S3C2410 的 Linux2.6.33 内核移植以及搭建交叉编译环境

作者：邯郸学院嵌入式专业胡峰

整理于 2010-3-13 嵌入式实验室

所需软件以及工具：wmware workstation 虚拟机 小红帽 linux gimp-2.6.4-i686-setup.exe（用于后期的 logo 画面转换） FlashFXP.exe（也可直接用虚拟机直接挂在访问） vivioflinux2.6.30.5.tar.gz（vi 移植） root1.9oflinux2.6.30.5.tar.gz（root 移植）arm-linux-gcc-3.4.1.tar.bz2 和 armv4l-tools-2.95.2.tar.bz2（交叉编译器）

步骤一：搭建交叉编译环境

1：启动虚拟机进入 linux 安装好虚拟机的共享工具 wmtool 具体做法大家应该都会 不在一一列举了。

2：arm-linux-gcc-3.4.1.tar.bz2 和 armv4l-tools-2.95.2.tar.bz2 拷贝到虚拟机中 进行安装，两个编译器其实都一样只不过版本高低不同，因为 2.6.33 是最新版本 我们不知道用何种版本的 gcc 所以先装两个，后来经过试验发现 高版本的 arm-linux-gcc-3.4.1.tar.bz2 可以编译 kernel，低版本的 armv4l-tools-2.95.2.tar.bz2 可以编译 vivi，解压当前文件夹命令$ tar jxvf arm-linux-gcc-3.4.1.tar -C /

$ tar jvxf armv4l-tools-2.95.2.tar.bz2 -C /

默认安装目录为 /opt/host/armv4l/bin ,在你的 bashrc 中添加环境变量即可：

vi /etc/bashrc

在最后一行加入 ：export PATH=$PATH:/opt/host/armv4l/bin $PATH 保存退出后执行：

source /etc/bashrc

步骤二：vivi 移植

1：修改 vivi/Makefile 文件

ARCH ? =arm

25 行：CROSS_COMPILE ? = /opt/host/armv4l/bin/armv4l-unknown-linux-

如图：

```
export KBUILD_BUILDHOST := $(SUBARCH)
ARCH            ?= arm
CROSS_COMPILE   ?=/usr/local/arm/3.4.1/bin/arm-linux-

# Architecture as present in compile.h
UTS_MACHINE     := $(ARCH)
SRCARCH         := $(ARCH)
```

2：按照自己的需求自定义 mtd 分区，修改 arch/s3c2410/smdk.c 文件

修改 NAND flash 分区如下：

mtd_partition_t default_mtd_partitions[] = {

{

```
        name:       "vivi",
        offset:     0,
        size:       0x00020000,
        flag:       0
    }, {
        name:       "param",
        offset:     0x00020000,
        size:       0x00010000,
        flag:       0
    }, {
        name:       "kernel",
        offset:     0x00030000,
        //by threewater
        size:       0x00200000,
        //size:     0x000C0000,
        flag:       0
    }, {
        name:       "root",
        offset:     0x00230000,
        size:       0x00300000,
        flag:       MF_BONFS
    }, {
        name:       "jffs2",
        offset:     0x00530000,
        size:       0x03A00000,
        flag:       MF_JFFS2
    }
};
```

3：修改 char linux_cmd[] = "noinitrd root=/dev/mtdblock3 init=/linuxrc console=ttySAC0,115200";

4、在 vivi 目录下执行 make menuconfig，选择 Load an Alternate Configuration File，输入 arch/def-configs/smdk2410 保存

5、执行 make 在 vivi 目录下将生成 vivi 的二进制可执行文件

6、下载 vivi

vivi>load flash vivi x 回车

当出现 Ready for downloading using xmodem...

Waiting...

□□□□□□□□□□□□□□□□.这时点击超级终端任务栏上“传送”下拉菜单中的“发送文件”，选择好镜像文件 vivi,协议为 Xmodem,点击“发送"通过串口 重新烧录一遍,至此你的 vivi 就烧写到 flash 里了

注释：关于超级终端的配置见最后的补充文档

7、执行 bon part 0 128k 192k 2240k 5312k:m 64704k

NANDFLASH 分区参考：

分区 起始地址 分区大小 分区作用

Part0 0x0 0x00020000 (128k) bootloader

Part1 0x00020000 0x00010000 (64k) bootloader params

Part2 0x00030000 0x00200000 (2m) linux kernel

Part3 0x00230000 0x00300000 (3m) root filesystem

Part4 0x00530000 0x03A00000 (58m) yaffs filesystem

步骤三：linux2.6.33 内核移植

1、解压 linux-2.6.33 tar.gz2 源码包

2、修改内核 mtd 分区和 bootloader（vivi）使其一致

编辑文件 arch/arm/plat-s3c24xx/common-smdk.c

```
static struct mtd_partition smdk_default_nand_part[] = {
    [0] = {
        .name   = "vivi",
        .size   = 0x00020000,
```

```
        .offset = 0,
    },
    [1] = {
        .name   = "param",
        .offset = 0x00020000,
        .size   = 0x00010000,
    },
    [2] = {
        .name   = "kernel",
        .offset = 0x00030000,
        .size   = 0x00200000,
    },
    [3] = {
        .name   = "root",
        .offset = 0x00230000,
        .size   = 0x00300000,
    },
    [4] = {
        .name   = "yaffs",
        .offset = 0x00530000,
        .size   = 0x03A00000,
    }
};
```

3、修改文件 drivers/mtd/nand/s3c2410.c

将 s3c2410_nand_init_chip 函数里的 NAND_ECC_SOFT 改为 NAND_ECC_NONE（只改搜出的第一个，第二个不用改。因为在内核中默认配置不支持 ECC 校验，当然也可以在内核配置时选中 ECC 选项）

4、解压 yaffs.tar.gz2 源码包，进入 yaffs2 文件夹，给内核打上补丁使内核支持 yaffs2（可以去网上下最新版本的）

```
# ./patch-ker.sh c /root/linux-2.6.33
```

（这是“更新”用的）

5、修改 Makefile 文件

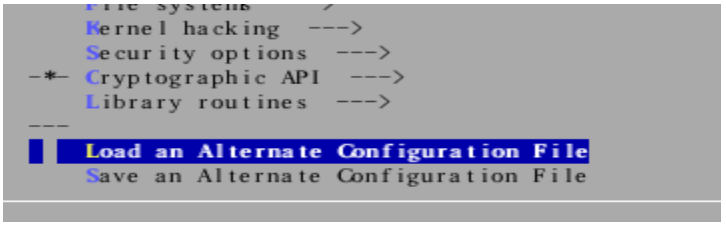
189 行 ： ARCH ?=arm

190 行 ： CROSS_COMPILE ?=/usr/local/arm/3.4.1/bin/arm-linux-

在 vi 命令行中输入：set nu 显示行号

：/ARCH 可以对 ARCH 进行变色处理方便你寻找定位

6、到 llinux-2.6.33 目录下执行命令



#cp arch/arm/configs/s3c2410_defconfig .config

这里是为了保存原来的 config 当然你也可以在

中导入配置 arch/arm/configs/s3c2410_defconfig 目录下的配置

7、执行 make menuconfig，并对选项进行配置

选项 Userspace binary formats

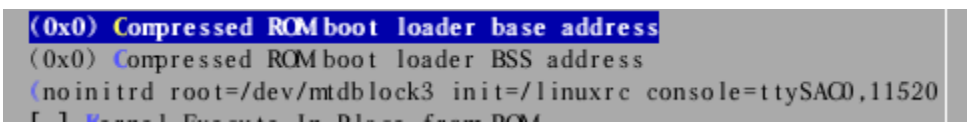
[ ]Kernel support for a.out and ECOFF binaries (BINFMT_AOUT)（去除该选项）

