UltraFast 嵌入式设计方法指南

UG1046 (v2.1) 2015 年 4 月 22 日

条款中英文版本如有歧义，概以英文本为准。

XILINX
ALL PROGRAMMABLE™

修订历史

下表列出了本文档的修订历史。

日期	版本	修订
2015 年 4 月 22 日	2.1	• 新增 7 页的嵌入式设计方法检查表。 • 新增 8 页的访问技术文档和培训资料。
2015 年 3 月 26 日	2.0	• 新增第 8 章：SDSoC 环境。 • 新增 158 页的相关设计中心。
2014 年 10 月 20 日	1.1	• 移除过时信息 • 在第 2 章：系统级考虑事项中，添加以下部分的信息 。 性能 。 时钟和复位
2014 年 10 月 8 日	1.0	初始版本

目录

第 1 章：引言

嵌入式设计方法检查表 . . . 7
访问技术文档和培训资料 . . . 8

第 2 章：系统级考虑事项

性能 . . . 10
功耗 . . . 13
时钟和复位 . . . 25
中断 . . . 29
嵌入式器件安全性 . . . 32
剖析 (Profiling) 和划分 (Partitioning) . . . 35

第 3 章：硬件设计考虑事项

配置和启动器件 . . . 44
存储器接口 . . . 48
外设 . . . 52
设计 IP 模块： . . . 65
硬件性能考虑事项 . . . 70
数据流程 . . . 74
PL 时钟方法 . . . 76
ACP 和高速缓存一致性 . . . 80
PL 高性能端口访问 . . . 82
系统管理硬件辅助 . . . 84
管理硬件重新配置 . . . 87
GP 和来自 APU 的直接 PL 访问 . . . 91

第 4 章：软件设计考虑事项

处理器配置 . . . 93
OS 和 RTOS 选择 . . . 97
库和中间件 . . . 104
启动加载器 . . . 106
软件开发工具 . . . 110

第 5 章：硬件设计流程

简介 . . . 117
使用 Vivado IDE 构建 IP 子系统 . . . 117
基于规则的连接 . . . 119
创建层次化 IP 子系统 . . . 119
板器件接口 . . . 119
生成模块设计 . . . 119
创建和封装供重用的 IP . . . 120

创建定制接口 . . . 121
管理定制 IP . . . 121
高层次综合 (HLS) . . . 121
总结 . . . 122

第 6 章：软件设计流程
板启动开发 . . . 125
驱动开发 . . . 127
应用开发者 . . . 133
赛灵思 SDK 工具和封装 . . . 138
赛灵思软件开发工具 . . . 141

第 7 章：调试
简介 . . . 142
纯软件调试 . . . 143
基于仿真的调试 . . . 146
板调试 . . . 147
硬件与软件协同调试 . . . 148
虚拟平台 . . . 148

第 8 章：SDSoC 环境
引言 . . . 151
整体使用流程 . . . 152
剖析 . . . 154
性能估算 . . . 154
生成并运行完整的软件 - 硬件系统 . . . 154
使用 C 语言调用 RTL IP 库优化性能 . . . 154
使用 HLS 优化 IP 性能 . . . 154
优化系统性能 . . . 155
调试系统 . . . 156
性能测量与分析 . . . 156
专家使用模型 . . . 157

附录 A：附加资源与法律提示
赛灵思资源 . . . 158
解决方案中心 . . . 158
Xilinx Documentation Navigator . . . 158
参考资料 . . . 158
培训资料 . . . 162
请阅读：重要法律提示 . . . 162

XILINX
ALL PROGRAMMABLE™

第 1 章

引言

通过一款 All Programmable SoC 产品，赛灵思为设计人员提供了一条迅速、有效和可靠地构建更智能系统的有力渠道。更智能的系统往往意味着更大复杂性。这既是优势也是挑战。它之所以是优势，是因为客户能够打造之前无法构建或很难构建出的产品。而它之所以又是挑战是因为产品的复杂性使正确的设计决策变得更为重要，尤其是在产品生命周期的早期阶段。系统软件、应用和硬件间的相互作用需要新的思考方式和解决系统级问题的新途径。赛灵思通过为客户提供包括软件工具、用户指南、参考手册和参考设计在内的综合性工具箱，助力加快使用 All Programmable SoC 的产品开发，战胜上述挑战。

一个典型的嵌入式开发团队由系统架构师、软件工程师和硬件工程师组成。每位团队成员往往先使用熟悉工具开始设计，而过去这种方法对嵌入式开发通常是行之有效的。但是对于那些没有提前考虑开发方法的团队， All Programmable SoC 的丰富功能可能会造成问题。为使开发团队更具成效，赛灵思已为嵌入式系统开发人员编制了此方法指南。该指南作为 《Vivado Design Suite UltraFast 设计方法指南》(UG949) [参照 16] 的补充，主要面向 FPGA 设计人员。

“方法”一词对不同的人可能有不同的含义。流程图、方法、原理、规则和规定都属于众多含义中的几种可能。本方法指南不会给出一套循序渐进实现成功的流程。相反，它的目的在于为设计人员提供设计嵌入式系统的信息与指导，这样他们在使用上述工具箱时就能做出明智的决策。部分内容普遍适用于嵌入式系统，而其余内容则针对赛灵思 All-Programmable SoC 产品。其内容是用户经验和赛灵思内外积累的系统开发知识的体现。内容包括主要原则、具体的注意事项、最优做法和如何避免常见错误。在某些主题中，提供了用例来阐释概念。

该指南在组织结构上围绕重要的功能方面，与开发团队内的具体技能组合相对应。这些章节包括：

- 第 2 章"系统级考虑事项"
- 第 3 章"硬件设计考虑事项"
- 第 4 章"软件设计考虑事项"
- 第 5 章"硬件设计流程"
- 第 6 章"软件设计流程"
- 第 7 章"调试"
- 第 8 章"SDSoC 环境"

关于章节与团队成员关联性，请查看表 1-1。不过，建议整个团队在开始开发工作前先完整地阅读方法指南。理解其他章节中描述的挑战和提供的指导将受益显著，即便其内容与工程师的直接职责领域并无关联。软件工程师和硬件工程师之间的界限在逐渐模糊，工程师们有必要突破自身主要职责范畴，让团队合作更具成效。例如软件工程人员需要理解底层硬件的工作方式，而硬件工程人员应理解做出的硬件决策对软件产生的影响。

表 1-1 : 与设计团队成员相关的章节

书籍章节	系统架构师	硬件设计人员	软件设计人员
第 2 章"系统级考虑事项"	X	X	X
第 3 章"硬件设计考虑事项"	X	X	
第 4 章"软件设计考虑事项"	X		X
第 5 章"硬件设计流程"		X	
第 6 章"软件设计流程"			X

表 1-1 : 与设计团队成员相关的章节

书籍章节	系统架构师	硬件设计人员	软件设计人员
第 7 章"调试"		X	X
第 8 章"SDSoC 环境"			X

通常，嵌入式设计依照本指南中列出的顺序完成，从系统级设计开始，止于测试与调试。本指南章节采取了便于依任何次序阅读的撰写方式。这一特点在图 1-1 中得到了体现。图中所示的是本指南中各个章节之间的依存关系。

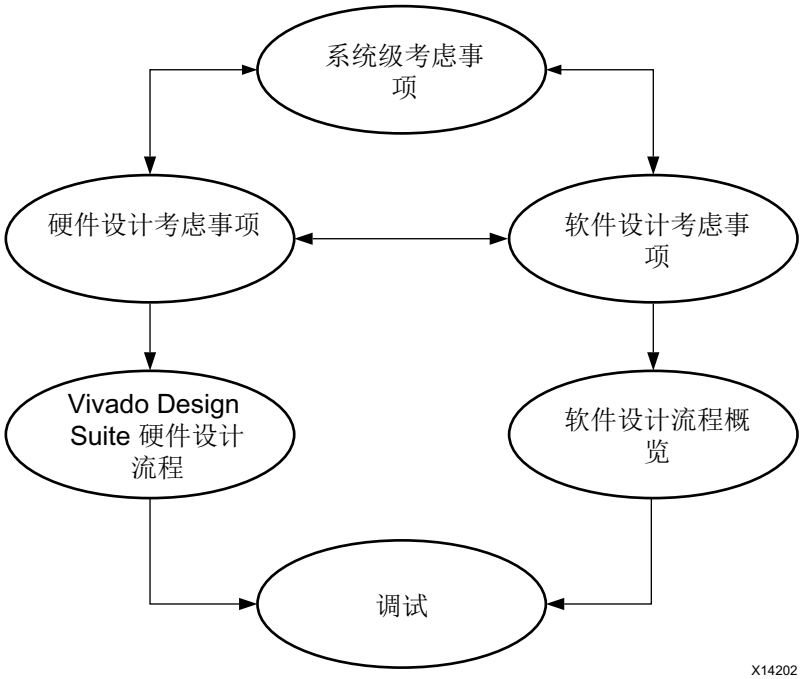

图 1-1 : 本方法指南各章节间的相互关联

注释： 第 8 章"SDSoC 环境"未出现在图 1-1 中，因为它是一个协助用户使用第 2 - 7 章中描述的所有原则和方法构建系统的工具。此外，第 8 章描述的是针对 SDSoC 流程的建议。

在阅读本指南后，设计人员将能够以更明智和更有效的方式运用赛灵思提供的各种工具和附属资料。每个主题领域的知识点应能让设计人员以更深的理解程度阅读 All-Programmable SoC 的详细文档。

熟悉《Zynq-7000 All Programmable SoC 技术参考手册》(UG585) [参照 4] 和《Zynq-7000 All All Programmable SoC 软件开发人员指南》(UG821) [参照 7] 的用户将在阅读本指南后将受益更多。

不熟悉 Zynq®-7000 AP SoC 架构的读者应先参考 《Zynq-7000 All Programmable SoC 总览》(DS190) [参照 29] 了解详细说明。总体而言，该指南能让嵌入式设计团队迅速完成利弊权衡评估，避免瓶颈和问题，从而让团队为成功的嵌入式系统开发做出正确决策。

嵌入式设计方法检查表

要全面发挥嵌入式设计方法的作用，应将本指南与设计方法检查表结合使用。该检查表包含从规划到所有后续设计阶段等整个设计流程中要考虑的常见问题及建议采取的行为。检查表中重点列出了通常难以发现或经常忽略的问题，而这些典型的问题会导致设计决策在后期形成分歧。

赛灵思建议在开展检查表阶段之前先阅读本指南。大多数链接提供对本指南的相互对照和对其他赛灵思文件的链接。这些参考资料可指导您如何解决因这些问题而产生的设计问题。

本检查清单属于赛灵思 Documentation Navigator 的组成部分，是读者在使用赛灵思产品时可用于访问文档的免费工具。读者可以将文档导航器作为单机产品下载或是作为读者的软件开发套件 (SDK) 或 Vivado 安装的组成部分下载（见使用 Documentation Navigator）。

您可用 Documentation Navigator v2015.1 或其更高版本访问检查表功能。在 Documentation Navigator 中可通过以下步骤开始使用设计方法检查表：

1. 点击“Design Hub View”标签。
2. 在左侧菜单顶部点击“Create Design Checklist”。
3. 在新设计检查表对话框中输入信息并点击“OK”。
4. 打开新的检查表。检查表顶部上的各个标签 （见：图 1-2）提供了导航功能。“Title Page”标签提供了使用检查表所需的基本信息。您也可点击其它标签查看检查表相关问题及指南。

图 1-2 : Documentation Navigator 中的“Embedded Design Methodology Checklist”标签

此外，设计方法检查表还提供了电子数据表版本。

http://china.xilinx.com/support/documentation/sw_manuals/c_xtp397-embedded-design-methodology-checklist.xlsx

访问技术文档和培训资料

在适当的时间获得正确的信息，对于及时完成设计并确保整体设计成功而言十分重要。参考手册、用户指南、教程和视频能够帮助您尽快掌握赛灵思工具。本节为您列出了部分技术文档和培训资料的来源。

使用 Documentation Navigator

赛灵思嵌入式工具配套提供“Xilinx Documentation Navigator”，用于访问和管理全套赛灵思软/硬件文档、培训资料和辅助材料。借助 Documentation Navigator，您可查看赛灵思最新及过去的技术文档。通过版本、文档类型或设计任务来筛选技术文档显示内容。结合搜索功能可帮助您快速找到正确的信息。

Documentation Navigator 会扫描赛灵思网站，以检测并提供技术文档更新。 “Update Catalog”功能可提醒您有可用的更新内容，并提供有关文档的具体信息。赛灵思建议您在出现提醒时要更新目录，使其保持最新。您可以为指定的文档建立本地技术文档目录并对其进行管理。

Documentation Navigator 中有一个“Design Hub View”标签。 “Design Hub”是指与设计活动 （Zynq-7000 设计简介、PetaLinux 工具， SDK）相关的文档集。文档和视频被纳入每个设计中心内，以简化相关领域的学习过程。每个设计中心均包含“Embedded Processor Design” （嵌入式处理器设计）部分，以及“Support Resources”（辅助性资料）部分。 “Embedded Processor Design”部分可为新用户提供清晰的入门指导。

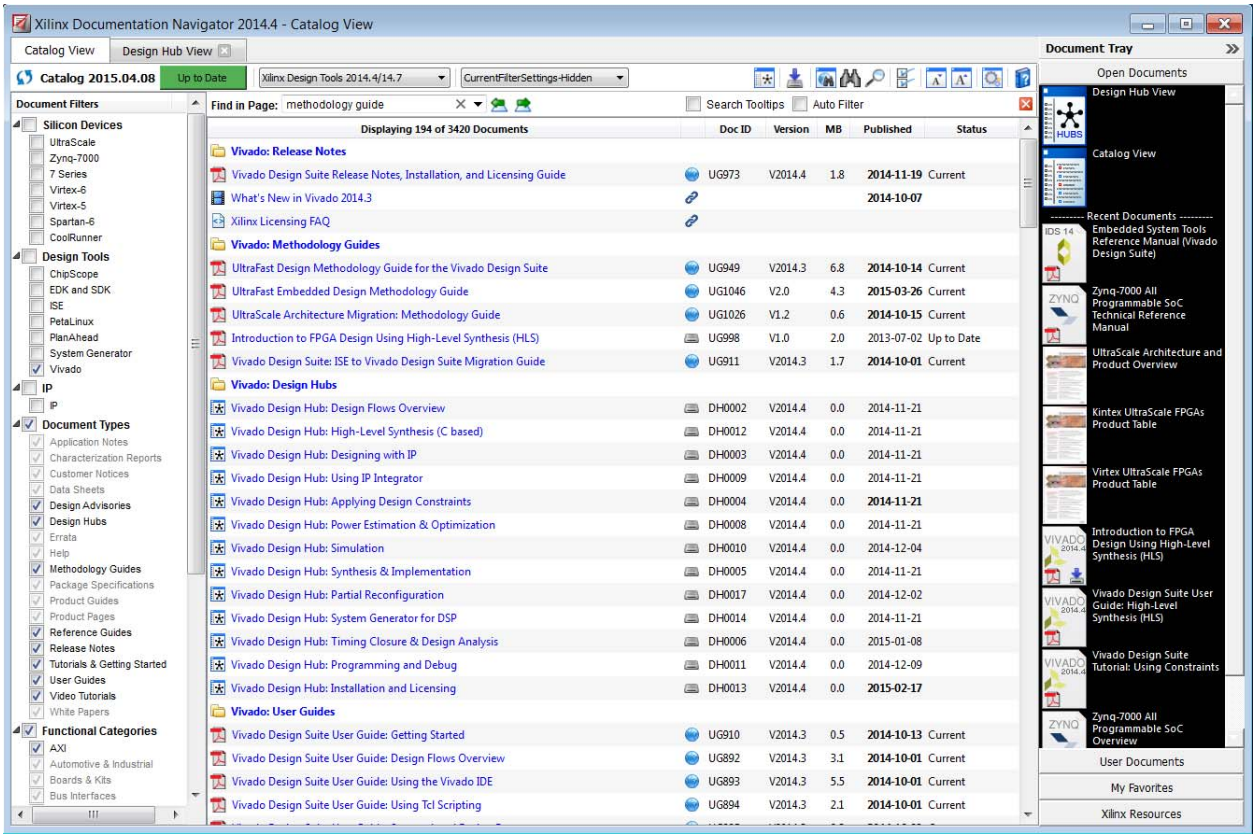

图 1-3 : Xilinx Documentation Navigator “Catalog”视图

访问 QuickTake 视频教程

赛灵思 QuickTake 视频教程为使用赛灵思工具功能提供了指南，包括 SDK、SDSoC，PetaLinux 工具。这些教程简明扼要，可在赛灵思网站 视频教程页面上观看，也可在赛灵思优酷频道观看，还可以本地下载。

提示： 如果连接速度影响观看质量，可将视频下载到本地观看。QuickTake 视频教程也可通过 Documentation Navigator 获得，参见。

XILINX
ALL PROGRAMMABLE™

第 2 章

系统级考虑事项

本章涵盖在使用 Zynq®-7000 AP SoC 进行设计时需要考虑的如下系统级设计问题:

- 第 10 页的“性能”：通常，目标用途决定整体系统性能。系统性能目标需要在硬件与软件之间进行分配，并进一步分配到子组件。
- 第 13 页的“功耗”：系统性能是功耗的主要驱动因素。设计团队经常要在这两个问题之间艰难地做出权衡选择。本节将介绍用于优化 Zynq-7000 AP SoC 的功耗的相关设计考虑因素。
- 第 25 页的“时钟和复位”：要理解什么时钟资源可用，以便规划最充分地利用这些资源的方案，这一点非常重要。同样，复位系统也有很多不同的源，一定要理解它们如何影响不同复位目的地。
- 第 29 页的“中断”：系统级中断环境可为您提供全面的功能集，用以排列您应用中硬件和软件资源的优先顺序。
- 第 32 页的“嵌入式器件安全性”：不同的用途对于安全等级的要求存在很大差异。如您能掌握 Zynq-7000 AP SoC的各种安全特性，那将有助于为您的应用实现正确的安全等级。
- 第 35 页的“剖析 (Profiling) 和划分 (Partitioning)”：您需要做的重要系统级决策是如何在硬件与软件之间对应用系统的功能进行分区。特性剖析工具有助于制定这些决策。

性能

对于 Zynq-7000 AP SoC 器件，系统性能取决于最终的应用目标。例如，实时系统的系统性能可能取决于中断服务例程时延；视频系统的性能可能取决于能否在片外接口上保持 60 帧/秒的帧率。本部分介绍性能目标在不同设计团队成员之间的分配，以及通过数据移动和计算设计选择来实现这些性能目标时需要考虑哪些因素。本部分的最后对 Zynq-7000 AP SoC 器件监控选项进行讨论，让设计人员能够在软件和可编程逻辑 (PL) 中构建自定义的性能监测功能。

系统性能设计目标

系统性能目标需要在 Zynq-7000 AP SoC 器件设计团队中跨不同工程学科进行划分。本文档中讨论的三个工程学科是硬件、软件和系统。

硬件工程师

使用 Zynq 器件的硬件工程师利用 PL 组件、AXI 连接 IP、高速片外接口和定制逻辑来实现设计。 PL 的选择受性能要求的推动，例如吞吐量和时延，但也可受系统软件性能约束以及硬件交互的推动。硬件工程师必须考虑与处理系统 (PS) 的交互，因为数据移动和同步对 PL 吞吐量和时延有很大影响。PL 数据移动和监测点可用来指导设计决策，并在本部分后半部分加以探讨。并未涵盖传统的 PL 指标，例如最大频率和资源利用。如需了解更多信息,请参阅： 《UltraFAST 设计方法指南 (适用于 Vivado Design Suite) (UG949)》[参照 16]。

软件工程师

软件工程师关注在 PS 内运行的系统软件及其与储存器、I/O 和 PL 的交互。例如，与 PL 通信的用户软件具有多个通信选项，每个选项都各有优缺点。软件工程师可用 Zynq 器件独有的软件性能监测功能来调节性能，本部分后半部分将对其加以介绍。另外，原有的 ARM 性能剖析和监控工具形成了丰富的生态系统，可与双核 Cortex-A9 处理器一起使用，

用以优化性能。如需了解性能监测功能方面的更多信息，请参阅 ARM DS-5 Development Studio 性能优化分析器文档 [参照 72]。

系统工程师

对于利用 Zynq 器件实现的设计，其性能目标可以在硬件和软件之间进行划分。在所有设计阶段中都应考虑性能目标的划分。初始阶段，硬件和软件工程师可以相对比较独立地工作，但是需要早期的性能估计来制定现实的性能目标和初步的划分决策。系统工程师必须考虑所有的硬件和软件性能瓶颈，并进行权衡。系统架构师可对数据和通信路径进行早期的性能估计，随后使用系统性能监测点和工具对路径进行精细调节。

所有设计团队成员一般都考虑对数据移动和数据计算性能的影响。在对 Zynq 器件的性能进行设计和优化时，监控系统事件的能力非常有用。这些考虑因素将在以下部分中进行介绍，包括建议采用的设计方法，用以指导 Zynq 器件的设计流程。

系统数据移动

在系统中移动数据是常见的系统级性能问题。Zynq 器件有多个 AXI 主机，可以直接或在 DMA 的协助下驱动事务处理。本部分将介绍实现 Zynq 器件中的数据移动所涉及的不同选项和权衡因素。

ARM CPU 可利用直接储存器传输（例如 memcpy）来移动数据。这种方式可用来进行 4KB 或更小的数据块传输，但传输较大的数据块需要 DMA 的协助。还要考虑数据的来源和目的缓冲器位置。例如，PS DMA 数据传输到 PL 通常要经过 32 位主机 GP 端口。《Zynq-7000 All Programmable SoC 技术参考手册》(UG585) [参照 4] 包含 PS DMA 控制器使用方面的技术。PS DMA 控制器运行自己的微代码。另一种方法是使用 PL 中的 DMA 通过 64 位 ACP 或 HP 端口移动数据，这种方法性能更高，并且可使用 PL 资源。

当确定系统数据移动性能时应考虑 PS 中的其他 AXI 主机。IOP DMA 存在于多个 IP 模块 中：GigE 控制器、SDIO 控制器、USB 控制器和器件配置接口 (DevC)。《Zynq-7000 All Programmable SoC 技术参考手册》 (UG585) [参照 4] 介绍了 IP 模块功能；所提供的驱动是为了方便它们在操作系统中使用，例如 Linux 和独立设计。《Zynq-7000 AP SoC 中 PS 与 PL 以太网性能和 PL 以太网的巨型帧支持》 (XAPP1082) [参照 38] 进一步介绍了 GigE 控制器性能特性。其余内核具有可提供附加功能的驱动层，但这里没有对它们的性能特性加以介绍。

AXI 主机可通过 PL 对 ACP、HP 或 slave GP 端口上的数据移动进行驱动。下面几个部分将进一步介绍这些端口以及相关性能。

- 第 80 页的“ACP 和高速缓存一致性”
- 第 82 页的“PL 高性能端口访问”
- 第 91 页的“GP 和来自 APU 的直接 PL 访问”

连接到这些端口的 DMA 最适合用于在 AXI4 储存器映射接口与 AXI4 流接口之间进行转换，这样，对于以数据为中心的处理 （例如视频流水线），对其储存器映射的访问就可实现完全分离。但是， DMA 并非总是首选的方法。如果流接口性能低而且无需考虑，那么 AXI4-Stream FIFO 内核是一种从处理器获得和同步 AXI-Stream 接口的简单方法。用户应用程序可具有自定义的 AXI 主机，这些主机使用标准 AXI 储存器映射事务处理来进行直接的数据移动。这种自定义数据传输的吞吐量和时延性能特征最终会与标准 PL DMA 数据传输非常相似。可使用第 12 页的“系统监控”中描述的计数器和定时器进行性能对比。

DMA 将数据从源位置移动到目标位置。储存器经常作为数据缓冲器，用来平衡数据源、处理或数据接收器的速度差异，从而确保处理阶段能实现最大吞吐量。

将片外储存器连接到自定义 MIO 或 EMIO 引脚，以实现片外数据缓冲器位置。储存器特性会影响移动大缓冲器时的性能，例如 SD 卡上的文件系统，或者经 GigE 控制器实现的网络附加存储。

就片上缓冲而言，OCM、L2 缓存和 DDR 控制器是 PS 中三个主要的可共享缓冲空间源。在处理器与 ACP 端口之间共享数据时， L2 缓存和 DDR 控制器可提供出色的缓冲器访问时延性能。只有 ACP 能够通过 PL 访问 L2 缓存。比起 ACP，HP 端口更适合用于对 DDR 进行高带宽访问。软件应用程序可以使用 OCM 作为 PL 中所有主机均可访问的 256 KB 暂存器。使用 OCM 的优势在于，它具有出色的随机访问时延，而 L2 缓存和 DDR 储存器则受益于储存器访问位置。

系统计算

可在 ARM 内核或 PL 中完成计算。典型的 Zynq 器件具有在软件中编写的控制层，并在 ARM 内核上执行。以数据为中心的计算在 PL 中采用现有的和定制的 IP 的组合来执行。对于像线速率包处理这样的高性能任务，可将计算与控制移动到距离数据更近的位置。这通常需要定制的 PL IP 来管理数据流。

将计算从软件移动到 PL 是 Zynq-7000 AP SoC 平台的一个重要优势。并非所有软件都能移动到 PL，尤其是预编译代码、复杂库程序以及 OS 服务。如果可以将软件组件移动到 PL，那么考虑性能指标 （例如总的应用程序运行时间）时，应该在数据移动成本和加速优势之间进行权衡。可使用软件特性剖析工具进行成本与优势分析，以根据 Amdahl 法则确定潜在的加速可能。

$$S = \frac{1}{(1-\alpha) + \alpha/p}$$

在上面的公式中，*S* 是整体性能提高，α 是可以通过硬件加速实现算法加速的百分比。因此，1-α 是无法实现改进的算法的百分比。变量 *p* 是因加速而实现的速度提升。例如，要对完全在软件中实现的计算算法实现加速，可使用特性剖析工具 （例如 TCF 特性剖析器）找出频繁使用的功能。该功能使用所占的时间百分比与较大的 α 数相对应。然后，通过在硬件中实现，可对该功能进行加速。该功能的加速可映射到 *p*。得益于硬件加速，常用功能除以大型乘数可实现最大的算法计算性能改进。

特定功能的加速包含两个方面：进出加速模块的数据传输，以及由加速器执行的实际计算。当用 DMA 执行数据传输时，测量乘数 *p* 时必须考虑设置和管理 DMA 所用的开销。开销太大会束缚 *p* 值，可能导致 *p* 因数较小，从而降低改进效果。如需了解更多信息，请参阅第 35 页的 “剖析 (Profiling) 和划分 (Partitioning)”。

高层次综合是将软件组件移动至 PL 的绝佳方法，用以探索加速选项。如果所选的软件不适合高层次综合，那么赛灵思还可提供丰富的可编程逻辑 IP 库和对应驱动，用以取代在软件中实现的功能，从而改进整体系统性能。

系统监控

赛灵思具有丰富的工具生态系统，可用于监控 ARM 处理器。在 Zynq 器件中，完整的系统级性能监控还要用到 PS 和 PL 中提供的模块。它们是：

- **SCU 全局定时器 (PS)**：SCU 全局定时器可用来对单个时钟域内的系统事件进行时间戳标记。或者，操作系统经常提供高精度定时器，用于追踪软件事件，例如 Linux clock_nanosleep。
- **ARM 性能监控单元 (PS)**：每个 ARM 内核都有一个用来计算微架构事件数量的性能监控单元 (PMU)。这些计数器可直接由软件通过操作系统实用程序进行访问，或利用芯片调试器 （例如 Linux Perf 或 ARM Streamline）进行访问。计数器在软件开发套件 (SDK) 2014.2 的性能视图中可见。如需了解更多信息，请参阅 ARM 《Cortex-A9 技术参考手册》[参照 75] 中的“性能监控单元”部分。
- **L2 缓存事件计数器 (PS)**：L2 缓存具有可访问的、用于测量缓存性能的事件计数器。这些计数器在 SDK 的性能计数器视图中可见。如需了解更多信息，请参阅 《Zynq-7000 All Programmable SoC 技术参考手册》(UG585) [参照 4]。
- **GigE 控制器(PS)**：千兆位以太网控制器具有统计计数器，用来追踪接口上接收和传输的字节。如需了解更多信息，请参阅 《Zynq-7000 All Programmable SoC 技术参考手册》(UG585) [参照 4]。
- **AXI 性能监控 (PL)**：该内核可添加到 PL 中，用来监控吞吐量和时延等 AXI 性能指标。追踪功能可以为 AXI 踪迹（例如具有时间戳的 AXI 事务处理起点和终点）进行时间戳标记，以观察每个事务处理的时延。如需了解更多信息，请参阅 AXI 性能监控网页 [参照 43]。
- **AXI 定时器(PL)**：该内核可添加到 PL，以在 PL 中提供自由运行的定时器。这个定时器用来为 PL 时钟域中的事件进行时间戳标记。如需了解更多信息，请参阅 AXI 定时器/计数器网页 [参照 44]。
- **AXI 流量生成器 (PL)**：该内核可生成到 PS 接口的多种流量模式。当与 AXI 性能监控器一起使用时，流量生成器有助于提供早期的系统级性能估计。该内核可用来估计数据移动成本及验证设计分区选择。如需了解更多信息，请参阅 LogiCORE™ AXI 流量生成器网页 [参照 45]。

利用这些监控模块有助于了解整个系统性能表现。事件计数器和 PL 监控模块可针对具体设计性能目标进行自定义，或者用来获取系统性能的高级视图。在早期使用内建的性能监控功能可在整个设计周期内获得性能反馈，从而实现以数据为驱动的系统架构决策。

可通过不同的方法来运用监控数据，或从不同模块获得监控数据。大多数外设都配有实例应用，这些实例与赛灵思软件开发套件 (SDK) 配套提供的相应裸机驱动打包在一起。赛灵思 Linux 驱动维基页面 [参照 50] 中提供了可用的 Linux 驱动列表。

赛灵思 SDK 提供内建的系统性能建模与分析功能。性能监控器从系统中收集数据，这些数据既可以是实时数据，也可以是流量生成器创建的用来对现实事务处理进行建模的数据。可利用赛灵思 SDK 可视化工具显示这些数据，其中的统计信息包括：事务处理数量、带宽以及连接到 HP 和 ACP 端口的 APM （配置文件模式）的时延。如需了解更多信息，请参阅第 37 页的“赛灵思 SDK 系统调试器”。

功耗

Zynq-7000 AP SoC 的功耗是系统架构师和电路板设计人员需要考虑的一个重要因素。对于大部分应用系统来说功耗都是重要的考虑因素，而且，有些应用还指定了每张卡或每个系统的最大功耗。因此，设计人员必须在设计过程早期考虑功耗问题，而且往往是从器件选择开始。

降低 SoC 功耗能够减小电源的功耗，简化电源设计和热管理，降低对功耗分配层的要求，从而改善电路板设计。低功耗还有助于延长电池寿命及提高可靠性，因为温度较低的运行系统持续时间更长。

降低系统功耗需要全面且集中的方案来实现最佳效果。本部分介绍 Zynq-7000 AP SoC 功耗的几个方面，包括 PS 和 PL 的架构与特性、与 PL 有关的功耗组件，以及处理技术。另外，还介绍了功耗以及估计和测量功耗的传统方法。

功耗挑战

根据摩尔定律，晶体管尺寸随着处理技术的发展而缩小。随着尺寸减小，每个晶体管的漏电流 （器件不进行开关切换时所消耗的电流）会增加，导致静态功耗增大。提高 SoC 性能需要更高频率的时钟，导致动态功耗增大。因此，静态功耗由晶体管漏电流决定，动态功耗由晶体管开关频率决定。随着每代晶体管尺寸的缩小， FPGA 上可以装入更多数量的晶体管，导致问题更加严重。更多晶体管导致每个 FPGA 器件上产生更多漏电流以及更多晶体管在更高时钟频率下开关。

这就需要设计人员在设计周期更早地解决电源和热管理问题。在器件上使用散热器可能不足以解决这些问题。设计人员必须找机会减小设计逻辑的影响。

图 2-1 所示为在设计周期的不同节点，为降低功耗可以采取的行动。在设计过程早期解决功耗问题可实现最大优势。

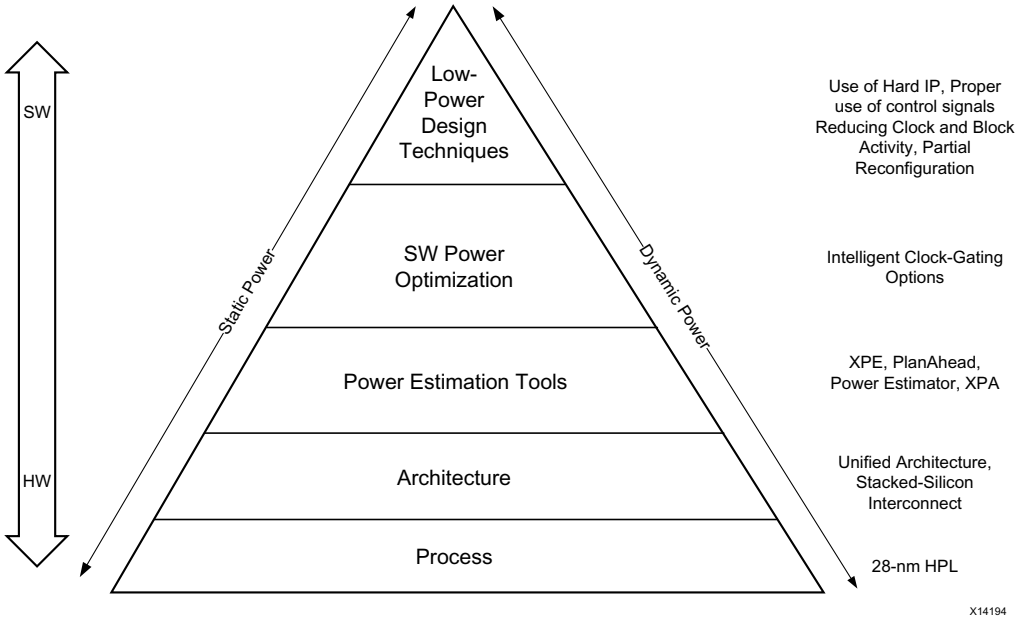

图 2-1 : 贯穿整个设计周期管理功耗问题

功耗和信号

Zynq-7000 AP SoC 器件被分成多个功耗域，如图 2-2 中所示。

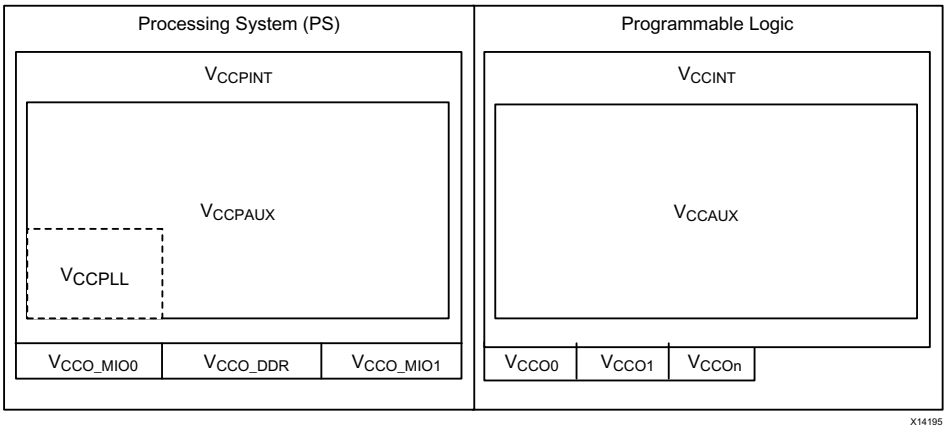

图 2-2 : Zynq-7000 AP SoC 功耗域

PS 和 PL 电源是独立的；不过，当 PL 电源活动时 PS 电源必须出现。在不需要 PL 的应用电路中，可将 PL 断电。表 2-1 中概括了 PS 和 PL 功耗引脚。以下文件介绍了电压排序和电气规格 《Zynq-7000 All Programmable SoC （Z-7010、 Z-7015 和 Z-7020）： DC 和 AC 开关特性》(DS187) [参照 28]。

表 2-1 : 功耗引脚

类型	引脚名称	额定电压	功耗引脚説明
PS 功耗	V_{CCPINT}	1.0V	内部逻辑
	V_{CCPAUX}	1.8V	I/O 缓冲器前置驱动器
	V_{CCO_DDR}	1.2V 至 1.8V 个	DDR 存储器接口
	V_{CCO_MIO0}	1.8V 至 3.3V 个	MIO 单元 0, 引脚 0:15
	V_{CCO_MIO1}	1.8V 至 3.3V 个	MIO 单元 1, 引脚 16:53
	V_{CCPLL}	1.8V	三个 PLL 时钟，模拟
PL 功耗	V_{CCINT}	1.0V	内部核逻辑
	V_{CCAUX}	1.8V	I/O 缓冲器前置驱动器
	$V_{CCO\_\#}$	1.8V 至 3.3V 个	I/O 缓冲器驱动器 （每个单元）
	V_{CC_BATT}	1.5V	PL 解密秘钥储存器备份
	V_{CCBRAM}	1.0V	PL 模块 RAM
	$V_{CCAUX\_IO\_G\#}$	1.8V 至 2.0V 个	PL 辅助 I/O 电路
XADC	VCCADC	1.8V	模拟电源和接地。
接地	GND	接地	数字和模拟接地

PS 功耗域

如需了解关于 Zynq-7000 AP SoC PS 功耗域的更多信息，请参阅第 5 章：《Zynq-7000 All Programmable SoC PCB 设计指南》(UG933) [参照 14] 中的“处理系统 (PS) 功耗和信号”。

PL 功耗域

需要多个电源为 Zynq-7000 AP SoC 中的不同 PL 资源供电。不同资源在不同电压水平下工作，以提高性能或信号强度，同时保持对噪声和寄生效应的抵抗力。

第 16 页的表 2-2, 中所示为 Zynq-7000 AP SoC 中的 PL 资源通常使用的电源。该表格仅作为指导，因为具体情况在 Zynq-7000 AP SoC 系列中可能有所不同。

表 2-2： PL 电源

电源	受供电的资源
V_{CCINT} & V_{CCBRAM}	• 所有 CLB 资源 • 所有布线资源 • 整个时钟树，包括所有时钟缓冲器 • 块状 RAM/FIFO • DSP slice • 所有输入缓存 • IOB (ILOGIC/OLOGIC) 中的逻辑元件 • ISERDES/OSERDES • 三模以太网 MAC • 时钟管理器 (DCM、 PLL、等) (次要) • MGT 的 PCIE 和 PCS 部分
V_{CCAUX} & V_{CCAUX_IO}	• 时钟管理器 (MMCM、 DCM、 PLL、等) • IODELAY/IDELAYCTRL • 所有输出缓存 • 差分输入缓存 • 基于 V_{REF} 的单端 I/O 标准，例如 HSTL18_I • 相位器
V_{CCO}	• 所有输出缓存 • 些输入缓存 • 数控阻抗 (DCI) 电路，又名片上端接 (OCT)
MGT	• 收发器的 PMA 电路

板级配电系统

在印刷电路板上，配电系统 (PDS) 将电力从配电至各种用电的芯片和器件。尽管 PDS 设计从简单到复杂不尽相同，但必须满足三个基本要求：

- PDS 必须提供经过良好调节的电压。功率调节必须在经过稳压的电源电路中进行，并得到一个或多个大容量电容器和 LC 滤波电路的支持。
- PDS 必须在所有电流负载条件下，在电路板的所有点上保持稳定。要在所有负载条件下达到稳定，需要满足两个子条件：
  - 分配系统必须具有低电阻和低电感。这通常要求系统具有用于分配的电源层和返回层，而且各种器件连接处采用低电感的垫片和过孔。
  - 无论哪里，无论何时需要电荷，都必须能提供可用的电荷。所需的电荷通常存储于电路板周围的旁路电容器中，少部分来自分散的平面电容。
- PDS 必须安静，意味着当器件开关时不会产生噪声，即不会干扰其他器件或生成 EMI。

如需了解针对 Zynq 器件的配电系统的更多信息，请参阅第 3 章：“配电系统”，《Zynq-7000 All Programmable SoC PCB 设计指南》(UG933) [参照 14]。

功耗管理

使用 Zynq-7000 AP SoC 有助于降低系统的静态功耗。 Zynq-7000 AP SoC PS 是经优化的硅元件，包含双核 ARM Cortex-A9 CPU 以及集成外设。PL 基于赛灵思 7 系列架构，采用 28nm 高性能、低功耗 (HPL) 流程构建而成，可在提供高性能操作的同时实现功耗的显著降低。选择了基于 HPL 流程构建的器件后就无需使用复杂、昂贵的静态电源管理方案。

降低系统功耗有许多不同的方法。以下部分将提供几个设计优化技巧，以满足系统功耗要求。

PS 功耗管理

本节将介绍对 Zynq-7000 AP SoC PS 的功耗进行优化时所涉及的考虑因素。这些因素包括：APU 单元、 PS 外设、时钟和 PLL、缓存、 SCU，以及 OCM 的电源管理。假定设计人员了解电源管理权衡对整体系统的影响。

系统设计考虑事项

PS 组件的电源管理方式如下：

- **应用处理单元 (APU)**：Zynq-7000 AP SoC APU 支持动态时钟门控。该功能可通过 CP15 电源控制寄存器进行启动。如果该功能启动，多个 CPU 内部模块的时钟会在空闲期内被动态禁用。门控模块为：
  - 整数内核
  - 系统控制模块
  - 数据引擎

  通过降低处理器内核电压和工作频率，可实现功耗的两倍降低。如需了解更多信息，请参阅 《Zynq-7000 All Programmable SoC 技术参考手册》(UG585) [参照 4] 的电源管理部分。
- **PS 外设**：PS 支持多个时钟域，每个时钟域都有独立的时钟门控。当系统出于运行模式，用户可关闭不用的时钟域以降低动态功耗。PS 外设——例如定时器、DMA、SPI、QSPI、SDIO，以及 DDR 控制器——的时钟可进行独立门控以节省功耗。如需了解系统时钟的更多信息，以及如何使用分频器、门电路和多路复用器控制它们，请参阅《Zynq-7000 All Programmable SoC 技术参考手册》(UG585) [参照 4] 中的第 25 章，“时钟”。
- **高速缓存**：L2 缓存控制器支持以下动态功耗降低功能。这些功能由 l2cpl310.reg15_power_ctrl 寄存器中相应的使能位控制。
  - **动态时钟门控**：当动态高级时钟门控功能启动后，缓存控制器时钟在控制器空闲时停止。时钟门控功能等控制器空闲几个周期之后停止
    时钟。
  - **待机模式**：L2 缓存控制器待机模式可与驱动 L2 缓存控制器的处理器等待中断 (WFI) 模式一起使用。当处理器处在 WFI 模式，而且待机模式启动时， L2 缓存控制器内部时钟即停止。如需了解关于使用 WFI 方面的更多信息，请参阅《Zynq-7000 All Programmable SoC 技术参考手册》(UG585) [参照 4] 中的第3章，“应用处理单元”。

  动态时钟门控功能是待机模式的超集。在待机模式下，时钟门控被限定在 WFI 状态，因此， L2 缓存访问在正常运行条件下具有更强的可预测性。
- **片上存储器 (OCM)**：通常， OCM 可用于在低功耗模式 （例如 Linux 待机模式）下降低总功耗。例如，当 DDR 处在低功耗模式时， OCM 可用于存储可执行代码。
- **监听控制单元 (SCU)**：SCU 具有待机模式，可通过设置 mpcore.SCU_CONTROL_REGISTER 中的相应位来启动该模式。待机模式启动后，当满足以下条件时，内部 SCU 时钟停止：
  - CPU 处于 WFI 模式。
  - ACP 上没有挂起的请求。
  - SCU 中没有剩余活动。

  当 CPU 离开 WFI 模式或者 ACP 上发生请求时， SCU 恢复正常运行。
- **PLL**：PLL 功耗取决于 PLL 输出频率，因此使用较低的 PLL 输出频率可以降低功耗。另外，将不用的 PLL 断电也可降低功耗。例如，如果所有时钟生成器都可由 DDR PLL 来驱动，那么就可以禁用 ARM 内核和 I/O PLL 以降低功耗。 DDR PLL 是唯一能够驱动所有时钟生成器的单元。当不用时，每个时钟都可以被单独禁用。有些情况下，单个子系统包含附加的时钟禁用功能以及其他电源管理特性。
- **物理储存器**：Zynq-7000 AP SoC 支持不同类型的物理储存器，例如 DDR2、 DDR3 和 LPDDR2。所支持的 DDR 储存器类型可运行 16 位和 32 位数据。 DDR 功耗在总功耗中占很大一部分，因此降低 DDR 功耗是降低系统功耗的重要方法。降低 DDR 功耗时需要考虑的内容包括：
  - DDR 控制器运行速度。
  - DDR 宽度选项，以及 ECC 是启动还是禁用。

- 所用的 DDR 芯片数量。
- DDR 类型，例如使用 LPDDR 大幅降低电压。
- 低功耗运行中使用不同 DDR 模式，例如 DDR 自刷新模式。如需了解更多 DDR 低功耗工作模式下的功耗信息，请参阅相应 DDR 标准。

如需了解关于实现 DDR 控制器时钟门控方面的更多信息，请参阅 《Zynq-7000 All Programmable SoC 技术参考手册》(UG585) [参照 4] 中的“时钟”章节。

- I/O：I/O 器件（例如 MIO 和 DDR IO）也会增加总功耗。如需了解关于 I/O 缓冲器控制电源管理方面的更多信息，请参阅 《Zynq-7000 All Programmable SoC PCB 设计指南》(UG933) [参照 14] 中的 “SelectIO 信号” 章节。

软件支持

Linux 内核支持以下电源管理状态：

- S0：冻结或低功耗空闲这是纯软件的精简低功耗通用状态。
- S1：待机或中断加电。所有处理器缓存清空，指令执行停止。维持处理器和 RAM 的电源。
- S3：中断-至-RAM。系统和器件状态保存到储存器。所有器件被中断和断电。 RAM 保持加电状态。

如需了解针对 Zynq-7000 AP SoC 的 Linux 电源管理支持方面的更多信息，请参阅 Zynq 电源管理维基页面 [参照 64]。

该链接还提供关于为 Zynq-7000 AP SoC 实现的 CPU 扩展框架方面的更多信息。CPU 扩展框架用来扩展运行时 CPU 频率对于不需要高处理性能的应用场合，可降低 CPU 频率以满足需求。与较高频率下的运行相比，较低时钟频率可显著降低运行功耗，

PL 功耗管理

在不需要 PL 的应用电路中，可将 PL 断电。为此，需要为 PS 和 PL 提供单独连接的电源。可断开的 PL 电源包括 VCCINT、 VCCAUX、 VCCBRAM，以及 VCCO。请参阅相应数据手册，以确定正确的电源排序。

当 PL 断电后，配置就会丢失，再次加电时必须重新进行配置。软件应确定何时可以安全地为 PL 断电。

本部分介绍在 Zynq-7000 AP SoC 上对 PL 功耗进行优化时的设计考虑因素。

逻辑资源利用

PL 资源使用占 Zynq-7000 AP SoC 总功耗相当大的一部分。所使用的 CLB 资源、专用硬件和布线的数量取决于具体设计，并可增加 PL 的静态和动态功耗。对 PL 架构进行深入理解有助于设计人员充分利用硅片资源。

为了降低功耗，设计人员必须找机会减少设计中的逻辑数量。这样有助于使用较小的器件，从而降低静态功耗。有一种选择是使用专用硬件模块，而非在 CLB 中实现功能。这样有助于降低静态和动态功耗，并且更容易满足时序要求。由于晶体管总数少于使用 CLB 逻辑构建的同等组件，模块能够降低静态功耗。

设计人员可使用 IP 目录自定义专用硬件，从而将具体资源例化。未使用的 PS IP 可用于其他不太明显的任务。例如，DSP48 slice 有很多逻辑功能，比如乘法器、加法器和累加器，宽逻辑比较器，移位器，模式匹配器，以及计数器。块状 RAM 可用作状态机、数学函数和 ROM。

《UltraFAST 设计方法指南 (适用于 Vivado Design Suite) (UG949)》[参照 16] 中介绍了所需的大部分编码技巧。

管理控制集

控制信号 （用来控制时钟、设置、复位和时钟使能等可控制同步元件的信号）可影响器件的密度、使用和性能。需要遵守一些指导以使这些信号的功耗影响保持最小。

避免在寄存器或锁存器上同时使用设置和复位。赛灵思 FPGA 中的触发器可支持异步和同步的复位和设置控制。但是，底层触发器本身一次只能够以原生形式实现一个设置、复位、预设或清零功能。在 RTL 代码中指定其中一种以上的功能，会导致使用触发器的 SR 端口的一个条件，以及在 PL 中的另一个条件的实现，从而占用更多 PL 资源。

如果其中一个条件是同步的，而另一个是异步的，那么异步条件通过 SR 端口来实现，同步条件在 PL 中实现。总之，应避免一个以上的设置、复位、预设或清零条件。另外， slice 中每个触发器组 （包含四个触发器）只有一个属性能够确定触发器的 SR 端口是同步还是异步。

应使用高电平有效控制信号，因为寄存器上的控制端口是高电平有效的。低电平有效信号使用更多查找表，因为它们在驱动寄存器控制端口之前需要反相逻辑。LUT 可能已经有了其他输入，以至于反相逻辑可能需要使用另一个 LUT。使用低电平有效控制信号会使实现的运行时间加长，并导致器件使用率降低，影响时序和功耗。因此，不推荐在 FPGA 设计中使用低电平有效
复位。

尽可能在 HDL 代码或例化组件中使用高电平有效控制信号。如果无法指定设计中的控制信号极性，应在顶层代码层级中将信号反相。 I/O 逻辑可吸收调用的的反相器，而且无需附加 FPGA 逻辑或布线资源，从而可实现更好的利用率、性能和功耗。

管理设置和复位

对不必要的设置和复位进行编码，可以阻止移位寄存器 LUT (SRL)、 LUT RAM、块状 RAM，以及其他逻辑结构的调用。尽管编码比较棘手，但可以使很多电路自动复位或者根本不需要复位。例如，假如触发器在数据流水线中，基本上没必要使用复位。几个周期后，整个流水线就可以运行，任何错误数据都会从系统中清理出去。

减少设置和复位的使用能提高器件利用率，从而实现更好的布局，提高性能，并降低功耗。

请参阅灵活运用复位：如需了解有关复位设计的更多信息，请参阅 《局部思维超越全局思维白皮书》(WP272) [参照 32]。

时钟门控

PL 动态功耗由工作时钟频率 (fclk)、节点
电容 (C)、 FPGA 工作电压 (V) 以及各个设计节点上的转换活动 (α) 决定。动态功耗的计算公式为：

$$\text{动态功耗} = \alpha \times \text{fclk} \times C \times V2 \qquad \text{等式 2-1}$$

对于大多数设计而言，部分参数由 FPGA 技术 （例如工作电压）或设计要求 （例如工作频率）决定。

如果有些路径的结果用不到，需要停止转换活动，这种情况下，最常用的方法是对时钟或数据路径进行门控。时钟门控可以停止所有同步活动，防止数据路径信号和毛刺传播。

Vivado® 工具能分析描述内容和网表，以检测到不需要的条件。然而，对设计人员所了解的应用、数据流和相关性而言，有些内容该工具没有提供，而是要由设计人员指定，以进一步去除不需要的条件。

有几个设计节点不会影响 PL 输出，但会继续进行切换。每个触发器、块状 RAM 和 DSP48 都有局部时钟使能。设计人员可对局部时钟使能信号进行门控，以消除不必要的触发器、块状 RAM 和 DSP 切换，如图 2-3 所示。

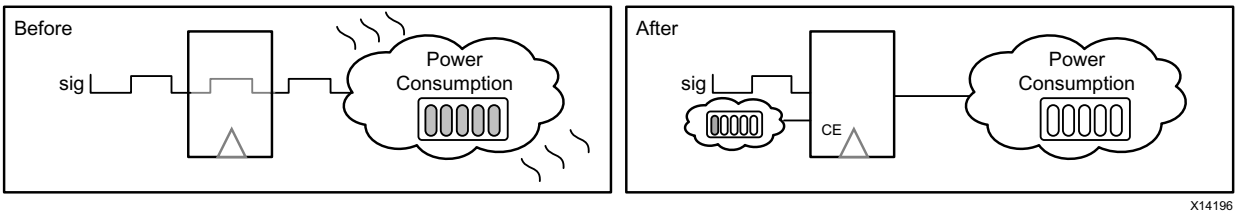

图 2-3 : 对局部时钟使能信号进行门控，以消除不必要的切换

设计人员应确保门控信号可以控制的元件数量尽可能多。例如，在其源位置对时钟域进行门控比用时钟使能信号控制每个负载更能节省功耗。设计人员可根据应用情况，利用 PL 中的不同时钟缓冲器原语对时钟进行门控。

BUFGCE 原语是一个具有时钟使能的全局时钟缓冲器。 BUFGCE 可用来对全局时钟或时钟域进行无毛刺的动态门控。使用该资源还能减少高时钟使能扇出，并节约 PL 结构布线资源。

如果需要对 PL 中的具体逻辑功能或时钟域进行门控，可使用 BUFHCE 或 BUFRCE 原语。这些原语可减少负载和电容，从而降低 PL 动态功耗。

BUFGMUX_CTRL 原语可用来将时钟分配至具体的区域，并避免不需要的 PL 结构动态切换。它还可用于在快速与慢速时钟之间进行切换，以降低功耗。

经常会有几个设计节点不会对 PL 输出造成影响，但是会持续进行切换，从而导致不必要的动态功耗。 FPGA 时钟使能信号可用于对这些节点进行门控。

充分利用时序约束在低功耗设计中也十分重要。如果应用电路在控温环境中运行，那么可降低其额定值以满足时序要求。为了使用最大指定时钟速率，应当对设计进行约束。如果使用的时钟速率高于必要值，通常会产生以下负面影响：

- 资源共享减少，导致使用更多 PL 资源。
- 经常需要对逻辑和寄存器进行复制，以满足严格的时序约束。
- 布线量增加。
- PL 专用功能的调用减少。

这些不利因素会严重影响动态功耗。

模块 RAM

块状 RAM 的功耗直接与它的开启时间成正比。为了节省功耗，应在不使用块状 RAM 的设计中将块状 RAM 使能端驱动至低电平。对功耗进行优化时，块状 RAM 使能率和时钟速率都是重要的考虑因素。

应在有效的读取或写入周期内启动块状 RAM。综合工具可能不会推断这些原语，因此，如果需要这些原语，应该利用原理图查看器对它们的调用进行验证，并在必要时候才进行例化，以节省功耗。

布局规划

跨越多个时钟域的设计会使用更多时钟资源并消耗更多功耗。在可能的情况下，应该将间歇使用的逻辑放在单个时钟域内，这有助于降低功耗，如图 2-4 中所示。尽管工具会尝试自动执行，但是有些设计可能需要手动操作，例如应用面积约束来达到此目的。

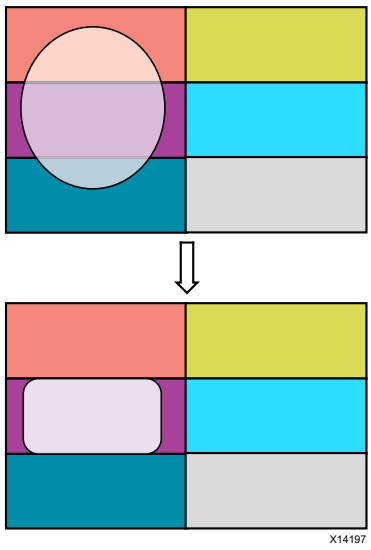

图 2-4 : 将间歇使用的逻辑约束到单个时钟域。

限制数据移动是另一种功耗降低技巧。避免围绕 PL 移动操作数，而是仅移动结果，如图 2-5 中所示。使用更少和更短的总线能够减小电容，从而改善性能，并降低功耗。布局规划过程中必须考虑管脚布局和相应的逻辑设计。

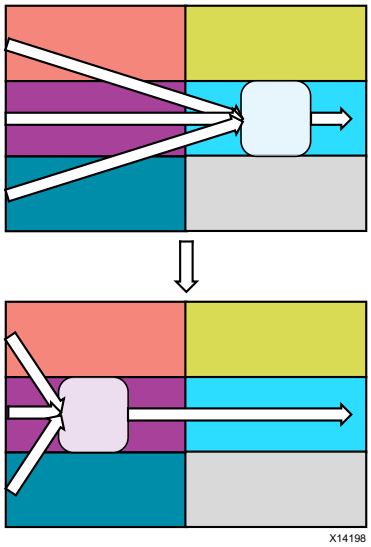

图 2-5 : 限制数据移动

I/O 功耗

I/O 功耗可在总功耗中占较大比重。对于有些设计，尤其是储存器使用量很大的系统，总功耗中多达一半都出自 I/O。

有些接口不需要快速的差分 I/O 功能。有的 I/O 标准，例如 HSLVDCI，可以在 FPGA 至 FPGA 通信和低速储存器接口中节省相当多的功耗。

所有 Zynq-7000 AP SoC 都提供可编程压摆率和驱动强度，用以降低 I/O 动态功耗。该器件支持数控阻抗 (DCI) 技术，并可实现三态状态。 I/O 输出启动状态下， DCI 无端接功耗，因此，器件只在进入周期才消耗端接功耗。

Zynq-7000 AP SoC 整合了针对 HSTL 和 SSTL 的用户可编程接收器电源模式。通过控制每个 I/O 上的可编程电源模式，可在功耗与性能之间进行权衡，以降低 DC 功耗

Zynq-7000 AP SoC 具有针对高性能和低抖动而进行优化的收发器。收发器提供多种低功耗运行特性，帮助设计人员自定义操作的灵活性和粒度，以实现功耗与性能权衡。

在收发器中，用户可使用共享 LC PLL 来节省功耗。在线速率相同的四通道设计 （例如 XAUI）中，用户可以使用四通道 PLL 而非单个通道 PLL。同样，由于 PLL 可在一定范围内以较高和较低的速率运行，因此可选择较低的运行范围以节省功耗。

可启动 RXPOWERDOWN 和 TXPOWERDOWN 选项。可在最低功耗模式 （例如 PCIe 系统中最常用的系统 D3 状态）下实现 PLL 断电。

您还可以通过以下方法节省 I/O 功耗：

- 利用时分复用技术减少 I/O 数量。
- 使用 I/O 数量最小的设计分区，这样有助于在不使用时关闭 I/O 组。
- 减少单元中所使用的 I/O 标准的数量。

部分重配置

降低静态功耗的一种方法是直接使用较小的器件。通过部分重配置，设计人员可在PL中为逻辑模块分配时间片，并独立运行设计的不同部分。这样就可以使用小得多的器件进行设计，因为并非设计的每个部分在 100% 的时间内都需要用到。

部分重配置有可能降低动态功耗以及静态功耗。例如，很多设计必须非常快速地运行，但是这种最高性能可能只在很小的时间百分比内需要用到。为节省功耗，设计人员可通过部分重配置用同一设计的低功耗版本将高性能设计置换出来——而非在 100% 的时间内都使用最高性能设计。当系统需要时，用户还可切换回高性能设计。

这个原则也可应用到 I/O 标准上，尤其在设计不需要一直使用高功耗接口的情况下。由于必须采用较大的 DC 电流供电，因此无论活跃度如何， LVDS 都属于高功耗接口。当设计不需要最高性能时，可使用部分重配置将 I/O 从 LVDS 改为低功耗接口，例如 LVCMOS；当系统需要高速传输时，再切换回 LVDS。

功耗估计

可在设计周期的三个不同阶段执行功耗计算。

- **概念阶段**：在这个阶段，可以根据对逻辑容量和活动率的估计结果计算出粗略的功耗估计。
- **设计阶段**：此时能够具体知道设计如何在 Zynq-7000 AP SoC 中实现，并以此来更准确地计算功耗。
- **系统集成阶段**：功耗在实验室环境中测量。

赛灵思可提供一整套工具和文档，用以在整个设计周期中评估系统的热学和电源要求。图 2-6 所示为设计周期中每个阶段可用的工具。

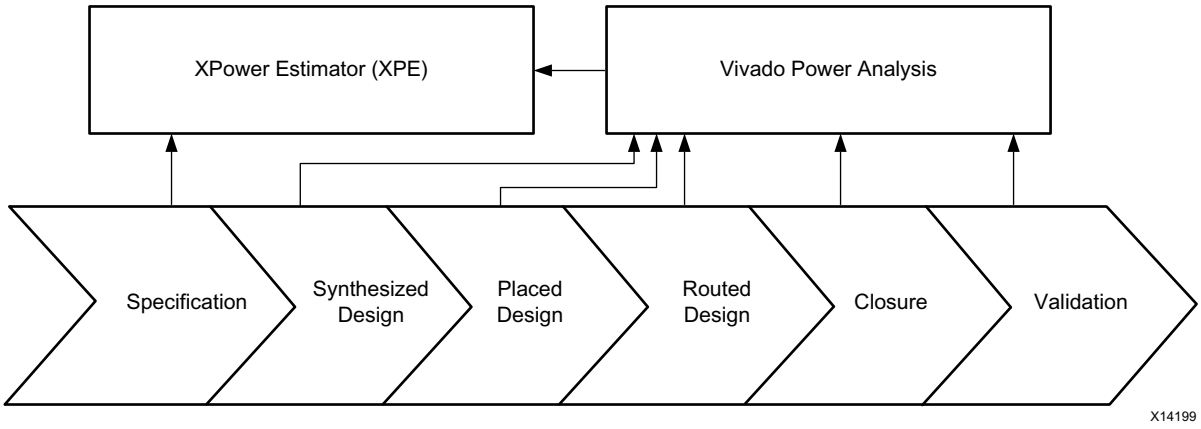

图 2-6 : FPGA 设计过程中的 Vivado 功耗估计与分析工具

有些工具是独立的，而其他工具则集成到实现过程中，以便使用设计过程每个阶段的信息所有工具都可以来回交换信息以实现高效分析。

系统已经实现并在实验室进行测量之后，才可以确定最终功耗。测量方法是手动探测开发板的电源线，或者提供一种机制，以便从外部可编程稳压器读取电压和电流。早期设计阶段准确计算功耗可以减少后期问题。

Xilinx Power Estimator (XPE)

赛灵思功耗估计器 (XPE) 电子数据表是一个通常在项目概念阶段使用的功耗估算工具。XPE 可协助进行架构评估和器件选择，并有助于为应用电路选择合适的电源和热管理组件。 Zynq-7000 AP SoC 的 XPE 接口如图 2-7 所示。设计人员可

