard for Ethernet (IEEE Std 802.3-2008) and capable of operating in either half or full-duplex mode in 10/100 mode and full-duplex in 1000 mode. The processing system (PS) is equipped with four gigabit Ethernet controllers. Each controller can be configured independently. Each controller uses a reduced gigabit media independent interface (RGMII) v2.0.

Access to the programmable logic (PL) is through the EMIO which provides the gigabit media independent interface (GMII). Other Ethernet communications interfaces can be created in the PL using the GMII available on the EMIO interface. GEM supports the serial gigabit media-independent interface (SGMII, 1000BASE-SX, and 1000BASE-LX) at 1000 Mb/s using the PS-GTR interface.

Registers are used to configure the features of the MAC, select different modes of operation, and enable and monitor network management statistics. The DMA controller connects to memory through the advanced eXtensible interface (AXI). It is attached to the controller's FIFO interface of the MAC to provide a scatter-gather capability for packet data storage in an embedded processing system.

Each GEM controller provides management data input/output (MDIO) interfaces for PHY management.

GEM Features

Each gigabit Ethernet MAC controller has the following features:

- IEEE Standard 802.3-2008 compatible, supporting 10/100/1000 Mb/s transfer rates.
- Full and half duplex operation.
- Several I/O options:
    - RGMII with external PHY attached to MIO pins.
    - GMII/MII interface to PL (TBI, RGMII v2.0).
    - SGMII to PS GTR transceivers (1000 Mb/s rate, only).
    - 1000BASE-SX and 1000BASE-LX to PS GTR transceivers.
- MDIO interface for physical layer management of an external PHY device.
- 64-bit AXI DMA master with scatter-gather capability.
- APB slave interface for control register access.
- Interrupt generation to signal receive and transmit completion, or errors and wake-up.
- Automatic pad and cyclic redundancy check (CRC) generation on transmitted frames.
- Automatic discard of frames received with errors.
- Programmable inter-packet gap (IPG) stretch.
- Full duplex flow control with recognition of incoming pause frames and 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 and wake-on-LAN.
- IEEE Std 802.1Q VLAN tagging with recognition of incoming VLAN and priority tagged frames.
- Ethernet loopback mode.
- IPv4 and IPv6 transmit and receive IP, TCP, and UDP checksum offload.
- Recognition of IEEE Precision Time Protocol (PTP) standard (IEEE Std 1588 rev. 2) frames.
- Statistics counter registers for RMON/MIB.
- Jumbo frames up to 10,240 bytes.
- Priority (Q1) on transmit and receive.

Ethernet Controller Block Diagram

Figure 34-1 shows a block diagram of the GEM Ethernet controller.

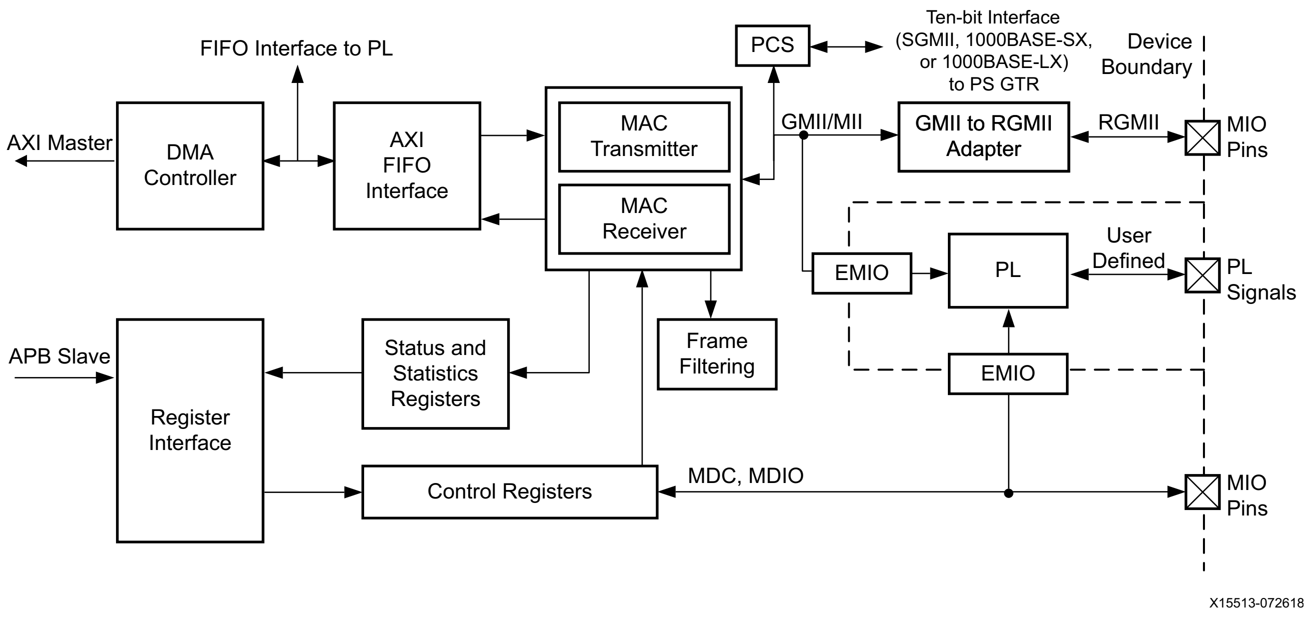


Figure 34-1: **GEM Ethernet Controller Block Diagram**

System Viewpoint

Figure 34-2 shows the GEM system viewpoint.

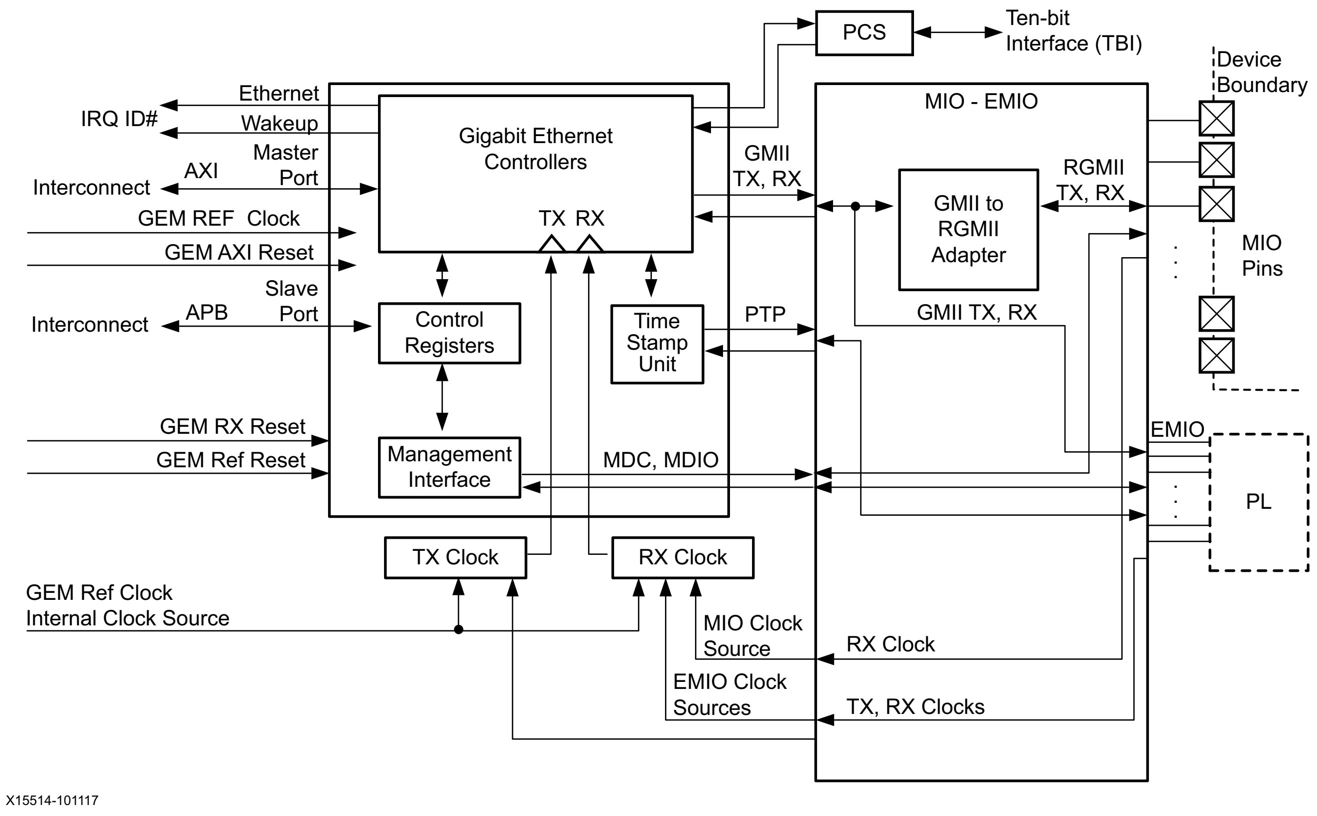


Figure 34-2: **GEM Ethernet System Viewpoint**

Clock Domains

The gigabit Ethernet controller has the following clocks.

- AXI clock: AXI clock used by DMA controller.
- APB clock: APB clock used by MAC registers.
- TSU clock: Alternate clock source for the time stamp unit (TSU).
- TX clock (tx_clk): MAC transmit clock used by MAC transmit unit in MII/RGMII/GMII/SGMII, 1000BASE-SX, or 1000BASE-LX mode.
- RX clock (rx_clk): MAC receive clock used by MAC receive synchronization in MII/RGMII/GMII/SGMII, 1000BASE-SX, or 1000BASE-LX mode.
- Invert TX clock: Inverted TX clock used in loopback mode.

- PCS transmit clock: In all modes except SGMII, 1000BASE-SX, and 1000BASE-LX, this clock can be sourced directly from tx_clk. In SGMII, 1000BASE-SX, or 1000BASE-LX applications, this clock is sourced from the serializer/deserializer and fixed at 125 MHz because the GEM PCS only operates at 1000 Mb/s.
- RBC0/RBC1 clock: Used in the PCS receive channel.

The following restrictions apply when generating a reference clock for GEM.

- Do not use fractional divisors in the PLL to generating the 125 MHz clock for the GEM module.
- Any frequency variation should be within 100 PPM.

Functional Description

The controller comprises four main components.

- MAC controlling transmit, receive, address checking, and loopback.
- Control and status registers, statistics registers, and synchronization logic.
- DMA controlling data transmit and receive through an AXI master interface.
- Time stamp unit (TSU) for calculating the IEEE Std 1588 timer values.

10/100/1000 Operation

The gigabit enable bit in the network configuration register selects between 10/100 Mb/s Ethernet operation and 1 Gb/s mode. The 10/100 Mb/s speed bit in the network configuration register is used to select between 10 Mb/s and 100 Mb/s.

SGMII, 1000BASE-SX, or 1000BASE-LX

The physical coding sublayer (PCS) can be configured to operate in SGMII, 1000BASE-SX, or 1000BASE-LX mode (1 Gb/s only). This allows the GEM to be used as a building block to support SGMII, 1000BASE-SX, or 1000BASE-LX as an interface to an external PHY, further reducing the pin count.

When bit [27] (SGMII mode) in the network configuration register (GEM{0:3}.network_config[sgmii_mode_enable]) is set, it changes the behavior of the auto-negotiation advertisement and link partner ability registers to meet the requirements of SGMII. Additionally, the time duration of the link timer is reduced from 10 ms to 1.6 ms.

Auto-negotiation is something that occurs between PHYs. SGMII is a MAC-PHY interconnect and the auto-negotiation functionality (defined in Clause 37 of IEEE Std 802.3) is used to transfer status information from the PHY to the MAC rather than to perform auto-negotiation. In SGMII mode, bits [11:10] of the link partner ability register return the data transfer rate of the link, which is previously negotiated by the PHY with its link partner PHY. The line rate is 1 Gb/s as SGMII hardwired to function at 1 Gb/s only. However, the data transfer rate can be forced down to 100 Mb/s or 10 Mb/s if the link partner is not capable. The 1000BASE-SX/LX only works at 1 Gb/s (both the data transfer rate and line rate). This information is used by configuration software to set bits [10] and [0] of the network configuration register.

The MAC transmit and receive data paths are reconfigured by the network configuration register bits [10, gigabit_mode_enable] and [11, pcs_select] for different modes and speeds of operation.

Note: When using the PS-GTR, the configuration remains the same for 1000BaseX or SGMII i.e., the external PHY will have to be configured for the required mode. However, in 1000BaseX mode, only a fixed speed of 1G can be used.

Rx and Tx FIFO Interfaces to PL

FIFO Interface to PL

Data is transmitted and received via the GEM RXFIFO and TXFIFO. There are two ways to access the FIFOs:

- GEM DMA engine as a master on the PS AXI interconnect (LPD) with a 32-bit data access width.
- Slave interface in the PL via the external FIFO interface with an 8-bit data access width.

The access to the FIFOs is selected by the GEM.external_fifo_interface [external_fifo_interface] register bit.

Interface Descriptions

Table 34-1 through Table 34-3 describe the signal names and the direction of the PL I/O for the transmit and receive FIFO interfaces and interface status.

Note: The transmit FIFO interface inputs must be pre-synchronized to the tx_clk clock domain.

Table 34-1: **Transmit FIFO Interface to PL**

Signal Name	PL Fabric	Description
tx_r_data_rdy	Input	When set to a logic 1, indicates enough data is present in the FIFO interface for Ethernet frame transmission to commence on the current packet.
tx_r_rd	Output	A single tx_clk clock-cycle wide active-High output requesting a word of information from the external FIFO interface. Synchronous to the tx_clk clock domain.
tx_r_valid	Input	A single tx_clk clock-cycle wide active-High input indicating that the requested FIFO data is now valid. Validates the following inputs: tx_r_data[7:0], tx_r_sop, tx_r_eop, and tx_r_err.

Table 34-1: **Transmit FIFO Interface to PL *(Cont'd)***

Signal Name	PL Fabric	Description
tx_r_data[7:0]	Input	FIFO data for transmission. This input is only valid when tx_r_valid is High.
tx_r_sop	Input	Start of packet, indicates the word received from the FIFO interface is the first in a packet. This input is only valid when tx_r_valid is High.
tx_r_eop	Input	End of packet, indicates the word received from the FIFO interface is the last in a packet. This input is only valid when tx_r_valid is High.
tx_r_err	Input	Error, active-High input indicating the current packet contains an error. This signal is only valid when tx_r_valid is High. It can be set at any time during the packet transfer.
tx_r_underflow	Input	FIFO underflow, indicating the transmit FIFO was empty when a read was attempted. This signal is only valid when the GEM has attempted a read by asserting tx_r_rd and the tx_r_valid signal is not yet asserted. tx_r_flushed should be asserted following this event to indicate to the GEM when it is safe to resume reading.
tx_r_flushed	Input	This signal must be driven High and then Low after a major error event to indicate to the GEM that the FIFO interface is flushed. It enables the GEM to resume reading data. Events that require this signal to be set are indicated by asserting any bit of the tx_r_status.
tx_r_control	Input	Set this input High at the start of a packet to indicate that the current frame is to be transmitted without appending a crc (tx_no_crc). This input is only valid when both tx_r_valid and tx_r_sop are High.

Table 34-2: **Transmit FIFO Interface to PL Status**

Signal Name	I/O	Description
dma_tx_end_tog	Output	Toggled to indicate that a frame is completed and status is now valid on the tx_r_status and tx_r_timestamp outputs that can be read by a PL system element. This signal is not activated when a frame is being retried due to a collision.
dma_tx_status_tog	Input	This signal must be toggled each time either dma_tx_end_tog or collision_occured are activated, to indicate that the status is acknowledged.
tx_r_status[3:0]	Output	[3]: fifo_underrun [2]: collision_occured [1]: late_coll_occured [0]: too_many_retries

Table 34-3: **Receive FIFO Interface to the PL**

Signal Name	I/O	Description
rx_w_wr	Output	A single rx_clk clock-cycle wide active-High output indicating a write to the FIFO interface.
rx_w_data[31:0]	Output	Received data for output to the FIFO interface. This output is only valid when rx_w_wr is High.
rx_w_sop	Output	Start of packet, indicates the word output to the FIFO interface is the first in a packet. This output is only valid when rx_w_wr is High.

Table 34-3: **Receive FIFO Interface to the PL *(Cont'd)***

Signal Name	I/O	Description
rx_w_eop	Output	End of packet, indicates the word output to the FIFO interface is the last in a packet. This output is only valid when rx_w_wr is High.
rx_w_status[44:0]	Output	Status signals: [44]: rx_w_code_error indicates a code error. [43]: rx_w_too_long indicates the frame is too long. [42]: rx_w_too_short indicates the frame is too short. [41]: rx_w_crc_error indicates the frame has a bad crc. [40]: rx_w_length_error indicates the length field is checked and is incorrect. [39]: rx_w_snap_match indicates the frame is SNAP encoded and has either no VLAN tag or a VLAN tag with the CFI bit not set. [38]: rx_w_checksumu indicates the UDP checksum is checked and is correct. [37]: rx_w_checksumt indicates the TCP checksum is checked and is correct. [36]: rx_w_checksumi indicates the IP checksum is checked and is correct. [35]: rx_w_type_match4: received frame is matched on type ID register 4. [34]: rx_w_type_match3: received frame is matched on type ID register 3. [33]: rx_w_type_match2: received frame is matched on type ID register 2. [32]: rx_w_type_match1: received frame is matched on type ID register 1. [31]: rx_w_add_match4: received frame is matched on specific address reg 4. [30]: rx_w_add_match3: received frame is matched on specific address reg 3. [29]: rx_w_add_match2: received frame is matched on specific address reg 2. [28]: rx_w_add_match1: received frame is matched on specific address reg 1. [27]: rx_w_ext_match4: received frame is matched by ext_match4 input signal. [26]: rx_w_ext_match3: received frame is matched by ext_match3 input signal. [25]: rx_w_ext_match2: received frame is matched by ext_match2 input signal. [24]: rx_w_ext_match1: received frame is matched by ext_match1 input signal. [23]: rx_w_uni_hash_match: received frame is matched as a unicast hash frame. [22]: rx_w_mult_hash_match: received frame matched as multicast hash frame. [21]: rx_w_broadcast_frame: received frame is a broadcast frame. [20]: rx_w_prty_tagged: VLAN priority tag detected with received packet. [19:16]: rx_w_tci[3:0]: VLAN priority of a received packet. [15]: rx_w_vlan_tagged: VLAN tag detected with a received packet. [14]: rx_w_bad_frame: a received packet is bad. [13:0]: rx_w_frame_length: number of bytes in a received packet.
rx_w_err	Output	Error, active-High output indicating the current packet contains an error.
rx_w_overflow	Input	FIFO overflow, indicates to the MAC that the RX FIFO has overflowed.
rx_w_flush	Output	FIFO flush, active-High output indicating that the RX FIFO should be cleared.

The tx_r_data_rdy signal indicates to the MAC that there is sufficient data in the FIFO to commence transmission. Once this signal becomes active, the transmit module initiates a read cycle by asserting tx_r_rd for one tx_clk cycle. The FIFO indicates valid data at the FIFO interface by asserting tx_r_valid for a single cycle. The latency between the read and valid data is controlled using the tx_r_valid response, which can be returned during the same cycle as the tx_r_rd request. Once a read commences, it must be terminated with tx_r_valid or tx_r_underflow, even if tx_r_data_rdy is deasserted.

The MAC transmitter searches for start of packet (SOP), indicated by a tx_r_sop, and transmission commences once this input becomes valid coincident with tx_r_valid. The MAC continuously searches for tx_r_sop when tx_r_data_rdy is set. Once the SOP is read, data is extracted from the FIFO on the tx_r_data input every time the tx_r_valid is set. The MAC continues to read the frame from memory using tx_r_rd and transmission takes place.

The end of frame is indicated by setting tx_r_eop coincident with tx_r_valid.

For frames smaller than the data width, both the SOP and end of packet (EOP) indicators must be set in the same data transfer. If two SOPs occur with no intervening EOP, then there is an underrun and both frames are lost, unless the second SOP occurs on the same cycle as EOP in which case the second SOP is ignored. A properly configured system should not generate two SOPs with no intervening EOP.

The tx_r_err signal can be asserted at any stage in the frame, it is driven coincident with setting tx_r_valid.

IMPORTANT: *The PL can assert the tx_r_err signal to flush the FIFOs when an error occurs. A subsequent bit is set in the tx_r_status signal (assuming the error happens after the first four bytes of the frame). When this bit set, a tx_r_flushed signal must be set to indicate the FIFO is flushed and is ready to start running following the error.*

In applications where the FIFO interface is required to operate in a half-duplex system, the tx_r_status information is available to indicate where collisions, excess collisions, late collisions, and under runs have occurred. Upon each of these conditions, it is necessary to flush the FIFO and the MAC must wait for this operation to complete before commencing further frames. A falling edge on the tx_r_flushed input signal indicates when the flush is complete. If a collision occurs, it is necessary for the FIFO interface to repeat the transfer of the current frame, so that the frame can be successfully retransmitted.

TIP: *The tx_r_flushed input signal must be driven High and then Low after a major error event to indicate to the GEM that the FIFO is flushed. This process enables the GEM to resume reading data. Events that require the tx_r_flushed signal to be set are indicated by asserting any bit of the tx_r_status signal. Before queuing a new frame, wait until the error is fully resolved. Set the tx_r_data_rdy signal after the flush is complete to indicate to the GEM that there is a new frame to transmit. The tx_r_status signal expects a new SOP with VALID after the GEM responds with the first tx_r_rd bit. If the tx_r_err signal is set, avoid sending a new frame into the core until the cleaning process is completed. One caveat to this process is that if a tx_r_err bit is set within the first four bytes of the frame, the tx_r_status signal is not set and the frame is silently dropped by the GEM. To handle the lack of a tx_r_status signal, continue without cleaning up and carry on with the next frame.*

The tx_r_status information must be acknowledged by the TX packet FIFO interface by toggling the tx_r_status_tog input to the MAC each time status is taken. This causes the tx_r_status bus to be cleared until the next end-of-frame or collision occurs.

For reception, once it is determined that a frame should be written to memory, the MAC receiver writes data to the FIFO using rx_w_wr and rx_w_data. SOP is indicated by rx_w_sop and EOP uses rx_w_eop. Rx_w_sop and rx_w_eop are not asserted in the same cycle.

The rx_w_err signal is set when the MAC encounters a reception error, such as a frame too short or a CRC error. An rx_w_status bus is available to give status about the frame being received (such as the frame length, matched internally or in the I/O, broadcast, and/or multi-cast).

The rx_w_eop signal is always asserted in the same cycle as rx_w_err. The rx_w_overflow input signal can be asserted in the rx_clk domain when an the FIFO fails to receive a frame from the FIFO interface to the PL. If rx_w_overflow is asserted sometime between the SOP and EOP writes, the remainder of the packet continues to be written out, but at the end of the packet, rx_w_err is asserted together with rx_w_eop.

Additionally, if rx_w_overflow is asserted, then the receive statistics registers do not count the frame as good. The rx_w_overflow signal must be asserted no later than one cycle after the rx_w_eop signal is asserted.

rx_w_flush is asserted when the GEM receive path is disabled in the network control register. If you disable frame reception while a frame is being transferred on the FIFO interface, you will not see an rx_w_eop indication. rx_w_flush is an rx_clk timed signal that is used to clear the receive FIFO when receive is disabled.

FIFO Interface Timing Criteria

Figure 34-3 shows the detailed timing relationships for a frame on the FIFO interface (MAC transmit) with one cycle between the read request and data valid.

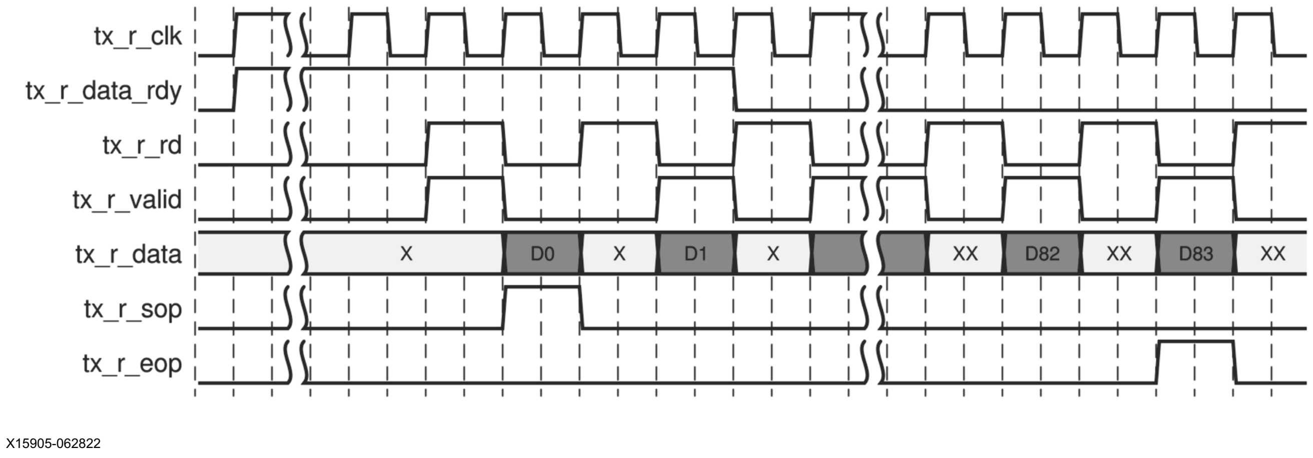


Figure 34-3: **FIFO Interface (MAC Transmit) with One Cycle**

Figure 34-4 shows the detailed timing relationships for a frame on the FIFO interface (MAC transmit) that incorporates a 2-byte frame with an SOP and an EOP in the same transfer.

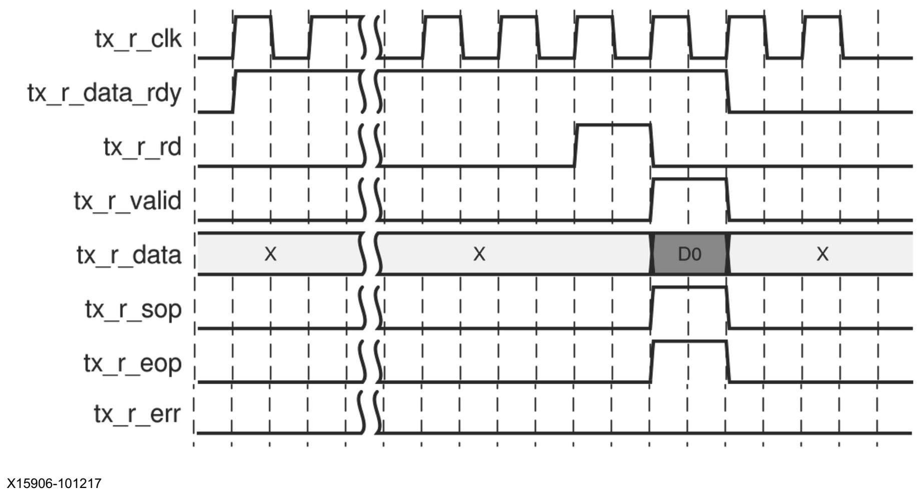


Figure 34-4: **FIFO Interface (MAC Transmit) with SOP and EOP in the Same Transfer**

Figure 34-5 shows the detailed timing relationships for a frame on the FIFO interface (MAC transmit) with a frame error.

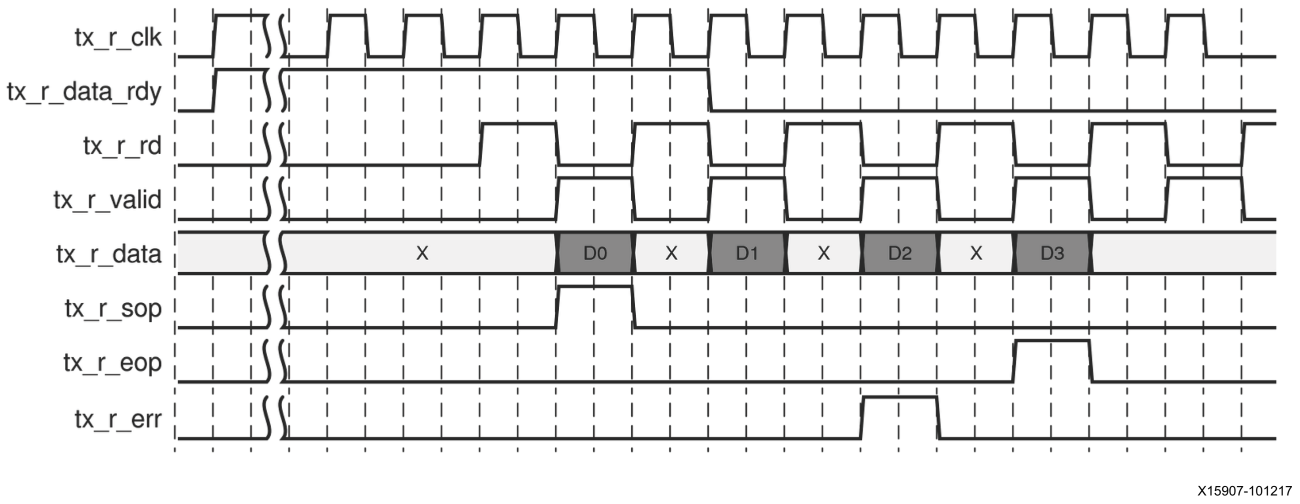


Figure 34-5: **FIFO Interface (MAC Transmit) with a Frame Error**

Figure 34-6 shows a frame on the FIFO interface (MAC receive).

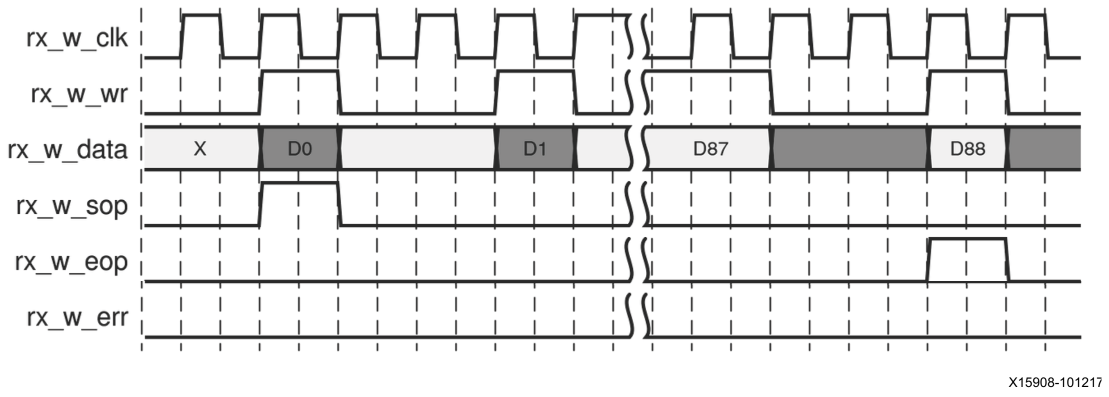


Figure 34-6: **FIFO Interface (MAC Receive)**

Figure 34-7 shows a frame on the external FIFO interface (MAC receive) with a frame error.

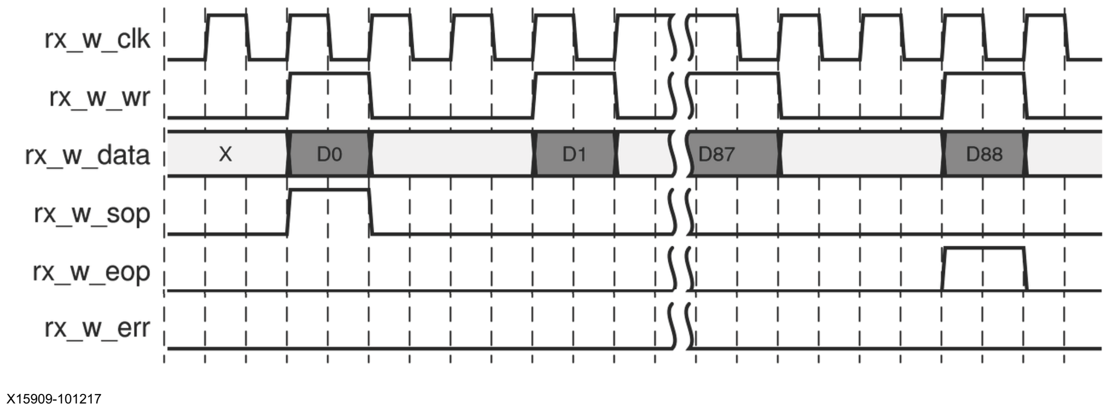


Figure 34-7: **FIFO Interface (MAC Receive) with a Frame Error**

Note: A FIFO output signal (tx_r_fixed_lat) is generated by the FIFO adapter to signal to the 8-bit FIFO interface when the latency on that internal interface is fixed. Only when the tx_r_fixed_lat signal is asserted can the latency through the design be assumed as fixed, then used to insert the correction field and/or timestamp fields into the datastream. For a 32-bit MAC datapath, the output signal is asserted later than the 22nd read on the 8-bit FIFO interface (mapping to offset 22 in the Ethernet frame). Only the assertion of this signal is of value, your design logic should detect the rising edge of this signal to determine when latency is fixed and ignore the deassertion.

MDIO Interface

The management data input/output (MDIO) interface is for physical layer management. The MDIO is a single bi-directional 3-state signal going between the GEM and one or more PHYs. The GEM signals mdio_in, mdio_out, and mdio_en are provided to control a chip-level 3-state buffer.

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 of the IEEE Std 802.3 specification is followed.

Frame assembly starts by adding the preamble and the start frame delimiter. Data is taken from the transmit FIFO a word at a time. When the controller is configured for gigabit operation, the data output to the PHY uses all eight bits of the txd[7:0] output. In 10/100 mode, transmit data to the PHY is nibble wide and the least significant nibble is first using txd[3:0] with txd[7:4] tied to logic 0.

If necessary, 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 bit is set in the second word of the last buffer descriptor of a transmit 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 interframe gap.

In half-duplex mode, the transmitter checks carrier sense. If asserted, the transmitter waits for the signal to become inactive, and then starts transmission after the interframe gap of 96-bit times. 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 transmit FIFO and a 10-bit pseudo-random number generator. The number of bits used depends on the number of collisions seen. After the first collision 1 bit is used, then the second 2 bits, and up to the maximum of 10 bits. All 10 bits are used above ten collisions. An error is indicated and no further attempts are made if 16 consecutive attempts cause a collision. This operation is compatible with the description in clause 4.2.3.2.5 of the IEEE Std 802.3 which refers to the truncated binary exponential backoff algorithm.

In 10/100 mode, both collisions and late collisions are treated identically, and backoff and retry are performed up to 16 times. When operating in gigabit mode, late collisions are treated as an exception and transmission is aborted, without 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.

MAC Receiver

All processing within the MAC receiver uses 16-bit datapaths. The MAC receiver checks for valid preamble, FCS, alignment, and length. It then sends the received frames to the FIFO (to either the DMA controller or interface to the PL) and stores the frames destination address for use by the address checking unit.

If, during frame reception, the frame is found to be too long, a bad frame indication is sent to the FIFO. The receiver logic ceases to send data to memory as soon as this condition occurs.

At end of frame reception the receive unit indicates to the DMA controller whether the frame is good or bad. The DMA controller recovers the current receive buffer if the frame is bad.

Ethernet frames are normally stored in memory via the DMA unit or to the FIFO complete with the FCS. Setting the FCS remove bit in the network configuration register (bit [17]) causes frames to be stored without their corresponding FCS. The reported frame length field is reduced by four bytes to reflect this operation.

The receive block signals to the register block to increment the alignment, CRC (FCS), short frame, long frame, jabber or receive symbol errors when any of these exception conditions occur.

If bit [26] is set in the network configuration CRC, 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 length field error frame discard bit of the network configuration register (bit [16]). When this bit is set, 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. This checking procedure is for received frames between 64 bytes and 1,518 bytes in length. 1,536 bytes if bit [8] is set in the network configuration register, 10,240 bytes if bit [3] is set in the network configuration register. Each discarded frame is counted in the 10-bit length field error statistics register.

MAC Filtering

The MAC filter determines which frames should be written to the AXI interface FIFO and onto the DMA controller. Whether a frame is passed depends on what is enabled in the network configuration register, the state of the I/O matching signals, the contents of the specific address, type, and hash registers and the frame's destination address and type field.

If bit [25] of the network configuration register is not set, a frame is not copied to memory if the gigabit Ethernet controller is 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, which is the LSB of the first byte of the frame, is the group or individual bit. This is one for multicast addresses and zero for unicast. The all-ones address is the broadcast address and a special case of multicast.

The gigabit Ethernet controller supports recognition of four specific addresses. Each specific address requires two registers, specific address register bottom and specific address register top. Specific address register bottom stores the first four bytes of the destination address and specific address register top contains the last two bytes. The addresses stored can be specific, group, local, or universal.

The destination address of received frames is compared against the data stored in the specific address registers once activated. The addresses are deactivated at reset or when their corresponding specific address register bottom is written. They are activated when the specific address register top is written. If a receive frame address matches an active address, the frame is written to the FIFO and on to the DMA controller, if used.

Frames can be filtered using the type ID field for matching. Four type ID registers exist in the register address space and each can be enabled for matching by writing a one to the MSBs (bit [31]) of the respective register. When a frame is received, the matching is implemented as an OR function of the various types of match.

The contents of each type ID registers (when enabled) 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 copied to memory 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 indicating which type ID register generated the match, if the receive checksum offload is disabled. The reset state of the type ID registers is zero, for this reason, each is initially disabled.

The following example illustrates the use of the address and type ID match registers for a MAC address of 21:43:65:87:A9:CB.

```
Preamble 55
SFD D5
DA (Octet 0 - LSB) 21
DA (Octet 1) 43
DA (Octet 2) 65
DA (Octet 3) 87
DA (Octet 4) A9
DA (Octet 5 - MSB) CB
SA (LSB) 00*
SA 00*
SA 00*
SA 00*
SA 00*
SA (MSB) 00*
Type ID (MSB) 43
Type ID (LSB) 21
Note: * - contains the address of the transmitting device.
```

The sequence shows the beginning of an Ethernet frame. Byte order of transmission is from top to bottom. For a successful match to specific address 1, the following address matching registers must be set up.

- Specific address 1 bottom (address `0x088`) `0x87654321`.
- Specific address 1 top (address `0x08C`) `0x0000CBA9`.

And for a successful match to the type ID, the following type ID match 1 register must be set up.

Type ID match 1 (address `0x0A8`) `0x80004321`.

Broadcast Address

Frames with the broadcast address of `0xFFFFFFFFFFFF` are stored to memory only if the *no broadcast* bit in the network configuration register is set to zero.

Hash Addressing

The hash address register is 64-bits long and takes up two locations in the memory map. The least significant bits are stored in hash register bottom and the most significant bits in hash register top.

The unicast hash enable and the multicast hash enable bits in the network configuration register enable the reception of hash matched frames. The destination address is reduced to a 6-bit index into the 64-bit hash register using the following hash function. The hash function is an XOR of every sixth bit of the destination address.

```
hash_index[05] = da[05]°^°da[11]°^°da[17]°^°da[23]°^°da[29]°^°da[35]°^°da[41]°^°da[47]
hash_index[04] = da[04]°^°da[10]°^°da[16]°^°da[22]°^°da[28]°^°da[34]°^°da[40]°^°da[46]
hash_index[03] = da[03]°^°da[09]°^°da[15]°^°da[21]°^°da[27]°^°da[33]°^°da[39]°^°da[45]
hash_index[02] = da[02]°^°da[08]°^°da[14]°^°da[20]°^°da[26]°^°da[32]°^°da[38]°^°da[44]
hash_index[01] = da[01]°^°da[07]°^°da[13]°^°da[19]°^°da[25]°^°da[31]°^°da[37]°^°da[43]
hash_index[00] = da[00]°^°da[06]°^°da[12]°^°da[18]°^°da[24]°^°da[30]°^°da[36]°^°da[42]
```

Where da[0] represents the least significant bit of the first byte received, that is, the multicast/unicast indicator, and da[47] represents the most significant bit of the last byte received.

If the hash index points to a bit that is set in the hash register then the frame is matched according to whether the frame is multicast or unicast.

A multicast match is signaled if the multicast hash enable bit is set, da[0] is a logic 1 and the hash index points to a bit set in the hash register. A unicast match is signaled if the unicast hash enable bit is set, da[0] is a logic 0 and the hash index points to a bit set in the hash register. To receive all multicast frames, set the hash register with all ones and set the multicast hash enable bit in the network configuration register.

Copy All Frames (or Promiscuous Mode)

If the copy all frames bit is set in the network configuration register, then all frames (except those that are too long, too short, have FCS errors, or have rx_er asserted during reception) are copied to memory. Frames with FCS errors are copied if bit [26] is set in the network configuration register.

Disable Copy of Pause Frames

Pause frames can be prevented from being written to memory by setting the disable copying of pause frames control bit [23] in the network configuration register. When set, pause frames are not copied to memory regardless of the copy all frames bit, whether a hash match is found, a type ID match is identified, or if a destination address match is found.

VLAN Support

An Ethernet encoded IEEE Std 802.1Q VLAN tag is shown in Table 34-4.

Table 34-4: **Ethernet Encoded IEEE Std 802.1Q VLAN Tag**

16-bit Tag Protocol Identifier (TPID)	16-bit Tag Control Information (TCI)
`0x8100`	First 3 bits priority, then CFI bit, last 12 bits VID

The VLAN tag is inserted at the 13th byte of the frame adding an extra four bytes to the frame. To support these extra four bytes, the gigabit Ethernet controller can accept frame lengths up to 1,536 bytes by setting bit [8] in the network configuration register.

If the VLAN identifier (VID) is null (`0x000`) a priority-tagged frame is indicated.

The following bits in the receive buffer descriptor status word 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 bit [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 receive block supports wake-on-LAN (WOL) by detecting the following events on incoming receive frames.

- Magic packets.
- Address resolution protocol (ARP) requests to the device IP address.
- Specific address 1 filter match.
- Multicast hash filter match.

If one of these events occurs, WOL detection is indicated by asserting the wake-up interrupt. These events can be individually enabled through bits[19:16] of the wake-on-LAN register.

Also, for WOL detection to occur, the receive enable must be set in the network control register.

IMPORTANT: *A receive buffer does not have to be available, but the descriptor must be fetchable from memory when the wake-up event occurs.*

The wake-up interrupt is asserted due to multicast filter events, an ARP request, or a specific address 1 match even in the presence of a frame error. 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.
- The frame's destination address matches the specific address 1 register.
- The frame is correctly formed with no errors.
- The 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.

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.
- The frame has a broadcast destination address (bytes 1 to 6).
- The frame has a type ID field of `0x0806` (bytes 13 and 14).
- The 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.

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.
- The frame's destination address matches the value programmed in the specific address 1 registers.
- A 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.
- The frame destination address matches against the multicast hash filter.
- The frame destination address is not a broadcast.

DMA Controller

The DMA controller is attached to the FIFO to provide a scatter-gather type capability for packet data storage in the embedded processing system.

Packet Buffer DMA

The DMA uses separate transmit and receive lists of buffer descriptors, with each descriptor describing a buffer area in memory. This allows Ethernet packets to be broken up and scattered around the AXI memory space.

The controller uses a packet buffer with the following advantages.

- 64 data bus width support.
- Achieve the maximum line rate by storing multiple frames in the packet buffer.
- Efficient use of the AXI interface.
- Full store and forward.
- Support for transmit TCP/IP checksum offload.
- Support for priority queuing.
- When a collision on the line occurs during transmission, the packet is automatically replayed directly from the packet buffer memory rather than having to re-fetch through the AXI interface.
- Received error packets are automatically dropped before any of the packet is presented to the AXI, reducing AXI activity.
- Supports manual RX packet flush capabilities.
- Optional RX packet flush when there is lack of AXI resources.

AXI Bus Master

Transfer size is set to 64-bit words using the AXI bus width select bits in the network configuration register, and burst size can be programmed to single access or bursts of 4, 8, 16, or 256 words using the DMA configuration register.

The DMA memory transactions will be routed to the CCI.

RX Buffers

Received frames, optionally including FCS, are written to receive AXI buffers stored in memory. The start location for each receive AXI buffer is stored in memory in a list of receive buffer descriptors at an address location pointed to by the receive-buffer queue pointer. The base address for the receive-buffer queue pointer is configured in software using the receive-buffer queue base address register.

The number of words in each buffer descriptor depends on the operating mode. Each buffer descriptor word is defined as 32 bits. The first two words (word 0 and word 1) are used for all buffer descriptor modes. In extended buffer descriptor modes (DMA configuration register bit 28 = 1), two buffer descriptor words are added for 64-bit addressing mode and two buffer descriptor words are added for timestamp capture. Therefore, there are either two, four, or six buffer descriptor words in each buffer descriptor entry depending on operating mode, and every buffer descriptor entry has the same number of words.

- Every descriptor is 64-bits wide when 64-bit addressing is disabled and the descriptor timestamp capture mode is disabled.
- Every descriptor is 128-bits wide when 64-bit addressing is enabled and the descriptor timestamp capture mode is disabled.
- Every descriptor is 128-bits wide when 64-bit addressing is disabled and the descriptor timestamp capture mode is enabled.
- Every descriptor is 192-bits wide when 64-bit addressing is enabled and the descriptor timestamp capture mode is enabled.

Table 34-5 includes details on the receive buffer descriptor list.

Each receive AXI buffer start location is a word address. The start of the first AXI 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.

Table 34-5: **RX Buffer Descriptor Entry**

Bit	Function
Word 0	
31:3	Address of beginning of buffer.
2	Address [2] of beginning of buffer, or in extended buffer descriptor mode (DMA configuration register [28] = 1), indicates a valid timestamp in the buffer descriptor entry.
1	Wrap: Marks the last descriptor in the receive buffer descriptor list.
0	Ownership: This bit must be zero for the controller to write data to the receive buffer. The controller sets this bit to 1 once the frame is written to memory. Software must clear this bit before the buffer can be used again.
Word 1	

Table 34-5: **RX Buffer Descriptor Entry *(Cont'd)***

Bit	Function
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	Specific address register match. The encoded matches are listed. `00b`: Specific address register 1 match `01b`: Specific address register 2 match `10b`: Specific address register 3 match `11b`: Specific address register 4 match If more than one specific address is matched, only one is indicated with priority 4 down to 1.
24	This bit indicates different information when the RX checksum offloading is enabled or disabled. • With RX checksum offloading 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. • With RX checksum offloading enabled, bit [24] is set in the network configuration. `0b`: The frame is not SNAP encoded and/or has a VLAN tag with the CFI bit set. `1b`: The frame is SNAP encoded and has either no VLAN tag or a VLAN tag without the CFI bit set.
23:22	These bits indicate different information when the RX checksum offloading is enabled or disabled. • With RX checksum offloading disabled, bit [24] is cleared in the network configuration type ID register match. The encoded matches are listed. `00b`: Type ID register 1 match `01b`: Type ID register 2 match `10b`: Type ID register 3 match `11b`: Type ID register 4 match If more than one type ID is matched, only one is indicated with priority 4 down to 1. • With RX checksum offloading enabled, bit [24] is set in the network configuration. `00b`: Both the IP header checksum and the TCP/UDP checksum were not checked. `01b`: The IP header checksum is checked and is correct. Both the TCP or UDP checksum were not checked. `10b`: Both the IP header and TCP checksum were checked and were correct. `11b`: 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 34-5: **RX Buffer Descriptor Entry *(Cont'd)***

Bit	Function
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	This bit 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. `0b`: Frame had good FCS. `1b`: 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 34-6 identifies the added descriptor words used when the 64-bit addressing mode is enabled.

Table 34-6: **RX Descriptor Words: 64-bit Addressing Mode**

Bit	Function
Word 2 (64-bit Addressing)	
31:0	Upper 32-bit address of the data buffer.
Word 3 (64-bit Addressing)	
31:0	Unused

Table 34-7 identifies the added descriptor words used when the descriptor timestamp capture mode is enabled.

Table 34-7: **RX Descriptor Words: Descriptor Timestamp Capture Mode**

Bit	Function
Word 2 (32-bit Addressing) or Word 4 (64-bit Addressing)	
31:30	Timestamp seconds [1:0]
29:0	Timestamp nanoseconds [29:0]
Word 3 (32-bit Addressing) or Word 5 (64-bit Addressing)	
31:4	Unused
3:0	Timestamp seconds [5:2]

The start location of the receive-buffer descriptor list must be written with the receive-buffer queue base address before reception is enabled (receive enable in the network control register). Once 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.

For low latency requirements, GEM supports receive partial store and forward in packet buffer mode. The rx watermark or cut-thru is user defined.

When the receive partial store and forward mode is activated, the receiver will only begin to forward the packet to the external AHB or AXI slave when enough packet data is stored in the packet buffer. The amount of packet data required to activate the forwarding process is programmable via watermark registers, which are located at the same address as the partial store and forward enable bits.

Enabling partial store and forward is useful to reduce latency, but there are performance implications. For example, the packet buffer DMA will start behaving in a similar way to the internal FIFO DMA mode when partial store and forward is enabled.

Note: When partial store and forward is enabled, checksum offload is not supported.

Since 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 sufficient frame data exists in the packet buffer. Therefore, several full buffers are used before error conditions can be detected. If a receive error is detected, the receive buffer currently being written will be recovered, but the previous buffers will not be recovered. For example, when receiving frames with CRC errors or excessive length, it is possible that a frame fragment may be stored in a sequence of receive buffers. Software can detect these fragments by looking for the 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, then 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 AMBA 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.

TX Buffers

Frames to transmit are stored in one or more transmit AXI buffers. Zero length AXI buffers are allowed and the maximum number of buffers permitted for each transmit frame is 128.

The number of words in each buffer descriptor depends on the operating mode. The first two words (word 0 and word 1) are used for all buffer descriptor modes. In extended buffer descriptor mode, two buffer descriptor words are added for 64-bit addressing mode and two buffer descriptor words are added for timestamp capture. Therefore, there are either two, four, or six buffer descriptor words in each buffer descriptor entry depending on operating mode, and every buffer descriptor entry has the same number of words.

- Every descriptor is 64-bits wide when 64-bit addressing is disabled and the descriptor timestamp capture mode is disabled.
- Every descriptor is 128-bits wide when 64-bit addressing is enabled and the descriptor timestamp capture mode is disabled.
- Every descriptor is 128-bits wide when 64-bit addressing is disabled and the descriptor timestamp capture mode is enabled.
- Every descriptor is 196-bits wide when 64-bit addressing is enabled and the descriptor timestamp capture mode is enabled.

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 transmit-buffer 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. Therefore, 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.

Table 34-8 includes details of the transmit buffer descriptor list.

Table 34-8: **TX Buffer Descriptor Entry**

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. Once 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.
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.

Table 34-8: **TX Buffer Descriptor Entry *(Cont'd)***

Bit	Function
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 34-9 identifies the added descriptor words used when the 64-bit addressing mode is enabled.

Table 34-9: **TX Descriptor Words: 64-bit Addressing Mode**

Bit	Function
Word 2 (64-bit Addressing)	
31:0	Upper 32-bit address of the data buffer.
Word 3 (64-bit Addressing)	
31:0	Unused

Table 34-10 identifies the added descriptor words used when the descriptor timestamp capture mode is enabled.

Table 34-10: **TX Descriptor Words: Descriptor Timestamp Capture Mode**

Bit	Function
Word 2 (32-bit Addressing) or Word 4 (64-bit Addressing)	
31:30	Timestamp seconds [1:0]
29:0	Timestamp nanoseconds [29:0]
Word 3 (32-bit Addressing) or Word 5 (64-bit Addressing)	
31:4	Unused
3:0	Timestamp seconds [5:2]

DMA Bursting on the AXI

The AXI DMA will always use 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 16 are used as appropriate.

