

文章编号:1001-9081(2005)07-1698-03

基于嵌入式 Linux 系统的 DVB-CI 设备驱动程序开发

周庆喜, 刘 强

(清华大学 软件学院, 北京 100084)

(zhouqingxi@power-digital.cn)

摘 要:介绍了在嵌入式 Linux 系统中,使用设备驱动程序隐藏具体 DVB-CI 设备的控制细节、向上层程序提供抽象 DVB-CI 设备的方法,并以 MontaVista 的 Hard Hat Linux 系统、IBM 的 STB02500 机顶盒硬件平台为例,说明了 DVB-CI 设备驱动程序的实现方法。

关键词:嵌入式 Linux; 数字电视; PCMCIA; DVB-CI; 条件接收

中图分类号: TP311.52 **文献标识码:** A

DVB-CI device driver development on embedded Linux system

ZHOU Qing-xi, LIU Qiang

(School of Software, Tsinghua University, Beijing 100084, China)

Abstract: The method was introduced to encapsulate the detailed controls of the concrete DVB-CI device with device driver and provide the up-layer program an abstract device on the embedded Linux system. A DVB-CI device driver based on MontaVista's Hard Hat Linux system and IBM STB02500 Set-Top Box Integrated Controller was implemented as an example.

Key words: embedded Linux; digital TV; PCMCIA; DVB-CI; CA(Conditional Access)

0 引言

嵌入式产品中的软件设计需要考虑硬件平台的体系结构与特性,具有同一种功能的产品一般都可以找到多种硬件实现方案。因此,如何提高软件在不同硬件平台的可移植性是开发人员必须要考虑的一个问题。解决这个问题的一般方法是软件分层设计,底层软件屏蔽不同硬件设备的具体特性,对上层软件提供统一的函数接口。

目前,嵌入式 Linux 系统被广泛应用。在嵌入式 Linux 系统中,由文件系统来屏蔽硬件设备的特性,具体设备被抽象成设备文件,用户程序可以通过标准的文件操作来实现对设备驱动程序的调用。这种架构很有利于嵌入式设备的驱动程序开发。

在数字电视领域,基于 DVB-CI 标准的机卡分离方案是一个重要的发展趋势。机卡分离,简单的说,是由数字电视一体机主机(Host)和控制用户收看节目的条件接收卡(Conditional Access Module, CAM)组成,两者之间通过 PCMCIA 接口进行通信。

DVB-CI(Digital Video Broadcasting, Common Interface)设备驱动程序开发主要受到硬件平台的影响。虽然主机与卡的接口规范是固定的,但不同的芯片厂商可能采用不同的 CPU 内核以及不同的内部总线,而且可能采用不同的 PCMCIA 控制芯片,因此对卡进行操作的实现方式是不同的。

本文介绍了在 IBM 的 STB02500 机顶盒硬件平台基础上,基于 MontaVista 嵌入式 Linux 系统的 DVB-CI 设备驱动程序的实现方法。文章首先介绍与 DVB-CI 有关的硬件设备,然后介绍软件的层次结构,最后根据硬件平台的特性,给出了本次 DVB-CI 设备驱动程序的设计方法。