使用工具来指定设计资源的使用、活动率、I/O 负载、CPU 时钟频率以及很多其他设计参数。XPE 将参数与器件模型进行组合，以计算估计的功耗分配。

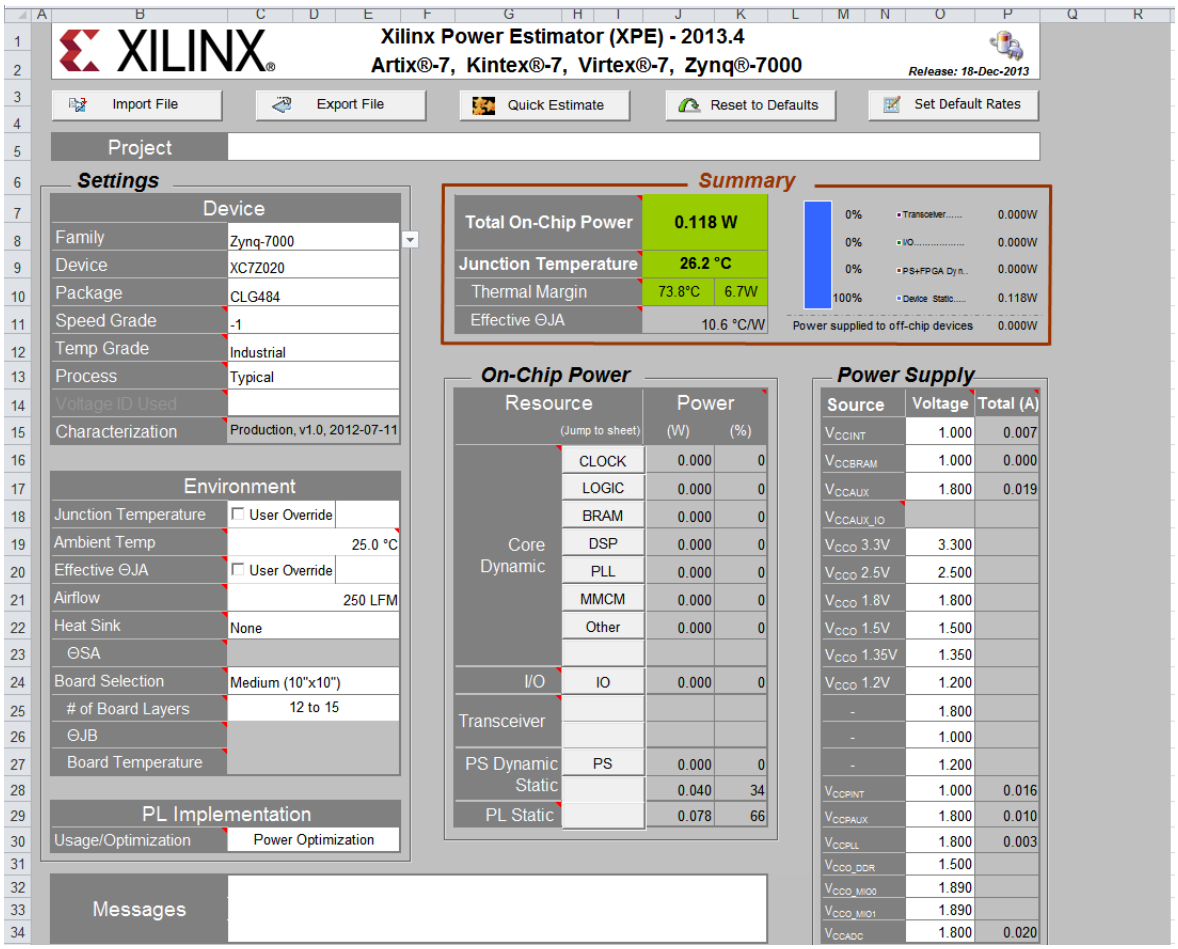

图 2-7 : 面向 Zynq-7000 AP SoC 的赛灵思功耗估计器

XPE 还普遍用于设计周期的后期，在实现和功耗收敛过程中使用，例如，对工程变更次序的功耗影响进行评估。如果是由多个团队实现的大规模设计，项目经理可以使用 XPE 导入每个团队模块的使用率和活动，然后监测总功耗并重新分配功耗预算，以确保满足约束条件。

系统级功耗分析

系统已经实现并在实验室进行测量之后，才可以确定最终的系统功耗。测量方法是手动探测开发板的电源线，或者提供一种机制，以便读取稳压器的电压和电流。

Zynq-7000 AP SoC 低功耗技术 第1部分——安装与运行功耗演示技术的技巧维基页 [参照 61] 提供了一个参考设计,用以演示在 Zynq-7000 AP SoC 的 PS 和 PL 执行不同应用应用实例时的系统级功耗。当对设计执行系统级功耗测量时，请参阅这篇技术文章。

Vivado 功耗分析

Vivado 功耗分析工具用于在设计阶段对经过布局和布线的设计进行功耗分析。它可提供全面的 GUI，有助于对指定工作条件下的功耗和热性能信息进行详细分析。图 2-8 所示为来自 Vivado 功耗分析的示例报告。

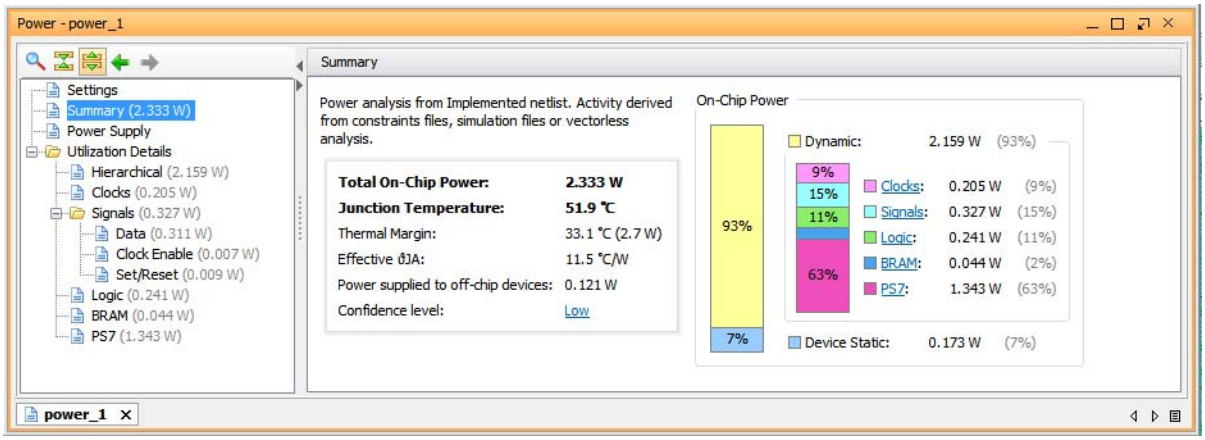

图 2-8 : Vivado 功耗分析示例报告

该工具可提供两个不同的功耗视图：

- 设计中各类模块的功耗，包括时钟树、逻辑、信号、 I/O，以及 PS IP，例如块状 RAM 和 DSP 块状。
- 整个设计层级的功耗。

设计人员可在两个视图之间切换，以执行详细的功耗分析。视图可提供高效方法以找出功耗最大的模块或设计部分，从而识别出哪里需要重点进行功耗优化。

可将值变化导出 (VCD) 和开关行为互换格式 (SAIF) 文件中的开关活动信息输入 Vivado 功耗分析工具中，以实现更准确的功耗估计。 VCD 文件包含头信息、变量定义以及每个仿真步骤的值变化细节。 SAIF 文件包含信号切换数量以及用于指定信号在 0、 1、 X 或 Z 电平上时间长度的时间属性。

Vivado 中的 PL 功耗优化

尽管设计人员可以按照之前的描述使用时钟门控对功耗进行优化，但该操作很少能完成。这是因为设计中包含其他来源的知识产权，或是因为如此精细的时钟门控涉及较大的工作量。 Vivado 工具可以自动进行功耗优化，以使降耗效果最大化，同时将工作量降到最低。

Vivado 设计工具提供多种功耗优化功能，帮助用户将动态功耗降低多达 30%。

Vivado 对整个 PL 设计 （包括原有和第三方 IP 模块）进行分析，以找出潜在的功耗节省机会。它检查给定时钟周期内对结果没有贡献的源寄存器的输出逻辑，然后创建精细的时钟门控和逻辑门控信号，以消除不必要的开关活动。

还对专用的块状 RAM 应用进行功耗优化。大部分功耗节省都是在无数据写入时以及输出未被使用时，通过禁用专用块状 RAM 的使能端来实现的。

如需详细了解如何使用功耗优化选项以及如何从设计中提取功耗信息，请参阅 《Vivado Design Suite 用户指南：功耗分析和优化》(UG907) [参照 10]。

纠正超预算的 PL 功耗

设计周期后期要面对压力，尽快将系统推向市场，此时，大多数系统参数已经设定好，例如电路板环境和冷却解决方案。尽管以上操作对能做的工程修改有限制作用，但仍有可能在 PL 中进一步降低功耗。以下流程有助于将精力重点集中在 PL 中最有可能降低功耗的区域。

1. **确定哪项功耗预算超标**

   Vivado GUI 用户可以查看 Vivado 功耗分析报告中的概括视图；命令行用户可以使用功耗报告文件中的总结部分。芯片和供电表格可提供功耗分配的高级视图。使用概括视图来确定超出预算的功耗类型和数量。

2. **识别重点区域**

   查看 Vivado 功耗分析报告或赛灵思功耗估计器中的各种详细视图。在资源利用、设计层级以及时钟域不同的情况下，对环境参数和功耗分配进行分析。当找到功耗比较高的设计区域时，所呈现的信息有助于确定造成高功耗的潜在因素。

3. **实验**

   制定了功耗优化重点区域的清单后，按照从易到难为清单排序，并决定接下来首先试行哪一项优化。功耗工具允许进行假设分析，以便快速进行设计变更，而且无需对代码或约束进行编辑，也无需重新运行实现工具即可得到功耗估计结果。

   本文档中列出了可以使用的功耗优化技术，如需了解更多信息，请参阅 《Vivado Design Suite 用户指南：功耗分析和优化》(UG907) [参照 10]。

利用 Vivado 功耗优化器功能进行实验

为了能最大程度节省功耗，当运行 Vivado 工具中的功耗优化器时，应对整个设计运行功耗优化，不要排除设计的任何部分。如果启动功耗优化后没有实现功耗节省，应该更仔细地检查三个方面：

- 全局设置和复位信号
- 块状 RAM 使能信号生成
- 寄存器时钟门控

如果功耗优化为以上任一方面生成的使能信号比较少，说明需要对编码方式或者为设计而设置的选项和属性进行检查。

利用 Vivado 功耗分析功能进行实验

重新运行分析之前可在 Vivado Report Power 对话框中做出调整，以检查功耗影响。如需详细了解如何使用 Vivado 功耗分析器工具中的不同选项来产生最佳结果，请参阅 《Vivado Vivado Design Suite 用户指南：功耗分析和优化》(UG907) [参照 10]。

利用赛灵思功耗估计器 (XPE) 进行实验

由多个来源开发的模块产生的 Vivado 功耗分析结果都可以导入 XPE。这样，一旦单独的 IP 模块在器件中实现，就可检查总体功耗。无需改变编码或网表，即可可以评估设计变更对功耗的影响。用 XPE 得到的估计结果不如用 Vivado 功耗分析工具得到的结果准确，因为 XPE 中无法在单独逻辑元件或信号级进行调整。

结论

在编码之前理解和实现功耗敏感型设计技术是降低系统功耗的最主要方法。在合适设计周期阶段使用不同赛灵思工具还有助于满足功耗规格，并为电路板设计人员提供选择电源数量、类型和尺寸时所需的信息。

时钟和复位

外部和内部时钟资源可作为 Zynq-7000 AP SoC 中各种 IP 模块的源时钟来使用。以下是对这些资源的介绍。本部分还对各种硬件、软件以及调试复位进行介绍，尤其是它们的使用时机和使用方法。

外部时钟

PS_CLK

在 PS 侧，通常使用 30–60 MHz 范围的定频振荡器来提供处理器时钟 PS_CLK。该时钟必须是单端 LVCMOS 信号，使用的电压水平与 MIO 单元 0 的 I/O 电压相同。其他 PS 内部时钟都从这个时钟生成，并基于三个 PLL：ARM、DRM 和 IO PLL。

赛灵思评估板上使用的默认 PS_CLK 频率为 33.3 MHz。可使用其他时钟频率，但以下项目取决于 PS 时钟频率，必须进行相应调整：

- LogiCORE IP 处理系统 7 配置 wizard 可计算每个 PLL 的衍生时钟分频器和乘法器，以及 I/O 外设时钟，例如基于所选 PS_CLK 频率的 SPI 或 UART。随后，第一阶段启动加载程序 (FSBL) 将在 PS 初始化过程中使用这些值。
- U-Boot 板配置包含文件。
- Linux 设计专用器件库。

其他

在 PL 侧，单端或差分定频振荡器可作为附加时钟资源，以实现更大灵活性。它们应连接到多区域时钟功能 (MRCC) 输入引脚，并符合相应 PL bank 的 I/O 标准和电压。PL 输入时钟抖动比 PS 衍生时钟的抖动小得多。对于某些用途（例如在视频系统中生成与视频分辨率相关的精确的像素时钟） ,应使用外部低抖动可编程时钟综合器。

某些电路板外设需要用外部晶体来操作 PHY （例如，以太网 PHY 需要25 MHz 晶体）。此外，有些外设 I/O 接口提供用来与相应 PHY 进行通信的输入和/或输出时钟 （例如以太网 RX 和 TX 时钟）。

内部时钟

PS

PS 时钟子系统生成的所有时钟都衍生自三个可编程 PLL：CPU、 DDR 和 I/O。每个 PLL 都与 CPU、 DDR 和外设子系统中的时钟松散关联。在正常运行期间， PLL 启动，并由 PS_CLK 时钟
引脚驱动。在旁路模式下， PS_CLK 引脚上的时钟信号为各种时钟生成器 （而非 PLL）提供时钟源。在启动过程之后，旁路模式和每个 PLL 的输出频率可由软件单独控制。

CPU 时钟域由四个独立时钟组成：CPU_6x4x、 CPU_3x2x、 CPU_2x，以及 CPU_1x。这四个时钟根据它们的频率命名，由两个比率中的一个进行关联：6:3:2:1 或 4:2:2:1 （分别缩写成 6:2:1 和 4:2:1）。所有 CPU 时钟都相互同步。有两个独立的 DDR 时钟域：DDR_3x 和 DDR_2x。这些时钟相互之间异步并与 CPU 时钟异步。大多数 I/O 外设时钟都有专用分频器。每个外设时钟都与其他时钟完全异步。

PS - PL 接口

PL AXI 通道 (AXI_HP、AXI_ACP 和 AXI_GP）具有 PS 与 PL 之间的异步接口。同步区，即时钟域交汇的地方，位于 PS 内部。因此， PL 向 PS 提供接口时钟。上述每个接口都可使用 PL 中的唯一时钟。

PS 向 PL 提供四个频率可编程的架构时钟 (FCLK [3:0])，并沿着 PS-PL 边界物理展开。这些时钟可通过设置时钟源（ARM、 DDR 或 I/O PLL）和时钟输出频率实现单独控制。四个 FCLK 时钟之间不存在确定的相位关系，即使共享相同时钟源时也是如此。当多个 FCLK 区域之间进行接口联系时，务必使用适当的设计约束。 FCLK 时钟处于禁用状态，直到 PS - PL 级电平移位器启动为止。

建议： 有一种良好实践就是，将单个 FCLK 布线引入 PL 内部例化的时钟向导 IP 核，以生成一个以上的相位对齐输出时钟。

以下是使用 FCLK 的利与弊：

- 在以下情况下， FCLK 是首选 PL 时钟：
  - 处理器控制 PL 时钟频率。
  - 不具备板上时钟生成器。
- 以下情况下， FCLK 不是首选的 PL 时钟：
  - PL 时钟频率在 FCLK 所支持的频率范围之外。
  - PL 使用由 FPGA 引脚提供的时钟这在使用输入时钟来采样接收数据的源同步协议中很普遍。
  - 需要低时钟抖动。
  - 有些需要特定时钟的 IP 模块不能使用 FCLK：
    - 储存器接口生成器 (MIG) 需要差分时钟。因此，除非在由于抖动而导致频率降低的情况下， FCLK 都无法用于 MIG。
    - GT 应该使用来自电路板的差分时钟作为参考时钟。

如需了解关于 PS 时钟系统的更多信息，请参阅《Zynq-7000 All Programmable SoC 技术参考手册》(UG585) [参照 4] 中的“时钟”章节。

PL

PL 提供 FPGA 器件中常见的时钟原语，例如全局或局部时钟缓冲器 (BUFG, BUFR)、锁相环 (PLL) 或混合模式时钟管理器 (MMCM)。为方便起见，时钟向导 IP 核利用多达八个可配置输出时钟实现一个围绕 MMCM/PLL 原语的封装。用户可使用动态重配置端口 (DRP) 或 AXI-Lite 接口可选地启动时钟频率的动态重配置。

提示： 最简单的情况是，完整 Zynq-7000 系统可用基于 PS_CLK 的单输入时钟来构建，而且，所有 PL 时钟都从所提供的 FCLK 和 FPGA 时钟资源生成。

PS 上电复位

PS 加电复位 (PS_POR_B) 是一个低电平有效信号，用来将 PS 保持在复位状态，直到所有 PS 电源都稳定并达到要求的电压水平为止。该信号应从电源 power-good 信号或电压监督芯片中生成。在 PS_POR_B 被释放时，系统时钟 (PS_CLK) 必须已经连续 2,000 个时钟周期保持稳定。 PS_POR_B 应被拉高至 VCCO_MIO0。当激活 PS_POR_B 时，脉冲长度必须长于 100 µs。

加电复位属于芯片主机复位。它将器件中每个可以复位的寄存器进行复位，复位所有 PS RAM （包括 OCM、 Fifo、缓冲器等），并启动 BootROM 执行，清除 PL 配置。当 PS_POR_B 保持在低电平时，所有 PS I/O 都保持在三态。

PS 系统复位

PS 系统复位 (PS_SRST_B) 是一个低电平有效的信号，主要是在调试时使用。 PS_SRST_B 必须是高电平，才能开始启动进程。如果 PS_SRST_B 未使用，可将其拉高至 VCCO_MIO1。当激活 PS_SRST_B 时，脉冲长度必须长于 1 µs。

PS 系统复位 (PS_SRST_B) 可在不影响调试环境的前提下对所有功能逻辑进行复位。这与加电复位 (PS_POR_B) 不同，因为后者会清除调试配置。 PS_SRST_B 清除所有 PS RAM，启动 BootROM 执行，并清除 PL 配置。与 PS_POR_B 不同的是，它不会重新对启动模式捆绑 (boot-mode strapping) 引脚进行采样。启动模式与前面的加电复位保持相同，上一个启动的安全等级得到保留。

PS_SRST_B 信号必须在 PS_POR_B 信号被激活之前解除。如果 PS_POR_B 复位所导致的 BootROM 执行被 PS_SRST_B 复位信号的激活中断，系统将会锁定。如果 PS_SRST_B 和 PS_POR_B 都被采用，那么 PS_POR_B 必须是最后解除的信号。如需了解更多信息，请参阅赛灵思解答记录 52847 [参照 68]。

如果您需要不清空 PL 的 PS 复位解决方案，请联系赛灵思技术支持。

系统软件复位

系统软件复位也称为 SLCR 软复位，通过写入 PSS_RST_CTRL[SOFT_RST] 来激活，并且具有与激活 PS_SRTS_B 引脚相同的效果。所有 PS RAM 都被清空，而且 PL 也被复位。

Watchdog 定时器复位

watchdog 定时器复位有两个来源：系统 Watchdog，即 SWDT；以及两个 ARM Watchdog 定时器，即 AWDT0 和 AWDT1。 SWDT 总是对整个系统进行复位，而任意 AWDT 都可用于复位相关的 ARM 内核（与 CPU 复位效果相同）或整个系统 （与系统软件复位效果相同）。

CPU 复位

有两个 CPU 复位，各针对一个通过写入 A9_CPU_RST_CTRL[A9_RSTx] 来激活的 ARM 内核。对单个处理器的 CPU 复位必须从另一个 CPU 通过 JTAG 或 PL 来应用。

调试复位

有两个调试复位，即调试系统复位和调试复位，均源自 ARM DAP 并由 JTAG 控制。调试系统复位具有与系统软件复位相同的效果，而调试复位只复位调试逻辑。

外设复位

单独的外设复位可在软件控制下使用 SLCR 中的可编程位进行激活。但是，不建议在外设模块级激活复位。在未完成所有进行中或挂起的事务处理时就对外设进行复位，这样会导致系统停止，因为 AXI 事务处理不支持超时机制。将复位激活到外设时，所有挂起和进行中的事务处理必须都已完成，并且在复位之前不能再发布新的事务处理。

PL 复位

PS 向 PL 提供四个可编程复位信号 (FCLK_RESET [3:0])。复位可单独编程，且独立于 PL 时钟。在进行 POR 或系统范围的复位之后，复位信号直到 BootROM 执行结束且 PS 至 PL 电平移位器启动后才能解除。

FCLK_RESET 与编号相同的 FCLK 松散关联。也就是说，FCLK 需要进行切换，以便于 FCLK_RESET 从 PS 传播出来。该复位是对 PL 的异步复位，如果需要，必须将 PL 内的复位进行同步。

建议： 为了同步 FCLK_RESET，需要将信号连接到 proc_sys_reset IP 核的外部复位输入端口。这样，就可以将连接其他 PL IP 核的复位输出信号同步到最慢的时钟。

如果 FCLK_RESET 信号驱动 PL 中基于 AXI 的 IP 复位，并且在上述初始解除后再激活，随后，如果在未完成进出处理模块的所有进行中或挂起的事务处理的情况下就复位 IP，会导致系统停止。系统必须让所有 AXI 主机空闲，并在激活复位之前结束所有 AXI 事物处理。

中断

通用中断控制器

由于 Zynq-7000 AP SoC 有很多中断源，因此 MPCore 多核处理器包含一个通用中断控制器 (GIC) 架构的实现，用以对这些中断源进行汇集、优先排序和仲裁。 GIC 将中断映射到具体的处理器 nIRQ 和 nFIQ 线路。多个同步的中断被顺序送到一条或多条处理器中断线路。

从概念上讲， GIC 包含一个中断分配时钟 （分配器）和两个处理器中断模块。每个模块包含一组寄存器。处理器中断模块中的寄存器只能由它们通过私有总线连接的处理器进行访问，系统中的其他 CPU 或其他 AXI 主机无法对其进行访问。任意处理器均可对分配器寄存器进行访问，另外，有些寄存器还具备安全访问和非安全访问的能力。

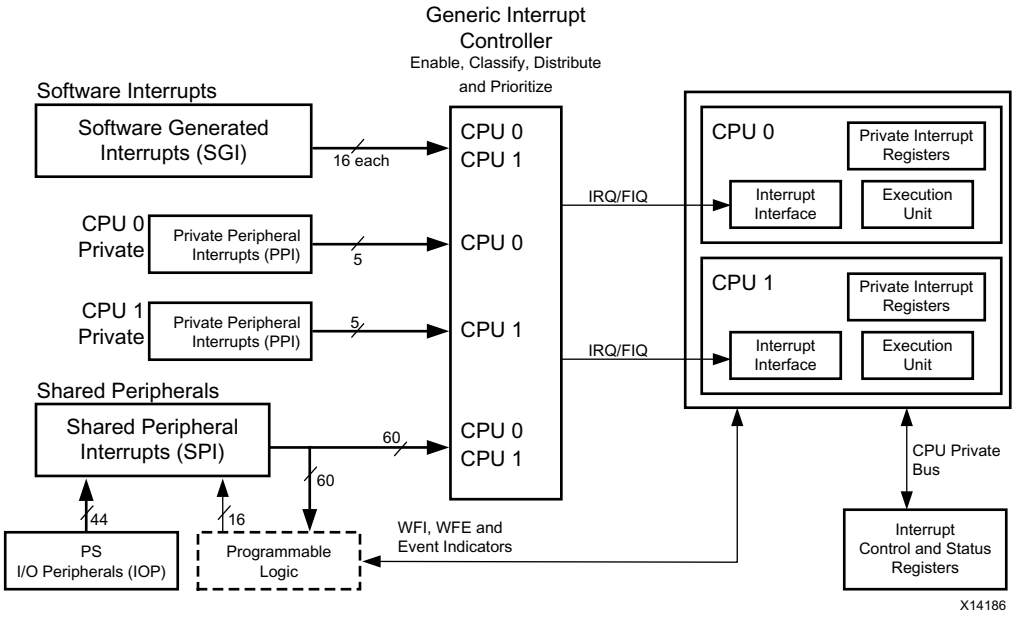

图 2-9 : 系统级模块设计

发送到 GIC 的中断可以是未托管的 （原有的）或托管的中断。原有的中断不受 GIC 控制，因此中断处理程序不应与 GIC 寄存器交互。托管理的中断由 GIC 控制，中断处理程序必须与 GIC 寄存器交互。

分配器按如下方式控制管理的中断源：

- 将中断定义为边缘敏感型或电平敏感型，并受硬件配置制约。
- 为中断分配 5 位的优先级，具有可编程二进制小数点。
- 为中断分配 TrustZone 技术的安全状态。标记为安全的中断称为 Group 0 中断，非安全中断标记为 Group 1 中断。
- 将布线共享外设中断发送到一个或两个处理器。

  注释： 私有外设中断 (PPI) 被每个 CPU 所专用，无法通过除 GIC 以外的途径布线到其他位置。
- 将软件生成的中断布线到一个或两个处理器。
- 保存和恢复每个中断的挂起状态。这在低功耗的应用场合下是有用的。

中断架构

这部分介绍 PS 和 PL 中断源、 PS 中断层级以及 Cortex-A9 处理器考虑因素。

PS 中断源

每个 Cortex-A9 处理器均可由以下中断源中断：

- 16 个软件生成的中断 (SGI) 可用，以便软件对任意处理器进行中断。
- 5 个私有外设中断 (PPI) 可用。其中两个中断来自 PL （FIQ 和 IRQ），另外三个分别来自全局定时器、私有定时器和 AWDT。
- 60 个共享外设中断可用。其中包含 44 个 PS I/O 外设中断和 16 个 PL 中断。
- 有 4 个 PL 中断可绕过 GIC，直接中断处理器，从而缩短中断时延。
- 尽管 WFE 指令不是中断，但 CPU 可以用它来故意拖延和等待专用输入线路上来自其他 CPU 或 PL 的事件。

PL 中断源

如果有必要， PL 中每个响应中断的器件都必须对自己的中断控制器进行安排。可将以下中断从 PS 送到 PL：

- 来自 PS 的 29 个共享外设中断。这些对应于很多 （但不是所有） PS 外设。
- 处理器也可向 PL 发出软件生成的中断，方法是使用 PL 中的 AXI_GPIO 输出通道 EMIO GPIO，或者利用 SEV 指令激活一条每处理器硬件事件线路。

中断 ID

Zynq-7000 AP SoC 支持 IRQ ID #0 至 #95，如下所示：

- IRQ ID #0–#15：这些编号被分配给软件生成的中断，使软件能够中断处理器 （包括管理该软件的处理器）。
- IRQ ID #16–#31：这些编号被分配给私有外设中断。 IRQ ID #16–#26 未使用。 Zynq-7000 AP SoC 实现 IRQ ID #27–#31，如下所示：
  a. IRQ ID #27 被处理器全局定时器使用。
  b. IRQ ID #28 被托管的 FIQ 使用，并且是未托管的输入。
  c. IRQ ID #29 被处理器私有定时器使用。
  d. IRQ ID #30 被处理器 AWDT 使用。
  e. IRQ ID #31 被托管的 IRQ 使用，并且是未托管的输入。
- IRQ ID #32–#95：这些被分配给共享外设中断。 IRQ ID #36 未使用。 Zynq-7000 AP SoC 实现 IRQ ID #32–#35 和 #37–#95。 31 个中断 (IRQ ID #32–#62) 都可针对用户更改进行解锁或锁定。

独特功能

中断可源自 PS 或 PL，另外， PS 处理器或 PL 架构中的专用硬件 （例如 MicroBlaze™）可响应中断。这样可实现 Zynq-7000 AP SoC 的独特功能，如以下部分所述。

架构内的中断处理

创建自定义硬件以使 PL 分担中断处理任务。这样，中断时延就会非常低并且具有确定性 （时钟周期精度）。此外，定制硬件可以执行在处理器上耗时较长的计算任务，与处理器并列执行。完成此操作后，可能需要进行数据整合，而且要使用 ACP 为 PL 主机提供缓存一致性。

定制硬件还可用来过滤中断，降低 PS 中断频率和处理器负载。方法是将多个低级中断转换为数量较少的高级中断。

处理器作为架构的扩展

或者，将 Zynq-7000 AP SoC 中的处理器作为架构硬件的扩展，在软件中执行那些在硬件中难以实现的任务。 PL 可以向 PS 发送中断，以初始化一个任务；也可以从 PS 向 PL 发送中断，以说明任务完成。该方法可与 PS DMA330 控制器配合使用，以提供附加的执行线程。

针对处理程序使用 OCM

从 DDR 储存器取出中断处理程序所需的时间可能会超出预期。 Zynq-7000 AP SoC 有一个较大的 L2 缓存，但当中断发生时可能不会对中断处理程序进行缓存。这种情况下，将中断处理程序放在片上储存器 (OCM) 中会更有利。这样，取出中断处理程序所需时间的变化就被限定在 L1 缓存缺失，从而减少抖动

处理器亲和度

由于有两个处理器，这样轮询责任和中断处理程序就可在二者之间划分，从而减少时延并改善响应时间。该操作在对称或非对称的多处理系统中均可完成。

使用 FIQ 和 IRQ

Zynq-7000 AP SoC 中的每个 Cortex-A9 都具有由 ARM e 通用中断控制器 (GIC) 驱动的两条中断线路 nFIQ 和 nIRQ。前缀“n”代表它们是低电平有效中断。 FIQ 的时延低于 IRQ，通常用于高优先级中断。 IRQ 用于不需要低延迟响应的中断。 FIQ 的优先级总是高于 IRQ，优先级较低的 IRQ 处理程序只有在 FIQ 处理程序结束后才能恢复执行。

TrustZone

Zynq-7000 AP SoC 使用 ARM TrustZone 技术，允许将系统组件定义为安全或非安全。例如，在不同时间内，每个处理器可工作在安全和非安全模式下。同样，中断也可分成安全和非安全。这样可将安全组件与非安全组件进行隔离。这种情况下，高优先级的 FIQ 用于安全中断，低优先级的 IRQ 用于非安全中断。还可以锁定很多安全设置以防止变更。

系统设计考虑事项

系统设计应考虑各种中断实现选项的性能影响。

- 应采用硬件或 OS 对时延和相关抖动进行特性分析和管理。如果 OS 将对掩膜中断一段时间，长度不可预知，或者由于更高优先级的中断正在执行中，那么将会发生抖动。
- 对于中断处理延迟的情况，应为外设和相关软件定义策略。策略包括：丢弃数据，限制或中止远程数据源，以及允许硬件以低性能运行。
- 假定平均和最坏情况，处理器利用率对于外设服务和其他任务也是足够的。
- 中断处理程序的储存器位置可影响指令取出性能。距离处理器较近的储存器具有较低时延。保持中断服务程序一致，以使其在缓存行边界启动，从而缩短时延。
- 当已中断的外设对可能缓存到处理器的数据进行操作时，处理器必须通过作废或清空缓存或者利用硬件一致性机制 （例如 ACP）来保持缓存一致性在 Zynq-7000 AP SoC 上，使用 ACP 端口的 PL 外设不受此限制。
- 在可实现非对称多处理 (AMP) 的系统中，必须建立中断 （及其他系统资源）的所有权策略。

嵌入式器件安全性

现代化嵌入式器件中的安全性定义取决于用途。对于移动电话而言，安全是指防止对个人数据 （例如，银行账号和密码）进行非授权访问。对于数据中心交换机来说，安全是指防止黑客恶意登陆和干扰网络服务。窃取和反编译存储在闪存中的二进制软件逻辑或者物理拆解器件以进行反相工程，这些也属于嵌入式器件的安全挑战。

针对嵌入式器件的潜在威胁包括：

- 嵌入式器件中的数据隐私
- 克隆嵌入式器件
- 拒绝服务
- 植入恶意软件以更改嵌入式器件行为
- 内部人员将秘钥提供给竞争对手

以上实例描述的是来自黑客和竞争对手的威胁。其他系统安全威胁可来自编写不佳的程序。这样的程序会意外导致其他程序以及存储在嵌入式器件闪存中的用户数据出错，或者独占处理器或外设带宽，从而干扰正常的嵌入式器件操作。

Zynq-7000 AP SoC 用于提高嵌入式系统的安全性。本部分重点介绍可用来提供不同安全水平的 Zynq-7000 AP SoC 安全特性，例如：

- 启动镜像和码流加密与身份验证 （安全启动）。
- 将系统分成独立的安全区域 (TrustZone)。
- 在 Zynq-7000 AP SoC 上部署异步多处理 (AMP)。
- 在 Zynq-7000 AP SoC 上部署 Linux。

安全启动

使用 Zynq-7000 AP SoC 的嵌入式系统通常要避免器件的使用方式与设计初衷不同。通常，采用 Zynq-7000 AP SoC 的嵌入式系统至少有一个可编程组件 （共两个）：软件 （PS 镜像）和 PL 码流。设计系统时必须确保在将系统部署到现场之后，不会使用不可信和遭破坏的可编程组件进行系统启动。另外，必须包含可避免可编程组件遭偷窃的安全措施，以防恶意竞争者窃取 PS 镜像和 PL 码流。身份验证用来防止用未授权的可编程组件来启动；加密用来防止可编程组件被盗。

Zynq-7000 AP SoC 提供多种硬件和软件(BootROM)功能，经过配置后可确保使用受信的可编程组件（软件和 PL 码流）来启动 Zynq-7000 AP SoC。安全启动模式仅限于将 NOR、NAND、SDIO 或 Quad-SPI 闪存作为外部启动设备。不允许从 JTAG 或任何其他外部接口进行安全启动。

PS 软件启动流程

图 2-10中给出了 PS 的启动流程。

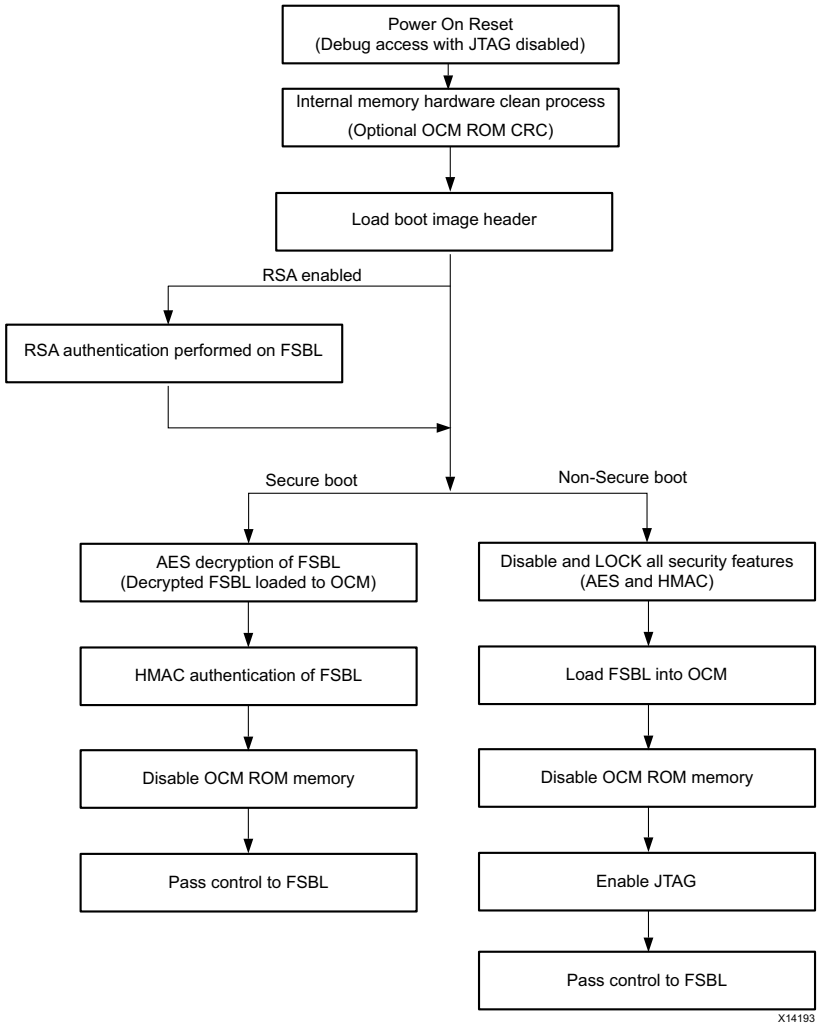

图 2-10 : PS 启动流程

加电 BootROM CRC 校验

加电后， BootROM 是第一个执行的软件组件。加载 FSBL 之前，可对 BootROM 进行 128KB CRC 校验。CRC 校验通过 eFuse 设置进行控制。如需详细了解如何编程 Zynq-7000 AP SoC 中的 eFuses，请参阅《Zynq-7000 All Programmable SoC 安全启动》(XAPP1175) [参照 40] 中的“安全秘钥驱动”部分。

在 BootROM 执行期间，按模式引脚的配置，从外部存储启动设备 （SDIO、 QSPI 闪存、 NAND 闪存或 NOR 闪存）读取启动报头。头文件包含的信息能表明设备是否可安全启动。如需了解关于模式引脚的更多信息， 请参阅《Zynq-7000 All Programmable SoC 技术参考手册》(UG585) [参照 4] 中的“启动与配置”章节。

FSBL 的 RSA 身份验证

执行期间， BootROM 可利用 RSA 公共秘钥身份验证技术在解密之前验证安全 FSBL，或者在执行之前验证非安全 FSBL。启动该功能的方法是熔断 PS eFuse 阵列中的 RSA 身份验证使能保险丝。对 FSBL 进行身份验证可以防止恶意或

毁坏的 FSBL 使用未授权的可编程组件从未授权的启动源启动，危害 Zynq-7000 AP SoC。如需了解关于 RSA 身份验证的更多信息，请参阅 《Zynq-7000 All Programmable SoC 安全启动》(XAPP1175) [参照 40]。

安全启动图像

单文件启动镜像 (BOOT.bin) 的可编程组件包括：

- PS 图像组件
  - 初始化头文件能选择性地向寄存器写入值。例如，初始化头文件可用来在 BootROM 复制和执行 FSBL 之前提高 CPU 时钟速度或启动设备速度。
  - FSBL。
  - 可选的二次启动加载程序，例如 U-Boot 或裸金属软件。
  - 可选的数据镜像和多个 ELF。
  - 可选的 Linux uImage。
- PL 码流

如果需要，可以将启动镜像做成安全的。使用哈希信息验证码 (HMAC) 对安全启动镜像中的 PS 镜像和 PL 码流进行身份验证，使用高级加密标准 (AES) 进行加密。如需了解创建安全启动镜像方面的更多信息，请参阅 《Zynq-7000 All Programmable SoC 安全启动》(XAPP1175) [参照 40] 和 《Zynq-7000 All Programmable SoC 安全启动入门指南》(UG1025) [参照 20]。

AES & HMAC 身份验证引擎

Zynq-7000 AP SoC PL 包含 AES 解密和 HMAC 身份验证引擎。因此， PL 必须在安全启动过程中上电，即使安全启动镜像没有 PL 码流。BootROM 在从启动设备读取加密镜像之前检查 PL 是否上电，因此嵌入式系统必须确保在加密之前 PL 上电。由于 PL 通过不同的电轨供电，因此嵌入式系统必须确保电轨与功耗调节器之间存在正确的连接。

Bootgen

使用名为 Bootgen 的赛灵思软件工具组装 BOOT.bin 的可编程组件。当选中安全启动选项时，Bootgen 还对可编程组件加密。请参阅 《Zynq-7000 All Programmable SoC 安全启动》(XAPP1175) 中的“创建安全启动图像”部分 [参照 40] 以及 《Zynq-7000 All Programmable SoC 开发人员指南》(UG821) 中的“使用 Bootgen”部分 [参照 7]。

生成秘钥

安全启动身份验证和解密过程需要使用加密秘钥。 Zynq-7000 AP SoC 使用的秘钥为：

- AES 256 位秘钥
- HMAC 秘钥
- RSA 主私有秘钥 (PSK)
- RSA 主公开秘钥 (PPK)
- RSA 辅私有秘钥 (SSK)
- RSA 辅公开秘钥 (SPK)

您可以为 AES 和 HMAC 引擎提供开发者秘钥，或者使用 Bootgen 生成以上秘钥。Bootgen 为 key0 和 HMAC 共同创建一个秘钥。如果需要为 key0 和 HMAC 创建唯一秘钥值，则应该运行 Bootgen 两次，取每次运行所得的秘钥，并在第三次运行时以用户秘钥集的方式提供。使用 Vivado 或安全秘钥驱动器将 AES 秘钥通过编程放到 PL 中的 eFuse 或 BBRAM 上。

在 Zynq AP SoC 中，主 RSA 秘钥用于验证辅秘钥。辅秘钥用于验证分区 （例如软件、数据和码流）。 OpenSSL 用于创建 RSA 主秘钥和辅秘钥。之所以使用 OpenSSL 是因为它是现成可用的；不过，您也可以使用其他方法生成秘钥。

OpenSSL 生成的 RSA 秘钥是私有/公开密钥对。公开秘钥是私有秘钥的子集。为了安全，一定要保护私有秘钥。在 RSA 中，私有秘钥用于在制造现场为分区签名；同时，将哈希公开秘钥编程到 Zynq AP SoC 嵌入式设备中用以验证签名。

eFuse/BBRAM 实现安全性

就 RSA 身份验证而言，哈希 PPK 存储在 PS eFuse 阵列里。 PS eFuse 利用安全秘钥驱动器进行编程。对于 AES 解密而言，秘钥存储在 PL 的 eFuse 或 BBRAM 中。eFuse Secure Boot 的 PL eFuse 控制位、BBRAM Key Disable 和 JTAG Chain Disable 通过 Vivado 或安全秘钥驱动器进行编程。 eFuse 属于一次性可编程 (OTP)。对 eFuse 进行编程之后需要加电复位 (POR)。

如需更详细了解安全秘钥驱动器以及本部分介绍的多种其他安全功能，请参阅 《Zynq-7000 All Programmable SoC 安全启动》(XAPP1175) [参照 40]。

将系统分为独立的安全区域 (Trust Zone)

安全启动通过加密和身份验证来防止使用未授权的软件进行启动，并可防止可编程 PS 镜像和 PL 码流被窃取。然而，系统设计人员可能希望提供附加保护层，例如，只有当受信任软件在系统上执行时才允许对特定系统组件进行访问。这样有助于防止动态加载的第三方应用访问私有数据或独占系统资源，进而降低系统性能。例如，系统设计人员可能不希望第三方软件对用于存储私人数据 （例如银行账户和密码）的系统闪存进行访问。

ARM TrustZone 技术帮助系统设计人员将系统分成逻辑分区，只允许受信任软件对硬件层的安全组件进行访问，从而确保运行时间安全。如需了解更多信息，请参阅：

- ARM 安全技术：使用 TrustZone® 技术构建安全系统 [参照 74]
- 《编程赛灵思 Zynq-7000 All Programmable SoC 上的 ARM TrustZone 架构用户指南》(UG1019) [参照 19]

通过异步多处理 (AMP) 提供安全性

异步多处理 (AMP) 可以用在 Zynq-7000 AP SoC 中，通过限制软件的访问区域来确保系统安全性。例如，使用一个 ARM Cortex-A9 处理器执行不受信的第三方应用程序；其他 ARM Cortex-A9 处理器用于执行受信的软件。为运行受信软件的 ARM Cortex-A9 处理器上的操作系统分配处理关键任务的系统资源，例如外设。运行非受信软件的 ARM 内核上的操作系统不能感知安全外设。这种方法实现逻辑隔离，使得非受信软件无法有意或无意地降低系统性能。该方法能确保运行时安全。利用安全启动来防止未授权的可编程组件对 Zynq-7000 AP SoC 的恶意启动。

Linux 部署

由于 Linux 被植入了 Zynq-7000 AP SoC，因此它具备 Linux 安全特性。 Linux 安全功能包括：文件访问控制，即利用储存器管理单元防止储存器受到非法访问，例如受到程序的访问。

基于 Linux 的平台 （例如 Android）还可以利用
虚拟机防止第三方应用程序直接访问硬件资源，从而提供设备安全性。

有很多针对 Linux 的其他第三方专有安全扩展。

剖析 (Profiling) 和划分 (Partitioning)

剖析工具可帮助用户确定在硬件与软件之间对应用程序功能进行分区的方法，以实现最优性能。

软件剖析

剖析是一种程序分析形式，用来协助软件应用程序的优化。其可用于测量多种应用程序代码属性，包括：

- 存储器
- 函数调用执行时间
- 函数调用频率
- 指令用法

剖析可以静态 （不执行软件程序）也可以动态 （软件应用在物理或虚拟处理器上运行时执行）进行。

静态剖析一般通过对源代码进行分析，或者有时对目标代码进行分析来实现。

动态剖析通常是侵入式过程，即通过中断程序在处理器上的执行来收集信息。一些处理器中的追踪机制可用来对剖析数据进行非侵入式收集。

设计人员可利用剖析识别出以下因素所导致的代码执行瓶颈，这些因素包括：

- 低效率的代码
- 函数与某 PL 模块之间通信不佳
- 软件中的函数之间通信不佳
- 在软件中实现了更适合在硬件中实现的算法或程序

一旦确定，重写初始软件功能或者将其移动到 PL 中进行加速，从而对瓶颈进行优化。或者，将一部分功能留在软件中，同时将有问题的部分移动到硬件。

对大型程序进行分析时也可使用剖析，这种程序规模太大，无法通过读取源代码的方式来分析。剖析可帮助设计人员找出通过其他方式没有观察到的漏洞。

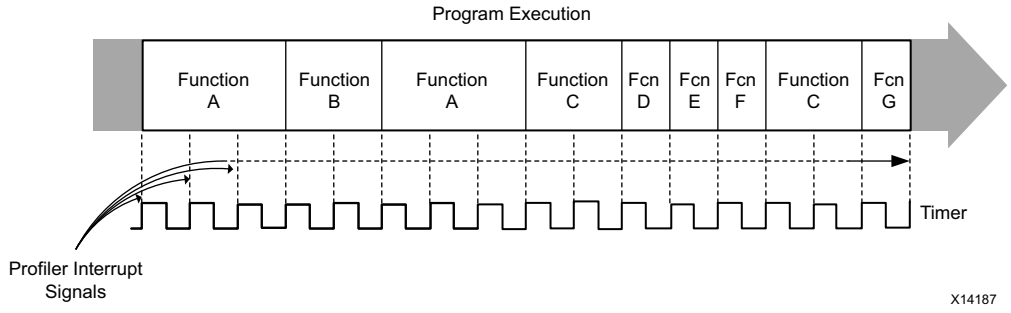

图 2-11 : 程序执行实例

各种功能的执行流程如图 2-11 中所示，并突出显示执行给定功能所需的时钟周期数量。通过对程序进行剖析，并以此确定执行每一单个功能所需的时钟周期数量，这样就能确定是否需要对该程序的某功能进行优化。研发过程中，软件工程师可能已经估计出功能在给定 PS 上执行所需的平均时间。这个估算可与剖析结果进行比较，并对较大差异进行调查研究。图 2-11 中的执行流程的剖析输出示例在图 2-12 中给出。

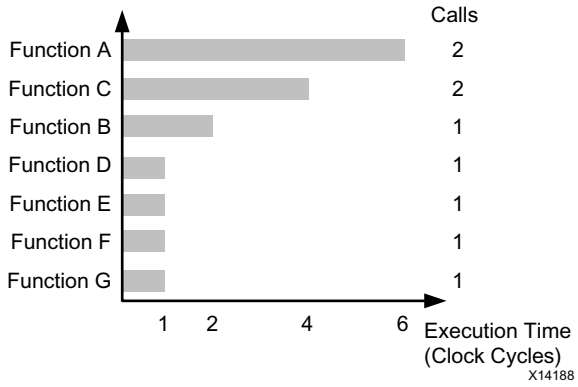

图 2-12 : 剖析输出示例

赛灵思 SDK 系统调试器

赛灵思 SDK 包含有助于找出代码中的瓶颈的剖析工具。瓶颈的产生原因是 PL 中执行的功能与处理器上执行的功能之间的交互。找出瓶颈后，可将整个功能迁移到 PL，优化处理器上的功能代码，或者将功能在处理器与 PL 之间进行拆分，以此来优化瓶颈。

SDK 支持层次剖析，有助于了解哪个调用功能和被调用的子功能对流程性能影响最大。

TCF 剖析

赛灵思 SDK 中的剖析可通过 TCF 剖析器来执行。 TCF 剖析器利用统计采样法定期检查系统，确定哪些代码正在执行，并更新适当的计数器。假设采样率足够，执行剖析的精度越高，所收集的剖析文件就越长。与使用中断的其他剖析器不同，如果被剖析代码关闭中断，该方法不会导致不准确。被剖析的程序不需要进行重新编译，相比之下，使用 gprof 时则需要。

性能监控

SDK 中的性能监测功能收集来自 PL 的 AXI 性能监测器 (APM) 事件计数模块数据、ARM 性能监测器单元 (PMU) 数据以及来自 Zynq-7000 AP SoC PS 的 L2 缓存数据。数据由 SDK 通过 JTAG 实时收集。每隔 10 毫秒对这些计数器的数值进行采样。这些值用来计算在专用视图中显示的指标。

SDK 中的“Performance”标签如图 2-13 所示。

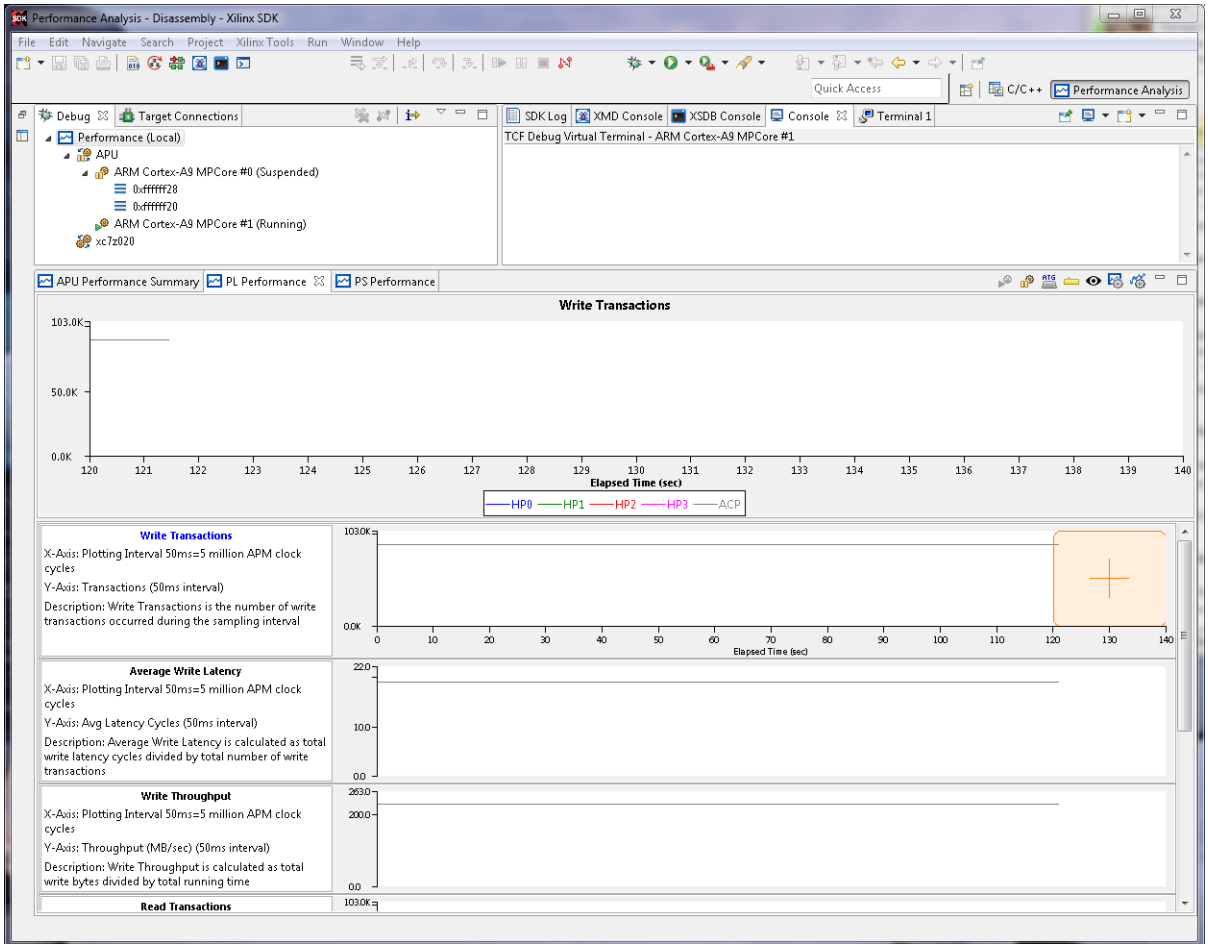

图 2-13 : “PL Performance”标签

针对每个 Cortex-A9 CPU，需监测以下 PMU 事件：

- 数据缓存填充
- 数据缓存访问
- 数据挂起
- 写挂起
- 指令重命名
- 分支错过

监测以下两个 L2C-PL310 （L2 缓存控制器）计数器：

- 缓存命中次数
- 缓存访问次数

当设计中使用 APM 时，可监测针对每个 HP 和 ACP 端口的以下 APM 计数器：

- 写字节数量
- 读字节数量

- 写事务处理数量
- 总的写延迟
- 读事务处理数量
- 总的读延迟

ARM Development Studio 5 (DS-5)

ARM Development Studio 5 (DS-5) 工具链是一套针对基于 ARM 处理器系统的完整软件开发工具。DS-5 覆盖基于 ARM 处理器的产品的所有研发阶段——从平台启动到应用程序剖析，同时还包括多种 ARM Linux 和 Android 具体特性。

Streamline 性能分析器是 ARM Development Studio 5 (DS-5) 工具链的组件。它是一款针对 Linux 和 Android 系统的系统级性能分析工具。它采用样本剖析、 Linux 内核跟踪以及软件注释技术。 Streamline 报告在七个窗格上提供多种性能相关信息，它们分别是：Timeline、 Call Paths、 Functions、 Code、 Call Graph、 Stack Analysis 和 Logs。

默认情况下，每隔 1ms 或 10ms 对程序计数器采样，以生成 Streamline 剖析报告。当启动基于事件的采样时， Streamline 在事件计数器达到所选的阀值时提取样本。这些样本对应于导致最后事件的指令，并使用基于事件的数据 （而非基于时间的数据）来填充剖析报告。例如，基于事件的采样可用来确定哪部分代码导致缓存缺失或分支错误预测。

实用应用

启动时间

应用程序的启动时间对一些系统来说是重要考虑因素。总启动时间包括 BootROM 、 FSBL、第2阶段启动加载程序 （例如 U-Boot）和操作系统的执行时间。如需关于启动时间的信息，请参阅赛灵思解答记录 55572 [参照 69]。

处理器负载

处理器负载是指实际执行的计算量占能够执行的总计算量的百分比。当处理器负载接近 100% 时，系统会开始失效或无响应。

Top 是一款用来实时查看处理器负载的 Linux 工具。它能显示系统中占用 CPU 最多的任务的列表，并提供用于操纵过程的交互接口。它可按照 CPU、 CPU 使用、储存器使用和运行时间来给任务分类。

系统时延

系统时延是指从发出系统操作启动请求到操作真正开始之间的时间长度。确定系统时延的原因并非易事，因为其中包含应用以及所有软件层，包括操作系统。

中断时延

中断时延是指从中断生成之后到执行中断之间的时间长度。中断时延往往通过 CPU 用来对当前处理环境进行中断以识别中断和响应所需的总时间来进行测量。另一种测量方法是对开始中断处理所需的时间进行测量。

SMP Linux 中，中断处理可由任意 CPU 执行，默认将在 CPU0 上运行。如果以默认方式执行中断处理，会导致一个 CPU 上的负载较大。可从用户空间对每个中断的 CPU 亲和度进行更改，以平衡处理负载。可借助一些应用程序来平衡中断负载，例如 irqbalance。

Cyclictest 是一种 Linux 工具，用来测量中断与中断响应开始之间的时间。将测量的时间与预期时间进行比较，两者之间的差值就是时延。多个事件可延迟实际的中断响应，此时可使用 cyclictest 识别和分析此延迟。

硬件剖析

进行剖析时，可将系统视为子系统的层级，每个都具有自身的性能监测能力。为了完全分析系统性能，对所有数据都要进行整合，并同步到一个视图中，以便设计人员了解系统中消耗时间的地方。

PL 具备传统的固定 SoC 所没有的灵活性，为设计人员提供多种实现选择。通过对设计进行剖析，能够充分理解具体实现方式的影响。

Program Trace Module (程序走线模块、 PTM)

PTM 模块基于 Program Flow Trace (PFT) 架构执行实时的指令流追踪。追踪工具利用 PTM 生成的信息重建全部或部分程序的执行。

PFT 架构假设，追踪工具 （例如 DS-5）可对被追踪代码的一个副本进行访问。因此， PTM 只在程序执行中的特定点（称为路径点）生成追踪数据。这样能减少 PTM 生成的追踪数据量。路径点是程序流程或事件 （例如异常处理）的变化。追踪工具使用路径点来跟随程序执行流程。

为实现完整的程序-流程重建， PTM 追踪：

- 间接分支——具有目标地址和条件代码
- 直接分支——只有条件代码
- 指令障碍指令
- 异常处理，指示异常在哪发生
- 处理器指令集状态的变化
- 处理器安全状态的变化
- Context-ID 变化
- 进入调试状态，退出调试状态 （当启动停止调试模式时）

经配置， PTM 还可追踪：

- 追踪路径点之间循环次数
- 全局系统时间戳
- 所选的直接分支的目标地址。

Performance Monitor Unit (性能监控部件、 PMU)

Cortex-A9 处理器 PMU 提供六个计数器，用来收集处理器和储存器系统的运行统计数据。每个计数器都可追踪 58 个事件——重要的系统性能测量结果——中的任意一个。软件应用，例如赛灵思系统调试器，可以利用事件计数结果以及软件执行时间来优化软件。赛灵思驱动可提供对此信息的访问，这样，软件应用就可以从驱动中取回并显示该信息。

Level 2 (L2) 缓存事件计数器

PL310 L2 缓存控制器包含两个用来监测缓存事件的计数器。这些事件有助于理解应用程序如何影响 L2 缓存以及如何得到优化。赛灵思驱动可提供对此信息的访问，这样，软件应用就可以从驱动中取回并显示该信息。

AXI Performance Monitor (AXI 性能监控、 APM)

LogiCORE IP AXI 性能监控器是一个软 IP 核，可构建到 PL 中用来测量 PL 中的 AMBA AXI 系统性能指标。性能监控器能测量系统中主机/从机 (AXI4/AXI3/AXI4-Stream)的总线延迟、特定时间内的储存器流量以及其他性能指标。该内核还可用来对软件应用进行实时剖析。

APM 无侵入地监测 AXI 系统，因此无需 CPU 处理。然而， ARM CPU 上的软件可初始化对具体数据的监测，然后从 APM 收集结果，实现系统剖析的最低侵入程度。赛灵思驱动可提供对此信息的访问，这样，软件应用就可以从驱动中取回并显示该信息。 APM 还可以由 System Debugger 在剖析模式下进行配置。

APM 包含以下剖析过程中的有用功能：

- 研究任意基于 AXI 的从机 （例如储存器控制器）的延迟，并调节内核。

- 获取系统级标准，例如写吞吐量、读吞吐量、平均互联读延迟等。
- 分析事务处理延迟，并识别在事务处理中导致更多空闲周期的代理程序。
- 比较两个相似的 AXI 代理程序。
- 计算除 AXI 之外的外部事件数量，例如 FIFO 上溢/下溢、中断等。
- 在监控器 slot 上记录具体事件，然后重建并分析行为/性能。

以太网统计寄存器

千兆位以太网控制器包含可由软件通过低层次接口进行访问的统计寄存器。统计寄存器负责记录与发送、接收操作相关的各种事件类型的数量。记数值有助于剖析和分析网络性能。赛灵思驱动可提供对此信息的访问，这样，软件应用就可以取回并显示该信息。

软/硬件划分

借助软件剖析，可识别出应用程序中的计算密集型功能。然后，可将这些功能编译到硬件并迁移到 PL 中，以实现更高性能。

硬件与软件之间的接口上存在可帮助二者之间进行数据交换的通信机制。将被加速功能的参数传送给 PL 中的硬件加速器，或使其可以为 PL 中的硬件加速器所使用，同时，会将硬件计算的结果返回到软件或使其可以为软件所用。

可利用以下任意数据移动方案，通过任意 PS AXI 端口 （AXI_ACP、 AXI_HP 或 slave AXI_GP）实现该通信功能，数据移动方案包括：

- 存储器映射寄存器
- AXI-Stream FIFO
- AXI-DMA

对硬件部署的端口和数据移动器的选择受多个因素影响，包括传输数据的大小、实现通信机制所需的资源，以及延迟要求。以下讨论一些最常用的解决方案。

在加速器中实现储存器映射寄存器可提供最简单的数据通信形式。在一种方法中，加速器是 AXI-slave。软件将所需的计算参数写入寄存器，并启动硬件。硬件完成后，加速器或者中断 PS，或者向软件轮询的状态寄存器位写入。在低功耗的应用案例中，使用事件输入可能要优于中断。然后，软件同过一系列寄存器读操作来获得结果。该方法的方框图如图 2-14 所示。

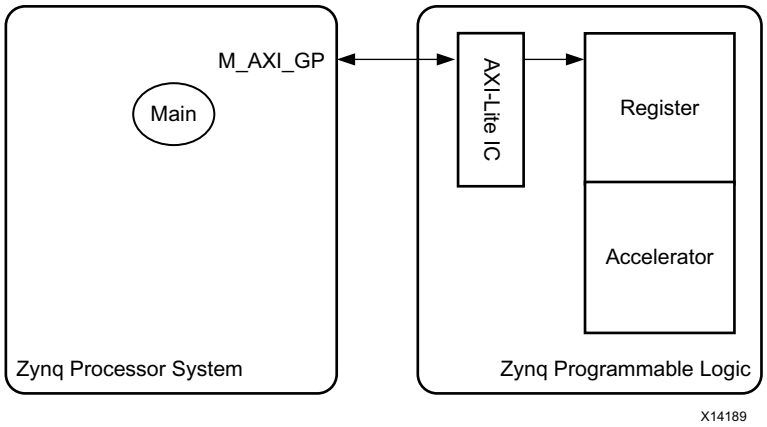

图 2-14 : 存储器映射寄存器

或者，将加速器作为 AXI-master，并在软件指导下从系统储存器或 PS 外设获取数据。该方法的方框图如图 2-15 所示。

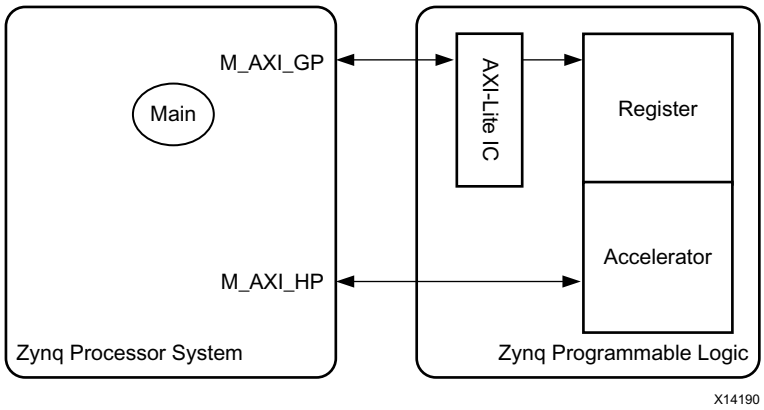

图 2-15 : 采用 AXI Master 的储存器映射寄存器

尽管基于寄存器的解决方案实现起来比较简单，但是只有当加速器在软件与硬件边界上传送较小的数据集时才能发挥最好效果。对单个寄存器进行访问需要软件开销，从而导致数据传输的成本高昂。对于较大的数据集，可针对流型接口使用 AXI FIFO。该方法的方框图如图 2-16 所示。

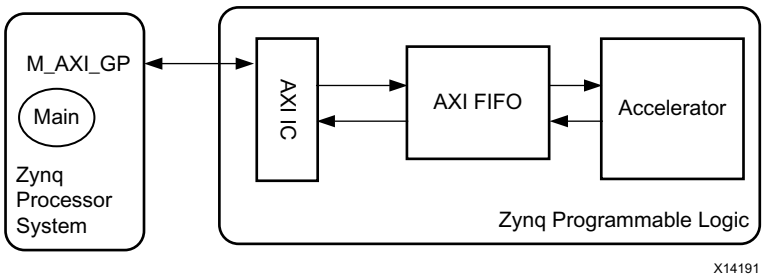

图 2-16 : 存储器映射 FIFO

硬件加速器经常用来处理大量数据，例如视频应用。数据存储在可由 PL 中的软件和硬件进行访问的储存器中, 例如 OCM 和 DDR。 DMA 用来将数据从储存器中取出，并在加速器处理完毕后再将数据送回储存器。该方法的方框图如图 2-17 所示。

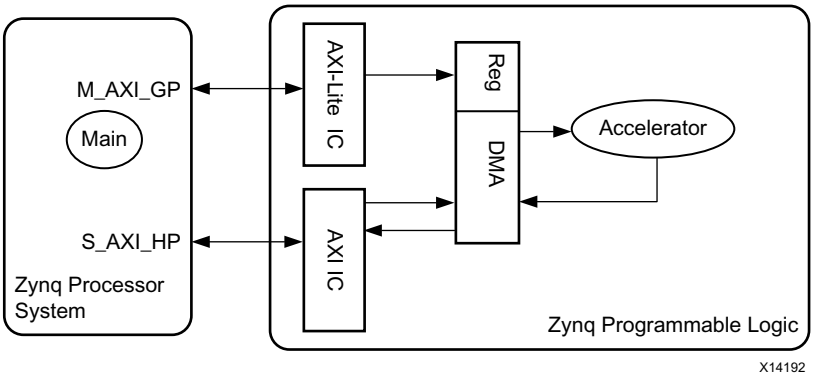

图 2-17 : 存储器映射 DMA

《Zynq-7000 All Programmable SoC ZC702 基础目标参考设计 (Vivado Design Suite 2014.2) 用户指南》(UG925) [参照 13] 是一个视频处理应用，用来在硬件和软件中实现 1080p60 视频流的边缘检测滤波器。在硬件实现方案中，视频

