第 36 卷 第 9 期 计 算 机 工 程 2010 年 5 月
Vol.36 No.9 Computer Engineering May 2010
·工程应用技术与实现· 文章编号：1000—3428(2010)09—0260—03 文献标识码：A 中图分类号：N945

Linux 下 RapidIO 子系统的分析与实现

丁 星，陈 洁，倪 明，毛 祺
(华东计算技术研究所，上海 200233)

摘 要：针对 Linux 驱动设计，提出 Linux 下 RapidIO 总线驱动的分层结构，其中包括全局层、总线层和设备层，并对每层进行阐述。实现 Linux 下 RapidIO 端设备驱动和基于 RapidIO 总线的全局共享存储，给出其关键实现流程和接口函数，并对全局共享存储实现进行分析，通过对实验数据进行分析得出 RapidIO 传输的高效性。

关键词：RapidIO 总线；设备驱动；全局共享存储

Analysis and Implementation of RapidIO Subsystem for Linux

DING Xing, CHEN Jie, NI Ming, MAO Qi
(East China Institute of Computer Technology, Shanghai 200233)

【Abstract】This paper presents the design of Linux drivers, a RapidIO bus driver under Linux's hierarchical structure, including the global layer, the bus layer and the device layer, and elaborates on each layer. RapidIO end device driver and global shared memory are carried out. It gives the realization of its key processes and interface functions, and analyzes global shared memory to achieve. Analysis of experimental data shows RapidIO transmission efficiency.

【Key words】RapidIO bus; device driver; global shared memory

1 概述

RapidIO(RIO)是一种分组交换结构，其目的是连接线路板上的芯片和连接机箱内的线路板[1]。

RIO 规范分为 3 层：逻辑层，传输层和物理层，见图 1。

(1)逻辑层：最高层。说明在 RIO 中应用程序是如何通信的。定义全部协议和包的格式，为端点器件发起和完成事务提供必要的信息。

(2)传输层：中间层。定义 RIO 的地址空间并为包在端点设备间传输提供必要的路由信息。

(3)物理层：最底层。描述设备间接口的细节，明确说明包传输机制、流控机制、电气特性和底层错误处理。

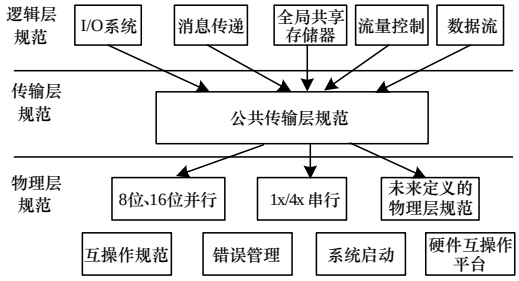


图 1 RIO 规范层次结构

这种层次划分便于在逻辑层增加新的事务类型而无需更改传输层和物理层规范，具有很好的扩展性。传输层和物理层通常在硬件中实现，Linux 下要做的就是硬件抽象、寄存器控制、配置空间读写的软件封装和逻辑层规范的实现。

2 Linux下RIO子系统驱动的设计

设备驱动程序是操作系统内核和机器硬件之间的接口，在 Linux 操作系统中扮演着一个特殊的角色，它们控制着设备的操作动作，并且向其他应用程序提供一个可用的程序接口与这个设备互动。设备驱动程序为应用程序屏蔽了硬件的细节，这样在应用程序看来，硬件设备只是一个设备文件，应用程序可以像操作普通文件一样对硬件设备进行操作。因此在操作系统中起着不可缺少的作用[2]。

为了防止在过渡期间产生的不必要的麻烦，如果一个成员的名字变了或是被去除了，那所有底层的驱动将会不可用；另一方面，如果只有总线层(并不是设备层)访问设备结构，那就只须修改需要修改的那一层即可。这种对系统中所有设备进行一种完全分层的组织的好处是可以相对容易地给用户空间提供一种完全分层次的设备关系图。RIO 作为一种总线，有别于其他的设备驱动，在 Linux 下其子系统设计分成 3 层：RIO 全局层处理全局公共数据称为 RIO 子系统核，RIO 总线层处理公共事务包含总线驱动、探测和枚举、交换设备支持，RIO 设备层提供设备的相关操作，包括互连服务、设备文件属性支持、具体的端设备驱动。具体框架见图 2。

