第33卷 第20期 计 算 机 工 程 2007年10月
Vol.33 No.20 Computer Engineering October 2007
·工程应用技术与实现· 文章编号：1000—3428(2007)20—0239—03 文献标识码：A 中图分类号：TN911.72

基于总线的多DSP交叉调试器的设计与实现

王白江，蔡惠智，冯欣欣，康通博
(中国科学院声学研究所，北京 100080)

摘 要：探讨了在使用总线与主机连接的多DSP系统上构建交叉调试器的方法，分析了在多DSP系统中使用硬件仿真器进行调试的不足，讨论了在DSP上实现软件调试器的基本原理。针对DSP资源有限的特点，提出了动态monitor的方式降低调试器对DSP的资源占用。测试和使用表明，与硬件仿真器相比，基于总线的软件调试器具有更好的性能、更高的性价比以及更好的多处理器支持。

关键词：交叉调试器；动态monitor；DSP

Design and Implementation Techniques of Cross Debugger in Bus-based Multi-DSP System

WANG Bai-jiang, CAI Hui-zhi, FENG Xin-xin, KANG Tong-bo
(Institute of Acoustics, Chinese Academy of Sciences, Beijing 100080)

【Abstract】 This paper discusses the methods about how to construct a cross debugger in bus-based multi-DSP system. Some drawbacks of hardware emulator are mentioned. The rationale of implementation of debugger for DSP is discussed. It presents a method called dynamic monitor to minimize the footprint of the debugger, shows the test result to prove that the bus-based cross debugger has better performance and multiprocessor support than hardware emulator.

【Key words】 cross debugger; dynamic monitor; DSP

1 概述

近年来随着DSP处理器技术的进步，越来越多的系统开始倾向于采用多DSP并行处理的方式来构建系统的信号处理模块。采用多DSP并行处理技术使系统具有很好的伸缩性和灵活性[1]。由多处理器组成的阵列信号处理系统，使通信、雷达、声纳进入数字化飞速发展的时代。

软件调试是多DSP系统开发过程中的重要一环，软件开发周期和软件质量很大程度上依赖于开发人员所使用调试器的性能及调试工具的完备程度[2]。因此，多DSP调试器的研究对于多DSP系统的开发起着至关重要的作用。

通常，硬件仿真器是对DSP软件进行调试的主要工具。目前大多数的DSP和其相应的硬件仿真器都使用JTAG(Joint Test Action Group)调试接口，即标准测试访问接口与边界扫描结构。与x86 等通用处理器系统中软件调试器得到广泛使用的情况不同，用于DSP调试的软件调试器很稀少，这是因为DSP处理器资源有限，交叉调试常用的GDB等调试器一般无法在DSP上工作。虽然硬件仿真器不占用DSP的片内资源，对DSP程序运行干扰很小，并且能够获取底层的硬件信息，但是硬件仿真器也有许多缺点：

(1)JTAG 仿真器同时支持的 DSP 数量有限，调试多 DSP 并行的系统比较困难。JTAG 调试时将多个 DSP 串联成一个 JTAG 链，链上的 DSP 数量有限。

(2)调试的实时性不强。受 JTAG 硬件接口的限制，调试速度比较慢。JTAG 接口采用串行方式传输数据，速率一般只有每秒几十千位，当 JTAG 链上的设备较多时，通信带宽由链上的 DSP 共享，通信速率严重下降。

(3)硬件仿真器成本较高。

20 世纪 90 年代之前的DSP处理器相对通用CPU来说提供的控制指令较少，对软件调试的支持也很少。但随着DSP处理器的不断发展，上述缺点逐渐得到了改善，在DSP上实现功能丰富的软件调试器变为可能。相对硬件仿真器，软件调试器大大提高了多处理器调试的便利性，提高了调试的响应速度，降低了开发成本[3]。

DSP 调试属于交叉调试，调试器运行在 PC 机端而被调程序运行在 DSP 上。被调 DSP 必须能够通过某种物理连接与主机通信。在多 DSP 并行处理的应用场合，主机一般通过总线与 DSP 硬件通信。本文提出一种适合 DSP 的基于总线的软件调试器，使得软件调试器能够在大部分的软件调试中取代硬件仿真器。

2 多DSP交叉调试器基本原理

2.1 影响调试器性能的因素

调试器的性能优劣主要体现在调试器对被调试程序的透明性上，即调试器对被调试程序的影响的大小上。实现 DSP 交叉调试器对被调试程序的透明需要考虑以下几个方面：

(1)资源占用。软件调试器要尽可能少占用 DSP 资源，特别是 DSP 的片上内存资源。DSP 处理器的内存一般只有几十千字节到几百千字节。如果软件调试器的内存占用不够小，那么程序开发人员就必须考虑调试器的内存占用，这势必会增加程序设计者的负担。本文提出的软件调试器可以将内存占用压缩到只有几个字的长度。

