基于嵌入式 Linux 的矩阵键盘模块的设计

傅 超[1,2],张昌华[1,2],孟劲松[1,2]

(1.电子科技大学 能源科学与工程学院,四川 成都 611731;
2.电子科技大学 电力系统广域测量与控制四川省重点实验室,四川 成都 611731)

摘 要:针对嵌入式系统的键盘驱动特点,以 Linux 2.6.21 内核为例,提出了一种基于嵌入式 Linux 的矩阵键盘的实现方案。介绍了矩阵键盘的结构及原理,设计了基于 Platform 机制的矩阵键盘驱动程序,并解决了按键去抖及重键问题。通过测试实践,证明该驱动程序工作高效、稳定可靠。

关键词:嵌入式 Linux;Platform 机制;矩阵键盘;键盘驱动程序

中图分类号:TP311 **文献标识码**:A **文章编号**:1674-7720(2012)21-0007-04

Design of matrix keyboard device module based on embedded Linux

Fu Chao[1,2], Zhang Changhua[1,2], Meng Jinsong[1,2]

(1.School of Energy Science and Engineering, University of Electronic Science and Technology, Chengdu 611731, China;
2.The Power System Wide-area Measurement and Control Key Laboratory of Sichuan Province, University of Electronic Science and Technology, Chengdu 611731, China)

Abstract: Mainly considering the characteristics of the embedded system keyboard driver, taking Linux 2.6.21 kernel as the example, the paper proposes an implementation scheme of matrix keyboard based on embedded Linux. The structure and working principle of matrix keyboard were introduced and the matrix keyboard driver based on the platform mechanism was designed, which resolved the Problem of jumping and multi-key. Finally, the driver was proved to be efficient, stable and reliable by the test practice results.

Key words: embedded Linux; Platform mechanism; matrix keyboard; keyboard driver

在嵌入式系统中,Linux 操作系统由于具有开放源码、良好的可移植性、多任务等优势,已成为开发嵌入式产品的优秀操作平台,其中键盘是人机交互设备中重要的输入设备,用于向设备输入数据和信息[1]。在嵌入式系统中,一般使用简易的键盘作为输入设备[2],它由一系列开关矩阵排列而成(包括数字键、字母键、符号键、功能键等)。实现键盘扫描的方法有采用特定芯片和软件方法两种。

采用特定芯片实现键盘扫描,会增加嵌入式系统开发的成本。而利用 ARM 处理器强大的功能,采用软件的方法实现键盘扫描不仅可以降低成本,还可以节省 CPU 的资源开销。因此,本文提出的键盘方案是以嵌入式 Linux 和 AT91RM9200 为软硬件平台,设计了基于 Platform 机制的矩阵键盘驱动程序,解决了按键去抖及重键问题,在实际应用中表明该方案具有很好的稳定性和实时性。

1 矩阵式键盘的结构及原理

硬件平台是基于 CE9200 架构的 AT91RM9200 处理器,工作于 180 MHz 时性能高达 200 MIPS,功耗较低,适用于高性能的嵌入式系统。

在键盘中,排列开关最常用、也最有效的方法是二维矩阵,所需的开关数目根据需求而定,开关放置在行与列的交点上。本系统设计的是一个 4×4 矩阵键盘(k1~k16),由 4 根行线和 4 根列线组成,分别使用 CPU 的 8 个通用输入/输出 GPIO(General Purpose I/O port)口,利用排阻作为上拉电阻。键盘按键使用锅片式,当按下某键,对应行和列的 GPIO 口相互导通[3]。驱动程序初始化时,所有行均为输入端,并设置为高电平;所有列均为输出端,置为低电平。其电路原理图如图 1 所示。

