# 北京理工大学本科生毕业设计（论文）外文翻译

|  |  |
| --- | --- |
| **外文原文题目：** | Reusability is FIRRTL Ground: |
|  | Hardware Construction Languages, |
|  | Compiler Frameworks, and Transformations |
| **中文翻译题目：** | 复用性是FIRRTL的基础： |
|  | 硬件构造语言、编译器框架及其转换 |

**基于Chisel语言实现的RISC-V处理器**

**Chisel Implementation of RISC-V Processor**

|  |  |
| --- | --- |
| 学 院： | 计算机学院 |
| 专 业： | 物联网工程 |
| 学生姓名： | 贺清 |
| 学 号： | 1120161774 |
| 指导教师： | 陆慧梅 |

**原创性声明**

本人郑重声明：所呈交的毕业设计（论文），是本人在指导老师的指导下独立进行研究所取得的成果。除文中已经注明引用的内容外，本文不包含任何其他个人或集体已经发表或撰写过的研究成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。

特此申明。

本人签名： 日 期： 年 月 日

**关于使用授权的声明**

本人完全了解北京理工大学有关保管、使用毕业设计（论文）的规定，其中包括：①学校有权保管、并向有关部门送交本毕业设计（论文）的原件与复印件；②学校可以采用影印、缩印或其它复制手段复制并保存本毕业设计（论文）；③学校可允许本毕业设计（论文）被查阅或借阅；④学校可以学术交流为目的,复制赠送和交换本毕业设计（论文）；⑤学校可以公布本毕业设计（论文）的全部或部分内容。

本人签名： 日 期： 年 月 日

指导老师签名： 日 期： 年 月

**复用性是FIRRTL的基础：**

**硬件构造语言、编译器框架及其转换**

摘　要

在现代语言和可重定向编译器的支持下，软件开发处于一个由大量强大的参数化库驱动的虚拟的“寒武纪爆炸”时代；但硬件开发却远远落后。我们假设现有的硬件构造语言（HCL）和新的硬件编译器框架（HCF）可以通过使新硬件库独立于包括FPGA映射的底层处理技术来将硬件开发放在类似的进化路径上。我们支持这一要求发的原因是认为Chisel，一种现有HCL，可以支持强大的参数化库的程度，并且了解了一种使用开源硬件中间表示FRRTL（RTL的灵活中间席表示）的HCF的概念和实现，将目标独立RTL转换成技术规范RTL。最后，我们评估了许多硬件编译器转换，包括简化转换、分析、优化、检测和专门化，这些都展示了HCL和HCF组合的强大功能。

**关键词：RTL; 设计; FPGA; ASIC; 硬件; 建模; 复用性; 硬件设计语言HDL; 硬件结构语言HCL; 中间表示法; 编译; 转换; Chisel; FIRRTL;**

**ReusabilityisFIRRTLGround:**

**Hardware Construction Languages,Compiler Frameworks, and Transformations**

Abstract

Enabled by modern languages and retargetable compilers, software development is in a virtual “Cambrian explosion” driven by a critical mass of powerfully parameterized libraries; but hardware development practices lag far behind. We hypothesize that existing hardware construction languages(HCLs) and novel hardware compiler frameworks (HCFs) can put hardware development on a similar evolutionary path by enabling new hardware libraries to be independent of underlying process technologies including FPGA mappings. We support this claim by (1) evaluating the degree with which Chisel, an existing HCL, can support powerfully parameterized libraries, and (2) introducing the concept and implementation of an HCF that uses an open-source hardware intermediate representation, FIRRTL (Flexible Intermediate Representation for RTL), to transform target-independent RTL into technology-speciﬁc RTL. Finally, we evaluate many hardware compiler transformations, including simplifying transformations, analyses, optimizations, instrumentations, and specializations, which demonstrate the power of a combined HCL and HCF approach.

**Index Terms：RTL; Design; FPGA; ASIC; Hardware; Modeling; Reusability; Hardware Design Language; Hardware Construction Language; Intermediate Representation; Compiler; Transformations; Chisel; FIRRTL;**

**目　录**

[摘　要 I](#_Toc42870052)

[Abstract II](#_Toc42870053)

[第1章 导言 1](#_Toc42870054)

[第2章 两种假设 3](#_Toc42870055)

[2.1 错误的假设 3](#_Toc42870056)

[2.1.1 假设1-现有HDL缺乏表现力 3](#_Toc42870057)

[2.1.2 假设2-底层复杂性要求RTL定制 4](#_Toc42870058)

[第3章 硬件构造语言、硬件编译器框架和转换 6](#_Toc42870059)

[3.1 硬件库的硬件构造语言 6](#_Toc42870060)

[3.2 硬件编译框架 7](#_Toc42870061)

[3.3 FIRRTL变换 10](#_Toc42870062)

[第4章 评价 13](#_Toc42870063)

[4.1 Chisel支持的硬件库 13](#_Toc42870064)

[4.2 HCF支持从后端隔离源 15](#_Toc42870065)

[4.3 案例研究：新ASIC工艺的定制设计 19](#_Toc42870066)

[结 论 21](#_Toc42870067)

[参考文献 22](#_Toc42870068)

[致 谢 27](#_Toc42870069)

# 第1章 导言

Dennard规模的终结和技术进步的放缓已经消除了数字电路相关的“免费的”电源、性能和面积改进。由于专用硬件实现比通用处理器上的软件具有更大的能量和性能改进需求，因此专用化很可能是硬件设计的未来[1][2][3]。这一趋势将体现在对含有不同专业RTL的各种产品的需求增加。用现有的方法来满足这种需求已经被证明是不可靠的[4]。

相比之下，软件行业的设计周期要比硬件行业快得多；一个小团队可以在不到两周的时间内从构思想法到得到适用软件。硬件行业能从软件界学到什么？

软件行业生产力的一个关键贡献因素是复用库，它分摊了新应用程序的开发和验证成本。这些库建立在具有可重定目标编译器的表达语言上，这些编译器对通用代码执行特定于平台的优化。

相比之下，硬件复用相对较少；没有广泛的可复用硬件库存在。然而，如果硬件项目复用更多的代码，工程师可能会花更少的时间来设计硬件，更重要的是，用更少的时间验证新的设计。既然复用代码的好处显而易见，为什么硬件工程师不编写可复用的库呢？本文从以下方面分析原因：

导致硬件库开发停滞的两个假设：

(1)现有的硬件描述语言缺乏支持硬件库的表达能力

(2)不同的底层实现需要RTL定制，限制了复用性。

重新强调和分析硬件构造语言（HCL）作为硬件库的主要工具：以前，许多影响广泛的工作都是在硬件构造语言的概念上引入和扩展的。本文仅在提供一个开发硬件库的平台的背景下对它们进行了重新评估。

硬件编译器框架（HCF）的开源实现，用于将RTL与实现约束隔离：当软件可重定目标编译器将通用代码转换为平台专用程序集时，HCFs将通用RTL转换为目标专用RTL。通过将这些转换形式化为编译器框架，我们可以实现健壮且可复用的RTL转换。