DMA 将视频数据从 DDR 取出，使其通过边缘检测引擎，然后再将视频流写回 DDR。软件可更新 DMA 寄存器，以控制视频数据的流动。

XILINX
ALL PROGRAMMABLE™

第 3 章

硬件设计考虑事项

本章介绍使用 Zynq®-7000 AP SoC 时需考虑的下列硬件设计问题:

- 第 44 页的“配置和启动器件”：您可以通过各种主要与次级启动器件来启动。本节介绍启动流程和启动器件选项。
- 第 48 页的“存储器接口”：您可以通过一个 16 位或 32 位的数据总线将 DDR 储存器控制器连接到各种 DDR 储存器器件中。同时支持 ECC 选项。
- 第 52 页的“外设”：Zynq-7000 AP SoC 支持各种外设。本节介绍了外设和它们与应用处理单元 (APU) 的相互作用。
- 第 65 页的“设计 IP 模块：”：IP 模块是预先设计、预先验证且可重复使用的功能模块，可以帮助您缩短设计时间。本节介绍设计包含 IP 块的解决方案流程。
- 第 70 页的“硬件性能考虑事项”：本节介绍可用于调整 AXI 主， AXI 从， AXI 数据路径性能的硬件性能指标和方法。
- 第 74 页的“数据流程”：本节介绍了在处理系统 (PS) 内和 PS 与可编程逻辑 (PL) 之间的数据流程。
- 第 76 页的“PL 时钟方法”：本节介绍了 PL 时钟的控制方法。介绍了不同的 PL 时钟源，以及建议用途。
- 第 80 页的“ACP 和高速缓存一致性”：本节介绍 ACP为 PL 主控器提供低时延访问的能力，包括 L1 缓存的可选一致性。
- 第 82 页的“PL 高性能端口访问”：您可以使用 HP 端口为 PL 提供 DDR 控制器与片上储存器 (OCM) 的直接访问。本节介绍了使用 HP 端口的设计驱动优化。
- 第 84 页的“系统管理硬件辅助”：本节讨论了包括基于用户特定输入的系统级参数控制的系统管理。
- 第 87 页的“管理硬件重新配置”：对于 FPGA 的非静态逻辑部分您可以应用部分重配置。本节介绍了部分重配置设计的过程。
- 第 91 页的“GP 和来自 APU 的直接 PL 访问”：本节介绍如何使用 GP 接口从 APU 访问 PL 内的寄存器和储存器。

配置和启动器件

本节介绍了 Zynq-7000 AP SoC 的启动器件以及它们是如何配置的。它涵盖了启动流程和以代理方式启动初始化序列的 BootROM 的作用。介绍了包括四通道 SPI， SD， NAND， NOR 闪存和 JTAG 启动模式第一启动选项与 Zynq-7000 AP SoC 中的 eMMC 和 PCIe 第二启动选项。本节着重介绍在系统级方案中启动流程和启动器件的可用性。 JTAG 启动模式被视为一个从启动模式，它始终是一个非安全启动模式。

典型的启动流程

Zynq-7000 AP SoC 的启动顺序包括从 NAND 或 NOR 闪存典型的外部静态存储读取第一阶段启动加载程序 (FSBL)。BootROM 包含一个上电复位后执行的代码，并且代码通过外部静态存储读取 FSBL。

在 PS_POR_B 引脚停止激活后，硬件采样启动引脚和配置 PS 与 PLL。 PS 内部的 BootROM 软件执行，代码首先配置 ARM 内核与所需外设，从选定的启动器件将 FSBL 复制到 OCM，然后代码切换到从 OCM 执行的模式。您可以从上述主启动器件复制 FSBL 或者 通过 JTAG 进行加载。或者，在非安全启动模式下，可以从支持就地执行的四通道 SPI/NOR 闪存执行 FSBL。

图 3-1 介绍了典型的启动流程。

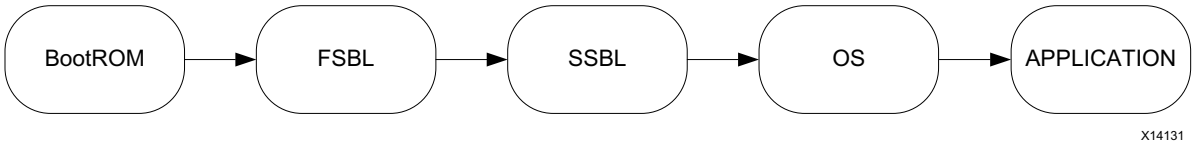

图 3-1 : 典型的启动流程

您可以在安全和非安全模式下启动 Zynq-7000 AP SoC。安全模式启动仅适用于静态储存器，而在非安全模式下可以通过 JTAG 和静态储存器启动。在系统开发阶段主要使用 JTAG 启动。

选择启动器件

下面是 Zynq-7000 AP SoC 的主要启动选项：

- 可就地执行的四通道 SPI 模式
- NAND 闪存
- 可就地执行的 NOR 闪存模式
- SD 存储卡
- JTAG

下面是 Zynq-7000 AP SoC 的二级启动选项：

- eMMC
- PCIe，以太网， USB， UART，或定制 FPGA 接口

您可以选择使用以下任何条件的 BootROM 器件：

- 器件可以与 Zynq-7000 AP SoC 的 BootROM 配合使用并且由赛灵思工具 (IMPACT， SDK) 以及更高级别的软件，如 U-Boot 和 Linux 提供支持。您可以在赛灵思解答记录 50991 [参照 67] 中找到赛灵思对于不同的启动选项推荐的一系列器件。
- 器件满足应用的配置引脚要求。例如， QSPI 启动选项比 NAND或 NOR 启动选项需要的引脚更少。
- 器件满足应用的大小要求。 NAND 和 SD 启动选项比 QSPI 和 NOR 启动选项提供的存储密度更高。
- 器件满足应用的配置速度要求。 QSPI 是最快的启动选项。
- 器件管理可以由应用进行适当处理。 NAND 器件比其它启动选项管理难度更大。针对坏模块，需要做出设计决策，决定事件发生时如何管理。

您的启动器件选择将会影响所需引脚数目，最大储存器大小，启动时间，以及器件管理软件的复杂性。表 3-1 总结了设计权衡。

表 3-1 : 闪存储存器比较

外设	引脚	启动器件	XIP	限制	读/写	最大尺寸	启动时间
QSPI	7 个单层， 8 个双堆叠， 13 个双并行	有 在单层与堆叠 16MB 与并行 32MB 的启动镜像。	有	每个启动器件最大容量为 16MB。 极慢擦 / 写 - 通常用为只读。 经常用在 eMMC 或 SD 卡中。	字节/模块	线性模式下为 16MB 在 I/O 下每个 QSPI 为 128MB每个控制器最多可支持两个 QSPI。	快速
NAND	X8–14 X16–22	有 启动镜像在第一个 128MB 中。	无	需要 ECC、损耗平衡、坏模块的软件管理。 对于原始闪存文件，如 JFFS2 或 UBIFS，需要文件系统。 需要 ECC。 硬件只支持 1-bit ECC。	模块/模块	1GB	中
SD	6	有	无	需要导线连接器	模块/模块	任意大小	缓慢
并行 NOR	37	有	有	支持类似大型 QSPI 的容量,但是需要高引脚数。	字节/模块	64MB	快速
eMMC	6	无	无	无法从 eMMC 启动	模块/模块	任意大小	不適用

以下提供每个启动选项的详细信息。

可选就地执行的四通道 SPI 模式

BootROM 可以使用宽度检测参数值 (0xAA995566) 和镜像识别参数值 (0x584C4E58)，检测四通道 SPI 接口的预期 I/O 宽度。四通道 SPI 是最快的配置方案。其中只有 SD 存储卡启动选项具有较低的引脚数量。您可以在 Zynq-7000 AP SoC 中访问作为线性储存器的四通道 SPI。器件管理更加简单，因为同例如 NAND 闪存这类其他器件相比，坏模块问题更少。在线性模式下，四通道 SPI 单模下最多支持 16MB，双模最多支持 32MB。当 QSPI 在 I/O 模式下操作时，支持超过 16MB。对于就地执行选项，在非安全启动模式下， BootROM 会使用四通道 SPI 控制器的线性寻址功能。可以使用 QSPI 进行 Multiboot。

推荐采用美光 (N25Q) 和 Spansion (25FL) 器件作为四通道 SPI 启动器件系列。

NAND 闪存

NAND 闪存启动选项是一种支持高密度器件的廉价解决方案。唯一的限制是启动镜像必须位于 NAND 闪存器件的第一个 1GB 地址空间中。该启动选项的性能比四通道 SPI 启动选项低。典型的 NAND 闪存解决方案比四通道 SPI 解决方案需要更多的引脚且具有更低的储存器带宽。应用设计需要包含在启动过程中管理不良时钟的机制。可以使用 NAND 进行 Multiboot。

推荐采用美光 （管芯 ECC）和 Spansion (S34) 器件作为 NAND 启动器件系列。美光 NAND 闪存器件通常需要多位 ECC，必须只能采用支持管芯 ECC 的器件。

可选就地执行 NOR 闪存模式

您可以通过 Zynq-7000 AP SoC 访问作为线性储存器的 NOR 闪存。此外，相比于在 NAND 闪存器件中的坏模块，这里坏模块的问题更加轻微。 NOR 闪存的密度堪比四通道 SPI。相对于其他启动选项使用了更多 MIO 引脚。一个典型的 NOR 闪存 （字节外设接口）采用 40 引脚的 MIO 并且最多可支持 64MB。

推荐采用美光 (M29EW) 和Spansion (29GL) 作为 NOR 闪存启动器件。

SD 存储卡

SD 存储卡密度比 NAND 闪存启动选项的更高。器件通常作为文件系统管理。坏模块不需要使用应用设计来管理。 SD 存储卡比四通道 SPI 启动选项慢，另外 SD 卡需要一个板上连接器。eMMC 的器件不属于主启动器件，但是您可以用它们作为次级启动源。在SD 启动模式下， BootROM 不执行报头搜索，并且不支持多启动选项。您可以通过将 ARM_CLK_CTRL 寄存器 (0x1F000200) 的 CPU 时钟分频器设置为 2 来提高 SD 启动模式启动速度。

JTAG

在 JTAG 启动模式中，您可以选择独立的 JTAG 模式支持将调试器连接到 ARM DAP 控制器以及将其他工具连接到赛灵思 PL TAP 控制器。使用 PL JTAG 接口连接到专用 PL 引脚来访问赛灵思 PL TAP 控制器。使用 EMIO JTA G接口来访问 ARM DAP 控制器 （用于其他工具）。这需要下载一个码流到 PL。您也可以使用赛灵思 PL TAP 控制器下载码流。

eMMC

当 QSPI 是主启动选项并且使用一个小型 QSPI 储存器时，您可以使用 eMMC 次级启动选项。通常情况下，FSBL 将加载到 QSPI，其他划分将加载到 eMMC。赛灵思建议只使用在标准速度模式下（25MHz 的最高频率）具有 SDIO 控制器的 eMMC。

PCIe、 Ethernet、 USB、 UART、和定制 FPGA 接口

您也可以使用 PCIe，以太网， USB， UART 或定制 FPGA 接口来实现次级启动。您可以选择适合您应用的选项。下面讨论基于 PCIe 的次级启动选项。

PCIe 次级启动使用 PCIe 协议，从使用 PCIe 根复合体的主系统获取第二阶段加载程序。在 Zynq-7000 AP SoC 的 PL PCIe 模块实现了终点功能，与相关的 PCIe 根复合体建立了通信链路。您可以从外部储存器 （QSPI， NAND 或 NOR 闪存）获取 FSBL 镜像，当 FSBL 执行完成后， FSBL 可以从主系统储存器的 PCIe 获取 U-Boot 次级镜像。

为了执行这个选项，您需要在设计中生成一个码流例化的 PCIe 模块。例如，您可以使用赛灵思 7 系列的 PCIe IP 模块实现编程 I/O (PIO) 的设计。为实现 APU 和主 CPU 之间的握手状态，您还需要执行一套 PL 状态寄存器，并将寄存器接口连接到 PS 主接口与 GP 端口。启动流程为：

- 从主启动器件装载 FSBL。
- APU 会等待 FSBL 执行外设初始化进程。
- 通过使用 PIO 设计和握手状态寄存器集 FSBL 将编程位文件。
- 当位文件编程完成后， APU 编程 bitDone 寄存器。
- 主 PIO 驱动读取 bitDone 寄存器并将 `U-Boot.elf` 文件写入 PL 块状 RAM。
- `U-Boot.elf` 写入后，状态被写入到 U-BootDone 寄存器。
- APU 轮询 U-BootDone 位，然后做将文件复制到 PS DDR 储存器。

文件系统

闪存选择会影响可实现的文件系统。下面介绍的文件系统的选择仅限于 Linux，但对于相关问题提供了良好的概括。

eMMC 和 SD 卡有内置的控制器，在运行的闪速转换层 (FTL) 固件时能够使器件在 OS 作为模块器件显示。传统的文件系统，如 FAT 或 ext3，可以与模块器件共同工作并且可以使用这些器件来实现。

原始闪存器件 （如 NAND， QSPI，或 并行 NOR）需要使用软件管理，使读入和写入到储存器单元能够正常工作。Linux 使用储存器技术器件 (MTD) 子系统，提供特定硬件器件驱动和更高级别的应用之间的抽象层。 Linux 支持那些在 MTD 器件，如 JFFS2 和 UBIFS 顶层的文件系统。这些文件系统设计目的包括处理诸如耗损均衡和坏模块管理问题的软件管理算法，并且为了保证器件正常运行必须使用。

优化启动时间

用于从每个闪存器件类型来读取的 BootROM 设置，选择要素在于最大兼容性，这往往是以牺牲性能为代价。通过在 BootROM 报头的寄存器初始化部分设置相应的控制寄存器，可以改善启动接口性能。设置取决于所使用的器件和电路板布局参数。经优化的寄存器值应从所用器件供应商的数据手册中获得。每个启动器件优化值的示例可以在 Zynq-7000 All Programmable SoC 技术参考手册 (UG585) [参照 4] 中的“通过寄存器初始化优化启动时间”章节和 赛灵思解答记录55572 [参照 69] 中找到。经优化的值可使用 Bootgen 添加到启动镜像报头。如需了解更多信息，请参阅：《Zynq-7000 All Programmable SoC 软件开发指南》(UG821) [参照 7] 中的“使用 Bootgen”附录。《Zynq-7000 All Programmable SoC 技术参考手册》(UG585) [参照 4] 中的“启动和配置”章节的 BootROM 性能部分提供了在不同的启动阶段提高时序和带宽成效的方法。

案例分析：QSPI 启动时间的变化

本案例研究介绍了以 100MHz 运行，并且改变 PS 频率使用不同 QSPI 器件通过赛灵思 ZC702 板完成启动时间的测量。

开机时间 （从 U-Boot 启动到 Linux 的提示），对于以 667MHz 运行、 PS 32MB 的 Spansion 单一器件大约是 3036 毫秒。对于在相同频率运行的 PS 64 MB Spansion 双并行器件启动时间是大约 2994 毫秒。因此，使用一个是单一 QSPI 器件容量两倍的双平行 QSPI 器件，在不改变 PS 频率情况下降低了近 41 毫秒的启动时间。

更改 PS 频率也会影响启动时间。如上所述，对于在 667MHz 运行的 PS 32 MB Spansion 单一器件开机时间大约是 3036 毫秒。但是相同的 QSPI 器件， PS 速度增加到 867MHz 的结果是启动时间约为 2523 毫秒。因此，在不改变 QSPI 器件的情况下， PS 运行频率从 667MHz 增加到 867MHz 减少了大约 513 毫秒启动时间。

存储器接口

DDR

Zynq-7000 AP SoC 的 DDR 多协议存储控制器支持 1.8V 的 VDDR2，1.2V LPDDR2，1.5V DDR3 和 1.35V 的 DDR3L。它可以配置为提供一个 16 位或 32 位宽的数据总线。所有器件都支持 16 位和 32 位数据总线宽度的选择，但 7z010 CLG225 器件仅支持 16 位数据总线宽度。控制器还支持 ECC 中的 32 位配置，具备 16 个数据位和 10 个校验位。支持 ECC 时数据宽度被限制为 16 位。将一个 1GB 地址映射分配给了 DDR。然而，如果使用 ECC，只有 512MB 地址空间是可用的。

DDR 储存器控制器包含三个主要模块：一个 AXI 储存器接口 (DDRI)，一个核心控制器与事务调度器 (DDRC)，以及数字 PHY (DDRP) 控制器。有关每个模块和其他控制器的详情，请参阅：《Zynq-7000 All Programmable SoC 技术参考手册》(UG585) [参照 4] 中的链接。

控制器包括数字 PHY，具有一组专属 I/O 。对于某些 Zynq-7000 AP SoC 器件的最高速度等级，支持的最大总线时钟为 DDR3 模式 666 2/3MHz。对于其它所有速度等级，支持的最大总线时钟为 DDR3模式 533 MHz。理论上的最大总线比特率是：

```
Data transfer rate = 666 2/3 MHz * 2 bits (for double data rate) = 1333 Mb/s per data IO
```

使用 32 位的最大总线宽度时，所述最大总线带宽为 42.6 Gb/s，或 5.3 GB /s。如图 3-2 所示这个带宽是由使用四个 DDRI 从端口连接的多个主共享。DDRI 模块连接到 4 个 64 位 AXI 同步接口并且同时作为多个 AXI 主控。每个 AXI 接口都有其自己的专用事务 FIFO。端口 S0 连接 L2 高速缓存，并且只服务于 PL、 CPU 和 ACP 接口，确保低时延和快速访问。端口 P1 由 PS 外设和 AXI GP 端口的所有中央互联主控共享。四个 PL 的 AXI_HP 接口成对向下多路复用，并且连接到端口 2 和 3，如图 3-2 所示。

最大总线带宽为 5.3 GB/s，但它是不可持续的。要提高 DDR 的效率，必须考虑与 DDR 相关的开销， DDR 效率受数据访问模式的影响。最大限度地减少页和行变化的地址模式降低了页面和行开销的变化，从而获得更高的利用率和更高的系统吞吐量。请参阅：《Zynq-7000 All Programmable SoC 技术参考手册》(UG585) [参照 4] 中的“DDR储存器控制器”章节的“行/行/列地址映射”链接)。

PL AXI 主控连接到 HP 端口的方式也决定了系统的吞吐量。多个 AXI 主控更有可能产生随机地址访问，引发更多的页面缺失，从而导致较低的 DDR 效率。如图 3-2 所示， AXI_HP 的 DDR 互联多路复用中 4 个 PL AXI_HP 分成两对，连接端口 S2 和 S3，两对都通过互联仲裁。当只有两个 PL AXI 主控时，可以由主控连接到端口 AXI_HP0 和 AXI_HP2，或 AXI_HP1 和 AXI_HP3，也就是 DDR 控制器的端口 2 和端口 3，从而实现更高性能。在一个典型基于 Linux 的视频设计中，从 AXI_HP 端口实现的系统带宽约为 50%。请参阅：《利用 Zynq-7000 All Programmable SoC 设计高性能视频系统》(XAPP792) [参照 34] 中的详细介绍。

控制器 DDRC 模块包含改进 DDR 控制时延一个的三阶段仲裁器。可以通过改变寄存器设置来控制时延。如需了解更多有关管理 DDR 时延的详情，请参阅：《Zynq-7000 All Programmable SoC 技术参考手册》(UG585) [参照 4]“DDR 储存器控制器”中的“仲裁 DDRC”小节 （链接）。

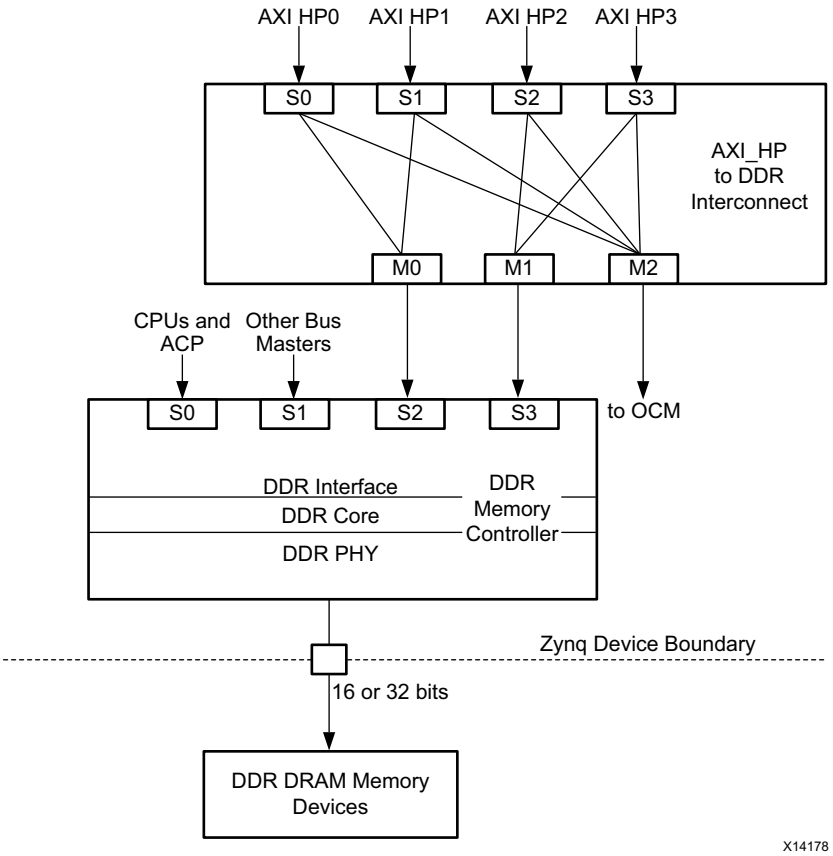

图 3-2 : DDR 储存器控制器框图和互联 AXI_HP 端口

为促进高带宽操作， DDR 接口时序必须正确地初始化和校准。为帮助自动确定数据与可靠数据采集的最佳窗口对齐所需的时延， DDRP 包含了 DRAM 培训功能。此功能介绍请参阅：《Zynq-7000 All Programmable SoC技术参考手册》(UG585) [参照 4]“DDR储存器控制器”中的“初始化和校准”小节链接。

Zynq-7000 AP SoC 工具流程有助于自动化地进行 DDR 启动。要做到这一点， PS DDRC 板参数需要根据赛灵思解答记录 46778 [参照 65] 进行配置。这将在硬件设计流程中导入板上 DDR 信号的时延特性。这些特征被用来确定自动训练算法中使用的初始值，或者当自动算法用于计算静态接口时序且不被特定的 DDR 标准支持时。时序值是设计硬件平台规格的一部分，并输出到 SDK 作为 FSBL 调用的 PS 初始化代码使用。 BootROM 不使用 DDR。

QSPI

PS QSPI 闪存控制器使用 QSPI 接口与外部串行闪存进行通信。储存器闪存单元平行排列，有时被称为 NOR 闪存。而这种配置的密度较小，并且具有比 NAND 闪存更小储存器容量，允许在单字节阵列中的任何地方进行读取。对于读取操作，它与标准地址映射储存器行为类似，非常适合代码存储。此外，它支持就地执行 (XIP) 功能， CPU 可以直接执行

QSPI 代码，而无需先读取代码到 DDR 或 OCM。《Zynq-7000 AP SoC启动 - 无外部储存器的启动和运行技术提示》维基页 [参照 59] 提供了在 ZC702 板上使用 XIP 的一个“Hello World”设计示例。

在 Zynq-7000 AP SoC 支持的所有闪存中， QSPI 具有最高的读取性能，并以一个非常低引脚数要求提供了最快启动解决方案。它支持基于页面的写入和基于扇区的擦除，但与其它类型的器件相比，其速率十分慢。对于具有快速启动时间和大容量储存器、具备成本效益的解决方案，您可以用小型 QSPI 作为存储 FSBL 的主启动器件。所有其他分区可以放置在一个较大的闪存上，如 eMMC 或 SD。如需了解更多关于如何实现此类系统的信息，请参阅：《Zynq-7000 All Programmable SoC 软件开发指南》(UG821) [参照 7] 中的 “eMMC 闪存器件”章节 （链接）。

QSPI 闪存控制器支持三种不同操作模式：I/O 模式，线性寻址模式，和传统 SPI 模式。线性寻址模式， QSPI 控制器接收的 AXI 事务被自动转录成连接到闪存器件 QSPI 总线上相应的命令和数据处理。控制器支持的闪存地址只有 24 位，因此 QSPI 的线性模式下最大限制为 16MB。 BootROM 访问线性模式的 QSPI。当 XIP 模式使用 QSPI 时， FSBL 的长度被限定于 QSPI 的线性模式时的容量，减去启动报头的长度。当启用 XIP 模式 （通过启动报头时）， BootROM 控制权交由 FSBL 直接从闪存 QSPI 执行，而不是复制 FSBL 到 OCM。当 XIP 模式不使用 QSPI 时， BootROM 复制 FSBL 到 OCM 执行。在这种情况下， FSBL 的大小由 192KB OCM 容量的限制。 OCM 的前四分之三位于低储存器，顶部四分之一被映射到高储存器。 FSBL 可以使用最顶部的四分之一进行栈和堆的项目。

在 I/O 模式中， TXD 寄存器的软件构成命令和数据，控制器以正确的格式从寄存器到闪存驱动内容。由闪存驱动的数据移入相应 RXD 寄存器中，然后数据由软件提取。使用 I/O 模式，软件可以发出闪存命令来专门修改闪存内特定的寄存器位，所以可以通过页面之间有效地切换来扩大 QSPI 闪存地址空间。通过这种方法，控制器支持的每个 QSPI 可以高达 128MB。如需了解更多有关 QSPI I/O 模式的详情，请参阅：《Zynq-7000 All Programmable SoC 技术参考手册》(UG585) [参照 4] 中的“Quad-SPI 闪存控制器”章节“I/O 模式”小节 （链接）。

当在电路板上使用的 QSPI 超过 16MB 时，任何 Zynq-7000 AP SoC 平台复位必须触发 QSPI 复位,从而使页面寄存器复位。这确保了在线性模式下，当 BootROM 从地址 0x0 读取时可以访问启动镜像。如需了解当使用闪存器件大于 16MB 时复位要求的更多信息，请参阅：赛灵思解答记录57744 [参照 70]。

如果采用 RSA 加密或 XIP 模式在闪存器件上存储大于 16MB 的启动镜像，启动镜像不能放在偏移量为 0x0 的位置。相反，启动镜像可以位于 0x0+32K 偏移处。如果镜像位于 0x0，重复的头镜像可以位于偏移量为 0x0+16MB 处，否则可以使用一个单一的 X1 QSPI。

取决于 QSPI 时钟频率和用于执行读取命令的类型，不同 QSP I闪存器件需要不同的虚拟时钟周期。根据 QSPI 的板上配置方式 （X1， X2， X4，单一，堆叠，或并行）和使用的特定闪存器件，为了实现闪存器件适当通信，控制器的 LQSPI_CFG 寄存器必须妥善设置。当使用赛灵思支持的 QSPI 闪存器件时，赛灵思的 PS QSPI 器件驱动自动向 LQSPI_CFG 寄存器写入适当的值。赛灵思支持美光公司和 Spansion 公司生产的 QSPI 器件。如需了解更多有关目前 Zynq-7000 AP SoC 工具支持的厂商闪存器件信息，请参阅：赛灵思解答记录 50991 [参照 67]。

BootROM 在 BootROM 报头通过检查宽度检测值和镜像识别值使用或者快速读取 (X1)，四输出读取 (X4)，双输出快速读取 (X2)，单模或并行模式自动执行读取命令。根据具体值， BootROM 使用支持度最广的 I/O 总线宽度来读取四通道 SPI 器件数据，但通过 X1 发送命令。 BootROM 将一组初始值写入 LQSPI_CFG 寄存器。如需了解关于这些值的更多详细信息，请参阅：《Zynq-7000 All Programmable SoC 技术参考手册》(UG585) [参照 4] 中的“启动和配置”章节“四通道 SPI 启动”小节 （链接）。

在一个储存器接口时钟大于 40 MHz 的高速 QSPI 应用中，必须使用 QSPI 反馈模式。如需了解更多有关 QSPI 反馈时钟的详情，请参阅：《Zynq-7000 All Programmable SoC 技术参考手册》 (UG585) [参照 4] 中的“四通道 SPI 闪存控制器”章节“四通道 SPI 反馈时钟”小节 （链接）。

QSPI 可以通过 U-Boot、 Linux、 iMPACT 以及 SDK 进行编程。

如需了解更多有关 QSPI 闪存控制器的详情，请参阅：《Zynq-7000 All Programmable SoC 技术参考手册》(UG585) [参照 4] 中的“四通道SPI闪存控制器”章节 （链接）。

静态储存器控制器

静态储存器控制器有两个接口模式：NAND 闪速接口模式和并行端口储存器接口模式。 NAND 闪存接口模式支持 NAND 闪存，而并口接口模式支持 NOR 闪存和同步 SRAM。因为 QSPI 和 NOR 闪存都使用基于 NOR 的存储单元，又因为在线性模式下 QSPI 被限制于 16Mb，容量成为了选择 NOR 闪存而非 QSPI 的差异化因素。然而，因为静态储存器控制器限制的地址行的数量为 26 位，它支持的最大 NOR 闪存仅为 64MB。16MB QSPI 限制（BootROM 读取数据不得超过 16MB）仅适用启动。在 BootROM 加载 FSBL 到 OCM 或 DDR 储存器之后，在闪存器件页面寄存器的支持下，

QSPI 控制器切换到 I/O 模式并可访问高达 128MB 的储存器。因此，与 NOR 闪存控制器支持 64MB 器件 （40 引脚情况下）的能力的相比， QSPI 支持 128MB 器件 （仅使用 8 引脚）使得 QSPI 超过 NOR 闪存，成为优先解决方案。

以下部分介绍了 NAND闪存接口模式。如需了解更多有关使用并口储存器接口的信息，请参阅：《Zynq-7000 All Programmable SoC 技术参考手册》(UG585) [参照 4]。

NAND 储存器控制器

在 NAND 闪存中，储存器单元串联布置，导致储存器单元更加密集，单位硅片面积内容量更高，与 NOR 闪存相比每比特成本更低。 NAND 闪存控制器通过 8 位或 16 位 I/O 总线地址/数据/指令可支持多达 1GB 的外部 NAND 闪存。支持开放式 NAND 闪存接口 1.0 规格。

致密存储单元在编程期间和擦除周期会压力更高，使它们更容易产生错误。采用 ECC 减少这些错误。 NAND 闪存控制器包括用于 1 位 ECC 校正的硬件支持。软件用于运行闪存管理算法，利用 ECC 数据来管理各种错误模式、处理坏模块、整个储存器单元损耗一致化，并提高单元耐力和数据保持能力。

因为控制器仅支持 1 位 ECC，仅具有片上 ECC 或一位 ECC 的 NAND 器件可以与 Zynq-7000 AP SoC 一起使用。目前，只有单级单元 (SLC) 器件符合 ECC 标准，并且不支持多级单元 (MLC) 器件。另外，控制器仅支持单个芯片选择。在 x8 和 x16 两个配置中，赛灵思支持 NAND 器件的容量范围包括从 128MB 到 1GB。赛灵思支持来自美光和 Spansion 的 NAND 器件。如需了解更多有关 Zynq-7000 AP SoC 支持的厂商闪存器件信息，敬参阅赛灵思解答记录50991 [参照 67]。

即使在高容量 NAND 器件中， Zynq-7000 AP SoC 的 BootROM 将在第一个 128MB 内寻找一个启动镜像启动。因此，标准和回读 （如果使用）启动镜像的开始部分必须在第一个 128MB 开始。

与 NAND 闪存通信基于一组器件间器件各不相同的 AC 时序参数。对于 Zynq-7000 AP SoC 与 NAND 器件使用正确时序通信，设计师基于 AC 时序值，需要输入一个的相关时序参数到 SMC 时序计算页面上的 CS0 列。如图 3-3 所示。

CS0 周期是基于 NAND 时钟频率自动计算。值输出到 SDK 的 PS 初始化代码,作为设计的硬件平台规格的一部分。初始化代码将这些值写入 smc.SET_CYCLE 寄存器。

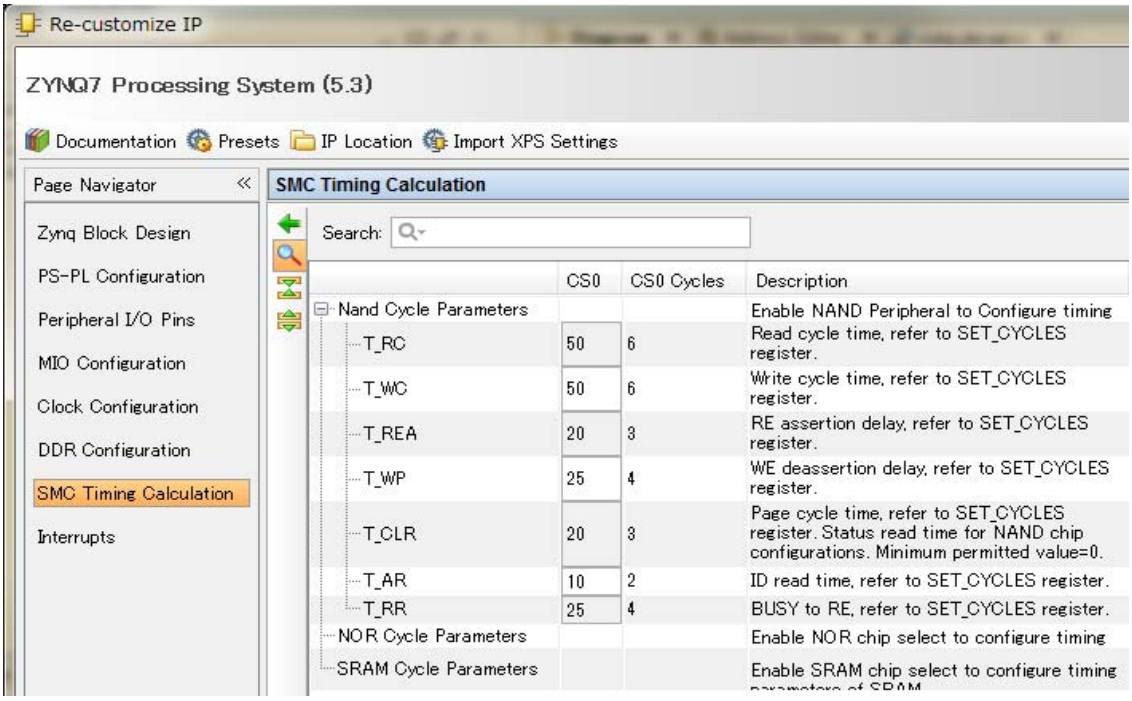

图 3-3 : SMC 时序页示例

BootROM 在执行 PS 初始化代码之前，使用一组存储在 smc.SET_CYCLE 寄存器的初始值从 NAND 闪存进行读取。如需了解有关这些值的更多详情，请查阅 《Zynq-7000 All Programmable SoC 技术参考手册》(UG585) [参照 4] 中的“启动和配置”章节“NAND 启动”小节。

在 NAND 闪存中，读取和写入发生在页面端，然而擦除发生在模块端。因为 NAND 闪存行为不同于随机存取储存器，Linux 系统将它作为一个抽象层储存器技术器件 (MTD) 使用，允许软件访问使用 MTD 子系统的 API 器件。在不同的闪存类型和技术中， API 较为常见。 MTD 不属于模块器件，并且它缺乏软件管理算法来处理诸如耗损均衡和坏模块管理问题。文件系统必须设计为在如 JFFS2 或 UBIFS 的原始闪存顶部工作，而不是传统的文件系统像 ext2， ext3 和 FAT（这些在模块器件上工作）。 JFFS2 在 MTD 子系统的顶部工作。 UBIFS 在 UBI 子系统顶部工作，并且在 MTD 子系统顶部工作的部分提供 NAND 器件所需的软件管理算法。

外设

本节对在 Zynq-7000 AP SoC PS 储存器映射的外设和其与 APU 和储存器控制器之间的互动进行了说明。读者应熟悉 AMBA 总线拓扑结构 （AHB 和 APB）并且对 USB， CAN， UART，以太网，和 SPI 协议具有一定的理解。

Zynq-7000 AP SoC 外设大致可分为：

- PS 外设
- PL 外设

PS 外设是硬连接的外设，并且可作为标准 ASSP 设计实现的一部分来实现。 PL 外设支持即时编程与即时配置。本节介绍 PS 外设，下面的部分包含了所有的 PL 外设。

PS 外设经过储存器映射，通过外设储存器映射也经过预定义。外设实行高级微控制器总线架构 (AMBA) 兼容协议的总线结构，并且连接至中央互联或应用处理器单元的一部分。通过 AMBA 指定事务与 ARM Cortex-A9 处理器进行通信。外设可识别 ARM 的 TrustZone 安全性，并可选拒绝来自非安全主控的非安全访问。连接到中央互联的外设可导向至 PL，实现 PL 访问。

图 3-4 显示了 PS 中外设的顶层框图。

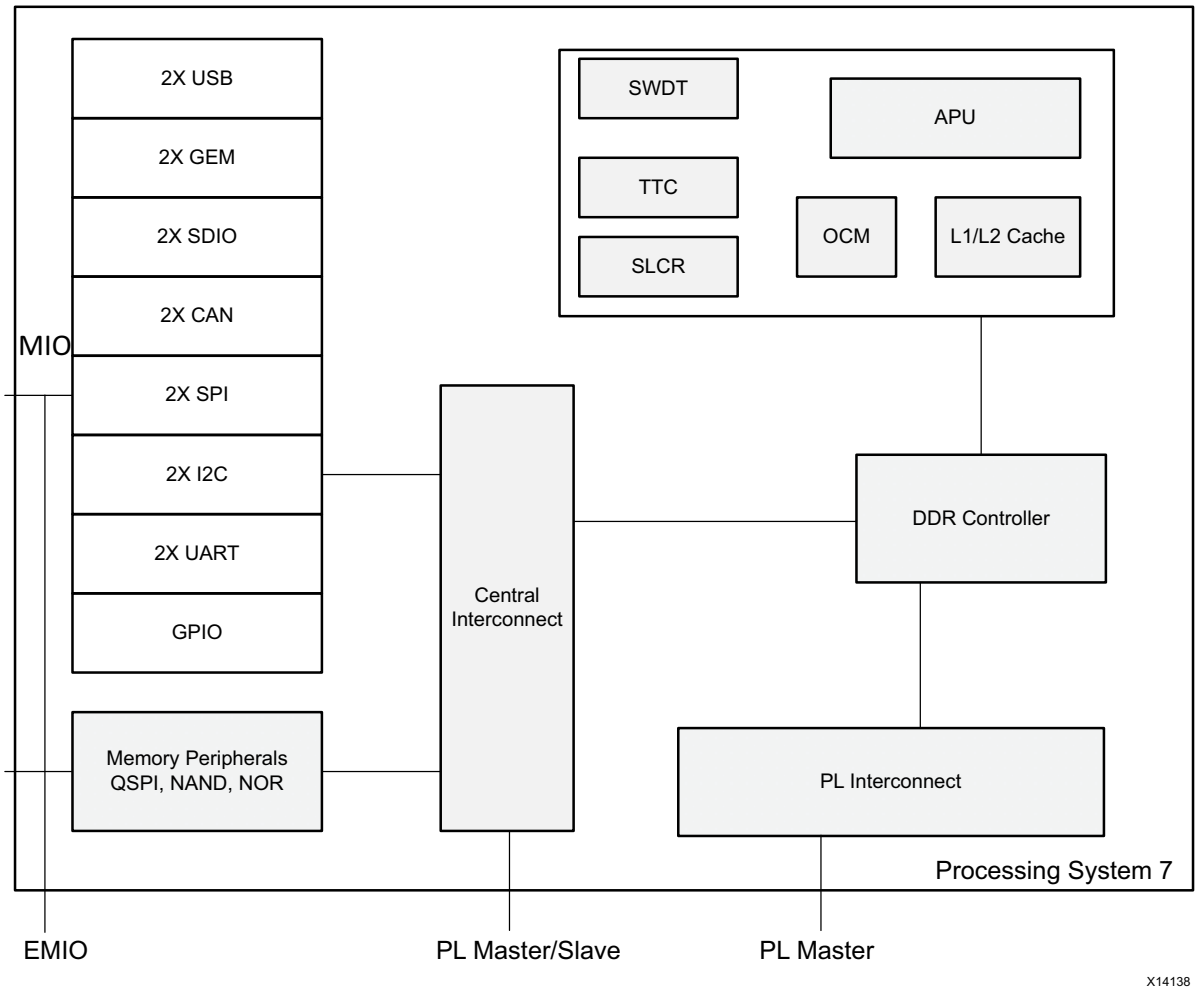

图 3-4 : PS 外设模块原理图

外设说明

以下是 PS 模块中的关键外设。

USB 外设

PS USB 控制器兼容 USB 2.0 标准，可以在以下任一模式下运行：

- 器件模式
- 主机模式
- On-The-Go (OTG) 模式

USB 控制器使用 ULPI 接口连接到外部 PHY，这是通过 MIO 的一个 8 行 SDR 数据总线来实现的。ULPI PHY 不属于 USB 控制器的一部分，并且您可以将任何兼容 ULPI 的 PHY 连接到 Zynq-7000 AP SoC 的 USB 控制器。端口控制器指示，电源选择，和电源故障信号可以通过 EMIO 导向至 PL。边带信号也可以导向至 SelectIO™ 引脚。 USB 控制器有一个内置 DMA 引擎，具有发送和接收用 FIFO，可实现至 AHB 总线的数据往来传输。

如需了解更多信息，请参阅：《Zynq-7000 All Programmable SoC技术参考手册》(UG585) [参照 4] 中的“USB 主控、器件和 OTG 控制器”章节 （链接）。

数据流

从储存器由 DMA 传送的数据使用 USB2.0 协议格式封装，并通过 ULPI 接口发送到 PHY。

系统级考虑事项

USB 控制器由两个时钟域计时：

- AHB 接口由 CPU_1x 时钟域计时。
- USB 协议引擎和端口控制器接口使用由 ULPI PHY 产生的 60 MHz 时钟。

您可以使用三种不同的方法重置 USB 控制器：

- 您可以通过 PS 复位，使用 APB 接口写入 USB 命令寄存器触发控制器复位。
- 您可以在 GPIO 信号的控制下触发 ULPI PHY 复位。
- 您可以在 OTG 模式下利用自动复位功能触发 USB 总线复位。

您可以配置控制器作为 USB 主控、USB 器件，或者作为 OTG。配置由您的应用决定。控制器主控模式要求 USB 主控制器的接口建立在 Linux 镜像中。您可以设置不同的器件模式选项，如大容量存储、同步、或基于中断的 USB。对于高速端点，批量传输的最大数据包大小为 512 字节，同步器件中为 1024 字节。用于流媒体应用可以使用同步模式，如视频。

以太网外设

千兆以太网 MAC (GEM) 使用了 10/100/1000 Mb/s 的兼容 IEEE802.3-2008 标准以太网 MAC。它能够以三种速度在任一半模式或全双工模式下运行。您可以独立配置每个 GEM 控制器。为了节约引脚，每个控制器通过 MIO 使用 RGMII 接口。通过提供 GMII 接口的 EMIO 访问 PL。

您可以在 PL 上使用 EMIO 接口的 GMII 创建其它以太网通信接口。

因为 Zynq-7000 AP SoC GEM 控制器只提供 MAC 功能，所以您必须使用一个独立的 PHY。

数据流

控制器中的 DMA 模块使用 AHB 总线从系统储存器中获取数据，并将数据存储在发送 FIFO中。取决于用户选择的 MIO 或 EMIO 接口， FIFO 数据由控制器转换为以太网协议格式，并且使用 RGMII 或 GMII 接口发送到以太网 PHY。

使用寄存器来配置 MAC 的功能，选择不同的操作模式，并提供和监控网络管理统计

控制器为 PHY 管理提供 MDIO 接口。您可以从 MDIO 接口控制 PHY。

系统级考虑事项

GEM 控制器提供了若干可根据系统需求实现的系统级功能。例如，您可以在任何需要更好的网络效率和更低 CPU 利用率的系统开展校验卸载。另一个系统级的考虑是 IEEE 1588 定义的精确时间协议 (PTP) 的实施。控制器能够检测和并相应 PTP 事件响应，这需要一个与 PTP 兼容的网络节点。

您还可以使用 DMA 接口从控制器到 PL 布线以太网数据包。此选项可能需要在 PL 中实现包加速或包检查的 IP。

GEM 控制器不支持巨型帧。如果需要支持巨型帧，您可以在 PL 中实现赛灵思 AXI 以太网 MAC IP。您可以使用 AXI DMA IP 在 PS DDR 储存器读写以太网帧。GEM 控制器不支持 3.3V I/O 电压。如需了解更多信息，请参阅：Zynq-7000 All Programmable SoC 技术参考手册 (UG585) [参照 4] 中的“千兆以太网控制器”章节 （链接）。

SDIO 外设

SD/SDIO 控制器与 SDIO 器件和 SD 存储卡通信。您可以通过 MIO 多路复用器 将 SDIO 接口导向 MIO 引脚，或在 PL 中通过 EMIO 导向 SelectIO 引脚。控制器可在大量便携式低功耗应用如 802.11器件、 GPS、 WiMAX 中支持 SD 和 SDIO 应用。

SD/SDIO 控制器与使用 SDMA（单项运行 DMA），ADMA1（4 KB 边界有限 DMA）,具备 ADMA2 支持的标准 SD 主控制器规格 2.0 版 A2 部分兼容。 ADMA2 允许数据在 32 位的系统储存器任何地点，以任何大小使用分散 - 聚集 DMA 传输。内核在 SD1 和 SD4 中还支持多达七种功能，但不支持 SPI 模式。内核不支持 SD 高速 (SDHS) 和大容量 SD (SDHC) 卡标准。您应熟悉掌握 SD2.0/SDIO2.0 规格 [参照 89]。

SD/SDIO 控制器通过 AHB 总线由 ARM 处理器访问。控制器还包括一个 DMA 单元与一个内部 FIFO 以满足吞吐量要求。 SD/SDIO 控制器符合 MMC 3.31 规格。

数据流

SD/SDIO 控制器具有内置的 DMA，可以使用 AHB 总线从系统储存器读取数据。控制器具有一对 FIFO 读写数据到连接卡以最大化吞吐量。数据被写入到连接卡中或由 APU 基于特定 SD 命令进行读回。

系统级考虑事项

SD 卡本质上是配备了闪存转换层 (FTL) 内置控制器的 NAND 闪存器件。 FTL 处理 ECC、模块管理与损耗水平，使储存器具备模块器件一样的行为。出于该原因，常规的文件系统 （例如 FAT， EXT2，和 ext3）都可以实现。由于多芯片封装技术和多级单元技术的进步，在现有 Zynq-7000 AP SoC 闪存选择中 SD 储存器件能够提供最高密度和最大容量。SD 存储器的缺点是对机械 SD 卡连接器的要求。当一款物理连接器不尽人意时，另一种选择就是 eMMC。 eMMC 包含闪存和封装在可直接安装在电路板上小球栅阵列 (BGA) 中的控制器，无需机SoC械连接器。SD 和 eMMC 的解决方案在他们充当启动器件的能力上有所不同。Zynq-7000 AP SoC 可以从 SD 卡直接启动，但它不能从 eMMC 启动。eMMC 的解决方案需要额外的启动器件，例如 QSPI。

除了 CLK， CMD 和数据信号，检测卡 (CDn) 信号显示 SD 卡的插入或存在，写保护 (WPn) 信号显示写保护开关在存储卡上的位置。有些软件驱动依赖于使用这些信号。因此，如果信号不可用，一个简单的解决方案就锁定激活 Cdn 与 Wpn 无效状态。

您还可以使用 SD/SDIO 外设作为启动模式器件。如需了解更多信息，请参阅：Zynq-7000 All Programmable SoC 技术参考手册 (UG585) [参照 4] 中的第 44 页的“配置和启动器件” 与“SD/SDIO 控制器”章节 （链接）。

UART 外设

UART 控制器是一个全双工异步接收器和发射器，支持多种可编程波特率和 I/O 信号格式。控制器支持自动校验生成和多主控检测模式。

由配置和模式寄存器控制 UART 的操作。 FIFO 的状态、调制解调器信号和其它控制器功能的读取通过状态、中断状态和调制解调器的状态寄存器实现。

数据流

控制器和 APU 之间的通信在 APB 总线中实现。所述控制器利用单独的 Rx 和 Tx 数据路径构建而成。每个路径都包括一个 64 字节的 FIFO。从系统储存器到达 APB 总线的数据被存储到发送 FIFO。从 MIO/EMIO 接口接收的数据存储域接收 FIFO，然后使用 APB 总线传输到系统储存器。

控制器在 Rx 和 Tx FIFO 上序列化与反序列化数据，包括支持回送配置 RxD 和 TxD 信号的模式开关。FIFO 中断状态位支持轮询或中断驱动的处理程序。软件利用 Rx 和 Tx 数据端口寄存器读取和写入数据字节。

系统级考虑事项

Zynq-7000 AP SoC 包括通常用作用于嵌入式系统调试端口的两个 UART 控制器。他们还常用于为主计算机提供终端连接。因为它们的实用效能，它们获得了所有层级的软件支持：FSBL、 U-Boot、单机运行与各种操作系统。

FSBL 在启动流程早期阶段使用 UART 将信息发送到终端。连接 UART 控制器最简单的方式需要两个信号，TX 和 RX。即使在使用具有两个测试点 TX 和 RX 的 PMOD 时，仍然建议用户访问 UART1 进行调试。

由于其低带宽、高电压摆幅和大型连接器等特点，大部分现代计算机不再搭载 UART 端口，而是使用 USB 作为标准外设总线。所以，可能有必要在电路板上增加一个 USB-to-UART 桥接控制器，使主 USB 端口与 Zynq-7000 AP SoC 的 UART 端口匹配连接。如图 3-5 所示。主 PC 终端软件将桥接识别为虚拟 COM 端口，同时可采用任何广泛使用的终端程序与 Zynq-7000 AP SoC 通信。

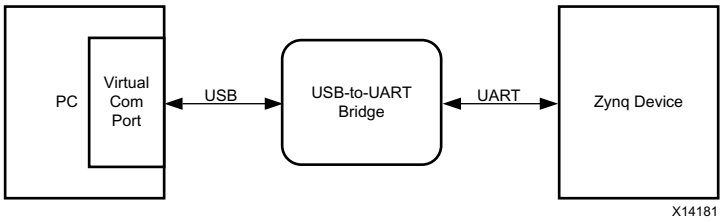

图 3-5 : 通过 USB-to-UART 桥接连接 Zynq-7000 AP SoC 的 UART 到 PC

图 3-6 是一个使用 Cypress USB-to-UART 桥接器件实现主计算机和 Zynq-7000 AP SoC 之间连接的示例。在最简单的设计实现中，仅需要 UART 控制器的 TxD 和 RxD 信号来支持终端。如果需要流量控制，可以使用经扩展的 MIO 来添加。Cypress 提供了一个免版税虚拟 COM 端口 (VCP) 驱动，允许 CY7C64225 USB-to-UART 桥接器以 COM 端口的形式显现在主计算机的通讯软件中，如 TeraTerm 或 HyperTerm。

表 3-2 : CY7C6 连接

EPP 引脚	EPP 中的 UART 功能	原理图网络名称	CY7C6 引脚	CY7C64225 中的 UART 功能
D11 (MIO Bank 1/501)	TX，数据输出	USB_1_RXD	23	RXD，数据输入
C15 (MIO Bank 1/501)	RX，数据输入	USB_1_TXD	4	TXD，数据输出

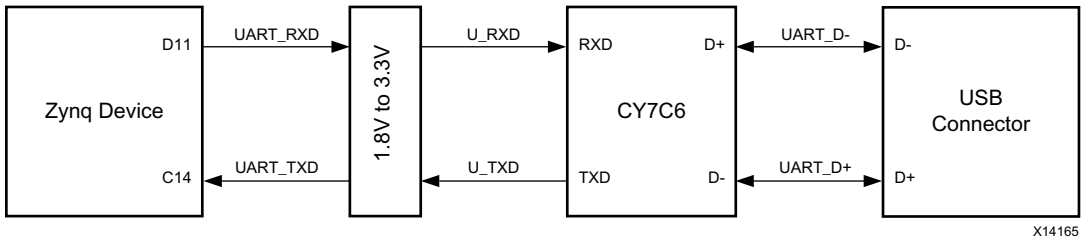

图 3-6 : 采用 Cypress USB-to-UART 桥接方式

一种替代实现方案使用了 Silicon Labs 的 CP2103GM USB-to-UART 桥接器件。如需了解更多信息，请参阅：《Zynq-7000 XC7Z020 All Programmable SoC 的用户指南》(UG850) [参照 8] 中的“ZC702 评估板”。

只有两引脚的 UART 可通过 MIO 布线。一个完整的 8 引脚 UART 需要通过 EMIO 布线。它支持可编程波特率，以及协议和奇偶校验位。如需了解关于 UART 控制器的详情，请参阅：《Zynq-7000 All Programmable SoC 技术参考手册》(UG585) [参照 4] 中的“UART 控制器”章节 （链接）。

CAN 外设

PS 中的 CAN 控制器是映射储存器的外设，支持高达 1 Mb/s的比特率。控制器实现了具备 64 消息存储容量的发送和接收 FIFO。控制器实现了接收信息的 16 位时间戳，并提供接收和发送错误计数器。

数据流

CAN 控制器的 APB 接口连接外设到中央互联，您可以用它来配置控制寄存器。可以将 CAN 发送和接收信号布线到 MIO 或通过 PL 布线至 EMIO。 CAN 控制器支持五种操作模式：

- 配置模式
- 正常模式
- 休眠模式
- 回送模式
- 嗅探模式

您可以使用连接到控制器的 APB 总线，通过寄存器的写操作初始化这些模式。如需了解更多信息，请参阅：Zynq-7000 All Programmable SoC 技术参考手册 (UG585) [参照 4] 中的“CAN 控制器”章节 （链接）。

I2C 外设

I2C （内部集成电路）是使用两个双向漏极开路的多主串行单端总线，串行数据 (SDA) 和串行时钟 (SCL) 均使用电阻上拉。协议包括一个从地址、从器件内的一个可选寄存器地址，以及每字节的 ACK/NACK 比特。 I2C 用于连接如传感器、EEPROM、I/O 扩展器、可编程时钟器件，或 A/D 和 D/A 转换器的低速外设到嵌入式系统。一些总线的实现由 I2C 总线分出，包括系统管理总线 (SMBus)，电源管理总线 (PMBus的)，以及智能平台管理接口 (IPMI)。

Zynq-7000 AP SoC 包括两个 I2C 控制器，可以在 I2C 总线上以 100 Kb/s（标准模式）和 400 Kb/s（快速模式）的常见速度运行。每个控制器可以作为主控或在多主控设计中作为从控功能。主控可以编程为使用正常 （7 位）寻址和扩展（10 位）寻址模式。主控负责生成时钟和控制数据传输。数据可以通过主控和从控模式配置进行传送或接收。

数据流

在从控监控模式中， I2C 接口可以被设置为监视从控忙碌状态。在这种模式下， I2C 将不断尝试传输到一个特定的从器件，直到从器件出现一个 ACK 响应。

在从控模式下，通过检测第一个地址字节位 [7:3] 的特定代码自动确定扩展地址支持。您可以设置 HOLD 位以防止主控继续传输，防止从控溢出情况。但是，您必须在 I2C 控制器发生超时之前清除 HOLD 位。您可以设定不同的超时值。

您可以使用 APB 从接口编程 I2C 控制寄存器。 I2C 接口的特定 SCL 和 SDA 信号可以通过 EMIO 导向 PL，或者导向 MIO。如果通过 EMIO 布线，SCL 和 SDA 信号经常通过三态 I/O 缓存与连接到 PL I/O 组之一的 I2C 器件进行通信实现。控制器引发完成中断，向主机指示传输完成。在主控模式下，任何由控制器收到的 NACK 都将通过 NACK 中断传递给主机。如需了解更多信息，请参阅：Zynq-7000 All Programmable SoC 技术参考手册 (UG585) [参照 4] 中的“I2C控制器”章节 （链接）。

系统级考虑事项

上拉电阻应布置在 SCL 和 SDA 线上，距离 Zynq-7000 AP SoC 最远的地方。如需了解更多上拉电阻的示例计算，请参阅：“稳健 I2C 通信设计计算” [参照 76]。典型电压为 +5 V 或 +3.3 V。取决于 I2C 器件的电压和所使用的 Zynq-7000 AP SoC I/O 组，可能需要使用一个电平转换器/中继器可能是。从 SDA 到 SCL 的 PCB 和封装时延偏差应小于 ±500 ps。

并非所有的 I2C 从器件都具有可编程地址，所以具有相同地址的多个器件连接到单个总线时，可能会发生地址冲突。出于这个原因，一个 I2C 适配器端口通常通过 I2C 总线开关布线，来连接到在不同的总线段具有相同地址的 I2C 从器件。应用必须首先设置地址，在与一个 I2C 总线端上的从器件通讯之前，应配置总线开关来选择所需的 I2C 通道。如果 I2C 地址不同，一个总线段的电容低于允许值，那么每个 I2C 总线段可以有多于一个的器件。第 58 页的图 3-7 展示了包含一个 1:8 总线开关的 I2C 总线拓扑示例。如需了解更多信息，请参阅：具有复位功能的 PCA9548A 低压 8 通道 I2C 开关 [参照 88]。

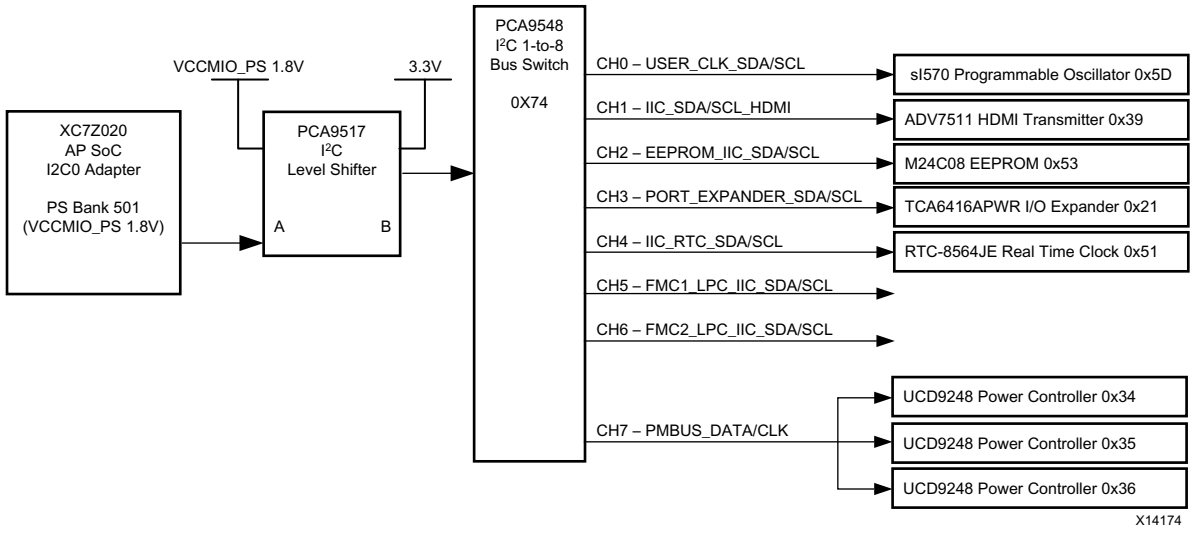

图 3-7 : I2C 总线拓扑例

单机、U-Boot 和 Linux 驱动可用来运行 I2C 控制器。在 FSBL 内，单机驱动可用于外设的早期初始化。所有常见的传输模式都支持主从模式。如需了解更多有关 Linux 驱动的信息，以及简单应用示例，请参阅：赛灵思 Linux I2C 驱动维基页 [参照 52]。

SPI 外设

SPI （串行外设接口）是支持全双工模式的同步串行总线。它采用四线：SCLK （串行时钟）， MISO （主输入，从输出）， MOSI （主输出，从输入）和 SS （从选择）。在主/从模式下，器件在主器件初始化数据传输的地方进行通信。在与从器件通信时，主器件不使用寻址机制。允许多个从器件存在，使用典型低电平有效的单独从选择线路。 SPI 通常用来与 A/D 和 D/A 转换器、闪存和 EEPROM 储存器、LCD和许多其他外设进行通话。JTAG 标准实质上是一个三线 SPI 协议的应用栈。

Zynq-7000 AP SoC 中包括 2 个 SPI 控制器，可以在主、从或多主器件模式下运行。控制器总是在全双工模式下工作，同时接收和传输数据。

数据流

在主模式下，使用单独的从选择信号，可以最多确定三个从器件。在板上添加一个 3 至 8 解码器的外设可以定位多达八个从器件。当使用 3 至 8 解码器选项时，您可以用软件来控制解码器的三个输出引脚。当器件存在单个双向 SPI 数据引脚，软件可以使用此特性来控制外部三态。

在从模式下，控制器从外部主器件接收消息并同时发送一个答复。控制器使用 32 位寄存器映射数据端口寄存器，通过 128 字节的 TX/RX FIFO 读取并写入从器件。 FIFO 的读写通过 APB 从接口在 SPI I/O 接口和控制器软件之间提供缓冲。FIFO 用于从模式和主 I/O 模式。如需了解更多信息，请参阅：《Zynq-7000 All Programmable SoC 技术参考手册》(UG585) [参照 4] 中的“SPI 控制器”章节 （链接）。

注意 ！ 所有 SPI 事务都必须字节对齐 （8 位的倍数）。不支持实现非字节对齐 SPI 事件的从器件，例如 10 位地址和 16 位数据。

系统级考虑事项

两种 SPI 接口， SPI[0,1] 可布线至 MIO 引脚或通过 EMIO 布线至 PL。如果通过 EMIO 布线，最常用的方法是使用三态 I/O 缓存与连接到 PL I/O 组之一的 SPI 器件进行通信（如果这样做，在 PL 编程之前， SPI 接口不可用）。主模式下使用内核时，从选择信号 SS[1,2] 可以有选择地启用。这些信号是真实的输出，因为仅当其内核配置为主器件时才可用。因为 SS0 信号也是在从模式中使用，所以它属于三态。当 I/O 信号布线至 MIO 引脚时， SCLK 时钟运行频率可高达 50 MHz。当 I/O 信号通过 EMIO 布线， SCLK 时钟运行频率可高达 25 MHz。

图 3-8 显示了 SPI 控制器为主模式配置并通过 MIO 布线。SS0 必须通过 EMIO 引脚进行布线。在此配置中，当使用 SS0 时，它需要使用一个上拉电阻来拉高并且不能被外部器件驱动为低电平。使用相应的从选线 (SSn)，最多可以同时连接并选择三个从器件。如果少于三个从连接时，可以使用任何 SS 信号。如果不使用 SS0，它必须被拉到 Vcc 并且不用于任何其他目的。

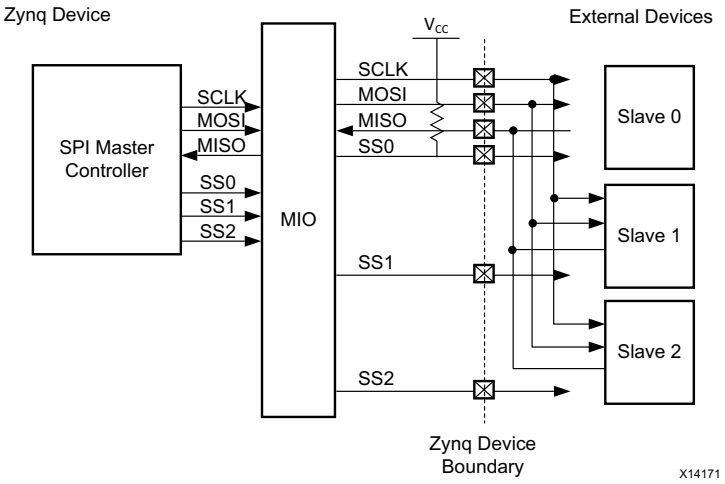

图 3-8 : 以主模式实现到 SPI 控制器的 SPI 从连接

当通过 EMIO 布线时， EMIOSPIxSSON0 输出信号连接到 PL 从器件， EMIOSPIxSSIN 输入信号连接到到 Vcc。将 SS0 置于高电位很重要，因为在主模式下控制器能嗅探这个信号来检测多主方式的情况。如果 SS0 低，控制器认定为多主模式并发出 Mode_Fail 中断。不建议采用多主模式。如需了解有关驱动SPI 控制信号的详情，请参阅：赛灵思解答记录 47511 [参照 66]。

