查看所有版本

Versal 自适应 SoC 设计指南

UG1273 (v2023.1) 2023 年 5 月 10 日

目录

第 1 章

概述

Versal 自适应 SoC 简介

AMD Versal™ 自适应 SoC 将标量引擎 (Scalar Engine)、自适应引擎 (Adaptable Engine) 和智能引擎 (Intelligent Engine) 与领先的存储器和接口技术有机结合，为所有应用提供强大的异构加速能力。其硬件和软件是专为数据科学家和软硬件开发者开展编程和优化工作而提供的。其器件受到诸多工具、软件、资源库、IP、中间件和框架的广泛支持，适用于所有业界标准的设计流程。

基于 TSMC 7 nm FinFET 工艺技术构建的 Versal 产品服务组合以前所未有的方式将软件可编程性和特定领域的硬件加速与自适应能力有机结合在一起，以满足当今快速创新节奏的需求。该产品服务组合包含 6 大器件系列，其架构是专为跨不同市场的各种应用（从云、网络、无线通信到边缘计算和端点）提供可缩放性和 AI 推断功能而构建的。

Versal 架构将不同类型的引擎与丰富的连接和通信功能以及可编程片上网络 (NoC) 相结合，从而支持实现覆盖整个器件的无缝式存储器映射访问。智能引擎包括适用于自适应推断和高级信号处理计算的 SIMD VLIW AI 引擎以及用于定点、浮点和复杂 MAC 运算的 DSP 引擎。自适应引擎将可编程逻辑块与存储器有机结合，它具备专为应对高计算密度需求而设计的架构。标量引擎使用 Arm® Cortex®-A72 和 Cortex-R5F 处理器，支持计算密集型任务。

Versal AI Edge 系列侧重于自动驾驶实时系统、预测性工厂和医疗保健系统以及其他各种应用的 AI 单位功耗性能。不仅仅是提高 AI 的速度，Versal AI Edge 系列还提高了整个应用的速度，包括传感器、AI 和实时控制装置，为它们提供了最高级别的安全性，让它们能够满足 ISO26262 和 IEC 61508 等关键标准。

Versal AI Core 系列可凭借 AI 引擎提供突破性的 AI 推断加速能力，其应用范围广泛，包括用于云端动态工作负载以及超高带宽网络，同时还可提供高级安全保障功能。AI 和数据科学家以及软硬件开发者均可充分利用高计算密度的优势来加速提升任何应用的性能。

Versal Prime 系列覆盖跨多个市场领域的 Versal 产品组合的基础和中端产品，适用范围最为广泛。其应用范围包括 100G 到 200G 联网设备、数据中心内部网络与存储加速、通信测试设备及广播等。此系列将主流 58G 收发器与经优化的 I/O 和 DDR 连接整合在一起，从而为多种多样的工作负载实现低时延加速和性能。

Versal Premium 系列在可灵活适应的平台内提供突破性的异构集成、超高性能计算、连接和安全性，同时最大限度降低功耗和占地面积。此系列是专为超越高带宽、计算密集型有线通信、数据中心、测试测量等应用的需求而设计的。Versal Premium 系列包含 112G PAM4 收发器和集成块，适用于 600G 以太网、600G Interlaken、PCI Express® Gen5 和高速加密技术。

Versal HBM 系列支持将快速存储器、自适应计算和安全连接聚合在一起。该系列架构旨在满足大部分存储器受限的计算密集型应用对于更大的存储器的需求，能够为数据中心、有线网络、测试测量、航空航天和国防等应用提供自适应加速。Versal HBM 系列集成了最先进的 HBM2e DRAM，可在单一器件内提供高存储器带宽和容量。

如需获取 Versal 架构文档套件，请访问：https://china.xilinx.com/versal。

按设计进程浏览内容

AMD 自适应计算文档按一组标准设计进程进行组织，以便帮助您查找当前开发任务相关的内容。所有 AMD Versal™ 自适应 SoC 设计进程的对应设计中心和设计流程助手资料均可在 Xilinx.com 网站上找到。本文档涵盖了以下设计进程：

- 系统和解决方案规划：确认系统级别的组件、性能、I/O 和数据传输要求。包括解决方案到 PS、PL 和 AI 引擎的应用映射。本文档中适用于此设计进程的主题包括：
  - 第 2 章：系统架构
  - 第 3 章：系统规划
  - 第 4 章：设计流程
  - 第 5 章：系统移植

  注释：如需了解更多信息，请参阅《Versal 自适应 SoC 系统和解决方案规划方法指南》(UG1504)。
- 嵌入式软件开发：从硬件平台创建软件平台，并使用嵌入式 CPU 开发应用代码。还涵盖 XRT 和 Graph API。本文档中适用于此设计进程的主题包括：
  - 在设计流程中使用 Vitis 环境
  - 仿真流程

  注释：如需了解更多信息，请参阅《AI 引擎工具和流程用户指南》(UG1076) 中的对 PS 主机应用进行编程。
- AI 引擎开发：创建 AI 引擎 Graph 及内核、库用法、仿真调试与剖析以及算法开发。还包含 PL 与 AI 引擎内核的集成。本文档中适用于此设计进程的主题包括：
  - AI 引擎
  - 在设计流程中使用 Vitis 环境

  注释：如需了解更多信息，请参阅《AI 引擎工具和流程用户指南》(UG1076) 和《AI 引擎内核与 Graph 编程指南》(UG1079)。
- 硬件、IP 和平台开发：为硬件平台创建 PL IP 块、创建 PL 内核、功能仿真以及评估 AMD Vivado™ 时序收敛、资源使用情况和功耗收敛。还涉及为系统集成开发硬件平台。本文档中适用于此设计进程的主题包括：
  - 在设计流程中使用 Vivado 工具
  - 仿真流程

  注释：如需了解更多信息，请参阅《Versal 自适应 SoC 硬件、IP 和平台开发方法指南》(UG1387)。
- 系统集成与确认：集成和确认系统功能性能，包括时序收敛、资源使用情况和功耗收敛。本文档中适用于此设计进程的主题包括：
  - 第 4 章：设计流程

  注释：如需了解更多信息，请参阅《Versal 自适应 SoC 系统集成和确认方法指南》(UG1388)。
- 开发板系统设计：通过原理图和开发板布局设计 PCB。还包含功耗、散热以及信号完整性注意事项。本文档中适用于此设计进程的主题包括：
  - 第 3 章：系统规划

  注释：如需了解更多信息，请参阅《Versal 自适应 SoC 开发板系统设计方法指南》(UG1506)。

关于本指南

本指南旨在提供 Versal 自适应 SoC 的高层次综述，如下所示：

- 第 2 章：系统架构：提供 Versal 自适应 SoC 概述，包含每个高层次集成块的摘要，其中包括每个块的用途以及彼此之间的关联。
- 第 3 章：系统规划：描述每个 Versal 器件系列与不同系统设计类型和设计流程之间的关联。
- 第 4 章：设计流程：描述 AMD 设计工具和支持用于 Versal 自适应 SoC 的设计流程。
- 第 5 章：系统移植：为以 Versal 自适应 SoC 为目标的设计提供高层次系统移植建议以及每个块的移植信息。
- 附录 A: 原语：提供有关 Versal 自适应 SoC 原语的信息。

第 2 章

系统架构

AMD Versal™ 器件按不同的应用和市场目标划分为多个不同系列。在不同系列之间，有些组件的可用性或功能是一致的，有些组件则不尽相同。主要资源包括但不限于：

- AI 引擎
- 可编程逻辑 (PL)
- 片上网络 (NoC)
- 高速 I/O (XPIO)
- 集成存储器控制器 (DDRMC)
- 高带宽存储器 (HBM)
- 处理器系统 (PS)
- 平台管理控制器 (PMC)
- Integrated Block for PCIe®，含 DMA 和高速缓存一致性互连 (CPM)
- 收发器 (GT)
- 高速调试端口 (HSDP)
- 高速连接和加密集成 IP

下表显示了不同系列之间各异的几项功能特性。如果列标题中所列某个器件中存在资源，则该资源的类型即为表中所列的类型。并非所有器件都包含资源，如需了解更多详细信息或者器件专属信息，请参阅《Versal 架构和产品数据手册概述》(DS950)。

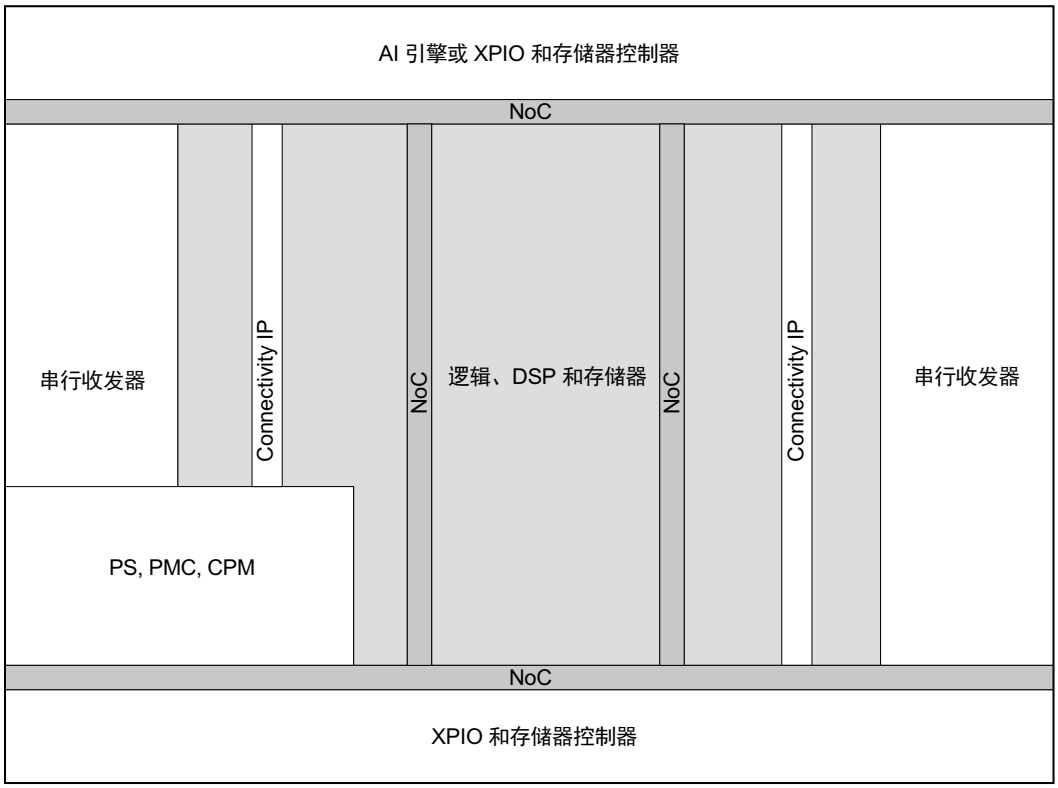
表 1：**按系列划分的 Versal 器件**

系列	AI Edge		AI Core		Prime		Premium			HBM
器件	**VE1xxx**	**VE2xxx**	**VC1xxx**	**VC2xxx**	**VM1xxx**	**VM2xxx**	**VP10xx**	**VP1xxx**	**VP2xxx**	**全部**
AI 引擎	AIE	AIE-ML	AIE	AIE-ML	-	-	-	-	AIE	-
处理器系统	PS	PS	PS	PS	PS	PS	PS	PS	PS	PS
GTY/GTYP	GTY	GTYP	GTY/ GTYP	GTYP	GTY/ GTYP	GTYP	GTY/ GTYP	GTY/ GTYP	GTYP	GTYP
GTM	-	-	-	-	-	58G	112G	112G	112G	112G
CPM	Gen4x16	Gen4x16	Gen4x16	Gen5x8	Gen4x16	Gen5x8	Gen4x4	Gen5x8	Gen5x8	Gen5x8
PCIe	Gen4x8	Gen4x8	Gen4x8	Gen5x4	Gen4x8	Gen5x4	Gen4x8	Gen5x4	Gen5x4	Gen5x4
Multirate Ethernet MAC	40G	100G	100G	100G	100G	100G	100G	100G	100G	100G
600G Ethernet MAC	-	-	-	-	-	-	600G	600G	600G	600G
600G Interlaken	-	-	-	-	-	-	600G	600G	600G	600G
400G High-Speed Crypto	-	-	-	-	-	-	400G	400G	400G	400G
HBM	-	-	-	-	-	-	-	-	-	是
VDU	-	是	-	是	-	-	-	-	-	-

Versal 器件应用可以利用此处每一项资源的功能。要创建设计或将设计移植到 Versal 器件，必须确定哪些资源最能满足应用的不同需求，并跨资源对应用进行分区。

下图显示了 Versal 器件的布局。

图 1：**Versal 器件布局**



以下章节简要介绍了构成 Versal 架构的块。如需了解有关这些块的详细信息，请参阅《Versal 架构和产品数据手册概述》(DS950)。

AI 引擎

Versal AI Core 系列可借助 AI 引擎提供突破性的 AI 推断加速，此 AI 引擎的计算性能较当前服务器级 CPU 高 100 倍。此系列应用范围广泛，包括用于云端动态工作负载以及超高带宽网络，同时还可提供高级安全性功能。AI 和数据科学家以及软硬件开发者均可充分利用高计算密度的优势来加速提升任何应用的性能。鉴于此 AI 引擎所具备的高级信号处理计算能力，它十分适合用于高度优化的无线应用，例如，射频、5G、回程 (backhaul) 和其他高性能 DSP 应用。

AI 引擎是超长指令字 (VLIW) 处理器阵列，具有高度优化的单指令流多数据流 (SIMD) 矢量单元，专用于各种计算密集型应用，尤其是数字信号处理 (DSP)、5G 无线应用和人工智能 (AI) 技术（如机器学习 (ML)）等。

AI 引擎是硬化的块，可提供多级并行处理能力，包括指令级并行处理和数据级并行处理。指令级并行度包括标量操作：最高 2 次移动、2 次矢量读取（加载）、1 次矢量写入（存储）和 1 条可执行的矢量指令，总计每个时钟周期达 7 路 VLIW 指令。数据级并行度是通过矢量级操作来实现的，其中每个时钟周期可执行多组数据操作。每个 AI 引擎都包含矢量处理器和标量处理器、专用程序存储器、本地 32 KB 数据存储器、支持访问三个相邻方向内任一方向的本地存储器。它还可访问 DMA 引擎和 AXI4 互连开关，以通过串流来与其他 AI 引擎进行通信或者与可编程逻辑 (PL) 或 DMA 进行通信。请参阅《Versal 自适应 SoC AI 引擎架构手册》(AM009) 以获取有关 AI 引擎阵列和接口的具体详细信息。

AI 引擎机器学习 (AIE-ML) 块可交付的计算吞吐量是上一代 AI 引擎块的 2 倍。AIE-ML 块的首要目标是机器学习推断应用，它能为广泛而丰富的推断应用提供业内首屈一指的单位功耗性能。

作为应用用户，您可以使用任一白箱或黑箱流程在 AIE-ML 上运行 ML 推断应用。白箱流程使用库元素，您可在 AIE-ML 编程环境中集成定制内核与数据流 graph。黑箱流程使用来自 AMD 的深度学习处理单元 (DPU) IP 来加速 AIE-ML 块中的 ML 工作负载，这些 IP 都经过性能最优化。

AMD Vitis™ AI 用作为前端工具，以解析网络 graph、执行最优化、graph 量化并生成量化网络模型，此类模型可在 AIE-ML 硬件上进行加速。AIE-ML 核拼块架构支持多种精度固定的数据类型和浮点数据类型，并拥有高密度高速度的流水线矢量处理片上存储器，可用于存储能对存储器中的多维张量进行寻址的片上张量和灵活的数据移动程序。通过正确选择片上/片外存储器中而输入/输出张量的覆层处理器架构以及空间和时间分布，即可提升 AIE-ML 处理核的计算效率。

可编程逻辑

Versal 自适应 SoC 可编程逻辑 (PL) 包括可配置逻辑块 (CLB)、内部存储器以及 DSP 引擎。每个 CLB 都包含 64 个触发器和 32 个查找表 (LUT)。半数 CLB LUT 可以配置为 1 个 64 位 RAM 和 1 个 32 位移位寄存器 (SRL32) 或配置为 2 个 16 位移位寄存器 (SRL16)。除了 LUT 和触发器之外，CLB 还包含：

- 超前进位逻辑，用于实现算术函数或宽逻辑函数
- 专用内部连接，以创建快速 LUT 级联，无外部布线

这样即可实现灵活的进位逻辑结构，使进位链能从链中的任何一位开始。除了 CLB 中的分布式 RAM（每个 64 位）功能，还有专用块用于在设计中以最优方式构建存储器阵列：

- 加速器 RAM (4 MB)（仅在部分 Versal 器件中可用）
- 块 RAM（每个 36 Kb），在简单的双端口模式下，每个端口可以配置为 4Kx9、2Kx18、1Kx36 或 512x72
- UltraRAM（每个 288 Kb），每个端口可以配置为 32Kx9、16Kx18、8Kx36 或 4Kx72

Versal 器件还包括许多低功耗 DSP 引擎，具有高速且尺寸小的特点，同时还保留了系统设计灵活性。DSP 引擎可配置为在各种模式下运行，以更好地匹配应用需求：

- 27×24 位 2 的补码乘法器和 58 位累加器
- 三元素矢量/INT8 点积
- 复杂 18bx18b 乘法器
- 单精度浮点

如需了解有关 PL 资源的更多信息，请参阅《Versal 自适应 SoC 可配置逻辑块架构手册》(AM005)、《Versal 自适应 SoC 存储器资源架构手册》(AM007) 以及《Versal 自适应 SoC DSP 引擎架构手册》(AM004)。

NoC

片上网络 (NoC) 属高速通信子系统，可在 PL、PS 和其他集成块中的 IP 端点之间传输数据，以提供统一的裸片内部连接。NoC 主接口和从接口可配置为 AXI3、AXI4 或 AXI4-Stream。NoC 将这些 AXI 接口转换为 128 位宽的 NoC 数据包协议，分别通过水平 NoC (HNoC) 和垂直 NoC (VNoC) 在器件上进行横向和纵向数据移动。HNoC 在 Versal 自适应 SoC 底部和顶部运行，靠近 I/O bank 和集成块（例如，处理器、存储器控制器和 PCIe）。 VNoC 数量（最多 8 个 VNoC）取决于器件和 DDRMC 的数量（最多 4 个 DDRMC）。对于使用堆叠硅片互联 (SSI) 技术的 Versal 器件，超级逻辑区域 (SLR) 之间的 NoC 使用 NoC 裸片间桥接 (NIDB) 来进行连接。如需了解有关 AXI 协议的更多信息，请参阅《Vivado Design Suite：AXI 参考指南》(UG1037)。

NoC 必须在早期启动时以及使用 NoC 数据路径前，通过 NoC 编程接口 (NPI) 完成配置或编程。NPI 用于对 NoC 寄存器进行编程，这些寄存器可定义布线表、速率调制和 QoS 配置。通过 NPI 对 NoC 进行编程通常无需用户干预。编程完全由平台管理控制器 (PMC) 嵌入式 NPI 控制器自动执行。如需了解有关启动和配置的更多信息，请参阅《Versal 自适应 SoC 技术参考手册》(AM011)。

Versal 自适应 SoC NoC IP 充当 Versal 自适应 SoC NoC 的逻辑表示法。NoC 主要用于在 DDR 控制器与器件其余部分之间高效移动数据。Versal 自适应 SoC NoC IP 支持多个主控制器通过高级服务质量 (QoS) 设置来访问共享 DDRMC。AXI NoC IP 是将 PS 或 PL 连接到 DDRMC 时所不可或缺的工具。AXI NoC IP 还可用于在 PS 与 PL 之间或在 PL 内的设计模块之间创建其他连接。

如需了解有关 NoC IP 和性能的更多信息，请参阅《Versal 自适应 SoC Programmable Network on Chip and Integrated Memory Controller LogiCORE IP 产品指南》(PG313)。

XPIO

Versal 自适应 SoC XPIO 与 AMD UltraScale™ 架构中的高速 I/O (HPIO) 相似。但 XPIO 是位于器件底部和/或顶部的外设，这与先前器件中的 I/O 列式布局不同。XPIO 所提供的 XPHY 逻辑与 UltraScale 器件原生模式类似。XPHY 逻辑可将经过校准的延迟与串行逻辑和解串逻辑封装在一起，以提供 6 个单端 I/O 端口（称为半字节）。每个 XPIO bank 含 9 个 XPHY 逻辑站点 (site)，支持多达 54 个单端 I/O 端口。XPHY 逻辑用于集成 DDRMC、软核存储器控制器和定制高性能 I/O 接口。如需了解有关 XPIO 的更多信息，请参阅《Versal 自适应 SoC SelectIO 资源架构手册》(AM010)。

适用于 DDR4、LPDDR4 和 LPDDR4X 的 DDRMC

DDRMC 是一种高效、低延时集成 DDR 存储器控制器，适用于包括通用中央处理器 (CPU) 以及其他传统的现场可编程门阵列 (FPGA) 应用在内的各种应用，如视频或网络缓冲等。

该控制器的运行时钟频率为 DRAM 时钟频率的一半，支持 DDR4、LPDDR4 和 LPDDR4X 标准，最高可达 4266 Mb/s。该控制器可配置为单一 DDR 存储器接口，数据宽度为 16、32 和 64 位，启用纠错码 (ECC) 后另加 8 个校验位。它也可以配置为 2 个独立或交织式 DDR 接口，每个接口各含 16 个或 32 个数据位。它支持 x4、x8、和 x16 DDR4 和 x32 LPDDR4 组件、小型双列直插式存储器模块 (SODIMM)、无缓冲 DIMM (UDIMM)、寄存式 DIMM (RDIMM) 和低负载 DIMM (LRDIMM)。DDRMC 可通过 NoC 访问。您可使用《Versal 自适应 SoC 外部存储器预规划工具》(XTP667) 来确定存储器接口的各种宽度、类型和速度的最佳组合。如需了解其他信息，请参阅 AMD GitHub 仓库中提供的存储器管脚分配教程。