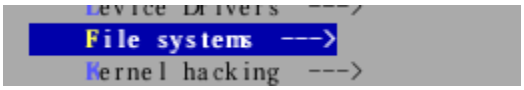
选项 Boot options（根据自己的 root 目录在那个 mtd 分区来改）

Default kernel command string

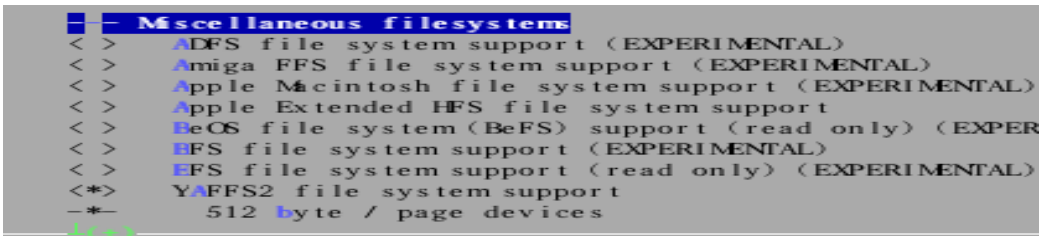
命令 param set linux_cmd_line

改成： noinitrd root=/dev/mtdblock3 init=/linuxrc console=ttySAC0,115200





选中



YAFFS2 file ststem support 以支持 yaffs

其余按默认配置

8、执行 make zImage，在 arch/arm/boot 目录下将生成一个 zImage 的文件

步骤三：cs8900 移植 sc2410

1 修改 drivers/net/arm/Makefile 文件添加：

obj-$(CONFIG_ARM_CS8900) += cs8900.o

2，修改 drivers/net/arm/Kconfig 文件添加：

config ARM_CS8900

tristate "CS8900 support"

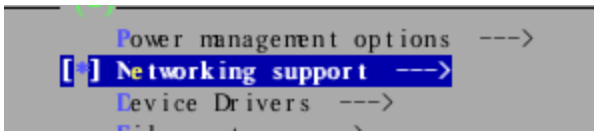
depends on NET_ETHERNET && ARM && ARCH_SMDK2410

help

Support for CS8900A chipset based Ethernet cards. If you have a network (Ethernet) card of this type, say Y and read the Ethernet-HOWTO,available from as well as .To compile this driver as a module, choose M here and read .The module will be called cs8900.o.

3，编译配置内核

make menuconfig

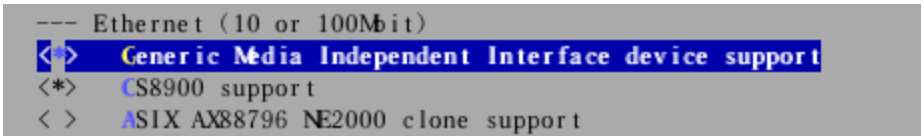


Device Drivers -->

[*] Network device support --->

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

<*> CS8900 support



4，修改 arch/arm/mach-s3c2410/mach-smdk2410.c 文件

static struct map_desc smdk2410_iodesc[] __initdata 最后添加：

{vSMDK2410_ETH_IO, pSMDK2410_ETH_IO, SZ_1M, MT_DEVICE}

```
static struct map_desc smdk2410_iodesc[] __initdata = {vSMDK2410_ETH_IO, pSMDK2410_ETH_IO, SZ_1M, MT_DEVICE
  /* nothing here yet */
};
```

5，修改 arch/arm/mach-s3c2410/include/mach/map.h 文件增加：

/* CS8900 */

#define pSMDK2410_ETH_IO __phys_to_pfn(0x19000000)

#define vSMDK2410_ETH_IO 0xE0000000

#define SMDK2410_ETH_IRQ IRQ_EINT9

```
*/
/* CS8900 */
#define pSMDK2410_ETH_IO        __phys_to_pfn(0x19000000)
#define vSMDK2410_ETH_IO        0xE0000000
#define SMDK2410_ETH_IRQ          IRQ_EINT9
#ifndef __ASM_ARCH_MAP_H
#define __ASM_ARCH_MAP_H

#include <plat/map-base.h>
```

6.把以下两个文件 cs8900.c cs8900.h 添加进 drivers/net/arm/内，然后编译内核。（后面有代码 你只需将代码粘贴复制到记事本里保存成 cs8900.c 和 cs8900.h 文件即可无需看清 所以为了篇幅考虑字体缩放的很小）

