Versal ACAP Design Guide

UG1273 (v2020.1) July 14, 2020

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
07/14/2020 Version 2020.1	
Initial release	N/A

Table of Contents

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: System Architecture
  - Chapter 3: System Methodology
  - Chapter 4: Design Flow
  - Chapter 5: System Migration
- **Embedded Software Development:** Creating the software platform from the hardware platform and developing the application code using the embedded CPU. Also covers XRT and Graph APIs. Topics in this document that apply to this design process include:
  - Vitis Environment Design Flow
  - Simulation Flows
- **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:
  - Vitis Environment Design Flow

    - Simulation Flows

- **AI Engine Development:** Creating the AI Engine graph and kernels, library use, simulation debugging and profiling, and algorithm development. Also includes the integration of the PL and AI Engine kernels. Topics in this document that apply to this design process include:
    - AI Engine
    - Vitis Environment Design Flow
- **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:
    - Vivado Tools Design Flow
    - Simulation Flows
- **System Integration and Validation:** Integrating and validating the system functional performance, including timing, resource use, and power closure. Topics in this document that apply to this design process include:
    - System Simulation Methodology
    - System Debug Methodology
    - Chapter 4: Design Flow

About This Guide

This guide provides a high-level overview of the Versal ACAP as follows:

- **Chapter 2: System Architecture:** Provides an overview of the Versal ACAP with a summary of each high-level integrated block, including the purpose of each block and how blocks are related to each other.
- **Chapter 3: System Methodology:** Provides high-level methodology recommendations.
- **Chapter 4: Design Flow:** Describes the Xilinx design tools and supported design flows available for Versal ACAPs.
- **Chapter 5: System Migration:** Provides high-level system migration recommendations as well as block-by-block migration information for designs targeting the Versal ACAP.
- **Appendix A: Primitives:** Provides information on Versal ACAP primitives.

System Architecture

The Xilinx® Versal™ ACAP is a collection of programmable resources that work together to form a system on chip (SoC). Following are the major resource blocks:

- AI Engine

  Note: AI Engine availability is device specific.

- Programmable logic (PL)
- Network on chip (NoC)
- High-speed I/O (XPIO)
- Integrated memory controllers LPDDR4 and DDR4 (DDRMC)
- Processing system (PS)
- Platform management controller (PMC)
- Integrated block for PCIe® with DMA and cache coherent interconnect (CPM)

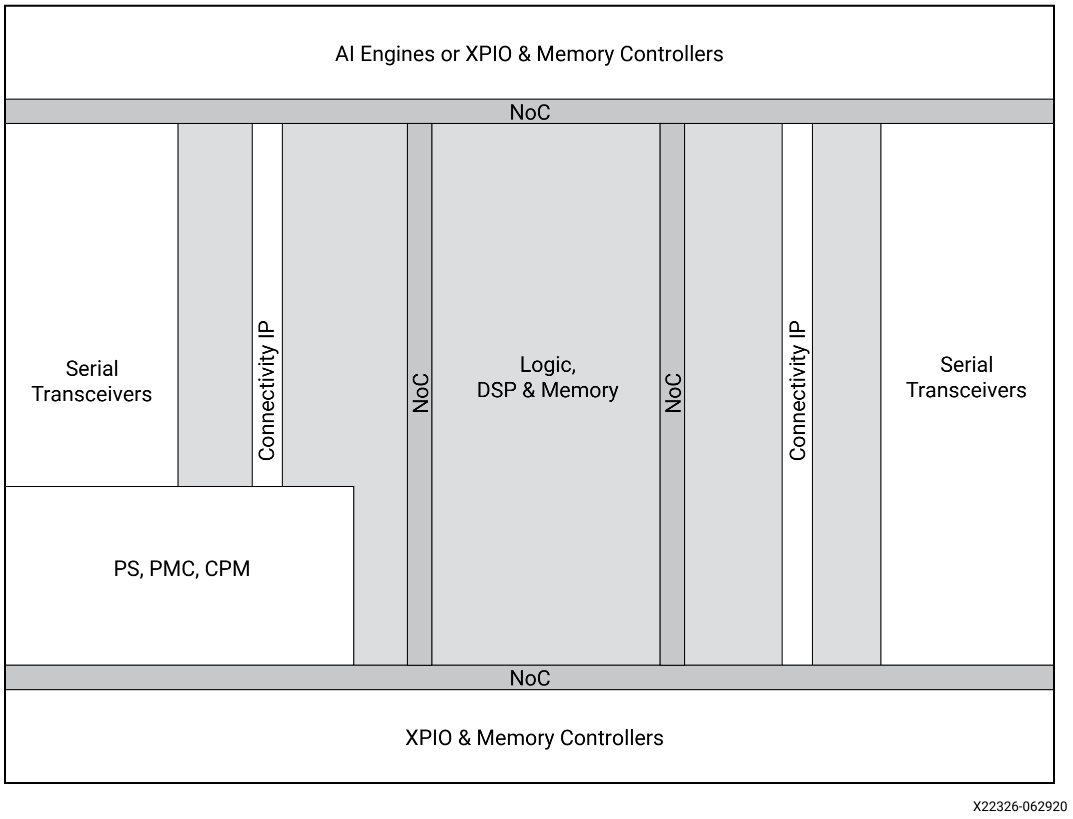
  Note: CPM availability is device specific.

- Transceivers (GT)
- High-speed debug port (HSDP)

Versal ACAP applications can exploit the capabilities of each of these resources. To create or migrate a design to a Versal ACAP, you must identify which resources best satisfy the different needs of the application and partition the application across those resources.

The following figure shows the layout of the Versal ACAP.

Figure 1: **Versal ACAP Layout**



The following sections provide a summary of the blocks that comprise the Versal architecture. For detailed information on these blocks, see the *Versal Architecture and Product Data Sheet: Overview* (DS950).

AI Engine

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. Given the AI Engine's advanced signal processing compute capability, it is well-suited for highly optimized wireless applications such as radio, 5G, backhaul, and other high-performance DSP applications.

AI Engines are an array of very-long instruction word (VLIW) processors with single instruction multiple data (SIMD) vector units that are highly optimized for compute-intensive applications, specifically digital signal processing (DSP), 5G wireless applications, and artificial intelligence (AI) technology such as machine learning (ML).

AI Engines provide multiple levels of parallelism including instruction-level and data-level parallelism. Instruction-level parallelism includes a scalar operation, up to two moves, two vector reads (loads), one vector write (store), and one vector instruction that can be executed—in total, a 7-way VLIW instruction per clock cycle. Data-level parallelism is achieved via vector-level operations where multiple sets of data can be operated on a per-clock-cycle basis. Each AI Engine contains both a vector and scalar processor, dedicated program memory, local 32 KB data memory, access to local memory in any of three neighboring directions. It also has access to DMA engines and AXI4 interconnect switches to communicate via streams to other AI Engines or to the programmable logic (PL) or the DMA. Refer to the *Versal ACAP AI Engine Architecture Manual* (AM009) for specific details on the AI Engine array and interfaces.

Programmable Logic

The Versal ACAP programmable logic (PL) comprises configurable logic blocks (CLBs), internal memory, and DSP engines. Every CLB contains 64 flip-flops and 32 look-up tables (LUTs). Half of the CLB LUTs can be configured as a 64-bit RAM, as a 32-bit shift register (SRL32), or as two 16-bit shift registers (SRL16). In addition to the LUTs and flip-flops, the CLB contains the following:

- Carry lookahead logic for implementing arithmetic functions or wide logic functions
- Dedicated, internal connections to create fast LUT cascades without external routing

This enables a flexible carry logic structure that allows a carry chain to start at any bit in the chain. In addition to the distributed RAM (64-bit each) capability in the CLB, there are dedicated blocks for optimally building memory arrays in the design:

- Accelerator RAM (4MB) (available in some Versal devices only)
- Block RAM (36 Kb each) where each port can be configured as 4Kx9, 2Kx18, 1Kx36, or 512x72 in simple dual-port mode
- UltraRAM (288 Kb each) where each port can be configured as 32Kx9, 16Kx18, 8Kx36, or 4Kx72

Versal devices also include many low-power DSP Engines, combining high speed with small size while retaining system design flexibility. The DSP engines can be configured in various modes to better match the application needs:

- 27×24-bit twos complement multiplier and a 58-bit accumulator
- Three element vector/INT8 dot product

- Complex 18bx18b multiplier
- Single precision floating point

For more information on PL resources, see the *Versal ACAP Configurable Logic Block Architecture Manual* (AM005), *Versal ACAP Memory Resources Architecture Manual* (AM007), and *Versal ACAP DSP Engine Architecture Manual* (AM004).

NoC

The NoC is a high-speed communication subsystem that transfers data between intellectual property (IP) Endpoints in the PL, PS, and other integrated blocks, providing unified intra-die connectivity. The NoC master and slave interfaces can be configured as AXI3, AXI4, or AXI4-Stream. The NoC converts these AXI interfaces to a 128-bit wide NoC packet protocol that moves data horizontally and vertically across the device via the HNoC and VNoC respectively. The HNoC runs at the bottom and top of the Versal ACAP, close to the I/O banks and integrated blocks (e.g., processors, memory controllers, PCIe interfaces). The number of VNoCs (up to 8) depends on the device and the amount of DDRMCs (up to 4 DDRMCs). For more information on the AXI protocol, see the *Vivado Design Suite: AXI Reference Guide* (UG1037).

The Versal ACAP NoC IP acts as the logical representation of the Versal ACAP NoC. The NoC main function is to efficiently move data between the DDR controllers and the rest of the device. The Versal ACAP NoC IP enables multiple masters to access a shared DDRMC with advanced quality of service (QoS) settings. The AXI NoC IP is required to connect the PS or the PL to the DDRMC. The AXI NoC IP can also be used to create additional connections between the PS and the PL or between design modules located in the PL.

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

XPIO

The XPIO in Versal ACAPs are similar to the high-speed I/O (HPIO) in the UltraScale™ architecture. However, the XPIO are located at the bottom and/or top periphery of the device, unlike the I/O columnar layout in previous devices. The XPIO provide XPHY logic that is similar to UltraScale device native mode. The XPHY logic encapsulates calibrated delays along with serialization and deserialization logic for 6 single-ended I/O ports known as nibble. Each XPIO bank contains 9 XPHY logic sites and supports up to 54 single-ended I/O ports. The XPHY logic is used for the integrated DDRMC, soft memory controllers, and custom high-performance I/O interfaces. For more information on the XPIO, see the *Versal ACAP SelectIO Resources Architecture Manual* (AM010).

DDRMC

The DDRMC is a high-efficiency, low-latency integrated DDR controller for a variety of applications, including general purpose central processing units (CPUs) as well as other traditional field programmable gate array (FPGA) applications, such as video or network buffering.

The controller operates at half the DRAM clock frequency and supports DDR4, LPDDR4, and LPDDR4X standards up to 4266 Mb/s. The controller can be configured as a single DDR interface with data widths of 16, 32, and 64 bits, plus an extra 8 check bits when error-correction code (ECC) is enabled. The controller can also be configured as 2 independent or interleaved DDR interfaces of 16 or 32 data bits each. The controller supports x4, x8, and x16 DDR4 and x32 LPDDR4 components, small outline dual in-line memory modules (SODIMMs), unbuffered DIMMs (UDIMMs), registered DIMMs (RDIMMs), and load-reduced DIMMs (LRDIMMs). The DDRMC is accessed through the NoC.

In Versal ACAP, the DDRMC is a system-wide, shared resource. It is shared between the PS and PL via the device-wide, high-performance NoC interface. The NoC IP core can be configured to include one or more integrated DDRMCs. If two or four DDRMCs are selected, the DDRMCs are grouped to form a single interleaved memory. In interleaved mode, the application views the participating DDRMCs as a single unified block of memory. The NoC supports interleaving across two or four DDRMCs by automatically dividing AXI requests into interleaved, block-sized subrequests and alternately sending the subrequests to each of the participating DDRMCs.

Note: You must use the NoC to connect between the PL, PS, CPM, or AI Engine and the DDRMC.

For more information on the DDRMC, see the *Network on Chip and Integrated Memory Controller v1.0 LogiCORE IP Product Guide* (PG313).

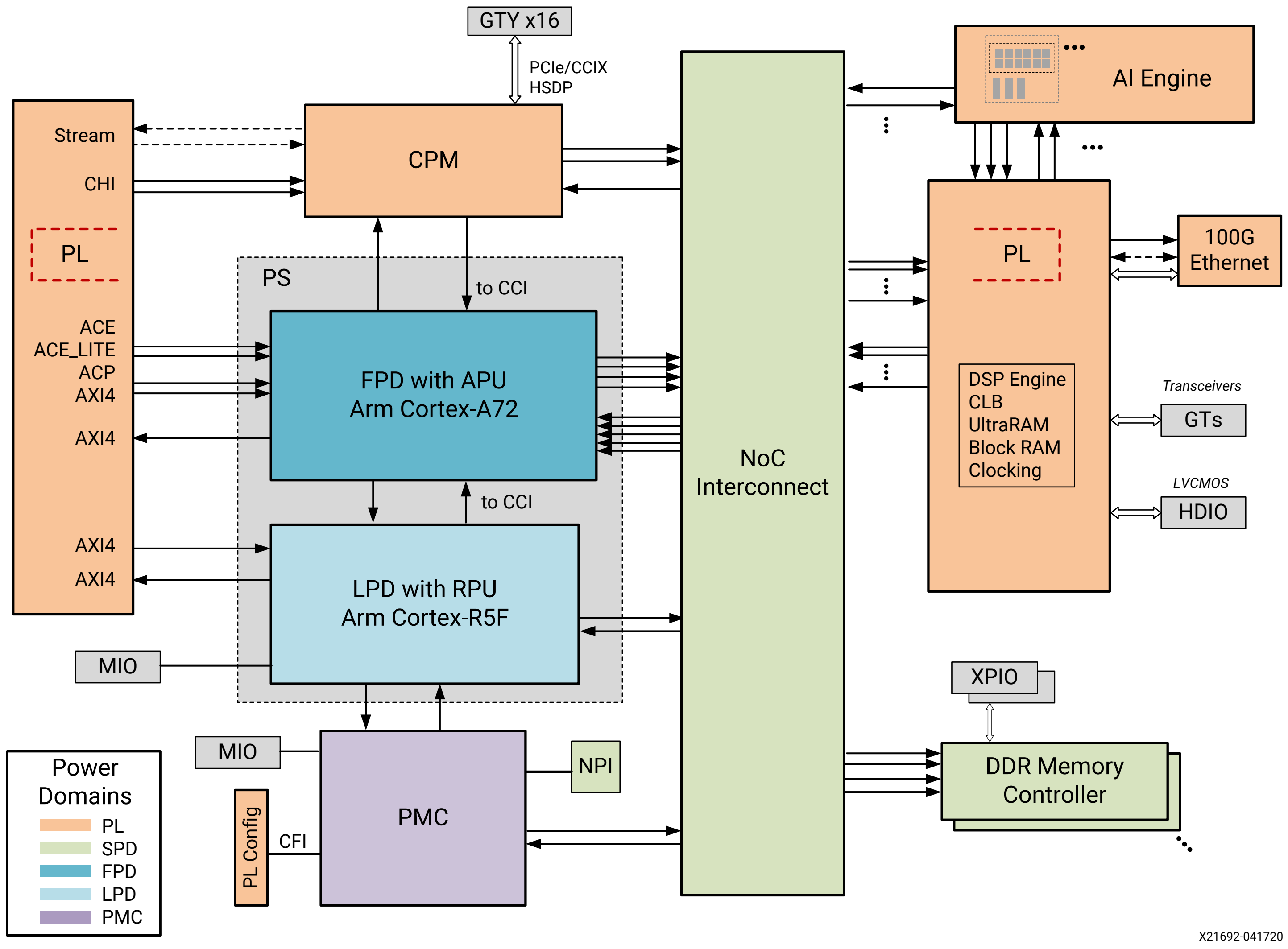
Note: Versal ACAP also supports soft memory controllers in the PL fabric, similar to previous device families.

CIPS

The PS, PMC, and CPM modules are grouped together and configured using the Control, Interface, and Processing System (CIPS) IP core as shown in the following figure.

Note: The Versal ACAP includes multiple power domains in the PS, PL and PMC. In the PS, the RPU is in the in the low-power domain (LPD), and the APU is in the full-power domain (FPD). There are two implementations of the CPM depending on the target device capability: CPM4 that is compliant with the PCIe Base Specification Revision 4.0 and CPM5 that is compliant with the PCIe Base Specification Revision 5.0. CPM4 is fully powered by the PL domain while CPM5 is powered by its own dedicated supply (VCCINT_CPM) as well as the PS LPD.

Figure 2: **System-level Interconnect Architecture**



PS

The PS contains the application processing unit (APU), real-time processing unit (RPU), and peripherals. The DDRMC is shared between the PS and PL via the device-wide, high-performance NoC interface.

APU

The Versal ACAP APU includes a dual-core Arm® Cortex™-A72 processor attached to a 1 MB unified L2 cache. The APU is designed for system control and compute-intensive applications that do not need real-time performance. The increased performance of Versal ACAP requires higher performance from the memory subsystem. To help meet these requirements, the Versal ACAP includes an increased L1 instruction cache size (32 KB to 48 KB) as well as multiple DDRMCs and the NoC, which improve the performance of the main memory.

The following table shows the difference between the Cortex-A53 in Zynq® UltraScale+™ MPSoCs and the Cortex-A72 processors in Versal ACAPs.

