文章编号:1008-1542(2023)03-0211-08

虚拟 GDB 远程调试技术及其在硬件仿真器中的应用

黄　侃[1,2],徐　勇[1],李艳荣[3]

(1.哈尔滨工业大学(深圳)计算机科学与技术学院,广东深圳　518000;2.国微集团(深圳)有限公司,广东深圳　518000;3.深圳国微晶锐技术有限公司,广东深圳　518000)

摘　要:为了解决当前硬件仿真器采用测试激励与探针采样调试方法导致的额外硬件资源需求增加以及调试方法复杂死板的问题,研究出一种虚拟 GDB 远程调试技术提高硬件仿真器的调试能力。首先,使用 JTAG 的消息代理器与事务处理器进行软件与硬件的信息交互,实现计算机与硬件仿真器的通信;然后,通过 VPI 接口实现 GDB 与 JTAG 代理器的软件信息交互,完成虚拟 GDB 调试;最后,在硬件仿真器中进行应用实验。结果表明,虚拟 GDB 远程调试系统可支持断点插入、单步运行、寄存器读写等功能,与插入探针的调试方法相比,使用虚拟 GDB 调试提速近百倍。虚拟 GDB 远程调试技术丰富了硬件仿真器的软件调试手段,增强了硬件仿真器的调试能力,提升了芯片设计验证效率。

关键词:计算机仿真;远程调试技术;硬件仿真加速器;SCEMI;JTAG 事务器;OpenOCD

中图分类号:TP391.9　　**文献标识码**:A　　**DOI**:10.7535/hbkd.2023yx03001

Virtual GDB remote debugging technology and its application in emulator

HUANG Kan[1,2], XU Yong[1], LI Yanrong[3]

(1. School of Computer Science and Technology, Harbin Institute of Technology (Shenzhen), Shenzhen, Guangdong 518000, China; 2. SMiT Group (Shenzhen) Company Limited, Shenzhen, Guangdong 518000, China; 3. Shenzhen SMiT Jingrui Technology Company Limited, Shenzhen, Guangdong 518000, China)

Abstract: In order to solve the problem that the testbench stimulation and probe insertion debugging method of emulator requires extra hardware resources and has insufficient flexibility in software debugging, a virtual GDB remote debugging technology was researched to improve the debugging ability of the emulator. Firstly, JTAG proxy and JTAG transactor were

收稿日期:2023-04-24;修回日期:2023-06-06;责任编辑:张士莹
基金项目:国家自然科学基金面上项目(61876051);深圳市科创委资助项目(KCXFZ20211020163402004)
第一作者简介:黄　侃(1990—),男,陕西蒲城人,博士,主要从事硬件仿真加速器系统及算法方面的研究。
E-mail:hkhuangkan@hotmail.com

黄侃,徐勇,李艳荣.虚拟 GDB 远程调试技术及其在硬件仿真器中的应用[J].河北科技大学学报,2023,44(3):211-218.
HUANG Kan,XU Yong,LI Yanrong. Virtual GDB remote debugging technology and its application in emulator[J]. Journal of Hebei University of Science and Technology,2023,44(3):211-218.

used to support the connection between the workstation and the emulator. Then the signals between GDB and JTAG proxy were transmitted through VPI to complete the debugging of virtual GDB. Finally, the virtual GDB remote debugging system was applied in emulator for experiments. The results show that the virtual GDB remote debugging system can support breakpoint insertion, single step running, register reading and writing, and other functions. Compared with debugging by probe insertion, the virtual GDB system is almost hundred times faster. It greatly enriches the software debugging method and enhances the debugging ability of emulator, and improves the efficiency of the chip design and verification.

Keywords: computer simulation; remote debugging technology; emulator; SCEMI; JTAG Transactor; OpenOCD

随着超大规模集成电路的不断增长,以及产品迭代周期的不断缩短,电路仿真验证变得愈加困难。基于软件模拟(software simulation)的验证方法逐渐无法适用于如今超大规模集成电路的仿真验证,尤其是在仿真速度上出现了瓶颈问题[1]。因此,使用能够高速运行的硬件进行仿真加速成为现代超大规模集成电路仿真验证的重要技术途径。基于硬件仿真器(emulator)的仿真验证技术成为芯片设计及验证从业人员极为关注的重要技术手段[2]。

