# Versal ACAP CPM CCIX

## *Architecture Manual*

**AM016 (v1.0) July 16, 2020**

# Revision History

The following table shows the revision history for this document.

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

# Table of Contents

*Chapter 1*

# Overview

## Introduction to Versal ACAP

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

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

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

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

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

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

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

## Navigating Content by Design Process

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

- **System and Solution Planning:** Identifying the components, performance, I/O, and data transfer requirements at a system level. Includes application mapping for the solution to PS, PL, and AI Engine. Topics in this document that apply to this design process include:
  - Chapter 2: CCIX Capable PCIe Controller
  - Chapter 3: CCIX Architecture
  - Chapter 4: Clock, Reset, and Debug
  - Appendix A: Using CIPS GUI to Configure CPM-CCIX
- **Host Software Development:** Developing the application code, accelerator development, including library, XRT, and Graph API use. Topics in this document that apply to this design process include:
  - Chapter 2: CCIX Capable PCIe Controller
  - Chapter 3: CCIX Architecture
- **Hardware, IP, and Platform Development:** Creating the PL IP blocks for the hardware platform, creating PL kernels, subsystem functional simulation, and evaluating the Vivado® timing, resource use, and power closure. Also involves developing the hardware platform for system integration. Topics in this document that apply to this design process include:
  - Chapter 4: Clock, Reset, and Debug
  - Appendix A: Using CIPS GUI to Configure CPM-CCIX

# List of Acronyms

*Table 1:* **List of Abbreviations**

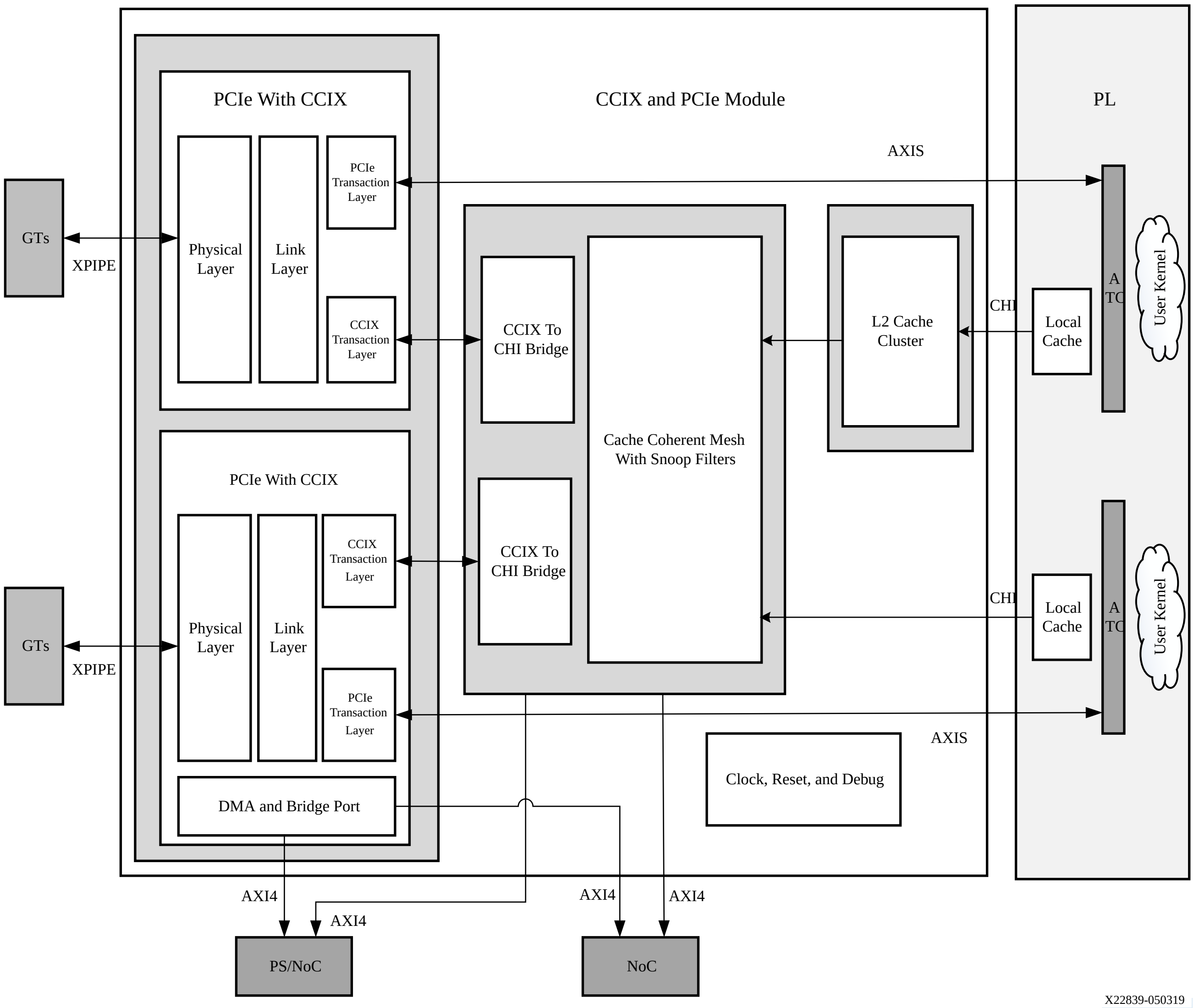
| Acronym | Definition |
|---|---|
| CPM | Integrated block for PCIe with DMA and cache coherent interconnect |
| CCIX | Cache Coherent Interconnect for Accelerator |
| CML | Coherent Multi-chip Link |
| CXS | CCIX Streaming Interface |
| ELA | Extended Logic Analyzer |
| ESM | CCIX Extended Speed Mode of 20 GT/s and 25 GT/s; also referred to as EDR in this document |
| CIPS | Control, Interfaces, and Processing System IP core. |
| PCSR | Programming Control and Status Register |
| CRC | Clock Reset CPM |
| CPI | CPM PL Interface |
| DVSEC | Designated Vendor Specific Extended Configuration Space |
| EDR | Extended Data Rate |
| PoC | Point of Coherency |
| NoC | Network on Chip |
| CMN | Coherent Mesh Network |
| RAS | Reliability, Availability, and Serviceability |

# Introduction to CPM

This architecture manual provides a detailed description of the Versal™ ACAP Integrated block for PCI Express® with DMA and cache coherent interconnect (CPM) operating in CCIX mode. The integrated PCI Express controllers operate as endpoints when used in CCIX mode. This chapter introduces the CPM. For details on PCI Express functionality as root ports or with integrated DMA blocks, see the *Versal ACAP CPM Mode for PCI Express Product Guide* (PG346).

The CPM comprises two instances of the integrated controller for PCI Express . Both instances support CCIX capability. The CPM also contains the necessary components to allow a logic accelerator to act as a CCIX-compliant accelerator. The following figure shows the block diagram for the CPM.

*Figure 1:* **CPM Block Diagram**




The CPM connects to the transceivers, programmable logic (PL), and processing system (PS). The PS provides paths to connect to the network on chip (NoC) and to the DDR.

The main components of the CPM are:

- Two integrated PCI Express controllers that support Gen4 with up to x8 independent lanes. CCIX-only 20 GT/s and 25 GT/s extended data rate (EDR) for x8 link width are also supported. Use of only one x16 Gen4 controller is also supported.
  - Virtual channel 0 (VC0) carries standard PCI Express traffic
  - VC1, when enabled, carries CCIX traffic

- A cache coherence block that includes the cache coherent interconnect and additional logic needed to connect to the CCIX port of each integrated PCI Express controller.
  - The cache coherent interconnect is realized using Coherent Mesh Network IP. For more information on the CMN-600 Coherent Mesh Network, see the Arm® CoreLink CMN-600 Coherent Mesh Network Technical Reference Manual.
  - One instance of the L2 cache controller interfaces with programmable logic through the CHI-B-compliant Coherent Hub Interface.

# About CCIX

The Cache Coherent Interconnect for Accelerators (CCIX) is a chip-to-chip interconnect that enables two or more devices to share data in a cache-coherent manner. Machine learning and big data applications are fundamentally changing the way that data is processed. Classic processor data flows are now being augmented with off-chip accelerators that can be customized for specific types of applications, from compute accelerators to network traffic acceleration. This has driven an industry-wide movement towards accelerators and heterogeneous compute.

For many of today's compute tasks, accelerators can complete the needed functionality both faster and with lower power consumption than the processor working on its own. However, unmanaged heterogeneity can bring software complexity. CCIX is poised to optimize and simplify how heterogeneous systems are designed while at the same time increasing bandwidth and reducing latency in the systems built with devices processing via processors with different instruction set architectures (ISAs), or application-specific accelerators. See the CCIX Consortium web page for more information.

# Features

The CPM supports these CCIX features:

- CCIX specification v1.0 compliance
- Extended speed modes: 20 GT/s and 25 GT/s
- Bifurcation to operate as single link of x16 or two links of x8 controllers
- Integrated L2 cache of 1 MB size
- Request agent functionality
- Home agent functionality
- Slave agent functionality
- 64-byte cache-line size support

