君正集成电路产品参考设计
RD4720_FCR

用户指南

北京君正集成电路有限公司

® **Ingenic Semiconductor Co.Ltd.**

修改日志

版本号	日期	描述
0.1	2005/09/16	新建
0.2	2005/11/29	增加有关 u-boot 和 JTAG 烧录工具的说明
0.3	2005/12/20	删除有关 NAND 闪存电子盘的内容，保留 MTD 设备说明
1.0a	2006/01/18	增加有关 NAND 闪存中文件系统的使用说明。 增加有关 NOR 闪存启动和 NAND 闪存启动的配置电阻的使用说明。
1.0b	2006/02/10	删除有关片上 RTC 的内容，修改和芯片有关的命名方式。

目录

Chapter 1 概述

FCR 是君正集成电路有限公司专门为蓬勃发展的中国税控收款机市场推出的一个参考设计方案。基于超过两年的系统设计和技术支持经验，我们大量采纳客户建议，在 JZ4720 中集成了更多的功能部件；这使得 FCR 成为真正的税控收款机单芯片解决方案。

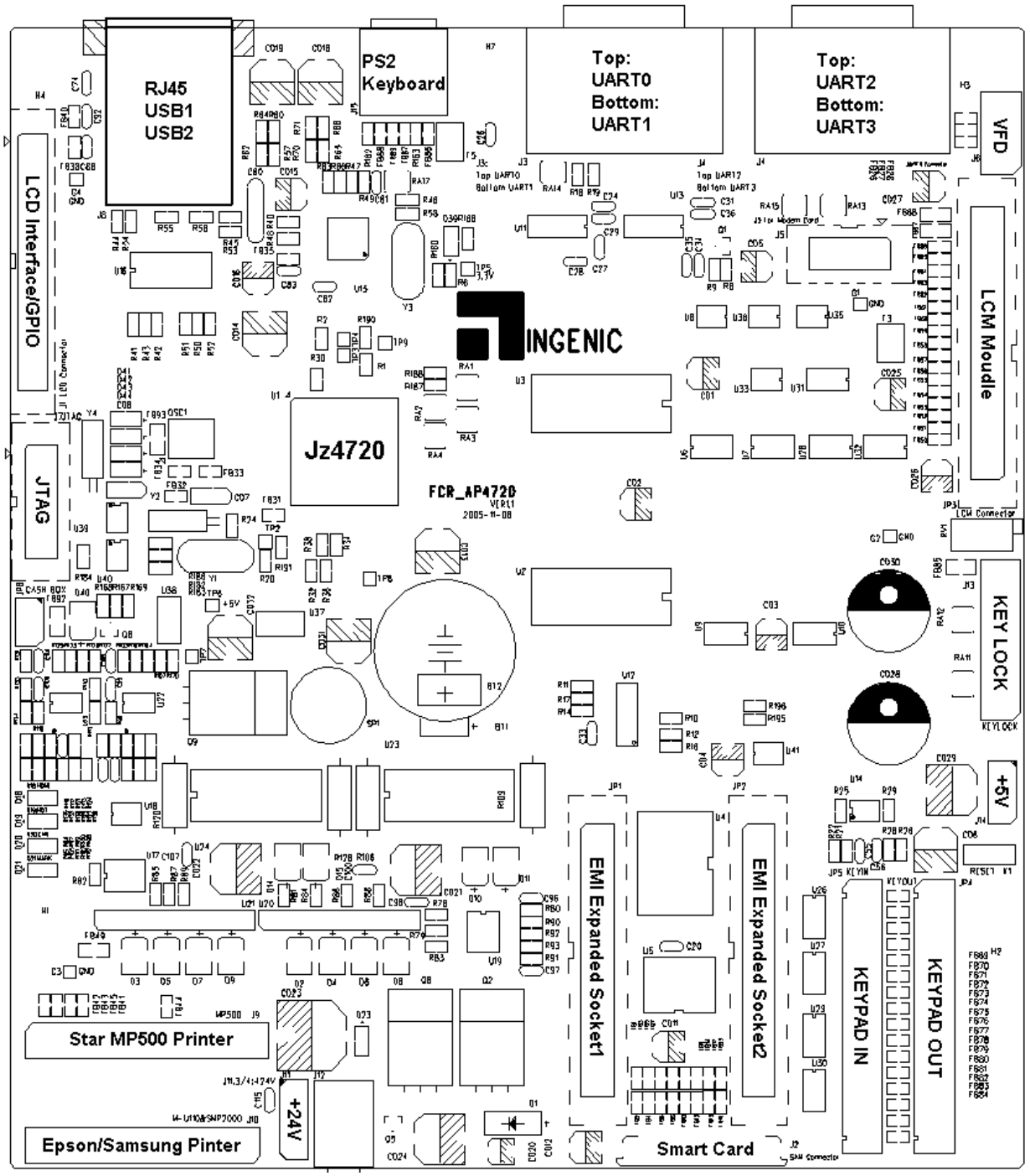
FCR 的硬件系统支持：

- 一个 10/100M 以太网控制器接口，支持 xDSL 等宽带接入
- 两个 USB 主控器接口，支持 U 盘报税
- 一个 LCD 控制器接口，支持最大分辨率到 800x600 的各式 STN 和 TFT 液晶面板
- 微型打印机控制器，支持 Epson U110-I/II, SAMSUNG SMP2000, Star MP512 打印机
- 两路相互独立的 ISO7816 智能卡控制器，用户卡插座同时支持 Memory 卡
- 4 路 16550 兼容的串行接口
- PS/2 键盘控制器接口
- 一个 VFD 客显模块，接在 SPI 总线上
- 一个最大支持到 16x16 的矩阵键盘，接在外部静态内存总线上
- 一个最大支持到 8 级的方式锁，接在外部静态内存总线上
- 支持一个自带控制器的 Smart LCD 模块，接在外部静态内存总线上
- 支持 16/32 位 SDRAM 总线。标准配置是一片 1Mx16bit-4Bank(8M 字节)的 SDRAM。
- 支持直接从 NAND 闪存启动。标准配置是一片 16Mx8bit(16M 字节)的 NAND 闪存。
- 法拉电容提供掉电后备能源支持，处理器子系统的后备供电时间可达 5 秒
- 支持一片 256K 字节的 NOR 闪存。提供产品开发阶段的系统核心引导，在最终产品中将被拿掉。

FCR 的核心部件是 JZ4720 高性能高集成度 SoC。上面运行面向消费电子应用的 CE-Linux™。系统有更快的启动速度和更高的可靠性。

如果您的目标板的启动配置是 NAND 闪存启动，那么请您转到最后一章。

本文帮助用户快速入门并且利用已有的软硬件资源开发出自己的产品来。您的参与和建议是对我们最大的鼓励和支持。


RJ45
USB1
USB2
PS2
Keyboard
Top:
UART0
Bottom:
UART1
Top:
UART2
Bottom:
UART3
VFD
LCD Interface/GPIO
LCM Moudle
JTAG
Jz4720
INGENIC
FCR_AP4720
KEY LOCK
EMI Expanded Socket1
EMI Expanded Socket2
KEYPAD IN
KEYPAD OUT
+5V
+24V
Star MP500 Printer
Epson/Samsung Pinter
Smart Card

Chapter 2 引导程序

