基于 VxWorks 系统的 PCI 配置与应用

李 岳

（中国电子科技集团公司第二十八研究所 南京 210007）

摘 要 VxWorks PCI 总线的配置是系统应用的重点和难点。本文从应用角度解析 PCI 的配置原理和方法，列举常用的 PCI 库编程接口，结合应用示例与大家交流学习。

关键词 VxWorks 操作系统 PCI 总线 PCI 设备 MMU 配置 PCI 网卡配置 板级支持包

引言

VxWorks 操作系统专门对 PCI 设备提供支持，包括 PCI 总线驱动和一套 PCI 支持函数。配置总线上的设备还与硬件的体系结构和 BSP 支持程度相关，本文结合 PCI 原理，谈谈实际使用中的经验与体会。

1 PCI 总线架构与功能

1.1 典型架构

结构图（如图 1 所示）是典型的 PCI 总线系统架构[1]。此图中处理器、高速缓存和内存子系统通过 PCI 桥连接到 PCI 总线。桥提供一种高速通道使得处理器可以直接访问 PCI 设备。它还提供高带宽的通道允许 PCI 直接访问主存。桥也包含了可选择的功能比如：总线仲裁和热插拔。

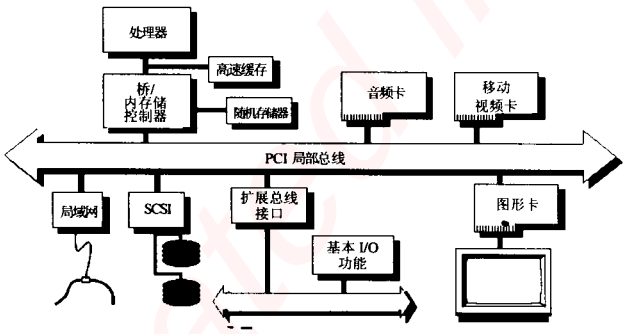

图 1 PCI 系统结构图

1.2 总线号、设备号、功能号

PCI 总线可以通过桥芯片级连，按与 CPU 总线所隔的桥数目和同一层总线的扫描顺序，从 0 开始依次编号，扩展最多到 256，0~256 就是总线号；在指定的局部总线上，按硬件扫描顺序从 0 开始依次编号，扩展最多到 32，0~32 就是设备号；在一个多功能 PCI 设备上，最多可以实现 8 种功能，按设备上配置存储区的顺序从 0 开始依次编号，称其为功能号，一般的设备只有一个功能。这三个号组合起来就可确定唯一的一个 PCI 设备，以及该设备上的某项功能。通过这三个参数就可操作 PCI 设备。

2 PCI 配置空间

根据 PCI 规范，每个 PCI 设备都要有一个 PCI 配置空间[2]，容量最大为 256 字节，称为配置寄存器。配置寄存器是 PCI 设备的硬件与其初始化软件信息交接的地方，软件可以通过它实现对设备的识别和监控。

2.1 配置空间组织结构

256 字节的空间分为头标区和设备关联区两部分，如图 2 所示。

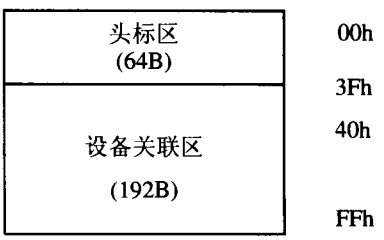

图 2 PCI 总线配置空间的组织

PCI 设备的头标区中寄存器布局和格式大致相同，其布局如图 3 所示。头标识区又分为两部分，头 16 个字节定义都相同，余下的 48 个字节根据设

备的功能类别布局不同。位于 0Eh 的头标识类型字段规定了后 48 个字节所用布局。目前的规范只规定了 00h 和 01h 两个头标识类型：01h 用于 PCI-to-PCI 桥；00h 用于其余 PCI 设备。

