计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2022,31(9):31−38 [doi: 10.15888/j.cnki.csa.008842]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于轻量化远程过程调用的 RISC-V 调试协议栈方案①

万瑞罡, 朱焕杰

(芯来科技 (武汉) 有限公司 基础软件部, 武汉 430073)
通信作者: 万瑞罡, E-mail: h@iloli.bid

摘　要: 近年来, 随着 RISC-V 架构以其独特的开源、精简、模块化等优势在工业界快速铺开, 市场中涌现出大量基于 RISC-V 架构的处理器 IP 核及基于 RISC-V 架构设计的片上系统 (system on chip). 现有调试器作为 RISC-V 软件开发过程中的一个重要部件, 存在性能低、部署成本高以及二次开发难度大等问题, 难以应对现今井喷发展的 RISC-V 架构芯片的 RTL 设计与验证、程序开发与调试、量产批量编程的需求. 为了解决这些问题, 本文提出一种全新的、开源的、模块化基于轻量级远程过程调用实现互操作的 RISC-V 调试协议栈方案—Morpheus. 实验及分析结果表明, 该调试方案能够有效提高调试性能, 降低部署成本和二次开发难度.

关键词: RISC-V; 在板调试; 调试器; 调试协议栈

引用格式: 万瑞罡,朱焕杰.基于轻量化远程过程调用的 RISC-V 调试协议栈方案.计算机系统应用,2022,31(9):31–38. http://www.c-s-a.org.cn/1003-3254/8842.html

Implementation of RISC-V Debugger Protocol Stack Based on Lightweight Remote Procedure Call

WAN Rui-Gang, ZHU Huan-Jie

(Basic Software Department, Nuclei System Technology (Wuhan) Co. Ltd., Wuhan 430073, China)

Abstract: In recent years, as RISC-V architecture spreads rapidly in the industry due to its advantages of open source, concision, and modularization, massive processor IP cores and system on chip (SoC) based on the RISC-V architecture have emerged in the market. The existing debuggers serve as an important tool in developing RISC-V software, but they face low performance, high deployment cost, and high difficulty in secondary development and struggle in meeting the needs of RTL design and verification, software development and debugging, and mass production/batch programming of RISC-V architecture-based chips. To solve these problems, this study proposes a new, open-source, and modularized RISC-V debugger protocol stack design scheme based on the lightweight remote procedure call—Morpheus. Experiments and analysis results have shown that this debugger protocol stack can effectively reduce the deployment cost and the difficulty of secondary development and improves the debugging performance.

Key words: RISC-V; on-board debugging; debugger; debugger protocol stack

1　引言

现代处理器和微控制器中一般拥有一组称为调试模块 (debug module, DM) 的硬件电路, 该电路集成在处理器内部, 为开发者提供编程、调试、(部分) 异常处理、性能记录、追踪等功能. 这些功能既可以帮助软件工程师对代码进行除错、性能分析与调优; 又在事实上成为了在硬件量产工序时, 自动测试设备 (auto test equipment) 对系统进行各项测试、对芯片内的存储器进行编程的首选接口[1]. 因此, 调试模块已经成为现代片上系统的一个不可或缺的组成部分. RISC-V 的

① 本文由“RISC-V 技术及生态”专题特约编辑武延军研究员、宋威副研究员、张科高级工程师以及邢明杰高级工程师推荐.
收稿时间: 2022-03-29; 修改时间: 2022-04-25; 采用时间: 2022-05-16; csa 在线出版时间: 2022-07-22

模块化的特性推动了芯片行业自动化设计流程的变革, 同时也对调试模块的设计思路带来了全新的挑战.

在 RISC-V 调试规范中, 完整可调试的硬件系统包含待测系统 (device under-test) 和调试系统, 调试系统由调试适配器 (debug transport hardware) 和调试翻译器 (debug translator) 等组件组成[2], 通过运行调试协议栈完成调试功能. 目前被广泛使用的基于 OpenOCD 的调试协议栈在主机端实现了几乎全部调试逻辑, 而将调试适配器实现为一个简单协议转换器. 这种紧耦合、无内聚的设计理念不仅难以维护, 不利于二次开发, 并且导致调试系统和待测系统间信息的传输速度受制于主机与调试适配器之间较高的 I/O 延迟, 进而导致数据传输效率的低下.

本文提出了一种全新的开源调试协议栈设计. 我们将调试翻译器中的调试前端、RISC-V 调试前端、待测系统驱动、传输后端等组件分离, 在组件间使用标准 API 接口和轻量级的远程过程调用 (remote procedure call, RPC) 框架建立异步的数据通路. 这样的设计允许部分对 I/O 延迟有要求的功能由主机端卸载 (offload) 至调试适配器上, 提高数据传输效率. 组件的模块化设计和合理的层次结构使得整个系统代码简洁, 维护方便, 易于二次开发.