//8900.c

```
/*
 * linux/drivers/net/cs8900.c
 *
 * Author: Abraham van der Merwe <abraham at
2d3d.co.za>
 *
 * A Cirrus Logic CS8900A driver for Linux
 * based on the cs89x0 driver written by
Russell Nelson,
 * Donald Becker, and others.
 *
 * This source code is free software; you can
redistribute it and/or
 * modify it under the terms of the GNU
General Public License
 * version 2 as published by the Free
Software Foundation.
 *
 * History:
 *    22-May-2002  Initial version (Abraham
vd Merwe)
 *    30-May-2002  Added char device support
for eeprom (Frank Becker)
 *    24-Jan-2004  Fixups for 2.6 (Frank
Becker)
 *    15-July-2004 Modified for SMDK2410 (Roc
Wu pwu at jadechip.com)
 */

#define VERSION_STRING "Cirrus Logic CS8900A
driver for Linux (Modified for SMDK2410)"

/*
 * At the moment the driver does not support
memory mode operation.
 * It is trivial to implement this, but not
worth the effort.
 */

/*
 * TODO:
 *
 *   1. Sort out ethernet checksum
 *   2. If !ready in send_start(), queue
buffer and send it in interrupt handler
 *      when we receive a BufEvent with
Rdy4Tx, send it again. dangerous!
 *   3. how do we prevent interrupt handler
destroying integrity of get_stats()?
 *   4. Change reset code to check status.
 *   5. Implement set_mac_address and remove
fake mac address
 *   6. Link status detection stuff
 *   8. Write utility to write EEPROM, do
self testing, etc.
 *   9. Implement DMA routines (I need a
board w/ DMA support for that)
 *   10. Power management
 *   11. Add support for multiple ethernet
chips
 */


#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/types.h>
#include <linux/version.h>
#include <linux/errno.h>
#include <linux/init.h>
#include <linux/delay.h>
#include <linux/ioport.h>
#include <linux/pm.h>
#include <linux/irq.h>


#include <mach/hardware.h>
#include <asm/io.h>
#include <asm/uaccess.h>
#include <linux/netdevice.h>
#include <linux/etherdevice.h>
#include <linux/skbuff.h>
#include <linux/fs.h>
#include <asm/mach-types.h>

#ifdef CONFIG_ARCH_SMDK2410
#include "mach/regs-irq.h"
#include "mach/regs-mem.h"
#include "mach/regs-gpio.h"
#include "plat/common-smdk.h"
#endif

#include "cs8900.h"

//#define FULL_DUPLEX
//#define DEBUG

typedef struct {
 struct net_device_stats stats;
 u16 txlen;
 int char_devnum;

 spinlock_t lock;
} cs8900_t;

int cs8900_probe (struct net_device *dev);
static struct net_device *cs8900_dev;

/*
```

```
 * There seems to be no way to determine the
exact size of the eeprom,
 * so we use the largest size.
 * FIXME: Verify it's safe to read/write past
the end of a 64/128
 *        byte eeprom.
 *
 * Possible eeprom sizes:
 * Cx46 -  64 bytes
 * Cx56 - 128 bytes
 * Cx66 - 256 bytes
 */
#define MAX_EEPROM_SIZE  256

/*
 * I/O routines
 */

static inline u16 cs8900_read (struct
net_device *dev, u16 reg)
{
 outw (reg, dev->base_addr + PP_Address);
 return (inw (dev->base_addr + PP_Data));
}

static inline void cs8900_write (struct
net_device *dev, u16 reg, u16 value)
{
 outw (reg, dev->base_addr + PP_Address);
 outw (value, dev->base_addr + PP_Data);
}

static inline void cs8900_set (struct
net_device *dev, u16 reg, u16 value)
{
 cs8900_write (dev, reg, cs8900_read (dev, reg)
 | value);
}

static inline void cs8900_clear (struct
net_device *dev, u16 reg, u16 value)




{
 cs8900_write (dev, reg, cs8900_read (dev, reg)
 &  value);
}

static inline void cs8900_frame_read (struct
net_device *dev, struct sk_buff *skb, u16
length)
{
 insw (dev->base_addr, skb_put
(skb, length), (length + 1) / 2);
}

static inline void cs8900_frame_write (struct
net_device *dev, struct sk_buff *skb)
{
 outsw (dev->base_addr, skb->data, (skb->len +
1) / 2);
}

/*
 * Debugging functions
 */

#ifdef DEBUG
static inline int printable (int c)
{
 return ((c >= 32 && c <= 126) ||
    (c >= 174 && c <= 223) ||
    (c >= 242 && c <= 243) ||
    (c >= 252 && c <= 253));
}

static void dump16 (struct net_device
*dev, const u8 *s, size_t len)
{
 int i;
 char str[128];

 if (!len) return;

 *str = '\0';

 for (i = 0; i < len; i++) {
  if (i && !(i % 4)) strcat (str, " ");
  sprintf (str, "%s%.2x ", str, s[i]);
 }

 for ( ; i < 16; i++) {
  if (i && !(i % 4)) strcat (str, " ");
  strcat (str, "   ");
 }

 strcat (str, " ");
 for (i = 0; i < len; i++) sprintf
(str, "%s%c", str, printable (s[i]) ? s[i] :
'.');

 printk (KERN_DEBUG "%s:     %s\n", dev-
>name, str);
}

static void hexdump (struct net_device
*dev, const void *ptr, size_t size)
{
 const u8 *s = (u8 *) ptr;
 int i;
 for (i = 0; i < size / 16; i++, s += 16)
dump16 (dev, s, 16);
 dump16 (dev, s, size % 16);
}

static void dump_packet (struct net_device
*dev, struct sk_buff *skb, const char *type)
{
 printk (KERN_INFO "%s: %s %d byte
frame %.2x:%.2x:%.2x:%.2x:%.2x:%.2x
to %.2x:%.2x:%.2x:%.2x:%.2x:%.2x type %.4x\n",
    dev->name,
    type,
    skb->len,
    skb->data[0], skb->data[1], skb-
>data[2], skb->data[3], skb->data[4], skb-
>data[5],
    skb->data[6], skb->data[7], skb-
>data[8], skb->data[9], skb->data[10], skb-
>data[11],
    (skb->data[12] << 8) | skb->data[13]);
 if (skb->len < 0x100) hexdump (dev, skb-
>data, skb->len);
}


#endif /* #ifdef DEBUG */

/*
 * Driver functions
 */

static void cs8900_receive (struct net_device
*dev)
{
 cs8900_t *priv = (cs8900_t *) dev->ml_priv;
 struct sk_buff *skb;
 u16 status, length;

 status = cs8900_read (dev, PP_RxStatus);
 length = cs8900_read (dev, PP_RxLength);

 if (!(status & RxOK)) {
  priv->stats.rx_errors++;
  if ((status & (Runt | Extradata))) priv-
>stats.rx_length_errors++;
  if ((status & CRCerror)) priv-
>stats.rx_crc_errors++;
  return;
 }

 if ((skb = dev_alloc_skb (length + 4)) ==
NULL) {
  priv->stats.rx_dropped++;
  return;
 }

 skb->dev = dev;
 skb_reserve (skb, 2);

 cs8900_frame_read (dev, skb, length);

#ifdef FULL_DUPLEX
 dump_packet (dev, skb, "recv");
#endif /* #ifdef FULL_DUPLEX */

 skb->protocol = eth_type_trans (skb, dev);

 netif_rx (skb);
 dev->last_rx = jiffies;

 priv->stats.rx_packets++;
 priv->stats.rx_bytes += length;
}

static int cs8900_send_start (struct sk_buff
*skb, struct net_device *dev)
{
 cs8900_t *priv = (cs8900_t *) dev->ml_priv;
 u16 status;

 spin_lock_irq(&priv->lock);
 netif_stop_queue (dev);

 cs8900_write  (dev, PP_TxCMD, TxStart  (After5));
 cs8900_write (dev, PP_TxLength, skb->len);

 status = cs8900_read (dev, PP_BusST);

 if ((status & TxBidErr)) {
  spin_unlock_irq(&priv->lock);
  printk (KERN_WARNING "%s: Invalid frame
size %d\n", dev->name, skb->len);
  priv->stats.tx_errors++;
  priv->stats.tx_aborted_errors++;
  priv->txlen = 0;
  return (1);
 }

 if (!(status & Rdy4TxNOW)) {
  spin_unlock_irq(&priv->lock);
  printk (KERN_WARNING "%s: Transmit buffer
not free!\n", dev->name);
  priv->stats.tx_errors++;
  priv->txlen = 0;
  /* FIXME: store skb and send it in
interrupt handler */
  return (1);
 }

 cs8900_frame_write (dev, skb);
 spin_unlock_irq(&priv->lock);

#ifdef DEBUG
 dump_packet (dev, skb, "send");
#endif /* #ifdef DEBUG */

 dev->trans_start = jiffies;

 dev_kfree_skb (skb);

 priv->txlen = skb->len;

 return (0);
}

static irqreturn_t cs8900_interrupt (int
irq, void *id)
{
 struct net_device *dev = (struct net_device
*) id;
 cs8900_t *priv;
 volatile u16 status;
 irqreturn_t handled = 0;

 if (dev->ml_priv == NULL) {
  printk (KERN_WARNING "%s: irq %d for
unknown device.\n", dev->name, irq);
  return 0;
 }

 priv = (cs8900_t *) dev->ml_priv;

 while ((status = cs8900_read (dev, PP_ISQ)))
{
  handled = 1;
  switch (RegNum (status)) {
  case RxEvent:
