# Linux-2.6.31 内核移植手册

2010-06-17 V1.4

**手册内容简介：**

本手册是天祥电子推出的 TX-2440A 开发板的配套手册之一，全面分析了 linux 操作系统内核在嵌入式开发板上的移植过程，手册中的部分内容都会在配套的视频教程中（第十五讲，第十六讲）进行讲解。

在开始进行移植之前，需要了解 linux 内核和驱动的相关知识，安装基本的开发工具，搭建好开发环境；建议先看视频教程学习，在具备了一定的能力后再来参考本手册来操作。

本手册分为 16 个部分，第 1 部分，准备移植，要先掌握这一部分的操作方法，包括（修改代码，配置内核选项，编译内核镜像，下载内核镜像到开发板，启动系统），然后再进入后面的部分。第 2、3 部分是为挂载文件系统做准备，这里容易出现错误（比如：文件系统不能成功挂载），可以先学习文件系统的相关知识（视频教程第十七讲），自己制作文件系统（参考《文件系统制作手册》），在能成功挂载文件系统后，再进入后面的部分。第 4-14 部分都是针对开发板上的硬件的驱动移植，还介绍硬件驱动的测试方法。第 15 部分列出了驱动程序在内核源码中的位置及设备名称，方便大家查阅。第 16 部分是内核更新的内容，对内核的功能和驱动进行优化。

**说明：**

由于个人能力有限，手册中难免会出现一些笔误和不足之处，如果发现问题，请及时提出，目前可以发到 QQ 群上（103105892 111874027），或发到我的博客上（blog.163.com/xgc94418297）；在学习过程中遇到的问题，可以联系我们，我们会提供技术支持。

本手册中的所有内容目前仅适用于 TX2440A 开发板，如果将其用在其他的开发板上，出现的一切问题，我们一律不提供技术支持。

最好不要从手册中直接复制代码，因为在编写手册时，有些字符可能会自动变成全角格式，在代码中是不允许有全角字符的，所以有可能会导致一些错误。我建议，最好是手动编写代码。

手册中的内容会不定期的更新，以后还会加入更丰富的内容，更新后会放到 FTP 服务器供大家下载。希望广大的 TX2440A 开发板用户能多提出宝贵的意见，我会根据大家的意见及时调整手册中的内容。

2010-06-17 相广超 制作
哈尔滨祥鹏科技有限公司
网址：www.txmcu.com
电话：0451-87572303

**内核版本：**

Linux-2.6.31

**交叉编译器版本：**

arm-linux-gcc 4.1.2

**操作系统平台：**

Linux -- Red Hat 9.0

**开发板平台：**

Arm -- TX2440A

**手册中字体颜色的约定：**

修改的代码用红色字体

执行的命令用红色字体，前面加 #

添加的大段代码用蓝色字体

在终端上打印出的信息用紫色字体

出现的错误信息用绿色字体

**说明的文字用红色粗体**

需要修改的文件加灰色底纹

**版本信息：**

| | | |
|---|---|---|
| 2009-11-10 | V1.0 | 初稿 |
| 2010-03-06 | V1.1 | |

1. 调整了第 1、3、6 部分的内容
2. 修改了第 14 部分测试背光驱动的错误

| | | |
|---|---|---|
| 2010-3-16 | V1.2 | |
| 2010-4-21 | V1.3 | 加入了内核更新（第十六章） |
| 2010-6-17 | V1.4 | |

1. 加入了 7 寸 LCD 和 LOGO 的支持（第四章）
2. 加入了串口 2 驱动（第十六章）

# 手册目录：

## 一> 准备移植

获得内核源码：

Linux-2.6.31.tar.bz2(在光盘资料/源码包/kernel 源码目录下)

也可到官方网站http://www.kernel.org/获得最新版本的内核源码

解压源码，进入目录：

```
#tar xjvf linux-2.6.31.bz2
#cd linux-2.6.31
```

修改 Makefile： 183 行：

```
ARCH ? = arm                          ←指定系统硬件架构
CROSS_COMPILE ?= arm-linux-           ←指定交叉编译器
```

修改时钟：

修改 arch/arm/mach-s3c2440/mach-smdk2440.c 163 行

```
static void __init smdk2440_map_io(void)
{
s3c24xx_init_io(smdk2440_iodesc, ARRAY_SIZE(smdk2440_iodesc));
s3c24xx_init_clocks(12000000);      ←输入时钟为 12MHz
s3c24xx_init_uarts(smdk2440_uartcfgs, ARRAY_SIZE(smdk2440_uartcfgs));
}
```

这个一定要设置对，否则会打印出乱码。

修改机器码（根据实际情况，这个要和 bootloader 的匹配）：

修改：arch/arm/tools/mach-types 379 行：

```
s3c2440 ····························  XXX     后面那个数就是机器码
```

配置：

```
#make menuconfig        ←进入图形化配置界面
```

在配置菜单中选择这一项：“Load an Alternate Configuration File”

输入 2440 的默认配置文件：arch/arm/configs/s3c2410_defconfig

**说明：这个文件就是 S3C24XX 系列开发板的板级支持包（BSP）**

**然后选择 OK，按回车**

进入“System Type”选项单，里面的选项保持默认

在“S3C24XX Machine”选项中只配置这几项（其他的选项取消）：

```
S3C2410 Machine --->
    [*] SMDK2410/A9M2410
S3C2440 Machine --->
    [*] SMDK2440
    [*] SMDK2440 with S3C2440 cpu moudle
```

配置完后，回到主菜单，选择这一项“Save an Alternate Configuration File”
输入要保存的配置文件名称：.config（默认）或自己取名：TX2440A_config
退出，编译内核： #make zImage

**说明：以后移植过程中的配置、编译，都是按这个步骤进行，但是只需要保存一次配置文件，以后就不需要再保存配置文件了，配置完后可以直接退出。**

编译完后，会在 arch/arm/boot 下生成 zImage 内核镜像文件
可以修改该目录下的 Makefile： 在第 57 行下面添加：

```
@cp  -f arch/arm/boot/zImage zImage
@echo `  Kernel: $@ is ready`
```

这样执行 make zImage 后，就把生成的 zImage 拷到内核根目录下

如果希望在在执行 make distclean 时，也同时把 zImage 删除，
可以修改内核源码目录下 Makefile 的第 1247 行，在后面加上：

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

把 zImage 镜像烧进 NandFlash 跑一下，看是否正常打印出信息
如果第一步能正常引导内核，那就开始进行第二步，添加驱动

注意，系统启动最后可能会出现这个错误：

```
Kernel panic - not syncing: Attempted to kill init!
```

然后出打印出一些很乱的东西。
因为用 4.X.X 版本的交叉编译器使用 EABI，但内核默认是不支持 EABI 编译的，所以编译出的系统会报错，但用 3.X.X 版本的编译器就不会出现这个问题。
解决办法是，配置内核支持 EABI 编译

```
Kernel Features  --->
    [*] Use the ARM EABI to compile the kernel
    [*]     Allow old ABI binaries to run with this kernel (EXPERIMENTA)
```