本文组织结构如下: 第 2 节介绍相关的背景知识及目前常用的调试协议栈与调试系统存在的问题; 第 3 节介绍 Morpheus 调试协议栈的基本架构设计思路与解决现有问题的方法; 第 4 节介绍 Morpheus 调试协议栈部分操作的具体实现流程; 第 5 节对本文做出总结.

2 研究背景

2.1 调试协议概述

交互式调试 (interactive debugging) 过程中对大批量高效率传输数据的需求, 几乎全部是对目标系统总线的访问. 总线读写请求的对象包括目标系统的随机访问存储器 (random access memory, RAM)、内存映射输入输出 (memory mapped input/output, MMIO) 寄存器与非易失性存储器 (non-volatile memory, NVM). 由于调试或量产时的编程操作需要对存储器进行顺序、大块数据的读取或写入, 会导致大量对 RAM 和 NVM 的读写请求.

根据目前的 RISC-V 调试规范, 对待测系统的总线访问请求可以使用 3 种方式实现[2]:

(1) 抽象命令 (abstract command)

(2) 程序缓存 (program buffer)

(3) 系统总线访问 (system bus access)

规范强制要求待测系统中的 DM 至少支持上述 3 种方式中的一种. 但无论选择实现其中哪一种, 数据传输均基于调试传输模块 (debug transport module, DTM) 所建立的数据通路.

图 1 展示了 RISC-V 调试系统的整体结构, 其中 RISC-V platform 是待测设备. RISC-V 调试规范所定义的 JTAG-DTM 模块对 DM 的交互依靠于 JTAG DR 寄存器 DMI (debug module interface)[3] 接口. 在通常的实现中, DTM 与 DM 之间使用不同的时钟域[4], 而 DMI 寄存器承担了在两个时钟域中时钟跨越的功能. 这种设计引入的跨时钟域握手机制将导致 DTM 阻塞, 即使在 DM 完全不阻塞的情况下. 任何对 DM 的访问请求, 一般分为如下几步.

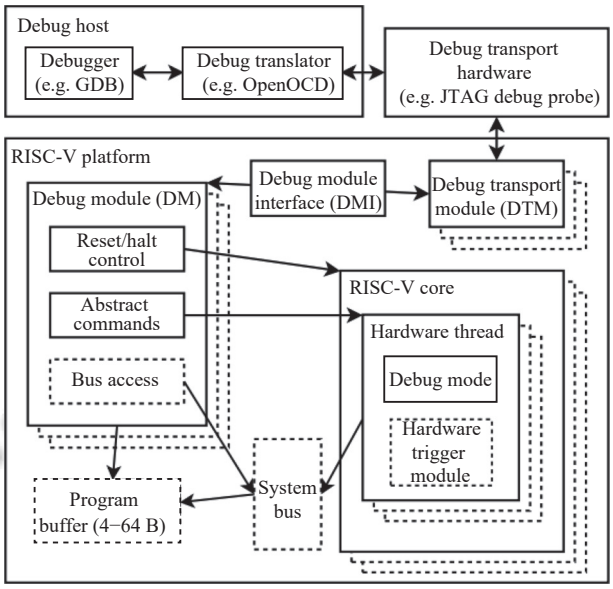

图 1 RISC-V 调试系统整体结构框图

(1) 填充 DMI 寄存器的 DM 地址、数据与操作码;

(2) 等待 DTM 模块中 DTMCS (DTM control and status) 寄存器中要求的 run-test-idle 周期;

(3) 回读 DMI 寄存器, 检查状态位. 若 DMI 忙, 则重复第 2 步, 等待操作结束为止. 若 DMI 错误, 则进入错误处理流程.

同样, 由于 DM 中不存在类似 ARM 调试系统的总线访问缓存机制[5], 无论使用 3 种总线访问方式中的哪一种, 不仅需要等待 DMI 就绪, 也需要查询 DM 中

的对应寄存器的状态位, 确保 DM 对请求的正确执行, 并确保在总线通路空闲的时候, 再进行下一笔数据的传输.

2.2 现存社区维护的工具存在的问题

目前, 开源社区存在一定数量的调试协议栈实现, 例如 OpenOCD、RV-Link 等. OpenOCD 由 RISC-V 基金会维护, 且对 RISC-V 架构处理器的调试功能支持最全面, 是现今应用最广泛的开源调试协议栈.

OpenOCD 将调试协议栈需要实现的所有功能, 包括对 DMI 的访问, 以及对 DM 各寄存器的操作, 均在主机侧完成[6]. 根据 RISC-V 调试规范, 每一笔对总线的访问, 长度最多与 RISC-V 处理器字长 (XLEN) 相等. 而每一笔对总线的操作, 均需要执行发出请求–查忙–完成的 3 步流程[2]. 其中查忙流程导致主机无法简单的向无处理能力的调试适配器, 如常与 OpenOCD 搭配的 FTDI 公司的多协议同步串行引擎 (multi protocol synchronous serial engine, MPSSE) 发送一块含有多笔请求的命令块实现, 而必须逐步的将命令分解为不同的、细碎的读写请求再发送给调试适配器, 并依赖每一步调试适配器返回的结果执行下一步请求[7]. 因此, 调试器大批量对存储器的读写请求, 很大程度上被主机到调试适配器间的往返延迟所限制.