   cs8900_receive (dev);
   break;

  case TxEvent:
   priv->stats.collisions += ColCount
(cs8900_read (dev, PP_TxCOL));
   if (!(RegContent (status) & TxOK)) {
    priv->stats.tx_errors++;
    if ((RegContent (status) & Out_of_window))
priv->stats.tx_window_errors++;
    if ((RegContent (status) & Jabber)) priv-
>stats.tx_aborted_errors++;
    break;
   } else if (priv->txlen) {
    priv->stats.tx_packets++;
    priv->stats.tx_bytes += priv->txlen;
   }
   priv->txlen = 0;
   netif_wake_queue (dev);
   break;

  case BufEvent:
   if ((RegContent (status) & RxMiss)) {
    u16 missed = MissCount (cs8900_read
(dev, PP_RxMISS));
    priv->stats.rx_errors += missed;
    priv->stats.rx_missed_errors += missed;
   }
   if ((RegContent (status) & TxUnderrun)) {
    priv->stats.tx_errors++;
    priv->stats.tx_fifo_errors++;

    priv->txlen = 0;
    netif_wake_queue (dev);
   }
   /* FIXME: if Rdy4Tx, transmit last sent
packet (if any) */
   break;

  case TxCOL:
   priv->stats.collisions += ColCount
(cs8900_read (dev, PP_TxCOL));
   break;

  case RxMISS:
   status = MissCount (cs8900_read
(dev, PP_RxMISS));
   priv->stats.rx_errors += status;
   priv->stats.rx_missed_errors += status;
   break;
  }
 }
 return IRQ_RETVAL(handled);
}

static void cs8900_transmit_timeout (struct
net_device *dev)
{
 cs8900_t *priv = (cs8900_t *) dev->ml_priv;
 priv->stats.tx_errors++;
 priv->stats.tx_heartbeat_errors++;
 priv->txlen = 0;
 netif_wake_queue (dev);
}

static int cs8900_start (struct net_device
*dev)
{
 int result;
    printk("cs8900_start \n");
 set_irq_type(dev->irq, (1<<1)|(1<<0));
    printk("enable the ethernet controlle\n");
 /* enable the ethernet controller */
 cs8900_set (dev, PP_RxCFG, RxOKiE | BufferCRC
| CRCerroriE | RuntiE | ExtradataiE);
 cs8900_set (dev, PP_RxCTL, RxOKA | IndividualA
| BroadcastA);
 cs8900_set (dev, PP_TxCFG, TxOKiE |
Out_of_windowiE | JabberiE);
 cs8900_set (dev, PP_BufCFG, Rdy4TxiE |
RxMissiE | TxUnderruniE | TxColOvfiE |
MissOvfloiE);
 cs8900_set (dev, PP_LineCTL, SerRxON |
SerTxON);
 cs8900_set (dev, PP_BusCTL, EnableRQ);

#ifdef FULL_DUPLEX
 cs8900_set (dev, PP_TestCTL, FDX);
#endif /* #ifdef FULL_DUPLEX */
 udelay(200);
 /* install interrupt handler */
printk("request_irq \n");
 if ((result = request_irq (dev->irq,
&cs8900_interrupt, 0, dev->name, dev)) < 0) {
  printk (KERN_ERR "%s: could not register
interrupt %d\n", dev->name, dev->irq);
  return (result);
 }

 /* start the queue */
 netif_start_queue (dev);

 return (0);
}

static int cs8900_stop (struct net_device
*dev)
{
 /* disable ethernet controller */
 cs8900_write (dev, PP_BusCTL, 0);
 cs8900_write (dev, PP_TestCTL, 0);
 cs8900_write (dev, PP_SelfCTL, 0);
 cs8900_write (dev, PP_LineCTL, 0);
 cs8900_write (dev, PP_BufCFG, 0);
 cs8900_write (dev, PP_TxCFG, 0);
 cs8900_write (dev, PP_RxCTL, 0);
 cs8900_write (dev, PP_RxCFG, 0);

 /* uninstall interrupt handler */
 free_irq (dev->irq, dev);

 /* stop the queue */
 netif_stop_queue (dev);

 return (0);
}

static struct net_device_stats
*cs8900_get_stats (struct net_device *dev)
{
 cs8900_t *priv = (cs8900_t *) dev->ml_priv;
 return (&priv->stats);
}

static void cs8900_set_receive_mode (struct
net_device *dev)
{
 if ((dev->flags & IFF_PROMISC))
  cs8900_set (dev, PP_RxCTL, PromiscuousA);
 else
  cs8900_clear (dev, PP_RxCTL, PromiscuousA);

 if ((dev->flags & IFF_ALLMULTI) && dev-
>mc_list)
  cs8900_set (dev, PP_RxCTL, MulticastA);
 else
  cs8900_clear (dev, PP_RxCTL, MulticastA);
}

/*
 * Driver initialization routines
 */

int __init cs8900_probe (struct net_device
*dev)
{
 static cs8900_t priv;
 int i, result;
 u16 value;

 //unsigned long extint1;
 //extint1 = __raw_readl(S3C2410_EXTINT1);
 //extint1 &= (7<<4);
 //extint1 |= (4<<4);
 //__raw_writel(extint1, S3C2410_EXTINT1);

 printk (VERSION_STRING"\n");

 memset (&priv, 0, sizeof (cs8900_t));

__raw_writel((__raw_readl(S3C2410_GPGCON)&~(0
x3<<2))|(0x2<<2),S3C2410_GPGCON);
 __raw_writel((__raw_readl(S3C2410_EXTINT1)&~(
0x7<<4))|(0x4<<4),S3C2410_EXTINT1);
__raw_writel(0x221d110,S3C2410_BWSCON);
__raw_writel(0x1ff7c,S3C2410_BANKCON3);

#if defined(CONFIG_ARCH_SMDK2410)
 dev->dev_addr[0] = 0x00;
 dev->dev_addr[1] = 0x00;
 dev->dev_addr[2] = 0x3e;
 dev->dev_addr[3] = 0x26;
 dev->dev_addr[4] = 0x0a;
 dev->dev_addr[5] = 0x00;

#endif

 dev->if_port   = IF_PORT_10BASET;
 dev->ml_priv       = (void *) &priv;

 spin_lock_init(&priv.lock);

 //SET_MODULE_OWNER (dev);

#if defined(CONFIG_ARCH_SMDK2410)
 dev->base_addr = (0xE0000000 + 0x300);
 dev->irq = IRQ_EINT9;
#endif /* #if defined(CONFIG_ARCH_SMDK2410)
*/

 if ((result = check_mem_region (dev-
>base_addr, 25)) {
  printk (KERN_ERR "%s: can't get I/O port
address 0x%lx\n", dev->name, dev->base_addr);
  return (result);
 }
 request_mem_region (dev->base_addr, 16, dev-
>name);

     /* verify EISA registration number for
Cirrus Logic */
 if ((value = cs8900_read
(dev, PP_ProductID)) != EISA_REG_CODE) {
  printk (KERN_ERR "%s: incorrect signature
0x%.4x\n", dev->name, value);
  return (-ENXIO);
 }

 /* verify chip version */
 value = cs8900_read (dev, PP_ProductID + 2);
 if (VERSION (value) != CS8900A) {
  printk (KERN_ERR "%s: unknown chip version
0x%.8x\n", dev->name, VERSION (value));
  return (-ENXIO);
 }
 /* setup interrupt number */
 cs8900_write (dev, PP_IntNum, 0);


 printk (KERN_INFO "%s: CS8900A rev %c
at %#lx irq %d",
  dev->name, 'B' + REVISION (value) - REV_B,
dev->base_addr, dev->irq);

 for (i = 0; i < ETH_ALEN; i += 2)
 cs8900_write (dev, PP_IA + i, dev->dev_addr[i]
| (dev->dev_addr[i + 1] << 8));

 printk (", addr:");
 for (i = 0; i < ETH_ALEN; i += 2)
 {
  u16 mac = cs8900_read (dev, PP_IA + i);
  printk ("%c%02X:%2X", (i==0)?' ':':', mac &
0xff, (mac >> 8));
 }
 printk ("\n");

 return (0);
}

static int __init cs8900_init (void)
{
struct net_device *ndev;
ndev = alloc_etherdev(sizeof (cs8900_t));
const static struct net_device_ops
new_netdev_ops ={
        .ndo_init               =
cs8900_probe,
        .ndo_open               =
cs8900_start,
        .ndo_stop               =
```

```
cs8900_stop,
                .ndo_start_xmit         =
cs8900_send_start,
                .ndo_get_stats          =
cs8900_get_stats,
                .ndo_set_multicast_list =
cs8900_set_receive_mode,
                .ndo_tx_timeout         =
cs8900_transmit_timeout,
};
        ndev->netdev_ops = alloc_etherdev(sizeof
(struct net_device_ops));
        if (!ndev) {
                printk("%s: could not
allocate device.", "cs8900");
                return -ENOMEM;
        }
        ndev->netdev_ops = &new_netdev_ops;
        cs8900_dev = ndev;
        ether_setup (ndev);
```

```
/*              ndev->open              =
cs8900_start;
        ndev->stop              =
cs8900_stop;
        ndev->hard_start_xmit   =
cs8900_send_start;
        ndev->get_stats         =
cs8900_get_stats;
        ndev->set_multicast_list =
cs8900_set_receive_mode;
        ndev->tx_timeout        =
cs8900_transmit_timeout; */
        ndev->watchdog_timeo    = HZ;
        return (register_netdev
(cs8900_dev));
}

static void __exit cs8900_cleanup (void)
{
```

```
 cs8900_t *priv = (cs8900_t *) cs8900_dev-
>ml_priv;
 if( priv->char_devnum)
 {
  unregister_chrdev(priv-
>char_devnum, "cs8900_eeprom");
 }
 release_mem_region (cs8900_dev-
>base_addr,16);
 unregister_netdev (cs8900_dev);
}

MODULE_AUTHOR ("Abraham van der Merwe
<abraham at 2d3d.co.za>");
MODULE_DESCRIPTION (VERSION_STRING);
MODULE_LICENSE ("GPL");

module_init (cs8900_init);
module_exit (cs8900_cleanup);
```