硬件仿真器常用的验证方法包含在线仿真模式(in-circuit emulation,ICE)[3]、基于事务加速模式(transaction based acceleration,TBA)[4]以及带目标的仿真加速模式(simulation acceleration with target)[5],通过测试激励对待测器件(device-under-test,DUT)进行激励并将反馈信号与预期结果相比较来查找问题。文献[6]采用测试激励对Xilinx公司的FFT(fast fourier transform) IP核进行激励并获取反馈数据进行仿真调试,但是其只能针对一定周期数的指定信号进行采样来验证IP核的正确性,该IP相对于复杂芯片仅是极其微小的一部分,对于完整芯片的验证调试复杂度会成指数级增长。文献[7]构建了基于TBA的验证平台,需要构建额外的生成器、检查器、监视器等器件来完成对待测器件的验证,其对额外硬件资源需求增加较为明显。同时,还需要验证人员开发配套器件,也提升了验证成本。文献[8]通过增加探针及额外I/O卡进行软件调试,增加了软件调试的复杂性。此外,常用硬件仿真器中实时调试的手段只能通过加入静态探针、动态探针等方法获取DUT内部信号数据进行调试[9]。以上的仿真调试验证方法不仅需要消耗额外的硬件资源,还存在对DUT内运行软件调试灵活性不足的问题。同时,企业级硬件仿真器物理硬件复杂,JTAG(joint test action group)接口不便于接入,更加影响了硬件仿真器中的软件调试手段。

因此,为了使硬件仿真加速器的调试手段更加灵活,获取DUT内部信号的能力更加便捷,同时为了解决硬件仿真器不方便接入JTAG仿真器,以及尽可能减少验证成本及时间等问题,结合GDB(gnu debugger)调试技术、VPI通信接口以及基于SCEMI协议的JTAG代理器及事务器技术,对虚拟GDB远程调试技术进行研究,并在硬件仿真器中进行应用实验。

1 硬件仿真器中的JTAG调试系统

基于国际标准测试协议(IEEE 1149.1)的JTAG调试是如今普遍使用的嵌入式调试技术[10]。JTAG接口可读取或写入芯片内部封装的边界扫描链(boundary scan chain)寄存器的信息,帮助设计测试人员对DUT内部的行为逻辑进行调试。经典的JTAG调试系统结构如图1所示[11],通过协议转换器将调试主机与调试对象相连。协议转换器主要将调试主机发送的高层调试命令转换为底层JTAG调试命令。现有JTAG协议转换器需要通过物理硬件来实现,如商用JLINK[12],ST-LINK[13]等。

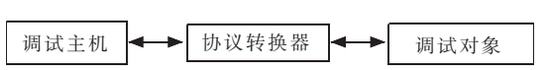

图1 JTAG调试系统结构

Fig. 1 JTAG debugging system structure

本文所用的JTAG调试命令通过TRSTn,CLK,TMS,TDI,TDO 5条控制及信号线进行数据交互,通过有限状态机控制各个信号的发送与接收,该状态机也是JTAG调试的核心运行逻辑,通过运行逻辑保证系统的正确运行,如图2所示。TRSTn信号为可选信号,用来对状态机复位,恢复到Test-Logic Reset状态。在TCK信号的驱动下,TMS信号对状态机状态进行控制,TDI与TDO信号访问边界扫描链的2种寄存器:指令寄存器(IR,instruction register)与数据寄存器(DR,data register)。通过对指令寄存器的配置可以控制待测器件按指定命令执行功能,通过对数据寄存器的写入或读取来控制待测器件的数据输入或反馈

