

引用格式: 娄冕, 张海金, 杨靓, 等. 基于 Chisel 语言的 RISC-V 处理器设计技术 [J]. 微电子学与计算机, 2021, 38(3): 51-55.
[LOU M, ZHANG H J, YANG L, et al. Design of RISC-V Processor based on Chisel [J]. Microelectronics & Computer, 2021, 38(3): 51-55.]

基于 Chisel 语言的 RISC-V 处理器设计技术

娄 冕, 张海金, 杨 靓, 刘思源, 赵 亮
(西安微电子技术研究所, 陕西 西安 710076)

摘 要: 近年来, RISC-V 在处理器领域的大行其道, 不仅仅在于其开源可扩展的指令集架构属性, 同时也得益于加州大学伯克利分校为其量身打造的敏捷化设计语言 Chisel, 极大降低了处理器设计门槛. 本文基于 Chisel 语言设计实现了一款带有扩展指令协处理器的多核 RISC-V 芯片, 相对于传统的硬件设计语言, 将硬件 IP 的设计与集成周期压缩 50% 以上, 并且依靠丰富的模板资源, 能够快速完成拓扑互连、时序分割、跨时钟域转换等影响处理器整体性能的全局性优化设计, 将芯片验证与实现的迭代周期缩短 30% 以上, 为开源处理器敏捷化开发探索了行之有效的技术手段.

关键词: Chisel; RISC-V; 处理器; 敏捷化

中图分类号: TN47

文献标识码: A

文章编号: 1000-7180(2021)03-0051-05

Design of RISC-V processor based on Chisel

LOU Mian, ZHANG Hai-jin, YANG Liang, LIU Si-yuan, ZHAO Liang

(Xi'an Microelectronics Technology Institute, Xi'an 710076, China)

Abstract: In recently years, RISC-V's popularity in the field of processors is not only due to its open-source and extensible instruction set architecture attributes, but also thanks to Chisel, an agile design language tailored by UC Berkeley, greatly reducing the threshold of processor design. A multi-core RISC-V chip with extended instruction coprocessor based on Chisel is designed and implemented in this paper. Compared with the traditional hardware design language, the design and integration time of hardware IP is compressed by more than 50%. At the same time, relying on rich template resources, it can quickly complete the global optimization design that affects the overall performance of the processor, such as topology interconnection, timing segmentation, and cross-clock domain conversion, reducing the iteration time of chip verification and implementation by more than 30%. The open-source processor agile development has explored effective technical means.

Key words: Chisel, RISC-V; processor; agile

1 引言

当前, 集成电路不断增加的功能复杂度和研发成本使得许多组织和企业无法参与研发前沿, 进一步限制了以处理器为代表的超大规模集成电路技术创新. 2017 年, 美国国防高级研究计划局基于对下一代电子技术路线发展预测, 启动了“电子复兴计划^[1]”, 目标之一就是传统聚焦于工艺微缩而转向

低成本、快速敏捷化的设计方法. 基于这种趋势, RISC-V 成为处理器领域耀眼的明星, 推出的包括处理器硬件原型、基础软件工具链等开源资源掀起处理器领域新的创新浪潮. 这其中, 加州大学伯克利分校打造的敏捷化设计语言 Chisel^[2], 通过更高层次的硬件抽象描述, 借助配套的底层硬件翻译器, 极大提升处理器开发效率, 这也与 RISC-V 自身开源特性相得益彰. 本文介绍基于 Chisel 语言开发的一

款多核 RISC-V 处理器,重点描述 IP 设计、拓扑互连、时序分割以及跨时钟域转换等典型环节的设计方法,为未来敏捷化设计趋势奠定基础。

2 芯片系统级设计语言概述

随着三星宣布攻克 3 nm 工艺所使用的环绕式栅极(Gate-All-Around, GAA)技术^[3],可以预见未来处理器芯片的集成度与复杂度将会进一步提升。传统 RTL 级的抽象层次已经无法应对数以亿计的门级设计与验证周期,必须向系统级设计与验证方法转变,这使得芯片系统级设计语言应运而生,目前已有的芯片系统级设计语言分为三类:

第一类是对传统硬件描述语言的补充,如 SystemVerilog。SystemVerilog 结合了 Verilog 在底层比特级的精确描述以及 C/C++ 在结构体等方面的高层抽象,极大缩减了信号定义与连接映射规模,而对数据类型的丰富以及线程间通信的扩展,使其在系统级验证领域得到了广泛应用。

第二类是高级软件语言,如 C/C++ 等,业界将此类设计方法称为 HLS(High Level Synthesis,高层次综合)^[4]。HLS 将芯片开发重心由底层电路转移到更高层次的算法层面,深挖算法中的程序并行性以提高执行效率,在当前人工智能设计中广泛应用。

第三类是专门的处理器设计语言,如 Lisa^[5]。Lisa 擅长设计 DSP 体系结构,可以描述指令集、流水线时序与行为模型,语法与 C 高度类似,且配套的软件开发工具能够生成包括编译器、链接器、仿真器和调试器等基础工具。

这其中,第二、三类设计语言应用较为专用,仅在特定场景下能够发挥优势,且依赖商用软件的支持,因此并未在通用处理器设计领域得到普及。

3 Chisel 设计语言特点

Chisel 语言推出的目标,就是配合开源的 RISC-V 指令集与翻译器,高效率的将处理器架构师的设计思想迅速转化成处理器硬件原型。这种效率优势得益于该语言的四个特点^[6]:

端口批量映射:该特点与 SystemVerilog 类似,能够快速完成相同类型信号的批量映射,节省大量工作量并防止出错。传统的 Verilog 则不支持此用法。

元编程:Chisel 是一种基于 Scala 的编程语言,通过抽取多个相似类或函数的共性部分形成模板,

能够减少代码冗余,实现全局性的统一配置。SystemVerilog 支持此功能用于验证,但不可综合,而 Verilog 则不支持此功能。

面向对象编程:Chisel 通过继承使得子类自动携带父类特征,可以降低层次化设计复杂度,同时重载特性可以提升代码可读性。SystemVerilog 支持继承,但同样不可综合,而 Verilog 不支持此功能。

函数式编程:Chisel 定义“容器”对包括信号、寄存器、端口、模块等电路元素或元素组合进行抽象,并对容器对象进行批量操作,进一步减少设计描述复杂度;而 SystemVerilog 及 Verilog 均不支持此功能。

4 基于 Chisel 的 RISC-V 处理器设计

4.1 RISC-V 处理器整体结构

基于 Chisel 构建的 RISC-V 处理器如图 1 所示,片上集成 8 个兼容 RV64GC 指令子集的处理器内核,每个处理器具有私有的一级指令 Cache 与数据 Cache 各 16 kB,同时 8 核共享 2 MB 的片上二级 Cache,其中处理器 0 集成自定义扩展指令协处理器,片上采用 TileLink + AMBA(Rev 4.0)为总线架构,集成有静态存储器控制器、DDR 控制器、UART、AES 加解密、DES 加解密、以太网控制器等外设。

全芯片互连拓扑复杂,包括为保证数据一致性的处理器核间 TileLink 协议互连,处理器作为主设备的 TileLink-AXI4、TileLink-AHB、TileLink-APB 协议桥,以及 DMA 作为主设备的 AXI4-TileLink 协议桥。

全芯片时钟资源丰富,划分有处理器与私有 Cache、共享 Cache 与片上互连、以太网控制器、DDR 控制器、慢速外设、硬件调试单元共 6 个时钟域,需要严格确保不同时钟域间的同步问题。

