文章编号:1674-2974(2024)06-0119-09 DOI:10.16339/j.cnki.hdxbzkb.2024272

一种面向异构片上系统的调试器框架

唐俊龙[1],李奕成[1],邹望辉[1],时洋[2†]
(1. 长沙理工大学 物理与电子科学学院,湖南 长沙 410114;
2. 国防科技大学 计算机学院,湖南 长沙 410073)

摘 要:异构片上系统具有定制化满足应用的特异性需求特点,成为多个领域内的主流解决方案.但用户在异构片上系统上进行开发需要面对多种计算资源带来的程序错误,而不同异构片上系统构建统一的调试器框架也面临巨大的挑战.针对上述问题,本文提出了一种面向异构片上系统的调试器框架,该调试器框架设计了异构处理器调试器框架通用接口,开发者可以通过框架功能接口快速构建异构调试器.该框架功能丰富,通过线程切换实现了对异构多核程序的调试,也实现了异构程序性能分析等功能.该框架生成的调试器与传统硬件调试器相比,加载异构程序速度快,是读内存速率的5.5倍,是写内存速率的16.5倍,调试速度大大提高.

关键词:异构片上系统;调试器;异构调试;多核调试;性能分析
中图分类号:TP311 **文献标志码**:A

A Debugger Framework for Heterogeneous System on Chip

TANG Junlong[1], LI Yicheng[1], ZOU Wanghui[1], SHI Yang[2†]
(1. School of Physics & Electronic Science, Changsha University of Science & Technology, Changsha 410114, China;
2. College of Computer Science and Technology, National University of Defense Technology, Changsha 410073, China)

Abstract: The heterogeneous system on chip has the characteristics of customization to meet the specific requirements of applications and has become the mainstream solution in many fields. However, users need to face program errors brought by various computing resources when developing on heterogeneous system on chip, and it is also a great challenge to build a unified debugger framework for different heterogeneous system on chip. To solve the above problems, a debugging framework for heterogeneous system on chip is proposed in this paper. A general interface of the debugging framework for heterogeneous processor is designed in this framework, which enables developers to quickly build heterogeneous debuggers through the framework functional interfaces. The debugger framework is rich in functions. It realizes debugging of heterogeneous multicore programs through thread switching and performances analysis of heterogeneous programs. Compared with traditional hardware debugging, the debugger generated by the framework loads heterogeneous programs faster, which is 5.5 times the read memory rate and 16.5

* **收稿日期**:2023-10-16
基金项目:柔性电子材料基因工程湖南省重点实验室开放基金(202015),Open Research Fund of Hunan Provincial Key Laboratory of Flexible Electronic Materials Genome Engineering(202015);湖南省自然科学基金(2023JJ40679), Natural Science Foundation of Hunan Province of China(2023JJ40679)
作者简介:唐俊龙(1973—),男,湖南武冈人,长沙理工大学副教授,博士
† 通信联系人,E-mail: shiyang14@nudt.edu.cn

times the write memory efficiency, and the debugging speed is greatly improved.

Key words: heterogeneous system on chip; debugger; heterogeneous debugging; multicore debugging; performance analysis

随着半导体产业的迅速发展,集成电路(Integrated Circuit,IC)将新型技术集成到一个芯片上,即片上系统(System on Chip,SoC),不同单核SoC芯片应用领域不同,例如体积小、功耗低、低成本、高性能、应用于处理器系统领域的ARM(Advanced Risc Machines),以及运行速度快、数字信号处理能力强、应用于图像处理领域的DSP(Digital Signal Processor)[1]等.但是单核SoC结构性能单一,不能满足日益增长的性能需求,多核SoC技术应运而生.不同处理器为了满足应用的差异化、定制化需求在SoC上集成多种核,加大了对异构多核处理器[2-3]的开发难度.而不同SoC的处理核来自不同厂商,例如TI公司的达芬奇[4]平台的DM6000系列,集成了ARM9和DSP核,Xilinx公司的Zynq7000系列集成了双核Cortex-A9和FPGA核等.而多核SoC芯片需要硬件平台提供多核调试硬件支持[5],不同的异构SoC提供的调试工具只能根据硬件支持来开发对应平台的调试工具.若没有异构调试工具,进行异构调试时,则需要对现有的异构程序进行拆解,并且封装到各自不同的处理核上可执行的程序,并对所有运行的处理核进行调试分析.这大大增加了异构程序设计员的工作量.

针对上述异构片上系统多核处理器的调试器研究现状和现有方案存在的问题,本文设计了一种调试器框架DOGS(Debugger for Heterogenous SoC),该调试器框架已在FT-Matrix系列的异构多核处理器实现调试验证.该调试器框架不仅具有基本的调试功能,提供异构程序调试功能,以及多核调试功能,还具有profile性能测试功能.调试器框架DOGS设计生成的调试器目前已在16个CPU核和4个GPDSP_Cluster(DSP簇)[6]构成的FT-Matrix异构片上系统高性能芯片以及同系列的异构嵌入式芯片实现测试验证.本文的主要贡献如下.

