嵌入式 Linux 学习手册 V0.4

[u-boot-2012.10/Linux3.6.7/QT4.4.3]

[2013-1-7]

朱兆祺

目 录

E-mail : jxlgzzq@163.com

纵观本书

本书第 1 章为 Makefile 基础知识。

本书第 2 章至第 8 章为 u-boot-2012.10 移植。

本书第 9 章至第 13 章为 Linux3.6.7 移植。

本书第 14 章至第 18 章为 Linux 设备驱动。

本书第 19 章至（待更新・・・）为 QT4.4.3 移植。

今天是 2012 年 12 月 10 日，好快了，不到半年的时间，我就要毕业了，就要离开这个梦想起飞的地方。

自从 11 月离开周立功单片机公司至今也有 1 个多月了，沉淀了自己的思想，开启了新的历程。其实还是很感谢周立功公司，正因为 7 月至 11 月 4 个月的夏令营和卓越班培训，才有我之前所写的《攻破 C 语言笔试与机试难点》这本书。正因为有了这本书，才促使我往下写《嵌入式 Linux 学习手册》的动力。

本书以一个学生的角度学习 OK6410 开发平台，以一个学生的视角去写一本适用于初学者的学习手册。在本人学习的过程中，很感谢各位网友，给我提供了较多学习思想。也很感谢现在的深圳亚泰光电技术有限公司，给了我充足的时间做毕业设计，才让我有时间写下这本适用于初学者的书籍。

(由于网友迫切需要资料，先将未完成的书稿公布，真切希望网友找出本书的错误，发送至我邮箱：jxlgzzq@163.com。未完待续……)

这是 2013 年第一次更新，也快是寒假了，希望这本书能给嵌入式领域的学子带来一点点帮助。之前网友给我提出了很多建议，谢谢大家。这些我会在完成本书初稿之后一并修改。宁静致远第四次更新于 2013 年 1 月 7 日。

宁静致远工作室

朱兆祺

2012 年 12 月 10 日

第 1 章 Makefile 的基本知识

在移植 uboot 之前，先熟悉一下将在 uboot 中用到的 Makefile 的一些基本语法。所谓“磨刀不误砍柴工”嘛。有兴趣的同学也可读阅《GNU make》，里面把 Makefile 讲得比较深透。

1.1 Makefile 规则

一个语句由目标、依赖条件、指令组成。如程序清单 1.1 所示。

程序清单 1.1 Makefile 基本组成

```
smdk6400_config  :  unconfig
    @mkdir -p $(obj)include $(obj)board/samsung/smdk6400
```

smdk6400_config：目标；

unconfig：先决条件；

@mkdir -p $(obj)include $(obj)board/samsung/smdk6400：命令。这里特别注意，“@”前面是 Tab 键，并且必须是 Tab 键，而不能是空格。

目标和先决条件是依赖关系，目标是依赖于先决条件生成的。

1.2 Makefile 变量

1.2.1 变量的引用方式

变量的引用方式是：“$（VARIABLE_NAME）”或者“${ VARIABLE_NAME }”来引用一个变量的定义。例如：“$(obj) ”或者“${obj}”就是取变量“obj”的值。如程序清单 1.2 所示。

程序清单 1.2 变量的引用

```
obj         := $(OBJTREE)/
OBJTREE       := $(if $(BUILD_DIR),$(BUILD_DIR),$(CURDIR))
export BUILD_DIR=/tmp/build
```

$(if $(BUILD_DIR),$(BUILD_DIR),$(CURDIR))的含义：如果“BUILD_DIR”变量值不为空，则将变量“BUILD_DIR”指定的目录作为一个子目录；否则将目录“CURDIR”作为一个子目录。详细请参考《GNU make》。

1.2.2 递归方式扩展的变量

这类变量的定义是通过“=”和“define”来定义的。这种类型的变量是其它版本的 make 所支持的类型。我们可以把这种类型的变量称为“递归展开”式变量。

其优点是： 这种类型变量在定义时，可以引用其它的之前没有定义的变量（可能在后续部分定义，或者是通过 make 的命令行选项传递的变量）。如程序清单 1.3 所示。

程序清单 1.3 递归方式扩展的变量

```
student = lilei
CLASS = $(student) $(teacher)
teacher = yang
```

```
all:
    @echo $(CLASS)
```

按照递归扩展的变量规则，输出是：lilei　yang。也就是说虽然 teacher 是在 CLASS 语句之后，但是还是会被替换掉。

其缺点是：

1. 使用此风格的变量定义，可能会由于出现变量的递归定义而导致 make 陷入到无限的变量展开过程中，最终使 make 执行失败。如程序清单 1.4 所示。

程序清单 1.4　递归扩展的变量陷入循环

```
x = $(y)
y = $(z)
z = $(x)
```

这样的话会使得 Makefile 出错，因为都最终引用了自己。

2. 第二个缺点：这种风格的变量定义中如果使用了函数，那么包含在变量值中的函数总会在变量被引用的地方执行（变量被展开时）。

1.2.3 直接展开式变量

为了避免“递归展开式”变量存在的问题和不方便。GNU make 支持另外一种风格的变量，称为“直接展开”式。这种风格的变量使用“:=”定义。在使用“:=”定义变量时，变量值中对其他量或者函数的引用在定义变量时被展开（对变量进行替换）。如程序清单 1.5 所示。

程序清单 1.5　直接展开式变量

```
X := student
Y := $(X)
X := teacher
all:
    @echo $(X) $(Y)
```

这里的输出是：teacher　student。

此风格变量在定义时就完成了对所引用变量和函数的展开，因此不能实现对其后定义变量的引用。

1.2.4 条件赋值

只有此变量在之前没有赋值的情况下才会对这个变量进行赋值。如程序清单 1.6 所示。

程序清单 1.6　条件赋值

```
X := student
X ?= teacher
all:
    @echo $(X)
```

由于 X 在之前被赋值了，所以这里的输出是 student。

1.2.5 变量的替换引用

对于一个已经定义的变量，可以使用“替换引用”将其值中的后缀字符（串）使用指定的字符（字符串）替换。格式为“$(VAR:A=B)”（或者“${VAR:A=B}”），意思是，替换变量“VAR”中所有“A”字符结尾的字为“B”结尾的字。“结尾”的含义是空格之前（变量值多个字之间使用空格分开）。而对于变量其它部分的“A”字符不进行替换。如程序清单 1.7 所示。

程序清单 1.7 变量的替换引用

```
X := fun.o main.o
Y := $(X: .o=.c)
all:
    @echo $(X) $(Y)
```

这里特别注意的是$(X: .o=.c)的“=”两边不能有空格。

这里的输出是：fun.o main.o fun.c main.c

1.2.6 追加变量值

一个通用变量在定义之后的其他一个地方，可以对其值进行追加。这是非常有用的。我们可以在定义时（也可以不定义而直接追加）给它赋一个基本值，后续根据需要可随时对它的值进行追加（增加它的值）。在 Makefile 中使用“+=”（追加方式）来实现对一个变量值的追加操作。如程序清单 1.8 所示。

程序清单 1.8 追加变量值

```
X = fun.o main.o
X += sub.o
all:
    @echo $(x)
```

这里输出是：fun.o main.o sub.o

1. 如果被追加值的变量之前没有定义，那么，“+=”会自动变成“=”，此变量就被定义为一个递归展开式的变量。如果之前存在这个变量定义，那么“+=”就继承之前定义时的变量风格。

2. 直接展开式变量的追加过程：变量使用“:=”定义，之后“+=”操作将会首先替换展开之前此变量的值，尔后在末尾添加需要追加的值，并使用“:=”重新给此变量赋值。实际的过程如下所示。

```
variable := value
variable += more
```

等效于：

```
variable := value
variable := $(variable) more
```

3. 递归展开式变量的追加过程：一个变量使用“=”定义，之后“+=”操作时不对之前此变量值中的任何引用进行替换展开，而是按照文本的扩展方式（之前等号右边的文本未发生变化）替换，尔后在末尾添加需要追加的值，并使用“=”给此变量重新赋值。

```
variable = value
```

```
variable += more
```

等效于 ：

```
temp = value
variable = $(temp) more
```

1.3 Makefile 函数

1.3.1 Addprefix

$(addprefix PREFIX,NAMES…)

函数名称：加前缀函数—addprefix。

函数功能：为“NAMES…”中的每一个文件名添加前缀“PREFIX”。参数“NAMES…”是空格分割的文件名序列，将“SUFFIX”添加到此序列的每一个文件名之前。

返回值：以单空格分割的添加了前缀“PREFIX”的文件名序列。

```
$(addprefix src/,foo bar)
```

返回值为“src/foo src/bar”。

OBJS := $(addprefix $(obj),$(OBJS))执行完成之后即为“$(obj)/ $(CPUDIR)/start.o”。

.....

(待更新)

第 2 章 u-boot-2012.10 移植之准备工作

2.1 安装交叉编译工具

版本：arm-linux-gcc 4.4.1

环境：ubuntu10.04.4（迄今为止，个人认为最为稳定和健全的版本）

2.1.1 安装步骤

1. 在/usr/local 下面创建一个文件夹：mkdir arm，将 arm-linux-gcc 4.4.1 放在 arm 文件夹里面。然后解压缩，命令根据压缩包的不同而不同。
2. 添加环境变量，vim /etc/profile。
3. 在最后一行添加：export PATH=$PATH:/usr/local/arm/4.4.1/bin。
4. 退出执行命令：source /etc/profile。
5. 检测安装是否成功：arm-linux-gcc -v ；如果成功，输出最后一行则会提示：gcc version 4.4.1 (Sourcery G++ Lite 2009q3-67)。

2.2 Linux 操作基本命令

1. 建立目录

当我们工作的需要，建立一个目录的时候，我们可以使用“mkdir”命令来建立一个目录，如： mkdir myfile。

2. 删除目录

如果这个目录不需要了，我们可以使用“rmdir”命令来删除一个目录，用法如： rmdir myfile。

3. 复制文件并且重命名

将 s3c6400.h 复制一份并且重命名为 s3c6410.h，如：cp s3c6400.h s3c6410.h。

4. 解压、打包、压缩

.tar

解包：tar xvf FileName.tar

打包：tar cvf FileName.tar DirName

（注：tar 是打包，不是压缩！）

.gz

解压 1：gunzip FileName.gz

解压 2：gzip -d FileName.gz

压缩：gzip FileName

.tar.gz 和 .tgz

解压：tar zxvf FileName.tar.gz

压缩：tar zcvf FileName.tar.gz DirName

.bz2

解压 1：bzip2 -d FileName.bz2

解压 2：bunzip2 FileName.bz2

压缩： bzip2 -z FileName

.tar.bz2

解压：tar jxvf FileName.tar.bz2

压缩：tar jcvf FileName.tar.bz2 DirName

.bz

解压 1：bzip2 -d FileName.bz

解压 2：bunzip2 FileName.bz

压缩：未知

.tar.bz

解压：tar jxvf FileName.tar.bz

压缩：未知

.Z

解压：uncompress FileName.Z

压缩：compress FileName

.tar.Z

解压：tar Zxvf FileName.tar.Z

压缩：tar Zcvf FileName.tar.Z DirName

.zip

解压：unzip FileName.zip

压缩：zip FileName.zip DirName

2.3 删除与修改

移植平台：s3c6410

2.3.1 删除与 s3c6410 无关文件

安装 tree，命令：sudo apt-get install tree 。完成之后使用 tree –L 1 查看第一级目录下有什么，初步观察下 u-boot 里面到底有什么东西。如图 2. 1 所示。

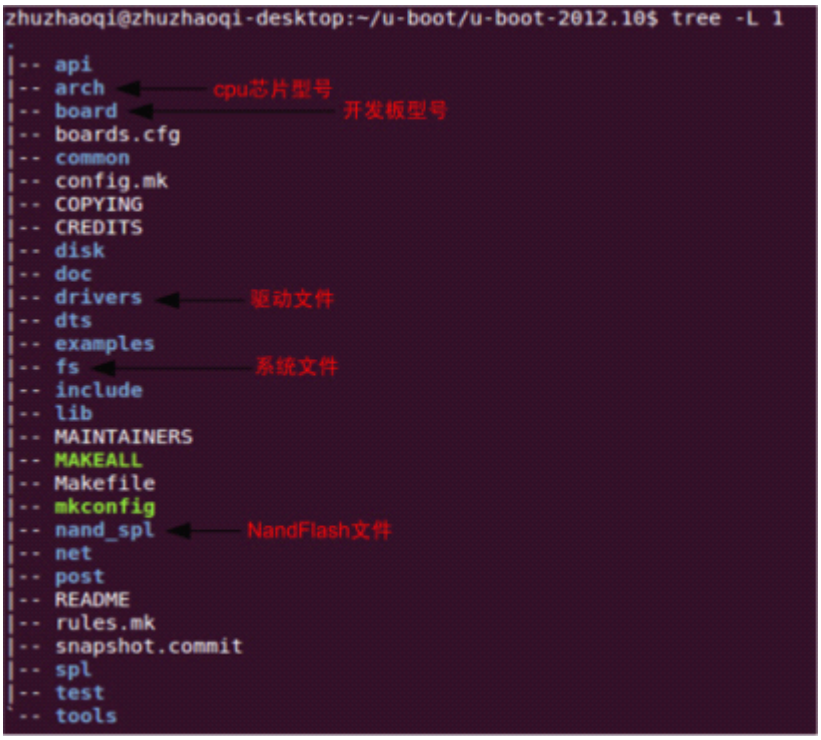


图 2.1 uboot 框架

1. 进入\arch，对于文件夹，除了 arm 之外，全部删除。
2. 进入\arch\arm\cpu，保留 arm1176 和 u-boot.lds，其余文件夹可删除。
3. 进入\arch\arm\cpu\arm1176，文件夹有 3 个，保留 s3c64XX 即可，删除其他 2 个。
4. 进入\arch\arm\include\asm，仅对 arch-*文件夹而言，除了 arch-s3c64XX 之外全部删除。
5. 进入\board，除\samsung 以外的文件夹全部删除。
6. 进入\board\samsung，smdk*中，除了 smdk6400，其余文件夹可删除。

上面操作可操作，亦可不操作，我删除与 s3c6410 部分无关文件是为了避免寻找文件时繁杂。

2.4 初步修改文件

本次初步操作是为了将 6400 变成 6410。

1. 在\board\samsung 下，新建一个 smdk6410，将 smdk6400 下的所有文件拷贝到 smdk6410 下面。将 smdk6410 下的 smdk6400.c 文件修改成 smdk6410.c，smdk6400_nand_spl.c 文件修改成 smdk6410_nand_spl.c。将 smdk6410 文件夹下面的 Makefile 中的：
   COBJS-y := smdk6400.o 修改成 COBJS-y := smdk6410.o。
2. 在\nand_spl\board\samsung 下，新建一个 smdk6410，将 smdk6400 下的所有文件拷贝到 smdk6410 下面。将 smdk6410 下的 Makefile 中的：
   COBJS = nand_boot.o nand_ecc.o s3c64xx.o smdk6400_nand_spl.o nand_base.o 修改为
   COBJS = nand_boot.o nand_ecc.o s3c64xx.o smdk6410_nand_spl.o nand_base.o;
   @ln -s $(TOPDIR)/board/samsung/smdk6400/lowlevel_init.S $@修改成
   @ln -s $(TOPDIR)/board/samsung/smdk6410/lowlevel_init.S $@;

$(obj)smdk6400_nand_spl.c:修改成$(obj)smdk6410_nand_spl.c: ；
@ln -s $(TOPDIR)/board/samsung/smdk6400/smdk6400_nand_spl.c $@修改成
@ln -s $(TOPDIR)/board/samsung/smdk6410/smdk6410_nand_spl.c $@。

3. 在\include\configs 下，将 smdk6400.h 拷贝一份重命名为 smdk6410.h。将 CONFIG_S3C6400 修改为 CONFIG_S3C6410，将 CONFIG_SMDK6400 修改为 CONFIG_SMDK6410。
4. 在\arch\arm\include\asm\arch-s3c64xx 下，将 s3c6400.h 拷贝一份重命名为 s3c6410.h。打开 s3c6410.h，将#ifndef __S3C6400_H__和#define __S3C6400_H__修改成#ifndef __S3C6410_H__和#define __S3C6410_H__，其余稍后修改。
5. 进入\arch\arm\cpu\arm1176\s3c64xx，打开 Makefile，将 CONFIG_S3C6400 修改成 CONFIG_S3C6410。
6. 进入\board\samsung\smdk6410，打开 smdk6410.c 和 lowlevel_init.s；
进入\arch\arm\cpu\arm1176\s3c64xx，打开 cpu_init.s、reset.s、speed.c 和 timer.c；
进入\drivers\mtd\nand，打开 s3c64xx.c；
进入\drivers\serial，打开 s3c64xx.c；
进入\drivers\usb\host，打开 s3c64xx-hcd;
将上面文件中的#include <asm/arch/s3c6410.h>修改成#include <asm/arch/s3c6410.h>。
7. 修改最顶层的 Makefile，添加编译工具：将 CROSS_COMPILE 修改成 CROSS_COMPILE=/usr/local/arm/4.4.1/bin/arm-linux-；在 arm1176 部分，作如下修改。

```
smdk6410_noUSB_config   \
smdk6410_config  :   unconfig
    @mkdir -p $(obj)include $(obj)board/samsung/smdk6410
    @mkdir -p $(obj)nand_spl/board/samsung/smdk6410
    @echo "#define CONFIG_NAND_U_BOOT" > $(obj)include/config.h
    @echo "CONFIG_NAND_U_BOOT = y" >> $(obj)include/config.mk
    @if [ -z "$(findstring smdk6410_noUSB_config,$@)" ]; then
        \
        echo "RAM_TEXT = 0x57e00000" >>
$(obj)board/samsung/smdk6410/config.tmp;\
    else                                \
        echo "RAM_TEXT = 0xc7e00000" >>
$(obj)board/samsung/smdk6410/config.tmp;\
    fi
    @$(MKCONFIG) smdk6410 arm arm1176 smdk6410 samsung s3c64xx
    @echo "CONFIG_NAND_U_BOOT = y" >> $(obj)include/config.mk
```

此时 make smdk6410_config，接着 make，如果上面操作没有错误的话，应该是编译成功，但是此时还是基于 s3c6400，只是披着 s3c6410 皮的 s3c6400 罢了。至此为止，u-boot 版本号为:u-boot-2012.10.1

注：如果你是初步接触 u-boot 或者不想太麻烦，可以不进行本章节的 2.3 操作，就是用

s3c6400 的原版，到时候全部修改完成就是披着 s3c6400 皮的 s3c6410 了！

这样本章节你只需修改一个地方：在最顶层的 Makefile 中，添加编译工具：将 CROSS_COMPILE 修改成 CROSS_COMPILE=/usr/local/arm/4.4.1/bin/arm-linux-。

经过本章节的操作，进行编译。

```
make smdk6410_config
......
make all
```

等待编译，结果如图 2.2 所示。

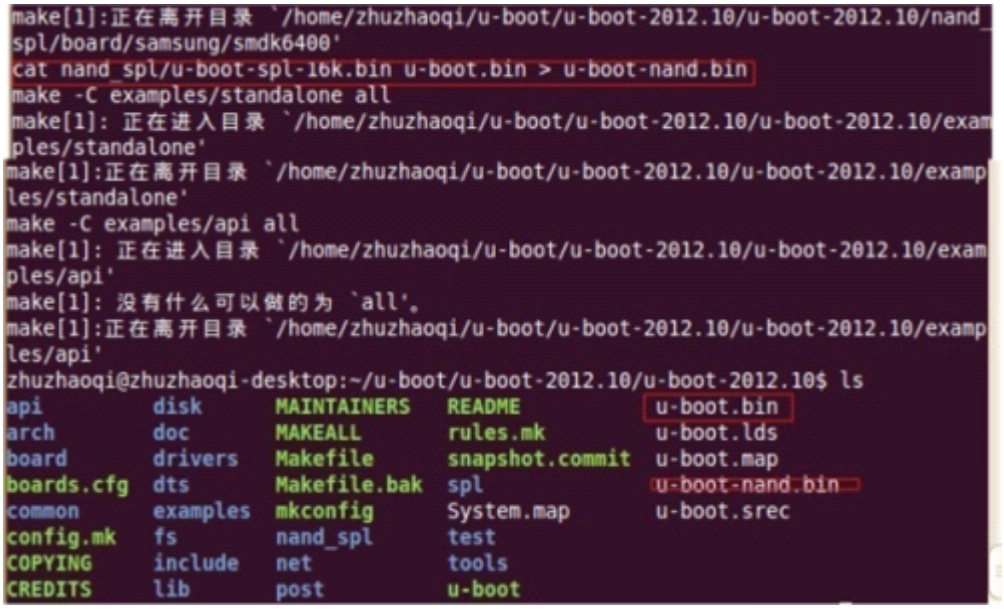


图 2.2 编译结果

2.5 CRT 工具

SecureCRT_CN 工具是一个很方便查看的窗口，即可连通 ubuntu、即可作为 OK6410 串口输出查看窗口，还可以很方便的传输文件。

开启 ubuntu 上的 ssh 功能，先安装，安装后就自动开启了。

```
sudo apt-get install openssh-server openssh-client
```

接着安装 CRT 工具。

第 3 章 u-boot-2012.10 移植之 start

u-boot 版本不断更新，但是根基是万不能变，所以 u-boot 还是从 start.s 中开始执行。打开最顶层的 Makefile，最先构建的也是 start.o，如下所示。

```
216  OBJS  = $(CPUDIR)/start.o
```

3.1 硬件设备初始化

在 start.s 中第一行代码，一上电，系统复位。

```
.globl _start
_start: b reset
```

.globl 如果一个符号没有用.globl 声明，就表示这个符号不会被链接器用到。

b 是跳转指令，ARM 的跳转指令可以从当前指令向前或者向后的 32MB 的地址空间跳转，这类跳转指令有以下 4 种：

1) B 跳转指令
2) BL 带返回的跳转指令
3) BX 带状态切换的跳转指令
4) BLX 带返回和状态切换的跳转指令

```
_start: b reset
```

而这句跳转时，PC 寄存器的值将不会保存到 LR 寄存器中。

回到 reset，这是 s3c6410 一上电做的第一件事情。如下所示。

1. 将 cpu 的工作模式设置为管理模式

```
reset:
    /*
     * set the cpu to SVC32 mode
     */
    mrs r0, cpsr
    bic r0, r0, #0x3f
    orr r0, r0, #0xd3
    msr cpsr, r0
```

接着进入 cpu_init_crit，即 cpu 初始化阶段。

2. 初始化 CACHE

```
    mov r0, #0
    mcr p15, 0, r0, c7, c7, 0   /* flush v3/v4 cache */
    mcr p15, 0, r0, c8, c7, 0   /* flush v4 TLB */
```

3. 初始化 MMU

```
    mrc p15, 0, r0, c1, c0, 0
```

```
    bic r0, r0, #0x00002300 @ clear bits 13, 9:8 (--V- --RS)
    bic r0, r0, #0x00000087 @ clear bits 7, 2:0 (B--- -CAM)
    orr r0, r0, #0x00000002 @ set bit 2 (A) Align
    orr r0, r0, #0x00001000 @ set bit 12 (I) I-Cache
```

4. 初始化外设，指明外设的基地址

```
#ifdef CONFIG_PERIPORT_REMAP
    /* Peri port setup */
    ldr r0, =CONFIG_PERIPORT_BASE
    orr r0, r0, #CONFIG_PERIPORT_SIZE
    mcr p15,0,r0,c15,c2,4
#endif
```

接下来是执行带返回跳转指令：bl lowlevel_init，进入 lowlevel_init(位于 board\samsung\smdk6410\lowlevel_init.s)。

5. LED 初始化

原始是与 s3c6400 配置，如下所示。

```
    ldr r0, =ELFIN_GPIO_BASE
    ldr r1, =0x55540000
    str r1, [r0, #GPNCON_OFFSET]

    ldr r1, =0x55555555
    str r1, [r0, #GPNPUD_OFFSET]

    ldr r1, =0xf000
    str r1, [r0, #GPNDAT_OFFSET]
```

这里应该改成与 s3c6410 相适应的配置。由于本移植是基于飞凌公司的 OK6410，所以由 OK6410 开发板的原理图。如图 3. 1 所示。

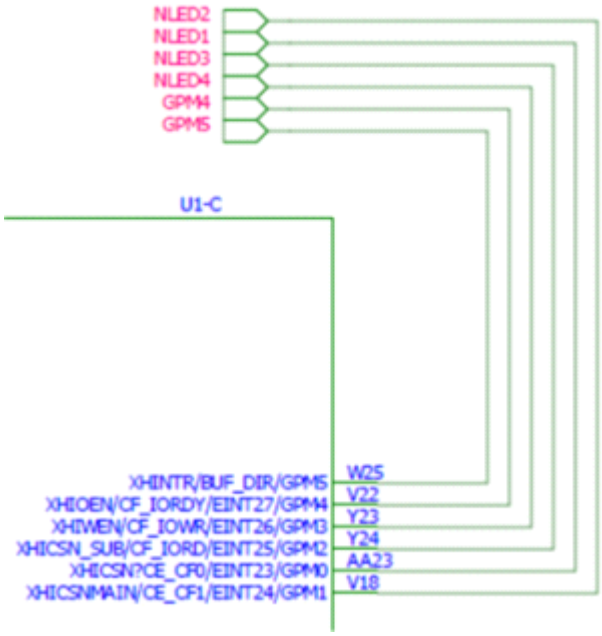


图 3. 1 LED 原理图

根据 s3c6410 用户手册中的端口 M 控制寄存器章节可以对程序作出如下修改。

```
/* LED on only #8 */
ldr r0, =ELFIN_GPIO_BASE
ldr r1, =0x00111111
str r1, [r0, #GPMCON_OFFSET]

ldr r1, =0x00000555
str r1, [r0, #GPMPUD_OFFSET]
 /* all of LEDs are power on */
ldr r1, =0x000f
str r1, [r0, #GPMDAT_OFFSET]
```

根据需要，LED 测试自行修改：

```
/* LED test */
ldr r0, =ELFIN_GPIO_BASE
ldr r1, =0x0003
str r1, [r0, #GPMDAT_OFFSET]
```

6. 关闭看门狗

```
ldr r0, =0x7e000000     @0x7e004000
orr r0, r0, #0x4000
mov r1, #0
str r1, [r0]
```

7. 关闭中断

```
/* External interrupt pending clear */
ldr r0, =(ELFIN_GPIO_BASE+EINTPEND_OFFSET)  /*EINTPEND*/
ldr r1, [r0]
str r1, [r0]

ldr r0, =ELFIN_VIC0_BASE_ADDR  @0x71200000
ldr r1, =ELFIN_VIC1_BASE_ADDR  @0x71300000

/* Disable all interrupts (VIC0 and VIC1) */
mvn r3, #0x0
str r3, [r0, #oINTMSK]
str r3, [r1, #oINTMSK]

/* Set all interrupts as IRQ */
mov r3, #0x0
str r3, [r0, #oINTMOD]
str r3, [r1, #oINTMOD]
```

```
    /* Pending Interrupt Clear */
    mov r3, #0x0
    str r3, [r0, #oVECTADDR]
    str r3, [r1, #oVECTADDR]
```

bl system_clock_init，跳转到系统时钟初始化。

8. 系统时钟初始化

程序较长，不加以罗列。系统时钟初始化起始于：

```
system_clock_init:
    ldr r0, =ELFIN_CLOCK_POWER_BASE    /* 0x7e00f000 */
```

……

其中将

```
/* FOUT of EPLL is 96MHz */
    ldr r1, =0x200203
```

修改成：

```
   ldr  r1, =0x80200203
```

我们看到

```
#ifndef CONFIG_S3C6410
    ldr r1, [r0, #OTHERS_OFFSET]
    bic r1, r1, #0xC0
    orr r1, r1, #0x40
    str r1, [r0, #OTHERS_OFFSET]
```

由于这里还是 S3C6410，那么这里肯定要进行修改。我们找到 CONFIG_S3C6400 的宏定义处(在 include\configs\smdk6410.h 中)。

在此就先分析一下 smdk6410.h 这个文件。

```
#define CONFIG_S3C6400     1   /* in a SAMSUNG S3C6400 SoC     */
#define CONFIG_S3C64XX     1   /* in a SAMSUNG S3C64XX Family  */
#define CONFIG_SMDK6400    1   /* on a SAMSUNG SMDK6400 Board  */
```

修改成：

```
#define CONFIG_S3C6410     1   /* in a SAMSUNG S3C6410 SoC     */
#define CONFIG_S3C64XX     1   /* in a SAMSUNG S3C64XX Family  */
#define CONFIG_SMDK6410    1   /* on a SAMSUNG SMDK6410 Board  */
```

……

```
#define CONFIG_SYS_SDRAM_BASE 0x50000000
```

这里 SDRAM 的基地址是：0x50000 0000。这个无需修改。

……

```
/*
```

```
 * Architecture magic and machine type
 */
#define CONFIG_MACH_TYPE      1270
```

这个是 s3c6400 的型号，应该修改为 s3c6410，否则在启动内核的时候会出问题。修改成：

```
#define CONFIG_MACH_TYPE      1626
```

……

```
#define CONFIG_BOOTDELAY   10
```

为了在 u-boot 延时稍长，故修改为 10s。

……

```
/* Monitor Command Prompt    */
#define CONFIG_SYS_PROMPT     "SMDK6400 # "
```

修改成：

```
/* Monitor Command Prompt    */
#define CONFIG_SYS_PROMPT     "zzq6410 >>> "
```

……

……

```
#define CONFIG_SYS_HZ         1000
```

这个时间是在 PCLK = 50MHz,故应该修改为：

```
#define CONFIG_SYS_HZ         1562500
```

……

```
#define CONFIG_NR_DRAM_BANKS  1
/* SDRAM Bank #1 */
#define PHYS_SDRAM_1       CONFIG_SYS_SDRAM_BASE
#define PHYS_SDRAM_1_SIZE  0x08000000    /* 128 MB in Bank #1    */
```

因为 s3c6410 是 256MB 的 SDRAM，故应该修改成：

```
#define CONFIG_NR_DRAM_BANKS  1
/* SDRAM Bank #1 */
#define PHYS_SDRAM_1       CONFIG_SYS_SDRAM_BASE
#define PHYS_SDRAM_1_SIZE  0x10000000    /* 256 MB in Bank #1    */
```

……

```
/* Total Size of Environment Sector */
#define CONFIG_ENV_SIZE       0x4000
```

堆栈的大小要修改为：

```
/* Total Size of Environment Sector */
#define CONFIG_ENV_SIZE       0x80000
```

……

```
#define CONFIG_IDENT_STRING   " for SMDK6400"
```

修改成：

```
#define CONFIG_IDENT_STRING   " for zzq6410"
```

……

```
#define CONFIG_BOOTCOMMAND  "nand read 0x50018000 0x60000 0x1c0000;"
\
             "bootm 0x50018000"
```

修改为：

```
#define CONFIG_BOOTCOMMAND "nand read 0x50018000 0x100000 0x500000;"
\
             "bootm 0x50018000"
```

……

```
#define CONFIG_ENV_OFFSET     0x0040000
```

修改为：

```
#define CONFIG_ENV_OFFSET     0x0080000
```

……

```
/* Offset to RAM U-Boot image */
#define CONFIG_SYS_NAND_U_BOOT_OFFS  (4 * 1024)
/* Size of RAM U-Boot image   */
#define CONFIG_SYS_NAND_U_BOOT_SIZE  (252 * 1024)
```

修改成：

```
/* Offset to RAM U-Boot image */
#define CONFIG_SYS_NAND_U_BOOT_OFFS  (16 * 1024)
/* Size of RAM U-Boot image   */
#define CONFIG_SYS_NAND_U_BOOT_SIZE  (496 * 1024)
```

……

```

/* NAND chip page size     */
#define CONFIG_SYS_NAND_PAGE_SIZE    2048
/* NAND chip block size    */
#define CONFIG_SYS_NAND_BLOCK_SIZE   (128 * 1024)
/* NAND chip page per block count  */
#define CONFIG_SYS_NAND_PAGE_COUNT   64
```

根据 K9F2G08U0A 手册，修改成：

```
/* NAND chip page size     */
#define CONFIG_SYS_NAND_PAGE_SIZE    (2048 * 2)
/* NAND chip block size    */
#define CONFIG_SYS_NAND_BLOCK_SIZE   (128 * 4 * 1024)
/* NAND chip page per block count  */
#define CONFIG_SYS_NAND_PAGE_COUNT   (64 * 2)
```

回到 lowlevel_init.s 中，

```
#ifndef CONFIG_S3C6410
```

```
    ldr r1, [r0, #OTHERS_OFFSET]
    bic r1, r1, #0xC0
    orr r1, r1, #0x40
    str r1, [r0, #OTHERS_OFFSET]

wait_for_async:
    ldr r1, [r0, #OTHERS_OFFSET]
    and r1, r1, #0xf00
    cmp r1, #0x0
    bne wait_for_async
#endif
```

……

……

```
#elif !defined(CONFIG_S3C6400)
    /* According to 661558um_S3C6400X_rev10.pdf 0x20 is reserved */
    ldr r1, [r0, #OTHERS_OFFSET]
    bic r1, r1, #0x20
    str r1, [r0, #OTHERS_OFFSET]
```

修改成：

```
#elif !defined(CONFIG_S3C6410)
    /* According to 661558um_S3C6400X_rev10.pdf 0x20 is reserved */
    ldr r1, [r0, #OTHERS_OFFSET]
    bic r1, r1, #0x20
    str r1, [r0, #OTHERS_OFFSET]
```

系统初始化结束之后，执行

```
mov pc, lr
```

即是回到系统初始化执行的地方。即为：

```
#ifndef CONFIG_NAND_SPL
    /* for UART */
    bl uart_asm_init
#endif
```

进入 UATR 初始化。

9. UART 初始化

```
uart_asm_init:
    /* set GPIO to enable UART */
    ldr r0, =ELFIN_GPIO_BASE
    ldr r1, =0x220022
    str r1, [r0, #GPACON_OFFSET]
    mov pc, lr
```

接着回到当初跳转的地方，即是：

```
#ifdef CONFIG_BOOT_NAND
    /* simple init for NAND */
    bl nand_asm_init
#endif
```

进入 Nand 初始化。

10. Nand 初始化

```
nand_asm_init:
    ldr r0, =ELFIN_NAND_BASE
    ldr r1, [r0, #NFCONF_OFFSET]
    orr r1, r1, #0x70
    orr r1, r1, #0x7700
    str r1, [r0, #NFCONF_OFFSET]

    ldr r1, [r0, #NFCONT_OFFSET]
    orr r1, r1, #0x07
    str r1, [r0, #NFCONT_OFFSET]

    mov pc, lr
```

回到跳转之处，即为：

```
/* Memory subsystem address 0x7e00f120 */
    ldr r0, =ELFIN_MEM_SYS_CFG

    /* Xm0CSn2 = NFCON CS0, Xm0CSn3 = NFCON CS1 */
    mov r1, #S3C64XX_MEM_SYS_CFG_NAND
    str r1, [r0]

    bl  mem_ctrl_asm_init
```

进入内存管理初始化。

11. 内存管理初始化

这里跳入到 arch/arm/cpu/arm1176/s3c64xx/ mem_ctrl_asm_init.s 中。代码较长，请读者自行查看。

完成内存管理之后，程序将跳回 start.s 执行。

12. MMU 表

```
       /* 128MB for SDRAM 0xC0000000 -> 0x50000000 */
    .set __base, 0x500
    .rept 0xC80 - 0xC00
    FL_SECTION_ENTRY __base, 3, 0, 1, 1
```

```
    .set __base, __base + 1
    .endr

    /* access is not allowed. */
    .rept 0x1000 - 0xc80
    .word 0x00000000
    .endr
```

修改成：

```
    /* 128MB for SDRAM 0xC0000000 -> 0x50000000 */
    .set __base, 0x500
    .rept 0xD00 - 0xC00
    FL_SECTION_ENTRY __base, 3, 0, 1, 1
    .set __base, __base + 1
    .endr

    /* access is not allowed. */
    .rept 0x1000 - 0xD00
    .word 0x00000000
    .endr
```

回到 start 之后，设置堆栈指针

```
/* Set stackpointer in internal RAM to call board_init_f */
call_board_init_f:
    ldr sp, =(CONFIG_SYS_INIT_SP_ADDR)
    bic sp, sp, #7 /* 8-byte alignment for ABI compliance */
    ldr r0,=0x00000000
    bl  board_init_f
```

