2020-2021 学年 第 一 学期

实 训 报 告

课程名称：ARM体系结构实训

学院名称： 智 能 装 备 学 院

专业班级： XXXXXXXXXXXXXXXXXXX

学生姓名： Sunqker

学生学号： XXXXXXXXXXXX

目录

[一、实训目的 3](#_Toc60409338)

[二、实训内容 3](#_Toc60409339)

[三、实训原理 3](#_Toc60409340)

[四、实训过程 4](#_Toc60409341)

[（一）交叉编译工具gcc的安装 4](#_Toc60409342)

[（二）、移植Uboot 10](#_Toc60409343)

[（三）、内核移植 26](#_Toc60409344)

[（四）、根文件系统移植 43](#_Toc60409345)

[（五）、串行端口程序设计与移植 51](#_Toc60409346)

[五、实训总结 56](#_Toc60409347)

# 一、实训目的

通过实训使学生熟练掌握ARM指令运用技术，掌握ARM处理器引导各个阶段的工作与原理，掌握U－boot各阶段移植时使用的技术细节，掌握移植U-boot移植的技术特点。通过实训使学生提高自我锤炼技术，自我拓展知识，不断积累技术，培养其运用所学知识与掌握技术进行实际工作的能力，培养其扎实的工作作风与合作精神，不断养成学生的创新意识并拓展其创新能力。

# 二、实训内容

（1）U-boot移植（已做）

（2）内核移植

（3）根文件系统移植

（4）串行端口程序设计与移植

# 三、实训原理

U-Boot不仅仅支持嵌入式Linux系统的引导，它还支持NetBSD, VxWorks, QNX, RTEMS, ARTOS, LynxOS, android嵌入式操作系统。其目前要支持的目标操作系统是OpenBSD, NetBSD, FreeBSD,4.4BSD, Linux, SVR4, Esix, Solaris, Irix, SCO, Dell, NCR, VxWorks, LynxOS, pSOS, QNX, RTEMS, ARTOS, android。这是U-Boot中Universal的一层含义，另外一层含义则是U-Boot除了支持PowerPC系列的处理器外，还能支持MIPS、 x86、ARM、NIOS、XScale等诸多常用系列的处理器。这两个特点正是U-Boot项目的开发目标，即支持尽可能多的嵌入式处理器和嵌入式操作系统。就目前来看，U-Boot对PowerPC系列处理器支持最为丰富，对Linux的支持最完善。其它系列的处理器和操作系统基本是在2002年11 月PPCBOOT改名为U-Boot后逐步扩充的。从PPCBOOT向U-Boot的顺利过渡，很大程度上归功于U-Boot的维护人德国DENX软件工程中心Wolfgang Denk[以下简称W.D]本人精湛专业水平和执着不懈的努力。当前，U-Boot项目正在他的领军之下，众多有志于开放源码BOOT LOADER移植工作的嵌入式开发人员正如火如荼地将各个不同系列嵌入式处理器的移植工作不断展开和深入，以支持更多的嵌入式操作系统的装载与引导。

Linux内核的主要模块（或组件）分以下几个部分：存储管理、CPU和进程管理、文件系统、设备管理和驱动、网络通信，以及系统的初始化（引导）、系统调用等。

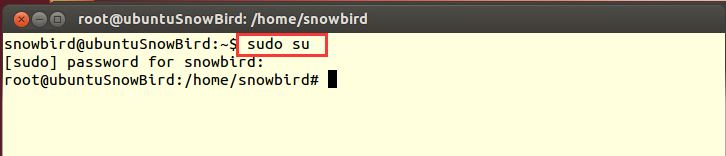
文件系统是对一个存储设备上的数据和元数据进行组织的机制。这种机制有利于用户和操作系统的交互。尽管内核是 Linux 的核心，但文件却是用户与操作系统交互所采用的主要工具。这对 Linux 来说尤其如此，这是因为在 UNIX 传统中，它使用文件 I/O 机制管理硬件设备和数据文件”，在Linux没有文件系统的话，用户和操作系统的交互也就断开了，例如使用最多的交互shell，包括其它的一些用户程序，都没有办法运行。在这里可以看到文件系统相对于Linux操作系统的重要性。

# 四、实训过程

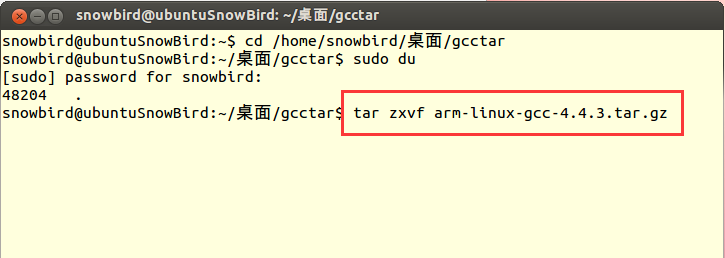
## （一）交叉编译工具gcc的安装

#### gcc的安装过程

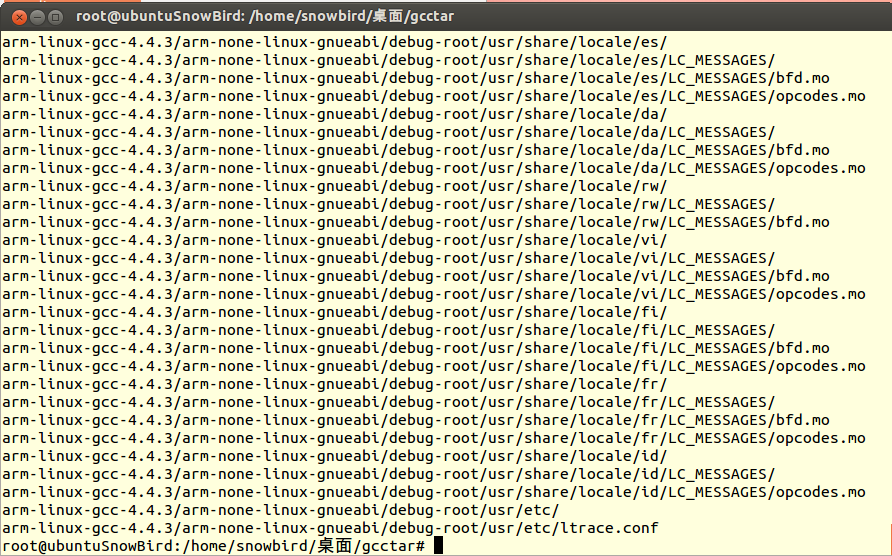
1.1、进入管理员模式



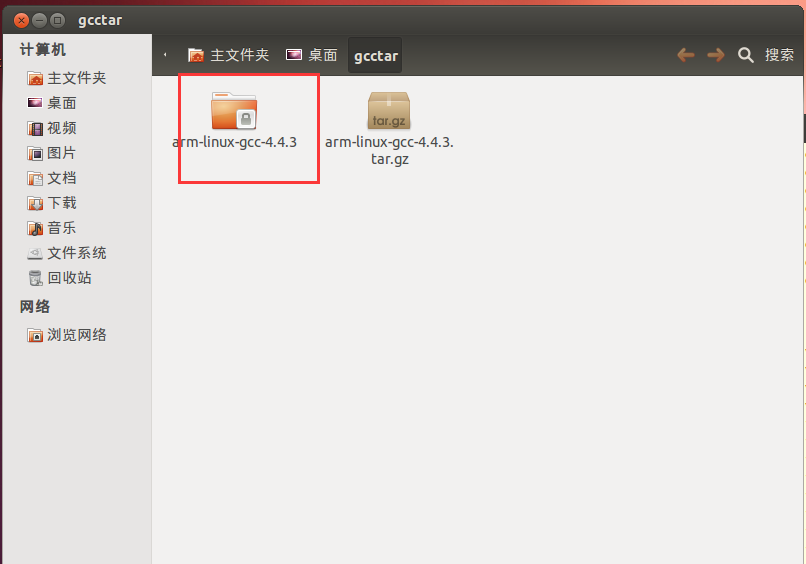
1.2、解压gcc安装包



* 1. 回车后，进行解压，命令框输出如下：



1.4、解压出来的文件夹如下：



1.5、配置环境变量

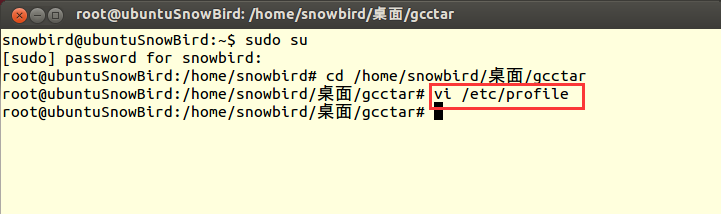
配置环境变量，把交叉编译工具链的路径添加到环境变量PATH中去，这样就可以在任何目录下使用这些工具。记下上一步中的安装路径，使用命令：

vi /etc/profile编辑profile文件；

添加环境变量，在profile中最后一行添加

export PATH=$PATH:/usr/local/arm-linux-gcc-4.4.3/bin

输入：vi /etc/profile



回车，在最下面添加：

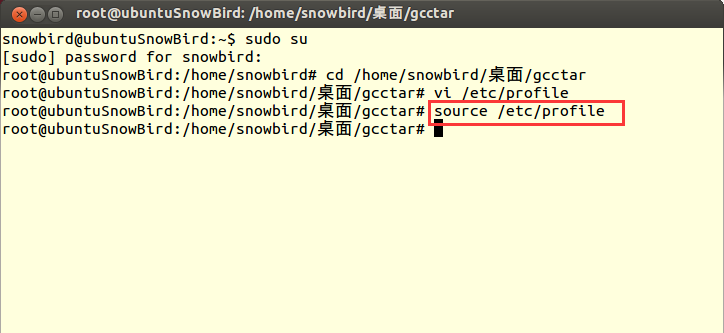
export PATH=$PATH:/usr/local/arm-linux-gcc-4.4.3/bin，然后按“esc”，输入“:wq”回车退出vi编辑。