作者简介：王白江(1980－)，男，博士研究生，主研方向：阵列信号处理，多处理器调试环境；蔡惠智，研究员、博士生导师；冯欣欣，助理研究员；康通博，博士

收稿日期：2006-10-25 **E-mail：**jason20042008@hotmail.com

(2)实时性。调试器的实时性主要体现在调试器对调试命令的响应速度以及调试器中止和恢复被调程序的切换时间。对调试命令响应速度的要求一般不会太高，通常情况下响应时间在毫秒级就已足够。而调试器中止和恢复被调试程序的切换时间则会影响到被调程序的实时性。实时性和资源占用是矛盾的，二者往往不可兼得。

(3)兼容性。调试器的兼容性是指，调试器可以用来调试不同类型的代码且不影响它们的正常工作。对于软件调试器来说，兼容性尤其重要。DSP 应用程序一般会链接一些较为低层的库函数或者驱动，比如 LibC、RTOS 库等，这些库可能会用到 DSP 的某些特权指令，而软件调试器往往也会用到这些指令，因此可能与应用程序冲突。当用户在 DSP 上使用了 RTOS 后，调试器的兼容性还有另一层含义，就是对操作系统的调试支持，即 real time operating system awareness。调试器可以使用户直接查看和修改 RTOS 内核资源。

2.2 基于总线的 DSP 软件调试器结构

多 DSP 阵列信号处理系统的硬件组成结构如图 1 所示。

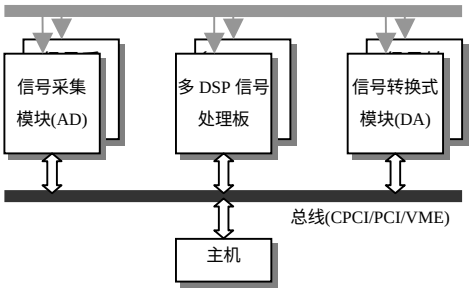

图 1 多 DSP 阵列信号处理系统硬件结构

在此硬件结构基础上，DSP 交叉调试器的结构如图 2 所示。

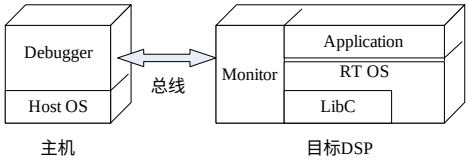

图 2 DSP 交叉调试器结构

DSP 交叉调试器分为两个部分：主机端调试器和 DSP 端的 monitor。Monitor 是调试器在 DSP 上的代理，负责协助主机控制 DSP 的运行并在 DSP 运行状态改变时通知主机。与一般交叉调试器在 target 端的 stub 不同，DSP 调试器的 monitor 只是起控制 DSP 和通知主机的作用，调试信息的收集并不由它来完成。

使用总线调试的好处是主机端的调试器可直接获取它感兴趣的绝大部分 DSP 调试信息，这样就减轻了 monitor 的负担，使得 monitor 的功能和大小都得到了精简。

3 动态 monitor 的基本原理和实现

3.1 结构

Monitor 的主要功能是当主机端调试器感兴趣的事件发生时接管 DSP 运行的控制权并通知主机。通常 monitor 作为一个中断服务程序存在，其结构如图 3 所示。

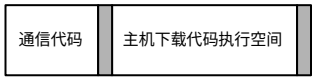

图 3 动态 monitor 的结构

Monitor 所占用的内存空间主要分为两部分。起始部分是一段通信代码，用来通知主机 DSP 上发生了某个调试事件。Monitor 中还有一段空间专门用来保存和执行主机下载的指令。通常的 monitor 含有一些收集寄存器数据的代码，将寄存器中的数据保存在 monitor 的一段内存空间上。即使主机可以通过总线访问 DSP 片上的大部分资源，有些资源(如内部寄存器)通过总线仍然是无法获取的。Monitor 必须将这些寄存器保存到内存中供主机读取。相对通用 CPU 来说，DSP 的寄存器数量众多，读取指令及寄存器数据将占用很大的内存空间，使 monitor 非常臃肿。使用动态 monitor 可以解决内存占用的问题

3.2 动态 monitor 的实现

基于总线的调试方法为构建尺寸非常小的动态 monitor 提供了便利。主机端的调试器可以直接修改 monitor 的指令。在主机代码执行空间的两端，有一组指令限制 PC 指针不会越界，每当主机下载了新的代码，通过修改指令使 DSP 能够运行下载的代码。