Table 1: **Cortex-A53 and Cortex-A72 Comparison**

Cortex-A53	Cortex-A72	Versal ACAP Benefits
Armv8A architecture (64-bit and 32-bit operations)		No application code changes required
EL0-EL3 exception levels		
Secure/non-secure operation		
Advanced SIMD NEON floating-point unit		
Integrated memory manager		
Power island control		
Up to 1500 MHz	Up to 1700 MHz	Higher frequency
2.23 DMIPS per MHz	5.74 DMIPS per MHz	2 times higher raw performance (per Arm benchmarks)
3.65 SPEC2006int per GHz	6.84 SPEC2006int per GHz	
2-way super scalar	3-way super scalar	More efficient instruction cycle
In-order execution	Out-of-order execution	Higher performance and fewer memory stalls
Power efficient	Improved power efficiency	20% lower power
8-stage pipeline	15-stage pipeline	More instructions queued and executed
Conditional branch prediction	Two-level branch prediction	Higher cache hits and less memory fetches

RPU

The Versal ACAP RPU Arm Cortex-R5F processor has faster clocking frequencies than the Zynq UltraScale+ MPSoC. The Versal Arm Cortex-R5F processor supports Vector Floating-Point v3 (VFPv3) whereas the Zynq UltraScale+ MPSoC Arm Cortex-R5F processor supports VFPv2.

Standard Peripherals

Versal ACAP standard I/O peripherals are located in the low-power domain (LPD) and in the PMC. The NoC must be configured to provide access to the DDRMC so that the peripherals with direct memory access (DMA) can access the DDR interfaces.

The following table shows the difference between the standard peripherals in Zynq UltraScale+ MPSoCs and Versal ACAPs.

Table 2: **Standard Peripherals Comparison**

Peripheral	Zynq UltraScale+ MPSoC	Versal ACAP
CAN, CAN-FD	2 controllers with standard CAN	2 controllers with controller area network - flexible data rates (CAN-FD)
GEM	4 controllers	2 controllers with time-sensitive networking (TSN) feature
GPIO	1 controller	2 controllers
I2C	2 controllers	2 controllers in LPD (general purpose) 1 controller in PMC (general purpose)
NAND	1 controller	N/A
PCIe (Gen1, Gen2)	1 controller	N/A
PCIe (Gen3, Gen4)	1 controller	Varies by device
SPI	2 controllers	2 controllers
SATA	1 controller	N/A
UART	2 controllers with standard UART	2 controllers with Server Base System Architecture (SBSA)
USB 2.0, 3.0 (host, device, on-the-go)	2 controllers	N/A
USB 2.0 (host, device, dual-role device)	N/A	1 controller

AMBA Specification Interfaces

The PS-PL Arm Advanced Microcontroller Bus Architecture (AMBA) specification interfaces in the Versal ACAP have similar functionality to Zynq UltraScale+ MPSoCs, as shown in the following table.

***Note*:** Enabling and disabling the different power domains in the LPD, FPD, and PL enables and disables the AXI connections to those domains.

IMPORTANT! *Because the DDRMC is shared between the PS and PL via the device-wide, high-performance NoC interface, there are fewer PS-PL AXI interconnects.*

Table 3: **AMBA Interface Comparison**

PS-PL AMBA Interface	Master	Coherency	Zynq UltraScale+ MPSoC		Versal ACAP	
			Name	Count	Name	Count
Accelerator Coherency Port (ACP)	PL	I/O	S_AXI_ACP_FPD	1	S_ACP_FPD	1
AXI Coherency Extensions (ACE)	PL	2-way	S_AXI_ACE_FPD	1	S_ACE_FPD	1
PL-to-FPD AXI	PL	-	S_AXI_HPx_FPD	4	S_AXI_HP	1
PL-to-FPD AXI	PL	I/O	S_AXI_HPCx_FPD	2	S_AXI_HPC	1

Table 3: **AMBA Interface Comparison** *(cont'd)*

PS-PL AMBA Interface	Master	Coherency	Zynq UltraScale+ MPSoC		Versal ACAP	
			Name	Count	Name	Count
PL-to-LPD AXI	PL	-	S_AXI_LPD	1	S_AXI_LPD	1
FPD-to-PL AXI	FPD	-	M_AXI_HPMx_FPD	2	M_AXI_FPD	1
LPD-to-PL AXI	LPD	-	M_AXI_HPM0_LPD	1	M_AXI_LPD	1

PMC

The PMC subsystem includes the following functions:

- Boot and configuration management
- Dynamic Function eXchange (DFX)
- Power management
- Reliability and safety functions
- Life-cycle management, including device integrity, debug, and system monitoring
- I/O peripherals, including PMC I2C and GPIO

The PMC block executes the BootROM and platform loader and manager (PLM) to handle the boot and configuration for the PS, CPM, PL, NoC register initialization and settings, and I/O and interrupt configuration settings. In addition to boot and configuration, the PLM provides life-cycle management services. The PMC bus architecture and centralized integration enables significantly faster configuration and readback performance when compared with previous devices. The following table shows the Zynq UltraScale+ MPSoC blocks that are comparable to the Versal ACAP blocks.

Table 4: **Block Comparison**

Zynq UltraScale+ MPSoC	Versal ACAP
Configuration security unit (CSU) and platform management unit (PMU)	PMC
CSU	ROM code unit (RCU)
PMU	Platform processing unit (PPU)
First stage boot loader (FSBL) and PMU firmware	PLM

For more information on the PMC, see the *Versal ACAP Technical Reference Manual* (AM011). For more information on the PLM, see the *Versal ACAP System Software Developers Guide* (UG1304).

Flash Memory Controllers

The PMC includes three types of flash memory controllers, any of which can be used as a boot device or by the application. The following table shows the difference between the flash memory controllers in Zynq UltraScale+ MPSoCs and Versal ACAPs.

Table 5: **Flash Memory Controllers Comparison**

Peripheral	Zynq UltraScale+ MPSoC	Versal ACAP
Octal SPI (OSPI)	N/A	1 controller
Quad SPI (QSPI)	1 controller	1 controller that does not support linear address mode
SD/eMMC	2 controllers	2 controllers with the same functionality and updated DLL

Note: Versal ACAPs can also support secondary boot modes (e.g., Ethernet, USB, etc.). For more information, see the *Versal ACAP System Software Developers Guide* (UG1304).

CPM

The Versal architecture includes several blocks for implementation of high performance, standards-based interfaces built on PCI™-SIG technologies. In Versal ACAPs that contain a CPM, the CPM provides the primary interfaces for designs following the server system methodology. As part of the Versal architecture integrated shell, the CPM has dedicated connections to the NoC over which it can access DDR and other hardened IP. The CPM is configured separately from the programmable logic, which enables the integrated shell to become operational quickly after boot without the need to configure the PL. This separate configuration addresses a common power-up and reset timing challenge imposed by the PCIe specification. Two implementations of the CPM exist: CPM4 and CPM5.

In Versal ACAPs with an available CPM4, the block is compliant with the PCIe Base Specification Revision 4.0 and capable of supporting defined line rates up to the maximum of 16 GT/s. CPM4 contains two PCIe controllers with shared access to 16 GTY transceivers, and integrates a single direct memory access (DMA) controller functionality (either QDMA or XDMA that is user selectable) associated with CPM PCIe Controller #0. Cache Coherent Interconnect for Accelerators (CCIX) support in CPM4 complies with CCIX Base Specification Revision 1.0.

In Versal ACAPs with an available CPM5, the block is compliant with the PCIe Base Specification Revision 5.0 and capable of supporting defined line rates up to the maximum of 32 GT/s. CPM5 contains two PCIe controllers with dedicated access to 16 GTYP transceivers. CPM5 integrates two DMA controllers (both QDMA) each associated with CPM PCIe Controller #0 and CPM PCIe Controller #1. CCIX support in CPM5 complies with CCIX Base Specification Revision 1.1.

CPM4 and CPM5 include the following additional components:

- The coherent mesh network (CMN) forms the CCIX block, which is based on the Arm CoreLink CMN-600.
- There are two Coherent Hub Interface (CHI) PL interface (CPI) blocks. CPM4 has one L2 cache instance, and CPM5 has two L2 cache instances. CPI blocks interface with the accelerators in the PL and perform 512-to-256 bit data width conversion and clock domain crossing into the internal core clock.
- The non-coherent interconnect block, which interfaces with the PS for access to the NoC and DDRMC. The interconnect is connected to all of the other sub-blocks via an advanced peripheral bus (APB) or AXI slave interface for configuration.
- A clock/reset block, which includes a phase-locked loop (PLL) and clock dividers.

CPM availability is device specific. For information, see the *Versal Architecture and Product Data Sheet: Overview* (DS950). For more information on CPM, see the *Versal ACAP CPM CCIX Architecture Manual* (AM016) and *Versal ACAP CPM Mode for PCI Express Product Guide* (PG346).

***Note*:** Versal ACAP also supports implementation of subsystems based on PCI-SIG technologies in the PL fabric, similar to previous device families.

GT

GTs provide several protocols for high-speed interfaces, such as Ethernet and Aurora IP. Versal ACAP features the XPIPE mechanism to connect the PCIe block to the GT at high speed. XPIPE and GTs are shared between PL-based IP and PS-based IP (e.g., CPM, Ethernet, Aurora link for debug, etc.). For Versal ACAP, GT components are updated from Common/Channel to a quad granularity. For more information on the GT, see the *Versal ACAP GTY Transceivers Architecture Manual* (AM002).

HSDP

The heterogeneous nature of the Versal ACAP necessitates a system-level high-bandwidth debug and trace solution. The high-speed debug port (HSDP) is a new feature in Versal ACAP that provides unified, at-speed debugging and tracing of the various integrated, fabric-based, and processor blocks in the device under test (DUT). HSDP functions are accessed via high-speed GT-based interfaces, such as the integrated Aurora interface in the PS block.

Chapter 3

System Methodology

System methodology consists of understanding all of the system requirements based on the target application. This includes identifying the appropriate Versal™ device with the correct features (e.g., the number of DDRMC IP, AI Engines, etc.). You must also consider power and thermal requirements. With the appropriate device selection, the next steps are system design, including hardware/software co-design of the target application on the device, system simulation, and bring-up and debug.

The following sections walk through the various steps on the hardware/software co-design and subsequent steps for simulation and validation of systems designed with Versal ACAP.

System Design Methodology

Versal ACAP is a heterogeneous compute platform with multiple compute engines introduced in the previous chapter. A wide range of applications can be mapped on Versal ACAP, predominantly in the areas of signal processing, wireless, machine learning inference, and video processing algorithms. Versal ACAP offers very high system bandwidth using high-speed serial I/Os, NoC, DDR4/LPDDR4 memory controllers, and multi-rate Ethernet Media Access Controllers (MRMACs).

The system applications use mode of Versal ACAP falls in the following key categories:

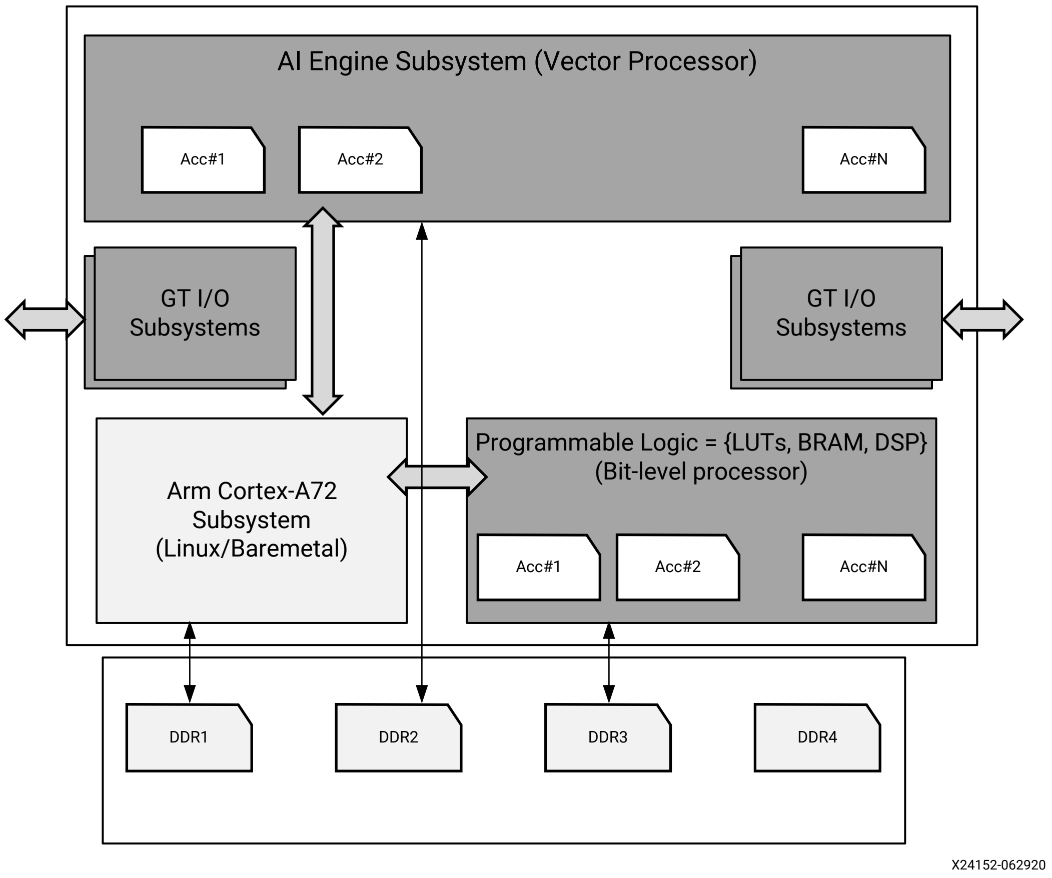
- **Embedded system:** embedded processing system with compute acceleration
- **Server system:** data center host attached compute acceleration

Embedded System Methodology

An embedded system comprises the embedded processor in Versal ACAP and the acceleration logic built into two key categories of acceleration components, the traditional PL (LUTs, BRAMs, URAMs, DSPs) and the AI Engines. For Versal ACAP, the embedded compute system comprises the Arm® Cortex™-A72 and Cortex-R5F processors. The use models in this system category can range from a sophisticated embedded software stack to a simple bare-metal stack only required to support programming of the acceleration units.

An embedded system runs a software stack executed on the built-in embedded processor that serves as an overall control plane for the kernels running on the acceleration units. Data transfer is managed between the server and Versal ACAP by the Xilinx Runtime (XRT) application programming interfaces (APIs). These APIs also have function calls for managing the acceleration units.

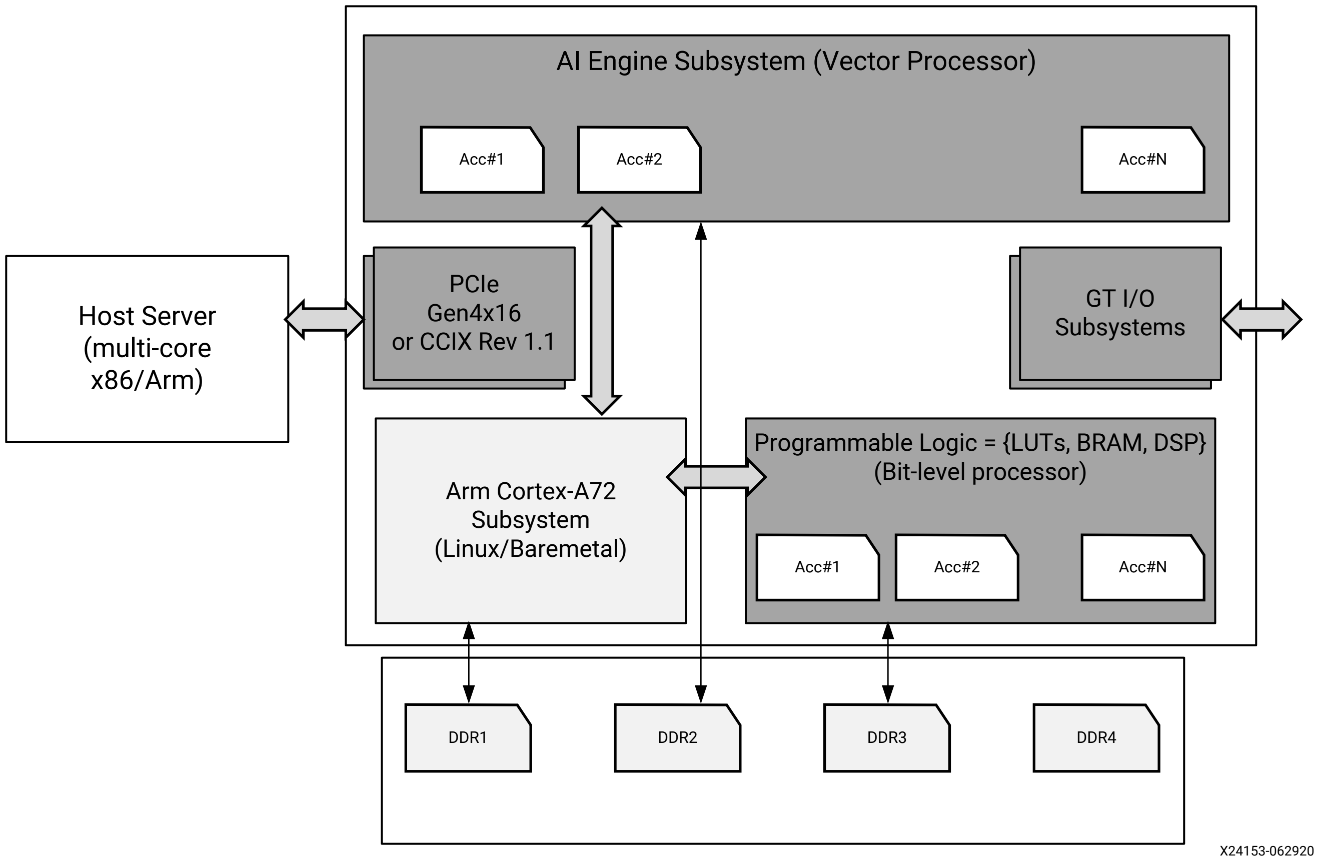
Figure 3: **Embedded System**


AI Engine Subsystem (Vector Processor)
Acc#1
Acc#2
Acc#N
GT I/O Subsystems
GT I/O Subsystems
Arm Cortex-A72 Subsystem (Linux/Baremetal)
Programmable Logic = {LUTs, BRAM, DSP} (Bit-level processor)
Acc#1
Acc#2
Acc#N
DDR1
DDR2
DDR3
DDR4
X24152-062920

Server System Methodology

A server system comprises a server class central processing unit (CPU) system (e.g., x86 host, Arm processor, etc.) to which the Versal ACAP device is attached via PCIe® or a cache coherent interface, such as CCIX. The application acceleration is implemented on the two key categories of acceleration components, traditional PL (LUTs, BRAMs, URAMs, DSPs) and the AI Engines. PCIe-based Versal ACAP platforms are managed by XRT open-source software. XRT provides tools and APIs for data transfer between server device, acceleration image download (Dynamic Function eXchange (DFX)), acceleration kernel management, and board management.

