

**研究生学位论文开题报告**

**报告题目 基于SPARC模拟器的多核调试技术研究与实现**

**学生姓名 李戈 学号 2014E8013261153**

**指导教师 唐志敏 职称 研究员**

**学位类别 工程硕士**

**学科专业 计算机技术**

**研究方向 处理器结构**

**培养单位 中国科学院计算技术研究所**

**填表日期 2016.08.15**

**中国科学院大学制**

**填 表 说 明**

1. 本表内容须真实、完整、准确。
2. “学位类别”名称填写：哲学博士、教育学博士、理学博士、工学博士、农学博士、医学博士、管理学博士，哲学硕士、经济学硕士、法学硕士、教育学硕士、文学硕士、理学硕士、工学硕士、农学硕士、医学硕士、管理学硕士等。
3. “学科专业”名称填写： “二级学科”全称。

**报告提纲**

1. 选题的背景及意义
2. 国内外本学科领域的发展现状与趋势
3. 课题主要研究内容、预期目标
4. 拟采用的研究方法和技术路线
5. 已有科研基础与所需的科研条件
6. 研究工作计划与进度安排
7. 参考文献

目录

[1 本文的研究背景和意义 6](#_Toc459646433)

[1.1 嵌入式多核调试技术 6](#_Toc459646434)

[1.2 基于SPARC架构模拟器的多核调试 6](#_Toc459646435)

[2 国内外本学科领域的发展现状与趋势 7](#_Toc459646436)

[2.1 常用多核调试技术 7](#_Toc459646437)

[2.1.1 硬件调试 7](#_Toc459646438)

[2.1.2 软件调试 7](#_Toc459646439)

[2.1.3 常用技术对比 8](#_Toc459646440)

[2.2 SPARC架构模拟器 9](#_Toc459646441)

[2.2.1 模拟器框架结构 9](#_Toc459646442)

[2.2.2 BM3803处理器核 10](#_Toc459646443)

[3 课题主要研究内容、预期目标 11](#_Toc459646444)

[3.1 主要研究内容 11](#_Toc459646445)

[3.2 预期目标 11](#_Toc459646446)

[4 拟采用的研究方法和技术路线 11](#_Toc459646447)

[4.1 可扩展的多核调试协议的交互机制研究 11](#_Toc459646448)

[4.1.1 RSP协议 12](#_Toc459646449)

[4.1.2 Flex集成以及接口设计 13](#_Toc459646450)

[4.2同步多核调试下断点处理机制的研究。 13](#_Toc459646451)

[4.2.1 传统同步多核断点处理机制 14](#_Toc459646452)

[4.2.2 SimICT模拟器中的断点机制 14](#_Toc459646453)

[4.2.3 同步模式模拟器调试命令处理 15](#_Toc459646454)

[4.3 异步多核调试机制 16](#_Toc459646455)

[4.3.1 主机端与目标机端异步交互 16](#_Toc459646456)

[4.3.2 调试流程 17](#_Toc459646457)

[5 已有科研基础与所需的科研条件 18](#_Toc459646458)

[5.1 已有科研基础 18](#_Toc459646459)

[5.2 所需的科研条件 18](#_Toc459646460)

[6 研究工作计划与进度安排 18](#_Toc459646461)

[7 参考文献 19](#_Toc459646462)

# 1 本文的研究背景和意义

## 1.1 嵌入式多核调试技术

嵌入式技术已被广泛应用于科学研究、工程设计、军事技术等各个领域，成为当前 IT 领域中炙手可热的技术。而随着近年来计算机硬件水平的高速发展和用户应用需求的提高，基于多核处理器的嵌入式系统以其高性能、低功耗等优势，在当前嵌入式系统中得到了广泛的应用[1-3]。基于多核处理器芯片的开发在提高了计算机运算速度的同时也为嵌入式系统的开发带来了很大的困难和挑战[4]。在嵌入式系统的开发过程中，系统开发平台是最基本的，而其本身的实现也是难度最大的，因为开发平台中涉及到的编译器、调试器等工具，都与目标板的体系结构相关。因此，设计实现一套良好的基于多核处理器的平台工具，便成为了嵌入式系统开发中的重点以及难点。其中好的调试器，可以使得那些需要运行在多核上的应用程序的实现更简单快捷，加快整个嵌入式系统的开发速度。

在硬件处理能力不断增强的过程中，嵌入式系统软件的调试手段也在经历着一系列的变迁，原来的调试手段已经逐渐退出历史舞台，新的调试手段也在不断涌现。在嵌入式系统软件调试发展过程中，主要使用的调试手段[5]有以下几种：ROM Monitor方式，Debug Stub/Server，在线仿真（In-Circuit Emulation，ICE）方式，片上调试（On-Chip Debug，OCD）方式，以及模拟器调试。由于嵌入式系统往往是资源有限的计算机系统，CPU 处理速度慢、自带存储空间小、没有人机交互设备等都给嵌入式系统软件调试带来了很大阻碍。调试手段的好坏直接影响到软件的开发效率以及软件的质量。为此，使用一套操作方便、安全性好、可靠性高及性能完善的调试工具对嵌入式系统软件开发人员来说非常重要[6]。

## 1.2 基于SPARC架构模拟器的多核调试

在航天领域中，要求航天器中的计算机系统(星载计算机系统)[7]可以在特殊的环境下高效完成复杂应用的处理，同时还要控制成本。目前，在世界范围内星载计算机系统中所使用的处理器架构只有两种，一种是由美国使用的POWERPC架构，另一种就是欧洲主导的SPARC架构。处于战略安全问题的考虑，中国星载计算机系统大多采用开源的SPARC架构。因此，完成基于SPARC架构的应用开发和调试是十分必要的。

为了加速调试功能的开发，本文在没有操作系统或操作系统不完善的模拟器上，研究并实现了对多核程序的调试。同时，由于模拟器采用了模块化的设计思想，通过注册模块接口和回调机制，可以很容易实现对不同处理器核的调试。

本文中模拟器的处理器核采用了BM3803MG[8]，BM3803MG 是基于 SPARC V8 体系结构的 32 位 RISC 嵌入式处理器, 可用于板上嵌入式实时计算机系统,能够满足各种航天应用的功能以及性能指标要求,只要加上存储器和与应用相关的外围电路,就可以构成完整的单板计算机系统。

GDB 是一款开放性的源码调试器，同时高版本的GDB(7.0以后)还增加了对多核调试的支持，因此选用了GDB作为主机端端调试器，并针对调试协议进行模拟器多核调试的研究。

