Xcell 软件刊 Journal

为全可编程世界
量身打造的
软件解决方案

01 期
2015年，秋季刊

C/C++、OpenCL 编程中的下个逻辑步骤

探索 SDSoC 环境：
构建一个设计样例

如何在 Zynq SoC 上进行
系统级软硬件优化

面向 FPGA 的 SDAccel 软件应用
设计流程

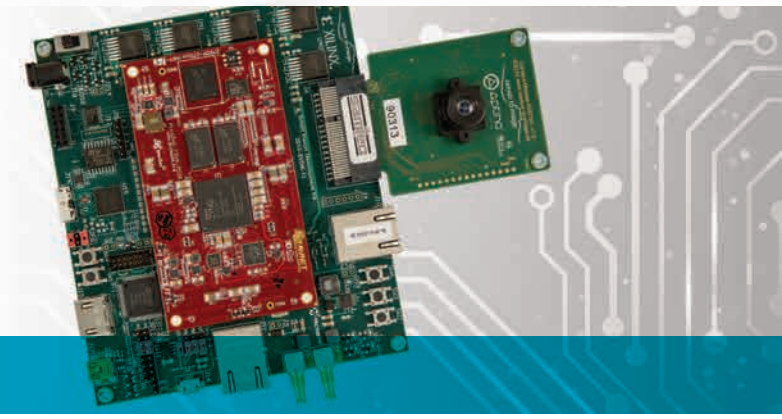
MathWorks：在桌面而不是实验室进行
设计权衡

XILINX
ALL PROGRAMMABLE.

china.xilinx.com/xcell

安富利产品简介

智能视觉开发套件



Electronics Marketing　Technology

picoZed™ Compatible

sensor to image

智能视觉开发套件采用 PicoZed 模块化系统 (SOM)，并提供适合机器视觉应用的载卡。该套件为工程师进行嵌入式视觉应用开发提供所需的各种必要组件。

智能视觉开发工具套件与德国智能和机器视觉系统专家 Sensor to Image 公司合作开发。它是一种低成本的全功能套件，包括 5 分钟即可上手的指南文件，并提供业界标准的视觉连接接口，包括 GigEVision、USB3Vision 和 CoaXPress 等。此套件包括一个 Aptina 120 万像素摄像头模块，此外还提供并支持业界领先的软件工具，能加速开发周期。

特性

- PicoZed 7015 SOM
- 机器视觉载卡，包括:
  - GigEVision
  - CoaXPress
  - USB3Vision
  - HDMI 输出
- Aptina 120 万像素摄像头模块

目标应用

- 机器视觉
- 视频分析
- 制造产品
  - 产品检验
  - 机器人
  - 质保
- 其它应用
  - 智能交通监控
  - 条码扫描
  - 赝品检测
  - 运动模拟

套件包括

- PicoZed (Zynq–7015) SOM 模块
- 机器视觉载卡
- Aptina 120 万像素摄像头模块
- 通用电源适配器
- Mini–USB–B 转 USB–A 电缆
- Vivado Design Edition — 器件锁定版
- 可供下载的参考文档
- 可供下载的参考设计/评估
  - Sensor to Image GigEVision (GEV)
  - Sensor to Image USB3Vision (U3V)
  - Sensor to Image CoaXpress (CXP)
  - Embedded Visual Applets — 演示版本
  - HALCONtrailrun — 演示版本
- 以太网电缆

订购

型号	硬件描述	零售价
AES–Z7PZ–SVDK–G	Smart Vision Development Kit	$595.00

联系我们

北京: 010 8414 8118　西安: 029 8835 6518　南京: 025 8483 8138　青岛: 532 8097 0718　福州: 591 8771 0381
重庆: 023 6879 7512　武汉: 027 8732 2806　广州: 020 2808 7301　杭州: 571 8580 0912　香港: 852 2212 7848
上海: 021 3367 8387　沈阳: 189 0400 8421　成都: 028 8652 8262　深圳: 755 2658 4925　苏州: 512 6956 7753

Copyright © 2015 Avnet, Inc. AVNET and the AV logo are registered trademarks of Avnet, Inc. All other brands are the property of their respective owners.　LIT # PB–AES–Z7PZ–SVDK–G–V2

想获取更多信息，请访问 picozed.org

Accelerating Your Success™

Xcell 软件刊 journal

发行人 Mike Santarini
mike.santarini@xilinx.com
1-408-626-5981

编辑 Diana Scheben

艺术总监 Scott Blair

设计/制作 Teie, Gelwicks & Assoc.
1-408-842-2627

广告销售 Judy Gelwicks
1-408-842-2627
xcelladsales@aol.com

国际 Melissa Zhang,
Asia Pacific
melissa.zhang@xilinx.com

Christelle Moraga,
Europe/Middle East/Africa
christelle.moraga@xilinx.com

Tomoko Suto,
Japan
tomoko@xilinx.com

过往期刊订购 1-408-842-2627

编辑顾问 Tomas Evensen
Lawrence Getman
Mark Jensen

Xilinx, Inc.
2100 Logic Drive
San Jose, CA 95124-3400
Phone: 408-559-7778
FAX: 408-879-4780
www.xilinx.com/xcell/

© 2015 Xilinx, Inc. All rights reserved. XILINX, the Xilinx Logo, and other designated brands included herein are trademarks of Xilinx, Inc. All other trademarks are the property of their respective owners.

The articles, information, and other materials included in this issue are provided solely for the convenience of our readers. Xilinx makes no warranties, express, implied, statutory, or otherwise, and accepts no liability with respect to any such articles, information, or other materials or their use, and any use thereof is solely at the risk of the user. Any person or entity using such information in any way releases and waives any claim it might have against Xilinx for any loss, damage, or expense caused thereby.

发行人致信

欢迎订阅《赛灵思中国通讯软件刊》

赛灵思今年早些时候发布了其 SDx™ 系列开发环境，帮助非 FPGA 专家用 C/C++ 和 OpenCL™ 对赛灵思器件进行逻辑编程。SDx 环境旨在让软件开发人员和嵌入式系统架构师能够像编程 GPU 或 CPU 一样轻松对我们的器件进行编程。赛灵思的 FPGA 硬件技术一直能够加速算法运行，不过直到最近，我们可用底层软件技术和硬件平台共同打造开发环境，满足 C/C++ 和 OpenCL 用户更广泛的软件和系统架构需求。

进入新千年，硬件平台发展迅速。在 21 世纪初，半导体产业改变了软件开发人员的工作格局。为避免今后芯片达到太阳的能量密度，MPU 厂商从单芯片 MPU 转向同构多核分布式处理架构。这种转变推动半导体产业继续按照摩尔定律不断推出新一代产品，甚至推动异构多核处理系统的创新，如我们今天所熟知的片上系统 (SoC)。不过，转向多核给软件开发人员设计能够在这些新型分布式处理架构上有效运行的软件带来了巨大挑战。为此，赛灵思推出其 SDx 系列开发环境，帮助软件开发人员简化设计工作。这种环境让开发人员能够大幅提升其 C/C++ 和 OpenCL 代码在采用新一代处理架构的系统上的运行速度。目前在 FPGA 技术的支持下，运行速度日益提升。

事实上，FPGA 加速处理架构与 MPU 和 FPGA 相结合正在数据中心及其它高计算强度市场领域中快速取代高功耗 CPU/GPU 架构。同样，在嵌入式系统领域，赛灵思的 Zynq®-7000 All Programmable SoC 和即将推出的赛灵思 UltraScale+™ MPSoC 等新型异构多核处理器能够在单芯片上无缝集成多个处理器和 FPGA 逻辑，帮助企业打造出具有无与伦比的高性能和差异化的新一代系统。FPGA 此前一直是硬件工程师的专长，现在则不然。

现在，赛灵思推出了用于硬件平台的 SDx 系列开发环境，这样，软件界能使嵌入式软件开发人员和嵌入式系统开发人员在其所熟悉的环境中用 C/C++ 或 OpenCL 充分发挥 FPGA 的加速功能。这种高层次综合 (HLS) 工具流的强大底层编译技术与编程语言和异构架构工具完美融合在一起，能为软件设计人员和系统设计人员填补最后的空白，帮助他们在自己的异构 SoC 中打造定制硬件加速器。

《赛灵斯中国通讯软件刊》专门帮助您利用 SDx 开发环境以及美国国家仪器公司 (NI) 和 MathWorks® 等赛灵思联盟成员提供的开发环境。本季刊内容丰富，重点涵盖软件趋势、成功案例、技巧教程、最新动态以及未来展望，旨在满足快速增长的用户群需求。我相信，通过阅读本季刊文章，您会深受启发，进一步探索赛灵思资源，并通过赛灵思软件开发者专区搜索可用的 SDx 开发环境。我建议您阅读赛灵思午后加油站专栏 (Xcell Daily Blog)，特别是 Adam Taylor 的 SDSoC 开发环境使用日志。同时，我也诚邀您为新刊物投稿，与同行们分享您在 FPGA 加速系统编程尖端领域的精彩体验。

— Mike Santarini
发行人

又一名蓝调贝斯手魂归天国...
谨以此纪念已逝分析专家和 ESL 前驱 Gary Smith，1941 – 2015。

目录

2015 年，秋季刊
01 期

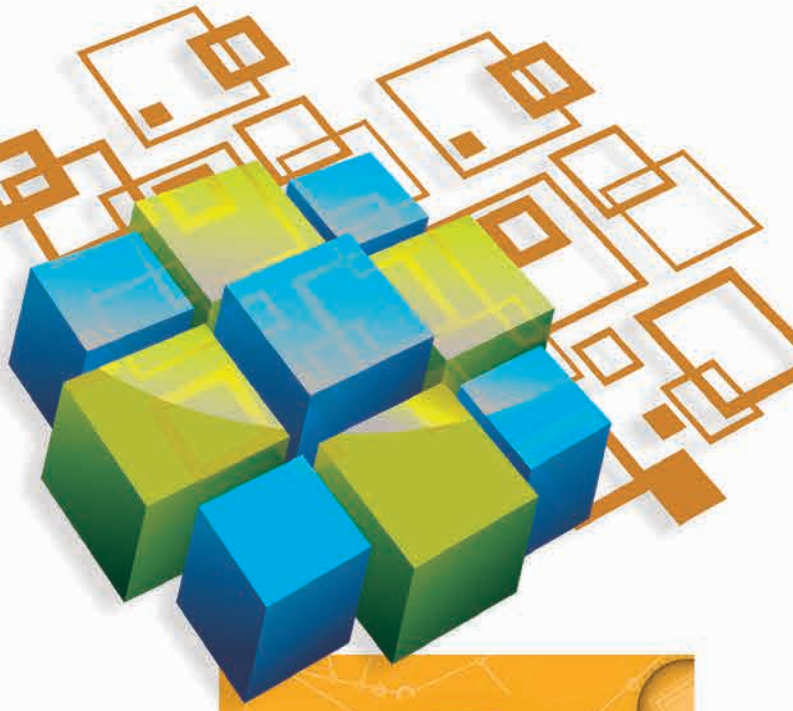
观点

发行人致信
欢迎订阅《Xcell 软件刊》 . . . 3

封面专题

C/C++、OpenCL 编程中的
下个逻辑步骤

6



SDSOC 在嵌入式开发方面的卓越表现

SDSoC手把手：逐步构建设计样例 . . . 14

如何使用 SDSoC IDE 在 Zynq SoC 上进行系统级软硬件优化 . . . 20

SDACCEL 在应用加速领域方面的卓越表现

编译、调试与优化 . . . 30

使用 C++ 库开发 OpenCL 图像应用 . . . 36

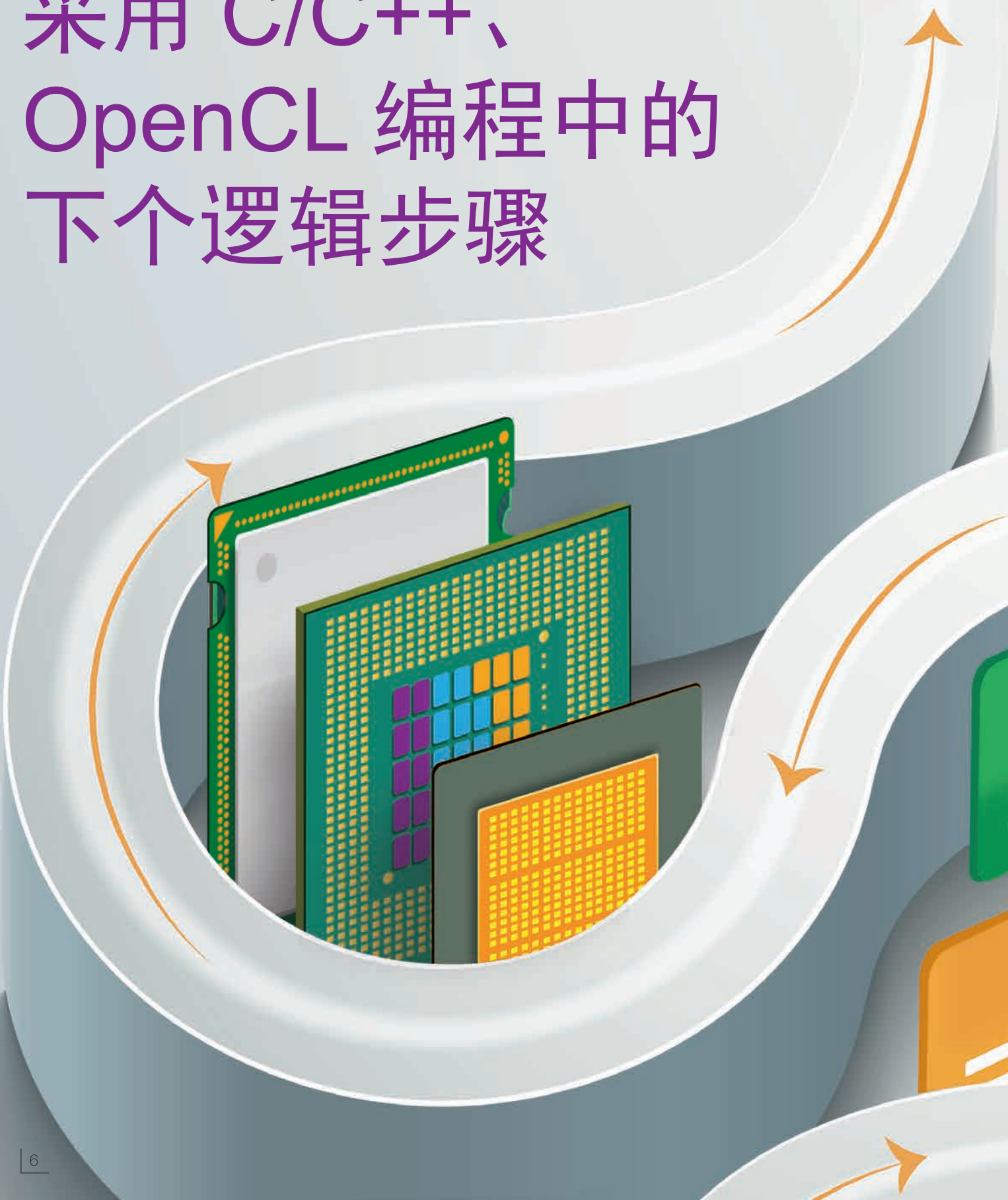
联盟成员优异表现

MATLAB 和 Simulink 协助 Zynq SoC 软硬件协同设计 . . . 42

扩展信息

开发者资源： IDE 更新以及更多资源 . . . 50

采用 C/C++、OpenCL 编程中的下个逻辑步骤



新型环境让您能够将代码性能最大化。

作者：Mike Santarini
赛灵思公司 Xcell 出版物发行人
mike.santarini@xilinx.com

Lawrence Getman
赛灵思公司企业战略与市场营销副总裁
larryg@xilinx.com

自从赛灵思在上世纪 80 年代初期开发和推出世界首款 FPGA 后，这些用途丰富的可编程逻辑器件就成为硬件工程师的 MacGyver 万能工具。赛灵思最近发布了 SDx™ 系列开发环境（即 SDAccel™、SDSoC™ 和 SDNet™），使软件开发人员和系统工程师（非 FPGA 设计人员）能够利用赛灵思器件轻松创建自己的定制化软件定义硬件，从而助力更多创意头脑实现非凡的创新技术。

介绍这些新型环境以及赛灵思及其联盟成员提供的其他软件开发资源之前，我们先来思量一下处理架构的演变及其对软件开发的影响。

这是软件问题…

2000 年以前，典型的微处理器主要由一个带有板载内存的巨型单片处理器内核和一些其他零部件组成，使 MPU 成为一种用以开发新一代应用的相对简单直观

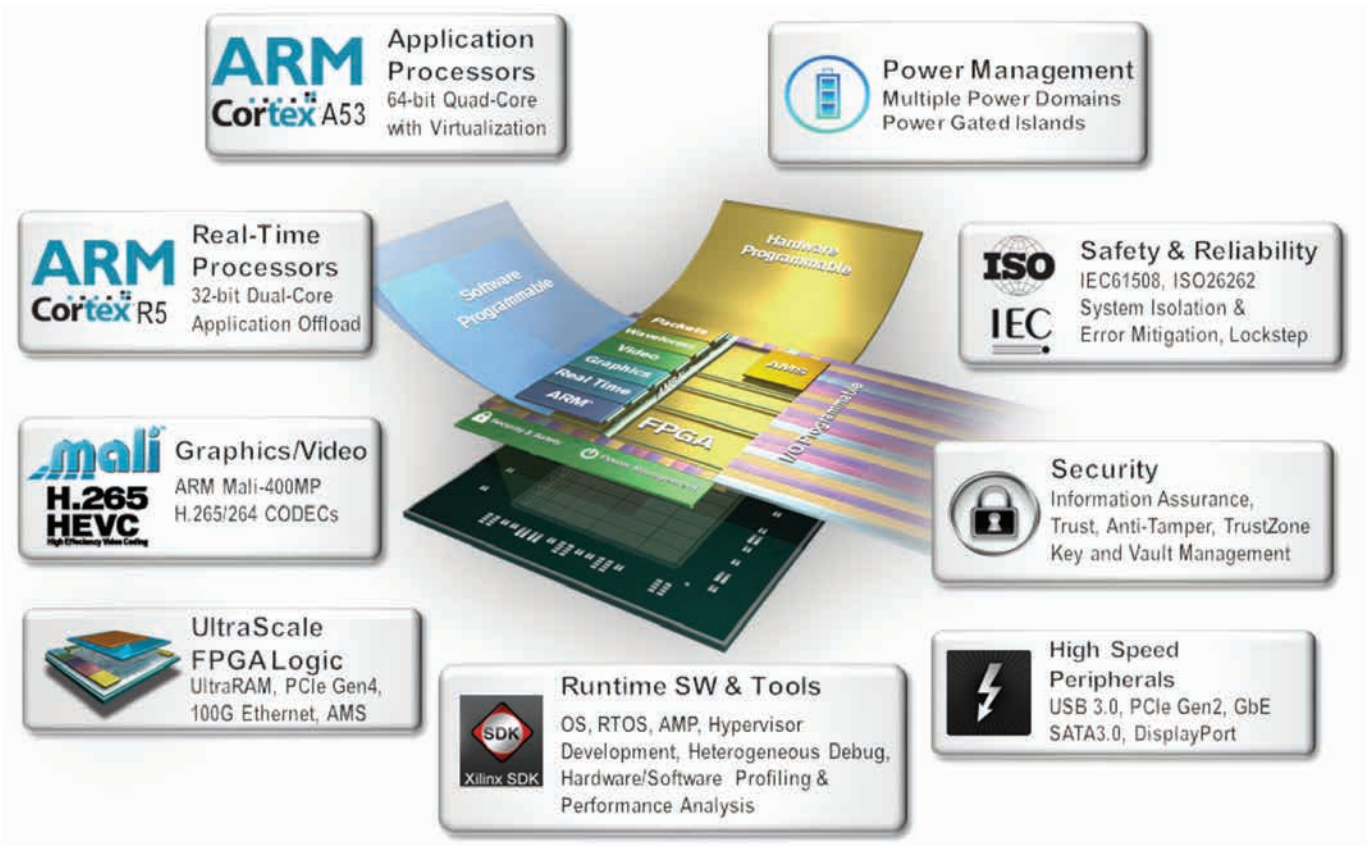


图1 - Zynq UltraScale+ MPSoC

的平台。在那之前的三十年里，微处理器厂商每隔 22 个月就会以摩尔定律的增长速度推出具有更高容量和性能的器件。他们会简单地以增加时钟速率的方式来提高性能。当时速度最快的单片 MPU 是英特尔奔腾 4 处理器 (Pentium 4 Pro)，其极限速度刚刚超过 4GHz。这种演进对于开发人员来说非常重要，因为随着每代产品的推出，他们的程序会变得日趋复杂，能执行更细化的功能，而且程序运行速度更快。

