第一章 移植内核

1.1 Linux 内核基础知识

在动手进行 Linux 内核移植之前，非常有必要对 Linux 内核进行一定的了解，下面从 Linux 内核的版本和分类说起。

1.1.1 Linux 版本

Linux 内核的版本号可以从源代码的顶层目录下的 Makefile 中看到，比如 2.6.29.1 内核的 Makefile 中：

VERSION = 2

PATCHLEVEL = 6

SUBLEVEL = 29

EXTRAVERSION = .1

其中的“VERSION”和“PATCHLEVEL”组成主版本号，比如 2.4、2.5、2.6 等，稳定版本的德主版本号用偶数表示(比如 2.6 的内核)，开发中的版本号用奇数表示(比如 2.5)，它是下一个稳定版本内核的前身。“SUBLEVEL”称为次版本号，它不分奇偶，顺序递增，每隔 1~2 个月发布一个稳定版本。“EXTRAVERSION”称为扩展版本号，它不分奇偶，顺序递增，每周发布几次扩展本版号。

1.1.2 什么是标准内核

按照资料上的习惯说法，标准内核(或称基础内核)就是指主要在 http://www.kernel.org/维护和获取的内核，实际上它也有平台属性的。这些 linux 内核并不总是适用于所有 linux 支持的体系结构。实际上，这些内核版本很多时候并不是为一些流行的嵌入式 linux 系统开发的，也很少运行于这些嵌入式 linux 系统上，这个站点上的内核首先确保的是在 Intel X86 体系结构上可以正常运行，它是基于 X86 处理器的内核，如对 linux-2.4.18.tar.bz2 的配置 make menuconfig 时就可以看到，Processor type and features--->中只有 386、486、586/K5/5x86/6x86/6x86MX、Pentium-Classic、Pentium-MMX、Pentium-Pro/Celeron/Pentium-II、Pentium-III/Celeron(Coppermine)、Pentium-4、K6/K6-II/K6-III 、Athlon/Duron/K7 、Elan 、Crusoe、Winchip-C6 、Winchip-2 、Winchip-2A/Winchip-3 、CyrixIII/C3 选项，而没有类似 Samsun 2410 等其他芯片的选择。如果需要用在其他特定的处理器平台上就需要对内核进行打补丁，形成不同的嵌入式内核。实际上，不同处理器系统的内核下载站点中提供的也往往是补丁 patch 而已，故原 x86 平台上的内核变成了基础内核，也被称为标准内核了。

1.1.3 Linux 操作系统的分类

第一层次分类：以主要功能差异和发行组织区分(基础 linux 系统/内核)。

1、标准 linux

2、μClinux

无 MMU 支持的 linux 系统，运行在无 MMU 的 CPU 上。

3、Linux-RT

是最早在 linux 上实现硬实时支持的 linux 发行版本。

4、Linux/RTAI

支持硬实时的 linux，于 RT-linux 最大的不同之处在于 RTAI 定义了 RTHAL，它将 RTAI 需要在 linux 中修改的部分定义成一组 API 接口，RTAI 只使用 API 接口与 linux 交互。

5、Embedix

由 Lineo 公司开发，基于 PowerPC 和 x86 平台开发的。

6、Blue Cat Linux

7、Hard Hat Linux

8、其他

第二层分类：以应用的嵌入式平台区分(嵌入式 linux 系统/内核，使上面第一类中的各种 linux 系统扩展为对特定目标硬件的支持，成为一种具体的嵌入式 linux 系统)

由于嵌入式系统的发展与 linux 内核的发展是不同步的，所以为了要找一个能够运行于目标系统上的内核，需要对内核进行选择、配置和定制。因为每一种系统都是国际上不同的内核开发小组维护的，因此选择 linux 内核源码的站点也不尽相同。

第二层分类中的 linux 系统/内核相对于第一层分类的标准内核来说，也可以称为嵌入式 linxu 系统/内核。如应用在 ARM 平台上的嵌入式 Linux 系统通常有 arm-linux(常运行在 arm9 平台上)， μ Clinux(常用在 arm7 平台上)，在标准 linux 基础上扩展对其他的平台的支持往往通过安装 patch 实现，如 armlinux 就是对 linux 安装 rmk 补丁(如 patch-2.4.18-rmk7.bz2)形成的，只有安装了这些补丁，内核才能顺利地移植到 ARM Linux 上。也有些是已经安装好补丁的内核源码包，如 linux-2.4.18-rmk7.tar.bz2。