2 Platform 总线模型下键盘驱动

2.1 Platform 总线模型

Platform 总线是 Linux 2.6 kernel 中引入的一种虚拟

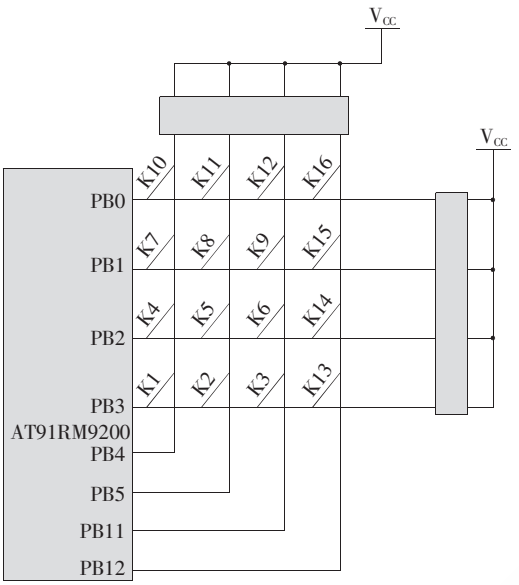

图 1 矩阵键盘电路原理图

总线，Platform 机制中将设备本身的资源注册进内核，由内核统一管理。在驱动程序中通过 platform_device 提供的标准接口进行申请并使用这些资源。platform_driver 通过 platform bus 获取 platform_device，platfrom_driver 的根本目的是为了统一管理系统的外设资源，为驱动程序提供统一的接口来访问系统资源，将驱动和资源分离，从而来提高程序的可移植性[4]。

2.2 键盘驱动

在 Platform 总线模型下，键盘驱动通常是采用层次式结构，由上层键盘抽象层和下层键盘硬件处理层来实现[5]。上层是键盘驱动程序中的核心部分，实现将扫描码转换成键码，再将键码转换成目标码存放到键值缓冲区等功能。上层键盘抽象层中还定义了一些系统调用函数，而这些系统调用功能是由下层硬件处理层来实现的。下层是直接对硬件进行操作，其具体实现是由不同的硬件所决定的。

3 键盘驱动程序的实现

键盘驱动程序的实现可分为初始化函数的实现、系统调用函数的实现以及键盘扫描的实现三部分。

3.1 初始化函数的实现

初始化中主要完成设备注册到系统内核、资源申请、键盘设备检测等工作。其具体实现过程可分为两步：platform_device 与 platform_driver 的定义及初始化、系统探测函数 at91key_probe 的实现。

3.1.1 platform_device 与 platform_driver 的定义及初始化

首先，注册、初始化 platform_device 结构变量，并将 platform_device 添加到 platform 总线；然后再进行设备号的申请；最后对 platform_driver 进行注册，注册函数如下：

```
Ret=platform_driver_register(&at91key_driver);
```

platform_driver_register () 注册时，会将当前注册的 platform_driver 中的 name 变量的值和已注册的所有 platform_device 中的 name 变量的值进行比较，只有找到具有相同名称的 platform_device 才能注册成功。当注册成功时，会调用 platform_driver 结构元素 probe 函数指针（即 at91key_probe）。

3.1.2 系统探测函数 at91key_probe 的实现

在函数指针 at91key_probe 所指向的系统探测函数里，主要完成以下工作：

（1）键盘端口（即 8 个 GPIO 端口）进行初始化，初始化函数如下：

```
Init_Keyboard();
```

在函数 Init_Keyboard 中，所有行的管脚均为输入端，并设置为高电平；所有列的管脚为输出端，并置为低电平。初始化函数如下：

```
void  Init_Keyboard(void)
{
    //行线
at91_set_gpio_input(AT91_PIN_PB0,1);
at91_set_gpio_input(AT91_PIN_PB1,1);
at91_set_gpio_input(AT91_PIN_PB2,1);
at91_set_gpio_input(AT91_PIN_PB3,1);
    //列线
at91_set_gpio_output(AT91_PIN_PB4,0);
at91_set_gpio_output(AT91_PIN_PB5,0);
at91_set_gpio_output(AT91_PIN_PB11,0);
at91_set_gpio_output(AT91_PIN_PB12,0);
at91_sys_write (AT91_PMC_PCER, (0x1 << AT91RM9200_ID_PIOB));
}
```

（2）将已分配到的设备号以及设备操作接口（即为 struct file_operations 结构）赋予 struct cdev 结构变量，用 cdev_init()函数初始化已分配到的结构并与 file_operations 结构关联起来，再调用 cdev_add()函数将设备号与 struct cdev 结构进行关联并向内核正式报告新设备的注册。其注册函数如下：

```
cdev_init(&at91_key->chrdev,&key_fops);
ret=cdev_add(&at91_key->chrdev,dev_id,1);
```