# 2 国内外本学科领域的发展现状与趋势

## 2.1 常用多核调试技术

在嵌入式系统中，受限于目标机的性能、功耗等问题，绝大部分调试都采用远程调试。而对于定位为航天应用的计算机系统，必然需要采用远程调试的策略。

常用的远程多核调试策略分为硬件调试和软件调试两大类。

### 2.1.1 硬件调试

硬件调试方法，一般需要在宿主机和目标机之间通过特殊硬件来连接，如JTAG、EJTAG、Trace 等，通过使处理器进入调试态来查看处理器核的内部运行情况。硬件调试器通常用于较底层的系统开发过程中的调试，如开发 Bootloader程序，操作系统内核的开发或移植过程，或者设备驱动程序的开发等。硬件调试方法优点是：速度较快，并且因为采用了外部调试硬件，所以调试程序本身不占用目标机的系统资源，调试环境和最终的程序运行环境基本一致，不改变程序执行的行为，因此具有低侵入性。缺点是：由于调试硬件价格较高，不利于团队的整体开发，芯片面积和功耗都有所增加。常用的硬件调试方法有：

1. ICE (In Circuit Emulator)

在线仿真器 ICE，是一种完全仿造调试目标 CPU 设计的设备，可以执行目标机 CPU 的指令，但 ICE的 CPU 引脚线比目标机实际 CPU 的引脚线多，用于将内部信号输出到目标机上[9]。目标机对用户来说是完全透明的、可控的。在调试时，目标机与 ICE 之间用仿真头连接，ICE 与宿主机之间用串口、并口或以太网口等连接。ICE 可以真正运行所有目标 CPU 的动作，ICE 上的内存也可以被映射到用户的程序空间，从而给调试过程带来很多便利。这样在采用 ICE调试时可以连接目标机，也可以不连接目标机。宿主机端运行的调试器通过 ICE来控制运行在目标机上的被调试程序。

1. OCD (On-Chip Debug)

片上调试的设计思想是，在处理器设计初期就考虑了调试的需求[10]，在芯片内部加入专门的调试处理逻辑。根据研究，绝大部分用户仅仅使用了一些基本的调试功能,如断点、单步、处理器访问、内存访问等， OCD 将实时跟踪与运行控制分开来，将运行控制放到目标机系统的 CPU 核内由专门的调试控制逻辑模块来实现，并用一个专用的串行信号接口开放给用户。采用这种方式宿主机端的调试器可直接向目标机发送命令，并读写目标机的内存和寄存器，控制目标程序的运行。

### 软件调试

软件调试方法，通过在目标机上植入调试代理，辅助宿主机对被调试程序的执行过程进行控制，以便随时查看和修改被调试程序执行的状态。这种方法通常用于应用软件的开发，少数情况下亦可用于底层系统的开发，宿主机和目标机之间一般通过串口或网口来连接。常用的软件调试方法有：

1. ROM monitor

对于嵌入式系统来说,其地址空间里通常都会有一段只读空间,ROM monitor的调试手段正是利用了这一点,通过在嵌入式系统只读空间中写入辅助调试的软件,达到调试程序的目的[11]。ROM monitor有两个功能:用来控制嵌入式目标板上运行的程序;负责与PC端的调试器进行调试信息通信,一般使用远程调试协议(RSP, Remote Serial Protocol)[12]来进行通信。由于ROM monitor是驻留于只读空间,当嵌入式系统上电后,首先会执行这段ROM monitor代码, 达到对目标系统的控制,然后通过PC端调试器与ROM monitor相连,这样便可以进行调试,基本的ROM monitor包含了基本调试使用的命令,如断点、单步、寄存器访问、内存访问等。比较复杂的ROM monitor还有一些高级功能,如系统性能分析、代码密度分析、ROM空间访问等。

1. Debug Stub/Server

调试桩(Debugging Stub)和调试代理(Debugging Server),虽然他们也是需要运行在嵌入式系中,但是在使用中,并不需要烧写到ROM中去,而且系统运行开始的时候,并不需要首先运行调试桩和调试代理。要使用调试桩或调试代理,首先需要做的是使用其他手段将软件下载到嵌入式目标板上,然后运行程序,它们都是通过RSP协议与PC机软件进行通信的,且也都能完成基本的调试命令,如断点、单步、寄存器访问、内存访问等。

调试桩使用时不需要嵌入式操作系统的软件支持,它需要与嵌入式软件一起绑定运行,即需要在系统的编译阶段将调试桩功能加入进去,这种调试方式是一种侵入式的调试,一般用来调试嵌入式底层的一些软件,如板级支持包、硬件抽象层等;调试代理则需要嵌入式类Unix系统的软件调试接口的支持,它的运行机制与本地机的调试器运行类似,通过系统接口达到对目标程序的控制,是一种非侵入式的调试方式。使用调试代理时,嵌入式目标板首先运行类Unix操作系统,然后运行调试代理和被调试的应用程序,用户可以通过调试代理来调试嵌入式应用程序。

1. 模拟器调试

通常使用的Simulator是指令级的模拟器,它相当于在宿主机上虚拟了一台目标机。该目标机可以是和宿主机的CPU不同的类型。利用指令集模拟器进行的交叉调试是一种完全软件模拟的调试方法,根本不需要目标板的支持,就连I/O等设备也都是软件模拟的。通过基于RSP实现的软件调试接口与宿主机GDB进行调试会话,实现调试。使用模拟器调试的优点在于,这种调试是一种纯软件的模拟调试,其可扩展性非常好,常常可以根据需求很快的变动,这在嵌入式软件开发的起步阶段是非常有实用价值的,可以大大降低研发正本。

### 常用技术对比

ICE有非常多的调试手段,其中实时跟踪是ICE提供的最有特色的调试手段,它可以获取处理器的运行情况但又不会占用系统运行的时钟周期,是典型的非侵入(nonintrusive)调试。但ICE也有一些缺点:1.过高的调试成本;2.较差的可扩展性;3.适用不够广泛。一些功能较为强大的ICE,其价格要上万美金,普通的ICE也要几千美金,对于团队开发,很少有企业能够支撑这种研发成本;ICE是为特定的处理器定制的,一旦该处理器有所升级,那么ICE本身会很难跟进,而且,当开发团队使用的处理器升级换代时,那么整套ICE将不能再使用了。正因为上述种种限制,在嵌入式业界,ICE并没有大规模的使用。