Monitor 起始部分的通信代码在每次 DSP 进入 monitor 时运行一次，完成通知主机的任务后通信代码就再无用处。因此通信代码和主机下载代码执行空间可以复用相同的内存空间，进一步缩小 monitor 的尺寸。当主机收到 monitor 的通知后就可以将通信代码部分也作为主机下载代码执行空间使用，在 monitor 将控制返回用户代码之前主机再恢复通信代码部分。

动态 monitor 的本质是将原本 monitor 的代码保存在主机端调试器，由主机端调试器来决定 monitor 何时执行什么指令。这是一种根据程序状态来组装代码的内存复用方法。Monitor 同主机端调试器配合来完成调试。根据主机端调试器的状态，主机将 monitor 的代码设置成不同的形态来完成不同的功能。

Monitor 的状态转换如图 4 所示。

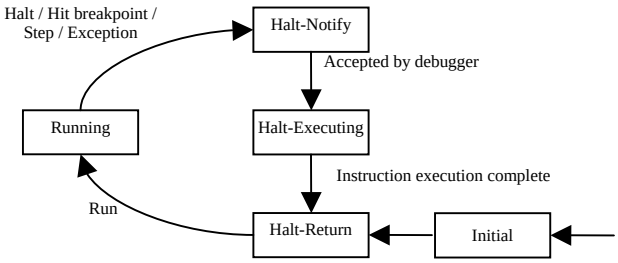

图 4 动态 monitor 状态

3.3 monitor 的植入

被调程序与 monitor 是两个独立的部分，但是要共享相同的 DSP，有两种方法使得它们融合在一起。简单的方法是将 monitor 的代码与被调程序链接到一个可执行文件中。如果一个可执行文件没有链接 monitor，那么主机调试器将无法对其进行调试。另一种方法是在程序加载过程中将 monitor 也加载到 DSP 中去。由于两部分代码并不是链接到一起的，因此可能在内存空间上冲突。动态 monitor 由于尺寸很小，因此将其放在用户并不常用的内存位置上时与用户代码在空间上冲突的概率会比较小。有几种方法可以避免 monitor 在空间上与用户代码冲突：

(1)延迟定位，即在嵌入 monitor 前才决定 monitor 的位置。用户代码的加载由调试器完成，因此调试器能够知道用户使用了哪些内存空间，从而避免 monitor 放在用户使用的内存空间上。

(2)在链接时限制用户的内存使用，留出 monitor 的空间。DSP 代码在链接时都会有相应的内存描述文件，用户可以修改内存描述文件，空出 monitor 的内存空间。

3.4 含有 monitor 的 DSP 程序加载

加载程序的方式主要有两种：下载和引导(Boot)。对于基于总线的调试器来说，主机可以通过 DSP 外部总线访问 DSP 的片内存储器，因此适合采用下载方式加载程序，DSP 处于 idle 状态而所有的加载工作都交给主机来完成。引导方式是先将一段短小的引导内核传送给被加载方，然后 DSP 开始运行这段引导内核，接着 DSP 通过某种通信方式从加载方获取程序代码和数据，从而开始主程序的运行。这两种加载方式的主要区别是DSP开始执行代码的时机不同：采用下载方式时 DSP 直到所有的指令和数据都已经写入内存后才开始执行代码；采用引导方式时DSP在接收到引导内核后就立即开始执行。引导的加载方式比较灵活，适用于加载方无法直接访问被加载方存储器的情况。当然，基于总线的调试器也能够支持引导的加载程序方式。

加载完成后需要将 DSP 的控制权交给 monitor，即 DSP 是从 monitor 中的代码开始运行。在 monitor 将 DSP 的控制权交给真正的应用程序之前还需要做一些 DSP 的初始化工作，例如中断和控制寄存器的初始化。这些工作可以由主机完成，或者由主机交给 monitor 完成。

4 用于调试的 stdio 库实现

DSP 处理器不具有输入输出设备，但是 printf 等函数又是经常使用的一种调试工具，使 DSP 支持 stdio 库会使程序调试更加便捷。stdio 的实现原理是将 stdio 操作都归结到 IO 路由模块，将 IO 数据保存在缓冲区中，由主机端调试器代理进行实际的 IO 操作。实现框图如图 5 所示。