对许多转换的评估，证明了我们框架的广泛适用性：我们的HCF实现采用硬件中间表示，FRTRTL（RTL的灵活中间表示），作为许多不同变换的基础，包括简化变换、分析、优化、仪器和专门化。

# 第2章 两种假设

软件库在软件开发中无处不在，因为通过代码复用，它们降低了新应用程序的开发和验证成本。现代软件依赖于数以千计的图书馆——例如Ubuntu 14.04在本地安装了大约35000个包。

直接比较而言，硬件设计师通常不会在项目之间复用模块，更不用说开发广泛且可复用的库了。

其他复用硬件的尝试也取得了喜忧参半的成果。在SoC级别增加对大型复杂定制IP块的复用有许多好处，包括更快的上市时间和减少验证工作。但是，自定义IP块通常非常专用化，而不是像队列、算术单元、乘法器、缓存等硬件的基本构建块，并且比典型的可复用库带来更多的集成挑战。重申一下：为什么硬件工程师不编写可复用的库呢？

## 2.1 错误的假设

有人说硬件库的缺乏源于缺乏努力；然而根据作者的经验，许多公司试图建立硬件组件的内部可复用库，但都失败了。

也有人说硬件库的缺乏源于开源社区的缺乏；然而，流行的开源软件通常是由一两个贡献者编写的。流行的JavaScript可视化库D3[5]主要是由一个工程师编写的，但是仍然得到了广泛的应用。

2.1.1 假设1-现有HDL缺乏表现力

自从20世纪80年代设计了大多数流行的硬件描述语言（HDL）以来，编程语言已经有了显著的改进。主流编程语言的现代发展使得java、C++、Python、Perl和Ruby等语言变得非常强大。面向对象、多态性和高阶函数能够使用良好的软件工程原则，如抽象、关注点分离和模块化；这些最终鼓励并支持代码复用。HDL采用这些模式的速度非常慢。

加法器归约树说明了这个问题：Verilog和VHDL不能表示递归生成语句，因此设计师必须手动展开循环并计算每个实例的索引。由于缺少参数化，因此在需要不同宽度的树时无法复用。

另一个例子是过滤数据包的模块。过滤器模块或外部模块必须对过滤器条件进行编码。第一种方法违反了关注点分离的原则，而第二种方法违反了封装。然而，高阶函数为这个问题提供了一个优雅的软件工程解决方案。

System Verilog，创建于2002，试图通过与面向对象编程和经典Verilog元素混合的现代思想来改进现有HDL。其结果是一种极其复杂的语言，难以支持和理解，但仍然缺少其他现代功能，如高阶函数。据作者所知，没有任何商业System Verilog编译器实现了整个规范[6]。

高级综合（HLS）采用了一种不同的方法，使用更高级的语言进行用户设计，编译器将其转换为RTL。输入语言可以是类C语言[7][8][9][10][11]、并行类C语言[12][13][14]、通用语言[15]或领域专用语言[16][17][18][19][20][21]。许多HLS工具的评估标准包括使用的简单性、相对于手工编码实现的性能、简洁性和资源占用率；他们设计可复用硬件库的能力通常不被考虑。

不幸的是，HLS方法面临两个相互矛盾的问题：

1)表达性源语言能够实现更好的软件工程（和更多的复用）；

2)表达性源语言更难翻译成硬件，也更难创建更多的编译/抽象层，这些层会阻碍用户对设计进行优化。

2.1.2 假设2-底层复杂性要求RTL定制

尽管逻辑综合取得了成功，但许多潜在的制约因素仍然影响着RTL的设计。

ASIC实现通常需要RTL定制。例如，Verilog缺乏显式的内存结构；用户必须使用寄存器数组。在现代技术中，sram是由制造公司提供的，因为大的存储器通常有助于设计关键路径。RTL设计者必须重写他们的设计，用黑盒sram替换这些寄存器阵列；这就消除了将来不使用该ASIC技术或性能包线的任何复用。

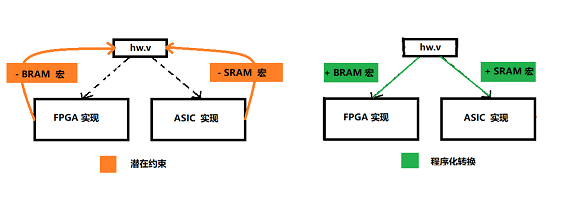


图2-1：ASIC与FPGA实现的底层约束意味着同一RTL不能在两个平台上都获得好的结果。这限制了任何RTL设计的可复用性。为了解决这个问题，程序化的RTL转换必须采用通用RTL并针对给定的平台对其进行专门化。

现场可编程门阵列的实现并没有什么不同；为了提高设计质量，许多FPGA都对逻辑块进行了硬化处理。设计师可以通过修改其RTL使其对特定的FPGA综合工具更友好，从而获得显著的性能、功耗或利用率改进。然而，这些改变可能对ASIC实现或另一个FPGA实现有害。

为了解决这个问题，一些设计师编写了一组自定义脚本来进行即席编程的RTL修改；这些脚本既不可复用、稳定，也不可组合。

商业CAD工具也不能完全解决这个问题。虽然有些工具包含RTL到RTL的转换，但CAD工具主要关注综合、布局与布线。它们也没有在开源编译器框架中进行组织，并且对于可能有不受支持的用例的自定义窗口不够完善。

一个例外是Yosys[22]，一个用于Verilog RTL合成的开源框架，它将Verilog映射到ASIC标准单元库或Xilinx FPGAs。Yosys的主要关注点是逻辑合成，而不是RTL到RTL的转换。因此，它的内部设计表示是非常低级的，不能表示更高级的构造，如聚合类型、宽度推断和条件赋值。

独立于CAD工具，存在一种独立的RTL修改器，但许多是不开源的[23]，并且与商业CAD工具一样，不容易支持自定义。PyVerilog[24]是一个例外，它是专门为Verilog定制的RTL到RTL修改器。因此，它使得它很难按照设计师的意图行事，而不是在Verilog构造中直接表示。PyVerilog不支持SRAM推断或聚合类型，考虑到其内部电路表示，这些特性很难支持。另一个例外是Verific[25]，这是一个解析Verilog/VHDL设计并允许用户基于源AST或独立于语言的netlist格式编写转换的商业工具。因为Verific支持这些语言的每个细节，所以自定义转换必须支持所有语言细节（这增加转换复杂度），或者可以在网表格式上操作（缺少设计者意图）。

# 第3章 硬件构造语言、硬件编译器框架和转换

我们断言，表达性语言和编程定制支持可复用库。本节着重介绍硬件构造语言（HCL）如何实现具有表现力的硬件设计，以及硬件编译器框架（HCF）如何实现编程自定义。然后，我们介绍我们的开源HCF实现及其中间表示（IR），并展示了示例转换以证明其广泛的适用性。