OCD的功能单元在CPU内部,流片时会预留出芯片的访问管脚,显然这种用于研发阶段的管脚越少越好,所以一般厂商在设计、制造芯片的调试模块时,会使用JTAG(Joint Test Action Group)标准[13]来进行访问,尽量在做到保证通用性的基础之上使用最少的硬件资源。OCD技术的体系结构非常多,包括PowePC、MIPS、ARM、x86等,这中间绝大部分体系结构的调试访问标准都是JTAG。JTAG占用了非常少的硬件资源,但因为JTAG属于串口通信,较慢的数据传输速度势必限制了使用OCD实现的CPU的调试,且OCD技术还不能够实现诸如实时跟踪这种复杂的调试功能。

ROM monitor的调试方式在实际使用中并不普遍，主要由于：1.本身也需要开发;2.平台相关度大。由于ROM monitor本身也是嵌入式软件,因此它也需要开发出来,一些特殊情况下,ROM monitor的开发难度要超过嵌入式软件开发的难度,使得嵌入式调试变得得不偿失;此外,由于ROM monitor本身是烧写到特定的嵌入式系统中的,因此其平台相关性必然很大,这也导致了 ROM monitor难以在不同平台上进行移植开发。

软件调试方法不需要专用的硬件支持，尤其是在系统开发初期，没有实际的硬件环境，只能使用软件调试方法。GDB Stub需要在编译期间将调试功能写入应用程序中，这对编译器和程序编写能力要求较高；GDB Server相当于运行在目标机上的一个应用程序，通过操作系统提供的相关接口，访问其它应用程序的运行情况，这需要目标机上有操作系统的支持。这种方式由于其非侵入性，在软件调试中经常被使用。

在基于SPARC架构的多目标调试项目中，实验环境为基于SPARC V8架构的模拟器。因此，硬件调试方法不可行，只可以利用软件方式进行调试。使用GDB Stub方式需要在应用程序编译期间将调试功能编译到程序中，这对于大量涉及保密的航天应用来说，是不现实的。被普遍使用的GDB Server技术，虽然可以实现非侵入式调试，但需要操作系统的支持。而在模拟器设计初期阶段，还没有实现操作系统的模拟，同时模拟器的结构以及应用程序调试功能的需求，在这一阶段可能经常发生变动。为了加快调试功能的开发，采用模拟器调试是最合适的选择。

## SPARC架构模拟器

### 2.2.1 模拟器框架结构

模拟器基于SimICT框架，模拟的每个处理器核都作为一个模块，需要在框架中注册，运行时由框架调用模块的接口。框架整体结构如图1所示：

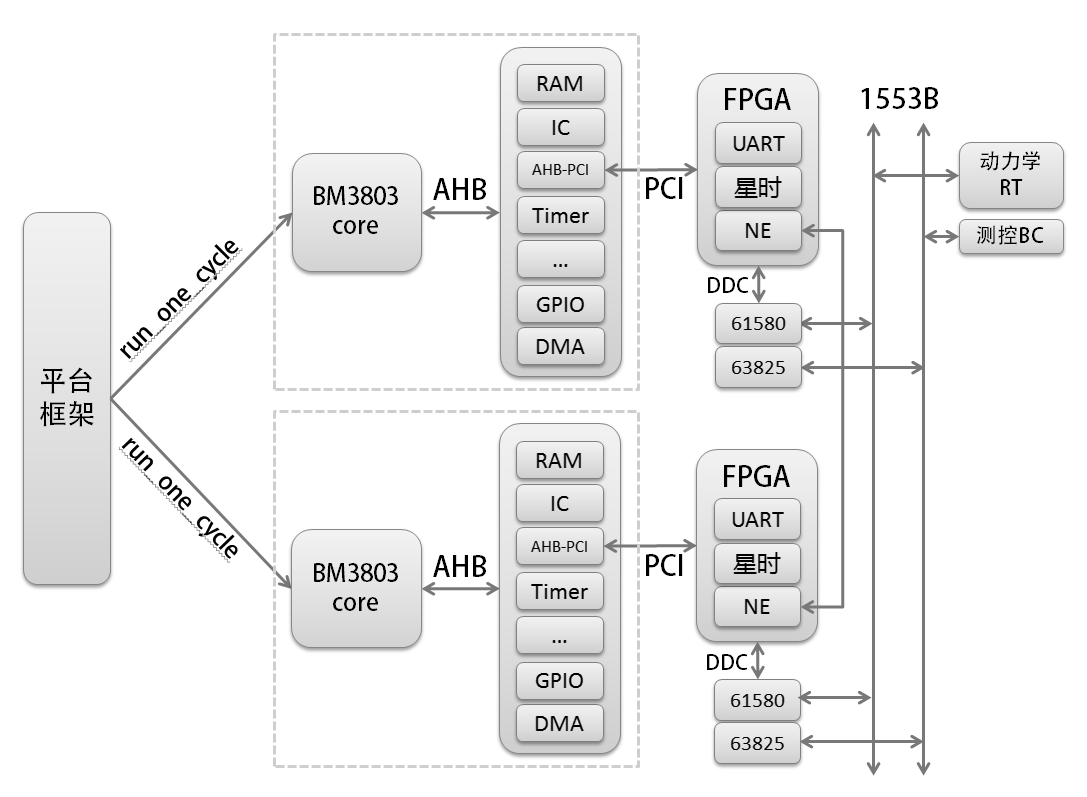


图1 SimICT框架结构图

图1给出了两个BM3803模拟核的情况。图中所有模块中，BM3803是可执行的，SimICT会在模拟时循环调用所有BM3803模块的执行接口，驱动其执行。出于性能等方面的考虑，其它的模块则是被动的，在执行时不会被SimICT直接执行，而是在其接口被BM3803调用时，间接地获得执行机会。被动模块的状态在模拟过程中不是时刻更新的，其最新状态的计算被延迟到间接地获得执行机会时。以FPGA的星时接口为例，其星时数据被BM3803读取的次数要远远低于BM3803获得的执行次数，因而被动地执行可以获得更优的性能。当FPGA的接口被BM3803调用时，它需要首先获取当前的模拟时间，并根据被读取星时的类型（GPS锁存、控制周期锁存等）计算其当前最新的锁存值，并将其返回给BM3803。

### 2.2.2 BM3803处理器核

BM3803 是基于 SPARC V8 体系结构的 32 位 RISC 嵌入式处理器, 可用于板上嵌入式实时计算机系统,能够满足各种航天应用的功能以及性能指标要求,只要加上存储器和与应用相关的外围电路,就可以构成 完整的单板计算机系统。

