

# 基于天嵌科技的 **SKY2440/TQ2440** 的 **Linux-2.6.25.8** 系统移植手册

广州天嵌计算机科技有限公司荣誉出品

首发网站: [www.embedsky.net](http://www.embedsky.net)

# 版权声明

本手册版权归属广州天嵌计算机科技有限公司(以下简称“天嵌科技”)所有，并保留一切权力。非经天嵌科技同意(书面形式)，任何单位及个人不得擅自摘录本手册部分或全部内容，违者(我们)公司将追究其法律责任。

## 内容提要

本手册全面介绍了嵌入式 Linux 系统的开发过程中，各种驱动的源码获取、源码修改、配置和移植，驱动测试程序的编写，根文件系统的构造（包括移植 busybox、glibc、制作映像文件等），DIY 自己的驱动程序和测试程序的编写。

本手册从获取内核源码开始，一步一步的讲解一个完整的内核的移植，使读者最终可以配置、移植、裁剪内核，编写驱动程序，从而掌握整个嵌入式 Linux 系统的开发方法。

本手册由浅入深，循序渐进，非常适合刚接触嵌入式 Linux 的初学者学习，也可作为各种嵌入式培训教材使用。

# 前言

很久以前就想写一份详尽的移植手册了，一方面是为公司的 2440 开发板做配套使用教材，另外也是为广大的嵌入式爱好者提供更好，更详细的嵌入式学习参考资料。

刚好公司承接的一个项目要用到高版本的 Linux 内核，在完成该项目后，觉得很有必要把整个项目开发过程与广大的嵌入式爱好分享，于是便把整个开发过程中的记录整理成册，《基于天嵌科技的 SKY2440/TQ2440 的 Linux-2.6.25.8 系统移植手册》就这样诞生了。本手册中的所有程序和代码都是作者耗时，耗力的亲自调试成功的，希望该手册能够发挥应有的作用，为嵌入式行业发展贡献力量。

本手册的硬件平台是天嵌科技的 SKY2440 和 TQ2440 开发板，软件平台为 RedHat9.0 的交叉编译环境，硬件请参考 SKY2440 或 TQ2440 开发板配套光盘的原理图，本手册提到的 2.6.13 的文件系统请参考 SKY2440 或 TQ2440 开发板配套光盘里的文件系统。为了尽快让本手册和广大的天嵌科技的 ARM 开发板的用户以及其他想学习基于 ARM 的 Linux 开发的朋友见面，手册里面很多方面没有进行讲解，建议大家到我们的论坛和 QQ 讨论群中进行交流，共同学习和提高。有问题时，请在我们的论坛贴出该问题的截图之类的，方便他人参考和回答问题。对于自己已经解决问题的，也请贴出解决方法，以方便后来人。

本手册是天嵌科技将会推出的系列教程中的第一份教程，以后将会陆续推出无操作系统的程序开发实例方面的教材、bootloader 开发教程、uCOS-II 移植开发教程以及 WinCE 驱动及应用程序开发教程等涵盖整个嵌入式开发的系列教材。

由于水平有限，手册中难免遗漏和不足之处，恳请广大读者提出宝贵意见。

最后感谢广大的客户的支持，天嵌科技才能一步一步成长。

天嵌科技——黄健

2009-2-23

# 在本手册中的一些约定

## 约定 1

本手册首发地点为广州天嵌计算机科技有限公司的官方网站：[www.embedsky.net](http://www.embedsky.net)，同时配套的论坛为：[www.embedsky.net/bbs](http://www.embedsky.net/bbs)；建议广大读者到论坛发贴讨论。

## 约定 2

PC 的命令表示方法：在 PC 的终端使用的命令，均在其前面加 “#” 号并用红色表示。

比如：解压源码：`#tar xvfj linux-2.6.25.8.tar.bz2 -C /opt/EmbedSky/`

## 约定 3

开发板的命令表示方法：在开发板的终端（也就是 PC 的超级终端或别的串口软件）上面运行的基于开发板的命令，均在其前面加 “\$” 号并用红色表示。

比如：LCD 背光控制程序的操作：`$backlight off`

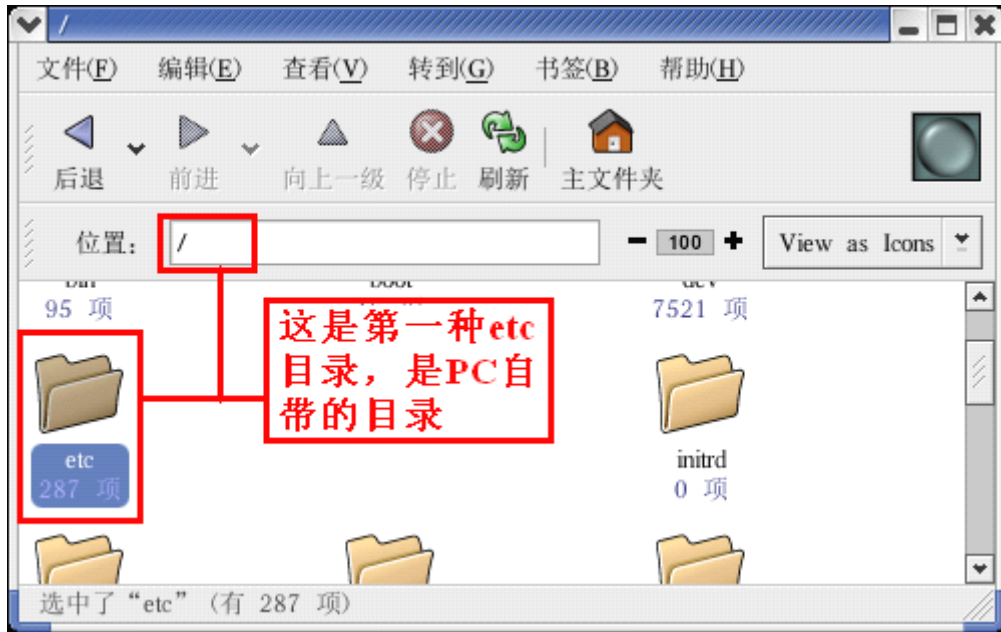
## 约定 4

关于路径的说明：“`/opt/EmbedSky/`” 这个就是绝对路径；内核文件 “`arch/arm/mach-s3c2440/ mach-s3c2440.c`” 这就是相对路径。区别在于最前面的那个 “/” 号。

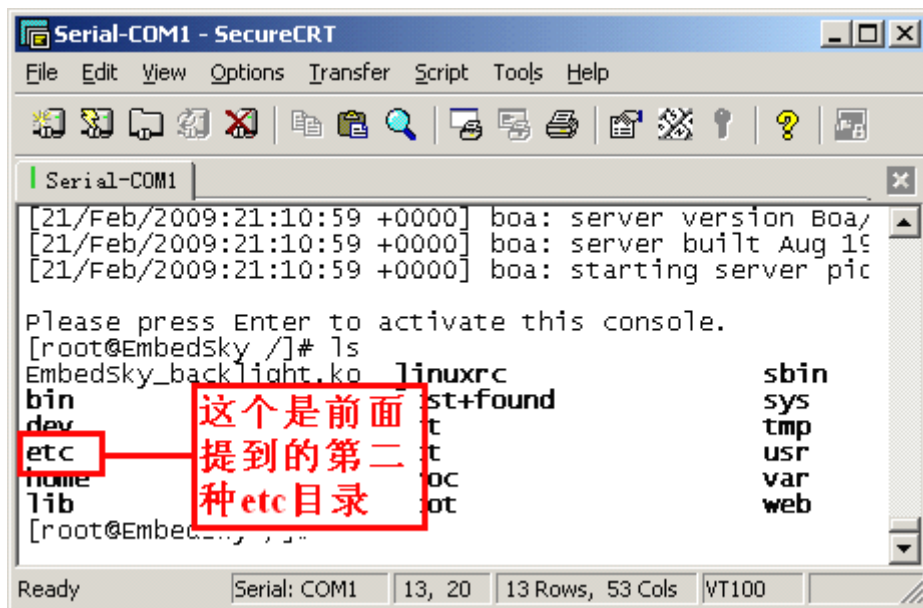
## 约定 5

文件所在地的区别：区别比如 “`/etc/`” 目录，在 PC 里面的 “`/etc/`”（1）目录就是绝对路径下的一个存在在 PC 上的目录；而开发板的文件系统的 “`/etc/`”（2）和 “`etc/`”（3）目录均是指的基于开发板的一个目录，差别在于一个是烧写到开发板上了，一个是还没有烧写到开发板上。下面的图应该能够说明其中的差别，请注意区分，我也会在本书中说明是 PC 的目录还是开发板的目录：

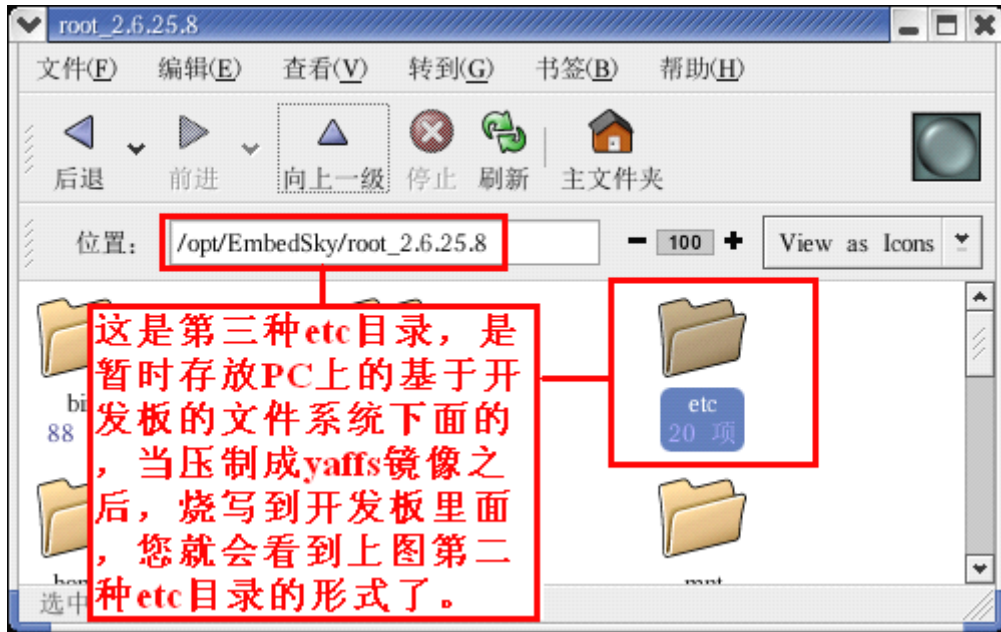
第一种 “`/etc/`” 目录的存在截图：



第二种“/etc/”目录的存在截图：



第三种“/etc/”目录的存在截图：



## 约定 6

本手册使用的编译环境是 RedHat9，所使用的交叉编译器为天嵌科技提供的 crosstools\_3.4.5\_softfloat 交叉编译器。

安装该编译器的方法：

从我们的官方网站获取交叉编译器的安装包，名为：crosstools-3.4.5\_softfloat.tar.bz2，然后将其解压到 RedHat9 的 “/opt/EmbedSky/” 目录下，使用命令：

```
#tar xvfj crosstools-3.4.5_softfloat.tar.bz2 -C /
```

解压完毕后，修改 PC 的 “/etc/profile” 文件，使用命令：

```
#gedit /etc/profile
```

然后打开该文件，在大概 21 行添加如下内容：（红色部分所示）

```
# Path manipulation
if [ `id -u` = 0 ]; then
    pathmunge /sbin
    pathmunge /usr/sbin
    pathmunge /usr/local/sbin
    pathmunge /opt/EmbedSky/crosstools_3.4.5_softfloat/gcc-3.4.5-glibc-2.3.6/arm-linux/bin
#   pathmunge /opt/EmbedSky/crosstools_3.4.1_softfloat/arm-linux/gcc-3.4.1-glibc-2.3.3/bin
#   pathmunge /usr/local/arm/3.3.2/bin
#   pathmunge /usr/local/arm/2.95.3/bin
Fi
```

**注意：**可能您哪里的内容和这里列出来的不太一样，只需要将自己添加的其他的交叉编译器前面加 “#” 屏蔽掉，仅留下 crosstools\_3.4.5\_softfloat 交叉编译器即可。

# 目录

版权声明.....	2
内容提要.....	3
前言.....	4
目录.....	8
Step 1、获取 Linux 系统源码.....	11
Step 2、解压系统源码.....	11
Step 3、在系统中添加对 ARM 的支持.....	11
Step 4、修改平台输入时钟.....	12
Step 5、制作 TQ2440/SKY2440 的配置单.....	14
Step 6、修改机器码.....	17
Step 7、编译镜像.....	18
7.1 镜像.....	18
7.2 把镜像存放到指定位置.....	18
让系统“跑”起来.....	20
Step 8、Nand Flash 驱动移植.....	20
8.1 完善源码.....	20
8.2 添加对应的驱动配置.....	21
8.3 驱动的使用情况.....	21
Step 9、移植 yaffs 文件系统.....	22
9.1 获取 yaffs 源码.....	22
9.2 在内核中添加对 yaffs 的支持.....	22
9.3 在配置单中添加对 yaffs 的支持.....	23
Step 10、编译 BusyBox.....	25
10.1 获取 BusyBox 源码.....	25
10.2 修改并配置 BusyBox.....	25
10.3 编译并安装 BusyBox.....	27
Step 11、构建文件系统.....	28
11.1 构建框架.....	28
11.2 添加内容.....	28
Step 12、完善串口驱动.....	34
12.1 修改源码.....	34
12.2 串口测试.....	34
Step 13、网卡驱动移植.....	36
13.1 驱动源码获取.....	36
13.2 修改驱动源码.....	36
13.3 配置并编译内核.....	39
13.4 网卡配置.....	40
13.5 向文件系统添加 FTP 和 TELNET 功能.....	41
Step 14、USB 设备驱动移植.....	45
14.1 USB 设备的配置.....	45



14.2 U 盘的挂载.....	45
14.3 实现把开发板当“U 盘”1——修改源码.....	46
14.4 实现把开发板当“U 盘”2——配置内核.....	47
14.5 实现把开发板当“U 盘”3——“U 盘”之又见“U 盘”.....	48
14.6 实现双 USB Host 功能 1——源码修改.....	49
14.6 实现双 USB Host 功能 2——配置并设置双 USB Host 口.....	50
14.7 USB 摄像头驱动移植 1——源码获取.....	50
14.8 USB 摄像头驱动移植 2——修改驱动源码.....	51
14.8 USB 摄像头驱动移植 3——配置内核.....	52
14.9 USB 摄像头驱动移植 4——USB 摄像头的测试.....	53
Step 15、LCD 驱动移植.....	55
15.1 LCD 时钟计算方法分析.....	55
15.2 简化 LCD 时钟计算方法.....	55
15.3 修改 LCD 参数设置.....	57
15.4 添加对多种 LCD 的支持.....	58
15.5 制作 Linux 的开机 logo 之 1——图片处理.....	62
15.6 制作 Linux 的开机 logo 之 2——支持更多 LCD 的设置.....	65
15.6 配置内核.....	67
15.7 LCD 背光控制之 1——添加驱动代码.....	68
15.7 编写 LCD 背光控制程序.....	69
15.8 测试 LCD 背光开关.....	71
Step 16、触摸驱动移植.....	72
16.1 添加触摸驱动补丁.....	72
16.2 添加 TSDEV 的补丁.....	74
16.3 配置内核.....	76
16.4 文件系统里面添加 Qte.....	76
Step 17、声卡驱动移植.....	78
17.1 添加声卡驱动补丁.....	78
17.2 修改内核 DMA 支持.....	79
17.3 配置内核.....	79
17.4 测试声卡.....	79
17.5 添加 Madplay 到文件系统.....	80
Step 18、RTC 驱动移植.....	82
18.1 添加平台对 RTC 的支持.....	82
18.2 配置内核.....	82
18.3 设置并测试 RTC.....	82
Step 19、看门狗驱动移植.....	85
19.1 启动看门狗.....	85
19.2 配置内核.....	85
19.3 没有喂狗的情况.....	85
19.4 喂狗程序的编写.....	85
19.5 文件系统里面实现喂狗操作.....	87
Step 20、SD 卡驱动移植.....	89
20.1 添加 SD 卡的驱动.....	89

20.2 添加 SD 设备到设备列表.....	89
20.3 去除 SD 驱动 bug.....	90
20.4 配置内核.....	90
20.5 挂载 SD 卡.....	90
20.6 去除多余信息.....	91
DIY 驱动到系统中.....	92
Step 21、编写第一个驱动程序.....	92
21.1 编写第一个驱动.....	92
21.2 在内核源码中添加对 hello 驱动的支持.....	93
21.3 配置内核.....	94
Step 22、GPIO 口控制驱动的编写.....	95
22.1 硬件分析.....	95
22.2 去除内核自带的 LED 灯驱动.....	95
22.3 编写自己的 LED 驱动.....	98
22.4 在内核源码中添加对 LED 灯驱动的支持.....	101
22.5 编写 LED 灯控制程序.....	103
Step 23、中断处理的驱动编写.....	106
23.1 硬件分析.....	106
23.2 编写键盘驱动.....	107
23.3 在内核源码中添加对按键驱动的支持.....	113
23.4 配置内核.....	114
23.5 编写按键测试程序.....	114
Step 24、PWM 控制器的驱动编写.....	117
24.1 硬件分析.....	117
24.2 编写 PWM 驱动源码.....	117
24.3 在内核源码中添加对 Beep 的支持.....	121
24.4 配置内核.....	122
24.5 编写 PWM 测试程序.....	123
Step 25、内核裁剪.....	126
结束语.....	128

# 让系统“动”起来

## Step 1、获取 Linux 系统源码

下载内核源码：<ftp://ftp.kernel.org/pub/linux/kernel/v2.6/linux-2.6.25.8.tar.bz2>，这里我选择了 2.6.25.8 的内核，因为最近给客户做项目用到这个版本的内核，就采用了它。

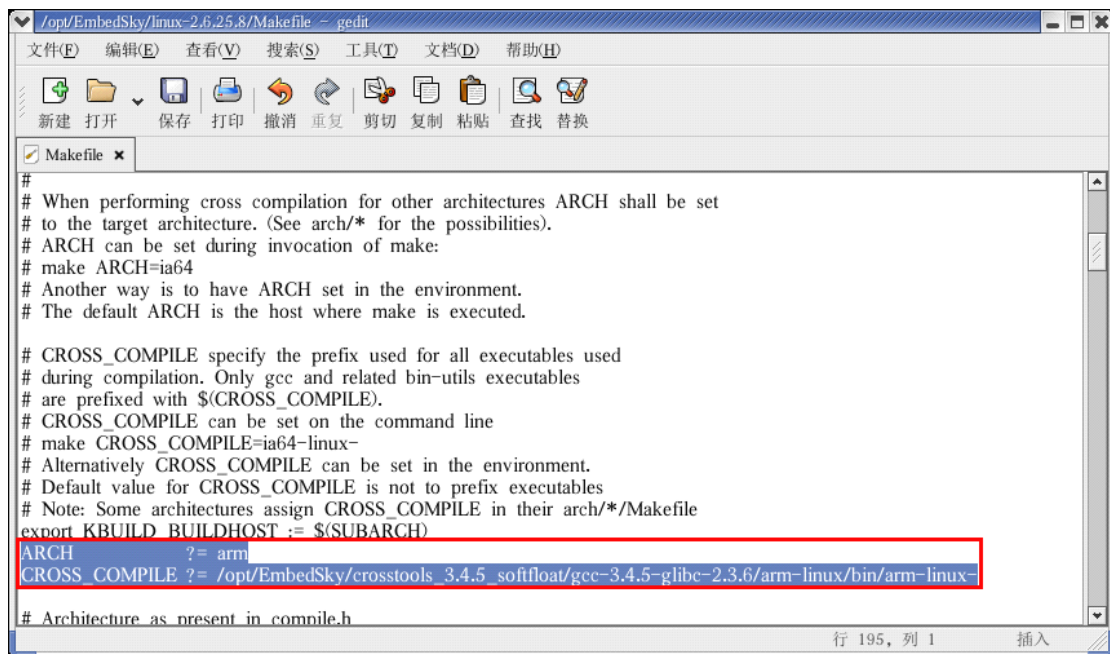
## Step 2、解压系统源码

解压内核源码到 PC 中，使用解压命令：`#tar xvfj linux-2.6.25.8.tar.bz2 -C /opt/EmbedSky/`，然后解压到 PC 的“/opt/EmbedSky/”目录下。

## Step 3、在系统中添加对 ARM 的支持

进到内核源码，修改“**Makefile**”文件，在大概 193 行“ARCH ? = (SUBARCH)”和 194 行“CROSS\_COMPILE ? =”，将其修改为“ARCH ? =arm”和“CROSS\_COMPILE ? =/opt/EmbedSky/crosstools\_3.4.5\_softfloat/gcc-3.4.5-glibc-2.3.6/arm-linux/bin/arm-linux-”，然后保存。

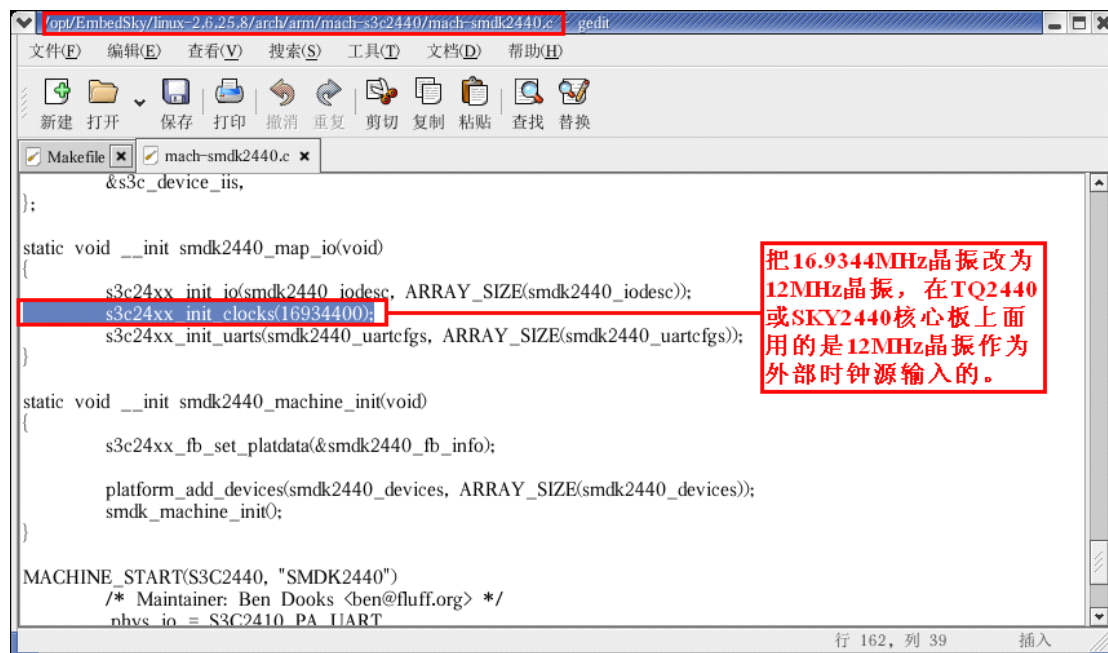
如下图所示：



## Step 4、修改平台输入时钟

修改平台的时钟频率，以满足 TQ2440 或 SKY2440 的工作频率。修改内核源码“arch/arm/mach-s3c2440/mach-s3c2440.c”文件的 162 行，把 16.9344MHz 改为 12MHz，因为 TQ2440 和 SKY2440 使用的就是 12MHz 的外部时钟源输入。

如下图所示：



然后保存刚刚设置好的参数，然后就可以开始尝试编译出镜像使其运行到开发板中。

输入：`#make menuconfig`，然后进入配置菜单界面，如下图所示：

Linux Kernel Configuration

Arrow keys navigate the menu. <Enter> selects submenus --->. Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [\*] built-in [ ] excluded <M> module < >

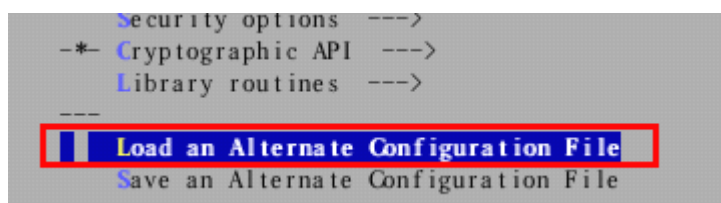
**General setup --->**

- [\*] Enable loadable module support --->
- \*- Enable the block layer --->
  - System Type --->
  - Bus support --->
  - Kernel Features --->
  - Boot options --->
  - Floating point emulation --->
  - Userspace binary formats --->
  - Power management options --->
  - Networking --->
  - Device Drivers --->
  - File systems --->
  - Kernel hacking --->
  - Security options --->
- \*- Cryptographic API --->
- Library routines --->
- 
- Load an Alternate Configuration File
- Save an Alternate Configuration File

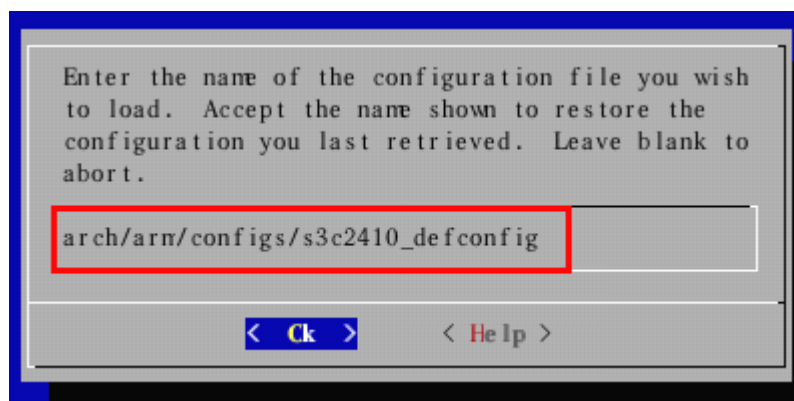
## Step 5、制作 TQ2440/SKY2440 的配置单

这里推荐的一种做法，就是先调用自带的一个默认配置单，该配置单在内核源码的“arch/arm/configs/”目录下，名为：“s3c2410\_defconfig”，改配置文件里面选择了几乎所有的和 S3C24XX 系列 CPU 相关的配置选项，我们完全可以在该配置单的基础上进行配置。当然，如果您有兴趣、时间以及精力的话，完全可以自行配置，具体配置方法，建议摸索，不要害怕出错或失败。我在这里就不讲那么多了。

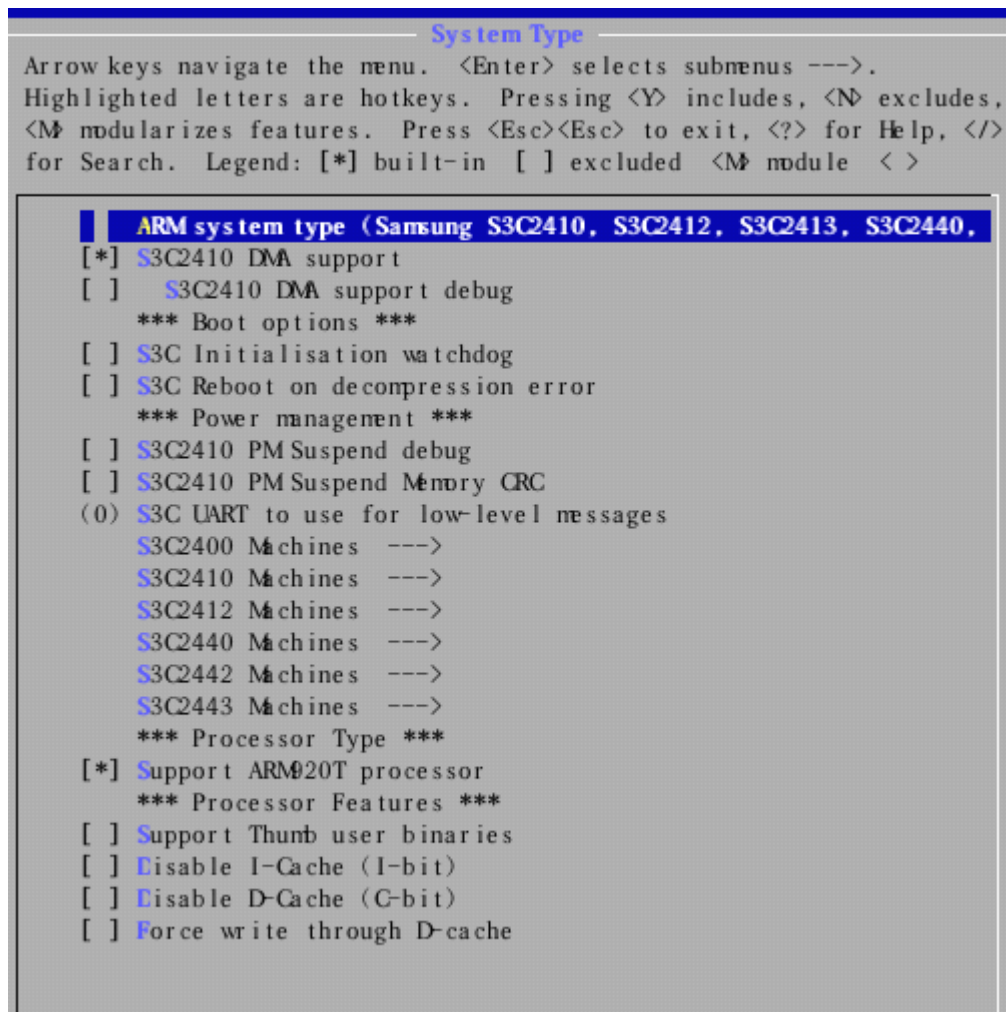
在配置菜单中选择选项：“Load an Alternate Configuration File”，然后调用刚刚说到的那个配置文件，如下图：



然后进入到如下界面，输入刚刚提到的路径和配置文件名称回车即可：



然后返回到配置界面，进入到“System Type”选项下的配置单：



然后配置各个平台如下所示:

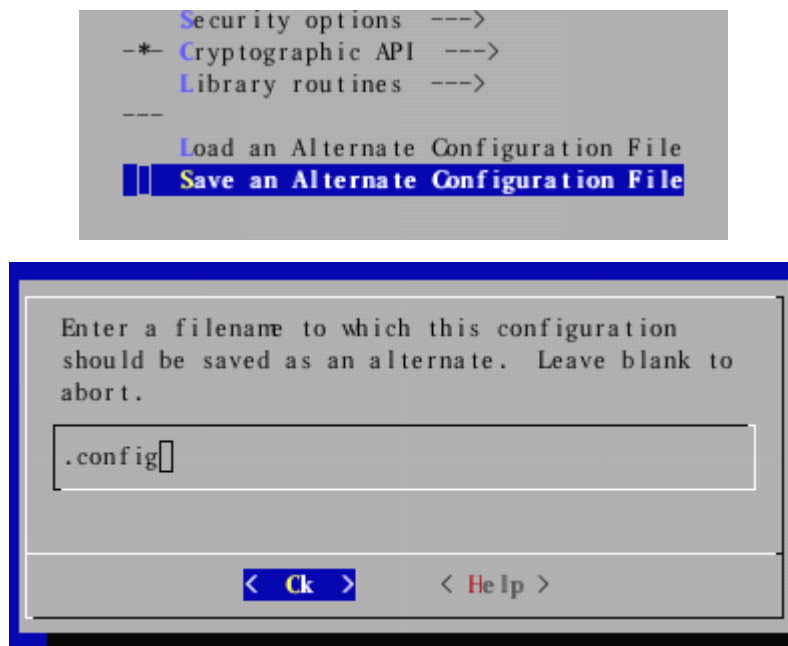
```
System Type --->
  S3C2410 Machines --->
    [*] SMDK2410/A9M2410
    [ ] IPAQ H1940
    [ ] Acer N30
    [ ] Simtec Electronics BAST (EB2410ITX)
    [ ] NexVision OTOM Board
    [ ] AML M5900 Series
    [ ] Thorcom VR1000
    [ ] QT2410
  S3C2412 Machines --->
    [ ] SMDK2413
    [ ] SMDK2412
    [ ] VMSTMS
  S3C2440 Machines --->
    [ ] Simtec Electronics ANUBIS
    [ ] Simtec IM2440D20 (OSIRIS) module
    [ ] HP iPAQ rx3715
```

```

[*] SMDK2440
[ ] NexVision NEXCODER 2440 Light Board
[*] SMDK2440 with S3C2440 CPU module
S3C2442 Machines --->
[ ] SMDM2440 with S3C2442 CPU module
S3C2443 Machines --->
[ ] SMDK2443

```

配置完毕这个地方后，退回到最初的配置单。然后选择选项：“Save an Alternate Configuration File”，将其保存为：“**.config**”文件，因为编译系统时会调用该文件。如下面的图所示：



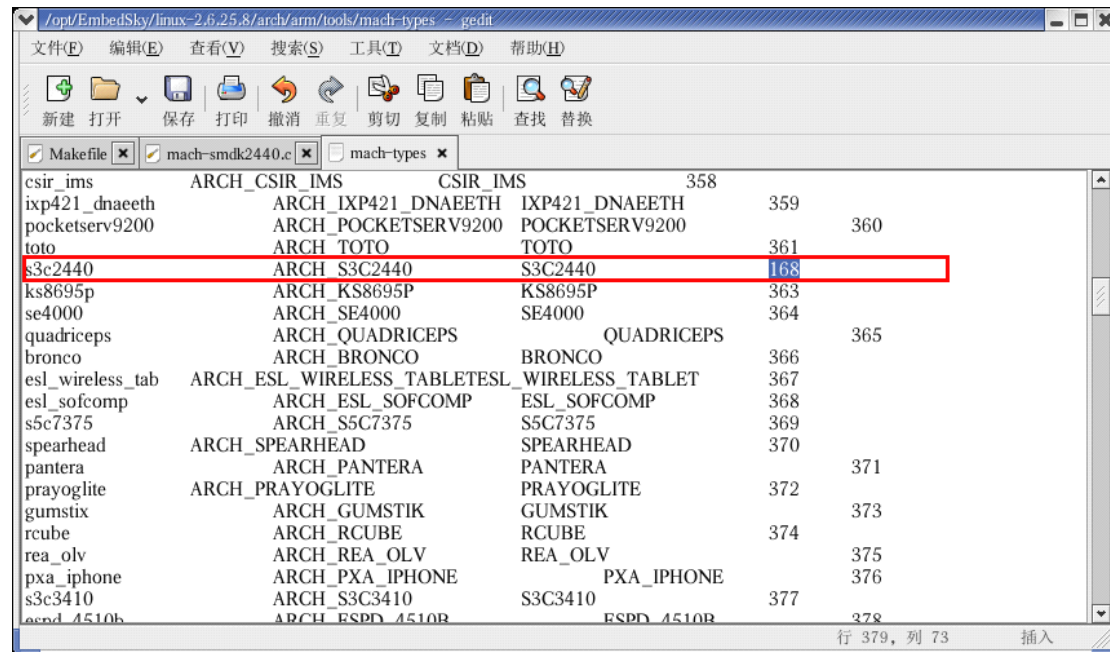
保存为“**.config**”文件后，退出配置单。

**注意：**每次执行了**#make distclean**命令，或想直接使用制作好的配置文件，建议在终端执行：**#cp config\_EmbedSky .config**，然后再编译或进入配置单进行修改。



## Step 6、修改机器码

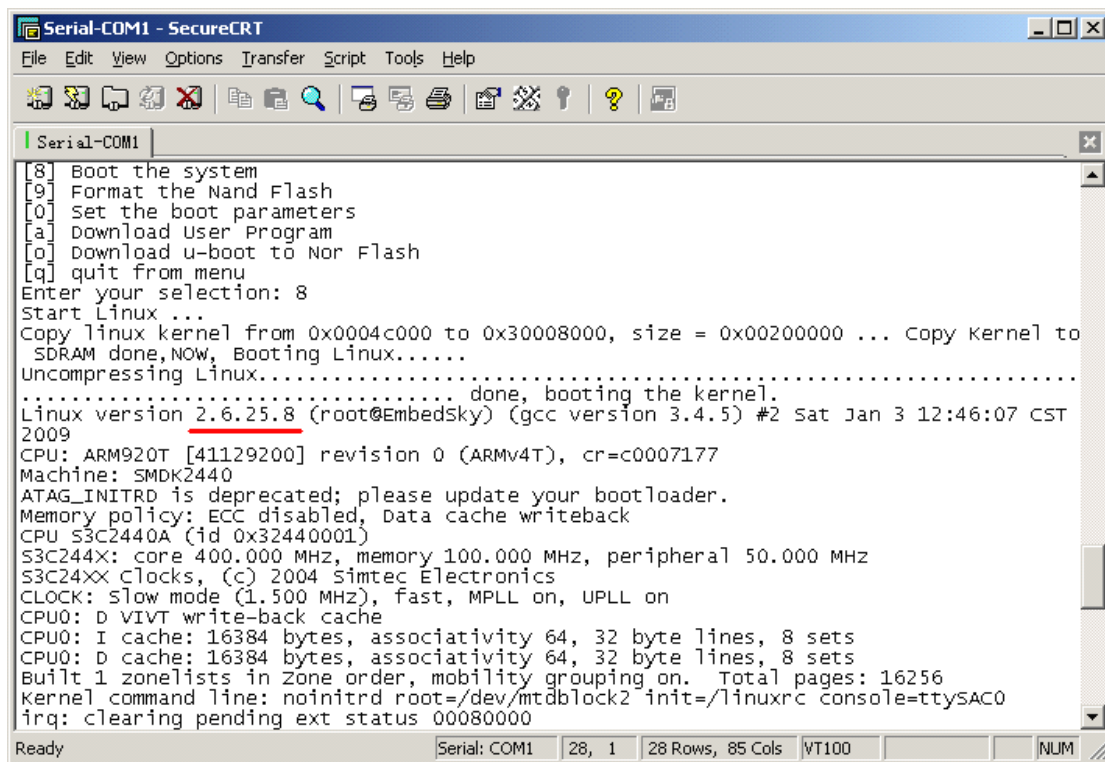
在 TQ2440 或 SKY2440 使用的 uboot 的机器码是 168，这里需要修改机器码，否则会出现不能启动的情况。机器码保存在内核源码的“arch/arm/tools/mach-types”文件中，在大概 379 行，把原来的 362 改为 168 保存即可。