图 3-9 显示了 SPI 控制器为从模式配置并通过 MIO 布线。从选择行 SS0 用于将外部主器件连接到从控制器。根据主器件的功能，其他外部从器件可实现片外连接。

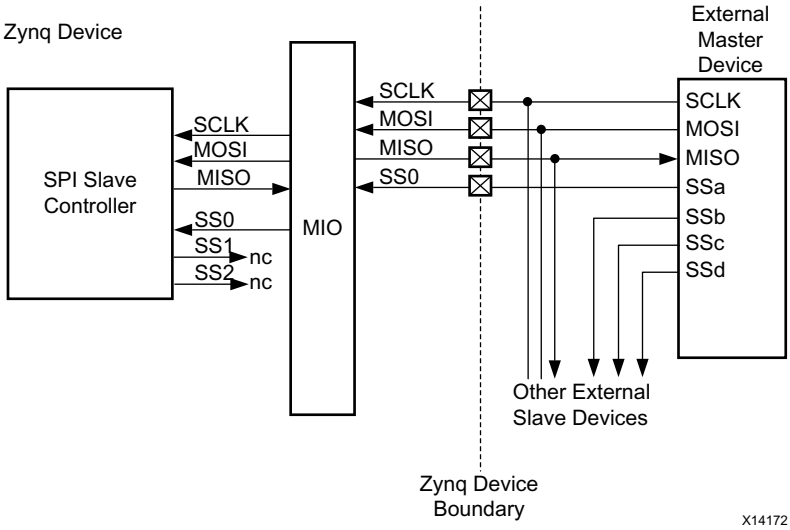

图 3-9 : 以从模式实现到 SPI 控制器的 SPI 主连接

建议在 SCLK、 MISO、 MOSI 和 SS 线都采用匹配长度，这有助于满足设置和保持时间。 PCB 和相对 SCLK 的 MISO、MOSI 和 SS 封装时延偏差应小于 ±50 ps。

单机，U-Boot 和 Linux 驱动可用于运行 SPI 控制器。如果控制器通过 MIO 布线，那么单机驱动可以在 FSBL 内用于早期外设初始化。如果控制器是通过 EMIO 布线，位文件下载后初始化才能进行。所有常见的传输模式都支持主从模式。如需了解更多有关 Linux 驱动信息,以及简单应用示例，请参阅：赛灵思 Linux SPI 驱动维基页 [参照 53]。

GPIO 外设

GPIO （通用输入/输出）可以在运行时被用户编程，包括输入和输出引脚。 GPIO 引脚通常用于连接 LED、 DIP 开关和按钮，或连接板外设的中断和复位信号。通过 EMIO 使用 GPIO 为控制 PL 复位提供了一个解决方案，不再需要担心控制 FCLK_RST 的副作用。它还通过 EMIO 接口为 PL 64 输入提供了访问，包括， 64 真实输出和 64 输出使能。如果不需要输出使能，您可以通过控制方向寄存器将输出使能作为额外的 64 输出使用。每个 GPIO 都独立并动态编程为输入、输出或中断感应。当配置为感应中断，可以将它设置为电平敏感 （高或低）或边沿敏感 （正、负、或两者）。

系统级考虑事项

GPIO 控制器有四个 Bank。 Bank 0 有 32 个引脚， Bank 1 有 24 个引脚。两个 Bank 中共 54 个 GPIO 引脚专为 MIO 所用。这些引脚是三态的，可以配置为输入或输出。 Bank 2 和 3 各具有 32 个引脚，通过 EMIO 将 64 个 GPIO 引脚连接到 PL。这些 GPIO 包含三种信号：输入、输出和输出使能。三态缓存可在器件的边界例化，但是， PS 与 PL 连接是简单的导线。

每个 GPIO 都独立并动态编程为输入、输出或中断感应。当配置为感应中断，可以将它设置为电平敏感 （高或低）或边沿敏感 （正、负、或两者）。软件可以使用单个加载指令读取 Bank 之内的所有 GPIO 值，或者使用单个存储指令将数据写入到一个或多个 GPIO （整个范围的 GPIO 内）。

注意 ！ MIO 引脚 [8：7] 只能用于输出。 GPIO 通道 7 和 8 只能配置为输出。

I/O 标准和板上外设的电压需要匹配 MIO 或 PL I/O 引脚配置。

单机和 Linux 驱动可用于运行 GPIO 控制器。在 FSBL 内，单机驱动可用于外设的早期初始化。如需了解更多有关 Linux 驱动的信息，以及简单应用示例，请参阅：赛灵思 Linux GPIO 驱动维基页 [参照 51]。

Cortex-A9 多处理外设

Cortex-A9 多处理外设包括 SWDT 和具备自动递减特性的 TTC，它们可以用作通用定时器。这些定时器作为一种从待机模式下启动处理器的机制。

系统级控制寄存器 (SLCR) 作为 APU 和外部 PL 主器件的外设。您只能在主器件安全模式下请求寄存器访问，从而实现 SLCR 访问。 SLCR 包括用于配置各种时钟、复位和外设的安全设置的寄存器。

PS DMA 控制器

可以使用 PS DMA 控制器在 PS-PL 边界将数据从任何 PS 或 PL 外设传输到 DDR、OCM、线性 QSPI SMC、PL 外设，或连接 M_AXI_GP 端口的线性寻址储存器。

您可以在需要硬件协同处理器加速的应用中使用 DMA 控制器，通过 PL 算法用来处理 Cortex-A9 写入到 DDR 储存器的数据。在数据由 PS 和 PL 外设处理的应用中， DMA 控制器起着重要的作用。

您可以使用 DMA 控制器节省系统功耗。 CPU 能够进入低功耗模式，并且通过 PS DMA 接口实现批量数据传输。因为 CPU 时钟比 DMA 时钟快得多，这种方法节省了大量的动态功耗。如需了解更多信息，请参阅：Zynq-7000 All Programmable SoC 技术参考手册 (UG585) [参照 4] 中的“DMA 控制器”章节。

XADC

XADC 包含两个具有独立跟踪保持能力的放大器和 12 位 1-MSPS ADC、一个片上模拟多路复用器，以及片上温度和电源传感器。两个 ADC 可以配置为同时采样两个外部模拟输入通道。跟踪和保持放大器支持各种模拟输入信号类型包括单极、双极和差分。外部输入包括一对专用差分模拟输入，可以支持模拟输入也可以用作数字 I/O (Vauxn/P) 的 (Vp/Vn) 和十六个多功能引脚。专用模拟输入可以在 1MSPS 采样速率下至少支持 500KHz 的信号带宽，或在辅助通道支持 250KHz。

XADC 还包括一些支持测量片上电源电压与核心温度的传感器。从任何来源的 ADC 转换的结果 （温度、电源电压，或模拟输入通道）都存储于一组 XADC 模块内部状态寄存器。因为温度会变化并且时钟是可编程的，所以当温度发生变化时，您可以调整 Zynq-7000 AP SoC 的时钟。

XADC 模块还包括一组用于配置和控制 XADC 运行的控制寄存器。这包含用于指定内部测量传感器 （温度和电压）自动告警阈值的告警寄存器。当温度或电压下降到寄存器定义的可接受的范围之外时，告警会自动触发一个 PS 中断。

XADC 模拟输入可用于支持电动机控制、触摸传感器，或需要一个模拟前端的许多其他应用。电压和温度感应能力可以用于监控系统运行状况，当系统处于异常温度或电压条件时进行适当响应。

有两个应用指南介绍用于系统监控的 XADC 和对外部模拟信号进行的 ADC 操作：

- 《在赛灵思模数转换器 (XADC) 专用接口使用 Zynq-7000 处理系统 (PS) 实现系统监控和外部通道测量》(XAPP1172) [参照 39]
- 《使用具有 XADC AXI 接口的 Zynq-7000 AP SoC 处理系统进行系统监控》(XAPP1182) [参照 41]

如图第 62 页的图 3-10 所示，可使用并行接口 DR P或串行 JTAG-DRP 接口访问 XADC 状态和控制寄存器。LogiCORE™ IP 可用于封装包含必要逻辑的整个 XADC 硬模块，将 XADC 转换成可连接到 PS 主 AXI_GP 端口的 32位 AXI 从外设。必须采用该解决方案才能实现 PS 1Mbps 的数据访问速率。虽然AXI XADC LogiCORE IP 的实现需要消耗 PL 资源，但它提供了一种快速、整洁的 XADC 接口。如 《使用具有 XADC AXI 接口的 Zynq-7000 AP SoC 处理系统进行系统监控》(XAPP1182) [参照 41] 中所述，它具备并行数据路径的完全优势，是可以实现全 1MSPS 速率的唯一方法。

另一种解决方案是使用内置的 PS_XADC 接口模块。其优点在于，它不需要额外的 PL 逻辑 （在FPGA 使用内置功能时不必进行配置）。 XADC DRP 寄存器的读写需要 XADCIF_CMDFIFO 的适当写命令和来自 XADCIF_RDFIFO 的适当读命令。但是，因为模块通过DRP JTAG 接口序列化 FIFO 内容，并且以一次一位的速度进行 XADC 硬模块的数据转移，这种解决方案更慢，而且跟不上较高的数据速率。如 《在赛灵思模数转换器 (XADC) 专用接口使用 Zynq-7000 处理系统 (PS) 实现系统监控和外部通道测量》(XAPP1172) [参照 39] 所述，此接口最大仅能支持 100KHz。此外，PL-JTAG 接口和内部的 PS-XADC 接口不能同时使用。

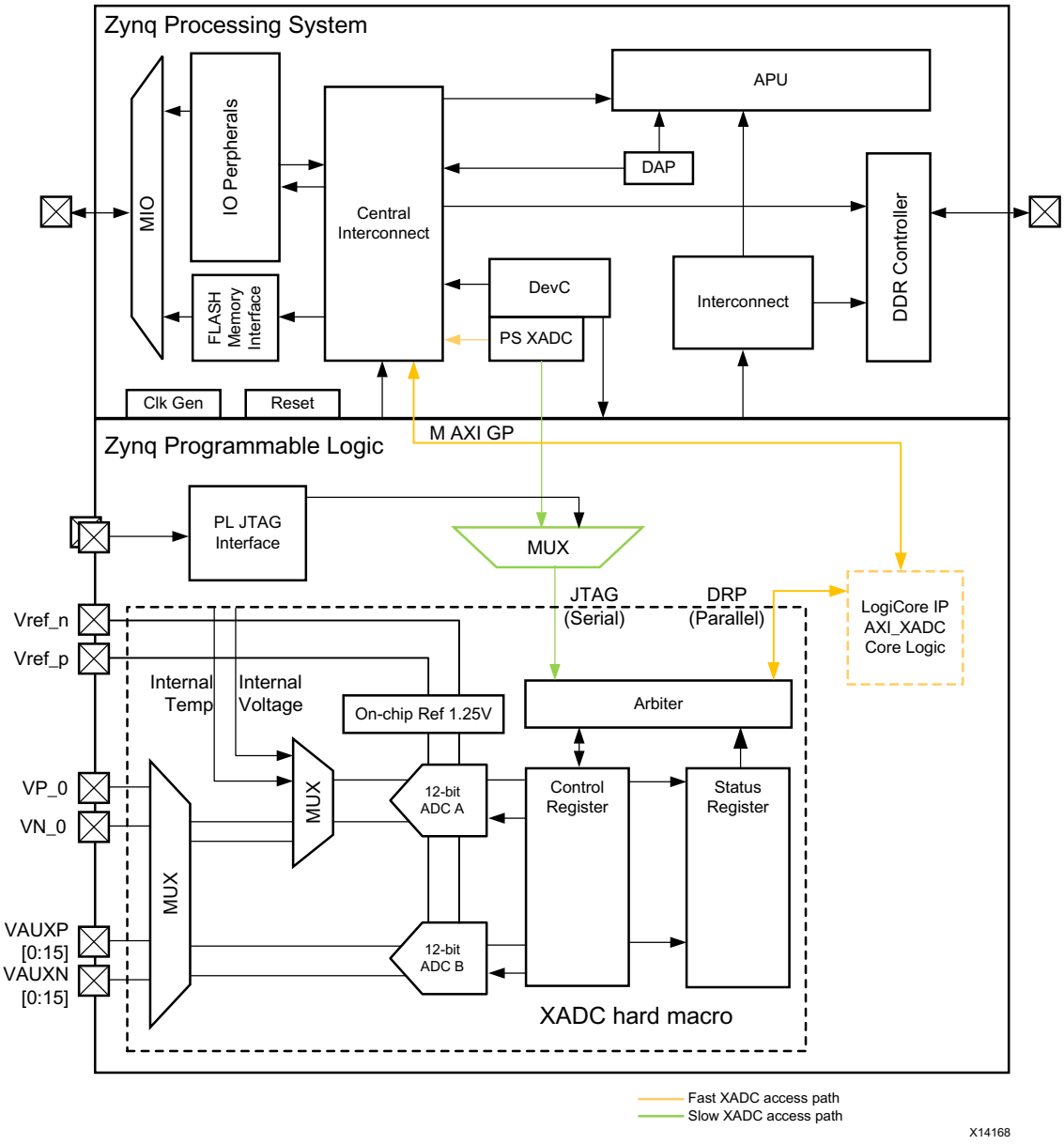

图 3-10 : XADC 模块原理图

与外设之间的通信

上面所讨论的外设实行特定 AMBA 的总线结构与 APU 和储存器系统进行通信。USB、GEM、SDIO 和 SPI外设有两种类型的通信接口：

- APB 接口将外设从器件连接到 APU 主器件。接口利用中央互联通信。
- AHB 接口在嵌入外设的 DMA 控制器与系统储存器之间启动高速总线事务。接口利用中央互联连接到系统储存器。

图 3-11 显示了外设总线连接到主储存器和 APU。图 3-11 是实现 DMA 和控制寄存器集的外设通用框图。

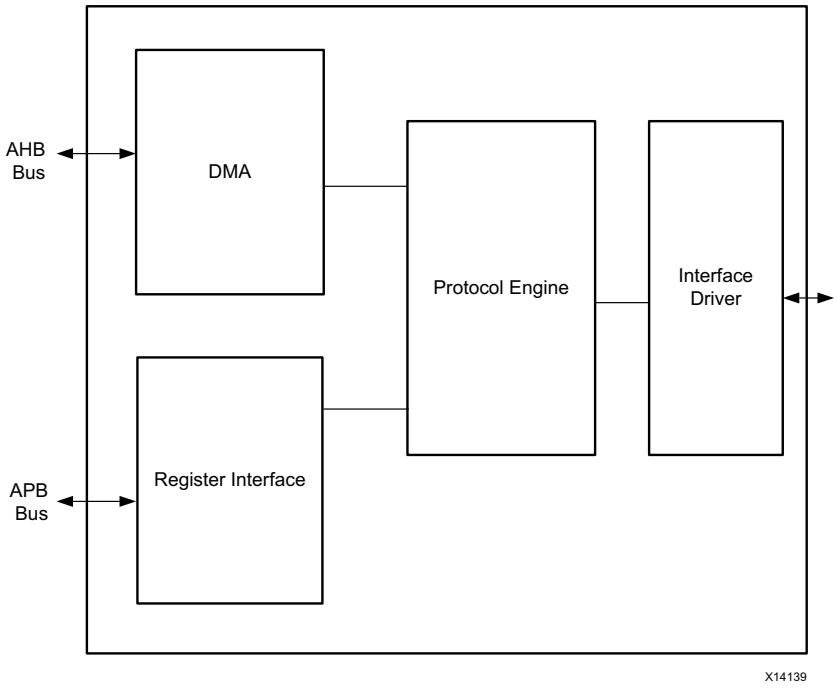

图 3-11 : 外设通信总线

外设设计示例

这个示例展示了外设设计时的考虑因素。在示例中使用 Zynq-7000 AP SoC 的 GEM 控制器和两个不同的物理接口。一个控制器 RGMII PHY 使用 MIO 接口，另一个控制器的 1000BASE-X PHY 使用 EMIO 接口。您可以选择任何支持的接口来满足您的要求。

1. PS 有 GEM 控制器的两个实例。一个使用 RGMII 接口，另一个使用 1000BASE-X 接口。
2. Cortex-A9 多核心外设具备 TrustZone 感知的安全访问。
3. 您可以使能和禁用 GEM 硬件的 TCP 校验和卸载，请注意两个选项的 CPU 利用率和性能的差异。

图 3-12 显示了硬件结构框图。

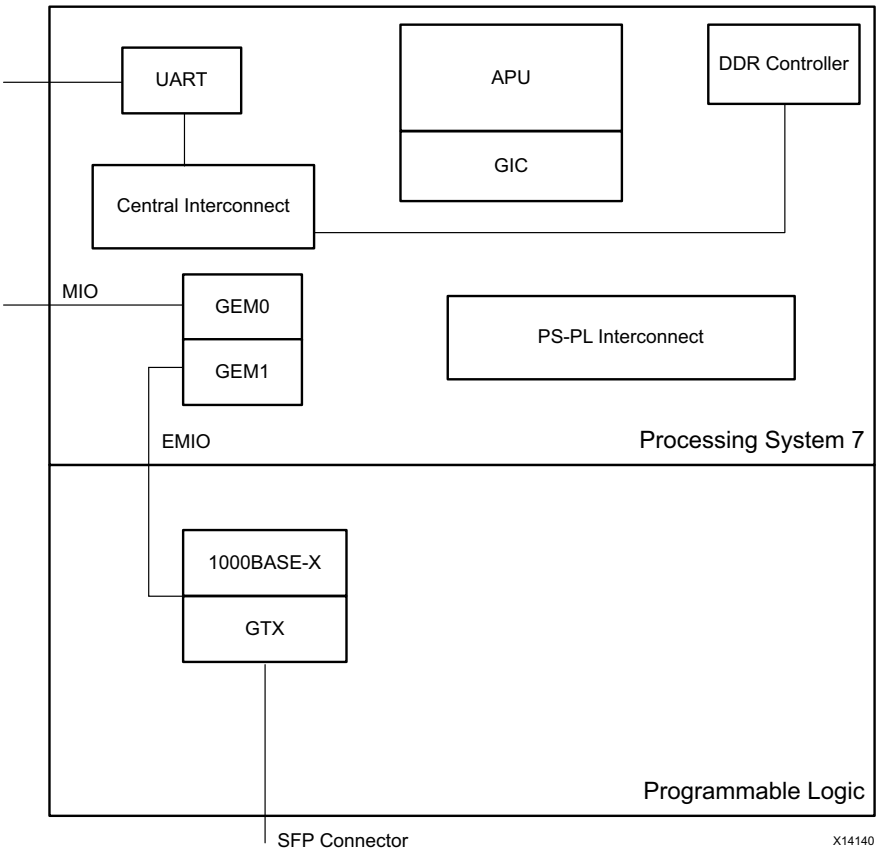

图 3-12 : GEM 例化

虽然这个示例是围绕千兆以太网外设构建的，您可以将概念扩展应用到其他 PS 外设。

硬件设计考虑事项

您可以使用 Vivado® IP 集成器创建硬件设计。您必须使能 GEM0 的 MIO 连接和 GEM1 的 EMIO 连接。使能 GEM1 的 EMIO 将发送和接收的 GMII 信号送达 PS 的 IP 顶层实例。用户必须在 IP 集成器中例化赛灵思 1000BASE-X IP 核，并将其连接到 GEM1 EMIO 端口。您可以使用 Vivado Design Suite 设计实现流程来生成码流。如需了解更多信息，请参阅：《Zynq-7000 AP SoC 中通过 PL 以太网实现 PS 与 PL 以太网性能和巨型帧支持》(XAPP1082) [参照 38]。

软件设计考虑事项

GEM 的外设支持卸载 TCP 校验和至硬件来提高以太网性能，并通过硬件内执行计算密集型校验和计算来提高 CPU 的利用率。用户可以通过 APB 接口写入 GEM 配置寄存器使能校验和卸载。

GEM 驱动配置 GEM DMA 特定传输尺寸。驱动还设置了描述符环、DMA 分配和回收，以及源与目的 MAC 地址的编程。

在通过写入网络控制寄存器使能发射器和接收器之后， DMA 传输开始。

设计 IP 模块：

预验证的知识产权 (IP) 可用于缩短上市时间，并且使能 SoC 上功能丰富的数字和模拟电路。这些 IP 模块包括嵌入式 CPU、储存器模块、接口模块、模拟模块，以及应对特定应用处理功能的组件。使用例如如 AMBA 高速规格和 AXI 规格的标准接口规格，可以提高 IP 模块的复用性。如需了解选定 Zynq-7000 AP SoC 器件支持 IP 模块的详细信息，请参阅：Vivado IP 目录。

IP 模块主要分两大类：

- 软 IP 模块：您可以在 FPGA 结构中实现这些模块，使用 RT L或更高级别的说明。它们更适用于数字核，因为硬件描述语言 (HDL) 与进程无关，可以综合到门级。 HDL 具有灵活性、可移植性和可复用性优势，但是具有不能保证时序或功耗特性的缺点。
- PS IP 模块：这些模块具有固定的布局并且针对特定应用和进程进行了优化。它们的主要优点是具有可预测的性能，但是需要着额外的工作和成本投入，而且缺乏便携性，这可能大大地限制了其应用范围。 PS IP 模块通常通过了预审，这意味着供应商已经对芯片进行了测试。这大大增加了其正确性的保障。

SoC 设计的固有复杂性导致了处理模块周围的更多、更复杂的接口的使用。 Zynq-7000 AP SoC 上的高速串行收发器是一个典型示例。收发器的高速串行 I/O 能够以数个 GHz 为单位传输数据。收发器模块能够跨多个收发器通道进行字节边界对齐、时钟数据恢复，以及偏差消除。针对使用独立发送和接送时钟并具有 ppm 变化的西戎，它还能够进行时钟补偿。用户逻辑通过复杂的接口连接到收发器，因为它必须控制收发器内的各种功能模块，所以功能逻辑复杂。

IP 模块立足用户界面，通过屏蔽底层复杂性，有着处理收发器配置的重要作用。赛灵思 Aurora IP 模块用户界面符合 AXI4 流标准，能够开展收发器初始化、时钟补偿、信道绑定，和字节边界对齐。用户无需了解底层的收发器的复杂性，就能构建基于 AXI4 流接口上的收发器的应用。

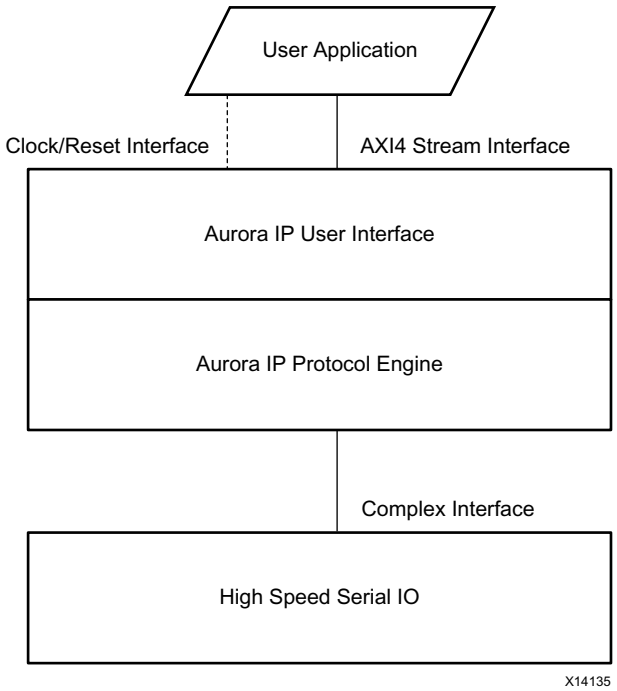

图 3-13 : Aurora IP 示例

图 3-13 展示了如何将 Aurora IP 插入一个 ZYNQ-7000 AP SoC 的收发器接口。

IP 核设计方法

赛灵思使不同的方法来创建 IP 模块，并提供一个易于维护的方法。为 Zynq-7000 AP SoC 和其它如 MicroBlaze™ 处理器的处理系统创建的嵌入式系统设计的 IP 模块能够实现复用。下面通过赛灵思工具链介绍了 IP 创建方法。

System Generator

System Generator 这种工具利用 FPGA 系统设计的 MathWorks Simulink，可以创建基于用户配置的 DSP IP 模块。为系统设计提供了高级别的基于模型的环境。在 System Generator 中，有大量构建模块都是可用，包括从简单的数学运算符到复杂的 DSP 运算。

在 Vivado Design Suite 中，IP 打包器编译目标，允许用户对 System Generator 生成的 IP 封装，并将其纳入 Vivado IP 目录中。您可以通过 IP 目录像其他的 IP 模块一样使用 System Generator 设计，并在设计中进行例化。

HDL Coder

HDL 编码器是一个 MathWorks 工具，可从 MATLAB 功能和 Simulink 模型生成综合的 HDL 代码。它提供了用于分析 MATLAB/Simulink 模型的工作流，并将其从浮点转换到定点模型，提供高级抽象。工作流还提供了验证码，允许使用原始的 MATLAB/Simulink 模型 HDL 代码对其进行测试。

并不是所有的 MATLAB 和 Simulink 功能都支持 HDL 的生成。由于这个原因，可能必须要对一些模型的功能进行修改，使其功能或模块支持 HDL 生成。

Vivado 高层次综合

Vivado 高层次综合 (HLS) 是一个由赛灵思提供的工具，它能够在赛灵思 FPGA 器件上将基于 C 语言的设计转换为 RTL 设计文件实现。 Vivado HLS 工具流程如图 3-14 所示。

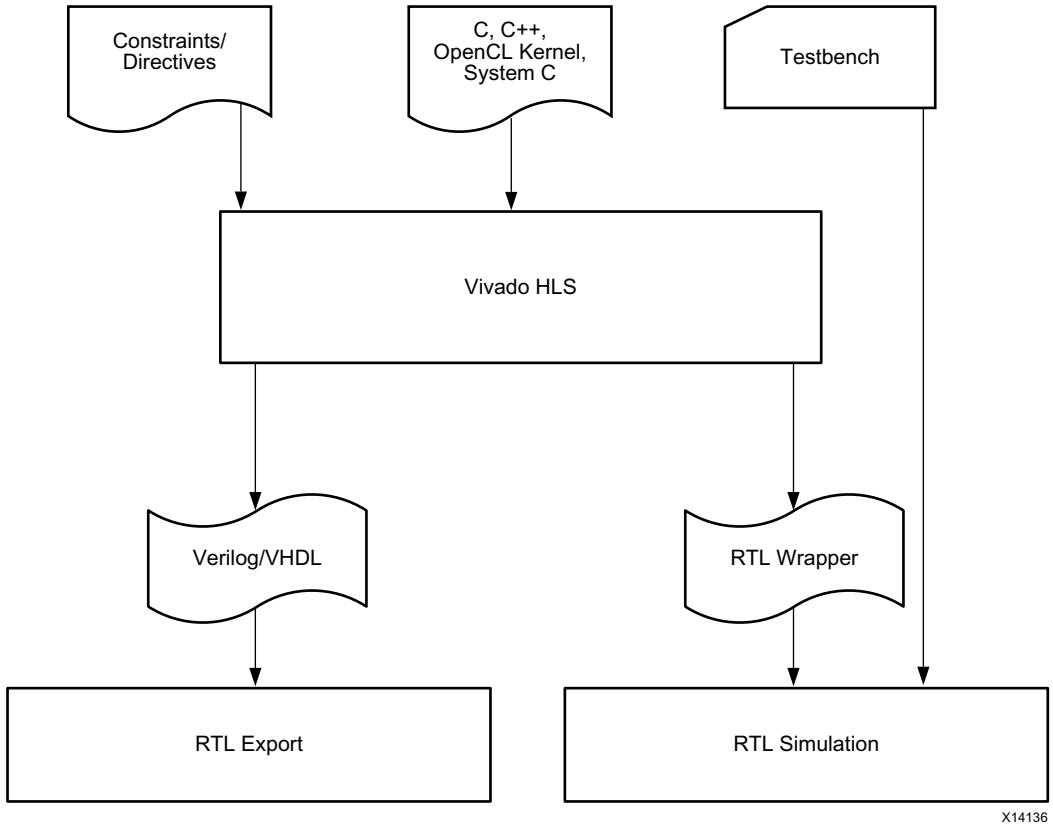

图 3-14 : Vivado HLS 流程

Vivado HLS 流程将 RTL 打包输出为以下格式，让您轻松集成输出到其他赛灵思设计工具：

- IP 目录：包含一个您可以添加到 Vivado IP 目录的 ZIP 文件。
- 合成的检查点 (.dcp)：一个您可以在 Vivado Design Suite 中直接加入到设计的 Vivado 检查点文件。
- DSP System Generator：您可以将此输出添加到 DSP 的 Vivado 版 System Generator 中。
- System Generator for DSP (ISE®)：您可以将此输出添加到 DSP 的 ISE 版 System Generator 中。
- EDK 的 Pcore：您可以将此输出添加到赛灵思 Platform Studio 中。

除了封装输出格式， RTL 文件可作为独立的文件 （不属于推荐的封装格式组成部分）。

IP 核设计考虑事项

在设计 IP 模块时，应考虑多个嵌入式系统设计和不同用户的配置选项中 IP 模块的复用性。在大多数系统级方案中，标准总线拓扑的参数化和实现允许 IP 模块进行复用。

在设计一个 IP 模块时，您应考虑以下几点：

- 参数：参数包括接口级的配置和 IP 特性的具体配置。当 IP 模块与其他 IP 模块连接时，接口级的配置是非常重要的。用户必须控制不同的总线参数，包括数据和地址总线宽度、时钟、复位接口配置等。在IP 模块中，启用或禁用 IP 特性具体配置的某种功能取决于用户的选择。

您可以使用多种方式处理 IP 模块的参数。参数化的基本水平可以在 HDL 代码中实现。HDL 参数可以先通过 Vivado IP 集成器封装 IP 模块，然后在 IP 配置向导中进行配置。您可以利用自定义的环境来参数化 IP 模块。

- 总线拓扑例：在设计 IP 模块时， IP 模块的复用性属于重要的考虑因素。您可以通过实现符合工业标准总线接口的总线拓扑来提高复用性。其中最广泛使用的标准是 AMBA，存在一大类符合总线规格的 IP 模块。使用业界标准接口的总线型拓扑结构在系统级提供了方便的即插即用和主从 IP 模块功能。这种 IP 模块还针对选择标准协议和接口层次结构的特定版本上提供了灵活性。

  总线接口的选择取决于系统级的需求。AXI4-Lite 总线主要用作 PS 控制总线， AXI4 储存器映射总线用作 IP 核间储存器传输。 AXI4-Stream 总线用于连接流体传输模块中没有地址关联的设计。下图显示了在基于 AXI DMA 的设计中的 AXI4-Lite、 AXI4 储存器映射和 AXI4-Stream 总线。在图中， AXI DMA 负责使用高性能端口对 PS 系统储存器和 AXI4-Stream 索贝尔 IP 之间开展数据传输。 PS 使用 AXI4-Lite 界面设置 AXI DMA 的缓存描述符。

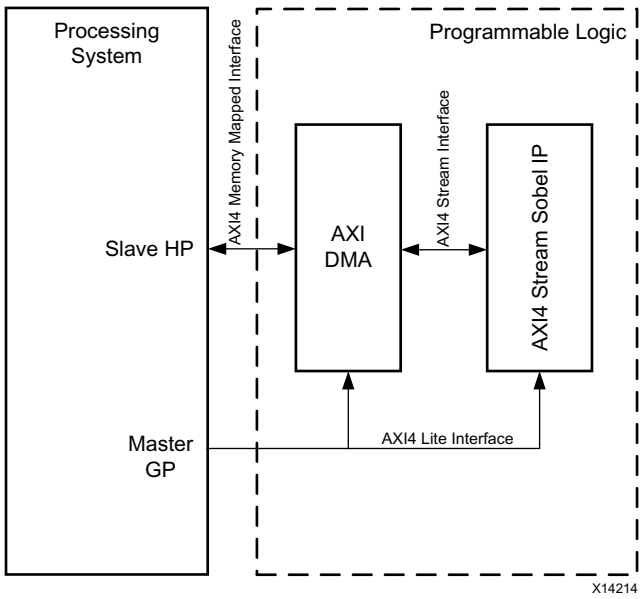

图 3-15 : AXI4-Lite、 AXI4 Memory-Mapped、和 AXI4-Stream 总线

- 安全性和文档：防止设计遭侵权是设计 IP 模块的重要考虑因素。可以采用各种安全算法来加密 HDL 文件。赛灵思工具允许您加密 HDL，并使用 AES 算法解密 HDL。

将 IP 模块连接到 PS

当 PS 和 PL 之间的接口采用标准的 AMBA 接口时，可以使用软 IP 模块实现设计。 PS-PL 接口有一个 GP 端口 （主和从）可以用于实现一个 AXI3 或 AXI4 接口，并可以将 PS 与软 IP 核寄存器接口连接。您可以在 PL 上将高性能 (HP) 端口用作 AXI3 或 AXI4 主 IP 模块驱动的从 AXI3 端口。您可以以类似的方式使用 ACP 接口。ACP 接口实行 AWAWCACHE 和 ARCACHE 信号，实施 ACP 主控制器的用户 IP 模块需要实现这些信号。如需了解更多信息，请参阅第 80 页的“ACP 和高速缓存一致性”。

ACP 接口连接 L2 高速缓存，并可以将此接口用于需要显著 CPU 利用率的情况,使 PL 中 IP 模块协同处理计算密集型操作。

您可以实现 AXI3/4 储存器映射或 AXI4 流协议。AXI3/4 储存器映射接口具有读/写地址信号和数据接口信号。AXI4 流接口没有地址信号。

当使用 IP 模块设计一个 Zynq AP SoC 时， PL 必须考虑以下几点：

1. IP 模块符合 AXI3/4协议规格。在 PS-PL 边界的 AXI 接口与 AXI3 协议兼容。您可以使用赛灵思提供的 AXI 互联 IP 模块将 AXI4 标准 IP 连接到 PS-PL 接口端口。

2. IP 模块是 IP 集成器资源库的一部分，能够简化将 IP 模块集成到系统级设计的工作。
3. PS-PL 接口支持设计实现外设的储存器映射访问。
4. 您可以选择 AXI4 流接口连接不需要地址关联的 IP 模块。该 AXI4 流接口广泛用于基于视频的 IP 模块。您可以使用连接到 HP 端口接口的视频 DMA IP 模块将视频 IP 模块的流视频输入存储到 PS DDR 储存器中。

案例分析：利用赛灵思 IP 模块设计

本节将介绍如何使用赛灵思 IP 集成器目录中的 IP 模块设计系统。您可以将此方法应用到任何 IP 模块。

案例研究介绍了一个陷波滤波器，是一个能够处理模拟样本以消除由外部传感器引入非线性的 DSP IP。例如,当模拟信号通过一个 RTD 这样的非线性传感器时，传感器可以在模拟信号输出端引入非线性行为。这是由于根据温度变化,传感器的传递函数产生了变化。非线性输出需要补偿，您可以在模拟或数字域中实现补偿算法。数字算法提供更具成本效益和更加准确的非线性信号的重建。

Zynq-7000 AP SoC 在 PL 中有一个赛灵思 ADC，例化 XADC 的 IP 模块为用户提供了一个封装选项,将 XADC 样本转换为 AXI4-Strem 兼容接口。用户可以通过查找多项式系数提前纠正 XADC 样品,为非线性传感器设计补偿 IP 模块。

您可以使用 Zynq-7000 AP SoC 上 的 DSP48 模块设计传感器补偿 IP 模块。 IP 可以实现两种用户接口：数据 AXI4 流接口和用于控制的 AXI4-Lite 界面。AXI4-Stream 接口可以连接 XADC 向导 IP，AXI4-Lite 接口可以连接 PS 的 GP0 主接口来控制特定的 IP 参数。

第 70 页的图 3-16 显示了 IP 模块的框图。

创建 IP 模块的流程：

1. 编写 HDL 代码，实现数据 AXI4 流接口和用于控制的 AXI4-Lite 接口。利用内插系数实现多重 AXI4-Stream 数据。
2. 使用 Vivado IP 打包器封装 HDL 代码。自动调用接口并将始终与每个接口关联。
3. 建立本地库并将经封装的 IP XACT 文件复制到本地库。
4. 创建 Vivado 项目，并添加到本地库。
5. 通过其他 IP 模块例化本地 IP 模块来创建模块设计，并将它们连接。

6. 使用 Vivado Design Suite 实现设计。

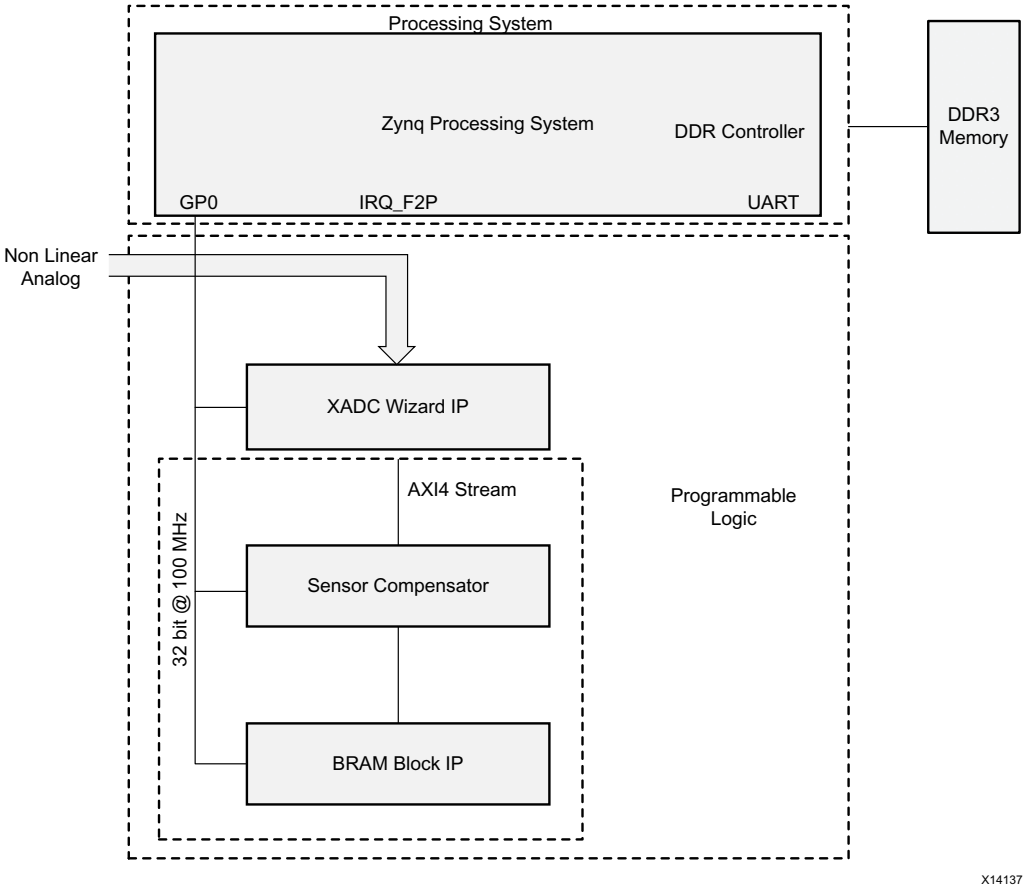

图 3-16 : IP 用例展示

硬件性能考虑事项

Zynq-7000 AP SoC 的 PL 具有多个用于软件和硬件共享数据与资源的通信路径。每个路径具有用于改变特定硬件的性能的配置设置。在本节中，将定义硬件性能指标，并对 AXI 主、 AXI 从、 AXI 数据路径的优化性能方法进行说明。

硬件性能指标

PL 的一个重要性能指标是 AXI 吞吐量与 Zynq-7000 AP SoC 内访问性能关键 IP 时的时延。赛灵思采用 PL 上的 ARM AMBA AXI 互联并映射用户 IP 到一个全局地址空间。一些应用数据路径将实施吞吐量约束 （如视频和网络）或时延约束 （如实时应用或特定协议的约束）。通常，当开始 Zynq-7000 AP SoC 设计时，很多 Zynq-7000 AP SoC 客户发现，系统架构师只能通过吞吐量和时延这两个简单指标来指定性能。

为突出时延和吞吐量之间的权衡，一个发出八个未完成 32 字节事务的 ACP 主器件将比单未完成事务主器件平均时延高出 130%。然而，具有八个未完成事务的主器件平均吞吐量要高出 262%。这是因为多个未完成的事务为流水线化，从而导致更高的吞吐量和时延性能信息。设计者可以判定对系统性能来说时延和吞吐量哪个更重要。

这两个指标构成了可视化和优化应用性能的基础。下面介绍 Zynq-7000 AP SoC 中一个可能受 IP 访问的 AXI 接口。这里不考虑 AXI IP 设计的最佳实践。

高性能 AXI 主

AXI 主的设计和配置影响其驱动高性能通信的能力。以下是从 AXI 接口看到的 AXI 主具有不同性能特性两个例子。

AXI 主 #1：支持 32 位 AXI-Lite 接口并每次发出一个未完成的 AXI 事务。主会产生单拍事务并在 ZC702 板上以 100MHz 运行。

AXI 主 #2：主使用一个 64 位的 AXI 4 接口，可以同时发出 16 个未完成的事务，每笔事务最大突发长度 16 次。主可以在同一个 Zynq-7000 AP SoC 板以 200 MHz 运行。

独立于数据路径和 AXI 目的从，AXI 主 #1 的性能将不会与 AXI 主 #2 匹配。为最大限度减少吞吐量约束，AXI 主应充分利用 AXI 地址和数据路径。因为事务将按顺序在 AXI 数据路径流水线化，所以为了尽量减少时延约束，要限制未完成事务的数量。

这个简单的示例可能比较直观，但构建高性能 AXI 主设计增加了复杂性，并且可能会消耗更多的资源。通常情况下，只有在性能目标没有达到时才考虑性能。以下部分概述了设计一个 AXI 主时需要考虑的性能。

构建高性能 AXI 主

许多设计是使用预勾线 IP 生成的，并且性能选择是有限。但是，下面的一般性准则有助于建立高性能 AXI 主，或在可以参数化时，调整现有的 IP。

- 对于吞吐量受限的主，将支持多个未完成的事务：这一要求增加了 AXI 主的复杂性，但 Zynq-7000 AP SoC 接口内部可以支持数目不等的多个事务。此数据流程水线可实现 AXI 数据路径更高的利用率，而数据流程水线往往是基于吞吐量的设计中唯一重要的通道。
- 对于低时延的主，节流未完成的事务数量来最大限度地提高时延与吞吐量行为：该项可能很难量化，但通过实验改变时延曲线的形状，在允许多个未完成的事务的同时可以减少主出现时延。图 3-17 展示了请求事务数量的增加如何影响时延的时延通量曲线。

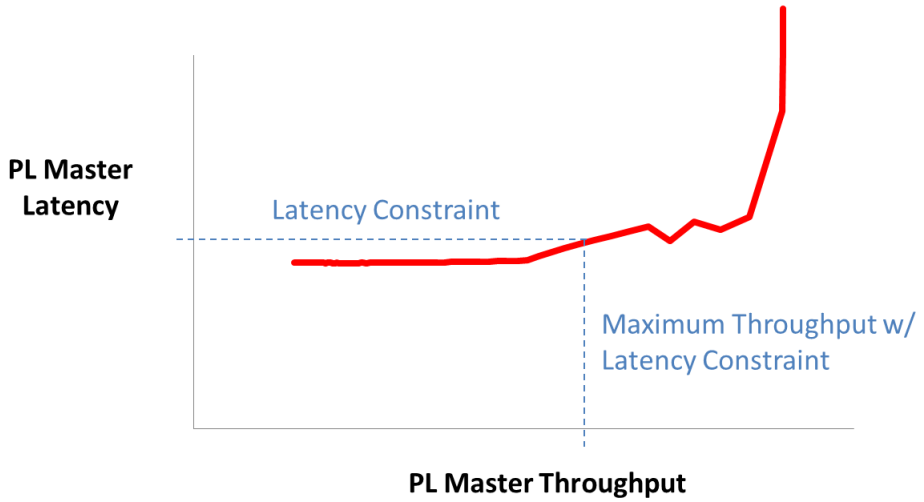

图 3-17 : 典型 HP 端口读取时延吞吐量曲线

- 忽略 AXI 响应通道：如果可能的话，使用贴写降低事务时延。例如，将 AXI rdlast 或 wrlast 信号的事务视为完成，便可以忽略写响应信道响应。在一个构建良好的设计中，不会出现 AXI 主必须管理的通道或从错误。如果写入无法发布，则通道不能被忽略。然而，在许多 Zynq-7000 AP SoC 数据路径中， AXI 写入响应通道通常会为事务增加几十个周期。
- 考虑 AXI 主处理的行为和目标 AXI 从的响应：一个 AXI 主生成的地址顺序会影响目标从服务流量的能力。例如，如果一个 AXI 主生成随机地址到基于 DDR 的储存器，它可能会导致储存器的性能下降。此外，多个主访问 DDR 地址范围可能会导致 DDR 调度欠佳。这并种问题并非一直存在，因为一些储存器从，如 OCM，能更好地处理随

机访问。此外，如果将随机访问限制在一个小的地址范围内，则 L2 高速缓存可以服务此流量。将在后面对这些 AXI 从进行讨论。

- 最大限度地提高时钟速率、数据宽度和事务突发大小：这些参数应匹配什么是由 Zynq-7000 AP SoC 本地支持的（如内部 64 位数据路径和 16 次最大 AXI3 突发长度）。通常情况下，增加 AXI 主的时钟速率仅影响总数据路径的一部分，因为事务可能还通过 PS 和它的时钟域传输。在需要的时候， PL 数据宽度将在内部转换到 PS 数据宽度。以突发大小为示例，如果一个 PL AXI 主驱动 AXI4 流量到 PS 中，事务可能会转化为 AXI3 事务。

使用性能关键的 AXI 从

性能关键从响应来自 PL 的数据移动请求。Zynq-7000 AP SoC 中有三个 AXI 从的性能特别关键：OCM、L2 高速缓存和 DDRC。也可以为 PL 添加一个可配置储存器接口生成器 (MIG)。不同从的好处在于：

- **片上储存器：**OCM 是能从 GP、HP、以及 ACP 端口访问 ARM 内核与 PL 的 256KB 储存器模块。OCM 是同步或暂存应用的理想组件。
- **L2 高速缓存：**只能从 PL 上的 ACP 访问 512KB 的 L2 高速缓存。它属于两个 ARM Cortex-A9 内核的共享资源。当数据流程量在 512KB 以内时，从具有优异的性能特性。但是，如果存在重度 CPU 争用，性能将会降低。 ACP 和一致的储存器访问将在第 80 页的“ACP 和高速缓存一致性”中进行介绍。
- **DDR 控制器：**您可以将 DDRC 连接到各种片外存储器件上，如 DDR3 和 LPDDR2。从 PL 使用 AXI 主接口 （GP、HP、以及 ACP）访问 DDRC。每个 PL 接口到 DDRC 具有唯一的数据路径。 DDRC 具有比 OCM 和 L2 高速缓存更高的存储容量和吞吐量性能。
- **附带 MIG 的 PL （可选）：**在 PL 上， Zynq-7000 AP SoC 能够支持 MIG 生成的储存器控制器。这种软 IP 选项提供了另一种高性能储存器接口，可以将 PL 储存器与来自处理器系统的软件对 DDRC 的访问隔离。 MIG 导出 AXI 接口，允许设计附加标准 AXI 主 IP。该核的最大好处是具有为特定应用定制 bank 分配与地址方案，针对特定应用纳入 PHY。

选择性能关键的 AXI 从

选择用于数据路径的 AXI 从通常能够满足功能要求，例如 CPU 一致性或储存器大小。一个 AXI 从的选择也会受性能要求影响。下面是被多个 Zynq-7000 AP SoC 客户采用的选择考虑因素：

- **避免或隔离性能关键数据路径：**在设计过程中，某些数据路径经常标记为对系统的性能起关键作用，其他路径则标记为次级关键。例如，网络处理应用可能需要片外储存器专用访问，而数据包检查可以通过关键性较低、速率较慢的数据路径实现。您可以通过限制 ACP 存取储存器访问占位来将 ACP 到 L2 高速缓存的访问与 HP DDRC 的流量隔离。同样，来自 GP的 OCM 访问可以与 ACP 和 HP 的流量进行隔离。
- **低时延：**时延的容许量取决于数据流程量的系统级应用和行为。 OCM 具有优异的低时延特性。通过 ACP 访问储存器较小的区域，能够为应用提供低时延 L2 高速缓存访问的优势。
- **高吞吐量：**高性能端口提供了从 PL 向 PS DDRC 的高吞吐量。 HP 性能的一个常见的问题是 DDRC 的 L2 缓存争用。占用大量储存器的应用软件可能会导致 HP 吞吐量减少。但是，您可以通过使用 DDRC 优先级设置和在 DDRC 数据路径到 L2 缓存的 QoS301 设置减少争用。
- **可配置性和信心：**一些设计中可能使用了一个 PL MIG，以避免与处理器子系统储存器访问争用。其他设计可能会使用基于以前的设计或开发经验的自定义储存器接口。 Zynq-7000 AP SoC 原型板支持 MIG （如 ZC706板），使设计人员能够评估一个自定义 MIG 核，并将结果与标准 Zynq-7000 AP SoC DDRC 的数据路径进行比较。

高性能数据路径

PS 有两个与 PL （主 GP0，主 GP1）进行通信的对称数据路径， PL AXI 主有七个用于驱动流量至 PS 的数据路径。

下面的章节将介绍 AXI 接口：

- 第 80 页的“ACP 和高速缓存一致性”
- 第 82 页的“PL 高性能端口访问”
- 第 91 页的“GP 和来自 APU 的直接 PL 访问”

对使用系统性能目标在这些接口之间进行选择的流程进行说明。

选择一个高性能数据路径

数据路径的选择通常基于设计功能。然而，性能方面的考虑因素也可以影响数据路径选择。

- **在 ACP 和 HP 端口之间进行选择：**当高速缓存和用户 AXI4 信号设置为高速缓存访问时， ACP 端口提供 L2 高速缓存访问，并保持与 ARM Cortex-A9 内核高速缓存一致。利用无高速缓存使能的 ACP 对 PL 用处很大，因为储存器映射与 Cortex-A9 内核匹配。 L2 高速缓存是一个共享资源，使用 AC P端口可以影响 CPU 储存器带宽。另外，如果 AXI 主生成的流量跨涵盖很大的地址范围 （如视频帧）或具备随机性质 （如分散 - 集中 DMA）， L2 高速缓存错失将增加，导致时延的相应增加，并可能降低吞吐量。

  HP 端口支持 32 位和 64 位数据路径，提供 DDRC 高吞吐量访问。针对突发吞吐量 HP 端口包含 FIFO ，您还能控制它们的优先级。HP 端口还提供低时延和高带宽的 OCM。如果与缓存相比 OCM 足够大，您可能会发现加速器对 PL 非常有用。

- **选择 GP 端口：**对于 ACP 和 HP 端口上隔离出来的性能关键数据路径，从 GP 端口为 PL 主提供了直接访问。
- **储存器访问占用：**如果一个设计使用局域性较差的 ACP 和高速缓存非一致访问，那么 DDRC 将服务于大部分 ACP 流量。在一个 Zynq-7000 AP SoC 部署示例中，通过提升高速缓存的事务局域性，将 ACP 时延降低高达 53%。 HP 和 G P事务不能访问 L2 高速缓存，并且不直接受到高速缓存行为影响。如需了解更多有关使用 ACP 一致性访问的详细信息，请参阅：第 80 页的“ACP 和高速缓存一致性”。
- **PS 数据路径性能控制：**系统时钟速率、 QoS-301 信号、 DDRC优先级设置都能够影响数据路径性能。在设计时，您可以设置大部分的储存器映射寄存器和使用 IP 集成器的功能。您可以在运行时设置 QoS-301 位。虽然较高的系统时钟频率有助于更快的传输数据， QoS-301 信号和 DDRC 优先级设置可以在争用 AXI 主之间变化性能。例如，L2 到 DDRC 数据路径的 QoS-301 设置可能会限制 L2 高速缓存到 DDRC 的事务，允许 HP 流量由 DDRC 更频繁地进行服务。 DDRC 优先设置也可用于 DDRC 的优先读取和写入事务。

监控硬件性能

如图 3-18 所示，赛灵思提供 AXI 性能监控 (APM) 的核，您可以用它来观察和测量 PL AXI 接口的时延和吞吐量。该核存在于 IP 集成器中，是本文件中引用的所有 AXI 性能测量的基础。

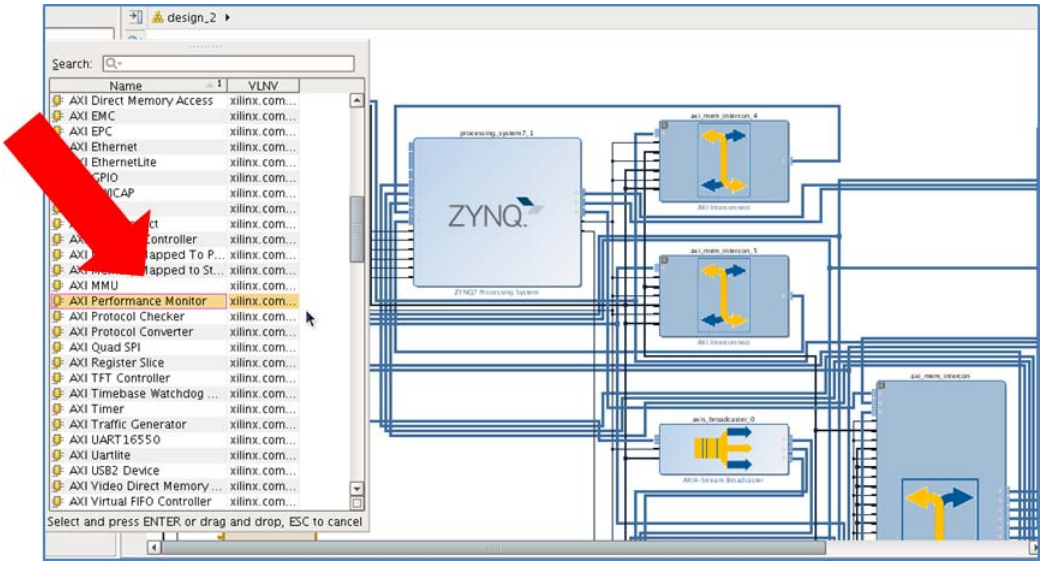

图 3-18 : 在 Vivado 2013.4 IP 集成器设计中添加 AXI 性能监视器

Zynq-7000 AP SoC 开发人员可以将这个 APM 像其他 IP 一样插入到他们的 PL 设计中，并在整个设计周期中验证性能指标。

数据流程

本节提供了 Zynq-7000 AP SoC 中数据流程程的概述。第一部分介绍了 PS 中的数据流程，重点是 PS DDR 的 APU 访问和连接到 PS DDR 的 PS 外设。第二部分介绍了从 PL 到 PS 的数据流程，重点讨论 PS-PL AXI 接口：通用接口、 ACP 接口、以及高性能接口。本节对主要系统设计特性和考虑因素进行了说明。如需了解更多信息，请参阅：Zynq-7000 All Programmable SoC 技术参考手册 (UG585) [参照 4]。

Zynq-7000 AP SoC 器件内所使用的各种互联模块设计用于满足各种功能模块的需要。PS 互联是基于高性能数据路径的开关，提供各种外设和 DDR 储存器之间，或在 APU 和 DDR 或 OCM 储存器之间的数据传输。PS-PL 接口在 PL 中提供 AXI 接口，将 PL 外设连接到 PS 和后续数据流程。

图 3-19 框图显示了本介中介绍的主要数据路径。

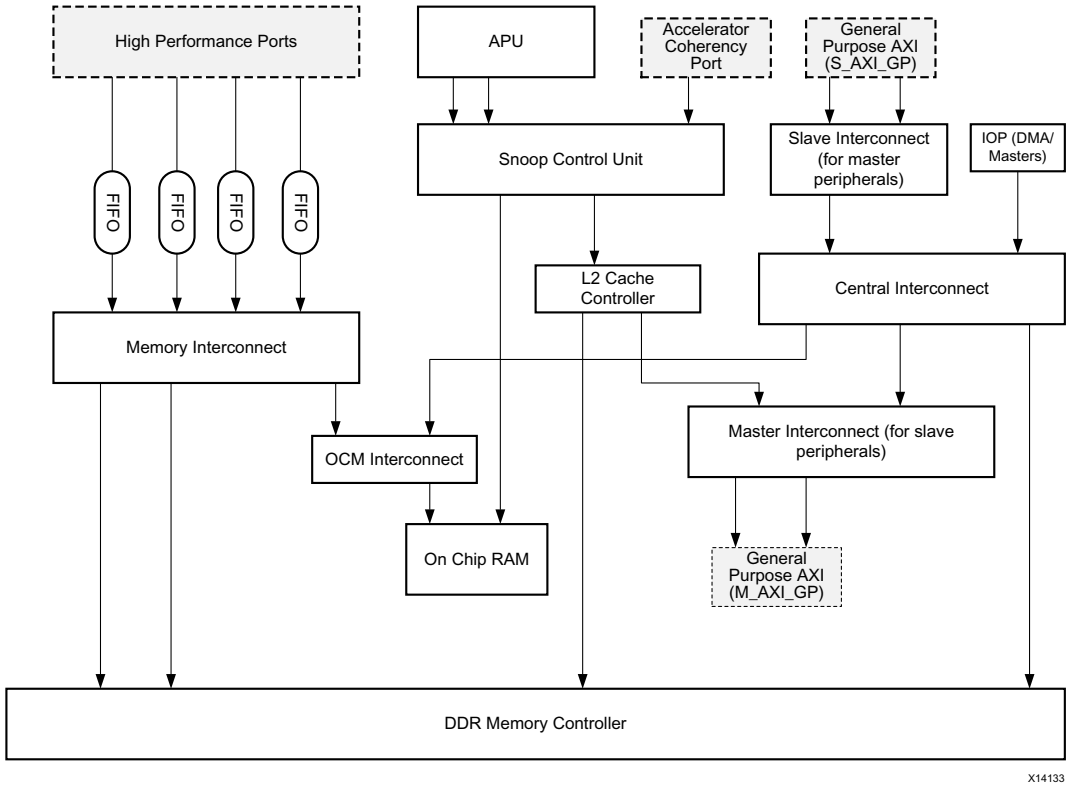

图 3-19 : 模块级数据路径概览

PS 内互联

在 PS 中有两个主要的数据路径：互联到 PS DDR 的 APU 与互联到 PS DDR 的 PS。

中央互联是 64 位的，同时也是 ARM NIC-301 开关的核心。主从互联布线低到中等水平的流量至中央互联与一般性互联，以及 PS 中的 I/O 外设。储存器互联提供从 PLHP 接口至 DDR 储存器控制器的直接高速数据路径。它还通过 OCM 互联提供了片上 RAM 访问。

互联无法提供一个完整的横杆结构;不是所有的主都可以访问所有的从。如需了解有关主可以访问哪些从的详情，请参阅：Zynq-7000 All Programmable SoC 技术参考手册 (UG585) [参照 4] 中的“互联数据路径表” （链接）。

APU 访问 PS-DDR

每个处理器提供两个 64 位伪 AXI 接口。所有 AXI 事务通过 SCU 布线至基于它们的地址的 OCM 或 L2 高速缓存控制器。APU 通过 SCU 和 L2 高速缓存控制器访问 DDR 储存器。 SCU 和 L2 高速缓存控制器行为与开关类似，这是因为他们具有地址过滤功能。SCU 涵盖 PL 中 APU 或 ACP 外设到 OCM 提供最低时延路径，还包含管理两个 CPU 和 L1 高速缓存之间数据一致性的智能。

当 L2 高速缓存控制器被禁用，事务直接保存到基于其地址的 DDR 控制器或主互联。

PS 外设访问 PS-DDR

I/O 外设 （例如， USB、 GEM 和 SDIO）实现用于通信的两个接口：

- 一个 APB 接口外设从连接到 APU 主
- 一个 AHB 接口提供了高速总线事务，将告诉外设中的内嵌 DMA 连接至系统储存器

APB 接口主要用于控制外设。 AHB 接口 （来自从一个外设的 DMA 主）连接到中央互联，提供 DDR 控制器访问。

PS-PL AXI 接口

本节介绍了 PS-PL AXI 接口，用于将基于 PL 的外设和功能连接到 PS。在各自的章节中介绍了每个接口的使用模式。

通用 AXI 接口 (AXI_GP)

AXI_GP 接口有四个 32 位通用端口，还有两个主端口和两个从端口;PL 是两个端口的主， PS 为其他两个端口的主。端口连与 PS 中主从互联端口连接。

所提供的 AXI_GP 端口用于一般用途，并未针对高性能设计。

M_AXI_GP 端口的一个典型应用模式是将它连接到 PL 中实现的外设控制平面接口。您可以使用 S_AXI_GP 端口与 PL 内有地址的 PS 外设进行通信。当对 PL 外设性能要求不高时，您也可以使用 S_AXI_GP 访问 PS-DDR。

加速器一致性端口 (ACP)

ACP 提供了直接连接到 SCU 的 PL 主接口。PL 主保持与 L1 高速缓存的储存器一致性，并且可对 L2 高速缓存和 OCM 进行直接访问。对于外设和储存器的， ACP 与 CPU 具有相同的连接水平。如需了解更多详情，请参阅：Zynq-7000 All Programmable SoC 技术参考手册 (UG585) [参照 4] 中“系统地址”的地址映射部分 （链接）。

ACP 端口地址由 SCU 嗅探提供全面的 I/O 一致性。 ACP 的读取能够在任何 CPU 的 L1 数据高速缓存内命中，而写入将使 L1 的陈旧的数据无效，并写入 L2。这提供了显著性能优点并简化了驱动软件。

ACP 允许外部 PL-DMA 之类的器件直接访问 CPU 一致性数据，无论数据处于 CPU 高速缓存或储存器层次结构的任何地方。系统一致性减少了软件驱动执行耗资源操作，例如缓存刷新，并有助于改善系统性能。

ACP 为高速缓存行长度传输进行了优化，同时应考虑到使用 ACP 设计系统时的某些限制。更多细节请参阅：第 80 页的“ACP 和高速缓存一致性”。

应谨慎考虑高性能应用中的 ACP 使用，因为连接到 ACP 的外设将与 CPU 竞争高速缓存与 PS 储存器子系统的访问，从而影响整体性能。

高性能端口 (AXI_HP)

提供了四个高性能 AXI 接口用于使能 PL 主以及 PS 中 DDR 和 OCM 储存器之间高吞吐量数据路径。每个数据路径都可以在 32 位和 64 位之间进行选择。一个专用的存储互联在 PS 中提供了 AXI-H P端口 DDR 储存器控制器的直接连接。此外，还存在 FIFO 缓存的读取和写入，以提供高性能。这些端口也称为 AXI FIFO 接口 (AFI)。

四个 AXI 的 HP 端口成对向下多路复用，并连接到储存器控制器上的两个端口。储存器互联在两个端口之间进行开关仲裁。当使用两个 HP 端口时，建议使用 port-0 和 port-2，因为这些都是单独的 DDR 接口。

HPAXI 接口提供额外的功能，如 QoS， FIFO 占用以及互联发行限制。当存在不同类型流量的多个主时，将使能带宽管理。您可以使用 PL 信号或通过静态配置 APB 寄存器来控制 QoS 信号。

除了上面讨论的 PS-PL AXI 接口，还有额外的 PS-PL 接口：

a. 扩展 MIO 允许 I/O 外设信号导向 PL。这在 PS IOP 控制器和 PL 中的用户逻辑之间提供了一个接口，而且它也允许使用的 PL 器件引脚。如需了解更多详情，敬请访问：Zynq-7000 All Programmable SoC 技术参考手册 (UG585) [参照 4] 中的链接。

b. PS-PL 时钟、复位和中断接口在 PL 中提供了时钟、复位和中断信号。如需了解更多详情，敬请访问：Zynq-7000 All Programmable SoC 技术参考手册 (UG585) [参照 4] 中的链接。

c. 器件配置接口，用于配置 PS 软件控制下的 PL。如需了解更多详情，敬请访问：Zynq-7000 All Programmable SoC 技术参考手册 (UG585) [参照 4] 中的链接。

d. PS 和 PL 之间的 DMA 接口可用于对 PL IP 模块和 PS 其他外设或储存器块之间的高速数据传输。 PS 和结构之间的接口可以用来传播 CPU 中断事件。

