# 基于嵌入式Linux的IPMI驱动程序设计

来源：网络 作者：风华正茂 更新时间：2024-12-10

*第一篇：基于嵌入式Linux的IPMI驱动程序设计基于嵌入式Linux的IPMI驱动程序设计2024-04-28丁四华，张志政，东南大学摘 要：针对Linux内核下通用IPMI协议实现部分过于复杂、繁琐、占用过多内存资源，不利于某些简单嵌...*

**第一篇：基于嵌入式Linux的IPMI驱动程序设计**

基于嵌入式Linux的IPMI驱动程序设计

2024-04-28

丁四华，张志政，东南大学

摘 要：针对Linux内核下通用IPMI协议实现部分过于复杂、繁琐、占用过多内存资源，不利于某些简单嵌入式应用场合的不足，提出了通过在FPGA逻辑模拟I2C通道的HOST单板与IPMC子卡的硬件环境上，采用经过简化的IPMI请求/应答消息格式，借助专业Linux的I2C核心框架，实现了相应的IPMI驱动程序功能，方便了管理员监测、管理、诊断系统状态，并根据系统崩溃时的状态，来及时恢复系统。本文详细描述了简化的IPMI协议原理，以及基于Linux内核I2C核心框架的IPMI驱动的各功能模块的设计原理，并详细描述了IPMI驱动功能模块的数据收发流程。研究结果表明，改进的简化IPMI驱动功能模块有效的降低了系统的复杂度、节省了内存资源，达到了精简嵌入式应用系统的目的。关键词：Linux；IPMI协议；FPGA；IPMC子卡；I2C总线 1.引言

IPMI（Intelligent Platform Management Interface）是一种无代理的智能平台管理接口，是由Intel 等公司推出的一个重要的开放标准。IPMI定义了管理员如何监测系统硬件状态，控制系统组件和检索重要系统事件的日志以进行远程管理和恢复。

近年来，广泛采用IPMI协议来监测硬件系统的状况，例如温度、风扇、电压和硬件错误（存储、网络等）和机箱防盗，刀片支持等。由于IPMI可以独立于操作系统之外，即使操作系统已经暂停或服务器已经关闭，管理员照样可以监测、管理、诊断和恢复系统。

针对Linux 内核自带的IPMI驱动过于复杂、繁琐、占用过多内存资源的不足，本文根据ATCA（Advanced Telecom Computing Architecture）方案设计，采用简化的IPMI协议来管理ATCA机框内各硬件单板的物理地址与状态。ATCA平台的HOST单板通过I2C总线接口同IPMC（Intelligent Platform Management Controller）子卡通信,从而获得启动所需的单板物理地址等消息, HOST 单板和IPMC子卡之间采用IPMI协议进行通信。本文讨论了该方案在Linux 下IPMI 驱动程序功能的设计。2.IPMI 协议原理

HOST单板和IPMC子卡之间的硬件接口通过I2C总线来通信，软件上采用IPMI协议。HOST单板不能直接获取单板的相关物理信息，必须要通过IPMI 协议从IPMC子卡上获取。

下面描述IPMI 协议设计要点。

（1）IPMI协议采用Request/Response 的模式，我们通常把IPMI的请求消息称为命令，通过采用Request/Response 模式可使IPMI消息在不同的传输通道上传送，在我们的系统中，采用了I2C总线作为物理上面的传输通道进行消息的传送，上面传输的消息的格式按照IPMI协议规定的消息格式。

（2）IPMI命令是一个功能命令的集合，通过IPMI消息中的Network Function字段来表示，这些命令集合中包括了与事件相关的一些命令集合。通过在命令集合里面不同的字段的不同含义来代表该条消息的具体请求。

（3）IPMI中所有的请求消息中都包括一个网络功能(Network Function)，命令(command)，以及可选的数据(data)字段。IPMI的响应消息和IPMI的请求消息采用同样的消息格式。IPMI消息分为请求和应答两种消息，具体的消息格式如下所示：

图1 IPMI请求消息格式

图2 IPMI应答消息格式

在IPMI消息中，一次请求消息的最大长度不能超过32Bytes，如图1中所示，在这最长的32个BYTES的消息结构中，包括了接收该消息的地址（Responser Address），自身的地址(Requester Address)，消息中的NetFn/LUN字段总共是一个BYTE,前面6个bits表示的是NetFn，后面的2个bits表示的请求方的LUN，Seq/Lun字段与NetFn/LUN字段相似，只不过前面的6个bits表示的是序列号，command字段用来表示具体的命令，data字段中存放的是具体的数据，最大长度不能超过25bytes，除了上面的三个字段外，还有两个校验和字段，分别用来对消息头和消息体来进行校验。

应答消息（参见图2）和请求消息在结构上面相同，在回应请求的应答消息中用该将请求的地址和响应的地址字段相互调换，序列号保持不变，NetFn字段加1，LUN填写的是将请求消息中的字段互换。例如在HOST通过IPMC获取32bits的物理信息，HOST按照请求的消息格式发送一条请求到IPMC，IPMC收到后，按照响应的消息的格式发送应答到HOST，并将物理信息放到应答消息的DATA字段中，HOST接收到回应后就可以从DATA字段中直接获取到相关的数据信息。3.Linux下的IPMI驱动设计

Linux下IPMI模块驱动采用分层设计，详细设计说明见各以下各小节。3.1 Linux下的IPMI驱动架构

Linux下IPMI模块驱动分为两部分，一部分为I2C驱动模块，位于Linux内核态，为IPMI驱动提供底层Linux 硬件驱动；另一部分为BSP子系统PMI接口封装层，位于Linux用户态，为上层应用提供IPMI模块初始化、消息发送、消息接收接口。模块位置参见图3。

图3 Linux下的IPMI驱动架构

3.2 硬件环境描述

I2C总线是PHILIPS（飞利浦）公司推出的两线式串行总线，用于连接微控制器及其外围设备，具有简单、高效等特点。由于其接口直接在组件之上，因此I2C总线占用的空间非常小，减少了电路板的空间和芯片引脚的数量，降低了互联成本，特别适用于嵌入式产品。

HOST单板通过FPGA（Field Programable Gate Array）逻辑模拟的I2C模块与IPMC子卡通过FPGA逻辑模拟的I2C模块通信，该I2C模块目前只支持主发，从收功能。参见图4。另外，在该条I2C总线上，可以同时挂接其它的I2C设备，例如，温度监控器等。

图4 HOST单板与IPMC子卡通信图示

3.3 Linux I2C驱动模块设计

Linux下IPMI软件模块代码在整个Linux系统中属于字符设备驱动，按照模块主要功能来划分，整个驱动大体可以分为以下五个模块：

（1）I2C核心框架功能实现子模块；（2）FPGA逻辑模拟I2C设备驱动子模块；

（3）FPGA逻辑模拟I2C适配器驱动子模块；

（4）I2C设备方法子模块；

（5）IPMI用户态接口封装子模块。3.3.1 I2C核心框架功能实现子模块

该模块提供了核心数据结构的定义、I2C适配器驱动和设备驱动的注册、注销管理，I2C通信方法上层的、与具体适配器无关的代码、检测设备地址的上层代码等。

内核中I2C 相关代码可以分为三个层次（参见图5）：

（1）I2C core框架：提供了核心数据结构的定义和相关接口函数，用来实现I2C适配器驱动和设备驱动的注册、注销管理，以及I2C通信方法上层的、与具体适配器无关的代码，为系统中每个I2C总线增加相应的读写方法。

（2）I2C总线适配器驱动：定义描述具体I2C总线适配器的i2c\_adapter数据结构、实现在具体I2C适配器上的I2C总线通信方法，并由i2c\_algorithm数据结构进行描述。

（3）I2C设备驱动：定义描述具体设备的i2c\_client和可能的私有数据结构、借助I2C core提供的函数接口完成设备在内核的注册，并实现具体的功能，包括read, write以及ioctl等对用户层操作的接口。

总体而言，Linux中I2C总线的驱动分为两个部分：总线驱动(BUS)和设备驱动(DEVICE)。I2C core与I2C总线适配器驱动完成了硬件上的主机总线驱动(BUS)，而I2C driver则实现了从机设备驱动。在设计中，I2C core提供的接口是统一的，不需要修改，我们只需要实现特定I2C总线适配器驱动和I2C设备驱动，这样大大提高了系统的可移植性。

图5 Linux内核I2C框架模块关系图示

3.3.2 FPGA 逻辑模拟I2C 设备驱动子模块

该模块描述具体设备的i2c\_client和可能的私有数据结构、借助I2C框架的i2c\_probe函数实现注册设备的attach\_adapter方法、提供设备可能使用的地址范围、以及设备地址检测成功后创建i2c\_client数据结构的回调函数。这部分功能主要由bsp\_ipmi\_drv.c文件来描述。3.3.3 FPGA逻辑模拟I2C适配器驱动子模块

FPGA模拟I2C适配器驱动子模块，定义描述具体I2C总线适配器的i2c\_adapter数据结构、实现在具体I2C 适配器上的I2C总线通信方法，并由i2c\_algorithm 数据结构进行描述。这部分功能主要由bsp\_ipmi\_adap.c文件来描述。

3.3.4 I2C设备方法子模块

该模块提供创建I2C适配器的/dev/ipmi/%d 设备节点，提供IPMI设备访问方法（read & write, open & release, ioctl）等，这部分功能主要由bsp\_ipmi\_devintf.c文件来描述。3.3.5 IPMI用户态接口封装子模块 该模块主要实现I2C通信从设备的选取、I2C设备方法系统调用的封装，和IPMI消息封装、数据收发的格式化等操作。这部分功能主要由bsp\_ipmi\_api.c文件来描述。

3.3.6 采用I2C框架的IPMI驱动的数据收发流程

IPMI模块驱动采用内核I2C核心框架，HOST单板IPMI模块驱动的数据收发流程分为四个层次（参见图6）

数据发送流程：

（1）上层用户调用Linux用户态IPMI消息发送函数BSP\_IpmiSend（）函数向IPMC子卡发送消息；

（2）BSP\_IpmiSend（）函数通过write系统调用切换到Linux内核态，调用I2C设备接口驱动层的write方法实现ipmcdev\_write（）函数；