BM3803 芯片内部包含整数处理单元,浮点处理单元,独立的指令和数据 Cache,硬件乘法器和除法器,中断控制器,带有跟踪缓冲器的硬件调试单元,两个24 位定时器,通用 I/O 接口,看门狗,能够支持 PROM、 SRAM、SDRAM 和 I/O 映射空间访问的存储器控制器,具有软件可以控 制的省电工作模式,具有可实现 PCI 主机桥(Host bridge)和从属桥(Guest bridge)功能的 PCI 控制器。

# 3 课题主要研究内容、预期目标

## 主要研究内容

本文的主要工作是研究SPARC架构模拟器的多核调试技术，并实现多核调试功能。嵌入式系统的多核调试已经越来越成为嵌入式领域的重点研究方向，比较成功的调试工具有ARM RealView Debugger，TotalView Debugger，WindRiver Workbench，DS-5 Debugger等，这些调试工具有的是针对某种架构的工具，有的需要硬件支持，并且作为商用工具，都没有开源并且使用成本较高。在国内外的研究中，基于SPARC架构的多核调试几乎没有公开发表的研究。

针对SPARC架构模拟器以及多核调试的需求，本文主要进行以下几个方面的研究：

* **可扩展的多核调试协议的交互机制研究。**本文采用开源且功能强大的GDB作为调试工具，GDB与目标机通过RSP(Remote Serial Protocol，远程串行协议)进行交互。需要考虑如何高效的完成交互过程，同时考虑以后如何支持更多的协议类型。
* **同步多核调试下断点处理机制的研究。**断点处理是调试过程的核心，如何维护多个断点的信息，以及如何对普通断点，条件断点等不同类型的断点进行处理是这部分研究的重点，同时在此基础上，完成对其它常见调试命令的处理。
* **异步多核调试机制研究。**在完成同步多核调试的基础上，加入对异步调试的支持。研究如何支持主机端与模拟器进行异步交互，如何保证多个处理器核之间调试的独立性。

## 3.2 预期目标

* 完成模拟器与主机端的正确交互。
* 完成对模拟器的同步和异步多核调试。
* 将GDB调试与Eclipse集成，完成图形化界面下对模拟器的远程调试。

# 4 拟采用的研究方法和技术路线

在SimICT模拟器框架的基础上，结合对SPARC架构处理器核的模拟，实现多核调试功能。模拟器使用C语言实现，用户调试界面基于Eclipse插件开发。

## 4.1 可扩展的多核调试协议的交互机制研究

在基于模拟器的调试中，用模拟器模拟目标机，模拟器可以运行在主机上也可以运行在远程服务器上，模拟器与主机通过网络或串口连接。利用RSP(Remote Serial Protocol)完成主机与模拟器之间的交互。图2给出了两者之间的交互过程。

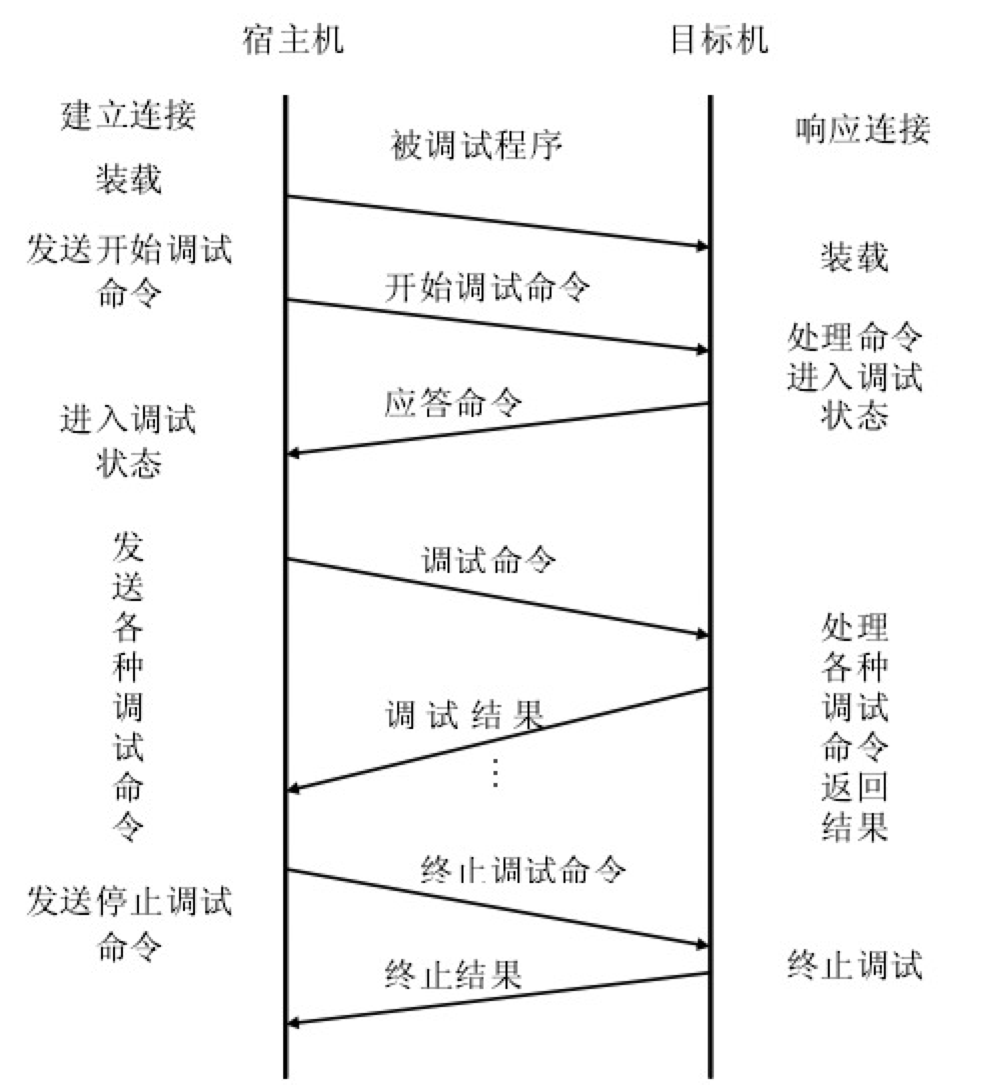


图2 基于GDB的远程调试交互过程

### 4.1.1 RSP协议

主机端与模拟器之间通过RSP协议[14]交互。RSP制定了一系列具备一定格式的调试命令。RSP 协议包含了起始位，数据，结束位和校验和。一个典型的 RSP 协议包的结构如图3：