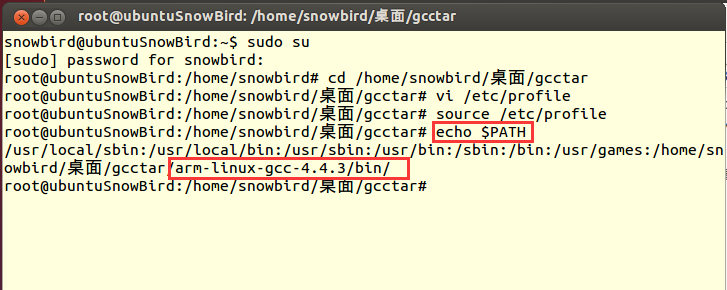


使环境变量生效并验证，然后使用命令：source /etc/profile

使环境变量生效（在不同用户下最好都运行一下此语句，例如shgao和root用户下都运行一下）。

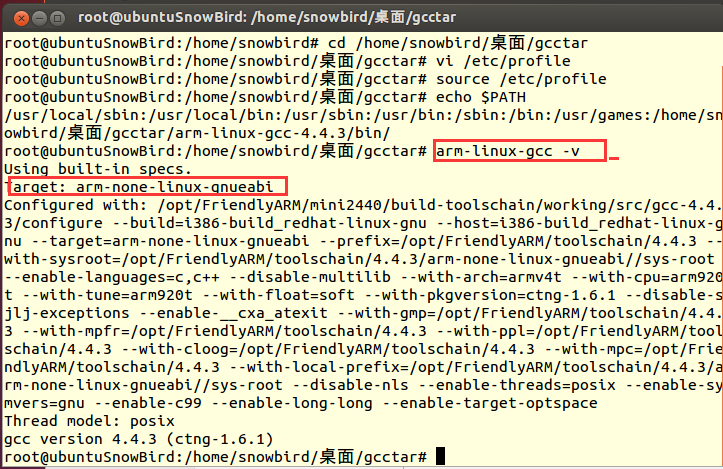


检查是否将路径加入到PATH：echo $PATH：



显示的内容中有/usr/local/arm-linux-gcc-4.4.3/bin，说明已经将交叉编译器的路径加入PATH。

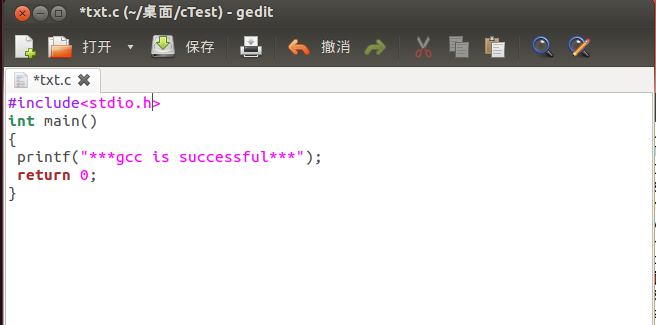
利用arm-linux-gcc -v查看环境是否搭建成功，如下图：



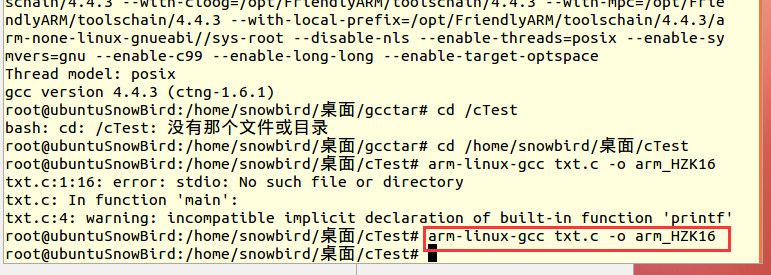
可以看出环境已经搭建成功。

1.6、编译生成二进制文件

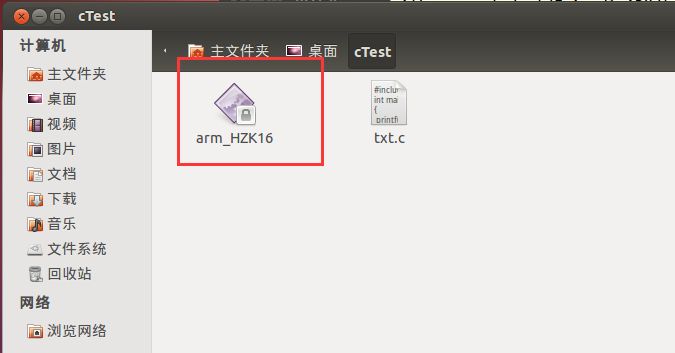
新建一个C语言文件夹，如下图：



编译生成可以在arm上运行的二进制文件：



回车后，可以看到生成了arm\_HZK16文件，arm\_HZK16文件为arm上运行的二进制文件，是一个可执行文件。



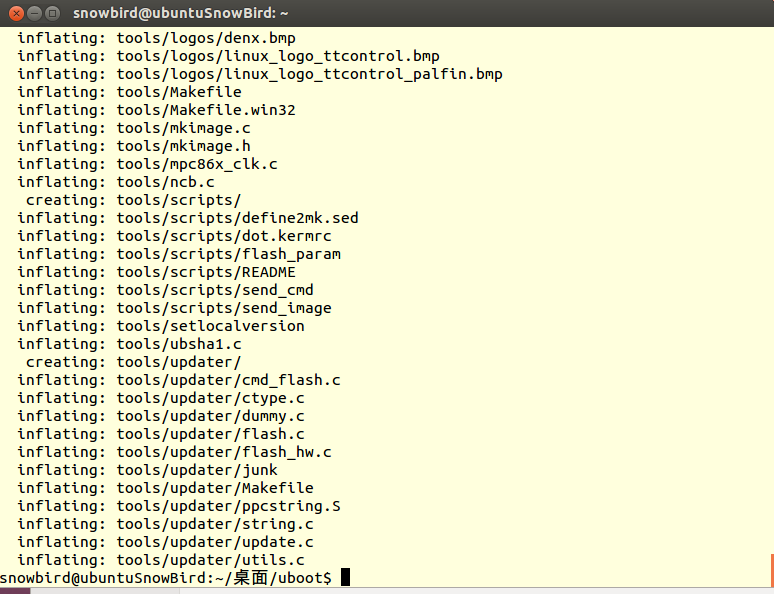
到这里已经完成了Ubuntu系统下arm-linux-gcc交叉编译环境搭建过程。

## （二）、移植Uboot

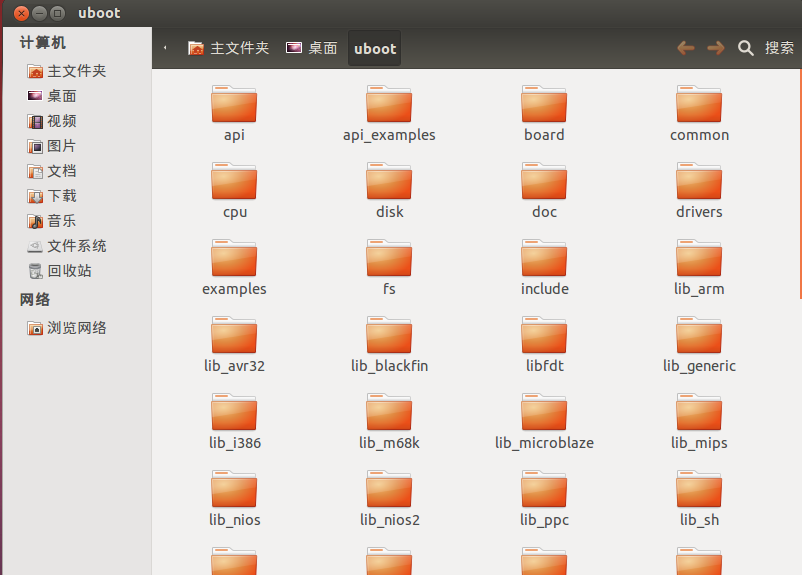
#### 1、Uboot移植过程

1.1、解压

输入命令：unzip u-boot-1.3.4.zip 进行解压



解压完成后显示如下的目录：

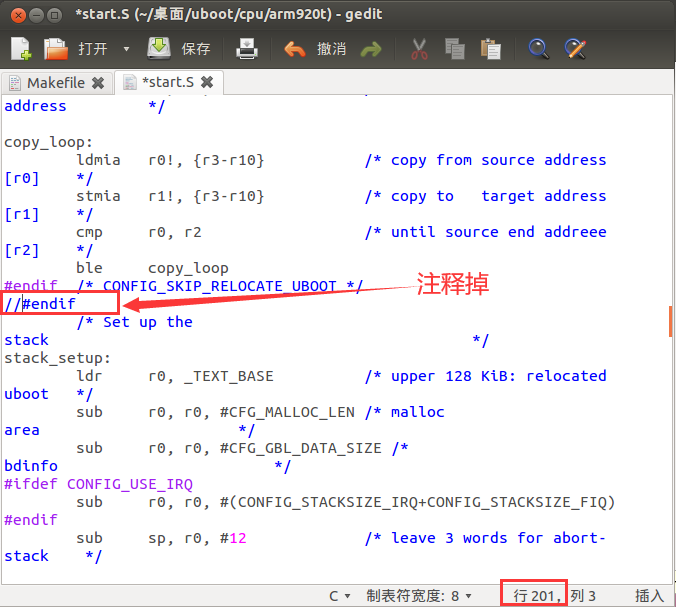


1.2、修改源码让u-boot支持从Nand Flash启动。

打开“uboot/cpu/arm920t/start.S”文件：

（1）181行和201行前加上 // 注释掉，如下所示



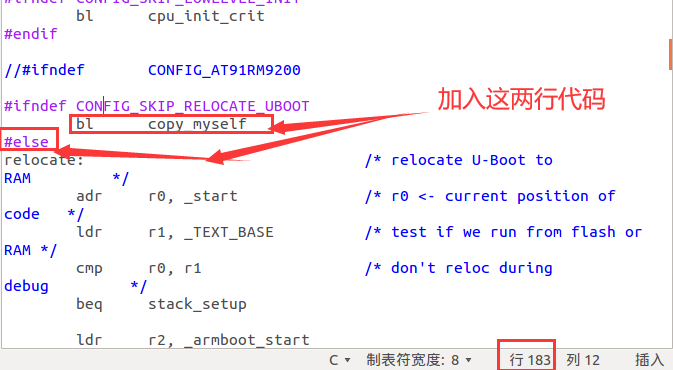


（2）找到183行，在#ifndef CONFIG\_SKIP\_RELOCATE\_UBOOT之后加入：

#ifdef CONFIG\_S3C2410\_NAND\_BOOT

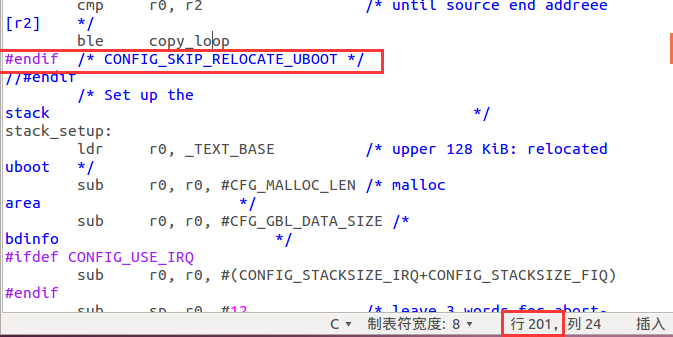
Bl copy\_myself

#else



（3）“ble copy\_loop”语句之后加入：

#endif



（4）“\_start\_armboot:.word start\_armboot”语句之后加入copy\_loop子程序：

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* copy u-boot to ram

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

#ifdef CONFIG\_S3C2410\_NAND\_BOOT

copy\_myself:

mov r10, lr @save return address to r10

ldr sp, DW\_STACK\_START

mov fp, #0

bl NF\_Init

ldr r0, =UBOOT\_RAM\_BASE

mov r1, #0x0

mov r2, #0x30000

bl nand\_read\_whole

tst r0, #0x0

beq ok\_nand\_read

1:b 1b

ok\_nand\_read:

mov r0, #0x00000000

ldr r1, =UBOOT\_RAM\_BASE

mov r2, #0x400

go\_next:

ldr r3, [r0], #4

ldr r4, [r1], #4

teq r3, r4

bne notmatch

subs r2, r2, #4

beq done\_nand\_read

bne go\_next

notmatch:

1: b 1b

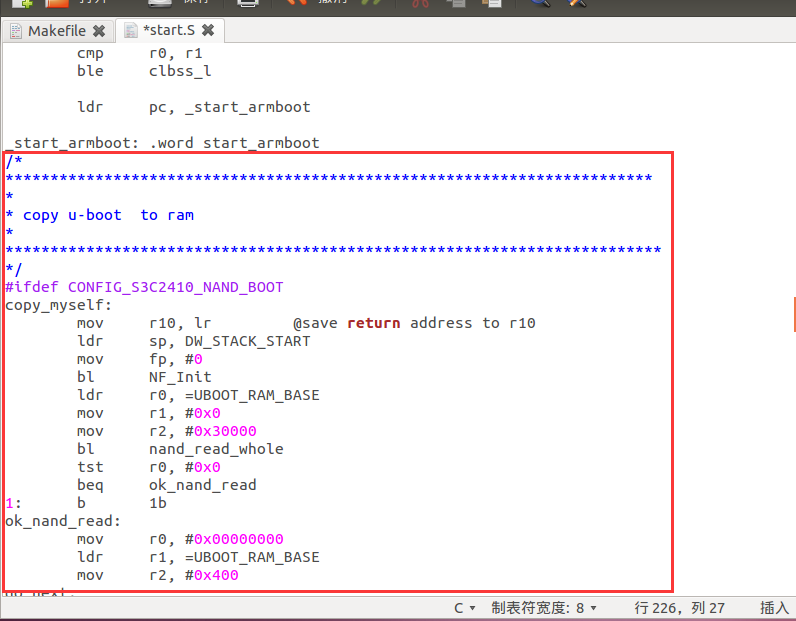
done\_nand\_read:

mov pc, r10

#endif

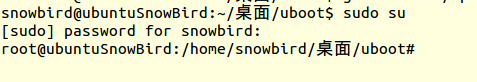
DW\_STACK\_START:

.word STACK\_BASE+STACK\_SIZE-4



（5）新建“u-boot-1.3.2/ board/smdk2410/nand.c”文件

进入管理员模式



在nand.c文件里添加如下内容：

#include <common.h>

#include <s3c2410.h>

#include <config.h>

#define TACLS 0

#define TWRPH0 3

#define TWRPH1 0

#define U32 unsigned int

extern unsigned long nand\_probe(unsigned long physadr);

static void NF\_Reset(void)

{

int i;

NF\_nFCE\_L();

NF\_CMD(0xFF);

for(i=0;i<10;i++);

NF\_WAITRB();

NF\_nFCE\_H();

}

void NF\_Init(void)

{

rNFCONF=(1<<15)|(1<<14)|(1<<13)|(1<<12)|(1<<11)|(TACLS<<8)|(TWRPH0<<4)|(TWRPH1<<0);

NF\_Reset();

}

int nand\_read\_whole(unsigned char \*buf, unsigned long start\_addr, int size)

{

int i, j;

if((start\_addr & NAND\_BLOCK\_MASK) || (size & NAND\_BLOCK\_MASK))

return 1;

NF\_nFCE\_L();

for(i=0; i<10; i++);

i = start\_addr;

while(i < start\_addr + size) {

rNFCMD = 0;

rNFADDR = i & 0xff;

rNFADDR = (i >> 9) & 0xff;

rNFADDR = (i >> 17) & 0xff;

rNFADDR = (i >> 25) & 0xff;

NF\_WAITRB();

for(j=0; j < NAND\_SECTOR\_SIZE; j++, i++) {

\*buf = (rNFDATA & 0xff);

buf++;

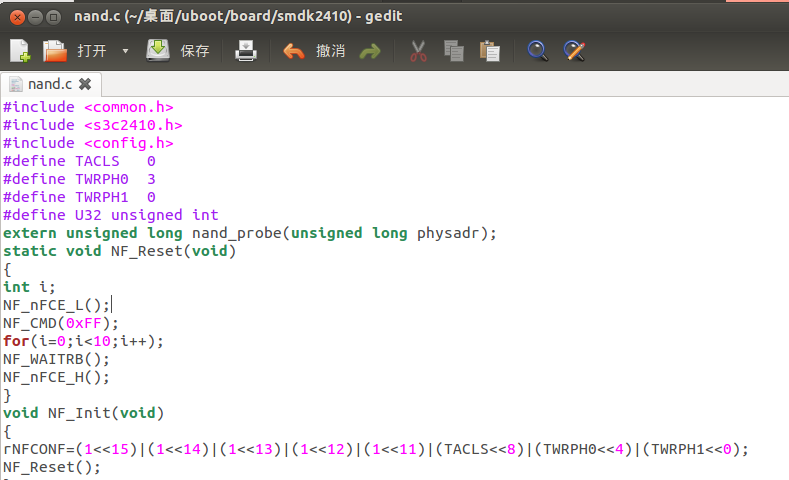
}

}

NF\_nFCE\_H();

return 0;