（3）ipmcdev\_write（）函数通过调用I2C核心层i2c\_master\_send（）函数来发送消息；

（4）i2c\_master\_send（）函数最后调用具体I2C适配器驱动层的i2c\_ipmc\_xfer（）函数来操作I2C总线，该函数受到I2C框架总线锁的保护，任何时刻只能由一个进程访问，符合硬件I2C总线的实际情况。

（5）i2c\_ipmc\_xfer（）函数根据入参调用ipmc\_write（）函数来向I2C 硬件设备发送数据。

数据接收流程：

（1）上层用户调用Linux用户态IPMI消息接收函数BSP\_IpmiRecv（）函数接收来自IPMC子卡的消息；

（2）BSP\_ IpmiRecv（）函数通过read系统调用切换到Linux内核态，调用I2C设备接口驱动层的read方法实现ipmcdev\_read（）函数；

（3）ipmcdev\_read（）函数通过调用I2C核心层i2c\_master\_recv（）函数来接收消息；

（4）i2c\_master\_read（）函数最后调用具体I2C适配器驱动层的i2c\_ipmc\_xfer（）函数来操作I2C总线，该函数受到I2C框架总线锁的保护，任何时刻只能由一个进程访问，符合硬件I2C总线的实际情况。

（5）i2c\_ipmc\_xfer（）函数根据入参调用ipmc\_read（）函数来阻塞接收来自I2C硬件设备的数据。

图6 IPMI模块采用I2C核心框架的数据收发流程

4.总结

本文主要介绍了简化的嵌入式Linux的IPMI驱动程序设计。通过全文可以看出，基于FPGA逻辑模拟I2C通道的HOST 单板与IPMC 子卡的硬件环境，借助Linux内核稳定而又专业的I2C核心框架，使得设计和实现基于I2C核心框架的简化IPMI驱动变得非常容易。同时本文详细描述了IPMI协议的原理和简化后的消息格式。通过该驱动，管理员可以方便的监测、管理、诊断系统状态，并恢复系统。总之，改进的简化IPMI驱动功能模块有效的降低了系统的复杂度、节省了内存资源，达到了精简嵌入式应用系统的目的。

**第二篇：基于嵌入式Linux的设备驱动程序设计**

基于嵌入式Linux的设备驱动程序设计

Linux为是一个成熟而稳定的操作系统。将Linux植入嵌入式设备具有众多的优点，包括可剪裁和容易移植等，所以Linux操作系统在嵌入式领域获得了广泛的应用。嵌入式Linux一直是嵌入式领域的研究热点，与PC架构不同，嵌入式系统的硬件具有多样性和差异性，嵌入式系统的开发需要对特定系统进行硬件设计，同时还要针对这些硬件来编写驱动程序。Linux内核就是通过驱动程序来同外围设备打交道的，系统设计人员必须为每个设备编写驱动程序，否则设备无法在操作系统下正常工作。设备驱动程序设计的基本概念与模型

设备驱动程序是操作系统内核与机器硬件之间的接口，它为应用程序屏蔽了硬件的细节，在应用程序看来，硬件设备只是一个设备文件，应用程序可以象操作普通文件一样对硬件设备进行操作，设计驱动程序是内核的一部分，可以实现以下功能：

对设备初始化和释放;

把数据从内核传送到硬件，以及从硬件读取数据;

读取应用程序传送给设备文件的数据，以及回送应用程序请求的数据;

检测和处理设备出现的错误。

前面已经提到驱动程序的作用，而编写驱动程序就是构造一系列可供应用程序调动的函数(包括open、release、read、write、llseek、ioctl等)。在用户自己的驱动程序中，首先要根据驱动程序的功能，实现file\_operations结构中的函数，不需要的函数接口可以直接在file\_operations结构中初始化为NULL;file\_operations变量会在驱动程序初始化时注册到系统内部。当操作系统对设备操作时，会调用驱动程序注册的file\_operations结构中的函数指针。

以下是嵌入式linux2.4设备驱动程序的最简模型。

具体实现前面定义的函数时，需注意下面几点：

1)在test\_init函数中要通过调用register\_chrdev()函数来向内核注册字符设备驱动程序。如果是块设备，则还需调用mmmap()进行地址空间的映射，再调用register\_blkdev()函数来向内核注册块设备驱动程序，在Linux系统中，对中断的处理是属于系统核心部分，因而如果设备与系统之间以中断方式进行数据交换，则必须把该设备的驱动程序作为系统核心的一部分，也就是说设备驱动程序要通过调用request\_irq()函数来申请中断，通过free\_irq()函数来释放中断(在test\_cleanup中实现)。

2)open()函数和release()函数的具体实现有着一定的对应性，在open()函数中主要是执行打开设备时的一些初始化代码，如果该驱动程序需要管理多个设备，那么还要获取从设备号，根据从设备号来判断需要操作的设备，其中，从设备号可通过调用函数MINOR(inode->i\_rdev)来获得，然后再调用宏MOD\_INC\_USE\_COUNT来使得驱动程序使用计数器加1，而在release()函数中则要进行相反的处理。即调用宏MOD\_DEC\_USE\_COUNT来减小驱动程序使用计数器。

3)归根到底，驱动函数的实现就是调用内核所支持的函数(包括内核提供的API和用户自己定义的寄存器操作函数)来完成对设备的操作，虽然嵌入式系统设备的种类众多，不同设备操作的具体实现方法不可能相同，但是Linux操作系统提供了一系列特殊API，为开发内核驱动程序带来了很大的方便，调用这些API时需要注意的是：通常情况下，应用程序是通过内核接口访问驱动程序的(这是驱动程序的主要使用方式)，因此驱动程序需要与应用程序交换数据，但是操作系统内核和驱动程序在内核空间中运行，而用户程序在用户空间中运行，用户程序不能访问内核空间，操作系统内核和驱动程序也不能使用指针或memcpy()等常规的C库函与用户空间传输数据，造成这种状况的主要原因是linux操作系统使用了虚拟内存机制，使用了虚拟内存机制后，用户空间的内存可能被换出，当内核使用用户空间指针时，对应的页面可能已经不在内存中了，因此在使用调用函数时要注意：设备驱动程序在申请和释放内存时不是调用malloc()和free()，而调用kmalloc()和kfree();用于内核空间与用户空间进行数据拷贝的函数主要有access\_ok()(检查某内存空间是否有权访问)，copy\_to\_user()和put\_usr()(内核函数向用户空间传输数据)，copy\_from\_user()和get\_user()(用户空间向内核空间传输数据);关于内核空间与I/O空间的数据交换，不同体系结构的处理器对I/O的处理方式也不同，x86系列处理器中，I/O与内存完成不同，它是分开编址的，访问它要使用专用的指令;而对ARM体系结构的处理器来说，则是不区分I/O和内存，统一编址的，可以使用同样的指令访问，在驱动程序中可以使用一系列函数来访问I/O口，如outb()、outw()、outl()inb()、inw()、inl()、outsb()、outsw()、outsl()、insb()、insw()和insl()等。

Linux2.6与2.4内核驱动程序的区别

为了彻底防止对正在被使用的内核模块进行错误操作，linux2.6内核在加载和导出内核模块方面都较2.4内核有所改进，避免了用户执行将导致系统崩溃的操作(例如强制删除模块等)。同时，当驱动程序需要在多个文件中包含 头文件时，不必定义宏来检查内核的版本。与2.4内核相比，2.6内核在可扩展性、吞吐率等方面有较大提升，其新特性主要包括：使用了新的调度器算法;内核抢占功能显著地降低了用户交互式应用程序;多媒体应用程序等类似应用程序的延迟;改进了线程模型以及对NPTL的支持，显著改善了虚拟内存在一定成程度负载下的性能;能够支持更多的文件系统;引进了内存池技术;支持更多的系统设备，在2.4内核中有约束大型系统的限制，其支持的每一类设备的最大数量为256，而2.6内核则彻底打破了这些限制，可以支持4095种主要的设备类型，且每个单独的类型又可以支持超过一百万个的子设备;支持反向映射机制(reverse mapping)，内存管理器为每一个物理页建立一个链表，包含指向当前映射页中每个进程的页表条目的指针。该链表叫PTE链，它极大的提高了找到那些映射某个页的进程的速度。

Linux操作系统的设备驱动程序是在内核空间运行的程序，其中涉及很多内核的操作，随着Linux内核版本的升级，驱动程序的开发必然也要作出相应的修改，总之，在linux2.6内核上编写设备驱动程序时具体要注意以下几个方面：

1)Linux2.6内核驱动程序必须由MODULE\_LICENSE(“Dual BSD/GPL”)语句来定义许可证，而不能再用2.4内核的MODULE\_LICENSE(“GPL”)。否则，在编译时会出现警告提示。

2)Linux2.6内核驱动程序可以用int try\_module\_get(&module)来加载模块，用module\_put()函数来卸载模块，而以前2.4内核使用的宏MOD\_INC\_USE\_COUNT和MOD\_DEC\_USE\_COUNT则可不用。

3)前面给出的字符型设备驱动程序模型中结构体file\_operations的定义要采用下面的形式。这是因为在Linux内核中对结构体的定义形式发生了变化，不再支持原来的定义形式。

4)就字符型设备而言，test\_open()函数中向内核注册设备的调用函数register\_chrdev()可以升级为int register\_chrdev\_region(dev\_t from，unsigned count，char \* name)，如果要动态申请主设备号可调用函数int alloc\_chrdev\_region(dev\_t \* dev,unsigned baseminor,unsigned count,char \* name)来完成;原来的注册函数还可以用，只是不能注册设备号大于256的设备，同理，对于块设备和网络设备的注册函数也有着相对应的代替函数。

5)在声明驱动程序是否要导出符号表方面有着很大的变化。当驱动程序模块装入内核后，它所导出的任何符号都会变成公共符合表的一部分，在/proc/ksyms中可以看到这些新增加的符号。通常情况之下，模块只需实现自己的功能，不必导出任何符号，然而，如果有其他模块需要使用模块导出的符号时，就必须导出符号，只有显示的导出符号才能被其他模块使用，Linux2.6内核中默认不导出所有的符号，不必使用EXPORT\_NO\_SYMBOLS宏来定义;而在2.4内核中恰恰相反，它默认导出所有的符号，除非使用EXPORT\_NO\_SYMBOLS，因此在上面给出的范例中可以省略去该定义语句。

