开源芯片、RISC-V与敏捷开发

王诲喆[1,2]，唐丹[1]，余子濠[1,2]，刘志刚[1,2]，解壁伟[1]，包云岗[1,2]
1. 中国科学院计算技术研究所计算机体系结构国家重点实验室，北京 100190；
2. 中国科学院大学，北京 100049

摘要
随着摩尔定理的几近失效，传统的追求通用性能的芯片开发策略将难以持续，但芯片领域过高的门槛和商业限制阻碍了进一步的创新和对市场的响应速度。因此需要通过开源芯片、统一的生态平台和现代化的设计方法激发芯片领域的创造力和生产效率。介绍了开源芯片的作用和发展历史，讨论了有望成为下一代芯片开发根基的RISC-V指令集架构的特点和影响以及前端设计中的敏捷开发实践，并对芯片开发的新发展与不足做出了总结。

关键词
开源芯片；敏捷开发；计算机架构；指令集架构

中图分类号：TP303 文献标识码：A doi: 10.11959/j.issn.2096-0271.2019032

Open-source chip, RISC-V and agile development

WANG Huizhe[1,2], TANG Dan[1], YU Zihao[1,2], LIU Zhigang[1,2], XIE Biwei[1], BAO Yungang[1,2]
1. Key Laboratory of Computer System and Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China
2. University of Chinese Academy of Sciences, Beijing 100049, China

Abstract
Due to the end of Moore's Law, traditional chip developing effort focusing on general-purpose performance cannot last long. However, the high entrance requirements and commercial limits block the further innovation and delay its time to market. Therefore open-source chips, universal platform and modern developing methodology are essential. The significance and development of open-source chips, the merit and impact of RISC-V instruction architecture and the agile development practice in logical design were introduced, and this new trend and its remaining defects were commented on in the end.

Key words
open-source chip, agile development, computer architecture, instruction set architecture

1 引言

芯片是信息社会的基石，支撑信息生活的应用与服务软件必须运行在计算芯片上才能实现其功能。软件的需求形形色色，但芯片却只能提供有限的几类选择。这种软硬件不对称的根源在于芯片设计与生产的门槛极高。芯片的功能开发、逻辑验证、物理实现都要消耗大量的人力，相关生产工具、上游产品（如已经实现特定功能的 IP 核）的授权费用和最后的流片成本也十分高昂。过去，计算机芯片一方面保证通用计算能力，从而应对不断变化的软件需求；另一方面为多媒体等普遍的计算任务准备专门的运算部件，从而保障用户的体验。通过这种泛用性能和个别领域特定性能的平衡，一款芯片可以应对足够多的场景，这样高成本和高门槛导致的多样性问题便可以得到缓解，信息领域也因此不断发展。但是随着摩尔定律的失效，人工智能的兴起对计算性能提出了极大的需求，移动平台和物联网对芯片性能与功能的诉求也日益多样化，旧的芯片开发体系正面临着巨大的挑战。

因为摩尔定律的失效，通用计算芯片的性能增长陷入瓶颈。为了进一步提高芯片的性能，满足以人工智能为首的软件任务的性能需求，业界开始研发更多的硬件加速器，针对常见的特定类型的计算任务，辅助CPU进行加速。除了加速器的内部设计之外，其如何与 CPU 互连，如何进行集成，都是需要考虑的问题。为了提高系统的集成度，芯片往往通过片上系统（system on chip，SoC）集成需要用到的硬件模块，以灵活性为代价获得性能、能耗、资源和面积等方面的优势。但是加速器的选取和设计要考虑到硬件的应用场景，如果将加速器集成到SoC里，则其灵活性和覆盖面将受到影响。为了按需集成加速器并获得SoC的优势，芯片开发需要一个统一开放的生态环境以及开源且设计良好的 SoC 开发框架。

在物联网场景下，芯片的工作环境和成本更加受限，功能需求更加明确，但需求的种类十分多样。此前追求性能和通用性的计算芯片并不能满足物联网的需求，厂商迫切需要具有可以充分定制、剪裁功能的芯片产品，从而控制芯片的预算。上游芯片设计企业以及原始设备制造商（original devive manufacturer，ODM）也需要缩短开发周期，以应对多样且变化迅速的消费者和市场需求。因此在新的需求场景下，芯片设计生产需要更加敏捷快速的开发模式。

在芯片开发体系无法满足当下需求的同时，互联网领域已经通过统一开放的开发平台、开放开源的行业环境、敏捷迅速的开发模式取得了辉煌的市场成绩。在基础设施方面，互联网基本做到了统一、开放和共享。报文通信使用的网络协议（如TCP、HTTP）和浏览器、页面渲染使用的HTML、CSS、JavaScript等互联网基础设施基本做到了统一和开放。各类开源的互联网前后端开发框架则充分体现了共享精神。开发方面主要有各种编程语言的开源工具链，在此之上还有庞大的免费共享的软件包社区和先进的包管理器。当开发者需要实现某个基础功能时，可以从社区免费获取既有的实现代码库，并且因为包管理工具的存在，代码库的发布与获取变得极其便利。虽然作为执行平台的浏览器往往是闭源的私有软件，但是因为有统一的Web标准，浏览器底层的实现差异基本被抹平。在开发流程上，互联网领域提倡快速原型、敏捷开发、在线部署，并为此开发了各类测试驱动和持续集成框架。

反观芯片设计领域，除了硬件描述语言规范有公开的标准、开源社区在软件仿真和波形上有不错的成绩外，逻辑设计流程基本被大公司的电子设计自动化（electronic design automation，EDA）工具垄断，开发者需要面对高昂的许可证费用以及完全封闭隔离的工具链体系。硬件描述语言本身的发展相比互联网行业也显得落后，主流的硬件描述语言Verilog和SystemVerilog缺少高级的、对开发者友好的语言特性，EDA工具本身的编辑器也不够智能，使得硬件描述语言的编写仍处于一个相对原始的水平。一些自动化的功能由EDA工具本身的软件功能提供，不具有可移植性。EDA工具厂商提供的软核处理器可作为硬件系统的基础，如新思科技（Syopsys）公司的Nios II和赛灵思（Xilinx）公司的Microblaze，但是它们都面临着严格的许可证的制约，其应用范围受到很大限制。第三方提供的实现特定功能的 IP 核也受到类似的知识产权的限制。而芯片的后端设计更是和芯片制造厂商绑定在一起，对外基本封闭，前后端的交流和迭代都十分困难，导致开发效率低下。

芯片设计领域在新时代下的诉求与互联网的成功经验有一定程度的重合，即开放性、灵活性、统一平台和开发效率。本文将针对这几个要素，首先介绍开源芯片的发展情况，然后介绍正在蓬勃发展、有希望为开源芯片提供统一稳定的生态平台的RISC-V 指令集架构，最后介绍目前芯片设计领域在前端开发实践上的一些进展。

2 开源芯片

2.1 开源芯片的意义

对于初创企业和高校的科研团队来说，芯片设计的门槛极高，这是因为芯片设计与制造的每个环节都有巨额的成本。芯片设计制造涉及 EDA 工具的授权与使用、前端逻辑设计与验证、IP模块的选用与购买、模块集成测试、后端物理设计以及最终的流片与封装。其中，EDA工具和IP模块都需要高昂的授权费用；逻辑设计开发效率低下，容易出错；测试验证迭代周期漫长，硬件问题难以定位；前后端交接效率低下；流片封装成本高昂，而且需要竞争晶圆代工厂的产能。这样的高门槛极大地阻碍了芯片领域新鲜血液的出现与创新，同时由于失败成本极高，测试与验证方面的投入不得不进一步加大，使得门槛进一步提高，在领域多样性的发展层面形成一个死锁。

