文章编号:1674-2974(2024)06-0187-08 DOI:10.16339/j.cnki.hdxbzkb.2024238

基于RISC-V的IOMMU设计

王镇道†,班贵龙,胡锦,焦旭峰
(湖南大学 物理与微电子科学学院,湖南 长沙 410082)

摘 要:在半导体技术受到管控的背景下,实现芯片的完全自主可控已成为现今半导体技术发展的重点.由于RISC-V具有开源、应用广泛的特性,研究RISC-V架构对于我国微处理器的自主可控具有重要研究意义.在微处理器系统中,由于物理资源的有限性和直接访问存储可能潜在危害,DMA访问I/O设备时将会受到诸多限制,从而影响访问性能.目前主流的方法是通过将I/O事务虚拟化,可以很好地解决这一问题.本文首次提出了一种基于RISC-V的I/O虚拟化架构,极大地加速了I/O访问进程,仅花费几个时钟周期就可快速完成I/O设备对内存的DMA请求.本设计将来可以作为IP,集成到RISC-V架构的处理器中,加速I/O设备对内存的访问.

关键词:虚拟化;缓存;RISC-V
中图分类号:TP332 **文献标志码**:A

Design of IOMMU Based on RISC-V

WANG Zhendao†,BAN Guilong,HU Jin,JIAO Xufeng
(College of Physics and Microelectronics Science, Hunan University, Changsha 410082, China)

Abstract: In the realm of semiconductor technology control, achieving complete autonomous chip control has emerged as a focal point in today's semiconductor technology advancement. Given its features of open source and widespread adoption, the study of RISC-V architecture holds significant importance for enabling microprocessor autonomous controllability. Within microprocessor systems, limitations on physical resources and potential risks associated with direct storage access necessitate restrictions on DMA access to I/O devices, thereby impacting access performance. The prevailing approach involves virtualizing I/O transactions to effectively address this issue. This article firstly proposes a I/O virtualization architecture based on RISC-V, which greatly accelerates the I/O access process, this architectrue consums a few clock period to complete DMA requests from I/O devices to memory. This design will be integrated into RISC-V architecture CPU as an IP, accelerating the access of I/O devices to memory.

Key words: virtualization; cache memory; RISC-V

在半导体技术受到封锁和制裁的背景下,实现 芯片的完全自主可控已经成为现今半导体技术发展

* **收稿日期**:2023-05-30
基金项目:国家自然科学基金资助项目(Z202301432394), National Natural Science Foundation of China(Z202301432394)
作者简介:王镇道(1974—),男,湖南沅江人,湖南大学副教授,博士
† 通信联系人,E-mail:Wangzd@hnu.edu.cn

的重点.RISC-V作为一种开源指令集架构,在实现芯片自主可控上具有重要的意义.RISC-V在高性能处理器、服务器、人工智能及机器学习领域有广泛应用前景,RISC-V生态发展正呈现明显的加速态势[1].计算机体系中,虚拟化技术实质上是对具体的物理计算资源抽象的过程,通过提高具体物理资源的利用率来大幅提升计算机性能[2].IOMMU(I/O Memory Management Unit)即I/O虚拟化,属于虚拟化的一种,其主要功能是对I/O设备发起的DMA与中断进行重映射[3-5],以此来实现I/O设备与虚拟机之间的隔离与保护.目前暂未出现基于RISC-V的I/O虚拟化,且原有体系结构的I/O虚拟化不适用于RISC-V.本文首次提出了一种基于RISC-V的I/O虚拟化架构,极大地加速了I/O访问进程,可快速处理I/O设备发起的DMA请求.本设计可以作为IP,集成到RISC-V架构的处理器中,改善大量I/O访问下的系统性能.

根据使用的技术方案不同,I/O虚拟化也分为全虚拟化、半虚拟化和硬件辅助虚拟化.其中,全虚拟化指的是在管理程序(hypervisor)或宿主机完全模拟设备的行为,虚拟机(Virtual Machine,VM)对外设的所有操作都会触发VM陷入hypervisor,之后由hypervisor将VM对设备的操作信息截获下来进行处理[6].半虚拟化本质上也是使用软件来模拟设备的行为,VM知道自己是在虚拟化环境当中,并且要求VM上的软件要配合虚拟化环境来完成设备模拟[7-8].PCI-SIG协会在以上3种I/O虚拟化之外,提出了Single Root I/O Virtualization(SRIOV),即单根虚拟化,该方法中单个物理设备可供多个虚拟机同时访问[9-10].硬件辅助I/O虚拟化是将虚拟化的功能通过一块专用的硬件来实现设备直通[11-12].业界存在3种主流硬件辅助I/O虚拟化技术,分别是Intel公司的VT-d、ARM公司的SMMU与AMD公司的IOMMU,本文实现的IOMMU也属于硬件辅助类型.