cs8900.c文件代码完毕

```
//8900.h

#ifndef CS8900_H
#define CS8900_H

/*
 * linux/drivers/net/cs8900.h
 *
 * Author: Abraham van der Merwe <abraham at
2d3d.co.za>
 *
 * A Cirrus Logic CS8900A driver for Linux




 * based on the cs89x0 driver written by
Russell Nelson,
 * Donald Becker, and others.




 *
 * This source code is free software; you can
redistribute it and/or




 * modify it under the terms of the GNU


General Public License
 * version 2 as published by the Free
Software Foundation.
 */

/*
 * Ports



 */

#define PP_Address  0x0a /* PacketPage
Pointer Port (Section 4.10.10) */
#define PP_Data     0x0c /* PacketPage Data
Port (Section 4.10.10) */

/*
 * Registers
 */

#define PP_ProductID    0x0000 /* Section 4.3.1
Product Identification Code */



#define PP_MemBase      0x002c /* Section 4.9.2
Memory Base Address Register */

#define PP_IntNum       0x0022 /* Section 3.2.3
Interrupt Number */
#define PP_EEPROMCommand 0x0040 /* Section
4.3.11  EEPROM Command */
#define PP_EEPROMData   0x0042 /* Section
4.3.12  EEPROM Data */
#define PP_RxCFG        0x0102 /* Section 4.4.6
Receiver Configuration */
#define PP_RxCTL        0x0104 /* Section 4.4.8
Receiver Control */
#define PP_TxCFG        0x0106 /* Section 4.4.9
Transmit Configuration */
#define PP_BufCFG       0x010a /* Section 4.4.12
Buffer Configuration */
#define PP_LineCTL      0x0112 /* Section 4.4.16
Line Control */
#define PP_SelfCTL      0x0114 /* Section 4.4.18
Self Control */
#define PP_BusCTL       0x0116 /* Section 4.4.20
Bus Control */
#define PP_TestCTL      0x0118 /* Section 4.4.22
Test Control */
#define PP_ISQ          0x0120 /* Section 4.4.5
Interrupt Status Queue */
```

```
#define PP_TxEvent      0x0128 /* Section 4.4.10
Transmitter Event */
#define PP_BufEvent     0x012c /* Section
4.4.13  Buffer Event */
#define PP_RxMISS       0x0130 /* Section 4.4.14
Receiver Miss Counter */
#define PP_TxCOL        0x0132 /* Section 4.4.15
Transmit Collision Counter */
#define PP_SelfST       0x0136 /* Section 4.4.19
Self Status */
#define PP_BusST        0x0138 /* Section 4.4.21
Bus Status */
#define PP_TxCMD        0x0144 /* Section 4.4.11
Transmit Command */
#define PP_TxLength     0x0146 /* Section 4.5.2
Transmit Length */
#define PP_LA           0x0158 /* Section 4.6.2
Individual Address (IEEE Address) */
#define PP_RxStatus     0x0400 /* Section 4.7.1
Receive Status */
#define PP_RxLength     0x0402 /* Section 4.7.1
Receive Length (in bytes) */
#define PP_RxFrame      0x0404 /* Section 4.7.2
Receive Frame Location */
#define PP_TxFrame      0x0a00 /* Section 4.7.2
Transmit Frame Location */

/*
 * Values
 */

/* PP_IntNum */
#define INTRQ0    0x0000
#define INTRQ1    0x0001
#define INTRQ2    0x0002
#define INTRQ3    0x0003

/* PP_ProductID */
#define EISA_REG_CODE 0x630e

#define REVISION(x)    (((x) & 0x1f00) >> 8)
#define VERSION(x)     ((x) & 0x1f00)

#define CS8900A     0x0000
#define REV_B    7
#define REV_C    8
#define REV_D    9

/* PP_RxCFG */
#define Skip_1    0x0040
#define StreamE   0x0080
#define RxOKiE    0x0100
#define RxDMAonly 0x0200
#define AutoRxDMAE  0x0400
#define BufferCRC   0x0800
#define CRCerroriE  0x1000
#define RuntiE    0x2000
#define ExtradataiE  0x4000

/* PP_RxCTL */
#define IAHashA     0x0040
#define PromiscuousA 0x0080
#define RxOKA       0x0100
#define MulticastA  0x0200
#define IndividualA  0x0400
#define BroadcastA  0x0800
#define CRCerrorA   0x1000
#define RuntA       0x2000
#define ExtradataA  0x4000

/* PP_TxCFG */
#define Loss_of_CRSiE 0x0040
#define SQEerroriE  0x0080
#define TxOKiE      0x0100
#define Out_of_windowiE 0x0200
#define JabberiE    0x0400
#define AnycolliE   0x0800
#define T16colliE   0x8000

/* PP_BufCFG */
#define SWint_I     0x0040
#define RxDMAiE     0x0080
#define Rdy4TxiE    0x0100
#define TxUnderruniE 0x0200
#define RxMissiE    0x0400
#define Rx128iE     0x0800
#define TxColOvfiE  0x1000
#define MissOvfloiE 0x2000
#define RxDestiE    0x8000

/* PP_LineCTL */
#define SerRxON     0x0040
#define SerTxON     0x0080
#define AUIonly     0x0100
#define AutoAUI_10BT 0x0200
#define ModBackoffE 0x0800
#define PolarityDis  0x1000
#define L2_partDetDis 0x2000
#define LoRxSquelch  0x4000

/* PP_SelfCTL */
#define RESET     0x0040
#define SWSuspend   0x0100
#define HWSleepE  0x0200
```

```
#define HWStandbyE  0x0400
#define HCIE    0x1000
#define HCIE    0x2000
#define HCBQ    0x4000
#define HCB1    0x8000

/* PP_BusCTL */
#define ResetRxDMA  0x0040
#define DMAextend   0x0100
#define UseSA       0x0200
#define MemoryE     0x0400
#define DMABurst    0x0800
#define IOCHRDYE    0x1000
#define RxDMAsize   0x2000
#define EnableIRQ   0x8000

/* PP_TestCTL */
#define DisableLT   0x0080
#define ENDECloop   0x0200
#define AUIloop     0x0400
#define DisableBackoff 0x0800
#define FDX         0x4000

/* PP_ISQ */
#define RegNum(x) ((x) & 0x3f)
#define RegContent(x)  ((x) & 0x3d)

#define RxEvent     0x0004
#define TxEvent     0x0008
#define BufEvent    0x000c
#define RxMISS      0x0010
#define TxCOL       0x0012

/* PP_RxStatus */
#define IAHash    0x0040
#define Dribblebits  0x0080
#define RxOK      0x0100
#define Hashed    0x0200
#define IndividualAdr 0x0400
#define Broadcast   0x0800
#define CRCerror    0x1000
#define Runt      0x2000
#define Extradata   0x4000

#define HashTableIndex(x)  ((x) >> 0xa)

/* PP_TxCMD */
#define After5    0
#define After381    1
#define After1021   2
#define AfterAll    3
#define TxStart(x)  ((x) << 6)

#define Force     0x0100
#define Onecoll     0x0200
#define InhibitCRC  0x1000
#define TxPadDis    0x2000

/* PP_BusST */
#define TxBidErr    0x0080
#define Rdy4TxNOW   0x0100

/* PP_TxEvent */
#define Loss_of_CRS   0x0040
#define SQEerror    0x0080
#define TxOK      0x0100
#define Out_of_window   0x0200
#define Jabber      0x0400
#define T16coll     0x8000

#define TX_collisions(x)  (((x) >> 0xb) &
0x8000)

/* PP_BufEvent */
#define SWint     0x0040
#define RxDMAFrame  0x0080
#define Rdy4Tx    0x0100
#define TxUnderrun  0x0200
#define RxMiss    0x0400
#define Rx128     0x0800
#define RxDest    0x8000

/* PP_RxMISS */
#define MissCount(x)  ((x) >> 6)

/* PP_TxCOL */
#define ColCount(x)  ((x) >> 6)

/* PP_SelfST */
#define I3VActive   0x0040
#define INITD     0x0080
#define SIBSY     0x0100
#define EEPROMpresent 0x0200
#define EEPROMOK    0x0400
#define ELpresent   0x0800
#define EEsize    0x1000

/* PP_EEPROMCommand */
#define EEWriteEnable   0x00F0
#define EEWriteDisable  0x0000
#define EEWriteRegister 0x0100
#define EEReadRegister 0x0200
#define EEEraseRegister 0x0300
#define ELSEL     0x0400

#endif /* #ifndef CS8900_H */
```