在 FCR 中有三个引导程序可供产品开发人员使用：专门用作调试系统从 NOR 闪存启动的 U-Boot 和 RedBoot™和从 NAND 闪存启动的应用于最终产品的 NANDBoot。我们“强烈建议”用户在设计自己的硬件系统的时候，同时支持这两种闪存的启动。在产品研发调试阶段使用 NOR 闪存作为启动设备。而在批量生产阶段就只支持 NAND 闪存启动。选择模式的方法可以参考我们提供的参考设计中的配置电阻方式。

U-Boot 是为嵌入式平台开发而提供的开放源代码的引导程序，其前身是专为 PowerPC 开发的 PPCBoot。U-Boot 功能强大，提供串行口下载、以太网下载等多种下载方式，同时提供 NOR 闪存和 NAND 闪存存取管理，以及 USB 驱动和文件系统等的支持。而且 U-Boot 简单易学，配置和编译过程比较简单，非常适合开发人员使用。

RedBoot™基于 RedHat™提供的开放源代码的嵌入式实时系统 eCos™。它使用串行口终端并提供多种引导途径，包括：串行口下载、以太网下载和 NOR 闪存存取管理，是理想的协助开发工具。在 CE-Linux™系统核心开发阶段，我们强烈建议使用以太网下载核心映像的开发模式。在这种模式下，CE-Linux™系统核心每次修改后都无需写入到目标板的闪存中，利用网络引导核心、运行应用程序，效率非常的高。

NANDBoot 是我们自己开发的 NAND 闪存引导程序，它的二进制映像不足 4096 字节，放在 NAND 闪存物理第一个块里。使用它，可以极快地引导系统。但是，它不利于 CE-Linux™的系统核心开发。对于多数开发者而言，NANDBoot 不是一个熟悉的东西，而且里面的一些技巧性很强的代码过于晦涩和艰深。在本章将有专门小节介绍 NANDBoot，及一些和它有关的小技巧。

2.1 下载、编译和安装 GNU 工具链

编译 U-Boot 和 Linux 核心需要专门的交叉编译工具链，我们建议使用我们整合的基于 gcc 3.3.1 的版本。为了方便大家使用，请从我们的网站下载并按下面的流程安装配置编译工具链 mips_fp_le.tar.bz2 及使用环境。

2.1.1 在目标目录里解压缩工具链包

```
$ cd /home
$ mkdir mipstools
$ cd mipstools
$ tar -jxf mips_fp_le.tar.bz2
```

2.1.2 修改 bash 资源文件$HOME/.bashrc，追加命令执行搜索路径和动态链接库搜索路径

```
$ cd
$ vi .bashrc
export PATH=/home/mipstools/mips_fp_le/bin:$PATH
```

2.1.3 编译 RedBoot™则需要另外的交叉编译工具链，请按照下面的步骤安装和配置使用环境

请您登录下列网站，下载 GNU 的工具链源代码：binutils-2.10.tar.gz, gcc-3.2.tar.bz2, newlib-1.10.0.tar.gz. 编译 GCC 花费时间较长，请您耐心等待。

http://www.gnu.org/software/binutils
http://gcc.gnu.org
http://sources.redhat.com/newlib

2.1.4 解压缩源码包，准备编译、安装环境

```
$ cd
$ tar xfz binutils-2.10.tar.gz
$ tar xfj gcc-3.2.tar.bz2
$ tar xfz newlib-1.10.0.tar.gz
$ mv newlib-1.10.o gcc-3.2/newlib
$ mkdir binutils-work
$ mkdir gcc-work
$ mkdir /opt/mipsisa32 -p
```

