软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, [doi: 10.13328/j.cnki.jos.006065]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

FPGA 加速系统开发工具设计:综述与实践*

刘焰强, 戚正伟, 管海兵

(上海交通大学 软件学院,上海 闵行 200240)

通讯作者: 戚正伟, E-mail: qizhwei@sjtu.edu.cn

摘 要: 近年来,现场可编程逻辑门阵列(FPGA)在异构计算领域因其优异的可定制性和可重配置特点吸引了工业界和学术界广泛的关注.基于 FPGA 的硬件加速系统设计涉及到深度的软硬件协同开发,利用软硬件各自开发工具分别开发再集成的传统开发方式具有学习门槛高,集成、测试、部署耗时长等缺陷,开发人员难以利用 FPGA 可快速重配置的特点来实现系统开发过程中的快速原型和快速迭代.如何让硬件加速系统的开发利用到现代软件工程和程序语言领域的成果,研究者们已经经过了长期的探索,本文首先根据相关研究总结了硬件及硬件加速系统开发工具设计的历史教训和成功经验,然后介绍我们的设计实践,最后进行总结并提出对未来的展望.

关键词: FPGA;领域专用编程语言;软硬协同开发;硬件描述语言;高层次综合

中图法分类号: TP311

中文引用格式: 刘焰强,戚正伟,管海兵.FPGA 加速系统开发工具设计:综述与实践.软件学报,2020,31(10). http://www.jos.org.cn/1000-9825/6065.htm

英文引用格式: Liu YQ, Qi ZW, Guan HB. FPGA acceleration system development tools: survey and practice. Ruan Jian Xue Bao/Journal of Software, 2020,31(10) (in Chinese). http://www.jos.org.cn/1000-9825/6065.htm

FPGA Acceleration System Development Tools: Survey and Practice

LIU Yan-Qiang, QI Zheng-Wei, GUANG Hai-Bing

(School of Software Engineering, Shanghai Jiao Tong University, Shanghai 200240, China)

Abstract: Field-Programmable Gate Arrays (FPGAs) in heterogeneous computing have been attracting more and more attention due to its customizabilily and reconfigurability. Development of acceleration systems based on FPGAs involves the cooperation of both hardware and software developers. Building the systems by integrating software part and hardware part that are developed by independent tool chains introduces steep learning curve and difficulties in testing and deployment, thus preventing rapid prototyping. It has been a long academic history on how to make hardware design benefit from the progress in software engineering and software programming languages. This article will first present a survey on the design of development tools for hardware or hardware acceleration systems, and then will show the work of ourselves. Finally we draw a conclusion and discuss the future propect.

Key words: FPGA; DSL; Software/hardware Co-design; HDL; HLS

随着大数据分析和人工智能计算的大规模普及和应用,数据中心的计算负载规模在不断扩大[1].在此背景下,将计算任务进行分门别类,并根据不同任务的特点用专门设计的加速处理芯片取代通用处理器从长远来看成为了一种可行的降低成本或提高服务质量的方式.其中,为人工智能计算特化的通用图形处理器(General-purposed GPU)和应用专用处理芯片(Application-specific Integrated Circuit)TPU[2]已经在该领域取得了显著成就.但是,数据中心中除了日益庞大的人工智能应用,其他配套的支持计算也在相应的扩大规模,

* 基金项目: 国家自然科学基金(61672344, 61525204, 61732010); 国家重点研发计划(2016YFB1000502)

Foundation item: National Natural Science Foundation of China (61672344, 61525204, 61732010)); National Key Research & Development Program of China (2016YFB1000502)

收稿时间: 2020-02-09; 修改时间: 2020-04-04; 采用时间: 2020-05-09; jos 在线出版时间: 2020-06-10

这些方面包括数据库处理、网络通讯、虚拟化支持等等.利用专用芯片来加速这些基础计算同样能获得可观的收益.使用硬件加速的理想情况是能够直接部署经过专门优化的 ASIC,但是 ASIC 的设计、生产和部署都需要较长的周期和较高的成本,在需要快速迭代的生产环境中或者在需要控制风险的情况下,FPGA 作为一种可重新配置的硬件平台成为了一种首选的过渡或者替代方案[3,4].

FPGA 除了能够在生产环境中作为专用芯片的一种过渡和替代,它优秀的可定制性和可重配特点使其在异构计算的学术研究中也发挥着重要作用.与 ASIC 相比,基于 FPGA 的研究主要有两点优势:1）区别于 ASIC 的静态功能,FPGA 可重配特点引入的功能动态变化在系统设计时能创造更多可能；2）区别于 ASIC 的流片流程,FPGA 的快速烧写在降低了成本的同时也使应用可以快速的在硬件上验证并测试端到端的性能.这些特性近年来吸引了许多学者投入到 FPGA 的应用研究中,这些研究从最初的加速器设计慢慢扩展到了与 FPGA 相关的操作系统、体系结构等领域[5,6,7].

然而,面向 FPGA 硬件平台的硬件设计和系统集成方法相较于软件工程和程序语言领域的发展仍然处于十分初级的阶段.目前主流的 FPGA 加速系统的开发可以大致的分为三个部分:硬件加速器设计、软件驱动设计和软件应用集成.硬件设计虽然经过了多年的发展,已经形成了成熟的工具链,但是这些初衷是辅助专业硬件工程师的开发工具和语言只提供十分底层的硬件抽象,例如寄存器和逻辑运算.由于缺少指令集抽象,软件工程师无法直接将算法映射成硬件逻辑,对于希望快速验证硬件加速有效性的云应用开发者和实验室学者等非专业的硬件工程师来说,硬件设计无疑成为了利用 FPGA 加速上门槛最高的一关.而第二个难点是需要实现驱动软件来构建底层软件和硬件的数据通讯机制并提供接口给上层应用.这一部分工作即使利用现成驱动（例如 PCIe DMA 驱动）,也需要对底层通讯协议有充足的了解,并从软硬件两方面进行适配:软件需要对数据进行序列化和反序列化,硬件则需要构建加速器的外围数据通讯模块.相比于硬件加速核心的设计和验证,这一步的工程量并不逊色甚至犹有过之.克服前两点之后,需要经过最后一步的软硬件集成,开发者才能真正的对应用进行加速.此时,可能由于前期软硬件工程师协同的沟通失误或者考虑欠缺,应用的性能受限于不理想的硬件抽象和接口,如果需要改进,则往往需要重新进行各部分的设计.

FPGA 的高度可定制化给其带来灵活性的同时也引入了功能是由硬件来实现还是由软件来实现的权衡,如果开发框架不能提供清晰的软硬件交互界面和设计空间定义,加速系统容易陷入软硬件功能过于耦合,调试、迭代、维护十分困难的境地.为了提高 FPGA 加速系统的开发效率,研究者们在一脉相承的传统硬件开发工具基础上提出了许多针对 FPGA 特点的优化框架来应对不同的需求.从传统的硬件描述语言开始,这些新的框架和工具主要走上了两条道路:高层次综合（High-level Synthesis）和高级语言实现的硬件描述语言（High-level Hardware Description Language）.近年来,国内外 FPGA 应用的研究已经颇具规模[8,9],但是与国外传承有序的 EDA（Electronic Design Automation）研究历史相比,国内关于硬件开发工具与框架的研究资料依然十分缺乏,本文旨在对 FPGA 相关开发工具的发展状况做一定的总结和归纳,以提供从事本领域工作的国内学者参考.接下来,本文第一章将综述近年来硬件及硬件加速系统开发工具相关的典型工作,第二章介绍我们自己针对 FPGA 加速系统开发框架的设计实践,第三章总结全文并提出未来展望.