## 二> 支持 NandFlash：

修改：arch/arm/plat-s3c24xx/common-smdk.c 文件，在第 110 行：
这里我们要使 nandflash 同时支持 64M，256M 或更高容量。

```
static struct mtd_partition smdk_default_nand_part[] = {
#if defined(CONFIG_64M_NAND)
    [0] = {
        .name   = "boot",
        .offset     = 0,
        .size   = SZ_1M,
    },
    [1] = {
        .name   = "kernel",
        .offset = SZ_1M + SZ_128K,
        .size   = SZ_4M,
    },
    [2] = {
        .name   = "yaffs2",
        .offset = SZ_1M + SZ_128K + SZ_4M,
        .size   = SZ_64M - SZ_4M - SZ_1M - SZ_128K,
    }
#elif defined(CONFIG_256M_NAND)
    [0] = {
        .name   = "boot",
        .offset     = 0,
        .size   = SZ_1M,
    },
    [1] = {
        .name   = "kernel",
        .offset = SZ_1M + SZ_128K,
        .size   = SZ_4M,
    },
    [2] = {
        .name   = "yaffs2",
        .offset = SZ_1M + SZ_128K + SZ_4M,
        .size   = SZ_256M - SZ_4M - SZ_1M  -  SZ_128K,
```

```
    }
#endif
};
```

这个分区名字可以随便起。

接下来修改 Nand 读写匹配时间，这个改不改应该问题都不大，我认为是根据 Nand 的读写特性相关的，也就是查芯片资料得到的值，每种 Nand 的值都不一样，还是在这个文件中第 140 行：

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

修改 Kconfig 文件，在配置时选择 NAND 类型，修改 driver/mtd/nand/Kconfig,在 172 行，添加：

```
choice
    prompt "Nand Flash Capacity Select"
    depends on MTD

config 64M_NAND
    boolean "64M NAND For TX-2440A"
    depends on MTD

config 256M_NAND
    boolean "256M NAND For TX-2440A"
    depends on MTD

endchoice
```

配置内核，支持 NandFlash

```
Device Drivers --->
     <*> Memory Technology Device (MTD) support --->
```

```
[*]   MTD partitioning support
<*> NAND Device Support --->
    <*>   NAND Flash support for S3C2410/S3C2440 SoC
    [*]     S3C2410 NAND Hardware ECC      //这个一定要选上
      Nand Flash Capacity Select(256M Nand For TX-2440A)--->
```

启动时输出：

```
S3C24XX NAND Driver, (c) 2004 Simtec Electronics
s3c24xx-nand s3c2440-nand: Tacls=1, 10ns Twrph0=3 30ns, Twrph1=1 10ns
s3c24xx-nand s3c2440-nand: NAND hardware ECC
NAND device: Manufacturer ID: 0xec, Chip ID: 0xda (Samsung NAND 256MiB 3,3V
8-bit)
Scanning device for bad blocks
Creating 3 MTD partitions on "NAND 256MiB 3,3V 8-bit":
0x000000000000-0x000000100000 : "boot"
0x000000120000-0x000000520000 : "kernel"
0x000000520000-0x000010000000 : "yaffs2"
```

## 三> 支持 yaffs2 文件系统

下载 yaffs2 源码(在光盘资料/源码包/其他软件源码/目录下)
解压，进入 yaffs2 目录：

```
#tar xzvf yaffs2.tar.gz
#cd cvs/yaffs2/
```

给内核打上 yaffs2 文件系统的补丁，执行：

```
#./patch-ker.sh c /·····/linux-2.6.31/
```

**←这个是你的内核源码的目录**

这时内核源码 fs 目录下多了一个 yaffs2 目录，同时 Makefile 文件和 Kconfig 文件也增加了 yaffs2 的配置和编译条件。

**配置对 yaffs2 支持：**

这部分配置的比较多，可根据自己的需要进行配置，把不用的文件系统都去掉，下面是几个主要的配置：

```
File systems --->
    DOS/FAT/NT Filesystems --->
        <*> MSDOS fs support
```

```
        <*> VFAT (Windows95) fs support
    Miscellaneous filesystems --->
     <*> YAFFS2 file system support
     [*] Autoselect yaffs2 format
```

配置语言选项：

```
    Native Language support --->
        (iso8859-1) Default NLS Option
        <*> Codepage 437(United States, Canada)
        <*> Simplified Chinese charset(CP936, GB2312)
        <*> NLS ISO8859-1 (Latin 1; Western European Language)
        <*> NLS UTF-8
```

**说明：现在内核已经支持 NandFlash 和 yaffs2 文件系统，将内核烧入 NandFlash 后，再烧入 yaffs2 文件系统，可以使用制作好的文件系统，也可以自己制作，详细的制作文件系统方法，请查看《文件系统制作手册》**

启动时（成功挂载文件系统）输出：

```
yaffs: dev is 32505858 name is "mtdblock2"
yaffs: passed flags ""
yaffs: Attempting MTD mount on 31.2, "mtdblock2"
yaffs: auto selecting yaffs2
yaffs_read_super: isCheckpointed 0
VFS: Mounted root (yaffs filesystem) on device 31:2.
Freeing init memory: 196K
```

## 四> LCD 驱动的移植

内核里已经有很完善的 LCD 驱动了，只要根据所用的 LCD 进行简单的修改，在内核源码 drivers/video/s3c2410fb.c 是 LCD 驱动的源码，首先要设置 LCD 的时钟频率，有一个计算公式，很复杂，不用管它，直接修改程序实现。

在第 365 行开始的函数：

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

这几句是在 s3c2410fb_display 结构体中加入了 setclkval 变量，我们需要在结构体原型中加入这个变量，在 arch/arm/mach-s3c2410/include/mach/fb.h 中第 40 行加入：unsigned setclkval;  /*clkval*/

**说明：在视频教程中修改 s3c2410fb.c 文件时，和手册上有一点误差，手册上写的是正确的，请按照手册上操作**

修改 LCD 参数配置：（这个要查看所用 LCD 的手册来确定参数）

修改 arch/arm/mach-s3c2440/mach-smdk2440.c 中第 107 行的结构体，我修改的参

数如下：

```
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 60 MHz, divisor 10 */
    .setclkval    = 0x3,  /*add by xgc*/
    .xres       = 320,
    .yres       = 240,
    .bpp        = 16,
    .left_margin    = 19,
    .right_margin = 24,
    .hsync_len = 44,
    .upper_margin = 7,
    .lower_margin = 5,
    .vsync_len = 15,
};
```