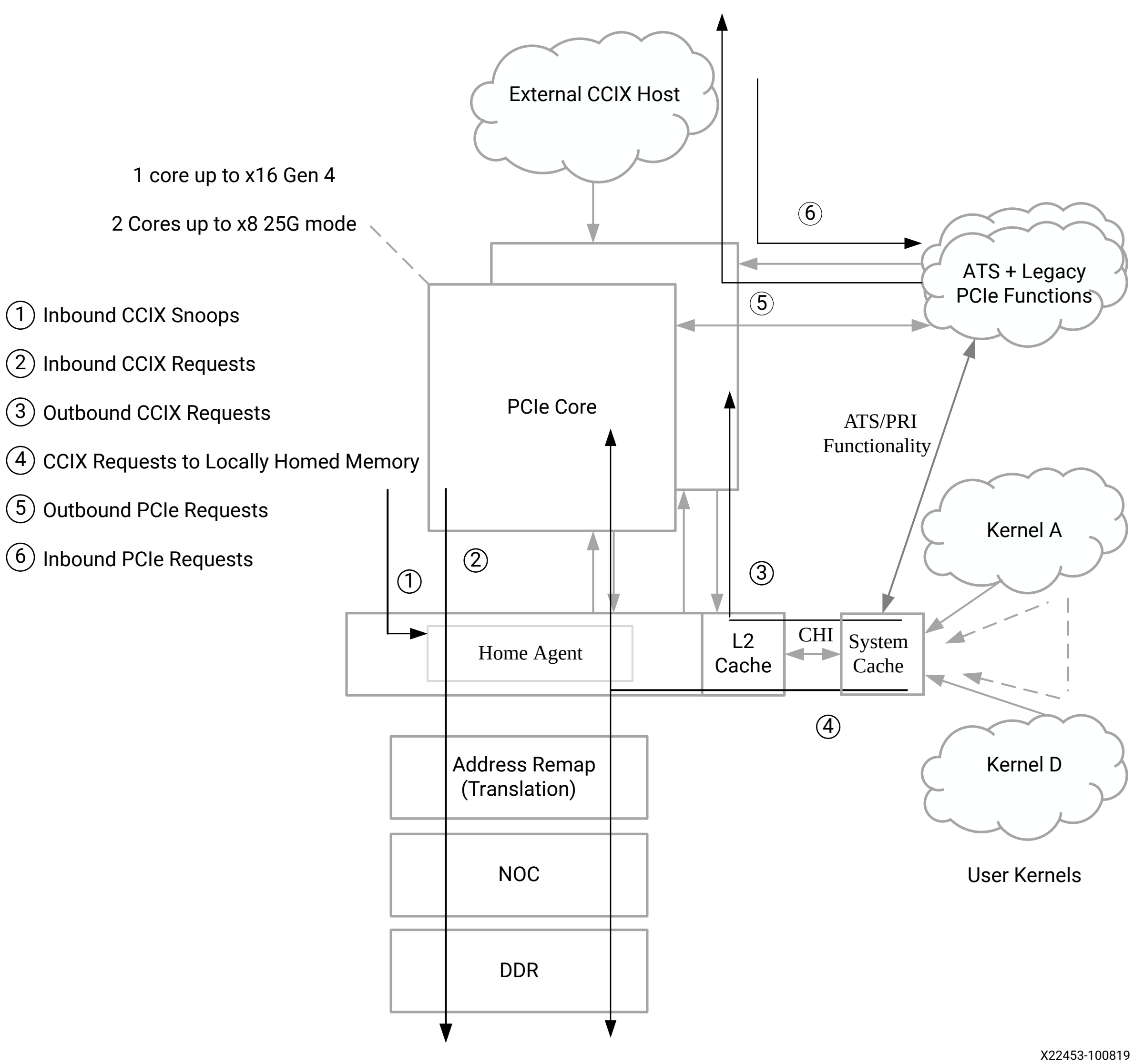
- Support for CCIX features such as port aggregation, message packing, message chaining, and protocol error reporting (PER)
- Concurrent CCIX/PCIe access to Versal™ ACAP devices as endpoints

**Note**: All references to the Coherent Hub Interface (CHI) in this document are CHI-B compliant.

# Use Case Modes

This section provides an overview of CCIX modes of operation along with transaction flows. The following figure shows the transaction flow overview for the three CCIX agents in the CPM.

*Figure 2:* **CCIX Agent Transaction Flow Overview**



***Note*:** The numbering in the figure refers to transaction flow in the following section. The numbering does not imply any ordering requirement.

The CCIX utilizes the traditional VC0 path for programming the user kernels (control path operations) and address translation services (ATS) to enable shared virtual memory paradigms where user kernels work directly with virtual addresses. This is depicted by the outbound PCIe requests (for example, ATS requests) and inbound PCIe requests (for example, register programming in user kernel) paths. Appendix A: Using CIPS GUI to Configure CPM-CCIX provides an overview of Vivado Control, Interfaces, and Processing System (CIPS) configuration options for various agents and modes in the CPM.

CCIX agent types supported are:

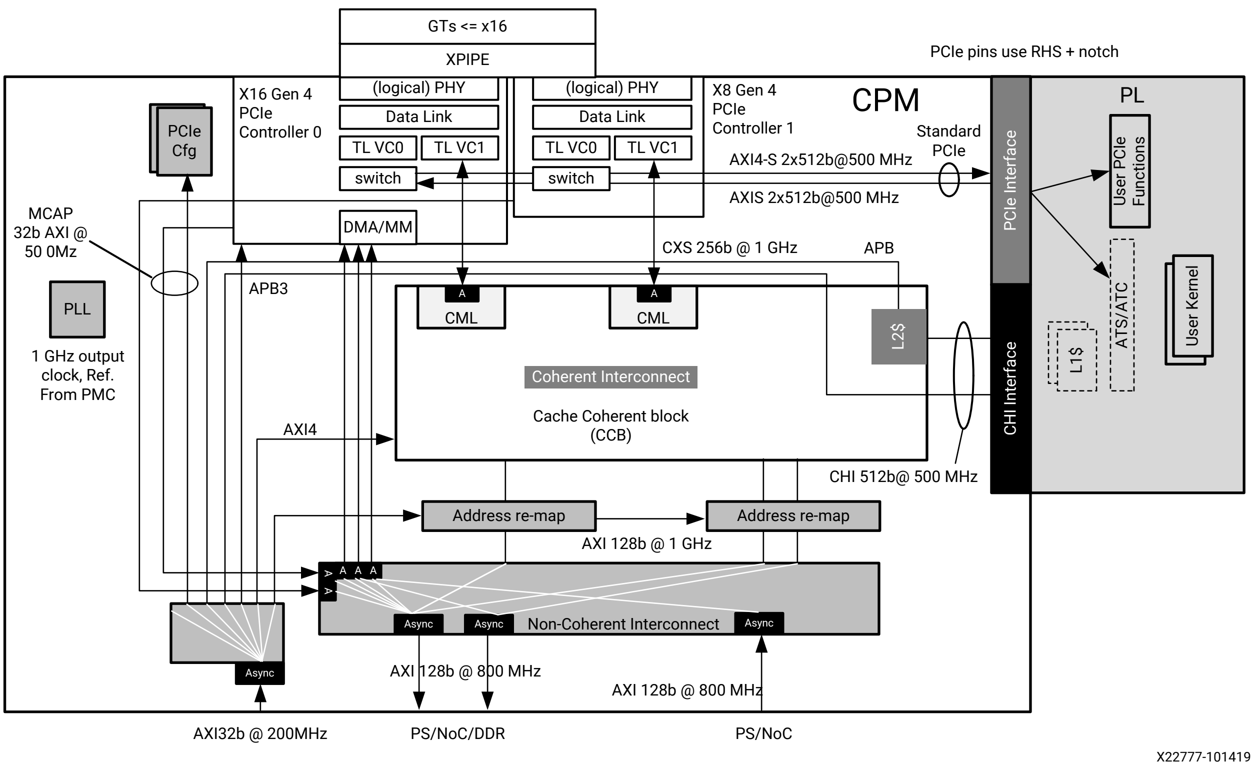
- **Request Agent:** A request agent (RA) is a CCIX agent that is the source of the read and write transactions. Each of the CCIX RAs might have one or more internal initiators or acceleration functions (AF). With regards to transaction flow, user kernels (implemented in programmable logic) issue read or write requests (outbound CCIX requests) and remote home agent issues snoops (shown as inbound snoops).
- **Home Agent:** A home agent (HA) is a CCIX agent that manages coherency and access to memory for a given address range. An HA manages coherency by sending snoop transactions to the required RA when a cache state change is required for a cache line. Each CCIX home agent acts as a Point of Coherency (PoC) and Point of Serialization (PoS) for a given address. With regards to transaction flow, remote-RA (on the host end) issues read or write requests (inbound CCIX requests) and if local RA (with kernel in programmable logic) is also enabled, requests to local memory (CCIX requests to locally homed memory) are seen.
- **Slave Agent:** CCIX enables expanding system memory to include memory attached to an external CCIX device. When a home agent resides on a separate chip, the resulting new architectural component (expansion memory) is referred to as a slave agent (SA). An SA is never accessed directly by an RA. An RA always accesses an HA, which in turn accesses the SA. In terms of transaction flow, this always receives memory requests from remote-HA (inbound CCIX requests).

As there are two integrated controllers for PCI Express, each agent can be supported in a single-port or dual-port configuration. A higher bandwidth connectivity between two CCIX devices can be achieved by aggregating multiple CCIX ports; this is referred to as port aggregation. Port aggregation is supported in dual-port configuration.

# CCIX Capable PCIe Controller

The Xilinx® Versal™ ACAP CPM PCIe® block is a high-bandwidth, scalable, and reliable serial interconnect building block solution for use with Versal ACAP devices. Xilinx offers two PCIe controllers in the Versal CPM: the PCIe 0 controller, and the PCIe 1 controller. Both controllers are CCIX capable. The CPM PCIe block diagram is shown in the following figure.

*Figure 3:* **CPM PCIe Block Diagram**




The controllers in CCIX mode support EDR configurations at x4 and x8 widths and at speeds of 20 GT/s and 25 GT/s, as well as all other lower performance configurations: Gen1 (2.5 GT/s), Gen2 (5.0 GT/s), and Gen3 (8 GT/s). Only one controller supports Gen4 (16 GT/s). Both controllers are compliant to the CCIX Transport Specification 1.0.

# Features

The CCIX PCIe® controller provides these features:

- CCIX Transport Specification 1.0 compliant
- Transport DVSEC
- Protocol DVSEC
- CCIX Transaction Layer implemented in Virtual Channel 1 (VC1)
- Extended Data Rate (EDR) support
    - EDR16 (2.5 GT/s, 5 GT/s, 8 GT/s, and 16 GT/s)
    - EDR25 (20 GT/s and 25 GT/s)
- Up to Gen3 speed in x16, up to EDR25 in x4 and x8 link configuration
- 256-bit CCIX interface
- PCIe Compliant Mode and Direct Attach Mode
- Multi-function, SR-IOV, and Multi-controller support

# Interfaces

CCIX architecture requires implementations to use Virtual Channel 0 (VC0) as the control channel. For more information regarding AXI4-ST data interfaces on VC0 and control and status interfaces, refer to the *Versal ACAP CPM Mode for PCI Express Product Guide* (PG346). All CCIX packets received via PCI Express Virtual Channel 1 (VC1) are routed internally to the CPM and to the CMN. All VC0 data and control interfaces from PCIe controllers are synchronous to the user_clk and are reset with the user_reset signal.

# CCIX VC1 Configuration Interface

The CCIX VC1 configuration is described in the following table.

