

Versal ACAP Design Guide

UG1273 (v2020.2) March 26, 2021



Revision History

The following table shows the revision history for this document.

Section	Revision Summary
03/26/2021 Version 2020.2	
Chapter 3: System Planning	Moved content to <i>Versal ACAP System and Solution Planning Methodology Guide (UG1504)</i> .
System Design Types	Added new section.
Chapter 4: Design Flows	Updated with information on traditional and platform-based design flows.
02/04/2021 Version 2020.2	
Application Mapping and Design Partitioning	Added sections on design partitioning.
Debug Interfaces	Added new section.
Design Closure	Added details on design closure.
Logic Simulation Using SystemC Models	Added new section.
MRMAC	Added new section.
Security	Updated table.
12/04/2020 Version 2020.2	
NoC	Added information the NPI.
Chapter 4: Design Flows	Updated with information on the main design flows and Versal™ ACAP support for each.
Vivado Tools Design Flow	Updated information on the block design flow and RTL design flow.
I/O Planning	Added information on MIO and EMIO.
Power Closure	Added new section.
Simulation Flows	Moved to Vitis™ Environment Design Flow section.
Boot Image Generation	Added commands for PDI generation.
GT	Added information on block automation.
Power and Error Handling	Added information on power islands.
PCIe Subsystems	Updated migration information.
Boot and Configuration	Moved to System Migration chapter.
Clocking	Added new section.
Appendix A: Primitives	Added VHDL examples.
Block RAM Primitives	Added information on XPM inference.
UltraRAM Primitives	Added information on XPM inference.
Coding Style and Primitive Instantiation Examples	Added new section.
07/14/2020 Version 2020.1	
Initial release	N/A

Table of Contents

Revision History	2
Chapter 1: Overview	5
Introduction to Versal ACAP.....	5
Navigating Content by Design Process.....	6
About This Guide.....	7
Chapter 2: System Architecture	9
AI Engine.....	10
Programmable Logic.....	11
NoC.....	12
XPIO.....	13
DDRMC.....	13
CIPS.....	14
GT.....	20
HSDP.....	20
MRMAC.....	21
Chapter 3: System Planning	22
System Design Types.....	22
Chapter 4: Design Flows	25
Traditional Design Flows.....	25
Platform-Based Design Flows.....	27
Vivado Tools Design Flow.....	27
Vitis Environment Design Flow.....	38
Boot Image Generation.....	49
Chapter 5: System Migration	50
CLB.....	51
On-Chip Memory Resources.....	52
DSP.....	52
Clocking.....	53

I/O.....	54
Soft Memory Controllers.....	54
AXI Interconnect.....	55
GT.....	55
PCIe Subsystems.....	56
MRMAC.....	58
Processor and Peripherals.....	58
System Debug.....	59
System Monitor.....	60
Power and Error Handling.....	60
Security.....	60
Boot and Configuration.....	61
PL Configuration and JTAG.....	62
Appendix A: Primitives.....	64
RAM Primitives.....	64
DSP Primitives.....	68
CLB Primitives.....	73
Coding Style and Primitive Instantiation Examples.....	75
Appendix B: Additional Resources and Legal Notices.....	76
Xilinx Resources.....	76
Documentation Navigator and Design Hubs.....	76
References.....	76
Please Read: Important Legal Notices.....	79

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. All Versal™ ACAP design process [Design Hubs](#) can be found on the Xilinx.com website. 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 Planning](#)
 - [Chapter 4: Design Flows](#)
 - [Chapter 5: System Migration](#)

Note: For more information, see the *Versal ACAP System and Solution Planning Methodology Guide (UG1504)*.

- **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](#)

Note: For more information, see the [Programming the PS Host Application](#) in the AI Engine Documentation flow of the *Vitis Unified Software Platform Documentation (UG1416)*.

- **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](#)

Note: For more information, see the *Versal ACAP AI Engine Programming Environment User Guide (UG1076)* and *Versal ACAP AI Engine Kernel Coding User Guide (UG1079)*.

- **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](#)

Note: For more information, see the *Versal ACAP Hardware, IP, and Platform Development Methodology Guide (UG1387)*.

- **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:

- [Chapter 4: Design Flows](#)

Note: For more information, see the *Versal ACAP System Integration and Validation Methodology Guide (UG1388)*.

- **Board System Design:** Designing a PCB through schematics and board layout. Also involves power, thermal, and signal integrity considerations. Topics in this document that apply to this design process include:

- [Chapter 3: System Planning](#)

Note: For more information, see the *Versal ACAP Board System Design Methodology Guide (UG1506)*.

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 Planning:** Describes how each Versal device series relates to different system design types and design flows.
- **Chapter 4: Design Flows:** 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

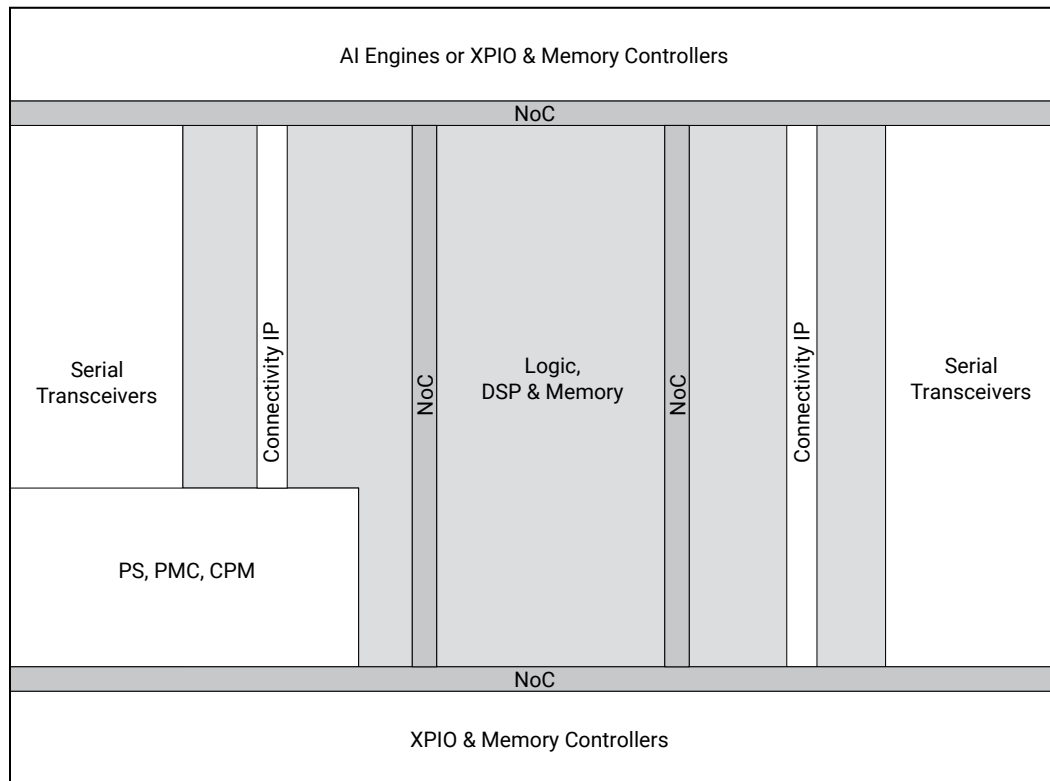
The Xilinx[®] Versal[™] ACAP is a collection of programmable resources that work together to form a system on chip (SoC). Following are the major resource blocks:

- 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)
- Multirate Ethernet MAC (MRMAC)

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



X22326-062920

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 (4 MB) (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 network on chip (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 NoC must be configured or programmed from the NoC programming interface (NPI) at early boot and before the NoC data paths are used. The NPI programs NoC registers that define the routing table, rate modulation, and QoS configuration. Programming of the NoC from the NPI normally requires no user intervention. Programming is fully automated and executed by the platform management controller (PMC)-embedded NPI controller. For more information about boot and configuration, see the *Versal ACAP Technical Reference Manual* ([AM011](#)).

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 memory 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 memory 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. The optimal combination of memory interfaces with various width, type, and speed can be identified by using the *Versal ACAP External Memory Pre-Planning Tool* ([XTP667](#)).

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 *Versal ACAP Programmable Network on Chip and Integrated Memory Controller LogiCORE IP Product Guide* ([PG313](#)).

