[19] 中华人民共和国国家知识产权局

[51] Int. Cl.
G06F 11/36 (2006.01)

[12] 发明专利申请公布说明书

[21] 申请号 200710121864.9

[43] 公开日 2008 年 2 月 13 日

[11] 公开号 CN 101122880A

[22] 申请日 2007.9.17
[21] 申请号 200710121864.9
[71] 申请人 福建星网锐捷网络有限公司
地址 350015 福建省福州市马尾区快安大道 M9511 工业园
[72] 发明人 王龙顺
[74] 专利代理机构 北京同立钧成知识产权代理有限公司
代理人 刘 芳

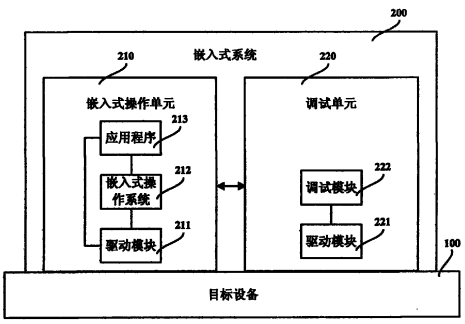
权利要求书 2 页 说明书 22 页 附图 3 页

[54] 发明名称

内嵌调试器的嵌入式系统及嵌入式系统调试方法

[57] 摘要

本发明涉及一种内嵌调试器的嵌入式系统，包括嵌入式操作单元，其中还包括调试单元，与嵌入式操作单元连接；调试单元包括：驱动模块以及与驱动模块连接的调试模块；驱动模块，用于控制调试命令的输入和输出；调试模块，用于触发嵌入式操作单元产生异常，捕获异常，以及对异常信息进行分析处理，根据异常信息的类型选择预设方法对嵌入式操作单元中的硬件和/或软件进行调试。本发明还涉及一种嵌入式系统调试方法。本发明通过在嵌入式系统中内嵌调试单元，以实现在嵌入式系统上直接对软件和/或硬件进行调试，从而降低了设置额外硬件和/或开发额外的软件而导致的成本，方便大规模的推广应用。



1、一种内嵌调试器的嵌入式系统，包括嵌入式操作单元，其特征在于还包括调试单元，与所述嵌入式操作单元连接；所述调试单元包括：驱动模块以及与驱动模块连接的调试模块；驱动模块，用于控制调试命令的输入和输出；调试模块，用于触发嵌入式操作单元产生异常，捕获异常，以及对异常信息进行分析处理，根据异常信息的类型选择预设方法对嵌入式操作单元中的硬件和/或软件进行调试。

2、根据权利要求 1 所述的嵌入式系统，其特征在于，还包括调试界面，所述驱动模块与所述调试界面连接，用于控制调试界面的显示和调试命令的输入；所述调试模块与所述调试界面连接，用于接收并响应由所述调试界面发送的调试命令。

3、根据权利要求 1 所述的嵌入式系统，其特征在于，所述嵌入式操作单元包括通知模块，用于向所述调试单元发送调试命令。

4、根据权利要求 1 所述的嵌入式系统，其特征在于，所述调试模块包括：

异常处理子模块，用于捕获异常，对异常信息进行分析处理，根据异常信息的类型将异常信息发送至调试功能子模块；

调试功能子模块，用于触发嵌入式操作单元产生异常，并与所述异常处理子模块连接，用于采用所述调试功能子模块内的预设方法对嵌入式操作单元中的硬件和/或软件进行调试。

5、根据权利要求 4 所述的嵌入式系统，其特征在于，所述调试功能子模块包括：固定断点处理子模块、动态断点处理子模块、单步跟踪处理子模块、数据监视处理子模块、和/或指令监视处理子模块。

6、根据权利要求 5 所述的嵌入式系统，其特征在于，所述调试功能子模块还包括：反汇编处理子模块、源代码调试子模块、数据访问子模块、和/或路径跟踪调试子模块。

7、一种嵌入式系统调试方法，其特征在于，包括:

调试模块触发嵌入式操作单元产生异常;

调试模块捕获异常，对异常信息进行分析处理，根据异常信息的类型选择预设方法;

调试模块采用所述预设方法对嵌入式操作单元中的硬件和/或软件进行调试。

8、根据权利要求 7 所述的调试方法，其特征在于，在调试模块对目标设备和嵌入式操作单元的访问和/或控制之前还包括: 驱动模块或嵌入式操作单元向调试模块输入调试命令。

9、根据权利要求 7 所述的调试方法，其特征在于，所述预设方法包括固定断点实现方法、动态断点实现方法、单步跟踪实现方法、数据监视实现方法和/或指令监视实现方法。

10、根据权利要求 9 所述的调试方法，其特征在于，所述预设方法还包括: 反汇编实现方法、源代码调试实现方法、数据访问实现方法、和/或路径跟踪实现方法。

内嵌调试器的嵌入式系统及嵌入式系统调试方法

技术领域

本发明涉及嵌入式系统软件和/或硬件的调试系统及方法，尤其是涉及一种内嵌调试器的嵌入式系统及嵌入式系统调试方法。

背景技术

调试是嵌入式系统软件开发过程中必不可少的环节，通用的桌面操作系统与嵌入式操作系统在调试环境上存在明显的差别。前者，调试器与被调试的程序往往是运行在同一台机器、相同的操作系统上的两个进程，调试器进程通过操作系统专门提供的接口控制、访问被调试进程（比如微软开发的visual C++,它集开发环境、编译和调试于一身，程序的开发、编译和调试都由Visual C++完成，Visual C++调试和被调试程序都运行在同一台机器下）。

嵌入式开发和调试有别于通用的桌面软件的开发，嵌入式开发采用交叉开发的模式(Cross Developping)：开发系统是建立在软硬件资源丰富的 PC机(或者工作站)上，我们称之为宿主机（HOST）,嵌入式程序的编辑，编译，链接过程都在host端完成,而程序最终运行的却是和host有很大区别的嵌入式设备，我们称之为目标机（Target）,目标机和宿主机的差别主要指:

硬件环境差别:通常CPU类型不同，例如HOST的CPU为intel X86,而Target为freescale公司的powerpc。

软件环境的差异：在host上都有成熟的操作系统的应用软件支持，比如windows,以及freescale公司的codewarrior IDE集成开发和调试器。而Target一般都是裸机，或者需要调试的嵌入式操作系统。

嵌入式系统的调试和故障诊断一般有以下几种方法:

（1）添加打印调试信息和调试命令：这种方法是程序开发人员在程序开发阶段在需要调试的程序中插入打印信息的代码(比如在C语言中使用printf语句输出调试信息），可以打印程序变量的值、内存地址或者寄存器的内容、收/发数据包的内容等；程序运行时，当指定位置的打印程序被执行后，它就会打印出指定的信息（打印信息的输出位置可以多种多样，可以是串口，也可以把输出信息保存到文件中，或者从通信端口输出等）；开发人员通过分析输出的打印信息，来分析程序执行的过程，观察程序变量的值、或者收发包的内容等是否和期望的一致来进行调试。和打印调试信息类似，开发人员通过添加一些调试命令来显示程序的运行状态（打印变量的值，寄存器的值，统计信息等），程序运行后，开发人员通过输入调试命令让程序输出调试信息，通过对比输出信息来进行调试。此种调试方法存在以下缺陷：

（11）不够灵活：程序开发人员需要预先在期望的位置添加代码，或者预先添加调试命令，打印的位置以及打印的内容在开发阶段就已经固定，程序运行后不能修改；因为在调试阶段开发人员对于需要在何处打印信息？打印什么信息？不是非常的清楚，这会导致打印的信息不充分，无法进行故障定位，或者打印的信息很多都没用，无助于故障分析和调试；这导致的结果是程序开发人员需要不断修改添加打印信息，然后重新编译运行，重新分析打印信息，这样不断循环直至故障解决。

(12)无法支持指令级的调试：很显然这种调试方法无法进行设置断点和单步跟踪；断点：指的是用户可以指定程序执行到什么位置先停下来，停下来后用户可以查看机器的当前状态（内存的值，或者寄存器的值），可以让程序从断点的位置继续执行，单步跟踪指的是目标机，每执行一条指令，会停下来，让用户观察机器状态。