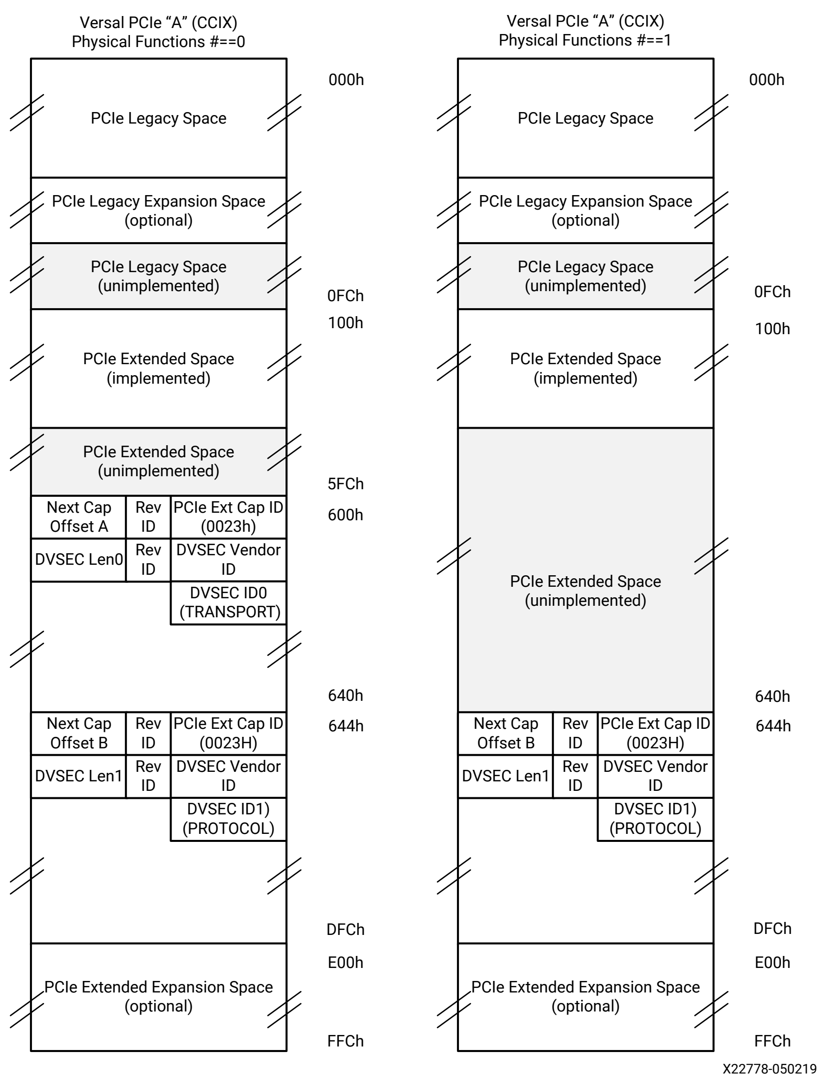
*Table 2:* **CCIX VC1 Configuration Interface**

| Name | Direction | Width | Description |
|---|---|---|---|
| cfg_vc1_enable | Output | 1 | • Configuration VC1 Enable: VC1 Resource<br>• Register: VC<br>• Enable bit:<br>◦ 1: Software has enabled VC1 operation<br>◦ 0: VC1 disabled |
| cfg_vc1_negotiation_pending | Output | 1 | • Configuration VC1 Negotiation Pending: VC1 Resource Status<br>• Register: VC Negotiation Pending bit<br>◦ Asserted when active High<br>◦ VC1 negotiation (initialization or disabling) is in the pending state |
| cfg_ccix_edr_data_rate_change_req | Output | 1 | This signal is asserted (active-High) before entry into the EDR 20 GT/s or 25 GT/s data rate. When asserted, the user design must drain all pending TLPs on both VC0 and VC1 data paths. The signal is deasserted in response to the assertion of cfg_ccix_edr_data_rate_change_ack. This signal must be kept asserted until cfg_ccix_edr_data_rate_change_req is deasserted. |
| cfg_ccix_edr_data_rate_change_ack | Input | 1 | This signal is asserted (active-High) by the user design in response to the assertion of cfg_ccix_edr_data_rate_change_req after all pending TLPs are drained on both the VC0 and VC1 data paths. This signal must be kept asserted until |
| cfg_edr_enable | Output | 1 | This signal reflects the state of Physical Function #0, CCIX Transport |

# CCIX Transport and Protocol Layer DVSEC PCIe Configuration Space

CCIX Transport and Protocol DVSEC layout is shown in the following figure.

*Figure 4:* **CCIX DVSEC**



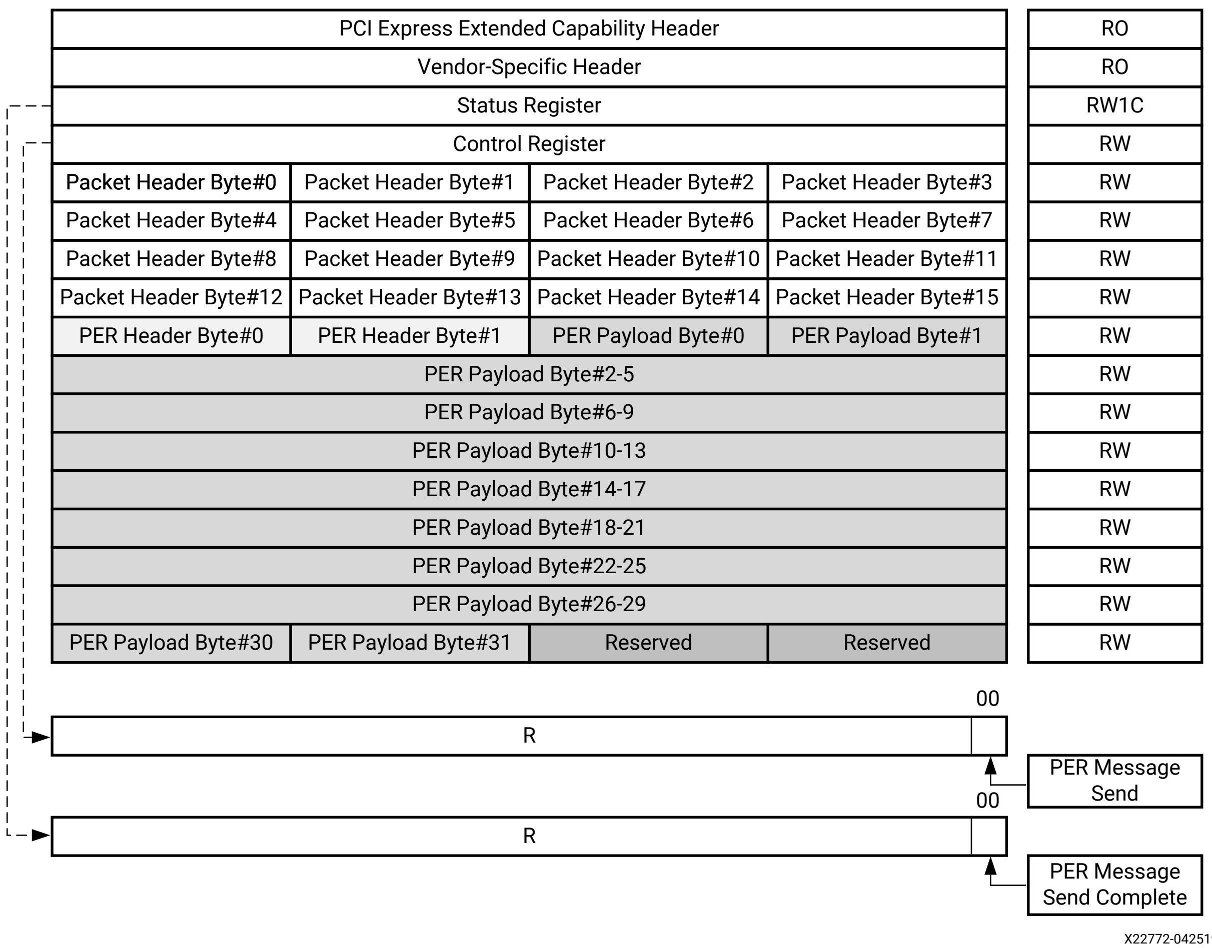

The CCIX Transport and Protocol DVSEC layout is implemented in the PCIe extended space area of the configuration space, as defined in the *Versal ACAP CPM Mode for PCI Express Product Guide* (PG346). CCIX Transport DVSEC is implemented in Physical Function #0 only, whereas CCIX Protocol DVSEC is implemented in both Physical Function #0 (PF0) and Physical Function #1 (PF1).

# CCIX PER Message

The CCIX Protocol Error message (PER) is supported through CCIX Transport Vendor Specific Extended Capability (DVSEC) located in Function #0. This capability is in PCI configuration space starting at byte address 0xDBC (DWORD/register address 0x36F). The Capability registers are shown in the following figure.

***Note*:** This region is accessed by the firmware running on the Versal™ device to generate PER messages. This is not to be used by the host side software.

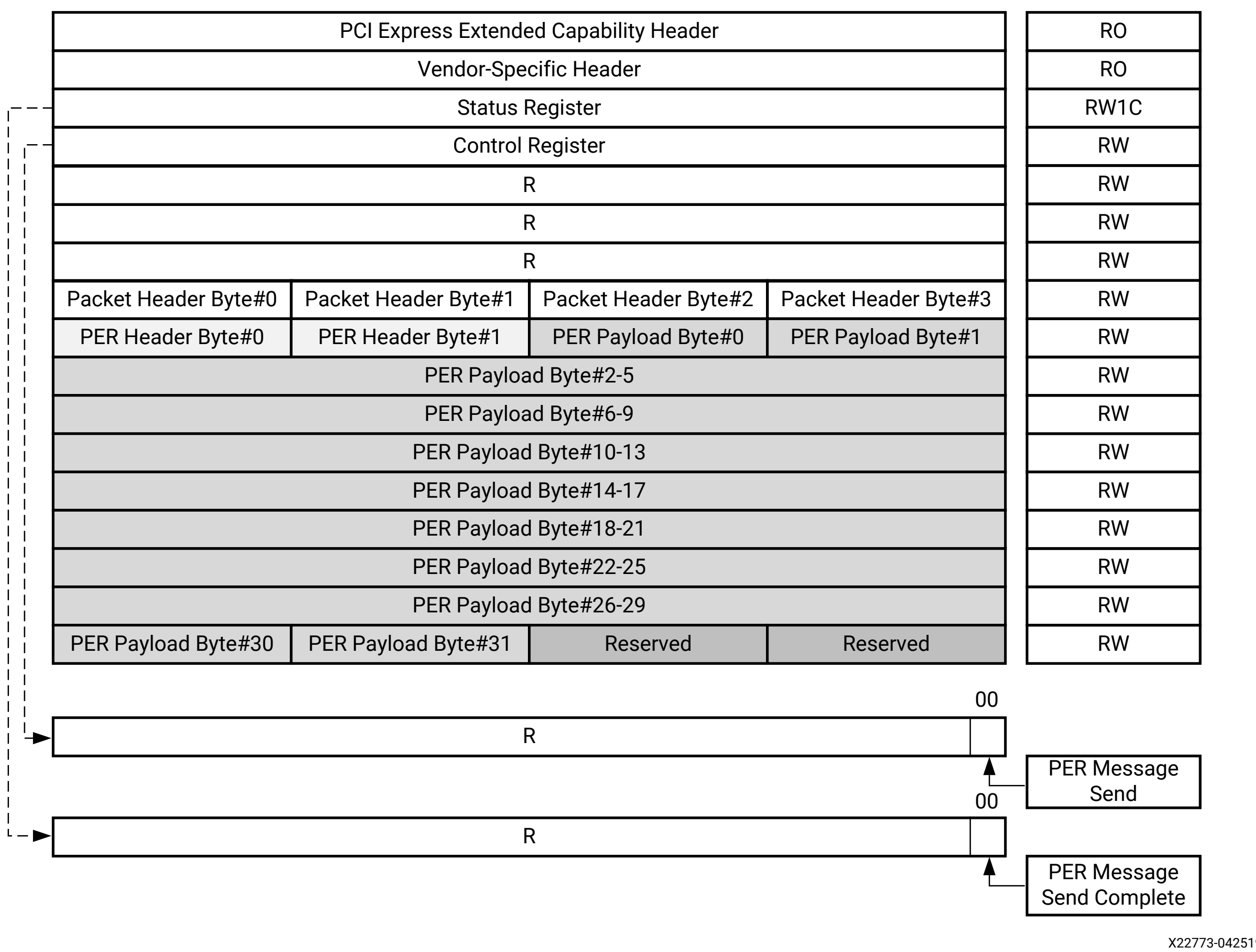
*Figure 5:* **PER Message Capability (Non Optimized TLP Format)**



Control register (byte address 0xDC8), bit 0, is PER Message Send. Writing a 1 to this register asserts *cfg2tl_ccix_per_msg_send* to the transaction layer, sending the PER message. Reads to this register always return 0. Status register, (byte address 0xDC4), bit 0, is PER Message Send Complete. This bit is set by the signal *tl2cfg_ccix_per_msg_send_done* from the transaction layer. Writing a 1 to this register clears the status. When a 1 is written to the PER Message Send register bit, the contents of the Packet Header, PER Header, and PER Payload registers are sent by the transaction layer. The non-optimized TLP format is shown in the following figure.