在 Versal 自适应 SoC 中，DDRMC 是适用于整个系统的共享资源。在 PS 和 PL 之间通过遍布整个器件的高性能 NoC 接口来共享 DDRMC。NoC IP 核可配置为包含 1 个或多个集成 DDRMC。如果选择 2 个或 4 个 DDRMC，则可通过对 DDRMC 进行分组来形成单一交织式存储器。在交织模式下，该应用会将其中包含的 DDRMC 视为单一统一存储器块。NoC 通过自动将 AXI 请求划分为块大小的交织式子请求，并将子请求交替发送到其中包含的每个 DDRMC，以支持跨 2 个或 4 个 DDRMC 实现交织。

重要提示！您必须使用 NoC 在 PL、PS、CPM 或 AI 引擎和 DDRMC 之间进行连接。

如需了解有关 DDRMC 的更多信息，请参阅《Versal 自适应 SoC Programmable Network on Chip and Integrated Memory Controller LogiCORE IP 产品指南》(PG313)。

注释：Versal 自适应 SoC 还支持 PL 互连结构中的软核存储器控制器，与先前的器件系列相似。

HBM

Versal 高带宽存储器 (HBM) 控制器根据所选器件，支持访问一个或两个栈，前者对应 4 个高堆叠器件支持最高 128 Gb (16 GB)，后者对应 8 个高堆叠器件支持最高 256 Gb (32 GB)。8 个独立 HBM 控制器连接到单个栈。每个 HBM 控制器均支持 2 条伪通道，每条通道都大部分独立，并用于对一个专用的 HBM 段进行寻址。每条伪通道都有一个位宽为 64 位的数据总线，含共享的命令/地址/控制 (CAC) 总线。每个 HBM 栈的总数据位宽为 1,024 位，平均分布于 16 条伪通道间。对于 3,200 MT/s 的数据传输率，控制器和 PHY 的最高工作频率为 1,600 MHz。在每个 HBM 控制器 128 位、每个栈 8 个控制器且大部分器件含 2 个栈的前提下，可产生最大 819 GB/s 的吞吐量。

HBM 控制器通过 NoC 对接到 PL 中的用户逻辑。每条伪通道都有一对专用 NoC 从端口。对于 HBM 伪通道构成的每个四通道，都会添加一个 8x8 开关。NoC 与这些开关相结合，即可支持从连接到 NoC 的任意主接口对整个 HBM 栈进行全局寻址。NoC IP 核可配置为包含一小部分或者所有集成 HBM 控制器。

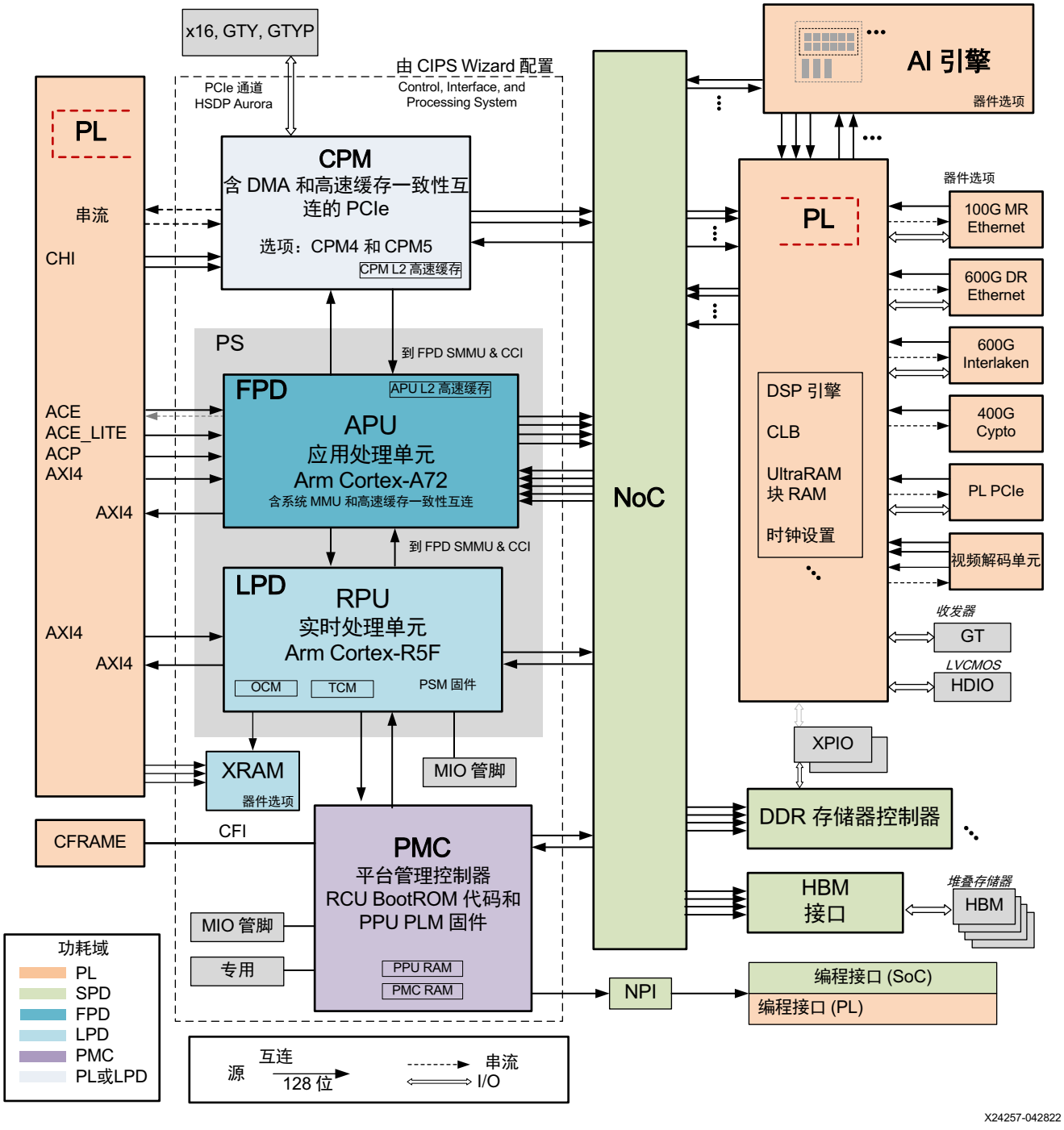
如需了解有关 HBM 控制器的更多信息，请参阅《Versal 自适应 SoC Programmable Network on Chip and Integrated Memory Controller LogiCORE IP 产品指南》(PG313)。

适用于 PS、PMC 和 CPM 的 CIPS

如下图所示，PS 模块、PMC 模块与 CPM 模块组合在一起，并使用 Control, Interface, and Processing System (CIPS) IP 核进行配置。

注释：Versal 自适应 SoC 包含多个功耗域。在 PS 中，RPU 位于低功耗域 (LPD) 中，APU 位于全功耗域 (FPD) 中，而平台管理控制器 (PMC) 则位于 PMC 功耗域中。根据目标器件功能，CPM 有两种实现方式：符合 PCI Express 基本规范修订版 4.0 的 CPM4，以及符合 PCI Express 基本规范修订版 5.0 的 CPM5。CPM4 完全由 PL 域供电，而 CPM5 由其专用电源 (VCC_CPM5) 和 PS LPD 供电。如需了解有关功耗域的更多信息，请参阅《Versal 自适应 SoC 技术参考手册》(AM011)。

图 2：**器件级互连架构**



PS

处理器系统 (PS) 包含应用处理单元 (APU)、实时处理单元 (RPU) 和外设。通过遍布整个器件的高性能 NoC 接口在 PS 和 PL 之间共享 DDRMC。

APU

应用处理单元 (APU) 含连接到 1 MB 统一 L2 高速缓存的双核 Arm® Cortex®-A72 处理器。此 APU 专为无需实时性能的系统控制和计算密集型应用而设计。要提高 Versal 自适应 SoC 性能，就需要提高存储器子系统的性能。为了帮助满足这些要求，Versal 自适应 SoC 包含更大的 L1 指令高速缓存（32 KB 至 48 KB）以及多个 DDRMC 和 NoC，用于提高主存储器的性能。

下表显示了 AMD Zynq™ UltraScale+™ MPSoC 中的 Cortex-A53 与 Versal 自适应 SoC Cortex-A72 处理器的区别。

表 2：**Cortex-A53 与 Cortex-A72 之比较**

Cortex-A53	Cortex-A72	Versal 自适应 SoC 优势
Arm v8A 架构（64 位和 32 位操作）		无需更改应用代码
EL0-EL3 异常级别		
安全/非安全操作		
高级 SIMD NEON 浮点单元		
集成存储器管理器		
电源岛控制		
上限 1500 MHz	上限 1700 MHz	频率更高
每 MHz 2.23 DMIPS	每 MHz 5.74 DMIPS	原始性能提高 2 倍（根据 Arm 标准测试）
3.65 SPEC2006int	6.84 SPEC2006int	
2 路超标量	3 路超标量	指令周期更有效
按序执行	无序执行	性能更高且存储器停滞更少
功耗效率高	功耗效率更高	功耗降低 20%
8 阶流水线	15 阶流水线	排队的指令更多，执行的指令更多
条件分支预测	两级分支预测	缓存命中率更高，存储器提取更少

RPU

实时处理单元 (RPU) Arm Cortex-R5F 处理器的时钟频率高于 Zynq UltraScale+ MPSoC。Versal Arm Cortex-R5F 处理器支持矢量浮点 (Vector Floating-Point) v3 (VFPv3)，而 Zynq UltraScale+ MPSoC Arm Cortex-R5F 处理器则支持 VFPv2。

标准外设

Versal 自适应 SoC 标准 I/O 外设位于低功耗域 (LPD) 和 PMC 中。NoC 必须配置为支持访问 DDRMC，以便具有直接存储器访问 (DMA) 的外设能够访问 DDR 存储器接口。

下表显示了 Zynq UltraScale+ MPSoC 中的标准外设与 Versal 自适应 SoC 中的标准外设之间的区别。

表 3：**标准外设对比**

外设	Zynq UltraScale+ MPSoC	Versal 自适应 SoC
CAN，CAN-FD	2 个具有标准 CAN 的控制器	2 个控制器，含控制器区域网络 - 灵活数据速率 (CAN-FD)
GEM	4 个控制器	2 个控制器，含时间敏感网络 (TSN) 功能
GPIO	1 个控制器	2 个控制器

表 3：**标准外设对比** (续)

外设	Zynq UltraScale+ MPSoC	Versal 自适应 SoC
I2C	2 个控制器	2 个控制器，位于 LPD 内（通用） 1 个控制器，位于 PMC 内（通用）
NAND	1 个控制器	不适用
PCIe（Gen1，Gen2）	1 个控制器	不适用
PCIe（Gen3，Gen4）	1 个控制器	随器件而异
SPI	2 个控制器	2 个控制器
SATA	1 个控制器	不适用
UART	2 个具有标准 UART 的控制器	2 个具有服务器基础系统架构 (SBSA) 的控制器
USB（主机、器件、双功能器件）	2 个 USB 2.0/3.0 控制器	1 个 USB 2.0 控制器

AMBA 规格接口

如下表所示，Versal 自适应 SoC 中的 PS-PL Arm 高级微控制器总线架构 (AMBA) 规格接口具有与 Zynq UltraScale+ MPSoC 相似的功能。

注释：在 LPD、FPD 和 PL 中启用和禁用不同的功耗域将启用和禁用与这些域的 AXI 连接。

重要提示！由于通过遍布整个器件的高性能 NoC 接口在 PS 和 PL 之间共享 DDRMC，因此 PS-PL AXI 互连数量较少。

表 4：**AMBA 接口比较**

PS-PL AMBA 接口	主接口	一致性	Zynq UltraScale+ MPSoC		Versal 自适应 SoC	
			名称	数量	名称	数量
加速器一致性端口 (ACP)	PL	I/O	S_AXI_ACP_FPD	1	S_ACP_FPD	1
AXI 一致性扩展 (ACE)	PL	两路	S_AXI_ACE_FPD	1	S_ACE_FPD	1
PL-to-FPD AXI	PL	-	S_AXI_HPx_FPD	4	S_AXI_HP	1
PL-to-FPD AXI	PL	I/O	S_AXI_HPCx_FPD	2	S_AXI_HPC	1
PL-to-LPD AXI	PL	-	S_AXI_LPD	1	S_AXI_LPD	1
FPD-to-PL AXI	FPD	-	M_AXI_HPMx_FPD	2	M_AXI_FPD	1
LPD-to-PL AXI	LPD	-	M_AXI_HPM0_LPD	1	M_AXI_LPD	1

PMC

平台管理控制器 (PMC) 子系统含以下功能：

- 启动和配置管理
- Dynamic Function eXchange (DFX)
- 功耗管理
- 可靠性和安全功能

- 生命周期管理，包括器件完整性、调试和系统监控
- I/O 外设

PMC 块可通过执行 BootROM 与 Platform Loader and Manager (PLM) 来对处理器系统、CPM、PL、NoC 寄存器初始化和设置以及 I/O 和中断配置设置的启动和配置操作进行处理。除了启动和配置外，PLM 还可提供生命周期管理服务。与先前器件相比，PMC 总线架构和集中集成可显著提高配置速度和回读性能。下表显示了 Zynq UltraScale+ MPSoC 块与 Versal 自适应 SoC 块的比较结果。

表 5：**块的比较**

Zynq UltraScale+ MPSoC	Versal 自适应 SoC
配置安全单元 (CSU) 和平台管理单元 (PMU)	PMC
CSU	ROM 代码单元 (RCU)
PMU	平台处理单元 (PPU)
第一阶段启动加载程序 (FSBL) 和 PMU 固件	PLM

如需了解有关 PMC 的更多信息，请参阅《Versal 自适应 SoC 技术参考手册》(AM011)。如需了解有关 PLM 的更多信息，请参阅《Versal 自适应 SoC 系统软件开发者指南》(UG1304)。

闪存控制器

PMC 包括 3 种类型的闪存控制器，每一种都可以用作启动器件，也可供应用使用。下表显示了 Zynq UltraScale+ MPSoC 中的闪存控制器与 Versal 自适应 SoC 中的闪存控制器之间的区别。

表 6：**闪存控制器对比**

外设	Zynq UltraScale+ MPSoC	Versal 自适应 SoC
八通道 SPI (OSPI)	不适用	1 个控制器
四通道 SPI (QSPI)	1 个控制器	1 个不支持线性地址模式的控制器
SD/eMMC	2 个控制器	2 个具有相同功能和更新版 DLL 的控制器

注释： Versal 自适应 SoC 可以支持辅助启动模式（如以太网、USB 等）。欲知详情，请参阅《Versal 自适应 SoC 系统软件开发者指南》(UG1304)。

CPM

Versal 架构包括多个块，用于实现基于 PCI®-SIG 技术的高性能标准接口。在包含 CPM 的 Versal 自适应 SoC 中，CPM 遵循服务器系统方法论来为设计提供主要接口。作为 Versal 架构集成 shell 的一部分，CPM 与 NoC 之间存在专用连接，通过该连接可以访问 DDR 和其他硬化 IP。CPM 与可编程逻辑分开配置，使集成 shell 在启动后即可快速运行，而无需配置 PL。这种单独配置方法解决了 PCIe 规范带来的常见上电和复位时序问题。CPM 有 2 种实现方式：CPM4 和 CPM5。

在具有可用 CPM4 的 Versal 自适应 SoC 中，此块符合 PCIe 基本规范修订版 4.0，并能够支持已定义的最大 16 GT/s 的线速率。CPM4 包含 2 个 PCIe 控制器（共享访问 16 个 GTY 收发器），并集成了与 CPM PCIe 控制器 #0 相关的单一直接存储器访问 (DMA) 控制器功能（用户可以选择 QDMA 或 XDMA）。CPM4 中的加速器高速缓存一致性互连 (CCIX) 支持符合 CCIX 基本规范修订版 1.0。

在具有可用 CPM5 的 Versal 自适应 SoC 中，此块符合 PCIe 基本规范修订版 5.0，并能够支持已定义的最大 32 GT/s 的线速率。CPM5 包含 2 个 PCIe 控制器，这些控制器具有对 16 个 GTYP 收发器的专享访问权。CPM5 集成了 2 个 DMA 控制器（均为 QDMA），每个都与 CPM PCIe 控制器 #0 和 CPM PCIe 控制器 #1 相关联。CPM5 中的 CCIX 支持符合 CCIX 基本规范修订版 1.1。

CPM4 和 CPM5 包括以下附加组件：

- 一致性网状网络 (CMN) 可构成基于 Arm CoreLink CMN-600 的 CCIX 块。
- 有 2 个一致性集线器接口 (CHI) PL 接口 (CPI) 块。CPM4 有 1 个 L2 高速缓存实例，CPM5 有 2 个 L2 高速缓存实例。CPI 块与 PL 中的加速器连接，并执行 512 到 256 位的数据宽度转换，并与内部核块执行时钟域交汇。
- 非一致性互连块，与 PS 连接以访问 NoC 和 DDRMC。此互连结构通过高级外设总线 (APB) 或 AXI 从接口连接到所有其他子块，以进行配置。
- 时钟/复位块，包括锁相环 (PLL) 和时钟分频器。

CPM 可用性与器件相关。如需了解更多信息，请参阅《Versal 架构和产品数据手册概述》(DS950)。如需了解有关 CPM 的更多信息，请参阅《Versal 自适应 SoC CPM CCIX 架构手册》(AM016)、《Versal 自适应 SoC CPM Mode for PCI Express 产品指南》(PG346) 和《Versal 自适应 SoC CPM DMA and Bridge Mode for PCI Express 产品指南》(PG347)。

注释： Versal 自适应 SoC 还支持在 PL 互连结构中基于 PCI-SIG 技术来实现子系统，与先前器件系列相似。

GT

GT 为高速接口提供了多种协议，如以太网和 Aurora IP。Versal 自适应 SoC 采用了 XPIPE 机制，以在 PCIe 块与 GT 之间建立高速连接。XPIPE 和 GT 在基于 PL 的 IP 与基于 PS 的 IP（如 CPM4、以太网、用于调试的 Aurora 链路等）之间共享。对于 Versal 自适应 SoC，GT 组件的粒度已从公共/通道更新为四通道。如需了解有关 GT 的更多信息，请参阅《Versal 自适应 SoC Transceivers Wizard LogiCORE IP 产品指南》(PG331)、《Versal 自适应 SoC GTY 和 GTYP 收发器架构手册》(AM002) 和《Versal 自适应 SoC GTM 收发器架构手册》(AM017)。如需获取有关 CPM5 的 GT 选择和管脚分配指导信息，请访问此链接以参阅《Versal 自适应 SoC CPM DMA and Bridge Mode for PCI Express 产品指南》(PG347) 中的相关信息。

HSDP

Versal 自适应 SoC 的异构性质和性能需要具备系统级的高带宽调试和追踪解决方案。高速调试端口 (HSDP) 是 Versal 自适应 SoC 中的一项新功能，能够对被测器件 (DUT) 中各种集成互连结构处理器块进行统一的快速调试和追踪。HSDP 提供了通过专用 Aurora 接口和高速调试电缆（如 SmartLynq+）来执行调试和走线捕获的选项。对于通过 PCIe 连接到主机的远程系统，也可以通过 PCIe 进行高速调试。HSDP 功能可通过基于 GT 的高速接口访问，如 PS 块中的集成 Aurora 接口或 CPM 块中的 PCIe 接口。

如需了解更多信息，请参阅以下资料：

- 请访问此链接以参阅《Versal 自适应 SoC 技术参考手册》(AM011) 中的相应内容
- 《SmartLynq+ 模块用户指南》(UG1514)
- 利用 SmartLynq+ 模块搭配高速调试端口 (HSDP) 完成系统设计的示例

高速连接和加密集成 IP

MRMAC

Versal 自适应 SoC Multirate Ethernet MAC (MRMAC) 可提供高性能低时延的以太网端口，支持广泛的自定义和统计数据收集功能。MRMAC 支持 IEEE 标准所定义并要求的下列前向纠错 (FEC)：适用于 25/50/100GE NRZ 支持的第 91 条 RS(528, 514) KR4 FEC、适用于 50/100GE PAM4 支持的第 91 条 RS(544, 514) KP4 FEC 以及适用于 10/25/40/50GE 低时延支持的第 74 条 FEC。MRMAC 具有丰富的旁路模式，用于支持访问仅限 FEC 模式（面向定制协议）和 FEC +PCS（面向协议测试器）。如需了解更多信息，请参阅《Versal 器件 Integrated 100G Multirate Ethernet MAC (MRMAC) LogiCORE IP 产品指南》(PG314)。

注释：MRMAC 可用性与器件相关。

DCMAC

该 Versal 自适应 SoC 600G Channelized Multirate Ethernet Subsystem (DCMAC) 属于高性能、自适应、以太网集成硬核 IP，适用于多种客户联网应用。此块可配置为最多 6 个端口，这些端口含独立 MAC 和 PHY 功能，符合 IEEE 标准 MAC 速率（从 100GE 到 400GE），并且总体最大带宽达 600 Gb/s。此 IP 支持各种 FEC 和 IEEE 1588 网络测量和控制系统的精密时钟同步协议标准 (IEEE 1588) 硬件时间戳功能。不仅如此，此 IP 还可配置为提供 600 Gb/s 的 MAC 处理能力，适用于最多 40 条用户定义带宽的通道。欲知详情，请参阅《Versal 自适应 SoC 600G Channelized Multirate Ethernet Subsystem (DCMAC) 产品指南》(PG369)。