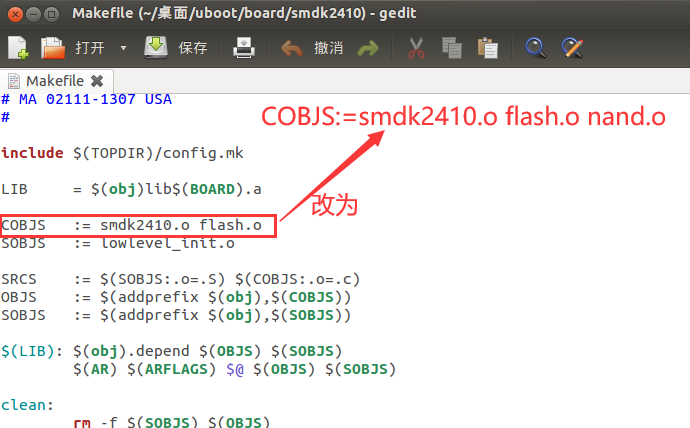
}



保存退出

（6）修改“u-boot-1.3.2/board/smdk2410/Makefile”文件

找到该文件夹，然后将“COBJS:= smdk2410.o flash.o”改为“COBJS:=smdk2410.o flash.o nand.o”

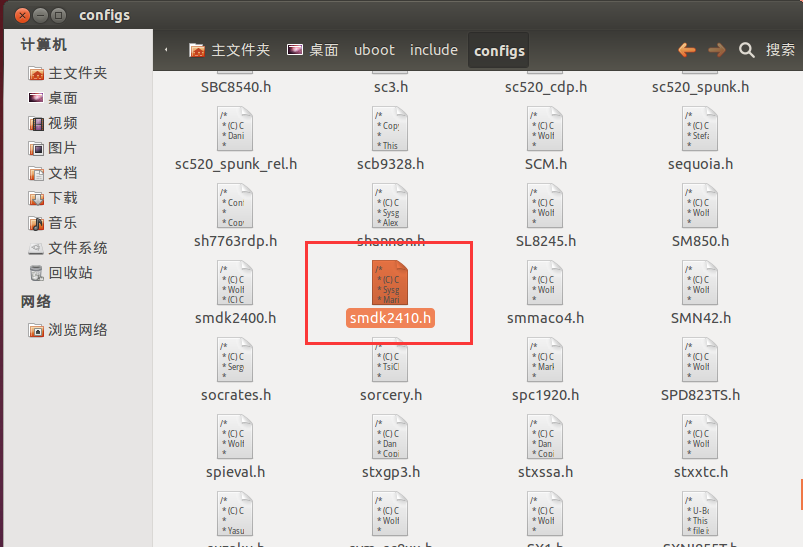


即改完后：



（7）修改“uboot/include/configs/smdk2410.h”文件

找到文件“uboot/include/configs/smdk2410.h”：



打开smdk2410.h，在该文件的最后一个#endif的前面添加如下内容：

#define CONFIG\_S3C2410\_NAND\_BOOT 1

#define STACK\_BASE 0x33f00000

#define STACK\_SIZE 0x8000

#define UBOOT\_RAM\_BASE 0x33f80000

#define CFG\_NAND\_BASE 0x4E000000

#define CFG\_MAX\_NAND\_DEVICE 1

#define SECTORSIZE 512

#define NAND\_SECTOR\_SIZE SECTORSIZE

#define NAND\_BLOCK\_MASK (NAND\_SECTOR\_SIZE - 1)

#define ADDR\_COLUMN 1

#define ADDR\_PAGE 2

#define ADDR\_COLUMN\_PAGE 3

#define NAND\_ChipID\_UNKNOWN 0x00

#define NAND\_MAX\_FLOORS 1

#define NAND\_MAX\_CHIPS 1

#define WRITE\_NAND\_COMMAND(d, adr) do {rNFCMD = d;} while(0)

#define WRITE\_NAND\_ADDRESS(d, adr) do {rNFADDR = d;} while(0)

#define WRITE\_NAND(d, adr) do {rNFDATA = d;} while(0)

#define READ\_NAND(adr) (rNFDATA)

#define NAND\_WAIT\_READY(nand) {while(!(rNFSTAT&(1<<0)));}

#define NAND\_DISABLE\_CE(nand) {rNFCONF |= (1<<11);}

#define NAND\_ENABLE\_CE(nand) {rNFCONF &= ~(1<<11);}

#define NAND\_CTL\_CLRALE(nandptr)

#define NAND\_CTL\_SETALE(nandptr)

#define NAND\_CTL\_CLRCLE(nandptr)

#define NAND\_CTL\_SETCLE(nandptr)

#define CONFIG\_MTD\_NAND\_VERIFY\_WRITE 1

#define rNFCONF (\*(volatile unsigned int \*)0x4e000000)

#define rNFCMD (\*(volatile unsigned char \*)0x4e000004)

#define rNFADDR (\*(volatile unsigned char \*)0x4e000008)

#define rNFDATA (\*(volatile unsigned char \*)0x4e00000c)

#define rNFSTAT (\*(volatile unsigned int \*)0x4e000010)

#define rNFECC (\*(volatile unsigned int \*)0x4e000014)

#define rNFECC0 (\*(volatile unsigned char \*)0x4e000014)

#define rNFECC1 (\*(volatile unsigned char \*)0x4e000015)

#define rNFECC2 (\*(volatile unsigned char \*)0x4e000016)

#define NF\_CMD(cmd) {rNFCMD=cmd;}

#define NF\_ADDR(addr) {rNFADDR=addr;}

#define NF\_nFCE\_L() {rNFCONF&=~(1<<11);}

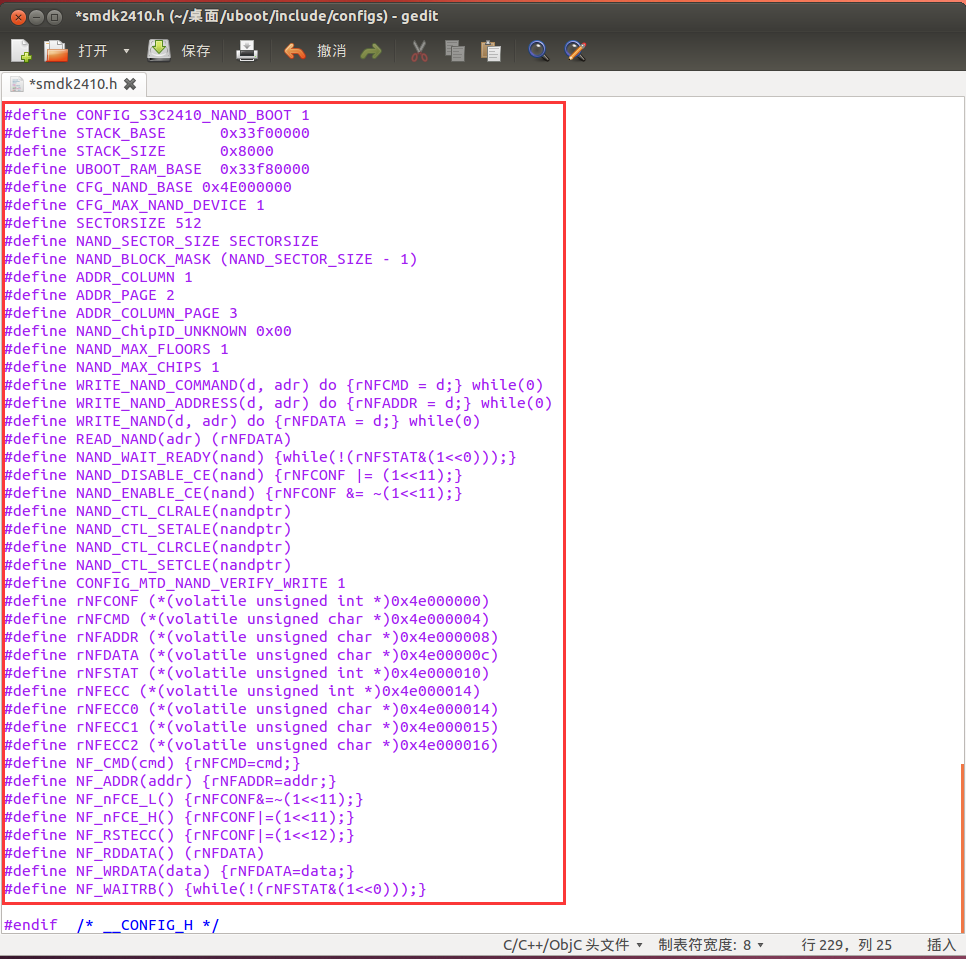
#define NF\_nFCE\_H() {rNFCONF|=(1<<11);}

#define NF\_RSTECC() {rNFCONF|=(1<<12);}

#define NF\_RDDATA() (rNFDATA)

#define NF\_WRDATA(data) {rNFDATA=data;}

#define NF\_WAITRB() {while(!(rNFSTAT&(1<<0)));}



（8）使u-boot支持dm9000网卡

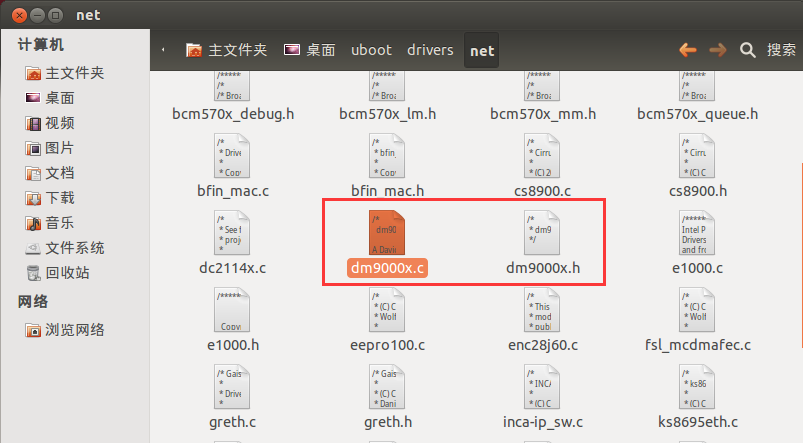
步骤如下：

一、 将dm9000x.h和dm9000x.c两个文件拷贝到drivers/net/下。

这两个的文件路径如下：

dm9000x.h：uboot/drivers/net/dm9000x.h

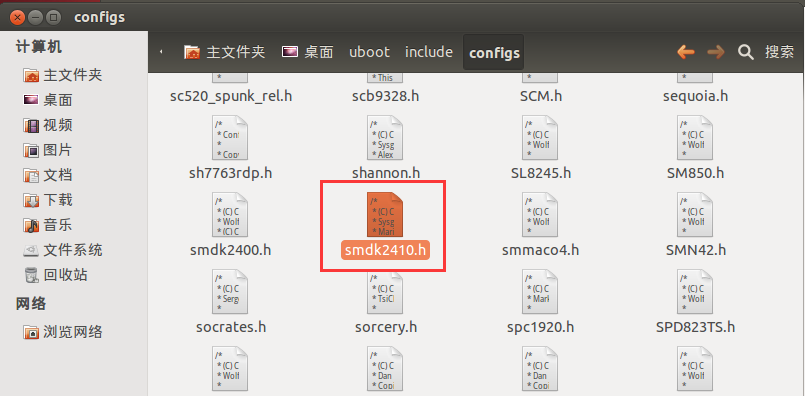
dm9000x.c：uboot/drivers/net/dm9000x.c



发现两个文件已经在drivers/net/下了。

6、 修改“uboot/include/configs/smdk2410.h”文件

找到文件



打开：

* 将原网卡“CS8900”改为“DM9000”
* 添加网络命令
* 修改IP地址
* 修改命令提示符
* 修改默认下载地址
* 修改环境变量在Nand Flash中的存储地址

具体如下：

（1）将原网卡“CS8900”改为“DM9000”

注释掉：

// #define CONFIG\_DRIVER\_CS8900 1 /\* we have a CS8900 on-board \*/

// #define CS8900\_BASE 0x19000300

// #define CS8900\_BUS16 1 /\* the Linux driver does accesses as shorts \*/

这3行之后添加：

#define CONFIG\_DRIVER\_DM9000 1

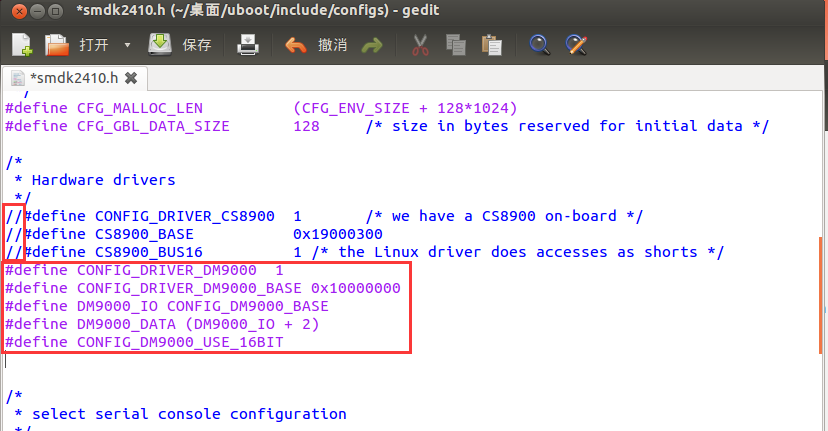
#define CONFIG\_DRIVER\_DM9000\_BASE 0x10000000

#define DM9000\_IO CONFIG\_DM9000\_BASE

#define DM9000\_DATA (DM9000\_IO + 2)