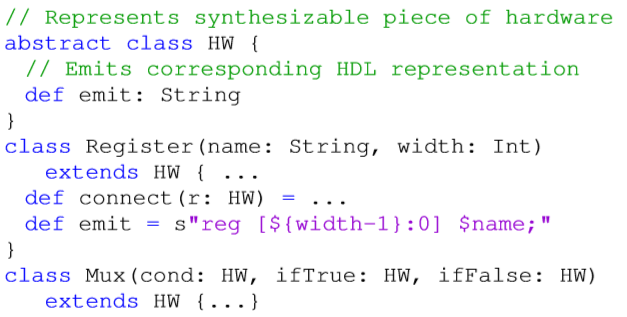
## 3.1 硬件库的硬件构造语言

硬件构造语言（HCL）在现有的编程语言中嵌入类似HDL的硬件原语。因为HCL直接使用RTL提取，所以在任何HDL上使用HCL都没有性能/区域开销。HCL设计人员使用通用语言丰富的控制结构和抽象来创建模块化的、可参数化的、可复用的和性能良好的设计，与同等的HDL设计相比[26][27][28][29][30][31][32]。

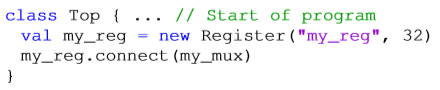
Chisel[33]是一个开源的HCL，它依附在Scala[34]中，Scala是一种现代的面向对象的函数式语言。

(1)HCL结构：

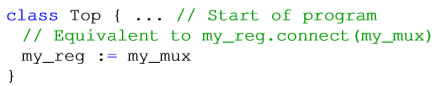
所有HCL都是软件库，具有构建可合成RTL的接口。为了说明，下面的示例HCL具有表示寄存器或多路选择的类：



然后，设计器可以创建一个寄存器，并通过实例化register对象并调用其connect方法将其连接起来：



诸如运算符重载之类的语言功能还可以减少复杂程度：



通过执行该HCL代码，可以生成完整的设计；这个过程叫做精化。每个HCL方法调用都构建一个表示硬件设计实例的底层数据结构。然后可以将该设计发展到现有HDL。在设计良好的HCL中进行开发，可以很好地模拟在HDL中编写的体验。

(2)启用硬件库：

HCL本身不提供任何新的硬件抽象。然而，宿主语言特性允许设计更具参数化和模块化。

例如，Chisel用户可以编写递归Scala函数来构造加法器归约树，该树根据位宽度参数化。与Verilog中所需的显式展开版本不同，同一个生成器可以在任何需要加法器树的地方复用。

类似地，一个Chisel设计者可以编写一个过滤器模块，它以一个更高阶的函数作为参数来创建条件检查硬件。这个模块的用户只需要编写过滤条件，重新使用基本过滤结构。

最终，HCL的宿主语言表达能力是实现可复用硬件库开发的关键。

## 3.2 硬件编译框架

当软件编译器将通用代码转换为专用程序集时，硬件编译器将通用RTL转换为专用RTL。通过将这些转换收集到编译器框架中，我们可以实现健壮且可复用的RTL转换。

任何编译器的中心部分都是它的中间表示（IR），所有的转换都是在它上面操作的。本节概述了我们的硬件编译器框架及其开源IR、FIRRTL的设计。

(1)HCF结构：

像LLVM[35]这样的现代软件编译器框架由前端，转换和后端组成。前端分析程序用专门的C语言编写（如C++或RIST）到编译器特定 IR中。从IR到IR的转换，例如优化过程，可以操作和修改程序的结构。最后，后端将IR转换为目标ISA中的程序，例如ARM或x86。这种将输入语言转换为IR的结构可以复用多个设计和语言之间的转换。

我们的HCF结构类似：Chisel和Verilog前端将设计转换为FIRRTL，转换通道提供简化、优化，生成的FIRRTL可以直接模拟，也可以传递给模拟器、FPGA或ASIC技术过程量身定制的许多Verilog后端之一。

(2)FIRRTL设计理由：

设计一个IR是任何编译器的一个重要组成部分，我们评估IR的这三个理想但有时相互竞争的特性：

清晰性：语义简单

简单性：一组小的IR节点

丰富性：捕捉用户意图

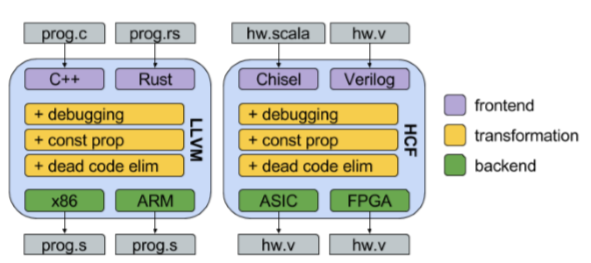


图3-1：LLVM可以创建一个C++到X86的编译器或Rust到ARM的编译器，并在LLVM IR上共享内部转换。同样，我们的HCF可以创建一个Chisel到ASIC的Verilog编译器或Verilog到FPGA的Verilog编译器，并共享内部转换。

所有操作RTL级或门级设计的工具都有一个他们操作的IR，无论是否严格定义。每个工具的IR根据其使用情况做出不同的权衡：一个只在行为Verilog-2005上运行的IR应该比一个只在netlists上运行的IR更丰富，但不那么清晰和简单。

我们的硬件IR，FIRRTL（灵活的中间表示RTL），代表RTL数字电路，并被设计为专门来源于源代码RTL代码从底层实现[ 36 ]。在搜索中，FIRRTL首先优先考虑丰富性，以尽可能多地捕获RTL用户的源意图。例如，FIRRTL包含显式内存节点、聚合类型、时钟类型和类型安全连接，以使其他语言（如Chisel）能够映射到这些结构并捕获用户的意图。

由于我们的HCF最终必须为下游模拟器和工具发出不太丰富的表示，所以FIRRTL也很简单。最后，FIRRTL是明确的，因为它是严格定义的，具有直接的宽度推断和类型推断规则。

(3)FIRRTL概述：

FIRRTL定义了用于封装的硬件模块、用于状态元素的寄存器和存储器、用于组合逻辑的基本操作和复用。

为了弥补捕获用户意图和下游格式之间的差距，FIRRTL由三个定义良好的表单（高表单、中表单、低表单）组成，每个表单都比前一个表单具有更小、更严格和更简单的FIRRTL特征子集。FIRRTL的低表单包含一组低层特性，这些特性直接映射到Verilog结构，在各种目标上具有直接的语义。

任何转换都可以指定它使用哪个FIRRTL表单，但总是可以发出一个随后降低的表单。不太丰富的输入具有较少的特定情况，修改或生成FIRRTL的转换通过访问丰富的IR功能简化。

(4)内存中的FIRRTL表示：

FIRRTL设计的内存结构显著地影响了编写转换的容易程度。正如在软件编译器中通常所做的那样，FIRRTL设计在内部用抽象语法树（AST）结构表示，其中转换递归地遍历嵌套元素来操作AST。如果需要非本地信息，转换首先遍历树以构建自定义数据结构，然后再次遍历树以操作AST。

