摘要

嵌入式系统是将先进的计算机技术、半导体技术和电子技术和各个行业的具体应用相结合后的产物。嵌入式系统已经成为当前 IT 产业的焦点之一，但同时大量的嵌入式应用也对嵌入式系统的性能和功能提出了更高的要求。随着嵌入式技术的发展，8 位、16 位单片机已经越来越不能满足应用的需要。而集成电路技术的发展使得 32 位微控制器的价格已经不比 8 位机高多少，并且基于 32 位 RISC 处理器的嵌入式系统更加受到用户的青睐。

税控收款机是带计税功能的收款机，广泛应用于国税系统大、中、小型商品零售行业，地税系统餐饮业、服务业、娱乐业等各个行业。税控收款机是国家金税工程带动的新兴产业。但是，目前市场上出现的主流税控收款机的处理器芯片大都是以 8 位或者 16 位单片机为主，现有的功能已经越来越不能满足客户的需求。以此为契机，我们研制和开发了新一代的基于 32 位 ARM 处理器的嵌入式系统的税控收款机 ZTax，以满足市场的需要。

本文首先简要介绍了税控收款机，指出了目前市场流行的税控收款机的不足之处。第二章介绍了 ZTax 的系统设计，首先介绍了它的开发环境，然后整体设计了系统的硬件和软件方面的系统结构，其中也研究了 uClinux 操作系统。第三章具体设计并实现了系统中出现的几个主要设备的驱动程序，包括 LCD 等等。第四章介绍了 ZTax 对于数据保护的处理方法，其中详细分析了文件系统 YAFFS 的实现机制，并完成了它的移植工作，也介绍了 Flash 卡的选用情况，然后论述了系统对于掉电保护的处理方法。第五章实现了 ZTax 基于 QTE 的应用程序的设计。最后是总结和展望。

关键字：嵌入式系统，uClinux，设备驱动，文件系统，YAFFS，税控收款机

Abstract

The embedded system is the outcome of the combination of computer technology, semiconductor, electronic technology and the application of kinds of industries. The embedded system is becoming the one of the focus in the IT industry. But meanwhile, the widely-used embedded applications need better performance and functionality of embedded systems. With the development of the embedded system, single chip board with 8 or 16 bit MCU can't fulfill the meet of applications. And with the rapid development of the integrated circuit technology, the price of 32-bit MCU is not very higher than 8-bit MCU now, and the RISC machines based on 32-bit are becoming more and more popular.

Tax-controlled Cash Register (TCR) is a cash register, which can control tax, and it is widely used in supermarkets, ordinary shops, pubs and other such kinds of places. However, the MCUs of Tax-controlled Cash Registers mainly sailed on today's market are most 8 or 16-bit MCUs, and their functions can not meet the needs of consumers any more. So, under this background, we have designed and developed the new-generation of Tax-controlled Cash Register based on embedded system with 32-bit ARM CPU.

At first, this thesis introduces what Tax-controlled Cash Register is, and outlines the deficiencies of the prevalent TCRs. In chapter 2, it introduces the designation of the system. First it introduces the environment of this project and the preparations before development. Then it designs the whole system including hardware and software. It also introduces uClinux, including its architecture, memory management, and multi-process management. In chapter 3, it introduces the device driver under uClinux, and how to implement, particularly the device of LCD and etc. are detailed. In chapter 4, it introduces an approach to protect the data. It analyzes the mechanism of the

implementation of YAFFS, and completes the porting of the file system YAFFS. It also chooses a kind of FLASH card. And it discusses how to handle the system when power is off illegally. In chapter 5, it introduces the application software of the Tax-controlled Cash Register based on QTE. At last, it gives some conclusions and foresights.

KEY WORDS: Embedded system, uClinux, Device driver, File system, YAFFS, Tax-controlled Cash Register

目　录

第一章　综述 6
1.1.　税控收款机介绍 6
1.2.　税控收款机现状 7
1.3.　嵌入式系统介绍 9
1.4.　研究内容和文章组织 11
第二章　ZTax 的系统设计 12
2.1.　主机平台及开发环境的建立 12
2.1.1.　主机平台 12
2.1.2.　交叉编译工具 13
2.1.3.　调试工具 14
2.1.4.　交互控制终端 15
2.2.　硬件设计 15
2.2.1.　电源模块设计 17
2.2.2.　网络模块设计 17
2.2.3.　存储模块设计 18
2.3.　软件设计 18
2.3.1.　整体设计 18
2.3.2.　操作系统 19
第三章　驱动程序的设计与实现 26
3.1.　Linux 下的设备驱动 26
3.2.　LCD 驱动设计及实现 28
3.2.1.　LCD 控制器 28
3.2.2.　FrameBuffer 介绍 30
3.2.3.　LCD 驱动程序 30
3.2.4.　对帧缓冲区的操作 36
3.3.　IIC 驱动程序 36

3.3.1. 数据结构 37
3.3.2. 相关文件 38
3.3.3. 主要模块 38
3.4. 其它驱动程序 39
第四章 ZTax 的数据保护方案设计 41
4.1. 问题的提出及解决方案 41
4.2. FLASH 卡 42
4.3. 文件系统的选择 46
4.4. YAFFS 文件系统 46
4.4.1. MTD 设备 47
4.4.2. YAFFS 文件系统的分析 51
4.4.3. YAFFS 文件系统的实现 57
4.5. 对掉电保护的处理 60
第五章 ZTax 应用程序的设计和实现 62
5.1. 设计与实现 62
5.1.1. 工作状态 62
5.1.2. 函数接口 65
5.2. GUI 的选择 67
第六章 总结与展望 69
参考文献 70
论文和参与的项目 73
致谢 74

第一章 综述

1.1. 税控收款机介绍

税控收款机是在能独立满足“税控”和发票管理基本要求的基础上，同时满足普通收款机在餐饮、娱乐、服务业的基本管理和普通票据及报表打印要求的税控装置产品[1]。税控收款机是在电子收款机的基础上发展起来的[2]，利用电子收款机的特点，税控机将电子收款机里面加入税务监控的“黑盒子”，把进行数据记录下来，使税务机关的监控触角延伸到纳税人的经营数据形成的过程中，税务稽查有一个依据。推广使用税控收款机可以如实记录商家的实际流水，明确交易发生额，准确统计应税额，保证所有交易有打印凭证即发票确认，上报税控收款机统计并打印的计税报表等。

税控收款机系统应解决的问题是[1]：

1、商业收款机功能与税控功能的有机结合；

2、机内程序不可复制、更改和数据的不可更改、清除；

3、安全、易用、可靠的申报手段；

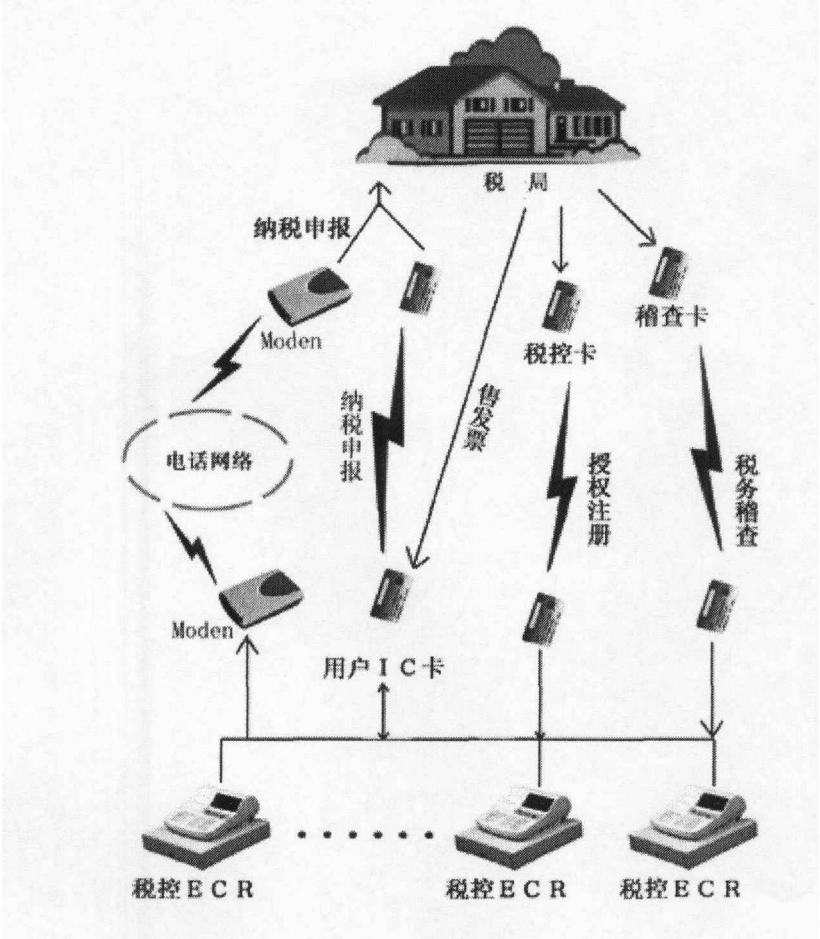
4、对发行、维修、更新安全性的支持；

5、对历史数据稽查的支持；

6、对专业犯罪集团对税控收款机侵入的可靠对抗；

7、事务完整性。

税控收款机实际上是一个系统的概念[4]，包括四个层次的内容：一是税控收款机本机；二是税控机制；三是税务征管部门发行、申报、管理、稽查系统；四是管理、运作、监控规范。如下图所示：



图表 1 税控系统

本文研究的是税控收款机本机，下文如果没有特别说明的话，都是指这一概念。

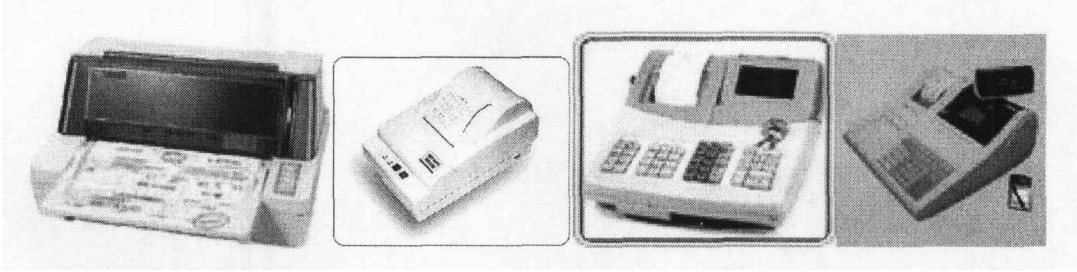
1.2. 税控收款机现状

税控收款机是国家金税工程带动的新兴产业。税控收款机的应用广泛，从大型百货、大超市到小型零售百货业、小型超市，药店，各种专卖店，餐饮、娱乐业，旅店，服务业等都可以看到它的身影。1994 年，金税工程试点启动。经过近 10 年的积累准备，税控收款机行业目前正在迅速崛起。随着金税工程的不断完善，中国税控机产业的前景备受瞩目。2003 年 10 月份，业界翘盼多时的“税控收款机国家标准”尘埃落定[3]。

目前中国收款机拥有量只有几十万台。据有关部门统计，按全国 3000 多万纳税户计算，税控收款机理论市场需求量至少为 3000 万台，按每台 3000 元计算，则全国的市场规模至少为 900 亿元[3]。

随着国民经济的发展，中国收款机市场将会迎来更加广阔的前景，中国将会成为世界最大的收款机市场。中国已加入 WTO，中国收款机市场的竞争将会更加激烈。竞争的结果，在保证质量的情况下价格和利润将会不断下降，国产收款机厂商只有上档次、上规模，发展中国的名牌收款机，才能在竞争中立于不败之地。

现在国内税控机具的生产厂家已近 100 家[5]。市场的庞大需求，使国内家电、IT 企业纷纷上马税控项目。各厂家纷纷生产出各自的产品，下图，我们列出了一些国内知名厂家的税控收款机。



映美 FP-530K　　海康 CETC-7000　　浪潮 RE400FB　　亿利达 FCR3500

图表 2 目前的代表产品

经过调查后我们发现，目前虽然市场上税控收款机品种多样，但大都有如下不足之处：

1、所用的处理器芯片一般是 8 位、16 位的；

2、大多没有采用操作系统；

3、液晶显示屏都是点阵式的（当然这也受限于所用的处理器）；

4、存储容量比较小；

5、可使用外设比较少，比如一般不带条形扫描仪；

6、功能还是不够强大。

基于目前这样现状以及市场前景，我们利用现有的成熟的嵌入式系统开发新一代的税控收款机 ZTax 来迅速打入这一市场。

1.3. 嵌入式系统介绍

嵌入式系统（Embedded System）指的是以微处理器（MCU，Micro Control Unit）为核心，用以控制特定设备的专门的系统。嵌入式系统到现在为止还没有统一的定义，一般可被定义为：以应用为中心、以计算机技术为基础、软件硬件可裁剪、适应应用系统对功能、可靠性、成本、体积、功耗严格要求的专用计算机系统[9]。嵌入式系统广泛应用于社会各个领域，如制造工业、过程控制、通讯、仪器、仪表、汽车、船舶、航空、航天、军事装备、消费产品等。

一般的，嵌入式系统是面向应用的，即整个系统作为一项特定的应用提供服务，如电话中的嵌入系统负责电话的拨出、接听、通话；微波炉中的嵌入系统负责微波的辐射量和辐射时间等等。每一种嵌入式系统都是根据特定应用要求而定做的。早在20世纪60年代，在通信领域就出现了用来控制电话的电子式机械交换设备，被称为“存储程控控制”（Stored Program Control）。今天嵌入式系统带来的工业年产值已超过了1万亿美元[9]。

为使嵌入式系统具有更强的功能、更好的扩展性，将操作系统加入到嵌入式系统中已经成为嵌入式系统发展的热点和大方向。嵌入式操作系统（Embedded Operating System）支持嵌入式系统硬件平台，封装了硬件细节，向应用软件提供高层软件开发接口。

目前的EOS主要有三类，一类是通用的由第3方软件厂商提供的操作系统，如Java OS（SUN）、WinCE（Microsoft）、VRTX（Microtex）、pSOS（WindRiver Systems Inc.）、VxWorks（WindRiver Systems Inc.）等现成的操作系统；采用这种方式的优点是系统软件结构清晰，具有较好的扩展性，同时便于应用软件的移植. 但最终用户必须为软件支付较高的费用，而在这些软件所提供的功能中有很多并非用户所需要的。

第二类是根据需要专门开发的针对自己系统的专用实时操作系统。这类系统一般是在单片机上直接开发，操作系统和目标代码之间没有明显的界限，操作系统是一段嵌入在目标代码中的程序，系统复位后它首先执行，相当于应用的主程序。这类系统的一大优点是系统的总代码长度比较短，而这对于嵌入式系统的小存储要求则是再好不过了。由于是针对具体的单片机开发的系统，因

此可以最大程度的利用该类单片机的专有特性。正是由于系统代码和目标代码的共存，导致了系统编制和测试的困难度加大，而值得庆幸的是，随着各种模拟器和编程器的出现，使得这类系统的编制和调试也越来越方便。

第三类是在一些开放源代码的现有操作系统的核心基础上，通过定制方式生成符合嵌入式需要的操作系统，这类系统的典型代表是在 Linux 操作系统的基础定制的各种嵌入式实时操作系统。如 Lineo 公司在 Linux 的基础上开发的几种嵌入式操作系统有 Embedix™ （embedded Linux OS），Embrowser™ （embedded Web browser），DR DOS® （component-based embedded OS）。EOS 的体系结构主要有三种：模块化单块结构、层次化内核结构和微内核结构。

鉴于 EOS 是为嵌入式系统服务的操作系统，具备嵌入式系统的一些共同特点：

- 实时性。在多任务嵌入式系统中，由于各个任务的重要性各不相同，因此对任务进行统筹兼顾的合理调度就成为保证每个任务及时执行的关键。这种任务调度单纯通过提高处理器速度是无法完成，而且是没有效率的，它只能通过优化编写的系统软件来完成，因此 EOS 的高实时性是嵌入式应用的基本要求。
- 最小存储尺寸。为了提高执行速度和系统可靠性，EOS 一般都固化在存储器芯片或单片机本身中。尽管半导体技术的发展使处理器速度不断提高、片上存储器容量不断增加，但在大多数应用中，存储空间仍然是宝贵的，还存在实时性的要求，而且为了降低系统的成本，嵌入式系统的容量必须尽可能的小。
- 代码要高质量、高可靠性。嵌入式应用在开发完成后，一般都要固化在存储器芯片或单片机中，以后基本不能再修改。目前应用程序也可以存放在如 Flash 卡可修改的载体中，而且可以通过网络升级应用程序，大大方便了用户。另外由于嵌入式系统对存储空间的限制，因此要求程序编写和编译工具的质量要高，以减少程序二进制代码长度、提高执行速度。
- 系统的可配置性。EOS 必须具有高度的可配置性，以便开发人员可以根据应用的需要对其进行裁减，使最终的应用不但能高效的工作，且不会

产生过多的存储浪费。

基于EOS开发嵌入式系统正在成为开发嵌入式系统的大方向，目前国际已经有比较成熟的、商品化的嵌入式操作系统，如pSOS、Palm OS、Windows CE、OS9、Java OS以及各种基于Linux的嵌入式操作系统。EOS也有了很多成熟的应用，最明显的便是各种移动设备，包括各种手机、PDA、定位仪等等。

1.4. 研究内容和文章组织

本文以ARM7+uClinux目前比较流行的嵌入式系统为基础，开发了新一代税控收款机ZTax。uClinux已经在诸多处理器芯片上提供了直接的支持，比如三星公司的S3C4510B（同样也是ARM7TDMI内核），本身只要配置正确即可。不过考虑到S3C4510B没有LCD控制器，因此我们选择带LCD控制器的S3C44B0X。但是uClinux 不直接提供对S3C44B0X的支持，我们还必须要移植uClinux到S3C44B0X上去。