2.1.5 编译安装 binutils

```
$ cd binutils-work
$ ../binutils-2.10/configure --prefix=/opt/mipsisa32 --exec-
prefix=/opt/mipsisa32 --target=mipsisa32-elf
$ make; make install
```

2.1.6 编译安装 gcc

```
$ export PATH=$PATH:/opt/mipsisa32/bin
$ cd gcc-work
$ ../gcc-3.2/configure --prefix=/opt/mipsisa32 --exec-prefix=/opt/mipsisa32
--target=mipsisa32-elf --enable-languages=c,c++ --with-gnu-as --with-gnu-ld
--with-newlib --with-gxx-include-dir=/opt/mipsisa32/mipsisa32-elf/include
$ make; make install
```

2.1.7 修改 bash 资源文件$HOME/.bashrc，追加命令执行搜索路径

```
$ cd
$ vi .bashrc
export PATH=$PATH:/opt/mipsisa32/bin
```

2.2 编译生成 U-Boot 的二进制映像

首先，从我们的技术网站的相关链接下载两个包：U-Boot 的源代码“u-boot-1.1.3.tar.bz2”和我们提供的补丁“u-boot-1.1.3-jz-yyyymmdd.patch.gz”。然后按下面的步骤给源码打上补丁：

```
$ tar xfj u-boot-1.1.3.tar.bz2
$ cd u-boot-1.1.3
$ zcat ../u-boot-1.1.3-jz-yyyymmdd.patch.gz | patch -p1
```

接下来，使用 make 命令配置 U-Boot：

```
$ make fcr_config
```

用户可以根据需要来修改自己的配置，FCR 平台的配置文件在 u-boot-1.1.3/include/configs/fcr.h 里。

然后，使用 make 命令编译生成 U-Boot 的二进制映像：

```
$ make
```

经过编译、链接，您所想要的二进制映像就放在: u-boot-1.1.3/u-boot.bin。借助 JTAG 工具，您可以把它写入到您的目标平台里。

2.2.1 使用 U-Boot

U-Boot 的二进制映像烧录到 NOR 闪存后，启动目标板，在串口输出的命令行里运行“help”命令，用户可以看到 U-Boot 可以支持的所有命令，运行“help command”可以查看具体命令的格式和使用方法。如果想从网络下载 Linux 核心并运行，可以参考下面的步骤：（这里假设您的服务器 IP 地址为 10.0.0.1，并且服务器已经启动了 BOOTP，TFTP 和 NFS 服务）

```
FCR # setenv bootfile uImage
FCR # setenv bootargs mem=16M console=ttyS0,115200n8 ip=bootp
nfsroot=10.0.0.1:/nfsroot/mipsroot rw
FCR # bootp
FCR # setenv serverip 10.0.0.1
FCR # tftp
FCR # bootm
```

察看当前的环境变量，使用 `printenv` 命令。设置新的环境变量使用 `setenv` 和 `askenv` 命令。保存新设置的环境变量到 `Flash` 中，使用 `saveenv` 命令。

2.2.2 Linux™命令行参数

U-Boot 的命令行参数采用设置环境变量的方式传给 Linux 核心，用户只需要设置一个环境变量 bootargs 即可。编译生成的 U-Boot 二进制映像设置了缺省的环境变量，用户可以修改 FCR 的配置文件来修改这些设置。

2.3 编译生成 RedBoot™的二进制映像

首先，从我们的技术网站的相关链接下载两个包：eCos-2.0 的源代码“ecos-2.0-i386linux.tar.bz2”和我们提供的补丁“ecos-2.0-jz-yyyymmdd.patch.gz”。然后按下面的步骤给源码打上补丁：

```
$ tar xfj ecos-2.0-i386linux.tar.bz2
$ cd ecos-2.0
$ zcat ../ecos-2.0-jz-yyyymmdd.patch.gz | patch -p1
```

接着，您应该使用 eCos™的配置工具来配置 eCos™的目标平台。注意：请您确认配置工具所在的目录在您的搜索路径里。如果不在，您可以在 bash 的资源文件：$HOME/.bashrc 文件中加上。如下：

```
export PATH=$PATH:/home/fcruser/ecos-2.0/tools/bin
```

eCos™有两个配置工具，一个是命令行模式的 ecosconfig；另一个是图形界面的 configtool。我们在这里只介绍 ecosconfig.

```
$ export ECOS_REPOSITORY=/home/fcruser/ecos-2.0/packages
$ cd ecos-2.0
$ mkdir work-fcr
$ cd work-fcr
$ ecosconfig new fcr redboot
$ ecosconfig add flash net_drivers
$ ecosconfig tree
$ make
```

经过编译、链接，您所想要的二进制文件就放在: install/bin/redboot.bin。借助 JTAG 工具，您可以把它写入到您的目标平台里。

2.3.1 配置 RedBoot™启动选项

在运行 RedBoot™之前，请您注意您的硬件平台是否选择了正确的启动模式(请查看硬件手册上有关启动配置电阻的章节)。运行 RedBoot™需要 NOR 闪存引导。同时请您检查硬件平台是否是插好了网线？而且局域网里有 BOOTP 服务器？如果没有，请您耐心的等待一会儿，RedBoot™就会通过串口往外输出信息了。您手里购买的电路板是我们经过严格测试的，所以请不要怀疑它是否已损坏。开机等待时间长一般是由于网络自适应失败和得不到 IP 地址所致。

