计算机科学
COMPUTER SCIENCE

RVTDS:面向微处理器的追踪调试系统

高轩, 何港兴, 车文博, 扈啸

引用本文

高轩, 何港兴, 车文博, 扈啸. RVTDS:面向微处理器的追踪调试系统[J]. 计算机科学, 2023, 50(12): 66-74.

GAO Xuan, HE Gangxing, CHE Wenbo, HU Xiao. RVTDS:A Trace Debugging System for Microprocessor [J]. Computer Science, 2023, 50(12): 66-74.

相似文章推荐（请使用火狐或 IE 浏览器查看文章）

Similar articles recommended (Please use Firefox or IE to view the article)

程序调试中的树形结构演变可视化模型

Tree Structure Evaluation Visualization Model for Program Debugging

计算机科学, 2021, 48(5): 68-74. https://doi.org/10.11896/jsjkx.200100133

反向调试技术研究综述

Summary on Reverse Debugging Technology

计算机科学, 2021, 48(5): 9-15. https://doi.org/10.11896/jsjkx.200600152

一种基于变异分析的BPEL程序故障定位技术

Mutation Based Fault Localization Technique for BPEL Programs

计算机科学, 2021, 48(1): 301-307. https://doi.org/10.11896/jsjkx.200900051

基于日志可视化分析的微服务系统调试方法

Method of Microservice System Debugging Based on Log Visualization Analysis

计算机科学, 2019, 46(11): 145-155. https://doi.org/10.11896/jsjkx.181102210

面向混合并行计算系统编程环境的研究与实现

Research and Implementation of Parallel Programming Environment for Hybrid Parallel Computing System

计算机科学, 2010, 37(4): 143.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.230100030

RVTDS:面向微处理器的追踪调试系统

高 轩 何港兴 车文博 扈 啸
国防科技大学计算机学院 长沙 410073
(gaoxuan@alumni.nudt.edu.cn)

摘 要 软件调试是嵌入式系统开发中最具挑战性的难点之一。在进行高复杂性、高实时性系统调试时,单步-断点时间开销大,易破坏程序执行行为;采用串接机制的JTAG接口,在实现对处于工作状态的复杂多核处理器的并行访问时存在缺陷。片上追踪调试技术通过专用硬件非侵入地获取程序执行状态,有效解决了上述问题。现有的片上追踪调试技术相关研究以追踪完整信息为主,易产生大量无意义的数据;此外,也未考虑压缩后的数据在窄总线上的传输问题。文中设计并实现了一种基于RISC-V指令集的面向多核微处理器的非侵入式追踪调试系统RVTDS,通过复用RISC-V核内平台级别中断控制器,解决多核微处理器高速并行调试时的数据丢失问题;提出了面向片上总线的数据流追踪方案和基于指令位域匹配的控制流过滤机制以实现信息筛选,提供总线带宽统计功能;提出了基于差分编码的数据压缩方法,数据平均压缩率达82%以上;提出了一种数据打包方案以实现窄总线上的数据传输问题,每拍有效数据平均可容纳约1.5个路径信息。系统验证结果表明,RVTDS与传统片上追踪调试方法相比,追踪数据量小,可以灵活高效地完成复杂多核微处理器多种片内运行信息的采集、传输和存储。

关键词:片上追踪调试;非侵入;调试;RVTDS;多核微处理器
中图法分类号 TN492

RVTDS:A Trace Debugging System for Microprocessor

GAO Xuan,HE Gangxing,CHE Wenbo and HU Xiao
School of Computer Science,National University of Defense Technology,Changsha 410073,China

Abstract Software debugging is one of the most challenging factors in embedded system development. When debugging high-complexity,high real-time systems,single-step-breakpoint debugging time overhead is high,and tends to corrupt program execution behavior;the JTAG interface with serial-connection mechanism is flawed in achieving parallel access to complex multicore processors in operation. The on-chip tracing and debugging technology solves the problems of traditional debugging methods such as single-step-breakpoint and JTAG in debugging highly complex and real-time systems by non-intrusively obtaining program execution status through dedicated hardware. The existing on-chip tracing and debugging technologies mainly trace complete information,and generate a large amount of meaningless data,which easily causes path blocking or data loss,especially during concurrent debugging. In addition,the transmission of compressed data on narrow buses is not considered. A non-intrusive trace debugging system based on RISC-V for multicore microprocessors,RVTDS,is designed and implemented,which solves the data loss problem during high-speed parallel debugging of multicore microprocessors by reusing the platform-level interrupt controller within RISC-V cores. A data-flow tracing scheme for an on-chip bus and a control-flow filtering mechanism based on instruction bit domain matching are proposed to realize signal filtering and provide bus bandwidth statistics. A data compression method based on differential coding is proposed with an average data compression rate of more than 82%. A data packing scheme is proposed to realize the data transmission problem on a narrow bus with an average of about 1.5 path information per effective data beat. The system verification results show that RVTDS has a small amount of trace data compared with traditional debugging methods,and accomplishes the acquisition,transmission,and storage of multiple on-chip operation information of complex multicore microprocessors flexibly and efficiently.

Keywords On-chip tracing and debugging,Non-intrusive,Debug,RVTDS,Multi-core Microprocessor

到稿日期:2003-01-05 返修日期:2023-06-26
基金项目:国家重大科技专项(2017-V-0014-0066)
This work was supported by the National Science and Technology Major Project(2017-V-0014-0066).
通信作者:车文博(99179725@qq.com)

1　引言

嵌入式系统的高度复杂化与集成化，加剧了其调试调优难度。据统计，工程中 50% 的时间都在进行调试[1]，调试已成为嵌入式系统开发中最具挑战性的问题之一[2]。

传统调试以侵入式为主，通过控制程序运行状态实现系统调试，可观测范围较大。单步-断点调试是最常用的侵入式调试方式。开发人员在程序某处设置断点，程序运行至断点位置时暂停，再逐步追踪程序执行流程，即时比对输出结果，可获知程序每一步执行情况。单步-断点调试在小型程序调试时具备优异性能，但面向大规模程序调试，单步追踪已不可取，添加断点实现错误溯源也将耗费大量时间。实际工程中，部分错误仅在程序运行现场才能复现[3]，而侵入式调试可能破坏程序执行行为。例如，多核处理器并行调试时，单步-断点调试仅能实现对局部行为的中断操作，一个处理器核中断时，其他处理器核与独立工作的功能单元并未立即暂停执行，从而导致程序运行错误，甚至系统失控[4-5]。

JTAG(Joint Test Action Group)是面向可测试性设计提出的一种调试机制，该机制采用菊花链模式，多个处理器核的测试访问端口可串接到单一仿真器。芯片处于调试状态时，启动边界扫描单元，可配合单步-断点调试机制实现对其输入/输出信号的观察与控制。芯片处于工作状态时，允许 JTAG 非侵入式访问片内功能单元，然而，串接机制不利于 JTAG 对处于工作状态的复杂多核处理器实现并行访问[6]。