屏蔽掉第 150 行的语句：

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

**配置内核，支持 LCD：**

```
Device Drivers:
    Graphics Support  --->
        <*>support for frame buffer devices --->
            [*] Enable frameware EDID
            [*] Enable Vidoe Mode Handling Helpers
```

```
        <*> S3C24X0 LCD framebuffer support
    Console display driver support --->
        <*> Framebuffer Console Support
    [*] Bootup Logo --->
        <*> Standard 224-color Linux logo
```

启动时输出：

```
Console: switching to colour frame buffer device 40x30
fb0: s3c2410fb frame buffer device
```

**添加对 7 寸 LCD 的支持：**

我们使用的 7 寸屏是群创的 AT070TN83 V.1（带触摸屏），分辨率是 800X480。
同样修改 arch/arm/mach-s3c2440/mach-smdk2440.c 中第 107 行的结构体，加入 7 寸屏的配置参数：

```
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_S3C2410_W35)
    .width      = 320,
    .height         = 240,

    .pixclock   = 100000, /* HCLK 60 MHz, divisor 10 */
    .setclkval    = 0x3,  /*add by xgc*/
    .xres       = 320,
    .yres       = 240,
    .bpp        = 16,
    .left_margin  = 19,
    .right_margin = 24,
```

```
    .hsync_len = 44,
    .upper_margin = 7,
    .lower_margin = 5,
    .vsync_len = 15,

#elif defined(CONFIG_FB_S3C2410_Q70)
    .width     = 800,
    .height        = 480,

    .pixclock  = 40000, /* HCLK 40 MHz*/
    .setclkval   = 0x1,   /*add by xgc*/
    .xres      = 800,
    .yres      = 480,
    .bpp       = 16,
    .left_margin   = 40, /*HFPD*/
    .right_margin = 40, /*HBPD*/
    .hsync_len = 48, /*HSPW*/
    .upper_margin = 13, /*VFPD*/
    .lower_margin = 29, /*VBPD*/
    .vsync_len = 3, /*VSPW*/

#endif
};
```

修改 drivers/video/Kconfig 文件，在 1942 行加入：

```
choice
    prompt "LCD size select"
    depends on FB_S3C2410
    help
        s3c2410 lcd size select

config FB_S3C2410_W35
    boolean"The 3.5 inch LCD with resolution 320X240"
    depends on FB_S3C2410
    help
        3.5 inch LCD 320X240
```

```
config FB_S3C2410_Q70
    boolean"The 7 inch LCD with resolution 800X480"
    depends on FB_S3C2410
    help
        7 inch LCD 800X480
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
```

**配置内核，支持 7 寸 LCD：**

```
Device Drivers:
    Graphics Support  --->
        <*>support for frame buffer devices --->
            <*> S3C24X0 LCD framebuffer support
            LCD size select(The 3.5 inch LCD with resolution 320X240) --->
```

进入“LCD size select(The 3.5 inch LCD with resolution 320X240)--->”选项，在这里选择不同的 LCD 类型（默认的是 3.5 寸屏），按空格键即可选中，前面有“X”的表示选中的，如果我们要选择 7 寸屏，将光标移至第二行，按空格键，就选中了对 7 寸屏的支持

```
（ ）The 3.5 inch LCD with resolution 320X240
（X）The 7 inch LCD with resolution 800X480
```

## 五> DM9000 驱动的移植

修改 arch/arm/mach-s3c2440/mach-smdk2440.c 160 行

Platform_device 结构体中，加入：

```
&s3c_device_dm9000,
```

修改 arch/arm/plat-s3c24xx/devs.c 在最前面 38 行加入：

```
#include <linux/dm9000.h>  //别忘加这个头文件

/*DM9000*/
static struct resource s3c_dm9000_resource[] = {
    [0] = {
        .start = S3C2410_CS4,
        .end   = S3C2410_CS4 + 3,
        .flags = IORESOURCE_MEM,
    },
    [1] = {
        .start = S3C2410_CS4 + 4,
        .end   = S3C2410_CS4 + 4 + 3,
        .flags = IORESOURCE_MEM,
    },
    [2] = {
        .start = IRQ_EINT18,  /*use eint18 GPG10*/
        .end   = IRQ_EINT18,
        .flags = IORESOURCE_IRQ,
    }
};

static struct dm9000_plat_data s3c_dm9000_platdata = {
    .flags    = DM9000_PLATF_16BITONLY,
};

static struct platform_device s3c_device_dm9000 = {
    .name     = "dm9000",
    .id       = 0,
    .num_resources = ARRAY_SIZE(s3c_dm9000_resource),
    .resource = s3c_dm9000_resource,
    .dev      = {
        .platform_data = &s3c_dm9000_platdata,
    }
};
```

```
EXPORT_SYMBOL(s3c_device_dm9000);
```

在 arch/arm/plat-s3c/include/plat/devs.h 中加入一行：

```
extern struct platform_device s3c_device_dm9000;
```

编译时出现一个奇怪的错误：

```
arch/arm/plat-s3c24xx/devs.c:63:    error:    static    declaration    of
's3c_device_dm9000' follows non-static declaration
arch/arm/plat-s3c/include/plat/devs.h:27: error: previous declaration of
's3c_device_dm9000' was here
```

s3c_device_dm9000 首先在 devs.c 中定义，在 devs.h 中声明，在 mach-s3c2440.c 中使用，看了好几遍代码，应该没什么问题。查不到什么原因，我觉得是跟编译器有关，在 devs.c 中做了一下修改，问题解决，编译时只出现一个 warning

把 devs.c 中的这句：

```
static struct platform_device s3c_device_dm9000 = {
```

的 static 改成 extern,就可以了

下面修改 dm9000.c 源码，在 drivers/net/dm9000.c 中

1. 添加头文件，在第 43 行加入：

```
#if defined(CONFIG_ARCH_S3C2410)
#include <mach/regs-mem.h>
#endif
```

2. 指定注册时的中断触发方式，在第 1019 行加入：

```
static int dm9000_open(struct net_device *dev)
{
.....
    irqflags |= IRQF_SHARED;

#if defined (CONFIG_ARCH_S3C2410)
    if(request_irq(dev->irq,&dm9000_interrupt,IRQF_SHARED|IRQF_TRIGGER_RI
SING,dev->name,dev))
#else
    if(request_irq(dev->irq,&dm9000_interrupt,IRQF_SHARED,dev->name,dev))
```

```
#endif
    //if (request_irq(dev->irq, &dm9000_interrupt, irqflags, dev->name,
dev))
    return -EAGAIN;
.....
}
```