## Step 7、编译镜像

### 7.1 镜像

然后输入：`#make zImage`，就可以进行编译了，编译完毕后，会在内核源码的“`arch/arm/boot/`”目录下面生成名为“`zImage`”的镜像，然后将其烧写到开发板中，启动情况如下：



```
Serial-COM1 - SecureCRT
File Edit View Options Transfer Script Tools Help

Serial-COM1

[8] Boot the system
[9] Format the Nand Flash
[0] Set the boot parameters
[a] Download User Program
[o] Download u-boot to Nor Flash
[q] quit from menu
Enter your selection: 8
Start Linux ...
Copy linux kernel from 0x0004c000 to 0x30008000, size = 0x00200000 ... Copy Kernel to
SDRAM done,NOW, Booting Linux.....
Uncompressing Linux..... done, booting the kernel.
Linux version 2.6.25.8 (root@Embedsky) (gcc version 3.4.5) #2 Sat Jan 3 12:46:07 CST
2009
CPU: ARM920T [41129200] revision 0 (ARMv4T), cr=c0007177
Machine: SMDK2440
ATAG_INITRD is deprecated; please update your bootloader.
Memory policy: ECC disabled, Data cache writeback
CPU S3C2440A (id 0x32440001)
S3C244X: core 400.000 MHz, memory 100.000 MHz, peripheral 50.000 MHz
S3C24XX Clocks, (c) 2004 Simtec Electronics
CLOCK: slow mode (1.500 MHz), fast, MPPLL on, UPLL on
CPU0: D VIVT write-back cache
CPU0: I cache: 16384 bytes, associativity 64, 32 byte lines, 8 sets
CPU0: D cache: 16384 bytes, associativity 64, 32 byte lines, 8 sets
Built 1 zonelists in Zone order, mobility grouping on. Total pages: 16256
Kernel command line: noinitrd root=/dev/mtdblock2 init=/linuxrc console=ttySAC0
irq: clearing pending ext status 00080000

Ready Serial: COM1 28, 1 28 Rows, 85 Cols VT100 NUM
```

### 7.2 把镜像存放到指定位置

为了方便，我在这里还修改了内核源码的“`arch/arm/boot/`”目录下面“`Makefile`”文件，在 58 行添加了如下内容（红色部分所示），实现了把生成的 `zImage` 文件复制到内核源码根目录下面，即 `linux-2.6.25.8` 目录下：

```
$(obj)/zImage: $(obj)/compressed/vmlinux FORCE
```

```
$(call if_changed,objcopy)
```

```
@cp -f arch/arm/boot/zImage zImage
```

```
@echo ' Kernel: $@ is ready'
```

同时修改内核源码根目录下面的“`Makefile`”文件，在 1156 行，添加如下内容（红色部分所示），在 `#make distclean` 清除产生的文件时，把内核源码根目录下的 `zImage` 也清除的目的：

```
distclean: mrproper
```

```
@find $(srctree) $(RCS_FIND_IGNORE) \
```

```
\( -name '*.orig' -o -name '*.rej' -o -name '*~' \
```

```
-o -name '*.bak' -o -name '#*#' -o -name '!.orig' \
```

```
-o -name '*.rej' -o -size 0 \
```

```
-o -name '*%' -o -name '*.cmd' -o -name 'core' \) \
```

```
-type f -print | xargs rm -f rm zImage
```

**注意：**这里仅仅是能够引导了，因为还未做 Nand Flash 方面的移植，所以，这里只完成了第一步工作。

# 让系统“跑”起来

## Step 8、Nand Flash 驱动移植

进行 Nand Flash 的移植，其实在 linux 里面已经做好了 Nand Flash 的驱动，我们只需要进行修改就可以使用了。

### 8.1 完善源码

修改内核源码“arch/arm/plat-s3c24xx/common-smdk.c”文件，在 109 行左右，有一个结构体名为：smdk\_default\_nand\_part[]，将其修改为如下列表所示：

```
static struct mtd_partition smdk_default_nand_part[] = {  
    [0] = {  
        .name      = "TQ2440_uboot",  
        .size = 0x00040000,  
        .offset    = 0x00000000,  
    },  
    [1] = {  
        .name      = "TQ2440_kernel",  
        .offset    = 0x0004C000,  
        .size = 0x00200000,  
    },  
    [2] = {  
        .name      = "TQ2440_yaffs2",  
        .offset    = 0x0024C000,  
        .size = 0x03DB0000,  
    },  
};
```

然后修改 Nand Flash 的读写匹配时间，修改 common-smdk.c 文件的刚刚修改后的大概 140 行左右的 smdk\_nand\_info 结构体，修改内容如下：（红色部分所示）

```
static struct s3c2410_platform_nand smdk_nand_info = {  
    .tacs      = 10,  
    .twrph0    = 25,  
    .twrph1    = 10,  
    .nr_sets   = ARRAY_SIZE(smdk_nand_sets),  
    .sets      = smdk_nand_sets,  
};
```

## 8.2 添加对应的驱动配置

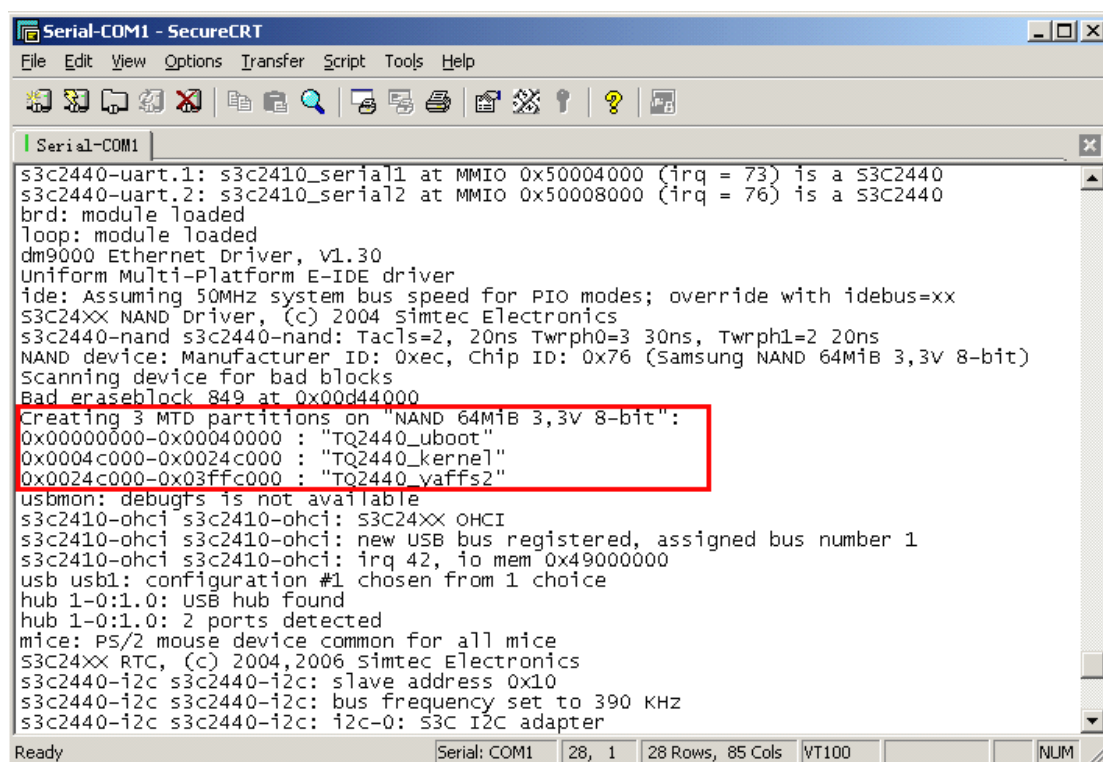
然后需要在刚刚完成的配置单中添加上对 Nand Flash 支持的配置选项，输入：`#make menuconfig`，进入配置单选项，然后配置如下所示：

```
Device Drivers --->
  <*> Memory Technology Device (MTD) support --->
    [*] MTD partitioning support
    <> RedBoot partition table parsing
    [ ] Command line partition table parsing
    <*> NAND Device Support --->
      <*> NAND Flash support for S3C2410/S3C2440 SoC
```

配置完毕这些之后，保存配置单。

## 8.3 驱动的使用情况

然后编译出镜像，然后烧写到开发板，启动开发板，你可以在打印信息中看到刚刚做好的 Nand Flash 的分区信息：



```
Serial-COM1 - SecureCRT
File Edit View Options Transfer Script Tools Help

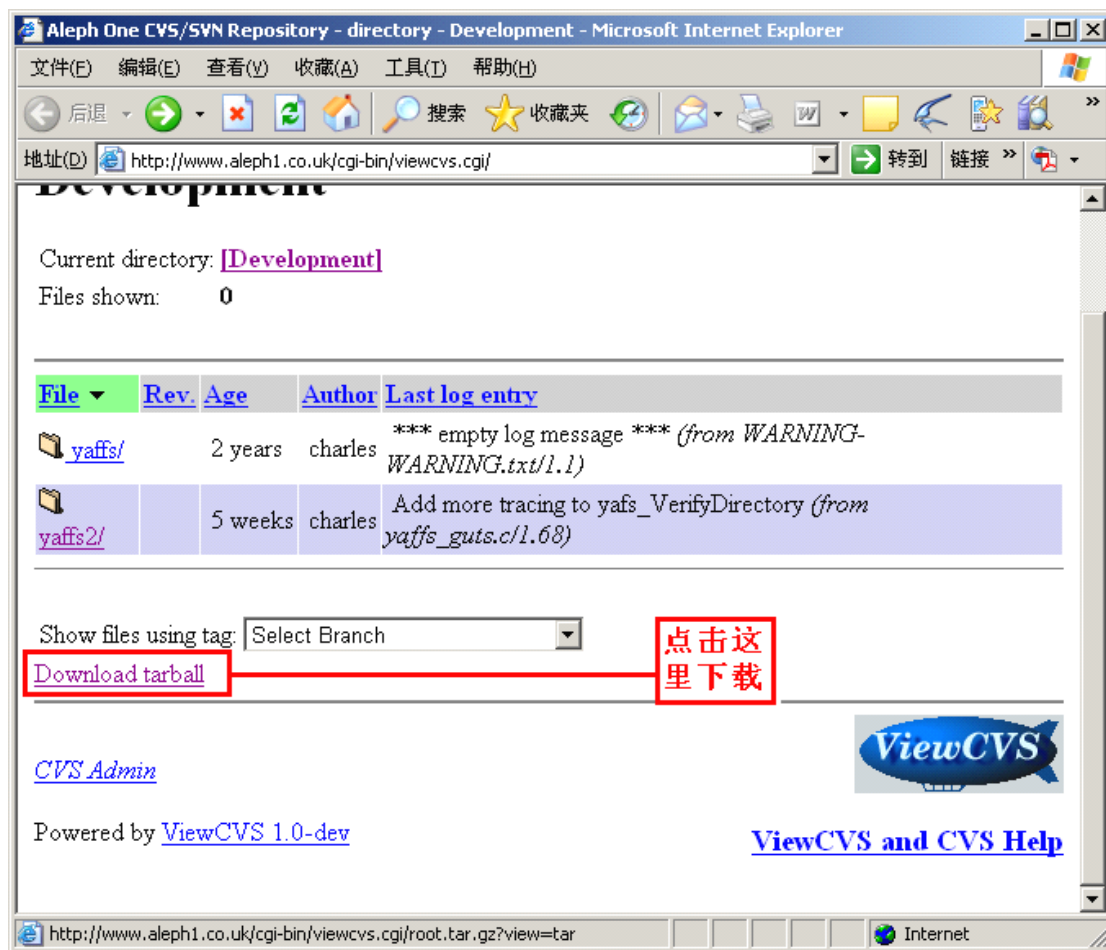
Serial-COM1
s3c2440-uart.1: s3c2410_serial1 at MMIO 0x50004000 (irq = 73) is a S3C2440
s3c2440-uart.2: s3c2410_serial2 at MMIO 0x50008000 (irq = 76) is a S3C2440
brd: module loaded
loop: module loaded
dm9000 Ethernet Driver, v1.30
Uniform Multi-Platform E-IDE driver
ide: Assuming 50MHz system bus speed for PIO modes; override with idebus=xx
S3C24XX NAND Driver, (c) 2004 Simtec Electronics
s3c2440-nand s3c2440-nand: Tacls=2, 20ns Twrph0=3 30ns, Twrph1=2 20ns
NAND device: Manufacturer ID: 0xec, Chip ID: 0x76 (Samsung NAND 64MiB 3,3v 8-bit)
Scanning device for bad blocks
Bad eraseblock 849 at 0x00d44000
Creating 3 MTD partitions on "NAND 64MiB 3,3v 8-bit":
0x00000000-0x00040000 : "TQ2440_uboot"
0x00040000-0x0024c000 : "TQ2440_kernel"
0x0024c000-0x03ffc000 : "TQ2440_yaffs2"
usbmon: debugfs is not available
s3c2410-ohci s3c2410-ohci: S3C24XX OHCI
s3c2410-ohci s3c2410-ohci: new USB bus registered, assigned bus number 1
s3c2410-ohci s3c2410-ohci: irq 42, io mem 0x49000000
usb usb1: configuration #1 chosen from 1 choice
hub 1-0:1.0: USB hub found
hub 1-0:1.0: 2 ports detected
mice: PS/2 mouse device common for all mice
S3C24XX RTC, (c) 2004,2006 Simtec Electronics
s3c2440-i2c s3c2440-i2c: slave address 0x10
s3c2440-i2c s3c2440-i2c: bus frequency set to 390 KHz
s3c2440-i2c s3c2440-i2c: i2c-0: S3C I2C adapter

Ready Serial: COM1 28, 1 28 Rows, 85 Cols VT100 NUM
```

## Step 9、移植 yaffs 文件系统

### 9.1 获取 yaffs 源码

现在移植 yaffs 文件系统是很容易的事情，首先到 <http://www.aleph1.co.uk/cgi-bin/viewcvs.cgi/> 网站下载一个 yaffs 文件系统的补丁，打开网页后，点击“Download tarball”下载补丁 root.tar.gz。

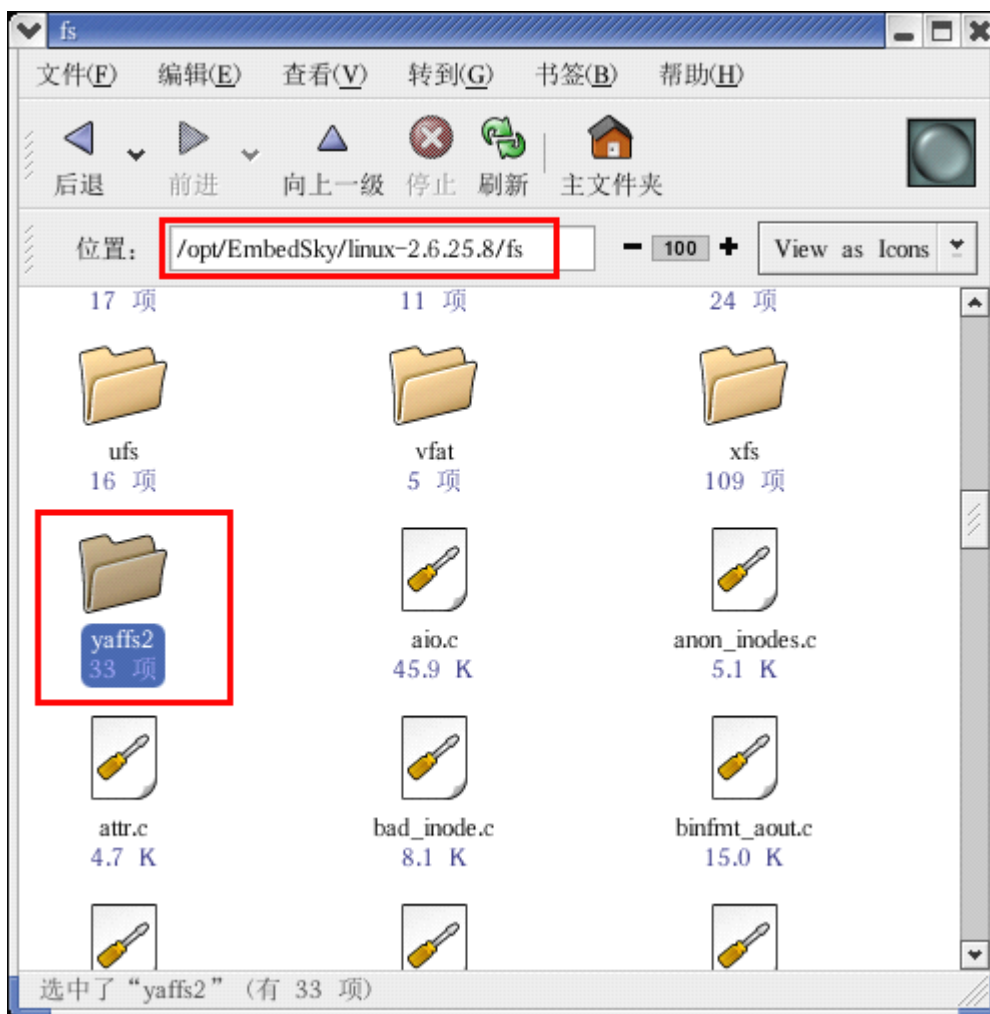


### 9.2 在内核中添加对 yaffs 的支持

得到补丁包后，解压补丁，然后打上 yaffs2 的补丁，方法如下：

```
#tar xvfz root.tar.gz (解压得到补丁目录 Development)
#cd Development/yaffs2/ (进到 yaffs2 的补丁目录下)
#./patch-ker.sh c /opt/EmbedSky/linux-2.6.25.8/ (执行补丁脚本，打补丁到内核中)
```

此时到内核源码的“fs/”目录下面您将看到新添加了一个名为“yaffs2”的目录，同时 fs/目录下面的 Makefile 文件和 Kconfig 文件也添加了 yaffs2 的配置和编译条件。



### 9.3 在配置单中添加对 **yaffs** 的支持

然后输入: `#make menuconfig`, 进入配置单, 然后配置如下所示:

```
File systems --->
  <> Second extended fs support
  <> Ext3 journalling file system support
  <*> Kernel automounter support
  <*> Kernel automounter version 4 support (also supports v3)
  <*> Filesystem in Userspace support
  CD-ROM/DVD Filesystems --->
    <*> ISO 9660 CDROM file system support
    [ ] Microsoft Joliet CDROM extensions
    [ ] Transparent decompression extension
  <> UDF file system support
  DOS/FAT/NT Filesystems --->
    <*> MSDOS fs support
    <*> VFAT (Windows-95) fs support
    (437) Default codepage for FAT
```

```

(iso8859-1) Default iocharset for FAT
< > NTFS file system support
Pseudo filesystems --->
[*] Virtual memory file system support (former shm fs)
[*] Tmpfs POSIX Access Control Lists
<*> Userspace-driven configuration filesystem
Miscellaneous filesystems --->
<*> YAFFS2 file system support
-*- 512 byte / page devices
[ ] Use older-style on-NAND data format with pageStatus byte
[*] Lets Yaffs do its own ECC
[ ] Use the same ecc byte order as Steven Hill's nand_ecc
-*- 2048 byte (or larger) / page devices
[ ] Autoselect yaffs2 format
[*] Disable lazy loading
[*] Turn off wide tnodes
[*] Force chunk erase check
[ ] Cache short names in RAM
-*- Native language support --->
--- Native language support
(iso8859-1) Default NLS Option
<*> Codepage 437 (United States, Canada)
<*> Simplified Chinese charset (CP936, GB2312)
<*> NLS ISO 8859-1 (Latin 1; Western European Languages)
<*> NLS UTF-8

```

配置好之后，保存配置单为 “.config” 文件后，编译出镜像。

到这一步就算是把 yaffs 文件系统移植完毕了，下面我们就开始制作 yaffs 文件系统。

**制作文件系统，首先需要用 busybox 编译出文件系统所需要的应用程序，然后我们再做文件系统。**



## Step 10、编译 BusyBox

### 10.1 获取 BusyBox 源码

首先我们下载一个 busybox 的源码，到网页 <http://www.busybox.net/downloads/>，就可以下载了，这里我下载了 busybox-1.13.0.tar.bz2 这个压缩包。

### 10.2 修改并配置 BusyBox

解压 busybox，使用命令 `#tar xvfj busybox-1.13.0.tar.bz2 -C /opt/EmbedSky/`。

然后进到源码中，修改 Makefile 文件，把 164 行修改为：

```
CROSS_COMPILE?=/opt/EmbedSky/crosstools_3.4.5_softfloat/gcc-3.4.5-glibc-2.3.6/arm-linux/bin/arm-linux-
```

把 189 行修改为：

```
ARCH ?=arm
```

然后输入：`#make menuconfig`，进入配置单：

```
Busybox Configuration
Arrow keys navigate the menu. <Enter> selects submenus --->.
Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes,
<M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </>
for Search. Legend: [*] built-in [ ] excluded <M> module <>

|| Busybox Settings --->
--- Applets
  Archival Utilities --->
  Coreutils --->
  Console Utilities --->
  Debian Utilities --->
  Editors --->
  Finding Utilities --->
  Init Utilities --->
  Login/Password Management Utilities --->
  Linux Ext2 FS Progs --->
  Linux Module Utilities --->
  Linux System Utilities --->
  Miscellaneous Utilities --->
  Networking Utilities --->
  Print Utilities --->
  Mail Utilities --->
  Process Utilities --->
  Runit Utilities --->
  Shells --->
  System Logging Utilities --->
---
  Load an Alternate Configuration File
  Save Configuration to an Alternate File
```

然后配置如下：（下面只列出需要注意的地方；其它未列出的地方，可以采用默认，可以自己添加。）

## Busybox Settings --->

### General Configuration --->

#### Buffer allocation policy (Allocate with Malloc) --->

- [\*] Show verbose applet usage messages
- [\*] Store applet usage messages in compressed form
- [\*] Support --install [-s] to install applet links at runtime
- [\*] Enable locale support (system needs locale for this to work)
- [\*] Support for --long-options
- [\*] Use the devpts filesystem for Unix98 PTYs
- [\*] Support writing pidfiles
- [\*] Runtime SUID/SGID configuration via /etc/busybox.conf
- [\*] Suppress warning message if /etc/busybox.conf is not readable
- (/proc/self/exe) Path to BusyBox executable

### Build Options --->

- [\*] Build BusyBox as a static binary (no shared libs)
- [\*] Build with Large File Support (for accessing files > 2 GB)

### Installation Options --->

- [ ] Don't use /usr

#### Applets links (as soft-links) --->

(./\_install) BusyBox installation prefix

### Busybox Library Tuning --->

- (6) Minimum password length
- (2) MD5: Trade Bytes for Speed
- [\*] Faster /proc scanning code (+100 bytes)
- [\*] Command line editing
- (1024) Maximum length of input
- [\*] vi-style line editing commands
- (15) History size
- [\*] History saving
- [\*] Tab completion
- [\*] Fancy shell prompts
- (4) Copy buffer size, in kilobytes
- [\*] Use ioctl names rather than hex values in error messages
- [\*] Support infiniband HW

## Linux Module Utilities --->

(/lib/modules) Default directory containing modules

(modules.dep) Default name of modules.dep

[\*] insmod

[\*] rmmod

[\*] lsmod

[\*] modprobe

--- Options common to multiple modutils

[ ] Support version 2.2/2.4 Linux kernels

[\*] Support tainted module checking with new kernels

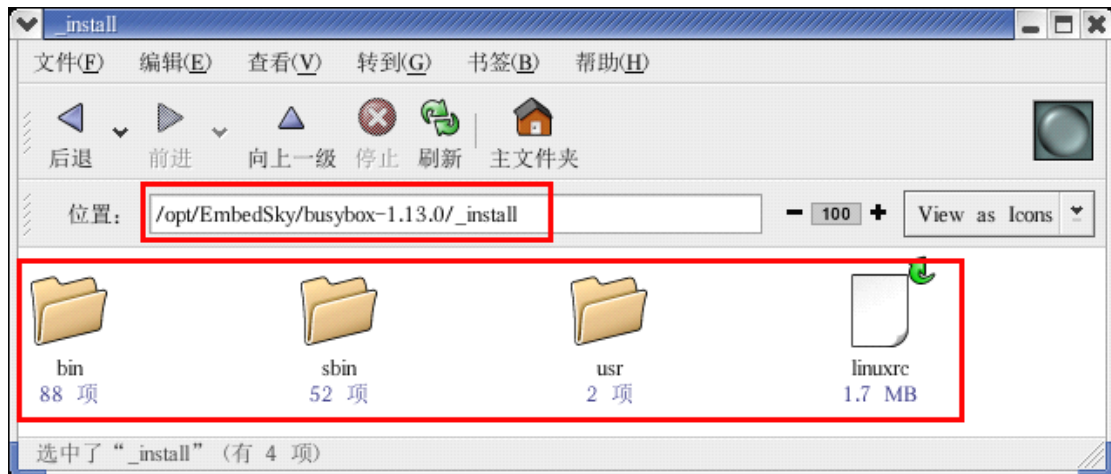
[\*] Support for module.aliases file

[\*] Support for module.symbols fileLinux System Utilities --->

然后退回到根配置单，选择“Save Configuration to an Alternate File”选项，保存刚刚的配置为 `config_EmbedSky`。

## 10.3 编译并安装 BusyBox

然后退出配置单，然后编译出 busybox 即可，使用命令：`#make; make install`，编译结束后会在 busybox-1.13.0 目录下面生成一个名为“`_install`”的目录，该目录下面情况如下图所示：



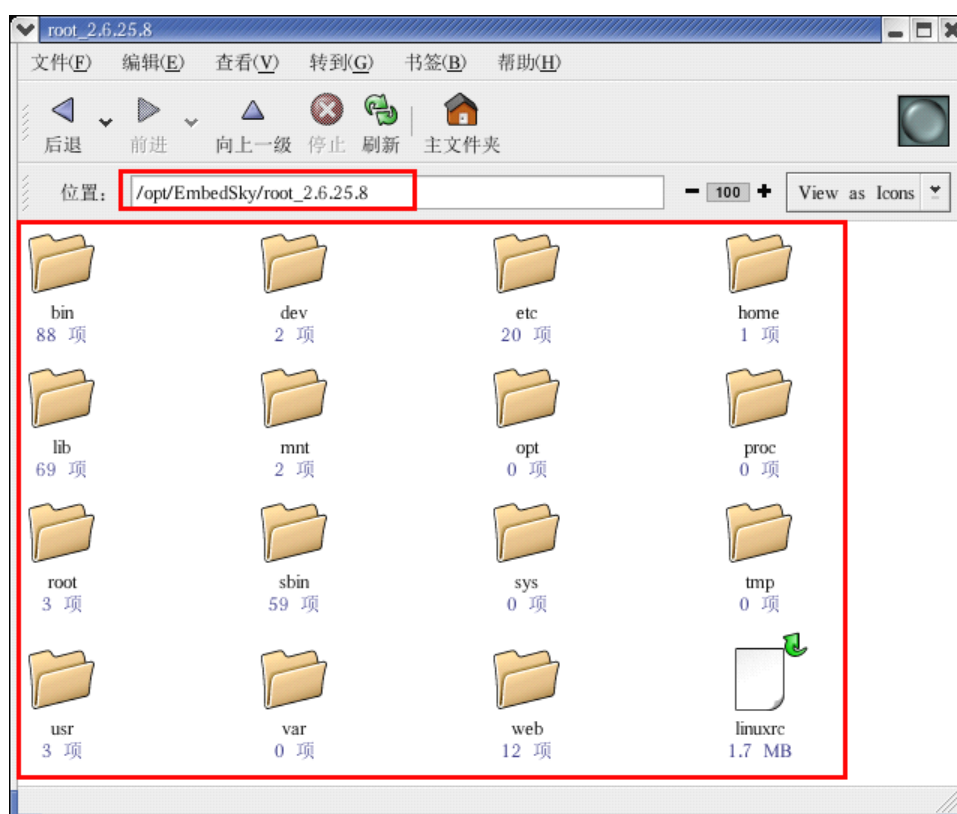
完成 busybox 的编译之后，我们就可以进行文件系统的构建了。

## Step 11、构建文件系统

### 11.1 构建框架

在“/opt/EmbedSky/”目录下面建立一个“root\_2.6.25.8”的目录，复制刚刚编译出来的 busybox 的“bin”目录、“sbin”目录、“usr”目录和“linuxrc”到“root\_2.6.25.8”目录下，然后新建“dev”、“etc”、“home”、“lib”、“mnt”、“opt”、“proc”、“root”、“sys”、“tmp”、“var”和“web”目录，同时在原有的“usr”目录下面新建一个“lib”目录。

如下图所示：



### 11.2 添加内容

#### “dev”目录

下面的内容在内核挂载完毕文件系统后，系统会使用 mdev 自动建立。

不过 mdev 是调用 init 进程来启动的，在使用 mdev 构建“dev”目录前，init 进程需要用到“/dev/console”和“/dev/null”这两个设备文件，所以，我们需要在制作文件系统时静态创建这两个设备文件，方法如下：

```
#cd /opt/EmbedSky/root_2.6.25.8/dev
```

```
#mknod console c 5 1
```

```
#mknod null c 1 3
```

如果不创建这两个设备文件，在文件系统启动时会出现类似这样的错误信息：“Warning: unable to open an initial console.”。这句话来源于内核源码的“init/main.c”文件的 778 行的“init\_post（）”函数，也就是没有打开控制台。

## “etc” 目录

用来存放系统的配置文件。

在“etc”目录下面有如下常用的文件：（主要是下面的这几个，以后要添加文件时会说明的）

**fstab**: 指明需要挂载的文件系统；

**group**: 用户组；

**inittab**: init 进程的配置文件；

**passwd**: 密码文件；

**profile**: 用户环境配置文件；

**mdev.conf**: 因为 2.6.18 版本开始 linux 放弃使用 devfs 而采用 udev（mdev 是 udev 的简化版本），这里的 mdev.conf 文件可以是空，也可以按照一定规则来编写，这里我采用了为空；

**resolv.conf**: 存放 DNS 信息的文件，访问外网时需要 DNS 的信息。

常用的目录有：

**init.d** 目录：启动文件目录，该目录下面有个“rcS”的文件，里面存放了系统启动时配置以及自启动加载的进程等；

**sysconfig** 目录：在我们的文件系统里面，该目录下面存放了名为“HOSTNAME”的文件，该文件内容为：EmbedSky，这句话就是我们在文件系统里面看到那个“[root@EmbedSky /]”里面的 EmbedSky；

**rc.d** 目录：在我们的文件系统里面用来存放一些自启动所要调用的脚步等；

**boa** 目录：我们的文件系统特有的目录，用来存放 web 服务器的配置脚本。

下面分别列出“etc”下面各个文件的内容。

【fstab】:

# device	mount-point	type	options	dump	fsck order
proc	/proc	proc	defaults	0	0
tmpfs	/tmp	tmpfs	defaults	0	0
sysfs	/sys	sysfs	defaults	0	0
tmpfs	/dev	tmpfs	defaults	0	0
var	/dev	tmpfs	defaults	0	0

【group】:

root: \*:0:

daemon: \*:1:

bin: \*:2:

sys: \*:3:

adm: \*:4:

tty: \*:5:

disk: \*:6:

lp: \*:7:lp

mail: \*:8:

news: \*:9:

uucp: \*:10:

proxy: \*:13:

```
kmem:*.15:
dialout:*.20:
fax:*.21:
voice:*.22:
cdrom:*.24:
floppy:*.25:
tape:*.26:
sudo:*.27:
audio:*.29:
ppp:x:99:
500:x:500:sky
501:x:501:sky
```

【inittab】: 参考 busybox 源码下面的“examples/inittab”文件

```
# /etc/inittab
::sysinit:/etc/init.d/rcS
s3c2410_serial0::askfirst:/bin/sh
::ctrlaltdel:/sbin/reboot
::shutdown:/bin/umount -a -r
```

注意: 上面红色部分在 busybox 的参考文件中是 ttySAC0, 而在 s3c24xx 系列芯片的串口驱动里面我们用了 s3c2410\_serial 做为设备名 (在内核源码的“drivers/serial/s3c2410.c”文件的 949 行), 然后我们又是用的串口 0 作为控制台的, 所以这里我们使用 s3c2410\_serial0。

【passwd】:

```
root::0:0:root:/bin/sh
ftp::14:50:FTP User:/var/ftp:
bin:*.1:1:bin:/bin:
daemon:*.2:2:daemon:/sbin:
nobody:*.99:99:Nobody:/:
sky:$1$8GIZx6d9$L2ctqdXbYDzkbxNURpE4z/:502:502:Linux User,,:/home/sky:/bin/sh
```

注意: 上面红色那部分是密码, 是不可逆转的编码, 获取方法: 在文件系统做好之后, 使用 passwd 命令然后设定密码, 在打开这个文件, 就有了。

【profile】:

```
# Ash profile
# vim: syntax=sh

# No core files by default
#ulimit -S -c 0 > /dev/null 2>&1

USER=`id -un`
LOGNAME=$USER
PS1='[u@h \W]# '
PATH=$PATH

HOSTNAME=`/bin/hostname`
```

```
export USER LOGNAME PS1 PATH
```

```
【mdev.conf】:
```

```
( 为空 )
```

```
【 resolv.conf 】:
```

```
nameserver 202.96.128.166
```

```
【 init.d/rcS 】:
```

```
#!/bin/sh
```

```
PATH=/sbin:/bin:/usr/sbin:/usr/bin
```

```
runlevel=S
```

```
prevlevel=N
```

```
umask 022
```

```
export PATH runlevel prevlevel
```

```
#
```

```
# Trap CTRL-C &c only in this shell so we can interrupt subprocesses.
```

```
#
```

```
mount -a
```

```
mkdir /dev/pts
```

```
mount -t devpts devpts /dev/pts
```

```
echo /sbin/mdev > /proc/sys/kernel/hotplug
```

```
mdev -s
```

```
mkdir -p /var/lock
```

```
/bin/hostname -F /etc/sysconfig/HOSTNAME
```