1 相关工作综述

本节介绍一般硬件以及硬件加速系统开发工具和框架的相关工作,并引入硬件加速系统设计的基本概念和背景知识.

1.1 硬件描述语言

广义来说,后文将提到的高层次综合工具使用的描述语言和高级硬件描述语言都可以归纳进硬件描述语言,本节讨论狭义的硬件描述语言,即以 VHDL 和 Verilog 为代表的经典硬件描述语言.

硬件描述语言的出现可以追溯到上世纪 60 年代,在 1971 年,C. Gordon Bell 和 Allen Newell 在他们的教材中提出了影响至今的硬件设计抽象级别——寄存器传输级[10]（Register-transfer Level）.基于该抽象,DEC（Digital Equipment Corporation）首先设计了 ISP 语言以及其扩展版本来描述 PDP-8 和 PDP-16 机器中的基于

寄存器传输模块（Register-tansfer Module）的硬件行为[11].在 1979 年左右,Kaiserslautern 大学的团队开发出 KARL,来支持结构化硬件设计和日益兴起的超大规模集成电路（VLSI）设计[12].同时间,为了满足同样兴起的可编程逻辑设备（PLD）的商业化需求,DATA I/O 公司也开始了设计面向可编程设备的 ABEL[13].

随着超大规模集成设计的逐渐流行和现场可编程设备的发明,传统的基于电路图绘制（Schematic Capture）的硬件设计方法受限于适合处理的晶体管数量（通常是数百个）已经越来越难以满足设计效率的需求,在上世纪 80 年代,Verilog[14]和 VHDL[15]相继出现并逐步发展成熟.VHDL 和 Verilog 的设计初衷都是希望用开发者熟悉的编程语言来描述设计好的电路图以便于工程管理,其中 VHDL 借鉴了大量 Ada 语言的理念和语法而 Verilog 借鉴了 C 语言的编程风格.相比于其他的硬件描述语言,类 Ada 和类 C 的语言风格更受开发者青睐,随着 VHDL 和 Verilog 的标准化,两者渐渐取代了其他的硬件描述语言,并统治硬件设计领域直到今日.初期版本的 VHDL 与 Verilog 的功能十分类似,相比于 Verilog,VHDL 有更高的抽象级别和语言复杂度.但是进入 21 世纪以后,经过 Verilog 到 System Verilog 的演变,以及多次迭代（最新标准是 IEEE Standard 2017[16]）,Verilog 引入了许多提高生产效率的语言特性,并提高了对大规模系统的描述能力.在此基础上又由于 C 语言使用的广泛性,Verilog 相比于 VHDL 得到了更大规模的应用.

基于寄存器传输级抽象的硬件描述语言在逻辑综合技术引入之前并不足以完成完整的硬件设计,VHDL 和 Verilog 初期都只是用于描述和归档门电路级的电路图设计.类比高级软件编程语言,逻辑综合器可以被理解为编译器,他将寄存器传输级的设计翻译成门电路级的网表设计.但是与高级语言的编译器不同,由于没有统一的指令集抽象和确定的硬件布局,描述门电路连接方式的网表还不足以在硬件上“运行”.以一般 FPGA 开发为例,我们还需要经过“布局布线”（Place and Route）步骤,将网表与特定硬件平台上的物理资源对应起来,并通过下载配置才能让 FPGA 实现我们想要的功能（放置与路由由于涉及到 FPGA 硬件的技术细节,目前这一步一般都是由 FPGA 生产商提供的闭源软件来实现,因此本文讨论的开发框架将不包括这部分内容）.系统设计的经验告诉我们,提高抽象层次的代价往往是性能的损失,而这点在硬件设计上也没有例外.基于硬件描述语言和逻辑综合器生成的设计相比于有经验工程师使用传统方法完成设计,几乎总是需要更大的硬件面积和提供更低的性能.然而随着硬件集成度的提高,硬件本身性能的发展和综合算法的改进,人们在设计质量和开发效率的权衡中,最终接受了高开发效率的工具,这点与软件编程语言的发展是类似的.

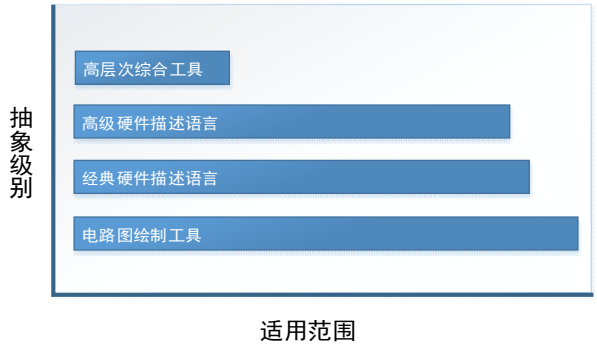

Fig 1 The hierarchy of development tools

图 1 开发工具层次

硬件描述语言经过三十多年的使用,尽管自身依然在迭代和发展,但面对新的需求和挑战,它们遇到了与自己前身类似的问题:语言本身的局限性限制了开发效率的进步.近年来 FPGA 应用研究的兴起吸引了大量非专业硬件背景的工程师投入到 FPGA 的开发中,不同于 C、Ada 等底层语言,他们熟悉的工作语言经常是 C++、Java、Scala 等面向对象语言或者是 Python、JavaScript 等脚本语言.对于新的使用群体来说,硬件描述语言曾经

语言友好的优势不复存在,而另一方面,区别于硬件工程师是从门电路级抽象提升到寄存器传输级抽象,软件工程师却是从算法级抽象降低到有明显语义鸿沟的寄存器器传输级抽象,抽象级别的优势也不复存在.而对于传统的硬件工程师来说,现代软件编程框架中体现的软件工程理念同样吸引着他们,经典硬件描述语言中缺乏面向对象编程,参数化类型,类型安全等特性,这在项目规模不断扩大的今天,已经开始对项目管理、维护造成巨大困扰.为了适应新的需求,在经典硬件描述语言的基础上,新的硬件开发工具或框架基本走上了两条道路:高层次综合[17,18,19]和高级硬件描述语言[20].并形成了如图 1 所示层次,接下来将依次介绍高层次综合和高级硬件描述语言.