1) DOGS框架设计,设计内存读写接口以实现操作系统异构统一调试,提供了高效可移植的可拓展功能接口.

2) 针对FT-Matrix的DOGS框架调试器实例化,展示其可行性以及实现流程.

3) 基于DOGS框架调试器的性能测试,验证了DOGS框架调试器的基本调试功能、多核调试功能、性能测试功能.

4) 基于DOGS框架生成的调试器与传统的JTAG硬件调试对比,加载异构调试程序,程序内存读写速度提升数倍,性能测试简便.

1 调试方式

目前,市面上异构SoC运用调试方式分为硬件调试方式和软件调试方式.常见的硬件调试方案有JTAG的边界扫描技术,ARM公司的CoreSight架构,风河公司的Workbench调试方案等几种;而JTAG(Joint Test Action Group)以及CoreSight架构属于片上调试技术(OCD)[7].软件调试方案有GDB(GNU Symbolic Debugger)、LLDB(Low Level Debugger)等.

1.1 硬件调试方式

JTAG[8-9]边界扫描技术具有良好的可见性和可控制性,是当前硬件调试的重要手段.其具有代表性的菊花链边界扫描技术,是将各个核的TDI(Test-Data Input)和TDO(Test Data Output)信号串联,共享TMS(Test Mode Selector),TCK(Test Clock)的串行调试方法.但是随着芯片SoC的不断发展,JTAG边界扫描技术也显示出相应的局限性.JTAG边界扫描需要CPU控制芯片时钟信号,导致扫描时间过长.并且设计的调试系统要与芯片设计结构紧密结合.若芯片进行更换或者修改,则必须重新设计相应的JTAG扫描链,不便于调试功能移植拓展.

ARM公司的CoreSight[10]架构,是用于对复杂SoC实现debug和trace的架构.CoreSight架构包括trace,debug,trigger三个通路.trace通路将ARM核以及DSP核部信息输出到外部.debug通路实现对DSP以及ARM核进行调试.trigger通路用于ARM核与DSP核之间的信号传递.ARM公司的CoreSight架构是基于总线的设计,不能简便有效地移植到其他SoC芯片平台上.

Workbench调试方案通过集中化,实现多核的调试功能,支持多系统、多线程和多处理器的应用环境[11].开发人员可以在一个或者多个内核中设置观测点,通过控制芯片内核来实现多核调试.Workbench片上调试技术可以配合JTAG扫描技术,提高调试定位异常的效率以及准确率.

1.2 软件调试方式

GDB是GNU工具集中的调试器[12],是一款纯软件开发的调试软件.同GNU工具集中的GCC配套组成一套完整的开发工具,GDB是UNIX/Linux系统中强大的调试工具,GDB源码程序开放、代码简便、是主流的调试框架,它不仅能够调试软件并分析软件的执行过程,帮助开发者调查程序的正确行为,而且能用来分析程序崩溃的原因.

LLDB是LLVM(Lower Level Virtual Machine)工具集中的一个高性能调试器[13-14],同GDB一样属于纯软件开发的调试器.其充分利用LLVMProject中现有库中的相关函数来构建相关调试程序.LLDB是macOS上Xcode的默认调试器,支持在IOS设备、模拟器以及台式器上调试C、C++、Objective-C等程序.

在嵌入式芯片处理器用户调试工具方面,ARM和Siffiv公司都配备了相应的上位机或兼容主流开源的IDE,如Keil和Ecilpse等,IDE与配套的调试器相结合,提供了寄存器以及内存观测读写、程序加载、断点、单步执行等远程调试功能[15].

硬件调试方式是大部分异构片上系统调试方式,只需异构片上系统具有硬件调试接口即可实现.但是硬件调试需要配套的硬件接口,加载调试程序到特定的调试核上进行调试,调试时间过长.软件调试方式对比硬件调试方式,针对不同的异构片上系统移植简单方便,并且具有良好的功能拓展性.本文提出的DOGS框架属于软件调试方式.该框架继承了软件调试方式移植、简便、调试功能易拓展的优点.不需要异构片上系统特定的硬件调试接口,即可加载调试程序到调试核,加载异构程序比传统硬件调试方式准确,并且读写内存速度快.同时DOGS框架设计开发了多核异构调试以及异构程序性能测试功能.

2 DOGS调试框架

2.1 调试框架设计

DOGS框架设计总体结构如图1所示.以GDB软件框架为主体,进行功能拓展,实现异构片上系统设备文件的读写访问.以此设计出断点调试接口、多核调试接口、profile功能接口.