但在 21 世纪初期，半导体行业改变了游戏规则，迫使开发人员根据一套新的规则进行调整。这种转变起始于人们意识到了如果 MPU 产业继续在新的单片 MPU 架构中提高时钟速率，那么鉴于芯片工艺技术的发展路线图以及日趋恶化的晶体管漏电流问题，MPU 将很快达到与太阳相同的功率密度。

正因为如此，MPU 产业快速向同质多处理架构转型；在这种同质多处理架构中，计算任务被分配给多个以较低时钟速率运行的小型内核。新的处理模型让 MPU 和半导体厂商能够继续生产新一代容量更高的器件，并将更多功能集成到单个芯片中以获得更高性能。然而，现有程序无法利用新型分布式架构，因此软件开发人员必须想办法开发出能够跨多个处理器内核高效运行的程序。

同时，后来的几代芯片工艺技术继续使晶体管数量成倍增长，让半导体公司能够采取另一个创新

SDAccel 环境包括一个快速的架构优化编译器，能高效利用片上 FPGA 资源。

举措，即将不同类型的内核集成到同一颗芯片中，创建出 SoC。这些异构多处理器架构给嵌入式软件开发人员提出了更多挑战，使软件开发人员不得不开发定制软件协议栈，让应用能够在目标系统上实现最佳运行。

如今，半导体行业又一次改变游戏规则，但这次软件开发人员则对转变持欢迎态度。面对另一个功耗窘境，半导体和系统公司正在将目光转向由 FPGA 加速的异构处理架构。这种架构将 MPU 与 FPGA 紧密配合在一起，能以最小的功耗成本提高系统性能。这种新兴架构最显著的应用是用于新型数据中心处理架构中。在一篇现在著名的论文中，微软研究人员展示了在架构上将 MPU 与 FPGA 组合在一起能实现 90% 的性能提升，同时功耗仅仅增加 10%，在单位功耗性能方面远远优于 MPU 与高功耗 GPU 相结合而实现的架构。

通过 FPGA 加速的异构多处理架构的优势已经超出了数据中心应用范畴。赛灵思 Zynq®-7000 All Programmable SoC 器件在单芯片上完美集成了 ARM 处理器和可编程逻辑，让采用该器件的嵌入式系统受益良多。使用即将推出的 Zynq UltraScale+™ MPSoC 的系统注定将会更加出色。Zynq UltraScale+ MPSoC 在单个器件中集成了多个 ARM® 内核（4 个 Cortex™-A53 应用处理器、2 个 Cortex-R5 实时处理器和 1 个 Mali™-400MP GPU）、可编程逻辑、多级安全、更高安全性以及高级电源管理模块（如图 1 所示）。

不过，要让这些 FPGA 加速的异构架构适合大规模部署并方便软件开发人员使用，FPGA 厂商必须开发新的环境。在这方面，赛灵思提供了三个开发平台：针对数据中心开发人员的 SDAccel、针对嵌入式系统开发人员的 SDSoC，以及针对网络线卡架构师和开发人员的 SDNet。这些赛灵思新环境让开发人员能够轻松将代码的缓慢部分放在可编程逻辑上编程以加速程序，从而创建出最佳系统。

面向 OpenCL、C/C++ 的 SDAccel 通过编程实现 FPGA 加速处理

最新的赛灵思 SDAccel 开发环境为数据中心应用开发人员提供一套基于 FPGA 的完整软硬件解决方案（图 2）。SDAccel 环境包含一个能有效利用片上 FPGA 资源的快速的架构优化编译器。该环境为开发人员提供类似于 CPU/GPU 的熟悉的工作环境和软件开发流程，并具有一个用于代码开发、特性分析与调试的基于 Eclipse 的集成设计环境 (IDE)。利用该环境，开发人员可以创建针对不同数据中心应用可即时换入换出而优化的动态重配置加速器 , 可即时换入换出。开发人员可使用该环境创建诸多应用，这些应用能在运行时间内将众多内核换入和换出 FPGA，且不会干扰服务器 CPU 与 FPGA 之间的接口连接，从而实现不间断的应用加速。SDAccel 环境针对的是基于 x86 服务器处理器的主机系统，并提供现成商用插电式 PCIe 卡，以增加 FPGA 功能。

凭借 SDAccel 环境，开发人员即使先前没有 FPGA 经验也能使用 SDAccel 似曾相识的工作流程来优化他们的应用，并充分发挥 FPGA 平台作用。

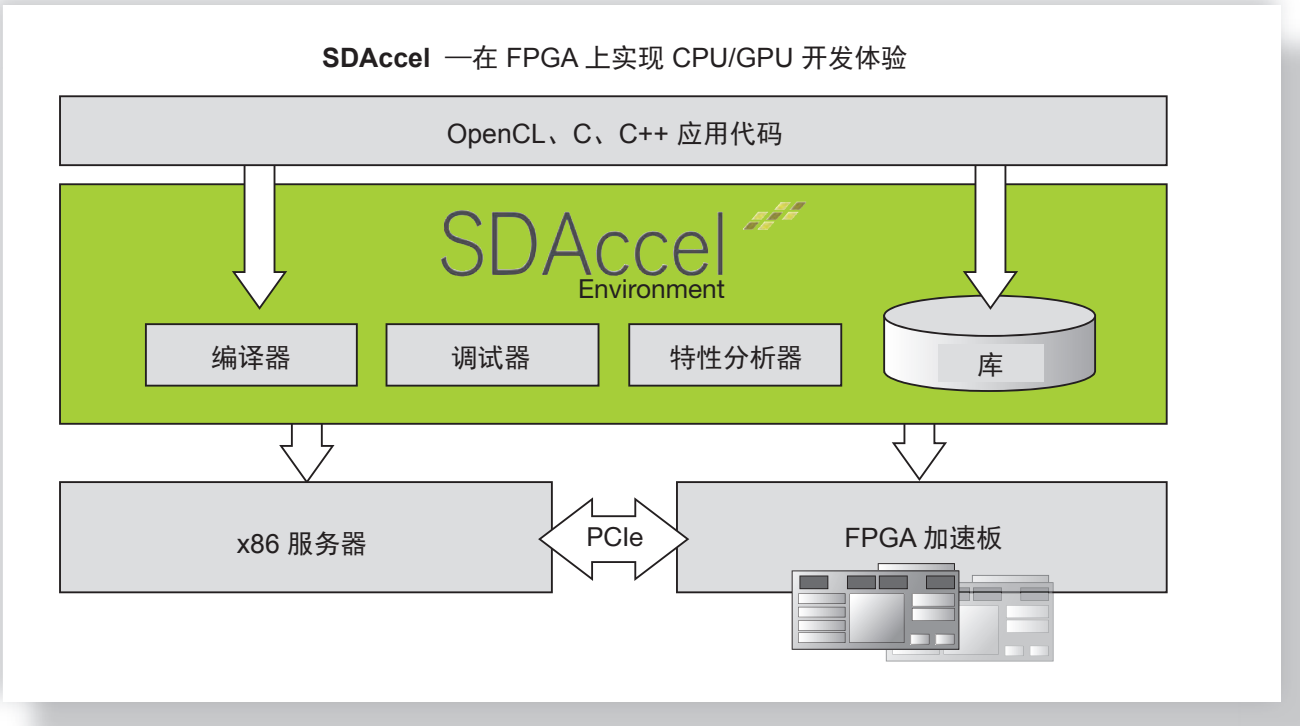


图 2 - 面向 OpenCL、C 和 C++ 的 SDAccel 开发环境可将单位功耗性能提高 25 倍，利用 FPGA 实现数据中心应用加速。

该 IDE 提供编码模板和软件库，并可用来针对所有开发目标进行编译、调试和特性分析，包括 x86 上的仿真、使用快速仿真进行性能验证，以及 FPGA 处理器上本地执行。该开发环境在面向数据中心的 FPGA 平台上执行应用，而且能自动插入工具以实现所有开发目标。赛灵思设计的 SDAccel 环境能够让 CPU 和 GPU 开发人员轻松将应用迁移到 FPGA 上，同时在熟悉的工作流程中维护和重用 OpenCL™、C 和 C++ 代码。

SDAccel 库对于 SDAccel 环境能实现类似于 CPU/GPU 的开发体验起到了重大作用。SDAccel 库包含低级数学库以及生产力更高的库（如 BLAS、OpenCV 和 DSP 库）。这些库用 C++（而非 RTL）编写而成，因此开发人员可在所有开发和调试阶段完全按所编写的内容来使用它们。在项目早期，所有开发工作都在 CPU 主机上完成。因为 SDAccel 库用 C++ 编写，所以它们能够与 CPU 目标（创建一个虚拟原型）的应用代码一起编译，从而允许所有测试、调试和初始特性分析在主机上进行。这个阶段无需 FPGA。

SDSOC 支持 ZYNQ SOC 和 MPSOC 系统的嵌入式开发

赛灵思为嵌入式系统开发人员设计了 SDSoC 开发环境，用以针对赛灵思 Zynq SoC 和即将推出的 Zynq UltraScale+ MPSoC 进行编程。SDSoC 环境提供极大简化的嵌入式 C/C++ 应用编程体验，包括一个可在裸机或操作系统（例如 Linux 和 FreeRTOS）上运

图 3 – SDSoC 开发环境提供熟悉的嵌入式 C/C++ 应用开发体验，包括简单易用的 Eclipse IDE 和综合而全面的设计环境，以实现异构 Zynq All Programmable SoC 和 MPSoC 部署。

行的简单易用的 Eclipse IDE。该环境是一种用于异构 Zynq SoC 和 Zynq MPSoC 平台部署的综合而全面的开发平台（图 3）。SDSoC 环境还配套提供业界首款 C/C++ 全系统优化编译器，支持系统级特性分析、可编程逻辑中的自动软件加速、自动系统连接生成，以及可加快编程速度的多种库。该环境还为客户和第三方平台开发人员提供开发流程，以使平台能够在 SDSoC 开发环境中使用。

SDSoC 为 ZC702 和 ZC706 等 Zynq All Programmable SoC 开发板，以及包括 ZedBoard、MicroZed、ZYBO 和视频与成像开发套件等在内的第三方和市场专用平台提供板支持包 (BSP)。BSP 包含能够从软件开发人员和系统架构师的代码中将平台抽象出来的元数据，以简化智能异构系统的创建、集成和验证。

SDNET 支持 FPGA 加速线卡的设计和编程

SDNet 是一种软件定义规范环境，它使用直观的类似于 C 的高级语言来设计要求并创建网络线卡规范（图 4）。该环境使网络架构师和开发人员能够创建“软”定义网络，将可编程性和智能化从控制层扩展到数据层。

传统的软件定义网络架构采用固定的数据层硬件，通过狭窄的南向 API 连接到控制层，而软定义网络则不然，其基于可编程数据层，具有内容智能功能和宽泛的北向 API 控制层连接。这样能实现多种突破性的功能，包括：支持独立的线速服务，避免各种协议的复杂性；提供以流程为单位的灵活服务；支持革命性的创新型“无损业务”升级，同时以 100% 的线路速率运行。

这些独特的功能使运营商和多业务系统运营商

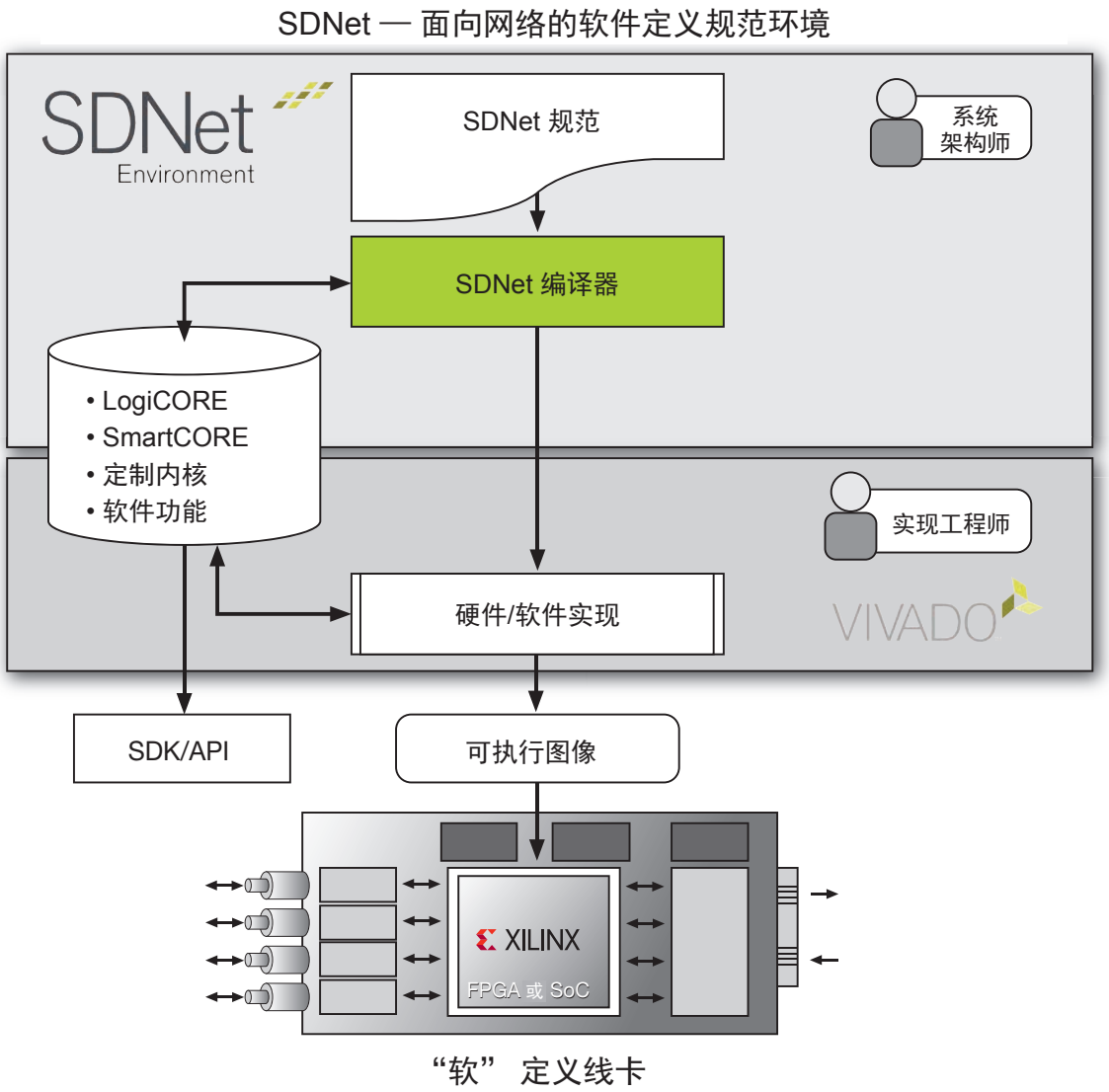


图4 - SDNet 环境使网络架构师能以类似于 C 的语言创建规范。硬件团队完成设计后，开发人员可使用 SDNet 现场为线卡升级或添加协议。

(MSO) 能够动态提供差异化服务，且不会对现有服务造成任何中断，也不需要硬件质量认证或进行上门服务。该环境的动态服务提供功能使服务提供商能够增加营收并加快服务上市速度，同时降低资本支出和运营支出。网络设备供应商意识到同样能够通过 SDNet 平台获得相同的优势，使他们能够通过部署 SDNet 环境编程的内容感知式数据层硬件，实现更大的差异化。

嵌入式开发环境

为了进一步帮助嵌入式软件工程师进行编程，赛灵思提供全套嵌入式工具和运行时间环境，以帮助

MathWorks 和赛灵思技术的这种组合帮助公司客户打造出众多创新产品。

嵌入式软件开发人员从概念到生产高效编程。赛灵思为开发人员提供一个名为赛灵思软件开发套件 (SDK) 的基于 Eclipse 的 IDE，其中包含编辑器、编译器、调试器、驱动程序和多种库，针对的是 Zynq SoC 或含有赛灵思 32 位 MicroBlaze™ 软核的 FPGA。该环境开箱即用并支持多种高级功能，例如构建在赛灵思独特 Zynq SoC 和 MPSoC 之上的安全和虚拟化软件驱动程序。这让开发人员可以构建智能、安全的真正差异化的连接系统。

赛灵思提供全套开源资源，可用以开发、引导、运行、调试和维护在赛灵思 SoC 或仿真平台上运行的 Linux 应用。赛灵思提供实例应用、内核构建、Yocto 方法、多处理与实时解决方案、驱动程序和论坛，以及社区链接。Linux 开源开发人员会找到一个非常舒适的环境，在其中进行学习和开发，并与其他具有相同兴趣和需求的用户互动交流。

不断壮大的强大编程环境联盟

除了为开发人员提供新的 SDx 开发环境和 SDK 以外，赛灵思还在过去十年中与很多已经具有完善开发环境的公司组建了强大的联盟，这些开发环境都在特定的细分市场为开发人员服务。

美国国家仪器公司 (NI)（美国德克萨斯州 Austin）提供的硬件开发平台倍受控制和测试系统创新人员青睐。赛灵思的 FPGA 和 Zynq SoC 则为 NI RIO 平台注入动力。NI 的 LabVIEW 开发环境是一款用户友好型图形化程序，可以运行赛灵思的 Vivado Design Suite，这样 NI 的客户就无需知道任何 FPGA 设计细节，有些客户或许都不知道赛灵思器件位于 RIO 平台的核心。但他们可在 LabVIEW 环境中对系统简单编程，并让 NI 的硬件为他们所开发的设计实现进行性能加速。

MathWorks®（美国马萨诸塞州 Natick）在十多年前就在其 MATLAB®、Simulink®、HDL Coder™ 和 Embedded Coder® 中提供了 FPGA 支持，赛灵思的 ISE® 和 Vivado 工具可在其中全自动运行。这样，用户（主要是数学专业的算法开发人员）就可以在开发算法时简便地在 FPGA 架构上运行算法，并显著提升算法性能。

赛灵思在十多年前为其 ISE 开发环境添加了名为 System Generator 的 FPGA 架构级工具，最近，又将该工具添加到 Vivado Design Suite 中，以便让具有 FPGA 知识背景的团队能够对设计进行调整，以进一步提升算法性能。MathWorks 和赛灵思技术的这种组合帮助公司客户打造出出众多创新产品。

ARM、劳特巴赫有限公司 (Lauterbach)、日本横河数字计算机公司 (Yokogawa Digital Computer Corp) 和京都微机集团 (Kyoto Microcomputer Corp) 等众多赛灵思联盟成员可提供支持 SDx 和 Alliance 环境的开发工具。在 OS 和中间件支持方面，赛灵思以及联盟成员为客户提供多种软件选择，诸如 Linux、RTOS、裸机，甚至包括管理程序和支持 Trust-Zone 内核的解决方案，满足保密性和安全性需求。

如需了解有关 SDx 环境以及赛灵思广泛且不断扩展的开发人员解决方案，敬请访问赛灵思最新的软件开发者园地。■

运用 SDSoC 逐步构构建一个设计样例

作者：Adam Taylor
e2v 公司总工程师
aptaylor@theiet.org

ZedBoard 实例证明采用无缝环境可以快速构建并优化设计。

在赛灵思 SDSoC™ 开发环境发布前，标准 SoC 设计方法涉及一系列不同的工程设计技巧。通常来说，如果系统架构师根据要求生成系统架构和子系统，那么解决的方案会分为硬件功能的实现（逻辑侧）和软件功能的实现（处理器侧）。这样，FPGA 和软件工程师要分别开发各自的功能，再根据集成测试计划进行组成和测试。这种方法持续运用了多年，但赛灵思 Zynq®-7000 All Programmable SoC 以及即将推出的赛灵思 Zynq UltraScale+™ MP-SoC 等功能更强的 SoC 的问世要求采用新的设计方法。

SDSoC 方法能帮助更广泛的工程师用户群开发极高性能系统。在 SDSoC 开发环境中进行开发的工程师新手会发现，可以轻松快速地启动并运行系统，而且系统优化也同样便捷。

我们举一个简单而具代表性的例子来说明如何完成上述任务并获得相应的收益。我们针对的是运行 Linux 的 ZedBoard，并采用内置示例之一：矩阵乘法器和加法模板。

设计方法简史

可编程逻辑器件自 1980 年推出以来一直迅速发展。起初，工程师通过原理图输入法对器件进行编程（虽然 22v10 等较早期的可编程逻辑器件 (PLD) 是通过逻辑方程编程的）。这需要电子工程师执行大部分 PLD 开发工作，因为逻辑设计和优化通常属于电子工程学范畴。随着器件尺寸和功能的不断提升，这种原理图输入法自然开始面临局限性问题，因为设计和验证时间会随着设计复杂性的增加而显著延长。工程师需要具备在更高抽象层上工作的能力。