注意： 在上面红色部分是关于 mdev 的使用，请参考 busybox 的“docs/mdev.txt”文件。

```
【 rc.d/init.d/httpd 】:
```

```
#!/bin/sh
```

```
base=boa
```

```
# See how we were called.
```

```
case "$1" in
```

```
start)
```

```
    /sbin/$base
```

```
    ;;
```

```
stop)
```

```
    pid=`/bin/pidof $base`
```

```
    if [ -n "$pid" ]; then
```

```
        kill -9 $pid
```

```
    fi
```

```
    ;;
```

esac

exit 0

【boa/boa.conf】:

（内容省略，该文件的来历，请见 SKY2440 或 TQ2440 开发板的使用手册的 web 服务器移植篇）

【sysconfig/HOSTNAME】:

EmbedSky

暂时就在“etc/”目录下添加这些内容。

## “home”目录

存放用户文件的目录，在这里，我们建立一个名为“sky”的目录，还记得前面的 passwd 文件吧，里面有个用户名就是 sky，对应这里的“sky”的目录。

## “lib”目录

用来存放常用的库文件，获取库文件的方法：

```
#cp -f /opt/EmbedSky/crosstools_3.4.5_softfloat/gcc-3.4.5-glibc-2.3.6/arm-linux/arm-linux/lib/*so* lib -a
```

这里拷贝了常用的库文件，如果需要特殊的库，需要再从相应的位置复制过来。

## “mnt”目录

我们一般用来挂载的 U 盘之类的外设，这里建立两个目录“udisk”和“sd”，分别用来挂载 U 盘和 SD 卡。

## “opt”目录

我们用来保存 Qt 的相关目录。

## “proc”目录

提供一些目录和虚拟文件系统。

## “root”目录

超级用户的目录。

## “sys”目录

mdev 可能会在下面建立某些文件。

## “tmp”目录

存放临时文件的目录。

## “var”目录

存放临时文件的目录。

## “web”目录

存放 web 服务器的相关文件的目录，建议直接从以前的文件系统拷贝过来。

到这里，文件系统的框架就基本搞好了，然后使用制作 yaffs 文件系统的软件，把它做出 yaffs



文件系统格式的镜像就可以使用了。使用如下命令制作：

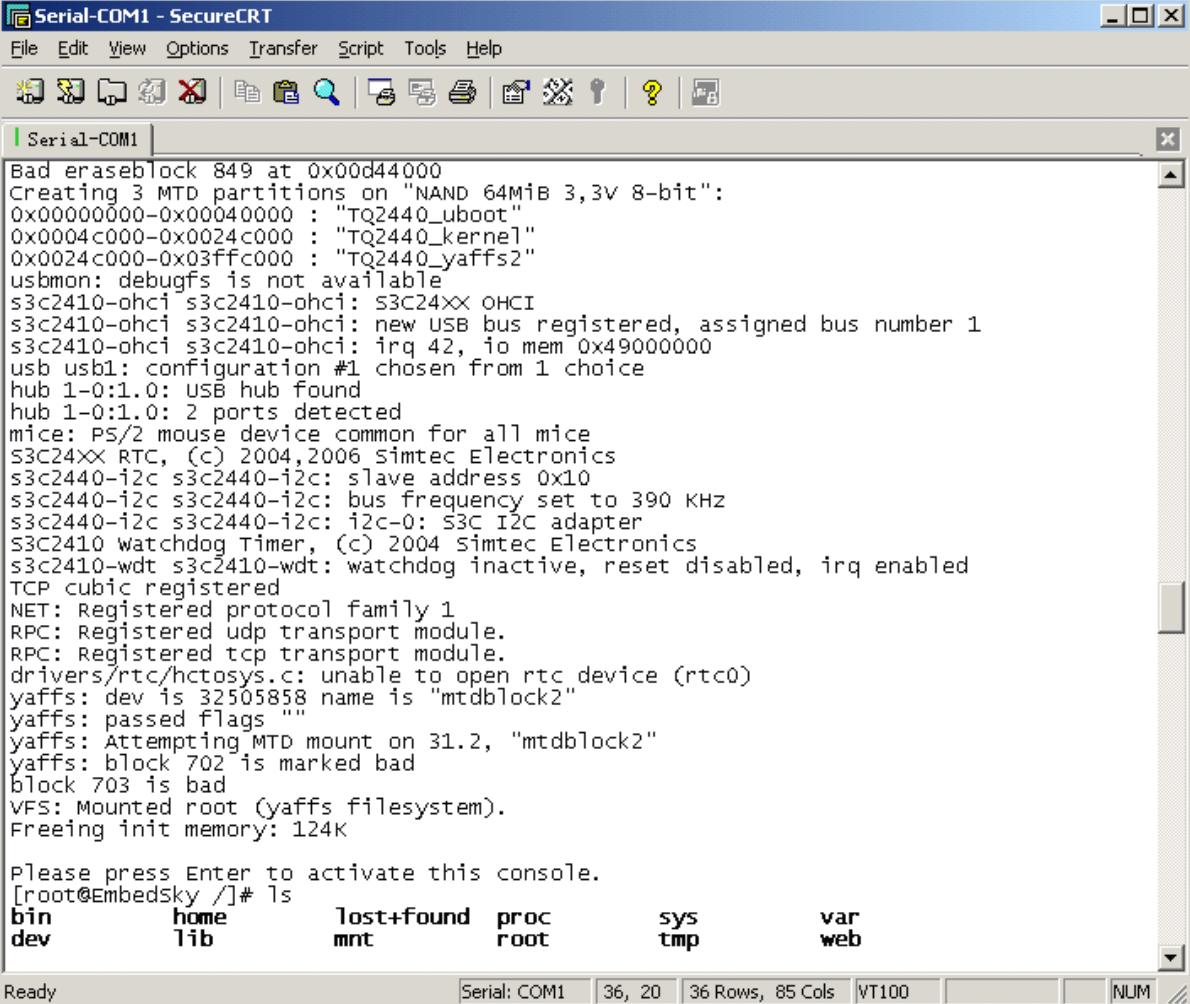
```
#cd /opt/EmbedSky
```

```
#mkyaffsimage_2 root_2.6.25.8/ root_2.6.25.8.yaffs
```

注意：上面的操作命令直接的空格键。

然后把制作好的镜像 root\_2.6.25.8.yaffs 复制出来，烧写到开发板中，就可以运行了。

下面是运行 yaffs 的截图：



```
Serial-COM1 - SecureCRT
File Edit View Options Transfer Script Tools Help

Serial-COM1
Bad eraseblock 849 at 0x00d44000
Creating 3 MTD partitions on "NAND 64MiB 3,3V 8-bit":
0x00000000-0x00040000 : "TQ2440_uboot"
0x00040000-0x0024c000 : "TQ2440_kernel"
0x0024c000-0x03ffc000 : "TQ2440_yaffs2"
usbmon: debugfs is not available
s3c2410-ohci s3c2410-ohci: S3C24XX OHCI
s3c2410-ohci s3c2410-ohci: new USB bus registered, assigned bus number 1
s3c2410-ohci s3c2410-ohci: irq 42, io mem 0x49000000
usb usb1: configuration #1 chosen from 1 choice
hub 1-0:1.0: USB hub found
hub 1-0:1.0: 2 ports detected
mouse: PS/2 mouse device common for all mice
S3C24XX RTC, (c) 2004,2006 Simtec Electronics
s3c2440-i2c s3c2440-i2c: slave address 0x10
s3c2440-i2c s3c2440-i2c: bus frequency set to 390 KHz
s3c2440-i2c s3c2440-i2c: i2c-0: S3C I2C adapter
S3C2410 Watchdog Timer, (c) 2004 Simtec Electronics
s3c2410-wdt s3c2410-wdt: watchdog inactive, reset disabled, irq enabled
TCP cubic registered
NET: Registered protocol family 1
RPC: Registered udp transport module.
RPC: Registered tcp transport module.
drivers/rtc/hctosys.c: unable to open rtc device (rtc0)
yaffs: dev is 32505858 name is "mtdblock2"
yaffs: passed flags ""
yaffs: Attempting MTD mount on 31.2, "mtdblock2"
yaffs: block 702 is marked bad
block 703 is bad
VFS: Mounted root (yaffs filesystem).
Freeing init memory: 124K

Please press Enter to activate this console.
[root@EmbedSky /]# ls
bin      home    lost+found  proc      sys      var
dev      lib     mnt        root      tmp      web
```

## Step 12、完善串口驱动

### 12.1 修改源码

在内核里面只支持 2 个串口，也就是芯片的 UART0 和 UART1，而 UART2 的驱动是针对红外接口的，而不是串口驱动，这里我们将其修改为串口驱动。

修改内核源码 “arch/arm/mach-s3c2440/mach-smdk2440.c” 文件的 99 行，将其改为：

```
.ulcon = 0x03,
```

修改内核源码 “drivers/serial/s3c2410.c” 文件，在 518 行开始添加如下内容（红色部分所示）：

```
static int s3c24xx_serial_startup(struct uart_port *port)
{
    struct s3c24xx_uart_port *ourport = to_ourport(port);
    int ret;

    dbg("s3c24xx_serial_startup: port=%p (%08lx,%p)\n",
        port->mapbase, port->membase);

    if(port->line == 2)
    {
        s3c2410_gpio_cfgpin(S3C2410_GPH6, S3C2410_GPH6_TXD2);
        s3c2410_gpio_pullup(S3C2410_GPH6, 1);

        s3c2410_gpio_cfgpin(S3C2410_GPH7, S3C2410_GPH7_RXD2);
        s3c2410_gpio_pullup(S3C2410_GPH7, 1);
    }

    rx_enabled(port) = 1;

    ret = request_irq(RX_IRQ(port),
        s3c24xx_serial_rx_chars, 0,
        s3c24xx_serial_portname(port), ourport);

    if (ret != 0) {
        printk(KERN_ERR "cannot get irq %d\n", RX_IRQ(port));
        return ret;
    }
}
```

然后重新编译出镜像，烧写到开发板中，三个串口都能使用了。

### 12.2 串口测试

下面我们采用最简单的串口测试方法，在 busybox 中有个 getty 的命令，它的功能是完成控制

台的转换，而在嵌入式系统里面一般情况下串口就是控制台，所以，我们用 `getty` 的命令进行串口的测试。

## 串口 1 的测试：

在默认的系统里我们把串口 1 也就是 UART0 作为控制台了，它的测试就直接输入任意数据，然后串口工具有对应信息打印出来，则表示该串口是能正常工作的。

## 串口 2 的测试：

在串口工具输入：`$getty /dev/s3c2410_serial1 115200`（就是要把控制台交给 UART1，波特率设置为 115200），然后把 PC 的串口接到开发板的串口扩展板的 COM1 上，然后输入：`root`，然后回车，就可以看到控制台转移到串口 2 了，然后随意输入一些数据，您可以看到控制台会相应的打印出对应的数据。

## 串口 3 的测试：

在串口工具输入：`$getty /dev/s3c2410_serial2 115200`（就是要把控制台交给 UART2，波特率设置为 115200），然后相应的可以进入到串口 3 的控制台。

下面的截图是操作时的截图：

```
Serial-COM1 - SecureCRT
File Edit View Options Transfer Script Tools Help

Serial-COM1
drivers/rtc/hctosys.c: unable to open rtc device (rtc0)
yaffs: dev is 32505858 name is "mtdblock2"
yaffs: passed flags ""
yaffs: Attempting MTD mount on 31.2, "mtdblock2"
yaffs: block 3525 is marked bad
block 3526 is bad
VFS: Mounted root (yaffs filesystem).
Freeing init memory: 124k
eth0: link down

Please press Enter to activate this console.
[root@Embedsky /]# getty /dev/s3c2410_serial1 115200
root
[root@Embedsky /]# abcdefghijklmnopqrstuvwxyz1234567890
-sh: abcdefghijklmnopqrstuvwxyz1234567890: not found
[root@Embedsky /]# getty /dev/s3c2410_serial2 115200
root
[root@Embedsky /]# abcdefghijklmnopqrstuvwxyz1234567890
-sh: abcdefghijklmnopqrstuvwxyz1234567890: not found
[root@Embedsky /]# getty /dev/s3c2410_serial0 115200
root
login[796]: root login on 's3c2410_serial0'
[root@Embedsky /]# abcdefghijklmnopqrstuvwxyz1234567890
-sh: abcdefghijklmnopqrstuvwxyz1234567890: not found
[root@Embedsky /]#
```

分别把控制台从串口1交给串口2再交给串口3然后又回到串口1。root是采用root用户登录控制台

## Step 13、网卡驱动移植

### 13.1 驱动源码获取

在内核里面网卡驱动是相当完善的，这里需要注意一件事情，从 2.6.25 开始的内核的“drivers/net/dm9000.c”这个文件对应的 DM9000 的驱动（版本为 1.3 版）并不适合 DM9000E 这颗芯片，而在 SKY2440 和 TQ2440 开发板上用的是 DM9000E 这颗芯片，所以我们需要更换“dm9000.c”这个驱动程序，我们只要找到 1.2 版的 DM9000 的驱动就可以支持 DM9000E 这颗芯片了，下载一个 linux-2.6.24.tar.bz2 的源码包，然后解压，提取“drivers/net/dm9000.c”这个文件，打开看一下，您可以发现这个驱动文件是 1.2 版本的，我们用它去替换掉 2.6.25.8 里面的对应源码即可开始进行我们的移植操作。

### 13.2 修改驱动源码

我们只需要进行简单的修改就可以成功的驱动上 SKY2440 或 TQ2440 上面的 DM9000 的网卡芯片。

修改内核源码的“arch/arm/plat-s3c24xx/common-smdk.c”文件：

在 46 行添加如下内容（红色部分所示）：

```
#include <asm/plat-s3c24xx/common-smdk.h>
#include <asm/plat-s3c24xx/devs.h>
#include <asm/plat-s3c24xx/pm.h>
#if defined(CONFIG_DM9000) || defined(CONFIG_DM9000_MODULE)
#include <linux/dm9000.h>
#endif
```

```
/* LED devices */
```

```
static struct s3c24xx_led_platdata smdk_pdata_led4 = {
```

然后在 151 行左右，添加如下内容（红色部分所示）：

```
static struct s3c2410_platform_nand smdk_nand_info = {
    .tacs      = 10,
    .twrph0    = 25,
    .twrph1    = 10,
    .nr_sets   = ARRAY_SIZE(smdk_nand_sets),
    .sets      = smdk_nand_sets,
};
```

```
#if defined(CONFIG_DM9000) || defined(CONFIG_DM9000_MODULE)
```

```
/* DM9000 */
```

```
static struct resource s3c_dm9k_resource[] = {
```

```
[0] = {
```

```

        .start = S3C2410_CS4,      /* ADDR2=0，发送地址时使用这个地址 */
        .end   = S3C2410_CS4 + 3,
        .flags = IORESOURCE_MEM,
    },
    [1] = {
        .start = S3C2410_CS4 + 4,  /* ADDR2=1，传输数据时使用这个地址 */
        .end   = S3C2410_CS4 + 4 + 3,
        .flags = IORESOURCE_MEM,
    },
    [2] = {
        .start = IRQ_EINT7,        /* 中断号 */
        .end   = IRQ_EINT7,
        .flags = IORESOURCE_IRQ,
    }
};

```

```
};
```

```

/* for the moment we limit ourselves to 16bit IO until some
 * better IO routines can be written and tested
 */

```

```

static struct dm9000_plat_data s3c_dm9k_platdata = {
    .flags      = DM9000_PLATF_16BITONLY,
};

```

```

static struct platform_device s3c_device_dm9k = {
    .name       = "dm9000",
    .id        = 0,
    .num_resources = ARRAY_SIZE(s3c_dm9k_resource),
    .resource   = s3c_dm9k_resource,
    .dev       = {
        .platform_data = &s3c_dm9k_platdata,
    }
};

```

```
#endif /* CONFIG_DM9000 */
```

```
/* devices we initialise */
```

```
static struct platform_device __initdata *smdk_devs[] = {
```

然后在 199 行左右，添加如下内容（红色部分所示）：

```

static struct platform_device __initdata *smdk_devs[] = {
    &s3c_device_nand,
    &smdk_led4,
    &smdk_led5,

```

```

&smdk_led6,
&smdk_led7,
#ifdef(CONFIG_DM9000) || defined(CONFIG_DM9000_MODULE)
    &s3c_device_dm9k,
#endif
};

```

修改内核源码的“drivers/net/dm9000.c”文件：

在 73 行添加如下内容（红色部分所示）：

```

#include <asm/delay.h>
#include <asm/irq.h>
#include <asm/io.h>
#ifdef(CONFIG_ARCH_S3C2410)
#include <asm/arch-s3c2410/regs-mem.h>
#endif

```

```

#include "dm9000.h"

```

把从 118 行开始修改为如下内容（红色部分所示）：

```

#ifdef CONFIG_BLACKFIN
#define readsb insb
#define readsw insw
#define readsl insl
#define writesb outsb
#define writew outsw
#define writel outsl
#define DM9000_IRQ_FLAGS (IRQF_SHARED | IRQF_TRIGGER_HIGH)
#elif defined(CONFIG_ARCH_S3C2410)
#define DM9000_IRQ_FLAGS (IRQF_SHARED | IRQF_TRIGGER_RISING)
#else
DM9000_IRQ_FLAGS IRQF_SHARED
#endif

```

在 414 行添加如下内容（红色部分所示）：

```

static int
dm9000_probe(struct platform_device *pdev)
{
    struct dm9000_plat_data *pdata = pdev->dev.platform_data;
    struct board_info *db; /* Point a board information structure */
    struct net_device *ndev;
    unsigned long base;
    int ret = 0;
    int iosize;
    int i;
    u32 id_val;

#ifdef(CONFIG_ARCH_S3C2410)

```

```

    unsigned int oldval_bwscon;
    unsigned int oldval_bankcon4;
#endif

/* Init network device */
在 428 行添加如下内容（红色部分所示）：
/* Init network device */
ndev = alloc_etherdev(sizeof (struct board_info));
if (!ndev) {
    printk("%s: could not allocate device.\n", CARDNAME);
    return -ENOMEM;
}

SET_NETDEV_DEV(ndev, &pdev->dev);

#if defined(CONFIG_ARCH_S3C2410)
    oldval_bwscon = *((volatile unsigned int *)S3C2410_BWSCON);
    *((volatile unsigned int *)S3C2410_BWSCON) = (oldval_bwscon & ~(3<<16)) \
        | S3C2410_BWSCON_DW4_16 | S3C2410_BWSCON_WS4 | S3C2410_BWSCON_ST4;

    oldval_bankcon4 = *((volatile unsigned int *)S3C2410_BANKCON4);
    *((volatile unsigned int *)S3C2410_BANKCON4) = 0x1f7c;
#endif
    PRINTK2("dm9000_probe()");

在 628 行添加内容（红色部分所示）：
out:
    printk("%s: not found (%d).\n", CARDNAME, ret);
#if defined(CONFIG_ARCH_S3C2410)
    *((volatile unsigned int *)S3C2410_BWSCON) = oldval_bwscon;
    *((volatile unsigned int *)S3C2410_BANKCON4) = oldval_bankcon4;
#endif

    dm9000_release_board(pdev, db);
    free_netdev(ndev);

    return ret;
}

```

## 13.3 配置并编译内核

修改完以上的内容之后，输入：`#make menuconfig`，进入配置单，然后添加上对 DM9000 网卡的配置：

```
Device Drivers --->
```

```
[*] Network device support --->
```

```
[*] Ethernet (10 or 100Mbit) --->
```

```
-*- Generic Media Independent Interface device support
```

```
<*> DM9000 support
```

```
(4) DM9000 maximum debug level
```

配置好后，保存配置单，然后编译出镜像，烧写到开发板中。

## 13.4 网卡配置

然后还需要修改文件系统的“/etc/init.d/rcS”文件下面列出修改的内容（红色部分所示）：

```
mount -a
```

```
mkdir /dev/pts
```

```
mount -t devpts devpts /dev/pts
```

```
echo /sbin/mdev > /proc/sys/kernel/hotplug
```

```
mdev -s
```

```
mkdir -p /var/lock
```

```
ifconfig lo 127.0.0.1 #（设置本地回环设备 IP 地址，缺省值）
```

```
ifconfig eth0 hw ether 10:23:45:67:89:ab #（设置网卡的 MAC 值）
```

```
ifconfig eth0 192.168.1.6 up #（设置网卡的 IP 并启动网卡）
```

```
route add default gw 192.168.1.2 #（设置网卡的网关）
```

```
/bin/hostname -F /etc/sysconfig/HOSTNAME
```

修改完毕文件系统后，重新制作为 yaffs 文件系统，烧写到开发板中，然后在启动开发板时就会完成对网卡 IP 等信息的设置。

下面我们添加 web 服务器到文件系统中，在前面制作文件系统时，已经将 web 服务器的相关文件添加到文件系统了，这里，我们只需要在系统启动时加载 web 服务器的进程即可，修改文件系统的“/etc/init.d/rcS”文件，添加上对 web 服务器的调用即可：（红色部分所示）

```
#!/bin/sh
```

```
PATH=/sbin:/bin:/usr/sbin:/usr/bin
```

```
runlevel=S
```

```
prevlevel=N
```

```
umask 022
```

```
export PATH runlevel prevlevel
```

```
#
```

```
# Trap CTRL-C &c only in this shell so we can interrupt subprocesses.
```

```
#
```

```
mount -a
```

```
mkdir /dev/pts
```



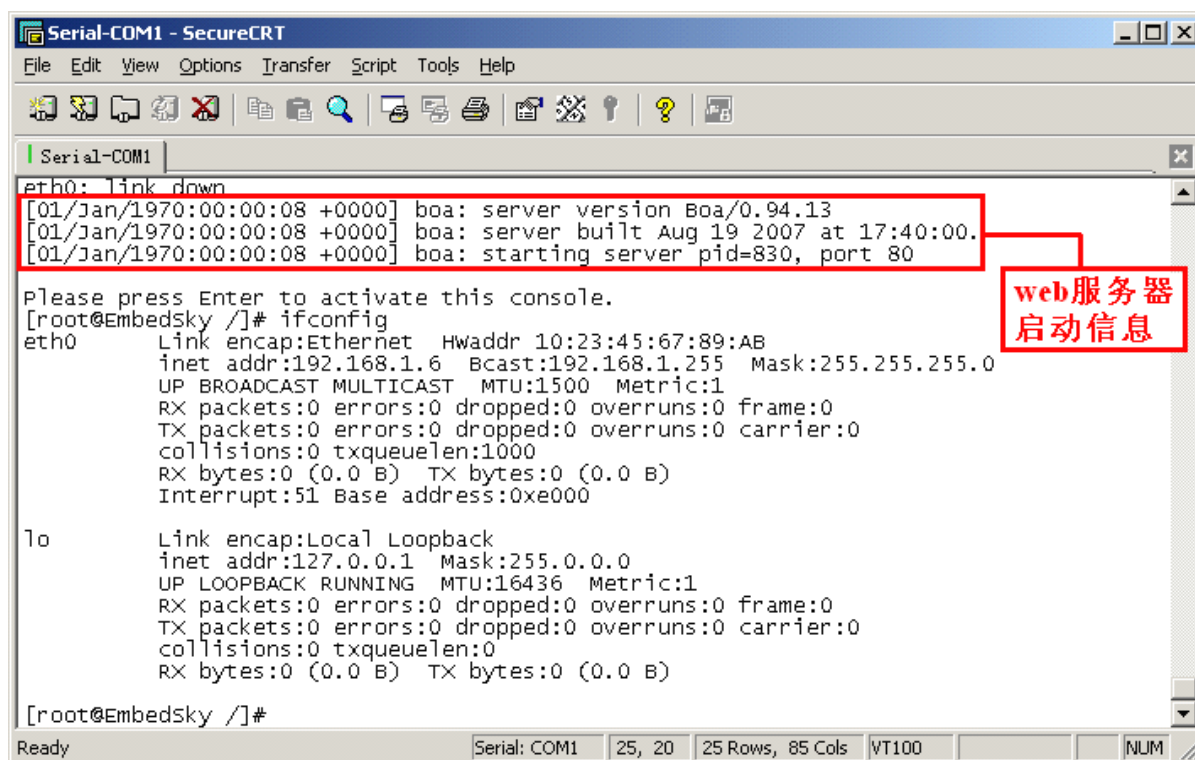
```
mount -t devpts devpts /dev/pts
echo /sbin/mdev > /proc/sys/kernel/hotplug
mdev -s
mkdir -p /var/lock
```

```
ifconfig lo 127.0.0.1
ifconfig eth0 hw ether 10:23:45:67:89:ab
ifconfig eth0 192.168.1.6 up
route add default gw 192.168.1.2
```

```
/etc/rc.d/init.d/httpd start #启动 web 服务器的代理
```

```
/bin/hostname -F /etc/sysconfig/HOSTNAME
```

完成后，重新制作文件系统的镜像，烧写到开发板中，在开发板的终端输入：`$ifconfig`，你可以看到设置的网络的信息和 Web 服务器启动的信息，如下图所示：



The screenshot shows a terminal window titled "Serial-COM1 - SecureCRT". The terminal output displays the results of the `ifconfig` command, showing the configuration for `eth0` and `lo`. A red box highlights the output of the `boa` server startup, which includes the version (Boa/0.94.13), build date (Aug 19 2007), and the server starting on pid=830, port 80. A red callout box points to this output with the text "web服务器启动信息".

```
Serial-COM1
File Edit View Options Transfer Script Tools Help

Serial-COM1
eth0: link down
[01/Jan/1970:00:00:08 +0000] boa: server version Boa/0.94.13
[01/Jan/1970:00:00:08 +0000] boa: server built Aug 19 2007 at 17:40:00.
[01/Jan/1970:00:00:08 +0000] boa: starting server pid=830, port 80
Please press Enter to activate this console.
[root@EmbedSky /]# ifconfig
eth0      Link encap:Ethernet  Hwaddr 10:23:45:67:89:AB
          inet addr:192.168.1.6  Bcast:192.168.1.255  Mask:255.255.255.0
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
          Interrupt:51 Base address:0xe000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

[root@EmbedSky /]#
```

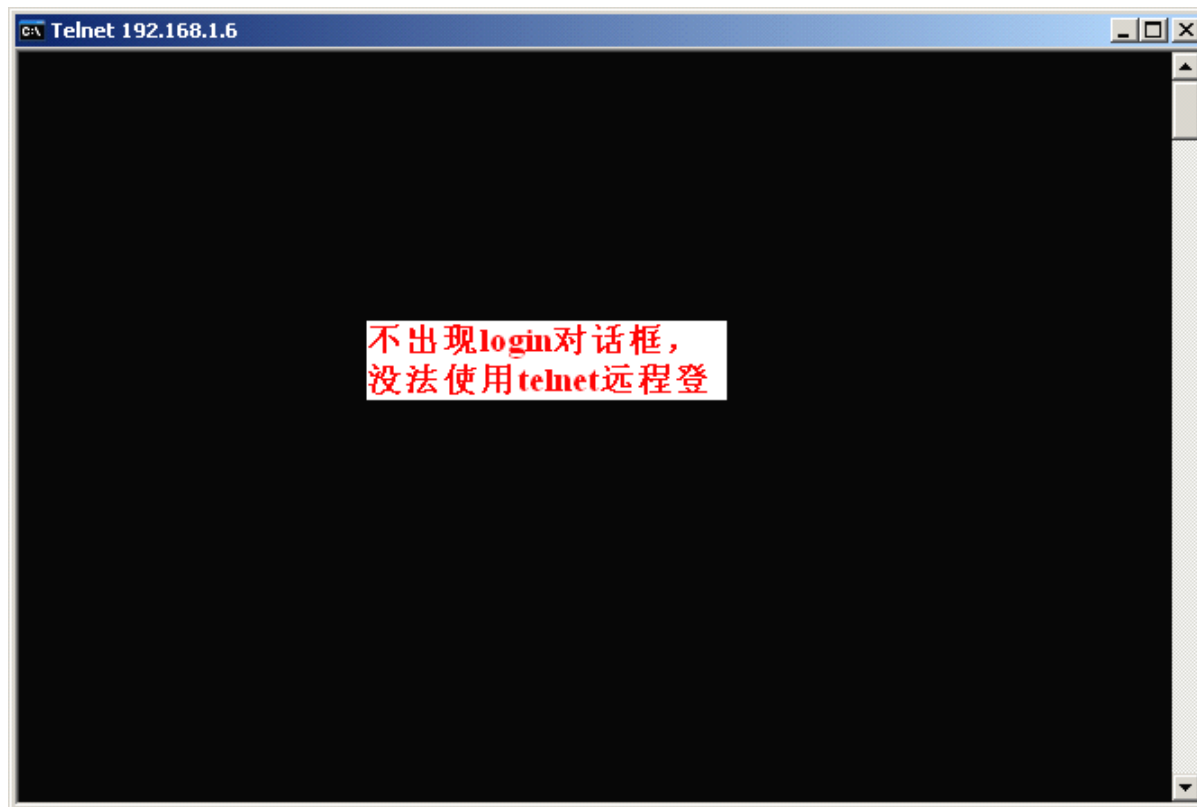
## 13.5 向文件系统添加 FTP 和 TELNET 功能

这里可以直接从 2.6.13 的文件系统里面复制对应的文件过来，首先复制 2.6.13 的文件系统的“`/etc/`”目录下面的“`rc.d/init.d/netd`”、“`ftphroot`”、“`ftpusers`”、“`inetd.conf`”和“`servers`”文件到新的文件系统的“`/etc/`”目录下，同时修改新的文件系统的“`/etc/inittab`”文件为如下所示：（红色部分为新添加的内容）

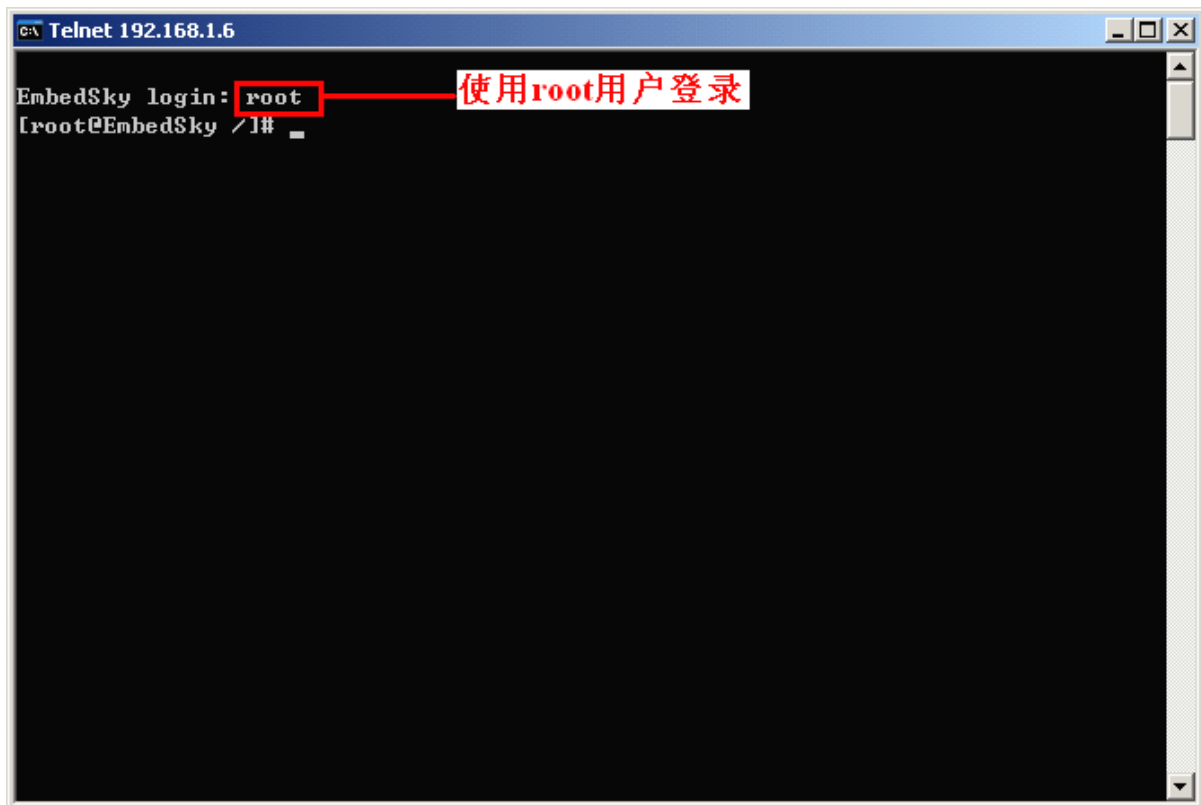
```
# /etc/inittab
::sysinit:/etc/init.d/rcS
```

```
s3c2410_serial0::askfirst:~/bin/sh  
::once:/usr/sbin/telnetd -l /bin/login  
::ctrlaltdel:/sbin/reboot  
::shutdown:/bin/umount -a -r
```

如果不添加这句话，在使用 telnet 登录时会有类似下图这样的提示信息出现，不出现正常登录 telnet 后出现的 login 对话框：



下图是正常登录时的截图：



复制 2.6.13 文件系统的“/lib/”目录下的“libwrap.so.0”库文件到新文件系统的“/lib/”目录下，因为 ftp 使用时会调用到该库文件。

复制 2.6.13 文件系统的“/usr/sbin/”目录下的“in.ftpd”和“tcpd”文件到新文件系统的“/usr/sbin/”目录下。

修改新文件系统的“/etc/init.d/rcS”文件，添加内容如下红色部分所示：

```
ifconfig lo 127.0.0.1
ifconfig eth0 hw ether 10:23:45:67:89:ab
ifconfig eth0 192.168.1.6 up
route add default gw 192.168.1.2
```

```
/etc/rc.d/init.d/httpd start
/etc/rc.d/init.d/netd start
```

```
/bin/hostname -F /etc/sysconfig/HOSTNAME
```

完成以上操作后，重新压制 yaffs 文件系统的镜像，然后烧写到开发板中，就可以使用 FTP 和 TELNET 功能了：

FTP 功能：

```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [版本 5.1.2600]
(C) 版权所有 1985-2001 Microsoft Corp.

C:\Documents and Settings\yellow>ftp 192.168.1.6
Connected to 192.168.1.6.
220 EmbedSky FTP server (Version 6.4/OpenBSD/Linux-ftpd-0.17) ready.
User (192.168.1.6:(none)): sky
331 Password required for sky.
Password:
230 User sky logged in.
ftp> by
221 Goodbye.

C:\Documents and Settings\yellow>
```

TELNET 功能:

```
C:\Telnet 192.168.1.6

EmbedSky login: root
[root@EmbedSky /]# ls
bin      home      lost+found  proc      sys      var
dev      lib       mnt        root      tmp      web
etc      linuxrc   opt        shin      usr

[root@EmbedSky /]#
```

## Step 14、USB 设备驱动移植

下面我们完成对 USB 设备的移植，其中包括 U 盘，USB 鼠标键盘，USB 摄像头等驱动的移植。

在 2.6.25.8 的内核中，已经支持 U 盘，USB 鼠标键盘了，这里我们只需要进行对应的配置，然后就可以完成对他们的支持。

### 14.1 USB 设备的配置

输入：#`make menuconfig`，然后进入配置单，配置如下：

```
Device Drivers --->
  SCSI device support --->
    <*> SCSI device support
    [*] legacy /proc/scsi/ support
    <*> SCSI disk support
    <*> SCSI CDROM support
  [*] HID Devices --->
    <*> USB Human Interface Device (full HID) support
    [*] /dev/hiddev raw HID device support
  [*] USB support --->
    <*> Support for Host-side USB
    [*] USB device filesystem
    [*] USB device class-devices (DEPRECATED)
    <*> OHCI HCD support
    <*> USB Mass Storage support
```