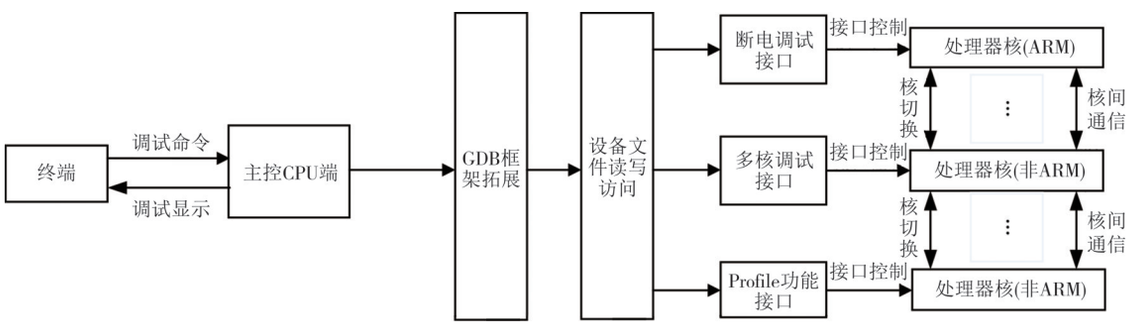


图1　DOGS调试框架图

Fig.1　DOGS Debugging frame diagram

2.2 内存设备读写访问模块

异构片上系统集成一个CPU操作系统(Linux系统)进行资源访问.异构片上系统通过Linux内核驱动函数,将片上系统抽象成设备文件资源,通过Linux内核驱动函数进行设备文件资源的管理访问.Linux操作系统中ioctl是Linux内核中设备驱动程序控制接口函数,通过增设ioctl命令方式来拓展新的功能.ioctl(int fd, int cmd, …)函数具有三个参数,fd是文件描述符,cmd是交互协议.第三个参数(…)是可变参数arg. fd传输设备文件信息,cmd控制码实现驱动控制.arg参数对驱动程序进行传参.以结构体指针方式传递异构片上系统处理核的相关信息.在Linux操作系统中,每个设备都对应一个文件,在内核中具有一个相应的索引节点.通过索引节点对文件进行系统调用.ioctl命令中的fd通过Linux内核找到对应的Linux内核设备文件索引,将设备文件指针传递给驱动函数.

DOGS框架内存读写模块为用户添加多种ioctl驱动函数,实现CPU端对处理器核资源访问控制,实现异构片上系统中核与核之间资源的调度分配.

DOGS框架的内存读写模块功能如图2所示,通过内存读写模块添加ioctl驱动函数设计出断点调试接口、多核调试接口、以及profile功能接口.DOGS框架通过ioctl驱动接口实现了操作系统的异构统一调试.

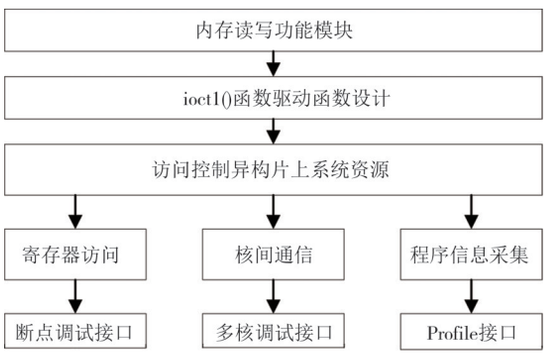


图2 异构片上资源控制图

Fig .2 Heterogeneous on-chip resource control diagram

2.3 断点设置与取消接口设计

DOGS断点功能是将异构程序运行到断点位置,之后异构程序中断执行,进入调试中断,所以断点功能是调试中重要功能,其可以跟踪异构程序的状态.DOGS断点设置接口设计如下.

1)首先创建一个软件断点链表,收集设置的软件断点.

2)扫描当前异构程序,获取当前正在运行的处理器核.

3)判断断点在断点链表中是否存在,若断点链表中没有当前设置的断点,将断点地址以及指令相关信息写入断点链表中.

4)若断点已经写入断点链表中,利用内存读写模块对当前断点再次进行断点指令写入操作.

对不需要进行观测的程序断点,对断点进行删除,再次进行调试时,异构程序则不会在取消的断点处进行程序中断.DOGS断点取消接口设计如下.

1)首先判断取消的断点在断点链表中是否存在.

2)若存在相应的断点,将设置断点的目标指令恢复,并获取指令以及断点原来存储的地址等相关信息,把恢复的指令写回到相应的地址.

3)然后在软件断点链表中将目标指令删除.

4)如果软件断点链表中无这个软件断点,则终端端口提示错误.

2.4 多核调试接口

异构多核程序的硬件调试方式需要对每个核进行观测点设置,而DOGS框架多核调试接口将多核调试转变为多线程调试.将多个核的观测点设置到调试线程上,分批进行异构线程调试,以多线程调试实现异构多核调试.DOGS多核调试具体接口设计如下所示.

1)当前异构程序需要调用多核资源时,DOGS框架捕获所有正在运行的处理核信息.

2)为当前运行的多个处理器核创建调试线程,每个处理核对应一个调试线程,通过内存设备文件读写模块控制对应的处理核,调试各个处理器核.

3)通过切换调试线程来实现对处理器核程序切换,实现多核调试.

2.5 profile功能接口

profile功能接口通过栈帧信息获取处理器核程序正在运行函数的调用关系、函数的占用资源比例、以及函数对整体程序运行的耗时比例.程序员以此优化异构程序中的函数,提高处理器核程序的优化效率.