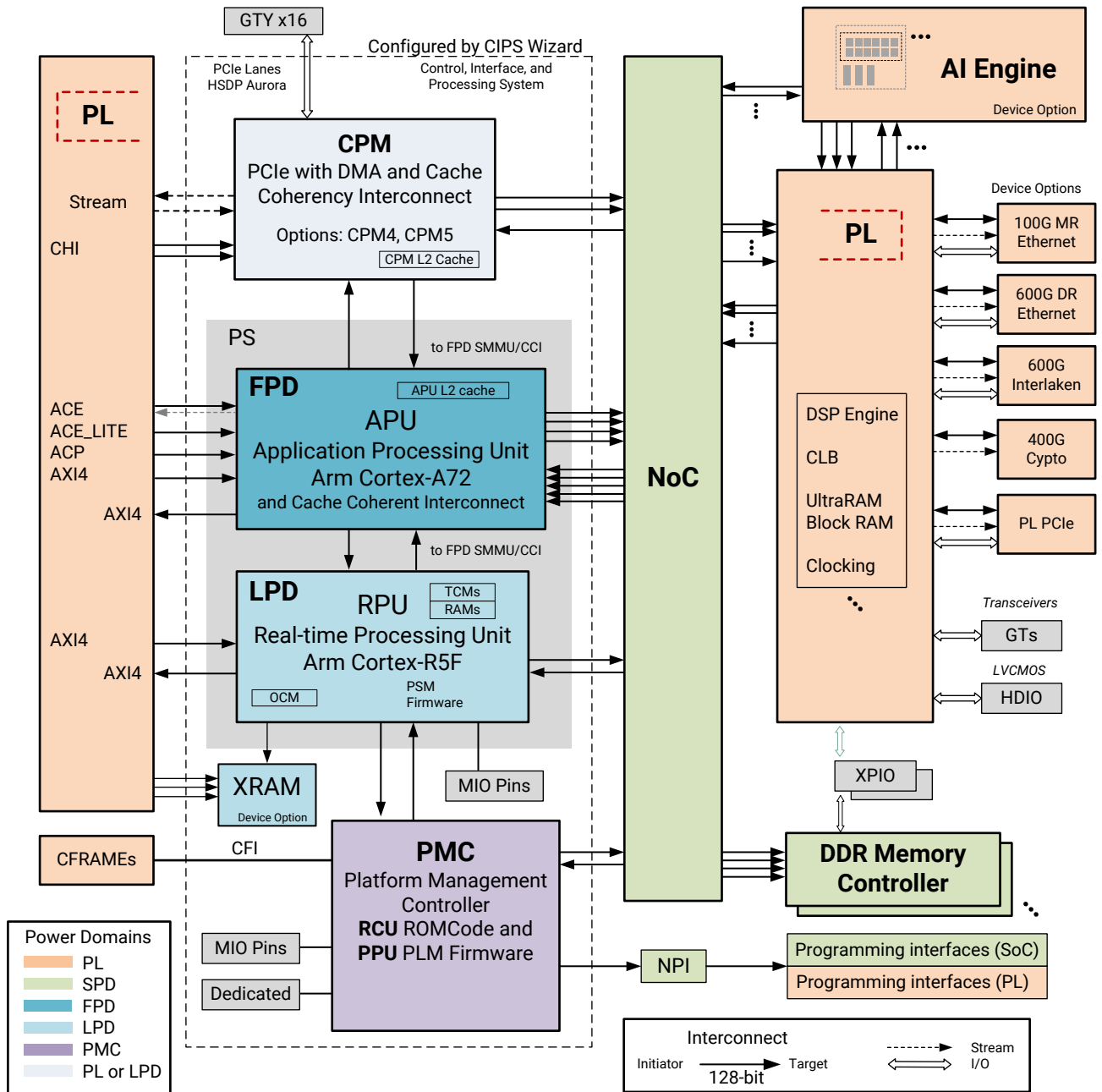
Note: Versal ACAP also supports soft memory controllers in the PL fabric, similar to previous device families.

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, the RPU is 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



X21692-111320

PS

The processing system (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 application processing unit (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
ELO-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	6.84 SPEC2006int	
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 real-time processing unit (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 memory 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 (host, device, dual-role device)	2 USB 3.0/2.0 controllers	1 USB 2.0 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_HP _x _FPD	4	S_AXI_HP	1
PL-to-FPD AXI	PL	I/O	S_AXI_HPC _x _FPD	2	S_AXI_HPC	1
PL-to-LPD AXI	PL	-	S_AXI_LPD	1	S_AXI_LPD	1

Table 3: AMBA Interface Comparison (cont'd)

PS-PL AMBA Interface	Master	Coherency	Zynq UltraScale+ MPSoC		Versal ACAP	
			Name	Count	Name	Count
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 platform management controller (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](#)), *Versal ACAP CPM Mode for PCI Express Product Guide* ([PG346](#)), and *Versal ACAP CPM DMA and Bridge Mode for PCI Express Product Guide* ([PG347](#)).

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 and GTYP Transceivers Architecture Manual* ([AM002](#)).

HSDP

The heterogeneous nature and performance 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.

MRMAC

The Multirate Ethernet MAC (MRMAC) provides high-performance, low latency Ethernet ports supporting a wide range of customization and statistics gathering. The MRMAC supports the following forward error corrections (FECs) defined and required by IEEE standards: Clause 91 RS(528, 514) KR4 FEC for 25/50/100GE NRZ support, Clause 91 RS(544, 514) KP4 FEC for 50/100GE PAM4 support, and Clause 74 FEC, for 10/25/40/50GE low-latency support. The MRMAC has a rich set of bypass modes to enable access to FEC-only mode (for custom protocols) and FEC+PCS (for protocol testers).

System Planning

To properly plan your design, you must understand the system requirements based on your target application or system design type. This includes identifying the appropriate Versal™ device with the correct features (e.g., the number of DDRMC IP, AI Engines, etc.).

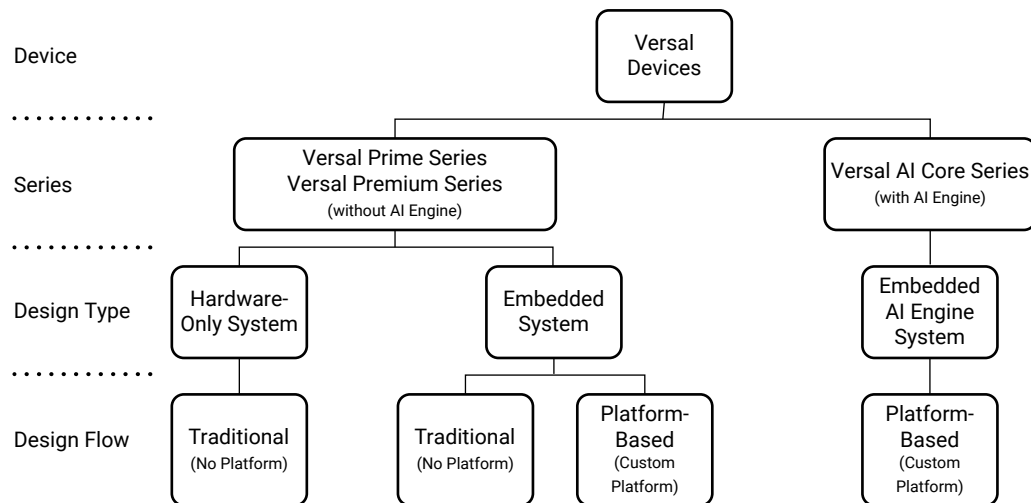
Note: For information on the recommended system planning methodology, see the *Versal ACAP System and Solution Planning Methodology Guide* ([UG1504](#)).

System Design Types

Versal™ ACAP is a heterogeneous compute platform with multiple compute engines. A wide range of applications can be mapped on Versal ACAP, including signal processing for wireless systems, machine learning inference, and video processing algorithms. In addition to multiple compute engines, Versal ACAP offers very high system bandwidth using high-speed serial I/Os, network on chip (NoC), DDR4/LPDDR4 memory controllers, and multi-rate Ethernet Media Access Controllers (MRMACs). Versal devices are categorized into the Versal Prime, Premium, and AI Core series. The following figure shows the different system design types and design flows supported for each Versal device series.

Note: The design flows for Versal Prime and Premium series are similar to the flows used with Xilinx® FPGAs. The design flow for Versal AI Core series requires that you design for a heterogeneous compute platform, which has special hardware configuration and software support requirements.

Figure 3: System Design Types



X25009-032321

The following table shows the system design types and design flows supported for each Versal device series. As shown in the table, a majority of the design flows are based on building a platform.

Table 6: System Design Types

Design Type	Device Series	Design Flow	Platform Source	GitHub Examples
Hardware-only system	Versal Prime Series Versal Premium Series	Traditional	N/A	Versal Device Architecture Tutorials
Embedded system	Versal Prime Series Versal Premium Series	Traditional	N/A	Versal Embedded Design Tutorial
		Platform-based	Custom	Versal Prime Series VMK180 Targeted Reference Designs
Embedded AI Engine system	Versal AI Core series	Platform-based	Custom	AI Engine Development Design Tutorials VCK190 Base TRD



TIP: Check [Xilinx GitHub](#) for additional examples, which are updated periodically.

Following is a summary of each system design type:

- **Hardware-only system:** programmable logic designs. Create this system using the traditional design flow.
- **Embedded system:** embedded processing system with software running on the Arm[®] Cortex[®]-A72 and Cortex-R5F processors and hardware content in the PL. Create this system using either the traditional or platform-based design flow.

- **Embedded AI Engine system:** embedded processing system with software running on the Arm Cortex-A72 and Cortex-R5F processors, hardware content in the PL, and algorithmic content in the AI Engine. Create this system using either the traditional or platform-based design flow.

Following is a summary of the platform sources used for each design flow:

- **Traditional design flow (no platform):** When using the traditional design flow, a platform is not needed because acceleration is not performed. The Vitis™ embedded software is used for embedded tools, such as drivers, run time and multi-OS environments, compilers, debuggers, and profiling tools. However, the Vitis software platform is not required.
- **Platform-based design flow (custom platform):** When using the platform-based design flow, a custom platform must be used when developing accelerated applications on the PL or AI Engine on a custom board. The Vitis software platform is required, and the XSA is built for the specific custom board.

Design Flows

There are two design flows for Versal™ ACAP: the traditional design flow and the platform-based design flow. To use Versal ACAP resources to their full potential, it is important to choose the correct design flow. The following table shows which design flow to use based on the design type and targeted device series.

Table 7: Design Flows

Design Type	Device Series	Design Flow
Hardware-only system	Versal Prime Series Versal Premium Series	Traditional
Embedded system	Versal Prime Series Versal Premium Series	Traditional
		Platform-based
Embedded AI Engine system	Versal AI Core series	Platform-based

Note: For more information on design types, see the *Versal ACAP System and Solution Planning Methodology Guide* ([UG1504](#)).

Traditional Design Flows

Traditional Design Flow for Hardware-Only Systems

If your design consists of PL components only (RTL and IP only), you can use the Vivado® tools to generate a programmable device image (PDI) to program the Versal device. Like previous architectures, design sources are added to the Vivado tools and compiled through the Vivado implementation flow. If you are using this flow, following are important considerations:

- The PMC is incorporated into the CIPS IP and must be configured for the Versal device to boot properly. Therefore, all Versal device designs must include CIPS IP.
- The hardened DDR memory controllers are only accessible through the NoC IP. To use the DDRMC, your design must include NoC IP.
- Hardware debug cores connect through the CIPS IP by default. JTAG is still available but no longer the preferred flow. You must be familiar with changes to the hardware debug connectivity and flow.