例如, 使用 USB 2.0 全速协议连接主机与调试适配器, 假设操作系统和主机互联系统不引入任何延迟. 每个 USB IN 令牌包为 59-bit, 每个握手包为 44-bit[8]. 倘若 USB 主机接口 (host controller interface) 不间断的发送 IN 令牌包, 则最高轮询速率为:

$$12\ \text{Mb/s} \times (59+44)^{-1} \approx 116\ \text{kHz} \quad (1)$$

实际上, 由于 HCI 和操作系统实际引入的损耗, 实测基于 USB 2.0 全速主机-适配器通讯接口, 由 OpenOCD 配合基于 FTDI 公司的 MPSSE 的调试适配器, 每秒钟可被目标执行的命令速率约为 14 kHz, 实际传输速率约为 57 KB/s. 由于系统性能瓶颈存在于主机-调试适配器间较高的通讯往返延迟与较大的查询开销. 即使在调试适配器对目标系统的 DTM 接口上使用较高的 JTAG 传输频率, 也无法对传输性能和传输效率做进一步的提升, 体现在逻辑分析仪捕获的 DTM 接口波形上即为较长的请求间隔, 如图 2 所示.

RV-Link 项目与上述项目不同, 它将整个调试协议栈卸载到调试适配器上, 对主机通过 GDB server 协议通讯, 其大幅度降低了延迟, 提高了系统的传输效率, 达到了近似下文所述商业工具 J-Link 的性能[9]. 但它存在以下的问题:

(1) 调试适配器本身资源有限, 难于存储不同的设备支持表、外设支持驱动等必要的信息, 使得支持设备的数量较少且二次开发门槛较高.

(2) 使用 GDB server 协议与主机通讯, 难以承载其他的调试需求, 如裸 JTAG 通讯等.

(3) 无完善的固件升级方案, 难于解决使用固件升级的方式对系统增加功能, 或是修复问题.

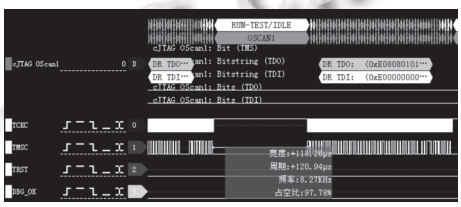

图 2 基于 OpenOCD 调试器总线访问操作时序图

2.3 现存商业化工具存在的问题

SEGGER 公司推出的 J-Link V11 调试器提供了对 RISC-V 架构的支持. 与 OpenOCD 所不同的是, J-Link 将调试协议栈中绝大多数功能卸载到调试适配器上运行[10], 有效缓解了主机-调试适配器间通讯延迟高造成的开销, 从而大幅提升了调试器的编程速率.

但是, J-Link 如传统的商业化工具一样, 存在以下几个较为明显的问题, 使其应用难于铺开:

(1) 大量功能需要在调试适配器上直接运行, 对调试适配器本身的处理性能有一定要求.

(2) J-Link 并非开源系统, ASIC 设计厂商和开发者难以改进 J-Link 内置的设备支持或增加其他芯片的支持.

(3) J-Link 对第三方的开放接口十分有限, 且经常变化. 第三方开发者如果基于 J-Link 开发了专有的目标系统驱动程序, 如 Flash 存储器编程驱动等, 后续升级 J-Link 软件版本的成本非常高.

(4) 截至 2022 年 3 月, J-Link 对多核 RISC-V 系统的支持仍然不佳, 对部分处理器 IP 核与专有 DTM 协议的兼容性较差.

(5) J-Link 的组件耦合紧密, 单独剥离使用 (如在产线上用作离线编程器) 较为困难.

(6) J-Link 购置成本高, 难以大规模推广.

3 Morpheus 调试协议栈

针对以上现有工具的不足, 本文提出一种基于异

步 I/O、模块化、轻量级 RPC 的调试协议栈的设计方案. 该方案充分结合了各现有调试方案的优势, 并为未来的扩展留下了足够的空间.

3.1 设计理念

在缓存设置合理的情况下, 异步传输的吞吐率并不会因为延迟增加而产生明显下降. 同步传输由于需要在每一步状态改变之后等待对面设备响应, 实际吞吐率和全链路的延迟呈现负相关. 因此, 在设计新调试器方案时, 为了解决系统的传输瓶颈, 应当将同步传输局限于调试适配器与待测系统之间, 而在主机-调试适配器间使用异步传输, 以求最大化的降低延迟.