片上追踪调试技术采用专用硬件资源，非侵入实时记录程序执行信息，解决了单步-断点和 JTAG 等调试方法在多核处理器系统调试时的局限性问题。目前，片上追踪调试技术相关研究存在以下问题：1）未考虑高速系统调试时的追踪缓冲区溢出问题，停顿并将缓冲区数据转储至片外的方式严重阻碍了有效调试；2）未考虑多核微处理器系统并行追踪时的数据丢失问题，若来自不同核的多个追踪源同时访问一个追踪缓冲区，数据易丢失；3）忽略追踪信息筛选，将产生大量无意义数据；4）数据压缩时多采用 LZ 字典压缩算法，硬件开销较大；5）未考虑追踪信息在窄带宽总线上的转储过程。

本文的主要贡献如下：

1）针对问题 1 和问题 2，基于 RISC-V 指令集设计并实现面向微处理器的追踪调试系统 RVTDS，采用触发-中断-转储的工作机制，通过复用 RISC-V 核内中断仲裁机制控制多个追踪源的数据实时转储，同一时刻仅允许一个追踪源执行追踪信息转储。

2）针对问题 3，提出采用匹配-触发机制追踪指定数据流与指令段，提供总线带宽统计功能。

4）针对问题 4，提出一种基于差分编码的数据压缩方案，数据平均压缩率达 82%以上。

5）针对问题 5，提出一种数据打包方案，追踪信息转储时，每拍有效数据平均可容纳约 1.5 个路径信息。

本文第 2 章讨论片上追踪调试技术相关工作；第 3 章对 RVTDS 整体框架及追踪处理器进行说明；第 4 章介绍数据追踪采样单元；第 5 章介绍路径追踪采样单元；第 6 章对系统功能进行验证与分析；最后总结全文并展望未来。

2　相关工作

当前，关于片上追踪调试技术的相关研究主要从追踪信号过滤机制、追踪信息处理方法和数据转储结构这 3 个方面展开。

追踪信息按类型可分为控制流信息、时间信息、数据流信息和其他信息[5]。控制流信息直接反映程序运行时的实时决策问题，也被称为路径信息。Hsieh 等设计了一种嵌入式数字信号处理器调试结构[7]，通过记录完整 PC 值实现路径追踪，而实际工程要关注的路径信息有限，该方法将产生大量无意义的追踪结果。时间信息包含时间戳和时钟速率信息。数据流信息可以是内存地址或具体数值，研究集中在对系统总线的追踪[8-10]。Duan 等先后实现了对 AHB 总线和 AXI 总线的追踪[8-9]，这种记录完整总线数据流信息的方式将产生大量数据，造成通路拥塞。Ramirez 通过配置 Analysis，Reference和 Offset 这 3 个寄存器实现对总线数据的筛选，由 Analysis决定哪个输入信号与 Reference 和 Offset 进行比较[10]。这种过滤方法可产生多种组合，但在追踪无规律的数据通道时效果一般。其他信息是一类事件信息，如内部缓冲区溢出或 Cache 失效[11-13]等。

追踪信息处理过程分为两部分：一是按特定格式打包数据，添加时间戳等信息，增加追踪信息可分析性；二是压缩打包后的数据，基于有限片上资源，提供足够的追踪信息。霍夫曼编码是一种基于统计的数据压缩算法，具有压缩率高的特点，然而在芯片设计阶段，追踪信息不可预知，先验统计值难以获取。自适应霍夫曼编码解决了上述问题，但在硬件实现上，其存储成本与计算成本较高[14]。LZ 编码是一种高效便捷的通用压缩算法，基于字典实现，当前关于追踪信息处理的大部分研究采用 LZ 编码，但硬件开销较大[7,15-16]。差分编码是硬件实现简单、资源开销较小的一类压缩方式，路径信息存在相关性，差分编码可显著降低数据量，从而减少硬件资源消耗[16]。然而，上述研究均未考虑追踪信息在窄带宽总线上的转储过程。

将处理后的追踪信息存放至特定追踪缓冲区再转储至调试主机是较为主流的数据存储模式。Cao 等利用实时追踪基础部件替代追踪缓冲区[13]，仅记录稀疏性事件流信息；Rout 等利用各核 L2 Cache 作为本地追踪缓冲区，缓冲区内数据写满后执行数据转储[17]；Oh 等研究了一种基于 DRAM 的调试方法[18]，缓冲区数据写满后，将转储至 DRAM 中。然而，上述 3 种方式均未考虑转储过程中的系统延时必将导致数据溢出风险，也忽略了数据转储时追踪信息仍在实时生成。

3　RVTDS 框架

3.1　系统总体结构

RVTDS 架构如图 1 所示，系统由追踪处理器(Trace Processor，TP)和分布式追踪采样单元(Trace Sample Unit，TSU)组成。TP 基于 RISC-V 指令集实现，是 RVTDS 的控制中心，TSU 预配置以及追踪信息读写均由 TP 控制执行。TSU 是执行追踪任务的实施单元，由数据追踪采样单元(Da-

ta-TSU)和路径追踪采样单元(Path-TSU)组成,Data-TSU 和 Path-TSU 均包含寄存器预配置模块、追踪采样模块和转储模块 3 部分。追踪采样模块追踪到的数据记为 Trace-Item,为满足窄总线传输要求,该模块对其切分,生成 Trace-Subitem,并流入转储模块中。Data-TSU 通过监听 AXI 总线执行地址/数据追踪和总线流量统计。Path-TSU 实现路径追踪,记录用户感兴趣的指令信息。TP 与 TSU 的数据交互基于 APB 总线实现。