### 14.2 U 盘的挂载

配置完毕上面的信息，然后编译出镜像烧写到开发板中，启动后，插入 U 盘，然后挂载上 U 盘即可，如下图所示：

```
Serial-COM1 - SecureCRT
File Edit View Options Transfer Script Tools Help

Serial-COM1
yaffs: dev is 32505858 name is "mtdblock2"
yaffs: passed flags ""
yaffs: Attempting MTD mount on 31.2, "mtdblock2"
yaffs: block 3525 is marked bad
block 3526 is bad
VFS: Mounted root (yaffs filesystem).
Freeing init memory: 124k
eth0: link down

Please press Enter to activate this console. scsi 0:0:0:0: Direct-Access      Generic
STORAGE DEVICE    9451 PQ: 0 ANSI: 0
sd 0:0:0:0: [sda] 1961984 512-byte hardware sectors (1005 MB)
sd 0:0:0:0: [sda] write Protect is off
sd 0:0:0:0: [sda] Assuming drive cache: write through
sd 0:0:0:0: [sda] 1961984 512-byte hardware sectors (1005 MB)
sd 0:0:0:0: [sda] write Protect is off
sd 0:0:0:0: [sda] Assuming drive cache: write through
sda: sda1
sd 0:0:0:0: [sda] Attached SCSI removable disk

[root@EmbedSky /]# mount /dev/sda1 /mnt/udisk/
[root@EmbedSky /]# ls /mnt/udisk/
dcim  navione
[root@EmbedSky /]# umount /mnt/udisk/
[root@EmbedSky /]#

Ready Serial: COM1 25, 20 25 Rows, 85 Cols VT100 NUM
```

## 14.3 实现把开发板当“U 盘”1——修改源码

在 2.6.25.8 的内核源码中把 S3C2440 的第二个 USB 口，也就是复用的那个 USB 口默认设置为 USB device 功能的，下面我们分成两部分来说明这个 USB 口的用法以及对应的驱动的移植。

首先把开发板当成从设备来用，也就是该 USB 口使用 USB Device 功能，修改内核源码的“arch/arm/mach-s3c2440/mach-smdk2440.c”文件：

在 48 行添加如下内容（红色部分所示）：

```
#include <asm/plat-s3c24xx/udc.h>
```

在 151 行添加如下内容（红色部分所示）：

```
.lpcsel      = ((0xCE6) & ~7) | 1<<4,
};
```

```
static void EmbedSky_udc_pullup(enum s3c2410_udc_cmd_e cmd)
{
    printk(KERN_DEBUG "EmbedSky udc: pullup(%d)\n",cmd);
    switch (cmd)
    {
        case S3C2410_UDC_P_ENABLE :
            s3c2410_gpio_setpin(S3C2410_GPG12, 1);
            break;
        case S3C2410_UDC_P_DISABLE :
            s3c2410_gpio_setpin(S3C2410_GPG12, 0);
            break;
        case S3C2410_UDC_P_RESET :
```

```

                                break;
                        default:
                                break;
                }
        }

static struct s3c2410_udc_mach_info EmbedSky_udc_cfg = {
        .udc_command      = EmbedSky_udc_pullup,
};

```

```

static struct platform_device *smdk2440_devices[] __initdata = {
        &s3c_device_usb,
        &s3c_device_lcd,
        &s3c_device_wdt,
        &s3c_device_i2c,
        &s3c_device_iis,
};

```

在 180 行添加如下内容（红色部分所示）：

```

static struct platform_device *smdk2440_devices[] __initdata = {
        &s3c_device_usb,
        &s3c_device_lcd,
        &s3c_device_wdt,
        &s3c_device_i2c,
        &s3c_device_iis,
        &s3c_device_usb gadget,
};

```

在 196 行添加如下内容（红色部分所示）：

```

static void __init smdk2440_machine_init(void)
{
        s3c24xx_fb_set_platdata(&smdk2440_fb_info);

        platform_add_devices(smdk2440_devices, ARRAY_SIZE(smdk2440_devices));
        smdk_machine_init();
        s3c2410_gpio_setpin(S3C2410_GPG12, 0);
        s3c2410_gpio_cfgpin(S3C2410_GPG12, S3C2410_GPIO_OUTPUT);
        s3c24xx_udc_set_platdata(&EmbedSky_udc_cfg);
}

```

## 14.4 实现把开发板当“U 盘”2——配置内核

然后需要添加上对 USB Device 的配置，输入：`#make menuconfig`，进入“Device Drivers”选项的配置单的“USB support”选项的配置单的最后项“USB Gadget Support”选项的配置单，加入对 UDC 设备的支持，以及文件系统的支持（以模块的形式），然后配置如下所示：

```

Device Drivers --->
[*] USB support --->
    <*> USB Gadget Support --->
        USB Peripheral Controller (S3C2410 USB Device Controller)
        S3C2410 USB Device Controller
        <M> USB Gadget Drivers
        <M> File-backed Storage Gadget

```

配置完毕后，编译出镜像，烧写到开发板中，然后在 PC 的终端上使用 `#make modules SUBDIRS=drivers/usb/gadget/` 命令可以编译出来刚刚以模块形式配置的文件系统的驱动模块，最后在内核源码的“`drivers/usb/gadget/`”目录下生成名为：“`g_file_storage.ko`”的驱动模块，将其复制到文件系统的“`/lib/`”目录下面，然后烧写新的文件系统到开发板中，启动开发板。

## 14.5 实现把开发板当“U 盘”3——“U 盘”之又见“U 盘”

然后在开发板的串口终端使用下面的命令挂载刚刚编译出来的驱动模块：

```
$insmod /lib/g_file_storage.ko file=/dev/mtdblock2 removable=1
```

**注意：**红色部分就是表示要把什么地方作为挂载的“U 盘”，这里我使用了 yaffs 文件系统所在的分区作为“U 盘”的存储部分，当然，如果此时插入 U 盘到开发板的 USB 口里面，然后可以根据开发板的情况使用“`/dev/sda1`”。

然后挂载完毕后，插入 USB 线到开发板的 USB Device 接口（也就是烧写时用的那个 USB 口），然后可以在 Windows XP 中发现可移动硬盘，此时整个开发板就是一个“U 盘”，然后 yaffs 文件系统的分区就是“U 盘”的存储器，不过需要特别注意的地方：此时虽然在 windows 里面能够识别为 U 盘了，不过您对其操作时，系统会告诉要求格式化“U 盘”，强烈建议不能格式化，否则您将需要重新烧写文件系统。如果测试插入 U 盘到开发板中，“U 盘”的存储器就是插入的 U 盘，然后 windows 访问到就是出入的那个 U 盘。希望读者在这里能够区分带引号的“U 盘”和不带引号的 U 盘：带引号的“U 盘”是开发板虚拟的。

下面的截图是挂载驱动时使用了 U 盘作为存储设备时的情况：

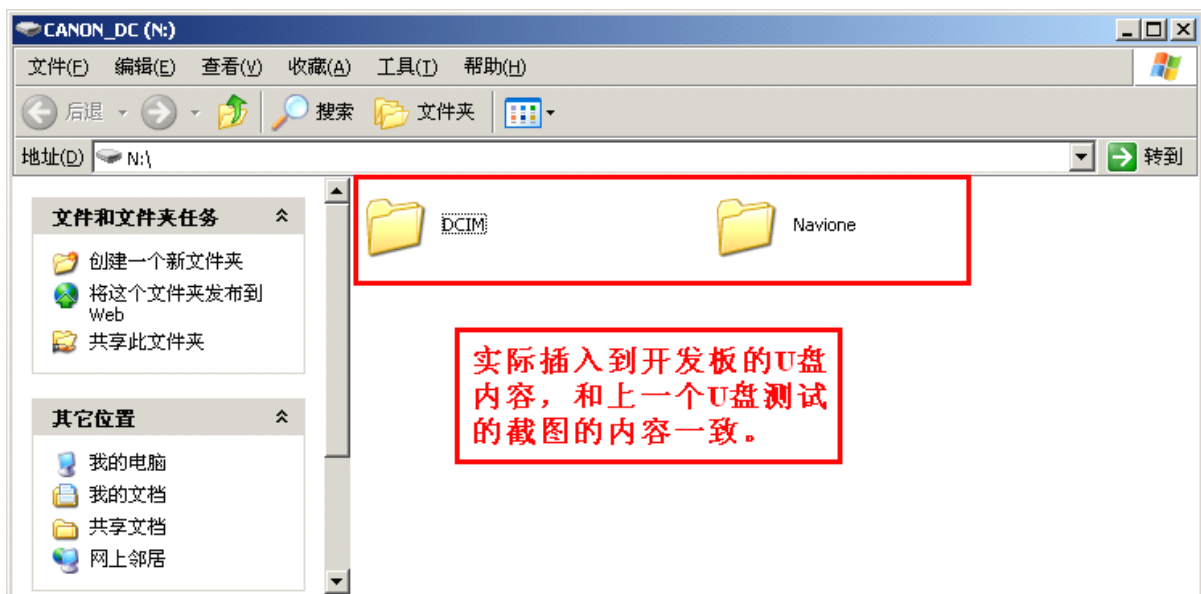


```
Serial-COM1 - SecureCRT
File Edit View Options Transfer Script Tools Help
Serial-COM1
block 3526 is bad
VFS: Mounted root (yaffs filesystem).
Freeing init memory: 124K
usb 1-1: new full speed USB device using s3c2410-ohci and address 2
usb 1-1: configuration #1 chosen from 1 choice
scsi0 : SCSI emulation for USB Mass Storage devices
eth0: link down

Please press Enter to activate this console.
[root@EmbedSky /]# scsi 0:0:0:0: Direct-Access      Generic  STORAGE DEVICE   9451 PQ:
0 ANSI: 0
sd 0:0:0:0: [sda] 1961984 512-byte hardware sectors (1005 MB)
sd 0:0:0:0: [sda] Write Protect is off
sd 0:0:0:0: [sda] Assuming drive cache: write through
sd 0:0:0:0: [sda] 1961984 512-byte hardware sectors (1005 MB)
sd 0:0:0:0: [sda] Write Protect is off
sd 0:0:0:0: [sda] Assuming drive cache: write through
sda: sda1
sd 0:0:0:0: [sda] Attached SCSI removable disk

[root@EmbedSky /]# insmod /lib/g_file_storage.ko file=/dev/sda1 removable=1
g_file_storage gadget: File-backed Storage Gadget, version: 7 August 2007
g_file_storage gadget: Number of LUNs=1
g_file_storage gadget-lun0: ro=0, file: /dev/sda1
[root@EmbedSky /]#
```

下图是 Windows XP 访问刚刚插上的“U 盘”（开发板）的情况：



## 14.6 实现双 USB Host 功能 1——源码修改

下面我们实现 USB 复用的第二个功能，USB Host 的驱动，根据 S3C2440 的 datasheet，要想它实现 USB Host 的功能，我们只需要设置 MISCCR 寄存器的位 3 即可。

修改内核源码的“drivers/usb/host/Kconfig”文件，在第 7 行添加如下内容：

```
config EmbedSky_TWO_USB_HOST
bool "EmbedSky TWO USB HOST"
---help---
```

S3C2440 can use two usb Host or one usb host and one usb device.

修改内核源码的“drivers/usb/host/ohci-s3c2410.c”文件，

在 27 行添加如下内容（红色部分所示）：

```
#include <asm/hardware.h>
#include <asm/arch/usb-control.h>
#ifdef CONFIG_EmbedSky_TWO_USB_HOST
#include <asm/arch/regs-gpio.h>
#endif
```

```
#define valid_port(idx) ((idx) == 1 || (idx) == 2)
```

在 353 行添加如下内容（红色部分所示）：

```
static int usb_hcd_s3c2410_probe (const struct hc_driver *driver,
                                struct platform_device *dev)
{
    struct usb_hcd *hcd = NULL;
    int retval;
```

```
#ifdef CONFIG_EmbedSky_TWO_USB_HOST
    printk("Initial EmbedSky TWO USB HOST Driver!\n");
    unsigned long reg_misccr;
    reg_misccr = 0;
    reg_misccr = readl(S3C2410_MISCCR);
    reg_misccr = reg_misccr | S3C2410_MISCCR_USBHOST;
    writel(reg_misccr, S3C2410_MISCCR);
#endif
```

```
s3c2410_usb_set_power(dev->dev.platform_data, 1, 1);
```

```
s3c2410_usb_set_power(dev->dev.platform_data, 2, 1);
```

## 14.6 实现双 USB Host 功能 2——配置并设置双 USB Host 口

修改完毕后，输入：`#make menuconfig`，进入“Device Drivers”选项配置单下的“USB support”选项配置单，选择对“EmbedSky\_TWO\_USB\_HOST”的支持，

```
Device Drivers --->
```

```
[*] USB support --->
```

```
[*] EmbedSky TWO USB HOST
```

配置完毕后，编译出镜像，然后烧写到开发板中，在硬件上面 SKY2440 支持有两个 USB Host 的设计，选择 J11 和 J12 两个跳线，使其支持 USB Host 功能，然后就可以使用第二个 USB 口对 U 盘或 USB 鼠标或 USB 键盘进行操作了。

## 14.7 USB 摄像头驱动移植 1——源码获取

下面进行 Z301 系列芯片组的 USB 摄像头的驱动的移植。

<http://mxhaard.free.fr> 这个网站是一个法国的老爷子专门制作 USB 摄像头驱动的开源网站。

这是驱动的下载站点: <http://mxhaard.free.fr/spca50x/Download/gspcav1-20071224.tar.gz>

不过好像现在大陆这边登录不了。对于这个 USB 摄像头驱动的移植, 我之前下载了一个驱动包, 然后我们就在该驱动包里面修改并移植到 2.6.25.8 的内核里面。

## 14.8 USB 摄像头驱动移植 2——修改驱动源码

解压 USB 摄像头的驱动板 `gspcavl.tar.bz2` (该压缩包我们已经修改过了, 所以下面涉及到修改该压缩包内文件的信息, 您均可以不用对其进行操作), 然后复制解压后的驱动包到内核的 “`drivers/media/video/`” 目录下, 修改文件 “`drivers/media/video/Kconfig`”, 在 783 行左右, 添加如下内容 (红色部分所示):

```
config USB_OV511
```

```
    tristate "USB OV511 Camera support"
```

```
    depends on VIDEO_V4L1
```

```
    ---help---
```

```
        Say Y here if you want to connect this type of camera to your
```

```
        computer's USB port. See <file:Documentation/video4linux/ov511.txt>
```

```
        for more information and for a list of supported cameras.
```

```
        To compile this driver as a module, choose M here: the
```

```
        module will be called ov511.
```

```
config USB_SPCA5XX
```

```
    tristate "USB SPCA5XX Sunplus/Vimicro/Sonix jpeg Cameras"
```

```
    depends on USB && VIDEO_V4L1
```

```
    ---help---
```

```
        Say Y or M here if you want to use one of these webcams:
```

```
        The built-in microphone is enabled by selecting USB Audio support.
```

```
        This driver uses the Video For Linux API. You must say Y or M to
```

```
        "Video For Linux" (under Character Devices) to use this driver.
```

```
        Information on this API and pointers to "v4l" programs may be found
```

```
        at <file:Documentation/video4linux/API.html>.
```

```
        To compile this driver as a module, choose M here: the
```

```
        module will be called spca5xx.
```

修改文件 “`drivers/media/video/Makefile`”, 在 117 行添加: (下面红色部分所示)

```
obj-$(CONFIG_USB_DABUSB)      += dabusb.o
```

```
obj-$(CONFIG_USB_OV511)      += ov511.o
```

```
obj-$(CONFIG_USB_SPCA5XX)    += gspcavl/
```

```
obj-$(CONFIG_USB_SE401)      += se401.o
```

```
obj-$(CONFIG_USB_STV680)     += stv680.o
```

```
obj-$(CONFIG_USB_W9968CF)    += w9968cf.o
```

```
obj-$(CONFIG_USB_ZR364XX) += zr364xx.o
obj-$(CONFIG_USB_STKWEBCAM) += stkwebcam.o
```

修改 “drivers/media/video/gspcavl/Makefile” 文件（注意，我们已经修改过了），内容如下：

```
gspca-objs := gspca_core.o decoder/gspcdecoder.o
obj-$(CONFIG_USB_SPCA5XX) += gspca.o
```

clean:

```
rm -f *.oas] *.flags *.ko *.cmd *.d *.tmp *.mod.c
rm -rf .tmp_versions
```

修改内核 “drivers/media/video/gspcavl/gspca\_core.c” 文件，把 36 行改成如下内容（红色部分所示）：

```
static const char gspca_version[] = "01.00.20";
#define VID_HARDWARE_GSPCA 0xFF
```

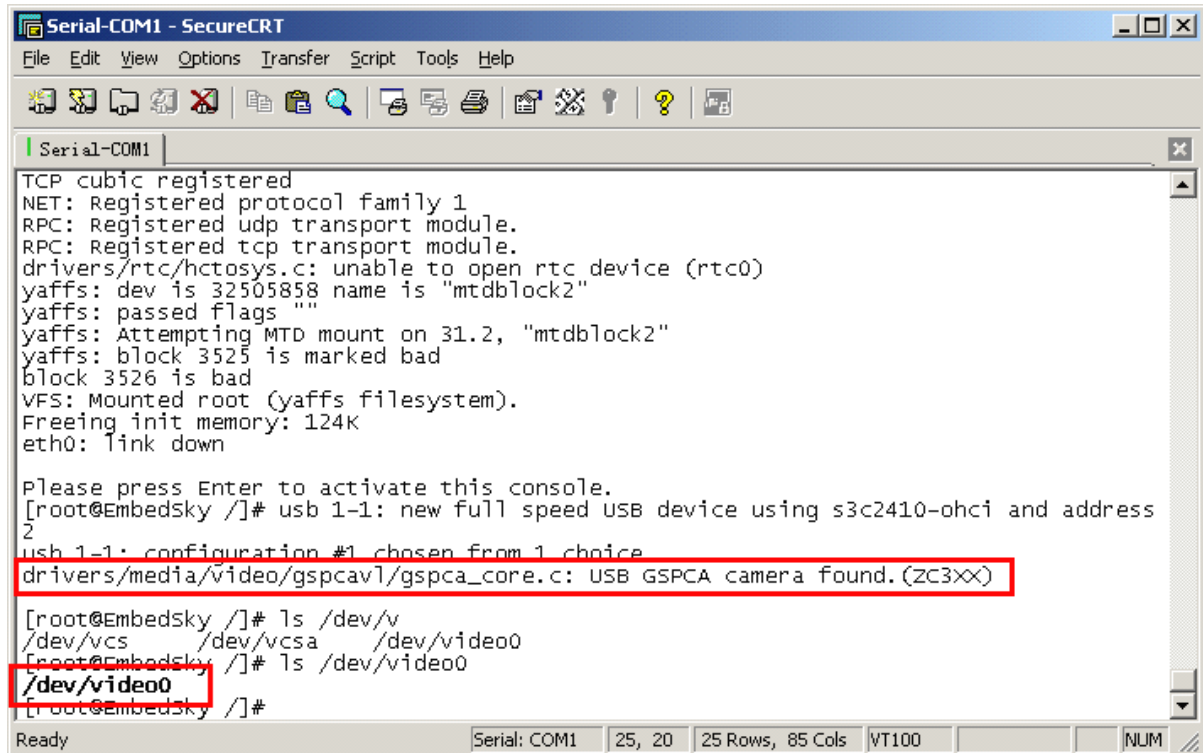
## 14.8 USB 摄像头驱动移植 3——配置内核

修改完毕后，输入：#make menuconfig，然后配置如下：

```
Device Drivers --->
  Multimedia devices --->
    <*> Video For Linux
    <*> Video For Linux
    [*] Enable Video For Linux API 1 (DEPRECATED)
    -* Enable Video For Linux API 1 compatible Layer
    [*] Video capture adapters --->
      [*] Autoselect pertinent encoders/decoders and other helper chip
      [*] Autoselect pertinent encoders/decoders and other helper chi
      [*] V4L USB devices --->
        <*> USB SPCA5XX Sunplus/Vimicro/Sonix jpeg Cameras
        <> USB ZC0301[P] Image Processor and Control Chip support
```

注意：这里有个选项是 “USB ZC0301[P] Image Processor and Control Chip support”，该选项对应的驱动没法使用 Z301 的 USB 摄像头的，所以一定不要选择对该选项的支持。

然后退出保存配置单，编译出镜像，烧写到开发板中，启动系统后，插入 USB 摄像头，会正确识别到 USB 摄像头的，同时会在文件系统的 “/dev” 目录下产生名为 “video0” 的设备，如下图所示：



```
Serial-COM1 - SecureCRT
File Edit View Options Transfer Script Tools Help

Serial-COM1
TCP cubic registered
NET: Registered protocol family 1
RPC: Registered udp transport module.
RPC: Registered tcp transport module.
drivers/rtc/hctosys.c: unable to open rtc device (rtc0)
yaffs: dev is 32505858 name is "mtdblock2"
yaffs: passed flags ""
yaffs: Attempting MTD mount on 31.2, "mtdblock2"
yaffs: block 3525 is marked bad
block 3526 is bad
VFS: Mounted root (yaffs filesystem).
Freeing init memory: 124K
eth0: link down

Please press Enter to activate this console.
[root@Embedsky /]# usb 1-1: new full speed USB device using s3c2410-ohci and address
2
usb 1-1: configuration #1 chosen from 1 choice
drivers/media/video/gspcav1/gspca_core.c: USB GSPCA camera found.(ZC3XX)

[root@Embedsky /]# ls /dev/v
/dev/vcs      /dev/vcsa    /dev/video0
[root@Embedsky /]# ls /dev/video0
/dev/video0
[root@Embedsky /]#

Ready Serial: COM1 25, 20 25 Rows, 85 Cols VT100 NUM
```

## 14.9 USB 摄像头驱动移植 4——USB 摄像头的测试

拷贝 2.6.13 的文件系统的“/usr/bin/”目录下面的“servfox”和“spcacad”到新的文件系统的“/usr/bin/”目录下面，然后修改新文件系统的“/etc/init.d/rcS”文件，添加内容如下：

```
mount -a
mkdir /dev/pts
mount -t devpts devpts /dev/pts
echo /sbin/mdev > /proc/sys/kernel/hotplug
mdev -s
mkdir -p /var/lock
```

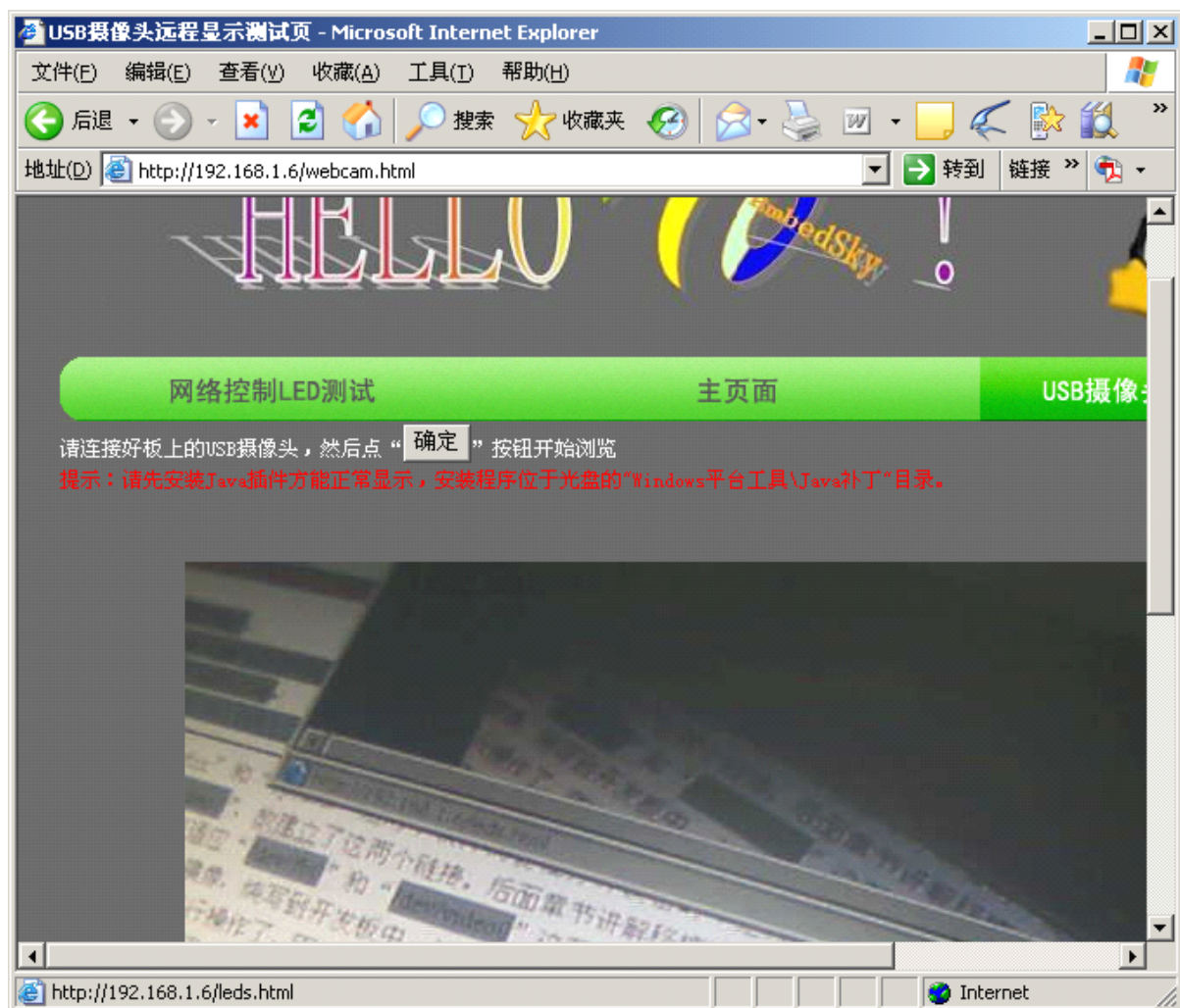
```
mkdir /dev/fb /dev/v4l
ln -s /dev/fb0 /dev/fb/0
ln -s /dev/video0 /dev/v4l/video0
```

```
ifconfig lo 127.0.0.1
ifconfig eth0 hw ether 10:23:45:67:89:ab
ifconfig eth0 192.168.1.6 up
route add default gw 192.168.1.2
```

添加的原因是“servfox”和“spcacad”这两个软件需要在打开设备时不是打开开发板上默认的“/dev/fb0”和“/dev/video0”，故建立了这两个链接。后面章节讲解移植这两个软件时，可以修改这两个应用程序的源码以适应“/dev/fb0”和“/dev/video0”这两个设备。

完成后，压制出新的 yaffs 镜像，烧写到开发板中，然后待系统启动后，插入 USB 摄像头，我们就可以使用 servfox 对摄像头进行操作了，因为这里还没有移植 LCD 的驱动，所以\$**servfox -L** 这

个命令暂时不能用了，我们可以使用 web 服务器，进行查看：



关于 servfox 和 spccat 这两个命令的使用可以参考 SKY2440 或 TQ2440 的使用手册里面的说明。



## Step 15、LCD 驱动移植

### 15.1 LCD 时钟计算方法分析

因为前面移植 USB 摄像头后，如果想要测试 USB 摄像头需要用到 LCD，那么我们这里就先进行 LCD 的移植。

在 2.6.25.8 的内核里面已经有了完善的 LCD 驱动程序了，并且也提供了一个参考的配置单，我们只需要简单的处理即可应用 TQ2440 或 SKY2440 的 LCD 控制器了。

内核源码的“drivers/video/s3c2410fb.c”文件，就是 LCD 的驱动源码。这里我们只分析一下 VCLK 的计算方法，根据 S3C2440 的技术手册描述，对于 TFT 的 LCD 而言， $VCLK = HCLK / [(CLKVAL + 1) * 2]$ （公式 1）；在该驱动文件的 149 行处开始，有个名为“s3c2410fb\_calc\_pixclk”的函数，该函数实现 LCD 的时钟频率的计算，在 431 行的“int clkdiv = s3c2410fb\_calc\_pixclk(fbi, var->pixclock) / 2;”中的 clkdiv 变量就是 S3C2440 的技术手册里面讲到的 CLKVAL，注意从 437 行到 446 行，因为我们用的是 TFT 的 LCD，所以将 clkdiv 自减 1，然后调用函数 S3C2410\_LCDCON1\_CLKVAL（）把 CLKVAL 的值添加到 LCDCON1 寄存器的相应位置中。所以，我们这里得到  $(CLKVAL + 1) \times 2 = \text{div} = \text{pixclk} \times \text{clk} \div 10^{12}$ ，即  $CLKVAL = \text{pixclk} \times \text{clk} \div 10^{12} \div 2 - 1$ （公式 2），然后再根据 152 行 clk 的计算公式，我们可以知道  $\text{clk} = HCLK$ ，然后将公式 1 和公式 2 合并，得到如下的计算公式： $VCLK = 10^{12} \div \text{pixclk}$ ，这里的 pixclk 的值就是“arch/arm/mach-s3c2440/mach-smdk2440.c”文件中的 120 行的“pixclock = 166667;”。比如，我们将 LCD 的工作频率 VCLK 设置为：3.84MHz，那么 pixclock 的值就是：260000。

### 15.2 简化 LCD 时钟计算方法

虽然前面我们分析出来了 VCLK 和 pixclock 的关系，可是这个计算公式还是太麻烦了，要是能像 S3C2440 的技术手册描述那样直接设置 CLKVAL 的值，而不用套用公式计算就非常方便了，Linux 的精髓就在 DIY，这里我们修改“drivers/video/s3c2410fb.c”文件，修改内容如下（红色部分）：

```
static void s3c2410fb_activate_var(struct fb_info *info)
{
    struct s3c2410fb_info *fbi = info->par;
    void __iomem *regs = fbi->io;
    int type = fbi->regs.lcdcon1 & S3C2410_LCDCON1_TFT;
    struct fb_var_screeninfo *var = &info->var;
    struct s3c2410fb_mach_info *mach_info = fbi->dev->platform_data;
    struct s3c2410fb_display *default_display = mach_info->displays +
                                                mach_info->default_display;
    int clkdiv = s3c2410fb_calc_pixclk(fbi, var->pixclock) / 2;

    dprintk("%s: var->xres = %d\n", __FUNCTION__, var->xres);
    dprintk("%s: var->yres = %d\n", __FUNCTION__, var->yres);
    dprintk("%s: var->bpp = %d\n", __FUNCTION__, var->bits_per_pixel);
```

```

if (type == S3C2410_LCDCON1_TFT) {
    s3c2410fb_calculate_tft_lcd_regs(info, &fbi->regs);
    --clkdiv;
    if (clkdiv < 0)
        clkdiv = 0;
} else {
    s3c2410fb_calculate_stn_lcd_regs(info, &fbi->regs);
    if (clkdiv < 2)
        clkdiv = 2;
}

// fbi->regs.lcdcon1 |= S3C2410_LCDCON1_CLKVAL(clkdiv);
fbi->regs.lcdcon1 |= S3C2410_LCDCON1_CLKVAL(default_display->setclkval);

/* write new registers */

```

完成了这部分修改之后，我们在 s3c2410fb\_display 结构体中添加了一个 setclkval 的变量，那么我们需要在该结构体的原型中添加上该变量，修改“include/asm-arm/arch-s3c2410/fb.h”文件，在 40 行添加如下内容（红色部分所示）：

```

/* LCD description */
struct s3c2410fb_display {
    /* LCD type */
    unsigned type;

    /* Screen size */
    unsigned short width;
    unsigned short height;

    /* Screen info */
    unsigned short xres;
    unsigned short yres;
    unsigned short bpp;

    unsigned pixclock; /* pixclock in picoseconds */
    unsigned setclkval; /* clkval */
    unsigned short left_margin; /* value in pixels (TFT) or HCLKs (STN) */
    unsigned short right_margin; /* value in pixels (TFT) or HCLKs (STN) */
    unsigned short hsync_len; /* value in pixels (TFT) or HCLKs (STN) */
    unsigned short upper_margin; /* value in lines (TFT) or 0 (STN) */
    unsigned short lower_margin; /* value in lines (TFT) or 0 (STN) */
    unsigned short vsync_len; /* value in lines (TFT) or 0 (STN) */

    /* lcd configuration registers */
    unsigned long lcdcon5;

```



```
};
```

## 15.3 修改 LCD 参数设置

然后我们修改 LCD 各个参数的配置，该配置参数在“arch/arm/mach-s3c2440/mach-smdk2440.c”文件中的由 107 行开始的结构体中，然后将其改为如下内容即可（红色部分就是修改的内容）：

（**注意：**添加上刚刚添加的那个变量 setclkval 的赋值）

```
/* LCD driver info */
```

```
static struct s3c2410fb_display smdk2440_lcd_cfg __initdata = {
```

```
    .lcdcon5 = S3C2410_LCDCON5_FRM565 |  
                S3C2410_LCDCON5_INVVLINE |  
                S3C2410_LCDCON5_INVVFRAME |  
                S3C2410_LCDCON5_PWREN |  
                S3C2410_LCDCON5_HWSWP,
```

```
    .type     = S3C2410_LCDCON1_TFT,
```

```
    .width    = 320,  
    .height   = 240,
```

```
    .pixclock  = 100000, /* HCLK 100 MHz, divisor 3 */  
    .setclkval = 0x3,  
    .xres      = 320,  
    .yres      = 240,  
    .bpp       = 16,  
    .left_margin = 33,  
    .right_margin = 22,  
    .hsync_len  = 44,  
    .upper_margin = 9,  
    .lower_margin = 3,  
    .vsync_len  = 15,  
};
```

```
static struct s3c2410fb_mach_info smdk2440_fb_info __initdata = {
```

```
    .displays = &smdk2440_lcd_cfg,  
    .num_displays = 1,  
    .default_display = 0,
```

```
#if 0
```

```
    /* currently setup by downloader */  
    .gpcccon = 0xaa940659,  
    .gpcccon_mask = 0xffffffff,
```

```

.gpcup      = 0x0000ffff,
.gpcup_mask = 0xffffffff,
.gpdcon     = 0xaa84aaa0,
.gpdcon_mask = 0xffffffff,
.gpdup      = 0x0000faff,
.gpdup_mask = 0xffffffff,
#endif

// .lpcsel      = ((0xCE6) & ~7) | 1<<4,
};

```

## 15.4 添加对多种 LCD 的支持

为了支持更多分辨率的 LCD，我们这里再次修改这个结构体，修改内容如下（红色部分为新添加的内容）：

```

/* LCD driver info */

static struct s3c2410fb_display smdk2440_lcd_cfg __initdata = {

    .lcdcon5 = S3C2410_LCDCON5_FRM565 |
               S3C2410_LCDCON5_INVVLINE |
               S3C2410_LCDCON5_INVVFRAME |
               S3C2410_LCDCON5_PWREN |
               S3C2410_LCDCON5_HWSWP,

    .type      = S3C2410_LCDCON1_TFT,

#if defined(CONFIG_FB_S3C24X0_T240320)
    .width      = 240,
    .height     = 320,

    .pixclock   = 100000, /* HCLK 100 MHz, divisor 4 */
    .setclkval  = 0x4,
    .xres       = 240,
    .yres       = 320,
    .bpp        = 16,
    .left_margin = 3, /* for HFPD */
    .right_margin = 6, /* for HBPD */
    .hsync_len   = 1, /* for HSPW */
    .upper_margin = 2, /* for VFPD */
    .lower_margin = 1, /* for VBPD */
    .vsync_len   = 1, /* for VSPW */

#elif defined(CONFIG_FB_S3C24X0_S320240)