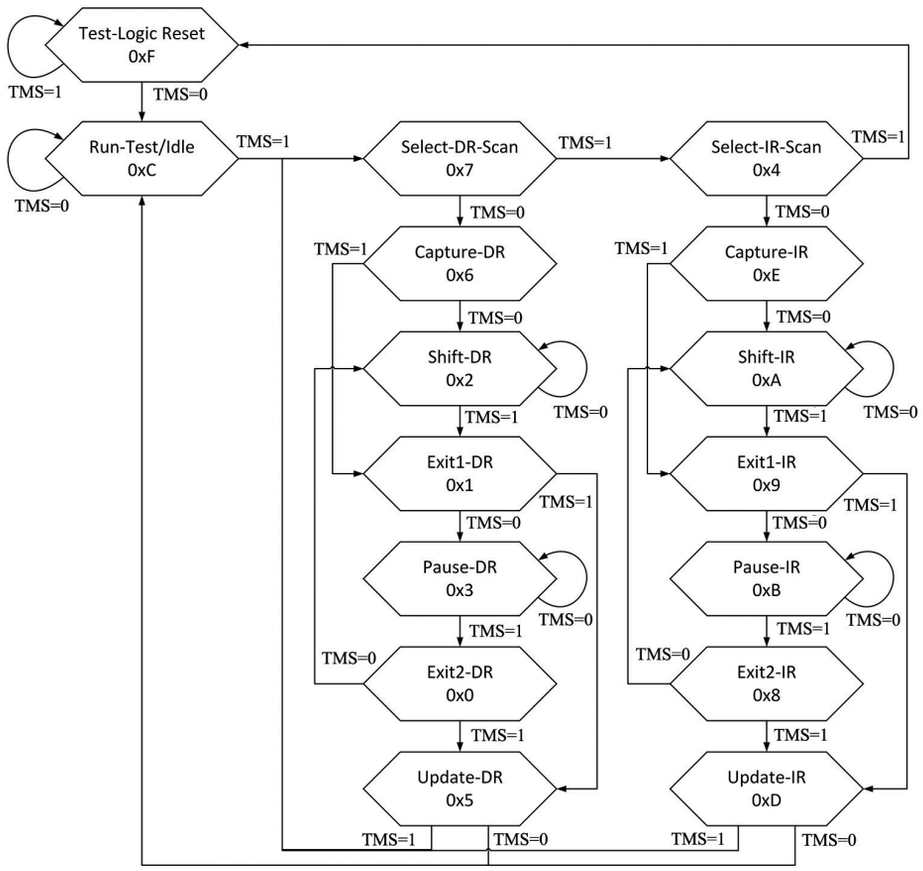

图 2　JTAG 有限状态机示意图

Fig. 2　Flow chart of JTAG finite state machine

输出。由以上方式完成待测器件的 JTAG 调试。

在硬件仿真器中,本文基于 SCE-MI(standard co-emulation modeling interface)协议[14]生成 JTAG 的事务器模块,用来进行 JTAG 协议转换。硬件仿真器中的 JTAG 调试系统结构如图 3 所示。

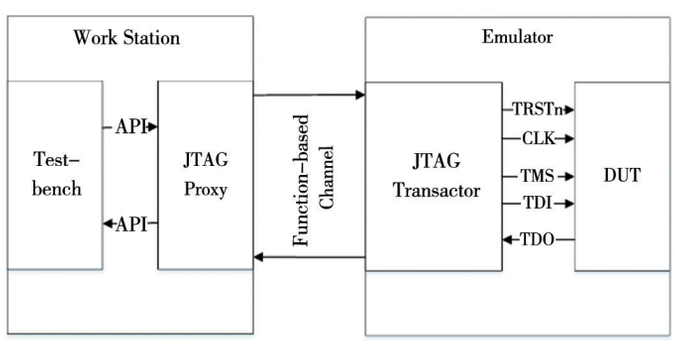

图 3　硬件仿真器中的 JTAG 调试系统

Fig. 3　JTAG debugging system in emulator

图 3 中 JTAG Proxy 及 JTAG Transactor 为基于 SCE-MI 协议 Function-based Channel 实现的 JTAG 软件端代理器和硬件端事务器模块。通过 SystemVerilog 的 DPI (direct programming interface)接口[15]实现接口函数,完成代理器及事务器的消息互通。JTAG Proxy 及 Transactor 中所调用及声明的函数接口如表 1 所示。

表 1　JTAG Proxy 及 Transactor 中接口列表

Tab. 1　Interface list of JTAG Proxy and Transactor

接　口	功　能
Send_result_to_server()	Transactor 发送反馈信号给 Proxy
Send_cmdend_to_server()	Transactor 发送完成信号给 Proxy
Send_cmd_to_xtor()	Proxy 发送命令及数据给 Transactor
Send_cmd()	Testbench 发送指令到 Proxy
Send_data()	Proxy 发送数据到 Testbench

Send_result_to_server()及 Send_cmdend_to_server()是在代理器中实现的 DPI 接口,在事务器中进行主动调用,从而将 JTAG 中 TDO 的输出信号以及输出信号的终止信号发送给代理器。Send_cmd_to_xtor()是事务器提供的 DPI 接口,供代理器调用,用来发送命令及数据给事务器。Send_cmd()是代理器提供的 API 接口,用来让测试激励给代理器发送命令。Send_data()是代理器提供的 API 接口,用来给测试激励发送反馈数据。