3. 设置 BANK4，设置 MAC 地址，在 1215 行，dm9000_probe 函数中加入：

```
    int ret = 0;
    int iosize;
    int i;
    u32 id_val;

#if defined(CONFIG_ARCH_S3C2410)
    unsigned int oldval_bwscon = *(volatile unsigned int *)S3C2410_BWSCON;
    unsigned int oldval_bankcon4 = *(volatile unsigned int
*)S3C2410_BANKCON4;
#endif

    /* Init network device */
    ndev = alloc_etherdev(sizeof(struct board_info));
    if (!ndev) {
        dev_err(&pdev->dev, "could not allocate device.\n");
        return -ENOMEM;
    }
```

在 1231 行加入：

```
    SET_NETDEV_DEV(ndev, &pdev->dev);

    dev_dbg(&pdev->dev, "dm9000_probe()\n");

#if defined(CONFIG_ARCH_S3C2410)
    *((volatile unsigned int *)S3C2410_BWSCON) = (oldval_bwscon & ~(3<<16))
| S3C2410_BWSCON_DW4_16 | S3C2410_BWSCON_WS4 | S3C2410_BWSCON_ST4;
    *((volatile unsigned int *)S3C2410_BANKCON4) = 0x1f7c;
#endif
```

在 1390 行加入：

```
    db->mii.mdio_read    = dm9000_phy_read;
    db->mii.mdio_write   = dm9000_phy_write;

#if defined(CONFIG_ARCH_S3C2410)
    printk("Now use the default MAC address: 08:90:90:90:90:90 ");
    mac_src = "www.txmcu.com";
    ndev->dev_addr[0] = 0x08;
    ndev->dev_addr[1] = 0x90;
    ndev->dev_addr[2] = 0x90;
    ndev->dev_addr[3] = 0x90;
    ndev->dev_addr[4] = 0x90;
    ndev->dev_addr[5] = 0x90;
#else

    mac_src = "eeprom";

    /* try reading the node address from the attached EEPROM */
    for (i = 0; i < 6; i += 2)
        dm9000_read_eeprom(db, i / 2, ndev->dev_addr+i);

    if (!is_valid_ether_addr(ndev->dev_addr) && pdata != NULL) {
        mac_src = "platform data";
        memcpy(ndev->dev_addr, pdata->dev_addr, 6);
    }

    if (!is_valid_ether_addr(ndev->dev_addr)) {
        /* try reading from mac */

        mac_src = "chip";
        for (i = 0; i < 6; i++)
            ndev->dev_addr[i] = ior(db, i+DM9000_PAR);
    }

    if (!is_valid_ether_addr(ndev->dev_addr))
        dev_warn(db->dev, "%s: Invalid ethernet MAC address. Please "
```

```
            "set using ifconfig\n", ndev->name);
#endif
    platform_set_drvdata(pdev, ndev);
    ret = register_netdev(ndev);

    if (ret == 0)
        printk(KERN_INFO "%s: dm9000%c at %p,%p IRQ %d MAC: %pM (%s)\n",
               ndev->name, dm9000_type_to_char(db->type),
               db->io_addr, db->io_data, ndev->irq,
               ndev->dev_addr, mac_src);
    return 0;

out:
#if defined(CONFIG_ARCH_S3C2410)
    *(volatile unsigned int *)S3C2410_BWSCON = oldval_bwscon;
    *(volatile unsigned int *)S3C2410_BANKCON4 = oldval_bankcon4;
#endif
```

**配置内核，支持网卡：**

```
    Device Drivers --->
        [*] Network device support --->
            [*] Ethernet(10 or 100 Mbit) --->
             <*> DM9000 support
             (4) DM9000 maximum debug level
```

启动时输出：

```
dm9000 Ethernet Driver, V1.31
Now use the default MAC address: 08:90:90:90:90:90
eth0: dm9000e at c881c000,c8820004 IRQ 62 MAC: 08:90:90:90:90:90
```

系统启动后，可能会出现这个错误：

```
ifconfig: SIOCSIFFLAGS: Cannot assign requested address
```

原因是 MAC 地址没有设置或没有设置对，在启动脚本中加上：

```
ifconfig eth0 down
ifconfig eth0 hw ether XX:XX:XX:XX:XX:XX  ←MAC 地址，随便设
```

```
ifconfig eth0 up
```

一般问题可以解决，如果还提示有错，再改一下 MAC 地址

测试网卡的方法：连接好计算机和开发板之间的网线，如果网开发板网卡的灯亮起，说明已经连接到计算机；
可以在计算机的命令行窗口下（开始->运行->cmd）使用 PING 命令测试网络：
ping 192.168.1.10 **←开发板的 IP 地址**
也可以在开发板的串口终端下 PING 计算机的网卡

## 六> UDA1341 声卡驱动的移植：

硬件接法：L3MODE -> GPB2 L3DATA->GPB3 L3CLOCK->GPB4

内核自带的声卡驱动，可以正常编译，也会打印出正确的配置信息，但是播放时没有声音，也不能进行录音。要替换掉内核自带的驱动(注意先备份)，用 2.6.29.4 内核中的声卡驱动

将 2.6.29.4 内核源码（光盘资料->源码包->kernel 源码->linux-2.6.29.4.tar.bz2)目录下的：
**sound 文件夹，**
**include/sound 文件夹，**
**include/asm-arm/plat-s3c24xx 文件夹，**
**arch/arm/mach-s3c2410/include/mach/audio.h 文件**
复制到 2.6.31 内核源码的相应目录下，

声卡驱动的移植步骤和 DM9000 的移植大体相同

修改 arch/arm/mach-s3c2440/mach-smdk2440.c 161 行
Platform_device 结构体中，加入：

```
&s3c24xx_uda134x,
```

修改 arch/arm/plat-s3c24xx/devs.c,在 DM9000 那段代码下面加入：

```
static struct s3c24xx_uda134x_platform_data s3c24xx_uda134x_data = {
        .l3_clk = S3C2410_GPB(4),
        .l3_data = S3C2410_GPB(3),
        .l3_mode = S3C2410_GPB(2),
```

```
        .model = UDA134X_UDA1341,
};

extern struct platform_device s3c24xx_uda134x = {
        .name             = "s3c24xx_uda134x",
        .dev = {
                .platform_data    = &s3c24xx_uda134x_data,
        }
};

EXPORT_SYMBOL(s3c24xx_uda134x);
```

在 arch/arm/plat-s3c/include/plat/devs.h 中加入一行：

```
extern struct platform_device s3c24xx_uda134x;
```

注意：编译时会出错，提示 S3C2410_GPBX，UDA134X_UDA1341，13_mode 等没有定义，这里需要在 devs.c 中包含两个头文件

```
#include <mach/regs-gpio.h>    //这个是 S3C2410 的 GPIO 定义
#include <sound/s3c24xx_uda134x.h>
```