Xilinx recommends using the Vivado IP integrator to instantiate, configure, and connect the CIPS IP, the NoC/DDRMC IP, and hardware debug IP to take advantage of block design automation when iterating through design changes. The Vivado IP integrator also provides special support for GT IP and connectivity IP (such as MRMAC IP), which simplifies GT-based design creation and I/O planning.

You can integrate the complete design with the Vivado IP integrator using custom packaged IP, RTL module referenced blocks, and other IP available through the IP catalog. Alternatively, you can use the Vivado IP integrator to configure and connect critical Versal ACAP IP (such as the CIPS IP and the NoC/DDR IP) and then instantiate the resulting block design in the RTL design. For more information, see this [link](#) in the *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator (UG994)*.



IMPORTANT! *This design flow does not support programming of the AI Engine cores and is therefore only suitable for Versal Prime and Versal Premium devices.*

Related Information

[System Debug](#)

Traditional Design Flow for Embedded Systems

You can also use the traditional design flow to create designs with both PL and embedded software components. In this case, the flow is similar to the embedded software design flow used for Zynq® UltraScale+™ MPSoCs. The hardware team is responsible for creating, verifying, and implementing a hardware design that is used by the software team to develop the embedded software application. This hardware design is considered a *fixed* platform, because the Vivado tools create the programmable logic on the device, and the Vitis™ tools cannot modify the PL.

Note: All recommendations for the traditional design flow for hardware-only systems apply to the traditional design flow for embedded systems.

Following are the main steps in this flow:

1. Create and verify the hardware design using the Vivado IP integrator.
2. Implement the hardware design using the Vivado implementation tools.
3. Export the hardware design to the Vitis embedded software development flow.
4. Develop the software application on top of the fixed hardware design using the Vitis embedded software development flow.



IMPORTANT! *This design flow does not support programming of the AI Engine cores and is therefore only suitable for Versal Prime and Versal Premium devices.*

Platform-Based Design Flows

In the platform-based design flow, the hardware design is conceptually divided in two distinct elements: a platform and the processing subsystem. The platform contains essential Versal IP blocks (including CIPS, NoC, AI Engine, and Clocking Wizard) and board interface IP blocks (including high-speed I/Os and memory controllers). The processing subsystem contains the application-specific part of the system and can be composed of both programmable logic and AI Engine blocks. The platform is considered *extensible*, because the platform does not contain the entirety of the programmable logic content. Instead, the platform is extended by the addition of the processing subsystem.

Following are the main steps in this flow:

1. Create the hardware platform using the Vivado IP integrator and RTL code.
2. Create the processing subsystem using the Vitis tools.
3. Integrate the acceleration subsystem with the platform using the Vitis linker to create a fixed hardware design that is implemented using the Vivado tools.
4. Develop the software application on top of the fixed hardware design using the Vitis embedded software development flow.
5. When using the Versal AI Core series, develop the AI Engine program as part of the processing subsystem.



IMPORTANT! This is the only flow that supports programming of the AI Engine cores and is therefore required for Versal AI Core devices.



TIP: Xilinx provides off-the-shelf platforms for Versal ACAP evaluation kits, such as the VCK190.

Related Information

[Vivado Tools Design Flow](#)
[Subsystem Design Flow](#)

Vivado Tools Design Flow

The Vivado Design Suite is a key component in all Versal ACAP design flows. Following are the primary use models for the Vivado tools based on your design flow:

- Traditional design flows
 - Creating RTL and IP designs

- Platform-based design flows
 - Creating and packaging RTL kernels for use in the Vitis environment design flow
 - Creating and generating platforms for use in the Vitis environment design flow



IMPORTANT! *If you are using the platform-based design flow, 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. For more information, see the Vitis Embedded Platforms tab of the [Downloads](#) page on the Xilinx website.*

You can use the Vivado tools for design creation, implementation, and analysis of PL. Typical tasks include the following:

- Logic simulation
- Constraint definition and timing analysis
- NoC compilation
- I/O and clock planning
- Logic synthesis and implementation
- Visualization of design logic
- Design rule checks (DRC) and design methodology checks
- Implementation results analysis
- Power and thermal analysis
- Programming and debugging

Creating RTL and IP Designs

The Vivado tools support the traditional RTL and IP design flow, and the Vivado IP integrator is available to automate the assembly of your design. RTL developers must understand the new IP available in Versal ACAP and the requirements surrounding their usage, including the following:

- All designs require the CIPS IP, which contains the PMC used to boot the device. CIPS IP is also used to configure the PS peripherals and the SYSMON IP. For more information, see the *Control, Interface and Processing System LogiCORE IP Product Guide* ([PG352](#)).
- The only way to access the DDRMCs on the device is through the NoC IP. For more information, see the *Versal ACAP Programmable Network on Chip and Integrated Memory Controller LogiCORE IP Product Guide* ([PG313](#)).
- Hardware debug flows are different from previous devices. For more information, see the *Vivado Design Suite User Guide: Programming and Debugging* ([UG908](#)).

Packaging RTL Kernels

You can use the Vivado tools to package RTL kernels for use by the Vitis linker. This option is available in the Vivado IP packager, which packages the IP into an XO file to be linked into the final design using the Vitis system linker. For more information about RTL kernels, including restrictions, see [RTL Kernels](#) in the Application Acceleration Development flow of the *Vitis Unified Software Platform Documentation* (UG1416).



RECOMMENDED: Xilinx recommends that RTL developers use this approach to incorporate existing logic when using the Vitis environment design flows.

Generating Platforms

You can create an extensible hardware platform using the Vivado tools that is then extended with a processing subsystem using the Vitis tools. The platform typically includes basic system-level resources that are shared by all accelerators, such as the PS, NoC, DDRMCs, and primary I/Os. For more information on hardware platform definition, see the [Vitis Accelerated Software Development Flow Documentation](#) in the Application Acceleration Development flow of the *Vitis Unified Software Platform Documentation* (UG1416).

Note: This is the only design flow that supports the use of AI Engine resources.

Xilinx recommends the following:

- Include only essential Versal ACAP blocks and board interface IP in the platform
 - Essential blocks: CIPS, NoC, AI Engine, Clock wizard, interrupt controller
 - Interface blocks: High-speed I/Os, memory controllers
- Keep application-specific blocks outside the platform as kernels in the Vitis subsystem (e.g., fast Fourier transform (FFT), filters, etc.)

Following are the benefits of this approach:

- Ensures the platform is highly reusable
- Promotes separation of tasks
- Improves the ability to automate the integration process
- Increases scope and opportunity for DFX

Block Design Flow

Xilinx highly recommends using the Vivado IP integrator cockpit for designs that target Versal ACAPs. The Vivado IP integrator is a graphical and Tcl-based tool that allows you to combine various Xilinx® and user-packaged IP-based subsystems into the overall design. This allows you to 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. The Vivado IP integrator also provides special support for GT IP and connectivity IP (such as MRMAC IP), which simplifies GT-based design creation and I/O planning.

For Versal devices, IP integrator facilitates the integration of designs partitioned for different domains (PS/PL/AI Engine). For example, you can create a hardware platform in the PL domain, which contains various blocks that perform computation and interface to the PS domain, external memory, and I/O. This hardware platform can also be connected to an AI Engine block.

Following are the advantages of using IP integrator:

- Allows automatic configuration updates between Versal device-specific blocks.
- Allows automatic connectivity between various blocks, which prevents errors.
- Provides seamless interaction with the Vitis tools, allowing export of custom hardware platforms.

You can use an IP integrator block design (BD) in the following ways:

- Sub-module as part of a design
- Top-level of the design hierarchy



RECOMMENDED: For design that target Versal devices, Xilinx recommends using the IP integrator BD as the top-level of the design rather than as a sub-module within an RTL-based top-level. This flow helps in both creation and validation of the Versal ACAP design.

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).

CIPS IP Core

The CIPS IP allows you to configure the following:

- Device clocking to the PMC, PS, NoC, and optionally, PL
- PMC flash controllers, peripherals, and their associated multiplexed I/O (MIO)
- 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

AI Engine IP

To generate an extensible platform for the Vitis environment, the AI Engine IP must be instantiated and connected to the rest of the design. Then, the Vitis environment must be used to generate the AI Engine configuration. The AI Engine IP 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 extensible platform creation.

For more information, see the *AI Engine LogiCORE IP Product Guide* ([PG358](#)) and *Versal ACAP AI Engine Programming Environment User Guide* ([UG1076](#)).

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](#)).

Transceivers Bridge

The Versal ACAP transceivers are highly configurable and tightly integrated with the PL block. The Versal ACAP 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 Vivado Design Suite I/O planning tools to add physical GT locations. For more information, see the *Versal ACAP Transceivers Wizard LogiCORE IP Product Guide* (PG331).

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

You can use the RTL design flow to create modules, instantiate IP, or assemble the top-level design, similar to previous architectures. However, you must follow Xilinx recommendations for using Versal device-specific blocks in the RTL design flow, including the CIPS and NoC IP. The CIPS IP provides access to device configuration features, and the NoC IP connects PL to one or several DDRMC hardened IP.

Xilinx highly recommends using the Vivado IP integrator to instantiate and configure the CIPS and NoC IP. However, you do not need to use the IP integrator for your entire design. You can configure the CIPS and NoC in IP integrator, specify the interface to the rest of the design, and instantiate the resulting block design in the top-level RTL. Using this approach, IP integrator automates the CIPS and NoC configuration, allowing you apply additional changes as needed.


I/O Planning

For Versal ACAP, the I/O planning flow for high-performance I/O (XPIO) differs from previous architectures, as described in the following sections. The I/O planning flow for low-performance I/O, also known as high-density I/O (HD I/O), remains the same.