调用Linux系统中fork(·)函数创建出父子进程,在一定频率的时钟信号的影响下,每隔一段时间父进程和子进程进行信号量交互.父进程对子进程发出获取调用函数栈帧信息信号时,也会进入死循环等待子进程返回信号.当获取到父进程发出的获取调试栈帧信号时,子程序开始不断获取对应的调试程序的函数调试栈帧信息,进行栈帧消息记录.获取栈帧消息后,子进程对父进程发出跳出循环信号,父进程发出暂停获取调试信息信号给子进程.子进程退出获取调试信息,调试程序继续执行,当子进程获取被调程序退出信号时,profile功能将调试程序函数信息进行显示,并清除被调函数程序调用列表.

2.6 DOGS框架移植

DOGS框架是在GDB源码的基础上进行修改调试实现的.DOGS框架移植的基本流程与GDB软件移植一致.DOGS移植流程步骤如下.

1)在bfd文件夹中设计一个针对异构片上系统目标体系架构的二进制可执行(BFD)文件.

2)在opcodes文件夹下设计一个异构片上系统目标体系架构下可执行的一个反汇编器.

3)修改GDB后端源码文件将DOGS框架接口功能进行写入,使其具有DOGS框架具有的调试功能以及多核调试功能、profile功能.

3 基于FT-Matrix芯片DOGS框架实现

DOGS框架在基于以FT-Matrix处理器为代表的

E级异构片上系统体系架构的高性能芯片进行了实例化,生成了FT-Matrix-gdb.该调试器实现了对FT-Matrix处理器系统上基本调试功能以及多核调试、性能测试等多种功能拓展.

3.1 FT-Matrix芯片架构

FT-Matrix系列异构SoC是一款面向E级计算[16]的自主研发的异构多核DSP处理器.FT-Matrix处理器DSP内核大多采用了基于超长指令字(Very Long Instruction Word,VLIW)的标量/向量协同处理器架构,由标量处理单元(Scalar Processing Unit,SPU),向量处理单元(Vector Processing Unit,VPU),DMA传输部件以及仿真调试部件(ET)所组成.其中SPU负责标量计算以及程序流程控制,VPU负责向量计算,VPE用于支持向量定点和浮点运算.DMA部件接收SPU配置的传输参数对指定存储资源的访问.仿真调试部件(ET)接收外部主机或者外部JTAG的调试请求,实现DSP内核调试访问.

3.2 ET仿真调试模块支持

FT-Matrix的调试支持由DSP核内部仿真调试部件提供.通过JTAG控制器或者CPU来对DSP核进行调试.FT-Matrix调试模块通过接收调试指令的coreID标记的调试的DSP核,来定位进行调试的DSP核.

FT-Matrix异构SoC对DSP核调试通过DSP核内的ET仿真调试模块进行访问控制,DSP核内ET仿真调试模块如图3所示.ET模块访问芯片数据空间的读写访问,对芯片配置寄存器进行访问,FT-Matirx处理器对其空间进行访问,以此获得表征的用户程序性能.

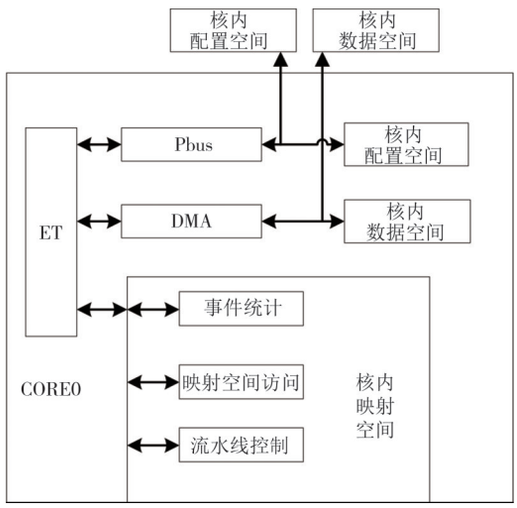


图3　ET仿真模块图

Fig.3　ET simulation module diagram

DOGS框架设计的内存读写模块控制FT-Matrix处理器的ET仿真模块实现对处理器DSP核内资源访问以及控制.ET模块通过直接存储方式(DMA)对核内数据空间访问.DOGS框架以FT-Matrix系列异构片上系统的ET模块设计了et_write( ),et_read( )等ET函数,调用内存读写模块ioctl驱动函数实现CPU对设备DSP核数据的读写.实现对DSP核内资源控制.DOGS框架接口设计ET模块如图4所示.

