第5章

嵌入式系统开发环境的建立

本章主要学习嵌入式系统开发环境的建立方法，通过本章的学习，可以掌握和了解以下知识点。

- 在宿主机端建立开发环境。
- 配置 minicom 终端。
- 嵌入式 Linux 系统内核的编译。
- 嵌入式开发板内核及系统文件的烧写方法。

5.1 建立宿主机开发环境


视频讲解

5.1.1 交叉编译

绝大多数的软件开发都是以本地(Native)编译方式进行的，即在普通计算机上开发编译、在本机上运行的方式。但是，这种方式通常不适合于嵌入式系统的软件开发，因为对于嵌入式系统来说，本机(即板上系统)没有足够的资源来运行开发工具和调试工具。因此，嵌入式系统软件的开发通常采用交叉编译的方式。交叉编译，就是指在一个平台上生成可以在另一个平台上执行的代码。

编译最主要的工作是将程序转化成运行该程序的 CPU 所能识别的机器代码。由于不同的硬件体系结构之间存在差异，它们的指令系统也不尽相同，因此，不同的 CPU 需要用与其相应的编译器来编译代码。交叉编译就是在特殊的编译环境下，把程序代码编译成不同的 CPU 所对应的机器代码。

通常把嵌入式系统的开发放置到普通计算机端，在一台普通计算机(称为宿主机)上建立交叉编译环境，而把嵌入式开发板(称为目标板)作为开发的测试平台，对编写的代码进行检验和调试。其连接方式如图 5.1 所示。

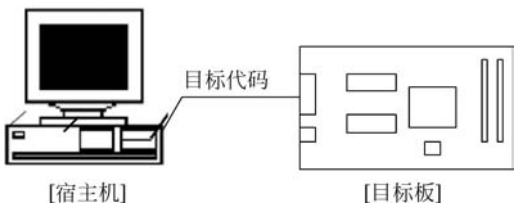

图 5.1 建立交叉编译的连接方式

开发时使用宿主机上的交叉编译、汇编及链接工具形成可执行的二进制代码，这种可执行代码并不能在宿主机上执行，而只能在目标板上执行。然后再把可执行文件下载到目标

机上运行。

5.1.2 建立交叉编译开发环境

对于嵌入式 Linux 的开发和应用,在进行开发前首要的工作就是要搭建一个完整的交叉编译开发环境。即在宿主机上安装与目标板相应的编译器及库函数,以便能在宿主机上应用开发工具编译在目标板上运行的 Linux 引导程序、内核、文件系统和应用程序。

1. 下载和安装 arm-linux-gcc 编译工具链

1) 下载 arm-linux-gcc

下载或复制 arm-linux-gcc 到当前目录下。例如,本书使用的 arm-linux-gcc 版本为 arm-linux-gcc-4.5.1.tar.gz。

2) 解压 arm-linux-gcc

应用下列解压命令。

```
#tar -zxvf arm-linux-gcc-4.5.1.tar.gz
```

解压后,将 arm-linux 文件目录及其所有文件复制到/usr/local/下。

```
#cp -rvarm-linux /usr/local/
```

现在交叉编译工具都在/usr/local/arm-linux/bin 目录下了。

这时,可以用 ls 命令查看安装在/usr/local/arm-linux/bin 目录下的嵌入式系统开发交叉编译器,显示情况如下。

```
[root@localhostroot]# cd /usr/local/arm-linux/bin
[root@localhost bin]# ls
arm-linux-addr2line  arm-linux-cpp          arm-linux-gcov     arm-linux-ranlib
arm-linux-ar         arm-linux-g++          arm-linux-ld       arm-linux-readelf
arm-linux-as         arm-linux-gcc          arm-linux-nm       arm-linux-size
arm-linux-c++        arm-linux-gcc-4.4.3    arm-linux-objcopy  arm-linux-strings
arm-linux-c++filt    arm-linux-gccbug       arm-linux-objdump  arm-linux-strip
```

2. 在系统配置文件 profile 中设置环境变量

为了可以在所有目录下直接使用 arm-linux-gcc 这个工具,需要修改 profile 文件。

1) 方法一

在 profile 文件中加入搜索路径 pathmunge /usr/local/arm-linux/bin。

命令如下所示。

```
[root@localhost root]#gedit  /etc/profile
```

这时在终端窗口显示如下(其中,pathmunge /usr/local/arm-linux/bin 为新加入)。

```
…
# Path manipulation
if [ 'id -u' = 0 ]; then
            pathmunge /sbin
            pathmunge /usr/sbin
            pathmunge /usr/local/sbin
            pathmunge /usr/local/arm-linux/bin
```

```
fi
unset pathmunge
```

关于 if 语句的使用参见教材第 4 章的“Linux shell 编程”。

2）方法二

修改环境变量，把交叉编译器的路径加入 PATH 中。

在 profile 文件的最后加入搜索路径。

```
export  PATH = $ PATH: /usr/local/arm - linux/bin
export  PATH
```

3. 立即使新的环境变量生效，不用重启计算机

1）运行 source 命令，使设置生效

```
# source  /etc/profile
```

2）检查是否将路径加入 PATH 中

```
# echo  $ PATH
```

显示的内容中有/usr/local/arm-linux/bin，说明已经将交叉编译器的路径加入 PATH。至此，交叉编译环境安装完成。

3）测试是否安装成功

```
# arm - linux - gcc  - v
```

如果该命令能显示交叉编译工具的版本情况，如图 5.2 所示，则表明安装成功。

图 5.2　测试交叉编译工具是否安装成功

4. 编译 Hello World 程序,测试交叉工具链

编写下面的 Hello World 程序,保存为 hello.c:

```
#include <stdio.h>
int main()
{
   printf("Hello World!\n");
   return 0;
}
```

执行下面的命令。

```
# arm-linux-gcc -o hello hello.c
```

若源程序有错误,则会有提示;若没有任何提示,表示通过了编译,就可以下载到 ARM 开发板上运行了。

接着可以输入 file hello 命令,查看生成的 hello 文件的类型。要注意的是,生成的可执行文件只能在 ARM 体系下运行,不能在 PC 上运行。

如果是建立 S3C2410 系统的主机开发环境,在运行完毕开发商提供的安装脚本程序/install.sh 后,将在根目录生成一个/linuette 目录,该目录为 S3C2410 系统的工作目录,与安装 PXA270 系统所生成的/pxa270_linux 目录类似。

另外,生成一个 /opt/host/armv4l 目录(目录名 armv4l 最后的是字母 l,不是数字 1),主编译器为 armv4l-unknown-linux-gcc。