为实现IOMMU的I/O虚拟化功能,IOMMU中的DMA重映射遵循RISC-V特权架构下的两级地址转换内容.基于I/O虚拟化,本文对RISC-V特权规范中虚拟地址转换等内容进行研究.使用Verilog HDL对电路进行建模,设计了一款完全符合RISC-V IOMMU规范的电路,该电路主要功能包括:实现基于device_id与process_id的设备上下文(Device Context)和进程上下文(Process Context)翻译、单级或两级虚拟地址翻译、设备MSI中断的重映射、PCIe设备的ATS接口、位于内存的队列接口等功能.为提升IOMMU的性能,同时对IOATC(I/O Address Translation Caches)进行研究,使用PLRU更新算法来设计对应Cache.

经过实验证明,相较于一般的DMA访问性能,本文设计的RISC-V架构IOMMU在较高命中率的前提下极大地提升了I/O访问的性能,实现I/O设备与系统内存的安全性和隔离性.

1 IOMMU架构介绍

1.1 设计架构

RISC-V下的IOMMU与Intel的VT-d、AMD的IOMMU、ARM的SMMU的工作原理和流程大致相似.位于可以进行DMA的I/O设备与系统内存之间,属于系统级的内存管理单元.该方案主要由TCU和IOATC模块组成,可以实现I/O设备对系统内存的安全访问,并加速其访问进程.

IOMMU在DMA转换时,使用软件提供的页表来翻译虚拟地址的过程可能会影响DMA访问(需要访问主存).CPU中的MMU通常使用TLB(Translation Look-aside Buffer)来缓存翻译的结果,这样可以加速地址翻译过程.RISC-V的IOMMU中也使用到了类似的机制,即IOATC(I/O Address Translation Caches).

在PCIe的情况下,如设备端支持ATS服务,根端口可将ATS接口合并至IOMMU,以支持IOMMU的PCIe ATS协议.在支持ATS的PCIe端点设备内部,有设备端的ATC(DevATC,Device Address Translation Cache),用来保存PCIe设备使用ATS协议从IOMMU处的转换.

本论文中设计的IOMMU在SOC系统中的位置如图1所示.I/O设备通过AXI总线向IO Bridge发送DMA请求,IO Bridge初步解析请求后,向IOMMU发起地址转换请求.在IOMMU转换完成或对主存进行隐式访问时,IO Bridge会在PMA与PMP检查无误后,通过AXI总线来对N-core(系统互联)发起访问主存的请求.RISC-V core通过APB总线来对IOMMU内部寄存器来进行读写配置.对于PCIe设备支持的ATS服务,采用DTI-ATS协议,该协议本质上仍是基于AXI总线,且PCIe设备发起ATS请求同样经过AXI总线.

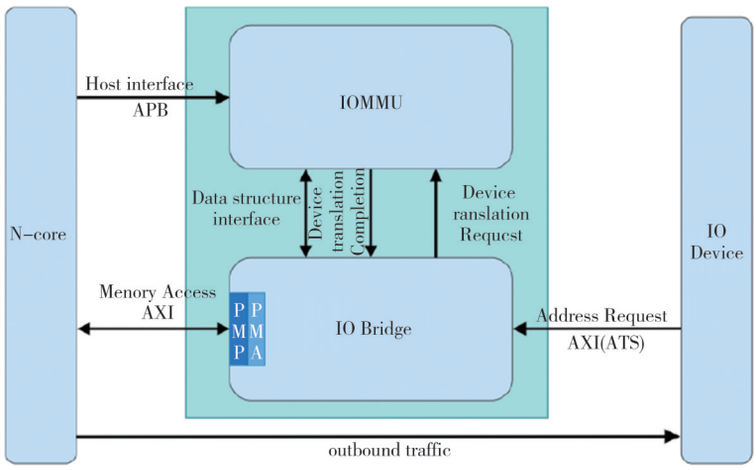


图 1　SOC 中的 IOMMU 架构

Fig.1　IOMMU architecture in SOC

当I/O设备具有DMA能力时,可以选择直接连接到IO Bridge,I/O设备可以发送使用IOVA(VA、GVA、GPA)的事务,由IO Bridge来调用IOMMU将IOVA地址转换为系统物理地址(SPA).当IOMMU未开启时,IO Bridge不会调用IOMMU来处理DMA事务,DMA事务直接穿过IO Bridge进行下一级传递.