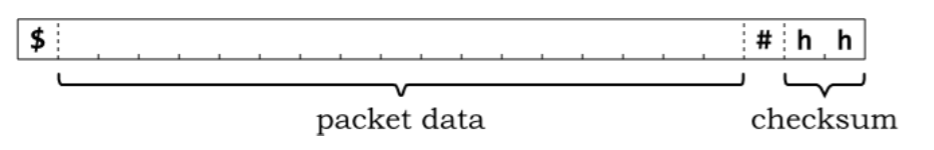


图3 RSP协议数据包格式

每个数据包都以$开口，packet data指出来命令类型，命令的参数以及具体数据，数据部分以#结尾，checksum是packet data部分的校验和模256后的16进制表示。

### 4.1.2 Flex集成以及接口设计

RSP协议中除了常用的调试命令外，还有很多高级命令。在实现调试功能初期，无法全部支持所有命令，同时，为了应对之后RSP协议的扩展，需要设计一套可扩展性强的交互接口。

Flex是一个自动化工具，在.l文件中定义一系列规则以及对应的动作(action)，经过编译后，可以按照定义好的规则自动生成一个C函数yylex()，也称为扫描器（Scanner）[15]。这个C函数把文本串作为输入，按照定义好的规则分析文本串中的字符，找到符合规则的字符序列后，就执行在规则中定义好的动作。

在模拟器交互设计中，将目前支持的调试命令对应的正则表达式作为Flex中的规则，同时指定规则中的action为解析协议的功能函数。rsp\_parse.l中除了包含了这些规则以及action，还包含rsp\_parse()。rsp\_parse()是交互过程中向模拟器展示的接口，在内部调用了Flex生成的yylex()，将输入的协议解析成定义好的rsp\_packet结构体返回给模拟器，图4给出了协议解析的处理过程。

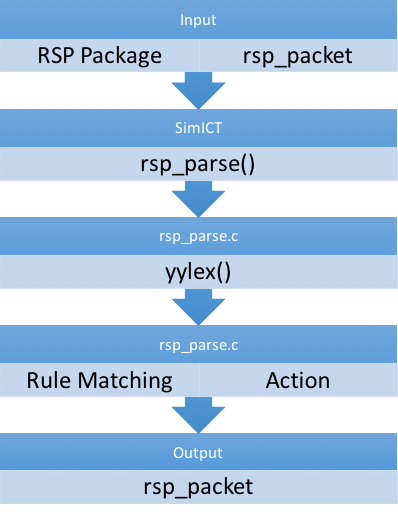


图4 协议解析过程

将Flex集成到模拟器中后，模拟器只需要调用rsp\_parse()接口就可以完成协议数据的解析。同时，如果要增加新的协议命令，只需要在修改rsp\_parse.l，增加新的规则和action，并完成对新命令的解析。这些操作对模拟器都是屏蔽的，在模拟器中只需要在rsp\_packet中增加新的命令类型即可，这极大提高了模拟器交互的可扩展性。

## 4.2同步多核调试模式下断点处理机制的研究。

断点处理是调试过程的核心，常用的调试功能都会涉及到断点处理。例如在主机端对某一行代码下断点，运行到断点处暂停，查看程序状态；程序单步执行时，GDB也会在下一条指令处下断点，然后执行到断点处暂停。

在GDB中可以根据行号或函数名设置断点，这也是最常用的断点类型，对于这类断点，程序运行到断点处暂停，并且继续运行后断点仍有效；还可以设置临时断点，即断点只有效一次，当越过断点后断点即失效；此外，还可以设置条件断点，当条件表达式成立时程序才会在断点处暂停。

### 4.2.1 传统同步多核断点处理机制

在常见的软件调试方式中，最常用的是在目标机端运行gdbserver，并通过系统调用ptrace()[16]完成对应用程序的调试。当向某个地址插入断点时，gdbserver 需要借助系统调用 ptrace()：先获取该地址处的原始指令，并保存起来；然后将断点指令写入，当程序执行到这个位置时，就会触发断点指令，完成相应的功能。

gdbserver 的断点管理中使用了两类断点数据结构：原始断点和高级断点。

原始断点数据结构记录断点的低级信息，包括断点地址、原始数据和引用计数等信息。每个指令地址最多对应一个原始断点。在 gdbserver 中，原始断点用结构体 raw\_breakpoint 表示和保存，其字段含义如下[17]：

struct raw\_breakpoint {

struct raw\_breakpoint \*next;

int refcount; //引用数：一个原始断点可对应多个高级断点

CORE\_ADDR pc; //断点对应的地址

unsigned char old\_data[8]; //保存原始内存数据（指令）的缓冲区,

//gdbserver 将其设置为 8 字节

int inserted; //非 0 时表示断点已经插入 inferior 的内存中

int shlib\_disabled; //非 0 时表示断点当前无效

};

高级断点的结构如下：

struct breakpoint {

struct breakpoint \*next; //把 inferior 的所有断点链在一起

enum bkpt\_type type; //断点的类型

struct raw\_breakpoint \*raw; //指向原始断点的指针，一般非 NULL

int (\*handler) (CORE\_ADDR); //当命中断点时，调用的函数

};

其中 bkpt\_type 是高级断点的类型：最常用的是 gdb\_breakpoint 类断点，表示用 RSP 的‘Z0’包插入的断点[18]（软件断点指令）；reinsert\_breakpoint 类断点，表示软件单步（software-single-step）断点；other\_breakpoint 表示其他类型断点。

handler 是当断点命中时，需要调用的处理函数，对于 gdb\_breakpoint 类断点，处理函数为 NULL。如果该函数返回 1，则断点应该被删除；如果该函数指针为NULL 或返回 0，则断点继续保持插入状态。

### 4.2.2 SimICT模拟器中的断点机制

在没有操作系统的模拟器上，采用快速断点匹配策略以及有效性判断策略，使模拟器本身对调试功能进行支持。

为了支持多种类型的断点，设计保存断点的数据类型。对于同步模式下的断点，至少需要保存断点类型(type)，断点地址(address)，引用计数(count)，条件表达式(expr)，有效位(valid)。

一个程序的调试过程中，可能会设置或删除多个断点，必须将有效的断点保存起来，当被调试核的PC等于某个有效断点时，暂停所有处理器核的执行。

由于缺少断点指令，不能像ptrace()一样在断点位置插入指令。而如果每次PC改变都需要与所有断点比较，势必会降低模拟器的速度，因此，使用哈希表存储断点信息，加速断点匹配过程。