Interlaken

Versal 自适应 SoC Integrated 600G Interlaken with FEC (ILKNF) 属于高性能自适应 Interlaken 集成硬核 IP，适用于多种客户联网应用。此块可配置为单个 Interlaken 端口，带宽高达 600 Gb/s，支持各种通道计数和通道速率配置。此 IP 支持 FEC，可通过高速收发器通道搭配 Interlaken 使用。此外，FEC 逻辑无需 Interlaken 即可使用，允许该核支持任意数量的协议，包括以太网。欲知详情，请参阅《Versal 自适应 SoC 600G Interlaken LogiCORE IP 产品指南》(PG371)。

高速加密技术

Versal 自适应 SoC Integrated 400G High Speed Channelized Cryptography Engine 子系统（HSC 子系统）属于高性能自适应加密集成硬核 IP，适用于多种客户加密应用。此块可配置为最多 4 个端口，速率范围为 100 Gb/s 到 400 Gb/s，总体最大带宽达 400 Gb/s。此 IP 支持 MACsec、IPsec 和批量加密方法。不仅如此，此 IP 还可配置为提供 400 Gb/s 的加密处理能力，适用于最多 40 条用户定义带宽的通道。欲知详情，请参阅《Versal 自适应 SoC 400G High Speed Channelized Cryptography Engine Subsystem LogiCORE IP 产品指南》(PG372)。

第 3 章

系统规划

要正确规划系统，您必须根据自己的目标应用或系统设计类型明确系统要求。其中包括识别具有正确特性（例如，DDRMC IP 数量、AI 引擎等）的相应 AMD Versal™ 器件。

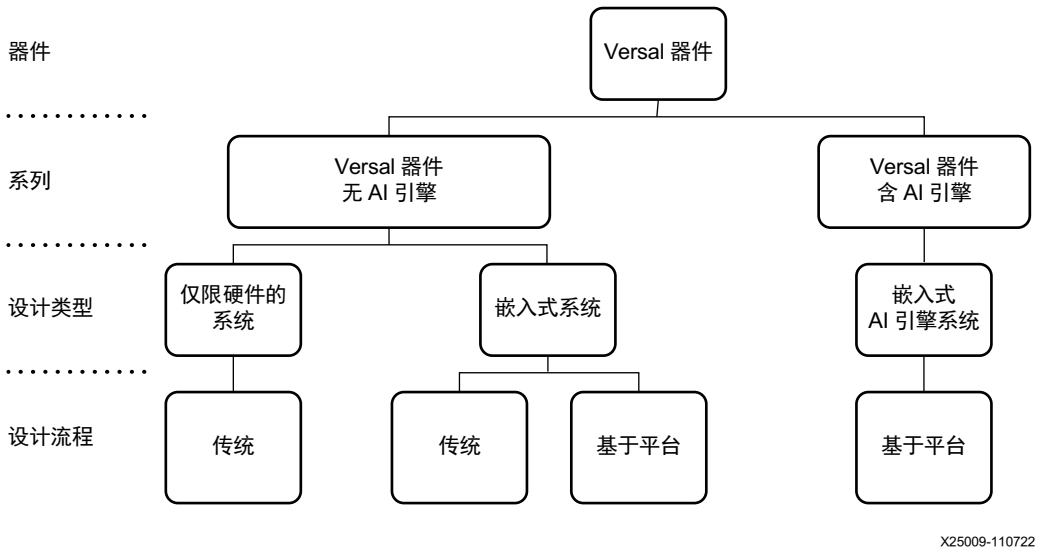
注释：如需了解有关系统规划方法论建议的信息，请参阅《Versal 自适应 SoC 系统和解决方案规划方法指南》(UG1504)。

系统设计类型

AMD Versal™ 自适应 SoC 属于异构计算平台，具有多个计算引擎。在 Versal 自适应 SoC 上可映射各种应用，包括对无线系统、机器学习推断和视频处理算法进行信号处理。除了多个计算引擎外，Versal 自适应 SoC 还可使用高速串行 I/O、片上网络 (NoC)、DDR4/LPDDR4 存储器控制器、HBM 控制器和多重速率以太网媒体访问控制器 (MRMAC) 来提供超高系统带宽。Versal 器件分类为以下几个系列：Versal Prime、Premium、HBM、AI Core 和 AI Edge。下图显示了每种 Versal 器件系列所支持的不同系统设计类型和设计流程。

注释：Versal Prime 系列、Premium 系列和 HBM 系列的设计流程与 AMD FPGA 所使用的流程类似。Versal AI Core 系列、AI Edge 系列以及 Versal Premium VP2502 和 VP2802 器件的设计流程要求您面向异构计算平台进行设计，此平台具有特殊的硬件配置和软件支持要求。

图 3：**系统设计类型**



下表显示了每种 Versal 器件系列所支持的系统设计类型和设计流程。如该表中所示，大部分设计流程都以构建平台为基础。

表 7：**系统设计类型**

设计类型	器件系列	设计流程	平台源文件	GitHub 示例
仅限硬件的系统	Versal Prime 系列 Versal Premium 系列 Versal HBM 系列	传统	不适用	Versal 器件架构教程
嵌入式系统	Versal Prime 系列 Versal Premium 系列 Versal HBM 系列	传统	不适用	Versal 自适应 SoC 嵌入式设计教程
		基于平台	定制	Versal Prime 系列 VMK180 目标参考设计
嵌入式 AI 引擎系统	Versal AI Core 系列 Versal AI Edge 系列 Versal Premium VP2502 器件和 VP2802 器件	基于平台	定制	AI 引擎开发设计教程 VCK190 基本 TRD

提示：请访问 GitHub 以获取更多示例，这些示例会定期更新。

以下提供了每种系统设计类型的汇总信息：

- 仅限硬件的系统：可编程逻辑设计。使用传统设计流程创建此系统。
- 嵌入式系统：嵌入式处理器系统，软件在 Arm® Cortex®-A72 或 Cortex-R5F 处理器上运行，硬件内容则位于 PL 内。使用传统设计流程或基于平台的设计流程创建此系统。
- 嵌入式 AI 引擎系统：嵌入式处理器系统，软件在 Arm Cortex-A72 或 Cortex-R5F 处理器上运行，硬件内容位于 PL 内，算法内容则位于 AI 引擎内。使用基于平台的设计流程创建此系统。

Versal 自适应 SoC 的设计流程如下所示：

- 传统设计流程：在传统设计流程中，系统的整个 PL 部分都是在单个 AMD Vivado™ 工程中定义的。该工程必须包括 Versal 基础硬件 IP 块（例如，Control, Interface, and Processing System (CIPS)、NoC、I/O 控制器）以及工程所需的任何其他定制 RTL 和 IP 块。设计源文件将添加到 Vivado 工具中，并通过 Vivado 实现流程进行编译。如果系统仅包含 PL 组件，那么可使用 Vivado 工具来生成可编程器件镜像 (PDI)，以便对 Versal 器件进行编程。如果系统还包含嵌入式软件内容，那么将在从 Vivado 工具导出的固定硬件设计上的 AMD Vitis™ 环境中开发软件应用。此流程类似于用于 AMD Zynq™ UltraScale+™ MPSoC 的传统流程。
- 基于平台的设计流程：在基于平台的设计流程中，硬件系统分为下列不同元素：可复用的基本平台以及基本硬件扩展，此平台是在 Vivado 中开发的，而扩展则是在 Vitis 中通过基本平台的可扩展区域内精确定义的一组连接接口来开发的。大部分硬件设计是在 Vivado 中开发的，但设计中以 C++ 而非硬件描述语言 (HDL) 指定的部分大多是在 Vitis 中自然开发并集成的。后者示例包括 AI 引擎 graph 与内核以及以通过高层次综合 (HLS) 编译的 PL 作为目标的内核函数。

  您可根据自身工作效率来选择任一设计分区方式：基本平台或可扩展区域。在整个设计周期过程中，基本硬件和可扩展区域均可进化，精心设计的基本平台能为多种应用奠定基础，以便 Vitis 工具在其中对可扩展区域进行扩展。相应开发团队可通过合理的松散耦合与紧密耦合将设计内容从 Vivado 导出到 Vitis，反之亦然，这有助于促进组成异构系统的不同元素的并发开发和集成。

第 4 章

设计流程

AMD Versal™ 自适应 SoC 支持 2 种设计流程：传统设计流程和基于平台的设计流程。要充分利用 Versal 自适应 SoC 资源，重要的是选择正确的设计流程。下表显示了根据设计类型和目标器件系列所使用的设计流程。

表 8：**设计流程**

设计类型	器件系列	设计流程
仅限硬件的系统	Versal Prime 系列 Versal Premium 系列 Versal HBM 系列	传统
嵌入式系统	Versal Prime 系列 Versal Premium 系列 Versal HBM 系列	传统
		基于平台
嵌入式 AI 引擎系统	Versal AI Core 系列	基于平台

注释：如需了解有关设计类型的更多信息，请参阅《Versal 自适应 SoC 系统和解决方案规划方法指南》(UG1504)。

传统设计流程

面向仅限硬件系统的传统设计流程

如果您的设计仅包含 PL 组件（仅含 RTL 和 IP），那么您可使用 AMD Vivado™ 工具来生成可编程器件镜像 (PDI)，以便用于对 Versal 器件进行编程。与先前架构类似，设计源被添加到 Vivado 工具中，并通过 Vivado 实现流程来进行编译。

重要提示！平台管理控制器 (PMC) 整合到 CIPS IP 中，必须对其加以配置才能使 Versal 器件正确启动。因此，所有 Versal 器件设计必须包含 CIPS IP。

以下另提供了其他重要注意事项：

- 硬化的 DDR 存储器控制器和 HBM 控制器只能通过 NoC IP 来访问。要使用 DDRMC 或 HBM 控制器，您的设计必须包含 NoC IP。
- 硬件调试核默认情况下通过 CIPS IP 来连接。JTAG 仍可用，但不再作为首选流程。您必须熟悉硬件调试连接和流程方面的更改。

您必须使用 Vivado IP integrator 来例化、配置和连接 CIPS IP、NoC/DDRMC IP 以及硬件调试 IP，才能在设计变更迭代过程中充分利用块设计自动化。Vivado IP integrator 还可为 GT IP 和连接 IP（如 MRMAC IP）提供特殊支持，从而简化基于 GT 的设计创建和 I/O 管脚分配。

您可使用定制封装 IP、RTL 模块参考的块以及 IP 目录提供的其他 IP，将完整设计与 Vivado IP integrator 集成。或者，您可使用 Vivado IP integrator 来配置并连接关键 Versal 自适应 SoC IP（例如，CIPS IP 和 NoC/DDR IP），然后在 RTL 设计中例化生成的块设计。欲知详情，请访问此链接以参阅《Vivado Design Suite 用户指南：采用 IP integrator 设计 IP 子系统》(UG994) 中的相应内容。

注释：仅在工程模式下支持 Vivado IP integrator。

重要提示！此设计流程不支持对 AI 引擎核进行编程，因此仅适用于 Versal Prime 器件、Versal Premium 器件和 Versal HBM 器件。

相关信息

系统调试

面向嵌入式系统的传统设计流程

您也可以使用传统设计流程来创建同时包含 PL 组件和嵌入式软件组件的设计。在此情况下，流程与用于 AMD Zynq™ UltraScale+™ MPSoC 的嵌入式软件设计流程相似。硬件团队负责创建、验证和实现硬件设计，以供软件团队用于开发嵌入式软件应用。

注释：适用于面向仅限硬件系统的传统设计流程的所有建议都同样适用于面向嵌入式系统的传统设计流程。

以下是此流程中的主要步骤：

1. 使用 Vivado IP integrator 创建并验证硬件设计。
2. 使用 Vivado 实现工具来实现硬件设计。
3. 将硬件设计导出至 Vitis 嵌入式软件开发流程。
4. 在固定硬件设计上使用 Vitis 嵌入式软件开发流程开发软件应用。

注释：仅在工程模式下支持 Vivado IP integrator。

重要提示！此设计流程不支持对 AI 引擎核进行编程，因此仅适用于不含 AI 引擎的 Versal Prime 器件、Versal Premium 器件和 Versal HBM 器件。

基于平台的设计流程

在基于平台的设计流程中，硬件设计按概念分为 2 个不同要素：平台和处理器系统。平台包含基本 Versal IP 块（包括 CIPS、NoC、AI 引擎和 Clocking Wizard）和开发板接口 IP 块（包括高速 I/O 和存储器控制器）。处理器系统包含特定于应用的系统部分，这部分由可编程逻辑与 AI 引擎块组成。此平台为可扩展平台，因为它不含可编程逻辑的全部内容。而是改为通过添加处理器系统来扩展此平台。

下面是此流程中的主要步骤。前 3 个步骤可以并行完成。您可在最终完成固定硬件平台后单独更新 AI 引擎程序。

1. 使用 Vivado IP integrator 和 RTL 代码开发硬件平台。
2. 使用 Vitis 工具开发 AI 引擎 graph 和内核。

   注释：仅当使用 Versal AI Core 系列和 AI Edge 系列时，AI 引擎才可用。
3. 使用 Vitis 工具（C++ 内核）或 Vivado 工具（RTL 内核）开发 PL 内核。

4. 汇编 AI 引擎程序和 PL 内核以构成处理器系统，并使用 Vitis 连接器将处理器系统与平台相集成以创建固定硬件设计。
5. 使用 Vivado 工具在固定硬件设计上实现和执行设计收敛。
6. 在固定硬件设计上使用 Vitis 嵌入式软件开发流程开发软件应用。

注释：仅在工程模式下支持 Vivado IP integrator。

重要提示！这是支持对 AI 引擎核进行编程的唯一流程，因此对于 Versal AI Core 器件、AI Edge 器件或 Premium 器件而言，此流程是必需的。

提示：AMD 为 Versal 自适应 SoC 评估套件（如 VCK190）提供现成的平台。

相关信息

在设计流程中使用 Vivado 工具
Vitis 环境设计方法论

在设计流程中使用 Vivado 工具

Vivado Design Suite 是所有 Versal 自适应 SoC 设计流程中的关键组件。基于您的设计流程，Vivado 工具的主要使用模型如下所示：

- 传统设计流程
  - 创建 RTL 和 IP 设计
- 基于平台的设计流程
  - 创建和封装 RTL 内核，以供在 Vitis 环境设计流程中使用
  - 创建并生成平台，以供在 Vitis 环境设计流程中使用

重要提示！如果您使用的是基于平台的设计流程，AMD 提供了标准平台作为起点，而 Vivado IP integrator 则可通过自定义和重新生成平台来更好地满足目标系统应用的需求。欲知详情，请参阅 AMD 网站上的下载页面的“Vitis 嵌入式平台”选项卡中的内容。

您可使用 Vivado 工具来执行设计创建、实现和 PL 分析。典型任务包括：

- 逻辑仿真
- 约束定义和时序分析
- NoC 编译
- I/O 管脚分配和时钟规划
- 逻辑综合与实现
- 设计逻辑可视化
- 设计规则检查 (DRC) 和设计方法论检查
- 实现结果分析
- 功耗和散热分析

- 编程和调试

创建 RTL 和 IP 设计

Vivado 工具支持传统 RTL 和 IP 设计流程，Vivado IP integrator 可用于自动执行设计装配。RTL 开发者必须了解 Versal 自适应 SoC 中可用的新 IP 及其使用要求，包括：

- 所有设计都需要 CIPS IP，包括用于启动器件的 PMC。CIPS IP 也可用于配置 PS 外设和 SYSMON IP。欲知详情，请参阅《Control, Interface and Processing System LogiCORE IP 产品指南》(PG352)。
- 访问器件上的 DDRMC 的唯一途径是通过 NoC IP。欲知详情，请参阅《Versal 自适应 SoC Programmable Network on Chip and Integrated Memory Controller LogiCORE IP 产品指南》(PG313)。
- 硬件调试流程与先前器件不同。如需了解更多信息，请参阅《Vivado Design Suite 用户指南：编程和调试》(UG908)。

创建和封装 RTL 内核

Vivado 工具可用于封装 RTL 内核，以供 Vitis 连接器使用。该选项在 Vivado IP 封装器中提供，可用于将 IP 封装到 XO 文件中，并使用 Vitis 系统连接器将其连接到最终设计。如需了解有关 RTL 内核的更多信息（包括相关限制），请访问此链接以参阅《Vitis 统一软件平台文档：应用加速开发》(UG1393) 中的相应内容。

建议：AMD 建议 RTL 开发者在使用 Vitis 环境设计流程时，使用此方法来整合现有逻辑。

创建和生成平台

您可使用 Vivado 工具创建可扩展硬件平台，随后可使用 Vitis 工具通过处理器系统来扩展此平台。此平台通常包含基本系统级资源，供所有加速器共享，如 PS、NoC、DDRMC 和基准 I/O。如需了解有关硬件平台定义的更多信息，请参阅《Vitis 统一软件平台文档：应用加速开发》(UG1393)。

注释：这是支持使用 AI 引擎资源的唯一设计流程。

AMD 建议：

- 仅在平台中包含基本 Versal 自适应 SoC 块和开发板接口 IP
  - 基本块：CIPS、NoC、AI 引擎、Clocking Wizard、中断控制器
  - 接口块：高速 I/O、存储器控制器
- 对于通过串流接口或存储器映射接口以及其他平台串流接口与 AI 引擎进行交互的 RTL 模块，请将这些模块映射到 Vitis RTL 内核

以下是此方法的优点：

- 确保平台的高可复用性
- 促进任务分发
- 提升集成流程自动化的能力
- 增加 DFX 的范围和机会

块设计流程

对于以 Versal 自适应 SoC 为目标的设计，AMD 需要使用 Vivado IP integrator。平台管理控制器 (PMC) 包含在 CIPS IP 中，可用于启动和配置 Versal 器件。Versal CIPS IP 仅在 IP integrator 块设计中可用。因此，必须使用 Vivado IP integrator 至少创建部分 Versal 自适应 SoC 设计。

Vivado IP integrator 是基于 Tcl 的图形化工具，支持您将基于 AMD IP 和用户封装的 IP 的各种子系统组合到整体设计中。这样您就可以在设计画布上对来自 Vivado IP 目录的 IP 核进行例化，并在这些 IP 核之间建立互连。Vivado IP integrator 是专为简化基于 Versal 自适应 SoC AXI 的 IP 连接而设计的。Vivado IP integrator 还可为 GT IP 和连接 IP（如 MRMAC IP）提供特殊支持，从而简化基于 GT 的设计创建和 I/O 管脚分配。

对于 Versal 器件，如果其设计按不同域（PS/PL/AI 引擎）进行分区，那么可借助 IP integrator 来简化此类设计的集成。例如，您可在 PL 域内创建硬件平台，其中包含各种块，用于执行计算以及连接到 PS 域、外部存储器和 I/O。此硬件平台还可连接到 AI 引擎块。

以下是使用 IP integrator 的益处：

- 允许在特定于 Versal 器件的块之间执行自动配置更新
- 允许在各块之间自动建立连接，这样可防止出错
- 允许自动定义系统地址映射
- 提供与 Vitis 工具的无缝交互，允许导出定制硬件平台

您可通过以下方式来使用 IP integrator 块设计 (BD)：

- 作为子模块包含在设计中
- 作为设计层级的顶层

建议：对于以 Versal 器件为目标的设计，虽然不作要求，但 AMD 建议使用 IP integrator BD 作为设计顶层，而不是作为基于 RTL 的顶层内部的子模块。使用此方法可确保系统中的所有存储器映射组件均为顶层块设计的后代，并且对 Vitis 和 PetaLinux 软件栈可见。

以下章节提供了有关您可从 Vivado IP integrator 访问的重要 IP 的信息，这些 IP 可供您用于创建和配置自己的 Versal 自适应 SoC 设计。如需了解有关使用方法信息以及常规硬件平台生成信息，请参阅《Vivado Design Suite 用户指南：采用 IP integrator 设计 IP 子系统》(UG994)。

CIPS IP 核

CIPS IP 支持您完成以下配置：

- 配置 PMC、PS、NoC 和（可选）PL 的器件时钟设置
- 配置 PMC 闪存控制器、外设及其关联的多路复用 I/O (MIO)
- 配置 PS 外设及其关联的 I/O
- 配置 PS-PL 中断和交叉触发
- 配置 CPM（含 DMA 和高速缓存一致性互连的 Integrated Block for PCIe®）
- 配置连接至 NoC 和 PL 的 PS 和 CPM AXI 接口
- 配置系统监控器供电和温度监控和警报
- 配置 HSDP 用于高速调试

如需了解更多信息，请参阅《Control, Interface and Processing System LogiCORE IP 产品指南》(PG352)。

AI 引擎 IP

要为 Vitis 环境生成可扩展平台，必须例化 AI 引擎 IP 并将其连接到设计其余部分。基础平台硬件设计包括块设计，其中包含最低配置的 AI 引擎 IP 块，此块包含已启用并连接到专用 NoC 总线主单元的所有存储器映射从接口 AXI 连接。

AI 引擎的所有其它配置都是通过编译用户 ADF 图和 AI 引擎内核来执行的，编译使用的是 Vitis aiecompiler 和 aiecompiler `libadf.a` 的 Vitis v++ 链接（含可扩展平台设计）。从 AI 引擎到基础平台的连接包括 AXI4-Stream 主接口和从接口连接、存储器映射 AXI 总线与 NoC 的连接以及总线接口的时钟。IP 内触发的 AI 引擎事件通过 AXI4 连接传输到存储器并穿过 XSDB。

如需了解更多信息，请参阅《AI Engine LogiCORE IP 产品指南》(PG358) 和《AI 引擎工具和流程用户指南》(UG1076)。

AXI NoC IP

NoC 是使用 AXI NoC IP 配置的。此 IP 充当 NoC 的逻辑表示法。AXI NoC IP 支持 AXI 存储器映射协议，并且 AXIS NoC IP 也支持 AXI4-Stream 协议。Versal 自适应 SoC 设计可包含每种 IP 类型的多个实例。