#define CONFIG\_DM9000\_USE\_16BIT

如下图



（2）添加网络命令

找到下面的几行：

#define CONFIG\_CMD\_CACHE

#define CONFIG\_CMD\_DATE

#define CONFIG\_CMD\_ELF

在其下面添加下面几行：

#define CONFIG\_CMD\_REGINFO

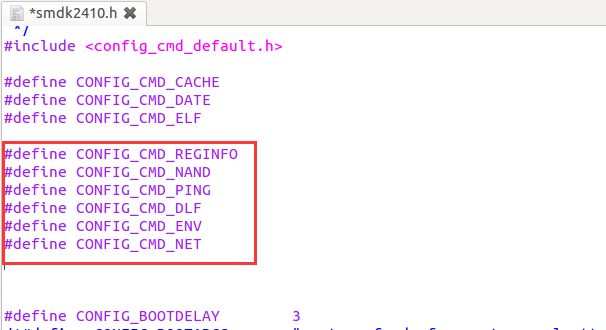
#define CONFIG\_CMD\_NAND

#define CONFIG\_CMD\_PING

#define CONFIG\_CMD\_DLF

#define CONFIG\_CMD\_ENV

#define CONFIG\_CMD\_NET



（3）修改IP地址

找到下面几行：

#define CONFIG\_BOOTDELAY 3

/\*#define CONFIG\_BOOTARGS "root=ramfs devfs=mount console=ttySA0,9600" \*/

/\*#define CONFIG\_ETHADDR 08:00:3e:26:0a:5b \*/

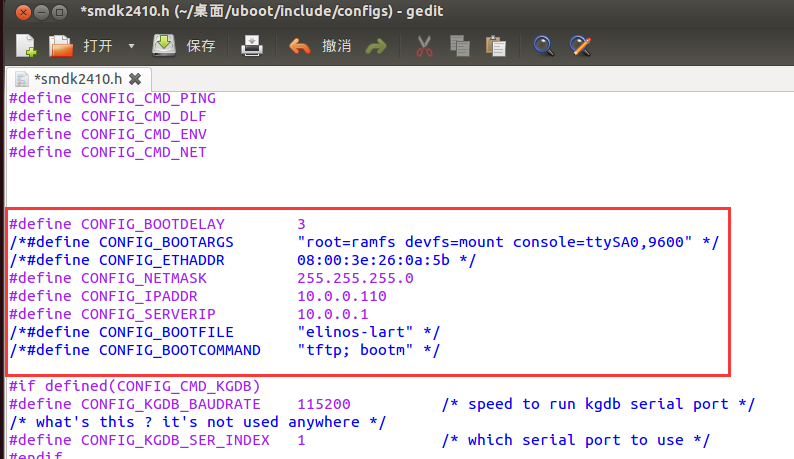
#define CONFIG\_NETMASK 255.255.255.0

#define CONFIG\_IPADDR 10.0.0.110

#define CONFIG\_SERVERIP 10.0.0.1

/\*#define CONFIG\_BOOTFILE "elinos-lart" \*/

/\*#define CONFIG\_BOOTCOMMAND "tftp; bootm" \*/



修改为：

#define CONFIG\_BOOTDELAY 3

#define CONFIG\_BOOTARGS "root=/dev/mtdblock3 init=/linuxrc console=ttySAC0,115200"

#define CONFIG\_ETHADDR 08:00:3e:26:0a:5b

#define CONFIG\_NETMASK 255.255.255.0

#define CONFIG\_IPADDR 192.168.1.13

#define CONFIG\_SERVERIP 192.168.1.12

#define CONFIG\_BOOTFILE "uImage"

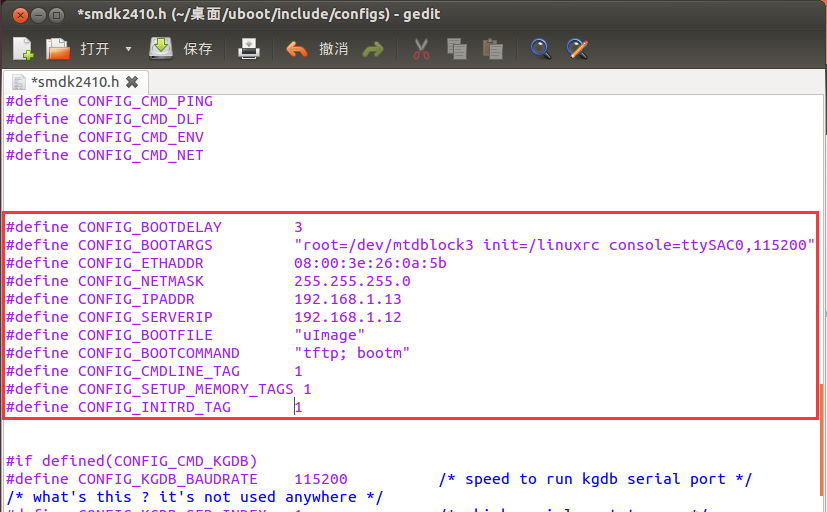
#define CONFIG\_BOOTCOMMAND "tftp; bootm"

#define CONFIG\_CMDLINE\_TAG 1

#define CONFIG\_SETUP\_MEMORY\_TAGS 1

#define CONFIG\_INITRD\_TAG 1

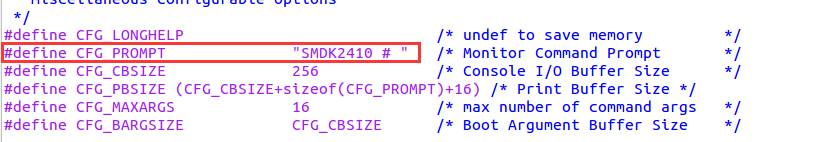
修改后如下图所示



（4）修改命令提示符

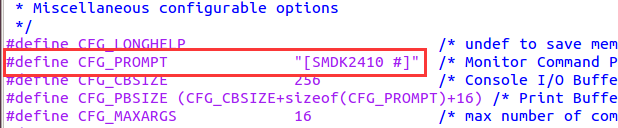
找到下面一行：

#define CFG\_PROMPT "SMDK2410 # "



修改为：

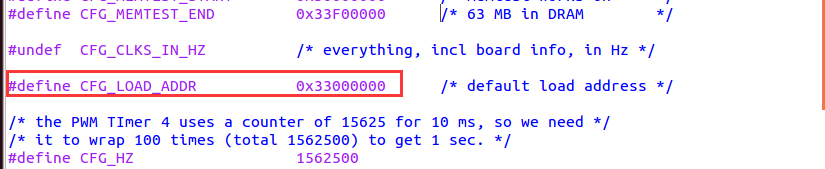
#define CFG\_PROMPT "[ smdk2410 #]"



（5）修改默认下载地址

找到下面一行：

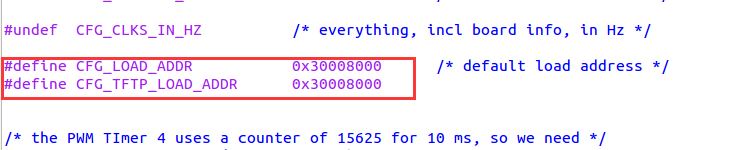
#define CFG\_LOAD\_ADDR 0x33000000



修改为：

#define CFG\_LOAD\_ADDR 0x30008000

#define CFG\_TFTP\_LOAD\_ADDR 0x30008000

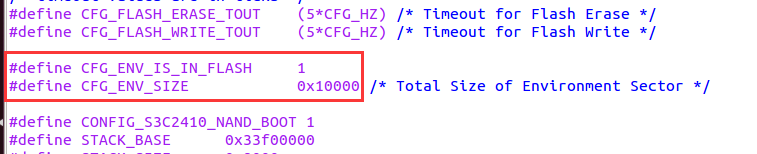


（6）修改环境变量在Nand Flash中的存储地址

找到下面两行：

#define CFG\_ENV\_IS\_IN\_FLASH 1

#define CFG\_ENV\_SIZE 0x10000

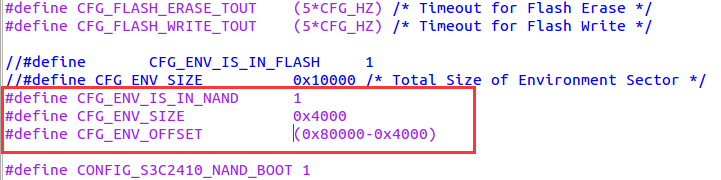


修改为：

#define CFG\_ENV\_IS\_IN\_NAND 1

#define CFG\_ENV\_SIZE 0x4000

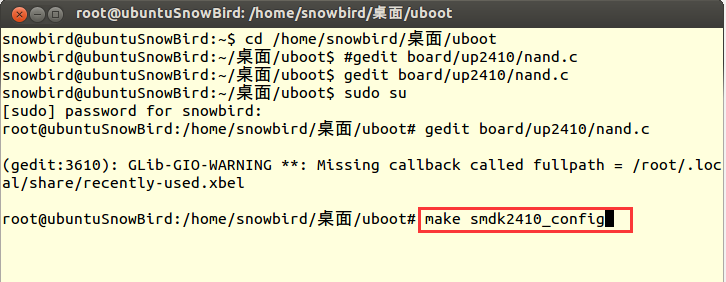
#define CFG\_ENV\_OFFSET (0x80000-0x4000)



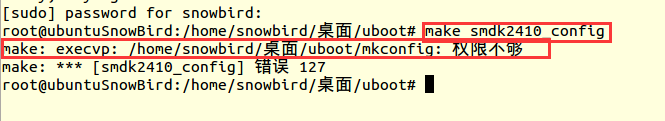
7、编译生成u-boot

依次执行以下命令：

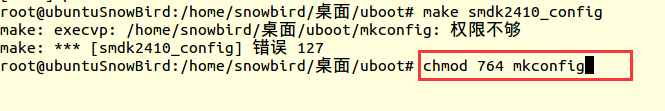
#make smdk2410\_config



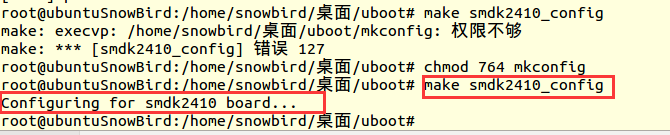
发现提示权限不够，



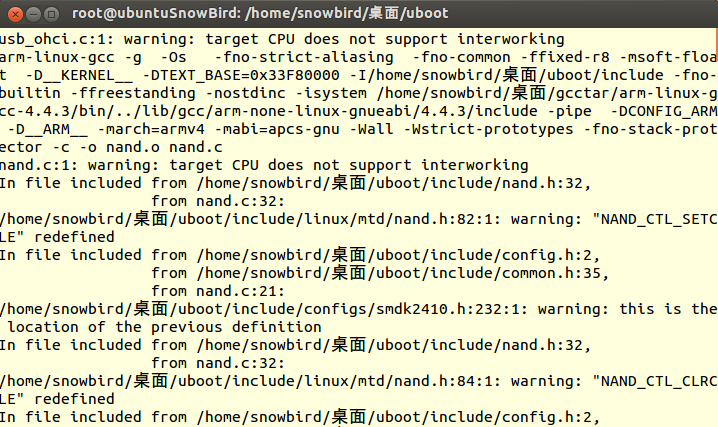
加上权限，



然后再执行make smdk2410\_config，发现可成功执行



#make



编译完成之后就生成了u-boot.bin文件，这个文件就是我们要烧写到开发板上的文件。

#### 2、存在的问题及解决方案

## （三）、内核移植

#### 1、内核移植过程

1.1下载linux内核

下载 linux­2.6.14.1 内核至 home/arm/dev\_home/kernel. [root@localhost~]#su arm

[arm@localhost~]#cd $KERNEL

[arm@localhost kernel]#tar ­xzvf linux­2.6.14.1.tar.gz

[arm@localhost kernel]# pwd

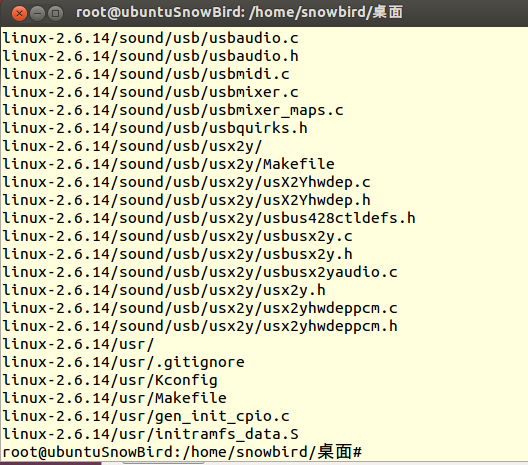
/home/arm/dev home/kernel

[arm@localhost kernel]# cd linux­2.6.14

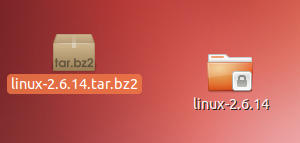
进入内核解压后的目录，以后示例中，只要是相对路径全部是相对于

/home/arm/dev\_home/kernel/linux­2.6.14/此目录

解压：tar zxvf linux-2.6.14.tar.bz2



解压后：



1.2修改 Makefile

修改内核目录树根下的的 Makefile，指明交叉编译器

#gedit Makefile

找到 ARCH 和 CROSS\_COMPILE，修改为

ARCH ?= arm

CROSS\_COMPILE   ?= arm­linux­



然后设置你的 PATH环境变量，使其可以找到你的交叉编译工具链

# echo $PATH

/usr/local/arm/3.4.4/bin:/usr/kerberos/bin:/usr/local/bin:/usr/bin:/bin:/usr/X11R6/bin:/home/ly/bin