```

```
.width      = 320,  
.height     = 240,
```

```
.pixclock    = 80000, /* HCLK 100 MHz, divisor 3 */  
.setclkval   = 0x3,  
.xres        = 320,  
.yres        = 240,  
.bpp         = 16,  
.left_margin = 15,    /* for HFPD */  
.right_margin = 5,    /* for HBPD */  
.hsync_len   = 8,     /* for HSPW */  
.upper_margin = 5,    /* for VFPD */  
.lower_margin = 3,    /* for VBPD */  
.vsync_len   = 15,    /* for VSPW */
```

```
#elif defined(CONFIG_FB_S3C24X0_W320240)
```

```
.width      = 320,  
.height     = 240,
```

```
.pixclock    = 80000, /* HCLK 100 MHz, divisor 3 */  
.setclkval   = 0x3,  
.xres        = 320,  
.yres        = 240,  
.bpp         = 16,  
.left_margin = 28,    /* for HFPD */  
.right_margin = 24,   /* for HBPD */  
.hsync_len   = 42,    /* for HSPW */  
.upper_margin = 6,    /* for VFPD */  
.lower_margin = 2,    /* for VBPD */  
.vsync_len   = 12,    /* for VSPW */
```

```
#elif defined(CONFIG_FB_S3C24X0_LCD480272)
```

```
.width      = 480,  
.height     = 272,
```

```
.pixclock    = 40000, /* HCLK 100 MHz, divisor 1 */  
.setclkval   = 0x1,  
.xres        = 480,  
.yres        = 272,  
.bpp         = 16,  
.left_margin = 15,    /* for HFPD */  
.right_margin = 5,    /* for HBPD */  
.hsync_len   = 35,    /* for HSPW */  
.upper_margin = 4,    /* for VFPD */
```

```

        .lower_margin = 2, /* for VBPD*/
        .vsync_len    = 8, /* for VSPW*/

#elif defined(CONFIG_FB_S3C24X0_TFT640480)
        .width        = 640,
        .height       = 480,

        .pixclock      = 40000, /* HCLK 100 MHz, divisor 1 */
        .setclkval     = 0x1,
        .xres          = 640,
        .yres          = 480,
        .bpp           = 16,
        .left_margin   = 40, /* for HFPD*/
        .right_margin  = 67, /* for HBPD*/
        .hsync_len     = 31, /* for HSPW*/
        .upper_margin  = 5, /* for VFPD*/
        .lower_margin  = 25, /* for VBPD*/
        .vsync_len     = 1, /* for VSPW*/

#elif defined(CONFIG_FB_S3C24X0_S800480)
        .width        = 800,
        .height       = 480,

        .pixclock      = 40000, /* HCLK 100 MHz, divisor 1 */
        .setclkval     = 0x1,
        .xres          = 800,
        .yres          = 480,
        .bpp           = 16,
        .left_margin   = 15, /* for HFPD*/
        .right_margin  = 47, /* for HBPD*/
        .hsync_len     = 95, /* for HSPW*/
        .upper_margin  = 9, /* for VFPD*/
        .lower_margin  = 5, /* for VBPD*/
        .vsync_len     = 1, /* for VSPW*/

#endif
};

```

然后修改“drivers/video/Kconfig”文件，把从 1798 到 1819 行的内容改成如下所示：

```

config FB_S3C24X0
    tristate "S3C24X0 LCD framebuffer support"
    depends on FB && ARCH_S3C2410
    select FB_CFB_FILLRECT
    select FB_CFB_COPYAREA

```

```
select FB_CFB_IMAGEBLIT
```

```
---help---
```

```
Frame buffer driver for the built-in LCD controller in the Samsung  
S3C2410 processor.
```

```
This driver is also available as a module (= code which can be  
inserted and removed from the running kernel whenever you want). The  
module will be called s3c2410fb. If you want to compile it as a module,  
say M here and read <file:Documentation/kbuild/modules.txt>.
```

```
If unsure, say N.
```

```
choice
```

```
prompt "LCD select"
```

```
depends on FB_S3C24X0
```

```
help
```

```
S3C24x0 LCD size select
```

```
config FB_S3C24X0_S320240
```

```
boolean "3.5 inch 320x240 Samsung LCD"
```

```
depends on FB_S3C24X0
```

```
help
```

```
3.5 inch 320x240 Samsung LCD
```

```
config FB_S3C24X0_W320240
```

```
boolean "3.5 inch 320x240 WanXin LCD"
```

```
depends on FB_S3C24X0
```

```
help
```

```
3.5 inch 320x240 WanXin LCD
```

```
config FB_S3C24X0_T240320
```

```
boolean "3.5 inch 240X320 Toshiba LCD"
```

```
depends on FB_S3C24X0
```

```
help
```

```
3.5 inch 240x320 Toshiba LCD
```

```
config FB_S3C24X0_LCD480272
```

```
boolean "4.3 inch 480X272 CHIMEI LCD"
```

```
depends on FB_S3C24X0
```

```
help
```

```
4.3 inch 480X272 CHIMEI LCD
```

```
config FB_S3C24X0_TFT640480
```

```
boolean "8.4 inch 640x480 TFT LCD"
```

```

depends on FB_S3C24X0
help
8.4 inch 640x480 TFT LCD

config FB_S3C24X0_S800480
    boolean "7 inch 800x480 Samsung LCD"
    depends on FB_S3C24X0
    help
    7 inch 800x480 Samsung LCD

endchoice

config FB_S3C2410_DEBUG
    bool "S3C2410 lcd debug messages"
    depends on FB_S3C2410
    help
    Turn on debugging messages. Note that you can set/unset at run time
    through sysfs

```

然后修改“drivers/video/Makefile”文件，把 109 行的内容改成如下所示：（红色部分所示）

```

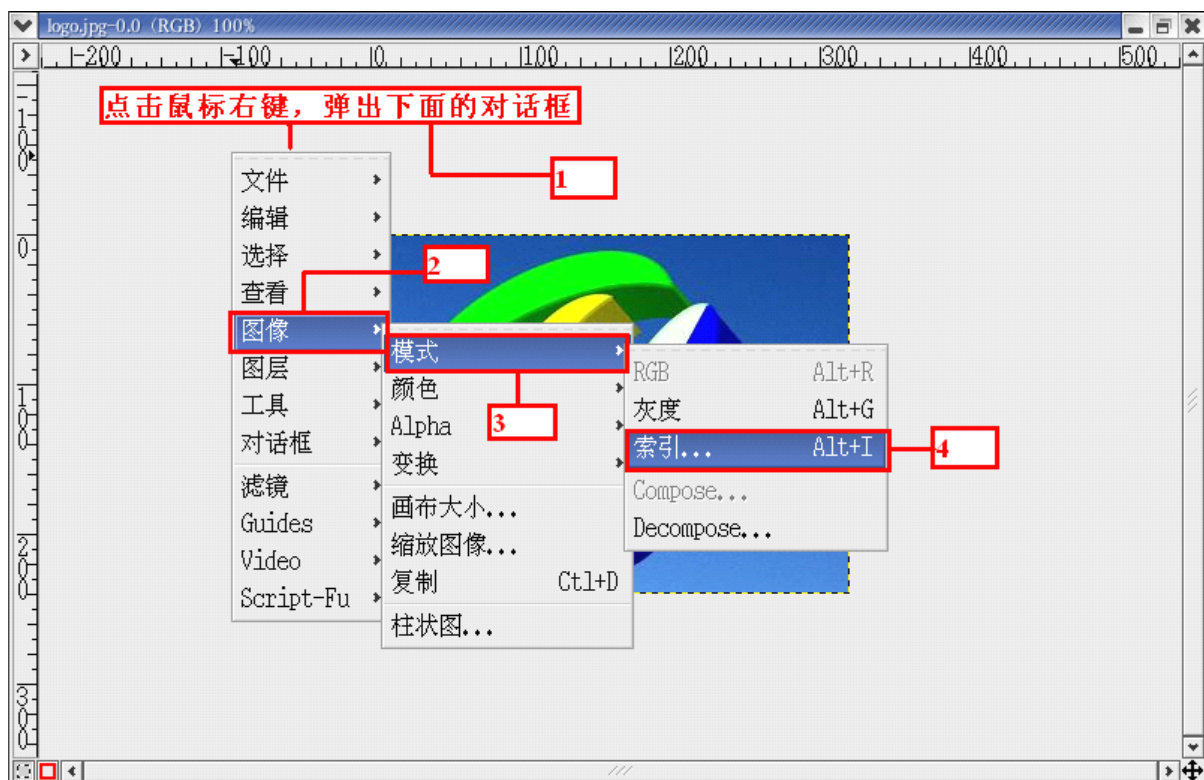
obj-$(CONFIG_FB_MAXINE)      += maxinefb.o
obj-$(CONFIG_FB_METRONOME)   += metronomefb.o
obj-$(CONFIG_FB_S1D13XXX)    += s1d13xxfb.o
obj-$(CONFIG_FB_IMX)         += imxfb.o
obj-$(CONFIG_FB_S3C24X0)     += s3c2410fb.o
obj-$(CONFIG_FB_PNX4008_DUM)  += pnx4008/
obj-$(CONFIG_FB_PNX4008_DUM_RGB) += pnx4008/
obj-$(CONFIG_FB_IBM_GXT4500) += gxt4500.o
obj-$(CONFIG_FB_PS3)         += ps3fb.o
obj-$(CONFIG_FB_SM501)       += sm501fb.o
obj-$(CONFIG_FB_XILINX)      += xilinuxfb.o

```

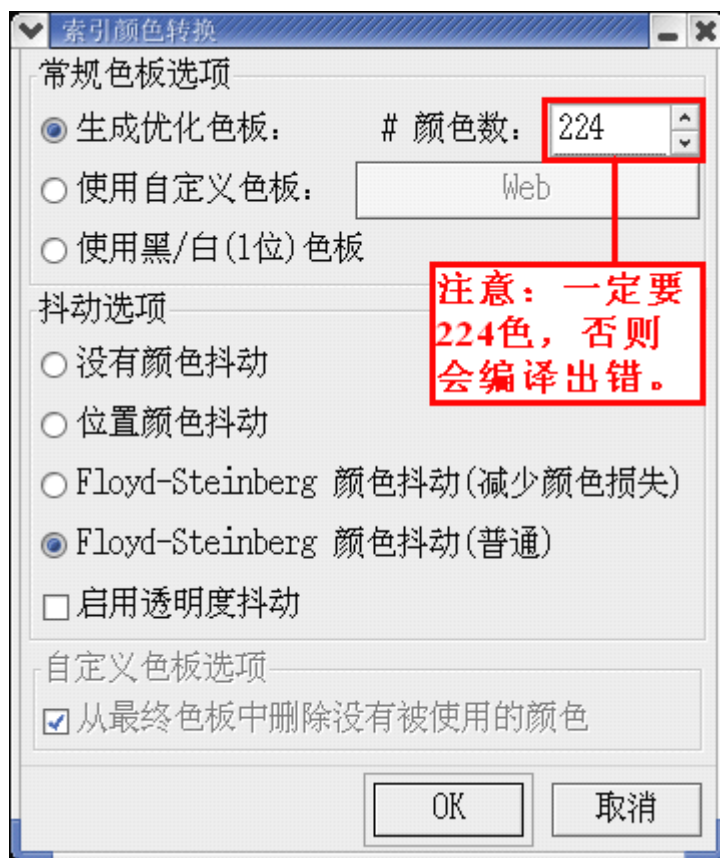
## 15.5 制作 Linux 的开机 logo 之 1——图片处理

修改“drivers/video/logo/logo\_linux\_clut224.ppm”文件，实际上是替换该文件。

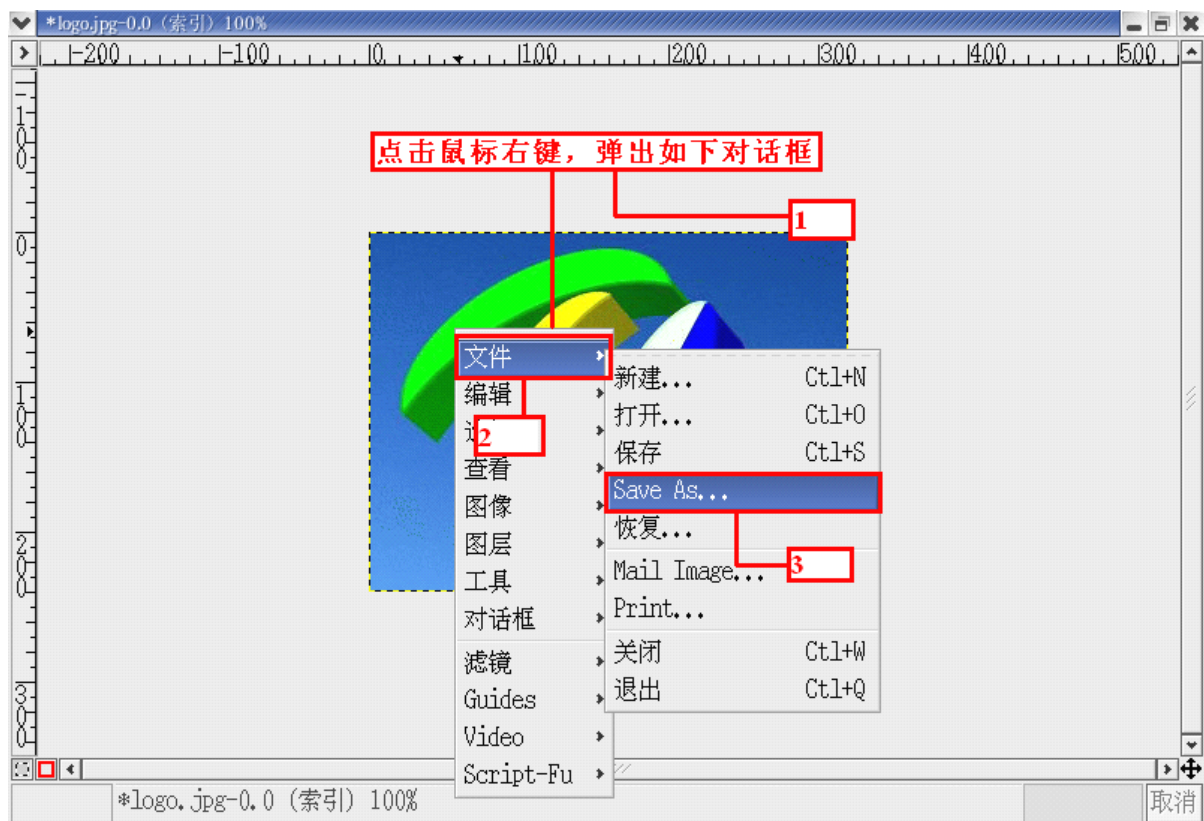
复制开机 logo 的图片文件到 RedHat9 下面，然后对着 logo 图片点击鼠标右键，在出现的弹出菜单中选择“打开方式-》The GIMP”，然后使用 GIMP 软件打开 logo 图片文件，打开之后，然后操作如下图所示：



打开索引颜色转换的对话框之后，操作如下图所示：

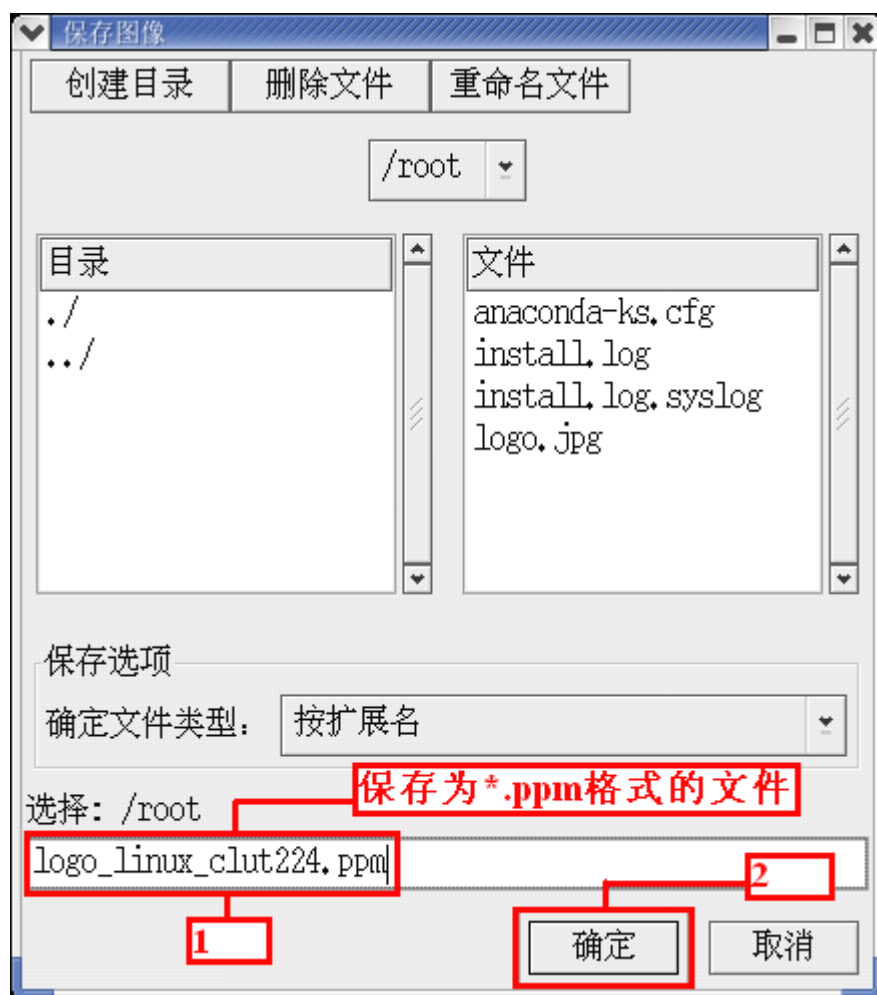


设置完毕颜色数之后，点击 OK 确认，然后操作如下图所示：

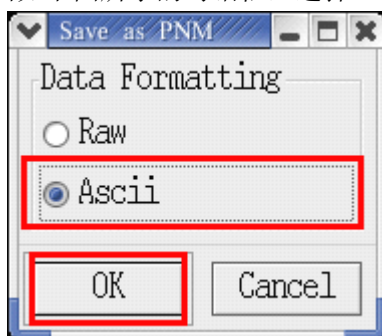


然后在弹出来的保存图像对话框中输入需要保持的文件名，同时把后缀名（.ppm 格式的图片文件）也添加上，如下图所示：





点击上图的“确定”后，弹出如下图所示的对话框，选择“Ascii”选项，如下图所示：



点击上图中的“OK”选项后，GIMP 软件将会自动把图片处理成 224 色的 PPM 格式的图片，然后系统处理完毕后，将该文件复制到内核源码中的“`drivers/video/logo/`”目录下替换掉原有的“`logo_linux_clut224.ppm`”文件。

## 15.6 制作 Linux 的开机 logo 之 2——支持更多 LCD 的设置

为了支持更多的 LCD 的开机 logo，我们还需要进行添加支持别的分辨率的开机 logo 图片，在配置文件中添加上对新添加的图片的支持。

这里我在“`drivers/video/logo/`”目录下添加了对 7 寸屏和东芝屏支持的 logo 图片：`logo_linux_s70_clut224.ppm` 和 `logo_linux_t35_clut224.ppm` 两个文件。

然后修改内核源码“`drivers/video/logo/`”目录下的“`Kconfig`”文件，在第 26 行添加如下内容：

（红色部分所示）

```
config LOGO_LINUX_VGA16
```

```
bool "Standard 16-color Linux logo"
```

```
default y
```

```
config LOGO_LINUX_CLUT224
```

```
bool "Standard 224-color Linux logo"
```

```
depends on LOGO && FB_S3C24X0_S320240 || FB_S3C24X0_W320240
```

```
default y
```

```
config LOGO_LINUX_S70_CLUT224
```

```
bool "Standard 224-color Linux logo for S70"
```

```
depends on LOGO && FB_S3C24X0_S800480
```

```
default y
```

```
config LOGO_LINUX_T35_CLUT224
```

```
bool "Standard 224-color Linux logo for T35"
```

```
depends on LOGO && FB_S3C24X0_T240320
```

```
default y
```

```
config LOGO_DEC_CLUT224
```

```
bool "224-color Digital Equipment Corporation Linux logo"
```

```
depends on MACH_DECSTATION || ALPHA
```

```
default y
```

然后修改同目录下的“Makefile”文件，在第 6 行处开始添加如下内容：（红色部分所示）

```
# Makefile for the Linux logos
```

```
obj-$(CONFIG_LOGO) += logo.o
```

```
obj-$(CONFIG_LOGO_LINUX_MONO) += logo_linux_mono.o
```

```
obj-$(CONFIG_LOGO_LINUX_VGA16) += logo_linux_vga16.o
```

```
obj-$(CONFIG_LOGO_LINUX_CLUT224) += logo_linux_clut224.o
```

```
obj-$(CONFIG_LOGO_LINUX_S70_CLUT224) += logo_linux_s70_clut224.o
```

```
obj-$(CONFIG_LOGO_LINUX_T35_CLUT224) += logo_linux_t35_clut224.o
```

```
obj-$(CONFIG_LOGO_DEC_CLUT224) += logo_dec_clut224.o
```

```
obj-$(CONFIG_LOGO_MAC_CLUT224) += logo_mac_clut224.o
```

```
obj-$(CONFIG_LOGO_PARISC_CLUT224) += logo_parisc_clut224.o
```

然后修改同目录下的“logo.c”文件，

在第 26 行添加如下内容：（红色部分所示）

```
extern const struct linux_logo logo_linux_mono;
```

```
extern const struct linux_logo logo_linux_vga16;
```

```
extern const struct linux_logo logo_linux_clut224;
```

```
extern const struct linux_logo logo_linux_t35_clut224;
```

```
extern const struct linux_logo logo_linux_s70_clut224;
extern const struct linux_logo logo_dec_clut224;
extern const struct linux_logo logo_mac_clut224;
extern const struct linux_logo logo_parisc_clut224;
extern const struct linux_logo logo_sgi_clut224;
extern const struct linux_logo logo_sun_clut224;
extern const struct linux_logo logo_superh_mono;
extern const struct linux_logo logo_superh_vga16;
extern const struct linux_logo logo_superh_clut224;
extern const struct linux_logo logo_m32r_clut224;
```

在第 77 行添加如下内容：（红色部分所示）

```
if (depth >= 8) {
#ifdef CONFIG_LOGO_LINUX_CLUT224
    /* Generic Linux logo */
    logo = &logo_linux_clut224;
#endif
#ifdef CONFIG_LOGO_LINUX_T35_CLUT224
    /* Generic Linux logo */
    logo = &logo_linux_t35_clut224;
#endif
#ifdef CONFIG_LOGO_LINUX_S70_CLUT224
    /* Generic Linux logo */
    logo = &logo_linux_s70_clut224;
#endif
#ifdef CONFIG_LOGO_DEC_CLUT224
    /* DEC Linux logo on MIPS/MIPS64 or ALPHA */
    logo = &logo_dec_clut224;
#endif
}
```

## 15.6 配置内核

做完以上的这些操作之后，我们就可以对 LCD 进行配置了，输入：`#make menuconfig`，进入配置单，然后进行如下配置：

```
Device Drivers --->
  Graphics support --->
    <*> Support for frame buffer devices --->
      [*] Enable firmware EDID
      [*] Enable Video Mode Handling Helpers
      *** Frame buffer hardware drivers ***
      <*> S3C24X0 LCD framebuffer support
      LCD select (3.5 inch 320x240 WanXin LCD) --->
      <*> Framebuffer Console support
      [*] Bootup logo --->
```

[\*] Standard 224-color Linux logo

在“LCD select (3.5 inch 320x240 WanXin LCD)”这个选项这里选择不同的 LCD 类型，下面就是进入该配置选项出现的对话框的情况，有“X”的是表示选中的：

```
( ) 3.5 inch 320x240 Samsung LCD
(X) 3.5 inch 320x240 WanXin LCD
( ) 3.5 inch 240X320 Toshiba LCD
( ) 4.3 inch 480X272 CHIMEI LCD
( ) 8.4 inch 640x480 TFT LCD
( ) 7 inch 800x480 Samsung LCD
```

上图是配置的东华屏；如果要配置三星 3.5 寸屏，需要选择：“3.5 inch 320x240 Samsung LCD”选项；如果要配置东芝 3.5 寸屏，需要选择：“3.5 inch 240x320 Toshiba LCD”选项；如果要配置奇美 4.3 寸屏，需要选择：“4.3 inch 480x272 CHIMEI LCD”选项；如果要配置 7 寸屏，需要选择：“7 inch 800x480 Samsung LCD”选项。

**注意：**在上面选择不同类型 LCD 配置时，同时开机 logo 的配置选项也会随之改变。

进入“Bootup logo”选项的配置单，并配置如下图所示：（**注意：**这里前面配置的不同类型的 LCD 在这里会出现不同的配置选项，我们只需要选择是 224 色的就行了。）

```
----- Bootup logo -----
Arrow keys navigate the menu. <Enter> selects submenus --->.
Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes,
<M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </>
for Search. Legend: [*] built-in [ ] excluded <M> module <>

--- Bootup logo
[ ] Standard black and white Linux logo
[ ] Standard 16-color Linux logo
[*] Standard 224-color Linux logo
```

上图是 320×240 分辨率的 LCD 所使用的开机 logo 的配置选项。而对于 240×320 的 LCD 所示用的配置选项为“Standard 224-color Linux logo for T35”，对于 800×480 的 LCD 所示用的配置选项为“Standard 224-color Linux logo for S70”。

配置完毕后，编译出内核镜像，烧写到 TQ2440 或 SKY2440 开发板中，启动系统时，就可以看到开机 logo 的画面，这就表明 LCD 驱动完成了。

这里也可以插入 USB 摄像头，在开发板的串口终端输入：**\$servfox -L**，把 USB 摄像头采集到的图像播放到 LCD 上面。

## 15.7 LCD 背光控制之 1——添加驱动代码

控制 LCD 背光的开关对于天嵌科技的 2440 开发板来讲，就是控制 S3C2440 的 LCD 控制的 LCD\_PWREN 脚，根据 S3C2440 的 datasheet，可以知道在 LCDCON5 寄存器的 PWREN 位是控制 LCD 是否输出的，当为 0 时 LCD 不输出，此时 LCD\_PWREN 脚为低，就会关闭 LCD 的背光；当为 1 时 LCD 输出，此时 LCD\_PWREN 脚为高，开启 LCD 的背光。

修改内核源码的“drivers/video/s3c2410fb.c”文件的 603 行到 620 行的“s3c2410fb\_blank”函数，修改为如下所示：

```
static int s3c2410fb_blank(int blank_mode, struct fb_info *info)
{
    struct s3c2410fb_info *fbi = info->par;
```

```

/* void __iomem *tpal_reg = fbi->io;

dprintk("blank(mode=%d, info=%p)\n", blank_mode, info);

tpal_reg += is_s3c2412(fbi) ? S3C2412_TPAL : S3C2410_TPAL;

if (blank_mode == FB_BLANK_UNBLANK)
    writel(0x0, tpal_reg);
else {
    dprintk("setting TPAL to output 0x000000\n");
    writel(S3C2410_TPAL_EN, tpal_reg);
}
*/

void __iomem *regs = fbi->io;
u_long flags;
local_irq_save(flags);

fbi->regs.lcdcon5 = __raw_readl(regs + S3C2410_LCDCON5);
switch( blank_mode )
{
    case 0:
        fbi->regs.lcdcon5 &= ~S3C2410_LCDCON5_PWREN;
        printk(KERN_INFO "Turn off The LCD Backlight\n");
        break;
    case 1:
        fbi->regs.lcdcon5 |= S3C2410_LCDCON5_PWREN;
        printk(KERN_INFO "Turn on The LCD Backlight\n");
        break;
    default:
        break;
}

__raw_writel(fbi->regs.lcdcon5, regs + S3C2410_LCDCON5);

local_irq_restore(flags);

return 0;
}

```

修改完以上的内容，重新编译出内核镜像，然后烧写到开发板中。

## 15.7 编写 LCD 背光控制程序

下面列出 LCD 背光驱动的控制程序的源码，内容如下：

```

/*****

```

NAME:backlight.c

COPYRIGHT:www.embedsky.net

```
*****/
```

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include <unistd.h>
```

```
#include <sys/ioctl.h>
```

```
#include <fcntl.h>
```

```
#include <linux/fb.h>
```

```
int main(int argc, char **argv)
```

```
{
```

```
    int turn;
```

```
    int fd;
```

```
    if ( argc == 1 )
```

```
    {
```

```
        fprintf(stderr, "\nUsage: backlight on|off !\n\n");
```

```
        exit(1);
```

```
    }
```

```
    fd = open("/dev/fb0", O_RDWR);
```

```
    if (fd < 0) {
```

```
        perror("open LCD device !");
```

```
        exit(1);
```

```
    }
```

```
    if ( strcmp(argv[1], "on" ) == 0)
```

```
        turn = 1;
```

```
    else if ( strcmp(argv[1], "off" ) == 0)
```

```
        turn = 0;
```

```
    else
```

```
    {
```

```
        fprintf(stderr, "\nUsage: backlight on|off !\n\n");
```

```
        exit(1);
```

```
    }
```

```
    ioctl(fd, FBIOLANK, turn);
```

```
    close(fd);
```

```
    return 0;
```

```
}
```

下面是对应的 Makefile 文件的内容:

```
CROSS=arm-linux-
```

```
all: backlight
```

```
backlight:backlight.c
```

```
$(CROSS)gcc -o backlight backlight.c
```

```
$(CROSS)strip backlight
```

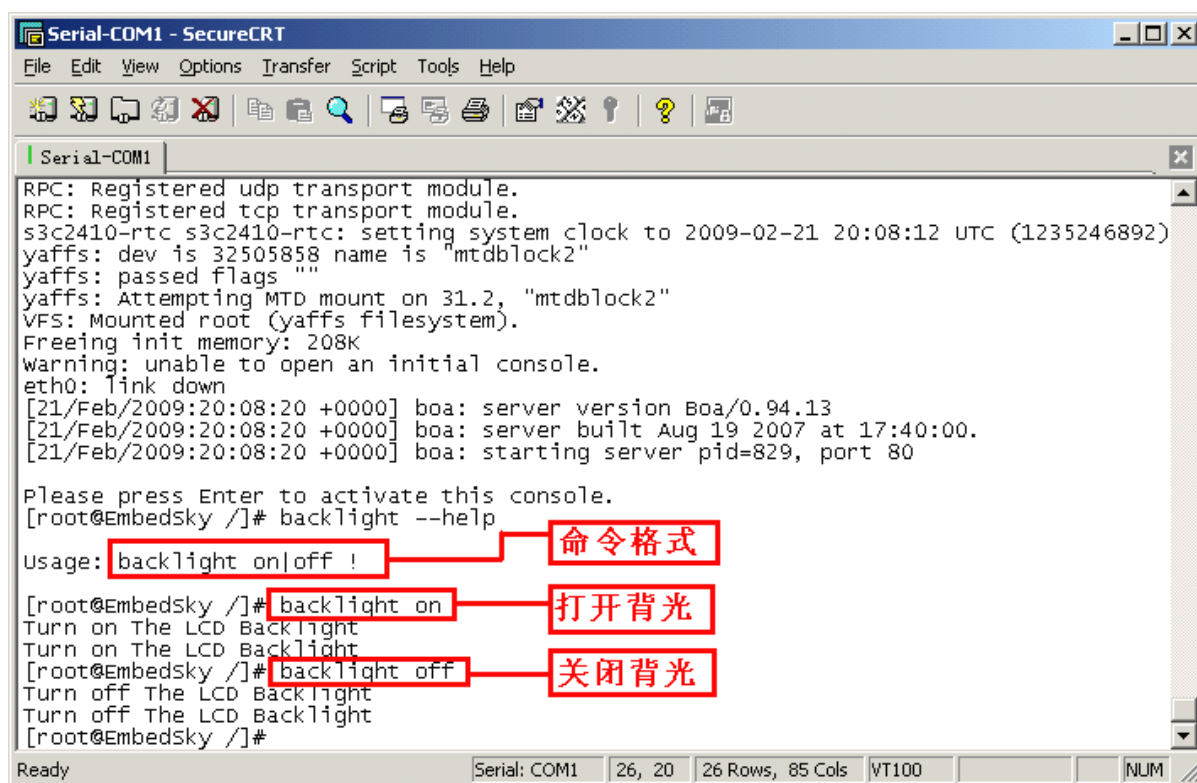
```
clean:
```

```
@rm -vf backlight *.o *~
```

## 15.8 测试 LCD 背光开关

将编译出来的名为“**backlight**”的 LCD 背光控制程序复制到文件系统的“**/sbin/**”目录下,然后重新压制成 yaffs 镜像,然后烧写到开发板中,启动开发板后,在串口终端输入: **\$backlight on** 打开背光, **\$backlight off** 关闭背光。

下图就是控制背光的操作截图:



```
Serial-COM1 - SecureCRT
File Edit View Options Transfer Script Tools Help

Serial-COM1
RPC: Registered udp transport module.
RPC: Registered tcp transport module.
s3c2410-rtc s3c2410-rtc: setting system clock to 2009-02-21 20:08:12 UTC (1235246892)
yaffs: dev is 32505858 name is "mtdblock2"
yaffs: passed flags ""
yaffs: Attempting MTD mount on 31.2, "mtdblock2"
VFS: Mounted root (yaffs filesystem).
Freeing init memory: 208K
warning: unable to open an initial console.
eth0: link down
[21/Feb/2009:20:08:20 +0000] boa: server version Boa/0.94.13
[21/Feb/2009:20:08:20 +0000] boa: server built Aug 19 2007 at 17:40:00.
[21/Feb/2009:20:08:20 +0000] boa: starting server pid=829, port 80

Please press Enter to activate this console.
[root@Embedsky /]# backlight --help
usage: backlight on|off !
[root@Embedsky /]# backlight on
Turn on The LCD Backlight
Turn on The LCD Backlight
[root@Embedsky /]# backlight off
Turn off The LCD Backlight
Turn off The LCD Backlight
[root@Embedsky /]#
```

命令格式

打开背光

关闭背光

## Step 16、触摸驱动移植

以上完成了 LCD 驱动的移植，下面我们将进行触摸屏驱动的移植。

### 16.1 添加触摸驱动补丁

在 2.6.25.8 的内核里面是没有针对 S3C2440 的触摸驱动的，这里我们对其打上触摸的驱动补丁，首先复制我们准备好的触摸驱动源码（名为：`EmbedSky_ts.c`）到内核源码的“`drivers/input/touchscreen/`”目录下和触摸驱动的头文件（名为：`ts.h`）到内核源码的“`include/asm-arm/arch-s3c2410/`”目录下；

然后修改“`drivers/input/touchscreen/`”目录下面的“`Kconfig`”和“`Makefile`”文件，以支持对刚刚添加的文件的支持。

修改内核源码的“`drivers/input/touchscreen/`”目录下面的“`Kconfig`”文件，在文件的最后添加如下内容（红色部分所示）：

```
config TOUCHSCREEN_USB_GOTOP
    default y
    bool "GoTop Super_Q2/GogoPen/PenPower tablet device support" if EMBEDDED
    depends on TOUCHSCREEN_USB_COMPOSITE
```

```
config EmbedSky_TOUCHSCREEN
    tristate "EmbedSky touchscreen"
    depends on ARCH_S3C2410 && INPUT && INPUT_TOUCHSCREEN
    select SERIO
    help
        To compile this driver as a module, choose M here: the
        module will be called EmbedSky_ts.ko.
```

```
config TOUCHSCREEN_EmbedSky_DEBUG
    boolean "EmbedSky touchscreen debug messages"
    depends on EmbedSky_TOUCHSCREEN
    help
        Select this if you want debug messages
```

```
Endif
```

修改同目录下的“`Makefile`”文件，在文件的最后添加如下内容（红色部分所示）：

```
#
# Makefile for the touchscreen drivers.
#
# Each configuration option enables a list of files.
```



```

obj-$(CONFIG_TOUCHSCREEN_ADS7846) += ads7846.o
obj-$(CONFIG_TOUCHSCREEN_BITSY) += h3600_ts_input.o
obj-$(CONFIG_TOUCHSCREEN_CORGI) += corgi_ts.o
obj-$(CONFIG_TOUCHSCREEN_GUNZE) += gunze.o
obj-$(CONFIG_TOUCHSCREEN_ELO) += elo.o
obj-$(CONFIG_TOUCHSCREEN_FUJITSU) += fujitsu_ts.o
obj-$(CONFIG_TOUCHSCREEN_MTOUCH) += mtouch.o
obj-$(CONFIG_TOUCHSCREEN_MK712) += mk712.o
obj-$(CONFIG_TOUCHSCREEN_HP600) += hp680_ts_input.o
obj-$(CONFIG_TOUCHSCREEN_HP7XX) += jornada720_ts.o
obj-$(CONFIG_TOUCHSCREEN_USB_COMPOSITE) += usbtouchscreen.o
obj-$(CONFIG_TOUCHSCREEN_PENMOUNT) += penmount.o
obj-$(CONFIG_TOUCHSCREEN_TOUCHRIGHT) += touchright.o
obj-$(CONFIG_TOUCHSCREEN_TOUCHWIN) += touchwin.o
obj-$(CONFIG_TOUCHSCREEN_UCB1400) += ucb1400_ts.o
obj-$(CONFIG_EmbedSky_TOUCHSCREEN) += EmbedSky_ts.o

```

打好了触摸驱动的补丁后，还需要添加触摸屏设备和配置信息。  
 修改内核源码的“arch/arm/mach-s3c2440/mach-smdk2440.c”文件，  
 在 41 行添加上对触摸驱动头文件的调用，内容如下：

```

#include <asm/plat-s3c/regs-serial.h>
#include <asm/arch/regs-gpio.h>
#include <asm/arch/regs-lcd.h>

```

```

#include <asm/arch/idle.h>
#include <asm/arch/fb.h>
#include <asm/arch/ts.h>

```

```

#include <asm/plat-s3c24xx/s3c2410.h>
#include <asm/plat-s3c24xx/s3c2440.h>
#include <asm/plat-s3c24xx/clock.h>

```

在 259 行开始添加内容如下：（红色部分所示）

```

//touch screen
struct platform_device s3c_device_ts = {
    .name      = "EmbedSky-ts",
    .id        = -1,
};

static struct EmbedSky_ts_mach_info EmbedSky_ts_info = {
    .delay = 10000,
    .presc = 49,
    .oversampling_shift = 2,
};

```

```
static struct platform_device *smdk2440_devices[] __initdata = {
    &s3c_device_usb,
    &s3c_device_lcd,
    &s3c_device_wdt,
    &s3c_device_i2c,
    &s3c_device_iis,
    &s3c_device_usb gadget,
    &s3c_device_ts,
};

static void __init smdk2440_map_io(void)
{
    s3c24xx_init_io(smdk2440_iodesc, ARRAY_SIZE(smdk2440_iodesc));
    s3c24xx_init_clocks(12000000);
    s3c24xx_init_uarts(smdk2440_uartcfgs, ARRAY_SIZE(smdk2440_uartcfgs));
}

static void __init smdk2440_machine_init(void)
{
    s3c24xx_fb_set_platdata(&smdk2440_fb_info);
    s3c_device_ts.dev.platform_data = &EmbedSky_ts_info;

    platform_add_devices(smdk2440_devices, ARRAY_SIZE(smdk2440_devices));
    smdk_machine_init();
    s3c2410_gpio_setpin(S3C2410_GPG12, 0);
    s3c2410_gpio_cfgpin(S3C2410_GPG12, S3C2410_GPIO_OUTPUT);
    s3c24xx_udc_set_platdata(&EmbedSky_udc_cfg);
}
}
```

## 16.2 添加 TSDEV 的补丁

然后再打上 TSDEV 设备的补丁，复制补丁文件“tsdev.c”到内核源码的“drivers/input/”目录下，然后修改同目录下的“Kconfig”和“Makefile”文件。

修改内核源码“drivers/input/”目录下的“Kconfig”文件，在 117 行添加如下内容（红色部分所示）：

```
config INPUT_TSDEV
    tristate "Touchscreen interface"
    ---help---
    Say Y here if you have an application that only can understand the
    Compaq touchscreen protocol for absolute pointer data. This is
    useful namely for embedded configurations.
```

If unsure, say N.

To compile this driver as a module, choose M here: the module will be called tsdev.

