

基于 RISC 结构的 Java 处理器研究与设计

张金钟, 胡 平

(南京工业大学 电子信息与工程学院, 江苏 南京 211816)

摘 要: 文中结合 PicoJava 和 JOP 等一些经典的 Java 处理器的优势, 设计了一种基于 RISC 结构的 Java 处理器. 它充分利用了 Java 指令折叠技术和精简指令集处理器的优势, 不仅降低了设计复杂度, 而且在很大程度上提高了 Java 处理器的性能.

关键词: 指令折叠器; Java 处理器; RISC; 字节码; FPGA

中图分类号: TP302

文献标识码: A

文章编号: 1000-7180(2011)07-0061-04

Study and Design for a Java Processor Based on RISC Architecture

ZHANG Jin-zhong, HU Ping

(College of Electronics and Information Engineering, Nanjing University of Technology, Nanjing 211816, China)

Abstract: In this paper, a new Java processor was designed, which combines the advantages of the classic Java processors such as PicoJava and JOP, has high performance and low complexity features, takes full advantage of Java instruction folding technology and the Reduced Instruction Set Computer architecture of advantage. So it not only reduces the design complexity, but has greatly improved the performance of Java processors.

Key words: instruction folding; Java processor; RISC; bytecode; FPGA

1 引言

由于传统的 PC 难以满足更多人对 Internet 随时随地访问的要求, 各种各样能够访问 Internet 的非 PC 类信息设备应运而生, 典型产品包括智能手机、信息家电、智能电话、机顶盒等. 对 Java 语言的支持已经成为这些信息设备不可缺少的功能. 但在基于传统微处理器系统中, 一般采用软件实现 Java 语言. 这种方式速度较慢, 需要占用较多的内存, 对宿主主机硬件和操作系统也有较高要求, 不太适合于嵌入式应用. 为了克服这些缺点, 出现了可以直接执行字节码的 Java 芯片, 但是由于受字节码堆栈结构的限制, 不能有效提高性能.

文中针对 Java 指令执行效率低下的情况, 结合诸如 PicoJava 和 JOP^[1] 等一些经典的 Java 处理器, 设计了一种高性能、低复杂度的 Java 处理器, 它充分利用了 Java 指令折叠技术和 RISC 的优势, 大大提高了 Java 处理器的性能.

2 Java 处理器构架

本 Java 处理器是一种支持指令折叠的微处理器内核, 它是以 RISC 结构为基础实现的, 面向嵌入式应用. 在 RISC 结构流水线基础上加了一级指令折叠与转换单元, 从而把 Java 字节码转换成 RISC 指令. 然后进入 RISC 经典的流水线中执行, 因为经典的 RISC 结构已经有很多的实现模型, 所以整个设计过程关键的一部分包括指令折叠和转换单元, 以及流水线协调工作和两种指令系统不匹配带来问题的解决办法. Java 处理器构架框图如图 1 所示, 系统包括 8KB 的数据缓存(DCache)和 8KB 的指令缓存(ICache), 因此大大提升了整个系统性能.

2.1 流水线的设计

Java 处理器由六级流水线组成. 它是以经典的 RISC 五级流水线的基础上扩展的, 由取指级(IF)、字节码折叠转换级(FB)、译码级(ID)、执行级(EX)、内存访问级(MA)和寄存器回写级(WB)

组成.