6)Linx内核统一了很多设备类型，同时也支持更大的系统和更多的设备，原来Linux2.4内核中的变量kdev\_t已经被废除不可用，取而代之的是dev\_t。它拓展到了32位，其中包括12位主设备号和20位次设备号。调用函数为unsigned int iminor(struct inode \* inode)和unsigned int imajor(struct inode \* inode)，而不再用Linux2.4版本中的int MAJOR(kdev\_t dev)和int MINOR(kdev\_t dev)。

所有的内存分配函数不再包含在头文件 中，而是包含在 中，而原来的 已经不存在。所以当在驱动程序中要用到函数kmalloc()或kfree()等内存分配函数时，就必须要定义头文件 而不是。同时，前面提到的申请内存和释放内存函数的具体参数也有了一定的改变，包括：分配标志GFP\_BUFFER被取消，取而代之的是GFP\_NOIO和GFP\_NOFS;新增了\_GFP\_REPEAT、\_GFP\_NOFAIL和\_GFP\_NORETRY分配标志等，使得内存操作更加方便。

8)因为内核中有些地方的内存分配是不允许失败的，所以为了确保这种情况下得成功分配，linux2.6版本内核中开发了一种称为“内存池”的抽象。内存池其实相当于后备的高速缓存，以便在紧急状态下使用。要使用内存池的处理函数时，必须包含头文件。内存池处理函数主要有以下几个：mempool\_t \*mempool\_create()、void\*mempool\_alloc()、void mempool\_free()、int mempool\_resize();

另外值得一提的是：2.6内核为了区别以.o为扩展名的常规对象文件，将内核模块的扩展名改为.ko，所以驱动程序最后是被编译为ko后缀的可加载模块，在应用程序中加载驱动程序模块时要注意。结语

驱动程序的开发作为嵌入式linux系统开发过程当中最重要的环节之一，与硬件特性和操作系统的内核有着紧密的联系。随着linux内核版本的升级，内核驱动程序必然要作出相应的改进，相信随着嵌入式Linux系统在各个领域中的广泛应用，具有可抢占实时性的Linux2.6内核必定会在嵌入式领域大显身手。本文会对广大的驱动程序开发人员有一定的帮助。

**第三篇：嵌入式Linux驱动**

嵌入式Linux驱动 简介

设备驱动程序是操作系统内核和机器硬件之间的接口。设备驱动程序为应用程序屏蔽了硬件的细节，这样使硬件对应用程序来说是透明的，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。设备驱动程序是嵌入式Linux内核的一部分，它完成以下的功能： · 对硬件设备初始化和释放

· 把数据从内核传送到硬件，或从硬件读取数据

· 读取应用程序传送给设备文件的数据和回送应用程序请求的数据 ·检测和处理设备出现的错误和异常

Linux系统的设备分为字符设备，块设备和网络设备。用户进程通过设备文件实现与硬件的交流。每个设备文件都有其文件属性，表示是字符设备还是块设备。另外每个文件都有两个设备号：第一个是主设备号，标识驱动程序；第二个是从设备号，标识使用同一个设备驱动程序的不同硬件设备。

设备驱动程序可以分为3个主要组成部分：(1)自动配置和初始化

用于负责检测所要驱动的硬件设备是否存在和是否能正常工作。如果该设备正常，则对这个设备及其相关设备驱动程序需要的软件状态进行初始化，如设置寄存器的值，初始化驱动程序用到的数据结构。这部分驱动程序仅在初始化的时候被调用一次。

(2)服务于I/O请求的子程序

调用服务于I/O请求的子程序是由于系统调用的结果，如read，write调用。这部分程序在执行的时候，系统仍认为是和进行调用的进程属于同一个进程，只是由用户态变成了核心态，它们的运行环境和进行此系统调用的用户程序一样，因此可以在其中调用sleep()等与进程运行环境有关得函数。(3)中断服务子程序

在UNIX系统中，并不是直接从中断向量表中调用设备驱动程序的中断服务子程序，而是由UNIX系统来接收硬件中断，再由系统调用中断服务子程序。中断可以发生在任何一个进程运行的时候，因此在中断服务程序被调用的时候，不能依赖于任何进程的状态，也就不能调用任何与进程运行环境有关的函数。因为设备驱动程序一般支持同种类型的若干设备，所以一般在系统调用中断服务子程序的时候，都带有一个或多个参数，以唯一标识请求服务的设备。2 一个简单的Linux2.6内核驱动模块(hello world)/\* hello.c \*/ #include /\* Needed by all modules \*/ #include /\* Needed for KERN\_ALERT \*/ #include /\* Needed for the module-macros \*/ static int \_\_init hello\_init(void)// Module entry function specified by module\_init(){ printk(KERN\_ALERT “Hello,world!n”);return 0;} static void \_\_exit hello\_exit(void)//Module exit function specified by module\_exit(){ printk(KERN\_ALERT “Goodbye,cruel world!n”);} module\_init(hello\_init);module\_exit(hello\_exit);MODULE\_LICENSE(“Dual BSD/GPL”);//should always exist or you’ll get a warning MODULE\_AUTHOR(“BENSON”);//optional MODULE\_DESCRIPTION(“STUDY\_MODULE”);//optional /\* Makefile \*/ # Makefile 2.6 obj-m = hello.o KDIR:=/lib/modules/$(shell uname-r)/build #PWD=$(shell pwd)all: make-C $(KDIR)M=$(PWD)modules clean: make-C $(KDIR)M=$(PWD)clean \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* obj-m := hello.o表示编译后生成hello.o模块。

$(KDIR)指定了内核源码的路径，“M=”表示这是个外部模块，M=$(PWD)指定了该模块文件所在的路径。

注： makefile预定义了$(PWD)变量，此处可以不必重复定义。执行#make编译成功后 加载模块

#insmod hello.ko 字串7 #lsmod 输出内核已加载模块信息，可以查看到刚刚加载成功的hello模块 „„

Module Size Used by hello 5632 0 可以在日志里查看加载模块时的信息 #vi /var/log/messages „„

Sep 27 13:25:21 localhost kernel: Hello,world!卸载模块 #rmmod hello.ko #lsmod 发现hello模块已经被卸载 查看日志信息

#vi /var/log/messages „„

Sep 27 13:26:58 localhost kernel: Goodbye,cruel world!3 一个简单Linux2.6内核的字符驱动程序

Linux下的设备驱动程序被组织为一组完成不同任务的函数的集合，通过这些函数使得Windows的设备操作犹如文件一般。在应用程序看来，硬件设备只是一个设备文件，应用程序可以象操作普通文件一样对硬件设备进行操作，如open()、close()、read()、write()等。

Linux主要将设备分为二类：字符设备和块设备。字符设备是指设备发送和接收数据以字符的形式进行；而块设备则以整个数据缓冲区的形式进行。字符设备的驱动相对比较简单。//globalvar.c #include #include #include #include MODULE\_LICENSE(“GPL”);#define MAJOR\_NUM 254 //主设备号

static ssize\_t globalvar\_read(struct file \*, char \*, size\_t, loff\_t\*);static ssize\_t globalvar\_write(struct file \*, const char \*, size\_t, loff\_t\*);//初始化字符设备驱动的file\_operations结构体 struct file\_operations globalvar\_fops = { read: globalvar\_read, write: globalvar\_write, };static int global\_var = 0;//“globalvar”设备的全局变量 static int \_\_init globalvar\_init(void){ int ret;

//注册设备驱动

ret = register\_chrdev(MAJOR\_NUM, “globalvar”, &globalvar\_fops);if(ret){

printk(“globalvar register failure”);} else {

printk(“globalvar register success”);} return ret;} static void \_\_exit globalvar\_exit(void){ int ret;//注销设备驱动

ret = unregister\_chrdev(MAJOR\_NUM, “globalvar”);

if(ret){

printk(“globalvar unregister failure”);} else {

printk(“globalvar unregister success”);} } static ssize\_t globalvar\_read(struct file \*filp, char \*buf, size\_t len, loff\_t \*off){ //将global\_var从内核空间复制到用户空间

if(copy\_to\_user(buf, &global\_var, sizeof(int))){

returnEFAULT;} return sizeof(int);} module\_init(globalvar\_init);module\_exit(globalvar\_exit);//makefile 同上面的hello world，只需修改hello.o为globalvar.o便可以。接着我们创建设备节点，用户进程通过/dev/globalvar这个路径就可以访问到这个全局变量虚拟设备了。： mknod /dev/globalvar c 254 0 我们写一个用户态的程序globalvartest.c来验证上述设备： #include #include #include #include main(){ int fd, num;//打开“/dev/globalvar” fd = open(“/dev/globalvar”, O\_RDWR, S\_IRUSR | S\_IWUSR);if(fd!=-1){

//初次读globalvar

read(fd, &num, sizeof(int));

printf(“The globalvar is %dn”, num);

//写globalvar printf(“Please input the num written to globalvarn”);scanf(“%d”, &num);write(fd, &num, sizeof(int));

//再次读globalvar

read(fd, &num, sizeof(int));

printf(“The globalvar is %dn”, num);

//关闭“/dev/globalvar” close(fd);} else {

printf(“Device open failuren”);} }

1、引言

记得在学习VC++和C语言的时候，一开始都会以一个HELLO WORLD的例子作为演示，将学者逐渐引入殿堂，这个几乎成了计算机编程语言学习必经的一个入门之路。

当然，在学习linux编程的时候也是这样，下面的例子应该是再熟悉不过了：

首先用VI编写一个C程序：vi hello.c #include “stdio.h” int main(){ printf(“hello world！!n”);return 0;} 接着用GCC进行编译：gcc-o hello hello.c 最后运行该程序：./hello 在终端上你会看到：hello world！!上面的是在操作系统基础上进行的用户应用程序的开发。然而对于linux驱动程序的开发是绝然不同的，因为驱动程序的开发是运行在内核空间的，而应用程序是运行在用户空间的。虽然hello world是一个简单得不能再简单的程序，但是对于嵌入式linux驱动程序的初学者来说，通过这个过程的操作可以对linux驱动程序开发的过程和其中的一些概念有一个深刻的认识。所以，我在这里也就以前学习linux的基础上整理了一下，写了这篇博客。一方面是自己对这方面知识的回顾和巩固，另一方面更是希望这里的内容能给大家提供那么一点点有用的信息，小弟心里就很高兴了。当然希望有高手可以做下评价和指导，及时纠正小弟的错误，谢谢先。