```
config INPUT_TSDEV_SCREEN_X
    int "Horizontal screen resolution"
    depends on INPUT_TSDEV
    default "1024"
```

```
config INPUT_TSDEV_SCREEN_Y
    int "Vertical screen resolution"
    depends on INPUT_TSDEV
    default "768"
```

```
config INPUT_EVDEV
    tristate "Event interface"
    help
        Say Y here if you want your input device events be accessible
        under char device 13:64+ - /dev/input/eventX in a generic way.
```

To compile this driver as a module, choose M here: the module will be called evdev.

修改同目录下的“Makefile”文件，在 16 行添加如下内容（红色部分所示）：

```
#
# Makefile for the input core drivers.
#

# Each configuration option enables a list of files.

obj-$(CONFIG_INPUT) += input-core.o
input-core-objs := input.o ff-core.o

obj-$(CONFIG_INPUT_FF_MEMLESS) += ff-memless.o
obj-$(CONFIG_INPUT_POLLDEV) += input-polldev.o

obj-$(CONFIG_INPUT_MOUSEDEV) += mousedev.o
obj-$(CONFIG_INPUT_JOYDEV) += joydev.o
obj-$(CONFIG_INPUT_EVDEV) += evdev.o
obj-$(CONFIG_INPUT_TSDEV) += tsdev.o
obj-$(CONFIG_INPUT_EVBUG) += evbug.o

obj-$(CONFIG_INPUT_KEYBOARD) += keyboard/
```

```
obj-$(CONFIG_INPUT_MOUSE)+= mouse/
obj-$(CONFIG_INPUT_JOYSTICK) += joystick/
obj-$(CONFIG_INPUT_TABLET) += tablet/
obj-$(CONFIG_INPUT_TOUCHSCREEN) += touchscreen/
obj-$(CONFIG_INPUT_MISC) += misc/

obj-$(CONFIG_INPUT_APMPOWER) += apm-power.o
```

## 16.3 配置内核

修改完毕之后，输入：`#make menuconfig` 进入配置单，然后配置如下：

```
Device Drivers --->
  Input device support --->
    <*> Touchscreen interface
      (1024) Horizontal screen resolution
      (768) Vertical screen resolution
    <*> Event interface
      <> Event debugging
    [*] Touchscreens --->
      <*> EmbedSky touchscreen
        [ ] EmbedSky touchscreen debug messages
```

配置完毕后，保存配置，然后编译内核，烧写镜像到开发板中，触摸就能够使用。

## 16.4 文件系统里面添加 Qte

下面我们讲解制作带有 Qte 的 yaffs 文件系统，详细的方法可以参考 SKY2440 开发板的使用手册的 5.4.8 节或 TQ2440 开发板的使用手册的 6.4.8 节，也可以在 2.6.13 的文件系统里面直接复制：“/opt/” 目录和 “/root/” 目录到 2.6.25.8 的文件系统里面，替换掉以前的目录。

然后再在文件系统的 “/bin/” 目录下新建一个名为：“qtopia” 的可执行文件，内容如下：

```
#!/bin/sh

export set HOME=/root
export set QTDIR=/opt/qt
export set QPEDIR=/opt/qtopia
export set KDEDIR=/opt/kde
export set QWS_KEYBOARD="USB:/dev/input/event1"
export set QWS_MOUSE_PROTO="TPanel:/dev/touchscreen/0"
export set PATH=$QPEDIR/bin:$PATH
export set LD_LIBRARY_PATH=$QTDIR/lib:$QPEDIR/lib
$QPEDIR/bin/qpe > /dev/null 2>/dev/null
```

修改 “/etc/init.d/rcS” 文件，下面列出新添加的内容：（红色部分所示）

```
#!/bin/sh
```

```
PATH=/sbin:/bin:/usr/sbin:/usr/bin
```

```
runlevel=S
```

```
prevlevel=N
```

```
umask 022
```

```
export PATH runlevel prevlevel
```

```
#
```

```
# Trap CTRL-C &c only in this shell so we can interrupt subprocesses.
```

```
#
```

```
mount -a
```

```
mkdir /dev/pts
```

```
mount -t devpts devpts /dev/pts
```

```
echo /sbin/mdev > /proc/sys/kernel/hotplug
```

```
mdev -s
```

```
mkdir -p /var/lock
```

```
mkdir /dev/fb /dev/v4l
```

```
ln -s /dev/fb0 /dev/fb/0
```

```
ln -s /dev/video0 /dev/v4l/video0
```

```
ln -s /dev/ts0 /dev/h3600_tsraw #建立一个触摸设备的链接
```

```
ifconfig lo 127.0.0.1
```

```
ifconfig eth0 hw ether 10:23:45:67:89:ab
```

```
ifconfig eth0 192.168.1.6 up
```

```
route add default gw 192.168.1.2
```

```
/etc/rc.d/init.d/httpd start
```

```
/etc/rc.d/init.d/netd start
```

```
qtopia & #启动 Qte 的脚本
```

```
/bin/hostname -F /etc/sysconfig/HOSTNAME
```

然后重新制作出 yaffs 文件系统的镜像，烧写到开发板中，就可以使用触摸进行操作了。

## Step 17、声卡驱动移植

下面进行声卡驱动的移植，主要就是对 UDA1341 驱动的移植。

### 17.1 添加声卡驱动补丁

在 2.6.25.8 的内核里面是没有针对 S3C2440 的声卡驱动的，这里我们对其打上声卡的驱动补丁，首先复制我们准备好的声卡驱动源码（名为：**EmbedSky\_uda1341.c**）到内核源码的“**sound/oss/**”目录下；然后修改同目录下面的“**Kconfig**”和“**Makefile**”文件，以支持对刚刚添加的文件的支持。

修改内核源码的“**sound/oss/**”目录下的“**Kconfig**”文件，在 8 行开始添加如下内容（红色部分所示）：

```
# drivers/sound/Config.in
#
# 18 Apr 1998, Michael Elizabeth Chastain, <mailto:mec@shout.net>
# More hacking for modularisation.
#
# Prompt user for primary drivers.
```

```
config EmbedSky_SOUND
    tristate "EmbedSky UDA1341 driver"
    depends on ARCH_S3C2410
    default y
    help
        To compile this driver as a module, choose M here: the
        module will be called EmbedSky_uda1341.ko.
```

```
config SOUND_BCM_CS4297A
    tristate "Crystal Sound CS4297a (for Swarm)"
    depends on SOUND_PRIME && SIBYTE_SWARM
    help
        The BCM91250A has a Crystal CS4297a on synchronous serial
        port B (in addition to the DB-9 serial port). Say Y or M
        here to enable the sound chip instead of the UART. Also
        note that CONFIG_KGDB should not be enabled at the same
        time, since it also attempts to use this UART port.
```

修改同目录下的“**Makefile**”文件，在 34 行添加如下内容（红色部分所示）：

```
obj-$(CONFIG_SOUND_MSNDCLAS) += msnd.o msnd_classic.o
obj-$(CONFIG_SOUND_MSNDPIN) += msnd.o msnd_pinnacle.o
obj-$(CONFIG_SOUND_VWSND) += vwsnd.o
obj-$(CONFIG_SOUND_AU1550_AC97) += au1550_ac97.o ac97_codec.o
obj-$(CONFIG_SOUND_TRIDENT) += trident.o ac97_codec.o
```

```
obj-$(CONFIG_SOUND_BCM_CS4297A) += swarm_cs4297a.o
```

```
obj-$(CONFIG_EmbedSky_SOUND) += EmbedSky_uda1341.o
```

```
obj-$(CONFIG_SOUND_WM97XX) += ac97_plugin_wm97xx.o
```

```
obj-$(CONFIG_DMASOUND) += dmasound/
```

## 17.2 修改内核 DMA 支持

然后还需要修正内核对 DMA 的处理，修改内核源码的“arch/arm/plat-s3c24xx/dma.c”文件，在 829 行改为如下内容：

```
s3c2410_dma_ctrl(chan->number|DMACH_LOW_LEVEL, S3C2410_DMAOP_STOP);
```

在 918 行改为如下内容：

```
s3c2410_dma_ctrl(chan->number| DMACH_LOW_LEVEL, S3C2410_DMAOP_STOP);
```

## 17.3 配置内核

完成以上操作之后，输入：`#make menuconfig` 进入配置单，然后配置如下所示：

```
Device Drivers --->
```

```
Sound --->
```

```
<*> Sound card support
```

```
Open Sound System --->
```

```
<*> Open Sound System (DEPRECATED)
```

```
<*> EmbedSky UDA1341 driver
```

完成配置后，保存配置单，编译出内核，烧写镜像到开发板之后，可以在开发板上使用声卡设备了。

## 17.4 测试声卡

在开发板的串口终端使用：`#cat /dev/dsp > /tmp/abc.wav` 这条命令进行录音测试，然后再使用：`#cat /tmp/abc.wav > /dev/dsp`，就可以播放刚刚的录音。

如下图所示：

```
Serial-COM5 - SecureCRT
File Edit View Options Transfer Script Tools Help

Serial-COM5
usbcore: registered new interface driver hiddev
usbcore: registered new interface driver usbhid
drivers/hid/usbhid/hid-core.c: v2.6: USB HID core driver
UDA1341 audio driver initialized
TCP cubic registered
NET: Registered protocol family 1
RPC: Registered udp transport module.
RPC: Registered tcp transport module.
s3c2410-rtc s3c2410-rtc: hctosys: invalid date/time
yaffs: dev is 32505858 name is "mtdblock2"
yaffs: passed flags ""
yaffs: Attempting MTD mount on 31.2, "mtdblock2"
yaffs: block 3245 is marked bad
block 3246 is bad
VFS: Mounted root (yaffs filesystem).
Freeing init memory: 200K
eth0: link down

Please press Enter to activate this console.
[root@Embedsky /]# ls /dev/dsp
/dev/dsp
[root@Embedsky /]# cat /dev/dsp > /tmp/abc.wav
UDA1341:audio_set_dsp_speed:44100 prescaler:66
^C
[root@Embedsky /]# cat /tmp/abc.wav > /dev/dsp
UDA1341:audio_set_dsp_speed:44100 prescaler:66
[root@Embedsky /]#
```

## 17.5 添加 Madplay 到文件系统

我们也可以使用 madplay 播放器来播放 MP3 等音频文件，关于 madplay 的移植，在后续的章节讲解，这里只讲解添加到文件系统里面。

从 2.6.13 的文件系统的“/sbin/”目录下复制“madplay”到新的文件系统的“/sbin/”目录下；从“/lib/”目录复制“libz.so.1”库文件到新文件系统的“/lib/”目录下，因为 madplay 运行时会调用该库文件。

完成后，压制新的文件系统为 yaffs 镜像，然后烧写到开发板中，就可以使用 madplay 播放音频文件了：



```
Serial-COM1 - SecureCRT
File Edit View Options Transfer Script Tools Help

Serial-COM1
RPC: Registered udp transport module.
RPC: Registered tcp transport module.
s3c2410-rtc s3c2410-rtc: setting system clock to 2009-02-23 19:31:21 UTC (1235417481)
yaffs: dev is 32505858 name is "mtdblock2"
yaffs: passed flags ""
yaffs: Attempting MTD mount on 31.2, "mtdblock2"
VFS: Mounted root (yaffs filesystem).
Freeing init memory: 188K
eth0: link down
[23/Feb/2009:19:31:24 +0000] boa: server version Boa/0.94.13
[23/Feb/2009:19:31:24 +0000] boa: server built Aug 19 2007 at 17:40:00.
[23/Feb/2009:19:31:24 +0000] boa: starting server pid=277, port 80

Please press Enter to activate this console.
[root@EmbedSky /]# madplay /root/Documents/Test.mp3
MPEG Audio Decoder 0.15.0 (beta) - Copyright (C) 2000-2003 Robert Leslie et al.
UDA1341:audio_set_dsp_speed:44100 prescaler:66
  Title: EYES ON ME
  Artist: 王菲
  Album: 天籟村
  Year: 2000
  Genre: Pop
  Comment: http://tdk.126.com
UDA1341:audio_set_dsp_speed:44100 prescaler:66

Ready Serial: COM1 25, 1 25 Rows, 85 Cols VT100 NUM
```

## Step 18、RTC 驱动移植

下面对实时时钟进行移植。

### 18.1 添加平台对 RTC 的支持

在 2.6.25.8 上的实时时钟驱动是非常完善，我们只需要添加到 RTC 设备到设备初始化列表中就行了。修改内核源码“`arch/arm/mach-s3c2440/mach-smdk2440.c`”文件，在行添加如下内容：（红色部分所示）

```
static struct platform_device *smdk2440_devices[] __initdata = {
    &s3c_device_usb,
    &s3c_device_lcd,
    &s3c_device_wdt,
    &s3c_device_i2c,
    &s3c_device_iis,
    &s3c_device_usb gadget,
    &s3c_device_ts,
    &s3c_device_rtc,
};
```

### 18.2 配置内核

然后配置 RTC，输入：`#make menuconfig`

然后配置如下所示：

```
Device Drivers --->
  <*> Real Time Clock --->
    [*] Set system time from RTC on startup and resume
    (rtc0) RTC used to set the system time
    [*] /sys/class/rtc/rtcN (sysfs)
    [*] /proc/driver/rtc (procfs for rtc0)
    [*] /dev/rtcN (character devices)
    <*> Samsung S3C series SoC RTC
```

### 18.3 设置并测试 RTC

配置完毕后，编译内核烧写到开发板中，然后使用 busybox 自带的 `hwclock` 和 `date` 命令可以设置 RTC，

```
$date -s 月日時分年 //其中月、日、时和分都是两位数，年是四位数。
//比如：020416102009 就是 2009 年 2 月 4 日 16 点 10 分
$hwclock -w //保存刚刚设置的时钟。
```

下图就是设置前和设置后的情况：

```
Serial-COM5 - SecureCRT
File Edit View Options Transfer Script Tools Help

Serial-COM5
TCP cubic registered
NET: Registered protocol family 1
RPC: Registered udp transport module.
RPC: Registered tcp transport module.
s3c2410-rtc s3c2410-rtc: hctosys: invalid date/time
yaffs: dev is 32505858 name is "mtdblock2"
yaffs: passed flags ""
yaffs: Attempting MTD mount on 31.2, "mtdblock2"
yaffs: block 3245 is marked bad
block 3246 is bad
VFS: Mounted root (yaffs filesystem).
Freeing init memory: 200K
eth0: link down

Please press Enter to activate this console.
[root@Embedsky /]# date
Thu Jan 1 00:00:30 UTC 1970
[root@Embedsky /]# date -s 020416102009
wed Feb 4 16:10:00 UTC 2009
[root@Embedsky /]# hwclock -w
[root@Embedsky /]# date
wed Feb 4 16:10:04 UTC 2009
[root@Embedsky /]#
```

设置好之后关闭电源，实时时钟依然会工作。为了让系统启动时能够同步设置好的 RTC，然后我们需要在文件系统的“`/etc/init.d/rcS`”文件中添加一条 RTC 同步的命令，添加后，重新制作 yaffs 文件系统，烧写到开发板里面，启动后，系统就会自动同步 RTC 了。

下面是新的 rcS 文件的内容：

```
#!/bin/sh
```

```
PATH=/sbin:/bin:/usr/sbin:/usr/bin
```

```
runlevel=S
```

```
prevlevel=N
```

```
umask 022
```

```
export PATH runlevel prevlevel
```

```
#
```

```
# Trap CTRL-C &c only in this shell so we can interrupt subprocesses.
```

```
#
```

```
mount -a
```

```
mkdir /dev/pts
```

```
mount -t devpts devpts /dev/pts
```

```
echo /sbin/mdev > /proc/sys/kernel/hotplug
```

```
mdev -s
```

```
mkdir -p /var/lock
```

```
mkdir /dev/fb /dev/v4l
```

```
ln -s /dev/fb0 /dev/fb/0
```

```
ln -s /dev/video0 /dev/v4l/video0
```

```
ln -s /dev/ts0 /dev/h3600_tsraw
```

```
hwclock -s #系统同步 RTC
```

```
ifconfig lo 127.0.0.1
```

```
ifconfig eth0 hw ether 10:23:45:67:89:ab
```

```
ifconfig eth0 192.168.1.6 up
```

```
route add default gw 192.168.1.2
```

```
/etc/rc.d/init.d/httpd start
```

```
/etc/rc.d/init.d/netd start
```

```
qtopia &
```

```
/bin/hostname -F /etc/sysconfig/HOSTNAME
```

这里就完成了对 RTC 部分的操作。

## Step 19、看门狗驱动移植

下面对看门狗进行移植。

### 19.1 启动看门狗

在 2.6.25.8 的内核中对看门狗的驱动已经添加了的，不过在驱动程序还需要进行简单的修改。

修改内核源码的“`drivers/watchdog/s3c2410_wdt.c`”文件，修改 62 和 63 行，内容如下：（红色部分所示）

```
#undef S3C_VA_WATCHDOG
```

```
#define S3C_VA_WATCHDOG (0)
```

```
#include <asm/plat-s3c/regs-watchdog.h>
```

```
#define PFX "s3c2410-wdt: "
```

```
#define CONFIG_S3C2410_WATCHDOG_ATBOOT (1) //启动看门狗
```

```
#define CONFIG_S3C2410_WATCHDOG_DEFAULT_TIME (30) //设置时间
```

```
static int nowayout = WATCHDOG_NOWAYOUT;
```

```
static int tmr_margin = CONFIG_S3C2410_WATCHDOG_DEFAULT_TIME;
```

```
static int tmr_atboot = CONFIG_S3C2410_WATCHDOG_ATBOOT;
```

```
static int soft_noboot = 0;
```

```
static int debug = 0;
```

### 19.2 配置内核

修改完毕后，输入：`#make menuconfig`，然后进行配置，如下所示：

```
Device Drivers --->
```

```
[*] Watchdog Timer Support --->
```

```
<*> S3C2410 Watchdog
```

### 19.3 没有喂狗的情况

配置完毕后，编译出镜像，烧写到开发板中，启动开发板，您会发现过一段时间，系统就会自动重启，这是因为看门狗已经开始工作了。

### 19.4 喂狗程序的编写

为了避免老是自动重启，我们需要编写一个看门狗的应用程序，并且在开机时自动启动该程序。

下面列出了我们提供的名为：“EmbedSky\_wdg.c”的看门狗应用程序的源码，内容如下：

```
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <linux/watchdog.h>

int main(int argc, char **argv)
{
    int fd = 0;
    int n = 0;
    fd = open("/dev/watchdog", O_RDONLY);

    if(fd < 0) {
        perror("/dev/watchdog");
        return -1;
    }
    for(;;) {
        ioctl(fd, WDIOC_KEEPALIVE);    //让看门狗一直有效
        sleep(3);
    }
    close(fd);
    return 0;
}
```

然后这里列出了对应的“Makefile”文件的内容：

```
CC=/opt/EmbedSky/crosstools_3.4.1_softfloat/arm-linux/gcc-3.4.1-glibc-2.3.3/bin/arm-linux-gcc
LD=/opt/EmbedSky/crosstools_3.4.1_softfloat/arm-linux/gcc-3.4.1-glibc-2.3.3/bin/arm-linux-ld
```

```
EXEC=EmbedSky_wdg
OBJS=EmbedSky_wdg.o
```

```
CFLAGS +=
LDFLAGS +=
```

```
all:$(EXEC)
```

```
$(EXEC):$(OBJS)
    $(CC) $(LDFLAGS) -o $@ $(OBJS) $(LDLIBS$(LDLIBS_.$@))
```

```
clean:
    -rm -f $(EXEC) *.elf *.gdb *.o
```

## 19.5 文件系统里面实现喂狗操作

编译出该应用程序后，把它复制到文件系统的“/sbin/”目录下面，然后修改“/etc/init.d/rcS”文件，下面列出修改后的内容：

```
#!/bin/sh
```

```
PATH=/sbin:/bin:/usr/sbin:/usr/bin
```

```
runlevel=S
```

```
prevlevel=N
```

```
umask 022
```

```
export PATH runlevel prevlevel
```

```
#
```

```
# Trap CTRL-C &c only in this shell so we can interrupt subprocesses.
```

```
#
```

```
mount -a
```

```
mkdir /dev/pts
```

```
mount -t devpts devpts /dev/pts
```

```
echo /sbin/mdev > /proc/sys/kernel/hotplug
```

```
mdev -s
```

```
mkdir -p /var/lock
```

```
mkdir /dev/fb /dev/v4l
```

```
ln -s /dev/fb0 /dev/fb/0
```

```
ln -s /dev/video0 /dev/v4l/video0
```

```
ln -s /dev/ts0 /dev/h3600_tsraw
```

```
hwclock -s
```

```
EmbedSky_wdg & #启动看门狗喂狗程序
```

```
ifconfig lo 127.0.0.1
```

```
ifconfig eth0 hw ether 10:23:45:67:89:ab
```

```
ifconfig eth0 192.168.1.6 up
```

```
route add default gw 192.168.1.2
```

```
/etc/rc.d/init.d/httpd start
```

```
/etc/rc.d/init.d/netd start
```

```
qtopia &
```

```
/bin/hostname -F /etc/sysconfig/HOSTNAME
```

制作好文件系统后，烧写到开发板中，再次启动系统，就不会出现刚刚的不断重启的情况了。



## Step 20、SD 卡驱动移植

下面对 SD 卡驱动进行移植。

### 20.1 添加 SD 卡的驱动

在 2.6.25.8 的内核里面是没有对 S3C2440 的 SD 卡进行支持的，我们需要对 SD 卡打驱动补丁，SD 卡的补丁我们可以从 [http://svn.openmoko.org/branches/src/target/kernel/2.6.24.x/patches/s3c\\_mci.patch](http://svn.openmoko.org/branches/src/target/kernel/2.6.24.x/patches/s3c_mci.patch) 下载，然后复制它到内核源码中，然后打上该补丁，在 PC 上使用命令：`#patch -p1 < s3c_mci.patch`，打补丁时会在内核源码中添加如下文件，这里列出了对应的相对路径：

“include/asm-arm/arch-s3c2410/regs-sdi.h” 文件

“include/asm-arm/arch-s3c2410/mci.h” 文件

“drivers/mmc/host/s3cmci.c” 文件

“drivers/mmc/host/s3cmci.h” 文件

同时修改了：

“drivers/mmc/host/Kconfig” 文件

“drivers/mmc/host/Makefile” 文件

“arch/arm/mach-s3c2412/s3c2412.c” 文件

“arch/arm/mach-s3c2440/s3c2440.c” 文件

“arch/arm/mach-s3c2442/s3c2442.c” 文件。

### 20.2 添加 SD 设备到设备列表

打完补丁后，还需要添加到 SD 设备到设备初始化列表中，修改内核源码的 “arch/arm/mach-s3c2440/mach-smdk2440.c” 文件，在 280 行添加如下内容：（红色部分所示）

```
static struct platform_device *smdk2440_devices[] __initdata = {
    &s3c_device_usb,
    &s3c_device_lcd,
    &s3c_device_wdt,
    &s3c_device_i2c,
    &s3c_device_iis,
    &s3c_device_usb gadget,
    &s3c_device_ts,
    &s3c_device_rtc,
    &s3c_device_sdi,
};
```

## 20.3 去除 SD 驱动 bug

然后修改“`drivers/mmc/host/s3cmci.c`”文件，屏蔽掉 1246 和 1247 行，内容如下：（红色部分所示）

```
if (s3c2410_dma_request(host->dma, &s3cmci_dma_client, NULL)) {  
    dev_err(&pdev->dev, "unable to get DMA channel.\n");  
    //ret = -EBUSY;  
    //goto probe_free_irq_cd;  
}
```

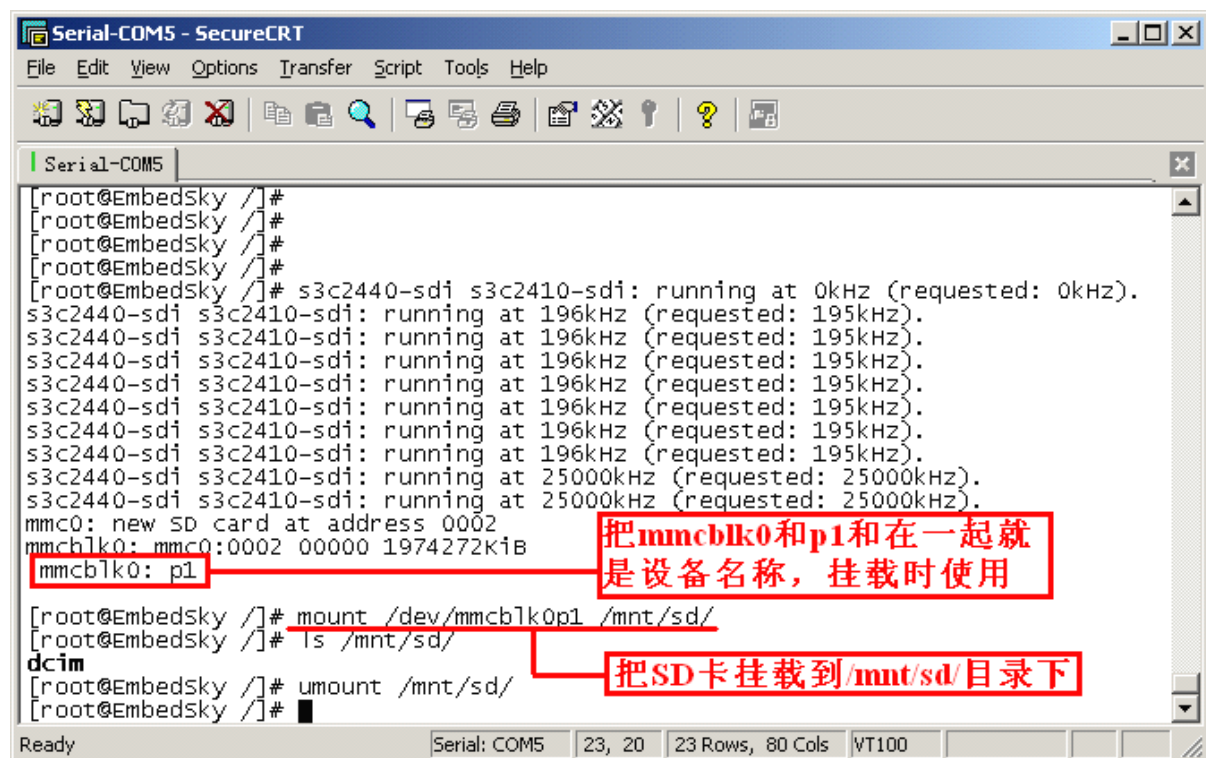
## 20.4 配置内核

完成之后，输入：`#make menuconfig`，进行配置，如下所示：

```
Device Drivers ---->  
  <*> MMC/SD card support --->  
    <*> MMC block device driver  
    [*] Use bounce buffer for simple hosts  
    <*> Samsung S3C24xx SD/MMC Card Interface support
```

## 20.5 挂载 SD 卡

然后保存配置，编译出镜像，烧写到开发板中，启动系统后，插入 SD 卡，下图就是挂载情况：



```
Serial-COM5 - SecureCRT  
File Edit View Options Transfer Script Tools Help  
Serial-COM5  
[root@Embedsky /]#  
[root@Embedsky /]#  
[root@Embedsky /]#  
[root@Embedsky /]# s3c2440-sdi s3c2410-sdi: running at 0kHz (requested: 0kHz).  
s3c2440-sdi s3c2410-sdi: running at 196kHz (requested: 195kHz).  
s3c2440-sdi s3c2410-sdi: running at 196kHz (requested: 195kHz).  
s3c2440-sdi s3c2410-sdi: running at 196kHz (requested: 195kHz).  
s3c2440-sdi s3c2410-sdi: running at 196kHz (requested: 195kHz).  
s3c2440-sdi s3c2410-sdi: running at 196kHz (requested: 195kHz).  
s3c2440-sdi s3c2410-sdi: running at 196kHz (requested: 195kHz).  
s3c2440-sdi s3c2410-sdi: running at 196kHz (requested: 195kHz).  
s3c2440-sdi s3c2410-sdi: running at 25000kHz (requested: 25000kHz).  
s3c2440-sdi s3c2410-sdi: running at 25000kHz (requested: 25000kHz).  
mmc0: new SD card at address 0002  
mmcblk0: mmc0:0002 00000 1974272KiB  
mmcblk0: p1  
[root@Embedsky /]# mount /dev/mmcblk0p1 /mnt/sd/  
[root@Embedsky /]# ls /mnt/sd/  
dcim  
[root@Embedsky /]# umount /mnt/sd/  
[root@Embedsky /]#  
Ready Serial: COM5 23, 20 23 Rows, 80 Cols VT100
```

把mmcblk0和p1和在一起就是设备名称，挂载时使用

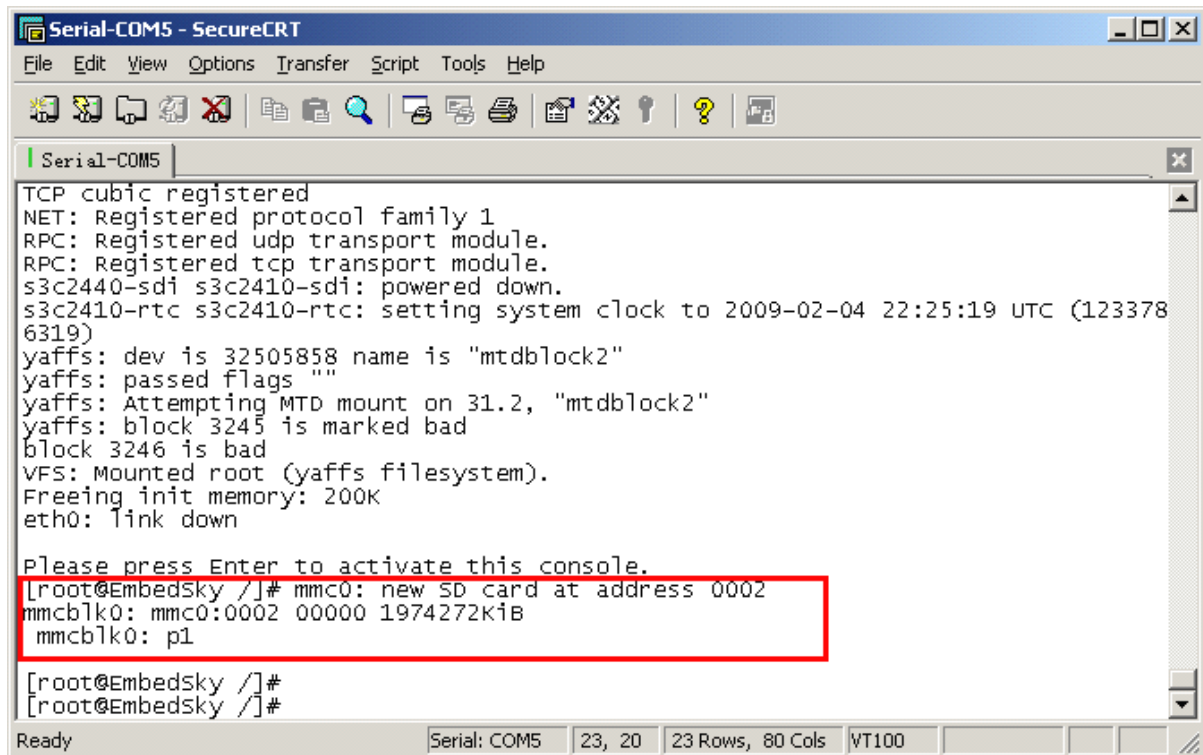
把SD卡挂载到/mnt/sd/目录下

## 20.6 去除多余信息

上图中出现了很多频率的打印信息，感觉太多了，咱们可以通过修改内核源码的“drivers/mmc/host/s3cmci.c”文件来实现屏蔽这些打印信息，修改该文件的 1090 和 1091 两行，屏蔽掉即可，内容如下：

```
//      dbg(host, dbg_conf, "running at %lukHz (requested: %ukHz).\n",  
//      host->real_rate/1000, ios->clock/1000);
```

下图是屏蔽之后，插入 SD 卡时的情况：