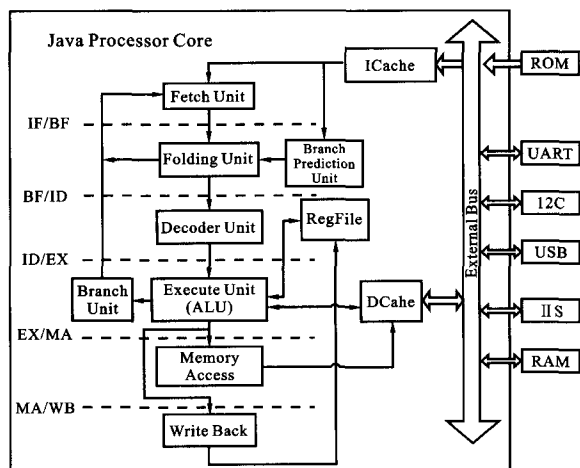


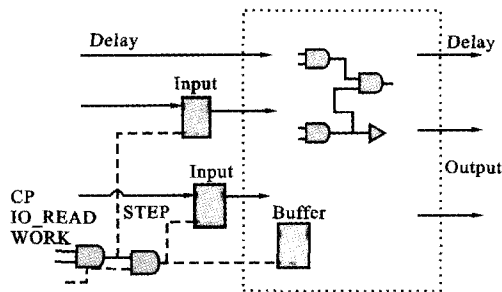
图 1 Java 处理器整体架构图

所有的流水线级都有确定统一的时序和控制规则. 它们基本上都按照图 2(a)的方式构建的. 流水线每一级的接口包括输入的同步数据信号和控制信号, 输入时钟信号 CP 和工作信号 IO_READY 和 WORK 信号, 以及一些偶尔不能与时钟上升沿同步到达的延迟信号. 这些信号可能需要延迟到在时钟周期内才到达. 在对外端口一边, 流水线级在时钟周期结束之前产生数据和控制信号, 但是有时也有延迟送出的信号. 流水线级的内部机构包含一个或多个寄存器, 它们根据在流水线级起始时的时钟和工作信号的状态完成装载. 大多数输入寄存器都只在流水线拍有效 (WORK=1) 是进行装载, 其它寄存器作为缓冲器, 可以时时进行装载.

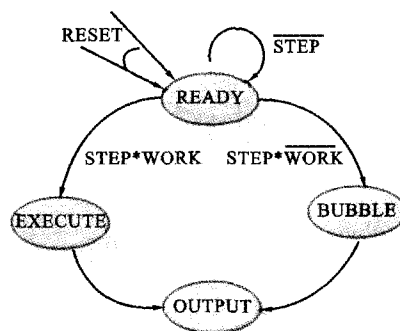
图 2(b)给出了流水线各级运行周期的状态图. 从任何一个状态, RESET 信号都可以把状态置位到 READY, 只有在 STEP 信号有效的情况下状态机才能脱离 READY 状态. 如果 WORK 信号无效, 流水线级转到 BUBBLE 状态执行一个空操作. 例如, 当 JUMP 跳转指令执行到执行级, 但却没有实际的操作要进行时, 就是这种情况. 当输出信号有效时, 流水线状态转移到 OUTPUT, 并且在短暂的延迟后返回到起始状态 READY.

2.2 Java 指令折叠原理

Java 虚拟机是一个基于堆栈操作的机器, 并且大多数操作都必须基于堆栈的, 因此这些数据的相关性约束了指令执行的并行性. 并且严重影响了系统的性能. 于是 Sun Microelectronics 公司提出了 Java 指令折叠技术. 当指令在执行的时候, 一个预定义和预存储的折叠类型同在指令顺序流中的字节码匹配. POC 是由 Producer、Operator、Consumer



(a) 流水线级的基本原理



(b) 流水线级的行为

图 2 流水线设计

组成, 所有的字节码都被分成三种主要的 POC 的类型和典型堆栈操作. 步骤如下:

- ① producer 把被访问的数据从常数池或局部变量区写到操作数堆栈上.
- ② operator 从堆栈顶取出数据.
- ③ operator (ALU、逻辑指令、分支指令或复杂指令) 操作堆栈上的数据.
- ④ 当需要时 operator 会把结果写到堆栈上.
- ⑤ consumer 从堆栈上取得数据并且写回到局部变量区.

这个执行过程在图 3(a)用数字表示.

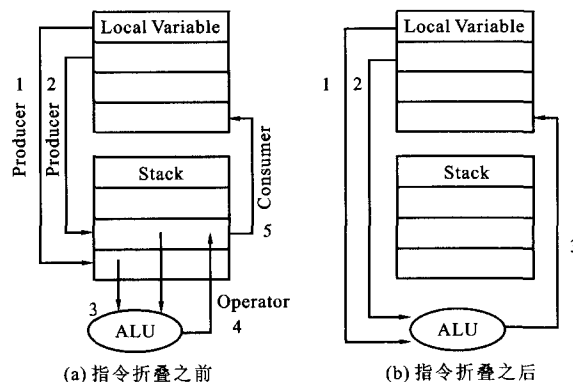


图 3 指令折叠过程

如果在堆栈操作过程中出现了数据相关,就把它们折叠在一起.正如图 3(b)第一步所示折叠之后的执行顺序也将改变.过程如下:

- ① 直接从 producer 的源头取得数据.
- ② operator 操作那些数据.
- ③ 如果需要的话 operator 把结果写回到 consumer 的目的地.