2、概念

驱动程序作为系统内核的一部分，它工作在核心态，而应用程序工作在用户态。也就是说，程序不能直接通过指针，把用户空间的数据地址传递给内核（因为MMU映射的地址根本不一样）。要想在应用程序和驱动程序之间传递数据（指针），就需要经过转换。把用户态“看到”的空间地址转换成内核态可访问的地址。Linux系统提供了一系列方便的函数实现这种转换，如get\_user、put\_user、copy\_from\_user、copy\_to\_user等，它们自己负责访问权限的检查，使用时，不需要关系更多的问题。

Linux内核把驱动程序划分为3种类型：字符设备、块设备和网络设备。字符设备和块设备可以像文件一样被访问。它们的主要区别不在于能否seek，而是在于系统对于这两种类型设备的管理方式。应用程序对于字符设备的每一个I/O操作，都会直接传递给系统内核对应的驱动程序；而应用程序对于块设备的操作，要经过系统的缓冲区管理，间接传递给驱动程序处理。块设备的这种管理方式是为存储提供优化的；而字符设备的管理方式是为操作提供优化的。至于网络设备，它在Linux系统中是一类比较特殊的设备它不像字符设备或块设备那样通过对应的设备文件节点去访问，内核也不再通过read和write等调用去访问网络设备。Linux的网络系统主要是基于BSD UNIX的套接字机制，在系统和驱动程序之间有专门的数据结构进行数据传输，系统支持对数据发送和数据接收缓存，提供流量控制机制，提供更多的协议支持。

在linux系统中，驱动程序都做成模块的形式，也就是module。简单的说，一个模块提供一个功能，这些模块是可以按照需要随时装入内核空间和从内核空间卸载的。因此，内核模块是为了给内核动态增减功能而设计的，并不仅仅是限于驱动程序。

关于内核模块初始化（加载）函数

当用户输入命令“insmod 模块文件名”（或者其他方式）加载内核模块时，系统会检测此模块能否被加载，如果能被加载，内核调用模块的初始化函数。在linux 2.4中，内核模块的初始化函数名为init\_module()。但如果驱动程序需要编译进内核，则初始化函数不能与内核的其他部分（包括其他内核模块）的函数同名。这样，如果程序可能编译到内核中时就比较麻烦。不过在这个头文件中定义了宏module\_init()，用来屏蔽两者的差别，事实上，使用这个宏可以使驱动程序向上兼容linux 2.6，而兼容linux 2.0也比较方便。

关于内核模块清除（卸载）函数

当用户输入命令“rmmod 模块文件名”（或者其他方式）卸载内核模块时，此时，系统会检测此模块是否能被卸载，内核将调用模块清除函数。在linux 2.4中，清除函数的函数名称为cleanup\_module()。但如果驱动程序需要编译进内核，则初始化函数不能与内核的其他部分（包括其他内核模块）的函数同名。这样，如果程序可能编译到内核中时就比较麻烦。不过在这个头文件中定义了宏module\_exit()，用来屏蔽两者的差别。

3、实例

因为内核模块需要加载到内核空间，所以其程序的编写与一般应用程序不同，在里面再也找不到类似main()这样的入口函数，下面对应函数相应的源代码hello.c，介绍一个驱动模块的写法。(由于尖括号不能在博客上显示，固用双括号代替，在编程调试的时候换回来就可以了)。

#ifndef \_\_KERNEL\_\_ #define \_\_KERNEL\_\_ #endif #ifndef MODULE #define MODULE #endif

#include 《linux/module.h》 //所有模块都需要的头文件 #include 《linux/sched.h》 #include 《linux/kernel.h》

#include 《linux/init.h》 // init和exit相关宏

MODULE\_LICENSE(“GPL”);

int text\_init(void){ printk(“Hello World!”);return 0;} void text\_cleanup(void){ printk(“Goodbye World!”);}

module\_init(text\_init);//注册加载时执行的函数 module\_exit(text\_cleanup);//注册卸载时执行的函数、调试

一个Linux内核模块需包含模块初始化和模块卸载函数，前者在insmod的时候运行，后者在rmmod的时候运行。初始化与卸载函数必须在宏module\_init和module\_exit使用前定义，否则会出现编译错误。

程序中的MODULE\_LICENSE(“GPL”)用于声明模块的许可证。

编译：

gcc-c-I /usr/src/linux-2.4/include/ hello.c 运行：

insmod hello.o 终端上会显示：

localhost kernel:Hello World!同时在/proc/modules里面会看到相应的设备信息：

more /proc/modules 你会看到(或许后面的数值不一样)：

hello 844 0(unused).......卸载驱动程序：

rmmod hello 你将会在终端上面看到：

localhost kernel:Goodbye World!

5、注意：

在gcc编译选项中增加-c 在gcc编译选项中定义两个宏：-DMODULE-D\_\_KERENL\_\_ 或直接在源文件中定义这两个宏：

#define MODULE #define \_\_KERNEL\_\_ 在源文件中包括module.h文件：

＃include “linux/module.h”

假定你现在运行的内核的源码目录绝对路径是MyKernelSrcPath，在gcc编译时增加选项：

-I $MyKernelSrcPath/include(如-I /usr/src/linux/include)某些时候用insmod-f能够成功加载，但需谨慎使用。如果看不到用printk打印的信息，可以用dmesg命令看。打印消息受级别的限制,消息级别可以通过printk设置,如: printk(“ 《n》something”);/\* 其中0 /proc/sys/kernel/printk

**第四篇：嵌入式程序设计课程设计**

课程设计

课 程 名嵌入式软件开发技术

题 目 基于嵌入式Linux的温度监测系统的

设计与实现

专 业 计算机科学与技术（嵌入式系统方向）班 级 13计算机嵌入式系统班 学 号 学生姓名

2024年6月

摘要

温度是个很普遍而又非常重要的参数，在日常生活、工农业生产以及科研领域都有着广泛的应用。因此，研制能够准确地测量和记录这个参数值的系统具有十分重要的意义。

基于ARM的嵌入式温度监测系统是采用嵌入式Linux作为操作系统，针对以S5PV210为处理器的开发板设计的一个嵌入式温度监测系统。论文在分析了Linux设备驱动程序的基本工作原理基础上，讨论了开发中经常会碰到的中断处理、拥塞处理、I/O端口，并在此基础上实现了基于S5PV210嵌入式处理器的开、读、写、关外部RAM的字符设备驱动和网络驱动。结合高精度温度传感器DS18B20，实现温度的正确采集，并通过以太网络将数据上传给上位机客户端。

论文首先介绍了通信网络中各种设备特性、总线结构及传输技术，然后根据单片机与PC机之间的串行通信原理，用ubantu完成温度监测系统的软件设计与实现，为用户提供一个友好的人机界面，对监测系统进行控制并显示采集后的数据。本系统还通过多线程实现了多个客户端与服务器的通信。

关键词：S5PV210；嵌入式Linux操作系统；DS18B20；网络编程

I

Abstract

Temperature is a very common and very important parameter, in daily life, industrial and agricultural production and scientific research fields have a wide range of applications.Therefore, it is very important to develop a system that can accurately measure and record the value of this parameter.The temperature monitoring system of base on the ARM is use of embedded Linux as the operating system for the processor to S5PV210 development board designed for an embedded temperature monitoring system.Based on the analysis of the basic working principle based on the Linux device drivers discussed development often encounter interrupt handling, congestion handling, I / O ports, and on this basis to achieve the embedded processor based on open S5PV210 reading, writing, characters off the external RAM device driver and network drives.Combined with precision temperature sensor DS18B20, to achieve the correct temperature acquisition, and upload the data via Ethernet to a PC client.At first,the paper introduces the characteristics of various devices in a communication network, the bus structure and transmission technology, and according to the principle of serial communication between SCM and PC, with ubantu complete temperature monitoring system software design and implementation, to provide users with a friendly man-machine interface, the monitoring system to control and display the data after collection.The system also enables communication via a plurality of multi-threaded client and the server.Key words:S5PV210;embedded Linux operating system;DS18B20;Network programming

II

1.引言.......................................................................................................................................1 1.1 设计背景及意义...............................................................................................................1 1.2 设计的主要内容................................................................................................................2 2.相关技术...............................................................................................................................2 2.1 嵌入式Linux......................................................................................................................2 2.2 S5PV210.............................................................................................................................3 2.3 socket网络编程..............................................................................................................3 3.具体实现功能.......................................................................................................................4 3.1总体框架图........................................................................................................................4 3.2客户端功能........................................................................................................................5 3.3 服务器功能........................................................................................................................5 3.4 实验板输出信息...............................................................................................................5 4.具体实现过程.......................................................................................................................6 4.1 交叉编译工具的安装.......................................................................................................6 4.2 客户端模块的设计...........................................................................................................7 4.3 服务器模块.......................................................................................................................8 4.4 LED点亮模块..................................................................................................................10 4.5 温度感应模块.................................................................................................................11 5.测试结果分析.....................................................................................................................13 5.1 各模块运行的效果.........................................................................................................13 5.2 可扩展功能......................................................................................................................15 6.总结与展望.......................................................................................................................15

第一章 引言

1.1 设计背景及意义

温度作为工业、农业、国防和科研等部门最普遍的测量项目。它在工农业生产、现代科学研究以及高新技术开发过程中也是一个极其普遍而又非

常重要的参数。因此，在这些领域中，对于这个参数的测量与控制就显得尤为重要，特别是在纺织工业、冶金、化工、食品、温室种植，汽车制造以及气象预报和科研实验室等许多地方，都具有举足轻重的作用。

以往这些工作大多是由人工完成，不但工作量大，记录的数据少，对温度的调节缺乏实时性，而且电路复杂，标定和校准也比较麻烦，难以满足现代温度测量的要求[1]。自从传感器技术、微控制器技术和计算机技术日渐成熟之后，现代的温度测量与控制系统克服了以往系统中存在的一些问题，比如对环境温度的控制与调节以及数据的记录都由微控制器或计算机自动完成，人们的工作量大大地降低，而且测得的数据也更加的精确，对环境温度的调节更具有实时性[2]。

1.2 设计的主要内容

此次主要采用嵌入式Linux作为操作系统，针对以S5PV210为处理器的开发板设计的一个嵌入式温度监测系统。在Linux设备驱动程序的基本工作原理基础上，讨论了开发中经常会碰到的中断处理、拥塞处理、I/O端口，并在此基础上实现了基于S5PV210嵌入式处理器的开、读、写、关外部RAM的字符设备驱动和网络驱动。结合高精度温度传感器DS18B20，实现温度的正确采集，并通过以太网络将数据上传给上位机客户端。