基于上述特点,本文选择从处理器设计到实现中的四个典型环节:IP 设计、时钟域同步、系统集成、时序优化,对 Chisel 在处理器敏捷化开发效果进行阐述。

4.2 RISC-V 处理器详细设计

4.2.1 面向指令扩展的协处理器设计

立足于 RISC-V 指令集支持用户自定义扩展能力,本文基于 Chisel 语言开展面向 RISC-V 指令扩展的协处理器结构设计。基本原理是:扩展指令协处理接收主流流水线译码级解析后发出的操作码、立即数、逻辑寄存器编号等信息,根据预先确定的扩展指

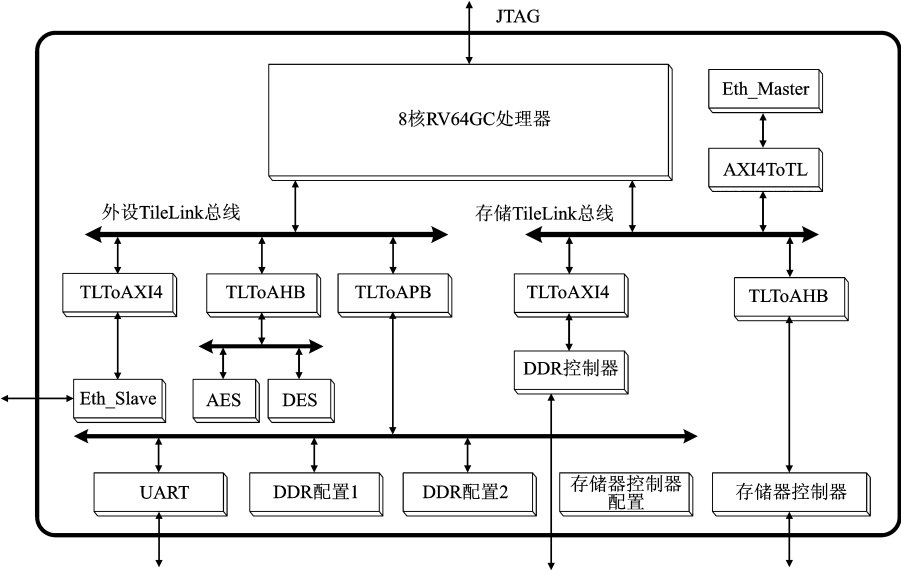
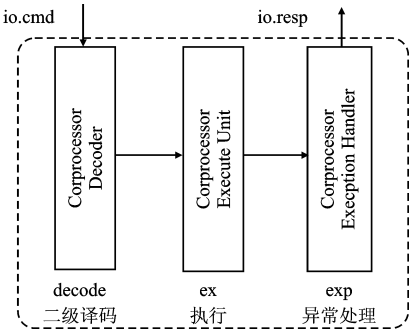


图 1 RISC-V 处理器整体架构

令格式进行二级译码、执行并进行异常处理,将运算结果传递给主流水线写回级完成寄存器文件更新,其结构如图 2(a)所示。



(a) 协处理器结构框图

```
① class XXCoproprocessor (opcodes : OpcodeSet)(implicit p : Parameters)
  extends LazyRoCC (opcodes) {
    override lazy val module = new XXCoproprocessorImp (this)
  }
② class XXCoproprocessorImp (outer : XXCoproprocessor)(implicit p : Parameters)
  extends LazyRoCCModuleImp (outer) with HasCoreParameters {
③   val decode = Module (new CoprocessorDecoder)
④   val ex = Module (new CoprocessorExecuteUnit)
⑤   val exp = Module (new CoprocessorExceptionHandler)
⑥   decode.in <=> io.cmd
⑦   ex.in <=> decode.out
⑧   exp.in <=> ex.out
⑨   io.resp <=> exp.out
}
```