如果/usr/local/arm/3.4.4/bin 搜索路径, 加入下面语句在~/.bashrc 中

[arm@localhost linux­2.6.14]# vi~/.bashrc

export PATH=/usr/local/arm/3.4.4/bin:$PATH

再重新登陆。

[arm@localhost linux­2.6.14]#su arm

1.3 设置 flash 分区

此处一共要修改3个文件，分别是：

①指明分区信息

在 arch/arm/mach­s3c2410/devs.c 文件中:

[arm@localhost linux­2.6.14]$ vi arch/arm/mach­s3c2410/dev s.c

添加如下内容：

#include <linux/mtd/partitions.h>

#include <linux/mtd/nand.h>

#include <asm/arch/nand.h>



②建立Nand Flash分区表

/\*一个 Nand Flash 总共 64MB, 按如下大小进行分区 \*/

Static struct mtd\_partition partition\_info[]={

{/\* 1MB \*/

name:"bootloader",

size:0x00100000,

offset:0x0,

},{/\*3MB\*/

name:"kernel",

size:0x00300000,

offset:0x00100000,

},{/\*40MB\*/

name:"root",

size:0x02800000,

offset:0x00400000,

},{/\*20MB\*/

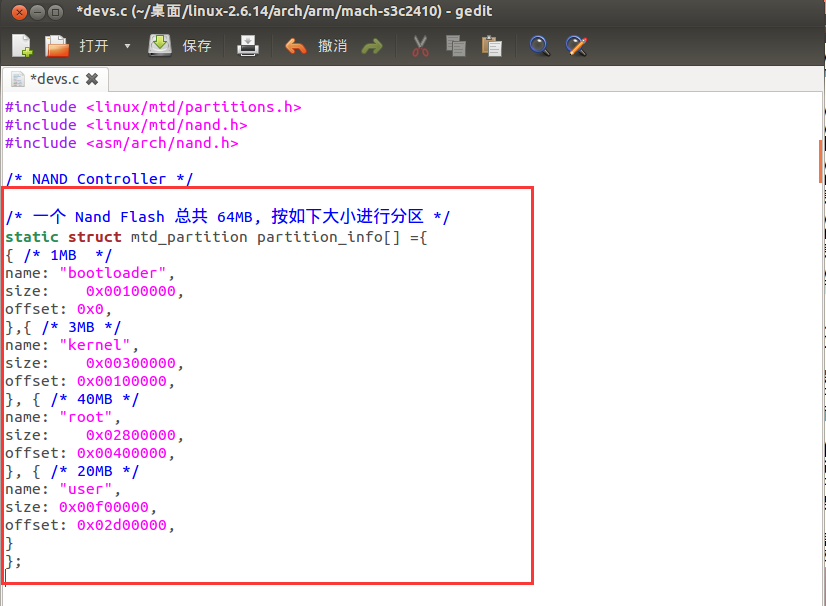
name:"user",

size:0x00f00000,

offset:0x02d00000,

}

};



name:代表分区名字

size:代表 flash 分区大小(单位：字节)

offset:代表 flash 分区的起始地址(相对于 0x0 的偏移)

目标板计划分4个区，分别存放 bootloader，kernel，rootfs以及以便以后扩展使用的用户文件系统空间。

各分区在Nand flash中起始地址。分区大小记录如下：

bootloader:

start:0x00000000

len:0x00100000

1MB

kernel:

start:0x00100000

len:0x00300000

3MB

rootfs:

start:0x00400000

len:0x02800000

40MB

User:

start:0x02c00000

len:0x01400000

20MB

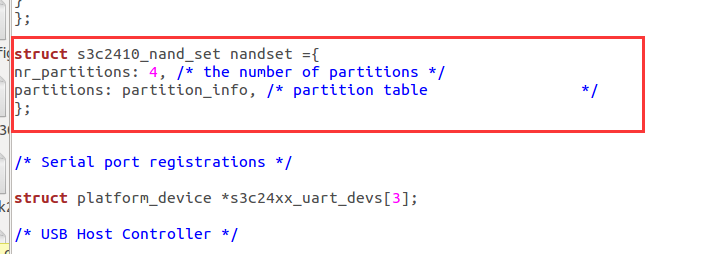
③ 加入 Nand Flash 分区

Struct s3c2410\_nand\_set nandset={

nr\_partitions:4,/\* the number of partitions \*/

partitions:partition\_info,/\*partition table \*/

};



nr\_partitions: 指明partition\_info中定义的分区数目

partitions: 分区信息表

④ 建立 Nand Flash 芯片支持

Struct s3c2410\_platform\_nand superlpplatform={

tacls:0,

twrph0:30,

twrph1:0,

sets:&nandset,

nr\_sets: 1,

};



tacls,twrph0,twrph1 的意思见 S3C2410 手册的 6­3，这3个值最后会被设置。

sets:支持的分区集

nr\_set:分区集的个数

⑤ 加入 Nand Flash 芯片支持到 Nand Flash驱动

另外，还要修改此文件中的 s3c\_device\_nand 结构体变量,添加对dev成员的赋值

Struct platform\_device s3c\_device\_nand={

.name="s3c2410­nand",/\* Device name \*/

.id= ­1,/\* Device ID \*/

.num\_resources = ARRAY\_SIZE(s3c\_nand\_resource),

.resource = s3c\_nand\_resource,

/\*Nand Flash Controller Registers \*/

/\*Add the Nand Flash device \*/

.dev = {

.platform\_data = &superlpplatform

  }

};



name:设备名称

id:有效设备编号,如果只有唯一的一个设备为­1，有多个设备从 0 开始计数。

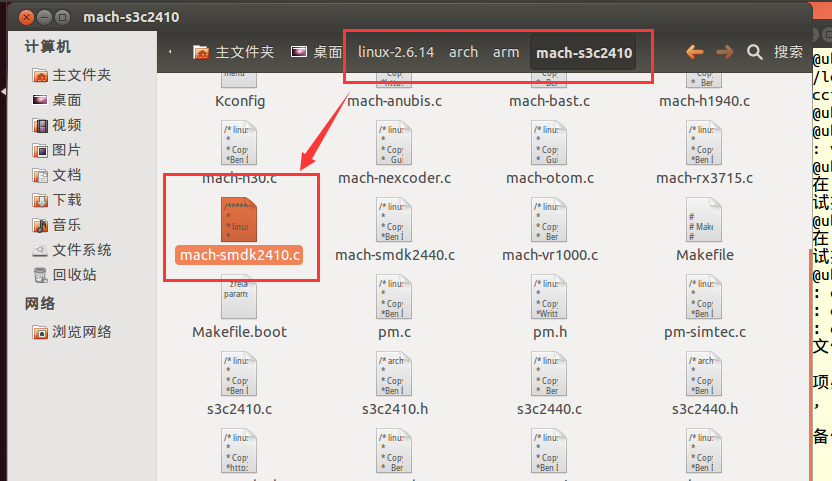
num\_resource:有几个寄存器区

resource:寄存器区数组首地址

dev:支持的 Nand Flash设备

⑥ 指定启动时初始化

kernel启动时依据我们对分区的设置进行初始配置，修改 arch/arm/mach­s3c2410/mach­smdk2410.c文件。



修改smdk2410\_devices[]。指明初始化时包括我们在前面所设置的 flash 分区信息。

Static struct platform\_device \*smdk2410\_devices[] \_\_initdata = {

&s3c\_device\_usb,

&s3c\_device\_lcd,

&s3c\_device\_wdt,

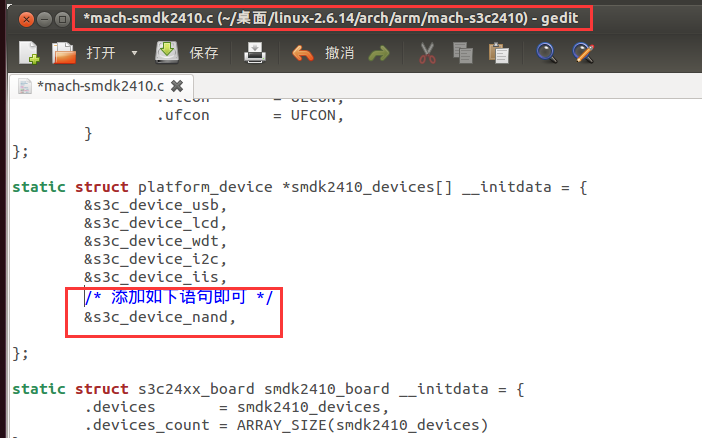
&s3c\_device\_i2c,

&s3c\_device\_iis,

/\* 添加如下语句即可 \*/

&s3c\_device\_nand,

};



保存，退出。

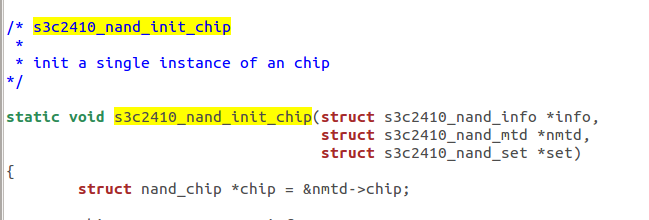
⑦ 禁止 Flash ECC 校验

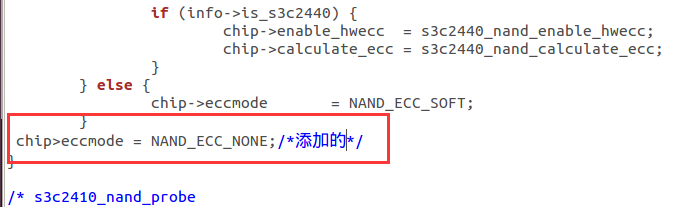
我们的内核都是通过 UBOOT写到 Nand Flash 的，UBOOT通过的软件 ECC算法产生 ECC 校验码，这与内核校验的 ECC 码不一样，内核中的 ECC码是由 S3C2410 中 Nand Flash 控制器产生的。所以，我们在这里选择禁止内核 ECC 校验。

修改 drivers/mtd/nand/s3c2410.c文件：

找到路径/home/snowbird/桌面/linux-2.6.14/drivers/mtd/nand下的s3c2410.c文件，然后找到s3c2410\_nand\_init\_chip()函数，在该函数体最后加上一条语句:

chip­>eccmode = NAND\_ECC\_NONE;



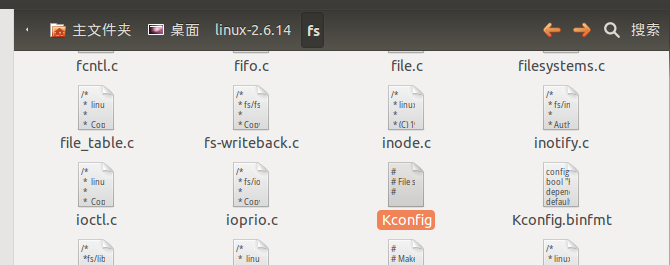


我们的关于 flash 分区的设置全部完工。

1.4 配置内核

① 支持启动时挂载 devfs

为了我们的内核支持devfs以及在启动时并在/sbin/init 运行之前能自动挂载/dev为devfs文件系统，修改fs/Kconfig 文件，在/home/snowbird/桌面/linux-2.6.14/fs路径下找到Kconfig文件夹



找到 menu "Pseudo filesystems"，添加如下语句：

config DEVFS\_FS

bool "/dev file system support (OBSOLETE)"

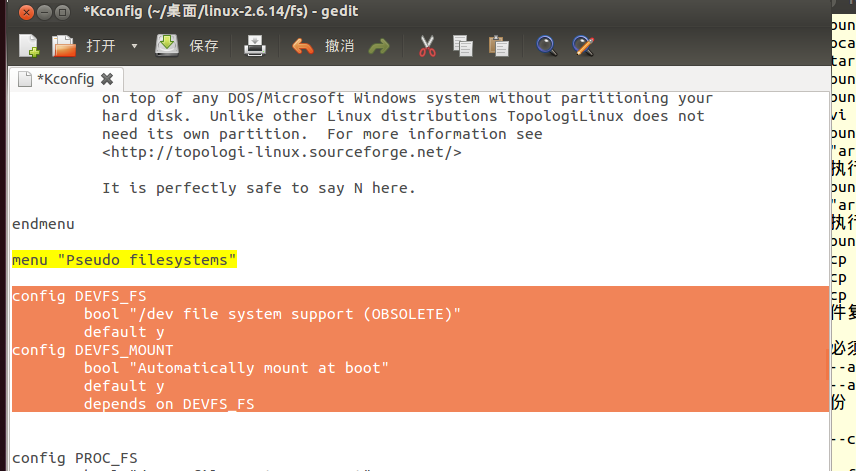
default y

config DEVFS\_MOUNT

bool "Automatically mount at boot"

default y

depends on DEVFS\_FS



② 配置内核产生.config 文件

在 smdk2410\_defconfig 基础上，我所增删的内核配置项如下：

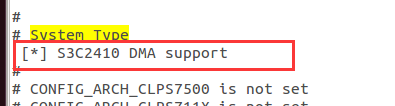
Loadable module support­­­>

[\*] Enable loadable module support

[\*] Automatic kernel module loading



System Type ­­­> [\*] S3C2410 DMA support



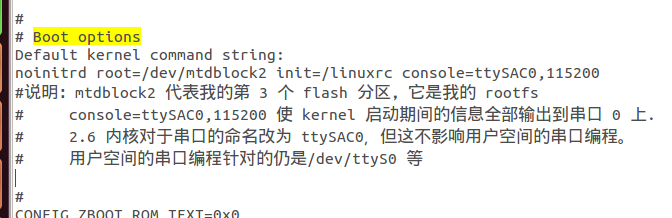
Boot options­­­> Default kernel command string:

#说明：mtdblock2 代表我的第 3 个 flash 分区，它是我的 rootfs

#console=ttySAC0,115200 使kernel 启动期间的信息全部输出到串口0 上。

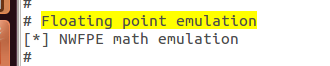
#2.6 内核对于串口的命名改为 ttySAC0，但这不影响用户空间的串口编程。

#用户空间的串口编程针对的仍是/dev/ttyS0 等。



Floating point emulation ­­­>

[\*] NWFPE math emulation



#接下来要做的是对内核 MTD 子系统的设置

Device Drivers ­­­>

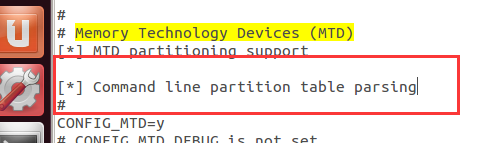
Memory Technology Devices (MTD) ­­­>

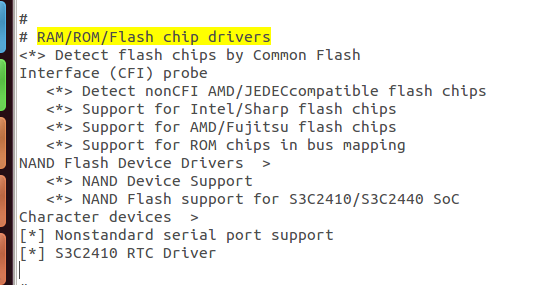
[\*] MTD partitioning support



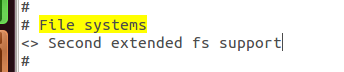
#支持MTD分区，这样我们在前面设置的分区才有意义

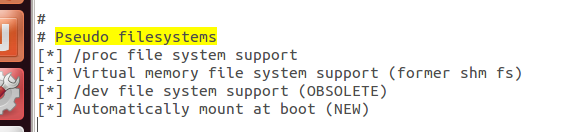
[\*]Command line partition table parsing





#接下来做的是针对文件系统的设置，本人实验时目标板上要上的文件系统是 cramfs，故做如下配置

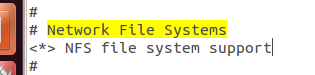
#去除对 ext2 的支持



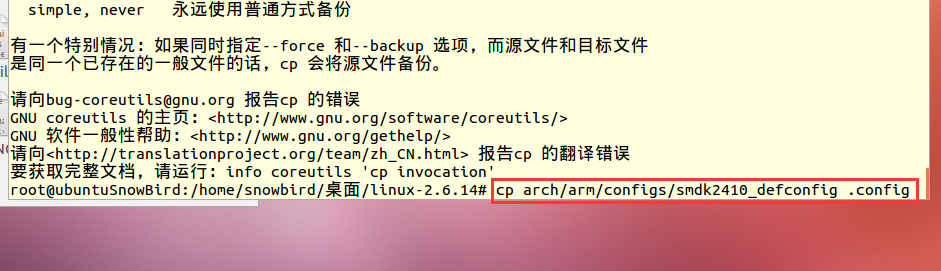
    #这里会看到我们前先修改 fs/Kconfig 的成果，devfs 已经被支持上了



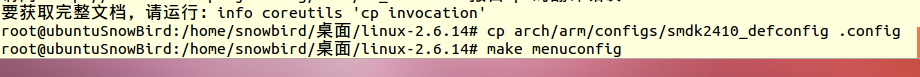
#支持 cramfs



[arm@localhost linux­2.6.14]$ cparch/arm/configs/smdk2410\_defconfig .config

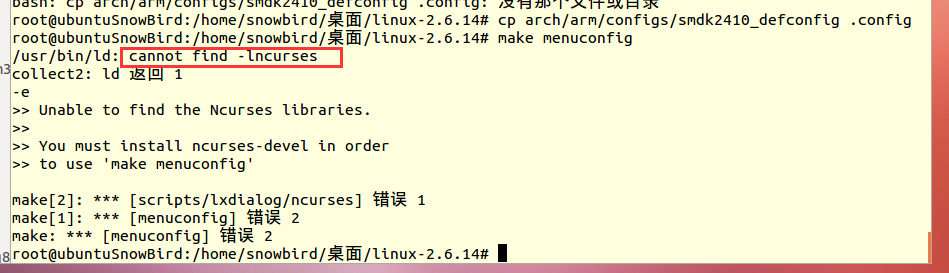


[arm@localhost linux­2.6.14]$ make menuconfig



#### 2、存在的问题及解决方案

回车之后产生如下的错误：



会发生这样的原因有以下三种情形：

1. 系统没有安装相对应的lib
2. 相对应的lib版本不对

③ lib(.so档)的symbolic link 不正确，没有连结到正确的函式库文件(.so)

解决方法：

(1)先判断在/usr/lib 下的相对应的函式库文件(.so) 的symbolic link 是否正确

若不正确改成正确的连结目标即可解决问题。

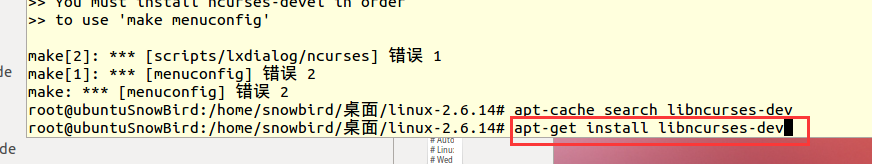
(2)若不是symbolic link 的问题引起，而是系统缺少相对应的lib安装lib即可解决。

(3)如何安装缺少的lib：

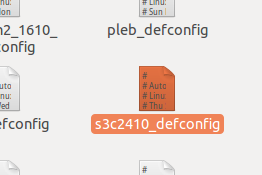
先搜寻相对应的LIB再进行安装的作业：



安装对应的LIB



运行完后会产生一个这样的文件：

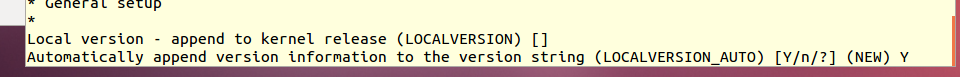


.config 文件能从提供的 2.4.14.1 的内核包中找到，文件名为 config.back。

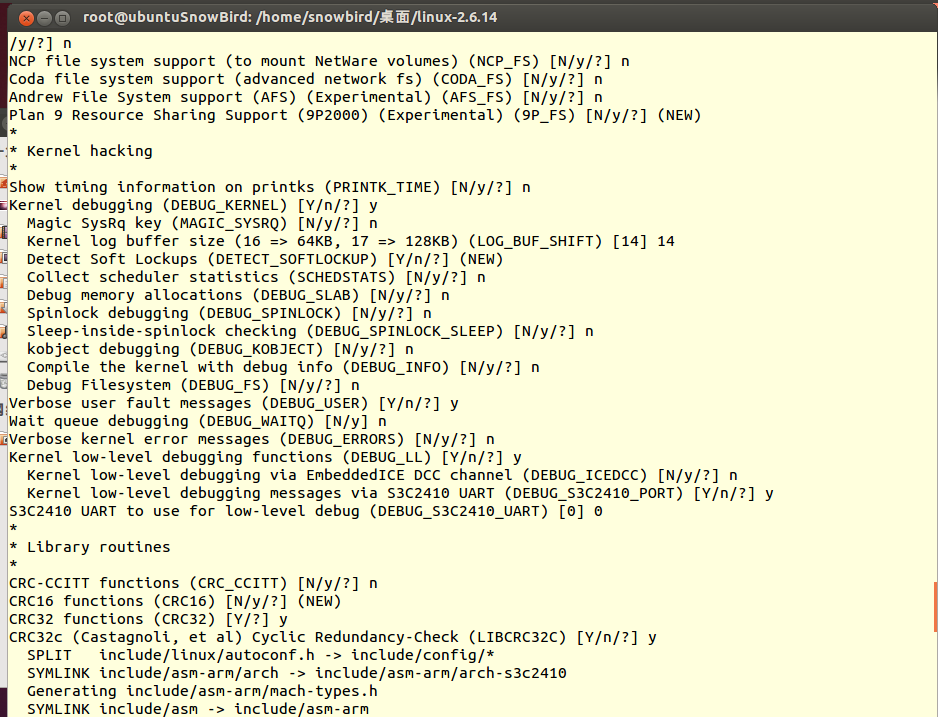
#### 3、编译内核

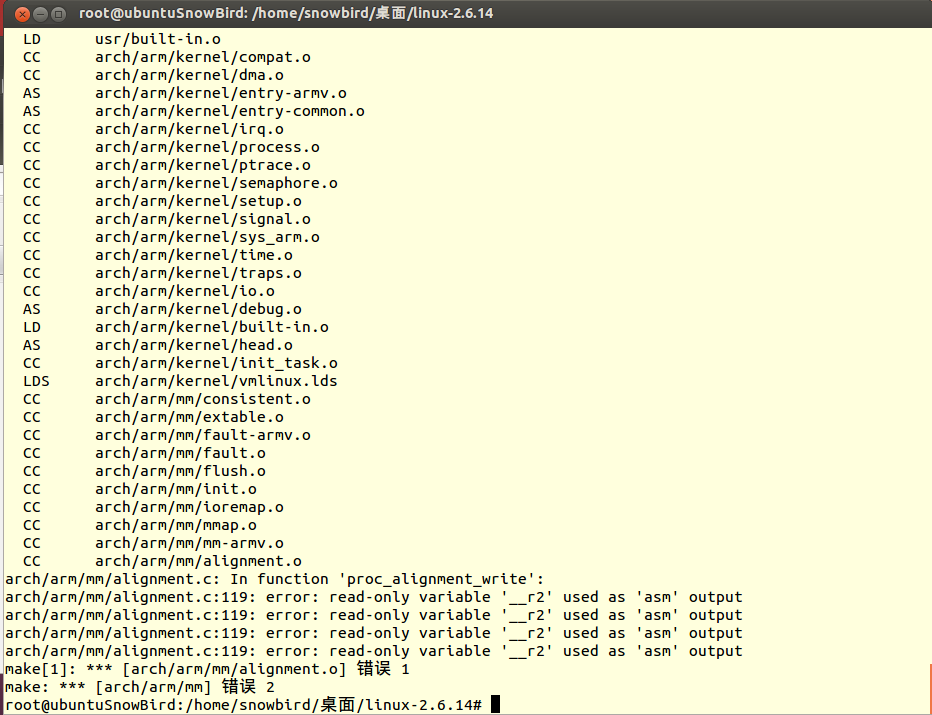
[arm@localhost linux­2.6.14]$ make zImage





这里直接回车，之后也是一路回车，直到内核编译完成，这时部分的界面如下：





#### 4、下载 zImage 到开发板

CRANE2410 # tftp 0x30008000 zImage

TFTP from server 192.168.1.6; our IP address is 192.168.1.5

Filename 'zImage'.

Load address: 0x30008000

Loading: ################################################################################################################################

done

Bytes transferred = 1142856 (117048 hex)

CRANE2410 # bootm 0x30008000

1.4.5 目标板启动信息 如 下

IRQ Stack: 33fc149c

FIQ Stack: 33fc249c

1

1

DRAM Configuration:

Bank #0: 30000000 64 MB

1

NAND:64 MB

In:    serial

Out:   serial

Err:   serial

Hit any key to stop autoboot:  0

zImage magic = 0x016f2818

NOW, Booting Linux......

Uncompressing Linux............................................................................ don.Linux version 2.6.14.1 (arm@dozec) (gcc

version 3.3.2) #15 Thu Jul 6 14:26:29 CST 2006

CPU: ARM920Tid(wb) [41129200] revision 0 (ARMv4T)

Machine: SMDK2410

Warning: bad configuration page, trying to continue

Memory policy: ECC disabled, Data cache writeback

CPU S3C2410A (id 0x32410002)

S3C2410: core 202.800 MHz, memory 101.400 MHz, peripheral 50.700 MHz

S3C2410 Clocks, (c) 2004 Simtec Electronics

CLOCK: Slow mode (1.500 MHz), fast, MPLL on, UPLL on

CPU0: D VIVT write­back 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

Kernel command line: noinitrd root=/dev/mtdblock2 init=/linuxrc console=ttySAC0,115200

irq: clearing subpending status 00000002

PID hash table entries: 128 (order: 7, 2048 bytes)

timer tcon=00500000, tcnt a509, tcfg 00000200,00000000, usec 00001e4c

Console: colour dummy device 80x30

Dentry cache hash table entries: 4096 (order: 2, 16384 bytes)

Inode­cache hash table entries: 2048 (order: 1, 8192 bytes)

Memory: 16MB = 16MB total

Memory: 13712KB available (1927K code, 422K data, 104K init)

Mount­cache hash table entries: 512

CPU: Testing write buffer coherency: ok

softlockup thread 0 started up.