我们需要配置的启动选项是加载 CE-Linux™核心映像和执行 Linux™并传入命令行参数。这是两个命令，可以写在 RedBoot™的配置脚本里。已经熟练使用 Windows™下超级终端和 Linux 下 minicom 的用户还可以方便的使用拷贝-粘贴来快速录入 RedBoot™配置脚本。下面是 RedBoot™启动时的一个截图。

```
Ethernet eth0: MAC address 00:47:20:fc:00:01
IP: 192.168.9.117/255.255.255.0 , Gateway: 192.168.9.1
Default server: 192.168.9.1, DNS server IP: 0.0.0.0

RedBoot (tm) bootstrap and debug environment [ROM]
Non-certified release, version v2_0 - built 08:55:23, Oct 10 2005

Platform: FCR (JZ4720)
Copyright (C) 2000, 2001, 2002, Red Hat, Inc.

RAM: 0x80000000-0x80800000, 0x8001108a-0x80800000 available
FLASH: 0x9FC00000 - 0x9FC40000, 64 blocks of 0x00001000 bytes each.
== Executing boot script is 1.000 seconds - enter ^C to abort
^C
RedBoot>
```

现在我们可以逐项配置 RedBoot™了。

```
RedBoot> fc
Run script at boot: false true
Boot script:
...
Enter script, terminate with empty line
>> load -m tftp -h 192.168.9.20 mips/vm-fcr
>> exec -c "mem=8M console=ttyS0,115200 ip=bootp nfsroot=192.168.9.20:/
nfsroot/developroot rw"
>>
Boot script timeout (1000ms resolution): 1
Use BOOTP for network configuration: true
DNS server IP address:
GDB connection port: 9000
Force console for special debug messages: false
Network debug at boot time: false
Update RedBoot non-volatile configuration - continue (y/n)?y
... Unlock from 0x9fc3e000-0x9fc3f000: .
... Erase from 0x9fc3e000-0x9fc3f000: .
... Program from 0x80ffd000-0x80ffe000 at 0x9fc3e000: .
... Lock from 0x9fc3e000-0x9fc3f000: .
RedBoot>
```

如果您的开发环境里没有 BOOTP 服务器，您可以使用指定静态 IP 地址的配置方式。如下：

```
...
Enter script, terminate with empty line
>> load -m tftp -h 192.168.9.20 mips/vm-fcr
```

```
>> exec -c "mem=8M console=ttyS0,115200 ip=192.168.9.200:::::eth0 nfsroot=
192.168.9.20/nfsroot/developroot rw"
>>
Boot script timeout (1000ms resolution): 1
Use BOOTP for network configuration: false
Gateway IP address:
Local IP address: 192.168.9.200
Local IP address mask:
Default server IP address:
DNS server IP address:
GDB connection port: 9000
Force console for special debug messages: false
Network debug at boot time: false
Update RedBoot non-volatile configuration - continue (y/n)?y
...
```

2.3.2 LINUX™命令行参数

在上个小节里提到了命令行参数这个概念。同时配置举例里也有两个例子。这一节将就命令行参数中的每个小项作进一步说明。需要指出的是：命令行小项之间没有先后顺序。

2.3.2.1 指明内存容量

开发板上只有一片 8M 字节的 SDRAM，所以命令行参数应该这样设置：

```
mem=8M
```

如果 SDRAM 的容量增加到 16M 字节，那么设置中的 8 可以改成 16。当然，您也可以不改，这样，您就只能使用 16M 字节内存中的 8M 字节了。并非所有的内存都给用户程序使用。LINUX™的内存将被系统核心和 init RAMDISK 占用一部分。如果用户使用静态链接编译应用程序，那么系统内存还将被动态链接库占用一部分。

2.3.2.2 系统控制台

系统控制台的实现语法是"console=设备，设备控制参数", 一个系统中可以有多个控制台。最后一个声明的是主控制台。系统控制台用来显示输出来自系统核心的一些启动信息。如果您不想显示这些信息，可以使用"quiet"来屏蔽它们。下面是两个常用的控制台设备：

```
console=tty1
console=ttyS0,115200
```

第二个例子中的 115200 是指 ttyS0 的波特率。而 ttyS0 在 LINUX™指的是串口 0。

2.3.2.3 IP 地址

我们可以静态的指定 IP 地址，也可以通过 BOOTP 来获得动态的 IP 地址。LINUX 支持 ARP, BOOTP 和 DHCP 等 3 种方式。在一些资料里，DHCP 本质是 BOOTP 的扩展，理论上两者兼容。但是，在实际的应用中，两者是有区别的。RedBoot 只支持 BOOTP，而不支持 BOOTP 的扩展-DHCP。而 LINUX 系统核心是同时支持的。

```
ip=bootp
ip=192.168.9.200:::::eth0
```

2.3.2.4 LINUX™根设备

任何 LINUX™系统都需要一个根文件系统。我们提供两个典型的根文件系统：一个是用于开发的，运行环境比较齐全的根文件系统；另一个是经过特别裁减的、供客户制作产品的根文件系统。产品根文件系统非常的简单，缺少很多文件，用户得根据实际需要添加。根文件系统只有一个，它可以放在不同的介质上。这个介质就是所谓的 LINUX™根设备。我们接触最多的根设备是闪存，INITRD 和 NFSROOT。