Figure 4: **Server System**


AI Engine Subsystem (Vector Processor)
Acc#1
Acc#2
Acc#N
PCIe Gen4x16 or CCIX Rev 1.1
GT I/O Subsystems
Host Server (multi-core x86/Arm)
Arm Cortex-A72 Subsystem (Linux/Baremetal)
Programmable Logic = {LUTs, BRAM, DSP} (Bit-level processor)
Acc#1
Acc#2
Acc#N
DDR1
DDR2
DDR3
DDR4
X24153-062920

Embedded and Server System Design Considerations

Following are the key steps in the development flow for embedded and server system designs. Each step has its unique challenges depending on whether a system is embedded or server attached.

1. Hardware and software compute acceleration development
    a. Accelerator hardware/software co-design
        i. Architecture
        ii. Datapath and transport layer
        iii. Control plane
        iv. Memory hierarchy
        v. Kernel design and verification
    b. Hardware platform design

   c. Design verification
      i. Hardware and software co-simulation
      ii. Performance validation
   d. Timing closure
   e. Hardware validation
      i. Hardware and software bring-up and validation
      ii. System debug
      iii. Power and performance validation
2. Software development for effective use of hardware acceleration
   a. Boot and OS considerations
   b. Software application development
   c. Software debug

Hardware and Software Compute Acceleration Development

Accelerator Hardware and Software Co-Design

The following accelerator design steps are common to both embedded and server systems:

1. Architecture
2. Datapath and transport layer
3. Control plane
4. Memory hierarchy
5. Kernel design and verification

Architecture

The primary challenge to resolve as part of system design architecture is power and performance optimization. Your choice of acceleration hardware, whether PL or AI Engines, depends on the type of algorithm and data ingress and egress paths. In the case of streaming data ingress and egress to and from sensors (e.g., LiDAR, RADAR, dual-camera vision systems), data is available to fabric through high-speed transceivers. This data is aggregated from external protocol interfaces on AXI4-Stream buses and can be distributed to the PL or AI Engines.

The Scalar Engines (processor subsystem), Adaptable Engines (programmable logic), and Intelligent Engines (AI Engines) form a tightly-integrated, heterogeneous compute platform. Scalar Engines provide complex software support. Adaptable Engines provide flexible custom compute and data movement. Given their high compute density, AI Engines are well suited for vector-based algorithms.

During this step, you develop a mapping of the core application and of each algorithm to the most appropriate architectural area (e.g., AI Engine, PS, PL, NoC, DDRMC) in the Versal ACAP. This consists of mapping all of the major blocks in the application and considering requirements on these major blocks in terms of bandwidth and availability. This application mapping and design partition step is manual.

You must consider which architecture is best for which task as follows:

- Scalar processing elements like CPUs are very efficient at complex algorithms with diverse decision trees and a broad set of libraries. However, these elements are limited in performance scaling. Application control code is well suited to run on the scalar processing elements.
- Programmable logic can be precisely customized to a particular compute function, which makes them best at latency-critical real-time applications (e.g., automotive driver assist) and irregular data structures (e.g., genomic sequencing). However, algorithmic changes have traditionally taken hours to compile versus minutes.
- The AI Engine processors deliver more compute capacity per silicon area versus PL implementation of compute-intensive applications. AI Engines also reduce compute-intensive power consumption by 50% versus the same functions implemented in PL and also provide deterministic, high-performance, real-time DSP capabilities. Because the AI Engine kernels can be written in C/C++, this approach also delivers greater designer productivity. Signal processing and compute-intensive algorithms are well suited to run on the AI Engines.

You can implement additional features, such as clock gating, for regions of the PL and AI Engine that are not used concurrently. You can handle traditional multi-clock domain fabric design and datapath clock domain crossing using the same approach that is used with FPGA architectures.

Datapath and Transport Layer

In an embedded design, the primary purpose of the datapath is to capture the dataflow of interest in the system application. For vision processing systems, the dataflow might include incoming data from image sensors (e.g., camera, LiDAR, etc.) that is stored in on-chip and off-chip system memory (e.g., BRAM, URAM, DRAM) to be processed inline. For networking systems, the dataflow might be a protocol bridge between two standards like Ethernet and Interlaken.

In a server system, the datapath defines the primary dataflow between the server processor system and the Versal ACAP. There are two key datapath configurations that are available for the server to attach to the Versal ACAP, PCI Express® or cache coherent interface. The PCI Express datapath allows standard PCI Express I/O transactions between the server and Versal ACAP. The datapath also supports more complex DMA transactions for the transfer of larger blocks of information. The Versal ACAP CPM also includes support for cache coherent interconnect using a protocol overlaid on the PCIe transport layer and enables a cache coherent view of the server system memory for all of the accelerators running on Versal ACAP.

The PL accelerators datapath (ingress and egress) is mapped to AXI4-Stream interfaces and control interfaces, which are mapped to memory-mapped AXI interconnect. The control interface enables these accelerators to be controlled by the software stack.

Similarly, the AI Engine accelerators datapath is mapped to the AXI4-Stream interfaces. The AI Engine kernels have a run-time parameter (RTP) interface that enables run-time updates. You can access the RTP feature through the Xilinx run-time (XRT) APIs.

Control Plane

The control path orchestrates the system control functions, such as accelerator initialization, data transfer initialization, and acceleration execution. Xilinx recommended using the XRT APIs for control path design for both embedded and server systems. For more information, see the *XRT Release Notes* (UG1451).

Memory Hierarchy

For data-intensive applications that demand high-bandwidth memory, the best approach is to construct a domain-specific memory hierarchy, such as memory caching from DDR to PL RAMs (URAM/BRAM). Versal devices can access the DDR memory through the NOCs. The DDR is connected to the device fabric, and the Arm Cortex-A72 and the AI Engines are connected through the NoC. Using DMA data movers in the PL, you can coordinate data movement to and from the hard IP to the DDR, leveraging the intermediate PL RAM stages for caching or data buffering.

Versal devices provide access to DDR memories using DMAs. The DDR memories are connected to the device fabric and other hard IP through the hard memory controller and NoC. You can use DMA data movers in the PL to coordinate data movement to and from the DDR. You can also configure the NoC for maximum bandwidth using the DDR controllers available on the Versal device.

Following are additional recommendations:

- Use the NoC compiler to find the optimal solution for a required aggregate bandwidth.
- For data-intensive applications that demand high-bandwidth memory, construct a domain-specific memory hierarchy, such as memory caching from DDR to PL RAMs (URAM/BRAM).

Kernel Design and Verification

The accelerator kernels are the executable functional block implemented on the Versal ACAP platform. There are two key types of kernels, AI Engine kernels and PL kernels:

- AI Engine kernels can be designed and simulated in the Vitis™ environment.
- PL kernels designed with register transfer level (RTL) languages can be individually simulated in the Vivado® tools using AXI Verification IP (VIP).
- PL kernels designed in C/C++/OpenCL™ can be simulated using the Vitis HLS tool.

Hardware Platform Design

After the design architecture is complete, the next step is to create a Vitis platform that comprises several components, including CIPS, NoC, DDR, GTs, AI Engine, etc. Any number of subsequent designs can leverage a single platform. For more information, see the Vitis Unified Software Development Platform Documentation.

Design Verification

For information, see System Simulation Methodology.

Timing Closure

All typical timing closure practices that apply to the use of PL resources also apply to the PL in the Versal ACAP. In addition, with the introduction of the hard AI Engine system that is physically located above the fabric, the AI Engine compiler provides the directive for AI Engine-PL interface placement.

Following are general guidelines to improve system-level timing closure issues:

- Leverage RTL and architecture design guidelines to achieve better design performance.
- Use AXI register slices to pipeline the AXI4-Stream datapath and assist the Vivado placer in spreading the logic in the fabric for better timing closure. The AXI register slice can also be used to help timing closure on the AI Engine-PL interface.

Software Development

Boot and OS

Versal devices have a centralized PMC that boots the device after power on reset. Versal devices support different boot modes, including JTAG, SD, eMMC, OSPI, QSPI, SelectMAP, and PCIe boot modes. Depending on the boot time requirement, you must select the appropriate boot device.

- For PCIe-based applications where there is a requirement to detect an Endpoint in less than 100 ms of system power on, use a faster boot device like OSPI. The image can be partitioned so that a minimal boot image boots from the OSPI boot mode in less than 100 ms, and the larger boot partition can be transferred over PCIe in Tandem PCIe boot mode.
- Use SD and eMMC boot devices for general embedded applications. Use eMMC boot mode for embedded applications that require higher density.
- Use JTAG boot mode for initial system bring-up and for debug. To enable system debug, switch to JTAG boot mode by configuring the boot mode register from the TAP chain.

Applications that require device-level security need to implement boot image encryption and authentication supported by the Versal ACAP hardware. With encryption and authentication enabled, there is a corresponding increase in the system boot time.

Versal ACAP boot devices support partition fallback to avoid catastrophic boot failure. During a field upgrade, if the upgraded image has an error, the PLM can fall back to a golden image to recover the boot mode. The golden image can reside in the same boot device as other upgrade images.

Select the OS depending on your application-specific use case. For applications requiring buffer management, locked access, and interrupt handling with multiple processes running in parallel, use the Linux OS. These applications can leverage the open source Linux framework that provides higher level abstraction. Typical applications include video, OpenCL, OpenCV, and networking stacks that can leverage the Linux framework. Xilinx provides Linux run time support using the Xilinx run time (XRT) stack that handles interrupt management, starting and stopping of kernels, buffer allocation, and sharing. XRT can interface with higher-level software stacks like OpenCL, OpenCV, FFmpeg, and Python-based frameworks. One downside of using the Linux-based stack is the stack overhead, which is not suitable for real-time operations. Also, the Linux OS requires FPD power to be on. Applications requiring significant power saving can use Arm Cortex-R5F processor that works in LPD.

Applications that require real-time processing can leverage the Arm Cortex-R5F processors in Versal ACAP. Arm Cortex-R5F processors are ASIL-C safety compliant. Typical application mapping involves system monitoring, hardware monitoring, direct hardware control with light weight stack, and so forth. If the Arm Cortex-R5F processor is targeted for a functional safety application, Xilinx recommends fitting the application code in the TCM instead of accessing the code from DDR memory. The Arm Cortex-R5F processor can also work as a co-processor to the Linux OS, monitoring specific hardware functionality and providing hardware status to the Linux application. The communication between the Linux OS and RTOS/bare-metal OS on the Arm Cortex-R5F can happen via the inter-processor interrupt.

Versal devices support virtualization. You can use the same hardware for multiple guest OS using a hypervisor. There is a longer interrupt processing overhead with virtualized hardware.

For applications that require low latency processing, use dedicated hardware.

Software Application Development

To manage PL accelerators written in C/C++/OpenCL (HLS) or RTL, as well as AI Engine kernels, Xilinx recommends using the XRT APIs. Any IP that is in the platform must be explicitly managed by the designer. Any accelerator (PL or AI Engine) linked to the platform using the Vitis linker (`v++ --link`) is best managed by the XRT APIs. Tailor the design architecture to allow the application to reset the user-defined PL IP to a known good state to be able to handle errors. You can develop other high-level software applications using OpenCL, similar to previous 16 nm FPGA SDAccel™ flows. The Vitis tools provide a complete software development environment.

Software Debug

For information, see System Debug Methodology.

System Simulation Methodology

The complexity of Versal ACAPs with its different compute domains challenges traditional FPGA simulation methods. With traditional FPGA simulation, most of the design can be verified using logic simulation. With Versal ACAP, the programmable logic is only one of the compute domains, and the simulation methodology must consider the software domain as well as the AI Engine domain when used.

The system simulation methodology for Versal ACAP is based on a hierarchical approach. This methodology acknowledges the need to simulate each compute domain independently, while also being able to simulate the entire system when appropriate.

The system simulation methodology is built around the following key concepts:

- **Scope of the simulation:** The simulation can include the entire system or just portions of the system. Xilinx recommends testing blocks and functions individually before integrating and simulating them in the entire system. You can use different simulation flows to test the different compute domains, including the PS, PL, and AI Engine.
- **Abstraction of the simulation:** In some cases, you can simulate specific functions at different abstraction levels. This is true of both AI Engine and HLS code, which you can simulate either as untimed or as cycle-accurate models. This is also the case for specific Versal ACAP infrastructure blocks, such as the NoC or DDR controllers, which you can simulate as SystemC transaction-level models (TLM) or RTL models. Abstraction allows you to trade simulation speed for simulation accuracy.
- **Purpose of the simulation:** The purpose of each simulation can vary. For example, is the focus on functional validation or performance measurement? Is the intention to test a single function or the interactions between multiple functions? Different simulation purposes rely on different simulation setups and configurations. Purpose is closely related to scope and abstraction.

Related Information
Simulation Flows

Simulation Recommendations

Following are Versal ACAP simulation recommendations:

- Choose the appropriate simulation flow and abstraction based on the scope and intended purpose.
- Simulate and verify each component individually before integrating them and running hardware emulation, including the AI Engine graph, HLS kernels, RTL blocks, hardware platform, and PS code.

- Reuse test benches and test vectors whenever possible when testing different blocks and functions. For example, if the output of one block is the input to another block, reusing test vectors to simulate these two blocks eases the integration process.
- Perform gradual system integration. You do not need to run hardware emulation on the entire system. Running hardware emulation with a subset of the PS, PL, and AI Engine components can establish a known foundation, and you can gradually add functionality.
- Simulate and verify every design change. The earlier an issue is caught, the easier it is to address.

System Debug Methodology

The Versal ACAP includes an HSDP feature that enables enhanced system debug methodology capabilities. This feature is designed to work in any environment, including the lab, data center, and edge computing environments.

The HSDP feature consists of a centralized debug packet controller (DPC), which is the packet processing engine of the HSDP. The packets that are processed by the DPC are referred to as the debug and trace packets (DTP). These packets are decoded by the DPC to determine the commands, the destinations, and any potential higher-level flow control and management tasks. The DPC processes the DTP sent by a host, executes any commands embedded in the packets, and generates responses that are sent back to the host.

The HSDP DPC can be accessed from any of the following interfaces:

- JTAG interface
- Integrated Aurora via GTs
- PL fabric (using soft Aurora, PCIe, or any other suitable interface)
- PCIe interface

The HSDP feature enables debug of the use cases described in the following sections.

Note: The various APUs and RPUs of the PS can be debugged via the Arm CoreSight™ infrastructure that is integrated into the PS. The CoreSight infrastructure is accessible via the JTAG-DAP and HSDP interfaces.

Debug via JTAG

The PMC includes a JTAG interface that can be used for both programming and debugging designs running on the Versal ACAP. The JTAG interface consists of two cascaded blocks: the debug access port (DAP) and test access port (TAP). The DAP is primarily used to access the various debug features of the PS and can also be used for low-bandwidth read/write access to any accessible register and memory location within the address range of the PS. The TAP interface is primarily used for accessing the device configuration and boundary scan infrastructure, and also includes instructions for configuring and accessing the HSDP DPC functionality.

Debug via Aurora

The PS includes an integrated Aurora 64B/66B block that is dedicated for use in accessing the HSDP function via a high-speed GT-based interface. The Aurora interface to the HSDP DPC provides bidirectional access to the device from an external host debug/trace cable, allowing for high-speed debug and trace operations.

***Note*:** The integrated Aurora interface is not available in all Versal ACAPs.

Debug via PL Fabric

The HSDP DPC can also be accessed from the PL fabric. This allows you to integrate HSDP into your systems using methods other than the dedicated CPM/PCIe pathway, integrated Aurora, and JTAG.

Debug via PCIe

In data center applications and other PCIe interface-hosted systems, the HSDP DPC can be accessed via the same PCIe interface in the CPM that is used for host-to-ACAP communication. You can choose how to map the HSDP function to a PCIe physical function and BAR space. The HSDP function includes a dedicated HSDP DMA engine that is used to move debug data between the HSDP and host memory.

***Note*:** CPM availability is device specific. For information, see the *Versal Architecture and Product Data Sheet: Overview* (DS950).

Debugging the AI Engine

Every AI Engine in the AI Engine array has a debug interface, which can be used to read/write to all the individual AI Engine registers. The following files can be read and written over this interface: scalar register files, vector register files, status registers (for overflow, saturation, etc.), special registers like stack pointer, and zero overhead loop registers.

Requests for reading and writing AI Engine registers are sent via the AXI4 memory-mapped (AXI4-MM) interface and are then forwarded to the AI Engine Debug Interface. All the registers in the AI Engine are mapped on the AXI4-MM. The AXI4-MM interface has a 32-bit read/write bus. You can specify any AXI4-MM mapped address to read over the AXI4-MM interface. Any external AXI4-MM master (e.g., PS) can issue a stall signal to a specific AI Engine by writing into the control/status register. There are independent registers for system control (e.g., normal program flow) and the debugger. The Vitis System Debugger provides a comprehensive source code debugger that helps debug AI Engine graphs and kernels.

Events are another important feature of AI Engine debug and performance analysis. Events are similar to triggers. An event signal is high in the cycle for which the condition associated with that event is true. Examples of events include Conflict DM bank 0, Lock 11 Released, Floating point Overflow, and PC event 0. Each event has a unique 7-bit number, and there are up to 128 events in each AI Engine. Following are additional details:

- **Event Actions:** can be configured to perform a task whenever a specific event occurs. Examples of event actions include Debug halt core, Single step core, and Increment Performance Counter.
- **Event Broadcast:** can be used to send events signals to a neighboring AI Engine and the PL.
- **Event Trace Unit:** can collect cycle-by-cycle event activity from 8 numbered event signals and send compressed trace information from the AI Engine array via the AXI4-Stream network. The Vitis environment allows you to capture specific events from an AI Engine that can be analyzed in the Vitis Analyzer tool.