进入 uboot 在 SDRAM 的内存空间配置（在 arch/arm/lib 的 board.c 中）。

13. uboot 在 SDRAM 的内存空间配置

board_init_f 这个函数是 u-boot 执行的第一个 C 语言函数：void board_init_f(ulong bootflag)。我们先看下面这段代码。

```
    /* Pointer is writable since we allocated a register for it */
    gd = (gd_t *) ((CONFIG_SYS_INIT_SP_ADDR) & ~0x07);
    /* compiler optimization barrier needed for GCC >= 3.4 */
    __asm__ __volatile__("": : :"memory");
    memset((void *)gd, 0, sizeof(gd_t));
    gd->mon_len = _bss_end_ofs;

#ifdef CONFIG_OF_EMBED
    /* Get a pointer to the FDT */
```

```
    gd->fdt_blob = _binary_dt_dtb_start;
#elif defined CONFIG_OF_SEPARATE
    /* FDT is at end of image */
    gd->fdt_blob = (void *)(_end_ofs + _TEXT_BASE);
#endif

    /* Allow the early environment to override the fdt address */
    gd->fdt_blob = (void *)getenv_ulong("fdtcontroladdr", 16,
                    (uintptr_t)gd->fdt_blob);
```

gd_t 是一个结构体类型，其定义在 arch/arm/include/asm 目录下的 global_data.h 文件中。如下所示。

```
typedef   struct global_data {
    bd_t          *bd;
    unsigned long flags;
    unsigned long baudrate;
    unsigned long have_console; /* serial_init() was called */

#ifdef CONFIG_PRE_CONSOLE_BUFFER
    unsigned long precon_buf_idx;  /* Pre-Console buffer index */
#endif

    unsigned long env_addr; /* Address  of Environment struct */
    unsigned long env_valid;    /* Checksum of Environment valid? */
    unsigned long fb_base;  /* base address of frame buffer */

#ifdef CONFIG_FSL_ESDHC
    unsigned long sdhc_clk;
#endif

#ifdef CONFIG_AT91FAMILY
    /* "static data" needed by at91's clock.c */
    unsigned long cpu_clk_rate_hz;
    unsigned long main_clk_rate_hz;
    unsigned long mck_rate_hz;
    unsigned long plla_rate_hz;
    unsigned long pllb_rate_hz;
    unsigned long at91_pllb_usb_init;
#endif

#ifdef CONFIG_ARM
    /* "static data" needed by most of timer.c on ARM platforms */
    unsigned long timer_rate_hz;
    unsigned long tbl;
    unsigned long tbu;
```

```
    unsigned long long  timer_reset_value;
    unsigned long lastinc;
#endif

#ifdef CONFIG_IXP425
    unsigned long timestamp;
#endif

    unsigned long relocaddr;    /* Start address of U-Boot in RAM */
    phys_size_t       ram_size; /* RAM size */
    unsigned long mon_len;  /* monitor len */
    unsigned long irq_sp;       /* irq stack pointer */
    unsigned long start_addr_sp;   /* start_addr_stackpointer */
    unsigned long reloc_off;

#if !(defined(CONFIG_SYS_ICACHE_OFF) &&
defined(CONFIG_SYS_DCACHE_OFF))
    unsigned long tlb_addr;
#endif

    const void   *fdt_blob;    /* Our device tree, NULL if none */
    void      **jt;     /* jump table */
    char      env_buf[32]; /* buffer for getenv() before reloc. */

#if defined(CONFIG_POST) || defined(CONFIG_LOGBUFFER)
    unsigned long post_log_word; /* Record POST activities */
    unsigned long post_log_res; /* success of POST test */
    unsigned long post_init_f_time; /* When post_init_f started */
#endif

} gd_t;
```

我们可以看到 gd_t 结构体中定义了很多变量。而这些变量恰恰是 u-boot 中重要的全局变量。

```
gd = (gd_t *) ((CONFIG_SYS_INIT_SP_ADDR) & ~0x07);
```

gd 是一个结构体指针，指向(gd_t *) ((CONFIG_SYS_INIT_SP_ADDR) & ~0x07)。

```
memset((void *)gd, 0, sizeof(gd_t));
```

这条程序是将 gd 这个结构体中所有变量都清零了。

```
gd->mon_len = _bss_end_ofs;
```

而_bss_end_ofs 是这样定义的：

```
extern ulong _bss_start_ofs;  /* BSS start relative to _start */
extern ulong _bss_end_ofs;    /* BSS end relative to _start */
extern ulong _end_ofs;     /* end of image relative to _start */
```

在 start.s 中有这么一段代码：

```
.globl _bss_start_ofs
_bss_start_ofs:
    .word __bss_start - _start

.globl _bss_end_ofs
_bss_end_ofs:
    .word __bss_end__ - _start

.globl _end_ofs
_end_ofs:
    .word _end - _start
```

接着看 board_init_f 这个函数，

```
    for (init_fnc_ptr = init_sequence; *init_fnc_ptr; ++init_fnc_ptr)
{
        if ((*init_fnc_ptr)() != 0) {
            hang ();
        }
    }
```

首先 init_fnc_ptr 定义为：

```
init_fnc_t **init_fnc_ptr;
```

而 init_sequence 定义为：

```
typedef int (init_fnc_t) (void);
init_fnc_t *init_sequence[] = {
    arch_cpu_init,      /* basic arch cpu dependent setup */

#if defined(CONFIG_BOARD_EARLY_INIT_F)
    board_early_init_f,
#endif

#ifdef CONFIG_OF_CONTROL
    fdtdec_check_fdt,
#endif

    timer_init,      /* initialize timer */
#ifdef CONFIG_BOARD_POSTCLK_INIT
    board_postclk_init,
#endif

#ifdef CONFIG_FSL_ESDHC
    get_clocks,
#endif
```

```
    env_init,     /* initialize environment */
    init_baudrate,      /* initialze baudrate settings */
    serial_init,     /* serial communications setup */
    console_init_f,     /* stage 1 init of console */
    display_banner,     /* say that we are here */

#if defined(CONFIG_DISPLAY_CPUINFO)
    print_cpuinfo,      /* display cpu info (and speed) */
#endif

#if defined(CONFIG_DISPLAY_BOARDINFO)
    checkboard,      /* display board info */
#endif

#if defined(CONFIG_HARD_I2C) || defined(CONFIG_SOFT_I2C)
    init_func_i2c,
#endif

    dram_init,       /* configure available RAM banks */
    NULL,
};
```

*init_sequence[]这个数组中存放的是指针，而数组中存放的是各个函数的函数指针，通过调用函数指针来调用函数。这些函数是进行一些初始化的作用，在此不细究。

通过函数指针调用函数确实是一个效率较高的办法，其实 u-boot 和 linux 内核中都蕴藏着很多精巧算法。

接下来：

```
addr = CONFIG_SYS_SDRAM_BASE + gd->ram_size;
```

这条程序告诉我们 SDRAM 的末位物理地址为 0x5800 0000，即 SDRAM 的空间分布为 0x5000 0000~0x57FF FFFF。说明 SDRAM 一共 128MB 的空间。

下面的程序就是 u-boot 在这 128MB 进行内存分配。

```
#ifdef CONFIG_PRAM
    /*
     * reserve protected RAM
     */
    reg = getenv_ulong("pram", 10, CONFIG_PRAM);
    addr -= (reg << 10);       /* size is in kB */
    debug("Reserving %ldk for protected RAM at %08lx\n", reg, addr);
#endif /* CONFIG_PRAM */

#if !(defined(CONFIG_SYS_ICACHE_OFF) &&
defined(CONFIG_SYS_DCACHE_OFF))
    /* reserve TLB table */
```

```
    addr -= (4096 * 4);
    /* round down to next 64 kB limit */
    addr &= ~(0x10000 - 1);
    gd->tlb_addr = addr;
    debug("TLB table at: %08lx\n", addr);
#endif

    /* round down to next 4 kB limit */
    addr &= ~(4096 - 1);
    debug("Top of RAM usable for U-Boot at: %08lx\n", addr);
```

将SDRAM的最后64K（addr &= ~(0x10000 - 1)）分配给TLB，即为：0x57FF 0000~0x57FF FFFF。

```
#ifdef CONFIG_LCD
#ifdef CONFIG_FB_ADDR
    gd->fb_base = CONFIG_FB_ADDR;
#else
    /* reserve memory for LCD display (always full pages) */
    addr = lcd_setmem(addr);
    gd->fb_base = addr;
#endif /* CONFIG_FB_ADDR */
#endif /* CONFIG_LCD */

    /*
     * reserve memory for U-Boot code, data & bss
     * round down to next 4 kB limit
     */
    addr -= gd->mon_len;
    addr &= ~(4096 - 1);

    debug("Reserving %ldk for U-Boot at: %08lx\n", gd->mon_len >> 10,
addr);
```

这段代码是在SDRAM 中从后往前给u-boot分配BSS、数据段、代码段。地址为：0x57F7 5000~0x57FE FFFF。

```
/*
     * reserve memory for malloc() arena
     */
    addr_sp = addr - TOTAL_MALLOC_LEN;
    debug("Reserving %dk for malloc() at: %08lx\n",
          TOTAL_MALLOC_LEN >> 10, addr_sp);
```

从后往前紧挨着代码段开辟一块malloc 空间，地址为：0x57E6 D000~0x57E7 4FFF。

```
    /*
     * (permanently) allocate a Board Info struct
     * and a permanent copy of the "global" data
```

```
     */
    addr_sp -= sizeof (bd_t);
    bd = (bd_t *) addr_sp;
    gd->bd = bd;
    debug("Reserving %zu Bytes for Board Info at: %08lx\n",
          sizeof (bd_t), addr_sp);
```

随后为 bd 结构体分配空间，地址为：0x57E6 CFD8~0x57E6 CFFF。

```
    addr_sp -= sizeof (gd_t);
    id = (gd_t *) addr_sp;
    debug("Reserving %zu Bytes for Global Data at: %08lx\n",
          sizeof (gd_t), addr_sp);
```

这是给 gd 结构体分配空间，地址为：0x57E6 CF60~0x57E6 CFD7。

```
    /* setup stackpointer for exeptions */
    gd->irq_sp = addr_sp;
#ifdef CONFIG_USE_IRQ
    addr_sp -= (CONFIG_STACKSIZE_IRQ+CONFIG_STACKSIZE_FIQ);
    debug("Reserving %zu Bytes for IRQ stack at: %08lx\n",
       CONFIG_STACKSIZE_IRQ+CONFIG_STACKSIZE_FIQ, addr_sp);
#endif
    /* leave 3 words for abort-stack    */
    addr_sp -= 12;

    /* 8-byte alignment for ABI compliance */
    addr_sp &= ~0x07;
#else
    addr_sp += 128;  /* leave 32 words for abort-stack   */
    gd->irq_sp = addr_sp;
#endif

    debug("New Stack Pointer is: %08lx\n", addr_sp);
```

分配异常中断空间，地址：0x57E6 CF50~0x57E6 CF5F。

综合上面 SDRAM 的分配，那么内存图如图 3. 2 所示。SDRAM 的空间大小为 128MB。

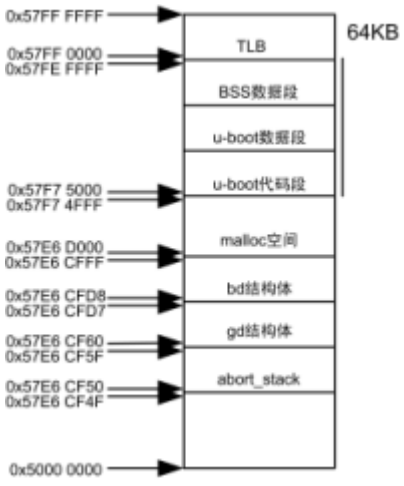


图 3. 2 SDRAM 分配图

其中在 smdk6410.h 中，有这么一个宏定义：

```
#define CONFIG_SYS_SDRAM_BASE 0x50000000
```

这说明 SDRAM 的起始地址是：0x5000 0000。

接着是 gd 结构体的初始化，然后

```
relocate_code(addr_sp, id, addr);
```

这条程序告诉我们应该回到 start.s 中，在 start.s 中有这么段代码。

```
    .globl relocate_code
relocate_code:
    mov r4, r0 /* save addr_sp */
    mov r5, r1 /* save addr of gd */
    mov r6, r2 /* save addr of destination */
```

这里将 relocate_code()带回来的三个参数分别装入 r4、r5、r6 中。

但是注意到，relocate_code 这个函数的声明实在 commom.h 中，如下：

```
void   relocate_code (ulong, gd_t *, ulong) __attribute__
((noreturn));
```

relocate_code 函数的三个参数分别栈顶地址、数据 ID（即全局结构 gd）在 SDRAM 中的起始地址和在 SDRAM 中存储 U-Boot 的起始地址。

接着又是回到 start.s 中。

14. 设置堆栈指针

```
/* Set up the stack                        */
stack_setup:
    mov sp, r4

    adr r0, _start
    cmp r0, r6
    moveq  r9, #0     /* no relocation. relocation offset(r9) = 0 */
    beq clear_bss     /* skip relocation */
    mov r1, r6        /* r1 <- scratch for copy_loop */
    ldr r3, _bss_start_ofs
    add r2, r0, r3       /* r2 <- source end address      */

copy_loop:
    ldmia  r0!, {r9-r10}    /* copy from source address [r0]    */
    stmia  r1!, {r9-r10}    /* copy to   target address [r1]    */
    cmp r0, r2        /* until source end address [r2]    */
    blo copy_loop
```

r4 是刚刚传回来的堆栈指针，那么将 r4 给 sp。设定堆栈指针。

r6 是在 SDRAM 中存储 u-boot 的起始地址，将 r0 和 r6 进行比较。如果此时的 u-boot 已经是在 SDRAM 中，则 beq　clear_bss；如果不是，在 NandFlash 中，则要将 u-boot 复制到 SDRAM 中。

```
clear_bss:
#ifndef CONFIG_SPL_BUILD
    ldr r0, _bss_start_ofs
    ldr r1, _bss_end_ofs
    mov r4, r6        /* reloc addr */
    add r0, r0, r4
    add r1, r1, r4
    mov r2, #0x00000000     /* clear            */

clbss_l:cmp   r0, r1        /* clear loop... */
    bhs clbss_e          /* if reached end of bss, exit */
    str r2, [r0]
    add r0, r0, #4
    b   clbss_l
clbss_e:
#ifndef CONFIG_NAND_SPL
    bl coloured_LED_init
    bl red_led_on
#endif
#endif
```

上面这段代码是进行 BSS 清零。

15. 代码启动

```
/*
 * We are done. Do not return, instead branch to second part of board
 * initialization, now running from RAM.
 */
#ifdef CONFIG_NAND_SPL
    ldr     pc, _nand_boot

_nand_boot: .word nand_boot
#else
    ldr r0, _board_init_r_ofs
    adr r1, _start
    add lr, r0, r1
    add     lr, lr, r9
    /* setup parameters for board_init_r */
    mov r0, r5     /* gd_t */
    mov r1, r6     /* dest_addr */
    /* jump to it ... */
    mov pc, lr

_board_init_r_ofs:
    .word board_init_r - _start
```

```
#endif
```

在这段代码之前有一段代码（对 rel.dyn 进行重定向）不详细列出。

```
#ifdef CONFIG_NAND_SPL
    ldr     pc, _nand_boot

_nand_boot: .word nand_boot
```

这段代码如果是 NAND 启动的话，那么就设置 SP 后跳到 nand_boot 函数里面进行复制代码到 SDRAM，然后跳到 uboot 在 SDRAM 的起始地址开始运行。但是由于 CONFIG_NAND_SPL 没有宏定义，则是执行 else。在进入 board_init_r 之前，给了两个参数: r5、r6。

r5：数据 ID（即全局结构 gd）在 SDRAM 中的起始地址。

r6：在 SDRAM 中存储 u-boot 的起始地址。

16. 重定位，进入 board_init_r 函数

进入 board_init_r 函数：

```
void board_init_r(gd_t *id, ulong dest_addr)
```

```
gd = id;
```

gd 获取到起始地址。

```
gd->flags |= GD_FLG_RELOC; /* tell others: relocation done */
```

给标志值赋值，说明代码应该重定位到了 SDRAM 中。

```
monitor_flash_len = _end_ofs;
```

u-boot 的长度。

然后执行很多初始化函数：

```
/* Enable caches */
    enable_caches();

    debug("monitor flash len: %08lX\n", monitor_flash_len);
    board_init(); /* Setup chipselects */
    /*
     * TODO: printing of the clock inforamtion of the board is now
     * implemented as part of bdinfo command. Currently only support
for
     * davinci SOC's is added. Remove this check once all the board
     * implement this.
     */
#ifdef CONFIG_CLOCKS
```

```
    set_cpu_clk_info(); /* Setup clock information */
#endif

#ifdef CONFIG_SERIAL_MULTI
    serial_initialize();
#endif

    debug("Now running in RAM - U-Boot at: %08lx\n", dest_addr);

#ifdef CONFIG_LOGBUFFER
    logbuff_init_ptrs();
#endif

#ifdef CONFIG_POST
    post_output_backlog();
#endif
```

```
    /* The Malloc area is immediately below the monitor copy in DRAM
*/
    malloc_start = dest_addr - TOTAL_MALLOC_LEN;
    mem_malloc_init (malloc_start, TOTAL_MALLOC_LEN);
```

这里是将整个 malloc 空间清零。

```
#if !defined(CONFIG_SYS_NO_FLASH)
    puts("Flash: ");

    flash_size = flash_init();
    if (flash_size > 0) {
#ifdef CONFIG_SYS_FLASH_CHECKSUM
        char *s = getenv("flashchecksum");

        print_size(flash_size, "");
        /*
         * Compute and print flash CRC if flashchecksum is set to 'y'
         *
         * NOTE: Maybe we should add some WATCHDOG_RESET()? XXX
         */
        if (s && (*s == 'y')) {
            printf("  CRC: %08X", crc32(0,
                (const unsigned char *) CONFIG_SYS_FLASH_BASE,
                flash_size));
        }
        putc('\n');
```

```
#else  /* !CONFIG_SYS_FLASH_CHECKSUM */
       print_size(flash_size, "\n");
#endif /* CONFIG_SYS_FLASH_CHECKSUM */
   } else {
       puts(failed);
       hang();
   }
#endif
```

计算 s3c 中 Flash 的大小。

```
#if defined(CONFIG_CMD_NAND)
   puts("NAND:  ");
   nand_init();     /* go init the NAND */
#endif
```

初始化 NandFlash。

```
/* nand_init() - initialize data to make nand usable by SPL */
void nand_init(void)
{
   /*
    * Init board specific nand support
    */
   mtd.priv = &nand_chip;
   nand_chip.IO_ADDR_R = nand_chip.IO_ADDR_W =
       (void  __iomem *)CONFIG_SYS_NAND_BASE;
   board_nand_init(&nand_chip);

#ifdef CONFIG_SPL_NAND_SOFTECC
   if (nand_chip.ecc.mode == NAND_ECC_SOFT) {
       nand_chip.ecc.calculate = nand_calculate_ecc;
       nand_chip.ecc.correct = nand_correct_data;
   }
#endif

   if (nand_chip.select_chip)
       nand_chip.select_chip(&mtd, 0);
}
```

这个函数是在 drivesr/mtd/nand/nand.c 文件中。

```
   /* initialize environment */
   env_relocate();
```

这个是环境变量初始化。

```
    stdio_init(); /* get the devices list going. */
```

这是外设初始化函数。有 I2C、LCD、VIDEO、KEYBOARD、USB、JTAG 等。

```
    jumptable_init();
```

跳转表初始化函数。其原型是：

```
void jumptable_init(void)
{
    gd->jt = malloc(XF_MAX * sizeof(void *));
#include <_exports.h>
}
```

```
#if defined(CONFIG_API)
    /* Initialize API */
    api_init();
#endif

    console_init_r();   /* fully init console as a device */
```

控制台初始化。

```
#if defined(CONFIG_ARCH_MISC_INIT)
    /* miscellaneous arch dependent initialisations */
    arch_misc_init();
#endif
#if defined(CONFIG_MISC_INIT_R)
    /* miscellaneous platform dependent initialisations */
    misc_init_r();
#endif

     /* set up exceptions */
    interrupt_init();
    /* enable exceptions */
    enable_interrupts();
```

中断初始化和中断使能。

```
    /* Perform network card initialisation if necessary */
#if defined(CONFIG_DRIVER_SMC91111) || defined
(CONFIG_DRIVER_LAN91C96)
    /* XXX: this needs to be moved to board init */
    if (getenv("ethaddr")) {
        uchar enetaddr[6];
        eth_getenv_enetaddr("ethaddr", enetaddr);
```

```
        smc_set_mac_addr(enetaddr);
    }
#endif /* CONFIG_DRIVER_SMC91111 || CONFIG_DRIVER_LAN91C96 */

    /* Initialize from environment */
    load_addr = getenv_ulong("loadaddr", 16, load_addr);
```

从环境变量中获取 loadaddr 参数，得到需要加载的地址。

函数原型：

```
ulong getenv_ulong(const char *name, int base, ulong default_val)
```

```
#ifdef CONFIG_BOARD_LATE_INIT
    board_late_init();
#endif

#ifdef CONFIG_BITBANGMII
    bb_miiphy_init();
#endif
#if defined(CONFIG_CMD_NET)
    puts("Net:   ");
    eth_initialize(gd->bd);
#if defined(CONFIG_RESET_PHY_R)
    debug("Reset Ethernet PHY\n");
    reset_phy();
#endif
#endif

#ifdef CONFIG_POST
    post_run(NULL, POST_RAM | post_bootmode_get(0));
#endif

#if defined(CONFIG_PRAM) || defined(CONFIG_LOGBUFFER)
    /*
     * Export available size of memory for Linux,
     * taking into account the protected RAM at top of memory
     */
    {
        ulong pram = 0;
        uchar memsz[32];

#ifdef CONFIG_PRAM
        pram = getenv_ulong("pram", 10, CONFIG_PRAM);
#endif
```

```
#ifdef CONFIG_LOGBUFFER
#ifndef CONFIG_ALT_LB_ADDR
       /* Also take the logbuffer into account (pram is in kB) */
       pram += (LOGBUFF_LEN + LOGBUFF_OVERHEAD) / 1024;
#endif
#endif

       sprintf((char *)memsz, "%ldk", (gd->ram_size / 1024) - pram);
       setenv("mem", (char *)memsz);
   }
#endif
```

进行完一大堆的初始化函数之后，程序进入一个 for 循环。

17. main_loop()函数

```
/*main_loop()can return to retry autoboot,ifsojust run it again. */
   for (;;) {
      main_loop();
   }
```

程序至此死在了 main_loop()中，那么追根究底，看看 main_loop()函数是什么？

```
void main_loop (void)
```

main_loop 是无入口参数也无返回值的函数。

```
#ifndef CONFIG_SYS_HUSH_PARSER
   static char lastcommand[CONFIG_SYS_CBSIZE] = { 0, };
   int len;
   int rc = 1;
   int flag;
#endif
```

HUSH 的相关初始化。

```
#if defined(CONFIG_BOOTDELAY) && (CONFIG_BOOTDELAY >= 0)
   char *s;
   int bootdelay;
#endif
```

bootdelay 的一些初始化。

```
#ifdef CONFIG_PREBOOT
   char *p;
#endif

#ifdef CONFIG_BOOTCOUNT_LIMIT
```

```
    unsigned long bootcount = 0;
    unsigned long bootlimit = 0;
    char *bcs;
    char bcs_set[16];
#endif /* CONFIG_BOOTCOUNT_LIMIT */
```

上面都是一些变量的定义。

```
#ifdef CONFIG_BOOTCOUNT_LIMIT
    bootcount = bootcount_load();
```

上面这行代码加载保存的启动次数。

```
    bootcount++;
```

启动次数加 1。

```
    bootcount_store (bootcount);
```

更新启动次数。

```
    sprintf (bcs_set, "%lu", bootcount);
```

将启动次数通过串口输出。

```
    setenv ("bootcount", bcs_set);
    bcs = getenv ("bootlimit");
    bootlimit = bcs ? simple_strtoul (bcs, NULL, 10) : 0;
#endif /* CONFIG_BOOTCOUNT_LIMIT */
```

这段代码蕴含的东西较多。启动次数限制功能，启动次数限制可以被用户设置一个启动次数，然后保存在 Flash 存储器的特定位置，当到达启动次数后，u-boot 无法启动。该功能适合一些商业产品，通过配置不同的 License 限制用户重新启动系统。

```
#ifdef CONFIG_MODEM_SUPPORT
    debug ("DEBUG: main_loop:   do_mdm_init=%d\n", do_mdm_init);
    if (do_mdm_init) {
        char *str = strdup(getenv("mdm_cmd"));
        setenv ("preboot", str);  /* set or delete definition */
        if (str != NULL)
            free (str);
        mdm_init(); /* wait for modem connection */
    }
#endif  /* CONFIG_MODEM_SUPPORT */
```

如果系统中有 Modem 功能，打开其功能可以接受其他用户通过电话网络的拨号请求。Modem 功能通常供一些远程控制的系统使用

```
#ifdef CONFIG_VERSION_VARIABLE
```

```
    {
        setenv ("ver", version_string);  /* set version variable */
    }
#endif /* CONFIG_VERSION_VARIABLE */
```

设置 u-boot 的版本号。打开动态版本支持功能后，u-boot 在启动的时候会显示最新的版本号。

```
#ifdef CONFIG_SYS_HUSH_PARSER
    u_boot_hush_start ();
#endif

#if defined(CONFIG_HUSH_INIT_VAR)
    hush_init_var ();
#endif
```

初始化 HUSH 功能。

```
#ifdef CONFIG_PREBOOT
    if ((p = getenv ("preboot")) != NULL) {
# ifdef CONFIG_AUTOBOOT_KEYED
        int prev = disable_ctrlc(1);  /* disable Control C checking */
# endif
```

关闭 Ctrl+C 组合键。

```
        run_command_list(p, -1, 0);
```

运行 u-boot 参数。

```
# ifdef CONFIG_AUTOBOOT_KEYED
        disable_ctrlc(prev);    /* restore Control C checking */
# endif
    }
#endif /* CONFIG_PREBOOT */
```

回复 Ctrl+C 组合键功能。

```
#if defined(CONFIG_UPDATE_TFTP)
    update_tftp (0UL);
#endif /* CONFIG_UPDATE_TFTP */
```

启动 tftp 功能。

```
#if defined(CONFIG_BOOTDELAY) && (CONFIG_BOOTDELAY >= 0)
    s = getenv ("bootdelay");
    bootdelay = s ? (int)simple_strtol(s, NULL, 10) : CONFIG_BOOTDELAY;
```

```
    debug ("### main_loop entered: bootdelay=%d\n\n", bootdelay);
```

在进入主循环之前，如果配置了启动延迟功能，需要等待用户从串口或者网络接口输入。如果用户按下任意键打断，启动流程，会向终端打印出一个启动菜单。

```
#if defined(CONFIG_MENU_SHOW)
    bootdelay = menu_show(bootdelay);
#endif
```

向终端打印出一个启动菜单。

```
# ifdef CONFIG_BOOT_RETRY_TIME
    init_cmd_timeout ();
# endif   /* CONFIG_BOOT_RETRY_TIME */
```

初始化命令行超时机制。

```
#ifdef CONFIG_POST
    if (gd->flags & GD_FLG_POSTFAIL) {
        s = getenv("failbootcmd");
    }
    else
#endif /* CONFIG_POST */
#ifdef CONFIG_BOOTCOUNT_LIMIT
    if (bootlimit && (bootcount > bootlimit)) {
        printf ("Warning: Bootlimit (%u) exceeded. Using
altbootcmd.\n",
                (unsigned)bootlimit);
```

检测是否超出启动次数限制。

```
        s = getenv ("altbootcmd");
    }
    else
#endif /* CONFIG_BOOTCOUNT_LIMIT */
        s = getenv ("bootcmd");
```

获取启动命令参数。

```
    debug ("### main_loop: bootcmd=\"%s\"\n", s ? s : "<UNDEFINED>");

    if (bootdelay != -1 && s && !abortboot(bootdelay)) {
```

检测是否支持延迟启动功能。

```
# ifdef CONFIG_AUTOBOOT_KEYED
        int prev = disable_ctrlc(1);  /* disable Control C checking */
# endif

        run_command_list(s, -1, 0);
```

运行启动命令行。

```
# ifdef CONFIG_AUTOBOOT_KEYED
       disable_ctrlc(prev);    /* restore Control C checking */
# endif
```

关闭 Ctrl+C 组合键功能。

```
   }

# ifdef CONFIG_MENUKEY
   if (menukey == CONFIG_MENUKEY) {
       s = getenv("menucmd");
       if (s)
           run_command_list(s, -1, 0);
   }
#endif /* CONFIG_MENUKEY */
#endif /* CONFIG_BOOTDELAY */
```

运行启动命令行。

```
   /*
    * Main Loop for Monitor Command Processing
    */
#ifdef CONFIG_SYS_HUSH_PARSER
   parse_file_outer();
   /* This point is never reached */
   for (;;);
```

由于 CONFIG_SYS_HUSH_PARSER 没有宏定义，上面这个死循环不会执行，而是执行下面这个死循环。

```
#else
   for (;;) {

#ifdef CONFIG_BOOT_RETRY_TIME
       if (rc >= 0) {
           /* Saw enough of a valid command to
            * restart the timeout.
            */
           reset_cmd_timeout();
       }
#endif

       len = readline (CONFIG_SYS_PROMPT);

       flag = 0; /* assume no special flags for now */
```

```
       if (len > 0)
          strcpy (lastcommand, console_buffer);
```

保存输入的数据。

```
       else if (len == 0)
          flag |= CMD_FLAG_REPEAT;
```

如果输入数据为零，则重复执行上次的命令，如果上次输入的是一个命令的话。

```
#ifdef CONFIG_BOOT_RETRY_TIME
       else if (len == -2) {
          /* -2 means timed out, retry autoboot
           */
          puts ("\nTimed out waiting for command\n");
# ifdef CONFIG_RESET_TO_RETRY
          /* Reinit board to run initialization code again */
          do_reset (NULL, 0, 0, NULL);
# else
          return;       /* retry autoboot */
# endif
       }
#endif

       if (len == -1)
          puts ("<INTERRUPT>\n");
       else
          rc = run_command(lastcommand, flag);
```

执行命令。

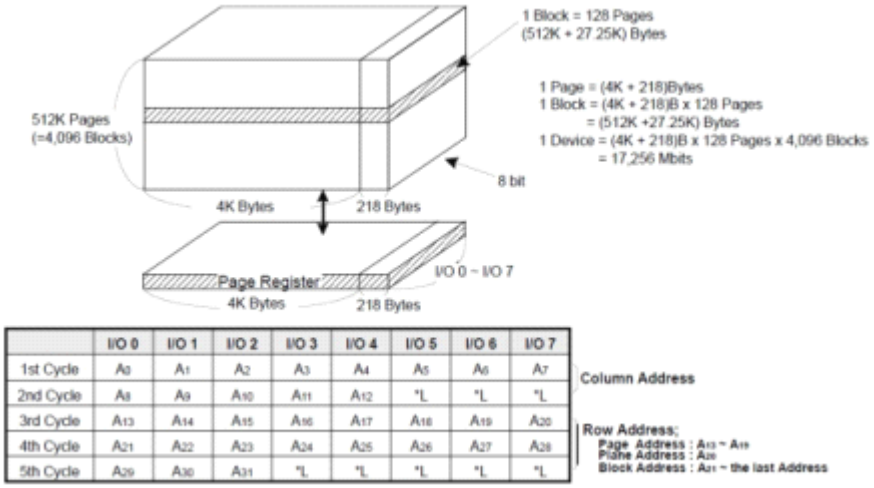
```
       if (rc <= 0) {
          /* invalid command or not repeatable, forget it */
          lastcommand[0] = 0;
       }
   }
#endif /*CONFIG_SYS_HUSH_PARSER*/
```

执行失败，则清空记录。

18.

第 4 章 u-boot-2012.10 移植之 NandFlash

在移植操作之前，很有必要将 NandFlash 的结构看一下。OK6410 开发板上的 NandFlash 芯片的型号为：K9GAG08U0D。如图 4.1 所示。



	I/O 0	I/O 1	I/O 2	I/O 3	I/O 4	I/O 5	I/O 6	I/O 7
1st Cycle	A0	A1	A2	A3	A4	A5	A6	A7
2nd Cycle	A8	A9	A10	A11	A12	*L	*L	*L
3rd Cycle	A13	A14	A15	A16	A17	A18	A19	A20
4th Cycle	A21	A22	A23	A24	A25	A26	A27	A28
5th Cycle	A29	A30	A31	*L	*L	*L	*L	*L

图 4.1 K9GAG08U0D

开发平台：OK6410 256MBSDRAM 2GNandFlash

由于 NandFlash 的移植和开发平台是有关系的。

4.1 NandFlash 启动

在之前的修改中，我们已经在 smdk6410.h 中的 NandFlash 修改成了如下：

```
/* Offset to RAM U-Boot image */
#define CONFIG_SYS_NAND_U_BOOT_OFFS  (4 * 1024)
/* Size of RAM U-Boot image   */
#define CONFIG_SYS_NAND_U_BOOT_SIZE  (252 * 1024)
```

修改成：

```
/* Offset to RAM U-Boot image */
#define CONFIG_SYS_NAND_U_BOOT_OFFS  (16 * 1024)
/* Size of RAM U-Boot image   */
#define CONFIG_SYS_NAND_U_BOOT_SIZE  (496 * 1024)
```

……

从图 4.1 中可以知道，一页的大小为 4K，即为 4096Bytes；一块共有 128 页，那么一块的大小就为： $128 \times 4096\text{Bytes}$ 。

那么就有了如下修改：

```
/* NAND chip page size     */
#define CONFIG_SYS_NAND_PAGE_SIZE    2048
```

```
/* NAND chip block size    */
#define CONFIG_SYS_NAND_BLOCK_SIZE   (128 * 1024)
/* NAND chip page per block count  */
#define CONFIG_SYS_NAND_PAGE_COUNT   64
```

根据 K9GAG08U0D 手册，修改成：

```
/* NAND chip page size     */
#define CONFIG_SYS_NAND_PAGE_SIZE    (4 * 1024)
/* NAND chip block size    */
#define CONFIG_SYS_NAND_BLOCK_SIZE   (128 * 4 * 1024)
/* NAND chip page per block count  */
#define CONFIG_SYS_NAND_PAGE_COUNT   (64 * 2)
```

进入 smdk6410_nand_spl.c 文件，里面只有一个函数：

```
void board_init_f(unsigned long bootflag)
{
    relocate_code(CONFIG_SYS_TEXT_BASE - TOTAL_MALLOC_LEN, NULL,
          CONFIG_SYS_TEXT_BASE);
}
```

relocate_code()这个函数已经很熟悉了，三个参数分别栈顶地址、数据 ID（即全局结构 gd）在 SDRAM 中的起始地址和在 SDRAM 中存储 u-boot 的起始地址。我们现在很有必要看一下 u-boot 存储器映射，如图 4.2 所示。

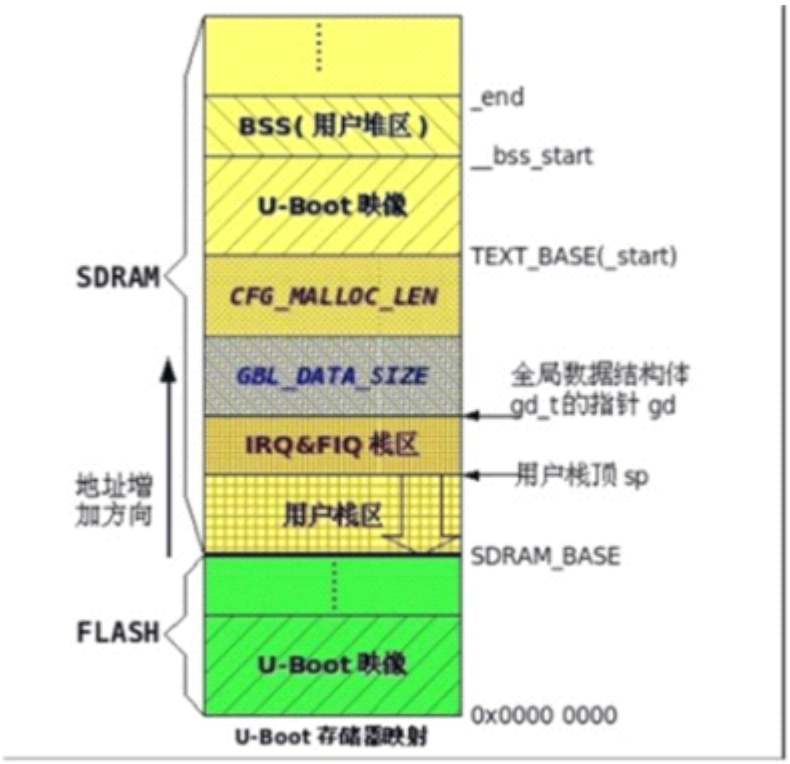


图 4.2 u-boot 存储器映射

对于 CONFIG_SYS_TEXT_BASE 的值，也就是所谓的运行地址，在 board/samsung/smdk6410 下的 config.mk 有了描述：

```
ifndef CONFIG_NAND_SPL
CONFIG_SYS_TEXT_BASE = $(RAM_TEXT)
else
CONFIG_SYS_TEXT_BASE = 0
endif
```

如果没有宏定义 CONFIG_NAND_SPL 则 CONFIG_SYS_TEXT_BASE = $(RAM_TEXT)，否则为 0。而在 Makefile 中定义了 RAM_TEXT=0x57e0 0000。通过 smdk6410.h 可以知道 CONFIG_NAND_SPL 是没有宏定义的，那么也就是说 CONFIG_SYS_TEXT_BASE = 0x57e0 0000。

```
void board_init_f(unsigned long bootflag)
```

这个函数我们要修改为：

```
void board_init_f(unsigned long bootflag)
{
    relocate_code(8*1024, NULL,CONFIG_SYS_TEXT_BASE);
}
```

接着进入 nand_spl/board \samsung/smdk6410/config.mk 这个文件，将

```
PAD_TO := $(shell expr $(CONFIG_SYS_TEXT_BASE) + 4096)
```

修改成：

```
PAD_TO := $(shell expr $(CONFIG_SYS_TEXT_BASE) + 8192)
```

在这里为了，进入 timer.c 中，

```
46  static ulong timer_load_val;
```

修改成

```
46  DECLARE_GLOBAL_DATA_PTR;
```

……

```
/* Internal tick units */
/* Last decremneter snapshot */
static unsigned long lastdec;
/* Monotonic incrementing timer */
static unsigned long long timestamp;
```

把这个定义删除。

……

```
    timers->TCFG0 = PRESCALER << 8;
    if (timer_load_val == 0) {
        /* 100s */
        timer_load_val = get_PCLK() / PRESCALER * (100 / 4);
```

```
        timers->TCFG1 = (timers->TCFG1 & ~0xf0000) | 0x20000;
    }
```

修改成：

```
    timers->TCFG0 = PRESCALER << 8;
    /* 100s */
    gd->timer_rate_hz = get_PCLK() / PRESCALER * (100 / 4);
    timers->TCFG1 = (timers->TCFG1 & ~0xf0000) | 0x20000;
```