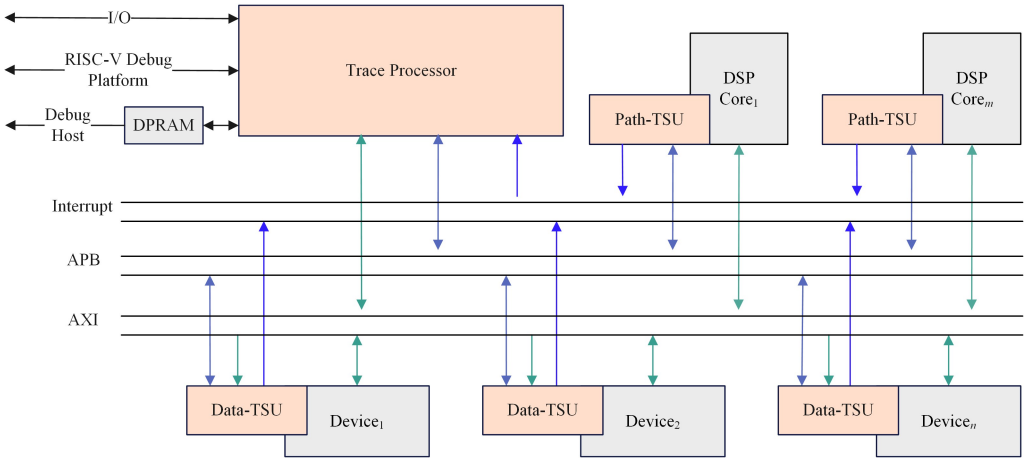


图 1 RVTDS 架构

Fig. 1 Framework of RVTDS

为实现对复杂多核系统多个功能模块的追踪,TP 采用触发-读取-写入的追踪记录方式,以减少多数据源记录追踪信息时的硬件开销,并解决短时间内接收到来自不同 TSU 的追踪信息时产生的"写"冲突问题。RVTDS 执行流程如图 2 所示。

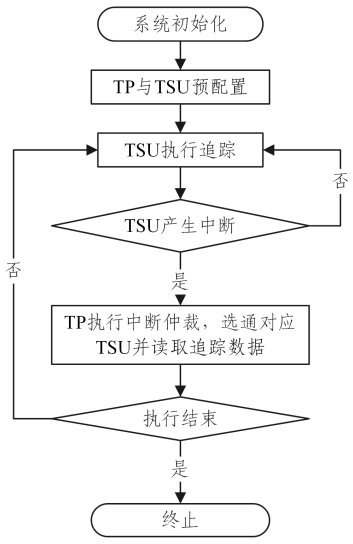


图 2 RVTDS 执行流程

Fig. 2 Execution process of RVTDS

首先,用户利用 RISC-V 调试平台完成对 TP 和 TSU 的预配置。其次,分布在 SoC 各功能单元上的 TSU 执行追踪任务,当检测到有需要记录的内容时,产生中断并告知 TP。最后,TP 执行中断仲裁,选通某个 TSU 并启用中断服务程序,将追踪信息从对应 TSU 中读出。追踪信息会存放在追踪缓冲区内,调试主机可直接读取。

为解决数据溢出问题,RVTDS 规定,追踪信息首先写入 TSU 内置缓冲区,缓冲区写满后将不再记录新数据,对应的追踪任务将暂停,直至 TP 选通该 TSU 并将追踪信息转储至追踪缓冲区内。

3.2 追踪处理器

TP 核上分布着中断管理器、2 个 APB-Master 接口和 1 个 AXI-Master 接口,如图 3 所示。

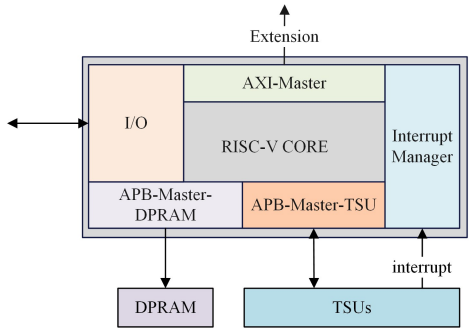


图 3 追踪处理器结构

Fig. 3 Trace processor architecture

中断管理器由中断控制器和中断服务程序两部分组成。中断控制器基于 RISC-V 核内平台级别中断控制器实现,执行中断仲裁与分发,TSU 为中断源,中断服务程序为中断目标。

AXI-Master 和 APB-Master 均用于追踪信息转储。某个 TSU 选通后,中断服务程序控制 APB-Master-TSU 将追踪信息从 TSU 中取出。若追踪信息来自 Path-TSU 且数据量较小,APB-Master-DPRAM 将其写入双端口 RAM(Dual PortRam ,DPRM)中;若追踪信息数据量较大或来自 Data-TSU,追踪信息将经 AXI-Master 写入外置存储。AXI-Master 可作为扩展接口使用,以应对总线死机等极端情况。

4 数据追踪采样单元

Data-TSU 主要追踪流经 AXI 总线的数据。图 4 中,追踪采样模块由地址/数据追踪子模块和字节量/时间追踪子模块组成。地址/数据追踪子模块基于地址/数据匹配完成对

AXI 总线读/写地址通道和读/写数据通道的追踪；字节量/时间追踪子模块主要用于实现 AXI 总线流量统计。

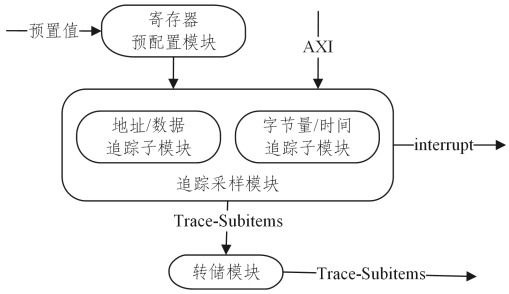


图 4　Data-TSU 结构

Fig. 4　Data-TSU architecture

4.1　地址/数据筛选方法

地址/数据追踪子模块基于掩码机制实现，通过匹配指定地址/数据，检测 AXI 总线是否访问了某个地址/地址段，或是否传递了某个数值/数据段。

若追踪某类信息 X(Addr/Data)，设 X_M 为掩码寄存器值，X_T 为触发寄存器值，式(1)表示实时 X 的匹配结果。

$$g(X)=X_M\&X==X_T \tag{1}$$

在 ADT-Sm 中，设 X_{CNT} 为触发阈值，若满足式(2)：

$$\sum g(X)\geqslant X_{\mathrm{CNT}} \tag{2}$$

即有效匹配个数不小于 X_{CNT} 时，将产生一个有效触发，并记录地址信息 A_t 或数据信息 D_t。

在执行追踪时，用户可配置实施读事务追踪或写事务追踪，读/写事务追踪不能同时进行。

4.2　总线流量统计功能

总线流量统计旨在记录总线数据流速，通过字节量/时间追踪子模块实现。总线数据流速计算常用两种方法，一是记录总线流经固定字节量所需时间(见式(3))，二是记录单位时间总线流经字节量(见式(4))。

$$f_B(t,B)=\frac{B}{t-t_{\mathrm{pre}}} \tag{3}$$

$$f_T(T,b)=\frac{b-b_{\mathrm{pre}}}{\mathrm{T}} \tag{4}$$

其中，f_B 表示基于 Bytes(B)求解的总线数据流速，B 为固定字节量，b 为当前总字节量，b_{pre} 为上次触发时的总字节量；f_T 表示基于 $Time(T)$求解的总线数据流速，T 为固定时间，t 为当前时间，t_{pre} 为上次触发时刻。