Debugging the PS

The DAP supports Arm CoreSight debug and trace of the PS. This includes both secure and non-secure debug support. The DAP also interfaces to external Arm debug tools via JTAG, according to Arm debug interface version 5 (ADIv5). TAP has access via JTAG Boundary Scan to the ERROR_STATUS register containing various error and alarm status bits. The Vitis System Debugger provides an integrated design environment (IDE) for PS debug with Baremetal and Linux-based applications.

Design Flow

Xilinx recommends using the Vitis™ and Vivado® tools to create and migrate designs targeted to Versal™ ACAPs. The design flows are similar to the flows used when targeting previous architectures.

The Versal ACAP Vitis environment supports a variety of applications that target the following:

- Server systems with the x86 host communicating through PCIe® blocks with a Xilinx® acceleration platform and with Vitis kernels in the PL and in the AI Engines
- Embedded systems with the PS (Arm® processor-based) host communicating with a Vivado Design Suite PL design packaged as a hardware platform and with Vitis kernels in the PL and in the AI Engines
- Embedded systems with the PS (Arm processor-based) host communicating with a Vivado Design Suite PL design packaged as a hardware platform

The Versal ACAP Vivado tools support creation of hardware platforms and designs either using the Vivado IP integrator or register transfer level (RTL).

Note: Depending on your design flow, you run either the Vitis or Vivado tools. If you are running the Vitis tools, the Vivado tools are run automatically at certain points during the flow.

Vitis Environment Design Flow

Versal ACAP designs are enabled by the Vitis tools, libraries, and IP. The Vitis environment lets you program, run, and debug the different elements of a Versal ACAP AI Engine application, which can include AI Engine kernels and graphs, PL, high-level synthesis (HLS) IP, RTL IP, and PS applications. Each domain has its own set of tools and methodologies. For more information, see the Vitis Unified Software Development Platform Documentation.

The Vitis environment includes AI Engine tools for programming, debugging, and deploying graph algorithms, including the aiecompiler, SystemC simulator (aiesimulator), and x86 simulator (x86simulator). The Vitis compiler (`v++ --compile`) allows integration of kernels to the graph running in the PL region of the device or running alongside the graph to define additional subsystems. The Vitis embedded software development flow (with the system software stack including PetaLinux) provides support for the PS domain of the embedded processor. The Vitis environment facilitates the creation and integration of subsystems for each of these domains, providing standardized interface requirements and data handoff between the different domains.

The Vitis tools take a platform-based approach, separating the essential services provided by the platform from the user-specific features of the application provided through the subsystems.

Platforms

Platforms come in two halves, the hardware platform and the software platform. The hardware platform includes the PS, NoC, DDR controllers, primary I/Os, AI Engine array, and any other user-specified IP blocks. The software platform defines the domains, device tree, and OS.

The platform insulates application developers from the details of low-level infrastructure and lets them focus on development of a specific subsystem function, such as software, AI Engine graph, or PL kernel logic. It is common for application developers to start their work by targeting a standard Xilinx platform before transitioning to a custom platform developed for a specific board and application. Custom platforms are developed using the Vivado tools.

Subsystems

Subsystems perform well-defined functions within the application. Subsystems are designed, debugged, and eventually integrated with other subsystems to form the top-level application. Using this approach, a complete Versal ACAP system is built using a collection of subsystems on a platform. This approach is similar to designing large FPGA designs.

A subsystem can include PS firmware, AI Engine graphs, and PL kernels. The subsystem is a standalone functional entity, performing well-defined functions under the supervision and coordination of the PS or PL. The subsystem always includes controlling software that configures the system as well as orchestrates the execution of subsystems in the AI Engine and PL fabric. A subsystem can interact with other subsystems via shared memory and streams.

The PL and AI Engine components of a subsystem are assembled using the Vitis compiler and linker (`v++ --compile` and `v++ --link`), and the PS firmware is integrated with the Vitis packager (`v++ --package`).

***Note*:** Currently, the AI Engine domain can only be part of a single subsystem.

Developing independent subsystems allows the concurrent development of multiple subsystems and integration into the platform. Custom platform development can also occur at the same time as application development, allowing simultaneous development of the custom application and the custom platform to deploy the application. The top-level system project comprises multiple subsystems, whether delivered by one team working on different elements at different times or by multiple teams working on multiple subsystems to build the system.

Subsystem Methodology

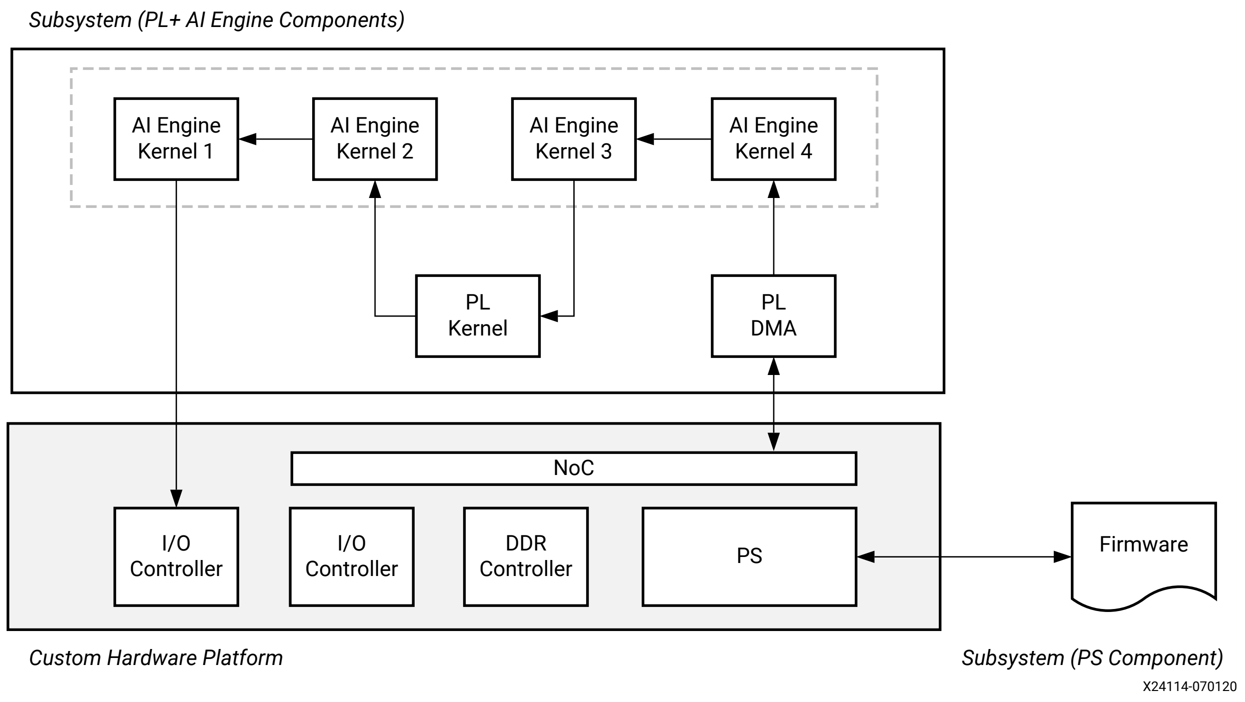
Prior to starting development, you must choose the Versal device that is best suited for your application and partition the design into functions targeted to the PS, AI Engine, and PL, depending on the application requirements. At this point, you must have an understanding of the following:

- System design considerations, such as throughput and latency
- Domain and inter-domain capabilities, including compute and bandwidth
- Data flows and control flows throughout the entire system and the various subsystems

In addition, you must consider the type of platform to target. You must plan and design for the peripherals and interfaces on the board and the memory resources available on your custom board.

The following figure shows a subsystem that targets a custom hardware platform.

Figure 5: **Custom Hardware Platform Subsystem**



The Vitis design flow is an iterative process that might loop through each step multiple times, adding layers or elements to the subsystem through subsequent iterations. Teams can iterate through the early steps more quickly and take more time with later steps, which provide more detailed performance data.

Following are the recommended steps for creating your design in the Vitis environment.

- **Kernel and Graph Development:** This step includes the development and functional verification of application kernels. These kernels can run on the AI Engine domain or the PL domain.
- **Subsystem Assembly and Verification Using Hardware Emulation:** This step includes assembling the AI Engine and PL kernels with the platform as well as building for hardware emulation using a Xilinx standard platform.
- **Subsystem Assembly and Verification on Hardware:** This step includes building the subsystem against the Xilinx standard platform and testing in real hardware on a Xilinx standard board.
- **Subsystem Integration on Custom Platform:** This step includes building the subsystem against your custom platform and testing using your custom board.

The Vitis environment design flow makes a distinction between platforms and subsystems, which insulates subsystem developers from internal platform details and allows them to build fully functional designs independently. The first three steps of the subsystem design flow assume you are using Xilinx-provided platforms and you are integrating the subsystem to your custom platform in the final step. The custom platform is developed using the Vivado Design Suite and can happen in parallel with the subsystem, which is developed using the Vitis tool flow. This approach reduces risk and uncertainty and increases the chances of success when integrating the subsystem with the custom platform.

Kernel and Graph Development

The first step in this design flow includes the development and functional verification of the individual components of the subsystem: AI Engine graph and PL kernels (HLS and/or RTL). During this step, these components are typically developed and tested independently from one another. However, it is possible to use the Vitis environment hardware emulation flow to start testing the integration of these components.

In this step, verification focuses primarily on functional considerations. Performance information generated for each component typically assumes ideal I/O patterns and no backpressure with data always available. However, it is important to make note of the available performance data, because the system performance is not likely to improve as you progress through your design. Be sure to meet your performance objectives in each step, starting with the first step in the design flow.

Developing the AI Engine Graph and Kernels

An AI Engine program comprises a data-flow graph specification written in C++, which consists of nodes and edges. Nodes represent compute kernel functions, and edges represent data connections. Kernels in the application can be compiled to run on the AI Engines or in the PL region of the device. The AI Engine graph specification is compiled using the aiecompiler and executed with the aiesimulator.

Xilinx recommends gradually refining and testing the graph, slowly progressing from scalar to vectorized operations. Using scalars, you can target AI Engine tiles without having to code with intrinsics right away. This allows you to set up your system (e.g., build scripts, functional correctness, etc.) without having to do low-level AI Engine coding.

The graph is tested with a user-written test bench that drives and manages the graph using the graph APIs. The test bench and graph APIs serve as the foundation for the development of PS firmware in later steps. There are multiple methods for getting data into and out of a graph. Run-time parameters (RTPs) are programmable registers for scalar values. GMIOs provide a direct connection from the AI Engine to global memory. Streaming connections provide a direct connection between AI Engine kernels and PL kernels modeled with PLIOs in the simulation. At this stage in development, file I/O is often the simplest and most effective way to get data into and out of your graph.

Meeting performance in the aiesimulator at this early stage in the design is not a benchmark of the final system performance, because performance data is idealistic at this point. The impact of going out of or into the graph through the PLIOs is difficult to model, which limits the ability to accurately estimate performance.

Developing PL Kernels with Vitis HLS

PL kernels can be developed using C/C++ code and the Vitis HLS tool. The Vitis HLS tool simplifies the use of C/C++ functions for implementation as PL kernels in the Vitis application acceleration development flow.

The Vitis HLS tool automates much of the code modifications required to implement and optimize the C/C++ code in programmable logic and to achieve low latency and high throughput. The Vitis HLS tool allows inference of required pragmas to produce the right interface for your function arguments and to pipeline loops and functions within your code.

Note: Although HLS development is done outside of the AI Engine tool environment, it is possible to optionally include HLS kernels in the AI Engine graph C++ specification.

The Vitis HLS design flow includes the following main steps:

1. Compile, simulate, and debug the C/C++ algorithm.
2. View reports to analyze and optimize the design.
3. Synthesize the C algorithm into an RTL design.

4. Verify the RTL implementation using RTL co-simulation.
5. Compile the RTL implementation into a compiled object file (.xo), or export to an RTL IP.

For more information, see the *Vitis High-Level Synthesis User Guide* (UG1399).

Developing PL Kernels with RTL and the Vivado Design Suite

PL kernels can also be developed using RTL kernels and the Vivado Design Suite. This approach is convenient for hardware engineers that have existing RTL IP, including Vivado IP integrator-based designs, or prefer creating new functions by writing RTL code.

An RTL kernel is a regular design packaged as Vivado Design Suite IP, but the kernel must comply with specific interface rules and requirements to be usable in the Vitis environment design flow. For more information about RTL kernels, see RTL Kernels in the Application Acceleration Development flow of the *Vitis Unified Software Platform Documentation* (UG1416).

Creating an RTL kernel follows traditional RTL design guidelines. Xilinx highly recommends that you create custom test benches and use behavioral simulation to thoroughly verify the RTL code before packaging and using the code as PL kernels in the Vitis environment design flow. After an RTL design is fully verified and meets all the requirements for a Vitis kernel, the design can be compiled into a Vitis kernel object (XO file) using the `package_xo` command.

Subsystem Assembly and Verification Using Hardware Emulation

In the second step of this design flow, you gradually assemble subsystem components (PS, PL, and AI Engine) on top of the target platform and use the Vitis hardware emulation flow to simulate the integrated system. Hardware emulation is a cycle approximate simulation of the system. The AI Engine graph runs in the SystemC simulator. RTL behavioral models of the PL run in the Vivado simulator or a supported third-party simulator. The software code executing on the PS is simulated using the Xilinx Quick Emulator (QEMU).

The target platform contains all of the necessary hardware and software infrastructure resources required for the project. It is possible to target a standard Xilinx platform or a custom platform for your project. At this step in the flow, Xilinx recommends using a standard and pre-verified platform to reduce uncertainty in the process and focus efforts on the system components (graph and kernels).

The Vitis linker (`v++ --link`) is used to assemble the compiled AI Engine graph (`libsdf.a`) and PL kernels (.xo) with the targeted platform. The Vitis linker establishes connections between the AI Engine ports, PL kernels, and other platform resources.

Because this design flow progresses gradually, certain elements might not exist in early iterations. You might need to terminate unconnected signals, drive signals, or provide sinks. In this case, unterminated streaming connections between the AI Engine graph and PL kernels (PLIOs and AXI4-Stream) require the addition of simulation I/Os and traffic generator IP for emulation purposes, which can be added during the linking process using `v++` options.

The Vitis linker automatically inserts FIFOs on streaming interfaces as well as clock domain converters (CDC) and data width converters (DWC) between the AI Engine and PL kernels as needed. On the Versal ACAP, the clock on the AI Engine array can run at 1 GHz, but the clock in the PL region runs at a different, lower frequency. This means there can be a difference between the data throughput of the AI Engine kernels and the PL kernels based on their clock frequencies. When linking the subsystem, the Vitis compiler can insert CDCs, DCWs, and FIFOs to match the throughput capacities of the PL and AI Engine regions.

The Vitis packager (`v++ --package`) is used to add the PS application and firmware and to generate the required setup to run hardware emulation. The PS application controls the AI Engine graph, including how it is loaded, initialized, run, and updated, and the PL kernels. To control the AI Engine graph, you must use the graph APIs generated by the aiecompiler or the standard XRT APIs. To control the PL kernels, Xilinx recommends using the standard XRT APIs. XRT is an open-source library that makes it easy to interact with PL kernels and AI Engine graphs from a software application, either embedded or x86-based.

Optionally, you can build higher-level functionality on top of the graph and PL drivers. For the PS subsystem, you write code in this step that did not fully exist in the first step. Drivers or firmware interact directly with the kernels and a higher-level application that uses these drivers.

You can develop PS firmware, graph drivers, and PL kernels as follows:

- **PS firmware:** Use the test bench from the first step in the design flow, which drives and manages the graph using graph APIs.
- **Graph drivers:** Use the graph APIs to test the graph and to interact with RTPs and GMIOs.
- **PL kernel drivers:** Use XRT APIs or UIO drivers to interact with the PL kernels.

In this step, most models are cycle accurate. However, some models are only approximate, and other models are transaction-level models (TLM). PL kernels are simulated using the target clock, which is not guaranteed to be met during implementation. The interactions between the AI Engine graph and PL kernels are modeled at the cycle level, but overall accuracy depends on the accuracy of the patterns produced by the traffic generators and other test bench modules. The impact of other subsystems or complex I/O interactions cannot be accurately modeled. The slower performance of the emulation environment limits the number of traffic/vectors that can be tested.

***Note*:** Meeting performance in hardware emulation is necessary but is not a guarantee of results. Hardware emulation is cycle approximate with better accuracy in performance than during the first step in the design flow. However, performance results are still not final at this stage.

Subsystem Assembly and Verification on Hardware

After hardware emulation provides a good view of the subsystem, you can proceed to the hardware build on a Xilinx standard platform. Targeting a Xilinx standard platform helps to eliminate some uncertainty from the test environment.

In this step, you are reusing the subsystem from the previous step but are now targeting the hardware build. Using the Vitis linker, you take the assembled PL kernels through synthesis and place and route. Using the Vitis packager, you package the PS and AI Engine programs to generate the required output files to load and run the application on the Xilinx standard development board.

In the early stages of the design, this step is similar to iterating through Vivado synthesis, place and route, and timing closure to achieve optimal results. Iterate until the performance objectives are met, including Fmax, throughput, and resource utilization.

Like the previous two steps of this design flow, this step also allows an incremental approach in which different components are gradually added to the subsystem and taken to hardware. This gradual approach allows you to safely build upon previously verified components, which is a proven strategy to manage design complexity.

From a performance standpoint, running in real hardware gives you more accurate numbers than running in hardware emulation. Potential sources of differences between this step and the preceding step include the following:

- Implementation results in potentially lower clock frequency
- More accurate execution profile of control code running on the PS
- More realistic I/O patterns, resulting in more realistic exercising of stalls and back pressure
- Discovery of corner cases that cannot be reached in the slower hardware emulation runs

Subsystem Integration on Custom Platform

In the final step of this flow, all the elements of the subsystem from the preceding step are integrated and built on the custom platform, which was developed to deploy the application. This step is similar to the previous step but uses the custom platform instead of a Xilinx platform.