某些转换可能需要设计的其他表示。例如，组合循环检测在类似网格表的有向图上操作。我们的编译器框架有一个附带的库，转换可以使用它来构建这些数据结构。

FIRRTL 的AST由非内存对象表示的IR节点组成，每个节点都是以下IR抽象类之一的子类：电路，模块，端口，语句，表达式，类型。每个IR节点可以有其他IR节点类的子对象，其关系如图3-2所示。图3-3演示了如何在内存中将FIRRTL电路表示为IR节点的AST。

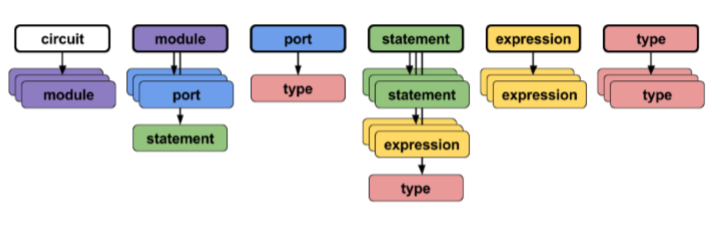


图3-2：使用这些AST节点表示FIRRTL电路。每个节点可以有一个或多个不同类型的子节点。例如，FIRRTL语句可以有子语句、表达式和/或类型。

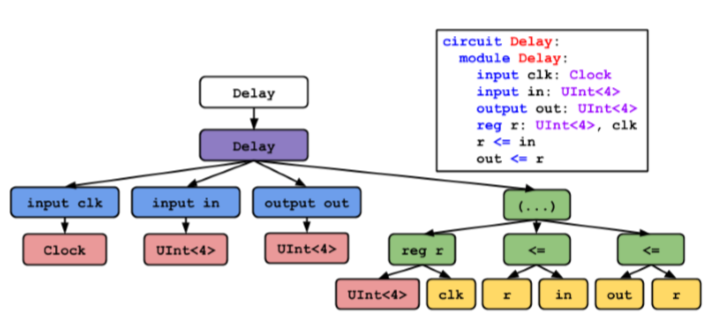


图3-3：一个示例FIRRTL电路在其AST与文本表示中的对比。该电路包含一个输出延迟一周的输入信号的模块。（…）语句是一个只包含多个子语句的块语句-此节点使在AST的一次遍历中用多个语句替换单个语句变得容易。

以下递归算法访问电路中的所有表达式节点：首先，访问每个模块的语句节点。对于每个访问的语句，访问其每个子语句和表达式节点。对于每个访问的表达式，访问其每个子表达式节点。

所有转换都使用FIRRTL的 AST的递归遍历来修改电路。

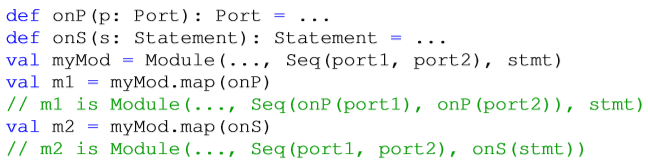
## 3.3 FIRRTL变换

变换总是消耗和产生一个良好的AST，并且很容易连接一个接一个。每次转换后都可以检查设计上的约束。这种结构使得插入新的转换变得简单和安全，不像使用脆弱的即席脚本。

(1)转换示例：

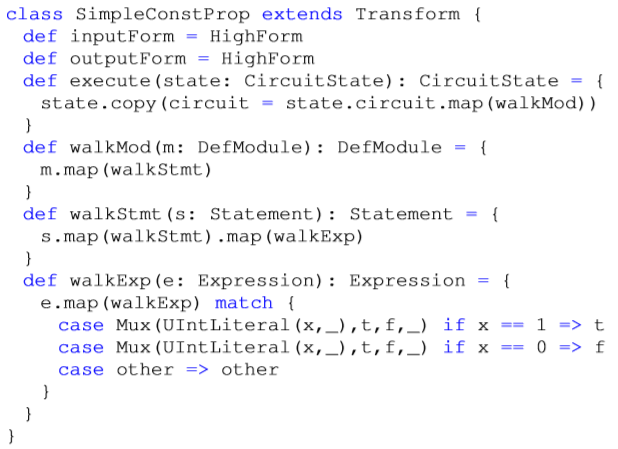
为了表示递归遍历，每个IR节点都实现了一个自定义映射函数；节点的映射将用户指定的函数应用于其节点类型与函数的输入和返回节点类型匹配的子节点的子集。

下面的示例演示如何使用接受并返回端口的函数和返回语句的函数调用模块的映射。



虽然很简单，但是使用图递归地遍历FIRRTL AST是非常强大的。

下面的示例是一个优化转换，它使用常量谓词在mux上进行常量传播。我们通过调用模块、语句和表达式上的映射递归地遍历所有FIRRTL模块、语句和表达式。对于我们看到的任何mux，我们检查我们的恒定传播条件，如果为真，则执行优化。请注意，此代码以postorder遍历方式访问表达式，只需要一次通过AST。



1)简化转换：

简化转换采用一个FIRRTL电路，并将其简化为较低的形式。有两种简化转换：

从高到中，它以高形态出现，发出中形态；

从中到低，呈中间形态，发出低形态。

例如，从高到中转换的一个任务是删除FIRRTL的大容量连接操作符。此运算符允许具有聚合类型的组件以类型安全的方式与单个语句连接，从而捕获用户意图。但是，较低的表单只支持基元类型之间的连接，因此高到中转换将大容量连接重写为一系列单独的连接。

(2)分析转换：

设计人员通常希望深入了解编译器以了解优化的程度。节点计数、早期区域估计和模块层次结构描述是设计周期早期的三种有用的分析转换。

(3)优化转换：

实现的三个主要优化转换的方法是常数传播、公共子表达式消除和死代码消除。因为下游工具进行积极的逻辑分析和其他优化，这些转换对门级设计几乎没有影响，但对代码可读性至关重要。

(4)检测转换：

我们的HCF的模块化结构使得添加简单的指令传递变得非常简单。这些可以包括插入硬件计数器、硬件断言，甚至改进模拟线路覆盖率检测。

(5)专业化转换：

不同的后端目标，特别是FPGA和ASIC工艺节点，需要RTL修改以获得良好的效果。

为了解决第2章中描述的Verilog的内存问题，Chisel有一个直接发出FIRRTL内存的高级内存结构。我们的内存转换要么发出寄存器数组，要么发出技术专用的SRAM宏。对于FPGA，我们可以发出样式化的Verilog，以确保正确推断BRAM，并通过直接实例化FPGA模板来实现对硬宏的定位。其他ASIC后端专业化包括与衬垫框架的集成，以及用于平面规划的细粒度、重复数据消除和模块移动。