使用 NAND 闪存上建立的 mtd 设备
```
root=/dev/mtdblock0 rw
```

使用 NFSROOT
```
nfsroot=192.168.9.20:/nfsroot/developroot rw
```

使用 INITRD 时，应该先把 INITRD 的映像文件下载到 SDRAM 里，然后使用下面的命令行参数
```
initrd=0x00700000,0x008e8dff root=/dev/ram0 rw
```
当然，我们也支持把 INITRD 直接链接到核心中这样，我们的命令行参数就成这样了：
```
root=/dev/ram0 rw
```

2.4 NANDBoot

NANDBoot 的任务是完成系统初始化工作，把 LINUX 核心映像或 INITRD 映像从 NAND 闪存里加载到系统内存里。然后传入命令行参数并且运行 LINUX 核心。需要用户关心的是系统初始化工作和 LINUX 命令行参数。这些和用户的最终产品密切相关。

2.4.1 系统初始化

系统初始化分为 3 个主要步骤：

- 时钟和电源管理单元的初始化，时钟和电源管理单元控制着 JZ4720 的工作时钟和功能模块的开启关闭状态。依据您的设计，选择合适的工作时钟。
- GPIO 功能管脚初始化，片上设备的简单初始化。JZ4720 的功能管脚都可以复用作 GPIO 管脚，而这些 GPIO 管脚既可以做 I/O 也可以做中断输入用。
- SDRAM 初始化。SDRAM 初始化过程较为复杂，依据相关硬件规范，一方面要设置正确的时序参数，另一方面要按要求的状态机变换过程初始化 SDRAM 芯片。

我们通常的做法是，先用 RedBoot 正确引导系统后，在目标板上运行程序把 EMC 单元的各个控制寄存器的值显示出来。然后再利用这些值来做 SDRAM 的配置。当然，SDRAM 模式寄存器的值是显示不出来的。这需要用户自己来计算和设置。如果用户对 SDRAM 的初始化过程知道的不多，我们也可以帮助用户来完成这一步工作，我们所需要的资源只是一块您提供的可以工作的产品板和相关 SDRAM 芯片的数据手册。

2.4.2 Linux™命令行参数

在我们提供的 NANDBoot 源代码里，您可以找到有关 Linux™命令行部分。NANDBoot 提供的最大命令行长度为 256 字节。您可以按照我们前面提到过的命令行细节来组织自己的命令行。我们通常会提供两个 NANDBoot 二进制映像，一个以 NFSROOT 作为根设备，用于应用程序开发。另一个以 NAND 闪存作为根设备，用于最终的产品。

2.5 用 JTAG 工具烧录二进制映像

用户可以使用 JTAG 工具在线烧录 U-Boot，RedBoot™和 NANDBoot 的二进制映像到目标板的 NOR 闪存和 NAND 闪存里。具体使用 JTAG 工具的帮助可以参考 JTAG 自带的 README 文件，这里我们举几个简单的例子供用户参考。

烧录 U-Boot 二进制映像到目标板的 NOR 闪存，使用下面的命令：

```
$ ./jflash u-boot.bin
```

烧录 NANDBoot 二进制映像到目标板的 NAND 闪存的第一个块起始的地方，使用下面的命令：

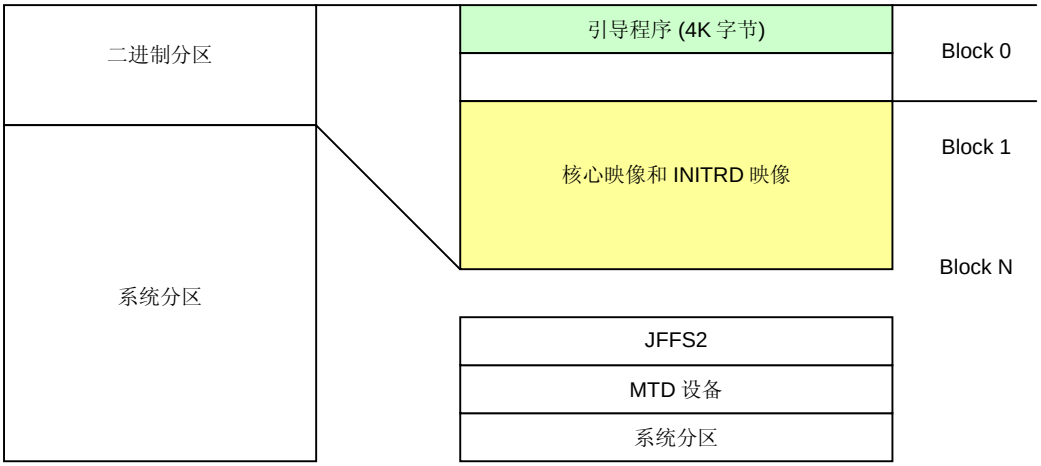
```
$ ./jflash -n -s 0 nandboot.bin
```

烧录 Linux 二进制映像到目标板的 NAND 闪存的第二个块起始的地方，使用下面的命令：

```
$ ./jflash -n -s 1 zImage
```

Chapter 3 NAND 闪存的使用

由于 JZ4720 芯片内部提供 NAND 闪存的引导机制，使得用户可以把自己的引导程序放在 NAND 闪存的最开始 4K 字节空间里。系统在启动时会先执行这 4K 字节引导程序，由这 4K 字节引导程序完成系统初始化，加载系统映像，完成最后的引导。NAND 闪存分为两部份：二进制分区和系统分区。二进制分区专门用作 NAND 闪存引导。而系统分区则是用来做文件系统使用的。我们提供 MTD 设备来建立 JFFS2 文件系统。