DDRMC 已集成到 AXI NoC IP 中。AXI NoC 实例可配置为包含 1、2 或 4 个 DDRMC 实例。您必须使用 NoC IP 来与集成 DDRMC 进行通信。在确认步骤中，Versal NoC 编译器按统一流量规格运行。确认后，“NoC Viewer”（NoC 查看器）窗口支持您查看并编辑 NoC 解决方案。

如需了解有关 NoC 和相关 IP 的配置详情以及有关系统地址映射的详细信息，请参阅《Versal 自适应 SoC Programmable Network on Chip and Integrated Memory Controller LogiCORE IP 产品指南》(PG313)。

Transceivers Bridge

Versal 自适应 SoC 收发器具备高度可配置性，并与 PL 块紧密集成。Versal 自适应 SoC Transceiver Bridge 支持针对基于 GT 的 IP 采用基于 Vivado IP integrator 的设计输入。这样您即可生成使用多个四通道的设计，或者也可以生成与多个协议 IP 共享四通道的设计。您必须使用 Vivado Design Suite I/O 管脚分配工具来添加物理 GT 位置。如需了解更多信息，请参阅《Versal 自适应 SoC Transceivers Wizard LogiCORE IP 产品指南》(PG331)。

设计地址映射

Versal 自适应 SoC 使用单一的统一系统地址映射。所有 AXI 存储器映射传输事务都必须遵循此映射。Versal 自适应 SoC 系统地址映射定义了 Versal 自适应 SoC 中从接口的默认地址位置。地址映射已构建到基于 PL 的 SmartConnect 和 NoC 内部。Vivado IP integrator 会基于 AXI NoC IP 自定义中所选的 DDR4 存储器选项自动解析地址区域的基本名称、偏移地址和范围。这些地址供 AXI 主接口用于与 DDR 进行通信。您可使用 Vivado IP integrator 地址编辑器 (Address Editor) 来为设计中所有存储器映射块选择或自动分配合规的地址。如需了解有关 NoC 和相关 IP 的配置详情以及有关系统地址映射的详细信息，请参阅《Versal 自适应 SoC Programmable Network on Chip and Integrated Memory Controller LogiCORE IP 产品指南》(PG313)。

RTL 设计流程

您可以使用 RTL 设计流程来创建模块、例化 IP 或者汇编顶层设计，与先前架构类似。但您必须遵循 AMD 建议，在 RTL 设计流程中使用特定于 Versal 器件的块，包括 CIPS 和 NoC IP。CIPS IP 支持访问器件配置功能，而 NoC IP 则可将 PL 连接至一个或多个 DDRMC 硬化的 IP。

AMD 强烈建议使用 Vivado IP integrator 来例化和配置 CIPS 和 NoC IP。但您无需将 IP integrator 用于自己的整个设计。CIPS IP、NoC IP 以及系统的其它存储器映射组件均可在块设计中使用 IP integrator 来进行配置。随后，生成的块设计即可在顶层 RTL 中进行例化。您可使用此方法通过传统 RTL 流程来构建设计中的大部分内容。

I/O 管脚分配

对于 Versal 自适应 SoC，适用于高性能 I/O (XPIO) 的 I/O 管脚分配流程不同于先前架构，欲知详情，请参阅以下章节。而适用于低性能 I/O 的 I/O 管脚分配流程（也称为高密度 I/O (HD I/O)）则与先前架构相同。

高性能 I/O

Versal 自适应 SoC 中的高性能 I/O 称为 XPIO。不同于先前器件中的列式 I/O 架构，高性能 I/O 是位于器件底层的外设。器件左侧处理器系统下存在的高性能 I/O 端口和器件右侧 GT 下存在的高性能 I/O 端口统称为角落 I/O。角落 I/O 用途有限，如用于集成 DDRMC 和有限时钟设置。如需了解有关 XPIO 的更多信息，请参阅《Versal 自适应 SoC SelectIO 资源架构手册》(AM010)。如需了解有关角落 I/O 的更多信息，请参阅《Versal 自适应 SoC 封装和管脚分配架构手册》(AM013)。

XPIO 所提供的 XPHY 逻辑与 AMD UltraScale™ 器件原生模式类似。XPHY 逻辑可将经过校准的延迟与串行逻辑和解串逻辑封装在一起，以提供 6 个单端 I/O 端口（称为半字节）。每个 XPIO bank 含 9 个 XPHY 逻辑站点 (site)，支持最多 54 个单端 I/O 端口。XPHY 逻辑用于集成 DDRMC、软核存储器控制器和任意高性能 I/O 接口。

重要提示！各组件模式单元（例如，IDELAY、ODELAY、ISERDES、OSERDES、IDDR 和 ODDR）均已被删除，以便为高性能接口提供支持。ISERDES 和 OSERDES 原语在 Versal 架构中不受支持，但支持通过 XPHY 逻辑来实现相似的功能。

在 XPIO 和 HD I/O bank 上均包含未经校准的 IDELAY、ODELAY、IDDR 和 ODDR（称为 I/O 逻辑 (IOL)），以支持运行速度不高于 250 Mb/s 的低性能旧接口。

由于使用 XPHY 逻辑，因此高性能接口的 I/O 管脚分配流程不同于先前架构。如果您先前使用 AMD Memory Interface Generator、“High-Speed SelectIO™”Wizard 或 SelectIO 组件模式生成了高性能接口，那么您必须使用“Versal IP”Wizard 来重新构建这些接口。

下表显示了高性能 UltraScale 器件 I/O 生成方式到 Versal 器件 I/O 生成方式的映射。

表 9：**器件 I/O 生成方式比较**

UltraScale 器件 I/O 生成方式	Versal 自适应 SoC I/O 生成
软核存储器控制器	通过 Versal NoC IP 使用集成 DDRMC 软核存储器控制器
“High Speed SelectIO”Wizard	“Versal Advanced I/O”Wizard
UltraScale 组件模式 · 高性能接口 · 经校准的 IDELAY、ODELAY、ISERDES、OSERDES、IDDR 和 ODDR	“Versal Advanced I/O”Wizard
UltraScale 组件模式 · 低性能接口（不超过 250 Mb/s） · 未经校准的 IDELAY、ODELAY、IDDR 和 ODDR	RTL 中例化的 I/O 逻辑

为 Versal 自适应 SoC 重新生成 IP 后，可使用 Advanced I/O Planner 来执行 I/O 管脚分配，此操作与 UltraScale 器件的软核存储器控制器 I/O 管脚分配流程类似。Advanced I/O Planner 会逐步指导您完成使用 XPHY 逻辑将接口映射到目标 XPIO bank 的流程，并确保您的高速接口以合规方式映射到 XPHY 逻辑。

AMD 建议按以下顺序对高速接口执行 I/O 管脚分配，以便最大程度利用可用的 XPHY 逻辑资源：

1. 通过 NoC 使用集成 DDRMC
2. 软核存储器控制器

3. Advanced I/O Wizard
4. I/O 逻辑

欲知详情，请参阅以下文档：

- 如需了解 DDR4 和 LPDDR4 管脚分配规则，请参阅《Versal 自适应 SoC Programmable Network on Chip and Integrated Memory Controller LogiCORE IP 产品指南》(PG313)。
- 如需了解软核存储器控制器规则，请参阅《Versal 自适应 SoC Soft DDR4 SDRAM Memory Controller LogiCORE IP 产品指南》(PG353) 和《Versal 自适应 SoC Soft RLDRAM 3 Memory Controller LogiCORE IP 产品指南》(PG354)。
- 如需了解有关“Advanced I/O”Wizard 的信息，请参阅《Advanced I/O Wizard LogiCORE IP 产品指南》(PG320)。

高密度 I/O

低性能 I/O 在 Versal 自适应 SoC 中被称为高密度 I/O (HD I/O)。HD I/O 通过未经校准的 IDELAY、ODELAY、IDDR 和 ODDR 原语（称为 I/O 逻辑）支持一小部分 UltraScale 器件组件模式原语。HD I/O 保留了先前器件中的列式 I/O 架构。

HD I/O 的 I/O 管脚分配流程与先前架构并无差异。您仍可继续在 HDL 代码中例化 I/O 逻辑原语。这些工具支持采用基于 XDC 的约束流程来分配 PACKAGE_PIN 约束。就像先前架构一样，您可从“I/O Ports”（I/O 端口）窗口直接拖放到“Package”（封装）窗口上。此外，您可在 Versal 自适应 SoC 中的 HD I/O bank 和 XPIO bank 之间移动 I/O 逻辑原语。

重要提示！XPIO 与 HD I/O 之间的电压范围并无重叠。XPIO 支持的电压范围低于 HD I/O。要了解特定 bank 的电压限制，请参阅对应您的器件的数据手册。

多路复用 I/O

Versal 自适应 SoC 多路复用 I/O (MIO) 与 Zynq UltraScale+ MPSoC 上的 MIO 类似。在 Versal 器件中，有 78 个 MIO 管脚、PMC MIO（bank 500 和 501）上有 52 个信号，LPD MIO (bank 502) 中则有 26 个信号。如需了解有关 MIO 管脚分配的详细信息，请访问此链接和此链接，以参阅《Versal 自适应 SoC 技术参考手册》(AM011) 中的相应内容。Versal 自适应 SoC Control, Interfaces, and Processing System (CIPS) IP 用于选择要使用的 MIO 并指定其功能。

扩展多路复用 I/O

某些 PMC 和 LPD 外设可借助扩展多路复用 I/O (EMIO) 接口通过 PL 布线到 XPIO，或者也可以借助 I/O 逻辑布线到 HD I/O。如需了解有关可接入 EMIO 的外设的详细信息，请访问此链接以参阅《Versal 自适应 SoC 技术参考手册》(AM011) 中的相应内容。CIPS IP 中的“IO Configuration”（I/O 配置）页面可用于选择接入 EMIO 的外设。由于 EMIO 使用的是 I/O 逻辑，因此 EMIO 的管脚分配是使用 Vivado Design Suite 中的传统拖放式管脚分配来完成的。

电源/功耗提示：在 XPIO 或 HD I/O 中对 EMIO 进行管脚分配时，选择 bank 类型时请考虑外设接口的 I/O 电压要求。XPIO bank 可支持的 I/O 电压上限为 1.5V，而 HD I/O bank 可支持的 I/O 电压则不低于 1.8V。对于 XPIO bank，建议是按速度从高到低的顺序对 I/O 逻辑进行布局，以便最大程度提升封装管脚的利用率，并且根据先前所分配的 I/O，这样可将 I/O 标准的选择范围缩小到有限范围内。您必须确保您的接口在一个或多个目标 bank 中可用。您可使用 IBIS 模型根据针对您的 bank 选定的 I/O 标准，按所需速度对接口进行仿真。

逻辑仿真

逻辑仿真可测试以 PL 互连结构为目标的硬件设计，它属于传统 FPGA 仿真流程。此仿真的范围可调整，从单个硬件块到整个硬件平台都适用。仿真的模型通常为 RTL，从而保证抽象层的周期精确性。仿真速度与测试设计大小成比例，设计越大，仿真耗时越长。要提升仿真性能，可将部分 Versal 自适应 SoC IP 块替换为 SystemC 传输事务级模型，此类模型仿真速度更快，但无法再保障周期精确性。此仿真的目的是先验证并调试详细的硬件功能，然后在器件上实现设计。

逻辑仿真可通过 Vivado Design Suite 获取。如需了解更多信息，请参阅《Vivado Design Suite 用户指南：逻辑仿真》(UG900)。

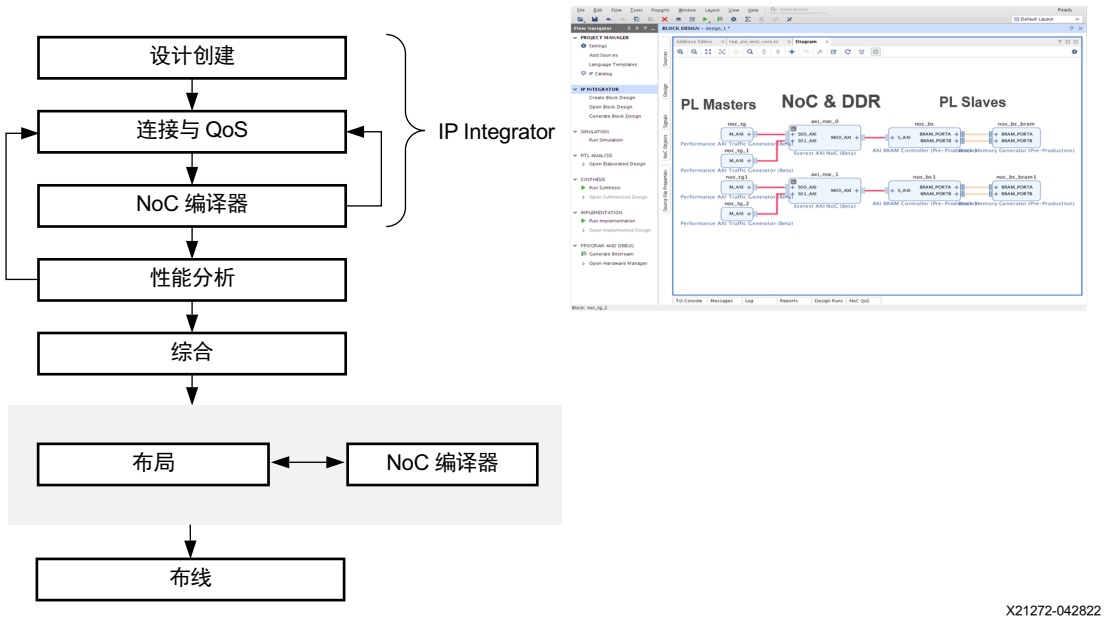
注释：在传统设计流程和基于平台的设计流程中均可执行逻辑仿真。

实现

您必须使用 Vivado 工具来执行综合与实现。Versal 自适应 SoC 包含新原语供 Vivado 综合工具用于推断。

器件互连结构相关的设计收敛方法与前几代器件系列相似。除了考量时序、拥塞和线长度外，布局器还会再次调用 NoC 编译器以考量发生修改的 NoC 端口位置约束或者用于在保证满足原有 QoS 要求的前提下，在 Dynamic Function eXchange (DFX) 模式下合并流量，如下图所示。

图 4：**NoC 编译器流程**



相关信息

原语

功耗收敛

为实现功耗收敛，请运行以下步骤。

应用约束和实现设计

确定开发板设计后，您必须确保设计保持在已明确的功耗、供电、时序和管脚分配约束范围内，以便确保设计能够满足您的要求。至少，AMD 建议约束设计总功耗，并提供最大环境温度和 Theta Ja，以便实现最高估算准确性。AMD 允许您通过组合连接到相同调节器的电源并为该调节器指定最大电流来约束器件功耗和供电解决方案。您可使用 `report_power` 来检查是否有任何电源使用率过高。如需了解更多信息，请参阅《Vivado Design Suite 用户指南：功耗分析与最优化》(UG907)。

以下提供了器件功耗约束示例：

```
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
```

提示：Vivado 工具也支持电源轨约束。欲知详情，请访问此链接以参阅《Vivado Design Suite 用户指南：功耗分析与最优化》(UG907) 中的相应内容。

功耗报告

在已实现的设计上运行 `report_power` 命令以确保应用于设计的功耗约束能够反映功耗估算和散热设计。`report_power` 命令还可基于约束显示任意裕度，从而确保设计功耗受到相应控制。通过多次运行以达成时序收敛时，AMD 建议运行 `report_power`，因为裕量最大的设计通常从功耗角度来看并非最佳设计。但只要存在裕量，设计即可被视为已达成时序收敛，而每次运行均生成功耗报告同样可便于您选择功耗的最佳实现。如需了解更多信息并获取脚本示例，请参阅答复记录 76056。

电源/功耗提示：如果您的设计超出功耗约束，那么快速运行假设场景分析的方法之一是将结果导出至 电源设计管理器 (PDM) 工具，并对其进行调整以查看对功耗的影响。

检查设计是否在预算范围内

此时，您必须对约束进行评估。如果功耗过高，则请尝试以下操作：

- 降低设计中的功耗。此方法最省时。欲知详情，请访问此链接以参阅《Vivado Design Suite 用户指南：功耗分析与最优化》(UG907) 中的相应内容。
- 改善散热设计约束中的热处理解决方案。例如，更改散热片结构，以改善热处理解决方案，如添加散热导管或强制风冷。
- 改善开发板和热处理解决方案（如有可能）。但请注意，改变散热可能导致设计功耗增加。
- 降低设计复杂性或减少设计中的功能。这是最终手段。例如，您可通过降低时钟频率、资源使用率或翻转率来降低功耗。您可使用 PDM 工具来执行假设场景分析，以便快速估算设计变更的影响。

设计收敛

Versal 架构引入了全新的硬件特性，同时，为了达成设计收敛（包括时序收敛和性能收敛）而需要考量的因素也随之增加。与先前的 AMD 器件架构类似，时序汇总报告用作为时序收敛的验收报告。Vivado Design Suite 编译工具可通过下列报告提供指南：

- 设计规则检查可防止出现无效的硬件配置 (`report_drc`)。任何此类问题都会导致器件镜像文件无法生成，因此必须得到解决。

- 方法论检查可以提升 PL 最大操作频率，并识别常见的不安全设计结构，此类问题可能导致硬件故障或者不稳定（`report_methodology` 和 `report_cdc`）。严重违例和警告违例必须加以解决，以帮助实现时序收敛和保证硬件稳定性。
- AMD 还建议解决 log 日志文件中的严重警告。

重要提示！为减少时序收敛迭代，您必须在实现流程中尽可能提前审查并解决时序违例，尤其是在综合后和布局后。

鉴于 Versal 架构的异构性质，设计性能主要依赖于 PL 和 AI 引擎中的 NoC QoS、DDR 存储器访问和软件效率以及 PL 操作频率和流水打拍数量。如需了解有关时序、系统性能和功耗设计收敛的信息，请参阅《Versal 自适应 SoC 系统集成和确认方法指南》(UG1388)。

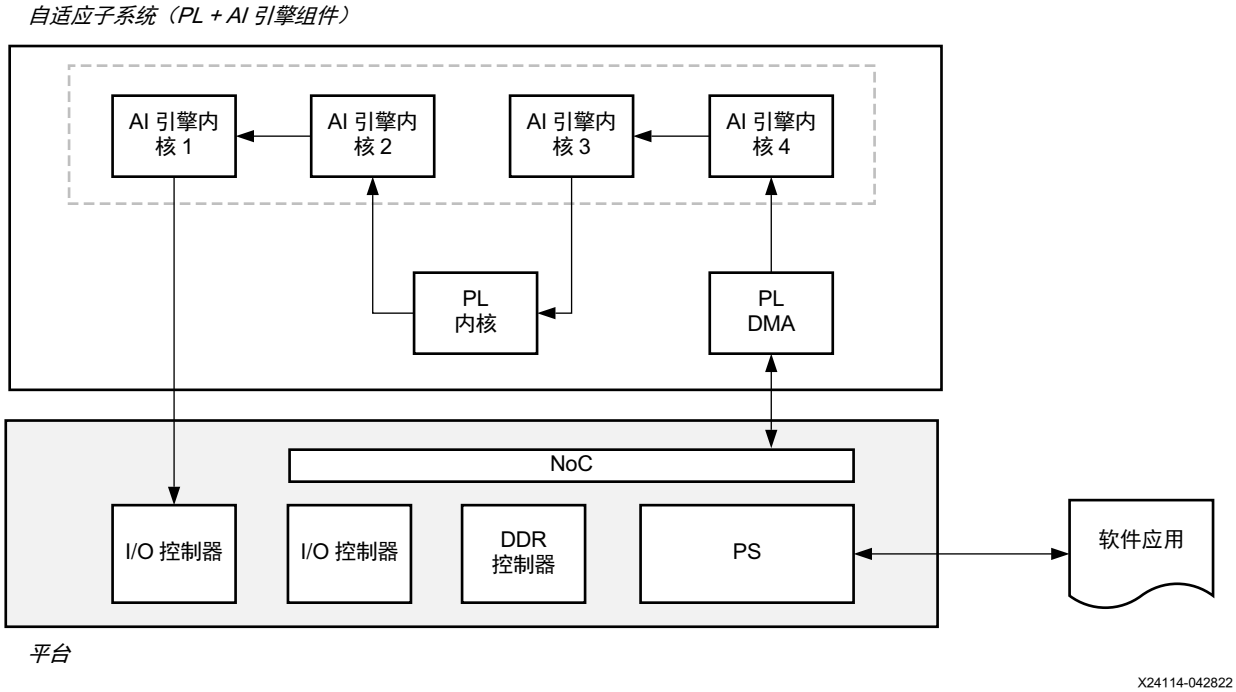
在设计流程中使用 Vitis 环境

Vitis 环境由工具、库和 IP 组成，支持您对 Versal 自适应 SoC 应用的不同要素进行编程、运行和调试，包括 AI 引擎内核与计算图、可编程逻辑 (PL) 函数以及处理器系统 (PS) 上运行的软件应用。Vitis 工具使用基于平台的方法，其中系统按概念分为以下要素，这些要素可并行开发和测试：

- 平台
- 自适应子系统
- 软件应用

下图显示了此基于平台的方法中所使用的要素。

图 5：**定制平台、自适应子系统和软件应用**



平台

平台可提供可构建和集成自适应子系统和软件应用的基础硬件 IP 块和软件功能。平台包含 2 个部分：硬件平台和软件平台。硬件平台包含 Versal 基础硬件 IP 块，其中包括 CIPS、NoC、I/O 控制器、AI 引擎阵列和其他用户指定的 IP 块。软件平台则定义了域、设备树和操作系统。

此平台使应用开发者能够免于应付低层次基础架构的细枝末节，转而将注意力集中于开发自适应系统的特定功能，例如，该软件、AI 引擎计算图或 PL 内核逻辑。平台的硬件侧是使用 Vivado 工具创建的。平台的软件侧则是使用 PetaLinux 或 Yocto 创建的。