JTAG 事务器中通过有限状态机控制命令的解析、执行以及反馈。命令控制状态机的示意图如图 4 所示。该事务器有限状态机是控制事务器与代理器运行逻辑的核心模块,需要该状态机来控制事务器与代理器信号的发送与接收,从而保证事务器与代理器的正确运行。

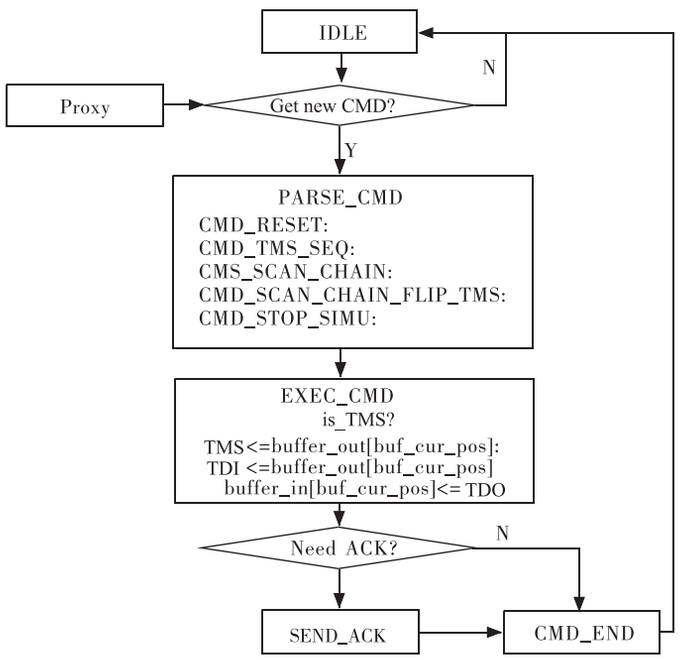

图 4 JTAG Transactor 命令控制状态机

Fig. 4 Command FSM of JTAG Transactor

该状态机中 IDLE 为空闲状态,PARSE_CMD 为解析命令状态,EXEC_CMD 为执行命令状态,SEND_ACK 为发送回复状态,CMD_END 为命令完成状态。其中 PARSE_CMD 状态中的命令包含 RESET,STOP_SIMU,TMS_SEQ,SCAN_CHAIN, SCAN_CHAIN_FLIP_TMS 共 5 条命令,用来根据上位机发送的 CMD 命令控制 TMS 信号。EXEC_CMD 状态中根据解析得到的 TMS 信号,控制发送 TMS 和 TDI 信号给 DUT,或者接收 DUT 发送的 TDO 信号。SEND_ACK 状态将接收到的 TDO 信号传送给 Proxy。代理器与事务器间交互的消息包含 CMD,buffer_in,buffer_out,length,nb_bits 共 5 个数据。其中 CMD 为命令状态机中的命令;buffer_in 存储 TDO 信号;buffer_out 存储 TMS 或 TDI 信号;length 存储代理器中数据的长度;nb_bits 存储事务器中需解析的数据位数。

通过 JTAG 事务器中命令控制状态机的控制以及代理器与事务器间消息的传递,JTAG 代理器与事务器间即可构成调试信息通道,从而进行对待测器件的信息输入与反馈信号的获取。

2 虚拟 GDB 调试技术及其在硬件仿真器中的应用

使用 JTAG Proxy 及 Transactor 调试系统需要编写复杂的测试激励,对于 DUT 上运行的软件调试来说过于复杂。本文结合 JTAG 调试系统开发出虚拟 GDB 调试技术,使硬件仿真器可以脱离 JTAG 仿真器物理硬件,仅通过 Host Server 以及 Emulator 即可进行 GDB 调试。虚拟 GDB 技术通过 VPI (virtual protocol interconnect)接口协议,将 GDB 命令解析转译为 JTAG Proxy 可识别的 JTAG 命令,结合 JTAG Proxy 及 Transactor 形成一整套 GDB 调试系统[16],用于调试硬件仿真器中的 DUT。其在硬件仿真器中的调试系统如图 5 所示。