温度监测系统根据Linux中的网络通信技术和串口通信技术来传输数据。用ubantu完成温度监测系统的软件设计与实现，为用户提供一个友好的人机界面，对监测系统进行控制并显示采集后的数据。本系统还通过多线程实现了多个客户端与服务器的通信。

第二章 相关技术

2.1 嵌入式Linux Linux是UNIX系统的一套免费使用和自由传播的类Unix操作系统，是一个基于POSIX和UNIX的多用户、多任务、支持多线程和多CPU的操作系统。它能运行主要的UNIX工具软件、应用程序和网络协议。支持32位和64位硬件。Linux继承了Unix以网络为核心的设计思想，是一个性能稳定的多用户网络操作系统。它诞生与1991年的10月5日。以后借助与Internet

网，并进过全世界各地计算机爱好者的共同努力下，现已成为世界上使用最多的一种UNIX类操作系统，并且使用人数还在迅猛增涨。

本次设计采用Linux作为嵌入式操作系统的原因有以下几点：[3] 1)低成本开发系统： 2)可应用于多种硬件平台 3)可定制的内核 4)性能优异 5)良好的网络支持

2.2 S5PV210

S5PV210又名“蜂鸟”（Hummingbird），是三星推出的一款适用于智能手机和平板电脑等多媒体设备的应用处理器。

S5PV210采用了ARM CortexTM-A8内核，ARM V7指令集，主频可达1GHZ，64/32位内部总线结构，32/32KB的数据/指令一级缓存，512KB的二级缓存，可以实现2000DMIPS（每秒运算20亿条指令集）的高性能运算能力。

包含很多强大的硬件编解码功能，内建MFC（Multi Format Codec），支持MPEG-1/2/4，H.263，H.264等格式视频的编解码，支持模拟/数字TV输出。JPEG硬件编解码，最大支持8000x8000分辨率

内建高性能PowerVR SGX540 3D图形引擎和2D图形引擎，支持2D/3D图形加速，是第五代PowerVR产品，其多边形生成率为2800万多边形/秒，像素填充率可达2.5亿/秒，在3D和多媒体方面比以往大幅提升，能够支持DX9，SM3.0，OpenGL2.0等PC级别显示技术。

具备IVA3硬件加速器，具备出色的图形解码性能，可以支持全高清、多标准的视频编码，流畅播放和录制30帧/秒的1920×1080像素（1080p）的视频文件，可以更快解码更高质量的图像和视频，同时，内建的HDMIv1.3，可以将高清视频输出到外部显示器上。

2.3 socket网络编程

Socket是进程通讯的一种方式，即调用这个网络库的一些API函数实现分布在不同主机的相关进程之间的数据交换。[4] 几个定义：

（1）IP地址：即依照TCP/IP协议分配给本地主机的网络地址，两个进程要通讯，任一进程首先要知道通讯对方的位置，即对方的IP。

（2）端口号：用来辨别本地通讯进程，一个本地的进程在通讯时均会占用一个端口号，不同的进程端口号不同，因此在通讯前必须要分配一个没有被访问的端口号。

（3）连接：指两个进程间的通讯链路。

（4）半相关：网络中用一个三元组可以在全局唯一标志一个进程：（协议，本地地址，本地端口号）

这样一个三元组，叫做一个半相关,它指定连接的每半部分。（4）全相关：一个完整的网间进程通信需要由两个进程组成，并且只能使用同一种高层协议。也就是说，不可能通信的一端用TCP协议，而另一端用UDP协议。因此一个完整的网间通信需要一个五元组来标识：（协议，本地地址，本地端口号，远地地址，远地端口号）

这样一个五元组，叫做一个相关（association），即两个协议相同的半相关才能组合成一个合适的相关，或完全指定组成一连接。

第三章 具体实现功能

3.1总体框架图

图1 项目总体框架图

3.2客户端功能

1)显示简单的用户界面 2)发送命令给服务器 3)接收服务器传输的数据 4)显示温度数据

3.3 服务器功能

1)接收客户端发送的命令 2)处理命令

3)把命令转发给硬件 4)获取硬件处理所得的数据 5)把该数据传给客户端

3.4 实验板输出信息

1)LED灯按照客户端的指令亮灭 2)蜂鸣器唱歌

3)核心板控制输出温度传感器的数据到服务器

第四章 具体实现过程

个应用系统要完成各项功能，首先必须有较完善的硬件作保证。同时还必须得到相应设计合理的软件的支持，尤其是微机应用高速发展的今天，许多由硬件完成的工作，都可通过软件编程而代替。甚至有些必须采用很复杂的硬件电路才能完成的工作，用软件编和有时会变得很简单。因此充分利用其内部丰富的硬件资源和软件资源。

程序设计语言有三种：机器语言、汇编语言、高级语言。本系统运用的是高级语言所编写，也就是C语言。所用到的开发平台为ubuntu系统。

4.1 交叉编译工具的安装

(1)考虑到现今Linux平台发展，交叉编译平台编统一为arm-linux-gcc-4.4.3(2)在windows系统下，建立一个共享目录，如：e:/linux\_file(3)将光盘目录linux中的arm-linux-gcc-4.4.3.tar.gz 复制到e:/linux\_file(4)注意在进行虚拟机设置时使共享目录的有效，并添加共享目录e:/linux\_file(5)进入linux操作系统，root目录下建立一个Armcode的子目录，将共享目录下的文件arm-linux-gcc-4.4.3.tar.gz复制到该目录(6)然后进入到该目录，执行解压命令：#cd /root/Armcode;#tar xvzf arm-linux-gcc-4.4.3.tgz –C /;注意：C 后面有个空格，并且C 是大写的，它是英文单词“Change”的第一个字母，在此是改变目录的意思。(7)执行

该

命

令，将

把

arm-linux-gcc

安

装

到/opt/FriendlyARM/toolschain /4.4.3/bin 目录。

(8)把编译器路径加入系统环境变量，运行命令:#gedit /root/.bashrc 编辑/root/.bashrc 文件，在最后一行添加：export PATH=$PATH: /opt/FriendlyARM/toolschain/4.4.3/bin(9)重新登录系统(不必重启机器，开始->logout 即可)，使以上设置生效，在命令行输入:arm-linux-gcc –v，会出现如下信息，这说明交叉编译环境已经成功安装。

4.2 客户端模块的设计

用户界面的显示：

void interface\_print(char \*temp){ system(“clear”);printf(“e[31m\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*2024梧\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*e[0mn”);printf(“e[31m\*

\*e[0mn”);printf(“e[31m\*e[0m e[32m点亮LED1: on1

on2e[0me[31m

\*e[0mn”);printf(“e[31m\*e[0m e[33m关闭LED1: off1

off2e[0me[31m

\*e[0mn”);printf(“e[31m\*e[0m e[34m开蜂鸣器: onb

songe[0me[31m

\*e[0mn”);printf(“e[31m\*e[0m e[35m获取温度: get

e[31m \*e[0mn”,temp);printf(“e[31m\*e[0m e[36m退

出: e[0me[31m

\*e[0mn”);printf(“e[31m\*

\*e[0mn”);printf(“e[31m\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*未

来\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*e[0mn”);}

发送命令给服务器：

/\*5.调用I/O函数(read/write)与客户端通讯。\*/ int i = 5;int pos;while(1){

ret = poll(pfd,2,-1);

if(ret > 0){

if(pfd[0].revents == POLLIN){

interface\_print(temp);

/\*从终端读取数据\*/

memset(buf,0,sizeof(buf));

ret = read(pfd[0].fd,buf,sizeof(buf)-1);7

州学院

|

点亮LED2:

|

关闭LED2:

|

播放音乐:

|

温度:[%s]e[0m quit

|

的大

神

们

} if(ret > 0){

}

/\*发送数据给服务器\*/ write(sockfd,buf,ret);

接受服务器传输的数据：

if(pfd[1].revents == POLLIN){

if(!strncmp(buf,“get”,3)){

/\*接收服务器发送的信息\*/

memset(temp,0,sizeof(temp));

ret = read(pfd[1].fd,temp,sizeof(temp)-1);

if(ret > 0){

temp[ret-1] = \'\';

/\*把信息显示到终端\*/

interface\_print(temp);

}

} } 4.3 服务器模块

接收客户端发送的命令：

/\*读取客户端信息\*/

memset(buf,0,sizeof(buf));

ret = read(newfd,buf,sizeof(buf)-1);

if(ret > 0){

把命令转发给硬件：

/\*发送命令给硬件\*/

if(!strncasecmp(buf,“on1”,3)){

cmd = LED\_ON;

val = 3;

}else if(!strncasecmp(buf,“on2”,3)){

cmd = LED\_ON;

val = 4;

}else if(!strncasecmp(buf,“off1”,4)){

cmd = LED\_OFF;

val = 3;

}else if(!strncasecmp(buf,“off2”,4)){

cmd = LED\_OFF;

}

val = 4;

}else if(!strncasecmp(buf,“onb”,3)){

pwm\_on();

}else if(!strncasecmp(buf,“song”,4)){

pwm\_music\_fun();

}

if(ioctl(fd\_led,cmd,(unsigned long)val)led\_class = class\_create(THIS\_MODULE, LED\_MODULE);if(IS\_ERR(led\_dev->led\_class)){ printk(“class create failedn”);errno = PTR\_ERR(led\_dev->led\_class);goto err2;} /\*创建一个设备文件，之后系统会自动在/dev目录下自动创建一个设备文件\*/ led\_dev->led\_device = device\_create(led\_dev->led\_class,NULL,MKDEV(LED\_MAJOR,0),NULL,“led”);if(IS\_ERR(led\_dev->led\_device)){

printk(“class create failedn”);

errno = PTR\_ERR(led\_dev->led\_device);

goto err3;} led\_init();

return 0;