(13)调试信息不能应用在实际运行系统中，当程序开发和调试完毕，程序作为最终程序被实际用户使用前，原来添加的打印调试信息要被删除，否则最终用户会看到一堆看不懂的调试信息，影响用户的使用，而且这会大大

影响实际运行系统的性能（打印信息会很大耗费 CPU 的资源），因此在程序发布前需要删除这些调试信息的程序。这就导致调试信息无法在实际运行系统中使用。如果最终程序出问题，就无法使用原来的调试信息进行调试。

(2)系统中附带故障诊断和分析模块：该模块属于系统软件的一部分，正常情况下该模块没有工作，在发现问题后，用户使用该模块诊断系统的故障（主要指的是硬件故障）进行分析和诊断，例如：Dell PC 的 BIOS 上就附带有故障诊断模块，用户如果发现 PC 工作不正常，就可以进入 BIOS 的故障诊断模块进行测试，看看是哪个硬件可能有问题；该模式可以在正式系统中附带该功能，非常方便产品维护和维修人员使用，有这个模块产品客户支持人员就能快速定位产品是否有问题；但是该模式也有自己的弊端：

(21)只能对硬件故障进行诊断，不能诊断软件故障；但是在嵌入式系统中更多的故障是软件引起，而且也无法作为开发阶段软件调试和诊断手段，只适合产品维护和维修人员使用。

(22)只能诊断可预知的故障，对于可预知的每个设备的每个故障都要有相应的诊断程序和他对应。由于每个设备的每个故障都要有相对应的诊断程序，因此程序的开发工作量大，而且每新增加或者修改一个设备就要相应增加和修改程序，灵活性低。

(3)桩（Stub）模式交叉调试方法：这种模式需要预先在目标机增加 Stub 模块，Stub 是运行在目标系统的一段程序，它负责监控目标机上被调试程序的运行，通常和 host 端的调试器一起完成应用程序的调试，包括以下步骤：调试器控制、访问被调试程序：调试器的这类请求实际上都将转换成对被调试程序的地址空间或目标平台的某些寄存器的访问，这些访问通过协议的方式传给 Stub,Stub 接收到请求后对目标地址或者寄存器直接进行访问，并把处理结果通过 Stub 发送给调试器，调试器处理后把结果显示给用户；断点调试和单步跟踪。Stub 在需要产生断点的位置用一条异常指令代替原来的指令，当程序执行到该位置，产生异常，异常被 Stub 模块捕获通知调试模块，并循

环等待调试模块的命令；这时调试模块就会解释 Stub 发送过来的信息为断点异常，并且通知用户 CPU 已经停下来，这时用户可以通过调试模块命令 Stub，查看目标机的内存和寄存器等信息进行调试；断点恢复：Stub 首先恢复断点位置的指令，并且退出异常处理，让程序从断点位置继续执行；单步跟踪：单步跟踪和断点调试原理类似，Stub 会通过设置 CPU 特殊寄存器，让 CPU 每执行完一条指令，就产生一次异常，异常被 Stub 接管后，后续流程和断点调试流程一致。该方法存在以下缺陷：

（31）该方法的实质是用软件接管目标系统的全部异常处理（exception handler）及部分中断处理，在其中插入调试端口通信模块，与主机的调试器交互。它只能在目标操作系统初始化，特别是调试通信端口初始化完成后才起作用，所以一般只用于调试运行于目标操作系统之上的应用程序，而不宜用来调试目标操作系统，特别是无法调试目标操作系统的启动过程。

（32）HOST 端需要调试器软件。

（33）Stub 要支持和调试器相配套的通讯协议，通讯协议实现复杂，而且每个调试器的通讯标准都不一样，需要为每个调试器开发相应的 Stub，开发工作量大。

（34）必须改动目标操作系统，这一改动即使没有对操作系统在调试过程中的表现造成不利影响， 至少也会导致目标系统多了一个不用于正式发布的调试版，也就是说最终系统不能带有该功能，而实际上很多故障只有在最终系统上出现，因此该模式也无法支持在实际运行系统中的调试。

（4）片上调试（On Chip Debugging）交叉调试方法：该方法和 Stub 类似，片上调试把 Stub 模块做在一个特殊的硬件设备上(我们称之仿真器)，仿真器通过串口、以太口、usb 和主机相连，同时仿真器还通过专用的接口（比如 JTAG）和目标板上的处理器相连接，片上调试需要在处理器内部嵌入额外的控制模块，当满足了一定的触发条件时进入某种特殊状态。在该状态下被调试程序停止运行，主机的调试器可以通过专用接口（比如 JTAG）访问各种

资源（寄存器、存储器等）并执行指令。内嵌的控制模块以基于微码的监控器（microcode monitor）或纯硬件资源的形式存在，包括一些提供给用户的接口（如断点寄存器等）。该方法存在以下缺陷:

（41）该方法需要 CPU 支持调试模块，现有的 CPU 并不一定都支持调试模块。

（42）这种方案需要仿真器，仿真器价格昂贵不适合大规模使用，

（43）目标板上需要可供仿真器连接的接插件，由于接插件会占用物理空间，而且对信号有影响，因此实际产品时不会预留供调试用的接插件接口，另外调试时需要预先连接好仿真器，因此该调试方案也不适合在实际产品上的调试。

综上所述，现有嵌入式系统的故障诊断和调试方案都有各自优缺点：但都存在如下问题:

无法解决在实际系统上的直接进行故障诊断和调试功能;

无法支持对未知故障的现场诊断功能;

无法对未知协议的分析功能;

而且需要特殊的软、硬件支持，增加投入，因此不利于大规模采用;

软件开发投入大，灵活性差，扩展性差。

发明内容

本发明的目的是针对上述现有技术的缺陷，提出了内嵌调试器的嵌入式系统以及嵌入式系统调试方法，使得处于目标设备上的嵌入式系统，同时包括嵌入式操作单元以及调试单元，以实现在嵌入式系统上直接对软件和/或硬件进行调试，从而降低了设置额外硬件和/或开发额外的软件而导致的成本。

为实现上述目的，本发明提供了一种内嵌调试器的嵌入式系统，包括嵌入式操作单元，其中还包括调试单元，与所述嵌入式操作单元连接；所述调试单元包括：驱动模块以及与驱动模块连接的调试模块；驱动模块，用于控

制调试命令的输入和输出；调试模块，用于触发嵌入式操作单元产生异常，捕获异常，以及对异常信息进行分析处理，根据异常信息的类型选择预设方法对嵌入式操作单元中的硬件和/或软件进行调试。

为实现上述目的，本发明还提供了一种嵌入式系统调试方法，其中包括：调试模块触发嵌入式操作单元产生异常；调试模块捕获异常，对异常信息进行分析处理，根据异常信息的类型选择预设方法；调试模块采用所述预设方法对嵌入式操作单元中的硬件和/或软件进行调试。

由以上技术方案可知，本发明一种内嵌调试器的嵌入式系统及嵌入式系统调试方法，通过在嵌入式系统中内嵌调试单元，使得处于目标设备上的嵌入式系统，同时包括嵌入式操作单元以及调试单元，以实现在嵌入式系统上直接对软件和/或硬件进行调试，从而降低了设置额外硬件和/或开发额外的软件而导致的成本，方便大规模的推广应用。同时由于嵌入式操作单元与调试单元位于同一目标设备上，因此当嵌入式操作单元出现故障或需调试时，可以直接利用调试单元进入调试模式，从而加快了疑难问题的解决速度。