1 系统相关硬件介绍

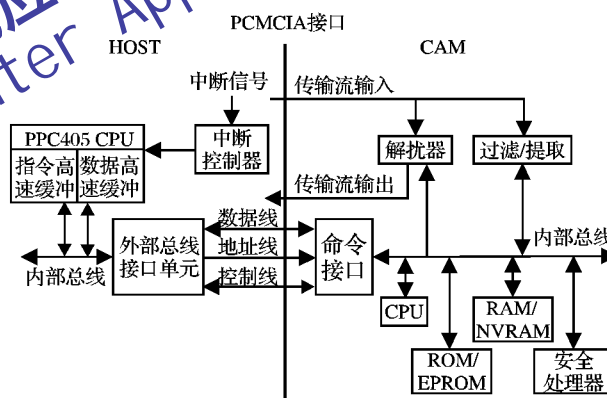


图1 主机与 CAM 硬件连接示意图

IBM 的 STB02500 机顶盒硬件平台采用的是 PowerPC 内核,通过外部总线接口单元(External Bus Interface Unit, EBIU)对 PCMCIA 插槽进行控制。图1 简要显示了主机和 CAM 卡之间的硬件连接情况。主机接收数字电视节目传输流,通过 PCMCIA 接口传送给 CAM 卡,传输流经过 CAM 卡的解扰器,解扰器对加扰的码流进行解扰,然后再传送回主机。CAM 卡的命令接口由四个寄存器构成,分别是数据寄存器、命令/状态寄存器和数据长度寄存器。主机对卡的控制与数据交换主要通过这几个寄存器完成。数据寄存器是 8 位宽度,偏移地址为 0;命令/状态寄存器是 8 位宽度,偏移地址为 1,主机写操作时写入的是命令,读操作时读出的是卡的状态;数据长度寄存器是 16 位宽度,低 8 位的偏移地址是 2,高 8 位的偏移地址是 3。对这几个寄存器的访问必须以 I/O 操作的方式进行。

另外, CAM 卡还包括属性存储器(未在图中画出),保存

收稿日期:2004-12-14;修订日期:2005-03-02

作者简介:周庆喜(1976-),男,天津人,硕士研究生,主要研究方向:软件工程、项目管理; 刘强(1963-),女,山东济南人,副教授,主要研究方向:软件工程、项目管理。

卡的描述和配置信息。通过这些信息,可以判断插入 PCMCIA 插槽的卡是否是 CAM 卡,并根据配置信息对卡进行配置。对属性存储器的操作也通过 EBIU 来完成,但必须以内存操作方式进行。

EBIU 接口支持这两种时序不同的操作方式。主机内部地址总线是 32 位宽度(第 0 位为地址最高位),其中第 6 位控制 EBIU 接口的外部时序是 I/O 方式还是内存方式。地址线第 6 位为 0,EBIU 外部时序为内存方式;第 6 位为 1,EBIU 外部时序为 I/O 方式。驱动程序必须根据操作的对象是命令接口还是属性存储器来切换 EBIU 的外部时序。

2 软件层次结构

图 2 显示了主机中与 CAM 卡相关的程序的层次结构。其中,资源层程序为 CAM 卡中的应用程序提供信息显示、用户输入等服务,使运行于卡中的程序可以完成与用户的交互。资源程序与 CAM 卡中的应用程序间的通信通过分层实现的 DVB-CI 协议栈来完成。目前,很多硬件厂商的芯片都支持基于 PCMCIA 接口的机卡分离,因为不同芯片的硬件设计方法是不同的,为了使 DVB-CI 协议栈可以方便地移植到不同的硬件平台,协议中最低的物理层要求底层程序提供对卡状态检测、属性存储器和命令接口寄存器的操作函数。DVB-CI 设备驱动程序正是为协议栈的物理层提供这些服务调用的。

嵌入式 Linux 的文件系统为设备驱动程序的开发提供了统一的形式:为硬件设备分配主设备号和次设备号后,使用 mknod 命令建立设备文件;设备驱动程序编译为可以动态插入内核的模块,使用 insmod 命令把驱动程序插入内核;文件系统通过设备文件的主设备号查找设备驱动程序,使用次设备号区别同类设备。这样,在用户空间就可以使用标准的文件操作来判断 CAM 卡的状态,读写属性存储器,读写命令接口寄存器。

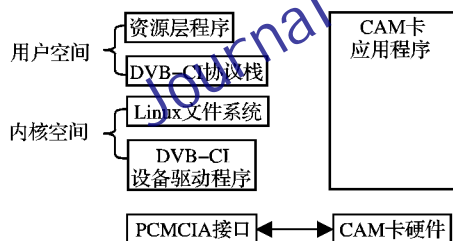


图2 主机相关软件结构图

3 设备驱动程序的实现

3.1 命令定义

在 Linux 系统中,用户空间的程序一般通过 open, close, ioctl, read, write 等调用,通过文件系统使用设备驱动程序来完成对设备的控制。因此,首先定义一些用户程序使用的命令。下面列出了部分重要的命令定义:

```

/* ioctl 命令定义 */
...
#define CI_DEV_MAGIC 0xF2
//读属性存储器
#define CI_READ_MEM_IOCTL
    (CI_DEV_MAGIC, 10, unsigned long *)
//写属性存储器
#define CI_WRITE_MEM_IOCTL
    (CI_DEV_MAGIC, 11, unsigned long *)
//读命令接口的数据寄存器

```

```

#define CI_READ_IO_DATA_IOCTL
    (CI_DEV_MAGIC, 12, unsigned char *)
//写命令接口的数据寄存器
#define CI_WRITE_IO_DATA_IOCTL
    (CI_DEV_MAGIC, 13, unsigned char *)
...