自适应子系统

自适应子系统利用 PL 块和 AI 引擎计算图（适用于包含 AI 引擎的器件）来执行精确定义的功能。自适应子系统中的 PL 块通常被称为 PL 内核。PL 内核可以是 RTL、Vivado IP 或高层次综合 (HLS) 块。AI 引擎程序是使用以 C++ 编写的数据流计算图规范来开发的。自适应系统的组件在汇编并使用 Vitis 连接器与平台集成之前会单独进行设计和验证。

软件应用

软件应用在 PS 上运行，并在与自适应子系统进行交互期间执行高级应用任务。软件应用是使用 Vitis 嵌入式软件开发流程来开发的。

Vitis 工具

以下 Vitis 工具有助于创建、验证和集成完整系统的不同要素：

- AI 引擎工具：AI 引擎程序的编程、仿真和调试。此工具套件包含 aiecompiler、AI 引擎仿真器 (aiesimulator) 和 x86 仿真器 (x86simulator)。
- Vitis HLS 和 Vitis 编译器 (`v++ --compile`)：通过 C/C++ 源代码创建 PL 内核。
- Vivado IP 封装器：将现有的 IP 或 RTL 代码封装到 Vitis PL 内核中。
- Vitis 连接器 (v++ --link)：将 AI 引擎计算图和 PL 内核与平台集成。
- Vitis 封装器 (`v++ --package`)：集成系统的 PS 组件（软件应用）并生成启动镜像。
- Vitis 仿真流程：在与 Vitis 连接器集成之后但在实际硬件上运行之前，对 PS、PL 和 AI 引擎组件的行为进行仿真。
- Vitis 分析器：对于使用 Vitis 工具所创建的系统，可提供这些系统在进行编译、链接和执行期间所生成的报告。
- Vitis 嵌入式软件开发流程（具有系统软件栈，包含 PetaLinux）：支持嵌入式处理器的 PS 域。

注释：Model Composer 同样可供熟悉 MATLAB® 软件的用户使用。欲知详情，请参阅《Vitis Model Composer 用户指南》(UG1483) 以及 Vitis Model Composer 示例和教程。

如需了解有关这些工具的更多信息，请参阅《Vitis 统一软件平台文档》(UG1416)。

Vitis 环境设计方法论

要求

开始开发前，您必须选择最适合您的应用的 Versal 器件，然后根据应用要求按功能目标（PS、AI 引擎和 PL）对设计进行分区。在此情况下，您必须了解：

- 系统设计注意事项，例如吞吐量和时延

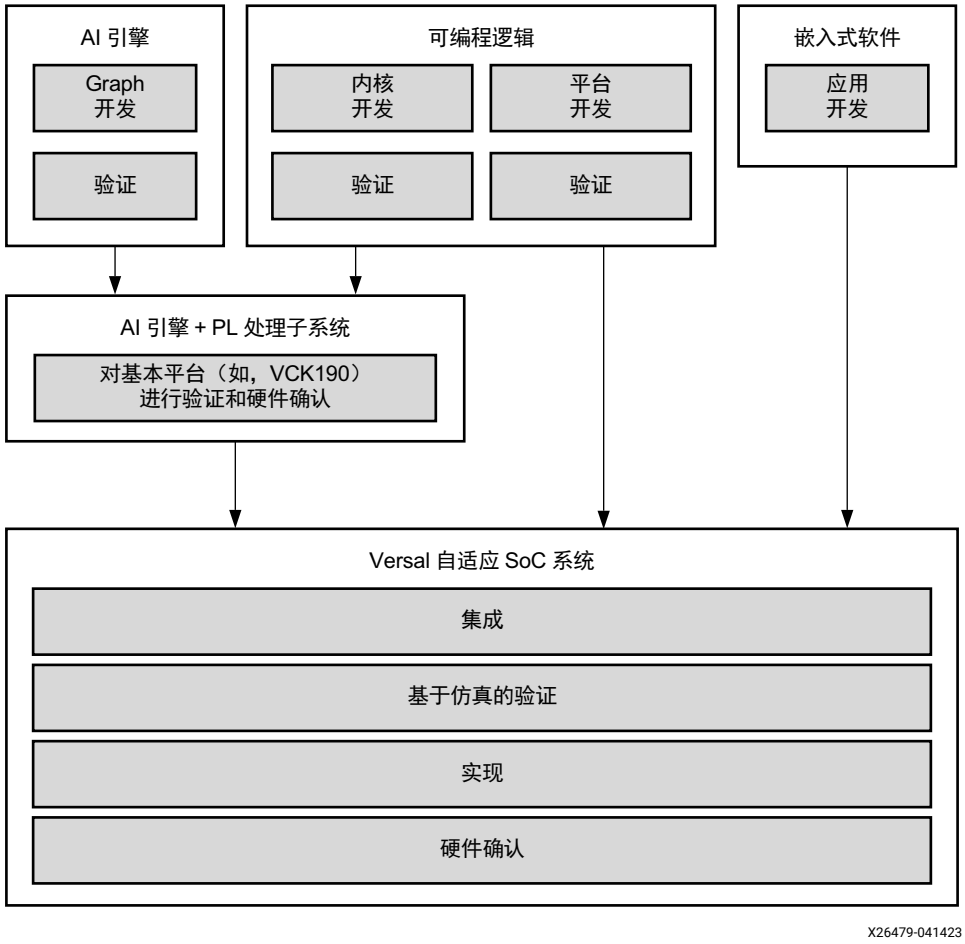
- 域功能以及域间功能，包括计算和带宽
- 整个系统和各子系统中的数据流和控制流程

此外，您必须注意平台的目标类型。您必须规划并设计开发板上的外设和接口以及定制开发板上可用的存储器资源。

方法论概述

下图显示了基于 Vitis 环境设计流程的开发方法论的高层次表示法。

图 6：**Versal 器件的基于平台的设计流程**



Vitis 环境开发方法论反应了 Versal 自适应 SoC 系统的异构性质，此类系统通常是由 PS、PL 和 AI 引擎功能组成的。您可使用 Vitis 工具来独立开发并验证这些组件，并逐渐将其加以集成以构成最终系统。

Vitis 环境设计流程是迭代性进程，可能多次循环执行每个步骤，并通过后续迭代来向自适应系统添加更多层级或元件。各团队可以快速迭代早期步骤，将更多时间用于后续步骤，以便提供更详细的性能数据。

最佳实践

Vitis 环境设计方法论的基础是迭代方法和并行开发。因此，AMD 强烈建议采用如下最佳实践：

- 并行开发自适应子系统和定制平台。

对系统进行精确分区，即上述两个要素可各自单独开发和验证，从而节省时间和精力。

- 单独调试并验证 AI 引擎计算图和每个 PL 内核，然后再进行集成。

  采用此方法可以尽可能提升在集成阶段快速融合的可能性。已知所有组件都正确无误的前提下，集成问题的调试难度大大降低。

- 使用标准 AMD 平台（例如，vck190）来集成并验证由 AI 引擎计算图和 PL 内核组成的自适应子系统，然后再将目标瞄准定制平台。

  AMD 平台都经过预验证，可立即部署到硬件上。通过使用标准 AMD 平台，AI 引擎计算图和 PL 内核的开发者即可使用仿真或硬件开发板来验证自适应子系统，同时可以避免定制平台的不确定性和复杂性。

- 确保在流程每个阶段都能满足性能目标。

  在硬件中运行完整系统与在隔离环境中对个别组件进行仿真相比，性能结果并无明显改善。因此，有必要在流程中尽早对任何性能问题进行完整检查和调试。在组件级别确保满足性能目标难度远低于在包含所有组件间交互的复杂系统环境内满足性能目标。

开发和验证 AI 引擎计算图

AI 引擎程序包含以 C++ 语言编写的数据流计算图规范，此规范由节点和边缘组成。节点表示计算函数（称为内核），边缘则表示数据连接。AI 引擎计算图规范使用 Vitis aiecompiler 来编译，并使用 Vitis aiesimulator 来执行仿真。

AMD 建议对计算图进行逐步微调和测试，缓慢从标量操作向矢量化操作进化。这样您无需执行低级 AI 引擎专用编码即可设置系统（例如，构建脚本、功能正确性等）。

计算图使用用户编写的测试激励文件来进行测试，此文件使用计算图应用编程接口 (API) 来驱动和管理此计算图。以下提供了可用于在计算图中导入和导出数据的方法：

- 运行时参数 (RTP) 属于可编程寄存器，用于在 PS 应用与 AI 引擎内核之间交换值。
- GMIO 可通过水平 NoC 提供从 AI 引擎到全局存储器的直接连接。
- PLIO 可提供 AI 引擎内核与 PL 块之间的直接串流连接。

测试激励文件可包含基本文件 I/O 操作或者以 Python、C/C++ 或 HDL 编写的更强大的流量生成器，用于在 AI 引擎计算图中输入和输出数据。

重要提示！在此阶段重要的是监控性能结果并满足性能目标，因为进入开发阶段后，系统性能将无法再提升。但在此设计早期阶段满足 aiesimulator 的性能并不代表最终系统性能，因为 aiesimulator 是在假定最佳操作条件下对 I/O 流量进行仿真的。

如需了解有关如何开发和仿真 AI 引擎计算图与内核的更多信息，请参阅 Versal 自适应 SoC 设计进程文档：AI 引擎开发。

使用 Vitis HLS 开发 PL 内核

PL 内核可使用 C/C++ 语言代码和 Vitis HLS 工具来进行开发。Vitis HLS 工具可简化 C/C++ 语言函数的使用，以便在 Vitis 应用加速开发流程中作为 PL 内核来实现这些函数。

在可编程逻辑中实现和最优化 C/C++ 语言代码以及实现低时延和高吞吐量所需的大部分代码修改操作均可通过 Vitis HLS 工具来自动执行。Vitis HLS 工具支持通过推断所需的编译指示来为函数实参生成正确的接口，并对代码内的循环和函数进行流水打拍。

Vitis HLS 设计流程包含以下主要步骤：

1. 编译、仿真和调试 C/C++ 语言算法。

2. 查看报告以分析和最优化设计。
3. 将 C 语言算法综合到 RTL 设计中。
4. 使用 Vitis HLS 协同仿真流程来验证 RTL 实现。
5. 将 RTL 实现编译到已编译的对象文件（扩展名为 `.xo`）中，或者导出到 RTL IP。

如需了解更多信息，请参阅 Versal 自适应 SoC 设计进程文档：硬件、IP 和平台开发指南 - 平台的“使用 HLS 创建 PL 内核”部分。

使用 RTL 和 Vivado Design Suite 开发 PL 内核

PL 内核同样可使用 RTL 内核与 Vivado Design Suite 来开发。此方法对于拥有现成 RTL IP（包括基于 Vivado IP integrator 的设计）或者倾向于通过编写 RTL 代码来创建新功能的硬件设计师而言十分便利。

RTL 内核属于封装为 Vivado Design Suite IP 的常规设计，但此内核必须遵循特定接口规则和要求才能在 Vitis 环境设计流程中使用。如需了解有关 RTL 内核的更多信息，请访问此链接以参阅《Vitis 统一软件平台文档：应用加速开发》(UG1393) 中的相应内容。

创建 RTL 内核的过程遵循传统 RTL 设计指南。AMD 强烈建议您先创建专用测试激励文件并使用行为仿真来完整验证 RTL 代码，然后在 Vitis 环境设计流程中封装此代码并将其用作为 PL 内核。完整验证 RTL 设计并满足 Vitis 内核的所有要求后，即可使用 Vivado IP 封装器将此设计封装到 Vitis 内核对象（XO 文件）中。

如需了解有关如何开发和仿真 RTL 内核的更多信息，请参阅 Versal 自适应 SoC 设计进程文档：硬件、IP 和平台开发指南 - 平台的“使用 RTL 创建 PL 内核”部分。

使用标准 AMD 平台集成自适应子系统

在独立完成 AI 引擎计算图和 PL 内核的开发和验证后，可将其集成以形成自适应子系统。随后，您可使用标准 AMD 平台（例如，vck190）来验证和测试该自适应子系统。AMD 平台已经过预验证，奠定了稳定的构建基础，使您能够专注于处理 AI 引擎计算图和 PL 内核。Vitis 环境设计流程允许您使用仿真或硬件开发板验证自适应子系统，如以下章节中所述。

注释：自适应子系统可独立于定制平台进行集成和验证。

仿真

基于仿真的自适应子系统验证方法被称为 Vitis 硬件仿真流程。硬件仿真流程可详细查看协同仿真系统（AI 引擎、PS 和 PL）的所有要素。您可针对系统 PL 区域中的任意信号在源代码中设置断点、追踪变量并绘制波形，从而使硬件仿真成为非常实用的调试平台，用于跟踪定位任何潜在集成问题的根源。

在硬件仿真期间，可使用以下工具来对每个要素进行仿真，支持对子系统进行渐进式汇编和验证：

- 适用于 AI 引擎计算图的 AI 引擎仿真器 (aiesimulator)
- Vivado 仿真器或受支持的第三方仿真器，用于对 PL 的 RTL 行为模型进行仿真
- AMD Quick Emulator (QEMU)，用于在 PS 上执行软件代码
- Python、C/C++ 或 HDL 流量生成器同样可用于对源自外部 I/O 的数据进行建模或者用于将缺失的功能替换为短截线

满足硬件仿真性能是必要条件，但结果无法保证。硬件仿真是周期近似行为，因此该阶段的性能结果并非最终结果。

如需了解有关如何对自适应子系统进行汇编和仿真的更多信息，请参阅 Versal 自适应 SoC 设计进程文档：系统集成和确认的“系统仿真”部分。

硬件测试

通过使用 Vitis 设计流程，同样的子系统不仅可以使用硬件仿真流程来进行仿真，也可以在硬件上实现和部署。以硬件开发板为目标时，Vitis 连接器会通过 Vivado 综合以及布局布线来发送该系统的 PL 区域。

通过在硬件中进行测试即可尽早得到有关时序收敛和资源使用情况的反馈。最重要的是，子系统运行速度不亚于在硬件上运行的速度，故而能够得到更真实的性能结果，如下所示：

- PS 上运行的控制代码的执行剖析精度更高
- I/O 模式更真实，导致停滞和反压的实践更真实
- 能够发现各种极端情况，此类情况在较慢的硬件仿真运行中无法达成

通过将编译时和运行时选项相结合使用，Vitis 环境设计流程即可允许您选择要剖析或追踪的具体信息。在硬件开发板上运行系统时，系统将自动收集此信息，您可使用 Vitis 分析器工具来查看和分析此数据。

建议：AMD 强烈建议使用标准 AMD 平台来调试并解决所有性能问题（例如，死锁、停滞或气泡），然后再将子系统与定制平台集成。

如需了解有关如何在硬件上汇编和验证子系统的更多信息，请参阅 Versal 自适应 SoC 设计进程文档：系统集成和确认的“实现”部分和“系统初始化和确认”部分。

开发和验证定制平台

定制平台属于可启动设计，Vitis 连接器可为其添加自适应子系统。定制平台必须包含下列基本组件：

- 从 Vivado Design Suite 导出的基础硬件设计
- 基础软件设计，其中包含 Linux 内核、根文件系统和设备树

  注释：或者，也支持裸机。

建议：AMD 建议采用并行方法来实现开发周期加速。例如，自适应子系统开发者在 AI 引擎图和 PL 内核上工作，其它开发者则同时在定制平台上工作。

如需了解有关如何使用 Vivado 工具创建平台的信息，请参阅 创建和生成平台。

使用定制平台集成和测试系统集成

使用标准平台完成子系统验证后，该子系统即可与定制平台集成。在此步骤中，将集成整个系统，包括定制平台的 PL 资源。这一步的焦点是使用定制平台达成时序收敛并满足性能目标。因此至关重要的是使用标准平台验证并调试前述步骤中的 AI 引擎与 PL 集成问题。

设计的 PL 部分将运行 Vivado 综合和布局布线，并应用标准 Vivado 实现和最优化技巧。随后，Vitis 环境会创建并配置 Vivado 工程。这样您即可获得对 Vivado 工程以及中间检查点的完整访问权限，从而使您能够运行任何必要的 Vivado Tcl 脚本来满足您的目标。

如需了解有关如何在硬件上汇编和验证自适应系统的更多信息，请参阅 Versal 自适应 SoC 设计进程文档：系统集成和确认的“实现”部分和“系统初始化和确认”部分。

仿真流程

为了应对仿真范围、仿真抽象和仿真目的等方面的不同需求，AMD 为 AMD Versal™ 器件设计的各组件提供了专用的流程，包括 AI 引擎、PS 和 PL。此外，AMD 还支持对由 PL、PS 和（可选）AI 引擎组件组成的完整系统进行协同仿真。各设计团队必须先在功能级别确认功能，然后再将其集成到部分系统应用或整个系统中。

下表显示了每个 Versal 器件块可用的仿真模型。

表 10：**Versal 器件块支持的仿真模型**

块	周期精确	性能
PS	QEMU（仅限功能运行）	QEMU（仅限功能运行） CIPS 验证 IP (VIP)
NoC	行为级 SystemVerilog（周期近似）	SystemC
DDRMC	行为级 SystemVerilog	SystemC
HBM 控制器	行为级 SystemVerilog	行为级 SystemVerilog
基于 PL 的软核存储器控制器	行为级 SystemVerilog	行为级 SystemVerilog
CPM	行为级 SecureIP	行为级 SecureIP
GT	行为级 SecureIP	文件 I/O（仅适用于 Vitis 软件平台）
基于 GT 的 IP	行为级 SecureIP	AXI Verification IP 文件 I/O（仅适用于 Vitis 软件平台）
基于 HLS 的 IP	RTL	RTL
其他 IP	因 IP 而异	因 IP 而异
PL	行为级 Verilog VHDL SystemVerilog	行为级 Verilog VHDL SystemVerilog
AI 引擎	SystemC（周期近似）	SystemC

以下章节提供了有关每个仿真流程的范围和目的的详细信息。

注释：这些仿真流程中大部分均可用于传统设计流程和基于平台的设计流程。但仅限在基于平台的设计流程中才能对完整系统执行协同仿真。

逻辑仿真

逻辑仿真可测试以 PL 互连结构为目标的硬件设计，它属于传统 FPGA 仿真流程。此仿真的范围可调整，从单个硬件块到整个硬件平台都适用。仿真的模型通常为 RTL，从而保证抽象层的周期精确性。仿真速度与测试设计大小成比例，设计越大，仿真耗时越长。要提升仿真性能，可将部分 Versal 自适应 SoC IP 块替换为 SystemC 传输事务级模型，此类模型仿真速度更快，但无法再保障周期精确性。此仿真的目的是先验证并调试详细的硬件功能，然后在器件上实现设计。

逻辑仿真可通过 Vivado Design Suite 获取。如需了解更多信息，请参阅《Vivado Design Suite 用户指南：逻辑仿真》(UG900)。

注释：在传统设计流程和基于平台的设计流程中均可执行逻辑仿真。

使用 SystemC 模型执行逻辑仿真

SystemC 属于 C++ 语言库，支持硬件建模。此库可提供结构元素（例如，模块、端口和接口）以及数据类型。除周期精确的仿真模型外，AMD 还为部分 Versal 自适应 SoC 基础架构提供了能够快速精确完成传输事务的 SystemC 仿真模型，以供其在 Vitis 硬件仿真流程中使用。SystemC 模型仿真速度比 RTL 模型更快，这样有助于缩短总体仿真时间。

总之，SystemC 模型可用于性能分析、架构探索、DMA 同步以及地址追踪生成和性能建模。但是，如果精确性和调试更为重要，那么 AMD 建议使用 RTL 模型，比如用于解决 DMA 传输事务或时序相关问题。

HLS 仿真

HLS 仿真专用于测试 HLS 代码，它是 HLS 开发流程中不可或缺的一环。此仿真的范围是单个 HLS 内核。受支持的抽象层分为以下 2 种：未定时和 RTL（周期精确）。这两个抽象层分别被称为 C 语言仿真 (Csim) 和协同仿真 (Cosim)。在 Csim 流程中，应使用 C 语言仿真通过测试激励文件确认要综合的函数。C 语言测试激励文件包含一个 `main()` 顶层函数，用于调用函数以供 Vitis HLS 工程进行综合。在 Cosim 流程中，HLS 编译器生成的 RTL 代码输出将与 Csim 结果的输出进行自动比对。Cosim 流程的目的是验证 RTL 的功能正确性，以及确认独立环境内的性能，它与其他函数无交互。

有关 HLS 仿真的信息，可通过 Vitis 统一软件平台来获取。欲知详情，请访问此链接以参阅 Vitis HLS 用户指南 (UG1399) 中的相应内容。

注释：在传统设计流程和基于平台的设计流程中均可执行 HLS 仿真。

AI 引擎仿真

您可以使用以下仿真器流程来仿真 AI 引擎 graph 和设计。这些流程可在 AI 引擎 graph 与内核开发阶段，在仿真速度与准确性之间做出权衡：

- 未定时 (x86simulator)：验证代码功能是否正确，并确认独立环境内的性能，它与其他函数无交互。此仿真流程有助于验证 AI 引擎 graph 与内核的功能准确性并提供最快的仿真运行速度。
- 周期近似 (aiesimulator)：更精确地计算核矢量负载和存储器访问，您可以使用它来估算 AI 引擎 graph 性能并提高功耗估算的准确性。此外，该仿真流程将 GMIO 和 PLIO 接口建模到 AI 引擎 graph 并对 NoC、PS 和 PL 进行建模，以提供周期近似结果。

有两种方法可用于向 AI 引擎仿真提供激励。要执行快速功能验证，可将 `input_plio` 语句添加到 graph 文件中，以提供矢量化输入列表。或者，可使用流量生成器来对动态环境进行更准确的建模。如需了解更多信息，请参阅 AXIS 外部流量生成器功能特性教程。