使用的 OpenOCD 是一个开源调试软件，其与 GDB 调试工具可进行联合工作，支持现有 ARM 处理器、MIPS 以及 Intel FPGA 等核心芯片的调试，也支持 JLINK 等商用的 JTAG 仿真器。OpenOCD 主要包含 3 个模块：GDB 服务模块、Proxy 服务模块、目标模块配置模块。GDB 服务模块用于 GDB 与 OpenOCD 的通信，其会建立一个线程循环监听通信连接，当收到 GDB 发送的 RSP 信息包格式的信息时，提取命令信息并发送给 OpenOCD；执行或者收到返回数据时，将返回数据封装成 RSP 信息包返回给 GDB[17]。Proxy 服务模块是基于调试访问端口(debug access port，DAP)实现的模块，用来将 JTAG 接口数据转译成不同的系统总线事务[18]从而与 JTAG 代理器进行通信。目标模块配置模块根据目标芯片进行匹配设置，进而与目标芯片进行连接配置确认，确认整个系统与对应的 DUT 建立连接。

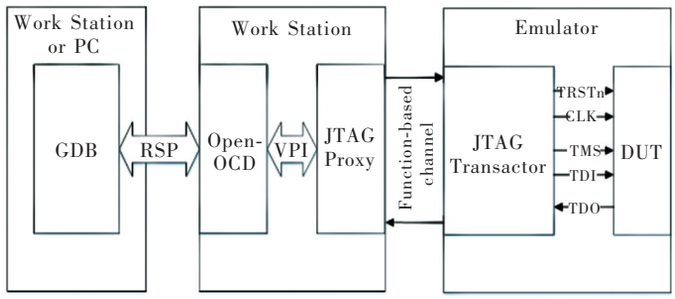

图 5　虚拟 GDB 调试系统

Fig. 5　Virtual GDB debugging system

GDB 服务模块与 Proxy 服务模块使用默认配置建立连接，修改目标模块配置部分，使用 VPI 接口协议对硬件仿真器中的 JTAG Proxy 进行通信适配。配置 JTAG_VPI_PORT 端口为 5555，JTAG_VPI_ADDERSS 地址为 127.0.0.1，通过地址端口建立连接。配置目标芯片为 Cortex-M3，配置 JTAG 时钟为 1 MHz，内存起始地址为 0x200000000，大小为 0x10000，指令位宽为 4，掩码为 0xf。

JTAG Proxy 中增加 Init_jtag_server()函数，用来创建线程循环监听端口连接，端口与 JTAG_VPI_PORT 保持一致。当收到 OpenOCD 发送的数据时，首先将其转译为 Proxy 中可识别的命令存入数据结构中，然后调用 Send_cmd()函数将命令发送给 Transactor，完成命令的发送。再将 Send_data()函数进行适配，将收到的 JTAG 信号封装为 VPI 数据，发送给 Open-OCD，完成数据的返回。

3　测试结果与讨论

本文所用测试用例为开源的 Cortex-M3 内核[19]，经自主研发的硬件仿真器配套编译器编译后写入硬件仿真器。所用硬件仿真器为自主研发的企业级硬件仿真器，采用 IntelS10 系列 FPGA 组成阵列，单颗 FPGA 包含 5 172 480 个 LUT(look-up table)，7 758 720 个 FF(flip-flop)，12 950 个 BRAM(block RAM)，以及 6 912 个 DSP(digital signal processor)。可支持 10 亿 ASIC 门的芯片设计进行硬件仿真验证。

所用服务器 CPU 为 Intel Xeon Platinum8268 @2.9 GHz 128 核，内存 1 TB，在 CentOS 系统上启动软硬件。硬件仿真器以 TBA 模式启动成功后可显示 Porxy 正在监听端口 5555，然后开启 OpenOCD，读取端口连接信息及 DAP 的相关信息。例如：连接端口 127.0.0.1:5555，JTAG tap 为 cm3，器件 TAPID 为 0x4ba00477，如图 6 所示。启动 GDB 后执行 target remote 127.0.0.1:3333，连接成功后执行 load 命令，将

```
[qqhuang@a03 emu]$ openocd -f m3.cfg
***openocd main
Open On-Chip Debugger 0.11.0
Licensed under GNU GPL v2
For bug reports 1, read
        http://openocd.org/doc/doxygen/bugs.html
Info : only one transport option; autoselect 'jtag'
Info : Set server port to 5555
Info : Set server address to 127.0.0.1
cortex_m reset_config sysresetreq

Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
Info : Connection to 127.0.0.1 : 5555 succeed
Info : This adapter doesn't support configurable speed
Info : JTAG tap: cm3.cpu tap/device found: 0x4ba00477 (mfg: 0x23b (ARM Ltd), part: 0xba00, ver: 0x4)
Info : cortex_m_examine ***

Info : Cortex-M3 r2p1 processor detected
Info : cpuid: 0x412fc231
Info : cortex_m_examine FPCTRL 0x00000260***

Info : FPB fpcr 0x260, numcode 6, numlit 2
Info : cm3.cpu: hardware has 6 breakpoints, 4 watchpoints
```