VHDL 和 Verilog 应运而生。二者开始都是用来描述和模拟逻辑设计（特别是 ASIC）的语言。VHDL 甚至有自己的军用标准。既然我们要在硬件描述语言 (HDL) 中描述逻辑行为，那么自然应该综合所需的逻辑电路。综合工具的开发使得工程师能描述通常在寄存器传输层的逻辑行为。HDL 也能大幅改进验证方法，有助于开发行为测试平台，支持结构化验证。HDL 也首次支持模块化和厂商独立性。

HDL 内在并行性、寄存器传输层设计法和实现流程则需要了解优化和时序收敛，这就意味着 PLD 开发任务大部分落到了电子工程师的身上。

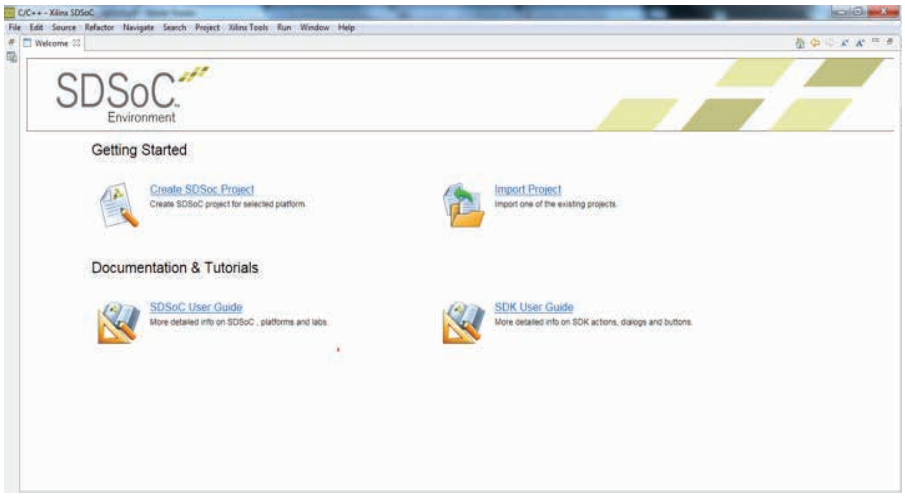


图 1 - SDSoC 欢迎界面

熟悉的环境

SDSoC 开发环境基于 Eclipse，大多数软件开发人员对这种环境应该是比较熟悉的（图 1）。该环境通过采用能处理 C 或 C++ 程序的全新 SDSoC 编译器无缝支持器件可编程逻辑 (PL) 侧功能加速。

HDL 一直是 PLD 开发领域的事实标准，但多年发展中也考虑到整体产业的发展需求。单就 VHDL 而言，相继于 1987 年（IEEE 采用的第一年）、1993 年、2000 年、2002 年、2007 年、2008 年进行了修订。不过，跟原理图输入法一样，HDL 也逐渐开始面临一系列局限性问题，包括开发和验证时间太长、器件功能要求提高等。

随着 PLD 的角色从胶合逻辑向加速外设甚至最终向系统核心发展，业界需要新的设计方法来满足有关发展需求。近年来，高层次综合 (HLS) 越来越受欢迎。根据高层次综合法，设计通过 C/C++（用赛灵思的 Vivado® HLS）或其他工具（如 MathWorks 公司的 MATLAB® 或国家仪器公司的 LabVIEW）输入。由于新的设计方法能够充分发挥器件的功能，这种方法开始将设计和实现从电子工程领域转向软件领域，从而显著扩大了潜在的 PLD 设计人员群体，并巩固了 PLD 作为系统核心的地位。

因此，SoC 设计自然能用 HLS 来生成紧密集成的开发环境，工程师则能在环境中无缝加速设计逻辑侧功能。在此情况下，SDSoC 环境应运而生。

SDSoC 环境中使用最高抽象层进行开发，具体如下：

1. 在 C 或 C++ 中开发应用。
2. 对应用进行特性分析，明确性能瓶颈问题。
3. 使用特性分析信息，明确器件 PL 侧哪些功能需要加速。
4. 随后构建系统并生成 SD 卡映像。
5. 硬件到位后，进一步分析性能并根据要求优化加速功能。

我们能在 SDSoC 环境中开发出可在裸机、FreeRTOS 或 Linux 操作系统上运行的各种不同应用。该环境内置支持 ZedBoard、MicroZed 和 Digilent ZYBO 等大部分 Zynq SoC 开发板。这样，我们不仅能加速开发应用，还能通过该功能定义自己的底层硬件平台，在自己的定制硬件平台能用于集成时使用。

在 SDSoC 环境中编译程序时，构建过程输出提供了从 SD 卡配置 Zynq SoC 所需的系列文件。该系列文件包括一级和二级引导载入程序，以及操作系统所需的应用和映像。

SDSOC 实例

下面我们看看 SDSoC 环境如何工作，多快就能让实例启动并运行起来。我们针对的 ZedBoard 开发板运行 Linux 操作系统并使用内置矩阵乘法器和加法模板。

首先一如既往地创建项目。我们能通过欢迎窗口（图 1）或通过菜单选择“File -> New -> SDSoC”项目来创建项目。这两种方法都能打开对话框，让我们给项目命名，选择开发板和操作系统（图 2）。

这将在 SDSoC GUI 左侧 Project Explorer 下生成项目。我们在该项目下会看到以下文件夹，每个文件夹都有自己唯一的图标：

- SDSoC 硬件功能：这里我们能看到转移到硬件的功能。一开始这个文件夹是空的，等待我们转移。
- Includes：展开这个文件夹能看到构建过程中使用的所有 C/C++ 报头文件。
- src：包含演示用源代码。

为确保我们正确配置所有事项，包括 SDSoC 安装和环境以及开发板，我们要构建演示，其只在器件片上处理系统 (PS) 侧运行。

当然，下一步就是构建项目了。在菜单上选中项目，我们选择“Project->Build Project”。构建时间不会太长。完成后，会看到如图 3 所示的文件夹出现在 Project Explorer 中的项目下。除了上述文件夹之外，还有：

- 二进制文件：这里能找到软件编译过程生成的可执行连接 (ELF) 文件。
- 归档：链接对象文件，创建其中的二进制文件。
- SDRelease：包含我们的引导文件和报告。

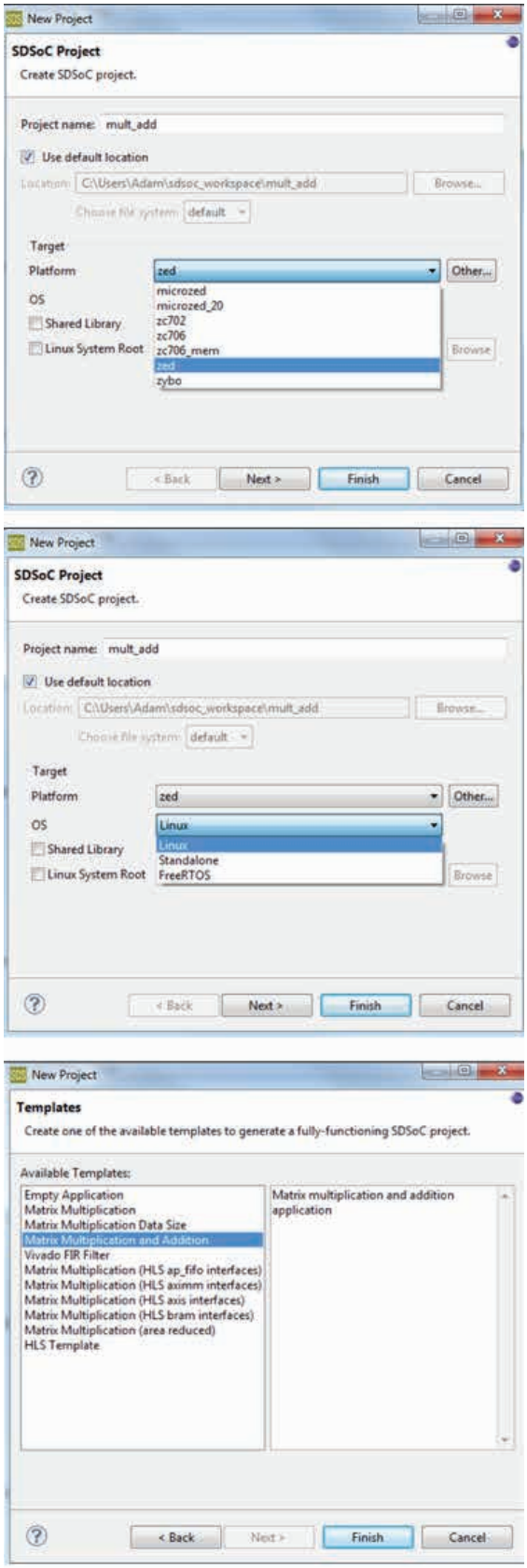


图 2 – 创建项目

我们已经构建了首个演示，只能在 Zynq SoC 的 PS 上运行，下面我们来看看如何确保其按预期正常运行。对应用进行特性分析，明确 SDSoC 加速运行。工程师用特性分析信息明确应该移动哪些功能。

我们用已提供的 sds_lib.h 库在基本层面进行特性分析，这能在 64 位全局计数器上获得基本时间戳 API，让我们测出每个功能的用时。我们用 API 能简单记录功能开始和停止时间，这一时间差就是整个过程的执行时间。

源代码包括两个版本的矩阵乘法和加法算法。所谓的黄金版不用于转移到片上可编程逻辑 (PL)；另一版则针对这个目的。通过在 PS 中构建并运行，我们能确保有针对性地对比，而且两个流程执行时间基本相同。

构建完成后，我们能拷贝“Project Explorer 下 SDRelease -> sd_card”文件夹中的所有文件到 SD 卡，然后将 SD 卡插到 ZedBoard 上（SD 卡配置的模式引脚要正确设置）。终端程序连接以及引导序列完成后，我们就要运行程序。输入“/mnt/mult_add.elf”（这里的 mult_add 是我们创建的项目的名称）。在 ZedBoard 上运行，得到如图 4 所示的结果，显示两个功能运行的时间基本相同。

验证执行时间基本相同后，我们将乘法函数移到 SoC 的 PL 侧，这步很容易实现。

分析实例中 src 目录的文件结构，我们看到：

- main.cpp：这包含主函数、黄金计算、时间戳和器件硬件侧所用的乘法加法函数调用。
- mmult.cpp：这包含在硬件部分运行的乘法函数。
- madd.cpp：这包含在硬件部分运行的加法函数。

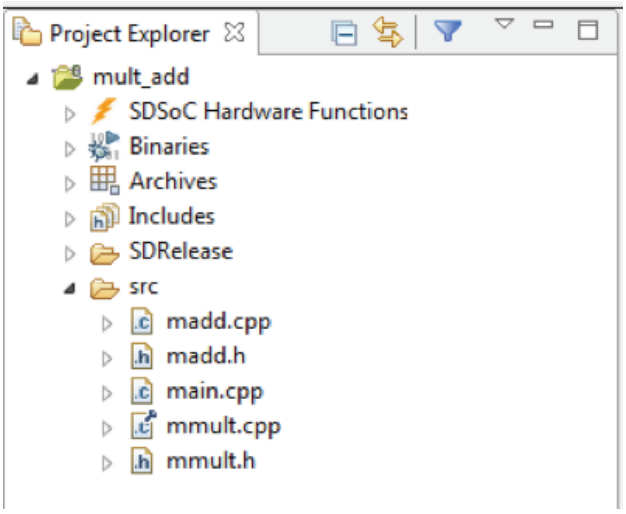


图 3 - 构建后的 Project Explorer 视图

```
sh-4.3# /mnt/mult_add.elf
Testing with A_NROWS = A_NCOLS = B_NCOLS = B_NROWS = 32
Testing mmult ...
Average number of processor cycles for golden version: 181607
Average number of processor cycles for hardware version: 183289
TEST PASSED
sh-4.3#
```

图 4 – PS 中两个功能的执行时间

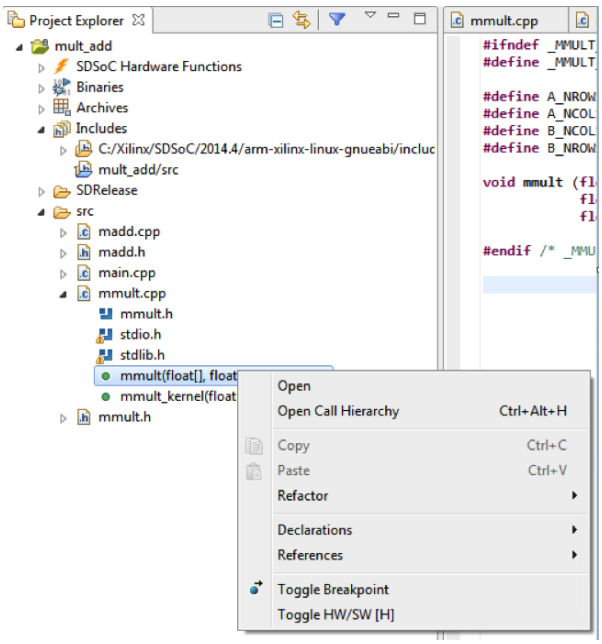


图 5 - 用 Project Explorer 移动乘法器内核到 PL 侧

完成上述步骤后，下次构建项目时，SDSoC 链接器会自动调用赛灵思 Vivado HLS 和Vivado 来执行 SoC 中 PL 侧的功能。

下一步就是将一个函数转移到 SoC 的 PL 侧。我们通过以下两种方法之一能完成这一任务：

1. 在 Project Explorer 中，我们展开文件，看到文件中的函数，选择感兴趣的函数，右键点击并选择“Toggle HW/SW [H]”（图 5）。
2. 打开文件并执行右侧“outline”标签下的同样选项，这也能显示函数（图 6）。

开关硬件中加速的 mmult() 函数能让 [H] 添加到函数后面（图 7）。

我们还会看到 SDSoC Hardware Functions 下选择的功能（“Project Explorer”标签中项目下方，图 8）。这能让我们方便地看到设计中加速的所有功能。

完成上述步骤后，下次构建项目时，SDSoC 链接器会自动调用赛灵思 Vivado HLS 和 Vivado Design Suite 其余部分来执行 SoC 中 PL 侧的功能。与此同时，会创建软件驱动程序，支持功能加速。我们看到，功能转移到器件 PL 侧是无缝的，而且还会提高性能。

编译和生成 SD 卡映像后，我将 mmult() 函数移到硬件，在 ZedBoard 上运行。如图 9 所示，执行时间（处理器工作循环）只有 52,444 / 183,289 = 0.28，也就是此前期间 PS 侧 183,289 个处理器工作循环（执行时间）的 28%（图 4）。考虑到器件 PS 侧相同功能的性能，我们看到只需简单点击鼠标就能大幅改进执行时间。

这里给出的例子可以直接看出 SDSoC 环境和紧密集成的 HLS 功能的强大无缝功能。■

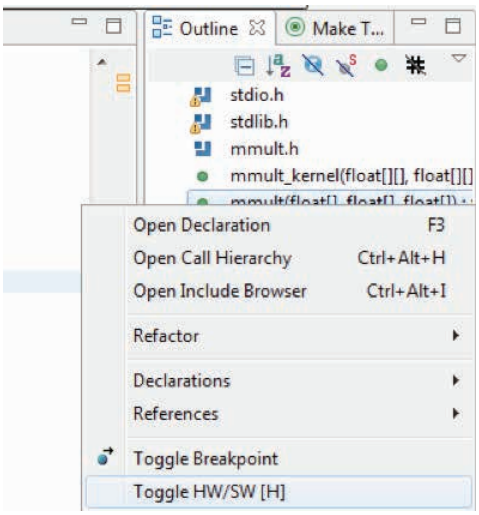


图 6 - 用“Outline”窗口将乘法器内核移到 PL 侧

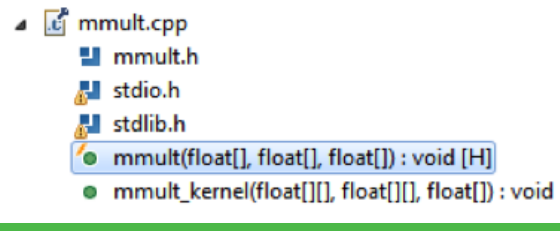


图 7 — 硬件中的 mmult() 函数

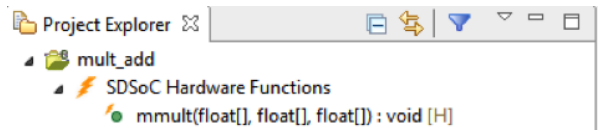


图 8 - 明确加速功能

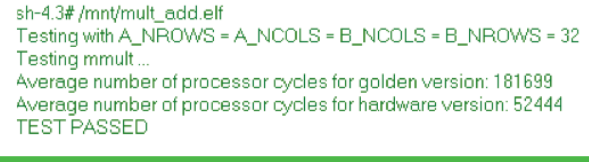
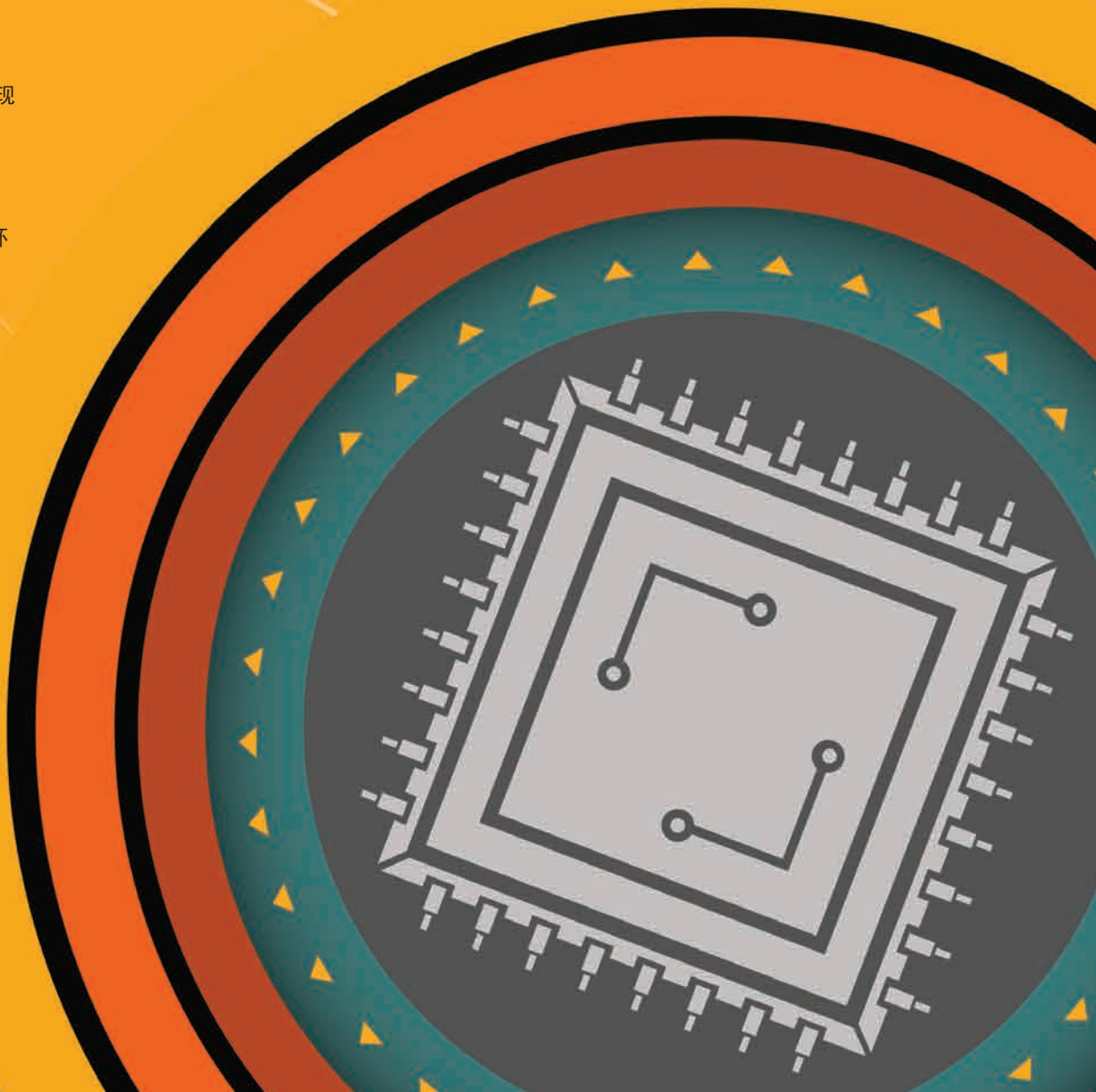