图5给出了模拟器处理设置断点的过程，通过对断点地址进行哈希计算，查找是否这个位置已经设置过断点。如果该位置没有设置过断点，则增加一个有效断点；如果设置过断点，判断是否是同类型断点，如果是则引用计数加一，否则增加一个有效断点。

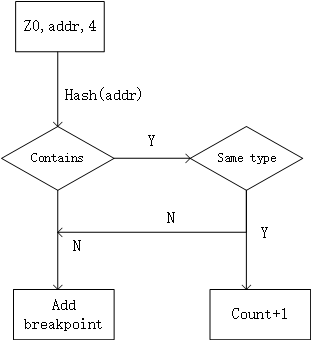


图5 增加断点处理流程

### 4.2.3 同步模式模拟器调试命令处理

在同步模式下，当被调试的处理器核遇到断点暂停执行后，其他处理器核也会暂停运行[19]。在调试时，选择需要调试的处理器核，通过GDB发送的命令对当核程有效，在调试过程中可以更换处理器核，查看其它核的状态。

图6给出了同步模式下模拟器调试命令的处理流程，其中左边的流程图表示在每个处理器核运行一个周期后，会检查是否遇到断点，如果遇到断点就进入handle\_debug()中进行调试。右边的图给出了handle\_debug()的流程，其中handle\_packet()根据命令类型进行相应的处理，并生成相应的调试信息返回到主机端。

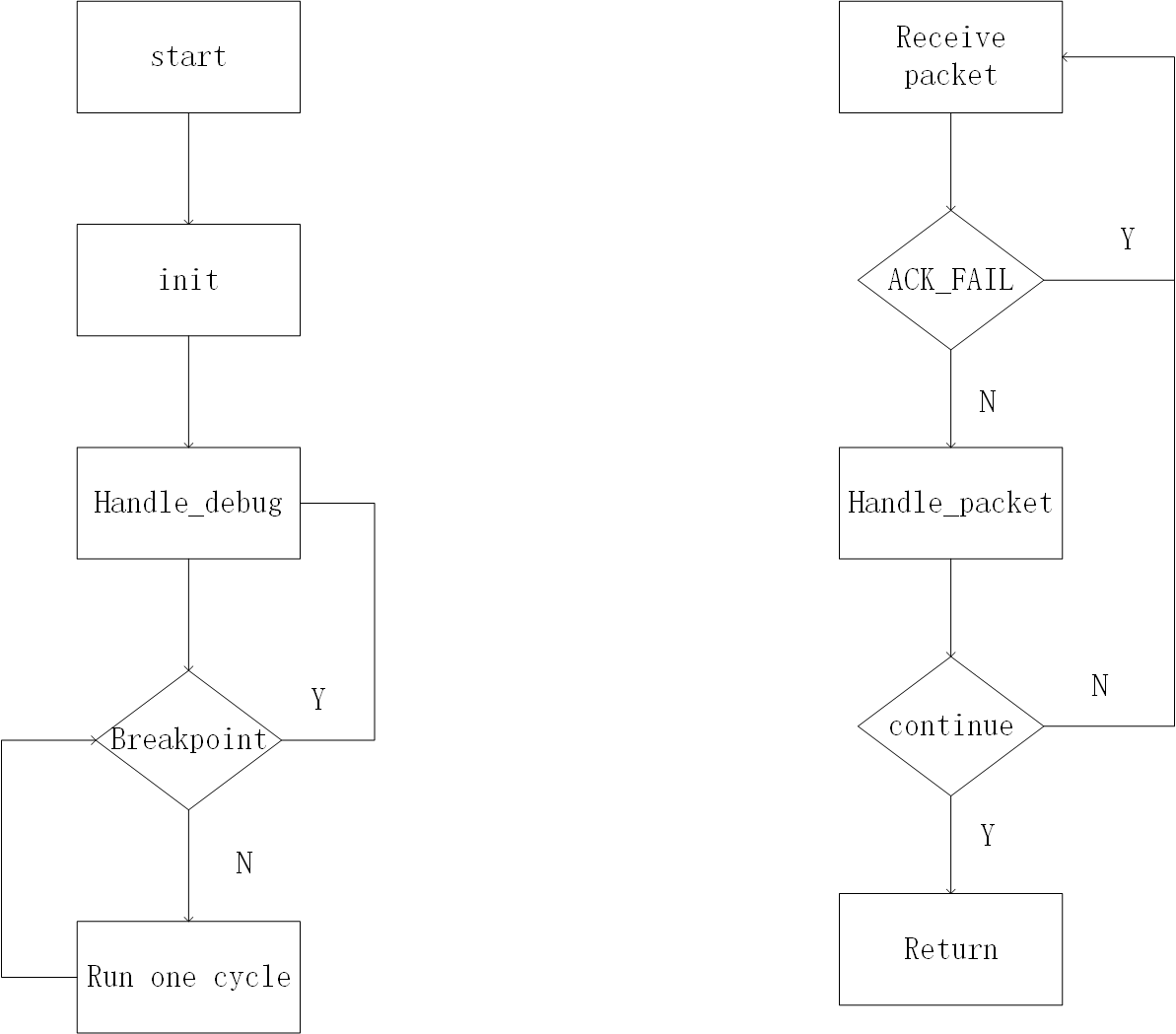


图6 调试命令处理流程

## 4.3 异步多核调试机制

在模拟器多个处理器核上执行的多个程序一般是完成整个程序的不同部分，相当于一个进程的多个线程。在同步机制下，当一个处理器核上的程序遇到断点暂停后，其余处理器也会暂停执行，这与实际应用情况不是十分吻合。在实际中，一个处理器核出现中断时，允许其他处理器继续执行，同时在主机端也应该允许对其它处理器核的调试[20]。

以上需求表明了异步多核调试机制的必要性。一方面支持主机端与目标机端端异步交互，另一方面支持多个处理器核的异步调试。

### 4.3.1 主机端与目标机端异步交互

在同步模式中，主机端发送调试命令后freewheel+李戈+中国科学院大学+Java开发需要等待目标机返回调试信息后才可以继续调试。而在异步模式中，主机端可以发送多个调试命令到目标机，采用事件缓存策略，在目标机端对命令进行缓存，同时在主机端对返回对调试信息也进行缓存。