DMA Packet Buffer

The DMA uses packet buffers for both transmit and receive paths. This mode allows multiple packets to be buffered in both transmit and receive directions. This allows the DMA to withstand far greater access latencies on the AXI and make more efficient use of the AXI bandwidth.

Full packets are buffered, which allows the following.

- Discard packets with error on the receive path before they are partially written out of the DMA. This saves AXI bus bandwidth and driver processing overhead.
- Retry collided transmit frames from the buffer. This saves AXI bus bandwidth.
- Process the transmit IP/TCP/UDP checksum generation offload.

With the packet buffers included, the structure of the controller datapaths is as shown in Figure 34-8.

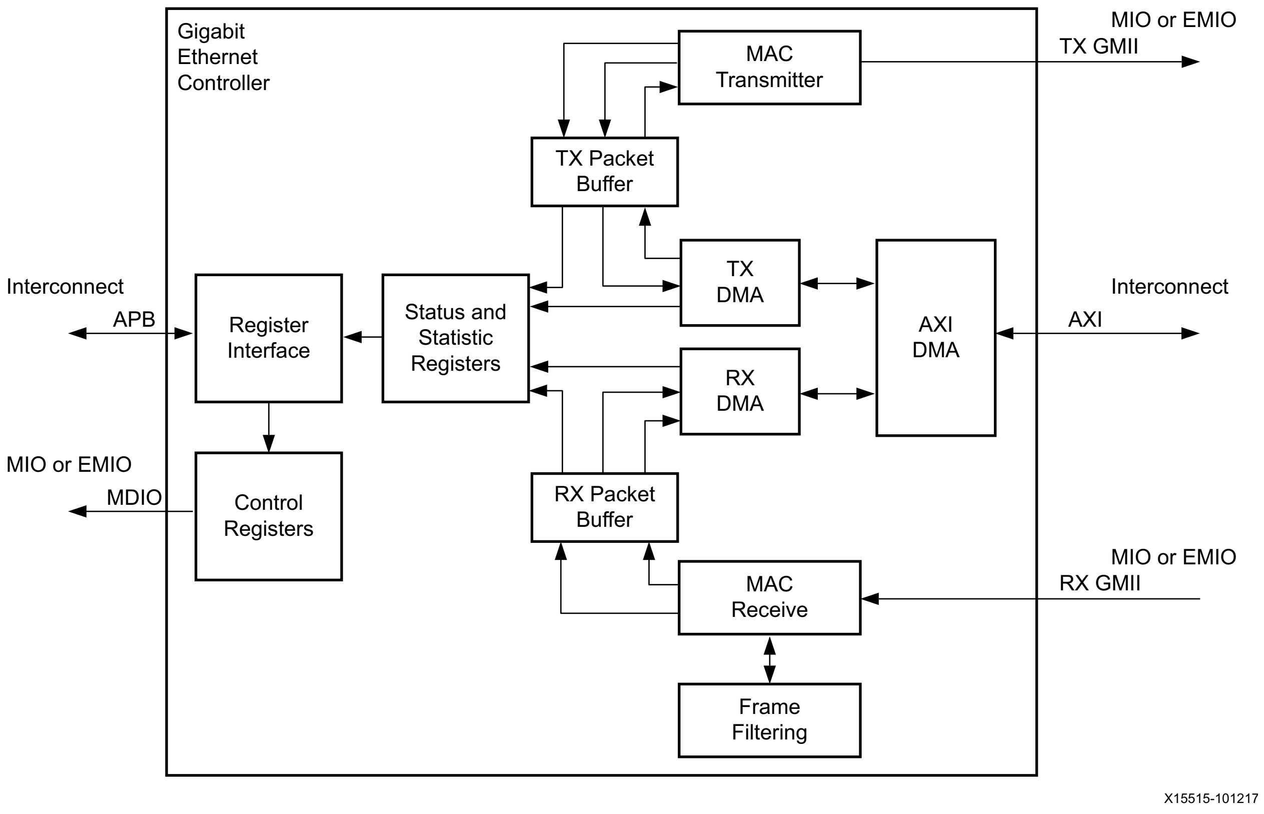


Figure 34-8: **DMA Packet Buffer**

In the transmit direction, the DMA continues to fetch packet data up to a limit of 2048 packets, or until the buffer is full. The size of the buffer has a maximum usable size of 32 KB.

In the receive direction, if the buffer becomes full, then an overflow occurs. An overflow also occurs if the limit of 2048 packets is breached. The size of the packet buffer has a maximum usable size of 32 KB.

TX Packet Buffer

The transmitter packet buffer continues to attempt to fetch frame data from the AXI system memory until the packet buffer itself is full, it then attempts to maintain the full level.

To accommodate the status and statistics associated with each frame, three words per packet are reserved at the end of the packet data. If the packet was bad and it should be dropped, the status and statistics are the only information held on that packet. Storing the status in the DPRAM is required to decouple the DMA interface of the buffer from the MAC interface, to update the MAC status/statistics, and to generate interrupts that are in the order that the packets they represent were fetched from the AXI memory.

If any errors occur on the AXI while reading the transmit frame, the fetching of packet data from AXI 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 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 AXI when space is available in the packet buffer memory. If space is not available, it 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.

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 re-transmitted (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.

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.

RX Packet Buffer

The receive packet buffer stores frames from the MAC receiver along with their status and statistics.

Frames with errors are flushed from the packet buffer memory, good frames are pushed onto the DMA AXI interface.

The receiver packet buffer monitors the FIFO writes from the MAC receiver and translates the FIFO pushes into packet buffer writes. At the end of the received frame, the status and statistics are buffered to use the information when the frame is read out. 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 still used to update the controller's 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 are the only information held on that packet.

The receiver packet buffer can detect a full condition and an overflow condition can also be detected. If this occurs, subsequent packets are dropped and an RX overflow interrupt is raised.

The DMA only begins packet fetches when the status and statistics for a frame are available. If the frame has a bad status due to a frame error, the status and statistics are passed onto the controller's registers. If the frame has a good status, the information is used to read the frame from the packet buffer memory and burst onto the AXI using the DMA buffer management protocol. After the last frame data is transferred to the FIFO, the status and statistics are updated to the controller's registers.

Checksum Offloading

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.

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 DPRAM 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.

IEEE Std 1588 Time Stamp Unit

IEEE Std 1588 is a standard for precision time synchronization in local area networks. It 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.

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 GEM 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.

For example, if the slave was assuming zero delay the actual delay is half the difference between the transmit and receive time of the delay request frame (assuming equal transmit and receive times) because the slave clock is lagging the master clock by the delay time already.

For GEM controller 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.

IEEE Std 1588 version 2 defines two additional PTP event 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.

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.

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 through the APB 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.

The MAC also provides an option to timestamp all received packets by replacing the packet's FCS word with the nanoseconds portion of the timestamp. This eliminates the need to respond to received timestamp interrupts and to associate the timestamps with the correct received packets.

MAC 802.3 Pause Frame

TIP: *See Clause 31, and Annex 31A and 31B of the IEEE Std 802.3 for a full description of pause operation.*

The start of an IEEE Std 802.3 pause frame is shown in Table 34-11.

Table 34-11: **Pause Frame Information**

Destination Address	Source Address	Type (MAC Control Frame)	Pause Opcode	Pause Time
`0x0180C2000001`	6 bytes	`0x8808`	`0x0001`	2 bytes

The controller supports both hardware controlled pause of the transmitter upon reception of a pause frame and hardware generated pause frame transmission.

IEEE Std 802.3 Pause Frame Reception

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 transmission pause, 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 `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.

IEEE Std 802.3 Pause Frame Transmission

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.

MAC PFC Priority-based Pause Frame Support

TIP: *Refer to 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 is shown in Table 34-12.

Table 34-12: **PFC Priority-based Pause Frame Information**

Destination Address	Source Address	Type (MAC Control Frame)	Pause Opcode	Priority Enable Vector	Pause Times
`0x0180C2000001`	6 bytes	`0x8808`	`0x0101`	2 bytes	8 x 2 bytes

PFC 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. Once a priority-based pause frame is received and matched, then from that moment on the controller only matches on priority-based pause frames (this is an IEEE Std 802.1Qbb requirement, known as PFC negotiation). Once 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.

Note: The rx_pfc_paused[7:0] port is not connected to any status register and is unavailable for software use. Therefore, RX pause frame use is not supported.

I/O Signals

The I/O Ethernet signals connect to the MIO and EMIO interfaces as listed in Table 34-13.

MIO-EMIO Interface Routing

The I/O interface is routed to the MIO for RGMII, and to the EMIO for GMII/MII connectivity. The PL can modify the GMII/MII interface from the MAC to construct other Ethernet interfaces that connect to external devices via PL pins. The routing of the Ethernet communications signals are shown in Figure 34-9. The Ethernet communications ports are independently routed to the MIO pins (as RGMII) or to a set of EMIO interface signals (as GMII). When using the EMIO interface, both the TX and RX clocks are inputs to the PS.

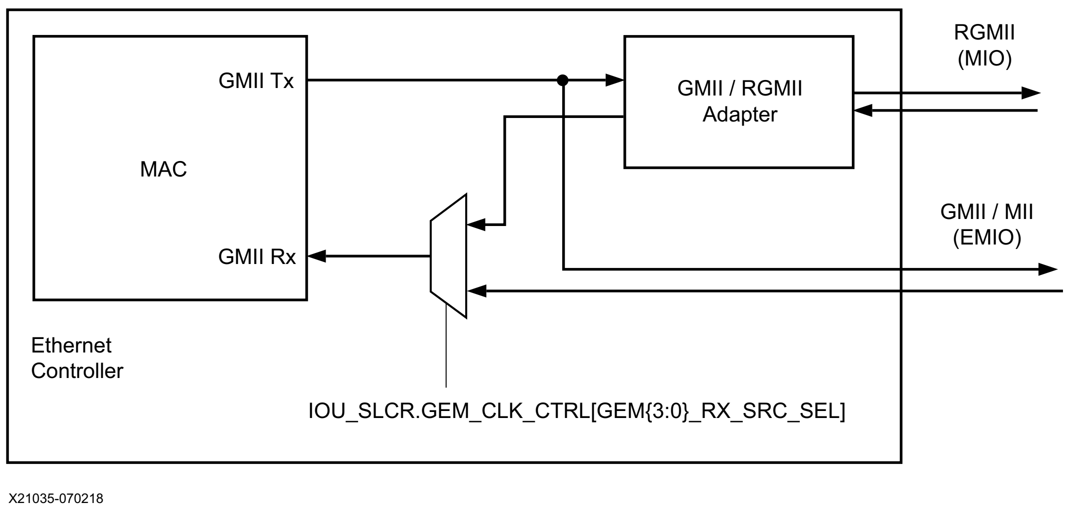


Figure 34-9: **Ethernet Interface Select Multiplexer**

RGMII Interface via MIO

An example Ethernet communications wiring connection is shown in Figure 34-10.

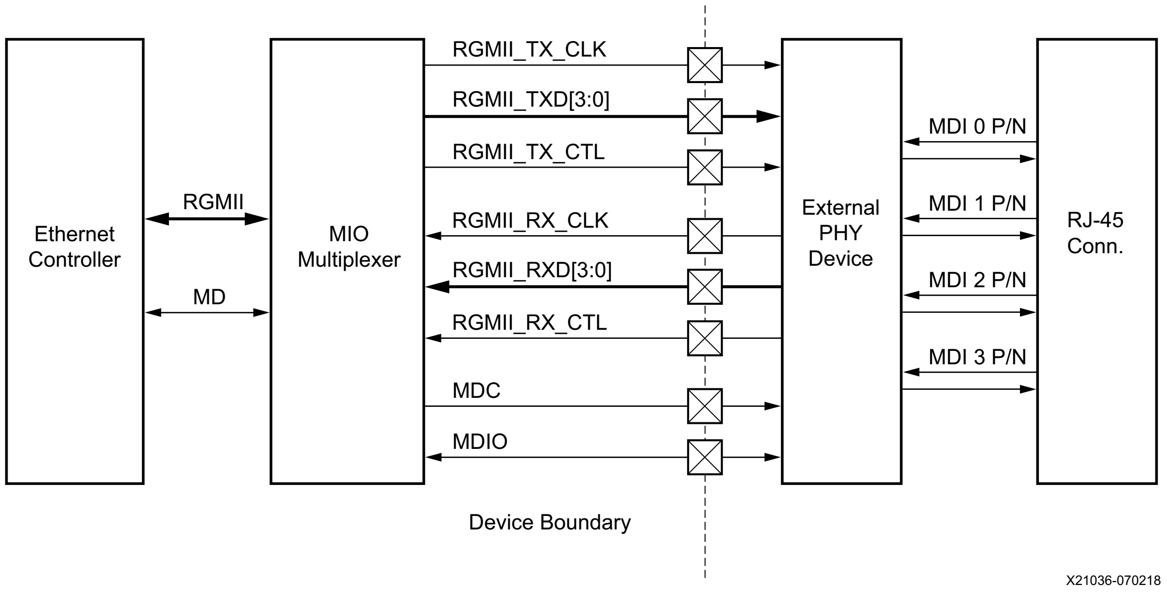


Figure 34-10: **Ethernet MIO Wiring Connections**

All Ethernet I/O pins routed through the MIO are on MIO Bank 1 and Bank 2 (see Table 34-13).

Table 34-13: **Ethernet RGMII Interface Signals via MIO Pins**

Controller Signal		**MIO Pins**					
Signal Description	**Default Controller Input Value**	**GEM 0**	**GEM 1**	**GEM 2**	**GEM 3**	**Name**	**I/O**
Tx clock to PHY	~	26	38	52	64	RGMII_TX_CLK	O
Tx control to PHY	~	31	43	57	69	RGMII_TX_CTL	O
Tx data 0 to PHY	~	27	39	53	65	RGMII_TXD[0]	O
Tx data 1 to PHY	~	28	40	54	66	RGMII_TXD[1]	O
Tx data 2 to PHY	~	29	41	55	67	RGMII_TXD[2]	O
Tx data 3 to PHY	~	30	42	56	68	RGMII_TXD[3]	O
Rx clock from PHY	0	32	44	58	70	RGMII_RX_CLK	I
Rx control from PHY	0	37	49	63	75	RGMII_RX_CTL	I
Rx data 0 from PHY	0	33	45	59	71	RGMII_RXD[0]	I
Rx data 1 from PHY	0	34	46	60	72	RGMII_RXD[1]	I

Table 34-13: **Ethernet RGMII Interface Signals via MIO Pins *(Cont'd)***

Controller Signal		MIO Pins					
Signal Description	**Default Controller Input Value**	**GEM 0**	**GEM 1**	**GEM 2**	**GEM 3**	**Name**	**I/O**
Rx data 2 from PHY	0	35	47	61	73	RGMII_RXD[2]	I
Rx data 3 from PHY	0	36	48	62	74	RGMII_RXD[3]	I
GEM TSU clock options	~	50,51	50,51	50,51	50,51	GEM_TSU_CLK	I

GMII/MII Interface via EMIO

There are options to provide further external interface standard support by linking the GMII signals on the EMIO interface to the PL. Logic can be designed and connected to generate other interface standards on the PL pins. TBI support can be provided by connecting the GMII to a TBI compatible logic core in the PL, which provides the PCS functions required for ten-bit interfacing to an external PHY via the PL pins. SGMII or 1000 Base-X support can be provided by connecting the GMII to an SGMII or 1000 Base-X compatible logic core, which provides the required PCS functions and signal adaptation and drives an MGT for serial interfacing to an external PHY.

An example illustrating the GMII interface connections through the PL to the PL pins is shown in Figure 34-11. Ethernet GMII/MII interface signals routed through the EMIO are identified in Table 34-14.

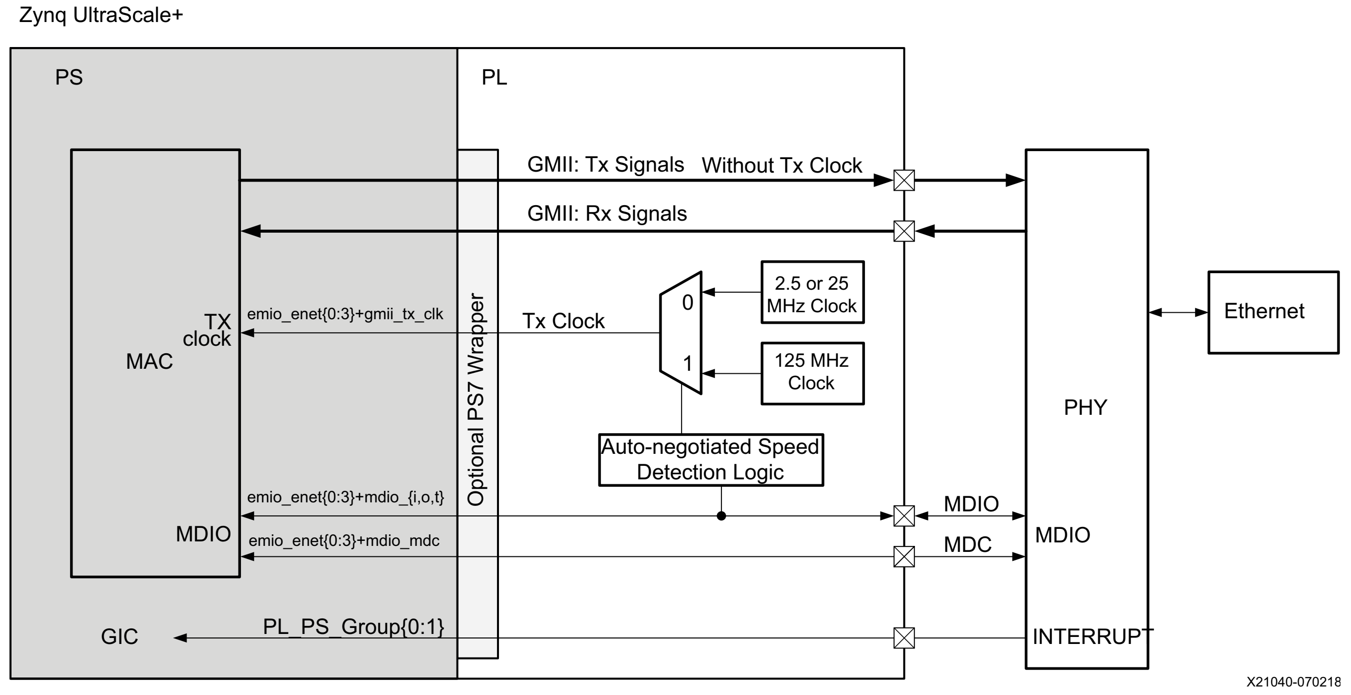


Figure 34-11: **GMII Interface via EMIO Connections**

Table 34-14: **Ethernet GMII/MII Interface Signals via EMIO Interface**

Interface Signal	Reference Clock	Default Controller Input Value	EMIO Interface Signals	
			Name	I/O
Carrier sense	~		emio_enet{0:3}_gmii_crs	I
Collision detect	~		emio_enet{0:3}_gmii_col	I
Controller interrupt wake-up	~		emio_enet{0:3}_ext_int_in	I
Speed mode (2:0)[3]	~		emio_enet{0:3}_speed_mode	O
Tx Signals				
Tx Clock	~		emio_enet{0:3}_gmii_tx_clk	I
Tx Data (7:0)	Tx Clk	~	emio_enet{0:3}_gmii_txd	O
Tx Enable	Tx Clk	~	emio_enet{0:3}_gmii_tx_en	O
Tx Error	Tx Clk	~	emio_enet{0:3}_gmii_tx_er	O
Rx Signals				
Rx Clock	~		emio_enet{0:3}_gmii_rx_clk	I
Rx Data (7:0)	Rx Clk		emio_enet{0:3}_gmii_rxd	I
Rx Data valid	Rx Clk		emio_enet{0:3}_gmii_rx_dv	I
Rx Error	Rx Clk		emio_enet{0:3}_gmii_rx_er	I
TSU				
TSU increment control(1:0)	TSU Clk		emio_enet{0:3}_tsu_inc_ctrl(1)	I
TSU clock source from PL	~		fmio_gem_tsu_clk_from_pl	I
TSU timer compare value	TSU Clk		emio_enet{0:3}_tsu_timer_cmp_val	O
TSU clock source from IP Block in the PL	~		emio_enet_tsu_clk	I

Notes:

1. The timer sync strobe registers (tsu_strobe_msb_sec, tsu_strobe_sec, and tsu_strobe_nsec) are loaded with the value of the timer when the input signal emio_enet{0:3}_tsu_inc_ctrl[1:0] = `2'b00`. However, the timer sync strobe registers are updated only when emio_enet{0:3}_tsu_inc_ctrl signal toggles between `2b'11` and `2'b00`.
2. If using MII, connect the RX[7:4] bits to logic zero.
3. See Table 34-15 for more information.

Table 34-15: **Speed Mode Bits (2:0)**

Speed Mode Bits (2:0)	Function
11x	1000 Mb/s using TBI Interface
01x	1000 Mb/s using GMII Interface
001	100 Mb/s using MII Interface
000	10 Mb/s using MII Interface
101	100 Mb/s using SGMII Interface

Precision Time Protocol via EMIO

The PTP signals connected to the Ethernet controller provide the capability to handle IEEE-1588 precision time protocol (PTP) signaling.

Table 34-16: **IEEE 1588 PTP frame recognition and Time Stamp Unit**

Signal Name	I/O	Description
sof_tx	O	Asserted high synchronous to tx_clk when the SFD is detected on a transmit frame, deasserted at end of frame.
sync_frame_tx	O	O Asserted high synchronous to tx_clk if PTP sync frame is detected on transmit.
delay_req_tx	O	Asserted high synchronous to tx_clk if PTP delay request frame is detected on transmit.
pdelay_req_tx	O	Asserted high synchronous to tx_clk if PTP peer delay request frame is detected on transmit.
pdelay_resp_tx	O	Asserted high synchronous to tx_clk if PTP peer delay response frame is detected on transmit.
sof_rx	O	Asserted high synchronous to rx_clk when the SFD is detected on a receive frame.
sync_frame_rx	O	Asserted high synchronous to rx_clk if PTP sync frame is detected on receive.
delay_req_rx	O	Asserted high synchronous to rx_clk if PTP delay request frame is detected on receive.
pdelay_req_rx	O	Asserted high synchronous to rx_clk if PTP peer delay request frame is detected on receive.
pdelay_resp_rx	O	Asserted high synchronous to rx_clk if PTP peer delay response frame is detected on receive.
tsu_clk	I	Alternative clock source for the time stamp unit. If gem_tsu_clk is defined in the gem_defs.v file then the TSU is clocked by tsu_clk rather than pclk. This clock must have a frequency greater than 1/8th the frequency of tx_clk or rx_clk. Timestamp accuracy improves with higher frequencies.

Table 34-16: **IEEE 1588 PTP frame recognition and Time Stamp Unit *(Cont'd)***

Signal Name	I/O	Description
gem_tsu_ms	I	TSU master/slave. Used with gem_tsu_inc_ctrl to control incrementing of the TSU and loading the sync strobe register.
gem_tsu_inc_ctrl[1:0]	I	Used to control incrementing of the TSU and synchronous to tsu_clk or pclk. Drive high when not being used.
tsu_timer_cnt[93:0]	O	TSU timer count value, synchronized to tsu_clk or pclk. Upper 48 bits are seconds value and lower 46 bits are nanoseconds / sub-nanoseconds. Bit 46 toggles every second, i.e. 1 pps.
tsu_timer_cmp_val	O	TSU timer comparison valid, synchronized to tsu_clk or pclk. Asserted high when upper 70 bits of TSU timer count value is equal to programmed comparison value.

There are three different TSU clock sources allowed:

- Internal PLL

  To enable clock source via internal, PLL GEM_CLK_CTRL bit[21:20] should be 0b00 and TSU_REF_CLK_CTRL bit[24] should be 1, along with appropriate clock source and divisor fields. This mode required no additional signal connections in the design and no additional configuration in Vivado block.

- Via MIO 50 or 51

  To enable clock source via internal, MIO 50/51 GEM_CLK_CTRL bit[21:20] should be 0b11 and TSU_REF_CLK_CTRL bit[24] should be 0. MIO_PIN_50/51 register should be configured for tsu. The selected MIO signal should be connected on board to an appropriate TSU clock source. The value of this should be specified in the Vivado clock configuration.

- Via EMIO

  To enable clock source via internal, PLL GEM_CLK_CTRL bit[21:20] should be 0b11 and TSU_REF_CLK_CTRL bit[24] should be 0. MIO_PIN_50/51 register should NOT be configured for tsu (so the clock is automatically picked from EMIO). Connect emio_enet0_tsu_clk to the appropriate TSU clock source on board.

Additional TSU signal configuration:

- Whenever exposed in the Vivado design, it is recommended to loop the feedback signals fmio_gem_tsu_clk_to_pl_bufg and fmio_gem_tsu_clk_from_pl.
- Whenever exposed, gem_tsu_inc_ctrl[1:0] SHOULD BE tied to 0b11 in order for GEM TSU to increment normally and function as a PTP slave.

1 PPS signal

The 1PPS signal can be obtained from the inverse of bit 45 in tsu_timer_cnt[93:0] signal. It is essential to take care of feedback signals and inc_ctrl signal. This signal can be obtained with any TSU clock source.

MDIO Interface Signals via MIO – EMIO

MDIO interface signals routed through the MIO and EMIO are identified in Table 34-17.

Table 34-17: **MDIO Interface Signals via MIO and EMIO**

MDIO Interface	Default Value	MIO Pins						EMIO Interface Signals	
		GEM 0	GEM 1	GEM 2	GEM 3	I/O	Name	Name	I/O
MD clock output	~	76	50,76	76	76	O	GEM{0:3}_MDC	enet{0:3}_mdio_mdc	O
MD data output	~	77	51,77	77	77	IO	GEM{0:3}_MDIO	enet{0:3}_mdio_o	O
MD data 3-state	~							enet{0:3}_mdio_t	O
MD data input	0							enet{0:3}_mdio_i	I

MAC Loopback

MAC local loopback can be enabled on MII/GMII by setting the GEM{0:3}.network_control[loopback_local] = 1.

In MAC internal loopback mode, both transmit and receive clock are sourced from the internal Ethernet reference clocks (see GEM Ref clock internal clock source in Figure Figure 34-2).

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.*

Also, TBI mode must be disabled for internal loopback by setting GEM{0:3}.network_config[pcs_select] = 0.

Programming Model

The controller functionality is described in detail in the Functional Description. All of the controller registers are listed in Table 34-21 and Table 34-22. Figure 34-12 summarizes the flow of programming model.

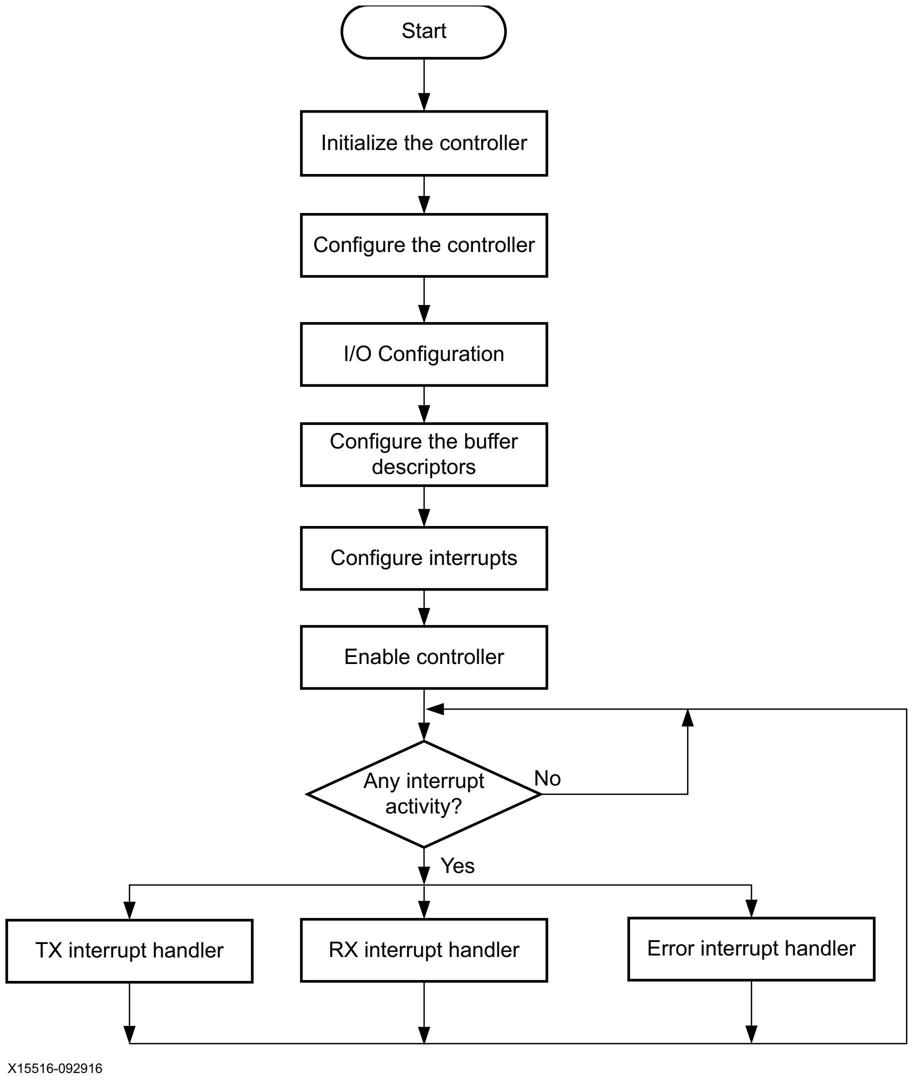


Figure 34-12: **Ethernet Controller Programming Model**

Example: Programming Steps

1. Initialize the controller
2. Configure the controller
3. I/O configuration
4. Configure the PHY
5. Configure the buffer descriptors
6. Configure interrupts
7. Enable the controller
8. Transmitting frames
9. Receiving frames
10. Debug guide

Initialize the Controller

1. Clear the network control register. Write `0x0` to the gem.network_control register.
2. Clear the statistics registers. Write a `1` to the gem.network_control [clear_all_stats_regs].
3. Clear the status registers. Write a `1` to the status registers. gem.receive_status = `0x0F` and gem.transmit_status = `0xFF`.
4. Disable all interrupts. Write `0x7FF_FEFF` to the gem. int_disable register.
5. Clear the buffer queues. Write `0x0` to the gem.receive_q{ , 1}_ptr and gem.transmit_q{ , 1}_ptr registers.

Note: The GEM controller has two receive-buffer queue pointer registers (GEM{0:3}.receive_q_ptr, GEM{0:3}.receive_q1_ptr) and two transmit-buffer queue pointer registers (GEM{0:3}.transmit_q_ptr, GEM{0:3}.transmit_q1_ptr). Any combination of transmit-buffer and receive-buffer queues can be used, but it is important to ensure that the unused queues are tied off properly with a dummy or terminate descriptor, otherwise it does not work.

Priority Queuing

The DMA is configured to use packet buffer memories. The GEM_GXL can optionally select two priority queues, q and q1. Each queue has an independent list of buffer descriptors pointing to separate data streams.

In the transmit direction, higher priority queues are always serviced before lower priority queues. This priority scheme requires the user to ensure that high priority traffic is constrained so that lower priority traffic will have the required bandwidth.

The DMA determines the next queue to service by initiating a sequence of buffer descriptor reads interrogating the ownership bits of each. The buffer descriptor corresponding to the highest priority queue is read first. If the ownership bit of this descriptor is set, then the DMA will progress to reading the second highest priority queue's descriptor.

If all the descriptors return an ownership bit set a resource error has occurred. An interrupt is generated and transmission is automatically halted. Transmission can only be restarted by setting the START bit in the network control register. The DMA will identify the highest available queue to transmit from when the START bit in the network control register is written to and the TX is in a halted state or when the last word of any packet has been fetched from external AHB or AXI memory.

The following sequence illustrates how to route receive packets to q or q1.

Note: In this case, filtering received packets based on ether type value (in this case IPv4) and routing matching packets to queue1 and rest of the packets to queue0.

Configure Rx queue pointers

1. Write Address to:

   gem0_rm.receive_q_ptr

   gem0_rm.receive_q1_ptr

2. Configure screening type2 register0 to check IPv4 ether type ID (0x0800).
3. Write ethertype_enable bit and queue number as 1 in gem. screening_type_2_register_0.
4. Write EtherType compare value for your type ID, say, 0x0800 in gem.screening_type_2_ethertype_reg_0
5. Enable receive bit in the gem.network_control_register

   Note: When a screener is matched, the received frame will be routed to a queue defined inside bits 3:0 of the screener register. Unmatched frames are routed to queue 0.

In the receive direction, each data packet is written to external AHB/AXI data buffers in the order that it is received. There are separate receive buffer queue base address registers for each queue. Every received packet will pass through a programmable screening algorithm which will allocate to that frame a queue to route it to.

The user interface to the screener is via two banks of programmable registers, screener type match registers 1 and 2. Screener type 1 registers allow the user to route received frames based on particular IP and UDP fields extracted from the received frame. These fields are DS, TC, and/or the UDP destination port. These fields are compared against the values stored in the each of the screener type 1 match registers.

If the result of this comparison is positive, then the received frame is routed to the priority queue specified in that screener type 1 register. The number of type 1 screener is determined by a define in the gem defines file. Screener Type 2 match registers operate independently and offer additional match capabilities.

Configure the Controller

The following example describes a typical programming sequence for configuration of the controller. Refer to register details for further details on the controller registers.

1. Program the network configuration register (gem.network_config). The network configuration register is used to set the mode of operation.

   Examples:

   a. Enable full duplex. Write a `1` to the gem.network_config[full_duplex] bit.

   b. Enable gigabit mode. Write a `1` to the gem.network_config[gigabit_mode_enable] bit.

   c. Enable reception of broadcast or multicast frames. Write a `0` to the gem.network_config[no_broadcast] register to enable broadcast frames and write a `1` to the gem.network_config[multicast_hash_en] bit to enable multicast frames.

   d. Enable promiscuous mode. Write a `1` to the gem.network_config[copy_all_frames] bit.

   e. Enable TCP/IP checksum offload feature on receive. Write a `1` to the gem.network_config[receive_checksum_offload_enable] bit.

   f. Enable pause frames. Write a `1` to gem.network_config[pause_enable] bit.

   g. Set the MDC clock divisor. Write the appropriate MDC clock divisor to the gem.network_config[mdc_clock_division] bit.

2. Set the MAC address. Write to the gem.spec_add1_bottom and gem.spec_add1_top registers.

   The least significant 32 bits of the MAC address go to gem.spec_add1_bottom and the most significant 16 bits go to gem.spec_add1_top.

3. Program the DMA configuration register (gem.dma_config).

   a. Set the receive buffer size to 1,600 bytes. Write a value of `8'h19` to the gem.dma_config[rx_buf_size] bit field.

      Note: For Jumbo packet support set the receive buffer size to 10,304 bytes. Write a value of `8'h0A1` to the gem.dma_config[rx_buf_size] bit field.

   b. Set the receiver packet buffer memory size to the full configured addressable space of 32 KB. Write `2'b11` to the gem.dma_config[rx_pbuf_size] bit field.

   c. Set the transmitter packet buffer memory size to the full configured addressable space of 32 KB. Write `a 1` to the gem.dma_config[tx_pbuf_size] bit.

   d. Enable TCP/IP checksum generation offload on the transmitter. Write `a 1` to the gem.dma_config[tx_pbuf_tcp_en] bit.

   e. Configure for a little endian system. Write a `0` to the gem.dma_config[endian_swap_packet] bit.

   f. Configure AXI fixed burst length. Write `5'h10` to the gem.dma_config[amba_burst_length] bit field to use INCR16 AXI burst for higher performance.

4. Program the network control register (gem.network_control).

   a. Enable the MDIO. Write a `1` to the gem.network_control[man_port_en] bit.

   b. Enable the transmitter. Write a `1` to the gem.network_control[enable_transmit] bit.

   c. Enable the receiver. Write a `1` to the gem.network_control[enable_receive] bit.

I/O Configuration

The Ethernet Controller Block Diagram describes the connection details of the Ethernet.

GEM Ethernet using MIO

The controller provides an RGMII through the MIO pins.

TIP: *The clock might have to be reprogrammed in the system level registers to provide the required reference frequency to achieve the negotiated speed.*

GEM Ethernet using EMIO

The EMIO interface allows for derivation of other physical MIIs using appropriate shim-logic in the PL. The controller provides a GMII through the EMIO.

Note: If GEM is routed via EMIO to use MII, connect the RX[7:4] bits to logic zero.

Configure Clocks

When the reference clock frequency, GEM_REF_CLK is sourced from the PS clock unit, its frequency is controlled by the CRL_APB.GEM_TSU_REF_CTRL register.

Note: The GEM_TSU_REF_CTRL register divisor fields are only applicable when the clock is set to MIO.

Configure the PHY

The PHY connected to the controller is initialized through the available MDIO interface using the PHY management register (gem.phy_management).

Writing to this register starts a shift operation and is signaled as complete when the bit gem.network_status[man_done] is set.

The MDIO interface clock (MDC) for gigabit Ethernet is generated by dividing down the LPD_LSBUS_CLK clock.

TIP: *MDC is active only during MDIO read or write operations while the PHY registers are read or written.*

The MDC must not exceed 2.5 MHz as defined by IEEE Std 802.3. The gem.network_config[mdc_clock_division] bit field is used to set the divisor for the IOU_SWITCH_CLK clock.

Example: PHY Read/Write Operation

1. Check to see that no MDIO operation is in progress. Read until gem.net_status[man_done] = `1`.
2. Write data to the PHY management register (gem.phy_management). This initiates the data shift operation over MDIO.
3. Wait for completion of operation. Read until gem.net_status[man_done] = `1`.
4. Read data bits for a read operation.

The PHY register data is available in gem.phy_management [phy_write_read_data].

Example: PHY Initialization

1. Detect the PHY address. Read the PHY identifier fields in PHY registers 2 and 3 for all the PHY addresses ranging from 1 to 32. The register contents are valid for a valid PHY address.
2. Advertise the relevant speed/duplex settings. These bits can be set to suit the system. Refer to the PHY vendor data sheet for more information.
3. Configure the PHY as applicable. This could include options to set PHY mode, timing options in the PHY, or others as applicable to the system. Refer to the PHY vendor data sheet for more information.
4. Wait for completion of auto-negotiation. Read the PHY status register. Refer to the PHY vendor data sheet for more information.
5. Update the controller with auto-negotiated speed and duplex settings. Read the relevant PHY registers to determine the negotiated speed and duplex. Set the speed in

gem.network_config[gigabit_mode_enable], gem.network_config[speed] bits, and the duplex in gem.network_config[full_duplex] bit.

Configure the Buffer Descriptors

Receive Buffer Descriptor List

The data received by the controller is written to pre-allocated buffer descriptors in system memory. These buffer descriptor entries are listed in the receive buffer queue. Refer to DMA Controller and Table 34-5 for more information on implementation and structure of the RX buffer descriptor.

The receive-buffer queue pointer registers (gem.receive_q{ , 1}_ptr) points to this data structure as shown in Figure 34-13.

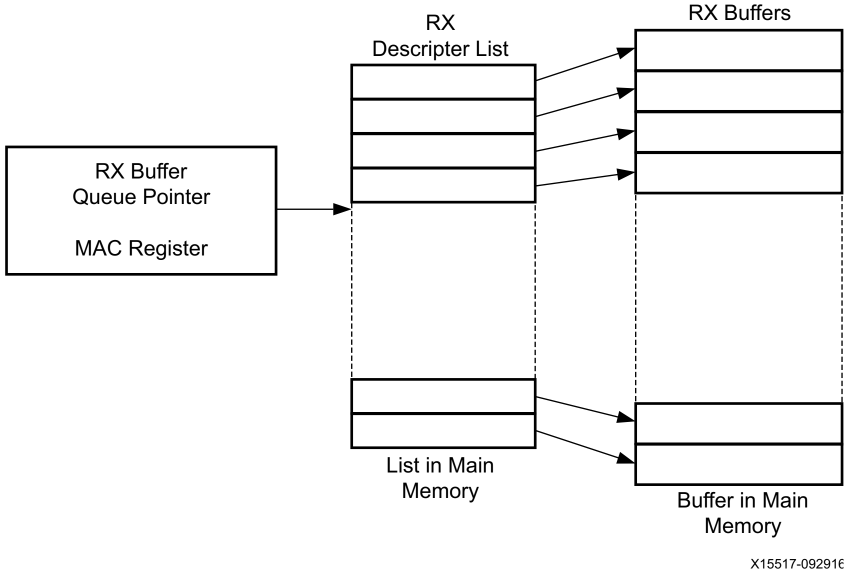


Figure 34-13: **RX Buffer Queue Structure**

To create a list of buffers:

1. Allocate a number (N) of buffers of X bytes in system memory, where X is the DMA buffer length programmed in the DMA configuration register.

   Example: This controller assumes that the maximum size of an Ethernet packet without jumbo frame support can reach up to x bytes. Allocate N number of buffers each with a size of 1,536 bytes in system memory. The buffers typically need to be aligned to cache-line boundaries to improve performance. Typical values of N can be 64 or 128.

2. Each buffer descriptor length is 8 bytes. Allocate an area of 8N bytes for the receive buffer descriptor list in system memory. This creates N entries in this list.

RECOMMENDED: *A single cache line for the APU L2 cache is 64 bytes and can contain 8 buffer descriptors. This means flushing or invalidating a single buffer descriptor entry in the cache memory results in flushing or invalidation of a cache line which in turn affects the adjacent buffer descriptors. This can result in undesirable behavior. It is typical to allocate the buffer descriptor list in an un-cached memory region.*

3. Mark all entries in this list as owned by controller. Set bit [0] of word [0] of each buffer descriptor to `0`.
4. Mark the last descriptor in the buffer descriptor list with the wrap bit, (bit [1] in word [0]) set.
5. Fill the addresses of the allocated buffers in the buffer descriptors (bits [31-2], Word [0])
6. Write the base address of this buffer descriptor list to the gem.receive_q{ , 1}_ptr registers.

Note: See the Q pointer note under Initialize the Controller.

Transmit Buffer Descriptor List

The data to be transmitted is read from buffers present in system memory. These buffers are listed in the transmit buffer queue. Refer to DMA Controller and Table 34-5 for more information on implementation and structure of the TX buffer descriptor. The transmit buffer queue pointer registers (gem.transmit_q{ , 1}_ptr) points to this data structure.

To create a list of buffer descriptors with N entries:

1. Each buffer descriptor is 8 bytes in length. Allocate an area of 8N bytes for the transmit buffer descriptor list in system memory which creates N entries in this list. It is advisable to use un-cached memory for allocating the complete buffer descriptor list for the reasons already described for the Receive Buffer Descriptor List.
2. Mark all entries in this list as owned by the controller. Set bit [31] of word [1] to `0`.
3. Mark the last descriptor in the list with the wrap bit. Set bit [30] in word [1] to `1`.
4. Write the base address of transmit buffer descriptor list to Controller registers gem.transmit_q{ , 1}_ptr.

Note: See the Q pointer note under Initialize the Controller.

Status and Wakeup Interrupts

Each GEM Ethernet unit has 26 interrupt conditions that are detected and OR-ed together to generate an IRQ system interrupt. Additionally there is a wake-on-LAN interrupt driven from the Ethernet controller. Eight IRQ system interrupts (two from each GEM unit) are then routed to the RPU, APU, and Proxy GIC interrupt controllers and outputs in the PL. Refer to the gem.int_status register description for more information on the list of interrupt conditions detected by the controller.

Example: Configure the Interrupts

An appropriate handler for the interrupt should be registered with the CPU for processing an interrupt condition. The CPU suspends its normal activity, moves to interrupt processing mode and executes the corresponding handler for an interrupt condition.

1. Register a handler. There are two interrupts generated by the controller: wake-on-LAN and another interrupt for all other functions. Register the handler for each of these interrupt types with the CPU.

   Note: In a typical case, a single handler is used for both transmit and receive.

   Once CPU execution reaches the handler, the software should read the gem.int_status register to determine the interrupt source and perform the relevant function.

2. Enable the necessary interrupt conditions. The relevant bits in the gem.int_enable register must be set. The interrupt conditions necessary are determined by the system architecture.

   Note: The read-only register gem.int_mask contains the current state of the interrupt mask each interrupt. If an interrupt bit is asserted in the status register gem.int_status and the corresponding mask bit is disabled, then the IRQ is activated.

Enable the Controller

The receiver and transmitter must be enabled after configuration.

1. Enable the transmitter. Write a `1` to gem.network_control[enable_transmit].
2. Enable the receiver. Write a `1` to gem.net_ctrl[enable_receive].

Transmitting Frames

Example: Transmitting a Frame

1. Allocate buffers in system memory to contain the Ethernet frame. Gigabit Ethernet supports scatter-gather functionality; an Ethernet frame can be split into multiple buffers with each buffer processed by a buffer descriptor.
2. Write the Ethernet frame data in the allocated buffers. These Ethernet frames should have their header fields such as destination MAC address, source MAC address, and type/length field set appropriately.
    - The FCS field is added by the MAC in most cases. However, if there is a need to append a custom FCS, bit [16] in word [1] of the corresponding buffer descriptor must be set.
    - The buffer that contains the Ethernet frame data should be flushed from cache if cached memory is being used.
3. Allocate buffer descriptor(s) for the Ethernet frame buffers. This involves setting bits [0-31] in the buffer descriptor word [0] with the address of the buffer and setting bits [0-13] in word [1] with the length of the buffer to be transmitted.
    - For single buffer Ethernet frames, bit [15] (last buffer bit) of the word [1] must also be set.
    - For Ethernet frames scattered across multiple buffers the buffer descriptors must be allocated serially and the buffer descriptor containing the last buffer should have the bit [15] of word [1] set.

      Example: For an Ethernet frame of 1,000 bytes split across two buffers with the first buffer containing the Ethernet header (14 bytes) and the next buffer containing the remaining 986 bytes, the buffer descriptor with index N should be allocated for the first buffer and the buffer descriptor with index N+1 should be allocated for the second buffer. Bit [15] of word [1] of the N+1 buffer descriptor must also be set to mark it as the last buffer in the scattered list of Ethernet frames.
4. Clear the used bit, (bit [31]), in the word [1] of the allocated buffer descriptors.

RECOMMENDED: *Clear the used bit (bit[31]) of the first buffer descriptor after clearing all the descriptors in the chain.*

5. Enable transmission. Write a `1` to gem.network_control[tx_start_pclk].
6. Wait until the transmission is complete. An interrupt is generated by the controller upon successful completion of the transmission. Successful transmission can be determined by reading the gem.int_status [transmit_complete] bit as a 1. By reading this register, the [transmit_complete] bit is cleared by the hardware. Also read and clear the gem.transmit_status register by writing a `1` to gem.transmit_status[transmit_complete] bit. Clear all bits in the buffer descriptor (BD) except the used and wrap bits.

TX Queue Sequence

1. Create two separate TX buffer description lists, one for each TX Q. The process for this setup remains the same for TX Q0 and TX Q1.
2. Program the TX queue pointer registers transmit_q_ptr and transmit_q1_ptr registers with start of respective buffer descriptor lists. The TX MSB address bits remain common for both queues.
3. Enable transmission and queue data to the desired queue via its buffer descriptor list. SW can queue a packet to either Q0 or Q1 based on the application and priority.
4. Both queues can be initialized and used. If not in use, a queue can be terminated using a dummy buffer descriptor where WRAP and USED bit are set.

Receiving Frames

When a frame is received with the receive circuits enabled, the controller checks the address and the frame is written to system memory in the following cases.

- The destination address matches one of the four specific address registers. This is applicable for cases where the MAC address for the controller is set in the gem.spec_add{1:4}_bottom and gem.spec_add{1:4}_top registers.
- The received frame's type/length field matches one of the four type ID registers. The available type ID registers are gem.spec_type{1:4}. This is applicable for cases where Ethernet type/length field based filtering is required.
- Unicast or multicast hash is enabled through gem.network_config[unicast_hash_enable] or gem.network_config[multicast_hash_enable] register bits, then the received frame is accepted, only if the hash is matched.
- The destination address is a broadcast address (`0xFFFFFFFFFFFF`) and broadcasts are allowed.

  This option is set using the gem.network_config[no_broadcast] bit.
- The controller is configured for promiscuous mode writing a 1 to the gem.network_config[copy_all_frames] bit.
- A match is found in the I/O address filtering interface.

  The register gem.receive_q{ , 1}_ptr points to the next entry in the receive buffer descriptor list and the controller uses this as the address in system memory to write the frame. When the frame is completely received and written to system memory, the controller then updates the receive buffer descriptor entry with the reason for the address match, marks the area as being owned by software, and sets the receive complete interrupt (gem.int_status[receive_complete] = 1). Software is then responsible for copying the data to the application area and releasing the buffer.

If the controller is unable to write the data at a rate to match the incoming frame, then the receiver overrun interrupt is set (gem.int_status[receive_overrun] = 1). If no receive buffer is available, (that is, the next buffer is still owned by software), a receive-buffer not available interrupt is set. If the frame is not successfully received, a statistic register is incremented and the frame is discarded without informing software.

Example: Handling a Received Frame

1. Wait for the controller to receive a frame. The receive complete interrupt, gem.int_status[receive_complete], is generated when a frame is received.
2. Service the interrupt. Read and clear the gem.int_status[receive_complete] register bit by writing a `1` to the bit in the interrupt handler. Also, read and clear the gem.receive_status register by writing a `1` to gem.receive_status[frame_received] bit.
3. Process the data in the buffer. Scan the buffer descriptor list for the buffer descriptors with the ownership bit, (bit [0], word [0]), set. When the DMA receive buffer size programmed to 1,600 bytes (gem.dma_config[rx_buf_size] = `0x19`), the packets on the receive side are not scattered and always go into a single buffer. For a buffer descriptor with the ownership bit set, process the buffer allocated in the corresponding buffer descriptor and set the ownership bit to `0`. Read other bit fields in the relevant buffer descriptor word [1], take necessary action, and clear them.

Gigabit Ethernet Debug Guide

The gigabit Ethernet can encounter different kinds of errors while receiving or transmitting Ethernet frames. Refer to *Zynq UltraScale+ MPSoC Register Reference* (UG1087) [Ref 4] register details for more information on the transmit and receive error conditions listed in the description for gem.transmit_status and gem.receive_status registers, respectively.

Some common errors and the action necessary are described in Table 34-18 and Table 34-19.

Table 34-18: **RX Status Errors**

Error Condition	Necessary Action
RESP not OK	This is a condition where it is not easy for the controller to recover. Re-initialize the controller and buffer descriptors for receive and transmit paths after clearing the relevant register status bits: gem.receive_status[resp_not_ok] and gem.int_status[resp_not_ok].
Receive overrun	This condition implies that the packet is dropped because the packet buffer is full. It occurs occasionally when the controller is unable to process the packets when they arrive very fast. In most conditions, no action for error recovery needs to be taken. Ensure that the packet buffer is configured for 32 KB (see Configure the Controller) and clear bits gem.receive_status[receive_overrun] and gem.int_status[receive_overrun].

Notes:

On RX, there is no hard requirement to have multiple buffer descriptors, although it is recommended that you minimize the chance of getting buffer resource errors (where the hardware has a frame to write to memory, but there is no free buffer(s) to write to). Extreme overflow conditions in general are more likely when these buffer resource errors occur.