本文完成了系统的整体设计，在其上开发了硬件的设备驱动程序，而且解决了税控收款机上数据处理的关键问题，并设计与实现了系统的应用程序。

本文第二章介绍了ZTax的系统设计。首先介绍了它的开发环境，然后整体设计了系统的硬件和软件方面的系统结构，其中也介绍了uClinux操作系统。

第三章具体介绍了系统中出现的主要几个设备的驱动程序，包括LCD，IIC等等，以及开发的具体步骤，实现难点及相关的一些技术细节。

第四章介绍了ZTax对于数据保护的处理方法，其中涉及了文件系统YAFFS的分析及它的实现，也介绍了Flash卡的一些情况，然后介绍了系统对于掉电保护的处理过程。

第五章我们实现了ZTax基于QTE的应用程序的设计和实现。

第二章 ZTax 的系统设计

2.1. 主机平台及开发环境的建立

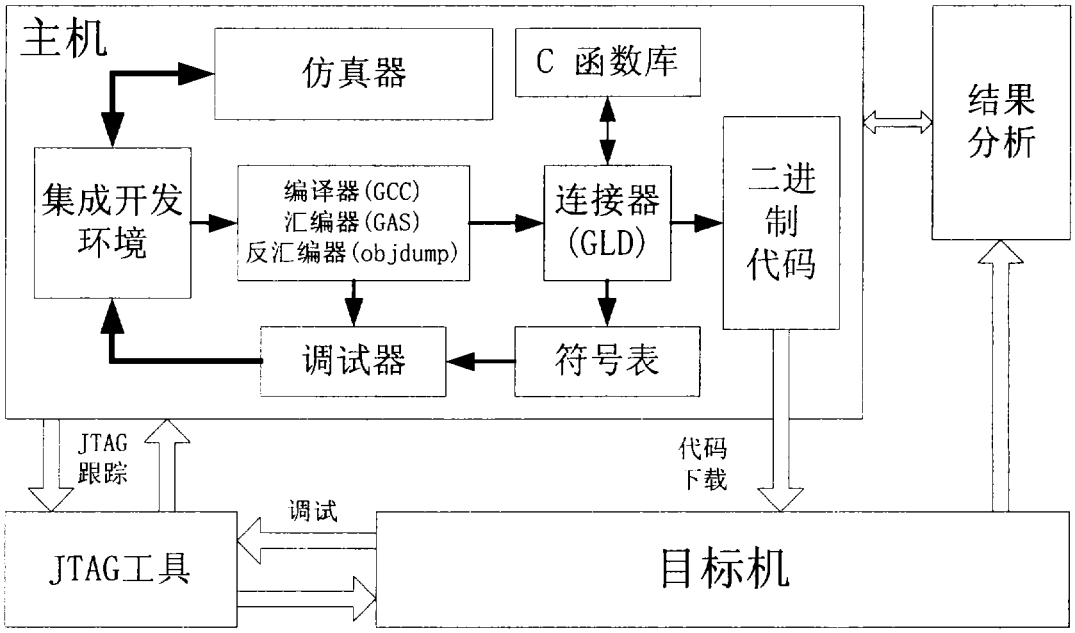
在明确了研究目标之后，就可以开始为我们的工作做准备。在准备阶段我们首先要在主机平台上建立开发环境：

1）准备主机平台：一般说来，嵌入式系统首先需要在主机平台上开发，而且我们的绝大部分工作都需要在主机平台上完成。

2）准备交叉编译工具。

3）准备跟踪调试等一些主机跟目标机协同工作的工具。

下图是我们开发嵌入式系统的整个工作环境：



图表 3 整个工作环境

2.1.1. 主机平台

推荐工作平台的配置：处理器至少 PII 以上，内存容量至少 128MB 以上，硬盘空间至少有 500MB 空闲为好。

操作系统采用 Linux 为好。

2.1.2. 交叉编译工具

支持一种新的处理器，必须具备一些编译，汇编工具，使用这些工具可以形成可运行于这种处理器的二进制文件。对于内核使用的编译工具同应用程序使用的有所不同。我们先对 GCC 连接做一些说明：

- ld（link description）文件：ld 文件是指出连接时内存映象格式的文件。
- crt0.S：应用程序编译连接时需要的启动文件，主要是初始化应用程序栈。
- PIC：Position Independence Code，与位置无关的二进制格式文件，在程序段中必须包括 reloc 段，从而使得代码加载时可以进行重新定位。

内核编译连接时，使用 ld 文件，形成可执行文件映象，所形成的代码段既可以使用间接寻址方式（即使用 reloc 段进行寻址），也可以使用绝对寻址方式。这样可以给编译器更多的优化空间。因为内核可能使用绝对寻址，所以内核加载到的内存地址空间必须与 ld 文件中给定的内存空间完全相同。

应用程序的连接与内核连接方式不同。应用程序由内核加载，由于应用程序的 ld 文件给出的内存空间与应用程序实际被加载的内存位置可能不同，这样在应用程序加载的过程中需要一个重新定位的过程，即对 reloc 段进行修正，使得程序进行间接寻址时不至于出错。

由上述讨论，至少需要两套编译连接工具：

1） 二进制工具包 Binutils

GNU Binutils 工具包包括了汇编工具（GAS）、链接器（GLD）和基本的目标文件处理工具（如反汇编工具 OBJDUMP 等）。对 Binutils 包的设置定义了所需的目标文件的格式和字节顺序。Binutils 包中的工具都使用了二进制文件描述符（BFD）库来交换数据。通过设置文件 config.bfd，可以指定默认的二进制文件格式（例如 elf little endian）和任何工具可用的格式，比如在 config.bfd 中添加的用来指定目标二进制格式的代码：

```
arm-*-uClinux* | armel-*-uClinux*
```

```
tag_defvec=bfd_elf32_littlearm_vec
targ_selvecs="bfd_elf32_bigarm_vec armcoff_little_vec armcoff_big_vec"
```

2） C编译器

GNU 编译器 GCC 是通过使用一种叫做“寄存器转换语言”（RTL）的方式实现的。假定现在有一种基本的机器描述性文件，它已经能满足大家的需要。现在要做的仅仅是设置默认情况下使用的参数和如何将文件组合成可执行文件的方式。GNU 的文档提供了所有必需的资料，使得用户可以为新型的处理器的指令集合提供支持。如果要针对体系的机器建立一个新的目标机器，那么就必须指定默认编译参数和定制系统的特定参数，比如，在 uClinux-arm.h 中：

```
#undef TARGET_DEFAULT
#define TARGET_DEFAULT(ARM_FLAG_APCS_32|ARM_FLAG_NO_GOT)
```

对于特定的目标系统，可以使用 TARGET_DEFAULT 宏作为在 target.h 文件中定义编译器的开关。目标 t-makefile 段指定了应该构建哪一个额外的例程和其编译的方式。

不过，目前网上都已经提供了相应的目标机器的编译器二进制代码[16]，以下的一些工具包是我们在本系统开发中采用的版本：

arm-elf-gcc-2.95.3-2.i386.rpm

arm-elf-binutils-2.11-5.i386.rpm

arm-elf-gdb-5.0-2.i386.rpm

genromfs-0.5.1-1.i386.rpm

ncurses4-5.0-5.i386.rpm

2.1.3. 调试工具

除了上面提到的一些工具之外，我们还需要一系列调试工具。在 Linux 下，我们可以采用：

- Gdb：调试器，它可使用多种交叉调试方式；
- gdb-bdm：背景调试工具；

- gdbserver：使用以太网络远程调试。

另外，比较常用的是实时在线仿真系统 ICE(In-CircuitEmulator)。它是通过 JTAG 口调试的。在计算机辅助设计非常发达的今天，它仍是进行嵌入式应用系统调试最有效的开发工具。高级的 ICE 带有完善的跟踪功能，可以将应用系统的实际状态变化、微控制器对状态变化的反应、以及应用系统对控制的响应等连续记录下来以供分析，在分析中优化控制过程[17]。

2.1.4. 交互控制终端

通常情况下，uClinux 的默认终端是串口，内核在启动时所有的信息都打印到串口终端（使用 printk 函数打印），同时也可以通过串口终端与系统交互。

uClinux 在启动时启动了 telnetd（远程登录服务），操作者可以远程登录上系统，从而控制系统的运行。至于是否允许远程登录可以通过烧写 romfs 文件系统时由用户决定是否启动远程登录服务。

在主机平台的 Linux 环境下，我们可以使用 minicom 作为与目标机的交互终端。在 Windows 下，我们可以使用 Windows 自带的超级终端或者其他类似的终端，如现在比较用的多的是 sscom，目前最新的版本是 3.2。

2.2. 硬件设计

税控收款机一般由显示系统、打印系统、中央处理系统、专用税控处理系统、专用外围设备驱动系统、电源系统、键盘等几部分组成。中央处理芯片采用三星公司的 S3C44B0X。它的内核是 ARM7TDMI。S3C44B0X 已内置的资源有：

（1） 2 通道 UART，波特率可高达 115200bps，并内置 16 字节的 FIFO，同时兼容 Irda1.0 规范；

（2） 1 通道 IIC 接口（支持多主模式）；

（3） 1 个 IIS 接口（音频数据接口）；

（4） 1 个 SIO 接口，兼容 SPI/SCI 接口；

（5） 8 通道 10bit ADC（采样速率为 100KBPS）；

（6） 4 通道 PWM 输出；

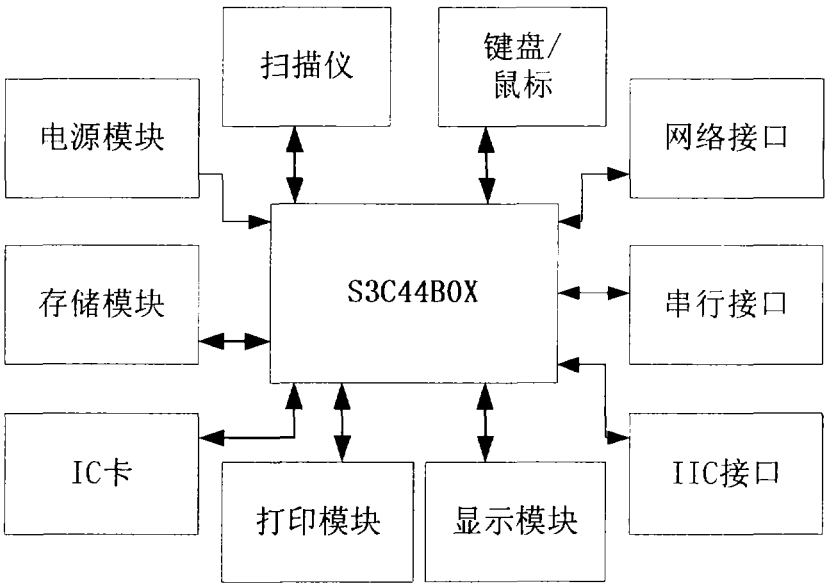
（7） 8 条外部中断口；

（8） 1 个 RTC（实时时钟）；

（9） EmbeddedICE（JTAG）接口；

（10） LCD 控制器（可直接控制 DSTN/STN 的各种灰度/256 彩色 LCD 屏，最大支持分辨率为 1600*1600）。

由于 S3C44B0X 自带 LCD 控制器，因此我们不需要另外扩展 LCD 的控制模块，但是没有网络控制器，根据实际需要，我们要另外扩一个网络控制器。另外，外围设备还有打印机（EPSON 的 m-u110 型号的打印机）、条形扫描仪等等。下图是硬件结构框图：



图表 4 目标机硬件结构框图

根据这个逻辑框图，我们实现了低成本的硬件设计方案：

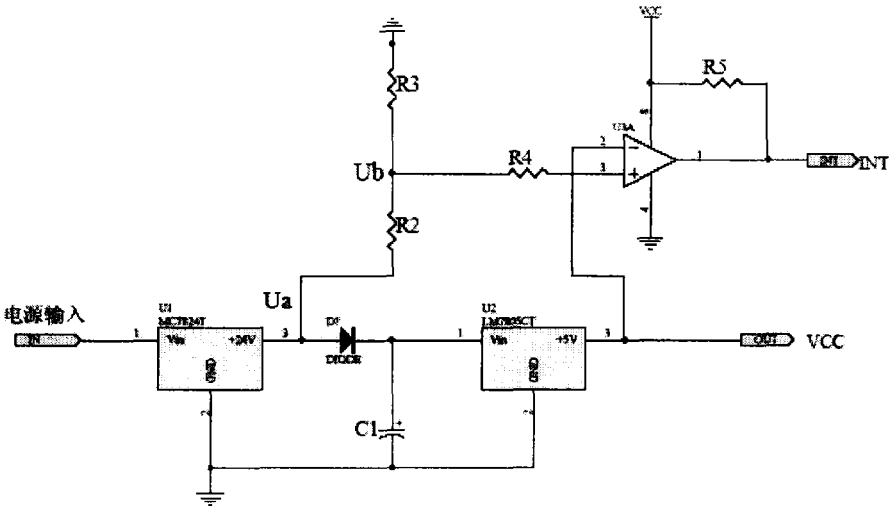
1、S3C44B0X 丰富的外围接口模块减少了需要其他控制芯片的数量，从而降低主板成本。

2、采用低成本的 NAND FLASH 作为数据和程序存储器。

3、采用 CPU 直接控制打印机，不需要另外的打印机控制器。

2.2.1. 电源模块设计

由于税控收款机对掉电保护有严格的要求，在电源管理模块中，我们的工作电压是 5v，在它前面是一个高电压 24v，分压后和 5v 标准电压通过一个电压比较器进行比较，当电压低于 24v 时，电压比较器得到低电平，发送给处理器一个中断到处理器，告知已经掉电，但是现在由于电容的作用，可以维持一定的时间，使系统进入掉电中断程序，处理未完成的工作。



图表 5 电源控制模块

在上图中，我们用 7805 和 7824 分别产生 5v 和 24v 的电压，5v 是工作电压，Ua 点是 24v，一路作为 7805 的输入电压，一路接大电容 C1 保存电量，还有一路经过 R2 和 R3 的分压，在 Ub 点产生略大于 5v 的电压。在正常工作情况下，比较器（运放采用 LM339）同相输入端电压大于反相输入端电压，于是在比较器输出端——INT 端（接入 S3C44B0X 的中断输入）得到一个高的电平；一旦掉电后，Ua 端不能维持 24v 的电压，于是比较器同相输入端电压小于反相输入端电压，在 INT 端得到一个低电平，这样就产生了一个中断触发，此时 C1 开始放电以供应系统需要，达到了我们的设计目的。

2.2.2. 网络模块设计

由于 S3C44B0X 本身不包含网卡控制器，因此如果系统需要支持网络功能，

必须要求扩展一个网络模块。在参考了已有电路的基础上，我们制作了网络模块的电路图。我们选用的网卡芯片是 8019。

2.2.3. 存储模块设计

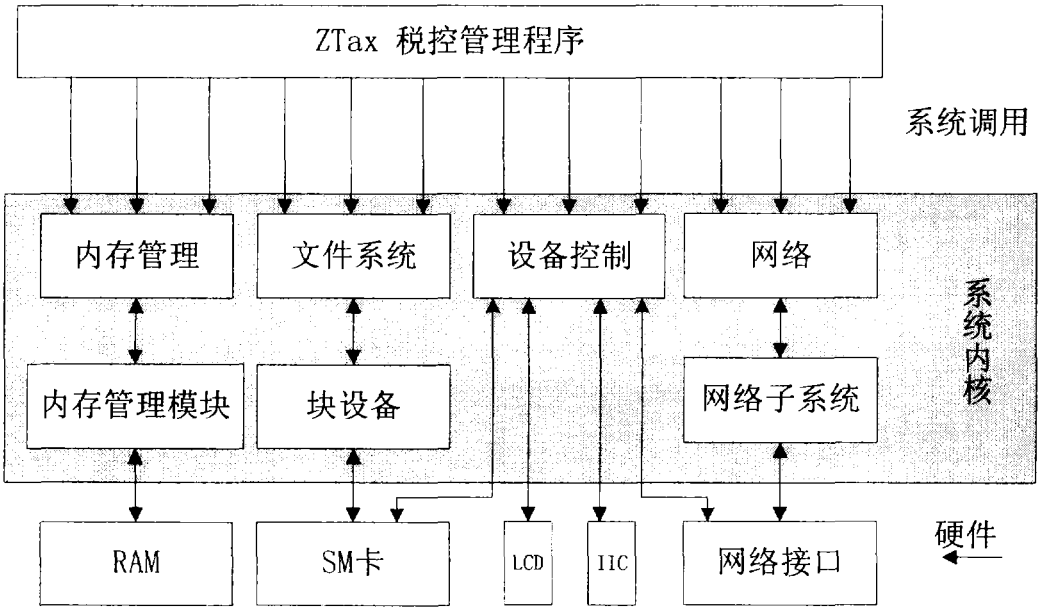
由于税控收款机需要存储大量的历史交易数据，因此还需要在片外扩展一片存储器，还可以用来存放操作系统内核以及应用程序代码，我们选用 32MB 的 SM 卡，以后也可以根据需要选用 64MB 或者 128MB 的 SM 卡。

2.3. 软件设计

2.3.1. 整体设计

软件部分分为系统和应用两个层次。

系统部分最主要的是操作系统，操作系统是应用程序和硬件设备之间的接口，它要实现对每个设备的驱动。应用软件部分实现税控收款机具体的功能，包括实现对商品的条码扫描，手工录入，商品信息的管理，税票的管理，打印输出等功能。整个软件部分体系结构如下图。



图表 6 软件功能模块

下文我们具体介绍操作系统，而对设备驱动的设计，ZTax 系统数据安全的处理，以及用户应用程序将分别在以下几章中阐述。

2.3.2. 操作系统

一般目前的嵌入式系统在操作系统方面主要有三种方案：

第一种就是没有操作系统，用户自己在应用程序中直接控制硬件设备，或者是通过一些类操作系统的库文件来管理硬件设备。这种方法对一些小型系统特别是 8 位机还比较适合，并且也是比较有效的一种方法，但是如果系统庞大，有许多外围设备，那么管理起来将是非常麻烦的。