High-Performance I/O

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 XPIO, see the *Versal ACAP SelectIO Resources Architecture Manual (AM010)*. 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 HD I/O 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 8: 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 <ul style="list-style-type: none"> High-performance interfaces Calibrated IDELAY, ODELAY, ISERDES, OSERDES, IDDR, and ODDR 	Versal Advanced I/O Wizard
UltraScale Component Mode <ul style="list-style-type: none"> 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.

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 soft memory controller rules, see the *Versal ACAP Soft DDR4 SDRAM Memory Controller LogiCORE IP Product Guide* ([PG353](#)) and *Versal ACAP Soft RLDRAM 3 Memory Controller LogiCORE IP Product Guide* ([PG354](#)).
- For information on the Advanced I/O wizard, see the *Advanced I/O Wizard LogiCORE IP Product Guide* ([PG320](#)).

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. For the voltage limits for specific banks, see the data sheet for your device.*

Multiplexed I/O

The Versal ACAP multiplexed I/O (MIO) are similar to the MIO on the Zynq UltraScale+ MPSoCs. In Versal devices, there are 78 MIO pins, 52 signals in the PMC MIO (banks 500 and 501), and 26 signals in the LPD MIO (bank 502). For details on MIO pin planning, see this [link](#) and this [link](#) in the *Versal ACAP Technical Reference Manual* ([AM011](#)). The Versal ACAP Control, Interfaces, and Processing System (CIPS) IP is used to select the MIOs to use and specify their functionality.

Extended Multiplexed I/O

Certain PMC and LPD peripherals can be routed via the extended multiplexed I/O (EMIO) interface through the PL to XPIO or HD I/O via I/O logic. For details on which peripherals can access the EMIO, see this [link](#) in the *Versal ACAP Technical Reference Manual* ([AM011](#)). The IO Configuration page in the CIPS IP is used to select which peripherals access the EMIO. Because EMIO uses I/O logic, the pin planning of the EMIO is completed using the traditional drag-and-drop pin planning in the Vivado Design Suite.



POWER TIP: When pin planning the EMIO in the XPIO or HD I/O, consider the I/O voltage requirements of the peripheral interface in choosing the bank type. XPIO banks can support I/O voltages of 1.5V and below while HD I/O banks can support I/O voltages of 1.8V and above. For XPIO banks, the recommendation is to place the lower-speed I/O logic last to maximize package pin utilization, and this might leave a limited selection of I/O standards based on previously assigned I/O. You must ensure that your interface is feasible in the desired bank or banks. You can use the IBIS models to simulate the interface at your required speed with the I/O standard selected for your bank.

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 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](#)).

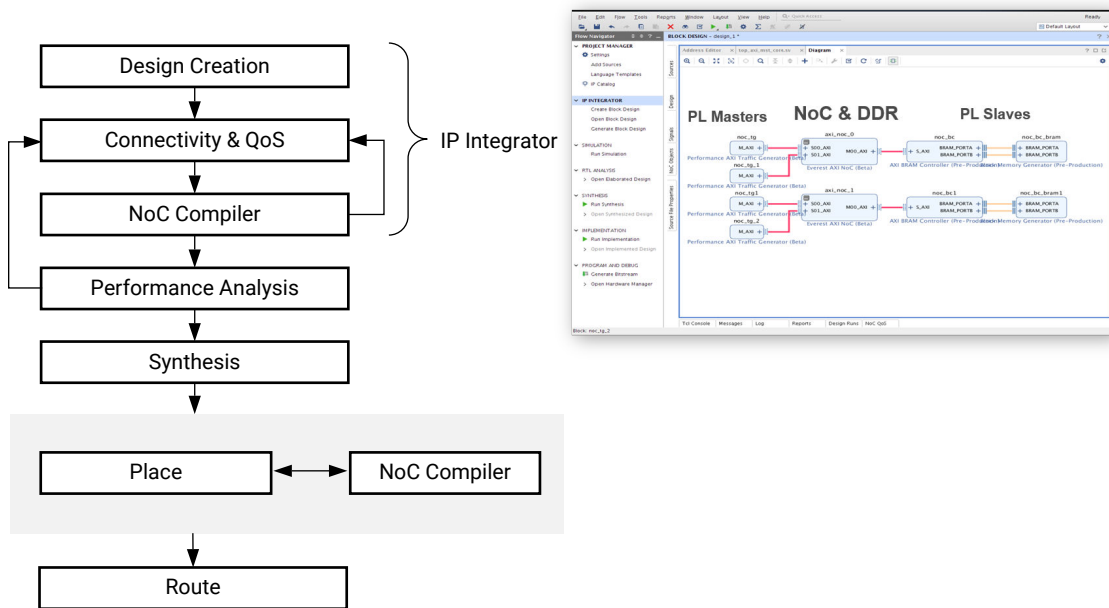
Note: Logic simulation is possible in both the traditional design flow and the platform-based design flow.

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 4: NoC Compiler Flow



X21272-022321

Related Information

[Primitives](#)

Power Closure

To achieve power closure, run the following steps.

Apply Constraints and Implement Design

After the board design is fixed, you must ensure that the design stays within the constraints identified for power, power delivery, timing, and pinout to ensure the design meets your requirements. At a minimum, Xilinx recommends constraining the design for total power and providing maximum ambient and Theta Ja for the most accurate estimation. Xilinx allows you to constrain both the device power and the power delivery solution by combining the power supplies that are connected to the same regulator and specifying the maximum current for the regulator. You can use `report_power` to check whether any supplies are overutilized. For more information, see the *Vivado Design Suite User Guide: Power Analysis and Optimization (UG907)*.

Following are examples of device power constraints:

```
set_operating_conditions -process maximum
set_operating_conditions -ambient_temp 40
set_operating_conditions -thetaja 1.5
set_operating_conditions -design_power_budget 10.2
```

Report Power

Run the `report_power` command on an implemented design to ensure that the power constraints applied to the design reflect the initial power estimation and thermal design. The `report_power` command also shows any margin based on the constraints, ensuring the power is kept in check. When you are generating multiple runs to close timing, Xilinx recommends running `report_power`, because often the design with the most slack might not be the best from a power perspective. However, as long as there is slack, a design can be considered to have closed timing, and having the report power results for every run allows you to select the best implementation for power as well.



POWER TIP: *If your design exceeds the power constraints, a quick way to run a “what if” analysis is to export the results to XPE and make adjustments to see the impact on power.*

Check Whether Design is Within Budget

At this point, you must evaluate the constraints. If power is too high, try the following:

- Reduce power in the design. This approach is the least costly in terms of time. For information, see this [link](#) in the *Vivado Design Suite User Guide: Power Analysis and Optimization (UG907)*.
- Improve the thermal solution within the thermal design constraints. For example, change the heat sink composition to improve the thermal solution, such as adding heat pipes or forced air.
- Improve the board and thermal solution, if possible. However, be aware that thermal changes might increase the power in the design.
- Reduce the complexity or features in the design. Use this approach as a last resort. For example, you can reduce power by reducing clock frequencies, resources used, or toggle rate. You can use XPE for “what if” analysis to quickly estimate the impact of design changes.

Design Closure

The Versal architecture introduces new hardware features that require additional considerations to reach design closure, including timing and performance closure. Similarly to previous Xilinx device architectures, the timing summary report is the signoff report for timing closure. Vivado Design Suite compilation tools provide guidance via the following reports:

- Design rule checks prevent invalid hardware configurations (`report_drc`). Any such issue prevents the device image file generation and must be addressed.

- Methodology checks improve the PL maximum operating frequency and identify common unsafe design structures, which can lead to hardware malfunction or instability (`report_methodology`, `report_cdc`). Critical and warning violations must be addressed to help timing closure and hardware stability.
- Xilinx also recommends addressing critical warnings in the log files.



IMPORTANT! To reduce timing closure iterations, you must review and address the timing violations as early as possible in the implementation flow, especially after synthesis and after placement.

Due to the heterogeneous nature of the Versal architecture, the design performance mostly depends on the NoC QoS, DDR memory access, and software efficiency in the PS and AI Engines in addition to the PL operating frequency and amount of pipelining. For information on timing, system performance, and power design closure, see the *Versal ACAP System Integration and Validation Methodology Guide* ([UG1388](#)).

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 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 Platform Documentation* ([UG1416](#)).

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.

Note: Model Composer is also available for users familiar with MATLAB® software. For more information, see the *Model Composer and System Generator User Guide* ([UG1483](#)).

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, 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 Design Flow

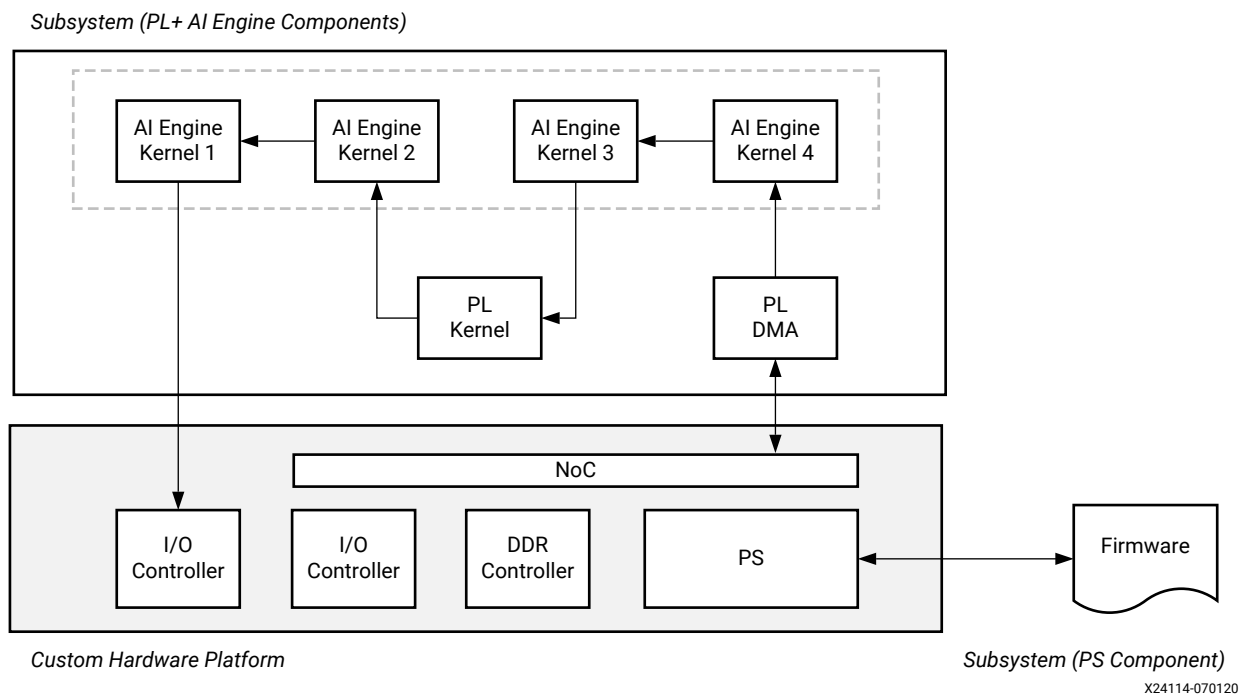
Prior to starting development, you must choose the Versal device that is best suited for your application and partition the design into functions targeted to the PS, AI Engine, and PL, depending on the application requirements. At this point, you must have an understanding of the following:

- System design considerations, such as throughput and latency
- Domain and inter-domain capabilities, including compute and bandwidth
- Dataflow and control flow 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 dataflow 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 the Vitis HLS co-simulation flow.
5. Compile the RTL implementation into a compiled object file (.xo), or export to an RTL IP.

For more information, see the [Vitis HLS Documentation](#) in the Application Acceleration Development flow of the *Vitis Unified Software Platform Documentation* (UG1416).

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 (aiesimulator). 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 (`libadf.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, DWCs, 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 Xilinx Runtime (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.

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. In addition, Xilinx also provides the ability to co-simulate a complete system comprised of PL, PS and optionally AI Engine components. The following sections provide details on the scope and purpose of each of the simulation flows.

Note: The majority of these simulation flows are available in both the traditional design flow and the platform-based design flow. However, co-simulation of a complete system is only possible in the platform-based design flow.

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. For more information, see this [link](#) in the *Versal ACAP System Software Developers Guide* ([UG1304](#)).

Note: Embedded software simulation is possible in both the traditional design flow and the platform-based design flow.

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 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](#)).

Note: Logic simulation is possible in both the traditional design flow and the platform-based design flow.

Logic Simulation Using SystemC Models

SystemC is a C++ library that enables hardware modeling. This library provides structural elements, such as modules, ports, and interfaces, as well as data types. In addition to cycle-accurate simulation models, Xilinx provides fast, transaction-accurate, SystemC simulation models for some Versal ACAP infrastructure blocks for use in Vitis hardware emulation flows. SystemC models allow faster simulation compared to RTL models, which helps to reduce overall simulation time.

In general, SystemC models are used for performance analysis, architecture exploration, DMA synchronization, and address trace generation and performance modeling. However, Xilinx recommends using RTL models when accuracy and debugging are more important, such as with DMA transaction or timing dependent issues.

Table 9: Supported Simulation Models for Versal ACAP Blocks

Block	Cycle Accurate	Performance
PS	QEMU (functional only)	QEMU (functional only)
NoC	Behavioral SystemVerilog (cycle approximate)	SystemC
DDPMC	Behavioral SystemVerilog	SystemC
PL-based soft memory controller	Behavioral SystemVerilog	Behavioral SystemVerilog
CPM	Behavioral SecureIP	Behavioral SecureIP
GT	Behavioral SecureIP	File I/O (for Vitis software platform users only)
GT-based IP	Behavioral SecureIP	AXI verification IP File I/O (for Vitis software platform users only)
HLS-based IP	RTL	RTL
Other IP	Varies by IP	Varies by IP
PL	Behavioral Verilog VHDL SystemVerilog	Behavioral Verilog VHDL SystemVerilog
AI Engine	SystemC	SystemC

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 HLS Documentation](#) in the Application Acceleration Development flow of the *Vitis Unified Software Platform Documentation* (UG1416).

Note: HLS simulation is possible in both the traditional design flow and the platform-based design flow.

AI Engine Simulation

AI Engine simulation tests applications running on the Versal ACAP 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 (x86simulator) and cycle-approximate (aiesimulator), 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. When using the cycle-approximate abstraction, you can also use simulation results to improve the accuracy of the power estimation by more precisely calculating core vector load and memory access.

AI Engine simulation is available through the Vitis unified software platform. For more information, see [AI Engine SystemC Simulator](#) in the AI Engine Documentation flow of the *Vitis Unified Software Platform Documentation* (UG1416).

Note: AI Engine simulation is only possible in the platform-based design flow.

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 the [Vitis Accelerated Software Development Flow Documentation](#) in the Application Acceleration Development flow of the *Vitis Unified Software Platform Documentation* (UG1416) and the *Versal ACAP AI Engine Programming Environment User Guide* (UG1076).

Note: Hardware emulation is only possible in the platform-based design flow.

Boot Image Generation

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. For more information on the PDI file format and generation, see the *Bootgen User Guide* ([UG1283](#)).

Generation of the PDI varies based on your design flow:

- **Traditional design flow for hardware-only systems:** Run the `write_device_image` command. For more information, see this [link](#) in the *Vivado Design Suite User Guide: Programming and Debugging* ([UG908](#)).
- **Traditional design flow for embedded systems:** Run the Bootgen tool. For more information, see [Bootgen Tool](#) in the Embedded Software Development flow of the *Vitis Unified Software Platform Documentation* ([UG1416](#)).
- **Platform-based design flows:** Run the `v++ --package` command. For more information, see [Packaging the System](#) in the AI Engine Documentation flow of the *Vitis Unified Software Platform Documentation* ([UG1416](#)).



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.

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. For designs migrated from Zynq® UltraScale+™ MPSoCs, Xilinx recommends recreating the PS functions and connectivity by instantiating the CIPS IP in a new design instead of attempting migration via tools automation.

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

Table 10: Block and Functionality Migration Support

Block	Automated
Configurable logic block (CLB)	Yes
On-chip memory (OCM) resources (block RAM and UltraRAM)	Most
DSP	Yes
Clocking	Some
I/O	Some
Soft memory controllers	No
AXI Interconnect	No
GT	No
PCIe subsystems	Some

Table 10: Block and Functionality Migration Support (cont'd)

Block	Automated
MRMAC	No
Processor and peripherals	No
System debug	No
System monitor (SYSMON)	No
Power and error handling	No
Security	No
Boot and configuration	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 ACAP 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, even if the PS features are not used. Other designs migrating from Zynq UltraScale+ MPSoCs are expected to already have a PS block. For more information about the CIPS IP, see the *Control, Interface and Processing System LogiCORE IP Product Guide* (PG352).

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](#)

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 tool, 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](#)

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 or the Vitis™ HLS tool. If you want to use this mode in your RTL design, update the Floating-Point Operator IP in your migrated design.

Xilinx supports automated migration of instantiated DSP primitives to the Versal ACAP legacy primitive (DSP48E5). To achieve higher performance and utilization, Xilinx recommends updating your RTL to the Versal ACAP RTL templates and resynthesizing your design.

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. You can optionally use the Model Composer and System Generator flows to compare the PL and AI Engine implementations for designs created with MATLAB® and Simulink® software. For more information, see the *Model Composer and System Generator User Guide* ([UG1483](#)).

Related Information

[DSP Primitives](#)

[AI Engine](#)

Clocking

To achieve optimal clocking results in the Versal architecture, Xilinx highly recommends the following:

- Use the Clocking Wizard to configure the Versal ACAP clock management primitives. Relying on the Vivado tools to migrate your clock management functions from a previous architecture will likely result in a sub-optimal configuration. For more information, see the *Clocking Wizard for Versal ACAP LogiCORE IP Product Guide* ([PG321](#)).
- Review the physical locations of clock management primitives in Versal ACAP versus your clocking topology used in a previous architecture.

For more information about designing your clock network for your design, see the *Versal ACAP Hardware, IP, and Platform Development Methodology Guide* ([UG1387](#)). For more information on features and locations of the clock management primitives in Versal devices, see the *Versal ACAP Clocking Resources Architecture Manual* ([AM003](#)).

Although Versal devices have clocking features similar to UltraScale devices, you must be aware of the following important migration considerations.

Clock Management Functions

- The clock management functions are provided by the MMCME5, XPLL, and DPLL primitives in Versal devices. The clock management primitives in Versal devices contain additional deskew logic features when compared against similar primitives contained within UltraScale devices.
- The location of the clock management primitives in Versal devices are no longer in a regular structure when compared to the columnar architecture in UltraScale devices, and the primitives are only placed where required in Versal devices. In some scenarios, this can result in limited placement flexibility when migrating to a Versal device, and you must carefully review your clock structure during migration.
- The UltraScale+ device primitives migrate to Versal device primitives as follows:
 - The UltraScale+ device primitive MMCME4_ADV migrates to the MMCME5 Versal device primitive. The MMCME5 does not support ZHOLD compensation. The MMCME5 settings as a result of a migration from a previous architecture are likely to be sub-optimal, and Xilinx recommends using the Clocking Wizard to directly configure the MMCME5 for optimal performance in the Versal architecture.

- The UltraScale+ device primitive PLLE4_ADV migrates to the XPLL Versal device primitive. The XPLL settings as a result of a migration from a previous architecture are likely to be sub-optimal, and Xilinx recommends using the Clocking Wizard to directly configure the XPLL for optimal performance in the Versal architecture.

Global Clock Buffers

- Global clock buffers from previous architectures, such as BUFGCE, BUFGCE_DIV, BUFGCTRL, BUFG_PS, and BUFG_GT, automatically migrate to the Versal architecture.
- New multi-clock buffer (MBUG) primitives in Versal devices allow for clock division at the leaf level to reduce clock track utilization and improve timing on synchronous clock domain crossings.