图 6　OpenOCD 与 Proxy 连接成功

Fig. 6　Successful connection between OpenOCD and Proxy

之前编译好的 elf 文件下载到 Cortex-M3 中进行调试。

本次测试对 Cortex-M3 内部寄存器进行赋值并通过单步执行获取每次赋值后的寄存器值，如图 7 所示，寄存器 r0 成功赋值为 1。

该结果表明，整个虚拟 GDB 远程调试系统成功应用于硬件仿真器中，可进行 GDB 调试方法中的单步调试、寄存器值获取等操作，有效增强了硬件仿真器的调试能力。

当使用 ICE 模式运行时，接入 JLINK 仿真器到硬件仿真器的 JTAG 接口，同样可以进行远程调试，OpenOCD 的连接情况如图 8 所示。

使用 Cortex-M3 在硬件仿真器中的编译及运行参数如表 2 所示。由表 2 可知，事务器所消耗的额外硬件资源量仅占单颗 FPGA 的 0.2%，而运行频率可达到 6 MHz，相比于其他硬件仿真器的 1 MHz 提升了 6 倍。

本文还使用插入探针的方法对 DUT 进行调试，通过探针抓取 TCK，TMS，TDI 以及 TDO 信号，得到的信号波形如图 9 所示。

```
14                                      movs r1, #2
(gdb) i r
r0             0x1              1
r1             0x0              0
r2             0x0              0
r3             0x0              0
r4             0x0              0
r5             0x0              0
r6             0x0              0
r7             0x0              0
r8             0x0              0
r9             0x0              0
r10            0x0              0
r11            0x0              0
r12            0x0              0
sp             0xfffffff0       0xfffffff0
lr             0xffffffff       -7
pc             0xa              0xa <start+2>
xPSR           0x1000003        16777219
msp            0xfffffff0       0xfffffff0
psp            0x0              0x0 <_start>
primask        0x0              0
basepri        0x0              0
faultmask      0x0              0
control        0x0              0
(gdb) n
FPB num ** = 0x00000000
15                                      movs r2, #3
```

图 7 GDB 调试运行图

Fig. 7 Debugging figure of GDB

```
qqhuang@ubuntu:~/Desktop$ openocd -f interface/jlink.cfg -f ./stm32f2x.cfg
Open On-Chip Debugger 0.11.0-rc2
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
Info : auto-selecting first available session transport "jtag". To override use 'transport select <
transport>'.
cortex_m reset_config sysresetreq

Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
Info : J-Link V9 compiled Apr 21 2017 11:08:06
Info : Hardware version: 9.30
Info : VTarget = 3.325 V
Info : clock speed 100 kHz
Info : JTAG tap: stm32f2x.cpu tap/device found: 0x4ba00477 (mfg: 0x23b (ARM Ltd), part: 0xba00, ver
: 0x4)
```

图 8 ICE 模式下 OpenOCD 与 JLINK 连接

Fig. 8 Connection of OpenOCD and JLINK in ICE mode

表 2 Cortex-M3 在硬件仿真器中的编译运行性能

Tab. 2 Performance of Cortex-M3 in emulator

项 目	GDB 调试性能	Probe 调试性能
DUT 编译时间/min	20	21
DUT 资源占用	LUT:75397 FF:20421 BRAM:131 DSP:2	LUT:75401 FF:20453 BRAM:131 DSP:2
Transactor 资源占用	LUT:9272 FF:5921	LUT:9272 FF:5921
DUT 运行频率/MHz	6	6
JTAG 运行频率/MHz	1	1
调试时间/s	<2	~180

本次插入 4 个静态探针，探针占用资源需要增加 4 个 LUT 以及 32 个 FF，同时对调试时间进行比对。使用 GDB 调试获取寄存器数据，从启动 GDB 到获取数据用时不超过 2 s；使用探针进行信号抓取，从启动测