RV-Link 式的将所有功能均卸载到调试适配器上, 仅对主机暴露最终调试接口的设计范式, 也能够满足上述要求. 但调试适配器本身处理性能较主机弱、存储容量较主机小. 这些缺点无疑限制了它的扩展性与它能够达到的性能上限. 因此, 对调试协议栈整体架构的设计, 既不能将所有模块均放置于主机端, 这意味着较高的延迟导致的较低传输效率; 也不应当将所有模块均放置于调试适配器端, 这意味着较低的处理能力导致的受限的扩展性和有限的功能.

综合考虑, 意味着 Morpheus 必须对软件模块进行拆分, 摒弃 OpenOCD 等软件传统的紧耦合式大状态机结构, 并且让拆分出的小状态机能够在不同架构的硬件上表现一致, 同时, 尽量将必须同步通信的模块设计在同一硬件实体上运行, 并依靠 RPC 机制实现其他模块间的高效跨设备异步传输. 上述模块拆分能力同时让接口的标准化成为现实. 对 Morpheus 系统中任何功能的厂商定制都可以通过实现一组标准的软件接口来完成, 无需厂商自行维护整个调试器的一个单独分支. 这一设计有效避免了现有开源调试器软件碎片化、各公司提供的版本和功能不一致的问题, 也为低成本的离线编程器提供了实现的基础.

3.2 系统结构

为了降低模块间耦合, 提高传输效率, 使得整个调试系统灵活、可持续扩展, 同时又不引入模块间调用的巨大开销, 依据第 3.1 节中的设计理念, 借鉴各个现存调试系统的优势和成功经验, 调试协议栈整体被分拆为如下组件, 如图 3 所示. 其中, 图中标示为 PC 的组件可运行于主机系统上, 标示为 HW 的组件可运行于调试适配器上.

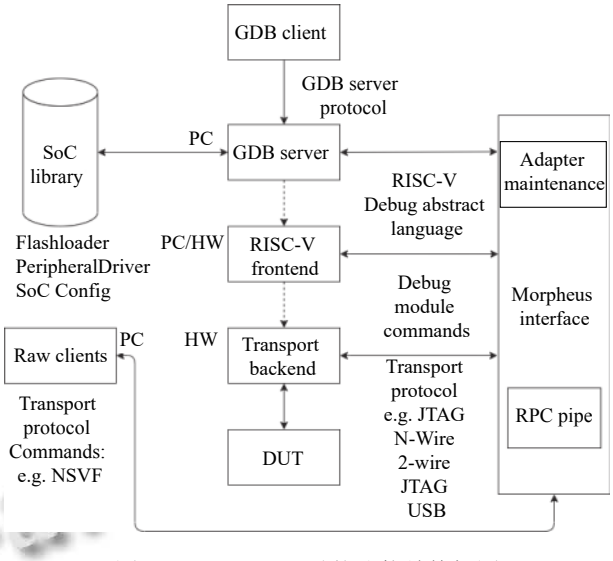

图 3 Morpheus 调试协议栈结构框图

(1) SoC library: 提供目标系统的配置信息、驱动程序、地址映射表 (memory-map) 等必要信息.

(2) GDB server: 直接提供对 GDB 或其他 GDB 协议客户端 (如 Eclipse) 的调试服务功能, 并结合 SoC library 提供的目标系统信息, 产生合适的调试抽象命令 RVDAL (RISC-V debug abstract language) 提供给 RISC-V frontend.

(3) RISC-V frontend: 根据待测系统的调试协议版本, 将 RVDAL 命令转换成相应的 DM 控制命令序列.

(4) Transport backend: 将对应的 DM 控制命令序列转换成 DTM 所需要的时序, 并连接 DTM, 提供调试的物理通路. 如 JTAG/cJTAG[11]/Nuclei-Wire/USB[8] 等接口.

(5) Morpheus interface: 提供各个组件之间的互操作与异步 I/O 接口的定义和参考实现. 实现轻量化的 RPC 协议[12] 与可选的调试适配器维护 (如固件升级) 等功能. 未来将支持可选的对外裸接口功能. 使得其他的调试工具, 如 FPGA 配置工具也能复用 transport backend 提供的数据通路, 达到仅需单调试器与调试线路, 可同时调试不同种类型的待测系统的目的, 如 Xilinx Zynq 异构集成系统.

此外, Morpheus 使用 C 语言 (符合 C89 标准) 开发, 各基础组件均能跨平台工作, 耦合度低且不依赖特定操作系统. 因此, 其中部分组件可以独立运作来实现某些定制功能. 例如 SoC library、RISC-V frontend、transport backend、Morpheus interface 单独提取出来即

可作为一个基本的离线编程器使用.