Clock Routing Resources

- Versal devices have a clock routing structure similar to UltraScale devices, where global clocking is used throughout the device but the loads can be placed regionally or globally.
- Versal devices do not have a columnar I/O architecture, and there are only 12 horizontal routing tracks in clock regions without XPIO banks. Clock regions with XPIO banks have 24 horizontal routing tracks.

I/O

The IDDR/ODDR and IBUF/OBUF primitives are automatically migrated.

Related Information

[I/O Planning](#)

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. For more information on soft memory controller IP, including detailed information on pinout, see the following guides:

- *Versal ACAP Programmable Network on Chip and Integrated Memory Controller LogiCORE IP Product Guide* ([PG313](#))
- *Versal ACAP Soft DDR4 SDRAM Memory Controller LogiCORE IP Product Guide* ([PG353](#))
- *Versal ACAP Soft RDRAM 3 Memory Controller LogiCORE IP Product Guide* ([PG354](#))
- *Versal ACAP Soft QDR-IV SRAM Memory Controller LogiCORE IP Product Guide* ([PG355](#))

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](#)).

GT

For Versal ACAP, GT components are updated from Common/Channel to a GT 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 that use single or multiple GT Quads. The design entry for custom IP that connects to GT Quads is through the Bridge IP, which instantiates, configures, and connects single or multiple GT Quad-based IP through Block Automation. For more information, see the *Versal ACAP Transceivers Wizard LogiCORE IP Product Guide* ([PG331](#)) and this [link](#) in the *Versal ACAP Hardware, IP, and Platform Development Methodology Guide* ([UG1387](#)). 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 and GTYP Transceivers Architecture Manual* ([AM002](#)).

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 interfaces are significantly enhanced implementations of the integrated blocks for PCIe interfaces found in previous architectures. Two implementations of the PL PCIe interfaces 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. The core can be configured in Endpoint, Root port, or Switch mode. DMA/Bridge subsystems for use with the PL PCIE4 are available through the Vivado ACAP 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. The core can be configured in Endpoint, Root port, or Switch mode. 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.

If your design needs to be migrated from an integrated block for PCIe in a previous architecture to a Versal ACAP PL PCIE4 or PCIE5, consider the following:

- Only the Vivado IP integrator-based block design flow is currently supported with manual or automatic connectivity.
- The required GT and PHY IP blocks for Versal ACAP PL PCIe interfaces are outside of the Versal ACAP PL PCIE4 IP.
- Configure the PCIe subsystem with the required link speed, width, and features using the PL PCIE4 core, and either run block automation or instantiate and connect Versal ACAP PHY and GT Quad IPs manually.
- Xilinx recommends driving fundamental reset for the PCIe controller using the I/O inside the PS, which must be configured in the CIPS IP.
- Manually map RQ/RC/CQ/CC streaming interfaces and side band signals, which are similar to their respective IP implementation from previous architectures.

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

- Configure the PCIe subsystem with the required link speed, width, and features in the CPM using the CIPS IP core.

Note: The CPM has fixed connectivity to GTs based on the CPM configuration and this cannot be altered.

- Fundamental reset for the PCIe controller is driven by the I/O inside the PS, which must be configured in the CIPS IP.
- Only `user_clk`, which can have a frequency of 62.5, 125, or 250 MHz depending on the configured link speed and width, is available for the programmable logic.
- Manually map RQ/RC/CQ/CC streaming, sideband signals, XDMA streaming, and QDMA streaming interfaces to Versal ACAP CPM PL interfaces. These interfaces are similar to their respective IP implementation from previous architectures.
- Pipe mode is not supported.
- 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.

Tandem PCIe interface 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.

For solutions that require PCIe Tandem and DFX over PCIe interfaces, migrate to CPM. CPM natively supports 100 ms Endpoint boot times and has hardened connectivity to the PMC to enable configuration and DFX programming. There are no current plans to support Tandem PCIe interfaces for PL-based PCIe controllers.

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 DMA and Bridge Subsystem for PCI Express Product Guide* ([PG344](#))
- *Versal ACAP PCIe PHY LogiCORE IP Product Guide* ([PG345](#))
- *Versal ACAP CPM Mode for PCI Express Product Guide* ([PG346](#))
- *Versal ACAP CPM DMA and Bridge Mode for PCI Express Product Guide* ([PG347](#))

Related Information

[CPM](#)

MRMAC

If you are migrating from UltraScale or UltraScale+ device Integrated 100G Ethernet (CMAC) hard block or soft 10G/25G/40G or 50G Ethernet IP, consider the following:

- MRMAC provides wider customization for line rate, clocking, and user interface:
 - Supported configurations are: 1 x 100GE, 2 x 50GE, 1 x 40GE; 4 x 25GE, and 4 x 10GE.
 - MRMAC now has an integrated AXIS interface for MAC+PCS operation as opposed to CMAC, which offered integrated 512-bit LBUS interface with optional AXIS interface.
 - Various AXIS bus widths and clocking options are available and depending on configuration, vary from those available in UltraScale or UltraScale+ device CMAC or soft core solutions.
 - There is a new Flex Port option for access to PCS level.
- The GT is not included as part of the MRMAC core. IP integrator block automation is used to connect between the MRMAC and GT.
- Instead of provided statistic counter increment vectors, statistics registers are now integrated as part of the hard block and available over AXI4-Lite.
- The MRMAC also supports a new high-precision timestamping feature to enable sub-nanosecond accuracy on IEEE Std 1588 timestamps.

For more information on the MRMAC and details on generating the MRMAC example design, see the *Versal Devices Integrated 100G Multirate Ethernet MAC (MRMAC) LogiCORE IP Product Guide* ([PG314](#)).

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](#)).

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 AXI-Debug Hub IP core has both AXI4-Stream control interfaces (for connection to fabric debug IP cores) and an AXI4-Memory Map slave interface for connection from the host. The Debug Hub IP used 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.
- The AXI4-Stream-based ILA core supports selection of block RAM or UltraRAM as the trace storage memory.

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 `pIO_resetn` is enabled on the CIPS. Alternatively, an AXI4 Debug Hub can be manually added. For details, see this [link](#) in the *Vivado Design Suite User Guide: Programming and Debugging (UG908)*.

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). To migrate your design, you must manually remove the SYSMON IP from the design. In Versal ACAP, you enable the system monitor features in the CIPS IP Configuration wizard. Dynamic reconfiguration port (DRP) access is replaced by memory mapped registers. For more information, see the *Versal ACAP System Monitor Architecture Manual* ([AM006](#)).

Power and Error Handling

Zynq UltraScale+ MPSoCs have some power modes that can be mapped to the power modes in Versal ACAP. Peripherals that are power islands are shared in the Versal ACAP. These power islands are automatically turned off when not in use by the PLM. External power rails that supply power domains will be supported in a future release. 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](#)).

Security

The security architecture of Versal ACAP is significantly enhanced from previous generations. The root of trust starts with the PMC ROM, which authenticates and optionally, decrypts the PLM software. The PMC ROM can only be loaded into and run from the PPU in the PMC. After the PLM software is authenticated, the PLM ensures secure loading of the remaining firmware and software. For more information, see the *Versal ACAP System Software Developers Guide* ([UG1304](#)), *Versal ACAP Technical Reference Manual* ([AM011](#)), or visit the [Design Security Lounge](#) (registration required) on the Xilinx website for access to detailed security related information. The following table highlights the possible secure boot configurations for Versal ACAP and shows a comparison with Zynq UltraScale+ MPSoC.

Note: Although there are similarities between the Zynq UltraScale+ MPSoC Encrypt Only (EO) flow and the Versal ACAP Symmetric Hardware Root of Trust (S-HWRoT), the two modes are significantly different in implementation.

Table 11: Cumulative Secure Boot Operations

Boot Type	Operations			Hardware Crypto Engines	
	Authentication	Decryption	Integrity (Checksum Verification)	Zynq UltraScale+ MPSoC	Versal ACAP
Non-secure	No	No	No	N/A	N/A
Hardware Root-of-Trust (HWRoT)	Yes	Optional	Integrity via Authentication	RSA, SHA3	N/A
Encrypt Only (EO)	Yes via GCM	Yes	Integrity via Authentication	AES-GCM	N/A
Asymmetric Hardware Root-of-Trust (A-HWRoT)	Yes	Optional	Integrity via Authentication	N/A	RSA/ECDSA and SHA3
Symmetric Hardware Root-of-Trust (S-HWRoT)	Yes via GCM and eFUSES	Yes Must use PUF KEK	Integrity via Authentication	N/A	AES-GCM/PUF
A-HWRoT + S-HWRoT	Yes	Yes Must use PUF KEK	Integrity via Authentication	N/A	RSA/ECDSA, SHA3, AES-GCM, PUF
Authentication + Decryption	Yes	Yes	Integrity via Authentication	RSA, SHA3, AES-GCM	RSA/ECDSA, SHA3, AES-GCM
Decrypt Only	No	Yes	Yes	AES-GCM	AES-GCM
Checksum Verification	No	No	Yes	SHA3	SHA3

Boot and Configuration

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 this [link](#) in the *Versal ACAP Technical Reference Manual (AM011)*, this [link](#) in the *Versal ACAP System Software Developers Guide (UG1304)*, and the *Bootgen User Guide (UG1283)*.

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

Table 12: 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.

PL Configuration and JTAG

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

- The BSCANE2 primitive is replaced by four JTAG TAP user instructions available in the CIPS IP.
- The STARTUPE3 primitive is replaced by the combination of the QSPI controller MIO and CIPS IP (global asynchronous set/reset signal, global 3-state, end of startup (EOS) signal, PL clocks (PLO-PL3) source configuration).
- The DNA_PORTE2 primitive is replaced by the JTAG DNA register or AXI memory mapped accessible 32-bit registers DNA_0, DNA_1, DNA_2, and DNA_3 to read out the device DNA.
- The EFUSE_USR primitive is replaced by the AXI memory mapped EFUSE_CACHE registers.