系统级设计考虑事项

在管理 PS-PL AXI 接口的带宽时，应考虑以下因素：

- 通用端口应用于低到中等类型的流量，如从 PS 控制 PL 的外设，或作为数据路径用于低速的 PL 外设。例如，因为端口满足了带宽和时延要求，您可以通过 S_AXI_G P端口将太网数据包移至 PL。然而，在需要保证时延和高性能的应用中，例如在 PL 中实现视频，使用 HP 端口更加合适。
- 当在 PL 应用需要保持与 CPU L1 高速缓存移至并减少软件开销时，引脚加速器一致性端口能够提供帮助。
- 当 PL 应用需要 DDR 控制器或 OCM 高带宽访问时，推荐选择高性能端口。提供多个端口，这样就可以在四个 HP 端口分配 PL 主以改善负载平衡，而不是将 PL 主限制到单个 HP 端口中。您可以使用多个 HP 端口集 （HP0 和 HP2，或 HP1 和 HP3）在一个 PL 主中获得更高的带宽。

  如需了解有关使用多端口集的详细信息，请参阅《Zynq-7000 AP SoC 技术参考手册》(UG585) [参照 4] 中的链接。

PL 时钟方法

本节涵盖了 Zynq-7000 AP SoC 的 PL 时钟方法，介绍 PL 和它们的推荐用法，以及可用的不同时钟源。这些时钟源是：

- 来自 PS 的时钟 (FCLK)
- 从 GT 恢复时钟
- 外部时钟源
- 由 MMCM 生成的时钟

在本节的最后，将讨论一个视频系统设计，作为使用各种时钟源的典型示范。

来自 PS 的时钟 (FCLK)

如需了解使用 FCLK 的详情，请参阅：第 2 章中的时钟和复位。

从 GT 恢复时钟

GT 收发器的接收器时钟数据恢复 (CDR) 电路从输入数据流中提取时钟和数据。恢复的时钟与收发器线路速率和数据路径宽度有直接关系。对于需要通过高速串行收发器与链路伙伴通信的 PL 设计，您可以使用恢复的时钟。所恢复的时钟通常存在固有抖动，在输入抖动敏感的设计中使用时钟之前必须先纠正该时钟。在一般情况下，恢复的时钟用于采样 PL 逻辑接收的数据。 CDR 架构如图 3-20 所示。

如需了解更多信息，请参阅：《7 系列 FPGA GTX/GTH 收发器》(UG476) [参照 2] 与 《7 系列 FPGA GTP 收发器》(UG482) [参照 3] 中的“接收器”章节。

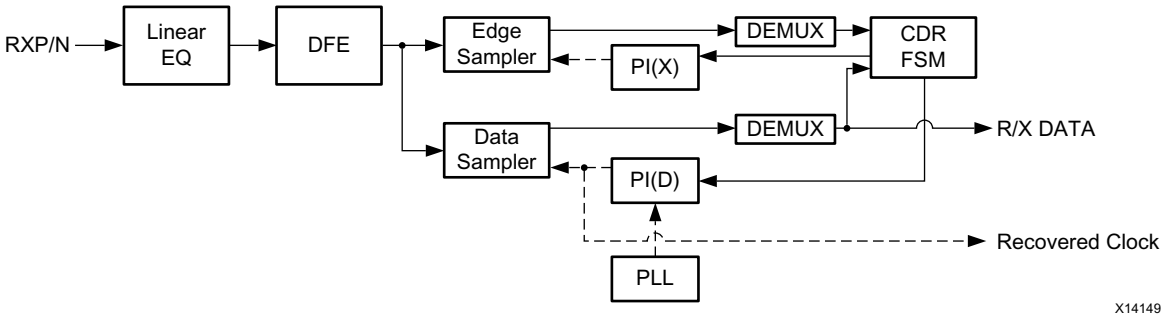

图 3-20 : CDR 架构

外部时钟源

能够当做 FPGA 时钟信号的 I/O 引脚可以提供一个 PL 时钟，引脚需支持差分时钟。如果使用单端时钟，应当连接时钟功能输入引脚对中的P （主）一侧。

单端与差分输入时钟的选择取决于设计的时钟要求。建议使用差分时钟，因为它消除了电源噪声和线路耦合噪音。如果设计的引脚有限，可以使用单端时钟。

当将一个单端时钟连接到一个输入引脚对的 P 侧时，那么 N 侧不能连接另一单端时钟。然而，您可以将它作为一个用户 I/O 使用。

如第 78 页的图 3-21 所示，能够作为时钟信号的引脚为内部全局和局部时钟源提供了专用高速访问。

如需了解更多信息，请参阅：《7 系列 FPGA 时钟资源》(UG472) [参照 1]。

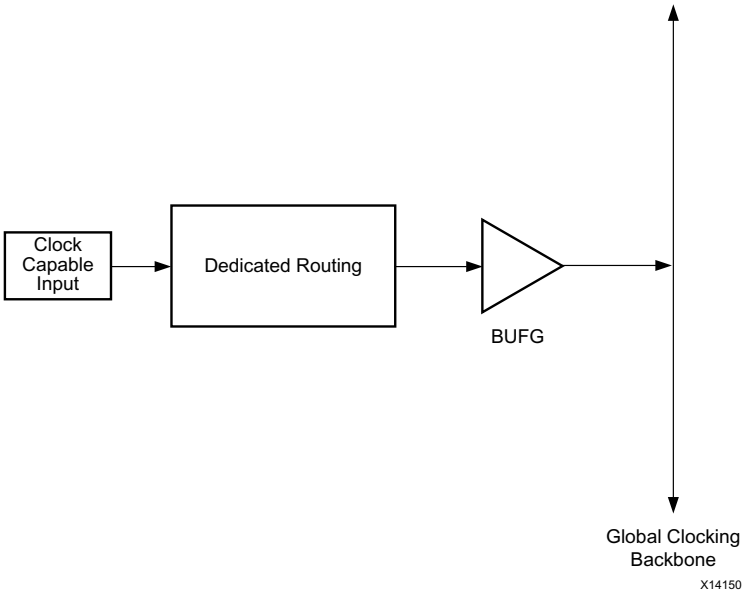

图 3-21 : 时钟输入引脚

由 MMCM 生成的时钟

MMCM 用于生成具有不同频率和相位关系的多个时钟。MMCM 还可以进行去偏差时钟输出。图 3-22 展示了使用模型。

MMCM 原语 MMCME2_ADV 提供了上述功能，并且使用动态重置端口 (DRP) 选择输入时钟。

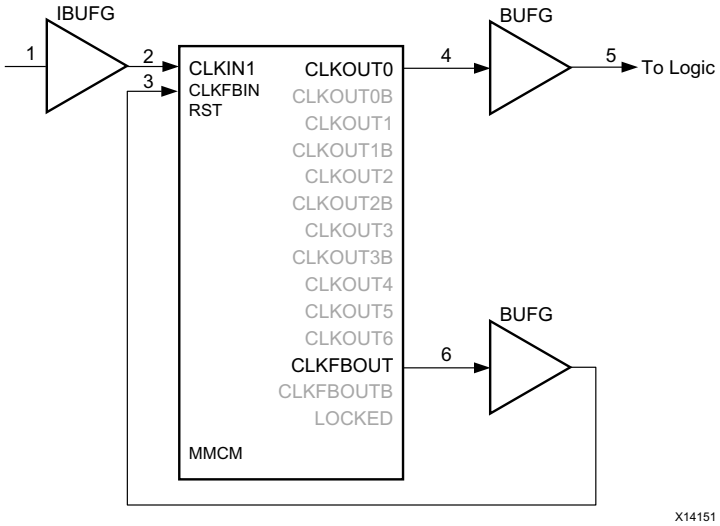

图 3-22 : MMCM 使用模型

FCLK 使用模型

FCLK 使用模型示例：在显示监控器上显示测试图案的视频应用。图 3-23 展示了该模型。

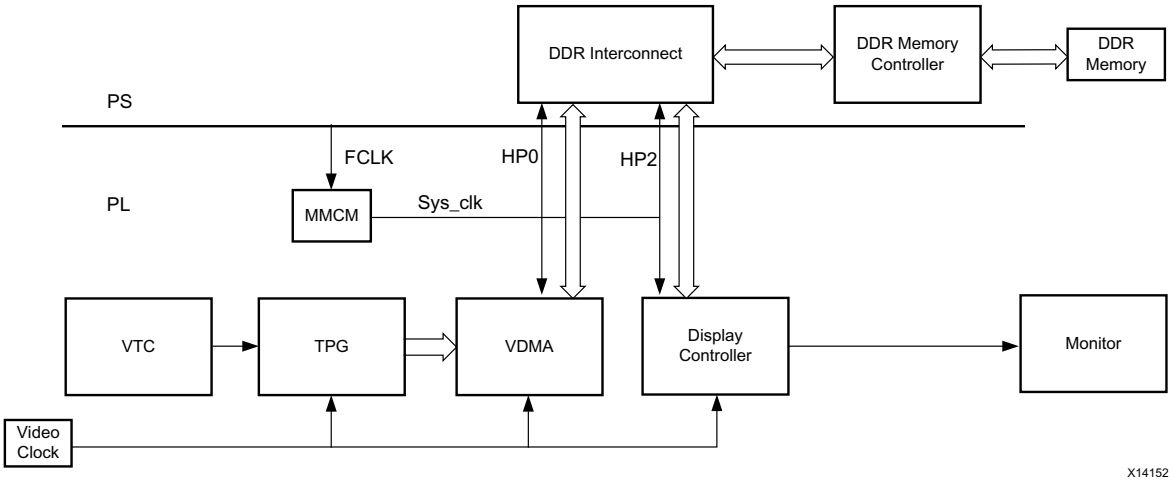

图 3-23 : FCLK 使用模型

使用模型具有以下 IP 模块：

- 测试图形生成器 (TPG)：生成不同测试图案的视频帧。
- 视频时序控制器 (VTC)：生成用于 TPG 的时序。
- 视频直接储存器访问模块 (VDMA)：将 TPG 生成的视频帧写入 DDR 储存器。
- 显示控制器：通过 HP2 端口获取来自 DDR 的帧，并在监控器上显示。
- 板载时钟综合器：作为视频时钟驱动所有输入视频模块和显示控制器。可由 PS 进行编程。
- 使用 MMCM 从 FCLK 衍生时钟：时钟具有比 Sys_clk 更高的频率。因为显示控制器的时延要求，由 VDMA 和显示控制器 AXI 主接口使用。

在这个示例中，设计可以使用一个外部时钟布线连接 PL 和生成所需 Sys_clk 输出的 MMCM。PS AXI 互联提供了 HP/GP 端口和 PL 之间的时钟域。

ACP 和高速缓存一致性

加速器一致性端口为 PL 主提供低时延访问，包括可选的与双核 ARM Cortex-A9 CPU 高速缓存一致性。从系统的角度来看，ACP 接口具有与 APU CPU 类似的连接。正因为如此，ACP 与 APU CPU 竞争 ACU 模块外部资源的访问。图 3-24 展示了 ACP 连接性。

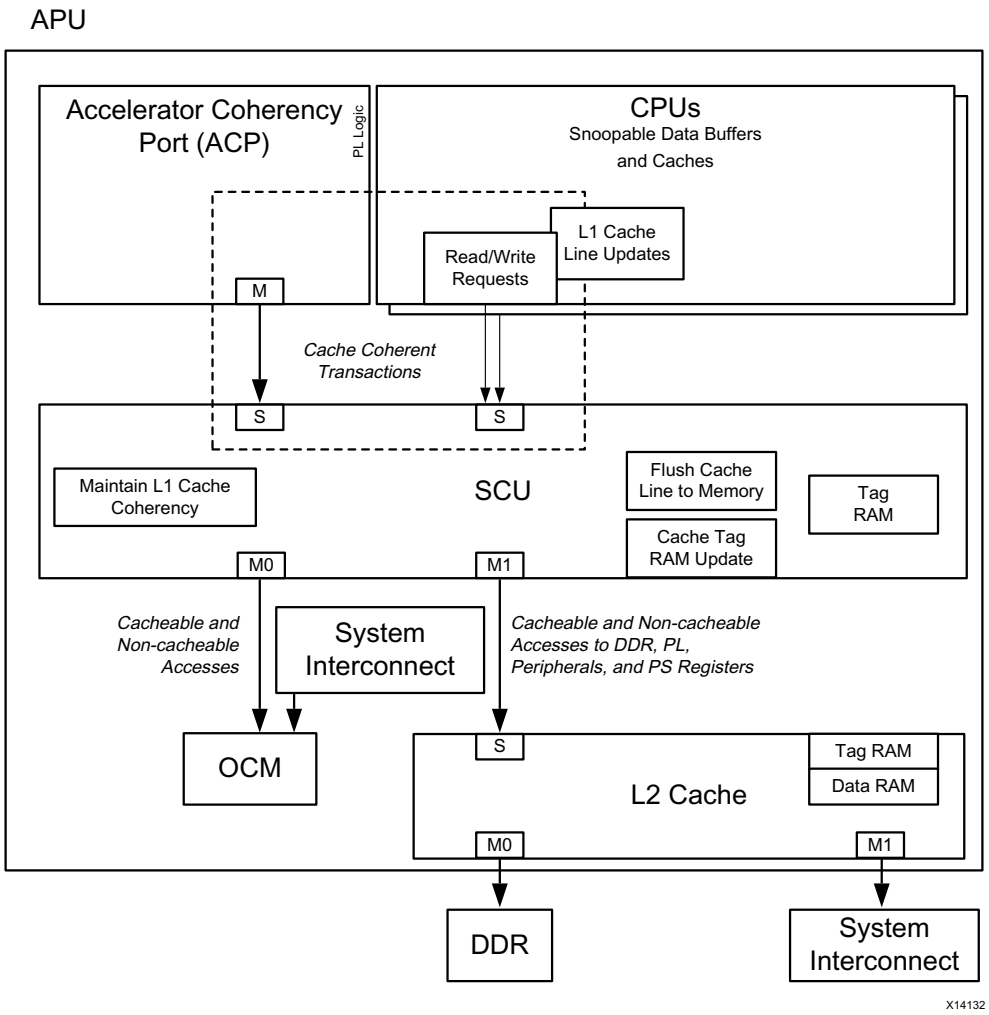

图 3-24 : ACP 连接图

相比其他 PL 主，ACP 的主要好处在于获取 CPU 高速缓存一致储存器和使用低时延 512KB L2 高速缓存。在一个常见的 Zynq-7000的AP SoC 实现中，一个 PL 加速器被连接到 ACP 端口。它由 CPU 上 的一个 GP 端口配置，然后加速器在 ACP 上进行数据移动。在硬件上进行一致数据的移动管理，使 PL ACP- 主发出标准的 AXI 读取和写入指令。 ACP 高速缓存一致访问的唯一要求是，这些事务的 A*CACHE 和 A*USER 位均设置为 1。

此硬件管理一致性模型是让 PL 加速器使用 HP 或 GP 端口直接从 DD R控制器访问非一致储存器的替代方案。这里，软件管理一致性是必要的， CPU 必须将共享储存器刷新回 DDR 来保证一致性。虽然有许多实现软件管理一致性的方法，但是在硬件中使用 ACP 能够缓解软件设计压力。

由于 ACP 支持高速缓存一致性， ACP 相连的 AXI 主将与 CPU 的 L2 高速缓存产生资源竞争 （L1 高速缓存仅适用于 CPU）。来自 ACP 的任何 AXI 读或写事务将使用 L2 高速缓存资源来服务数据移动 （仲裁、标签匹配、高速缓存行提取到 DDR 控制器）。如果此数据移动与 CPU 活动协调 （例如上述加速器的示例），则共享储存器资源有利于较大的系统

中的应用。下一节将会介绍，如果该数据移动在与 CPU 任务无关的并行操作中运行，这对缓解 CPU 和 ACP 主争用提供了更多机遇。

ACP 设计方法

下面有效利用 ACP 设计方法的最佳实践基于几个客户的使用案例，以及 Zynq-7000 AP SoC ACP 的性能分析。

ACP- 主设计方法

- 避免缓存颠簸：512KB L2 高速缓存是两个 ARM 处理器与 ACP 之间共享的资源。 ACP 访问大批量数据的能力具有降低软件性能的潜在风险。高速缓存锁定和划分有助于确保共享数据具有较高的缓存命中率。举例来说， ACP 端口的视频式访问几乎不存在高速缓存局部性，不进行高速缓存管理将会损害储存器系统的性能。

  在性能测试实例中，相比于使用没有缓存局部性的 ACP 访问模式，具有良好高速缓存局部性的 ACP 访问展示出 54 % 的时延缩短。结果视乎各应用而不同。

- 将 AXI 事务参数与 ACP 匹配：ACP 是兼容 AXI3 的 64 位接口。连接 ACP 兼容 AXI3 或 AXI4 标准的主事务将被转换为 64 位宽事务，最大突发长度为 16。所有的转换如协议转换器或 AXI 互联不再使用软 IP。转换后的非 64 位事务和高节拍数事务有可能造成额外的数据传输周期。其他 ACP 访问优化和约束将在 ARM 信息中心加速器一致性端口网页进行介绍 [参照 73]。
- 选择阻塞与非阻塞事务：对于 ACP 端口上完成十条的订单必须重点关注。在某些情况下，写和/或读可以无序完成，除非使用 axresp 通道用作后续事务之间的屏障。理想情况下，响应通道应被忽略，从而促使事务尽快完成数据传输。然而，需要注意在相同的地址范围内发出多个读取和写入指令。对于 DMA 批量传输，这通常是没有问题的。但是，在多个未完成读写事务向一个小地址范围发出时，您就会发现问题。

  在一个用例中，如果 ACP 的响应通道在多个读 - 修改 - 写操作到相同的地址被忽略时，可能会发生乱序读后写冲突。

- 使用 ACP 进行非一致性问：建议将来自 PL 的非高速缓存一致通信通过 HP 或 GP 端口布线。但是，如果需要，您可以设置 AXI 4 位缓存字段，修改其 ACP 访问行为。如需了解更多信息，请参阅：《Cortex-A9 技术参考手册》[参照 75] 中第 2.4 节“加速器一致性端口”。

针对 ACP 设计方法的 PS 配置

- L2 高速缓存时钟和预取：L2 高速缓存匹配 CPU 的时钟速率，具有良好的高速缓存本地性，将服务于广大 ACP 的访问。虽然可以提高 PL 时钟来降低访问时延，您也可以将 L2 高速缓存时钟设置得尽可能快。您可以使用 L2 预取设置来开启和关闭预取，并使用其他设置来控制高速缓存每错过取行数。例如，有少量本地性高速缓存的ACP 访问可能无法从预取中受益，因为他们会从 L2 到 DDR 控制器导致误判提取。

  如需了解更多有关 ACP 访问优化 PS 设置和修改 L2 高速缓存设置的信息，请参阅：Zynq-7000 All Programmable SoC 技术参考手册 (UG585) [参照 4]。

CPU 软件争用方法论

- 考虑 CPU 储存器访问模式：处理器与高速缓存的争用效果也由 CPU 的储存器访问模式决定。完全不在 CPU L1 高速缓存上运行的软件不受 ACP 访问的影响。然而， ACP 访问可能会影响低高速缓存局部，诸如储存器、跨步应用。

  例如，在 ARM Cortex-A9 上运行的储存器密集型应用软件不受重大 ACP 流量访问 L2 高速缓存的单个 4 KB 储存器页面影响。

  举进一步的例子，在 CPU 上运行 L1 高速缓存绑定基准，如 Dhrystone 或 CoreMark，不受重大 ACP 流量访问 L2 高速缓存内 512 KB 缓存的影响。

- 高级高速缓存管理：您可以使用相关技术，如高速缓存锁定和缓存划分来协助多个 CPU 和 ACP 主之间的共享缓存和缓存隔离。这些技术可能需要修改应用储存器布局的连接器脚本和定制 ARM 储存器描述符。如需了解如何锁定缓存的教程，以及可能带来的性能影响，请参阅：Zynq-7000 AP SoC 启动——L2 高速缓存锁定与执行的维基页 [参照 60]。

- 争用监控：当设计目标没有得到满足时，监控并衡量竞争是非常重要的。为提高 ACP 性能，您可以监控 L2 缓存、CPU 甚至 ACP 本身。您可以使用事件计数器监控 L2 高速缓存、使用 ARM 性能监控单元 (PMU)，还有使用 PL 中实现的赛灵斯 AXI 性能监控器与 ACP。第 10 页的“性能” 对这些监控方法进行了进一步介绍。

PL 高性能端口访问

HP 端口可提供 PL 对 DDR 控制器和片上储存器 (OCM) 直接访问。 Zynq-7000 AP SoC 技术参考手册对 HP 端口和可能的性能优化提供了完善的功能介绍。本节将介绍使用 HP 端口在客户设计中实现的常见设计驱动优化。

HP 端口通常于一个设计需要 PL 高吞吐量访问 OCM 的情况，或更常见的 DRAM 连接 DDR 控制器流程。图 3-25 显示了 HP 端口彼此争用和与 PS 架构中位于不同点 的 PS 主争用情况。下面介绍存在多个 HP 接口争用系统中实现高吞吐量设计的方法。

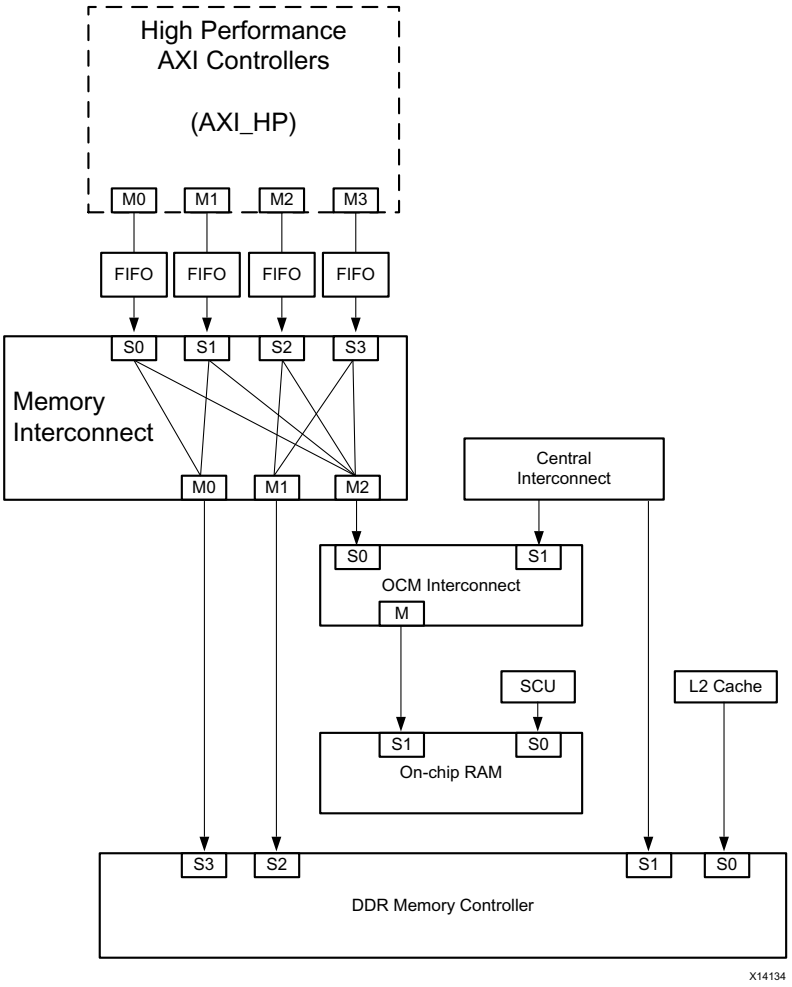

图 3-25 : 高性能 (HP) 端口连接性

HP 主设计方法

- 添加 HP 端口的顺序：因为 HP 数据路径的关系，应添加 HP 端口以最大限度减少数据路径争用。一般情况下， HP 端口应按以下顺序添加：HP0、HP2、HP1 和 HP3。依照这一顺序， HP 主可以充分利用两个专用 DDR 控制器端口的优势。

  例如，在一个需要两个 HP 端口的设计中，如果使用 HP0 和 HP2 代替 HP0 和 HP1 布线重度写流量，能够实现 15% 的总体吞吐量提升。由于更高的吞吐量流量得到驱动，性能差异将会增大。

  如果需要四个以上的 HP 主，您可以在 PL 中使用一个 AXI 互联将主复用到四个 HP 端口上。当将主分配至 HP 端口时，目标是最大化 HP 端口吞吐量。例如，您可以将高流量的主分配至另外的 HP 端口并与低流量的主配对。一个重要的考虑因素是，单个 HP 端口可能在一个低于 DDRC 饱和吞吐量的水平达到饱和。

- AXI 寻址优化：如果可能的话，应考虑标准 DRAM 访问优化，包括地址优化，并尽量减少读取/写入组合。《使用 Zynq-7000 All Programmable SoC 设计高性能视频系统》(XAPP792) [参照 34] 为使用非线性寻址匹配视频应用 DRAM 的行-组-列布局提供了指导。此外，应在可能的情况下尽量减少读/写混合，尽管由于设计行为的原因这并不总是可行。

  例如，在具有只读 HP0 和 HP2 端口的系统中，当 HP2 被改为只写时，HP0 读取吞吐量下降 0.12%。性能下降的范围取决于流量寻址和生成的事务速率。

- 提高 PL 时钟速率：提高可编程时钟速度是最大限度提高应用性能的常用方法。但是，当在 PL 和 DDR 控制器之间的数据移动存在性能瓶颈时，增加 PL 时钟速率会导致收益递减。请参阅：图 3-25 中的数据路径，虽然 PL 主是在 PL 时钟域发起流量，但是大多数数据路径存在于 PS 时钟域。此外，应考虑 DDRC 的最大带宽，因为 DDRC 最终会由于增加的 PL 主流量而饱和。

  例如，在一个具有两个 HP 端口驱动的重读写到 533MHz 的 DDR 控制器设计中， PL 时钟速度从 50MHz 增至 100MHz 合计增加 HP 吞吐量达到 92%，而从 100MHz 增加到 200MHz 时， HP 吞吐量增加了 13%。后一种情况下，更常见 PL 驱动流量，从而产生饱和 DDR 控制器。

面向 HP 设计方法的 PS 配置

当利用 HP 端口设计时，DDR 控制器的高吞吐量流量与连接的 DRAM 目的地争用不容忽视。HP 端口有两个专用端口连接到 DDR 控制器，但这些都与 L2 高速缓存和 DDR 控制器中心互联仲裁。如果 L2 缓存保存请求数据，由于 L2 缓存服务 CPU 和 ACP 的请求，您可以最大限度地减少来自 512KB L2 缓存端口的争用。然而，如果 CPU 或 ACP 访问储存器具有很小的缓存位置，由 DDRC 提供的高速缓存访问服务会潜在地损害 HP 性能。此外， DDRC 中的中央互联争用由应用决定，可能有来自 USB 或前朝以太网主的重度流量。

您可以使用两种先进的技术来限制 L2 和中央互联接入到 DDRC，目标在于在高争用的设计实现里提高 HP 吞吐量。当尝试了默认设置而 HP 性能目标无法时，您应使用这些技巧。您可以修改下列中一项：

- DDRC 优先级设置
- PS 中的 QoS-301 设置

该技巧将 HP 流量优先定向 DDR，但不一定会提高整体性能。相反，放慢其它 PS 主对 HP 将带来益处。

- DDR 控制器优先级设置：|这些设置控制 DDR 控制器端口与 DDR PHY 服务的事务顺序。高优先读取功能和 go2critical 计数器值是 DDR 控制器优先级设置控制的最重要特点。您可以使用每个 DDR 控制器端口上的高优先级读取选项来提高端口读入优先级，使之比每个其他 DDR 控制器端口优先级更高。这种优先级提升可在影响高负载系统 HP 吞吐量 （例如，所有的 HP 端口驱动高吞吐量流量）。当 Zynq-7000 AP SoC 的性能达到极限时，比如 DDR 控制器达到理论最大吞吐量时，这些高级功能将非常有帮助。如需了解更多信息,请参阅：Zynq-7000 All Programmable SoC 技术参考手册 (UG585) [参照 4] 中的链接。
- QoS-301 设置：从 L2 高速缓存到 DDR 控制器的 AXI 数据路径具有目前只能从应用通过直接寄存器写入访问的 QoS-301 设置。使用 Vivado GUI 界面则无法修改它们。该功能允许路径上的事务限制，对在重度 DDR 流量情况在提高端口性能有益。当 L2 高速缓存区域性较差的储存器密集型软件运行时，最好使用此功能。如需了解更多信息,请参阅：Zynq-7000 All Programmable SoC 技术参考手册 (UG585) [参照 4] 中的链接。

DDR 控制器方法争用缓解

类似于 ACP 争用源，HP 性能受系统负载的影响，特别是在 DDR 控制器中。争用的最大潜在来源是 L2 高速缓存进行高速缓存行请求，服务 CPU 或 ACP 访问。CPU 和 ACP 争用将取决于超过 512KB L2 高速缓存容量的储存器访问模式。来自中心开关的以太网和 USB 流量也会为 DDR 控制器带来大量流量。

- 争用监控：当设计目标没有得到满足时，监控并衡量竞争是非常重要的。为保障 HP 性能，您可以监控 L2 高速缓存、以太网控制器和 HP 端口。您可以使用核内事件计数器监控 L2 高速缓存和以太网控制器，使用 PL 中实现的赛灵斯 AXI 性能监控器来监控 HP 端口。
- 什么样的争用程度会构成问题？当 DDR 总系统流量接近 DDR 控制器的理论最大值时，大部分争用缓解技巧都能够显著改善性能。例如，在 ZC702 板上以 533MHz 运行的 32 位 DDR 控制器理论上可以维持 4.3Gb/s 的储存器吞吐量。DDR 控制器设计用于公平仲裁和维持数个高吞吐量源，在 DDR 控制器利用率较低时，将不会出现明显争用现象。
- 使用 PL 储存器附带的接口生成器 (MIG)：赛灵斯还提供了片外 DDR 的 PL 储存器接口，可以补充或代替 HP 到 DDR 控制器的数据路径。在功能上，您可以从 PS 储存器映射上隔离此添加的储存器，也可以使用 GP 端口访问它。如需了解更多信息，敬请参阅：赛灵思解答记录 58387 [参照 71]。

  对于从单一的主想相邻地址传输大批量数据的应用，如以太网和视频帧，为了获得更好的性能，MIG 通常是更好的储存器选项。此外，该储存器选项是最适合缓冲来自 PS 访问的隔离数据，因为 CPU 直接访问 PL DDR 会通过 32 位的 GP端口，限制 MIG 性能。否则， PS DDR 控制器为 HP 的工作负载提供了优秀的储存器性能。

系统管理硬件辅助

本节介绍 Zynq-7000 AP SoC PS 中有助于系统管理的硬件组件。系统管理包括基于用户特定输入的系统级参数控制。芯片级管理包括晶片温控、错误管理、系统闲置时的低功耗模式、对外设的安全和非安全访问管理。

执行系统管理时软件组件发挥着重要作用。本节的重点是 Zynq-7000 AP SoC 上系统管理的硬件辅助。

赛灵思模数转换器

赛灵斯模数转换器 (XADC)是 PL 中有助于监测温度和电源传感器的灵活模拟接口。 XADC 具有三个接口， JTAG、 DRP 和 PS-XADC，系统应用可以利用这三个接口监控温度和电压的状态，并采取适当的行动。 JTAG 和 DAP 接口存在于所有 7 系列 FPGA中。然而， PS-XADC 接口仅存在于 Zynq-7000 AP SoC 器件中。

图 3-26 显示 XADC 的 AXI XADC DRP 接口和 PL-JTAG 或 PS-XADC 接口之间如何进行仲裁。

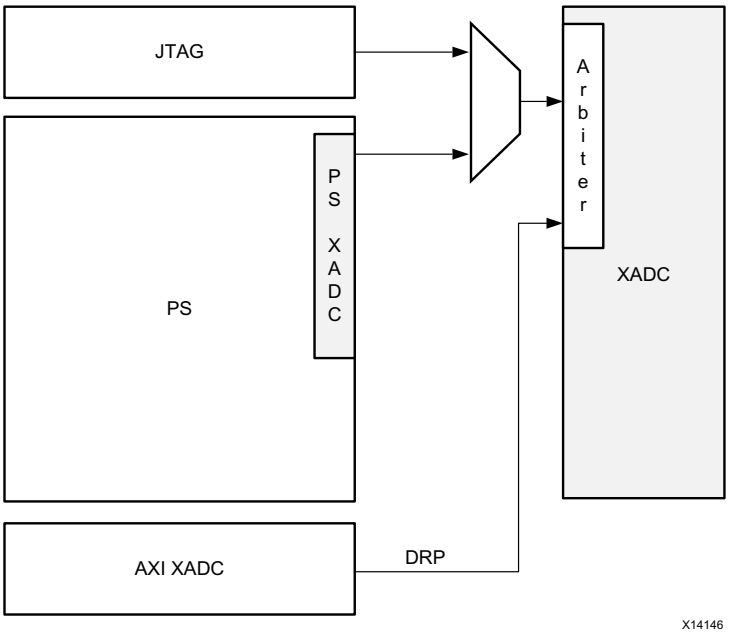

图 3-26 : XADC 仲裁

TrustZone 安全性

ARM TrustZone 技术在多个层面支持系统级的安全性，包括硬件、软件和存储。Zynq-7000 AP SoC PS 架构支持多种工作模式，包括主管、系统和用户模式，并且在应用层面提供不同级别的防护。

处理器在两个独立的环境，安全环境和正常环境之间进行切换，通过处理器模式又称为监控模式实现，如图 3-27 所示。 TrustZone 技术在许多的 PS 组件中实现，包括：APU、 L1 高速缓存控制器、储存器管理单元、 SCU、 SLCR、三时序计数器、 watch-dog 定时器、 I2C、 GPIO、 SPI、 CAN、 UART、 QSPI、 NOR、 DDR 储存器控制器、 L2缓存控制器、 AXI 互联片上储存器、 DMAC 千兆以太网控制器、 SDIO 控制器和 USB 控制器。

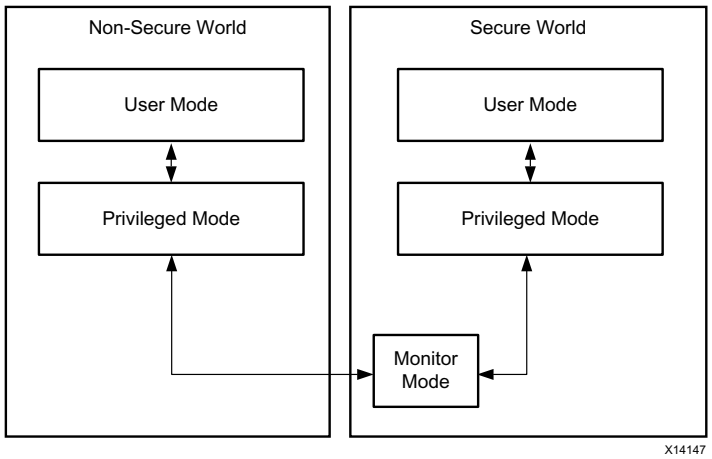

图 3-27 : 安全和非安全环境之间的切换

DDR 储存器控制器

您可以在 64MB 段配置 DDR 储存器，并且可以使用控制寄存器， TZ_DDR_RAM (0xF8000430) 将每段配置为安全或非安全。

- 0 位值表示用于相关储存器段的安全储存器区域。
- 1 位值表示用于相关储存器段的非安全储存器区域。

当 DDR 储存器的安全段出现非安全访问时， DECERR 响应将返回给事务的发起者。写事务被屏蔽，导致 DDR 储存器不发生写操作，所有读取事务将返回零。

L2 高速缓存控制器

L2 高速缓存控制器通过向缓存数据增加一个 NS 位支持 TrustZone 技术。在这种方式中，高速缓存控制器像在两个不同的存储空间之中对待安全和非安全段。对于一次读传输，高速缓存控制器发送一个线路填充命令到外部储存器，从外部储存器将任何安全错误传播到处理器，并且如果存在错误，便不在 L2 高速缓存上分配行。

DDR 错误恢复

纠错码 (ECC) 机制支持 16 位 DDR 内容。这使 DDR 损坏数据得到恢复。每个 16 数据位的 ECC 位都支持单位错误更正和双位错误检测。当使能 ECC 用于写入时，ECC 码与数据一起被计算并写入 DDR。当 ECC 被读出时，DDR 控制器检查比对存储 ECC 码与数据。不使能 ECC 就写入 DDR 储存器，然后使能 ECC 读取，可能会返回 ECC 错误。

当发生的错误是可纠正时，则控制器纠正并发送数据，它不产生中断或 AXI 响应。当一个错误不可纠正时，控制器发送 AXI SLVERR 响应的受损数据到 AXI 主。 PS 中的 AXI 主也可能会生成 L2 或 DMA 中断，或 CPU 预取或数据异常。当他们这样做时，同样的 AXI SLVERR 响应将发送到 PS AXI 主。

低功耗

您可以对 Zynq-7000 AP SoC 采取不同的功耗优化方式：PL 电源关闭、 CPU 待机状态、 PS 子系统时钟门控、 PLL 配置或 I/O 电压控制。

当 PL 不使用时，就可以断电整个 PL。然而，在断电时， PL 会失去状态，当它再次加电时就必须重置。系统软件应判断 PL 何时断电，以及当再次需要 PL 时重置它。

时钟门控

PS 具有多个时钟域，并且每个时钟具有门控以停止时钟。当时钟域处于未使用的状态时，系统软件可以使能特定时钟的门控。这减少了动态功耗。

您可以使用系统级控制寄存器 (SLCR) 禁用起始地址为 0xF8000000 的时钟。

例如，当 SPI 控制器处于未使用状态时，可以通过写入一个合适的值来注册 SPI_CLK_CTRL (0xF8000158) 来禁用控制器参考时钟。

图 3-28 显示了 SPI 参考时钟的时钟门控工作原理。

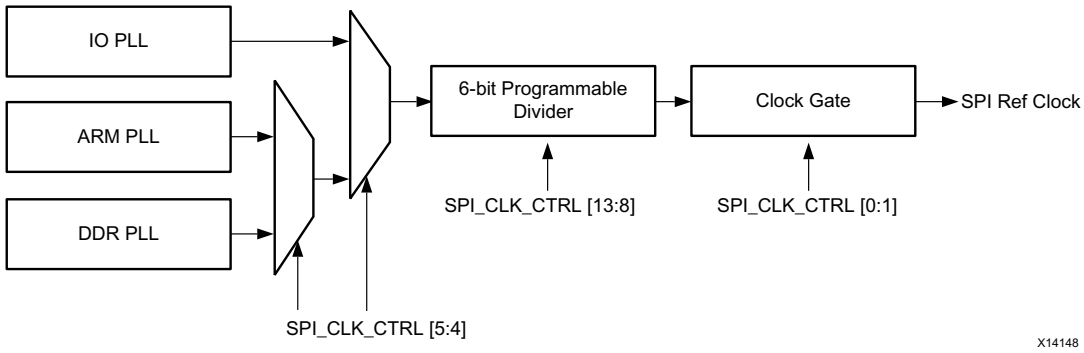

图 3-28 : 时钟门控

管理硬件重新配置

部分重配置是一种重置 FPGA 一部分的能力，能够在不扰乱片上其余逻辑的情况下实现不同的逻辑功能。该技术使 FPGA 以一个时间分片的方式来实现不同的功能模块。

提示： Documentation Navigator 中的“部分重配置设计中心”提供了有关部分重配置附加信息的链接。如需了解更多信息，请参阅第 158 页的“相关设计中心”。

FPGA 部分重配置通过下面的结构支持：

- 一个在无需关闭供电电压的情况下使能 SRAM FPGA 部分重配置的配置选择行。
- 一个可以存储用于部分重配置的配置数据配置锁存器。

在 Zynq-7000 AP SoC PL 架构中，基本的可编程单元是由 LUT、 DSP48 和 BRAM 组成的一个配置框架。在部分重配置中，配置帧利用处理器配置访问端口 (PCAP) 作为重配置代理来重新编程。每个配置帧具有唯一的地址，通过顶/底部的地址位识别、一个主要的地址，以及一个较小的地址。使用赛灵思 Vivado 工具流生成部分码流，而部分码流由 PCAP 解析进而重置帧。

在一个传统的完全配置中部分重配置具有以下优点：

- 减少了硬件资源的利用：通过动态时间分割设计功能，设计人员可以在现有器件内容纳更多逻辑。
- 提高生产力和可扩展性：只有修改功能需要与已验证的设计关联执行
- 增强可维护性，减少系统停机时间：您可以部署新的功能，并且在系统启动并运行时动态插入

图 3-29 显示了部分重配置方法的概念视图。静态设计不随时间而改变，并建立了部分重配置模块和静态设计之间的通信。静态逻辑和部分重配置模块之间的通信是通过一组由双向通信实体和三态门构成的通信宏。

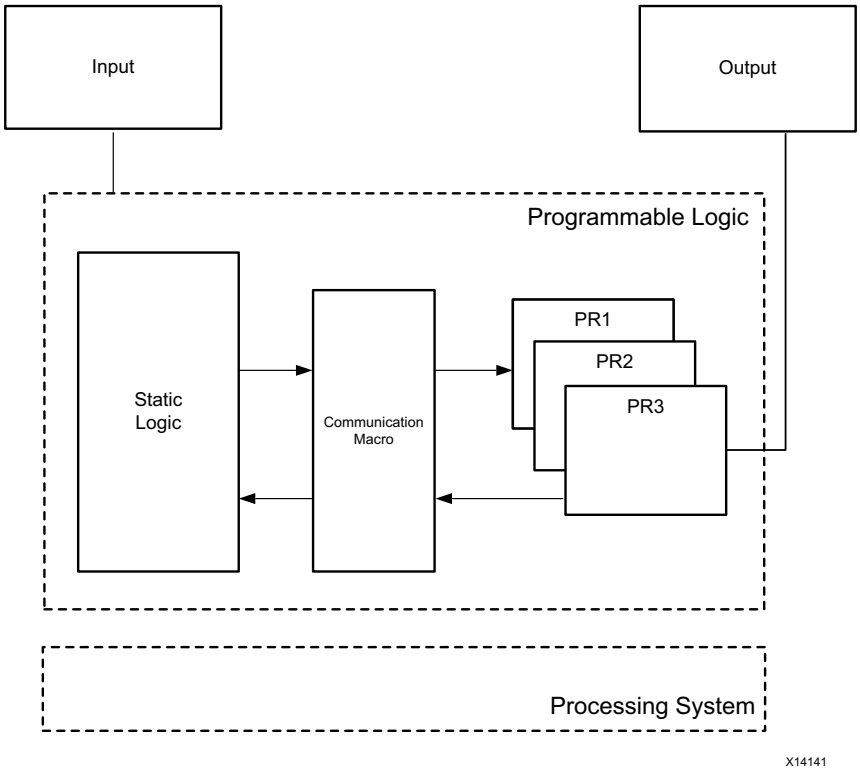

图 3-29 : 部分重配置概念视图

Zynq-7000 AP SoC 有一个 PL 部分，您可以通过部分冲配置来重新配置。通过处理器配置访问端口 (PCAP) 对 PL 进行编程,PCAP 是 PS 器件配置接口的一部分。PCAP 模块通过一个 PCAP 至 APB 桥接 将 APB 事务转换为 PCAP 读/写事务，从而实现与 CPU 和系统存储器的通信。您可以通过赛灵思 Vivado 设计工具存储所生成的部分码流，并在 PCAP 接口 中通过部分冲配置方法配置 PL 时取回。

部分重配置命名

可重新配置分区是指在 FPGA 中为部分重配置选定的物理位置。设计的其余部分称为静态逻辑。可将一个特定的设计实现作为可重新配置模块。配置定义一个完整的 FPGA 设计并且在可重新配置模块和静态逻辑中产生一个全码流，并为可重新配置模块产生一个部分码流。

系统级考虑事项

当您在设计流程中使用部分重配置时，应考虑以下的系统级因素。

建议： 如需了解如何在 Vivado 中通过部分重配置功能开展设计，请参阅 Documentation Navigator 中的部分重配置设计中心。如需了解更多信息，请参阅第 158 页的“相关设计中心”。

硬件设计流程

创建部分重配置设计的第一步是使用可重新配置模块和静态设计流程创建一个设计。部分重配置设计流程的实现需要使用赛灵思 Vivado 设计工具，步骤如下：

- 使用赛灵思 Vivado 设计工具针对评估平台创建部分重配置的项目，并导入网表和约束文件。
- 定义可重新配置分区。这个分区确保了每组多重设计中常用的逻辑与布线是相似的。

- 通过增加相应的网表和约束文件为可重重新配置分区创建可重新配置模块。只适用于特定可重新配置模块的约束就必须限定在模块级，并应提供相应的网表。应用到静态逻辑的约束，以及任何在可重新配置模块中共享的约束应纳入顶层约束文件中。
- 通过设置分区的物理容量和所希望资源的类型，布局规划可重新配置分区。赛灵思 FPGA 支持 CLB （触发器、查找表、分布式 RAM、多路复用器等）， BRAM 和 DSP模块的重新配置，以及所有相关的布线资源。设计人员必须布局规划可重新配置分区，使其能够容纳可重新配置模块所需的资源。出于指导目的， 20% 的开销应假定为布线资源。 PL 内静态逻辑可重新配置分区的位置取决于数据流程和可重新配置模块如何与设计的其余部分进行通信。这里有一个简单的策略，就是以最高的资源利用率实现配置而不依赖布局规划，确定大部分资源都在哪个区域内布局，并围绕这个区域创建一个足以容纳所有资源的分区。
- 构建重新配置设计配置时，实现第一个配置应是最具挑战性的。
- 运行部分重配置验证的配置实用工具来验证重新配置模块实现之间的一致性。
- 为可重新配置模块生成全部和部分码流。

图 3-30 显示了硬件设计流程实现部分重配置情况。

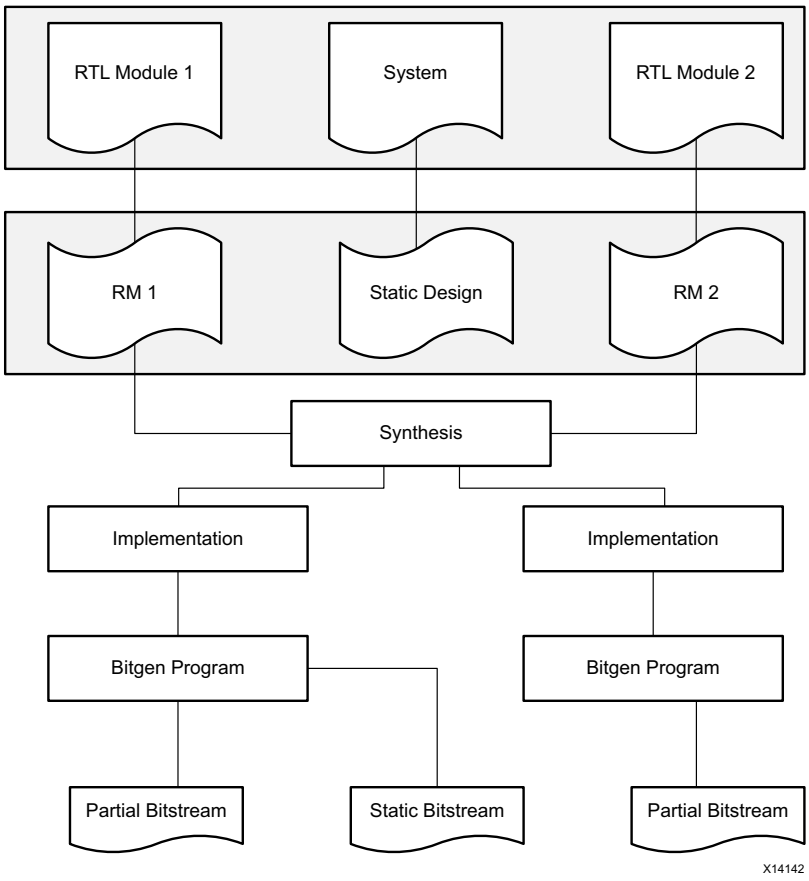

图 3-30 : 硬件设计流程部分重配置

系统设计流程

包含 AXI-PCAP 桥接的器件配置 (DevC) 接口用于实现 Zynq-7000 AP SoC 的部分重配置流程。

下面的示例总结了部分重配置启动顺序：

1. 上电复位后， BootROM 决定了外部储存器接口或启动模式 （SD 闪存）和加密状态 （非安全）。 BootROM 使用 DevC 的 DMA 将第一阶段启动加载程序 (FSBL) 加载到片上 RAM (OCM)。

2. BootROM 关闭并将 CPU 控制释放给 FSBL，后者通过 PCAP 使用全部码流来配置 PL。
3. 在裸机 OS 中， FSBL 加载和释放控制到用户应用。
4. 用户应用在启动时加载部分码流至 DDR 储存器。这是为了最大化 PCAP 接口的配置吞吐量，缩短配置时间，并且充分利用缓存。
5. 应用可以使用部分码流随时修改预先定义的 PL 区域，而 FPGA 的其余部分仍然保持完全活跃和不间断状态。这是通过 PCAP 将可重新配置模块码流从 DDR 传送到 PL 完成的。
6. 单个配置引擎处理全部配置和部分重配置。因为配置帧寻址信息包含在部分码流中，所以加载部分码流到 PL 的任务不需要了解可重新配置模块的位置。

第 90 页的图 3-31 部分重配置系统级流量。赛灵思器件配置驱动实现低级别 API 与 PCAP 块进行交互，用户应用可以调用 API 来启动部分重配置流程。用户应用必须感知到其中部分码流存储在 DDR 源地址中。

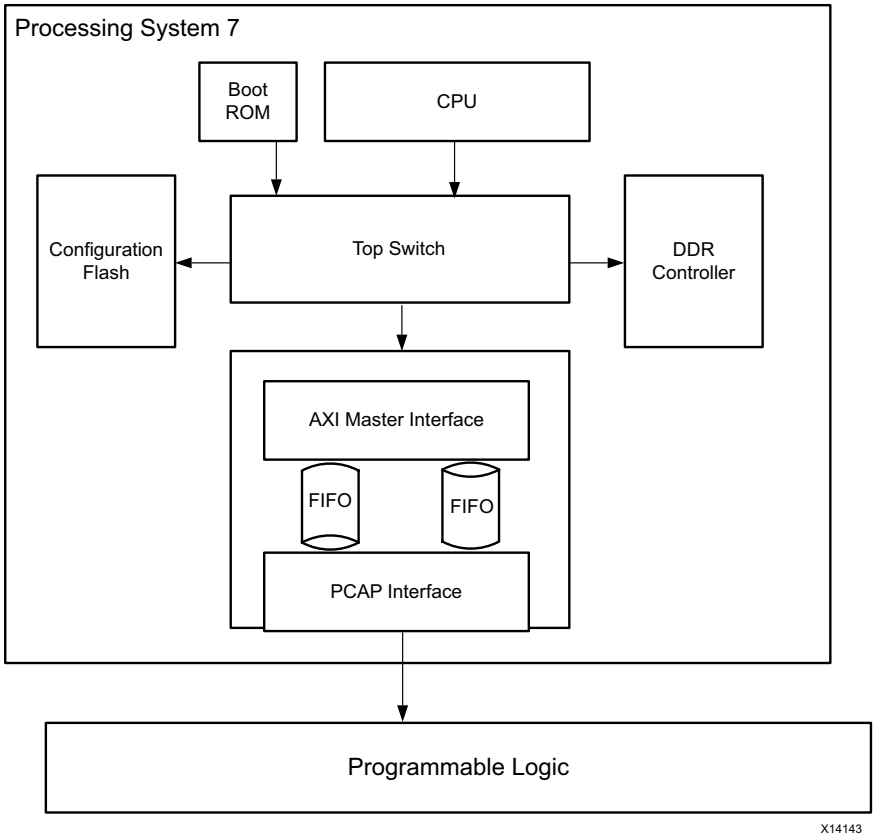

图 3-31 : PR 系统级流程

上面的示例说明了如何从 PS-DDR 储存器中加载部分码流。在一般情况下，码流可以存在于任何器件配置接口能够访问的外部储存器中。

使用 U-Boot 和 Linux 管理重新配置

您可以使用 Linux 和 U-Boot 将可重新配置码流动态加载到 PL。如果应用在 Linux 上运行但是硬件的一部分必须重新配置，应用可以发送请求到 Linux 内核，加载适当的 PL 码流和相应的内核模块来支持硬件器件。

如果 FSBL 执行时间至关重要，您可以使用部分重配置来划分设计，以便较小的码流在 FSBL 执行过程中能够加载更快。这降低了整体的启动时间。 U-Boot 可以对剩余的设计配置并初始化。当特定的 PL 功能必须在特定的时间限制内启动时，分区是非常有用的。

可以使用密钥来加密部分重配置的码流。根据不同的应用需求，您可以使用 U-Boot 或 Linux 解密。一旦解密，可以使用码流通过 PCAP 接口配置 PL。

GP 和来自 APU 的直接 PL 访问

本节将介绍 APU 通过 AXI 通用接口在 PL 内访问寄存器与储存器、 PL 中 AXI 从的抵制要求，以及安全与性能的相关问题。介绍了一种处理 ARM 核直接访问 PL 和典型用例的系统设计。

PS 中的 APU 可以使用 GP 主 AXI 接口 （PS-PL 接口的一部分）访问 PL 中的寄存器和储存器。有两个 GP 主 AXI 接口，APU 可以用他们来对 PL 中实现的从发起一个 AXI 事务。

M_AXI_GP 端口具有以下特点：

- AXI-3 协议。
- 32 位数据总线宽度。
- 12 位主端口 ID 宽度。
- 主端口八个读取和八个写入的发行能力。

如表 3-3 所示， AXI 接口、 M_AXI_GP0 和 M_AXI_GP1 各自占据的系统地址空间为 1GB。

表 3-3 : GP 端口的地址范围

接口	低地址	高地址
M_AXI_GP0	0x40000000	0x7FFFFFFF
M_AXI_GP1	0x80000000	0xBFFFFFFF

如表 3-3 所示，地址范围内的所有的 APU 访问通过 PS 中的互联，根据地址采用 M_AXI_GP0 或 M_AXI_GP1 实现。如果由 APU 发起的事务在这些地址范围之外，事务将不会被布线至主 GP 端口。

图 3-32 显示了 PL 中 APU 访问一个 AXI 从的路径。

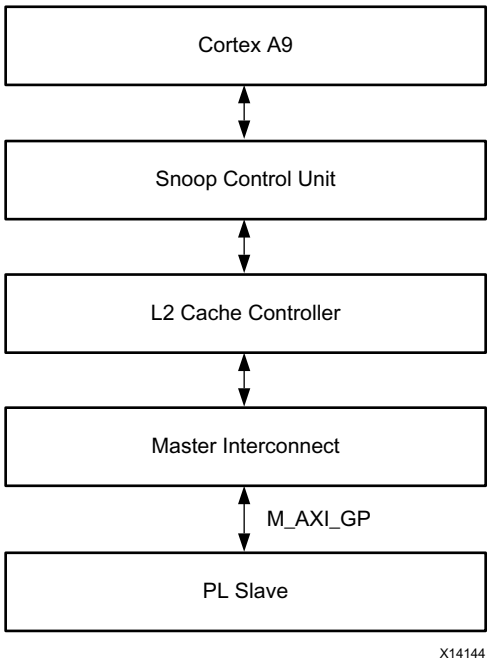

图 3-32 : APU 通过 GP 端口访问 PL 从

性能

与具有 FIFO 缓冲区以提高性能和吞吐量的 HP 端口不同， GP 端口不使用 FIFO 缓存，直接连接到 PS 主互联端口。因此， GP 端口用于一般目的，而不用于高性能目的。 GP 端口具有 38 个 M_AXI_GP 接口时钟周期的时延。

AXI 主 GP 端口的设计考虑因素

- 当 PL 中 AXI 互联连接到 PS GP 主 AXI 接口和 HP AXI 从接口时，您需要禁用来自主端口 GP 的 HP 从端口访问。IP 集成器地址编辑器中，在 processing_system7 寻址下，取消分配 HP 端口地址，从而避免地址冲突。
- 默认情况下访问 GP 接口不进行缓存。您可以通过更改转换后备缓存 (TLB) 进行访问高速缓存。
- 在 PL 中与 AXI 互联一同使用 GP 接口时，系统软件不应访问 AXI 互联中不存在的地址。这将避免永久 AXI 互联锁定 （不存在由 AXI 互联设置的超时）。
- 在 APU 使用主 GP 接口访问 PL 中的 AXI 从之前，必须对 PS-PL 边界的电压转换器进行配置。这通常是由 FSBL 完成。

用例

图 3-33 显示了 PS 上 GP 主的 AXI 接口如何通过访问 PL 中的 AXI 从寄存器来配置外设\读取状态，并访问 DDR 储存器。

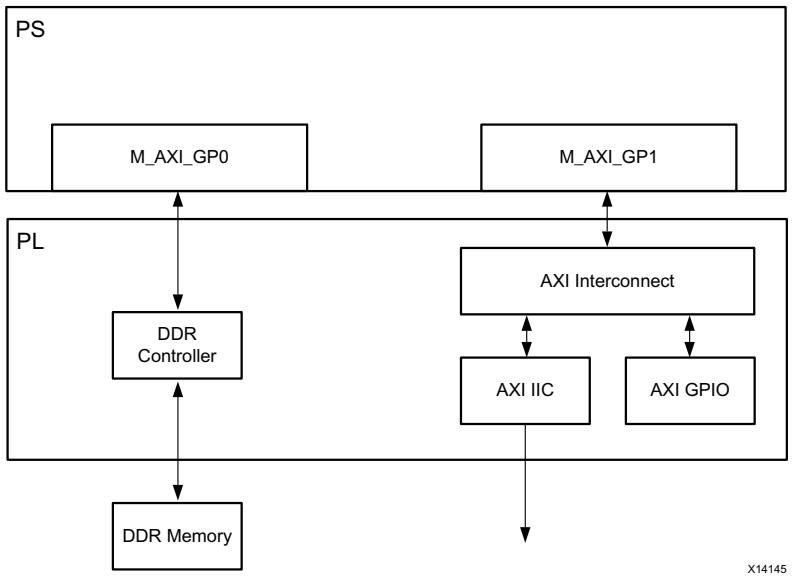

图 3-33 : APU 访问 PL 中的 AXI 从

GP0 端口与 DDR 控制器连接,用于通过 PL 中的 DDR 控制器 增加 PS 可用存储器。 GP1 端口用于配置 AXI IIC 和 AXI GPIO IP 模块。必须被改变 TLB，以允许 PL DDR 储存器进行高速缓存。

XILINX
ALL PROGRAMMABLE™

第 4 章

软件设计考虑事项

本章探讨下列在使用 Zynq®-7000 AP SoC 设计时需要考虑的软件设计问题：

- 第 93 页的“处理器配置”：对设计嵌入式系统而言，配置处理器、使之满足系统要求是一项重要的考量。本章的目的是帮助您优化 ARM 核心，使之最符合您的应用需求。
- 第 97 页的“OS 和 RTOS 选择”：本章的目的是在为您的嵌入式系统选择合适的软件平台时 （如 OS、 RTOS 或裸机时）帮助您理解各种优势与缺点。
- 第 104 页的“库和中间件”：库帮助您把通用功能和有用功能结合到一个地方，方便您的应用重复使用它们。中间件位于操作系统顶端，通过将原本在应用层中导致冗余的软件栈集中统一来降低复杂性。本章节介绍了您能如何运用库和中间件来提高应用编程的生产力和可靠性。
- 第 106 页的“启动加载器”：在您的嵌入式系统中，您有多种途径可以用于启动和启动应用。本章节介绍从加电到应用执行的 Zynq-7000 AP SoC 启动流程选项。
- 第 110 页的“软件开发工具”：本章节介绍可以用于为您的 Zynq-7000 AP SoC 构建软件组件的赛灵思软件开发工具。

处理器配置

要在嵌入式系统中完整地使用 SoC，必须要配置处理器以满足系统要求。本章节介绍了需要考虑的各个方面。本章节的内容专门针对 Zynq-7000 AP SoC 系统，但也可以应用于任何系统。

时钟速度和多核心

在功耗与性能之间取得良好平衡能让嵌入式系统极为有效与高效。处理器可配置为以低于最大限值的时钟频率运行，从而节省功耗。如果操作系统支持多个处理器核心，那么让多个核心以较低频率运行而非让单个核心以高频率运行可以起到节省功耗的作用。通过分析功耗要求和运行在较低时钟频率下的多个处理器核心的功耗，并与运行在高时钟频率下的单个核心的功耗做比较，可以确定功耗得到了节省。

处理器的时钟频率可使用系统软件动态调整。在同时运行两个处理器核心时，它们将一直以同频率运行。跨多个核心的多线程软件有助于充分发挥多核系统的作用。这样的软件通过并行执行线程提升性能，避免迫使系统采用最大时钟频率运行。

SMP 和 AMP 配置

使用多核处理器的决定会影响包括操作系统选择在内的软件系统设计。大多数多核 SoC 提供将系统作为非对称多核处理器 (AMP) 或对称多核处理器 (SMP) 运行的选项。

AMP 配置将每个处理器核心用于不同的 （一般是无关的）任务。每个核心独立工作，有时候向其他内核发送信号或与其他内核同步以交换消息。一般情况下每个核心都有自己的储存器空间，用于传递消息的储存器除外。实践中 AMP 配置一般用于运行不同的操作系统或同一操作系统的不同的独立实例。

SMP 配置使用多个处理器核心在同一操作环境中完成多项任务，比如当多核 Linux 同时运行多重任务时。通用的调度器负责把任务派发到不同的处理器内核上。因为系统把每个核心视为相似的核心，故这种运行是对称的。

不同的 AMP 和 SMP 配置使用不同的缓存一致性和储存器共享设计。使用处理器核心的系统控制专用寄存器 （例如 ARM v7 核心中的 CP15）就能够控制系统的 AMP 或 SMP 行为。

图 4-1 和图 4-2 分别显示了 AMP 和 SMP 系统的典型示例 （“处理器”指的是“处理器核心”）。

图 4-1 所示的是两个 AMP 的示例：

- 在示例 1 中，每个处理器分配有专用储存。该储存器不在处理器间共享。两个处理器都能访问公共储存器和一套外设。
- 在示例 2 中，一个处理器负责全部的储存器运算，但不能访问外设。第二个处理器负责访问外设但不负责开展储存器运算。

在图 4-2 中，两个处理器对所有外设和储存器有均等的访问权限。

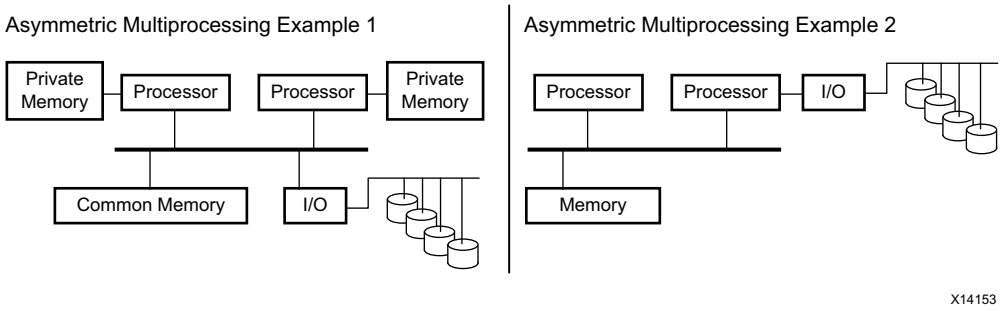

图 4-1 : 典型 AMP 配置

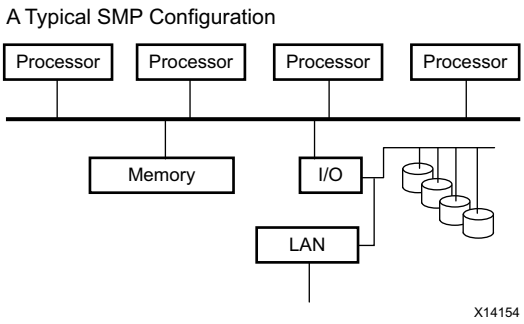

图 4-2 : 典型 SMP 配置

使用协同处理器

硬件协同处理器能够提高系统的性能。例如，如果系统要执行大量浮点或 SIMD 运算，使用 NEON 专用指令就能够改善性能。这些指令使用 ARM Cortex-A9 MPCore 处理器中的 NEON 媒体协同处理器功能。

虽然 NEON 技术针对功耗优化，但大多数协同处理器功耗较大。而且由于协同处理器是为特定任务设计的，与使用主处理器执行这些任务相比，它们速度更快，能效更高。在工作负荷下降时协同处理器可减少 CPU 带宽，让 CPU 降低功耗。