（3）利用函数 class_create 和 class_device_create 自动创建设备节点。其函数如下：

```
key_class=class_create(THIS_MODULE,KEY_NAME);
cls_key_dev =class_device_create (key_class,NULL,dev_id,
&pdev->dev,KEY_NAME);
```

（4）启动系统内核定时器 key_run_timer()，按固定的时间间隔（即定时器处理函数触发时间为 100 ms）来执行键盘扫描函数 Scan_Keyboard()。

3.2 系统调用函数的实现

Linux 为字符设备提供了统一的操作函数接口，内核使用 file_operations 结构建立主设备号和设备驱动程序的连接[6]。file_operations 数据结构指明能够对其设备

文件进行的操作，其中大部分是指向用户自己编写的设备操作函数的函数指针，其相当于一个指针跳转表。在Linux系统中，设备驱动程序以文件系统结构的方式为I/O设备提供一组入口点。因此，对此结构的访问就相当于操作设备文件。

在内核中是使用file_operation结构中函数指针来访问驱动程序的函数，文件可以认为是一个“对象”，操作它的函数是“方法”，这些方法主要负责系统调用的实现。

下面介绍本键盘驱动中打开函数、关闭函数及读函数等系统调用的具体实现。

3.2.1 打开函数及关闭函数的实现

应用程序打开设备文件时，会执行驱动中的打开设备文件描述符的操作。通过file_opreation结构中设备文件操作结构的映射，调用驱动中的key_open函数。此函数主要是使用try_module_get(THIS_MODULE)，去增加管理此设备的THIS_MODULE模块的使用计数。

同样地，当应用程序中使用close函数来关闭设备文件时，实质是通过对应文件的file_opreation结构中的release函数指针来执行系统调用函数key_release。在函数key_release中使用module_put(THIS_MODULE)减少对管理此设备的THIS_MODULE模块的使用计数。

这样，当设备在使用时，管理此设备的模块就不能被卸载，只有设备不再使用时模块才能被卸载。

3.2.2 读函数的实现

键盘读函数通过copy_to_user()函数将从缓冲区读取的键值复制到用户数据区，上层应用程序通过调用读函数即可获取该键值。键盘读函数执行流程如图2所示。

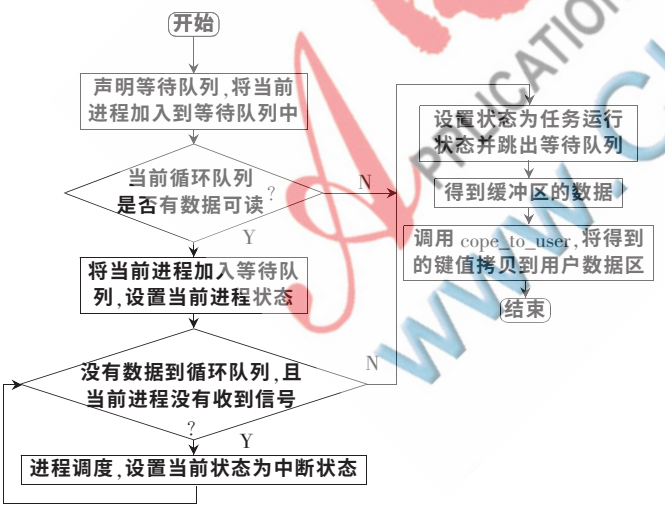

图2 键盘读函数流程图

在键盘驱动中进行读操作时，声明等待队列之后，判断当前循环队列是否有数据可读，若无数据可读，则直接跳出等待队列，得到缓冲区的数据，调用函数cope_to_user，将得到的键值拷贝到用户数据区；若有数据可读，设置当前进程的状态，利用中断状态来等待数据循环队列。当有数据到循环队列，设置状态为任务运行状态并跳出等待队列，缓冲区的数据拷贝到用户数据区。一旦上层用户程序进行读操作，系统调用将通过key_read()函数来获取用户数据区的键值。

等待队列是由等待某些事件发生的进程组成的简单链表。内核中每个等待队列都要一个等待队列头(wake_queue_head)，等待队列头是一个类型为wake_queue_head_t的数据结构。等待队列可通过DECLARE_WAITQUEUE()静态创建。