Table 34-19: **TX Status Errors**

Error Condition	Necessary Action
RESP not OK	This is a condition where it is not easy for the controller to recover. Re-initialize the controller and buffer descriptors for receive and transmit paths after clearing the relevant register status bits: gem.transmit_status[resp_not_ok] and gem.int_status[resp_not_ok].
Transmit underrun	This implies a severe error condition on the transmit side in processing of the transmit buffers and buffer descriptors. For effective error recovery, the software must disable the transmitter by writing a `0` to the network_control[enable_transmit] bit, then re-initialize the buffer descriptors on the transmit side and enable the transmitter by writing a `1` to the gem.network_control[enable_transmit] bit. The bit gem.transmit_status[transmit_under_run] must be cleared in the interrupt handler.
Transmit buffer exhausted	This is a severe error condition on the transmit side. For effective error recovery, the software must disable the transmitter by writing a `0` to the network_control[enable_transmit] bit, then re-initialize the transmit buffer descriptors and transmitter. The register bits gem.transmit_status[amba_error] and gem.int_status[amba_error] must be cleared in the interrupt handler.
Retry limit exceeded	This implies there are a series of collisions for which an Ethernet frame could not be sent out even with multiple retries in half-duplex communication. Ethernet frames are dropped at the transmitter. The bits gem.transmit_status[retry_limit_exceeded] and gem.int_status[retry_limit_exceeded_or_late_collision] must be cleared in the interrupt handler. No drastic measures need to be taken for this error. However, it could also mean that there is a duplex setting mismatch.
Collisions	This error indicates that there are collisions for half duplex communication. Some collisions are expected in half-duplex mode and can be ignored. When a collision occurs, the frame is retransmitted after a while and the frame is not dropped. The register bit gem.transmit_status[collision_occurred] must be cleared in the interrupt handler.

Notes:

On TX, GigE needs multiple descriptors with the last descriptor in the BD ring having the used bit set. It is needed to ensure the GigE does not wrap and attempt to transmit the same frames more than once.

Register Overview

Clock Control Register

The clock control register drives the clocks for all GEM instances including the selection of the TSU interface and source clocks, the FIFO interface clock, SGMII, 1000BASE-SX, 1000BASE-LX, non-SGMII mode, the gem{0:3}_ref_clk (used as the PLL reference clock, the EMIO PLL clock, or the GTX clock), and the gem{0:3}_rx_clock (MIO/EMIO).

Table 34-20: **Ethernet Clock Control Register**

Function	Register Name	Description
Clock control	iou_slcr.GEM_CLK_CTRL	GEM clock control.

Control Registers

Control registers (Table 34-21) drive the management data input/output (MDIO) interface, set-up DMA activity, start frame transmission, and select the different modes of operation such as full duplex, half duplex, and 10/100/1000 Mb/s operation.

Table 34-21: **Ethernet Control Register Overview**

Function	Register Name	Description
MAC configuration	network_{config,control,status} tx_pause_quantum pause_time tx_pfc_pause stretch_ratio stacked_vlan	Network control, configuration, and status. RX and TX pause clocks. IPG stretch.
DMA unit	transmit_status receive_status transmit_q{ , 1}_ptr receive_q{ , 1}_ptr dma_config	Control. Receive and transmit status. Receive and transmit queue base address control.
Interrupts	int_{status,enable,disable, mask}	Interrupt status, enable/disable, and mask.
PHY maintenance	phy_management	PHY maintenance.

MAC address filtering and ID match	hash_{top,bottom}
	spec_add{1:4}_{bottom,top}
	mask_add1_{bottom,top}
	spec_type{1:4}
		Hash address.
		Specific {4:1} addresses High/Low.
		Match type.

IEEE Std 1588: Precision time protocol	tsu_timer_ {sec,nsec}
	tsu_timer_{adjust,incr}
	tsu_strobe_{sec,nsec}[1]
		IEEE Std 1588: second, nanosecond counter and adjustment, increment.

	tsu_ptp_tx_{sec,nsec}
	tsu_peer_tx_{sec,nsec}
		IEEE Std 1588: TX normal/peer second, nanosecond counter.

	tsu_ptp_rx_{sec,nsec}
	tsu_peer_rx_{sec,nsec}
		IEEE Std 1588: RX normal/peer second, nanosecond counter.

Note: The timer sync strobe registers (tsu_strobe_msb_sec, tsu_strobe_sec, and tsu_strobe_nsec) 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 get updated only when emio_enet{0:3}_tsu_inc_ctrl signal toggles between `11b` and `00b`.

Status and Statistics Registers

The statistics registers (Table 34-21) 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.

Table 34-22: **Ethernet Status and Statistics Register Overview**

Function	Hardware Register Name	Description
Frame TX statistics	frames_txed_ok	Error-free TX frame, pause frame counts, and bytes counts.
	broadcast_frames_tx	
	multicast_txed	
	frames_txed_64 frames_txed_65 frames_txed_128 frames_txed_256 frames_txed_512 frames_txed_1024 frames_txed_1519	64-byte frames transmitted 65 to 127-byte frames transmitted 128 to 255-byte frames transmitted 256 to 511-byte frames transmitted 512 to 1023-byte frames transmitted 1024 to 1518-byte frames transmitted Greater than 1518-byte frames transmitted
	octets_txed_{top,bottom}	Octets transmitted.
	deferred_frames	Deferred transmission frames
	pause_frames_txed	Pause and transmit under-run frames.
	tx_underruns	
Frame TX statistics for half-duplex transmission	{single,multiple}_collisions excessive_collisions late_collisions crs_errors	Single/multiple frame, excessive/late collisions, deferred TX frames, TX carrier sense error counters.

Table 34-22: **Ethernet Status and Statistics Register Overview *(Cont'd)***

Function	Hardware Register Name	Description
Frame RX Statistics	frames_rxed_ok	Error-free frames received: normal, broadcast, multicast, pause.
	broadcast_rxed	
	multicast_rxed	
	frames_rxed_64 frames_rxed_65 frames_rxed_128 frames_rxed_256 frames_rxed_512 frames_rxed_1024 frames_rxed_1519	64-byte frames received 65 to 127-byte frames received 128 to 255-byte frames received 256 to 511-byte frames received 512 to 1023-byte frames received 1024 to 1518-byte frames received 1519 to maximum byte frames received
	undersize_frames	Undersize, oversize, and jabber frames.
	excessive_rx_length	
	rx_jabbers	
	fcs_errors	Frame sequence, length, symbol, alignment error counters.
	rx_length_errors	
	octets_rxed_{top,bottom}	Octets Received
	rx_symbol_errors	RX resource, overrun and last statistic clearing offset for clearing.
	alignment_errors	
	rx_resource_errors	
	rx_overruns	
Frame RX Checksum Error Statistics	rx_ip_ck_errors rx_tcp_ck_errors rx_udp_ck_errors	Checksum error counters: IP Header, TCP, UDP

PS-PL AXI Interfaces

Introduction

The Zynq® UltraScale+™ MPSoC integrates a feature-rich quad-core or dual-core Arm® Cortex™-A53 MPCore™ based processing system (PS) and Xilinx programmable logic (PL) in a single device.

The PS and PL can be tightly or loosely coupled using multiple interfaces and other signals. This enables the designer to effectively integrate user-created hardware accelerators and other functions in the PL logic that are accessible to the processors and can also access memory resources in the PS. Using a Zynq UltraScale+ MPSoC in your design allows end-product differentiation through customized applications in the PL.

The processors in the PS always boot first, allowing a software centric approach for PL configuration. The PL can be configured as part of the boot process or configured at some point in the future. A portion of the PL can be reconfigured while other parts of the PL remain active using partial reconfiguration (PR). PR can be used to time-multiplex logic functions and algorithms, update coefficients, and reconfigure I/O. This capability is analogous to the dynamic loading and unloading of software modules. For more information, see Chapter 11, Boot and Configuration.

The PL power domain can be powered down while the PS continues to operate. In this mode, the PL consumes no static or dynamic power, thus significantly reducing the power consumption of the device. The PL must be reconfigured after power-up. You will need to account for the re-configuration time of the PL in your particular application as this varies depending on the size of the bitstream.

The PS communicates with the PL using general-purpose interconnect blocks. They support a variety of interfaces between the PL and PS and for data transfer between the PL and PS, interrupt, clock, and reset, while also connecting PS peripherals to the PL for routing to PL I/Os. Additionally, the debug cross-trigger and trace interface supports integrated HW/SW code debugging.

This chapter provides details on the PS-PL interfaces, information on use-case consideration for various interfaces, and other interface usage where appropriate.

Block Diagram and Features

The entire system-level view with both the PL and PS shown Figure 35-1. The PS-PL AXI interfaces are shown in the Programmable Logic (PL), upper right corner.

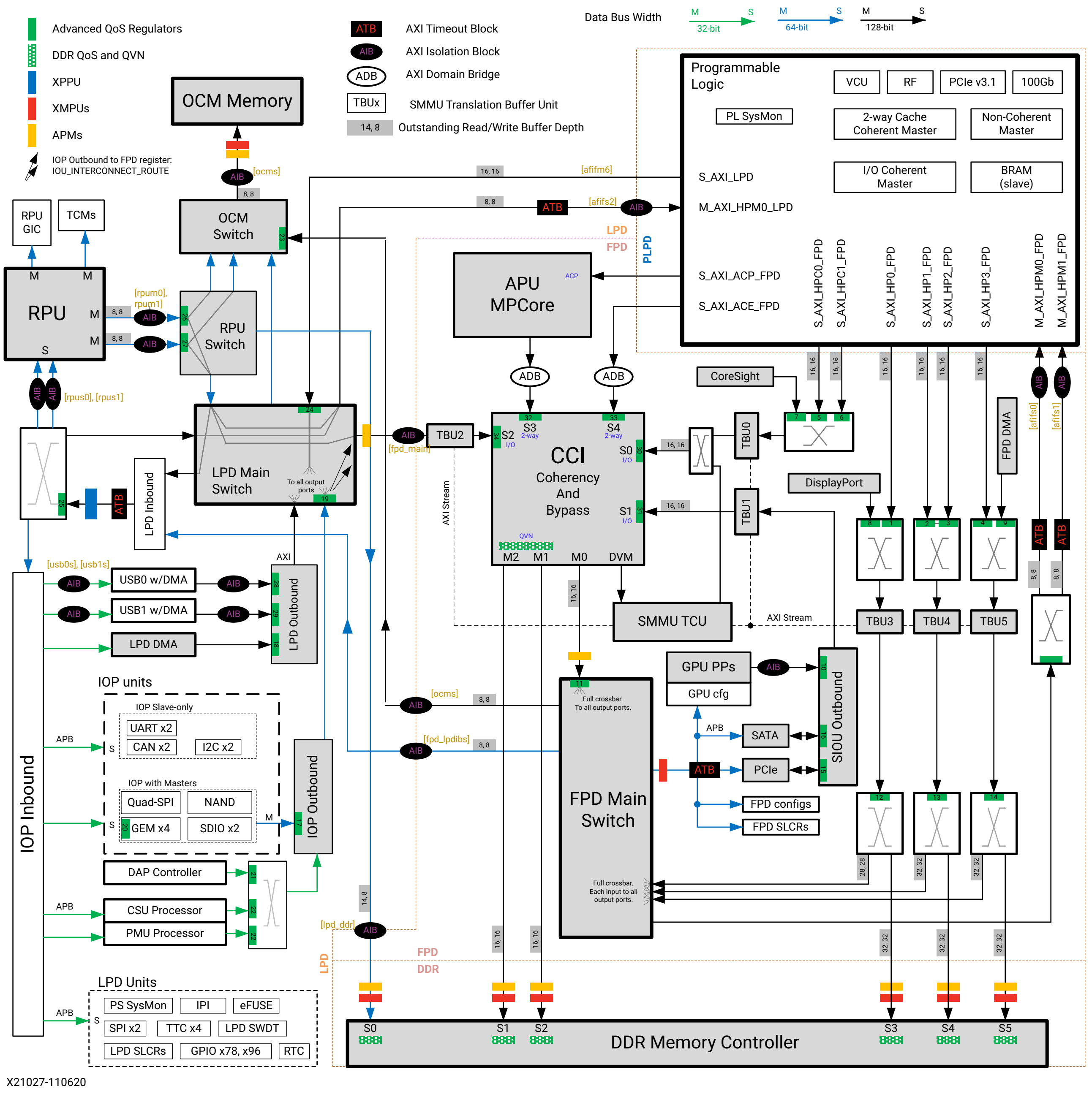


Figure 35-1: **PS-PL Interfaces**

The main features of the PS-PL interfaces are summarized in this section.

- AXI interfaces provide the following:
    - High-performance AXI4 interface with FIFO support in the PS.
        - Selectable native PL data bus width support (32/64/128).
        - Independent read and write clocks.
        - Three interfaces support I/O coherency through the cache-coherent interconnect (CCI).
    - System Memory Management Unit (SMMU) for PS bound transactions (virtual to physical address translation).
    - Dedicated low-latency path between the low-power domain (LPD) and PL.
    - Accelerator coherency port (ACP) interface for I/O coherency and allocation into the APU's L2 cache.
    - AXI coherency extensions (ACE) interface for full coherency. Usable as ACE-Lite for I/O coherency.
- 32 bits for general-purpose input and 32 bits for output from the platform management unit (PMU) for communication with the PL.
- 16 shared interrupts and four inter-processor interrupts.
- Dedicated interfaces from the gigabit Ethernet controller (GEM) and the DisplayPort protocol.
- Other PS-PL interfaces, such as extended MIO and PL clocks.

Functional Description

There are several types of PS-PL AXI interfaces and other PS-PL signals to support a heterogeneous processing system.

The Zynq UltraScale+ MPSoC provides different types of datapath ports between the PS and PL. Specific applications can use one or more of these ports.

Figure 35-2 provides a simplified top-level summary of the datapaths for the interfaces.

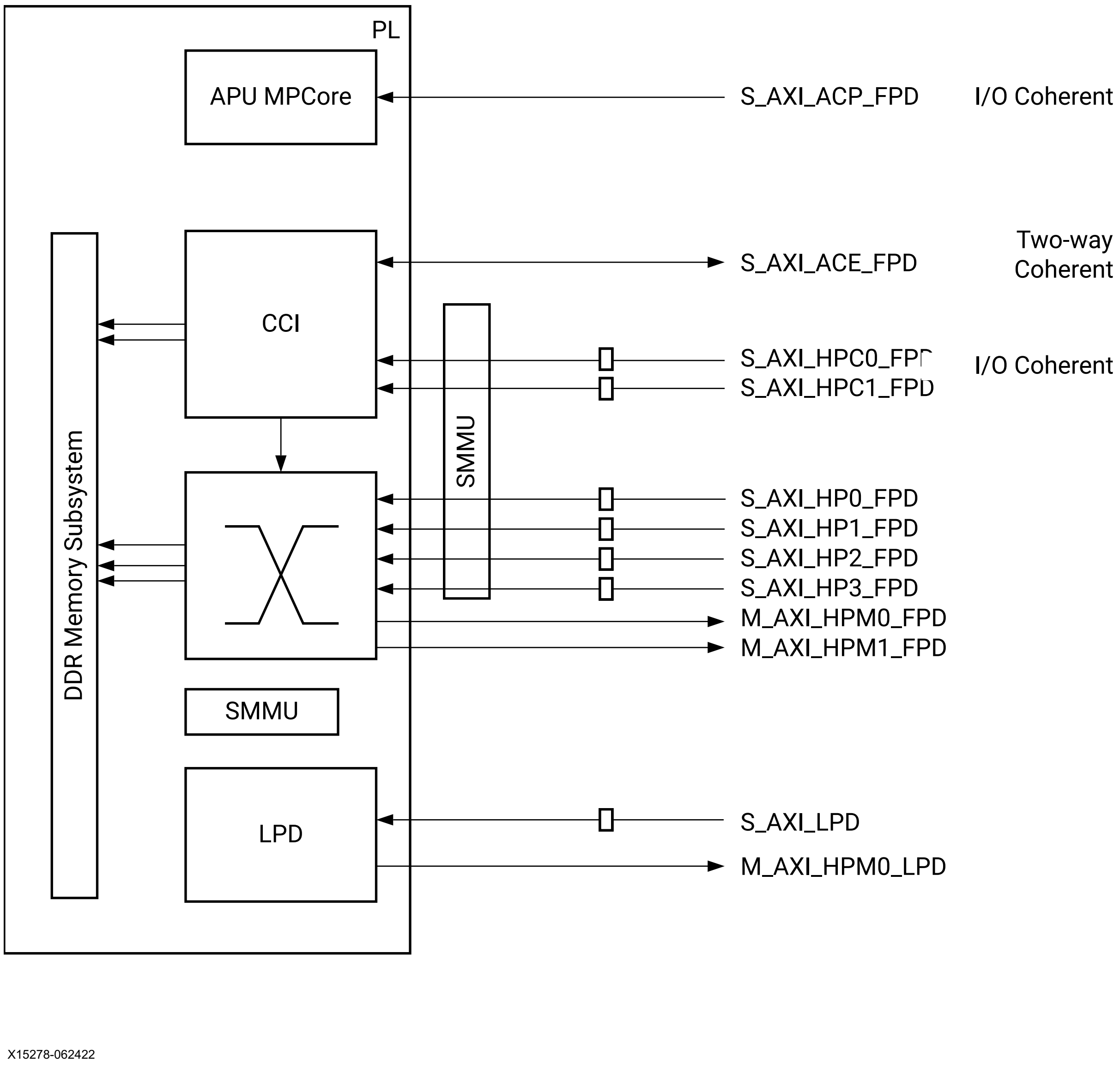


Figure 35-2: **PS-PL AXI Interface Datapaths**

The Zynq UltraScale+ MPSoC supports a maximum of 40 bits of physical address space and up to 49 bits of virtual address space.

Table 35-1 is a summary of the PS-PL AXI interfaces.

Table 35-1: **PS-PL Interface Summary**

Interface Name	Abbreviation	Data Width	Master ID Width	Address Width	Master	Slave	Description
S_AXI_ACP_FPD	ACP	128	5	40	PL	PS FPD	Accelerator coherency port: I/O coherent with CCI with L2 cache allocation.
S_AXI_ACE_FPD	ACE	128	6	40	PL	PS FPD	AXI coherency extensions: two-way coherent path between memory in PL and CCI.
S_AXI_HPC{0,1}_FPD	HPC{0,1}	128	6	49	PL	PS FPD	High-performance coherent interface passing through the CCI and SMMU providing one way (I/O) coherency (AFI_{0,1}). No L2 cache allocation.
S_AXI_HP{0:3}_FPD	HP{0:3}	32/64/128	6	49	PL	PS FPD	High-performance interface: I/O coherent with CCI and no L2 cache allocation. (AFI_{2:5}).
S_AXI_LPD	PL_LPD	32/64/128	6	49	PL	PS LPD	High-performance non-coherent path from PL to low-power domain (LPD). Access between the PL and RPU is allowed even when the full-power domain (FPD) is powered down (AFI_6).
M_AXI_HPM{0,1}_FPD	HPM{0,1}	32/64/128	16	40	PS FPD	PL	High-performance master from the FPD into the PL.
M_AXI_HPM0_LPD	LPD_PL	32/64/128	16	32	PS LPD	PL	High-performance, low-latency port from the LPD into the PL.

FPD-PL Interfaces

This section describes the PL to PS interfaces going into the full-power domain (FPD).

- Six high-performance interfaces provide the PL bus masters access to all PS slaves. However, these are designed to provide high-bandwidth datapaths to the DDR memory.
- Two high-performance masters from the FPD into the PL. Primarily these are used by high-performance PS masters like the APU, FPD DMA, and PCIe.

PL-PS Interface Specifics

The PL-PS interfaces are designed to provide a high-throughput datapath between the PL masters and PS memories, including the DDR and OCM memories. The main features of these interfaces are outlined in this section.

- Support for AXI4. The conversion to AXI3 takes place in the PS.

  Note: Even though the channels within the PS can be AXI4 (for example, SMMU TBU to DDR memory controller) the transaction burst length is restricted to a maximum of 16, due to this conversion to AXI3 in the AXI FIFO interface (AFI).
- 32, 64, or 128-bit data-wide master interfaces that are independently programmed for read and write per port.
- Efficient dynamic upsizing for all full-width AXI INCR commands.
- Asynchronous clock frequency domain crossing for all AXI interfaces between the PL and PS. Two PL clocks per interface, one for read and one for write.

TIP: *Not all HP I/O ports have the exact same path to the various system resources especially the DDR memory control AXI port interface (XPI).*

The S_AXI_HP1_FPD and S_AXI_HP2_FPD interfaces share exclusive access to an AXI Port Interface (XPI 4). This facilitates high throughput and relatively low-latency access from the PL directly to the DDR memory. S_AXI_HP0_FPD shares an XPI port on the memory controller with the DisplayPort master in the PL and S_AXI_HP3_FPD with the FPD DMA controller.

In video-based systems, S_AXI_HP0_FPD is typically used for video-type traffic and S_AXI_HP3_FPD is used for best-effort traffic.

Table 35-2: **AXI Interfaces and Associated Registers**

Interface Name	Register Name
M_AXI_HPM0_FPD	FPD_SLCR
M_AXI_HPM1_FPD	FPD_SLCR
M_AXI_HPM0_LPD	LPD_SLCR

Table 35-2: **AXI Interfaces and Associated Registers *(Cont'd)***

Interface Name	Register Name
S_AXI_HPC0_FPD	AFIFM0
S_AXI_HPC1_FPD	AFIFM1
S_AXI_HP0_FPD	AFIFM2
S_AXI_HP1_FPD	AFIFM3
S_AXI_HP2_FPD	AFIFM4
S_AXI_HP3_FPD	AFIFM5
S_AXI_LPD	AFIFM6

APU Coherent Interfaces

S_AXI_HPC0_FPD and S_AXI_HPC1_FPD can optionally support I/O coherency to the APU's L1 and L2 caches as these interfaces connect to the cache coherent interconnect (CCI). These ports can snoop APU caches through CCI provided ports. This avoids the need for software to provide coherency by flushing APU caches when APU data is shared with the I/O masters. Hardware managed I/O coherency simplifies software, improves system performance, and reduces power. Because both the S_AXI_HPC0_FPD and S_AXI_HPC1_FPD interfaces are routed through the CCI before reaching the DDR memory controller, these two ports have a longer latency to DDR.

RECOMMENDED: *Set the AxCACHE bits appropriately to enable snooping into APU caches. Drive any non-zero value on AxCACHE[3:2] for coherency; AxCACHE[3:2] = 2'b00 indicates a non-coherent transaction. Snooping should also be enabled by writing to the appropriate registers in the CCI. The Snoop_Control_Register_S3[Enable_snoops] bit should be set to generate a snoop request to the APU ACE interface.*

For further details on coherency through CCI refer to Arm's CCI TRM.

Address Translation and Protection

All high-performance interfaces into the FPD pass through the system memory management unit (SMMU).

SMMU translates the address of the incoming master requests to the physical memory address and performs checks for permissions to access that physical address, based on the information provided in the translation page-tables. Refer to SMMU Architecture in Chapter 3 for further information.

SMMU in the path of these high-performance PL interfaces provides the following support.

- Support for the use of virtual addresses (same address as is used by the software application) in the PL masters.
- Protection as SMMU performs access checks for a transaction

AXI FIFO Interface

The AXI FIFO interface (AFI) is included to provide high-throughput datapaths between the PL masters and the PS DDR Memory Controller.

- The access latency through the multi-ported DDR memory controller in the PS is expected to vary across a wide range and can vary under loaded conditions. The AXI FIFO interface helps to smooth out this variable latency, allowing the ability to *stream* data continuously between the DDR and corresponding PL master.
- This module also helps provide rate adjustment between the PL and PS clock domains.

- The PL interface is AXI4 while the PS interface is AXI3-compliant. The AFI converts between AXI4 and AXI3 formats.

The block diagram in Figure 35-3 shows the AXI FIFO interface. There are two sets of AXI ports, one set connecting directly to the PL (blue) and the other (PS) connecting to the AXI switch matrix (red), providing access to the PS DDR memory and other slaves.

TIP: *The 32, 64, and 128-bit programmable logic interfaces are programmable; the PS-side AXI interface is always 128 bits.*

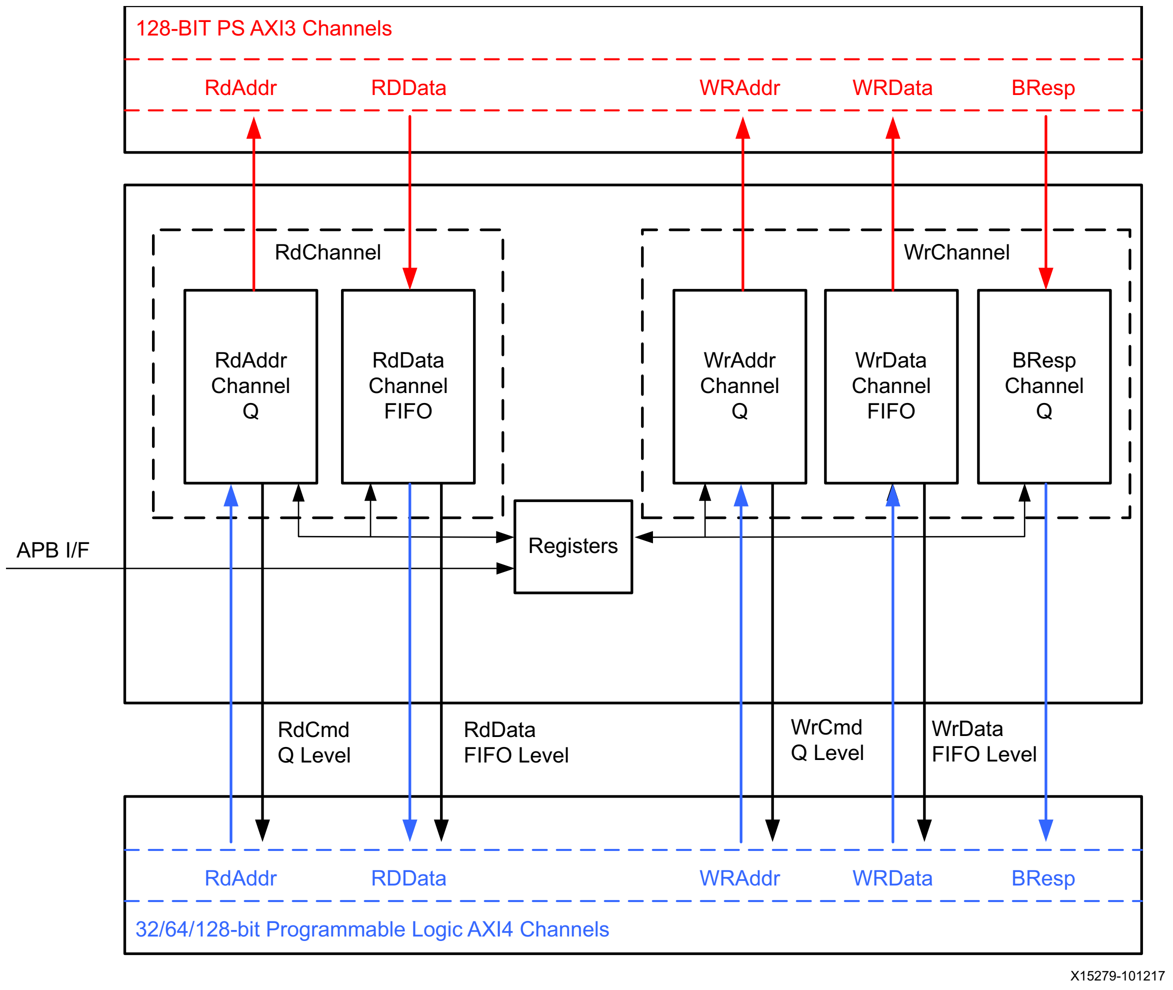


Figure 35-3: **AXI FIFO Interface for HP I/O Interface**

The level of the data FIFOs as well as the command queues for both read and write are exported to the PL, to provide visibility to programmable logic applications.

TIP: *The FIFO levels should be used as a relative level as opposed to an exact level, because clock domain crossings are involved; that is, the read and write FIFO levels indicate a pessimistic count of read/write data words when the FIFO interfaces are operating at an asynchronous clock frequency and do not represent the actual words stored in the FIFO.*

AXI Interface Programming

An advanced peripheral bus (APB) interface is provided to allow for control and monitoring of the module's functions.

The FIFO interface contains the following features.

- 8-deep write and read command queue depths.
- Read and write command acceptance capability of eight.
- The maximum number of outstanding unique IDs issued to the PS is eight per port, per channel.
- 128 x 128-bit deep read and write data FIFOs.
- Programmable release modes for write commands.
- Programmable issuing capability per port, per channel, up to a maximum of 16. This is possible only if the limit of eight outstanding IDs is not exceeded and there is space available in the data FIFO.
- Command and data FIFO fill-level exported to the programmable logic.
- The ability to write to the data FIFO without the writing the corresponding write commands.
- Upsizing for full-width, aligned, and unaligned INCR-type bursts.
- Dynamic command upsizing translation supported between 32-bit or 64-bit PL interfaces and 128-bit PS-side, controllable with the AxCACHE[1] bit.

TIP: *Upsizing occurs for full-width, INCR burst-type commands when the AxCACHE[1] bit is set. All other command-types are expanded. The process of upsizing involves modification of the AWSIZE field to 128-bit, as well as adapting the AWLEN field appropriately.*

Expansion or upsizing can be dynamically controlled, on a per-command basis, based on AxCACHE[1] bit value.

Note: The write latency (i.e., the time from when the write request is sent to the reception of the BRESP) is dependent on factors such as system load and DDR latency. The FIFO interface sends the write command/data all the way to the destination slave. The slave responds with a BVALID and the BVALID is returned to the FIFO interface and then to the PL. There is no capability for an early BRESP, that is, an early response to the PL from the AFI is not sent, but the DDR controller has the capability to send an early response to the AXI interface.

Additional Per Port HP I/O PL Signals

The additional signals provided to the PL (in addition to the standard AXI4 signals) are listed in Table 35-3. The QoS priority and FIFO occupancy management functions (read and write data and command buffer counts) are discussed in the following sections.

Table 35-3: **Additional Per Port HP I/O Signals**

Type	PS-PL Signal Name	I/O	Description
FIFO Occupancy	*_RCOUNT[7:0]	O	Fill level of read data channel FIFO.
	*_WCOUNT[7:0]	O	Fill level of write data channel FIFO.
	*_RACOUNT[3:0]	O	Fill level of read address channel FIFO.
	*_WACOUNT[3:0]	O	Fill level of write address channel FIFO.
Quality of Service	*_AWQOS[3:0]	I	Write address channel QOS input. Qualified by corresponding *_AWVALID.
	*_ARQOS[3:0]	I	Read address channel QOS input. Qualified by corresponding *_ARVALID.

QoS Priority

Quality of service (QoS) can be used to assign an arbitration priority to the read and write commands. QoS can be controlled from physical PL signals or statically configured in the AXI interface's APB registers (RDQoS or WRQoS registers in the AFIFM register set). Driving the signals allows QoS values to be changed on a per transaction basis. The register control is static for all commands.

Read and Write Data Buffers

The HP interfaces provide 128-entry deep data buffers for each read and write data channel. The HP interfaces also provide buffer level information via count ports. The read data buffer information level is used when the PL-master data consumer logic is decoupled from the read request logic. Similarly, the write data buffer level information is used when the PL-master write data producer logic streams out data before a write request is generated.

The read data buffer is used as a pre-allocation or pre-fetch buffer, where the PL can issue a large number of read requests without having its own read pre-allocation or pre-fetch buffer. The write data buffer is used to stream the write data before a write request.

Based on the relative levels of the count values provided, a PL controller can dynamically change the priority of the individual read and write requests into the high-performance AXI interface block(s). The FIFO level count should be used as a relative level, as opposed to an exact level, because clock domain crossing is involved.

Traffic Quality of Service

In general, traffic can be categorized into three traffic classes based on the quality of service (QoS) value.

- High priority (low latency).
- Isochronous (regular, time sensitive, e.g., audio and video traffic).
- Best effort (bulk transfers).

The low-latency traffic class is primarily intended for CPU (or CPU like) latency critical traffic, and is not recommended for use by transfers with an AXI interface that includes a FIFO.

On each of the FIFO-enabled AXI interfaces, a traffic shaper (QoS controller) is implemented that can be configured to shape the traffic. The S_AXI_HP{0:3}_FPD interfaces are designed to provide a latency guarantee for DDR memory controller accesses. Details on traffic categorization are described in Chapter 17, DDR Memory Controller. For details on system-level QoS, refer to Chapter 15, PS Interconnect.

IMPORTANT: *The [WR_RELEASE_MODE] bit in the AFIFM.WRCTRL register controls the write command release mode. For example, when 1, a write command is released as available and when 0, the data is buffered into the FIFO and a command is released when either 16-beats are enqueued or there is a WLAST (whichever occurs first). When this bit is set to 1, ensure that any masters issuing write transactions do not provide a command without data. Issuing a command without data can lead to starvation and other system-level issues. In other words, before the [WR_RELEASE_MODE] bit is set to 1, choose masters that will only issue write transactions after data is present.*

High Performance PS to PL AXI Interfaces

Two high-performance interfaces, M_AXI_HPM0_FPD and M_AXI_HPM1_FPD (data width selectable to be 32/64/128-bit), are provided to allow the CPUs, DMAs, and PCIe to push large amount of data from the PS to the PL. They are also AXI FIFO interfaces that enable the following.

- Conversion from the AXI3 to AXI4 protocols as the PL interfaces are AXI4 compliant, while the internal PS interfaces are AXI3 compliant. AXI4 access in the PL is limited to a burst length of 16.
- Clock domain crossing between PS and PL interfaces. A single clock is available in the PL interface for read and write operation.

The PS interconnect assigns the master ID bits and transfers these bits on the AxUSER bits of the associated AXI transaction. The AxUSER[9:0] bits correspond to the master IDs listed in Table 16-13. The AxUSER[15:10] bit might be used for other purposes by the system including coherency and transaction poisoning.

LPD-PL Interfaces

The high-performance interface port (S_AXI_LPD) from the PL to the LPD includes the following features.

- Configurable to 32, 64, or 128-bit data widths on the PL side.
- Preferred interface for PL access to the OCM and TCMs with the lowest latency.
- Access to all of the global address map (especially to access the PS DDR memory).

This port can be used in physical or virtual mode by setting the AxUSER bit. In virtual mode, it cannot directly access the LPD. Instead, virtual mode accesses are routed as follows. AxUSER should be set to 1'b1 to select virtual mode or 1'b0 to select physical mode.

PL → LPD → FPD (SMMU/CCI) → LPD

The S_AXI_LPD is a PL interface that connects into the low-power domain. For situations where the FP domain is powered down, this interface provides a high-performance mastering capability from the PL. Due to the interconnect topology, this port has a relatively long latency to DDR.

The low-latency interface port (M_AXI_HPM0_LPD) from the LPD to the PL includes the following features.

- Configurable to 32, 64, or 128-bit data widths on the PL side.
- AXI4 access in the PL, but is limited to a burst length of 16.
- Responds to lowest 512 MB memory in LPD's 32-bit address space.
- Enables direct access to the PL (for example for block RAM, DDR) for the safety use cases.

LPD bus masters can use the M_AXI_HPM0_LPD interface to access the PL without the FPD being powered-up.

IMPORTANT: *Exclusive access by the APU cannot be made to the M_AXI_HPM0_LPD signal due to an ID converter in the path.*

PL ACE Interface to CCI

The AXI coherency extension (ACE) protocol extends the AXI4 protocol and provides support for hardware coherent caches.

A Note About the ACE Protocol

The ACE is backwards compatible to AXI4 and supports coherent interconnects. In addition to five AXI4 channels, ACE adds three additional snoop channels and some extra signals. The ACADDR channel is a snoop-address input to the master. The CRRESP channel is used by the master to signal the response to snoops to the interconnect. The CDDATA channel is output from the master to transfer snoop data to the originating master and/or external memory. ARSNOOP and AWSNOOP indicate the type of snoop transactions for shareable transactions on the read and write channels, respectively. ARBAR and AWBAR are used for barrier signaling. ARDOMAIN indicates the masters to snoop for snoop transactions and the masters to be considered or the ordering of barrier transactions. RRESP has additional bits for shared read transactions that are indirectly driven by the CRRESP outputs from a snooped master. In addition to the full ACE interface, the AMBA-4 specification also defines ACE-Lite, which has the additional signals on the existing channels but not the new channels. ACE-Lite masters can snoop ACE-compliant masters, but cannot themselves be snooped.

For more details on ACE signaling, refer to Arm ACE protocol specification.

The ACE interface connects to cache coherent interconnect (CCI) and is configurable to support I/O and full coherency.

- I/O coherency though the use of ACE-Lite where I/O-coherent masters can snoop APU caches.
- Full coherency though the use of ACE where fully-coherent masters can snoop each other’s caches.

The two-way coherent S_AXI_ACE_FPD interface uses a 40-bit wide physical address. The ACE port enables the PL-masters to have their caches in PL. The PL-ACE master cannot allocate into the APU L2 cache however, it has coherent access to L2 cache.

If a PL ACE port is not used or used as ACE-Lite, then its snoop channels must be disabled (using the CCI ACCHANNELEN input that is controlled by a LPD_SLCR.LPD_CCI register). This ensures that the CCI does not generate snoop to the PL.

Note: Although the programmable logic (PL) ACE port can be used as an AXI4 interface, Xilinx recommends against this usage.

ACE-Lite Interface for I/O Coherency

The ACE-Lite interface is a defined subset of the full ACE interface. ACE-Lite is used by master components that do not have hardware coherent caches, but can issue transactions that could be held in the hardware coherent caches of other masters. ACE-Lite enables uncached masters to snoop ACE coherent masters.

The S_AXI_ACE_FPD port can be used as ACE-Lite with some limitations. In addition to providing one way coherency, ACE-Lite can be used to force flush or invalidate an APU cache.

The following describes using S_AXI_ACE_FPD as ACE-Lite.

- An ACE slave needs RACK and WACK inputs. But ACE-Lite master does not have RACK and WACK outputs. The PL must drive these signals because they are used to release transactions from the internal trackers in the CCI.
- In the CCI, the ACE interface does not support burst splitting. The PL master must ensure that any shareable transactions (ReadOnce, WriteUnique) do not cross a 64-byte boundary. Furthermore, if using a fine-grained interleaving (<4 KB), then the PL master must ensure that no transaction crosses the interleaving boundary.
- The ACE DVM [ACCHANNELEN] bit should be set Low (using the LPD_SLCR.LPD_CCI register). This will ensure that requests are never sent on the AC channel of this ACE.

The I/O coherent masters only need to indicate the shareability of a read or write transaction using AxDOMAIN. Other signals, such as AxSNOOP, AxBAR, and AxUNIQUE, can be tied to zero.

ACE-Lite provides I/O coherency-like S_AXI_HPCx_FPD ports. However, ACE-Lite requires physical address, additional signals, and ACE-related restrictions must be followed. Comparably, S_AXI_HPCx_FPD uses a virtual address and is AXI compliant. When possible, using S_AXI_HPCx_FPD is preferred for I/O coherency (instead of ACE-Lite).

ACE Interface for Full Coherency

Full-coherent masters can snoop each other's caches. For fine-grain data sharing between the APU and the PL, a system can have cache implemented in PL. Full coherency is provided through the CCI ACE ports. ACE provides additional signals that allow CCI to request data cached by various masters (APU or PL).

TIP: *For full coherency, transactions can only have a 64-byte cache line size.*

IMPORTANT: *When using the PL-ACE as an ACE-lite or AXI4 port, you must ensure that the PL master does not generate transactions with burst lengths greater than 16. From the ACE port to the DDR controller, there is an AXI4 path without a mechanism to split longer burst lengths into smaller transactions similar to how they are split by the FIFO-enabled AXI interfaces. Failure to limit transaction burst lengths can lead to lockups on the bus for many cycles, starvation on other DDR ports, and very high latencies observed on other masters in the system.*

Note: A cached PL master connected to the PL-ACE interface, that is required to be in an inner-shareable domain with the APU, should tie-off the BROADCASTINNER signal to 1. This can be done by writing to the LPD_SLCR.LPD_APU register. This tie-off signal must be either High or Low before the APU reset is deasserted.

ACP Interface

The accelerator coherency port (ACP) is a 128-bit AXI slave interface on the snoop control unit (SCU) that provides an asynchronous cache-coherent access point directly from the PL to the APU. Several PL masters can use this interface to access the caches and the memory subsystem in the same way the APU processors use to simplify software, increase overall system performance, or improve power consumption.

From a system perspective, the ACP interface has connectivity similar to the APU CPUs. Due to this close connectivity, the ACP directly competes with them for resource access outside of the APU MPCore.

TIP: *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 cache, then there is also the opportunity to hit in L2 cache before finally being forwarded to the DDR memory. 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 accesses do not go through either the APU's MMU or the System's SMMU, hence, their request-address is a 40-bit physical address.

IMPORTANT: *Since the PL-ACE does not have an AXI FIFO interface to regulate sending write data with or before the write command, care must be taken when choosing a master. Failure to choose a master that presents data before or along with the write command can lead to starvation and other system level issues.*

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®-A53 MPCore Processor Technical Reference Manual [Ref 46].

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. 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. Therefore, ACP is not recommended for fine-grained instruction level acceleration. Instead, for coarse-grain acceleration, such as video frame-level processing, 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. Therefore, ACP is optimal for medium-grain acceleration, such as a block-level crypto accelerator and video macro-block level processing.

The ACP port 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).

RECOMMENDED: *For the best power and performance, Xilinx recommends using either an S_AXI_HPCx_FPD port or the ACE port to provide I/O coherency as the preferred approach over ACP.*

Table 35-4: **PS-PL AXI Interfaces**

Interface Name	Abbreviation	Type	Master	Data Width	Master ID Width	Usage Description
S_AXI_HP{0:3}_FPD	HP{0:3}	AXI4	PL	128/64/32	6	Non-coherent paths from PL to FPD main switch and DDR. No L2 cache allocation.
S_AXI_HPM0_LPD	PL_LPD	AXI4	PL	128/64/32	6	Non-coherent path from PL to IOP in LPD.
S_AXI_ACE_FPD	ACE	ACE	PL	128	6	Two-way coherent path between memory in PL and CCI.
S_AXI_ACP_FPD	ACP	AXI4	PL	128	5	I/O coherent with CCI. With L2 cache allocation.
S_AXI_HPC{0, 1}_FPD	HPC{0, 1}	AXI4	PL	128	6	I/O coherent with CCI. No L2 cache allocation.
M_AXI_HPM{0, 1}_FPD	HPM{0, 1}	AXI4	PS	128/64/32	16	FPD masters to PL slaves.
M_AXI_HPM0_LPD	LPD_PL	AXI4	PS	128/64/32	16	LPD masters to PL slaves.

CAUTION! *Avoid the use of the ACP in security/safety critical applications requiring isolation within the APU and/or between PS and PL. The ACP, if enabled, has unrestricted access to the entire L2 cache of the APU and untrusted IP in the PL can make itself appear secure by modifying its AxPROT bits. If the ACP is used in this system, two precautions are highly recommended. The first, is to wrap any IP in the PL that has ACP access with trusted security logic that controls the AxPROT bits rather than allowing the IP to do so. The second is to ensure that access to the entirety of L2 cache by this IP does not violate the system security goals.*

Choosing a Programmable Logic Interface

This section discusses various options to connecting programmable logic (PL) to the processing system (PS). A qualitative overview of data transfer use cases is shown in Table 35-5 followed by a detailed discussion of certain use cases.

Table 35-5: **PL Interface Comparison**

Method	Benefits	Considerations	Application
APU/RPU Programmed I/O	• Simple software. • Least PL resources. • Simple PL slaves.	Low bandwidth demand.	Control functions.
FPD DMA LPD DMA	• Least PL resources. • Multiple channels. • Simple PL slaves. • Coherency (LPD DMA only).	FPD DMA is not coherent. LPD DMA is optionally coherent.	• FPD DMA for data movement between PS-DDR and PL. • LPD DMA for data movement between OCM and PL and safety use-cases.
S_AXI_HPC{0,1}_FPD DMA	• High throughput. • Multiple interfaces. • AXI FIFO interface with QoS-400 traffic shaping. • Hardware assisted coherency; no cache flush/invalidate in software driver. • Virtualization support with SMMU in path.	• More complex PL master design. • PL design to drive AxCACHE as needed for coherency. • Impacts the CCI and degrades APU and other masters accessing memory via the CCI.	Coherent, high-performance DMA for large datasets.
S_AXI_HP{0:3}_FPD DMA	• High throughput. • Multiple interfaces. • AXI FIFO interface with QoS-400 traffic shaping. • Virtualization support with SMMU in path.	• Software driver to handle cache flush/invalidate. • More complex PL master design.	Non-coherent, high-performance DMA for large datasets.
S_AXI_ACP_FPD DMA	• Lowest latency to L2 cache. • Two-way cache coherency. • Option to allocate into L2 cache.	• Limited to 16B and 64B transactions; impacting PL DMA design. • Shares APU MPCore interconnect bandwidth. • More complex PL master design.	• PL logic tightly coupled with APU. • Medium granularity CPU offload.

Table 35-5: **PL Interface Comparison *(Cont'd)***

Method	Benefits	Considerations	Application
S_AXI_ACE_FPD DMA	• Optional cache coherency. • APU can snoop into PL cached masters (two-way coherency).	• Burst length limited to 64B when CCI snoops PL master. • For ACE-Lite, long bursts from PL to PS may hang the APU MPCore due to the direct path from CCI to DDR memory, impacting others waiting for memory. • Complex PL design that require support for ACE.	• Cached accelerators in PL. • System cache in PL using block RAM.
S_AXI_LPD DMA	• Fastest, low latency path to the OCM and TCM. • Optional CCI coherency. • SMMU in datapath provides option for virtualization. • PL access to LPD when FPD is powered off.		Safety applications.

The data movement use-cases in Table 35-5 are described in the following sections.

APU Perspective

From the software perspective, the least intrusive method of programming the I/O is to use a processor in the APU MPCore to move data between the PS and PL. As shown in Figure 35-4, data is directly moved by the CPU, thus removing the need to handle events from a separate DMA. Access to the PL is provided through the two M_AXI_HPMx_FPD master ports, which target a memory address range in the PL. The PL design is also simplified because a single AXI slave can be implemented to service the CPU requests.

Some drawbacks of using a CPU to move data is that a valuable CPU is spending cycles performing simple data movement instead of complex control and computation tasks, and the available throughput is limited.

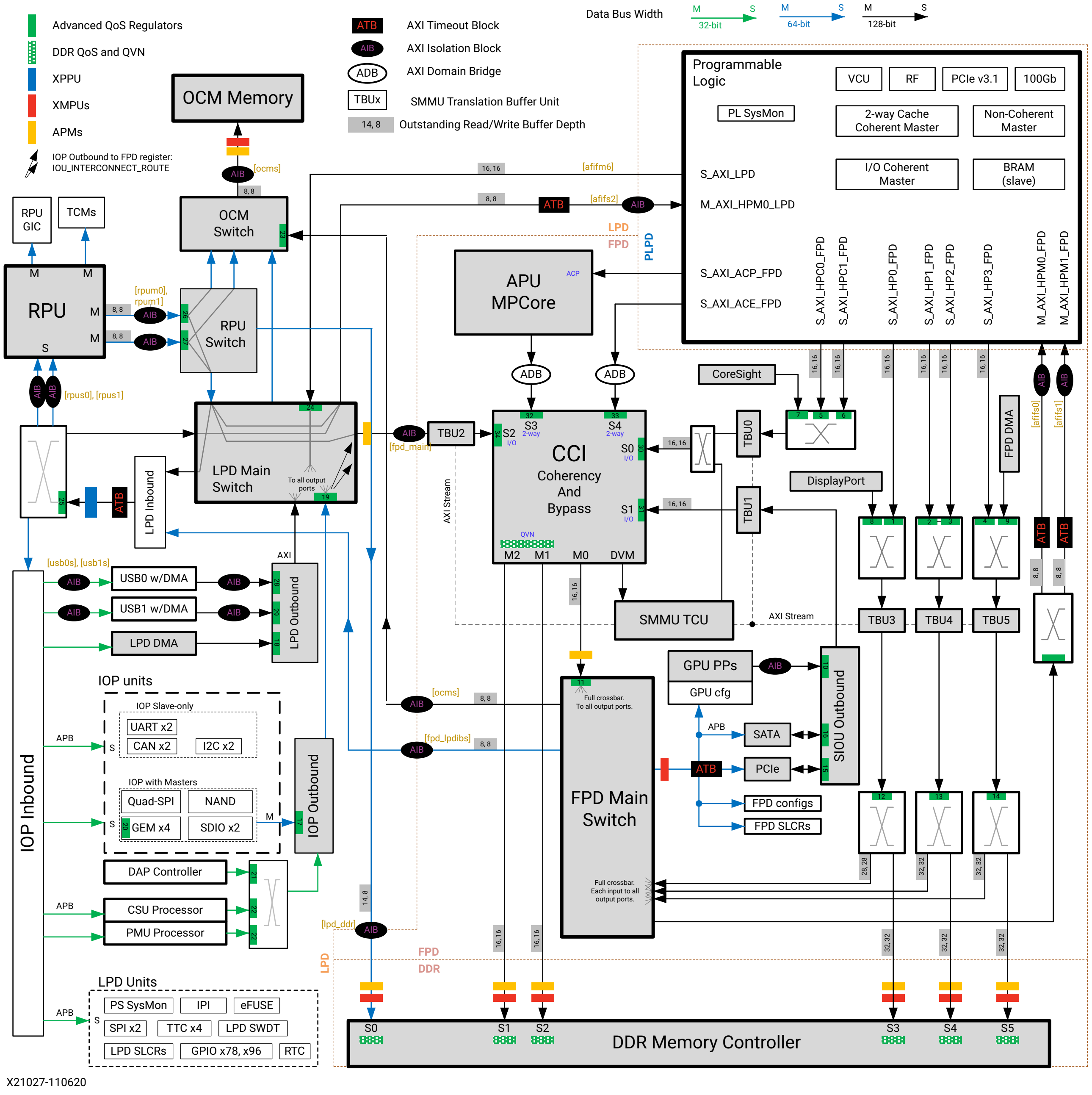


Figure 35-4: **RPU and APU Masters**

RPU Perspective

The RPU and LPD to PL interface are similar to the APU Perspective case. In Figure 35-4, note that this data movement is purely limited to the LPD and PL and does not involve the FPD. This is useful for systems where FPD might be powered down.

FPD and LPD DMAs

Figure 35-5 shows one use case of the LPD and FPD DMA units for the following scenarios.

- The FPD DMA unit for data movement between the DDR memory controller in the PS FPD and block RAM in the PL where the APU manages the FPD DMA unit and the M_AXI_HPM{0,1}_FPD interface is used for connectivity into the PL.
- The LPD DMA unit for data movement between the OCM memory in the PS LPD and block RAM in the PL where the RPU manages the LPD DMA unit and HPM0_LPD interface is used for connectivity with the PL.