1.2 高层次综合

事实上,在基于寄存器传输级抽象的硬件描述语言流行之前,学术界就已经开始了将算法级或行为级描述映射成硬件设计的研究.这些早期的工作大约从上世纪七十年代一直持续到九十年代初,在这段时间涌现了许多论文和学术著作.其中影响力较大的是 Pierre Paulin 与 John Knight[21],Raul Camposano 与 Wayne Wolf[22],Dan Gajski[23]以及 Giovanni De Micheli[24]等学者的工作.这些工作探索了高层次综合的基本原理和概念,并发展出了高层次综合工具的基本特征:专用输入语言加编译综合器.尽管对后世影响深远,这些工作的商业化尝试都失败了.在当时的背景下,硬件工程师才刚开始接受寄存器传输级的设计抽象,硬件描述语言都还没有普及,高层次综合技术的引入过于超前,具体来说该技术当时有以下两个问题:

(1) 较高的学习成本和较低的硬件设计质量.高层次综合工具一般使用特别定义的行为或算法描述语言作为输入语言,这些语言与硬件描述语言相比更加专用化且没有通用的规范,基本依赖于厂商的支持.同时在可以预计的硬件设计质量下降的前提下,开发人员没有学习的欲望;

(2) 适用的专用领域没有广泛的市场.当时的许多高层次综合工具是为数字信号处理器(DSP)的开发设计的,这类设计的特点是专注数据流和结构化的硬件,这在大部分开发人员都在进行非结构化随机逻辑集成这类侧重控制流的设计的背景下是不合时宜的.

虽然早期学术成果商业化的尝试失败了,但是高层次综合提升开发效率的前景以及硬件描述语言蓬勃发展的现状依然吸引了 Synopsys,Cadence 和 Mentor Graphics 这些大型的电子设计自动化(EDA)公司投入到了高层次综合工具的研发中,典型的代表包括 Synopsys 公司的 Behavioral Compiler[25],Cadence 公司的 Visual Architect 和 Mentor Graphics 公司的 Monet tool[26].在上世纪九十年代的中后期,他们的产品一度吸引了广泛的关注.这些工具的特点是使用单独设计的行为级描述语言作为输入语言,并直接生成门电路级的设计.不过这次尝试依然没能取得期待中的成功并且在之前的基础上暴露出了有关产品定位的新问题:

(1) 错误的目标群体,并以经典硬件描述语言为竞争对手.这时期的商业产品以正在接受寄存器传输级抽象,开始使用 VHDL 和 Verilog 等硬件描述语言的硬件设计师为目标用户,提出了基于算法或行为级抽象的新语言和相应的综合器.这些工具直接从高抽象级别的描述综合生成门电路级的设计,与硬件描述语言并不兼容且互相替代,这迫使工程师们必须从中做出选择.在已经逐步接受硬件描述语言的背景下,当时的硬件工程师们关心的问题是,新的工具能否能用相同的工作量设计出更高质量的硬件,和是否能在硬件设计质量不变的情况下减少工作量,以及学习曲线是否陡峭.遗憾的是,新的工具不仅学习曲线陡峭,而且几乎不能提高硬件设计质量或者是降低工作量;

(2) 没有清晰的界定基于控制流的设计和基于数据流的设计,导致没能发挥出自身优势.控制流设计多包含随机逻辑和跳转结构,而数据流的设计则多是结构化的硬件.基于寄存器传输级的综合器能同时处理好控制流和数据流的设计.高层次综合由于有更高的抽象级别,很难生成性能理想的复杂控制流硬件,但对于更适合抽象和采取针对优化的数据流设计,高层次综合往往能生成性能良好的硬件.遗憾的是,这时的高层次综合工具没能专注于自身的特长,反而试图在一个框架能完成各种类型的硬件设计,以致失去了自身的竞争优势.

时间进入 21 世纪,由于超大规模集成电路的持续发展,Verilog 和 VHDL 的开发效率达到了瓶颈,同时,由于异构计算和硬件加速的兴起,大量软件工程师和系统架构师开始尝试协同设计.在软硬件工程师对硬件开发效

率提高的共同需求下,之前高层次综合推广的失利没能阻止工业界和学术界的新一轮尝试.与之前的环境相比,在新世纪高层次综合的应用有了以下优势:

(1) 大数据处理相关应用的规模不断扩大,针对特定类型应用的硬件设计工具拥有了可观市场,尤其是有关人工智能和数据库的应用.这些应用在规模大的同时更侧重数据流处理,让高层次综合有了用武之地.

(2) 经典硬件描述语言开发效率已经到达瓶颈,开发人员有了对更高层次抽象的明确需求.

在这一阶段,不断有新的高层次综合工具取得学术上和工业上的成功,以致直到 2019 年依然不断有新的工具被提出,其中影响较大的包括 Bluespec[27],LegUp[28,29],Vivado HLS[30]和 Intel HLS Compiler[31].新的工作吸取了之前工作的教训,大多做出了如下改进:

(1) 主要使用或支持 C,C++等广泛使用的通用编程语言作为输入语言.这使得工具具有更广泛的受众并降低了学习成本;

(2) 以 Verilog 或(和)VHDL 为输出结果.这使得新的工具可以兼容老的设计,并在生成的硬件性能不理想时提供了手动优化的可能,进而丰富了工具的适用场景;

(3) 针对特定类型设计的综合算法进行专门优化.以牺牲通用性换取性能,这强化了高层次综合的优势.

到目前为止,我们讨论的工作都是针对单独的硬件设计,对于传统的硬件来说,从硬件设计到硬件应用中间有较长的时间跨度,也会由各自的团队来完成.但是随着 FPGA 的兴起,由于 FPGA 可以通过快速的下载配置直接形成对应的硬件,设计和部署的分工变的模糊起来,人们需要集成的框架来快速构建加速应用.而要构建一个完整的加速系统,如引言中提到的,我们还需要硬件驱动和软件集成.将驱动和集成功能包含在一个框架下,在我们不知道用户会设计拥有什么功能的硬件和使用什么样的接口进行数据传输的情况下是十分困难的.如果设计十分底层的接口来提高适用性和灵活性,则会依然保留较多的工作量给开发者;而如果规定接口协议和做各种假设来提高抽象层次,则会对用户的硬件设计造成各种限制.随着一些高层次综合工具开始针对性地处理特定应用,和 OpenCL 等计算框架的兴起,工具开发人员不再面对硬件功能未知的情况,而完整的开发框架设计也变得可行.

Xilinx 和 Intel 分别推出了 SDAccel[32]和 Intel SDK for OpenCL[33],用于将 OpenCL 定义的加速器部署到 FPGA 上,并包含基于 PCIe 的主机与 FPGA 数据传输机制的集成.对于特定领域的应用,DnnWeaver[34]和 Inter OpenVINO[35]可以根据的 Caffe,TensorFlow 等机器学习框架定义的模型自动生成硬件并部署到主机+FPGA 的加速平台上,而 p4FPGA[36]框架可以将用 p4 语言定义的数据面板部署成 FPGA 包处理器.这些框架相比于通用的高层次综合工具更受系统架构师和软件工程师青睐,它们能显著降低开发门槛,提高开发效率,方便系统的早期验证.虽然这些框架生成的硬件设计与使用硬件描述语言完成的设计相比依然存在不小的性能差距,但是从硬件描述语言的发展历史来看,对开发效率的需求将推动综合算法的改进,同时由于硬件本身性能的进步,更高生产效率的工具将逐渐取代生产效率低的工具.