3.3 键盘扫描的实现

矩阵键盘通常是采用逐行(或列)扫描的方式识别按键，通常分两步进行：(1)识别键盘有无键按下；(2)在有键按下时识别出具体的按键。键盘的工作方式有3种：编程扫描、定时扫描和中断扫描。本方案采用高效率的定时扫描，定时扫描按照内核定时器指定的时间间隔来执行扫描工作。

键盘扫描算法流程图如图3所示。

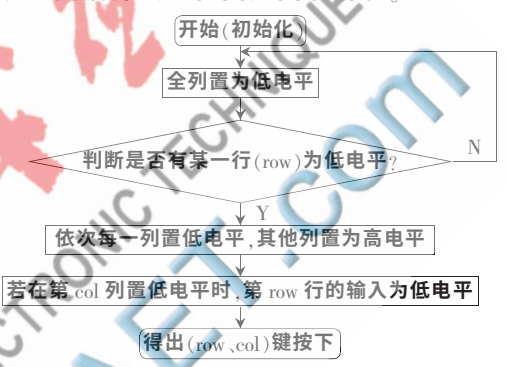

图3 键盘扫描算法流程图

键盘扫描过程是微处理器通过定时查看键盘矩阵以确定是否有键按下，并查询被按下的键。驱动给每个按键分配一个键值，即按键的唯一标识符。应用程序通过按键键值识别被按下的键。初始化时，所有行均为输入端，并设置为高电平，所有的列为输出端，置为低电平；当无键按下时，将从所有作为输入端的行中读到高电平。只要有按键闭合，其中一行将变为低电平。因此，微处理器只需检测是否有某行电平变为低电平即可确定是否有键按下。例如，在图1中，如果PB1变为低电平，则表示k7、k8、k9和k15中至少有一个按键被按下。

在确定按键操作所在行的位置之后，下一步就是要查看按键操作所在列的位置。在4个列输出端口中，轮流将其中某一个端口的输出置为低电平，其他3个端口的输出置为高电平。这样逐列进行扫描，直到按键所在的列端口输出为低电平，此时按键操作所在行的管脚的输入端口的值会变成低电平。例如，在确认k7、k8、k9和k15这行中有按键按下之后，进行逐列扫描。若发现在PB5为低电平时(其他端口输出均为高电平)，PB1管脚的输入端口变为低电平，则可以断定按键k8被按下了。因此可从行号和列号对应的二维数组(也就是键值映射表)中找到该键的键值。

4 按键抖动及重键问题的解决

嵌入式系统中常用机械式按键，由于受到弹性作用的影响，键盘在被按下或释放时，通常会产生机械抖动，

需经过一段时间后才能稳定下来，因此处理器不能随着按键的按下或释放而产生明确的电平 1 或者 0。虽然肉眼看来开关能够快速稳定地闭合，但与处理器运行的速度相比，开关的动作则相对较慢。

为了消除按键抖动的问题，根据开关的回弹特性，处理器按照一定的时间间隔对键盘进行扫描，该时间间隔被称为去除回弹周期，一般为 30 ms~100 ms。

键盘去抖的流程图如图 4 所示，流程描述如下：

（1）初始化时，将键盘的状态标志变量 Bsflag 置为 1。按内核定时器设置的 100 ms 时间间隔对键盘进行逐行扫描，若发现有键按下的信号出现时，此时就要确定是正常击键行为还是抖动。

（2）在检测是不是抖动时，先启动一个延时 20 ms 的定时器，20 ms 之后再次对键盘进行扫描，判断硬件上是否有键按下，若没有，则显然是抖动；若有键按下，则是用户正常击键行为，因此将此键值 iscancode 存入键值缓冲区里，同时 Bsflag=0。之后就启动一个 100 ms 的定时器，这个定时器的作用是判断用户何时松开键盘（注意这里是 100 ms 的定时器，与刚才的 20 ms 不同）。

（3）在 100 ms 定时器定时时间到了之后，要判断此键是否已经弹起。若还是按下，继续启动延时 100 ms 的定时器，在下一个 100 ms 时再进行判断。若是弹起，则要进一步判断是抖动现象还是已完全弹起，进行一个 20 ms 的延迟去抖就可以完全判断出来。当判断出键是完全弹起，则将此键值 iscancode 加上 0x80 存入键值缓冲区里，同时 Bsflag=1。此时按键已经完全地被松开弹起了。