NET: Registered protocol family 16

S3C2410: Initialising architecture

SCSI subsystem initialized

usbcore: registered new driver usbfs

usbcore: registered new driver hub

S3C2410 DMA Driver, (c) 2003­2004 Simtec Electronics

DMA channel 0 at c1800000, irq 33

DMA channel 1 at c1800040, irq 34

DMA channel 2 at c1800080, irq 35

DMA channel 3 at c18000c0, irq 36

NetWinder Floating Point Emulator V0.97 (double precision)

devfs: 2004­01­31 Richard Gooch (rgooch@atnf.csiro.au)

devfs: boot\_options: 0x1

Console: switching to colour frame buffer device 80x25

fb0: Virtual frame buffer device, using 1024K of video memory

S3C2410 RTC, (c) 2004 Simtec Electronics

s3c2410\_serial0 at MMIO 0x50000000 (irq = 70) is a S3C2410

s3c2410\_serial1 at MMIO 0x50004000 (irq = 73) is a S3C2410

s3c2410\_serial2 at MMIO 0x50008000 (irq = 76) is a S3C2410

io scheduler noop registered

io scheduler anticipatory registered

io scheduler deadline registered

io scheduler cfq registered

RAMDISK driver initialized: 16 RAM disks of 4096K size 1024 blocksize

Cirrus Logic CS8900A driver for Linux (Modified for SMDK2410)

eth0: CS8900A rev E at 0xe0000300 irq=53, no eeprom , addr: 08: 0:3E:26:0A:5B

S3C24XX NAND Driver, (c) 2004 Simtec Electronics

s3c2410­nand: mapped registers at c1980000

s3c2410­nand: timing: Tacls 10ns, Twrph0 30ns, Twrph1 10ns

NAND device: Manufacturer ID: 0xec, Chip ID: 0x76 (Samsung NAND 64MiB 3,3V 8­bit)

NAND\_ECC\_NONE selected by board driver. This is not recommended !!

Scanning device for bad blocks

Creating 4 MTD partitions on "NAND 64MiB 3,3V 8­bit":

0x00000000­0x00100000 : "bootloader"

0x00100000­0x00500000 : "kernel"

0x00500000­0x02d00000 : "root"

0x02d00000­0x03c00000 : "User"

usbmon: debugfs is not available

116x: driver isp116x­hcd, 05 Aug 2005

s3c2410­ohci s3c2410­ohci: S3C24XX OHCI

s3c2410­ohci s3c2410­ohci: new USB bus registered, assigned bus number 1

s3c2410­ohci s3c2410­ohci: irq 42, io mem 0x49000000

usb usb1: Product: S3C24XX OHCI

usb usb1: Manufacturer: Linux 2.6.14.1 ohci\_hcd

usb usb1: SerialNumber: s3c24xx

hub 1­0:1.0: USB hub found

hub 1­0:1.0: 2 ports detected

sl811: driver sl811­hcd, 19 May 2005

usbcore: registered new driver cdc\_acm

drivers/usb/class/cdc­acm.c: v0.23:USB Abstract Control Model driver for USB modems and ISDN

adaptesdrivers/usb/class/bluetty.c: USB Bluetooth support registered

usbcore: registered new driver bluetty

drivers/usb/class/bluetty.c: USB Bluetooth tty driver v0.13

usbcore: registered new driver usblp

drivers/usb/class/usblp.c: v0.13: USB Printer Device Class driver

Initializing USB Mass Storage driver...

usbcore: registered new driver usb­storage

USB Mass Storage support registered.

mice: PS/2 mouse device common for all mice

NET: Registered protocol family 2

IP route cache hash table entries: 256 (order: ­2, 1024 bytes)

TCP established hash table entries: 1024 (order: 0, 4096 bytes)

TCP bind hash table entries: 1024 (order: 0, 4096 bytes)

TCP: Hash tables configured (established 1024 bind 1024)

TCP reno registered

TCP bic registered

NET: Registered protocol family 1

NET: Registered protocol family 17

Reading data from NAND FLASH without ECC is not recommended

VFS: Mounted root (cramfs filesystem) readonly.

Mounted devfs on /dev

Freeing init memory: 104K

Reading data from NAND FLASH without ECC is not recommended

mount /etc as ramfs

re­create the /etc/mtab entries

­­­­­­­­­­­­mount /dev/shm as tmpfs

­­­­­­­­­­­­mount /proc as proc

­­­­­­­­­­­­mount /sys as sysfs

init started:  BusyBox v1.1.3 (2006.07.03­03:43+0000) multi­call binary

Starting pid 28, console /dev/tts/0: '/etc/init.d/rcS'

in /etc/init.d/rcS

­­­­­­­­­­­­­/sbin/ifconfig eth0 192.168.1.5

Please press Enter to activate this console.

#

## （四）、根文件系统移植

#### 1、根文件系统移植的过程

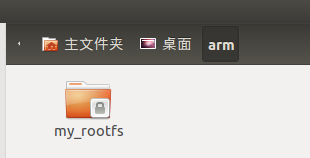
1.1构造目标板的根目录及文件系统

建立一个目标板的空根目录

我们将在这里构建构建根文件系统，创建基础目录结构. 存放交叉编译后生成的目标应用程序

（BUSYBOX，TINYLOGIN），存放库文件等。





1.2 在 my\_rootfs 中建立Linux目录树

[arm@localhostmy\_rootfs]#mkdir bin dev etc home lib mnt proc sbin sys tmp root usr

[arm@localhost my\_rootfs]#mkdir mnt/etc

[arm@localhost my\_rootfs]#mkdir usr/bin usr/lib usr/sbin

[arm@localhost my\_rootfs]#touch linuxrc

[arm@localhost my\_rootfs]#tree

|­­ bin

|­­ dev

|­­ etc

|­­ home

|­­ lib

|­­ linuxrc /\* 此文件为启动脚本，是一 shell 脚本文件\*/

|­­ mnt

|   `­­ etc

|­­ proc

|­­ sbin

|­­ sys

|­­ tmp

|­­ root

`­­ usr

|­­ bin

|­­ lib

`­­ sbin

注意的是：etc目录存放配置文件，这个目录通常是需要修改的，所以在 linuxrc脚本当中将 etc目录挂载为 ramfs 文件系统，然后将 mnt/etc目录中的所有配置文件拷贝到 etc目录当中，这在下一节的 linuxrc脚本文件当中会有体现。

1.3 创建linuxrc文件

[arm@localhost my\_rootfs]#vi linuxrc

#!/bin/sh

#挂载/etc 为 ramfs, 并从/mnt/etc 下拷贝文件到/etc目录当中

echo "mount /etc as ramfs"

/bin/mount ­n ­t ramfs ramfs /etc

/bin/cp ­a /mnt/etc/\* /etc

echo "re­create the /etc/mtab entries"

# re­create the /etc/mtab entries

/bin/mount ­f ­t cramfs ­o remount,ro /dev/mtdblock/2 /

#mount some file system

echo "­­­­­­­­­­­­mount /dev/shm as tmpfs"

/bin/mount ­n ­t tmpfs tmpfs /dev/shm

#挂载/proc为 proc文件系统

echo "­­­­­­­­­­­­mount /proc as proc"

/bin/mount ­n ­t proc none /proc

#挂载/sys 为 sysfs 文件系统

echo "­­­­­­­­­­­­mount /sys as sysfs"

/bin/mount ­n ­t sysfs none /sys

exec /sbin/init

2. 修改权限

[arm@localhost my\_rootfs]#chmod 775 linuxrc

[arm@localhost my\_rootfs]#ls linuxrc ­al

­rwxrwxr­x  1 root root  533 Jun  4 11:19 linuxrc

当编译内核时，指定命令行参数如下

Boot options  ­­­> Default kernel command string: 我的命令行参数如下

noinitrd root=/dev/mtdblock2 init=/linuxrc console=ttySAC0,115200

其中的 init 指明 kernel 执行后要加载的第一个应用程序，缺省为/sbin/init，此处指定为/linuxrc

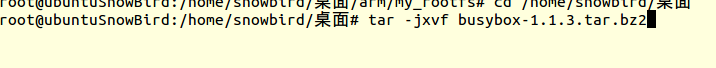
#### 2、移植 Busybox

2.1下载 busybox

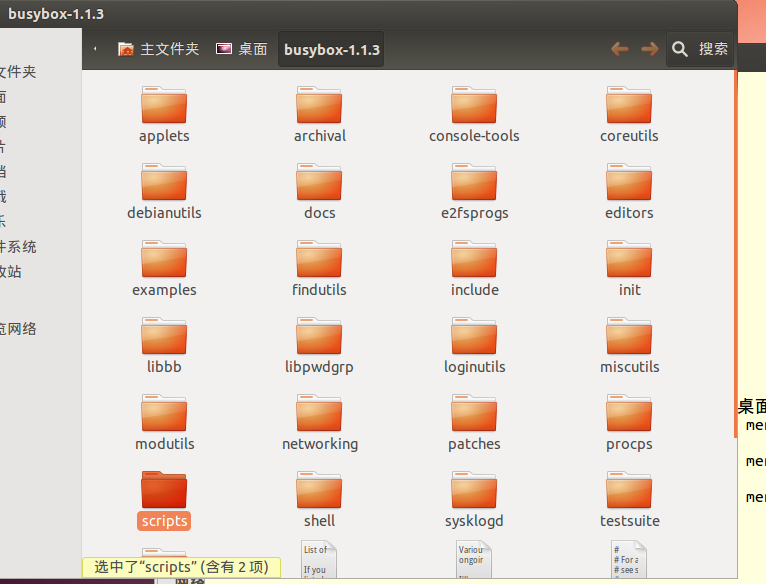
网址：<https://busybox.net/downloads/>



并解压：



2.2进入解压后的目录，配置 Busybox



执行make menuconfig进入图形界面：



进行如下的设置：

Busybox Settings  ­­­>

General Configuration  ­­­>

[\*] Support for devfs

Build Options  ­­­>

[\*] Build BusyBox as a static binary (no shared libs)

/\* 将 busybox 编译为静态连接，少了启动时找动态库的麻烦 \*/

[\*] Do you want to build BusyBox with a Cross Compiler?

(/usr/local/arm/3.3.2/bin/arm­linux­) Cross Compiler prefix

/\* 指定交叉编译工具路径 \*/

Init Utilities  ­­­>

[\*] init

[\*] Support reading an inittab file

    /\* 支持 init 读取/etc/inittab 配置文件，一定要选上 \*/

Shells  ­­­>

Choose your default shell (ash)  ­­­>

  /\* (X) ash 选中 ash，这样生成的时候才会生成 bin/sh 文件

   \* 看看我们前头的 linuxrc脚本的头一句：

   \* #!/bin/sh 是由 bin/sh 来解释执行的

\*/

[\*] ash

Coreutils  ­­­>

[\*] cp

[\*] cat

[\*] ls

   [\*] mkdir

[\*] echo (basic SuSv3 version taking no options)

[\*] env

[\*] mv

[\*] pwd

[\*] rm

[\*] touch

Editors  ­­­>  [\*] vi

Linux System Utilities  ­­­>

[\*] mount

[\*] umount

[\*] Support loopback mounts

[\*] Support for the old /etc/mtab file

Networking Utilities  ­­­>

[\*] inetd

/\*

 \* 支持 inetd 超级服务器

 \* inetd 的配置文件为/etc/inetd.conf文件，

 \* "在该部分的 4: 相关配置文件的创建"一节会有说明

\*/

2.2编译并安装Busybox

[arm@localhost busybox­1.1.3]$ make TARGET\_ARCH=arm CROSS=arm­linux­ \

PREFIX=/home/arm/dev\_home/rootfs/my\_rootfs/ all install

PREFIX指明安装路径：就是我们根文件系统所在路径。

这里需要注意一点的是，只要 install busybox，我们根文件系统下先前建好的 linuxrc就会被覆盖为一同名二进制文件。所以要事先备份我们自己的 linuxrc，在安装完 busybox 后，将 linuxrc复制回去就好。

编译：

[arm@localhost netkit­base­0.17]$ CC=arm­linux­gcc ./configure

[arm@localhost netkit­base­0.17]$ make

2.3拷贝 inetd 到根文件系统的 usr/sbin 目录中

[arm@localhost netkit­base­0.17]$ cp inetd/inetd /home/arm/dev\_home/rootfs/my\_rootfs/usr/sbin/

2.4拷贝 inetd 的配置文件 inetd.conf到根文件系统的/mnt/etc目录中

[arm@localhost netkit­base­0.17]$ cp etc.sample/inetd.conf /home/arm/dev\_home/rootfs/my\_rootfs/mnt/etc

2.5根 据需 要，修改 inetd.conf 配置文件

例如：支持 telnetd 的 inetd.conf 配置文件如下

# <service\_name> <sock\_type> <proto> <flags> <user> <server\_path> <args>telnet stream tcp nowait root /usr/sbin/telnetd

2.6拷贝配置文件

etc.sample目录下有许多网络相关配置文件，其中有一些需要拷贝到根文件系统的 etc目录当中，记录如下：

[arm@localhost netkit­base­0.17]$ cd etc.sample/

[arm@localhost etc.sample]$ cp host.conf /home/arm/dev\_home/rootfs/my\_rootfs/mnt/etc/

[arm@localhost etc.sample]$ cp hosts /home/arm/dev\_home/rootfs/my\_rootfs/mnt/etc/

[arm@localhost etc.sample]$ cp networks /home/arm/dev\_home/rootfs/my\_rootfs/mnt/etc/

[arm@localhost etc.sample]$ cp protocols /home/arm/dev\_home/rootfs/my\_rootfs/mnt/etc/

[arm@localhost etc.sample]$ cp resolv.conf /home/arm/dev\_home/rootfs/my\_rootfs/mnt/etc/

[arm@localhost etc.sample]$ cp services /home/arm/dev\_home/rootfs/my\_rootfs/mnt/etc/

以上重要配置文件说明如下：

host.conf:在系统中同时存在着 DNS域名解析和/etc/hosts 的主机表机制时,由文件/etc/host.conf来说明了解析器

的查询顺序

hosts:记录主机名到IP地址的映射

protocols:记录常用网络协议及端口别名关系，网络应用程序依赖于此文件

resolv.conf:指定 DNS 服务器

services:记录知名网络服务及端口，网络编程依赖于此文件

2.7建立根目录文件系统包

①建立 CRAMFS 包

②下载 cramfs工具

在http://prdownloads.sourceforge.net/cramfs/cramfs­1.1.tar.gz下载源代码包.