1.3 高级硬件描述语言

高层次综合工具在侧重数据流处理的加速系统中得到了良好应用,但对于控制流复杂的硬件设计,比如非结构化的硬件设计和基于指令集的处理器设计,高层次综合工具往往需要与其他寄存器传输级设计工具相互配合.因此,在对硬件性能要求较高,需要在微架构上进行创新的设计情景下,寄存器传输级设计工具依然在发挥着重要作用.将经典硬件描述语言使用新的高级编程语言重新包装以获得高级语言在编程效率,维护管理上的优势同样吸引了大量研究人员.本文将提供寄存器传输级抽象设计接口,并使用高级编程语言作为输入语言的硬件描述语言归纳为高级硬件描述语言.这些高级硬件描述语言通常会利用高级语言中的面向对象、语法糖等机制来扩展自身的数据结构和提供更便利的语法接口,或者以库的形式来模板化特定类型硬件的设计.在一定程度上,这些方法提供了更高层次的设计抽象,但与高层次综合不同,这种抽象更多的是语言机制上的抽象,它没有隐藏硬件数据传输的底层机制,也就是说没有沟通软硬件描述的语意鸿沟.因此,我们认为目前大部分高级硬件描述语言依然处在寄存器传输级抽象.表 1 根据输入语言的特征列举了 2000 年以来典型的高级硬件描

述语言:

Table 1 High-level HDLs classified by characteristics of input languages

表 1 根据输入语言特征分类的高级硬件描述语言

编程风格	语言	年份	实现语言	实现方式
函数式	HML[37]	2000	SML	非内嵌
	Lava[38]	2000	Haskell	内嵌
	Clash[41]	2009	Haskell	非内嵌
命令式	JHDL[39]	2000	Java	非内嵌
	MyHDL[43]	2004	Python	内嵌
	RHDL[40]	2006	Ruby	非内嵌
	PyMTL[44]	2014	Python	内嵌
	Pyverilog[45]	2015	Python	内嵌
混合式	Chisel[42]	2012	Scala	内嵌
	SpinalHDL[46]	2015	Scala	内嵌

函数式编程风格一度被认为是最适合进行硬件描述的编程风格,函数式语言作为输入语言的开发工具几乎涵盖了硬件设计的各个抽象层次,其中一部分工作可以被归纳为高层次综合工具,例如 Bluespec,还有一部分则可以被归纳为高级硬件描述语言,例如表格中所示.函数式硬件定义的好处十分突出:简明的语义结构便于形式化验证和对递归的良好支持.这些优点使得函数式硬件描述语言在定义大规模组合逻辑电路时十分简练而易于测试验证.但是,在定义时序电路时,函数式定义却不是那么直观,以至于各种工具都采取了不同的定义方式,而没能达成一致的抽象.由于缺乏标准,学习成本过高,函数式编程在应用开发上使用较少等因素,在当前背景下,函数式硬件描述语言大多成为了学术语言,较少在实际生产中得到应用.

另一方面,基于 Java、Python、Scala 等支持面向对象编程和命令式语言风格的高级语言构建的硬件描述语言,由于本身语言更优秀的生态和更广泛的用户基础,近年来我们能观察到持续的更新和更深入的发展.尤其是内嵌于高级语言的硬件描述语言,因为继承了原本语言的语法,有更低的学习成本,吸引了更多开发者的使用.其中基于 Scala 语言的 Chisel[42],已经形成了硬件定义和测试一体的集成框架,并通过对 RISC-V 设计的特化支持,扩展了自身的硬件设计生态.

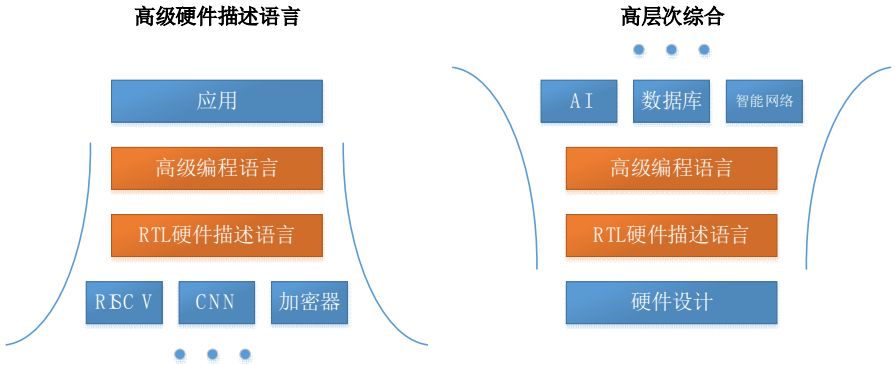

Fig 2 The prospectives of High-level HDL and HLS

图 2 高层级硬件描述语言和高层次综合的发展方向

高级硬件描述语言总体来看目前还处于学术研究阶段.一方面的原因是这些语言还没有形成充实的生态基础,无法吸引硬件工程师付出学习成本进行转型;另一方面的原因则是这些语言并不专注于提供特定的抽象和完整的集成框架来降低开发门槛,使得对软件工程师也没有足够的吸引力.如图 2 所示,相比于高层次综合工具正在向应用层特化发展,逐步让底层硬件细节透明化不同,高级硬件描述语言则在确定了寄存器传输级抽象的基础上,在扩充自身的硬件实现生态,以形成类似于 C++,JAVA 等语言的丰富库支持.接下来的章节将介绍我们在高级硬件描述语言方向上的实践工作,其中包括高级硬件描述语言 ScalaHDL,以及基于 ScalaHDL 的系统

集成开发框架 VeriScala[47].

2 基于高级硬件描述语言的 FPGA 加速系统开发框架设计

在硬件加速的学术研究中,我们经常需要进行微架构的创新,常常遇到 Verilog 或 VHDL 学习门槛高,开发周期长,软硬件集成麻烦的困难.Xilinx 推出了 Vitis 工具,通过将高层次综合和细粒度的 RTL 优化集成在一个框架下来满足这样的需求.但是,Vitis 虽然提供了统一的框架,开发者依然需要使用多种的语言来进行综合开发,这显然增加了工程维护和管理的难度.在高层次硬件描述语言的基础上构建集成开发框架可以很好解决这个问题.本节将介绍我们实现的 ScalaHDL 硬件描述语言,和基于 ScalaHDL 的开发框架 VeriScala.

2.1 设计目标

基于高级硬件描述语言的加速系统开发框架设计目标主要有两方面:充分利用高级语言在开发效率、维护管理上的优势和提供简便的集成策略.具体来说有以下几点:

(1) 支持丰富的代码重用机制.需要提供包括继承,参数化类型在内的特性来提高代码重用率和可维护性.

(2) 易用的语言抽象.一方面需要通过支持面向对象编程来进行模块化设计,另一方面还需要将 RTL 抽象在高级语言进行映射来帮助软件工程师理解.

(3) 支持库的设计和使用.需要通过支持硬件设计的引用方法,让现有设计逐渐形成丰富的生态,进一步提高生产效率.