在PCIe的情况下,如设备端支持ATS服务,根端口可将ATS接口合并至IOMMU,以支持IOMMU的PCIe ATS协议.在支持ATS的PCIe端点设备内部,有设备端的ATC(DevATC,Device Address Translation Cache),用来保存PCIe设备使用ATS协议从IOMMU处的转换.

IOMMU主要分两部分:TCU(Translation Control Unit)与IOATC(IO Address Translation Cache),如图2所示.其中,IOATC类似于MMU中的TLB,用来缓存IOMMU在地址翻译过程中所需要访问的数据类型,包括DDT中的设备上下文DC、PDT中的进程上下文PC、第一阶段或第二阶段地址翻译的页表项、MSI的页表项等内容.

在从主存中读取特定数据的过程中往往需要访问多级页表,这意味着需要多次访问主存,而访问主存带来的时间代价是相当大的.而在IOATC中缓存

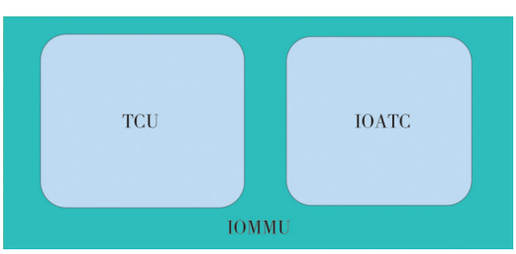


图 2　IOMMU 主要模块

Fig.2　IOMMU main module

IOMMU在page table walk过程中读取回来的页表项之后,每当需要从主存中查询数据时,首先查询IOATC中对应的Cache,如命中则直接取Cache的内容,不需要访问主存,可以大大地降低时序上的延迟,从而提高性能.

存在Cache就会有一致性问题,IOMMU中对于IOATC一致性的维护是通过与软件的交互完成的.当内存中的页表结构有更新或失效操作时,会向IOMMU的命令队列CQ发送相应的失效命令,IOMMU在取到失效命令之后,向IOATC发送失效请求,通过请求中包含的tag信息来对全部或部分Cache进行失效操作.通过这样的方式来维护Cache与主存中数据的一致性.

1.2　功能点分析

在RISC-V IOMMU规范的基础上,结合具体I/O虚拟化应用的需求,将本次设计的功能特性介绍如下:

· 支持存储于内存的DDT结构,可以通过DMA请求中提供的硬件信息device_id来定位并取出DDT中的设备上下文DC.所支持的device_id位宽最宽可为24 bits,可选地支持3、2、1级DDT结构.

· 支持存储于内存的PDT结构,可以通过DMA请求中提供的硬件信息process_id来定位并取出PDT中的进程上下文PC.所支持的process_id位宽最宽可为20 bits,可选地支持3、2、1级PDT结构.

· 支持基于页面的虚拟地址系统,与RISC-V特权规范中虚拟地址翻译保护的内容一致,可以使用与CPU使用公共的页表,也可以对IOMMU使用单独的页表.

· 规范中支持最高57位的虚拟地址,56位物理

地址和59位的GPA.在本设计中,支持Sv39和Sv48,故本文支持最高48位虚拟地址和50位的GPA.

· 支持上文中所述的对于虚拟中断文件写入MSI的重映射,MSI的页表项可参见RISC-V Advanced Interrupt Architecture(AIA)协议.

· 硬件性能监视器(Hardware Performance Monitor).

根据具体应用的不同,所支持的功能也有所差别,软件可以通过读取capabilities寄存器的内容来发现IOMMU所具体实现的功能.

2 TCU设计

TCU的工作流程如图3所示.IOMMU在接收到地址转换请求后,需要先根据地址转换请求中附带的信息,来获取发起请求外设的设备上下文DC和进程上下文PC.由于IOATC中存储了部分设备上下文的页表项,所以先向IOATC发送一个查询DC的请求,如果IOATC中有对应的DC项,则直接获取DC;如IOATC miss,则TCU需要根据ddtp寄存器中的iommu_mode字段(决定DDT是几级页表)和ppn字段(根DDT的物理页号)以及device_id来进行page table walk去主存中获取DC的内容.

在成功获得对应的设备与进程上下文后,也就获得了对IOVA进行地址转换的策略及相关信息.地址转换的流程见图4,以两级地址转换为例,在接收到地址转换请求后,首先根据PSCID、GSCID以及IOVA去IOATC中查询是否有对应该第一阶段地址翻译的页表项,若IOATC命中则直接取出该页表项;否则进行PTW,每一级地址翻译的GPA还需要调用第二阶段地址翻译转换为SPA.