附图说明

图 1 为本发明一种内嵌调试器的嵌入式系统实施例一的结构示意图；

图 2 为本发明一种内嵌调试器的嵌入式系统实施例二的结构示意图；

图 3 为本发明一种内嵌调试器的嵌入式系统实施例三的结构示意图；

图 4 为调试模块的一结构示意图；

图 5 为本发明一种嵌入式系统调试方法的流程图；

图 6 为调试模块的另一结构示意图。

具体实施方式

下面通过附图和实施例，对本发明的技术方案做进一步的详细描述。

图 1 为本发明一种内嵌调试器的嵌入式系统实施例一的结构示意图。该

实施例中内嵌调试器的嵌入式系统 200，安装在目标设备 100 上，包括嵌入式操作单元 210，其中还包括与嵌入式操作单元 210 连接的调试单元 220。嵌入式操作单元 210 包括：彼此互相连接的驱动模块 211、嵌入式操作系统 212 以及应用程序 213。调试单元 220 包括：驱动模块 221 以及与驱动模块 221 连接的调试模块 222；驱动模块 221，用于控制调试命令的输入和输出；调试模块 222，用于触发嵌入式操作单元 210 产生异常，捕获异常，以及对异常信息进行分析处理，根据异常信息的类型选择预设方法嵌入式操作单元 210 中的硬件和/或软件进行调试。调试模块 222 触发嵌入式操作单元 210 产生异常具体为：调试模块 222 对目标设备 100 和/或嵌入式操作单元 210 的访问和/或控制，如对目标设备 100 和/或嵌入式操作单元 210 的寄存器或者地址空间进行数据读写，修改，配置，通过修改或设置相应的数据对目标设备 100 和/或嵌入式操作单元 210 进行控制，如设置断点、在断点处设置特殊指令来产生异常。嵌入式操作系统是固化在硬件里面的系统，比如手机、路由器里面的系统。常见的嵌入式操作系统有 Linux、uClinux、WinCE、PalmOS、Symbian 等。

一般情况下，调试模块在接收到调试命令的通知时，才进行调试操作。用户可以根据测试需要或开发经验，手动地通过人机操作界面的方式选择或输入调试命令。嵌入式操作单元也可以根据其内部的自检功能模块，定时地向调试模块发送调试命令的通知；或实时地根据嵌入式操作单元的运行状态，当检测到故障状态或准故障状态时，自动地向调试模块发送调试命令的通知。

图 2 为本发明一种内嵌调试器的嵌入式系统实施例二的结构示意图。该实施例与实施例一的区别在于，调试单元 220 还包括调试界面 223，驱动模块 221 与调试界面 223 连接，用于控制调试界面 223 的显示和调试命令的输入和输出；调试模块 222 与调试界面 223 连接，用于接收并响应由调试界面 223 发送的调试命令。该实施例中用户通过调试界面 223 输入调试命令或用户通过调试界面 223 选择一条调试命令，然后由调试界面 223 将相应的输入或选择的调试命令发送至调试模块 222。

图 3 为本发明一种内嵌调试器的嵌入式系统实施例三的结构示意图。该实施例与实施例一的区别在于，嵌入式操作单元 210 还包括通知模块 214，用于向调试单元 220 发送调试命令。该实施例中，嵌入式操作单元 210 中的通知模块 214 可以定时地向调试单元 220 中的调试模块 222 发送调试命令的通知；或实时地根据嵌入式操作单元的运行状态，当检测到故障状态或准故障状态时，自动地向调试模块发送调试命令的通知。

图 4 为调试模块的一结构示意图。调试模块包括：异常处理子模块 A 以及与所述异常处理子模块 A 连接的调试功能子模块 B。异常处理子模块 A，捕获异常，对异常信息进行分析处理，根据异常信息的类型将异常信息发送至调试功能子模块；调试功能子模块 B，用于触发嵌入式操作单元产生异常，并与所述异常处理子模块连接，用于采用所述调试功能子模块内的预设方法对嵌入式操作单元中的硬件和/或软件进行调试。其中调试功能子模块又可以包括但不限于固定断点处理子模块 B1、动态断点处理子模块 B2、单步跟踪处理子模块 B3、数据监视处理子模块 B4 及指令监视处理子模块 B5 中的任一种或其组合。

固定断点处理子模块 B1，用于采用固定断点实现方法对嵌入式操作单元中的硬件和/或软件进行调试。固定断点实现方法中，断点的位置在开发阶段就决定，一般用于调试系统的初始化，系统运行到固定断点位置可以产生固定断点异常，系统暂停，并等待用户调试。

动态断点处理子模块 B2，用于采用动态实现方法对嵌入式操作单元中的硬件和/或软件进行调试。动态断点实现方法中，可以在系统运行后通过命令的方式添加/删除断点，系统运行到动态断点位置可以产生动态断点异常，系统暂停，并等待用户调试。

单步跟踪处理子模块 B3，用于采用单步跟踪实现方法对嵌入式操作单元中的硬件和/或软件进行调试。单步跟踪实现方法中，系统每执行一步可以产生单步跟踪异常，系统暂停，并等待用户调试。

数据监视处理子模块 B4，对嵌入式操作单元中的数据访问进行监控，监视系统对指定空间的访问，如果指定空间被访问，系统产生数据监视异常，报告监视地址被访问，系统暂停，并等待用户调试。

指令监视处理子模块 B5，对嵌入式操作单元中的指令执行进行监控，监视指定空间的指令是否被运行，如果指定空间的指令被运行，系统产生指令监视异常，报告监视地址被运行，系统暂停，并等待用户调试。

图 5 为本发明一种嵌入式系统调试方法的流程图。如图 5，其中包括:

步骤 1、嵌入式操作单元或用户通过调试界面向调试模块输入调试命令;

步骤 2、调试模块响应调试命令，触发嵌入式操作单元产生异常;

步骤 3、调试模块捕获异常，对异常信息进行分析处理，根据异常信息的类型选择预设方法;

步骤 4、调试模块采用所述预设方法对嵌入式操作单元中的硬件和/或软件进行调试。

所述预设方法为固定断点实现方法、动态断点实现方法或单步跟踪实现方法、数据监视实现方法、指令监视实现方法或其他实现方法。

下面分别对上述技术方案中提到的异常处理、固定断点方法、动态断点实现方法、单步跟踪实现方法、数据监视实现方法以及指令监视实现方法进行描述。

（1）异常处理原理

CPU 执行指令是按照顺序执行的，当有外部事件产生时比如中断（在 powerpc 中中断也是一种异常），CPU 要先暂停当前程序的执行，转而执行每个异常对应的入口程序，当异常程序退出后，CPU 能够恢复到刚才的位置继续执行，比如当程序执行到 100 位置时，产生一个外部中断，这是 CPU 直接跳转到中断所对应的异常入口（如 powerpc: 0x900）当中断处理完毕后退出中断，程序继续从 100 处开始运行，而在 powerpc 中，多种事件可以产生异常，可以采用如下异常用于调试目的:

异常	异常向量	用途	说明	使用的寄存器和指令（如何产生异常）	调用的处理模块
Program	0x0700	固定断点产生异常	当异常产生时SRR0＝产生异常的指令地址，SRR0+4就是用户用断点的地址	使用 twge r2，r2 指令作为固定断点指令。	固定断点处理子模块
System Call	0x0C00	动态断点产生异常	当异常产生时SRR0＝产生异常的指令地址，SRR0+4就是用户断点的地址	使用 sc 指令作为动态断点指令。	动态断点处理子模块
Trace Exception	0x00D00	单步跟踪产生异常	当异常产生时SRR0＝下一次将被执行的指令地址	设置MSR[SE] ＝1:，每执行该指令就会产生该异常	单步跟踪处理子模块
Data Storage	0x0300	用户监视CPU对数据的访问，当CPU访问（读/写）的地址空间落在监视范围内时产生该异常	当异常产生时SRR0＝产生异常的指令地址 DAR＝访问的数据地址。	DABR和DABR2用于设定监视的地址和访问方式 DBCR：用于设置监视地址匹配和组合方式	数据监视处理子模块
Instruction	0x01300	用户监视	当异常产生时	IABR和	指令监视处理

Storage		CPU 的执行，当 CPU 执行指令的地址空间落在监视范围内时产生该异常	SRR0＝产生异常的指令地址	IABR2用于设定监视的地址和访问方式IBCR:用于设置监视地址匹配和组合方式	子模块