要建立主编译器的搜索路径,则修改/root/.bash_profile 文件,将文件中的 PATH 变量值设为 PATH=$PATH:$HOME/bin:/opt/host/armv4l/bin。存盘后,执行 source /root/.bash_profile,这时设置的搜索路径生效。


视频讲解

5.2 配置超级终端 minicom

Linux 系统的 minicom 很像 Windows 系统的超级终端,它是一个串口通信工具。可以利用 minicom 作为在宿主机端与目标板进行通信的终端。下面介绍 minicom 的配置方法。

(1) 在宿主机 Linux 终端中输入 minicom-s 或 minicom,然后再按 Ctrl+A+O 快捷键。弹出超级终端 minicom 的设置菜单,如图 5.3 所示。

(2) 选择 Serial port setup 项,首先选择串口,如果使用第 1 个串口,则设置串口号为 ttyS0;如果使用第 2 个串口,则将串口号设为 ttyS1。

(3) 将串口配置为:波特率 115200,8 位数据位,1 位停止位,没有数据流控制。如图 5.4 所示,按 A 键进行端口号配置,按 E 键进行串口配置。

(4) 选择 Save setup as dfl 项,将设置保存为默认值(这里,dfl 代表 default),如图 5.5 所示。

(5) 选择 Exit 退回到 minicom 界面。

正确连接串口线,PC 端使用在 MINICOM 中被配置的串口(ttyS0 或 ttyS1),目标板使

用第 1 个串口(电路板上标示为 SERIAL PORT 0)或第 2 个串口(电路板上标示为 SERIAL PORT 1)。

图 5.3　超级终端 minicom 配置窗口

图 5.4　设置串口为 ttyS0,波特率为 115200

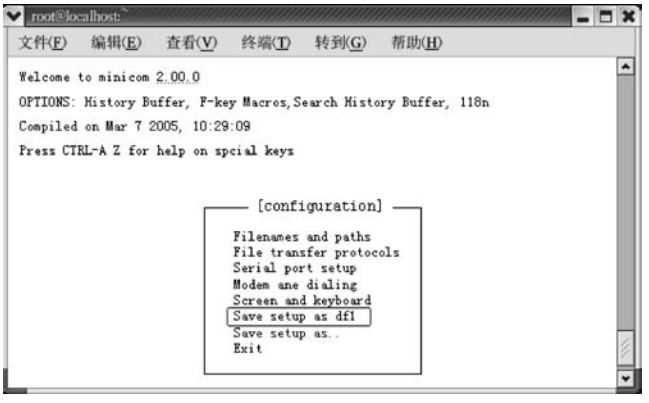

图 5.5　保存为 df1


视频讲解

5.3 编译嵌入式 Linux 系统内核

完成了主机的开发环境搭建等前期准备工作之后,接下来,就可以编译嵌入式 Linux 的内核了。本节主要介绍嵌入式 Linux 内核的编译过程。

编译嵌入式 Linux 内核都是通过 make 的不同命令来实现的,它的执行配置文件是 Makefile。Linux 内核中不同的目录结构里都有相应的 Makefile,而不同的 Makefile 又通过彼此之间的依赖关系构成统一的整体,共同完成建立依存关系、建立内核等功能。

编译内核需要 3 个步骤,分别是内核配置、建立依存关系、建立内核。下面分别讲述这 3 个步骤。

5.3.1 内核裁剪配置

1. 确定处理器类型

编译内核的第一步是根据目标板微处理器类型来确定微处理器架构,不同的微处理器架构在编译内核时会有不同的处理器选项。例如,ARM 就有其专用的选项,如 Multimedia capabilities port drivers 等。因此,在此之前,必须在 ARM 系统文件的根目录中的 Makefile 下为 ARCH 设定目标板微处理器的类型值。例如:

```
ARCH := arm
```

或输入命令进行设置。

```
[root@localhostlinux]# export  ARCH=arm
```

2. 确定内核配置方法

内核支持 4 种不同的配置方法,这 4 种方法只是与用户交互的界面不同,其实现的功能是一样的。每种方法都会读入和修改内核源码根目录下的一个默认的.config 配置文件(该文件是一个隐藏文件)。这 4 种方式简介如下。

(1) make config:基于文本的最为传统的配置界面,不推荐使用。

(2) make menuconfig:基于文本菜单的配置界面,字符终端下推荐使用。

(3) make xconfig:基于图形窗口模式的配置界面,Xwindow 下推荐使用。

(4) make oldconfig:自动读入.config 配置文件,并且只要求用户设定前次没有设定过的选项。

在这 4 种模式中,make menuconfig 的使用最为广泛。

【例 5-1】 以 make menuconfig 为例进行 S5PV210 系统的内裁剪核配置。

(1) 首先下载或复制适合开发板微处理器型号的 Linux 内核源码,进入内核源码的系统根目录。

(2) 运行 make menuconfig 命令,弹出内核裁剪配置窗口,如图 5.6 所示。

```
[root@localhostlinux]# make menuconfig
```

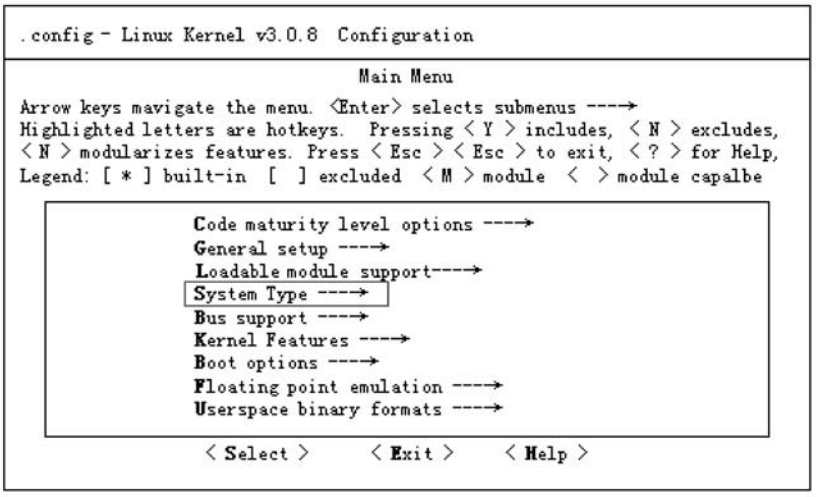

图 5.6　make menuconfig 内核裁剪配置界面

从图 5.6 可以看出，Linux 内核允许用户对其各类功能逐项配置，共有 19 类配置选项，如表 5.1 所示。

表 5.1　内核裁剪配置选项表

序号	选 项 名 称	说　　明
1	Code maturity level options	代码成熟度选项。当内核中包含有不成熟的代码或驱动程序进行调试时，一般选择该项
2	General setup	通用选项。例如，进程间通信方式等
3	Loadable module support	系统模块选项。提供系统模块支持
4	Block layer	系统调度方式选项
5	System Type	系统类型选项，提供微处理器型号及特性配置
6	Bus Support	提供总线接口支持选项
7	Kernel Features	内核特性选项
8	Boot options	内核启动选项
9	Floating point emulation	提供和浮点运算相关选项
10	Userspace binary formats	用户空间使用的二进制文件格式选项
11	Power management options	电源管理选项
12	Networking	网络协议相关选项
13	Device drivers	提供所有设备驱动程序相关的配置选项
14	File System	提供文件系统的配置选项
15	Profiling support	提供和程序性能分析相关的选项
16	Kernel hacking	与内核调试相关的选项
17	Security options	有关安全的配置选项
18	Cryptographic options	加密算法配置选项
19	Library routines	提供 CRC 校验的库选项