如需了解有关这些仿真流程的更多信息，请参阅以下资源：

- 请访问此链接以参阅《AI 引擎工具和流程用户指南》(UG1076) 中的相应内容
- Vitis 教程：AI 引擎开发调试演示示例
- Vitis 教程：AI 引擎 Versal 集成

嵌入式软件仿真

嵌入式软件仿真可测试仅以 PS 为目标的软件设计。它以 Quick Emulator (QEMU) 为基础，后者可对 Versal 自适应 SoC 内集成的双核 Arm® Cortex®-A72 的行为进行仿真。此仿真支持对平台操作系统进行快速紧凑的功能确认。此流程包含 SystemC 传输事务级系统模型，支持尽早进行系统浏览和验证。

有关嵌入式软件仿真的信息，可通过 Vitis 统一软件平台来获取。如需了解更多信息，请访问此链接以参阅《Versal 自适应 SoC 系统软件开发者指南》(UG1304) 中的相应内容，并请参阅赛灵思维基：QEMU 用户文档。

注释：在传统设计流程和基于平台的设计流程中均可执行嵌入式软件仿真。

硬件仿真

硬件仿真用于对整个 Versal 自适应 SoC 系统（包含 AI 引擎、PS 和 PL）进行仿真。通过使用 Vitis 软件平台即可集成以全部 3 个计算域为目标的块和功能。Vitis 连接器会自动生成完整的协同仿真设置，包括 RTL、SystemC 和 QEMU 模型：

- PS 上运行的嵌入式软件代码是使用 QEMU 来仿真的。
- AI 引擎上运行的代码是使用 SystemC AI 引擎仿真器来仿真的。
- 用户 PL 内核作为 RTL 代码来进行仿真。
- 根据可用或所选模型的类型，硬件平台中的 IP 块作为 RTL 或 SystemC TLM 来进行仿真。

因此，Vitis 硬件仿真的抽象层非常接近但未完全达成周期精确。Versal 自适应 SoC 平台的部分细节是使用 TLM 模型来抽象化的，目的是为了保证仿真速度。

Vitis 硬件仿真的范围同样定义了其目的。硬件仿真允许您对整个设计进行仿真，并在实现前测试 PL、PS 和 AI 引擎之间的交互。由于硬件仿真可通过调试查看应用的方方面面，因此在此环境中对复杂问题进行调试比在真实硬件中进行调试更简单。

有关硬件仿真的信息，可通过 Vitis 统一软件平台来获取。如需了解更多信息，请参阅《Vitis 统一软件平台文档：应用加速开发》(UG1393) 和《AI 引擎工具和流程用户指南》(UG1076)。

您可使用 AXI Traffic Generator 来代替使用基于文件的输入和输出。AXI Traffic Generator 与静态且受限的基于文件的输入和输出相反，通过动态方式来生产和使用数据。如需了解更多信息，请参阅 AXIS 外部流量生成器功能特性教程。

使用 AXI Traffic Generator 的优势如下：

- 允许您复用 AI 引擎仿真期间所设置的仿真集
- 允许您利用流量生成器抽取系统中的某些块，这样能够更快执行仿真

注释：仅限在基于平台的设计流程中才能执行硬件仿真。

NoC 仿真

NoC 仿真支持随 SystemVerilog 或 SystemC 中的行为模型一起提供。与 SystemVerilog 模型相比，SystemC 模型的仿真速度要快得多，但周期近似且精度较低。

注释：您可使用“IP Project Settings”（IP 工程设置）来选择自己首选的仿真模型。使用“rtl”设置即可选择 SystemVerilog，使用“tlm”设置即可选择 SystemC。这些设置适用于整个工程。

虽然可以同时使用 SystemC 和 SystemVerilog 模型来验证功能，但建议使用 SystemVerilog 模型进行性能分析。使用 SystemVerilog 模型的性能分析误差在硬件的 ±5% 范围内。

您可以使用 Vivado 工具中的仿真器或使用 Vitis 环境提供的硬件仿真流程来仿真 NoC。

重要提示！如需了解有关 NoC 仿真设置和性能调优的更多信息，请访问此链接和此链接，以参阅《Versal 自适应 SoC Programmable Network on Chip and Integrated Memory Controller LogiCORE IP 产品指南》(PG313) 中的相应内容。如需了解有关 AXI Traffic Generator 的更多信息，请参阅 GitHub 仓库中提供的 Versal 片上网络性能 AXI Traffic Generator 教程。

CIPS Verification IP

Control, Interfaces, and Processing System (CIPS) Verification Intellectual Property (VIP) 支持对 Versal 自适应 SoC 应用执行功能仿真。其目标是通过模拟处理器系统 (PS)-PL 接口和 PS 逻辑的 OCM 存储器来支持对可编程逻辑 (PL) 进行功能验证。此 VIP 是作为 SystemVerilog 模块封装包来交付的。VIP 操作通过使用一连串 SystemVerilog 任务来加以控制。如需了解更多信息，请参阅《Versal 自适应 SoC CIPS Verification IP 数据手册》(DS996)。

在设计流程中生成启动镜像

Versal 自适应 SoC PMC 使用专用的启动和配置文件格式来对 Versal 自适应 SoC 进行编程和配置，此格式称为可编程器件镜像 (programmable device image, PDI)。PDI 由以下几个部分组成：头文件、PLM 镜像和将加载到 Versal 自适应 SoC 中的设计数据镜像分区。PDI 还包含配置数据、ELF 文件、NoC 寄存器设置等。通过 PMC 块由 BootROM 和 PLM 对 PDI 镜像进行编程。如需了解有关 PDI 文件格式和生成的更多信息，请参阅《Bootgen 用户指南》(UG1283)。

PDI 的生成因设计流程而异：

- 面向仅限硬件系统的传统设计流程：运行 `write_device_image` 命令。欲知详情，请访问此链接以参阅《Vivado Design Suite 用户指南：编程和调试》(UG908) 中的相应内容。
- 面向嵌入式系统的传统设计流程：运行 Bootgen 工具。欲知详情，请访问此链接以参阅《Vitis 统一软件平台文档：嵌入式软件开发》(UG1400) 中的相应内容。
- 基于平台的设计流程：运行 `v++ --package` 命令。欲知详情，请访问此链接以参阅《AI 引擎工具和流程用户指南》(UG1076) 中的相应内容。

重要提示！ Vivado 工具要求在设计中包含 CIPS IP 才能创建 PDI。对于纯 RTL 设计，请访问 Vivado IP 目录中的 CIPS IP 并使用默认配置例化 CIPS IP 以启用 PDI 创建操作。

第 5 章

系统移植

将设计从 AMD UltraScale™ 或 AMD UltraScale+™ 架构移植到 AMD Versal™ 自适应 SoC 时，AMD 工具只能自动移植部分 PL 原语和集成的 IP 块。在某些情况下，AMD Versal™ 器件中没有等效的功能或连接。虽然可执行部分移植，但此方法通常会导致硬件和应用性能欠佳。因此，AMD 建议改为使用以下步骤：

- 对各主要块之间的所有高带宽连接进行重新架构，以使用 NoC 代替基于 PL 的 AXI Interconnect 或类似 IP。
- 利用全新集成块（例如，集成存储器控制器、DMA 和 AI 引擎）来减少 PL 逻辑。
- 将先前架构中例化的 PL 原语替换为等效的 RTL 描述或 XPM（例如，存储器块、DSP、进位逻辑、多路复用器等）。
- 重新生成或重新创建所有 IP 块。
- 对整个设计进行重新综合，而不是移植为先前架构创建的网表。

您必须仔细审查设计中自动移植的每一部分，以确保能够满足应用性能、资源和功能。对于从 AMD Zynq™ UltraScale+™ MPSoC 移植的设计，AMD 建议通过在新设计中例化 CIPS IP 来重新创建 PS 功能和连接，而不是尝试通过工具自动化来进行移植。

下表显示了可使用自动移植的块和功能。

表 11：**块和功能移植支持**

块	自动移植
可配置逻辑块 (CLB)	是
片上存储器 (OCM) 资源（块 RAM 和 UltraRAM）	大部分
DSP	是
时钟设置	部分
I/O	部分
软核存储器控制器	否
AXI 互连	否
GT	否
PCIe 子系统	否
以太网 MAC	否
处理器与外设	否
系统调试	否
系统监控器 (SYSMON)	否
功耗与错误处理	否
安全性	否
启动和配置	否
PL 配置和 JTAG	否

重要提示！如果现有设计包含 Versal 自适应 SoC 中已弃用的块，那么您必须手动将这些块移植到对应的 Versal 自适应 SoC 块。欲知详情，请参阅相应的 Versal 自适应 SoC 架构手册。

对于从 AMD Kintex™ UltraScale™、Kintex UltraScale+、AMD Virtex™ UltraScale™ 或 Virtex UltraScale+ 器件移植的设计，必须添加 CIPS IP 以启用基本功能（如器件配置和硬件调试功能），即使不使用 PS 功能也是如此。从 Zynq UltraScale+ MPSoC 移植的其他设计应已包含 PS 块。如需了解有关 CIPS IP 的更多信息，请参阅《Control, Interface and Processing System LogiCORE IP 产品指南》(PG352)。

性能和 PL 最大频率

Versal 器件提供了许多全新的专用 IP 块，例如，NoC、DDRMC、CPM 和 AI 引擎。这些专用 IP 块可以通过高带宽数据移动和接口来进一步提升系统级单位功耗性能。为了适应这些新集成的专用 IP 块，Versal 器件可编程逻辑 (PL) 已在 UltraScale+ 器件 PL 基础上进行了升级，不仅提升了硅片面积的效率，同时保留相似的 PL 性能。升级后，Versal 器件的处理方式较之前产生的主要变化如下：

- 现在，对于映射到先前架构中的 PL 的许多常用硬件函数，这些专用 IP 块能够提供有效支持，节省了大量 PL 资源。
- PL 布线互连和 CLB 的延迟分布以及时钟偏差和抖动特性与先前架构有所不同。此差异导致有部分逻辑路径更快，而部分逻辑路径则更慢。在本章后续小节中将详解有关 CLB 和时钟设置的主要差异。
- 随着下一代应用所需的 PL RAM 资源（包括硅片高效型 UltraRAM）量和特殊 IP 块列的数量增加，导致布线延迟出现更多变化。

将 PL 函数移植到 Versal 器件时，旧 RTL 设计可能需要微调以减少进位运算符周围的逻辑级数，并对流水线寄存器之间的逻辑级数进行重新平衡，以确保同等器件速度等级上的平均可编程逻辑互连结构性能与前几代保持不变。如需获取硬件设计建议，请参阅《Versal 自适应 SoC 硬件、IP 和平台开发方法指南》(UG1387)。如需获取时序收敛建议，请参阅《Versal 自适应 SoC 系统集成和确认方法指南》(UG1388)。

在传统 Fmax 基准测试中，会对目标技术之间 RTL 设计可实现的最大 PL 时钟速度进行比较，而现在，已不适合采用此方法来将 Versal 自适应 SoC 与前几代 FPGA 和 SoC 进行比对评估，原因如下：

- Versal 架构是专为自适应加速而经过最优化的架构。因此，仅围绕 PL 时钟速度来评估会导致忽略 Versal 器件专用 IP 块的优势。AMD 建议，改为专注于比较系统级计算和吞吐量指标。
- 全新的 Versal 自适应 SoC 高层次构建块并非通过 RTL 来推断，而是使用 AMD Vitis™ 环境或 AMD Vivado™ IP integrator 来进行设计。因此，比较 RTL 设计会导致过高估计 Versal 器件 PL 使用率，从而忽略因使用 Versal 器件专用 IP 块所节省的使用率和功耗。

CLB

Versal 自适应 SoC 中的 CLB 相比先前架构已得到了增强。Versal 自适应 SoC 中不再支持的 CLB 资源（例如，CARRY8、MUXF7、MUXF8、MUXF9 等）可通过推断相应的 Versal 自适应 SoC 块来自动移植。RTL 例化同样可自动移植。为了实现最优面积和时序约束结果，AMD 建议不要对 Versal 自适应 SoC 中不再支持的 CLB UNISIM 进行例化，并且建议您对自己的 RTL 进行重新综合，以推断相应的 Versal 自适应 SoC 块。如需了解详细的架构差异，请参阅《Versal 自适应 SoC 可配置逻辑块架构手册》(AM005)。

相关信息

CLB 原语

片上存储器资源

先前架构中的设计中使用的块 RAM 和 UltraRAM 可通过推断相应的 Versal 自适应 SoC 块来自动进行移植。RTL 例化同样可自动移植。如果某些 BRAM 和 UltraRAM 配置在 Versal 自适应 SoC 中不受支持，则会发出一条严重警告消息，并将实例转换为黑盒元件。在此情况下，必须更改设计，使其遵循 Versal 自适应 SoC 所支持的配置。AMD 建议您在设计移植后检验配置设置，以确保已自动选中正确的默认选项和设置。AMD 建议使用 AMD 可参数化宏 (XPM) 来推断 FIFO 和其他存储器。在 Versal 自适应 SoC 中不支持内置 FIFO。在 Vivado IP integrator 中，Embedded Memory Generator 和 Embedded FIFO Generator 取代了 Block Memory Generator 和 FIFO Generator IP。Block Memory Generator 和 FIFO Generator IP 无法自动完成移植。如需了解详细的架构差异，请参阅《Versal 自适应 SoC 存储器资源架构手册》(AM007)。

部分 Versal 自适应 SoC 包含加速器 RAM，此 RAM 具有额外的 4 MB 的片上存储器，其中包含位于 PS 外部的 ECC。该存储器提供两种访问方式：支持从 RPU 通过 128 位 AXI 接口来直接访问，也可从 PL 通过 256 位 AXI 接口来对其进行访问。该存储器分为 3 个 bank，支持从 PL 和 RPU 对不同 bank 进行读取或写入访问。如需了解有关 PS RAM 和加速器 RAM (XRAM) 的详细信息，请参阅《Versal 自适应 SoC 技术参考手册》(AM011)。

部分 Versal 器件的北侧边缘处包含 AI 引擎 tile 拼块阵列。AI 引擎阵列是 AI 引擎拼块构成的二维阵列，每个阵列均包含：一个 AI 引擎、一个高性能 VLIW 矢量 (SIMD) 处理器、集成数据存储器及用于串流、配置和调试的互连结构。

在每个 AI 引擎内都有一个专用单端口 16 KB 程序存储器，其位宽为 128 位，深度为 1k。该程序存储器支持指令压缩，并具有 ECC 保护和报告功能。

独立于 AI 引擎的每个 AI 引擎拼块都包含 32 KB 的数据存储器供 AI 引擎使用，并拆分为 8 个单端口 bank。如需了解有关 AIE 阵列专用 RAM 的更多详细信息，请参阅《Versal 自适应 SoC AI 引擎架构手册》(AM009)。

相关信息

RAM 原语

DSP

Versal 自适应 SoC 包含 DSP58 slice，它是 UltraScale+ 器件 DSP48E2 slice 的超集，并与之向后兼容。此外，Versal 自适应 SoC DSP 引擎支持在单一 DSP58 slice 内执行浮点运算，并且可将含专用互连的 2 个连续 DSP58 slice 组合在一起，以构建单一 18 位复数乘法器或复数乘积累加 (MACC)。在 Versal 自适应 SoC 中，可通过 Floating-Point Operator IP 或 Vitis HLS 工具支持 DSPFP32 模式。如要在 RTL 设计中使用此模式，请在移植后的设计中更新 Floating-Point Operator IP。

AMD 支持将已例化的 DSP 原语自动移植到 Versal 自适应 SoC 旧原语 (DSP48E5)。为提升性能和使用率，AMD 建议将 RTL 更新至 Versal 自适应 SoC RTL 模板，并对设计进行重新综合。

如需了解详细的架构差异，请参阅《Versal 自适应 SoC DSP 引擎架构手册》(AM004)。

重要提示！要充分发挥 Versal 自适应 SoC 的潜力以提升性能，请考虑数据路径中哪些部分可从 PL 移植到 AI 引擎。您可选择使用 Model Composer 和 System Generator 流程来为使用 MATLAB® 和 Simulink® 软件创建的设计比较 PL 与 AI 引擎实现。如需了解更多信息，请参阅《Vitis Model Composer 用户指南》(UG1483)。

相关信息

DSP 原语

时钟设置

为了在 Versal 架构中实现最优时钟设置结果，AMD 强烈建议：

- 使用“Clocking”Wizard 来配置 Versal 自适应 SoC 时钟管理原语。使用 Vivado 工具从先前架构移植时钟管理函数可能导致出现次优配置。如需了解更多信息，请参阅《适用于 Versal 自适应 SoC 的 Clocking Wizard LogiCORE IP 产品指南》(PG321)。
- 请复查 Versal 自适应 SoC 中的时钟管理的物理位置，对比先前架构中使用的时钟设置拓扑结构。

如需了解有关如何根据您的设计需求来设计时钟网络的更多信息，请参阅《Versal 自适应 SoC 硬件、IP 和平台开发方法指南》(UG1387)。如需了解有关 Versal 器件中的时钟管理原语的特征和位置的更多信息，请参阅《Versal 自适应 SoC 时钟资源架构手册》(AM003)。

虽然 Versal 器件的时钟设置特征与 UltraScale 器件相似，但您必须留意以下重要移植注意事项。

时钟管理函数

- 时钟管理函数由 Versal 器件中的 MMCME5、XPLL 和 DPLL 原语提供。相比于 UltraScale 器件中包含的相似原语，Versal 器件中的时钟管理原语包含额外的纠偏逻辑功能。
- 相比于 UltraScale 器件中的列式架构，Versal 器件中的时钟管理原语的位置不再位于常规结构中，并且仅当 Versal 器件中需要这些原语时，才会进行布局。在某些情况下，移植到 Versal 器件时，这可能导致布局灵活性受限，您必须在移植期间仔细审查时钟结构。
- UltraScale+ 器件原语按如下方式移植到 Versal 器件原语：
  - UltraScale+ 器件原语 MMCME4_ADV 移植到 MMCME5 Versal 器件原语。MMCME5 不支持 ZHOLD 补偿。从先前架构移植后生成的 MMCME5 设置可能为次优设置，AMD 建议使用“Clocking”Wizard 来直接配置 MMCME5，以在 Versal 架构内实现最优性能。
  - UltraScale+ 器件原语 PLLE4_ADV 移植到 XPLL Versal 器件原语。从先前架构移植后生成的 XPLL 设置可能为次优设置，AMD 建议使用“Clocking”Wizard 来直接配置 XPLL，以在 Versal 架构内实现最优性能。

全局时钟缓冲器

- 来自先前架构的全局时钟缓冲器（如 BUFGCE、BUFGCE_DIV、BUFGCTRL、BUFG_PS 和 BUFG_GT）会自动移植到 Versal 架构。
- Versal 新增器件中新增的多时钟缓冲器 (MBUFG) 原语支持叶级时钟分频，以降低时钟轨道使用率，并改进同步时钟域交汇上的时序约束。

时钟布线资源

- Versal 器件的时钟布线结构与 UltraScale 器件相似，在整个器件中使用全局时钟设置，但负载可采用局部或全局布局。
- Versal 器件不采用列式 I/O 架构，在不含 XPIO bank 的时钟区域内仅有 12 条水平布线轨道。含 XPIO bank 的时钟区域则含有 24 条水平布线轨道。

I/O

IDDR/ODDR 原语和 IBUF/OBUF 原语均可自动完成移植。您可使用 Advanced I/O Wizard 和 Advanced I/O Planner 为 Versal 器件构建高速 I/O 接口，这种接口类似于使用 SelectIO™ 原生模式为 UltraScale 器件创建的接口。

相关信息

I/O 管脚分配

软核存储器控制器

如果先前设计使用的是软核存储器控制器 IP，那么可使用 Versal 自适应 SoC 软核存储器控制器 IP，或者也可使用集成 DDRMC。AMD 建议使用集成 DDRMC 代替 Versal 自适应 SoC 软核存储器控制器 IP。在 Versal 自适应 SoC 中，您只能通过 NoC 来使用集成 DDRMC。NoC 和 DDRMC 具有极高的带宽，但通常时延比独立软核存储器控制器更高。部分 I/O bank 仅支持集成 DDRMC。如需了解有关 DDRMC 的更多信息，请参阅《Versal 架构和产品数据手册概述》(DS950)。

如果您使用的是软核存储器控制器 IP，则必须为 Versal 自适应 SoC 重新生成 IP。在 Versal 自适应 SoC 中，每个 I/O bank 均由 9 个半字节组成，每个半字节含 6 个管脚。根据器件和封装，部分 I/O bank 或任一 I/O bank 中的部分半字节专用于集成 DDRMC。软核存储器控制器无法使用这些专用管脚。专用于集成 DDRMC 的管脚在封装文件中名为“DDRMC ONLY”的列下指定为“YES”。软核存储器控制器只能使用指定为“NO”的管脚。如需了解有关软核存储器控制器 IP 的更多信息，请参阅以下指南：

- 《Versal 自适应 SoC Programmable Network on Chip and Integrated Memory Controller LogiCORE IP 产品指南》(PG313)
- 《Versal 自适应 SoC Soft DDR4 SDRAM Memory Controller LogiCORE IP 产品指南》(PG353)
- 《Versal 自适应 SoC Soft RLDRAM 3 Memory Controller LogiCORE IP 产品指南》(PG354)
- 《Versal 自适应 SoC Soft QDR-IV SRAM Memory Controller LogiCORE IP 产品指南》(PG355)

HBM

如果先前设计使用了 HBM IP，那么您可使用集成的 HBM 控制器。在 Versal 自适应 SoC 中，您只能通过 NoC 来使用集成的 HBM 控制器。将设计移植到 NoC 资源时，您必须使用 Vivado IP integrator 重新生成 IP，以在块设计中例化 NoC IP。如需了解更多信息，请参阅《Versal 自适应 SoC Programmable Network on Chip and Integrated Memory Controller LogiCORE IP 产品指南》(PG313) 和《Versal 架构和产品数据手册概述》(DS950)。