# 第4章 评价

本文声称，HCL和HCF通过促进代码复用和将源代码与后端特定优化/约束隔离，实现了灵活的硬件库。我们证明了以下几点：

Chisel提供了支持高度参数化和可复用性所必需的表示性；

我们的HCF实现具有广泛的适用性，可以为许多不同的用例定制RTL。

4.1 Chisel支持的硬件库

表达性语言需要更少的代码行来更充分地参数化设计。这种参数化允许在不同的上下文中用不同的参数复用相同的代码，从而可能生成完全不同的硬件。

下面将从表达性、参数化以及最终的可复用性方面对Chisel进行评估。

(1)表达性:

使用由现代编程语言支持的软件工程方法，我们应该期望更少的代码行来表示类似的项目。

RocketChip[37]是一个开源硬件库，用Chisel编写，可以生成对称多处理器系统（SMP）的许多不同实例。OpenPITON[38]是另一种开放源码的MyCype处理器和用于研究的框架，主要是在Verilog编写的，并使用一些Python Verilog生成脚本进行增强，这些脚本使用Sun UltraSPARC T1（OpenSPARC）内核，具有定制的互连和一致性框架。

OpenPiton和RocketChip在许多地方有很多相似之处——它们都是SOC生成器，包含核心、缓存、网络协议、一致性域、测试等等。两者都用于计算机体系结构的研究，已经在silicon中实现，并引导Linux。OpenPiton的内核是一个简单的有序设计，具有多线程功能，而RocketChip内核要么是有序（Rocket）要么是无序（BOOM），但都没有多线程功能。如表I所示，RocketChip核心具有2.42至4.49的核心分数，而OpenSPARC具有1.32[39]的核心分数。相比之下，同样大小的工业无序核心Cortex-A9的核心得分为3.71[39]。

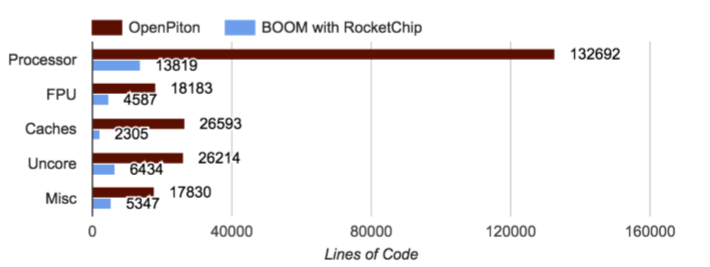
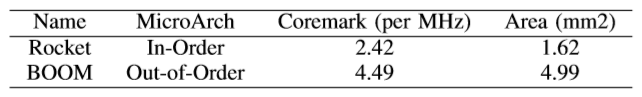


图4-1：类似的硬件结构在代码大小上显示出巨大的差异，从3倍到10倍不等。由于它们的特征集不同，这种评价不应作为严格的比较；相反，我们认为这是一个普遍的趋势，即使用Chisel可以实现更具表现力的编码风格。

虽然很明显是苹果对橘子的比较，但图4-1描述了代码基之间的比较。OpenPiton需要3到10倍的代码来表示相似的硬件结构；OpenPiton和RocketChip之间代码大小的巨大差异不能仅仅用它们不同的特性集来解释。此外，据作者所知，RocketChip的无序核心BOOM[40]用最少的代码行来实现任何开源的无序核心实现。

表1-1：Rocket和BOOM的参数比较。



虽然我们期望OpenSPARC内核中的大部分不是完全手写的（像编辑器扩展这样的工具可以被使用），但是我们觉得语言表达能力的比较仍然有效：Chisel显然比Verilog更具表现力，这一点可以从RocketChip代码大小的显著减少中看出。

1)参数化：

参数化先于有效的可复用性-灵活的参数化模块更有用，因此更可重用。

虽然很难定量评估通用编程语言提供HCL的参数化的灵活性、规模和程度，但我们定性地描述了RocketChip的参数化类型和程度：

无序参数：获取宽度（1、2、4）、发布宽度（1、2、3、4）、分支预测值（BTB、GShare、TAGE）

数据并行：并行数据操作数（4到32），精度（半、字、双）

多核：核数（1、2、4、8、16）

缓存：大小（64KB到2MB）、关联性（直接映射、双向）、类型（暂存、阻塞、非阻塞）、一致性策略（MSI、MESI）

请注意，这些参数的叉积都是有效的，并且这些设计点中的许多（但不是全部）已经进行了试验甚至实现。

此外，这些参数不仅仅是位宽度，而是影响控制逻辑、接口定义和通信协议。在表一中，我们核心的两个不同参数化导致了具有非常不同的微体系结构、性能结果和面积数的巨大不同的设计。

2)可复用性：

我们分析了三个用Chisel写的处理器去了解参数化设计是否促进了可复用性：

BOOM[40]：RocketChip的无序核心

Rocket：RocketChip单一有序核心

DecVec：一种解耦矢量协处理器

如图4-2所示，大约5000行代码与所有三个设计共享，甚至更多的代码在设计对之间共享。总之，这三种设计彼此共享一半或更多的代码基。



图4-2：Rocket、BOOM和DecVec三个处理器复用彼此的代码量。这三种设计所使用的模块包括ALU、MulDiv单元、ICache、TLB、解码器和FPU。Rocket和BOOM使用的模块包括非阻塞数据缓存、PTW、CSR和BTB。

4.2 HCF支持从后端隔离源

当源代码更改反映后端特定定制时，会限制代码的可复用性。我们的HCF旨在实现许多不同类别的定制，以转换设计。

首先，我们评估了我们的IR，FIRRTL的丰富性，然后证明它可以支持类似程度的优化，CAD工具可以使用。然后，我们演示并评估了一些工具和专门化转换，以说明我们的框架的广泛适用性。

(1)FIRRTL评估：

在Yosys和PyVerilog中，FIRRTL与IR的区别在于它捕获用户意图的能力。递归聚合类型（向量和束）支持对相关信号进行分组。条件连接语句支持直接的浮点信号检查。批量连接启用类型安全分配。最后，它的显式内存节点允许直接和强大的内存优化。所有这些FIRRTL特性都应该支持简洁的表达式，这些表达式需要不太丰富的IR来使用更多的代码行来表示。

为了演示丰富IR的实用性，我们分析了以下三种设计：重新排序缓冲区，分支重新排序缓冲区，重命名空闲列表的寄存器。由于设计的丰富功能被简化为FIRRTL的中、低格式，我们记录了表示设计所需的代码行。为了确保我们的HCF不影响FIRRTL的代码大小，我们在编译的最后阶段运行优化过程。

如图4-3所示，尽管进行了FIRRTL优化，但一些设计在简化过程中显示出代码大小的巨大增长；这说明了如果设计师或前端选择使用丰富的功能，富IR如何简洁地表达设计。

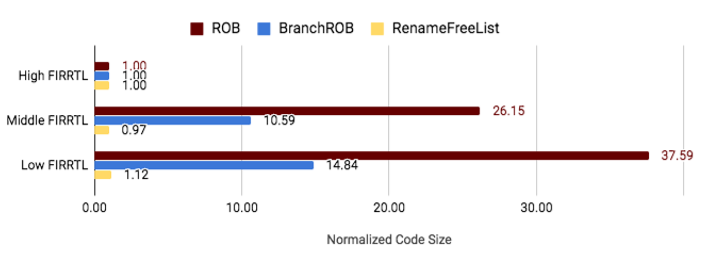


图4-3：代码大小规范化为高FIRRTL中表示的大小。ROB和BranchROB都使用聚合类型、大容量连接和内存节点，而RenameFreeList主要由逻辑组成，不使用丰富的FIRRTL功能。