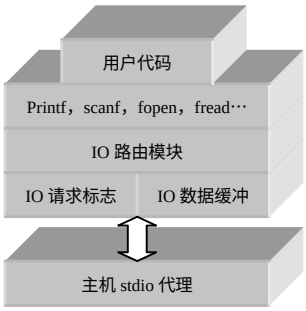

图 5 用于调试的 stdio 的实现结构

DSP 的 stdio 请求可以通过 monitor 来通知主机，也可以自定义与主机通信的接口。使用前一种方式时，IO 路由模块会在发生 IO 请求时通知 monitor，由 monitor 通知主机进行 IO 操作。这种方式虽然省去了通知主机的代码，但由于动态 monitor 在不同状态间切换时需要耗费一些时间，因此会对 stdio 的性能产生一些影响。采用独立的 stdio 与主机的握手方式可以提高 stdio 的响应速度。

5 主机端调试器的实现

具有动态 monitor 的 DSP 交叉调试器的主机端具有如图 6 所示的结构。

加载控制模块对调试目标文件进行分析，将代码和数据加载到 DSP 的内存中，并将符号和源码对应部分交给符号表和源码对应模块管理。加载完成后 DSP 控制模块调用 monitor 管理模块使 DSP 程序开始运行。Monitor 通信、管理模块负责管理 monitor 的不同状态，向 monitor 下载代码以及接收 monitor 的通知。Stdio 代理负责执行 DSP 发送来的各种 stdio 命令，并将结果返回 DSP。DSP 运行时分析模块负责对 DSP 的运行状态作统计和分析，并提交给调试人员。多处理器管理模块负责管理系统中的众多 DSP，并根据调试的需要，将调试焦点对准某一个 DSP。

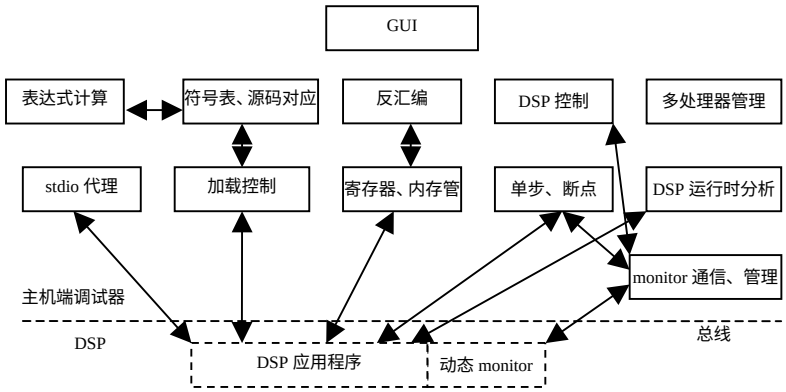

图 6 主机端调试器的结构

6 对比测试

笔者测试了 ADI 公司 TS-101 处理器在 Visual DSP++环境下使用 JTAG 调试与本文中基于 CPCI 总线的调试器调试时的加载程序的速度以及 printf 的速度。用来进行加载测试的程序是一个 747KB 的 ELF 文件，测试将其加载到 4 片 DSP 上所用的时间。stdio 的测试不停地调用 printf 函数，每次输出 10 个字符，统计每秒钟调用 printf 函数的次数。测试结果如表 1 所示。

表 1 软件调试器和 JTAG 性能对比

比较项目	JTAG	基于总线的软件调试器
加载时间/s	31.6	0.6
printf 次数	6.3	93.3

从测试数据可以看出，基于总线的软件调试器加载速度是 JTAG 的 52 倍多，每秒钟处理 printf 调用的次数是使用 JTAG 的 14.8 倍。软件调试器加载的速度和 printf 的处理速度与 DSP 硬件板卡型号和主机硬件都有关系，表 1 中的测试数据只是在测试主机上对特定 DSP 板卡得出的测试结果，但是这些数据能够说明软件调试器的性能相对 JTAG 仿真器在数量级上的优势。

在调试器的兼容性测试上无疑是 JTAG 占了上风。使用 JTAG 仿真器基本上可以调试任何的 DSP 程序。而软件调试器则无法调试软件异常级别的中断服务程序，即无法在这些中断服务程序中单步执行或者设置断点。在资源占用方面，笔者在 TS-101 上实现的软件调试器仅占用了 8 个 32-bit 字的内存空间，对于被调试的应用程序几乎不会有影响。在功能上，软件调试器和 JTAG 仿真器一样都支持源码级和汇编级的单步、断点，支持数据观察、修改、支持 stdio 库函数的调用，支持 DSP 程序的运行时分析等。

在测试和使用中，基于总线的软件调试器与硬件仿真器相比具有以下优点：(1)更快的程序加载速度和 stdio 函数的调用速度；(2)更方便的使用，不需要额外的调试硬件；(3)更好的多处理器支持，DSP 数量无限制；(4)更高的性价比。

(下转第 244 页)