3.3 轻量级 RPC 的实现

Morpheus 各组件之间使用 Morpheus interface 所提供的接口进行调用. 其中 RPC 协议基于简单的 TLV (type-length-value)[13] 帧格式, 运行在 RPC 管道上. RPC 管道是一个保证按序交付的帧式管道的虚拟实体. 目前有一个基于 USB 虚拟串口 (CDC-ACM 协议)[14] 与一个共享内存的 RPC 管道参考实现, 未来可扩展基于以太网或其他协议的传输管道, 达到灵活部署的目的. 其中, 基于共享内存的管道用于在单实例中函数间的调用场景. RPC 协议的帧格式如表 1 所示.

表 1 RPC 协议帧格式

序号	名称	长度	备注
1	帧类型	8-bit	指示帧类型
2	调用实体Id	24-bit	RPC函数编号
3	载荷1	*n*-bit	载荷1 (可选)
n	载荷*n*	*n*-bit	载荷*n* (可选)

表 1 中, 调用方的载荷, 作为被调用方的输入参数使用. 若被调用方存在返回值, 则在应答帧中填入函数返回值的载荷, 使得载荷返回调用方. 载荷格式如表 2.

表 2 载荷格式

序号	名称	长度	备注
1	载荷Id	8-bit	指示载荷名称
2	载荷类型Type	8-bit	指示载荷类型
3	载荷长度 Len	16-bit	载荷长度(可选)
4	载荷数据Value	*n*-byte	载荷数据

该轻量化的 RPC 协议实现了主机与调试适配器之间灵活的调用关系, 并具有多样化的载荷类型扩展能力, 使得使用统一的编程模型, 对主机程序的开发与调试适配器固件的开发成为可能.

3.4 验证方法与哑传输后端

为了在无实际硬件的情况下 (如回归仿真环境) 对调试协议栈的正确性进行快速测试, 并向 EDA 环境提供调试接口. Morpheus 调试协议栈提供了两种特殊的传输后端, 分别为哑传输后端与 Verilog 过程接口 (Verilog procedural interface, VPI) 传输后端.

其中哑传输后端实现了一个简单的基于 RISC-V 调试规范 0.13 版本的 DM 后端, 能够简单的对 RISC-V frontend 进行覆盖测试. 同时, 为了能兼容存量 MPSSE 调试适配器, Morpheus 提供了基于 MPSSE 的传输后端, 但此时所有 Morpheus 组件均运行于主机, 没有降低 I/O 延迟所带来的传输效率提升.

VPI 传输后端提供了类似 OpenOCD 的 JTAG-VPI 接口[15], 能与 EDA 仿真环境互通, 操作 EDA 环境中的 RISC-V 核心的 DM 或 DTM, 提供在 EDA 仿真环境中对处理器调试模块的访问与控制能力.

3.5 系统初始化流程

由于 Morpheus 调试系统使用同一套代码库编译出可跨平台运行的各组件, 组件间使用 RPC 方式互操作并建立数据通路. 因此启动时必须保证主机端与适配器端的固件版本完全一致, 否则 RPC 无法正常工作. 因此, 在进行初始化流程时必须加入版本确认过程, 确保适配器与主机端程序版本完全一致, 如图 4 所示.

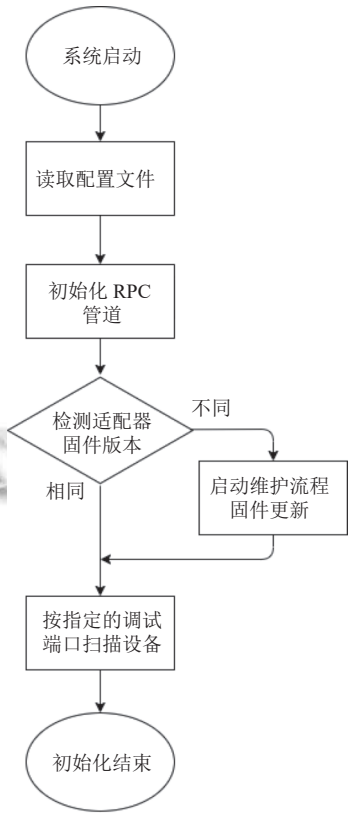

图 4 Morpheus 调试协议栈非哑传输模式下启动流程图

4 性能优化的具体实现

4.1 总线访问流程

在处理总线访问请求时, 由位于主机的 GDB server 负责接收或提供数据, 以 RPC 的方式对位于调试适配器固件中的 RISC-V frontend 组件进行调用, RISC-V

frontend 根据命令和总线访问方式, 进行可选的虚拟地址-物理地址转换[16], 产生对应的 DM 访问序列, 由之前已选择好的 DTM 传输接口所对应的后端发出. 由于 RISC-V frontend 组件与 transport backend 组件均可运行在调试适配器上, 且调试适配器使用微控制器实现, 故相比于主机直接控制的方式能提供较低的输入输出延迟, 由此达到提高传输效率, 降低数据传输时间的目的, 其流程如图 5 所示.