For optimized TLPs, the Packet Header Byte#0 to Byte#3 (byte address 0xDD8) is sent (as shown in the following figure), along with the PER Header and PER Payload. The PER Header bytes and PER Payload bytes are from section 14.6.5 PER message of the CCIX Protocol specification.

*Figure 6:* **PER Message Capability (Optimized TLP Format)**



# Clock Frequencies and Interface Widths in EDR Mode

*Table 3:* **Clock Frequencies and Interface Widths**

| PCIe Link Speed Cap | PCIe Link Width Cap | AXI4 Streaming Interface Data Width (bits) | user_clk2 Frequency (MHz) G1-4/E1-4/E5/E6 |
|---|---|---|---|
| EDR | x4 | 256 | 250/250/312.5/390.625 |
| | x8 | 512 | 500/500/625/781.25 |

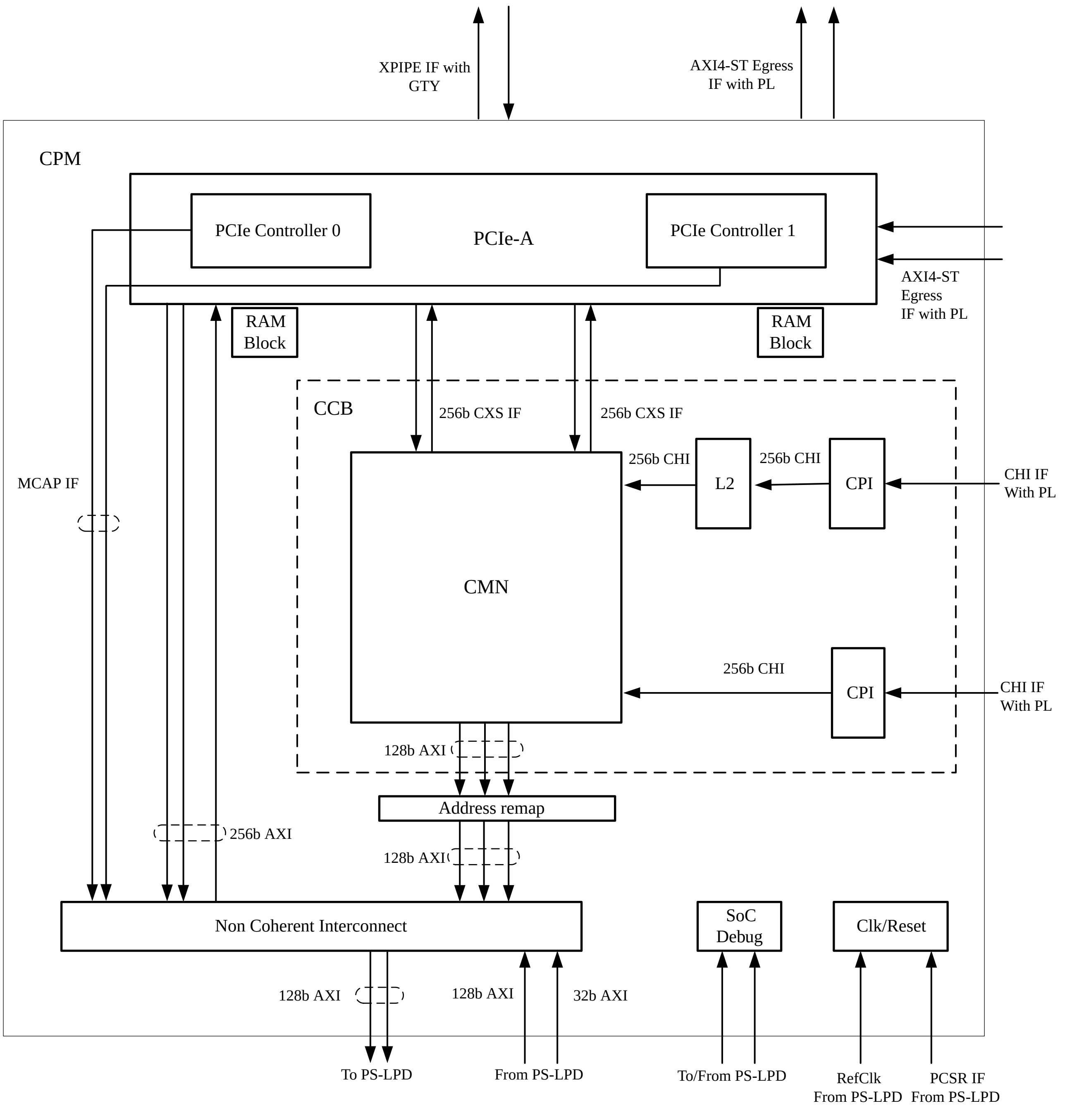
**Notes:**

1. EDR mode data rates:
    a. Required Data Rates: 2.5GT/s (E1), 5.0 GT/s (E2), 8.0 GT/s (E3), 16.0 GT/s (E4), 20.0 GT/s (E5), and 25.0 GT/s (E6)
2. EDR mode configurations:
    a. Initial link up in PCIe compliant mode (before transitioning to EDR mode) can be based on either x4Gen4 or x8Gen4 capabilities.
    b. Link width achieved during initial link up in PCIe compliant mode must be equal to the configured width to achieve the configured link width in EDR mode.

# CCIX Architecture

This chapter describes the various components in CPM for CCIX operation. The CCB (Cache Coherency Block) specifies CPM cache and coherency architecture. CCB is the CCIX gateway to on-chip and off-chip processing nodes and memory. The CCIX gateway capabilities are achieved by a combination of the CHI-B and PCI Express transport interconnect. The transport requirements for the integrated block for PCI Express specific to CCIX have been covered in Chapter 2: CCIX Capable PCIe Controller. This chapter describes the other blocks in detail. The following figure shows the architectural details of the CPM block. For information on clock speeds, refer to data sheet.

*Figure 7:* **CPM Block Diagram**




The CCB manages coherency between local accelerators (in programmable logic) and other CCIX devices including the host. There are two CHI-B interfaces exposed to the programmable logic for accelerator connectivity to the CCB. The first CHI-B port connects to the CCB via the asynchronous CPI module and L2 cache. The second CHI-B interface connects to the CCB vis the asynchronous CPI module. There are two memory ports (AXI4 interfaces) from the CCB that connect (via the PS) to the NoC and then to the DDR.

# Cache Coherency Block

The Cache Coherency block provides the home-node cache-coherency network, coherent programmable logic accelerator ports into the cache-coherency network, and second-level caches. The various components in the CCB are described in the following sections.

## Coherency Mesh Network

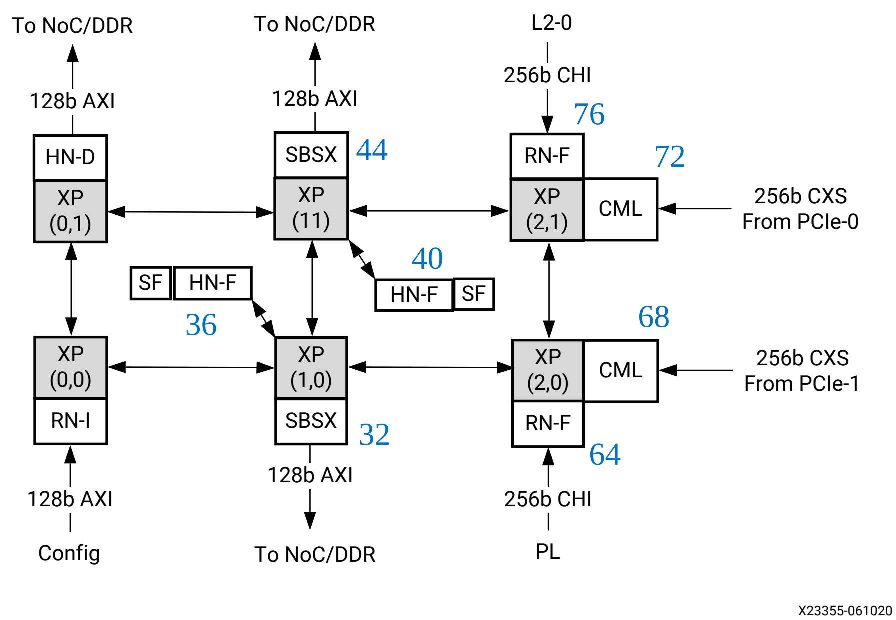
The Coherency Mesh Network (CMN) block provides the functionality required for the CCIX accelerator side cache-coherent home node. It is responsible for maintaining cache coherency between all of the requester nodes in the system. The main functions of CMN are:

- **Point of Coherency:** (PoC) is responsible for coordinating coherence actions for address ranges mapped to the home node. It includes serializing requests to any single address, snooping all of the other caches in the system, accessing memory for read/write access when required, and sending appropriate completion indication to the requesting agent.
- **Snoop Filter:** (SF) keeps track of addresses mapped to local memory that have been cached by caches in the system. CCB snoop filters are split into two halves to provide the required snoop bandwidth with 2 MB equivalent tags for each SF, tracking a total of 4 MB cache tags. It has precise tracking for up to 2 local request agents and 16 remote request agents out of a maximum possible 64 CCIX request agents. When the number of remote RA exceeds 16, the SF sends unicast snoops to all CCIX request agents present.
- **Coherent Multi-chip Link:** (CML) provides bridge functionality between CHI-B and CCIX protocol. The CCIX streaming interface (CXS) is implemented as an interface to the CCIX transport layer (PCIe in this case). The CML block receives transactions from CMN over CHI-B and converts them to a CXS stream sent to the PCIe controller. In the other direction, it receives a CXS stream from the PCIe controller and converts it to CHI-B transactions to be consumed by the CMN.