**配置内核，支持声卡：**

Device Drivers:
* Sound card support →
* Advanced Linux Sound Architecture→
* CCS Mixer API
* CSS PCM(digital audio) API
* Verbose procfs contents
* ALSA for SoC audio support→
* SoC audio for the Samsung S3C24XX chips
* SoC I2S Audio support for UDA134X wired to a S3C24XX

编译内核，会报错：

```
Sound/core/info.c:159:error:' struct proc_dir_entry' has no member named
 'owner'
Sound/core/info.c:982:error:' struct proc_dir_entry' has no member named
 'owner'
```

在 include/linux/proc_fs.h 文件中定义这个结构体成员，在第 70 行加入：

```
struct module *owner;
```

继续编译，又会出现一个错误：

Sound/soc/s3c24xx/s3c24xx-i2s.c:407:error:implicit declaration of function ‘s3c2410_gpio_cfgpin’

在 2.6.31 内核中，s3c2410_gpio_cfgpin 是在 include/linux/gpio.h 中定义的，要添加这个文件

在 sound/soc/s3c24xx/s3c24xx-i2s.c 中，第 24 行添加：

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

继续编译，又会出现很多错误：

sound/soc/s3c24xx/s3c24xx-pcm.c 中的 S3C2410_DISRCC_INC 等常量没有定义，s3c2410_dam_config 函数的参数个数不对

原因是 2.6.31 内核中 dma 相关的文件改变了，以前的跟 dma 有关的代码就不能使用了，我们只需把原来代码中的 sound/soc/s3c24xx/s3c24xx-pcm.c 这个文件替换回来就可以了。

编译时同样会出现和 DM9000 一样的错误，解决方法也是把

```
static struct platform_device s3c24xx_uda134x = {
```

(static 改成 extern)

如果没有问题了，启动时会打印出以下信息：

```
Advanced Linux Sound Architecture Driver Version 1.0.18a.
No device for DAI UDA134X
No device for DAI s3c24xx-i2s
S3C24XX_UDA134X SoC Audio driver
UDA134X SoC Audio Codec
asoc: UDA134X <-> s3c24xx-i2s mapping ok
ALSA device list:
  #0: S3C24XX_UDA134X (UDA134X)
```

前面两句不用管，只要后面的信息都打出来就 OK 了，可以在源码下把这两条警告给屏蔽掉

下面测试一下声卡，设备名称为：/dev/dsp

用命令：#cat /dev/dsp > /tmp/1.wav 进行录音，录完后 Ctrl+C
用命令：#cat /tmp/1.wav > /dev/dsp 进行放音，如果听到刚才的录音，就说明声卡好使了，再用 madplay 测试一下音质，一般都没什么问题。

## 七> SD 卡驱动移植：

内核自带 SD 卡驱动，在 drivers/mmc/目录下
在 arch/arm/mach-s3c2440/mach-smdk2440.c plat_device 结构体中加入：
&s3c_device_sdi,
修改 drivers/mmc/host/s3cmci.c 在 1335 行加入
host->irq_cd = IRQ_EINT16;
s3c2410_gpio_cfgpin(S3C2410_GPG8, S3C2410_GPG8_EINT16);
这两句指定 SD 的中断为 EINT16

修改同文件，屏蔽掉 1358-1359 行：

```
    if (s3c2410_dma_request(S3CMCI_DMA, &s3cmci_dma_client, NULL) < 0) {
        dev_err(&pdev->dev, "unable to get DMA channel.\n");
//      ret = -EBUSY;
//      goto probe_free_irq_cd;
    }
```

再将 1147-1148 行输出的多余信息屏蔽掉：

```
    if ((ios->power_mode == MMC_POWER_ON) ||
        (ios->power_mode == MMC_POWER_UP)) {
//      dbg(host, dbg_conf, "running at %lukHz (requested: %ukHz).\n",
//          host->real_rate/1000, ios->clock/1000);
    } else {
        dbg(host, dbg_conf, "powered down.\n");
    }
```

**配置内核，支持 SD：**

```
    Device Drivers --->
        <*> MMC/SD/SDIO card support --->
            <*> MMC block device driver
            <*>    Use bounce buffer for simple hosts
```

<*> Sumsung S3C SD/MMC Card Interface support

启动时输出：

s3c2440-sdi s3c2440-sdi: powered down.
s3c2440-sdi s3c2440-sdi: initialisation done.
s3c2440-sdi s3c2440-sdi: powered down.

挂载 SD 卡：

插入 SD 卡后，会提示：

mmc0: new SD card at address b368
mmcblk0: mmc0:b368 SD 970 MiB
mmcblk0: p1

那么 SD 卡的设备名称就是 mmcblk0p1，将它挂载到/mnt/sd 目录下：

#mount /dev/mmcblk0p1 /mnt/sd

进入/mnt/sd 目录就可以查看 SD 卡中的内容了。

卸载 SD 卡，用命令：

#umount /mnt/sd

## 八> RTC 驱动移植：

内核源码自带 RTC 驱动，在 arch/arm/mach-s3c2440/mach-smdk2440.c 中添加 RTC 设备，在 plat_device 结构体中加入：

&s3c_device_rtc,

**配置内核，支持 RTC：**

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 drivers)
<*>Samsung S3C series SoC RTC

启动时输出：

S3C24XX RTC, (c) 2004,2006 Simtec Electronics

```
s3c2410-rtc s3c2410-rtc: rtc disabled, re-enabling
s3c2410-rtc s3c2410-rtc: rtc core: registered s3c as rtc0
```

在终端下用 busybox1.15.2 自带的 date 命令来查看和设置时间

```
#date    输入命令
Thu Jan  1 00:01:36 UTC 1970    显示时间
#date -s 2009.10.22-16:30:10   设置时间格式：年.月.日-时:分:秒
Thu Oct 22 16:30:10 UTC 2009
#hwclock -w    保存时间
```

然后在文件系统的启动脚本中加入命令：hwclock -s
每次启动系统时就会自动同步硬件 RTC 时间：

```
s3c2410-rtc s3c2410-rtc: setting system clock to 2009-10-22 16:32:07 UTC
```

## 九> 触摸屏驱动移植：

内核里没有完善的触摸屏驱动，可以用我们提供的触摸屏驱动
有三个文件：s3c2410_ts.c, s3c2440_adc.c, s3c2440adc.h
(在光盘资料/源码包/驱动源码/触摸屏驱动)
将 s3c2410_ts.c 拷贝到 drivers/input/touchscreen 目录下,修改该目录下 Kconfig 文件，在第 14 行加入：

```
config TOUCHSCREEN_S3C2410
    tristate "Samsung S3C2410 touchscreen input driver"
    depends on INPUT && S3C2440_ADC
    help
    Say Y here if you have the s3c2410 touchscreen.
If unsure, say N.
To compile this driver as a module, choose M here: the
module will be called s3c2410_ts.
```