虽然芯片领域出现了高度的分工，但在传统的芯片设计授权模式下，指令集和公版设计都要收取一笔客观的授权费用，同时再针对指令集收取版税，这对芯片公司来说是不小的开销。同时，对公版设计的修改往往会被加以限制，想要深入修改还需要再支付一笔费用。这阻碍了芯片公司的创新，使得深入核心内部的充分改造与优化有着高昂的额外成本。创新性的功能实现与优化设计往往只能以IP核的形式在芯片外部实现，在印刷电路板（printed circuit board，PCB）上与芯片互联，但这样便限制了性能、增加了能耗、浪费了面积。

如果开源芯片能够普及，首先可以节省 IP 模块方面的费用，降低一定程度的研发成本，同时由于开源的设计可以由社区持续地改进，所有人都能享受到最新、最优化的成果，这样便可以提高行业整体的发展水平。繁荣的开源社区往往会催生脱离个别商业公司限制的统一的开源工具链。对于芯片领域来说，则是EDA工具的授权费用得以节省，同时由于工具链的统一，设计经验和代码资源的复用将变得更加容易，在一定程度上能够进一步提升开发

的效率。如果以开放指令集代替版权指令集，除了能节省研发和生产的成本外，还能使软件生态更加繁荣、统一、开放。

此外，在安全方面，微架构设计一直存在着各种各样的威胁，除了各种侧信道、隐藏信道和像 Spectre这样由推测执行导致的理论上的漏洞外，也有类似于Meltdown和超线程等实现导致的漏洞。对于开源芯片而言，实现上的漏洞会受到社区的持续审视，可以更早地被发现，从而及时进行修补并制定解决方案。

自20世纪70年代开始的开源软件运动已经通过互联网领域的市场繁荣证明了自己的价值。形成统一标准并持续、开放地发展源码的结果是企业能够迅速地对市场做出反应，提供产品，或者以极低的成本验证创新性的项目。

尽管开源硬件的起步并没有明显落后于开源软件，但是受限于物理成本和垄断化、商业化的开发环境，其发展并没有开源软件那般充分。但是，随着芯片制造技术越来越难以达到摩尔定律预言的发展水平，硬件厂商需要更加灵活多样、针对细分市场的芯片来支撑其业务。在这样的背景下，开源的作用将得以凸显。廉价的现场可编程门阵列（field-programmable gate array，FPGA）设备和免费EDA工具的出现，使得根据源代码制造真实硬件所需的成本相对下降，以源代码的形式发布和分享硬件设计更加可行，需求也更加强烈。限制开源芯片发展的历史条件一部分正在逐渐消解，另一部分则是当下的重点攻关项目，此外开源芯片进一步发展的条件也逐渐成熟。在需求显现、条件可行的当下，开源芯片将如同开源软件那样，为芯片领域注入新的活力，并促进其发展。

2.2 开源芯片的发展

目前，开源芯片已经有了一定程度的发展。这一方面是因为商业模式的创新使得芯片设计门槛降低，另一方面是因为开源社区不断地在软件层面提供开源的替代方案。

由于制造芯片的成本极其高昂，从1978年到1981年，麻省理工学院的Lynn Conway等人开展了一系列对超大规模集成电路（very-large-scale integration，VLSI）设计和快速芯片原型方面的总结和探索，创造性地提出了在同一块晶圆上集成多个不同设计的芯片以降低单一芯片流片成本的设计方案——多项目晶圆（multi-project wafer，MPW），其一系列成果最终催生了金属氧化物半导体实现服务（metal oxide silicon implementation service，MOSIS）机构。该机构由美国国防高级研究计划局（DARPA）资助，专门为商业公司、大学和科研机构提供流片服务[1]。

除了硬件生产层面的创新外，学术界在EDA工具方面也有所发展，比如加州大学伯克利分校（UCB）的SIS[2]，加州大学洛杉矶分校的系统快速原型设计（rapid system prototyping，RASP）[3]和科罗拉多大学博尔得分校的 BOLD[4]，它们都保持了开源和免费的特性。学术界的这些EDA工具启发了EDA公司（如Cadence公司和Synopsis公司），但到了20世纪90年代，商业公司的闭源EDA工具成为主流标准，学术界在EDA工具研发方面变得小众且发展有限。

在1980年以前，大部分生产芯片的公司也进行芯片设计，因此被称为垂直整合制造商（integrated device manufacturer，IDM）。虽然在1969年就已经有无晶圆厂的纯设计公司（如LSI计算机系统公司）出现了，但是这些无晶圆厂公司需要等待IDM的产能空闲下来才能进行生产，效率受到限制。MOSIS的诞生催生了纯粹的晶圆代工厂，如台湾积体电路制造股份有限公司（TSMC）。由于分

工的明确化，越来越多的无厂半导体企业出现了，如英伟达（NVIDIA）公司、高通（Qualcomm）公司、博通（Broadcom）公司和赛灵思公司等。此后有越来越多的兼顾芯片设计的垂直整合制造商（如联华电子股份有限公司、超威半导体（AMD）公司、英特尔（Intel）公司）转型、分拆、提供晶圆代工业务。虽然制造芯片的难度并没有降低，但是通过晶圆代工和无厂半导体企业的商业模式，初创公司进行开源芯片起步投产的门槛大大降低。

FPGA的出现使得以源码形式发布硬件设计变得更为可行。通过EDA工具将源码进行综合，并存储在闪存等外部持久化存储设备中，与FPGA芯片和其他芯片一起焊接在PCB上，接通电源后将电路设计烧录到FPGA中，这样便在一定程度上获得了可以工作的硬件系统。约定好FPGA芯片型号和其他另购的基础部件后，一个开源硬件的硬件描述代码便可以快速地部署到真实的硬件中，方便他人进行生产制造，而不需要对源代码进行修改和适配。处理器的核心是硬件系统里不可或缺的组成部分，FPGA使得通过源码发布的软核（以源代码形式发布，部署在FPGA上，系统断电后无法保存结构的处理器核心，与之相对的硬核则直接生成固定的集成电路，分发给使用者的处理器核心，处理器的功能和结构无法修改）变得可用，维持并促进了软核的用户生态和社区发展。

由于物理后端的技术门槛较高，开源社区主要提供的是局限于前端硬件描述语言源码级别的芯片开源实现（相比之下，PCB和3D打印在获取设计描述文件后可以直接通过设备进行实物生产），也就是以IP核的形式传播。具有代表性的开源处理器IP核有 OpenRisc[5]、LEON[6]和OpenSparc[7]。受开源指令集 RISC-V 的影响，Wave Computing 公司（MIPS指令集的持有者）也计划开放部分处理器的实现源码以供参考。同时，为了规避总线协议的版权问题，社区提出了开源的 Wishbone 总线协议以及其简化版本——简化总线结构（simple bus architecture，SBA）。