(4) 支持软件模拟.需要设计对接的软件模拟器,让高级语言定义的硬件设计直接利用高级语言的测试框架进行测试.

(5) 支持快速的系统集成和部署.需要在框架内处理好底层硬件数据传输机制,并提供简洁的软件接口.

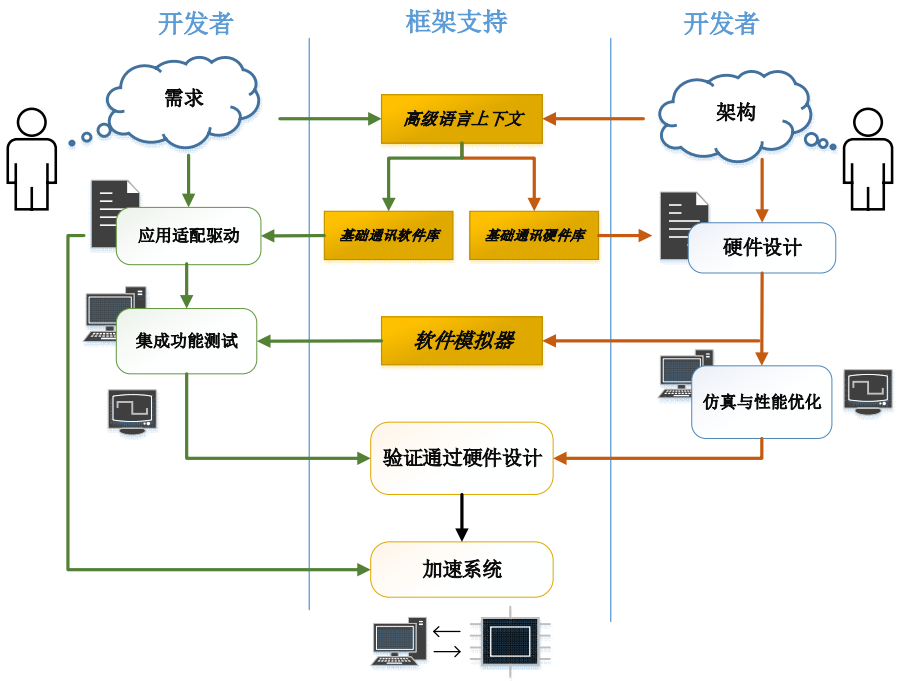

Fig 3 An ideal workflow

图 3 理想开发流程

理想的设计流程如图 3 所示,软硬件设计在同一种高级语言的上下文中,其中软件计算需求可以通过函数

接口来定义,而硬件设计则可以使用内嵌的高级硬件描述语言来定义.框架本身将包含通讯相关经过适配的软硬件库,基于基础通讯驱动可以实现应用的后端,而基于硬件通讯子系统可以实现加速硬件的通讯外围.接下来,完成的硬件设计可以直接输入给软件模拟器进行硬件功能测试,而软件模拟器又可以作为模拟硬件与驱动对接,完成整个加速系统的功能测试.最后用验证过的加速硬件替换模拟硬件就完成了加速系统的集成.

面向上文的设计目标和开发流程,我们设计实现了基于 ScalaHDL 语言的 VeriScala 框架.后面将在 2.2 节介绍 ScalaHDL 语言的设计,在 2.3 节介绍 VeriScala 中软硬件协同的集成支持,以及在 2.4 节展示 VeriScala 的代码示例和实验结果.

2.2 输入语言与硬件设计抽象

ScalaHDL 是一种内嵌于 Scala 语言的硬件描述语言,它被实现成了 Scala 库而不需要修改 Scala 编译器.选择 Scala 作为框架输入语言是因为 1)Scala 是一种多范式的编程语言,既可以支持面向对象编程,也能良好的支持函数式编程,提供了更多设计可能性;2)Scala 基于 Java 虚拟机运行,可以受益于丰富的 Java 生态.如图 4 展示了 ScalaHDL 的整体框架,主要分成硬件定义和测试两个模块.ScalaHDL 使用高扩展性、可插拔的模块化设计,核心功能实现在 HDLBaseClass 和 SimulationSuite 两个类中(图中红色模块),分别支持硬件的定义和测试.而扩展的语言特性,例如新的硬件描述语句语义或者更高的设计抽象则可以实现在 Scala 的 Trait 定义之中(图中浅绿模块),只需的在硬件定义时指定需要的 Trait,即可使用对应的特性.在 ScalaHDL 中我们实现了 BasicOps 来支持基本的硬件定义功能,以及 Translator 来完成 ScalaHDL 到 Verilog 的翻译.

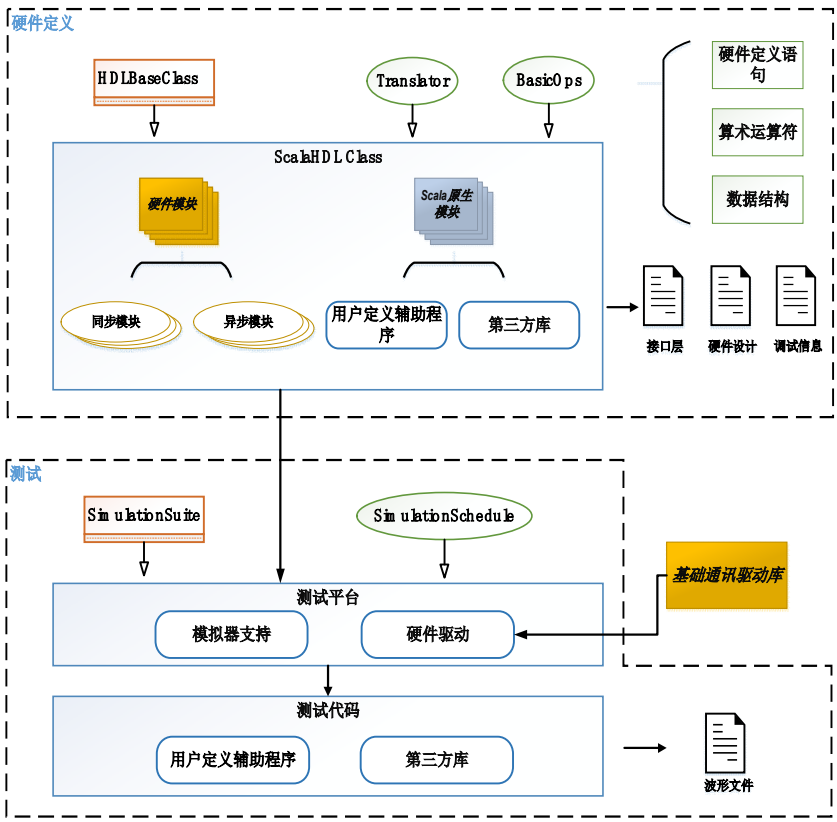

Fig 4 The architecture of ScalaHDL

图 4 ScalaHDL 架构

ScalaHDL 使用 Scala 语言中原生的 Class 来作为硬件模块定义的容器,通过继承 HDLBaseClass 并引用 BasicOps 特性,开发者可以在原生的 Scala 类中使用 BasicOps 中定义的类型、语句和结构来描述硬件设计.ScalaHDL 的设计应用了 Lightweight Modular Staging[48]的思想,即 ScalaHDL 的翻译通过 Scala 的反射机制,依据运行时扫描类定义中实体得到的类型信息,来生成对应的 Verilog 描述.在一个硬件定义类中,我们会包含两部分代码:硬件定义和原生 Scala 语句.翻译时,包含 ScalaHDL 特殊硬件类型的语句会被识别为硬件描述,而原生的 Scala 代码则会被执行并发挥类似于编译指令的功能.这样的方式使我们可以很方便的利用类型系统,将 RTL 抽象在 Scala 中进行了映射.具体来说,硬件定义实体的类型可以为模块、语句块、语句和值,其中语句块包括同步语句块和异步语句块.