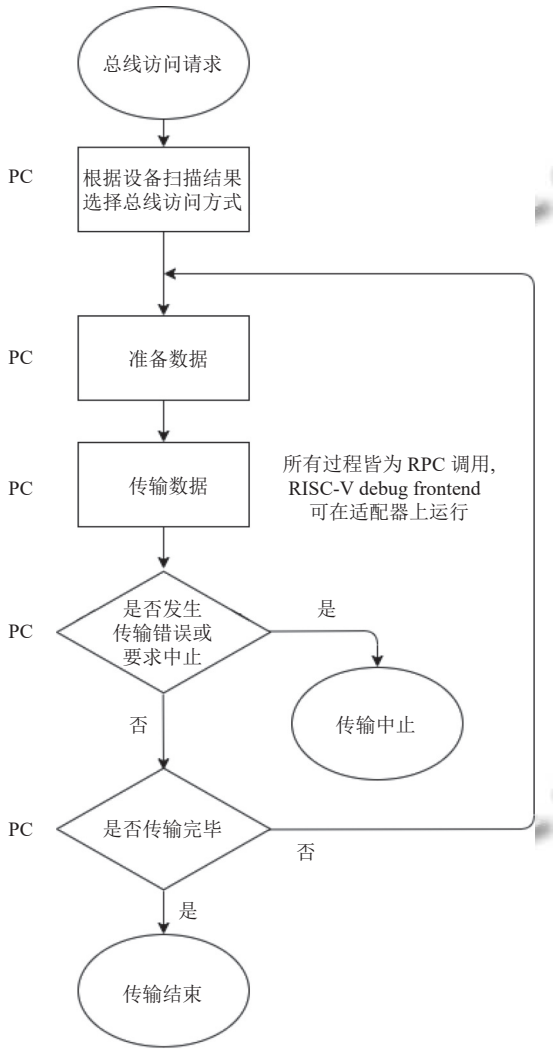

图 5 总线访问流程图

4.2 Flash 存储器编程流程

Flash 存储器具有电可擦除, 无需后备电源保持数据、可重复编程、存储密度高、低功耗、低成本等特点, 是一类应用广泛的 NVM 存储器[17]. 但相比 RAM 而言, Flash 存储器需要特殊的写入时序进行编程, 编程前必须擦除原有内容[18]; 不同 Flash 亦具有不同的编程模型, 互不兼容, 故无法用简单的总线访问流程解决对 Flash 存储器的编程问题. 因此我们引入一个中间层, 即 Flash 驱动机制[19], 解决对不同系统、不同型号 Flash 的编程问题.

由于主机侧对待测系统通信延迟高, Morpheus 调试协议栈为提高效率和简化设计, 不支持类似 OpenOCD 将 Flash 驱动直接实现在主机侧的 Flash 驱动方案, 强制使用基于 Flashloader 的 Flash 驱动方案.

Flashloader 是一组简单的、平台相关的、可被重定位的运行于待测系统上的程序[20], 存储于 SoC library 中. 若用户请求对某块地址空间的写入, GDB server 将对比 SoC library 中的系统配置, 确定用户是否请求对 Flash 区间进行编程. 若用户访问的地址空间不属于 Flash 区间, 则系统转入总线访问流程.

若用户访问的地址空间属于 Flash 区间, 则 GDB 将对 GDB server 请求目标系统地址的映射表[21], 然后, GDB server 将根据系统配置, 在待测系统上寻找一块合适的可被执行的内存块, 申请足够大小的内存, 该内存块被称为工作区. 申请到工作区后, 将工作区分为程序区与数据缓冲区, 将 Flashloader 写入程序区. 再使用 Flashloader 与 GDB server 已约定好的 ABI, 运行 Flashloader 的初始化过程与擦除过程, 在确认初始化和擦除过程成功完成后, 进入编程循环: GDB server 将对申请到的数据缓冲区写入部分数据, 随后运行 Flashloader 的写入流程, 不断重复编程循环, 直到所有数据均被正确编程到 Flash 中. 在 Flash 编程完毕后, GDB server 将运行 Flashloader 的反初始化流程, 确保系统被复原. 然后销毁工作区, 完成整个编程流程, 流程如图 6 所示.

使用 Flashloader 的 Flash 编程方式, 相较于原本直接由主机或调试适配器上运行的固件对 Flash 控制器的寄存器直接操作的方式有效降低了往返延迟, 提高了编程速率, 并充分利用了待测系统的运行性能, 且降低了 Morpheus 本身的复杂度: Morpheus 本身不需要内置任何 Flash 控制器的驱动程序, 仅需要按照标准接口增加 Flashloader, 就能快速便捷的支持新的 Flash 控制器. 在用户软件开发能力不足时, 无需重新编译整个项目, 仅需简单增加 SoC library 配置即可支持, 可大大缩短开发周期; Morpheus 维护团队则仅需维护一份平台无关的代码, 即可实现对不同结构、不同外设驱动的 SoC 的支持, 有效减轻了软件测试压力.