RVTDS 允许系统通过上述两种方式实现总线流量统计，同时支持两种时间统计模式，一是从追踪执行开始统计，二是从首次检测到有效地址时开始统计。

5　路径追踪采样单元

Path-TSU 基于指令 PC 位域匹配机制实现，如图 5 所示，寄存器预配置模块将指定的指令段转换成多个预置寄存器组，并完成寄存器预置。追踪采样模块用于完成指令过滤和追踪数据的后处理。经追踪采样模块处理后的数据将流入转储模块。转储模块采用一个异步 FIFO 作为追踪缓冲区缓存数据，待 TP 读取。

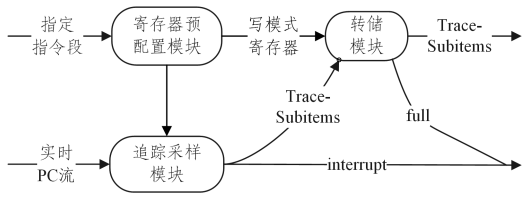


图 5　Path-TSU 结构

Fig. 5　Path-TSU architecture

5.1　寄存器预置值推荐方法

单个 Path-TSU 有 16 个预置寄存器组，每个预置寄存器组对应追踪采样模块中的 1 个比较选择通道(Comparing and Selecting Channel，CSC)，由触发寄存器(Trigger Reg，TR)、掩码寄存器(Mask Reg，MR)、触发次数寄存器(Trigger Count Reg，TCR)以及起始指令寄存器(Initial Instruction Reg，IIR)组成，可完成对单个核指令执行情况的追踪。

Path-TSU 提供寄存器预置值推荐功能，采用聚类算法实现，一个 Path-TSU 最多可区分 16 个类。Path-TSU 将两条指令 PC 值间的汉明距离作为聚类的判决条件。聚类结束后，根据结果类集合 C 中各类元素的数量大小对 C 重排序，使每个 CSC 的触发个数依次减少，此时，靠前的 CSC 必将覆盖更多的指令。TCR 基于聚类后各类元素个数设置，IIR 与各类的第一个 PC 值相同。

若追踪数据量较大，缓冲区写满时追踪尚未完成，仍有追踪信息生成，可增加写模式寄存器，以允许 FIFO 执行“回写覆盖”或“暂停等待”。

5.2　指令过滤与处理方法

如图 6 所示，指令经指令过滤子模块实现筛选；时间戳子模块记录当前时刻时间戳信息；数据后处理子模块对上述信息进行打包压缩，并切分为 Trace-Subitems 送入转储模块，供后续 TP 读取。

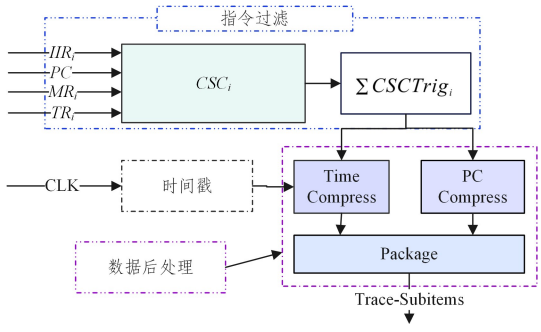


图 6　Path-TSU 指令过滤与处理结构

Fig. 6　Path-TSU instruction filtering and processing architecture

5.2.1　基于指令位域匹配的信号过滤机制

指令过滤由 CSC 完成。寄存器预配置结束后，系统可获知该追踪采样模块共需启动 CSC 的个数，即聚类后的类个数 N。若 $N<16$，则未启动的 CSC 所有使能 CSC_on 均置零。当 PC 进入追踪采样模块时，每个 CSC 按照算法 1 执行匹配触发任务。

算法 1　指令过滤算法

输入：PC，IIR，MR，TR，TCR

输出：isTrigger，PCBackup

```
1. for every PCdo
2.   for every valid CSCdo
3.     CSC_on_i=0, IIR_en_i=1, CSCTri g_i=0
4.     if(IIR_en_i && (PC==IIR_i))==1, then
5.       CSC_on_i=1, IIR_en_i=0
6.     end if
7.     if CSC_on_i==1 then
8.       while cn t_i≤TC R_i do
9.         if PC&MR_i==T_i, 且任意 CSC_j(j<i)未触发 then
10.          CSCTri g_i=1, cn t_i++
11.        else
12.          CSCTri g_i=0
13.        end if
14.      end while
15.      CSC_on_i=0
16.    end if
17.  end for
18.  if Σ_{i=0}^{15} CSCTri g_i>0then
19.    is Trigger=1, PCBackup=PC
20.  else
21.    isTrigger=0
22.  end if
23. end for
```

算法 1 中，“&”表示逻辑“与”，“&&”表示逻辑“位与”。参数定义如表 1 所列。

表 1 指令过滤算法参数定义

Table 1 Parameter definition of instruction filtering algorithm

参数	释义
i	第 i 个 CSC，$i=0,1,\cdots,N-1$
cnt_i	触发个数
IIR_{en_i}	IIR 匹配检测使能
CSC_{on_i}	使能当前 CSC 开启时为 1
$CSCTrig_i$	使能当前 CSC 触发时为 1
$isTrigger$	是否触发标志
$PCBackup$	触发时 PC 备份

程序开始时 CSC_on_i 为 0，IIR_en_i 为 1。若检测到当前 PC 等于 IIR_i，则 CSC_on_i 置 1，IIR_en_i 置 0，打开 CSC_i 以运行指令筛选。式(5)成立时，表示当前 PC 匹配结果为真，CSC_i 触发并记录其触发次数 $cn\ t_i$。$cn\ t_i$ 等于 TCR_i 时，CSC_i 将关闭。

$$\sum_{i=0}^{15} CSCTrig_i>0 \tag{5}$$

本文规定，若某 PC 在当前 CSC 中触发，则不会在其他 CSC 中触发，且满足从 CSC_0 到 CSC_{15} 的优先触发顺序。

5.2.2 基于差分编码的数据压缩方法

完成指令过滤后，需要对数据进行打包压缩，以减少资源开销。追踪采样模块将先对数据进行差分压缩，生成 Trace-Item；其次，为满足 16 位数据总线传输要求，对压缩后的数据打包切分，处理后的每个 16 位数据包称为一个 Trace-Sub-item。

市场上常见的嵌入式系统多为 32 位系统，可混合使用 32 位指令和 16 位指令。记相邻时钟周期所对应指令 PC 间的差值为 ΔPC。有研究使用周期精确的指令集仿真器对 DSPstone 中 18 个基准测试集进行了评估，发现大多数情况下 ΔPC 为 0，ΔPC 为 2 和 4 的概率依次减少，ΔPC 大于 4 的概率不足 10%[7]。因此，数据打包前，利用差分编码对压缩追踪信息，将大幅减少信息冗余度。