图 9 - 加速结果

如何使用 SDSoC IDE 在 Zynq SoC 上进行系统级软硬件优化

作者：Daniele Bagni
赛灵思公司 DSP 专家级现场应用工程师
danieleb@xilinx.com

Nick Ni
赛灵思公司SDSoC开发环境部产品经理
nickn@xilinx.com



Choleksy 矩阵分解实例在几分钟内生成加速估算结果。

赛灵思 Zynq®-7000 All Programmable SoC 系列代表了嵌入式设计的新局面，为嵌入式系统工程设计群体带来前所未有的高性能和灵活性。这些产品在单个器件上集成了特性丰富的双核 ARM® Cortex™-A9 MPCore™ 处理系统和赛灵思可编程逻辑。3,000 多个互联点将片上处理系统 (PS) 与片上可编程逻辑 (PL) 相连，使两个片上系统之间的性能达到了任何双芯片处理器 FPGA 都无法匹敌的高度。赛灵思在 2011 年发布这款 Zynq SoC 器件时，便立即得到了很多精通硬件设计语言和方法以及嵌入式软件开发的嵌入式系统工程师和架构师的追捧。如今，首款 Zynq SoC 已被用于无线基础设施、智能工厂和智能视频 / 视觉等多种嵌入式应用领域，并且快速成为高级驾驶员辅助系统约定俗成的标准平台。

为了让这款性能卓越的器件能够被具有很强软件背景但没有 HDL 经验的嵌入式工程师所用，赛灵思在今年年初推出了基于 Eclipse 的 SDSoC™ 集成开发环境，使软件工程师能够对 Zynq SoC 中的可编程逻辑和 ARM 处理系统进行编程。

我们仔细了解一下 Zynq SoC [1] 的特性以及软件工程师如何利用 SDSoC 环境来创建用任何其他“处理器 +FPGA”系统均无法实现的系统设计。为进行研究，我们将使用包含一个 Zynq Z-7020-1 器件的赛灵思 ZC702 评估板 [2] 作为硬件平台。

如图 1 所示，Zynq SoC 包含两大功能模块：PS（包含应用处理器单元、存储器接口、外设和互联）和 PL（传统的 FPGA 架构）。

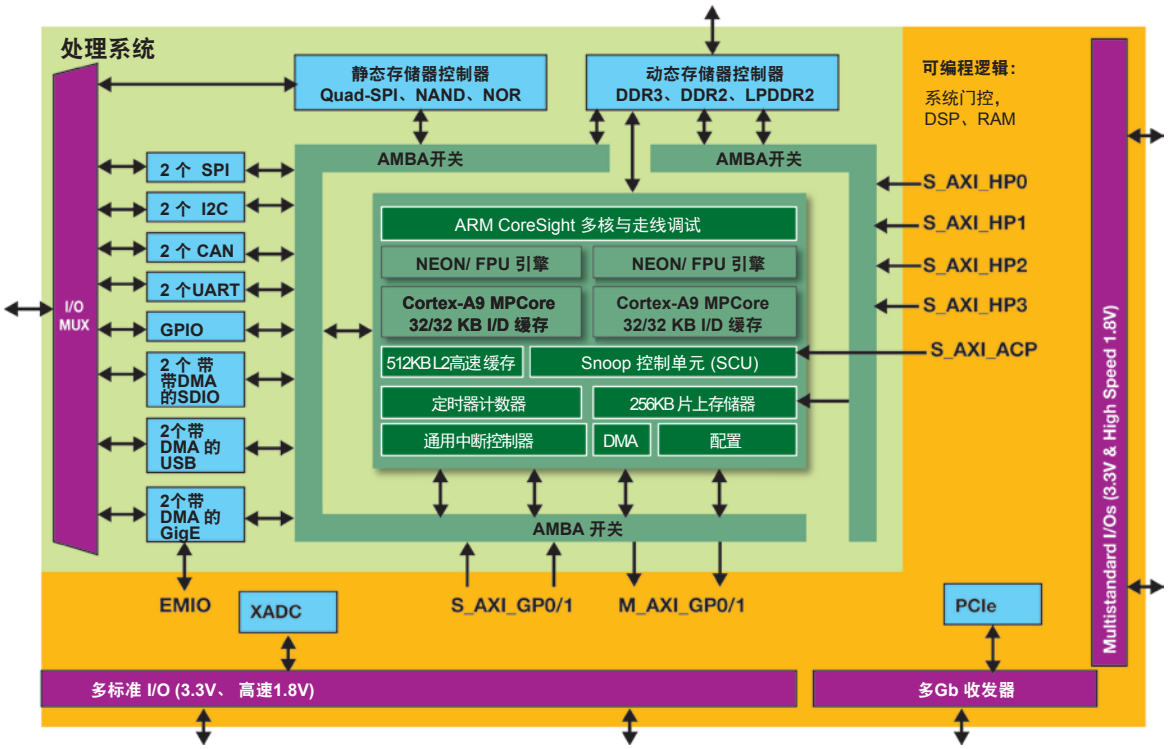


图 1- Zynq 高级架构简介

PS 和 PL 通过与 ARM® AMBA® AXI4 接口兼容的互联链路紧密耦合在一起。四个高性能 (HP) AXI4 接口端口将 PL 连接到 PS 中的异步 FIFO 接口 (AFI) 模块，从而在 PL 和 PS 存储器系统（DDR 和片上存储器）之间提供一条高吞吐量的数据路径。AXI4 加速器一致性端口 (ACP) 允许直接从 PL 主设备对 L1 和 L2 高速缓存进行低时延的一致性访问。通用 (GP) 端口包含可从 PS 和 PL 访问的低性能通用端口。

在传统的以硬件设计为核心的流程中，通过使用赛灵思的 Vivado® Design Suite 在 Zynq SoC 上设计嵌入式系统大致需要四个步骤：

1. 系统架构师确定硬件 - 软件分区方案。计算密集型算法最适合硬件。特性分析结果是鉴别性能瓶颈以及在数据移动成本与加速优势之间进行权衡研究的基础。

2. 硬件工程师处理被分到硬件中的功能，并将它们转换或设计成 IP 核，例如，用 Vivado 取代 VHDL 或 Verilog；用 Vivado 高层次综合 (HLS) 实现 C/C++ 高层次综合 [3]，或者用 Vivado System Generator for DSP 实现模型化设计 [4]。

3. 然后，工程师使用 Vivado IP Integrator [5] 创建整个嵌入式系统的模块化设计。整个系统的开发需要用到不同的数据移动工具（AXI-DMA、AXI Memory Master、AXI-FIFO 等），以及连接 PL IP 与 PS 的 AXI 接口（GP、HP 和 ACP）。一旦 IP Integrator 中所有设计规则检查都通过，就可将项目导出至赛灵思软件开发套件 (SDK) [6]。

4. 软件工程师使用赛灵思 SDK 开发针对 PS 中 ARM 处理器的驱动程序和应用。

最近几年，赛灵思对 Vivado Design Suite 的简便易用性实现了显著提高，该套件能够让工程师缩短 IP 开发时间以及 IP 模块连接步骤（上述步骤 2 以及步骤 3 的部分内容）。就 IP 开发而言，采用这些最新设计技术（例如在 Vivado HLS 工具中实现 C/C++

SDSoC 环境会自动安排所有必要的赛灵思工具，以生成针对 Zynq SoC 的完整软硬件系统，而且所需的用户介入程度很小。

高层次综合；用 Vivado System Generator for DSP 实现模型化设计）可以显著缩短开发和验证时间，同时让设计团队能够使用高层次抽象探索更大的架构范围。对于使用 VHDL 或 Verilog 需要数周才能完成的设计，利用新型工具只需数天就能完成。

赛灵思通过 Vivado IP Integrator 可进一步强化流程。Vivado Design Suite 的这个功能使设计人员只需要在图形用户界面 (GUI) 中连接 IP 模块，就可以设计复杂的硬件系统（嵌入式或非嵌入式），从而实现快速的硬件系统集成。

新的 Vivado Design Suite 功能可帮助设计与开发团队简化 Zynq SoC 的使用。但对于以硬件为中心的优化流程而言，要缩短探索不同数据移动工具和 PS-PL 接口（步骤 3 的一部分）以及写入和调试驱动程序与应用（步骤 4）所需的开发时间，办法并不多。如果整个系统不能满足吞吐量、时延或占位面积等方面的设计要求，那么设计团队不得不修改步骤 3 中的系统连接，以重构硬件架构。这些修改会不可避免地导致步骤 4 中的软件应用发生变化。有些情况下，加速不足或者硬件利用率过高会迫使开发团队重新考虑最初的软硬件分区。多个硬件和软件团队将不得不再次反复设计系统，以探索可能满足最终要求的其他架构。

这些实例表明手动进行系统优化对上市时间的影响。系统优化对于紧密集成的系统（例如 Zynq SoC）很关键，因为瓶颈经常发生在 PS 与 PL 之间的系统连接上。

SDSoC 环境能大大简化 Zynq SoC 开发过程，可将步骤 2、3、4 实现高度自动化，以缩短总开发时间。该开发环境会生成必要的硬件和软件组件，

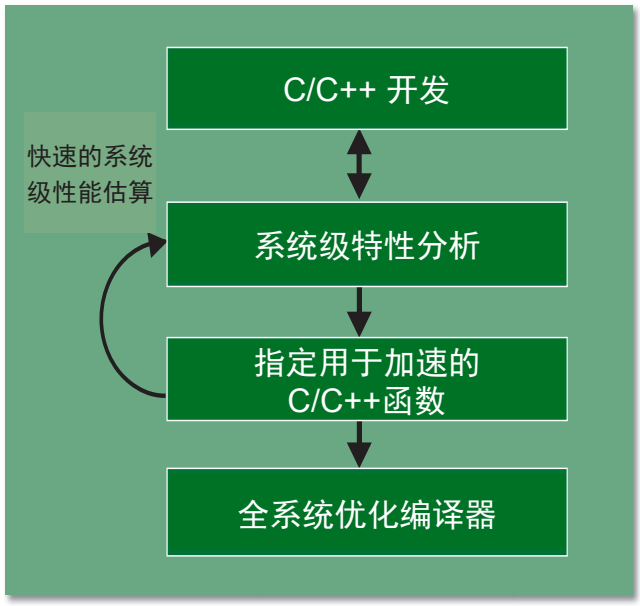


图 2 — SDSoC 设计流程中的主要步骤

用以同步硬件和软件并保存源程序语义，同时支持任务级并行处理和流水线化的通信与计算，从而实现高性能。SDSoC 环境会自动安排所有必要的赛灵思工具（Vivado、IP Integrator、HLS 和 SDK），以生成针对 Zynq SoC 的完整软硬件系统，而且所需的用户介入程度很小。

假设我们已有一个针对 PS 的完全用 C/C++ 描述的应用，而且已经决定将哪些功能分配到 PL 以实现加速，那么 SDSoC 开发流程大致按照如下所示（图 2）：

1. SDSoC 环境使用快速估算流程（通过调用内含的 Vivado HLS）构建应用项目。这样在数分钟内就能大致估算出性能和资源情况。

2. 如果我们认为有必要，我们会用适当的指令优化

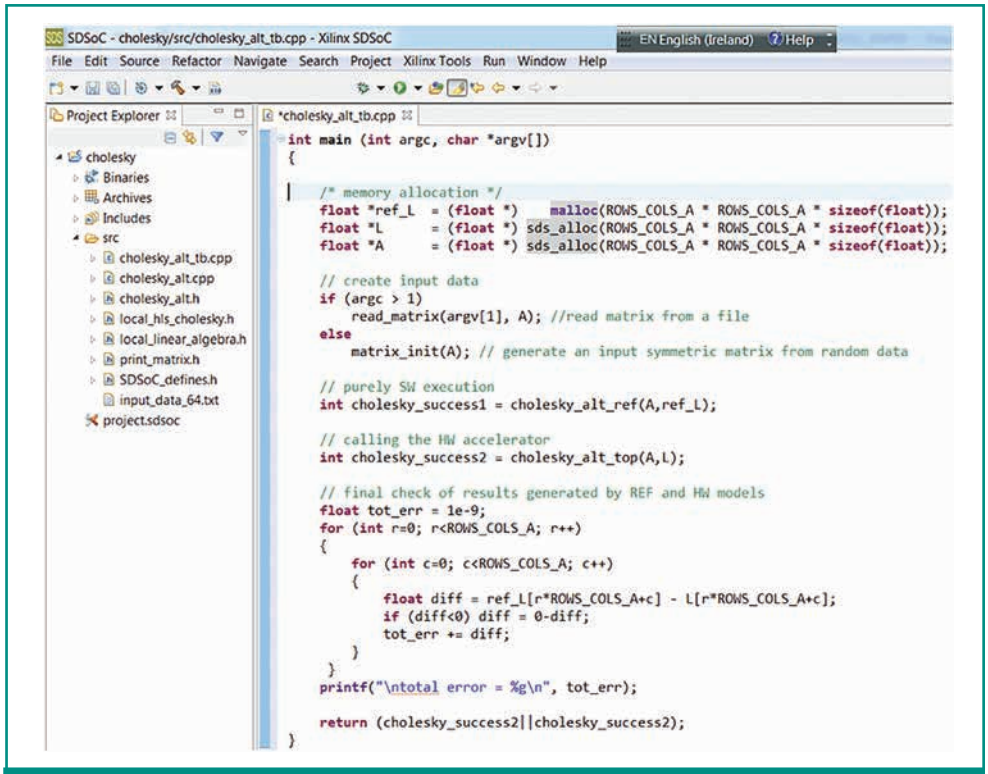


图 3 — 针对 SDSoC 环境的 C/C++ 测试平台结构

Cholesky 分解将正定矩阵变换为具有严格正对角线的下三角和上三角矩阵的乘积。矩阵 B 在三角矩阵 L 中分解，即 B = L’ * L，其中 L’ 是 L 的转置矩阵，如下面的 4 x 4 矩阵 MATLAB® 代码所示：

```
A = ceil(64*randn(4,4)) % generate random
                          data
B = A * A'              % make the matrix to
                          be symmetric
L = chol(B)             % compute cholesky
                          decomposition
B2 = (L' * L)           % reconstruct the
                          original matrix B
A =
      -13        53        41        20
      -19        98        12         9
        2        30       -65        33
        4       -13        61        17
B =
     5059      6113      -441      2100
     6113     10190      2419      -465
     -441      2419      6218     -3786
     2100      -465     -3786      4195
L =
  71.1266   85.9453   -6.2002   29.5248
        0   52.9472   55.7513  -56.7077
        0         0   55.4197   -7.9648
        0         0         0    6.6393
B2 =
     5059      6113      -441      2100
     6113     10190      2419      -465
     -441      2419      6218     -3786
     2100      -465     -3786      4195
```

让我们看看如何估算应用设计的性能和资源利用

C/C++ 应用和硬件功能，并重新运行估算直到实现所需的性能和占位面积。

3. 然后，SDSoC 环境构建整个系统。该过程会生成完整的 Vivado Design Suite 项目和比特流，以及一个针对 Linux、FreeRTOS 或裸机的可引导的运行时间软件映像。

用 SDSOC 环境进行硬件 VS 软件的性能估算

线性代数几乎在任何工程领域中都是基础而强大的工具，能够计算求解具有多维变量的方程组。例如，工程师可将线性控制理论系统描述为“状态”和“状态变换”矩阵。图像的数字信号处理是线性代数的另一个典型应用。尤其是，通过 Cholesky 分解进行矩阵求逆被认为是求解方程组或矩阵求逆的最有效的方法之一。现在让我们仔细看一看 32 位浮点实际数据 64 x 64 矩阵的 Cholesky 分解，并作为 Zynq SoC 上软硬件分区的应用实例。

选择加速器很简单，只需在 SDSoC 环境的图形用户界面 (GUI) 中用鼠标点击特定功能即可。

率，而且无需经过整个构建周期。

图 3 给出了适合 SDSoC 环境的测试平台结构。主程序为所有空矩阵分配动态存储器并填入数据（从文件中读入或者随机生成）。然后，主程序调用参考软件函数和硬件备选函数。最后，主程序检查两个函数计算出的数值结果以测试有效正确性。

注意，这里针对每个输入 / 输出数组使用了一个名为 sds_alloc 的专用存储分配器，以让 SDSoC 环境自动在硬件加速器的每个 I/O 端口之间插入一个 Simple DMA IP；相比之下，malloc 则实例化一个 Scatter-Gather DMA，用以处理分布在物理地址空间中多个非连续页面上的数组。Simple DMA 从占位面积和性能开销上将要比 Scatter-Gather DMA 廉价，但需要 sds_alloc 获得物理上的连续存储空间。

选择加速器很简单，只需在 SDSoC 环境的图形用户界面 (GUI) 中用鼠标点击特定功能即可。如图 4 所示，例程 cholesky_alt_top 标记为“H”，以表明它将被送到硬件加速器。我们还可以为加速器和数据移动内核选择时钟频率（如图 4 的 SDSoC 项目页面中所示为 100 MHz）。

现在，我们可以启动“估算加速”过程。在经过几分钟的编译后，我们就可以在 Vivado 项目中生成所有内核和数据移动网络。SDSoC 环境还生成一个 SD 卡映像，其中包含 Linux 引导映像，里面有 FPGA 比特流以及纯软件版本的二进制应用程序。我们从这个 SD 卡引导，并在 ZC702 目标平台上运行