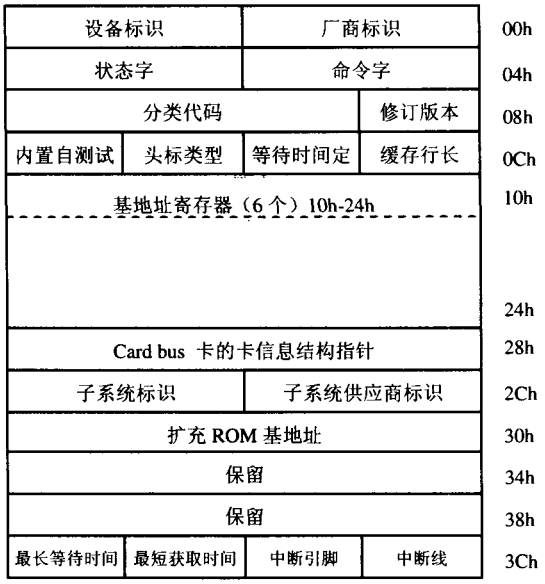

图 3 PCI 总线配置空间头标区寄存器的布局

2.2 配置空间的功能

类型 0 配置空间头标区里的寄存器实现了很多功能，这里介绍常用的几类。

2.2.1 设备识别

在头标区内有五个寄存器是与一个 PCI 设备的识别有关的，配置软件通过访问这些寄存器很容易确定在该 PCI 总线上有什么 PCI 设备。这五个寄存器都是只读的。

·厂商标识：标识设备的制造厂商，由 PCI 规范的权威组织 SIG 统一分配，以保证唯一性。如 Intel 公司的标识码为“8086h”，“FFFFh”是无效标识。

·设备标识：标识某一设备，由制造厂商分配。

·修订版本标识：标识设备具体的修订版本，由厂商选择。

·头标类型：表明头标区中 10h 到 3Fh 字节空间的布局类型和该设备是否是多功能设备。位 7 用来标识一个多功能设备，为 1 则该设备是多功能设备；为 0 则是单一功能设备。位 6 到 0 规定头标区中 10h 到 3Fh 字节的布局类型，如上所述，目前只有两类：00h 和 01h。

·分类代码：该寄存器有三个字节段：高字节（0Bh）是基本分类码，粗略地对功能分类；中字节（0Ah）是子分类码，标识具体功能；低字节（09h）标识所用寄存器一级的编程接口。

2.2.2 设备控制

位于配置空间 04h 偏移地址处的命令寄存器是控制设备产生和响应 PCI 周期的。是可读写寄存器，其各位定义和作用如下（这里仅描述软件常用的位）：

·I/O 空间使能位：控制 I/O 空间访问的响应，为 1 时，允许设备响应 I/O 空间的访问；为 0 时禁止。

·存储空间使能位：控制本设备对存储器空间访问的响应，为 1 时允许，为 0 时禁止。

2.2.3 基地址

头标区中有 6 个双字的基地址寄存器，在加电时，6 个寄存器向 POST 软件反映该设备需要多少存储器空间和 I/O 空间。POST 软件就了解到 PCI 总线上有哪些存储设备和 I/O 设备，根据它们的容量需求把它们映射定位到适当的存储器地址和 I/O 地址，并把起始地址写入基地址寄存器，再启动系统。

·基地址寄存器

位 0 只读，用来表示此寄存器申请的是存储器空间还是 I/O 空间。

如果位 0 为 1 则该基址寄存器用于 I/O 空间映射。位 1 是保留位，位 2~31 用来把从寄存器映射到系统 I/O 空间，POST 再把系统分配的系统 I/O 空间基地址写回寄存器的 2~31 位。

表 1 基地址寄存器位 2、位 1 的含义

位	含 义
00	基地址寄存器为 32 位宽，可以映射到 32 位存储空间的任何地方
01	为 32 位宽，必须映射到 1M 以下的空间
10	为 64 位宽，可以映射到 64 位空间的任何地方
11	保留

如果位 0 为 0 则要映射到存储器空间。位 1~3 是只读的，其中 1~2 是类型编码，表示映射要求，在表 1 中列出了类型编码。

64 位寄存器占据两个字，其后面的基地址寄存器的偏移地址要向后推一个双字，但总容量不变，保持 6 个双字大小。

· 地址映射

映射过程分为两个步骤：

(1) 确定空间类别和容量要求：读取各设备基址寄存器，获取需要的空间和容量。类别可通过最低位 0 位判断。容量判断是把寄存器中的基地址字段的若干低位从硬件上连接到地，从 PCI 总线向寄存器全写-1，再读回其值。由于接了地，读回的值中，从 bit4（存储器类型）和 bit2（I/O 类型）到最高位返回的 0 的个数反映了所需空间。

(2) 分配基地址：POST 软件分配起始地址，并向各寄存器高位字段写入分配空间的基地址，把寄存器对应的存储器和 I/O 映射到系统物理地址空间去。

2.3　VxWorks 对 PCI 设备的支持

VxWorks 操作系统对 PCI 设备提供了专门的支持库，用户可以利用该库实现对 PCI 设备的控制。通过库里的配置函数，可以动态寻找指定的 PCI 设备，并可以读取和修改该设备的配置空间内容。在配置空间的映射过程中这些函数非常有用，下面列举几个常用的函数，并简单描述其功能以供参考，如表 2 所示。

表 2　PCI 总线配置函数

函　数	描　述
PciConfigLibInit	初始化 PCI 配置支持库
PciFindDevice	寻找以厂商标记、设备标记指定的设备
PciFindClass	寻找以类别指定的 PCI 设备
PciDevConfig	配置指定的设备
PciConfigBdfPack	返回组合为整型数的总线号、设备号、功能号