3.1 利用 JTAG 烧录 NAND 闪存

如前所述，我们的 JTAG 工具提供了对 NAND 闪存的烧录功能。这个功能对整个 NAND 闪存是有效的，所以您在使用的时候要非常小心，要确保该操作仅仅在二进制分区范围内，不要误写系统分区。

3.2 利用 nandutil 烧录 NAND 闪存

我们也提供在目标板上运行的 NAND 闪存烧录程序"nandutil"。"nandutil" 具有和 JTAG 烧录工具类似的功能。当您把系统分区用作 MTD 设备时，请遵循 MTD 设备的常规操作流程。

3.2.1 烧录 NANDBoot

```
# nandutil -b loader.bin
```

3.2.2 烧录 CE-Linux™ 系统核心

```
# nandutil -b zImage -s 1
```

3.2.3 如果系统有 INITRD，并且放在 NAND 闪存的块 52 开始的地方

```
# nandutil -b initrd.bin -s 52
```

3.2.4 擦除 NAND 闪存(从 block 0 开始，连续 1024 个 block)

```
# nandutil -e 1024
```

如果系统的 NAND 闪存是从旧货市场买来的，普通的擦除可能会失败。原因是我们避开了 NAND 闪存出厂时标注的坏块，而使用过的 NAND 闪存可能已经把好的块也给标注成坏块了。您可以使用附加参数-f 来强行擦除所有的块，包括已经标注成坏块的块。这里有个问题：强行擦除操作也会擦掉 NAND 闪存出厂时标注的坏块标志，这使得真正的坏块有机会混入好的块里，给将来的使用埋下隐患。

3.3 MTD 设备

MTD (Memory Technology Device) 是 LINUX™的标准设备。这也是嵌入式应用开发工程师非常熟悉的设备。上面可以创建 JFFS2 文件系统。在基于 NAND-flash 的 MTD 设备上创建 JFFS2 文件系统的步骤如下：

格式化 MTD 分区：

```
# flash_eraseall -j /dev/mtd0
```

mount 文件系统并安装目标程序：

```
# mount -t jffs2 /dev/mtdblock0 /mnt
# cd /mnt
# cp /home/destdir/* . -a
# sync
# cd /
# umount /mnt
```

如果您的 nand 闪存以前写入过数据，那么请用我们提供的 nandutil 进行强行擦除。如下：

```
# nandutil -e 1024 -f
```

NAND 闪存 MTD 设备的使用和 NOR 闪存 MTD 设备略有不同。因为写 NAND 闪存需要做 ECC 校验。不做 ECC 校验的写操作是非常危险的。所以，以往在 NOR 闪存上取得的经验，在 NAND 闪存上是不可行的。

Chapter 4 CE-Linux™系统核心

CE-Linux™是面向消费电子类应用的嵌入式 Linux™。它在启动速度、核心尺寸和电源功耗管理支持方面都有明显的改善。对于普通用户而言，CE-Linux™的使用和普通 Linux™没有太大的区别。请到我们的技术支持网站下载相关的源代码和补丁程序包。另外，我们的源代码中包含一些二进制目标文件，得用 uudecode 解码后才能使用。所以，您的工作主机上应该有工具软件 uudecode。如果您没有 uudecode，请您安装 sharutils 的 RPM 文件包，在您的 Linux™安装光盘上就有一个。

4.1 基本的核心开发流程

4.1.1 解压缩源代码，并且打上补丁

```
$ tar xfj celinux-040503.tar.bz2
$ cd celinux-040503
$ zcat ../celinux-jz-yyyymmdd.patch.gz | patch -p1
```

4.1.2 配置核心

打补丁后的 CE-Linux 具有一个缺省的配置文件称为：".config"。按照下面的方法，我们可以把这个文件填上我们需要的配置项，不需要用户到配置菜单里自己选择了。

```
$ make defconfig-fcr
$ make oldconfig
```

4.1.3 编译核心

```
$ make dep
$ make zImage 或者 make uImage
```

上面的两步是必须要的，不可以省略第一步直接去做第二步。用户可以编译生成两种格式的 Linux 核心，一种是压缩的二进制格式的核心 zImage，可以由 Redboot 下载并运行；另外一种是只可以在 U-Boot 中使用的 uImage。

make zImage 最后会生成 3 个核心映像：

- 带有调试信息的 ELF 格式的非压缩核心。它放在：
`celinux-040503/vmlinux`
- 压缩的 ELF 格式的核心。它放在：
`celinux-040503/arch/mips/zboot/images/vmlinuz.elf`
- 压缩的二进制格式的核心。它放在：
`celinux-040503/arch/mips/zboot/images/zImage`

这三个核心映像各有各的用途：第一个用于反汇编和调试核心用；第二个用于 RedBoot™下载；第三个用于 NANDBoot。请用户加倍注意！

make uImage 最后也会生成 3 个核心映像：

- 带有调试信息的 ELF 格式的非压缩核心。它放在：
`celinux-040503/vmlinux`
- 压缩的 uImage 核心。它放在：
`celinux-040503/arch/mips/uboot/uImage`
- 非压缩的 uImage 核心。它放在：

```
celinux-040503/arch/mips/uboot/uImage.uncompressed
```