……

```
    /* load value for 10 ms timeout */
    lastdec = timers->TCNTB4 = timer_load_val;
```

修改成：

```
    /* load value for 10 ms timeout */
    gd->lastinc = timers->TCNTB4 = gd->timer_rate_hz;
```

……

```
99  timestamp = 0;
```

修改成：

```
99  gd->timer_reset_value = 0;
```

……

```
unsigned long long get_ticks(void)
{
    ulong now = read_timer();

    if (lastdec >= now) {
        /* normal mode */
        timestamp += lastdec - now;
    } else {
        /* we have an overflow ... */
        timestamp += lastdec + timer_load_val - now;
    }
    lastdec = now;

    return timestamp;
}
```

修改成：

```
unsigned long long get_ticks(void)
{
    ulong now = read_timer();

    if (gd->lastinc >= now) {
        /* normal mode */
        gd->timer_reset_value += gd->lastinc - now;
    } else {
```

```
        /* we have an overflow ... */
        gd->timer_reset_value += gd->lastinc + gd->timer_rate_hz -
now;
    }
    gd->lastinc = now;

    return gd->timer_reset_value;
}
```

……

```
ulong get_tbclk(void)
{
    /* We overrun in 100s */
    return (ulong)(timer_load_val / 100);
}

ulong get_timer_masked(void)
{
    unsigned long long res = get_ticks();
    do_div (res, (timer_load_val / (100 * CONFIG_SYS_HZ)));
    return res;
}
```

修改成：

```
ulong get_tbclk(void)
{
    /* We overrun in 100s */
    return (ulong)(gd->timer_rate_hz / 100);
}

ulong get_timer_masked(void)
{
    unsigned long long res = get_ticks();
    //do_div(res, (gd->timer_rate_hz / (100 * CONFIG_SYS_HZ)));
    return res;
}
```

修改完成这个四个文件之后，编译，将 u-boot-nand.bin 文件通过烧写入 OK6410 中，启动，通过 dnw，可以看到如图 4. 3 所示。

```
U-Boot 2012.10 (Dec 07 2012 - 11:31:03) for OK6410


CPU:     S3C6410@533MHz
         Fclk = 533MHz, Hclk = 133MHz, Pclk = 66MHz (ASYNC Mode)
Board:   SMDK6410
DRAM:  256 MiB
WARNING: Caches not enabled
Flash: *** failed ***
### ERROR ### Please RESET the board ###
```

图 4. 3　NandFlash 启动

通过 dnw 的输出，我们可以看出还是存在问题，Caches 没有使能、Flash 失败，并且要求重启 board。

很明显这些都是在 board 的初始化过程，那么进入 board.c(arch/arm/lib)这个文件。在 void board_init_r(gd_t *id, ulong dest_addr)这个函数中，有这么一行程序：

```
   /* Enable caches */
   enable_caches();
```

那么就再进入 enable_caches()这个函数。这个函数位于 cache.c(arch/arm/lib)这个文件中。这个函数有调用了 void __enable_caches(void)，对这个函数做修改：

```
   icache_enable();
   if(!icache_status()){
      puts("WARNING: iCaches not enabled\n");
   }
   dcache_enable();
   if(!icache_status()){
      puts("WARNING: dCaches not enabled\n");
   }
```

在 void board_init_r(gd_t *id, ulong dest_addr)中，有：

```
#if !defined(CONFIG_SYS_NO_FLASH)
   puts("Flash: ");

   flash_size = flash_init();
   if (flash_size > 0) {
#ifdef CONFIG_SYS_FLASH_CHECKSUM
      char *s = getenv("flashchecksum");
```

```
        print_size(flash_size, "");
        /*
         * Compute and print flash CRC if flashchecksum is set to 'y'
         *
         * NOTE: Maybe we should add some WATCHDOG_RESET()? XXX
         */
        if (s && (*s == 'y')) {
            printf("  CRC: %08X", crc32(0,
                (const unsigned char *) CONFIG_SYS_FLASH_BASE,
                flash_size));
        }
        putc('\n');
#else  /* !CONFIG_SYS_FLASH_CHECKSUM */
        print_size(flash_size, "\n");
#endif /* CONFIG_SYS_FLASH_CHECKSUM */
    } else {
        puts(failed);
        hang();
    }
#endif
```

但是 OK6410 是没有 Flash 的，所以:

```
flash_size = flash_init();
    if (flash_size > 0)
```

上面的 if 不会执行，那就只有执行

```
else {
        puts(failed);
        hang();
    }
```

但是 hang()是一个死循环：

```
void hang(void)
{
    puts("### ERROR ### Please RESET the board ###\n");
    for (;;);
}
```

这就是图 4. 3 为什么会出现的结果。

面对这个问题，有两种解决办法：其一

```
#if !defined(CONFIG_SYS_NO_FLASH)
......
#endif
```

让这段代码都别执行。

其二，就算执行这段代码，不能进入 hang()，那就是屏蔽掉 hang()，效果如图 4.4 所示。

```
U-Boot 2012.10 (Dec 07 2012 - 16:40:54) for OK6410

Author :  zhuzhaoqi
E-mail : jxlgzzq@163.com

CPU:     S3C6410@533MHz
         Fclk = 533MHz, Hclk = 133MHz, Pclk = 66MHz (ASYNC Mode)
Board:   OK6410
DRAM:  256 MiB
Flash: *** failed ***
NAND:  No oob scheme defined for oobsize 218
2048 MiB
*** Warning - bad CRC, using default environment
```

图 4.4　屏蔽 hang()

但是从归根结底的办法，选择前者。

```
#if !defined(CONFIG_SYS_NO_FLASH)
```

这行代码的意思是：如果没有定义 CONFIG_SYS_NO_FLASH 则会执行 if 里面的代码，那么要使得不执行，那就在 smdk6410.h 中宏定义 CONFIG_SYS_NO_FLASH：

```
#define    CONFIG_SYS_NO_FLASH
```

这样编译会导致出错，因为在 drivers/mnt/sfi_flash.c 文件的：

```
/* FLASH chips info */
flash_info_t flash_info[CFI_MAX_FLASH_BANKS];
```

flash_info_t 的定义在 flash.h 中：

```
#ifndef CONFIG_SYS_NO_FLASH
/*-----------------------------------------------------------------------
--------
 * FLASH Info: contains chip specific data, per FLASH bank
 */

typedef struct {
   ulong  size;         /* total bank size in bytes       */
   ushort sector_count;    /* number of erase units       */
   ulong  flash_id;     /* combined device & manufacturer code  */
   ulong  start[CONFIG_SYS_MAX_FLASH_SECT];   /* virtual sector
start address */
   uchar  protect[CONFIG_SYS_MAX_FLASH_SECT]; /* sector protection
status */
#ifdef CONFIG_SYS_FLASH_CFI
```

```
    uchar  portwidth;       /* the width of the port      */
    uchar  chipwidth;       /* the width of the chip      */
    ushort buffer_size;     /* # of bytes in write buffer    */
    ulong  erase_blk_tout;  /* maximum block erase timeout       */
    ulong  write_tout;      /* maximum write timeout      */
    ulong  buffer_write_tout;/* maximum buffer write timeout    */
    ushort vendor;          /* the primary vendor id      */
    ushort cmd_reset;       /* vendor specific reset command */
    ushort interface;       /* used for x8/x16 adjustments       */
    ushort legacy_unlock;   /* support Intel legacy (un)locking  */
    ushort manufacturer_id; /* manufacturer id         */
    ushort device_id;       /* device id           */
    ushort device_id2;      /* extended device id         */
    ushort ext_addr;     /* extended query table address      */
    ushort cfi_version;     /* cfi version             */
    ushort cfi_offset;      /* offset for cfi query       */
    ulong   addr_unlock1;   /* unlock address 1 for AMD flash roms  */
    ulong   addr_unlock2;   /* unlock address 2 for AMD flash roms  */
    const char *name;       /* human-readable name               */
#endif
} flash_info_t;
```

那么我们现在对

```
#ifndef CONFIG_SYS_NO_FLASH
```

修改成：

```
#ifdef CONFIG_SYS_NO_FLASH
```

这样就没有问题。

编译：

```
make smdk6410_config
……
make all
```

将生成的 u-boot-nand.bin 烧写进开发板。可以看到如图 4. 5 所示信息。

```
U-Boot 2012.10 (Dec 08 2012 - 19:43:30) for OK6410

Author :  zhuzhaoqi
E-mail : jxlgzzq@163.com

CPU:     S3C6410@533MHz
         Fclk = 533MHz, Hclk = 133MHz, Pclk = 66MHz (ASYNC Mode)
Board:   OK6410
DRAM:  256 MiB
NAND:  No oob scheme defined for oobsize 218
2048 MiB
*** Warning - bad CRC, using default environment
```

图 4. 5　去除 flash 编译

从输出的信息，依旧还是存在问题。

```
NAND:  No oob scheme defined for oobsize 218
2048 MiB
```

进入\drivers\mtd\nand\nand_base.c，添加：

```
static struct nand_ecclayout nand_oob_218_128Bit = {
    .eccbytes = 104,
    .eccpos = {
        24,25,26,27,28,29,30,31,32,33,
        34,35,36,37,38,39,40,41,42,43,
        44,45,46,47,48,49,50,51,52,53,
        54,55,56,57,58,59,60,61,62,63,
        64,65,66,67,68,69,70,71,72,73,
        74,75,76,77,78,79,80,81,82,83,
        84,85,86,87,88,89,90,91,92,93,
        94,95,96,97,98,99,100,101,102,103,
        104,105,106,107,108,109,110,111,112,113,
        114,115,116,117,118,119,120,121,122,123,
        124,125,126,127},
    .oobfree =
```

```
    {
        {
            .offset = 2,
            .length = 22
        }
    }
};
```

在 int nand_scan_tail(struct mtd_info *mtd)函数中添加：

```
        case 218:
            chip->ecc.layout = &nand_oob_218_128Bit;
            break;
```

修改之后如图 4.6 所示。

```
U-Boot 2012.10 (Dec 08 2012 - 20:46:56) for OK6410

Author :  zhuzhaoqi
E-mail : jxlgzzq@163.com

CPU:     S3C6410@533MHz
         Fclk = 533MHz, Hclk = 133MHz, Pclk = 66MHz (ASYNC Mode)
Board:   OK6410
DRAM:  256 MiB
NAND:  2048 MiB
*** Warning - bad CRC, using default environment

In:    serial
Out:   serial
Err:   serial
Net:   CS8900-0
```

图 4.6　nand 修改最终版本

4.2　8 位 ECC 校验

ECC 的全称是 Error Checking and Correction，是一种用于 Nand 的差错检测和修正算法。如果操作时序和电路稳定性不存在问题的话，NAND Flash 出错的时候一般不会造成整个 Block 或是 Page 不能读取或是全部出错，而是整个 Page（例如 512Bytes）中只有一个或几个 bit 出错。ECC 能纠正 1 个比特错误和检测 2 个比特错误，而且计算速度很快，但对 1 比特以上的错误无法纠正，对 2 比特以上的错误不保证能检测。

在 s3c64xx.c 文件中添加：

```
/* Nand flash definition values by jsgood */
#define S3C_NAND_TYPE_UNKNOWN 0x0
#define S3C_NAND_TYPE_SLC     0x1
#define S3C_NAND_TYPE_MLC     0x2

/* Nand flash global values by jsgood */
int cur_ecc_mode = 0;
int nand_type = S3C_NAND_TYPE_UNKNOWN;
```

下面的#ifdef/#endif 是用来调试使用，亦可不加。

```
/* Nand flash definition values by jsgood */
#ifdef S3C_NAND_DEBUG
......
#endif
```

上面的#endif 终止于：int board_nand_init(struct nand_chip *nand)函数之前。

在 s3c64xx.c 文件的

```
#ifdef CONFIG_SYS_S3C_NAND_HWECC
......
......
/*
 * This function is called before encoding ecc codes to ready ecc engine.
 * Written by jsgood
 */
static void s3c_nand_enable_hwecc(struct mtd_info *mtd, int mode)
```

之间添加：

```
#if defined(CONFIG_NAND_BL1_8BIT_ECC) && (defined(CONFIG_S3C6410) ||
defined(CONFIG_S3C6430))
/*************************************************************
 * jsgood: Temporary 8 Bit H/W ECC supports for BL1 (6410/6430 only)
 *************************************************************/

/*
 * Function for checking ECCEncDone in NFSTAT
 * Written by jsgood
 */
static void s3c_nand_wait_enc(void)
{
    while (!(readl(NFSTAT) & NFSTAT_ECCENCDONE)) {}
}
```

```
/*
 * Function for checking ECCDecDone in NFSTAT
 * Written by jsgood
 */
static void s3c_nand_wait_dec(void)
{
    while (!(readl(NFSTAT) & NFSTAT_ECCDECDONE)) {}
}

/*
 * Function for checking ECC Busy
 * Written by jsgood
 */
static void s3c_nand_wait_ecc_busy(void)
{
    while (readl(NFESTAT0) & NFESTAT0_ECCBUSY) {}
}

static void s3c_nand_wait_ecc_busy_8bit(void)
{
    while (readl(NF8ECCERR0) & NFESTAT0_ECCBUSY) {}
}

void s3c_nand_enable_hwecc_8bit(struct mtd_info *mtd, int mode)
{
    u_long nfcont, nfconf;

    cur_ecc_mode = mode;

    /* 8 bit selection */
    nfconf = readl(NFCONF);

    nfconf &= ~(0x3 << 23);
    nfconf |= (0x1 << 23);

    writel(nfconf, NFCONF);

    /* Initialize & unlock */
    nfcont = readl(NFCONT);
    nfcont |= NFCONT_INITECC;
    nfcont &= ~NFCONT_MECCLOCK;

    if (mode == NAND_ECC_WRITE)
        nfcont |= NFCONT_ECC_ENC;
```

```
    else if (mode == NAND_ECC_READ)
        nfcont &= ~NFCONT_ECC_ENC;

    writel(nfcont, NFCONT);
}

int s3c_nand_calculate_ecc_8bit(struct mtd_info *mtd, const u_char
*dat, u_char *ecc_code)
{
    u_long nfcont, nfm8ecc0, nfm8ecc1, nfm8ecc2, nfm8ecc3;

    /* Lock */
    nfcont = readl(NFCONT);
    nfcont |= NFCONT_MECCLOCK;
    writel(nfcont, NFCONT);

    if (cur_ecc_mode == NAND_ECC_READ)
        s3c_nand_wait_dec();
    else {
        s3c_nand_wait_enc();

        nfm8ecc0 = readl(NFM8ECC0);
        nfm8ecc1 = readl(NFM8ECC1);
        nfm8ecc2 = readl(NFM8ECC2);
        nfm8ecc3 = readl(NFM8ECC3);

        ecc_code[0] = nfm8ecc0 & 0xff;
        ecc_code[1] = (nfm8ecc0 >> 8) & 0xff;
        ecc_code[2] = (nfm8ecc0 >> 16) & 0xff;
        ecc_code[3] = (nfm8ecc0 >> 24) & 0xff;
        ecc_code[4] = nfm8ecc1 & 0xff;
        ecc_code[5] = (nfm8ecc1 >> 8) & 0xff;
        ecc_code[6] = (nfm8ecc1 >> 16) & 0xff;
        ecc_code[7] = (nfm8ecc1 >> 24) & 0xff;
        ecc_code[8] = nfm8ecc2 & 0xff;
        ecc_code[9] = (nfm8ecc2 >> 8) & 0xff;
        ecc_code[10] = (nfm8ecc2 >> 16) & 0xff;
        ecc_code[11] = (nfm8ecc2 >> 24) & 0xff;
        ecc_code[12] = nfm8ecc3 & 0xff;
    }

    return 0;
}
```

```
int s3c_nand_correct_data_8bit(struct mtd_info *mtd, u_char *dat,
u_char *read_ecc, u_char *calc_ecc)
{
    int ret = -1;
    u_long nf8eccerr0, nf8eccerr1, nf8eccerr2, nfmlc8bitpt0,
nfmlc8bitpt1;
    u_char err_type;

    s3c_nand_wait_ecc_busy_8bit();

    nf8eccerr0 = readl(NF8ECCERR0);
    nf8eccerr1 = readl(NF8ECCERR1);
    nf8eccerr2 = readl(NF8ECCERR2);
    nfmlc8bitpt0 = readl(NFMLC8BITPT0);
    nfmlc8bitpt1 = readl(NFMLC8BITPT1);

    err_type = (nf8eccerr0 >> 25) & 0xf;

    /* No error, If free page (all 0xff) */
    if ((nf8eccerr0 >> 29) & 0x1)
        err_type = 0;

    switch (err_type)
    {
    case 8: /* 8 bit error (Correctable) */
        dat[(nf8eccerr2 >> 22) & 0x3ff] ^= ((nfmlc8bitpt1 >> 24) & 0xff);
        printk("s3c-nand: %d bit(s) error detected, corrected
successfully\n", err_type);

    case 7: /* 7 bit error (Correctable) */
        dat[(nf8eccerr2 >> 11) & 0x3ff] ^= ((nfmlc8bitpt1 >> 16) & 0xff);
        printk("s3c-nand: %d bit(s) error detected, corrected
successfully\n", err_type);

    case 6: /* 6 bit error (Correctable) */
        dat[nf8eccerr2 & 0x3ff] ^= ((nfmlc8bitpt1 >> 8) & 0xff);
        printk("s3c-nand: %d bit(s) error detected, corrected
successfully\n", err_type);

    case 5: /* 5 bit error (Correctable) */
        dat[(nf8eccerr1 >> 22) & 0x3ff] ^= (nfmlc8bitpt1 & 0xff);
        printk("s3c-nand: %d bit(s) error detected, corrected
successfully\n", err_type);
```

```
    case 4: /* 4 bit error (Correctable) */
        dat[(nf8eccerr1 >> 11) & 0x3ff] ^= ((nfmlc8bitpt0 >> 24) & 0xff);
        printk("s3c-nand: %d bit(s) error detected, corrected
successfully\n", err_type);

    case 3: /* 3 bit error (Correctable) */
        dat[nf8eccerr1 & 0x3ff] ^= ((nfmlc8bitpt0 >> 16) & 0xff);
        printk("s3c-nand: %d bit(s) error detected, corrected
successfully\n", err_type);

    case 2: /* 2 bit error (Correctable) */
        dat[(nf8eccerr0 >> 15) & 0x3ff] ^= ((nfmlc8bitpt0 >> 8) & 0xff);
        printk("s3c-nand: %d bit(s) error detected, corrected
successfully\n", err_type);



    case 1: /* 1 bit error (Correctable) */
        printk("s3c-nand: %d bit(s) error detected, corrected
successfully\n", err_type);
        dat[nf8eccerr0 & 0x3ff] ^= (nfmlc8bitpt0 & 0xff);
        ret = err_type;
        break;

    case 0: /* No error */
        ret = 0;
        break;

    }

    return ret;
}

void s3c_nand_write_page_8bit(struct mtd_info *mtd, struct nand_chip
                              *chip,const uint8_t *buf)
{
    // printf("Uboot-SD write 8bit ecc.....................\n");

    int i, eccsize = 512;
    int eccbytes = 13;
    int eccsteps = mtd->writesize / eccsize;
    uint8_t *ecc_calc = chip->buffers->ecccalc;
    uint8_t *p = buf;

    for (i = 0; eccsteps; eccsteps--, i += eccbytes, p += eccsize) {
```

```
        s3c_nand_enable_hwecc_8bit(mtd, NAND_ECC_WRITE);
        chip->write_buf(mtd, p, eccsize);
        s3c_nand_calculate_ecc_8bit(mtd, p, &ecc_calc[i]);
    }

    for (i = 0; i < eccbytes * (mtd->writesize / eccsize); i++)
        //jkeqiang
        //chip->oob_poi[i] = ecc_calc[i];
        chip->oob_poi[i+24] = ecc_calc[i];
#if 0
        int k=0;
        for(k=0;k<mtd->oobsize;k++)
        {

            if((k+1)%6==0)
            {
                printf("%x ",chip->oob_poi[k]);
                printf("\n");

            }else
            {
                printf("%x ",chip->oob_poi[k]);

            }

        }

        printf("\n");
        printf("<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<\n");
#endif

    chip->write_buf(mtd, chip->oob_poi, mtd->oobsize);
}

int s3c_nand_read_page_8bit(struct mtd_info *mtd, struct nand_chip
                            *chip,uint8_t *buf)
{
    int i, stat, eccsize = 512;
    int eccbytes = 13;
    int eccsteps = mtd->writesize / eccsize;
    int col = 0;
    uint8_t *p = buf;

    /* Step1: read whole oob */
```

```
    col = mtd->writesize;
    chip->cmdfunc(mtd, NAND_CMD_RNDOUT, col, -1);
    chip->read_buf(mtd, chip->oob_poi, mtd->oobsize);

    col = 0;
    for (i = 0; eccsteps; eccsteps--, i += eccbytes, p += eccsize) {
        chip->cmdfunc(mtd, NAND_CMD_RNDOUT, col, -1);
        s3c_nand_enable_hwecc_8bit(mtd, NAND_ECC_READ);
        chip->read_buf(mtd, p, eccsize);
        //jkeqiang
        //     chip->write_buf(mtd, chip->oob_poi + (((mtd->writesize
/ eccsize) - eccsteps) * eccbytes), eccbytes);
        chip->write_buf(mtd, chip->oob_poi + 24 + (((mtd->writesize /
eccsize) - eccsteps) * eccbytes), eccbytes);
        s3c_nand_calculate_ecc_8bit(mtd, 0, 0);
        stat = s3c_nand_correct_data_8bit(mtd, p, 0, 0);

        if (stat == -1)
            mtd->ecc_stats.failed++;

        col = eccsize * ((mtd->writesize / eccsize) + 1 - eccsteps);
    }

    return 0;
}

/*************************************************************/
#else
```

……

……

```
    nand->ecc.hwctl     = s3c_nand_enable_hwecc;
    nand->ecc.calculate = s3c_nand_calculate_ecc;
    nand->ecc.correct   = s3c_nand_correct_data;
```

修改为：

```
#ifdef CONFIG_NAND_BL1_8BIT_ECC
    debug("USE HWECC 8BIT\n");
    nand->ecc.hwctl = s3c_nand_enable_hwecc_8bit;
    nand->ecc.calculate = s3c_nand_calculate_ecc_8bit;
    nand->ecc.correct = s3c_nand_correct_data_8bit;
    nand->ecc.read_page = s3c_nand_read_page_8bit;
    nand->ecc.write_page = s3c_nand_write_page_8bit;
#else
    printf("USE HWECC default\n");
```

```
    nand->ecc.hwctl     = s3c_nand_enable_hwecc;
    nand->ecc.calculate = s3c_nand_calculate_ecc;
    nand->ecc.correct   = s3c_nand_correct_data;
#endif
```

进入 commom/cmd_nand.c 文件中，在 do_nand 函数中的：

```
if (strncmp(cmd, "read", 4) == 0 || strncmp(cmd, "write", 5) == 0)
{
        size_t rwsize;
        ulong pagecount = 1;
        int read;
        int raw;
```

后面添加：

```
        s = strchr(cmd, '.');

         if (!strcmp(s, ".uboot")&&!read)  {

            printf("Re-program the u-boot to nand flash\n");
            if (argc < 3)
                goto usage;

            addr = (ulong)simple_strtoul(argv[2], NULL, 16);
            run_command ("nand erase 0 100000", 0);
            //printf("nand erase 0 100000......\n");
            size = 0x1000;
            puts("\nwrite u-boot to nand flash......1 \n");
            for (i = 0; i < 4; i++, addr += 0x800)
            {
                nand_write_skip_bad(nand, size*i, (size_t *)&size,
(u_char *)(addr), 0);
            }
            puts("write u-boot to nand flash......2 \n");
            for(i=4; i< 64; i++, addr += 0x1000)
            {
                nand_write_skip_bad(nand, size*i, (size_t *)&size,
(u_char *)(addr), 0);
            }
            return 0;
        }
```

……

```
U_BOOT_CMD(
```

```
nand, CONFIG_SYS_MAXARGS, 1, do_nand,
```

在这添加：

```
"nand write.uboot - memaddr - Re-program the u-boot to nand flash\n"
```

其实这里就是添加 nand write.uboot 命令。

ECC 校验完成。

通过图 4. 6 可以看到 Net 的网卡是 CS8900，但是 OK6410 得网卡是 DM9000。接下来需要移植的网卡部分。

第 5 章 u-boot-2012.10 移植之网卡驱动

5.1 DM9000 网卡驱动移植

由于 u-boot-2012.10 默认支持的是 CS8900 网卡，而 OK6410 开发平台上的网卡是 DM9000，则需要对网卡驱动部分进行修改，使得 u-boot-2012.10 支持 DM9000 的网卡驱动。

进入 smdk6410.h 文件，找到有关 CS8900 驱动的宏定义：

```
/*
 * Hardware drivers
 */
#define CONFIG_CS8900            /* we have a CS8900 on-board  */
#define CONFIG_CS8900_BASE       0x18800300
#define CONFIG_CS8900_BUS16      /* follow the Linux driver */
```

将这段代码用

```
#if 0
......
#endif
```

注释掉，添加有关 DM9000 网卡的宏定义：

```
/*
 *  DM9000 drivers
 */
#define CONFIG_NET_MULTI        1
#define CONFIG_DM9000_NO_SROM   1
#define CONFIG_dm9000
#define CONFIG_DRIVER_DM9000    1
#define CONFIG_DM9000_BASE      0x18800300
#define DM9000_IO               CONFIG_DM9000_BASE
#define DM9000_DATA             (CONFIG_DM9000_BASE+4)
#define CONFIG_DM9000_USE_16BIT 1

#define CONFIG_ETHADDR          00:40:5c:26:0a:5b
#define CONFIG_NETMASK          255.255.255.0
#define CONFIG_IPADDR           172.16.114.20
#define CONFIG_SERVERIP         172.16.114.10
#define CONFIG_GATEWAYIP        172.16.114.1
//#define CONFIG_DM9000_DEBUG
```

其中 IP 地址、子网掩码等这些数据根据具体情况进行修改。

进入 smdk6410.c 这个文件，在 int board_eth_init(bd_t *bis)函数中，添加 DM9000 的初始化：

```
#if defined(CONFIG_DRIVER_DM9000)
   rc = dm9000_initialize(bis);
```

```
#endif
```

编译启动，如图 5. 1 所示。

图 5. 1 DM9000 网卡驱动

从图 5. 1 可以看到 Net 已经为 DM9000，此时可以使用 ping www.baidu.com 进行检测，查看 DM9000 是否有效。

使用 ok6410ping 宿主机 ubuntu：

```
zzq6410 >>> ping 192.168.1.2
dm9000 i/o: 0x18000300, id: 0x90000a46
DM9000: running in 16 bit mode
MAC: 00:40:5c:26:0a:5b
operating at 100M full duplex mode
Using dm9000 device
host 192.168.1.2 is alive
```

如果你的开发板此时出现了

```
*** Warning - bad CRC, using default environment
```

那说明你的 NandFlash 出现了坏块，如图 5. 1 所示也提示出现了坏块，那么可停在 u-boot 中，键入 nand bad 查看坏块情况，接着键入 nand scrub 进行坏块清除。

在使用开发板过程中难免会出现坏块，但是我还是建议尽量少使用 nand scrub 这个指令。

5.2 支持 TFTP

在 ubuntu 系统中下载 tftp 所需软件包，建立/tftpboot，并且赋予/tftpboot 最大的权限。

```
sudo apt-get install tftp-hpa tftpd-hpa
mkdir /tftpboot
chmod 777 tftproot
```

修改 tftp 设置，

```
sudo vim /etc/default/tftpd-hpa
```

```
# /etc/default/tftpd-hpa

TFTP_USERNAME="tftp"

#TFTP_DIRECTORY="/var/lib/tftpboot"
TFTP_DIRECTORY="/tftpboot"

TFTP_ADDRESS="0.0.0.0:69"

#TFTP_OPTIONS="--secure"
TFTP_OPTIONS="-l-c-s"
```

完成之后重启 tftp 服务即可。

```
sudo service tftpd-hpa restart
```

第 6 章 u-boot-2012.10 移植之 USB 驱动

6.1 USB 驱动

从华为网盘中 http://dl.vmall.com/c0a2nblpbd，下载 cmd_usbd.c、usbd-otg-hs.c 及其 usbd-otg-hs.h 三个文件。这三个文件是 USB 驱动文件，作者已做 s3c6410 的修改，下载之后直接可用。

将 cmd_usbd.c 放在 common 文件加下，并在 common 文件夹下的 Makefile 中添加：

```
ifdef CONFIG_CMD_USB
......
COBJS-y += cmd_usbd.o
......
endif
```

将 usbd-otg-hs.c 及其 usbd-otg-hs.h 放在\drivers\usb\host 文件加下，并\drivers\usb\host 文件下的 Makefile 中添加：

```
COBJS-$(CONFIG_USB_S3C64XX) += usbd-otg-hs.o
```

在 s3c64x0.h 中添加 USB 的结构体定义：

```
/* USB HOST (see manual chapter 12) */
typedef struct {
   volatile  u32 HcRevision;
   volatile  u32 HcControl;
   volatile  u32 HcCommonStatus;
   volatile  u32 HcInterruptStatus;
   volatile  u32 HcInterruptEnable;
   volatile  u32 HcInterruptDisable;
   volatile  u32 HcHCCA;
   volatile  u32 HcPeriodCuttendED;
   volatile  u32 HcControlHeadED;
   volatile  u32 HcControlCurrentED;
   volatile  u32 HcBulkHeadED;
   volatile  u32 HcBuldCurrentED;
   volatile  u32 HcDoneHead;
   volatile  u32 HcRmInterval;
   volatile  u32 HcFmRemaining;
   volatile  u32 HcFmNumber;
   volatile  u32 HcPeriodicStart;
   volatile  u32 HcLSThreshold;
   volatile  u32 HcRhDescriptorA;
   volatile  u32 HcRhDescriptorB;
   volatile  u32 HcRhStatus;
   volatile  u32 HcRhPortStatus1;
   volatile  u32 HcRhPortStatus2;
```

```
} /*__attribute__((__packed__))*/ S3C64XX_USB_HOST;
```

在 s3c6410.h 中添加：

```
/*
 * USB2.0 HS OTG (Chapter 26)
 */
#define USBOTG_LINK_BASE   (0x7C000000)
#define USBOTG_PHY_BASE    (0x7C100000)

/* Core Global Registers */
/* OTG Control & Status */
#define S3C_OTG_GOTGCTL    (USBOTG_LINK_BASE + 0x000)
/* OTG Interrupt */
#define S3C_OTG_GOTGINT    (USBOTG_LINK_BASE + 0x004)
/* Core AHB Configuration */
#define S3C_OTG_GAHBCFG    (USBOTG_LINK_BASE + 0x008)
/* Core USB Configuration */
#define S3C_OTG_GUSBCFG    (USBOTG_LINK_BASE + 0x00C)
/* Core Reset */
#define S3C_OTG_GRSTCTL    (USBOTG_LINK_BASE + 0x010)
/* Core Interrupt */
#define S3C_OTG_GINTSTS    (USBOTG_LINK_BASE + 0x014)
/* Core Interrupt Mask */
#define S3C_OTG_GINTMSK    (USBOTG_LINK_BASE + 0x018)
/* Receive Status Debug Read/Status Read */
#define S3C_OTG_GRXSTSR    (USBOTG_LINK_BASE + 0x01C)
/* Receive Status Debug Pop/Status Pop */
#define S3C_OTG_GRXSTSP    (USBOTG_LINK_BASE + 0x020)
/* Receive FIFO Size */
#define S3C_OTG_GRXFSIZ    (USBOTG_LINK_BASE + 0x024)
/* Non-Periodic Transmit FIFO Size */
#define S3C_OTG_GNPTXFSIZ  (USBOTG_LINK_BASE + 0x028)
/* Non-Periodic Transmit FIFO/Queue Status */
#define S3C_OTG_GNPTXSTS   (USBOTG_LINK_BASE + 0x02C)

/* Host Periodic Transmit FIFO Size */
#define S3C_OTG_HPTXFSIZ   (USBOTG_LINK_BASE + 0x100)
/* Device Periodic Transmit FIFO-1 Size */
#define S3C_OTG_DPTXFSIZ1  (USBOTG_LINK_BASE + 0x104)
/* Device Periodic Transmit FIFO-2 Size */
#define S3C_OTG_DPTXFSIZ2  (USBOTG_LINK_BASE + 0x108)
/* Device Periodic Transmit FIFO-3 Size */
#define S3C_OTG_DPTXFSIZ3  (USBOTG_LINK_BASE + 0x10C)
```

```
/* Device Periodic Transmit FIFO-4 Size */
#define S3C_OTG_DPTXFSIZ4  (USBOTG_LINK_BASE + 0x110)
/* Device Periodic Transmit FIFO-5 Size */
#define S3C_OTG_DPTXFSIZ5  (USBOTG_LINK_BASE + 0x114)
/* Device Periodic Transmit FIFO-6 Size */
#define S3C_OTG_DPTXFSIZ6  (USBOTG_LINK_BASE + 0x118)
/* Device Periodic Transmit FIFO-7 Size */
#define S3C_OTG_DPTXFSIZ7  (USBOTG_LINK_BASE + 0x11C)
/* Device Periodic Transmit FIFO-8 Size */
#define S3C_OTG_DPTXFSIZ8  (USBOTG_LINK_BASE + 0x120)
/* Device Periodic Transmit FIFO-9 Size */
#define S3C_OTG_DPTXFSIZ9  (USBOTG_LINK_BASE + 0x124)
/* Device Periodic Transmit FIFO-10 Size */
#define S3C_OTG_DPTXFSIZ10 (USBOTG_LINK_BASE + 0x128)
/* Device Periodic Transmit FIFO-11 Size */
#define S3C_OTG_DPTXFSIZ11 (USBOTG_LINK_BASE + 0x12C)
/* Device Periodic Transmit FIFO-12 Size */
#define S3C_OTG_DPTXFSIZ12 (USBOTG_LINK_BASE + 0x130)
/* Device Periodic Transmit FIFO-13 Size */
#define S3C_OTG_DPTXFSIZ13 (USBOTG_LINK_BASE + 0x134)
/* Device Periodic Transmit FIFO-14 Size */
#define S3C_OTG_DPTXFSIZ14 (USBOTG_LINK_BASE + 0x138)
/* Device Periodic Transmit FIFO-15 Size */
#define S3C_OTG_DPTXFSIZ15 (USBOTG_LINK_BASE + 0x13C)

/* Host Global Registers */
/* Host Configuration */
#define S3C_OTG_HCFG       (USBOTG_LINK_BASE + 0x400)
/* Host Frame Interval */
#define S3C_OTG_HFIR       (USBOTG_LINK_BASE + 0x404)
/* Host Frame Number/Frame Time Remaining */
#define S3C_OTG_HFNUM      (USBOTG_LINK_BASE + 0x408)
/* Host Periodic Transmit FIFO/Queue Status */
#define S3C_OTG_HPTXSTS    (USBOTG_LINK_BASE + 0x410)
/* Host All Channels Interrupt */
#define S3C_OTG_HAINT      (USBOTG_LINK_BASE + 0x414)
/* Host All Channels Interrupt Mask */
#define S3C_OTG_HAINTMSK   (USBOTG_LINK_BASE + 0x418)

/* Host Port Control & Status Registers */
/* Host Port Control & Status */
#define S3C_OTG_HPRT       (USBOTG_LINK_BASE + 0x440)

/* Host Channel-Specific Registers */
```

```
/* Host Channel-0 Characteristics */
#define S3C_OTG_HCCHAR0    (USBOTG_LINK_BASE + 0x500)
/* Host Channel-0 Split Control */
#define S3C_OTG_HCSPLT0    (USBOTG_LINK_BASE + 0x504)
/* Host Channel-0 Interrupt */
#define S3C_OTG_HCINT0     (USBOTG_LINK_BASE + 0x508)
/* Host Channel-0 Interrupt Mask */
#define S3C_OTG_HCINTMSK0  (USBOTG_LINK_BASE + 0x50C)
/* Host Channel-0 Transfer Size */
#define S3C_OTG_HCTSIZ0    (USBOTG_LINK_BASE + 0x510)
/* Host Channel-0 DMA Address */
#define S3C_OTG_HCDMA0     (USBOTG_LINK_BASE + 0x514)


/* Device Global Registers */
/* Device Configuration */
#define S3C_OTG_DCFG       (USBOTG_LINK_BASE + 0x800)
/* Device Control */
#define S3C_OTG_DCTL       (USBOTG_LINK_BASE + 0x804)
/* Device Status */
#define S3C_OTG_DSTS       (USBOTG_LINK_BASE + 0x808)
/* Device IN Endpoint Common Interrupt Mask */
#define S3C_OTG_DIEPMSK    (USBOTG_LINK_BASE + 0x810)
/* Device OUT Endpoint Common Interrupt Mask */
#define S3C_OTG_DOEPMSK    (USBOTG_LINK_BASE + 0x814)
/* Device All Endpoints Interrupt */
#define S3C_OTG_DAINT      (USBOTG_LINK_BASE + 0x818)
/* Device All Endpoints Interrupt Mask */
#define S3C_OTG_DAINTMSK   (USBOTG_LINK_BASE + 0x81C)
/* Device IN Token Sequence Learning Queue Read 1 */
#define S3C_OTG_DTKNQR1    (USBOTG_LINK_BASE + 0x820)
/* Device IN Token Sequence Learning Queue Read 2 */
#define S3C_OTG_DTKNQR2    (USBOTG_LINK_BASE + 0x824)
/* Device VBUS Discharge Time */
#define S3C_OTG_DVBUSDIS   (USBOTG_LINK_BASE + 0x828)
/* Device VBUS Pulsing Time */
#define S3C_OTG_DVBUSPULSE (USBOTG_LINK_BASE + 0x82C)
/* Device IN Token Sequence Learning Queue Read 3 */
#define S3C_OTG_DTKNQR3    (USBOTG_LINK_BASE + 0x830)
/* Device IN Token Sequence Learning Queue Read 4 */
#define S3C_OTG_DTKNQR4    (USBOTG_LINK_BASE + 0x834)

/* Device Logical IN Endpoint-Specific Registers */
```

```
/* Device IN Endpoint 0 Control */
#define S3C_OTG_DIEPCTL0   (USBOTG_LINK_BASE + 0x900)
/* Device IN Endpoint 0 Interrupt */
#define S3C_OTG_DIEPINT0   (USBOTG_LINK_BASE + 0x908)
/* Device IN Endpoint 0 Transfer Size */
#define S3C_OTG_DIEPTSIZ0  (USBOTG_LINK_BASE + 0x910)
/* Device IN Endpoint 0 DMA Address */
#define S3C_OTG_DIEPDMA0   (USBOTG_LINK_BASE + 0x914)

/* Device Logical OUT Endpoint-Specific Registers */
/* Device OUT Endpoint 0 Control */
#define S3C_OTG_DOEPCTL0   (USBOTG_LINK_BASE + 0xB00)
/* Device OUT Endpoint 0 Interrupt */
#define S3C_OTG_DOEPINT0   (USBOTG_LINK_BASE + 0xB08)
/* Device OUT Endpoint 0 Transfer Size */
#define S3C_OTG_DOEPTSIZ0  (USBOTG_LINK_BASE + 0xB10)
/* Device OUT Endpoint 0 DMA Address */
#define S3C_OTG_DOEPDMA0   (USBOTG_LINK_BASE + 0xB14)

/* Power & clock gating registers */
#define S3C_OTG_PCGCCTRL   (USBOTG_LINK_BASE + 0xE00)

/* Endpoint FIFO address */
#define S3C_OTG_EP0_FIFO   (USBOTG_LINK_BASE + 0x1000)


/* OTG PHY CORE REGISTERS */
#define S3C_OTG_PHYPWR     (USBOTG_PHY_BASE+0x00)
#define S3C_OTG_PHYCTRL    (USBOTG_PHY_BASE+0x04)
#define S3C_OTG_RSTCON     (USBOTG_PHY_BASE+0x08)
```

```
static inline S3C64XX_USB_HOST *S3C64XX_GetBase_USB_HOST(void)
{
   return (S3C64XX_USB_HOST *)ELFIN_USB_HOST_BASE;
}
```