第二种是用户自己编写一个适合于本硬件系统的操作系统，这种方案工作量比较大，并且从头开始设计也不太现实。

第三种是采用现有的比较成熟的操作系统进行裁减，移植到我们的系统上去。这种方案为大多数嵌入式系统应用所采用。

综合各方面的权衡比较后得出结论，第三种方案比较适合这个系统。而且由于硬件上采用了 32 位的 S3C44B0X，本身已经有比较成熟的操作系统可以支持。为力求系统整体的成本最低化，我们只考虑两个优秀的免费操作系统：uClinux 和 uCOS(II)。

2.3.2.1. uCOS 操作系统

uCOS 是一种免费公开源代码、结构小巧、具有可剥夺实时内核的实时操作系统。其内核提供任务调度与管理、时间管理、任务间同步与通信、内存管理和中断服务等功能[14]。

作为实时操作系统，uCOS 是采用的可剥夺型实时多任务内核。可剥夺型的实时内核在任何时候都运行就绪了的最高优先级的任务。uCOS 中最多可以支持 64 个任务，分别对应优先级 0~63，其中 0 为最高优先级。调度工作的内容可以分为两部分：最高优先级任务的寻找和任务切换。其最高优先级任务的寻找是通过建立就绪任务表来实现的。uCOS 中的每一个任务都有独立的堆栈空间，并有一个称为任务控制块 TCB(Task Control Block)数据结构。

uCOS 是面向中小型嵌入式系统的，如果包含全部功能(信号量、消息邮箱、消息队列及相关函数)，编译后的 uCOS 内核仅有 6~10KB，所以系统本身并没有对文件系统的支持。

2.3.2.2. uClinux 操作系统

uClinux 是在 Linux kernel 2.0 之后出现的一个 Linux 的变种，它的目标是将 Linux 应用于没有内存管理单元（MMU，Memory Management Units）的处理器。uClinux 是针对“微控制领域而设计的 Linux 系统”。嵌入式操作系统是嵌入式系统的灵魂，而且在同一个硬件平台上可以嵌入不同的嵌入式操作系统。比如 ARM7TDMI 内核，可以嵌入 Nucleus、VxWorks、uClinux 等多种操作系统[9]。

由于 uClinux 是在 Linux 的基础之上添加了对没有 MMU 的微处理器的支持，所以它一方面继承了 Linux 的大部分优点：稳定性、各种网络协议栈的支持以及不同类型的文件系统支持等等，另一方面，它广泛地应用于嵌入式领域，内核精简（一般小于 512k），并支持很多常用的嵌入式微控制器系列，例如 Motorola Dragon Ball、ColdFire、ARM7TDMI（ Atmel AT91x、Samsung S3C4510X，S3C44B0X）等。

一个最小型的嵌入式 uClinux 系统只需要下面三个基本元素：1、引导工具；2、uClinux 微内核——由内存管理、进程管理和事务处理构成；3、初始化进程。如果要系统要有实际功用，且继续保持小型化，还需要：硬件驱动程序，提供所需功能的一个或更多的应用程序，文件系统（在 ROM 或 RAM 中），TCP / IP 网络协议栈，存储系统等等。uClinux 可以通过定制使内核小型化，还可以加上 GUI(图形用户界面)和定制应用程序，并将其放在 ROM、RAM、FLASH 或 Disk On Chip 中启动。由于嵌入式 uClinux 操作系统的内核定制高度灵活性，开发者可以很容易地对其进行按需配置，来满足实际应用需要。又由于 uClinux 是源代码公开，因此开发人员只有了解内核原理就可以自己开发部分软件，例如增加各类驱动程序。

uClinux 内核结构

uClinux 内核结构与 Linux 基本相同，不同的只是对内存管理和进程管理进行改写，以满足无 MMU 处理器的要求。uClinux 是 Linux 操作系统的一种，是由 Linux2.0 内核发展来的（随着 Linux 内核的发展，现在 uClinux 内核也随之发展到了 2.4.x），是专为没有 MMU 的微处理器（如 ARM7TDMI、Coldfire 等）设计的嵌入式 Linux 操作系统。另外，由于大多数内核源代码都被重写，uClinux 的内核要比原 Linux 内核小的多，但保留了 Linux 操作系统的主要优点：稳定性，优异的网络能力以及优秀的文件系统支持。

从物理组成情况来看，通常一个编译好的内核主要是由以下几个部分组成的：

1）初始化程序段(init)

由内核中所有初始化程序的代码组成。这部分代码是用__init 来定义，在内核初始化结束后，它们所占的存储空间就会被释放以节省 RAM 资源。

2）数据段(data)

由内核中所有含初始化值的全局变量，数据结构组成。因为这些数据都是在内核运行的过程中被读写的变量，所以这个段必须放在 RAM 中，一般有 50~100k 左右。

3）未初始化数据段(bss)

由内核中所有未被初始化的全局变量，数据结构组成。这个段也必须放在 RAM 中，但这个段中的变量只有地址和大小，没有初始值，在最后生成的二进制下载映像文件中并不占用空间。这个段所对应的 RAM 区域通常在内核启动前初始化为 0。一般在 RAM 中将会占用 100k~150k 左右。

4）代码段(text)

由内核中所有非初始化程序的代码组成。这个段是内核代码的主体部分，为了节省 RAM 空间，可以把它放到 ROM 中执行，这个段的大小通常为 300k 左右。

5）文件系统(romfs)

romfs 是 uClinux 缺省使用的一种文件系统类型，它比较简单实用，支

持 romfs 文件系统的代码占用的 RAM 也比 Ext2 的小很多，非常适合嵌入式应用的需求。

配置内核的一个重要工作就是决定上述的各个内核组成模块在 RAM 或者 ROM 中的地址。通常，init、data、bss 和 text 的地址是由编译链接时的定位文件 vmlinux.lds 决定的，在这个文件中定义了各个段的起始地址，而 romfs 的文件系统的地址是在 blkmem.c 文件中给出的。通常要将 uClinux 移植到特定的硬件平台上，都会需要根据各自的情况对上述文件做一定的修改。

uClinux 的内存管理

uClinux 同标准 Linux 的最大区别就在于内存管理。标准 Linux 是针对有 MMU 的处理器设计的。在这种处理器上，虚拟地址被送到 MMU，MMU 把虚拟地址映射为物理地址。通过赋予每个任务不同的虚拟—物理地址转换映射，支持不同任务之间的保护。对于 uClinux 来说，其设计针对没有 MMU 的处理器，不能使用处理器的虚拟内存管理技术。

uClinux 不能使用处理器的虚拟内存管理技术，应该说这种不带有 MMU 的处理器在嵌入式设备中相当普遍。这是因为，一方面 MMU 在处理器中会占用相当大比例的硅片面积，基于成本和尺寸的考虑，没有 MMU 会更适合于嵌入式设备的应用；另一方面，当发生页面失效时，从磁盘加载进程的页面到内存中，会相当消耗时间和处理器资源，这对系统工作的影响很大，容易带来不稳定的因素。

uClinux 仍采用存储器的分页管理，系统在启动时把实际存储器进行分页。在加载应用程序时程序分页加载。但是由于没有 MMU 管理，所以实际上 uClinux 采用实存储器管理策略(real memory management)。这一点影响了系统工作的很多方面。这样一个进程在执行前，系统必须为之分配足够的连续地址空间，然后全部载入主存储器中。因为在嵌入式开发中，通常都是针对特定环境下的应用，所以这样的实现也是可行的，从执行效率上讲也更会高一些。

uClinux 系统对于内存的访问是直接的，它对地址的访问不需要经过 MMU，而是直接送到地址线上输出，所有程序中访问的地址都是实际的物理地址。操

作系统对内存空间没有保护（这实际上是很多嵌入式系统的特点），各个进程实际上共享一个运行空间（没有独立的地址转换表）。

一个进程在执行前，系统必须为进程分配足够的连续地址空间，然后全部载入主存储器的连续空间中。与之相对应的是标准 Linux 系统在分配内存时没有必要保证实际物理存储空间是连续的，而只要保证虚存地址空间连续就可以了。此外磁盘交换空间也是无法使用的，系统执行时如果缺少内存将无法通过磁盘交换来得到改善。

uClinux 对内存的管理减少同时就给开发人员提出了更高的要求。开发人员不得不参与系统的内存管理。从编译内核开始，开发人员必须告诉系统这块开发板到底拥有多少的内存，从而系统将在启动的初始化阶段对内存进行分页，并且标记已使用的和未使用的内存。系统将在运行应用时使用这些分页内存。

由于应用程序加载时必须分配连续的地址空间，而针对不同硬件平台的可一次成块（连续地址）分配内存大小限制是不同（目前针对 EZ328 处理器的 uClinux 是 128k，而针对 Coldfire 处理器的系统内存则无此限制），所以开发人员在开发应用程序时必须考虑内存的分配情况并关注应用程序需要运行空间的大小。另外由于采用实存储器管理策略，用户程序同内核以及其它用户程序在一个地址空间，程序开发时要保证不侵犯其它程序的地址空间，以使得程序不至于破坏系统的正常工作，或导致其它程序的运行异常。

从内存的访问角度来看，开发人员的权力增大了（开发人员在编程时可以访问任意的地址空间），但与此同时系统的安全性也大为下降。此外，系统对多进程的管理将有很大的变化，这一点将在 uClinux 的多进程管理中说明。

uClinux 的多进程处理

uClinux 没有 MMU 管理存储器，在实现多个进程时（fork 调用生成子进程）需要实现数据保护。由于 uClinux 的多进程管理是通过 vfork 来实现，因此 fork 等于 vfork。这意味着 uClinux 系统 fork 调用完成后，要么子进程代替父进程执行（此时父进程已经 sleep）直到子进程调用 exit 退出；要么调用 exec 执行一个新的进程，这个时候将产生可执行文件的加载，即使这个进程

只是父进程的拷贝，这个过程也不能避免。当子进程执行 exit 或 exec 后，子进程使用 wakeup 把父进程唤醒，使父进程继续往下执行。

当内核收到 fork 请求时，它会先查核三件事：首先检查存储器是不是足够；其次是进程表是否仍有空缺；最后则是看看用户是否建立了太多的子进程。如果上述说三个条件满足，那么操作系统会给子进程一个进程识别码，并且设定 CPU 时间，接着设定与父进程共享的段，同时将父进程的 inode 拷贝一份给子进程运用，最终子进程会返回数值 0 以表示它是子进程，至于父进程，它可能等待子进程的执行结束，或与子进程各做各的。

uClinux 的这种多进程实现机制同它的内存管理紧密相关。uClinux 是针对没有 MMU 处理器的开发，所以被迫使用一种 flat 方式的内存管理模式，启动新的应用程序时系统必须为应用程序分配存储空间，并立即把应用程序加载到内存。缺少了 MMU 的内存重映射机制，uClinux 必须在可执行文件加载阶段对可执行文件 reloc 处理，使得程序执行时能够直接使用物理内存。

针对实时性的解决方案

uClinux 本身并没有关注实时问题，它并不是为了 Linux 的实时性而提出的。另外有一种 Linux 也就是 Rt-linux 关注实时问题。Rt-linux 执行管理器把普通 Linux 的内核当成一个任务运行，同时还管理了实时进程。而非实时进程则交给普通 Linux 内核处理。这种方法已经应用于很多的操作系统用于增强操作系统的实时性，包括一些商用版 UNIX 系统，Windows NT 等等。这种方法优点之一是实现简单，且实时性能容易检验。优点之二是由于非实时进程运行于标准 Linux 系统，同其它 Linux 商用版本之间保持了很大的兼容性。优点之三是可以支持硬实时时钟的应用。uClinux 可以使用 Rt-linux 的 patch，从而增强 uClinux 的实时性，使得 uClinux 可以应用于工业控制、进程控制等一些实时要求较高的应用。

2.3.2.3. 分析结论

通过对 uCOS 和 uClinux 的比较，可以看出这两种操作系统在应用方面各

有优劣。uCOS 占用空间少，执行效率高，实时性能优良，且针对新处理器的移植相对简单，主要适合一些控制系统。uClinux 则占用空间相对较大，实时性能一般，针对新处理器的移植相对复杂。但是，uClinux 具有对多种文件系统的支持能力、内嵌了网络协议，可以借鉴 Linux 丰富的资源，对一些复杂的应用，uClinux 具有相当优势。

毫无疑问，由于我们硬件本身具有丰富的资源，而且实际的用户需求也对操作系统提出了要求，当然应该选择 uClinux。

第三章 驱动程序的设计与实现

uClinux 驱动程序的基本结构和标准 Linux 驱动程序的结构类似，因此我们下文可以用 Linux 来叙述。但标准的 Linux 支持模块化(module)，所以大部分的设备驱动都写成模块的形式，而且要求可以在不同的体系结构上安装。uClinux 是可以支持模块功能的，在编译内核的时候可以选择支持或不支持模块功能；但嵌入式系统是针对具体应用的，一般都不需要这一功能，而且嵌入式系统中内存空间资源比较紧张，所以在编译内核的时候，一般选择不支持模块功能。这样，uClinux 的驱动程序都是直接编译到内核中的[20]。

3.1. Linux 下的设备驱动

Linux 将设备分为最基本的两大类，字符设备和块设备。字符设备是以单个字节为单位进行顺序读写操作，通常不使用缓冲技术，如鼠标等，驱动程序实现比较简单；而块设备则是以固定大小的数据块进行存储和读写的，如硬盘，软盘等。为提高效率，系统对于块设备的读写提供了缓存机制，由于涉及缓冲区管理，调度，同步等问题，实现起来比字符设备复杂得多。

Linux 的设备管理是和文件系统解密结合的，各种设备都以文件的形式存放在/dev 目录下，称为设备文件。应用程序可以打开，关闭，读写这些设备文件，完成对设备的操作，就像操作普通的数据文件一样。为了管理这些设备，系统为设备编了号，每个设备号又分为主设备号和次设备号。主设备号用来区分不同种类的设备，而次设备号用来区分同一类型的多个设备。对于常用设备，Linux 有约定俗成的编号，如硬盘主设备号是 3。

Unix / Linux 的特点之一，是为所有的文件，包括设备文件，提供了统一的操作函数接口，在 uClinux 源代码 uClinux-dist/linux-2.4.x/include/linux/fs.h 中，定义了 uClinux 驱动程序中必须使用的 file_operations 结构，定义如下：

```
struct file_operations {
  struct module *owner;
```

```
        loff_t (*llseek) (struct file *, loff_t, int);
        ssize_t (*read) (struct file *, char *, size_t, loff_t *);
        ssize_t (*write) (struct file *, const char *, size_t, loff_t *);
        int (*readdir) (struct file *, void *, filldir_t);
        unsigned int (*poll) (struct file *, struct poll_table_struct *);
        int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned long);
        int (*mmap) (struct file *, struct vm_area_struct *);
        int (*open) (struct inode *, struct file *);
        int (*flush) (struct file *);
        int (*release) (struct inode *, struct file *);
        int (*fsync) (struct file *, struct dentry *, int datasync);
        int (*fasync) (int, struct file *, int);
        int (*lock) (struct file *, int, struct file_lock *);
        ssize_t (*readv) (struct file *, const struct iovec *, unsigned long, loff_t *);
        ssize_t (*writev) (struct file *, const struct iovec *, unsigned long, loff_t *);
        ssize_t (*sendpage) (struct file *, struct page *, int, size_t, loff_t *, int);
        unsigned long (*get_unmapped_area)(struct file *, unsigned long, unsigned long,
    unsigned long, unsigned long);
    };
```

结构体中的成员为一系列的接口函数，如用于读/写的 read/ write 函数，用于控制的 ioctl 等。打开一个文件就是调用这个文件 file_operations 中的 open 操作。不同类型的文件有不同的 file_operations 成员函数。如普通的磁盘数据文件，接口函数完成磁盘数据块读写操作；而对于各种设备文件，则最终调用各自驱动程序中的 I/O 函数进行具体设备的操作。这样，应用程序根本不用考虑操作的是设备还是普通文件，可一律当作文件处理，具有非常清晰统一的 I/O 接口。所以 file_operations 是文件层次的 I/O 接口。

但是，由于外设的种类繁多，操作方式也各不相同。如显示设备驱动要提供对显存的操作，硬盘驱动要处理复杂的缓冲区结构，网络设备驱动和 socket 联系紧密。如果 file_operations 中的函数都让驱动程序的开发人员来写，则就要处理大量的细节，几乎是不可能的。为了解决设备多样性的问题，Linux 采用了特殊情况特殊处理的办法，为不同设备定义好了文件层次 file_operations 结构中的接口函数，其中处理了大多数设备相关的操作，如各种缓冲区的申请和释放

等等，而具体操作底层硬件的一小部分则留给开发人员。所以 Linux 另外提供一个文件层到底层驱动程序的接口，通常为一个结构体，其中包含成员变量和函数指针。不同的设备驱动有不同的结构体。这样，一方面保证了文件层 I/O 接口 file_operations 的一致性，另一方面驱动程序的开发人员也不用了解太多细节，只专注于硬件相关的 I/O 操作就可以了。