The goal of this step is to meet timing and performance closure on the actual target platform. By running through Vivado synthesis and place and route, you can address any differences in timing, utilization, and power that occur when you switch from the Xilinx standard platform to your custom platform.

This step also allows you to test and debug the custom platform as well as the application and subsystem. Testing the subsystem with external I/Os means you might encounter some differences between the previous step and this step in the hardware execution. However, if you designed your custom platform correctly, the standardized interfaces of the platform can insulate subsystem testing from such differences.

Vivado Tools Design Flow

The Vivado Design Suite offers multiple ways to accomplish the tasks involved in Xilinx device design, implementation, and verification. You can use the traditional register transfer level (RTL) to device configuration design flow. You can also use system-level integration flows that focus on intellectual property (IP)-centric design and C-based design. Design analysis and verification is enabled at each stage of the flow. Design analysis features include logic simulation, I/O and clock planning, power analysis, constraint definition and timing analysis, design rule checks (DRC), visualization of design logic, analysis and modification of implementation results, programming, and debugging.

IMPORTANT! *Most Versal ACAP designs are created with the Vitis environment design flow and require a Vitis hardware platform. Xilinx provides standard platforms as starting points, which can be customized and regenerated by the Vivado IP integrator to better fit the target system application.*

System Hardware Design Flow

Xilinx highly recommends using the Vivado IP integrator cockpit for designs that target Versal ACAPs. The Vivado IP integrator lets you create complex system designs by instantiating and interconnecting IP cores from the Vivado IP catalog onto a design canvas. The Vivado IP integrator is designed to simplify Versal ACAP AXI-based IP connectivity.

Vivado IP Integrator

The following sections provide information on significant IP that you can access from the Vivado IP integrator to create and configure your Versal ACAP design. For usage information and general hardware platform generation information, see the *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994).

Versal CIPS IP Core

The CIPS IP allows you to configure the following:

- Device clocking to the PMC, PS, NoC, and optionally, PL
- PS peripherals and their associated I/O
- PS-PL interrupts and cross-triggering

- CPM (the integrated block for PCIe with DMA and cache coherent interconnect)
- PS and CPM AXI interfaces to NoC and PL
- System Monitor supply and temperature monitoring and alarms
- HSDP for high-speed debugging

Versal AI Engine IP

The AI Engine is configured using the AI Engine IP, which lets you define the number of:

- AXI4-Stream master and slave interfaces to and from the AI Engine and PL
- AXI4-Stream clock ports for the PL and NoC channels
- Memory-mapped AXI interfaces to and from the AI Engine to the NoC
- Events being triggered and monitored both from AI Engine and the PL

Note: AI Engine IP is used only for custom platform creation.

Versal AXI NoC IP

The NoC is configured using the AXI NoC IP. The IP acts as logical representations of the NoC. The AXI NoC IP supports the AXI memory-mapped protocol, and the AXIS NoC IP supports the AXI4-Stream protocol. A Versal ACAP design can include multiple instances of each type of IP.

The DDRMC is integrated into the AXI NoC IP. An instance of the AXI NoC can be configured to include one, two, or four instances of the DDRMC. You must use the NoC IP to communicate with the integrated DDRMC. During the validate step, the Versal NoC compiler is run on the unified traffic specification. After validation, the NoC Viewer window allows you to review and edit the NoC solution.

For configuration details on the NoC and related IP as well as details on the system address map, see the *Versal ACAP Programmable Network on Chip and Integrated Memory Controller LogiCORE IP Product Guide* (PG313).

Versal Clocking Wizard

The Versal Clocking wizard generates source RTL code to implement a clocking network matched to your requirements. This wizard generates the clocking network for all Versal clocking primitives.

Versal Transceivers Bridge

The Versal ACAP transceivers are highly configurable and tightly integrated with the PL block. The Versal Transceiver Bridge enables Vivado IP integrator-based design entry for GT-based IP. This allows you to generate designs that use multiple quads or designs that share quads with multiple protocol IP. You must use the Advanced I/O planner to add physical GT locations.

Versal Advanced I/O Wizard

The I/O planning flow for high-performance I/O in Versal ACAP includes significant improvements from previous devices. The I/O planning flow is unified for both memory and non-memory interfaces in the Advanced I/O Planner. If you previously generated high-performance interfaces using the High-Speed SelectIO™ wizard or SelectIO component mode, you must rebuild the interfaces using the Versal Advanced I/O wizard, which now supports multi-bank interface generation.

Design Address Map

The Versal ACAP uses a single, unified system address map. All memory-mapped AXI transactions must adhere to this map. The Versal ACAP system address map defines the default address locations of slaves in the Versal ACAP. The address map is built into the fabric interconnect and the NoC. The Vivado IP integrator automatically resolves the base name, offset address, and range of the address region based on the DDR4 memory options selected in the AXI NoC IP customization. These addresses are used by the AXI master to communicate with the DDR. You use the Vivado IP integrator Address Editor to select or automatically assign compliant addresses for all the memory-mapped blocks within the design. For configuration details on the NoC and related IP as well as details on the system address map, see the *Versal ACAP Programmable Network on Chip and Integrated Memory Controller LogiCORE IP Product Guide* (PG313).

RTL Design Flow

The RTL design flow is provided for designers that do not want to use Vivado IP integrator. If you choose not to use Vivado IP integrator, essential automation related to the CIPS and NoC IP is *not* available, including the following:

- PS functions (e.g., Arm cores)
- Automation across all blocks connected to CIPS and NoC
- Connection automation between CIPS and NoC

In a pure RTL flow, you can instantiate the CIPS IP primarily for accessing device configuration features and use the NoC to exclusively connect several AXI masters in the PL to one or several DDRMC IP. Alternatively, you can use Vivado IP integrator to integrate and configure the CIPS IP and NoC IP together, specify the interface to the rest of the design, and instantiate the resulting block design in the top-level RTL. Although the final method allows access to all Versal ACAP advanced hardware features, the design must maintain both the RTL and the block design in parallel when making any functionality or connectivity change.

Vivado IP Catalog

The following sections provide information on essential IP that you can access from the Vivado IP catalog to create and configure your Versal ACAP design.

Versal CIPS IP

The CIPS IP allows you to configure the following:

- Device clocking to the PMC, PS, NoC, and optionally, PL
- PS peripherals and their associated I/O
- PS-PL interrupts and cross-triggering
- CPM (the integrated block for PCIe with DMA and cache coherent interconnect)
- PS and CPM AXI interfaces to NoC and PL
- System Monitor supply and temperature monitoring and alarms
- HSDP for high-speed debugging

Versal AXI NoC IP

Depending on your design, you can use the IP in one of the following ways:

- In designs with fabric-based logic that access the DDRMC, you can use the AXI NoC IP from the Vivado IP catalog.
- In designs where the DDR memory is shared across the PL and the PS, you must use the Vivado IP integrator design flow to create a block design (BD) with at least both CIPS and AXI NoC IP and instantiate the BD in the RTL design.

The NoC is configured using the AXI NoC IP. The IP acts as logical representations of the NoC. The AXI NoC IP supports the AXI memory-mapped protocol, and the AXIS NoC IP supports the AXI4-Stream protocol. A Versal ACAP design can include multiple instances of each type of IP.

The DDRMC is integrated into the AXI NoC IP. An instance of the AXI NoC can be configured to include one, two, or four instances of the DDRMC. You must use the NoC IP to communicate with the integrated DDRMC. During the validate step, the Versal NoC compiler is run on the unified traffic specification. After validation, the NoC Viewer window allows you to review and edit the NoC solution.

For configuration details on the NoC and related IP as well as details on the system address map, see the *Versal ACAP Programmable Network on Chip and Integrated Memory Controller LogiCORE IP Product Guide* (PG313).

Versal Clocking Wizard

The Versal Clocking wizard generates source RTL code to implement a clocking network matched to your requirements. This wizard generates the clocking network for all Versal clocking primitives.

Versal Transceivers Wizard

The Versal ACAP transceivers are highly configurable and tightly integrated with the PL block. The Versal Transceivers wizard automatically generates XDC file templates that configure the transceivers and contain placeholders for transceiver placement information. This wizard also supports generating example designs to simulate key features like rate switching and quad sharing across various protocols.

***Note*:** The Vivado IP integrator is recommended for designs sharing GT quads.

Versal Advanced I/O Wizard

The I/O planning flow for high-performance I/O in Versal ACAP includes significant improvements from previous devices. The I/O planning flow is unified for both memory and non-memory interfaces in the Advanced I/O Planner. If you previously generated high-performance interfaces using the High-Speed SelectIO™ wizard or SelectIO component mode, you must rebuild the interfaces using the Versal Advanced I/O wizard, which now supports multi-bank interface generation.

Design Address Map

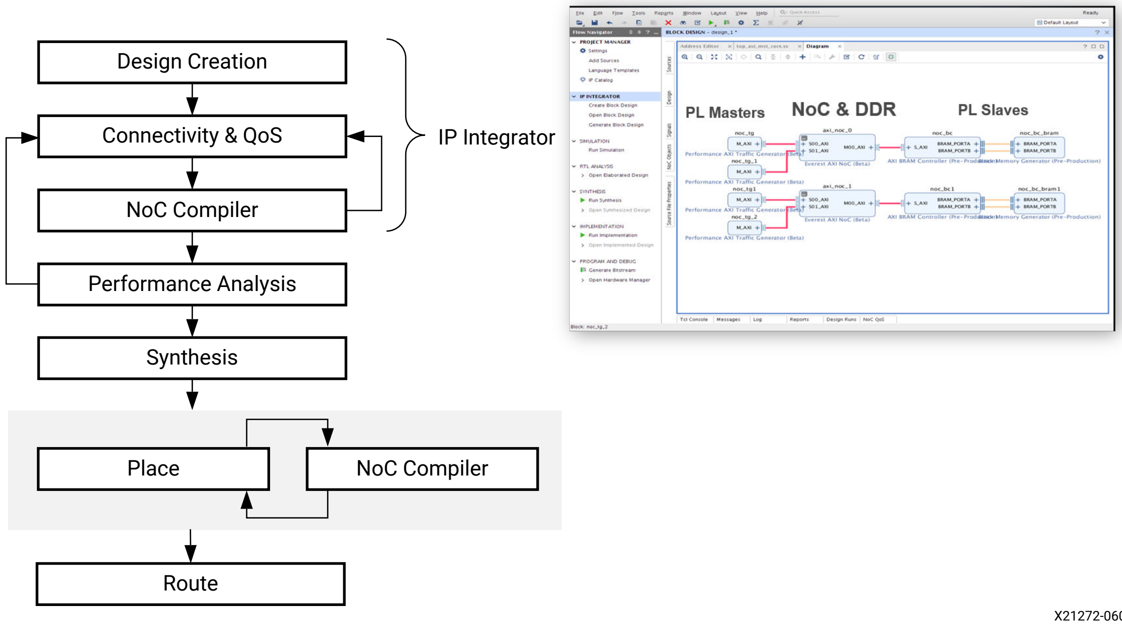
The Versal ACAP uses a single, unified system address map. All memory-mapped AXI transactions must adhere to this map. The Versal ACAP system address map defines the default address locations of slaves in the Versal ACAP. The address map is built into the fabric interconnect and the NoC. The Vivado IP integrator automatically resolves the base name, offset address, and range of the address region based on the DDR4 memory options selected in the AXI NoC IP customization. These addresses are used by the AXI master to communicate with the DDR. You use the Vivado IP integrator Address Editor to select or automatically assign compliant addresses for all the memory-mapped blocks within the design. For configuration details on the NoC and related IP as well as details on the system address map, see the *Versal ACAP Programmable Network on Chip and Integrated Memory Controller LogiCORE IP Product Guide* (PG313).

Implementation

You must use Vivado tools for synthesis and implementation. The Versal ACAP includes new primitives that the Vivado synthesis tool infers.

Design closure techniques related to the device fabric are similar to previous device families. In addition to considerations for timing, congestion, and wirelength, the placer calls the NoC compiler again to account for modified NoC port location constraints or for merging traffic in Dynamic Function eXchange (DFX) mode while meeting the original QoS requirements, as shown in the following figure.

Figure 6: **NoC Compiler Flow**



Related Information

Primitives

Power and Thermal

Use the Versal ACAP Xilinx Power Estimator (XPE) to estimate power requirements. XPE is a powerful tool for getting early power estimates for your design. The Versal ACAP version of XPE is similar to the XPE for previous devices but includes various enhancements, including a new Power Design sheet with voltage regulator assignment and decoupling capacitor recommendations. In addition, XPE can import data files from the AI Engine Compiler and NoC Compiler with accurate resource information for power estimation.

***Note*:** AI Engine Compiler data is not simulation based and is an estimation from the tool.

Versal devices are available in both lidded and lidless packaging. Where possible, Xilinx recommends lidless packaging, which offers an optimized thermal solution that minimizes thermal resistance and provides the most effective Theta Ja possible. From the XPE Power Design sheet, you can access Xilinx-provided thermal models that support Siemens Simcenter Flotherm and Ansys Icepac. For verified power delivery solutions for Versal ACAP, see the Power page on the Xilinx website.

I/O Planning

For Versal ACAP, the I/O planning flow for high-performance I/O differs from previous devices, and the I/O planning flow for low-performance I/O (e.g., below 500 Mb/s) remains the same.

XPIO

The high-performance I/O in Versal ACAP is known as XPIO. The XPIO are located at the bottom periphery of the device, unlike the columnar I/O architecture found in previous devices. XPIO ports that exist below the PS on the left side of the device and below the GTs on the right side of the device are known as corner I/O. Corner I/O have limited use, such as for the integrated DDRMC and limited clocking. For more information on corner I/O, see the *Versal ACAP Packaging and Pinouts Architecture Manual* (AM013).

The XPIO provide XPHY logic that is similar to UltraScale™ device native mode. The XPHY logic encapsulates calibrated delays along with serialization and deserialization logic for 6 single-ended I/O ports known as nibble. Each XPIO bank contains 9 XPHY logic sites and allows for up to 54 single-ended I/O ports. The XPHY logic is used for the integrated DDRMC, soft memory controllers, and any high-performance I/O interfaces.

IMPORTANT! *Individual component mode cells, such as IDELAY, ODELAY, ISERDES, OSERDES, IDDR, and ODDR, are eliminated for high-performance interfaces. The ISERDES and OSERDES primitives are not supported in the Versal architecture, but similar functionality is supported through the XPHY logic.*

Uncalibrated IDELAY, ODELAY, IDDR, and ODDR, known as I/O logic (IOL), exist in both XPIO and HDIO banks to support legacy low-performance interfaces operating at 500 Mb/s and below.

The I/O planning flow for high-performance interfaces is different from previous architectures due to the use of XPHY logic. If you previously generated high-performance interfaces using the Xilinx Memory Interface Generator, High-Speed SelectIO wizard, or SelectIO component mode, you must rebuild the interfaces using Versal IP wizards.

The following table shows how the high-performance UltraScale device I/O generation maps to the Versal device I/O generation.

Table 6: **Device I/O Generation Comparison**

UltraScale Device I/O Generation	Versal ACAP I/O Generation
Soft memory controllers	Integrated DDRMC via the Versal NoC IP Soft memory controllers
High Speed SelectIO Wizard	Versal Advanced I/O Wizard
UltraScale Component Mode • High-performance interfaces • Calibrated IDELAY, ODELAY, ISERDES, OSERDES, IDDR, and ODDR	Versal Advanced I/O Wizard
UltraScale Component Mode • Low-performance interfaces (500 Mb/s and below) • Uncalibrated IDELAY, ODELAY, IDDR, and ODDR	I/O logic instantiated in RTL

After you regenerate the IP for the Versal ACAP, you can perform I/O planning using the Advanced I/O Planner, which is similar to soft memory controller I/O planning flow for UltraScale devices. The Advanced I/O Planner guides you through the process of mapping your interfaces to the desired XPIO banks using the XPHY logic, ensuring that your high-speed interfaces are legally mapped to the XPHY logic. For more information, see the *Advanced I/O Wizard LogiCORE IP Product Guide* (PG320).

Xilinx recommends I/O planning high-speed interfaces in the following order to achieve the maximum utilization of available XPHY logic resources:

1. Integrated DDRMC via NoC
2. Soft memory controllers
3. Advanced I/O wizard
4. I/O logic

For information, see the following documents:

- For DDR4 and LPDDR4 pinout rules, see the *Versal ACAP Programmable Network on Chip and Integrated Memory Controller LogiCORE IP Product Guide* (PG313).
- For information on the Advanced I/O wizard, see the *Versal ACAP SelectIO Resources Architecture Manual* (AM010).

High-Density I/O

The low-performance I/O in Versal ACAP are known as high-density I/O (HD I/O). The HD I/O support a subset of the UltraScale device component mode primitives through uncalibrated IDELAY, ODELAY, IDDR, and ODDR primitives known as I/O logic. HD I/O maintain the columnar I/O architecture found in previous devices.

The I/O planning flow for HD I/O is unchanged from previous architectures. You can continue to instantiate the I/O logic primitives in your HDL code. The tools support an XDC-based constraints flow for assigning PACKAGE_PIN constraints. As with previous architectures, you can drag and drop from the I/O Ports window onto the Package window. In addition, you can move I/O logic primitives between HD I/O banks and XPIO banks in the Versal ACAP.

IMPORTANT! *The voltage ranges between XPIO and HD I/O do not overlap. XPIO supports a lower range of voltage than HD I/O.*

Clocking

The Versal ACAP clocking architecture introduces some hardware improvements and changes from the UltraScale architecture. For detailed architecture information, see the *Versal ACAP Clocking Resources Architecture Manual* (AM003).

Design Closure

The Versal architecture introduces new hardware features that require additional considerations to reach design closure, including timing and performance closure.