(2)优化评估：

用Yosys这样的综合工具进行位级分析，可以执行比FIRRTL更积极的优化。然而，如图8所示，我们的优化过程将节点数减少了71%，而Yosys的减少了76%。运行FIRRTL和Yosys的优化过程可以进一步减少节点数量。

(3)仪器评估：

我们实现了一个FIRRTL线覆盖变换，它在模拟中执行时检测电路以打印其覆盖信息。这种工具对于Chisel是必要的，因为它的一些构造不能直接映射到Verilog，所以必须首先简化。这会破坏Verilog行覆盖工具所依赖的源级别信息，使它们在很大程度上失效。此转换通过将高级源代码行信息与低级执行语句关联来工作。

图4-5显示了RocketChip SoC实例中模块的百分比，该百分比由测试的模块源代码行的百分比表示。我们展示了三种不同的SoC配置的结果；三种配置中都存在覆盖率较低的模块，而且基于配置的覆盖率没有明显的趋势。一般来说，大多数模块在给定的测试套件上都有很高的覆盖率，只有少数模块测试得很轻。这种转换使设计师能够针对低覆盖率模块进行新的测试并改进验证。

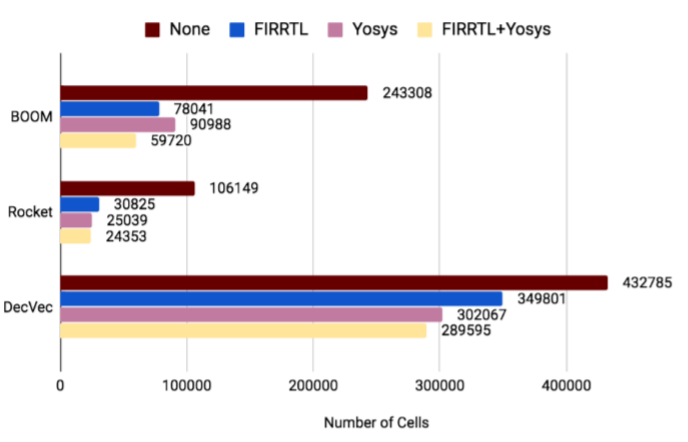


图4-4：FIRRTL优化过程将单元数减少到与Yosys优化过程相似的程度。

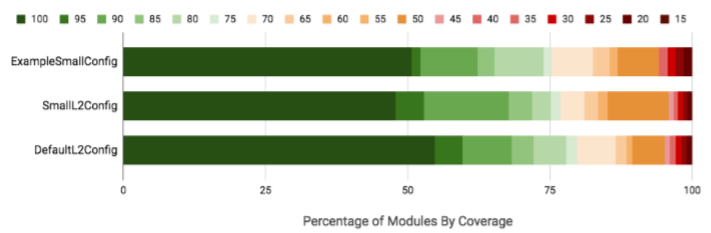


图4-5：我们展示了三种不同的SoC配置的结果，最小规模的配置，ExampleSmallConﬁg，中等规模的配置，带有小型二级缓存，SmallL2Conﬁg，以及默认规模的配置，带有256KB二级缓存，DefaultL2Conﬁg。大多数模块都有很高的覆盖率，但仍有一些模块需要有针对性的测试。

(4)FPGA模拟转换：

在FPGA上进行模拟时，设计几乎没有默认可见性。像Chipscope[42]这样的商业工具能够进行实时分析，但是需要很长的迭代周期来选择特定的信号。此外，它不提供对FPGA上BRAM存储器的可见性，因此无法在给定周期提供设计的完整“快照”。

我们实现了一个仪器转换，它允许暂停在FPGA上的设计（分离主机和目标时间），以及一个转换，它允许读取FGPA上目标设计的状态快照。这些转换包括将使能信号线程化到所有寄存器，插入缓冲区以记录输入和输出跟踪，插入地址生成硬件以读取内存状态，以及附加自定义菊花链以从FPGA扫描出寄存器和BRAM状态。这些类型的转换是许多用例的关键部分，包括快速而精确的功率模拟[43]和调试电路（未来的工作）。

除了检测之外，其他优化的目的是在将设计映射到FPGA时提供最有效的资源使用。特别是，BRAMs是一种宝贵的资源，通过复制很容易被浪费，以适应高端口数；相反，其中一个FPGA专用通道可以通过提供时钟倍增和粘合逻辑，自动将具有高端口数的存储器转换为具有一半端口的双泵浦存储器。尽管这可能会降低最大时钟速率，但BRAM宏的高速潜力意味着许多微体系结构将比吞吐量减半的情况更糟糕。作为这种折衷的交换，对于消耗三个操作数并每周期产生一个结果的流向量算术块，关口的BRAM利用率降低了3倍，如图4-7所示。

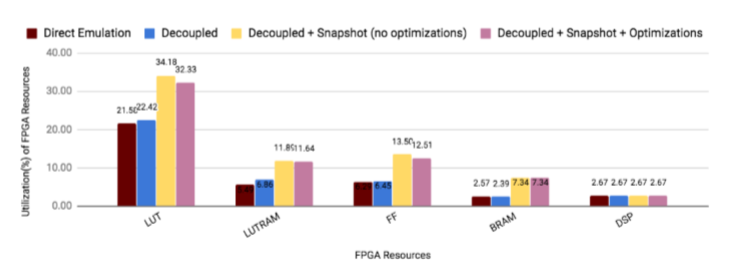


图4-6：去耦和快照转换可以增加对FPGA资源的显著需求，但确实提供了以前无法实现的设计可见性。基线设计和转换设计以40MHz运行。

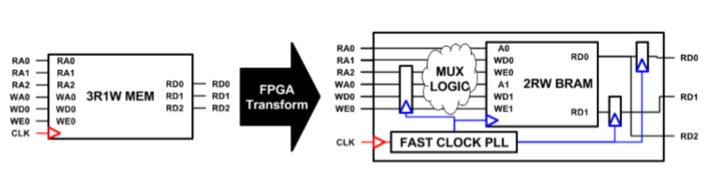


图4-7：自动双泵浦通过模拟昂贵的高端口存储器节省了FPGA资源。这种方法维护有助于复用的抽象。

(5)ASIC制造转换：

ASIC设计得益于针对特定功能的高度优化硬IP宏的使用。例如，基于大内存的设计通常映射到供应商提供的sram而不是寄存器，以提高QoR（在面积、功耗和定时关闭方面）和工具运行时。如图4-8所示，综合后，用sram（4组512深度存储器）实现的基于2048点存储器的FFT（20位实数和虚数）比用寄存器实现的相同FFT小6倍。由于在基于寄存器的设计中产生了过多的路由惩罚，因此在放置和路由之后，节省的空间将增加。此外，与基于寄存器的设计相比，综合基于SRAM的设计所需的时间要少得多，因为这些工具需要处理的硬件实例要少得多。