```

3.2 EBIU 接口配置函数

用户程序可以使用以上的命令来完成对 CAM 卡的控制。对于设备驱动程序来说,要完成上述命令,必然涉及到对设备的访问。这就必须解决设备在主机中的地址分配、操作时序等与硬件关系非常密切的问题;另外,还必须考虑虚拟地址与物理地址的映射。

在 STB02500 主芯片中,设备的物理地址是由外部总线接口单元(EBIU)确定的。经过配置,EBIU 可以响应主机内部地址总线对特定范围内物理地址的访问。主芯片 CPU 需要访问外部设备的时候,只需要驱动内部总线。如果内部总线访问的物理地址在 EBIU 接口响应的物理地址范围之内,则 EBIU 接口以规定的操作时序驱动外部的数据线、地址线和控制线,完成对设备的访问。图 3 是 EBIU 配置寄存器,其中地址基址和地址范围确定了 EBIU 接口响应的地址范围。在前面提到过,内部总线地址第 6 位控制 EBIU 对外部总线的操作是内存时序还是 I/O 时序,这就在硬件的层次决定了以内存时序和 I/O 时序操作外部设备的同一个地址的时候,内部总线的物理地址相差 0x02000000。但 EBIU 接口的地址范围最大是 16M,所以,在内存时序和 I/O 时序切换的时候,必须重新配置 EBIU 接口。

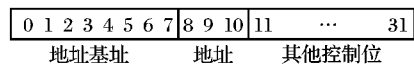


图3 EBIU 配置寄存器控制位

为了便于配置 EBIU,在设备驱动程序中增加了如下宏定义:

```

#define CI_PORT_IO_BASE 0xFE000000
// I/O 时序下的设备物理基址
#define CI_PORT_IO_OFFSET_MAX 0x00000004
#define CI_PORT_MEM_BASE 0xFD000000
// 内存时序下的设备物理地址
#define CI_PORT_MEM_OFFSET_MAX 0x00008000
#define BRCCR1_IO_MODE_VAL 0xE0185CEE //EBIU 控制字
#define BRCCR1_MEM_MODE_VAL 0xD0185CEE //EBIU 控制字
如果向 EBIU 配置寄存器写入 0xE0185CEE,则 EBIU 接口以 I/O 时序驱动外部总线,能够响应的内部总线物理地址范围为 0xFE000000 至 0xFE100000;如果向 EBIU 配置寄存器写入 0xD0185CEE,则 EBIU 接口以内存时序驱动外部总线,能够响应的内部总线物理地址范围为 0xFD000000 至 0xFD100000。通过 ioremap 把物理地址映射到虚拟地址之后,就可以在设备驱动程序中访问设备了。在设备驱动程序中 ci_base 被用来记录映射之后的虚拟地址,调用 ci_port_alloc 函数来配置 EBIU 并进行地址映射。下面是 EBIU 配置程序的 C 语言实现:
#define CI_MODE_MEM 0 //内存时序模式
#define CI_MODE_IO 1 //I/O 时序模式
#define CI_MODE_UNKNOWN -1 //未知模式
static int ci_mode = CI_MODE_UNKNOWN; //记录 EBIU 模式
static unsigned long ci_base = 0;
static int ci_port_alloc(unsigned char sw)

```

```

{
    unsigned long reg = 0;
    if ((sw != CI_MODE_MEM) && (sw != CI_MODE_IO)) {
        printk("ci_port_alloc input para wrong\n\r");
        return -1;
    }
    if (ci_mode == sw)
        return 0; //不必更改 EBIU 配置模式
    if (sw == CI_MODE_MEM) {
        reg = BRCR1_MEM_MODE_VAL; //内存模式
    }
    else {
        reg = BRCR1_IO_MODE_VAL; //I/O 模式
    }
    MT_DCR(0x0081, reg); //向 EBIU 配置寄存器写入控制字
    ... //部分省略
    if (ci_base) {
        iounmap((unsigned long *) ci_base);
    }
    //物理地址到虚拟地址的映射
    if (sw == CI_MODE_MEM) {
        ci_base = CI_PORT_MEM_BASE;
        ci_base = (unsigned long) ioremap(
            ci_base, CI_PORT_MEM_OFFSET_MAX);
    }
    else {
        ci_base = CI_PORT_IO_BASE;
        ci_base = (unsigned long) ioremap(
            ci_base, CI_PORT_IO_OFFSET_MAX);
    }
    if (ci_base == 0) {
        printk("ioremap error\n");
        return -1;
    }
    ci_mode = sw; //记录 EBIU 当前配置模式
    return 0;
}