修改同目录下 Makefile 文件，在第 9 行加入：

```
obj-$(CONFIG_TOUCHSCREEN_S3C2410)       += s3c2410_ts.o
```

将 s3c2440_adc.c, s3c2440adc.h 拷贝到 drivers/char 目录下，修改同目录下 Kconfig 文件，在第 7 行加入：

```
config S3C2440_ADC
     bool "ADC driver for S3C2440 development boards"
     help
     this is ADC driver for S3C2440 development boards
     Notes: the touch-screen-driver required this option
```

修改同目录下 Makefile 文件，在第 12 行加入：

```
obj-$(CONFIG_S3C2440_ADC)       += s3c2440_adc.o
```

**配置内核，支持触摸屏：**

```
Device Drivers --->
    Character devices --->
        [*] ADC driver for S3C2440 development boards
    Input devices support --->
        <*> Event interface
        [*] Touchscreens --->
            <*> Samsung S3C2410 touchscreen input driver
```

启动时输出：

```
s3c2410 TouchScreen successfully loaded
input: s3c2410 TouchScreen as /class/input/input0
```

## 十> USB 设备驱动移植

内核里已经做好了很完善的 USB 驱动了，可以支持大多数 USB 设备，TX2440A 板子上使用了 USB HUB，扩展出四个 USB，内核里也有对 USB HUB 的支持，可直接使用。

**配置内核，支持 USB：**

```
Device drivers --->
    SCSI Device support --->
     <*> SCSI device support
     [*] legacy /proc/scsi/ support
     <*> SCSI disk 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)
        <*> USB Monitor
        <*> OHCI HCD support
        <*> USB Mass Storage support
```

USB Human Interface Device (full HID) support 是对 USB 鼠标键盘的支持
SCSI disk support 和 USB Mass Storage support 是对 U 盘的支持

启动时输出：

```
Initializing USB Mass Storage driver...
usbcore: registered new interface driver usb-storage
USB Mass Storage support registered.
usbcore: registered new interface driver libusual

usb 1-1: configuration #1 chosen from 1 choice
hub 1-1:1.0: USB hub found
hub 1-1:1.0: 4 ports detected
```

如果出现这个，说明已经找到 USB HUB，可以使用四个 USB 设备。

**U 盘的挂载：**

插入 U 盘后会提示：

```
[root@TX2440A /dev]# usb 1-1.4: new full speed USB device using s3c2410-ohci
and address 3          ← 插入第三个 USB 口
usb 1-1.4: configuration #1 chosen from 1 choice
scsi0 : SCSI emulation for USB Mass Storage devices
scsi 0:0:0:0: Direct-Access     Kingston DataTraveler G2  1.00 PQ: 0 ANSI:
2
sd 0:0:0:0: [sda] 7831552 512-byte logical blocks: (4.00 GB/3.73 GiB)
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: Attached scsi generic sg0 type 0
```

```
sd 0:0:0:0: [sda] Assuming drive cache: write through
 sda: sda1     ← 设备名称
sd 0:0:0:0: [sda] Assuming drive cache: write through
sd 0:0:0:0: [sda] Attached SCSI removable disk
```

U 盘的设备名称就是 sda1，将它挂载到/mnt/udisk3 目录下：

```
#mount /dev/sda1 /mnt/udisk3
```

进入/mnt/udisk3 目录就可以查看 U 盘中的内容了。

卸载 U 盘，用命令：

```
#umount /mnt/udisk3
```

如果再插入一个 U 盘，会提示：

```
[root@TX2440A /]# usb 1-1.1: new full speed USB device using s3c2410-ohci
and address 4
usb 1-1.1: configuration #1 chosen from 1 choice
scsi1 : SCSI emulation for USB Mass Storage devices
scsi 1:0:0:0: Direct-Access     Lenovo   USB Flash Drive  1100 PQ: 0 ANSI:
0 CCS
sd 1:0:0:0: [sdb] 15663104 512-byte logical blocks: (8.01 GB/7.46 GiB)
sd 1:0:0:0: Attached scsi generic sg1 type 0
sd 1:0:0:0: [sdb] Write Protect is off
sd 1:0:0:0: [sdb] Assuming drive cache: write through
sd 1:0:0:0: [sdb] Assuming drive cache: write through
 sdb: sdb1    ← 设备名称
sd 1:0:0:0: [sdb] Assuming drive cache: write through
sd 1:0:0:0: [sdb] Attached SCSI removable disk
```

注意，新插入的这个 U 盘的设备名称为 sdb1，依此类推，再插入的 U 盘设备名称将会是 sdc1 和 sdd1。

## 十一> USB 摄像头驱动移植

**配置内核，支持 USB 摄像头：**

```
    Device Drivers --->
        <*> Multimedia support --->
```

```
<*> Video For Linux
[*] Enable Video For Linux API 1 (DEPRECATED)
[*] Video capture adapters --->
   [*] V4L USB devices --->
      <*> USB Video Class (UVC)
      [*] UVC input events device support
      [*] GSPCA based webcams --->
```

到这里，我们就可以选择所需要的 USB 摄像头驱动，当然也可以选择所有的 USB 摄像头驱动支持（这样编译出的内核会比较大）

GSPCA 是一个万能摄像头驱动程序，进入 GSPCA based webcams 进行选择。

插入 USB 摄像头（我使用的 UVC 摄像头），会提示：

```
usb 1-1.2: new full speed USB device using s3c2410-ohci and address 5
usb 1-1.2: configuration #1 chosen from 1 choice
uvcvideo: Found UVC 1.00 device Saturn USB 2.0 Camera. (0ac8:3313)
input: Saturn USB 2.0 Camera. as /class/input/input1
```

它的设备名称是：/dev/video0

## 十二> CMOS 摄像头驱动移植

准备好 CMOS 摄像头驱动源码，包含 5 个文件：

s3c2440_ov9650.c s3c2440_camif.c s3c2440_camif.h sccb.c sccb.h

(在光盘资料/源码包/驱动源码/camera 驱动)

将这 5 个文件复制到 drivers/media/video 目录下，修改 drivers/media/目录下 Kconfig 文件，在 101 行加入：

```
config  S3C2440_CAMERA
    tristate "OV9650 on the S3C2440 driver"
    depends on VIDEO_DEV && ARCH_S3C2410
    default y if (VIDEO_DEV && ARCH_S3C2410)
```

修改 drivers/media/video 目录下 Makefile 文件，在 15 行加入：

```
s3c2440_camera-objs := s3c2440_ov9650.o s3c2440_camif.o sccb.o
```

在后面 165 行加入：

```
obj-$(CONFIG_S3C2440_CAMERA) += s3c2440_camera.o
```

**配置内核，支持 CMOS 摄像头：**

```
Device Drivers --->
    <*> Multimedia support --->
        <*> OV9650 on the S3C2440 driver
```