由图4流程图的描述中,可以看到查询页表项的两种方式,一种是通过查询IOATC,另一种是通过Page table walker去主存中查询.如图4所示,存在两种获取数据结构的方式:查询IOATC或者通过Page table walker去内存中查询.由于IOATC位于IOMMU内部直接与TCU进行交互,在IOATC中查询命中返回仅需1个或2个时钟周期,时间代价很小.然而当IOATC miss的情况下需要通过多级访问主存来查询页表,访问主存的时间可能是几十个时钟周期,PTW情况下查询DC的时间代价极大.因此,高命中率的

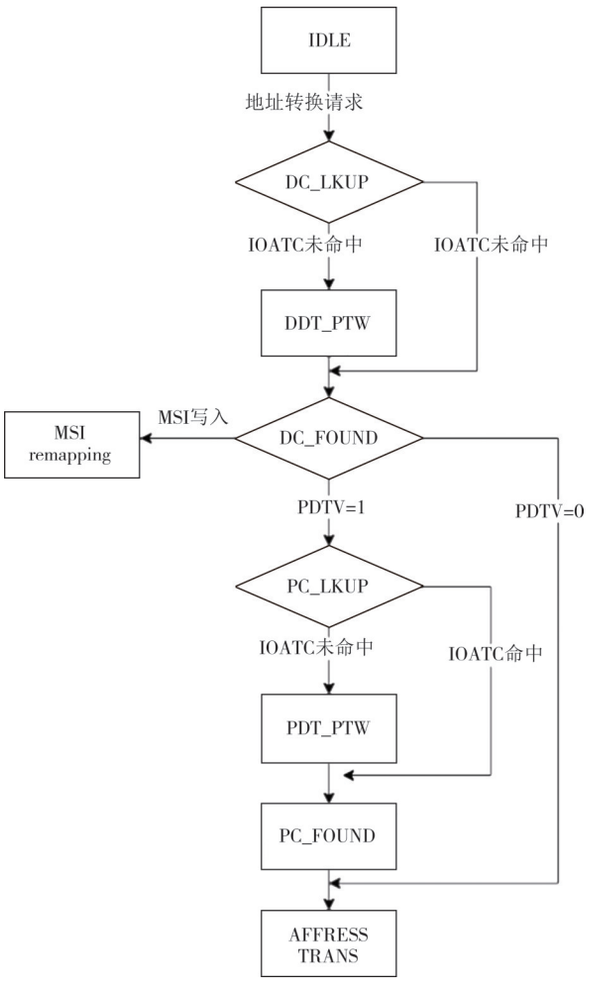


图3 上下文获取流程

Fig.3 Context acquisition process

IOATC会大大减少IOMMU的延时,是IOMMU整体设计的关键,接下来详细介绍IOATC的设计.

3 IOATC设计

IOATC用于缓存IOMMU页表查找过程中查找到的叶子项页表信息,即页表条目.当IOMMU接收到来自设备的地址转换请求时,其根据当前地址转换的阶段,首先到IOATC中获取相应页表数据的缓存信息.若命中地址转换所需的信息,则可以快速实现地址的转换,而不需要花费大量的时间到系统内存中去获取地址翻译所需的数据结构和所必须经历的页表转换.在IOMMU中,专门用于页表转换的模块为TCU,TCU与IOATC之间的主要接口如图5所示.