上面这个函数中用到了 inline，在 c++中，为了解决一些频繁调用的小函数大量消耗栈

空间或者是叫栈内存的问题，特别的引入了 inline 修饰符，表示为内联函数。

栈空间就是指放置程序的局部数据也就是函数内数据的内存空间，在系统下，栈空间是有限的，如果频繁大量的使用就会造成因栈空间不足所造成的程序出错的问题，函数的死循环递归调用的最终结果就是导致栈内存空间枯竭。

修改之后的 s3c64x0.h 和 s3c6410.h 文件我已经上传至华为网盘 http://dl.vmall.com/c0a2nblpbd，读者可自行下载使用。

编译启动，如图 6.1 所示。

图 6.1 USB 驱动

第 7 章 u-boot-2012.10 移植之 MMC 驱动

7.1 MMC 驱动

进入 include/configs/，在 smdk6410.h 中添加：

```
/*
 *  MMC
 */
#define CONFIG_GENERIC_MMC   1
#define CONFIG_MMC           1
#define CONFIG_S3C64X0_MMC   1
#define CONFIG_CMD_MMC            /* MMC support */
```

在 smdk6410.c 中添加：

```
#ifdef CONFIG_GENERIC_MMC
int board_mmc_init(bd_t *bis)
{
    return s3c64x0_mmc_init(0);
}
#endif
```

进入 drivers/mmc，打开 Makefile，添加：

```
COBJS-$(CONFIG_S3C64X0_MMC) = s3c64x0_mmc.o
```

打开 s3c64x0.h，添加：

```
struct s3c64x0_mmc {
    unsigned int     sysad;
    unsigned short   blksize;
    unsigned short   blkcnt;
    unsigned int     argument;
    unsigned short   trnmod;
    unsigned short   cmdreg;
    unsigned int     rspreg0;
    unsigned int     rspreg1;
    unsigned int     rspreg2;
    unsigned int     rspreg3;
    unsigned int     bdata;
    unsigned int     prnsts;
    unsigned char    hostctl;
    unsigned char    pwrcon;
    unsigned char    blkgap;
```

```
    unsigned char    wakcon;
    unsigned short   clkcon;
    unsigned char    timeoutcon;
    unsigned char    swrst;
    unsigned int     norintsts;    /* errintsts */
    unsigned int     norintstsen;  /* errintstsen */
    unsigned int     norintsigen;  /* errintsigen */
    unsigned short   acmd12errsts;
    unsigned char    res1[2];
    unsigned int     capareg;
    unsigned char    res2[4];
    unsigned int     maxcurr;
    unsigned char    res3[0x34];
    unsigned int     control2;
    unsigned int     control3;
    unsigned int     control4;
    unsigned char    res4[0x6e];
    unsigned short   hcver;
    unsigned char    res5[0xFFF02];
};
```

在 s3c6410.h 中添加：

```
struct mmc_host {
    struct s3c64x0_mmc *reg;
    unsigned int version;  /* SDHCI spec. version */
    unsigned int clock; /* Current clock (MHz) */
};
int s3c64x0_mmc_init(int dev_index);
```

在华为网盘中下载 s3c64x0_mmc.c，将其放在/drivers/mmc 中。

在 include/mmc.h 中的 struct mmc 结构体中添加：

```
int (*detect_mmc)(struct mmc_host *mmc_host);
```

在 lowlevel_init 中修改：

```
    /* FOUT of EPLL is 96MHz */
    /*ldr  r1, =0x200203*/
    ldr r1, =0x80200203
    str r1, [r0, #EPLL_CON0_OFFSET]
    ldr r1, =0x0
    str r1, [r0, #EPLL_CON1_OFFSET]
```

第 8 章 u-boot-2012.10 移植之添加 u-boot 命令

8.1 小试 u-boot 命令

在 s3c6410.h 中的：

```
/*-----------------------------------------------------------------
 * Physical Memory Map
 */
 /* burst 4, 13-bit row, 10-bit col */
#define DMC1_MEM_CFG    0x00010012
#define DMC1_MEM_CFG2   0xB45
/* 0x5000_0000~0x5fff_ffff (256 MiB) */
#define DMC1_CHIP0_CFG  0x150F8
#define DMC_DDR_32_CFG  0x0        /* 32bit, DDR */
```

修改成：

```
/*-----------------------------------------------------------------
 * Physical Memory Map
 */
/* burst 4, 14-bit row, 10-bit col */
#define DMC1_MEM_CFG    0x0001001a
#define DMC1_MEM_CFG2   0xB45
/* 0x5000_0000~0x5fff_ffff (256 MiB) */
#define DMC1_CHIP0_CFG  0x150F0
#define DMC_DDR_32_CFG  0x0        /* 32bit, DDR */
```

如果你是在华为网盘中下载的 s3c64x0.h 和 s3c6410.h，那么无需修改，华为网盘中的这两个文件是作者最后完成之后上传的。

打开 command.h 头文件中，可以找到 cmd_tbl_t 这个结构体，u-boot 中每个命令都用这样的一个结构体来描述，类型定义如下：

```
typedef struct cmd_tbl_s cmd_tbl_t;
struct cmd_tbl_s {
    char *name; // 命令的名称
    int maxargs; // 最多支持的参数的个数
    int repeatable; // 是否可重复执行
    // 命令对应的处理函数
    int (*cmd)(struct cmd_tbl_s *, int, int, char *const[]);
    char *usage; // 命令简要使用信息
    char *help; // 命令详细帮助信息
}
```

按照这个格式，现在 commom 文件夹下建一个 cmd_zzq.c 文件：

```
#include <common.h>
#include <command.h>

int do_zzq(cmd_tbl_t *cmdtp, int flag, int argc, char * const argv[])
{
    printf("Name   :  zhuzhaoqi!\n");
    printf("E-mail:  jxlgzzq@163.com\n");
    return 0;
}

U_BOOT_CMD(zzq, CONFIG_SYS_MAXARGS, 1, do_zzq,
          "usage info", "help info");
```

同时修改 commom 文件夹下的 Makefile，添加：

```
COBJS-$(CONFIG_CMD_TEST) += cmd_zzq.o
```

打开 smdk6410.h 文件，添加 CONFIG_CMD_TEST 的宏定义：

```
#define      CONFIG_CMD_TEST
```

编译、烧写、启动，通过 DNW 看到如图 8.1 所示。

图 8.1　自行添加 u-boot 命令

其实添加 u-boot 命令在 4.2 章节就进行过，读者可自行添加自己想要的 u-boot 命令。

第 9 章 Linux3.6.7 移植之 make menuconfig

9.1 mkimage

mkimage 这个工具位于 u-boot-2012.10 中的 tools 文件夹下，mkimage 可以可以用来制作不压缩或者压缩的多种可启动映象文件。mkimage 在制作映象文件的时候，是在原来的可执行映象文件的前面加上一个 0x40 字节的头，记录参数所指定的信息，这样 uboot 才能识别这个映象是针对哪个 CPU 体系结构的，哪个 OS 的，哪种类型，加载内存中的哪个位置， 入口点在内存的哪个位置以及映象名是什么。

```
zhuzhaoqi@zhuzhaoqi-desktop:~/u-boot/u-boot-2012.10/u-boot-2012.1
0/tools$ ./mkimage
Usage: ./mkimage -l image
         -l ==> list image header information
      ./mkimage [-x] -A arch -O os -T type -C comp -a addr -e ep -n
name -d data_file[:data_file...] image
         -A ==> set architecture to 'arch'
         -O ==> set operating system to 'os'
         -T ==> set image type to 'type'
         -C ==> set compression type 'comp'
         -a ==> set load address to 'addr' (hex)
         -e ==> set entry point to 'ep' (hex)
         -n ==> set image name to 'name'
         -d ==> use image data from 'datafile'
         -x ==> set XIP (execute in place)
      ./mkimage [-D dtc_options] -f fit-image.its fit-image
      ./mkimage -V ==> print version information and exit
```

-A 指定 CPU 的体系结构，如表 9. 1 所示。

表 9. 1 CPU 体系结构

取值	表示的体系结构	取值	表示的体系结构
alpha	Alpha	arm	ARM
x86	Intel x86	ia64	IA64
mips	MIPS	mips64	MIPS 64 Bit
ppc	PowerPC	s390	IBM S390
sh	SuperH	sparc	SPARC
sparc64	SPARC 64 Bit	m68k	MC68000

-O 指定操作系统类型，可以取以下值：

openbsd、netbsd、freebsd、4_4bsd、linux、svr4、esix、solaris、irix、sco、dell、ncr、lynxos、vxworks、psos、qnx、u-boot、rtems、artos。

-T 指定映象类型，可以取以下值：

standalone、kernel、ramdisk、multi、firmware、script、filesystem。

-C 指定映象压缩方式，可以取以下值：

none 不压缩；

gzip 用 gzip 的压缩方式；

bzip2 用 bzip2 的压缩方式。

-a 指定映象在内存中的加载地址，映象下载到内存中时，要按照用 mkimage 制作映象时，这个参数所指定的地址值来下载。

-e 指定映象运行的入口点地址，这个地址就是-a 参数指定的值加上 0x40（因为前面有个 mkimage 添加的 0x40 个字节的头）。

-n 指定映象名。

-d 指定制作映象的源文件。

现在将 u-boot-2012.10 下的 tools 这个文件夹下中的 mkimage 这个工具复制到/user/bin 下。

9.2 配置 menuconfig

打开最顶层的 Makefile，有这么两行程序。

```
ARCH ?= $(SUBARCH)
CROSS_COMPILE ?= $(CONFIG_CROSS_COMPILE:"%"=%)
```

ARCH 是 CPU 体系结构，OK6410 是 arm，那么这句就得修改成 arm。CROSS_COMPILE 是编译工具链，和 u-boot 配置一样。则修改成：

```
ARCH ?= arm
CROSS_COMPILE ?= /usr/local/arm/4.4.1/bin/arm-linux-
```

进入 arch/arm/mach-s3c64xx，打开 Kconfig 文件。其中：

```
# S3C6410 machine support
```

所支持的有：

```
config MACH_ANW6410
config MACH_MINI6410
config MACH_REAL6410
config MACH_SMDK6410
```

但是没有 OK6410，那么现在就修改文件，使得 Linux3.6.7 能适用于 OK6410 开发平台。修改肯定是在以上的四种平台上进行，就取 MINI6410。

在当前 arch/arm/mach-s3c64xx 文件下，复制一份 mach-mini6410.c 并且重命名为 mach-ok6410.c。使用命令：

```
cp mach-mini6410.c mach-ok6410.c
```

打开 mach-ok6410.c 文件，将 mini6410(MINI6410)修改为 ok6410(OK6410)。这里可以使用批量字符串替换，方法较多。

回到 Kconfig 文件，在

```
config MACH_MINI6410
```

后面添加 OK6410 的配置：

```
config MACH_OK6410
    bool "OK6410"
    select CPU_S3C6410
    select SAMSUNG_DEV_ADC
    select S3C_DEV_HSMMC
    select S3C_DEV_HSMMC1
    select S3C_DEV_I2C1
    select SAMSUNG_DEV_IDE
    select S3C_DEV_FB
    select S3C_DEV_RTC
    select SAMSUNG_DEV_TS
    select S3C_DEV_USB_HOST
#   select S3C_DEV_USB_HSOTG
    select S3C_DEV_WDT
    select SAMSUNG_DEV_KEYPAD
    select SAMSUNG_DEV_PWM
    select HAVE_S3C2410_WATCHDOG if WATCHDOG
    select S3C64XX_SETUP_SDHCI
    select S3C64XX_SETUP_I2C1
    select S3C64XX_SETUP_IDE
    select S3C64XX_SETUP_FB_24BPP
    select S3C64XX_SETUP_KEYPAD
    help
      Machine support for the feiling OK6410
```

打开 Makefile，在

```
obj-$(CONFIG_MACH_MINI6410)            += mach-mini6410.o
```

后面添加 ok6410 的配置：

```
obj-$(CONFIG_MACH_OK6410)              += mach-ok6410.o
```

进入 arch/arm/tools，打开 mach-types 文件，在

```
machine_is_xxx    CONFIG_xxxx         MACH_TYPE_xxx          number
mini6410          MACH_MINI6410        MINI6410               2520
```

后面添加：

```
ok6410            MACH_OK6410          OK6410                 1626
```

回到 linux3.6.7 文件夹下，配置 menuconfig，输入：

```
zhuzhaoqi@zhuzhaoqi-desktop:~/Linux/linux-3.6.7$ make menuconfig
```

如果你的虚拟机之前没有更新过或者没安装过编译 linux 的相关软件，应该会出现如图

9.1 错误。

```
zhuzhaoqi@zhuzhaoqi-desktop:~/Linux/linux-3.6.7$ make menuconfig
 *** Unable to find the ncurses libraries or the
 *** required header files.
 *** 'make menuconfig' requires the ncurses libraries.
 ***
 *** Install ncurses (ncurses-devel) and try again.
 ***
make[1]: *** [scripts/kconfig/dochecklxdialog] 错误 1
make: *** [menuconfig] 错误 2
```

图 9.1 make menuconfig 报错

这里提示没有安装 ncurses 这个库。那么接下来进行安装。

```
sudo apt-get install libncurses*
```

为了使得在 make xconfig 不出错，现在将 build-essential、kernel-package 和两个 QT 库 libqt3-headers、libqt3-mt-dev 安装。

make menuconfig 提供一个基于文本的图形界面，它依赖于 ncurses5 这个包，键盘操作，可以修改选项，一般推荐用这个；make xconfig 需要你有 x window system 支持，就是说你要在 KDE、GNOME 之类的 X 桌面环境下才可用，好处是支持鼠标，坏处是 X 本身占用系统周期，而且 X 环境容易引起编译器的不稳定。

再一次输入：

```
zhuzhaoqi@zhuzhaoqi-desktop:~/Linux/linux-3.6.7$ make menuconfig
```

如果上面的库都安装好了的话，就会出现如图 9.2 所示界面。

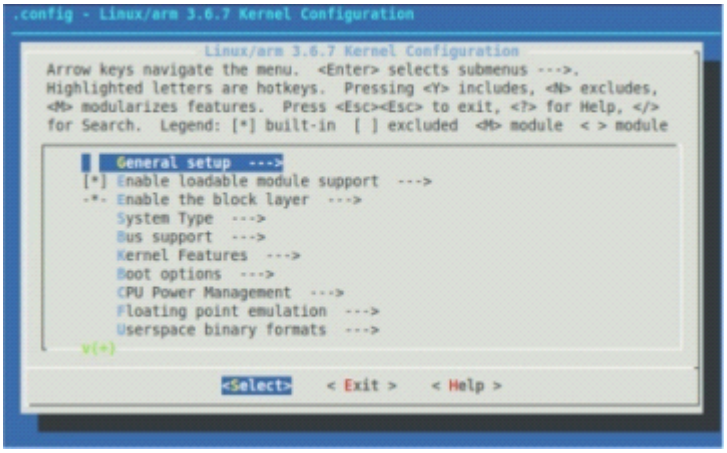


图 9.2 make menuconfig 界面

先选择 Load an Alternate Configuration File，输入 arch/arm/configs/s3c6400_defconfig，操作如图 9.3 所示。

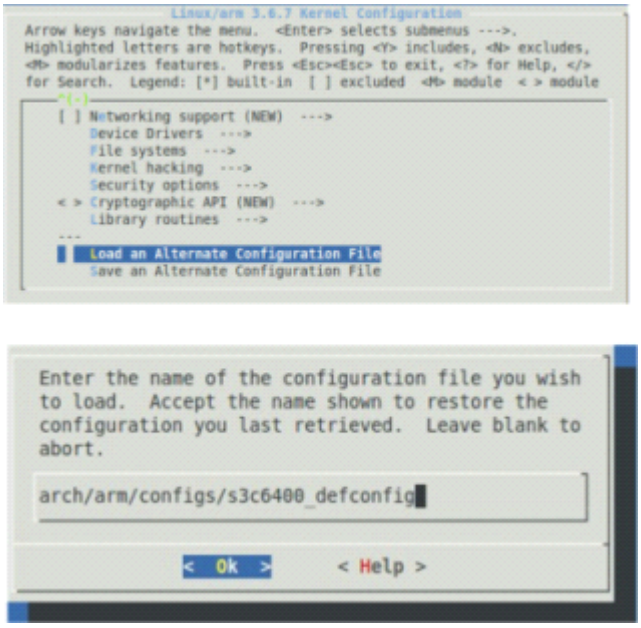


图 9.3 menuconfig 配置第一步

进入 General Setup，打开 Cross_compiler tool perfix，输入 编译工具链存放处 /usr/local/arm/4.4.1/bin/arm-linux- 。操作如图 9.4 所示。

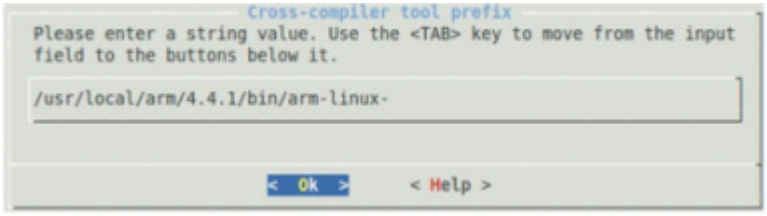


图 9.4 menuconfig 配置第二步

进入 System Type,取消 SMDK6400 ，A&W6410，SMDK6410 等平台，只选择 OK6410。操作如图 9.5 所示。

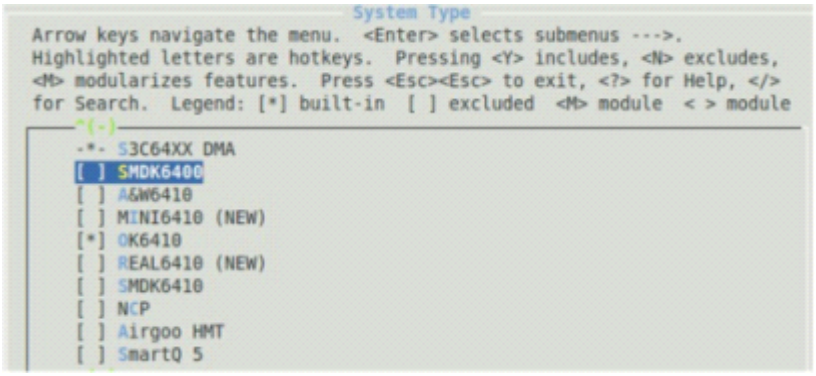


图 9.5 menuconfig 配置第三步

进入 Save an Alternate Configuration File，保存为.config 然后退出。

接着就是 make uImage，如果前面操作没错的话，应该是编译成功了的：

```
  ......
  OBJCOPY arch/arm/boot/zImage
  Kernel: arch/arm/boot/zImage is ready
  UIMAGE  arch/arm/boot/uImage
Image Name:   Linux-3.6.7
Created:      Tue Dec 11 16:55:32 2012
Image Type:   ARM Linux Kernel Image (uncompressed)
Data Size:    1618312 Bytes = 1580.38 kB = 1.54 MB
Load Address: 50008000
Entry Point:  50008000
  Image arch/arm/boot/uImage is ready
```

Load Address 和 Entry Point 是存在问题，这个我们后续解决。进入 arch/arm/boot，可以看到已经生成了 uImage 和 zImage。将 uImage(需要将名称改为 zImage)与 u-boot.bin 放在一起烧写，看看 u-boot 是否能引导内核。

引导之后通过 dnw 显示出来的信息是：

```
NAND read: device 0 offset 0x100000, size 0x500000
 5242880 bytes read: OK
## Booting kernel from Legacy Image at 50018000 ...
   Image Name:   Linux-3.6.7
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    1618312 Bytes = 1.5 MiB
   Load Address: 50008000
   Entry Point:  50008000
   Verifying Checksum ... OK
   Loading Kernel Image ... OK
OK

Starting kernel ...
```

停在这里就不动了，无法启动内核。

这里注意：

```
## Booting kernel from Legacy Image at 50018000 ...
```

但是：

```
   Load Address: 50008000
   Entry Point:  50008000
```

将

```
#define CONFIG_BOOTCOMMAND "nand read 0x50018000 0x100000 0x500000;"
\"bootm 0x50018000"
```

修改成：

```
#define CONFIG_BOOTCOMMAND "nand read 0x50008000 0x100000 0x500000;"
\"bootm 0x50008000"
```

```
//#define CONFIG_BOOTARGS     "console=ttySAC,115200"
#define CONFIG_BOOTARGS       "root=/dev/mtdblock2
rootfstype=cramfs console=ttySAC0,115200"
```

第 10 章 Linux3.6.7 移植之 Load Address 和 Entry Point

10.1 Load Address 和 Entry Point 的分析

在 9.1 章节中有：

```
./mkimage [-x] -A arch -O os -T type -C comp -a addr -e ep -n name
-d data_file[:data_file...] image
```

-a 指定映象在内存中的加载地址，映象下载到内存中时，要按照用 mkimage 制作映象时，这个参数所指定的地址值来下载。

-e 指定映象运行的入口点地址，这个地址就是-a 参数指定的值加上 0x40（因为前面有个 mkimage 添加的 0x40 个字节的头）。

这也就是说：Entry Point = Load Address + 0x40。

Load Address 和 Entry Point 这两个地址都是 mkimage 时指定的。

在此延伸另外两个地址：bootm address 和 kernel 运行地址。

bootm address：bootm 为 uboot 的一个命令，以此从 address 启动 kernel 。

kernel 运行地址：在具体 mach 目录中的 Makefile.boot 中指定，为 kernel 启动后实际运行的物理地址 。

1) bootm 地址和 load address 一样

此种情况下，bootm 不会对 uImage header 后的 zImage 进行 memory move 的动作，而会直接 go 到 entry point 开始执行。因此此时的 entry point 必须设置为 load address + 0x40。如果 kernel boot 过程没有到 uncompressing the kernel，就可能是这里设置不对。

boom address == load address == entry point - 0x40

2) bootm 地址和 load address 不一样(但需要避免出现 memory move 时出现覆盖导致 zImage 被破坏的情况）

此种情况下，bootm 会把 uImage header 后的 zImage 文件 move 到 load address，然后 go 到 entry point 开始执行。这段代码可以在 common/cmd_bootm.c 中 bootm_load_os 函数，如程序清单 10. 1 所示。 由此知道此时的 load address 必须等于 entry point。

boom address != load address == entry point

程序清单 10. 1 bootm 和 load address

```
case IH_COMP_NONE:
    if (load == blob_start || load == image_start)
    {
        printf("   XIP %s ... ", type_name);
        no_overlap = 1;
    }
    else
    {
        printf("   Loading %s ... ", type_name);
        memmove_wd((void *)load, (void *)image_start,
```

```
                image_len, CHUNKSZ);
        }
        *load_end = load + image_len;
        puts("OK\n");
        break;
```

zImage 的头部有地址无关的自解压程序，因此刚开始执行的时候，zImage 所在的内存地址（entry point）不需要同编译 kernel 的地址相同。自解压程序会把 kernel 解压到编译时指定的物理地址，然后开始地址相关代码的执行。在开启 MMU 之前，kernel 都是直接使用物理地址（可参看内核符号映射表 System.map）。

10.2 Load Address 和 Entry Point 的修改

Load Address 和 Entry Point 是在 scripts/makefile.lib 中：

```
318    UIMAGE_LOADADDR  ?= arch_must_set_this
319    UIMAGE_ENTRYADDR ?= $(UIMAGE_LOADADDR)
```

由于 Entry Point = Load Address + 0x40，那么进行如下修改：

```
318    UIMAGE_LOADADDR    ?= arch_must_set_this
319    #UIMAGE_ENTRYADDR ?= $(UIMAGE_LOADADDR)
320    UIMAGE_ENTRYADDR  ?=$(shell echo $(UIMAGE_LOADADDR) |
                            sed -e "s/..$$/40/")
```

sed -e "s/..$$/40/"的意思是，把输出的字符串的最后两个字符删掉，并且用 40 来补充，也可以理解为，把字符串最后两个字符用 40 来替换。

如若之前的：

```
Load Address: 50008000
Entry Point:  50008000
```

那么更改为第 320 行代码之后就应该为：

```
Load Address: 50008000
Entry Point:  50008040
```

编译内核，

```
  Kernel: arch/arm/boot/zImage is ready
  UIMAGE  arch/arm/boot/uImage
Image Name:   Linux-3.6.7
Created:      Wed Dec 12 09:32:08 2012
Image Type:   ARM Linux Kernel Image (uncompressed)
Data Size:    1618312 Bytes = 1580.38 kB = 1.54 MB
Load Address: 50008000
Entry Point:  50008040
  Image arch/arm/boot/uImage is ready
```

让 u-boot 启动 uImage(要更名为 zImage)，

```
NAND read: device 0 offset 0x100000, size 0x500000
 5242880 bytes read: OK
```

```
## Booting kernel from Legacy Image at 50008000 ...
   Image Name:   Linux-3.6.7
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    1618296 Bytes = 1.5 MiB
   Load Address: 50008000
   Entry Point:  50008040
   Verifying Checksum ... OK
   XIP Kernel Image ... OK
OK
Starting kernel ...

Starting kernel ...

Uncompressing Linux... done, booting the kernel.
Booting Linux on physical CPU 0
Linux version 3.6.7 (zhuzhaoqi@zhuzhaoqi-desktop) (gcc version 4.4.1
(Sourcery G++ Lite 2009q3-67) ) #1 Wed Dec 12 11:55:06 CST 2012
CPU: ARMv6-compatible processor [410fb766] revision 6 (ARMv7),
cr=00c5387d
CPU: PIPT / VIPT nonaliasing data cache, VIPT nonaliasing instruction
cache
Machine: OK6410
Memory policy: ECC disabled, Data cache writeback
CPU S3C6410 (id 0x36410101)
S3C24XX Clocks, Copyright 2004 Simtec Electronics
camera: no parent clock specified
S3C64XX: PLL settings, A=533000000, M=533000000, E=24000000
S3C64XX: HCLK2=266500000, HCLK=133250000, PCLK=66625000
mout_apll: source is fout_apll (1), rate is 533000000
mout_epll: source is epll (1), rate is 24000000
mout_mpll: source is mpll (1), rate is 533000000
usb-bus-host: source is clk_48m (0), rate is 48000000
audio-bus: source is mout_epll (0), rate is 24000000
audio-bus: source is mout_epll (0), rate is 24000000
audio-bus: source is mout_epll (0), rate is 24000000
irda-bus: source is mout_epll (0), rate is 24000000
camera: no parent clock specified
CPU: found DTCM0 8k @ 00000000, not enabled
CPU: moved DTCM0 8k to fffe8000, enabled
CPU: found DTCM1 8k @ 00000000, not enabled
CPU: moved DTCM1 8k to fffea000, enabled
CPU: found ITCM0 8k @ 00000000, not enabled
CPU: moved ITCM0 8k to fffe0000, enabled
CPU: found ITCM1 8k @ 00000000, not enabled
```

```
CPU: moved ITCM1 8k to fffe2000, enabled
Built 1 zonelists in Zone order, mobility grouping on.  Total pages:
65024
Kernel command line: root=/dev/mtdblock2 rootfstype=cramfs
console=ttySAC0,115200
PID hash table entries: 1024 (order: 0, 4096 bytes)
Dentry cache hash table entries: 32768 (order: 5, 131072 bytes)
Inode-cache hash table entries: 16384 (order: 4, 65536 bytes)
Memory: 256MB = 256MB total
Memory: 256612k/256612k available, 5532k reserved, 0K highmem
Virtual kernel memory layout:
    vector  : 0xffff0000 - 0xffff1000   (   4 kB)
    DTCM    : 0xfffe8000 - 0xfffec000   (  16 kB)
    ITCM    : 0xfffe0000 - 0xfffe4000   (  16 kB)
    fixmap  : 0xfff00000 - 0xfffe0000   ( 896 kB)
    vmalloc : 0xd0800000 - 0xff000000   ( 744 MB)
    lowmem  : 0xc0000000 - 0xd0000000   ( 256 MB)
    modules : 0xbf000000 - 0xc0000000   (  16 MB)
      .text : 0xc0008000 - 0xc02aa5e8   (2698 kB)
      .init : 0xc02ab000 - 0xc02c6174   ( 109 kB)
      .data : 0xc02c8000 - 0xc02f3000   ( 172 kB)
       .bss : 0xc02f3024 - 0xc0324084   ( 197 kB)
SLUB: Genslabs=13, HWalign=32, Order=0-3, MinObjects=0, CPUs=1,
Nodes=1
NR_IRQS:246
VIC @f6000000: id 0x00041192, vendor 0x41
VIC @f6010000: id 0x00041192, vendor 0x41
sched_clock: 32 bits at 100 Hz, resolution 10000000ns, wraps every
4294967286ms
Console: colour dummy device 80x30
Calibrating delay loop... 353.89 BogoMIPS (lpj=1769472)
pid_max: default: 32768 minimum: 301
Mount-cache hash table entries: 512
CPU: Testing write buffer coherency: ok
Setting up static identity map for 0x50204fa8 - 0x50205004
DMA: preallocated 256 KiB pool for atomic coherent allocations
OK6410: Option string ok6410=0
OK6410: selected LCD display is 480x272
s3c64xx_dma_init: Registering DMA channels
PL080: IRQ 73, at d0846000, channels 0..8
PL080: IRQ 74, at d0848000, channels 8..16
S3C6410: Initialising architecture
bio: create slab <bio-0> at 0
usbcore: registered new interface driver usbfs
```

```
usbcore: registered new interface driver hub
usbcore: registered new device driver usb
ROMFS MTD (C) 2007 Red Hat, Inc.
io scheduler noop registered
io scheduler deadline registered
io scheduler cfq registered (default)
start plist test
end plist test
s3c-fb s3c-fb: window 0: fb
Serial: 8250/16550 driver, 4 ports, IRQ sharing disabled
s3c6400-uart.0: ttySAC0 at MMIO 0x7f005000 (irq = 69) is a S3C6400/10
console [ttySAC0] enabled
s3c6400-uart.1: ttySAC1 at MMIO 0x7f005400 (irq = 70) is a S3C6400/10
s3c6400-uart.2: ttySAC2 at MMIO 0x7f005800 (irq = 71) is a S3C6400/10
s3c6400-uart.3: ttySAC3 at MMIO 0x7f005c00 (irq = 72) is a S3C6400/10
brd: module loaded
loop: module loaded
S3C24XX NAND Driver, (c) 2004 Simtec Electronics
s3c24xx-nand s3c6400-nand: Tacls=4, 30ns Twrph0=8 60ns, Twrph1=6 45ns
s3c24xx-nand s3c6400-nand: System booted from NAND
s3c24xx-nand s3c6400-nand: NAND soft ECC
NAND device: Manufacturer ID: 0xec, Chip ID: 0xd5 (Samsung NAND 2GiB
3,3V 8-bit), page size: 4096, OOB size: 218
No oob scheme defined for oobsize 218
------------[ cut here ]------------
kernel BUG at drivers/mtd/nand/nand_base.c:3278!
Internal error: Oops - BUG: 0 [#1] ARM
Modules linked in:
CPU: 0    Not tainted  (3.6.7 #1)
PC is at nand_scan_tail+0x4c4/0x678
LR is at nand_scan_tail+0x4c4/0x678
pc : [<c019c190>]    lr : [<c019c190>]    psr: 60000013
sp : cf82feb0  ip : c02d933c  fp : c01a23e0
r10: c01a27c0  r9 : c01a27b4  r8 : 00000001
r7 : 00000000  r6 : 00001480  r5 : cf80a000  r4 : cf80a210
r3 : c02d933c  r2 : 00000001  r1 : 00000000  r0 : 00000025
Flags: nZCv  IRQs on  FIQs on  Mode SVC_32  ISA ARM  Segment kernel
Control: 00c5387d  Table: 50004008  DAC: 00000017
Process swapper (pid: 1, stack limit = 0xcf82e268)
Stack: (0xcf82feb0 to 0xcf830000)
fea0:                                cf80a000 cf854300 cfa3fa80
c01a29a0
fec0: 00000000 00000000 c02d7a40 c0323918 c02ef9c8 c02d7a40 c02ef9c8
00000049
```

```
fee0: c02c5ed8 c02bf274 00000000 c0183994 c018397c c01824d4 c02d7a40
c02ef9c8
ff00: c02d7a74 00000000 00000049 c01826e4 c02ef9c8 cf82ff20 c0182658
c0180dd8
ff20: cf803878 cf8239a0 c02ef9c8 c02ef9c8 c02ec488 cfa3f9c0 00000000
c0181620
ff40: c0278edc cf82e000 cf82e000 c02ef9c8 c02f3040 00000000 00000049
c0182ca0
ff60: cf82e000 00000007 c02f3040 00000000 00000049 c0008618 00000000
c05268ae
ff80: c0297e94 c02bf274 00000000 60000013 c027c1e4 00000000 00000006
00000006
ffa0: c02da758 c02c1968 00000007 c02f3040 c02ab1b0 00000049 c02c5ed8
c02c1970
ffc0: 00000000 c02ab310 00000006 00000006 c02ab1b0 00000000 00000000
c02ab230
ffe0: c000fb28 00000013 00000000 00000000 00000000 c000fb28 ffffffff
ffffffff
[<c019c190>] (nand_scan_tail+0x4c4/0x678) from [<c01a29a0>]
(s3c24xx_nand_probe+0x1d4/0x4c8)
[<c01a29a0>] (s3c24xx_nand_probe+0x1d4/0x4c8) from [<c0183994>]
(platform_drv_probe+0x18/0x1c)
[<c0183994>] (platform_drv_probe+0x18/0x1c) from [<c01824d4>]
(driver_probe_device+0x78/0x1fc)
[<c01824d4>] (driver_probe_device+0x78/0x1fc) from [<c01826e4>]
(__driver_attach+0x8c/0x90)
[<c01826e4>] (__driver_attach+0x8c/0x90) from [<c0180dd8>]
(bus_for_each_dev+0x54/0x80)
[<c0180dd8>] (bus_for_each_dev+0x54/0x80) from [<c0181620>]
(bus_add_driver+0x1cc/0x288)
[<c0181620>] (bus_add_driver+0x1cc/0x288) from [<c0182ca0>]
(driver_register+0x78/0x194)
[<c0182ca0>] (driver_register+0x78/0x194) from [<c0008618>]
(do_one_initcall+0x34/0x178)
[<c0008618>] (do_one_initcall+0x34/0x178) from [<c02ab310>]
(kernel_init+0xe0/0x1ac)
[<c02ab310>] (kernel_init+0xe0/0x1ac) from [<c000fb28>]
(kernel_thread_exit+0x0/0x8)
Code: e3510008 0a00002a e59f018c eb0199db (e7f001f2)
---[ end trace 823dd878ddb6298b ]---
Kernel panic - not syncing: Attempted to kill init! exitcode=0x0000000b
```

E-mail : jxlgzzq@163.com

第 11 章 Linux3.6.7 移植之内核分区

11.1 内核分区

进入 arch/arm/mach-s3c64xx，打开 mach-ok6410.c 这个文件。

```
static struct mtd_partition ok6410_nand_part[] = {
	[0] = {
		.name   = "uboot",
		.size   = SZ_1M,
		.offset = 0,
	},
	[1] = {
		.name   = "kernel",
		.size   = SZ_2M,
		.offset = SZ_1M,
	},
	[2] = {
		.name   = "rootfs",
		.size   = MTDPART_SIZ_FULL,
		.offset = SZ_1M + SZ_2M,
	},
};
```