SimulationSuite 和 SimulationSchedule 结合实现了软件模拟功能.由于我们在硬件类型实现中包含了不同类型在模拟时的行为,因此我们可以通过事件接口直接驱动硬件定义类实例的运行来实现功能模拟,这极大的简化了软件模拟器的实现.通过与实现好的基础通讯驱动对接,测试平台也可以针对真实硬件进行测试.

2.3 底层数据传输与集成方法

要提供简便的软硬件集成支持,基本思路是通过抽象和包装将繁琐的底层数据传输细节隐藏起来.在 Verilog 中我们通过用 Scala 实现的基础通讯驱动和根据用户模块定义自动生成的硬件数据接口层来完成软硬件的对接和传输细节的隐藏.图 5 展示了 VeriScala 框架中的运行时系统.在软件方面,Scala 实现的基础通讯驱动包装内核中的 PCIe DMA 驱动,将上层 Scala 定义的数据结构序列化后传给内核模块,并通过相应的接口控制 PCIe 通道的负载.应用可以在基础通讯驱动的基础上开发特定应用的数据传输后端来实现类似函数调用语义的硬件调用,也可以直接使用基础驱动来进行数据传输.在硬件方面,我们设计了通用的接口层实现库来处理的底层传输协议握手和数据缓存.通过用户顶层设计的接口定义,ScalaHDL 的翻译器可以在生成硬件设计时自动识别接口信号并进行相关的适配.同时在调试时,用户可以指定需要监视的信号,让翻译器额外生成调试服务硬件,通过与软件驱动配合来实现类似于 GDB（GNU Debugger）风格的硬件调试工具.

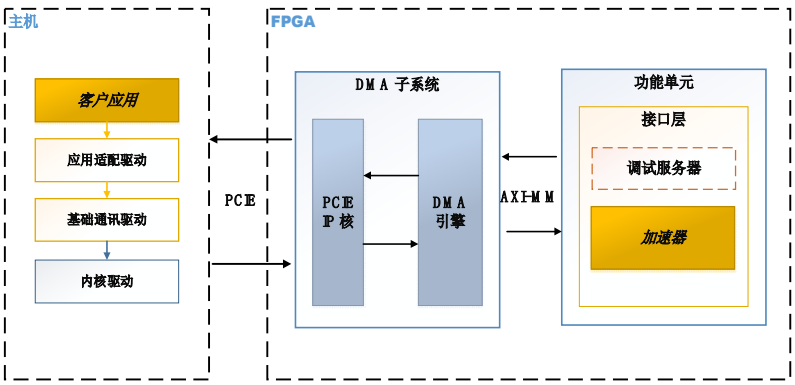

Fig 5 The runtime system

图 5 运行时系统

2.4 示例与实验结果

基于之前两节的阐述,VeriScala 框架中形成了如图 6 所示的代码流程.图 6 中以简单的加法器定义为例,在文件 *Adder.scala* 中首先定义模块的容器类 *Adder* 及其构造接口,然后在类中定义模块 *add*,*add* 的逻辑由一个同步块来描述,由于加法器的功能简单,该同步块中仅包含一个条件分支语句.事实上,在 Scala 中,我们可以将运算函数作为参数来传递,如文件 *Arithmetic.scala* 中定义的 *Arithmetic* 类可以在实例时接受一个函数 f 作为参数来生成不同的算术模块,这对于硬件定义代码的复用提供了方便.对于定义好的硬件,开发者可以利用 VeriScala 中

的软件模拟器,通过编写测试平台和测试代码,进行软件模拟,也可以通过生成的一系列 Verilog 代码,利用对应厂商的工具链和 VeriScala 的运行时框架进行硬件测试和加速系统构建.

为了评估 ScalaHDL 的生成代码质量,我们针对不同复杂度的常用电路分别使用 ScalaHDL 和 Verilog 进行定义,并根据 Xilinx Vivado 软件编译自动生成代码和手动编写代码的资源消耗报告,得到统计结果如表 2 所示.

Table 2 Resource consumptions of both variants (generated Verilog code and direct Verilog code).

表 2 生成的 Verilog 代码与手写 Verilog 代码的资源消耗对比（生成 Verilog 代码/手写 Verilog 代码）

模块		#LUT	#Register	模块		#LUT	#Register
基础电路	RAM	3/3	3/3	常用电路	Encoder	2/2	0/0
	ROM	0/0	2/2		Mealy FSM	3/3	1/1
	FIFO	9/9	17/17		Moore FSM	11/11	7/7
	ADD	5/5	6/6		Hash Map	119/119	111/111
	SUB	5/5	6/6		Bitonic Sort	40/40	0/0
	AND	1/1	1/1	复杂电路	MIPS CPU	1237/1237	1112/1112
	OR	1/1	1/1		SQL Filter	417/417	198/198

我们发现,ScalaHDL 和 Verilog 在定义表中电路时完全消耗相同的硬件资源.由于 ScalaHDL 与 Verilog 处于相同的抽象层次,这样的结果与预期吻合.