IOATC中的各个子模块缓存相应数据结构页表的子块数据,也就是IOMMU在地址转换过程中所必

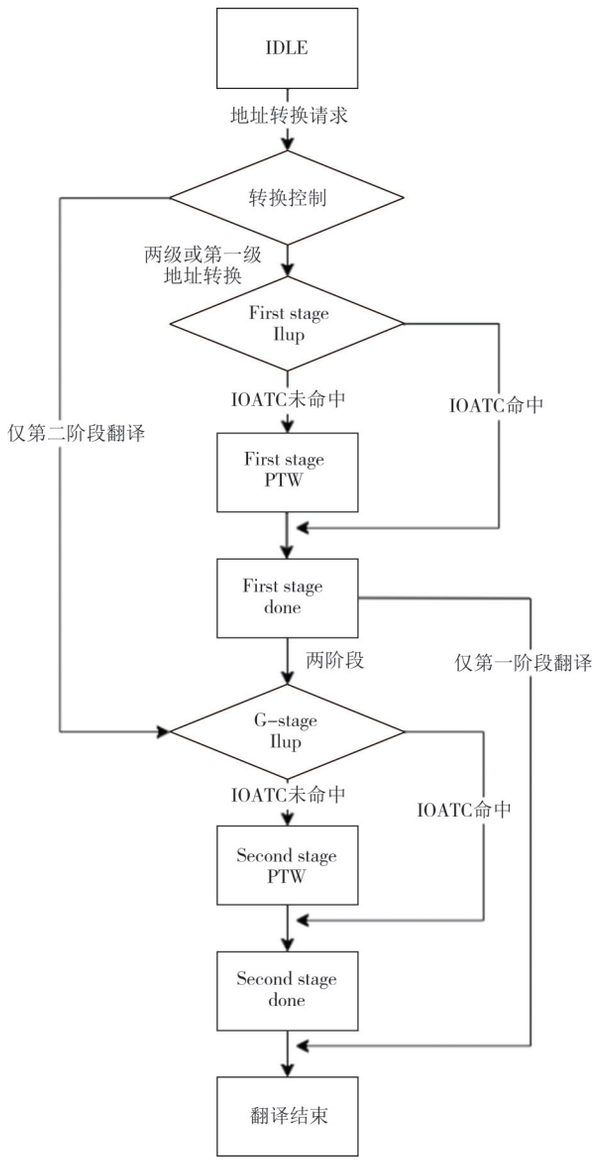


图 4　地址转换流程

Fig.4　Address translation process

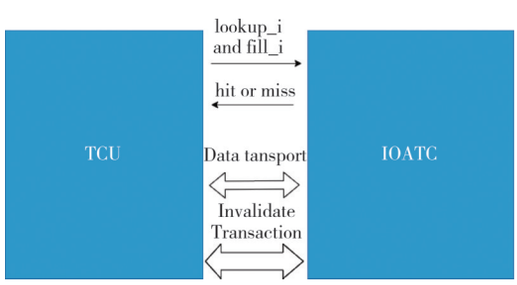


图 5　IOATC 与 TCU 交互接口

Fig.5　Interface between IOATC and TCU

需的设备上下文(DC)、进程上下文(PC),以及各阶段页表地址转换的 PTE 等数据信息.IOATC 中包含的子模块如图 6 所示.

其中每个子模块的功能如下,每个子模块为相应数据结构的缓存项以及控制逻辑:

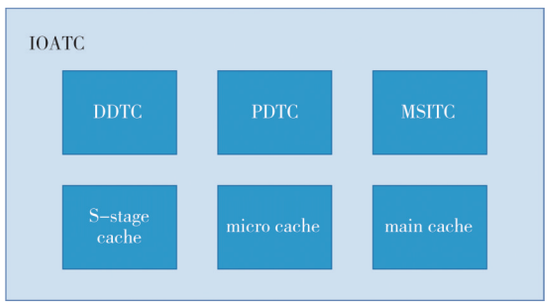


图 6　IOATC 子模块

Fig.6　IOATC submodule

•DDTC:缓存设备目录表(DDT)的叶子项数据,当命中时向 walke 返回设备上下文(DC).

•PDTC:缓存进程目录表(PDT)的叶子项数据,当命中时向 TCU 返回进程上下文(PC).

•S-stage_Cache:缓存单阶段(S-stage)地址转换的叶子项数据.

•MSITC:缓存直写模式的 MSI 表项数据,加速 MSI 地址转换.

•micro_Cache:缓存两阶段地址转换(VS-stage 和 G-stage)的系统物理地址(SPA),当 tag 标志匹配时,可直接实现虚拟地址到物理地址的转换.

•main_Cache:在两阶段查找中,当 micro_Cache 查找 miss 时,开始查找 main_Cache.main_Cahce 中的 VS-stage_Cache 和 G-stage_Cache 分别缓存了 VS-stage 地址转换的叶子项数据和 G-stage 地址转换的叶子项数据.

在设计了 IOATC 后,IOMMU 的访问 IOATC 工作流程如图 7 所示.

当设备发出地址转换请求时,IOMMU 首先到 DDTC 中查找设备对应的设备上下文 DC,若命中 DDTC 则不需要花费较长的时间去内存中查找 PDT 页表.获取到设备地址转换所必需的 DC 之后,根据 DC 中的信息判断该地址请求的类型:

·若请求为 MSI 地址转换,则 IOMMU 将访问 MSITC,如果命中了 MSITC 中的缓存项,则完成 MSI 的地址转换,并将其路由到中断控制器.如果未命中则需要花费较长时间到内存中访问 MSI 页表,再完成后续的过程.