对于不同的异常有不同的异常处理入口（异常向量），调试模块通过改写异常向量的指令（可以是开发阶段就改写，也可以在运行阶段才改写），让 CPU 产生异常时执行的是调试系统的指令，这样调试系统就接管的 CPU 的运行。

当 CPU 跳转到异常向量，并执行异常向量的程序时，CPU 会在特定的位置记录程序返回的地址（异常的地址），并且记录其他有用的信息，调试模块可以获取这些信息进行处理。

为了让 CPU 执行完异常处理后能够返回到原来的位置，因此进入异常后要进行现场保护，退出异常前再恢复现场。异常处理流程如下:

a1）保护现场（保存异常时 CPU 的相关寄存器到内存），

a2）调用调试系统程序

a3）调试系统程序，分析异常的类型，获取所需要的异常前的信息，并调用其它处理子模块进行处理。

a4）调试系统退出时复现场（从内存中获取，并且改写 CPU 的寄存器，这样现场恢复后 CPU 的所有寄存器就和异常前一样，CPU 如同没有发生异常一样），让 CPU 继续从原来的位置继续执行。

（2）固定断点实现方法

固定断点为开发人员经常使用。开发人员在开发程序时，在希望产生断点的程序位置插入一行，由调试系统提供固定断点程序（例如开发人员希望

程序执行到第 100 行停下来，那么开发人员需要在第 100 行代码中插入 SET_BREAKPOINT 这行程序），该固定断点程序和其他程序被一起编译链接，并被加载到目标系统中运行，当程序执行到固定断点位置时程序就会停下来。

实际上该行程序（固定断点程序）是一条可以产生异常的特殊指令，当目标系统执行该指令会产生异常，而该异常会产生异常中断，产生异常中断后会被调试系统接管，后续调试系统就可以对系统进行调试，（比如通过宏的形式来代替 #define SET_BREAKPOINT twge r2, r2,mpc8248 CPU(freescale 公司的一款通讯 CPU）中可以是 twge r2,r2 指令,当 CPU 执行到该指令时产生一个 progame 异常（异常向量为 0x700）。

在进入异常处理前 CPU 会把产生异常指令的地址记录在 srr0 寄存器中，并进入调试模式。(记录第 100 行程序的位置在 srr0 寄存器中，而程序继续执行时需要从 101 开始继续执行,因此 srr0 寄存器的值+4 就等于断点的下一行程序位置（在 powerpc 中指令每条指令占用 4 个字节），程序退出调试模式（断点继续执行），从 srr0 寄存器的值+4 的位置开始执行。

在进入调试模式后，调试系统可以根据产生异常的位置(比如 0x700)，知道是固定断点异常，根据 srr0 的值就可以知道程序执行到何处停下来，继续执行时该从何处继续执行（srr0 寄存器的值+4）。进入调试命令模式后，以字符命令或者图形的方式显示断点的位置，调试系统指示程序在固定断点停下来（可以指示源代码的位置），并等待用户的下一步调试操作。

在调试模式下等待用户输入命令进行调试（比如查看内存情况，查看寄存器状态，查看源代码等操作）。

用户调试完毕后，希望程序从断点继续执行，用户可以通过特殊的调试命令，退出调试模式，从断点继续执行（比如通过 quit_breakpoint 命令），调试系统接收到该调试命令后，进行现场恢复后（恢复异常前所有相关的寄存器，保证异常前和退出异常后寄存器都一致）通过设置特殊的寄存器和执行特定的指令，让程序从断点出继续执行。（比如在 mpc8248 中，设置 srr0

寄存器为第 101 行程序的地址，并且最后执行 ri 指令，那么 CPU 就会从 101 行开始执行）

程序从断点恢复后 CPU 进入嵌入式模式，继续执行嵌入式系统的指令。

对于不同的 CPU 有不同的指令来产生固定断点异常，固定断点异常时断点保存在哪一个寄存器中不同 CPU 也不一样，但是所有 CPU 的原理都类似，就是让一条特殊的指令产生固定断点异常，而产生异常时固定断点的位置会记录在 CPU 的寄存器中，调试模块就是根据这些信息进行调试。

（3）动态断点实现方法

动态断点的原理和固定断点原理一样，也就是在断点处用一条特殊的指令来产生异常，由调试模块捕捉和分析该异常，可以知道是动态断点产生异常，并知道何处产生异常，程序该从何处继续执行；不同的地方在于动态断点不是在程序开发的时候确定（不需要写程序），而是等待程序运行起来后，在嵌入式模式或者调试模式下，输入特殊的命令，添加动态断点的位置，因此动态断点不需要预先开发，可以在程序运行后动态添加，过程如下:

b1）在嵌入式系统模式（嵌入式系统中需要添加一条动态断点添加的命令）或者在调试模式下，通过某个特殊的调试命令，用户添加一条动态断点（例如: add_breakpoing [breakpoint address]，这里的 breakpoint address 可以通过查看编译器产生的 MAP 文件-源代码和二进制代码对应关系文件，得到某一行源代码对应的执行地址，或者通过后文介绍的“源代码级别调试实现方法”直接查看运行时的源代码及对应二进制代码的运行地址）。

b2）该添加调试动态断点的命令，会获取用户输入的动态断点的位置（也就是 breakpoing address）。

b3）保存断点处原来的程序指令（把 breakpoing address 对应的指令保存起来）。

b4）把断点处指令替换成可以产生异常的特殊指令（比如把原来 breakpoint address 位置的指令替换成 mpc8248 的 SC 指令- - 系统调用指

令)。

b5）当程序执行到断点处（breakpoint address），由于执行特殊指令，会产生一个异常，在进入异常处理前 CPU 会把产生异常指令的下一条指令的地址记录在 srr0 寄存器中，也就是说 srr0 的寄存器保留的是断点位置+4,因此产生异常的指令位置为 srr0 寄存器值-4，并进入调试模式。

b6）进入调试命令模式后，以字符命令或者图形的方式显示断点的位置，调试系统指示程序在动态断点停下来（可以指示源代码的位置），并等待用户的下一步调试操作。

b7）在调试模式下等待用户输入命令进行调试（比如查看内存情况，查看寄存器状态，查看源代码等操作）。

b8）用户调试完毕后，希望程序从断点继续执行，用户可以通过特殊的调试命令，退出调试模式，从断点继续执行（比如通过 quit_breakpoint 命令），调试系统接收到该调试命令后， 进行现场恢复后（恢复异常前所有相关的寄存器，保证异常前和退出异常后寄存器都一致），还要把 breakpoint address 位置的指令恢复成原来的指令(因为在用户设置断点时已经设置成特殊指令），并且让程序继续从 breakping address 执行让程序从断点出继续执行。例如 mpc8248 的处理方法是 srr0 设置成原来 srr0 寄存器的值－4，并且调用 rti 返回到原来的断点处极性执行。

b9）程序从断点恢复后 CPU 进入嵌入式模式，继续执行嵌入式系统的指令。

B10）因为在退出异常时动态断点的指令已经被替换成正常的指令，如果下一次程序执行到该位置就再不会产生断点异常，因此需要一种方法让它退出异常后，后续如果再执行到该位置，仍然可以产生动态断点（除非用户删除动态断点），处理方法是程序重新执行完真正的断点处指令后，让它产生一个单步跟踪异常，在单步跟踪异常处理中，又重新把断点位置的指令替换成特殊指令，使它下一次能够继续进行断点跟踪。

b11）当然动态断点也可以删除，删除方法比较简单，只要把断点处的指令换成原来的指令就可，这样程序指令到该位置时指令的是正常的指令，不会产生断点异常。断点就不存在。

b12）对于不同的CPU有不同的指令来产生动态断点异常，动态断点异常时断点保存在哪一个寄存器中不同 CPU 也不一样，但是所有 CPU 的原理都类似，就是让一条特殊的指令产生动态断点异常，而产生异常时动态断点的位置会记录在CPU的寄存器中，调试模块就是根据这些信息进行调试。