### *CMN Topology*

The CMN is a scalable mesh network of interconnect to which different agents in the CHI protocol can be attached. The following figure shows the CMN topology in the CPM. The numbers adjacent to each agent in the figure indicate node id.

*Figure 8:* **CMN Topology in the CPM**




XP (crosspoint) is a switch and router logic unit and the fundamental component of the CMN interconnect. Each XP unit consists of four interconnect ports to connect to adjacent XP units and two device ports to connect to the different agents in the CHI. As shown in the figure, the CMN has six XP units configured as a 3x2 mesh network. RN-F refers to a fully coherent requesting node, HN-F refers to a fully coherent home node, and HN-D refers to the device which includes the I/O home node, DVM node, and configuration node.

***Note*****:** The CMN does not support data check (parity) on the CHI interfaces and the CXS interfaces. Because of this the CPM does not provide end-to-end parity protection for CCIX use cases.

**IMPORTANT!** *The CMN has some registers that are required to be accessed by TZ secure transactions. The Master used to access these registers should be properly configured to generate secure transactions. Refer to the CMN TRM for information on registers that need TZ access.*

RN-F and CML are assigned to common a crosspoint. Sharing a crosspoint for an RN-F/CML combination allows direct connectivity (thereby avoiding mesh traversal) for the following traffic types:

- Remote request traffic from accelerator kernel in the PL
- Remote snoop traffic from the CCIX

This benefits both latency and bandwidth between the RN-F and CML.

HN-F provides home node functionality including snoop filters. SBSX is a CHI-to-AXI bridge that allows the CMN to connect to a memory controller. HN-F and SBSX share a crosspoint. This allows direct connectivity providing latency and bandwidth benefits to the following traffic types:

- Memory traffic between HN-F and SBSX

- Snoop data from HN-F

***Note***: Each memory port from the CCB is capable of issuing 32 cacheline size (64 byte) write transactions with unique AXI IDs per write and 64 cacheline (64 byte) read transactions with unique AXI IDs per read. Address interleaving to DDR (when CCIX is enabled) is handled by CMN-600 using address hashing. Both ports from CCB into the PS interconnect route all transactions based on address without performing interleaving.

***Note***: The following features are not supported:

- CHI-B direct memory transfers (where the SBSX does not transfer data to the RN-F or CML, directly bypassing HN-F)
- CHI-B and CXS data check
- CHI-B direct cache transfer (direct transfer between the CML and RN-F when either one has a dirty copy bypassing the HN-F)
- CCIX partial cache states and 128 byte cacheline support (this prevents the accelerator from using CHI-B *partial* opcodes)

## *Interrupts and Error Handling*

Errors detected by a component in CMN are classified into three main categories as shown in the following table:

*Table 4:* **Interrupts and Error Handling**

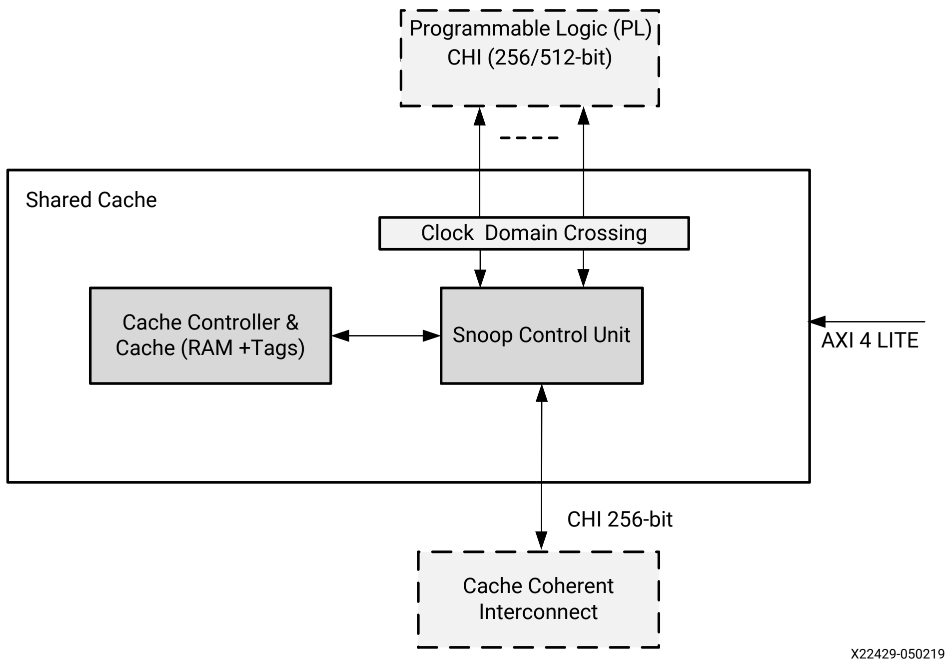
| Error Type | Description | Examples | Action taken by hardware |
|---|---|---|---|
| Correctable Errors | Errors that can be corrected using ECC or other methods. | • A single-bit ECC error in any of the RAMs<br>• An error that is recovered using a local retry | 1. Logs the error.<br>2. Counts the occurrence of these errors.<br>3. Signals the error to the global RAS block that can be controlled using a threshold count. |
| Deferred Errors | Uncorrectable errors detected in one node of the CMN, but the data is not used within the same node, and poison bits are set for the data. With these errors, the system can typically operate for a period of time without being corrupted. | A request packet received with an unsupported opcode. | 1. Sends a response with a RespErr value of data error or non-data error.<br>2. Logs the error.<br>3. Signals error to the global RAS block within the CMN. |
| Uncorrectable Fatal Errors | These are errors in the control logic in a node, where continuing operation might corrupt the system beyond recovery. | • A double-bit ECC error in a data read from snoop filter<br>• A packet received with an error in the target ID<br>• An internal logic error | 1. Logs the error.<br>2. Signals the error to the global RAS block. |

The global RAS (reliability, availability, and serviceability) block signals these error interrupts which are handled by the CCIX firmware. The firmware is responsible for generating the protocol error reporting (PER) message packet, as appropriate. Details on PER generation are covered in Chapter 2: CCIX Capable PCIe Controller.

## Level 2 Cache

The integrated shared level-2 cache is provided to minimize the latency and bandwidth of the acceleration function in programmable logic to memory accesses. The accelerator function can implement a small level-1 cache as well.

*Figure 9:* **CPM Shared Level 2 Cache**




The L2 block with the cache controller also provides an AXI4lite interface to access the internal configuration registers. The slave port of the L2 block instance connects to the CMN on the RN-F device port. Main components of the L2 cache controller block are:

- Physically addressable, physically tagged cache of 1 MB in size
- Fixed cacheline size of 64 bytes
- 48-bit physical address
- Snoop control unit

L2 cache supports atomic transactions (on the CHI) with SnoopMe opcode enabled. Additional notes are as follows:

- **L2 Tag ECC double bit error interrupt:** This applies to L2 cache tag and directory double-bit ECC errors. The appropriate invalidation engine is required to be triggered as part of the uncorrectable/fatal interrupt handling sequence to invalidate the contents of the tag or directory arrays.
- **First WriteBack request generation at L2-CMN:** After the CleanInvalid request appears at the CPI-L2 interface, L2 takes about 36 clock cycles to generate the first WriteBack request at the L2-CMN interface.
- **Write request:** L2 waits to receive the write data before it forwards the write request downstream.

# Address Re-map Block

When a home agent or slave agent is enabled in CPM, the host assigns address space to the memory advertised using HA or SA. The host and the accelerators in programmable logic share the same coherent memory space, but this might conflict with the Versal™ ACAP system address map. The physical address ranges that have a conflict need to be mapped to different memory regions of the Versal ACAP address map. The address re-map block performs this function. It can re-map up to eight address regions. The address re-map block remaps the addresses of three AXI interfaces from the CMN to the NoC. The same eight address regions apply across all three AXI interfaces. The address re-map registers are programmed by the CCIX firmware with the host-programmed base address table (BAT) as the source address and the memory address assigned by the Xilinx tools as the destination address.

In addition to address re-mapping, the local memory owned by CCIX-HA or CCIX-SA is reserved and protected by the memory protection unit (XMPU) so that no other master outside of the CCIX coherency domain can access it. These isolation and protection settings are performed by the Xilinx Control, Interfaces, and Processing System (CIPS) IP core.

# Non Coherent Interconnect

The non coherent interconnect connects the CPM, processing system (PS), and DDR via the NoC. It is important to note that the processing system is not in the CCIX coherency domain. This block receives AXI transactions from the PCIe controller block and CMN, and forwards them to the PS after performing a clock domain transfer. In the other direction, it receives AXI transactions on two AXI slave ports from the PS and forwards them to the destination block in the CPM after a clock domain transfer. An ingress AXI interface configures the different blocks in the CPM.

### SMID Mapping

A 10-bit SMID field is used in the processing system to indicate context for enforcing memory protection. All AXI transactions in PS and CPM are required to carry a 10 bit SMID on A*USER[9:0] signals. This SMID is used by various memory protection elements (XMPU, XPPU) in the PS Interconnect to enforce access privileges. The SMID is applied on these transactions before they enter the Non-Coherent Interconnect. For transactions from CMN:

SMID[9:0] = { 1’b0 , BASE_SMID Register [8:0] }

## CPM PL Interface

The CPM PL Interface (CPI) interfaces with the accelerators in the PL on one side using one CHI slave port. CPI interfaces with the L2 or CMN on the other side via another CHI master port. The features of CPM PL Interface (CPI) are:

- Perform clock domain crossing of CHI port between CPM and PL.
- Perform up-sizing/downsizing of the data channels between 512-bit CHI interface on the PL side and 256-bit CHI interface on the L2 side.
- Perform credit management at L2 and PL interfaces in both directions independently.
- Track link states independently in both directions.
- Follow the link activation and deactivation sequences as described in the CHI specification.
- Support for L2 bypass feature.

## CCIX DVSEC Support

CCIX devices are discovered and managed as PCIe devices. Components and their capabilities are discovered through a Designated Vendor-Specific Extended Capability (DVSEC) region in the PCIe configuration space. The CCIX DVSEC defines capabilities and also provides fields for control and status. A CCIX device is required to support the following DVSEC regions:

- Transport DVSEC which contains the control and status registers for CCIX Physical, Data Link and Transport layers
- Protocol DVSEC which contains control and status registers for CCIX protocol layer

These DVSEC regions are implemented in PCIE_DVSEC_0 and PCIE_DVSEC_1 regions for each PCIe port. A DVSEC firmware library is responsible for populating the appropriate protocol DVSEC capability, control and status registers in this region based on user configuration selection. At runtime when host CCIX discovery software programs the control fields in protocol DVSEC region, the firmware programs the corresponding registers in various blocks such as CMN-600, L2 cache, address remap etc. Likewise, in case of errors, firmware updates appropriate status bits in DVSEC region and is also responsible for triggering a Protocol Error Reporting (PER) message via interface provided by PCIe controller. For more details on DVSEC region and PER message generation, see Chapter 2: CCIX Capable PCIe Controller.