4.5 温度感应模块

static void WriteOneChar(unsigned char dat){ unsigned char i=0;

writel((readl(gph1con)& 0xFFFFFFF0)| 0x1, gph1con);//gph1\_0,output spin\_lock(&lock);for(i=0;i>=1;} spin\_unlock(&lock);}

static unsigned int ReadTemp(void){ unsigned char T\_h=0;unsigned char T\_l=0;unsigned int temp=0;

writel((readl(gph1pud)& 0xFFFc)| 0x02, gph1pud);//gph1\_0,pull-up enabled

spin\_lock(&lock);Init\_DS18B2O();spin\_unlock(&lock);\_\_udelay(400);writel((readl(gph1con)& 0xFFFFFFF0)| 0x1, gph1con);//gph1\_0,output writel(readl(gph1dat)| 0x1 , gph1dat);WriteOneChar(0xcc);WriteOneChar(0x44);mdelay(100);spin\_lock(&lock);Init\_DS18B2O();spin\_unlock(&lock);\_\_udelay(400);writel((readl(gph1con)& 0xFFFFFFF0)| 0x1, gph1con);//gph1\_0,output writel(readl(gph1dat)| 0x1 , gph1dat);WriteOneChar(0xcc);WriteOneChar(0xBE);T\_l=ReadOneChar();//L T\_h=ReadOneChar();//H

temp=(unsigned int)(((unsigned int)T\_h load flash kernel x 烧写更新内核，传送zImage文件;等待传送内核文件

传送内核：

vivi>load flash root j 烧写更新文件系统;烧写新的文件系统 load flash root j

(3)EmbestIDE Pro for ARM: 应用于嵌入式软件开发的新一代集成开发环境，是一个高度集成的图形界面操作环境，包含编辑器、编译汇编链接器、调试器、工程管理、Flash 编程等工具；支持的开发语言包括标准C和汇编语言。（4）cygwin: 一个在windows平台上运行的unix模拟环境，它对于学习unix/linux操作环境，或者从unix到windows的应用程序移植，或者进行某些特殊的开发工作，尤其是使用gnu工具集在windows上进行嵌 5

入式系统开发，把gcc，gdb，gas等开发工具进行了改进，能够生成并解释win32的目标文件。

2、硬件

S3C2410处理器是Samsung公司基于ARM公司的ARM920T处理器核，32位微控制器。该处理器拥有：独立的16KB指令Cache和16KB数据Cache，MMU，支持TFT的LCD控制器，NAND闪存控制器，3路UART，4路DMA，4路带PWM的Timer，I/O口，RTC，8路10位ADC，Touch Screen接口，IIC-BUS 接口，IIS-BUS 接口，2个USB主机，1个USB设备，SD主机和MMC接口，2路SPI。S3C2410处理器最高可运行在203MHz。

2.3、设计所需工具

1．软件： Embest Online Flash Programmer For ARM，HYPER TERMINAL（超级终端），EmbestIDE Pro for ARM，cygwin 1.硬件：s3c2410开发板，Embest实验箱

2.4、平台构建过程

2.4.1、硬件平台搭建

硬件流程图：

(1)Vivi烧写过程

1)首先把SW104断开，Flash Programmer的Program，在File选择Open打开要烧写的配置文件S3C2410&NandFLash\_vivi.cfg，在Flash Programmer的Program页中选择要烧写的文件vivi.bon&load.bin。点击按钮 Progarm 开始烧写，直到烧写成功

2)连接串口线到 PC 机 COM1，运行光盘中提供的 Windows 超级终端Hyper Terminal.ht 把开发板重新加电，程序运行后，在超级终端上可以看到串口输出Wating，表示正在等待用户从超级终端下载文件。这时，请点击超级终端菜单“传送”选择 Xmodem 方式下载 vivi.nand 文件，点击 OK 后等待下载烧写结束即可。(2)内核zImage烧写

1)首先SW104设为短接（从Nand Flash启动），并确定已经烧写vivi.nand，加电。)在vivi启动等待中，敲入空格键进入vivi界面环境，并输入以下命令：vivi> load flash kernel x 烧写更新内核约1分钟即可烧写完毕 3)点击超级终端菜单中的“传送”，选“发送文件”zImage” 并选择xModem方式传送）烧写结束，重起实验板，观测超级终端窗口提示信息就可以启动linux内核，(3)新文件系统的烧写

1）首先SW104设为短接（从Nand Flash启动），确定已经成功烧写vivi.nand，加电运行可以看到vivi启动信息，输入空格进入命令状态；

2）双击运行Download.pjf（该文件在/tmp/edukit-2410/image/中）工程（将启动Embest IDE环境），点击连接Remote connect，程序应该正在运行（命令按钮STOP为红色）；在串口输入help，看看有没有反应，如果没反应，点击IDE 按钮：Reset->Start(F5)；再输入help测试，直到有反应为止；

3）如果可以输出一些信息，再点击IDE中的Stop，配置Debug的Download地址为0x30000000，并点击IDE菜单Project选择Settings项，在Download页下拉Category到Download项，在Download File选择root.cramfs文件，点击确定后：

点击IDE菜单DEBUG选择Download下载文件系统映象约1分钟

下载完毕后，点击Start(F5)然后在超级终端里输入： load flash root j(烧写更新文件系统)约1分钟即可烧写完毕

注意：只能在“vivi的烧写”操作完成后，才可以按以上方法正确烧写root映象到Nand Flash。

重起实验板，观测超级终端窗口提示信息，引导整个系统启动到linux行命令输入状态。

2.4.2根文件系统的制作（1）根文件系统

根文件系统是Linux系统的核心部分，包含系统使用的软件和库，以及所有用来为用户提供支持架构和用户使用的应用软件，并作为储存数据读写结果的区域。在Linux系统启动时，首先完成内核安装及环境初始化，最后会寻找一个文件系统作为根文件系统被加载。Linux系统中使用“/”来唯一表示根文件系统的安装路径。嵌入式系统中通常可以悬着的根文件系统有：Romfs、CRAMFS、RAMFS、JFFS2、EXT2等，甚至还可以使用NFS作为根文件系统。

（2）cramfs文件系统

Cramfs是Linux创始人Linux torvalds开发的一个适用于嵌入式系统的小文件系统。Cramfs是一个只读文件系统，采用zlib压缩，压缩比一般可以达到1：2，但仍可以做到高效的随机读取。Linux系统中，通常把需要修改的目录压缩存放，并在系统引导的时候再将压缩文件解开。因为cramfs不会影响系统读取文件的速度，而且是一个高度压缩的文件系统，因此非常广泛应用于嵌入式系统中。

（3）cygwin简介

Cygwin是一个在windows平台上运行的unix/Linux模拟环境，是cygnus solutions公司开发的自由软件。Cygwin中，“/”表示根目录，即cygwin的安 8

装目录。我们常用的set\_env\_linux.sh中定义的目录有：

SOURCEDIR：/tmp/edukit-2410存储了vivi、linux、fs等源代码和例程 WORKDIR：/usr/local/src/edukit-2410工作区。

一般情况下都要把已经规划好的目录结构转换成一个映象文件，即使用命令工具 mkcramfs（cygwin下为 mkcramfs.exe），把相应的 cramfs 目录树压缩为单一的映象文件。其命令格式为：

mkcramfs [-h] [-e edition] [-i file] [-n name] dirname outfile 可以使用我们提供的 mkcramfs.exe 在 cygwin 下编译生成文件系统映象文件 root.cramfs，再固化到开发系统 FLASH 上运行。

（4）常用的Linux行命令

1)、cd 改变当前目录（文件夹）。例如下，cd/ 返回到根目录 cd..退回到上级目录

cd/tmp/edukit-2410/进入/tmp/edukit-2410/文件夹 2)、ls 列出当前目录中的内容。Ls 简单格式列表 ls–l 使用详细格式列表。3)、pwd 显示当前所在的目录。

（5）tar工具命令

tar 程序用于储存或展开 tar 存档文件。命令格式：

tar [-参数] ［文件名］［路径］-x ：extract |--get 从存档展开文件-v ：--verbose 详细显示处理的文件-j ：--有 bz2 属性的必须包含

-f ：--file [HOSTNAME:]F 指定存档或设备（缺省为 /dev/rmt0）

（6）解压原文件系统（命令+解压目录的存放）

1）先将 root.cramfs.tar.bz2文件放在C:cygwin目录中

2）解压文件系统

运行cygwin，执行以下命令解压安装：

$> source /tmp/edukit-2410/set\_env\_linux.sh Linux编译环境变量设置 $> cd /

$> tar-xvjf root.cramfs.tar.bz2 $> ls „ root „

root文件夹中就是我们想要的cramfs文件系统 3)如果在根目录中产生root文件夹，解压成功 4）在root目录中新建xx文件夹，用于存放应用程序

进入该目录后执行以下命令编译链接测试程序： $> cd root $>mkdir xx

（7）编译应用程序 ts.c(命令+生成文件格式+存放位置): 将编写好的ts.c程序放在C:cygwin目录中 进入该目录后执行以下命令编译链接测试程序： $> cd / $> arm-linux-gcc-o ts ts.c（也可以编写Makefile来编译）

生成文件： ts 如下图所示

将ts文件放入root 下的xx文件夹中

（8）新文件系统的制作: 把刚才编译输出的ts文件拷贝到文件系统所在的工作目录root目录下，执行以下命令生成新的文件系统映象： $> cd /

$> mkcramfs root root.new

刚刚编译生成的文件系统映象 root.new 中已经包含测试程序即生成文件。解压文件系统

解压成功如下

在root目录中新建xx文件夹，用于存放应用程序

将编写好的ts.c程序放在C:cygwin目录中

生成文件： ts 如下图所示

新文件系统的制作

生成文件：

第三章 程序

3.1.程序流程图：

3.2.分析驱动

触摸屏驱动在/kernel/drivers/char/s3c2410-ts.c 文件中。

3.2.1、触摸屏设备驱动中数据结构

（1）触摸屏的file\_operations static struct file\_operations s3c2410\_fops={ owner: THIS\_MODULE, open: s3c2410\_ts\_open, read: s3c2410\_ts\_read，release: s3c2410\_ts\_release, #ifdef USE\_ASYNC fasync: s3c2410\_ts\_fasync,//异步通知

#endif poll: s3c2410\_ts\_poll,//轮询 };（2）触摸屏设备结构体的成员与按键设备结构体的成员类似，也包含一个缓冲区，同时包括自旋锁、等待队列和fasync\_struct指针 typedef struct { unsigned int penStatus;/\* PEN\_UP, PEN\_DOWN, PEN\_SAMPLE \*/ TS\_RET buf[MAX\_TS\_BUF];/\* protect against overrun（环形缓冲区）\*/ unsigned int head, tail;/\* head and tail for queued events（环形缓冲区的头尾）\*/ wait\_queue\_head\_t wq;//\* 等待队列数据结构 spinlock\_t lock;//\* 自旋锁