在这个例子中,指令访问的数量由 5 减到 3,因此系统性能在通过折叠以后有了明显的提高.在 POC 折叠模型中,两种连续的 POC 字节码指令被送到 POC 折叠检测单元去决定它们能否折叠.能折叠的字节码被称为 FBI.进一步的折叠去决定那两个字节码指令能否用后一个指令编程一个更大的 FBI.生成的信息被重新送回到 POC 折叠模型中去找连续字节码指令的最大数.

2.3 Java 复杂指令的设计

现在已经出现一些经典结构的 Java 处理器,但是它们的设计都并不能够仅仅通过硬件来执行所有的 Java 指令.由于 Java 指令本身是一种复杂指令集,其中有些指令(如面向对象指令 new)非常复杂,显然这种复杂指令并不适合流水线的处理.因此如何提供一种有效的途径来解决 Java 的复杂指令处理,对提高流水线的整体性能有着重大的意义.

复杂指令的设计可以分为两类,一类使用频率很低的指令通过软件来实现,另外一类就是较为频繁使用的指令则通过微码实现. PicoJava 中使用的异常处理方案.在这种方案中当译码段遇到特定的复杂指令,其将会产生相应的异常信号.此时处理器将会进入特定的入口,通过软件的形式处理该复杂指令.虽然通过异常处理模式能简化硬件的设计,然而通过异常的方式,处理复杂指令将会消耗大量的时钟周期.而在 JOP 体系结构中所有的复杂指令都通过微指令的形式实现,它的每一条 Java 指令都在微指令 ROM 中拥有一条或多条映射.这样可以简化 Java 处理器的设计,每一条 Java 指令可以通过查找微码表来实现,但是在性能和设计灵活性上却要值得考虑.

软硬件划分的标准通常是基于以下两点考虑:如果速度要求高,那么更多的功能需要用硬件实现,如果成本开销要低,那么就考虑尽量用软件实现功能.下面就以典型的 Java 字节码指令 imul 为例^[2],两个 32bit 有符号整数相乘,分别用硬件描述语言、微码和 Java 字节码三种不同的方式实现.表 1 为这

三种方法的资源率和执行时间.执行时间是用两个同为负的操作数来衡量的,即软件的执行情况最坏.用软件 Java 来执行的速度要比用微码慢,这是因为把 Java 方法从主存储器加载到字节码缓存中耗费了一定时间.

表 1 imul 指令三种实现方法的比较

	硬件电路	存储单元	执行时间
HDL	156	10	35
微码	0	73	750
软件	0	0	2300

而在设计中,只为那些需要多条简单指令实现的复杂指令设置微指令 ROM 映射.但是还有一些指令由于太过复杂,用微码实现较难.因此我们选择用软件陷阱来实现.从而使硬件资源与处理器性能达到折中.充分发挥了 PicoJava 和 JOP 两种处理器的优势,使性能得以提高.图 4 显示了复杂指令在微指令 ROM 中的实现的过程.

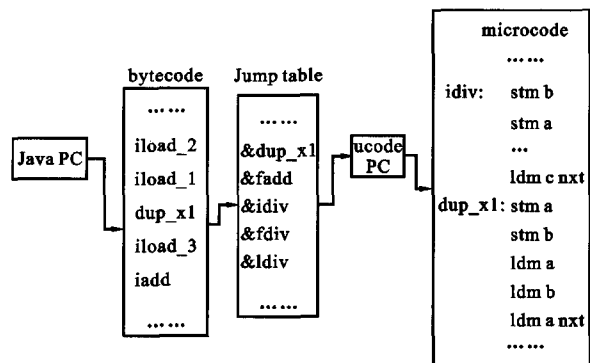


图 4 Java 微码查找原理

3 Java 处理器在 FPGA 上的实现与评估

3.1 VerilogHDL 实现

本设计是用 VerilogHDL 对 Java 处理器来进行描述实现的,如图 5 所示,最上层是 Java 处理器的顶层模块,它又包含整个处理器的四大模块,分别是处理器内核、存储器接口、外设接口和扩展模块接口,这四大模块再进行细化.所以整个设计纵向分层逐步细化,横向通过流水线协调逐级设计.分六级流水线分别为取指级、指令折叠转换级、译码级、执行级、访存级和回写级.其中每一级用一个时钟周期来完成,所以每一条指令要花费六个时钟周期.但是由于流水线的原理,每条指令真正的执行时间还是一个时钟周期.处理器内部是 32 位数据总线,对外是 8 位的 Wishbone 总线.