4.2 FCR 的 CE-Linux™设备驱动程序

FCR 的 CE-Linux™设备驱动程序极大的丰富了标准 LINUX™的驱动程序代码库。用户可以利用这些源代码来设计自己的硬件驱动程序。这一节会详细的介绍各特殊驱动程序的特点及用户需要注意的地方。

4.2.1 FCR 设备驱动列表

文件	描述
drivers/char/jzchar/scankbd.c drivers/char/jzchar/scankbd.h	JZ SoC 的矩阵键盘驱动
drivers/char/rtc_ds12887.c drivers/char/rtc_ds1337.c drivers/char/rtc_pcf8563.c drivers/char/rtc_rx8025.c	挂接在静态内存总线或者 I2C 总线上的 RTC 模块的驱动
drivers/char/jzchar/memcard.c drivers/char/jzchar/memcard.h	可以在用户卡插槽里使用的 Memory 卡的驱动
drivers/char/jzchar/scc.c drivers/char/jzchar/scc.h	ISO7816 的驱动
drivers/char/jzchar/vfd_16311.c drivers/char/jzchar/vfd_16311.h	SPI 总线上的 VFD 驱动，同时还控制钱箱
drivers/mtd/nand/jzsoc.c	MTD 设备驱动
drivers/video/Jzlcd.c drivers/video/Jzlcd.h	JZ SoC 的集成 LCD 控制器驱动
drivers/video/lcdm.h drivers/video/19264.c drivers/video/sd1335.c drivers/video/t6963c.c	挂接在静态内存总线上的 Smart LCD 液晶模块的驱动

*系统里的还有些设备使用 Linux 标准驱动代码，这里就没有列出来，例如：16550 兼容的串行口；有些设备可以在应用程序里直接访问，这里也没列出来，例如：方式锁；还有些设备没有提供源代码，这里没有列出来，例如：集成的微型打印机控制器。

4.2.2 矩阵键盘

挂接在静态内存上的矩阵键盘千差万别，我们给出了一个 16x16 的扫描键盘的驱动程序的例子。您可以根据您的实际需要来修改定义在头文件中的宏。放开"`#if 0`"屏蔽的代码，您还可以得到按键和扫描码表的一一对应关系。当然，最后的产品里面还得继续屏蔽这段代码。

```
Character devices
   [●]y [ ]- [ ]n JzSOC char devices support
      JzSOC char devices support
         [●]y [ ]- [ ]n JzSOC scan keyboard support (Max to 16x16)
```

4.2.3 实时时钟 RTC

我们提供 4 种实时时钟的驱动。DS12887 是自带晶振和后备电源并且挂接在静态内存总线上的 RTC 模块。DS1337, PCF8563 和 RX8025 都是挂接在 I2C 总线上。RX8025 内部自带高精度晶振。任何 RTC 模块的时钟精度由该模块的驱动时钟晶振有关。我们的 参考设计里可以使用的 RTC 时钟是挂接在 I2C 总线上的 PCF8563。

```
Character devices
   [ ]y [ ]- [●]n Dallas DS12887 Real Time Clock (Memory Bus)
   [0x00000000  ]  Index register address:
```

```
[0x00000000  ]  Value register address:
[ ]y [ ]- [●]n Dallas DS1337 Real Time Clock (I2C Bus)
[●]y [ ]- [ ]n Philips PCF8563 Real Time Clock (I2C Bus)
[ ]y [ ]- [●]n RX8025 Real Time Clock (I2C Bus)
```

4.2.4 Memory 卡和智能卡驱动

在我们的参考设计中，Memory 卡和税控用户卡共用一个卡槽。您可以通过向 Memory 卡驱动程序发送一个 IOCTL 命令来完成两种卡的功能切换，具体方法请您参考我们提供的源代码示例。当您的卡座在 Memory 卡工作状态时，智能卡放进去操作是会失败的，反之亦然。我们的智能卡驱动程序被设计成为一种数据通信通道，它不提供 ISO7816 协议的翻译和控制。但是它提供给用户一些控制接口：您可以设置额外保护时间，速率因子和选择字符传输或块传输。请您参考我们提供的应用示例。我们建议您在写和智能卡相关的应用之前，请仔细阅读有关 ISO7816-3 的标准规范文本。

Memory 卡的设备文件的 I-node 属性是这样的：字符设备，MAJOR(238)， MINOR(4)。
智能卡的设备文件有两个，它们的文件 I-node 属性是：字符设备，MAJOR(238), MINOR(32 和 33)。

```
Character devices
   [●]y [ ]- [ ]n JzSOC char devices support
      JzSOC char devices support
         [●]y [ ]- [ ]n JzSOC On-chip SmartCard Controller support
         [●]y [ ]- [ ]n JzSOC SAM card slot shared memory card support
```

如果您的目标板的根文件系统中没有这些设备文件，请您用 mknod 创建。下面的例子是创建 Memory 卡的驱动：

```
# mknod /dev/memcard c 238 4
```

4.2.5 VFD 客显

VFD 客显驱动还提供对钱箱的控制。设备文件 I-node 属性是：字符设备，MAJOR(238)，MINOR(6)