在嵌入式Linux的内核源码安装目录中通常有以下几个配置文件：.config、autoconf.h、config.h。其中，.config文件是make menuconfig默认的配置文件，位于源码安装目录的根目录中，autoconf.h和config.h是以宏的形式表示的内核的配置。当用户使用make menuconfig做了一定的更改之后，系统会自动在autoconf.h和config.h中做出相应的更改。后两个文件位于源码安装目录的/include/linux/下。

在menuconfig的配置界面中是纯键盘的操作，用户可使用上下键和Tab键移动光标以进入相关子项，图5.7所示为进入了System Type→子项的界面，该子项是一个重要的选项，主要用来选择处理器的类型。这里，带有→的选项表示当前项还有下一级菜单子项。

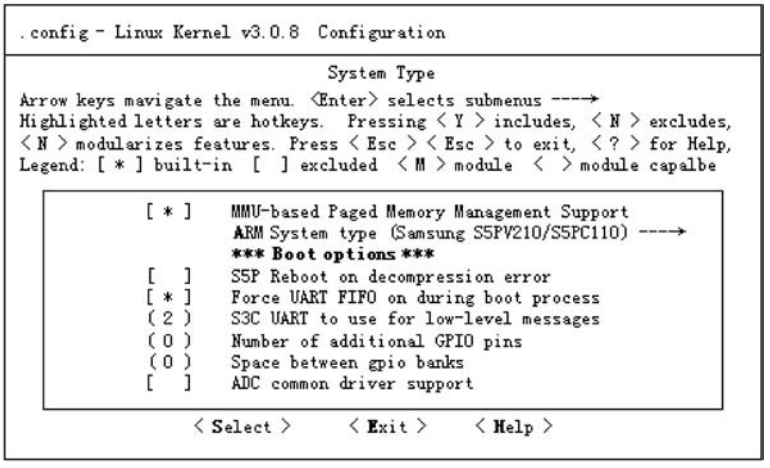

图5.7 System Type →子项的界面

每个选项前都有一对括号，可以通过按空格键或Y键表示包含该选项；按N表示不包含该选项。

选项前面的括号有3种形式，即中括号、尖括号或圆括号。

(1) [ ]表示该选项有两种选择。

- [*]表示选择该项编译进内核。
- [ ]表示不编译该选项。

(2) < >表示该选项有3种选择。

- < * >将该项选进内核。
- < M >将该项编译成模块，但不编译进内核。
- < <表示不编译该选项。

可以用空格键选择相应的选项。

(3) ()表示该项可以输入数值。

一般情况下，使用嵌入式系统设备厂商提供的默认配置文件都能正常运行，所以用户初次使用时可以不用对其进行额外的配置，在以后需要使用其他功能时再另行添加，这样可以大大减少出错的概率，有利于错误定位。在完成配置之后，就可以保存退出，如图5.8所示。

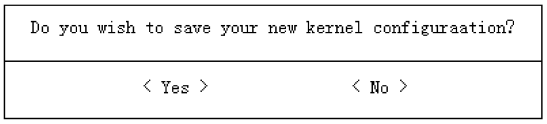

图 5.8　保存内核设置

5.3.2　内核编译

在嵌入式系统的项目设计过程中，经常需要修改及重新编译内核，下面介绍编译内核文件的方法。

1. 内核配置系统的基本结构

Linux 内核的配置系统由 3 个文件组成，对它们分别简介如下。

(1) Makefile：分布在 Linux 内核源代码根目录及各层目录中，定义 Linux 内核的编译规则。

(2) 内核配置菜单 Kconfig：它是配置界面的源文件，给用户提供配置选择的功能。

(3) 配置文件.config：编译内核所依据的配置项，决定将哪些驱动编译进内核。该文件通常由 menuconfig 生成。

2. 内核配置菜单 Kconfig

Kconfig 文件是 menuconfig 的关键文件。Kconfig 用来配置内核，它就是各种配置界面的源文件，内核的配置工具读取各个 Kconfig 文件，生成配置界面供开发人员配置内核，最后生成.config 配置文件。

Kconfig 文件的一般格式如下。

```
menuconfig "菜单入口名称"
   tristate  "菜单选项"
```

3. 编译内核的步骤

静态加载设备驱动程序需要以下几个步骤。

(1) 创建.config 配置文件。

(2) 编写编译驱动程序的 Makefile 文件。

(3) 修改上层目录中的 Kconfig 和 Makefile 文件。

(4) 运行内核配置界面 menuconfig，生成编译内核的配置文件.config。

(5) 运行内核源代码根目录下的 Makefile 或 build 文件，编译内核。

下面通过一个简单示例，说明把设备驱动程序编译到内核中的方法。

【例 5-2】 设有一个设备驱动程序 drv_led.c，将其编译到内核文件中。

把一个设备驱动程序编译到内核中的具体步骤如下。

(1) 建立工作目录。

为了防止破坏原生系统以及方便后期管理，在 Linux 系统内核源程序中，建立一个工作目录 menu_test。

```
/Linux-kernel/drivers/char/menu_test
```

(2) 创建 Kconfig 文件。

Kconfig 文件是 menuconfig 的关键文件。Kconfig 用来配置内核,它是配置界面的源文件,内核的配置工具读取各个 Kconfig 文件,生成配置界面供开发人员配置内核,最后生成配置文件.config。

下面是为了测试而创建的 Kconfig 文件,文件存放路径为/Linux-kernel/drivers/char/menu_test/Kconfig。

Kconfig 文件的代码如下。

```
menuconfig menu_test
    tristate " menu_test driver"

if menu_test
config LED
    tristate "LED driver"
config MOTO
    tristate "MOTO driver"
endif # menu_test
```