下面我们以 LCD 和 IIC 的驱动为例做详细介绍。

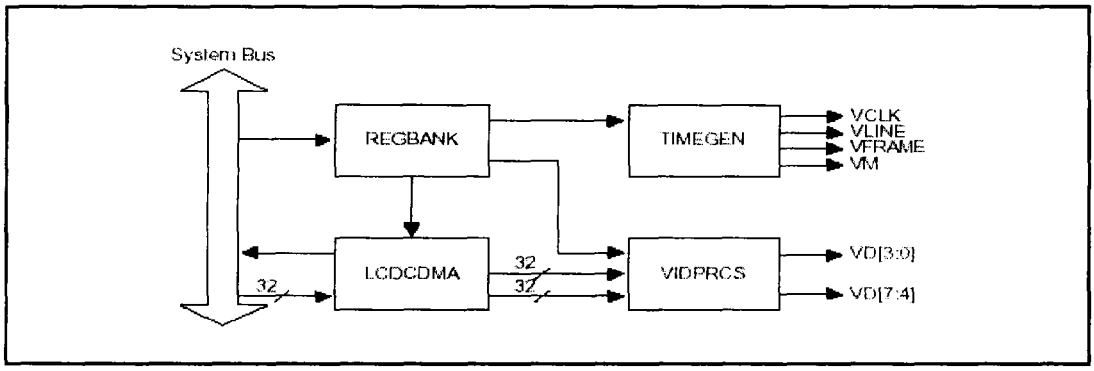
3.2. LCD 驱动设计及实现

3.2.1. LCD 控制器

控制器功能

一般的液晶显示控制器主要由接口部分，驱动部分和控制部分组成。液晶显示控制器的接口部分由指令寄存器/译码器、数据缓冲器和状态字寄存器以及相关逻辑电路组成。它用来接收计算机发来的指令和数据，并向计算机反馈所需的数据信息。驱动部分是液晶显示控制器对液晶显示驱动系统的接口。时序发生器产生基础时钟提供给显示时序电路，显示时序电路产生显示时序脉冲序列提供给驱动部分。液晶显示控制器的控制部分是液晶显示控制器的核心。产生工作的时钟直接提供给时序发生器以生成控制时序和显示时序。控制时序将驱动逻辑电路以管理和操作各功能电路。

S3C44B0X 内置 LCD Controller [24]功能是非常强大的，可以用来控制 2、4、16 级灰度的单色 STN LCD 以及 256 色的彩色 DSTN LCD。并且 LCD 的刷新率可通过设置 CPU 内部的寄存器来进行调节。S3C44B0X 的 LCD 控制器主要功能就是把在系统内存的视频数据传送到外部的 LCD 驱动器。S3C44B0X 的 LCD 控制器由以下各功能模块组成，如下图所示：



图表 7 S3C44B0X 的 LCD 控制器功能模块

以下对所涉及到的模块的功能做简单的介绍：

- REGBANK 模块：对 LCD 控制器的配置
- LCDCDMA 模块：自动的把帧缓冲中的视频数据传到 LCD 驱动器
- VIDPRCS 模块：把从 LCDCDMA 模块传过来的数据经过适当的格式转换后通过 VD[7:0]口发送到 LCD 驱动器。
- TIMEGEN 模块：由可编程逻辑组成，支持多种不同的 LCD 驱动器的时序和频率。这个模块产生 VFRAME, VLINE, VCLK, VM 等信号。

特殊寄存器

S3C44B0X 的 LCD 控制器中主要含有下列特殊寄存器：

LCDCON1：主要负责各模式设定，时钟寄存器等等；

LCDCON2：主要负责垂直，水平扫描参数的设定；

LCDCON3：测试模式的使能设定；

LCDSADDR1：帧缓冲开始地址 1；

LCDSADDR2：帧缓冲开始地址 2；

LCDSADDR3：虚拟屏幕地址设置；

REDLUT，GREENLUT，BLUELUT：红、绿、蓝三原色查找表寄存器。

3.2.2. FrameBuffer 介绍

编写 LCD 驱动，就会涉及到 FrameBuffer。FrameBuffer，帧缓冲，有时简称为 fbdrv，基于 fbdrv 的 console 也被称之为 fbcon。这是一种独立于硬件的抽象图形设备。帧缓冲设备对图象硬件设备提供了一种抽象化处理。它代表了一些视频硬件设备，允许应用软件通过定义明确的界面来访问图象硬件设备，因此软件无需了解任何涉及硬件底层驱动的东西(如硬件寄存器)。FrameBuffer 的优点在于其高度的可移植性、易使用性、稳定性。帧缓冲设备属于字符设备，采用“文件层-驱动层”的接口方式。在文件层次上，Linux 为其定义了 file_operations 的数据结构：

```
static struct file_operations fb_fops = {
  owner: THIS_MODULE,
  read: fb_read, /* 读操作 */
  write: fb_write, /* 写操作 */
  ioctl: fb_ioctl, /* 控制操作 */
  mmap: fb_mmap, /* 映射操作 */
  open: fb_open, /* 打开操作 */
  release: fb_release, /*释放操作*/
};
```

其中的成员函数都在文件 linux-2.4.x/driver/video/fbmem.c 中定义。

由于显示设备的特殊性，在驱动层的接口中不但要包含底层函数，还要有一些记录设备状态的数据。Linux 为帧缓冲设备定义的驱动层接口为 struct fb_info 结构，在 include/linux/fb.h 中定义。幸运的是，嵌入式系统要求的显示操作比较简单，只涉及到结构中少数几个成员。

3.2.3. LCD 驱动程序

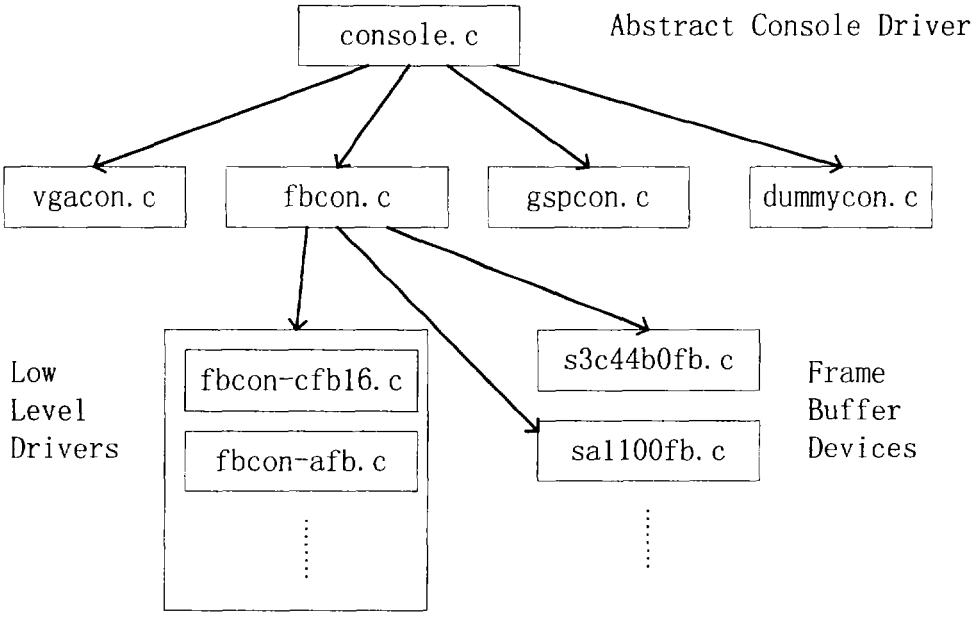
3.2.3.1. 相关文件

本系统要采用图形界面，所以 LCD 驱动实际上与帧缓冲的内容相关，主要涉

及到以下一些主要文件：

- fbcon.：　　通用的基于 framebuffer 的 console 的实现；
- fbcon-cfb16.c：　　底层的基于 framebuffer 的 console 的 16bpp 驱动的实现；
- fbmem.c：　　对 framebuffer 的通用操作接口；
- fbcmap.c：　　处理 color map；
- modedb.c：　　各显示 mode 的数据库以及操作；
- fonts.c
- font_acorn_8x8.c　　与上面的文件都是处理字体的；
- s3c44b0fb.c　　针对具体的显示器件的驱动程序，这个文件需要我们自己添加。

在帧缓冲设备控制台（console）驱动中，主要由三部分组成：一是 fbdev，控制了具体的硬件设备，二是 fbcon，是通用的帧缓冲设备控制台，三是 fbcon-*，底层的绘制程序的实现。其中有关的主要文件的相互关系如下图描述：



图表 8 各有关 console 的驱动文件之间的相互关系

3.2.3.2. 数据结构

LCD 驱动主要的数据结构有：fb_info、fb_fix_screeninfo、

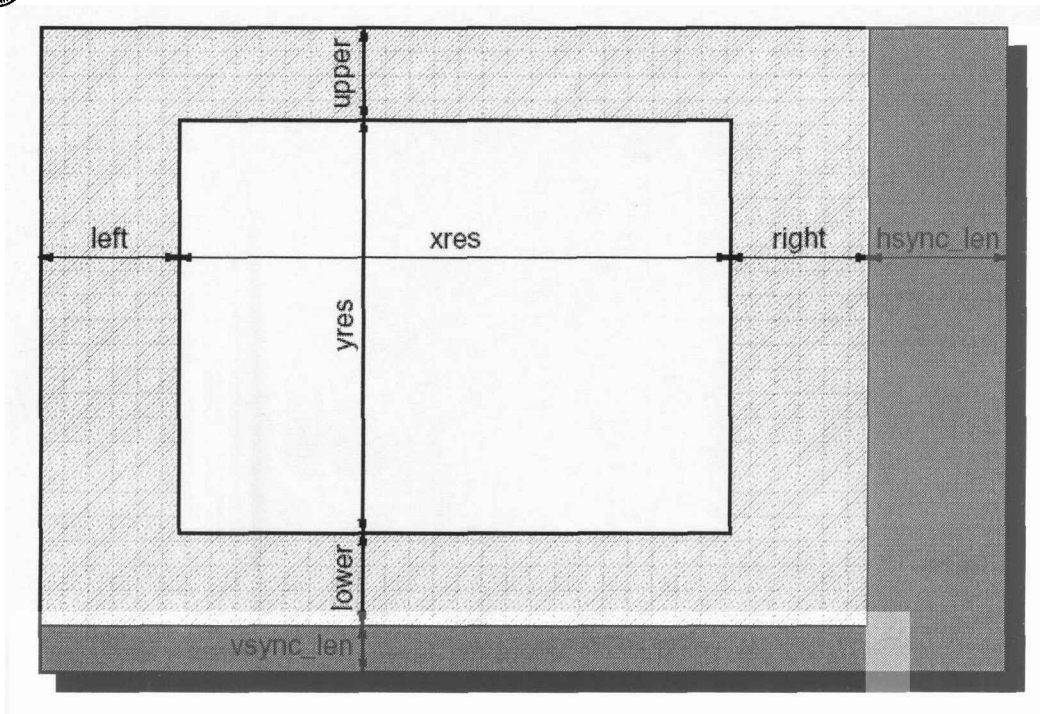
fb_var_screeninfo 以及针对特定的 LCD 控制器的数据结构，这里是 s3c44b0fb_info。

fb_info 中纪录了帧缓冲设备的全部信息，包括设备的设置参数，状态以及操作函数指针。每一个帧缓冲设备都必须对应一个 fb_info 结构。其中成员变量 modename 为设备名称，fontname 为显示字体，fbops 为指向底层操作的函数的指针，这些函数是需要驱动程序开发人员编写的。成员 fb_var_screeninfo 和 fb_fix_screeninfo 也是结构体。其中 fb_var_screeninfo 记录用户可修改的显示控制器参数，包括屏幕分辨率和每个像素点的比特数。fb_var_screeninfo 中的 xres 定义屏幕一行有多少个点，yres 定义屏幕一列有多少个点，bits_per_pixel 定义每个点用多少个字节表示。它的信息可以通过 ioctl 函数的 FBIOGET_VSCREENINFO 获得，可以通过 FBIOPUT_VSCREENINFO 更新。fb_fix_screeninfo 中记录用户不能修改的显示控制器的参数，如屏幕缓冲区的物理地址，长度。同样，它的信息可以通过 ioctl 的 FBIOGET_FSCREENINFO 获得。当对帧缓冲设备进行映射操作的时候，就是从 fb_fix_screeninfo 中取得缓冲区物理地址的。上面所说的数据成员都是需要在驱动程序中初始化和设置的。[57]

结构体 fb_var_screeninfo 中一些主要成员如下：

```
struct fb_var_screeninfo {
  __u32 xres, yres, xres_virtual, yres_virtual;
  __u32 xoffset, yoffset;
  __u32 bits_per_pixel;
  struct fb_bitfield red, green, blue, transp;
  __u32 pixclock;
  __u32 left_margin, right_margin, upper_margin, lower_margin;
  __u32 hsync_len, vsync_len;
};
```

其中一些参数的意义如下图所示：



图表 9 显示示意图

整个区域是一个虚拟的显示区域，图中 xres，yres 是指实际的显示区域。参数 xres 对应的是 LINEVAL＋1，参数 yres 对应的是 HOZVAL＋1。

3.2.3.3. 主要函数

添加文件 drivers/video/s3c44b0fb.c，其中主要操作为：

```
static struct fb_ops s3c44b0fb_ops = {
    owner:        THIS_MODULE,
    fb_get_fix:   s3c44b0fb_get_fix,
    fb_get_var:   s3c44b0fb_get_var,
    fb_set_var:   s3c44b0fb_set_var,
    fb_get_cmap:  s3c44b0fb_get_cmap,
    fb_set_cmap:  s3c44b0fb_set_cmap,
};
```

- **s3c44b0fb_init**

初始化函数：初始化函数首先初始化 LCD 控制器，通过写寄存器设置显示模式和显示颜色数，然后分配 LCD 显示缓冲区。在 Linux 可通过 kmalloc 函数分配

一片连续的空间。缓冲区通常分配在大容量的片外 SDRAM 中，起始地址保存在 LCD 控制器寄存器中。最后是初始化一个 fb_info 结构，填充其中的成员变量，并调用 register_framebuffer(&fb_info)，将 fb_info 登记入内核。

以下是 init 函数的简要分析：

```
int __init s3c44b0fb_init(void)
{
    struct s3c44b0fb_info *fbi;
    int ret = -ENOMEM;
fbi = s3c44b0fb_init_fbinfo();
        =>开辟 fbi 的数据结构的空间，初始化，设置 fb.fix，
fb.var（res, bpp 等等）的参数
    if (!fbi)
    goto failed;
ret = s3c44b0fb_map_video_memory(fbi);
        =>开辟 frame buffer 的内存空间，初始地址
为 fbi->fb.fix.smem_start，大小为 fbi->fb.fix.smem_len
    if (ret)
    goto failed;

s3c44b0_lcd_init();
        =>LCD 控制器的端口初始化
s3c44b0fb_set_var(&fbi->fb.var, -1, &fbi->fb);
        =>
ret = register_framebuffer(&fbi->fb);
        =>注册一个 frame buffer 设备
MOD_INC_USE_COUNT ;
    return 0;

failed:
    if (fbi)
    kfree(fbi);
    return ret;
```

```
}
```

fb_ops 结构中的函数都是用来设置或者获取 fb_info 结构中的成员变量的。当应用程序对设备文件进行 ioctl 操作时候会调用它们。

- **s3c44b0fb_get_fix**

应用程序传入的是 fb_fix_screeninfo 结构，在函数中对其成员变量赋值，主要是对固定值 smem_start(缓冲区起始地址)和 smem_len(缓冲区长度) 赋值，最终返回给应用程序。

```
static int
s3c44b0fb_get_fix(struct fb_fix_screeninfo *fix,
int con, struct fb_info *info)
{
    struct display *display = get_con_display(info, con);

    *fix = info->fix;

    fix->line_length = display->line_length;
    fix->visual      = display->visual;
    return 0;
}
```

- **s3c44b0fb_get_var**

这个函数主要是得到一些可变的参数值，如实际显示区域 xres、yres，虚拟显示区域 xres_virtual、yres_virtual 等。

```
static int
s3c44b0fb_get_var(struct fb_var_screeninfo *var, int con,
struct fb_info *info)
{
    *var = *get_con_var(info, con);
    return 0;
}
```

- **s3c44b0fb_set_var**

s3c44b0fb_set_var()函数的传入参数是 fb_var_screeninfo，函数中需要对 xres，yres,和 bits_per_pixel 赋值。

3.2.4. 对帧缓冲区的操作

通过/dev/fb，应用程序的操作主要有这几种：

1. 读/写(read/write)/dev/fb：相当于读/写屏幕缓冲区。例如用 cp /dev/fb0 tmp 命令可将当前屏幕的内容拷贝到一个文件中，而命令 cp tmp > /dev/fb0 则将图形文件 tmp 显示在显示设备上。由于 uClinux 可以直接访问物理内存（不象标准 Linux 工作在保护模式，每个应用程序都有自己的虚拟地址空间，在应用程序中是不能直接访问物理缓冲区地址的），对于帧缓冲设备，用户可以直接访问屏幕缓冲区的物理地址。
2. I/O 控制：对于帧缓冲设备，对设备文件的 ioctl 操作可读取/设置显示设备及屏幕的参数，如分辨率，显示颜色数，屏幕大小等等。ioctl 的操作是由底层的驱动程序来完成的。

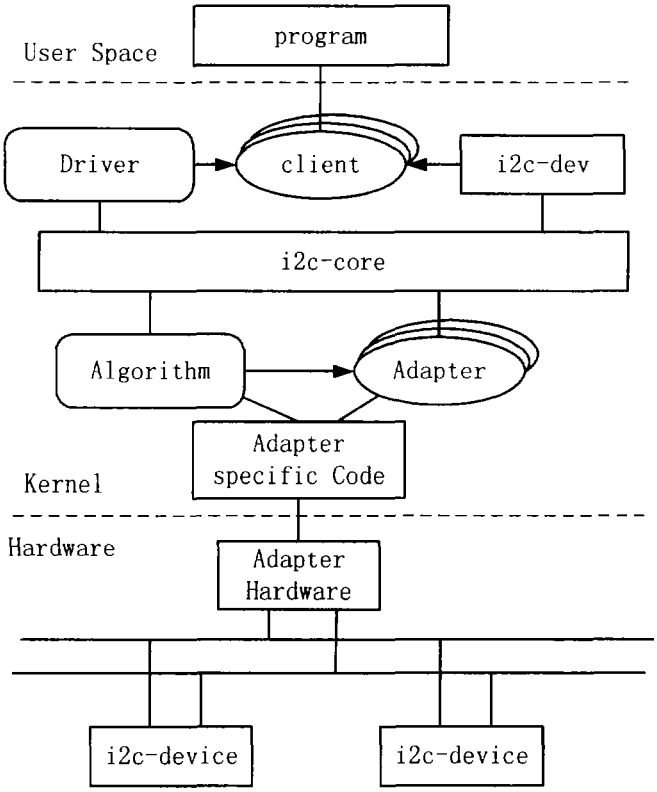
在应用程序中，操作/dev/fb 的一般步骤如下：

1. 打开/dev/fb 设备文件。
2. 用 ioctl 操作取得当前显示屏幕的参数，如屏幕分辨率，每个像素点的比特数。根据屏幕参数可计算屏幕缓冲区的大小。
3. 读写屏幕缓冲区，进行绘图和图片显示。