在设计工具方面，前端仿真有一定程度的发展，Verilator等开源仿真器有足够的性能，同时众多的开放波形文件格式也基本解决了查看波形时大文件读取效率低下的问题，与商业工具处于相当水平。EDA工具方面也并非一片空白，但大部分EDA工具（比如KiCAD、gEDA等[8]）是面向生产门槛较低的PCB设计的，面向集成电路（integrated circuit，IC）以芯片为目标的设计工具较少，主要有Qflow、Electric[8]和学术物理设计套件（academic physical design kit，APDK）。一些企业（如Efabless公司等）在提供免费开源EDA工具链的同时，也在设计者和晶圆代工厂之间建立了渠道，并且只在生产原型时收取一定的费用。

在社区交流方面，Github这样的社交型代码托管网站对开源社区起到了不可忽视的催化作用，而开源芯片社区也有类似的平台。

OpenCores是1999年起步的数字设计者交流社区，数字设计者在OpenCores上发布自己的开源数字逻辑设计，类似于现在开源界流行的Github服务。目前OpenCores由初创公司Oliscience公司进行资助，并为芯片开发和IP集成提供专业咨询。自由与开源硬件（Free and Open Source Silicon，FOSSi）是类似自由软件基金会的非营利组织。与软件类似，他们认为硬件部件的组建也应当是自由而开放的，并且这一性质应当传递下去。FOSSi建立了类似OpenCores的交流平台——

LibreCores，对开源芯片项目进行收录汇总，并通过用户评价和自动分析提供优化的检索和评估服务。LibreCores还致力于实现被称为fusesoc的开源芯片包管理器、持续集成和完善文档等工作，以使芯片开发变得更加方便。

但是开源芯片社区仍然存在着不足。首先，对于最终的流片而言，只提供源码级IP核并不只是增加修改逻辑功能那么简单，FPGA和专用集成电路（application-specific integrated circuit，ASIC）的时序特性不尽相同，芯片工艺、代工厂之间也存在着差异，这些差异会深刻影响前端逻辑实现的决策，这意味着获取的芯片源码仍然要经历大量的适配性调整。同时，开源芯片使用的架构往往不同，有些甚至试图兼容具有商业版权的指令集，这就给软件生态带来了极大的分裂，前人的软件开发与移植成果很可能不能在新的芯片上加以复用，各芯片最理想的情况也只能是使用最基本的系统运行库，任何高级软件都要由有需求的组织负责移植，并测试和验证正确性；由于用户数量较少，软件代码版本往往存在一定的滞后性。这些都是对运行在芯片上的软件生态不利的因素，而这些不利因素会反过来制约企业确立项目时对开源芯片的考量。在后端工具方面，对于中高端的芯片，依然缺乏功能完备的开源EDA工具链与工艺库（工艺库指用于生成电路的基础原件集合，工艺库越先进，可发掘的性能空间就越大，对电路布线的要求就越宽松）。

目前，开源芯片领域仍然不像开源软件领域那般成熟，各个方面都有巨大的改进和提升的空间。开放指令集RISC-V致力于解决软件生态和硬件平台问题，越来越多的现代开发工具和开发框架则在一定程度上缓解了调整源码和复用代码的难度。

3 开放指令集 RISC-V

3.1 RISC-V 介绍

计算机系统有许多抽象层，比如操作系统的系统调用和设备文件抽象、网络传输协议的分层架构等，其中最重要的是处于软硬件交界处的指令集架构。抽象层都有一套协议或者规范，它们绝大多数是由非商业的委员会制定的，在业界免费公开，这使得计算机行业的参与者可以在遵守规范的情况下对各个模块不断进行改良，择优使用，进行纯粹的技术竞争，同时保证整个生态环境的兼容性和稳定性。目前计算机行业的发展状态也证明了这种开放规范的模式的成功。但是，在计算机行业处于支配地位的商业指令集（如x86和ARM）并不遵循这一模式，它们的指令集设计是有版权的，要想研制支持该指令集的处理器，必须支付授权费用，有时甚至无法获取授权，这为计算机芯片的竞争引入了技术外的门槛。此外，这些流行的商用指令集有庞大而复杂的兼容性要求，在这样的架构下开展芯片设计的研究，不得不先将其实现完整。这对于学术界和初创企业来说是巨大的负担。此外，学术界和开发社区做出的改进往往不会直接被指令集所有者接受，也很难直接转化到实际的芯片产品中。因此无论是工业界还是学术界，都迫切需要全新设计的、没有历史包袱和版权壁垒的开源指令集，并将其作为统一的研发平台。

RISC-V便是为了解决上述问题而诞生的开源精简指令集架构，由UCB于2010年发布[9]。UCB团队以CC 4.0协议发布了RISC-V指令集的规范文档，并以BSD开源协议发布了RISC-V架构的开源硬件

实现Rocketchip。这些许可协议保证了RISC-V的开放性，人们都可以为RISC-V开发硬件，而不需要付出额外的成本。

RISC-V充分吸收了既有精简指令集（reduced instruction set computer，RISC）的经验教训，使其设计更加现代化。RISC概念刚刚被提出时，体系结构的发展有限，使得很多RISC对单一架构设计过度依赖（比如单发射5段流水线），产生了分支指令延迟槽这样的过度优化设计，给异常处理和乱序执行设计带来了很大的麻烦。另外一些过度优化的设计（如OpenSPARC的滑动寄存器窗口和条件码分支指令）在实践中收益较低，并且极大复杂化了处理器的设计，也是应当舍弃的。其他一些设计（如条件移动指令）也对使用寄存器重命名技术的现代乱序处理器极其不友好（移动寄存器值是否需要等待指令执行后才能确认，是不确定行为，但是被写入寄存器的重命名需要在指令执行前完成，与条件执行的语义相冲突）。在寻址模式上，RISC-V更加注重基于程序计数器（program counter，PC）的寻址，使得位置无关代码（position-indenpendent code，PIC）的应用更加便利。当下，因为资源有限，嵌入式场景要求程序尽可能小，因此对压缩指令有广泛的需求，非商业化RISC指令集往往使用较宽的立即数编码（13～16位），在设计时没有考虑压缩指令的需求，使得日后追加压缩指令时需要准备另一套编码，并进行模式切换。ARM虽然在ARMv7中提供了Thumb压缩指令集，但是其性能较差；随后又推出了Thumb-2，指令集提高了性能，但是这导致要完整支持 ARMv7 需要实现3套指令集，十分复杂；而ARMv8作为 64 位的指令集架构，因为本身设计得比较压缩，便移除了专门的压缩指令集，虽然代码尺寸比ARMv7的非压缩指令集小，但是依然无法达到具有非固定指令长度的压缩指令集的压缩率。RISC-V设计时采用12位立即数，并提供兼容的RVC压缩指令集拓展，使得压缩指令的使用与实现更加灵活方便。在指令格式设计上，RISC-V保证了不同指令格式同语义字段的位置相同（如**图1**所示），其主要体现是立即数可能会分散在指令中间，这样对译码电路的简化有很大的帮助[10]。**图1**中，操作码（opcode）和功能码（用funct表示）字段用于区分指令。rs1、rs2 分别表示第一个和第二个源操作数，rd 表示目的操作数，imm 表示立即数（即直接存储在指令里的数值）。