其中:

menuconfig menu_test 定义了要传给 Makefile 的参数 menu_test,在生成的.config 文件中会多一项 CONFIG_ menu_test;

tristate " menu_test driver"语句用来在执行 make menuconfig 命令时,为配置界面增加一个<> menu_test driver 选项。其中的 tristate 定义该选项为三态的,即可以有< >< * >和< M > 3 种状态。若用 bool 定义,则该选项只有两种状态两个选择,即[ ]和[*]。

if menu_test 和 endif 代码段只有在 menu_test 被定义时才执行。也就是说,必须是

```
< M > menu_test driver --- >
```

或

```
< * > menu_test driver --- >
```

时才会在配置界面中显示代码段中定义的内容。该代码段运行后的结果如图 5.9 所示。

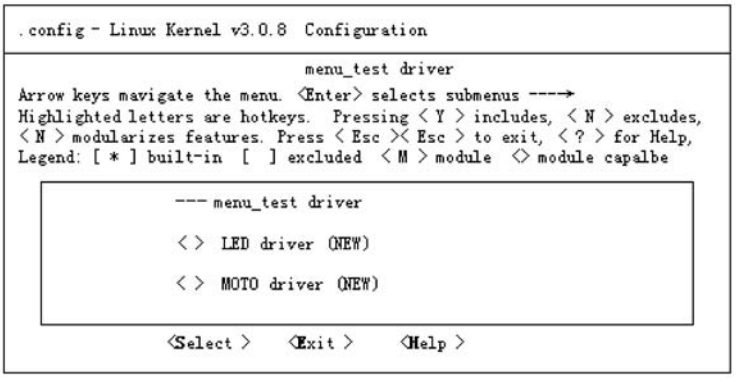

图 5.9 定义内核配置选项

(3) 创建 Makefile。

Makefile 是编译内核的重要文件。下面在/Linux-kernel/drivers/char/menu_test/目录下,创建 Makefile 文件,其代码如下。

```
obj- $(CONFIG_LED)          += drv_led.o
obj- $(CONFIG_MOTO)   += drv_moto.o
```

Makefile 中的每一项都对应 Kconfig 代码中的一个条目,例如:

```
obj- $(CONFIG_LED)          += drv_led.o
```

在 Kconfig 中对应的条目为:

```
config LED
    tristate "LED driver"
```

(4) 修改上层目录的 Kconfig。

工作目录下的 Kconfig 创建好之后,需要将它添加到原有内核的 Kconfig 中去。为此需修改上一层目录/Linux-kernel/drivers/char/下的 Kconfig 文件,将下面语句添加到该文件中。

```
source "drivers/char/menu_test/Kconfig"
```

一般是添加到文件的最后面,但需要注意的是,要在 endmenu 之前。Kconfig 文件中#开头的语句为注释语句,可以在 Kconfig 文件中添加一些自己的注释语句。

(5) 修改上层目录的 Makefile。

还需要把新的驱动程序添加到原有的 Makefile 中,这样在编译系统的时候才能编译新添加的驱动。修改上一层目录/Linux-kernel/drivers/char/下的 Makefile 文件,将下面内容添加到该文件中。

```
obj- $(CONFIG_menu_test)          += menu_test/
```

其中的 CONFIG_menu_test 对应的是/Linux-kernel/drivers/char/menu_test/Kconfig 中的语句:

```
menuconfig menu_test
    tristate " menu_test driver"
```

至此,已经将设备驱动程序 drv_led.o 添加进 menuconfig 中了。

(6) 运行 menuconfig。

执行命令:

```
make menuconfig
```

在弹出的配置窗口中,选择 Device Drivers →项,再在新进入的选项窗口中选择 Character devices 项,可以看到新建的< > menu_test driver(NEW)选项,如图 5.10 所示。

(7) 做好相应的配置之后,保存配置,在 Linux 内核根目录下便会生成.config 文件。在如图 5.11 所示的提示框中,选择< Yes >项,则选择的设备驱动程序项目编译到.config 内核配置文件中。

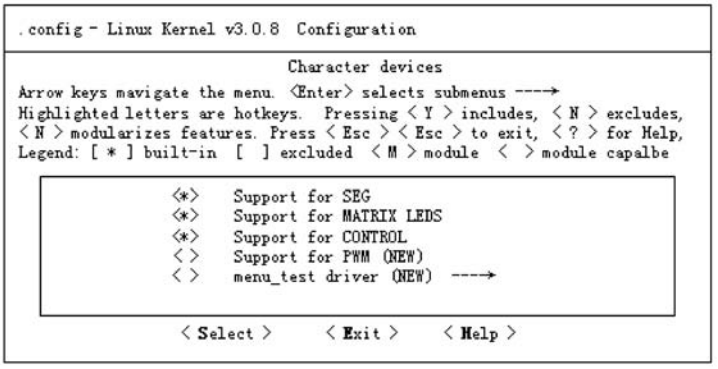

图 5.10 选择所定义的内核配置选项标题

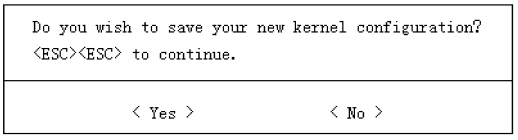

图 5.11 确认把设备驱动程序保存到编译内核的配置文件中

(8) 在内核源代码的根目录,运行 Makefile 或 build 文件,则把.config 系统内核配置文件中所有项目均编译并打包压缩到 zImage 文件中。

5.4 文件系统的制作

文件系统是嵌入式 Linux 系统必备的一个组成部分,是系统文件和应用文件存储的地方。通常使用的 PC 上的文件系统包括很多功能,容量达几百兆字节之多,在嵌入式系统中要使用这样的文件系统是不可能的。因此,嵌入式系统中的文件系统是一个 Linux 文件系统的简化版。文件系统中仅包含必需的目录和文件,完成需要的功能即可。

下面对文件系统中需要包含的目录和文件进行简要的说明。

1. 文件目录

文件系统要求建立的目录有/bin、/sbin、/etc、/dev、/lib、/mnt、/proc 和/usr。

- /bin:目录下需要包含常用的用户命令,如 sh 等。
- /sbin:目录要包含所有系统命令,如 reboot 等。
- /etc:目录下是系统配置文件。
- /boot:目录下是内核映像。
- /dev:目录包含系统所有的特殊设备文件。
- /lib:目录包含系统所有的库文件。
- /mnt:目录只用于挂接,可以是空目录。
- /proc:目录是/proc 文件系统的主目录,包含了系统的启动信息。
- /usr:目录包含用户选取的命令。

2. 文件目录应该包含的文件和子目录

1）目录/bin

目录/bin 中至少应包含命令文件 date、sh、login、mount、umount、cp、ls、ftp 和 ping。这些命令文件的主要作用如下。

- date：查取系统时间值。
- sh：是 bash 的符号链接。
- login：登录进程启动后，若有用户输入，此程序就提供 password 提示符。
- mount：挂接根文件系统时使用的命令，有些 Linux 开发商将此文件安排在/sbin 下。
- umount：卸载文件系统时使用的命令。
- cp：文件复制命令。
- ls：列出目录下的文件需使用的命令。
- ftp：根据文件传输协议实现的命令，可以用于 FTP 登录。
- ping：基本的网络测试命令，运行在网络层。