·若请求为单阶段地址转换,则 IOMMU 将访问 S-stage Cache.如果在访问过程中命中了缓存项,将使用缓存中的物理页号与虚拟地址中的偏移量组成物理地址,快速完成单阶段地址转换的过程.如果未命中缓存项,则需要耗费大量时间到系统内存中查

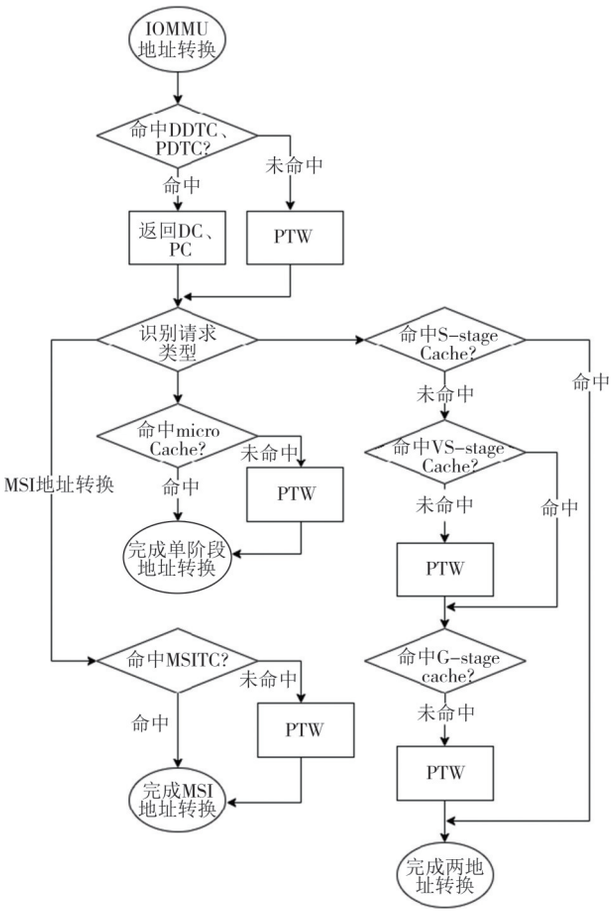


图 7 IOMMU 访问 IOATC 工作流程

Fig.7 IOMMU access IOATC workflow

找 S 阶段页表.

·若请求为两阶段地址转换,则 IOMMU 首先将访问 micro Cache.如果命中 micro Cache 的缓存项,那么可以直接实现由虚拟地址到物理地址的转换.如果未命中 micro Cache,则将访问 main Cache 进行两级查找.若在 main Cache 中命中了 VS 阶段的缓存项,则可直接实现由虚拟地址到客户物理地址的转换,然后再对 G 阶段的缓存项进行查找.若命中了 G 阶段的页表项,则实现客户物理地址到系统物理地址(SPA)的转换,从而完成了两阶段地址转换.在查找 main Cache 的过程中,若未命中缓存项都将花费大量时间到内存中进行页表翻译.

Walker操作从内存中获取数据的时间往往是命中 IOATC 的数百倍,而命中 IOATC 可极大节省访问的时间.由图 7 流程中可以明显看到 IOATC 避免了从系统内存中获取数据结构,加速了 IOMMU 地址转换的进程.尤其在两阶段地址转换中,若命中 IOATC 则可以直接实现两阶段地址转换,而不需要花费大量时间去系统内存中进行各级查找的操作.

4 仿真结果

4.1 上下文 PTW

在将 TCU 与 IOATC 集成为 IOMMU 后,对上下文 PTW 进行仿真的结果如图 8 所示.

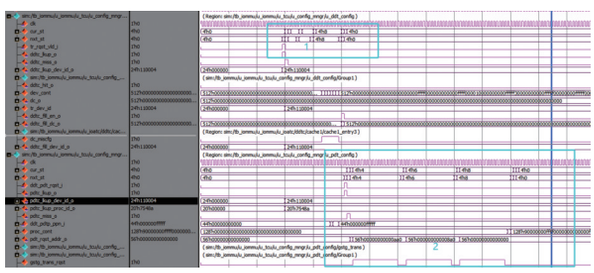


图 8 上下文 PTW 仿真结果

Fig.8 Context page table walk simulation result

IOMMU 在初始化后,IOATC 未缓存任何内容,所以各个数据结构都需要进行 PTW 来获得主存中的数据.在 PTW 结束后,会将相关信息填充至 IOATC 当中.两阶段地址翻译的 PTW 过程见图 9.

4.2 两阶段 PTW

如图 9 中箭头所示,可以看到两阶段地址转换中的第一级地址转换过程中,每次进行 PTW 都会调用第二阶段地址转换.

当第一阶段地址转换结束后,再使用转换过的 GPA 再进行第二阶段地址转换,转换结束后 ptw_done 信号拉高,同时输出最后的结果 SPA.且在每次地址转换完成后,都需要对相应的 Cache 进行填充操作.