RISC-V最大的特点是模块化，UCB的设计者们希望RISC-V能够用一套规范

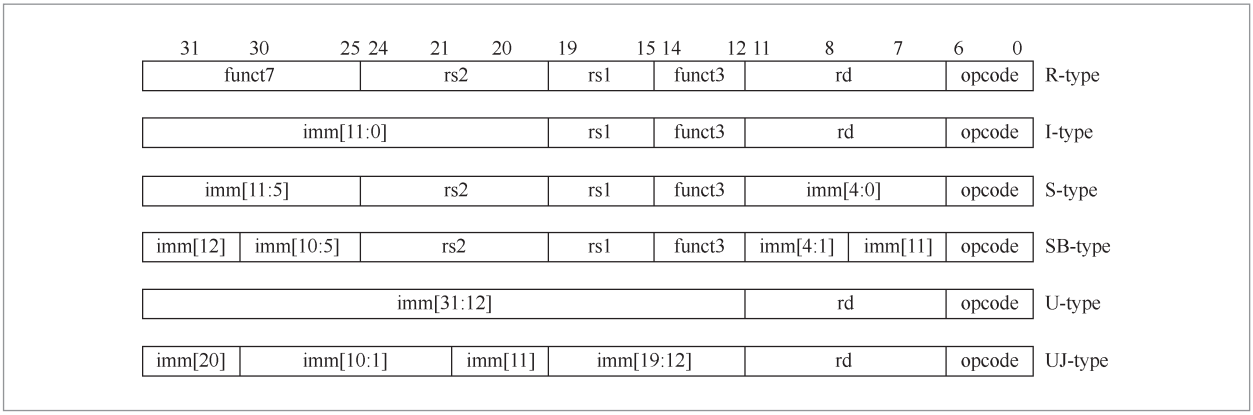

图 1 RISC-V 不同指令格式及其字段分布

适应从低功耗嵌入式设备到高性能计算乃至领域专用硬件的不同应用场景。RISC-V的标准指令根据所属的应用场景被划分成I（基础整数）、M（整数乘除）、A（多核同步）、F（单精度浮点）、D（双精度浮点）5类，芯片实现者只要实现I指令即可实现对RISC-V程序最基础的运行支持，之后可以再根据实际需求进一步实现标准扩展指令。在RISC-V 2.2版本规范[11]中，基础的I 和扩展的M/A/F/D这几个重要部分皆已冻结，表明RISC-V的基础设计已经稳定。RISC-V对向量指令（V扩展）的支持也正处于起草阶段，以在日后满足对高性能计算的需求，有别于传统的单指令多数据流（single instruction multiple data，SIMD）指令，RISC-V的V拓展将内部向量寄存器的宽度等结构信息与指令集解耦，可以在线重构，并支持多种数据形式（标量、向量、矩阵），具有更强的灵活性。

RISC-V在设计指令时预留了充足的空间。如果不需要压缩指令集（RISC-V compress，RVC），则有3块 30 bit 的编码空间可供使用。在基础和标准扩展指令集里，也有4份对应 25 bit 编码空间的自定义操作码可供使用，不过其中2个将来会和128位规范的标准指令重合。通过这些自定义指令空间，RISC-V鼓励设计者集成需要的专门加速部件，这些部件与核心直接相连，通信效率高，使得RISC-V可以灵活地为领域特定计算进行服务。

RISC-V同样为操作系统和其他场景提供了更高的权限模式。除了基础的M模式（machine mode）外，还有操作系统使用的S模式（supervisor mode）和用户软件使用的U模式（user mode）。M模式是必须要实现的，简单的嵌入式系统也只需要支持M模式即可。M模式和U模式直接组合可以实现简单的基于地址寄存器的内存隔离，而更复杂的基于分页的虚拟内存方案需要依靠S模式实现。RISC-V委员会正在探讨S模式的递归虚拟化，这样即可在不引入新的模式和概念的情况下实现云场景的托管运行（为此删除了H模式，即hypervisor mode）。RISC-V提供了一套异常委托机制，默认情况下中断和异常都在M模式下处理（虽然经过系统软件配置后可以通过mret指令很快地返回需要的特权级进行处理），通过委托机制，便可以选择性地将中断和异常交给S模式甚至U模式进行处理，完全绕过M模式，减少不必要的特权级切换代码路径，提高性能。

RISC-V的高度模块化使得其最基本的实现为I系列指令+M模式特权级，尤其适合嵌入式芯片。而为了保证兼容和避免生态碎片，ARMv8强制要求实现所有的SIMD 指令。

3.2 RISC-V生态下开源芯片的发展

UCB除了RISC-V自身规范的开放和参考实现的开源外，还准备了完善的软件工具服务社区。比如提供指令集模拟器验证RISC-V程序的正确性；为系统仿真器QEMU增加RISC-V架构支持；为全系统模拟器gem5提供RISC-V架构支持；为Openocd和GDB提供RISC-V架构支持和调试接口；基于主机端到目标端接口（host-target interface）的代理核（proxy kernel，一个RISC-V程序）快速部署程序到RISC-V处理器核心（即目标端，target）上，并在主机端（一个非RISC-V的完整系统）通过通用总线与RISC-V处理器核进行通信，代理RISC-V核心对内存和外设的访问，以缓解RISC-V生态初期外部设备不足的问题，使得RISC-V从诞生之初就处于一个比较高的起点。

RISC-V已逐渐成为芯片设计领域的主流指令集之一，且广受各大厂商青睐。

截至2019年1月，已有包括Google、NVidia等在内的200多个公司和高校资助和参与RISC-V项目。其中，部分单位已经开始将RISC-V集成到产品中。例如硬盘厂商西部数据（Western Digital）公司把每年各类存储产品中嵌入的 10 亿个处理器核换成RISC-V；Google公司利用RISC-V实现主板控制模块；Nvidia公司将在GPU上引入RISC-V等。此外，阿里巴巴集团、华为技术有限公司、联想集团等公司都在逐步研究各自的RISC-V实现。RISC-V在工业界获得了广泛应用。

在社区发展方面，由AntMicro、Esperanto Technologies、Google、SiFive和Western Digital等IT公司组建的CHIPS联盟希望促进芯片开发的互助环境，以加速面向各个场景开发高效和灵活的芯片。该联盟隶属于Linux基金会，与上述的开源芯片社区不同，该联盟首先专注于RISC-V生态的发展，为RISC-V指令集提供有参考价值的微架构实现和指令集无法涉及的加速器设计与实现。

3.2.1 开源 SoC 实现

如前文所述，一个灵活而开源的SoC实现对于开源社区至关重要。而RISC-V社区除了各类开源核心和功能部件外，也有很多成熟的开源SoC项目。

Rocketchip Generator（以下简称RocketChip）[12]是RISC-V较为知名的SoC项目，它本身自带顺序流水线核心Rocket，并且有丰富的配置项目（如图2所示）。RocketChip基于RISC-V自定