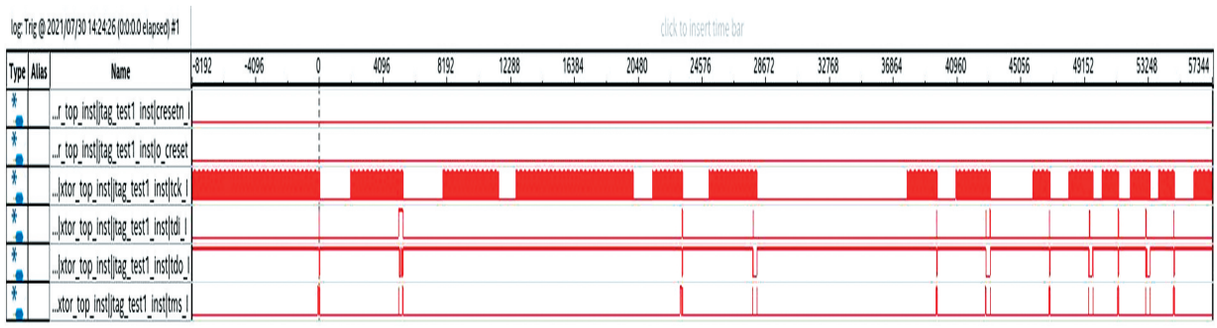

图 9　探针抓取 JTAG 引脚波形图

Fig. 9　Waveform of the JTAG pins by probe

试激励到信号抓取完成需 180 s。可以看到，使用虚拟 GDB 调试系统可以加速调试时间近百倍。这是因为，探针抓取需要将信号引入专用的调试硬件 IP，而后将数据存入存储器进行数据对齐等处理，最后才能显示波形，数据处理和显示需要消耗大量时间。另外，探针抓取的信号数量还受到存储器大小的约束，数据量超过存储器大小时将无法完整地保留所有数据。而使用虚拟 GDB 进行调试时，只需要输入对应的激励就可以即时获取想要的数据，这也体现出虚拟 GDB 调试系统的灵活性。

4　结　语

本文设计和开发了虚拟 GDB 远程调试系统，并成功应用于硬件仿真器中。相比于探针抓取的调试方法，虚拟 GDB 远程调试系统减少了近百倍的调试时间。该系统同时支持硬件仿真器的 ICE 模式和 TBA 模式，丰富了硬件仿真器的软件调试手段，同时还缩短了 DUT 的调试时间，方便硬件仿真器用户进行远程调试，为国产硬件仿真器提供了更加便捷灵活的软件调试方法。

但是，对于软件调试方法来说单纯 GDB 调试仅是基础功能。未来可将虚拟 GDB 远程调试技术与更高层次的集成开发环境（如 eclipse 等）相结合，开发出更为便捷的软件调试工具，方便芯片从业人员基于硬件仿真器对所设计的芯片进行仿真验证。

参考文献/References：

[1] 徐宣哲，宁珂，郑学敏，等. 基于硬件仿真系统的边缘计算人工智能视觉芯片设计验证[J]. 物联网学报，2022，6(1)：20-28.
XU Xuanzhe，NING Ke，ZHENG Xuemin，et al. Verification of an artificial intelligence vision chip design for edge computing based on hardware simulation system[J]. Chinese Journal on Internet of Things，2022，6(1)：20-28.

[2] 郝强. 基于硬件仿真器的 PCIe 接口验证方法的研究和实现[J]. 电子技术应用，2020，46(8)：77-79.
HAO Qiang. Research and implementation of verification method for PCIe interface based on emulator[J]. Application of Electronic Technique，2020，46(8)：77-79.

[3] 王雷，王晨光，吴斌. 基于硬件加速器的高性能芯片仿真与验证[J]. 电子技术应用，2019，45(8)：61-62.
WANG Lei，WANG Chenguang，WU Bin. Debugging a high-performance WLAN chip with palladium XP emulator[J]. Application of Electronic Technique，2019，45(8)：61-62.

[4] BRAHME D S，COX S，GALLO J，et al. The Transaction-based verification methodology[J]. Cadence Berkeley Labs，2000. DOI：10.1109/IEEESTD.2013.6515989.

[5] 徐健，罗天柱. CADENCE 硬件仿真器在 Ethernet 交换芯片验证中的应用[J]. 中国集成电路，2005(9)：76-78.
XU Jian，LUO Tianzhu. Implementation of CADENCE emulator in verification of Ethernet exchange chip[J]. China Integrated Circuit，2005(9)：76-78.