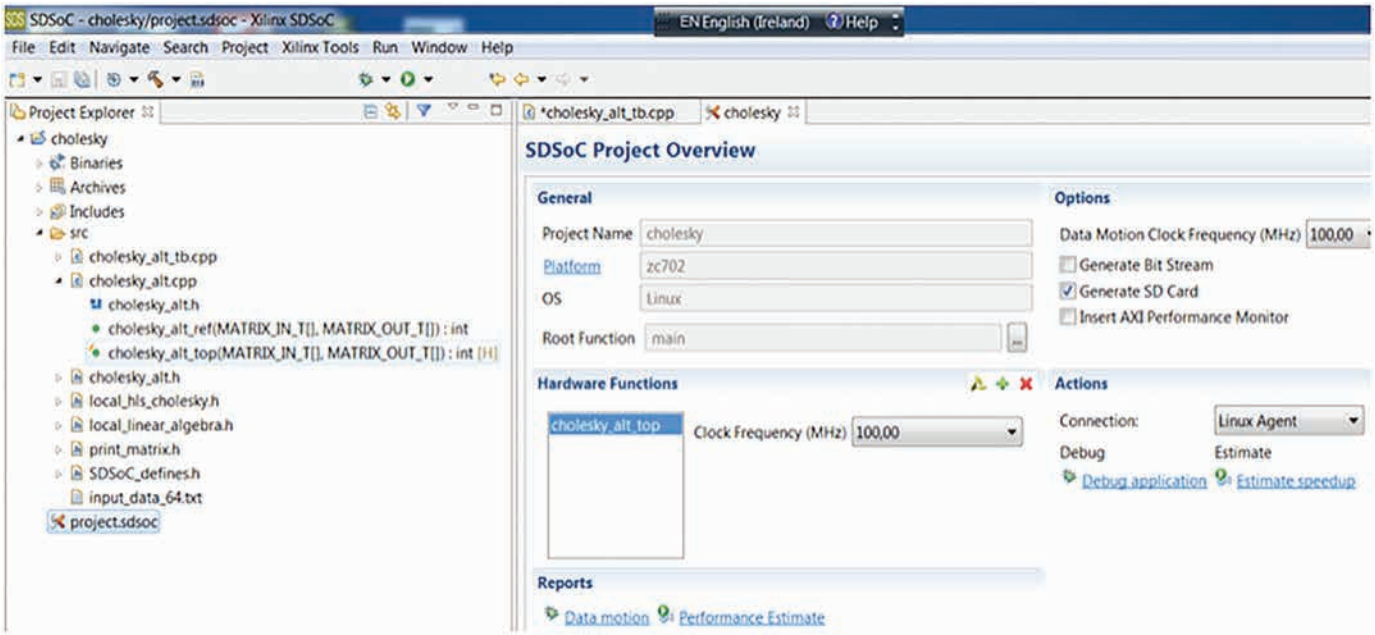


图 4 - 从 SDSoC 项目页面设定硬件加速器内核及其时钟频率

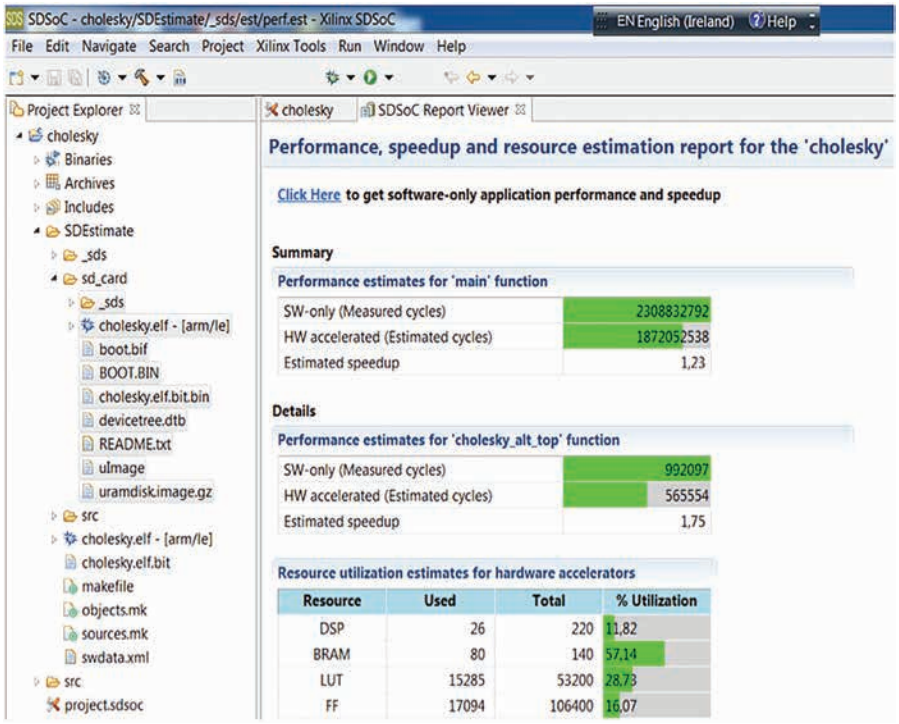


图 5 - SDSoC生成的性能、加速和资源估算报告

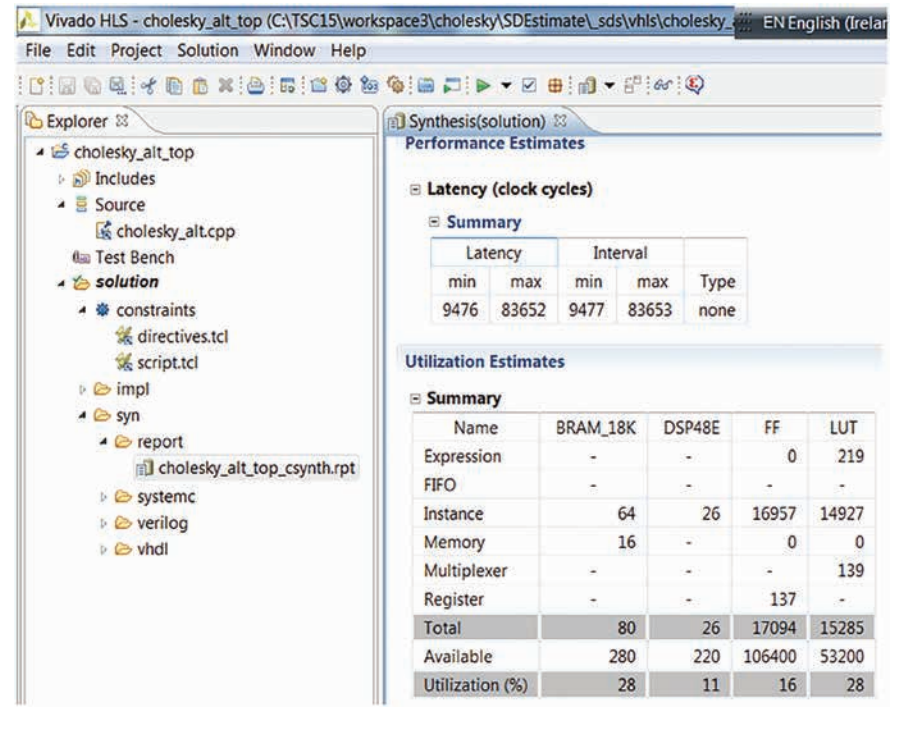


图 6 - Vivado HLS综合估算报告

应用。

Linux 在开发板上启动之后，我们可执行纯软件应用，然后，SDSoC 环境生成图 5 中的性能估算报告。如果在硬件中而非软件中执行，我们可看到 cholesky_alt_top 函数的 FPGA 资源利用率 (26 DSP、80 BRAM、15,285 LUT、17,094 FF) 和性能加速 (1.75)。

在主应用程序方面，我们还可以看到总体加速比较低 (1.23)，原因在于其他软件开销，例如 malloc 和数据传输。我们的整个应用比较小，主要在于展示 SDSoC 流程和设计方法；我们需要在 PL 中加速更多例程，但这超出了本文的范畴。

使用 SDSoC 环境，只要几分钟就能生成该信息，且无需综合和布局布线这样的 FPGA 编译过程；这些过程根据硬件系统的复杂程度可能需要数小时才能完成。这样的估算通常足以用来分析硬件 - 软件分区的系统级性能，并让用户能够迅速进行设计迭代以创建出最佳系统。

了解性能估算结果

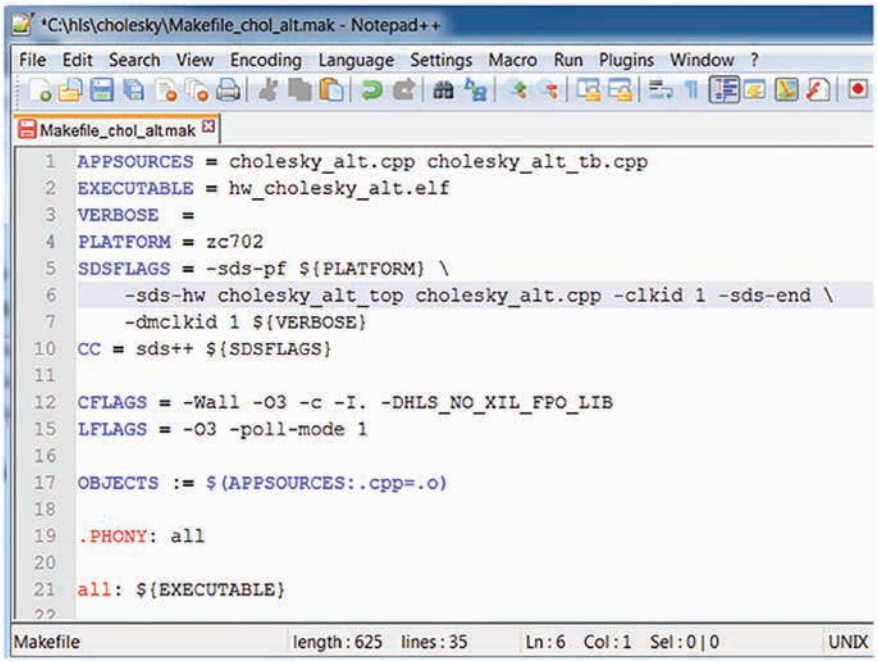


图 7 – 用于软件构建的 Makefile

当 SDSoC 开发环境针对估算加速进程编译应用代码时，会生成一个中间目录（图 5 中的 _sds)，用来放置所有中间项目 (Vivado HLS、Vivado IP Integrator 等)。它会特别在源代码中插入对自由运行的 ARM 性能计数器函数 sds_clock_counter() 的调用，以测量程序函数中关键部分的执行时间。这也就是为什么目标板需要在估算加速进程中与 SDSoC 环境的 GUI 连接。图 5 中报告的所有数字都在运行执行过程中用这些计数器测量得到。唯一的例外是硬件加速函数，该函数直到整个 FPGA 构建（包括布局布线）完成后才存在。不过，Vivado HLS 会在有效 Vivado HLS 的综合步骤过程中在估算资源利用率的同时计算硬件加速函数的估算周期数。

假设备选硬件加速器函数以 FHW MHz 时钟频率运行，并需要 CKHW 个时钟周期完成整个计算（这是时延概念），并且假设在 ARM CPU 执行时，函数在 FARM MHz 时钟频率下占用 CKARM 个周期，那么，如果计算时间相同，硬件加速器就能实现与 ARM CPU 相同的性能，即 CKHW / FHW= CKARM / FARM。从这个公式中我们得到 CKARM =CKHW*FARM / FHW。这代表加速器能为处理器分担的最大时钟周期数量，以展示将函数迁移到硬件而获得的加速效果。图 6 中报告了 Vivado HLS 综合估算结果。注意，硬件加速器时延为 CKHW = 83,652 个周期，时钟频率 FHW = 100-MHz。在 ZC702 板中，FARM= 666 MHz，而且 CKARM = CKHW*FARM / FHW =83,653*666/100 = 557,128，获得的硬件加速效果与图 5 中 SDSoC 环境报告的 565,554 个周期达到了很好的匹配。这就是为什么 SDSoC 环境能估算加速器所需的时钟周期数量而又不需要进行实际的布局布线。

用 SDSOC 环境构建硬件-软件系统

在确定硬件加速有效果之后，我们可以用 SDSoC 环境实现整个硬件和软件系统。我们需要做的是添加正确的指令（以 Pragma 命令的形式）来分别指定 FIFO 接口（由于 I/O 阵列的连续扫描）；在运行时针对任何加速器调用而需传送的数据量；连接 PL 中的 IP 核与 PS 的 AXI 端口类型；以及数据移动工具的类型。下面的 C/C++ 代码表明了这些指令的使用。注意，实际情况下最后的指令不需要，原因是 SDSoC 环境会因 sds_alloc 的使用而实例化一个

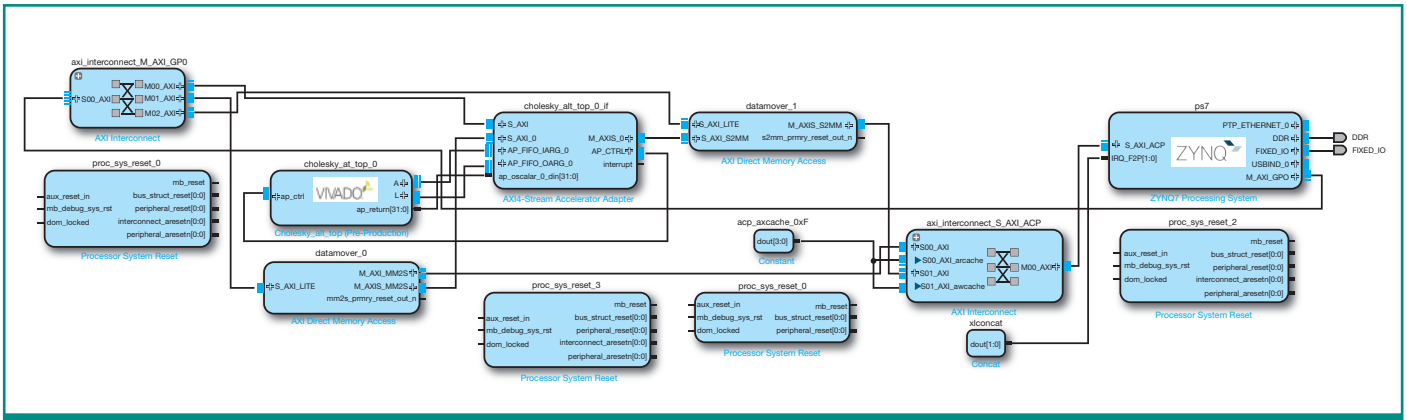


图 8 — 用 SDSoC 环境制作的 IP Integrator 模块化设计

Simple DMA；我们在这里提到它只是为了说清楚。

我们可以在 SDSoC 环境的 GUI 中直接在

```
#pragma SDS data access_pattern(A:SEQUENTIAL, L:SEQUENTIAL) //fifo interfaces
#pragma SDS data copy(A[0:BUF_SIZE], L[0:BUF_SIZE])         // amount of data transf
#pragma SDS data sys_port (A:ACP, L:ACP)                    // type of AXI ports
#pragma SDS data data_mover (A:AXI_DMA_SIMPLE, L:AXI_DMA_SIMPLE) // type of DMAs

int cholesky_alt_top(MATRIX_IN_T  A[ROWS_COLS_A*ROWS_COLS_A],
                     MATRIX_OUT_T L[ROWS_COLS_A*ROWS_COLS_A]);
```

Release 配置中构建项目，或者，也可使用图 7 中的 Makefile 并从 SDSoC 工具命令语言 (Tcl) 解释器中启动。不管是 Vivado Design Suite 中的什么工具，设计人员都可以采用 GUI 或 Tcl 脚本。为了提高加速效果，我们将硬件加速器的时钟频率增加至 FHW =142 MHz（通过 -clkid 1 makefile 标志设置）。

FPGA 编译完成后不到半个小时，我们就得到对 ZC702 板进行编程的比特流以及在 Linux OS 上执行的可执行连接文件 (ELF) 格式文件。然后，我们在 ZC702 板上测量性能：纯软件时是 995,592 个周期，有硬件加速时是 402,529 个周期。因此，cholesky_alt_top 函数的有效性能提升 2.47。

图 8 给出了整个嵌入式系统的方框图；SDSoC 环境以对用户透明的过程调用 Vivado IP Integrator（为了清楚起见，只显示了 AXI4 接口）。此外 SDSoC 环境将 Vivado IP Integrator 方框图以 HTML 文件报告的形式给出，以使其易于阅读（图 9）。该报告清楚显示，硬件加速器通过简单 AXI4-DMA 连接 ACP 端口，而通用端口用来通过 AXI4-Lite 接口设置加速器。

在嵌入式系统启动并运行时，我们需要花多长时间为 ZC702 板生成 SD 卡？我们需要一个工作日来编写适合 Vivado HLS 和 SDSoC 环境的 C++ 测试平台，然后，用一个小时进行实验以从 Linear Algebra HLS Library 中获得好的结果，再用一个小时的时间通过 SDSoC 环境创建嵌入式系统（FPGA 编译过程）。这个过程共需要 10 个小时。我们估算手动完成所有这些工作（步骤 3 用 Vivado IP Integrator，步骤 4 用赛灵思 SDK）至少需要两周的辛苦工作，这还不算高效使用这些工具所需的时间。

SDSoC 开发环境使更多嵌入式系统和软件开发

FPGA 编译完成后不到半个小时，我们就得到对 ZC702 板进行编程的比特流以及在 Linux OS 上执行的 ELF 格式文件。

人员能够凭借熟悉的嵌入式 C/C++ 开发经验针对 Zynq SoC 开展工作。包括业界首款 C/C++ 全系统优化编译器的 SDSoC 环境提供系统级特性分析、可编程逻辑中的自动软件加速、自动系统连接生成和库，以加快开发速度。如需了解更多详情以及了解如何获得该工具，敬请访问：*http://china.xilinx.com/products/design-tools/software-zone/sdsoc.html*. ■

参考资料

1. UG1165, Zynq-7000 All Programmable SoC：嵌入式设计教程
2. UG850, 针对 Zynq- 7000 XC7Z020 All Programmable SoC 的 ZC702 评估板用户指南
3. UG871, Vivado Design Suite 教程：高层次综合
4. UG948, Vivado Design Suite 教程：使用 System Generator 的模型化 DSP 设计
5. UG994, Vivado Design Suite 用户指南：使用 IP Integrator 设计 IP 子系统
6. UG782, 赛灵思软件开发套件 (SDK) 用户指南

数据移动网络

加速器	参数	IP端口	方向	声明尺寸（字节）	编译指示	连接
cholesky_alt_top_0	A	A	进	4096*4	• 长度: (BUF_SIZE) • sys_port:ACP	S_AXI_ACP:AXIDMA_SIMPLE
	L	L	OUT	4096*4	• 长度: (BUF_SIZE) • sys_port:ACP	S_AXI_ACP:AXIDMA_SIMPLE
	return	AP_return	OUT	4		M_AXI_GP0:AXILITE:0xC0

加速器调用点

加速器	调用点	IP 端口	发送尺寸（字节）	分页或连续 不可缓存	可缓存或
cholesky_alt_top_0	cholesky_alt_tb.cpp:246:23	A	(BUF_SIZE) * 4	连续	可连续
		L	(BUF_SIZE) * 4	连续	可连续
		ap_return	4	分页	可连续

图 9 - SDSoC 连接报告

编译、调试与优化

作者：
Jayashree Rangarajan
赛灵思公司交互设计工具部高级工程总监
jayr@xilinx.com

Fernando Martinez Vallina
赛灵思公司 SDAccel 产品部软件开发经理
vallina@xilinx.com

Vinay Singh
赛灵思公司 SDAccel 产品部高级产品营销经理
singhj@xilinx.com

赛灵思的 SDAccel 开发环境为 FPGA 提供软件应用设计流程。

赛灵思 FPGA 器件主要由可编程逻辑架构组成，能让应用设计人员利用空间和时间并行性，最大化算法性能或大型应用中关键内核的性能。位于这种架构核心的是由基于查找表的逻辑元、分布式存储器单元和乘法 - 累加单元构成的阵列。设计人员可以用不同方式组合这些元件，把逻辑实现在算法中，同时满足功耗、吞吐量和时延方面的设计目标要求。把 FPGA 架构元件组合为逻辑功能一直是硬件工程人员的工作范畴，这个过程更类似于汇编层面的编码，与现代软件设计实务相去甚远。虽然常用软件设计流程早已脱离汇编编码的层面，但由于 CPU 编译和 FPGA 编译之间内在的差异，FPGA 设计实践的发展步伐要缓慢一些。

在 CPU 和 GPU 领域，硬件是固定的，所有的程序均根据静态指令集架构 (ISA) 编译。虽然 CPU 和 GPU 之间的 ISA 不同，但基本的底层编译方法是相同的。这种相似性促使设计实践从手动汇编编码向使用 OpenCL™ C、C 和 C++ 等软件开发常用的编程语言进行编译、调试和优化的设计流程发展演化。

在 FPGA 设计方面，设计人员能够为运行特定工作负载开发自己的处理架构。FPGA 的一大重要优势就是能够根据特定系统需求定制架构，但这不利于 FPGA 应用开发中采用软件开发实践方法。

六年前赛灵思开始积极的研发工作，通过开发一种把直观的软件开发设计循环引入 FPGA 的开发环境，来打破这一障碍。面向 OpenCL C、C 和 C++ 的赛灵思 SDAccel™ 能够让 FPGA 器件的应用编译、调试与优化方式类似于 CPU 和 GPU 的流程，而且在用于数据中心应用加速时，可将性能功耗比提升高达 25 倍。

软件设计人员可以使用 SDAccel 开发环境开发和加速众多功能与应用。下面介绍如何使用 SDAccel 环境实现中值滤波器应用的编译、调试与优化设计循环。

中值滤波器

中值滤波器是一种常在图像处理中用于降噪目的的空间函数（图 1）。中值滤波器使用的算法用围绕中心像素的 3x3 像素窗口，根据所有相邻像素的中值计算中心像素值。这一计算公式为：

```
outputPixel[i][j] =
median(inputPixel[i-1][j-1], inputPix-
el[i-1][j], inputPixel[i-1][j+1],
    inputPixel[i][j-1],     inputPixel[i]
[j],     inputPixel[i][j+1],
    inputPixel[i+1][j-1], inputPixel[i+1]
[j], inputPixel[i+1][j+1]) ;
```

编译

运用 OpenCL C 等语言编写中值滤波器函数后，就进入开发的第一个阶段 —— 编译。在 CPU 或 GPU 上，编译是软件设计流程中的一个必要的也是必然的步骤。目标 ISA 是固定而公开的，编程人员只需关心可用的处理内核数量和算法的高速缓存缺失情况。FPGA 编译更像一个开放问题：在编译时目标 ISA 并不存在，逻辑资源还有待组合成处理架构，系统存储器架构也尚未定义。

SDAccel 开发环境中的编译器提供三项功能，可帮助编程人员应对如下这些挑战：自动在循环内的语句间和跨循环迭代提取并行性、根据对阵列的读写模式自动调用存储器架构、以及对给定 FPGA 器件内部基本逻辑元的类型和数量的架构感知能力。这三项功能对中值滤波器的源代码的重要性见图示（图 2）。

中值滤波器的运算可以表达为由一系列由两个部分组成的嵌套循环。第一个部分负责从名为“输入”的外部存储器中阵列取出数据，然后把值存储在本地阵列“RGB”中。算法第二个部分是一个围绕 getMedian 函数的“for”循环。getMedian 就是计算发生的位置。

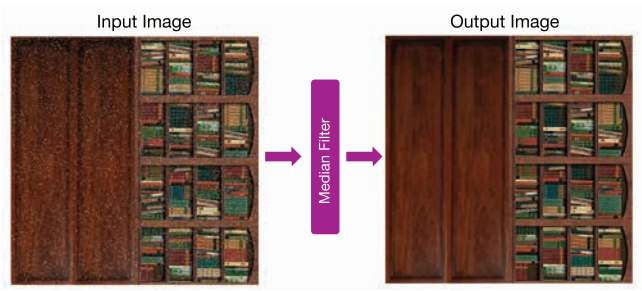


图 1 - 中值滤波器运算

通过分析图 2 中的代码，SDAcell 环境认为 RGB 阵列上不存在循环 - 承载数据依赖关系。每次循环迭代都有一个专有 RGB 副本，这个副本可以存储在不同物理资源上。从这个代码看出的 SDAccel 环境的另一大特点是对 getMedian 函数调用的独立性。

```
for (int y=0; y < height; y++) {
    int offset = y * width;
    int prev = offset - width;
    int next = offset + width;

    for (int x=0; x < width; x++) {
      // Get pixels within 3x3 aperture
      uint rgb[SIZE];
      rgb[0] = input[prev + x - 1];
      rgb[1] = input[prev + x];
      rgb[2] = input[prev + x + 1];

      rgb[3] = input[offset + x - 1];
      rgb[4] = input[offset + x];
      rgb[5] = input[offset + x + 1];

      rgb[6] = input[next + x - 1];
      rgb[7] = input[next + x];
      rgb[8] = input[next + x + 1];

      uint result = 0;

      // Iterate over all color channels
      for (int channel = 0; channel < 3;
      channel++) {
        result |= getMedian(channel,
rgb);
      }

      // Store result into memory
      output[offset + x] = result;
    }
```