3.3. IIC 驱动程序

IIC（I^2C）总线[55]是一种具有多端控制能力的双线双向串行数据总线，能连接的各种集成电路和功能模块。它是一种用于 IC 器件之间连接的二线制总线。它通过 SDA（串行数据线）及 SCL（串行时钟线）两根线在连到总线上的器件之间传送信息，并根据地址识别每个器件，无论是微控制器、LCD 驱动器、存储器

或键盘接口等。I²C 总线最显著的特点是规范的完整性、结构的独立性和用户使用时的简单化。I²C 总线有严格的规范，如接口的电气特性、信号时序、信号传输的定义、总线状态设置、总线管理规则及总线状态处理等。



图表 10 Linux 内核 I²C 总线驱动体系结构

如上图的总线驱动体系结构所示，client 对应一个具体的 I²C 总线设备，它通过 Driver 绑定到具体的 adapter 上。i2c-dev 为设备提供了/dev 接口。Client 的所有的操作包括初始化，读，写，控制都需要调用 i2c-core 中导出的内核函数来完成。i2c-core 的内核函数再依据具体的 adapter 调用其使用的 Algorithm 中的函数。Algorithm 中函数再调用 adapter 硬件相关的函数来完成操作[56]。

3.3.1. 数据结构

在 I²C 总线驱动体系结构中，主要有如下几个数据结构：

1. i2c_client：表示一个连接到 I²C 总线上的设备。包含指向所使用的 adapter

的指针和指向从设备的 I²C 总线地址。

2. i2c_driver：处理 I²C 总线适配器上的一个或多个设备的驱动器。
3. i2c_algorithm：I²C 总线传输数据时使用的算法。它输出使用这个算法的 adapter 模块要调用的内核函数 i2c_*_add_bus 和 i2c_*_del_bus。这个结构中的 master_xfer 函数指针指向传输数据的函数。对 I²C 总线的读和写都是通过这个指针所指向的函数来实现的。
4. i2c_adapter：用来表示一个 I²C 总线适配器。

另外在 i2c-core 中两个重要的数组：

1. drivers 数组：包含已经初始化了的 client 的 drivers；
2. adapters 数组：包含已经初始化了的 adapters。

3.3.2. 相关文件

主要分成三类文件，一是基本的核心模块，主要有：

1. i2c-core.c：I²C 总线驱动体系的核心文件，这个文件定义了许多其他文件必须调用的函数。所以当其它 I²C 的模块被调用时，这个模块相应的必须被调用；
2. i2c-dev.c：这个文件定义了对 I²C 总线的/dev 接口，它通过这个接口可以从用户空间访问各种设备；
3. i2c-proc.c：设备驱动（client）的/proc 接口。

第二类是各种特定算法的文件，如 i2c-algo-pcf.c 等等；

第三类是支持各种特定的适配器的文件，如 i2c-adap-pxa.c 等等。

3.3.3. 主要模块

i2c_algo_*模块

这是一个中间模块，主要定义了在 I²C 总线上传输数据时采用的算法。在这个模块中，需要初始化一个系统的 i2c_algorithm 数据结构的 i2c_algo 和定义

传输数据时使用的函数，对于我们特定的 i2c_algo_s3c44b0 模块，主要的函数有：

1. s3c44b0_xfer：采用用 i2c_s3c44b0_algo 的 master_xfer 指针指向这个函数。所有的数据的读写操作都会调用这个函数。这个函数再根据具体的情况调用 s3c44b0_sendbytes 和 s3c44b0_readbytes 来把操作传递到下层函数。
2. s3c44b0_sendbytes：把数据发送到 I^2C 总线上。
3. s3c44b0_readbytes：从 I^2C 总线上获得数据。
4. s3c44b0_control：这个函数实现了 ioctl 控制，它用 i2c_s3c44b0_algo 的 algo_control 指针指向这个函数。
5. i2c_s3c44b0_add_bus：每个使用这个算法的 adapter 初始化时调用该函数。
6. i2c_s3c44b0_del_bus：与上一个函数相对应。

i2c_adap_*模块

这个模块具体实现了在 S3C44B0X 的 I^2C 总线上如何传输数据，即如何向 I^2C 总线输出和从 I^2C 总线读一个字节。在我们的模块 i2c_adap_s3c44b0 中我们要初始化一个 i2c_adapter 数据结构和一个 i2c_algo_s3c44b0_data 数据结构，所以这两个结构中的函数指针成员都需要被定义。

3.4. 其它驱动程序

本系统还需要实现特定外围器件的驱动程序，比如专用打印机驱动程序，Flash 卡（这里我们采用的是三星公司的 SM 卡）的驱动程序，IIS（音频输出，对于我们这个系统来说比较次要）驱动程序等等。其中，打印机驱动程序由于没有采用控制芯片，所以么完全通过 CPU 来控制时序等。另外，对掉电中断输入我们也把它作为一个特殊的设备，我们取名为 poff 设备，并为它编写特定的设备

驱动程序。在这个特殊的设备中，我们额外的工作就是需要在设备初始化（init）的时候调用一个 request_irq 函数来注册中断，同时也需要有它的中断处理函数，这里我们设定为 poff_interrupt，这里我们先给出 request_irq，中断处理函数在下文我们会有所叙述。

```
request_irq(
  dev->irq,  /* The number of the poff IRQ on ARM */
  poff_interrupt,  /* 要调用的中断处理程序 */
  SA_INTERRUPT,
  /* SA_INTERRUPT 意思是这是一个紧急中断.
   */
  "power_off_irq_handler", NULL);
```

第四章 ZTax的数据保护方案设计

4.1. 问题的提出及解决方案

我们在第一章综述中就已经提到了税控收款机系统应解决的关键问题，其中有两点：一是税控收款机要求机内程序不能随便更改；二是要求能够记录保存数据，增强数据访问的安全性，并保证事务的完整性。本章就这些问题做出详细的解决方案。

对于第一个问题，程序的不可随意更改，可以采用加载romfs的办法来处理。romfs是只读的一个文件系统，用户不能对它进行任何的写入操作，因此本身就已经达到了这个目标。

对数据的保存我们可以通过文件系统来解决。uClinux支持多种文件系统如ext2、romfs、fat、jffs2、ISO 9660、proc等，其默认根文件系统为romfs，这种文件系统相对于一般的ext2文件系统要求更少的空间。但romfs文件系统不支持动态擦写保存，对于系统需要动态保存的数据一般来说有两种方式可以采用：

1. 采用虚拟ram盘。即在内存中开辟一块RAM，在其上面挂载文件系统，把它当作一个ram盘使用。其最大的缺点是系统掉电后ram盘的内容全部丢失，不能永久保存数据。
2. 采用Flash卡，并在Flash上挂接可读写的Ext2、JFFS2或YAFFS文件系统的方法进行处理。

对于第二种方式，由于一般Flash卡有寿命限制，一般只能进行10万次左右的擦写，当然好点的Flash也有可以擦写100万次的。因此我们不能频繁的对Flash卡进行动态数据保存。

不过，我们是否可以把上述两种方法结合起来呢？答案是肯定的。采用RAM临时存放税控和交易数据。在数据达一定程度（如16KB）后，将数据写入Flash中，以减少Flash的擦写次数。现在的问题就转化为在系统掉电后如何把可能还没有保存的数据（仅只有不到16KB）转入到Flash卡上去。这将在下文中详细

论述。

事务完整性要求，首先当然是要保证数据的一致性，这个我们可以通过采用一个日志文件系统解决这一问题。另外事务完整性要求体现在税控收款机中的一种比较特殊的情况就是打印到一半突然掉电，由于发票不能随便作废，在重新上电以后仍旧要打印完这张发票。怎么解决以上提及的问题呢？一种好的方式也是“日志”技术，它借用了如 Oracle 等大型数据库的思想，能够更好地保证数据和操作的完整性、安全性，实现灾难性恢复。

4.2. FLASH 卡

FLASH 是一种可擦除的存储器，具有低成本，高可靠性和高存储密度的特征，因此被广泛应用在以嵌入式系统为基础的各类产品中，其特点是只能以扇区为单位来擦除(扇区大小和芯片的型号有关)。长期以来 FLASH 一直被用来存放可执行代码、变量和其他暂态数据，随着其应用的日趋广泛和深入，如何在其上建立一种稳定可靠、便于使用的、针对 FLASH 特点的文件系统便被提上议事日程。

在嵌入式 Linux 中，FLASH 设备有两种作为文件系统的使用方式，其一是在 FLASH 上构造一个伪文件系统来模拟标准的块设备，然后在块设备上挂接普通的文件系统，另一种方式是直接在 FLASH 设备上使用 JFFS2、YAFFS 等文件系统。

FLASH 上的数据保存

对于简单的数据，小的配置文件等，而且不是非常频繁的（例如一分钟写 10 次）写入的，可以直接自己在 FLASH 的空余处（例如第二片 FLASH 上）划出一块区域，以自己定义的方式写入，并在板子启动时自动读出。这样作最大的好处是用户的控制程度最大，形式最灵活，保存的可以是自定义的数据，可以不是文件的形式。

对于比较多的配置文件，一般的都先写在 RAM 盘中，然后选择保存，一次性写入 FLASH 的某几个扇区。这时就可使用 flatfsd 软件，它的使用需要内核的配合支持，即要在 blkmem.c 中为其指定保存数据的几个扇区的起始/结束地址。

另外还有一种方法是采用可读写的文件系统，或者干脆就用这个文件系统取

代 ROMFS 作根文件系统。这样板子的目录就是可写的，就像硬盘一样，不需要额外的工具来负责将数据写入 FLASH。

NAND 和 NOR Flash

NOR 和 NAND 是现在市场上两种主要的非易失闪存技术。Intel 于 1988 年首先开发出 NOR flash 技术，彻底改变了原先由 EPROM 和 EEPROM 一统天下的局面。紧接着，1989 年，东芝公司发表了 NAND flash 结构，强调降低每比特的成本，更高的性能，并且像磁盘一样可以通过接口轻松升级。大多数情况下闪存只是用来存储少量的代码，这时 NOR 闪存更适合一些。而 NAND 则是高数据存储密度的理想解决方案。

NOR 和 NAND Flash 相比，NOR 的特点是芯片内执行(XIP, eXecute In Place)，这样应用程序可以直接在 Flash 闪存内运行，不必再把代码读到系统 RAM 中。NOR 的传输效率很高，在 1～4MB 的小容量时具有很高的成本效益，但是很低的写入和擦除速度大大影响了它的性能。

NAND 结构能提供极高的单元密度，可以达到高存储密度，并且写入和擦除的速度也很快。应用 NAND 的困难在于 flash 的管理和需要特殊的系统接口。下面我们对 NAND Flash 和 NOR Flash 做各方面的比较。

在性能方面，Flash 闪存是非易失存储器，可以以块为单位进行擦写和再编程。任何 Flash 器件的写入操作只能在空的或已擦除的单元内进行，所以大多数情况下，在进行写入操作之前必须先执行擦除操作。NAND 器件执行擦除操作是十分简单的，而 NOR 则要求在进行擦除前先要将目标块内所有的位都写为 0。

在接口方面，NOR Flash 带有 SRAM 接口，有足够的地址引脚来寻址，可以很容易地存取其内部的每一个字节。NAND 器件使用复杂的 I/O 口来串行地存取数据，各个产品或厂商的方法可能各不相同。8 个引脚用来传送控制、地址和数据信息。NAND 读和写操作是以块为单位的，这一点有点像硬盘管理此类操作，很自然地，基于 NAND 的存储器就可以取代硬盘或其他块设备。特别的是，它的块或者页字节数为（512+16）字节，512 字节用来保存数据，而 16 字节用来作标志位。

在容量方面，NAND Flash 的单元尺寸几乎是 NOR 器件的一半，由于生产过

程更为简单，NAND 结构可以在给定的模具尺寸内提供更高的容量，也就相应地降低了价格。NOR Flash 占据了容量为 1～16MB 闪存市场的大部分，而 NAND Flash 只是用在 8～128MB 的产品当中，这也说明 NOR 主要应用在代码存储介质中，NAND 适合于数据存储。

在耐用性方面，在 NAND 闪存中每个块的最大擦写次数是一百万次，而 NOR 的擦写次数是十万次。NAND 存储器除了具有 10 比 1 的块擦除周期优势，典型的 NAND 块尺寸要比 NOR 器件小 8 倍，每个 NAND 存储器块在给定的时间内的删除次数要少一些。

在易用性方面，可以非常直接地使用基于 NOR 的闪存，可以像其他存储器那样连接，并可以在上面直接运行代码。由于需要 I/O 接口，NAND 要复杂得多。各种 NAND 器件的存取方法因厂家而异。在使用 NAND 器件时，必须先写入驱动程序，才能继续执行其他操作。向 NAND 器件写入信息需要相当的技巧，因为设计师绝不能向坏块写入，这就意味着在 NAND 器件上自始至终都必须进行虚拟映射。在 NOR 器件上运行代码不需要任何的软件支持，在 NAND 器件上进行同样操作时，通常需要驱动程序，也就是内存技术驱动程序(MTD)，NAND 和 NOR 器件在进行写入和擦除操作时都需要 MTD。

表格 1 NOR 和 NAND 在一些方面的比较

比较项目	NOR	NAND
访问方式	线形访问	页访问
价格	$2/MB	$0.5/MB
容量	小，最多 8M	大，可以达到 128M
擦除速度	慢	快
设计目的	作为 ROM 的替换	磁盘
写入方式	字节写入	按页写入

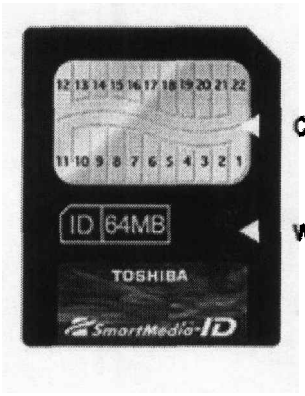
基于以上理由，根据我们的实际需要，我们选择了 NAND Flash。

SM 卡

SmartMedia 卡是由东芝公司 Toshiba America Electronic Components（TAEC）在 1995 年 11 月发布的 Flash Memory 存贮卡，三星公司 Samsung Electronic Co.在 1996 年购买了生产和销售许可，这两家公司成为主要的 SmartMedia 厂商。

SmartMedia 采用了 NAND 型 Flash Memory，使得 SmartMedia 具有比较高的擦写性能。它的存贮卡上只有 Flash Memory 模块和接口，而并没有包括控制芯片，所以使用 SmartMedia 的设备必须自己装置控制机构。但这却带来了兼容性问题：由于各家的控制不尽相同，所以存在着格式互不兼容的现象。另外旧的设备是根据以前的 SmartMedia 规范来设计的，所以当 SmartMedia 推出新的升级产品时，规范往往会改变，以前的设备很多就无法使用。

SmartMedia 使用了物理格式和逻辑格式，其中物理格式确保不同设备模型之间的兼容性，是系统和控制厂商所必须遵从的。物理格式基于 ATA 和 DOS 文件的 FAT 标准，以使得不同的系统间交换数据容易一些，但物理格式的配置会在页面大小上有所不同，这取决于内存的类型和存贮卡的容量。至于逻辑格式则采用了 DOS-FAT 格式，就是柱面磁头扇区参数、主扇区和分区。



图表 11 SM 卡

4.3. 文件系统的选择

上面我们已经提到可以采用在 Flash 卡上采用文件系统并使用日志的方法。我们发现，JFFS2、YAFFS 文件系统正是基于“日志”的文件系统，而 Ext2 文件系统为非日志文件系统，首先被我们排除。

下面我们比较一下 JFFS2[52]和 YAFFS 文件系统。

表格 2 JFFS2 与 YAFFS 的比较

比较项目	JFFS2	YAFFS
占用额外空间	16 bytes per node(4M for 128M)	2 bytes per page(512k for 128M)
加载时间	25s	3s
是否压缩	是	否
复杂度	复杂	简单
支持的操作系统	Linux，ecos	多个 OS，移植简单

我们可以看到，YAFFS 在各方面占有明显的优势，因此我们选择了 YAFFS 文件系统。不过它是不经过压缩的，也由于这个原因，我们在系统中没有把 YAFFS 就作为根文件系统，而只是作为一个分区，用来存放用户数据。这样，kernel 以及固定应用程序可以放在 romfs（也是经过压缩的），可以来减少空间，也可以防止用户的修改。