2）目录/sbin

目录/sbin 至少应包含命令文件 mingetty、reboot、halt、sulogin、update、init、fsck、telinit 和 mkfs。这些命令的主要作用如下。

- reboot：系统重新启动的命令。
- halt：系统关机命令，它与 reboot 共享运行的脚本。
- init：它是最早运行的进程，从 Start_kernel()函数中启动。此命令可以实现 Linux 运行级别切换。

3）目录/etc

目录/etc 至少应包含配置文件 HOSTNAME、bashrc、fstab、group、inittab、nsswitch、pam. d、passwd、pwdb. conf、rc. d、securetty、shadow、shells 以及 lilo. conf。这些配置文件的主要作用如下。

- HOSTNAME：用于保存 Linux 系统的主机名。
- fstab：用于保存文件系统列表。
- group：用于保存 Linux 系统的用户组。
- inittab：用于决定运行级别的脚本。
- passwd：保存了所有用户的加密信息。
- shadow：密码屏蔽文件。
- shells：支持的所有 Shell 版本。

4）目录/dev

目录/dev 至少应包含设备文件 console、hda1、hda2、hda3、kmem、mem、null、tty1 和 ttyS0。这些特殊设备文件的作用如下。

- console：表示控制台设备。
- hda1：表示第 1 个 IDE 盘的第 1 个分区。
- hda2：表示第 1 个 IDE 盘的第 2 个分区。
- hda3：表示第 1 个 IDE 盘的第 3 个分区。
- kmem：描述内核内存的使用信息。

- mem：描述内存的使用信息。
- null：表示 Linux 系统中的空设备，可用于删除文件。
- tty1：第 1 个虚拟字符终端。
- ttyS0：第 1 个串行口终端。

5）目录/lib

目录/lib 至少应包含库文件 libc. so. 6、ld-linux. so. 2、libcom_err. so. 2、libcrypt. so. 2、libpam. so. 0、libpam_misc. so. 2、libuuid. so. 2、libnss_files. so. 2、libtermcap. so. 2 和 security。这些库文件的作用如下。

- libc. so. 6：Linux 系统中所有命令的基本库文件。
- ld-linux. so. 2：基本库文件 libc. so. 6 的装载程序库。
- libcom_err. so. 2：对应命令出错处理的程序库。
- libcrypt. so. 2：对应加密处理的程序库。
- libpam. so. 0：对应可拆卸身份验证模块的程序库。
- libpam_misc. so. 2：对应可拆卸身份验证模块解密用的程序库。
- libuuid. so. 2：对应于身份识别信息程序库。
- libnss_files. so. 2：对应名字服务切换的程序库。
- libtermcap. so. 2：用于描述终端和脚本的程序库。
- security：此目录用来提供保证安全性所需的配置，与 libpam. so. 0 配合使用。

6）目录/mnt 和/proc

目录/mnt 和/proc 可以为空。

3. 制作文件系统的镜像文件

Linux 支持多种文件系统，同样，嵌入式 Linux 也支持多种文件系统。虽然嵌入式系统由于资源受限，它的文件系统和 Linux 的文件系统有较大的区别（前者往往是只读文件系统），但是，它们的总体架构是一样的，都是采用目录树的结构。下面介绍在嵌入式 Linux 中常见的文件系统 cramfs 及 jffs2 文件系统的镜像文件的制作。

【例 5-3】 制作 cramfs 文件系统。

cramfs 文件系统是一种经压缩的、极为简单的只读文件系统，因此非常适合嵌入式系统。要注意的是，不同的文件系统都有相应的制作工具，但是其主要的原理和制作方法是类似的。

制作 cramfs 文件系统需要用到的工具是 mkcramfs，下面就来介绍使用 mkcramfs 制作文件系统映像的方法。

假设用户已经在目录/fs/root/下建立了一个文件系统，/fs/root 目录下的文件如下所示。

```
[root@localhostfs]# ls  root
bin dev etc home lib linuxrc proc Qtopia ramdisk sbin tmp usr var
```

接下来就可以使用 mkcramfs 工具了，命令格式如下。

```
mkcramfs  系统文件目录名  生成的镜像文件名
```

设当前目录为/fs，现将系统文件子目录 root 生成镜像文件 camare_rootfs. cramfs：

```
[root@localhost fs]# ./mkcramfsroot camare_rootfs.cramfs
```

则在当前目录/fs下生成了系统文件的镜像文件camare_rootfs.cramfs，mkcramfs在制作文件镜像的时候对该文件进行了压缩。

【例5-4】 制作jffs2文件系统。

jffs2是一种可读/写的文件系统。制作它的工具叫作mkfs.jffs2，可以用下面的命令来生成一个jffs2的文件系统。

```
#./mkfs.jff2 -r [系统文件目录] -o [系统文件名.jffs2] -e [烧写到flash的地址]
-p=[文件长度]
```

设在/fs/root目录中有文件系统的源文件，烧写命令如下。

```
[root@localhostfs]# ls root
bin etc mnt root sbin usr dev lib proc tmp var

[root@localhostfs]# ./mkfs.jffs2 -r root -o xscale_fs.jffs2 -e 0x40000 -p=0x01000000
```

这样，就会在/fs目录下生成一个文件名为xscale_fs.jffs2的文件系统镜像文件。

5.5 嵌入式系统开发板的烧写方法

嵌入式linux系统可以安装在开发板上，在嵌入式开发板上安装Linux的过程称为“烧写系统”或“刷机”。下面介绍在嵌入式系统开发板上烧写linux系统的方法。

5.5.1 引导加载程序 Bootloader

引导加载程序从开发板通电到应用程序开始工作，经历了一个非常复杂的加载过程，下面简单介绍这个加载过程。

1. 基本概念

一个嵌入式Linux系统从软件的角度看通常分为4个层次：引导加载程序Bootloader、Linux内核、文件系统和用户应用程序。

简单地说，Bootloader就是在操作系统内核运行之前运行的一段程序，它类似于PC机中的BIOS程序。通过这段程序，可以完成硬件设备的初始化，并建立内存空间的映射图的功能，从而将系统的软硬件环境带到一个合适的状态，为最终调用系统内核做好准备。