不同处理器系统的内核/内核补丁下载站点：

处理器系统	适合的内核站点	下载方式
x86	http://www.kernel.org/	ftp, http, rsync
ARM	http://www.arm.linux.org.uk/developer/	ftp, rsync
PowerPC	http://penguinppc.org/	ftp, http, rsync, BitKeeper
MIPS	http://www.linux-mips.org/	ftp, cvs
SuperH	http://linuxsh.sourceforge.net/	cvs, BitKeeper
M68K	http://linux-m68k.org/	ftp, http
non-MMU CPUs	http://www.uclinux.org/	ftp, http

这些站点不仅仅是 linux 内核站点，它们可能直接提供了针对你的目标硬件系统的 linux 内核版本。

1.1.4 linux 内核的选择

选择内核版本是很困难的，应该与负责维护该内核的小组保持联系，方法是通过订阅一些合适的邮件列表（maillist）并查看邮件中相关的重要新闻，以及浏览一些主要站点，可以得到该内核的最新发展动态。如针对 ARM 的 Linux 内核，可以访问 http://www.arm.linux.org.uk/ 并订阅该网站上提供的 maillist 就可以了。如果觉得查阅邮箱中的邮件列表耗费太多时间，那么至少每周访问所关心的内核网站，并阅读 Kernel Traffic 提供的过去一周中在内核邮件清单中发生的重要的摘要，网址为 http://kt.zork.net/kernel-traffic 这样就可以得到相关 Linux 内核的最新信息。

并不是 Linux 的每个版本都适合 ARM-Linux 的移植，可以加入其邮件列表（maillist）以获得内核版本所支持硬件的相关信息，表中列出的资源可以帮助你找到哪些没有列出的功能可以被你的系统支持。ARM Linux 的移植，建议使用 2.4.x 或 2.6.x 版本。Linux 内核补丁可以到 ARM Linux 的 ftp（ftp://ftp.arm.linux.org.uk ）下载。

1.2 Linux 内核启动过程概述

一个嵌入式 Linux 系统从软件角度看可以分为四个部分：引导加载程序（Bootloader），Linux 内核，文件系统，应用程序。其中 Bootloader 是系统启动或复位以后执行的第一段代码，它主要用来初始化处理器及外设，然后调用 Linux 内核。Linux 内核在完成系统的初始化之后需要挂载某个文件系统做为根文件系统（Root Filesystem）。根文件系统是 Linux 系统的核心组成部分，它可以做为 Linux 系统中文件和数据的存储区域，通常它还包括系统配置文件和运行应用软件所需要的库。应用程序可以说是嵌入式系统的“灵魂”，它所实现的功能通常就是设计该嵌入式系统所要达到的目标。如果没有应用程序的支持，任何硬件上设计精良的嵌入式系统都没有实用意义。

1.2.1 Bootloader 启动过程

Bootloader 在运行过程中虽然具有初始化系统和执行用户输入的命令等作用，但它最根本的功能就是为了启动 Linux 内核。

1 、Bootloader 的概念和作用

Bootloader 是嵌入式系统的引导加载程序，它是系统上电后运行的第一段程序，其作用类似于 PC 机上的 BIOS。在完成对系统的初始化任务之后，它会将非易失性存储器（通常是 Flash 或 DOC 等）中的 Linux 内核拷贝到 RAM 中去，然后跳转到内核的第一条指令处继续执行，从而启动 Linux 内核。由此可见，

Bootloader 和 Linux 内核有着密不可分的联系，要想清楚的了解 Linux 内核的启动过程，我们必须先得认识 Bootloader 的执行过程，这样才能对嵌入式系统的整个启动过程有清晰的掌握。

2 、Bootloader 的执行过程

不同的处理器上电或复位后执行的第一条指令地址并不相同，对于 ARM 处理器来说，该地址为 0x00000000。对于一般的嵌入式系统，通常把 Flash 等非易失性存储器映射到这个地址处，而 Bootloader 就位于该存储器的最前端，所以系统上电或复位后执行的第一段程序便是 Bootloader。而因为存储 Bootloader 的存储器不同，Bootloader 的执行过程也并不相同，下面将具体分析。