（4）单步跟踪实现方法

单步跟踪的原理是通过设置 CPU 的特殊寄存器，让 CPU 每执行完一条指令产生异常，调试模块接管该异常，调试模块根据异常的信息知道这是单步跟踪异常，程序执行的位置等信息，实现方法如下:

c1）用户在调试模式的命令下输入单步跟踪命令（比如输入 trance）。

c2）同固定断点和动态断点处理类似，调试模块退出，让CPU从上一次断点处继续执行，但是不同的地方是在退出前需要设置 CPU 特殊的寄存器，让 CPU 每执行完毕一条指令就会产生一个单步跟踪异常，比如 MPC8248 的处理方法是通过MSR[SE]寄存器(该寄存器设置CPU每执行完毕一条指令就会产生一个单步跟踪异常），这里还有一个特殊的地方是，在调试模块不直接设置MSR[SE]寄存器，因为如果直接设置MSR[SE],那么在调试模式下每执行一条指令也会产生一个单步异常，这是不期望的；对于 MPC8248 的处理方式是设置SRR1寄存器，因为在调用 rti 指令是SRR1寄存器会被赋值给MSR寄存器，这样当程序开始从原来断点处继续执行是MSR[SE]才被设置，因此可以产生单步跟踪异常。

c3）当程序退出异常（调试模式）继续执行嵌入式程序时，由于CPU已经被设置使能单步跟踪功能，因此当 CPU 每执行完毕一条指令就会产生单步跟踪异常。

c4）在进入异常处理前CPU会把下一条指令的地址记录在 srr0 寄存器中。

c5）进入调试模式，清楚CPU的单步跟踪异常使能位，让CPU退出调试模式继续运行用户模式时不会继续产生单步跟踪。

c6）进入调试命令模式后，以字符命令或者图形的方式显示断点的位置，调试系统指示程序在单步跟踪位置（可以指示源代码的位置），并等待用户的下一步调试操作。

c7）在调试模式下等待用户输入命令进行调试（比如查看内存情况，查看寄存器状态，查看源代码等操作）。

c8）用户调试完毕后，希望程序从断点继续执行，用户可以通过特殊的调试命令，退出调试模式，从断点继续执行（比如通过quit_breakpoint命令），由于srr0保存的断点处下一条将要执行的指令，因此退出异常时程序直接从srr0指示的位置继续执行。

c9）程序从断点恢复后CPU进入嵌入式模式，继续执行嵌入式系统的指令。

c10）对于不同的CPU有使能CPU单步跟踪功能的方法不一样，异常时断点保存在哪一个寄存器中不同CPU也不一样，但是所有CPU的原理都类似，就是使能CPU的单步跟踪功能（要求CPU支持该功能），让CPU每执行一条指令产生一次单步跟踪异常，而产生异常时断点的位置会记录在CPU的寄存器中，调试模块就是根据这些信息进行调试。

c11）以上所诉的单步跟踪指的时指令级的单步跟踪，很多源代码级别的单步跟踪（每执行完毕一行源代码才停下来，而一条源代码可以对应多条指令）可以通过添加、删除动态断点的方式来实现（在每行源代码的位置自动添加一个动态断点），不同的动态断点的删除和添加都是通过调试模块实现，无需通过用户来操作。

（5）数据监视实现方法

数据监视的方法是通过设置CPU的特殊寄存器(要求CPU支持该功能)，当程序访问指定位置的数据是，会产生数据监视异常，调试模块可以获取产生异常的指令的位置，以及产生异常时访问数据的地址以及访问方式，有这些

信息调，试模块就能把结果显示给用户。

d1）在嵌入式系统模式（嵌入式系统中需要添加一条数据监视的命令）或者在调试模式下，通过个添加数据监视命令，用户添加一条数据监视点（例如：add_data_watchpoint [watchpoint address] [mode]）

d2）该添加数据监视点的命令获取用户输入的监视的地址(watchpoint address)和监视方式（mode））

d3）根据地址和监视方式，设置CPU的数据监视寄存器，让CPU访问该地址时产生数据监视异常，（例如在mpc8248 通过设置DABR/DABR2/DBCR 寄存器，可以用来监视数据存储，当 CPU 访问的地址和该寄存器的地址相等时，就会产生异常，异常地址为0x300.）

d4)当程序访问监视地址时，CPU会产生数据监视异常。

d5)CPU 在进入异常前会记录产生异常是指令地址记录和访问的数据地址记录在特定的寄存器中。

d6)调试模块接管异常处理，并且获取产生异常指令的地址，以及访问数据的地址，进入调试模式把讯息显示给用户，并等待用户的下一步调试操作。

d7)在调试模式下等待用户输入命令进行调试（比如查看内存情况，查看寄存器状态，查看源代码等操作）。

d8)对于不同的 CPU 有使能设置数据监视地址和监视方式的方法或者指令以及对应的寄存器地址都不同。

d9)和添加相对应，用户可以通过命令删除数据监视点，删除的方法就是通过清除数据监视寄存器，让CPU不使能数据监视功能。

（6）指令监视实现方法

指令监视的方法是通过设置CPU的特殊寄存器(要求CPU支持该功能)，当该位置的指令被执行时，会产生指令监视异常，调试模块可以获取产生异常的指令的位置，试模块就能把结果显示给用户。

e1)在嵌入式系统模式（嵌入式系统中需要添加一条指令监视的命令）或

者在调试模式下，通过个添加指令监视命令，用户添加一条指令监视点（例如：add_instruction_watchpoint [watchpoint address] [mode]）。

e2)该添加指令监视点的命令获取用户输入的监视的地址(watchpoint address)。

e3)设置 CPU 的指令监视寄存器，让 CPU 执行该地址的指令时产生指令监视异常，（例如在 mpc8248 通过设置 IABR/IABR2/IBCR 寄存器，可以用来监视指令执行，当 CPU 执行指令的地址和该寄存器的地址相等时，就会产生异常，异常地址为 0x1300.）。

e4)当程序执行该地址的指令时，CPU 会产生指令监视异常。

e5)CPU 在进入异常前会记录产生异常是指令地址记录在特定的寄存器中。

e6)调试模块接管异常处理，并且获取产生异常指令的地址，并等待用户的下一步调试操作。

e7)在调试模式下等待用户输入命令进行调试（比如查看内存情况，查看寄存器状态，查看源代码等操作）。

e8)对于不同的 CPU 有使能设置指令监视地址或者指令以及对应的寄存器地址都不同。

e9)和添加相对应，用户可以通过命令删除指令监视点，删除的方法就是通过清除指令监视寄存器，让 CPU 不使能指令监视功能。

为配合实现调试模块的以上功能，调试模块中的调试功能子模块还包括有用于实现反汇编的反汇编处理子模块，用于实现源代码级别调试的源代码调试子模块以及用于查看断点处程序执行的过程的路径跟踪调试子模块，以及用户访问和修改嵌入式操作单元的数据访问子模块中的任一种或其任意组合，以配合固定断点处理子模块、动态断点处理子模块、单步跟踪处理子模块、数据监视处理子模块或指令监视处理子模块以完成调试工作。如图 6 所示，调试模块中的调试功能子模块 B 还包括：反汇编处理子模块 B6、源代码

调试子模块 B7 及路径跟踪调试子模块 B8，数据访问子模块 B9。相应地为实现上述实施例中的固定断点方法、动态断点实现方法、单步跟踪实现方法、数据监视实现方法或指令监视实现方法，还需要有反汇编实现方法、源代码调试实现方法、数据访问实现方法、和/或路径跟踪实现方法。如系统暂停后，用户可以通过调试命令的方式触发汇编、源代码调试子模块查看断点处或者其他位置的程序的汇编或者源代码，用户可以通过调试命令的方式触发路径跟踪子模块，查看断点处系统运行的过程，可以通过数据访问子模块查看/修改嵌入式操作单元的数据。下面分别对反汇编实现方法、源代码调试实现方法、以及路径跟踪实现方法，数据访问实现方法进行描述。