图 2 - 中值代码

图 2 所示的算法版本在带有固定边界的“for”循环内部执行 getMedian 函数。根据滤波器的性能目标和所选的 FPGA，SDAccel 环境可以跨所有三个通道重复利用计算资源，或是分配更多逻辑，实现空间并行性，同时运行所有通道。这项决定相应地会影响如何为阵列 RGB 实现存储器存储。

从应用编程人员的角度出发，上述步骤是透明的，可视为 GNU 编译器套件 (GCC) 中的 –O1 到 –O3 优化。

SDAccel 环境中的 printf 设计无需耗用额外逻辑资源即可提供此项功能。

调试

软件开发的一条公理就是应用编译不等同于应用正确。只有在应用在目标硬件上开始运行之后，编程人员才能开始发现、追踪和纠正应用中的错误，换句话说进行调试。

CPU 应用的调试是一个容易理解的问题，商业厂商和开源社区提供有大量工具来解决这个问题。FPGA 再度成为另类。应用编程人员如何才能调试好创建用于实现一段代码的功能的软件，使之实现既定性能目标呢？

SDAccel 开发环境借鉴了 CPU 环境的两个概念 —— printf 和 GDB 调试 —— 来解决这个问题。

printf 功能是软件编程人员工具箱中的基本工具。每一种编程语言都会提供 printf 功能，用于显示程序执行过程中主要应用变量的状态。对 CPU 器件而言，这就同监测寄存器状态一样简单。printf 功能不占用硬件资源。

就 FPGA 的情况来说，实现 printf 可能会消耗本可用于实现逻辑功能的逻辑资源。在 SDAccel 环境中实现 printf 无需消耗额外的逻辑资源即可提供此项功能。该环境是通过将 printf 数据生成独立于解码和用户表示层来实现的。就硬件资源而言，为 printf 生成数据会占用几个寄存器，在寄存器丰富的 FPGA 架构中这完全可以忽略不计。数据解码在针对 FPGA 的驱动程序中完成。通过让主机 CPU 执行 printf 的数据解码和表示层，软件编程人员可以在基本不占用 FPGA 资源的情况下使用 printf。

从 CPU 借鉴的第二个调试技巧是使用 GNU 项目调试器 (GDB) 等工具，在代码中各处插入断点和单步执行。编程人员可以使用 SDAccel 环境的仿真模式，将 GDB 连接到正在运行的仿真流程。该仿真流程是对开发人员准备在 FPGA 器件上执行的专用硬件的仿真。在仿真流程环境下，GDB 能够观察各种变量的状态，在代码各处插入断点和单步执行。从应用编程人员的角度，这与 GDB 在 CPU 上工作的方式完全相同。

优化

完成编译和调试后，软件开发周期的下一步是优化应用。在 FPGA 上进行应用优化的原理与在 CPU 上优化应用一致。唯一差别在于方法不同。对 CPU 来说，需要细调应用代码，使之适应处理器的高速缓存和算术单元的边界。在 FPGA 中，计算逻辑是为当前应用定制汇编的。因此决定优化约束的

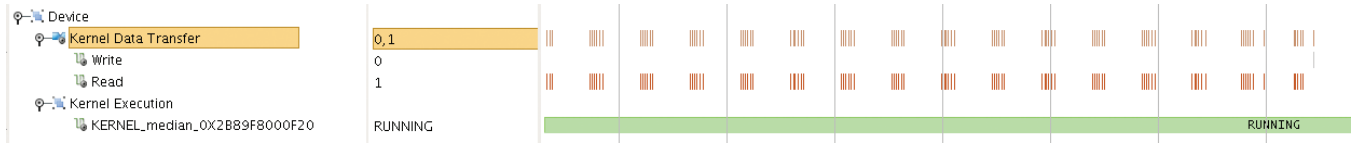


图 3 - 存储器访问事务处理跟踪

使用 SDAccel 的软件编程人员可以借助逻辑架构的灵活性，不必了解有关硬件设计的细节，即可构建高性能、低功耗应用。

是 FPGA 的大小和应用的目标性能。

SDAccel 环境的编译器能自动优化计算逻辑。编程人员可通过对从代码调用的数据传输模式的分析来协助自动优化工作。图 3 所示的是输入和输出过程中发生的中值滤波器代码与存储器之间的读写事务处理。

图中的每条垂直线代表一个到存储器的事务处理。绿条所示的是中值滤波器函数的活动时间。从图中可以看出，虽然中值滤波器一直处于活动状态，存储器事务处理之间存在明显的间隔。这些间隔代表的是中值滤波器从一个事务处理切换到下一个事务处理所使用的时间。由于每项对存储器的事务处理只访问一个值，事务处理之间的间隔说明该应用存在严重的性能瓶颈。

解决该性能问题的途径之一是在应用代码中明示地声明从外部存储器到本地存储器的突发事务处理。图 4 所示的代码片段使用了 OpenCL C 语言内核采用的 async_work_group_copy 函数。这项功能的目的是通知编译器每项对存储器的事务处理都是一个包含多重值的突发事件。这样能够更高效率地利用目标器件上的可用存储器带宽，并减少对存储器的事务处理总数。在图 4 的代码中，async_work_group_copy 函数把 DDR 存储器中输入图像的全部行内容提交给内核数据路径内的存储器。图 5 中的存储器事务处理迹线体现了使用 async_work_group_copy 的结果。如图 5 所示，在存储器事务处理发生之前内核有一个建立时间，但这并没有体现在中值滤波器的原始代码中（图 2）。

```
  for (int line = 0; line < height; line++)
{
      local uint linebuf0[MAX_WIDTH];
      local uint linebuf1[MAX_WIDTH];
      local uint linebuf2[MAX_WIDTH];
      local uint lineres[MAX_WIDTH];
      // Fetch Lines
      if (line == 0) {
        async_work_group_copy(linebuf0,
        input, width, 0);
        async_work_group_copy(linebuf1,
        input, width, 0);
        async_work_group_copy(linebuf2,
        input + width, width, 0);
      }
  …
  }
```

图 4 - 使用明示突发存储器传输的中值滤波器代码

这部分建立时间差异用于处理从代码中衍生出的逻辑。在图 2 的原始代码中，应用立即启动对存储器的单次事务处理，然后等待数据可用。与此相反，图 4 中的优化代码能判断是否需要进行存储器事务处理或是否内核的本地存储器中的数据已经可用。它还能让生成的逻辑背对背地调度存储器事务处理和覆盖读写事务处理。

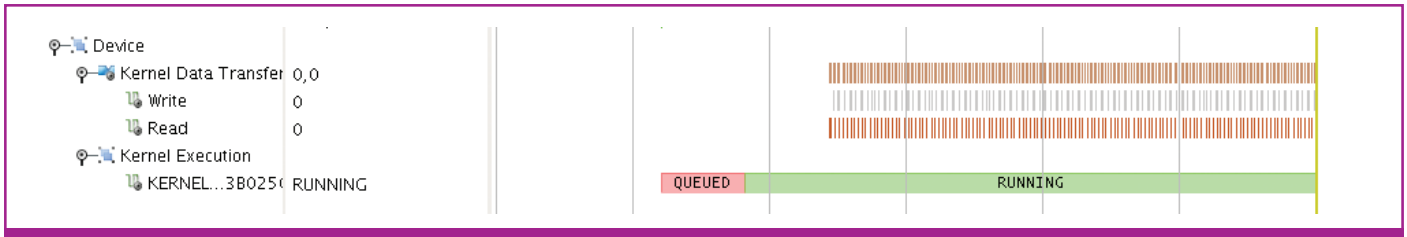


图 5 - 代码优化后的存储器访问事务处理迹线

不管最终器件是 CPU 还是 FPGA，特性分析都是应用开发过程中的主要组成部分。SDAccel 环境的可视化和分析器功能能让应用编程人员从内核占用、存储器事务处理和存储器带宽利用率等方面对代码修改和应用要求进行特征描述。

由编译、调试和优化操作建立的设计环路是软件开发流程的基础。SDAccel 开发环境使用类似于 CPU 开发环境的工具和技巧来实现这一设计环路，同时基于 FPGA 的应用加速可将性能功耗比提升 25 倍，时延则缩短 50 至 75 倍。使用 SDAccel，软件编程人员无需了解有关硬件设计的所有细节，就能够充分发挥 FPGA 逻辑架构的灵活性，从而开发出高性能、低功耗应用。■

GET A DAILY DOSE OF XCELL

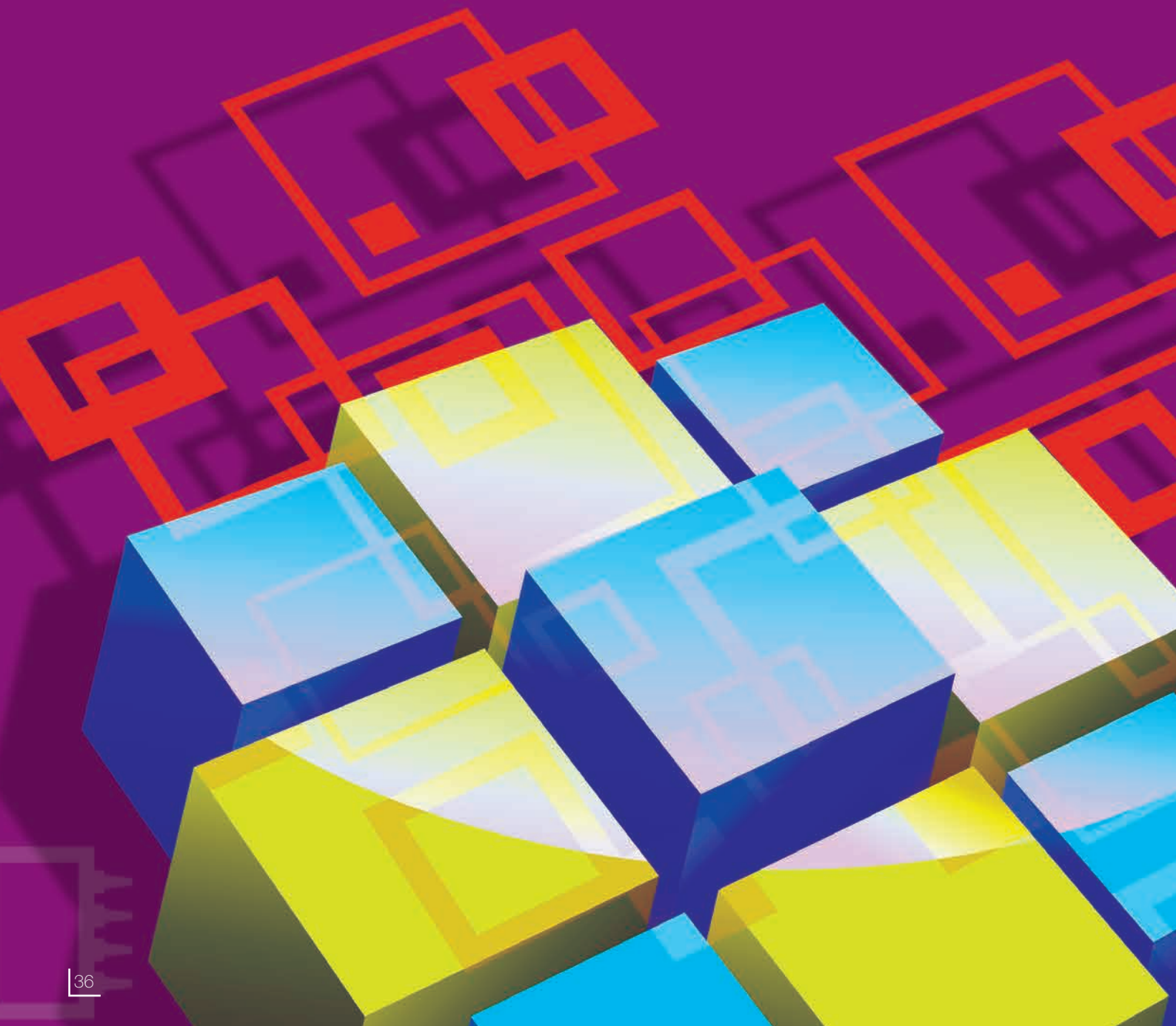
Xilinx has extended its award-winning journal and added an exciting new Xcell Daily Blog.
The new site provides dedicated readers with a frequent flow of content to help engineers and developers leverage the flexiblility and extensive capabilities of all Xilinx products and ecosystems.

What's Recent:

- Half Wheelchair, Half Segway, Half Battlebot: Unprecedented mobility for the disabled—controlled by Zynq
- Regular Universal Electronic Control Unit tester for vehicles up and running in two months thanks to NI LabVIEW and LabVIEW FPGA
- Radar looks deep into Arctic snow and ice to help develop sea-level climate models
- Passive, Wi-Fi radar that sees people through walls prototyped with NI LabVIEW and two FPGA-based USRP-2921 SDR platforms
- 500-FPGA Seismic Supercomputer performs real-time acoustic measurements on its heart of stone to simulate earthquakes

XILINX
ALL PROGRAMMABLE.

使用 C++ 库开发 OpenCL 图像应用



作者：Stephen Neuendorffer，
赛灵思公司 Vivado HLS 首席工程师
stephenn@xilinx.com

Thomas Li
赛灵思公司 Vivado HLS 软件工程师
thl@xilinx.com

Fernando Martinez Vallina
赛灵思公司 SDAccel 开发经理
vallina@xilinx.com

赛灵思 SDAccel 开发环境借助现有库之力加快应用设计速度。

随着线上图片与视频、机器人和驾驶员辅助应用的快速推广，近年来成像应用的规模和普及率均在提升。这些领域的核心算法非常相似，都需要一种能够让应用开发人员根据市场和部署目标迅速实现产品重构与差异化的开发方法。

针对这样的需求，成像应用一般先从针对 CPU 的软件程序开始，使用库调用功能来调用标准函数。软件设计方法与随时可用的库相结合，不仅可以轻松启动开发工作，而且还容易在桌面上开发出功能正确的应用。

对开发人员而言，挑战在于针对执行对象优化成像应用。赛灵思 SDAccel™ 开发环境借助赛灵思 Vivado® HLS 技术，让为 FPGA 开发 OpenCL™ 应用的开发人员能够方便地使用 C++ 库。

一组并行计算任务

成像应用的关键特征之一是它们基本上是相对于周边相邻像素在空间上（对某些应用在时间上）在某个像素上开展的一组运算。因此我们可以把成像应用视为开发人员能在 CPU、GPU 或 FPGA 上执行的一组并行计算任务。

CPU 一直是应用启动最方便的目标器件。在考虑优化之前，代码一般已经在 CPU 上运行，因此可以运用丰富的现有库。在 CPU 上执行成像工作负载的问题在于可实现的持续性能。总体性能受缓存命中率 / 未命中率，以及将并行化转化为运

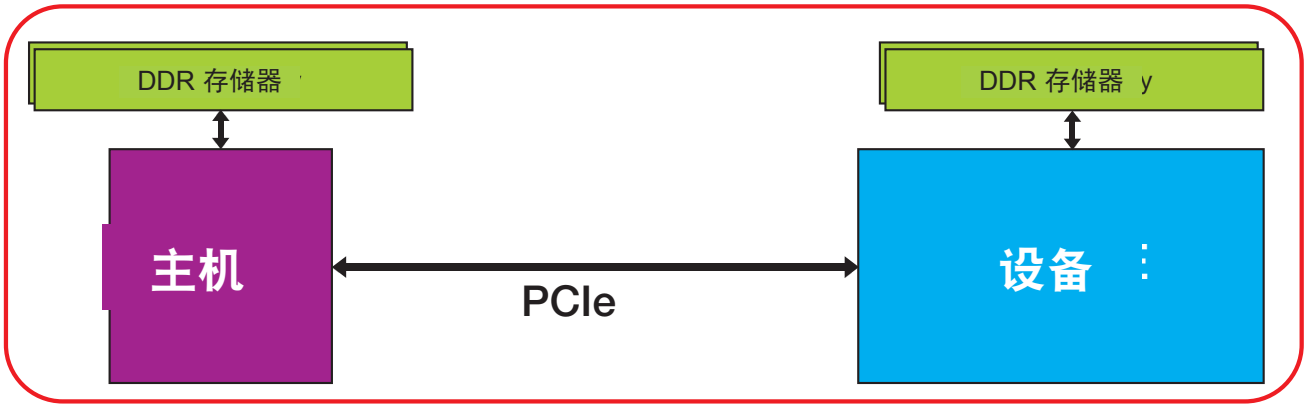


图 1 - 基本的 OpenCL 平台包含一个主机和至少一台设备。

行在多个 CPU 内核上的多个线程这个重要任务的制约。

在成像应用方面，GPU 有望提供远超 GPU 的性能，因为 GPU 的硬件是专门为成像工作负载构建的。直到最近几年，GPU 的不足之处就是在在一般成像应用方面一直采用编程模型。GPU 编程与 CPU 编程的区别在于 GPU 模型不能在 GPU 器件系列之间移植。随着针对采用 OpenCL 框架的 GPU 等并行系统的编程的标准化，这种情况已经发生变化。

FPGA 为成像工作负载提供了一种可选实现方法。开发人员可以将 FPGA 逻辑架构定制为工作负载专用线路。FPGA 架构的灵活性让应用开发人员在充分发挥定制逻辑的性能和功耗优势的同时，还能避免 ASIC 设计带来的成本和工作量。

和 GPU 一样，采用 FPGA 器件的障碍之一也是编程模型。传统上 FPGA 一直采用 Verilog 或 VHDL 等寄存器传输语言 (RTL) 编程。虽然这些语言能够表达并行性，但其精细度远低于 CPU 或 GPU 编程所要求的水平。但就和 GPU 一样，采用 OpenCL 标准，用软件编程人员熟悉的方式表达 FPGA 编程，已经克服编程模型障碍。

OpenCL 框架

OpenCL 框架为表达数据并行程序提供了一个通用编程模型。该框架已发展成为行业标准，其采用所有支持 OpenCL 的器件厂商通用的平台和存储器模型。一个器件可以定义为任何能够执行 OpenCL 内核的硬件，不管是 CPU、GPU 还是 FPGA。

OpenCL 应用内部的平台负责定义执行应用的硬件环境。图 1 显示了构成 OpenCL 平台的主要元件。

OpenCL 平台常常包含一个一般实现在处理器上的主机。该主机负责在器件上发起任务和负责明示地协调主机和器件之间的全部数据传输。

除主机外，该平台还包含至少一台设备。OpenCL 平台中的设备是能够执行 OpenCL 内核代码的硬件元件。在 OpenCL 应用环境中，内核代码是算法中需要加速的高计算强度部分。

在 CPU 和 GPU 器件上，内核代码在器件中的一个或多个内核上执行。每个内核从器件规格来说完全相同。这种严格性迫使应用开发人员修改代码以最大限度地提高固定架构的性能。

与此相反，对 FPGA 而言，SDAccel 开发环境能根据应用内核的具体计算要求生成定制内核。因

对 FPGA 而言，SDAccel 开发环境能根据应用内核的具体计算要求生成定制内核。

此应用开发人员能根据算法需要自由地探索实现架构，以降低整体系统时延和功耗。

第二个 OpenCL 组件是存储器模型（图 2）。该模型对所有厂商是通用的，用于定义统一的存储器层级，以便开发人员能够创建可移植应用。该存储器模型的主要组件有主机存储器、全局存储器、本地存储器和专用存储器。主机存储器指只能供主机处理器访问的内存空间。FPGA（设备）可见的存储器分为全局存储器、本地存储器和专用存储器。

全局内存空间一般实现在 FPGA 连接的 DDR 中，主机和设备都能访问。根据加速板上使用的具体 FPGA 不同，部分全局存储器还可实现在 FPGA 架构内部。本地存储器和专用存储器仅对在 FPGA 架构内部执行的内核可见，使用 Block RAM（BRAM）和寄存器资源完全内置在架构内。