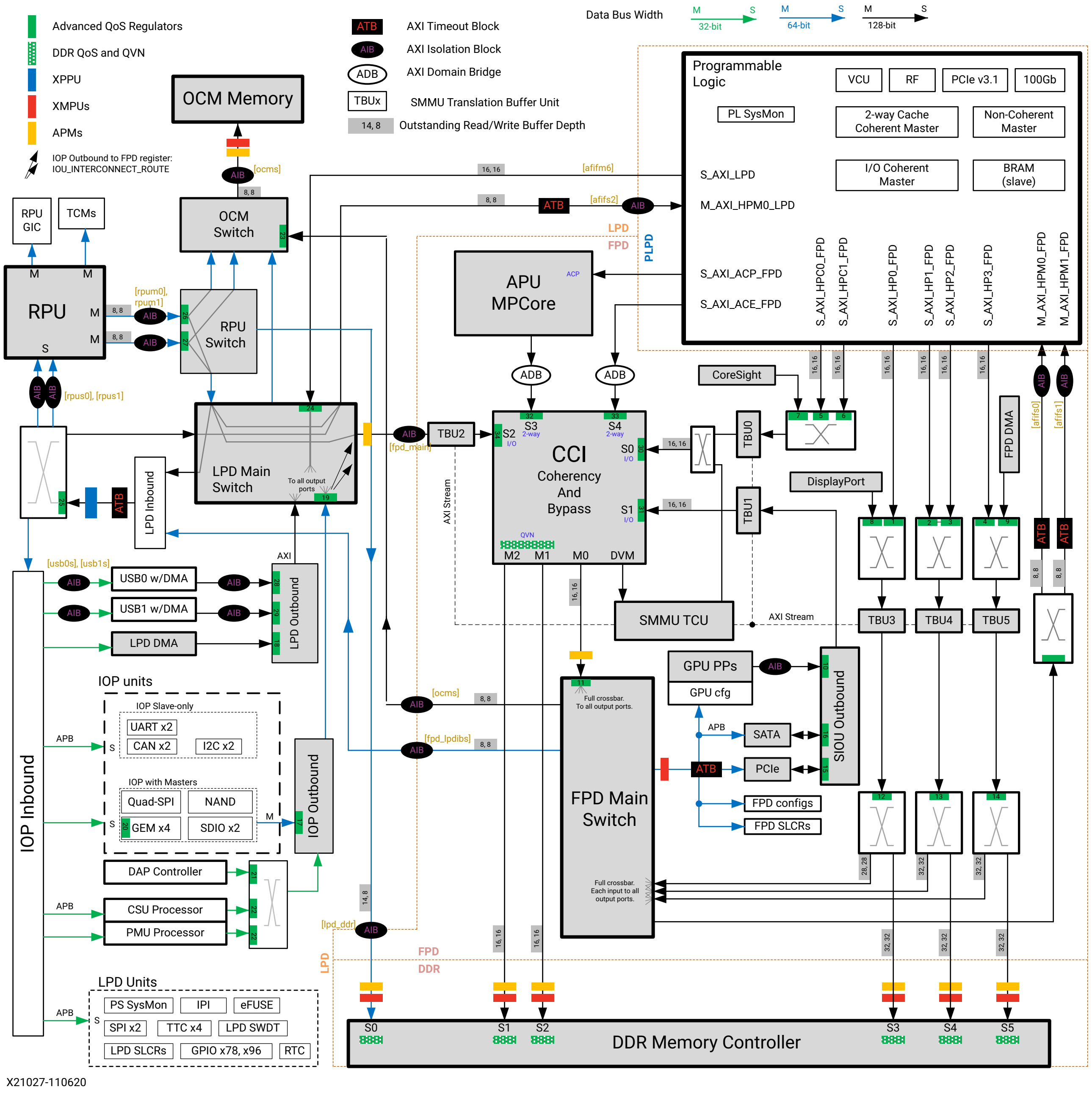


Figure 35-5: **DMA Masters**

While the FPD DMA unit can be used for the OCM memory to PL block RAM transfers and the LPD DMA unit can be used for the DDR memory controller to PL block RAM transfers, the use case in this section is an example of how the LPD can operate independently when the FPD is powered down.

PL DMA using the HP and HPC Interfaces

The HP and HPC interfaces provide a high-performance datapath to the PS-DDR and the OCM memories. When using the HPC interface, requests from the PL to the PS-DDR go through the CCI which manages the APU MPCore cache coherent environment. These use-case topologies are shown in Figure 35-6 and are described in the following section.

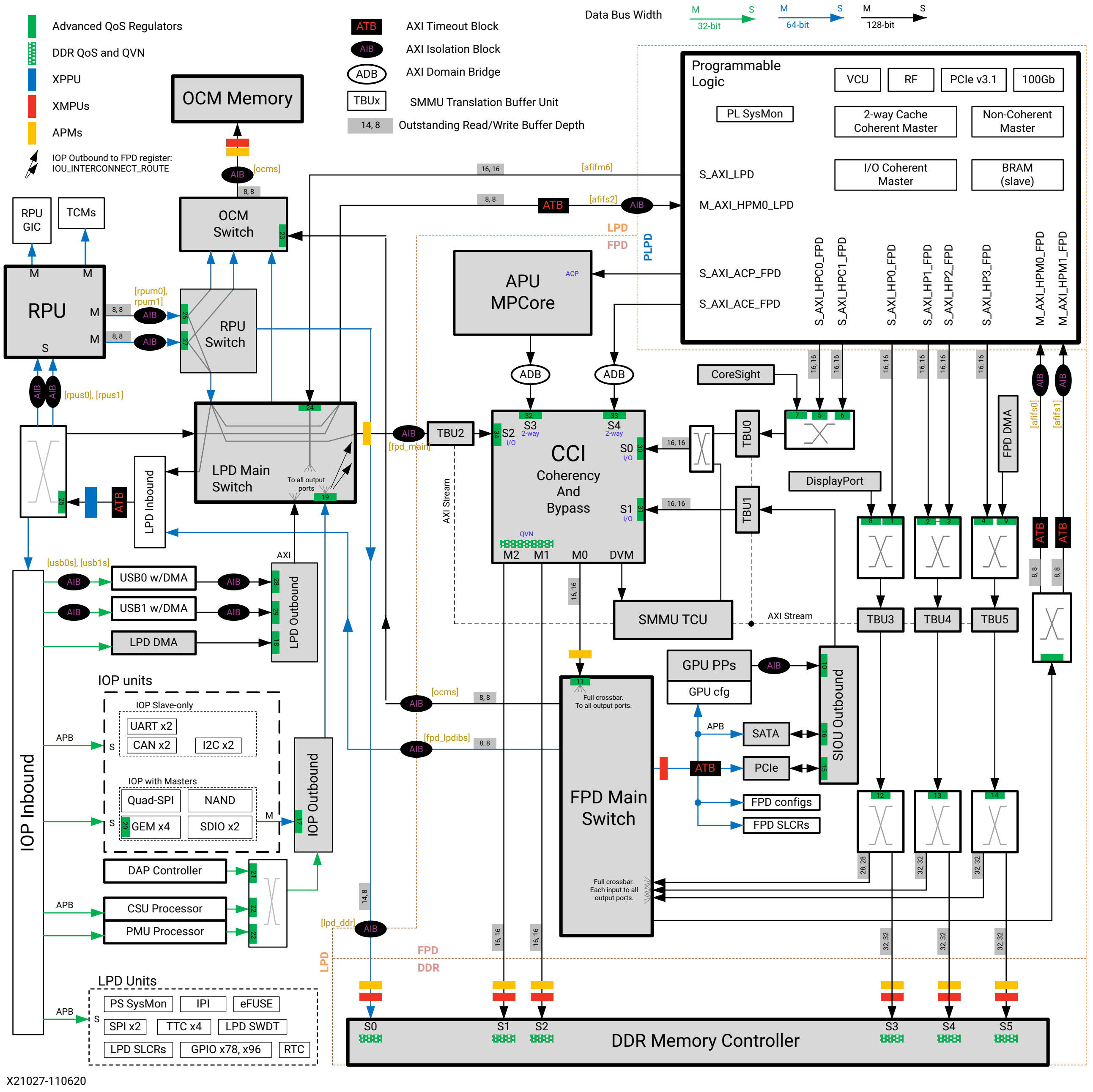


Figure 35-6: **PL Coherent Masters**

PL Accelerator Block and FPD Interaction

The DMA and the accelerator block are controlled by the APU through the M_AXI_HPMx_FPD interfaces. The DMA can access the PS-DDR through the S_AXI_HPCx_FPD or S_AXI_HPx_FPD interfaces. The difference is that the hardware assisted cache coherency using the HPC ports helps the software driver avoid costly cache flush/invalidate operations.

PL Accelerator Block and LPD Interaction

In the PL accelerator block to LPD interaction, there is no path through the FPD (no PS-DDR access) to ensure functionality when FPD is powered down. The RPU controls the PL-based DMA and the accelerator block through the M_AXI_HPM0_LPD interface. The DMA can access the OCM through the S_AXI_LPD port.

PL DMA via ACP

The AXI ACP interface (S_AXI_ACP_FPD) provides a user IP topology similar to the high-performance S_AXI_HPx_FPD interfaces.

The ACP differs from the HP performance ports due to connectivity inside the PS. The ACP connects to the snoop control unit (SCU) that is also connected to the CPU L1 and the L2 cache.

This connectivity allows the ACP transactions to interact with the cache subsystems, potentially decreasing total latency for data to be consumed by a CPU. These cache-coherent operations can prevent the need to invalidate and flush cache lines. The ACP also has the lowest memory latency to memory of the PL interfaces. The connectivity of the ACP is similar to that of the CPUs.

The drawbacks from using the ACP include PL design complexity due to support of only two burst length transactions. Memory accesses through the ACP utilize the same interconnect paths as the APU, potentially decreasing CPU performance.

The ACP low-latency access allows opportunity for algorithm acceleration of medium granularity.

System Cache using ACE

In scenarios where the AXI interconnect in the PL does not support ACE, all ACE accesses go to the system cache using a point-to-point interface instead of an AXI-ACE interconnect. System cache uses block RAM in the PL to implement the memory cells of cache.

Figure 35-7 shows an example of the PL accelerator implementation with system cache using ACE. The PL accelerator can write the data into system cache and the APU can access the same data through ACE. This improves APU performance (as read latency is reduced when compared to reading from DDR) and reduces the DDR bandwidth requirement as DDR access is reduced.

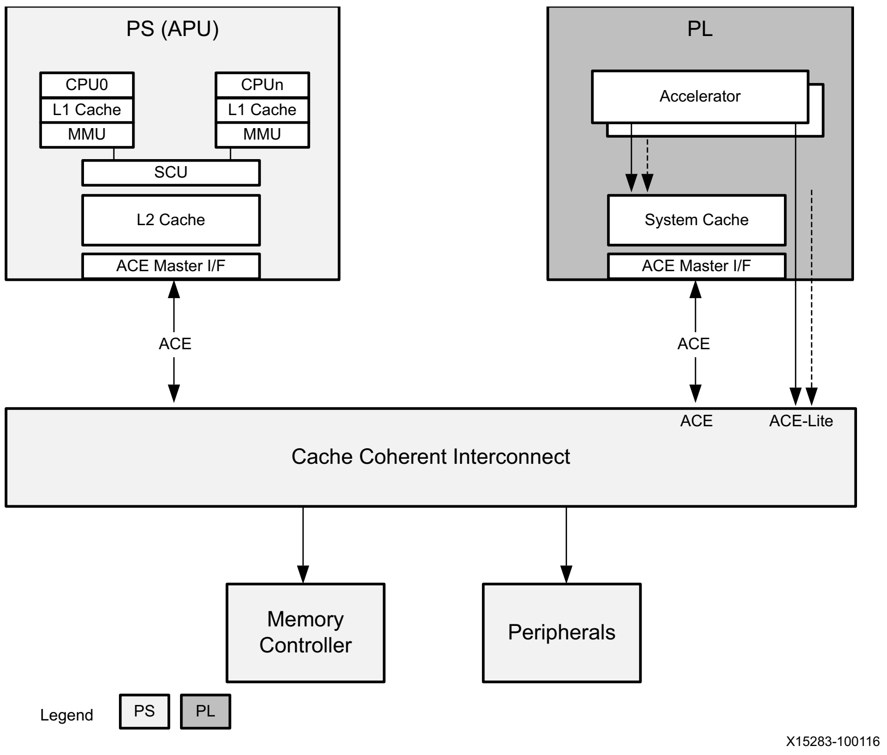


Figure 35-7: **Use of PL Block RAM as System Cache**

Signal Overview

PS-PL Interrupts

The interrupts from the processing system I/O peripherals (IOP) are routed to the PL. In the other direction, the PL can asynchronously assert 16 interrupts to the PS. These interrupts are assigned a priority level routed to interrupt controllers which aggregate and route them to appropriate processor. Additionally, FIQ/IRQ interrupts are available which are routed directly to the private peripheral interrupt unit of the interrupt controller. Table 35-6 summarizes the interrupts.

Table 35-6: **PS-PL Interrupts Summary**

Type	Number of Interrupts	Start ID	End ID	Description
PL to PS interrupts	8	89	96	PL to PS shared peripheral interrupts.
	8	104	111	PL to PS shared peripheral interrupts.
	1	29	29	PL to PS (RPU, APU) inter-processor interrupt.
	1	30	30	PL to PS (RPU, APU) inter-processor interrupt.
	1	31	31	PL to PS (RPU, APU) inter-processor interrupt.
	1	32	32	PL to PS (RPU, APU) inter-processor interrupt.
	4			PL to APU legacy FIQ
	4			PL to APU legacy IRQ
	2			nFIQ (PL to RPU0 and RPU1)
	2			nIRQ (PL to RPU0 and RPU1)
PS to PL interrupt outputs	100	~	~	Interrupts generated by I/O peripherals in the LPD and distributed to the GICs and PL. See Figure 13-1[1].
	64	~	~	Interrupts generated by I/O peripherals in the FPD and distributed to the GICs and PL. See Figure 13-1[1].

Notes:

1. The interrupts generated from the I/O peripherals are distributed to GIC and PL and can only be routed to either GIC or to PL, but not both at the same time.

For more information on interrupts, refer to Chapter 13, Interrupts.

Processor Event Signals

The PS supports processor events (Table 35-7) to and from the PL. These signals are asynchronous to the PS and PS provided PL clocks.

Table 35-7: **PL Event Signals**

Type	Signal	Description
FPD events	APU event input	Causes CPU to wake from the wait for event (WFE) state.
	APU event output	Asserted when one of the CPUs has executed a send event (SEV) instruction.
LPD events	RPU event input (2)	Event input to the RPU (one event signal to each RPU).
	RPU event output (2)	Event output from the RPU (one event signal from each RPU).
Standby (FPD)	APU WFE	CPU standby mode. Asserted when CPU is waiting for an event.
	APU WFI	CPU standby mode. Asserted when a CPU is waiting for an interrupt.

For further details, refer to Chapter 3, Application Processing Unit and Chapter 4, Real-time Processing Unit.

Register Overview

This section describes a few of the registers used for setting up options across various PS-PL interfaces. The registers in Table 35-8 are available for every AXI FIFO interface S_AXI_HPC{0, 1}_FPD, S_AXI_HP{0:3}_FPD, and S_AXI_LPD).

Table 35-8: **PL Master Registers**

Register Name	Description
RDCTRL	Read channel control.
RDISSUE	Read issuing capability.
RDQoS	QoS read channel.
RDDEBUG	Read channel debug.
WRCTRL	Write channel control.
WRISSUE	Write issuing capability.
WRQoS	QoS write channel.
I_STS, I_EN, I	Interrupt status.
I_EN	Interrupt enable.
I_DIS	Interrupt disable.
I_MASK	Interrupt mask.
CONTROL	General control.
SAFETY_CHK	Register access integrity test register.

Chapter 36

PL Peripherals

Introduction

The Xilinx® UltraScale™ architecture in the programmable logic (PL) provides an extensive set of functions and resources. The processing syst/em (PS) boots the system and includes the real-time processing unit (RPU) and application processing unit (APU) MPCores as two separate software processing structures in the low-power domain (LPD) and full-power domain (FPD), respectively. The PL is a configurable hardware resource, and provides block RAMs, gates, clock structures, standard and high-range I/O, DSPs, and LUTs. The Zynq® MPSoC devices include several peripherals controllers and functional units.

- PCI Express Integrated
- 100G Ethernet
- DisplayPort Video and Audio Interfaces
- Interlaken
- GTH and GTY Transceivers
- PL System Monitor
- Video Codec Unit
- RFSoC

PCI Express Integrated

The PL includes integrated blocks for PCIe® technology that can be configured as an Endpoint or Root Port, compliant to the PCI Express Base Specification Revision 3.1 for Gen3 and lower data rates and compatible with the PCI Express Base Specification Revision 4.0 (rev 0.5) for Gen4 data rates. The Root Port is used to build the basis for a compatible Root Complex, to allow custom chip-to-chip communication via the PCI Express protocol, and to attach ASSP Endpoint devices, such as Ethernet controllers or Fibre Channel HBAs, to the MPSoC. This block is highly configurable to system design requirements and can operate on 1, 2, 4, 8, or 16 lanes at up to 2.5 Gb/s, 5.0 Gb/s, 8.0 Gb/s, or 16 Gb/s data rates. For high-performance applications, advanced buffering techniques of the block offer a flexible maximum payload size of up to 1,024 bytes. The integrated block interfaces to integrated high-speed transceivers for serial connectivity and to block RAMs for data buffering. Combined, these elements implement the physical layer, data-link layer, and transaction layer of the PCI Express protocol.

Xilinx provides a light-weight, configurable, easy-to-use IP wrapper that ties the various building blocks (the integrated block for PCIe, the transceivers, block RAM, and clocking resources) into an Endpoint or Root Port solution. You have control over many configurable parameters in your system: link width and speed, maximum payload size, MPSoC logic interface speeds, reference clock frequency, and base address register decoding and filtering.

The PCIe controller is documented in the *UltraScale+ Devices Integrated Block for PCI Express Product Guide* (PG213) [Ref 28].

100G Ethernet

The 100G Ethernet controllers are compliant to the IEEE Std 802.3ba, and provide low latency 100 Gb/s Ethernet ports with a wide range of user customized solutions and statistics gathering. With support for 10 x 10.3125 Gb/s (CAUI) and 4 x 25.78125 Gb/s (CAUI-4) configurations, the integrated 100G Ethernet includes both the 100G MAC and PCS logic with support for IEEE Std 1588v2 1-step and 2-step hardware time stamping.

The 100G Ethernet controllers contain a Reed-Solomon forward error correction (RS-FEC) block, compliant to IEEE Std 802.3bj, that can be used with the Ethernet block or stand alone in user applications. These families also support OTN mapping mode where the PCS can be operated without using the MAC.

DisplayPort Video and Audio Interfaces

The DisplayPort controller implements a flexible display and audio pipeline architecture. The DisplayPort controller can source data from memory (non-live input) or from the PL (live input), process data, and send it out through the DisplayPort source-only controller block to external display devices or to the PL (live output).

A brief introduction is included in this section. For additional information, see the *Video PHY Controller LogiCORE IP Product Guide* (PG230) [Ref 29].

Live Video/Graphics Input

In the live input case, video and graphics data can be sourced from the PL. The video and graphics frame synchronization signals are input to the live input interface. The video timing can be controlled either from the PS or from the PL.

Live Video Output

The output of the video rendering pipeline can optionally be routed to the PL through the live video output interface.

See Table 33-2 for PS-PL video interface signals and to Table 33-3 for live video timing on the PS-PL interface.

Audio

The DisplayPort controller supports a non-live audio channel from memory and a live audio channel from the PL. The audio mixer block is capable of mixing the two audio channels based on predefined gain settings. The output of the mixer can either be sourced to the DisplayPort source-only controller or to the PL.

Audio Live Input

Live audio can be sourced from PL. The A/V buffer manager handles multiplexing between live and non-live audio input and provides two audio streams to the audio mixer block. For more details about the live audio interface, see the PS-PL Audio Interface in Chapter 33.

Audio Live Output

The audio processing stage involves mixing two audio streams based on a predefined gain setting. The mixed audio in AXI-S format is forwarded to the DisplayPort source controller and the PL. A small holding buffer that supports AXI-S is used to handshake with the PL. The audio channel does not accept any back pressure from the PL interface. Refer to Table 33-6 for live audio interface signals.

Interlaken

The integrated Interlaken is a scalable chip-to-chip interconnect protocol designed to enable transmission speeds from 10 Gb/s to 150 Gb/s. The integrated Interlaken unit is compliant to revision 1.2 of the Interlaken specification with data striping and de-striping across 1 to 12 lanes. Permitted configurations are: 1 to 12 lanes at up to 12.5 Gb/s and 1 to 6 lanes at up to 25.78125 Gb/s, enabling flexible support for up to 150 Gb/s per integrated block. Multiple integrated Interlaken units in the PL can provide reliable Interlaken switches and bridges.

The Interlaken controller is documented in the *Integrated Interlaken 150G LogiCORE IP Product Guide* (PG169) [Ref 30].

GTH and GTY Transceivers

Ultra-fast serial data transmission between devices on the same PCB over backplanes and across even longer distances is becoming increasingly important for scaling to 100 Gb/s and 400 Gb/s line cards. Specialized dedicated on-chip circuitry and differential I/O capable of coping with the signal integrity issues are required at these high data rates. There are three types of transceivers: GTH, GTY, and PS-GTR. All transceivers are arranged in groups of four, known as a transceiver Quad. Each serial transceiver is a combined transmitter and receiver. GTH and GTY Quads are PL peripherals and PS-GTR is a PS peripheral.

GTH serial gigabit transceivers are available in the CG and EG grade Zynq UltraScale+ devices except where noted in the *Zynq UltraScale+ MPSoC Product Overview* (DS891) [Ref 1]. The GTH transceivers are power efficient, supporting line rates from 500 Mb/s to 16.375 Gb/s.

The GTY transceivers are power efficient, supporting line rates from 500 Mb/s to 30.5 Gb/s in UltraScale FPGAs, and 32.75 Gb/s in UltraScale+ FPGAs.

The GTH and GTY transceivers are configured using the PS configuration wizard (PCW) available in the Vivado® Integrated Design Environment (IDE).

Transmitter

The transmitter is fundamentally a parallel-to-serial converter with a conversion ratio of 16, 20, 32, 40, 64, or 80 for the GTH transceiver and 16, 20, 32, 40, 64, 80, 128, or 160 for the GTY transceiver This allows a trade-off of datapath width against the timing margin in high-performance designs. These transmitter outputs drive the PC board with a single-channel differential output signal. TXOUTCLK is the appropriately divided serial data clock and can be used directly to register the parallel data coming from the internal logic. The incoming parallel data is fed through an optional FIFO and has additional hardware support for the 8B/10B, 64B/66B, or 64B/67B encoding schemes to provide a sufficient number of transitions. The bit-serial output signal drives two package pins with differential signals. This output signal pair has a programmable signal swing, as well as programmable pre- and post-emphasis to compensate for PC board losses and other interconnect characteristics. For shorter channels, the swing can be reduced to reduce power consumption.

Receiver

The receiver is fundamentally a serial-to-parallel converter, changing the incoming bit-serial differential signal into a parallel stream of words, each 16, 20, 32, 40, 64, or 80 bits in the GTH transceiver or 16, 20, 32, 40, 64, 80, 128, or 160 for the GTY transceiver. This allows a trade-off of internal datapath width against logic timing margin. The receiver takes the incoming differential data stream, feeds it through the programmable DC automatic gain control, linear and decision feedback equalizers (to compensate for PC board, cable, optical and other interconnect characteristics), and uses the reference clock input to initiate clock recognition. There is no need for a separate clock line. The data pattern uses non-return-to-zero (NRZ) encoding and optionally ensures sufficient data transitions by using the selected encoding scheme. Parallel data is then transferred into the device logic using the RXUSRCLK clock. For short channels, the transceivers offer a special low-power mode (LPM) to reduce power consumption by approximately 30%. The receiver DC automatic gain control and linear and decision feedback equalizers can optionally auto-adapt to automatically learn and compensate for different interconnect characteristics. This enables even more margin for tough 10G+ and 25G+ backplanes.

Out-of-Band Signaling

The transceivers provide out-of-band (OOB) signaling, often used to send low-speed signals from the transmitter to the receiver while high-speed serial data transmission is not active. This is typically done when the link is in a powered-down state or has not yet been initialized. This benefits PCIe and SATA/SAS and QPI applications.

For more details on GTH transceivers, see *UltraScale Architecture GTH Transceiver User Guide* (UG576) [Ref 12] and for GTY transceivers, see *UltraScale Architecture GTY Transceiver User Guide* (UG578) [Ref 13].

PL System Monitor

The PL SYSMON unit is used to enhance the overall safety, security, and reliability of the system by monitoring the physical environment via on-chip power supply and temperature sensors. The PL and PS SYSMON units are based on the Xilinx SYSMONE4 architecture. The units have different sensor channels.

The PL SYSMON and PS SYSMON units are described in Chapter 9, System Monitors. For more information, see *UltraScale Architecture System Monitor User Guide* (UG580) [Ref 6].

Video Codec Unit

The video codec unit (VCU) provides multi-standard video encoding and decoding, including support for the high-efficiency video coding (HEVC) H.265 and advanced video coding (AVC) H.264 standards. The unit contains both encode (compress) and decode (decompress) functions, and is capable of simultaneous encode and decode.

The VCU is an integrated block in the PL of selected Zynq UltraScale+ MPSoCs with no direct connections to the PS.

A brief introduction is included in this section. For more information, see the *H.264/H.265 Video Codec Unit LogiCORE IP Product Guide* (PG252) [Ref 31].

Video Codec Unit Features

- H.264 and H.265 standards encoding/decoding.
- Up to eight simultaneous streams.
- 8K x 4K at a reduced frame rate.
- Progressive video only (no interlace support).
- I, IP, and IPB encoding/decoding.
- 8-bit and 10-bit color depth, YCbCr 4:2:2 and 4:2:0 video formats, and up to a 4K x 2K@60/8K x 2K@15 Hz rate.
- Low-latency mode.
- Low software overhead for slice-level management and multi-stream switching.
- Functions to queue tasks to eliminate dependency on CPU interrupt response time.
- Power management.
    - Clock and Power Island Controls.
    - Built-in power sequencing state machine or interface to an external one.
- Performance monitoring.
    - Task execution time.
    - Bandwidth and AXI transaction count.
    - Min, max, and average AXI transaction latency.

The VCU contains encoder and decoder interfaces. The VCU also contains additional functions that facilitate the interface between the VCU and the PL.

VCU operation requires a small amount of processor activity. Interrupt response time must not be critical. The encoder is controlled by the CPU through a task list prepared in advance, and the CPU response time is not in the execution critical path. The VCU has no audio support. Audio encoding and decoding is supported on the PS application processors.

Block Diagram

Figure 36-1 shows the VCU block diagram

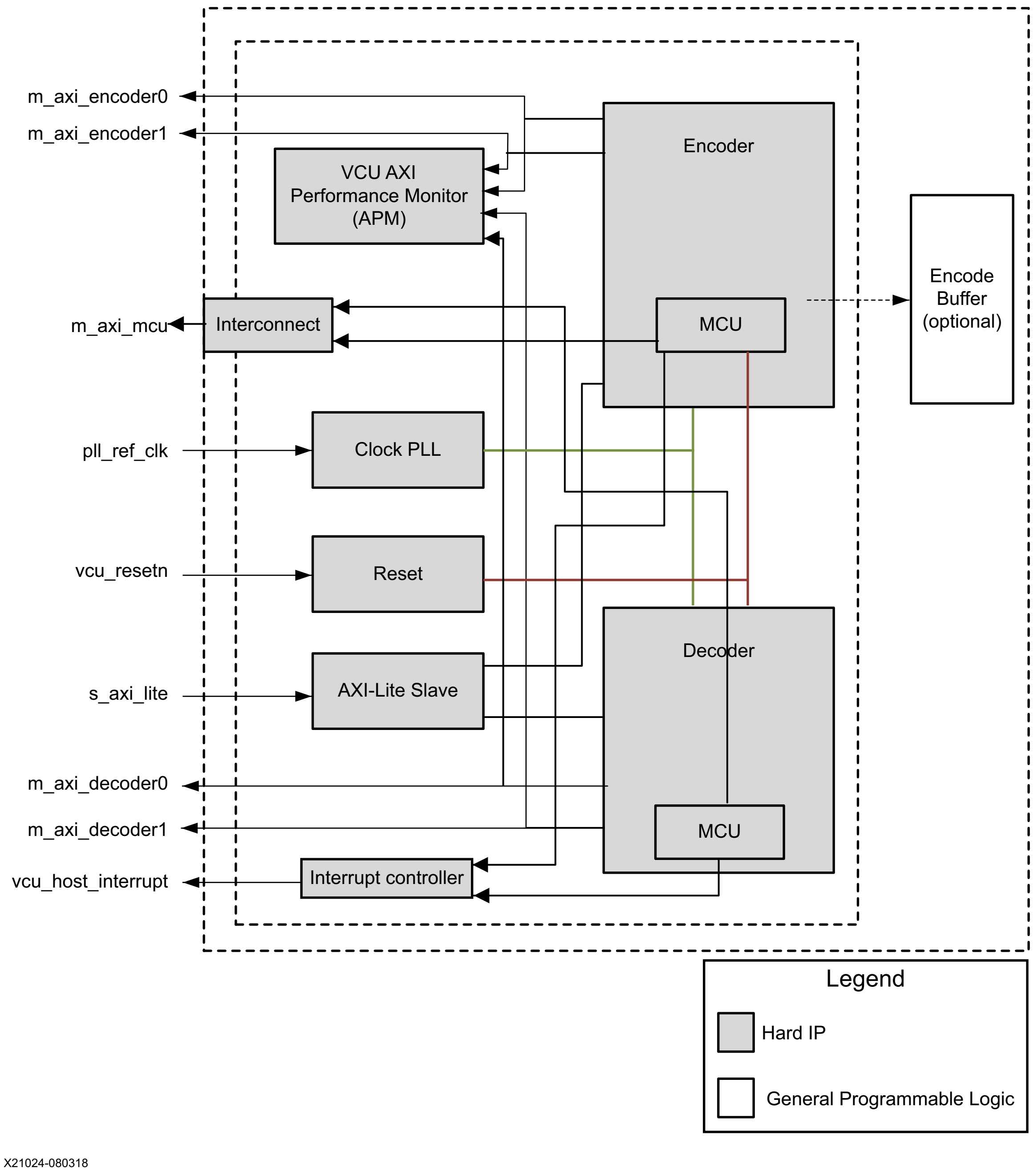


Figure 36-1: **VCU Top-level Block Diagram**

Video Encoder

The VCU encoder includes four interconnected HEVC/AVC encoders. It also contains global registers, an interrupt controller, and a timer. The VCU encoder is controlled by a microcontroller (MCU) subsystem. A 32-bit APB slave interface is used by the system CPU to control the MCU (to configure encoder parameters). Two 128-bit AXI4 master interfaces are used to fetch video input data and store video output data from/to the system memory. Two 32-bit AXI4 master interfaces are used to fetch the MCU software (instruction cache interface) and load/store additional MCU data (data cache interface).

Video Decoder

The VCU decoder includes two interconnected HEVC/AVC decoders. It also contains global registers, an interrupt controller, and a timer.

The VCU decoder is controlled by a microcontroller (MCU) subsystem. A 32-bit APB slave interface is used by the system CPU to control the MCU. Two 128-bit AXI4 master interfaces are used to fetch video input data and store video output data from/to the system memory. Two 32-bit AXI4 master interfaces are used to fetch the MCU software (instruction cache interface) and load/store additional MCU data (data cache interface).

Each decoder includes control registers, a bridge unit and a set of internal memories. The bridge unit manages the request arbitration, burst addresses, and burst lengths for all external memory accesses required by the decoder. It also handles format conversion and border extension.

The VCU has a direct access to the system data bus through a high-bandwidth master interface to transfer video data to/from an external memory.

The VCU control software is partitioned into two layers. The application software runs on the RPU or APU MPCores, and the low-level code is implemented in the MCU. The processor communicates with the embedded MCU through a slave interface, which is also connected to the system bus.

RFSoC

The Zynq® UltraScale+™ RFSoC family integrates key subsystems for multiband, multi-mode cellular radios and cable infrastructure (DOCSIS) into an SoC platform that contains a feature-rich 64-bit quad-core Arm® Cortex™-A53 and dual-core Arm Cortex-R5F based processing system.

A brief introduction is included in this section. For more information, see the *Zynq UltraScale+ RFSoC RF Data Converter LogiCORE IP Product Guide* (PG252) [Ref 26].

Combining the processing system with UltraScale™ architecture programmable logic and RF-ADCs, RF-DACs, and soft-decision FECs, the Zynq UltraScale+ RFSoC family is capable of implementing a complete software-defined radio including direct RF sampling data converters, enabling CPRI™ and gigabit Ethernet-to-RF on a single, highly programmable SoC.

Zynq UltraScale+ RFSoCs integrate up to 16 channels of RF-ADCs and RF-DACs. The RF-ADCs can sample input frequencies up to 4 GHz at 4.096 GSPS with excellent noise spectral density. The RF-DACs generate output carrier frequencies up to 4 GHz using the 2nd Nyquist zone with excellent noise spectral density at an update rate of 6.554 GSPS. The RF data converters also include power efficient digital down converters (DDCs) and digital up converters (DUCs) that include programmable interpolation and decimation, NCO, and complex mixer. The DDCs and DUCs can also support dual-band operation.

The soft-decision FEC (SD-FEC) is a highly flexible forward error correction engine capable of operating in Turbo decoding mode for wireless applications such as LTE and LDPC encode/decode mode used in 5G wireless, backhaul, and DOCSIS 3.1 cable modems.

Figure 36-2 shows the key components of the Zynq UltraScale+ RFSoC devices.

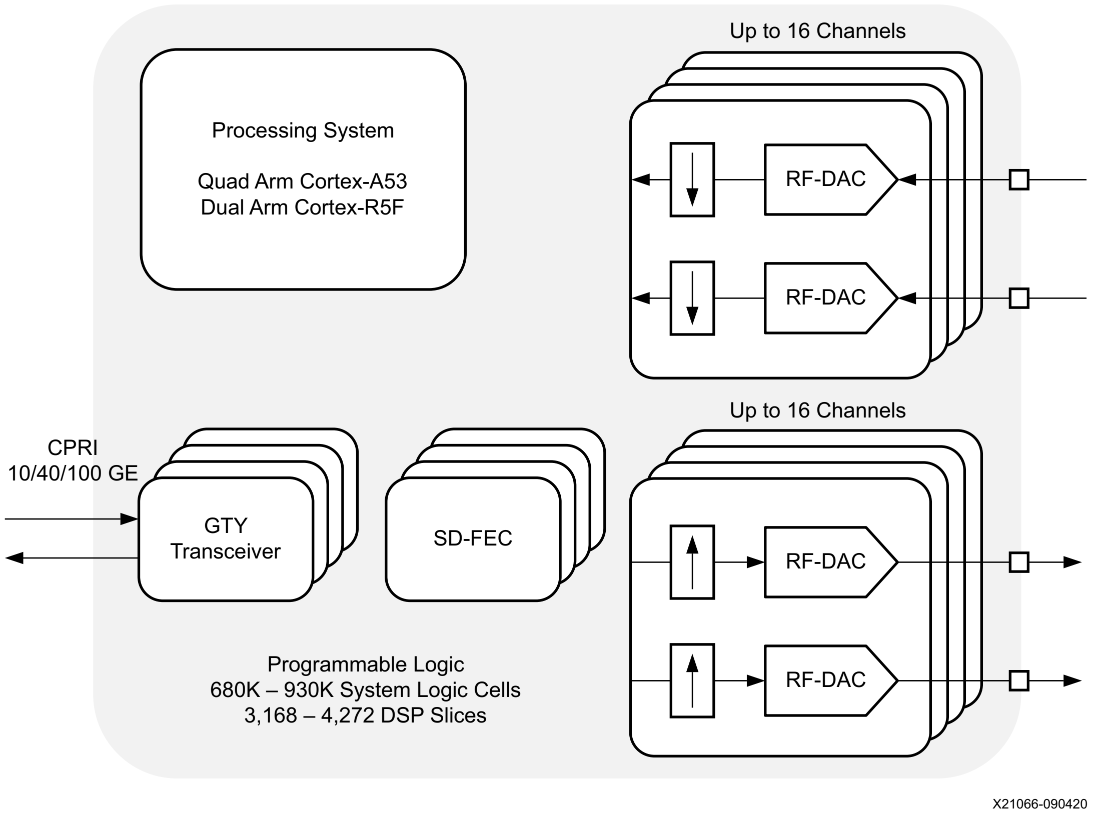


Figure 36-2: **Zynq UltraScale+ RFSoC**

RF Data Converter Subsystem Overview

Most Zynq UltraScale+ RFSoCs include an RF data converter subsystem, which contains multiple radio frequency analog to digital converters (RF-ADCs) and multiple radio frequency digital to analog converters (RF-DACs). The high-precision, high-speed, power efficient RF-ADCs and RF-DACs can be individually configured for real data or can be configured in pairs for real and imaginary I/Q data. The 12-bit RF-ADCs support sample rates up to 2.058 GSPS or 4.096 GSPS, depending on the selected device. The 14-bit RF-DACs support sample rates up to 6.554 GSPS.

RF-ADC Features

- Tile oriented
    - Four RF-ADCs and one PLL per tile
    - 12-bit resolution
    - Implemented as either 4 channels of 2.058 GSPS, or 2 channels of 4.096 GSPS (device dependent)

- Decimation filters
  - 1x, 2x, 4x, 8x
  - Full bandwidth data-rate support
  - 80% pass band, 89dB stop-band attenuation
- Mixer
  - Full complex mixers
  - 48-bit NCO per RF-ADC
  - Fixed Fs/4, Fs/2 low-power mode
- Single/multiband flexibility
  - 2x bands per 2.058 GSPS RF-ADC pair
  - Can be configured for real or imaginary (I/Q) inputs
- Signal amplitude threshold
  - Two programmable flags per RF-ADC
- Quadrature modulator correction
  - Gain/phase/offset correction per RF-ADC pair
- Multi-chip synchronization
- Flexible interconnect logic interface
  - N words x frequency selection
- RF-DAC Features
  - Tile oriented
  - Four RF-DACs and one PLL per tile
  - 14-bit resolution
  - Sampling speed 6.554 GSPS per RF-DAC
  - 4GHz full power output bandwidth
- Interpolation
  - 1x, 2x, 4x, 8x
  - Full bandwidth data rate support
  - 80% pass band, 89 dB stop band attenuation
- Mixing
  - Full complex mixers
  - 48-bit NCO per RF-DAC

    - Fixed Fs/4, Fs/2 low-power mode
    - 1st/2nd Nyquist zone RF-DAC operation support
- Single/multiband flexibility
    - 2x bands per RF-DAC pair
    - Can be configured for real or imaginary (I/Q) outputs
- Quadrature modulator correction
    - Gain/phase/offset correction per RF-DAC pair
- Sinx/x correction
- Sample delay correction
- Multi-chip synchronization
- Flexible interconnect logic interface
    - N words x frequency selection

See the *Zynq UltraScale+ RFSoC RF Data Converter LogiCORE IP Product Guide* (PG269) [Ref 26] for more information.

Soft Decision Forward Error Correction (SD-FEC)

The SD-FEC is a highly flexible soft-decision FEC decoder and LDPC encoder with the following features.

LDPC Decoding/Encoding

- Highly configurable codes.
    - A range of quasi-cyclic codes can be configured over an AXI4-Lite interface
    - Code parameter memory can be shared across up to 128 codes
    - Codes can be selected on a block-by-block basis
    - Encoder can re-use suitable decoder codes
- Normalized min-sum decoding algorithm
    - Normalization factor programmable (from 0.0625 to 1 in steps of 0.0625) for layers
- Number of iterations between 1 and 63
    - Specified for each codeword
- Early termination
    - Specified for each codeword to be none, one, or both of the following:
        - Parity check passes

- No change in hard information or parity bits since last iteration

- Soft or hard outputs
  - Specified for each codeword to include information and optional parity
  - 6-bit soft log likelihood ratio (LLR) input and 8-bit output (8-bit interface, 2 fractional bits, with external saturation before input to symmetric range –7.75 to +7.75)
- In- or out-of-order execution of blocks, with user specified ID field to identify blocks

Turbo Decoding

- Max, Max Scale (scale factor is programmable as a multiple of 0.0625), or Max Star
- Number of iterations between 1 and 63
  - Specified for each block via streaming control interface
- Early termination
  - Specified for each codeword to be none, one, or both of the following:
    - No change in hard decision since last iteration
    - CRC pass
- Soft or hard outputs
  - Specified for each codeword to include systematic and optionally parity 0 and parity 1
  - 8-bit soft LLR on input and output (8-bit interface, 2 fractional bits, with external saturation before input to symmetric range –31.75 to +31.75)

Interfaces

- Separate clocks on each interface to ease integration
- Wide data interfaces on input and output with configurable support for 1, 2, or 4 lanes
- Ability to specify number of LLR values on each lane on either a block-by-block basis, or transfer basis
- Separate inputs to specify control parameters and receive status output on a block-by-block basis

PS Clock Subsystem

Introduction

The PS clocking system generates clocks for the processors, peripherals, interconnect, and other system elements. There are five system PLLs to generate high-frequency signals that are used as clock sources for the several dozen clock generators in the LPD and FPD.

System PLL Clock Units

Two system PLL clock units are in the LPD and three are in the FPD power domain. Each PLL unit has two clock dividers on its output; one in the LPD and one in the FPD. These clock dividers can provide two different clocking frequencies from one PLL (in the two clock domains). The PLL output and clock frequencies are specified in the *Zynq UltraScale+ MPSoC Data Sheet: DC and AC Switching Characteristics* (DS925) [Ref 2]. The maximum clock output frequencies are somewhat lower for clocks crossing power domains.

Each system PLL unit has a suggested usage, but the individual clock generators can select from one of the three PLL clocks routed to it as defined by the registers listed in the Clock Generator Control Registers section.

The system PLL units reside in the LPD and FPD power domains:.

- Low power domain system PLLs:
    - I/O PLL (IOPLL): provides clocks for all low speed peripherals and part of the interconnect.
    - RPU PLL (RPLL): provides clocks for the RPU MPCore and part of the interconnect.
- Full-power domain system PLLs:
    - APU PLL (APLL): provides clocks for the APU MPCore clock and part of the interconnect.
    - Video PLL (VPLL): provides clocks for the video I/O.
    - DDR PLL (DPLL): provides clocks for the DDR controller and part of the interconnect.

Note: There are six DDR PHY PLLs in the DDR memory controller that are used for the DRAM address and control output signals, and the data and ECC byte lanes. The PHY PLLs are dedicated to the DDR I/O interface and cannot be used as a clock source for the clock generator units.

The five system PLLs (RPLL, IOPLL, APLL, DPLL, and VPLL) are powered by one voltage supply, V_{CC_PSPLL}. The six DDR PLLs for the DRAM address/control and I/O byte lanes are powered by $V_{CC\_PSDDR\_PLL}$.

It is possible to use the PLL output from one power domain in the other power domain. The IOPLL and RPLL output in the low-power domain (LPD) can be an input to the full-power domain (FPD) using a separate 6-bit programmable divider. Similarly, the APLL, DPLL, or VPLL output clock can be an input clock to individual 6-bit programmable dividers in the LPD. The 6-bit programmable divider are controlled using (for example) the crf_apb.APLL_TO_LPD_CTRL register.

Clock Generators

Clock generators are needed for processors, peripherals, interconnect, and other system elements in the LPD, FPD, and PL. The five system PLLs generate high-frequency signals that are used as sources for the several dozen clock generators. Three of the PLL outputs are routed to each clock generator.

The basic and the two special clock generator architectures are as follows:

- APU MPCore (unique).
- DDR memory controller (unique).
- RPU MPCore (basic clock generator with one divider and two clock enables).
- Basic clock generator (with two dividers).
- Basic clock generator (with one divider).

Basic Clock Generator Unit

The majority of the clock generators have the same basic circuit as shown in Figure 37-4 and use the same basic programming model shown in the Clock Generator Programming Example section. The programming models for these system and peripheral units are similar. Some of the basic clock generators have multiple clock enables controlling sub-elements of a subsystem.

Clock System Overview

The PS subsystem clock unit has five PLL clock units and many clock generators for system elements. The landscape of the system clock units are represented in Figure 37-1. Clock generators have either one or two programmable divider units. Some clock generators have more than one clock-active control. The RTC and PMU have standalone clock generators.

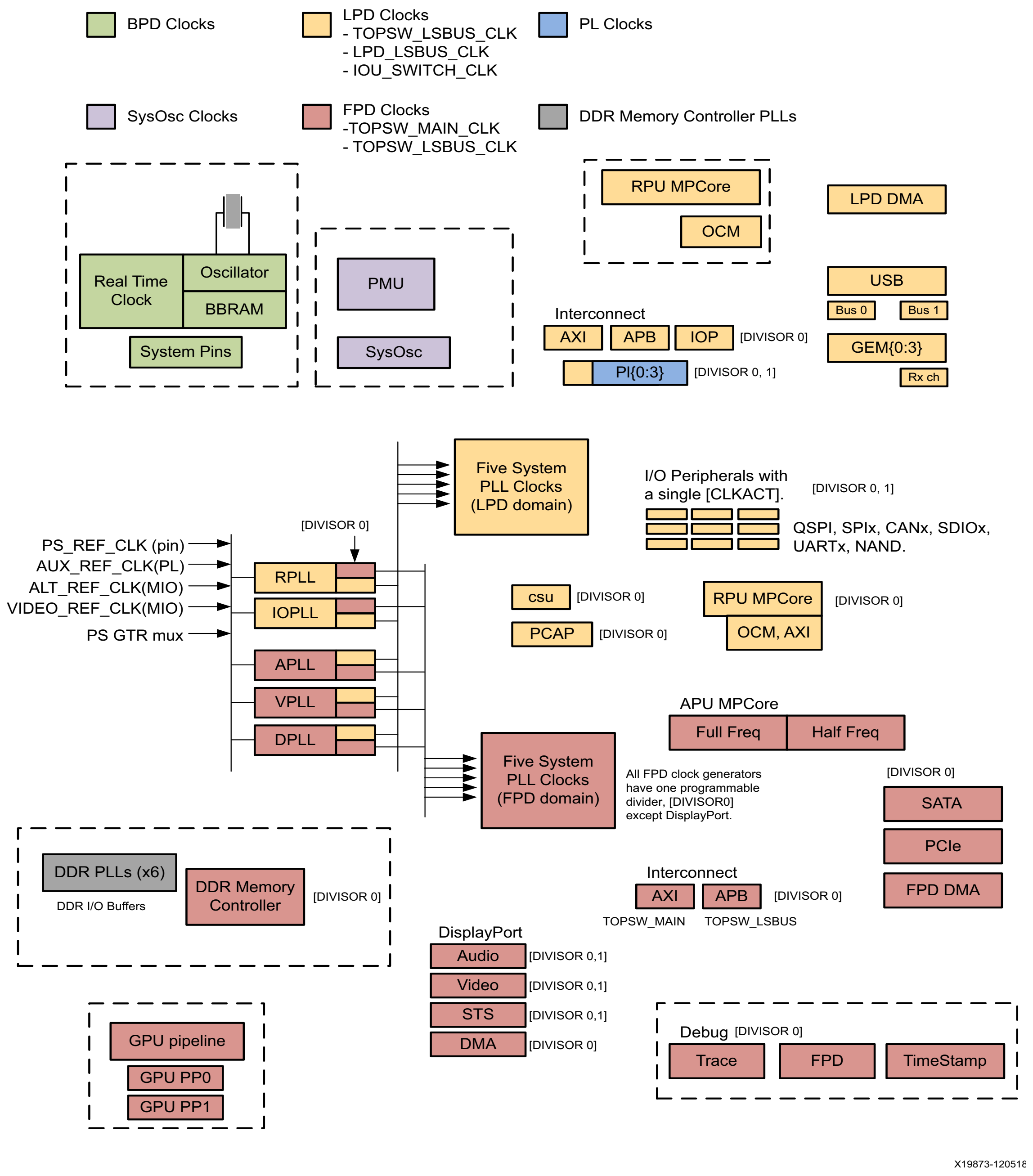


Figure 37-1: **System Clocks Block Diagram**

Real-time Clock Domain

The low-power 32 KHz clock unit is self-contained and used by the real-time clock (RTC) to maintain an accurate time base. The RTC is described in Chapter 7, Real Time Clock.

PMU Clock Domain

The PMU is clocked by the SysOsc clock unit. This clock is generated by an on-chip ring oscillator circuit that is trimmed during production.

Clock Monitor

The clock monitor measures the frequency of one clock using another clock as a reference. This monitor does not monitor duty cycle, jitter, or quality. The clock monitor uses a reference clock that counts for a predetermined number of cycles set by the control register. During that time, another counter in the second clock domain is counting. When the reference clock counter is done, the second clock domain is signaled to stop counting and then compares its count value to two pre-programmed registers. If the counted value is within the bounds of the registers, then the clock being measured is within tolerable parameters. If it is not, then an interrupt is provided to the interrupt controller.

Glitch-Free Clock Controls

The clock multiplexers and enable controls within the PS clock subsystem includes circuitry to provide glitch-free output clocks that satisfy minimum high and low pulse widths. Multiplexers and enables in other parts of the system do not usually include the glitch-free control feature.

PL Clock Throttle

The PL clock generator has an optional feature to limit the number of clocks that are generated from a start command. A PL fabric input can also be used to stop the PL output clock from the PL clock generator.

System PLL Units

The five system PLLs provide a 750 to 1600 MHz clock to the clock generators. The frequency and jitter specifications for the APLL, DPLL, RPLL, IOPLL, and VPLL system PLLs are in the *Zynq UltraScale+ MPSoC Data Sheet: DC and AC Switching Characteristics* (DS925) [Ref 2].

PLL Source Clocks

The source clock for the PLL clock units is selected from one of five sources (see Figure 37-2). All of these source clocks are inputs to each PLLs clock unit.

- PS_REF_CLK (device pin, normal source).
- ALT_REF_CLK (one of two MIO pins).
- VIDEO_REF_CLK (one of two MIO pins).
- AUX_REF_CLK (PL fabric source).
- GTR_REF_CLK (multiplexer output from GTR serial unit).

The GTR_REF_CLK clock is rarely used but can be sourced from a PS GTR peripheral selected using the SIOU.CRX_CTRL [refclk_sel] bit field. The GTR clock specifications are listed in *Zynq UltraScale+ MPSoC Data Sheet: DC and AC Switching Characteristics* (DS925) [Ref 2].

- 0: PCIe/USB
- 1: DisplayPort
- 2: SATA
- 3: SGMII

The PS_REF_CLK clock is always used for booting the system and is the default clock source for the system PLLs. After the system boots, the other reference clock sources can be selected to drive any of the PLL system clock units.

The PL clock throttle function is described in the Programmable Clock PL Throttle section.

Note: Actively used PLL units must be put into bypass mode (xPLL_CTRL [BYPASS]) before reprogramming the clock frequency. After programming, wait for the PLL_STATUS [xPLL_LOCK] status bit to assert, then select the PLL output by disabling bypass.