For more information on the memory mapped registers, including address mapping, see the *Versal ACAP Technical Reference Manual* ([AM011](#)) and the *Versal ACAP Register Reference* ([AM012](#)).

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 figures show examples.

Figure 6: Verilog 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

```

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

```

architecture beh of test is

type my_ram_type is array (511 downto 0) of std_logic_vector(71 downto 0);
signal my_ram : my_ram_type;
signal my_ram_out : std_logic_vector(71 downto 0);

begin

process(w_clk) begin
if (w_clk'event and w_clk='1') then
if (we = '1') then
my_ram(to_integer(unsigned(w_addr))) <= din;
end if;
end if;
end process;

process(r_clk) begin
if (r_clk'event and r_clk='1') then
my_ram_out <= my_ram(to_integer(unsigned(r_addr)));
dout <= my_ram_out;
end if;
end process;

end beh;

```

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.

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.

Using XPMs

Block RAMs can also be inferred using XPMs. The advantage of using this approach is that XPMs always have the correct coding style for any type of RAM needed. For more information on XPMs, see the *Vivado Design Suite User Guide: System-Level Design Entry* (UG895).

UltraRAM Primitives

Following is the UltraRAM primitive in Versal ACAPs. To force Vivado synthesis to infer the UltraRAM, 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

In Versal ACAPs, the UltraRAMs can be initialized to a non-zero value. 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. For examples on how to code asymmetric RAMs, see this [link](#) in the *Vivado Design Suite User Guide: Synthesis (UG901)*.

XPM Inference

UltraRAMs can also be inferred using XPMs. The advantage of using this approach is that XPMs always have the correct coding style for any type of RAM needed. For more information on XPMs, see the *Vivado Design Suite User Guide: System-Level Design Entry (UG895)*.

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 8: Elaborated View of 27x24 Multiplier with 58-Bit Adder and 27-Bit Pre-Adder

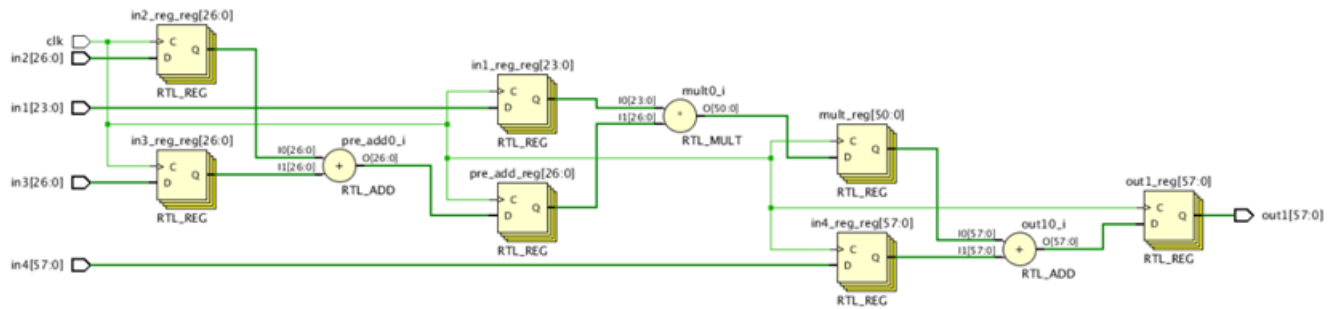


Figure 9: Verilog RTL for a 27x24 Multiplier with 58-Bit Adder and 27-Bit Pre-Adder

```

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

```

architecture beh of test is

    signal in1_reg : std_logic_vector(23 downto 0);
    signal in2_reg, in3_reg : std_logic_vector(26 downto 0);
    signal in4_reg : std_logic_vector(57 downto 0);
    signal mult : std_logic_vector(50 downto 0);
    signal pre_add : std_logic_vector(26 downto 0);

begin

    process(clk) begin
        if (clk'event and clk='1') then
            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 if;
    end process;

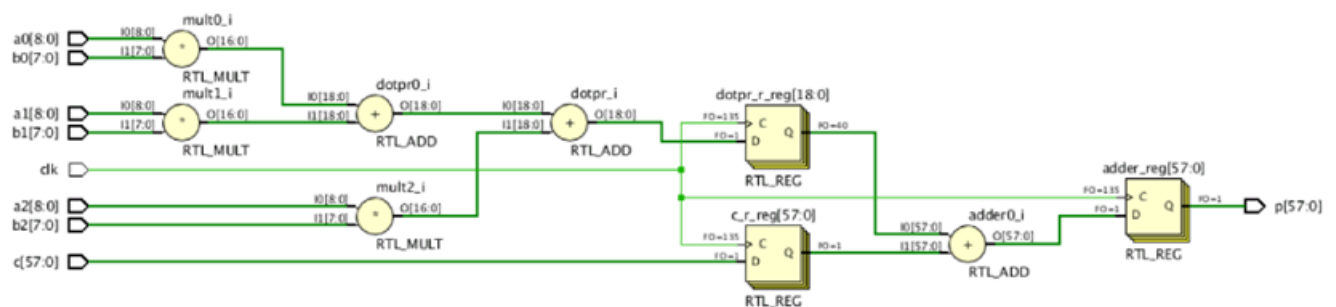
end beh;
    
```

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 11: Elaborated View of a Dot Product with an Extra Adder



The following figures show the RTL for a dot product.

Figure 12: Verilog 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
  
```

Figure 13: VHDL RTL for the Dot Product

```

process(clk) begin
  if clk'event and clk='1' then
    c_r <= c;
  end if;
end process;

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

process(clk) begin
  if clk'event and clk='1' then
    dotpr_r <= dotpr;
    adder <= dotpr_r + c_r;
  end if;
end process;

p <= adder;
  
```

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 figures show the RTL for the DSPCPLX.

Figure 14: Verilog RTL to Synthesize the DSPCPLX

```

always @(posedge clk) begin
    ar_d <= ar;
    ai_d <= ai;
    bi_d <= bi;
    br_d <= br;
end
//Balance Pipeline ADREG=0
assign addcommon = ar_d - ai_d;
assign addr = br_d - bi_d;
assign addi = br_d + bi_d;
//Common factor (ar-ai)*bi, shared for calculations of real & imaginary final
//products
assign multcommon = bi_d * addcommon;
assign multr = ar_d * addr;
assign multi = ai_d * addi;
//Multiplier outputs are registered MREG=1
always @(posedge clk) begin
    multcommon_d <= multcommon;
    multr_d <= multr;
    multi_d <= multi;
end
//Complex outputs are registered PREG=1
always @(posedge clk) begin
    pr <= multcommon_d + multr_d;
    pi <= multcommon_d + multi_d;
end

```


Figure 15: VHDL RTL to Synthesize the DSPCPLX

```

process(clk)
begin
if clk'event and clk = '1' then
    ar_d <= ar;
    ai_d <= ai;
    bi_d <= bi;
    br_d <= br;
end if;
end process;

addcommon <= resize(ar_d,19) - resize(ai_d,19);
addr <= resize(br_d,19) - resize(bi_d,19);
addi <= resize(br_d,19) + resize(bi_d,19);
multcommon <= bi_d * addcommon;
multr <= ar_d * addr;
multi <= ai_d * addi;

--Multiplier outputs are registered MREG=1
process(clk)
begin
if clk'event and clk = '1' then
    multcommon_d <= multcommon;
    multr_d <= multr;
    multi_d <= multi;
end if;
end process;

--Complex outputs are registered PREG=1
process(clk)
begin
if clk'event and clk = '1' then
    pr <= multcommon_d + multr_d;
    pi <= multcommon_d + multi_d;
end if;
end process;