Simulation Flows

To address the different needs in simulation scope, abstraction, and purpose, Xilinx provides dedicated flows for the various components of a Versal ACAP design, including the AI Engine, PS, and PL. The hardware emulation system simulation flow also allows for simulating the entire system in a single setup.

AI Engine Simulation

AI Engine simulation tests code running on the Versal AI Engines. The scope of this simulation is an arbitrary number of AI Engine graphs and kernels, and you can include PL kernels inside the graph modeled in C++ or SystemC. Two abstractions are supported, untimed and cycle-approximate, providing a trade-off between simulation speed and accuracy. The purpose of this simulation is to verify the functional correctness of the code and to validate performance in a standalone context, independently of interactions with other functions.

AI Engine simulation is available through the Vitis unified software platform.

HLS Simulation

HLS simulation exclusively tests HLS code and is an integral part of the HLS development process. The scope of this simulation is a single HLS kernel. Two abstractions are supported, untimed and RTL (cycle-accurate). These two abstractions are referred to as Csim and Cosim respectively. In the Cosim flow, the output of RTL code generated by the HLS compiler is automatically compared against the output of the original C code. The purpose of this flow is to verify the functional correctness of the RTL and to validate performance in a standalone context, independently of interactions with other functions.

HLS simulation is available through the Vitis unified software platform. For more information, see the *Vitis High-Level Synthesis User Guide* (UG1399).

Embedded Software Simulation

Embedded software simulation tests a software design that targets only the PS. It is based on the Quick Emulator (QEMU), which emulates the behavior of the dual-core Arm® Cortex™-A72 integrated in the Versal ACAP device. This simulation enables a fast, compact functional validation of the platform OS. This flow includes a SystemC transaction-level model of the system, which allows for early system exploration and verification.

Embedded software simulation is available through the Vitis unified software platform.

Logic Simulation

Logic simulation tests a hardware design targeting the PL fabric and is the traditional FPGA simulation flow. The scope of this simulation is scalable, ranging from individual hardware blocks to the complete hardware platform. The simulated models are generally RTL, making the abstraction cycle-accurate. Simulation speed is proportional to the size of the test design, and larger designs take comparatively longer to simulate. To improve simulation performance, you can replace some Versal ACAP IP blocks with SystemC transaction-level models, which simulate faster but are no longer be cycle-accurate. The purpose of this simulation is to verify and debug detailed hardware functionality before implementing the design on the device.

Logic simulation is available through the Vivado Design Suite. For more information, see the *Vivado Design Suite User Guide: Logic Simulation* (UG900).

Hardware Emulation

Hardware emulation simulates a complete Versal ACAP system composed of the AI Engine, PS, and PL. Using the Vitis software platform, you can integrate blocks and functions targeting all three compute domains. The Vitis linker automatically generates a complete co-simulation setup involving RTL, SystemC, and QEMU models:

- Embedded software code running on the PS is emulated using QEMU.
- Code running on the AI Engines is emulated using the SystemC AI Engine simulator.
- User PL kernels are simulated as RTL code.
- IP blocks in the hardware platform are simulated either as RTL or SystemC TLM, based on the types of models available or selected.

As a result, the abstraction of the Vitis hardware emulation is very close to but not fully cycle-accurate. Some details of the Versal ACAP platform are abstracted with TLM models for simulation speed purposes.

The scope of the Vitis hardware emulation also defines its purpose. Hardware emulation allows you to simulate the entire design and test the interactions between the PL, PS, and AI Engine prior to implementation. Because hardware emulation provides full debug visibility into all aspects of the application, it is easier to debug complex problems in this environment than in real hardware.

Hardware emulation is available through the Vitis unified software platform. For more information, see *Vitis Unified Software Platform Documentation: Application Acceleration Development* (UG1393).

Boot and Configuration

Versal ACAPs have a centralized PMC subsystem responsible for the boot-up process, security, power management, and integrated debug. The Versal ACAP includes a separate power domain for the PMC that must be powered and must perform boot-up prior to configuring the PL, NPI, and PS elements. Versal ACAP uses the BootROM and PLM for booting.

If you are migrating from UltraScale+™ device families, consider the following:

- **UltraScale+ device designs:** These devices contain integrated configuration logic that supports a set of configuration modes on power-up. With Versal ACAP, there are changes to the boot and configuration flows.
- **Zynq UltraScale+ MPSoC PS designs:** Zynq® UltraScale+™ MPSoCs have a PMU and CSU to manage and carry out the boot-up process. There are changes in the boot flow methodology.

***Note*:** For more information, see the *Versal ACAP Technical Reference Manual* (AM011).

The following table compares the boot and configuration modes of UltraScale+ devices with Versal ACAP.

Table 7: **Boot Mode Comparison**

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

Notes:

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

Programmable Device Image

The Versal ACAP PMC uses a proprietary boot and configuration file format called the programmable device image (PDI) to program and configure the Versal ACAP. The PDI consists of headers, the PLM image, and design data image partitions to be loaded into the Versal ACAP. The PDI also contains configuration data, ELF files, NoC register settings, etc. The PDI image is programmed through the PMC block by the BootROM and PLM.

IMPORTANT! *The Vivado tools require the CIPS IP to be present in the design to create the PDI image. For pure RTL designs, access the CIPS IP in the Vivado IP catalog and instantiate the CIPS IP using the default configuration to enable the PDI creation.*

For information on the differences between the Versal architecture and previous device architectures, see the *Versal ACAP Technical Reference Manual* (AM011).

Chapter 5

System Migration

When you migrate designs to Versal™ ACAP from the UltraScale™ architecture, the Xilinx® tools can only automatically migrate some of the PL primitives and integrated IP blocks due to the functional and connectivity differences. Partial migration is possible but generally leads to sub-optimal hardware and application performance. Therefore, Xilinx recommends using the following steps instead:

- Rearchitect any high-bandwidth connections between major blocks to use the NoC instead of the PL-based AXI Interconnect or similar IP.
- Reduce PL logic by leveraging all new integrated blocks, such as the integrated memory controller, DMA, and AI Engine.
- Replace instantiated PL primitives from previous architectures with the equivalent RTL descriptions or XPMs (e.g., memory blocks, DSPs, carry logic, multiplexers, etc.).
- Regenerate or recreate all IP blocks.
- Resynthesize the complete design instead of migrating netlists created for previous architectures.

Any portion of the design that is automatically migrated must be carefully reviewed to ensure that the application's performance, resource, and power will be met.

The following table shows the blocks for which automatic migration is available.

Table 8: **Block Migration Support**

Block	Automated
Soft memory controllers	No
GT	No
AXI Interconnect	No
Power and error handling	No
System monitor (SYSMON)	No
DSP	Yes
On-chip memory (OCM) resources (block RAM and UltraRAM)	Most
Configurable logic block (CLB)	Yes
System debug	No
Processor and peripherals	No
I/O	Some

Table 8: **Block Migration Support** *(cont'd)*

Block	Automated
PCIe subsystem	Some
Tandem PCIe interfaces	No
PL configuration and JTAG	No

IMPORTANT! *If your existing design contains blocks that are discontinued in Versal ACAP, you must manually migrate these blocks to a corresponding Versal ACAP block. For details, see the appropriate Versal Architecture Manual.*

For designs migrating from Kintex® UltraScale™ or Virtex® UltraScale™ devices, the CIPS IP must be added to enable essential functionality, such as device configuration and hardware debug features. Other designs migrating from Zynq® UltraScale+™ MPSoCs are expected to already have a PS block.

Soft Memory Controllers

If your previous design used soft memory controller IP, you can either use the Versal ACAP soft memory controller IP or the integrated DDRMC. Xilinx recommends using the integrated DDRMC rather than using the Versal ACAP soft memory controller IP. In Versal ACAP, you can use the integrated DDRMC only via the NoC. The NoC and DDRMC have very high bandwidth but generally have a higher latency than a standalone soft memory controller. For some I/O banks, only the integrated DDRMC is supported. For more information on the DDRMC, see the *Versal Architecture and Product Data Sheet: Overview* (DS950).

If you are using the soft memory controller IP, you must regenerate the IP for Versal ACAP. In Versal ACAP, an I/O bank comprises nine nibbles, and each nibble has six pins. Depending on the device and package, some I/O banks or some nibbles in an I/O bank are dedicated for the integrated DDRMC. Soft memory controllers cannot use these dedicated pins. Pins dedicated for the integrated DDRMC are designated as YES in the package file under the column named DDRMC ONLY. The soft memory controllers can only use pins designated as NO.

GT

For Versal ACAP, GT components are updated from Common/Channel to a quad granularity. To enable some of the GT sharing use cases, GT wizard flows are modified to use the Vivado® IP integrator. Use the Vivado IP integrator to build system designs and Advanced I/O planner to add physical GT locations. This results in some manual migration effort for most of the non-Xilinx GT-based IP. When migrating your design, you must be aware of the full GT quad layout and supported configuration options. For detailed architectural differences, see the *Versal ACAP GTY Transceivers Architecture Manual* (AM002).

AXI Interconnect

The soft IP AXI Interconnect is fully replaced by a combination of the integrated NoC resources and SmartConnect IP. When migrating your design, first consider using NoC resources for all memory access pathways as well as to reduce PL resource utilization and support the high-bandwidth connections. You can then use SmartConnect to accommodate some conversions onto the NoC or to offload traffic from a fully-utilized NoC network. When migrating your design to NoC resources, Xilinx strongly recommends using Vivado IP integrator for the instantiation and configuration of the NoC IP. For more information, see the *Versal ACAP Programmable Network on Chip and Integrated Memory Controller LogiCORE IP Product Guide* (PG313) and *SmartConnect LogiCORE IP Product Guide* (PG247).

Power and Error Handling

Zynq UltraScale+ MPSoCs have some power modes that can be mapped to the power modes in Versal ACAP. Because the DDR is shared in the Versal ACAP, the DDR power modes, including DDR self refresh and system power domain switching, are handled in the PLM. Zynq UltraScale+ MPSoCs had error handling that was bound to the PS. In Versal ACAP, the PS handles its own errors but sends a summary to the PMC if action is required by the PMC. Errors from the DDR, PL, and SYSMON are handled by the PMC in Versal ACAP instead of the PS. For detailed architectural differences, see the *Versal ACAP Technical Reference Manual* (AM011).

System Monitor

The Versal ACAP provides system monitoring capabilities similar to UltraScale+ devices. In UltraScale+ device designs you instantiated the SYSMON IP on the PL side and used the System Management wizard to set up the register configuration for instantiation in the hardware description language (HDL). In Versal ACAP, you configure the System Monitor settings in the CIPS IP core. To migrate your design, you must manually remove the SYSMON IP from the design. Xilinx recommends configuring the System Monitor using the CIPS IP. For more information, see the *Versal ACAP System Monitor Architecture Manual* (AM006).

DSP

The Versal ACAP includes the DSP58 slice, which is a superset of and backward compatible with the UltraScale+ device DSP48E2 slice. In addition, the Versal ACAP DSP Engine supports floating point operations in a single DSP58 slice and can combine two back-to-back DSP58 slices with dedicated interconnect to build an 18-bit complex multiplier or complex multiply-accumulate (MACC). The DSPFP32 mode in Versal ACAP is supported through the Floating-Point Operator IP. If you want to use this mode, update the Floating- Point Operator IP in your migrated design.

Xilinx supports automated migration of designs with DSP slices by inferring the appropriate internal Versal ACAP legacy primitive (DSP48E5) for the DSP slices. Register-transfer level (RTL) instantiations are also automatically migrated. When migrating inferred code, Xilinx supports automatic migration using the same templates and resources. However, inference templates for previous architectures do *not* automatically update to take advantage of the new modes of operation in the DSP58 (e.g., DSPCPLX). Xilinx recommends manual migration to take advantage of the new features in the Versal ACAP DSP58 slice for superior performance.

For detailed architectural differences, see the *Versal ACAP DSP Engine Architecture Manual* (AM004).

IMPORTANT! *To take advantage of the Versal ACAP potential for increasing performance, consider which parts of the datapath can be ported from the PL and into the AI Engines.*

Related Information

DSP Primitives

AI Engine

On-Chip Memory Resources

Block RAM and UltraRAM used in designs from previous architectures are automatically migrated by inferring the appropriate Versal ACAP block. RTL instantiations are also automatically migrated. If certain block RAM and UltraRAM configurations are not supported in Versal ACAP, a critical warning message is issued and the instance is converted to a black box element. The design must be changed to adhere to the supported configurations for Versal ACAP. Xilinx recommends that you examine the configuration settings after design migration to ensure the correct defaults and settings were automatically selected. Xilinx recommends using Xilinx parameterizable macros (XPMs) to infer FIFOs and other memories. Integrated FIFOs are not supported in Versal ACAP. In the Vivado IP integrator flow, the Embedded Memory Generator and Embedded FIFO Generator replace the Block Memory Generator and FIFO Generator IP. The migration for the Block Memory Generator and FIFO Generator IP is *not* automatic. For detailed architectural differences, see the *Versal ACAP Memory Resources Architecture Manual* (AM007).

Related Information

RAM Primitives

CLB

CLBs in Versal ACAP have been enhanced from previous architectures. CLB resources that are no longer supported in Versal ACAP (e.g., CARRY8, MUXF7, MUXF8, MUXF9, etc.) are automatically migrated by inferring the appropriate Versal ACAP block. RTL instantiations are also automatically migrated. For optimal area and timing results, Xilinx recommends that you do not instantiate CLB UNISIMs that are no longer supported in Versal ACAP and that you re-synthesize your RTL to infer the appropriate Versal ACAP block. For detailed architectural differences, see the *Versal ACAP Configurable Logic Block Architecture Manual* (AM005).

Related Information

CLB Primitives

System Debug

Debugging designs in PL fabric is similar to previous architectures, but there are several key differences:

- All fabric debug IP cores have AXI4-Stream slave control interfaces. Previous architectures used a proprietary interface standard.

- The debug hub IP core has both AXI4-Stream master control interfaces (for connection to fabric debug IP cores) and an AXI4-Memory Map slave interface for connection from the host. Debug hub IP in previous architectures relied on proprietary interfaces for connection to the debug cores and host.
- The debug flows in the Vivado tools now support both automated and manual connectivity between debug hub and debug cores.
- The JTAG-to-AXI soft debug IP is no longer offered as an option in the Versal ACAP architecture. The DAP and DPC can be used to access AXI-based blocks in your design.
- The AXI4-Stream-based integrated logic analyzer (ILA) core supports both ILA and System ILA functionality. In previous architectures, these were offered as separate IP cores.

When migrating, consider the following:

- **Vivado IP integrator:** You must manually remove or replace previously instantiated legacy debug cores. Replace the legacy debug cores with the new AXIS-ILA cores in the block design using IP integrator.
- **Netlist:** Xilinx design constraints (XDC) commands for inserting ILA cores into the synthesized design automatically migrate to the new AXIS-ILA debug IP.
- **RTL:** Due to the new interface requirements, the fabric debug cores from previous architectures are not automatically migrated to the new AXI4-Stream-based debug IP cores. If debug cores from previous architectures are instantiated in the design, new debug IP must be manually recustomized, regenerated, and reinstantiated in the design.
- **IBERT and soft memory controller calibration:** The integrated bit error ratio test (IBERT) IP functionality is part of the GT blocks and can be used with any design that uses the transceivers. Memory controller calibration debug is available for both DDRMC blocks and for fabric-based soft memory controller IP.
- **Debug Hub:** Due to the new interface requirements, the legacy Debug Hub is automatically inserted into the netlist only if pl0_resetn is enabled on the CIPS. Alternatively, an AXI4 Debug Hub can be manually added.

Processor and Peripherals

Software stack for bare-metal applications and Linux applications on the PS in Versal ACAP are similar to Zynq UltraScale+ MPSoCs. Versal ACAP uses the PLM for booting. Zynq UltraScale+ MPSoC designs that target the APU can be migrated to work with the Versal ACAP APU. The Versal ACAP RPU uses the same Arm® Cortex™-R5F processor with the same GIC as Zynq UltraScale+ MPSoCs. The functionality and programming models are very similar. UltraScale+ device designs that target the RPU can be migrated to work with the Versal ACAP RPU. When migrating to Versal ACAP, you must take into consideration the device driver changes, multiplexed I/O (MIO) configuration, and pinouts. For more information, see the *Versal ACAP Technical Reference Manual* (AM011).

I/O

IDDR and ODDR are the only primitives that are automatically migrated.

Related Information

I/O Planning

PCIe Subsystems

The Versal architecture includes several blocks for implementation of high performance, standards-based interfaces built on PCI™-SIG technologies. In addition to the CPM, the Versal architecture includes support for implementation of PCIe® interfaces in the PL. PL PCIe are significantly enhanced implementations of the integrated blocks for PCIe found in previous architectures. Two implementations of the PL PCIe exist: PL PCIE4 and PL PCIE5.

In Versal ACAPs with available PL PCIE4, the block is compliant with the PCIe Base Specification Revision 4.0 and capable of supporting defined line rates up to the maximum of 16 GT/s. DMA/Bridge subsystems for use with the PL PCIE4 are available through the Vivado IP catalog as additional soft IP. PL PCIE4 does not provide CCIX support.

In Versal ACAPs with available PL PCIE5, the block is compliant with the PCIe Base Specification Revision 5.0 and capable of supporting defined line rates up to the maximum of 32 GT/s. DMA/Bridge subsystems for use with the PL PCIE5 are available through the Vivado IP catalog as additional soft IP. CCIX support in PL PCIE5 complies with CCIX Base Specification Revision 1.1 and enables solutions via additional soft IP.

Previous designs that use the integrated blocks for PCIe are automatically migrated to Versal ACAP PL PCIE4 or PL PCIE5. Therefore, most of the configuration selections and IP interfaces are compatible, and design migration is automated.

If your design needs to be migrated from an integrated block for PCIe in a previous architecture to a Versal architecture CPM, use the following methods:

- Configure the PCIe subsystem in the CPM using the CIPS IP core.
- Manually map the AXI4 memory-mapped (AXI4-MM) interfaces, including the AXI4-MM bridge, Xilinx DMA memory-mapped (XDMA-MM) interface, and queue DMA memory-mapped (QDMA-MM) interface, into the Versal ACAP NoC infrastructure. This requires setting up various components in the design, such as the NoC, PS, address translation, and address allocation.
- Manually map RQ/RC/CQ/CC streaming, XDMA streaming, and QDMA streaming interfaces to Versal CPM PL interfaces. These interfaces are very similar to their respective IP implementation from previous architectures.