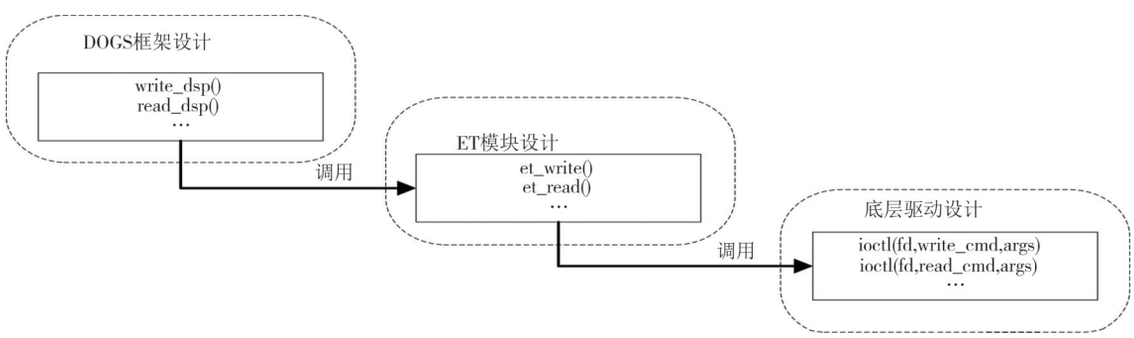


图4　ET模块函数关系图

Fig.4　ET module function diagram

3.3 FT-Matrix芯片软件断点设置与取消实现

FT-Matrix-gdb在FT-Matrix处理器调试异构程序时,通过DOGS断点设置接口实现断点设置,具体流程实现如算法1所示.

由于FT-Matrix处理器指令命令长度不一,需要判断断点地址的指令长度(算法1第4行).将断点指令位以及断点地址写入断点链表中(算法1第5行).不同的指令进行断点指令位修改,生成新的断点指令(算法1第6行到第10行),通过FT-Matrix处理器ET模块写回至处理器核的目标指令加载处,实现断点的设置功能.

FT-Matrix-gdb在FT-Matrix处理器上取消断点时,通过DOGS断点取消接口实现,断点取消具体流程实现如算法2所示.

算法1:DOGS断点设置

输入:处理核core_id,断点地址addr
输出:断点信息设置
1. bp=bp_addr_exit(addr);
2. **if** bp is not exit **then**
3. bp = create_bp(addr);
4. bit = get_bp_bit(addr);
5. add_bp(bp);
6. **if** bit is exit **then**
7. bp_handle_fst(bp);
8. **else**
9. bp_handle_sec(bp);
10. **end**
11. **else**
12. instrNew = bp→instrNew;
13. **end**
14. et_write(core_id,et_addr,instrNew);

调用DOGS断点取消接口,在FT-Matrix处理器实现断点取消功能.从断点链表中判断取消的断点地址是否存在,若取消的断点在断点链表中存在,从断点链表中获取指令信息,进行指令的恢复(算法2第6行到第10行).然后将断点恢复的指令地址通过ET模块写回到目标指令加载处.然后对断点地址进行删除.若取消的断点在断点链表中不存在则发出错误信息.

算法2:DOGS断点取消

输入:处理核core_id,断点地址addr
输出:断点信息取消
1. bp=bp_addr_exit(addr);
2. **if** bp is not exit **then**
3. print_error( );
4. **else**
5. bit = get_bp_bit(bp);
6. **if** bit is exit **then**
7. instrbit = bp_rsu_fst(bp);
8. **else**
9. instrbit = bp_rsu_sec(bp);
10. **end**
11. remove_bp(addr);
12. **end**
13. et_write(core_id,et_addr,instrbit);

3.4 FT-Matrix芯片多核调试功能实现

FT-Matrix-gdb在FT-Matrix处理器调试异构多核程序时,多核程序开始运行时FT-Matrix处理器在每个DSP运行核上设置一个扫描位.FT-Matrix-gdb开始调试多核程序时,扫描FT-Matrix处理器所有DSP核,FT-Matrix调试器获取正在运行的DSP核,并加载DSP核函数信息,根据运行DSP核创建线程组(算法3第4行),把多核调试转变为多线程调试.每个运行核上分配一个调试线程.通过线程切换命令进行多线程程序的切换(算法3第9行)实现多核调试.多核调试功能流程实现如算法3所示.

算法3:DOGS多核调试

输入:运行核run_core
输出:调试核程序加载,核程序切换
1. Initialization;
2. **for** not end of all_cores **do**
3. **if** run_core is scaned **then**
4. set_thread_debug_array(run_core);
5. **end**
6. **end**
7. ……
8. **for** run_core in debug_array **do**
9. thread_switch(run_core);
10. **end**

3.5 FT-Matrix芯片profile功能实现

FT-Matrix-gdb开始profile功能时,获取当前调试程序的进程号,调用fork( )函数创建父子进程.若进入子进程调试,子进程利用sigaction( )函数设置两个自定义信号child_SIGUSR1,child_SIGUSR2,表示父进程对子进程的控制(算法4第4行).若进入父进程调试,父进程初始化(算法4第7行),创建函数关系链表(算法4第8行),sigaction( )函数设置prof_handle( )时钟信号函数(算法4第9行),设置两个自定义信号parent_SIGUSR1,parent_SIGUSR2(算法4第11行和第12行),子进程对父进程发送两个自定义信号表示子进程对父进程的反馈.profile信号设置伪代码如算法4所示.