（7）反汇编实现方法

程序执行的都是二进制代码，但是每个 CPU 能执行的指令，指令都有固定的格式，根据这个格式可以汇编指令翻译成二进制代码，也可以把二进制代码翻译成汇编代码。实现过程如下:

f1）在嵌入式系统模式（嵌入式系统中需要添加一条反汇编命令）或者在调试模式下，用户输入一条反汇编命令，并且指明需要反汇编代码的位置（例如 assemble [instruction address]）。

f2）读取指定位置的指令内存中的指令。

f3）根据 CPU 指令格式，把该二进制指令翻译成汇编指令，并且显示给用户。

f4）可以在调试模式直接反汇编断点处的指令，这时用户不用指定反汇编地址，因为调试模块已经知道断点的位置。其中不同的 CPU 指令格式不同。

（8）源代码调试实现方法

在编译程序时可以通过设置编译器的选项让编译器编译链接产生的文件有附带源代码信息，也就是该文件会记录每条二进制代码对应的源代码的位置，以及源代码的内容，这样就可以根据程序执行的位置找到对应的源代码，如果要支持该功能嵌入式系统中还需要保存该文件。

g1)在嵌入式系统模式（嵌入式系统中需要添加一条源代码调试命令）或者在调试模式下，用户输入一条源代码调试命令，并且指明需要调试代码的位置（例如 source [instruction address]）。

g2)读取指定位置的指令内存中的指令(instruction address)。

g3)查找二进制代码和源代码对应关系的文件，找到源代码，并且把源代码信息显示给用户。

g4)可以在调试模式直接查看断点处的源代码，这时用户不用指定指令地址，因为调试模块已经知道断点的位置。

（9）路径跟踪调试实现方法

当程序执行到断点时，CPU 只知道断点指令的位置，但是不知道程序调用的过程，也就是说该断点对应的函数是被什么函数调用的，路径跟踪就是要知道断点位置的程序是被什么上级函数调用，它的函数又被什么函数调用，依此类推直到找到整个调用过程的源头，反过来可以知道从源头一直到断点函数调用关系。路径跟踪实现过程如下:

h1)用户通过如上介绍的功能模块或者命令，设置断点（动态或者固定），或者设置数据或者指令监视，导致 CPU 产生异常。

h2)CPU 产生异常后调试模块接管异常，并按如上描述的功能对异常进行处理。

h3)CPU 获取异常指令的位置。获取 SP 堆栈信息，则 *sp 的值指向上一个函数的堆栈指针，**sp 指向上上一个堆栈指针，以次类推，而*(sp+4)指向上一个函数，*(*sp+4) 指向上上一个函数，以次类推，获取调用路径对应的指令地址，通过反汇编模块或者源代码查看模块再显示处调用过程对应的汇编代码或者源代码。

（10）数据访问实现方法

用户输入命令后，该命令直接使用向指定的空间写入、读出数据，并且把结果显示给用户。对于不同 CPU，不同空间的写入和读出方法不一样。也可以采用嵌入式操作单元提供的接口，访问或者修改嵌入式操作单元数据。

本领域普通技术人员可以理解:

(1)实现上述方法实施例的全部或部分步骤可以通过程序指令相关的硬件来完成,前述的程序可以存储于一计算机可读取存储介质中,该程序在执行时,执行包括上述方法实施例的步骤;而前述的存储介质包括:ROM、RAM、磁碟或者光盘等各种可以存储程序代码的介质。

(2)以上实施例所述的调试单元不仅可以在具有嵌入式操作系统的嵌入式系统中实现,也可以在具有通用操作系统的通用系统中实现,或在裸机中实现,只要在添加调试单元后,不影响正常的程序(目标设备被最终用户实际使用的程序)的情况均属于本发明保护的范围。

(3)上述实施例中所描述的调试单元的功能以及功能模块仅用于说明内嵌调试器的嵌入式系统及嵌入式系统调试方法,而非限制,可以根据实际需要,如嵌入式操作单元及目标设备的类型,减少或扩展调试功能,或者结合多种功能模块实现更加高级的调试功能。

(4)本发明不局限于调试单元与嵌入式操作单元完全独立的情况,调试单元也可以与嵌入式操作单元共用嵌入式操作系统及驱动模块。本发明不局限于目标设备的类型以及目标设备的处理器的类型,目标设备可以为计算机、手机终端或路由器等,目标设备的处理器可以为嵌入式 CPU,也可以是通用CPU,或者数字信号处理器(Digital Signal Processing,简称 DSP),或者其它具有可编程能力的器件。

(5)上述实施例中,不同类型的 CPU 可以采用不同异常处理方式,即使针对同一类型的 CPU,如 powerpc,也可以采用不同的异常处理方式。

(6)上述实施例中,调试界面用于获取用户的调试请求,显示调试结果信息,本发明不限制采用何种实现方式,可以使用如 DOS 调试命令的方式,也可以采用 windows 可视化调试界面。

综上所述,本发明一种内嵌调试器的嵌入式系统及嵌入式系统调试方法的实施例具有以下有益效果:

(1)通过在嵌入式系统中内嵌调试单元,使得处于目标设备上的嵌入式

系统，同时包括嵌入式操作单元以及调试单元，以实现在嵌入式系统上直接对软件和/或硬件进行调试，从而降低了设置额外硬件和/或开发额外的软件而导致的成本，方便大规模的推广应用。同时由于嵌入式操作单元与调试单元位于同一目标设备上，因此当嵌入式操作单元出现故障或需调试时，可以直接利用调试单元进入调试模式，从而加快了疑难问题的解决速度。

（2）嵌入式系统中的调试单元与嵌入式操作单元相互独立，调试单元的存在不会影响原有的嵌入式系统，因此实际系统中可以带调试单元，以实现直接在实际系统上对软件和/或硬件的故障诊断和调试。

（3）对现有的嵌入式系统改造简单易行。

（4）支持在现场环境对软硬件已知和未知故障进行诊断和分析，以及支持在现场环境对未知通讯协议的分析和诊断，方便开发人员和维护人员使用。

最后应说明的是：以上实施例仅用以说明本发明的技术方案，而非对其限制；尽管参照前述实施例对本发明进行了详细的说明，本领域的普通技术人员应当理解：其依然可以对前述各实施例所记载的技术方案进行修改，或者对其中部分技术特征进行等同替换；而这些修改或者替换，并不使相应技术方案的本质脱离本发明各实施例技术方案的精神和范围。