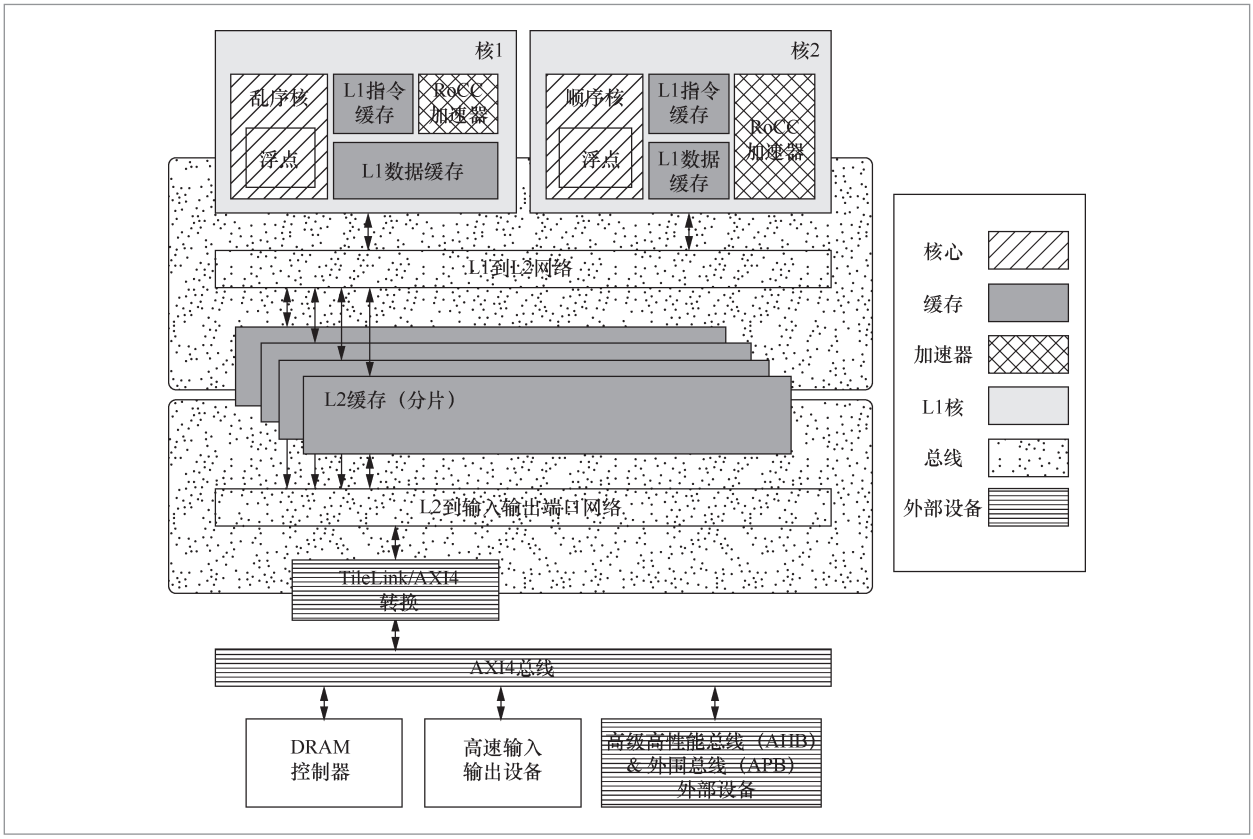

图 2 RocketChip 的基本架构

义指令空间，设计了标准的加速器接口Rocket自定义协处理器（Rocket custom coprocessor，RoCC），它复用Rocket处理器核的译码逻辑，可以快速地实现基于指令的加速器软件接口。它与核心直接相连，不受访存总线干扰，通信上更有效率。

LowRISC是由剑桥大学团队开发的RISC-V架构的开源SoC项目，它使用标签内存（tagged memory）等创新性技术拓展其安全方面的功能，并通过开源使其代码可以被用户审阅，进一步保证安全性。LowRISC可以说是开源芯片优势的一个很好的例证。

并行超低功耗（parallel ultra low power，PULP）[13]是苏黎世联邦理工大学主导的基于RISC-V架构的开源SoC平台，致力于在保证低功耗的同时提供高度可拓展的配置，使其应用场景可以从物联网（Internet of things，IoT）跨越到高性能计算（如**图3**所示）。

3.2.2 开放总线协议TileLink

SoC设计绕不开总线协议，而前述的开放总线协议Wishbone缺少对缓存一致性的支持，因此UCB团队设计了一套适用于SoC内模块互联的总线协议——TileLink[14]。TileLink本身与架构无关，可以支持任意缓存一致性算法，而且和RISC-V一样，是免费而开放的标准。TileLink具有5个通道，分别如下。

- A 通道：传输请求，访问或者修改数据。
- B 通道：传输从属设备（slave）对主设备（master）缓存数据的请求，获取可能被缓存的数据或者请求写回脏数据块。

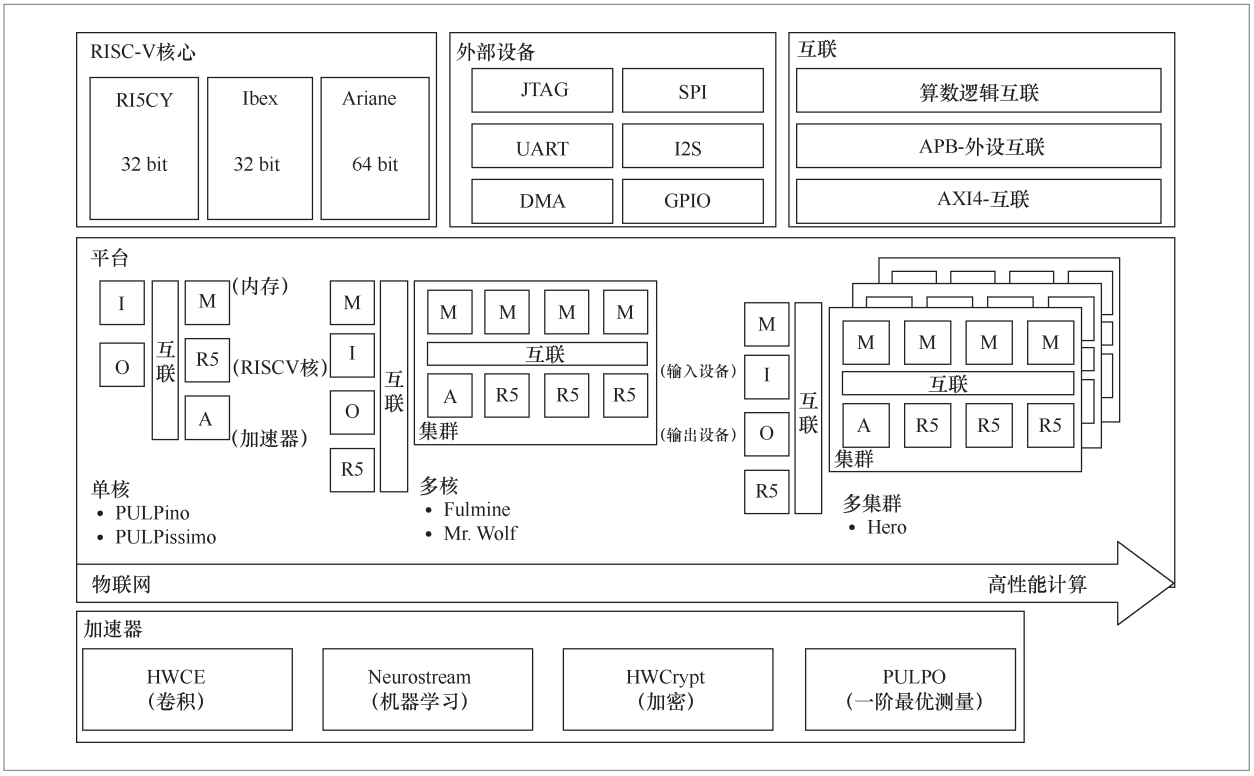

图 3 PULP 的基本架构

● C 通道：传输对B通道请求的响应以及可能的写回数据。

● D 通道：传输对A通道请求的响应和数据。

● E 通道：传输缓存块传输完成的结束标志，用于请求的序列化。

由于结构相对简单，嵌入式设备资源相对紧张，需要更精简的总线协议，因此被广泛使用的先进微控制器总线架构（advanced microcontroller bus architecture，AMBA）提供了先进可拓展接口（advanced extensible interface，AXI4-Lite）、先进高性能总线（advanced high-performance bus，AHB）、先进外设总线（advanced peripheral bus，APB）等简单但低性能的协议来满足嵌入式市场的需求。TileLink也提供精简版本的子协议：TL-UL、TL-UH（TL对应TileLink，U对应Uncache，无缓存设备，L对应低性能，H对应高性能）。TL-UL和TL-UH都不支持缓存设备，其中TL-UL只有基本的数据读写功能，而TL-UH相比于TL-UL，增加了对原子传输和示意（hint）请求（用于可能存在的预取功能）的支持。