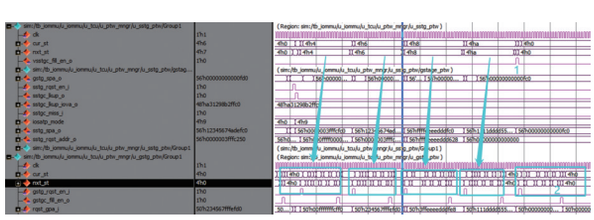


图 9 两阶段地址转换 PTW

Fig.9 Two stage translation page table walk

4.3 IOATC 全命中

IOMMU 在正常工作过程中全部命中 IOATC 的仿真如图 10 所示.

对比命中 IOATC 与进行 PTW 的时序情况,可见当 IOATC 未命中时,进行 PTW 所带来的耗时是

IOATC命中时的几十倍甚至几百倍.所以高命中率的IOATC会大幅提升IOMMU的性能.

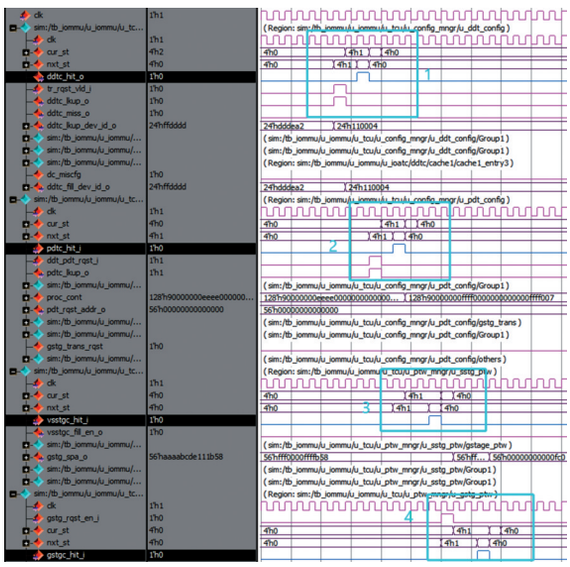


图10　地址转换全命中

Fig.10　Address translation all hit

4.4　MSI重映射

IOATC中缓存有MSI重映射所需要的PTE,所以在处理MSI中断重映射时,需要查询MSI Cache.如图11所示展示了MSI Cache命中时的情形.
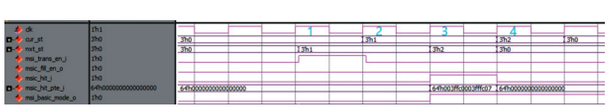


图11　MSI缓存命中

Fig.11　MSI Cache hit

在第1个时钟周期时,接收到MSI重映射请求,nxt_st进入MSI_LOOK_UP状态(3'h1),向Cache发送gpa、gscid以及查询请求.第2个时钟周期等待Cache返回信号,第3个时钟周期Cache返回hit信号及命中的MSI PTE,nxt_st进入MSI_READY状态,表明已读取到MSI PTE内容.第4个时钟周期将PTE读取到MSI模块内部.

MSI重映射地址转换模式有基本翻译模式与MRIF模式,IOMMU协议中建议MSI Cache中仅缓存基本翻译模式的PTE,具体实现采用了这一做法.故从Cache中读取的MSI PTE都为基本翻译模式.在图11中第3个时钟周期,msi_basic_mode拉高,表明该PTE为基本翻译模式.

4.5　命中率评估

IOMMU中的命中率是评估虚拟化性能的重要指标,如若虚拟化后命中率过低则将会降低外设访问的速率,从而影响性能.本文的命中率根据公式:命中率=命中次数/总访问次数计算得出.

由于初始阶段IOATC中缓存为空,此时命中率较低.经过大量的访问后,得益于PLRU算法以及本文中的缓存策略,IOATC的命中率有了较大提升,并且维持在较高水平.在命中率稳定之后,截取任意的12次访问,仅有1次未命中,计算出通过PLRU算法优化后的IOATC命中率为91%左右.

4.6　综合结果

IOMMU在计算机体系架构中处于I/O外设与系统互联之间,通常不会做单独的芯片产品,往往是处于SOC架构中或者以IP的形式输出,无论是SOC或IP,都需要经过FPGA综合与仿真验证.使用FPGA对IOMMU中的TCU进行的综合报告见表1所示.

表1　TCU综合报告

Tab.1　TCU synthesis report

Family	Cyclone IV E
Total logic elements	7 656
Total combinational functions	5 703
Dedicated registers	4 160

可以看出,作为IOMMU主要的控制与进行PTW的模块,TCU中的组合逻辑单元使用为5 703,寄存器资源使用为4 160.寄存器资源较多是因为TCU内部存在上下文的缓存寄存器、地址转换请求缓存buffer,以及三种队列的缓存buffer,整体电路面积适中.