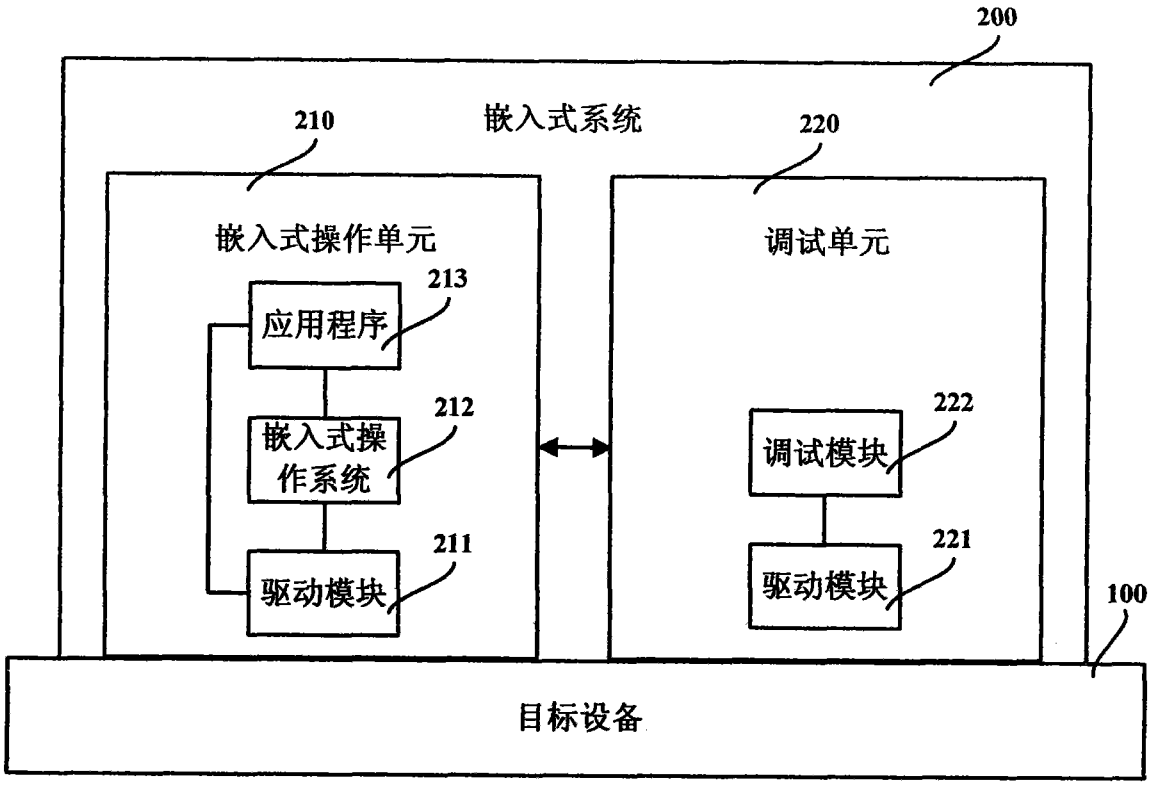

200
嵌入式系统
210
220
嵌入式操作单元
调试单元
213
应用程序
212
嵌入式操作系统
211
驱动模块
222
调试模块
221
驱动模块
100
目标设备

图1

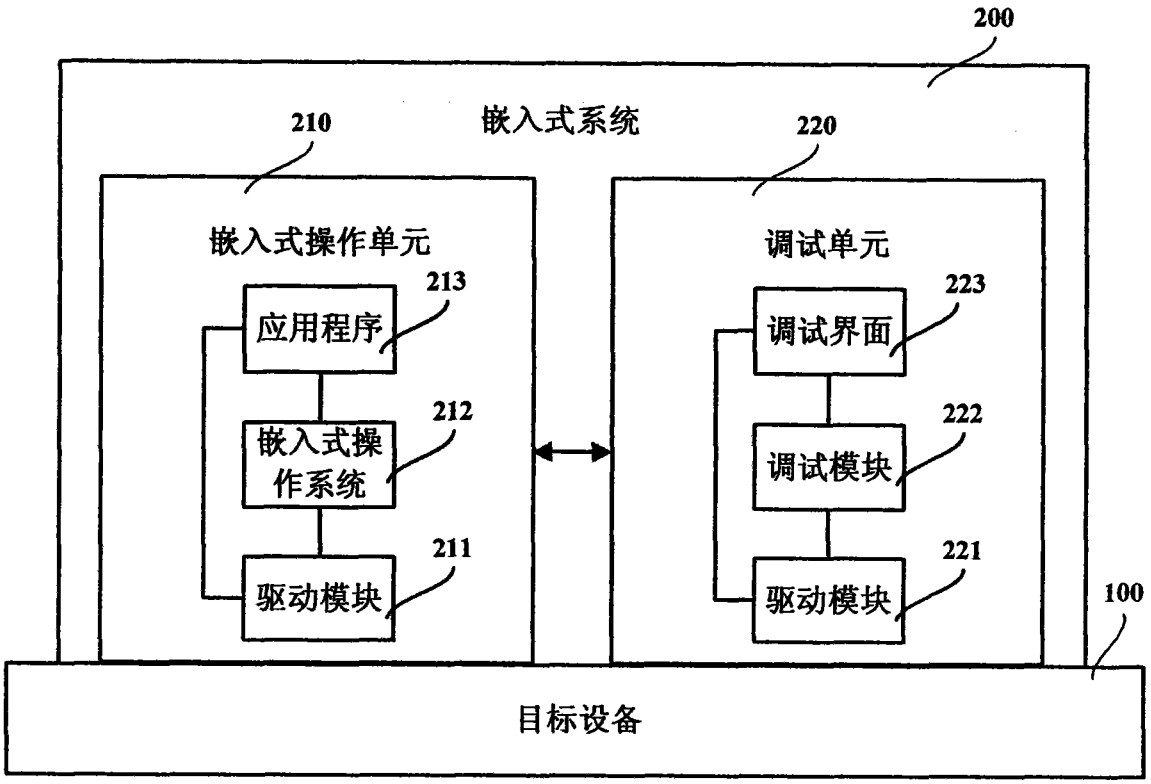

200
嵌入式系统
210
220
嵌入式操作单元
调试单元
213
应用程序
212
嵌入式操作系统
211
驱动模块
223
调试界面
222
调试模块
221
驱动模块
100
目标设备

图2

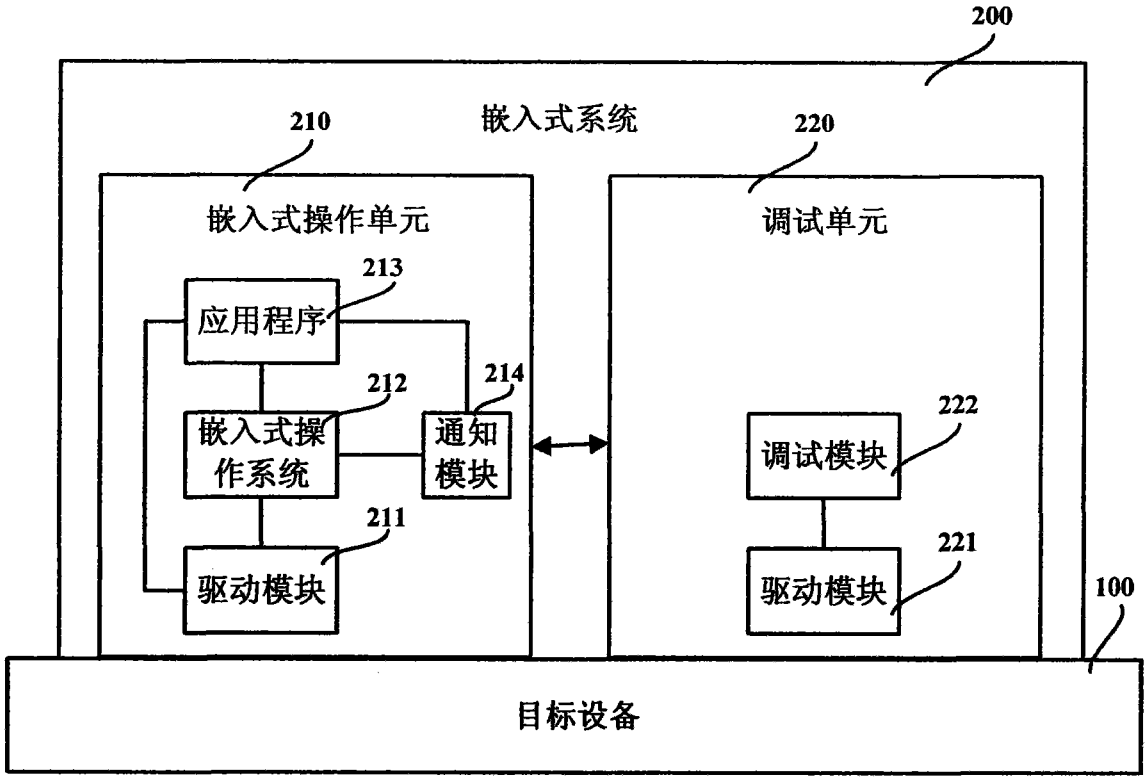

200
嵌入式系统
210
220
嵌入式操作单元
调试单元
213
应用程序
212
214
嵌入式操作系统
通知模块
222
调试模块
211
221
驱动模块
驱动模块
100
目标设备