(b) 基于 Chisel 的协处理器模块集成

图 2 协处理器集成设计

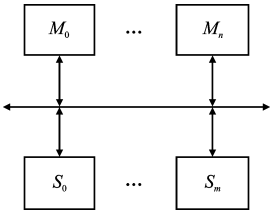
以信号处理协处理器设计为例,其 Chisel 实现代码如图 2(b)所示。运用 Chisel 语言实现的协处理

器代码逻辑清晰,其中①②完成协处理器的定义,该协处理器继承自 RocketChip^[7] 的 LazyRoCC 类,其输入输出接口无需定义,减少信号映射的代码量;③~⑤完成协处理译码部件、执行部件、异常处理部件等声明,这些部件的调用使用函数式编程方式实现,相对传统 HDL 语言其代码更加紧凑、可读性更强;⑥实现流水线向协处理器控制与数据的正向注入,而⑨则实现协处理器向流水线运算结果的反向输出;⑦⑧完成协处理器内部处理部件的相互连接。整体可见,运用 Chisel 的端口批量映射完成整体和局部的端口互连,较传统 HDL 语言更加简洁且不易出现连接错误。

4. 2. 2 片上多核互连体系设计

片上互连是 SoC 设计中信号规模最庞大的环节,极易出现问题,而 Chisel 语言有效简化了片上总线互连体系结构的设计。典型 SoC 结构如图 3(a)所示,其中总线实现多个主机和多个从机之间的互连,主机 $M_0 \sim M_n$ 一般为处理器核、通用 DMA 或调试接口等,从机一般为存储控制器和包括 UART、以太网等在内的外设 I/O。运用传统 HDL 进行 SoC 互连集成往往面临代码量大、信号不对应、多驱动、浮空与方向混淆等问题。Chisel 语言的端口批量映射、函数式编程等特性很好的解决了 SoC 片上互连集成所面临的问题。以 AXI3 总线为例,每增加一个主机或从机接口,运用传统 HDL 进行集成时信号类型声明、主机或从机接口和总线桥的连接至少需要增加 108 行代码,而运用 Chisel 集成仅需要增加

2 行代码. 图 3(b)给出图 3(a)中基于 Chisel 的 SoC 互连结构实现代码, 其中①继承 Chisel 语言的 Module 类^[8]对 SoC 顶层进行定义, ②~④对 SoC 主机和从机进行调用, ⑤运用参数化配置方式实现总线桥的例化, ⑥⑦运用端口批量映射实现总线桥与主机、从机之间的互连.



(a) 典型 SoC 结构示意图

```
① class SoCTop extends Module{
②   val cpu = Module(new CPU)
③   val gdma = Module(new GeneralDMA)
④   val mem = Module(new MemCtrl)
⑤   ...
⑥   val xbar = Module(new BUSXbar (in=4,out = n))
⑦   xbar.io.in (0) <> cpu.io.FetchInterface
    xbar.io.in (1) <> cpu.io.DataInterface
    xbar.io.in (2) <> gdma.io.out
⑧   xbar.io.out(0) <> mem.io.in
    ...
}
```