此外，路径追踪时，相邻两个不同的指令所在时刻分别记为 t_{last} 和 t_{next}，令：

$$\Delta t=t_{next}-t_{last} \tag{6}$$

本文统计了随机的 12 个程序相邻指令 PC 时间戳差值 Δt，即单个 PC 执行时钟周期数，如图 7 所示。

分析图 7 可知，大多数程序执行时单周期指令占比超过 90%，执行周期超过 6 拍的指令不足 1%，通常情况下指令执行周期数集中在 1～6 拍之间，从而采用少量比特即可记录 Δt。

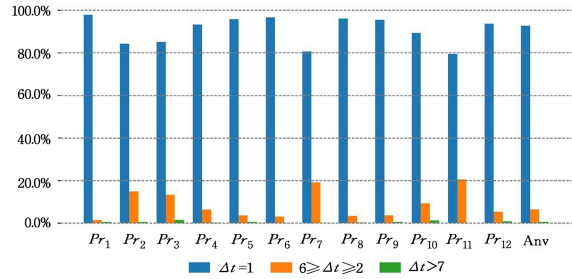


图 7 指令运行时钟周期数统计图

Fig. 7 Statistics of instruction running clock cycles

用 Y 表示 PC 或 Timestamp。Path-TSU 采用图 8 所示的格式对 Y 进行编码，将 Y 分为 Y-ID 和 Y-value 两部分。其中 Y-ID 为 Y 标识位，用于标记位宽；Y-value 为对应位宽的 ΔY。

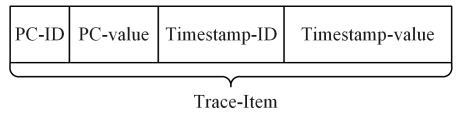


图 8 Trace-Item 格式

Fig. 8 Trace-Item format

本文采用表 2 所列的格式对指令 PC 进行编码，PC-ID 采用变长编码。若 ΔPC 为 2 或 4，则仅用 PC-ID 即可唯一标识当前 PC，Trace-Item 中不记录 PC-Value；若 $\Delta PC<2^{12}$，则 PC-ID 采用 4 位编码；$\Delta PC\geqslant 2^{12}$ 或 $\Delta PC<0$ 时，采用 5 位编码。本文规定，Timestamp 和 PC 初值为零。

表 2 PC 编码格式

Table 2 PC encoding format

PC-ID			ΔPC
00			保留
01	00		$\Delta PC\in\{6,8,10,12,14\}$
	01		$\Delta PC\in\{x\mid 16\leqslant x\leqslant 2^8, x\in Z\}$
	10		$\Delta PC\in\{x\mid 2^8<x<2^{12}, x\in Z\}$
	11	0	$\Delta PC\in\{x\mid x\geqslant 2^{12}, x\in Z\}$
		1	$\Delta PC\in\{x\mid x<0, x\in Z\}$
10			2
11			4

Path-TSU 利用 4 位二进制数对 Timestamp-ID 编码以减少信息冗余度，如表 3 所列。Δt 值为 1～6 时，Timestamp-ID 可分别编码为 0x1－0x6，此时，直接使用 Timestamp-ID 表示整个 Timestamp 值，Trace-Item 中不记录 Timestamp-value。

需要注意的是，若 FIFO 写模式为“回写覆盖”，为保证数据的可恢复性，FIFO 写满回写的第一个 Trace-Item 应包含完整时间戳信息与 PC 值，以及与上一个追踪信息的差分编码结果，Timestamp-ID 将被编码为 0xF。

表 3　Timestamp 编码格式

Table 3　Timestamp encoding format

Timestamp-ID	Δt
0x0	保留
0x1－0x6	分别表示 Δt 为 1～6
0x7	$\Delta t\in\{x\mid 6<x<2^6, x\in Z\}$
0x8	$\Delta t\in\{x\mid 2^6\leqslant x<2^{12}, x\in Z\}$
0x9	$\Delta t\in\{x\mid 2^{12}\leqslant x<2^{18}, x\in Z\}$
0xA	$\Delta t\in\{x\mid 2^{18}\leqslant x<2^{24}, x\in Z\}$
0xB	$\Delta t\in\{x\mid 2^{24}\leqslant x<2^{30}, x\in Z\}$
0xC	$\Delta t\in\{x\mid 2^{30}\leqslant x<2^{36}, x\in Z\}$
0xD	$\Delta t\in\{x\mid 2^{36}\leqslant x<2^{42}, x\in Z\}$
0xE	$\Delta t\in\{x\geqslant 2^{42}, x\in Z\}$
0xF	写模式为“回写覆盖”时使用

5.2.3　适应窄总线传输的数据打包方法

为解决追踪信息在窄总线上的传输问题，Path-TSU 采用图 9 所示的格式对压缩后的数据进行打包切分。

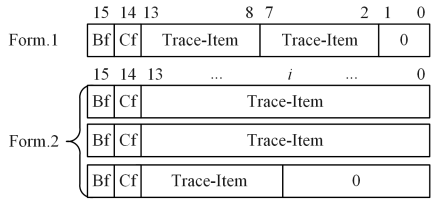


图 9　Trace-Subitem 格式

Fig. 9　Trace-Subitem format

Trace-Subitem 中，Bf 位为归属标志位，同一个 Trace-Item切分的 Trace-Subitem，Bf 位相同；相邻 Trace-Item 之间，Bf 位 0 和 1 交替出现。Cf 表示该 Trace-Subitem 中包含的 Trace-Item 个数。路径追踪大部分时间，Trace-Item 位宽为 6，若相邻两个 Trace-Item 位宽均为 6，可以打包为 1 个 Trace-Subitem，并将 Cf 位置“1”，图中 Form. 1 即为此类型。

6　验证与分析

为评估 RVTDS 系统的实用性，本文基于某多核处理器 FT-XDSP 的 FPGA 原型验证平台搭建了一个功能验证系统，如图 10 所示。首先，RVTDS 代码经编译、综合后生成比特流，烧写至 FPGA 验证平台；其次使用 RISC-V 调试平台完成 RVTDS 预配置；最后，在嵌入式开发环境上运行程序。

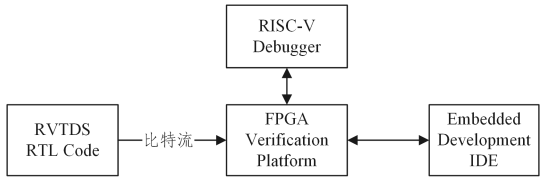


图 10　功能验证系统结构

Fig. 10　Structure of functional verification system

6.1　系统实用性分析

本节采用注入 bug 的方式验证系统实用性，程序执行 DMA 控制下的共享存储控制器与 L2 Cache 间的数据交互过程。搬运过程代码如程序 1 所示。

程序 1　示例中 DMA 搬运执行代码

```
1. DMA_Cfg(DMA_Num,0x0,0x10,SRC_ADDR,DATA _SIZE,DST_
   ADDR,0x0);
2. DMA_Start(DMA_Num,0x0,0x0,0x1);
3. DMA_End(DMA_Num,0x1,0x0);
4. COMPARE(SRC_ADDR,DST_ADDR,0x20).
```