把下载包拷贝到 dev\_home/tools 下.

[arm@localhost tools]$tar ­xzvf cramfs­1.1.tar.gz

[arm@localhost tools]$cd cramfs­1.1

[arm@localhost tools]$make

[arm@localhost tools]$su root

[root@localhost tools]$cp mkcramfs /usr/bin

[arm@localhost tools]$exit

注意：如果你的系统中已经安装了 mkcramfs工具, 则在/usr/bin 目录下是一个软 link, 请先删除该文件之后, 再拷

贝该 mkcramfs到/usr/bin 下.

2.8制作 cramfs 包

[arm@localhost tools]$mkcramfs my\_rootfs my\_rootfs.cramfs

2.9写 cramfs 包到 Nand Flash

[arm@localhost tools]$su root

[root@localhost tools]$cp my\_rootfs.cramfs /tftpboot/

打开 minicom, 进行 ARM板的终端模式:

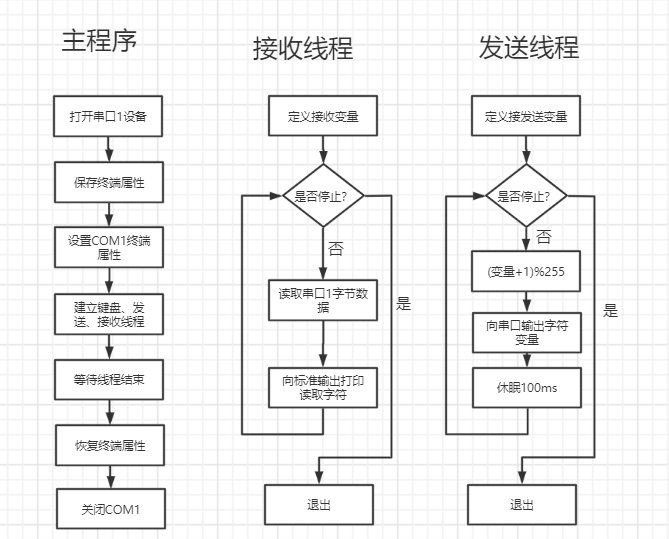
CRANE2410 #

## （五）、串行端口程序设计与移植

#### 1、串行端口程序设计与移植过程

1.1整体流程

Linux操作系统从一开始就对串行口提供了很好的支持，为进行串行通讯提供了大量的函数，我们的实验主要是为掌握在Linux中进行串行通讯编程的基本方法。本实验的程序流程图如下:



1.2程序设计

#include <termios.h>

#include <stdio.h>

#include <unistd.h>

#include <fcntl.h>

#include <sys/signal.h>

#include <pthread.h>

#define BAUDRATE B115200#define COM1 "/dev/ttySo"t#tdefine COM2 "/dev/ttyS1"

#define ENDMINITERM 27/\* ESC to quit miniterm\*/#define FALSE 0

#define TRUE 1

volatile int STOP=FALSE;volatile int fd;

void child\_handler(int s)

{

printf("stop!!! \n");STOP=TRUE;

}

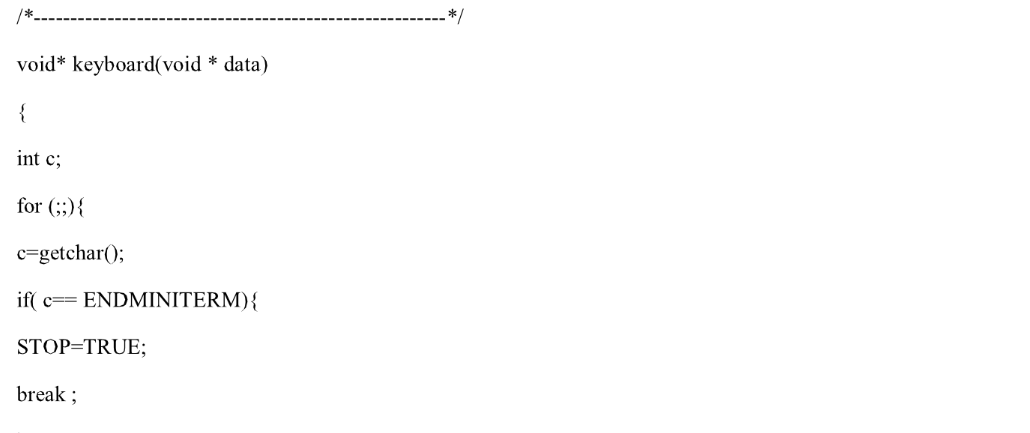
void\* keyboard(void \* data){

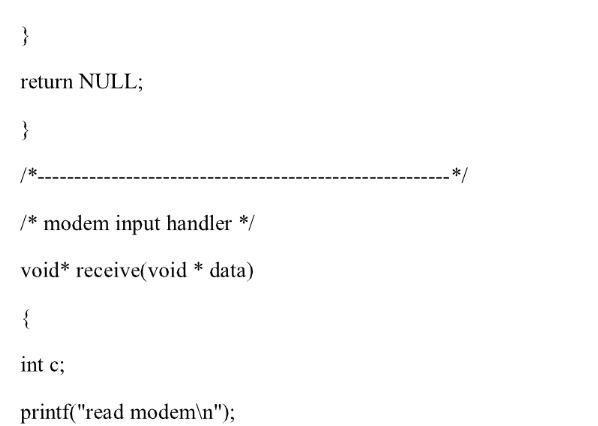
int c;for (;;){c=getchar();

if( c==ENDMINITERM){STOP=TRUE;

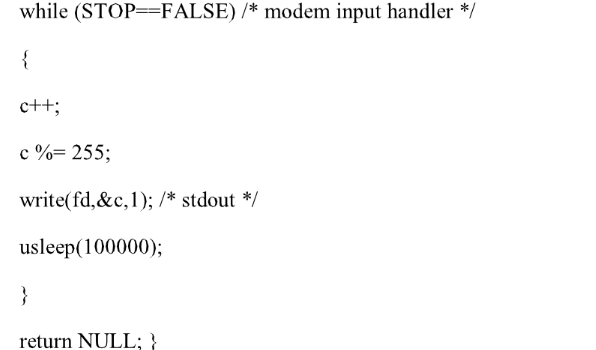
break ;

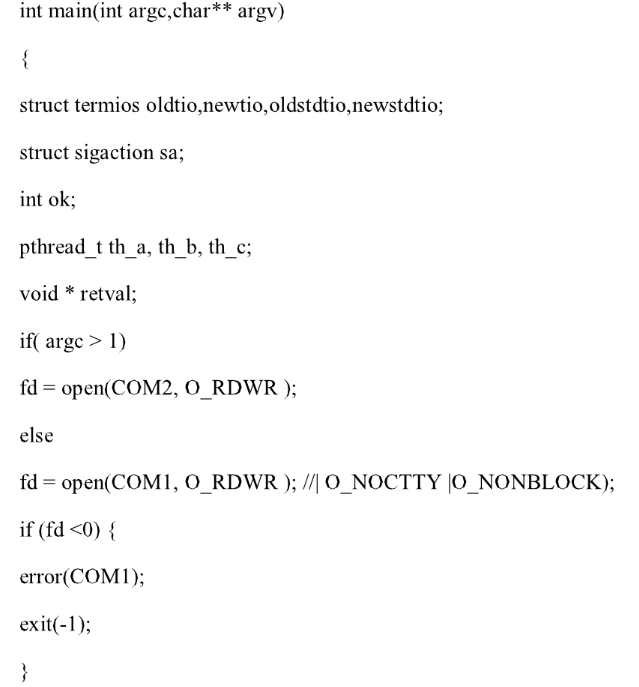
}

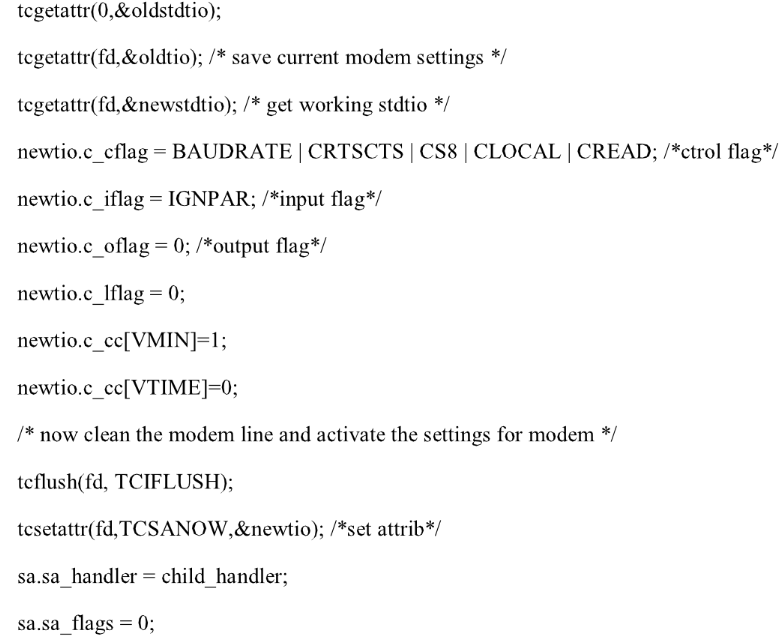


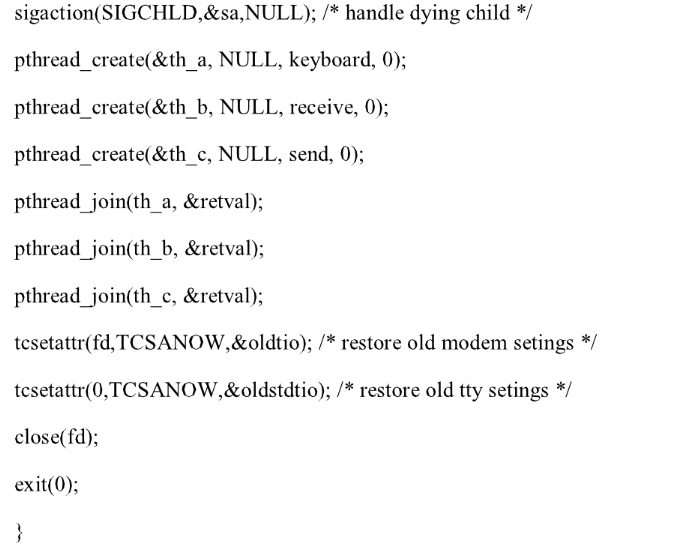












1.3编译应用程序

运行make产生term可执行文件

[root@zxt root]# cd /arm2410s/exp/basic/03\_tty/

[root@zxt 03\_tty]# make

armv4l-unknown-linux-gcc -c -o term.o term.c

armv4l-unknown-linux-gcc -o ../bin/term term.o -lpthread

armv41-unknown-linux-gcc -o term term.o -lpthread

[root@zxt 03\_tty]# 1s

Makefile Makefile.bak term term.c term.o tty.c

1.4下载调试

切换到minicom终端窗口，使用NFS mount开发主机的/arm2410s 到/host目录。进入exp\basic\03\_tty 目录，运行term，观察运行结果的正确性。

[root@zxt root]# minicom

[/mnt/yaffs]mount -t nfs -o nolock 192.168.0.56:/arm2410s /host[/mnt/yaffs]cd /host/exp/basic/03\_tty/

[/host/exp/basic/03\_tty]./term

read modem

send data

123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVwx

由于内核已经将串口1作为终端控制台，所以可以看到term发出的数据，却无法看到开发主机发来的数据，可以使用另外一台主机连接串口2进行收发测试。Ctrl+c可使程序强行退出。

#### 2、存在的问题及解决方案

在执行./term时出现下面的错误，可以通过我们前文提到的方法建立一个连接来解决。

/dev/ttyS0: No such file or directory

解决方法:

[/mnt/yaffs] cd /dev

[ / dev] ln -sf /dev/tts/0 ttyso（注意首字母是l，不是数字1)

# 五、实训总结

经过了为期一周的arm实训，学到了许多，也懂得了许多。这一周的实训说长也不长，说短也不短，这次的实训是对一学期的arm学习的最终考研与考核。

在这次实训中一次完成了U-boot移植、内核移植、根文件系统移植、串行端口程序设计与移植这四大项目，每一个小项目的完成都费了不小的周折，每一项的完成都不是简简单单就可以实现的，因为这些实训的内容并不只是课堂上所讲的那些理论，还有不同理论知识之间的相互联系与先后的逻辑关系。需要按照一定的流程与规则来一步一步的去实现一个个的小任务。

实训不只是简简单单的去实训，更是一个不断学习提高自己的机会。在这次实训中我知道了uboot属于bootloader的一种，是用来引导启动内核的，它的最终目的就是从flash中读出内核，放到内存中，启动内核。此次实验，分析了部分代码。实际操作了U-Boot在 NAND Flash上启动，并在相应的flash上进行烧写。

在这次实验中，还是犯了一些教条主义错误，“纸上得来终觉浅，绝知此事要躬行”，关于能不能在NOR Flash 上烧写一些纠结了不少时间，在网上查了一些资料，再看2410开发资料，得知在本次实验的板子上是没有NOR Flash的。

内核移植过程中上来就遇到了一个问题，就是在内核版本选择上纠结不定，最终选择了linux-2.6.14版本。这让我知道了我对各个内核版本的不了解，也知道了自己在上机实操方面还是经验欠缺。

此次实验有还有不足之处,就是对核心板的相关重要的寄存控制器了解不是很充分，硬件与软件怎么进行交互的具体原理还没有弄清楚。

但是，路漫漫其修远兮，吾将上下而求索!