协同处理器执行专为其用途设计的特定指令。例如 《Zynq-7000 AP SoC 频谱分析器第二部分：构建 ARM NEON 库技术提示》[参照 62] 说明了如何使用 NEON 库指向 Zynq-7000 AP SoC 的 NEON 功能。

缓存考量

缓存能显著地改善系统性能，应尽量采用缓存。大多数 SoC 已集成 L1 和 L2 缓存。每个 CPU 核心一般有能单独配置供该核心使用的专用 L1 缓存。 L2 缓存一般由所有 CPU 核心共享。缓存间的一致性使用侦测控制单元(SCU)维持。一致性决策由 AMP 或 SMP 多处理器配置决定。

缓存在启用之前应先使之失效。 L1 指令缓存必须在启动流程的开始启用 （一般由第一阶段启动加载器），此后不应禁用。 L1 和 L2 数据缓存可以在稍后配置和启用。

所有缓存操作都在缓存行中完成。缓存控制器可以按缓存通道或物理地址写回 （清除）和使缓存行失效。

一般来说 SoC 内含次序不同的储存器 （例如高次序储存器或设备储存器），且系统能够或者不能在该储存器上使用缓存操作。另外还可以有缓存策略，比如与具体程序挂钩的写通或写回。

如果任何系统外设使用与 CPU 共享的储存器，该储存器和 CPU 缓存间的一致性应由软件使用缓存操作 （代价高昂）来维持。类似的，在某些复杂系统中，外设可以访问已缓存的储存器，以改善性能。在这种情况下，外设缓存、储存器和 CPU 缓存间的一致性应由软件维持。维持一致性产生的软件开销会带来时延。使用缓存一致端口提供的硬件一致性可以避免这种时延，比如使用加速器一致端口 (ACP)。

Zynq-7000 AP SoC 系统提供的 ACP 可让外部端口访问处理器缓存。ACP 还可以让其他未缓存的主外设和加速器，例如 DMA 引擎或加密加速器内核与处理器核心 L1 缓存保持缓存一致性。

加电重置后的处理器状态

在 Zynq-7000 AP SoC 中，两个处理器核心都在加电重置 (POR) 后启动。 CPU0 在地址 0x0 处启动，这也是 BootROM 加载第一阶段启动加载器的位置。CPU1 处于等待事件 (WFE) 循环中。任何中断都会导致 CPU1 唤醒并退出 WFE 状态。启动加载器、内核或等效软件一般负责在之后，即准备使用 CPU1 的时候启动 CPU1。

在大多数情况下 DDR 储存器不会在 POR 时初始化，因此第一阶段启动加载器先加载到较简单储存器中，例如片上储存器，然后再映射到 SoC 中的地址 0x0。

ARM 处理器在 POR 后在管理员 (SVC) 模式下启动。随后内核或等效软件根据需要修改 ARM 处理器的模式。

中断处理

ARM 核心提供的是通用中断控制器 (GIC) 和专用外设中断 (PPI) 控制器用于处理中断。 GIC 对处理器间通信以及系统中断的路由和优先排序提供灵活的方法。软件能够控制多达 224 个独立中断。每个中断都能分配给一个或同时两个处理器核心，经硬件优先次序排序后，在操作系统和 ARM TrustZone 技术软件管理层之间路由 （见第 96 页的“安全配置”）。同时还提供优先级屏蔽寄存器，让低于特定优先级的中断被忽略。

除了 GIC，处理器核心可通过在处理器系统控制寄存器中设置中断屏蔽，屏蔽所有进入的中断。在 ARM 处理器中，IRQ 和 FIQ 信号可对每个处理器核心独立屏蔽。

中断处理器应在内核启动之初即完成配置。有三种可能的中断源：

- 专用外设中断 (PPI)：

  这类隶属单个处理器核心的中断功能有限。它们包括 FIQ、CPU 看门狗、CPU 定时器和部分可能包含来自 FPGA 逻辑的中断信号的 IRQ。

- 共享外设中断 (SPI)：

这些中断在处理器核心之间共享。 GIC 决定如何分配 SPI 给特定处理器核心。这类中断一般是硬件中断，数量可能会非常大。

- 软件生成中断 (SGI)：

  这些属于通过在系统上的特定 SGI 寄存器上写入中断数值而生成的同步中断。

您可以配置 GIC 寄存器以处理上面列出的中断。图 4-3 显示的是 GIC V1.0 (PL310) 的中断路由机制。

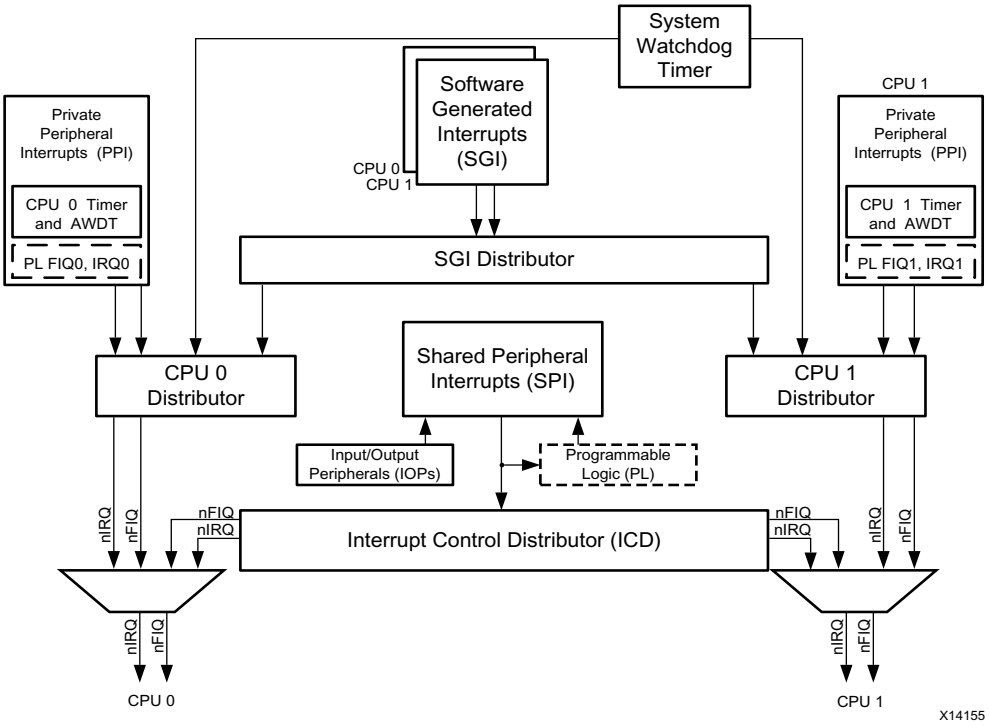

图 4-3 : 中断处理

定时器

每个处理器核心都拥有能在该处理器核心情景下使用的专用定时器。这些定时器对跟踪核心而非整个系统的运行时很有帮助。该 SoC 还拥有提供系统层面时序信息的全局时钟。这些定时器生成的中断属于专用外设中断 (PPI) 并依每个处理器核心分配。时钟中断往往对系统运行十分重要，因为它们被用于调度任务和为任务安排时间片。

MMU 配置

Linux 等大多数操作系统要求使用虚拟储存器系统。储存器管理单元 (MMU) 是一种硬件模块，能便利对虚拟储存器的访问。 MMU 使用页表等数据结构转换地址。操作系统内核必须配置 MMU 并设置页表来实现虚拟储存器系统。不同的操作系统模式要求不同的页表，例如 TrustZone 技术的安全模式。

安全配置

Zynq-7000 AP SoC 通过运用 TrustZone 技术机制提供把系统划分为安全和非安全两个环境的选项。处理器模式必须切换为监测模式 （安全模式）才能进入安全环境，进行安全操作。另外对不同的环境必须提供单独的 MMU 配置。

关于 TrustZone 技术的更详细介绍，请参阅第 85 页的“TrustZone 安全性”。

OS 和 RTOS 选择

操作系统软件管理计算系统的计算和保护计算系统的硬件资源。 OS 或 RTOS 能支持多任务并管理资源使用。裸机软件可起到必要的系统管理功能的最小实现的作用。本章节介绍系统管理类型的对比、硬件要求、可用的 OS 和 RTOS 端口、运行多个 OS 以及开发和调试端口。

在独立操作系统和微内核之间还有安全考量。像 Linux 这样的全功能独立操作系统将 MMU 用于安全和非安全储存器领域，而微内核 RTOS 不使用 MMU，故不能提供与独立内核相同的储存器保护。详细内容请参阅“赛灵思安全解决方案”网页 [参照 46]。

嵌入式系统软件类型

表 4-1 比较的是裸机、实时操作系统 （RTOS）和操作系统 （OS）系统管理软件的特性。

表 4-1 : 裸机软件、实时操作系统 （RTOS）和操作系统 （ OS）的比较

	裸机软件	实时操作系统 （RTOS）	通用操作系统 （OS）
功能	简单	简单到高级	高级
硬件抽象	无	最小	高级
多任务	无	有	有
实时	有保证响应时间	有保证响应时间	尽快
储存器管理	无	基本储存器保护	先进虚拟 储存器管理
代码大小	小 (KB)	小到中 （KB 到 MB）	中到大 (MB)
示例	有 SDK 的裸机软件	VxWorks、 FreeRTOS	Windows 嵌入、 Linux
开源考虑	随 Vivado® 和 SDK 工具提供。这些工具要求专门的许可证。	部分随开源许可证提供，部分为付费版本。	部分随开源许可证提供，部分为付费版本。

系统管理软件的选择取决于下列因素：

- 以往经验和/或现有解决方案：这种选择系统管理软件的方法节省时间和验证工作量。它还能简化采用现有 OS 的变形开展的系统设计或采用有附加特性的 OS 开展的系统设计。这是因为使用拥有不同的受支持特性集的新 OS 或 RTOS 学习难度大。
- 功能：OS 和 RTOS 在其特性和中间件支持方面存在差异。应该根据总体系统设计的需求选择其一，比如为 USB、TCP/IP 和文件系统支持选择中间件栈。
- 可配置性：部分 RTOS 拥有基于 GUI 的配置选项，用于启用缓存、为中断优先级排序、选择 RTOS 时钟周期等。功能完善的 OS，例如 Linux 可以使用诸如 `menuconfig` 等工具配置特性。这些配置接口和工具可节省时间并突出显示可用的特性选项。
- 性能：不同 OS 和 RTOS 之间的中断和任务切换时延可能不同。在选择 OS 或 RTOS 时应细致地考查硬实时时延、软实时时延和总体时延。
- 储存器大小：在选择 OS 或 RTOS 时需要估算对 OS、 RTOS 和应用任务的储存器大小要求。
- 成本：可用基于 GNU 许可证的免费 OS 或 RTOS。部分厂商会根据自己的支持和服务模式收费：
  - 裸机 BSP 和驱动程序随 SDK 等赛灵思开发环境免费提供。
  - 根据特性和支持模式，可根据免费或付费商业许可证选择 RTOS。
  - 根据特性和支持模式，可根据免费或付费商业许可证选择 Linux。
  - 商业版本会附带支持和更多特性，例如 USB 中间件栈或 RTOS 文件系统栈。

- 应用要求：
  - 如果应用需要的是拥有现成库的高级软件，就考虑全功能 OS，比如 Linux。
  - 如果应用需要的对大量任务的硬实时响应，就考虑像 freeRTOS 这样的 RTOS。
  - 如果应用既需要全功能 OS 和实时响应，就考虑多 OS 环境，如带实时补丁的 Linux 对称多处理版或让 Linux 和 RTOS 运行在双 ARM 核心或程序管理器上的非对称多处理 (AMP)。

运行 OS 或 RTOS 的硬件要求

OS 或 RTOS 有如下要求：

- 系统时钟：用于内核计时的硬件时钟，例如 Linux OS 中的基菲。该时钟被 OS 或 RTOS 用于所有时间相关的内核活动，如调度和时延。
- 储存器占用：对 RTOS 应低至 5 到 30 KB。对 Linux OS 根据内核和服务配置选择，应为数 KB 到数百 MB。
- 储存器管理单元 (MMU)：Linux 等全功能通用 OS 需要。也可用于某些 RTOS 设计的储存器保护。

Zynq-7000 AP SoC 提供运行 RTOS 和/或 Linux 等完整功能 OS 的全部硬件资源。实现实时解决方案的时间确定性取决于许多因素，除 RTOS 外还包括资源分配和访问时间、总体解决方案架构等。

OS 和 RTOS 端口

下面是对几个到 Zynq-7000 AP SoC 的示例 OS 和 RTOS 端口的介绍，但不包含所有端口。

Linux OS 和 PetaLinux 工具

赛灵思 Linux 版本包括 Linux OS 和用于 Zynq-7000 AP SoC 的完整配置、构建和部署环境。因为 PetaLinux 功能齐全且经集成、测试和文档化，是向赛灵思客户推荐并提供支持的 Linux 部署机制。作为一款赛灵思产品， PetaLinux 得到的产品管理、支持、缺陷跟踪和重视水平上与其他赛灵思工具一样。

虽然使用 PetaLinux 工具构建的 Linux OS 基于稳定、经充分测试和来自赛灵思 Git 服务器的 Linux 内核，PetaLinux 提供的内容远远超过从赛灵思 Git 服务器下载的内容。 PetaLinux 包含一个安装程序、一套开发工具、板级支持包 (BSP)、平台管理实用工具、应用和库框架以及文档。 PetaLinux 的提供免费且无需商业许可证，如 PetaLinux 许可页上的介绍。所有 PetaLinux 用户都有权获得社区 （基于论坛的）支持；商业许可用户还可以直接从赛灵思获得授权。详细介绍请参阅赛灵思维基页面 [参照 54] 和赛灵思 PetaLinux 维基页面 [参照 56]。

赛灵思的PetaLinux工具支持为 Zynq-7000 AP SoC 上的全功能 OS 配置、定制、构建和封装 Linux BSP。关于如何配置和使用这些 BSP 的详细说明，请参阅赛灵思 PetaLinux 工具网站 [参照 48]。

OS 和 RTOS 生态系统

除了上文介绍的赛灵思 OS 和 RTOS 端口， Zynq-7000 AP SoC 拥有来自第三方和联盟成员的强大生态系统支持产品。这些产品在受支持特性、情景切换性能、中断时延、技术支持和对文件系统、USB、 PCIe、以太网等的中间件支持方面可能存在差异。

表 4-2 和下列章节介绍的是第三方和联盟成员生态系统解决方案。更详细说明请参阅赛灵思 Zynq-7000 AP SoC 生态系统网页 [参照 47]。

表 4-2 : OS、 RTOS 和中间件产品

联盟成员	OS、 RTOS 和中间件产品	更多信息
Adeneo Embedded	Windows Embedded Compact 7, Linux, Android, and QNX	[参照 84]
Discretix	Security-centric software and IP solutions	[参照 77]
ENEA Software AB	OSE RTOS and ENEA Linux	[参照 78]
eSOL	uITRON 4.0 RTOS, T-Kernel RTOS and IDE	[参照 79]

表 4-2 : OS、 RTOS 和中间件产品 （续）

联盟成员	OS、 RTOS 和中间件产品	更多信息
Green Hills Software	INTEGRITY RTOS	[参照 81]
Express Logic	ThreadX RTOS	[参照 80]
iVeia	Android On Zynq-7000 AP SoC	[参照 84]
Mentor Graphics	Nucleus RTOS	[参照 85]
Micrium	µC/OS RTOS	[参照 86]
MontaVista Software	MontaVista Carrier Grade Linux	[参照 87]
Open Kernel Labs	OKL4 Microvisor	[参照 90]
QNX	QNX Neutrino RTOS	[参照 91]
Quadros	RTXC RTOS 关于特性的介绍，请参阅第 99 页的“Quadros Systems 特性”。	[参照 92]
Real Time Engineers Ltd.	FreeRTOS	[参照 93]
Sciopta	Sciopta RTOS	[参照 94]
Sierraware	开源 SierraVisor 程序管理器和 SierraTEE 受信任执行环境	[参照 95]
SYSGO	PikeOS	[参照 96] [参照 97]
Timesys	LinuxLink	[参照 99]
Wind River	VxWorks、 Linux 和 Workbench IDE	[参照 100]

Quadros Systems 特性

来自 Quadros Systems 的 RTXC RTOS 拥有下列特性：

- 储存器占用小、可扩展，能适合任何ROM或RAM预算 （一般为 25 KB 或更小）
- 以源代码方式交付并为使用 ARM DS-5 开发工具的工作项目提供样本代码
- 为构建应用的 RTOS 提供便利的图形配置程序 RTXCgen
- 用于查看系统性能和行为的图像跟踪工具 RTXCview
- 用于非对称多处理 (AMP) 的 RTXC/mp
- 需要许可证费

除 RTXC RTOS 外还支持下列中间件栈：

- 以太网 TCP/IP v4 和 v6
- 以太网数据包优先排序
- 以太网安全套件
- 应用服务器：
  - AJAX、 JSON 和 XML-RPC
  - 带永久性套接口的远程设备管理 (WebSockets)
  - 远程 GUI
- USB 主机和设备
- 高性能故障安全文件系统：
  - NAND 和 NOR

- SD、 MMC 和闪存驱动器

针对 Zynq-7000 AP SoC 和 ZC702 开发板的支持：

- 时钟初始化
- UART 初始化
- MMU 初始化
- 中断控制器设置和初始化
- 指令和数据缓存初始化。
- 针对ARM向量浮点 (VFP) 和 NEON 的高级支持：
  - 为使用 VFP 或 NEON 的任务明示地或自动启用VFP寄存器情景扩展功能。
  - 可选择支持能降低情景切换开销的 VFP 和 NEON 寄存器集。
  - 使用“惰性倒换”(lazy swapping) 为 VFP 和 NEON 寄存器集最大程度地降低中断时延和情景切换开销。
  - 控制 VFP 情景，比如手工暂停以及自动或手工恢复。精细控制让应用最大限度降低非 VFP 和非 NEON 执行路径中的 VFP 寄存器情景开销。在任务几乎不需要 VFP 或 NEON 的大多数整数环境中有用，除了在被隔离的功能中。在与“惰性倒换”结合使用的情况下，可将明示暂停控制功能用于改善总体系统的吞吐能力。

Real Time Engineers Ltd.

FreeRTOS 无需暴露定制代码即可免费使用。 FreeRTOS 支持实时任务、队列、二进制信号标、计数信号标、互斥元等，用于任务和中断服务例行程序之间的通信与同步。该 FreeRTOS 的储存器占用约为 4 到 9 KB。用 freeRTOS 支持的中间件栈包括：

- FAT 文件系统
- UDP/IP 和 TCP/IP 栈
- CLI
- 安全性
- CyaSSL SSL/TSL

多 OS

有多种方式可以实现同时运行在 Zynq-7000 AP SoC 中的双核 ARM Cortex-A9 MPCore 处理器的两个 CPU 核心上的多个操作系统。请查阅下列链接了解关于对称和非对称多处理选项的信息：

- 《Zynq-7000 All Programmable SoC 软件开发人员指南》(UG821) [参照 7]
- 赛灵思多 OS 支持维基网页[参照 55]

简单 AMP

简单的多 OS AMP 设计的组成可包含运行一套 Linux OS 和一个裸机应用，也可以是运行两个裸机应用。

ARM 处理器的两个 CPU 核心共享储存器和外设。非对称多核处理器 （AMP）是一种让两个核心同时运行自己的操作系统或裸机应用的机制，并提供通过共享资源松弛耦合这些应用的可能性。图 4-4 显示的是 AMP 搭配 Linux OS 和裸机应用的示例。第 102 页的图 4-5 显示的是 AMP 搭配 Linux OS 和 RTOS 的示例。

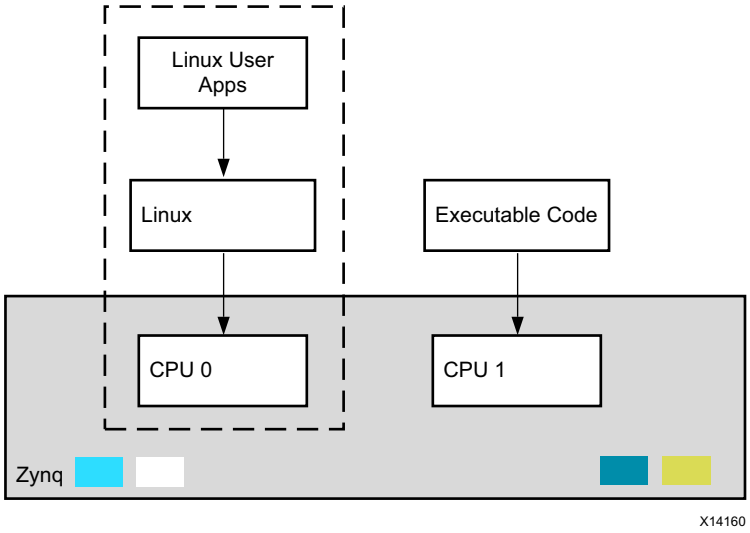

图 4-4 : AMP 搭配 Linux OS 和裸机应用

根据图 4-4， AMP 搭配 Linux OS 和裸机应用的运行过程如下：

- Linux 运行在 CPU0 上。
  - Linux 随机启动由 FSBL 加载到储存器中的 CPU1 可执行文件。
- 逻辑应用运行在 CPU1上 的专用储存器空间里。
  - MMU0 由 Linux 按正常方式使用。 MMU1 用于定义裸机应用的储存器情景，但无需专门编码。
  - 逻辑应用不在 Linux 储存器情景内运行。
  - Linux 不能感知 CPU1 使用的储存器。

MMU 能用于约束应用。

  - MMU 定义的是应用能够正常访问的地址 （储存器和 AXI 器件）。
  - CPU1 对 Linux 储存器空间或共享器件 (ICD 或 SCU) 访问不受限制，但 Linux 能够检测是否有限制发生，发生的时间并采取适当的措施。
- 实现方法。
  - Linux 和裸机应用之间的通信使用 OCM。
  - 为改善确定性， OCM 缓存被禁用。
- 该用例得到赛灵思全球技术支持部的支持。

更多介绍请参阅：

- 简单 AMP：《同时运行在两个 Cortex-A9 处理器上的裸机系统》(XAPP1079) [参照 37]
- 《同时在两个 Zynq-7000 AP SoC 处理器上运行 Linux 和裸机系统的简单 AMP》(XAPP1078) [参照 36]

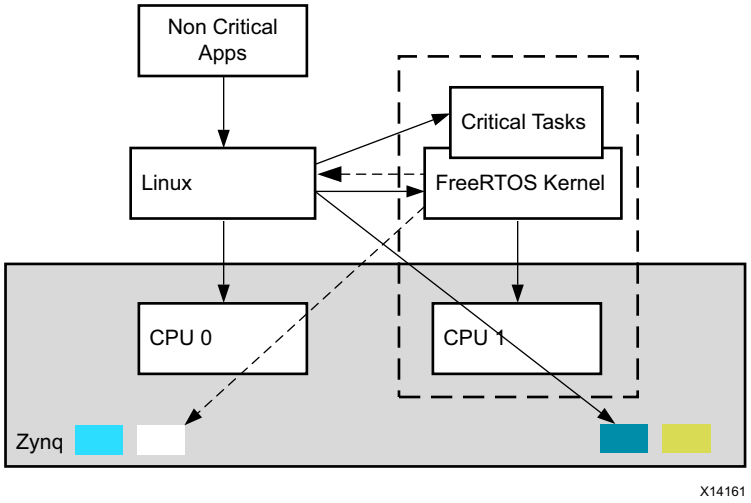

图 4-5 : 使用 inux OS 和 RTOS 的 AMP

参阅图 4-5，配套 Linux OS RTOS 的 AMP工作方式如下：

- 该固件运行在地址 0x0000_0000 的专用储存器空间里。
  - FreeRTOS 内核不会运行在储存器情景内。
  - Linux 能够和读写其他任何设备储存器一样读写本储存器空间。
- MMU 会限制固件影响 Linux 的方式。
  - MMU 定制的是应用能够正常访问的地址 （存储器和 AXI 设备）。
  - 粒度为 1 MB。如果两个设备位于一个 1 MB 区域内，您就无法将它们彼此隔离。
  - 该 GIC 被固件和 Linux 共享。
    - 中断由 GIC 路由到固件或 Linux。
    - 固件可以对 GIC 进行重配置操作。
- Linux 内核未被阻止破坏固件。
  - 但是只有明确的调用或通过特定类型的崩溃才会发生冲突。
  - 系统设计和综合测试可能能减轻客户的现实问题。
- 实现方法。
  - 应可应用于其他扁平型 RTOS。
  - 可能也可应用于使用虚拟储存器 （例如内核和用户空间） RTOS。

关于第 102 页的图 4-5 所示的 Linux OS 和 FreeRTOS AMP 设计的总体介绍，请参阅 《PetaLinux工具用户指南：Zynq All Programmable SoC Linux-FreeRTOS AMP 指南》(UG978) [参照 17]。

ARM Cortex-A9 TrustZone

根据“受信任基础系统架构”(Trusted Base System Architecture) 规格的建议， Zynq-7000 AP SoC 支持 ARM Cortex-A9 TrustZone 技术。该技术能开发支持“受信任执行环境”(TEE) 和安全感知应用及安全服务或受信任应用 (TA) 的平台。TEE 是一种小型安全内核，一般使用符合“全球平台”行业论坛制定的 TEE 规格的标准 API 开发。

TrustZone 技术通过创建工作模式，支持开发“正常”域之外的单独的“富操作系统”(Rich Operating System) 和 TEE 环境，也即所称的“监测”模式和“安全”域。 “安全”域与正常域有相同的功能，但工作在单独的储存器空间里。 “安全监测器”的作用是虚拟门卫，用于控制域之间的迁移。

开发和调试工具

可用于支持您的应用的开发和调试工具见如下描述。

赛灵思软件开发套件

赛灵思软件开发套件 (SDK) 提供多种类型的赛灵思软件包，包括用于开发软件平台的驱动程序、库、板支持包和完整的操作系统。

提示： Documentation Navigator 内的 Software Development Kit Design Hub 提供关于使用软件开发套件和嵌入式设计的附加说明的链接。更多说明请参阅第 158 页的“相关设计中心”。

赛灵思 SDK 是一个完整的开发环境，包括：

- 平台感知 BSP 生成
- 从储存器测试到 TCP/IP 回显服务器的样本应用代码生成
- 交叉编译工具
- 应用配置
- 连接器脚本生成
- 通过 JTAG 的调试接口
- 目标通信框架
- 远程调试
- 异构多核调试
- 闪存加载器
- 用于安全或非安全文件的启动文件创建
- 第一阶段启动加载器 (FSBL) 生成

SDK使用目标通信框架 (TCF) 代理和 GNU GDB 基础设施通过 JTAG 和应用线程级调试支持 Linux 内核调试。

ARM DS-5 Development Studio

ARM DS-5 Development Studio 是一个支持 Linux、安卓和 RTOS 调试的完整开发环境。关于使用采用 Zynq-7000 AP SoC 的 DS-5 的详细说明，请参阅 《赛灵思 XC702 DS-5 连接指南》[参照 101] 和 《使用 ARM DS-5 工具链的 Zynq-7000 平台软件开发》(XAPP1185) [参照 42]。

IAR

IAR Systems 的调试器支持 OS 和 RTOS 内核级调试。关于更多介绍，请参阅“IAR 集成解决方案合作伙伴项目”网站 [参照 83]。

库和中间件

本章节总结了如何使用静态和动态库及中间件改善应用编程生产力和可靠性。

库

库是一种把通用和有用功能结合到可供程序再使用的单个单元中的便利方法。库能抽象设计细节并提供能方便地结合到程序中的经测试经优化行为。库支持静态和动态链接。

库可以分类为通用库和域专用库：

- 通用库的示例包括 GNU C 库、 GNU C++ 库、 POSIX Pthread 库。
- 域专用库的示例有 Intel 流水线化构建模块 （并行处理）、 FFmpeg 音频、 OpenCV、直接渲染管理器 (DRM)、内核模式设置(KMS)、 ARM OpenMAX DL 样本软件库 （音频/视频处理）和 MATLAB 引擎库 （数学处理）。

库的优势

- 提供行为和代码可重用性 （编写一次，多次使用方法）。
- 通过抽象底层硬件和软件加快应用开发。
- 提供能降低测试资源要求的易于维护的代码。

例如 GNU C 语言库测试完善且定期修复缺陷。因此使用库可节省可观的测试和开发资源。

用于独立系统的赛灵思库

赛灵思软件开发套件 (SDK) 为 Zynq-7000 AP SoC 处理系统 (PS) 和可编程逻辑 (PL) 组件提供一套可重复使用的库和驱动程序。这些库和驱动程序包括：

- libxil.a — 用于外设的设备驱动程序。
- LibXil MFS — 储存器文件系统。
- LibXil FFS — 基于开源实现的通用 FAT 文件系统。该 SDK 主要配套 SD/eMMC 驱动程序使用。为把该 SDK 与此驱动器链接起来，实现了一层胶合层。
- LibXil 闪存 — 一个为并行闪存装置提供读取、写入、擦除、锁定、解锁和设备专用功能的库。
- LibXil Isf — 一个支持 SPI/QSPI 上的赛灵思系统内闪存硬件和串行闪存的系统内闪存库。
- LibXil SKey — LibXil SKey 库为用户定制 eFUSE 位提供编程机制。 PS eFUSE 用于保持启用或禁用部分 Zynq-7000 AP SoC 处理器特性的RSA主键散列位和用户特性位。
- lwIP — 第三方轻量级 TCP/IP 联网库。

根据链接对象的类型，库分为两种类型：静态和动态。

静态库

静态库是在链接后成为应用镜像组成部分的对象文件的集合。在 Linux 中这些库被称为存档。静态库的文件名称常规以 `.a` 后缀结尾。

优势

- 易于使用；外部相依关系在编译时完成解析。
- 编译时间较短。

- 运行略快。理论上连接到可执行文件的静态 ELF 库中的代码运行速度比共享库或动态加载库快 1% 到 5%。
- 因为链接器负责应用要求的对象文件搜索和提取，可简化生成文件 (makefile) 链接步骤。

不足

- 修改静态文件要求重建所有相依应用。
- 可执行文件储存器占用增大。
- 如果共享对象库的进程同时运行，对象库的代码和数据段会被拷贝到每个进程的地址空间。

何时使用

静态链接往往是分配应用的最简单方式，因为它让用户免除了为目标应用分配单独库的繁琐。

动态库

动态库在使用它们的应用执行的时候加载。对象库的单个拷贝在多个应用间共享，因此库代码和数据段不成为应用的地址空间的组成部分。共享对象未被拷贝到每个应用程序区域。而是在要求该库的第一个应用执行时，在运行状态下把对象库的单个拷贝加载到储存器中。

在 Linux 中，动态库文件的命名规则是 `lib` 前缀、库名称和字串 `.so` 并紧接随接口或设计变化增加的版本号。

优势

- 与静态库相比可执行储存器占用较小。
- 如果对象库变化，相依应用只需重新执行。

不足

- 库依赖于运行时加载器。在 Linux 中，动态加载器 (id.so) 是 GNU C 库的组成部分，负责加载所有的相依共享文件。
- 符号重定位在运行中完成。因此使用共享动态库的应用的执行用时比使用静态库的应用略长。

在目标平台上运行可执行文件需要把定制共享库路径添加到动态连接器的搜索路径中。 LD_LIBRARY_PATH 环境变量用为运行时连接器指定定制共享库驻留的部分特定位置。

何时使用

当在多个应用间共享库时使用动态库。例如在这种情况下使用GNU库等嵌入式运行库。

演示 ZC702 平台上的 ARM NEON 库的赛灵思 Zynq-7000 AP SoC 技术提示中包含了这样一个用例。请参阅 《赛灵思 Zynq-7000 AP SoC 频谱分析器第二部分：构建 ARM NEON 库技术提示》 [参照 62]。

该技术提示描述了取得和构建指向 Zynq-7000 AP SoC ZC702 平台的一套滤波功能的过程。许多可充分发挥 Zynq-7000 AP SoC 的处理功能的应用都涉及滤波、视频处理和信号处理中使用的复杂计算。

ARM 社区内的开源项目提供了一个由 NEON 加速的通用有用功能库，可供您用于开发应用。该库已经发展为 Ne10 项目和 Ne10 库。

中间件

中间件是操作系统和应用层间运行的一套库。在嵌入式系统中，中间件是处于操作系统顶层或有时组成操作系统一部分的系统软件。例如 TCP/IP 通信栈这样的常见中间件就是典型的现代操作系统组成部分。

中间件通过集中化可能形成应用层冗余的软件栈，起到降低复杂性的作用。有许多类型的中间件存在，包括消息通信中间件 (MOM)、远程流程调用 (RPC)、设备驱动程序层上和应用层下的联网协议。

示例

- TCP/IP 栈
- USB 主机栈
- 控制器局域网络 (CAN) 栈
- 多媒体中间件栈

优势

- 简化高可靠强功能应用的开发
- 只完成应用的独到要求
- 支持可扩展性和抽象
- 可靠性

用例

IwIP 是 BSD 协议下提供的开源 TCP/IP 协议套件虽然它可以配套操作系统使用， IwIP 是一个不依存操作系统的独立栈。该用例向您演示了如何把轻量级IP (IwIP) 网络栈等中间件集成到独立应用中。

赛灵思软件开发套件 (SDK) 提供 IwIP 软件定制。IwIP 栈支持 IP、ICMP、IGMP、UDP、TCP 和其他联网协议。通过使用 IwIP 栈，您可以把重点放在开发您的应用核心，把TCP/IP实现留给IwIP栈。

关于如何将开源网络栈集成到应用开发中的更详细说明，请参阅《轻量级 IP (IwIP) 应用示例 v4.0》(XAPP1026) [参照 35]。

启动加载器

启动加载器是指初始化系统，使之准备执行下一级软件，例如操作系统的软件。一般每个操作系统都有自己专用的一套启动加载器。启动加载器一般内置多种启动 OS 内核的方式，同时还内置用于调试和/或修改内核环境的命令。

启动流程的详细状况取决于完整系统设计和 OS 的选择，例如嵌入式 Linux、 RTOS 或裸机 OS。启动的级数也会根据系统设计而变化。本章节用于描述从加电到应用执行的 Zynq-7000 AP SoC 启动流。

启动过程

根据系统要求， Zynq-7000 AP SoC 启动过程可以采取多级启动方式。本章节介绍启动的三个粗略划分的阶段和它们在启动过程中的作用。第 107 页的图 4-6 显示了在 Zynq-7000 AP SoC 上启动嵌入式 Linux 和应用代码的步骤。

在为 SoC 加电时，启动过程从 BootROM 开始。启动过程先从片上储存器 (OCM) 加载然后启动执行第一阶段启动加载器 (FSBL)。 FSBL 负责配置具体的初始化。然后根据软件架构，第二阶段启动加载器，如使用嵌入式 Linux 时的 U-Boot 进行初始化并执行。 FSBL 和/或 SSBL 启动 RTOS 或嵌入式 Linux 以及应用代码。

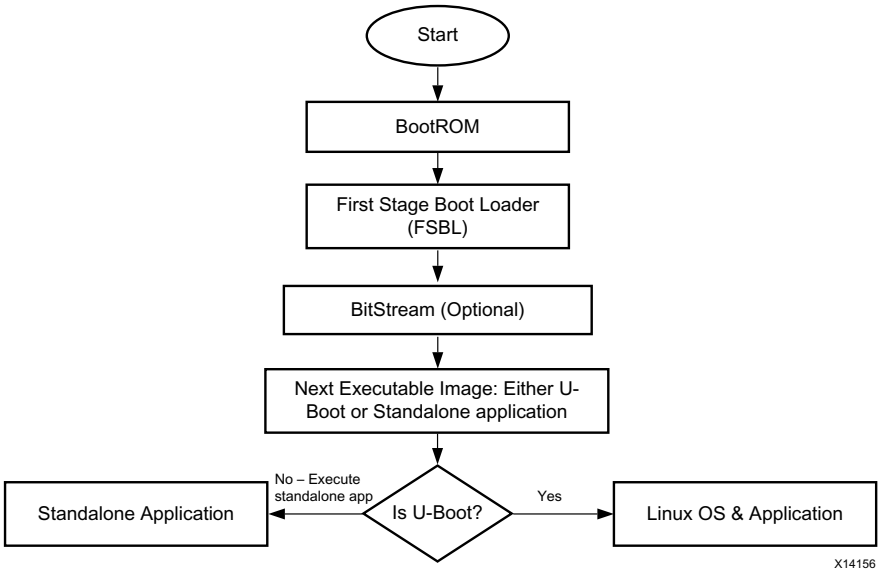

图 4-6 : 启动流

阶段 0

在加电或重置时， ARM 核心从 BootROM 运行初始化代码。 BootROM 代码不能修改；它是一个随同每个 Zynq-7000 AP SoC 提供的工厂预先编程代码。 BootROM 通过读取启动模式引脚判断下一级加载器所在的设备。根据启动模式设置，从 NAND、并行 NOR、串行 NOR (Quad-SPI) 或安全数字 (SD) 闪存储存器拷贝 FSBL 到 OCM。

图 4-7 显示了标准（非安全）启动流。安全启动流的介绍请参阅第 32 页的“嵌入式器件安全性”。

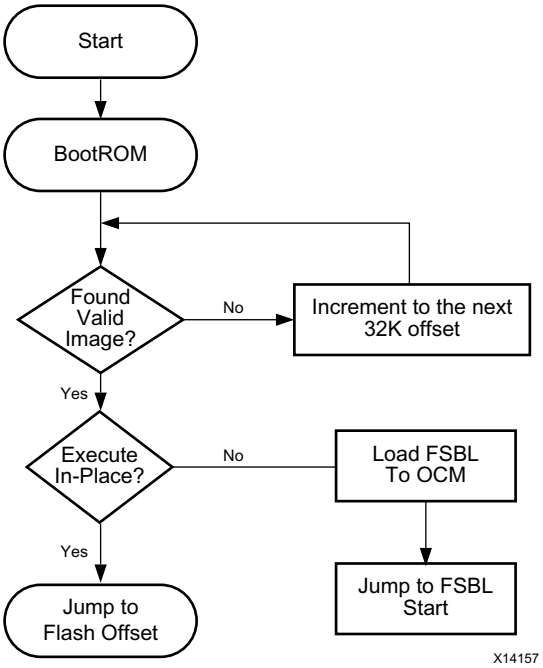

图 4-7 : 非安全启动流

阶段1

FSBL 从 BootROM 获取执行控制权并根据启动模式设置，或从 OCM 运行，或从芯片内执行闪存运行。 FSBL 初始化处理系统并寻找启动设备上的位文件。一旦找到， FSBL 将位文件写入到可编程逻辑中。无论是否找到位文件， FSBL 都会加载应用库和数据文件到储存器中，直到从启动设备读取完整镜像。然后 FSBL 开始执行已经加载的第一个应用二进制文件

一般情况下 FSBL 会初始化外部 RAM 并加载第二阶段启动加载器 (SSBL) 或独立应用。根据软件架构，如果是像嵌入式 Linux 这样的全功能 OS，将会初始化和执行 SSBL（例如嵌入式 Linux 中的 U-Boot）。FSBL 和/或 SSBL 启动 RTOS 或嵌入式 Linux 及应用。随后如果是像嵌入式 Linux 这样的全功能 OS， FSBL 通过跳到 SSBL 的开始地址移交控制权，或是移交控制权给 RTOS 或应用。

阶段 1 可以包含用户应用，前提是用户应用小到 OCM 储存器足以容纳。

Zynq-7000 AP SoC 同时支持安全启动和非安全启动方式。两种方式的 boot-ROM 代码流介绍如下。

对安全启动， SoC 的主要安全目标是奠定信任基础。它从加电到赋予控制这一阶段，通过使用完整性、保密性和认证来达成这一目标。该 SoC 还提供了一种保持这种信任的方法。具体实现方法是使用可编程逻辑内建的高级加密标准 (AES-256) 和散列消息认证代码 (HMAC) 引擎以及处理系统的 RSA 认证功能。

除非从安全模式切换到非安全模式， SoC 都能安全地启动。第一个决策点是 FSBL 的初始加载。如果 FSBL 是加密的，BootROM 代码就会安全启动且一边禁用 JTAG，一边安全地把控制权移交给 FSBL。如果 FSBL 是非加密的，AES 256 和 HMAC 引擎就会禁用， JTAG 端口可以访问。为启用 RSA 认证，必须对“RSA 启用 eFUSE”编程。

使用公共密钥 RSA 算法，赛灵思工具允许设定是否认证软件和比特流划分；使用 AES 和 HMAC 引擎，赛灵思工具允许设定是否加密和认证后续镜像。还可以指定未加密的划分。这样就可以以划分为基础同时使用公共密钥和私人密钥算法。

作为替代方法，如果配置时间有限，可以在安全和启动时间间做权衡取舍，因为相对于加密的划分来说，未加密的划分的配置速度更快。例如对相对较大的开源 U-Boot 或 Linux 镜像，镜像未加密时加载速度较快。如果选择下列安全特性任一，初始 FSBL 至少必须使用 AES-256 加密并使用 HMAC 算法认证。

第 109 页的图 4-8 显示了用于安全启动模式的 FSBL 流。第 110 页的图 4-9 显示了用于非安全启动模式的 FSBL 流。

关于 Zynq-7000 AP SoC 安全启动的详细介绍，请参阅下列文件：

- 《Zynq-7000 All Programmable SoC 中的安全启动》(WP426) [参照 33]
- 《Zynq-7000 All Programmable SoC 的安全启动》(XAPP1175) [参照 40]

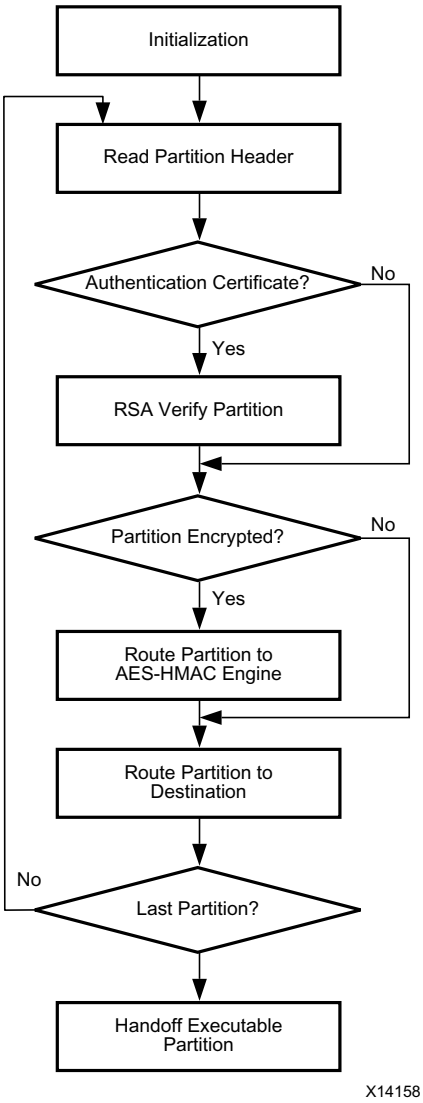

图 4-8 : 用于安全启动的 FSBL 流

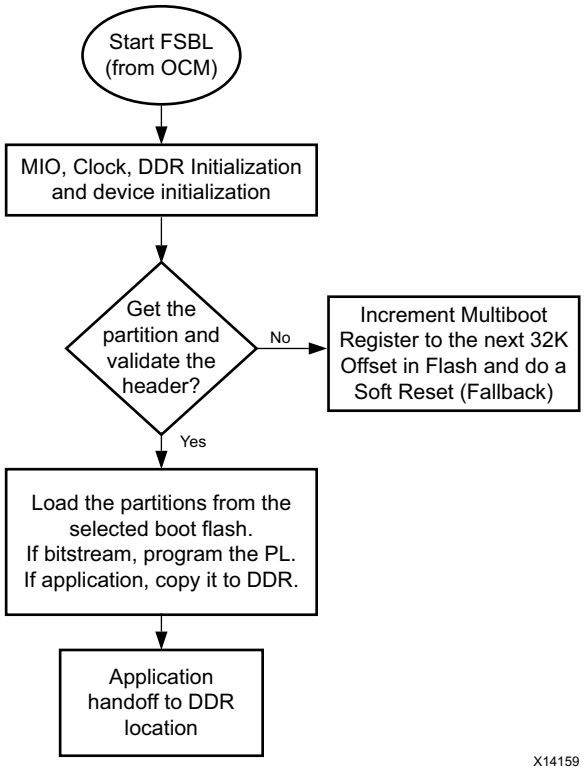

图 4-9 : 用于非安全启动的 FSBL 流

阶段2

这一阶段既可以是第二阶段启动加载器 (SSB)，例如 U-Boot，也可以是 RTOS 或应用。对嵌入式 Linux 这样的全功能 OS 而言，由SSBL U-Boot 在 CPU0 中运行以初始化和设置 OS 将要启动的环境。这一初始化包括配置 MMU 以使用储存器 （即虚拟位置和物理位置相同）。该启动加载器随后把内核镜像 （按启动模式设置）和其他信息，如启动变量传送到 RAM 中。然后该加载器将控制权移交给 OS，例如嵌入式 Linux。

就 Linux 而言， OS 负责检测并启用第二个处理器核心，配置和激活 MMU 和数据缓存，以及执行把完整系统提供给应用的其他操作。

对每个阶段的实际完成的内容没有限定，尽管在某个阶段中完成某些工作比在其他阶段中更加容易。例如，可以从 Linux 中从 NTFS USB 驱动器中读取一个文件，但要从启动加载器中读取就要求一定的开发工作。U-Boot 就能适应这种要求。

《Zynq-7000 All Programmable SoC 软件开发指南》(UG821) [参照 7] 介绍了在 Zynq-7000 AP SoC 上运行 Linux 和应用的架构详情、编程模型、 OS 考量和完整开发步骤。

软件开发工具

赛灵思提供能够用于构建供 Zynq-7000 AP SoC 的各种软件组件的各类软件开发工具，包括：

- 板支持包 (BSP)：这是一套用于访问底层硬件的 API。这些 API 按它们访问的外设和它们执行的功能分组。该 BSP 还包括启动、 CPU 初始化和 EABI 代码。
- 独立 （裸机）应用：这些是不支持多任务等复杂内核特性的简单应用。这些应用使用 BSP API 访问底层硬件。

- FSBL (第一阶段启动加载器)：这是一个小应用，负责执行可编程系统硬件初始化、使用比特流加载架构、有选择地加载附加数据和加载第二阶段启动加载器。 FSBL 是独立 （裸机）应用的示例。
- U-Boot (第二阶段启动加载器)：该启动加载器执行让内核开始执行所需的硬件初始化。在完成此初始化后， Linux 内核就会加载并开始执行。
- Linux 内核：Linux 是一种开源操作系统。该 Linux 内核的硬件相关部分 （设备驱动程序等）是为 Zynq-7000 AP SoC 提供的。该内核的硬件无关部分 （文件系统、联网等）与其他 Linux 机器相似。该内核按需配置并指向 ARM 处理器。
- 用户应用：运行在该 Linux 环境中的典型用户应用且具有硬件无关性。硬件访问通过驱动程序 API 完成。
- 设备树大对象：设备树是一个由用于描述硬件 （可编程系统外设、 ARM 处理器和处理逻辑外设）支持的组件和特性的节点和属性组成的结构。节点提供的信息如硬件组件映射地址、中断值、默省设置等。设备树在启动时编译并提供给 Linux 内核。启动时内核使用该信息加载对应的设备驱动程序。

另外有下列软件开发工具用于选择性地调试和配置硬件平台上的软件。

用于 Zynq-7000 AP SoC 的软件开发工具包括：

- 基于 GUI 的工具链：这些工具为您提供图形用户界面。工具和构建特性通过一套菜单选项选择。在完成选择后，从后台启动编译器/链接器并以可视方式显示编译进度。 GNU 工具集成在基于 GUI 的工具链应用中。赛灵思软件开发套件 (SDK) 即为基于 GUI 的工具链的示例。

  采用 IDE （集成开发环境），编程人员能构建各种类型的软件组件，例如：

  - BSP
  - 独立平台上的用户应用
  - Linux 平台上的用户应用
- 基于命令行的工具链：这些工具是为 ARM 处理器配置的标准 GNU 工具 （例如 gcc)。 CodeSourcery （Mentor Graphics 提供的支持 ARM 架构的 gcc 工具链）就用在针对 Zynq-7000 AP SoC 的开发工具中。编译器和链接器选项使用命令行或由 makefile 提供。 arm-linux-xilinx-gnueabi-gcc 即为基于命令行的工具链的示例。

  使用基于 GNU 的工具链，编程人员可构建各种类型的软件组件，例如：

  - Linux 内核和设备驱动程序
  - 设备树大对象
  - 用户应用
  - 可重用库
- 混合工具链：这些工具是 GUI 和命令行工具的结合。所有项目组件的配置都使用基于 GUI 的工具完成。但每个组件的构建使用命令行工具。 PetaLinux 即为混合工具链的示例。

  为 ARM 处理器配置的 GNU 工具集成在混合工具链中。采用混合工具链，编程人员可构建各种类型的软件组件，例如：

  - Linux 内核和设备驱动程序
  - 设备树大对象
  - 用户应用
  - 可重用库
  - 完整封装镜像

基于 GUI 的工具链

赛灵思 SDK 是一个基于 Eclipse 的集成开发环境 (IDE)。赛灵思 SDK 为创建、编译和调试软件应用提供环境。

该 SDK 在后台运行基于 GNU 的编译器工具链。支持的实用工具包括 GCC 编译器、链接器、 GDB 调试器和其他实用工具，如 JTAG 调试器、闪存编程器和库。此外该工具集成用于所有赛灵思 IP 核、示例应用、软件服务包 （例如 IwIP）

和裸机 BSP 的驱动程序。编程人员可以集成任意此类组件来构建应用。该 SDK 支持使用汇编、 C 和 C++ 语言开发的应用。

赛灵思 SDK 的特性包括：

- 支持 Zynq-7000 AP SoC 和 MicroBlaze™ 处理器。
- 基于 Eclipse C/C++ 开发工具 (CDT)。
- 直接接口 Vivado 工具的完整 IDE。
- 支持完整的软件设计和调试流程，包括多核、硬件和软件调试功能。
- 还集成有编辑器、编译器、构建工具、闪存储存器管理、 JTAG 和 GDB 调试器。
- 得到 Mentor Sourcery CodeBench Lite 赛灵思版的支持。
- 定制库和设备驱动程序。
- 该 SDK 包含针对一切受支持的赛灵思硬件 IP 内核、遵从 POSIX 的内核库及联网和文件处理库的用户可定制驱动程序。这些库和驱动程序可根据特性需求、储存器要求和硬件功能加以定制。
- makefile 管理、硬件平台管理、远程调试、软硬件调试间的交叉探查支持、闪存编程、启动镜像 (boot.bin) 创建、连接器脚本生成、库管理和 FPGA 编程。
- 配置支持。

赛灵思 SDK 调试器的特性包括：

- 基于 Eclipse 目标通信架构 (TCF)。
- 同构和异构多处理器支持。
- 目标平台上的 Linux 应用调试。
- 层级化配置。
- 裸机和 Linux 应用开发。
- 支持对称多处理和非对称多处理设计。
- 逐核心关联硬件和软件中断点。
- NEON 库支持。

应用或基于独立模型开发，或基于 Linux 平台模型开发。

独立平台

在独立平台中，应用以整体的可执行代码方式创建。因为不支持多任务，设备驱动程序随应用加载，应用直接调用驱动程序 API。

独立软件平台是一种单线程环境，在应用直接访问处理器功能时使用。独立软件平台支持程序配置，提供诸如处理器中断处理、例外处理和缓存处理等功能。

独立平台由不需要操作系统的小型专用系统使用。其优势包括：

- 应用开发一般简单快捷。
- 因为不使用情景切换和多任务，总体复杂性得到最小化。
- 开发人员能对从上至下的系统设计有更详尽、更全面的理解。
- 适用于新硬件的初始开发阶段。
- 可使用这种平台构建基于控制台 （文本）的应用。

不足包括：

- 不能构建需要多任务的复杂应用系统。

- 不适用于基于 GUI 的应用。

开发独立应用的步骤包括：

- 创建新工作空间，导入硬件平台到工作空间 （硬件平台从 Vivado 设计工具导出）
- 创建新版支持包 (BSP) 和应用
- 另外，也可导入现有的 BSP 和软件应用到工作空间中。 SDK 中的板支持包 (BSP) 内含在使用提供的 API 时可供软件应用使用的库和驱动程序。
- 修改 BSP 设置。
- 构建 BSP 和应用。
- 调试软件应用。
- 配置软件应用。

图 4-10 给出了使用 SDK 工具的开发周期。

关于使用 SDK 的详细说明请参阅下列文档：

- 《K7 嵌入式 TRD 2013.2》[参照 49]
- 《OS 和库文件集》 (UG643) [参照 5]

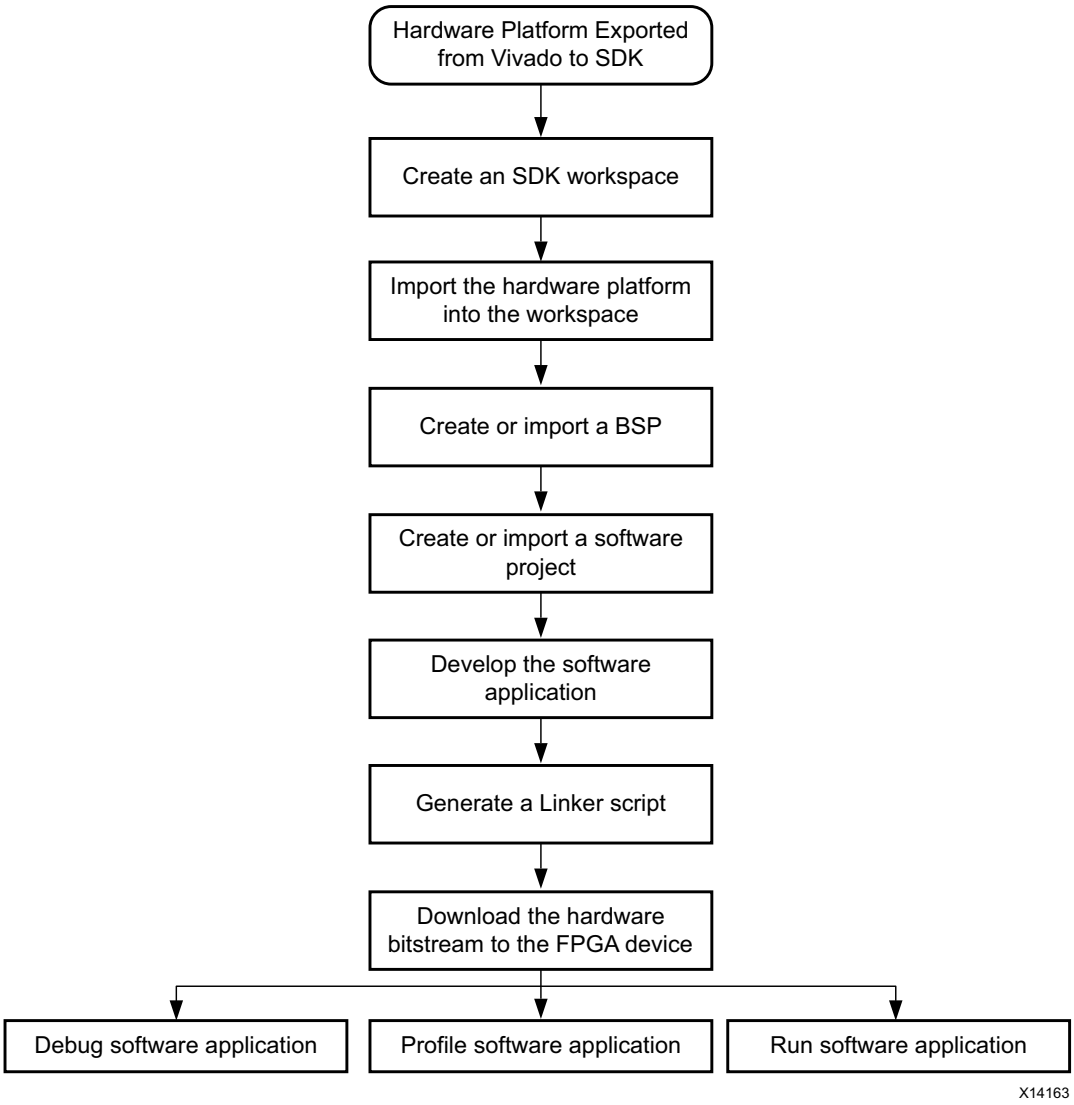

图 4-10 : SDK 软件开发流程

Linux 平台

Linux 是一种多进程操作系统，为 Linux 编写的应用与任何运行 Linux 的系统代码兼容。应用使用 Linux 库 API 并因此以抽象方式访问硬件资源。

针对 Linux 平台的应用开发与其他独立于硬件平台的 Linux 应用开发流程类似。在该 SDK 工作空间中创建应用的流程与独立平台类似，除了选择了 Linux 平台 BSP 的情况。

基于 GNU 的编译器工具链

Zynq-7000 AP SoC 支持以两种方式发起 GNU 工具链：

- GNU 开源工具链
- PetaLinux 框架

这些变体的介绍见下列章节：

GNU 开源工具链

开源工具链基于 Mentor Graphics 的 CodeSourcery 工具链。Sourcery CodeBench 是一个针对 ARM、Power、ColdFire 和其他架构上的嵌入式 C/C++ 开发的完整开发环境。 Sourcery CodeBench Lite 版包括：

- GNU C 和 C++ 编译器
- GNU 汇编器和链接器
- C 和 C++ 运行库
- GNU 调试器

对 ARM 处理器而言，该工具链支持用于包括 ARM 架构第 7 版在内的主动架构的 ARM、 Thumb 和 Thumb-2 指令。

GNU 开源工具链的更详细介绍见赛灵思 Zynq-7000 AP SoC Linux 维基页面 [参照 63]。

PetaLinux 框架

PetaLinux 工具是一种配合用于 Zynq-7000 AP SoC 的赛灵思硬件设计流的开发环境。定制用于提升设计生产力，该解决方案内置构建、开发、测试和部署嵌入式 Linux 系统所需的一切。

PetaLinux 由三个关键元素组成：

- 预配置二进制启动镜像。
- 用于赛灵思器件的完全可配置 Linux 操作系统。
- 包括跨越配置、构建和部署阶段实现复杂任务自动化的工具和实用工具在内的 PetaLinux工具。

因为 PetaLinux 功能齐备并经集成、测试和文档化，它是赛灵思向赛灵思客户建议和支持的 Linux 部署机制。PetaLinux BSP 为下列赛灵思器件提供完整的经集成和测试的 Linux 操作系统：

- BSP
- 启动加载器
- Linux 内核
- Linux 应用和库
- C 和 C++ 应用开发
- 调试
- 线程和 FPU 支持
- 用于简化网络管理和固件配置的集成 web 服务器

虽然 PetaLinux 以稳定和经精心测试的 Linux 版本为基础， PetaLinux 工具的内容超过赛灵思 Git 服务器提供的内容。PetaLinux 工具包括安装程序、开发工具、 BSP、平台管理实用工具、应用以及赛灵思开源 Linux (OSL) 方案中未提供的库框架和文档。

所有 PetaLinux 用户都有权取得社区 （基于论坛）支持。商业被许可人可直接从赛灵思获得支持。

PetaLinux 的工具链的详细介绍请参阅赛灵思 PetaLinux 工具网站 [参照 48]。

JTAG 调试器

基于 GUI 的 SDK 包括调试功能，让编程人员能以下列方式调试代码：

- 单步
- 中断点
- 储存器观察点
- 反汇编
- 调用栈
- 处理器寄存器转储

该 SDK 提供多种使用调试功能的途径。系统调试器基于 Vivado 硬件服务器 （基于 TCF）。系统调试器在 JTAG 或 TCP/IP 链路上开展调试操作。JTAG 链接用于局部调试，TCP/IP 链接用于与 Linux 中的 TCF 代理通信。该调试器还可以做为 XSDB 的命令行调试器的方式提供。使用 XSDB 时可以不调用 SDK。

基于 GUI 的 SDK 调试器的详细介绍请参阅 《K7 嵌入式 TRD 2013.2》 [参照 49]。

命令行 XMD 调试器的详细介绍请参阅 《嵌入式系统工具参考手册》(UG1043) [参照 21]。

JTAG 配置器

该基于 GUI 的 SDK 还包含一个软件配置器。该配置可让编程人员监测功能调用、处理器在每个功能上花费的时间，处理器占用的图示等。和调试功能一样，该配置器也以基于 TCF 的 XMD 工具为基础，通过 JTAG 链接完成配置操作。

详细介绍请参阅 《基于 AXI 界面的 KC705 嵌入式套件 MicroBlaze 处理器子系统教程》(UG915) [参照 12]。

第 5 章

硬件设计流程

本章介绍了使用 Zynq®-7000 AP SoC 开展设计时需要考虑的下列软件问题：

- 第 117 页的“使用 Vivado IDE 构建 IP 子系统”：通过 Vivado® Design Suite，您能够运用本章节介绍的各类工具构建 IP 子系统。
- 第 119 页的“基于规则的连接”：您可以通过运行设计验证流程发现错误。设计验证流程会对模块设计运行设计规则检查并报告警示和错误。本章节介绍设计验证流程。
- 第 119 页的“创建层次化 IP 子系统”：您可以使用 Vivado IP 集成器工具创建层次化 IP 子系统。
- 第 119 页的“板器件接口”：IP 集成器上的“Board Part Interfaces”视图显示的是所有出现在特定板件上的接口。
- 第 119 页的“生成模块设计”：在模块设计或 IP 子系统创建完成后，就能够生成源文件、 IP 约束和结构网表。在这一步完成后，设计就可以集成到更高级的 HDL 设计中，或用于综合和实现。
- 第 120 页的“创建和封装供重用的 IP”：Vivado IP 打包器能够让 Vivado IDE 用户和第三方 IP 开发人员轻松地准备用于 Vivado IP 目录的设计。该定制 IP 随后使用 Vivado 设计套件即可实例化到设计中。
- 第 121 页的“创建定制接口”：Vivado Design Suite 要求所有存储器映射接口均采用 AXI 接口。该套件的向导能够协助将定制 IP 接口转换为符合 AXI 接口标准的 IP 接口。
- 第 121 页的“管理定制 IP”：Vivado IP 目录包含内建用于添加来自其他源的 IP 核的库管理功能。
- 第 121 页的“高层次综合 (HLS)”：该高层次综合 (HLS) 工具将 C、 C++、 OpenCL 内核或 SystemC 设计规格转换为寄存器传输级 (RTL) 设计，用于随后综合到赛灵思 All Programmable 器件中。

简介

随着可编程器件日益庞大和复杂，同时设计日程不断缩短，使用第三方 IP 和设计重用成为了必备要求。赛灵思认识到了设计人员面临的挑战，已经在 Vivado® 设计套件中建立起一项强大的功能，用于帮助解决这些挑战。这项功能被称为 Vivado IP 集成器。

使用该 IP 集成器可创建模块化设计。这些模块设计在本质上属于包含任意数量用户配置 IP 和互联的 IP 子系统。 IP 集成器负责在 Vivado 设计工具中通过 Zynq®-7000 AP SoC、 MicroBlaze™ 处理器设计，以及非处理器主导设计来开展嵌入式处理器设计。它用于例化来自 Vivado HLS 的高层次综合 (HLS) 模块，来自系统生成器的 DSP 模块和使用创建—封装 IP 流制作的定制 IP。设计可在 IP 集成器的 GUI 界面上交互式地创建，或通过 Tcl 编程界面由编程创建。

详情，请参阅： 《Vivado Design Suite 用户指南：采用 IP 集成器设计 IP 子系统》(UG994) [参照 18]。

使用 Vivado IDE 构建 IP 子系统

Vivado IDE 可用于通过 IP 集成器 GUI、脚本化 Tcl 流或 IP 集成器提供的设计助理功能来构建 IP 子系统。

使用该 GUI 创建 IP 子系统

IP 集成器为创建 IP 子系统提供了一个强大的 GUI 环境。IP 核可采用各种自动化功能实现例化和互联。定制 IP 也可以针对 IP 集成器封装。设计应在 GUI 环境中重新装配并完成流程。在设计稳定后，即可编写用于创建和实现设计的脚本。关于 GUI 的功能的更详细介绍，请参阅 《Vivado Design Suite 用户指南：采用 IP 集成器设计 IP 子系统》(UG994) [参照 18]。

使用脚本流创建 IP 子系统

IP 集成器 GUI 界面上的每个行为都会产生等效的 Tcl 命令。这些 Tcl 命令还记录在 Vivado 设计套件的日志文件里。可以依据该日志文件编写脚本，然后后续将该脚本重复用于创建设计和实现设计。