end beh;
  
```

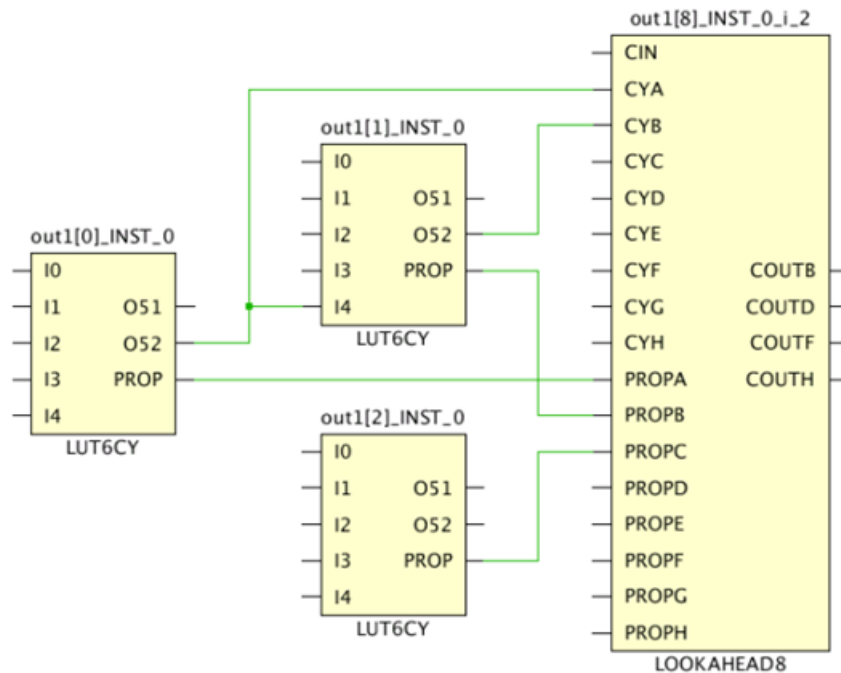
CLB Primitives

The configurable logic blocks (CLBs) in Versal ACAPs differ from those in UltraScale devices. Vivado synthesis handles the architectural differences, 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.

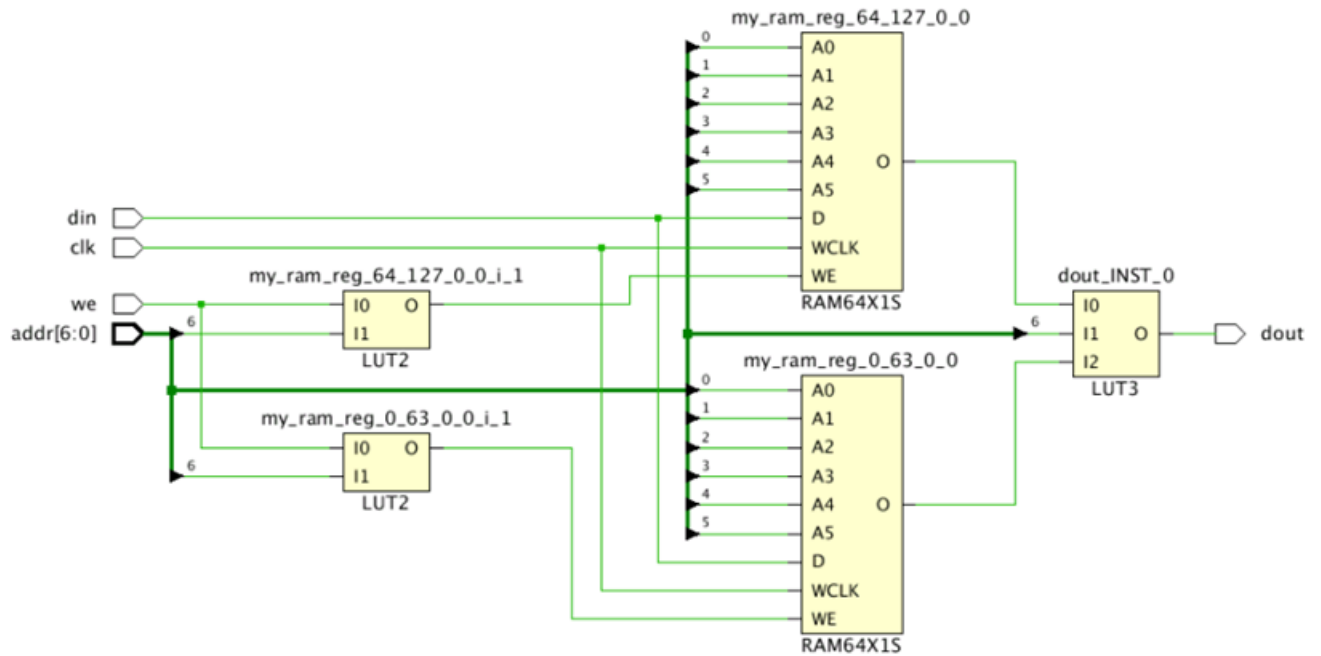
Figure 16: Extra LUTs Before the CARRY Chain



MUXF_x Primitives

Versal ACAPs do not include MUXF_x primitives. Because MUXF_x 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



Coding Style and Primitive Instantiation Examples

For coding style and primitive instantiation examples, see the following resources:

- Language Templates in the Vivado IDE
- This [link](#) in the *Vivado Design Suite User Guide: Synthesis (UG901)*

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. *Extending the Thermal Solution by Utilizing Excursion Temperatures* ([WP517](#))
3. *Versal ACAP GTY and GTYP Transceivers Architecture Manual* ([AM002](#))
4. *Versal ACAP Clocking Resources Architecture Manual* ([AM003](#))
5. *Versal ACAP DSP Engine Architecture Manual* ([AM004](#))
6. *Versal ACAP Configurable Logic Block Architecture Manual* ([AM005](#))
7. *Versal ACAP System Monitor Architecture Manual* ([AM006](#))
8. *Versal ACAP Memory Resources Architecture Manual* ([AM007](#))
9. *Versal ACAP AI Engine Architecture Manual* ([AM009](#))
10. *Versal ACAP SelectIO Resources Architecture Manual* ([AM010](#))
11. *Versal ACAP Technical Reference Manual* ([AM011](#))
12. *Versal ACAP Register Reference* ([AM012](#))
13. *Versal ACAP Packaging and Pinouts Architecture Manual* ([AM013](#))
14. *Versal ACAP CPM CCIX Architecture Manual* ([AM016](#))
15. *SmartConnect LogiCORE IP Product Guide* ([PG247](#))
16. *Versal ACAP Programmable Network on Chip and Integrated Memory Controller LogiCORE IP Product Guide* ([PG313](#))
17. *Versal Devices Integrated 100G Multirate Ethernet MAC (MRMAC) LogiCORE IP Product Guide* ([PG314](#))
18. *Advanced I/O Wizard LogiCORE IP Product Guide* ([PG320](#))
19. *Versal ACAP Transceivers Wizard LogiCORE IP Product Guide* ([PG331](#))
20. *Versal ACAP Integrated Block for PCI Express LogiCORE IP Product Guide* ([PG343](#))
21. *Versal ACAP DMA and Bridge Subsystem for PCI Express Product Guide* ([PG344](#))
22. *Versal ACAP PCIe PHY LogiCORE IP Product Guide* ([PG345](#))
23. *Versal ACAP CPM Mode for PCI Express Product Guide* ([PG346](#))
24. *Versal ACAP CPM DMA and Bridge Mode for PCI Express Product Guide* ([PG347](#))
25. *Control, Interface and Processing System LogiCORE IP Product Guide* ([PG352](#))
26. *Versal ACAP Soft DDR4 SDRAM Memory Controller LogiCORE IP Product Guide* ([PG353](#))
27. *Versal ACAP Soft RLDRAM 3 Memory Controller LogiCORE IP Product Guide* ([PG354](#))
28. *Versal ACAP Soft QDR-IV SRAM Memory Controller LogiCORE IP Product Guide* ([PG355](#))
29. *AI Engine LogiCORE IP Product Guide* ([PG358](#))
30. *Versal ACAP PCB Design User Guide* ([UG863](#))

31. *Vivado Design Suite User Guide: System-Level Design Entry* ([UG895](#))
32. *Vivado Design Suite User Guide: Logic Simulation* ([UG900](#))
33. *Vivado Design Suite User Guide: Synthesis* ([UG901](#))
34. *Vivado Design Suite User Guide: Power Analysis and Optimization* ([UG907](#))
35. *Vivado Design Suite User Guide: Programming and Debugging* ([UG908](#))
36. *Vivado Design Suite User Guide: Designing IP Subsystems Using IP Integrator* ([UG994](#))
37. *Vivado Design Suite: AXI Reference Guide* ([UG1037](#))
38. *Versal ACAP AI Engine Programming Environment User Guide* ([UG1076](#))
39. *Versal ACAP AI Engine Kernel Coding User Guide* ([UG1079](#))
40. *Xilinx Power Estimator User Guide for Versal ACAP* ([UG1275](#))
41. *Bootgen User Guide* ([UG1283](#))
42. *Versal ACAP System Software Developers Guide* ([UG1304](#))
43. *Versal ACAP Hardware, IP, and Platform Development Methodology Guide* ([UG1387](#))
44. *Versal ACAP System Integration and Validation Methodology Guide* ([UG1388](#))
45. *XRT Release Notes* ([UG1451](#))
46. *Model Composer and System Generator User Guide* ([UG1483](#))
47. *Versal ACAP System and Solution Planning Methodology Guide* ([UG1504](#))
48. *Versal ACAP Board System Design Methodology Guide* ([UG1506](#))
49. *Seven Steps to an Accurate Worst-Case Power Analysis using the Xilinx Power Estimator* ([XAPP1348](#))
50. *Versal ACAP Schematic Review Checklist* ([XTP546](#))
51. *Versal ACAP External Memory Pre-Planning Tool* ([XTP667](#))
52. [AI Engine SystemC Simulator](#) in the AI Engine Documentation flow of the *Vitis Unified Software Platform Documentation* ([UG1416](#))
53. [Bootgen Tool](#) in the Embedded Software Development flow of the *Vitis Unified Software Platform Documentation* ([UG1416](#))
54. [Debugging the AI Engine Application](#) in the AI Engine Documentation flow of the *Vitis Unified Software Platform Documentation* ([UG1416](#))
55. [Packaging the System](#) in the AI Engine Documentation flow of the *Vitis Unified Software Platform Documentation* ([UG1416](#))
56. [Programming the PS Host Application](#) in the AI Engine Documentation flow of the *Vitis Unified Software Platform Documentation* ([UG1416](#))
57. [RTL Kernels](#) in the Application Acceleration Development flow of the *Vitis Unified Software Platform Documentation* ([UG1416](#))

- 58. [Vitis Accelerated Software Development Flow Documentation](#) in the Application Acceleration Development flow of the *Vitis Unified Software Platform Documentation* (UG1416)
- 59. [Vitis HLS Documentation](#) in the Application Acceleration Development flow of the *Vitis Unified Software Platform Documentation* (UG1416)
- 60. *Vitis Unified Software Platform Documentation* ([UG1416](#))

Please Read: Important Legal Notices

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx's limited warranty, please refer to Xilinx's Terms of Sale which can be viewed at <https://www.xilinx.com/legal.htm#tos>; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at <https://www.xilinx.com/legal.htm#tos>.

AUTOMOTIVE APPLICATIONS DISCLAIMER

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING

OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.

Copyright

© Copyright 2020–2021 Xilinx, Inc. Xilinx, the Xilinx logo, Alveo, Artix, Kintex, Spartan, Versal, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. AMBA, AMBA Designer, Arm, ARM1176JZ-S, CoreSight, Cortex, PrimeCell, Mali, and MPCore are trademarks of Arm Limited in the EU and other countries. MATLAB and Simulink are registered trademarks of The MathWorks, Inc. OpenCL and the OpenCL logo are trademarks of Apple Inc. used by permission by Khronos. PCI, PCIe, and PCI Express are trademarks of PCI-SIG and used under license. All other trademarks are the property of their respective owners.