表 3　PCI 总线中断复用函数

函　数	描　述
PciIntLibInit	初始化 PCI 中断复用支持库
PciInt	PCI 中断服务程序，用于管理、调用 PCI 设备中断
PciIntConnect	挂接中断服务程序
PciIntDisconnect	清除中断服务程序的挂接

中断与中断服务几乎是所有设备运转的核心。PCI 设备也离不开中断与中断服务的支持，VxWorks 系统对 PCI 设备提供了强大的中断复用支持库，来支撑 PCI 设备挂接和断开中断服务程序。常用的几个函数如表 3 所示。

PCI 支持库中的函数较多且功能强大，详细内容可以参照开发工具 Tornado 所提供的帮助手册。

2.4　VxWorks 中 PCI 设备的地址映射

VxWorks 操作系统在加电过程中会对设备执行地址映射操作，以便程序可以正确访问设备。在使用全面的 MMU 管理组件时，这个过程是系统自动完成的，但如果系统只包含了基本的 MMU 功能，用户就必须手动修改 BSP 来配置。

先简单了解一下内存映射表。在 sysLib.c 中有一张表 sysPhysMemDesc 描述了系统物理内存的分配。如下所示：

```
PHYS_MEM_DESC sysPhysMemDesc [ ] =
{
/*地址和长度参数必须按页对齐（4KB/4MB 的倍数）*/
{/*低端内存*/
(void *)   LOCAL_MEM_LOCAL_ADRS,
(void *)   LOCAL_MEM_LOCAL_ADRS,
0xa0000,
VM_STATE_MASK_FOR_ALL,
VM_STATE_FOR_MEM_OS,
},
{/*显示内存等*/
(void *)   0xa0000,
(void *)   0xa0000,
0x60000,
VM_STATE_MASK_FOR_ALL,
VM_STATE_FOR_MEM_IO,
},
{/*为操作系统保留的高端内存*/
(void *)   0x100000,
(void *)   0x100000,
0x080000,
VM_STATE_MASK_FOR_ALL,
VM_STATE_FOR_MEM_OS,
},
{/*为应用程序保留的高端内存*/
(void *)   0x1800000,
(void *)   0x180000,
```

```
    LOCAL_MEM_SIZE - 0x180000, /* 在
sys_Mem_Top()中会改变 */
    VM_STATE_MASK_FOR_ALL,
    VM_STATE_FOR_MEM_OS,
    },
    /*用于动态映射的条目*/
    DUMMY_MMU_ENTRY,
    DUMMY_MMU_ENTRY,
    };
```

在上表中描述了系统物理内存在逻辑内存中所分配的空间。系统引导时会根据这里的描述管理内存空间。

手动配置 PCI 设备的方法就是根据 PCI 基地址寄存器的值在源码中修改上表。但是通常不推荐这样做，因为这样将不能适应硬件环境的变化局限了 BSP 的使用。

通常使用的方法是半自动配置方法。所谓半自动方法是相对于包含完整 MMU 的 VxWorks 所采用的自动化方法。自动化配置方法将自动发现 PCI 设备并自动为其分配所需空间。但是仅包含了基本 MMU 的 VxWorks 并不会自动完成上表配置，需要其他辅助步骤帮助系统映射内存，包括为所有设备分配不冲突的内存，以及编程使系统找到指定的 PCI 设备并分配空间。

3 PCI 设备配置应用实例

3.1 自动配置方法

首先根据设备类型（如显示设备、网络设备）查询总线，以此获取此类设备的 PCI 总线号、设备号、功能号、厂商号和设备标识，此处假设系统中没有相同的一类设备。然后，根据总线号、设备号、功能号获取该设备的 PCI 配置空间信息，并写入 MMU 用于系统物理内存分配。

3.2 显示设备的配置

要完成显示设备的配置，首先要包括 VxWorks 系统组件“basic MMU”基本内存控制单元。然后再将 PCI 显示设备的内存空间配置到 MMU 里面，否则该内存空间的访问会出现异常。如果该内存空间是预先知道的，可以在 syslib.c 的 sysPhysMemDesc 表中静态配置。如果该内存空间的地址和大小无法预先知道，则需要通过动态查询得到，然后再调用 MMU 配置函数加入 MMU 表。其简要过程如下：

(1) 在 PCI 总线上查找显示设备，并获取其总线号、设备号和功能号；

(2) 根据这三个号来读取该设备的内存基地址和需要的内存大小（注意：内存大小是从开始位到最高位所返回的零的个数，需要用户自己计算。内存地址要去掉低位非地址含义的位。）；

(3) 在 sysLib.c 的 sysHwInit()函数中调用配置函数 sysMmuMapAdd，该函数将基地址和内存大小配置入 MMU 表，整个配置工作就完成了。