```
Serial-COM5 - SecureCRT
File Edit View Options Transfer Script Tools Help
Serial-COM5
TCP cubic registered
NET: Registered protocol family 1
RPC: Registered udp transport module.
RPC: Registered tcp transport module.
s3c2440-sdi s3c2410-sdi: powered down.
s3c2410-rtc s3c2410-rtc: setting system clock to 2009-02-04 22:25:19 UTC (1233786319)
yaffs: dev is 32505858 name is "mtdblock2"
yaffs: passed flags ""
yaffs: Attempting MTD mount on 31.2, "mtdblock2"
yaffs: block 3245 is marked bad
block 3246 is bad
VFS: Mounted root (yaffs filesystem).
Freeing init memory: 200k
eth0: link down

Please press Enter to activate this console.
[root@Embedsky /]# mmc0: new SD card at address 0002
mmcblk0: mmc0:0002 00000 1974272KiB
mmcblk0: p1

[root@Embedsky /]#
[root@Embedsky /]#

Ready Serial: COM5 23, 20 23 Rows, 80 Cols VT100
```

# DIY 驱动到系统中

## Step 21、编写第一个驱动程序

### 21.1 编写第一个驱动

下面编写 Hello 驱动，

在内核源码的“drivers/char/”目录下新建一个名为“EmbedSky\_hello.c”的文件，内容如下：

```
/******
```

```
NAME:EmbedSky_hello.c
```

```
COPYRIGHT:www.embedsky.net
```

```
*****/
```

```
#include <linux/module.h>
```

```
#include <linux/kernel.h>
```

```
#include <linux/fs.h>
```

```
#include <linux/init.h>
```

```
#include <linux/miscdevice.h>
```

```
#include <linux/delay.h>
```

```
#include <asm/irq.h>
```

```
#include <asm/arch/regs-gpio.h>
```

```
#include <asm/hardware.h>
```

```
MODULE_LICENSE("GPL");
```

```
static int __init EmbedSky_hello_init(void)
```

```
{
```

```
    printk("<1>\n    Hello,EmbedSky!\n");
```

```
    printk("<1>\nThis is first driver program.\n\n");
```

```
    return 0;
```

```
}
```

```
static void __exit EmbedSky_hello_exit(void)
```

```
{
```

```
    printk("<1>\n    Exit!\n");
```

```
    printk("<1>\nGoodbye EmbedSky!\n\n");
```

```
}
```

```
module_init(EmbedSky_hello_init);  
module_exit(EmbedSky_hello_exit);
```

## 21.2 在内核源码中添加对 **hello** 驱动的支持

修改同目录下的“Kconfig”文件，在 7 行添加如下内容：（红色部分所示）

```
#  
# Character device configuration  
#  
  
menu "Character devices"  
  
config EmbedSky_HELLO  
    tristate "TQ2440/SKY2440 Hello Driver"  
    depends on ARCH_S3C2440  
    help  
        EmbedSky TQ2440/SKY2440 Hello.  
  
config VT  
    bool "Virtual terminal" if EMBEDDED  
    depends on !S390  
    select INPUT  
    default y if !VIOCONS
```

修改同目录下的“Makefile”文件，在 12 行添加如下内容：（红色部分所示）

```
#  
# Makefile for the kernel character device drivers.  
#  
  
#  
# This file contains the font map for the default (hardware) font  
#  
FONTMAPFILE = cp437.uni  
  
obj-y      += mem.o random.o tty_io.o n_tty.o tty_ioctl.o  
  
obj-$(CONFIG_EmbedSky_HELLO) += EmbedSky_hello.o  
obj-$(CONFIG_LEGACY_PTYS)    += pty.o  
obj-$(CONFIG_UNIX98_PTYS)   += pty.o  
obj-y      += misc.o  
obj-$(CONFIG_VT)            += vt_ioctl.o vc_screen.o consolemap.o \  
                             consolemap_deftbl.o selection.o keyboard.o
```

## 21.3 配置内核

然后输入：`#make menuconfig`，然后配置如下：

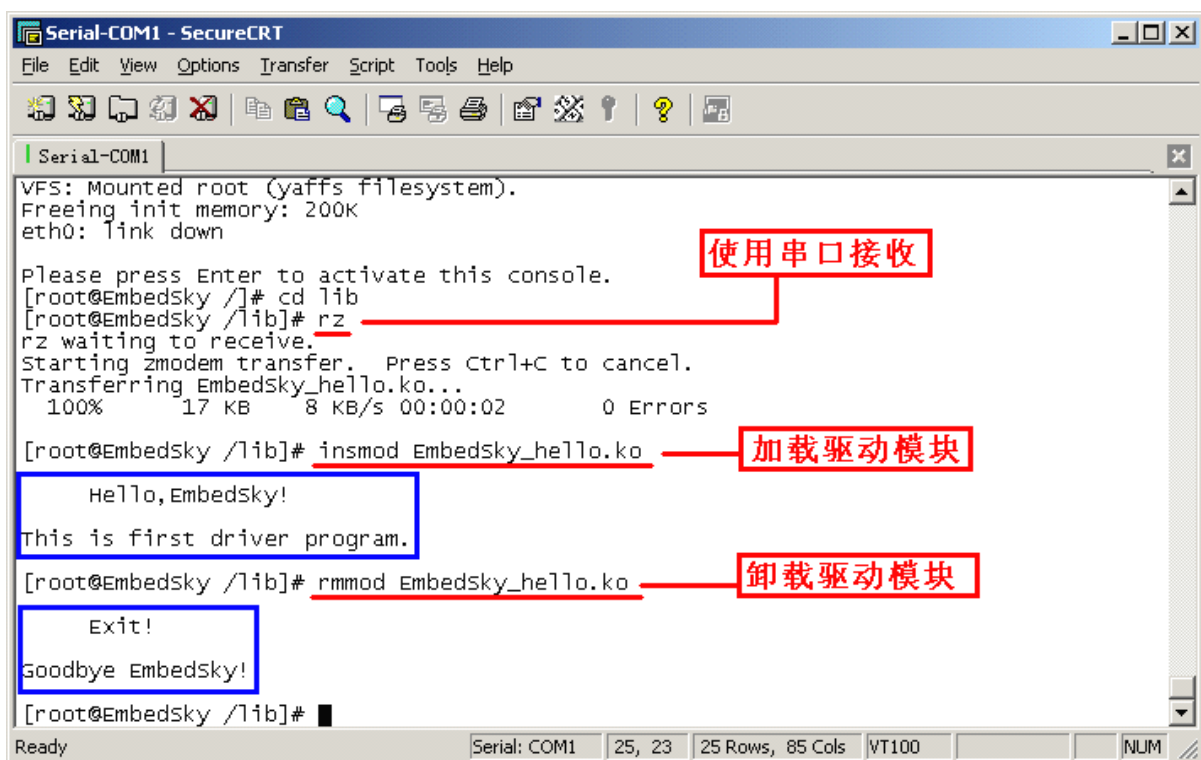
Device Drivers --->

Character devices --->

<M> TQ2440/SKY2440 Hello Driver

将其选择为“M”（模块），然后保存配置，编译出内核镜像烧写到开发板中。

然后再使用命令`#make SUBDIR=drivers/char/ modules`，然后编译出驱动模块，在内核目录下面的“`drivers/char/`”目录下面，名为：`EmbedSky_hello.ko`，将其复制到开发板中（可以使用U盘中转；也可以使用NFS，直接将其放到NFS里面；也可以使用FTP；还可以使用串口，方法请参考我们开发板的相应操作介绍），然后加载该驱动模块和卸载该驱动模块，截图如下：



```
Serial-COM1 - SecureCRT
File Edit View Options Transfer Script Tools Help

Serial-COM1
VFS: Mounted root (yaffs filesystem).
Freeing init memory: 200K
eth0: link down

Please press Enter to activate this console.
[root@Embedsky /]# cd /lib
[root@Embedsky /lib]# rz
rz waiting to receive.
Starting zmodem transfer. Press Ctrl+C to cancel.
Transferring Embedsky_hello.ko...
100% 17 KB 8 KB/s 00:00:02 0 Errors

[root@Embedsky /lib]# insmod Embedsky_hello.ko
Hello, EmbedSky!
This is first driver program.

[root@Embedsky /lib]# rmmod Embedsky_hello.ko
Exit!
Goodbye EmbedSky!

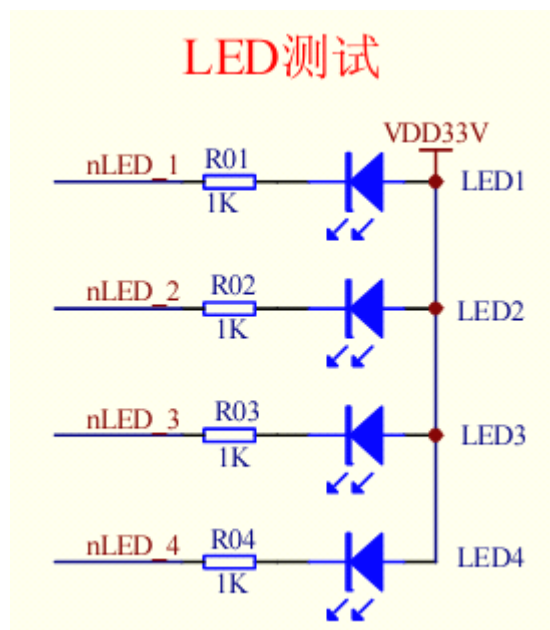
[root@Embedsky /lib]#
```

把上面生成的“`EmbedSky_hello.ko`”复制到文件系统的“`/lib/`”目录下面然后重新制作新的文件系统的镜像。

## Step 22、GPIO 口控制驱动的编写

### 22.1 硬件分析

在我们用的开发板的 4 个 LED 灯（TQ2440）或 3 个 LED 灯（SKY2440）分别使用了 S3C2440 芯片的：GPB5、GPB6、GPB7 和 GPB8（仅 TQ2440 使用），下面列出来对应的原理图：



根据上图我们可以知道，当 CPU 的 GPB5 到 8 是低电平时，LED 灯亮；当为高电平时 LED 灯灭。我们只需要在驱动中实现对 GPB 口电平的控制就可以实现对灯进行开关操作。

### 22.2 去除内核自带的 LED 灯驱动

在 2.6.25.8 的内核里面已经存在了一个 LED 灯的驱动程序，源码在“`drivers/leds/leds-s3c24xx.c`”，在“`arch/arm/plat-s3c24xx/common-smdk.c`”文件中有该驱动的注册信息等，这里我们重新写一个 LED 灯的驱动，我们将其屏蔽，处理办法，修改“`arch/arm/plat-s3c24xx/common-smdk.c`”文件，屏蔽掉 LED 部分的信息，删掉或屏蔽掉从 50 行到 108 行的内容，修改内容如下：（红色部分所示）

```
/* LED devices */
#ifdef CONFIG_LEDS_S3C24XX //这里的#ifdef 用于屏蔽
static struct s3c24xx_led_platdata smdk_pdata_led4 = {
    .gpio      = S3C2410_GPF4,
    .flags     = S3C24XX_LEDF_ACTLOW | S3C24XX_LEDF_TRISTATE,
    .name      = "led4",
    .def_trigger = "timer",
};

static struct s3c24xx_led_platdata smdk_pdata_led5 = {
```

```

        .gpio      = S3C2410_GPF5,
        .flags     = S3C24XX_LEDF_ACTLOW | S3C24XX_LEDF_TRISTATE,
        .name      = "led5",
        .def_trigger = "nand-disk",
    };

```

```

static struct s3c24xx_led_platdata smdk_pdata_led6 = {
    .gpio      = S3C2410_GPF6,
    .flags     = S3C24XX_LEDF_ACTLOW | S3C24XX_LEDF_TRISTATE,
    .name      = "led6",
};

```

```

static struct s3c24xx_led_platdata smdk_pdata_led7 = {
    .gpio      = S3C2410_GPF7,
    .flags     = S3C24XX_LEDF_ACTLOW | S3C24XX_LEDF_TRISTATE,
    .name      = "led7",
};

```

```

static struct platform_device smdk_led4 = {
    .name      = "s3c24xx_led",
    .id       = 0,
    .dev       = {
        .platform_data = &smdk_pdata_led4,
    },
};

```

```

static struct platform_device smdk_led5 = {
    .name      = "s3c24xx_led",
    .id       = 1,
    .dev       = {
        .platform_data = &smdk_pdata_led5,
    },
};

```

```

static struct platform_device smdk_led6 = {
    .name      = "s3c24xx_led",
    .id       = 2,
    .dev       = {
        .platform_data = &smdk_pdata_led6,
    },
};

```

```

static struct platform_device smdk_led7 = {
    .name      = "s3c24xx_led",

```



```

        .id      = 3,
        .dev      = {
            .platform_data = &smdk_pdata_led7,
        },
    };
#endif //这里的#endif 用于屏蔽

```

屏蔽的方法使用 `#if 0` 和 `#endif`。

然后屏蔽 196 到 199 行，内容如下：（红色部分所示）

```

static struct platform_device __initdata *smdk_devs[] = {
    &s3c_device_nand,
    // &smdk_led4,
    // &smdk_led5,
    // &smdk_led6,
    // &smdk_led7,
#ifdef CONFIG_DM9000 || defined(CONFIG_DM9000_MODULE)
    &s3c_device_dm9k,
#endif
};

```

然后再修改 209 行到 220 行，内容如下：（红色部分所示）

```

void __init smdk_machine_init(void)
{
    /* Configure the LEDs (even if we have no LED support)*/
    //设置 GPB5 到 GPB8 为输出口
    s3c2410_gpio_cfgpin(S3C2410_GPB5, S3C2410_GPB5_OUTP);
    s3c2410_gpio_cfgpin(S3C2410_GPB6, S3C2410_GPB6_OUTP);
    s3c2410_gpio_cfgpin(S3C2410_GPB7, S3C2410_GPB7_OUTP);
    s3c2410_gpio_cfgpin(S3C2410_GPB8, S3C2410_GPB8_OUTP);
    //使 LED1 和 3 亮，LED2 和 4 灭
    s3c2410_gpio_setpin(S3C2410_GPB5, 0);
    s3c2410_gpio_setpin(S3C2410_GPB6, 1);
    s3c2410_gpio_setpin(S3C2410_GPB7, 0);
    s3c2410_gpio_setpin(S3C2410_GPB8, 1);

    if (machine_is_smdk2443())
        smdk_nand_info.twrph0 = 50;
}

```

然后，输入： `#make menuconfig`，配置如下：

```

Device Drivers --->
[ ] LED Support --->

```

然后编译出内核镜像，烧写到开发板中，您可以看到开发板的 LED1 和 LED3 这两个灯亮，LED2 和 LED4 两个灯是灭了的。

**注意：**上面修改 208 到 219 这部分，仅仅只是为了说明使 LED 灯的亮灭该怎么设置电平，这里设置之后，再经过我们驱动程序的初始化之后，实际是没有任何用处的，当然，为了使这个操作有用，我们在初始化时就不进行任何操作了。

## 22.3 编写自己的 LED 驱动

然后我们再在内核源码的“drivers/char/”目录下新建一个名为“EmbedSky\_leds.c”的文件，内容如下：

```
/*
*****

NAME:EmbedSky_leds.c
COPYRIGHT:www.embedsky.net

*****

#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <linux/init.h>
#include <linux/delay.h>
#include <asm/irq.h>
#include <asm/arch/regs-gpio.h>
#include <asm/hardware.h>
#include <linux/device.h>

#define DEVICE_NAME    "EmbedSky-leds"    /* 加载模式后，执行” cat /proc/devices ”命令看
到的设备名称 */
#define LED_MAJOR      231                /* 主设备号 */

/* 应用程序执行 ioctl(fd, cmd, arg)时的第 2 个参数 */
#define IOCTL_LED_ON    1
#define IOCTL_LED_OFF   0

/* 用来指定 LED 所用的 GPIO 引脚 */
static unsigned long led_table [] =
{
    S3C2410_GPB5,
    S3C2410_GPB6,
    S3C2410_GPB7,
    S3C2410_GPB8,
};

/* 用来指定 GPIO 引脚的功能：输出 */
static unsigned int led_cfg_table [] =
```

```

{
    S3C2410_GPB5_OUTP,
    S3C2410_GPB6_OUTP,
    S3C2410_GPB7_OUTP,
    S3C2410_GPB8_OUTP,
};

```

```

/* 应用程序对设备文件/dev/EmbedSky-leds 执行 open(...)时，
 * 就会调用 EmbedSky_leds_open 函数
 */

```

```

static int EmbedSky_leds_open(struct inode *inode, struct file *file)

```

```

{

```

```

    // int i;

```

```

    // for (i = 0; i < 4; i++)

```

```

    // {

```

```

        // 设置 GPIO 引脚的功能：本驱动中 LED 所涉及的 GPIO 引脚设为输出功能

```

```

        // s3c2410_gpio_cfgpin(led_table[i], led_cfg_table[i]);

```

```

    // }

```

```

    return 0;

```

```

}

```

```

/* 应用程序对设备文件/dev/EmbedSky-leds 执行 ioctl(...)时，

```

```

 * 就会调用 EmbedSky_leds_ioctl 函数

```

```

 */

```

```

static int EmbedSky_leds_ioctl( struct inode *inode, struct file *file, unsigned int cmd, unsigned long arg)

```

```

{

```

```

    if (arg > 4)

```

```

    {

```

```

        return -EINVAL;

```

```

    }

```

```

    switch(cmd)

```

```

    {

```

```

        case IOCTL_LED_ON:

```

```

            // 设置指定引脚的输出电平为 0

```

```

            s3c2410_gpio_setpin(led_table[arg], 0);

```

```

            return 0;

```

```

        case IOCTL_LED_OFF:

```

```

            // 设置指定引脚的输出电平为 1

```

```

            s3c2410_gpio_setpin(led_table[arg], 1);

```

```

            return 0;

```

```

        default:
            return -EINVAL;
    }
}

/* 这个结构是字符设备驱动程序的核心
 * 当应用程序操作设备文件时所调用的 open、read、write 等函数，
 * 最终会调用这个结构中指定的对应函数
 */
static struct file_operations EmbedSky_leds_fops =
{
    .owner    =    THIS_MODULE,    /* 这是一个宏，推向编译模块时自动创建的 __this_module
变量 */
    .open     =    EmbedSky_leds_open,
    .ioctl    =    EmbedSky_leds_ioctl,
};

static char __initdata banner[] = "TQ2440/SKY2440 LEDS, (c) 2008,2009 www.embedsky.net\n";
static struct class *led_class;

/*
 * 执行 “insmod EmbedSky_leds.ko” 命令时就会调用这个函数
 */
static int __init EmbedSky_leds_init(void)
{
    int ret;
    printk(banner);

    /* 注册字符设备驱动程序
     * 参数为主设备号、设备名字、file_operations 结构；
     * 这样，主设备号就和具体的 file_operations 结构联系起来了，
     * 操作主设备为 LED_MAJOR 的设备文件时，就会调用 EmbedSky_leds_fops 中的相关成员函数
     * LED_MAJOR 可以设为 0，表示由内核自动分配主设备号
     */
    ret = register_chrdev(LED_MAJOR, DEVICE_NAME, &EmbedSky_leds_fops);
    if (ret < 0)
    {
        printk(DEVICE_NAME " can't register major number\n");
        return ret;
    }

    //注册一个类，使 mdev 可以在 "/dev/" 目录下面建立设备节点
    led_class = class_create(THIS_MODULE, DEVICE_NAME);

```

```

    if(IS_ERR(led_class))
    {
        printk("Err: failed in EmbedSky-leds class.\n");
        return -1;
    }

    //创建一个设备节点，节点名为 DEVICE_NAME
    class_device_create(led_class, NULL, MKDEV(LED_MAJOR, 0), NULL, DEVICE_NAME);

    printk(DEVICE_NAME " initialized\n");
    return 0;
}

/*
 * 执行” rmmod EmbedSky_leds.ko ” 命令时就会调用这个函数
 */
static void __exit EmbedSky_leds_exit(void)
{
    /* 卸载驱动程序 */
    unregister_chrdev(LED_MAJOR, DEVICE_NAME);
    class_device_destroy(led_class, MKDEV(LED_MAJOR, 0)); //删掉设备节点
    class_destroy(led_class); //注销类
}

/* 这两行指定驱动程序的初始化函数和卸载函数 */
module_init(EmbedSky_leds_init);
module_exit(EmbedSky_leds_exit);

```

```

/* 描述驱动程序的一些信息，不是必须的 */

```

```

MODULE_AUTHOR("http://www.embedsky.net"); // 驱动程序的作者
MODULE_DESCRIPTION("TQ2440/SKY2440 LED Driver"); // 一些描述信息
MODULE_LICENSE("GPL"); // 遵循的协议

```

**注意：**以上红色部分被屏蔽了，因为在“arch/arm/plat-s3c24xx/common-smdk.c”文件中已经完成了这个初始化了。

## 22.4 在内核源码中添加对 **LED** 灯驱动的支持

然后修改同目录下的“Kconfig”文件，在 13 行开始添加如下内容：（红色部分所示）

```

#
# Character device configuration
#

menu "Character devices"

config EmbedSky_HELLO

```

```

tristate "TQ2440/SKY2440 Hello Driver"
depends on ARCH_S3C2440
help
    EmbedSky TQ2440/SKY2440 Hello.

```

```

config EmbedSky_LEDS
    tristate "TQ2440/SKY2440 LEDs Driver"
    depends on ARCH_S3C2440
    help
        EmbedSky TQ2440/SKY2440 User leds.

```

```

config VT
    bool "Virtual terminal" if EMBEDDED
    depends on !S390
    select INPUT
    default y if !VIOCONS

```

然后修改同目录的“Makefile”文件，在 13 行添加如下内容：（红色部分所示）

```

#
# Makefile for the kernel character device drivers.
#
#
# This file contains the font map for the default (hardware) font
#
FONTMAPFILE = cp437.uni

```

```

obj-y      += mem.o random.o tty_io.o n_tty.o tty_ioctl.o

```

```

obj-$(CONFIG_EmbedSky_HELLO) += EmbedSky_hello.o
obj-$(CONFIG_EmbedSky_LEDS)  += EmbedSky_leds.o
obj-$(CONFIG_LEGACY_PTYS)    += pty.o
obj-$(CONFIG_UNIX98_PTYS)    += pty.o
obj-y                        += misc.o
obj-$(CONFIG_VT)             += vt_ioctl.o vc_screen.o consolemap.o \
    consolemap_deftbl.o selection.o keyboard.o

```

添加完以上内容之后，输入：#make menuconfig，然后配置如下：

```

Device Drivers  --->
    Character devices  --->
        <*> TQ2440/SKY2440 LEDs Driver

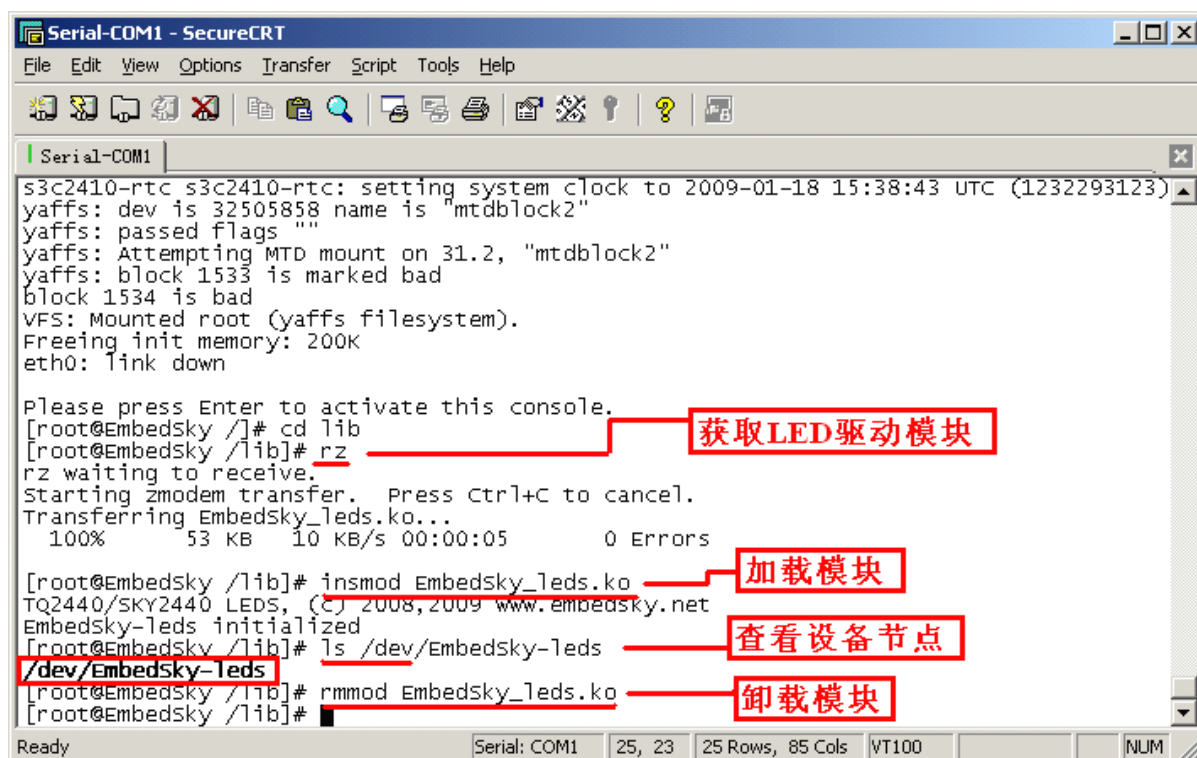
```

将其选择为“M”（模块），然后保存配置，编译出内核镜像烧写到开发板中。

然后再使用命令#make SUBDIR=drivers/char/ modules，然后编译出驱动模块，在内核目录下面

的“drivers/char/”目录下，名为：**EmbedSky\_led.ko**，将其复制到开发板中。

也可以将其配置为“\*”（添加到内核中），然后编译出镜像，烧写到开发板中，LED 灯就可以进行控制了。下面是加载模块的情况，在“/dev/”目录下有“**EmbedSky-leds**”的设备名：



```
Serial-COM1 - SecureCRT
File Edit View Options Transfer Script Tools Help

Serial-COM1
s3c2410-rtc s3c2410-rtc: setting system clock to 2009-01-18 15:38:43 UTC (1232293123)
yaffs: dev is 32505858 name is "mtdblock2"
yaffs: passed flags ""
yaffs: Attempting MTD mount on 31.2, "mtdblock2"
yaffs: block 1533 is marked bad
block 1534 is bad
VFS: Mounted root (yaffs filesystem).
Freeing init memory: 200K
eth0: link down

Please press Enter to activate this console.
[root@Embedsky /]# cd lib
[root@Embedsky /lib]# rz
rz waiting to receive.
Starting zmodem transfer. Press Ctrl+C to cancel.
Transferring Embedsky_leds.ko...
 100%   53 KB   10 KB/s 00:00:05    0 Errors

[root@Embedsky /lib]# insmod Embedsky_leds.ko
TQ2440/SKY2440 LEDs, (C) 2008,2009 www.embedsky.net
Embedsky-leds initialized
[root@Embedsky /lib]# ls /dev/Embedsky-leds
/dev/Embedsky-leds
[root@Embedsky /lib]# rmmod Embedsky_leds.ko
[root@Embedsky /lib]#
```

获取LED驱动模块

加载模块

查看设备节点

卸载模块

## 22.5 编写 LED 灯控制程序

下面列出 LED 灯的控制程序，这里的控制程序直接使用的 SKY2440 或 TQ2440 光盘提供的 LED 灯的控制程序，代码如下：（红色部分所示为设备文件名）

```
/******
```

```
NAME:leds.c
```

```
COPYRIGHT:www.embedsky.net
```

```
*****/
```

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include <unistd.h>
```

```
#include <sys/ioctl.h>
```

```
int main(int argc, char **argv)
```

```
{
```

```
    int on;
```

```
    int led_no;
```

```
    int fd;
```

```

    if (argc != 3 || sscanf(argv[1], "%d", &led_no) != 1 || sscanf(argv[2], "%d", &on) != 1 ||
        on < 0 || on > 1 || led_no < 1 || led_no > 4) {
        fprintf(stderr, "Usage: leds led_no 0|1\n");
        exit(1);
    }
    fd = open("/dev/EmbedSky-leds", 0);
    if (fd < 0) {
        perror("open device leds");
        exit(1);
    }
    ioctl(fd, on, (led_no-1));
    close(fd);
    return 0;
}

```

对应的 Makefile 文件的内容:

```

CROSS=arm-linux-

all: leds

leds:leds.c
    $(CROSS)gcc -o leds leds.c
    $(CROSS)strip leds
clean:
    @rm -vf leds *.o *~

```

参考 SKY2440 或 TQ2440 的用户手册关于 Linux 驱动开发的章节，我们添加一个 led-player 的应用程序，实现 LED 灯按照流水灯显示，将 “led-player” 和 “leds” 复制到文件系统的 “/sbin/” 目录下，然后从 2.6.13 内核的文件系统复制 “/etc/rc.d/init.d/” 目录下的 “leds” 文件到新的文件系统的 “/etc/rc.d/init.d/” 目录下，然后修改 “/etc/init.d/rcS” 文件，新的文件内容如下：（红色部分所示）

```

PATH=/sbin:/bin:/usr/sbin:/usr/bin
runlevel=S
prevlevel=N
umask 022
export PATH runlevel prevlevel

#
# Trap CTRL-C &c only in this shell so we can interrupt subprocesses.
#

mount -a

```



```
mkdir /dev/pts
mount -t devpts devpts /dev/pts
echo /sbin/mdev > /proc/sys/kernel/hotplug
mdev -s
mkdir -p /var/lock
```

```
mkdir /dev/fb /dev/v4l
ln -s /dev/fb0 /dev/fb/0
ln -s /dev/video0 /dev/v4l/video0
```

```
ln -s /dev/ts0 /dev/h3600_tsraw
```

```
hwclock -s
EmbedSky_wdg &
```

```
ifconfig lo 127.0.0.1
ifconfig eth0 hw ether 10:23:45:67:89:ab
ifconfig eth0 192.168.1.6 up
route add default gw 192.168.1.2
```

```
/etc/rc.d/init.d/httpd start
/etc/rc.d/init.d/netd start
/etc/rc.d/init.d/leds start #启动 led 灯的控制进程
```

```
qtopia &
/bin/hostname -F /etc/sysconfig/HOSTNAME
```

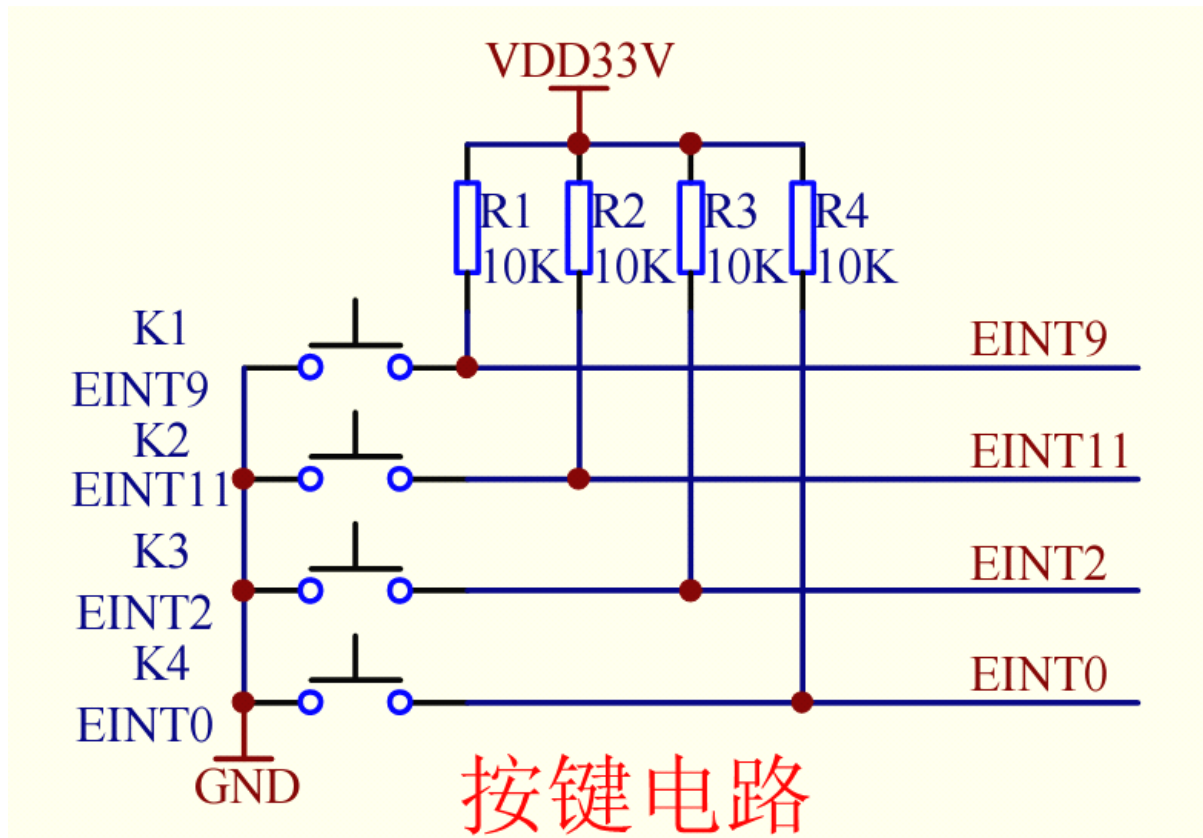
烧写新修改的文件系统的镜像到开发板中，您将看到开发板的 LED 灯在启动完成后会进行流水灯操作。