算法4:profile信号设置

输入:调试子进程child_pid
输出:profile信号流
1. Initialization;
2. fork( );
3. **if** child_pid == 0 **then**
4. sigaction(child_SIGUSR1,parent_to_child1( ));
5. sigaction(child_SIGUSR2,parent_to_child2( ));
6. **else**
7. initprof( );
8. createlist( );
9. sigaction(timeCLK,prof_handle( ));
10. sigaction(parent_SIGUSR1,child_to_parent1( ));
11. sigaction(parent_SIGUSR2,child_to_parent2( ));
12. **end**

profile 流程父进程伪代码如算法 5 所示.设置时钟周期函数 setimer( )(算法 5 第 2 行),每隔一段时钟周期 cpu 对父进程发送时钟信号 sigqueue(timeCLK)(算法 5 第 3 行),调用 prof_handle( )函数.对子进程发送 child_SIGUSR1 信号(算法 5 第 4 行),父进程进入死循环,开始等待 parent_SIGUSR2 信号.父进程接收到 parent_SIGUSR2 信号时,跳出死循环,函数关系调用表接收子进程调试程序函数数据(算法 5 第 8 行),对子进程发送 child_SIGUSR2,等待子进程发送 parent_SIGUSR1 信号(算法 5 第 9 行),父进程接收到 parent_SIGUSR1 信号时,显示函数关系调用列表显示当前调试程序的函数占用比和消耗时间(算法 5 第 11 行).父进程将 parent_SIGUSR1 以及 parent_SIGUSR2 信号清除(算法 5 第 12 行)并退出.

算法 5:profile 父进程伪代码

```
输入:调试父进程 parent_process,时间信号 timeCLK
输出:child_SIGUSR1 信号,child_SIGUSR2 信号,显示调试程序函数性能信息
1. IN parent_process;
2. setimer( );
3. sigqueue(timeCLK);
4. sigqueue(child_SIGUSR1);
5. while not get parent_SIGUSR2 do
6. end
7. getinfotolist( );
8. sigqueue(child_SIGUSR2);
9. while get parent_SIGUSR1 do
10.   displayinfo( );
11. deletesigaction(parent);
10. exit_process( );
13. end
```

profile 流程子进程伪代码如算法 6 所示.子进程在调试时,接收到 child_SIGUSR1 信号后,获取当前 DSP 核运行程序的 PC 信号,暂停当前 DSP 核进程,获取当前的栈帧信息(算法 6 第 3 行),利用 Hash 表方式进行存储栈帧信息.子进程暂停(算法 6 第 6 行)等待获取 child_SIGUSR2 信号,子进程继续执行调试程序命令(算法 6 第 8 行),即 profile 继续获取调试程序信息.当调试程序到运行结束时,PC 信号被设置为 exit,子进程发送 parent_SIGUSR1 信号给父进程.子进程把所有 child_SIGUSR1,child_SIGUSR2 信号清除(算法 6 第 11 行)并退出.

算法 6:profile 子进程伪代码

```
输入:child_SIGUSR1 信号,child_SIGUSR2 信号
输出:parent_SIGSUSR1 信号,parent_SIGUSR2 信号,删除调试子进程信息
1. IN child_process;
2. while get child_SIGUSR1 do
3. getinfo( );
4. sigqueue(parent_SIGUSR2);
5. end
6. halt( );
7. while get child_SIGUSR2 do
8. continue( );
9. end
10. while pc is exited do
11. sigqueue(parent_SIGUSR1);
12. deletesigaction(child);
13. end
```

4　DOGS 框架测试与评测

4.1　实验配置

DOGS 调试器框架在 FT-Matrix 系列中面向 E 级计算的异构多核处理器高性能芯片和同系列下的异构多核处理器小型嵌入式芯片上进行调试测试.DOGS 调试器框架实例化生成的 FT-Matrix-gdb,以国防科技大学微电子所编写的异构调试程序测试集进行测试,从以下三个方面对 DOGS 框架进行评估.

(1) FT-Matrix-gdb 支持的调试功能丰富度.

(2) FT-Matrix-gdb 调试功能测试.

(3) FT-Matrix-gdb 与传统硬件 JTAG 调试加载程序以及调试速度对比.

4.2　基本调试功能测试

FT-Matrix-gdb 调试功能的丰富度以 FT-Matrix-gdb 具有的调试功能为例,经异构程序调试测试,Ft-Matrix 调试功能列表如表 1 所示.

FT-Matrix-gdb 异构程序单步执行,断点设置,程序继续执行等功能如图 5 所示.异构测试程序 main 函数上打断点(b main),程序继续执行(c 命令),调试程序执行到 main 断点处,程序单步跟踪(n 命令),进入下一条指令(b = b_sum(a)).程序单步跟踪进入命令(s 命令),进入 b_sum()函数中.程序继续执行命令,测试程序退出.