Tandem PCIe configuration is different for Versal ACAP from previous architectures, because configuration occurs through the PMC rather than through the media configuration access port (MCAP) and internal configuration access port (ICAP). Currently, you must manually configure these connections and settings in your design.

Xilinx recommends using the CPM, if available, as the primary PCIe interface for Versal ACAP. This block has hardened paths to the NoC infrastructure and resources, including the PMC, PS, and other management resources.

Tandem support for Versal ACAP devices is expected to include CPM-based Tandem PCIe, CPM-based Tandem PROM, CPM-based DFX over PCIe, and PL-based Tandem PROM. There are no current plans to support Tandem PCIe for PL-based PCIe controllers.

***Note*:** Additional support is planned to assist with migration for PCIe-based Tandem and Dynamic Function eXchange solutions.

For more information, see the following documents:

- *Versal ACAP CPM CCIX Architecture Manual* (AM016)
- *Versal ACAP Integrated Block for PCI Express LogiCORE IP Product Guide* (PG343)
- *Versal ACAP PCIe PHY LogiCORE IP Product Guide* (PG345)
- *Versal ACAP CPM Mode for PCI Express Product Guide* (PG346)

Related Information

CPM

Security

Security is different for Versal ACAP from previous architectures. The root of trust starts with the PMC ROM, which authentications and/or decrypts the PLM software and can only be loaded into and run from the platform processing unit (PPU) in the PMC. After the PLM software is trusted, the PLM handles loading the rest of the firmware and software securely. The following table highlights the possible secure boot configurations. For more information, see the *Versal ACAP System Software Developers Guide* (UG1304) or contact Xilinx.

Table 9: **Cumulative Secure Boot Operations**

Boot Type	**Operations**			**Hardware Crypto Engines**
	Authentication	**Decryption**	**Integrity (Checksum Verification)**	
Non-secure boot	No	No	No	None
Hardware root of trust	Yes (Required)	No	No	RSA/ECDSA along with SHA3
Encrypt-only (Forces decryption of PDI with eFUSE black key)	No	Yes (Required PLM and Meta Header should be encrypted with eFUSE KEK)	No	AES-GCM
Hardware root of trust + Encrypt-only	Yes (Required)	Yes (Required)	No	RSA/ECDSA along with SHA3 and AES-GCM
Authentication + Decryption of PDI	Yes	Yes (Key source can be either from BBRAM or eFUSE)	No	RSA/ECDSA along with SHA3 and AES-GCM
Decryption (Uses user-selected key. The key source can be of any type such as BBRAM/ BHDR or even eFUSE)	No	Yes	No	AES-GCM
Checksum Verification	No	No	Yes	SHA3

PL Configuration and JTAG

The Versal architecture differs from previous architectures for boot and configuration. The PL configuration and JTAG primitives are not supported in Versal ACAP but similar capability exists as follows:

- The BSCANE2 primitive is replaced by four TAP user instructions available in the CIPS IP.
- The STARTUPE3 primitive is replaced by the QSPI controllers and MIO in the PMC available in the CIPS IP.

- The DNA and EFUSE_USR are replaced by memory mapped AXI registers that can be read to get the device DNA or the user-programmable 32-bit eFUSE value. For more information, including address mapping, see the *Versal ACAP Technical Reference Manual* (AM011).

Primitives

The Versal™ ACAP includes new primitives that the Vivado® synthesis tool infers.

***Note*:** This appendix only covers the Versal ACAP primitives that differ from those in the UltraScale+™ device families.

RAM Primitives

The Versal architecture supports both block RAM and UltraRAM primitives.

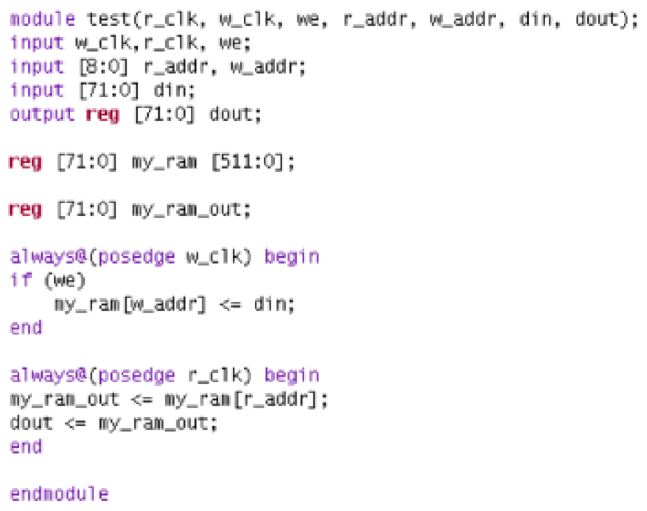
Block RAM Primitives

Following are the block RAM primitives in Versal ACAP.

Primitives	Supported Aspect Ratios	Supported Mode
RAMB36E5	1Kx36 2Kx18 4Kx9	x72 mode when running in simple dual-port (SDP) mode
RAMB18E5	1Kx18 2Kx9	x36 mode when running in SDP mode

In SDP mode, one address reads the RAM and the other address writes to the RAM. You can use different clocks for the read and the write, but the address lines must be separate. The following figure shows an example.

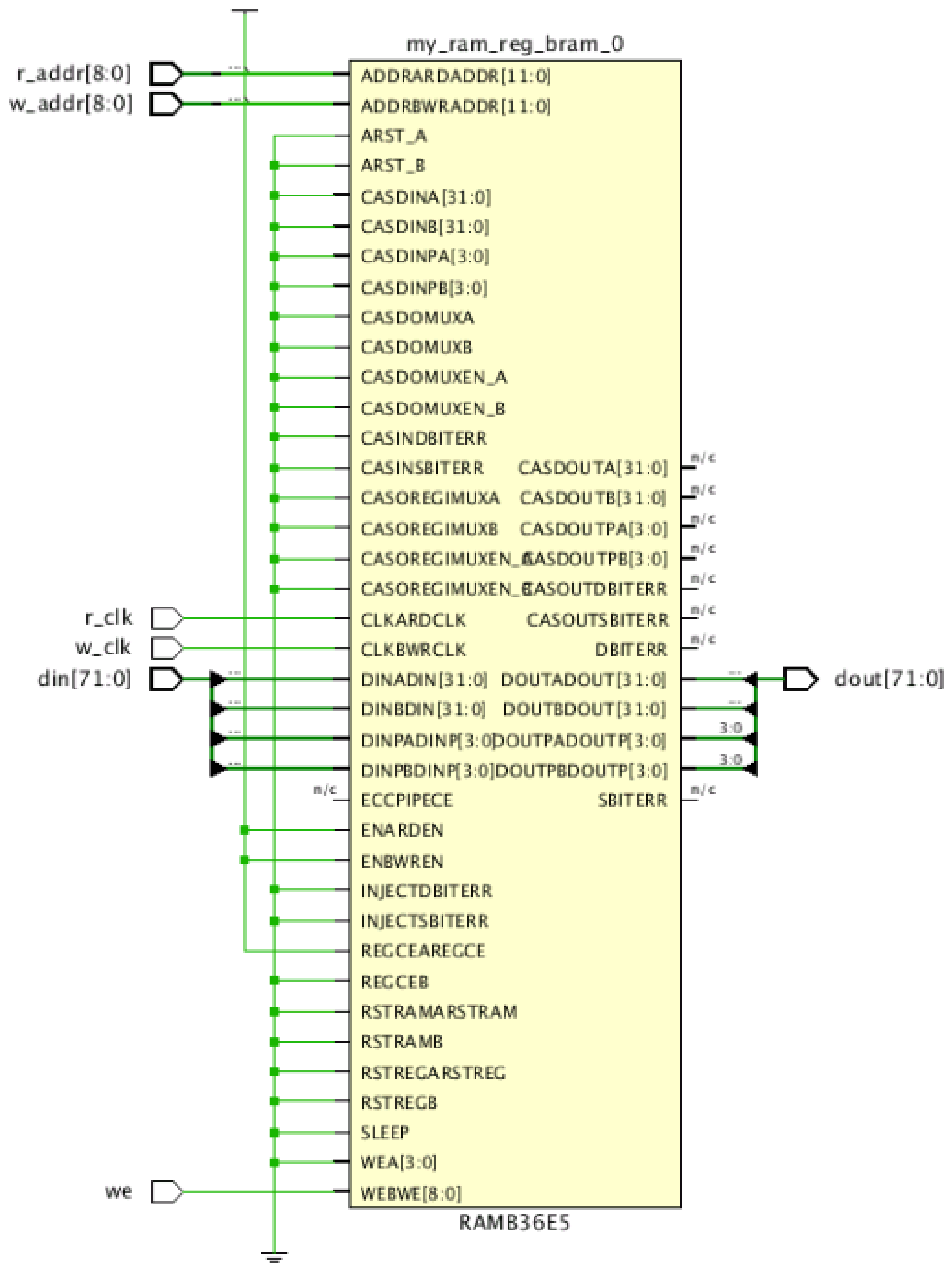
Figure 7: **Coding Style for a 512x72 Block RAM in SDP Mode**



```
module test(r_clk, w_clk, we, r_addr, w_addr, din, dout);
input w_clk,r_clk, we;
input [8:0] r_addr, w_addr;
input [71:0] din;
output reg [71:0] dout;

reg [71:0] my_ram [511:0];

reg [71:0] my_ram_out;

always@(posedge w_clk) begin
if (we)
    my_ram[w_addr] <= din;
end

always@(posedge r_clk) begin
my_ram_out <= my_ram[r_addr];
dout <= my_ram_out;
end

endmodule
```

The following figure shows the schematic for a RAMB36E5.

Figure 8: **RAMB36E5**



Resets

Following are the types of resets on the block RAM:

- Synchronous reset on the block RAM output, which uses the RESETRAMA or RESETRAMB pin
- Asynchronous reset on the block RAM output, which uses the ARST_A or ARST_B pin

Note: If the ARST_A or ARST_B pin is used, the RESETRAMA, RESETRAMB, RSTREGA, and RSTREGB pins are ignored.

- Synchronous reset that controls the optional output registers of the block RAM, which uses the RSTREGA or RSTREGB pin

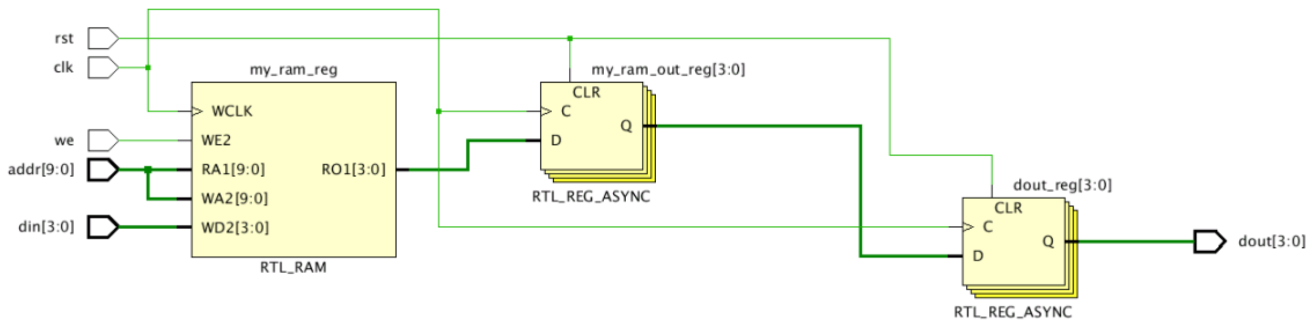
When using asynchronous resets:

- Both the RAM and optional output register must use the same asynchronous reset.

  Note: If the optional output register does not use the same reset, it is not inferred into the block RAM.
- The output enables and SRVAL properties are ignored.
- The asynchronous reset can only reset to a 0 value.

The following figure shows the schematic for a RAM that uses an asynchronous reset.

Figure 9: **RTL View of RAM Using an Asynchronous Reset**



Write Modes

The Versal ACAP block RAMs support the same write modes as UltraScale™ devices and use the same RTL coding styles:

- WRITE_FIRST outputs the newly written data onto the output bus.
- READ_FIRST outputs the previously stored data onto the output bus.
- NO_CHANGE maintains the previous value of the output bus.

Byte Write Enables

The control ports for byte write enables are the WEA and WEB pins, which vary based on usage:

- RAMB36E5
  - In non-SDP mode, the WEA and WEB [3:0] pins control 4 bytes of either size 8 or 9.
  - In SDP mode, the WEA and WEB [7:0] pins control 8 bytes of size 8 or 9.

- RAMB18E5
  - In non-SDP mode, the WEA and WEB [1:0] pins control 2 bytes of size 8 or 9.
  - In SDP mode, pins WEA and WEB [3:0] pins control 4 bytes of 8 or 9.

***Note*:** Size 9 names 8 bits with 1 parity bit.

Currently, Vivado synthesis only infers byte write RAM if sizes of 8 or 9 are used. In addition, Vivado synthesis only infers byte write enable RAM if the enables use one-hot state encoding. For example, in a byte write enabled RAM that is configured as true dual port with a data width of 36, there are 4 different bytes, but only 1 byte can be written to at a time. To infer the block RAM, make sure the RTL adheres to these restrictions.

Asymmetric RAMs

For asymmetric block RAMs in Versal ACAPs, use the same coding styles and rules that you use for asymmetric block RAMs in UltraScale devices. For information on setting up asymmetric block RAMs, see the *Vivado Design Suite User Guide: Synthesis* (UG901).

***Note*:** Currently, Vivado synthesis does not support asynchronous reset on asymmetric block RAMs in Versal ACAPs.

UltraRAM Primitives

Following is the UltraRAM primitive in Versal ACAPs. To instruct Vivado synthesis to infer the UltraRAM, you must set the `RAM_STYLE = "ultra"` attribute on the RAM.

***Note*:** Like UltraScale devices, the UltraRAM for Versal ACAPs includes only one clock.

Primitive	Supported Aspect Ratios	Supported Mode
URAM288E5	4Kx72 8Kx36 16Kx18 32Kx9	Dual port Single port

Extra Registers

In addition to the optional output registers, the UltraRAM supports input registers on the data lines. As with block RAMs, you can reset the optional registers either with synchronous or asynchronous reset signals.

RAM Initialization

You can initialize the UltraRAMs using the INIT_xx attribute on the RAM as follows:

- Verilog: Use the `readmemh` command.

- VHDL: Set up a function to read an external file in VHDL.

For details, see the *Vivado Design Suite User Guide: Synthesis* (UG901).

Byte Write Enables

The UltraRAM also supports byte write enable operations. As with block RAMs, the bytes can either be 8 bits or 9 bits using the extra parity bit. However, when using byte write with Versal ACAPs, read operations are ignored during writing. Therefore, only the NO_CHANGE mode is supported when describing UltraRAMs with byte write.

Asymmetric UltraRAMs

Versal ACAP UltraRAMs support asymmetric aspect ratios. However, Vivado synthesis does not currently infer UltraRAMs in asymmetric mode.

DSP Primitives

Following are the different types of DSP primitives for Versal ACAP.

Primitive	Description	Usage
DSP58	Standard integer/Fixed point mode	Inference or instantiation
DSPFP32	Floating point mode	Instantiation only
DSPCPLX	Complex multiplier	Inference or instantiation

DSP58

For Versal ACAPs, the DSP58 primitive includes the same features as in UltraScale devices, including a multiplier, adder, pre-adder, and registers to fully pipeline the primitive. However, sizing differs and the primitives include additional features.

Sizing

For signed logic, you can configure DSP58 as follows:

- Multiplier: 27x24
- Adder: 58-bit
- Pre-adder: 27-bit

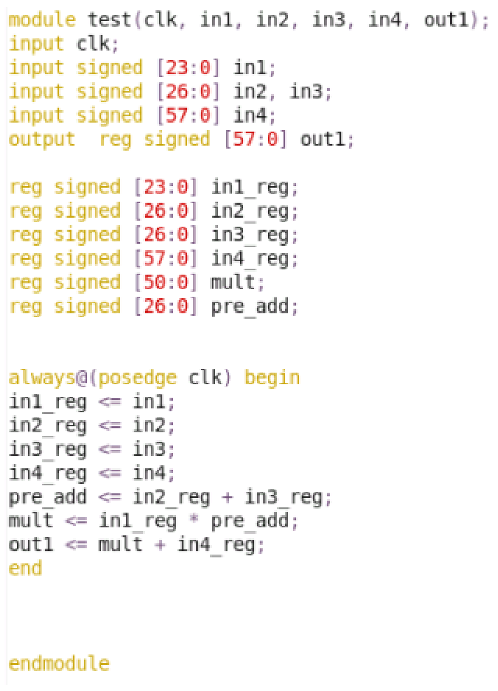
For unsigned logic, you can configure DSP58 as follows:

- Multiplier: 26x23
- Adder: 57-bit

- Pre-adder: 26-bit

The following figures show examples for signed logic.