修改为：

```
static struct mtd_partition ok6410_nand_part[] = {
	[0] = {
		.name      = "Bootloader",
		.offset    = 0,
		.size      = (2 * SZ_1M),
		.mask_flags = MTD_CAP_NANDFLASH,
	},
	[1] = {
		.name      = "Kernel",
		.offset    = (2 * SZ_1M),
		.size      = (5 * SZ_1M) ,
		.mask_flags = MTD_CAP_NANDFLASH,
	},
	[2] = {
		.name   = "File System",
		.offset = (7 * SZ_1M),
		.size   = (200 * SZ_1M) ,
	},
	[3] = {
		.name   = "User",
		.offset = MTDPART_OFS_APPEND,
```

```
                .size   = MTDPART_SIZ_FULL,
        },
};
```

编译内核。

第 12 章 Linux3.6.7 移植之 NandFlash 驱动

12.1 NandFlash 驱动

下载 s3c_nand.c 放入 drivers/mtd/nand/中。

修改 drivers/mtd/nand/下面的 Makefile，添加：

```
obj-$(CONFIG_MTD_NAND_S3C)      += s3c_nand.o
```

在 drivers/mtd/nand/下面的 Kconfig 中，添加：

```
config MTD_NAND_S3C
   tristate "NAND Flash support for S3C SoC"
   depends on (ARCH_S3C64XX || ARCH_S5P64XX || ARCH_S5PC1XX)&&
MTD_NAND
   help
     This enables the NAND flash controller on the S3C.
     No board specfic support is done by this driver, eachboard
     must advertise a platform_device for the driver toattach.
config MTD_NAND_S3C_DEBUG
   bool "S3C NAND driver debug"
   depends on MTD_NAND_S3C
   help
     Enable debugging of the S3C NAND driver
config MTD_NAND_S3C_HWECC
   bool "S3C NAND Hardware ECC"
   depends on MTD_NAND_S3C
   help
     Enable the use of the S3C's internal ECC generatorwhen
     using NAND. Early versions of the chip have hadproblems with
     incorrect ECC generation, and if using these, the defaultof
     software ECC is preferable.
     If you lay down a device with the hardware ECC, thenyou will
     currently not be able to switch to software, as thereis no
     implementation for ECC method used by the S3C
```

进入 arch/arm/plat-samsung/include/plat/regs_nand.h 加入寄存器定义。

```
/* for s3c_nand.c */
#define S3C_NFCONF S3C2410_NFREG(0x00)
#define S3C_NFCONT S3C2410_NFREG(0x04)
#define S3C_NFCMMD S3C2410_NFREG(0x08)
#define S3C_NFADDR S3C2410_NFREG(0x0c)
#define S3C_NFDATA8 S3C2410_NFREG(0x10)
#define S3C_NFDATA S3C2410_NFREG(0x10)
#define S3C_NFMECCDATA0 S3C2410_NFREG(0x14)
#define S3C_NFMECCDATA1 S3C2410_NFREG(0x18)
```

```
#define S3C_NFSECCDATA S3C2410_NFREG(0x1c)
#define S3C_NFSBLK S3C2410_NFREG(0x20)
#define S3C_NFEBLK S3C2410_NFREG(0x24)
#define S3C_NFSTAT S3C2410_NFREG(0x28)
#define S3C_NFMECCERR0 S3C2410_NFREG(0x2c)
#define S3C_NFMECCERR1 S3C2410_NFREG(0x30)
#define S3C_NFMECC0 S3C2410_NFREG(0x34)
#define S3C_NFMECC1 S3C2410_NFREG(0x38)
#define S3C_NFSECC S3C2410_NFREG(0x3c)
#define S3C_NFMLCBITPT S3C2410_NFREG(0x40)
#define S3C_NF8ECCERR0 S3C2410_NFREG(0x44)
#define S3C_NF8ECCERR1 S3C2410_NFREG(0x48)
#define S3C_NF8ECCERR2 S3C2410_NFREG(0x4c)
#define S3C_NFM8ECC0 S3C2410_NFREG(0x50)
#define S3C_NFM8ECC1 S3C2410_NFREG(0x54)
#define S3C_NFM8ECC2 S3C2410_NFREG(0x58)
#define S3C_NFM8ECC3 S3C2410_NFREG(0x5c)
#define S3C_NFMLC8BITPT0 S3C2410_NFREG(0x60)
#define S3C_NFMLC8BITPT1 S3C2410_NFREG(0x64)

#define S3C_NFCONF_NANDBOOT  (1<<31)
#define S3C_NFCONF_ECCCLKCON  (1<<30)
#define S3C_NFCONF_ECC_MLC  (1<<24)
#define S3C_NFCONF_ECC_1BIT  (0<<23)
#define S3C_NFCONF_ECC_4BIT  (2<<23)
#define S3C_NFCONF_ECC_8BIT  (1<<23)
#define S3C_NFCONF_TACLS(x)  ((x)<<12)
#define S3C_NFCONF_TWRPH0(x)  ((x)<<8)
#define S3C_NFCONF_TWRPH1(x)  ((x)<<4)
#define S3C_NFCONF_ADVFLASH  (1<<3)
#define S3C_NFCONF_PAGESIZE  (1<<2)
#define S3C_NFCONF_ADDRCYCLE  (1<<1)
#define S3C_NFCONF_BUSWIDTH  (1<<0)

#define S3C_NFCONT_ECC_ENC  (1<<18)
#define S3C_NFCONT_LOCKTGHT  (1<<17)
#define S3C_NFCONT_LOCKSOFT  (1<<16)
#define S3C_NFCONT_8BITSTOP  (1<<11)
#define S3C_NFCONT_MECCLOCK  (1<<7)
#define S3C_NFCONT_SECCLOCK  (1<<6)
#define S3C_NFCONT_INITMECC  (1<<5)
#define S3C_NFCONT_INITSECC  (1<<4)
#define S3C_NFCONT_nFCE1 (1<<2)
#define S3C_NFCONT_nFCE0 (1<<1)
```

```
#define S3C_NFCONT_INITECC  (S3C_NFCONT_INITSECC |
S3C_NFCONT_INITMECC)

#define S3C_NFSTAT_ECCENCDONE  (1<<7)
#define S3C_NFSTAT_ECCDECDONE  (1<<6)
#define S3C_NFSTAT_BUSY (1<<0)

#define S3C_NFECCERR0_ECCBUSY  (1<<31)
```

进入\drivers\mtd\nand\nand_base.c，添加：

```
static struct nand_ecclayout nand_oob_218_128Bit = {
    .eccbytes = 104,
    .eccpos = {
        24,25,26,27,28,29,30,31,32,33,
        34,35,36,37,38,39,40,41,42,43,
        44,45,46,47,48,49,50,51,52,53,
        54,55,56,57,58,59,60,61,62,63,
        64,65,66,67,68,69,70,71,72,73,
        74,75,76,77,78,79,80,81,82,83,
        84,85,86,87,88,89,90,91,92,93,
        94,95,96,97,98,99,100,101,102,103,
        104,105,106,107,108,109,110,111,112,113,
        114,115,116,117,118,119,120,121,122,123,
        124,125,126,127},
    .oobfree =
    {
        {
            .offset = 2,
            .length = 22
        }
    }
};
```

在 int nand_scan_tail(struct mtd_info *mtd)函数中添加：

```
        case 218:
            chip->ecc.layout = &nand_oob_218_128Bit;
            break;
```

在 static void __init ok6410_machine_init(void)函数中添加如下代码：

```
/*add by zzq at 2012-5-6*/

    s3c_device_nand.name = "s3c6410-nand";

/*end  */
```

```
    s3c_nand_set_platdata(&ok6410_nand_info);
```

make menuconfig：

找到 Device Drivers --> Memory Technology Device (MTD) support --->NAND Device Support ---> 取消 NAND Flash support for Samsung S3C SoCs ，选择 NAND Flash support for S3C SoC。

编译内核，启动，如：

```
S3C NAND Driver, (c) 2008 Samsung Electronics
dev_id == 0xd5 select s3c_nand_oob_mlc
****Nandflash:ChipType= MLC
ChipName=samsung-K9GAG08U0D***********
S3C NAND Driver is using hardware ECC.
NAND device: Manufacturer ID: 0xec, Chip ID: 0xd5 (Samsung NAND 2GiB
3,3V 8-bit), page size: 4096, OOB size: 218
Driver must set ecc.strength when using hardware ECC
------------[ cut here ]------------
kernel BUG at drivers/mtd/nand/nand_base.c:3382!
Internal error: Oops - BUG: 0 [#1] ARM
```

这说明在 nand_base.c 出现了错误。

```
kernel BUG at drivers/mtd/nand/nand_base.c:3382!
```

找到这个地方：

```
  if (!chip->ecc.strength)
  {
      pr_warn("Driver must set ecc.strength when using hardware
ECC\n");
      BUG();
  }
  break;
```

也就是说进入了 BUG()，把 BUG()注释掉看看。

编译启动：

```
NAND read: device 0 offset 0x100000, size 0x500000
 5242880 bytes read: OK
## Booting kernel from Legacy Image at 50008000 ...
   Image Name:   Linux-3.6.7
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    1617232 Bytes = 1.5 MiB
   Load Address: 50008000
   Entry Point:  50008040
   Verifying Checksum ... OK
```

```
   XIP Kernel Image ... OK
OK
Starting kernel ...

Starting kernel ...

Uncompressing Linux... done, booting the kernel.
Booting Linux on physical CPU 0
Linux version 3.6.7 (zhuzhaoqi@zhuzhaoqi-desktop) (gcc version 4.4.1
(Sourcery G++ Lite 2009q3-67) ) #1 Wed Dec 12 17:25:26 CST 2012
CPU: ARMv6-compatible processor [410fb766] revision 6 (ARMv7),
cr=00c5387d
CPU: PIPT / VIPT nonaliasing data cache, VIPT nonaliasing instruction
cache
Machine: OK6410
Memory policy: ECC disabled, Data cache writeback
CPU S3C6410 (id 0x36410101)
S3C24XX Clocks, Copyright 2004 Simtec Electronics
camera: no parent clock specified
S3C64XX: PLL settings, A=533000000, M=533000000, E=24000000
S3C64XX: HCLK2=266500000, HCLK=133250000, PCLK=66625000
mout_apll: source is fout_apll (1), rate is 533000000
mout_epll: source is epll (1), rate is 24000000
mout_mpll: source is mpll (1), rate is 533000000
usb-bus-host: source is clk_48m (0), rate is 48000000
audio-bus: source is mout_epll (0), rate is 24000000
audio-bus: source is mout_epll (0), rate is 24000000
audio-bus: source is mout_epll (0), rate is 24000000
irda-bus: source is mout_epll (0), rate is 24000000
camera: no parent clock specified
CPU: found DTCM0 8k @ 00000000, not enabled
CPU: moved DTCM0 8k to fffe8000, enabled
CPU: found DTCM1 8k @ 00000000, not enabled
CPU: moved DTCM1 8k to fffea000, enabled
CPU: found ITCM0 8k @ 00000000, not enabled
CPU: moved ITCM0 8k to fffe0000, enabled
CPU: found ITCM1 8k @ 00000000, not enabled
CPU: moved ITCM1 8k to fffe2000, enabled
Built 1 zonelists in Zone order, mobility grouping on.  Total pages:
65024
Kernel command line: root=/dev/mtdblock2 rootfstype=cramfs
console=ttySAC0,115200
PID hash table entries: 1024 (order: 0, 4096 bytes)
Dentry cache hash table entries: 32768 (order: 5, 131072 bytes)
```

```
Inode-cache hash table entries: 16384 (order: 4, 65536 bytes)
Memory: 256MB = 256MB total
Memory: 256608k/256608k available, 5536k reserved, 0K highmem
Virtual kernel memory layout:
    vector  : 0xffff0000 - 0xffff1000   (   4 kB)
    DTCM    : 0xfffe8000 - 0xfffec000   (  16 kB)
    ITCM    : 0xfffe0000 - 0xfffe4000   (  16 kB)
    fixmap  : 0xfff00000 - 0xfffe0000   ( 896 kB)
    vmalloc : 0xd0800000 - 0xff000000   ( 744 MB)
    lowmem  : 0xc0000000 - 0xd0000000   ( 256 MB)
    modules : 0xbf000000 - 0xc0000000   (  16 MB)
      .text : 0xc0008000 - 0xc02aa5e0   (2698 kB)
      .init : 0xc02ab000 - 0xc02c61b4   ( 109 kB)
      .data : 0xc02c8000 - 0xc02f3080   ( 173 kB)
       .bss : 0xc02f4024 - 0xc03250e4   ( 197 kB)
SLUB: Genslabs=13, HWalign=32, Order=0-3, MinObjects=0, CPUs=1,
Nodes=1
NR_IRQS:246
VIC @f6000000: id 0x00041192, vendor 0x41
VIC @f6010000: id 0x00041192, vendor 0x41
sched_clock: 32 bits at 100 Hz, resolution 10000000ns, wraps every
4294967286ms
Console: colour dummy device 80x30
Calibrating delay loop... 353.89 BogoMIPS (lpj=1769472)
pid_max: default: 32768 minimum: 301
Mount-cache hash table entries: 512
CPU: Testing write buffer coherency: ok
Setting up static identity map for 0x502047e8 - 0x50204844
DMA: preallocated 256 KiB pool for atomic coherent allocations
OK6410: Option string ok6410=0
OK6410: selected LCD display is 480x272
s3c64xx_dma_init: Registering DMA channels
PL080: IRQ 73, at d0846000, channels 0..8
PL080: IRQ 74, at d0848000, channels 8..16
S3C6410: Initialising architecture
bio: create slab <bio-0> at 0
usbcore: registered new interface driver usbfs
usbcore: registered new interface driver hub
usbcore: registered new device driver usb
ROMFS MTD (C) 2007 Red Hat, Inc.
io scheduler noop registered
io scheduler deadline registered
io scheduler cfq registered (default)
start plist test
```

```
end plist test
s3c-fb s3c-fb: window 0: fb
Serial: 8250/16550 driver, 4 ports, IRQ sharing disabled
s3c6400-uart.0: ttySAC0 at MMIO 0x7f005000 (irq = 69) is a S3C6400/10
console [ttySAC0] enabled
s3c6400-uart.1: ttySAC1 at MMIO 0x7f005400 (irq = 70) is a S3C6400/10
s3c6400-uart.2: ttySAC2 at MMIO 0x7f005800 (irq = 71) is a S3C6400/10
s3c6400-uart.3: ttySAC3 at MMIO 0x7f005c00 (irq = 72) is a S3C6400/10
brd: module loaded
loop: module loaded
S3C NAND Driver, (c) 2008 Samsung Electronics
S3C NAND Driver is using software ECC.
NAND device: Manufacturer ID: 0xec, Chip ID: 0xd5 (Samsung NAND 2GiB
3,3V 8-bit), page size: 4096, OOB size: 218
Creating 4 MTD partitions on "NAND 2GiB 3,3V 8-bit":
0x000000000000-0x000000200000 : "Bootloader"
0x000000200000-0x000000700000 : "Kernel"
0x000000700000-0x00000cf00000 : "File System"
0x00000cf00000-0x000080000000 : "User"
ohci_hcd: USB 1.1 'Open' Host Controller (OHCI) Driver
s3c2410-ohci s3c2410-ohci: S3C24XX OHCI
s3c2410-ohci s3c2410-ohci: new USB bus registered, assigned bus number
1
s3c2410-ohci s3c2410-ohci: irq 79, io mem 0x74300000
s3c2410-ohci s3c2410-ohci: init err (00000000 0000)
s3c2410-ohci s3c2410-ohci: can't start s3c24xx
s3c2410-ohci s3c2410-ohci: startup error -75
s3c2410-ohci s3c2410-ohci: USB bus 1 deregistered
s3c2410-ohci: probe of s3c2410-ohci failed with error -75
mousedev: PS/2 mouse device common for all mice
i2c /dev entries driver
sdhci: Secure Digital Host Controller Interface driver
sdhci: Copyright(c) Pierre Ossman
s3c-sdhci s3c-sdhci.0: clock source 0: mmc_busclk.0 (133250000 Hz)
s3c-sdhci s3c-sdhci.0: clock source 2: mmc_busclk.2 (24000000 Hz)
mmc0: SDHCI controller on samsung-hsmmc [s3c-sdhci.0] using ADMA
s3c-sdhci s3c-sdhci.1: clock source 0: mmc_busclk.0 (133250000 Hz)
s3c-sdhci s3c-sdhci.1: clock source 2: mmc_busclk.2 (24000000 Hz)
mmc0: mmc_rescan_try_freq: trying to init card at 400000 Hz
mmc0: mmc_rescan_try_freq: trying to init card at 300000 Hz
mmc1: SDHCI controller on samsung-hsmmc [s3c-sdhci.1] using ADMA
mmc0: mmc_rescan_try_freq: trying to init card at 200000 Hz
usbcore: registered new interface driver usbhid
usbhid: USB HID core driver
```

```
VFP support v0.3: implementor 41 architecture 1 part 20 variant b rev
5
drivers/rtc/hctosys.c: unable to open rtc device (rtc0)
VFS: Cannot open root device "mtdblock2" or unknown-block(0,0): error
-6
Please append a correct "root=" boot option; here are the available
partitions:
Kernel panic - not syncing: VFS: Unable to mount root fs on
unknown-block(0,0)
[<c0014d28>] (unwind_backtrace+0x0/0xf4) from [<c0201f9c>]
(panic+0x8c/0x1dc)
[<c0201f9c>] (panic+0x8c/0x1dc) from [<c02abd00>]
(mount_block_root+0x1e8/0x2ac)
[<c02abd00>] (mount_block_root+0x1e8/0x2ac) from [<c02abf88>]
(prepare_namespace+0x160/0x1b8)
[<c02abf88>] (prepare_namespace+0x160/0x1b8) from [<c02ab394>]
(kernel_init+0x164/0x1ac)
[<c02ab394>] (kernel_init+0x164/0x1ac) from [<c000fb28>]
(kernel_thread_exit+0x0/0x8)
```

从输出的信息可以看出，NandFlash 驱动是没有问题。

第 13 章 Linux3.6.7 移植之根文件系统

13.1 YAFFS2 移植到 Linux3.6.7

下载 YAFFS2 源码，如：

```
...$ git clone git://www.aleph1.co.uk/yaffs2
```

如果之前没有安装 git-core，则会提示先得安装 git-core。则：

```
...$ sudo apt-get install git-core
```

下载好 yaffs2，先给 Linux3.6.7 打好补丁：

```
...$ ./patch-ker.sh c m /home/zhuzhaoqi/Linux/linux-3.6.7
```

完成之后，进入 linux-3.6.7 的 fs 文件夹下多了一个 yaffs2 的文件夹。这说明补丁成功。

```
make menuconfig
```

需要先选择 Device Drivers 进入，如图 13. 1 所示。

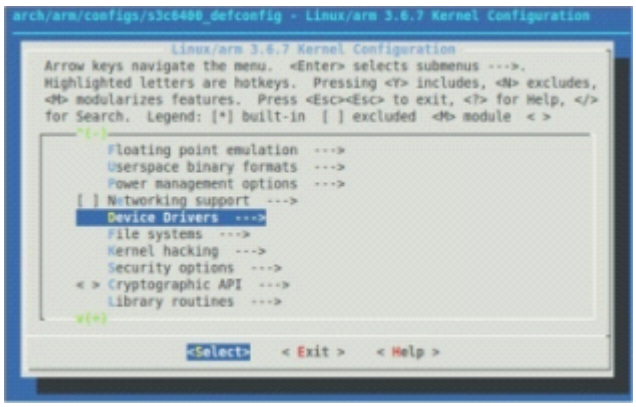


图 13. 1 yaffs2 配置步骤一

选择 MTD support 进入，如图 13. 2 所示。

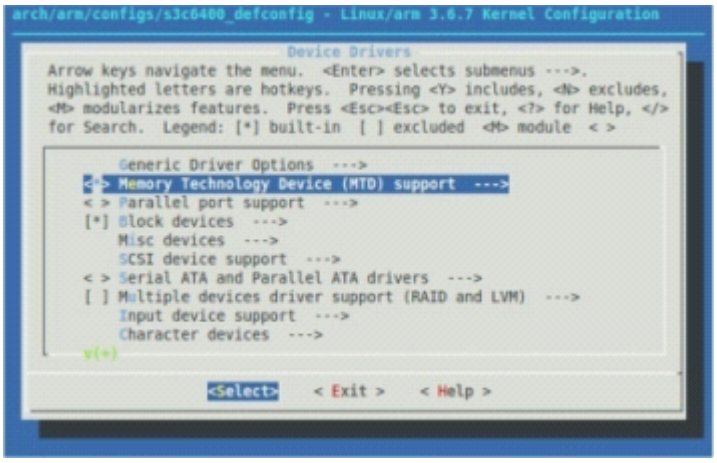


图 13. 2 yaffs2 配置步骤二

选择 Caching block device access to MTD devices，如图 13. 3 所示。

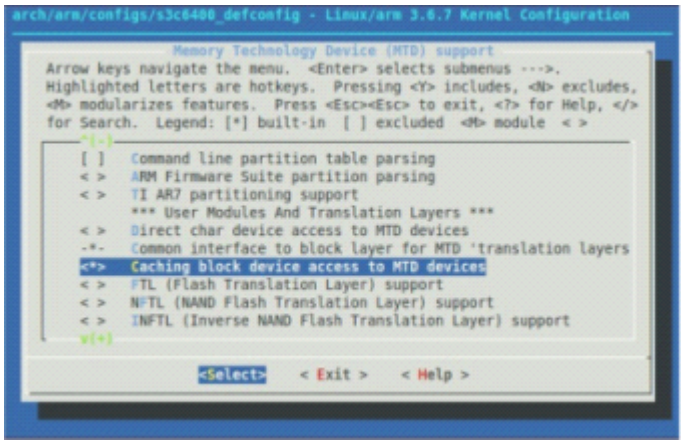


图 13.3 yaffs2 配置步骤三

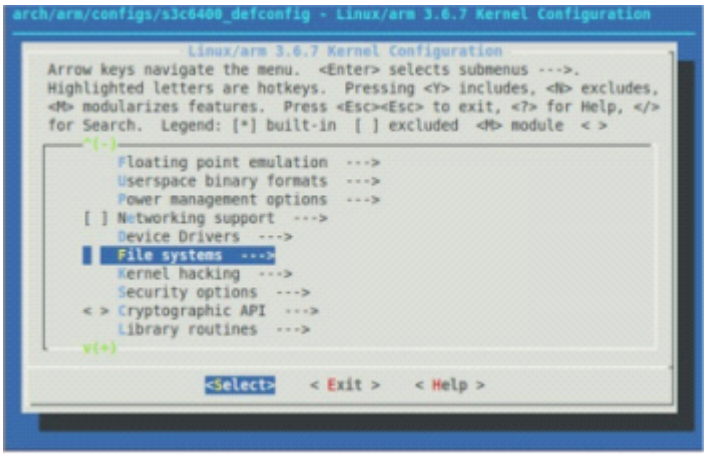


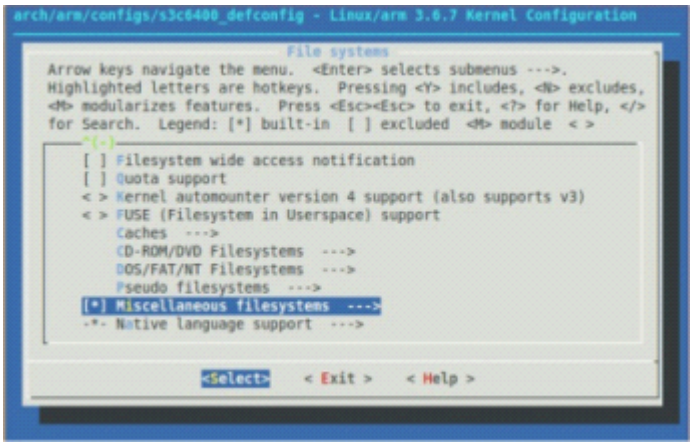
图 13.4 yaffs2 配置步骤四

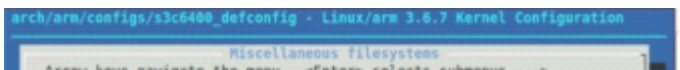
退回到和 Device Drivers 一个目录下，进入 File Systems，如图 13.4 所示。

选择 Miscellaneous filesystem 进入，如图 13.5 所示。

选择 yaffs2 file system support，如图 13.6 所示。

配置完成之后 make uImage，出现错误：





```
fs/yaffs2/yaffs_vfs.c:438: warning: initialization from incompatible
pointer type
fs/yaffs2/yaffs_vfs.c:439: warning: initialization from incompatible
pointer type
fs/yaffs2/yaffs_vfs.c:443: warning: initialization from incompatible
pointer type
fs/yaffs2/yaffs_vfs.c:445: warning: initialization from incompatible
pointer type
fs/yaffs2/yaffs_vfs.c:478: error: unknown field 'write_super'
specified in initializer
fs/yaffs2/yaffs_vfs.c:478: warning: initialization from incompatible
pointer type
fs/yaffs2/yaffs_vfs.c: In function 'yaffs_evict_inode':
fs/yaffs2/yaffs_vfs.c:873: error: implicit declaration of function
'end_writeback'
fs/yaffs2/yaffs_vfs.c: In function 'yaffs_do_sync_fs':
fs/yaffs2/yaffs_vfs.c:2203: error: 'struct super_block' has no member
named 's_dirt'
fs/yaffs2/yaffs_vfs.c:2214: error: 'struct super_block' has no member
named 's_dirt'
fs/yaffs2/yaffs_vfs.c:2216: error: 'struct super_block' has no member
named 's_dirt'
fs/yaffs2/yaffs_vfs.c: In function 'yaffs_put_super':
fs/yaffs2/yaffs_vfs.c:2506: error: 'struct mtd_info' has no member
named 'sync'
fs/yaffs2/yaffs_vfs.c:2507: error: 'struct mtd_info' has no member
named 'sync'
fs/yaffs2/yaffs_vfs.c: In function 'yaffs_touch_super':
fs/yaffs2/yaffs_vfs.c:2523: error: 'struct super_block' has no member
named 's_dirt'
fs/yaffs2/yaffs_vfs.c: In function 'yaffs_internal_read_super':
fs/yaffs2/yaffs_vfs.c:2699: error: 'struct mtd_info' has no member
named 'erase'
fs/yaffs2/yaffs_vfs.c:2700: error: 'struct mtd_info' has no member
named 'read'
fs/yaffs2/yaffs_vfs.c:2701: error: 'struct mtd_info' has no member
named 'write'
fs/yaffs2/yaffs_vfs.c:2702: error: 'struct mtd_info' has no member
named 'read_oob'
fs/yaffs2/yaffs_vfs.c:2703: error: 'struct mtd_info' has no member
named 'write_oob'
fs/yaffs2/yaffs_vfs.c:2704: error: 'struct mtd_info' has no member
named 'block_isbad'
```

```
fs/yaffs2/yaffs_vfs.c:2705: error: 'struct mtd_info' has no member
named 'block_markbad'
fs/yaffs2/yaffs_vfs.c:2729: error: 'struct mtd_info' has no member
named 'erase'
fs/yaffs2/yaffs_vfs.c:2730: error: 'struct mtd_info' has no member
named 'block_isbad'
fs/yaffs2/yaffs_vfs.c:2731: error: 'struct mtd_info' has no member
named 'block_markbad'
fs/yaffs2/yaffs_vfs.c:2731: error: 'struct mtd_info' has no member
named 'read'
fs/yaffs2/yaffs_vfs.c:2731: error: 'struct mtd_info' has no member
named 'write'
fs/yaffs2/yaffs_vfs.c:2733: error: 'struct mtd_info' has no member
named 'read_oob'
fs/yaffs2/yaffs_vfs.c:2733: error: 'struct mtd_info' has no member
named 'write_oob'
fs/yaffs2/yaffs_vfs.c:2754: error: 'struct mtd_info' has no member
named 'erase'
fs/yaffs2/yaffs_vfs.c:2754: error: 'struct mtd_info' has no member
named 'read'
fs/yaffs2/yaffs_vfs.c:2754: error: 'struct mtd_info' has no member
named 'write'
fs/yaffs2/yaffs_vfs.c:2756: error: 'struct mtd_info' has no member
named 'read_oob'
fs/yaffs2/yaffs_vfs.c:2756: error: 'struct mtd_info' has no member
named 'write_oob'
fs/yaffs2/yaffs_vfs.c:2946: error: implicit declaration of function
'd_alloc_root'
fs/yaffs2/yaffs_vfs.c:2946: warning: assignment makes pointer from
integer without a cast
fs/yaffs2/yaffs_vfs.c:2955: error: 'struct super_block' has no member
named 's_dirt'
make[2]: *** [fs/yaffs2/yaffs_vfs.o] 错误 1
make[1]: *** [fs/yaffs2] 错误 2
```

出现了错误，一个一个去寻找。

对于：

```
fs/yaffs2/yaffs_vfs.c:478: error: unknown field 'write_super'
specified in initializer
```

进入 include/linux，打开 fs.h 文件。

```
1857    struct super_operations {
            ......
}
```

在这个结构体中添加：

```
//--->zzq
      void (*write_super)(struct super_block *);
//<---zzq
```

对于：

```
fs/yaffs2/yaffs_vfs.c:2523: error: 'struct super_block' has no member
named 's_dirt'
```

进入 include/linux，打开 fs.h 文件。

```
1490   struct super_block {
          ......
}
```

在这个结构体中添加：

```
//--->zzq
      unsigned char         s_dirt;
//<---zzq
```

对于：

```
fs/yaffs2/yaffs_vfs.c:873: error: implicit declaration of function
'end_writeback'
```

进入 fs/yaffs2，打开文件 yaffs2_vfs.c 找到：

```
static void yaffs_evict_inode(struct inode *inode)
```

里面的 end_writeback(inode)注释掉：

```
#if 0
//--->zzq
      end_writeback(inode);
//<---zzq
#endif
```

修改为：

```
clear_inode(inode);
```

对于：

```
fs/yaffs2/yaffs_vfs.c:2946: error: implicit declaration of function
'd_alloc_root'
```

进入 fs/yaffs2，打开文件 yaffs2_vfs.c，找到 d_alloc_root(inode)函数，注释掉：

```
#if 0
//--->zzq
      root = d_alloc_root(inode);
//<---zzq
#endif
```

修改为：

```
    root = d_make_root(inode);
```

到这里就剩下：

```
fs/yaffs2/yaffs_vfs.c:2756: error: 'struct mtd_info' has no member
named  '......'
```

首先应该找到 struct mtd_info 这个结构体。这个结构体的定义位于 /include/linux/mtd/mtd.h 文件中。

里面出现报错的相关定义：

```
int (*_erase) (struct mtd_info *mtd, struct erase_info *instr);
int (*_point) (struct mtd_info *mtd, loff_t from, size_t len,
               size_t *retlen, void **virt, resource_size_t *phys);
int (*_unpoint) (struct mtd_info *mtd, loff_t from, size_t len);
unsigned long (*_get_unmapped_area) (struct mtd_info *mtd,
                                     unsigned long len,
                                     unsigned long offset,
                                     unsigned long flags);
int (*_read) (struct mtd_info *mtd, loff_t from, size_t len,
              size_t *retlen, u_char *buf);
int (*_write) (struct mtd_info *mtd, loff_t to, size_t len,
               size_t *retlen, const u_char *buf);
int (*_panic_write) (struct mtd_info *mtd, loff_t to, size_t len,
                     size_t *retlen, const u_char *buf);
int (*_read_oob) (struct mtd_info *mtd, loff_t from,
                  struct mtd_oob_ops *ops);
int (*_write_oob) (struct mtd_info *mtd, loff_t to,
                  struct mtd_oob_ops *ops);
int (*_get_fact_prot_info) (struct mtd_info *mtd,
                            struct otp_info *buf,
                              size_t len);
int (*_read_fact_prot_reg) (struct mtd_info *mtd, loff_t from,
                            size_t len, size_t *retlen, u_char *buf);
int (*_get_user_prot_info) (struct mtd_info *mtd,
                             struct otp_info *buf,
                              size_t len);
int (*_read_user_prot_reg) (struct mtd_info *mtd, loff_t from,
                            size_t len, size_t *retlen, u_char *buf);
int (*_write_user_prot_reg) (struct mtd_info *mtd, loff_t to,
                            size_t len, size_t *retlen, u_char *buf);
int (*_lock_user_prot_reg) (struct mtd_info *mtd, loff_t from,
                              size_t len);
int (*_writev) (struct mtd_info *mtd, const struct kvec *vecs,
                  unsigned long count, loff_t to, size_t *retlen);
```

```
void (*_sync) (struct mtd_info *mtd);
int (*_lock) (struct mtd_info *mtd, loff_t ofs, uint64_t len);
int (*_unlock) (struct mtd_info *mtd, loff_t ofs, uint64_t len);
int (*_is_locked) (struct mtd_info *mtd, loff_t ofs, uint64_t len);
int (*_block_isbad) (struct mtd_info *mtd, loff_t ofs);
int (*_block_markbad) (struct mtd_info *mtd, loff_t ofs);
int (*_suspend) (struct mtd_info *mtd);
void (*_resume) (struct mtd_info *mtd);
/*
* If the driver is something smart, like UBI, it may need to maintain
* its own reference counting. The below functions are only for driver.
*/
int (*_get_device) (struct mtd_info *mtd);
void (*_put_device) (struct mtd_info *mtd);
```

根据报错将一个个改正过来，再次编译。

```
fs/yaffs2/yaffs_mtdif.c: In function 'nandmtd_erase_block':
fs/yaffs2/yaffs_mtdif.c:53: error: 'struct mtd_info' has no member
named 'erase'
fs/yaffs2/yaffs_mtdif.c: In function 'yaffs_mtd_write':
fs/yaffs2/yaffs_mtdif.c:79: error: 'struct mtd_info' has no member
named 'write_oob'
fs/yaffs2/yaffs_mtdif.c: In function 'yaffs_mtd_read':
fs/yaffs2/yaffs_mtdif.c:115: error: 'struct mtd_info' has no member
named 'read_oob'
fs/yaffs2/yaffs_mtdif.c: In function 'yaffs_mtd_erase':
fs/yaffs2/yaffs_mtdif.c:168: error: 'struct mtd_info' has no member
named 'erase'
fs/yaffs2/yaffs_mtdif.c: In function 'yaffs_mtd_mark_bad':
fs/yaffs2/yaffs_mtdif.c:184: error: 'struct mtd_info' has no member
named 'block_markbad'
fs/yaffs2/yaffs_mtdif.c: In function 'yaffs_mtd_check_bad':
fs/yaffs2/yaffs_mtdif.c:196: error: 'struct mtd_info' has no member
named 'block_isbad'
make[2]: *** [fs/yaffs2/yaffs_mtdif.o] 错误 1
make[1]: *** [fs/yaffs2] 错误 2
```

错误和上次一样，接着进入 fs/yaffs2/yaffs_mtdif.c 进行逐一修改。

修改完成之后编译：

```
Image Name:   Linux-3.6.7
Created:      Thu Dec 13 15:58:44 2012
Image Type:   ARM Linux Kernel Image (uncompressed)
Data Size:    1675112 Bytes = 1635.85 kB = 1.60 MB
```

```
Load Address: 50008000
Entry Point:  50008040
  Image arch/arm/boot/uImage is ready
```

编译通过，启动之后信息：

```
......
......
S3C NAND Driver, (c) 2008 Samsung Electronics
S3C NAND Driver is using software ECC.
NAND device: Manufacturer ID: 0xec, Chip ID: 0xd5 (Samsung NAND 2GiB
3,3V 8-bit), page size: 4096, OOB size: 218
Creating 4 MTD partitions on "NAND 2GiB 3,3V 8-bit":
0x000000000000-0x000000200000 : "Bootloader"
0x000000200000-0x000000700000 : "Kernel"
0x000000700000-0x00000cf00000 : "File System"
0x00000cf00000-0x000080000000 : "User"
ohci_hcd: USB 1.1 'Open' Host Controller (OHCI) Driver
s3c2410-ohci s3c2410-ohci: S3C24XX OHCI
s3c2410-ohci s3c2410-ohci: new USB bus registered, assigned bus number
1
s3c2410-ohci s3c2410-ohci: irq 79, io mem 0x74300000
s3c2410-ohci s3c2410-ohci: init err (00000000 0000)
s3c2410-ohci s3c2410-ohci: can't start s3c24xx
s3c2410-ohci s3c2410-ohci: startup error -75
s3c2410-ohci s3c2410-ohci: USB bus 1 deregistered
s3c2410-ohci: probe of s3c2410-ohci failed with error -75
mousedev: PS/2 mouse device common for all mice
i2c /dev entries driver
sdhci: Secure Digital Host Controller Interface driver
sdhci: Copyright(c) Pierre Ossman
s3c-sdhci s3c-sdhci.0: clock source 0: mmc_busclk.0 (133250000 Hz)
s3c-sdhci s3c-sdhci.0: clock source 2: mmc_busclk.2 (24000000 Hz)
mmc0: SDHCI controller on samsung-hsmmc [s3c-sdhci.0] using ADMA
s3c-sdhci s3c-sdhci.1: clock source 0: mmc_busclk.0 (133250000 Hz)
s3c-sdhci s3c-sdhci.1: clock source 2: mmc_busclk.2 (24000000 Hz)
mmc0: mmc_rescan_try_freq: trying to init card at 400000 Hz
mmc0: mmc_rescan_try_freq: trying to init card at 300000 Hz
mmc1: SDHCI controller on samsung-hsmmc [s3c-sdhci.1] using ADMA
mmc0: mmc_rescan_try_freq: trying to init card at 200000 Hz
usbcore: registered new interface driver usbhid
usbhid: USB HID core driver
VFP support v0.3: implementor 41 architecture 1 part 20 variant b rev
5
drivers/rtc/hctosys.c: unable to open rtc device (rtc0)
```

```
List of all partitions:
1f00          2048 mtdblock0  (driver?)
1f01          5120 mtdblock1  (driver?)
1f02        204800 mtdblock2  (driver?)
1f03       1885184 mtdblock3  (driver?)
No filesystem could mount root, tried:  cramfs
Kernel panic - not syncing: VFS: Unable to mount root fs on
unknown-block(31,2)
[<c0014d28>] (unwind_backtrace+0x0/0xf4) from [<c0215ce0>]
(panic+0x8c/0x1dc)
[<c0215ce0>] (panic+0x8c/0x1dc) from [<c02c5d74>]
(mount_block_root+0x25c/0x2ac)
[<c02c5d74>] (mount_block_root+0x25c/0x2ac) from [<c02c5f88>]
(prepare_namespace+0x160/0x1b8)
[<c02c5f88>] (prepare_namespace+0x160/0x1b8) from [<c02c5394>]
(kernel_init+0x164/0x1ac)
[<c02c5394>] (kernel_init+0x164/0x1ac) from [<c000fb28>]
(kernel_thread_exit+0x0/0x8)
```