(b) 基于 Chisel 的 SoC 互连集成

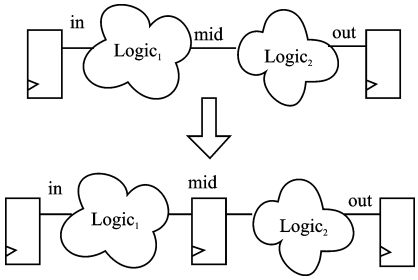
图 3 SoC 互连集成设计

4. 2. 3 面向时序收敛的路径分割设计

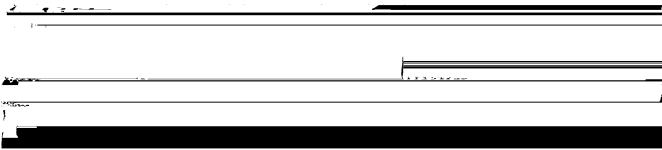
数字集成电路的关键路径往往在后端设计时予以暴露, 此时需要对关键时序路径进行分割. 如图 4(a)所示, 源信号 in 经过逻辑块 Logic1、Logic2 后产生目标信号 out 构成关键路径, 解决办法是进行路径分割, 如在 Logic1 和 Logic2 之间插入寄存器. 运用传统 HDL 进行路径分割时, 不仅需要修改当前逻辑所处的模块, 还需要对调用该模块的所有单元进行时序匹配. 在运用 Chisel 设计时可以将组合逻辑路径较长模块的延迟周期定义为可配置参数, 顶层模块以函数方式进行调用, 当需要进行路径分割时修改该模块的延迟周期参数即可. 如图 4(b)所示, 为实现时序路径分割, 对原有 Chisel 代码(左侧)进行修改的同时修改顶层模块中该模块的延迟参数, 在两个逻辑块之间插入寄存器 mid, 修改后代码(右侧)①使用 RegNext^[8]实现路径分割, ②根据需要在顶层调用时配置具体延迟参数.

4. 2. 4 跨时钟域结构设计

随着 SoC 系统时钟源数量的增多, 在进行 SoC 开发过程中不可避免需要开展跨时钟域结构设计.



(a) 关键路径分割原理



(b) 基于 Chisel 的延迟寄存器插入

图 4 基于 Chisel 的关键路径分割

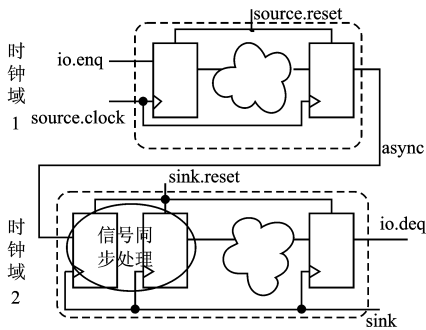
典型的跨时钟域结构设计如图 5(a)所示, 其实现了信号自 source 时钟域到 sink 时钟域下的同步处理. Chisel 具备 Scala 语言的元编程特性, 据此定义如图 5(b)所示的异步队列模板, 该模板实现了跨时钟域信号交互功能, 其对交互数据类型并未预先描述, 其中①对异步队列模板的全局 I/O 进行定义, ②③对内部不同时钟域两端信号进行定义, ④~⑦实现不同时钟域时钟复位信号的连接, ⑧实现了模板内部的信号连接. 运用该模板能够实现不同类型信号的跨时钟域结构的统一处理.

5 芯片实现

本文采用 Chisel 进行处理器芯片设计, 将同等规模处理器的设计与集成周期缩短超过 50% 以上, 同时将由于验证与实现阶段产生的功能与时序问题, 造成设计迭代的周期开销压缩 30% 左右, 极大提高了从设计到流片的研发效率. 本文所设计的 RISC-V 多核处理器, 成功完成所有原型测试、时序收敛与物理验证, 已经完成流片正在进行封装. 最终实现的裸芯片尺寸为 6.6 mm×5.95 mm, 图 6 中所示为版图信息.

6 结束语

在当前摩尔定律放缓的背景下, 深挖处理器体系结构的性能潜力成为推动集成电路持续繁荣的必



(a)典型跨时钟域结构

```
class AsyncQueue [ T <: Data ]( gen : T,
  params : AsyncQueueParams = AsyncQueueParams ()
  extends Crossing [ T ] {
  ① val io = IO( new CrossingIO ( gen ))
  ② val source = Module ( new AsyncQueueSource ( gen , params ))
  ③ val sink    = Module ( new AsyncQueueSink   ( gen , params ))
  ④ source . clock := io . enq _ clock
  ⑤ source . reset := io . enq _ reset
  ⑥ sink . clock := io . deq _ clock
  ⑦ sink . reset := io . deq _ reset
  ⑧ source . io . enq <=> io . enq
    io . deq <=> sink . io . deq
    sink . io . async <=> source . io . async
}
```