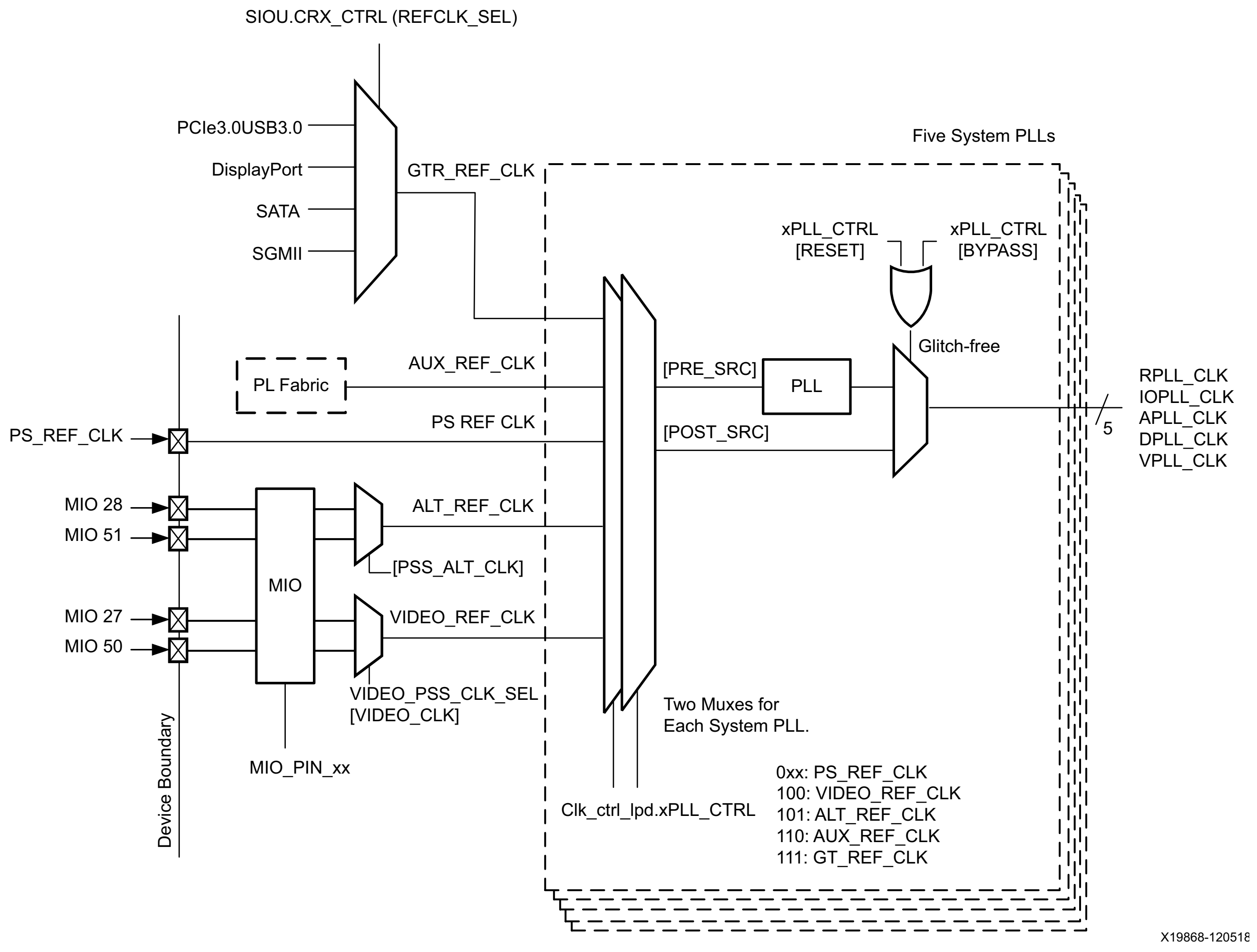


Figure 37-2: **PLL Clock Unit Block Diagram**

Note: Not all hardware features in this reference manual may be implemented in Xilinx design tools.

Power Domain Crossing of PLL Clocks

Each system PLL unit has a clock divider in its own power domain and one in the other power domain. Both sets of dividers are represented as boxes in Figure 37-1 and the controls for the power-domain crossed clocks are shown in Figure 37-3.

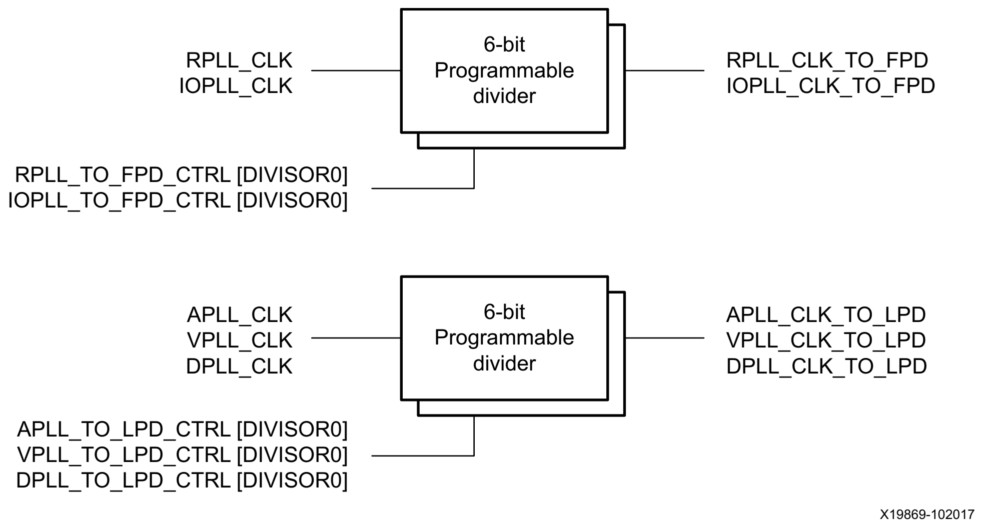

RPLL_CLK
IOPLL_CLK
6-bit Programmable divider
RPLL_CLK_TO_FPD
IOPLL_CLK_TO_FPD
RPLL_TO_FPD_CTRL [DIVISOR0]
IOPLL_TO_FPD_CTRL [DIVISOR0]
APLL_CLK
VPLL_CLK
DPLL_CLK
6-bit Programmable divider
APLL_CLK_TO_LPD
VPLL_CLK_TO_LPD
DPLL_CLK_TO_LPD
APLL_TO_LPD_CTRL [DIVISOR0]
VPLL_TO_LPD_CTRL [DIVISOR0]
DPLL_TO_LPD_CTRL [DIVISOR0]
X19869-102017

Figure 37-3: **System PLL Clock Power Domain Crossing**

Basic Clock Generators

The basic clock generator is shown in Figure 37-4. The architecture is used for all system elements except the special clock generators for the APU MPCore (Figure 37-5) and the DDR memory controller (Figure 37-6). Variations of the basic clock generator include the number of divider units, the three specific clock sources provided to the clock generator, and the number of clock active controls.

The system PLL clock source is selected using the CRx_APB.xPLL_CTRL [SRCSEL] bit field. The PLL is in bypass or active mode. The selected PLL output goes to a 6-bit clock divider in its native power domain and a 6-bit divider in the other PS power domain.

The CSU BootROM (CBR) and PMU pre-boot ROM code modifies several clock control registers, including divisor values and clock enables. The modifications are described in the System PLL Control Registers and Clock Generator Control Registers sections.

IMPORTANT: *All clock generator input multiplexers in Figure 37-4 have a default input clock selection of 0. The selected source clock is listed in the register overview tables. Before downloading the first stage boot loader (FSBL), all PLLs except for IOPLL and DPLL are held in reset. The system PLLs are programmed by the FSBL and system software for the application.*

Note: The clock multiplexers within the clock subsystem (system PLLs, basic, and special clock generators) include de-glitching logic to enable changes while the system is operating. However, clock multiplexers out in the system (e.g., I/O controllers) do not generally include this logic. In these cases, clocks might need to be stopped before switching, or the controller needs to be held in reset while switching. Refer to individual cases.

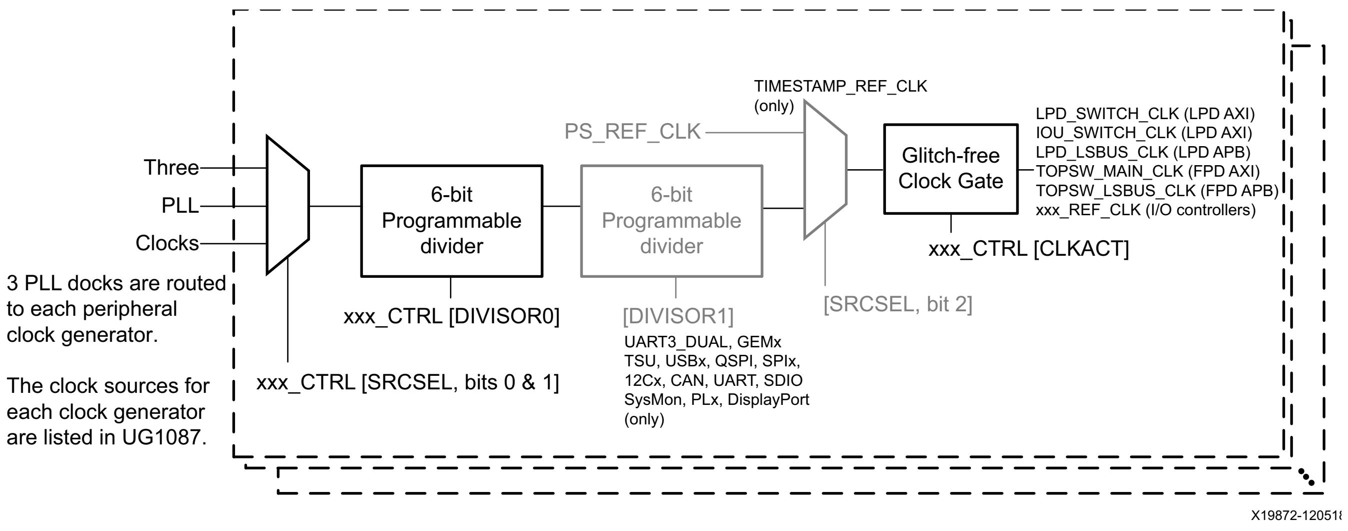

Three
PLL
Clocks
3 PLL docks are routed to each peripheral clock generator.
The clock sources for each clock generator are listed in UG1087.
6-bit Programmable divider
xxx_CTRL [DIVISOR0]
xxx_CTRL [SRCSEL, bits 0 & 1]
PS_REF_CLK
6-bit Programmable divider
[DIVISOR1]
UART3_DUAL, GEMx TSU, USBx, QSPI, SPIx, 12Cx, CAN, UART, SDIO SysMon, PLx, DisplayPort (only)
TIMESTAMP_REF_CLK (only)
[SRCSEL, bit 2]
Glitch-free Clock Gate
xxx_CTRL [CLKACT]
LPD_SWITCH_CLK (LPD AXI)
IOU_SWITCH_CLK (LPD AXI)
LPD_LSBUS_CLK (LPD APB)
TOPSW_MAIN_CLK (FPD AXI)
TOPSW_LSBUS_CLK (FPD APB)
xxx_REF_CLK (I/O controllers)
X19872-120518

Figure 37-4: **Basic LPD and FPD Clock Generator Block Diagram**

There are many basic clock generators. Their control registers are listed in the Clock Generator Control Registers section.

- Interconnect
- RPU MPCore
- LPD and FPD DMA units
- LPD, FPD, Trace, and Timestamp Debug
- PS SYSMON unit
- PL
- DisplayPort Video, Audio, STC, DMA
- GPU
- PCIe
- SATA
- IOP Peripherals (GEM, USB, UART, SPI, Quad-SPI, NAND, SDIO, CAN, I2C)

Several of these clock generators are described in more detail.

Interconnect Clock Generators

There are five clock generators for the AMBA interconnect structure. Each clock generator has a similar architecture with one divisor and one clock activate control as shown in Figure 37-4. The control registers are listed in Table 37-4.

- LPD_LSBUS_CLK: clocks the slave switches and the APB interfaces in the LPD, including LPD_SLCR, XMPU, CSU DMA, LPD DMA, DAP, eFUSE, RPU, IPI, OCM, and APM.
- IOU_SWITCH_CLK: clocks the AXI interfaces for the IOP peripherals in LPD.
- LPD_SWITCH_CLK: clocks the AXI interfaces for the non-IOP interfaces in LPD.
- TOPSW_LSBUS_CLK: clocks the slave switches and the APB interfaces in the FPD.
- TOPSW_MAIN_CLK: clocks the AXI interfaces in the FPD including the CCI, DDR ports, SMMU ports, and the SIOU DMA masters. Also clocks the PS-PL AXI interfaces on the FPD side.

RPU MPCore Clock Generator

The RPU MPCore reference clock reference is based on the basic clock generator design with one divider and two clock enables. The extra clock enable is for the subsystem.

The clock gating behavior of the RPU clock is controlled by the clock and reset module in the low power domain region. When the clock and reset module turns the clock gate enable on, the RPU and its GIC receive the clock. When clock gating is turned off, the entire RPU subsystem is not clocked.

Debug Clock Generators

There is a debug clock generator for each power domain, LPD and FPD and a separate clock generators for the trace buffer and timestamp unit in the FPD.

FPD Debug Clock

The FPD debug clock is controlled by the DBG_FPD_CTRL register and is used by the CoreSight-associated debug logic within the FPD.

LPD Debug Clock

The LPD debug clock is controlled by the DBG_LPD_CTRL register and is used by the CoreSight-associated debug logic within the LPD.

Trace Debug Clock

The trace debug clock is controlled by the clk_ctrl_fpd.DBG_TRACE_CTRL register with a single divisor. The clock should be programmed to twice the frequency of the desired trace port clock because it is used to derive the trace port clock. The frequency of trace port clock must be fast enough to allow the trace port to keep up with the amount of data being traced.

Timestamp Debug Clock

The timestamp debug unit is clocked by its own clock generator.

PL Clock Generators

There are four clock outputs to the PL from independent clock generators. The PL clock generators are based on the generic architecture with two clock dividers each plus the clock throttle feature.

The clocks are individually controlled and are asynchronous to each other and all other clocks.

Programmable Clock PL Throttle

The four generated clocks for the PL (PL_REF_CLKx) have clock throttling logic associated with each clock. By default, clock throttling is off and there is continuous clock output. For each of the clocks throttle logic, there are two registers.

- PL0_THR_CTRL: PL clock threshold control and status.
- PL0_THR_CNT: PL clock threshold count value.

The throttle behavior is controlled by indicating a desired number of clock pulses by writing a 16-bit value to the PL0_THR_CNT register. For example, if PL0_THR_CNT is set to 0, then the output is free running. If there is a programmed value, then the output is clocked using the number indicated in this register.

The output clock counting can be started or triggered by writing to the CPU_START bit of the PL0_THR_CTRL register. The output clock can also be halted by the PLx_THR_STOP signal from the PL logic when the counting mechanism is turned on. This pin stops the PL clock during PL logic debug.

The register PL0_THR_CTRL[CURR_VALUE] counts the amount of clocks produced since CPU_START was initiated.

DisplayPort Clock Generators

The video reference clock, dp_video_ref_clk, is based on the basic clock generator design with one divider and one clock enable as shown in Figure 37-3. The input video clock is typically a 27 MHz base clock. The output clock frequency generated from the video clock generation block can typically be 27 MHz, 81 MHz, 135 MHz, or 270 MHz depending on the data rate or the fractional divide values of the PLL can be configured to generate the unique frequencies needed for the DisplayPort controller. The fractional mode of the PLL can be used to generate a specific video clock frequency.

Similarly, the audio reference clock, dp_audio_ref_clk, is generated using the basic clock generator design but, with two divisors and one clock enable.

GPU Clock Generator

The GPU reference clock is used to clock the main logic of the GPU. Inside the GPU, the GPU reference clock is distributed to the GPU and to the pixel processors.

SATA Clock Generator

The SATA reference clock is used to clock the internal logic of the SATA controller. The AXI-side clock generation (determined by the crf_apb.SATA_REF_CTRL register) follows the generic clock generation model with one divider. The SerDes interface clock for SATA is generated using the PS-GTR reference clock option. For more details, see Chapter 29, PS-GTR Transceivers.

Special Clock Generators

There are two special clock generator architectures:

- APU MPCore
- DDR Memory Controller

APU MPCore Clock Generator

The APU MPCore uses two related clocks: the main, full-frequency APU clock and the half-speed clock. The clocks are shown in Figure 37-5.

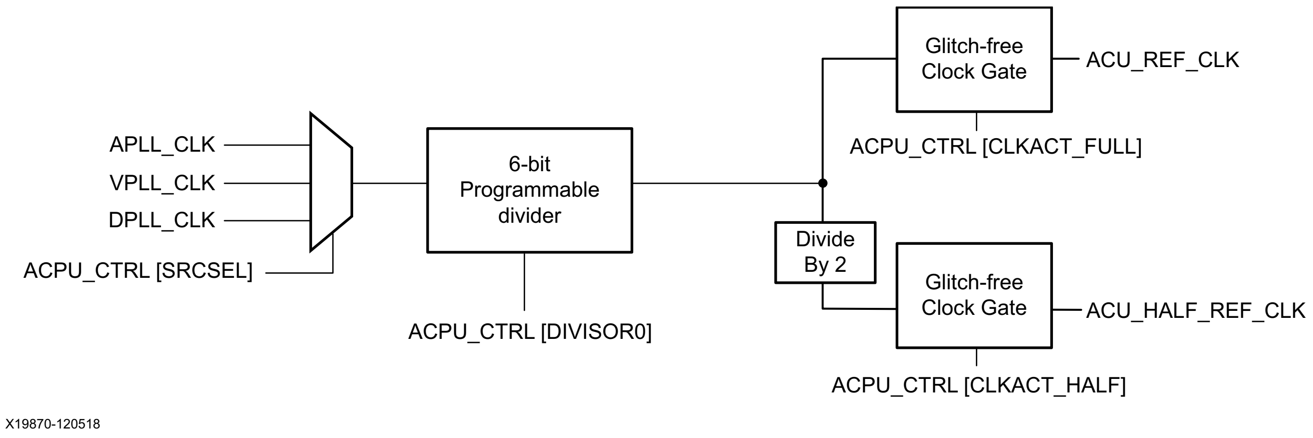


Figure 37-5: **APU MPCore Clock Generator**

DDR Memory Controller Clock Generator

The DDR clock is used for the majority of the DDR memory controller logic. The clock generator has one 6-bit divider that connects to either the DPLL or VPLL. The DDR memory subsystem also includes six PLLs for the DRAM I/O buffers that are described in Chapter 17, DDR Memory Controller.

The DDR memory controller clock generator is shown in Figure 37-6.

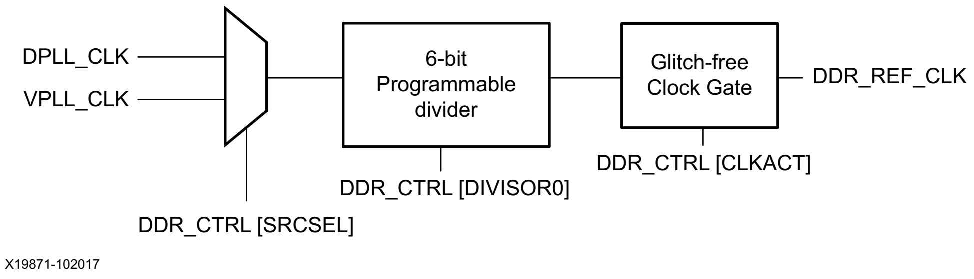


Figure 37-6: **DDR Memory Controller Clock Generator**

Programming Examples

The programming sequence for the PLL units require careful consideration for the oscillator based on the desired output frequency controlled by [FBDIV] and other parameters. Each PLL unit has three control registers:

- xPLL_CTRL [RESET, BYPASS, FBDIV, DIV2, PRE_SRC, POST_SRC]
- xPLL_CFG [RES, CP, LFHF, LOCK_CNT, LOCK_DLY]
- xPLL_FRAC_CFG [DATA, ENABLED]

Helper data is programmed into the xPLL_CFG registers. See PLL Integer Divide Helper Data Table for information on the helper data in the integer and fractional modes.

System PLL Operation

The voltage controlled oscillator (VCO) in the PLL synthesizes the output frequency based on the feedback multiplier. The VCO supports both fractional and integer multipliers. The fractional mode is enabled by setting xPLL_FRAC_CFG[ENABLED] to 1.

The VCO output frequency (F_{VCO}) is determined using the following equation.

$$F_{VCO} = F_{REFCLK} \times M.F$$

In this equation, the F_{REFCLK} is an input reference clock frequency, M is the integer part of the multiplier value, and F is the fractional part.

The output frequency (F_{CLKOUT}), after the divider stage, is determined by the following equation.

$$F_{CLKOUT} = F_{VCO}/O$$

In this equation, O is the output divider that can be set to 1 or 2.

Jitter Considerations

PLL jitter performance is better in integer mode. Whenever possible, always use integer mode. The fractional modulus of the PLL feedback divide is 16 bits wide, hence, 0.F can be set to any value equal to $n/2^{16}$, where n = 1, 2, ..., 2^{16}-1. In fractional mode, minimize jitter by generating the highest possible VCO frequency that is with its operating range.

Video Clock Example

To generate a 296.703 MHz video clock from a 27 MHz source, use the following parameters. The VCO frequency will be 2373.6 MHz and the VPLL clock output will be 1186.8 MHz.

- Program the multiplier. Set M = 87.
- Program the fractional modulus. Set F = 59775.
- Program VPLL to divide by 2. Set CRF_APB.VPLL_CTRL [DIV2] = 1.
- Program video clock generator to divide by 4. Set CRF_APB..DP_VIDEO_REF_CTRL [DIVISOR0] = 4.

Clock Source Programming Example

The programming steps to use video_ref_clk as the clock source for IOPLL are used in this example.

1. Program the PLL into bypass by setting IOPLL_CTRL[BYPASS] = `1`.
2. Assert the reset to IOPLL by setting IOPLL_CTRL[RESET] = `1`.
3. Set IOPLL_CTRL[PRE_SRC] = `100b`, the VIDEO_REF_CLK.
4. Deassert the IOPLL reset by configuring IOPLL_CTRL[RESET] = `0`.
5. Check for PLL lock by checking PLL_STATUS[IOPLL_LOCK] = `1`.
6. Disable bypass mode by setting IOPLL_CTRL[BYPASS] = `0`.

IMPORTANT: *The following clocks should never be made inactive: LPD_SWITCH_CLK, LPD_LSB_CLK, TOPSW_MAIN_CLK, and TOPSW_LSBUS_CLK. Whenever changing a CLK source, ensure that any downstream clocks are prevented from exceeding their maximum clock frequency.*

Integer Multiply and Divide Programming Example

This example assumes the input PS_REF_CLK frequency is 50 MHz, the [FBDIV] value is 40, and the output divider is turned on. The output clock is calculated to be 50 MHz x 40/2 = 1000 MHz. For a new frequency of 1600 MHz, the [FBDIV] value is switched to 32 and the output divider is turned off. This example uses the APLL.

Note: Before reprogramming the PLL clock output frequency, check that the downstream clocks are in a safe state before releasing. For instance, if the APU DIVISOR is set to 2.

1. Program the new FBDIV, CLKOUT value (do not modify other values in the APLL_CTRL register).

   Set APLL_CTRL = 0000_2000h: [DIV2] = 0, [FBDIV] = 20h.

2. Program the helper data for APLL_CFG using the helper data in Table 37-1.
3. Program the bypass.

   Set APLL_CTRL = 0000_2008h: [BYPASS] = 1.

4. Assert reset. This is when the new data is actually captured into the PLL.

   Set APLL_CTRL = 0000_2009h: [RESET] = 1.

5. Deassert reset.

   Set APLL_CTRL = 0000_2008h: [RESET] = 0.

6. Check for LOCK. Wait until: PLL_STATUS [APLL_LOCK] = 1
7. Deassert bypass.

   Set APLL_CTRL = 0000_2000h: [BYPASS] = 20h.

The PLL output clock is set to 1600 MHz.

Fractional Multiply and Divide Programming Example

The following example assumes that the input clock to the PLL is PS_REF_CLK at 50 MHz, the FBDIV value is 32, and the output divider is turned on. The output frequency is 1600 MHz. To change to the VIDEO_REF_CLK, which is at 27 MHz and produce a final frequency of 1090.125 MHz, the FBDIV divider must be 40.375. Because 1090.125 MHz is below the VCO operating range, a value of 80.75 is required and the div2 is used to produce 1090.125 MHz. This example uses the VPLL.

1. Program the bypass mode by configuring VPLL_CTRL[BYPASS] = 1.
2. Program the new FBDIV, CLKOUT, and PRE_SRC values.
   VPLL_CTRL[DIV2] = 1

VPLL_CTRL[FBDIV] = 50h
VPLL_CTRL[PRE_SRC] = 100b: VIDEO_REF_CLK

3. Program the VPLL_CFG register. Refer to the VPLL_CFG programming helper data in Table 37-1.
4. Program the fractional data. A value of 0.75 = 0Bh.
   VPLL_FRAC_CFG[ENABLED] = 1
   VPLL_FRAC_CFG[DATA] = C000h
   VPLL_FRAC_CFG = 8000_C000h
5. Assert the reset. This is required when the new data is actually captured into the PLL.
   VPLL_CTRL[RESET] = 1
6. Deassert the reset.
   VPLL_CTRL[RESET] = 0
7. Check for a locked signal.
8. Wait until: PLL_STATUS[VPLL_LOCK] = 1
9. Deassert the bypass.
   VPLL_CTRL[BYPASS] = 0

Similar steps are followed to program DPLL, RPLL, IOPLL, and APLL.

Clock Generator Programming Example

This example shows the programming steps to enable an LPD main switch clock with the IOPLL clock and to divide the result by four.

1. Set CRL_APB.LPD_SWITCH_CTRL[CLKACT] = 1.
2. Set CRL_APB.LPD_SWITCH_CTRL[SRCSEL] = 010b.
3. Program divider by writing to CRL_APB.LPD_SWITCH_CTRL[DIVISOR0] = 04h.

Clock Monitor Programming Example

This example shows the programming steps to program the clock monitors. The example assumes that the PS_REF_CLK is 50 MHz and the APB LPD bus clock (LPD_LSB_CLK) is 100 MHz. All registers are in the CRL_APB register set. The clock sources are listed with the definitions of the CHKRx_CTRL registers in the *Zynq UltraScale+ MPSoC Register Reference* (UG1087) [Ref 4].

1. Program the clock sources:

   Set CHKRx_CTRL [clka_mux_ctrl] = 011b (LPD_LSBUS_CLK).

   Set CHKRx_CTRL [clkb_mux_ctrl] = 0 (PS_REF_CLK).

2. Program the counter values:

   Set CHKRx_CLKB_CNT [value] = 0000_0000h.

   Set CHKRx_CLKA_UPPER [thrshld] = 0001_028Ah.

   Set CHKRx_CLKA_LOWER [thrshld} = 0000_FD76h.

3. Prime the pump:

   Set CHKRx_CTRL [enable] = 1.

4. Start the clock monitor:

   Set CHKRx_CTRL [start_single] = 1.

PLL Integer Divide Helper Data Table

For each unique value multiplier value, program the PLLs using registers in the CRL_APB and CRF_APB register sets (LPD and FPD). Each of the five PLLs have a set of integer programming parameters:

- {CRL, CRF}_APB.xPLL_CFG[CP]
- {CRL, CRF}_APB.xPLL_CFG[RES]
- {CRL, CRF}_APB.xPLL_CFG[LFHF]
- {CRL, CRF}_APB.xPLL_CFG[LOCK_DLY]
- {CRL, CRF}_APB.xPLL_CFG[LOCK_CNT]

Table 37-1 provides the PLL configuration register programming values when the PLL is in integer mode. The frequency of the VCO must stay within the range specified in *Zynq UltraScale+ MPSoC Data Sheet: DC and AC Switching Characteristics* (DS925) [Ref 2].

Table 37-1: **PLL Integer Feedback Divider Helper Data Values**

FBDIV	CP	RES	LFHF	LOCK_DLY	LOCK_CNT
25	3	10	3	63	1000
26	3	10	3	63	1000
27	4	6	3	63	1000
28	4	6	3	63	1000
29	4	6	3	63	1000
30	4	6	3	63	1000
31	6	1	3	63	1000
32	6	1	3	63	1000
33	4	10	3	63	1000
34	5	6	3	63	1000
35	5	6	3	63	1000
36	5	6	3	63	1000
37	5	6	3	63	1000
38	5	6	3	63	975
39	3	12	3	63	950
40	3	12	3	63	925
41	3	12	3	63	900
42	3	12	3	63	875
43	3	12	3	63	850

Table 37-1: **PLL Integer Feedback Divider Helper Data Values *(Cont'd)***

FBDIV	CP	RES	LFHF	LOCK_DLY	LOCK_CNT
44	3	12	3	63	850
45	3	12	3	63	825
46	3	12	3	63	800
47	3	12	3	63	775
48	3	12	3	63	775
49	3	12	3	63	750
50	3	12	3	63	750
51	3	2	3	63	725
52	3	2	3	63	700
53	3	2	3	63	700
54	3	2	3	63	675
55	3	2	3	63	675
56	3	2	3	63	650
57	3	2	3	63	650
58	3	2	3	63	625
59	3	2	3	63	625
60	3	2	3	63	625
61 to 82	3	2	3	63	600
83 to 102	4	2	3	63	600
103	5	2	3	63	600
104	5	2	3	63	600
105	5	2	3	63	600
106	5	2	3	63	600
107 to 125	3	4	3	63	600

Register Overview

There are several register sets used to control system and peripheral clocks.

Table 37-2: **Clock Configuration Registers**

Register Type	Register Name	Description
Low-Power Domain (LPD)		
Interrupt and error configuration (CRL_APB)	ERR_CTRL	Register address decode error on APB slave interface (SLVERR).
	IR_STATUS	Interrupt status register for interrupt. This is a sticky register that holds the value of the interrupt until cleared by a value of 1.
	IR_MASK	Interrupt mask register for interrupt. This is a read-only location and can be automatically altered by either the IDR or the IER.
	IR_ENABLE	Interrupt enable register. A write of 1 to this location unmasks the interrupt. (IMR: 0).
	IR_DISABLE	Interrupt disable register. A write of one to this location masks the interrupt (IMR: 1).
PLL configuration (CRL_APB)	IOPLL_CTRL	IOPLL clock control.
	IOPLL_CFG	IOPLL configuration.
	IOPLL_FRAC_CFG	IOPLL fractional control.
	RPLL_CTRL	RPLL clock control.
	RPLL_CFG	RPLL configuration.
	RPLL_FRAC_CFG	RPLL fractional control.
	PLL_STATUS	IOPLL and RPLL status.
	IOPLL_TO_FPD_CTRL	Control for a clock that is generated in the IOPLL targeting LPD, but used in the FPD as a clock source for the peripheral clock multiplexer.
	RPLL_TO_FPD_CTRL	Control for a clock that is generated in the RPU PLL in LPD, but used in the FPD as a clock source for the peripheral clock multiplexer.
Clock monitor (CRL_APB)	CLKMON_STATUS	Interrupt status. This is a sticky register that holds the value of the interrupt until cleared by a value of `1`.
	CLKMON_MASK	Interrupt mask. This is a read-only location and can be automatically altered by either the IDR or the IER.
	CLKMON_ENABLE	Interrupt enable register. A write of `1` to this location unmasks the interrupt.
	CLKMON_DISABLE	Interrupt disable register. A write of `1` to this location masks the interrupt.
	CLKMON_TRIGGER	Interrupt trigger register. A write of `1` to this location sets the interrupt status register related to this interrupt.

Table 37-2: **Clock Configuration Registers *(Cont'd)***

Register Type	Register Name	Description
Full-Power Domain (FPD)		
System controls (CRF_APB)	ERR_CTRL	Register address decode error on APB slave interface (SLVERR).
	IR_STATUS	Interrupt status register for intrN. This is a sticky register that holds the value of the interrupt until cleared by a value of 1.
	IR_MASK	Interrupt mask register for intrN. This is a read-only location and can be automatically altered by either the IDR or the IER.
	IR_ENABLE	Interrupt enable register. A write of 1 to this location unmasks the interrupt. (IMR: 0).
	IR_DISABLE	Interrupt disable register. A write of one to this location masks the interrupt (IMR: 1).
PLL configuration (CRF_APB)	APLL_CTRL	APLL clock control.
	APLL_CFG	APLL configuration.
	APLL_FRAC_CFG	APLL fractional control.
	DPLL_CTRL	DPLL clock control.
	DPLL_CFG	DPLL configuration.
	DPLL_FRAC_CFG	DPLL fractional control.
	VPLL_CTRL	VPLL clock control.
	VPLL_CFG	VPLL configuration.
	VPLL_FRAC_CFG	VPLL fractional control.
	PLL_STATUS	APLL, DPLL, VPLL status.

System PLL Control Registers

The PLL control registers are in the CRL_APB (LPD) and CRF_APB (FPD) register sets.

Table 37-3: **System PLL Clock Control Register Settings**

Register Name	Reset value, Address offset (LPD, FPD)	Register Parameter	Reset State	Pre-FSBL	Comments
RPLL_CTRL	`0001_2C09 h`, LPD 0x030	[RESET] [BYPASS] [FBDIV] [DIV2] [PRE_SRC] [POST_SRC]	Held in reset. Bypass enabled. `2C h`. Divide by 2. PS_REF_CLK. PS_REF_CLK.		
IOPLL_CTRL	`0001_3200 h`, LPD 0x020	[RESET] [BYPASS] [FBDIV] [DIV2] [PRE_SRC] [POST_SRC]	Held in reset. Bypass enabled. `2C h`. Divide by 2. PS_REF_CLK. PS_REF_CLK.	Released from reset.	
APLL_CTRL	`0001_2C09 h`, FPD 0x020	[RESET] [BYPASS] [FBDIV] [DIV2] [PRE_SRC] [POST_SRC]	Held in reset. Bypass enabled. `2C h`. Divide by 2. PS_REF_CLK. PS_REF_CLK.		
DPLL_CTRL	`0000_2C09 h`, FPD 0x02C.	[RESET] [BYPASS] [FBDIV] [DIV2] [PRE_SRC] [POST_SRC]	Held in reset. Bypass enabled. `2C h`. Pass-through. PS_REF_CLK. PS_REF_CLK.	Released from reset.	
VPLL_CTRL	`0000_2809 h`, FPD 0x038.	[RESET] [BYPASS] [FBDIV] [DIV2] [PRE_SRC] [POST_SRC]	Held in reset. Bypass enabled. `28 h`. Divide by 2. PS_REF_CLK. PS_REF_CLK.		
IOPLL_TO_FPD_CTRL	`0000_0400 h`, LPD 0x044.	[DIVISOR0]	`04 h`		

Table 37-3: **System PLL Clock Control Register Settings *(Cont'd)***

Register Name	Reset value, Address offset (LPD, FPD)	Register Parameter	Reset State	Pre-FSBL	Comments
RPLL_TO_FPD_CTRL	0000_0400 h, LPD 0x048.	[DIVISOR0]	04 h		
APLL_TO_LPD_CTRL	0000_0400 h, FPD 0x048.	[DIVISOR0]	04 h		
DPLL_TO_LPD_CTRL	0000_0400 h, FPD 0x04C.	[DIVISOR0]	04 h		
VPLL_TO_LPD_CTRL	0000_0400 h, FPD 0x050.	[DIVISOR0]	04 h		

Clock Generator Control Registers

The clock generator control registers are divided into the following tables:

- AMBA interconnect clocks
- Processors, DDR, and DMA clocks
- LPD and FPD system clocks
- LPD peripheral clocks
- FPD peripheral clocks

Table 37-4: **AMBA Interconnect Clock Control**

Register	Reset Value, Register Set, Address Offset	Register Parameter	Reset State	Pre-FSBL	Clock Source Options
LPD_LSBUS_CTRL (APB)	0100_1800 h, LPD 0x0AC	[SRCSEL]: [DIVISOR0]: [CLKACT]:	RPLL. 18 h. Enabled.		RPLL, IOPLL, or DPLL_CLK_TO_LPD.
IOU_SWITCH_CTRL (AXI LPD)	0000_1500 h, LPD 0x09C	[SRCSEL]: [DIVISOR0]: [CLKACT]:	RPLL. 15 h. Disabled.		RPLL, IOPLL, or DPLL_CLK_TO_LPD.
LPD_SWITCH_CTRL (AXI LPD)	0100_0500 h, LPD 0x0A8	[SRCSEL]: [DIVISOR0]: [CLKACT]:	RPLL. 05 h. Enabled.		RPLL, IOPLL, or DPLL_CLK_TO_LPD.

Table 37-4: **AMBA Interconnect Clock Control *(Cont'd)***

Register	Reset Value, Register Set, Address Offset	Register Parameter	Reset State	Pre-FSBL	Clock Source Options
TOPSW_LSBUS_CTRL (APB LPD)	0100_0800 h, FPD 0x0C4	[SRCSEL]: [DIVISOR0]: [CLKACT]:	APLL. 08 h. Enabled.		APLL, VPLL, DPLL.
TOPSW_MAIN_CTRL (AXI FPD)	0100_0400 h, FPD 0x0C0	[SRCSEL]: [DIVISOR0]: [CLKACT]:	APLL. 04 h. Enabled.		APLL, VPLL, DPLL.

Table 37-5: **Processors, DDR, and DMA Clock Control**

Control Register	Reset Value, Register Set, Address Offset	Register Parameter	Reset State	Pre-FSBL	Comments
CPU_R5_CTRL (RPU MPCore)	0200_0600 h, 0x090	[SRCSEL] [DIVISOR0] [CLKACT] [CLKACT_CORE]	RPLL. 06 h. Enabled. Enabled.		
ACPU_CTRL (APU MPCore)	0300_0400 h, 0x060	[SRCSEL] [DIVISOR0] [CLKACT_FULL] [CLKACT_HALF]	APLL. 04 h. Enabled. Enabled.		
CSU_PLL_CTRL	0100_1500 h, 0x0A0	[SRCSEL] [DIVISOR0] [CLKACT]	IOPLL. 15 h. Enabled.		
DDR_CTRL	0100_0500 h, 0x0A0	[SRCSEL] [DIVISOR0]	DPLL. 05 h.		
FPD_DMA_REF_CTRL	0100_0500 h, 0x0B8	[SRCSEL] [DIVISOR0] [CLKACT]	APLL. 05 h. Enabled.		
LPD_DMA_REF_CTRL	0000_2000 h, 0x0B8	[SRCSEL] [DIVISOR0] [CLKACT]	RPLL. 20 h. Disabled.		

Table 37-6: **System Clock Control**

Register Name	Reset Value, Address Offset	Register Parameter	Reset State	Pre-FSBL	Comments
DBG_LPD_CTRL	`0100_2000h`, LPD 0x068	[SRCSEL] [DIVISOR0] [CLKACT]	RPLL. `020 h`. Enabled.		
DBG_FPD_CTRL	`0100_2500 h`, FPD 0x068	[SRCSEL] [DIVISOR0] [CLKACT]	IOPLL_TO_FPD. `025 h`. Enabled.		
DBG_TRACE_CTRL	`0000_2500 h`, FPD 0x064	[SRCSEL] [DIVISOR0] [CLKACT]	IOPLL_TO_FPD. `025 h`. Clock stop.		
DBG_TSTMP_CTRL (Timestamp)	`0000_0A00 h`, FPD 0x0F8	[SRCSEL] [DIVISOR0]	IOPLL_TO_FPD. `0A h`.		The clock enable is controlled by DBG_FPD_CTRL [CLKACT].
AMS_REF_CTRL (PS SYSMON unit)	`0100_1800 h`, LPD 0x108	[SRCSEL] [DIVISOR0] [DIVISOR1] [CLKACT]	RPLL. `18 h`. `0 h`. Enabled.		
DLL_REF_CTRL	`0000_0000h`, LPD 0x104	[SRCSEL]	IOPLL.		
PCAP_CTRL	`0000_1500 h`, LPD 0x0A4	[SRCSEL] [DIVISOR0] [CLKACT]	IOPLL. `15 h`. Disabled.		
TIMESTAMP_REF_CTRL	`0000_1800 h`, LPD 0x128	[SRCSEL] [DIVISOR0] [DIVISOR1] [CLKACT]	IOPLL. `18 h`. 0 `h`. Disabled.		
PL{0:3}_REF_CTR	`0005_2000 h`, LPD 0x0C0 to 0x0CC	[SRCSEL] [DIVISOR0] [DIVISOR1] [CLKACT]	IOPLL. `20 h`. `25 h`. Disabled.		

Table 37-7: **LPD Peripheral Clock Control**

Register Name	Reset Value, Address Offset	Register Parameter	Reset State	Pre-FSBL	Comments
GEM{0:3}_REF_CTRL	0000_2500 h, 0x050 - 0x05C	[SRCSEL] [DIVISOR0] [DIVISOR1] [CLKACT] [RX_CLKACT]	IOPLL. 25 h. 00 h. Disabled. Disabled.		
GEM_TSU_REF_CTRL	0005_1000 h, 0x100	[SRCSEL] [DIVISOR0] [DIVISOR1] [CLKACT]	IOPLL. 10 h. 05 h. Disabled.		
USB{0,1}_BUS_REF_CTRL	0005_2000 h, 0x060 -0x064	[SRCSEL] [DIVISOR0] [DIVISOR1] [CLKACT]	IOPLL. 20 h. 05 h. Disabled.		
UART{0,1}_REF_CTRL	0100_1800 h, 0x074, 0x078	[SRCSEL] [DIVISOR0] [DIVISOR1] [CLKACT]	IOPLL. 18 h. 00 h. Enabled.	Same.	N/A
SPI{0, 1}_REF_CTRL	0100_1800 h, 0x07C, 0x080	[SRCSEL] [DIVISOR0] [DIVISOR1] [CLKACT]	IOPLL. 18 h. 00 h. Enabled.	Same.	N/A
QSPI_REF_CTRL	0100_0800 h, 0x068	[SRCSEL] [DIVISOR0] [DIVISOR1] [CLKACT]	IOPLL. 08 h. 00 h. Enabled.	Same. 0F h. 01 h. Same.	Quad-SPI boot: CBR: 0101_0F00 h.
NAND_REF_CTRL	0005_2000 h, 0x0B4	[SRCSEL] [DIVISOR0] [DIVISOR1] [CLKACT]	IOPLL. 20 h. 05 h. Disabled.	Same. h. h. Same.	NAND boot: CBR: h.
SDIO{0, 1}_REF_CTRL	0100_0F00 h, 0x06C, 0x070	[SRCSEL] [DIVISOR0] [DIVISOR1] [CLKACT]	IOPLL. 0F h. 00 h. Enabled.	Same. 19h. 01 h. Same.	SD card boot: CBR: 0101_1900 h.

Table 37-7: **LPD Peripheral Clock Control *(Cont'd)***

Register Name	Reset Value, Address Offset	Register Parameter	Reset State	Pre-FSBL	Comments
CAN{0, 1}_REF_CTRL	`0100_1800 h`, 0x084, 0x088	[SRCSEL] [DIVISOR0] [DIVISOR1] [CLKACT]	IOPLL. `18 h`. `00 h`. Enabled.	Same. `32 h`. `00 h`. Same	POR reset: PMU: `0100_3200 h`.
I2C{0, 1}_REF_CTRL	`0100_0500 h`, 0x120, 0x124	[SRCSEL] [DIVISOR0] [DIVISOR1] [CLKACT]	IOPLL. `05 h`. `00 h`. Enabled.	Same.	N/A

Table 37-8: **FPD Peripheral Clock Control**

Register Name	Reset Value, Address Offset	Register Parameter	Reset State	Pre-FSBL	Comments
DP_VIDEO_REF_CTRL	`0103_2300 h`, 0x070	[SRCSEL] [DIVISOR0] [DIVISOR1] [CLKACT]	DPLL. `23 h`. `0 h`. Enabled.		
DP_AUDIO_REF_CTRL	`0103_2300 h`, 0x074	[SRCSEL] [DIVISOR0] [DIVISOR1] [CLKACT]	VPLL. `23 h`. `0 h`. Enabled.		
DP_STC_REF_CTRL	`0120_3200 h`, 0x07C	[SRCSEL] [DIVISOR0] [DIVISOR1] [CLKACT]	VPLL. `32 h`. `20 h`. Enabled.		
DPDMA_REF_CTRL	`0100_0500 h`, 0x0BC	[SRCSEL] [DIVISOR0] [CLKACT]	APLL. `05 h`. Enabled.		
GPU_REF_CTRL	`0000_1500 h`, 0x084	[SRCSEL] [DIVISOR0] [CLKACT] [PP0_CLKACT] [PP1_CLKACT]	IOPLL_TO_FPD. `15 h`. Disabled. Disabled. Disabled.		
PCIE_REF_CTRL	`0000_1500 h`, 0x0B4	[SRCSEL] [DIVISOR0] [CLKACT]	IOPLL_TO_FPD. `15 h`. Disabled.		
SATA_REF_CTRL	`0100_1600 h`, 0x0A0	[SRCSEL] [DIVISOR0] [CLKACT]	IOPLL_TO_FPD. `16 h`. Enabled.		

Reset System

Introduction

The PS reset subsystem is responsible for handling the external reset input to the device and that all internal reset requirements are met for the system (as a whole) and for the functional units.

The processing system (PS) reset sequences are divided into three functional areas.

- The power-on reset sequence.
- The management of other signals in the system to trigger system reset.
- The ability (in software) to reset each individual functional unit.

Due to transactional complexity, resetting of individual peripherals should not be attempted without knowing that the system is quiet. Because the low-power domain (LPD) always has power and the full-power domain (FPD) might not have power, the reset block in the LPD contains most of the reset logic. The reset block in the FPD only contains the logic for software to reset individual peripherals in the FPD, and all debug resets come from the reset block in LPD as well. Every module used in the PS receives a reset signal generated from the reset block in the LPD or FPD, which is synchronized into the clock domain of that module and distributed throughout the module.

Features

- Power on reset and system reset.
- Independent PS reset capability while PL is still operating.
- Independent debug reset signals.
- Software identification of cause of reset from reset reason register.

Functional Description

The reset sequence is a two stage process. The first stage is handled by the reset controller present in the LPD and the second stage is handled by the platform management unit (PMU). The following sections discuss the resets.

Figure 38-1 shows how a reset is generated by the two reset controllers; one for each of the power domains (LPD and FPD). The primary device-level reset inputs are from the PS_POR_B and PS_SRST_B device pin, which must be asserted and deasserted based on specific conditions. The *Zynq UltraScale+ MPSoC Data Sheet: DC and AC Switching Characteristics* [Ref 2] contains those specifications. The power-on reset (POR) unit in the PS deasserts the reset signal to the LPD and the FPD clock and reset controllers when the PS power comes up and is stable. The system-level reset signals are controlled using the SLCR registers. The operation of the reset unit requires the PS_REF_CLK to be active.

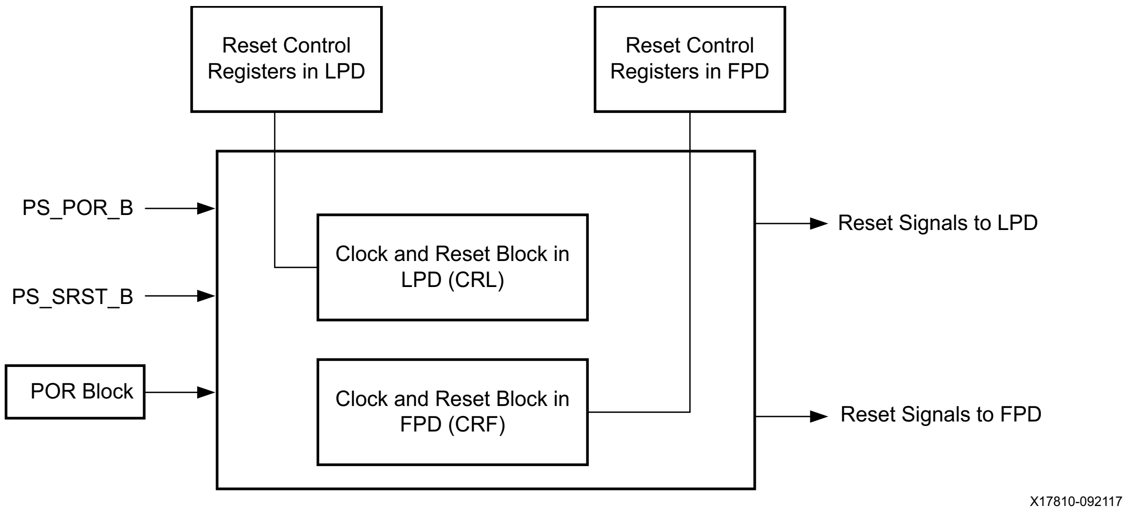

Reset Control Registers in LPD
Reset Control Registers in FPD
PS_POR_B
PS_SRST_B
POR Block
Clock and Reset Block in LPD (CRL)
Clock and Reset Block in FPD (CRF)
Reset Signals to LPD
Reset Signals to FPD
X17810-092117

Figure 38-1: **Top-level Reset Block Diagram**

POR Reset Sequence

The first stage is used to ensure that all the power rails are powered (up). The external PS_POR_B signal is taken from the IOB and passed through an AND gate with a signal from the power-on reset block. A glitch filter is used to ensure that the power is stable for 32 PS_REF_CLK cycles cycles. The sampling value from the boot mode pins are replicated three times and a voter circuitry is used to select the appropriate boot mode sample value. After releasing power-on reset (POR), the eFUSE is cached and scan clear starts up. The reset controller in the LPD holds full control of the system until the LPD reset sequence is completed. Post LPD reset sequence, the reset controller gives control to the PMU. See Figure 38-2 for the flow.

The system reset is deasserted once the reset logic hands off control to the PMU and is asserted back when an event, such as a debug system reset, occurs that needs to assert system reset.

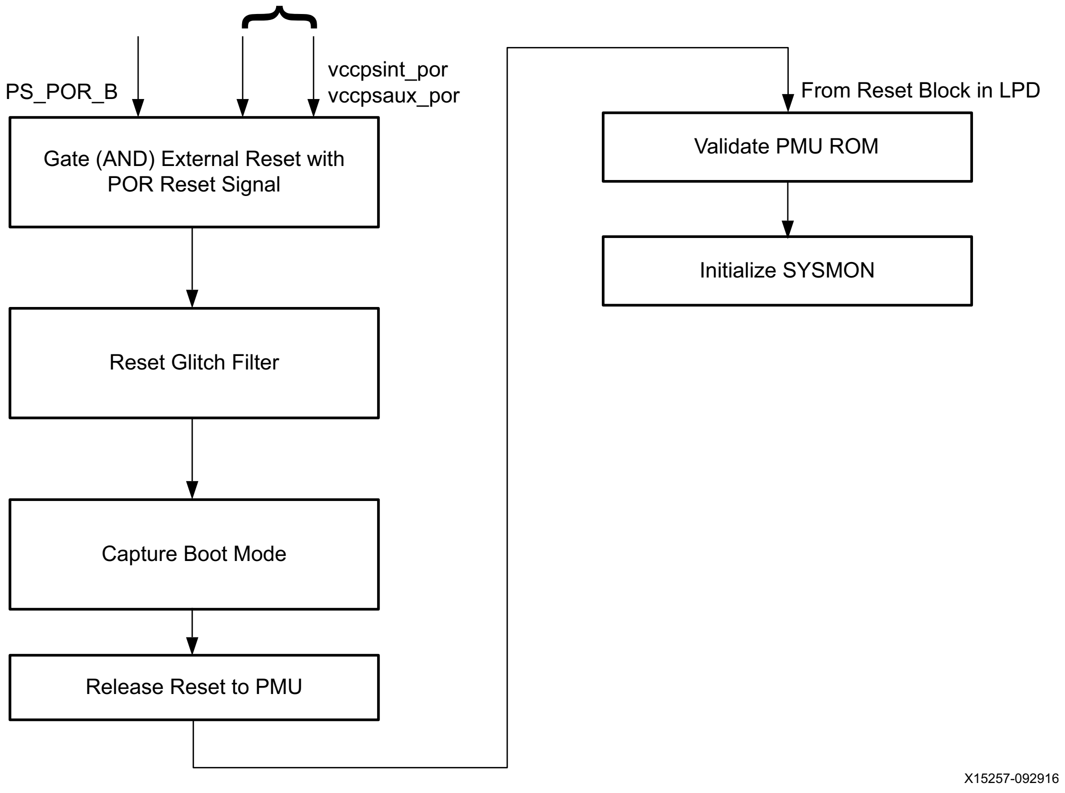


Figure 38-2: **Reset Flow Performed by Reset Controller and PMU**

PS_SRST_B Reset Pin During Hardware Boot

The PS_SRST_B input pin is disabled after PS_POR_B is released and stays disabled during the first part of CSU ROM execution.