//Cs8900.h 文件代码完毕

步骤四：lcd 移植

一、修改/arch/arm/mach-s3c2410/mach-smdk2410.c 文件

1、添加头文件

#include <mach/fb.h>

#include <mach/regs-lcd.h>

2、添加初始化 s3c2410的 LCD 控制器时所需的参数（copy from /arch/arm/mach-s3c2410/mach-qt2410.c，稍作修改）。

```
static struct s3c2410fb_display qt2410_lcd_cfg[] __initdata = {
    {
        /* Configuration for 640x480 SHARP LQ080V3DG01 */
        .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/4 */
        .xres       = 320,
        .yres       = 240,
        .bpp        = 16,
        .left_margin    = 13,
        .right_margin   = 8,
        .hsync_len  = 4,
        .upper_margin   = 2,
        .lower_margin   = 7,
        .vsync_len  = 4,
    },
};                                  （这些参数都是自己修改过的，符合要求的）

static struct s3c2410fb_mach_info qt2410_fb_info __initdata = {
    .displays   = qt2410_lcd_cfg,
    .num_displays   = ARRAY_SIZE(qt2410_lcd_cfg),
    .default_display = 0,

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

3、添加 LCD 控制器的寄存器参数设置函数

```
static void __init smdk2410_init(void)
{
```

```
    s3c24xx_fb_set_platdata(&qt2410_fb_info);//加入液晶的配置文件
    set_s3c2410ts_info(&MY2410_ts_cfg);//加入触摸屏的配置文件
    s3c_i2c0_set_platdata(NULL);
    platform_add_devices(smdk2410_devices, ARRAY_SIZE(smdk2410_devices));
    smdk_machine_init();
}
```

4、打开 LCD 的初始化。

```
static struct platform_device *smdk2410_devices[] __initdata = {
  &s3c_device_usb,
  &s3c_device_lcd,
  &s3c_device_wdt,
  &s3c_device_i2c0,
  &s3c_device_iis,
};
```

5、去除１０分钟左右自动关闭显示的程序（实验时可以这么做，我猜应该是节能作用，并未深究。）。

copy from《2410 lcd(ltv350)驱动在２.６.１４下的移植经历》：

“按 seigpao 大侠的做法，注释掉 drivers\char\vt.c 的 blank_screen_t(unsigned long dummy)的函数内容，否则，lcd 会在１０分钟左右关掉显示。”

6、二、配置内核

```
Device Drivers  --->
  Graphics support  --->
    Display device support  --->
      <*> Display panel/monitor support
    <*> Support for frame buffer devices
    <*> S3C2410 LCD framebuffer support
    Console display driver support  --->
      <*> Framebuffer Console support
      [*]   Framebuffer Console Rotation
      [*] Select compiled-in fonts
      [*]   VGA 8x8 font
      [*]   VGA 8x16 font
      [*]   Mini 4x6 font
      [*] Sparc console 8x16 font
    [*] Bootup logo  --->
      --- Bootup logo
      [*]   Standard 224-color Linux logo
```

7、开机画面的选择

这一部你可以有多种选择首先我先介绍一种简单的做法 即在 windows 下将开机画面转化好

转换：选择合适的图片转换成 pnm 转换成 224 色 像素大小 320*240 小了显示屏可以显示 大了显示屏就会变成黑色无法显示

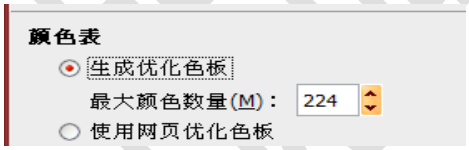
下面我现在介绍一种小工具 gimp-2.6.4-i686-setup.exe 当然你也可以用其他的 potoshop 美图秀秀等做图工具

下面现在 windows 下安装 gimp-2.6.4-i686-setup.exe

启动 

打开一个图片

在选择模式——索引进入索引颜色转换



另存图像格式选择 ppm 格式的文件 文件名为 ogo_linux_clut224.ppm 替换掉 drivers/video/logo 中的这个文件

如果你的 linux 装有 fblogo 工具、linux 下的图像编辑工具 GIMP 你也可以进行如下操作

首先把要开机 Logo 图片(png 格式)放在 linux2.6.30.5 文件中的 drivers/video/logo 中，终端选择进入 drivers/video/logo 目录，进行以下操作：

(假设开机图片名为 linux.png)

pngtopnm linux.png > linuxlogo.pnm

pnmquant 224 linuxlogo.pnm > linuxlogo224.pnm

pnmtoplainpnm linuxlogo224.pnm > linuxlogo224.ppm

mv linuxlogo224.ppm logo_linux_clut224.ppm //替换原来的启动文件

注意事项：备份原来的

步骤五：触摸屏移植

（1）s3c2410-ts.c 添加

#include <mach/gpio-nfs.h>

#define S3C2410_GPG12 S3C2410_GPIONO(S3C2410_GPIO_BANKG, 12)

#define S3C2410_GPG13 S3C2410_GPIONO(S3C2410_GPIO_BANKG, 13)

#define S3C2410_GPG14 S3C2410_GPIONO(S3C2410_GPIO_BANKG, 14)

#define S3C2410_GPG15 S3C2410_GPIONO(S3C2410_GPIO_BANKG, 15)

（2）#include 修改 arch/arm/mach-s3c2410/mach-smdk2410.c，在文件中添加 static struct s3c2410_ts_mach_info s3c2410_tscfg __initdata = {

.delay = 10000,

.presc = 49,

.oversampling_shift = 2,

};

（3）修改 static struct platform_device *smdk2410_devices[] __initdata = { &s3c_device_usb,

&s3c_device_lcd,

&s3c_device_wdt,

&s3c_device_i2c,

&s3c_device_iis,

&s3c_device_ts,

//添加此行 };

（4）在 static void __init smdk2410_init(void)中加入： s3c24xx_ts_set_platdata(&s3c2410_tscfg);

（5）在 driver/input/touchscreen/下把 s3c2410-ts.c 文件拷贝过去

5.内核配置

Device drivers

Input device support

<*>Touchscreens

<*>Samsung S3C2410 touchscreen input driver

[]Samsung s3c2410 touchscreen debug message

6：测试是否成功：

内核启动信息里有

s3c2410 TouchScreen successfully loaded

input: s3c2410 TouchScreen as /class/input/input0

dev 目录下多了一个主设备号为 13 的 event0 设备

cat /sys/class/input/input0/name 结果为

s3c2410 TouchScreen

说明移植成功进一步测试是否能使用触摸屏

你可以在 yaffs 下输入以下命令

Cat /dev/event0

点击触摸屏看是否有反应有的话 说明已经成功能够使用

最后一步 root 移植：我已经在 s3c2410 里写进了一个程序 要让实验箱已启动就开始执行那条指令

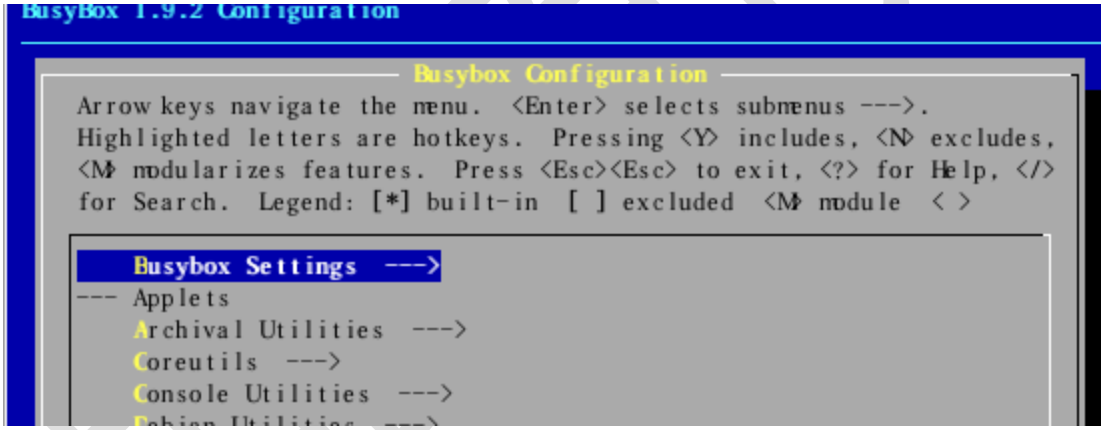
1.下载 busybox

http://www.busybox.net/downloads/busybox-1.9.2.tar.bz2

tar jxvf busybox-1.9.2.tar.bz2
cd busybox-1.9.2
vi Makefile