示例程序中，DMA_Cfg 用于实现 DMA 配置，DMA_Start 用于启动 DMA。数据搬运结束后，执行 DMA_End，关闭 DMA。COMPARE 用于比较搬运前后数据是否一致。DMA 配置时，SRC_ADDR 和 DST_ADDR 分别为起始地址和目的地址；DATA_SIZE 为数据量，其中高 16 位为数据帧数，低 16 位为每帧字节量。

RVTDS 系统允许追踪调试人员指定的指令段，指令筛选时将记录该指令执行时刻，这种追踪机制可以完整复现指令执行行为，通过与黄金模型比对，来确定系统执行是否正常。

追踪上述 4 个程序段将获知 DMA 是否正常完成数据搬运过程，其起始地址依次为 0x0C000824，0x0C0008E2，0x0C00094A 和 0x0C000A30。为方便起见，各指定程序段仅追踪其起始地址。完成寄存器预配置后执行追踪，追踪结果如图 11 所示。从图 11 可知，程序顺序执行了 DMA_Cfg，DMA_Start 和 DMA_End，未执行至 COMPARE。程序中 DATA_SIZE 为 0x0，所传输的数据帧数与字节量均为 0，数据搬运过程将无法执行。将其重置后运行程序，此时，Data-TSU 未产生有效地址触发信号。经检查发现，总线数据传输的目的地址有误。完成纠错后程序恢复正常。

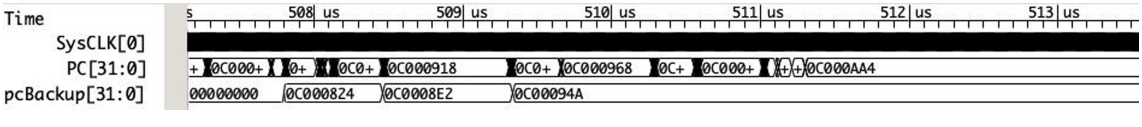


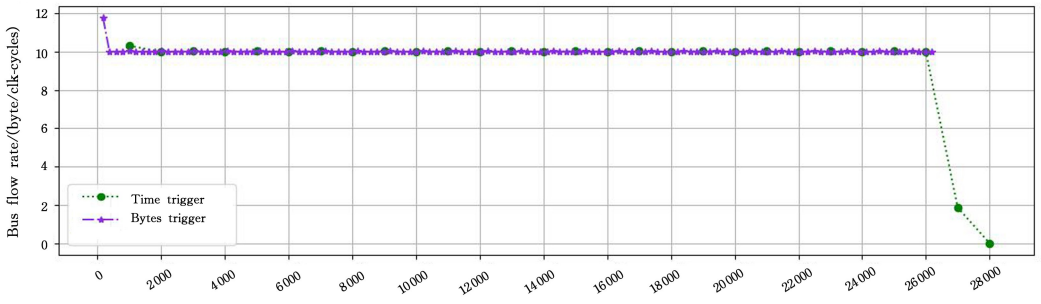
图 11　实例中的路径追踪结果

Fig. 11　Path tracing results in example

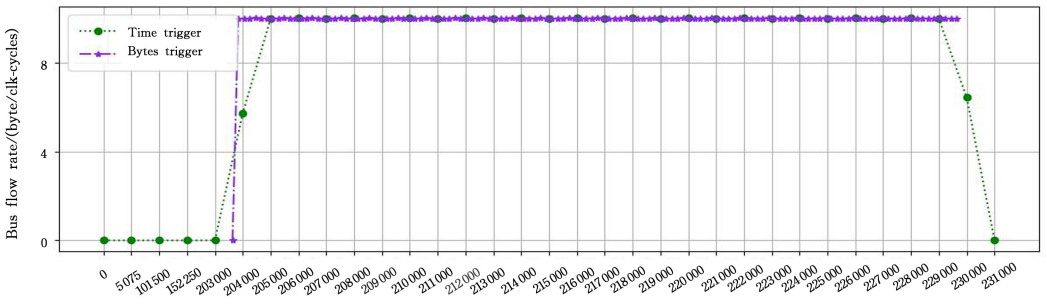
6.2　总线带宽分析

真实工程调试时，需要考虑两种情况，一是在系统执行时开始统计时间；二是仅当首次检测到有效地址时就开始统计时间。

图 12 给出了两种情况下基于字节量触发和基于时间触发的结果统计。



(a)从检测到有效地址开始统计的结果



(b)从程序执行开始统计的结果

图 12 字节量/时间触发结果统计

Fig. 12 Statistics of byte sizes/time triggered results

图 12 中,因两种触发方式的差异,字节量触发结果曲线将先于时间触发结果曲线结束,此时仍有数据传输,因不满足触发条件,将不再产生有效触发。如图 12(a)所示,若检测到有效地址后开始执行追踪,追踪初始阶段,两种触发方式有所不同;在数据稳定传输时,总线数据流速将保持一致,单个时钟周期可传输 10 个字节数据;在总线数据传输结束后,时间触发将继续进行,此时总线数据流速为零。由图 12(b)可知,数据交互过程发生在程序执行尾部,程序执行前期无总线数据交互行为。这种情况将产生大量无效统计,但可以监控总线全局行为。此外,从统计结果来看,记录完整数据流必占用大量资源。执行信息过滤,能大幅减少追踪数据量。

6.3 路径追踪结果分析

工程中随机选用 12 个不同程序,每个程序随机指定多个指令段进行追踪,以分析该系统的性能。

6.3.1 指令过滤结果分析

本文设置了 4 种d_{th}(0,1,2,3)值,以分析不同d_{th}对聚类结果的影响。对第 k 个程序 Pr_k,若不考虑重复指令,设$M_{all}(Pr_k)$为程序指令总个数,$M_t(Pr_k)$为待追踪指令个数。聚类判决条件为d_{th}时,令 $M(Pr_k,d_{th})$为可匹配到整个程序中的指令个数,$N(Pr_k,d_{th})$为聚类后类的个数,$p(Pr_k,d_{th})$表示额外追踪指令的比例,则:

$$M_t(Pr_k)\equiv M(Pr_k,0) \tag{7}$$

$$p(Pr_k,d_{th})=\frac{M(Pr_k,d_{th})-M_t(Pr_k)}{M_t(Pr_k)} \tag{8}$$

由式(7)可知,d_{th}为 0 时为完全匹配,即恒等于待追踪指令个数。如表 4 所列,本文统计了不同d_{th}值与其对应的匹配个数,以及d_{th}为 2 时 TCR 和 IIR 的设置对触发次数的影响。M_t表示追踪到的指令个数;IIR_B表示在检测到 IIR 前满足式(6)的个数;Unlimited 表示不对 TCR 进行限制;1.2×TC表示 TCR 基于 TCs 设置,TCR 设置为 TCs 的 1.2 倍。