[6] 江霞林，周剑扬，杨银涛，等. 基于 FPGA 的软硬件协同仿真加速技术[J]. 中国集成电路，2010，19(8)：30-33.
JIANG Xialin，ZHOU Jianyang，YANG Yintao，et al. FPGA based accelerator for hardware/software co-simulation[J]. China Integrated Circuit，2010，19(8)：30-33.

[7] 阎芳，李翔，徐双平，等. 基于硬件仿真加速器的 AFDX 端系统测试方法研究[J]. 电子技术与软件工程，2020(1)：76-77.

[8] Mentor Graphics 公司. 基于硬件仿真的软件调试方法[J]. 中国集成电路，2014，23(10)：45-47.

[9] 单祥茹. Palladium Z1 开创数据中心级硬件仿真加速新时代[J]. 电子技术应用，2016，42(1)：12-14.
SHAN Xiangru. Palladium Zi opens new era of data center level hardware simulation acceleration[J]. Application of Electronic Techno-

logy, 2016, 42(1):12-14.
[10] MAUNDER C M, TULLOSS R E, BHAVSAR D K, et al. IEEE Standard Test Access Port and Boundary-Scan Architecture[M]. [S. l.]:[s. n.],1993.
[11] 许琼. 基于 JTAG 的 ARM7TDMI 调试系统[J]. 计算机工程,2008,34(15):252-254.
XU Qiong. ARM7TDMI debugging system based on JTAG[J]. Computer Engineering,2008,34(15):252-254.
[12] SEGGER. Application Note: Analyzing HardFaults on Cortex-M CPUs[EB/OL]. https://www. segger. com/downloads/application-notes,2023-06-10.
[13] STMICROELECTRONICS. ST-LINK/V2 in-circuit Debugger/Programmer for STM8 and STM32 UserManual[EB/OL]. https://www. st. com/resource/en/user _ manual/dm00026748-st-link-v2-in-circuit-debugger-programmer-for-stm8-and-stm32-stmicroelectronics. pdf, 2023-02-14.
[14] Accellera Interfaces Technical Committee. Standard Co-Emulation Modeling Interface[EB/OL]. https://www. eda. org/images/downloads/standards/sce-mi/SCE-MI_v24-Nov2016. pdf,2016-11-06.
[15] 虞致国,魏敬和. 基于 SystemVerilog DPI 的 ARM SoC 虚拟调试验证平台的设计[J]. 微电子学与计算机,2009,26(11):117-123.
YU Zhiguo,WEI Jinghe. Design of virtual verification platform for ARM SoC based on System Verilog DPI[J]. Microelectronics & Computer,2009,26(11):117-123.
[16] GRANT R E,AFSAHI A,BALAJI P. Evaluation of ConnectX virtual protocol interconnect for data centers[C]//2009 15th International Conference on Parallel and Distributed Systems,Shenzhen:IEEE,2009:57-64.
[17] 殷绍剑,雷航,詹瑾瑜. 嵌入式远程调试原理研究与实现[J]. 计算机应用与软件,2014,31(6):240-243.
YIN Shaojian,LEI Hang,ZHAN Jinyu. Study and implementation of the principle of embedded remote debugging[J]. Computer Applications and Software,2014,31(6):240-243.
[18] 徐文昌. SoC 调试跟踪系统的设计与实现[D]. 西安:西安电子科技大学,2012.
XU Wenchang. Design and implementation of SoC debug and trace system[D]. Xi'an:Xidian University,2012.
[19] 令狐绍尉,邓睿,吴加加,等. 基于 Cortex-M3 的 SoC 嵌入式系统架构设计[J]. 移动信息,2022(9):22-24.

向本期载文的审稿专家致谢

本期《河北科技大学学报》共发表论文 12 篇,这些论文的发表是与有关专家的认真审读、细查资料、推敲分析、中肯评价分不开的。对此,本刊编辑部特向这些专家表示敬意,对他们的辛勤劳动表示感谢。

本期载文的审稿专家名单如下(按姓名的汉语拼音字母顺序排列):

陈书旺　崔春生　董　非　古双喜　顾泽茂　黄　伟
黄永华　简家文　李　瑞　林正喆　刘京雷　刘　晶
刘晓光　门进杰　王二威　王海军　王天友　王新武
谢志辉　杨　磊　张卫正　张永才　赵　旸　周丽娜
朱华结

(本刊编辑部)