Figure 10: **RTL for a 27x24 Multiplier with 58-Bit Adder and 27-Bit Pre-Adder**



```
module test(clk, in1, in2, in3, in4, out1);
input clk;
input signed [23:0] in1;
input signed [26:0] in2, in3;
input signed [57:0] in4;
output  reg signed [57:0] out1;

reg signed [23:0] in1_reg;
reg signed [26:0] in2_reg;
reg signed [26:0] in3_reg;
reg signed [57:0] in4_reg;
reg signed [50:0] mult;
reg signed [26:0] pre_add;


always@(posedge clk) begin
in1_reg <= in1;
in2_reg <= in2;
in3_reg <= in3;
in4_reg <= in4;
pre_add <= in2_reg + in3_reg;
mult <= in1_reg * pre_add;
out1 <= mult + in4_reg;
end



endmodule
```

Figure 11: **Elaborated View of 27x24 Multiplier with 58-Bit Adder and 27-Bit Pre-Adder**

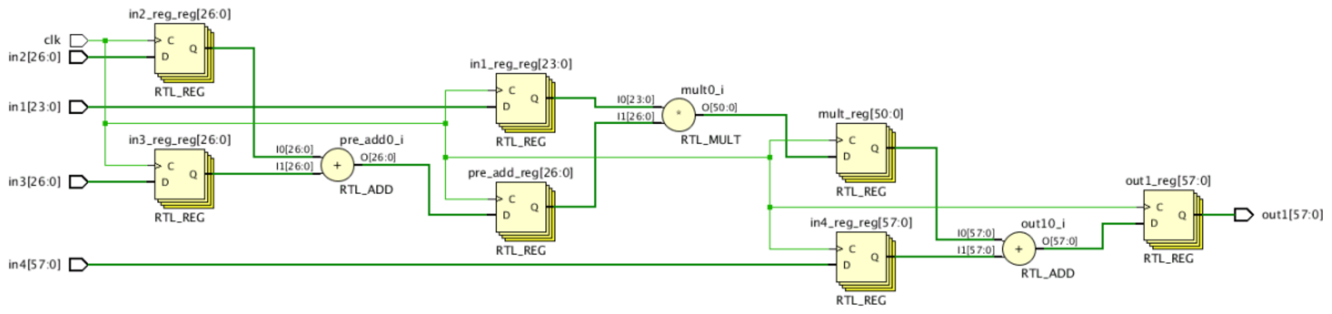
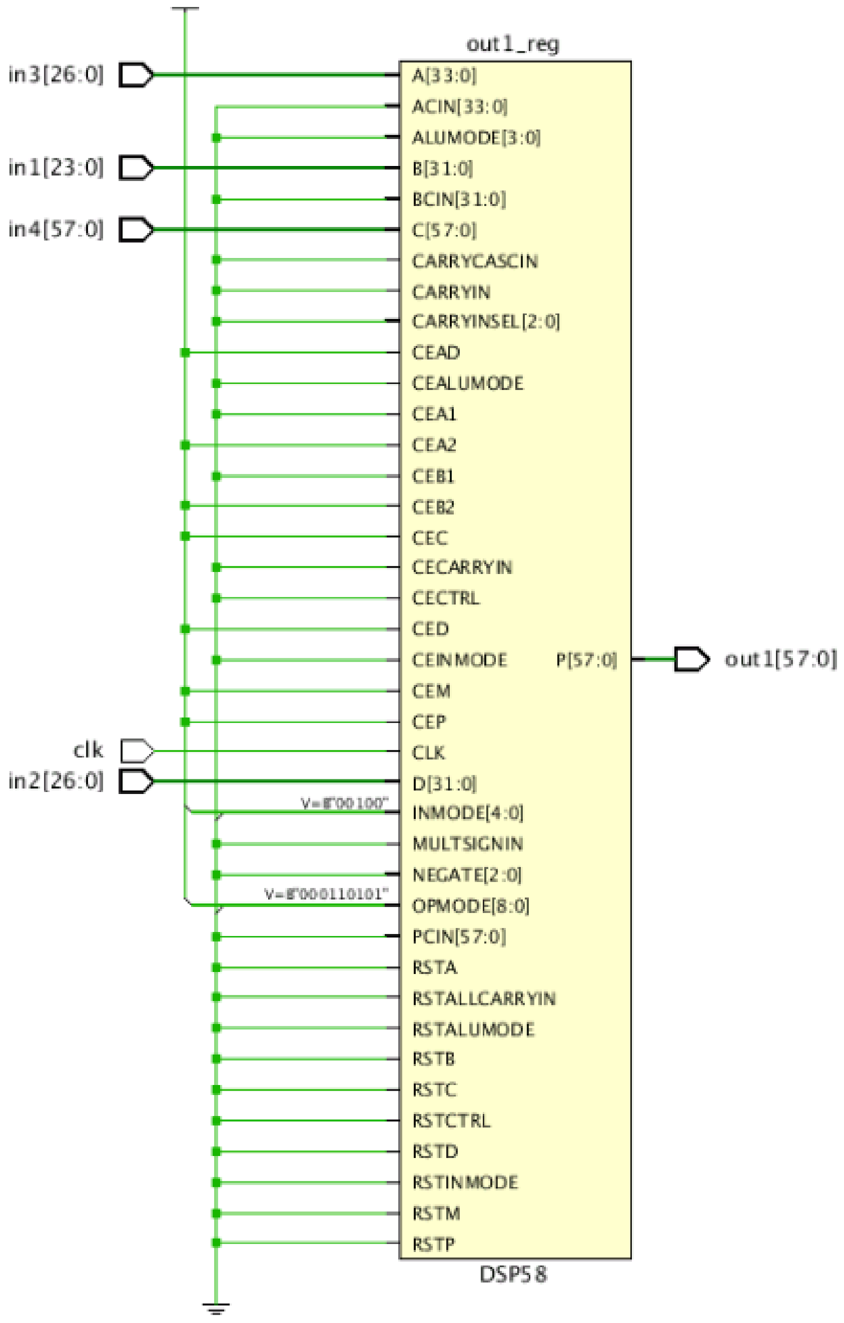


Figure 12: **Post-Synthesis View of 27x24 Multiplier with 58-Bit Adder and 27-Bit Pre-Adder**

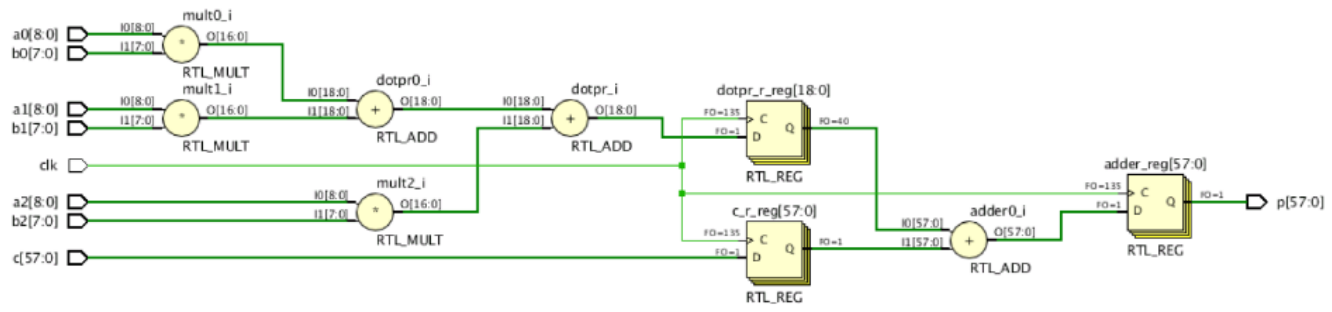


Dot Product

The DSP58 can implement a dot product, which is a multiplier that is represented as three smaller multipliers that are added together. Dot products are often used in filters in image processing. For more information, see the *Versal ACAP DSP Engine Architecture Manual* (AM004). The following figure shows an example of a dot product with an extra adder.

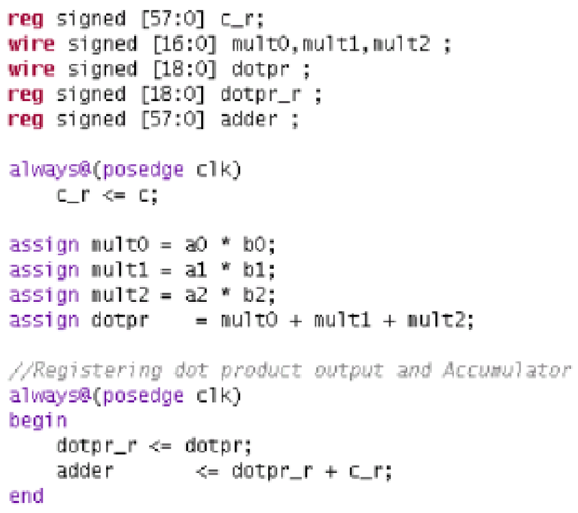
Note: For the dot product to infer, the RTL must use signed logic.

Figure 13: **Elaborated View of a Dot Product with an Extra Adder**



The following figure shows the RTL for a dot product.

Figure 14: **RTL for the Dot Product**



```
reg signed [57:0] c_r;
wire signed [16:0] mult0,mult1,mult2 ;
wire signed [18:0] dotpr ;
reg signed [18:0] dotpr_r ;
reg signed [57:0] adder ;

always@(posedge clk)
    c_r <= c;

assign mult0 = a0 * b0;
assign mult1 = a1 * b1;
assign mult2 = a2 * b2;
assign dotpr    = mult0 + mult1 + mult2;

//Registering dot product output and Accumulator
always@(posedge clk)
begin
    dotpr_r <= dotpr;
    adder       <= dotpr_r + c_r;
end
```

DSPFP32

DSPFP32 can perform floating point calculations. Vivado synthesis does not handle these calculations. Instead, various IP are provided, or the DSPFP32 primitive can be instantiated.

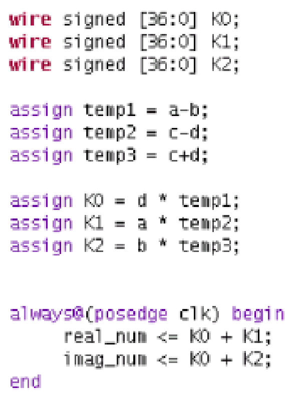
DSPCPLX

The DSPCPLX is designed to synthesize logic needed to solve for the real and imaginary portions of the following equation:

```
(a+bi)(c+di)
```

Each DSPCPLX occupies two DSP58 sites. The DSPCPLX can either be instantiated in the RTL or inferred. The following figure shows the RTL for the DSPCPLX.

Figure 15: **RTL to Synthesize the DSPCPLX**



```
wire signed [36:0] K0;
wire signed [36:0] K1;
wire signed [36:0] K2;

assign temp1 = a-b;
assign temp2 = c-d;
assign temp3 = c+d;

assign K0 = d * temp1;
assign K1 = a * temp2;
assign K2 = b * temp3;


always@(posedge clk) begin
     real_num <= K0 + K1;
     imag_num <= K0 + K2;
end
```

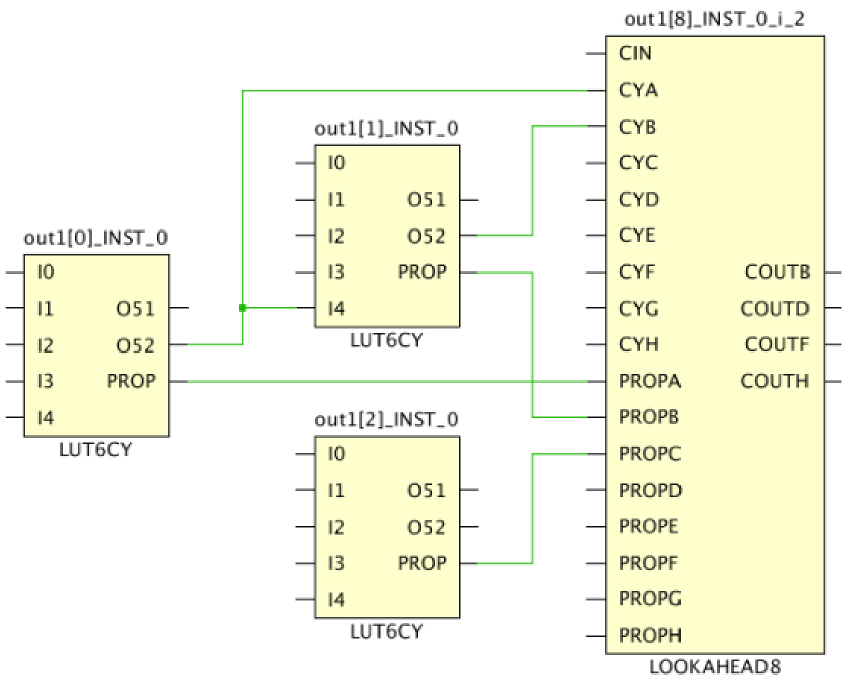
CLB Primitives

The configurable logic blocks (CLBs) in Versal ACAPs differ from those in UltraScale devices. Vivado synthesis sets up the CLBs to accurately map the RTL, but you must be aware of the differences noted in the following sections.

Carry Chains

Instead of the CARRY8 primitive in UltraScale devices, Versal ACAPs include a LOOKAHEAD8 primitive. The LOOKAHEAD8 primitive does not include MUXCYs and XORCYs for arithmetic operations. Instead, these operators must be inferred and as a result, the LUT count is slightly higher.

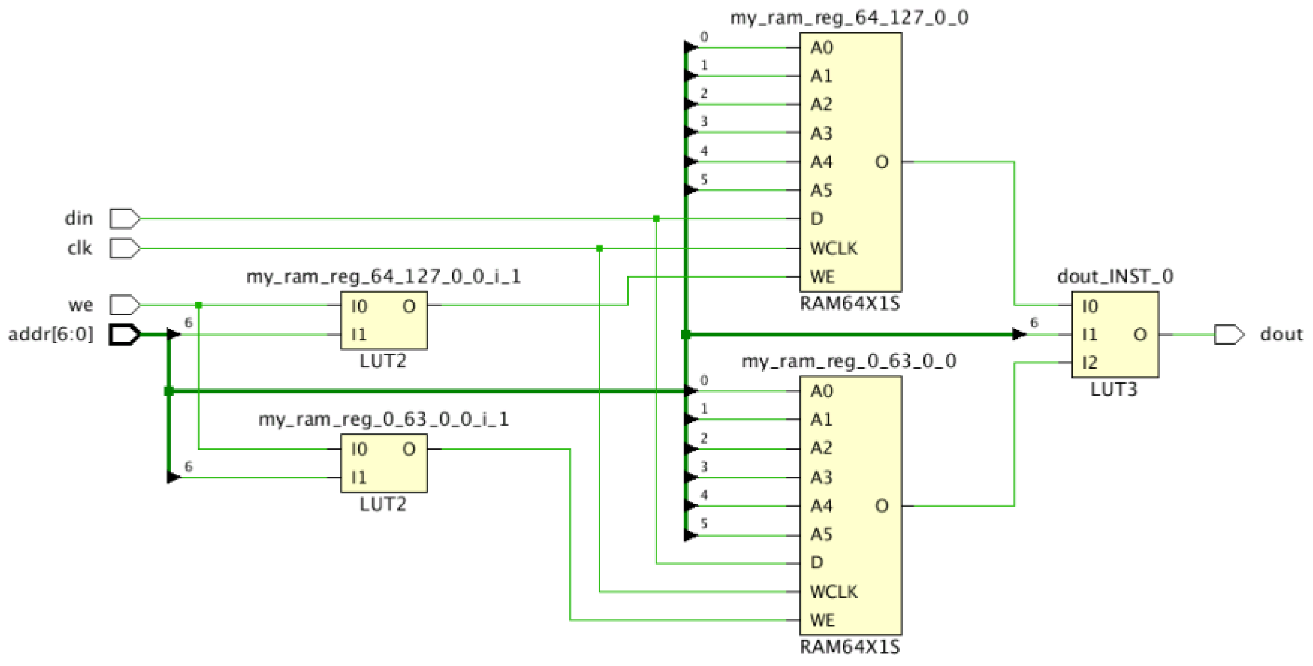
Figure 16: **Extra LUTs Before the CARRY Chain**



MUXFx Primitives

Versal ACAPs do not include MUXFx primitives. Because MUXFx primitives are often used for address decoding in distributed RAMs, large comparators, or MUX chains, expect extra LUT counts when using these types of structures in Versal ACAPs, as shown in the following figure.

Figure 17: **Extra LUTs for Address Decoding**



Additional Resources and Legal Notices

Xilinx Resources

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

Documentation Navigator and Design Hubs

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

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

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

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

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

References

These documents provide supplemental material useful with this guide:

1. *Versal Architecture and Product Data Sheet: Overview* (DS950)
2. *Versal ACAP GTY Transceivers Architecture Manual* (AM002)
3. *Versal ACAP Clocking Resources Architecture Manual* (AM003)
4. *Versal ACAP DSP Engine Architecture Manual* (AM004)
5. *Versal ACAP Configurable Logic Block Architecture Manual* (AM005)
6. *Versal ACAP System Monitor Architecture Manual* (AM006)
7. *Versal ACAP Memory Resources Architecture Manual* (AM007)
8. *Versal ACAP AI Engine Architecture Manual* (AM009)
9. *Versal ACAP SelectIO Resources Architecture Manual* (AM010)
10. *Versal ACAP Technical Reference Manual* (AM011)
11. *Versal ACAP Packaging and Pinouts Architecture Manual* (AM013)
12. *Versal ACAP CPM CCIX Architecture Manual* (AM016)
13. *SmartConnect LogiCORE IP Product Guide* (PG247)
14. *Versal ACAP Programmable Network on Chip and Integrated Memory Controller LogiCORE IP Product Guide* (PG313)
15. *Advanced I/O Wizard LogiCORE IP Product Guide* (PG320)
16. *Versal ACAP Integrated Block for PCI Express LogiCORE IP Product Guide* (PG343)
17. *Versal ACAP PCIe PHY LogiCORE IP Product Guide* (PG345)
18. *Versal ACAP CPM Mode for PCI Express Product Guide* (PG346)
19. *Vivado Design Suite User Guide: Logic Simulation* (UG900)
20. *Vivado Design Suite User Guide: Synthesis* (UG901)
21. *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* (UG994)
22. *Vivado Design Suite: AXI Reference Guide* (UG1037)
23. *Versal ACAP System Software Developers Guide* (UG1304)
24. *Vitis Unified Software Platform Documentation: Application Acceleration Development* (UG1393)
25. *Vitis High-Level Synthesis User Guide* (UG1399)
26. *XRT Release Notes* (UG1451)
27. RTL Kernels in the Application Acceleration Development flow of the *Vitis Unified Software Platform Documentation* (UG1416)
28. Vitis Unified Software Development Platform Documentation

Please Read: Important Legal Notices

AUTOMOTIVE APPLICATIONS DISCLAIMER

Copyright