表 4 匹配个数统计

Table 4 Number of matches

Pr	d_{th}										$d_{th}=2$			
	0	1			2			3			M_t	IIR_B	TCR	
		M	N	p/%	M	N	p/%	M	N	p/%			*Unlimit*	1.2×TC
1	388	497	24	28.1	500	4	28.9	725	3	86.9	601	95	10349	759
2	248	307	56	23.8	464	12	87.1	799	9	222.2	387	1027	23979	607
3	355	477	39	34.4	672	7	89.3	1256	4	253.8	429	239	19767	642
4	673	999	101	48.4	1287	16	91.2	468	3	623.3	695	160	14909	932
5	636	699	114	9.9	2915	12	358.3	3316	4	421.4	952	1947	24372	1419
6	455	665	133	46.2	1312	16	188.4	2259	3	396.5	803	1	73060	1019
7	585	1181	59	101.9	3814	14	552.0	7099	3	1113.5	746	901	30585	1294
8	341	536	68	57.2	695	15	103.8	3383	4	892.1	606	187	14435	943
9	363	499	37	37.5	553	13	52.3	602	7	65.8	540	329	15620	777
10	593	658	82	11.0	839	20	41.5	2979	3	402.4	965	126	1719	1439
11	292	339	43	16.1	796	17	172.6	5209	7	1683.9	546	31	36862	1061
12	1023	2340	107	128.7	1530	16	49.6	10290	3	905.9	1831	14234	422993	2774
anv	496	766	—	45.3	1281	—	151.2	3565	—	589.0	—	—	—	—

d_{th} 为 1 时，聚类后类的个数远大于单个 Path-TSU 的通道数，系统在信号筛选时不能完全覆盖待追踪指令。d_{th} 为 2 时，仅个别程序类的个数多于 16，且这些类唯一指向一个 PC，舍弃时对系统影响较小。d_{th} 为 3 时，系统可完全覆盖待追踪指令。考虑到 d_{th} 为 2 时平均额外追踪比为 151.2%，d_{th} 为 3 时平均额外追踪比为 589.0%，故本文建议，在聚类时 d_{th} 设置为 2。

此外，若不设置 IIR，执行指令追踪时，仅 Pr_6 和 Pr_{11} 实际追踪结果较待追踪指令出入不大，其余程序追踪结果均有较大问题。对于 Pr_2，Pr_5，Pr_7 和 Pr_{12}，在匹配待追踪指令前已记录大量无关指令。故设置 IIR，当各 CSC 检测到对应 IIR 时开启通道匹配选择功能，将有效减少冗余追踪。

单个程序 TCR 为定值时，会出现部分 CSC 未完全覆盖待追踪指令而部分 CSC 过度追踪的问题。若 TCR 不作限制，将追踪到大量无关指令，追踪完整指令流时更甚。为减少冗余追踪，尽可能完整匹配待追踪指令，本文基于 TC 完成 TCR 配置，在表 4 中，令其为 $1.2\times TC$，解决了指令筛选问题。

6.3.2　数据后处理结果分析

本文分析了压缩性能，结果如表 5 所列，其中 C 为数据总量，C_{cp} 为压缩后数据量，p_{cp} 为数据压缩率。不进行压缩编码时，每个 Trace-Item 位宽为 80 位，其中时间戳 48 位，PC 值 32 位，故数据总量 C 应为追踪到的指令个数×80bits。LZMA 是基于 LZ77 编码改良的一种通用压缩算法，在表 5 中，本文对其压缩结果进行了统计。

表 5　压缩性能统计

Table 5　Statistics of compression performance

Pr	M_t	C/bit	C_{cp}/bit	p_{cp}/%
1	759	60720	9376	84.56
2	607	48560	9312	80.82
3	642	51360	8368	83.71
4	932	74560	12832	82.79
5	1419	113520	19472	82.85
6	1019	81520	16800	79.39
7	1294	103520	16992	83.59
8	943	75440	14064	81.36
9	777	62160	9760	84.30
10	1439	115120	20256	82.40
11	1061	84880	14480	82.94
12	2774	221920	41264	81.41
Anv	13666	1093280	192976	82.35
LZMA	13666	1093280	427456	60.90

采用差分压缩编码后，数据平均压缩率可达 82%；同时，大部分程序执行路径追踪时，压缩率超过 80%，显著降低了信息冗余度。此外，LZMA 平均压缩率为 60.9%，低于差分压缩结果。

追踪信息在执行完压缩后，需要按照特定格式打包为 Trace-Item。为满足数据传输需求，Trace-Item 将被重新处理为 Trace-Subitem。表 6 列出了数据打包与切分结果，其中，(ts,ti) 表示 Trace-Subitem 个数为 ts 且对应的 Trace-Item 个数为 ti。

表 6　数据打包与切分结果统计

Table 6　Statistics of data packaging and segmentation results

Pr	Trace-Item 个数										$\sum N_{ts}$	$\frac{\sum N_{ts}}{M(Pr_k,0)}$/%
	(1,2)		(1,1)		(2,1)		(3,1)		(4,1)			
	N_{ts}	p_{ts}/%	N_{ts}	p_{ts}/%	N_{ts}	p_{ts}/%	N_{ts}	p_{ts}/%	N_{ts}	p_{ts}/%		
1	309	66.45	131	28.17	5	1.08	5	1.08	0	0.00	465	61.26
2	201	44.18	169	37.14	25	5.49	9	1.98	2	0.44	455	74.96
3	264	65.35	99	24.50	7	1.73	5	1.24	3	0.74	404	62.93
4	355	57.35	194	31.34	16	2.58	10	1.62	2	0.32	619	66.42
5	576	64.36	226	25.25	31	3.46	9	1.01	1	0.11	895	63.07
6	363	49.52	222	30.29	66	9.00	4	0.55	1	0.14	733	71.93
7	521	60.79	198	23.10	27	3.15	24	2.80	3	0.35	857	66.23
8	383	62.89	143	23.48	23	3.78	7	1.15	4	0.66	609	64.58
9	334	71.83	94	20.22	10	2.15	3	0.65	2	0.43	465	59.85
10	543	54.52	269	27.01	68	6.83	16	1.61	0	0.00	996	69.21
11	435	61.53	144	20.37	14	1.98	32	4.53	1	0.14	707	66.64
12	893	46.37	952	49.43	27	1.40	9	0.47	0	0.00	1926	69.43
Anv	5177	56.70	2841	31.11	319	3.49	133	1.46	19	0.21	9131	66.82