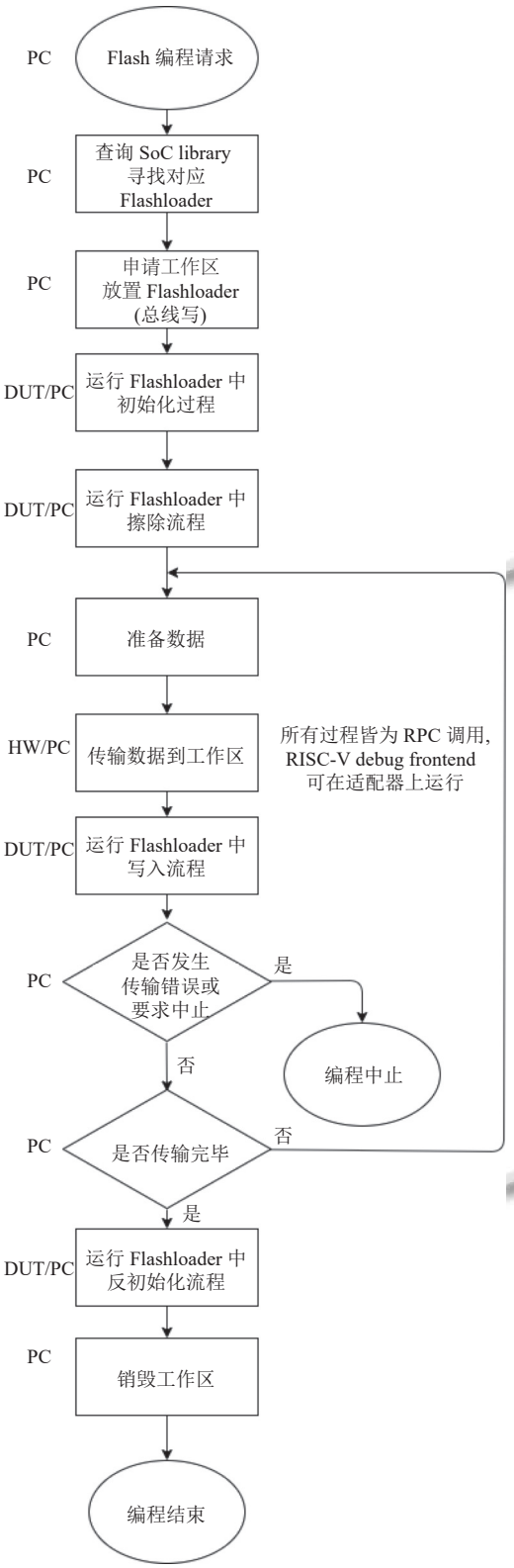

图 6 Flash 编程流程图

4.3 性能测试结果

在基于芯来科技 Bumblebee 内核的兆易创新 GD32VF103 芯片平台, JTAG TCK 频率设定为 4 MHz. OpenOCD 作为调试协议栈, FTDI 作为调试适配器的工况下, 对 RAM 的编程速率约为 57 KB/s, 对 Flash 的编程速率约为 10 KB/s.

在完全相同的 GD32VF103 硬件平台上, JTAG-TCK 频率设定仍为 4 MHz. Morpheus 作为调试协议栈, GD32VF103+Morpheus 固件作为调试适配器的工况下, 对 RAM 的编程速率约为 141 KB/s, 对 Flash 的编程速率约为 16 KB/s.

由于使用了异步 I/O 机制, 并应用了 RPC 架构, 将大量耗时的 I/O 操作由主机卸载到调试适配器上, Morpheus 相对 OpenOCD 的 RAM 编程、Flash 编程性能提升分别为+140%、+60%, 体现在 DTM 接口波形上, 即为较短的请求间隔. 逻辑分析仪抓取的 DTM 接口波形如图 7 所示, 可见相比图 2, 延迟由 118 μs 缩短为 2.3 μs.

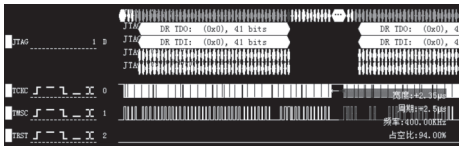

图 7 Morpheus 调试协议栈总线访问时序图

5 总结与展望

RISC-V 架构以它独特的开放、开源且模块化的优势得到了产业及学术界的青睐. 为了推进 RISC-V 生态的铺开, 开发工具与软件生态的建设十分重要. 本文在分析各现有方案不足的基础上, 提出并实现了一套开源、模块化并基于轻量化 RPC 实现互操作的全新的 RISC-V 调试协议栈——Morpheus. 该方案解决了目前现有调试方案性能低下、拥有成本高昂的问题, 成功的降低了调试器的开发、维护、测试成本.