- For non-secure boot, the PS_SRST_B input is enabled before loading the FSBL into OCM memory.
- For secure boot, the PS_SRST_B input remains disabled (not enabled by CSU ROM). Secure code can enable the PS_SRST_B reset input using the CRL_APB.RESET_CTRL register.

Example

In this example, the PS_SRST_B reset input is held Low (asserted) while PS_POR_B is deasserted and the CSU executes its ROM code. The CSU continues to execute the ROM code (regardless of the PS_SRST_B state) until the CSU can verify if the boot is secure or non-secure by reading the boot header. For non-secure boot and PS_SRTS_B asserted, the CSU execution halts when it enables the PS_SRST_B input. For secure boot and PS_SRST_B asserted, the CSU continues to execute and the FSBL is loaded into memory because the PS_SRST_B pin remains disabled.

System Reset Conditions

The device resets are summarized in Table 38-1.

Table 38-1: **Resets**

Reset Name	Source	Control	System Effects[1]	Subject to the PROG_GATE effect for PL?	Reset Reason
External POR	PS_POR_B pin	Deasserted after power up. Assert at any time with immediate affect.	Resets all logic, RAMs, and registers. Prepares device for possible secure boot. • Mode pins sampled.	No	[external_por]
Internal POR	System error	PMU_Global.ERROR_POR_{1,2}.	Same as External POR except: • ERROR_STATUS_{1, 2} registers unaffected.	No	[internal_por]

Table 38-1: **Resets *(Cont'd)***

Reset Name	Source	Control	System Effects[1]	Subject to the PROG_GATE effect for PL?	Reset Reason
External SRST	PS_SRST_B pin[2]	Device pin that is usually connected to the debugging tool. Disable RESET_CTRL [srst_dis]	Same as External POR except: Modes pins not sampled. Several registers unaffected (i.e., require a POR to reset): • System error enable. • PMU global persistent. • CSU_status, ENC_status • LOC_PWR_STATE (power state). • RAMs not cleared.	Yes	[srst]
Internal SRST	Register write	CRL_APB.RESET_CTRL [soft_reset].	Same as external SRST.	Yes	[soft]
	System error	PMU_GLOBAL.ERROR_SRST_{1,2}.	Same as external SRST.	Yes	[pmu_sys_reset]
	Register write	CRL_APB.RST_LPD_TOP [fpd_reset].	Same as external SRST except: • PL and LPD unaffected.	N/A	
	Register write	PMU_GLOBAL.GLOBAL_RESET [PS_ONLY_RST].	Same as external SRST except: • PL and LPD unaffected.	Yes	[psonly_reset_req]
Debug SRST	DAP controller	Arm DAP.	Same as external SRST except the debug logic state is preserved.[3]	Yes	[debug_sys]
Reset Debugger	DAP controller	BLOCKONLY_RST [debug_only].	Resets the CoreSight debug logic only.	N/A	No change.

Notes:

1. All resets have an immediate effect. Effects are driven by reset edges and levels.
2. The PS_SRST_B pin can be enabled and disabled by writing to the CRL_APB.RESET_CTRL [srst_dis] bit.
3. RPU debug logic is not preserved.

Reset Reason Register

The cause for each reset is stored in the crl_apb.RESET_REASON register. Table 38-2 summarizes the different values for the reset reason register.

Table 38-2: **Reset Reason Register**

Bit Field	Bit	Description
external_por	0	External POR; the PS_POR_B reset signal pin was asserted.
internal_por	1	Internal POR. A system error triggered a POR reset.
pmu_sys_reset	2	Internal system reset. A system error triggered a system reset.
psonly_reset_req	3	PS-only reset. Write to PMU_GLOBAL.GLOBAL_RESET [PS_ONLY_RST].
srst	4	External system reset; the PS_SRST_B reset signal pin was asserted.
soft	5	Software system reset. Write to RESET_CTRL [soft_reset].
debug_sys	6	Software debugger reset. Write to BLOCKONLY_RST [debug_only].

PS Only Reset

When the PS must be reset without resetting the PL, the PMU (only the PMU) must manage this reset sequence. Through the error mechanism in the PMU, all of the errors can be set to cause a PS only reset. The PMU asserts a signal to the PL power domain that blocks PS_PROG_B from assertion by the CSU. This bit is controlled by the PMU and can be set at the beginning of time or during an interrupt routine. Once this bit is set in the PMU, the only difference to the reset block in the LPD between a PS-only reset and a system-reset request is that the reset block in the LPD marks a different bit in the reset reason register when the PL is not reset.

System-level Software Reset

Each module in the PS has one or more software controlled resets that are asserted from the reset module to the PS block residing in the low-power or full-power domain. The resets are generated by the reset module that is in the same power domain as the consuming module. For instance, the APU resets come from the reset block in the FPD, while the Cortex®-R5F resets come from the reset block in the LPD. The reset block in the LPD is reset when there is a system-level reset. A reset applied to the reset block in the FPD resets all the blocks in the FPD.

Software can write to the FPD reset pin in register APB_CRL.RST_LPD_TOP [FPD_RESET] to reset the FPD logic. The PMU also has the ability to reset the FPD.

The WARMRSTREQ signal from the APU is routed to the PMU. It can be used to trigger a block reset to the APU system.

IMPORTANT: *The system can hang when software reset control is asserted during a pending AXI/APB transfer.*

Table 38-3 summarizes the block-level reset register for each of the blocks in the LPD and FPD.

Table 38-3: **Resets to System Elements**

System Element	Register	Description
LPD System Elements (CRL_APB Register Set)		
GEM	RST_LPD_IOU0[gem<0-3>_reset]	GEM Ethernet controllers.
GPIO	RST_LPD_IOU2[gpio_reset]	GPIO controller.
LPD DMA	RST_LPD_IOU2[lpd_dma_reset]	LPD DMA controller.
NAND	RST_LPD_IOU2[nand_reset]	NAND controller.
LPD SWDT	RST_LPD_IOU2[swdt_reset]	LPD watchdog timer (wdt1).
TTC	RST_LPD_IOU2[ttc{0:3}_reset]	TTC triple counter.
I2C	RST_LPD_IOU2[i2c{0:1}_reset]	I2C controller.
CAN	RST_LPD_IOU2[can{0:1}_reset]	CAN controller.
SDIO	RST_LPD_IOU2[sdio{0:1}_reset]	SDIO controller.
SPI	RST_LPD_IOU2[spi{0:1}_reset]	SPI controller.
UART	RST_LPD_IOU2[uart{0:1}_reset]	UART controller.
QSPI	RST_LPD_IOU2[qspi_reset]	Quad-SPI controller.
PS SYSMON	RST_LPD_TOP[sysmon_reset]	PS system monitor.
RTC	RST_LPD_TOP[rtc_reset]	Real-time clock.
APM	RST_LPD_TOP[apm_reset]	AXI performance monitor.
IPI	RST_LPD_TOP[ipi_reset]	Interprocessor interrupts (IPI).
USB	RST_LPD_TOP[usb{0:1}_apb_reset] RST_LPD_TOP[usb{0:1}_hiberreset] RST_LPD_TOP[usb{0:1}_corereset]	USB controller.
RPU	RST_LPD_TOP[rpu_pge_reset] RST_LPD_TOP[rpu_amba_reset] RST_LPD_TOP[rpu_r5{0:1}_reset]	RPU MPCore resets. • Entire RPU power island. • AXI interconnect. • Core resets.
OCM	RST_LPD_TOP[ocm_reset]	OCM memory.
PL-LPD interface	RST_LPD_TOP[s_axi_lpd_reset]	Resets from LPD to PL fabric.
FPD System Elements (CRF_APB Register Set except where noted)		
PCIe	RST_FPD_TOP[pcie_cfg_reset] RST_FPD_TOP[pcie_bridge_reset] RST_FPD_TOP[pcie_ctrl_reset]	PCIe controller: • Configuration reset. • Bridge reset (AXI interface). • Controller reset.
DisplayPort	RST_FPD_TOP[dp_reset]	DisplayPort controller.
FPD SWDT	RST_FPD_TOP[swdt_reset]	FPD watchdog timer (wdt0).

Table 38-3: **Resets to System Elements *(Cont'd)***

System Element	Register	Description
FPD DMA	RST_FPD_TOP[fpd_dma_reset]	FPD DMA controller.
SATA	RST_FPD_TOP[sata_reset]	SATA controller.
PS-GTR	RST_FPD_TOP[gt_reset]	PS GTR transceivers.
GPU	RST_FPD_TOP[gpu_pp{0:1}_reset]	GPU pixel processors.
HP ports	RST_FPD_TOP[s_axi_hp{0:3}_fpd_reset] RST_FPD_TOP[s_axi_hpc{0:1}_fpd_reset	PS to PL AXI interfaces.
Cortex-A53 CPU	RST_FPD_APU[acpu{0:3}_pwron_reset]	Individual resets to each APU core.
APU L2 cache reset	RST_FPD_APU[apu_l2_reset]	L2 cache reset.
DDR	PMU_GLOBAL.GLOBAL_RESET [FPD_RST]	The DDR controller can only be successfully reset using the FPD reset.
APM	RST_DDR_SS[apm_reset]	AXI performance monitors on DDR interface ports.

Debug Reset

Power-on reset asserts all the debug resets and then returns them to their default state as defined in the SLCR registers. During a debug reset, the PMU writes the APB register that acts as a software reset and toggles the debug reset that needs to be reset. System reset does not affect debug resets.

PL Reset

The Zynq UltraScale+ MPSoC has general-purpose output pins connected to the PMU block that can be used to reset blocks in the PL. Refer to Chapter 6, Platform Management Unit for more details.

PL Configuration Reset

PL configuration reset is the default (but optional) effect of the PS system reset. If enabled, the PL configuration reset begins as de-assertion of the system reset (not immediately at the assertion of the system reset). The Zynq UltraScale+ device has an independent PS reset capability while the PL is still operating. To support the feature described in the PS Only Reset section, the PL configuration reset triggers can be blocked. If enabled, the PL configuration reset occurs optionally and does not begin until the de-assertion of the system resets. When the PL configuration is reset, the PL I/O pins are tri-stated and the PL configuration is cleared. The following table describes system reset input pins that can optionally trigger a PL configuration reset.

Table 38-4: **System Reset Input Pins That Can Reset the PL Configuration**

Reset Pin Name	Description of Effect on PL Configuration
PS_POR_B	Power-on reset signal that resets the PS when asserted. Tri-stating of the PL I/O and clearing of PL configuration begins at the de-assertion of the PS_POR_B signal.
PS_SRST_B	System reset commonly used during debug to reset the PS. By default, but optional, tri-stating of the PL I/O and clearing of the PL configuration begins at the de-assertion of the PS_SRST_B signal.
PS_PROG_B	Direct PL configuration reset signal that tri-states PL I/O pins and clears the PL when asserted, except this input is blocked when PS_POR_B is asserted and blocked when the PS is setup for PS Only Reset

When there is a need to reset and stop operation of the whole Zynq UltraScale+ device, including the PL operation, Table 38-4 indicates that simply driving PS_POR_B Low is insufficient to reset and stop the PL operation. Instead, multiple options exist for resetting and stopping operation of the whole Zynq UltraScale+ device, including:

- Apply a High-Low-High pulse to PS_POR_B. Asserting PS_POR_B Low resets the PS, and the Low-to-High transition of the pulse assures the PL configuration is also reset.
- Assert PS_SRST_B and PS_PROG_B, but do not assert PS_POR_B, to reset the PS and reset the PL configuration.
- Assert PS_POR_B and use general output signals from the PMU to the PL to disable desired portions of the PL logic.

Register Overview

Table 38-5 describes the registers that can be used to configure resets belonging to different power domains.

Table 38-5: **Reset System Registers**

Register Type	Register Name	Description
Low-Power Domain		
LPD reset	RESET_CTRL	Reset control register. Controls miscellaneous functions with regards to triggers.
	BLOCKONLY_RST	Records the reason for the block-only reset.
	RESET_REASON	Records the reason for the reset in the RESET_REASON register.
	RST_LPD_TOP	Software control register for the LPD block.
	RST_LPD_DBG	Debug register for both the LPD and FPD. Only the POR can cause hardware to clear this register. During a debug_reset, the PMU resets this register.
	RST_LPD_IOU0	Software controlled reset for the GEM.
	RST_LPD_IOU1	Power-on reset type register.
	RST_LPD_IOU2	Software control register for the IOU block. Each bit causes a single peripheral or part of the peripheral to be reset.
Full-Power Domain		
FPD reset	RST_FPD_TOP	FPD block-level software controlled reset.
	RST_FPD_APU	APU block-level software controlled resets.

Programming Model

The examples in this section show how to perform reset sequencing to different blocks within the PS.

PS-only Reset Sequence

The PS-only reset requirement is to reset the PS while the PL remains active. The PS-only reset can be triggered by a hardware error signal or a software register write. If the PS-only reset is due to an error signal, then the error must also be indicated to the PL.

The PS-only reset can be implemented as a subset of the system-reset. However, it needs to gracefully terminate the PS to PL AXI transactions before initiating a PS-only reset. A PS-only reset sequence can be implemented as follows.

1. Set pmu_global.PS_CNTRL[prog_gate] to 1 to block the PL from being reset when the PS is reset.
2. An error interrupt is asserted and the action requires a PS-only reset. This request is sent to the PMU as an interrupt.
3. To indicate to the PL, set the PMU error (PS-only reset).
4. Block the FPD to PL and the LPD to PL interfaces with the help of the AMBA isolation block (AIB).
5. If the AIB acknowledgment is not received, then the PMU should timeout and continue.
6. Block the PL to FPD and PL to LPD interfaces with the help of the AIB (in the PL design).
7. If the AIB acknowledgment is not received, then the PMU should timeout and continue.
8. Initiate a PS-only reset by writing to the PMU global reset request register.
9. Assert a PS-only reset by writing to the pmu_global.GLOBAL_RESET[ps_only_rst] bit. This bit is self clearing and causes a PS only reset.
10. Release all signals from being isolated between the PS and PL.

FPD Reset Sequence

The FPD-reset resets all of the full-power domain (FPD). It can be triggered by errors or a software register write. If the FPD reset is due to an error signal, then the error must also be indicated to the LPD and the PL. The FPD reset can be primed by leveraging the FPD power-up sequence. However, it needs to gracefully terminate the FPD ingress/egress AXI transactions before initiating reset of the FPD. The FPD reset sequence can be produced as follows.

- An error interrupt is asserted a FPD reset is required. This request is sent to the PMU as an interrupt.
- Block the FPD to LPD interfaces with the help of the AIB.
- If an AIB acknowledgment is not received, then the PMU should timeout and continue.
- Block the FPD to PL interfaces with the help of the AIB (in the FPS).
- If the AIB acknowledgment is not received, then the PMU should timeout and continue.
- Block the LPD to FPD interfaces with the help of the AIB.
- Block the PL to FPD interfaces with the help of the AIB (in PL design). The PL wrapper should provide a timeout between this AIB and the FPD.
- Assert the FPD reset (by writing to a PMU global register).
- Unblock the FPD to LPD and FPD to PL interfaces.
- Deassert the FPD reset (including CCI), which enables the LPD requests to go to the FPD.

- Unblock the LPD to FPD and the PL to FPD interfaces.
- Deassert the APU L2/CPU resets, which results in an APU reboot.

RPU Reset Sequence

Each of the Arm Cortex-R5F real-time processors can be independently reset. In lock-step, only the R5_0 needs to be reset to reset both Cortex-R5F processors. It can be triggered by errors or a software register write. The Cortex-R5F reset can be triggered (due to a lock-step error) to reset and restart the RPU. It needs to gracefully terminate the Cortex-R5F ingress/egress transactions before initiating reset of the corresponding Cortex-R5F processor. The following steps describe a special case RPU reset.

- An error is asserted which requires a Cortex-R5F processor reset. This request is sent to the PMU as an interrupt.
- Block the Cortex-R5F processor master interfaces with the help of the AIB.
- If an AIB acknowledgment is not received, then the software should timeout and continue.
- Block the Cortex-R5F processor slave interfaces with the help of the AIB.
- If an AIB acknowledgment is not received, then the software should timeout and continue.
- Unblock the Cortex-R5F processor master interfaces.
- Assert the Cortex-R5F processor reset. Use the PMU global register.
- Deassert the Cortex-R5F processor reset, which will trigger a Cortex-R5F processor reboot.
- Unblock the Cortex-R5F processor slave interfaces.

Note: While in lock-step mode the R5F’s will continue to run until either the PMU intervenes, the error itself disrupts the R5F's operation or the normal operation causes a halt. There is no provision within the RPU to specifically inhibit/alter operation of the R5F’s in response to a mismatch.

System Test and Debug

Introduction

The system test and debug features provide intrusive and non-intrusive functionality of an interconnected PS and PL system to debug RPU and APU application software. There are other system elements in the PS and user-selected hardware elements in the PL that can be included in the debug environment.

This chapter is divided between the JTAG interfaces with the Arm DAP, PS TAP, and PL TAP controllers, and the CoreSight system debug functionality. The CoreSight debug functions are accessible by the Arm DAP controller or by system masters accessing the memory-mapped debug registers. The PS TAP controller is alway present and has system functions. The PL TAP controller provides boundary scan (BSCAN) and PL programming functions.

Features

JTAG Chain:

- Single JTAG chain: one, two, or all three TAP controllers.
- Split JTAG chain: PS/PL TAP controllers and Arm DAP controller.
- Triple redundant JTAG security controls.

PS TAP controller:

- IDCODE access.
- PL TAP and Arm DAP controller insertion.
- PS error-code read out.
- System JTAG controls.

PL TAP controller:

- Boundary-scan.
- Legacy PL configuration.
- Legacy PL debug (Vivado logic analyzer).

Arm DAP controller:

- The DAP can be accessed directly in any non-secure boot mode. In secure boot mode, the DAP is not accessible unless trusted software enables the JTAG connection for the DAP controller.
- Arm DAP requires the VCC_PSINTLP power supply.
- Nonvolatile flash programming.
- PS CoreSight debug architecture support.
- PS eFUSE and BBRAM programming.
- Access to AXI interconnect.

Security:

- PSJTAG interface signal tamper detection.
- Always secure from reset to boot header processing.

JTAG state machines:

- Interfaces are compatible with the IEEE Std 1149 specification.
- All states transition on the positive edge of TCK.
- Interfaces are controlled by the TMS signal.

IDCODE instruction:

- Indicates the type of device: device ID codes are listed in Table 1-2.
- Requires LPD to be powered up, but not the PLPD or FPD.
- Always accessible regardless of security state.

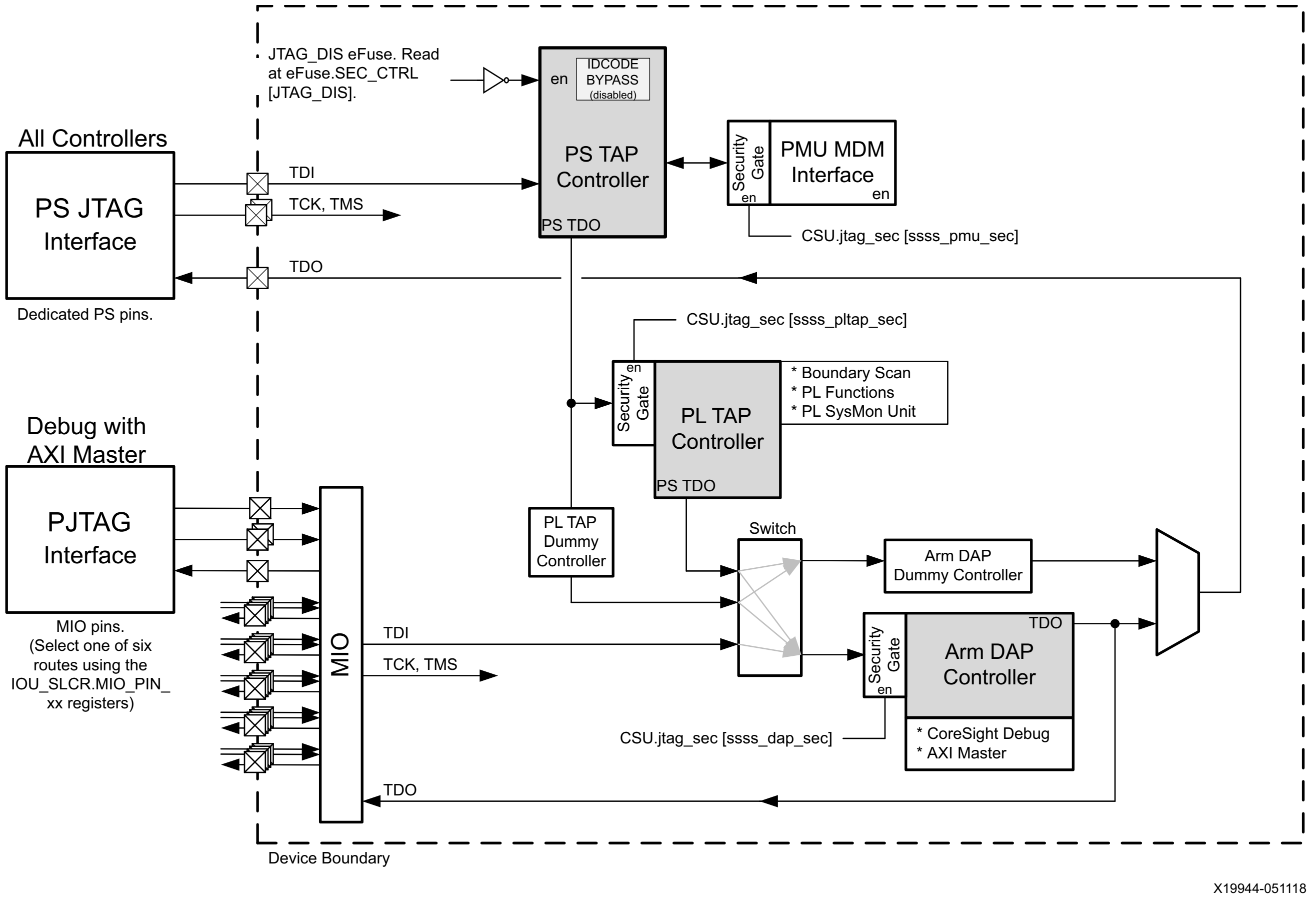


Figure 39-1: **JTAG Chain Block Diagram**

JTAG Functional Description

JTAG is the centerpiece of the debug features for software and PL development, and also serves as a test port for board-level test. Consequently, it is critical to keep JTAG as simple as possible with the least hardware dependency.

The JTAG architecture has three TAP controllers:

- PS TAP (main PS controller with IDCODE).
- PL TAP (PL configuration and boundary scan).
- DAP (Arm debug of RPU and APU using CoreSight).

After a POR reset (PS_POR_B or internal POR), only the dedicated PS JTAG signal pins are activated and only the PS TAP controller is visible on the JTAG chain. The PS TAP controller has limited functionality until the configuration security unit (CSU) has completed the PS boot sequence and granted further functionality.

Full functionality of the PS TAP controller and access to the DAP and PL TAP controllers can be made available after the boot sequence using a special command sent to the PS TAP controller.

Boundary-Scan

Boundary-scan logic is supported through the PL TAP controller. The boundary-scan can only be accessed after the system is booted and it requires PSJTAG interface access to the PL TAP controller.

Security

The Zynq® UltraScale™ MPSoC supports JTAG enable features to enable and disable JTAG to support secure and non-secure boot. When the system comes out of reset, the PL access and the Arm DAP are disabled. With non-secure boot, the CSU ROM enables the PL and DAP access. With secure boot, trusted software must enable the full JTAG debug system.

JTAG Security Gates

The secure JTAG interconnect routes the JTAG signals between the three controllers and controls three security gates (PMU MDM, PL TAP controller, and Arm DAP controller).

Access to the full JTAG chain, including the PS TAP and Arm DAP, can be granted by the following.

- The CSU bootROM code, if the device is booted non-secure.
- Secure software running on the PS.

The security gates are controlled by individual 3-bit fields in the CSU.jtag_sec register. Disabling the security gate does not automatically connect the PS TAP and Arm DAP to the JTAG chain. After access has been granted, the rest of the JTAG chain can be connected using the PS TAP. After adding or removing a controller from the JTAG chain, you must return to test-logic reset (TLR) by holding TMS High for five TCK cycles. This ensures that all TAP controller state machines on the chain are synchronized. The JTAG status can be determined by reading the JTAG_STATUS instruction on the PS TAP.

Arm® CoreSight™ components use four control signals, DBGEN, NIDEN, SPIDEN, and SPNIDEN to authenticate invasive and non-invasive debug based on a TrustZone secure or non-secure status. The debug authentication functionality is described in section Debug Authentication.

Toggle Detect on PSJTAG

The PSJTAG toggle detect is a security feature used to trigger a tamper response in the CSU. The toggle detect sends an alert to the CSU if the TCK is toggled. The alert is sticky and remains asserted until a POR is received. The alert to the CSU requires three cycles of TCK to generate. This helps to prevent false detects from board power-up or other circumstances.

The tamper response is only serviced by the CSU boot ROM when the tamper response register is set in the CSU. The JTAG toggle detect is disabled in the CSU if any of the JTAG security gates are disabled. This allows secure software to have a built-in *debug* mode.

The tamper sources originate external to the CSU (mostly) and are tied to the interrupts of the Secure Processor Block (SPB). When an interrupt is triggered to the SPB, the CSU ROM will read the tamper response register associated with that interrupt and execute the instruction contained in the register. Tamper sources include:

1. AMS alarms (19-bits)
2. External pin via MIO
3. Register in CSU
4. eFuse indicating that the PS has been disabled (from PL)
5. JTAG toggle detect
6. PL SEU error indication
7. JTAG_TOGGLE_DETECT is tied to the interrupts of SPB.

JTAG Chain Configuration

The JTAG chain can be configured to have the PS TAP and the Arm DAP controllers, or all three controllers on a single daisy chain. In each case, the instruction length remains fixed at 16 bits. When a controller is not present, dummy bits are accepted.

The JTAG chain configuration is controlled by the JTAG_DAP_CFG register. This register can be written to by an AXI master accessing a CSU register, by the PS TAP controller using the JTAG_CTRL instruction, or by issuing the JTAG_CTRL instruction to the PS TAP and then writing in the new configuration.

JTAG_CTRL register bits 1 and 0:

00: PS TAP controller

01: PS and PL TAP controllers

10: PS TAP and Arm DAP controllers

11: All three controllers: PS TAP, PL TAP, and Arm DAP

IMPORTANT: *Any time the number of controllers on the JTAG chain is switched, the PS TAP, PL TAP, and Arm DAP controller state machines must be synchronized by holding the TMS High for five cycles of the TCK.*

JTAG Chain Boot States

The JTAG chain is configured by the CSU BootROM during boot based on the security state of the boot. The values for the JTAG chain configuration registers for each boot mode are shown in Table 39-1 to Table 39-3. The state immediately after a POR is a secure state, but the state changes for non-secure device and PJTAG boot modes.

Table 39-1: **POR Boot State and Secure Boot Mode**

Register	Value	Description
JTAG_CHAIN_STATUS	`0x0`	Arm DAP and PL TAP are disabled.
JTAG_DAP_CFG	`0x0`	DAP debug disabled.
JTAG_SEC	`0x0`	Security gates enabled.

Table 39-2: **PJTAG Boot Mode**

Register	Value	Description
JTAG_CHAIN_STATUS (ro)	`0x1`	Arm DAP is disabled. PL TAP is enabled.
JTAG_DAP_CFG (r/w)	`0xFF`	DAP debug enabled (invasive and non-invasive).
JTAG_SEC (r/w)	`0x3F`	Security gates disabled.

Table 39-3: **Non-secure Boot Mode**

Register	Value	Description
JTAG_CHAIN_STATUS	`0x3`	Arm DAP and PL TAP are enabled.
JTAG_DAP_CFG	`0x3F`	DAP debug enabled (invasive and non-invasive).
JTAG_SEC	`0x3F`	Security gates disabled.

PJTAG Interface

An alternate option for communication with the Arm DAP is through the PJTAG signals. There are six PJTAG interfaces specified in the MIO. Using the MIO SLCR, you can select one of the PJTAG0-5 MIO interfaces to be the PJTAG interface. The PJTAG interface enters the JTAG security gate circuit, which routes the JTAG chain around the device.

To use the PJTAG interface, the following conditions must be met.

- The JTAG security gate is disabled by writing to the correct register in the CSU.
- The Arm DAP is *not* on the JTAG chain.

To prevent security holes, the PJTAG is multiplexed into the JTAG signaling before the security gate.

CAUTION! *The PJTAG interface can be disabled by the PS TAP controller when the Arm DAP controller is placed back on the JTAG chain using the JTAG_DAP_CFG register.*

The JTAG interface signals are listed in Table 39-13.

JTAG Disable

The JTAG block can be permanently disabled by programming the appropriate eFUSE in the efuse_pgm_addr (`0xFFCC_000C`) register. When this eFUSE is blown, the JTAG is restricted to two commands, IDCODE and BYPASS. IDCODE is only available by resetting the JTAG controller (going to the test-logic-reset state). All commands shifted into the IR are converted to BYPASS. Also, when the JTAG disable eFUSE is blown, all security gates are permanently enabled, making it impossible to reach the Zynq UltraScale+ MPSoC TAP or the Arm DAP.

- IDCODE available by shifting JTAG to test-logic-reset state.
- BYPASS available by shifting in any other instruction to the IR.
- All security gates permanently enabled.

Instruction Register

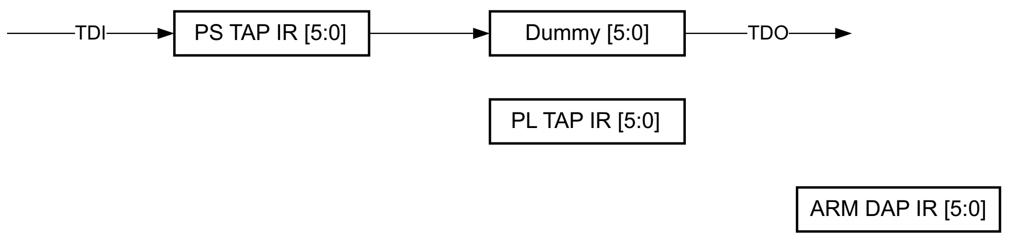
The instruction register allows instructions to be entered serially into the PS TAP controller during the Shift-IR state. Table 39-4 lists the PS TAP instructions.

Table 39-4: **PS TAP Controller Instructions**

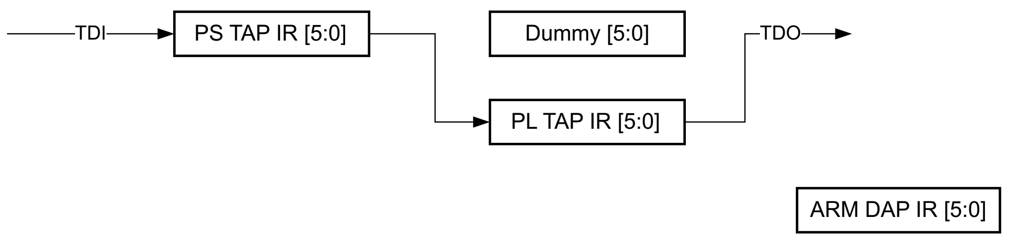
HEX Code	Instruction	Description
0x00	Reserved	Reserved.
0x03	PMU_MDM	Access the PMU MicroBlaze MDM Security gate must allow access [ssss_pmu_sec].
0x08	USERCODE	Access the USERCODE.
0x09	IDCODE	Access the IDCODE, see Table 1-2.
0x0A	HIGHZ	Allows the GTS_USR_B signal from PL TAP controller to enter the PS.
0x19	IP_DISABLE	IP disable status register.
0x1F	JTAG_STATUS	JTAG status register read.
0x20	JTAG_CTRL	Connect/disconnect the PL TAP and Arm DAP.
0x26	EXTEST	Asserts the bscan_extest signal in the PS.
0x3E	ERROR_STATUS	ERROR status register read (46-bit from PMU).
0x3F	BYPASS	

The PL TAP and PS TAP instruction registers are 6-bits each. If the PS TAP is not part of the JTAG chain, the last six bits of the instruction register are dummy bits. The DAP controller is daisy chained to the end of the JTAG when it is activated in the system.

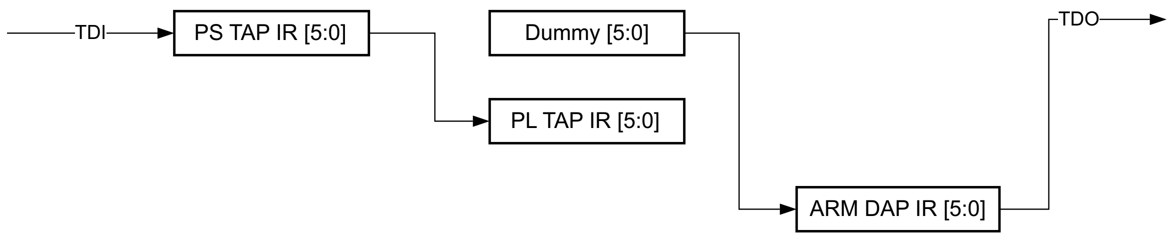
The following diagram shows the instruction register when only the PS TAP is active:



The following shows the instruction register when the PS and PL TAP are active:



The following shows the instruction register when the PS TAP and ARM DAP are active in the system:



The following shows the instruction register when the PS TAP and ARM DAP are active in the system:

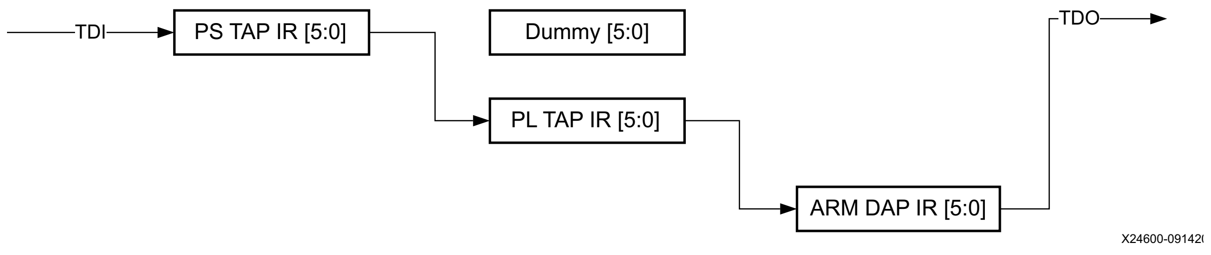


Figure 39-2: **PS and PL Tap**

Instruction Availability

The PS TAP has two modes of operation. These modes change the available instruction set to control what can be accessed by JTAG interface. During reset and pre-boot, only the IDCODE, IP_DISABLE, JTAG_STATUS, ERROR_STATUS, and BYPASS instructions are functional.

The rest of the instructions can become functional depending on the boot mode, eFUSE states, and register settings.

When the PMU processes the boot header, it determines if the system remains in its secure mode or transitions to a non-secure mode.

Control Register

The JTAG control register, JTAG_CTRL, enables the PL TAP and Arm DAP controllers onto the JTAG chain (the PS TAP controller is always present). The control bits are listed in Table 39-5. This register is reset by a system and POR. Regardless of these bit settings, the chain length remains at 12 bits.

Table 39-5: **PS TAP Controller JTAG Control Register**

Bit	Value	Description
31:2	Reserved	Reads 0.
1	Arm DAP	Write 1 to enable the Arm DAP controller.
0	PL TAP	Write 1 to enable the PL TAP controller.

Controller Status Register

The status register provides information about the PS hardware version, device boot mode, BIST results, security gates, and controller connections to the JTAG chain as listed in Table 39-6.

Table 39-6: **PS TAP Controller Status Register**

Bit	Name	Description
31-28	PS_VERSION	Indicates the PS version, same as csu.version [ps_version] register bit
27-24	PL_FABRIC_PIPE	Indicates which of the PSTP fabric port access have a pipeline stage. See Section 6.2.1 for details.
23-20	PSTP_CTRL	Indicates the operating mode of the PSTP. See Section 6.2.1 for details.
19	MODE_IS_DFT	Indicates that the part has successfully booted in DFT mode
18	Unused	Constant 0 value.
17-14	BOOT_MODE	Device boot mode
13	CBR_DONE	Configuration BootROM (CBR) has finished running and the full JTAG instruction set is available
12	SCAN_CLEAR_FAILED	Pre-boot SCAN CLEAR function failed
11	LBIST_FAILED	Pre-boot LBIST function failed
10	BISR_FAILED	Pre-boot BISR function failed
9	PL_PWR_STS	Power status of the PL, cannot connect to the PL TAP if this bit is 0
8	FUSE_MDM_DIS	Indicates that the CSU MDM disable fuses are programmed.
7	DDR_PHY_SEC_GATE	Indicates if the DDR_PHY Security Gate is disabled.
6	PMU_MDM_SEC_GATE	PMU MDM security gate is disabled
5	PL_TAP_SEC_GATE	PL TAP security gate is disabled
4	ARM_DAP_SEC_GATE	Arm DAP security gate is disabled
3	ARM_DAP	Arm DAP is connected in the JTAG chain

Table 39-6: **PS TAP Controller Status Register *(Cont'd)***

Bit	Name	Description
2	PL_TAP	PL TAP is connected in the JTAG chain
1	0	
0	1	

Error Status Register

JTAG is the primary method for transmitting error codes out of the device. The error status register connects the error status from the PMU to the JTAG. The data then shifts the status serially out of the device. The error status register is 121-bits long and the output can be masked with an eFUSE for security purposes.

The error status from the PMU is ORed with the eFUSE during the capture phase of the JTAG state machine when the error status instruction is selected. The errors are only masked to the JTAG error status register. The errors can still be read inside the device from the PMU or the PL (depending on the error).

- 47 bits for hardware errors (these also go to the PL or can be read from the PMU).
- 74 bits for software errors (these can be read from PMU).

The bits for the error status register are described in Table 6-13.

Note: While reading the JTAG_ERROR register, if BISR_failed is asserted, the JTAG_ERROR register must be read again. If it is still asserted after the second read, then this is a true error condition. Otherwise, ignore the first read.

PS TAP Controller

The PS TAP controller can provide system access for itself, and the PL TAP and Arm DAP controllers. The PS TAP controller also provides basic PS-related functions. The device IDCODE is accessible using the PS TAP controller in most all device modes. Boundary-scan requires access to the PL TAP controller, i.e., the device must boot and the security gate must be disabled.

The PS TAP controller is designed to always be active in the system. To ensure security, the PS TAP controller has a limited command set. Before the PS boot is complete (CSU ROM code completes), the PS TAP can perform these instructions:

- BYPASS
- IDCODE

After the PS is booted, the PS TAP controller can be used to control the configuration of the JTAG chain (PL and DAP access). Once access to the PL is established, boundary-scan functions can be performed. The PL access and Arm DAP can only be connected to the JTAG chain if the security gate has been disabled by either the CSU ROM (non-secure boots) or by secure software running on the PS. With non-secure boots, the CSU ROM automatically links the PL TAP and Arm DAP to the JTAG.

The PS TAP controller state machine is reset to the test-logic-reset state by power-cycling the LPD.

- The JTAG_TOGGLE_DETECT register is reset by PS_POR_B. The JTAG toggle detect is a security feature used to trigger a tamper response in the CSU when the JTAG signals are toggled.
- The JTAG_DAP_CFG register is reset by a system reset.
- The rest of the registers are reset when the PS TAP controller state machine is in the test-logic-reset state.
- Assert a reset to the PS TAP controller through the software.

PL TAP Controller

The PL TAP controller connects to the boundary scan logic and a PL configuration interface.

Arm DAP Controller

The DAP controller is based on the Arm debug interface version 5 (ADIv5) comprising a number of components supplied in a single configuration. All the supplied components fit into the various architectural components for the debug ports (DPs), which are used to access the DAP from an external debugger, and access ports (APs), to access on-chip system resources. The debug port and access ports together are referred to as the DAP. The DAP controller supports the following features.

- Central controller for all CoreSight debug components with the PS.
- Interface to external Arm debug tool through the JTAG interface.
- Direct address space access without halting CPUs.
- Invasive/non-invasive debug control.
- Secure/non-secure debug support.

Figure 39-3 shows the access port (AP) options.

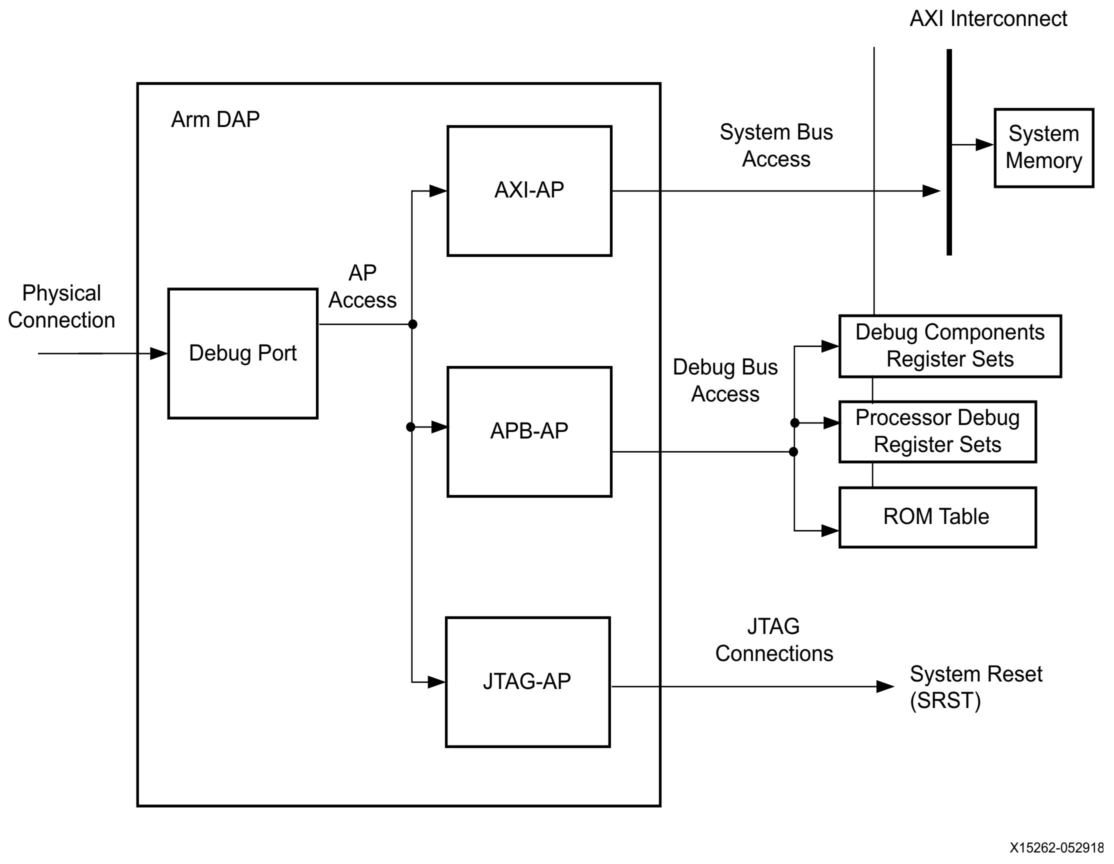


Figure 39-3: **Arm Debug Interface, Showing the Access Port Options**

Arm DAP Controller Functionality

External Flash Memory Programming

To program the non-volatile flash, the DAP controller must be enabled. After the DAP controller is enabled, flash programming routines can be downloaded into the OCM and a DAP initiated wake-up request for the RPU can be sent to the PMU to execute flash programming routines in the OCM. Using the OCM as a data FIFO, continue pushing flash programs into the OCM buffer through JTAG and the RPU, running flash programming routines to program flash using data from the OCM buffer. When the network is enabled, flash programming can be downloaded from the network as part of a flash programming routine.

PS Software Debug

The DAP controller must be enabled to use the software debug features in the device. Once enabled, the debug functionality described in the CoreSight Functional Description section can be used to debug the RPU, APU, and PL.

PS-PL Debug

This mode requires PS software and PL logic debug at the same time. There are two different methods to support this debug.

Xilinx Debug Tools

For a complete system debug environment, both the Arm DAP and PS TAP controllers must be enabled. The processor software debug for the PS uses the DAP controller (CoreSight) and the PL software debug process uses the PS TAP controller. These debug environments are described in the CoreSight Functional Description section.

Third-Party Tool Support

A third-party debugger can connect to the Arm DAP controller using one of the following methods.

1. Connect the PJTAG interface via the MIO pins. Enable the DAP controller onto the PJTAG interface chain.
2. Connect to the PS JTAG interface using the dedicated pins. Enable the DAP controller onto the PS JTAG interface chain (not the PJTAG chain).

   The third party debugger connects to the Vivado Design Suite using a Xilinx virtual cable (XVC) interface and can also connect to the DAP controller using a common cable interface.

Arm DAP Reset Mechanism

DAP is reset by any of the following.

- POR
- LBIST scan clear

CoreSight Functional Description

CoreSight Environment

The PS software debug system for the RPU and APU MPCores is built around the Arm® CoreSight™ SOC-400 components conforming to the Arm CoreSight version 2.0 specification [Ref 42]. The debug functionality is complemented by additional components from Xilinx that are documented in this chapter.

The CoreSight debug environment is accessed via JTAG to reach the Arm DAP and PS TAP controllers. The PL software debug features and the device boundary scan functions are controlled by the PL TAP controller.

The advanced trace bus (ATB) is an AMBA3 stream-like bus protocol used to transport trace data. The ATB components, such as the funnel and the trace memory controller (TMC), are used to manage trace data.

Debug Features

The CoreSight components provide the following capabilities.

- On-chip multicore debug including break point and single-step.
- For the APU MPCore, the embedded trace macrocell™ (ETM) is integrated into the MPCore and captures all CPU waypoints.
- For the RPU MPCore, the ETM captures CPU traces and is external to the MPCore.
- CoreSight system trace macrocell (STM) captures software driven trace and PL events.
- Cross-trigger interface (CTI) and cross-trigger matrix (CTM) allows cross triggering support among multiple trace-capture modules.
- Trace memory controller (TMC) with 8 KB ETF buffer captures and aggregates trace data from individual components. ETF can be used as a trace buffer, which software can read. It can also be used as a FIFO (to absorb bursts of trace traffic) for trace that is output into DDR or the trace-port interface unit (TPIU).
- TPIU is output to MIO or EMIO with double data rate and configurable width, with clock speed up to 125MHz.
- Arm CoreSight standard programming models for standard tool support.
- Standard bus interfaces for CoreSight compliant third-party cores.
- JTAG functionality:
    - All trace capture modules are accessible from external JTAG interface via the Arm DAP controller or an AXI bus master.

- Low-power debug mode. JTAG, through the DAP, has direct memory-space accessibility without stopping the CPU low-power debug mode.
- System debug:
    - Debug and trace visibility of the whole system.
    - On-chip and off-chip buffers and storage for trace data.
    - Time stamping to co-relate events.
    - Single debugger connecting point for entire system debug.
- Xilinx debug components:
    - Dump trace from selected AXI interconnect channels.
    - Packetized trace for compatibility with Arm tools.
    - General purpose signals to and from the PL.
    - Trigger signals to and from the PL.

System Test and Debug Overview

The DAP controller accesses system test and debug functions. It is a CoreSight component of the access and control class, and connects to other components using the programming bus. The DAP controller provides two interfaces to access the CoreSight infrastructure.

- External interface using JTAG, from the device pinout.
- Internal interface using the APB slave, from the slave interconnect.

A debugger can use JTAG to communicate with the CoreSight infrastructure, while software running on a CPU uses the APB through memory-mapped addresses assigned to the CoreSight infrastructure.

The DAP controller can forward system access requests arriving through either the JTAG or APB slave interfaces to the requested CoreSight components. Also, the DAP controller has an AHB master interface on the LPD AXI interconnect to access system elements in the PS other than the CoreSight components.

The DAP controller can forward system memory requests from the JTAG interface to the other system elements in the PS subject to authentication.

The debug system is spread across three power domains (LPD, FPD, and PLPD). Although all power domains should be turned on for full functionality, the basic JTAG functions work as long as the LPD and PLPD power is present. Power is discussed in Clocks, Reset, and Power Domains.

Debug Definition

Debug refers to features used to observe or modify the state of parts of the design. Features used for debug include the ability to read and modify register values of processors and peripherals. Debug also includes the use of complex triggering and monitoring resources. Debug frequently involves halting execution after a failure is observed, and collecting state information retrospectively to investigate the issue.

Trace Definition

CoreSight components provide features that allow for continuous collection of system information for later off-line analysis. Execution trace generation macrocells exist for use with processors, software can be implemented with dedicated trace generation, and some peripherals can generate performance monitoring trace streams. Trace and debug are used together at all stages in the design flow from initial platform bring-up, through software development and optimization, and into in-field debug or failure analysis. Historically, external JTAG and self-hosted internal monitor methods of debugging exist.

Conventional JTAG Debug (External Debug)

External debug is an invasive debug with the processor halted.

- Breakpoints and watch-points are used to halt the processor on a specific activity.
- A debug connection to examine and modify registers and memory, and provide single-step execution

Conventional Monitor Debug (Self-hosted Debug)

Self-hosted debug is an invasive debug with the processor running using a debug monitor that resides in memory.

Trace Debug

Trace debug is a non-invasive debug with the processor running at full speed.

- A collection of information on instruction execution and data transfers.
- Delivery off-chip in real-time, or capture in on-chip memory.
- Tools to merge data with source code on a development workstation for future analysis.
- The TPIU.EXTCTL_OUT_Port register must be set to output trace into the PL.

CoreSight technology addresses the requirement for a multi-processor debug and trace solution with high bandwidth for entire systems beyond the processor, despite increased complexity and clock speeds. Efficient use of pins made available for debug is crucial.

The entire CoreSight debug circuit is distributed across the low and full-power domains. Between these two domains, the low-power domain is the always-on power domain. To support RPU MPCore debug in low-power mode, and minimize CoreSight power, the key top-level debug components are allocated in the LPD.

For further information, see the CoreSight on-chip trace and debug [Ref 53] documentation.

Security

CoreSight components using TrustZone provide security using four authentication signals: DBGEN, NIDEN, SPIDEN, and SPNIDEN. Refer to Debug Authentication in the JTAG Resets section.

For further information, see the CoreSight components [Ref 51] and the Arm CoreSight architecture [Ref 52] documentation.

Debug Authentication

Arm CoreSight components use four control signals, DBGEN, NIDEN, SPIDEN, and SPNIDEN to authenticate invasive and non-invasive debug based on a TrustZone secure or non-secure status. An invasive debug is any debug operation that can cause the behavior of the system to be modified. A non-invasive debug, such as trace, is unaffected.

Note: References to secure and non-secure state in this section refer to the TrustZone state and have nothing to do with boot security.

The authentication rules are as follows.

- If DBGEN is Low, then no invasive debug must be permitted.
- If NIDEN is Low and DBGEN is Low, then no debug is permitted.
- If NIDEN is Low and DBGEN is High, then invasive and non-invasive debug are permitted.
- If SPIDEN is Low, then no secure invasive debug must be permitted.
- If SPNIDEN is Low and SPIDEN is Low, then no secure debug is permitted.
- If SPNIDEN is Low and SPIDEN is High, then invasive and non-invasive secure debug is permitted.