表1 调试功能列表

Tab.1 Debugging function list

指令类型	说明	支持的DOGS命令
断点类	支持软件断点、硬件断点和临时断点	break;delete;watch等
变量类	支持对全局变量和局部变量的读写	print;display;set;info local;info args等
寄存器类	支持寄存器访问	info register等
切换类	支持核与核切换,调试模式切换	thread id;opencore id;sethmode args等
基本命令类	软件调试控制的基本命令	step;next;continue;list等
性能测试类	异构程序函数性能测试	profile 等

```
(FT-Matrix-gdb) b main
Breakpoint 3 at 0x3efe: file main.c, line 39.
(FT-Matrix-gdb) c
Continuing.

Breakpoint 3, main () at main.c: 39
39                  a = f1();
(FT-Matrix-gdb) n
40                  b = b_sum(a);
(FT-Matrix-gdb) s
b = b_sum(a-1 000) at main.c: 16
16                  int b = 0;
(FT-Matrix-gdb) c
Continuing.

Breakpoint 2, 0x0000000000029c0 in exit ()
```

图5 基本功能测试图

Fig.5 Basic functional test diagram

4.3 异构多核调试功能测试

FT-Matrix-gdb调试异构多核程序时会自动检测是否是异构多核程序.测试如图6所示.图6中显示6个核的异构多核程序运行.显示thread命令切换到线程2,也就是从核1切换到核2.

```
(FT-Matrix-gdb) info thread
Id    Target Id          Frame
6     core 5(opened)     0x000000880000000 in c1_start ()
5     core 4(opened)     0x000000880000000 in c1_start ()
4     core 3(opened)     0x000000880000000 in c1_start ()
3     core 2(opened)     0x000000880000000 in c1_start ()
2     core 1(opened)     0x000000880000000 in c1_start ()
1     core 0(opened)     0x000000880000000 in c1_start ()

(FT-Matrix-gdb) thread 2
[Switching to thread 2 (core 1(opened))]
#0    0x000000880000000 in c1_start ()
```

图6 异构多核切换显示图

Fig.6 Heterogeneous multicore switching display diagram

4.4 profile功能测试

FT-Matrix-gdb的profile功能是FT-Matrix系列异构多核处理器测试异构程序功能.profile功能显示当前调试程序的运行时间占用比和程序函数调用关系.以FT-Matrix系列异构处理器自身的定时器计算功能计算异构调试程序函数占用比,与FT-Matrix-gdb的profile功能得出的测试结果进行对比.表2显示10^4次、10^5次与10^6次循环程序测试对比.其中A函数是循环阶乘函数,B函数是循环递增函数,C函数包含循环递增函数和循环阶乘函数.表2显示,在循环次数少的测试中,A函数与C函数的误差相对较大.产生该误差的原因是这两个函数相比于B函数较为复杂.在测试过程中,FT-Matrix-gdb程序本身的运行被profile功能捕获,循环次数越少其捕获程序本身函数的栈帧信号命中率也会受到影响,因此产生误差.在循环次数少的测试中,测试函数的误差较大.随着异构测试程序循环次数的增多,测试相关函数误差也越来越小.profile功能对大型异构程序测试结果具有可信性.

表2 Matrix测试与profile测试对比

Tab.2 Matrix test compared with profile test

10^4 circulate	Matrix-test	profile-test	Error rate
A Function	25.50%	28.57%	3.07%
B Function	19.43%	19.04%	0.40%
C Function	55.07%	52.38%	2.68%
10^5 circulate	Matrix-test	profile-test	Error rate
A Function	24.18%	25.87%	1.69%
B Function	19.76%	19.40%	0.36%
C Function	56.04%	54.72%	1.32%
10^6 circulate	Matrix-test	profile-test	Error rate
A Function	24.10%	24.29%	0.19%
B Function	19.79%	19.69%	0.10%
C Function	56.11%	56.02%	0.09%

4.5 FT-Matrix-gdb与JTAG调试对比

FT-Matrix-gdb与配套的JTAG硬件调试对同一个1.6 Mb大小的异构调试程序进行调试速度测试.测试结果如表3所示.

根据表3对比得出FT-Matrix-gdb的加载速率是JTAG硬件调试加载速率的5 000倍,加载程序速度快.FT-Matrix-gdb读内存速率是JTAG的5.5倍,约是JTAG写内存速率的16.5倍的写入速率,读写内存速度更快.相比于JTAG这种硬件调试,DOGS调试器框架生成的调试器提高了对异构程序的调试速率.

表 3　JTAG 调试与 FT-Matrix-gdb 测试对比

Tab.3　Comparison of JTAG debugging and FT-Matrix-gdb testing

项目	IDE(仿真器 30 MHz)	FT-Matrix-gdb
加载时间	18 s	0.003 3 s
读内存	712 kB/s	3 916.71 kB/s
写内存	203 kB/s	3 368.98 kB/s

5　总结与展望

本文提出 DOGS 框架——一种针对异构片上系统的调试器框架. 该 DOGS 框架通过操作系统实现异构统一调试,实现基本的调试功能,通过线程切换实现异构程序的多核调试功能,而且实现了性能分析功能.生成的调试器与 JTAG 硬件调试进行对比,加载异构程序速度快,是其读内存速率的 5.5 倍,是其写内存效率的 16.5 倍,调试速度大大提高. 在 FT-Matrix 异构多核处理器的体系结构特征的基础上,成功设计并实现了一款面向 FT-Matrix 异构片上系统的异构调试器.该异构程序调试器兼容 FT-Matrix 系列产品以及 ARM 端的调试器,并且兼容 FT-Matrix 系列的 IDE 产品.