嵌入式系统中广泛采用的非易失性存储器通常是 Flash，而 Flash 又分为 Nor Flash 和 Nand Flash 两种。 它们之间的不同在于：Nor Flash 支持芯片内执行（XIP， eXecute In Place），这样代码可以在 Flash 上直接执行而不必拷贝到 RAM 中去执行。而 Nand Flash 并不支持 XIP，所以要想执行 Nand Flash 上的代码，必须先将其拷贝到 RAM 中去，然后跳到 RAM 中去执行。

3、Bootloader 的功能

实际应用中的 Bootloader 根据所需功能的不同可以设计得很复杂，除完成基本的初始化系统和调用 Linux 内核等基本任务外，还可以执行很多用户输入的命令，比如设置 Linux 启动参数，给 Flash 分区等；也可以设计得很简单，只完成最基本的功能。但为了能达到启动 Linux 内核的目的，所有的 Bootloader 都必须具备以下功能：

(1)、初始化 RAM

因为 Linux 内核一般都会在 RAM 中运行，所以在调用 Linux 内核之前 bootloader 必须设置和初始化 RAM，为调用 Linux 内核做好准备。初始化 RAM 的任务包括设置 CPU 的控制寄存器参数，以便能正常使用 RAM 以及检测 RAM 大小等。

(2)、初始化串口

串口在 Linux 的启动过程中有着非常重要的作用，它是 Linux 内核和用户交互的方式之一。Linux 在启动过程中可以将信息通过串口输出，这样便可清楚的了解 Linux 的启动过程。虽然它并不是 Bootloader 必须要完成的工作，但是通过串口输出信息是调试 Bootloader 和 Linux 内核的强有力的工具，所以一般的 Bootloader 都会在执行过程中初始化一个串口做为调试端口。

(3)、检测处理器类型

Bootloader 在调用 Linux 内核前必须检测系统的处理器类型，并将其保存到某个常量中提供给 Linux 内核。Linux 内核在启动过程中会根据该处理器类型调用相应的初始化程序。

(4)、设置 Linux 启动参数

Bootloader 在执行过程中必须设置和初始化 Linux 的内核启动参数。目前传递启动参数主要采用两种方式：即通过 struct param_struct 和 struct tag（标记列表，tagged list）两种结构传递。struct param_struct 是一种比较老的参数传递方式，在 2.4 版本以前的内核中使用较多。从 2.4 版本以后 Linux 内核基本上采用标记列表的方式。但为了保持和以前版本的兼容性，它仍支持 struct param_struct 参数传递方式，只不过在内核启动过程中它将被转换成标记列表方式。标记列表方式是种比较新的参数传递方式，它必须以 ATAG_CORE 开始，并以 ATAG_NONE 结尾。中间可以根据需要加入其他列表。Linux 内核在启动过程中会根据该启动参数进行相应的初始化工作。

(5)、调用 Linux 内核映像

Bootloader 完成的最后一项工作便是调用 Linux 内核。如果 Linux 内核存放在 Flash 中，并且可直接在上面运行（这里的 Flash 指 Nor Flash），那么可直接跳转到内核中去执行。但由于在 Flash 中执行代码会有种种限制，而且速度也远不及 RAM 快，所以一般的嵌入式系统都是将 Linux 内核拷贝到 RAM 中，然后跳转到 RAM 中去执行。

不论哪种情况，在跳到 Linux 内核执行之前 CPU 的寄存器必须满足以下条件：r0＝0，r1＝处理器类型，r2＝标记列表在 RAM 中的地址。

1.2.2 Linux 启动过程

在 Bootloader 将 Linux 内核映像拷贝到 RAM 以后，可以通过下例代码启动 Linux 内核：

call_linux(0, machine_type, kernel_params_base)。

其中，machine_tpye 是 Bootloader 检测出来的处理器类型， kernel_params_base 是启动参数在 RAM 的地址。通过这种方式将 Linux 启动需要的参数从 bootloader 传递到内核。

Linux 内核有两种映像：一种是非压缩内核，叫 Image，另一种是它的压缩版本，叫 zImage。根据内核映像的不同，Linux 内核的启动在开始阶段也有所不同。zImage 是 Image 经过压缩形成的，所以它的大小比 Image 小。但为了能使用 zImage，必须在它的开头加上解压缩的代码，将 zImage 解压缩之后才能执行，因此它的执行速度比 Image 要慢。但考虑到嵌入式系统的存储空容量一般比较小，采用 zImage 可以占用较少的存储空间，因此牺牲一点性能上的代价也是值得的。所以一般的嵌入式系统均采用压缩内核的方式。