一个 Trace-Item 最多可生成 6 个 Trace-Subitem，单个 Trace-Subitem 最多可存储两个 Trace-Item 信息，故 (ts,ti) 共有(1,2)，(1,1)，(2,1)，(3,1)，(4,1)，(5,1)和(6,1)这 7 种情形，N_{ts} 为追踪结果中各情形所对应的 Trace-Subitem 个数，p_{ts} 表示 N_{ts} 所占 Trace-Subitem 总个数的比例。当 (ts,ti) 为(5,1)和(6,1)时，因每个程序统计量均为零，故未在表中描述。

表 6 中，各程序追踪信息中(1,2)的统计概率大都超过 50%，其中 Pr_9 大于 70%。此外，$ti=1$ 的概率均达到 80%以上，即不少于 80%的追踪信息可以用一个 Trace-Subitem 描述。在执行差分压缩与数据打包处理后，追踪信息中 Trace-Subitem 个数与需要追踪到的指令数之比均值为 66.82%，这意味着每个 Trace-Subitem 可传输约 1.5 个路径信息。

结束语　本文基于 RISC-V 架构设计了一种面向微处理器的追踪调试系统 RVTDS，旨在以相对可靠的方式协助调试人员更为高效地完成嵌入式开发调试。为解决追踪数据量大时的通路拥塞问题，本文面向片上总线提出了一种基于掩码机制的数据流追踪方案，系统提供基于字节量/时间触发的总线带宽统计功能。为解决对调试人员指定指令段的追踪问题，本文提出了一种基于 PC 位域匹配的指令过滤机制，提供预置寄存器值推荐功能；采用了基于差分编码的压缩方案，数据平均压缩率达 82%以上；为解决追踪信息在窄总线上的数据传输问题，提出了一种数据打包方案，追踪信息转储时，

每拍有效数据平均可容纳约 1.5 个路径信息。

然而，RVTDS 直接采用开源 RISC-V 架构，引入非必要功能，增加了系统功耗；此外，系统寄存器预配置由 RISC-V 调试平台实现，该平台与嵌入式系统开发环境相互独立，制约了系统可移植性。未来将对系统进行裁剪，剔除冗余功能；同时采用可移植库实现寄存器预配置功能，或提供 RISC-V 调试插件，以集成至嵌入式系统开发环境中。

参 考 文 献

[1] ALABOUDI A, LATOZA T D. An Exploratory Study of Debugging Episodes[J]. arXiv. 2105.02162, 2021.

[2] POUGOTI M, MILENKOVIC A. Enabling On-the-Fly Hardware Tracing of Data Reads in Multicores[J]. ACM Transactions on Embedded Computing Systems. 2019, 18(4): 1-27.

[3] BI R, HU H. A Non-Intrusive Real-time Debugging Method for Programs: CN107315685A[P]. 2017.

[4] JTAG Technologies. When does boundary-scan make sense[EB/OL]. https://www.jtag.com/white-paper/when-does-boundary-scan-make-sense/.

[5] THOMAS B P, SMITHA G, ABHI D R. Everything You Always Wanted to Know About Embedded Trace[J]. Computer, 2022, 55(2): 34-43.

[6] HU X L, JIN Y, LI Z L. A Parallel JTAG-based Debugging and Selection Scheme for Multi-core Digital Signal Processors[C]// 2018 IEEE International Conference of Safety Produce Informatization(IICSPI). 2018: 527-530.

[7] HSIEH M C, HUANG C T. An embedded infrastructure of debug and trace interface for the DSP platform[C]// IEEE Design Automation Conference. 2008.

[8] DUAN H X, YU L X, ZHOU H Y, et al. An On-Chip AHB Bus Tracer for Non-intrusive Debugging[C]// 2019 IEEE 3rd Advanced Information Management, Communicates, Electronic and Automation Control Conference(IMCEC). 2019: 322-326.

[9] DUAN H X, YU L X, ZHOU H Y, et al. An Embedded Tracing Debug Implementation for Crossbar Type Bus in Muti-core SoC [C]// 2020 IEEE 3rd International Conference on Electronics Technology(ICET). 2020: 63-67.

[10] RAMIREZ W, ROA E. Post-Silicon Debugging Platform with Bus Monitoring Capability to Perform Behavioral and Performance Analyses[C]// 2019 IEEE 10th Latin American Symposium on Circuits & Systems(LASCAS). 2019: 81-84.

[11] WANGER P, WILD T, HERKERSDORF A. DiaSys: On-Chip Trace Analysis for Multi-processor System-on-Chip[C]// International Conference on Architecture of Computing Systems. 2016.

[12] CHENG Y, LIH W, SHEN H H, et al. On Trace Buffer Reuse-Based Trigger Generation in Post-Silicon Debug [J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2018, 37(10): 2166-2179.

[13] CAO Y T, ZHENG H, RAY S. A Communication-Centric Observability Selection for Post-Silicon System-on-Chip Integration Debug[C]// 20th Inter-national Symposium on Quality Electronic Design(ISQED). 2019: 278-283.

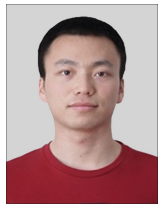
[14] XIAO W Y, ZHENG L X. Adaptive Huffman Coding System and Method: CN114900193A[P]. 2022-08-12.

[15] CHUNG-FU K, SHYH-MING H, ING-JER H. A Hardware Approach to Real-Time Program Trace Compression for Embedded Processors[J]. IEEE Transactions on Circuits & Systems, 2007, 54(3): 530-543.

[16] FU-CHING Y, YI-TING L, CHUNG-FU K, et al. An On-Chip AHB Bus Tracer With Real-Time Compression and Dynamic Multiresolution Supports for SoC[J]. IEEE Transactions on Very Large Scale Integration Systems, 2011, 19(4): 571-584.

[17] ROUT S S, DEB S, BASU K. Wind: An efficient post-silicon debug strategy for network on chip[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2020, 40(11): 2372-2385.

[18] OH H, CHOI I, KANG S. DRAM-Based Error Detection Method to Reduce the Post-Silicon Debug Time for Multiple Identical Cores[J]. IEEE Transactions on Computers, 2017, 66(9): 1504-1517.



GAO Xuan, born in 1996, postgraduate. His main research interests include hardware verification, microprocessor and embedded development.

CHE Wenbo, born in 1981, master, engineer. His main research interests include microelectronics, embedded development and computer architecture.

（责任编辑：何杨）