在目标机端，将每个调试命令都看作一个事件，对每个处理器核设置事件队列。在模拟器调度时，检查事件队列中是否有未处理的调试命令，如果有就进入调试命令处理的流程。

在主机端，也维护一个事件队列，将用户输入与目标机返回的调试信息都作为事件，缓存到队列中，通过select()系统调用[21]选取事件进行处理。

### 4.3.2 调试流程

在异步调试中，设置断点时除了要指明断点地址外，还要指明对哪个处理器核有效。在保存断点的数据结构中，增加目标处理器核(target\_core)信息，同时修改断点有效性判定方法，只有当目标处理器核运行到断点处才会暂停执行，其它处理器核不受影响。

在调试过程中，为了使每个处理器核互不影响，采用调试信息隔离机制，动态地为每个处理器核保存各自的调试信息。在模拟器设计中，为了提供充分的可扩展性，每个处理器核被当作一个模块注册到框架中，通过接口调用完成程序运行。在异步调试中，当注册处理器核时，为其开辟私有的调试空间，保存核相关的调试信息。在调试过程中框架调度处理器核运行时，会查看私有的事件队列中是否有待处理的调试命令，完成调试。为了统一完成调试信息的交互，在框架中维护一个调试消息队列，保存每个处理器核需要返回的调试信息。图7给出了基于上述设计的事件管理方案。