(b)基于 Chisel 的异步队列模板

图 5 基于 Chisel 的跨时钟域同步设计

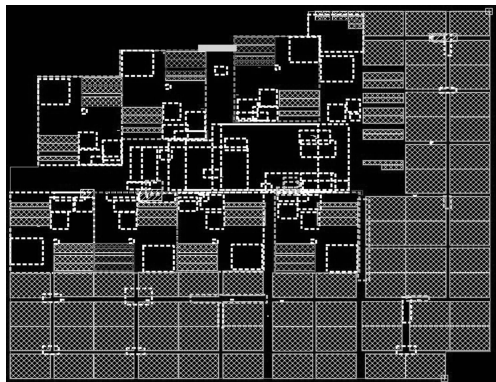


图 6 处理器版图

由之路。开源可扩展的 RISC-V 为处理器领域创新提供了全新的平台,而 Chisel 语言则为创新理念迅速落地创造了条件。本文借助 Chisel 语言,将诸多新的设计思路与 RISC-V 的可扩展特性相融合,完整实现了芯片设计全流程,为处理器的自主可控与敏捷化设计积累了宝贵经验。

参考文献:

[1] DARPA. DARPA electronics resurgence initiative [EB/OL]. [2018-11-23]. <https://www.darpa.mil/work-with-us/electronics-resurgence-initiative>.

[2] LEE Y, WATERMAN A, COOK H, et al. An agile approach to building RISC-V microprocessors [J]. IEEE Micro, 2016, 36 (2): 8-20. DOI: 10.1109/MM.2016.11.

[3] BAE G, BAE DI, KANG M, et al. 3nm GAA technology featuring multi-bridge-channel FET for low power and high performance applications[C]//Proceedings of 2018 IEEE International Electron Devices Meeting. San Francisco: IEEE, 2018. DOI: 10.1109/IEDM.2018.8614629.

[4] MAHAPATRA A, SCHAFER B C. VeriIntel2C: abstracting RTL to C to maximize high-level synthesis design space exploration[J]. Integration, 2019 (64): 1-12. DOI: 10.1016/j.vlsi.2018.03.011.

[5] NAN L M, YANG X, ZENG X Y, et al. A VLIW architecture stream cryptographic processor for information security[J]. China Communications, 2019, 16 (6): 185-199. DOI: 10.23919/JCC.2019.06.015.

[6] 余子濠, 刘志刚, 李一苇, 等. 芯片敏捷开发实践: 标签化 RISC-V[J]. 计算机研究与发展, 2019, 56 (1): 35-48.

YU Z H, LIU Z G, LI Y W, et al. Practice of chip agile development: labeled RISC-V[J]. Journal of Computer Research and Development, 2019, 56(1): 35-48.

[7] TERPSTRA W, WATERMAN A, COOK H, et al. Rocket Chip generator githubrepository [EB/OL]. [2018-11-23]. <https://github.com/freechips-project/rocket-chip>.

[8] BACHRACH J, VO H, RICHARDS B, et al. Chisel: constructing hardware in a scalaembedded language [C]//DAC Design Automation Conference 2012. San Francisco: IEEE, 2012. DOI: 10.1145/2228360.2228584.

作者简介:

娄冕男, (1987-), 博士, 高级工程师。研究方向为嵌入式处理器与 SoC 设计。

张海金(通讯作者)男, (1987-), 硕士, 工程师。研究方向为嵌入式处理器与 SoC 设计。E-mail: 421461708@qq.com.

杨靓男, (1975-), 博士, 研究员。研究方向为计算机系统结构、嵌入式处理器与 SoC 设计。

刘思源男, (1990-), 硕士, 工程师。研究方向为嵌入式处理器与 SoC 设计。

赵亮男, (1984-), 硕士, 高级工程师。研究方向为数字集成电路设计与实现。