```

3.3 文件操作函数

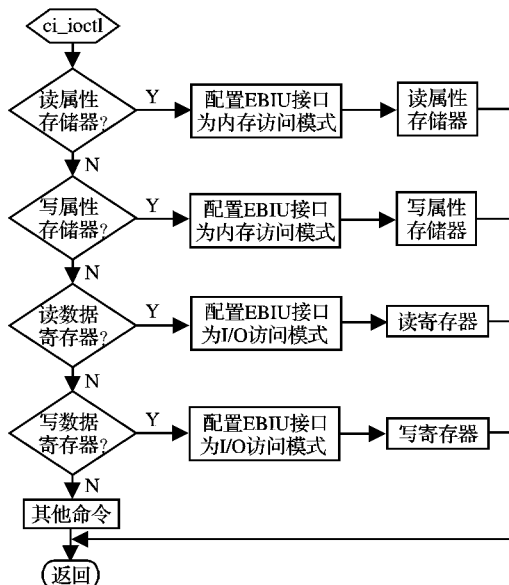


图4 ioctl 函数执行过程

用户空间程序所调用的 open, read, write, ioctl, close 等标准调用, 在驱动程序中都有对应的文件操作函数。文件操作

函数根据需要调用 EBIU 接口配置函数 ci_port_alloc 来配置 EBIU 接口, 完成规定的功能。

ci_open 函数的主要功能是增加模块的引用计数, ci_release 函数的功能是减少模块的引用计数。在设备驱动程序中, 比较重要的是 ci_ioctl, ci_read 和 ci_write 函数, 这些函数的实现都必须结合底层硬件设备的具体特性来进行。图 4 简要显示了 ci_ioctl 函数执行读写属性存储器和数据寄存器命令的流程, 是驱动程序中比较有代表性的部分。在函数执行读写命令之前, 首先要根据命令来判断读写的对象, 从而配置 EBIU 接口采用对应的操作方式, 然后再对内存或寄存器进行操作。

ci_write 和 ci_read 用来完成主机和 CAM 卡之间批量数据的交换, 执行过程与 ci_ioctl 读写数据寄存器类似。其中的主要区别是在对 EBIU 接口进行配置之后, 把主机用户空间的数据循环写入 CAM 卡寄存器或者把 CAM 卡寄存器的数据循环读出到主机的用户空间。

3.4 驱动程序加载、卸载相关函数

```
static struct file_operations ci_fops =
```

```

{
    open: ci_open,
    release: ci_release,
    ioctl: ci_ioctl,
    read: ci_read,
    write: ci_write
};

```

在使用 insmod 命令加载驱动程序的过程中, ci_init 被调用, 这个函数负责模块的初始化, 在执行过程中调用 register_chrdev 来注册与用户空间的标准文件操作相对应的文件操作 ci_fops; 在使用 rmmod 卸载模块的过程中, ci_exit 被调用, 这个函数在执行过程中调用 unregister_chrdev 从系统中删除已经注册的驱动程序。在驱动程序中使用下面的宏对这两个函数进行说明:

```

module_init(ci_init);
module_exit(ci_exit);

```

4 结语

从上面的介绍中我们可以看到, 嵌入式设备的驱动程序与硬件特性密切相关。即使是同一类设备, 在不同的硬件平台上, 其驱动程序也不尽相同。因此, 驱动程序的设计很大程度上受到硬件的制约。而嵌入式 Linux 的文件系统为设备驱动程序提供了一个很好的架构, 既有利于设备驱动程序的开发, 又把设备驱动程序和上层软件隔离开来, 提高了软件的复用性和可移植性。

参考文献:

- [1] IBM. STBx25xx Digital Set-Top Box Integrated Controllers Preliminary Datasheet[Z], 2002.
- [2] EN50221: 1997, Common Interface Specification for Conditional Access and other Digital Video Broadcasting Decoder Applications[S], 1997.
- [3] PCMCIA/JEITA. PC Card Standard Volume 2: Electrical Specification[S], 2001.
- [4] RUBINI L, COR J. Linux 设备驱动程序[M]. 第 2 版. 魏永明, 骆刚, 姜军, 译. 北京: 中国电力出版社, 2002.