启动时输出：

```
initializing s3c2440 camera interface......
s3c2440 camif init done
Loading OV9650 driver.........
SCCB address 0x60, manufacture ID 0xFFFF, expect 0x7FA2
```

## 十三> 其他字符设备驱动移植

TX2440A 开发板上还有很多外围设备：LED，按键，ADC，蜂鸣器，温度传感器，红外收发器，CAN 总线控制器。我们需要编写驱动，并加入到内核中。

由于这些驱动全部加入内核，会使内核体积增大，并且我们也不经常用到这些设备，最好的办法是把这些驱动编译成模块，放入文件系统中，然后可以根据需要动态的加载或卸载驱动模块，增加了驱动使用的灵活性。

将驱动程序源码(在光盘资料/源码包/驱动源码)
TX2440_adc.c TX2440_beep.c TX2440_button.c TX2440_ds18b20.c TX2440_led.c
放到 drivers/char/目录下，修改同目录下 Kconfig 文件，在第 13 行加入：

```
config TX2440_LED
    tristate "TX2440 LED Driver"
    depends on ARCH_S3C2440
    help
      this is LED Driver for TX2440 development boards

config TX2440_BEEP
    tristate "TX2440 BEEP Driver"
    depends on ARCH_S3C2440
    help
      this is BEEP Driver for TX2440 development boards
```

```
config TX2440_ADC
    tristate "TX2440 ADC Driver"
    depends on ARCH_S3C2440
    help
      this is ADC Driver for TX2440 development boards

config TX2440_BUTTON
    tristate "TX2440 BUTTON Driver"
    depends on ARCH_S3C2440
    help
      this is BUTTON Driver for TX2440 development boards

config TX2440_DS18B20
    tristate "TX2440 DS18B20 Driver"
    depends on ARCH_S3C2440
    help
      this is DS18B20 Driver for TX2440 development boards
```

修改同目录下 Makefile 文件，在 13 行加入：

```
obj-$(CONFIG_TX2440_LED) += TX2440_led.o
obj-$(CONFIG_TX2440_BEEP)   += TX2440_beep.o
obj-$(CONFIG_TX2440_ADC) += TX2440_adc.o
obj-$(CONFIG_TX2440_BUTTON) += TX2440_button.o
obj-$(CONFIG_TX2440_DS18B20) += TX2440_ds18b20.o
```

**配置内核，支持驱动模块：**

```
    Device Drivers --->
        Character devices --->
            [M] TX2440 LED Driver
            [M] TX2440 BEEP Driver
            [M] TX2440 ADC Driver
            [M] TX2440 BUTTON Driver
            [M] TX2440 DS18B20 Driver
```

‘M’表示将驱动编译成模块

执行：#make M=drivers/char/ modules

**注意：视频和原手册里写的是 #make SUBDIR=drivers/char/ modules，应改成上面这个命令**

编译完成后，会在 drivers/char/目录下生成 TX2440_xxx.ko 的文件，将这几个文件复制到根文件系统下 lib/modules/2.6.31/文件夹下。

在光盘资料/源码包/驱动测试程序源码 目录下有这几个驱动的测试程序，在这几个目录中执行 make，编译出可执行程序，复制到文件系统的 usr/bin 目录下。

烧入制作好的根文件系统，因为这里只是编译了驱动模块，内核并没有更新，所以不需要重新下载内核。

如果要加载驱动，在开发板上执行：

#insmod lib/modules/2.6.31/TX2440_xxx.ko

即可动态加载驱动，然后用测试程序，测试驱动的使用情况

如果要卸载驱动，在开发板上执行：

#rmmod TX2440_xxx 即可动态卸载驱动

如果要查看当前系统已加载的驱动，可执行：

#lsmod 会列出加载的驱动列表

**注意：在 2.6.31 版本的内核中，驱动模块必须放在 lib/modules/2.6.31 目录下，执行 insmod 和 rmmod 时，系统会自动查找该目录下的驱动模块。**

## 十四> LCD 背光驱动移植和开机 LOGO 的制作

将背光驱动源码(在光盘目录/源码包/驱动源码)TX2440_backlight.c 放到 drivers/video/目录下，修改同目录下 Kconfig 文件，在 1948 行加入：

```
config TX2440_BACKLIGHT
    tristate "Backlight support for TX2440A"
    depends on FB_S3C2410
    help
      LCD backlight driver for TX2440A
```

修改同目录下 Makefile 文件，在最后面加入：

```
obj-$(CONFIG_TX2440_BACKLIGHT) += TX2440_backlight.o
```

配置内核，支持 LCD 背光：

```
Device Drivers --->
    Graphics support --->
     <*> Support for frame buffer devices --->
       <*> Backlight support for TX2440A
```

系统启动时，自动加载背光驱动，设备名为：TX2440-backlight

测试背光驱动：

执行 #echo 0 > /dev/TX2440-backlight　　会使 LCD 背光熄灭

执行 #echo 1 > /dev/TX2440-backlight　　会使 LCD 背光点亮

**制作开机 LOGO：**

**方法一：**

drivers/video/logo/logo_linux_clut224.ppm 是默认的启动 LOGO 图片，格式为 ppm，把自己的 LOGO 图片（png 格式）转换成 ppm 格式，替换这个文件，同时删除 logo_linux_clut224.c logo_linux_clut224.o 文件，重新编译

ppm 图片的生成：

```
# pngtopnm logo_linux_clut224.png > logo_linux_clut224.pnm
# pnmquant 224 logo_linux_clut224.pnm > logo_linux_clut224.pnm
# pnmtoplainpnm logo_linux_clut224.pnm > logo_linux_clut224.ppm
```

然后重新编译内核，启动就可以了！

我们使用的屏是 320X240 的，要找一个 320X240 大小的 png 格式图片！

分析：先把 png 格式转换成 pnm 格式，但内核的 LOGO 最高只支持 224 色，所以要把颜色转换成 224 色(第二条命令),这时可能会出错:pnmcolormap ELF read error 之类的，可能是转成 png 格式时不对，确保 png 的格式文件正常

最后把 pnm 转成 ppm，文件名必须是这个 logo_linux_clut224.ppm。

**方法二：**

用 RedHat9 自带的图片编辑工具 GIMP。

找一个任意格式的图片（JPG，BMP 之类的），在图形界面中，右键单击这个图片，选“打开方式->The GIMP”，第一次打开需要安装 GIMP 软件，安装完后自动运行 GIMP，这是一个很强大的图像处理工具，可称为 Linux 下的 photoshop。

右键单击窗口中的图片选“图像->模式->索引”，把颜色数改为：224（这步很重要）。其他的都默认，OK 后右键“文件->Save As”，保存为 ppm 格式的文件，确定后弹出一个对话框，选择 Ascii，OK 后，GIMP 会把图片转换成 ppm 格式，把这个文件复制到 logo 文件夹中就可以了