#ifdef USE\_ASYNC struct fasync\_struct \*aq;#endif #ifdef CONFIG\_PM struct pm\_dev \*pm\_dev;//友善之臂专有的，我后面的代码删除了这段 #endif } TS\_DEV;

（3）触摸屏结构体中包含的TS\_RET值的类型定义，包含X、Y坐标和状态（PEN\_DOWN、PEN\_UP）等信息，这个信息会在用户读取触摸信息时复制到用户空 间

typedef struct { 14

unsigned short pressure;//\* 压力，这里可定义为笔按下，笔抬起，笔拖曳

unsigned short x;//\* 横坐标的采样值 unsigned short y;//\* 纵坐标的采样值 unsigned short pad;//\* 填充位 } TS\_RET;

（4）在触摸屏设备驱动中，将实现open()、release()、read()、fasync()和poll()函数，因此，其文件操作结构体定义

触摸屏驱动文件操作结构体：static struct file\_operations s3c2410\_fops={} 3.2.2、触摸屏驱动模块加载和卸载函数

（1）在触摸屏设备驱动的模块加载函数中，要完成申请设备号、添加cdev、申请中断、设置触摸屏控制引脚（YPON、YMON、XPON、XMON）等多项工作 触摸屏设备驱动的模块加载函数

static int \_\_init s3c2410\_ts\_init(void)触摸屏设备驱动模块卸载函数

static void \_\_exit s3c2410\_ts\_exit(void)（2）可知触摸屏驱动中会产生两类中断，一类是触点中断（INT-TC），一类是X/Y位置转换中断（INT-ADC）。在前一类中断发生后，若之前处于PEN\_UP状态，则应该启动X/Y位置转换。另外，将抬起中断也放在INT-TC处理程序中，它会调用tsEvent()完成等待队列和信号的释放 触摸屏设备驱动的触点/抬起中断处理程序

static void s3c2410\_isr\_tc(int irq, void \*dev\_id, struct pt\_regs \*reg)

当X/Y位置转换中断发生后，应读取X、Y的坐标值，填入缓冲区 触摸屏设备驱动X/Y位置转换中断处理程序

static void s3c2410\_isr\_adc(int irq, void \*dev\_id, struct pt\_regs \*reg)触摸屏设备驱动中获得X、Y坐标

static inline void s3c2410\_get\_XY(void)（3）tsEvent最终为tsEvent\_raw()，这个函数很关键，当处于PEN\_DOWN状态时调用该函数，它会完成缓冲区的填充、等待队列的唤醒以及异步通知信号的释放；否则（处于PEN\_UP状态），将缓冲区头清0，也唤醒等待队列并释放信号

触摸屏设备驱动的tsEvent\_raw()函数 static void tsEvent\_raw(void)（4）在包含了对拖动轨迹支持的情况下，定时器会被启用，周期为10ms，在每次定时器处理函数被引发时，调用start\_ts\_adc()开始X/Y位置转换过程

触摸屏设备驱动的定时器处理函数

static void ts\_timer\_handler(unsigned long data)（5）在触摸屏设备驱动的打开函数中，应初始化缓冲区、penStatus和定期器、等待队列及tsEvent时间处理函数指针 触摸屏设备驱动的打开函数

static int s3c2410\_ts\_open(struct inode \*inode, struct file \*filp)16

（6）触摸屏设备驱动的释放函数非常简单，删除为用于拖动轨迹所使用的定时器即可

触摸屏设备驱动的释放函数

static int s3c2410\_ts\_release(struct inode \*inode, struct file \*filp)3.2.3、触摸屏设备驱动的读函数

触摸屏设备驱动的读函数实现缓冲区中信息向用户空间的复制，当缓冲区有内容时，直接复制；否则，如果用户阻塞访问触摸屏，则进程在等待队列上睡眠，否则，立即返回-EAGAIN 触摸屏设备驱动的读函数

static ssize\_t s3c2410\_ts\_read(struct file \*filp, char \*buffer, size\_t count, loff\_t \*ppos)3.2.4、触摸屏设备驱动的轮询与异步通知

在触摸屏设备驱动中，通过s3c2410\_ts\_poll()函数实现了轮询接口，这个函数的实现非常简单。它将等待队列添加到poll\_table，当缓冲区有数据时，返回资源可读取标志，否则返回0 触摸屏设备驱动的poll()函数

static unsigned int s3c2410\_ts\_poll(struct file \*filp, struct poll\_table\_struct \*wait)而为了实现触摸屏设备驱动对应用程序的异步通知，设备驱动中要实现s3c2410\_ts\_fasync()函数 触摸屏设备驱动的fasync()函数

static int s3c2410\_ts\_fasync(int fd, struct file \*filp, int mode)3.2.5源程序触摸屏驱动代码：

/\* \* s3c2410-ts.c \* \* touchScreen driver for SAMSUNG S3C2410 \* \* Author: Janghoon Lyu \* Date : $Date: 2024/06/04 07:11:00 $ \* \* $Revision: 1.1.2.6 $ \* \* Based on pt036001b-ts.c \* \* This file is subject to the terms and conditions of the GNU General Public \* License.See the file COPYING in the main directory of this archive \* for more details.\* \* History: \* \* 2024-05-27: Janghoon Lyu \*PM 内靛啊 甸绢啊 乐变 窍瘤父 抛胶飘 登瘤 臼疽澜.\* \*/

#include #include #include #include

#include #include #include #include #include #include #include

#include

#ifdef CONFIG\_PM #include #endif

/\* debug macros \*/ #undef DEBUG #ifdef DEBUG #define DPRINTK(x...)printk(“s3c2410-ts: ” ##x)#else #define DPRINTK(x...)#endif

#define PEN\_UP 0

#define PEN\_DOWN 1 #define PEN\_FLEETING 2 #define MAX\_TS\_BUF 16 /\* how many do we want to buffer \*/

#undef USE\_ASYNC 1 #define DEVICE\_NAME “s3c2410-ts” #define TSRAW\_MINOR 1

typedef struct { unsigned int penStatus;/\* PEN\_UP, PEN\_DOWN, PEN\_SAMPLE \*/ TS\_RET buf[MAX\_TS\_BUF];/\* protect against overrun \*/ unsigned int head, tail;/\* head and tail for queued events \*/ wait\_queue\_head\_t wq;spinlock\_t lock;#ifdef USE\_ASYNC struct fasync\_struct \*aq;#endif #ifdef CONFIG\_PM struct pm\_dev \*pm\_dev;#endif } TS\_DEV;

static TS\_DEV tsdev;

#define BUF\_HEAD(tsdev.buf[tsdev.head])#define BUF\_TAIL(tsdev.buf[tsdev.tail])#define INCBUF(x,mod)((++(x))&((mod)-1))

static int tsMajor = 0;

static void(\*tsEvent)(void);

#define HOOK\_FOR\_DRAG #ifdef HOOK\_FOR\_DRAG #define TS\_TIMER\_DELAY(HZ/100)/\* 10 ms \*/ static struct timer\_list ts\_timer;#endif

#define wait\_down\_int(){ ADCTSC = DOWN\_INT | XP\_PULL\_UP\_EN |

XP\_AIN | XM\_HIZ | YP\_AIN | YM\_GND |

XP\_PST(WAIT\_INT\_MODE);} #define wait\_up\_int(){ ADCTSC = UP\_INT | XP\_PULL\_UP\_EN | XP\_AIN | XM\_HIZ |

YP\_AIN | YM\_GND | XP\_PST(WAIT\_INT\_MODE);} #define mode\_x\_axis(){ ADCTSC = XP\_EXTVLT | XM\_GND | YP\_AIN | YM\_HIZ |

XP\_PULL\_UP\_DIS | XP\_PST(X\_AXIS\_MODE);} #define mode\_x\_axis\_n(){ ADCTSC = XP\_EXTVLT | XM\_GND | YP\_AIN | YM\_HIZ |

XP\_PULL\_UP\_DIS | XP\_PST(NOP\_MODE);} #define mode\_y\_axis(){ ADCTSC = XP\_AIN | XM\_HIZ | YP\_EXTVLT | YM\_GND |

XP\_PULL\_UP\_DIS | XP\_PST(Y\_AXIS\_MODE);} #define start\_adc\_x(){ ADCCON = PRESCALE\_EN | PRSCVL(49)|

ADC\_INPUT(ADC\_IN5)| ADC\_START\_BY\_RD\_EN |

ADC\_NORMAL\_MODE;

ADCDAT0;} #define start\_adc\_y(){ ADCCON = PRESCALE\_EN | PRSCVL(49)|

ADC\_INPUT(ADC\_IN7)| ADC\_START\_BY\_RD\_EN |

ADC\_NORMAL\_MODE;

ADCDAT1;} #define disable\_ts\_adc(){ ADCCON &= ~(ADCCON\_READ\_START);}

static int adc\_state = 0;static int x, y;/\* touch screen coorinates \*/

static void tsEvent\_raw(void){ if(tsdev.penStatus == PEN\_DOWN){

BUF\_HEAD.x = x;

BUF\_HEAD.y = y;

BUF\_HEAD.pressure = PEN\_DOWN;

#ifdef HOOK\_FOR\_DRAG

ts\_timer.expires = jiffies + TS\_TIMER\_DELAY;

add\_timer(&ts\_timer);#endif } else { #ifdef HOOK\_FOR\_DRAG

del\_timer(&ts\_timer);#endif

BUF\_HEAD.x = 0;

BUF\_HEAD.y = 0;

BUF\_HEAD.pressure = PEN\_UP;}

tsdev.head = INCBUF(tsdev.head, MAX\_TS\_BUF);wake\_up\_interruptible(&(tsdev.wq));