图3

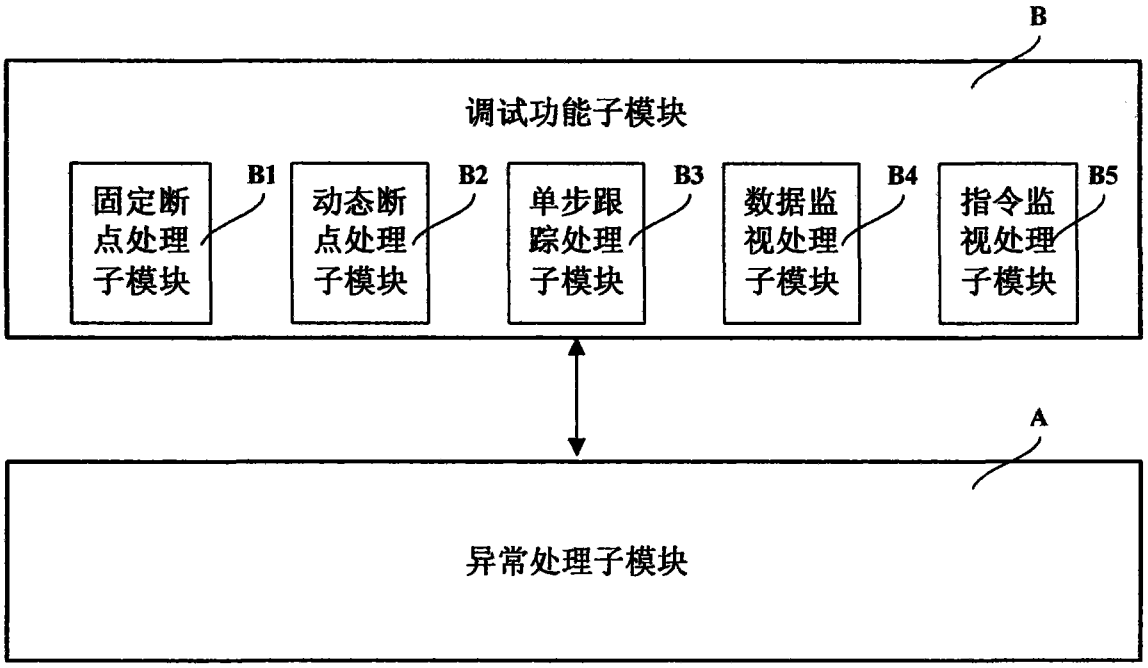

B
调试功能子模块
B1
固定断点处理子模块
B2
动态断点处理子模块
B3
单步跟踪处理子模块
B4
数据监视处理子模块
B5
指令监视处理子模块
A
异常处理子模块

图4

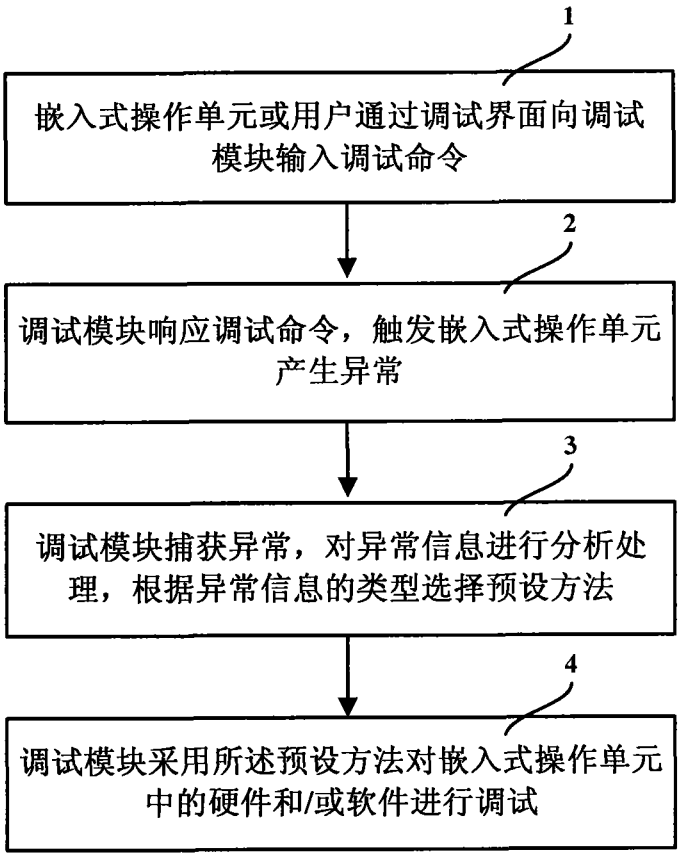

1
嵌入式操作单元或用户通过调试界面向调试模块输入调试命令
2
调试模块响应调试命令，触发嵌入式操作单元产生异常
3
调试模块捕获异常，对异常信息进行分析处理，根据异常信息的类型选择预设方法
4
调试模块采用所述预设方法对嵌入式操作单元中的硬件和/或软件进行调试

图5

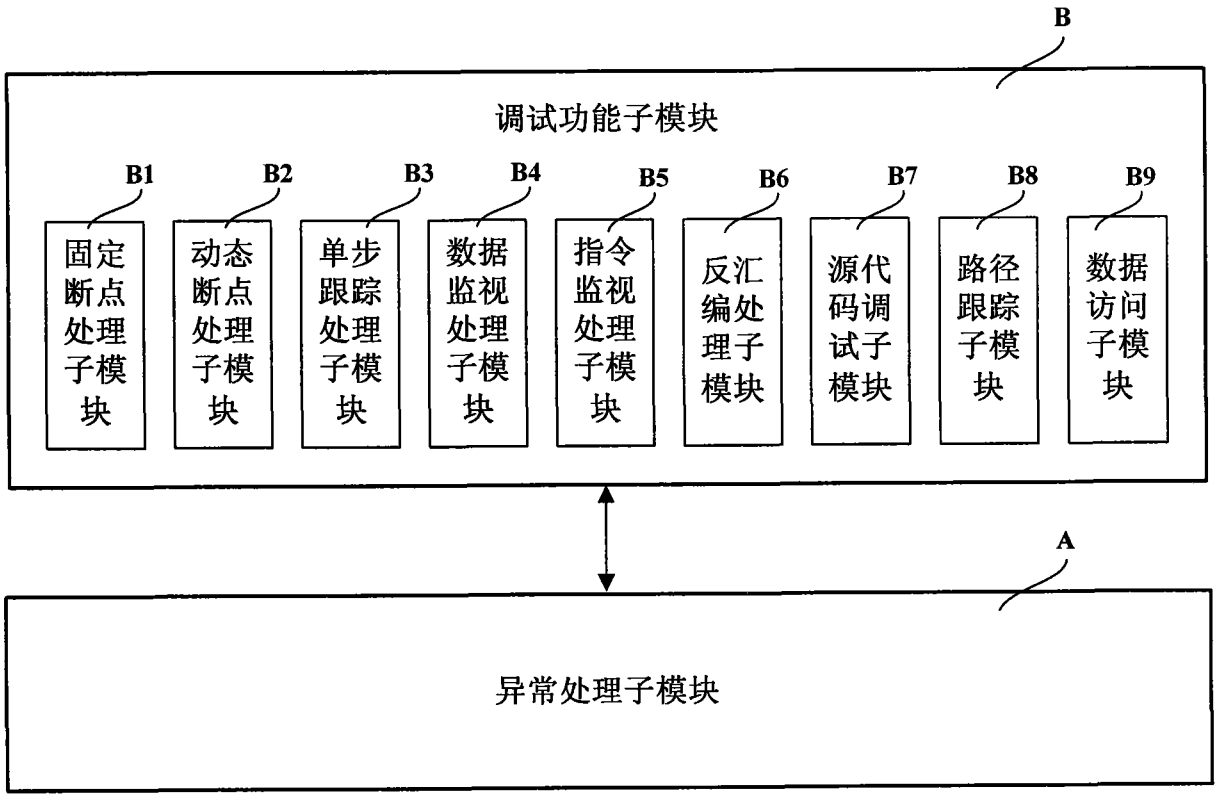

B
调试功能子模块
B1
固定断点处理子模块
B2
动态断点处理子模块
B3
单步跟踪处理子模块
B4
数据监视处理子模块
B5
指令监视处理子模块
B6
反汇编处理子模块
B7
源代码调试子模块
B8
路径跟踪子模块
B9
数据访问子模块
A
异常处理子模块

图6