**制作不同分辨率的开机 LOGO，以适应所使用的 LCD：**

我们要使开机 LOGO 自动适应于 3.5 寸和 7 寸 LCD，现在已经有了 320X240 的图片，再按照上面的方法制作一张 800X480 的图片，取名为：logo_linux_clut224_q70.ppm

修改 drivers/video/logo 目录下的 Kconfig 文件，在第 26 行添加：

```
config LOGO_LINUX_CLUT224
    bool "Standard 224-color Linux logo W35"
    depends on LOGO && FB_S3C2410_W35
    default y

config LOGO_LINUX_Q70_CLUT224
    bool "Standard 224-color Linux logo Q70"
    depends on LOGO && FB_S3C2410_Q70
    default y
```

修改同目录下的 Makefile 文件，在第 6 行添加：

```
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-$(CONFIG_LOGO_LINUX_Q70_CLUT224)   += logo_linux_q70_clut224.o
obj-$(CONFIG_LOGO_BLACKFIN_CLUT224)    += logo_blackfin_clut224.o
obj-$(CONFIG_LOGO_BLACKFIN_VGA16)  += logo_blackfin_vga16.o
```

修改内核源码目录下的 include/linux/linux_logo.h 文件，在第 37 行添加：

```
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_q70_clut224;
extern const struct linux_logo logo_blackfin_vga16;
extern const struct linux_logo logo_blackfin_clut224;
```

修改 drivers/video/logo/logo.c 文件，在第 64 行添加：

```
    if (depth >= 8) {
#ifdef CONFIG_LOGO_LINUX_CLUT224
        /* Generic Linux logo */
        logo = &logo_linux_clut224;
#endif

#ifdef CONFIG_LOGO_LINUX_Q70_CLUT224
        /* Generic Linux logo */
        logo = &logo_linux_q70_clut224;
#endif

#ifdef CONFIG_LOGO_BLACKFIN_CLUT224
        /* Blackfin Linux logo */
        logo = &logo_blackfin_clut224;
#endif
```

**配置内核：**

进入 LCD 配置选项（见第四部分），选择 LCD 类型，开机 LOGO 会自动适应你所选择的 LCD 尺寸。

如果选择了 3.5 寸屏，在 Bootup logo 里的配置选项为“Standard 224-color Linux logo W35”。

如果选择了 7 寸屏,在 Bootup logo 里的配置选项为“Standard 224-color Linux logo Q70”。

## 十五> 驱动程序在内核源码中的位置及设备名称：

1. LCD /dev/fb0
Linux-2.6.31_TX2440A/drivers/video/s3c2410fb.c
2. DM9000 无设备名
Linux-2.6.31_TX2440A/drivers/net/DM9000.c
3. UDA1341 /dev/dsp
Linux-2.6.31_TX2440A/sound/soc/s3c24xx/
4. SD /dev/mmcblk0

Linux-2.6.31_TX2440A/drivers/mmc/

5. RTC /dev/rtc0

Linux-2.6.31_TX2440A/drivers/rtc/rtc-s3c.c

6. 触摸屏 /dev/event0

Linux-2.6.31_TX2440A/drivers/input/touchscreen/s3c2410_ts.c

7. USB 摄像头 /dev/video0

Linux-2.6.31_TX2440A/drivers/media/video/gspca

8. CMOS 摄像头 /dev/camera

Linux-2.6.31_TX2440A/drivers/media/video/s3c2440_camif.c

9. USB 鼠标 /dev/mouse0

Linux-2.6.31_TX2440A/drivers/usb/musb

10. U 盘 /dev/sda

Linux-2.6.31_TX2440A/drivers/usb/storage/

11. NandFlash /dev/mtdblock2

Linux-2.6.31_TX2440A/drivers/mtd/nand/

12. yaffs2 文件系统

Linux-2.6.31_TX2440A/fs/yaffs2

13. LCD 背光 /dev/TX2440-backlight

Linux-2.6.31_TX2440A/drivers/video/TX2440_backlight.c

14. 串口 /dev/s3c2410_serial0

Linux-2.6.31_TX2440A/drivers/serial/s3c2440.c

15. 其他字符设备 /dev/TX2440-XXX

Linux-2.6.31_TX2440A/drivers/char/TX2440_xxx.c

## 十六> 内核更新（2010-6-17）

1. 解决系统下 camera 模块在 LCD 上显示图像颠倒的问题

**方法：**修改 camera 驱动，在 drivers/media/video/s3c2440_ov9650.c 文件中修改第 25 行，把{0x1e, 0x0c}改成{0x1e, 0x1c}

2. 去除 LCD 左上角的闪烁光标

系统启动后在 LCD 的左上角会有光标闪烁，如果觉得光标碍事，可以把它去掉。
**方法：**修改 drivers/video/console/fbcon.c 文件
将 367 行 static void fb_flashcursor(void *private)static

1263 行　void fbcon_cursor(struct vc_data *vc, int mode)
这两个函数中的内容注释掉（保留函数名，清空函数内容）

3. 修改 LCD 背光控制，使 LCD 常亮

系统启动后会开启一个定时器，当到达定时时间后就会关闭 LCD 显示，也就相当于进入了待机模式，它可以由一些事件来唤醒。
默认的待机时间是 10 分钟，你可以自己设置这个时间，也可以关闭待机功能，使 LCD 背光常亮。
**方法：**修改 drivers/char/vt.c 文件
第 176 行　static int blankinterval = 10*60; **←10 分钟**
改成：　　static int blankinterval = 0;　　**←改为 0，或自定义时间**

4. 加入串口 2 驱动
linux 下的通用串口驱动默认只支持 UART0 和 UART1,UART2 被配置成了红外驱动，由于我们要使用 UART2 接 GPS、蓝牙等模块，我们就需要将 UART2 配置成通用串口驱动

修改 arch/arm/mach-s3c2440/mach-smdk2440.c，修改第 100 行

```
[2] = {
    .hwport         = 2,
    .flags      = 0,
    .ucon       = 0x3c5,
    .ulcon      = 0x03,             <---0x43 改为 0x03
    .ufcon      = 0x51,
}
```

修改 drivers/serial/samsung.c，加入头文件：

```
#include <mach/regs-gpio.h>
#include <linux/gpio.h>
```

修改 s3c24xx_serial_startup 函数，在第 404 行加入：

```
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_GPH(6), S3C2410_GPH6_TXD2);
            s3c2410_gpio_cfgpin(S3C2410_GPH(7), S3C2410_GPH7_RXD2);
            s3c2410_gpio_pullup(S3C2410_GPH(6), 1);
            s3c2410_gpio_pullup(S3C2410_GPH(7), 1);
        }

        rx_enabled(port) = 1;
        ..........................
}
```

重新编译内核即可使 UART2 驱动生效