AXI Interconnect

软核 IP AXI Interconnect 已完全被集成 NoC 资源与 SmartConnect IP 的组合所取代。移植设计时，请首先考虑将 NoC 资源用于所有存储器存取路径以及用于降低 PL 资源利用率并支持高带宽连接。随后，您可使用 SmartConnect 来满足部分转换为 NoC 的需求，或者用于从利用率过高的 NoC 网络卸载流量。将设计移植到 NoC 资源时，必须使用 Vivado IP integrator 在块设计中例化 NoC IP。如需了解更多信息，请参阅《Versal 自适应 SoC Programmable Network on Chip and Integrated Memory Controller LogiCORE IP 产品指南》(PG313) 和《SmartConnect LogiCORE IP 产品指南》(PG247)。

GT

对于 Versal 自适应 SoC，GT 组件的粒度已从“Common/Channel”（公共/通道）更新为“GT Quad”（GT 四通道）。为支持部分 GT 共享用例，“GT”Wizard 流程已修改为使用 Vivado IP integrator。Vivado IP integrator 可用于构建系统设计，这些设计使用单个或多个 GT 四通道。定制 IP 的设计输入是通过 Bridge IP 来连接至 GT 四通道的，Bridge IP 可通过块自动化设置 (Block Automation) 来例化、配置和连接单个或多个基于 GT 四通道的 IP。如需了解更多信息，请参阅《Versal 自适应 SoC Transceivers Wizard LogiCORE IP 产品指南》(PG331)。移植设计时，您必须留意完整的 GT 四通道布局和受支持的配置选项。如需了解详细的架构差异，请参阅《Versal 自适应 SoC GTY 和 GTYP 收发器架构手册》(AM002) 和《Versal 自适应 SoC GTM 收发器架构手册》(AM017)。

PCIe 子系统

Versal 架构包括多个块，用于实现基于 PCI®-SIG 技术的高性能标准接口。除了集成 CPM 块外，Versal 架构还支持在 PL 内实现 PCIe®。PL PCIe 显著增强了先前架构中的 Integrated Block for PCIe 实现。

如果您的设计需从先前架构的 Integrated Block for PCIe 移植到 Versal 自适应 SoC PL PCIe® 块，请留意以下注意事项：

- 当前仅支持基于 Vivado IP integrator 的块设计流程（采用手动或自动连接）。
- Versal 自适应 SoC PL PCIe 所需的 GT 和 PHY IP 块不在 Versal 自适应 SoC PL PCIe® 块的范围内。
- 使用 PL PCIe® 块按所需链路速度、宽度和功能来配置 PCIe 子系统，并运行块自动化设置或者手动例化并连接所需的 GT 和 PHY 通道 IP。
- AMD 建议使用处理器系统内的 I/O 来驱动 PCIe 控制器的基本复位（必须在 CIPS IP 内配置）。
- 手动映射 RQ/RC/CQ/CC 串流接口和边带信号，这与其先前架构中各自的 IP 实现过程相似。

如果您的设计需从先前架构的 Integrated Block for PCIe 移植到 Versal 架构 CPM 块，请留意以下注意事项：

- 使用 CIPS IP 核按 CPM 中所需的链路速度、宽度和功能来配置 PCIe 子系统。

  注释：CPM 与 GT 间具有固定连接（基于 CPM 配置），且此连接无法更改。如需获取有关 CPM5 的 GT 选择和管脚分配指导信息，请访问此链接以参阅《Versal 自适应 SoC CPM DMA and Bridge Mode for PCI Express 产品指南》(PG347) 中的相关信息。
- PCIe 控制器的基本复位由处理器系统内的 I/O 来驱动（必须在 CIPS IP 内配置）。
- 可编程逻辑只能使用 user_clk，其频率根据已配置的链路速度和宽度可采用 62.5、125 或 250 MHz。
- 将 RQ/RC/CQ/CC 串流、边带信号、XDMA 串流和 QDMA 串流接口手动映射到 Versal 自适应 SoC CPM PL 接口。这些接口与其先前架构中各自的 IP 实现过程相似。
- 将 AXI4 存储器映射 (AXI4-MM) 接口（包括 AXI4-MM Bridge、AMD DMA Memory Mapped (XDMA-MM) 接口和 Queue DMA Memory Mapped (QDMA-MM) 接口）手动映射到 NoC 架构中。这需要在设计内设置各种组件，例如 NoC、处理器系统、地址转换和地址分配。

欲知详情，请参阅以下有关含 CPM 的器件的文档：

- 《Versal 自适应 SoC Integrated Block for PCI Express LogiCORE IP 产品指南》(PG343)
- 《Versal 自适应 SoC DMA and Bridge Subsystem for PCI Express 产品指南》(PG344)
- 《Versal 自适应 SoC PCIe PHY LogiCORE IP 产品指南》(PG345)

- 《Versal 自适应 SoC CPM Mode for PCI Express 产品指南》(PG346)
- 《Versal 自适应 SoC CPM DMA and Bridge Mode for PCI Express 产品指南》(PG347)

相关信息

CPM

以太网 MAC

如果要从 UltraScale 或 UltraScale+ 器件 Integrated 100G Ethernet (CMAC) 硬核块或者要从软核 10G/25G/40G 或 50G Ethernet IP 进行移植，请考量如下注意事项：

- MRMAC 可以为线速率、时钟设置和用户接口提供更宽的自定义范围：
  - 受支持的配置包括：1 x 100GE、2 x 50GE、1 x 40GE；4 x 25GE 和 4 x 10GE。
  - MRMAC 现具有集成 AXIS 接口用于执行 MAC+PCS 操作，这与 CMAC 相反，CMAC 可提供集成 512 位 LBUS 接口（具有可选 AXIS 接口）。
  - 可提供多种 AXIS 总线宽度和时钟设置选项以供使用，并且根据配置，这些选项与 UltraScale 或 UltraScale+ 器件 CMAC 或者软核解决方案中所提供的选项不同。
  - 全新的灵活端口 (Flex Port) 选项，可用于访问 PCS 级别。
- GT 不包含在 MRMAC 核内。IP integrator 块自动化设置可用于在 MRMAC 与 GT 之间建立连接。
- 原先提供的统计数据计数器增量矢量已被统计数据寄存器所替代，该寄存器已集成到硬核块中并且可通过 AXI4-Lite 来使用。
- MRMAC 还支持全新的高精度时间戳功能，可在 IEEE 1588 标准时间戳上实现亚纳秒级精度。

如需了解有关 MRMAC 的更多信息以及有关生成 MRMAC 设计示例的详细信息，请参阅《Versal 器件 Integrated 100G Multirate Ethernet MAC (MRMAC) LogiCORE IP 产品指南》(PG314)。

处理器与外设

在 Versal 自适应 SoC 中，PS 上的裸机应用和 Linux 应用的软件栈与 Zynq UltraScale+ MPSoC 中的软件栈相似。Versal 自适应 SoC 使用 PLM 来执行启动。以 APU 为目标的 Zynq UltraScale+ MPSoC 设计可通过移植来搭配 Versal 自适应 SoC APU 一起工作。Versal 自适应 SoC RPU 使用的 Arm® Cortex®-R5F 处理器（及其中所含 GIC）与 Zynq UltraScale+ MPSoC 相同。功能和编程模型极为相似。以 RPU 为目标的 UltraScale+ 器件设计可通过移植来搭配 Versal 自适应 SoC RPU 一起工作。移植到 Versal 自适应 SoC 时，必须考量器件驱动变更、多路复用 I/O (MIO) 配置和管脚分配。如需了解更多信息，请参阅《Versal 自适应 SoC 技术参考手册》(AM011) 和《Versal 自适应 SoC 系统软件开发者指南》(UG1304)。

系统调试

在 PL 互连结构中调试设计的方法与先前架构中类似，但存在如下几项关键差异：

- 所有互连结构调试 IP 核都具有 AXI4-Stream 从控制接口。先前架构使用的是专用接口标准。
- AXI Debug Hub IP 核具有 AXI4-Stream 控制接口（用于连接到互连结构调试 IP 核）和 AXI4 存储器映射从接口（用于来自主机的连接）。先前架构中所使用的 Debug Hub IP 依靠专用接口来连接到调试核和主机。
- Vivado 工具中的调试流程当前支持在 Debug Hub 与调试核之间采用自动连接和手动连接。
- 在 Versal 自适应 SoC 架构中不再提供 JTAG-to-AXI 软核调试 IP 作为选项。DAP 和 DPC 可用于访问设计中基于 AXI 的块。
- 基于 AXI4-Stream 的 Integrated Logic Analyzer (ILA) 核支持 ILA 功能和“System ILA”（系统 ILA）功能。在先前架构中，这些均作为独立 IP 核提供。
- 基于 AXI4-Stream 的 ILA 核支持选择块 RAM 或 UltraRAM 作为存储器用于走线存储。
- 在 Versal 自适应 SoC 架构中不再提供 PJTAG 作为选项。用户必须改用单一 JTAG 接口来访问 DAP 和 TAP。请咨询您的调试器供应商，以确认是否支持该解决方案。

移植时，请留意以下注意事项：

- Vivado IP integrator：您必须手动移除或替换先前已例化的旧调试核。在块设计中使用 IP integrator 将旧的调试核替换为新的 AXIS-ILA 核。
- 网表：用于将 ILA 核插入已综合的设计的赛灵思设计约束 (XDC) 命令会自动移植到新的 AXIS-ILA 调试 IP。
- RTL：由于存在全新的接口要求，来自先前架构的互连结构调试核不会自动移植到基于 AXI4-Stream 的新调试 IP 核。如果来自先前架构的调试核在设计中已例化，那么必须在设计中对新的调试 IP 进行手动重新自定义、重新生成和重新例化。
- IBERT 和软核存储器控制器校准：Integrated Bit Error Ratio Tester (IBERT) IP 功能是 GT 块的一部分，可与使用收发器的任意设计搭配使用。存储器控制器校准调试可用于 DDRMC 块和基于互连结构的软核存储器控制器 IP。
- Debug Hub：由于存在全新的接口要求，仅当在 CIPS 上启用 pl0_resetn 时，旧的 Debug Hub 才会自动插入网表。或者，也可手动添加 AXI4 Debug Hub。欲知详情，请访问此链接以参阅《Vivado Design Suite 用户指南：编程和调试》(UG908) 中的相应内容。

系统监控器

Versal 自适应 SoC 可提供与 UltraScale+ 器件相似的系统监控功能。在 UltraScale+ 器件设计中，您对 PL 侧的 SYSMON IP 进行了例化，并使用了“System Management”Wizard（系统管理向导）来设置寄存器配置，以便使用硬件描述语言 (HDL) 进行例化。要进行设计移植，您必须从设计中手动移除 SYSMON IP。在 Versal 自适应 SoC 中，可在“CIPS IP Configuration”Wizard 中启用系统监控器功能。动态重配置端口 (DRP) 访问则被存储器映射寄存器所替代。如需了解更多信息，请参阅《Versal 自适应 SoC 系统监控器架构手册》(AM006)。

功耗与错误处理

Zynq UltraScale+ MPSoC 包含某些功耗模式，这些模式可映射到 Versal 自适应 SoC 中的功耗模式。在 Versal 自适应 SoC 内共享属于电源岛的外设。这些电源岛在不使用时会被 PLM 自动关闭。对于为功耗域供电的外部电源轨，当前其中大部分都支持睡眠状态。在未来版本中将添加更多支持。在 Zynq UltraScale+ MPSoC 和 Versal 自适应 SoC 中，PL 功耗管理方法类似，并提供最大的功耗优势。欲知详情，请访问此链接以参阅《Versal 自适应 SoC 开发板系统设计方法指南》(UG1506) 中的相应内容。在 Versal 自适应 SoC 中，来自 DDR、PL、SYSMON 和其他系统块的错误将路由到 PMC 错误聚合模块 (EAM)，而 PS 错误将路由到 PS 管理控制器 (PSM) 中的 EAM。可能的错误操作包括上电复位 (POR)、系统复位 (SRST)、错误输出或中断软件代理。

如需了解详细的架构差异，请参阅《Versal 自适应 SoC 技术参考手册》(AM011) 和《Versal 自适应 SoC 系统软件开发者指南》(UG1304)。

安全性

Versal 自适应 SoC 的安全性架构相比前几代提升显著。信任根随 BootROM 一起启动，这样即可验证器件的安全状态。如果所有检查都通过，那么 BootROM 会对 PLM 固件进行身份验证，随后加载。如果您之前选择加密 PLM，那么 BootROM 还会在完成身份验证后对 PLM 进行解密。BootROM 只能从 PMC 中的 RCU 运行。当 PLM 固件加载完成并验证后，PLM 即可确保以安全方式加载剩余固件和软件。如需了解与安全性相关的详细信息（包括使用说明），请参阅 AMD 网站上设计安全性专区（需要注册）中提供的《Versal 自适应 SoC 安全手册》(UG1508)。下表高亮显示了 Versal 自适应 SoC 的可能的安全启动配置，并显示了与 Zynq UltraScale+ MPSoC 的比较。

表 12：**累积安全启动操作**

启动类型	操作			硬件加密引擎	
	身份验证	解密	完整性（校验和验证）	Zynq UltraScale+ MPSoC	Versal 自适应 SoC
非安全	否	否	否	是 不使用内建引擎	是 不使用内建引擎
硬件信任根 (HWRoT)	是	可选	采用身份验证来确认身份	是 RSA 和 SHA3	不适用
非对称硬件信任根 (A-HWRoT)	是	可选	采用身份验证来确认身份	不适用	是 RSA/ECDSA，SHA3 （AES-GCM 和 PUF 为可选）
对称硬件信任根 (S-HWRoT)	通过 GCM 和 eFUSE 予以支持	是 必须使用 PUF KEK	采用身份验证来确认身份	不适用	是 AES-GCM，PUF
A-HWRoT + S-HWRoT	是	是 必须使用 PUF KEK	采用身份验证来确认身份	不适用	是 RSA/ECDSA、SHA3、AES-GCM 和 PUF
校验和验证	否	否	是	是 SHA3	是 SHA3

启动和配置

如要从 AMD UltraScale+™ 器件系列进行移植，请留意以下注意事项：

- UltraScale+ 器件设计（不使用 PS）：这些设计包含集成配置逻辑，在上电时支持采用多个配置模式。对于 Versal 自适应 SoC，启动和配置流程都有所变化，需使用 CIPS IP。
- Zynq UltraScale+ MPSoC 和 Zynq UltraScale+ RFSoC 设计（使用 PS）：这些器件具有用于管理和执行启动流程的 PMU 和 CSU。对于 Versal 自适应 SoC，启动流程方法有所变化，此方法依靠 PMC 中的 RCU 和 PPU 来管理和执行启动流程。此外，需使用 CIPS IP 来配置启动外设。

如需获取有关 Versal 自适应 SoC 启动模式、启动顺序和启动镜像的更多信息，请参阅下列资源：

- 请访问此链接以参阅《Versal 自适应 SoC 技术参考手册》(AM011) 中的相应内容
- 请访问此链接以参阅《Versal 自适应 SoC 系统软件开发者指南》(UG1304) 中的相应内容
- 《Bootgen 用户指南》(UG1283)

下表对 UltraScale+ 器件与 Versal 自适应 SoC 的主要启动和配置模式进行了比较。

表 13：**启动模式比较**

模式	Virtex UltraScale+ 或 Kintex UltraScale+ FPGA	Zynq UltraScale+ MPSoC 或 Zynq UltraScale+ RFSoC	Versal 自适应 SoC
JTAG	是	是	是
OSPI	-	-	是
QSPI32	是	是	是
QSPI24	是	是	是
SelectMAP	是	-	是 [1]
eMMC1 (4.51)	-	是	是
SD1 (3.0)	-	是	是
SD1 (2.0)	-	是	是
SD0 (3.0)	-	-	是
SD0 (2.0)	-	是	-
PJTAG_0	-	-	-
PJTAG_1	-	是	-
串口	是	-	-
BPI	是	-	注释 2
NAND	-	是	注释 2
USB (2.0)	-	是	-

注释：

1. SelectMAP 模式可使用 BUSY 信号提供硬件流量控制。
2. Octal SPI 和 eMMC1 模式取代了上一代架构中的 BPI 和 NAND 模式。Octal SPI 和 eMMC1 模式可提供相似的性能，同时减少管脚数。

串联配置

串联配置属于分阶段配置，在 Versal 自适应 SoC 供电稳定后的 100 ms 内完成 PCIe 协议的初始化。这是使用阶段 1 可编程器件镜像 (PDI) 来达成的。器件其余部分（包括 PL）可由用户应用作为阶段 2 PDI 来完成下载。在 Versal 自适应 SoC 中，串联配置是使用 CPM 集成块来完成的。串联配置支持 Tandem PCIe（串联 PCIe）模式和 Tandem PROM（串联 PROM）模式。如需了解更多信息，请参阅《Versal 自适应 SoC CPM DMA and Bridge Mode for PCI Express 产品指南》(PG347)。

PL 配置和 JTAG

Versal 架构的启动和配置不同于先前架构。PL 配置和 JTAG 独立原语在 Versal 自适应 SoC 中不受支持，但存在如下类似的功能：

- BSCANE2 原语被 CIPS IP 中的 4 条 JTAG TAP 用户指令所替代。
- STARTUPE3 原语被 QSPI 控制器 MIO 与 CIPS IP（全局异步置位/复位信号、全局三态、启动结束 (EOS) 信号、PL 时钟 (PL0-PL3) 源配置）的组合所替代。
- DNA_PORTE2 原语则被 JTAG DNA 寄存器或 AXI 存储器映射可访问 32 位寄存器 DNA_0、DNA_1、DNA_2 和 DNA_3 所替代，用于读取器件 DNA。
- EFUSE_USR 原语被 AXI 存储器映射式 EFUSE_CACHE 寄存器所替代。
- ICAPE3 已由配置帧接口 (CFI) 总线替代，后者可通过 CIPS IP 进行访问。
- 在 PLM_RTCA（运行时配置区域）模块中，USR_ACCESS 原语已替换为 PMC USR_ACCESS 寄存器。如需了解有关 PDI 属性设置的更多信息，请参阅《Vivado Design Suite 用户指南：编程和调试》(UG908)。

如需了解详细比较结果，请访问此链接以参阅《Versal 自适应 SoC 技术参考手册》(AM011) 中的相应内容。如需了解有关存储器映射寄存器的更多信息，请参阅《Versal 自适应 SoC 技术参考手册》(AM011) 和《Versal 自适应 SoC 寄存器参考资料》(AM012)。

附录 A

原语

AMD Versal™ 自适应 SoC 包含新原语供 AMD Vivado™ 综合工具用于推断。如需了解有关 Versal 自适应 SoC 原语的详细信息，请参阅下列指南：

- 《Versal 架构 Prime 系列库指南》(UG1344)
- 《Versal 架构 Premium 系列库指南》(UG1485)
- 《Versal 架构 AI Core 系列库指南》(UG1353)

注释：本附录只涵盖不同于 AMD UltraScale+™ 器件系列的 Versal 自适应 SoC 原语。

RAM 原语

Versal 架构支持块 RAM 和 UltraRAM 原语。

块 RAM 原语

以下是 Versal 自适应 SoC 中的块 RAM 原语。

原语	支持的宽高比	支持的模式
RAMB36E5	1Kx36 2Kx18 4Kx9	在简单双端口 (SDP) 模式下运行时为 x72 模式
RAMB18E5	1Kx18 2Kx9	在 SDP 模式下运行时为 x36 模式

在 SDP 模式下，一个地址读取 RAM，另一个地址写入 RAM。可以使用不同的时钟进行读取和写入，但是地址行必须分开。示例如下图所示。

图 7：**SDP 模式下 512x72 块 RAM 的 Verilog 编码样式**

```
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
```

图 8：**SDP 模式下 512x72 块 RAM 的超高速集成电路硬件描述语言 (VHDL) 编码样式**

```
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;
```

复位

以下为块 RAM 上的复位类型：

- 块 RAM 输出上的同步复位，使用 RESETRAMA 或 RESETRAMB 管脚
- 块 RAM 输出上的异步复位，使用 ARST_A 或 ARST_B 管脚

  注释：如果使用 ARST_A 或 ARST_B 管脚，则忽略 RESETRAMA、RESETRAMB、RSTREGA 和 RSTREGB 管脚。
- 用于控制块 RAM 的可选输出寄存器的同步复位，使用 RSTREGA 或 RSTREGB 管脚

使用异步复位时：

- RAM 和可选输出寄存器都必须使用相同的异步复位。

  注释：如果可选输出寄存器不使用相同的复位，则无法推断到块 RAM 中。
- 输出使能和 SRVAL 属性将被忽略。
- 异步复位只能复位为 0 值。

写入模式

Versal 自适应 SoC 块 RAM 支持与 AMD UltraScale™ 器件相同的写入模式，并使用相同的 RTL 编码样式：

- WRITE_FIRST 将新写入的数据输出到输出总线上。
- READ_FIRST 将先前存储的数据输出到输出总线上。
- NO_CHANGE 保留输出总线先前的值。

字节写入使能

字节写入使能的控制端口是 WEA 和 WEB 管脚，根据使用情况而有所不同：

- RAMB36E5
  - 在非 SDP 模式下，WEA 和 WEB [3:0] 管脚控制大小为 8 或 9 位的 4 字节。
  - 在 SDP 模式下，WEA 和 WEB [7:0] 管脚控制大小为 8 或 9 位的 8 字节。
- RAMB18E5
  - 在非 SDP 模式下，WEA 和 WEB [1:0] 管脚控制大小为 8 或 9 位的 2 字节。
  - 在 SDP 模式下，WEA 和 WEB [3:0] 管脚控制 8 或 9 位的 4 字节。