通过图8~图10的分析,表2对普通的DMA访问(即通过PTW访问)与通过IOMMU的I/O访问进行了对比,普通DMA访问所耗费的资源通过开源DMA的IP仿真综合获得.相比普通DMA访问,I/O设备通过IOMMU进行内存访问的性能得到极大改善,仅需3~4个时钟周期即可实现访问.虽然造成了寄存器资源的消耗且所消耗的硬件资源相较于一般DMA访问增加了一倍,但其访问速度实现了数倍甚至数十倍的提升,故可认为硬件资源消耗是在可接受范围内的.最重要的是IOMMU实现了I/O设备与系统内存之间的隔离,提升了硬件资源的利用率.

表2 通过IOMMU访问性能比较

Tab.2 Performance comparison through IOMMU access

I/O访问方式	访问时间	寄存器资源	安全性
普通DMA访问	数十~数百个周期	4 160个	一般
IOMMU访问	3~4个周期	8 520个	较好

5 结论

在计算机体系中,随着计算性能的不断提升,I/O外设与虚拟机之间的数据流变得十分庞大且复杂.对外设的I/O事务进行虚拟化操作有助于实现虚拟机之间的隔离与安全,同时也会大幅提升物理资源利用率,从而提升计算机整体的性能.

由于目前暂未有RISC-V架构的I/O硬件辅助虚拟化实现,本文首次提出一种基于RISC-V架构的IOMMU实现方法.仿真结果表明,本文设计的IOMMU,IOATC命中率高达91%,极大地提高了I/O设备的DMA访问速度.占用资源较少,将来可以作为IP核,集成到RISC-V架构的处理器中.

参考文献

[1] 种丹丹. 基于RISC-V的开源芯片生态发展现状及未来机遇[J]. 中国集成电路,2021,30(8):25-30.
CHONG D D. Current status and future opportunities of open-source chip ecosystem based on RISC-V[J]. China Integrated Circuit,2021,30(8):25-30.(in Chinese)

[2] MORABITO R, COZZOLINO V, DING A Y, et al. Consolidate IoT edge computing with lightweight virtualization[J]. IEEE Network,2018,32(1):102-111.

[3] ZHANG W, LI Y D, LIU Y. A survey of direct interrupt delivery techniques in I/O virtualization[C]//2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS). Beijing,China. IEEE,2017:169-172.

[4] SCHULZ B, ANNIGHÖFER B. Evaluation of adaptive partitioning and real-time capability for virtualization with xen hypervisor[J]. IEEE Transactions on Aerospace and Electronic Systems,2022,58(1):206-217.

[5] SANG F L, LACOMBE É, NICOMETTE V, et al. Exploiting an I/OMMU vulnerability[C]//2010 5th International Conference on Malicious and Unwanted Software. Nancy,France. IEEE,2010:7-14.

[6] CASALE G, KRAFT S, KRISHNAMURTHY D. A model of storage I/O performance interference in virtualized systems[C]//2011 31st International Conference on Distributed Computing Systems Workshops. Minneapolis, MN, USA. IEEE, 2011:34-39.

[7] BABU S A, HAREESH M J, MARTIN J P, et al. System performance evaluation of Para virtualization, container virtualization, and full virtualization using Xen, OpenVZ, and XenServer [C]//2014 Fourth International Conference on Advances in Computing and Communications. Cochin, India. IEEE,2014:247-250.

[8] ARMSTRONG D, DJEMAME K. Performance issues in clouds: an evaluation of virtual image propagation and I/O paravirtualization[J]. The Computer Journal, 2011, 54(6):836-849.

[9] PITAEV N, FALKNER M, LEIVADEASY A, et al. Multi-VNF performance characterization for virtualized network functions[C]//2017 IEEE Conference on Network Softwarization (NetSoft). Bologna,Italy. IEEE,2017:1-5.

[10] MUENCH D, ISFORT O, MUELLER K, et al. Hardware-based I/O virtualization for mixed criticality real-time systems using PCIe SR-IOV[C]//2013 IEEE 16th International Conference on Computational Science and Engineering. Sydney, NSW, Australia. IEEE,2013:706-713.

[11] ĐORĐEVIĆ B, TIMČENKO V, KRALJEVIĆ N, et al. File system performance comparison in full hardware virtualization with ESXi and Xen hypervisors[C]//2019 18th International Symposium INFOTEH-JAHORINA (INFOTEH). East Sarajevo, Bosnia and Herzegovina. IEEE,2019:1-5.

[12] ERULANOVA A, YESSENBEKOVA G, ZHANYSBAYEVA K, et al. Hardware and software support of technological processes virtualization[C]//2020 7th International Conference on Electrical and Electronics Engineering (ICEEE). Antalya,Turkey. IEEE, 2020:333-337.