在程序中对键盘标志变量 Bsflag 和键值缓冲区键值（是否小于 0x80）进行有效的判断，完全可以解决按键的防抖及重键问题。

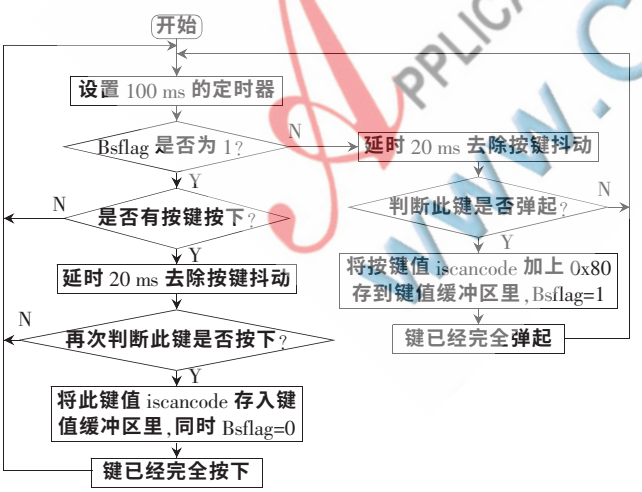

图 4 键盘去抖的流程图

5 键盘驱动的测试

驱动开发完成后，用 insmod 将模块加载入内核，在 PC 和目标板之间搭建好嵌入式交叉编译环境。在硬件平台 CE9200 目标板中的 Linux 系统下进行测试，通过 PC 上的超级终端将测试结果信息打印显示出来。

在图 5 中显示了键盘驱动设备的成功打开与关闭，以及对按键动作信号的高效准确的响应，并成功解决了按键防抖及重键问题，证明本设计的矩阵键盘工作高效、稳定。

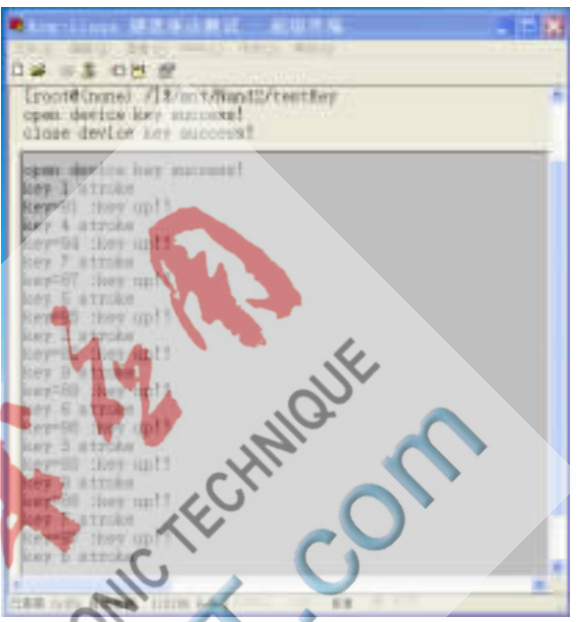

图 5 键盘驱动测试图

本文提出的一种基于 CE9200 平台和嵌入式 Linux 键盘驱动的实现方案，实现了操作的高效和稳定，已成功应用于工程实践中的多款嵌入式设备，证明了在一定的要求下该方案完全能够满足性能要求。

参考文献

[1] 怯肇乾.嵌入式人机界面中的键盘及其接口设计[J].单片机与嵌入式应用系统，2006，20（4）：24-27.

[2] Tool interface standard executable and linking format specification(Version 1.2)[S]. 1995.

[3] 华清远见嵌入式培训中心. 嵌入式 Linux 应用程序开发标准教程（第 2 版）[M].北京：人民邮电出版社，2009：335-356.

[4] 宋宝华.Linux 设备驱动开发详解（第 2 版）[M].北京：人民邮电出版社，2010：243-248.

[5] 林树新，吴朝晖.Linux 键盘驱动的移植分析及实现[J].计算机工程，2005，31（2）：211-213.

[6] Liu Kang, Qian Xu, Li Yaxu, et al. Research of matrix keyboard device driver based on embedded Linux [C]. 2010 Asia-Pacific Conference on Information Network and Digital Content Security (2010APCID), Scientific Research, 17-19 December 2010:239-243.

（收稿日期：2012-05-09）

作者简介：

傅超，男，1987 年生，工学硕士，主要研究方向：嵌入式系统。