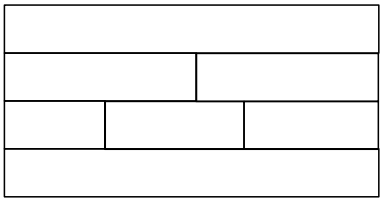


图 2 RIO 子系统框架

2.1 RIO全局层

RIO 子系统核包含 RIO 在内核中应用的全局数据结构，

作者简介：丁 星(1983－)，男，助理工程师，主研方向：嵌入式系统软件；陈 洁，高级工程师；倪明，研究员；毛 祺，助理工程师

收稿日期：2009-11-27 **E-mail：**dxing_1983@163.com

负责把硬件结构抽象到结构体中，由以下 4 个部分组成：

(1)rio_mport：RIO 主口信息，允许 RIO 事务在系统中发送和接收的接口，提供 Linux 到桥接器处理的桥梁。包含主口信息，邮箱、门铃等资源信息，端口号、名称等。

(2)rio_switch：RIO 交换信息，在端到端互连结构中到达目的的路由包。包含 RIO 交换设备的信息，交换链表中的节点、路由表、路由添加和获取的回调函数等。

(3)rio_dev：RIO 设备信息，系统中端设备和交换设备的相关信息。包含通用的配置信息、设备 ID、交换端口、资源信息、目的 ID 等。

(4)rio_net：RIO 网络信息，由互连结构中端和交换设备组成。包含系统网络的信息、设备列表、端口列表等。

2.2 RIO总线层

RIO 总线驱动提供一种公共统一的数据模型，负责初始化 RIO 总线，向 Linux 的设备模型中注册总线的设备 ID 和总线的类型，通过把一组数据和操作统一到全局可访问的数据结构中，为桥接器和设备增加具体总线的驱动，包括一组 RIO 的公共属性和一组公共的回调函数总线探测、匹配设备、总线关闭和指定设备属性文件的入口。

交换设备[3]支持通过制造商 ID、设备 ID、2 个更改和得到交换设备的路由入口回调函数，将交换设备注册到内核中供调用。为了初始化 RIO 系统，在 RIO 网络中必须初始化至少一个主口处理发送和接收事务，向系统初始化函数 device_initcall 传递 RIO 的初始化主口，由主机获取初始化引导代码，然后执行系统探测和枚举算法，枚举所有器件并将相关器件信息记录到器件结构链表，建立所有端点器件间的路由，最后映射地址空间。

探测和枚举系统中所有 RIO 连接器件并为其分配唯一的器件 ID。主机定义成一个大于或等于 0 主口 ID，通过维护事务访问设备的配置空间。维护事务用 2 个组件组成 1 个特定的设备：设备 ID(端设备用来决定是否需要接收包，在枚举阶段所有的设备忽略设备 ID 接收任何事务，交换设备不需要设备 ID)和跳数(到达交换设备的事务必须要有合适的跳数，只有当跳数大于 0，交换设备才会处理路由，否则当成自己的配置空间处理)。根据上电后器件的 ID 值设置优先级，ID 值较大的器件进入枚举，收集网络中有用的拓扑信息路由表和存储器映射信息等。

通过递归处理网络的深度，当一个设备被探测到，写主设备 ID 锁定寄存器确定枚举已经执行，按设备的能力决定设备是端或交换设备：如果是端设备，分配新的设备 ID 并写回终端，分配新的 rio_dev 并初始化；如果是交换设备，需要制造商 ID 和设备 ID 创建交换表，交换表包含交换路由操作，读写交换路由，分配新的 rio_dev 和 rio_switch。枚举交换机上每一个活动的端口，对每条活动的链接，到设备 ID 的路由被写入路由表。当主机完成枚举时，清除锁定寄存器，对系统中每个设备的组成标签寄存器写入“枚举完成”值，RIO 端设备设置为可见的地址范围与设备 ID 一起组成系统中唯一的地址。通过转换逻辑把这个唯一的地址写到系统中其他器件的本地地址空间。

2.3 RIO设备层

互连服务提供设备互连操作：通过本地设备执行主口检测获取基本/扩展设备 ID；请求 inbound 邮箱的所有权并给该资源约束一个回调函数；释放 inbound 邮箱的所有权；请求 outbound 邮箱的所有权并给该资源约束一个回调函数；释放 outbound 邮箱的所有权；向端口门铃事件列表中添加门铃的资源/回调对；请求 inbound 门铃的所有权并给该资源约束一个回调函数；释放 inbound 门铃的所有权；请求一个门铃消息范围，返回该资源；释放门铃消息范围；测试设备是否支持给定的 RIO 性能；提供端设备发现和获取功能。