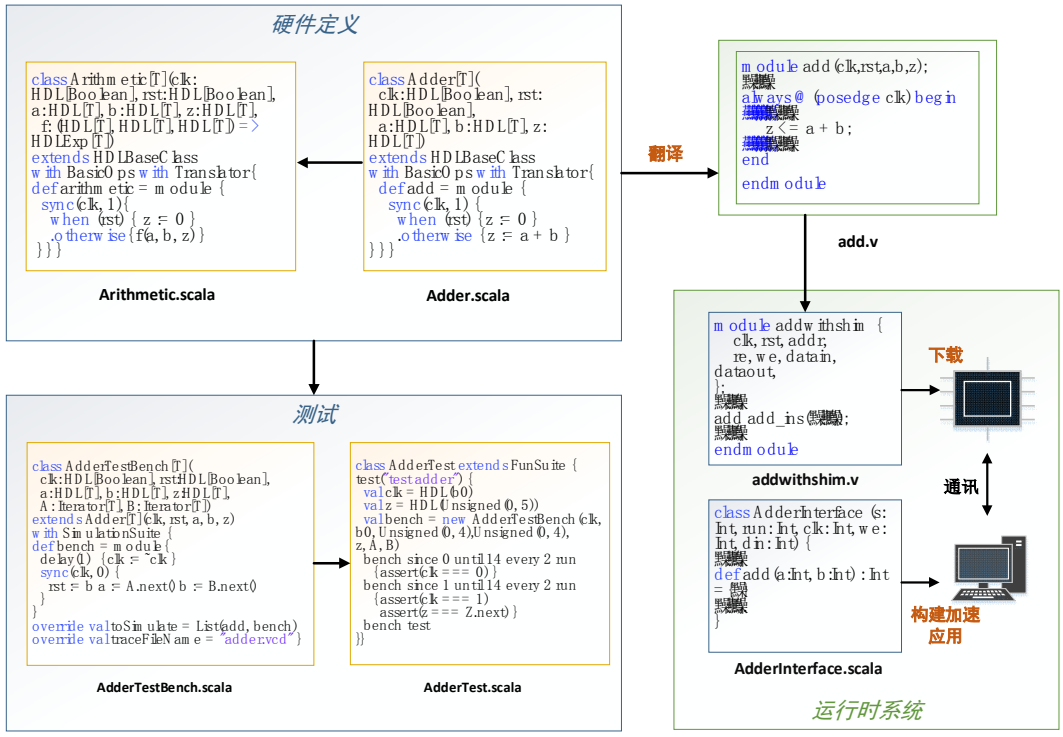

Fig 6 Code overview

图 6 代码流程总览

为评估 VeriScala 框架的实用性,我们基于 VeriScala 构建了典型的数据库过滤器加速系统,该应用使用 Scala 编写,将 Scala 管理的主机数据库表项发送给 FPGA 处理,再读回过滤后的数据.数据传输通过 PCIE 3.0 接口完成.作为对比基准,我们同样基于 Scala 实现了使用 CPU 完成的过滤程序.该测试使用的硬件配置如表 3 所示.

Table 3 Experiment environment

表 3 测试环境

类型	型号	核心数	频率	功率
CPU	I5 4590	4	3.3GHz	84w
FPGA	Kintex-7 XC7K325T	1	250MHz	3.534w

Table 4 Performance comparison between FPGA accelerated filter and CPU filter

表 4 FPGA 加速过滤器和 CPU 过滤器的性能对比

块规模 (KB)	4	8	16	32
CPU (ms)	1559	1559	1559	1559
FPGA (ms)	1675	1198	996	902
加速比	0.93X	1.30X	1.56X	1.73X

通过统计软件应用过滤 512MB 随机数据的平均（1000 次）任务运行时间,我们得到了表 4 展示的随单次传输块大小增加而变化的加速比,受限于实验使用的 FPGA 资源总量和我们采用的过滤器设计,单次传输 32KB 是我们能在 250MHz 下运行的最优值.从实验结果来看,VeriScala 框架能有效构建 CPU+FPGA 的加速系统.

3 总结和展望

2009 年,Grant Martin 和 Gary Smith 在他们的综述文章中谈到,当时不存在一种开发框架能够同时支持所有的领域:控制流和数据流,ASIC 和 FPGA,随机逻辑和结构化模块,以及硬件、软件和软硬协同各自的设计.他们指出,一个完整的能够探索整个设计空间的集成框架是设计师们的终极理想,同时也是未来的发展方向[18].经过十年的发展,我们已经形成了如图 1 所示的工具层次,高层次综合和高级硬件描述语言似乎都有潜力达到这一终极目标,而又有各自的不足.其中高层次综合有难以处理复杂控制流和随机逻辑的缺点,使其需要高级语言框架外的经典硬件描述语言来辅助优化；而高级硬件描述语言则依然处于较低的抽象层次,其开发效率瓶颈和软硬件的语义鸿沟没有完全消除.那么一个直接的想法是,这两者能不能结合起来呢？

从图 2 中我们可以发现,高级硬件描述语言和高层次综合并不完全独立.高级硬件描述语言将高级语言定义的寄存器传输级描述翻译成 Verilog 或者 VHDL,而高层次综合则是将高级语言定义的计算模型翻译成 Verilog 或者 VHDL.如果我们在高级硬件描述语言的基础上构建高层次综合工具,使得高层次综合首先生成高级硬件描述语言定义的硬件,然后再生成对应的经典硬件描述语言代码,或者我们将高层次综合功能实现为高级硬件描述语言的一个特殊库,是不是这样就能享受两者共同的优势呢？基于这个想法我们可以画出图 7 所示架构.

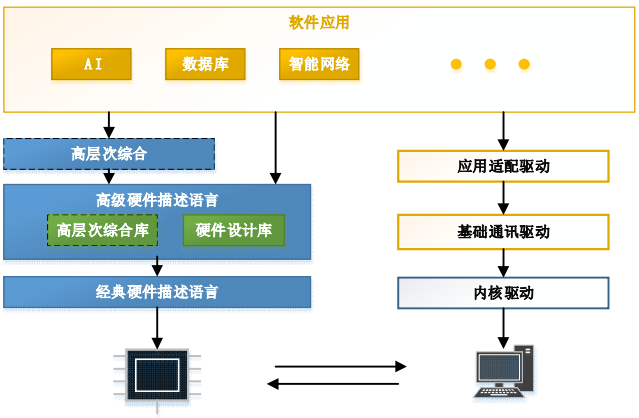

Fig 7 The future architecture

图 7 未来架构

在该框架中,我们可以在同一种高级语言的上下文中进行硬件与软件设计,一方面硬件设计可以兼顾控制流与数据流,另一方面软硬件集成变成了简单的库调用.该架构的实现在很大程度上可以复用现有的研究成果,以 ScalaHDL 为例,要想支持某种特定硬件的高层次综合,需要做的是实现一个 Scala Trait 来扩展已有的硬件类型和硬件定义语句,以及定义这些新的抽象在翻译时的机制.这点对于许多基于 Java、Scala 等可扩展性强的语言实现的框架来说都会有类似的解决方案.

硬件及硬件加速系统开发工具的选择取决于开发质量与开发效率的平衡,当硬件资源是瓶颈时,硬件设计质量发挥更大影响,而当硬件资源不是瓶颈时,开发效率发挥更大影响.因此,在开发工具的发展历史中,一种抽象层次不会被轻易淘汰,人们往往希望使用提供更大设计空间的开发工具来面对复杂的需求与限制.在当前环境下,我们无法判断是高层次综合还是高级硬件描述语言会得到更广泛的应用,两者可能会长期共存.因此,可以推测,类似于图 7 的综合开发框架是未来合理的发展方向.

References:

[1] Index C G C. Forecast and Methodology, 2018–2023 White Paper.

[2] Sato K. An In-Depth Look at Google's First Tensor Processing Unit (TPU). Google Cloud Big Data and Machine Learning Blog. Google Cloud Platform, 2017.

[3] Niemiec G S, Batista L M S, Schaeffer-Filho A E, et al. A Survey on FPGA Support for the Feasible Execution of Virtualized Network Functions. IEEE Communications Surveys & Tutorials, 2019.

[4] Yang HG, Sun JB, Wang. An Overview to FPGA Device Design Technologies. Journal of Electronics & Information Technology. 2010, 32(3).