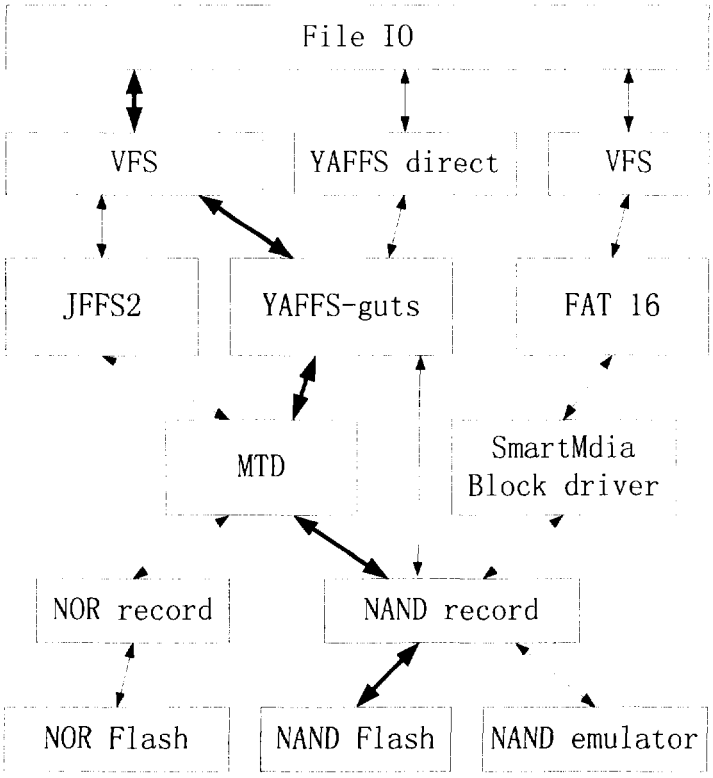
由于 uClinux 不直接支持 YAFFS，我们需要对文件系统进行移植。

4.4. YAFFS 文件系统

YAFFS 是 Yet Another Flash Filing System 的简写[32]，YAFFS 文件系统跟 JFFS2 文件系统一样，也是一种基于 Flash 的日志文件系统，能在掉电时自动提供可靠的数据记录、恢复，防止自身文件系统的崩溃，并能对坏块进行管理。它用独立的日志文件跟踪磁盘内容的变化。举例来说：当驱动程序需要写存储器中某一块时，它修改的是存放于文件日志中的一块镜像；只有当日志中的镜像复

制到日志文件系统中后，数据才真实地写到了那一块上。这种技术允许撤消对文件系统做的改变(只要把日志中的数据覆盖)，或者在发生崩溃时恢复数据(只要把镜像拷贝到文件系统中)。日志文件系统提供了更好的性能表现(数据是并行地写入进日志，然后再逐步写入文件系统)，和更好的崩溃恢复性能。它在设计时充分考虑了 FLASH 的读写特性，专门提供了 NAND Flash 芯片的优化技术。当然它对 NOR Flash 和 RAM 也提供了良好的支持。另外它也确保在读取文件时，如果系统突然掉电，其文件的可靠性不受到影响。当然，YAFFS 对 Flash 的读写采用 ECC 校验，增强数据访问的安全性。

YAFFS 是建立在 MTD(Memory technology device)基础之上的文件系统。下图是文件系统的整个体系结构：

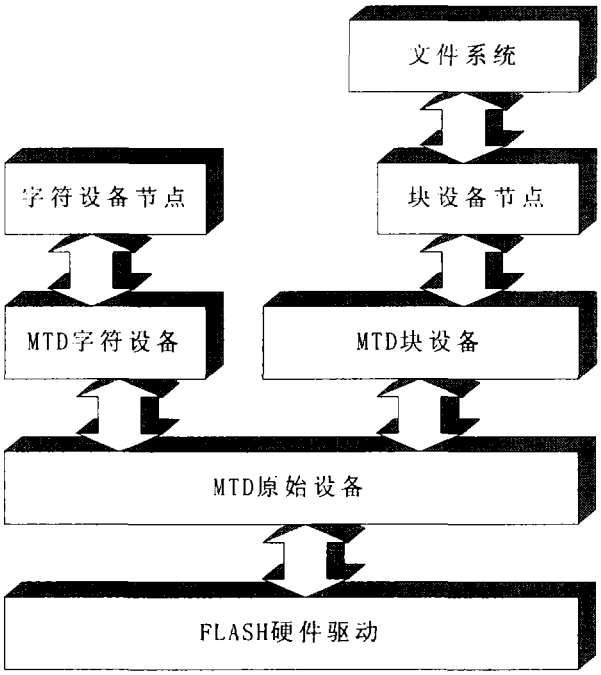


图表 12 文件系统的调用关系

4.4.1. MTD 设备

MTD (Memory Technology Devices) 是用于访问 Flash 设备的 Linux 子系统，其主要目的是使 FLASH 设备的驱动程序更加简单。MTD 在硬件和上层之间提供一个抽象的接口，MTD 可以理解为 FLASH 的设备驱动程序，其主要向上提供两个接

口，MTD 字符设备和 MTD 块设备。通过这两个接口，就可以像读写普通文件一样对 FLASH 设备进行读写操作。经过简单的配置后，MTD 在系统启动以后可以自动识别支持 CFI 或 JEDEC 接口的 FLASH 芯片，并自动采用适当的命令参数对 FLASH 进行读写或擦除。MTD 设备分为四层（从设备节点直到底层硬件驱动），这四层从上到下依次是：设备节点、MTD 设备层、MTD 原始设备层和硬件驱动层，如下图所示：



图表 13 MTD 设备

1、Flash 硬件驱动层：硬件驱动层负责在 init 时驱动 Flash 硬件。MTD 支持两种类型的 Flash：NOR 型和 NAND 型，为目前市场上两种主要的非易失闪存技术。Linux MTD 设备的 NOR Flash 芯片驱动遵循 CFI 接口标准，其驱动程序位于 drivers/mtd/chips 子目录下，NAND 型 Flash 的驱动程序则位于 /drivers/mtd/nand 子目录下。

2、MTD 原始设备：原始设备层有两部分组成，一部分是 MTD 原始设备的通用代码，另一部分是各个特定的 Flash 的数据，例如分区。用于描述 MTD 原始设备的数据结构是 mtd_info，这其中定义了大量的关于 MTD 的数据和操作函数。mtd_table（mtdcore.c）则是所有 MTD 原始设备的列表，mtd_part（mtd_part.c）是用于表示 MTD 原始设备分区的结构，其中包含了 mtd_info，因为每一个分区

都是被看成一个 MTD 原始设备加在 mtd_table 中的，mtd_part.mtd_info 中的大部分数据都从该分区的主分区 mtd_part->master 中获得。在 drivers/mtd/maps/ 子目录下存放的是特定的 flash 的数据，每一个文件都描述了一块板子上的 flash。其中调用 add_mtd_device()、del_mtd_device()建立/删除 mtd_info 结构并将其加入/删除 mtd_table（或者调用 add_mtd_partition()、del_mtd_partition()（mtdpart.c）建立/删除 mtd_part 结构并将 mtd_part.mtd_info 加入/删除 mtd_table 中）。

3、MTD 设备层：基于 MTD 原始设备，linux 系统可以定义出 MTD 的块设备 MTD_BLOCK（主设备号 31）和字符设备 MTD_CHAR（设备号 90）。MTD_CHAR 提供对闪存的原始字符访问，而 MTD_BLOCK 将闪存设计为可以在上面创建文件系统的常规块设备（如 IDE 磁盘）。与 MTD_CHAR 关联的设备是 /dev/mtd0、mtd1、mtd2 等等，而与 MTD_BLOCK 关联的设备是 /dev/mtdblock0、mtdblock1 mtdblock2 等等。由于 MTD_BLOCK 设备提供像块设备那样的模拟，通常更可取的是在这个模拟基础上创建像 JFFS2 和 YAFFS 那样的文件系统。MTD 字符设备的定义在 mtdchar.c 中实现，通过注册一系列 file operation 函数(lseek、open、close、read、write)。MTD 块设备则是定义了一个描述 MTD 块设备的结构 mtdblk_dev，并声明了一个名为 mtdblks 的指针数组，这数组中的每一个 mtdblk_dev 和 mtd_table 中的每一个 mtd_info 一一对应。

4、设备节点：通过 mknod 在/dev 子目录下建立 MTD 字符设备节点（主设备号为 90）和 MTD 块设备节点（主设备号为 31），通过访问此设备节点即可访问 MTD 字符设备和块设备。

5、文件系统：内核启动后，通过 mount 命令可以将 flash 中的其余分区作为文件系统挂载到 mountpoint 上。

为了访问特定的闪存设备并将文件系统置于其上，需要将 MTD 子系统编译到内核中。这包括选择适当的 MTD 硬件和用户模块。当前，MTD 子系统支持为数众多的闪存设备——并且有越来越多的驱动程序正被添加进来以用于不同的闪存芯片。

为了进行这个操作，可能需要创建分区表将闪存设备分拆到引导装载程序、内核和文件系统中。Linux 中 MTD 子系统的主要目标是在系统的硬件驱动程序

和上层，或用户模块之间提供通用接口。硬件驱动程序不需要知道像 YAFFS 和 JFFS2 那样的用户模块使用的方法。所有它们真正需要提供的就是一组对底层闪存系统进行 read、 write 和 erase 操作的简单例程。

MTD 设备节点的定义

当 YAFFS 文件系统作为根文件系统时 MTD 块设备的主设备号被分配为 31，MTD 字符设备的主设备号统一为 90。修改 dist/vender/ 中所使用的目标机类型的 Makefile 文件中的 DEVICES 目标，首选删除 DEVICES 目标下的所有主设备号为 31 的设备节点(主要为原 romfs 的设备节点)，按如下方式增加 DEVICES 目标：

```
mtd0,c,90,0  mtd1,c,90,1  mtd2,c,90,2 mtd3,c,90,3
mtdblock0,b,31,0  mtdblock1,b,31,1  mtdblock2,b,31,2 mtdblock3,b,31,3
```

具体的设备节点数目根据 MTD 设备的分区数而定。

由于 MTD 提供块设备的接口，通常在此基础上创建 YAFFS 文件系统。为了进行这个操作，需要创建分区表将 Flash 设备分拆为引导装载程序段、内核段和文件系统段（根文件系统和用户分区）。一般来说，描述 Flash 硬件设备特征的代码位于 linux-2.4.x/drivers/mtd/maps 目录下，可以根据具体的 Flash 和挂载方式选择合适的文件加以修改，以适应自己的要求。不过我们为简便起见，把这些代码都放到了 linux-2.4.x/drivers/mtd/smc_s3c44b0.c 的 SM 卡驱动程序文件中。

在这个驱动中，最关键的就是 partition_info 的定义根据系统的要求来建立适当的 Flash 分区，我们的分区是这样的：

```
static struct mtd_partition partition_info[] = {
    {
        name:       "bootloader",
        size:       256*1024,
        offset:     0
    }, {
        name:       "kernel",
        size:       768*1024,
        offset:     256*1024
    }, {
```

```
            name:       "root",
            size:       15*1024*1024,
            offset:     1024*1024,
        }, {
            name:       "yaffs filesystem",
            size:       16*1024*1024,
            offset:     16*1024*1024
        }
    };
```

此分区使用的 Flash 容量为 32MByte，分成四个部分，分别为 bootloader，kernel、root 和“yaffs filesystem”，这种情况是 yaffs 文件系统不是作为根文件系统，分别对应于前面所提及的 mtd0～mtd3、mtdblock0～mtdblock3。我们把系统程序，应用程序都放到不可写的 root 区，制作成 romfs 文件系统，这样可以防止程序被恶意的或无意的修改和删除。而第四个分区“yaffs filesystem”才是我们真正关注的，我们可以用来存放记录的数据等等。用户可以根据自己的要求适当增减 Flash 各个分区的大小。

要检验分区状况，在内核重新编译后，系统加载新的内核，用 cat /proc/mtd 查看，可以显示如下信息：

```
dev:    size   erasesize  name
mtd0: 00040000 00004000 "bootloader"
mtd1: 000c0000 00004000 "kernel"
mtd2: 00f00000 00004000 "root"
mtd3: 01000000 00004000 "yaffs filesystem"
```

4.4.2. YAFFS 文件系统的分析

4.4.2.1. 数据结构

所有的 YAFFS 涉及到的数据结构都在 yaffs_guts.h 文件中定义。主要数据结构有：yaffs_Object，yaffs_Tnode 以及 yaffs_Device。Tnode 和 Object 按组分配以减少总得内存分配和释放。已经被释放的 Tnode 和 Object 保存在一个空

闲链表上并被重新使用。

yaffs_Object

yaffs_Object 可以是一个文件，目录，软链接或者硬链接。yaffs_Object 知道他们在 NAND 中对应的 yaffs_ObjectHeader 以及这个对象的数据。yaffs_Object 把下列的对象都绑定到一起：

- parent 域：指向这个目录结构里的 yaffs_Object 的父指针。
- siblings 域:这个域把同一个目录里的 yaffs_Object 链起来成一个链表。
- children 域：如果对象是个目录，那么 children 域保存了这个目录里的链表的头指针。

yaffs_Object 定义如下：

```
struct yaffs_ObjectStruct
{
__u8 fake:1; // A fake object has no presence on NAND.
__u8 renameAllowed:1; // Are we allowed to rename it?
__u8 unlinkAllowed:1; // Are we allowed to unlink it?
__u8 dirty:1; // the object needs to be written to flash
__u8 valid:1; // When the file system is being loaded up, this
// object might be created before the data
// is available (ie. file data records appear before the header).
__u8 serial; // serial number of chunk in NAND. Store here so we don't have to
// read back the old one to update.
__u16 sum; // sum of the name to speed searching
struct yaffs_DeviceStruct *myDev; // the device I'm on
struct list_head hashLink; // list of objects in this hash bucket

struct list_head hardLinks; // all the equivalent hard linked objects
// live on this list
// directory structure stuff
struct yaffs_ObjectStruct *parent; //my parent directory
```

```
struct list_head siblings; // siblings in a directory
// also used for linking up the free list
// Where's my data in NAND?
int chunkId; // where it lives
__u32 objectId; // the object id value
__u32 st_mode; // protection
__u32 st_uid; // user ID of owner
__u32 st_gid; // group ID of owner
__u32 st_atime; // time of last access
__u32 st_mtime; // time of last modification
__u32 st_ctime; // time of last change

yaffs_ObjectType variantType;
yaffs_ObjectVariant variant;
};
```

yaffs_Object 的其他域的含义如下：

fake, renameAllowed, unlinkAllowed 域：这三个域是在处理“fake”对象的时候特殊的标志符，这个对象存在于文件系统中但不存在于 NAND Flash 中。目前，比较典型的有 root 对象和 lost+found 目录。他们都不能被重命名，也不能被删除。

dirty 域：表示对象的内容已经改变，Flash 中必须要重写一个新的 yaffs_ObjectHeader。

valid 域：表示对象已经被装载。 这个域只有在我们知道对象的存在然后在扫描过程中使用，而且，必须建立对象的头部信息。

hashlink 域：hashlink 是在同一个散列表中的对象链表。

File 域：保存了文件大小和最高层的指向这个文件的 tnode 树的指针。

Directorie 域：保存了子对象的链表。

Symlink 域：保存了指向字符串的别名的指针。

Hardlink 域：保存了区别相同对象的信息。

yaffs_Tnode

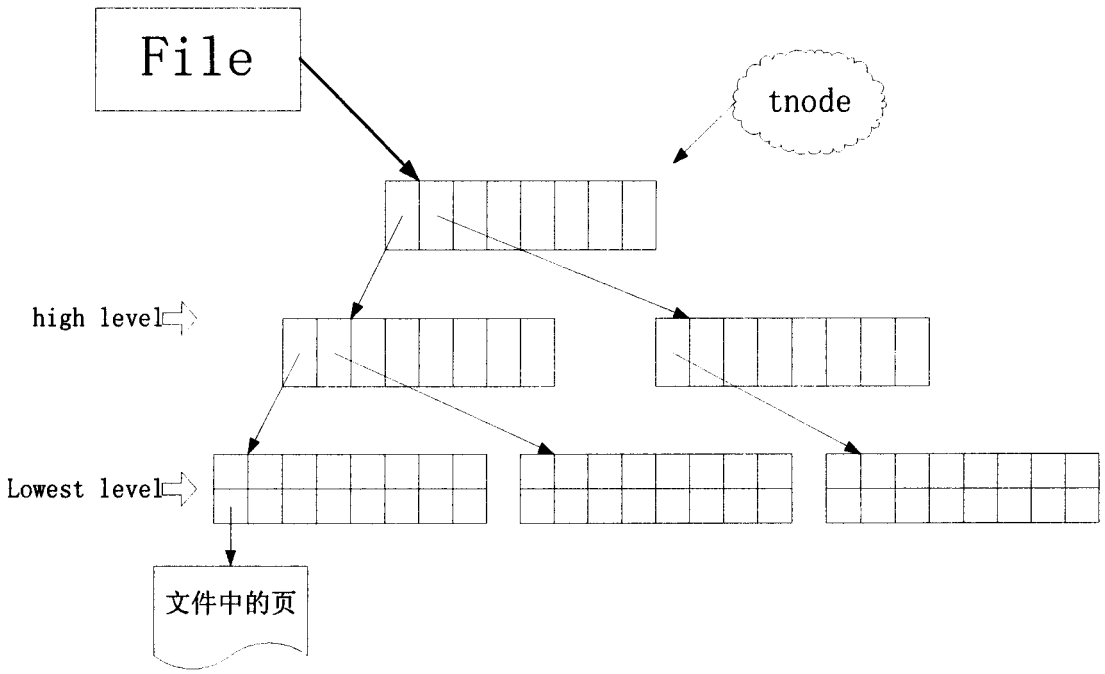
yaffs_Tnode 组成了一个树的数据结构，这样就能在一个文件中快速搜索到所需的数据块。随着文件的增大，树的层数也相应的增加。 Tnode 这个结构有固定的大小，32 个字节。最下面一层由 16 个 2 字节的入口组成，每个入口给出了用来查找块 ID 的索引。其他几层由 8 个 4 字节的指针组成，这些指针指向了树中的更底层的 Tnode。

yaffs_Device

yaffs_Device 保存了 NAND 设备的上下文，在某种程度上与 VFS 的超级块有点相似。它包含了用来访问 mtd 的数据，以及访问 NAND 数据得函数指针。

4.4.2.2. 文件组织结构

从上文得 yaffs_Tnode 的数据结构我们可以看出，最下面一层由 16 个 2 字节的入口组成，每个入口给出了用来查找物理页面的索引。其他几层由 8 个 4 字节的指针组成，这些指针指向了树中的更底层的 Tnode。如下图所示：



图表 14 文件组织结构图

当文件刚被分配到物理页面中的时候，它先分配一个 tnode，随着文件得逐渐增大，分配的页面页越来越多，到了一个 tnode 已经不能容纳下这些分配的页面的时候，然后就再分配一个 tnode，另外再加一个内部得 tnode，这个 tnode 里得指针指向上面两个 tnode。随着文件越来越大，底层和上层的 tnode 也就逐步多了起来。