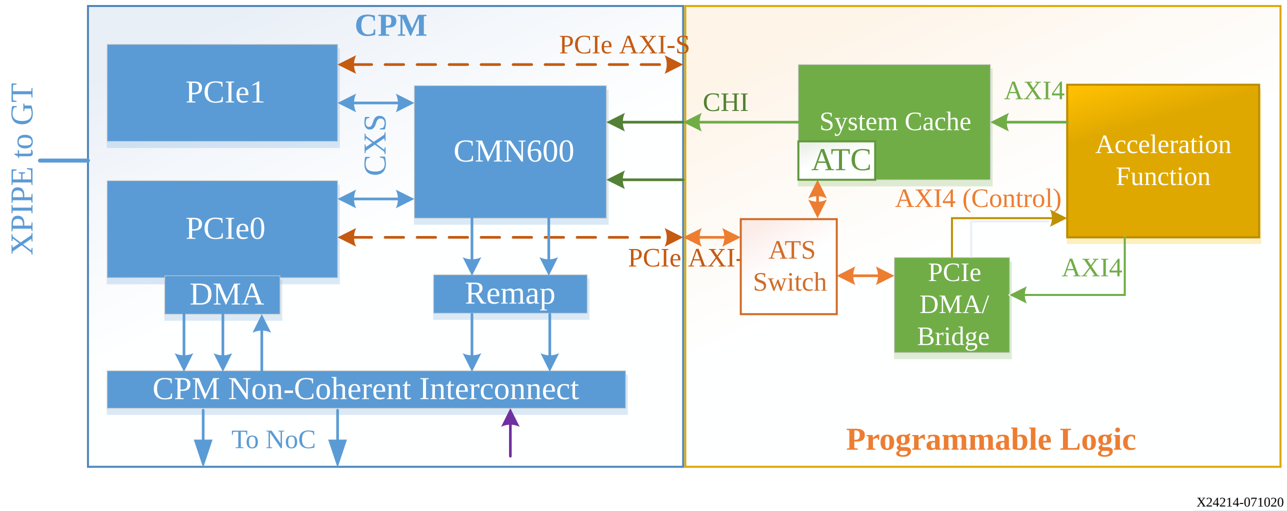
# CCIX Transaction Flows

This section describes the transaction flows for various transactions and agents in detail. Using the CPM as the CCIX-RA requires additional logic in the programmable logic along with the accelerator kernel. The additional modules required for CCIX-RA mode are discussed first.

**Request Agent**: A Request Agent (RA) is CCIX agent that is the source of the read and write transactions. Each of the CCIX RAs might have one or more internal initiators, also referred to as acceleration functions (AF).

The following block diagram provides an overview of PL modules required for CCIX-RA.

*Figure 10:* **CPM as CCIX-RA with Programmable Logic Block Diagram**




- The PCIe AXI-S interface out from CPM PCIe controller connects to the ATS switch which directs the ATS (Address Translation Services) and received transactions towards the ATC (Address Translation Cache).
- The PCIe AXI bridge interfaces with the PCIe AXI4-Stream interface for VC0 from the ATS switch block and provides a control interface to acceleration functional units.
- The system cache provides a user-consumable AXI interface instead of the CHI interface from the CPM into the PL. It also provides additional caching support if enabled during IP configuration The system Cache also implements the ATC logic.
- The user kernel is the acceleration function.

For an application on the host system that programs the user kernel with virtual addresses, the transaction flow is as follows:

- The system cache IP implements the address translation cache (ATC) and address translation services (ATS). When the ATS is enabled, for any received AX4 request, the system cache IP checks the ATC. In case of a hit, it uses the translated address from the ATC. In case of a miss, it issues an ATS request for this virtual address over the AXI4-Stream interface provided by the PCIe AXI bridge IP, uses the translated address received, and also caches the translation in the ATC.
- For kernel issuing read or write transaction to a virtual address, system cache performs address translation and then looks up its cache for hit. If there is a miss, the transaction is sent to the CPM via the CHI interface exposed in the PL.

### Transaction Flows

This section describes the basic transaction flows. With regards to terminology, 'local' refers to the device itself and 'remote' refers to peer agents accessed over the link.

- **Local requests to local memory – local cache hit:** In this scenario, the kernel in programmable logic generates a memory access that references local memory. This transaction can be a read or a write. It can hit in either the Level-1 cache mapped to the PL (implemented in System Cache IP) or the Level-2 cache (in CPM), if present and enabled. No request needs to be propagated beyond the Level-2 cache, if present and enabled, if the cached copy is unique.

- **Local requests to local memory – remote cache hit:** In this scenario, the reference to local memory generated by the kernel misses in all the local caches. The Home Agent serializes the request with respect to all the other requests in the system. If the snoop-filter indicates that a remote node might have a cached copy, then a snoop is sent to those caches (via the PCIe). A cache hit may result in data being returned to the requester (kernel in programmable logic in this case).

- **Local requests to local memory – local memory access:** In this scenario, the local caches miss & the snoop-filter indicates that there are no cached copies in the system. The Home Agent will read the data from the local DDR memory (SBSX via NoC) and return it to the kernel in programmable logic.

- **Local requests to remote memory – local cache hit:** In this scenario, the local kernel generates a memory access that references remote memory. The transaction can be a read or a write. It can hit in either the Level-1 cache mapped to the PL (implemented by System Cache IP) or the Level-2 cache, if present and enabled. No request needs to be propagated beyond the Level-2 cache, if present and enabled, if the cached copy is unique.

- **Local requests to remote memory – local cache miss:** In this scenario, the request is transmitted to the remote home-node where it is serialized at the PoC of the remote home-node. The remote home-node's PoC snoops other caches in the system if the snoop filter (on the remote home node) indicates that they have a cached copy, or the remote home-node's PoC sends a broadcast snoop if no snoop filter is present. A cache hit can return data to the remote home node and then the requestor. If all the caches miss, the PoC returns data from remote memory.

- **Remote requests to local memory:** In this scenario, requests from the remote RA to the accelerator-attached memory are received. The Home node PoC in the CCB serializes the requests against all the requests to the same address. The snoop filter is looked up and snoops are sent to caches as indicated by the snoop filter. In addition to the snoops, the PoC can also access local memory to satisfy the request.

- **Remote snoops:** Remote snoops arrive for remote references to addresses cached in the accelerator. The snoops look up the caches, update the state if necessary, and generate responses according to the protocol.

**IMPORTANT!** *PL system cache to CML WriteUnique (WU) bandwidth discrepancy: WU bandwidth drops from an ideal 16 GB/s through CML to 12.8 GB/s. This bandwidth drop is only observed through L2. Therefor if a use-case requires high WU bandwidth, then the L2 instance can be bypassed.*

# Clock, Reset, and Debug

This chapter details the clocking, reset, and debug details of the CPM specific to CCIX functionality. Details of clock and reset for CPM PCIe controllers are covered in Chapter 2: CCIX Capable PCIe Controller. The section on debug also describes the debug capabilities available within the CPM specific to the CCIX datapath.

## CPM Clock/Reset Microarchitecture

The CPM clock and reset module receives the same POR (Power On Reset) as the PL (por_vccint_b), and is driven by the Platform Management Controller (PMC).

### Clocking Architecture

The CPM has an internal PLL to generate the 1.2 GHz CPM core clock. The reference clock for this PLL comes from either the PMC or the PL. It receives the other clocks from the GTx, PS, and PL. Some clocks are internally generated by dividing the CPM core clock. The clocks in the CPM are listed in the following table.

The CPM sends 2 clocks: pcie0_user_clk and pcie1_user_clk – to 2 DPLLs in the GT clocking column of the PL. Each DPLL is used to deskew the respective pcie_user_clk and the output clock from the DPLL is then used by the PL to drive/capture signals on the AXI4-ST PCIe-PL interface. The following table lists the frequencies of the clocks used in typical CCIX design. Refer to the *Versal ACAP CPM Mode for PCI Express Product Guide* (PG346) for details on supported frequencies in the PL for pcie0_user_clk and pcie1_user_clk. The following table emphasizes that pcie_user_clk and pl_chi_clk are asynchronous.

*Table 5:* **Clocks**

| Clock Name | Source | Fmin (MHz) | Fmax (MHz) | Clk group | Description |
|---|---|---|---|---|---|
| pl_chi0_clk | PL | 100 | 391 | Async | Clock used by CPM-PL CHI port 0. This is generated from a PLL that resides in the PL. |
| pl_chi1_clk | PL | 100 | 391 | Async | Clock used by CPM-PL CHI port 1. This is generated from a PLL that resides in the PL. |

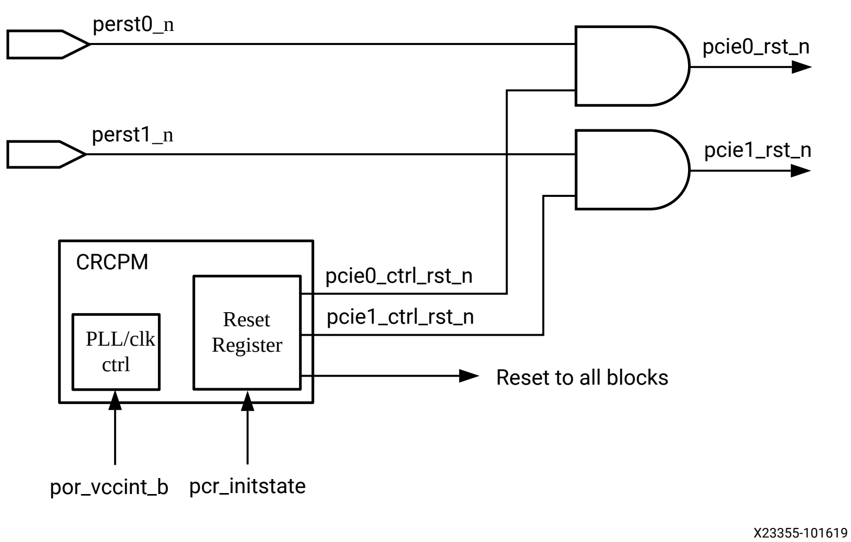
*Table 5:* **Clocks** *(cont'd)*

| Clock Name | Source | Fmin (MHz) | Fmax (MHz) | Clk group | Description |
|---|---|---|---|---|---|
| pcie0_user_clk | CPM internal | 62.5 | 500 | PCIe0 | Clock for PCIe0 AXI4-stream and CFG interfaces. 500 MHz in PL is not supported |
| pcie1_user_clk | CPM internal | 62.5 | 500 | PCIe1 | Clock for PCIe1 AXI4-stream and CFG interfaces. 500 MHz in PL is not supported |

## Reset Architecture

The CPM block receives a few reset inputs, as shown in the following figure. All are treated as asynchronous resets. The following figure shows the reset connectivity in the CPM.