3.3 PCI 网卡设备的探测与驱动加载

网络设备的初始化与显示设备的配置在原理上是一致的，其过程如下：

(1) 通过 PCI 设备类型查询函数找到网络设备的 PCI 总线号、设备号和功能号。再从配置空间读取设备厂商号、标识号和修订版本标识。

(2) 根据设备厂商号、标识号判断网卡的型号，再根据其修订版本标识来判断网卡的小型号，这样一个网卡的具体型号就可以确定。

(3) 在确定了网卡的具体型号后，加载相应的驱动程序即可。

当然，这样做的前提是已经在 BSP 中配好了相应的网卡驱动，否则无法保证探测到的网卡能被成功初始化。同理，可以利用这种方法在 BSP 层屏蔽其他 PCI 设备的差异，实现相对而言的通用，这在许多工程中有其实际应用价值。

4 小结

VxWorks 操作系统和 PCI 总线的内容比较丰富，还涉及硬件设备的工作原理以及整个系统的体系结构。本文基于 X86 体系结构，从基本原理出发，结合应用实例说明了 PCI 的配置方法。

参考文献：

[1] Tom Shanley. PCI X 系统的体系结构. 北京：清华大学出版社, 2002.

[2] 李方敏. VxWorks BSP 开发人员指南. 北京：中国水利出版社, 2004.

作者简介：

李 岳，男（1980-），助理工程师，现从事计算机软件系统集成工作。

嵌入式资源免费下载

总线协议：

1. 基于 PCIe 驱动程序的数据传输卡 DMA 传输
2. 基于 PCIe 总线协议的设备驱动开发
3. CANopen 协议介绍
4. 基于 PXI 总线 RS422 数据通信卡 WDM 驱动程序设计
5. FPGA 实现 PCIe 总线 DMA 设计
6. PCI Express 协议实现与验证
7. VPX 总线技术及其实现

VxWorks：

1. 基于 VxWorks 的多任务程序设计
2. 基于 VxWorks 的数据采集存储装置设计
3. Flash 文件系统分析及其在 VxWorks 中的实现
4. VxWorks 多任务编程中的异常研究
5. VxWorks 应用技巧两例
6. 一种基于 VxWorks 的飞行仿真实时管理系统
7. 在 VxWorks 系统中使用 TrueType 字库
8. 基于 FreeType 的 VxWorks 中文显示方案
9. 基于 Tilcon 的 VxWorks 简单动画开发
10. 基于 Tilcon 的某武器显控系统界面设计
11. 基于 Tilcon 的综合导航信息处理装置界面设计
12. VxWorks 的内存配置和管理

Linux：

1. Linux 程序设计第三版及源代码
2. NAND FLASH 文件系统的设计与实现
3. 多通道串行通信设备的 Linux 驱动程序实现
4. Zsh 开发指南-数组

WeChat ID: kontronn

5. 常用 GDB 命令中文速览
6. 嵌入式 C 进阶之道

Windows CE:

1. Windows CE.NET 下 YAFFS 文件系统 NAND Flash 驱动程序设计
2. Windows CE 的 CAN 总线驱动程序设计
3. 基于 Windows CE.NET 的 ADC 驱动程序实现与应用的研究
4. 基于 Windows CE.NET 平台的串行通信实现
5. 基于 Windows CE.NET 下的 GPRS 模块的研究与开发
6. win2k 下 NTFS 分区用 ntldr 加载进 dos 源代码
7. Windows 下的 USB 设备驱动程序开发
8. WinCE 的大容量程控数据传输解决方案设计
9. WinCE6.0 安装开发详解
10. DOS 下仿 Windows 的自带计算器程序 C 源码
11. G726 局域网语音通话程序和源代码
12. WinCE 主板加载第三方驱动程序的方法
13. WinCE 下的注册表编辑程序和源代码
14. WinCE 串口通信源代码
15. WINCE 的 SD 卡程序[可实现读写的源码]

PowerPC:

1. Freescale MPC8536 开发板原理图
2. 基于 MPC8548E 的固件设计
3. 基于 MPC8548E 的嵌入式数据处理系统设计
4. 基于 PowerPC 嵌入式网络通信平台的实现
5. PowerPC 在车辆显控系统中的应用

ARM:

1. 基于 DiskOnChip 2000 的驱动程序设计及应用

2. 基于 ARM 体系的 PC-104 总线设计
3. 基于 ARM 的嵌入式系统中断处理机制研究
4. 设计 ARM 的中断处理
5. 基于 ARM 的数据采集系统并行总线的驱动设计
6. S3C2410 下的 TFT LCD 驱动源码

Hardware：

1. DSP 电源的典型设计
2. 高频脉冲电源设计
3. 电源的综合保护设计
4. 任意波形电源的设计
5. 高速 PCB 信号完整性分析及应用
6.