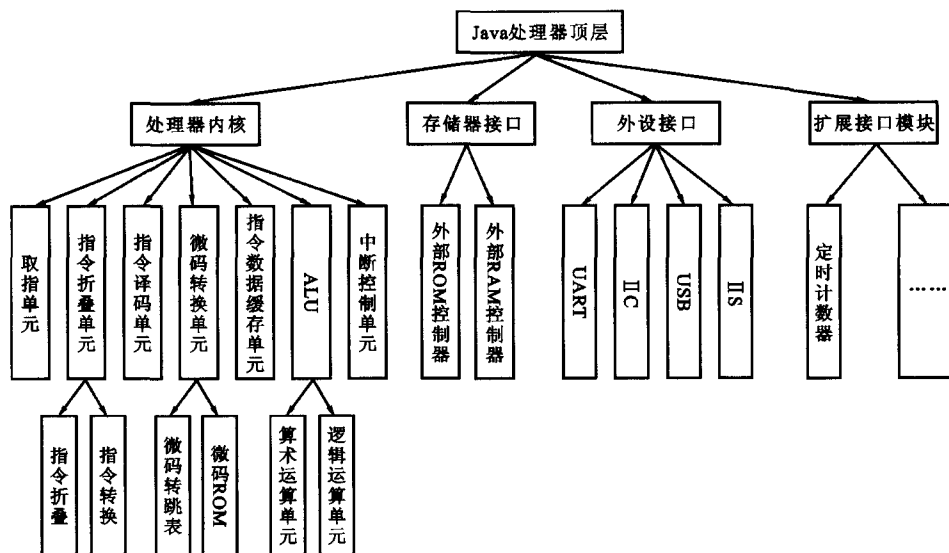


图5 Java 处理器 VerilogHDL 实现构架

3.2 仿真与评估

文中采用 Modelsim6.1 进行逻辑仿真,以确定这些设计的功能、逻辑关系及定时关系是否满足设计要求. 系统设计过程中一般要进行三次仿真:行为级仿真、RTL 级仿真和门级仿真. 各级所要达到的仿真的目的是不一样的,同时对 HDL 语言的描述要求也有所不同.

本 Java 处理器实现的最终配置是 8K 字节的数据缓存和 8K 字节的指令缓存,使用了 26.8K 个逻辑单元. 详细数据如表 2 所示.

表2 处理器各模块逻辑单元使用数

处理器模块	逻辑单元
数据缓存单元	1426
指令缓存单元	4121
执行单元	6300
字节码折叠单元 ^[3]	1053
微码转换单元	2895
软件陷阱单元	124
流水线控制单元	568
寄存器控制单元	3589
堆栈缓存	6520
总线接口单元	34
SRAM 接口	135
总和	26765

从表 2 中的数据可以看出,字节码折叠单元只用了 1053 个逻辑单元^[3],这种算法不仅使 Java 指

令的 POC 重新分类,并且提升了 Java 处理器性能.

4 结束语

文中借鉴了 RISC 构架成熟的流水线结构和 Java 指令折叠原理设计了一种实用的 Java 处理器,本 Java 处理器有六级流水线组成,基本上根据 Java 虚拟机规范来设计,从性能、功耗和复杂度之间寻找合理的折中. 由于应用指令折叠技术,比起基于堆栈实现的 Java 处理器来说,在性能上有很大提高. 处理器内核的大部分功能已在 FPGA 上实现,但是比如中断控制、多线程处理、流水线优化和外设接口等将在以后的工作中进一步研究实现.

参考文献:

- [1] Schoeberl M, JOP: a java optimized processor[J]. Lecture Notes in Computer Science, 2003;346—359.
- [2] 周佳筠. Java 处理器的 VLSI 实现[D]. 浙江: 浙江大学, 2006.
- [3] Tan Yiyu, Anthony S Fong, Yang Xiaojian. An instruction folding solution to a java processor[J]. Network and Parallel Computing. 2007;4672(2010):415—424.

作者简介:

胡平 男, (1962—), 副教授, 硕士生导师. 研究方向为嵌入式系统、计算智能.

张金钟 男, (1984—), 硕士. 研究方向为嵌入式系统、信号处理.