下文将介绍 SDAccel 环境利用 OpenCL 和 C++ 库实现立体成像模块匹配应用。

立体模块匹配

立体模块匹配功能使用来自两个摄像头的图像建立位于两个摄像头视野内的物体的形状表达。如图 3 所示，该算法使用左右摄像头输入的图像搜索

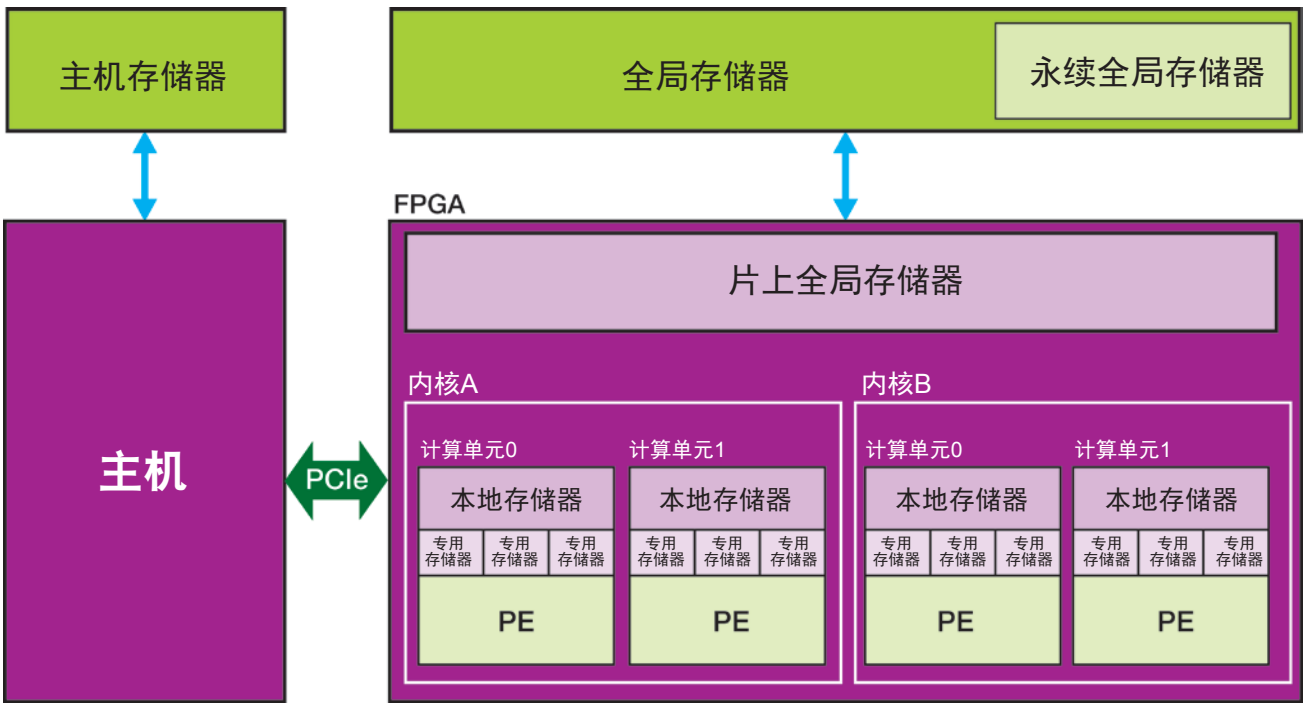


图 2 – OpenCL 存储器模型为应用开发定义统一的存储器层级。

SDAccel 开发环境将赛灵思 Vivado HLS C-to-RTL 编译器技术用作核心内核编译器，让 SDAccel 环境能够像使用用 OpenCL C 语言表达的内核一样使用用 C 语言和 C++ 语言表达的内核。

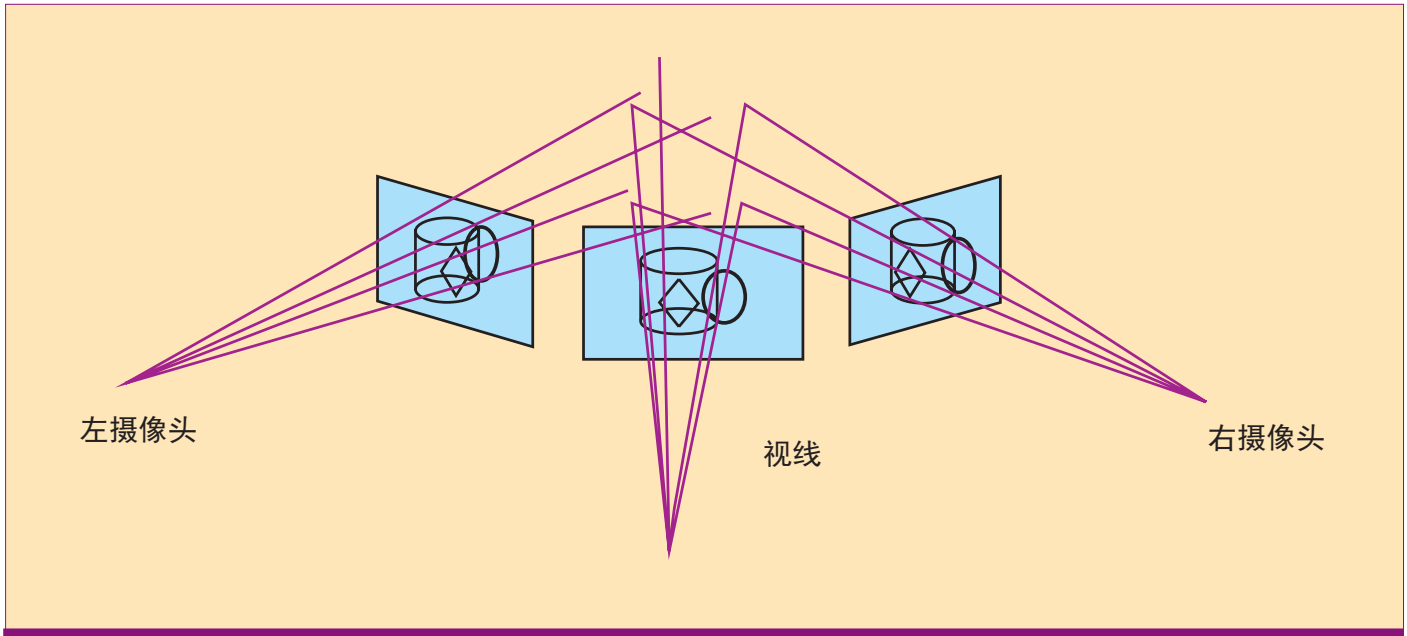


图 3 - 多摄像头处理概念图

图像间的对应性。这种多摄像头图像处理任务可应用于深度图、图像分割和前景 / 背景分离。举例而言，这些均属于驾驶员辅助系统中行人检测应用不可分割的组成部分。

将C++ 库用于视频。

SDAccel 开发环境将赛灵思 Vivado HLC C-to-RTL 编译器技术用作核心内核编译器的一部分，让 SDAccel 环境能够像使用用 OpenCL C 表达的内核一样使用用 C 和 C++ 表达的内核。因此应用开发人员就能使用之前在 Vivado HLS 中优化过的 C++ 库和代码提高生产力。立体模块匹配应用的主代码见下一页。

Vivado HLS 提供基于常用的 OpenCV 框架的图像处理函数。这些函数用 C++ 编写，并且为在 FPGA 中提供高性能进行了优化。在综合到 FPGA 实现方案中后，相当于每个时钟周期能同时执行数十条乃至数千条 RISC 处理器指令。

应用代码使用 Vivado HLS 视频处理函数创建该应用。该应用代码包含对 Vivado HLS 库的 C++ 函数调用和编译指示，以引导编译过程。编译指示分为用于接口定义的编译指示和用于性能优化的编译指示。

接口定义编译指示决定立体模块匹配加速器与系统其余部分的连接方式。由于该加速器用 C++ 而非 OpenCL 语言代码表达，因此应用编程人员必须

```
void stereobm(
        unsigned short img_data_lr[MAX_HEIGHT*MAX_WIDTH],
        unsigned char img_data_d[MAX_HEIGHT*MAX_WIDTH],
        int rows,
        int cols)
{
#pragma HLS INTERFACE m_axi port=img_data_lr offset=slave bundle=gmem0
#pragma HLS INTERFACE m_axi port=img_data_d offset=slave bundle=gmem1
#pragma HLS INTERFACE s_axilite port=img_data_lr bundle=control
#pragma HLS INTERFACE s_axilite port=img_data_d bundle=control
#pragma HLS INTERFACE s_axilite port=rows bundle=control
#pragma HLS INTERFACE s_axilite port=cols bundle=control
#pragma HLS INTERFACE s_axilite port=return bundle=control

    hls::Mat<MAX_HEIGHT, MAX_WIDTH, HLS_8UC2>  img_lr(rows, cols);
    hls::Mat<MAX_HEIGHT, MAX_WIDTH, HLS_8UC1>  img_l(rows, cols);
    hls::Mat<MAX_HEIGHT, MAX_WIDTH, HLS_8UC1>  img_r(rows, cols);
    hls::Mat<MAX_HEIGHT, MAX_WIDTH, HLS_16SC1> img_disp(rows, cols);
    hls::Mat<MAX_HEIGHT, MAX_WIDTH, HLS_8UC1>  img_d(rows, cols);

    hls::StereoBMState<15, 32, 32> state;

#pragma HLS dataflow
    hls::AXIM2Mat<MAX_WIDTH>(img_data_lr, img_lr);
    hls::Split(img_lr, img_l, img_r);
    hls::FindStereoCorrespondenceBM(img_l, img_r, img_disp, state);
    hls::SaveAsGray(img_disp, img_d);
    hls::Mat2AXIM<MAX_WIDTH>(img_d, img_data_d);
}
```

提供用于匹配 SDAccel 环境中 OpenCL 模型的推测的编译指示。

标识“m_axi”的编程指示用于声明缓存的内容将存储在设备的全局存储器中。标识“s_axilite”的编程指示则用于供加速器从主机接收全局存储器缓存中的基地址。

本代码中的性能优化编译指示是数据流。数据流编译指示生成的是一个加速器，其中的不同子函数也可同步执行。

在本加速器中由于底层实现的是“hls::Mat”数据类型，数据也在每项函数之间传递。这样一来只要 Split 函数生成像素，FindStereoCorrespondenceBM 函数就开始工作，无需等待生成完整的图像。这样可得到一种更高效的架构，与在各函数间等待完整帧缓存再顺序执行各函数相比，处理时延明显下降。

成像应用是一种计算密集型应用，需要用到大量现成的库。其挑战在于针对执行目标优化应用。SDAccel 环境让开发人员能够充分发挥 C++ 库的作用，为采用 OpenCL 编程的 FPGA 加快成像应用的开发速度。■

MATLAB 和 Simulink 协助 Zynq SoC 软硬件协同设计

作者：Eric Cigan 和 Noam Levine，
MathWorks 公司 FPGA/SoC
技术市场营销部

基于模型的设计工作流程让工程师不必进入实验室，在桌面上就能完成设计权衡。

2011 年赛灵思 Zynq®-7000 All Programmable SoC 的推出为 FPGA 行业带来了突破性创新。这些器件通过将双核 ARM® Cortex™-A9MPCore™ 处理器和丰富的可编程逻辑完美结合在一起，为大量应用提供了诸多优势。采用 Zynq SoC，设计人员既可享受在行业最常用的处理器上开发软件应用的好处，还能获得通过高速可编程逻辑架构上的硬件加速所提供的灵活性和吞吐量。

借助 MathWorks® 提供的 MATLAB® 和 Simulink®，今天的创新者能够运用高度集成的软硬件工作流程创建高度优化的系统。本文提供的案例研究将讲述这一基于模型的工作流程。

赛灵思于 2011 年 12 月推出业界首款 Zynq SoC 时，设计人员就意识到他们可以将其传统的用分立处理器和 FPGA 构建的多芯片解决方案移植到单芯片平台上。他们可以在新平台上创建基于 FPGA 的加速器，扫除软件执行障碍，利用赛灵思及其 IP 合作伙伴提供的一系列的现成的生产就绪型 IP，满足数字信号处理、网络、通信等应用需求。

尚待解决的问题是他们如何为这款新器件编程。憧憬软硬件联合设计潜力的设计人员寻求的是能够在 ARM 处理器和可编程逻辑间对设计进行智能分区的集成工作流程。但他们找到的却是各自为阵的软硬件工作流程：一面是针对 ARM 内核的常规嵌入式软件开发流程；另一面是针对可编程逻辑的 IP 组装、传统 RTL 和新兴高层次综合工具组合在一起的硬件工作流程。

集成工作流程

2013 年 9 月，MathWorks 使用基于模型的设计推出了一种 Zynq-7000 SoC 软硬件工作流程。在这个工作流程（图 1）中，设计人员可在 Simulink 中创建用于表达完整动态系统的模型（含用于针对 Zynq SoC 的算法的 Simulink 模型），还可直接从算法迅速创建针对 Zynq SoC 的软硬件设计。

系统设计人员和算法开发人员在 Simulink 中使用仿真为完整系统（通信、机电组件等）创建模型，以评估设计概念、进行高层权衡并将算法分配给软硬件。从 Simulink 生成 HDL 代码便于在 Zynq SoC 架构上创建 IP 核和高速 I/O 处理。从 Simulink 生成 C/C++ 代码便于为 Zynq SoC 的 Cortex-A9 内核编程，从而支持快速嵌入式软件迭代。

这种方法能够自动生成链接 ARM 处理系统和可编程逻辑的 AMBA® AXI4 接口，为 Zynq SoC 提供支持。通过与 C/C++ 编译、为 ARM 处理系统构建可执行文件、使用赛灵思设计工具生成比特流、下载到 Zynq 开发板等下游任务集成，实现了高速原型设计流程。

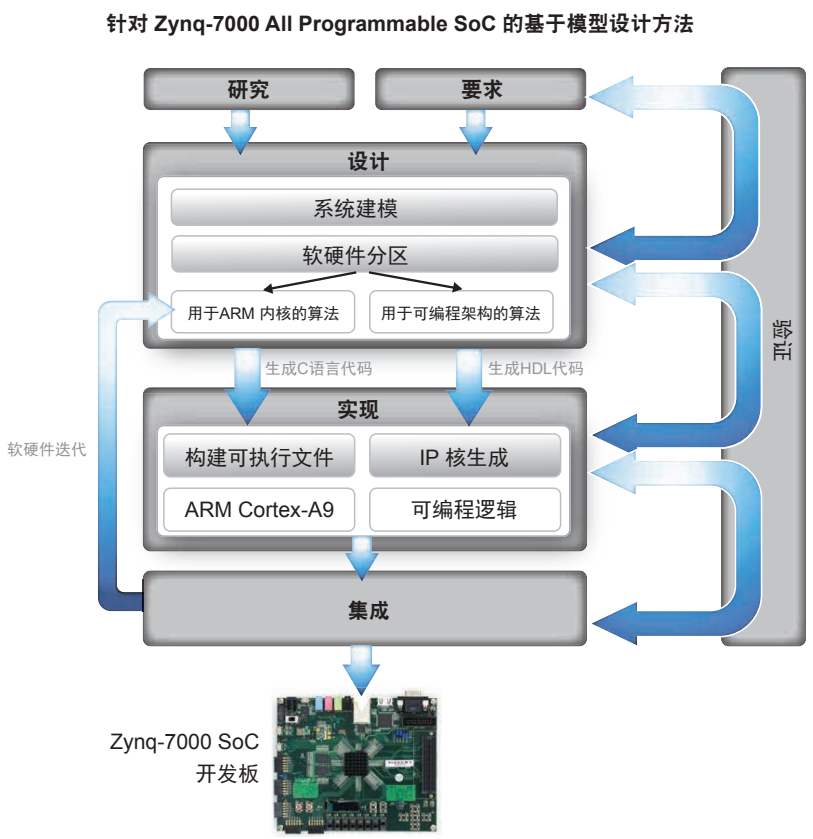


图 1 - 设计人员能够在 Simulink 中创建表达完整动态系统的模型，然后从模型直接为 Zynq 创建软硬件实现方案。

该工作流程的两大核心分别是：Embedded Coder® 和 HDL Coder™ 技术。Embedded Coder 负责从 MATLAB、Simulink 和 Stateflow 生成量产质量级的 C 和 C++ 代码，并对嵌入式系统进行针对性的优化。Embedded Coder 的应用已经相当广泛，当您驾驶现代乘用车、乘坐高速列车或搭乘商业航班时，这些交通工具极有可能就处于 Embedded Coder 实时生成的代码的控制之下。HDL Coder 与 Embedded Coder 对应，负责为 FPGA 和 ASIC 生成 VHDL 或 Verilog 并已紧密集成到赛灵思工作流程中。这种成熟的 C 和 HDL 代码生成技术是可编程 SoC 的基于模型的设计工作流程的基础。在通信、图像处理、智能电源和电机控制等领域中使用基于模型的设计的设计团队已经采用这一工作流程。

它是算法开发人员与硬件设计人员和嵌入式开发人员密切协作，加快算法在可编程 SoC 上实现的途径。当生成的 HDL 和 C 语言代码在硬件中完成原型设计，设计团队就可以使用赛灵思 Vivado® IP 集成器把该代码与其他生产所需的设计组件集成在一起。

案例研究：三相电机控制

出于多种原因，具有高效电源转换能力的定制电机控制器是可编程 SoC 领域最常见的应用之一。更高性能、更高效率的方案是一个因素。在电机驱动系统占全球耗电总量 46% 的情况下，运用新颖控制算法实现更高效率是一个电机驱动设计追求的永无止境的共同目标。赛灵思 Zynq 可编程逻辑能实现精确的时序，是设计低时延高效率驱动器的理想平台。

另一个促进因素是多轴向控制。可编程 SoC 上丰富的可编程逻辑和 DSP 资源为在单个可编程 SoC 上实现多个电机控制器开辟了无限可能。不论电机是单独运行还是组合运行，都可受控于集成动作控制系统之下。

工业网络 IP 集成是又一大因素。赛灵思及其 IP 合作伙伴提供的 IP 用于与 EtherCAT、PROFINET 和其他能方便地集成到可编程 SoC 中的工业网络协议

由于电机驱动系统占全球耗电量的 46%，用新颖的控制算法取得更高效率是电机驱动设计追求的永无止境的共同目标。

集成。

为说明这一工作流程在通用电机控制案例上的运用，将以实现在 Zynq-7020 SoC 上的三相电机磁场定向控制算法为例（有关该硬件原型设计平台的详情，请访问：http://www.mathworks.com/zidk）。该电机控制系统模型包含两个主要的子系统（图 2）：一个是针对已经在 Zynq 处理系统和可编程逻辑间完成分区的 Zynq SoC 的电机控制器；另一个是连接到配有测量轴角度的编码器的无刷直流电机的电机控制 FPGA 夹层卡 (FMC)。

可以从数据流的角度观察软硬件分区：

- 速度控制 (Velocity Control) 模块和模式选择 (Mode Select) 模块被分配给 ARM Cortex-A9 处理系统，因为这两个模块与模型的其余部分相比运行在较低速度下，也因为它们是设计在开发过程中最有可能修改和重新编译的部分。
- 运行在 ARM 内核上的模式选择 (Mode Select) 状态机负责判断电机控制器的运行模式（开环运行还是闭环调节）。该状态机负责管理在切换到闭环控制模式之前的启动、开环控制和编码器校准模式之间的切换工作。
- 编码器传感器信号通过外部端口传递给可编程逻辑中的编码器外设 (Encoder Peripheral)，然后传递给位置 / 速度估算 (Position/Velocity Estimate) 模块，计算电机的状态（轴位置和速度）。
- 一个 Σ Δ ADC 感知电机电流，随即由手动编码 ADC 外设模块处理该电流。