未来针对用户需求,我们将对该调试器框架的以下方面进行进一步研究: 1)该异构调试器框架功能接口拓展的 profile 功能是否能继续完善.2)本文只针对了 FT-Matrix 系列处理器架构 ARM+DSP 异构系统,未考虑其他处理器架构例如 ARM+FPGA 核异构系统.未来的工作将围绕不同的异构片上系统的调试器框架移植以及进一步完善 DOGS 框架功能接口开展.

参考文献

[1] EYRE J, BIER J. The evolution of DSP processors[J]. IEEE Signal Processing Magazine, 2000, 17(2): 43-51.

[2] CHANG S S, ZHAO X F, LIU Z Y, et al. Real-Time scheduling and analysis of parallel tasks on heterogeneous multi-cores[J]. Journal of Systems Architecture, 2020, 105: 101704.

[3] VENU B. Multi-core processors-an overview[J]. 2011. arXiv prepriut arXiv: 1110.3535, 2011.

[4] 庞泽峰,刘增力. 基于无线自组网的边境视频监控系统[J]. 计算机与数字工程,2018,46(5):956-961.
PANG Z F, LIU Z L. Video surveillance system based on the wireless ad-hoc network[J]. Computer & Digital Engineering, 2018, 46(5): 956-961. (in Chinese)

[5] 李鹏程. 面向嵌入式系统的多核调试工具研究与实现[D]. 成都:电子科技大学, 2017.
LI P C. Research and Implementation of Multi-core Debugging Tool for Embedded System [D]. Chengdu: University of Electronic Science and Technology of China, 2017. (in Chinese)

[6] LU K, WANG Y H, GUO Y, et al. MT-3000: a heterogeneous multi-zone processor for HPC[J]. CCF Transactions on High Performance Computing, 2022, 4(2): 150-164.

[7] MAIER K D. On-chip debug support for embedded Systems-on-Chip[C]//Proceedings of the 2003 International Symposium on Circuits and Systems, 2003. ISCAS '03. Bangkok, Thailand. IEEE, 2003: V.

[8] 常志恒,肖铁军,史顺波. 基于 JTAG 的片上调试器与调试系统的设计实现[J]. 计算机工程与应用,2012,48(30):78-82.
CHANG Z H, XIAO T J, SHI S B. Design of on-chip-debugger and debug system based on JTAG[J]. Computer Engineering and Applications, 2012, 48(30): 78-82. (in Chinese)

[9] 陈硕. 基于 USB 和 JTAG 接口的 DSP 在线调试系统实现[D]. 西安:西安电子科技大学,2017.
CHEN S. Realization of DSP Online Debugging System Based on USB and JTAG Interface[D]. Xi'an: Xidian University, 2017. (in Chinese)

[10] GHOSH P, SINHA K. A framework for evaluation of debug path performance in SoC[C]//2021 IEEE 34th International System-on-Chip Conference (SOCC). Las Vegas, NV, USA. IEEE, 2021: 188-193.

[11] 韩青. 多核环境中的高效率调试方法[J]. 今日电子, 2007(7): 70-72.
HAN Q. Efficient debugging method in multi-core environment[J]. Electronic Products, 2007(7): 70-72. (in Chinese)

[12] STALLMAN R, PESCH R, SHEBS S. Debugging with GDB[EB/OL]. Free Software Foundation, 1988.

[13] SAVIDIS A, TSIATSIANAS V. Implementation of live Reverse debugging in LLDB[J]. 2021.arXiv prepriut arXiv: 2015.12819, 2021.

[14] KEBIANYOR B, ITTERSHAGEN P, GRÜTTNER K. Towards stateflow model aware debugging with LLDB[C]//Proceedings of the Rapid Simulation and Performance Evaluation: Methods and Tools. Valencia Spain. ACM, 2019: 1-8.

[15] 钱思园, 王琼, 李暾. 嵌入式软件跟踪调试技术的研究与设计[J]. 单片机与嵌入式系统应用,2012,12(2):1-4.
QIAN S Y, WANG Q, LI T. Research and design of trace debugging technology for embedded software[J]. Microcontrollers & Embedded Systems, 2012, 12(2): 1-4. (in Chinese)

[16] 刘胜, 卢凯, 郭阳, 等. 一种自主设计的面向 E 级高性能计算的异构融合加速器[J]. 计算机研究与发展, 2021, 58(6): 1234-1237.
LIU S, LU K, GUO Y, et al. A self-designed heterogeneous accelerator for exascale high performance computing[J]. Journal of Computer Research and Development, 2021, 58(6): 1234-1237. (in Chinese)