通常，Bootloader严重地依赖于硬件设备。在嵌入式系统中，不同体系结构使用的Bootloader是不同的。除了体系结构，Bootloader还依赖于具体的嵌入式板级设备的配置。也就是说，对于两块不同的嵌入式开发板而言，即使它们基于相同的CPU构建，运行在其中一块电路板上的Bootloader，未必能够在另一块电路开发板上运行。因此，在嵌入式世界里建立一个通用的Bootloader几乎是不可能的。尽管如此，仍然可以对Bootloader归纳出一些通用的概念来指导用户完成特定的Bootloader设计与实现。

1) Bootloader所支持的CPU和嵌入式开发板

每种不同的CPU体系结构都有不同的Bootloader。有些Bootloader也支持多种体系结构的CPU，如后面要介绍的U-Boot就同时支持ARM体系结构和MIPS体系结构。除

了依赖于CPU的体系结构外,Bootloader实际上也依赖于具体的嵌入式板级设备的配置。

2) Bootloader的安装媒介

系统加电或复位后,所有的CPU通常都从某个由CPU制造商预先安排的地址上取指令。基于CPU构建的嵌入式系统通常都有某种类型的固态存储设备(如ROM、EEPROM或FLASH)被映射到这个预先安排的地址上。因此,在系统加电后,CPU将首先执行Bootloader程序。

3) Bootloader的启动过程

Bootloader的启动过程分为单阶段和多阶段两种。通常多阶段的Bootloader能提供更为复杂的功能,以及更好的可移植性。

4) Bootloader的操作模式

大多数Bootloader都包含两种不同的操作模式:启动加载模式和下载模式,这种区别仅对于开发人员才有意义。从最终用户的角度看,Bootloader的作用就是用来加载操作系统,而并感觉不到所谓的启动加载模式与下载工作模式的区别。

(1) 启动加载模式:这种模式也称为"自主"模式。也就是Bootloader从目标板上的某个固态存储设备(如Flash)上将操作系统加载到RAM中运行,整个过程没有用户的介入。这种模式是嵌入式产品发布时的通用模式。

(2) 下载模式:在这种模式下,目标机上的Bootloader将通过串口连接或网络连接等通信手段从宿主机下载文件。例如,下载内核映像文件和文件系统映像文件等。从主机下载的文件通常首先被Bootloader保存到目标板的RAM中,然后再被Bootloader写到目标板上的Flash之类固态存储设备中。Bootloader的这种模式通常在系统更新时使用。工作在这种模式下的Bootloader通常都会向它的终端用户提供一个简单的命令行接口,如vivi、Blob等。

5) Bootloader与主机之间进行文件传输所用的通信设备及协议

最常见的情况是,目标机上的Bootloader通过串口与主机之间进行文件传输,传输协议通常是xmodem、ymodem、zmodem协议中的一种。但是,串口传输的速度是有限的,因此通过以太网连接并借助TFTP协议来下载文件是个更好的选择。

2. Bootloader启动流程

Bootloader的启动流程一般分为两个阶段:stage1和stage2,下面分别对这两个阶段进行讲解。

1) Bootloader的stage1

在stage1中Bootloader主要完成以下工作。

(1) 基本的硬件初始化,包括屏蔽所有的中断、设置CPU的速度和时钟频率、RAM初始化、初始化LED、关闭CPU内部指令和数据cache等。

(2) 为加载stage2准备RAM空间,通常为了获得更快的执行速度,把stage2加载到RAM空间中来执行,因此必须为加载Bootloader的stage2准备好一段可用的RAM空间范围。

(3) 复制stage2到RAM中,在这里要确定以下两点。

· stage2的可执行映像在固态存储设备的存放起始地址和终止地址。

· RAM 空间的起始地址。

(4) 设置堆栈指针 sp,这是为执行 stage2 的 C 语言代码做好准备。

2) Bootloader 的 stage2

stage2 的代码通常用 C 语言来实现,以实现更复杂的功能和取得更好的代码可读性与可移植性。但是与普通 C 语言应用程序不同的是,在编译和链接 Bootloader 这样的程序时,不能使用 glibc 库中的任何支持函数。

在 stage2 中 Bootloader 主要完成以下工作。

(1) 初始化本阶段要使用到的硬件设备,包括初始化串口、初始化计时器等。在初始化这些设备之前,可以输出一些打印信息。

(2) 检测系统的内存映射,内存映射是指在整个内存物理地址空间中指出哪些地址范围被分配用来寻址系统的 RAM 单元。

(3) 加载内核映像和根文件系统映像,这里包括规划内存占用的布局和从 Flash 上复制数据。

(4) 设置内核的启动参数。

3. Bootloader 的烧写

要在嵌入式系统开发板中完成 Linux 操作系统的烧写,首先要把 Bootloader 烧写到 Flash 中。这时,要用到 JTAG。

用并口线通过 JTAG 小板将宿主机(打印输出口)与开发板连接,接线方法如图 5.12 所示。

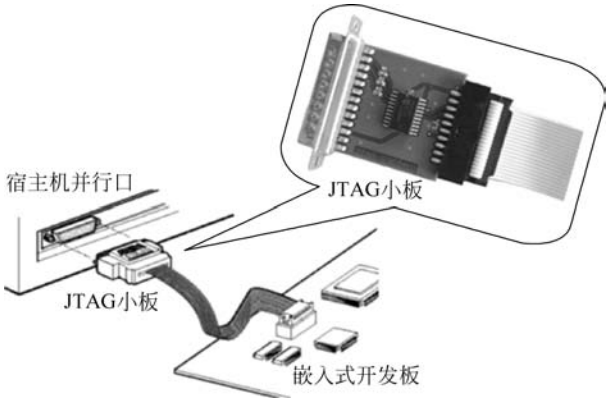

图 5.12　应用 JTAG 烧写 Bootloader 的接线图

由于 Bootloader 严重地依赖于硬件设备,不同微处理器的 Bootloader 是不相同的。表 5.2 列出了几种典型的微处理器开发板所使用的 Bootloader。

表 5.2　几种典型的微处理器开发板所使用的 Bootloader

微处理器型号	使用的 Bootloader
ARM　S3C2410	vivi
XSCALE　PXA270	blob
ARM Cortex A8	U-Boot


视频讲解

5.5.2 ARM Cortex A8 内核开发板的烧写

本节主要介绍基于 ARM Cortex A8 微处理器开发板 S5PV210 的 Bootloader、内核、文件系统的烧写方法。其系统引导程序 Bootloader 为 U-Boot。

1. U-Boot 简介

U-Boot(全称 Universal Boot Loader)是德国 DENX 小组开发的 Bootloader 项目,它支持数百种嵌入式开发板和各种微处理器。U-Boot 有以下两种操作模式。

1) 启动加载(Boot loading)模式

启动加载模式是 Bootloader 的正常工作模式,在嵌入式产品正式发布时,Bootloader 必须工作在这种模式下,Bootloader 将嵌入式操作系统从 Flash 中加载到 SDRAM 中运行,整个过程是自动的。