[5] Vaishnav A, Pham K D, Koch D. A survey on FPGA virtualization. 2018 28th International Conference on Field Programmable Logic and Applications (FPL). IEEE, 2018: 131-1317.

[6] Kachris C, Soudris D. A survey on reconfigurable accelerators for cloud computing. 2016 26th International conference on field programmable logic and applications (FPL). IEEE, 2016: 1-10.

[7] Khawaja A, Landgraf J, Prakash R, et al. Sharing, protection, and compatibility for reconfigurable fabric with amorphos. 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI). 2018: 107-127.

[8] Jeff Dorsch. FPGAs for All Seasons. Applications of IC, 2018(1):77-79.

[9] Wu YX, Liang K, Liu Y, Cui HM. The Progress and Trends of FPGA-Based Accelerators in Deep Learning. Chinese Journal of Computers. 2019(11).

[10] Bell C G, Newell A. Computer structures: Readings and examples. CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER SCIENCE, 1971.

[11] Bell C G, Newell A, Grason J. Designing computers and digital systems: using pdp16 register transfer modules. 1972.

[12] Fundamentals and Standards in Hardware Description Languages. Springer Science & Business Media, 2012.

[13] Van der Spiegel J. ABEL–HDL Primer. http://www.ee.usm.maine.edu/courses/ele172/abel/HDL-ABEL%20Primer.pdf. 1999.

[14] Thomas D, Moorby P. The Verilog® Hardware Description Language. Springer Science & Business Media, 2008.

[15] Shahdad M. An overview of VHDL language and technology. 23rd ACM/IEEE Design Automation Conference. IEEE, 1986: 320-326.

[16] IEEE System Verilog Working Group. IEEE Standard for SystemVerilog C Unified Hardware Design, Specification, and Verification (IEEE Std 1800–2017). 2017.

[17] Nane R, Sima V M, Pilato C, et al. A survey and evaluation of FPGA high-level synthesis tools. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2015, 35(10): 1591-1604.

[18] Martin G, Smith G. High-level synthesis: Past, present, and future. IEEE Design & Test of Computers, 2009, 26(4): 18-25.

[19] Gajski D D, Ramachandran L. Introduction to high-level synthesis. IEEE Design & Test of Computers, 1994, 11(4): 44-54.

[20] Kapre N, Bayliss S. Survey of domain-specific languages for FPGA computing. 2016 26th International Conference on Field Programmable Logic and Applications (FPL). IEEE, 2016: 1-12.

[21] Paulin P G, Knight J P. Force-directed scheduling for the behavioral synthesis of ASICs. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 1989, 8(6): 661-679.

[22] High-level VLSI synthesis. Springer Science & Business Media, 2012.

[23] Gajski D D, Dutt N D, Wu A C H, et al. High-Level Synthesis: Introduction to Chip and System Design. Springer Science & Business Media, 2012.

[24] Micheli G D. Synthesis and optimization of digital circuits. McGraw-Hill Higher Education, 1994.

[25] Knapp D W. Digital System Design Using the Synopsys Behavioral Compiler. 1996.

[26] Elliott J P. Understanding behavioral synthesis: a practical guide to high-level design. Springer Science & Business Media, 1999.

[27] Nikhil R. Bluespec System Verilog: efficient, correct RTL from high level specifications. Proceedings. Second ACM and IEEE International Conference on Formal Methods and Models for Co-Design, 2004. MEMOCODE'04. IEEE, 2004: 69-70.

[28] Canis A, Choi J, Aldham M, et al. LegUp: high-level synthesis for FPGA-based processor/accelerator systems. Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays. 2011: 33-36.

[29] Canis A, Choi J, Aldham M, et al. LegUp: An open-source high-level synthesis tool for FPGA-based processor/accelerator systems. ACM Transactions on Embedded Computing Systems (TECS), 2013, 13(2): 1-27.

[30] Feist T. Vivado design suite. White Paper, 2012.

[31] Intel. Intel High Level Synthesis Compiler 17.1. User Guide, 2017.

[32] Wirbel L. Xilinx SDAccel: a unified development environment for tomorrow's data center. The Linley Group Inc, 2014.

[33] Intel. Intel FPGA SDK for OpenCL Pro Edition. Programming Guide, 2019.

[34] Sharma H, Park J, Amaro E, et al. Dnnweaver: From high-level deep network models to fpga acceleration. the Workshop on Cognitive Architectures. 2016.

[35] Intel I. Distribution of OpenVINO™ toolkit for Linux with FPGA support–OpenVINO Toolkit. 2019.

[36] Wang H, Soulé R, Dang H T, et al. P4fpga: A rapid prototyping framework for p4. Proceedings of the Symposium on SDN Research. 2017: 122-135.

[37] Li Y, Leeser M. HML, a novel hardware description language and its translation to VHDL. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2000, 8(1): 1-8.

[38] Bjesse P, Claessen K, Sheeran M, et al. Lava: hardware design in Haskell. ACM SIGPLAN Notices, 1998, 34(1): 174-184.

[39] Bellows P, Hutchings B. JHDL-an HDL for reconfigurable systems. Proceedings. IEEE symposium on FPGAs for custom computing machines (Cat. No. 98TB100251). IEEE, 1998: 175-184.

[40] Philtomson. RHDL. https://github.com/philtomson/RHDL

[41] Clash. https://github.com/clash-lang/clash-compiler

[42] Bachrach J, Vo H, Richards B, et al. Chisel: constructing hardware in a scala embedded language. DAC Design Automation Conference 2012. IEEE, 2012: 1212-1221.

[43] Decaluwe J. MyHDL: a Python-Based Hardware Description Language. Linux journal, 2004 (127): 84-87.

[44] Lockhart D, Zibrat G, Batten C. PyMTL: A unified framework for vertically integrated computer architecture research. 2014 47th Annual IEEE/ACM International Symposium on Microarchitecture. IEEE, 2014: 280-292.

[45] Takamaeda-Yamazaki S. Pyverilog: A python-based hardware design processing toolkit for verilog hdl. International Symposium on Applied Reconfigurable Computing. Springer, Cham, 2015: 451-460.

[46] Charles Papon. SpinalHDL. https://github. com/SpinalHDL

[47] Liu Y, Li Y, Qi Z, et al. A Scala Based Framework for Developing Acceleration Systems with FPGAs. Journal of Systems Architecture, 2019.

[48] Rompf T, Odersky M. Lightweight modular staging: a pragmatic approach to runtime code generation and compiled DSLs. Proceedings of the ninth international conference on Generative programming and component engineering. 2010: 127-136.

附中文参考文献:

[4] 杨海钢, 孙嘉斌, 王慰. FPGA 器件设计技术发展综述. 电子与信息学报, 2010, 32(3):714-727.

[8] Jeff Dorsch. 现场可编程门阵列 FPGA 芯片及其应用. 集成电路应用, 2018(1):77-79.

[9] 吴艳霞, 梁楷, 刘颖, 崔慧敏. 深度学习 FPGA 加速器的进展与趋势. 计算机学报, 2019(11): 2461-2480.