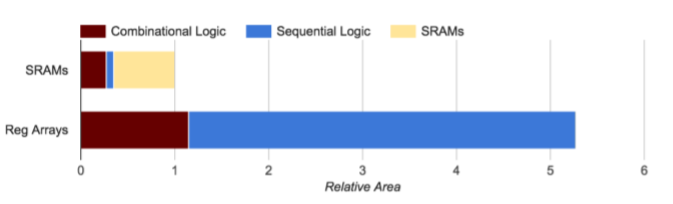


图4-8：在16nm的处理过程中，合成了一种硬件FFT Chisel设计，该设计包括有和没有存储器替换变换。我们的转换将利用面积提高了6倍。设计满足800ps（1.25GHz）的定时。有SRAM的综合时间为6分钟，而没有SRAM的综合时间为1小时44分钟。

然而，专门化RTL来在每种技术的基础上使用这些宏是非常重要的。供应商提供的SRAM通常需要正确连接额外的管脚以进行初始化和验证，但对高级功能没有贡献。为了解决这个问题，我们的内存替换转换用一个与供应商提供的SRAM端口匹配的自定义黑盒替换了一个通用FIRRTL内存。在不运行转换的情况下，FIRRTL将通用FIRRTL内存转换为一个大的寄存器数组。运行转换时，特定于设计的内存信号（数据、地址和启用）将连接到供应商提供的SRAM实例的端口，并且与SRAM之间的任何附加初始化和验证信号将跨模块边界自动连接到顶级端口。

默认情况下，通用FIRRTL存储器映射到双端口SRAM。但是，可以运行额外的优化过程，用单端口sram替换存储器。在进行替换之前，此过程遍历电路以验证读和写启用在逻辑上是互斥的。这大大减少了将通用RTL映射到优化硬件所需的设计工作量。

4.3 案例研究：新ASIC工艺的定制设计

为了演示使用HCL/HCF框架的工作流，我们创建了RocketChip的自定义参数化，合成和放置，并在28nm工艺上路由到DRC/LVS-clean GDS。该设计由两个核和一个大数据并行缓存相干加速器组成。二级缓存存储量很大，这需要多个SRAM。此外，还有多个时钟和电压域，以及多个高速片外IO。

由于RocketChip硬件库采用的参数化和重用，很容易指定所需的设计-只添加了1817行新代码，其中包括专门的配置参数、顶级粘合逻辑和相关的测试工具。许多模块已经在以前的项目中进行了验证和评估，因此需要较少的验证和设计工作。因此，几乎所有的验证工作都花在了新代码上，而这种可重用性是减少设计开销的关键。

以28nm工艺为目标，重复使用第三-C节所述的存储器转换和优化转换。然而，这一过程提出了两个新问题：我们的综合工具需要指定每个模块的时钟和电压域；SRAM有额外的初始化和控制管脚，这一过程是独一无二的。

由于HCF实现的模块性，我们编写了两个自定义过程来解决这些问题，并将它们作为HCF转换库的一部分添加到其他设计、后端和项目中，只需要680行新代码。我们的工具链（和运行时间）包括Chisel（12分钟）、FIRRTL（11分钟）、合成（3.8小时）、布局布线（>40小时）。

总的来说，94%的设计是重复使用的，未来的工作将在其他技术的设计中验证这种方法。

结 论

与软件行业不同，硬件行业由于缺乏通过库重用代码而受到抑制。为了启用硬件库，本文提供了以下内容：

(1)重新强调HCLs如何提供语言表达能力以实现可复用性

(2)我们的硬件编译器框架，基于一个新的硬件IR，FIRRTL，如何支持RTL定制

(3)硬件编译器框架的广泛应用

专业化是硬件设计的未来，提高硬件设计方法的可复用性是满足芯片多样性需求的关键。设计人员应该专注于开发可复用的硬件库，而研究人员和开发人员应该把可复用性作为未来语言和编译器的主要关注点。

参考文献

[1] G. Venkatesh, J. Sampson et al., “Conservation cores: Reducing the energy of mature computations,” in Proceedings of the Fifteenth Edition of ASPLOS on Architectural Support for Programming Languages and Operating Systems, ser. ASPLOS XV. New York, NY, USA: ACM, 2010, pp. 205–218.

[2] R. Hameed, W. Qadeer et al., “Understanding sources of inefﬁciency in general-purpose chips,” in Proceedings of the 37th Annual International Symposium on Computer Architecture, ser. ISCA ’10. New York, NY, USA: ACM, 2010, pp. 37–47.

[3] N. Goulding-Hotta, J. Sampson et al., “The greendroid mobile application processor: An architecture for silicon’s dark future,” IEEE Micro, vol. 31, no. 2, pp. 86–95, Mar. 2011.

[4] O. Shacham, O. Azizi et al., “Rethinking digital design: Why design must change,” IEEE Micro, vol. 30, no. 6, pp. 9–24, Nov. 2010.

[5] M. Bostock, V. Ogievetsky, and J. Heer, “D3 Data-Driven Documents,” IEEE Transactions on Visualization and Computer Graphics, vol. 17, no. 12, pp. 2301–2309, Dec. 2011.

[6] S. Sutherland and D. Mills, “Synthesizing systemverilog: Busting the myth that systemverilog is only for veriﬁcation,” in SNUG Silicon Valley, 2013.

[7] Xilinx, “Vivado High-Level Synthesis.” [Online]. Available: http: //www.xilinx.com/products/silicon-devices.html

[8] Mentor, “Catapult and PowerPro: High-Level Synthesis and RTL Low-Power.” [Online]. Available: <https://www.mentor.com/hls-lp/>

[9] P. Coussy, C. Chavet et al., “GAUT: A High-Level Synthesis Tool for DSPApplications,”inHigh-LevelSynthesis,P.CoussyandA.Morawiec, Eds. Springer Netherlands, 2008, pp. 147–169.

[10] A. Canis, J. Choi et al., “LegUp: High-level Synthesis for FPGA-based Processor/Accelerator Systems,” in Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays, ser. FPGA ’11. New York, NY, USA: ACM, 2011, pp. 33–36.

[11] J. L. Tripp, M. B. Gokhale, and K. D. Peterson, “Trident: From HighLevel Language to Hardware Circuitry,” Computer, vol. 40, no. 3, pp. 28–37, 2007.

[12] T. S. Czajkowski, U. Aydonat et al., “From opencl to high-performance hardware on FPGAS,” in 22nd International Conference on Field Programmable Logic and Applications (FPL), Aug. 2012, pp. 531–534.

[13] M. Owaida, N. Bellas et al., “Synthesis of Platform Architectures from OpenCLPrograms,”in 2011 IEEE 19th Annual International Symposium on Field-Programmable Custom Computing Machines, May 2011, pp. 186–193.