2) 下载(Downloading)模式

下载模式是 Bootloader 通过某些通信手段,将内核映像或根文件系统映像等从 PC 机中下载到目标板的 Flash 中。用户可以利用 Bootloader 提供的一些命令接口来完成自己想要的操作。

U-Boot 的常用命令见表 5.3。

表 5.3 U-Boot 常用命令

命　令	说　明
bootfile	默认的下载文件名
bootcmd	自动启动时执行命令
serverip	TFTP 服务器端的 IP 地址
ipaddr	本地的 IP 地址
setenv	设置环境变量
saveenv	保存环境变量到 nand Flash 中
nand scrub	擦除指令,擦除整个 nand Flash 的数据
loadlinuxramdisk	从网络 TFTP 服务器下载内核及文件系统到内存中

2. 主要烧写步骤

对于基于 Cortex A8 微处理器的 S5PV210 开发板,嵌入式 Linux 系统的 Bootloader、内核、文件系统将烧写到 Flash 中。烧写 S5PV210 开发板的步骤如下。

(1) 编译和启动系统引导程序 U-Boot。

(2) 通过 SD 卡把 u-boot 镜像文件写到 Flash 中。

(3) 通过 U-boot 烧写嵌入式 Linux 文件系统。

3. 在宿主机上编译 U-Boot

(1) 将开发板供应商提供的 u-boot-s5pv210.tar.gz 文件复制到 Linux 主机的工作目录下,用解压命令将其解压到 u-boot-s5pv210 目录。在这里,把 u-boot-s5pv210 目录设置为 U-Boot 的根目录,其命令如下。

```
# tar zxvf u-boot-s5pv210.tar.gz
# cd u-boot-s5pv210
```

（2）清理 U-Boot。对于已经运行过 U-Boot 的机器，需要先清理 u-boot，其命令如下。

```
# make clean
```

（3）检查 U-Boot 的交叉编译路径是否正确。打开 Makefile 文件，检查交叉编译路径是否配置正确。其配置交叉编译路径的代码为：

```
ifeq ( $ (ARCH), arm)
CROSS_COMPILE = arm - none - linux - gnueabi -
```

如图 5.13 所示。

（4）编译 U-Boot，生成镜像文件。在 U-Boot 的根目录（即 u-boot-s5pv210 目录），执行编译命令：

```
# make
```

编译完成之后，用 ls 列出文件内容，可以看到已经生成了系统引导程序的镜像文件 u-boot.bin，如图 5.14 所示。

图 5.13　检查 U-Boot 的交叉编译路径

图 5.14　查看系统生成的镜像文件 u-boot.bin

4. 制作启动 SD 卡

在 u-boot-s5pv210\sd_fusing 目录下，有一个名为 sd_fusing.sh 的脚本程序，该脚本程序可以对 SD 卡进行分区、格式化为 vfat 格式、写入 U-Boot 镜像文件等操作。

事先准备好一张空白的 SD 卡，在终端窗口进入 u-boot-s5pv210\sd_fusing 目录，执行制作启动 SD 卡的命令：

```
#./sd_fusing.sh /dev/sdb
```

其执行结果如图 5.15 所示。

然后在 SD 卡的根目录建立 sdfuse 目录，把镜像文件 u-boot.bin 复制到 SD 卡的 sdfuse 目录下。这样就制作好了一张可以启动的 SD 卡。

5. 通过 SD 卡把 U-Boot 镜像文件烧写到 Flash 中

1）从 SD 卡启动开发板

将串口线将宿主机的串口与开发板 UART 端口连接，将串口的波特率设置为 115200。然后再将开发板的启动模式设置为从 SD 卡启动模式，并把制作好的启动 SD 卡插到 MMC

```
root@VirtualOS:/home/cvtech/u-boot-s5pv210/sd_fusing# ./sd_fusing.sh /dev/sdb
/dev/sdb reader is identified.
make sd card partition
./sd_fdisk /dev/sdb
记录了1+0 的读入
记录了1+0 的写出
512字节(512 B)已复制，0.0950436 秒，5.4 kB/秒
mkfs.vfat -F 32 /dev/sdb1
mkfs.vfat 3.0.9 (31 Jan 2010)
BL1 fusing
记录了16+0 的读入
记录了16+0 的写出
8192字节(8.2 kB)已复制，3.38529 秒，2.4 kB/秒
u-boot fusing
记录了544+0 的读入
记录了544+0 的写出
278528字节(279 kB)已复制，32.8811 秒，8.5 kB/秒
U-boot image is fused successfully.
Eject SD card and insert it again.
root@VirtualOS:/home/cvtech/u-boot-s5pv210/sd_fusing#
```

图 5.15　制作启动 SD 卡

卡槽。

打开 S5PV210 开发板电源，当超级终端窗口显示 Hit any key to stop autoboot: 0 时，快速按空格键，屏幕显示 SMDKV210 # 的提示符，进入 U-Boot 命令行，如图 5.16 所示。

2）清空开发板 NAND Flash 数据

在超级终端窗口中的 SMDKV210 # 提示符输入后面，输入 nand　scrub 命令，将 NAND Flash 数据清空。

当出现 Really scrub this NAND flash? <y/N>提示时，按 y 键后，按 Enter 键确认，则完成清空 NAND Flash 数据的工作，如图 5.17 所示。

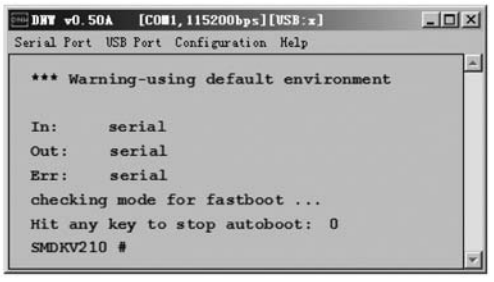

图 5.16　从 SD 卡启动，进入 U-Boot 命令行

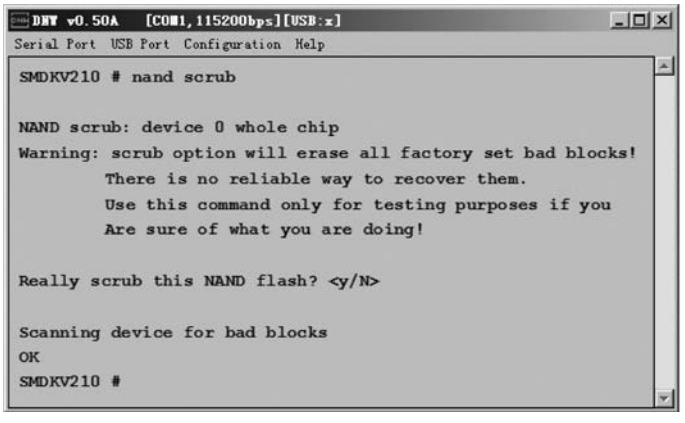

图 5.17　清空 NAND Flash 数据

3）把镜像文件 u-boot.bin 烧写到 NAND Flash 中

接上述步骤，继续执行 sdfuse flash bootloader u-boot.bin 命令，把 u-boot.bin 烧写到 NAND Flash 的 bootloader 分区中。