Table 39-7 shows the debug authentication logic.

Table 39-7: **Debug Authentication**

SPIDEN	DBGEN	SPNIDEN	NIDEN	Invasive		Non-invasive	
				Secure	Non-secure	Secure	Non-secure
X	0	X	0	No	No	No	No
0	0	0	1	No	No	No	Yes
0	0	1	1	No	No	Yes	Yes
0	1	0	X	No	Yes	No	Yes
0	1	1	X	No	Yes	Yes	Yes
1	0	X	1	No	No	Yes	Yes
1	1	X	X	Yes	Yes	Yes	Yes

Components

Figure 39-4 shows where the debug infrastructure is located in the PS. It provides a conceptual view. The four ETMs next to the APU MPCore CPUs and their four CTIs and one CTM are inside the APU block; the two ETMs next to the RPU MPCore CPUs and their two CTIs and one CTM are inside the RPU block. The debug infrastructure (Figure 39-4) is split into two power domains (gray for low power). As mentioned above, this figure is intended to show a conceptual view of the debug infrastructure. It is not detailed enough to provide exact connections. In particular, JTAG connections have been abstracted. Please refer to Figure 39-1 for a more detailed diagram of the JTAG Chain connectivity.

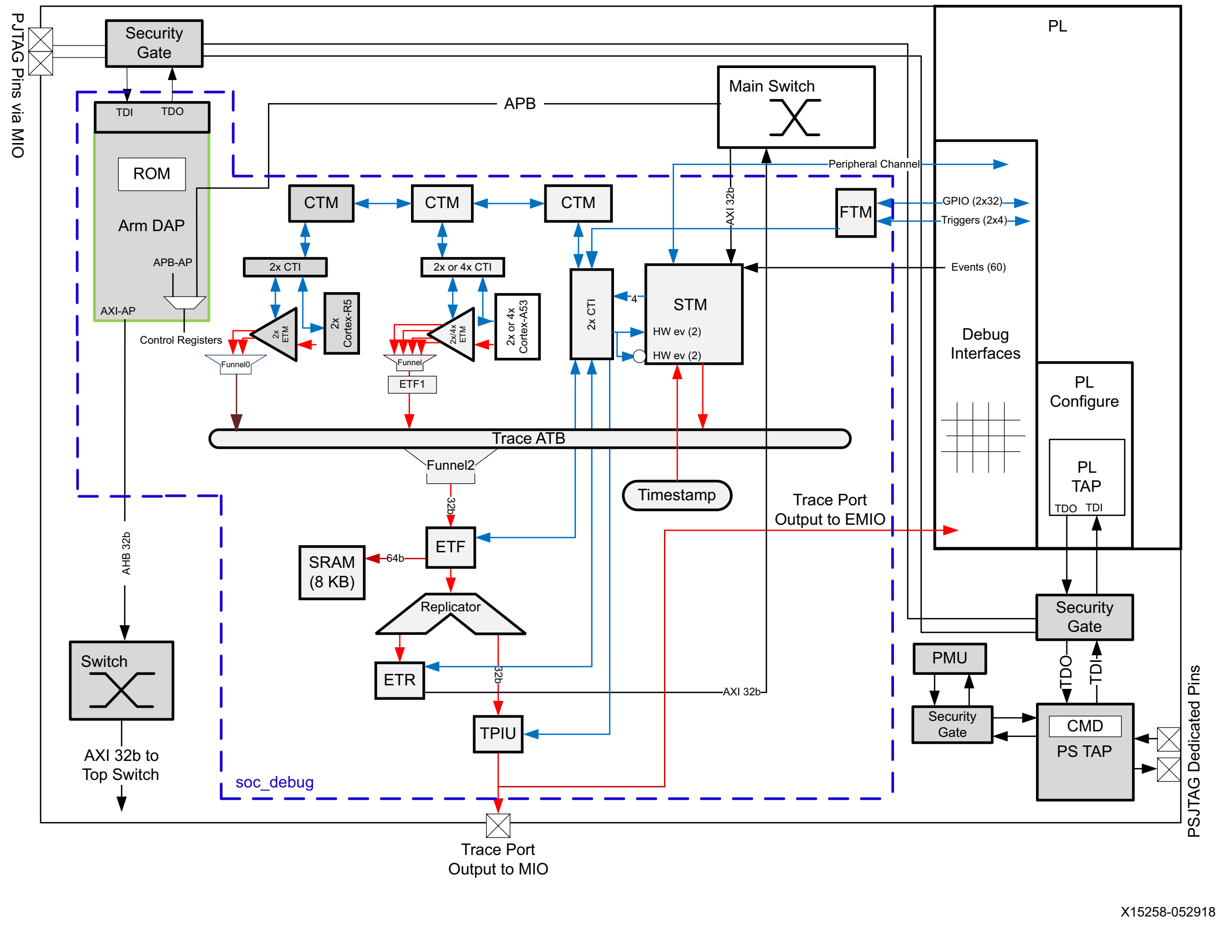


Figure 39-4: **CoreSight Debug Block Diagram**

JTAG and DAP Overview

The JTAG chain is accessed using a standard IEEE Std 1149.1 JTAG interface. It is designed to facilitate system debug software and PL development, and to serve as a test port for boundary scan for board-level testing. The JTAG interfaces and controllers are described in JTAG Chain:.

- Single JTAG port in the PS to support both PS and PL.
- Arm DAP for loading programs, system test, and PS debug.

Bus Structures

The CoreSight architecture employs the following buses to interact with each other within the debug infrastructure, and with the rest of the PS.

JTAG	Four standard JTAG pins (without the optional TRST). These pins are used by debugger tools to interact with the debug infrastructure.
ATB	AMBA trace bus. This bus has 32-bit data and a 7-bit ID, with a ready/valid handshake. The ATB also provides a flush mechanism.
Debug APB	AMBA APB protocol. The DAP controller is the master of this bus. The DAP controller uses this bus to access all other CoreSight components.
System APB	AMBA APB protocol. The DAP is a slave of the system APB bus. It is on the system memory map assigned to this APB bus.
AXI	AMBA AXI protocol. The DAP is the master of this bus. The DAP controller uses the AXI bus to access everything on the system map, subject to authentication.

Debug System Control and Access

This class of CoreSight components provides the capability to control and access the debug infrastructure, in particular, from an off-chip debugger tool.

Debug Access Port

The debug access port (DAP) provides off-chip debug tools with the capability to access the debug infrastructure and the PS, including all debug components and all memory-mapped locations of the PS. For security, authentication requirements must be met to be granted access rights. See Debug Authentication.

The DAP is divided into the following sub-components.

JTAG-DP	Processes JTAG requests, decodes to select an access port (***-AP), requests for power on, and requests for debug reset.
AXI-AP	Provides an AXI master port for access to system memory-mapped locations (subject to authentication).
APB-AP	Provides an APB master port for access to the debug APB.
JTAG-AP	Provides eight JTAG master ports to control on-chip TAP controllers.
APBMUX	Provides access to the debug APB from the system (internal).
DBGROM	Provides pointers to other CoreSight components on the debug APB.

Embedded Cross Trigger

The embedded cross trigger (ECT) provides coordination between CoreSight components. This ECT consists of several cross-trigger interfaces (CTI) and cross-trigger matrices (CTM) connected together. A single CTI can be operated without the requirement for a CTM. The debug system enables debug support for multiple logic-based debug cores and cross triggering between the cores and the processing system.

The main function of the ECT is to pass debug events from one debug component to another. For example, the ECT can communicate debug state information from one core to another, so that (if required) program execution on both processors can be stopped at the same time. The ECT can (optionally) be used to allow an Arm CPU and the programmable logic (PL) to cross-trigger each other, facilitating system-level software debug between the PS and PL.

When performing ECT, allow the Arm CPU and FPGA interconnect to cross trigger each other to improve system-level debug capability.

CTI	The CTI combines and maps the trigger requests, and broadcasts them to all other interfaces on the ECT as channel events. When the CTI receives a channel event, it maps it onto a trigger output. This enables subsystems to cross trigger with each other. The receiving and transmitting of triggers is performed through the trigger interface.
CTM	The CTM controls the distribution of channel events. It provides channel interfaces for connection to either CTIs or CTMs. This enables multiple CTIs to be linked together.

Figure 39-5 shows how CTIs and CTM are used in a generic setup.

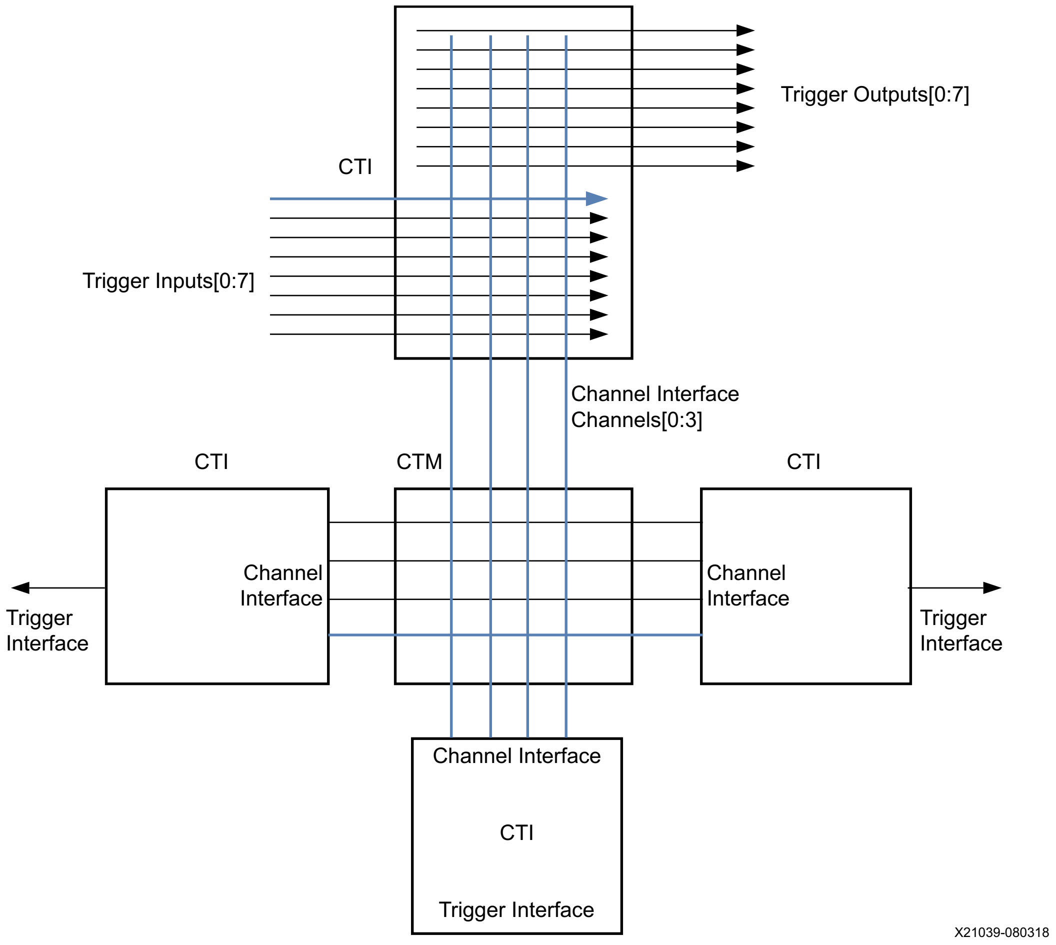


Figure 39-5: **Generic CTI and CTM Architecture**

CTM forms an event broadcasting network with multiple channels. A CTI listens to one or more channels for an event, maps a received event into a trigger, and sends the trigger to one or more CoreSight components connected to the CTI. A CTI also combines and maps the triggers from the connected CoreSight components and broadcasts them as events on one or more channels. Through its register interface, each CTI can be configured to listen to specific channels for events or broadcast triggers as events to specific channels.

In Figure 39-5, there are four channels. The CTI at the top is configured to propagate the trigger event on Trigger Input 0 to Channel 0. Other CTIs can be configured to listen to this channel for events and broadcast the events through trigger outputs, to the debug components connected to these CTIs. CTIs also support channel gating such that selected channels can be turned off, without having to disable the channel to trigger I/O mapping.

In Zynq UltraScale+ MPSoCs, ECT is configured with four broadcast channels, nine CTIs, and a CTM. Table 39-8 shows the trigger input and trigger output connections of each CTI, which are hard wired connections.

Table 39-8: **CTI Connections**

CTI Trigger Port	CTI Signal	CTI Trigger Port	CTI Signal
ETF (CORESIGHT_SOC_CTI_0)		R5-{0,1}	
IN 0	ETF 1 FULL	IN 0	DBGTRIGGER
IN 1	ETF 1 ACQCOMP	IN 1	PMUIRQ
IN 2	ETF 2 FULL	IN 2	ETM EXTOUT[0]
IN 3	ETF 2 ACQCOMP	IN 3	ETM EXTOUT[1]
IN 4	ETR FULL	IN 4	COMMRX
IN 5	ETR ACQCOMP	IN 5	COMMTX
IN 6	-	IN 6	ETM TRIGGER
IN 7	-	IN 7	-
OUT 0	ETF 1 FLUSHIN	OUT 0	EDBGRQ
OUT 1	ETF 1 TRIGIN	OUT 1	ETM EXTIN[0]
OUT 2	ETF 2 FLUSHIN	OUT 2	ETM EXTIN[1]
OUT 3	ETF 2 TRIGIN	OUT 3	- -(CTIIRQ, not connected)
OUT 4	ETR FLUSHIN	OUT 4	-
OUT 5	ETR TRIGIN	OUT 5	-
OUT 6	TPIU FLUSHIN	OUT 6	-
OUT 7	TPIU TRIGIN	OUT 7	DBGRESTART
FTM-STM (CORESIGHT_SOC_CTI_1)		A53-{0,1,2,3}	
IN 0	FTM	IN 0	DBGTRIGGER
IN 1	FTM	IN 1	PMUIRQ
IN 2	FTM	IN 2	-
IN 3	FTM	IN 3	-
IN 4	STM TRIGOUTSPTE	IN 4	ETM EXTOUT[0]
IN 5	STM TRIGOUTSW	IN 5	ETM EXTOUT[1]
IN 6	STM TRIGOUTHETE	IN 6	ETM EXTOUT[2]
IN 7	STM ASYNCOUT	IN 7	ETM EXTOUT[3]
OUT 0	FTM	OUT 0	EDBGRQ
OUT 1	FTM	OUT 1	DBGRESTART
OUT 2	FTM	OUT 2	CTIIRQ
OUT 3	FTM	OUT 3	-
OUT 4	STM HWEVENTS	OUT 4	ETM EXTIN[0]
OUT 5	STM HWEVENTS	OUT 5	ETM EXTIN[1]
OUT 6	-	OUT 6	ETM EXTIN[2]
OUT 7	HALT SYSTEM TIMER	OUT 7	ETM EXTIN[3]

PL to PS and PS to PL Cross Triggering

PL to PS and PS to PL are the most common use cases of cross triggering in Zynq UltraScale+ MPSoCs. There are four trigger inputs on PL CTI, which can be configured to halt (EDBGRQ) any of the CPUs. Similarly, the four PL CTI trigger outputs can be triggered when a CPU is halted (DBGACK). The PL trigger inputs and outputs can be connected to ILA cores so that an ILA trigger can halt the CPUs and the ILA can be triggered to capture the signals it is monitoring when any of the CPUs are halted. For more information on setting up cross triggers to the FTM in the Vivado tools, see the "Cross Trigger Design" section in *Vivado Design Suite: Embedded Processor Hardware Design* (UG940) [Ref 24].

Trace Sources

This class of CoreSight components captures traces, implementing the non-invasive part of the Arm CoreSight architecture. A trace source component normally compresses and formats trace information into packets and sends onto an ATB.

- APU MPCore Embedded Trace Macrocell
- RPU MPCore Embedded Trace Macrocell
- System Trace Macrocell
- ATB Protocol
- PL Fabric Trigger Macrocell

APU MPCore Embedded Trace Macrocell

The APU MPCore embedded trace macrocell (ETM) is a module that performs real-time instruction flow tracing for the APU MPCore, based on the program flow trace (PFT) architecture. The APU MPCore ETM generates information used by the trace tools to reconstruct the execution of all or part of a program. The PFT architecture assumes that the trace tools can access a copy of the application code being traced. For this reason, the ETM generates traces only at certain points in program execution, called waypoints. This reduces the amount of trace data generated by the ETM. Waypoints are changes in the program flow or events, such as an exception. The trace tools use waypoints to follow the flow of program execution.

The APU MPCore ETMs can trace the following.

- Indirect branches, with target address and condition code.
- Direct branches with only the condition code.
- Instruction barrier instructions.
- Exceptions, with an indication of where the exception occurred.
- Changes in processor instruction set state.

- Changes in the processor security state.
- Context-ID changes.
- Entry to and return from debug state when halting debug mode is enabled.
- Cycle count between traced waypoints.
- Global system timestamps (binary value of timestamp).
- Target addresses for taken direct branches.

RPU MPCore Embedded Trace Macrocell

The RPU MPCore ETM provides real-time instruction trace and data trace for the RPU MPCore. The RPU MPCore ETM generates information used by the trace software tools to reconstruct the execution of all or part of a program, with the following features. Each RPU MPCore CPU includes its own ETM.

- All instructions, including condition code pass/fail and dual issue information.
- Load/store address and data values.
- Data values used in coprocessor register transfers.
- Values of context-ID changes.
- Target addresses of taken direct and indirect branch operations exceptions.
- Changes in processor instruction set state.
- Entry to and return from a debug state when the halting debug mode is enabled.
- Cycle counts between executed instructions.

System Trace Macrocell

The system trace macrocell (STM) provides software trace instrumentation. The STM (Figure 39-6) provides an APB interface for software and debugger access, and connects to the ATB for trace output, along with authentication inputs, trigger events, and acknowledge.

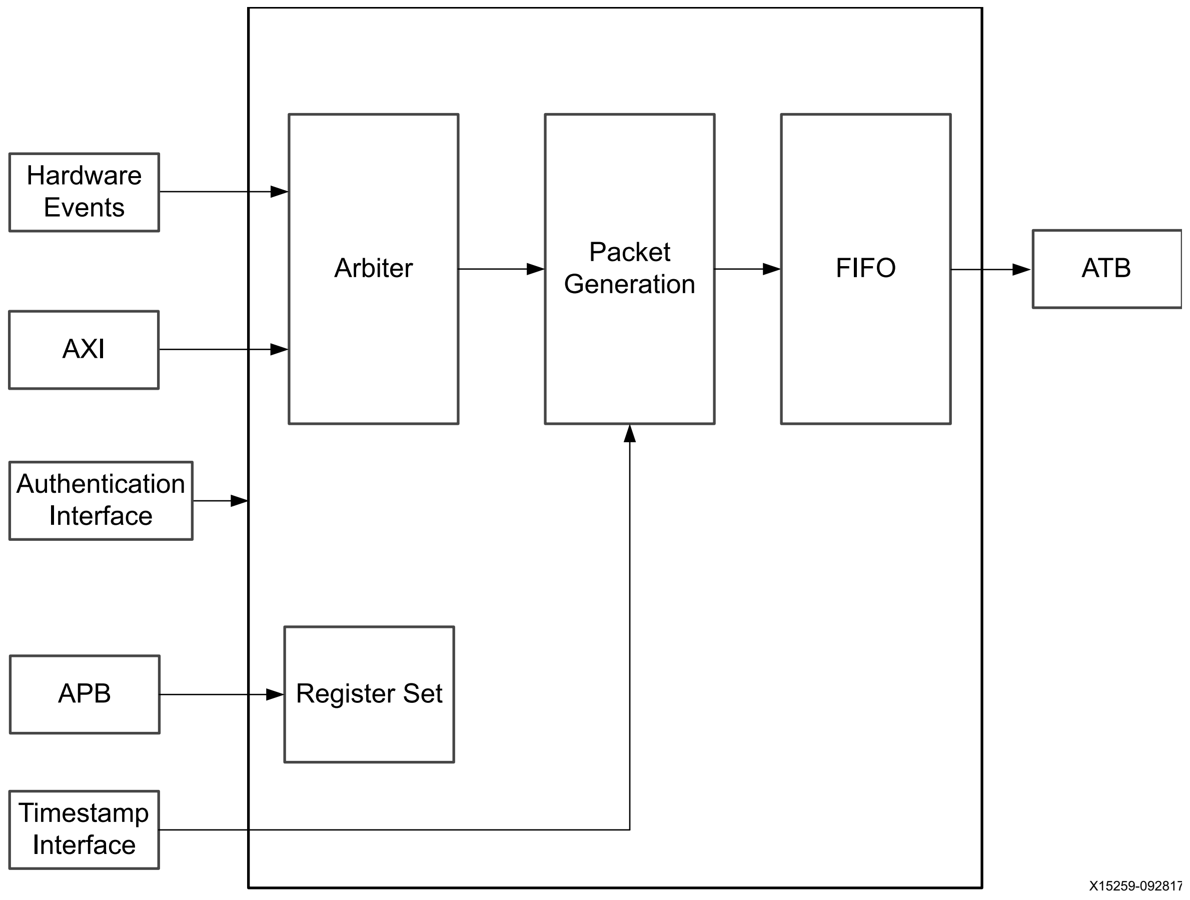


Figure 39-6: **STM Block Diagram**

The STM supports a trace stream that conforms to the MIPI System Trace Protocol version 2. The STM block is a software application driven trace source to generate an application software instrumentation trace (SWIT). The STM hardware event observation interface enables monitoring and tracing of 64 hardware events, each of which is represented by a single bit. This functionality can be used to monitor interrupts, cross-triggers, and other signals in the system.

STM bits [63:30] are rising edge trigger, where:

- 63:62 are inverted TRIGOUT[5:4] from cross-trigger interface (CTI)
- 61:60 are non-inverted TRIGOUT[5:4] from CTI
- 59:30 are PL events, stm_event[59:30]

STM bits [29:0] are level trigger, where:

- 29:0 are PL events, stm_event[29:0]

The STM supports the following functions.

- *printf* style debugging.
- Trace OS and application events.
- Emit diagnostic system information.
- Multiple channels for multiple processors to share without conflict with the other.
- Trace hardware events.
- Trace timestamp with global timestamp.
- Generates the MIPI STPv2.

The *Arm CoreSight STM-500 System Trace Macrocell Technical Reference Manual, r0p0ARM* [Ref 44] contains more details on the STM.

STM can provide support for up to 128 MasterIDs with 64k channels. The 16 MB aligned address space is allocated to the STM instrumentation trace for the channels. All of the MasterIDs overlap to the same 16 MB of address space. The channels are allocated by the software to the MasterID. The STM AXI slave is write-only. The reads to an STM AXI slave always returns OKAY with all-0 data.

The AXI AWADDRS[31:0] is used in the STM as follows:

- Bits [31:30] are not used.
- Bits [29:24] defines bits [5:0] of MasterID.
- Bits [23:8] define the Channel ID.
- Bits [7:0] define the address space of a single stimulus port.

The Zynq UltraScale+ MPSoC maps the interconnect AXI to the STM AXI-slave as shown.

```
STM_AWADDRS[23:0]  = AXI_AWADDR[23:0]       //Channels
STM_AWADDRS[29:24]  = function of MasterID  //MasterID
```

The MasterID can be mapped to the STM MasterID as listed in Table 39-9.

Table 39-9: **Zynq UltraScale+ MPSoC MasterID Mapped to the STM MasterID**

MasterID Indicates	STM_AWADDRS[29:24]	Notes
APU-CPU<n>	{MasterID[9:6], ACPU<n>}	APU request must be mapped to the STM MasterID.
RPU-CPU<n>	{MasterID[9:6], RCPU<n>}	AWID needs to be mapped to RPU#
Others	MasterID[9:4]	Use the upper 4 bits of the MasterID to uniquely identify the masters.

ATB Protocol

The ATB protocol is part of the AMBA 3 protocol family. The ATB protocol defines how trace information transfers between components in a trace system. The ATB is a common bus used by the trace components to pass format-independent trace data through a CoreSight system. A trace component or platform that has trace capabilities requires an ATB interface. The ATB interfaces are designated according to one of two functions.

- Master, an interface that generates trace data on the ATB interface.
- Slave, an interface that receives trace data on the ATB interface.

The ATB bus provides throughput to support the following debug conditions at typical trace debug settings.

- 4x APU MPCore PTM
- 1x APU MPCore ETM
- System trace macrocell (STM)

PL Fabric Trigger Macrocell

The PL fabric trigger macrocell (FTM) is a feed-through module for the cross-trigger signals to and from the PL. The FTM also provides a GPIO to/from the PL for simple communication. The FTM has the following features.

- A CoreSight component that is compliant with the Arm specification.
- Four trigger inputs and four trigger outputs.
- 32-bit general purpose inputs from the PL
- 32-bit general purpose outputs to the PL.
- Topology detection for trigger signals using integration mode.

Figure 39-7 shows the steps involved in FTM configuration.

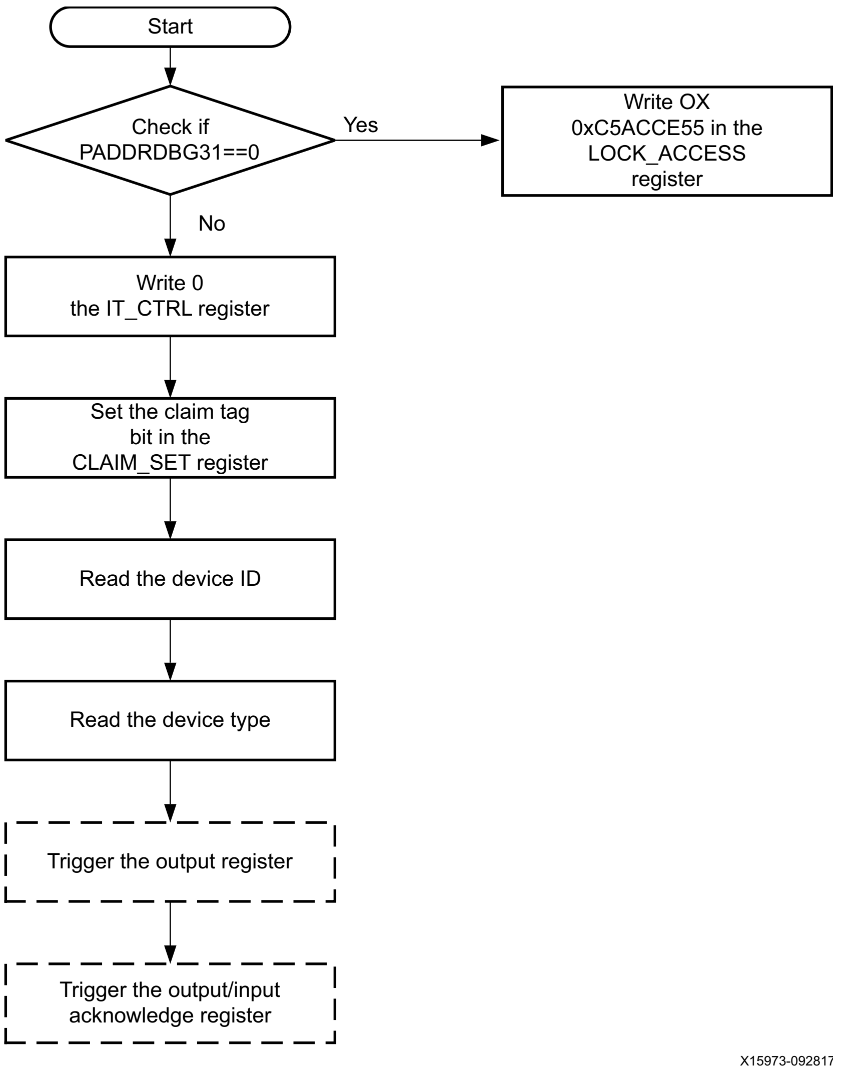

Start
Check if PADDRDBG31==0
Yes
Write OX 0xC5ACCE55 in the LOCK_ACCESS register
No
Write 0 the IT_CTRL register
Set the claim tag bit in the CLAIM_SET register
Read the device ID
Read the device type
Trigger the output register
Trigger the output/input acknowledge register
X15973-092817

Figure 39-7: **FTM Configuration Flowchart**

Trace Links

Funnels

The funnels combine packetized traces from several debug components onto the ATB bus. Each funnel includes memory-mapped registers for programming enable/disable, priority, etc. (refer to the FUNNEL3P and FUNNEL4P register sets).

There are three funnels:

- Funnel 0 connects to RPU in LPD (routed to FPD, FUNNEL3P).
- Funnel 1 connects to the APU MPCore (FPD, FUNNEL4P).
- Funnel 2 connects to Funnel 0, funnel 1, and STM (FPD, FUNNEL4P).

Replicator

The replicator duplicates a single ATB trace onto two ATB traces, with independent handshake (valid/ready) signals, so that the same trace can be fanned out to two trace sinks.

Trace Sinks

TPIU

The trace-port interface unit (TPIU) outputs packetized trace data to an off-chip analyzer. TPIU adds another layer of framing (packetization). The framing format is designed to make it easy for an analysis tool to re-synchronize to the frame boundary.

The TPIU acts as a bridge between the on-chip trace data with separate IDs to a data stream encapsulating IDs where required that is then captured by a trace port analyzer (TPA). The internal formatter inserts source ID signals into the data packet stream so that trace data can be re-associated with its trace source. It contains an asynchronous FIFO that enables trace data to be driven out at a speed that is not dependent on the on-chip bus clock. The internal trace-out block serializes formatted data before it goes off-chip. The TPIU includes a pattern generator unit that provides a simple set of defined bit sequences or patterns that can be output over the trace port and be detected by the TPA or other associated trace capture device (TCD). The TCD can use these patterns to indicate if it is possible to increase or decrease the trace port clock speed.

TMC

The trace memory controller (TMC) provides on-chip storage and buffering of trace data using RAMs. When configured as an embedded trace FIFO (ETF), the TMC functions as a FIFO to absorb bursts of traces, with the attached RAMs as the FIFO memory. When configured as an embedded trace router (ETR), the TMC can route the trace data into the PS interconnect, through an AXI bus, eventually reaching a large memory pool like external DDR or internal OCM.

There are two ETFs, one in the APU and the other one in the full-power domain. The reason for using an ETF in the APU is to absorb bursts of trace packets from the four CPUs, after they are combined and after the funnel in the APU.

There is one ETR, placed on one replicator output.

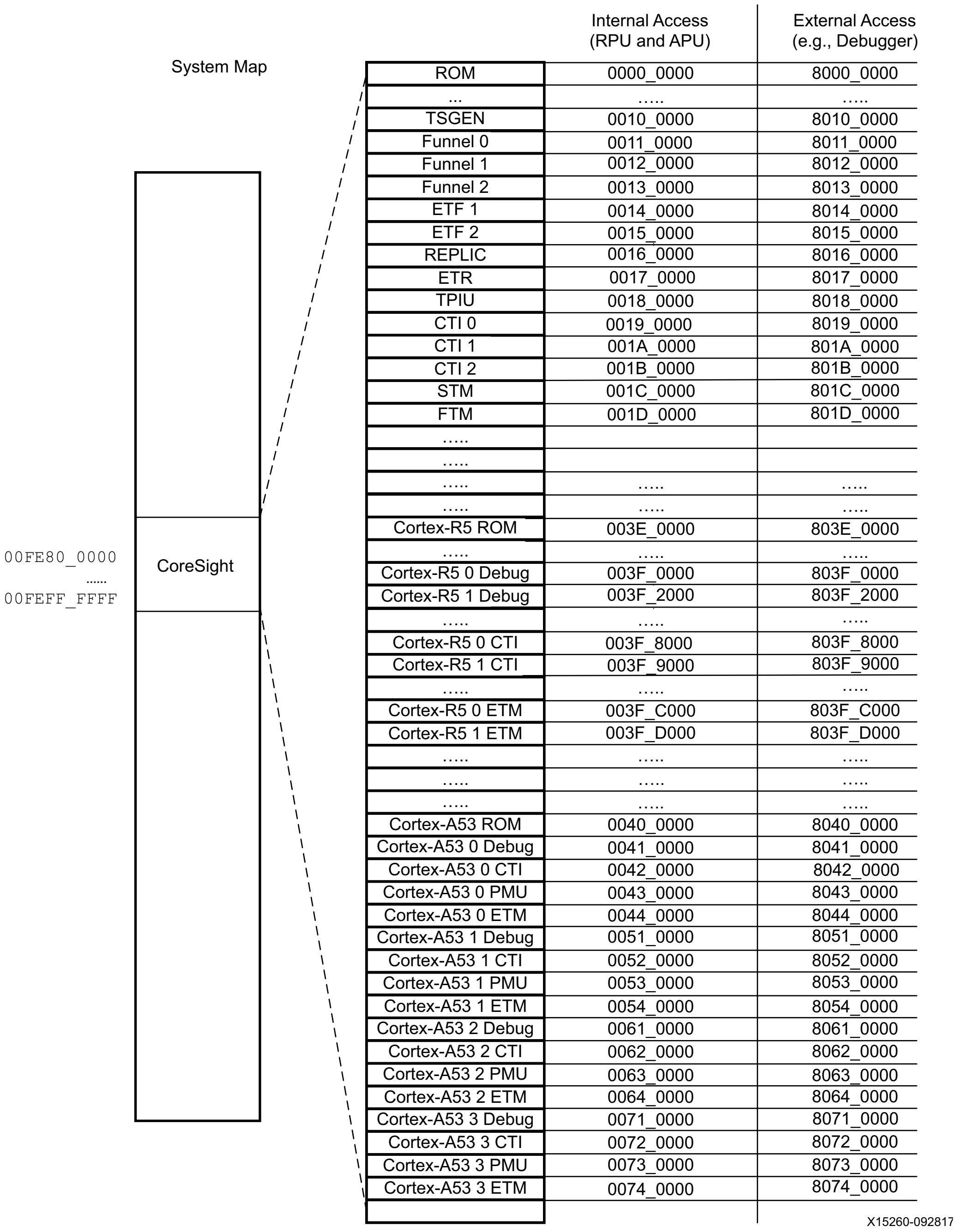
CoreSight Address Map

This section describes the CoreSight register architecture.

- Each component in the RPU is allocated a 4 KB register space; outside the RPU, each component is allocated a 64 KB register space.
- Within each component register space, there are fixed locations for fixed purposes.
- Address `0x0` on the debug APB is a ROM table, pointing to all other components.
- Each component can be accessed at two address locations on the debug APB.
    - When accessed from internal using system map, paddr[31] is forced to 0.
    - When accessed from external through JTAG, paddr[31] can be 1 or 0.
- paddr[31]=1 and paddr[31]=0 are subject to different authentications. This is useful for preventing rogue software on the RPU or APU MPCore from interfering with CoreSight components.

CoreSight components are allocated 8 MB of address space from `FE80_0000` to `FEFF_FFFF`. Figure 39-8 shows the detailed address space assignment for each debug component. References to the detailed address map within each component space are as follows.

DAP	Arm CoreSight SoC-400 Technical Reference Manual [Ref 39], chapter 3.
Timestamp	Arm CoreSight SoC-400 Technical Reference Manual [Ref 39], chapter 3.
Funnel	Arm CoreSight SoC-400 Technical Reference Manual [Ref 39], chapter 3.
TPIU	Arm CoreSight SoC-400 Technical Reference Manual [Ref 39], chapter 3.
CTI	Arm CoreSight SoC-400 Technical Reference Manual [Ref 39], chapter 3.
STM	Arm CoreSight STM-500 System Trace Macrocell Technical Reference Manual [Ref 44], chapter 3.
TMC	CoreSight Trace Memory Controller Technical Reference Manual [Ref 45], chapter 3.
Cortex-A53 ETM	Arm Cortex-A53 MPCore Processor Technical Reference Manual [Ref 46], chapter 13.
Cortex-R5 ETM	CoreSight ETM-R5 Technical Reference Manual [Ref 48], chapter 3.



System Map	Internal Access (RPU and APU)	External Access (e.g., Debugger)
ROM	0000_0000	8000_0000
...	.....	.....
TSGEN	0010_0000	8010_0000
Funnel 0	0011_0000	8011_0000
Funnel 1	0012_0000	8012_0000
Funnel 2	0013_0000	8013_0000
ETF 1	0014_0000	8014_0000
ETF 2	0015_0000	8015_0000
REPLIC	0016_0000	8016_0000
ETR	0017_0000	8017_0000
TPIU	0018_0000	8018_0000
CTI 0	0019_0000	8019_0000
CTI 1	001A_0000	801A_0000
CTI 2	001B_0000	801B_0000
STM	001C_0000	801C_0000
FTM	001D_0000	801D_0000
.....		
.....		
.....	.....	.....
.....	.....	.....
Cortex-R5 ROM	003E_0000	803E_0000
.....	.....	.....
Cortex-R5 0 Debug	003F_0000	803F_0000
Cortex-R5 1 Debug	003F_2000	803F_2000
.....	.....	.....
Cortex-R5 0 CTI	003F_8000	803F_8000
Cortex-R5 1 CTI	003F_9000	803F_9000
.....	.....	.....
Cortex-R5 0 ETM	003F_C000	803F_C000
Cortex-R5 1 ETM	003F_D000	803F_D000
.....	.....	.....
.....	.....	.....
.....	.....	.....
Cortex-A53 ROM	0040_0000	8040_0000
Cortex-A53 0 Debug	0041_0000	8041_0000
Cortex-A53 0 CTI	0042_0000	8042_0000
Cortex-A53 0 PMU	0043_0000	8043_0000
Cortex-A53 0 ETM	0044_0000	8044_0000
Cortex-A53 1 Debug	0051_0000	8051_0000
Cortex-A53 1 CTI	0052_0000	8052_0000
Cortex-A53 1 PMU	0053_0000	8053_0000
Cortex-A53 1 ETM	0054_0000	8054_0000
Cortex-A53 2 Debug	0061_0000	8061_0000
Cortex-A53 2 CTI	0062_0000	8062_0000
Cortex-A53 2 PMU	0063_0000	8063_0000
Cortex-A53 2 ETM	0064_0000	8064_0000
Cortex-A53 3 Debug	0071_0000	8071_0000
Cortex-A53 3 CTI	0072_0000	8072_0000
Cortex-A53 3 PMU	0073_0000	8073_0000
Cortex-A53 3 ETM	0074_0000	8074_0000

Figure 39-8: **CoreSight System Debug Address Map**

Clocks, Reset, and Power Domains

JTAG and Debug Clocks

Most of the components in the CoreSight debug logic have only one clock input. Power domain crossings use APB asynchronous bridges, and ATB asynchronous bridges. Table 39-10 lists the JTAG and debug clocks.

Table 39-10: **Clocks**

Clock Name	Power Domain	Source	Where Used
JTAG test clock	Aux	TCK input pin	JTAG-DP
AHB_CLK	LPD	SysOsc (internal ring oscillator)	JTAG-DP, AXI-AP, APB-AP, and JTAG-AP
DBG_LPD_CLK	LPD	CRL_APB.DBG_LPD_CTRL	RPU debug logic and funnel
DBG_FPD_CLK0	FPD	CRF_APB.DBG_FPD_CTRL	APU debug logic
DBG_FPD_CLK1	FPD		STM, funnel, ETF, ETR, TPIU, CTIs, and CTMs
DBG_TSTMP_CLK	FPD		Timestamp network
DBG_TRACE_CLK	FPD	CRF_APB.DBG_TRACE_CTRL	TPIU, double data rate
PL_PS_TRACE_CLK	FPD	PL input	TPIU, double data rate

Debug Logic Resets

Table 39-11 lists the reset signals generated by the CRL_APB.RST_LPD_DBG register in the PS reset subsystem.

Table 39-11: **Debug Logic Resets**

Bit Field	Power Domain	Description
rpu_dbg{0, 1}_reset	LPD	Reset debug logic in the RPU cores
dbg_lpd_reset	LPD	Reset debug logic in the LPD
dbg_fpd_reset	FPD	Reset debug logic in the FPD

JTAG Resets

The JTAG control unit is reset by a system (SRST) or a POR.

Power

Power-up Request and Acknowledge

The JTAG debug port (JTAG-DP) provides a few signals for conveniently requesting for power-on from the debugger. Further details are provided in the *CoreSight SoC-400 System Design Guide* [Ref 41]. The following is a summary.

Powering up both is required as the first step of debugging via CoreSight. Each pair consists of a request and an acknowledgment, and can be used to communicate to the PMU. Although they were originally intended to be used for *debug* power domain and *system* power domain, they are used instead as follows.

- CDBGPWRUPREQ, CDBGPWRUPACK
  - Controlled by CTRL/STAT[29:28] register bits of JTAG-DP
  - Used to power up the two Cortex-R5F MPCore CPU islands
- CSYSPWRUPREQ, CSYSPWRUPACK
  - Controlled by CTRL/STAT[31:30] register bits of JTAG-DP
  - Used to power up the entire FPD

Power Domains

The debug module is supported in three power domains. Refer to Table 39-12: JTAG and CoreSight Split Between Power Domains for more details. Whenever the DAP controller is available, the system is ready to debug the LPD; because the DAP is in the LPD. If the LPD is powered off, the DAP cannot be used.

A request needs to passed through the DAP (to the PMU_GLOBAL register) to power on the FPD and start an FPD debug.

JTAG and Debug Logic Power Supplies

The CoreSight debug system is spread across three power domains. Although all power-domains should be turned on, the basic JTAG functions work as long as the low-power domain (LPD) and PL power domain (PLPD) are active. The PL TAP controller is required to run a boundary scan (BSCAN). The DAP controller has components in the full-power domain (FPD) for APU debug. See Table 39-12.

Table 39-12: **JTAG and CoreSight Split Between Power Domains**

LPD	FPD	PLPD
PS TAP controller DAP controller Cortex®-A53 CTI, CTM, and ETM Debug APB (path to the Cortex-A53 debug APB) ATB and Funnel 0	All CTI/CTM except the Cortex-A53 CTI/CTM. Debug APB except branch to the Cortex-A53. All of trace related components: • Cortex-A53 ETM • STM • ATB and Funnels 1 and 2 • TMC • TPIU	PL TAP controller

I/O Signals

The JTAG test and TPIU debug signals are listed in Table 39-13 and Table 39-14.

JTAG Interface Signals on MIO

The PS JTAG interface is provided on a dedicated set of signals, which are listed in Table 2-3. The PJTAG interface signals are listed in table Table 39-13.

Table 39-13: **PJTAG I/O Interface**

MIO					EMIO
Signal Name	**Index[1]**	**Pins**	**I/O**	**Default Input to Controller**	**TBD**
TCK	3	0, 12, 26, 38, 52, 58	I	0	
TDI	0	1, 13, 27, 39, 53, 59	I	0	
TDO	1	2, 14, 28, 40, 54, 60	O	~	
TMS	2	3, 15, 29, 41, 55, 61	I	0	

Notes:
1. The index numbers are shown in Table 28-1.

TPIU Data Output on MIO and EMIO

The CoreSight TPIU data output signals are independently routed to the MIO using the IOU_SLCR.MIO_PIN_xx registers. Unrouted signals default to the EMIO interface. The TPIU output interface signals are listed in Table 39-14.

The MIO pins are described in the Chapter 28, Multiplexed I/O. The EMIO signals are described in the *Zynq UltraScale+ MPSoC Processing System: LogiCORE IP Product Guide* PG201 [Ref 5].

Table 39-14: **CoreSight TPIU I/O Interfaces**

MIO				EMIO	
Signal Name	**Index**[1]	**Pin**	**I/O**	**Signal Name**	**I/O**
dbg_trace_clk	0	0, 38, 52	O	pl_ps_trace_clk	I
dbg_ctl	1	1, 39, 53	O	ps_pl_tracectl	O
DQ[0]	2	2, 40, 54	O	ps_pl_tracedata[0]	O
DQ [1]	3	3, 41, 55	O	ps_pl_tracedata[1]	O
DQ [2]	4	4, 42, 56	O	ps_pl_tracedata[2]	O
DQ [3]	5	5, 43, 57	O	ps_pl_tracedata[3]	O
DQ [4]	6	6, 26, 58	O	ps_pl_tracedata[4]	O
DQ [5]	7	7, 27, 59	O	ps_pl_tracedata[5]	O
DQ [6]	8	8, 28, 60	O	ps_pl_tracedata[6]	O
DQ [7]	9	9, 29, 61	O	ps_pl_tracedata[7]	O
DQ [8]	10	10, 30, 62	O	ps_pl_tracedata[8]	O
DQ [9]	11	11, 31, 63	O	ps_pl_tracedata[9]	O
DQ [10]	12	12, 32, 64	O	ps_pl_tracedata[10]	O
DQ [11]	13	13, 33, 65	O	ps_pl_tracedata[11]	O
DQ [12]	14	14, 34, 66	O	ps_pl_tracedata[12]	O
DQ [13]	15	15, 35, 67	O	ps_pl_tracedata[13]	O
DQ [14]	16	16, 36, 68	O	ps_pl_tracedata[14]	O
DQ [15]	17	17, 37, 69	O	ps_pl_tracedata[15]	O
~	~	~	~	ps_pl_tracedata[16:31]	O

Notes:

1. The index numbers are shown in Table 28-1.

MBIST, LBIST, and Scan Clear (Zeroization)

There are three low-level hardware test and clear mechanisms for system logic and memory:

- Memory built-in self-test (MBIST) tests the RAM elements used to store values
- Logic BIST (LBIST) tests the logic used for control
- Scan clear removes system state

The MBIST always runs after the power-on rest (POR). The MBIST can also be initiated by the system software calling the platform management unit user firmware (PMU FW). The LBIST only runs after a POR and only if the LBIST_EN eFUSE is programmed. The PMU scan clear only runs after a POR. The LPD and FPD scan clear only works when LPD_SC and FPD_SC eFuses are programmed. If the eFuses are programmed the LPD and FPD scan clear are always running after a POR or after a secure lockdown only.

Running MBIST, LBIST, or scan clear will cause loss of state. Therefore, care must be taken when initiating BIST and clear functions during system operation.

SEU Occurrences

When an errant single-event upset (SEU) causes a failure, the system can be rebooted, or the system element can be tested again. If the element continues to fail, the fault is permanent.

MBIST

The MBIST tests RAM memory arrays. The arrays are tested during the hardware boot time and on demand using the PMU user firmware.

Hardware Boot Process

The PMU FW can initiate MBIST operations on the LPD and FPD; however, the LPD process does not affect the PMU RAM in the LPD. When a memory is tested or cleared using an MBIST operation, other parts of the system can be functioning.

MBIST Interfaces to System Elements

For most of the system elements, RAM is accessed directly by the MBIST hardware and it keeps the functional RAM interfaces in their reset state while the RAM is accessed. For a few system elements, including the APU core processors, the RAM is accessed by the MBIST hardware through the RAM's functional paths used by system software. In these cases, the functional units must be operational (clocked and not held in reset).

PMU User Firmware Controls

Registers are used to control and set the status of the MBIST memory controllers via the PMU global register set. This register set can be protected by a 64 KB aperture of the Xilinx peripheral protection unit (XPPU). There are five control and status registers:

- MBIST_RST controls the reset signal (rw)
- MBIST_PG_EN controls the PG_EN signal (rw)
- MBIST_SETUP controls the SETUP signal (rw)
- MBIST_DONE indicates when the test is completed (ro)
- MBIST_GOOD indicates the results of the test (ro)

To initiate an MBIST operation, set the bit in all three trigger registers: MBIST_RST, MBIST_PG_EN, and MBIST_SETUP. When the operation is finished, the software clears the associated bits in all three trigger registers. The MBIST_DONE bit goes High when the operation is finished. MBIST_GOOD provides the status of the operation: 0 (failure) or 1 (success). MBIST_DONE and MBIST_GOOD are read-only registers and cleared by the hardware when the trigger registers are cleared.

Table 39-15 lists the system elements that are tested by the MBIST and the bit assignments for the control and status registers.

Table 39-15: **MBIST Control Register Bit Fields**

Bit	Bit Field	System Element
0:1	CAN{0,1}	CAN {0,1} controller
2:5	GEM{0:3}	GEM {0:3} controller
6	IOU	IOP peripherals
7	RPU	RPU cores
8	RPU_TIEOFF_WRAPPER	LPD less RPU
9:10	USB{0,1}	USB controller {0, 1}
11	AFI_LPD	S_AXI_LPD interface with FIFO memory
12	OCM	OCM memory
13	PSS_CORE_TOP	PS top core (includes XPPU and APM)
14	FPD	FPD(1)
15	AFI_0	S_AXI_HPC0_FPD interface with FIFO memory
16	AFI_1	S_AXI_HPC1_FPD interface with FIFO memory
17:20	AFI_{2:5}	S_AXI_HP{0:3}_FPD interface with FIFO memory
21	APU	APU MPCore
22:25	ACPU_{0:3}	APU core {0:3}
26	DDR	DDR controller

Table 39-15: **MBIST Control Register Bit Fields *(Cont'd)***

Bit	Bit Field	System Element
27	GPU	GPU controller
28	M400_0	GPU pixel processor 0
29	M400_1	GPU pixel processor 1
30	SIOU	High-speed serial I/O
31	PCIE	PCIe controller

1. This bit is used to cover the rest of the memories in the FPD such as SMMU, debug, GDMA, and so on.

LBIST

The LBIST covers more than 90% of the system units. The LBIST operations are run once during the hardware boot. If an LBIST operation fails, a single status flag is raised in the JTAG status register, and the system stalls. This failure can be detected by reading the JTAG status [11] bit field; however, it cannot be determined which system element failed the LBIST operation.

LBIST operations require approx. 250 mA supply (LPD+FPD). After the LBIST goes to the scan clear (zeroization) state, the supply current is released back to the normal system.

For increased safety against an SEU, the LBIST trigger signal from the hardware can be gated off during normal system operation using the PMU global register bit SAFETY_GATE [LBIST_Enable].

The primary goal for the LBIST is to detect latent faults at boot time. Using the LBIST, these blocks are checked for latent faults:

- Lock-step checkers such as the Cortex-R5F processor, PMU, and configuration security unit (CSU)
- ECC generation and checking:
  - On-chip memory (OCM), CSU, and PMU RAM
  - Tightly coupled memory (TCM) and cache memory controllers
- Xilinx memory protection unit (XMPU)
- Common clock monitoring
- PS SYSMON interface
- PMU logic
- Error monitoring logic
- Reset controller

Note: LBIST coverage does not include the logic in the crypto engines.

PL Configuration Signals

The PS_INIT_B and PS_PROG_B signals should be considered input/output during the LBIST operation. These pins should be connected as open drain with a 4.7k pull-up resistor to VCCO_PSIO [3] as described in the *UltraScale Architecture PCB Design User Guide* (UG583) [Ref 15]. Driving these signals can cause LBIST failures.

LBIST Boot Sequence

1. The PS_POR_B reset signal must be asserted during the power-up sequence. Voltage must remain stable during boot and normal operation.
2. To use the FPD, the VCC_PSINTFP power must be valid and stable before deasserting PS_POR_B; otherwise, the LBIST disables the FPD during the boot process making it unavailable. The FPD remains disabled until the next POR with valid FPD power.
3. The PS_MGTRAVCC power must be valid and stable during the LBIST operation for the transceivers to be operational; otherwise they are disabled by the LBIST.
4. The LBIST controllers are in the LPD and operate before the FPD is enabled.
5. The PS_INIT_B signal is internally driven low during the LBIST operation and the signal must not be externally driven high; otherwise, the LBIST operation fails.
6. The PS_PROG_B signal must remain high during the LBIST operation and must not be externally driven low; otherwise, the LBIST operation fails.
7. The LBIST is activated when PS_POR_B is deasserted. Asserting PS_POR_B stops the LBIST operations.