4 芯片敏捷开发

芯片敏捷开发的概念是从软件敏捷开发借鉴而来的。在搞清楚什么是芯片敏捷开发之前，首先讨论软件敏捷开发的含义。

软件开发需要从“开发模式”和“设计模式”两个维度来考虑。其中“开发模式”指项目管理方式，“设计模式”是项目的具体实现方式。传统瀑布式开发和敏捷开发在“开发模式”上的主要区别是对需求的处理阶段和处理方式。在瀑布式开发模式中，要求在项目开始阶段把需求分解为开发人员能理解和实施的文档，在需求分解完成以后进行“需求冻结”，即不再接受新的需求。开发人员根据需求文档就可以按时保质保量地把项目完成。这种开发模式门槛较高，适合大型软件开发。敏捷开发模式在移动互联网项目开发中比较常见。移动互联网项目开发门槛较低，竞争激烈，需要快速试错。而且移动互联网需求变化迅速，不能像瀑布式开发模式那样从一开始就把需求分解清楚。因此一般采用需求快速迭代的方式逼近最终需求。设计模式的核心内容是代码复用以及修改灵活。为了能实现快速开发，软件开发经过了面向对象、软件工程、开发框架以及开源软件等一系列变革，使得软件开发的编程实践门槛大大降低。可以说软件开发的设计模式是开发模式的基石，更是敏捷开发能实现快速需求迭代的根本。

芯片开发与软件开发有很多相同之处。从“开发模式”来看，二者并没有本质区别。芯片开发可以采用传统瀑布式开发模式的“需求分解冻结”方式，同样也可以采用敏捷开发模式的“需求快速迭代”方式。只是芯片具有更高的迭代成本和更低的迭代效率。但在“设计模式”上，芯片开发与软件开发之间的差距极大。芯片开发在开发语言、面向对象、开发框架以及开源IP等方面几乎没有发展，几乎处于“设计模式”的“石器时代”，难以对芯片的敏捷开发模式提供支撑。

4.1 新型硬件描述语言

4.1.1 Chisel

UCB团队在推出 ISC-V参考设计Rocket Chip的同时，设计了一套全新的硬件描述语言Chisel[15]和RTL中间表示语言FIRRTL[16]。Chisel相当于Scala语言的类库，这使得硬件开发者可以充分

利用Scala的高级语言特性增加硬件描述语言(hardware description language, HDL)的代码复用程度和工程化可维护水平,极大地提高了开发大型硬件项目的开发效率。编写好的Chisel代码会被编译成Scala程序,然后该程序执行生成FIRRTL代码,再由FIRRTL编译器生成最终的Verilog代码,以供现有的EDA工具使用。

Chisel的主要目标是实现芯片开发代码重用,减少项目中的重复代码,提高代码密度,提升开发效率,提高代码的可读性和易维护性。**表1**总结了Chisel和传统的硬件描述语言(Verilog和SystemVerilog)之间的差异,并对Chisel语言的优势进行了详细描述。

Scala强大的类型系统使得人们可以很容易地拓展类型,在不修改或引用该类型的代码的情况下,可以轻松地对该类型的信号进行全局修改。此外,整体连接运算符“< >”会根据类型的定义把一个模块内相同类型且需要互联的2组信号(比如主设备端口和对应的从设备端口)的相应成员信号一一连接,省去重复而且易出错的连线代码。在工程项目中,笔者一般会在代码中的不同位置定义同种类型的信号,比如总线信号。在这种情况下,信号整体连接的特性能大幅减少项目中的重复代码。相比而言,使用Verilog语言进行开发时,若要对总线的成员信号进行改动,工程师只能对项目中所有用到总线的模块端口逐一进行改动,同时还需要手工添加或移除相应成员信号的连续赋值语句(即组合逻辑),这对于Verilog工程师来说是一件比较困难的事情。SystemVerilog的接口(interface)特性在一定程度上也能实现类似Chisel中信号整体连接的功能,但它仍然有一些局限性,例如modport(SystemVerilog的语法关键字,用于将已定义的端口分组封装)不能嵌套定义,

表1 Chisel、SystemVerilog、Verilog 在语言特性上的比较

特性	Chisel	SystemVerilog	Verilog
信号整体连接	支持	支持,但有局限性	不支持
元编程	支持	部分支持且不可综合	不支持
面向对象编程	支持	部分支持且不可综合	不支持
函数式编程	支持	不支持	不支持

使得人们无法从不同的接口中将相同的部分进一步抽象出来。Chisel中的信号整体连接特性比SystemVerilog强大,这使得人们可以进一步减少重复的代码,实现“一改全改”的效果,从而提升项目的开发效率。

Scala的元编程同样可以应用到Chisel代码的编写中,提高代码一致性与复用程度。一个常见的例子是使用模板实现队列原型。无论队列中的元素是何种结构的复合,队列本身的功能都是一样的。使用模板可以将队列本身的功能抽象成一个带类型参数的队列原型,然后在实例化时给出队列元素类型,就可以得到一个该种元素类型的队列。通过这种方式,不必分别实现不同元素类型的队列,只需要维护一份队列原型即可。更进一步地,需要修改队列元素类型时,不需要手动维护每个该元素的队列实例;队列的成员可以被类型系统检查,不会将意料之外的无关信号连接到队列模块的入队端口上,可在一定程度上减少混淆问题。要想在Verilog中实现这一需求,针对代码中的所有相关队列,可以修改队列元素的宽度,或者通过不同的队列来缓冲不同的元素,同时还需要维护不同队列之间的对应关系,十分烦琐。SystemVerilog也支持模板,但相应的代码是不可综合的,更多的是被用在测试激励的编写中。

面向对象编程中的继承特性允许将一些共同的电路结构通过一个父类抽象

出来，达到减少冗余代码的效果，同时层次化的类型系统还可以使类型检查的过程更加严格，从而降低代码出错的可能性。具体地，在实现一个模块时，只需要从父类继承，就可以让该模块自动拥有父类定义的所有电路结构，从而避免在不同的模块中重复实现这些结构。在硬件项目中，很多模块有一些共同的结构，例如缓存的不同替换算法需要读出并更新历史状态，而对于带有总线接口的模块，总线相关的参数也非常相似。在这些情况下，使用继承机制可以有效地节省项目的代码量，让代码功能一目了然。相比之下，传统的硬件描述语言难以实现类似的效果，例如若使用Verilog编写，只能通过子模块或者宏来复用这些共同的结构，甚至要在不同的模块中重复声明相关的参数，使得Verilog代码难以一目了然。而SystemVerilog虽然支持继承，但功能有限，而且相应的代码不可综合，无法对硬件构建提供帮助。