命令执行结果如图 5.18 所示。

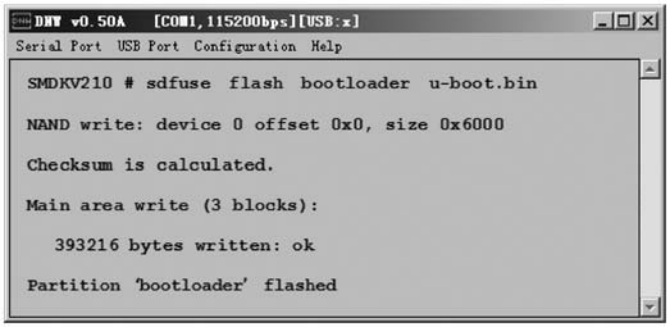

图 5.18　把镜像文件 u-boot.bin 烧写到 NAND Flash 中

至此，完成 U-Boot 的烧写。关闭开发板电源，从 MMC 卡槽中取出 SD 卡，将开发板的启动模式设置为从 NAND Flash 启动模式。重新将开发板电源打开，可以看到，开发板已经能正常启动。

6. 通过 U-Boot 烧写嵌入式 Linux 文件系统

在完成 U-Boot 烧写之后，使用串口线将宿主机与开发板的串口连接起来，再使用网线将宿主机与开发板的网口连接起来。

(1) 把要烧写的系统文件全部复制到 tftpd32.exe 文件所在的同一目录下。这些文件包括 zImage-ramdisk(内核映像文件)、ramdisk.gz(文件系统映像文件)。

(2) 打开超级终端后，开启开发板电源，U-Boot 启动过程中，按任意键，进入 U-Boot 命令行模式。

(3) 假设宿主机的 IP 地址设置为 192.168.0.100，运行 U-Boot 的 setenv　serverip 192.168.0.100 命令，在开发板上将宿主机的 IP 地址 192.168.0.100 设置为 TFTP 服务器的 IP 地址；然后运行 saveenv 命令将设置保存到环境变量中。命令执行结果如图 5.19 所示。

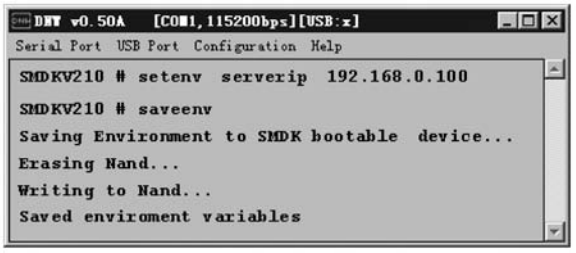

图 5.19　在开发板设置 TFTP 服务器地址

(4) 运行 U-BOOT 命令 sdfuse erase kernel；sdfuse erase system 擦除 kernel 区与 system 文件系统区数据，执行结果如图 5.20 所示。

(5) 运行 U-BOOT 命令 run loadlinuxramdisk，通过连接宿主机 TFTP 服务，将 Linux 内核与 ramdisk 文件系统下载到 SDRAM 中，命令执行情况如图 5.21 所示。

下载完成后，启动 Linux 文件系统，进入 Linux 命令行。至此，全部烧写工作结束。

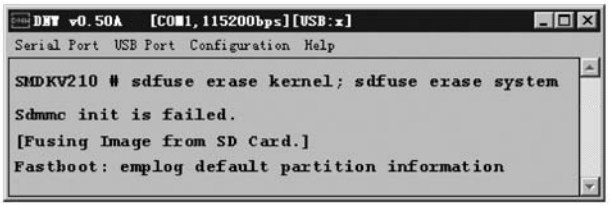

图 5.20 擦除开发板内核区及文件系统区数据

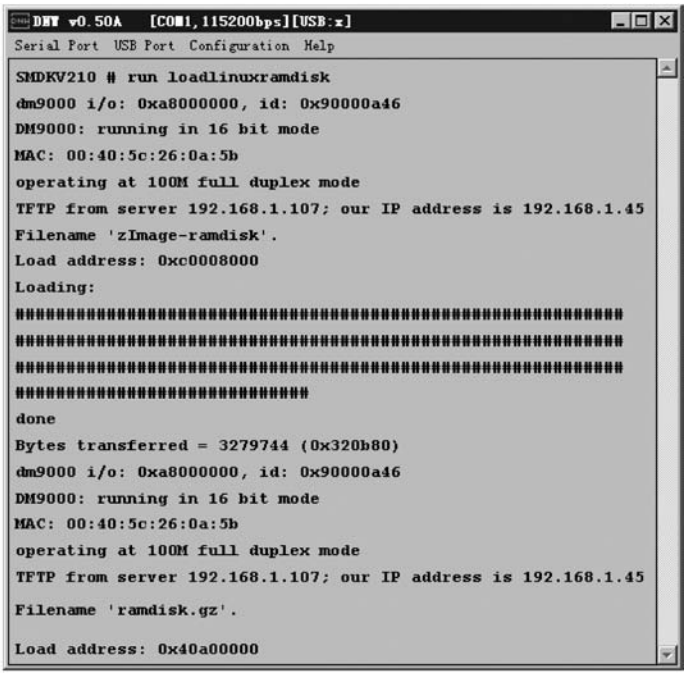

图 5.21 将 Linux 内核与 ramdisk 文件系统下载到 SDRAM 中

本章小结

本章详细讲解了在主机上建立嵌入式 Linux 开发环境,在主机端通过 minicom 终端窗口操作开发板的文件系统。详细讲解了建立嵌入式 Linux 的数据共享服务,包括 NFS 服务的配置,应用串口协议传输数据,在 VMware 虚拟机中设置 Windows——Linux 的数据共享。在本章还详细叙述了如何烧写开发板,如何移植嵌入式 Linux 内核以及如何制作文件系统。这些都是操作性很强的内容,而且在嵌入式的开发中也是必不可少的一部分,希望读者熟练掌握。

习　题

1. 仿照例 5-2,在 Linux 内核系统中建立一个名为 kernel_test 的项目,完成内核配置后,查看.config 文件中的配置结果。

2. 自己动手,完成对嵌入式开发板的系统烧写实验。