如果要遍历整个 tnode 树来查找文件中一个特定的物理页面页比较简单：每个上层的 tnode 使用 3 个 bit，而最底层的 tnode 使用 4 个 bit。例如，要寻找页面 0x235，也就是从文件得 0x46800 开始，可以通过以下步骤寻找：

0x235 二进制表示为 0000001000110101，可以分解为 000，000，100，011，0101，如下表所示：

表格 3 查找页面的计算方法

层	位	选择值
>=3	>=10	0
2	9~7	4 (100)
1	6~4	3 (011)
0	3~0	5 (0101)

于是，按照最顶层开始，选择第 0 个 tnode，一直到第 3 层，第 2 层选择第 4 个 tnode，第一层选第 3 个，到最底层选第 5 个即是要找的页面。

每个 tnode 占用 32 个字节，每个文件至少有一个最底层的 tnode，如果有一个 16MB 的文件系统，那么至少需要 16M/512/16 个底层的 tnode（假定 512 字节为一个物理页面）。

4.4.2.3. 函数接口

所有访问 NAND Flash 的操作都是通过下列 4 个函数进行的：

```
int WriteChunkToNAND(struct yaffs_DeviceStruct *dev, int chunkInNAND,
                     const __u8 *data, yaffs_Spare *spare)
int ReadChunkFromNAND(struct yaffs_DeviceStruct *dev, int chunkInNAND,
```

```
                    __u8 *data, yaffs_Spare *spare);
int EraseBlockInNAND(struct yaffs_DeviceStruct *dev, int blockInNAND);
int InitialiseNAND(struct yaffs_DeviceStruct *dev).
```

而对上层应用的文件系统，YAFFS 提供了如下接口：

```
int yaffs_open(const char *path, int oflag, int mode) ;
Supported flags O_CREAT, O_EXCL, O_TRUNC, O_APPEND, O_RDONLY, O_RDWR, O_WRONLY.
int yaffs_read(int fd, void *buf, unsigned int nbyte) ;
int yaffs_write(int fd, const void *buf, unsigned int nbyte) ;
int yaffs_close(int fd) ;
off_t yaffs_lseek(int fd, off_t offset, int whence) ;
int yaffs_unlink(const char *path) ;
int yaffs_rename(const char *old, const char *new) ;
int yaffs_stat(const char *path, struct stat *buf) ;
int yaffs_lstat(const char *path, struct stat *buf) ;
int yaffs_fstat(int fd, struct, struct stat *buf) ;
int yaffs_chmod(const char *path, mode_t mode) ;
int yaffs_fchmod(int fd, struct, mode_t mode) ;
int yaffs_mkdir(const char *path, mode_t mode) ;
int yaffs_rmdir(const char *path) ;
yaffs_DIR *yaffs_opendir(const char *dirname) ;
struct yaffs_dirent *yaffs_readdir(yaffs_DIR *dirp) ;
void yaffs_rewinddir(yaffs_DIR *dirp) ;
int yaffs_closedir(yaffs_DIR *dirp) ;
int yaffs_mount(const char *path) ;
int yaffs_unmount(const char *path) ;
int yaffs_symlink(const char *oldpath, const char *newpath);
int yaffs_readlink(const char *path, char *buf, size_t bufsiz);
int yaffs_link(const char *oldpath, const char *newpath);
int yaffs_mknod(const char *pathname, mode_t mode, dev_t dev);
off_t yaffs_freespace(const char *path);
```

4.4.3. YAFFS 文件系统的实现

YAFFS 在嵌入式 Linux 中有两种使用方式，其一是作为根文件系统，另一种方式是作为普通文件系统在系统启动以后被挂载。这里，我们把 YAFFS 作为一个普通的数据分区。

4.4.3.1. 文件系统的移植

1、首先，我们需要得到 YAFFS 文件系统的源代码，可以从以下网站上得到：http://www.alephl.co.uk/yaffs/

2、在 linux-2.4.x/fs/目录下创建 yaffs 目录；

3、把下载下来的 YAFFS 目录下的 devextras.h、yaffs_fs.c、yaffs_gets.c yaffs_guts.h 、 yaffs_mtdif.c 、 yaffs_mtdif.h 、 yaffsinterface.h 以及 yportenv.h 拷贝到所创建的目录，并拷贝 Makefile 文件；

4、修改 linux-2.4.x/fs/Config.in 文件，增加如下内容：

```
if [ "CONFIG_MTD_NAND" = "y" ]; then
tristate "Yaffs filesystem on NAND" CONFIG_YAFFS_FS
fi
```

4.4.3.2. 其他内核有关部分配置

此配置包含四个部分：

1、MTD 设备驱动程序的配置，

2、Block devices 的配置，

3、文件系统的配置，

4、Flash Tools 的配置。

MTD 设备驱动程序的配置。MTD 设备的配置包含几部分选项，应该把它们直接编译进内核。

首先，MTD 设备自身必需被使能：

```
CONFIG_MTD=y
```

下一步是根据 Flash 的使用情况来决定是否使用 MTD 分区(CONFIG_MTD_PARTITIONS)，由于在 Flash 中同时包含 bootloader、kernel 和 root 分区和 YAFFS 文件系统，则必需配置此选项。这里我们设置为：

```
CONFIG_MTD_PARTITIONS=y
```

之后的 CONFIG_MTD_CHAR 和 CONFIG_MTD_BLOCK 必需被使能，前者通知编译器在编译内核时增加对 MTD 字符设备的支持，用来支持直接对 Flash 的擦除和烧写，后者为对 MTD 块设备的支持，用来在其上挂接文件系统：

```
CONFIG_MTD_CHAR=y
CONFIG_MTD_BLOCK=y
```

如果需要在调试环境下，则可以设置调试选项：

```
CONFIG_MTD_DEBUG=y
CONFIG_MTD_DEBUG_VERBOSE=3
```

下面的“RAM/ROM/FLASH chip drivers”子选项中包含了 FLASH 的探测程序和 FLASH 芯片的读/写/擦除命令集，使能 CONFIG_MTD_CFI，则内核在启动过程中自动检测支持 CFI 接口的 FLASH 芯片，如 Intel 公司的 FLASH 芯片，使能 CONFIG_MTD_JEDECPROBE 则检测支持 JEDEC 接口的 FLASH 芯片，如 AMD 公司的芯片。如同时选中表示对这两种接口的芯片都检测，除增加一点检测时间外并没有其他的不良影响。其后的内容为 FLASH 的读/写/擦除命令集，如 CONFIG_MTD_INTELEXT 支持所有 Intel 公司的 FLASH 芯片。用户可以根据自己所使用的 FLASH 来适当配置所支持的命令集，可以多选，因 MTD 驱动程序会根据检测到的 FLASH 类型来从所配置的命令集中选择合适的命令集。

块设备“Block devices”的支持。默认支持的块设备为 ROM 盘块设备(CONFIG_BLK_DEVBLKMEM)和 RAM 盘(CONFIG_BLK_DEV_RAM)，在配置中应加入 RAM 盘的支持。

文件系统“File system”的选择。在此选项中使能 CONFIG_YAFFS_FS，其他文件系统的使用由用户自己根据情况而定。

完成以上工作后，我们需要重新编译内核和模块。如果 YAFFS 已经被编译入内核，那么现在这个新的内核就已经支持 YAFFS 文件系统了；如果是按照模块编译的，那么我们在系统启动完成后，还需要加载 YAFFS 模块。不过由于 uClinux 不支持模块动态加载，而且嵌入式 LINUX 不能够像桌面 LINUX 那样灵活的使用

insmod/rmmod 加载卸载设备驱动程序，因而这里我们采用的是将设备驱动程序静态编译进 uClinux 内核的方法。

我们可以通过查看/proc/filesystems 可以知道此时内核是否支持 YAFFS：

```
# cat /proc/filesystems
```

里面应该有 yaffs 的信息了，然后就可以创建挂载点，挂载 YAFFS 系统分区：

```
# mkdir /mnt/y                                  #创建文件系统的挂载点
# mount -t yaffs /dev/mtdblock3 /mnt/y          #挂载文件系统
```

4.4.3.3. 辅助工具的建立和使用

YAFFS 文件系统辅助工具常用的只有三种，即 mkyaffs、eraseall、erase。其中 mkyaffs 为产生 YAFFS 文件系统镜像的工具，运行于主机上；erase 和 eraseall 工具运行于目标机上，用来对 FLASH 芯片的擦除。

在 YAFFS 源代码包的目录下有一个 utils 目录，配置编译后就能产生 mkyaffs 工具。

FLASH 工具“FLASH Tools”选项的配置。首选使能 CONFIG_USER_MTDUTILS，其他的 FLASH 工具才能够被配置，常用的为前面提及的 erase 等工具，在配置中使能 CONFIG_USER_MTDUTILS_ERASE。

文件镜象的生成。这种情况是针对在初始状态下需要有一些文件直接放到 yaffs 分区。生成文件镜象的工具为 mkyaffs，使用方法如下所示，mkyaffs 的命令格式为：

```
mkyaffs  -d 根目录  [ -b| l ]  [-e 擦除块大小]  [-o 输出文件]
  [-v [0-9]]  [-q]
```

生成的文件就是具有 yaffs 的文件系统的镜像，可以在这个镜像文件下载到目标系统上后，使用 dd 等工具直接往 yaffs 分区/dev/mtd3 上写入。

erase、eraseall 工具的使用方式和 mkyaffs 不同，mkyaffs 使用在主机上，而 erase 使用在目标机上，主要用来对 FLASH 芯片的擦除，其参数主要有三个：mtd 字符设备的文件节点；FLASH 内要被擦除块的起始块号；要被擦除块的总数目，例如：

```
erase  mtd3  0  8
```

表示在第四个 FLASH 分区内，从首块开始连续擦除八个块。当以根文件系统挂接

YAFFS 时要慎用 erase 工具。

4.5. 对掉电保护的处理

上文也已经提到，由于税控收款机对每张发票都不能随便私自进行处理，比如说打印一张发票刚好打到一半就掉电了，那么这张发票不能作废，在重新上电以后仍旧要打印完这张发票。一般来说有两种方法处理这张发票，一是从上次打印的地方接着打印，另一种是从头开始重新打印。不过如果从头开始打印那么发票重新安装需要人工干预，而且安装的位置肯定与前一次有所不同，因此最后打印必然会产生很明显的二次打印效果。因此我们采用了折中的办法——对当前行重新打印，这样就不用人工干预，打印效果也比较良好。

系统已经在硬件上使用大电容对系统维持一段工作时间（如果要求达到将近 100ms 的时间，我们可以通过计算得到一个相应的电容值），而相应的在软件上也要配合设计一定的处理方法。

在一般情况下，系统在 RAM 上每写一段数据（如 16KB），就把用户数据写入 SM 卡，这就保证了最大限度上的数据保存。

系统的第一工作电压是 24v，掉电时，系统在收到硬件发出的低于 24V 的电压信号时，表明现在已经产生掉电情况，而在当电源电压下降到 5V 以下时，系统才停止工作。因此，我们要很好的利用这短短的 100ms。低于 24V 时，硬件发出一个中断，于是调用中断例程，把用户需要保存而还没有保存的数据（最多 16KB）以文件的形式存入到 SM 卡的 YAFFS 分区上，并额外加上一个日志文件，以便于下次系统启动时能根据日志文件知道上次工作到什么地方。这是一个硬中断，响应的时间极短，保证了实际的需要。

下面是掉电中断程序的伪码：

```
void poff_interrupt(int irq, void *dev_id, struct pt_regs * regs)
{
  关闭所有外围设备以节约电源;
  保存还未保存的数据;
  检查是否在打印状态;
```

```
        {
            如果是，记录当前打印位置;
        }
      写日志文件;
    }
```

根据 YAFFS 的技术指标，YAFFS 文件系统写入的速度是：1MB/0.53s，于是，假定可以用来写入的时间是 50ms，那么我们完全可以写入 100KB 的数据到 SM 卡上，即使原先 SM 卡不是初始状态（是指含有垃圾数据），YAFFS 覆盖写入的速度也达到 0.9MB/s，50ms 可以写入将近 50KB 的数据，这已经足够满足我们的数据保存的需要。

第五章 ZTax 应用程序的设计和实现

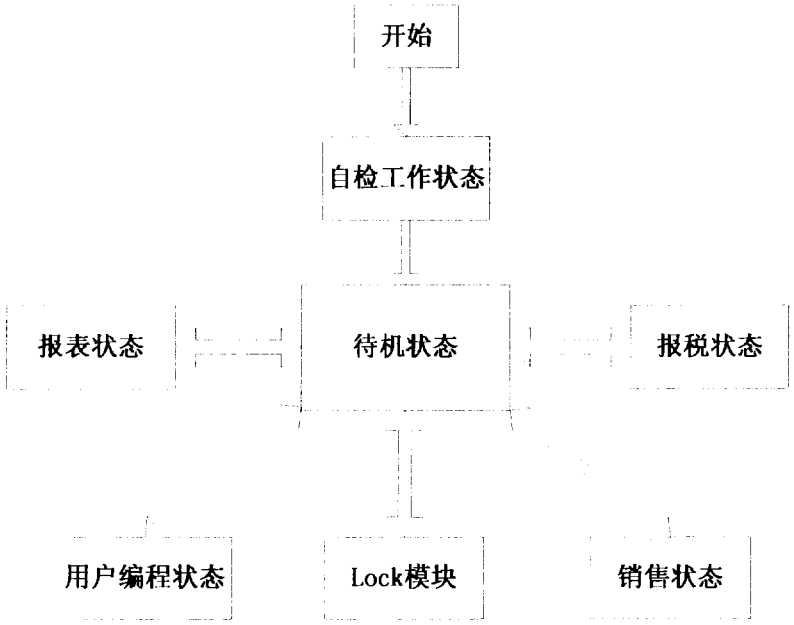
税控收款机是在能独立满足“税控”和发票管理基本要求的基础上，同时满足普通收款机在餐饮、娱乐、服务业的基本管理和普通票据及报表打印要求的税控装置产品。税控收款机是针对国内收款机使用现状，并结合税务征管法中有关计税要求开发的。我们的目标是使这款机器适用范围广泛、工作稳定、可靠。

新型的税控收款机采用内嵌式专用微型打印机，可快速登录打印多联发票，并能实现税控初始化、写卡报数、授权、打印发票、税务核查等一系列税控功能。税控收款机同时也是一台功能完善的商业收款机，不仅可方便纳税人报税，还可帮助纳税户实现商品管理、降低经营成本、避免营业员舞弊损耗。

5.1. 设计与实现

5.1.1. 工作状态

ZTax 应用程序有五种状态模式：自检工作状态、用户编程状态、销售状态、报表状态和报税状态。如下图所示：



图表 15 税控收款机状态模式转换图

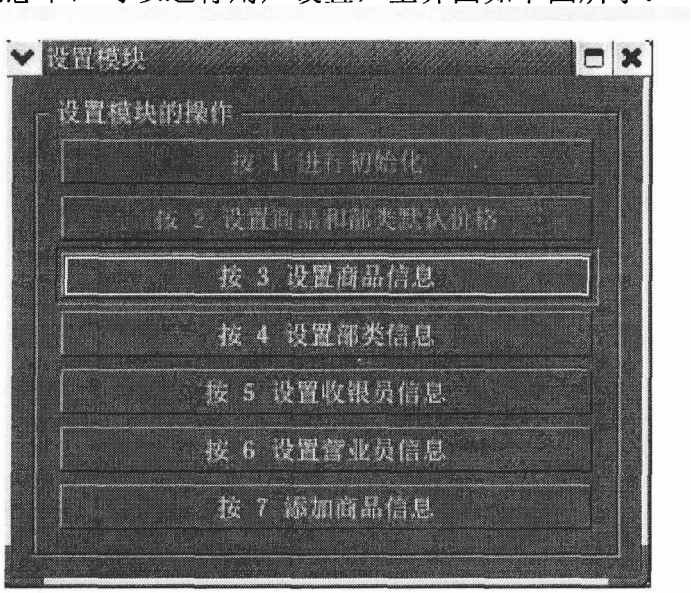
uClinux 起来以后，直接自动调用 ZTax 应用程序程序，如果 ZTax 通过自检，系统对各项记录做初始化，调入系统的日志文件，如果上次为非正常关机(掉电)，则处理前一次还未完成的任务。处理之后，系统进入待机状态，等待用户输入命令，并解析各项命令。

1. 自检工作状态

运行 ZTax 会自动完成对机器及其外围设备的检测工作，处于该状态时，能按预定的程序自动运行，如果自检不成功，可能硬件有故障，则报警输出关于那个有故障的硬件的信息，然后退出 ZTax 程序。