#ifdef USE\_ASYNC if(tsdev.aq)

kill\_fasync(&(tsdev.aq), SIGIO, POLL\_IN);#endif

#ifdef CONFIG\_PM pm\_access(tsdev.pm\_dev);#endif }

static int tsRead(TS\_RET \* ts\_ret){ spin\_lock\_irq(&(tsdev.lock));ts\_ret->x = BUF\_TAIL.x;ts\_ret->y = BUF\_TAIL.y;ts\_ret->pressure = BUF\_TAIL.pressure;tsdev.tail = INCBUF(tsdev.tail, MAX\_TS\_BUF);spin\_unlock\_irq(&(tsdev.lock));

return sizeof(TS\_RET);}

static ssize\_t s3c2410\_ts\_read(struct file \*filp, char \*buffer, size\_t count, loff\_t \*ppos){ TS\_RET ts\_ret;

retry: if(tsdev.head!= tsdev.tail){

int count;

count = tsRead(&ts\_ret);

if(count)copy\_to\_user(buffer,(char \*)&ts\_ret, count);

return count;} else {

if(filp->f\_flags & O\_NONBLOCK)

return-EAGAIN;

interruptible\_sleep\_on(&(tsdev.wq));

if(signal\_pending(current))

return-ERESTARTSYS;

goto retry;}

return sizeof(TS\_RET);}

#ifdef USE\_ASYNC static int s3c2410\_ts\_fasync(int fd, struct file \*filp, int mode){ return fasync\_helper(fd, filp, mode, &(tsdev.aq));} #endif

static unsigned int s3c2410\_ts\_poll(struct file \*filp, struct poll\_table\_struct \*wait){ poll\_wait(filp, &(tsdev.wq), wait);return(tsdev.head == tsdev.tail)? 0 :(POLLIN | POLLRDNORM);}

static inline void start\_ts\_adc(void){ adc\_state = 0;mode\_x\_axis();start\_adc\_x();}

static inline void s3c2410\_get\_XY(void){ if(adc\_state == 0){

adc\_state = 1;

disable\_ts\_adc();

y =(ADCDAT0 & 0x3ff);

mode\_y\_axis();

start\_adc\_y();} else if(adc\_state == 1){

adc\_state = 0;

disable\_ts\_adc();

x =(ADCDAT1 & 0x3ff);

tsdev.penStatus = PEN\_DOWN;

DPRINTK(“PEN DOWN: x: %08d, y: %08dn”, x, y);

wait\_up\_int();

tsEvent();} }

static void s3c2410\_isr\_adc(int irq, void \*dev\_id, struct pt\_regs \*reg){ #if 0 DPRINTK(“Occured Touch Screen Interruptn”);DPRINTK(“SUBSRCPND = 0x%08lxn”, SUBSRCPND);#endif spin\_lock\_irq(&(tsdev.lock));if(tsdev.penStatus == PEN\_UP)s3c2410\_get\_XY();#ifdef HOOK\_FOR\_DRAG else s3c2410\_get\_XY();#endif spin\_unlock\_irq(&(tsdev.lock));}

static void s3c2410\_isr\_tc(int irq, void \*dev\_id, struct pt\_regs \*reg){ #if 0 DPRINTK(“Occured Touch Screen Interruptn”);DPRINTK(“SUBSRCPND = 0x%08lxn”, SUBSRCPND);#endif spin\_lock\_irq(&(tsdev.lock));if(tsdev.penStatus == PEN\_UP){ start\_ts\_adc();} else { tsdev.penStatus = PEN\_UP;DPRINTK(“PEN UP: x: %08d, y: %08dn”, x, y);wait\_down\_int();tsEvent();

} spin\_unlock\_irq(&(tsdev.lock));}

#ifdef HOOK\_FOR\_DRAG static void ts\_timer\_handler(unsigned long data){ spin\_lock\_irq(&(tsdev.lock));if(tsdev.penStatus == PEN\_DOWN){

start\_ts\_adc();} spin\_unlock\_irq(&(tsdev.lock));} #endif

static int s3c2410\_ts\_open(struct inode \*inode, struct file \*filp){ tsdev.head = tsdev.tail = 0;tsdev.penStatus = PEN\_UP;#ifdef HOOK\_FOR\_DRAG init\_timer(&ts\_timer);ts\_timer.function = ts\_timer\_handler;#endif tsEvent = tsEvent\_raw;init\_waitqueue\_head(&(tsdev.wq));

MOD\_INC\_USE\_COUNT;return 0;}

static int s3c2410\_ts\_release(struct inode \*inode, struct file \*filp){ #ifdef HOOK\_FOR\_DRAG del\_timer(&ts\_timer);#endif MOD\_DEC\_USE\_COUNT;return 0;}

static struct file\_operations s3c2410\_fops = { owner: THIS\_MODULE, open: s3c2410\_ts\_open, read: s3c2410\_ts\_read，release: s3c2410\_ts\_release，#ifdef USE\_ASYNC fasync: s3c2410\_ts\_fasync, #endif poll: s3c2410\_ts\_poll, };

void tsEvent\_dummy(void){} #ifdef CONFIG\_PM static int s3c2410\_ts\_pm\_callback(struct pm\_dev \*pm\_dev, pm\_request\_t req，void \*data){ switch(req){

case PM\_SUSPEND:

tsEvent = tsEvent\_dummy;

break;

case PM\_RESUME:

tsEvent = tsEvent\_raw;

wait\_down\_int();

break;} return 0;} #endif

#ifdef CONFIG\_DEVFS\_FS static devfs\_handle\_t devfs\_ts\_dir, devfs\_tsraw;#endif static int \_\_init s3c2410\_ts\_init(void){ int ret;

tsEvent = tsEvent\_dummy;

ret = register\_chrdev(0, DEVICE\_NAME, &s3c2410\_fops);if(ret #include #include /\* 文件操作 \*/

#define PEN\_UP 0 /\* 触摸笔抬笔，即触摸屏不被压下 \*/ #define PEN\_DOWN 1 /\* 触摸笔下笔，即触摸屏被压下 \*/ #define PEN\_FLEETING 2 /\* 触摸笔拖动 \*/

typedef struct { unsigned short pressure;/\* 触摸笔动作 \*/ unsigned short x;

/\* 触点x座标值 \*/ unsigned short y;

/\* 触点y座标值 \*/ unsigned short pad;}TS\_RET;

int main(){ int fd,ret,i;

unsigned char suba;TS\_RET tsret;

fd = open(“/dev/touchscreen/0raw”, O\_RDWR);/\* 打开设备 \*/ if(fd ==-1){ printf(“nCan\'t open I2C device!n”);exit(-1);}

while(1){

ret = read(fd,(char \*)&tsret, sizeof(TS\_RET));

if(ret!= sizeof(TS\_RET))

{

printf(“read touch screen error!”);

close(fd);

exit(-1);

}

else

{

printf(“pressure is: %dn”, tsret.pressure);

printf(“x is: %dn”, tsret.x);

printf(“y is: %dn”, tsret.y);

} }

close(fd);return 0;} 3.3.应用程序的调试

使用s3c2410\_ts.c触摸屏驱动编写应用程序，读取触摸屏的触点坐标值及动作信息（触点x坐标值，y坐标及是否有压力值press），并在串口中断打印出来

对触摸屏设别的操作有打开设备，关闭设备，读操作等。编写应用程序读取触摸屏的触点坐标值及动作信息时，只需利用触摸屏驱动程序便可实现，先打开触摸屏设备，然后调用读函数即可。

其中，触摸笔动作取值如下： #define PEN\_UP 0 #define PEN\_DOWN

/\* 触摸笔抬笔，即触摸屏不被压下 \*/ /\* 触摸笔下笔，即触摸屏被压下 \*/

#define PEN\_FLEETING 2 结构体定义如下： typedef struct { unsigned short pressure;unsigned short x;unsigned short y;unsigned short pad;}TS\_RET 打开应用程序：

/\* 触摸笔拖动 \*/

/\* 触摸笔动作 \*/ /\* 触点x座标值 \*/ /\* 触点y座标值 \*/

3.2.6、实验结果显示：

第四部分 感想

第四部分 心得

4.1 课程设计心得体会：

为期几天的课程设计结束了，再次期间我积极亲自实验，用的目标板是s3c2410核心子板，用JTAG仿真器，用Cygwin模拟软件来学习触摸板的设计。我学会了很多，学会了很多。

首先我扪主要了解整个设计过程，以及实验环境的建立，这次用的是交叉编译环境，通过这次课设我更清楚搭建嵌入式系统的开发平台，我们用的目标板是s3c2410核心子板，用JTAG仿真器，用Cygwin模拟软件，课设的这几天我学会了熟练的使用Cygwin软件，掌握了一些常用的命令，加上研究生学长给我们的指导，知道了如何学习，如何思考，知道了运linux操作系统开发嵌入式与wince操作系统开发嵌入式的区别。

其次是学会vivi,内核，根文件系统的编译与移植（烧写），通过这个过程我熟悉了怎么把软件固化到硬件上，知道了软件怎么控制硬件，这个步骤很重要，要烧写不成功，目标板系统就运行不起来，实验就失败了，这个过程我们练习了好多变呢，大家都很累哦！

再次我们就开始写我们的应用程序，通过以上步骤实验系统搭建好了，只要调试好应用程序，然后再运行成功就行了，为此我又把课本上讲得触摸屏原理那章认真看了，又看了实验指导书，查资料，上网搜索，终于编出应用程序，经过不断的调试才编译成功，这个过程太辛苦了，加上实验板不太好，真是对我们的挑战，不过看到运行的 30

结果，大家都很高兴，也很有成就感啊！还看懂了一些s3c2410的驱动程序的源代码，了解了s3c2410一些控制器的使用，以及s3c2410A的一些接口原理与应用。

我明白了只有不断的努力，不断的学习，才能在将来遇到的问题中能够游刃有余，才能够不会捉襟见肘。

第五部分 参考文献

5.1【参考文献】 程昌南，方强等．《ARM Linux入门与实践 》【M】．北京：北京航空航天大学出版社，2024.10 2 张晓林等．《嵌入式系统设计与实践》【M】．北京：北京航空航天大学出版社，2024．1 3 李俊等．《嵌入式Linux设备驱动开发详解》【M】．北京：北京人民邮电出版社，2024．3 4 黄智伟，邓月明，王彦．《ARM9嵌入式系统设计基础教程》．北京：北京航空航天大学出版社，2024．8 5 [美]Wayne Wolf.嵌入式计算系统设计原理.孙玉芳, 梁彬 罗保国 等译.北京: 机械工业出版社, 2024 李剑, 赵鹏程, 汤建彬.32位ARM嵌入式处理器的调试技术.电子技术应用, 2024,(3)钟汉如, 王创生.嵌入式Linux的中断处理与实时调度的实现机制.计算机工程, 2024, 28(10)Arnold Berger.嵌入式系统设计.吕骏 译.北京: 电子工业出版社, 2024

本DOCX文档由 www.zciku.com/中词库网 生成，海量范文文档任你选，，为你的工作锦上添花,祝你一臂之力！