注释：大小为 9 位指包含 8 位字节和 1 个奇偶校验位。

当前，如果所用大小为 8 或 9 位，则 Vivado 综合仅推断字节写入 RAM。此外，如果使能使用独热状态编码，则 Vivado 综合仅推断字节写入使能 RAM。例如，在配置为真双端口（数据宽度为 36）的字节写入使能 RAM 中，有 4 个不同的字节，但每次只能写入 1 字节。要推断块 RAM，请确保 RTL 遵守这些限制。

非对称 RAM

对于 Versal 自适应 SoC 中的非对称块 RAM，请使用与 UltraScale 器件中非对称块 RAM 相同的编码样式和规则。如需了解有关设置非对称块 RAM 的更多信息，请参阅《Vivado Design Suite 用户指南：综合》(UG901)。

注释：当前，Vivado 综合不支持在 Versal 自适应 SoC 中的非对称块 RAM 上进行异步复位。

使用 XPM

XPM 同样可用于推断块 RAM。使用这种方法的优点是，对于所需的任何类型的 RAM，XPM 都具有正确的编码样式。如需了解有关 XPM 的更多信息，请参阅《Vivado Design Suite 用户指南：系统级设计输入》(UG895)。

UltraRAM 原语

以下是 Versal 自适应 SoC 中使用的 UltraRAM 原语。要强制执行 Vivado 综合以推断 UltraRAM，请在 RAM 上设置 `RAM_STYLE = "ultra"` 属性。

注释：与 UltraScale 器件一样，Versal 自适应 SoC 的 UltraRAM 仅含 1 个时钟。

原语	支持的宽高比	支持的模式
URAM288E5	4Kx72 8Kx36 16Kx18 32Kx9	双端口 单端口

额外寄存器数

除了可选的输出寄存器，UltraRAM 还支持在数据线上使用输入寄存器。与块 RAM 一样，您可以使用同步或异步复位信号来复位可选的寄存器。

RAM 初始化

在 Versal 自适应 SoC 中，UltraRAM 可初始化为非零 (0) 值。在 RAM 上使用 INIT_xx 属性初始化 UltraRAM，如下所示：

- Verilog：使用 `readmemh` 命令。
- VHDL：设置函数以读取 VHDL 中的外部文件。

如需了解更多详情，请参阅《Vivado Design Suite 用户指南：综合》(UG901)。

字节写入使能

UltraRAM 还支持字节写入使能操作。与块 RAM 一样，字节可为 8 位或 9 位（使用额外的奇偶校验位）。但是，在 Versal 自适应 SoC 中使用字节写入时，在写入过程中会忽略读取操作。因此，使用字节写入来描述 UltraRAM 时，仅支持 NO_CHANGE 模式。

非对称 UltraRAM

Versal 自适应 SoC UltraRAM 支持非对称纵横比。如需查看非对称 RAM 编码方式示例，请访问此链接以参阅《Vivado Design Suite 用户指南：综合》(UG901) 中的相应内容。

XPM 推断

XPM 同样可用于推断 UltraRAM。使用这种方法的优点是，对于所需的任何类型的 RAM，XPM 都具有正确的编码样式。如需了解有关 XPM 的更多信息，请参阅《Vivado Design Suite 用户指南：系统级设计输入》(UG895)。

DSP 原语

以下是 Versal 自适应 SoC 中不同类型的 DSP 原语。

原语	描述	用法
DSP58	标准整数/定点模式	推断或例化
DSPFP32	浮点模式	仅例化
DSPCPLX	复数乘法器	推断或例化

DSP58

对于 Versal 自适应 SoC，DSP58 原语包含与 UltraScale 器件相同的功能，包括乘法器、加法器、预加法器和寄存器，用于对原语进行完全流水打拍。但是，两者大小不同，并且原语包含其他功能。

大小

对于有符号逻辑，可以按以下方式配置 DSP58：

- 乘法器：27x24
- 加法器：58 位
- 预加法器：27 位

对于无符号逻辑，可以按以下方式配置 DSP58：

- 乘法器：26x23
- 加法器：57 位
- 预加法器：26 位

下图为有符号逻辑示例。

图 9：**具有 58 位加法器和 27 位预加法器的 27x24 乘法器详图**

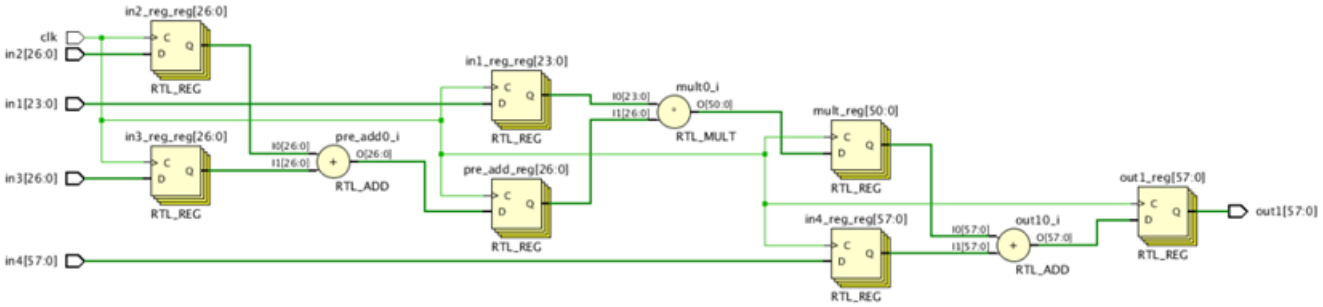


图 10：**具有 58 位加法器和 27 位预加法器的 27x24 乘法器的 Verilog RTL**

```
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
```

图 11：**具有 58 位加法器和 27 位预加法器的 27x24 乘法器的 VHDL RTL**

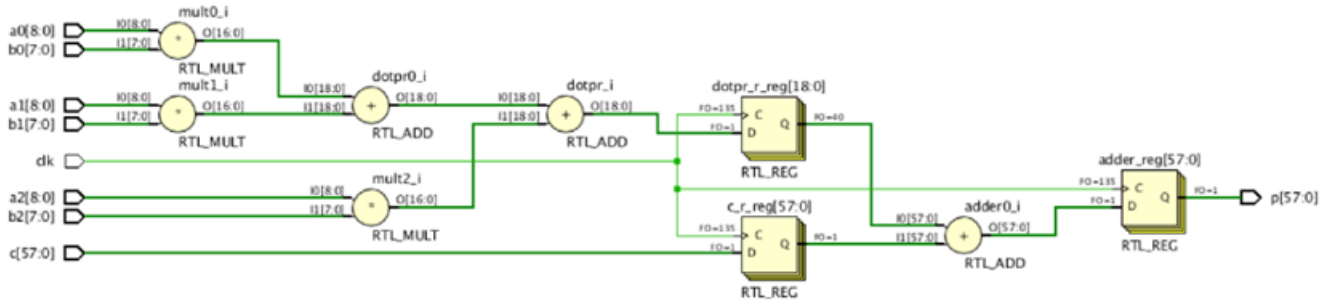
```
architecture beh of test is

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;
```

点积

DSP58 可以实现点积，该乘法器以 3 个较小的乘法器相加来表示。点积通常用于图像处理滤波器。如需了解更多信息，请参阅《Versal 自适应 SoC DSP 引擎架构手册》(AM004)。下图为带有附加加法器的点积示例。

注释：为了支持点积进行推断，RTL 必须使用有符号逻辑。

图 12：**带附加加法器的点积详图**



下图显示了点积的 RTL。

图 13：**点积的 Verilog RTL**

```
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
```

图 14：**点积的 VHDL RTL**

```
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 可以执行浮点计算。Vivado 综合不处理这些计算。而是改为提供各种 IP，或者也可以例化 DSPFP32 原语。

DSPCPLX

DSPCPLX 专为逻辑综合而设计，此逻辑是对以下公式的实部和虚部进行求解所必需的：

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

每个 DSPCPLX 占用 2 个 DSP58 site 位置。DSPCPLX 可以在 RTL 中进行例化，或者也可以通过推断得出。下图显示了 DSPCPLX 的 RTL。

图 15：**用于 DSPCPLX 综合的 Verilog RTL**

```
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
```

图 16：**用于 DSPCPLX 综合的 VHDL RTL**

```
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;
```

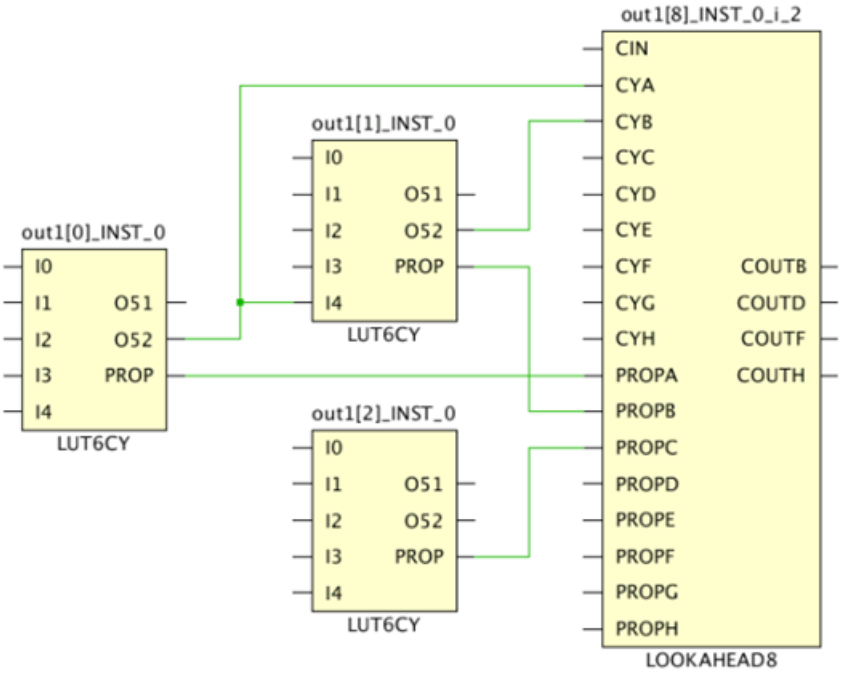
CLB 原语

Versal 自适应 SoC 中的可配置逻辑块 (CLB) 与 UltraScale 器件中的 CLB 不同。Vivado 综合可处理架构差异，但是您必须了解下文中提到的差异。

进位链

Versal 自适应 SoC 包含 LOOKAHEAD8 原语用于代替 UltraScale 器件中所含的 CARRY8 原语。LOOKAHEAD8 原语不包括用于算术运算的 MUXCY 和 XORCY。相反，必须对这些运算符进行推断，因此 LUT 数量稍微多一些。

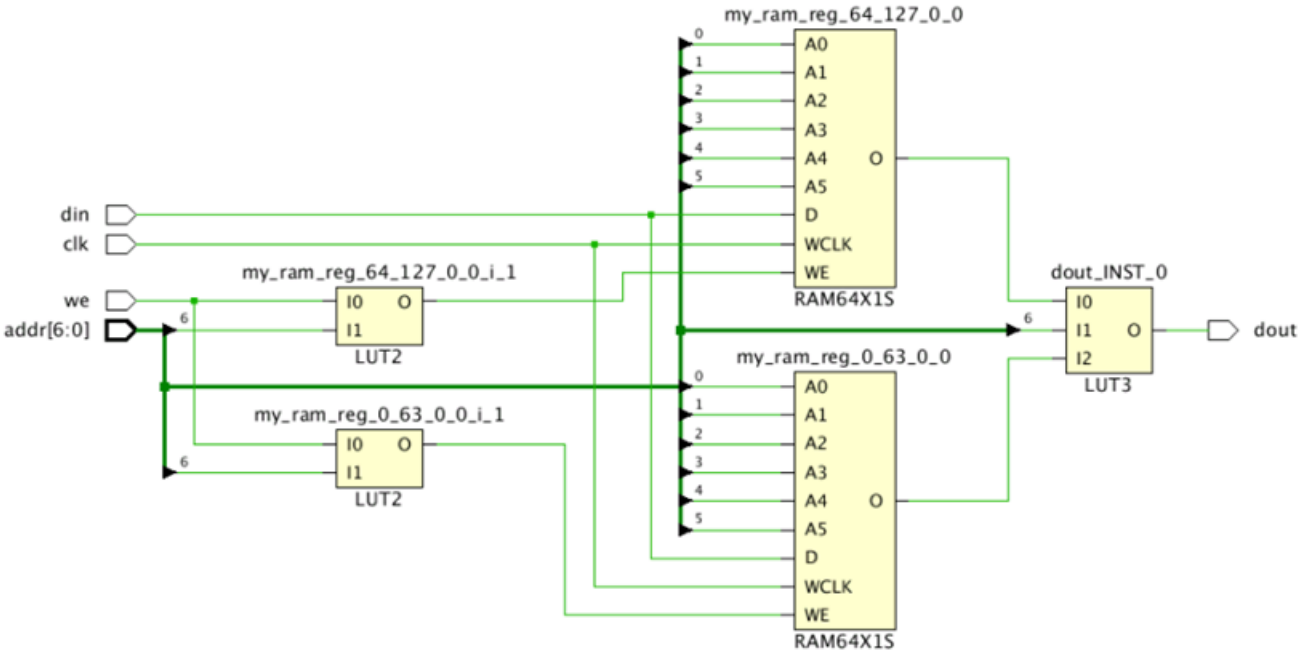
图 17：**进位链前的额外 LUT**



MUXFx 原语

Versal 自适应 SoC 不包括 MUXFx 原语。因为 MUXFx 原语常用于分布式 RAM、大型比较器或 MUX 链中的地址解码，所以在 Versal 自适应 SoC 中使用这些类型的结构时会有额外数量的 LUT，如下图所示。

图 18：**用于地址解码的额外 LUT**



编码样式和原语例化示例

如需获取编码样式和原语例化示例，请参阅以下资源：

- Vivado IDE 中的语言模板
- 请访问此链接以参阅《Vivado Design Suite 用户指南：综合》(UG901) 中的相应内容

附录 B

附加资源与法律声明

查找其他文档

文档门户

AMD 自适应计算文档门户是旨在使用您的网页浏览器提供健全的文档搜索和导航的在线工具。要访问文档门户，请转至 https://docs.xilinx.com。

注释：单击链接将打开英语版本，但您可从下拉列表中选择简体中文版本（如可用）。请注意，简体中文版本可能比英语版本旧。

Documentation Navigator

Documentation Navigator (DocNav) 是预安装的工具，支持访问 AMD 自适应计算文档、视频和支持资源，您可在其中通过筛选和搜索来查找信息。要打开 DocNav，请执行以下操作：

- 在 AMD Vivado™ IDE 中，单击“Help” → “Documentation and Tutorials”。
- 在 Windows 上，单击“Start”（开始）按钮并选中“Xilinx Design Tools” → “DocNav”。
- 在 Linux 命令提示中输入 `docnav`。

注释：如需了解有关 DocNav 的更多信息，请参阅《Documentation Navigator 用户指南》(UG968)。

注释：您无法从 DocNaV 访问简体中文版本。请使用设计中心网页。

设计中心 (Design Hub)

AMD 设计中心提供了根据设计任务和其他主题整理的文档链接，可供您用于了解关键概念以及常见问题解答。要访问设计中心，请执行以下操作：

- 在 DocNav 中，单击“Design Hubs View”选项卡。
- 转至设计中心网页。

支持资源

如需获取答复记录、技术文档、下载以及论坛等支持资源，请访问技术支持。

参考资料

以下技术文档是非常实用的补充资料，可配合本指南一起使用：

1. 《Versal 架构和产品数据手册概述》(DS950)
2. 《Versal 自适应 SoC CIPS Verification IP 数据手册》(DS996)
3. 《利用温度漂移扩展热处理解决方案》(WP517)
4. 《Versal 自适应 SoC GTY 和 GTYP 收发器架构手册》(AM002)
5. 《Versal 自适应 SoC 时钟资源架构手册》(AM003)
6. 《Versal 自适应 SoC DSP 引擎架构手册》(AM004)
7. 《Versal 自适应 SoC 可配置逻辑块架构手册》(AM005)
8. 《Versal 自适应 SoC 系统监控器架构手册》(AM006)
9. 《Versal 自适应 SoC 存储器资源架构手册》(AM007)
10. 《Versal 自适应 SoC AI 引擎架构手册》(AM009)
11. 《Versal 自适应 SoC SelectIO 资源架构手册》(AM010)
12. 《Versal 自适应 SoC 技术参考手册》(AM011)
13. 《Versal 自适应 SoC 寄存器参考资料》(AM012)
14. 《Versal 自适应 SoC 封装和管脚分配架构手册》(AM013)
15. 《Versal 自适应 SoC CPM CCIX 架构手册》(AM016)
16. 《Versal 自适应 SoC GTM 收发器架构手册》(AM017)
17. 《SmartConnect LogiCORE IP 产品指南》(PG247)
18. 《Versal 自适应 SoC Programmable Network on Chip and Integrated Memory Controller LogiCORE IP 产品指南》(PG313)
19. 《Versal 器件 Integrated 100G Multirate Ethernet MAC (MRMAC) LogiCORE IP 产品指南》(PG314)
20. 《Advanced I/O Wizard LogiCORE IP 产品指南》(PG320)
21. 《适用于 Versal 自适应 SoC 的 Clocking Wizard LogiCORE IP 产品指南》(PG321)
22. 《Versal 自适应 SoC Transceivers Wizard LogiCORE IP 产品指南》(PG331)
23. 《Versal 自适应 SoC Integrated Block for PCI Express LogiCORE IP 产品指南》(PG343)
24. 《Versal 自适应 SoC DMA and Bridge Subsystem for PCI Express 产品指南》(PG344)
25. 《Versal 自适应 SoC PCIe PHY LogiCORE IP 产品指南》(PG345)
26. 《Versal 自适应 SoC CPM Mode for PCI Express 产品指南》(PG346)
27. 《Versal 自适应 SoC CPM DMA and Bridge Mode for PCI Express 产品指南》(PG347)
28. 《Control, Interface and Processing System LogiCORE IP 产品指南》(PG352)
29. 《Versal 自适应 SoC Soft DDR4 SDRAM Memory Controller LogiCORE IP 产品指南》(PG353)
30. 《Versal 自适应 SoC Soft RLDRAM 3 Memory Controller LogiCORE IP 产品指南》(PG354)
31. 《Versal 自适应 SoC Soft QDR-IV SRAM Memory Controller LogiCORE IP 产品指南》(PG355)

32. 《AI Engine LogiCORE IP 产品指南》(PG358)
33. 《Versal 自适应 SoC 600G Channelized Multirate Ethernet Subsystem (DCMAC) 产品指南》(PG369)
34. 《Versal 自适应 SoC 600G Interlaken LogiCORE IP 产品指南》(PG371)
35. 《Versal 自适应 SoC 400G High Speed Channelized Cryptography Engine Subsystem LogiCORE IP 产品指南》(PG372)
36. 《Versal 自适应 SoC PCB 设计用户指南》(UG863)
37. 《Vivado Design Suite 用户指南：系统级设计输入》(UG895)
38. 《Vivado Design Suite 用户指南：逻辑仿真》(UG900)
39. 《Vivado Design Suite 用户指南：综合》(UG901)
40. 《Vivado Design Suite 用户指南：功耗分析与最优化》(UG907)
41. 《Vivado Design Suite 用户指南：编程和调试》(UG908)
42. 《Vivado Design Suite 用户指南：Dynamic Function eXchange》(UG909)
43. 《Vivado Design Suite 用户指南：采用 IP integrator 设计 IP 子系统》(UG994)
44. 《Vivado Design Suite：AXI 参考指南》(UG1037)
45. 《AI 引擎工具和流程用户指南》(UG1076)
46. 《AI 引擎内核与 Graph 编程指南》(UG1079)
47. 《Bootgen 用户指南》(UG1283)
48. 《Versal 自适应 SoC 系统软件开发者指南》(UG1304)
49. 《Versal 架构 Prime 系列库指南》(UG1344)
50. 《Versal 架构 AI Core 系列库指南》(UG1353)
51. 《Versal 自适应 SoC 硬件、IP 和平台开发方法指南》(UG1387)
52. 《Versal 自适应 SoC 系统集成和确认方法指南》(UG1388)
53. 《Vitis 统一软件平台文档：应用加速开发》(UG1393)
54. Vitis HLS 用户指南 (UG1399)
55. 《Vitis 统一软件平台文档：嵌入式软件开发》(UG1400)
56. 《Vitis 统一软件平台文档》(UG1416)
57. 《XRT 版本说明》(UG1451)
58. 《Vitis Model Composer 用户指南》(UG1483)
59. 《Versal 架构 Premium 系列库指南》(UG1485)
60. 《Versal 自适应 SoC 系统和解决方案规划方法指南》(UG1504)
61. 《Versal 自适应 SoC 开发板系统设计方法指南》(UG1506)
62. 《Versal 自适应 SoC 安全手册》(UG1508)
63. 《SmartLynq+ 模块用户指南》(UG1514)
64. 《电源设计管理器用户指南》(UG1556)
65. 《Versal 自适应 SoC 板级原理图审查检查表》(XTP546)
66. 《Versal 自适应 SoC 外部存储器预规划工具》(XTP667)

修订历史

下表列出了本文档的修订历史。

章节	修订综述
2023 年 5 月 10 日 2023.1 版	
文档标题	标题更改为《Versal 自适应 SoC 设计指南》(UG1273)。
第 2 章：系统架构	添加“按系列划分的 Versal 器件”表格。
AI 引擎	添加 AIE-ML。
HBM	新增章节。
系统设计类型	添加 HBM 描述。
第 4 章：设计流程	添加 HBM 描述。
面向仅限硬件系统的传统设计流程	添加 HBM 描述。
面向嵌入式系统的传统设计流程	添加 HBM 描述。
基于平台的设计流程	添加 AI Edge。
仿真流程	在“Versal 器件块支持的仿真模型”表格中添加 HBM。
HBM	新增章节。

请阅读：重要法律声明

版权声明