设计助理

为便于创建使用处理器的设计，可使用IP集成器的模块自动化和连接自动化功能。模块自动化功能应被用于配置处理器和相关IP内核，而连接自动化功能应被用于完成设计中不同引脚或端口的重复性连接。 IP 集成器还具有板件感知能力并支持赛灵思评估板。这意味着将一个评估板用作目标硬件时， IP 集成器知晓该板件上存在的全部接口。设计中的 I/O 端口通过连接自动化功能就能连接到目标板上的现有接口。设计助理功能则有助于时钟和复位连接。包含数个视图，例如“Signals”视图和“Board Part Interfaces”视图，有助于完成模块设计中的连接。设计助理功能帮助简化互联，避免意外发生的设计错误。

模块自动化

对于一些基于 IP 和处理器的复杂设计而言， IP 集成器可为其提供一种名为模块自动化的功能。该功能使用常用组件支持基于处理器或 IP 的子系统的快速装配。在用于嵌入式设计的基本构建模块装配完成后，就可以通过从目录添加 IP 核扩展基本系统。

连接自动化

在模块自动化完成后，基本系统已构建完成，需要连接外部 I/O 引脚。连接自动化能够实现这一目的。连接自动化有助于完成到 I/O 引脚的连接，还能帮助完成到设计本身上的不同源的连接。与板件感知功能相结合，连接自动化有助于连接模块设计的端口到目标板上的外部接口以及为这些端口创建物理约束。因此极力推荐您使用该功能。

基于规则的连接

IP 集成器在设计装配的过程中实时地运行基本的设计规则检查。但是在创建过程中有发生错误的可能性。例如时钟引脚的频率可能设置得不正确。这样的错误通过运行设计验证就能够发现。设计验证在模块设计上运行规则检查并报告任何与设计有关的告警或错误。这些告警和/或错误随后可以从消息视图交叉探测到原理方框图中。建议采用设计验证功能捕获那些可能直到设计流程后期才注意到的设计错误。

在运行设计验证功能的时候，还会在模块设计上运行参数传播功能，这是 IP 集成器最强大的功能之一。该功能使 IP 核根据自身在设计内的连接方式自动更新参数化。 IP 核可以使用特定的传播规则封装， IP 集成器在方框图生成的时候就会运行这些规则。

创建层次化 IP 子系统

IP 集成器可用于创建层次化 IP 子系统。这一功能适用于拥有大量模块的设计，否则此类设计在 GUI 菜单中可能很难管理。通过支持多个层次结构，使模块能够根据设计功能分组。在 IP 集成器内部使用层次结构有助于在 IP 集成器菜单中保持设计的模块化和整洁度。

设计对象的外观效果可以修改。例如可以为时钟和复位标注不同的颜色，各个层次可以启用也可以停用。

板器件接口

如上所述， IP 集成器具有板件感知能力。当模块设计指向特定的赛灵思评估板时， “Board Part Interfaces”视图就会显示该特定板件上出现的所有接口。显示在“Board Part Interfaces”视图中的接口可以连接起来，创建使用或不使用处理器的目标设计。这是一种创建模块设计的有力方法。当把某个接口选用于连接时，会显示能够连接到该特定接口的所有 IP 内核。例如，在 Kintex® KC705 板上如果选用 8 位 LED，就会显示三种能与该接口连接的 IP （GPIO、IO 模块和 MicroBlaze MCS IP）。根据相应的 IP 选择，就能形成到板上的 LED 的连接，或是借助设计助理，形成到模块设计其余部分的连接。

生成模块设计

在模块设计或 IP 子系统创建完成后，就会生成源文件、 IP 约束和结构网表。这一步完成后，设计就可以集成到更高级的 HDL 设计中，或是用于综合和实现。

为模块设计使用非关联综合

层次化设计流程能把设计划分为更小、更容易管理的模块，供单独处理。在 Vivado Design Suite 中，这些流程依靠的是对划分出的模块与设计其余部分开展无关联 (OOC) 综合的能力。IP 集成器的最常见用途是综合 OOC 模块并创建设计检查点 (DCP) 文件。该模块设计如果用作更大型 Vivado 设计的组成部分，那无需在每次设计的其他部分 （在 IP 集成器之外）发生修改时重新综合。这样可以节省大量运行时间，应该在运行时是关注点的时候加以考虑，尤其是在设计探索的早期阶段。

创建远程模块设计

IP 集成器能创建可供多个 Vivado 设计套件项目重复使用的独立模块设计。在设计创建完成并纳入版次控制之后，多个设计团队能够重复使用相同的模块设计创建多个项目。这是一项重要的 IP 集成器重用功能，应在团队开发环境中加以考虑。

创建和封装供重用的 IP

Vivado IP 打包器能帮助 Vivado IDE 用户和第三方 IP 开发人员方便地准备在 Vivado IP 目录中将要使用的 IP 设计。随后可以使用 Vivado Design Suite 将定制 IP 例化到设计中。使用下列封装流程可以确保一致的用户体验，无论是在 Vivado 设计套件中使用赛灵思 IP、第三方 IP 还是客户开发 IP。

封装定制 IP 和 IP 子系统

Vivado 创建与封装 IP 功能支持创建用于 Vivado IP 目录的定制 IP。采用行业标准 IP-XACT 格式打包 IP 核。将打包 IP 的位置添加到 Vivado Design Suite “Project Settings” 的“Repository Manager”部分。在一个或多个 IP 核的库完成添加后，该 IP 就会显示在 IP 目录中。用户能从这个目录选择和定制 IP 核。

使用 Vicado IP 打包器的流程如下：

1. 使用 Vivado IP 打包器创建和打包 IP HDL 和相关数据文件。 IP 打包器 目前不支持将 SystemVerilog 源作为顶层。需要围绕 SystemVerilog 源的 Verilog 或 VHDL 封装。
2. 为团队成员或客户提供封装后 IP。
3. 最终用户向 Vivado Design Suite“Project Settings” 的“Repository”部分添加 IP 位置。
4. 该 IP 核显示在 IP 目录中，最终用户可使用与赛灵思交付 IP 类似的方式选择和定制该 IP 核。

创建和封装 IP 功能提供两种创建定制 IP 的方法。第一种途径是使用创建和封装 IP 功能创建拥有一个或更多 AXI 接口的新 IP 核。在这些接口创建完成后，插入定制 IP 并连接到这些接口。第二个途径是封装该 IP 核并将它包含到 IP 目录中，供后续在 IP 集成器中使用。在用于该定制 IP 的所有 HDL 文件，包括 AXI 接口都可用的情况下可以采用第二种途径。

创建和封装 IP 功能可让 IP 最终用户拥有一致的体验，无论是使用赛灵思 IP、第三方 IP 还是定制 IP。关于创建和封装 IP 的更详细介绍，请参阅 《Vivado 设计套件用户指南：创建和封装定制 IP》(UG1118) [参照 22]。

更新 IP 目录

Vivado 设计套件 IP 目录是一个统一的库，支持搜索、审核详细信息，查看用于 IP 内核的文档。在 IP 内核封装完成后，用户可以在 Vivado 设计套件中指向包含新封装的 IP 内核的 IP 库并把该 IP 内核添加到 IP 目录中。新封装的 IP 随后就可以在 IP 集成器中使用。

创建定制接口

Vivado Design Suite 要求所有存储器映射接口均采用 AXI 接口。该套件提供的向导可帮助把定制IP接口转换为符合 AXI 接口标准的接口。 Create and Package IP 流程可生成三类 AXI 接口：

- AXI4：用于单个地址相位中最多允许 256 个数据传送周期的内存映射接口
- AXI4-Lite：轻量、单事务存，储器映射接口
- AXI4-Stream：无需地址阶段而且数据突发包的大小无限制的接口

Create and Package IP 流程可创建一个包含 HDL、驱动、测试应用、总线功能模型 (BFM)，实例模板的模板 AXI4 外设。在外设创建完毕后，添加设计文件即可完成定制 IP。详情，请参阅 《Vivado Design Suite 用户指南：采用 IP 进行设计》(UG896) [参照 22] 和 《Vivado Design Suite 用户指南：采用 IP 集成器设计 IP 子系统》(UG994) [参照 18]。

管理定制 IP

Vivado IP 目录包含用于添加来自其他源的 IP 核的内建库管理功能。为让该定制 IP 可见，应将该定制 IP 放置在从主机能够访问的位置。随后启动 Vivado Design Suite，从 IP 目录运行“IP Settings”功能，登记新的用户库位置并把这一新 IP 核纳入 IP 目录中。

这里有两种类型的库：

- 标准赛灵思库：Vivado Design Suite 提供标准赛灵思库。这些库一直处于启用状态，它们在赛灵思 IP 目录中不能更改。
- 配置用户库：从包含一个或多个 IP 核的有效机器可以看到用户库。

库管理器允许添加或移除用户库并且在库之间建立先后次序。 IP 则用对应厂商、库、名称和版本 (VLNV) 的元素排序清单加以区分。

如果多个库被引用而且在多个位置有相同的 IP VLNV， Vivado IDE 就会在有最高优先级的库中显示 IP 核。赛灵思 IP 库一直在启用状态，总是处于最低的优先级。

如果某个特定项目的库设置发生变化，库管理器就会在项目设置中存储变化。因此当项目在任何机器上再次打开时，就可以看到变动 （假设库路径也可用）。

高层次综合 (HLS)

高级综合 (HLS) 功能把 C、C++、OpenCL 内核或 SystemC 设计规格转换为寄存器传输级 (RTL) 设计，以便综合到赛灵思 All Programmable 器件中。

HLS 对设计执行两种类型的综合：

- 算法综合：算法综合把功能语句综合为可能占用多个时钟周期的 RTL 语句。
- 接口综合：接口综合把函数自变量 （或参数）转换为有特定时序协议的 RTL 端口，让设计能与系统中的其他设计通信。

与手工 RTL 设计中做出的许多决策相似，可用的设计与优化的数量巨大，它们彼此影响方式的组合数量极大。 HLS 将这些详细内容摘要，帮助在最短时间内创建最佳设计。

一般使用 HLS 的情况是在 SDK 或第三方工具中配置软件应用时发现软件的瓶颈。一旦发现瓶颈功能，下一步就是使用 HLS 把这些功能移动到硬件中。最终 HLS 设计可作为 IP 内核导出，在 IP 集成器中使用。 HLS 还能为可用于验证 IP 功能的 IP 核生成仿真模型。详情，请参阅 《VivadoDesign Suite 用户指南：高层次综合》(UG902) [参照 9]。

总结

一般而言，在 IP 集成器中采集嵌入式设计应遵循下列步骤：

1. 添加处理器 IP，例如 Zynq-7000 AP SoC 或 MicroBlaze 处理器。
2. 在处理器 IP 例化时，设计助理功能可用。使用设计助理功能配置处理器和外设。
3. 根据需要进一步定制处理器。如果需要更多功能和对时钟、复位、 I/O 端口等施加控制，就需要此步骤。
4. 添加外设，在设计助理功能可用时连接外设。
5. 为 GPIO、以太网等外部接口添加连接 IP。
6. 根据需要为处理逻辑添加定制加速器。
7. 使用“Signals”视图或 Make Connection 向导连接和审核时钟并复位域。
8. 通过验证设计运行设计规则检查。解决任何设计验证过程中提示的差错或告警，运行设计验证直至不再提示出错。
9. 综合设计，实现设计，为设计生成比特流。
10. 导出设计到 SDK 供软件开发。

第 6 章

软件设计流程

本章介绍用于 Zynq®-7000 AP SoC 的软件设计流程并为各类软件开发岗位 （合称“岗位”）提供指引。针对每个岗位的设计流包括对工具流的简介以及各阶段赛灵思及其合作伙伴提供的可用解决方案。另外在需要更多信息时，提供对外部文档和本指南中其他章节的索引。

与 ASIC 和 ASSP 不同，Zynq-7000 AP SoC 拥有可编程逻辑 （PL，即器件的硬件可编程 FPGA 部分）和处理系统 （PS，即两个 ARM Cortex-A9 核心）。 PL 的硬件设计和配置可使用赛灵思 Vivado® 工具完成。请参阅第 5 章"硬件设计流程"。来自 Vivado 工具的硬件设计文件被发送到赛灵思软件开发套件 (SDK) 供软件开发。图 6-1 所示的是用于 Zynq-7000 AP SoC 的硬件和软件开发流。

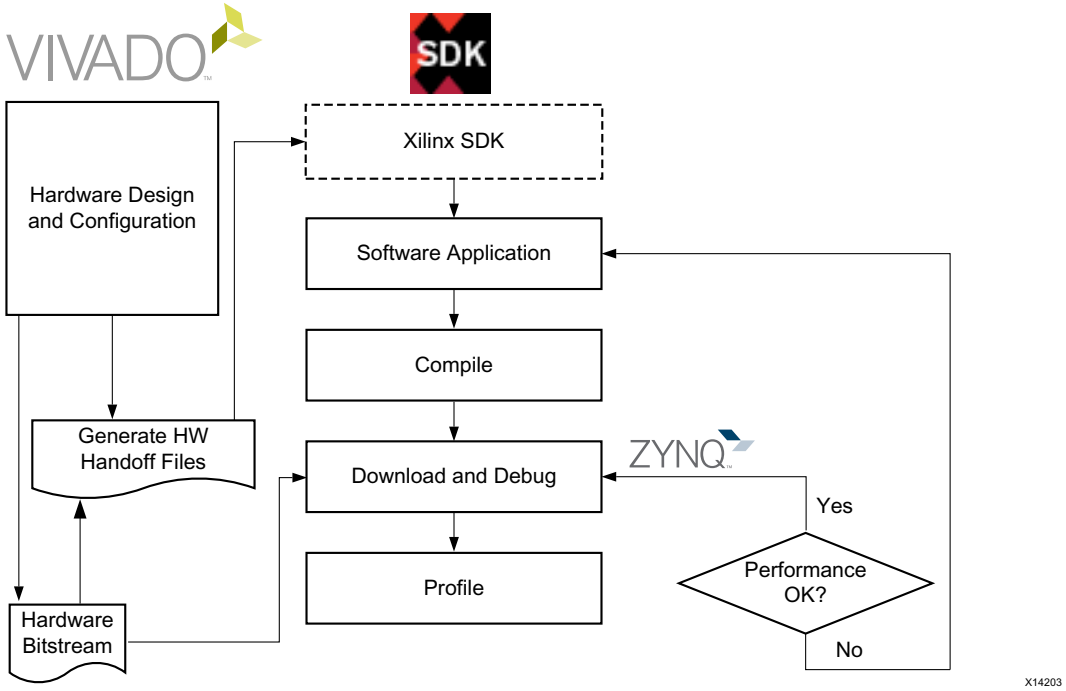

图 6-1 : 硬件和软件开发流

根据第 123 页的图 6-1，从硬件流发送到软件流的文件 （称为 handoff files）内含硬件规格、 PS 外设信息、寄存器储存器映射和用于 PL 的码流。交接文件的目的是将软件开发人员与 Zynq-7000 AP SoC 可配置硬件相隔离。交接文件让硬件对软件表现为 ASSP。软件开发人员可使用交接文件设计估计、驱动和板支持包 (BSP)。

一般而言， Zynq7000-AP SoC 的软件设计流涉及开发图 6-2 所示的一个或多个软件层。

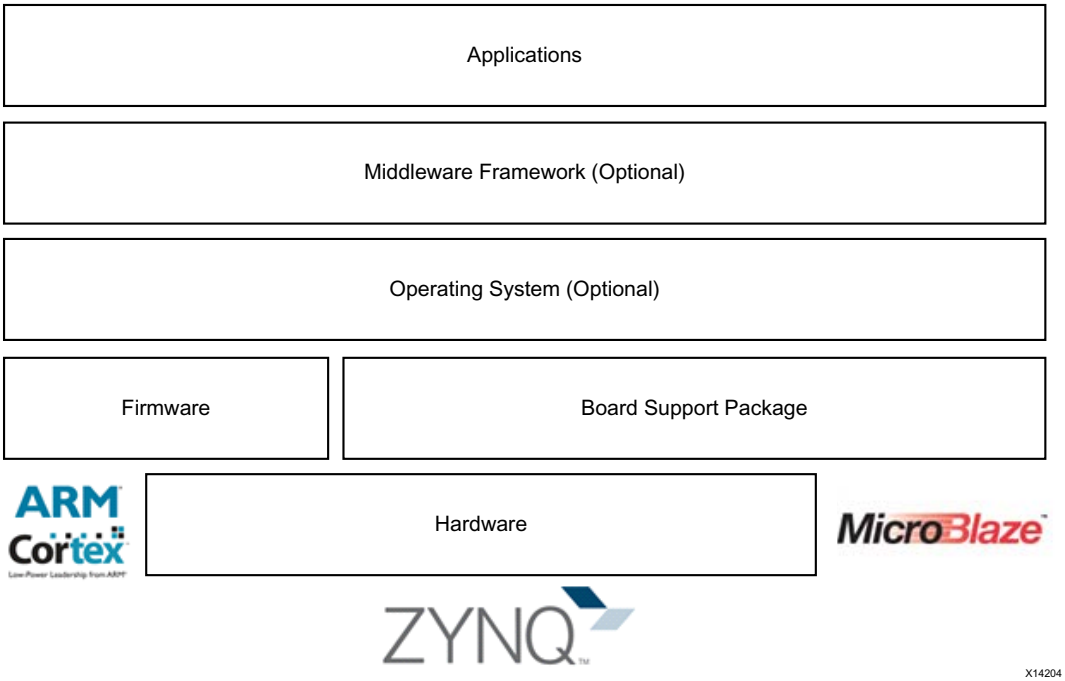

图 6-2 : 软件设计层

其余章节介绍三个岗位的软件开发流：板启动开发人员、驱动开发人员和应用开发人员。每个岗位工作在图 6-2 所示的一个层上。具体岗位可参阅本章中的对应章节，掌握与自己工作领域相关的软件开发流。

板启动开发

板启动开发工作包括开发低级固件、设置启动次序、针对接口和外设的基本测试。本章节介绍可用于基本板启动工作的赛灵思开发解决方案，并划分了下列阶段：

1. PS 初始化
2. PL 配置
3. 储存器和外设测试
4. 软件和硬件调试

PS 初始化

当 Vivado 导出设计时，赛灵思工具自动生成 `ps7_init.tcl` 和 `ps7_init.c` 文件。 `Ps7_init.c` 文件是一个 Vivado 自动生成的初始化文件，供第一阶段启动加载器 (FSBL) 初始化 PS。在启动电路板时，第一步是通过赛灵思调试器连接目标 Zynq-7000 AP SoC，并运行 `ps7_init.tcl` 脚本 （提供与 `ps7_init.c` 相同的结果），以初始化 PS 外设、时钟、 DDR、 PLL 和 MIO。为成功地验证初始化的 Ps，从 XSDB 调试器执行储存器读取和验证命令。可读取和写入系统外设以验证基本功能。

FSBL 能加载 U-Boot 等第二阶段启动加载器，也能加载和启动操作系统。在裸机设计中， FSBL 直接加载应用代码。

请参阅《Zynq-7000 All Programmable SoC 软件开发指南》(UG821) [参照 7] 进一步了解包括正常启动和安全启动在内的 FSBL 介绍。

在第三方调试器上，FSB L 能用于初始化 DDR 储存器和加载应用代码，以便调试器调试应用代码。例如 FSBL 可在 OCM 中加载，执行数秒钟、停止，然后使用调试工具 （通过 JTAG）在 DDR 中加载应用代码，供调试使用。大多数调试器支持自动化该流程的脚本机制。

赛灵思 SDK 库提供了用于板启动的裸机驱动。

U-Boot

U-Boot 是一种可作为主要或次级加载启动器使用的开源、通用启动加载器。 U-boot 可用于加载独立应用、码流或 Linux OS 内核。在 Zynq-7000 AP SoC 上，U-Boot 被用作次级启动加载器，但经配置和重构也可以用作主启动加载器。FSBL 设计用于启动像 U-Boot 这样的次级启动加载器。 U-Boot 可从外部设备加载，并且可用于初始化主启动加载器尚未初始化的 PS 外设。 U-Boot 或任何其他次级启动加载器都可以从最初产品中移除，以缩短启动时间。

图 6-3 中的流程图勾勒出启动流程中涉及的启动次序和文件概况。

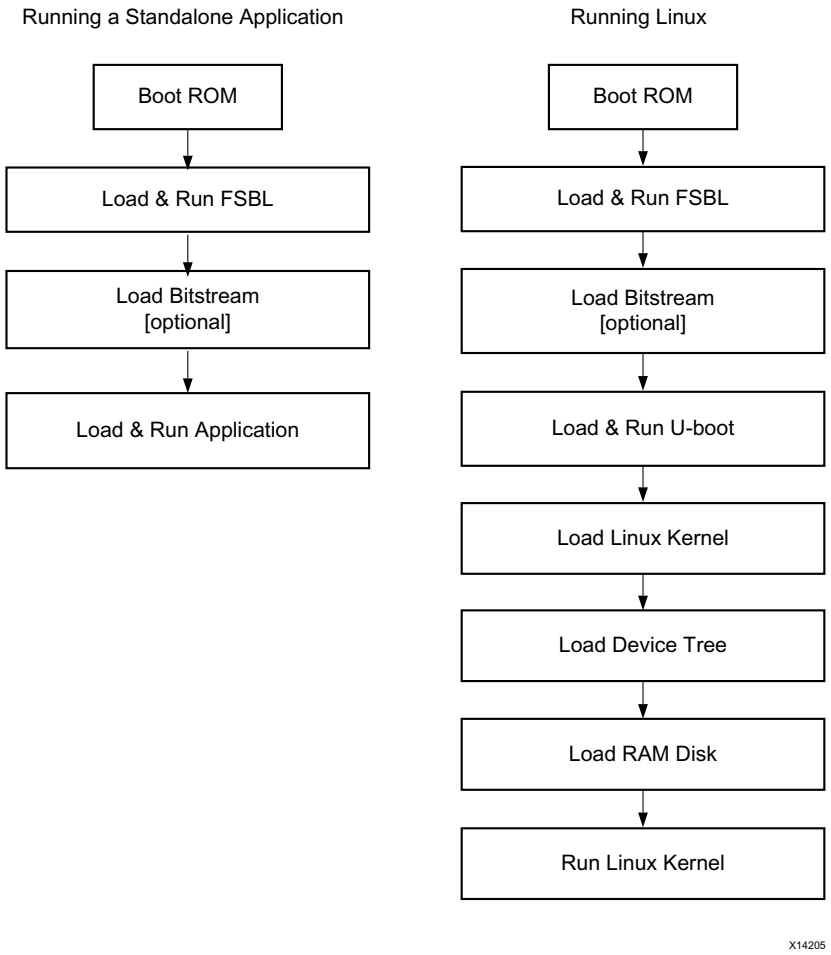

图 6-3 : 启动次序和启动文件

在赛灵思 Git 库中提供了开源的赛灵思 U-Boot 项目。赛灵思 U-Boot 也作为 PetaLinux 解决方案的组成部分提供。请参阅赛灵思 U-Boot 维基页面 [参照 57] 进一步了解配置和构建 U-Boot 的相关信息。

PL 配置

PL 的配置可通过运行在 ARM 核心上的软件，或可通过 JTAG 开展。赛灵思支持使用 FSBL、 U-Boot 或 Linux 配置 PL。

在量产板上 FSBL 能使用 PCAP 接口配置 PL。在开发和调试阶段，PL 可通过 Vivado 工具或赛灵思 SDK 提供的 JTAG 配置。在 PL 完成配置后，处理器就能以与访问 PS 外设相同的方式访问 PL 外设。调试器在这个阶段能读写 PL 外设寄存器。

储存器和外设测试

赛灵思 SDK 拥有储存器测试功能，能验证 DDR 储存器以及储存器控制器和 DDR 接口的信号完整性。裸机储存器测试应用可用于 DDR 储存器加电自检。 DRAM 测试应用模板位于您的安装目录 `SDK\2014.2\data\embeddedsw\lib\sw_apps\zynq_dram_test` 中。

有大量 PS 外设可用的示例驱动。这些驱动可在您的安装路径下的 `SDK\<Version>\data\embeddedsw\XilinxProcessorIPLib\drivers\`

<Peripheral> 文件夹找到。该文件夹还包含用于赛灵思软件 IP 内核的样本测试应用。这些测试可用于在板启动阶段测试外设。此外，这些测试经修改，可开发综合性的加电自检或自检测。

软件和硬件调试

在初始化 DDR 时可能遇到失败的情况。如果发生这种情况，通过调试 FSBL 阶段就能建立到调试器的连接。任何下载到 DDR 的应用都能在下载后进行调试。赛灵思 SDK 支持异构多核调试，允许同时调试 MicroBlaze™ 处理器和 ARM 核心。此外，在不清楚问题是在软件还是在硬件中时，可使用交叉触发。请参阅 《Vivado设计套件教程：嵌入式硬件设计》(UG940) [参照 15] 了解关于设置交叉触发的详情。

驱动开发

驱动开发人员为 SoC 和板载外设开发软件驱动，用于为 O S或裸机应用等更高软件层建立接口。必须为 ARM Cortex-A9 和 MicroBlaz e核心创建外设驱动。本章节的要点在于考虑何时为配置后硬件开发裸机和 Linux 驱动，并重点介绍可用的赛灵思解决方案。

裸机驱动

赛灵思 SDK 支持驱动开发中使用的各种文件类型（同时包括赛灵思 IP 驱动和定制驱动）和软件开发周期中使用的各种文件类型。具体文件类型和目录结构如图 6-4 所示。

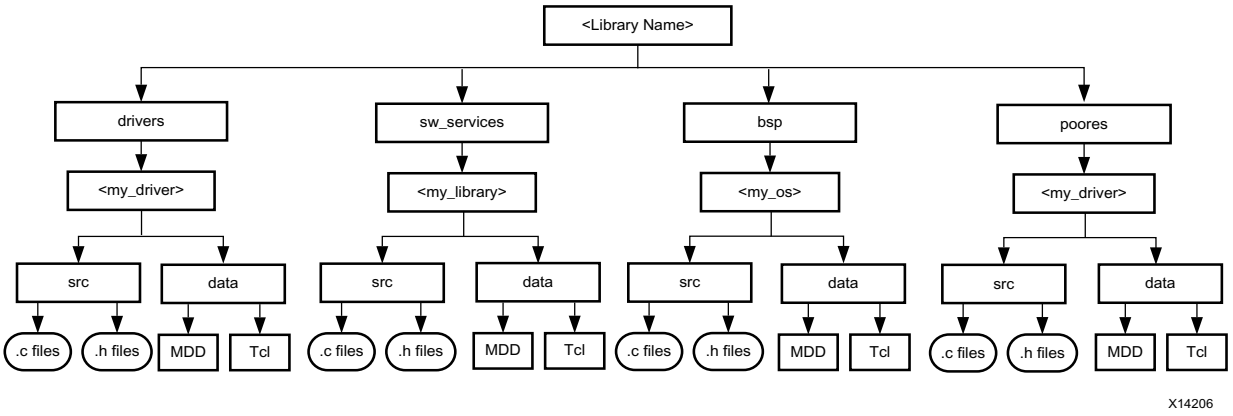

图 6-4 : 驱动、 OS 和库的目录结构

交接文件详情

从交接到调试的软件开发周期如图 6-5 所示。

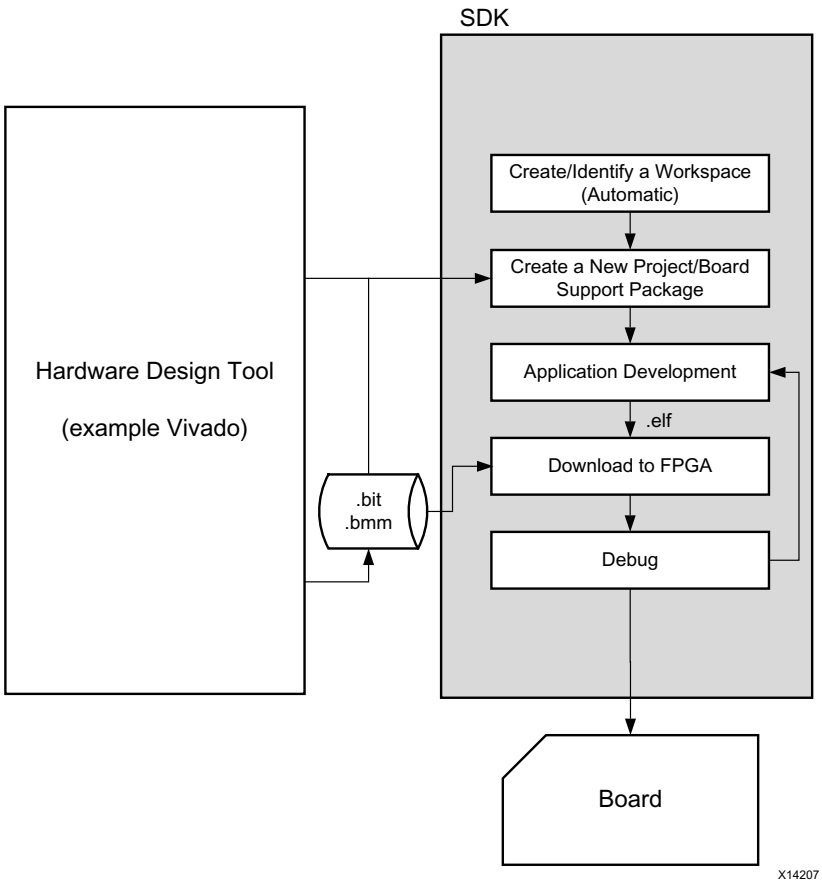

图 6-5 : 使用 SDK 的软件开发周期

软件开发流程始于在硬件设计流中创建交接文件。在使用 Vivado 工具准备硬件设计后，即创建硬件交接文件夹，供 SDK 在驱动和应用开发中使用。从 2014.2 版开始，Vivado 硬件导出功能会创建一个统一的压缩硬件定义文件 (`*.hdf`) 并把它写入到 `SDK_Export` 文件夹。该文件包含基本地址和版本信息等硬件详情。从 SDK 工作空间创建硬件平台项目，然后通过将该项目指向 `*.hdf` 文件，完成工作空间中的该项目的配置。除 `*.hdf` 文件以外， SDK （从 2014.2版开始）支持导入传统 XML 文件。

在创建 BSP 项目时， SDK 生成 `xparameters.h` 文件，然后以 `#define` 常数的形式补充硬件详情 （例如基本地址和中断 ID）。软件驱动 （独立平台）和应用能够在代码中引用这些常数。

驱动文件编制

赛灵思 IP （PS 外设 IP 和软 Ip）的驱动随 SDK 一道提供。这些驱动位于您的 SDK 安装文件夹的 `data/embeddedsw/XilinxProcessorIPLib/drivers` 文件夹中。每个驱动位于单独的文件夹中。例如 IIC PS 驱动 2.0 版本位于您的 SDK 安装文件夹的 `data/embeddedsw/XilinxProcessorIPLib/drivers/iicps_v2_0` 文件夹中。

每个驱动器文件夹包含四个子文件夹：

- `data`：该文件夹包含 MDD 文件和 Tcl 脚本文件。
- `doc`：该文件夹包含实现在驱动中的 API 和数据结构详情的帮助文件。

- `examples`：该文件夹包含 C 语言源文件，用于示范如何使用驱动的示例。这些示例随时可在硬件上构建和启动。遵照用于构建独立应用的流程，这些示例可立即使用。这些示例也可根据需要加以修改。
- `Src`：该文件夹内含驱动源代码 （C 和 H 文件）。

源文件库的组织形式见图 6-6。

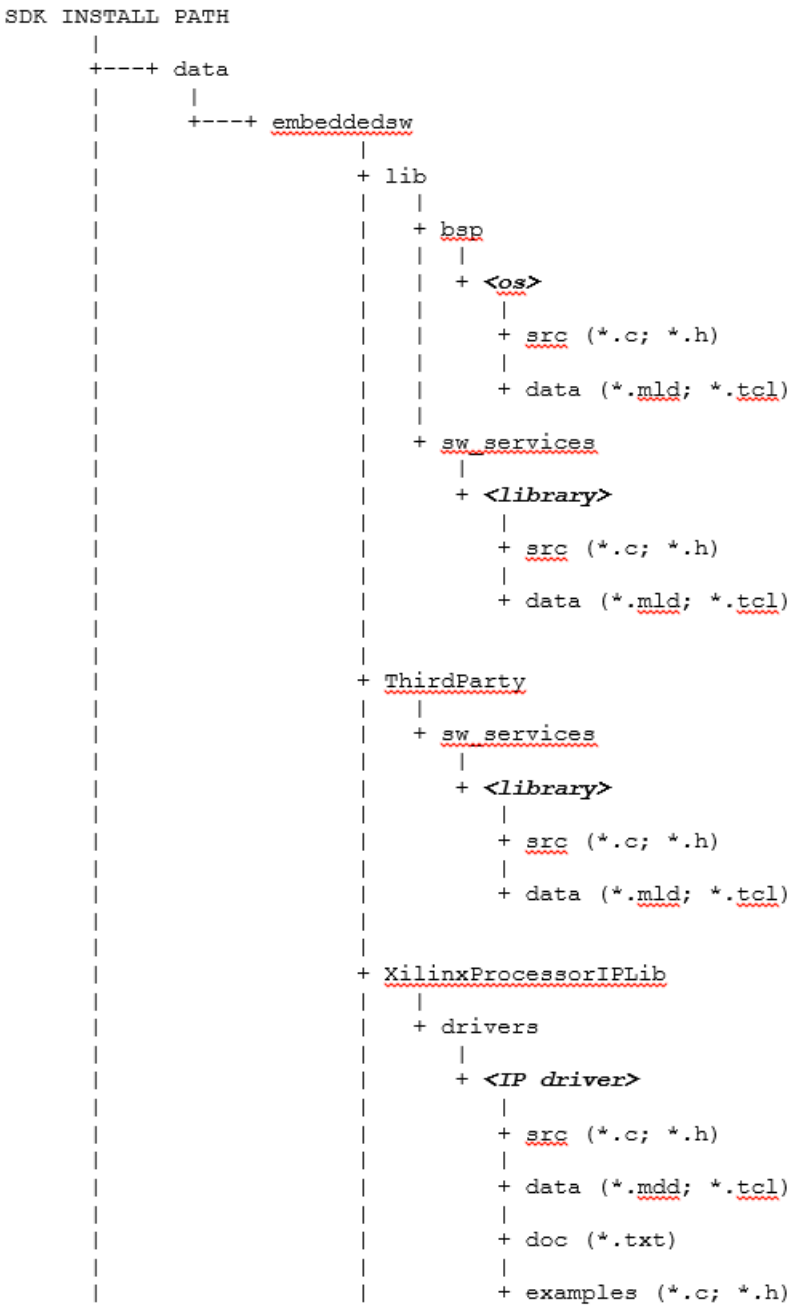

图 6-6 : 源文件库

为加载驱动源代码到项目工作空间，SDK 会读取每个版本文件夹中的 MDD 文件。SDK 会根据 MDD 文件中的参数值选择对应的驱动版本。当SDK发现有多重版本的驱动时，它就会把最高的版本视为默认。但是程序员可以从下拉菜单中选择所需的版本。

该 MDD 用于辅助下列任务：

- 集成定制驱动到 SDK。
- 修改已有驱动并将变体集成到 SDK 中。
- 修改 BSP，例如添加新的软件服务。

IP 模块和驱动

IP 模块和驱动可如表 6-1 所示一样特性化。正如表中所示， SDK 能为任一类型的 IP 模块识别和加载任何驱动源代码。

表 6-1 : SDK 支持的各种 IP 模块与驱动组合

	软 IP （PL）	PS IP （PS）
赛灵思驱动	支持	支持
定制驱动	支持	支持

PS IP

PS IP 模块系组成 Zynq-7000 AP SoC PS 一部分的硬件模块。该 PS 包括多个外设 （USB、 I2C、以太网等）以及 ARM Cortex-A9 处理器。

该应用需要驱动支持访问外设。提供裸机和 Linux 驱动。该SDK为所有 PS IP 外设提供预构建的裸机驱动和 Linux 内核（赛灵思支持）。

软 IP

软 IP 模块是作为设计组成部分构建并加载到 Zynq-7000 AP SoC PL 中的硬件模块。 PL 中的部分软件 IP 模块示例有 VDMA、 AXI 以太网和第三方 IP 模块。

该应用需要驱动支持访问外设。提供裸机和 Linux 驱动。裸机驱动由 SDK 提供， Linux 驱动由赛灵思提供的 Linux 内核源树提供。

必须遵循下列步骤，才能够在 SDK 中使用第三方 IP 驱动：

- 为 IP 模块创建驱动。
- 创建相关的数据定义文件 (MDD) 和数据生成文件 (Tcl) 供 SDK 使用。
- 选择驱动路径作为 SDK 项目的外部库。 SDK 现在能够正常地识别和加载驱动。

裸机驱动和应用开发流见第 131 页的图 6-7。

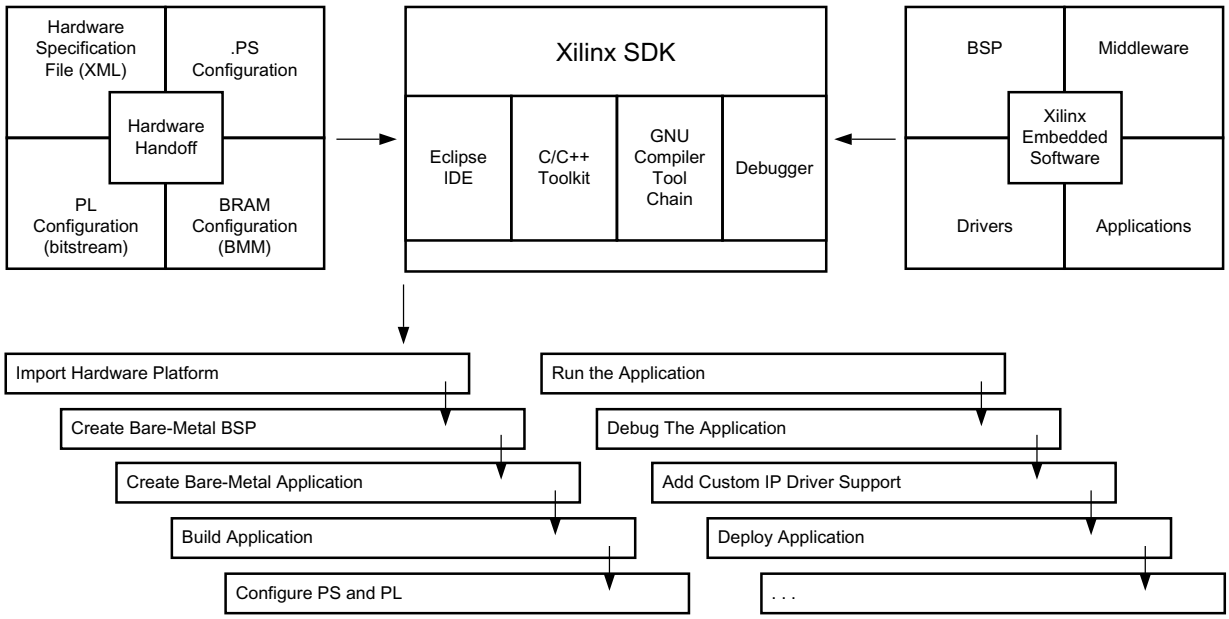

图 6-7 : 裸机应用开发简介。

如欲了解详情，请参阅 《Zynq-7000 All Programmable SoC 软件开发人员指南》(UG821) [参照 7]。

赛灵思驱动流程

软 IP 驱动随工具安装程序一起提供。这些驱动能直接从标准驱动位置导入到 BSP 中。

有两种修改和使用标准驱动的方式：

- 方法 1：
  - 在默认驱动路径中修改驱动源代码。
  - 再加载和构建 BSP。这一行为将导致修改后的驱动被加载。
- 方法 2：
  - 在本地路径创建驱动文件夹结构。
  - 复制标准驱动并分配不同的版本号码。
  - 修改源代码和对应的 MDD 文件。
  - 在 SDK 中指向本地库路径，以便让 SDK 使用修改后的驱动。

关于 MDD 参数的详情，请参阅生《成基本软件平台参考指南》(UG1138) [参照 23] 中的“微处理器驱动定义 (MDD)”信息。

定制驱动流程

定制 IP 软件可以库或驱动的形式存在。赛灵思 MDD 和 MLD 文件提供了为定制 IP 模块创建驱动所需的功能，让开发人员能够：

- 开发新驱动并把它们集成到 SDK 中
- 自定义现有驱动
- 自定义为 OS 工具链定制的 BSP 格式和文件夹结构
- 自定义基于硬件的 OS

MLD 格式描述可用于定制库和操作系统的参数。关于必须通过 Libgen 工具配置的用户编写库和操作系统的详情，请参阅 《生成基本软件平台参考指南》(UG1138) [参照 23]中的“微处理器软件规格 (MSS)”。

MDD 文件内置用于定制或创建软件驱动的指令。关于 MDD 格式，以及可用于定制或创建驱动的参数的详情，请参阅《生成基本软件平台参考指南》(UG1138) [参照 23] 中的“微处理器驱动定义 (MDD)”。

这种修改和使用定制驱动的方法与第 131 页的“赛灵思驱动流程”描述的用于赛灵思驱动的方法二相同。

Linux 驱动开发

Zynq-7000 AP SoC 上的 Linux 同时需要用于 PS Ip （外设）和软件 IP （PL 中实现的定制逻辑）的驱动。驱动也可作为内核的一部分或是作为可加载内核模块添加到 Linux 中。赛灵思提供针对 Zynq-7000 AP SoC 的完整 Linux 源代码。如欲了解详情和下载复本，请参阅赛灵思网页中的官方 Linux 内核 [参照 98]。赛灵思维基的 Linux 驱动页面列出了赛灵思提供的驱动有关信息清单。赛灵思维基的 Linux 页面还提供了与赛灵思提供的 Linux 相关解决方案的有关信息。

赛灵思在发布赛灵思工具时，会发布用于 Zynq-7000 AP SoC 评估板 （ZC702、 ZC706 和 ZedBoard）的预构建镜像。

设备树

关于添加或移除的硬件的信息（比如 PL 中的 IP 模块），可使用设备树发送到内核。设备树还可用于传递与硬件有关的特定信息。

赛灵思维基的构建设备树页面介绍如何使用赛灵思 SDK 构建设备树 BSP。在创建设备树 BSP 时， SDK 需要下列两个文件：

- 来自 Vivado 工具的硬件交接文件。
- 来自 SDK 库中的设备树版本的设备树文件 （`.tcl` 和 `.mld`）。

根文件系统

赛灵思提供用于 Zynq-7000 AP SoC 的初始根文件系统：ramdisk/initrd。其他受支持的文件系统可在系统启动后安装在根文件系统上。关于如何构建和修改根文件系统,赛灵思维基有更详细的说明。

Linux 内核调试

赛灵思 SDK 的系统调试器可通过 JTAG 用于 Linux 内核调试。该系统调试器是一种异构多核调试器，能实现 MicroBlaze 和 ARM 核心的同时调试。在 Linux 内核运行在 SMP 模式下时，通过系统调试器可以单独地控制各个处理器核心。 PL 地址空间可供调试，而且在调试过程中 PL 储存器空间显示在储存器视图中。

SDK 的帮助功能中有关于如何使用系统调试器调试 Zynq-7000 AP SoC 上的 Linux 内核的介绍。SDK 帮助功能还介绍了如何在调试配置过程中添加 Linux 内核符号文件的方法，以及在 Windows 环境中从 Linux 主机映射编译路径到 Linux 源树的步骤。

默认条件下，在使用 *attach to running target* 模式调试时， PL 寄存器不可访问。环境变量 HW_SERVER_ALLOW_PL_ACCESS 必须使用 SDK 外壳程序设置，而 hw_server 则采用下列次序启动：

1. 关闭 hw_server 的所有运行实例。
2. 从 SDK 启动终端命令提示符并设置 HW_SERVER_ALLOW_PL_ACCESS。
3. 从相同的外壳程序启动 hw_server。

此外 GDB 调试器可通过 JTAG 用于 Linux 内核的调试。ARM DS-5 也支持 Zynq-7000 AP SoC 器件上的 Linux 内核调试。

应用开发者

针对裸机或 Linux 或 FreeRTOS 等操作系统开发出了相关可运行应用。本章节介绍如何开发、调试和分析用于运行 在 Zynq-7000 AP SoC 上的 Linux 和裸机平台上的应用。同时提供体现 SDK 工具如何协助开发、调试和分析软件应用的 SDK 工具的用例。

裸机

该 SDK 支持在导出的硬件设计上的软件应用开发。该 SDK 从 Vivado 工具导出的设计中提取硬件信息。依据该信息 SDK 提供完整的应用开发环境。应用开发环境需要包含硬件初始化、驱动、 TCP/IP 栈 (lwIP) 等成套库和 FAT 文件系统 (FFS) 在内的板支持包。

赛灵思 SDK 提供两种类型的模板：

- 用于 Zynq-7000 AP SoC 的硬件 （纯 PS）模块。使用该模板，无需使用 Vivado 工具创建设计即可开发应用。有面向 ZC702、 ZC706、 ZedBoard 和 MicroZed 板的模板可用。
- 应用模板。使用可用外设可以为任何硬件设计生成预定义应用，例如 *hello world* 和一个空应用。 Lwip 等其他应用则可在 SDK 中以最小工作量创建。

SDK 还能以批处理方式使用，以便用户在无需调用 SDK IDE 的情况下通过命令行界面创建和构建应用。

更详细说明请参阅 《赛灵思软件开发套件 (SDK) 帮助》(UG782) [参照 6]。

赛灵思 SDK 配套有一套库，可供您用于辅助独立应用的开发。如需了解更多信息，请参阅：第 104 页的“用于独立系统的赛灵思库”。

用例：开发、调试和集成

图 6-8 中的对话框展示了 C/C++ 应用的创建。如需了解更多信息，请参阅 SDK 帮助 [参照 6]：赛灵思 C/C++ 新项目向导。

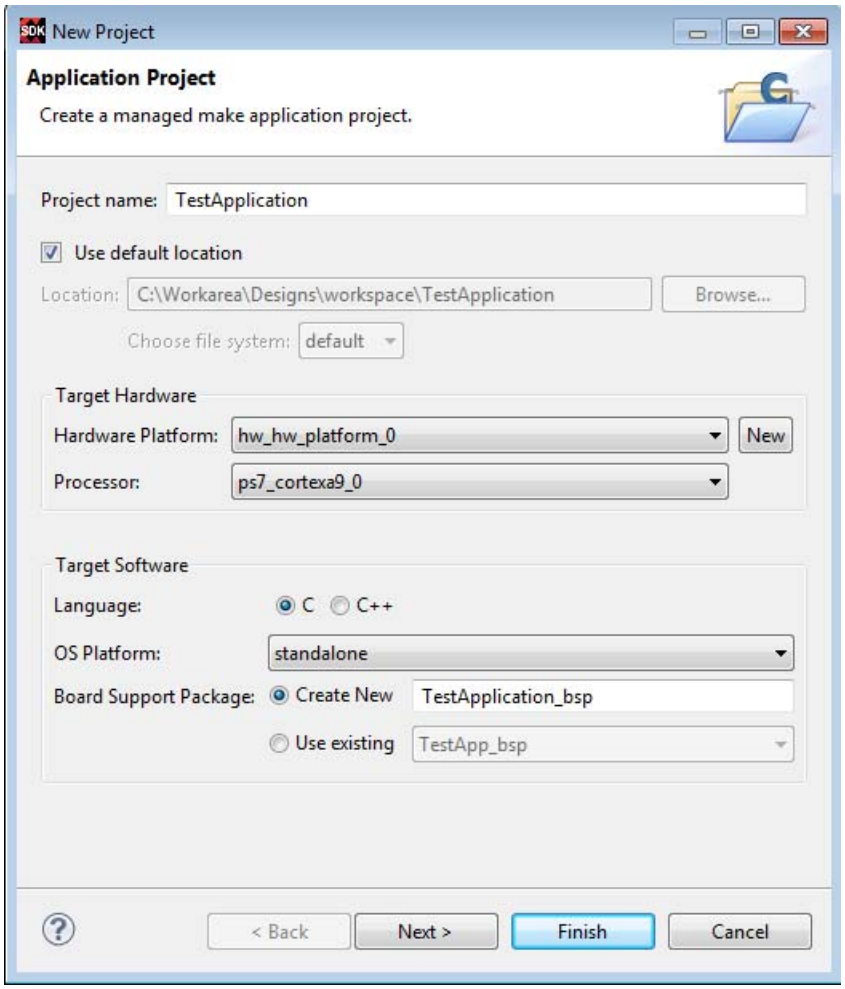

图 6-8 : C/C++ 应用对话框

在创建完成应用后，必须构建应用。如需了解更多信息，请参阅 SDK 帮助 [参照 6]：构建项目网页。

关于运行和调试应用的更多介绍，请参考 SDK 帮助 [参照 6]：运行、调试和剖析项目。

Linux 应用开发者

Linux 应用使用包含 Linux 内核、库、文件系统和驱动 API 在内的 Linux 软件栈。

部分影响 Linux 应用开发架构选择有：

1. 对称多处理 (SMP) 或非对称多处理 (AMP) 系统。
2. AMP 系统中的资源共享。需要映射储存器到单独核心的 AMP 系统。更多信息请参阅样本 AMP 配置。
3. SMP 系统中断亲和性。
4. 第三方库。如需了解更多信息,请参阅：第 104 页的“库和中间件”。应考虑在专属项目上使用的开源库和软件的许可。
5. 安全执行，例如 TrustZone 执行。
6. 用于定制 IP 的驱动支持。
7. 处理器间通信机制。如欲了解更多信息，请参阅《MicroBlaze 如何能与 Zynq-7000 AP SoC 和平共处》[参照 82]。

关于架构设计考虑因素的更多信息，请参阅第 4 章"软件设计考虑事项"。

赛灵思提供可在架构设计和评估阶段使用的文档、基本软件模块和参考设计。如需了解更多信息,请参阅赛灵思 Linux 维基页 [参照 54]。

应用开发流程

赛灵思 SDK 提供开发 Linux 应用所用的成套工具和建议开发流程。详细描述请参阅 《Zynq-7000 All Programmable SoC 软件开发指南》(UG821) [参照 7] 中的“软件应用开发流程”章节。

或者，可使用内部开发或第三方解决方案开发 Linux 应用。对用于开发面向 Zynq-7000 AP SoC 的应用的工具而言，需要得到下列支持：

- 包括主机上的交叉开发工具链的构建系统。
- Linux 内核、设备树、软件栈和库。
- Zynq-7000 AP SoC Linux 目标或模拟平台。

用于 Linux 应用开发的赛灵思工具流程

赛灵思以 PetaLinux 工具流和赛灵思 Linux 发行版本的组成部分的形式提供 Linux 驱动、BSP 和源代码。PetaLinux 流程是专为 FPGA 和与 Zynq-7000 AP SoC 类似的器件设计的。它为 Zynq-7000 AP SoC 提供配置和 Linux 部署功能，包括完整工具链、 BSP 和赛灵思评估板。关于配置 PetaLinux 的更多信息，请参阅赛灵思维基的 PetaLinux 页面。

PetaLinux 的详细介绍请参阅第 97 页的“OS 和 RTOS 选择”。

图 6-9 所示的是 Linux 应用开发中使用的赛灵思工具。

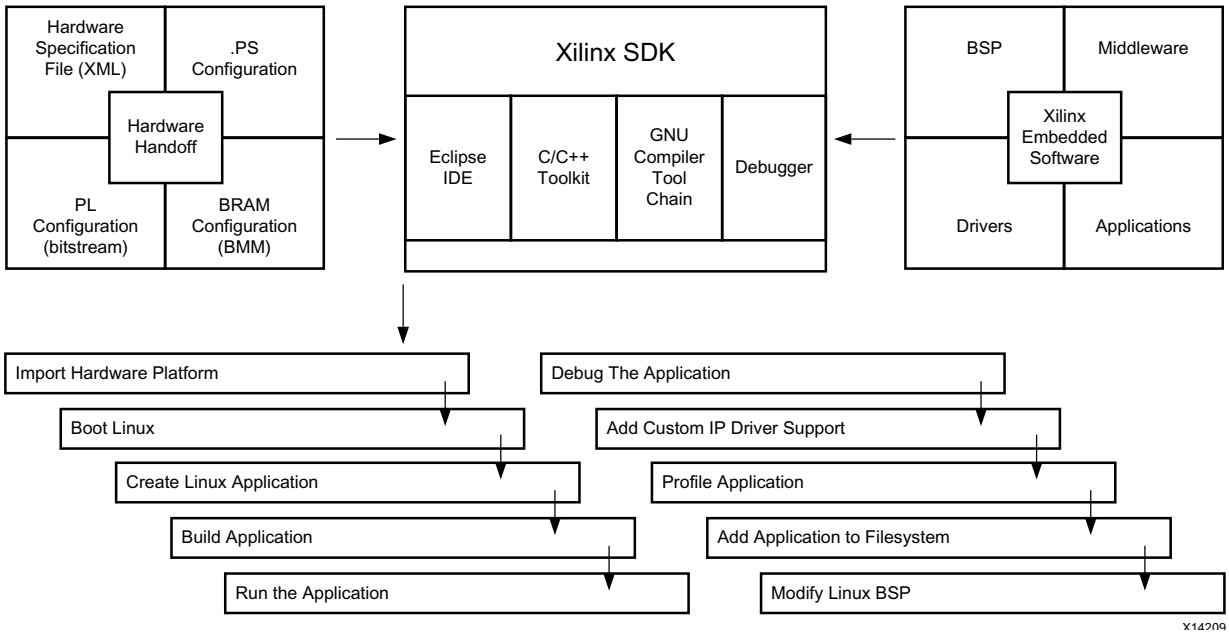

图 6-9 : 使用赛灵思工具的 Linux 应用开发

使用赛灵思 SDK 的应用开发

赛灵思 SDK 支持用 于 Zynq-7000 AP SoC 的 Linux 应用开发与调试。它支持 ARM Linux GCC 工具链 (Sourcery CodeBench Lite) 并为 Linux 应用开发提供各种赛灵思工具。该赛灵思 SDK 还与 Linux 应用模板和支持库捆绑，以帮助开发人员以最小设置工作量评估 Zynq-7000 AP SoC。

关于 Linux 应用开发流的详情，请参阅 《Zynq-7000 All Programmable SoC 软件开发指南》(UG821) [参照 7]。

图 6-10 所示的是如何使用 OpenCV 示例应用模板在 SDK 中创建 Linux 应用开发项目。SDK 提供了 OpenCV 应用所需的库。

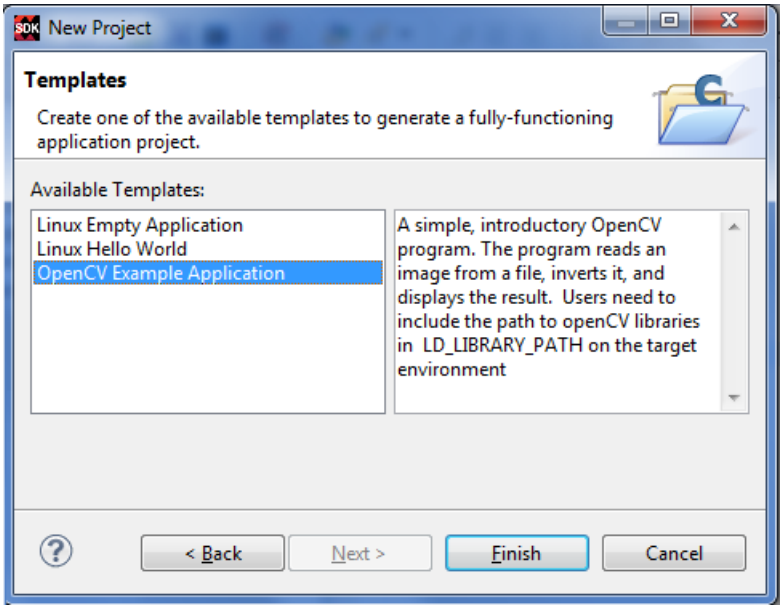

图 6-10 : 在赛灵思 SDK 中创建新的 Linux 应用

图 6-11 所示的是在使用 SDK 构建 Linux 应用时，如何选择不同的版本设置和库。

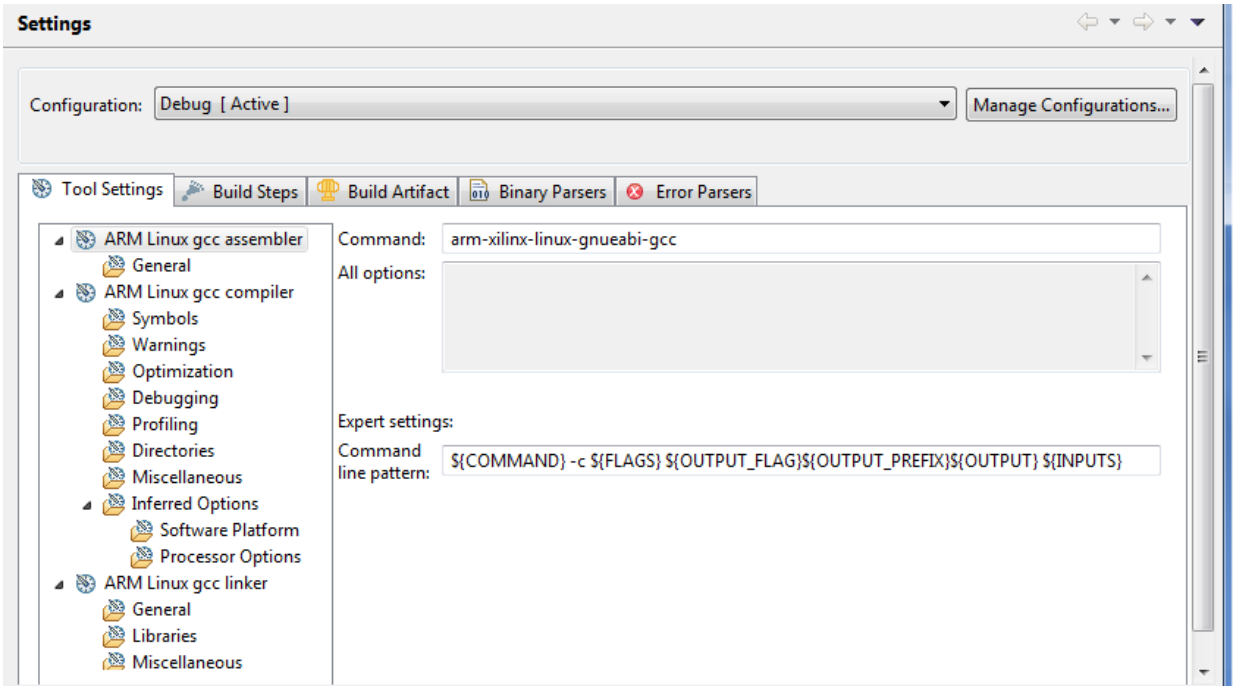

图 6-11 : 赛灵思 SDK 版本设置和库

SDK 能让用户使用 IP 地址连接到 Zynq-7000 AP SoC Linux 目标。在连接完成后，应用可执行文件就会拷贝到 Linux 目标并运行。

Linux 应用调试

Linux 应用可使用 GDB 或赛灵思系统调试器在 Zynq-7000 AP SoC 上调试。赛灵思 SDK 还能实现本地目标或远程目标的 Linux 应用调试。使用针对 GDB 的 gdbserver 或针对系统调试器的 TCF 代理，赛灵思 SDK 可支持 Linux 应用调试。TCF 代理进程必须运行在 Linux 目标上，以使用系统调试器调试 Linux 应用。

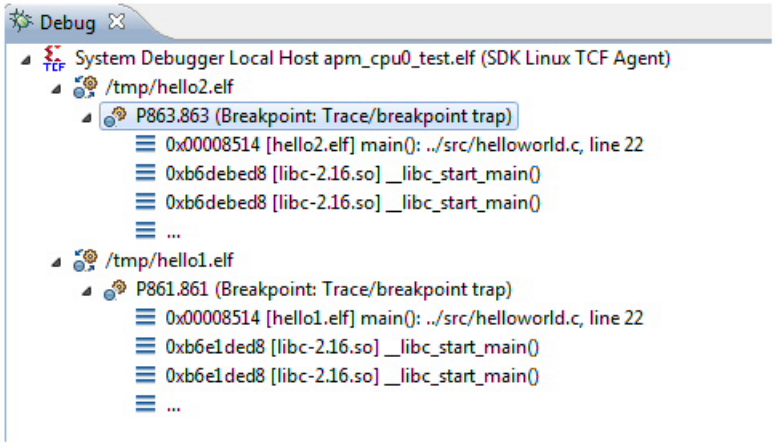

图 6-12 : 赛灵思 SDK 版本设置和库

Linux 应用还可以使用 ARM DS-5 等其他 IDE 开发和调试。如果希望采用命令行方法，可以使用内部开发的构建系统，也可以使用从 OS 厂商处获得的构建系统。Zynq-7000 AP SoC 生态系统受各种 OS 合作伙伴的支持。OS 合作伙伴的清单可在第 97 页的“OS 和 RTOS 选择”中找到。

赛灵思 SDK 工具和封装

赛灵思为开发 Zynq-7000 AP SoC 软件提供了工具和软件包。开发实用工具的具体发布有：

- 在 SDK IDE 中发布的赛灵思工具。
- 包括裸机 BSP 和驱动在内的软件包，以及源代码。
- 通过赛灵思 Git 发布的软件。

赛灵思 SDK 中发布的工具

赛灵思 SDK 中发布的工具和实用工具见下列章节的描述。

启动镜像创建

赛灵思 SDK 为创建启动镜像提供便利。 Bootgen 能把启动加载器 FSBL、比特流和应用可执行文件 （包括 U-Boot）合为一体，生成 `.bin` 或 `.mcs` 格式的启动镜像文件。该 SDK 还提供了 *Create Boot Image* 向导选项，用于添加分区镜像和创建可启动镜像。如欲了解更多信息，请参阅下列内容：

- 《赛灵思软件开发套件 (SDK) 帮助》(UG782) 中的 [参照 6] “为应用创建 Zynq 启动镜像”。
- 《Zynq-7000 All Programmable SoC 软件开发指南》(UG821) [参照 7] 中的“使用 Bootgen” 附录 （请点击本链接）。
- 如何使用 Xilinx SDK 创建 Zynq 启动图像

可编程实用工具

闪存编程

闪存变成实用工具可让用户将 .bin 和 .mcs 文件下载到板载闪存设备上。图 6-13 显示的是该实用工具支持的闪存设备。

图 6-13 : 程序闪存实用工具支持的闪存设备

FPGA 编程

PL 码流可以用不同方式编程。一种方式是使用 SDK GUI 中的 “Program FPGA” 实用工具。

调试器

赛灵思 SDK 提供支持 ARM 和 MicroBlaze 调试的异构调试器。调试功能得到 GUI 和命令行接口的支持。命令行接口支持 Tcl 脚本编写，能在 JTA G模式下实现命令执行。下列调试器受赛灵思 SDK 的支持：

- 系统调试器
- XSDB：赛灵思系统调试器 (命令行流程)
- XMD：赛灵思微处理器调试器
- GDB 调试器

为确保 MicroBlaze 核心对调试器可见，在 Vivado 中创建硬件设计时必须启用 MDM。

SDK 系统调试器提供的硬件-软件调试功能有助于调试运行在 Zynq-7000 AP SoC 上的代码。该调试器还便利硬件-软件交叉触发。

如欲了解更多信息，请参阅：第 7 章"调试"。

性能分析

性能分析是发现代码瓶颈的重要步骤。赛灵思 SDK 提供代码配置器和性能分析器。如欲了解更多信息，请参阅：第 2 章"剖析 (Profiling) 和划分 (Partitioning)"。