13.2 制作根文件系统

13.2.1 make menuconfig 进行配置

选择 Device Drivers，进入，如图 13. 7 所示。

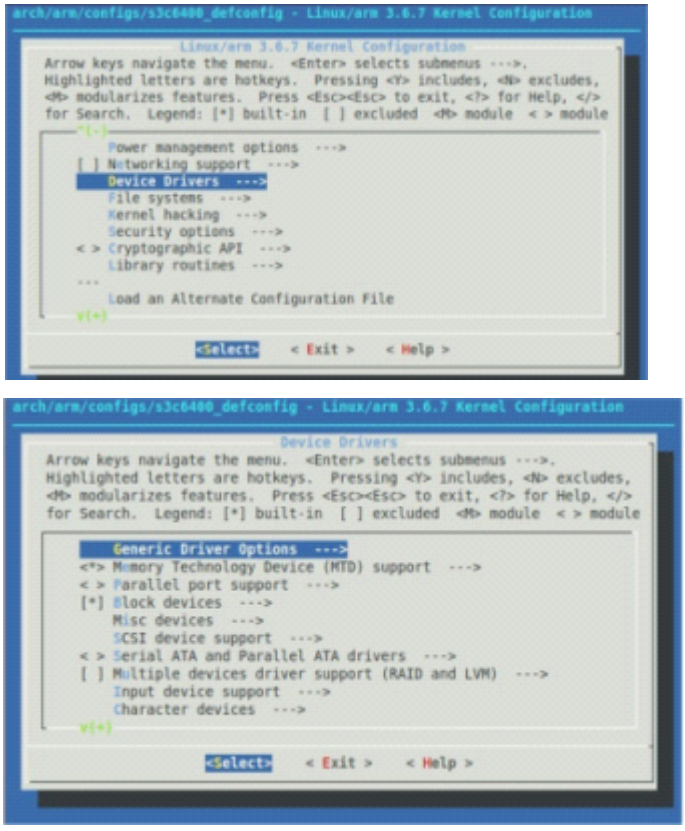


图 13. 8　根文件系统配置步骤二

进入 Generic Driver Options，如图 13. 8 所示。

选择 Maintain ...和 Automount...，如图 13. 9 所示。

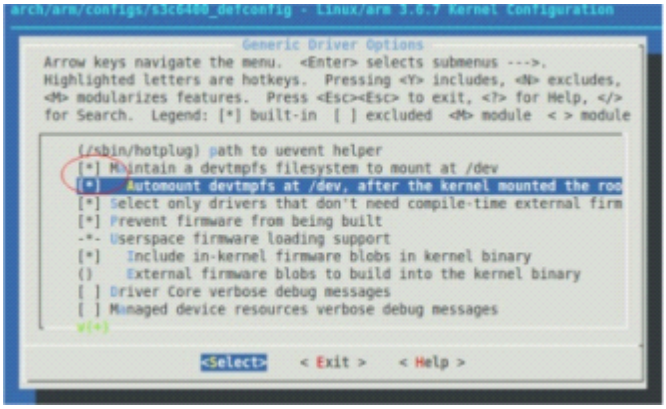


图 13. 9　根文件系统配置步骤三

回到和 Device Drivers 同一目录下，选择 File system，如图 13. 10 所示。

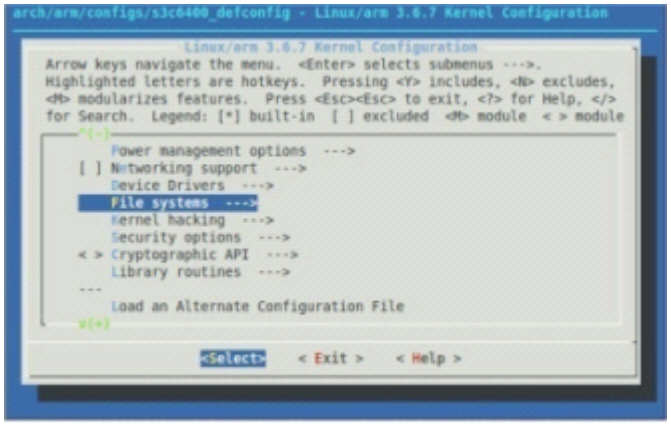


图 13. 10　根文件系统配置步骤四

取消 Second...和 Ext3...配置，如图 13. 11 所示。

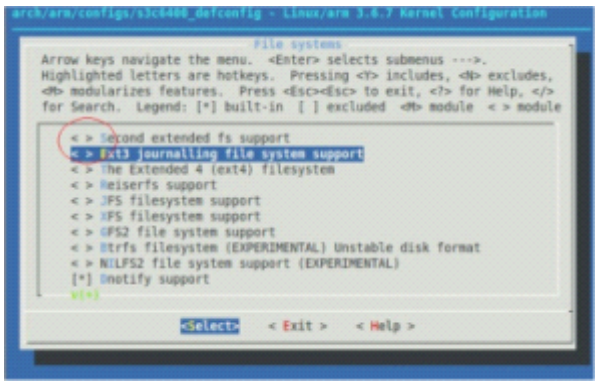


图 13. 11　根文件系统配置步骤五

选择 Miscellaneous filesystems，如图 13. 12 所示。

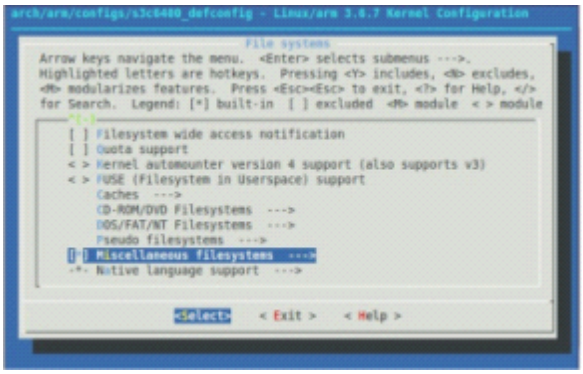


图 13. 12　根文件系统配置步骤六

选择 BeFS...，如图 13. 13 所示。

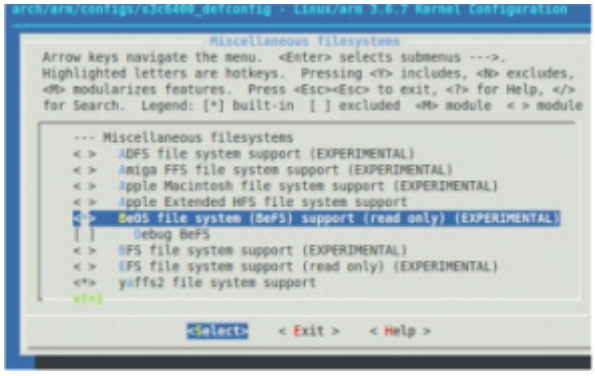


图 13. 13　根文件系统配置步骤七

回到和 Device Drivers 同一目录下，选择 Boot options，如图 13. 14 所示。

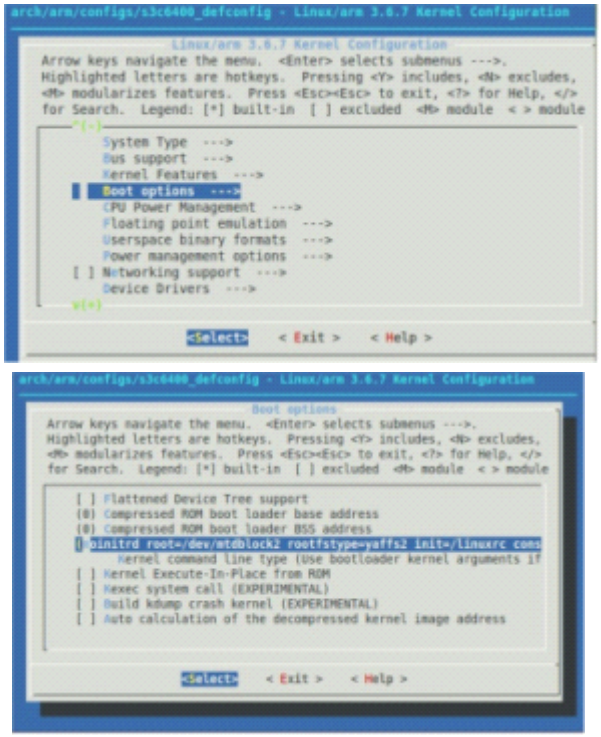


图 13. 15　根文件系统配置步骤九

输入"noinitrd root=/dev/mtdblock2 rootfstype=yaffs2 init=/linuxrc console=ttySAC0,115200"，如图 13. 15 所示。

13.2.2 制作 mkyaffs2image 工具

进入 yaffs2 源码(不是 fs/yaffse)的 utils 文件夹下，修改 mkyaff2image.c。

将 Nand 宏定义这三行注释掉：

```
#if 0
// Adjust these to match your NAND LAYOUT:
#define chunkSize      2048
#define spareSize      64
#define pagesPerBlock  64
#endif
```

更改为：

```
// Adjust these to match your NAND LAYOUT:
#define chunkSize      4096
#define spareSize      218
#define pagesPerBlock  128
```

进入 yaffs2 源码目录下 direct 目录，修改 yportenv.h。

增加：

```
#define CONFIG_YAFFS_DEFINES_TYPES
```

保存后进入 utils，执行 make 命令,这时就在 utils 目录生产 mkyaffs2image 文件，把这个文件拷贝到/usr/bin 目录下。

13.2.3 制作根文件系统

在 http://www.busybox.net/ 中下载 busybox1.20.2，解压缩。

修改 Makefile：

```
CROSS_COMPILE ?=/usr/local/arm/4.4.1/bin/arm-linux-
ARCH ?= arm
```

配置 busybox 菜单如下：

1) make defconfig（默认配置）
2) make menuconfig

选择 Busybox Settings，进入，如图 13. 16 所示。

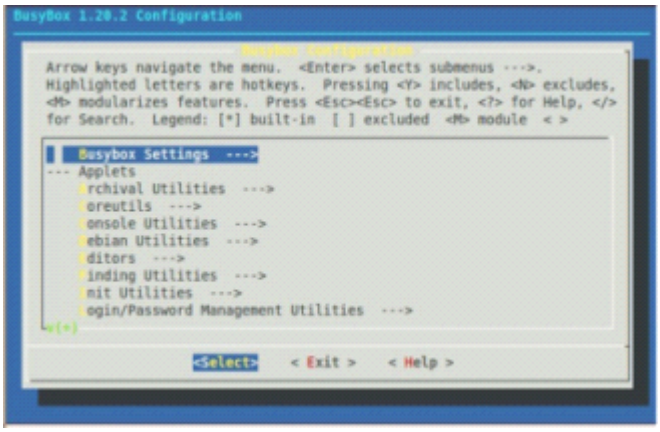


图 13. 16　busybox 配置步骤一

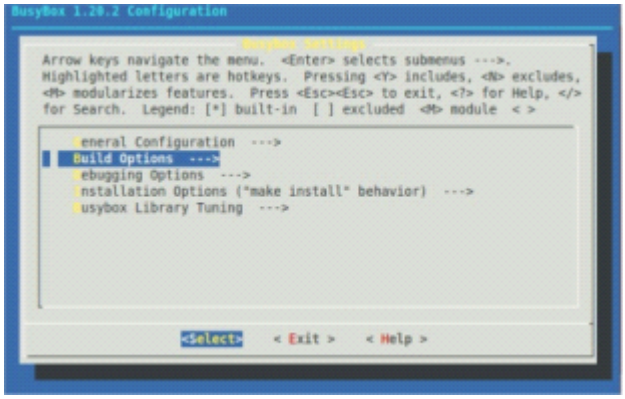


图 13. 17　busybox 配置步骤二

选择 Build Options，进入，如图 13. 17 所示。

选择 Cross Compiler prefix，进入进行编辑，如图 13. 18 和图 13. 19 所示。

回到 Busybox Settings 进入，选择 General Configuration，进入，如图 13. 20 所示。

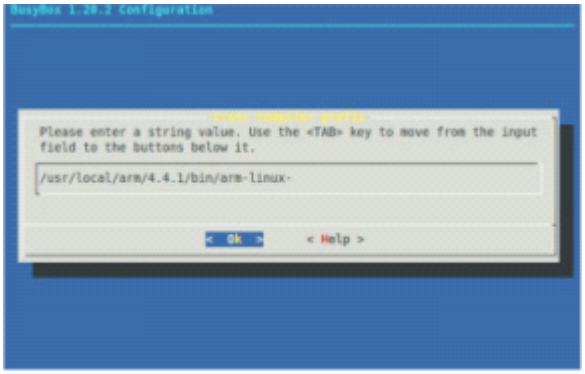


图 13. 18　busybox 配置步骤三

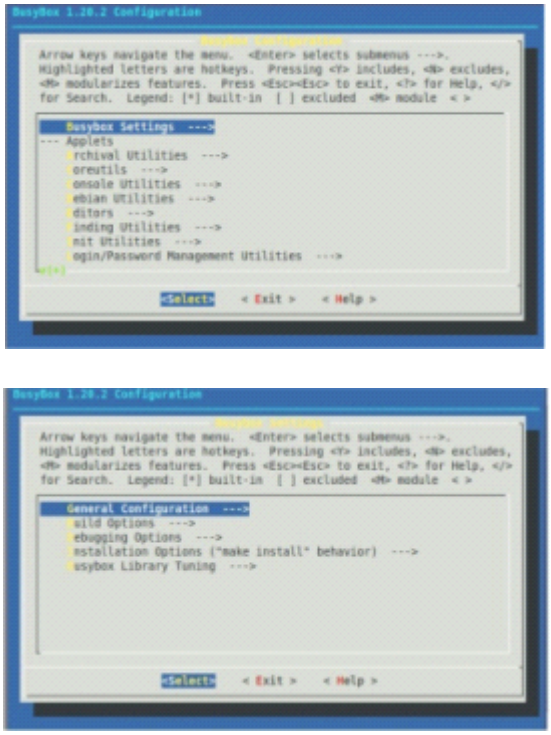


图 13. 20 busybox 配置步骤五

选择 Don't use /usr，如图 13. 21 所示。

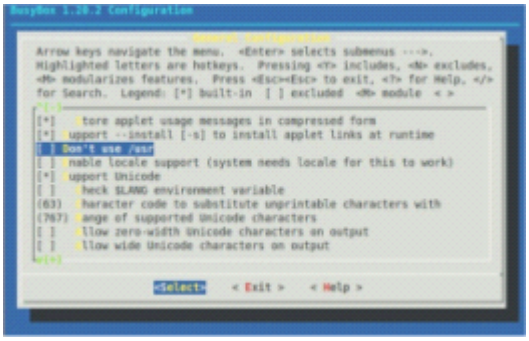


图 13. 21 busybox 配置步骤六

配置好菜单之后进行编译、安装

1) make

2) make install（由于前面没设置安装位置将默认安装在本文件夹的_install 文件夹下）

进入_install 建立其他根文件系统文件夹。为了方便建立，写脚本 creat_initramfs.sh。

```
#!/bin/sh
echo "--------Create root,dev......"
mkdir root dev etc bin sbin mnt sys proc lib home tmp var usr
mkdir usr/sbin usr/bin usr/lib usr/modules usr/etc
```

```
mkdir mnt/usb mnt/nfs mnt/etc mnt/etc/init.d
mkdir lib/modules
chmod 1777 tmp

sudo mknod -m 600 dev/console c 5 1
sudo mknod -m 666 dev/null c 1 3

echo "-------make direction done---------"
```

然后保存脚本修改权限：chmod +x creat_initramfs.sh

运行脚本：sh creat_initramfs.sh，如图 13. 22 所示。

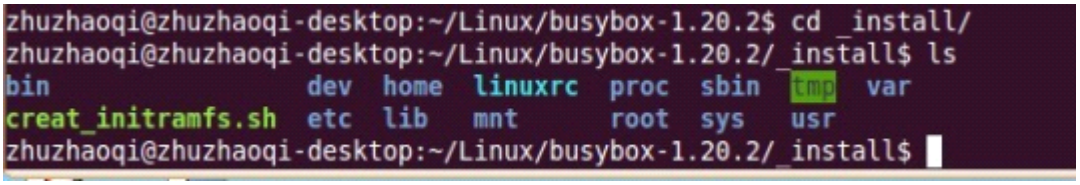


图 13. 22 建立根文件系统其他文件夹

为了保险起见，执行脚本之后执行：chmod 777 linuxrc

在_install/etc 文件夹下，创建 profile 文件，内容如下：

```
# Ash profile
# vim: syntax=sh
# No core files by default
ulimit -S -c 0 > /dev/null 2>&1
USER="'id -un'"
LOGNAME=$USER
PS1='[\u@\h \W]\# '
PATH=$PATH
HOSTNAME='/bin/hostname'
export USER LOGNAME PS1 PATH
```

在_install/etc 文件夹下建立 init.d 文件夹，并在 init.d 文件夹下面创建 rcS 文件，内容如下：

```
#! /bin/sh
PATH=/sbin:/bin:/usr/sbin:/usr/bin:/usr/local/bin:
runlevel=S
prevlevel=N
umask 022
export PATH runlevel prevlevel

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

```
#
trap ":" INT QUIT TSTP
/bin/hostname hcm

/bin/mount -n -t proc none /proc
/bin/mount -n -t sysfs none /sys
/bin/mount -n -t usbfs none /proc/bus/usb
/bin/mount -t ramfs none /dev

echo /sbin/mdev > /proc/sys/kernel/hotplug
/sbin/mdev -s
/bin/hotplug
# mounting file system specified in /etc/fstab
mkdir -p /dev/pts
mkdir -p /dev/shm
/bin/mount -n -t devpts none /dev/pts -o mode=0622
/bin/mount -n -t tmpfs tmpfs /dev/shm
/bin/mount -n -t ramfs none /tmp
/bin/mount -n -t ramfs none /var
mkdir -p /var/empty
mkdir -p /var/log
mkdir -p /var/lock
mkdir -p /var/run
mkdir -p /var/tmp

/sbin/hwclock -s -f /dev/rtc

syslogd
/etc/rc.d/init.d/netd start
echo " " > /dev/tty1
echo "Starting networking..." > /dev/tty1
#sleep 1
#/etc/rc.d/init.d/httpd start
#echo " " > /dev/tty1
#echo "Starting web server..." > /dev/tty1
#sleep 1
#/etc/rc.d/init.d/leds start
#echo " " > /dev/tty1
#echo "Starting leds service..." > /dev/tty1
#echo " "
#sleep 1
echo "************************************"
echo " Welcome to zzq Root! "
echo " "
```

```
echo " Name   : zhuzhaoqi "
echo " E-mali : jxlgzzq@163.com"
echo "************************************"

mkdir /mnt/disk
mount -t yaffs2 /dev/mtdblock3 /mnt/disk

mount -t vfat /dev/mmcblk0p1 /home/
mount -t yaffs2 /dev/mtdblock3 /mnt/
cd /mnt/
tar zxvf /home/urbetter-rootfs-qt-2.2.0.tgz
sync
cd /
umount /mnt/
umount /home/
/sbin/ifconfig lo 127.0.0.1
chmod +x etc/init.d/ifconfig-eth0
/etc/init.d/ifconfig-eth0
/bin/qtopia &
echo " " > /dev/tty1
echo "Starting Qtopia, please waiting..." > /dev/tty1
echo " "
echo "Starting Qtopia, please waiting..."
```

注意修改这个文件权限：chmod +x /etc/init.d/rcS。

在_install/etc 文件夹下面创建 fstab 文件，内容如下：

```
proc /proc proc defaults 0 0
none /tmp ramfs defaults 0 0
none /var ramfs defaults 0 0
mdev /dev ramfs defaults 0 0
sysfs /sys sysfs defaults 0 0
```

在_install/etc 文件夹下创建 inittab 文件，内容如下：

```
::sysinit:/etc/init.d/rcS
::askfirst:-/bin/sh
::ctrlaltdel:/bin/umount -a -r
::shutdown:/bin umount -a -r
::shutdown:/sbin/swapoff -a
```

在_install/usr/etc 文件夹下面创建 init 文件，内容如下：

```
#!/bin/sh
ifconfig eth0 192.168.1.0 up
```

```
ifconfig lo 127.0.0.1
```

修改权限：chmod +x usr/etc/init。

在_install/usr/etc 文件夹下面创建 mdev.conf(空文件)。

拷贝相应工具链中的库文件到 lib 当中，在_install 文件夹下面执行命令：

```
cp /usr/local/arm/4.4.1/arm-none-linux-gnueabi/libc/lib/*so* lib
```

用 mkyaffs2image 工具制作根文件系统：

```
…busybox-1.20.2$ mkyaffs2image _install rootfs.yaffs
```

修改 u-boot-2012.10/driver/mtd/nand 文件夹下的 nand_util.c 文件 nand_write_skip_bad()函数，红色程序为添加：

```
      if (!need_skip && !(flags & WITH_DROP_FFS))
      {
         if(flags & WITH_YAFFS_OOB)
         {
            printf ("NAND write to offset= %llx \n",offset);
         }

         else
         {

            rval = nand_write (nand, offset, length, buffer);
            if (rval == 0)
                  return 0;

            *length = 0;
            printf ("NAND write to offset %llx failed %d\n",
                  offset, rval);
            return rval;
         }
      }
```

编译之后，u-boot.bin，uImage，yaffs2 三个文件启动，出现如下错误：

```
List of all partitions:
1f00          2048 mtdblock0  (driver?)
1f01          5120 mtdblock1  (driver?)
1f02        204800 mtdblock2  (driver?)
1f03       1885184 mtdblock3  (driver?)
No filesystem could mount root, tried:  cramfs
```

```
Kernel panic - not syncing: VFS: Unable to mount root fs on
unknown-block(31,2)
[<c0014d28>] (unwind_backtrace+0x0/0xf4) from [<c0215ce0>]
(panic+0x8c/0x1dc)
[<c0215ce0>] (panic+0x8c/0x1dc) from [<c02c5d74>]
(mount_block_root+0x25c/0x2ac)
[<c02c5d74>] (mount_block_root+0x25c/0x2ac) from [<c02c5f88>]
(prepare_namespace+0x160/0x1b8)
[<c02c5f88>] (prepare_namespace+0x160/0x1b8) from [<c02c5394>]
(kernel_init+0x164/0x1ac)
[<c02c5394>] (kernel_init+0x164/0x1ac) from [<c000fb28>]
(kernel_thread_exit+0x0/0x8)
```

u-boot、linux、yaffs2 三个文件的 ECC 校验确保一致（都为硬件 ECC，在 make menuconfig 中配置）

在 u-boot 中设置：

```
set bootargs "noinitrd root=/dev/mtdblock2 rootfstype=yaffs2
init=/linuxrc console=ttySAC0,115200"
```

再次编译启动：

```
Starting kernel ...

Uncompressing Linux... done, booting the kernel.
Booting Linux on physical CPU 0
Linux version 3.6.7 (zhuzhaoqi@zhuzhaoqi-desktop) (gcc version 4.4.1
(Sourcery G++ Lite 2009q3-67) ) #1 Fri Dec 14 13:36:46 CST 2012
CPU: ARMv6-compatible processor [410fb766] revision 6 (ARMv7),
cr=00c5387d
CPU: PIPT / VIPT nonaliasing data cache, VIPT nonaliasing instruction
cache
Machine: OK6410
Memory policy: ECC disabled, Data cache writeback
CPU S3C6410 (id 0x36410101)
S3C24XX Clocks, Copyright 2004 Simtec Electronics
camera: no parent clock specified
S3C64XX: PLL settings, A=533000000, M=533000000, E=24000000
S3C64XX: HCLK2=266500000, HCLK=133250000, PCLK=66625000
mout_apll: source is fout_apll (1), rate is 533000000
mout_epll: source is epll (1), rate is 24000000
mout_mpll: source is mpll (1), rate is 533000000
usb-bus-host: source is clk_48m (0), rate is 48000000
audio-bus: source is mout_epll (0), rate is 24000000
audio-bus: source is mout_epll (0), rate is 24000000
```

```
audio-bus: source is mout_epll (0), rate is 24000000
irda-bus: source is mout_epll (0), rate is 24000000
camera: no parent clock specified
CPU: found DTCM0 8k @ 00000000, not enabled
CPU: moved DTCM0 8k to fffe8000, enabled
CPU: found DTCM1 8k @ 00000000, not enabled
CPU: moved DTCM1 8k to fffea000, enabled
CPU: found ITCM0 8k @ 00000000, not enabled
CPU: moved ITCM0 8k to fffe0000, enabled
CPU: found ITCM1 8k @ 00000000, not enabled
CPU: moved ITCM1 8k to fffe2000, enabled
Built 1 zonelists in Zone order, mobility grouping on.  Total pages:
65024
Kernel command line: noinitrd root=/dev/mtdblock2 rootfstype=yaffs2
init=/linuxrc console=ttySAC0,115200
PID hash table entries: 1024 (order: 0, 4096 bytes)
Dentry cache hash table entries: 32768 (order: 5, 131072 bytes)
Inode-cache hash table entries: 16384 (order: 4, 65536 bytes)
Memory: 256MB = 256MB total
Memory: 256664k/256664k available, 5480k reserved, 0K highmem
Virtual kernel memory layout:
    vector  : 0xffff0000 - 0xffff1000   (   4 kB)
    DTCM    : 0xfffe8000 - 0xfffec000   (  16 kB)
    ITCM    : 0xfffe0000 - 0xfffe4000   (  16 kB)
    fixmap  : 0xfff00000 - 0xfffe0000   ( 896 kB)
    vmalloc : 0xd0800000 - 0xff000000   ( 744 MB)
    lowmem  : 0xc0000000 - 0xd0000000   ( 256 MB)
    modules : 0xbf000000 - 0xc0000000   (  16 MB)
      .text : 0xc0008000 - 0xc029a3c8   (2633 kB)
      .init : 0xc029b000 - 0xc02b623c   ( 109 kB)
      .data : 0xc02b8000 - 0xc02e51c0   ( 181 kB)
       .bss : 0xc02e6024 - 0xc031712c   ( 197 kB)
SLUB: Genslabs=13, HWalign=32, Order=0-3, MinObjects=0, CPUs=1,
Nodes=1
NR_IRQS:246
VIC @f6000000: id 0x00041192, vendor 0x41
VIC @f6010000: id 0x00041192, vendor 0x41
sched_clock: 32 bits at 100 Hz, resolution 10000000ns, wraps every
4294967286ms
Console: colour dummy device 80x30
Calibrating delay loop... 353.89 BogoMIPS (lpj=1769472)
pid_max: default: 32768 minimum: 301
Mount-cache hash table entries: 512
CPU: Testing write buffer coherency: ok
```

```
Setting up static identity map for 0x501f5df8 - 0x501f5e54
devtmpfs: initialized
DMA: preallocated 256 KiB pool for atomic coherent allocations
OK6410: Option string ok6410=0
OK6410: selected LCD display is 480x272
s3c64xx_dma_init: Registering DMA channels
PL080: IRQ 73, at d0846000, channels 0..8
PL080: IRQ 74, at d0848000, channels 8..16
S3C6410: Initialising architecture
bio: create slab <bio-0> at 0
usbcore: registered new interface driver usbfs
usbcore: registered new interface driver hub
usbcore: registered new device driver usb
ROMFS MTD (C) 2007 Red Hat, Inc.
BeFS version: 0.9.3
io scheduler noop registered
io scheduler deadline registered
io scheduler cfq registered (default)
start plist test
end plist test
s3c-fb s3c-fb: window 0: fb
Serial: 8250/16550 driver, 4 ports, IRQ sharing disabled
s3c6400-uart.0: ttySAC0 at MMIO 0x7f005000 (irq = 69) is a S3C6400/10
console [ttySAC0] enabled
s3c6400-uart.1: ttySAC1 at MMIO 0x7f005400 (irq = 70) is a S3C6400/10
s3c6400-uart.2: ttySAC2 at MMIO 0x7f005800 (irq = 71) is a S3C6400/10
s3c6400-uart.3: ttySAC3 at MMIO 0x7f005c00 (irq = 72) is a S3C6400/10
brd: module loaded
loop: module loaded
S3C NAND Driver, (c) 2008 Samsung Electronics
dev_id == 0xd5 select s3c_nand_oob_mlc
****Nandflash:ChipType= MLC
ChipName=samsung-K9GAG08U0D************
S3C NAND Driver is using hardware ECC.
NAND device: Manufacturer ID: 0xec, Chip ID: 0xd5 (Samsung NAND 2GiB
3,3V 8-bit), page size: 4096, OOB size: 218
Driver must set ecc.strength when using hardware ECC
Creating 4 MTD partitions on "NAND 2GiB 3,3V 8-bit":
0x000000000000-0x000000200000 : "Bootloader"
0x000000200000-0x000000700000 : "Kernel"
0x000000700000-0x00000cf00000 : "File System"
0x00000cf00000-0x000080000000 : "User"
ohci_hcd: USB 1.1 'Open' Host Controller (OHCI) Driver
s3c2410-ohci s3c2410-ohci: S3C24XX OHCI
```

```
s3c2410-ohci s3c2410-ohci: new USB bus registered, assigned bus number
1
s3c2410-ohci s3c2410-ohci: irq 79, io mem 0x74300000
s3c2410-ohci s3c2410-ohci: init err (00000000 0000)
s3c2410-ohci s3c2410-ohci: can't start s3c24xx
s3c2410-ohci s3c2410-ohci: startup error -75
s3c2410-ohci s3c2410-ohci: USB bus 1 deregistered
s3c2410-ohci: probe of s3c2410-ohci failed with error -75
mousedev: PS/2 mouse device common for all mice
i2c /dev entries driver
sdhci: Secure Digital Host Controller Interface driver
sdhci: Copyright(c) Pierre Ossman
s3c-sdhci s3c-sdhci.0: clock source 0: mmc_busclk.0 (133250000 Hz)
s3c-sdhci s3c-sdhci.0: clock source 2: mmc_busclk.2 (24000000 Hz)
mmc0: SDHCI controller on samsung-hsmmc [s3c-sdhci.0] using ADMA
s3c-sdhci s3c-sdhci.1: clock source 0: mmc_busclk.0 (133250000 Hz)
s3c-sdhci s3c-sdhci.1: clock source 2: mmc_busclk.2 (24000000 Hz)
mmc0: mmc_rescan_try_freq: trying to init card at 400000 Hz
mmc0: mmc_rescan_try_freq: trying to init card at 300000 Hz
mmc1: SDHCI controller on samsung-hsmmc [s3c-sdhci.1] using ADMA
mmc0: mmc_rescan_try_freq: trying to init card at 200000 Hz
usbcore: registered new interface driver usbhid
usbhid: USB HID core driver
VFP support v0.3: implementor 41 architecture 1 part 20 variant b rev
5
drivers/rtc/hctosys.c: unable to open rtc device (rtc0)
yaffs: dev is 32505858 name is "mtdblock2" rw
yaffs: passed flags ""
mmc0: mmc_rescan_try_freq: trying to init card at 100000 Hz
VFS: Mounted root (yaffs2 filesystem) on device 31:2.
devtmpfs: error mounting -2
Freeing init memory: 108K
Failed to execute /linuxrc.  Attempting defaults...
Kernel panic - not syncing: No init found.  Try passing init= option
to kernel. See Linux Documentation/init.txt for guidance.
[<c0014d28>] (unwind_backtrace+0x0/0xf4) from [<c01f35ac>]
(panic+0x8c/0x1dc)
[<c01f35ac>] (panic+0x8c/0x1dc) from [<c00087f0>]
(init_post+0x80/0xd0)
[<c00087f0>] (init_post+0x80/0xd0) from [<c029b398>]
(kernel_init+0x168/0x1ac)
```

Device Driver ->Generic Driver Options ->(取消)devtmpfs。 error mounting -2 被解决了。如下所示：

```
yaffs: dev is 32505858 name is "mtdblock2" rw
yaffs: passed flags ""
mmc0: mmc_rescan_try_freq: trying to init card at 100000 Hz
VFS: Mounted root (yaffs2 filesystem) on device 31:2.
Freeing init memory: 108K
Failed to execute /linuxrc.  Attempting defaults...
Kernel panic - not syncing: No init found.  Try passing init= option
to kernel. See Linux Documentation/init.txt for guidance.
```

（待续）

13.3 NFS 文件系统挂载

采用 NFS 文件系统挂载的最大好处是可以及时更新文件系统中的文件。

首先在 ubuntu 中搭建好 NFS。

```
#sudo apt-get install portmap

#sudo apt-get install nfs-kernel-server
```

添加 rootfs 路径：

```
#sudo vim /etc/exports
```

如下：

```
# /etc/exports: the access control list for filesystems which may be
exported
#             to NFS clients.  See exports(5).
#
# Example for NFSv2 and NFSv3:
# /srv/homes       hostname1(rw,sync,no_subtree_check)
hostname2(ro,sync,no_subtree_check)
#
# Example for NFSv4:
# /srv/nfs4
gss/krb5i(rw,sync,fsid=0,crossmnt,no_subtree_check)
# /srv/nfs4/homes  gss/krb5i(rw,sync,no_subtree_check)
#

/home/zhuzhaoqi/rootfs *(rw,sync,no_root_squash)
/home/zhuzhaoqi/rootfs/mini_rootfs *(rw,sync,no_root_squash)
```

使/etc/exports 文件生效：

```
#sudo exportfs -rv
```

启动端口映射：

```
#/etc/init.d/portmap start（或：#sudo service portmap start）
```

最后启动 NFS 服务，此时 NFS 会激活守护进程，然后就开始监听 Client 端的请求：

```
#/etc/init.d/nfs-kernel-server restart
（或： #sudo service nfs-kernel-server restart）
```

参考：

http://www.cnblogs.com/yyangblog/archive/2011/06/14/2080636.html

和

http://blog.csdn.net/eastmoon502136/article/details/7905960

此时应该就搭建好了 NFS。

将 busybox/_install 文件夹下的所有文件（即是根文件系统）复制到/rootfs 文件夹下，即为 NFS 挂载地中。

启动 OK6410 开发板，设置启动参数：

```
set bootargs root=/dev/nfs console=ttySAC0,115200
nfsroot=192.168.1.9:/home/zhuzhaoqi/rootfs
ip=192.168.1.100:192.168.1.9:192.168.1.1:255.255.255.0::eth0:off
```

```
nfsroot=192.168.1.9:/home/zhuzhaoqi/rootfs
```

这是 nfs 文件系统存放在 ubuntu 中的地址，192.168.1.9 是 ubuntu 的 IP。

设置 IP：

```
OK6410 的 IP:
zzq6410 >>> set ipaddr 192.168.1.100

OK6410 的网关:
zzq6410 >>> set gatewayip 192.168.1.1

ubuntu 服务地址:
zzq6410 >>> set serverip 192.168.1.9

OK6410 的子网掩码:
zzq6410 >>> set netmask 255.255.255.0
```

启动开发板：

```
U-Boot 2012.10 (Dec 17 2012 - 09:33:49) for OK6410

Author :  zhuzhaoqi
E-mail : jxlgzzq@163.com

CPU:    S3C6410@533MHz
        Fclk = 533MHz, Hclk = 133MHz, Pclk = 66MHz (ASYNC Mode)
Board:   OK6410
DRAM:  256 MiB
NAND:  2048 MiB
In:    serial
Out:   serial
Err:   serial
Net:   dm9000
Unknown command 'mtdparts' - try 'help'
Hit any key to stop autoboot:  0

NAND read: device 0 offset 0x100000, size 0x500000
s3c-nand: 1 bit(s) error detected, corrected successfully
 5242880 bytes read: OK
## Booting kernel from Legacy Image at 50008000 ...
   Image Name:   Linux-3.6.7
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    2144632 Bytes = 2 MiB
   Load Address: 50008000
   Entry Point:  50008040
   Verifying Checksum ... OK
   XIP Kernel Image ... OK
OK

Starting kernel ...


Starting kernel ...