RTL中往往有批量重复的结构（比如标签匹配的比较器逻辑和对比较结果的汇合），Scala的函数式语法设施十分适合描述这些结构，可以使代码更紧凑，可读性更好。Scala的容器（collection）可以统一维护具有相同功能的电路元素，如信号线、寄存器、端口、模块等。通过遍历（foreach）、映射（map）、归约（reduce）等方法对容器中的对象进行批量操作（即批量生成电路）。由此可以通过少量代码描述复杂电路。Verilog虽然有for和generate语法，但它们只能基于整数进行迭代，功能非常有限，且表达不够简洁。综上所述，与传统的硬件描述语言相比，Chisel依托Scala的高级语言特性可以大量减少项目中的冗余代码，提高项目的开发效率，同时高密度的代码也提高了可读性，使得项目更容易维护。

4.1.2 Bluespec

除了Chisel外，学术界和工业界也有对改良硬件开发模式的探索，Bluespec[17]是Arvind教授自20世纪80年代开始研究硬件形式化验证后，基于规则组合（rule composition）理论[18]提出的硬件描述语言。它具有与Chisel类似的高级语言特性（如复合数据结构、继承等），虽然早期它也像Chisel依托于Scala一样依托于Haskell，但是目前商业使用的版本已经使用独立的语言前端进行编译，这也使得它的语法更加简洁凝练，但也缺少了一些原生高级语言具有的灵活性。它虽然不是高层次综合（high-level synthesis，HLS）语言，但是代表着不同于传统硬件描述语言的全新设计范式。在传统的RTL（如Verilog）中，一个寄存器的更新逻辑要集中于一个语句块（Verilog 语法：always block），不同的语句块不能更新同一个寄存器，这使得编码时需要对整个模块逻辑的所有情况有清楚的认识，而Chisel对寄存器更新逻辑的位置没有任何要求，需要开发人员自身去约束代码的结构与布局。而Bluespec则要求以操作为单位描述代码，每个模块由若干个规则组成，每个规则对应一个具体的线性功能，一个寄存器不能在一个规则中更新多次，而可以分散在不同的规则中。此外模块的接口不是信号端口，而是有特定输入输出的方法，每个方法自动生成握手信号。将子模块调用方法的就绪信号汇合起来作为规则的默认就绪信号，规则生效后调用子模块的方法也变为有效状态。规则的编写虽然专注于单一操作，但是可能产生冲突，Bluespec公司的综合器会提示这些冲突，从而使开发人员可以有的放矢地进行优化和迭代，降低了设计模块的难度。相比于其他硬件描述语言，Bluespec提供了完全不同的电路设

计思维模式。

然而，不同于开源且活跃开发的Chisel，Bluespec的前端和综合器本身是闭源的，且无法公开获取，只能向Bluespec公司提出申请以及授权。这也使得与Bluespec相关的资料相对较少。

4.1.3 MyHDL 和 PyRTL

MyHDL[19]和PyRTL[20]都是基于Python的HDL。Python具有亲和的学习门槛，在学术界和工业界都有广泛的应用。但是由于其设计哲学，Python表达的灵活性弱于Scala，会更多地暴露Python自身的语法结构，在书写RTL时会有更强的调用类库的感觉，而Chisel可以做到接近原生语言的体验。

4.2 硬件设计框架

通过源码设计SoC时，不仅要关注模块的功能，更要关注模块互联的可配置化、透明化和自动化。

Rocketchip项目作为SoC生成器，设计了一套名为“外交（diplomacy）”的机制[21]，用于总线设备间的参数协商与转换。在电子设计自动化工具（如赛灵思的Vivado块设计）模式下，也有类似的参数协商功能。但是块设计的参数协商只能针对先进可拓展接口（advanced extensible interface，AXI）系列协议，且完全由闭源软件实现，只能通过图形界面或者复杂的TCL（工具命令语言）脚本进行操作。而Rocketchip 的片上设备互联框架则充分利用了Scala的泛型机制，理论上可以支持任意总线协议，只要定义好总线的端口结构、时序逻辑和参数变换规则，就可以复用互联框架的代码，生成属于该总线的互联框架，且该框架下的互联依然通过Scala代码来描述，更加灵活、便利。

除了描述SoC结构的自动化、可配置化外，对功能模块的接口进行统一规定，局部进行优化，也是提高硬件开发效率的一个方案。Zhang S等人[22]对乱序处理器的关键模块进行划分，对其接口提出了一套规范，给出了乱序处理器的设计框架。这样便可以针对局部模块进行优化和持续改进，以优化整体性能，而不需要对整个设计都完全掌控。

4.3 快速原型验证与调试

除了用传统的Verilator等仿真器进行功能验证外，UCB团队提出基于亚马逊FPGA云的仿真加速方案 Firesim[23]，可以以更高的效率对系统级的硬件代码进行验证和评估，核心直接以FPGA的形式运行，有较高的频率，对分布式网络的模拟则通过软件完成，频率相对较低。

此外，最新的工作（如Schkufza E等人的即时编译Verilog（Verilog-jit）[24]）则动态地在FPGA和软件仿真间测试硬件代码，提供了FPGA的速度和软件仿真的交互调试能力。它可以一边在主机和FPGA上进行功能仿真，一边将硬件描述代码综合成硬件结构，部署到FPGA上，并将软件仿真的逻辑信号转移至综合好的FPGA部分，以加速仿真。当到了需要进行受控调试的阶段，则可以将 FPGA 的信号迁移回软件侧，提供更细致的交互能力，从而进行调试。Verilog-jit的运行时架构如**图4**所示。

5 结束语

本文介绍了芯片领域面临的问题，通过对比互联网开源模式，说明开源芯片对芯片领域发展的重要意义，并介绍了在开

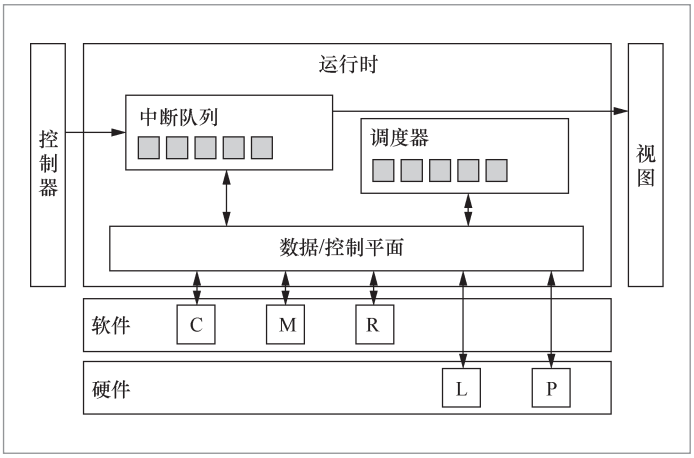

图 4 Verilog-jit 的运行时架构

源芯片生态发展中可能起到奠基作用的RISC-V指令集架构的优势及其影响。本文最后介绍了伴随开源芯片发展涌现出的与前端逻辑部分硬件敏捷开发相关的技术和方法论，通过高级编程语言的类型检查、复合类型、继承机制、函数式编程等特性，前端逻辑代码的复用程度、可读性和可维护性得到极大提升，使其工程化和开发效率达到了一个全新的水平。但仍然需要认识到，前端逻辑设计只占整个芯片开发的很小一部分，芯片开发效率和门槛依然严重受限于物理后端的发展水平。从设计到最终封装好的芯片，目前仍然严重依赖于商业化的EDA工具，要负担高昂的工具和关键IP授权以及流片的费用，试错成本仍然极高。要想进一步促进芯片领域的开源与创新，仍需要进一步发展开源EDA工具链，强化前后端的交流效率，并且在物理制造方面寻求新的突破。

参考文献:

[1] TOMOVICH C. MOSIS-a gateway to silicon[J]. IEEE Circuits and Devices Magazine, 1988, 4(2): 22-23.