4.2.6 MTD 设备驱动

我们还提供 NAND 闪存的 MTD 驱动。您可以在 MTD 分区上创建 jffs2 文件系统。在驱动程序源代码中，我们提供了一个分区方案，请您根据您的实际情况，把它改成满足您要求的形式。

```
Memory Technology Devices (MTD)
   [●]y [ ]m [ ]n Memory Technology Device (MTD) support
   [●]y [ ]m [ ]n  MTD partitioning support
   [●]y [ ]m [ ]n  Direct char device access to MTD devices
   [●]y [ ]m [ ]n  Caching block device access to MTD devices
   NAND Flash Device Drivers
      [●]y [ ]m [ ]n NAND Device Support
      [●]y [ ]m [ ]n NAND Flash device on JzSOC board
```

在 MTD 设备上使用 JFFS2 文件系统是许多嵌入式系统的经典选择。您只需要使用下面的配置：

```
File systems
   [●]y [ ]m [ ]n Journalling Flash File System v2 (JFFS2) support
   [0           ]  JFFS2 debugging verbosity (0 = quiet, 2 = noisy)
   [●]y [ ]- [ ]n JFFS2 support for NAND chips
```

4.2.7 Smart LCD 液晶模块

除了提供 LCD 控制器的驱动，我们还提供液晶模块的驱动。这些液晶模块通常自带控制器，所以称为 Smart LCD，它们通常挂接在静态内存总线上或 SPI 串行总线上。我们支持的三款液晶模块都是挂接在

静态内存总线上的，它们分别是：三个片选的 192x64 液晶模块，以 T6963c 为控制器的 240x128 液晶模块和以 SED1335 为控制器的 320x240 液晶模块。参考我们提供的 192x64 液晶模块驱动源代码，您可以非常方便的支持两个片选的 128x64 液晶模块。

我们在系统内存里划出一块虚拟显存，通过一个定时器来刷新液晶模块里的真正的显存。使得所有的 Smart LCD 驱动都提供标准 Linux Frame Buffer 驱动所提供的功能。您可以在上面运行复杂的 GUI 系统。我们还提供三个扩展的 IOCTL 功能：停刷新时钟，启动刷新时钟和手动刷新。他们的命令字的宏定义如下：

```
#define FBIOSTOPTIMER   0x4680
#define FBIOSTARTTIMER  0x4681
#define FBIOREFRESH     0x4682
```

在一个确定系统中，您只能选择一种液晶模块：

```
Console drivers
   Frame buffer support
      [●]y [ ]- [ ]n Support for frame buffer devices (EXPERIMENTAL)
      [●]y [ ]- [ ]n Smart LCD Module Support
      [●]y [ ]- [ ]n Tri-CS 192x64 module support
      [ ]y [ ]- [●]n Toshiba T6963c based 240x128 module support
      [ ]y [ ]- [●]n Epson SED1330/1335 based 320x240 module support
```

4.2.8 微型打印机驱动

设备文件 I-node 属性是：字符设备，MAJOR(238)，MINOR(0)

4.2.9 掉电保护驱动

设备文件 I-node 属性是：字符设备，MAJOR(238)，MINOR(7)

4.2.10 方式锁

我们没有专门的方式锁驱动。用户可以在应用程序里直接访问方式锁的操作地址来获取方式锁的状态。在 Linux™里，处理器访问的所有地址空间都是虚地址。所以，用户程序在访问一个实地址(物理地址)之前，应该先用库函数 mmap 把实地址给映射成虚地址；访问时，使用虚地址。相关细节请参考我们提供的源代码示例(gpio)。

4.2.11 其他设备

FCR 系统中还有大量别的常见设备：10/100M 以太网，USB 主控器，标准的 16550 兼容的串行口设备，PS2 键盘等。

Chapter 5 快速使用目标板

您手里的电路板的 NAND 闪存中已经灌入好了引导程序、系统核心和一个可以工作的根文件系统。请您完成下面的动作：

- 连接好各接口板。
- 使用交叉串口线接在 UART0 上，配置数据格式为 115200-8N1。
- 插好电源。

稍侯片刻，您的串口终端上就会有命令行提示符。为了保持输出的整洁，我们屏蔽了 LINUX 系统核心的调试信息的输出。

5.1 目标板上的根文件系统

5.1.1 应用程序

请您检查目录/bin 和/usr/local/bin。里面是所有您可以运行的程序。其中，/usr/local/bin 里存放的是一些和税控收款机有关的硬件设备的使用例程，它们的源代码可以在随机附送的光盘里找到。比较有用的是 portmap 和 mount.它们可以帮助您映射服务器上的 NFS 文件系统，以此来运行您自己的应用程序。如下：

```
# portmap
# mount -t nfs 10.0.0.1:/nfsroot/fcrroot /mnt
# /mnt/bin/fcr.exe
```

5.1.2 汉字字模

我们还在/usr/local/share 里放了三个字库文件，汉字字库支持 GB18030。我们提供的源代码里提供相应的计算字模索引的公式，这些公式支持到 GB13000(GBK)。

5.1.3 动态连接库

/lib 目录里是一些运行程序必须的动态连接库。您还可以使用 NFS 文件系统里的动态连接库。只需要设置下面的环境变量:

```
# export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/mnt/lib
```

5.2 怎样选择从 NOR 闪存启动？

您可以请硬件工程师把电阻 R34 焊下来放在 R38 上，这时：您的目标板就配置为从 NOR 闪存启动了。这时，您也许需要本文前面章节描述的知识。

5.3 警告！请不要随便修改 NAND 闪存的内容

如果，您不小心修改了 NAND 闪存的内容。有可能导致您的目标板引导失败。您可以按我们前面章节里提到的途径，自行恢复 NAND 闪存的内容。