[14] A. Papakonstantinou, K. Gururaj et al., “FCUDA: Enabling efﬁcient compilation of CUDA kernels onto FPGAs,” in 2009 IEEE 7th Symposium on Application Speciﬁc Processors, Jul. 2009, pp. 35–42.

[15] J. Auerbach, D. F. Bacon et al., “Lime: A Java-compatible and Synthesizable Language for Heterogeneous Architectures,” in Proceedings of the ACM International Conference on Object Oriented Programming Systems Languages and Applications, ser. OOPSLA ’10. New York, NY, USA: ACM, 2010, pp. 89–108.

[16] N. George, H. Lee et al., “Hardware system synthesis from DomainSpeciﬁc Languages,” in 2014 24th International Conference on Field Programmable Logic and Applications (FPL), Sep. 2014, pp. 1–8.

[17] F. Hannig, H. Ruckdeschel et al., “PARO: Synthesis of Hardware Accelerators for Multi-dimensional Dataﬂow-Intensive Applications,” in Reconﬁgurable Computing: Architectures, Tools and Applications. Springer, Berlin, Heidelberg, Mar. 2008, pp. 287–293.

[18] P. Milder, F. Franchetti et al., “Computer Generation of Hardware for Linear Digital Signal Processing Transforms,” ACM Trans. Des. Autom. Electron. Syst., vol. 17, no. 2, pp. 15:1–15:33, Apr. 2012.

[19] A. Hormati, M. Kudlur et al., “Optimus: Efﬁcient Realization of Streaming Applications on FPGAs,” in Proceedings of the 2008 International Conference on Compilers, Architectures and Synthesis for Embedded Systems, ser. CASES ’08. New York, NY, USA: ACM, 2008, pp. 41– 50.

[20] R. Nikhil, “Bluespec System Verilog: efﬁcient, correct RTL from high level speciﬁcations,” in Second ACM and IEEE International Conference on Formal Methods and Models for Co-Design, 2004. MEMOCODE ’04. Proceedings, Jun. 2004, pp. 69–70.

[21] “IEEE Standard for Standard SystemC Language Reference Manual,” IEEE Std 1666-2011 (Revision of IEEE Std 1666-2005), pp. 1–638, Jan. 2012.

[22] C. Wolf, Yosys Open SYnthesis Suite. [Online]. Available: http: //www.clifford.at/yosys/

[23] J. P. Bergmann and M. A. Horowitz, “Vex-a CAD toolbox,” in Proceedings 1999 Design Automation Conference, 1999, pp. 523–528.

[24] S. Takamaeda-Yamazaki, “Pyverilog: A Python-Based Hardware Design Processing Toolkit for Verilog HDL,” in Applied Reconﬁgurable Computing. Springer, Apr. 2015, pp. 451–460.

[25] “Veriﬁc Design Automation.” [Online]. Available: http://www.veriﬁc. com/

[26] J. Decaluwe, “MyHDL: A Python-based Hardware Description Language,” Linux J., vol. 2004, no. 127, p. 5, Nov. 2004.

[27] S. Sato and K. Kise, “ArchHDL: A Novel Hardware RTL Design Environment in C++,” in Applied Reconﬁgurable Computing. Springer, Cham, Apr. 2015, pp. 53–64.

[28] S. Takamaeda-Yamazaki, Veriloggen: A library for constructing a Verilog HDL source code in Python. [Online]. Available: https: //github.com/PyHDI/veriloggen

[29] D. Lockhart, G. Zibrat, and C. Batten, “PyMTL: A Uniﬁed Framework for Vertically Integrated Computer Architecture Research,” in 2014 47th Annual IEEE/ACM International Symposium on Microarchitecture, Dec. 2014, pp. 280–292.

[30] O. Shacham, S. Galal et al., “Avoiding game over: Bringing design to the next level,” in DAC Design Automation Conference 2012, Jun. 2012.

[31] P. Bellows and B. Hutchings, “JHDL-an HDL for reconﬁgurable systems,” in Proceedings. IEEE Symposium on FPGAs for Custom Computing Machines, Apr. 1998, pp. 175–184.

[32] Y. Li and M. Leeser, “HML, a novel hardware description language and its translation to VHDL,” IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 8, no. 1, pp. 1–8, Feb. 2000.

[33] J. Bachrach, H. Vo et al., “Chisel: Constructing Hardware in a Scala Embedded Language,” in Proceedings of the 49th Annual Design Automation Conference, ser. DAC ’12. New York, NY, USA: ACM, 2012, pp. 1216–1225.

[34] M. Odersky, S. Micheloud et al., “An overview of the Scala programming language,” Tech. Rep., 2004.

[35] C.Lattner and V. Adve, “LLVM: ACompilation Framework forLifelong Program Analysis and Transformation,” in CGO, San Jose, CA, USA, Mar. 2004, pp. 75–88.

[36] P. S. Li, A. M. Izraelevitz, and J. Bachrach, “Speciﬁcation for the FIRRTL Language,” EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-9, Feb. 2016.

[37] K. Asanovi´c, R. Aviˇzienis et al., “The Rocket Chip Generator,” EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS2016-17, Apr. 2016.

[38] J. Balkind, M. McKeown et al., “OpenPiton: An Open Source Manycore Research Framework,” in Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems, ser. ASPLOS ’16. New York, NY, USA: ACM, 2016, pp. 217–232.

[39] EEMBC, “Coremark: an EEMBC benchmark.” [Online]. Available: <https://www.eembc.org/coremark/>

[40] C. Celio, D. A. Patterson, and K. Asanovi´c, “The Berkeley Outof-Order Machine (BOOM): An Industry-Competitive, Synthesizable, Parameterized RISC-V Processor,” EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2015-167, Jun. 2015.

[41] R. Goldman, K. Bartleson et al., “Synopsys’ open educational design kit: Capabilities, deployment and future,” in IEEE International Conference on Microelectronic Systems Education, MSE ’09, San Francisco, CA, USA, July 25-27, 2009. IEEE Computer Society, 2009, pp. 20–24.

[42] Xilinx, “ChipScope Pro Debugging Overview.” [Online]. Available: https://www.xilinx.com/itp/xilinx10/isehelp/ise c process analyze design using chipscope.htm [43] D. Kim, A. Izraelevitz et al., “Strober: Fast and Accurate Sample-Based Energy Simulation for Arbitrary RTL,” in 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA), Jun. 2016, pp. 128–139.

致 谢

研究部分由DARPA奖号HR0011-122-0016资助；未来建筑研究中心，星网成员，由马可和DARPA赞助的半导体研究公司项目；DARPA CRAFT（HR0011-16-C-0052）；英特尔敏捷设计科学技术中心；ASPIRE实验室工业赞助商，并与英特尔、谷歌、HPE、华为、LGE、诺基亚、英伟达、甲骨文和三星合作。本文中的任何意见、结论或建议仅为作者的意见、结论或建议，并不一定反映发起人的立场或政策。