*Figure 11:* **CPM Reset Architecture**




*por_vccint_b* is the main reset signal for the CPM block. This is driven by the PMC and is the same reset that is applied to the PL. *por_vccint_b* is only used by the CRCPM block in the CPM. It resets the PLL and clock divider and other related circuits in the CRCPM. All internal clocks in the CPM are active only after the de-assertion of *por_vccint_b*.

The CRCPM block includes reset registers to all sub-blocks in the CPM. By default all sub-blocks are held in reset. The PCSR interface between the PS-LPD and CPM contains the pcr_initstate signal that acts as the system reset for the CPM. This reset needs to be de-asserted before the CPM Interconnect can be accessed. After pcr_initstate is removed, the reset registers can then be configured by means of the CPM Interconnect to bring individual sub-blocks out of reset.

*Table 6:* **CPM Resets**

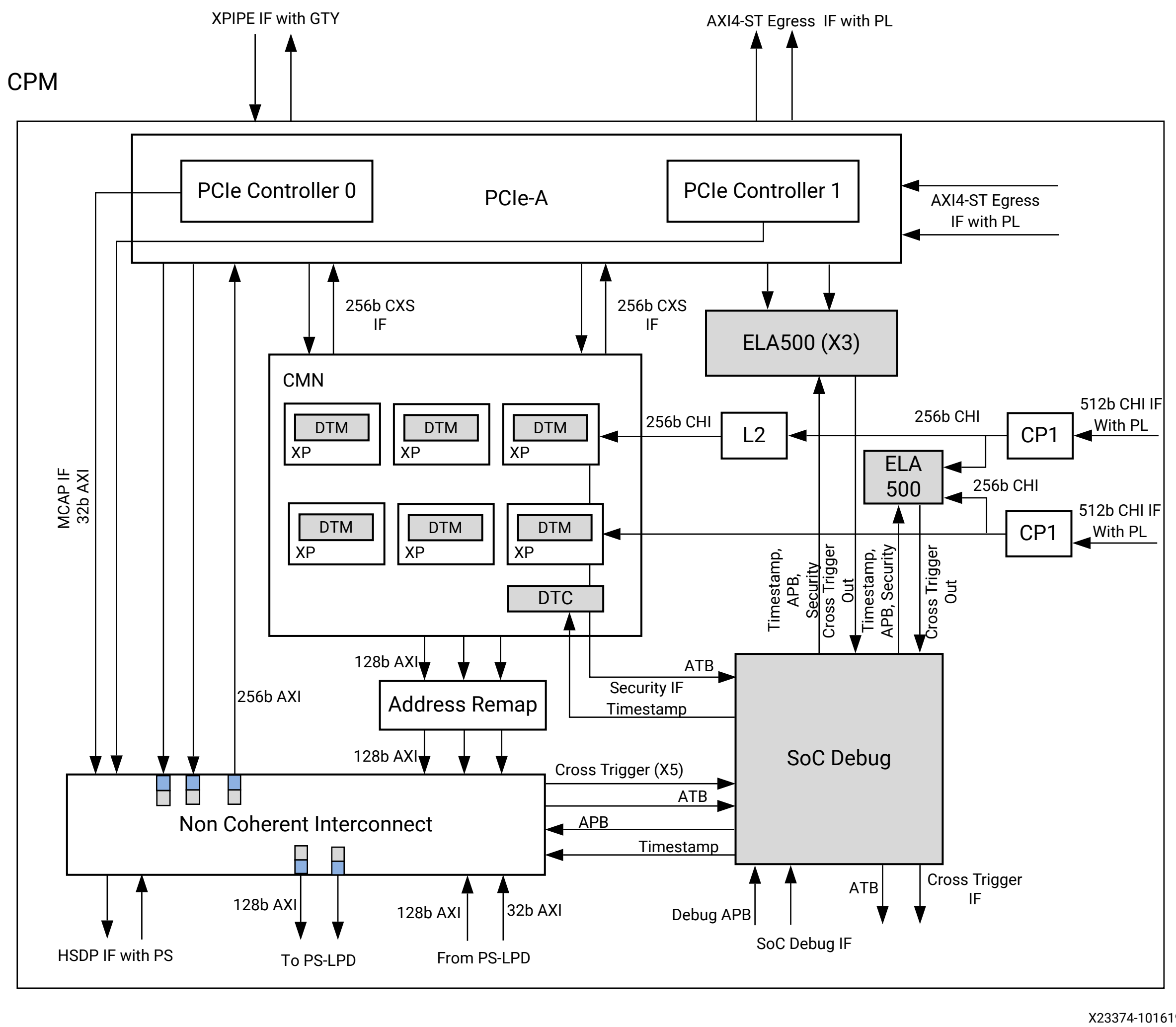
| Reset name | Direction | Description |
|---|---|---|
| por_vccint_b | Input | Power on reset for CPM. Driven by PS/PMC. |
| if_ps_cpm_pcsr.pcr_initstate | Input | Serves as CPM system reset. |
| perst0 | Input | Fundamental reset of PCIe controller. One per instance. Driven by IO inside PS. |
| perst1 | | |
| perst0_out | Output | Fundamental reset out from the PCIe controller. One per instance. This is simply a buffered output of perst0/1 and goes out to XPIPE. |
| perst1_out | | |

***Note*:** The reset inputs are defined to be asynchronous, as PCIe defines PERST# as asynchronous. These resets can assert without notice asynchronously with transactions outstanding. This has the potential to cause orphaned transactions in CPM, PS, PMC, NOC, DDR etc. An optional firmware-based reset sequence is required to protect against this issue.

# Debug Architecture

The CPM SoC Debug block interfaces with the Master SoC debug block in the PS and interacts with debug components in the CPM. The debug architecture in the CPM is based on ARM CoreSight technology. The following figure shows the various debug components in the CPM. The CPM includes an SoC Debug block that interacts with the debug components in various sub-blocks and transfers the information to the master debug block in the PS. The debug sub-system in the CPM then acts as a slave to the PS. Refer to the *Versal ACAP Technical Reference Manual* (AM011) for more information on SoC debug in the PS sub-system.

*Figure 12:* **CPM Debug Architecture**




CXS interfaces are observable through the extended logic analyzer (ELA). The connection details are provided in the following table.

*Table 7:* **ELA Signal Assignment – CXS Interfaces**

| ELA Instance | Signal Group | Bits | CXS Signal |
|---|---|---|---|
| ELA 0 | SIGNALGRP9 | [0:0] | cxs_active_req_tx |
| ELA 0 | SIGNALGRP9 | [1:1] | cxs_active_ack_tx |
| ELA 0 | SIGNALGRP9 | [2:2] | cxs_deact_hint_tx |
| ELA 0 | SIGNALGRP9 | [16:3] | cxs_cntl_tx |
| ELA 0 | SIGNALGRP9 | [17:17] | cxs_valid_tx |
| ELA 0 | SIGNALGRP9 | [18:18] | cxs_crdgnt_tx |
| ELA 0 | SIGNALGRP9 | [19:19] | cxs_crdrtn_tx |
| ELA 0 | SIGNALGRP9 | [51:20] | cxs_data_chk_tx |

*Table 7:* **ELA Signal Assignment – CXS Interfaces** *(cont'd)*

| ELA Instance | Signal Group | Bits | CXS Signal |
|---|---|---|---|
| ELA 0 | SIGNALGRP9 | [52:52] | cxs_cntl_chk_tx |
| ELA 0 | SIGNALGRP9 | [53:53] | cxs_valid_chk_tx |
| ELA 0 | SIGNALGRP9 | [54:54] | cxs_crdgnt_chk_tx |
| ELA 0 | SIGNALGRP9 | [55:55] | cxs_crdrtn_chk_tx |
| ELA 0 | SIGNALGRP9 | [56:56] | cxs_active_req_rx |
| ELA 0 | SIGNALGRP9 | [57:57] | cxs_active_ack_rx |
| ELA 0 | SIGNALGRP9 | [58:58] | cxs_deact_hint_rx |
| ELA 0 | SIGNALGRP9 | [72:59] | cxs_cntl_rx |
| ELA 0 | SIGNALGRP9 | [73:73] | cxs_valid_rx |
| ELA 0 | SIGNALGRP9 | [74:74] | cxs_crdgnt_rx |
| ELA 0 | SIGNALGRP9 | [75:75] | cxs_crdrtn_rx |
| ELA 0 | SIGNALGRP9 | [107:76] | cxs_data_chk_rx |
| ELA 0 | SIGNALGRP9 | [108:108] | cxs_cntl_chk_rx |
| ELA 0 | SIGNALGRP9 | [109:109] | cxs_valid_chk_rx |
| ELA 0 | SIGNALGRP9 | [110:110] | cxs_crdgnt_chk_rx |
| ELA 0 | SIGNALGRP9 | [111:111] | cxs_crdrtn_chk_rx |
| ELA 0 | SIGNALGRP9 | [159:128] | cxs_data_tx[ 31: 0] |
| ELA 0 | SIGNALGRP9 | [191:160] | cxs_data_tx[159:128] |
| ELA 0 | SIGNALGRP9 | [223:192] | cxs_data_rx[ 31: 0] |
| ELA 0 | SIGNALGRP9 | [255:224] | cxs_data_rx[159:128] |
| ELA 1 | SIGNALGRP9 | [255:0] | cxs_data_tx |
| ELA 2 | SIGNALGRP9 | [255:0] | cxs_data_rx |
| ELA 0 | SIGNALGRP10 | [0:0] | cxs_active_req_tx |
| ELA 0 | SIGNALGRP10 | [1:1] | cxs_active_ack_tx |
| ELA 0 | SIGNALGRP10 | [2:2] | cxs_deact_hint_tx |
| ELA 0 | SIGNALGRP10 | [16:3] | cxs_cntl_tx |
| ELA 0 | SIGNALGRP10 | [17:17] | cxs_valid_tx |
| ELA 0 | SIGNALGRP10 | [18:18] | cxs_crdgnt_tx |
| ELA 0 | SIGNALGRP10 | [19:19] | cxs_crdrtn_tx |
| ELA 0 | SIGNALGRP10 | [51:20] | cxs_data_chk_tx |
| ELA 0 | SIGNALGRP10 | [52:52] | cxs_cntl_chk_tx |
| ELA 0 | SIGNALGRP10 | [53:53] | cxs_valid_chk_tx |
| ELA 0 | SIGNALGRP10 | [54:54] | cxs_crdgnt_chk_tx |
| ELA 0 | SIGNALGRP10 | [55:55] | cxs_crdrtn_chk_tx |
| ELA 0 | SIGNALGRP10 | [56:56] | cxs_active_req_rx |
| ELA 0 | SIGNALGRP10 | [57:57] | cxs_active_ack_rx |
| ELA 0 | SIGNALGRP10 | [58:58] | cxs_deact_hint_rx |
| ELA 0 | SIGNALGRP10 | [72:59] | cxs_cntl_rx |
| ELA 0 | SIGNALGRP10 | [73:73] | cxs_valid_rx |