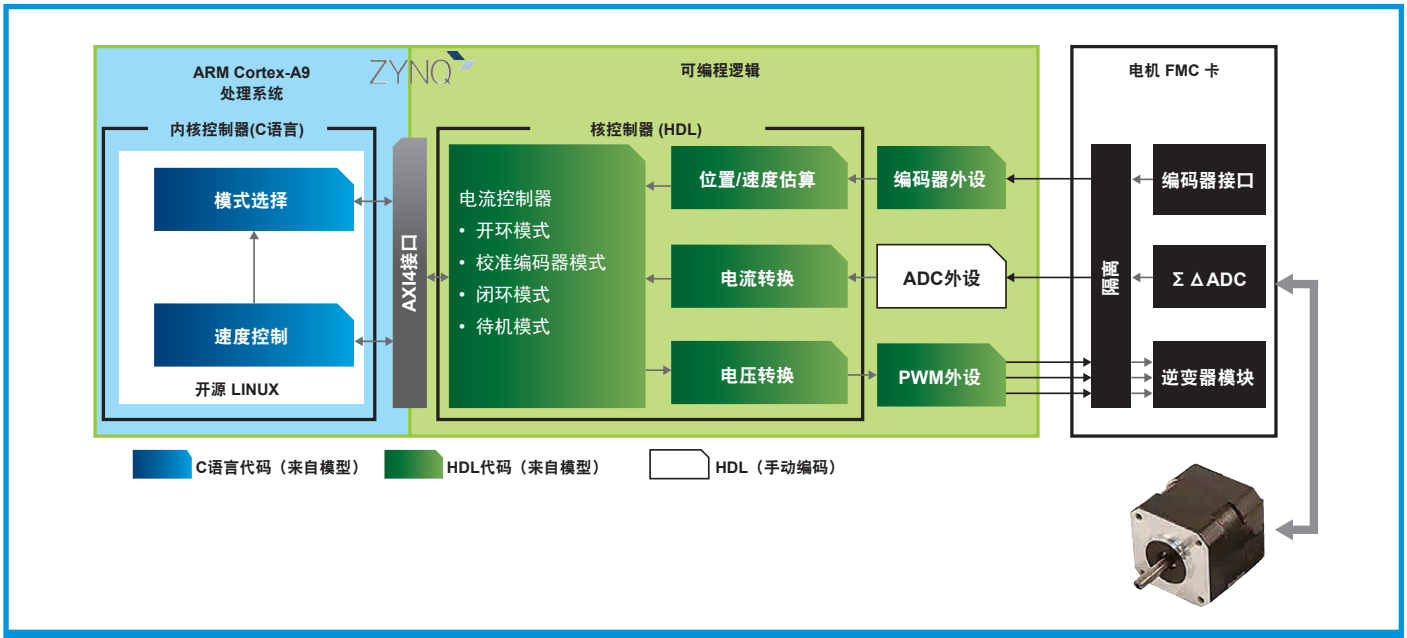


图 2 - 电机控制系统模型包含两个主要的子系统。

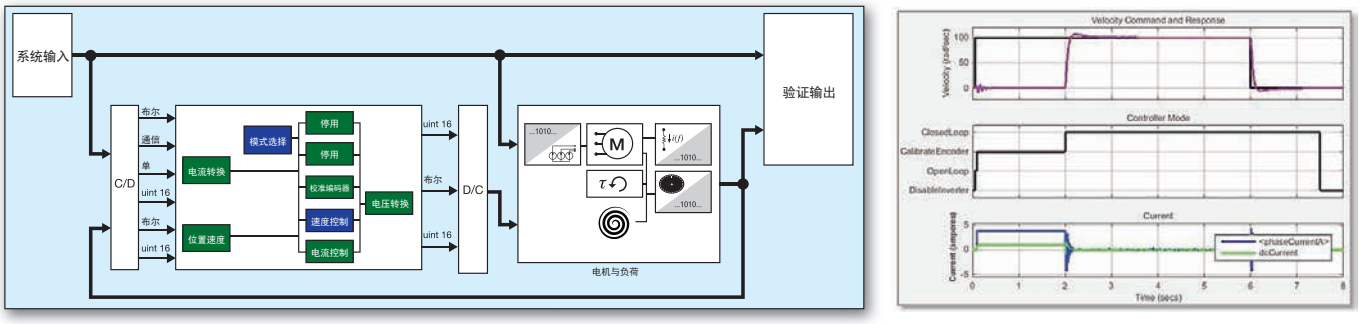


图 3 - 用于电机控制系统的控制环路模型，仿真结果体现的是对速度脉冲命令的响应。

- 电流控制器负责获取电机状态与电流、工作状态以及 ARM 内核通过 AXI4 接口传递的速度控制命令，据此计算电流控制器命令。在处于闭环模式下时，电流控制器使用比例积分 (PI) 控制原理，其增益可使用仿真和原型设计加以微调。
- 电流控制器命令穿越电压转换模块，通过 PWM 外设输出给电机控制 FMC，最终用于驱动电机。

设计人员能在 Simulink 中为完整系统建立模型（图 3）。

在基于模型的设计中，系统的顶层 Simulink 模型的组件数量增加到四个：

- 输入模型：负责向控制器提供受控轴速度和开 / 关命令，用作激励信号；
- 电机控制算法模型：主要用于 Zynq SoC；
- 设备模型：包括 FMC 驱动电子电路、无刷直流电机的永磁同步电机 (PMSM) 模型、电机轴上的惯性负载模型和编码器传感器模型；以及
- 输出验证模型：包含后处理和图形功能，有助于算法开发人员优化和验证模型。

在 Simulink 中可以在远早于开始硬件测试的时候就借助仿真彻底检验该算法。可以细调 PI 控制器的增益，尝试多种激励配置，检验不同处理速度的结果。在使用仿真的时候需要面对一个根本性问题：由于电机控制中一般存在着好几种千差万别的速度，即 1-10Hz 的总体机械响应速度、1-25 KHz 的内核控制器算法速度、10-50 MHz 的可编程逻辑运行速度，仿真运行时间从数分钟到数小时不等。我们可以借助一种对外设（PWM、电流感应和编码器处理）使用行为模型的控制环路模型解决这一问题，

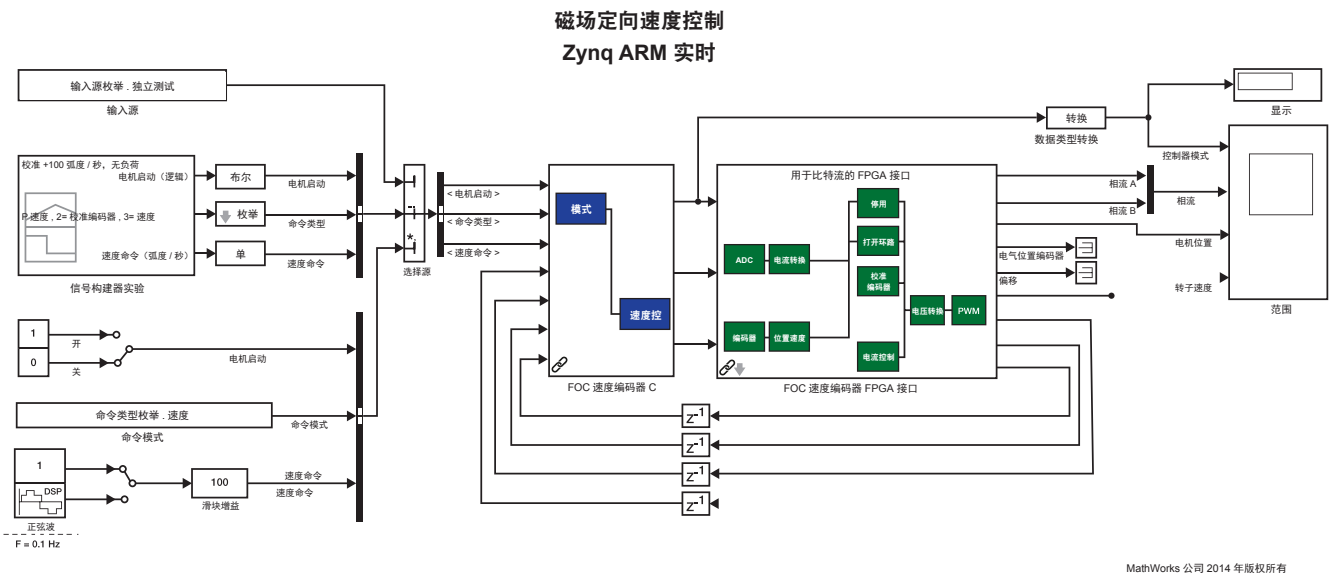
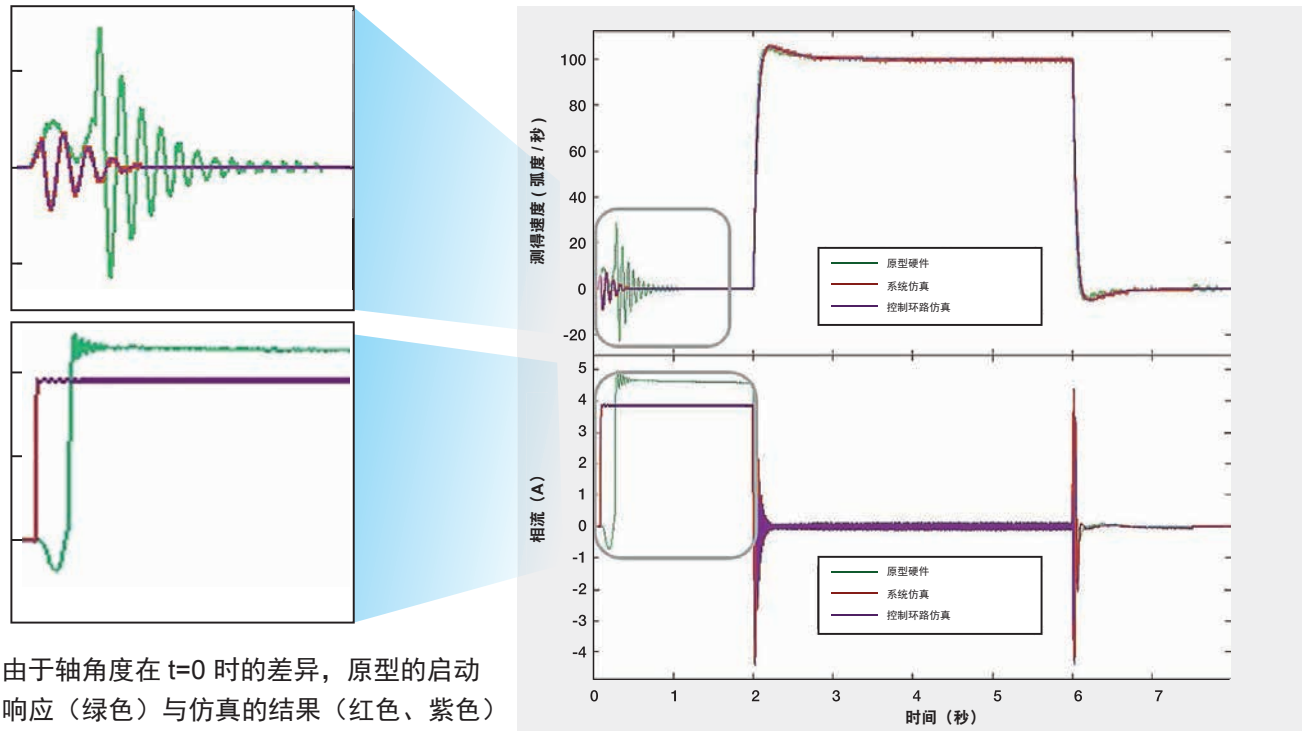


图 4（a） - 用于测试原型硬件的 Simulink 模型



由于轴角度在 t=0 时的差异，原型的启动响应（绿色）与仿真的结果（红色、紫色）不同。

图4（b） - 硬件原型和仿真的结果比较

仿真与硬件环境之间的持续验证让设计人员在设计流程中尽早发现并解决问题。

生成如图 3 所示的时间响应。

在使用控制环路模型微调控制器之后，下一步是使用包含外设的高保真度模型在仿真中证明控制器的合格性。方法是加入用于控制器的 C 和 HDL 组件的时序精度规格模型。这些规格模型拥有生成 C 和 HDL 代码所需的语义。通过仿真随后验证配备规格模型的系统能极为精确地追踪控制环路模型。

一旦性能使用高保真度模型验证完毕，接着在硬件中对控制器进行原型设计。根据图 1 所示的工作流程，首先从生成 IP 核入手。IP 核生成工作流程指导选择目标开发板，引导完成映射内核的输入和输出端口到目标接口的流程，包括 AXI4 接口和外部端口。

通过与 Vivado Design Suite 集成，该工作流程能完成比特流构建和 Zynq-7020 SoC 的架构编程。

现在 IP 核已经加载到目标器件上，下一步是从针对 ARM 内核的 Simulink 模型上生成嵌入式 C 语言代码。从生成 C 语言代码、编译 C 语言代码和用嵌入式 Linux 构建可执行文件整个过程完全自动化，然后原型即可投入运行。

为了运行原型硬件并验证其结果是否与仿真模型一致，我们构建了一个用作高级控制面板的修改版 Simulink 模型（图 4a）。在这个模型中删除了设备使用的仿真模型（即驱动电子电路、电机、负载和传感器部分），用连接到 ZedBoard 的 I/O 替代。

在 Simulink 会话中使用该模型可以启动电机，选择不同激励配置，监测相关信号，获取供在 MATLAB 中后续处理的数据，不过现在可以重复的是脉冲测试（图 3)。

图 4a 显示了硬件原型的轴转速和相流结果与仿真结果的对比情况。硬件原型的启动顺序与两个仿真模型的启动顺序有明显差别。不过这种情况的原因预计是硬件测试中电机的转子和定子的初始角与仿真中使用的初始角不同，导致电流控制算法在通过自己的编码器校准模式驱动电机时的响应不同。从施加脉冲第 2 秒开始，仿真和原型硬件的结果就基本完全吻合。

以这些结果为基础，可以继续进一步不同负载和运行条件下的测试，也可以开展进一步的 C 语言和 HDL 优化。

工程师正在转为使用基于模型的设计工作流程，在赛灵思 Zynq SoC 上实现算法的软硬件设计。Simulink 仿真能尽早评估算法，方便设计人员评估算法的有效性。无需实验室，在桌面上就能完成设计的权衡，显著提升生产力。业经验证的 C 语言和 HDL 代码生成技术，结合对赛灵思 All Programmable SoC 的硬件支持，为算法在真实硬件上运行提供快速且可重复的流程。仿真和硬件环境间的持续验证让设计人员能够在开发流程中尽早发现并解决问题。

MathWorks 为基于 Zynq 的开发板、软件定义无线电套件和电机控制套件提供工作流程支持。如需了解有关工作流程的更多信息，敬请访问：*http://www.mathworks.com/zynq*.

MATLAB 和 Simulink 均为 The MathWorks 公司的注册商标，详见：http://www.mathworks.com/trade- marks 列出的其他商标。*其他产品或品牌名称均为其各自所有者的商标或注册商标。*

号外，号外

赛灵思一直在不断优化软件，更新培训资源，帮助软件开发人员用赛灵思 SDxTM 开发环境及相关 FPGA、SoC 硬件平台设计创新产品。在此，列出了更多资源和阅读材料，欢迎关注每期的最新季度更新。

SDSOC™ 开发环境

SDSoC 环境提供熟悉的嵌入式 C/C++ 应用开发体验，包括简便易用的 Eclipse IDE 以及用于异构赛灵思 All Programmable SoC 和 MPSoC 部署的综合设计环境。SDSoC 结合业界首款 C/C++ 全系统优化编译器，可提供系统级特性分析、在可编程逻辑中实现自动软件加速、自动化系统连接生成，以及用于加快编程速度的各种库。SDSoC 使最终用户和第三方开发人员能够快速定义、集成和验证系统级解决方案，帮助他们为其终端客户提供定制的编程环境。

- SDSoC 背景资料（PDF）
- SDSoC 用户指南（PDF）
- SDSoC 用户指南：入门（PDF）
- SDSoC 用户指南：平台与库（PDF）
- SDSoC 版本说明（PDF）
- 开发板、套件和模块
- SDSoC 视频演示
- 购买 / 下载

SDACCEL™开发环境

针对 OpenCL™、C 和 C++ 的 SDAccel 环境利用 FPGA 能将数据中心应用加速的性能功耗比提升多达 25 倍。SDAccel 环境是赛灵思 SDx™ 系列的最新成员，将业界首款支持 OpenCL、C 和 C++ 内核任意组合的架构优化编译器、库、开发板完美结合在一起，在 FPGA 上首次实现了完全类似 CPU/GPU 的开发和运行时间体验。

- SDAccel 背景资料
- SDAccel 开发环境：用户指南
- SDAccel 开发环境：教程
- 赛灵思培训：SDAccel 视频教程
- 开发板与套件
- SDAccel 演示

SDNET™ 开发环境

SDNet 环境结合赛灵思 All Programmable FPGA 和 SoC 能让网络工程师定义线路卡架构、设计线路卡并用类似于 C 的环境进行升级。它能创建软定义网络，这一技术领域超越了今天的软件定义网络 (SDN) 架构。

- SDNet 背景资料 —— 赛灵思
- SDNet 背景资料 —— The Linley Group
- SDNet 演示

软件开发套件 (SDK)

SDK 这一款赛灵思开发环境，支持在 Zynq®-7000 All Programmable SoC 的微处理器以及 MicroBlaze™ 软处理器上开发嵌入式应用。SDK 是首款支持真正同构和异构多处理器设计调试的应用 IDE。

- 免费的 SDK 评估和下载 ■

便携式波形测试仪： WaveXpert

Xilinx ZYNQ SoC内置ARM Cortex-A9双核处理器

支持数多种AD/DA和射频模块选项，实现多种测量仪器功能

开源化设计，支持客户开发行业专用测试仪器

操作系统支持

Windows Linux POWERED ANDROID iOS 7

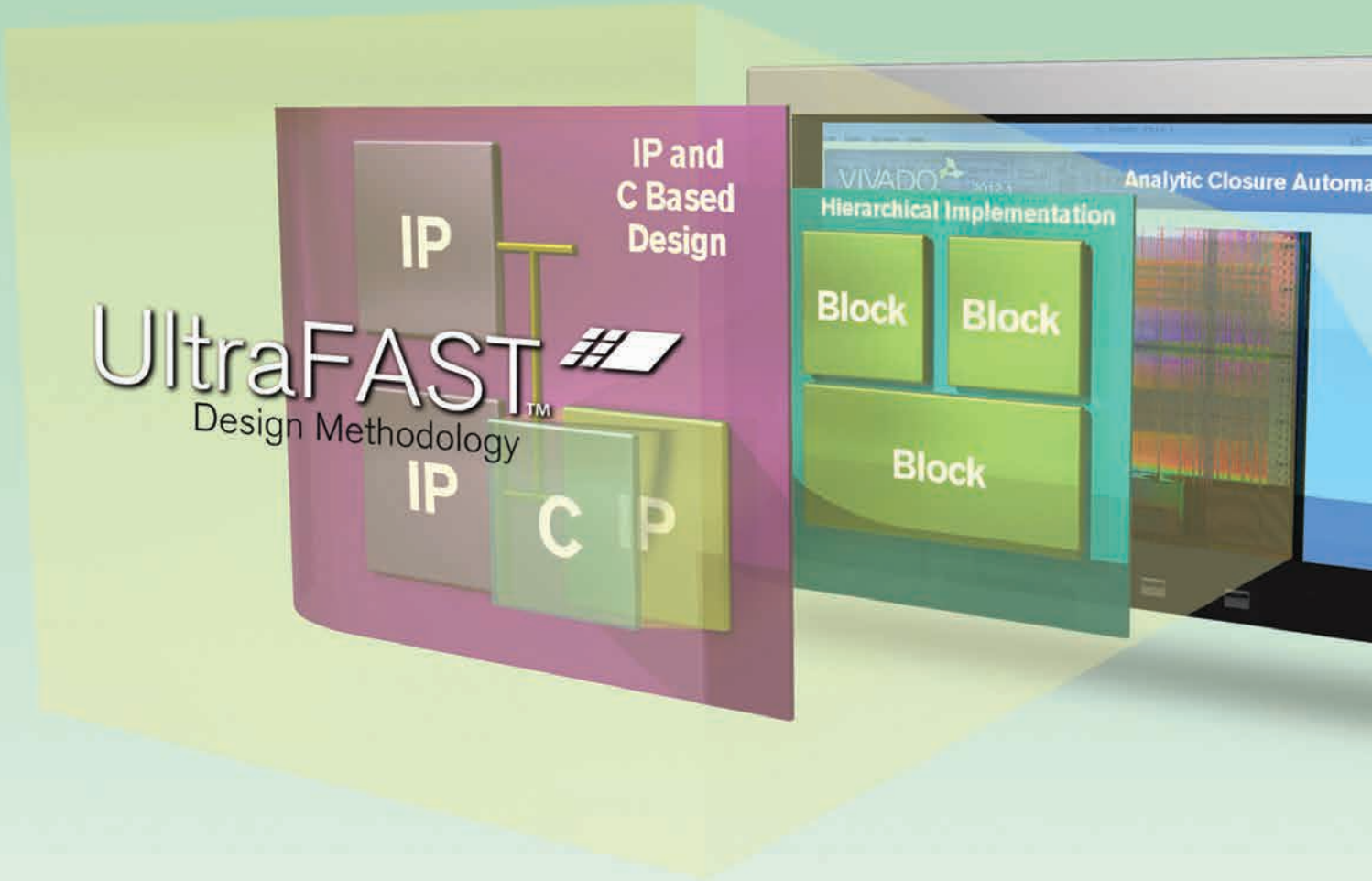
APP支持 WaveXpert SignalXpert

典型应用案例

虚拟仪器、便携式测量与测试、数据采集与分析、信号模拟、物联网信息采集终端、极客创新。

赛灵思推出

Vivado® 设计套件的 UltraFast™ 设计方法



赛灵思的 UltraFast™ 设计方法可加速设计进程并可预测设计周期。

VIVADO

XILINX