```
ARCH            ?= arm
CROSS_COMPILE   ?=/usr/local/arm/3.4.1/bin/arm-linux-
```

#Make menuconfig
做如下定制：



修改文件 applets/applets.c 第 21 行, 将 #error Aborting compilation.注释掉
否则出现错误：

applets/applets.c:21:2: error: #error Aborting compilation.
#make
...

#make CONFIG_PREFIX=/root_01 install
busybox 就被安装到根目录/root_01 下了(/root_01 下生成 bin linuxrc sbin usr)

/root_01 是自己在主机根目录下建的一空文件。 

2.在/root_01 中

创建文件 dev、/etc、/home、/home/usr、/lib、/mnt、/var、/tmp、/proc、/opt、/www

#mkdir dev etc home lib mnt var tmp proc sys opt www
将库文件拷到/lib 下,这里是在 3.4.1 的目录里的库，别的编译器库行不行没试过。

#cd /usr/local/arm/3.4.1/arm-linux/lib
#cp *.so* /root_01/lib -d

注意这两装载器对应的库文件（ld.so 和 ld-linux.so ）没有到主机里的 lib 找。

3.创建 etc 目录下文件

在 root_01/etc 目录下创建一个 inittab 文件

内容如下：

::sysinit:/etc/init.d/rcS

::respawn:-/bin/sh

tty2::askfirst:-/bin/sh

::restart:/sbin/init

::ctrlaltdel:/sbin/reboot

::shutdown:/bin/umount -a -r

::shutdown:/sbin/swapoff -a

~创建 etc/mdev.conf 文件，内容为空

4.同样的方法创建 etc/init.d/rcS 文件：

#!/bin/sh

echo "network interface"

ifconfig eth0 192.168.1.17

mount -a

echo "----------mount all"

mkdir /dev/pts

mount -t devpts devpts /dev/pts

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

mdev -s

mount -t yaffs /dev/mtdblock4 /mnt/yaffs

echo "**"

echo " hufeng root 2010.3.14 "

echo " Love Linux ! ! successfully! "

echo "***"

/bin/mkdir /dev/input

ln -s /dev/event0 /dev/input/event0

exec /mnt/yaffs/touchscreen/qtopia.sh

最后还要改变它的属性使它能够执行。
chmod +x etc/init.d/rcS

5.创建 etc/fstab 文件：

proc /proc proc defaults 0 0
tmpfs /tmp tmpfs defaults 0 0
sysfs /sys sysfs defaults 0 0
tmpfs /dev tmpfs defaults 0 0

6.在/root_01/dev 建立节点文件:

#mknod console c 5 1
#mknod null c 1 3

7．因为我要执行的程序内部需要一个跳转指令

```
export set TSLIB_TSDEVICE=/dev
cd /usr/qpe/bin/
./ts_calibrate
clear
```

ts——calibrate 并不在 usr/gpe/bin 文件下而在/mnt/yaffs/Qtopia/bin 目录下 所以我在虚拟机编辑根目录时先进行一下目录链接就是咱们所谓的快捷方式