设备文件属性支持保证无论何时在这个树上插入设备，内核都会为它创建一个目录，可以在目录下通过创建文件来导出该设备的数据或提供调整的接口。设备文件属性支持包括文件建立、文件清除、路由显示以及读/写当前的 RIO 信息。提供到/sys 和/proc 目录下的相应设备文件的显示。

3 RIO端设备驱动

RIO 端设备驱动程序主要由以下几个部分构成：初始化设备模块，操作和控制模块，卸载设备模块等。

3.1 初始化设备模块

初始化设备模块主要完成以下工作：注册驱动程序，初始化 RIO 端设备，为系统分配内存。初始化端设备的 RIO 硬件接口，根据系统指定的信息配置主口，向 RIO 子系统注册主口。初始化端设备流程见图 3。

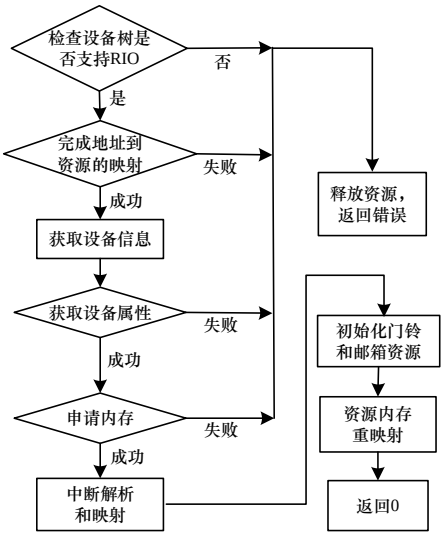


图 3 初始化端设备流程

3.2 操作控制模块

操作控制模块定义一组抽象访问硬件的接口，管理邮箱访问、门铃、消息的读写。

(1)打开 inbound 邮箱：根据应用要求初始化环形缓冲，初始化 inbound 消息环，指向环队列的第一个指针，清除中断，连接 inbound 消息句柄，配置 inbound 消息单元，设置队列入口数量，激活 inbound 消息单元。

(2)打开 outbound 邮箱：根据应用要求初始化环形缓冲，初始化 outbound 消息描述环，指向环队列的第一个指针，配置监听，清除中断，连接 outbound 消息句柄，配置 outbound 消息单元，设置队列入口数量，激活 outbound 消息单元。

(3)发送门铃消息：向寄存器写门铃数据。

(4)门铃中断处理：读寄存器，处理门铃中断。

(5)读 inbound 消息队列：根据消息寄存器读消息到物理缓冲，从物理缓冲读出消息到 inbound 缓冲，复制消息到应用缓冲，清除 inbound 缓冲。

(6)写 outbound 消息队列：获得目的端口，从应用缓冲复制消息到 outbound 缓冲，向寄存器设定消息域、目的 ID、中断优先级、源缓冲地址，队列指针增加。

3.3 卸载设备模块

卸载设备模块的主要工作有：释放对设备的控制权，释放 IO 内存资源，释放内存映射，注销设备驱动。

4 RIO全局共享存储的实现

RIO 全局共享存储(GSM)规范支持下列性能，设计这些性能是为了满足多种应用和系统的需要[4-6]：

(1)支持一致性高速缓存非均匀存储器访问(CC-NUMA)系统体系结构，用来在以点对点方式进行器件间互连环境中提供全局共享存储器模型。

(2)支持处理器的存储器的请求长度为缓存一致性粒度或者更小。

(3)GSM 协议支持多种缓存控制和其他的操作，如块刷新等。

在全局共享分布式存储器编程模型下，存储器可能在物理上分布于不同的角落，而却可被不同的处理器一致访问。一致性意味着当处理器从存储器中读取数据时，它得到的总是它所请求数据的最新版本。

基于 GSM 的严格要求，软件设计应实现如下功能：

(1)存储映射：映射 I/O 地址和系统内存空间到 RIO 的地址空间。

(2)内存分配：RIO 空间的 inbound 窗口和 outbound 窗口建立，实现 I/O 空间分配。

(3)存储域管理：系统能通过 outbound 窗口访问其他的 RIO 设备，也可被其他的 RIO 设备通过 inbound 窗口访问。

从完善功能的角度上，需提供如下接口：RIO 的 I/O 域请求资源，为 RIO 主口分配 I/O 资源，建立系统内存空间到 RIO 的 inbound 内存域的映射，建立系统内存空间到 RIO 的 outbound 内存域的映射，释放 inbound 内存域，释放 outbound 内存域，释放 inbound 域资源，请求 inbound 内存域，释放 outbound 域资源，为 outbound 映射准备 I/O 域，为 outbound 请求 I/O 域并获得 outbound 域。对使用者来讲，最重要的就是 inbound 和 outbound 内存域的访问。