IMPORTANT: *Care must be taken when initiating LBIST operations. The system element must be powered-up, clocked, and held in reset. When the LBIST operation is completed, the logic is left in its POR state.*

The LBIST failure state recovers only after the power cycle or PS_POR_B is asserted. If the LBIST error is not recoverable, there is no masking of the error or the LBIST eFUSE option once it is enabled.

Scan Clear (Zeroization)

Zeroization is a process in which zeros are shifted through all of the storage elements and then verified that the shift occurred correctly. This is achieved using the MBIST and scan clear functionality. The scan clear engines can only be controlled by the PMU and CSU processors through their direct interfaces to the engines. Other processors can request the PMU through the LOGCLR_TRIG register to start any specific scan clear engines.

Every power island and power domain have scan clear engines. The PMU and CSU blocks have separate scan clear engines even though they are not power islands. The PMU scan clear is triggered only on POR, and the CSU scan clear can only be triggered by the PMU.

The LPD and FPD scan clear operations only run after a POR and during secure lockdown if the LPD_SC and FPD_SC eFUSE are programmed. The PMU scan clear is a mandatory security operation in the boot flow, whereas the LPD and FPD scan clear operations are optional. See Boot Flow in Chapter 11 for details on PMU security operations during boot flow.

Control Registers

The scan clear control and status registers are in the PMU local register set and are only accessible by the PMU processor. These control and status registers are used by the PMU user firmware.

- LOGCLR_TRIG starts the scan clear operations (wo)
- LOGCLR_ACK indicates completion of the scan clear operation (ro)
- SERV_LOGCLR_ERR used by the PMU code to log scan clear errors (rw)

When a scan clear engine is started, the completion status signal from the engine transitions from 1 to 0. This signal, which is routed directly to a PMU LOGCLR_ACK register, communicates the completion status of the engine to the PMU. When a scan clear engine finishes its operation, its completion status bit toggles from 0 to 1 generating an interrupt to the PMU. The pass/fail status of the clearing operation can be checked by the bits in the PMU LOGCLR_STATUS global register that are directly driven by the pass/fail status of the engine.

The CSU only starts scan clear engines under a security lock-down scenario and there is no functional requirement for the CSU to check the pass/fail status or the completion status of the clearing operation.

For increased safety, the scan clear trigger signal from the hardware can be gated off during normal system operation using the PMU_Global.SAFETY_GATE [Scan_Enable] register bit.

The PMU user firmware can accumulate failures in the PMU_Local.SERV_LOGCLR_ERR register (reset only by a POR). This register is write protected using the PMU_Global.SAFETY_GATE [PMU_LOGCLR_Enable] register bit.

The bit assignments for the trigger and acknowledge registers are listed in Table 39-16.

Table 39-16: **Scan Clear TRIG and ACK Register Bit Fields**

Bit	Bit Field	System Element
0	ACPU0	APU core 0
1	ACPU1	APU core 1
2	ACPU2	APU core 2
3	ACPU3	APU core 3
6	PP0	GPU pixel processor 0
7	PP1	GPU pixel processor 1
10	RPU	RPU
12	USB0	USB controller 0
13	USB1	USB controller 1
16	LP	LPD except PMU, RPU, and USBs
17	FP	FPD except APU cores, and GPU

Appendix A

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Xilinx Support.

For a glossary of technical terms used in Xilinx documentation, see the Xilinx Glossary.

Solution Centers

See the Xilinx Solution Centers for support on devices, software tools, and intellectual property at all stages of the design cycle. Topics include design assistance, advisories, and troubleshooting tips.

Documentation Navigator and Design Hubs

Xilinx® Documentation Navigator provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open the Xilinx Documentation Navigator (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 the Xilinx Documentation Navigator, click the **Design Hubs View** tab.
- On the Xilinx website, see the Design Hubs page.

Note: For more information on Documentation Navigator, see the Documentation Navigator page on the Xilinx website.

References

1. *Zynq UltraScale+ MPSoC Product Overview* (DS891)
2. *Zynq UltraScale+ MPSoC Data Sheet: DC and AC Switching Characteristics* (DS925)
3. *Zynq UltraScale+ MPSoC Software Developer's Guide* (UG1137)
4. *Zynq UltraScale+ MPSoC Register Reference* (UG1087)
5. *Zynq UltraScale+ MPSoC Processing System: LogiCORE IP Product Guide* (PG201)
6. *UltraScale Architecture System Monitor User Guide* (UG580)
7. *Zynq UltraScale+ MPSoC Packaging and Pinout User Guide* (UG1075)
8. *UltraScale Architecture SelectIO Resources User Guide* (UG571)
9. *UltraScale Architecture Clocking Resources User Guide* (UG572)
10. *UltraScale Architecture Memory Resources User Guide* (UG573)
11. *UltraScale Architecture Configurable Logic Block User Guide* (UG574)
12. *UltraScale Architecture GTH Transceivers User Guide* (UG576)
13. *UltraScale Architecture GTY Transceivers User Guide* (UG578)
14. *UltraScale Architecture DSP Slice User Guide* (UG579)
15. *UltraScale Architecture PCB Design User Guide* (UG583)
16. *OS and Libraries Document Collection* (UG643)
17. *Zynq UltraScale+ MPSoC: Embedded Design Tutorial* (UG1209)
18. *AXI DMA v7.1 LogiCORE IP Product Guide: Vivado Design Suite* (PG021)
19. *System Management Wizard v1.3 LogiCORE IP Product Guide* (PG185)
20. *Programming BBRAM and eFUSEs Application Note* (XAPP1319)
21. *Zynq-7000 SoC: Technical Reference Guide* (UG585)
22. *AXI Performance Monitor LogiCORE IP Product Guide* (PG037)
23. *Xilinx Standalone Library Documentation: OS and Libraries Document Collection* (UG643)
24. *Vivado Design Suite: Embedded Processor Hardware Design* (UG940)
25. Zynq UltraScale+ MPSoC - 64-bit DDR Access with ECC technical article
26. *Zynq UltraScale+ RFSoC RF Data Converter LogiCORE IP Product Guide* (PG269)
27. *Zynq UltraScale+ MPSoC Processing System LogiCORE IP Product* Guide (PG201)
28. *UltraScale+ Devices Integrated Block for PCI Express Product Guide* (PG213)
29. *PHY Controller LogiCORE IP Product Guide* (PG230)

30. *Integrated Interlaken 150G LogiCORE IP Product Guide* (PG169)
31. *H.264/H.265 Video Codec Unit LogiCORE IP Product Guide* (PG252)
32. *Developing Tamper-Resistant Designs with Zynq UltraScale+ Devices* (XAPP1323)
33. *UltraScale Architecture Configuration User Guide* (UG570)
34. *External Secure Storage Using the PUF Application Note* (XAPP1333)
35. *Internal Programming of BBRAM and eFUSEs Application Note* (XAPP1283)
36. *Bootgen User Guide* (UG1283)
37. *Accelerating Cryptographic Performance on the Zynq UltraScale+ MPSoC* (WP512)
38. *Isolation Methods in Zynq UltraScale+ MPSoCs* (XAPP1320)

Arm References

39. Arm® CoreSight® SoC-400 Technical Reference Manual, r3p1ARM document number DDI 0480F.
40. Arm CoreSight SoC-400 User Guide, r3p1ARM document number DUI 0563F.
41. Arm CoreSight SoC-400 System Design Guide, r3p1ARM document number DGI 0018E.
42. Arm CoreSight SoC-400 Implementation Guide, r3p1ARM document number DII 0267F.
43. Arm CoreSight SoC-400 Integration Manual, r3p1ARM document number DIT 0037E.
44. Arm CoreSight STM-500 System Trace Macrocell Technical Reference Manual, r0p0ARM document number DDI 0528A.
45. CoreSight Trace Memory Controller Technical Reference Manual, r0p1ARM document number DDI 0461B.
46. Arm Cortex®-A53 MPCore Processor Technical Reference Manual, r0p2ARM document number DDI 0502D.
47. ARM Cortex-R5 and Cortex-R5F Technical Reference Manual, r1p1 document number DDI 0460C (ID021511)
48. CoreSight ETM-R5 Technical Reference Manual, r0p0ARM document number DDI 0469A.
49. Arm CoreSight Architecture Specification, v2.0ARM document number IHI 0029D.
50. Arm System Memory Management Unit Architecture Specification, SMMU Architecture version 2.0, Arm IHI 0062C (ID091613).
51. Arm CoreSight Components
52. Arm CoreSight Architecture
53. CoreSight Trace and Debug

54. Arm Mali GPU Application Optimization Guide

PCIe References

55. PCI-SIG ASPM Optionality ECN

Additional References

56. Recommendation for Applications Using Approved Hash Algorithms
57. SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions
58. Zynq UltraScale MPSoC Cache Coherency
59. System Software Mutexes on Zynq UltraScale

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
1/04/2023	2.3.1	Version 2.3 revision history updated to expand on changes made.
9/15/2022	2.3	Chapter 1: Corrected the RPU block connection in Figure 1-1. Revised Table 1-2 to include IDCODEs for ZU1, ZU42, ZU65, and ZU67 devices. Chapter 2: Revised Table 2-1 to include PS_REF_CLK for pin VCCO_PSIO[0:3]. Table 2-2 revised to include a reference to the T_{POR} specification. Updated the description for HP in Table 2-13. Chapter 3: Added note for clarification to System Memory Virtualization Using SMMU Address Translation using SMMU address translation. Chapter 4: Added reference to switches in Figure 4-1. Chapter 6: Revised Figure 6-2 to include error aggregator module block. Table 6-12 updated to define Bits [21:18] as PMU firmware interrupt bits. Chapter 7: Renamed PS SYSMON signal to PS SYSMON sensor in Figure 7-1. Chapter 8: Updated Introduction to include certification for ISO 26262:2018 and IEC 61508:2010 based applications for LPD, FPD, and PL domains. Chapter 9: Note added to Table 9-3 to address PL Sysmon initiation. Chapter 10: Corrected address from 1 GB 0x400_0000 to 1 GB 0x4000_0000 in Figure 10-1. Table 10-5 updated to include GPV register information. Added information on system software mutexes to System-level Control Registers. Chapter 11: Updated the gray key in Table 11-4. Added a description for error code 0x3F in Table 11-9. Missing ZU1 device details added to Table 11-10. Updated device names (XCZUx to ZUx) and removed in details of ZU58 and ZU59 parts in Table 11-10. ROM support updated in Table 11-11. Chapter 12: Updated Key Management to provide more information on the device key. Replaced PPK0_0 and PPK1_0 with PPK{0,1} and size of PPK1_0 changed from 32 to 384 in Table 12-13. Integration and Test Support (BH RSA Option) updated on BH RSA option. Chapter 13: Defined IDCICR in Table 13-1. IPI message buffers for channel numbers 0, 1 and 2 corrected in Table 13-3. Chapter 15: Corrected IOP Bus Masters section and added notes to expand on the AXI Performance Monitor section. Master AXI reset signal moved inside the AXI master block in Figure 15-2. The term position replaced with port and included the information about Qos_Control_Register_S2 in Table 15-6. Chapter 16: Corrected connection from LPD Main Switch to RPU switch for port S0 of the DDR memory controller in Figure 16-1. Updated XMPU Regions, Region Checking Operation, Configuration, and XPPU Self-Protection. Corrected System Masters for DDR XMPU0 to RPU in Table 16-5. Included SMMU TCU and CCI in Table 16-13. Value written to OCM_XMPU_CFG. R00_END register is changed from 0F_D0CFh to 0F_FFCFh in the Program the OCM XMPU section. Chapter 17: Revised Figure 17-1 to correct the connection from the LPD main switch to port S0 of the DDR memory controller. Table 17-1 updated to state 3DS DDR4 is not supported in PS. XMPU0 connection updated from RPU and LPD Masters to RPU

Date	Version	Revision
9/15/2022	2.3 *(cont'd)*	Figure 17-2. Updated Power and Reset, ECC Programming Model, and Programming Topics. Chapter 19: Updated Introduction to address memory to I/O buffer transfers. Added new section Total Transferred Bytes Overflow. Chapter 20: Assign MIO Pin to CAN TX Output section 0 has been replaced by 1 for BANK1_CTRL6 [22] bit in step 3 of Programming Example – Assign MIO Pin to CAN TX Output. Chapter 23: Corrected the data written to TX FIFO from 32 bits to 8 bits in Table 23-13. Corrected AR7358 hyperlink. Chapter 24: Removed descriptions for IOP transactions in the DMA–AXI Master section. Chapter 25: Removed descriptions for IOP transactions in the AXI Master Interface. Chapter 26: FIFO buffer size is hard coded and does not have any software control, FIFO Buffer section revised to address this. Removed IOP transaction descriptions in the PIO/DMA Controller section. Corrected the Block Buffer section to address the size of the hard coded block buffer. Receive Clock Tap Delay section updated with sd{0,1}_itapdlyena. SD Interface Voltage Translation updated to include a note addressing the SDx_DIR0 and SDx_DIR1 direction signals. Revised Figure 26-6 to include the voltage level shifter. Table 26-15 updated to reflect the SD3.0 specification. Chapter 28: Updated ulpi_clk_in_0 to ulpi_clk_in_1 and changed the block color to purple from green in Table 26-3. Chapter 29: Updated PS_MGTREFCLK0, PS_MGTREFCLK1,PS_MGTREFCLK2, and PS_MGTREFCLK3 in the Reference Clock Network section. Added note to Table 29-2 to address GTR reference clock PS_MGTREFCLKP/N. Chapter 31: PHY Loopback updated to address the external USB2.0 ULPI PHY RESET (ULPI_PHY_RESETB) signal. Chapter 33: Removed DP_MAIN_STREAM_POLARITY information from Table 33-13. Chapter 34: GEM does not support PL checksum offloading, GEM Features section updated accordingly. Corrected tx_r_valid signal in Figure 34-3. Removed descriptions for IOP transaction in the DMA Controller section. Note added to address RX pause frame not being supported in the PFC Pause Frame Reception section. Configure the PHY section updated to replace IOU_SWITCH_CLK with LPD_LSBUS_CLK. Removed duplicate step from the Receive Buffer Descriptor List section. Chapter 35: Connection from the LPD Main Switch to port S0 changed to RPU switch to port S0 of the DDR memory controller in Figure 35-1, Figure 35-4, Figure 35-5, Table 35-6. Corrected interface S_AXI_HPM{0,1}_FPD to M_AXI_HPM{0,1}_FPD in Figure 35-2. Added description to interface S_AXI_HPC{0,1}_FPD in Table 35-1. Table 35-6 updated to include interrupts reference. Chapter 37: Corrected value of CHKRx_CLKA_LOWER [thrshld} in Clock Monitor Programming Example from 0000_FD76Fh to 0000_FD76h. Revised Table 37-7 to correct addresses for SPI(0/1)_REF_CTRL and UART(0/1)_REF_CTRL. DP_VIDEO_REF_CTRL [SRCSEL] reset state corrected to DPLL from VPLL. Chapter 39: Information related to critical-function blocks removed from LBIST section and a note has been added to address the logic in the crypto engines.

Date	Version	Revision
12/04/2020	2.2	Chapter 1: Revised Figure 1-3, Table 1-2, and Table 1-4. Chapter 2: Table 2-1 and Table 2-2. Chapter 4: Revised Table 4-1 and TCM Access from a Global Address Space. Chapter 6: Revised. Chapter 11: Revised. Chapter 12: Revised. Chapter 13: Revised. Chapter 14: Revised Event Streams, Generic Timer Programming, Event Control Timer Operation, System Watchdog Timers, Table 14-27, Chapter 15: Revised Figure 15-1, Programming, Table 15-2, Register Overview, Table 15-6 Chapter 16: Revised Table 16-2, added Table 16-3 Chapter 17: Revised section and added Dynamic DDR Configuration section. Chapter 18: Revised Features. Chapter 19: Revised Simple DMA Mode, Scatter Gather DMA Mode and Rate Control sections. Chapter 23: Revised the Clocks section and Table 23-7. Chapter 24: Revised Quad-SPI Feedback Clock, Table 24-3, Quad-SPI Tap Delay Values Chapter 25: Revised Figure 25-2 and Figure 25-3. Chapter 26: Revised Table 26-11. Chapter 28: Revised Default Logic Levels section. Added Table 28-3. Chapter 29: Revised Features, Table 29-4, Receiver Termination Chapter 30: Revised the Bridge Initialization section. Chapter 31: Revised USB Controller Features, Register Overview, Table 31-40 Chapter 32: Revised Features and Figure 32-5. Chapter 33: Revised Table 33-2, Audio Metadata, DisplayPort DMA, CRC Field, and Table 33-23.Chapter 34: SGMII, 1000BASE-SX, or 1000BASE-LX, RX Buffers, IEEE Std 1588 Time Stamp Unit, and Table 34-14. Added Precision Time Protocol via EMIO and Priority Queuing Chapter 35: Revised Table 35-2 and Table 35-4. Chapter 37: Revised Choosing a Programmable Logic Interface and Table 37-3. Chapter 38: Revised Table 38-1 and RPU Reset Sequence. Added PL Configuration Reset. Chapter 39: Revised Figure 39-2, MBIST, LBIST, and Scan Clear (Zeroization), and Table 39-16
8/21/2019	2.1	Chapter 12: Revised Encrypt Only Secure Boot Details section. Chapter 15: Revised the QoS Regulator section. Updated Figure 15-1.

Date	Version	Revision
7/22/2019	2.0	Chapter 1: Revised the JTAG IDCODE section. Chapter 4: Revised the RPU Pin Configuration section. Chapter 5: Revised the Programming the GPU. Note updated. Chapter 6: Revised Full-Power Operation Mode section, Table 6-16 and added Table 6-14. Chapter 11: Revised Boot Modes section, PL Bitstream section and added Table 11-11. Chapter 12: Revised Secure Lockdown, SHA-3/384, Boot Operation, Encrypt Only Secure Boot Details, and Loading Bitstreams sections. Chapter 14: Revised APU Core Private Physical and Virtual Timers and Watchdog Enabled on Reset sections. Chapter 15: Revised Programming and PS Instances sections. Chapter 17: Revised Table 17-1 and PHY Description section. Chapter 20: Revised RX/TX Bit Timing Logic section. Chapter 24: Revised Reference Clock and Quad-SPI Interface Clocks, Quad-SPI Feedback Clock, and Quad-SPI Tap Delay Values sections. Chapter 26: Revised Tuning Unit, RX Clock Delay Unit, and Receive Clock Tap Delay sections. Chapter 29: Revised the EyeScan Module section. Chapter 30: Revised the Features section, added a note in Integrated Block for PCI Express section, and added an Important note in Single CPU Control section. Chapter 33: Revised Table 33-2. Chapter 34: Added a note under Table 34-19. Chapter 35: Revised the LPD-PL Interfaces section. Chapter 39: Revised the Toggle Detect on PSJTAG, PL TAP Controller, Trace Debug, Security, Components sections and updated Table 39-6 and Table 39-15. Appendix A: Arm References updated.

Date	Version	Revision
1/17/2019	1.9	Chapter 1: Revised Figure 1-2. Chapter 2: Revised Table 2-2. Chapter 3: Revised System Virtualization and Power Modes sections. Chapter 4: Revised Tightly Coupled Memory Address Map section. Chapter 6: Revised Figure 6-1, Figure 6-2, Table 6-3, MIO Pin Considerations, Table 6-12, and MIO Signals. Chapter 7: Revised. Chapter 8: Entire chapter revised. Chapter 9: Revised Table 9-1, Register Sets, and Table 9-7. Chapter 11: Revised Table 11-1 and Table 11-4. Chapter 12: Revised PUF Operations, Figure 12-7, Programming AES-GCM Engine, RSA Accelerator, Security Related eFUSEs, Secure Boot Introduction, Secure Boot Summary, Loading Bitstreams, Figure 12-16, and Secure Boot Image Format. Added Device DNA Identifiers, Initialization Vector Register, Secure Non-Volatile Storage, and Enhanced SPK Revocation. Added note to Table 12-13. Chapter 13: Revised SPI Interrupt Sensitivity. Chapter 14: Revised TTC Counter Features, TTC Block Diagram, Table 14-12, and Table 14-13. Chapter 15: Revised. Chapter 16: Revised TBU Instances. Chapter 17: Table 17-1, Table 17-2 table note added, and added a note after the third item in ECC Poisoning Multi-Purpose Register (DDR4 Only). Chapter 20: Revised. Chapter 21: Revised. Chapter 22: Revised Configure Clocks Chapter 23: Revised Table 23-2, Figure 23-1, FIFOs, and Clocks, Resets. Chapter 24: Revised Clocks and Resets, Table 24-3, Table 24-4, Table 24-21, Table 24-22, Table 24-23, and Table 24-32. Chapter 26: Revised Reference Clock, Interface Controller, DLL Clock Mode, Transmit CMD/DAT Delay, Receive Clock Tap Delay, Table 26-7, Table 26-12, Table 26-15, Table 26-16, and Table 26-23. Chapter 27: Figure 27-2 updated. Chapter 28: Default Logic Levels revised. Chapter 31: Device Programming revised. Chapter 33: Figure 33-1 and Figure 33-18 updated. Chapter 34: Configure the PHY revised. Chapter 37: Revised Figure 37-1, Figure 37-2, Figure 37-4, and Figure 37-5. Chapter 38: Revised Table 38-3 and FPD Reset Sequence.

Date	Version	Revision
8/03/2018	1.8	Chapter 19: Revised Chapter 20: Revised Chapter 22: Revised Chapter 23: Revised Chapter 24: Revised Chapter 25: Revised Chapter 26: Revised Chapter 27: Revised Chapter 28: Revised Chapter 29: Revised Chapter 30: Revised Chapter 31: Revised Chapter 32: Revised Chapter 33: Revised Chapter 34: Revised Chapter 35: Revised Chapter 36: Revised Chapter 37: Revised Chapter 38: Revised
12/22/2017	1.7	Revised Debug Features and added MBIST, LBIST, and Scan Clear (Zeroization) in Chapter 39.

Date	Version	Revision
11/01/2017	1.6	Chapter 1: Revised Figure 1-1. Added Figure 1-2 and Figure 1-3. Updated Introduction to the UltraScale Architecture. Added Functional Units and Peripherals, Device ID Codes, IP Revisions, System Software, and, Documentation. Chapter 2Chapter 2: Revised Figure 2-1. Revised Table 2-1, Table 2-2, Table 2-4, Table 2-5, Table 2-7, Table 2-8, Table 2-10, and Table 2-11. Added Table 2-13 and Table 2-14. Chapter 3: Revised Application Processor Unit Register Overview and Figure 3-8. Chapter 5: Revised Introduction, Graphics Processing Unit Level 2 Cache Controller, and Graphics Processing Unit Programming Model. Chapter 6: Revised Introduction. Updated Table 6-1 and Figure 6-16. Revised Figure 6-1 and Figure 6-2. Chapter 7: Revised Introduction and Functional Description. Updated Figure 7-1, Figure 7-2, and Figure 7-3. Added Interfaces and Signals. Revised Table 7-3. Chapter 9: Revised entire chapter, including changing chapter name. Chapter 10: Revised Figure 10-1. Updated Table 10-1. Updated System Address Register Overview. Added Table 10-9. Chapter 11: Updated Introduction, Boot Image Format, and Functional Units. Revised Table 11-1. Updated Figure 11-1 and Figure 11-3. Added CSU BootROM Error Codes and PL Bitstream. Chapter 12: Updated Device and Data Security and Secure Boot. Revised Figure 12-1, Figure 12-11, and Figure 12-16. Chapter 13: Revised Introduction, System Interrupts, IPI Interrupts and Message Buffers, CPU Private Peripheral Interrupts, and Programming Examples. Updated Figure 13-4 and Figure 13-5. Added Figure 13-6. Chapter 14: Revised APU Core Private Physical and Virtual Timers and System Watchdog Timers. Chapter 15: Updated Block Diagram, AXI Performance Monitor, Quality of Service, and Interconnect Register Overview. Revised Figure 15-1. Added ATB Timeout Description and Programming Example – Metric Counter. Updated Figure 15-3 and Figure 15-4. Chapter 16: Revised Introduction, TrustZone, SMMU Protection on CCI Slave Ports, XMPU Protection of Slaves, and XPPU Protection of Slaves. Added XMPU Register Set Overview, XPPU Register Set Overview, Programming Example, and Write-Protected Registers Table. Revised Figure 16-1, Figure 16-2, Figure 16-4, and Figure 16-6. Added Figure 16-3, Figure 16-5, and Figure 16-7. Chapter 17: Revised Figure 17-1, Figure 17-4, Figure 17-5, and Figure 17-8. Updated Introduction, DDR Subsystem Functional Description, Debugging PS DDR Designs, DDR Memory Controller Register Overview, and DDR Memory Controller Programming Model.

Date	Version	Revision
11/01/2017	1.6 *(cont'd)*	Chapter 18: Revised Figure 18-1 and updated On-chip Memory Programming Model. Chapter 19: Updated Introduction, DMA Controller Functional Description, DMA Data Flow, DMA Programming for Data Transfer, and DMA Programming Model for FCI. Updated Figure 19-1. Chapter 21: Revised Figure 21-1 and Figure 21-2. Added Figure 21-3, Figure 21-4, Figure 21-5, Figure 21-6, Figure 21-7, Figure 21-8, and Figure 21-9. Added MIO – EMIO Signals. Chapter 22: Revised Figure 22-1 and Figure 22-2. Updated I2C Controller Functional Description, I2C Controller Register Overview, and I2C Controller Programming Model. Chapter 23: Revised Figure 23-1. Updated Introduction and Programming Model. Chapter 24: Revised Figure 24-1, Figure 24-2, Figure 24-3, and Figure 24-4. Updated Introduction, System Control, Generic Quad-SPI Controller, Legacy Quad-SPI Controller, Register Overview, Programming and Usage Considerations, Generic Quad-SPI Controller Programming, and Legacy Quad-SPI Controller Programming. Chapter 25: Revised Introduction, Functional Description, and NAND Memory Controller Register Overview. Added Clocks and Resets. Chapter 26: Updated Introduction, Functional Description, and SD/SDIO/eMMC Controller Programming Model. Revised Figure 26-3 and Figure 26-4. Chapter 27: Updated Functional Description and Programming Model. Chapter 28: Updated MIO Table at a Glance and Register Overview. Chapter 29: Renamed chapter. Revised Introduction, Functional Description, Register Overview, and Configuration Program. Updated Figure 29-1, Figure 29-2, and Figure 29-3. Chapter 34: Renamed chapter. Revised Introduction, Functional Description, and Programming Model. Revised Figure 34-1 and Figure 34-2 through Figure 34-8. Chapter 35: Renamed chapter. Revised Introduction, Functional Description, Choosing a Programmable Logic Interface, PS-PL Miscellaneous Signals, Processor Event Signals, and Register Overview. Revised Figure 35-1, Figure 35-2, Figure 35-3, Figure 35-4, Figure 35-5, and Figure 35-6. Chapter 36: Updated Introduction, PL System Monitor, PL System Monitor, PL System Monitor, Video Codec Unit, GTH and GTY Transceivers, and Interlaken. Chapter 37: Renamed chapter. Revised Introduction and Register Overview. Added System PLL Units, Basic Clock Generators, Special Clock Generators, Programming Examples, PLL Integer Divide Helper Data Table, System PLL Control Registers, and Clock Generator Control Registers. Updated Figure 37-1 through Figure 37-5. Added Figure 37-6. Chapter 38: Updated Introduction, Functional Description, and Register Overview. Revised Figure 38-1. Chapter 39: Updated Introduction, JTAG Chain, and CoreSight Functional Description. Revised Figure 39-1, Figure 39-4, Figure 39-6, Figure 39-7, and Figure 39-8.

Date	Version	Revision
03/31/2017	1.5	Chapter 2: Revised the Introduction section including Figure 2-1. Restructured chapter, including revising Table 2-1 through Table 2-12. Chapter 3: Revised Cortex-A53 MPCore Processor Features, Arm v8 Architecture, Power Islands, and Application Processing Unit Reset. Chapter 4: Revised Real-time Processing Unit Features, RPU CPU Configuration, and Lock-step Operation. Chapter 5: Added *Note* to Programming the Mali GPU. Chapter 6: Revised Figure 6-1 and Figure 6-2. Revised Low-Power Operation Mode, including adding Table 6-1. Revised Full-Power Operation Mode. Added *Note* to PMU Processor and before Table 6-14. Added MBIST and Scan Clear Functionality and Interacting with the PMU sections. Revised Table 6-17. Chapter 7: Revised Figure 7-1. Updated RTC Controller Unit, RTC Clock Generation, Specifying the RTC Battery, RTC Controller, and Programming Model. Updated Table 7-3. Chapter 8: Updated chapter title. Revised Functional Safety Overview and Functional Safety Software Test Library. Revised Figure 8-1. Removed Security Features section. Chapter 9: Revised Introduction and Features. Revised Table 9-1. Added Table 9-2. Updated Figure 9-1. Chapter 10: Revised Figure 10-1. Updated Table 10-1, Table 10-2, and Table 10-4 through Table 10-8. Chapter 11: Updated Introduction, Boot Flow, Boot Modes, Golden Image Search, Loopback Mode, and Initialize PCAP Interface. Revised Table 11-1, Table 11-2, Table 11-3, Table 11-4, Table 11-8, and Table 11-9. Updated Figure 11-1 and Figure 11-2. Chapter 12: Updated chapter title. Updated Introduction, Secure Processor Block, Crypto Interface Block, Tamper Monitoring and Response, Key Management Summary, and Secure Boot. Added Device and Data Security and Protecting Test Interfaces. Revised Figure 12-1, Figure 12-2, Figure 12-9, Figure 12-10, Figure 12-11, and Figure 12-12. Added Table 12-2, Table 12-3, Table 12-4, Table 12-5, Table 12-6, Table 12-7, Table 12-8, Table 12-13, Table 12-14, and Table 12-15. Added Figure 12-4, Figure 12-5, Figure 12-6, Figure 12-7, Figure 12-14, Figure 12-15, Figure 12-16, Figure 12-17, Figure 12-18, Figure 12-19, and Figure 12-20. Chapter 13: Revised Introduction. Added System Interrupts, including Table 13-1. Added GIC Interrupt System Architecture, RPU GIC Interrupt Structure, APU GIC Interrupt Controller, IPI Interrupts and Message Buffers, GIC Proxy Interrupts, and CPU Private Peripheral Interrupts. Revised Figure 13-1, Figure 13-2, Figure 13-4, and Figure 13-5. Revised Table 13-3, Table 13-5, and Table 13-6.

Date	Version	Revision
03/31/2017	1.5 *(cont'd)*	Chapter 14: Reorganized and revised entire chapter, including the changes listed here. Updated chapter title. Updated Introduction, including revising Figure 14-1. Added APU MPCore System Counter and APU Core Private Physical and Virtual Timers. Revised Triple-timer Counters and System Watchdog Timers. Revised Figure 14-2 and Figure 14-3. Revised Table 14-1 through Table 14-4. Revised Table 14-11 through Table 14-12. Chapter 15: Revised Full Power Domain, including adding Figure 15-2 and Figure 15-3. Revised, including adding Low Power Domain in Figure 15-4 and Figure 15-5. Chapter 16: Revised Figure 16-1. Revised Table 1. Revised AXI and APB Isolation Block. Chapter 17: Revised Figure 17-1, Figure 17-2, Figure 17-4, Figure 17-18, and Figure 17-21. Revised DDR Memory Controller Features, DDR Subsystem Functional Description, Debugging PS DDR Designs, and DDR Memory Controller Programming Model. Revised Table 17-1, Table 17-2, Table 17-3, Table 17-9 through Table 17-37. Chapter 18: Revised Introduction, On-chip Memory Functional Description, On-chip Memory Register Overview, and On-chip Memory Programming Model. Revised Figure 18-1 and Figure 18-2. Updated Table 18-1 and Table 18-2. Chapter 19: Revised Introduction, DMA Controller Functional Description, DMA Data Flow, DMA Interrupt Accounting, DMA Over Fetch, and DMA Programming Model for FCI. Revised Table 19-5. Chapter 20: Revised Functional Description, Clocks, Resets, Controller Modes, Message Format, Message Buffering, Interrupts, RX Message Filtering, and CAN0-to-CAN1 Connection. Updated Figure 20-1 and Figure 20-3. Revised Table 20-1, Table 20-2, Table 20-3, Table 20-4, and Table 20-5. Chapter 22: Revised Figure 22-1. Updated Glitch Filter, revised Table 22-2 through Table 22-29. Chapter 23: Revised Table 23-1, Table 23-2, and Table 23-4. Chapter 24: Revised Controller Features and Quad-SPI Feedback Clock . Added Linear Addressing Mode (Memory Reads). Chapter 25: Revised Features and AXI Interface. Chapter 26: Updated System/Host Interfaces, Non-DLL Mode Clocking, DLL Mode, and SD Tap Delay Settings. Revised Table 26-1, Table 26-2, and Table 26-3 through Table 26-8. Revised Figure 26-2. Chapter 27: Updated Features, SDK and Hardware Design, Functional Description, MIO Pin Configuration, GPIO Channel Architecture, Device Pin Channels, MIO Signals, EMIO Signals, Interrupt Function, System Interfaces, Register Overview, and Programming Model. Revised Table 27-1, through Table 27-11. Updated Figure 27-3 and Figure 27-4. Chapter 28: Updated Introduction and Multiplexed I/O Programming Model - Example. Revised Figure 28-1. Revised Table 28-1 and Table 28-2. Chapter 29: Revised Figure 29-5, Data Selection Multiplexer, Predriver, and Voltage Mode Driver, and Table 29-5. Removed Transmitter Boundary Scan, Boundary Scan Receiver, and SIOU Registers sections. Chapter 30: Added *Important Note* in PCIe Domain Interrupts. Revised *Important Note* in Card to System Flow (EP Memory to Host Memory).

Date	Version	Revision
3/31/2017	1.5 (*cont'd*)	Chapter 31: Added *Note* to USB 2.0/3.0 Host, Device, and USB 2.0 OTG Controller Features. Chapter 32: Updated Features, SATA Host Controller Interface Description, AXI Master Port Security Features, and AXI Slave Port Security Features. Revised Figure 32-1 and Figure 32-5. Updated Table 32-4. Chapter 33: Revised Introduction, Features, DisplayPort DMA, and Register Overview. Updated Figure 33-18 and Figure 33-22. Revised Table 33-13, Table 33-14, and Table 33-15. Chapter 34: Revised IEEE Std 1588 Time Stamp Unit, Configure the Controller, Status and Wakeup Interrupts, Transmitting Frames, Receiving Frames, and Gigabit Ethernet Debug Guide. Added note to Gigabit Ethernet Controller using EMIO. Revised *Recommended* note in Configure the Buffer Descriptors. Chapter 35: Revised Programmable Logic Introduction, PS-PL Interface Features. Updated *Note in AXI Interface Programming*. Updated Table 35-8. Revised Figure 35-1. Chapter 36: Added High-Speed Transceivers (GTH Quad and GTY Quad) and DisplayPort Video and Audio Interfaces. Chapter 37: Revised Figure 37-1 and Figure 37-2. Updated Clocking Functional Description, LSBUS Clock, and TOPSW Main Clock. Chapter 39: Updated Features, JTAG and DAP Functional Description, and JTAG Chain Configuration. Added *Note* to JTAG Error Status Register.
2/02/2017	1.4	Chapter 11: Added an *Important* note on page 195. Chapter 20: Removed Step 1F on page a452, and Step 1F and Step 2F on page 471. Chapter 30: Updated the *Important* note on page 773. Chapter 34: Added a Tip on page 976. Chapter 35: Updated the *Note* on page 1001. Updated Table 35-8.

Date	Version	Revision
10/25/2016	1.3	Added the dual-core Arm® Cortex™-A53 MPCore™ revisions throughout. Added Chapter 7 and Chapter 8. Chapter 1: Updated Figure 1-1, Table 1-1, and the Register Reference with clarifications. Chapter 2: Revised the Introduction section including Figure 2-1. Updated the AXI Interfaces section and Table 2-5. Revised the Processor Communications section. Revised the section including adding examples. Added the PS-PL Miscellaneous Signals section. Added comments to Table 2-12. Chapter 3: Updated the I/O Coherency section. Updated and moved the ACE Master Interface section. Updated the Individual MPCore Shutdown Mode section. Revised Figure 3-8. Chapter 4: Added the Interrupt Injection Mechanism section. Update Table 4-5. Chapter 6: Updated the chapter including removing Figure 6-1: Power Modes. Revised Figure 6-1 and Figure 6-2. Updated the descriptions in the PMU RAM, PMU GPIs and GPOs, and PMU Programmable Interval Timers sections and removed sections of Table 6-14. Clarification of some descriptions in the Platform Management Unit Operation section. Updated the Reset Services section. Updated and added the register type to Table 6-17. Chapter 9: Revised the PS and PL System Monitor Programming Model section including replacing PSSYSMON with AMS_PS_SYSMON and PLSYSMON with AMS_PL_SYSMON. Chapter 10: Edited the Global Address Map discussion including Figure 10-1. Updated the start addresses for R5_0_ATCM_LSTEP and R5_0_BTCM_LSTEP in Table 10-2. Added the PL AXI Interface section. Updated the S_AXI_HPCx_FPD address descriptions in Table 10-8. Chapter 11: Updated Figure 11-1. Added the Tip on page 197. Updated the offset `0x038` description in Table 11-4. Chapter 12: Updated the Key Management section including Figure 12-3. Updated Table 12-11, Figure 12-5, Figure 12-7, Figure 12-10, and Figure 12-11. Added the Battery-Backed RAM, Programming the eFUSE, and Reading the eFUSE sections.

Date	Version	Revision
10/25/2016	1.3 (*cont'd*)	Chapter 13: Updated the Interrupts Functional Description section including Figure 13-1. Revised the Shared Peripheral Interrupts section. Updated Figure 13-4 and added a note on page 275. Added Table 13-3 and Table 13-4. Updated Figure 13-5. Updated the start and end ID numbers in Table 13-5. Updated and removed registers in the IPI section of Table 13-5 and added a note on page 265. Added the Clearing Pending Interrupts from the APU GICv2 section. Chapter 14: Updated the Introduction. Revised the Physical Counter, System Watchdog Timers, and Triple-timer Counters sections. Chapter 15: Revised the Introduction, Interconnect Functional Description, and Quality of Service sections. Revised Figure 15-1 and Figure 15-6. Removed the *Interconnect Submodules* and *Interconnect Programming Models* sections and added them to Chapter 16. Chapter 16: Updated the Introduction including adding use cases and a terminology section. Revised Figure 16-1. Numerous updates to the System Protection Unit Functional Description section including further information on poisoning a request. Added the AXI Timeout Block and AXI and APB Isolation Block sections from Chapter 15. In the XMPU Programming section replaced XMPUx with DDR_XMPUx, FPD_XMPU with FPD_XMPU_CFG, and OCM_XMPU with OCM_XMPU_CFG. Added the AXI/APB Isolation Block Programming section from Chapter 15. Chapter 17: Revised the DDR Memory Controller Features. Revised Figure 17-1. Revised Table 17-1 and Table 17-2. Updated Figure 17-5. Added the SDRAM Address Mapping section. Removed the DDRC traffic class and transaction discussions. Updated the Address Collision Handling section. Revised the Restriction on Data Mask when ECC is Enabled, PHY Utility Block, and Data Training sections and updated Figure 17-8. Replaced Figure 17-14 and added Figure 17-15, Figure 17-16, Figure 17-17. Chapter 18: Updated the Introduction section and added a features list. Added Figure 18-1 and the 64-bit ECC Support and Low Power Operation sections. Chapter 19: Updated the Features section. Chapter 20: Clarified Introduction and updated Figure 20-1. Updated the Interrupts section. Chapter 21: Updated the Features section. Chapter 22: Updated the Introduction section. Updated Figure 22-1. Added I2C Master Mode, I2C Slave Mode, and MIO-EMIO Signals sections. Removed individual register descriptions. Chapter 24: Updated the Introduction. Updated Figure 24-1 and Figure 24-2. Revised Table 24-3. Removed the *Read/Write Request Details* section. Revised Table 24-10, Table 24-11, and Table 24-12. Updated Figure 24-3 and Figure 24-4. Added the Controller Features, System-level View, Address Map and Device Matching For Linear Address Mode, and Legacy Quad-SPI Operating Restrictions sections. Updated the Using the Quad-SPI Controller section. Added the Dynamic Mode and Baud Rate Change Limitations and the Reference Clock Change Limitations sections. Updated the Quad-SPI Controller Programming and Usage Considerations section. Chapter 25: Removed the NAND Flash Device Sequence section. Added Change Timing Mode and NVDDR-SDR and ONFI Set Feature tables.

Date	Version	Revision
10/25/2016	1.3 (*cont'd*)	Chapter 26: Added the SD Host Controller Operation section. Updated the bit field descriptions around Figure 26-2. Added the SD Tap Delay Settings section. Revised Table 26-9 and Table 26-10. Added Table 26-11. Chapter 27: Moved the PS-PL MIO-EMIO Signals and Interfaces section from Chapter 28. Updated Figure 27-1 and replaced Figure 27-3. Updated the EMIO Signals section. Updated the Clock and Reset sections. Removed the Interrupts section and the MIO Programming section. Revised the register names in Table 27-3. Chapter 28: Revised the Multiplexed I/O Functional Description sections including the Output Multiplexer descriptions. Moved the Output MultiplexerPS-PL MIO-EMIO Signals and Interfaces section to Chapter 27. Added the Drive Strength section. Updated Note 1 in Table 28-1 and Table 28-2. Updated the register fields in Table 28-4 and Table 28-5. Chapter 29: Updated Figure 29-1 and the PS-GTR Transceiver Interface Features section. Updated Figure 29-2. Updated the Interconnect Matrix section to focus on using the PCW. Updated Figure 29-3 and Figure 29-4. Removed Figure 29-5 and the *PLL Clock and Reset Distribution* section. Added the Spread-Spectrum Clocking Transmitter Support section. Updated Figure 29-6 Added the Spread-Spectrum Clocking Receiver section. Removed the *PS-GTR Eye Scan* section including Figure 29-10, Figure 29-11, and Table 29-5. Added the TX Configurable Driver section. Updated the PS-GTR Transceiver Register Overview and PS-GTR Transceiver Programming Considerations sections. Chapter 30: Updated Figure 30-1, Figure 30-2, Figure 30-3, Figure 30-4, Table 30-1, and Table 30-2. Added an *Important* note on page 752 and another note on page 753. Added the PCIe and AXI Domain Interrupts section. Added a note on page 761. Added the Programmed I/O Transfers section including Figure 30-10 and Figure 30-11. Chapter 31: Added base addresses of the USB controllers on page 815. Chapter 32: Updated the SATA Host Controller Interface Description. Added an Important note to page 835. Updated the Link Layer section and added Figure 32-3 and Figure 32-4. Added the SATA Clocking and Reset section. Added register addresses to Table 32-1 through Table 32-4. Added PM clock frequency selection rows to Table 32-7. Chapter 33: Revised entire sections including the Introduction. After removing the DisplayPort *Controller Blocks* section, moved Figure 33-1 to the DisplayPort Controller System Viewpoint section. Added and revised a significant portion of the DisplayPort Controller Functional Description and DisplayPort Controller Programming Considerations sections. Updated the graphics in the Supported Video Formats section. Chapter 34: Updated the GEM Features list. Added an Important note on page 942 and a Tip on page 943. Added the Clock Control Register section. Chapter 35: Updated Figure 35-1. Updated the Recommended note on page 998. Updated the ACP Limitations section. Added the CG devices and revised the bitstream length values for the ZU7EV in Table 35-8. Added an Important note on page 1026. Updated the register names in Table 35-9. Chapter 37: Updated Figure 37-2 and the paragraph that follows on page 1040. Added the LSBUS Clock section. Clarifying edits to the Full-Power Domain section. Chapter 38: Updated the Reset System Functional Description section and added Figure 38-1. In Table 38-1, added PS_POR_B and updated SRST_B. Added Table 38-4.

Date	Version	Revision
10/25/2016	1.3 *(cont'd)*	Chapter 39: Updated the System Test and Debug Features section. Moved the following sections: System Test and Debug and JTAG and DAP Functional Description. Updated Table 39-1 and added Note 1. Updated Table 39-2. Revised Figure 39-2. Updated Table 39-7.
6/01/2016	1.2	Chapter 1: Added the Register Reference section. Chapter 2: Combined Table 2-2 and Table 2-4. Chapter 3: Added further descriptions on page 39. Added Figure 3-3 and Figure 3-4. Added an important note to page 53. Chapter 4: Updated the Normal (Split) Operation description. Added the Lock-step Sequence in Cortex-R5 Processors section. Chapter 5: Added a recommendation on page 94. Chapter 6: Added the Power Modes section. Updated the PMU System-level View section. Updated the descriptions in Table 6-3. Revised descriptions in the PMU Clocking section. Removed PMU local registers from Table 6-6. Updated descriptions in Table 6-13. Replaced Table 6-14 and Table 6-17. Updated register names in the Power Down and Power Up sections. Updated the Isolation Request description. Chapter 9: Updated the Features descriptions. Revised the steps in the PS and PL System Monitor Programming Model and added the PL SYSMON programming steps. Chapter 11: Revised SD1/MMC33 and SD1-LS in Table 11-1. Revised the SD0/SD1/MMC section and added SD1-LS support on page 197. Added Table 11-3. Clarifying edits to Figure 11-2 and the Initialize PCAP Interface section. Chapter 12: Clarified the secure processor reset in Figure 12-7. Chapter 13: Updated XPPU in Table 13-5. Chapter 14: Revised the Physical Counter description. Added the CSU_WDT to the System Watchdog Timers discussion. Chapter 15: Updated the device names in Table 16-8. Chapter 16: Added further address definition on page 347. Added Note 1 to Table 16-6. Added further information after Table 16-7. Chapter 17: Added the DDR Memory Types, Densities, and Data Widths section and updated the Traffic Classes section. Chapter 19: Added Figure 19-5. Chapter 24: Revised the descriptions and content in Table 24-8, Table 24-9, and Table 24-11. Chapter 25: Added a recommendation on page 627.. Chapter 26: Clarified eMMC throughout chapter. Updated eMMC Card Interface section. Updated Figure 26-4. Added Note 1 to Table 26-13. Added an important note to page 686. Updated Figure 26-6. Chapter 28: Updated Table 28-2 (QSPI pins 1-4, 8-11) and added Note 1 to SD0/1. Chapter 29: Updated Figure 29-5. Chapter 30: Updated the Configuration Control (APB Interface) section. Revised the Power Management section discussion on ASPM. Added important notes in the Accessing Bridge Internal Registers section. Added information on Endpoint Compliance. Added a Tip on page 762. Added an important note on page 773.

Date	Version	Revision
6/01/2016	1.2 (*cont'd*)	Chapter 33: Added Figure 33-1 to the DisplayPort *Controller Blocks* section. Added the Supported Video Formats section. Chapter 34: Added clock restrictions generating a reference clock for GEM on page 938. Added a note on page 946. Chapter 35: Added SACEFPD_ACLK to Table 35-3. Chapter 37: Updated the Tip on page 1037 and updated an important note and added a new Tip on page 1038. Updated Figure 37-1, Figure 37-2, and Figure 37-3. Added Table 37-1 and Table 37-2. Updated the Programmable Clock Throttle section. Chapter 39: Updated the JTAG and DAP Functional Description section. Updated the Third-Party Tool Support.
3/07/2016	1.1	Chapter 1: Updated the System Block Diagram section and the LLPP in Figure 1-1. Chapter 2: Updated Figure 2-1, Table 2-1, Table 2-2, and Table 2-3. Removed Table 2-7: *Debug Pins and Associated Signals*. Updated Table 2-5. Revised the Interrupts discussion. Revised Table 2-12. Chapter 4: Updated the Normal (Split) Operation and TCM Access from a Global Address Space sections. Chapter 6: Updated Figure 6-2. Revised the Platform Management Unit Functional Description section. Clarified descriptions in the PMU GPIs and GPOs section. Removed Table 6-12: *PMU Dedicated I/O*. Removed the *Use Case for System-level Reset* section. Chapter 9: Updated the Introduction. Updated Figure 9-1 and Table 9-3. Removed Table 7-4: *PS SYSMON Block Auxiliary Channel Registers*. Updated Table 9-9. Chapter 10: Updated the PMU_ROM in Table 10-2. Chapter 11: Revised Figure 11-2, Table 11-1, Table 11-4, and Table 11-5. Expanded the Boot and Configuration Functional Description section. Added a recommendation to the Boot Modes section. Updated the Initialize PCAP Interface section. Chapter 12: Added the Tamper Monitoring and Response section. Moved the Secure Stream Switch section to Chapter 11. Updated the Family Key description. Updated the Key Management, RSA Accelerator, and RSA Operations sections. Added the CSU BootROM Error Codes section. Replaced Figure 12-11. Added the Programming SHA-3 Engine section. Chapter 13: Revised the RPU GIC Interrupt Structure section. Chapter 15: Revised Figure 15-1. Removed Figure 13-3: *Monitor Points in the LPD* and updated the APM Points section. Added Quality of Service section. Chapter 16: Revised Figure 16-1. Added notes on page 349 and page 351. Updated the section. Added the XMPU SINK Register Summary. Chapter 17: Revised Figure 17-1. Removed the *BIST Loopback Mode* section. Updated the Data Training section. Updated Figure 17-20 and Figure 17-21. Chapter 18: Updated the Introduction. Revised Table 18-2 and Figure 18-2. Removed the *Adjust Extra Margin Access Register* section.

Date	Version	Revision
3/07/2016	1.1 *(cont'd)*	Chapter 19: Updated the Simple DMA Mode section and Step 4 under Simple Mode Programming. Chapter 24: Added the Quad-SPI Tap Delay Circuits section. Chapter 26: Updated Table 26-9 and Table 26-10 and the Card Detection section. Chapter 27: Removed the GPIO *Bypass Mode* section and Figure 27-4. Chapter 28: Updated the Boot from SD Card section and added the eMMC Mapping section. Chapter 29: Updated Figure 29-2. Updated the Reference Clock Network section including Figure 29-5. Chapter 30: Updated the Controller for PCI Express Features list. Updated Figure 30-5. Updated Table 30-10. Chapter 31: Revised the USB 2.0/3.0 Host, Device, and USB 2.0 OTG Controller Features section. Chapter 32: Updated the SATA Host Controller Interface Description and TrustZone Support sections. Chapter 33: Added Figure 33-5. Chapter 34: Updated the Gigabit Ethernet Controller Introduction and Clock Domains sections. Added the External FIFO Interface section. Updated bit 24 in Table 34-5. Minor revisions in the IEEE Std 802.3 Pause Frame Reception and PFC Pause Frame Reception sections. Chapter 35: Revised Figure 35-1, Figure 35-4, Figure 35-5, Figure 35-6. Added important notes on page 1007 and page 1008. Chapter 36: Updated Figure 36-1 and Figure 36-2. Chapter 37: Updated the System Viewpoint, APU Clock, and DDR Clock sections. Added the PLL Operation section. Added the Low-Power Domain section to Table 37-4. Updated the examples in Clocking Programming Considerations. Added the PLL Integer Divide Programming section. Chapter 38: Updated the Functional Description section and the Programming Model. Chapter 39: Added features and a flowchart to the Fabric Trigger Macrocell section. Moved the PJTAG Signals, JTAG Toggle Detect, JTAG Disable, JTAG Error Status Register, and JTAG Boot State. Added Figure 39-6.
11/24/2015	1.0	Initial Xilinx release.

Please Read: Important Legal Notices