2. 用户编程状态

在用户编程状态下，可以进行用户设置，主界面如下图所示：



图表 16 用户编程状态

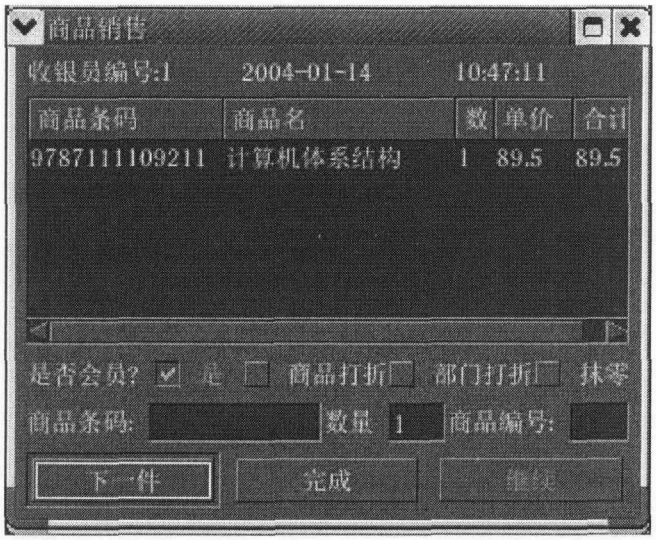
用户设置包括以下几个方面：

- 系统密码的设置
- 收银员名称及其密码的设置
- 设置系统时钟
- 项目的设置
- 服务费设置
- 折扣设置
- 设置信用卡名称

- 设置营业员姓名
- 进入（或退出）税控模式

3. 销售状态

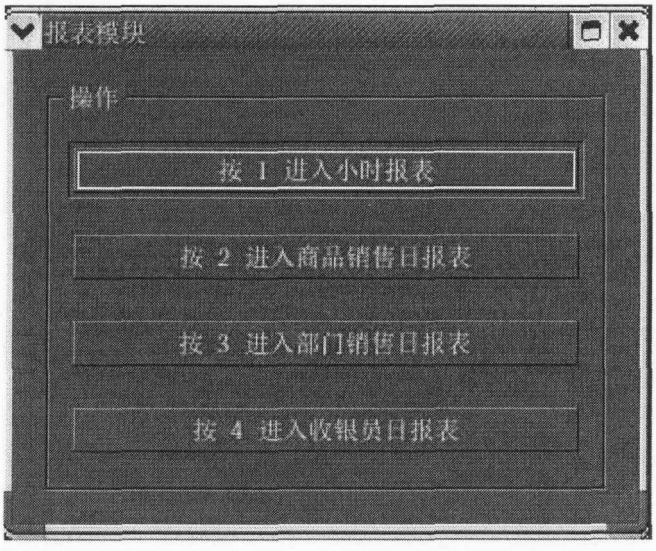
销售状态下可以完成各种商品的销售操作以及收银员的登录。界面如下图：



图表 17 销售状态

4. 报表状态

在报表状态下，可以完成各类报表的打印工作。



图表 18 报表状态

主要完成以下操作：

- 销售报表处理程序

- 日报表处理程序
- 阶段报表处理程序
- 查询销售额等操作（包括净销售额，现金总额，支票总额，信用卡金额）
- 打印发票日报表
- 人工废票处理程序
- 显示并打印购入的发票号码
- 训练模式下的税控操作

5. 报税状态

在报税状态下，可以完成报税操作。主要完成以下操作：

- 报表打印处理
- 打印业主登记记录
- 查询 FM 税控记录
- 纳税申报处理
- 税后登记处理
- 发票申报处理
- 测试 IC 卡

5.1.2. 函数接口

上面已经提到本税控收款机主要有 5 个模块组成，下面我们以报表报税模块为例来设计函数接口。

1. ZPosProc()函数

报表报税模块主程序 ZposProc 函数主要用来处理业务报表和与税务局打交道的报税，申报发票等操作。它的整体结构也是一个 while(true)的循环，不断接收用户键盘输入，对不同命令作不同业务操作。

2. Z_rep_proc()函数

函数 Z_rep_proc 是报表打印处理程序，它等待用户的输入，根据用户的命令键打印不同的报表。

- SomePluReport()函数，对应 case PLU[1]，用以打印全部 PLU 报表，它接收一个传入参数 pluth，表示从第 pluth 个 plu 开始打印。SomePluReport()通过调用 PrintReportHead()，PrintReportName()和 PrintReportTail()打印出报表头，报表名和报表尾。具体的报表内容则是遍历所有的 PLU，对每个 PLU 调用 OnePluReport()打出，这个函数的主要目的是打印出当前这个 PLU 的销售额占阶段销售总额的百分比。在系统中有全局变量 periodRpt，其中的一个分量 pluTotal 用来揭露阶段销售的总额。每次的销售操作都会及时地更新这个数据以及 plu 中的相应字段。
- HourReportProc()函数，对应 case XTIME（表示“个数/时间”），用以打印每日或每阶段中详细的每个小时销售状况的报表。它接收一个传入参数 flag，0 表示打印日报表，1 表示打印阶段报表。在系统中存在两个全局变量 dayRpt 和 periodRpt，在每次销售操作的时候都需要及时更新这两个全局变量。
- void SaleRptProc()函数，对应 case CASH（表示“收款”）和 REFUND（表示“退款”），用以打印每日或阶段中的总报表，所谓总报表就是指打印 dayRpt 或 periodRpt 中记录的所有内容。此函数接收一个传入参数 flag，0 表示打印日报表，1 表示打印阶段报表。SaleRptProc 调用 AllDeptReport()打印所有部门的报表，AllClerkReport()打印所有收银员的报表，AllWaiterRep()打印所有营业员的报表。
- InventoryReport()函数，对应 case EC（表示“即时订正”），用以打印存货报表。因为每个 plu 中都有一个分量 inventory 来表示当前这个 plu 还剩下的库存。这个函数需要遍历所有的 plu。
- SelectClearReport()函数，对应 case CLEAR，它接收用户的键盘输入，可以分别清除日报表（dayRpt 全局变量）中的内容，清除阶段报表（periodRpt 全局变量）中的内容，以及清除存货（plu 中的 inventory 分量）。

[1] PLU（PRICE LOOK UP），提供对商品的单价代码对照表，相当于有关所有商品的一个数据库，要初始化的信息有：代码，价格，所属部门，最高价格位，销售额，销售量，如果定义库存管理，还需要初始化库存量和进价。

5.2. GUI 的选择

ZTax 具有友好的人机界面接口。它的应用程序是基于 GUI 的图形用户界面。比较目前几个面向嵌入式系统的 GUI 发现，目前比较成熟，同时得到最多开发人员认可的有紧缩的 X Window 系统[41]、MiniGUI[42]、MicroWindows[40]、QT/Embedded [39]等系统。

紧缩的 X Window 系统其 X 服务器可以降低到 800K 的大小，但因为 X Window 系统的运行还需要其他程序和库的支持，包括 X 窗口管理器、XLib、建立在 XLib 之上的 GTK 和 QT 等函数库，因此，紧缩的 X Window 系统在运行期间所占用的系统资源很多，加上中文显示和中文输入等本地化代码之后，系统的整体尺寸和运行时的资源消耗将进一步变大。因此，嵌入式系统的开发商往往将紧缩的 X Window 系统定位在机顶盒等对资源要求并不苛刻的嵌入式系统上。

MiniGUI 和 MicroWindows 均为自由软件，只是前者遵循 LGPL 条款，后者遵循 MPL 条款。这两个系统的技术路线也有所不同。MiniGUI 的策略是首先建立在比较成熟的图形引擎之上，比如 Svgalib 和 LibGGI，开发的重点在于窗口系统、图形接口之上；MicroWindows 目前的开发重点则在底层的图形引擎之上，窗口系统和图形接口方面的功能还比较欠缺。举个例子来说，MiniGUI 有一套用来支持多字符集和多编码的函数接口，可以支持各种常见的字符集，包括 GB、BIG5、UNICODE 等，而 MicroWindows 在多字符集的支持上尚没有统一接口。

QT/Embedded 由于移植了大量的原来基于 QT 的 X Windows 程序，提供了非常完整的嵌入式 GUI 解决方案，可以说是一个成熟的软件。Qt/Embedded 是目前现有的基于 Linux 的图形系统中最适合嵌入式应用开发的图形系统。

Qt/Embedded

Qt/Embedded 是著名的 Qt 库开发商 TrollTech 发布的面向嵌入式系统的 Qt 版本[39]。因为 Qt 是 KDE 等项目使用的 GUI 支持库，所以有许多基于 Qt 的 X Window 程序可以非常方便地移植到 Qt/Embedded 版本上。因此，自从 Qt/Embedded 以 GPL 条款形势发布以来，就有大量的嵌入式 Linux 开发商转到

了 Qt/Embedded 系统上。比如韩国的 Mizi 公司，台湾省的某些嵌入式 Linux 应用开发商等等。

Qt 是一个跨平台的 C++ 图形用户界面库，由挪威 TrollTech 公司出品，目前包括 Qt，基于 Framebuffer 的 Qt Embedded，快速开发工具 Qt Designer，国际化工具 Qt Linguist 等部分。Qt 支持所有 Unix 系统，当然也包括 Linux，还支持 WinNT/Win2k，Win95/98 平台。目前最高的稳定版本是 Qt3.0.X。在 Unix 和类 Unix 的世界中，GUI 的库有很多，比如 Motif、Gtk＋、xcWindows 和 Xforms，作为与它们平行的图形库，Qt 具有许多无法比拟的优点：

- 可移植性：Qt 不只是适用于 Unix 和类 Unix 系统，它同时适用于 MS Windows。
- 易用性：Qt 是一个 C++工具包，它由几百个 C++类工程。因此，Qt 具有 OOP 的所有优点。
- 运行速度：Qt 非常容易使用，且也具有很快的速度。这两方面通常不可能同时达到。当我们谈论其他 GUI 工具包时，易用常意味着低速，而难用则常意味着快速（或者从另一个方面讲，低速意味着易用，而快速则意味着难以使用）。但当谈论 Qt 时，其易用性和快速则是密不可分的。这一优点归功于 Qt 的开发者的辛苦工作。他们花费了大量的时间来优化他们的产品。导致 Qt 比其他许多 GUI 工具包运行速度快的另一个原因是它的实现方式。Qt 是一个 GUI 仿真工具包，这意味着它不使用任何本地工具包作调用。Qt 使用各自平台上的低级绘图函数仿真 MS Windows 和 Motif（商用 Unix 的标准 GUI 库），当然，这能够提高程序速度。其他适用于多种平台的工具包，如 wxWindows，则是使用 API 层或 API 仿真，这些方法均以不同的方式使用本地工具包，从而降低了程序的运行速度。
- 丰富的 API：Qt 包括多达 250 个以上的 C++ 类，还替供基于模板的 collections，serialization，file，I/O device，directory management，date/time 类。甚至还包括正则表达式的处理功能。
- 支持 2D/3D 图形渲染，支持 OpenGL
- 大量的开发文档
- XML 支持

第六章 总结与展望

本文以嵌入式操作系统的迅速发展为背景，以研制开发新一代税控收款机 ZTax 为目标，分析了嵌入式操作系统和税控收款机的现状，指出了目前市场上主流税控收款机的不足之处。给出了税控收款机的总体设计，包括硬件设计和软件设计，并讨论了基于 uClinux 内核的嵌入式操作系统的移植，设备驱动的开发，ZTax 系统对数据保护的处理，其中分析并实现了一种操作系统 YAFFS 的移植，最后完成了税控机的应用程序的设计。

本税控收款机是跟杭州一家公司合作的项目，并得到了他们的大力支持。这个系统目前还在不断的开发、完善与测试中。另外本系统的平台是基于 ARM7 的，使用的操作系统是 uClinux 。由于 uClinux 没有内存管理单元，在好多地方还是有一定的局限性，所以以后我们把这个系统可以移植到 ARM9 上去，采用真正的 Linux，使得功能更强大，以满足更多的需要。而 YAFFS2 文件系统也已经发布，若移植到我们的系统，那么读写速度会更快，系统也会更加可靠。如果再增强网络功能，会更加方便用户进行管理。

今天，我国的税控收款机正得到蓬勃的发展，而中国收款机市场的竞争也会更加激烈。我们开发研制的税控收款机走在了行业的前头。我们相信，以 32 位处理器的嵌入式系统为基础的税控收款机必将成为今后收款机市场的主流。

参考文献

[1] 钟东江，税控收款机，信息与电脑，http://www.chinacc.com

[2] 时比特，纳税人经营数据可监控 税控机通用标准望出台，中国税务信息网，2002年9月16日

[3] 张然，金税工程启动税控行业，人民网，2004年2月3日

[4] 税控专栏，http://www.longfly.com.cn

[5] 众企业涉足金税工程 税控掀起"淘汰赛"，慧聪防伪商务网，2004年1月12日

[6] 林惠鹏，盘点2002年收款机市场，信息与电脑，2003年1月

[7] 刘铎，谈谈收款机与税控，信息与电脑，2002年10月

[8] 晓晨，众企业涉足税控收款机，金时网，2004年1月6日

[9] 嵌入开发技术论坛，http://www.embed.com.cn

[10] http://www.edw.com.cn/

[11] http://www.laogu.org

[12] 金西，黄汪，嵌入式Linux技术的现状与发展动向

[13] 孙永杰，"引人注目的嵌入式Linux"，微电脑世界，2000年第49期

[14] 李垣陵，UC/OS和UCLinux比较

[15] Abraham Silberschatz, Peter Baer Galvin, Greg Gagne, A.Silberschatz Operating System Concepts, 6th Edition, Wiley Text Books, June, 2001

[16] http://www.uClinux.org/

[17] 陈章龙，嵌入式系统——Intel XScale 结构与开发，将由北京航天航空大学出版社出版

[18] M.Beck et al. Linux Kernel Internels, 2nd Ed., Addison-Wesley, 1998.

[19] Anand K Santhanam, Vishal Kulkarni,嵌入式设备上的 Linux 系统开发,2002

[20] 陆宝辂、邵贝贝、李荐民，uClinux的设备驱动程序开发，单片机与嵌入式系统应用，2003年第6期

[21] Alessandro Rubini, Jonathan Corbet. Linux Device Drivers. 2nd Ed., O' Reilly, June 2001.

[22] 童诗白，华成英，模拟电子技术基础，第三版，高等教育出版社，2001 年 1 月

[23] 阎石，数字电子技术基础，第四版，高等教育出版社，1998年11月

[24] User’s Manual of SAMSUNG’s S3C44B0X 16/32-Bit RISC Microprocessor, http://www.samsung.com/Products/Semiconductor/SystemLSI/MobileSolutions/MobileASSP/MobileComputing/S3C44B0X/um_s3c44box.pdf

[25] ARM Architecture Reference Manual

[26] http://linux-fbdev.sourceforge.net/

[27] linux/usr/src/kernel/Documentation/

[28] http://www.mizi.com/

[29] http://kernelnewbies.org/documents/kdoc/kernel-api/linuxkernelapi.html

[30] http://www.enseirb.fr/~kadionik/embedded/uClinux/mtd/howto_mtd.html

[31] http://www.aleph1.co.uk/armlinux/projects/yaffs/index.html

[32] http://www.aleph1.co.uk/yaffs/

[33] http://www.linux-mtd.infradead.org/

[34] Karim Yaghmour, Building Embedded Linux Systems, O’Reilly, April 2003

[35] David Woodhouse, JFFS : The Journalling Flash File System, Red Hat, Inc.

[36] Michael Barr, Programming Embedded Systems in C and C++, O’Reilly, January 1999

[37] Joel R. Williams, A look at embedded systems and what it takes to build one

[38] 李明，Port uClinux on real hardware

[39] Qt/Embedded，http://www.trolltech.com/

[40] Microwindows，http://microwindows.censoft.com/

[41] X Window，http://www.x.org/

[42] http://www.minigui.org/

[43] Daniel Solin 著，袁鹏飞译，“24 小事学通 Qt 编程”，人民邮电出版社

[44] 魏永明，“实时嵌入式 Linux 系统上 GUI 的发展与展望”，微电脑世界，2000 年第 49 期

[45] Keith Haviland, Dina Gray, Ben Salama, Unix System Programming 2nd Edition, Addison-Wesley

[46] W.Richard Stevens, Advanced Programming in the UNIX Environment,

Addison-Wesley

[47] Daniel P. Bovet, Marco Cesati, Understanding the Linux Kernel, 2nd Ed. O' Reilly

[48] http://www.linuxdoc.org/

[49] Mitchell Bunnell, Solving the Embedded Linux Challenges, 2000

[50] John R. Levine. Linkers & Loaders, Revision: 2.2.

[51] Programming languages - C, International Standard. ISO/IEC 9899, First edition, 1990-12-15.

[52] David Woodhouse, JFFS: The Journalling Flash File System, Red Hat Inc.

[53] Geert Uytterhoeven, The Linux Frame Buffer Device Subsystem, http://www.cs.kuleuven.ac.be/~geert/

[54] Memory Product &Technology Division, ECC Algorithm, Samsumg

[55] I2C 总线规范

[56] 李祥兵等，Linux 中 I2C 总线驱动程序的开发

[57] 许庆丰，嵌入式 Linux 下彩色 LCD 驱动的设计与实现，http://www.edw.com.cn/

[58] 税控标准，中华人民共和国国家标准 GB18240.1-2003

论文和参与的项目

基于嵌入式系统的交叉汇编器的研究与实现，《工业控制计算机》，已录用，将发表于 2004 年 3 月刊。

在研究生期间，参加了浙江大学计算机系工程中心承担的国家重点科研项目“面向区域经济发展的高技术产品开发”中的“便携式动态心电图仪 DCG”项目的开发工作，已获得国家实用新型专利（申请号：01271646.4），发明专利（申请号：01140561.9）正在审批中。

同时参与浙大丽台研究中心的 GCC 项目的移植工作。同期还参加了“基于 uClinux 嵌入式系统的税控收款机的研制与开发”的项目。

致谢

首先，感谢郑扣根教授，他直接指导我从事的科研工作，领导整个项目的开发。他给予我非常多有益的指点和帮助。他渊博的知识大大开阔了我的视野，他诚恳严谨的处事态度给了我很多启发，他鲜明的个性给我留下了深刻印象。他是一位真正的良师益友。在此，向郑扣根老师表示最衷心的感谢。

在两年多的浙大生活中，有幸认识了好多朋友，他们在学习、生活等方面都有过有益的帮助。感谢在一起从事科研工作的师兄师姐以及师弟师妹，他们是朱奇波、苏斐琦、张磊、郑璨、许传玺、罗小华、郑南、缪强、冯刚、李祥兵、王万里、徐金星、方前、李龙连、高轶洁、王鹦鹉、冯骁斌、方琦等，感谢我的几位好友周文瑜、姜惠欣、庄旭东、韩峰、伍鹏、徐力、傅倬伟等同学。

浙江大学计算机系工程中心和浙江大学丽台研究中心给我的毕业设计提供了良好的科研和工作环境，感谢中心的老师和工作人员对我的帮助。

最后，将最深切的谢意献给我的家人。是他们的爱使我能有今天，无论什么时候，我都能从这份爱中得到安慰和鼓励。

王晓栋

2004 年 3 月于浙大求是园