请求 inbound 内存域流程见图 4，请求 outbound 内存域见图 5。

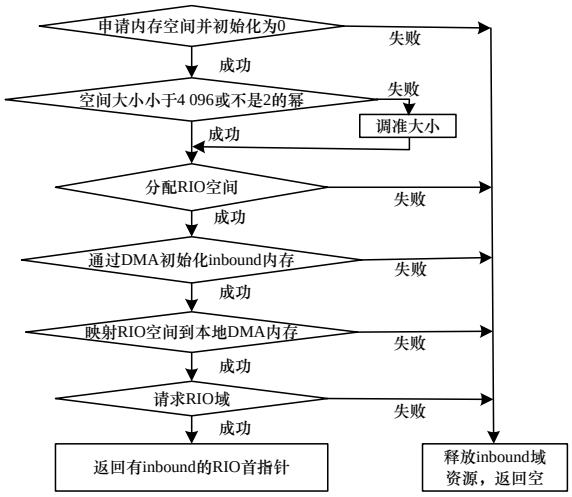


图 4 请求 inbound 内存域流程

有些程序可能对消息传递函数提供的可靠性没有要求。当数据量非常大的时候，使用可靠性协议会引起时延，并消耗大量 CPU 时间。当数据和数据流一直向输入端输入时，数据队列复制和其他机制都会消耗处理器周期。在这种情况下，引入低的传输延迟会非常有效。

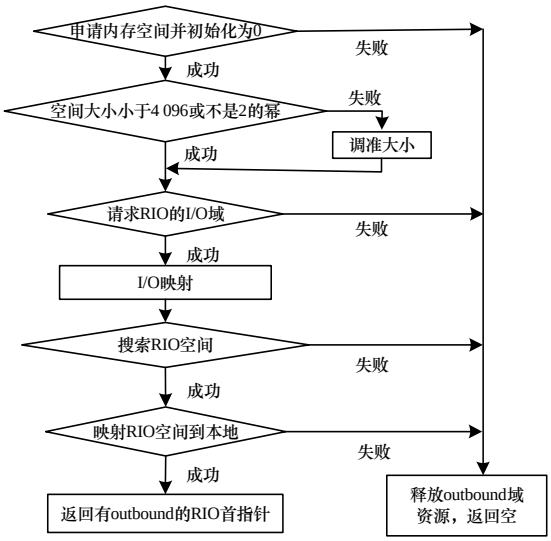
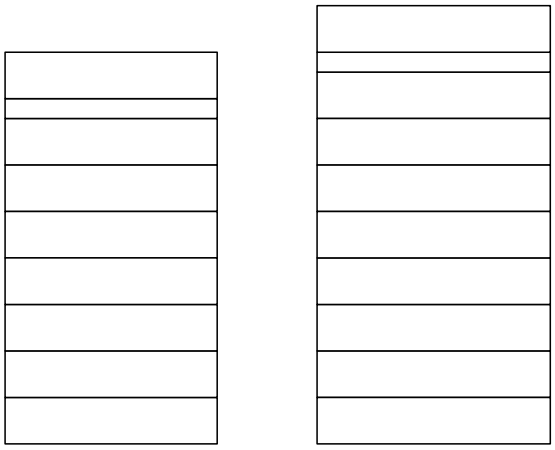


图 5 请求 outbound 内存域流程

GSM 正是为低时延地传送大量数据而设计的。GSM 提供了与消息传递函数不同的一系列服务。消息传递提供可靠的、面向连接的服务；而 GSM 通过提供接收缓冲实现快速发送到慢接收，而不是使用数据队列和限制发送方的发送量。这一类的服务可能比消息传递不可靠，但实际上，所有的传输请求都已在目的地缓冲器中执行。

RIO 端设备驱动和 GSM 实现的基本接口函数见图 6，其中 RIO_driver 为端设备驱动的接口函数，RIO_GSM 为 GSM 的基本接口函数。



(a)端设备驱动的接口函数　　(b)GSM 的基本接口函数

图 6 基本接口函数

采用 2 套 MPC8641 板卡各提供一路 1x 的 RIO 进行点对点对联，测试上述接口函数的端到端时间见表 1。由实验结果可以看出，即使在 Linux 环境下 RIO 传输也能提供比较高的带宽。

表 1 实验数据

传输类型	传输负载/Byte	端到端时间/μs
Mbox	2K	28.4
Dbell	2	7.8
Msg	4K	173.2

（下转第 265 页）