Uncompressing Linux... done, booting the kernel.
Booting Linux on physical CPU 0
Linux version 3.6.7 (root@zhuzhaoqi-desktop) (gcc version 4.4.1
(Sourcery G++ Lite 2009q3-67) ) #1 Wed Dec 19 17:42:07 CST 2012
CPU: ARMv6-compatible processor [410fb766] revision 6 (ARMv7),
cr=00c5387d
CPU: PIPT / VIPT nonaliasing data cache, VIPT nonaliasing instruction
cache
```

```
Machine: OK6410
Memory policy: ECC disabled, Data cache writeback
CPU S3C6410 (id 0x36410101)
S3C24XX Clocks, Copyright 2004 Simtec Electronics
camera: no parent clock specified
S3C64XX: PLL settings, A=533000000, M=533000000, E=24000000
S3C64XX: HCLK2=266500000, HCLK=133250000, PCLK=66625000
mout_apll: source is fout_apll (1), rate is 533000000
mout_epll: source is epll (1), rate is 24000000
mout_mpll: source is mpll (1), rate is 533000000
usb-bus-host: source is clk_48m (0), rate is 48000000
audio-bus: source is mout_epll (0), rate is 24000000
audio-bus: source is mout_epll (0), rate is 24000000
audio-bus: source is mout_epll (0), rate is 24000000
irda-bus: source is mout_epll (0), rate is 24000000
camera: no parent clock specified
CPU: found DTCM0 8k @ 00000000, not enabled
CPU: moved DTCM0 8k to fffe8000, enabled
CPU: found DTCM1 8k @ 00000000, not enabled
CPU: moved DTCM1 8k to fffea000, enabled
CPU: found ITCM0 8k @ 00000000, not enabled
CPU: moved ITCM0 8k to fffe0000, enabled
CPU: found ITCM1 8k @ 00000000, not enabled
CPU: moved ITCM1 8k to fffe2000, enabled
Built 1 zonelists in Zone order, mobility grouping on.  Total pages:
65024
Kernel command line: root=/dev/nfs console=ttySAC0,115200
nfsroot=192.168.1.9:/home/zhuzhaoqi/rootfs
ip=192.168.1.100:192.168.1.9:192.168.1.1:255.255.255.0::eth0:off
PID hash table entries: 1024 (order: 0, 4096 bytes)
Dentry cache hash table entries: 32768 (order: 5, 131072 bytes)
Inode-cache hash table entries: 16384 (order: 4, 65536 bytes)
Memory: 256MB = 256MB total
Memory: 255436k/255436k available, 6708k reserved, 0K highmem
Virtual kernel memory layout:
    vector  : 0xffff0000 - 0xffff1000   (   4 kB)
    DTCM    : 0xfffe8000 - 0xfffec000   (  16 kB)
    ITCM    : 0xfffe0000 - 0xfffe4000   (  16 kB)
    fixmap  : 0xfff00000 - 0xfffe0000   ( 896 kB)
    vmalloc : 0xd0800000 - 0xff000000   ( 744 MB)
    lowmem  : 0xc0000000 - 0xd0000000   ( 256 MB)
    modules : 0xbf000000 - 0xc0000000   (  16 MB)
      .text : 0xc0008000 - 0xc03b9d70   (3784 kB)
      .init : 0xc03ba000 - 0xc03d949c   ( 126 kB)
```

```
     .data : 0xc03da000 - 0xc040f780   ( 214 kB)
      .bss : 0xc0410024 - 0xc044ae20   ( 236 kB)
SLUB: Genslabs=13, HWalign=32, Order=0-3, MinObjects=0, CPUs=1,
Nodes=1
NR_IRQS:246
VIC @f6000000: id 0x00041192, vendor 0x41
VIC @f6010000: id 0x00041192, vendor 0x41
sched_clock: 32 bits at 100 Hz, resolution 10000000ns, wraps every
4294967286ms
Console: colour dummy device 80x30
Calibrating delay loop... 353.89 BogoMIPS (lpj=1769472)
pid_max: default: 32768 minimum: 301
Mount-cache hash table entries: 512
CPU: Testing write buffer coherency: ok
Setting up static identity map for 0x502c3ef0 - 0x502c3f4c
NET: Registered protocol family 16
DMA: preallocated 256 KiB pool for atomic coherent allocations
OK6410: Option string ok6410=0
OK6410: selected LCD display is 480x272
s3c64xx_dma_init: Registering DMA channels
PL080: IRQ 73, at d0846000, channels 0..8
PL080: IRQ 74, at d0848000, channels 8..16
S3C6410: Initialising architecture
bio: create slab <bio-0> at 0
usbcore: registered new interface driver usbfs
usbcore: registered new interface driver hub
usbcore: registered new device driver usb
NET: Registered protocol family 2
TCP established hash table entries: 8192 (order: 4, 65536 bytes)
TCP bind hash table entries: 8192 (order: 5, 163840 bytes)
TCP: Hash tables configured (established 8192 bind 8192)
TCP: reno registered
UDP hash table entries: 256 (order: 1, 12288 bytes)
UDP-Lite hash table entries: 256 (order: 1, 12288 bytes)
NET: Registered protocol family 1
RPC: Registered named UNIX socket transport module.
RPC: Registered udp transport module.
RPC: Registered tcp transport module.
RPC: Registered tcp NFSv4.1 backchannel transport module.
NFS: Registering the id_resolver key type
Key type id_resolver registered
Key type id_legacy registered
Installing knfsd (copyright (C) 1996 okir@monad.swb.de).
ROMFS MTD (C) 2007 Red Hat, Inc.
```

```
io scheduler noop registered
io scheduler deadline registered
io scheduler cfq registered (default)
start plist test
end plist test
Console: switching to colour frame buffer device 60x34
s3c-fb s3c-fb: window 0: fb
Serial: 8250/16550 driver, 4 ports, IRQ sharing disabled
s3c6400-uart.0: ttySAC0 at MMIO 0x7f005000 (irq = 69) is a S3C6400/10
console [ttySAC0] enabled
s3c6400-uart.1: ttySAC1 at MMIO 0x7f005400 (irq = 70) is a S3C6400/10
s3c6400-uart.2: ttySAC2 at MMIO 0x7f005800 (irq = 71) is a S3C6400/10
s3c6400-uart.3: ttySAC3 at MMIO 0x7f005c00 (irq = 72) is a S3C6400/10
brd: module loaded
loop: module loaded
S3C24XX NAND Driver, (c) 2004 Simtec Electronics
dm9000 Ethernet Driver, V1.31
dm9000 dm9000: eth%d: Invalid ethernet MAC address. Please set using
ifconfig
eth0: dm9000a at d08de000,d08e0004 IRQ 108 MAC: 8a:7d:53:40:7a:e7
(random)
ohci_hcd: USB 1.1 'Open' Host Controller (OHCI) Driver
s3c2410-ohci s3c2410-ohci: S3C24XX OHCI
s3c2410-ohci s3c2410-ohci: new USB bus registered, assigned bus number
1
s3c2410-ohci s3c2410-ohci: irq 79, io mem 0x74300000
s3c2410-ohci s3c2410-ohci: init err (00000000 0000)
s3c2410-ohci s3c2410-ohci: can't start s3c24xx
s3c2410-ohci s3c2410-ohci: startup error -75
s3c2410-ohci s3c2410-ohci: USB bus 1 deregistered
s3c2410-ohci: probe of s3c2410-ohci failed with error -75
mousedev: PS/2 mouse device common for all mice
i2c /dev entries driver
sdhci: Secure Digital Host Controller Interface driver
sdhci: Copyright(c) Pierre Ossman
s3c-sdhci s3c-sdhci.0: clock source 0: mmc_busclk.0 (133250000 Hz)
s3c-sdhci s3c-sdhci.0: clock source 2: mmc_busclk.2 (24000000 Hz)
mmc0: SDHCI controller on samsung-hsmmc [s3c-sdhci.0] using ADMA
s3c-sdhci s3c-sdhci.1: clock source 0: mmc_busclk.0 (133250000 Hz)
s3c-sdhci s3c-sdhci.1: clock source 2: mmc_busclk.2 (24000000 Hz)
mmc0: mmc_rescan_try_freq: trying to init card at 400000 Hz
mmc0: mmc_rescan_try_freq: trying to init card at 300000 Hz
mmc1: SDHCI controller on samsung-hsmmc [s3c-sdhci.1] using ADMA
mmc0: mmc_rescan_try_freq: trying to init card at 200000 Hz
```

```
usbcore: registered new interface driver usbhid
usbhid: USB HID core driver
TCP: cubic registered
NET: Registered protocol family 17
Key type dns_resolver registered
VFP support v0.3: implementor 41 architecture 1 part 20 variant b rev
5
drivers/rtc/hctosys.c: unable to open rtc device (rtc0)
mmc0: mmc_rescan_try_freq: trying to init card at 100000 Hz
dm9000 dm9000: eth0: link down
mmc1: mmc_rescan_try_freq: trying to init card at 400000 Hz
mmc1: mmc_rescan_try_freq: trying to init card at 300000 Hz
mmc1: mmc_rescan_try_freq: trying to init card at 200000 Hz
mmc1: mmc_rescan_try_freq: trying to init card at 100000 Hz
IP-Config: Complete:
    device=eth0, addr=192.168.1.100, mask=255.255.255.0,
gw=192.168.1.1
    host=192.168.1.100, domain=, nis-domain=(none)
    bootserver=192.168.1.9, rootserver=192.168.1.9, rootpath=
dm9000 dm9000: eth0: link up, 100Mbps, full-duplex, lpa 0x45E1
VFS: Mounted root (nfs filesystem) on device 0:9.
Freeing init memory: 124K
mount: mounting none on /proc/bus/usb failed: No such file or directory
/etc/init.d/rcS: line 21: /bin/hotplug: not found
hwclock: can't open '/dev/rtc': No such file or directory
/etc/init.d/rcS: line 38: /etc/rc.d/init.d/netd: not found
************************************
 Welcome to zzq Root!

 Name   : zhuzhaoqi
 E-mali : jxlgzzq@163.com
************************************
mkdir: can't create directory '/mnt/disk': File exists
mount: mounting /dev/mtdblock3 on /mnt/disk failed: No such file or
directory
mount: mounting /dev/mmcblk0p1 on /home/ failed: No such device
mount: mounting /dev/mtdblock3 on /mnt/ failed: No such file or
directory
tar: can't open '/home/urbetter-rootfs-qt-2.2.0.tgz': No such file
or directory
umount: can't umount /mnt/: Invalid argument
umount: can't umount /home/: Invalid argument
chmod: etc/init.d/ifconfig-eth0: No such file or directory
/etc/init.d/rcS: line 71: /etc/init.d/ifconfig-eth0: not found
```

```
Starting Qtopia, please waiting...

Please press Enter to activate this console. /etc/init.d/rcS: line
72: /bin/qtopia: not found

[YJR@zhuzhaoqi /]#
[YJR@zhuzhaoqi /]# ls
bin     etc     lib     mnt     root    sys     usr
dev     home    linuxrc  proc    sbin    tmp     var
[YJR@zhuzhaoqi /]#
```

第 14 章 Linux 驱动之交叉编译 Hello

14.1 Hello 程序

在/home/zhuzhaoqi/rootfs 中挂载好 nfs 文件系统之后，效果如图 14. 1 所示。

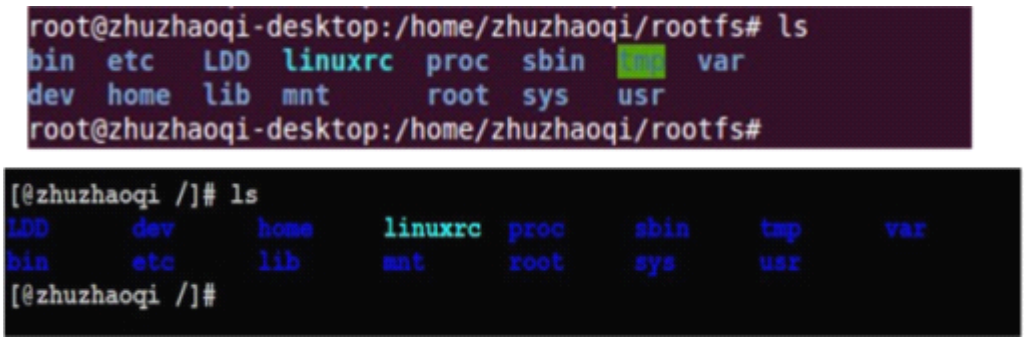


图 14. 1　nfs 文件系统效果

在 ubuntu 中的/home/zhuzhaoqi/rootfs 文件夹下建立 LDD 文件夹，用来编写驱动程序。如果更改/home/zhuzhaoqi/rootfs，重新启动 OK6410 即可同步 nfs 文件系统信息。

进入/home/zhuzhaoqi/rootfs/LDD 文件夹，建立一个 hello 文件夹，用来编写第一个程序。

Hello 程序如程序清单 14. 1 所示。

程序清单 14. 1　hello

```
#include <stdio.h>

int main(int argc, char* argv[])
{
    printf("\nHello,zhuzhaoqi!\n\n");

    return 0;
}
```

在 Linux 中要熟练编写 Makefile 文件，如程序清单 14. 2 所示。

程序清单 14. 2　Makefile

```
CC = /usr/local/arm/4.4.1/bin/arm-linux-gcc

hello:hello.o
	$(CC) -o hello hello.o

hello.o:hello.c
	$(CC) -c hello.c

clean:
	rm hello.o hello
```

make 之后可以得到：hello　hello.c　hello.o　Makefile。

重启开发板，可以看到 LDD 下面有着四个文件，执行./hello。效果如程序清单 14. 3 所示。

```
[@zhuzhaoqi /]# cd LDD/
[@zhuzhaoqi /LDD]# ls
hello
[@zhuzhaoqi /LDD]# cd hello/
[@zhuzhaoqi hello]# ls
Makefile  hello     hello.c   hello.o
[@zhuzhaoqi hello]# ./hello

Hello,zhuzhaoqi!

[@zhuzhaoqi hello]#
```

程序清单 14. 3　执行 hello 效果

第 15 章 Linux3.6.7 驱动之 LED

15.1 LED 裸板程序

在写 Linux3.6.7 驱动之 LED 之前，先来看一下 LED 的裸板程序。

很显然，先得明白 OK6410 开发平台上 4 个 LED 硬件电路图，如图 15. 1 所示。

图 15. 1 LED 连接图

既然控制 LED1~LED4 的四个 GPIO 口是 GPM0~GPM3，那么可以从 s3c6410 手册中得知：

```
/*
**  端口 M 控制寄存器
**  GPMCON 寄存器地址: 0x7F00 8820
**  GPMDAT 寄存器地址: 0x7F00 8824
**
*/
/*=============================================================
**  基地址的定义
=============================================================*/
#define    AHB_BASE      (0x7F00 0000)
/*************************************************************
**  GPX 的地址定义
*************************************************************/
#define    GPX_BASE      (AHB_BASE+0x0 8000)
......
......
/*************************************************************
**      GPM 寄存器地址定义
*************************************************************/
#define    GPMCON     (*(volatile unsigned long *)(GPX_BASE + 0x0820))
#define    GPMDAT     (*(volatile unsigned long *)(GPX_BASE + 0x0824))
#define    GPMPUD     (*(volatile unsigned long *)(GPX_BASE + 0x0828))
```

将 GPM0~GPM3 设置为输出功能：

```
/* GPM0,1,2,3 设为输出引脚 */
```

```
/*
**  每一个 GPXCON 的引脚有  4 位二进制进行控制
**  0000-输入      0001-输出
*/
GPMCON  = 0x1111;
```

点亮 LED1，则是让 GPM3~GPM0 输出：1110。

```
GPMDAT  = 0x0e;
```

点亮 LED3，则是让 GPM3~GPM0 输出：1011。

```
GPMDAT  = 0x0b;
```

15.2 Linux 中的 LED 驱动程序

有了上面裸板 LED 的基础，移植到 Linux 中，难度也不会很大了。但是在 Linux 中，特别注意几个方面。

其一，s3c6410 提供的 GPM 寄存器的地址不能直接用于 Linux 中。

一般情况下，Linux 系统中，进程的 4GB（　）内存空间被划分成为两个部分：用户

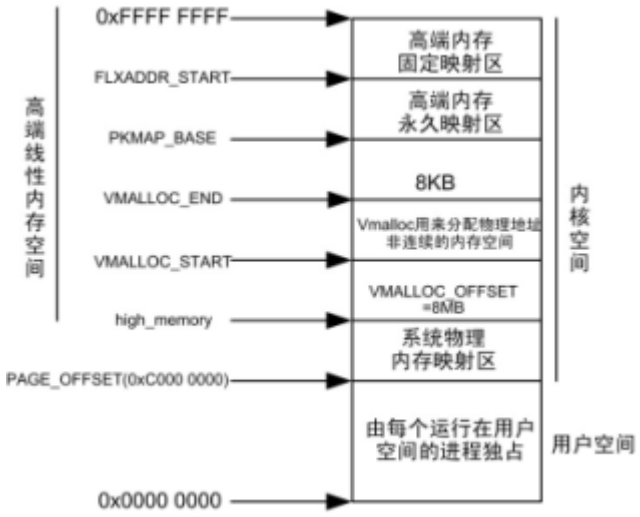


图 15.2　Linux 内存空间

空间(3G)和内核空间(1G)，大小分别为 0~3G，3~4G。如图 15.2 所示。

3~4G 之间的内核空间中，从低地址到高地址依次为：物理内存映射区、隔离带、vmalloc 虚拟内存分配区、隔离带、高端内存映射区、专用页面映射区、保留区。

用户进程通常情况下，只能访问用户空间的虚拟地址，不能访问到内核空间。

每个进程的用户空间都是完全独立、互不相干的，用户进程各自有不同的页表。而内核空间是由内核负责映射，它并不会跟着进程改变，是固定的。内核空间地址有自己对应的页表，内核的虚拟空间独立于其他程序。

在内核中，访问 IO 内存之前，我们只有 IO 内存的物理地址，这样是无法通过软件直

接访问的，需要首先用 ioremap()函数将设备所处的物理地址映射到内核虚拟地址空间（3GB~4GB）。然后，才能根据映射所得到的内核虚拟地址范围，通过访问指令访问这些 IO 内存资源。

一般来说，在系统运行时，外设的 I/O 内存资源的物理地址是已知的，由硬件的设计决定。但是 CPU 通常并没有为这些已知的外设 I/O 内存资源的物理地址预定义虚拟地址范围，驱动程序并不能直接通过物理地址访问 I/O 内存资源，而必须将它们映射到核心虚地址空间内（通过页表），然后才能根据映射所得到的核心虚地址范围，通过访内指令访问这些 I/O 内存资源。Linux 在 io.h 头文件中声明了函数 ioremap（），用来将 I/O 内存资源的物理地址映射到核心虚地址空间（3GB－4GB）中，如下所示：

```
void * ioremap(unsigned long phys_addr, unsigned long size,
               unsigned long flags);
```

iounmap 函数用于取消 ioremap（）所做的映射，如下所示：

```
void iounmap(void * addr);
```

到这里应该明白，像 GPMCON（0x7F00 8820 ）这个物理地址是不能直接操控的，必须通过映射到内核的虚拟地址中，才能进行操作。

对 IO 进行操作由很多种方法，先小试牛刀使用 readl()和 writel()这两个函数，这两个函数在：linux-3.6.7/arch/arm/include/asm 的 io.h 中。

```
#define readl(c)     ({ u32 __v = readl_relaxed(c); __iormb(); __v; })
#define writel(v,c)           ({ __iowmb(); writel_relaxed(v,c); })
```

这两个函数是对 32 为寄存器进行操作，先看 readl(c)这个函数：

```
#define readl_relaxed(c)
({ u32 __r = le32_to_cpu( (__force __le32)__raw_readl(c) ); __r; })
```

而

```
#define __raw_readl(a)
( __chk_io_ptr(a), *(volatile unsigned int __force   *)(a) )
```

其实 readl(addr)函数的作用就是从 IO 口地址 addr 中读取数值，而 writel(tmp,addr)即是将 tmp 的数值写入到 IO 口地址 addr 中。

有了之前的基础，现在就可以开始写第一个 LED 驱动程序了。

15.2.1 头文件

驱动程序的头文件有点多，如下所示：

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

moudle.h 包含了大量加载模块需要的函数和符号的定义。

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

init.h 来指定你的初始化和清理函数，例如：module_init(init_function)、module_exit(cleanup_function)。

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

kernel.h 以便使用 printk()等函数。

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

fs.h 包含常用的数据结构，如 struct file 等。

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

uaccess.h 包含 copy_to_user()，copy_from_user()等函数。

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

io.h 包含 inl()，outl()，readl()，writel()等 IO 口操作函数。

```
#include <linux/miscdevice.h>
#include <linux/pci.h>
#include <linux/delay.h>
#include <linux/device.h>
#include <linux/cdev.h>

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

irq.h 中断与并发请求事件。

```
#include <plat/gpio-cfg.h>
#include <mach/gpio.h>
#include <mach/regs-gpio.h>
#include <mach/hardware.h>
#include <mach/map.h>
```

这些 IO 口在内核的虚拟映射地址，涉及 IO 口的操作所必须包含的头文件。

15.2.2 寄存器地址

在前面已经讲过，OK6410 的 LED1~LED4 由 s3c6410 控制芯片的 GPM0~GPM3 相应地控制，但是 Linux 驱动程序不能直接控制寄存器的物理地址，必须通过内核映射的虚拟地址进行控制操作。GPM 管脚的三个寄存器虚拟地址如所示：

```
#define S3C64XX_GPMCON          (S3C64XX_GPM_BASE + 0x00)
#define S3C64XX_GPMDAT          (S3C64XX_GPM_BASE + 0x04)
#define S3C64XX_GPMPUD          (S3C64XX_GPM_BASE + 0x08)
#define S3C64XX_GPM_BASE         S3C64XX_GPIOREG(0x0820)
#define S3C64XX_VA_GPIO          S3C_ADDR_CPU(0x00000000)
```

在 map-base.h 中有这么个定义：

```
#define S3C_ADDR_CPU(x)          S3C_ADDR(0x0050 0000 + (x))
#define S3C_ADDR(x)             (S3C_ADDR_BASE + (x))
#define S3C_ADDR_BASE            0xF600 0000
```

在 regs-gpio.h 中是这么定义：

```
#define S3C64XX_GPIOREG(reg)   (S3C64XX_VA_GPIO + (reg))
#define S3C64XX_GPM_BASE        S3C64XX_GPIOREG(0x0820)
```

那么可以知道 S3C64XX_GPM_BASE 即为 0F650 0820。这个地址即是 GPM 在内核中的虚拟映射地址。

15.2.3 open 函数

open 函数的声明在 fs.h 中，如所示：

```
struct file_operations {
    struct module *owner;
    loff_t (*llseek) (struct file *, loff_t, int);
    ssize_t (*read)   (struct file *, char __user *, size_t, loff_t *);
    ssize_t (*write)   (struct file *, const char __user *, size_t,
loff_t *);
    ssize_t (*aio_read) (struct kiocb *, const struct iovec *, unsigned
long, loff_t);
    ssize_t (*aio_write) (struct kiocb *, const struct iovec *,
unsigned long, loff_t);
    int (*readdir) (struct file *, void *, filldir_t);
    unsigned int (*poll) (struct file *, struct poll_table_struct *);
    long (*unlocked_ioctl) (struct file *, unsigned int, unsigned
long);
    long (*compat_ioctl) (struct file *, unsigned int, unsigned long);
    int (*mmap) (struct file *, struct vm_area_struct *);
    int (*open) (struct inode *, struct file *);
    int (*flush) (struct file *, fl_owner_t id);
    int (*release) (struct inode *, struct file *);
    int (*fsync) (struct file *, loff_t, loff_t, int datasync);
    int (*aio_fsync) (struct kiocb *, int datasync);
    int (*fasync) (int, struct file *, int);
    int (*lock) (struct file *, int, struct file_lock *);
    ssize_t (*sendpage) (struct file *, struct page *, int, size_t,
loff_t *, int);
    unsigned long (*get_unmapped_area)(struct file *, unsigned long,
unsigned long, unsigned long, unsigned long);
    int (*check_flags)(int);
    int (*flock) (struct file *, int, struct file_lock *);
    ssize_t (*splice_write)(struct pipe_inode_info *, struct file *,
loff_t *, size_t, unsigned int);
    ssize_t (*splice_read)(struct file *, loff_t *, struct
pipe_inode_info *, size_t, unsigned int);
    int (*setlease)(struct file *, long, struct file_lock **);
    long (*fallocate)(struct file *file, int mode, loff_t offset,
            loff_t len);
};
```

在这个函数操作中，需要完成的事情是对 GPM 寄存器的模式进行设定：

```
int led_open(struct inode *inode,struct file *file)
{
```

```
        unsigned int tmp;
        tmp = readl(S3C64XX_GPMCON);
        printk("the pre GPMCON is %x \n", tmp);

        tmp = ( (tmp & (~0xffff)) | (0x1111));
        /*向 GPMCON 命令端口写命令字，设置 GPM0-3 为 output 口*/
        writel(tmp, S3C64XX_GPMCON);

        printk("zhuzhaoqi >>> s3c6410_led open... \n");
        return 0;
}
```

15.2.4 read 函数

read 函数的声明同样是在 fs.h 中，如所示：

```
ssize_t (*read)   (struct file *, char __user *, size_t, loff_t *);
```

在这个函数中，需要进行的操作是：

```
static ssize_t led_read(struct file *file,char __user *buf,
                        size_t count,loff_t * f_pos)
{
        unsigned tmp = readl(S3C64XX_GPMDAT);
        int num = copy_to_user(buf,&tmp,count);

        if (num == 0) {
           printk("copy_to_user successfully \n");
        }
        else {
           printk("sorry, copy_to_user failly \n");
        }

        printk("the GPMDAT is %x. \n",tmp);
        return count;
}
```

15.2.5 write 函数

write 函数同样是在 fs.h 这个文件中：

```
ssize_t (*write)(struct file *,const char __user *,size_t, loff_t *);
```

在这个函数中需要实现的是控制 GMP0~GPM3 的输出：

```
static ssize_t led_write(struct file * file,
                         const char __user * buf,
                         size_t count,loff_t * f_pos)
{
```

```
        char kbuf[10];
        unsigned int tmp;
        printk("zhuzhaoqi >>> s3c6410_led write... \n");
        int num = copy_from_user(kbuf,buf,count);

        if (num == 0) {
            printk("copy_from_user successfully \n");
        }
        else {
           printk("sorry, copy_from_user failly \n");
        }

        printk("zhuzhaoqi >>> the kbuf is %c \n",kbuf[0]);
        switch(kbuf[0])
        {
               /* 点亮LED1~LED4 */
               case 0:
                      tmp  = readl(S3C64XX_GPMDAT);
                      tmp |= 0x0f;
                      writel(tmp,S3C64XX_GPMDAT);
                      break;
               /* 熄灭LED1~LED4 */
               case 1:
                      tmp  = readl(S3C64XX_GPMDAT);
                      tmp &= (~0x0f);
                      writel(tmp,S3C64XX_GPMDAT);
                      break;
               default:
                      break;
        }

    return count;
}
```

15.2.6 release 函数

release 函数的声明同样是在 fs.h 中：

```
int (*release) (struct inode *, struct file *);
```

这个函数需要进行的操作：

```
int led_release(struct inode *inode,struct file *file)
{
        printk("zhuzhaoqi >>> s3c6410_led release \n");
        return 0;
}
```

15.2.7 file_operations 结构体

这是字符驱动程序的核心所在，当应用程序操作设备文件时调用 open、read、write 等函数的时候将会调用这个结构体中的相对应函数。

```
struct file_operations led_fops = {
       .owner   = THIS_MODULE,
       .open    = led_open,
       .read    = led_read,
       .write   = led_write,
       .release = led_release,
};
```

15.2.8 模块的加载和卸载

加载程序：

```
int __init led_init(void)
{
       int rc;
       printk("Test led dev \n");
       rc = register_chrdev(LED_MAJOR,"led",&led_fops);

       if (rc < 0)
       {
              printk("register %s char dev error\n","led");
              return -1;
       }

       printk("OK!\n");
       return 0;
}
```

卸载程序：

```
void __exit led_exit(void)
{
       unregister_chrdev(LED_MAJOR,"led");
       printk("module exit\n");
}
```

通过这个两个进行模块的加载和卸载：

```
module_init(led_init);
module_exit(led_exit);
```

将写好的 s3c6410_led.c 这个 LED 驱动程序放入到/drivers/char 这个文件夹下面，打开 Makefile，添加 s3c6410_led.c 这个驱动：

```
obj-m                          += s3c6410_led.o
```

然后再内核的根目录下执行：

```
root@zhuzhaoqi-desktop:/home/zhuzhaoqi/linux-3.6.7# make modules
```

如果没有出错的话，在/drivers/char 文件夹下面就会生成：s3c6410_led.ko 模块。将这个模块添加到根文件系统下面的/lib/modules/3.6.7/文件夹下。

加载模块操作：

```
[YJR@zhuzhaoqi 3.6.7]# insmod s3c6410_led.ko
Test led dev
OK!
```

可以看到时加载成功了。

15.2.9 测试程序

从应用层控制驱动程序：

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int main()
{
   printf("hello led device .");
   char buf[10]={0,1,0,1};
   int fd = open("/dev/s3c6410_led",2,0777);
   if(fd < 0){
      printf("can't open led device");
      return -1;
   }
   printf("open the led device successfully.");

   while(1)
   {
      int num = write(fd,&buf[0],1);
      if( num < 0 ) {
         printf("we set the led failly.");
      }
      else {
         printf("we set the led off");
      }

      sleep(1);
      write(fd,&buf[1],1);
      printf("we set the led on");
```

```
        sleep(1);
    }

    close(fd);
    printf("bye led device .");
    return 0;
}
```

通过这个应用层程序控制 LED 的驱动程序，为这个程序写一个 Makefile：

```
CC = /usr/local/arm/4.4.1/bin/arm-linux-gcc

led_test:led_test.o
        $(CC) -o led_test led_test.o

led_test.o:led_test.c
        $(CC) -c led_test.c

clean :
        rm led_test.o led_test
```

执行 Makefile 之后会生成 led_test 执行文件，将其复制到根文件系统的 usr/bin 文件下。接着建立设备文件：

```
[YJR@zhuzhaoqi /]# mknod dev/s3c6410_led c 240 0
```

然后进行测试：

```
[YJR@zhuzhaoqi bin]# ./led_test
the pre GPMCON is 111111
zhuzhaoqi >>> s3c6410_led open...
zhuzhaoqi >>> s3c6410_led write...
copy_from_user successfully
zhuzhaoqi >>> the kbuf is
```

可以看到 OK6410 的 LED1~LED4 在响应应用层的操作。

15.3 Linux 字符驱动之 LED（方法二）

其实 Linux 字符驱动的写法是多样化，关键在于方法的实用性，可扩展性的优劣。由于 Linux3.6.7 和之前的内核多多少少还是有一些差别的。

方法二和方法一类似，驱动程序如下所示：

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <linux/uaccess.h>
#include <linux/io.h>
#include <linux/miscdevice.h>
```

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

#include <linux/init.h>
#include <linux/delay.h>
#include <linux/device.h>
#include <linux/cdev.h>
#include <linux/gpio.h>
#include <asm/irq.h>
//#include <mach/gpio.h>
#include <mach/regs-gpio.h>
#include <plat/gpio-cfg.h>
#include <mach/hardware.h>
#include <mach/map.h>
```

上面是驱动程序所包含的头文件。

```
#define DEVICE_NAME    "led"
#define LED_MAJOR       240                 /*主设备号*/
```

驱动名称和主设备号。

```
#define LED_ON          0
#define LED_OFF         1
```

LED 的点亮和熄灭。

```
/* out put */
int led_open(struct inode *inode,struct file *file)
{
       unsigned int i;
       /*设置 GPM0-3 为 output*/
       for (i = 0; i < 4; i++) {
          s3c_gpio_cfgpin(S3C64XX_GPM(i),S3C_GPIO_OUTPUT);
          printk( "The GPMCON %x is %x \n",
                  i,s3c_gpio_getcfg(S3C64XX_GPM(i)) );
       }
       printk("zhuzhaoqi >>> s3c6410_led open... \n");
       return 0;
}
```

这里是设置 GPM0~GPM3 为输出模式，这里有所不同的是使用了 s3c_gpio_cfgpin()这个

设置函数。这个函数在下面会详细讲解。

```
/* LED contrl */
static long led_ioctl ( struct file *file, unsigned int cmd,
                        unsigned long argv )
{
    if (argv > 4) {
        return -EINVAL;
    }

    printk("LED ioctl... \n");

    switch(cmd) {

    case LED_ON:
        gpio_set_value(S3C64XX_GPM(argv),0);
        printk("LED on \n");
        printk( "S3C64XX_GPM(i) = %x \n",
                 gpio_get_value(S3C64XX_GPM(argv)) );
        return 0;

    case LED_OFF:
        gpio_set_value(S3C64XX_GPM(argv),1);
        printk("LED off \n");
        printk( "S3C64XX_GPM(i) = %x \n",
                gpio_get_value(S3C64XX_GPM(argv)) );
        return 0;
    default:
        return -EINVAL;
    }
}
```

这里是响应应用层的操作，对 LED1~LED4 进行控制。

```
int led_release(struct inode *inode,struct file *file)
{
       printk("zhuzhaoqi >>> s3c6410_led release \n");
       return 0;
}


struct file_operations led_fops = {
       .owner         = THIS_MODULE,
```

```
        .open           = led_open,
        .unlocked_ioctl = led_ioctl,
        .release        = led_release,
};
```

这是驱动程序的核心控制，Linux3.6.7 没有了 ioctl，这个函数的声明在 fs.h 中有详细说明。

```
int __init led_init(void)
{
        int rc;
        printk("Test led dev \n");
        rc = register_chrdev(LED_MAJOR,"led",&led_fops);

        if (rc < 0)
        {
                printk("register %s char dev error\n","led");
                return -1;
        }

        printk("OK!\n");
        return 0;
}

void __exit led_exit(void)
{
        unregister_chrdev(LED_MAJOR,"led");
        printk("module exit\n");
}

MODULE_LICENSE("GPL");

module_init(led_init);
module_exit(led_exit);
```

这是驱动程序的加载和卸载函数。

这里对上面这个驱动程序再加以详细说明。

```
s3c_gpio_cfgpin(S3C64XX_GPM(i),S3C_GPIO_OUTPUT);
```

这行代码是依次对 GPM0~GPM3 设置为输出模式。函数原型是在 gpio-cfg.h 中：

```
extern int s3c_gpio_cfgpin(unsigned int pin, unsigned int to);
```

内核对这个函数的注释是这样的：s3c_gpio_cfgpin()函数用于改变引脚的 GPIO 功能。参数 pin 是 GPIO 的引脚名称，参数 to 是需要将 GPIO 这个引脚设置成为的功能。

然而，GPIO 的名称在 arch/arm/mach-s3c6400/include/mach/gpio.h 有宏定义：

```
/* S3C64XX GPIO number definitions. */

#define S3C64XX_GPA(_nr)   (S3C64XX_GPIO_A_START + (_nr))
#define S3C64XX_GPB(_nr)   (S3C64XX_GPIO_B_START + (_nr))
#define S3C64XX_GPC(_nr)   (S3C64XX_GPIO_C_START + (_nr))
#define S3C64XX_GPD(_nr)   (S3C64XX_GPIO_D_START + (_nr))
#define S3C64XX_GPE(_nr)   (S3C64XX_GPIO_E_START + (_nr))
#define S3C64XX_GPF(_nr)   (S3C64XX_GPIO_F_START + (_nr))
#define S3C64XX_GPG(_nr)   (S3C64XX_GPIO_G_START + (_nr))
#define S3C64XX_GPH(_nr)   (S3C64XX_GPIO_H_START + (_nr))
#define S3C64XX_GPI(_nr)   (S3C64XX_GPIO_I_START + (_nr))
#define S3C64XX_GPJ(_nr)   (S3C64XX_GPIO_J_START + (_nr))
#define S3C64XX_GPK(_nr)   (S3C64XX_GPIO_K_START + (_nr))
#define S3C64XX_GPL(_nr)   (S3C64XX_GPIO_L_START + (_nr))
#define S3C64XX_GPM(_nr)   (S3C64XX_GPIO_M_START + (_nr))
#define S3C64XX_GPN(_nr)   (S3C64XX_GPIO_N_START + (_nr))
#define S3C64XX_GPO(_nr)   (S3C64XX_GPIO_O_START + (_nr))
#define S3C64XX_GPP(_nr)   (S3C64XX_GPIO_P_START + (_nr))
#define S3C64XX_GPQ(_nr)   (S3C64XX_GPIO_Q_START + (_nr))
```

而 S3C64XX_GPIO_M_START 的定义：

```
enum s3c_gpio_number {
    S3C64XX_GPIO_A_START = 0,
    S3C64XX_GPIO_B_START = S3C64XX_GPIO_NEXT(S3C64XX_GPIO_A),
    S3C64XX_GPIO_C_START = S3C64XX_GPIO_NEXT(S3C64XX_GPIO_B),
    S3C64XX_GPIO_D_START = S3C64XX_GPIO_NEXT(S3C64XX_GPIO_C),
    S3C64XX_GPIO_E_START = S3C64XX_GPIO_NEXT(S3C64XX_GPIO_D),
    S3C64XX_GPIO_F_START = S3C64XX_GPIO_NEXT(S3C64XX_GPIO_E),
    S3C64XX_GPIO_G_START = S3C64XX_GPIO_NEXT(S3C64XX_GPIO_F),
    S3C64XX_GPIO_H_START = S3C64XX_GPIO_NEXT(S3C64XX_GPIO_G),
    S3C64XX_GPIO_I_START = S3C64XX_GPIO_NEXT(S3C64XX_GPIO_H),
    S3C64XX_GPIO_J_START = S3C64XX_GPIO_NEXT(S3C64XX_GPIO_I),
    S3C64XX_GPIO_K_START = S3C64XX_GPIO_NEXT(S3C64XX_GPIO_J),
    S3C64XX_GPIO_L_START = S3C64XX_GPIO_NEXT(S3C64XX_GPIO_K),
    S3C64XX_GPIO_M_START = S3C64XX_GPIO_NEXT(S3C64XX_GPIO_L),
    S3C64XX_GPIO_N_START = S3C64XX_GPIO_NEXT(S3C64XX_GPIO_M),
    S3C64XX_GPIO_O_START = S3C64XX_GPIO_NEXT(S3C64XX_GPIO_N),
    S3C64XX_GPIO_P_START = S3C64XX_GPIO_NEXT(S3C64XX_GPIO_O),
    S3C64XX_GPIO_Q_START = S3C64XX_GPIO_NEXT(S3C64XX_GPIO_P),
};
```

S3C64XX_GPIO_NEXT 的定义：