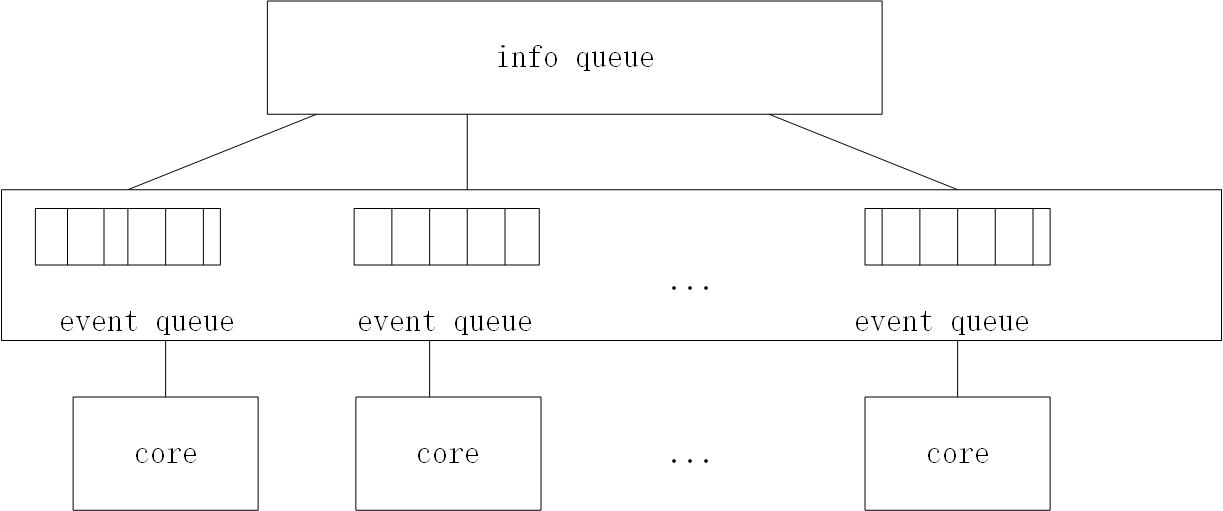


图7 异步模式下事件调试事件管理

在异步模式下，断点的处理也与同步模式不同。只有当程序运行到断点位置，并且调试核与设置断点时指定的核一致时，断点才生效。图8描述了异步模式下，断点命中的判断算法。

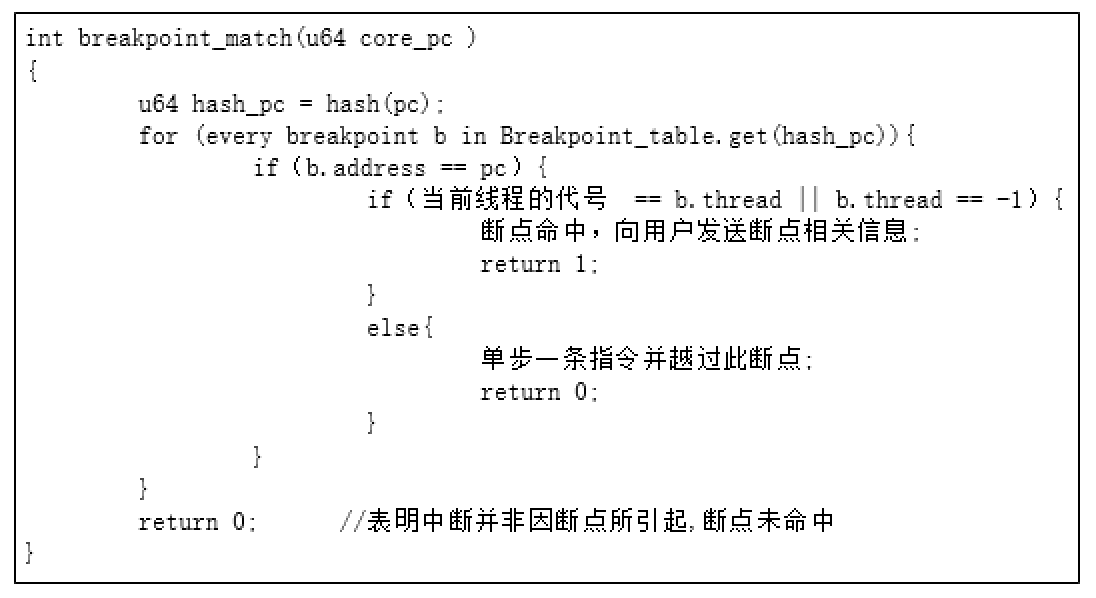


图8 异步模式下断点命中判断

图9给出了异步调试过程中主机端与目标机端的流程。由于维护了事件队列，调试命令的发送和处理可以被异步执行。

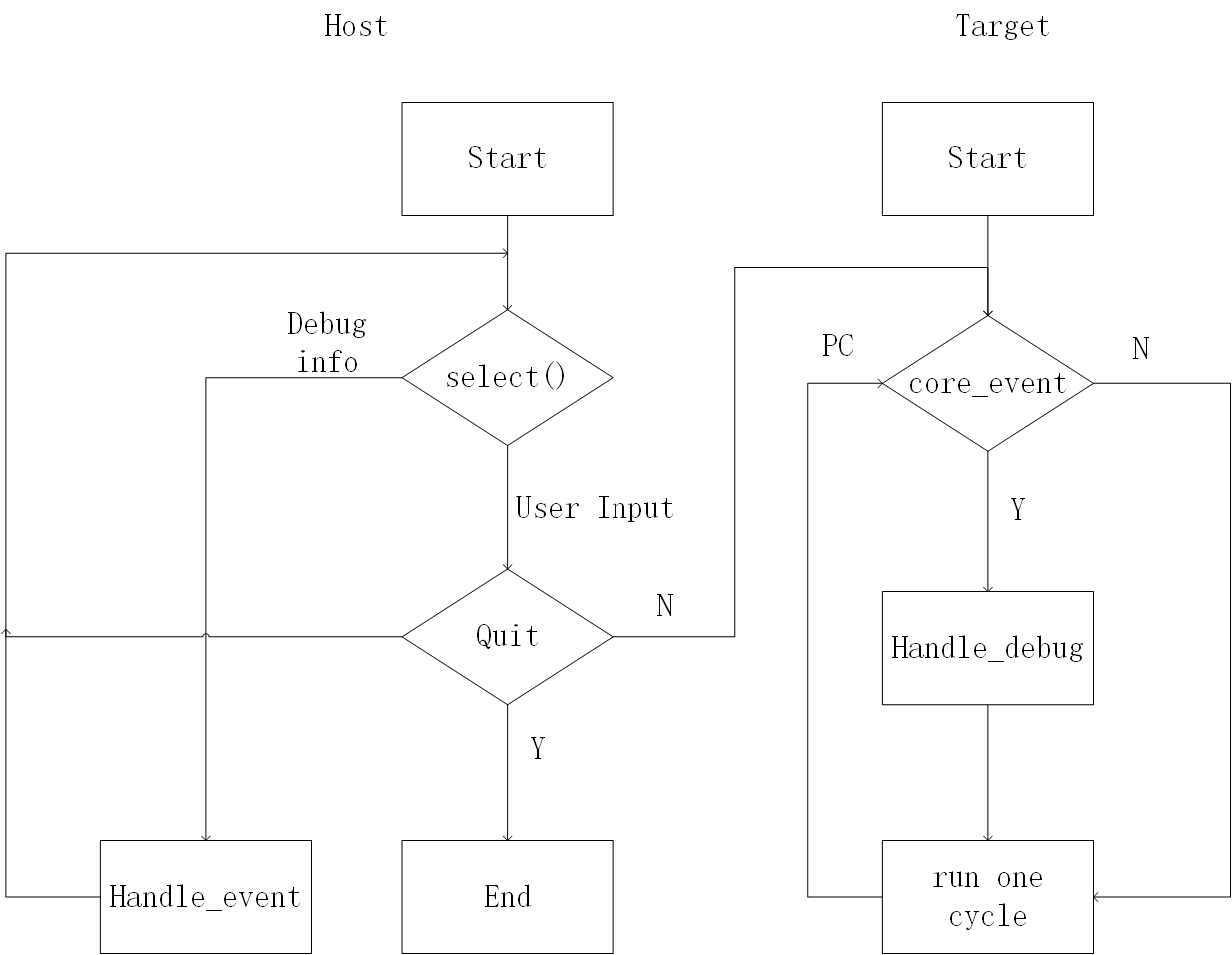


图9 异步调试流程

# 5 已有科研基础与所需的科研条件

## 5.1 已有科研基础

1.已经完成了双核SPARC模拟器的基本功能。

2.研究了GDB调试协议。

3.完成了部分eclipse插件定制开发。

## 5.2 所需的科研条件

实验环境需求：SPARC模拟器，eclipse开发环境。

# 6 研究工作计划与进度安排

2016.8~2016.11 完成基于SPARC模拟器框架的同步调试功能实现。

2016.11~2017.3 完成异步调试功能的研究，测试模拟器的调试功能，准备中期答辩。

2017.3~2017.5 完成毕业论文和答辩工作。

# 7 参考文献

[1] 高效伟.多处理器并行处理的新发展 Linux 操作系统内核对 SMP（对称多处理器）的支持，计算机应用研究,2002(9):62-67

[2] Jeff Parkhurst, John Darringer, Bill Grundmann. From Single Core to Multi-Core: Preparing for a new exponential.

[3] Asawaree Kalavade, Joe Othmer, Bryan Ackland. Software environment for a multiprocessor DSP. Annual ACM IEEE Design Automation Conference, 1999,827-830

[4] 彭冬脉，基于GDB的双核调试器的研究与实现，电子科技大学学位论文，2010

[5] 罗克露，陈云川.嵌入式软件调试技术[M].北京：电子工业出版社，2009.

[6] 范涛, SPARC平台模拟器源码级调试系统的研究与实现. 计算机工程与应用[J]. 2013,49(4)

[7] http://www.eepw.com.cn/article/114121.htm

[8] BM3803产品用户手册(C)

[9] 罗蕾.嵌入式实时操作系统及应用开发.北京:北京航空航天大学出版社,2005，77-81

[10] 蒋龙,基于GDB的嵌入式多任务调试器的设计实现与集成.浙江大学学位论文,2014

[11] 张克丰.嵌入式系统的调试方法研究及调试器后台处理模块设计.西安电子科技大学学位论文,2012.

[12] GNU organization. The GDB remote serial protocol[EB/OL].2011-08.

[13] 谭金伟.嵌入式JTAG仿真调试器的研究与实现.电子科技大学学位论文,2004.

[14] Jeremy Bennett. Howto: GDB Remote Serial Protocol

[15] <http://www.cnblogs.com/wanghetao/archive/2011/11/07/2240193.html>

[16] 何克右,周彩贞.Linux2.6 进程调度机制的剖析.华中师范大学学报，2007,41(4):520523

[17] http://blog.csdn.net/guojin08/article/details/25457311

[18] The gnu Source-Level Debugger. Debugging with gdb Tenth Edition.

[19] 李国徽, 诸方舟, 王安军, 马艳, 曹粟, 黄瑛, 李艳红. 事件循环机制在嵌入式多线程应用级调试器设计中的应用. 中国科技论文, 2013, 15(1): 67~71

[20] K. Chatterjee, L. De Alfaro, V. Raman, et al. Analyzing the impact of change in multi- threaded programs. In: Proceedings of the 13th international conference on Fundamental Approaches to Software Engineering. Berlin: Springer Verlag, 2010.

[21] 黄瑛，基于non\_stop调试模式的嵌入式应用级交叉调试器的研究, 华中科技大学学位论文，2013