[2] SENTOVICH E M, SINGH K J, LAVAGNO L, et al. SIS: a system for sequential circuit synthesis[R]. Berkeley: University of California, 1992.

[3] CONG J, PECK J, DING Y. RASP: a general logic synthesis system for SRAM-based FPGAs[C]//The 4th International ACM Symposium on Field-Programmable Gate Arrays, February 11-13, 1996, Napa Valley, USA. Piscataway: IEEE Press, 1996: 137-143.

[4] HACHTEL G, LIGHTNER M, BARTLETT K, et al. BOLD: the boulder optimal logic design system[C]//The 22nd Annual Hawaii International Conference on System Sciences, January 3-6, 1989, Kailua-Kona, USA. Piscataway: IEEE Press, 1989: 59-73.

[5] TANDON J. The openrisc processor: open hardware and Linux[J]. Linux Journal, 2011(212).

[6] GAISLER J. The LEON processor user's manual[Z]. Gaisler Research, 2001.

[7] PARULKAR I, WOOD A, HOE J C, et al. OpenSPARC: an open platform for hardware reliability experimentation[C]//The 4th Workshop on Silicon Errors in Logic-System Effects (SELSE), March 26-27, 2008, Austin, USA. [S.l.:s.n.], 2008: 1-6.

[8] NEHRA V, TYAGI A. Free open source software in electronics engineering education: a survey[J]. International Journal of Modern Education and Computer Science, 2014, 6(5): 15-25.

[9] ASANOVIĆ K, PATTERSON D A. Instruction sets should be free: the case for risc-v[R]. Berkeley: University of California, 2014.

[10] WATERMAN A, PATTERSON D. Design of the RISC-V instruction set architecture[R]. Berkeley: University of California, 2016.

[11] WATERMAN A, ASANOVIĆ K. The RISC-V instruction set manual, volume I: user-level ISA[Z]. 2017.

[12] ASANOVIC K, AVIZIENIS R, BACHRACH J, et al. The rocket chip generator[R].

Berkeley: University of California, 2016.
[13] ROSSI D, CONTI F, MARONGIU A, et al. PULP: a parallel ultra low power platform for next generation IoT applications[C]//2015 IEEE Hot Chips 27 Symposium (HCS), August 22-25, 2015, Cupertino, USA. Piscataway: IEEE Press, 2015: 1-39.
[14] SiFive, Inc. SiFive tilelink specification[Z]. 2018.
[15] BACHRACH J, VO H, RICHARDS B, et al. Chisel: constructing hardware in a scale embedded language[C]//DAC Design Automation Conference 2012, June 3-7, 2012, San Francisco, USA. Piscataway: IEEE Press, 2012: 1212-1221.
[16] LI P S, IZRAELEVITZ A M, BACHRACH J. Specification for the FIRRTL language[R]. Berkeley: University of California, 2016.
[17] NIKHIL R. Bluespec system verilog: efficient, correct RTL from high level specifications[C]//The 2nd ACM and IEEE International Conference on Formal Methods and Models for Co-Design, June 23-25, 2004, San Diego, USA. Piscataway: IEEE Press, 2004: 69-70.
[18] DAVE N, PELLAUER A M. Scheduling as rule composition[C]//The 5th IEEE/ACM International Conference on Formal Methods and Models for Codesign, May 30-June 2, 2007, Nice, France. Piscataway: IEEE Press, 2007: 51-60.
[19] DECALUWE J. MyHDL: a Python-based hardware description language[J]. Linux Journal, 2004(127): 84-87.
[20] CLOW J, TZIMPRAGOS G, DANGWAL D, et al. A pythonic approach for rapid hardware prototyping and instrumentation[C]//The 27th International Conference on Field Programmable Logic and Applications, September 4-8, 2017, Ghent, Belgium. Piscataway: IEEE Press, 2017: 1-7.
[21] COOK H, TERPSTRA W, LEE Y. Diplomatic design patterns: a TileLink case study[C]//The 1st Workshop on Computer Architecture Research with RISC-V, October 14, 2017, Boston, USA. [S.l.:s.n.], 2017.
[22] ZHANG S, WRIGHT A, BOURGEAT T, et al. Composable building blocks to open up processor design[C]//The 51st Annual IEEE/ACM International Symposium on Microarchitecture(MICRO), October 20, 2018, Fukuoka, Japan. Piscataway: IEEE Press, 2018: 68-81.
[23] KARANDIKAR S, MAO H, KIM D, et al. Firesim: FPGA-accelerated cycle-exact scale-out system simulation in the public cloud[C]//The 45th Annual International Symposium on Computer Architecture, June 1-6, 2018, Los Angeles, USA. Piscataway: IEEE Press, 2018: 29-42.
[24] SCHKUFZA E, WEI M, ROSSBACH C J. Just-in-time compilation for verilog: a new technique for improving the FPGA programming experience[C]//The 24th International Conference on Architectural Support for Programming Languages and Operating Systems, April 13-17, 2019, Providence, USA. New York: ACM Press, 2019: 271-286.

作者简介

王诲喆(1994-),男,中国科学院计算技术研究所计算机体系结构国家重点实验室先进计算机系统研究中心硕士生。目前负责标签化体系结构的研发及其在RISC-V平台的应用,主要研究方向为计算机系统的性能优化与实现。

作者简介

唐丹（1976- ），男，博士，中国科学院计算技术研究所高级工程师，中国计算机学会（CCF）会员，主要研究方向为计算机体系结构，长期从事高性能处理器I/O系统及超低功耗SoC芯片的研究与开发工作。在HPCA和JCST等高水平会议和期刊上发表多篇论文，获得多项国家发明专利。

余子濠（1991- ），男，中国科学院计算技术研究所计算机体系结构国家重点实验室先进计算机系统研究中心博士生，主要研究方向为计算机体系结构。

刘志刚（1995- ），男，中国科学院计算技术研究所计算机体系结构国家重点实验室先进计算机系统研究中心硕士生，研究方向为计算机体系结构。

解壁伟（1987- ），男，博士，中国科学院计算技术研究所助理研究员，主要研究方向为计算机体系结构、EDA技术和高性能计算等。

包云岗（1980- ），男，博士，中国科学院计算技术研究所研究员，计算机体系结构国家重点实验室教授，先进计算机系统研究中心主任，中国科学院大学岗位教授。担任中国计算机学会理事、普及工作委员会主任，中国科学院青年创新促进会理事。主要研究方向为计算机系统结构，主持研制多款达到国际先进水平的系统，在国际会议期刊发表了40余篇论文，相关技术在华为技术有限公司、阿里巴巴集团、英特尔公司等国内外企业应用，多次受邀担任ASPLOS、ISCA、MICRO、SC等国际会议的程序委员会委员。

收稿日期：2019-05-07

基金项目：国家重点研发计划基金资助项目（No.2016YFB1000201）；国家自然科学基金资助项目（No.61420106013）；中国科学院青年创新促进会优秀会员基金资助项目（No.2013073）

Foundation Items: National Key Research and Development Program of China (No.2016YFB1000201), The National Natural Science Foundation of China(No.61420106013), Youth Innovation Promotion Association of Chinese Academy of Sciences(No.2013073)