```
BC root]# ln -s /mnt/yaffs/Qtopia/  /root_01/usr/qpe
```

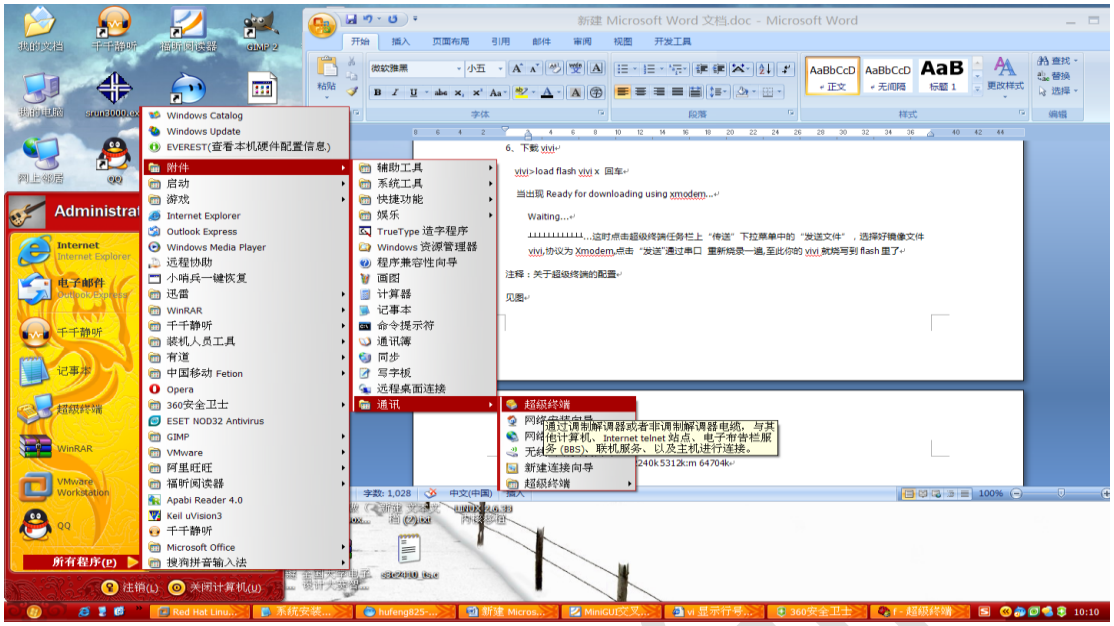
8.下载 cramfs-1.1.tar.gz 解压后在其顶层 make

得到 mkcramfs 与 root_01 放在一起。

执行

#mkcramfs root_01 myroot01.cramfs

生成根文件 myroot01.cramfs。



注意安装串口时一定要把虚拟机的串口关闭掉

以上的为最新 linux 内核 2.6.33 移植到 s3c2410 的资料 整理的可能还不详细或者有错误，欢迎大家批评指正，我的 qq：550230997 如有问题请及时反映

下面转载一篇关于 logo 的替代方法的具体解析

linux 启动 logo 修改

修改目标：用自定义的 ppm 图片替代/drivers/video/logo/logo_linux_clut224.ppm
同时删除 logo_linux_clut224.c logo_linux_clut224.o 文件这样才能使编译的使用 logo_linux_clut224.ppm 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！
更改的最简单的方法就是上面的了。下面介绍下相应的知识（以下来自网络）：
另外，相应的知识需要了解一下，2410 开发板-启动图标制做
1.方案分析
ARM-Linux 启动时会先在 LCD 左上角显示一个 ARMLINUX 的小企鹅图案,也就是我们所说的 bootlogo。
首先分析一下 Logo 的显示代码：
不难发现 Logo 显示是在加载显示驱动 fb 后由 kernel/drivers/video/fbcon.c 中的 fbcon_show_logo 函数来完成的。
fbcon_show_logo 函数中有很多兼容性代码，不是都会用到的！我们用的是 SHARP(夏普)的 TFT 真彩屏.
再仔细看一下，这里用到的显示数据均来自 kernel/include/linux/linux_logo.h。这里同样也有一些兼容性的数据，其中对我们有用的是：

linux_logo_red[] —— 调色板中的红色分量
linux_logo_green[] —— 调色板中的绿色分量
linux_logo_blue[] —— 调色板中的蓝色分量
linux_logo[] —— Logo 图案的点阵

以上数据都是不需要你修改的 在 fbcon_show_logo 里只做了 256 色的显示，所以要先设置一个 RGB 调色板。这里的调色板是从 0x20 色开始编的，所以实际好象只有最多 224 色的显示。linux_logo.h 里默认是做了一个 214 级的灰度调色板，如果想要更多色彩的话可以自己修改（注意调色板的颜色不能超过 224 种）
再看 linux_logo[]，这是 Logo 图案的点阵。因为是 256 色的图，所以每点为一个 Byte。Logo 图案是 80*80 大小，所以这里总共有 6400 Byte。一共是 800 行，每行 8 Byte 即 8 Point，每 10 行数据对应 Logo 图案中一行。可以先自己试试放一些简单图形进去显示看看。（同样注意颜色要从 0x20 开始，0-0x1f 的 32 种颜色是不确定的）。

以上分析是针对系统原版内核 256 色 80x80 的启动图标的说明。而我们要通过修改或替代原内核文件，来实现我们制做公司的的启动图标，要求如下：
1.开发板起动以后以全屏显示即 240x320 模式显示
2.图像以 224 色彩显示
2.方案设计
当然你不会只想显示一些简单图形，而是希望能把自己做的图片放上去。建议使用 PhotoShop8.0 这里提供三个方法加工图片：
1)先搞一个 256 色调色板，前面 32 个颜色空着，把后面 224 种颜色设一下吧。注意要尽量把颜色选全一些，要能覆盖到整个颜色区域上。这样就有一个通用的调色板，可适用所有图片，但颜色的真实性稍差。然后把自己的图片打开，先将颜色设置到 RGB 空间里，然后裁剪/缩放到 240x320 象数的尺寸，再将颜色优化到先前定义好的调色板里，图片就加工好了。
2)不用通用调色板，打开自己的图片，先将颜色设置到 RGB 空间，裁剪/缩放到 240x320 象素的尺寸，然后指定优化为 224 色图象。这样图片就加工好了，但其调色板就是专用的，不适合用到其他图片里。
3)GIMP+FBLOGO GIMP 为 LINUX 下一个绘图程序，我们使用它打开图片后在菜单栏先择： 图片->模式->索引颜色，然后通过工具选项将图片大小调整到 240x320 象数大小，保存为.png 格式。在使 FBLOGO 工具将其转换成 linux_logo.h 文件。

使用前两种方法加工图片，还需要把数据放到我们的 linux_logo.h 里。调色板可以直接保存为文件，然后可以通过自制的小程序读出来就可以了。图片的点阵比较难取，当然也可以自己做个程序去取屏幕上的点，不过比较困难。而后一种方法使用 fblogo 工具将用 GIMP 转换的图片文件自动生成 linux_logo.h 文件。
3.最终实施方案
通过以上内容分析,我们确定使用第三种方法 GIMP 加 fblogo 工具的方案进行实施，以下我们制做的流程及其方法：
软件环境：linux 虚拟机、fblogo 工具、linux 下的图像编辑工具 GIMP
硬件环境：X86PC 机一台、优龙 ST2410 开发板一个、交叉网线一条。
1.调试并编译开发板所相对应的内核包，调试通过后保留.config 配制文件，准备编译 bootlogo 时使用。
2.在 linux 上解压 fblogo_0.52.tgz 包，并进行软件安装，生成 fblogo 二进制可执行文件。
3.进入 linux 的 kde 图形界面，在终端下使用 gimp 命令运行 gimp 工具。
4.打开公司的 logo.jpg 图像文件，依次选择 图像->模式->索引颜色、并将其图片大小调至 240x320 象素，最后将文件保存为 logo.png 格式。
5.将上述文件保存到 fblogo 工具所在文件夹，首先在命令行下执行 convert -colors 223 logo.png logo.png 将文件保存为 224 色素（目前 fblogo 工具仅支持到 224 色），接着再使用 fblogo logo.png linux_logo.h 生成 linux_logo.h 内核启动代码中的图像缓冲区文件格式。
6.将 linux_logo.h 文件拷贝到 kernel 的 include/linux 目录即可。
7.重新编译 linux 系统内核，将生成的 zImage 内核镜像拷贝到 windows 下，准备对目标板进行烧写。
8.用串口将目标板与 PC 主机相连，接通电源，使用 bnw 工具进入目标板的 bios 程序并将生成的 zImage 内核镜像拷烧写到板子上，重起目标板，你会发现目标板上已经出现了美丽的 logo 图片。如果没有，请核对以上的步骤，并重新测试直到成功为止。
4 方案总结
4.1 方案实施过程中的关键问题
1.确保内核能够正常运行，编译时对开发板各硬件的驱动支持及其相关的配制，启动后能够自动加载文件系统。
2.安装 fblogo 工具，fblogo 工具的编译需要使用到第三方的库文件 libpng.so 库以及 zliba.a 库的支持，编译成功后，在命令行试运行 fblogo 如果安装成功将出现其使用的帮助信息。
3.fblogo 工具对图片的要求是仅支持 PNG 格式的 224 色图片。
4.最后针对不同的开发板的 LCD 显示屏应使用与其相适应的图片大小，否则可能使系统无法正常工作。
4.2 方案实施后的遗留问题
由于使用到了 logo 的制做工具 fblogo 所以图片在色彩上最大仅能支持到 224 色图片，在 bootlogo 启动文件 linux_logo.h 中有图像的色彩域点阵代码（即图片的 16 位进制的代码），如有需要可以自行修改，但其难度可想而知，建议可以将图片保存为 TIF 之类的非压缩格式，然后自己编写小程序将其从图象文件中的固定位置取点阵数据。
5 2.6 内核启动 Logo
在配置内核的时候选中了启动 Logo 的支持。
使用下面的方法可以将企鹅的 Logo 换成自己喜欢的任意图片。
首先准备一幅自己喜欢的图片，然后将背景涂成黑色。然后将该图片保存成 png 格式，例如 linuxlogo.png。在 Linux 下使用下面的命令：
pngtopnm linuxlogo.png > linuxlogo.pnm
pnmquant 224 linuxlogo.pnm > linuxlogo224.pnm
pnmtoplainpnm linuxlogo224.pnm > linuxlogo224.ppm
然.ppm 替换/usr/src/linux-2.6.8.1/drivers/后用生成的 linuxlogo224video/logo/logo_linux_clut224.ppm（最好先做好备份），然后删除同一目录下的 logo_linux_clut224.c 文件，重新编译内核，启动之后就可以在屏幕左上方看到自己的 Logo 了。