目前, Morpheus 尚未实现同类商业调试系统提供的全部的功能. 在未来, Morpheus 需要走的路还很长. 但基于灵活的 RPC 和异步 I/O 机制, Morpheus 的开发和维护, 相对传统紧耦合的调试器设计, 存在巨大的优势. 基于这一先进架构, Morpheus 开发团队将逐步实现更多的功能, 如对 Semihosting、Flash 软断点的支持以及对虚拟内存/对称多处理机系统的支持等, 助力 RISC-V 架构在行业内进一步的普及, 为 RISC-V 开发工具与基础软件生态建设作出贡献.

参考文献

1 Ehrenberg H. Reconfigurable tester hardware extends JTAG/boundary scan applications while simplifying ATE setup. Proceedings of 2007 IEEE Autotestcon. Baltimore: IEEE, 2007. 712–717. [doi: 10.1109/AUTEST.2007.4374289]

2 Newsome T, Wachs M. RISC-V external debug support version 0.13.2. https://riscv.org/wp-content/uploads/2019/03/riscv-debug-release.pdf. (2019-03-22).

3 IEEE. IEEE Std 1149.1-2013 IEEE standard for test access port and boundary-scan architecture. New York: IEEE, 2013, [doi: 10.1109/IEEESTD.2013.6515989]

4 SI-RISCV. Hummingbird E203 opensource processor core. https://github.com/SI-RISCV/e200_opensource. (2019-09-24).

5 ARM Developer. ARM debug interface architecture specification ADIv6.0. https://developer.arm.com/documentation/ihi0074/latest/. [2022-02-25].

6 OpenOCD. OpenOCD developer's guide. https://openocd.org/doc-release/doxygen/index.html. [2022-03-29].

7 FTDI. Command processor for MPSSE and MCU host bus emulation modes. https://www.ftdichip.com/Documents/AppNotes/AN_108_Command_Processor_for_MPSSE_and_MCU_Host_Bus_Emulation_Modes.pdf. (2011-09-09).

8 USB-IF. USB 2.0 specification. https://www.usb.org/document-library/usb-20-specification. (2021-08-10).

9 Zoomdy. RV-link repository: Gitee. https://gitee.com/zoomdy/RV-LINK. (2020-04-25).

10 SEGGER. J-Link/J-Trace user guide. https://www.segger.com/downloads/jlink/UM08001. [2022-05-18].

11 IEEE. IEEE Std 1149.7-2009 IEEE standard for reduced-pin and enhanced-functionality test access port and boundary-scan architecture. New York: IEEE, 2010. [doi: 10.1109/IEEESTD.2010.5412866]

12 Bershad B, Anderson T, Lazowska E, *et al*. Lightweight remote procedure call. ACM SIGOPS Operating Systems Review, 1989, 23(5): 102–113. [doi: 10.1145/74851.74861]

13 Przygienda T. Reserved type, length and value (TLV) codepoints in intermediate system to intermediate system. RFC3359, 2002. 1–5.

14 USB. Class definitions for communication devices 1.2. https://www.usb.org/document-library/class-definitions-communication-devices-12. (2007-02-09).

15 Dawson C, Pattanam SK, Roberts D. The verilog procedural interface for the verilog hardware description language. Proceedings. IEEE International Verilog HDL Conference. Santa Clara: IEEE, 1996. 17–23. [doi: 10.1109/IVC.1996.496013]

16 Waterman A, Asanović K. The RISC-V instruction set manual, Volume II: Privileged architecture. https://riscv.org/wp-content/uploads/2019/08/riscv-privileged-20190608-1.pdf. (2019-06-08).

17 Hidaka H. Evolution of embedded flash memory technology for MCU. Proceedings of 2011 IEEE International Conference on IC Design & Technology. Kaohsiung: IEEE, 2011. 1–4. [doi: 10.1109/ICICDT.2011.5783209]

18 关珊珊, 周洁敏. 基于 Xilinx FPGA 的 SPI Flash 控制器设计与验证. 电子器件, 2012, 35(2): 216–220. [doi: 10.3969/j.issn.1005-9490.2012.02.023]

19 Woodhouse D. Memory technology device (MTD) subsystem for Linux. http://www.linux-mtd.infradead.org/archive/index.html. (2005-03-12).

20 SEGGER. Open flashloader. https://wiki.segger.com/Open_Flashloader. [2022-05-09].

21 Ji JH, Woo G, Park HB, *et al*. Design and implementation of retargetable software debugger based on GDB. Proceedings of 2008 3rd International Conference on Convergence and Hybrid Information Technology. Busan: IEEE, 2008. 737–740.

(校对责编: 孙君艳)