*Table 7:* **ELA Signal Assignment – CXS Interfaces** *(cont'd)*

| ELA Instance | Signal Group | Bits | CXS Signal |
|---|---|---|---|
| ELA 0 | SIGNALGRP10 | [74:74] | cxs_crdgnt_rx |
| ELA 0 | SIGNALGRP10 | [75:75] | cxs_crdrtn_rx |
| ELA 0 | SIGNALGRP10 | [107:76] | cxs_data_chk_rx |
| ELA 0 | SIGNALGRP10 | [108:108] | cxs_cntl_chk_rx |
| ELA 0 | SIGNALGRP10 | [109:109] | cxs_valid_chk_rx |
| ELA 0 | SIGNALGRP10 | [110:110] | cxs_crdgnt_chk_rx |
| ELA 0 | SIGNALGRP10 | [111:111] | cxs_crdrtn_chk_rx |
| ELA 0 | SIGNALGRP9 | [159:128] | cxs_data_tx[ 31: 0] |
| ELA 0 | SIGNALGRP9 | [191:160] | cxs_data_tx[159:128] |
| ELA 0 | SIGNALGRP9 | [223:192] | cxs_data_rx[ 31: 0] |
| ELA 0 | SIGNALGRP9 | [255:224] | cxs_data_rx[159:128] |
| ELA 1 | SIGNALGRP10 | [255:0] | cxs_data_tx |
| ELA 2 | SIGNALGRP10 | [255:0] | cxs_data_rx |

ELA clocks can be gated using a ela_clkgate_en bit in CPM_SLCR.DEBUG_CTRL register. By default, ELA clocks are gated. The ela_clkgate_en bit of CPM_SLCR.DEBUG_CTRL register should be programmed to a 1 prior to using ELA. The ELA 500 instance here is used to monitor and capture various flits on the CHI0 instance (between L2 and CPI) and CHI1 instance (between CMN and CPI).

## CMN

Debug Trace capabilities of CMN includes:

- Watch point initiated transaction tracing
- CHI trace tag generation
- CoreSight ATB trace streaming
- Configuration register access to trace data
- Cross trigger support
- Secure debug support
- Event based interrupts

The CMN debug system consists of a set of debug trace monitors (DTMs) and debug trace controllers (DTCs) distributed across the mesh network. The CMN in CPM has 1 DTM instance located in each XP and 1 central DTC that is part of the HN-D component. All the DTMs collect the trace data and send it to the DTC. The trace packets can be accessed via control registers at the DTMs. The trace packets can also be streamed from the DTC over the ATB interface to the CPM SoC Debug block.

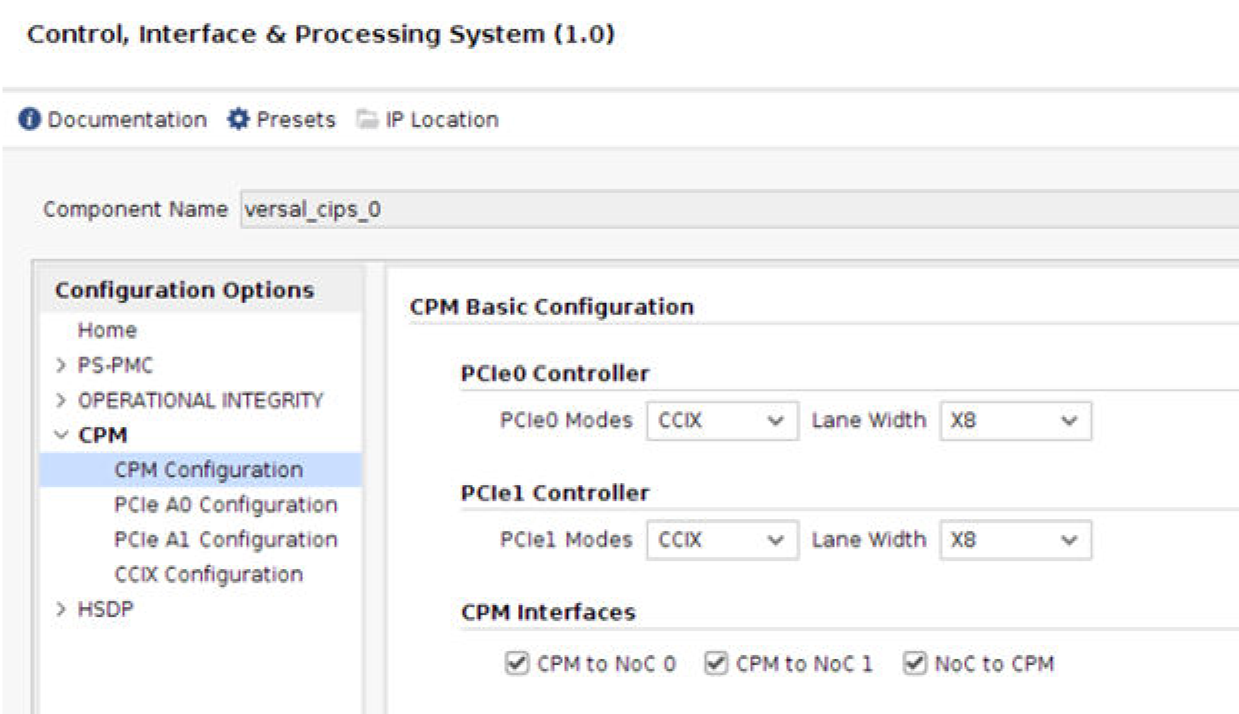
The CMN also includes Performance Monitoring Unit (PMU) capabilities. The PMU unit includes local and global counters that can be set up to monitor several types of events.

# Appendix A

# Using CIPS GUI to Configure CPM-CCIX

This section provides an overview of the Control, Interfaces, and Processing System (CIPS) IP core. CIPS is used to configure the CPM-CCIX subsystem. You can enable CCIX mode for PCIe controllers and other CPM interfaces in the main CPM configuration page..

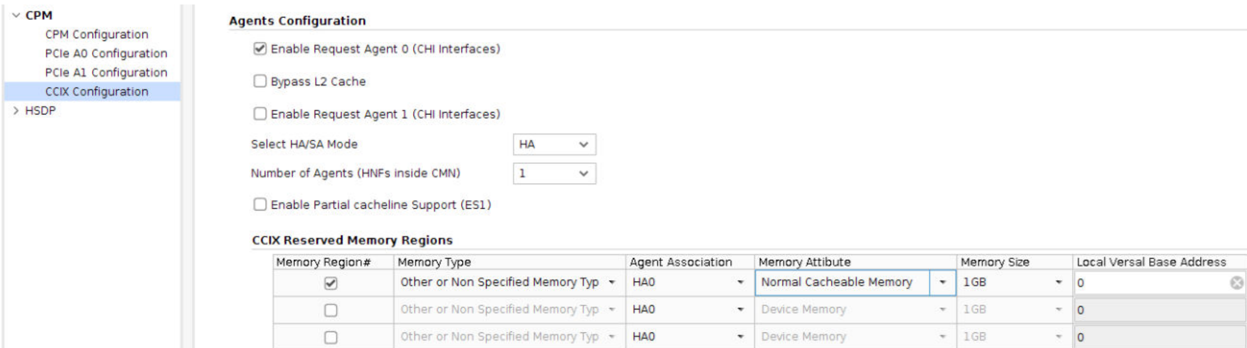
*Figure 13:* **CIPS CPM Configuration**




EDR mode can be selected in PCIe configuration windows. For agent configuration:

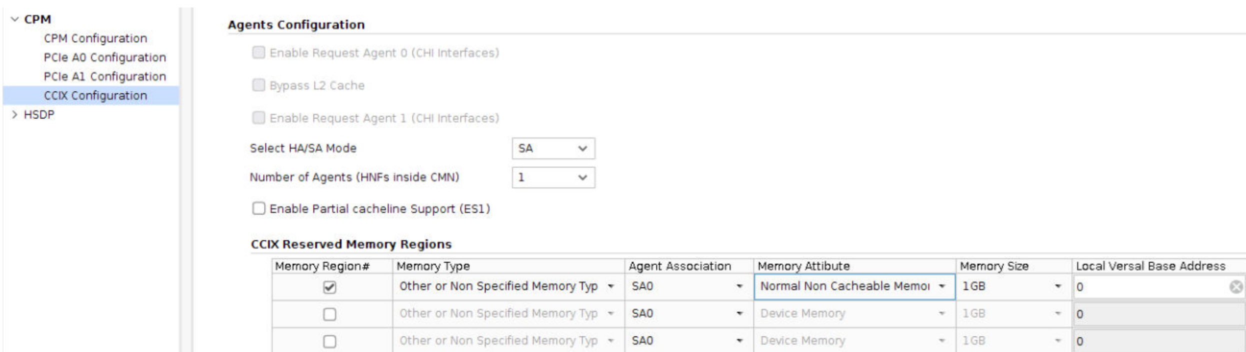
- CCIX-RA can be enabled and for RA0, L2 can be bypassed based on selection.
- CCIX-HA can be enabled (up to 2 home agents) and in the memory region table, various memory pools can be selected along with memory types and attributes.
    - When Vivado wizard assigns addresses, the local base address should be populated

*Figure 14:* **CIPS CCIX Configuration (Agents Configuration - 1)**




For slave agent selection similar options (like HA) apply except that RA cannot be enabled at the same time as SA.

*Figure 15:* **CIPS CCIX Configuration (Agents Configuration - 2)**




This programming via wizard generates relevant programming details (as configuration data object (CDO) files) to be used by the platform loader and manager (PLM).

*Appendix B*

# Additional Resources and Legal Notices

## Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Xilinx Support.

## Documentation Navigator and Design Hubs

Xilinx® Documentation Navigator (DocNav) provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open DocNav:

- From the Vivado® IDE, select **Help → Documentation and Tutorials**.
- On Windows, select **Start → All Programs → Xilinx Design Tools → DocNav**.
- At the Linux command prompt, enter `docnav`.

Xilinx Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs:

- In DocNav, click the **Design Hubs View** tab.
- On the Xilinx website, see the Design Hubs page.

***Note*:** For more information on DocNav, see the Documentation Navigator page on the Xilinx website.

## References

These documents provide supplemental material useful with this guide:

1. *Coherent Mesh Network Technical Reference Manual* https://developer.arm.com/docs/100180/0201
2. *CCIX Base Specification Rev. 1.0 v1.0* Available http://www.ccixconsortium.com/
3. *Versal ACAP CPM Mode for PCI Express Product Guide* (PG346)
4. *Versal ACAP Technical Reference Manual* (AM011)

# Please Read: Important Legal Notices

**AUTOMOTIVE APPLICATIONS DISCLAIMER**

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.

## Copyright