```
#define S3C64XX_GPIO_NEXT(__gpio) \
    ((__gpio##_START) + (__gpio##_NR) + CONFIG_S3C_GPIO_SPACE + 1)
```

也就是通过这个设置，可以很方便得选择想要的任何一个 GPIO 口进行操作。

GPIO 功能设置在 gpio-cfg.h 有：

```
#define S3C_GPIO_SPECIAL_MARK (0xfffffff0)
#define S3C_GPIO_SPECIAL(x)   (S3C_GPIO_SPECIAL_MARK | (x))

/* Defines for generic pin configurations */
#define S3C_GPIO_INPUT  (S3C_GPIO_SPECIAL(0))
#define S3C_GPIO_OUTPUT (S3C_GPIO_SPECIAL(1))
#define S3C_GPIO_SFN(x) (S3C_GPIO_SPECIAL(x))
```

也就是说，GPIO 的引脚功能有输入、输出、和你想要的任何功能设置，S3C_GPIO_SFN(x)这个函数即是通过设定 x 的值，实现任何存在功能的设置。如果要设置 GPM0~GPM3 为输出功能，则：

```
for (i = 0; i < 4; i++) {
    s3c_gpio_cfgpin(S3C64XX_GPM(i),S3C_GPIO_OUTPUT);
}
```

这样对于设置比较简洁实用。

```
gpio_set_value(S3C64XX_GPM(argv),1);
```

这行代码是设定 GMP(argv)输出为 1。这个函数的原型在 include/linux/gpio.h 中：

```
static inline void gpio_set_value(unsigned int gpio, int value)
{
    __gpio_set_value(gpio, value);
}
```

这里还需要特别注意的是：

```
static long led_ioctl ( struct file *file, unsigned int cmd,
                        unsigned long argv )
```

在 fs.h 的结构体中是这样声明的：

```
long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);
```

和原来老版本的内核参数有所变化。

应用程序如下所示：

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
```

```
#define   LED_ON    0
#define   LED_OFF   1

/*
 * LED contrl info
 */
void usage(char *exename)
{
   printf("Usage: \n");
   printf("    %s <led_number> <on/off> \n", exename);
   printf("    led_number = 1, 2, 3 or 4 \n");
}

/*
 *  应用程序主函数
 */
int main(int argc, char *argv[])
{
   unsigned int led_number;

   if (argc != 3) {
      goto err;
   }

   int fd = open("/dev/led",2,0777);
   if (fd < 0) {
      printf("Can't open /dev/led \n");
      return -1;
   }
   printf("open /dev/led ok ... \n");

   led_number = strtoul(argv[1], 0, 0) - 1;
   if (led_number > 3) {
      goto err;
   }

   /* LED ON */
   if (!strcmp(argv[2], "on")) {
      ioctl(fd, LED_ON, led_number);
   }
   /* LED OFF */
   else if (!strcmp(argv[2], "off")) {
      ioctl(fd, LED_OFF, led_number);
   }
```

```
    else {
        goto err;
    }

    close(fd);
    return 0;

err:
    if (fd > 0) {
        close(fd);
    }
    usage(argv[0]);
    return -1;

}
```

应用程序比较简单，不加详述。编译之后放入到根文件系统/usr/bin 文件夹下。

加载 led.ko：

```
[YJR@zhuzhaoqi 3.6.7]# insmod led.ko
Test led dev
OK!
```

建立设备文件：

```
[YJR@zhuzhaoqi /]# mknod /dev/led c 240 0
```

执行操作：

```
[YJR@zhuzhaoqi bin]# ./ledapp 1 on
The GPMCON 0 is fffffff1
The GPMCON 1 is fffffff1
The GPMCON 2 is fffffff1
The GPMCON 3 is fffffff1
zhuzhaoqi >>> s3c6410_led open...
LED ioctl...
LED on
S3C64XX_GPM(i) = 0
zhuzhaoqi >>> s3c6410_led release
open /dev/led ok ...
```

此时你应该可以看到开发板的 LED1 点亮。

E-mail : jxlgzzq@163.com

第 16 章 Linux 设备驱动之 DS18B20

（待更新 •••）

第 17 章 Linux 设备驱动之 ADC

s3c6410 控制芯片自带有 4 路独立 AD 转换通道，如图 17.1 所示。

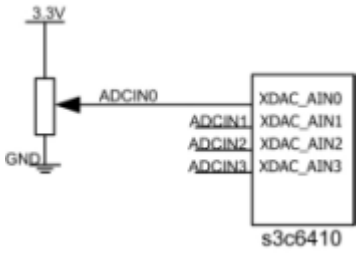


图 17.1 AD 转换连接图

17.1 ADC 控制寄存器简介

ADCCON 为 ADC 控制寄存器，地址为：0x7E00 B0000。以 0x7E00 B000 为基地址。ADCCON 的复位值为：0x3FC4，即为：0011 1111 1100 0100。

```
#define S3C_ADCREG(x)          (x)
#define S3C_ADCCON             S3C_ADCREG(0x00)
```

ADCCON 控制寄存器有 16 位数据进行控制其功能的实现。

ADCCON[0]：ENABLE_START，A/D 转换开始启用。如果 READ_START 启用，这个值是无效的。ENABLE_START = 0，无行动；ENABLE_START = 1，A/D 转换开始和该位被清理后开启。复位值为 0，无行动。

```
#define S3C_ADCCON_NO_ENABLE_START      (0<<0)
#define S3C_ADCCON_ENABLE_START         (1<<0)
```

ADCCON[1]：READ_START，A/D 转换开始读取。READ_START = 0，禁用开始读操作；READ_START = 1，启动开始读操作。复位值为 0，禁用开始读操作。

```
#define S3C_ADCCON_NO_READ_START        (0<<1)
#define S3C_ADCCON_READ_START           (1<<1)
```

ADCCON[2]：STDBM，待机模式选择。STDBM = 0，正常运作模式；STDBM = 1，待机模式。复位值为 1，待机模式。

```
#define S3C_ADCCON_RUN          (0<<2)
#define S3C_ADCCON_STDBM        (1<<2)
```

ADCCON[5:3]：SEL_MUX，模拟输入通道选择。SEL_MUX = 000，AIN0；SEL_MUX = 001，AIN1；SEL_MUX = 010，AIN2；SEL_MUX = 011，AIN3；SEL_MUX = 100，YM; SEL_MUX = 101，YP；SEL_MUX = 110，XM；SEL_MUX = 111，XP。复位值为 000，选用 AIN0 通道。

```
#define S3C_ADCCON_RESSEL_10BIT_1    (0x0<<3)
#define S3C_ADCCON_RESSEL_12BIT_1    (0x1<<3)
#define S3C_ADCCON_MUXMASK           (0x7<<3)
#define S3C_ADCCON_SELMUX(x)         (((x)&0x7)<<3) //任意通道的选择
```

ADCCON[13:6]：PRSCVL，ADC 预定标器值 0xFF。数据值：5～255。复位值为 1111

1111，即为 0xFF。

```
#define S3C_ADCCON_PRSCVL(x)      (((x)&0xFF)<<6) // 任意值设定
#define S3C_ADCCON_PRSCVLMASK     (0xFF<<6)  //复位值
```

ADCCON[14]：PRSCEN，ADC 预定标器启动。PRSCEN = 0，禁用；PRSCEN = 0，启用。复位值为 0，禁用 ADC 预定标器。

```
#define S3C_ADCCON_NO_PRSCEN      (0<<14)
#define S3C_ADCCON_PRSCEN         (1<<14)
```

ADCCON[15]：ECFLG，转换的结束标记（只读）。ECFLG = 0，A/D 转换的过程中; ECFLG = 1，A/D 转换结束。复位值为 0，A/D 转换的过程中。

```
#define S3C_ADCCON_ECFLG_ING      (0<<15)
#define S3C_ADCCON_ECFLG          (1<<15)
```

ADCDAT0 寄存器为 ADC 的数据转换寄存器。地址为：0x7E00 B00C。

ADCDAT0[9:0]：XPDATA，X 坐标的数据转换（包括正常的 ADC 的转换数据值）。数据值： 0x000～0x3FF。

ADCDAT0[11:10]：保留。当启用 12 位 AD 时作为转换数据值使用。

```
#define S3C_ADCDAT0_XPDATA_MASK          (0x03FF)
#define S3C_ADCDAT0_XPDATA_MASK_12BIT    (0x0FFF)
```

如果仅仅是 ADC 功能，上面寄存器足够。 LCD 触摸屏的寄存器在后续章节详细讲解。

17.2 Linux 设备驱动 ADC 程序

有了上一节寄存器的知识，接下来就可以开始写 ADC 驱动程序。adc.c 驱动程序需要添加的头文件：

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/slab.h>
#include <linux/input.h>
#include <linux/init.h>
#include <linux/errno.h>
#include <linux/serio.h>
#include <linux/delay.h>
#include <linux/clk.h>
#include <linux/sched.h>
#include <linux/cdev.h>
#include <linux/miscdevice.h>

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

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

#include <plat/regs-adc.h>
```

ADC 映射地址的宏定义：

```
static void __iomem * base_addr;
static struct clk *adc_clock;

#define   S3C_ADCREG(x)      (x)
#define   S3C_ADCCON         S3C_ADCREG(0x00)
#define   S3C_ADCDAT0        S3C_ADCREG(0x0C)

#define   __ADCREG(name)    (*(unsigned long int *)(base_addr + name))

/* ADC contrl */
#define   ADCCON            __ADCREG(S3C_ADCCON)
/* read the ADdata */
#define   ADCDAT0           __ADCREG(S3C_ADCCON)
```

ADCCON 寄存器和 ADCDAT0 寄存器的位功能宏定义：

```
/* The set of ADCCON */
#define   S3C_ADCCON_ENABLE_START         (1 << 0)
#define   S3C_ADCCON_READ_START           (1 << 1)
#define   S3C_ADCCON_RUN                  (0 << 2)
#define   S3C_ADCCON_STDBM                (1 << 2)
#define   S3C_ADCCON_SELMUX(x)             ( ((x)&0x7) << 3 )
#define   S3C_ADCCON_PRSCVL(x)             ( ((x)&0xFF) << 6 )
#define   S3C_ADCCON_PRSCEN               (1 << 14)
#define   S3C_ADCCON_ECFLG                (1 << 15)

/* The set of ADCDAT0 */
#define   S3C_ADCDAT0_XPDATA_MASK          (0x03FF)
#define   S3C_ADCDAT0_XPDATA_MASK_12BIT     (0x0FFF)
```

ADC 进行初始化，OK6410 的 AD 电压采样选用的是 AIN0 通道，初始化阶段需要完成的事情为：A/D 转换开始和该位被清理后开启、正常运作模式、模拟输入通道选择 AIN0、ADC 预定标器值 0xFF、ADC 预定标器启动。

```
/*
 *  AIN0 init
 */
static int adc_init(void)
```

```
{

    ADCCON  =  S3C_ADCCON_PRSCEN | S3C_ADCCON_PRSCVL(0xFF) | \
               S3C_ADCCON_SELMUX(0x00) | S3C_ADCCON_RUN;
    ADCCON |= S3C_ADCCON_ENABLE_START;

    return 0;
}
```

打开 adc 的驱动：

```
/*
 *  open dev
 */
static int adc_open(struct inode *inode, struct file *filp)
{
    adc_init();
    return 0;
}

/*
 *  release dev
 */
static int adc_release(struct inode *inode,struct file *filp)
{
    return 0;
}
```

读取 ADC 采用的数据：

```
/*
 * adc_read
 */
static ssize_t adc_read(struct file *filp, char __user *buff,
                        size_t size, loff_t *ppos)
{
    ADCCON |= S3C_ADCCON_ENABLE_START;
    /* check the adc Enabled ,The [0] is low*/
    while(ADCCON & 0x01);
    /* check adc change end */
    while(!(ADCCON & 0x8000));

    /* return the data of adc */
    return (ADCDAT0 & S3C_ADCDAT0_XPDATA_MASK);
}
```

ADC 驱动程序的核心控制：

```
static struct file_operations dev_fops =
{
    .owner   = THIS_MODULE,
    .open    = adc_open,
    .release = adc_release,
    .read    = adc_read,
};


static struct miscdevice misc =
{
    .minor = MISC_DYNAMIC_MINOR,
    .name  = DEVICE_NAME,
    .fops  = &dev_fops,
};
```

加载 insmod 驱动程序，这里使用到了 ioremap()函数，在内核驱动程序的初始化阶段，通过 ioremap()函数将物理地址映射到内核虚拟空间；在驱动程序的 mmap 系统调用中，使用 remap_page_range()函数将该块 ROM 映射到用户虚拟空间。这样内核空间和用户空间都能访问这段被映射后的虚拟地址。

ioremap()宏定义在 asm/io.h 内：

```
    #define ioremap(cookie,size)           __ioremap(cookie,size,0)
```

__ioremap 函数原型为(arm/mm/ioremap.c)：

```
    void __iomem * __ioremap(unsigned long phys_addr, size_t size,
                             unsigned long flags);
```

phys_addr：要映射的起始的 IO 地址；

size：要映射的空间的大小；

flags：要映射的 IO 空间和权限有关的标志。

该函数返回映射后的内核虚拟地址(3G-4G)，接着便可以通过读写该返回的内核虚拟地址去访问之这段 I/O 内存资源。

```
base_addr = ioremap(SAMSUNG_PA_ADC,0X20);
```

这行代码即是将 SAMSUNG_PA_ADC(0x7E00 B000)映射到内核，返回内核的虚拟地址给 base_addr。

clk_get(NULL,"adc")可以获得 adc 时钟，每一个外设都有自己的工作频率，PRSCVL 是 A/D 转换器时钟的预分频功能时 A/D 时钟的计算公式，A/D 时钟 =PCLK / (PRSCVL+1)。

注意：AD 时钟最大为 2.5MHZ 并且应该小于 PCLK 的 1/5。

```
    adc_clock = clk_get(NULL,"adc");
```

即为获取 adc 的工作时钟频率。

```
ret = misc_register(&misc);
```

创建杂项设备节点。这里使用到了杂项设备，杂项设备也是在嵌入式系统中用得比较多的一种设备驱动。在 Linux 内核的 include/linux 目录下有 Miscdevice.h 文件，要把自己定义的 misc device 从设备定义在这里。其实是因为这些字符设备不符合预先确定的字符设备范畴，所有这些设备采用主编号 10 ，一起归于 misc device，其实 misc_register 就是用主标号 10 调用 register_chrdev()的。也就是说，misc 设备其实也就是特殊的字符设备，可自动生成设备节点。

加载 insmod 驱动程序如下所示：

```
static int __init dev_init()
{
    int ret;

    /* Address Mapping */
    base_addr = ioremap(SAMSUNG_PA_ADC,0X20);
    if(base_addr == NULL)
    {
        printk(KERN_ERR"failed to remap \n");
        return -ENOMEM;
    }

    /* Enabld acd clock */
    adc_clock = clk_get(NULL,"adc");
    if(!adc_clock)
    {
        printk(KERN_ERR"failed to get adc clock \n");
        return -ENOENT;
    }
    clk_enable(adc_clock);

    ret = misc_register(&misc);
    printk("dev_init return ret: %d \n", ret);

    return ret;
}
```

卸载 rmmod 驱动程序：

```
static void __exit dev_exit()
{
    iounmap(base_addr);

    /* disable ths adc clock */
    if(adc_clock)
    {
```

```
        clk_disable(adc_clock);
        clk_put(adc_clock);
        adc_clock = NULL;
    }

    misc_deregister(&misc);
}
```

调用加载和卸载程序：

```
MODULE_LICENSE("GPL");
MODULE_AUTHOR("zhuzhaoqi jxlgzzq@163.com");

module_init(dev_init);
module_exit(dev_exit);
```

将 zzqadc.c 驱动程序添加到/driver/char 文件夹下，在 Makefile 中添加：

```
obj-m                          += zzqadc.o
```

回到根目录下：

```
/home/zhuzhaoqi/Linux/linux-3.6.7# make modules
```

将/driver/char 文件夹下生成的 zzqadc.ko 放入到文件系统的/lib/module/3.6.7 文件夹下。

编写应用程序：

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

int main()
{
    int fp,adc_data,i;
    fp = open("/dev/zzqadc",O_RDWR);

    if (fp < 0)
    {
        printf("open failed! \n");
    }
    printf("opened ... \n");

    for (i = 0; i < 100; i++)
    {
        adc_data = read(fp,NULL,0);
        printf("Begin the NO. %d test... \n",i);
        printf("adc_data = %d \n",adc_data);
```

```
        printf("The Value = %f V \n" , ( (float)adc_data )* 3.3 / 1024);
        printf("End the NO. %d test ..... \n \n",i);

        sleep(1);
    }

    close(fp);
    return 0;
}
```

为应用程序编写 Makefile：

```
CC = /usr/local/arm/4.4.1/bin/arm-linux-gcc

zzqadcapp:zzqadcapp.o
        $(CC) -o zzqadcapp zzqadcapp.o

zzqadcapp.o:zzqadcapp.c
        $(CC) -c zzqadcapp.c

clean :
        rm zzqadcapp.o zzqadcapp
```

将生成的 zzqadcapp 应用文件放入到跟文件系统/usr/bin 文件夹下。

将 zzqadc.ko 加载到根文件系统：

```
[YJR@zhuzhaoqi 3.6.7]# insmod zzqadc.ko
dev_init return ret: 0

[YJR@zhuzhaoqi /]# ls -l /dev/zzqadc
crw-rw----    1 0       0         10,  60 Jan  1 00:09 /dev/zzqadc
```

这说明加载成功。

执行 zzqadcapp 文件：

```
[YJR@zhuzhaoqi bin]# ./zzqadcapp
......
......
Begin the NO. 4 test...
adc_data = 379
The Value = 1.221387 V
End the NO. 4 text ......

Begin the NO. 5 test...
adc_data = 371
```

```
The Value = 1.195605 V
End the NO. 5 text ......

Begin the NO. 6 test...
adc_data = 368
The Value = 1.185937 V
End the NO. 6 text ......

Begin the NO. 7 test...
adc_data = 359
The Value = 1.156934 V
End the NO. 7 text ......

Begin the NO. 8 test...
adc_data = 350
The Value = 1.127930 V
End the NO. 8 text ......
```

第 18 章 Linux3.6.7 驱动之常见问题

18.1 模块许可证声明

许可证（LICENSE）：

```
[YJR@zhuzhaoqi 3.6.7]# insmod s3c6410_led.ko
s3c6410_led: module license 'unspecified' taints kernel.
Disabling lock debugging due to kernel taint
```

这里告诉我们没有声明 LICENSE，模块被加载时，给处理内核被污染(kernel taint)的警告，在驱动程序中加上：

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

18.2 卸载驱动模块

驱动模块被加载时，最好先在 lib/modules 文件夹下面建立一个文件夹：

```
[YJR@zhuzhaoqi modules]# mkdir $(uname -r)
[YJR@zhuzhaoqi modules]# ls
3.6.7
```

将需要被加载的模块放在这个文件夹下，这样的话，不会导致加载和卸载的一些小问题出现。

卸载不需要加后缀：

```
[YJR@zhuzhaoqi 3.6.7]# rmmod s3c6410_led.ko
[YJR@zhuzhaoqi 3.6.7]# ls
s3c6410_led.ko
[YJR@zhuzhaoqi 3.6.7]# lsmod
s3c6410_led 1855 0 - Live 0xbf000000 (P)
[YJR@zhuzhaoqi 3.6.7]# rmmod s3c6410_led
module exit
[YJR@zhuzhaoqi 3.6.7]# ls
s3c6410_led.ko
[YJR@zhuzhaoqi 3.6.7]# lsmod
```

18.3 段错误

在运行驱动模块的时候，对于新手很容易遇到段错误情况：

```
[YJR@zhuzhaoqi /]# cd usr/bin/
[YJR@zhuzhaoqi bin]# ls
led_test
[YJR@zhuzhaoqi bin]# ./led_test
Unable to handle kernel NULL pointer dereference at virtual address
00000000
pgd = cfb6c000
[00000000] *pgd=5fb20831, *pte=00000000, *ppte=00000000
```

```
Internal error: Oops: 17 [#1] ARM
Modules linked in: s3c6410_led [last unloaded: s3c6410_led]
CPU: 0    Tainted: P           (3.6.7 #1)
PC is at led_open+0x8/0x54 [s3c6410_led]
LR is at chrdev_open+0x120/0x140
pc : [<bf004020>]    lr : [<c00883f0>]    psr: a0000013
sp : cfb5bde8  ip : 22222222  fp : cfb5bf78
r10: cf6d8c00  r9 : cfb5a000  r8 : 00000000
r7 : 00000000  r6 : cfadf640  r5 : cfb18be0  r4 : 00000000
r3 : bf004018  r2 : 00000003  r1 : cfb18be0  r0 : cf6d8b28
Flags: NzCv  IRQs on  FIQs on  Mode SVC_32  ISA ARM  Segment user
Control: 00c5387d  Table: 5fb6c008  DAC: 00000015
Process led_test (pid: 997, stack limit = 0xcfb5a268)
Stack: (0xcfb5bde8 to 0xcfb5c000)
bde0:                cf6d8b28 c00883f0 cf6d8b28 00000000 cfb18be0
cf6d8b28
be00: cf6d8b78 c00882d0 cfb18be8 c0083aa4 cf87d8c0 cfb5bec8 00000000
cf6d8b78
be20: cfb5bec0 cfb18be0 cfb5bec8 c0083c2c cfb5bf00 c00920c4 cf408000
cfb0b005
be40: cfb5a000 cfb5be58 cfb5be74 00000000 00000002 00000000 8e7967a3
00000000
be60: cfb0b005 cf406ab0 00000000 00000026 cfaf8bf0 cf406ab0 cf6d8b28
cf408000
be80: cf9a7580 cfb5bf00 cfb18be0 cfb5a000 cfb5bf78 00000000 cfb5a000
cfb0b000
bea0: bec8ed2c c0092364 cfb5bec8 cfb0b000 cfaf1748 cf87d8c0 cfaf1700
80000007
bec0: cfaf8bf0 cf6dbda8 00000000 00000000 60000113 cfb5bf78 00000001
ffffff9c
bee0: cfb5bf00 cfb0b000 cfb5a000 00000000 bec8ed2c c00927a8 00000041
00000000
bf00: cfaf8bf0 cf6dbda8 8e7967a3 0000000b cfb0b005 c006dbb0 cf80e510
cf401720
bf20: cf6d8b28 00000101 00000000 00000000 00000000 cfafee64 00000000
00000002
bf40: cfafee48 c009c8cc 00000002 00000002 ffffff9c cfb0b000 00000003
ffffff9c
bf60: cfb0b000 00000001 c000dc48 c0083770 00000000 00000022 00000002
00000000
bf80: 00000026 00000100 00000022 000084bc 00000140 00000000 00000005
c000dc48
bfa0: 00000000 c000dac0 000084bc 00000140 000086e4 00000002 000001ff
00000001
```

```
bfc0: 000084bc 00000140 00000000 00000005 00000000 00000000 b6f11b60
bec8ed2c
bfe0: 00000000 bec8ed10 0000851c b6e9427c 60000010 000086e4 5fffe821
5fffec21
[<bf004020>] (led_open+0x8/0x54 [s3c6410_led]) from [<c00883f0>]
(chrdev_open+0x120/0x140)
[<c00883f0>] (chrdev_open+0x120/0x140) from [<c0083aa4>]
(do_dentry_open+0x18c/0x248)
[<c0083aa4>] (do_dentry_open+0x18c/0x248) from [<c0083c2c>]
(finish_open+0x40/0x54)
[<c0083c2c>] (finish_open+0x40/0x54) from [<c00920c4>]
(do_last+0x8f0/0xae4)
[<c00920c4>] (do_last+0x8f0/0xae4) from [<c0092364>]
(path_openat+0xac/0x410)
[<c0092364>] (path_openat+0xac/0x410) from [<c00927a8>]
(do_filp_open+0x30/0x7c)
[<c00927a8>] (do_filp_open+0x30/0x7c) from [<c0083770>]
(do_sys_open+0xe4/0x180)
[<c0083770>] (do_sys_open+0xe4/0x180) from [<c000dac0>]
(ret_fast_syscall+0x0/0x30)
Code: e8bd8008 bf004238 e92d4010 e3a04000 (e5941000)
---[ end trace c010485fde3219c3 ]---
Segmentation fault
```

遇到这种错误，寻找会比较麻烦，但不要畏惧，一层一层追寻。

```
Unable to handle kernel NULL pointer dereference at virtual address
00000000
```

无法处理内核空指针引用虚拟地址 00000000，初步判断可能是驱动模块中出现了指针错误，来源还不清楚，待查。

```
Internal error: Oops: 17 [#1] ARM
```

内部错误，在这里 Oops 信息序号为 1 号。

```
Modules linked in: s3c6410_led [last unloaded: s3c6410_led]
```

这里提示出错的模块是 s3c6410_led。

```
PC is at led_open+0x8/0x54 [s3c6410_led]
LR is at chrdev_open+0x120/0x140
pc : [<bf004020>]    lr : [<c00883f0>]    psr: a0000013
sp : cfb5bde8  ip : 22222222  fp : cfb5bf78
r10: cf6d8c00  r9 : cfb5a000  r8 : 00000000
```

```
r7 : 00000000  r6 : cfadf640  r5 : cfb18be0  r4 : 00000000
r3 : bf004018  r2 : 00000003  r1 : cfb18be0  r0 : cf6d8b28
```

这里输出出现错误时，各个寄存器的值。

```
Process led_test (pid: 997, stack limit = 0xcfb5a268)
Stack: (0xcfb5bde8 to 0xcfb5c000)
```

当前进程 led_text，pid = 997。这告诉我们，出现错误时的当前进程是 led_text。那么错误的出现多多少少和这个进程有着关系。待查。

```
Code: e8bd8008 bf004238 e92d4010 e3a04000 (e5941000)
```

出错指令附近指令的机器码。

```
[<bf004020>] (led_open+0x8/0x54 [s3c6410_led]) from [<c00883f0>]
(chrdev_open+0x120/0x140)
```

从这个栈回溯信息可知，错误时发生在 s3c6410_led.c 的 led_open 这个函数中。追寻到这个函数。

```
int led_open(struct inode *inode,struct file *file)
{
       unsigned tmp;
       tmp = inl(S3C64XX_GPMCON);
       tmp = inl(S3C64XX_GPMDAT);
       outl(0x00111111, S3C64XX_GPMCON);
       return 0;
}
```

第 19 章 QT 移植之搭建编译环境

19.1 tslib 的配置

首先安装 autoconf、automake、libtool 这三个安装包。

```
root@zhuzhaoqi-desktop:/home/zhuzhaoqi#apt-get install autoconf
root@zhuzhaoqi-desktop:/home/zhuzhaoqi#apt-get install automake
root@zhuzhaoqi-desktop:/home/zhuzhaoqi#apt-get install libtool
```

将 tslib.tar.gz 包放在虚拟机中，使用

```
tar zxvf tslib.tar.gz
```

解压缩得到 tslib--src.tar.gz。再次解压得到 tslib。

进入 tslib，设置环境变量：

```
/home/zhuzhaoqi/tslib/tslib# export
PATH=/usr/local/arm/4.4.1/bin:$PATH
/home/zhuzhaoqi/tslib/tslib# export TOOLCHAIN=/usr/local/arm/4.4.1
/home/zhuzhaoqi/tslib/tslib# export TB_CC_PREFIX=arm-linux-
/home/zhuzhaoqi/tslib/tslib# export
PKG_CONFIG_PREFIX=$TOOLCHAIN/arm-linux-
```

运行脚本：

```
/home/zhuzhaoqi/tslib/tslib#./autogen.sh
/home/zhuzhaoqi/tslib/tslib#echo
"ac_cv_func_malloc_0_nonnull=yes" >arm-linux.cache
```

配置安装参数：

```
/home/zhuzhaoqi/tslib/tslib#./configure --host=arm-linux-
--cache-file=arm-linux.cache --enable-inputapi=no
PLUGIN_DIR=/usr/local/arm/tslib/build -host=arm-linux
--cache-file=arm-linux.cache 2>&1 | tee conf_log
```

编译：

```
/home/zhuzhaoqi/tslib/tslib#make 2>&1 | tee make_log
/home/zhuzhaoqi/tslib/tslib#make install
```

编译完成之后，进入 usr/local/arm 下查看是否存在 tslib：

```
root@zhuzhaoqi-desktop:/usr/local/arm# ls
4.3.2  4.4.1  tslib
```

进入 tslib/etc

```
root@zhuzhaoqi-desktop:/usr/local/arm/tslib/build/etc# ls
ts.conf
```

修改 ts.conf:

```
# Uncomment if you wish to use the linux input layer event interface
module_raw input
```

并将 tslib 放到开发板的根文件系统的 usr/local 下：

```
[YJR@zhuzhaoqi /]# cd usr/local/
[YJR@zhuzhaoqi local]# ls
tslib
```

19.2 编译 QT4.4.3

在虚拟机中建一个文件夹 qt4.4.3，将 QT4.4.3 源码拷贝这个文件夹下：

```
root@zhuzhaoqi-desktop:/home/zhuzhaoqi/qt4.4.3# ls
ARM-qt-extended-opensource-src-4.4.3.tar.gz
```

解压缩后得到 build 编译脚本，在编译之前得安装：

```
/home/zhuzhaoqi/qt4.4.3# apt-get install libx11-dev libxext-dev
libxtst-dev
```

编译：

```
/home/zhuzhaoqi/qt4.4.3#./build
```

……

期间大概 3 小时。

……

编译完成之后得到：

```
root@zhuzhaoqi-desktop:/home/zhuzhaoqi/qt4.4.3# ls
ARM-qt-extended-opensource-src-4.4.3.tar.gz
qt-extended-opensource-src-4.4.3.tar.gz
build                                   qtopia4.4.3makelog
builddir                                qtopiaconfig.log
qt-extended-4.4.3
```

进入 builddir：

```
root@zhuzhaoqi-desktop:/home/zhuzhaoqi/qt4.4.3/builddir# ls
bin          config.tests  image    modules         qtopiacore  src
config.cache   etc          include   qbuild          scripts
tests
config.status  examples     Makefile  qbuild.solution  sdk
```

将 image 复制到开发板的根文件系统的 opt 下，并命名为 Qtopia4.4.3：

```
[YJR@zhuzhaoqi /opt]# ls
Qtopia4.4.3  sdk
```

编写 qtopia4 脚本，放在根文件系统的 bin 目录下：

```
#!/bin/sh
export TSLIB_ROOT=/usr/local/tslib
export TSLIB_TSDEVICE=/dev/input/event1
export TSLIB_TSEVENTTYPE=H3600
export TSLIB_CONFFILE=/usr/local/tslib/etc/ts.conf
export TSLIB_PLUGINDIR=/usr/local/tslib/lib/ts
```

```
export TSLIB_CALIBFILE=/etc/pointercal
export TSLIB_PLUGINDIR=$TSLIB_ROOT/lib/ts
export TSLIB_CONSOLEDEVICE=none
export TSLIB_FBDEVICE=/dev/fb0

export QWS_MOUSE_PROTO=Tslib:/dev/input/event1
#export QWS_MOUSE_PROTO=TPanel:/dev/input/event1
#export QWS_MOUSE_PROTO="Tslib:/dev/input/event1
MouseMan:/dev/input/mice"
#export QWS_MOUSE_PROTO=MouseMan:/dev/input/mice
export QWS_KEYBOARD="TTY:/dev/tty1"

if [ -c /dev/input/event1 ]; then
       export QWS_MOUSE_PROTO="Tslib:${TSLIB_TSDEVICE}"
       if [ -e /etc/pointercal -a ! -s /etc/pointercal ] ; then
              rm /etc/pointercal
       fi
else
       export QWS_MOUSE_PROTO="MouseMan:/dev/input/mice"
       >/etc/pointercal
fi
export QTDIR=/opt/Qtopia4.4.3
export QPEDIR=/opt/Qtopia4.4.3
export PATH=$QTDIR/bin:$PATH
export QT_PLUGIN_PATH=$QTDIR/plugins:$QTDIR/qt_plugins/
export QT_QWS_FONTDIR=$QPEDIR/lib/fonts/
export
LD_LIBRARY_PATH=$QTDIR/plugins/qtopialmigrate/:$QTDIR/qt_plugins/
imageformats/:$QTDIR/lib:/usr/local/tslib/lib:$LD_LIBRARY_PATH

FB_SIZE=$(cat /sys/class/graphics/fb0/virtual_size)


#export QWS_DISPLAY="LinuxFb:mmWidth76:mmHeight44:1"
case "$FB_SIZE" in
800,480)
export QWS_DISPLAY="LinuxFb:mmWidth91:mmHeight53:1"
;;
480,272)
export QWS_DISPLAY="LinuxFb:mmWidth76:mmHeight44:1"
;;
*)
export QWS_DISPLAY="LinuxFb:mmWidth91:mmHeight53:1"
;;
```

```
esac
export HOME=/root/Qtopia4Home

if [ ! -e $HOME ]; then mkdir -p $HOME; fi
if [ ! -e /usr/share ]; then mkdir /usr/share; fi
if [ ! -e $HOME/Settings/Trolltech/qpe.conf ]; then
   mkdir -p $HOME/Settings/Trolltech/
   cp $QPEDIR/etc/default/Trolltech/qpe.conf
$HOME/Settings/Trolltech/qpe.conf -f
fi

export LANG="en_US"
export QTOPIA_PHONE_DUMMY=1

exec $QPEDIR/bin/qpe 1>/dev/null 2>/dev/null
```

如果 rcS 中没有启动 QT，则要加以修改。

启动开发板，仍然进入不了 QT 界面，原因在于内核配置。请看 19.3 详解。

19.3 QT 启动错误

```
Cannot create semaphore /tmp/qtembedded-unknown/QtEmbedded-0 'd'
Error 38 Function not implemented
Cannot get display lock
```

内核配置没有选择，内核配置应该配上“system V IPC”。

修改内核配置之后可以启动 QT 界面，但是仍然存在问题：

```
[YJR@zhuzhaoqi bin]# ./qpe
Warning: Display size not set.  Using default DPI
language message - en_US
loading /opt/Qtopia4.4.3/i18n/en_US/qt.qm
loading /opt/Qtopia4.4.3/i18n/en_US/qpe.qm
loading /opt/Qtopia4.4.3/i18n/en_US/libqtopia.qm
loading /opt/Qtopia4.4.3/i18n/en_US/systemtime.qm
```

卡在这里就不再启动了。

而单独启动应用程序，可以启动，但是触摸屏失效：

```
ApplicationLayer: Unable to connect to server, application layer will
be disabled.
```

追其原因，在于 Linux 内核中没有启动 LCD 驱动，现在返回 Linux，添加 LCD 驱动。进入 Linux 内核：

```
root@zhuzhaoqi-desktop:/home/zhuzhaoqi/Linux/linux-3.6.7/arch/arm
/mach-s3c64xx# vim mach-ok6410.c
```

做如下修改：

```
//#include <plat/ts.h>
#include <mach/ts.h>
```

在飞凌提供的 Linux 源码中找到 ts.h 和 dev-ts.c 文件，拷贝：

```
zhuzhaoqi@zhuzhaoqi-desktop:/mnt/hgfs/zhuzhaoqi$ cp ts.h
/home/zhuzhaoqi/Linux/linux-3.6.7/arch/arm/mach-s3c64xx/include/m
ach/
zhuzhaoqi@zhuzhaoqi-desktop:/mnt/hgfs/zhuzhaoqi$ cp dev-ts.c
/home/zhuzhaoqi/Linux/linux-3.6.7/arch/arm/mach-s3c64xx/
```

并且在 dev-ts.c 中添加：

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

在/home/zhuzhaoqi/Linux/linux-3.6.7/arch/arm/mach-s3c64xx/Makefile 中添加：

```
#lcd
obj-$(CONFIG_TOUCHSCREEN_S3C)          += dev-ts.o
```

进入 root@zhuzhaoqi-desktop:/home/zhuzhaoqi/Linux/linux-3.6.7/arch/arm/mach-s3c64xx# vim mach-ok6410.c，在 static void __init ok6410_machine_init(void)中修改：

```
//      s3c24xx_ts_set_platdata(NULL);
       s3c_ts_set_platdata(&s3c_ts_platform);
```

并且添加：

```
static struct s3c_ts_mach_info s3c_ts_platform __initdata = {
    .delay          = 10000,
    .presc          = 49,
    .oversampling_shift = 2,
    .resol_bit      = 12,
    .s3c_adc_con    = ADC_TYPE_2,
};
```

进 root@zhuzhaoqi-desktop:/home/zhuzhaoqi/Linux/linux-3.6.7/drivers/input/touchscreen# vim Makefile，添加：

```
obj-$(CONFIG_TOUCHSCREEN_S3C)          += s3c-ts.o
```

进 root@zhuzhaoqi-desktop:/home/zhuzhaoqi/Linux/linux-3.6.7/drivers/input/touchscreen# vim Kconfig，添加：

```
config TOUCHSCREEN_S3C
       tristate "S3C touchscreen driver"
       depends on ARCH_S3C2410 || ARCH_S3C64XX || ARCH_S5P64XX ||
ARCH_S5PC1XX
       default y
       help
```

```
        Say Y here to enable the driver for the touchscreen on the
        S3C SMDK board.

        If unsure, say N.

        To compile this driver as a module, choose M here: the
        module will be called s3c_ts.
```

配置内核 make menuconfig。

Device Drivers--> Input Device support-->[*]Touchscreens -->[*]S3C touchscreen driver

出现多重定义错误，则在 plat-samsung 文件夹下的 devs.c 注释掉 s3c_device_ts 即可：

```
//--->zzq
#undef CONFIG_SAMSUNG_DEV_TS
//<---zzq
#ifdef CONFIG_SAMSUNG_DEV_TS
```

如果还有错误，则可根据错误追溯源头进行修改。

此时启动开发板，虽然 QT 可以启动，LCD 触摸屏也能使用，但是触摸屏很不灵敏。LCD 触摸屏还是存在问题。

19.4 LCD 触摸屏灵敏度修改