## Step 23、中断处理的驱动编写

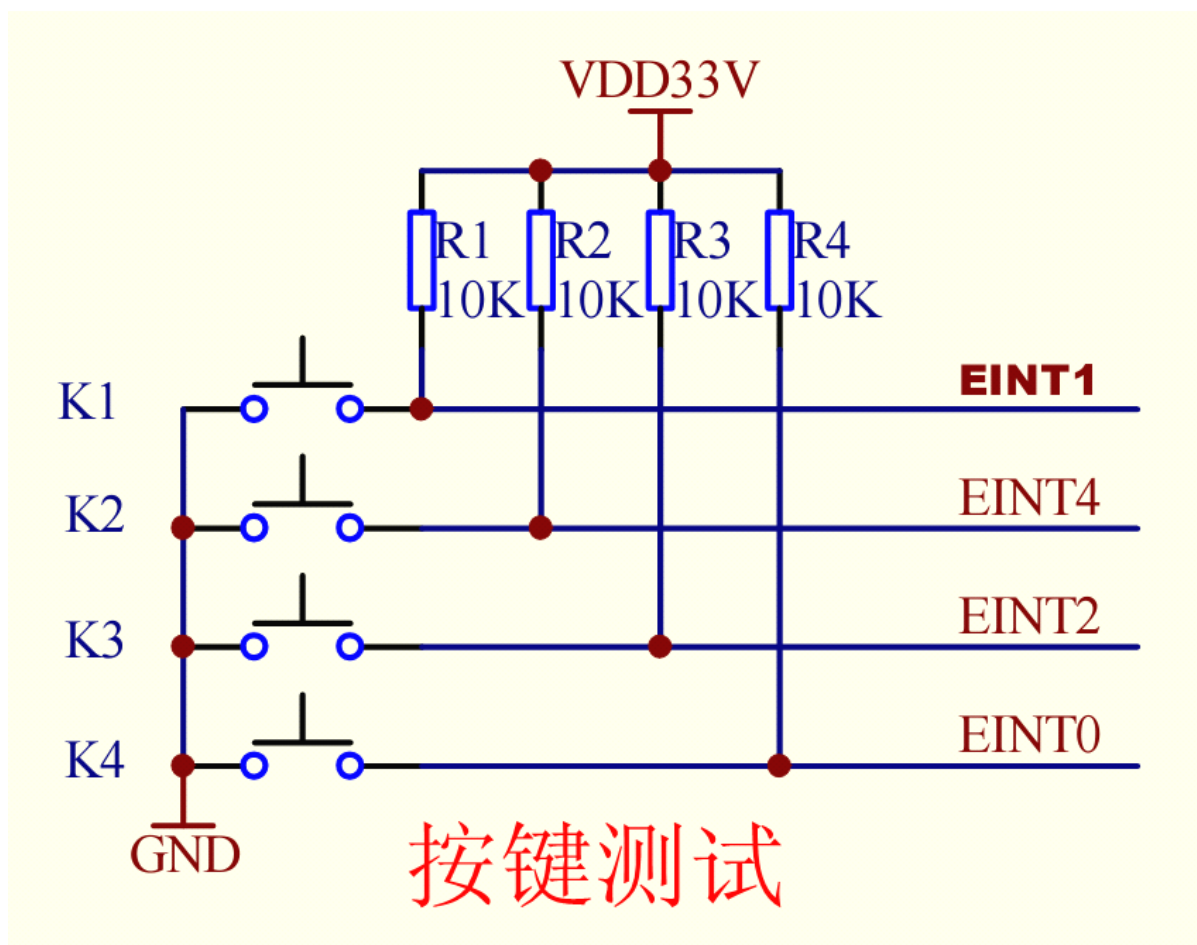
### 23.1 硬件分析

在 SKY2440 和 TQ2440 中的用户按键是使用的 S3C2440 的外部中断引脚，我们编写用户按键的驱动就是编写中断处理的驱动程序。

下图为 SKY2440 的用户按键的原理图：



下图为 TQ2440 的用户按键的原理图：



我们需要在驱动程序里面对所用到管脚初始化，设置其功能为中断，然后再设置触发电平类型即可。

## 23.2 编写键盘驱动

下面编写键盘的驱动，

我们在内核源码的“`drivers/input/keyboard/`”目录下新建一个名为“`EmbedSky_buttons.c`”的文件，内容如下：

```
*****
```

```
NAME:EmbedSky_buttons.c
```

```
COPYRIGHT:www.embedsky.net
```

```
*****/
```

```
#include <linux/module.h>
```

```
#include <linux/kernel.h>
```

```
#include <linux/fs.h>
```

```
#include <linux/init.h>
```

```
#include <linux/delay.h>
```

```
#include <asm/irq.h>
```

```

#include <linux/interrupt.h>
#include <asm/uaccess.h>
#include <asm/arch/regs-gpio.h>
#include <asm/hardware.h>
#include <linux/device.h>
#include <linux/poll.h>

#define DEVICE_NAME    "EmbedSky-buttons"    /* 加载模式后，执行” cat /proc/devices ” 命令看
到的设备名称 */
#define BUTTON_MAJOR    232                /* 主设备号 */

struct button_irq_desc
{
    int irq;
    int pin;
    int pin_setting;
    int number;
    char *name;
};

/* 用来指定按键所用的外部中断引脚及中断触发方式，名字 */
static struct button_irq_desc button_irqs [] =
{
    {IRQ_EINT1, S3C2410_GPF1,    S3C2410_GPF1_EINT1,    0, "KEY1"}, /* K1 */
    {IRQ_EINT4, S3C2410_GPF4,    S3C2410_GPF4_EINT4,    1, "KEY2"}, /* K2 */
    {IRQ_EINT2, S3C2410_GPF2,    S3C2410_GPF2_EINT2,    2, "KEY3"}, /* K3 */
    {IRQ_EINT0, S3C2410_GPF0,    S3C2410_GPF0_EINT0,    3, "KEY4"}, /* K4 */
};

/* 按键被按下的次数(准确地说，是发生中断的次数) */
static volatile int key_values [] = {0, 0, 0, 0};

/* 等待队列:
 * 当没有按键被按下时，如果有进程调用 EmbedSky_buttons_read 函数，
 * 它将休眠
 */
static DECLARE_WAIT_QUEUE_HEAD(button_waitq);

/* 中断事件标志，中断服务程序将它置 1，EmbedSky_buttons_read 将它清 0 */
static volatile int ev_press = 0;

static irqreturn_t buttons_interrupt(int irq, void *dev_id)
{
    struct button_irq_desc *button_irqs = (struct button_irq_desc *)dev_id;

```

```

int up = s3c2410_gpio_getpin(button_irqs->pin);

if (up)
    key_values[button_irqs->number] = (button_irqs->number + 1) + 0x80;
else
    key_values[button_irqs->number] = (button_irqs->number + 1);

ev_press = 1;                /* 表示中断发生了 */
wake_up_interruptible(&button_waitq); /* 唤醒休眠的进程 */

return IRQ_RETVAL(IRQ_HANDLED);
}

/* 应用程序对设备文件/dev/EmbedSky-buttons 执行 open(...)时，
 * 就会调用 EmbedSky_buttons_open 函数
 */
static int EmbedSky_buttons_open(struct inode *inode, struct file *file)
{
    int i;
    int err;

    for (i = 0; i < sizeof(button_irqs)/sizeof(button_irqs[0]); i++)
    {
        // 注册中断处理函数
        s3c2410_gpio_cfgpin(button_irqs[i].pin, button_irqs[i].pin_setting);
        err = request_irq(button_irqs[i].irq, buttons_interrupt, NULL, button_irqs[i].name, (void
*)&button_irqs[i]);
        if (err)
            break;
    }

    if (err)
    {
        // 释放已经注册的中断
        i--;
        for (; i >= 0; i--)
        {
            disable_irq(button_irqs[i].irq);
            free_irq(button_irqs[i].irq, (void *)&button_irqs[i]);
        }
        return -EBUSY;
    }

    return 0;
}

```

```
}
```

```
/* 应用程序对设备文件/dev/EmbedSky-buttons 执行 close(...)时,
```

```
 * 就会调用 EmbedSky_buttons_close 函数
```

```
*/
```

```
static int EmbedSky_buttons_close(struct inode *inode, struct file *file)
```

```
{
```

```
    int i;
```

```
    for (i = 0; i < sizeof(button_irqs)/sizeof(button_irqs[0]); i++)
```

```
    {
```

```
        // 释放已经注册的中断
```

```
        disable_irq(button_irqs[i].irq);
```

```
        free_irq(button_irqs[i].irq, (void *)&button_irqs[i]);
```

```
    }
```

```
    return 0;
```

```
}
```

```
/* 应用程序对设备文件/dev/EmbedSky-buttons 执行 read(...)时,
```

```
 * 就会调用 EmbedSky_buttons_read 函数
```

```
*/
```

```
static int EmbedSky_buttons_read(struct file *filp, char __user *buff, size_t count, loff_t *offp)
```

```
{
```

```
    unsigned long err;
```

```
    if (!ev_press)
```

```
    {
```

```
        if (filp->f_flags & O_NONBLOCK)
```

```
            return -EAGAIN;
```

```
        else
```

```
            /* 如果 ev_press 等于 0, 休眠 */
```

```
            wait_event_interruptible(button_waitq, ev_press);
```

```
    }
```

```
    /* 执行到这里时, ev_press 等于 1, 将它清 0 */
```

```
    ev_press = 0;
```

```
    /* 将按键状态复制给用户, 并清 0 */
```

```
    err = copy_to_user(buff, (const void *)key_values, min(sizeof(key_values), count));
```

```
    memset((void *)key_values, 0, sizeof(key_values));
```

```
    return err ? -EFAULT : min(sizeof(key_values), count);
```

```
}
```

```
/******
```

```
* 当用户程序调用 select 函数时，本函数被调用
```

```
* 如果有按键数据，则 select 函数会立刻返回
```

```
* 如果没有按键数据，本函数使用 poll_wait 等待
```

```
*****/
```

```
static unsigned int EmbedSky_buttons_poll( struct file *file, struct poll_table_struct *wait)
```

```
{
```

```
    unsigned int mask = 0;
```

```
    poll_wait(file, &button_waitq, wait);
```

```
    if (ev_press)
```

```
        mask |= POLLIN | POLLRDNORM;
```

```
    return mask;
```

```
}
```

```
/* 这个结构是字符设备驱动程序的核心
```

```
 * 当应用程序操作设备文件时所调用的 open、read、write 等函数，
```

```
 * 最终会调用这个结构中的对应函数
```

```
*/
```

```
static struct file_operations EmbedSky_buttons_fops =
```

```
{
```

```
    .owner    =    THIS_MODULE,      /* 这是一个宏，指向编译模块时自动创建的 __this_module
```

```
变量 */
```

```
    .open     =    EmbedSky_buttons_open,
```

```
    .release  =    EmbedSky_buttons_close,
```

```
    .read     =    EmbedSky_buttons_read,
```

```
    .poll     =    EmbedSky_buttons_poll,
```

```
};
```

```
static char __initdata banner[] = "TQ2440/SKY2440 LEDS, (c) 2008,2009 www.embedsky.net\n";
```

```
static struct class *button_class;
```

```
/*
```

```
 * 执行 “insmod EmbedSky_buttons.ko” 命令时就会调用这个函数
```

```
*/
```

```
static int __init EmbedSky_buttons_init(void)
```

```
{
```

```
    int ret;
```

```
    printk(banner);
```

```
/* 注册字符设备驱动程序
```

```
 * 参数为主设备号、设备名字、file_operations 结构；
```

```
 * 这样，主设备号就和具体的 file_operations 结构联系起来了，
```

\* 操作主设备为 BUTTON\_MAJOR 的设备文件时，就会调用 EmbedSky\_buttons\_fops 中的相关成员函数

```
* BUTTON_MAJOR 可以设为 0，表示由内核自动分配主设备号
*/
ret = register_chrdev(BUTTON_MAJOR, DEVICE_NAME, &EmbedSky_buttons_fops);
if (ret < 0)
{
    printk(DEVICE_NAME " can't register major number\n");
    return ret;
}
```

```
//注册一个类，使 mdev 可以在 "/dev/" 目录下面建立设备节点
button_class = class_create(THIS_MODULE, DEVICE_NAME);
if(IS_ERR(button_class))
{
    printk("Err: failed in EmbedSky-leds class. \n");
    return -1;
}
//创建一个设备节点，节点名为 DEVICE_NAME
class_device_create(button_class, NULL, MKDEV(BUTTON_MAJOR, 0), NULL,
DEVICE_NAME);
```

```
printk(DEVICE_NAME " initialized\n");
return 0;
}
```

```
/*
 * 执行 "rmmod EmbedSky_buttons.ko" 命令时就会调用这个函数
 */
static void __exit EmbedSky_buttons_exit(void)
{
    /* 卸载驱动程序 */
    unregister_chrdev(BUTTON_MAJOR, DEVICE_NAME);
    class_device_destroy(button_class, MKDEV(BUTTON_MAJOR, 0)); //删掉设备节点
    class_destroy(button_class); //注销类
}
```

```
/* 这两行指定驱动程序的初始化函数和卸载函数 */
module_init(EmbedSky_buttons_init);
module_exit(EmbedSky_buttons_exit);
```

```
/* 描述驱动程序的一些信息，不是必须的 */
MODULE_AUTHOR("http://www.embedsky.net"); // 驱动程序的作者
MODULE_DESCRIPTION("TQ2440/SKY2440 LED Driver"); // 一些描述信息
```



```
MODULE_LICENSE("GPL"); // 遵循的协议
```

**注意：**前面红色部分的内容是针对 TQ2440 的，如果使用的是 SKY2440，请根据实际情况修改为如下内容：

```
/* 用来指定按键所用的外部中断引脚及中断触发方式，名字 */
static struct button_irq_desc button_irqs [] =
{
    {IRQ_EINT9, S3C2410_GPG1, S3C2410_GPG1_EINT9, 0, "KEY1"}, /* K1 */
    {IRQ_EINT11, S3C2410_GPG3, S3C2410_GPG3_EINT11, 1, "KEY2"}, /* K2 */
    {IRQ_EINT2, S3C2410_GPF2, S3C2410_GPF2_EINT2, 2, "KEY3"}, /* K3 */
    {IRQ_EINT0, S3C2410_GPF0, S3C2410_GPF0_EINT0, 3, "KEY4"}, /* K4 */
};
```

## 23.3 在内核源码中添加对按键驱动的支持

然后修改同目录下的“Kconfig”文件，在 13 行开始添加如下内容：（红色部分所示）

```
if INPUT_KEYBOARD

config EmbedSky_BUTTONS
    tristate "TQ2440/SKY2440 buttons"
    depends on ARCH_S3C2440
    default y
    help
        EmbedSky TQ2440/SKY2440 Buttons.
```

```
config KEYBOARD_ATKBD
    tristate "AT keyboard" if EMBEDDED || !X86_PC
    default y
```

然后修改同目录下的“Makefile”文件，在 13 行开始添加如下内容：（红色部分所示）

```
#
# Makefile for the input core drivers.
#

# Each configuration option enables a list of files.

obj-$(CONFIG_EmbedSky_BUTTONS) += EmbedSky_buttons.o
obj-$(CONFIG_KEYBOARD_ATKBD) += atkbd.o
obj-$(CONFIG_KEYBOARD_SUNKBD) += sunkbd.o
obj-$(CONFIG_KEYBOARD_LKKBD) += lkkbd.o
obj-$(CONFIG_KEYBOARD_XTKBD) += xtkbd.o
```

## 23.4 配置内核

修改完以上的内容，输入：`#make menuconfig`，然后配置如下：

```
Device Drivers --->
  Input device support --->
    [*]   Keyboards --->
      <M>   TQ2440/SKY2440 buttons
```

然后保存配置单，分别编译出内核镜像和键盘驱动模块，编译模块的命令：`#make SUBDIR=drivers/input/keyboard/ modules`，然后在内核目录下面的“`drivers/input/keyboard/`”目录下，名为：`EmbedSky_buttons.ko`的模块，将其复制到文件系统的“`/lib/`”目录下。

## 23.5 编写按键测试程序

下面列出按键驱动的测试程序的源码，内容如下：

```
/******

NAME:buttons.c
COPYRIGHT:www.embedsky.net

*****/

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/select.h>
#include <sys/time.h>
#include <errno.h>

int main(void)
{
    int i;
    int buttons_fd;
    int key_value[4];

    /*打开键盘设备文件*/
    buttons_fd = open("/dev/EmbedSky-buttons", 0);
    if (buttons_fd < 0)
    {
```

```

        perror("open device buttons");
        exit(1);
    }

    for (;;)
    {
        fd_set rds;
        int ret;

        FD_ZERO(&rds);
        FD_SET(buttons_fd, &rds);

        /*使用系统调用 select 检查是否能够从/dev/EmbedSky-buttons 设备读取数据*/
        ret = select(buttons_fd + 1, &rds, NULL, NULL, NULL);

        /*读取出错则退出程序*/
        if (ret < 0)
        {
            perror("select");
            exit(1);
        }

        if (ret == 0)
        {
            printf("Timeout.\n");
        }

        /*能够读取到数据*/
        else if (FD_ISSET(buttons_fd, &rds))
        {
            /*开始读取键盘驱动发出的数据，注意 key_value 和键盘驱动中定义为一致的类型*/
            int ret = read(buttons_fd, key_value, sizeof(key_value));
            if (ret != sizeof(key_value))
            {
                if (errno != EAGAIN)
                {
                    perror("read buttons\n");
                    continue;
                }
            }
            else
            {
                /*打印键值*/
                for (i = 0; i < 4; i++)
                {
                    if (key_value[i] != 0)
                        printf("K%d %s, key value = 0x%02x\n", i+1, (key_value[i] & 0x80) ?
"released" : key_value[i] ? "pressed down" : "", key_value[i]);
                }
            }
        }
    }

```

```

    }
}
}
}

```

```

/*关闭设备文件句柄*/
close(buttons_fd);
return 0;
}

```

下面列出对应的“Makefile”文件的内容，如下：  
CROSS=arm-linux-

```
all: buttons
```

```

buttons:buttons.c
$(CROSS)gcc -o buttons buttons.c

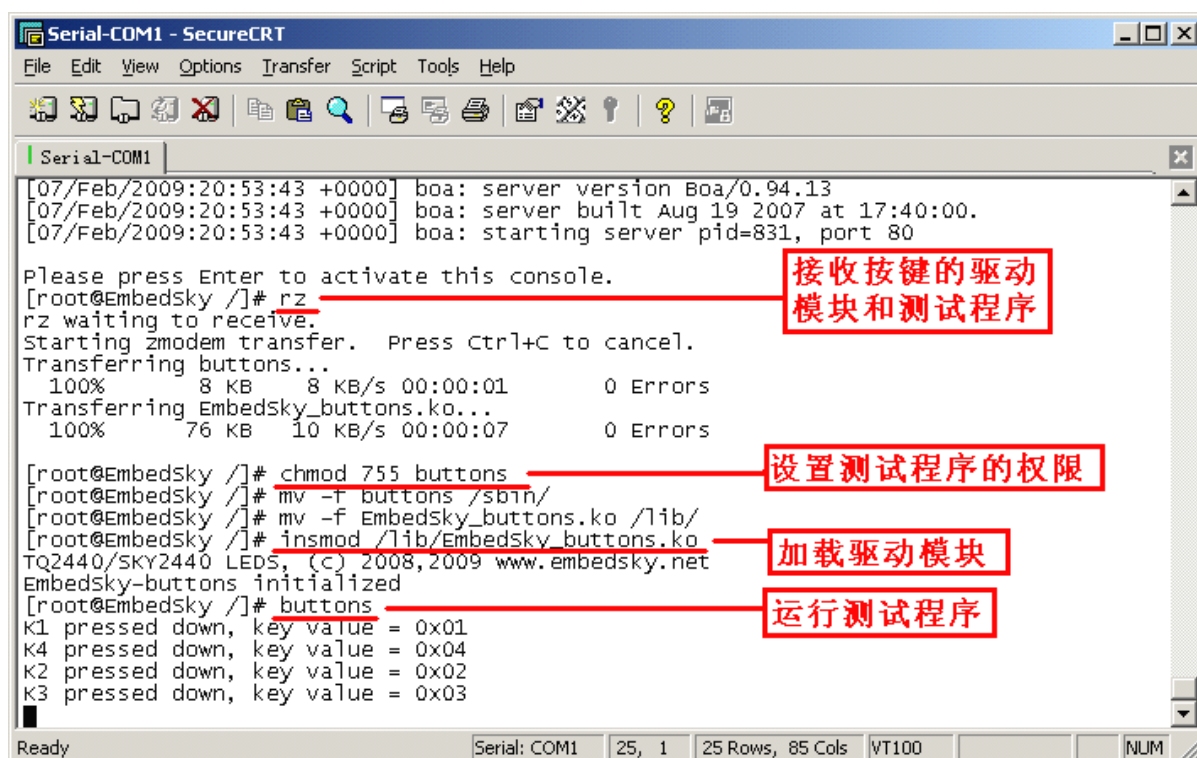
```

```

clean:
@rm -vf buttons *.o *~

```

然后编译出测试程序“buttons”，将其复制到文件系统的“/sbin/”目录下，重新制作文件系统的镜像，将其烧写到开发板中。下面是对键盘操作的截图：





```
*****/
```

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <linux/init.h>
#include <linux/device.h>
#include <linux/miscdevice.h>
#include <linux/delay.h>
#include <linux/mm.h>
#include <asm/irq.h>
#include <asm/plat-s3c/regs-timer.h>
#include <asm/arch/regs-gpio.h>
#include <asm/arch/map.h>
#include <asm/arch/regs-irq.h>
#include <asm/io.h>
#include <asm/hardware.h>
#include <asm/uaccess.h>
#include <asm/system.h>
```

```
#define DEVICE_NAME "EmbedSky-Beep" /* 加载模式后，执行” cat /proc/devices ” 命令看
到的设备名称 */
```

```
#define BEEP_MAJOR 233 /* 主设备号 */
```

```
/* 应用程序对设备文件/dev/EmbedSky-Beep 执行 ioctl(...)时，
```

```
 * 就会调用 EmbedSky_beep_ioctl 函数
```

```
*/
```

```
static int EmbedSky_beep_ioctl(
```

```
    struct inode *inode,
```

```
    struct file *file,
```

```
    unsigned int cmd,
```

```
    unsigned long arg)
```

```
{
```

```
    unsigned long temp;
```

```
    if(cmd <= 0)
```

```
    {
```

```
        temp = __raw_readl(S3C2410_GPBCON); //GPBCON
```

```
        temp &= ~3;
```

```
        temp |= 1;
```

```
        __raw_writel(temp, S3C2410_GPBCON);
```

```
        temp = __raw_readl(S3C2410_GPBDAT); //GPBDAT
```

```
        temp &= ~1;
```

```
        __raw_writel(temp, S3C2410_GPBDAT);
```

```

    }
    else
    {
        temp = __raw_readl(S3C2410_GPBCON); //GPBCON
        temp &= ~3;
        temp |= 2;
        __raw_writel(temp, S3C2410_GPBCON);

        temp = __raw_readl(S3C2410_TCFG0); //TCFG0
        temp &= ~0xff;
        temp |= 15;
        __raw_writel(temp, S3C2410_TCFG0);

        temp = __raw_readl(S3C2410_TCFG1); //TCFG1
        temp &= ~0xf;
        temp |= 2;
        __raw_writel(temp, S3C2410_TCFG1);

        temp = (50000000/128)/cmd;
        __raw_writel(temp, S3C2410_TCNTB(0));

        temp >>= 1;
        __raw_writel(temp, S3C2410_TCMPB(0));

        temp = __raw_readl(S3C2410_TCON); //TCON
        temp &= ~0x1f;
        temp |= 0xb;
        __raw_writel(temp, S3C2410_TCON);

        temp &= ~2;
        __raw_writel(temp, S3C2410_TCON);
    }
    return 0;
}

```

```

/* 这个结构是字符设备驱动程序的核心
 * 当应用程序操作设备文件时所调用的 open、read、write、ioctl 等函数，
 * 最终会调用这个结构中指定的对应函数
 */
static struct file_operations EmbedSky_beep_fops = {
    .owner    =    THIS_MODULE,
    .ioctl    =    EmbedSky_beep_ioctl,
};

```

```

static char __initdata banner[] = "TQ2440/SKY2440 Beep, (c) 2008,2009 www.embedsky.net\n";
static struct class *beep_class;

/*
 * 执行 “insmod EmbedSky_beep.ko” 命令时就会调用这个函数
 */
static int __init EmbedSky_beep_init(void)
{
    int ret;
    printk(banner);

    /* 注册字符设备驱动程序
     * 参数为主设备号、设备名字、file_operations 结构;
     * 这样，主设备号就和具体的 file_operations 结构联系起来了，
     * 操作主设备为 BEEP_MAJOR 的设备文件时，就会调用 EmbedSky_beep_fops 中的相关成员
     函数
     * BEEP_MAJOR 可以设为 0，表示由内核自动分配主设备号
     */
    ret = register_chrdev(BEEP_MAJOR, DEVICE_NAME, &EmbedSky_beep_fops);
    if (ret < 0) {
        printk(DEVICE_NAME " can't register major number\n");
        return ret;
    }

    //注册一个类，使 mdev 可以在 "/dev/" 目录下面建立设备节点
    beep_class = class_create(THIS_MODULE, DEVICE_NAME);
    if(IS_ERR(beep_class))
    {
        printk("Err: failed in EmbedSky-Beep class. \n");
        return -1;
    }

    //创建一个设备节点，节点名为 DEVICE_NAME
    class_device_create(beep_class, NULL, MKDEV(BEEP_MAJOR, 0), NULL, DEVICE_NAME);

    printk(DEVICE_NAME " initialized\n");
    return 0;
}

/*
 * 执行 “rmmod EmbedSky_beep.ko” 命令时就会调用这个函数
 */
static void __exit EmbedSky_beep_exit(void)
{

```



```

/* 卸载驱动程序 */
unregister_chrdev(BEEP_MAJOR, DEVICE_NAME);
class_device_destroy(beep_class, MKDEV(BEEP_MAJOR, 0)); //删掉设备节点
class_destroy(beep_class); //注销类
}

/* 这两行指定驱动程序的初始化函数和卸载函数 */
module_init(EmbedSky_beep_init);
module_exit(EmbedSky_beep_exit);

/* 描述驱动程序的一些信息，不是必须的 */
MODULE_AUTHOR("http://www.embedsky.net"); // 驱动程序的作者
MODULE_DESCRIPTION("TQ2440/SKY2440 Beep Driver"); // 一些描述信息
MODULE_LICENSE("GPL"); // 遵循的协议

```

## 24.3 在内核源码中添加对 **Beep** 的支持

然后修改同目录下的“Kconfig”文件，在 19 行开始添加如下内容：（红色部分所示）

```

#
# Character device configuration
#

menu "Character devices"

config EmbedSky_HELLO
    tristate "TQ2440/SKY2440 Hello Driver"
    depends on ARCH_S3C2440
    help
        EmbedSky TQ2440/SKY2440 Hello.

config EmbedSky_LEDS
    tristate "TQ2440/SKY2440 LEDs Driver"
    depends on ARCH_S3C2440
    help
        EmbedSky TQ2440/SKY2440 User leds.

config EmbedSky_Beep
    tristate "TQ2440/SKY2440 Beep Driver"
    depends on ARCH_S3C2440
    help
        EmbedSky TQ2440/SKY2440 Beep control.

```

然后修改同目录的“Makefile”文件，在 14 行添加如下内容：（红色部分所示）

```

#

```

```
# Makefile for the kernel character device drivers.
#
#
# This file contains the font map for the default (hardware) font
#
FONTMAPFILE = cp437.uni

obj-y      += mem.o random.o tty_io.o n_tty.o tty_ioctl.o

obj-$(CONFIG_EmbedSky_HELLO) += EmbedSky_hello.o
obj-$(CONFIG_EmbedSky_LEDS)  += EmbedSky_leds.o
obj-$(CONFIG_EmbedSky_Beep)  += EmbedSky_beep.o
obj-$(CONFIG_LEGACY_PTYS)    += pty.o
obj-$(CONFIG_UNIX98_PTYS)    += pty.o
obj-y                        += misc.o
obj-$(CONFIG_VT)             += vt_ioctl.o vc_screen.o consolemap.o \
                             consolemap_deftbl.o selection.o keyboard.o
```

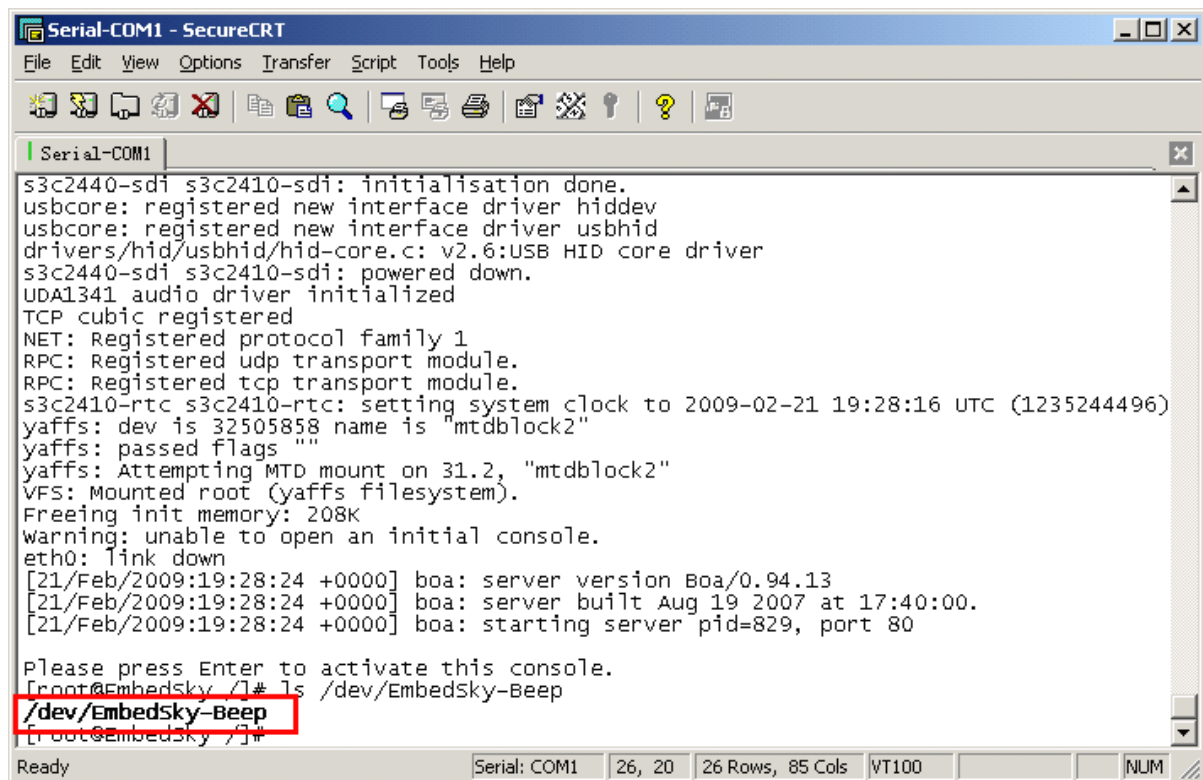
## 24.4 配置内核

添加完以上内容之后，输入： `#make menuconfig`，然后配置如下：

```
Device Drivers  --->
    Character devices  --->
        <*> TQ2440/SKY2440 Beep Driver
```

将其选择为“M”（模块），然后保存配置，编译出内核镜像烧写到开发板中，然后再使用命令 `#make SUBDIR=drivers/char/ modules`，然后编译出驱动模块，在内核目录下面的“`drivers/char/`”目录下面，名为：`EmbedSky_beep.ko`，将其复制到开发板中。

也可以将其配置为“\*”（添加到内核中），然后编译出镜像，烧写到开发板中，蜂鸣器就可以进行控制了。下面是加载模块的情况，在“`/dev/`”目录下面有“`EmbedSky-beep`”的设备名：



```
Serial-COM1 - SecureCRT
File Edit View Options Transfer Script Tools Help

Serial-COM1
s3c2440-sdi s3c2410-sdi: initialisation done.
usbcore: registered new interface driver hiddev
usbcore: registered new interface driver usbhid
drivers/hid/usbhid/hid-core.c: v2.6:USB HID core driver
s3c2440-sdi s3c2410-sdi: powered down.
UDA1341 audio driver initialized
TCP cubic registered
NET: Registered protocol family 1
RPC: Registered udp transport module.
RPC: Registered tcp transport module.
s3c2410-rtc s3c2410-rtc: setting system clock to 2009-02-21 19:28:16 UTC (1235244496)
yaffs: dev is 32505858 name is "mtdblock2"
yaffs: passed flags ""
yaffs: Attempting MTD mount on 31.2, "mtdblock2"
VFS: Mounted root (yaffs filesystem).
Freeing init memory: 208K
warning: unable to open an initial console.
eth0: link down
[21/Feb/2009:19:28:24 +0000] boa: server version Boa/0.94.13
[21/Feb/2009:19:28:24 +0000] boa: server built Aug 19 2007 at 17:40:00.
[21/Feb/2009:19:28:24 +0000] boa: starting server pid=829, port 80

Please press Enter to activate this console.
[root@Embedsky /]# ls /dev/Embedsky-Beep
/dev/Embedsky-Beep
[root@Embedsky /]#

Ready Serial: COM1 26, 20 26 Rows, 85 Cols VT100 NUM
```

## 24.5 编写 PWM 测试程序

下面列出蜂鸣器的控制程序，代码如下：（红色部分所示为设备文件名）

```
*****
```

```
NAME:beep.c
```

```
COPYRIGHT:www.embedsky.net
```

```
*****/
```

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#include <unistd.h>
```

```
#include <sys/ioctl.h>
```

```
#include <sys/stat.h>
```

```
#include <fcntl.h>
```

```
struct spihr
```

```
{
```

```
    unsigned short status;
```

```
    unsigned short dat;
```

```
};
```

```
int main(int argc, char **argv)
```

```
{
```

```
    int fd;
```

```

unsigned long temp =0;
int i;

fd = open("/dev/EmbedSky-Beep", O_RDWR);
if (fd < 0) {

    perror("open device EmbedSky-BEEP");
    exit(1);
}

for(i=0;i<2000;i++)
{
    scanf("%d",&temp);
    printf("temp= %d \n",temp);
    ioctl(fd, temp, 4);
    if(temp == 0)
        break;
}
close(fd);
return 0;
}

```

对应的 Makefile 文件的内容：

```

CROSS=arm-linux-

all: beep

beep: beep.c
    $(CROSS)gcc -o beep beep.c
    $(CROSS)strip beep
clean:
    @rm -vf beep *.o *~

```

将编译出来的名为“beep”的可执行文件复制到文件系统的“/sbin/”目录下，然后压制为 yaffs 镜像烧写到开发板中，运行 \$beep 命令后，输入数字，蜂鸣器就会按照您输入的数据进行鸣叫。

Serial-COM1 - SecureCRT

File Edit View Options Transfer Script Tools Help

Serial-COM1

```
eth0: link down
[21/Feb/2009:20:28:03 +0000] boa: server version Boa/0.94.13
[21/Feb/2009:20:28:03 +0000] boa: server built Aug 19 2007 at 17:40:00.
[21/Feb/2009:20:28:03 +0000] boa: starting server pid=829, port 80

Please press Enter to activate this console.
[root@EmbedSky /]# beep
Please enter the times number (0 is stop) :
188
times = 188
288
times = 288
388
times = 388
488
times = 488
588
times = 588
666
times = 666
888
times = 888
0
times = 0
Stop Control Beep!
[root@Embedsky /]#
```

Ready Serial: COM1 26, 20 26 Rows, 85 Cols VT100 NUM

输入数字，然后蜂鸣器  
会根据数字的大小进行  
发出不同频率的声音  
输入0，表示推出

## Step 25、内核裁剪

这里主要实现内核镜像尺寸的减小。

优化 1、取消虚拟内存的支持

General setup --->

☐ Support for paging of anonymous memory (swap)

优化 2、使用默认的 IO 调度器，取消其他的调度器

.\*- Enable the block layer --->

IO Schedulers --->

<> Anticipatory I/O scheduler

<\*> Deadline I/O scheduler

<> CFQ I/O scheduler

Default I/O scheduler (Deadline) --->

优化 3、取消对旧版本二进制执行文件的支持

Userspace binary formats --->

<> Kernel support for a.out and ECOFF binaries

优化 4、取消不必要的设备的支持

Device Drivers --->

<\*> Memory Technology Device (MTD) support --->

☒ MTD partitioning support

<> RedBoot partition table parsing

RAM/ROM/Flash chip drivers --->

<> Detect flash chips by Common Flash Interface (CFI) probe

<> Detect non-CFI AMD/JEDEC-compatible flash chips

<> Support for RAM chips in bus mapping

<> Support for ROM chips in bus mapping

<> Support for absent chips in bus mapping

<> Parallel port support --->

☐ Block devices --->

<> ATA/ATAPI/MFM/RLL support --->

Input device support --->

☒ Keyboards --->

<> AT keyboard

Character devices --->

☐ Non-standard serial port support

Serial drivers --->

<> 8250/16550 and compatible serial support

\*\*\* Non-8250 serial port support \*\*\*

<\*> Samsung S3C2410/S3C2440/S3C2442/S3C2412 Serial port support

☒ Support for console on S3C2410 serial port

☐ Legacy (BSD) PTY support

SPI support --->

☐ SPI support

<> Hardware Monitoring support --->

优化 5、取消不需要的文件系统的支持

File systems --->

<> Second extended fs support

<> Ext3 journalling file system support

<> Ext4dev/ext4 extended fs support development (EXPERIMENTAL)

Miscellaneous filesystems --->

<> Journalling Flash File System v2 (JFFS2) support

完成以上的优化配置有，内核镜像会由之前的 1.9MB 缩减到 1.7MB 左右。

# 结束语

当您看到这里的时候，整个移植手册也算是学习完毕了。天嵌科技还会陆续推出系列教程，帮助您掌握嵌入式设备的开发，学习在于积累，当您在遇到不会的时候或者程序编译出错时，建议您先仔细阅读出错信息，仔细分析不会的可能原因，可以利用网络搜索相关出错信息，综合分析后，再提问，这样对于您的能力提高有帮助。

如果对于移植过程中有什么疑问，可以列出来放到我们的论坛：<http://www.embedsky.net/bbs>上，然后我们会定期进行回答，对于提问了，并已经解决了的，也请贴出解决方法，以方便后来人。