应用模板

图 6-14 显示的是搭配在 SDK 中用于裸机编程的应用模板。

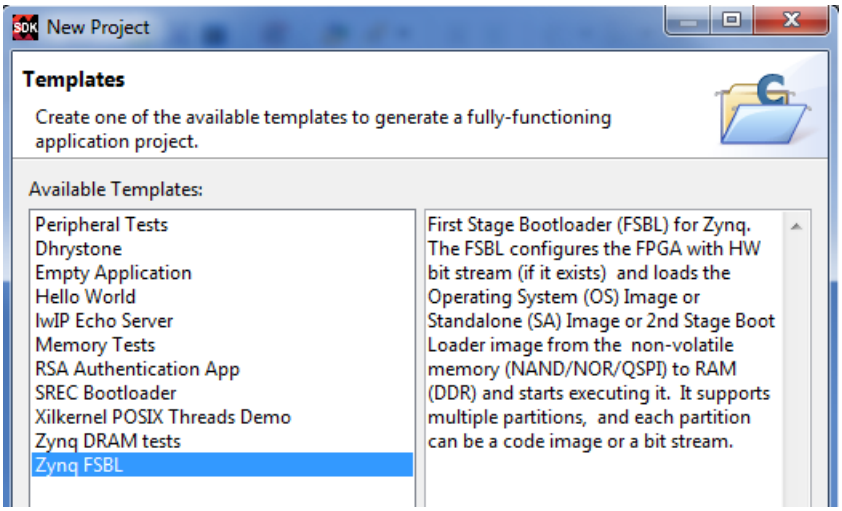

图 6-14 : SDK 应用模板

赛灵思 Git 软件解决方案发布

赛灵思在赛灵思 Git 页面上发布用于 Zynq-7000 AP SoC 和 MicroBlaze 目标的系统软件。发布在 Git 页面上的部分封装包括：

1. Linux xlnx：赛灵思 Linux 内核
2. U-Boot-xlnx：赛灵思 U-Boot 库
3. 器件树赛灵思 SDK Linux 设备树生成器
4. meta-xilinx：用于 Yocto/OE- 核心的赛灵思设备和板支持

赛灵思软件开发工具

赛灵思提供的软件开发工具包括 SDK 和赛灵思 Yocto 项目。

赛灵思软件开发套件 (SDK)

赛灵思软件开发套件 (SDK) 是一个基于 Eclipse 的 IDE，内含赛灵思工具和软件包。它能实现用于赛灵思 FPGA 和 Zynq-7000 AP SoC 器件的嵌入式软件开发。SDK 包含必备的软件包，例如用于赛灵思评估板、外设和 IP 设备驱动的预定义硬件配置，裸机 BSP 库，配备所需的支持库的固件以及应用模板。这些软件包以最新的源代码形式发布，以便根据需要修改。 SDK 还包含用于创建启动镜像，为处理器编程以及测试性能的其他工具。

赛灵思 SDK 是免费产品，可从 http://china.xilinx.com/tools/sdk.htm 下载。

PetaLinux

PetaLinux 为在 All Programmable 器件上开发和部署基于 Linux 的软件提供解决方案。PetaLinux 工具集包括安装程序、开发工具、用于赛灵思评估板的基于 Linux 的 BSP 和库框架。赛灵思维基百科包含对 PetaLinux 的详细介绍。该维基页还包含关于总体赛灵思 Linux 解决方案和赛灵思开源软件的文档。

赛灵思 Yoco 工程

针对在 MicroBlaze 和 Zynq-7000 AP SoC 上构基于 Linux 的定制系统，赛灵思 Yocto 项目提供了相关模板、工具和方法。所包含内容有元文件、开发板和 QEMU 仿真器。赛灵思维基 Yocto 页面提供了赛灵思 Yocto 的发布链接。这其中包括 meta-xilinx，这是一种支持 Linux 内核版本、 U-Boot和 Poky 分发的 Yocto BSP 层。

赛灵思 Yocto 项目还支持来自非赛灵思实体的相关驱动和软件包的上游化。欢迎客户和 OS 厂商开发有助于强化产品和整个赛灵思社区的合规软件和上游驱动。

关于赛灵思 Yocto 项目的更多介绍，请参阅一下内容：

- 赛灵思 Yocto 维基页 [参照 58]
- Yoco 工程维基页 [参照 102]

第 7 章

调试

本章介绍下列关于调试方法的信息：

- 第 142 页的“简介”：本节概括介绍使用 Zynq®-7000 AP SoC 开发应用程序时的调试方法。
- 第 143 页的“纯软件调试”：使用赛灵思软件开发套件 (SDK) 完成嵌入式软件调试。 SDK 调试方法提供了全面的调试环境。
- 第 146 页的“基于仿真的调试”：通过仿真在软件环境中模拟最终设计的行为。通过注入激励并观察输出结果，借助仿真验证设计功能。
- 第 147 页的“板调试”：可使用 SDK 工具实现硬件的软件调试。调试有助于找出 FPGA 或结构中的问题。
- 第 148 页的“硬件与软件协同调试”：赛灵思通过提供解决方案，让您能够同时调试处理系统 (PS) 和可编程逻辑 (PL) 中的外设。
- 第 148 页的“虚拟平台”：虚拟平台是指可以模仿硬件行为的软件系统。它们是嵌入式平台的高速功能性模型。

由于可编程逻辑 (PL) 密度和处理器系统的复杂性，本节介绍的调试方法对于使用 Zynq®-7000 AP SoC 的应用程序开发至关重要。

Zynq-7000 AP SoC 的测试和调试功能与 ARM CoreSight 技术相结合，使用户可以利用侵入和非侵入调试方法调试处理系统 (PS) 和 PL。用户可以调试完整的系统，包括 PS 和 PL。除了调试软件外，用户还可以调试 PS 中的硬件点以及 PL 中用户选择的硬件点。

测试和调试功能基于 ARM CoreSight v1.0 架构，并定义四类 CoreSight 技术组件：访问与控制，跟踪源，跟踪链接，跟踪吸入。另外，调试访问端口 (DAP) 不属于 CoreSight 组件，可提供对 CoreSight 组件和其他系统特性的访问。

简介

软件工具能够进行侵入式和非侵入式调试。具备用于侵入式调试的断察点。使用可捕捉指令流的 ARM 程序跟踪宏单元 (PTM) 实现非侵入式调试。只捕捉程序流中的变化，以实现流压缩。通过 2 至 32 跟踪输出端口引脚 （MIO/EMIO 的一部分）将数据发送到片外。另一种选择是将跟踪数据加载到嵌入式跟踪缓冲器 (ETB) 中，使用第三方工具通过 JTAG 上传跟踪数据。

可进行纯软件调试，或者将 Vivado® 集成逻辑分析器 (ILA)内核添加到设计中，实现硬件与软件组合调试。 PL JTAG 端口与 ARM 调试访问端口 (DAP) 采用菊链连接。通过 PL JTAG 访问 Vivado 逻辑分析器内核。ARM 和第三方工具连接到 ARM 调试访问端口 (DAP)。

如需了解有关 PS 和 PL 调试配置的更多信息，请参阅 《Zynq-7000 All Programmable SoC 技术参考手册》(UG585) 中的“JTAG 和 DAP 子系统” （点击此链接）和“系统测试与调试” （点击此链接）章节， [参照 4]。

PS 调试

Zynq-7000 AP SoC PS 提供以下针对系统级跟踪的功能：

- 利用单个调试器连接实现整个系统的调试和跟踪可见性
- SoC 子系统之间的交叉触发

- 单个流中的多源跟踪
- 比以往解决方案更高的数据压缩率
- 标准程序员工具支持模型
- 自动拓扑探索
- 针对第三方软核的开放式接口
- 低引脚数选项

PL 调试

赛灵思提供用于 PL 测试与调试的架构跟踪监视器(FTM)。它基于 ARM CoreSight 架构，是 Zynq-7000 AP SoC 内 CoreSight 系统中跟踪源类的一个组件。 FTM 从 PL 接收跟踪数据，并把数据变成跟踪数据包格式，以与来自其他跟踪源组件——例如程序跟踪宏单元 (PTM) 和仪器跟踪宏单元 (ITM)——的跟踪数据包组合。凭借此功能， PL 事件可与 PS 事件同时跟踪。FTM 还支持除跟踪倾倒功能外的 PS 与 PL 之间交叉触发。另外，FTM 还提供了 PS 与 PL 之间的通用调试信号。

PL 测试与调试特性为：

- ARM CoreSight 兼容
- 来自 PL 的 32 位跟踪数据
- 来自 PL 的 4 位跟踪 ID
- PL 与 PS 之间的时钟域交叉
- FIFO 跟踪数据包缓存，用以吸收 PL 跟踪数据突发
- 通过生成溢出数据包实现 FIFO 溢出指示
- 跟踪数据包兼容于 ARM 跟踪端口软件和硬件
- 进出 PL 的触发信号
- 进出 PL 的通用 I/O

纯软件调试

使用赛灵思软件开发套件 (SDK) 进行嵌入式软件调试。SDK 调试方法提供了全面的调试环境。调试窗口显示会话状态，包括调用栈、源代码、反汇编代码、进程存储器、寄存器信息和 XMD 控制台。使用标准调试器命令设置断点并进行执行控制。

典型的应用程序开发与调试过程概述包含图 7-1 中所示步骤。

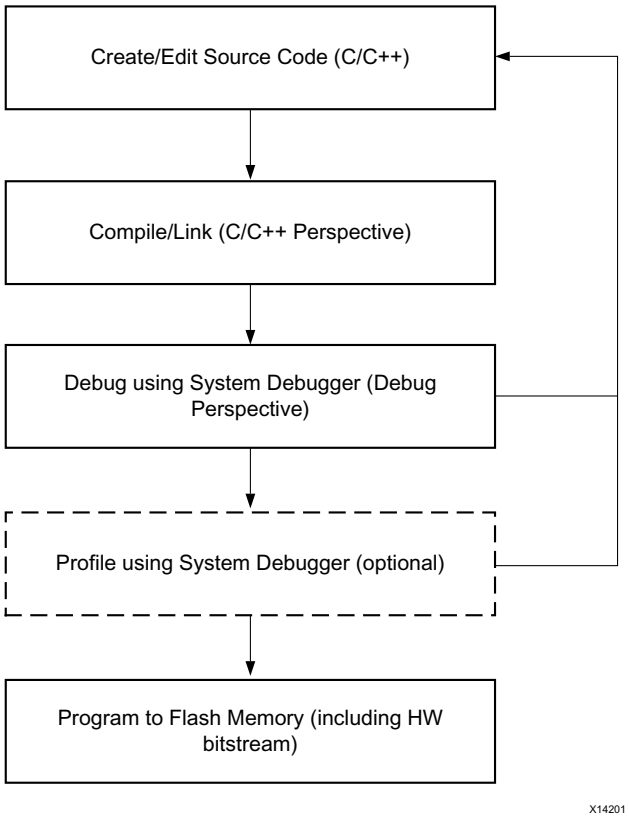

图 7-1 : 应用程序开发与调试简介

系统调试器和目标连接

当调试应用时，系统调试器通过 JTAG 使用目标通信框架 (TCF)连接目标板。调试开始时，系统调试器在本地主机上运行硬件服务器代理。或，当一台机器正在运行调试应用程序，另一台机器连接到电路板时，例如远程实验室环境，也可在远程主机上运行。系统调试器目标连接窗口允许对受调试板进行管理。

系统调试器

赛灵思系统调试器 (XSDB) 是 Zynq-7000 AP SoC 的首选软件调试方法。系统调试器能够在裸机或基于 Linux 的系统上在相同会话中对双 ARM 和多 MicroBlaze™ 处理器实现异构调试。它还包含层次剖析、对 ARM NEON 库的支持以及 XSDB 命令接口。连接电路板之后，可以识别系统中每个 CPU 的状态。调试器还提供了将目标应用程序(.elf 文件)连接至每个 CPU 的方式，为 Linux 调试分配符号，查看和设置寄存器，查看和设置断点，以及监控存储器位置。通过 XSDB 向系统发布命令，可使用停止和断点查看反汇编的应用程序代码。

如需了解更多信息，包括如何连接电路板，请参阅 《赛灵思软件开发套件帮助》(UG782) [参照 6]。

赛灵思系统调试器 (命令行)

XSDB 是用于加载和控制 CPU 的命令接口。可从命令行或使用 SDK 中的交互式 GUI 调用它。它包含一个工具命令语言 (Tcl) 接口，支持重复或复杂任务的脚本编写。 XSDB 还提供了相应方法来读取寄存器和存储器位置，以及设置代码断点。 XSDB 支持针对命令使用的动态帮助。

剖析 (Profiling)

SDK 中的系统调试器还支持代码剖析功能，能够显示函数调用执行时间，以便找出应用程序中最耗时的部分。函数调用可隔离，可对调用的子函数进行剖析，而且用户可对照源代码进行交叉探测。利用剖析来确定在哪里可将软件应用转换为PL或DSP功能，以加速系统性能。

GNU 工具链支持

除了系统调试器以外， SDK 还支持使用 GNU 工具链进行应用程序调试，工具链包括 GNU 调试器 (GDB) 和用于应用程序剖析的 gprof。

驱动和 BSP 调试

软件应用必须链接到或运行在使用所提供 API 的软件平台上。因此，在SDK 中创建和使用软件应用之前，必须创建一个板级支持包 (BSP)。板级支持包是库和驱动的集合，用以构成应用软件栈的最底层。

SDK 包含一个用于应用程序开发的独立板级支持包。这个程序包是一个简单的半托管式单线程环境，提供标准输入/输出以及处理器访问等基本功能。

可使用 SDK 板级支持包设置对话框配置板级支持包和库。可在 SDK 工作空间中创建多个板级支持包。这使用户可以同时使用不同的板级支持包配置，并将应用程序从针对一个板级支持包调整为针对另一个板级支持包。

可利用驱动创建应用程序。使用赛灵思系统调试器调试应用程序和驱动；系统调试器可以采用独立调试模式，或连接到运行着的目标。调试器可针对每个内核进行配置；在通过 JTAG 下载和运行独立应用程序之前，用户可选择的选项可运行初始化文件。

如需了解更多信息，请参阅 《赛灵思软件开发套件帮助》(UG782) [参照 6] 和 《Zynq-7000 All Programmable SoC 软件开发指南》(UG821) [参照 7]。

OS 调试

赛灵思系统调试器可使用“连接到运行目标”模式，用以调试操作系统和相关支持驱动。系统调试器配置允许用户添加内核符号文件，并提供编译路径映射以实现源代码调试。内核源代码必须用以下 flags set 进行编译。

```
CONFIG_DEBUG_KERNEL=y
CONFIG_DEBUG_INFO=y
```

调试器也可以调试在 PL 地址空间中运行的代码，而且调试过程中存储器视图支持 PL 存储器空间。

当使用“连接到运行的目标”模式进行调试时，默认情况下 PL 寄存器无法访问。环境变量 HW_SERVER_ALLOW_PL_ACCESS 需要通过 SDK shell 进行设置，然后使用 hw_server ，如下所示：

1. 关闭任何运行中的 hw_server 实例。
2. 从 SDK 启动终端命令提示符，并设置 `HW_SERVER_ALLOW_PL_ACCESS`。
3. 从同一个 shell 启动 hw_server。

GDB 调试器也可用来进行 Linux 内核调试。与系统调试器相同， GDB 是一个远程调试器，它连接到一个用来在 TCP 与 JTAG 之间进行转换的服务器。 ARM DS-5 Development Studio 也支持针对 Zynq-7000 AP SoC 的 Linux 内核调试。

如需了解更多信息，请参阅 《赛灵思软件开发套件帮助》(UG782) 中的“使用赛灵思系统调试器连接和调试 Linux 内核” 主题 [参照 6] 中的“使用赛灵思系统调试器连接和调试 Linux 内核 ”主题。

Linux 应用调试

开发者可选择系统调试器的 Linux 应用调试模式来调试 Linux 应用。赛灵思 SDK 还可实现针对 Linux 应用程序的本地目标或远程目标调试。赛灵思 SDK 使用 TCF 代理可支持 Linux 应用程序调试。TCF 代理进程必须运行在 Linux 目标上，用于使用系统调试器调试 Linux 应用程序。

GDB 调试器也能够调试 Linux 应用程序。远程 ARM Linux 应用程序调试器可与运行在 Zynq-7000 AP SoC 目标上的 gdbserver 进程通信，以支持调试。系统调试器窗口示例如图 7-2 中所示。

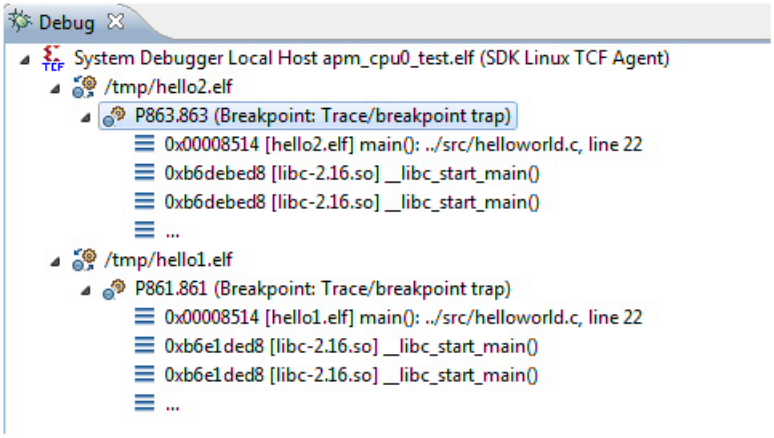

图 7-2 : SDK 系统调试器窗口

除了赛灵思 SDK，开发人员还可使用 IDE，例如 ARM DS-5 Development Studio，进行 Linux 应用程序开发与调试。有些开发人员在开发 Linux 应用程序时倾向于使用命令行工具。这些开发人员可使用内部工具或 OS 厂商提供的工具。Zynq-7000 AP SoC 生态系统得到了多个 OS 合作伙伴的支持。 OS 合作伙伴列表请见第 97 页的“OS 和 RTOS 选择”。

走线

Cortex-A9 包含一个程序跟踪模块 (PTM)，该模块兼容于 ARM CoreSigh t程序-流程跟踪功能。该模块适用于在任意 Cortex-A9 处理器，并实现完全的处理器指令流程可见性。 Cortex-A9 PTM 能利用循环计数观察到所有代码分支和程序流程变化，以实现剖析。 PTM 模块和 CoreSight 设计套件能够不唐突地跟踪多个处理器的执行历史。指令执行结果可存储在片上缓冲器中，或者使用标准跟踪接口发送到片外，以提高开发和调试过程中的可见性。

具有 DSTREAM 的 ARM DS-5 Development Studio 支持针对 Zynq-7000 AP SoC 器件的 ETB 和 PTM 指令跟踪功能。Lauterbach Trace32 也支持来自 PTM、 FTM、 ETB 和 AXI Monitor 的跟踪数据。

如需了解 SDK 软件调试方面的更多信息，请参考以下资料：

- 《Zynq-7000 All Programmable SoC 软件开发人员指南》(UG821) [参照 7]
- 《赛灵思软件开发套件帮助》(UG782) [参照 6]

基于仿真的调试

仿真是指在软件环境中模拟最终设计的行为。仿真通过注入激励并观察输出结果，有助于验证设计功能。

系统仿真

使用 Zynq-7000 AP SoC 总线功能模型 (BFM) 在 RTL 级仿真 Zynq-7000 AP SoC。 BFM 支持 Zynq-7000 AP SoC 应用程序的功能仿真。它能模仿 PS-PL 接口和 PS OCM/DDR 存储器，实现 PL 的功能验证。BFM 以加密 Verilog 模块数据包的形式提供。使用包含在 Verilog 语法文件中的一系列连续 Verilog 任务来控制BFM操作。如需了解关于Zynq-7000 AP SoC BFM的更多信息，请参阅 《Zynq-7000 总线功能模型，数据手册 v2.0》(DS897) 中的产品规格 [参照 30]。

如果使用 IP 集成器创建设计，当生成设计仿真文件时， Vivado 工具会自动使用 Zynq-7000 AP SoC BFM。使用 IP RTL 和 Zynq-7000 AP SoC BFM 仿真 PL 逻辑。典型的用例是使用 GP 端口从 PS 生成事务处理。这可通过一个 Verilog 任务来实现，该任务被定义为 Zynq-7000 AP SoC BFM 的一部分。

当使用 PL 侧 DDR 存储器时， MIG 从接口将通过 AXI-interconnect 连接到 Zynq-7000 AP SoC BFM。如果使用 Micron 存储器模块，相关的 Micron 仿真模型必须在设计仿真过程中连接到测试平台。

验证定制 IP

可通过仿真以及在硬件上验证使用 Vivado 工具封装的定制 IP。当使用创建和封装 IP 功能创建定制 IP 时，应使用 “Verify the Peripheral IP using the AXI BFM Simulation Interface” 选项。选择该选项能确保使用 AXI BFM 从机和主机内核创建模板 IP 集成器设计，从而仿真内核功能。该模板设计使用的测试平台采用 AXI BFM IP 中定义的 API。如需了解 AXI 总线功能模块就 API 和响应代码方面的详细情况，请参阅 《LogiCORE IP AXI BFM Cores v5.0 产品指南 (AXI)》(PG129) [参照 31]。

当测试由 CIP 向导生成的 IP 核时，应修改默认测试平台模板。如需详细了解 HDL 与定制 IP 模板的映射过程，请参阅《Vivado Design Suite用户指南：创建和封装定制IP》(UG1118) 中的此链接 [参照 22]。另外，如果端口被暴露，在设计仿真之前必须相应修改模板 IP 集成器设计和模板测试平台。

板调试

设计实现后，必须在硬件上测试。可使用 SDK 工具进行硬件的软件调试调试有助于找到 FPGA 或架构侧的问题。

纯 FPGA 调试

可使用 Vivado 集成逻辑分析器 (ILA) 调试 PL。 ILA 可在综合后包含在 RTL 代码中或插入 PL。可对它进行配置以监测设计接口和网络。可在暴露具体问题的运行时间内指定触发条件。可监测针对调试进行标记的或连接到 ILA 的网络，以找到问题源。

纯处理器调试

采用 SDK 调试软件问题。 SDK 允许用户查看代码，查看变量值，跟随控制流，并建立和分析断点。

在使用 SDK 之前，必须将设计导出到硬件。导出过程会创建一个设计目录，其中包含来自 Vivado 工具用于描述硬件配置的文件。文件中包含关于所有设计 IP 模块及其存储器映射的信息。借助该信息， SDK 能够创建 IP 驱动。使用导出功能提供的信息可以 在SDK 中创建不同的软件应用。

在 SDK 中建立软件应用程序后，可使用设计比特流编程 FPGA，并在硬件上启动应用程序。然后通过系统调试器调试软件应用。

硬件与软件协同调试

Zynq-7000 AP SoC 具备出色的设计潜力，支持用户将处理器功能与定制 FPGA 逻辑相结合。这种高潜力带来的是更高的复杂性，亟需强大的工具集。赛灵思提供的解决方案可帮助用户同时调试 PS 与 PL 中的外设。

使用 Vivado ILA 和赛灵思 SDK 实现硬件和软件协同调试。 ILA 能够对网络针对调试进行标记，并进行分析。 SDK 支持设置断点或观察点，单步调试程序执行，查看程序变量和堆栈，查看系统存储器内容。协同调试用于调试当软件和硬件相结合时的系统级问题。

交叉触发

嵌入式交叉触发 (ECT) 是一种用来调试硬件与软件之间交互的交叉触发机制。利用 ECT， CoreSight 组件可以通过发送和接收触发信号与其他组件交互。 ECT 包含两个组成部分：

- CTM：交叉触发矩阵
- CTI：交叉触发接口

交叉触发可用来协同调试运行在处理器和 PL 硬件上的应用程序。针对这一目的可采用 Vivado 硬件管理器和 SDK。Zynq-7000 AP SoC 中的交叉触发端口通过与 ILA 的握手信号实现该功能。包含针对每个 ARM 内核的接口，TRIGGER_IN 和 TRIGGER_OUT。设计人员可以在 PS-PL 接口上选择任意一个或两个 ARM 交叉触发接口。另外，设计者必须使用 PL 中的 ILA IP，并在 SDK GU I中设置交叉触发。当发生硬件触发条件时，可中断处理器软件，同样，当出现软件断点时，可探查硬件信号。如需详细了解如何设置交叉触发和使用 Vivado 进行调试，请参阅 《Vivado Design Suite 教程：嵌入式处理器硬件设计》(UG940) [参照 15]。

定制 IP 硬件验证

CIP 向导能够在 IP 集成器中生成定制 IP 设计模板，并实现在硬件上调试。 “创建和封装 IP”功能使用户能够生成示例设计，利用 AXI Master 内核将 JTAG 连接到定制 IP。该设计根据创建的外设 IP 包含 AXI BFM 主机和从机内核，并将它们与外设 IP 结合在一起。在电路板上对相应比特流进行编程之后，可以修改生成的测试向量 Tcl 文件，以测试板上的受测外设 IP 功能。测试向量文件包含一系列用于验证 IP 的写入和读取。

虚拟平台

虚拟平台是指模仿硬件行为的软件系统。它们属于嵌入式平台的高速功能性模型。它们与硬件一样执行相同的软件二进制文件，并运行在传统台式机、笔记本或服务器上。虚拟平台能够较早地以更高抽象水平完成系统设计。它们支持在硬件 （电路板或 HDL）就位之前进行软件开发。它们还支持无限制的错误注入与测试，从而在早期开发阶段找出并消除漏洞。由于可以快速更改硬件参数，虚拟平台的修改比实体硬件更方便、更快速。

Zynq-7000 AP SoC 虚拟平台

Zynq-7000 AP SoC 虚拟平台使用典型 SoC 中的 PS 组件的模型，包括：

- Cortex-A9 MPCore 处理器
- 处理器周围的外设，例如 UART、以太网、 USB、 CAN 等
- 在可编程架构中常用的组件模型，例如 PCIe 技术和视频

虚拟平台可将模型连接到多种接口，包括以太网、 USB、串行控制台和图形显示器。

虚拟平台使开发人员可以在 Zynq-7000 AP SoC PL 中创建和集成新的器件模型。还可在包含 Zynq-7000 AP SoC 的电路板中创建和集成模型其他解决方案需要将定制 IP 的硬件实现连接到虚拟目标，与此不同，赛灵思虚拟平台可完全在软件中修改，体现定制硬件设计。

赛灵思虚拟平台对产品开发的所有阶段都能带来优势，涵盖架构前产品定义、系统设计、开发、集成、测试直到最终交付。

这些优势包括：

- 上市速度
  - 早期软件开发：软件团队可在硬件开发出来之前开发设备驱动和应用程序。
  - 快速的软件开发：与用于硬件设计与验证的仿真器不同，虚拟平台运行较快，可满足软件开发人员要求。
  - 更智能的软件开发：虚拟平台让软件开发人员可以探索新技术和方法，显著缩短开发时间。
  - 无需重新定位目标：寄存器和指令精确模型确保虚拟平台运行与硬件实现中采用相同的二进制文件，从而简化到物理平台的软件移植。
  - 不存在硬件和软件依赖性：开发人员无需共享稀缺的开发设备。
- 产品质量
  - 改善系统设计：同时实验多种软件和硬件配置，以获得最佳系统设计。
  - 完整的系统可见性与控制：可监控和调试任何寄存器和存储器位置，哪怕是物理设备上无法访问的位置点。
  - 改善测试迭代：可并行调试和测试软件与硬件，以实现更全面的测试覆盖。
  - 强化软件回归测试：软件构建与测试可每晚在虚拟平台上运行。
  - 强化故障注入和指标验证：虚拟平台能实现硬件故障注入，即使在物理设备上无法访问的位置点也不例外。

Cadence 虚拟系统平台

尽管传统虚拟平台能够实现新开发方法，带来产品设计变化，但是当产品团队从开发平台转到产品硬件上时，则无法满足要求。可扩展的虚拟平台产品：

- 方便、清晰的模型：Zynq-7000 SoC 虚拟平台能够即开即用。软件团队可利用它来立即开发 Linux、 RTOS 和裸机应用程序。
- 可扩展性：赛灵思与 Cadence 帮助开发人员能够利用新器件(在 Zynq-7000 So C可编程架构中或在电路板上例化)或系统模型扩展虚拟平台。可扩展的虚拟平台具有物理硬件的灵活性和可配置性。采用 Cadence 工具，可支持用 C 或 System-C 编写的模型以及用 SystemVerilog 或 VHDL 编写的模型。

XILINX
ALL PROGRAMMABLE™

第 8 章

SDSoC 环境

本章介绍以下关于在 SDSoC 环境中开展设计的信息：

- 第 152 页的“整体使用流程”：您应了解 SDSoc 设计环境的总体使用流程，以及所支持的 OS 和硬件平台。
- 第 154 页的“剖析”：除赛灵思 SDK 以外， SDSoC 也集成了剖析功能，使您能够对应用程序进行剖析，而无需代码导入。
- 第 154 页的“性能估算”：SDSoC 环境提供了相应的工具，便于您在检查硬件加速功能时快速估算性能改善情况。
- 第 154 页的“生成并运行完整的软件-硬件系统”：在运行性能估算流程后，您可重建系统，并看到所生成系统的实际性能。
- 第 154 页的“使用 C 语言调用 RTL IP 库优化性能”：您能够使用 C 调用 RTL IP 库来优化性能。
- 第 154 页的“使用 HLS 优化 IP 性能”：您可以使用 Vivado 高层次综合 (HLS)来编译应用程序代码，传送到 SDSoC 环境中的硬件，然后利用 SDSoC 工具的通用策略改善代码。
- 第 155 页的“优化系统性能”：凭借 SDSoC，您可利用各种技术来优化系统性能。
- 第 156 页的“调试系统”：SDSoC 允许使用 SDSoC 环境来创建和调试项目。 SDSoC 与赛灵思 SDK 系统调试器的调试基础设施和方法相同。
- 第 156 页的“性能测量与分析”：SDSoC 与赛灵思 SDK 很多高级系统级性能测量与分析功能相同。它提供用来测量性能和鉴别性能瓶颈的多个可选功能。
- 第 157 页的“专家使用模型”：专家用户可以改写 SDSoC 做出的设计决策。

引言

SDSoC 通过基于 Eclipse 的易用开发环境提供了 C/C++ 编程体验，内含软件工程师和系统架构师进行 Zynq®-7000AP 开发所用的设计工具。SDSoC 包括全系统优化 C/C++ 编译器；该编译器在可编程逻辑中实现自动软件加速，以及提供软件与硬件之间的自动系统连通生成功能。

SDSoC 编程模型面向软件工程师采用了直观设计。应用程序以 C/C++ 代码编写，程序员先确认一个目标平台，并在应用程序中找出需要编译到硬件中的函数子集。然后， SDSoC C/C++ 编译器和链接器将应用程序编译到硬件和软件，以实现 Zynq器件上实施的完整嵌入式系统，包括带固件的完整启动镜像、操作系统和应用程序可执行文件。图 8-1 显示了完整系统流程。

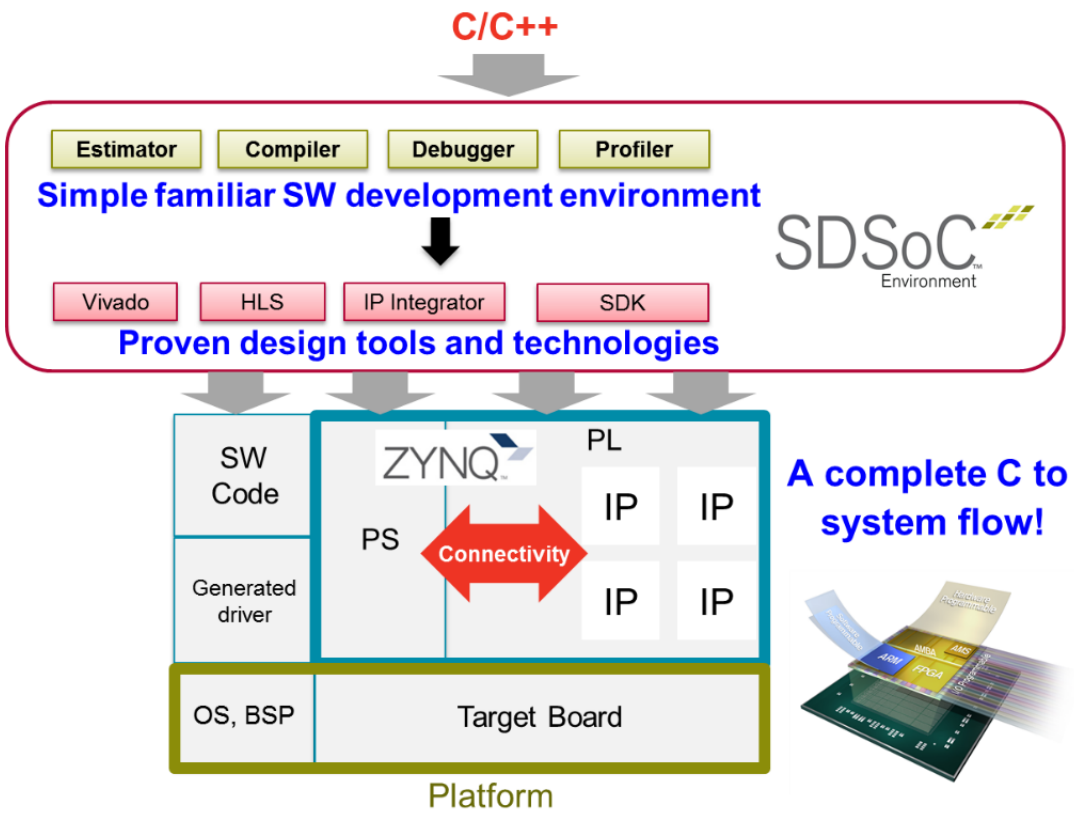

图 8-1 : 通过 SDSoC 实现完整的 C/C++ 应用到 Zynq 系统流程

SDSoC 将本指南中出现的硬件和软件设计流程统一成单个 C/C++ 流程，并能够自动化执行很多耗时的任务，尤其是系统联通生成和硬件-软件集成任务。图 8-2 展示了传统硬件-软件设计流程与使用 SDSoC 的设计流程之间的区别。通过自动执行设计任务， SDSoC 使软件工程师和系统架构师能够迅速实现应用程序并开展快速设计迭代。

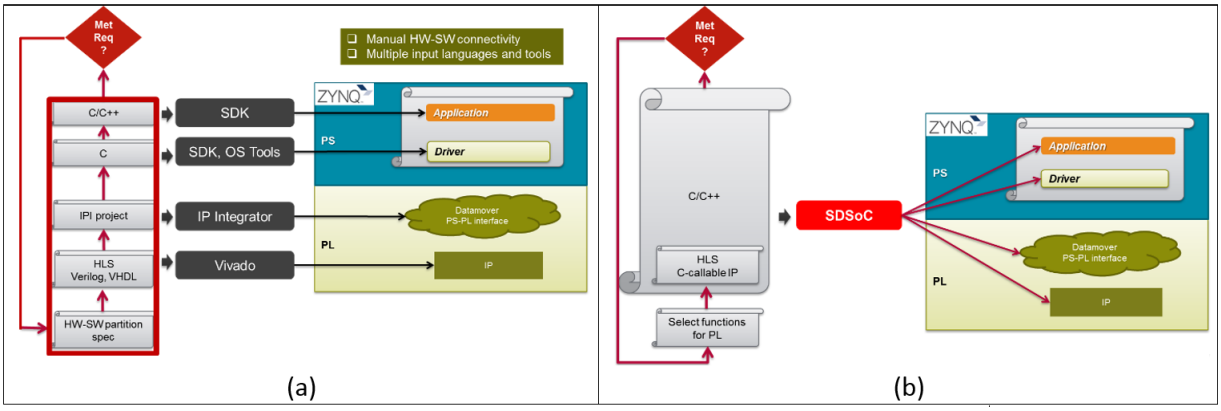

图 8-2 : 传统设计流程 (a) 和使用 SDSoC 的设计流程 (b)

硬件/软件接口的定义取决于为映射到硬件和剩余程序之中的函数之间的交互与数据流。映射到硬件的每个函数以独立线程的形式运行，但是编译器能够确保程序语义得到保留。 SDSoC 编译器并非通过对应用程序重新编码的方式来为每个硬件函数调用设备驱动，而是使用自身高效的数据移动器库自动将原始应用程序映射到所生成的硬件系统，用于在处理系统与可编程逻辑之间进行数据通信。

理解这一映射的最简单方法是认清 SDSoC 编译器可分析程序结构以及数据流，并生成能够高效实现程序数据流的硬件和软件系统，包括开展优化，直接从一个硬件模块向另一个传送数据，不需要总是通过系统主存储器。

应用程序可链接到由 C 语言调用的 IP 库，支持 IP 重用。当应用程序代码链接到作为平台组成部分的 C 调用库时，这些库的接口会将应用程序的平台专用部分封装起来。这样，应用程序就能够实现相当可观的平台间可移植性。

整体使用流程

图 8-3 显示了采用 SDSoC 设计环境的整体使用流程。就特定用户应用而言，第一步是使用剖析或其他方法识别出应用程序中计算强度较大的部分。这些部分应该已经存在或被重构成了函数。在 SDSoC 设计环境中，可将这些函数分配至硬件。 SDSoC 支持快速性能估算，无需生成完整比特流，就能够实现快速设计优化。根据性能速算结果，可面向下列目标实行优化：

- 将更多计算能力迁移到硬件端
- 优化迁移至硬件中的函数性能；例如，使用高层次综合 (HLS) 编译提示
- 优化处理系统与可编程逻辑之间的数据传送性能；例如，使用不同的数据移动器或不同的存储器分配

在实现所需的性能之后， SDSoC 可以生成整个系统的 SD 卡镜像，用以在目标平台上运行应用。通过运行所生成的系统，还可做出进一步优化。

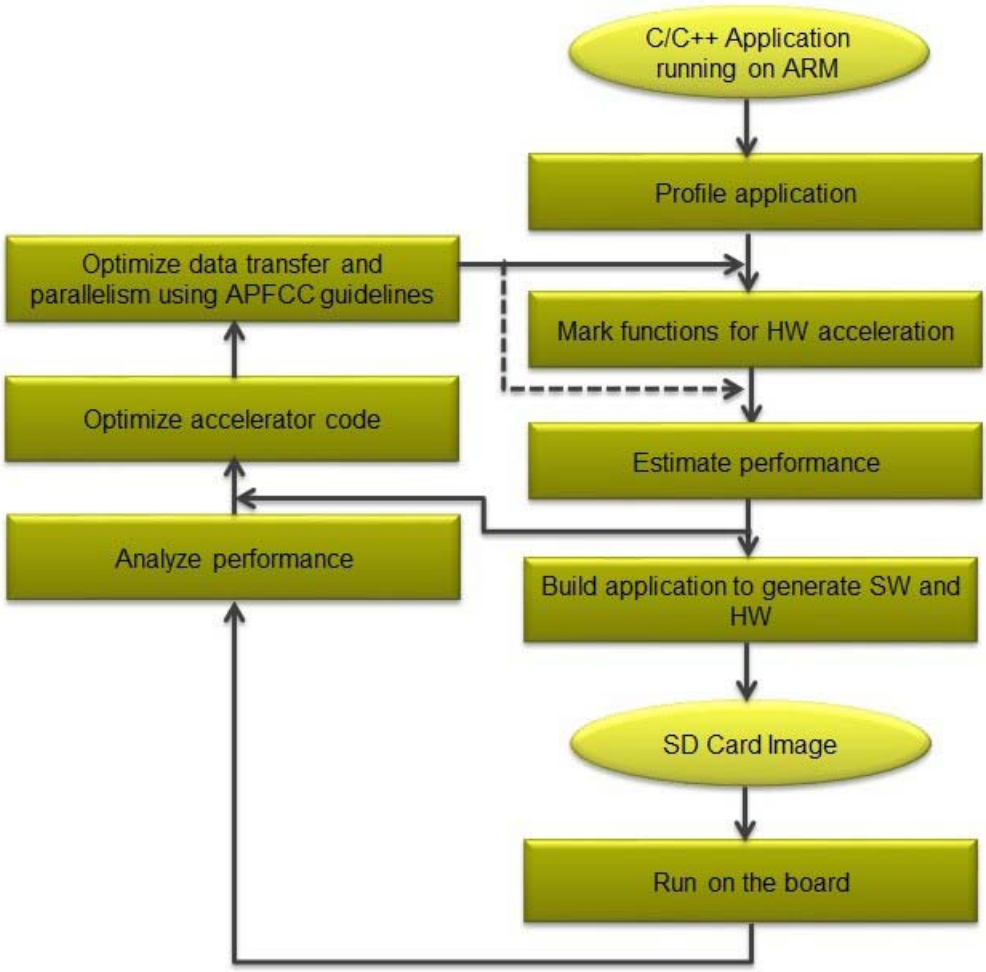

图 8-3 : SDSoC 用户流程

默认情况下，SDSoC 针对的是 Linux 操作系统。所生成的 SD 卡镜像包括 Linux 内核、文件系统和您应用程序。SDSoC 还支持 FreeRTOS 和独立(裸机)模式等 OS。对于 FreeRTOS 和独立模式，用户应用在编译时采用来自赛灵思独立系统 (LibXil) 的支持。如需进一步了解控制 OS 选择的选项，请参阅 《SDSoC 简介》(UG1027) [参照 25] 和 《SDSoC 入门指南》(UG1028) [参照 26]。

SDSoC 原生直接支持数种硬件平台。所支持平台包括 ZC702、 ZC706、 Zed 板和 MicroZed 等。当您设置项目时可在 SDK 中为应用选择目标平台，或者在 Makefile 中通过指定 `-sds-pf` 命令行选择。通过运行命令行工具 sdscc，选择 `-sds-pf-list` 选项，这样您就能够查看所有可用平台的列表。

SDSoC 还能帮助您和第三方以自己的 Vivado 设计开始生成平台。可在 SDK 或命令行中指向顶层平台目录，选择这类平台。如需进一步了解创建 SDSoC 平台，请参阅 《SDSoC 平台和库指南》(UG1146) [参照 27]。

剖析

前几章中已经详细介绍了进行性能分析与剖析的重要性。 SDSoC 通过工具中的插件和特性将这些原理集成体现。剖析功能基于赛灵思 SDK 中的相应功能，因此，应用开发者可以使用非侵入式的 TCF 剖析器来分析应用程序的特性，而且无需进行代码导入。根据剖析信息，可开始向硬件进行第一轮函数转移。更多详情请参阅第 6 章"软件设计流程"。

SDSoC 还支持基于 Eclipse 的标准工具，例如 gprof，用于剖析用户应用；请参阅第 7 章"调试"。SDSoC 用户可以修改应用程序的构建属性，并选择 `-pg` 和 `-g1` 选项获得剖析信息。选项选择完毕后，用户就可以构建项目并在板上执行，获得剖析信息文件 (`gmon.out`)。SDSoC 环境包含 `gmon.out` 文件的基于 Eclipse 的标准图形化视图，用于显示层级化函数图，并包括每种函数所用的时间百分比。

性能估算

在确定了用于进行硬件加速的函数之后， SDSoC 环境能够提供相应工具迅速估算加速措施可实现的性能提升。例如，典型应用程序的整个构建周期可能需要 40 分钟，性能估算流程可以给出以分钟为单位的应用程序加速效果。将构建配置改为 “SDEstimate” 并构建项目，运行性能估算流程。如需进一步了解上述流程，请参阅 《SDSoC 简介》 (UG1027) [参照 25] 以及 《SDSoC 入门指南》 (UG1028) 中的 Lab 4 [参照 26]。

生成并运行完整的软件-硬件系统

一旦您对利用性能估算流程通过迭代得到的估算性能感到满意，就可将构建配置改为 “SDRelease”，并重新构建项目。您还可以将生成的 SD 卡镜像复制到 SD 卡中，并通过板上运行观察所生成系统的实际性能。

使用 C 语言调用 RTL IP 库优化性能

如第 151 页的“引言”中所述，软件应用定义的数据流程可自动映射到特定应用 SoC 中的硬件和软件上。 SDSoC 编译器根据程序数据流程将操作映射和调度到硬件函数上。应用程序还可链接到库，将函数调用映射到用硬件描述语言 （例如 Verilog）定义的 IP 模块上。从应用程序代码和 SDSoC 编译器的角度看，无论函数是在编译时间使用 HLS 进行综合，还是映射到 HDL IP 模块上，调度和映射都是相似的任务。任何对硬件函数的调用都会被 SDSoC 编译器映射到硬件中。

针对IP 模块外设的传统“软件驱动”方案通常只显示控制接口；数据流由硬件系统采集。使用 SDSoC 一类全系统编译器，应用程序代码则可以直接调用数据处理功能，就像调用其他函数库一样。可抽象并自动调用数据传送 （例如使用 DMA），以有效实现函数调用。

如需了解更多信息，请参阅 《SDSoC 平台和库指南》(UG1146) 中的“SDSoC库”章节 [参照 27]。

使用 HLS 优化 IP 性能

在 SDSoC 环境中传送到硬件的应用程序代码通过 Vivado 高层次综合 (HLS) 编译器进行编译。第 5 章"硬件设计流程"中对这部分内容进行了描述。《Vivado Design Suite 用户指南：高层次综合》(UG902) 中提供了几种优化策略 [参照 9]。SDSoC 工具采用了几个常用的代码改善策略，例如添加编译指示以在开展高强度计算循环之前启动流水线化操作，以

及为内循环中访问的变量指定 array_partition 编译指示以便为内循环增加存储器带宽，如需进一步了解优化策略，请参阅 《SDSoC 简介》(UG1027) [参照 25]。

优化系统性能

通过加速来实现系统性能改善是 SDSoC 中集成的重要原理之一。第 2 章"系统级考虑事项"和第 3 章"硬件设计考虑事项"谈到了从软件和硬件角度改善性能的不同方法。本节介绍了 SDSoC 采用的其他一些技术，以及如何实现之前介绍过的几种技术，例如在系统存储器中存储数据以及数据流水线。

使用物理上连续的存储器

SDSoC 系统优化编译器结合了软件和硬件函数中的存储器分配权衡。在基于 Linux 的系统上，对由 `malloc()` 分配的存储器进行分页。内核将为您记录分页信息，并向您应用程序提供统一视图。然而，当通过 `malloc()` 分配的存储器转移到加速器时存在一处性能损失，即需要收集每个页并使用 AXI-DMA 中的分散-聚集模式将它们转移。 SDSoC 提供了可由用户访问的函数 `sds_alloc()`，该函数行为上与 `malloc()` 相似，不同的是它用于分配物理上的连续存储器。因此，在使用分散-聚集 DMA 这一类的数据移动器时能实现更快的性能，但是它也允许 SDSoC 选择 AXI-DMA 的简单模式数据移动器，可利用较低的开销显著提高性能。

使用本地缓存和流数据

如果数据量比较小，传送至通过 Vivado HLS 编译的代码中的数据在计算期间也能传送至可编程逻辑内的 BRAM 中保存。如果数据太大，可编程逻辑的 BRAM 无法容纳，或者无需在计算过程中多次使用，那么可将其作为数据流发送到所生成的加速器中。对于在处理器与加速器之间传送的数据是进行缓存还是流传送，您可自行选择，详见 《SDSoC 简介》(UG1027) [参照 25]。

使用共享存储器

除了可将数据从处理器的存储器传送到加速器的本地存储器，或通过加速器进行流传送以外，处理器和加速器还可以访问共享存储器的公用区域。您可向映射到硬件的函数添加代码编译指示来实现这一目的。使用这种技术时，用户应用程序必须使用 `sds_alloc()` 而不是 `malloc()` 来分配存储器。如需进一步了解，请参阅 《SDSoC 简介》(UG1027) 第10章 [参照 25]。

使用加速器-加速器直联数据传输

如果选用多个函数进行硬件加速， SDSoC 环境可以自动将对它们开展流水线化。这意味着数据能够直接在硬件之间传送，无需先复制到处理器，前提是处理器侧没有被占用。

使用相同硬件函数的多个实例

从单个源位置向映射到硬件中的函数发出的多个调用可以映射到单个加速器中。然而，SDSoC 提供了 `async` 代码编译指示，可指导编译器生成多个加速器实例，以便并行处理更多数据，从而改善应用程序性能。

通过计算将数据通信流水线化

对一个加速器的多个连续调用可在用户编译指示控制下由 SDSoC 编译器进行流水线化。这能够促成加速器计算与加速器后续调用的数据传送交覆。建议参照第 156 页的图 8-4，其中给出了用两个输入和一个输出进行加速器的顺序调用。

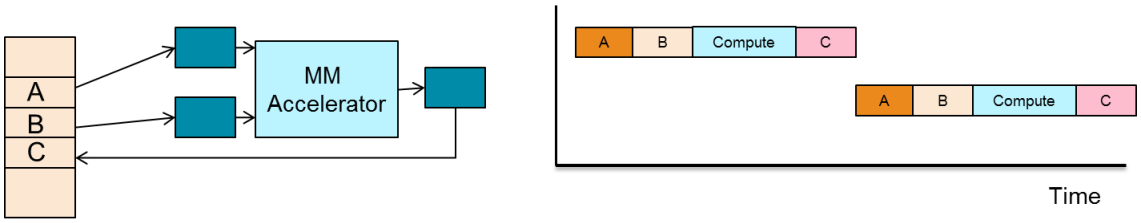

图 8-4 : 顺序执行加速器调用

图 8-5 显示了如何以流水线方式执行相同加速器的两个调用。编译器通过自动插入的多个缓存，确保第一个调用的数据传送刚一完成，第二个调用的数据传送就可以立即着开始。可在 SDSoC 环境中使用 `async` 编译指示来实现这上述目标，如 《SDSoC 简介》(UG1027) 第7章所述 [参照 25]。

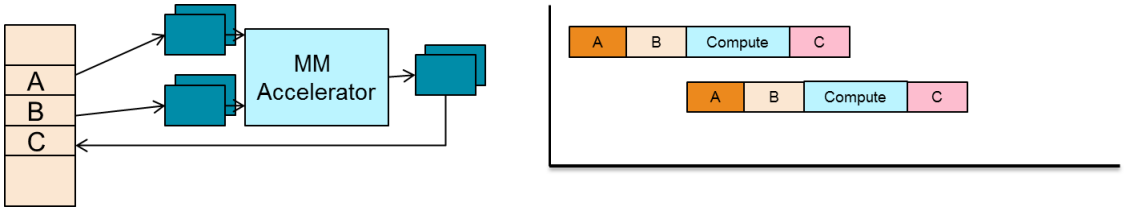

图 8-5 : 加速器调用的流水线执行

调试系统

SDSoC 与赛灵思 SDK 系统调试器共享调试基础设施和方法，包含支持独立平台调试应用程序的强化特性，以及查看物理地址空间中 IP 寄存器的功能。请参阅第 6 章"软件设计流程"。

SDSoC 允许使用 SDSoC IDE 创建和调试项目。也可使用用户创建的 Makefile 在 SDSoC IDE 外部创建项目，并通过命令行或使用 SDSoC IDE 来调试项目。.关于调试的详细信息请参阅 《SDSoC 简介》(UG1027) [参照 25] 和 《SDSoC 入门指南》(UG1028) [参照 26]。

性能测量与分析

为简化性能测量并鉴别性能瓶颈， SDSoC 提供了 `sds_clock_counter()` 函数，用以从硬件中的自由运行计数器中返回周期精确的时间戳。您可以采用这个计数器来计量代码并收集性能数据。还可使用构建目录中 `_sds` 文件夹内的 Vivado_HLS 报告，详细了解所生成的硬件，以及为改善性能可以做哪些更改。

SDSoC 与赛灵思 SDK 共享很多高级系统级性能测量与分析功能，请参阅第 6 章"软件设计流程"。SDSoC 编译器还可以向所生成的系统中插入 AXI 性能监测器 (APM)，用以监测 Zynq 器件可编程逻辑 (PL) 与处理系统 (PS) 之间的流量。通过使用 APM 观察 PS/PL 边界上的活动，设计人员可以判断系统是否正在使特定资源的容量达到饱和，例如缓存一致 ACP 端口。

计算过程中长期观察怠机数据传输接口的硬件函数或能识别出某个用于运算的系统可通过加速器微架构优化来提升吞吐量并减少时延。反之，饱和的数据传输接口意味着应该将流量导向 PS 上的其他传输接口。详情请参阅 《赛灵思软件开发套件 (SDK) 用户指南：系统性能分析》(UG1145) [参照 24]。

专家使用模型

您可按需改写 SDSoC 所做的每个设计决策，既可以向应用代码中插入编译指示，也可通过用 IDE/命令行参数来实现。

针对高级用户的控制功能包括：

- 数据移动器选择：控制数据传输向不同 IP 模块的映射，例如简单模式 DMA、分散-聚集 DMA 以及 2D DMA。
- Zynq-7000AP SoC 处理系统到 CPU 和 DDR 的接口选择：控制到缓存一致或高吞吐量 AXI 接口的映射。
- 硬件函数"端口"接口选择：控制硬件加速器上的硬件接口；例如，硬件函数与控制软件线程之前的"共享存储器"，或者本地 BRAM 中缓存数据的"复制入，复制出"任务。
- 具有各种运行时间的数据传输：对于遵循"复制入，复制出"语义的传输，应尽可能限制传输大小，从而提高系统性能。
- 使用 Vivado HLS 的硬件函数微架构：对于计算密集系统，可采用 Vivado HLS 编译指示和指令来改善吞吐量和时延。
- Blocking vs. non-blocking 函数调用：默认情况下，函数会保留初始的函数调用语义，但也可在代码中使用编译指示让编译器立即返回控制，例如实现软件任务流水线化。
- 将硬件函数子集映射到应用程序在运行时进行时分复用的不同比特流：可编程逻辑架构进行时分复用，以实现更多硬件函数——超出目标器件中可同时处理的函数量。

如需了解上述控制详情，请参阅 《SDSoC 简介》(UG1027) 中的"SDSoC 编译指示规范"部分 [参照 25]。

高级用户、硬件设计师和系统软件团队可能需要构建自己的 SDSoC 平台，包括部分硬件设计和软件组件——例如操作系统、驱动、文件系统和函数库。他们还能够将自己的 HDL IP 打包成 C 调用 IP 库。有一种完善而且低开销的 Vivado 硬件系统导出方法，用以支持 SDSoC，以及创建针对 HDL IP 的 C 调用库。如需了解详情，请参阅 《SDSoC 平台和函数库指南》(UG1146) [参照 27]。

SDSoC 将前几章介绍的多种技术结合在一个界面下，帮助您高效组建、设计和调试系统。它可用来实现以前介绍的方法原理，并在同一界面下着重显示新原理。该工具让软件设计师受益最大。这是软件设计师的必备工具，可用于加快具有硬件组件的系统开发速度。

附加资源与法律提示

赛灵思资源

如需了解问答、技术文档、下载以及论坛等支持性资源，敬请访问：赛灵思技术支持。

解决方案中心

如需了解设计周期各阶段有关器件、软件工具和 IP 等的技术支持，请参阅：赛灵思解决方案中心。相关专题包括设计辅助、建议和故障排除提示等。

Xilinx Documentation Navigator

赛灵思 Documentation Navigator 是一个免费工具，能够在您使用赛灵思产品时帮助您访问有关文档。Documentation Navigator 属于 Vivado 安装程序的一部分。当在您系统上完成安装后，您可以通过 “Start > Programs > Xilinx Design Tools > DocNav” 然后点击 “DocNav” 图标来访问。

有关赛灵思 Documentation Navigator 更多详情，请参阅 《Vivado Design Suite 用户指南：着手设计》(UG910) [参照 11] 中的链接

相关设计中心

通过设计中心可以迅速访问文档、培训、和特定设计任务的有关信息。下列设计中心可用于本指南中所描述的嵌入式开发与方法：

- 部分重配置设计中心
- 软件开发套件 (SDK) 设计中心

参考资料

下列参考资料提供了更多相关信息。

赛灵思用户指南与参考指南

1. 7 系列 FPGA 时钟资源 (UG472)
2. 7 系列 FPGA GTX/GTH 收发器 (UG476)
3. 7 系列 FPGA GTP 收发器 (UG482)
4. 《Zynq-7000 All Programmable SoC 技术参考手册》(UG585)
5. 《OS 与库文件集》(UG643)
6. 《赛灵思软件开发套件 (SDK) 帮助》(UG782)
7. 《Zynq-7000 All Programmable SoC 开发人员指南》(UG821)
8. 《Zynq-7000 XC7X020 All Programmable SoC ZC702 评估板用户指南》(UG850)
9. 《Vivado Design Suite 用户指南：高层次综合》(UG902)
10. 《Vivado Design Suite 用户指南：功耗分析和优化》(UG907)
11. 《Vivado Design Suite 用户指南：着手设计》(UG910)
12. 《基于 AXI 接口的 KC705 嵌入套装 MicroBlaze 处理器子系统软件教程》(UG915)
13. 《Zynq-7000 All Programmable SoC ZC702 Base 目标参考设计 (Vivado Design Suite 2014.2) 用户指南》(UG925)
14. 《Zynq-7000 All Programmable SoC PCB 设计指南》(UG933)
15. 《Vivado Design Suite 教程：嵌入式处理器硬件设计》(UG940)
16. 《UltraFAST 设计方法指南 （适用于 Vivado Design Suite）》(UG949)
17. 《PetaLinux 工具用户指南：Zynq All Programmable SoC Linux-FreeRTOS AMP 指南》(UG978)
18. 《Vivado Design Suite 用户指南：采用 IP 集成器设计 IP 子系统》(UG994)
19. 《采用赛灵思 Zynq-7000 All Programmable SoC 编程 ARM TrustZone 架构用户指南》(UG1019)
20. 《Zynq-7000 All Programmable SoC 安全启动入门指南》(UG1025)
21. 《嵌入式系统工具参考手册》(UG1043)
22. 《Vivado Design Suite 用户指南：创建和封装定制 IP》(UG1118)
23. 《基本软件平台生成参考指南》(UG1138)
24. 《赛灵思软件开发套件 (SDK) 用户指南：系统性能分析》(UG1145)

SDSoC 技术文档

下列 SDSoC 文档目前访问受限。如需申请访问这些文件，请联系赛灵思销售代表。

25. SDSoC 环境：用户指南 (UG1027)
26. SDSoC 环境：着手设计指南 (UG1028)
27. SDSoC 环境：平台和库指南 (UG1146)

其他赛灵思参考材料

数据手册

28. Zynq-7000 All Programmable SoC (Z-7010、 Z-7015、和 Z-7020): DC 和 AC 开关特性 (DS187)
29. Zynq-7000 All Programmable SoC 简介 (DS190)
30. 《Zynq-7000 总线功能模型数据手册 v2.0》(DS897)

产品指南

31. 《AXI BFM 内核 LogiCORE IP 产品指南》(PG129)

白皮书

32. 灵活运用复位：《局部思维超越全局思维白皮书》(WP272)
33. 《Zynq-7000 All Programmable SoC 安全启动》(WP426)

应用指南

34. 《采用 Zynq-7000 All Programmable SoC 设计高性能视频系统》(XAPP792)
35. 《轻量化 IP (lwIP) 应用示例 v4.0》(XAPP1026)
36. 《Zynq-7000 AP SoC 两款处理器上简单 AMP 运行 Linux 与裸机》(XAPP1078)
37. 《简单 AMP：两款 Cortex-A9 处理器运行裸机系统》(XAPP1079)
38. 《Zynq-7000 AP SoC 中 PS 与 PL 以太网性能和 PL 以太网的巨型帧支持》(XAPP1082)
39. 《针对赛灵思数模转换器 (XADC) 专用接口采用 Zynq-7000 处理系统 (PS) 实现系统监控与外部通道测量》(XAPP1172)
40. 《Zynq-7000 All Programmable SoC 安全启动》(XAPP1175)
41. 《使用 Zynq-7000 AP SoC 处理系统与 XADC AXI 接口开展系统监控》(XAPP1182)
42. 《通过 ARM DS-5 工具链开展 Zynq-7000 平台软件开发》(XAPP1185)

网页

43. AXI Performance Monitor 网页
44. AXI Timer/Counter 网页
45. LogiCORE™ AXI Traffic Generator 网页
46. Xilinx Security Solutions 网页
47. Xilinx Zynq-7000 AP SoC Ecosystem 网页
48. 赛灵思 PetaLinux 工具网页

维基页

49. K7 Embedded TRD 2013.2 维基页
50. 赛灵思 Linux 驱动维基页
51. 赛灵思 Linux GPIO 驱动维基页
52. 赛灵思 Linux I2C 驱动维基页
53. 赛灵思 Linux SPI 驱动维基页
54. 赛灵思 Linux 维基页
55. 赛灵思 Multi-OS 支持维基页
56. 赛灵思 PetaLinux 维基页
57. 赛灵思 U-Boot 维基页
58. 赛灵思 Yocto 维基页
59. Zynq-7000 AP SoC 启动 - 不通过外部存储器的启动与运行技术提示维基页
60. Zynq-7000 AP SoC 启动 - L2 缓存外闭锁与执行技术提示维基页

61. Zynq-7000 AP SoC 低功耗技巧第一部 - 功耗演示安装与运行技术提示维基页
62. 赛灵思 Zynq-7000 AP SoC 频谱分析器第二部 - ARM NEON 库构建技术提示
63. 赛灵思 Zynq Linux 维基页
64. 赛灵思 Zynq 功耗管理维基页

答复记录

65. 赛灵思问答记录 46778
66. 赛灵思问答记录 47511
67. 赛灵思问答记录 50991
68. 赛灵思问答记录 52847
69. 赛灵思问答记录 55572
70. 赛灵思问答记录 57744
71. 赛灵思问答记录 58387

来自其他公司的信息

72. ARM DS-5 Development Studio 性能优化分析器文档：
http://ds.arm.com/ds-5/optimize
73. ARM Infocenter 加速器端口一致性网页：
http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0407e/CACGGBCF.html
74. 《ARM 安全技术》：《采用 TrustZone® 技术构建安全系统》：
http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf
75. 《Cortex-A9 技术参考手册》：
http://infocenter.arm.com/help/topic/com.arm.doc.ddi0388e/DDI0388E_cortex_a9_r2p0_trm.pdf
76. 稳健型 I2C 通信计算设计：
http://www.edn.com/design/analog/4371297/Design-calculations-for-robust-I2C-communications
77. Discretix：http://www.discretix.com/
78. ENEA 软件 AB 解决方案：http://www.enea.com/solutions/
79. eSOL 嵌入式工程与使能方案：http://www.esol.com/
80. Express Logic ThreadX RTOS: http://rtos.com/products/threadx/xilinx_arm
81. GreenHills INTEGRITY RTOS: http://www.ghs.com/products/xilinx_zynq.html
82. 《MicroBlaze 如何能够与 Zynq-7000 AP SoC 和平共处》: http://www.eetimes.com/document.asp?doc_id=1280680
83. IAR Integrated Solutions 合作伙伴计划：http://www.iar.com/Products/RTOS/Integrated-RTOSes/
84. iVeia Adeneo 嵌入式板支持套装：http://www.adeneo-embedded.com/Products/Board-Support-Packages
85. Mentor Graphics Nucleus RTOS: http://www.mentor.com/embedded-software/nucleus/?sfm=auto_suggest
86. Micrium µC/OS RTOS：http://micrium.com/products/
87. MontaVista Carrier Grade Edition 7：http://www.mvista.com/product-carrier-grade-edition7.html
88. PCA9548A 低电压可复位 8 通道 I2C 开关：http://www.ti.com/lit/ds/symlink/pca9548a.pdf
89. SD Association, 最新简化规格: https://www.sdcard.org/downloads/pls/simplified_specs/
90. Open Kernel Labs OKL4 Microviso：http://www.ok-labs.com/products/okl4-microvisor
91. QNX Neutrino RTOS：http://www.qnx.com/products/neutrino-rtos/neutrino-rtos.html#overview|
92. Quadros RTOS：http://www.quadros.com/products/operating-systems

93. Real Time Engineers FreeRTOS 互动网页：http://www.freertos.org/Interactive_Frames/Open_Frames.html?http://interactive.freertos.org/entries/23277857-Updated-Xilinx-FreeRTOS-port-for-Zynq-to-SDK-14-4-release
94. Sciopta RTOS：http://www.sciopta.com/news/ZYNQ-7000.html
95. Sierraware Hypervisor 与 SierraTEE 受新任执行环境：http://www.sierraware.com/arm_hypervisor.html
96. 用于 ZC702 的 SYSGO PikeOS BSPs、 Zed、以及 Zynq-7000 All Programmable SoC BSP：http://www.sysgo.com/products/board-support-packages/pikeos-bsp-list/
97. SYSGO PikeOS 产品信息：http://www.sysgo.com/news-events/press/press/details/article/sysgos-certified-pikeos-supports-xilinxs-zynq-7000-all-programmable-soc/
98. 赛灵思官方 Linux 内核网页：https://github.com/Xilinx/linux-xlnx
99. Timesys LinuxLink: http://www.timesys.com/embedded-linux/linuxlink
100. Wind River VxWorks、 Linux、和 Workbench IDE：http://www.windriver.com/
101. 赛灵思 XC702 DS-5 连接指南：http://www.arm.com/files/pdf/zc702_ds5_2.pdf
102. Yocto Project 网页：http://git.yoctoproject.org/cgit/cgit.cgi/meta-xilinx/about/

培训资料

赛灵思提供多种多样的培训课程和 QuickTake 视频，可帮助用户进一步了解有关本文档中提出的概念。使用以下链接获取相关视频：

1. 如何使用 Xilinx SDK 创建 Zynq 启动图像
2. Vivado Design Suite QuickTake 视频教程

请阅读：重要法律提示