对于 ARM 系列处理器来说，zImage 的入口程序即为 arch/arm/boot/compressed/head.S。它依次完成以下工作：开启 MMU 和 Cache，调用 decompress_kernel()解压内核，最后通过调用 call_kernel()进入非压缩内核 Image 的启动。下面将具体分析在此之后 Linux 内核的启动过程。

1、 Linux 内核入口

Linux 非压缩内核的入口位于文件/arch/arm/kernel/head-armv.S 中的 stext 段。该段的基地址就是压缩内核解压后的跳转地址。如果系统中加载的内核是非压缩的 Image，那么 bootloader 将内核从 Flash 中拷贝到 RAM 后将直接跳到该地址处，从而启动 Linux 内核。不同体系结构的 Linux 系统的入口文件是不同的，而且因为该文件与具体体系结构有关，所以一般均用汇编语言编写。对基于 ARM 处理的 Linux 系统来说，该文件就是 head-armv.S。该程序通过查找处理器内核类型和处理器类型调用相应的初始化函数，再建立页表，最后跳转到 start_kernel()函数开始内核的初始化工作。检测处理器内核类型是在汇编子函数 __lookup_processor_type 中完成的。通过以下代码可实现对它的调用：

bl __lookup_processor_type。

__lookup_processor_type 调用结束返回原程序时，会将返回结果保存到寄存器中。其中 r8 保存了页表的标志位，r9 保存了处理器的 ID 号，r10 保存了与处理器相关的 stru proc_info_list 结构地址。

检测处理器类型是在汇编子函数 __lookup_architecture_type 中完成的。与 __lookup_processor_type 类似，它通过代码：“bl __lookup_processor_type”来实现对它的调用。该函数返回时，会将返回结构保存在 r5、r6 和 r7 三个寄存器中。其中 r5 保存了 RAM 的起始基地址，r6 保存了 I/O 基地址，r7 保存了 I/O 的页表偏移地址。当检测处理器内核和处理器类型结束后，将调用__create_page_tables 子函数来建立页表，它所要做的工作就是将 RAM 基地址开始的 4M 空间的物理地址映射到 0xC0000000 开始的虚拟地址处。对笔者的 S3C2410 开发板而言，RAM 连接到物理地址 0x30000000 处，当调用 __create_page_tables 结束后 0x30000000 ～ 0x30400000 物理地址将映射到 0xC0000000～0xC0400000 虚拟地址处。当所有的初始化结束之后，使用如下代码来跳到 C 程序的入口函数 start_kernel()处，开始之后的内核初始化工作：
b SYMBOL_NAME(start_kernel)

2 、start_kernel 函数

start_kernel 是所有 Linux 平台进入系统内核初始化后的入口函数，它主要完成剩余的与硬件平台相关的初始化工作，在进行一系列与内核相关的初始化后，调用第一个用户进程－init 进程并等待用户进程的执行，这样整个 Linux 内核便启动完毕。该函数所做的具体工作有：调用 setup_arch()函数进行与体系结构相关的第一个初始化工作；对不同的体系结构来说该函数有不同的定义。对于 ARM 平台而言，该函数定义在 arch/arm/kernel/Setup.c。它首先通过检测出来的处理器类型进行处理器内核的初始化，然后通过 bootmem_init()函数根据系统定义的 meminfo 结构进行内存结构的初始化，最后调用 paging_init()开启 MMU，创建内核页表，映射所有的物理内存和 IO 空间。创建异常向量表和初始化中断处理函数；初始化系统核心进程调度器和时钟中断处理机制；初始化串口控制台（serial-console）；ARM-Linux 在初始化过程中一般都会初始化一个串口做为内核的控制台，这样内核在启动过程中就可以通过串口输出信息以便开发者或用户了解系统的启动进程。创建和初始化系统 cache，为各种内存调用机制提供缓存，包括;动态内存

分配，虚拟文件系统（VirtualFile System）及页缓存。初始化内存管理，检测内存大小及被内核占用的内存情况；初始化系统的进程间通信机制（IPC）；当以上所有的初始化工作结束后，start_kernel()函数会调用rest_init()函数来进行最后的初始化，包括创建系统的第一个进程－init 进程来结束内核的启动。init 进程首先进行一系列的硬件初始化，然后通过命令行传递过来的参数挂载根文件系统。最后 init 进程会执行用户传递过来的“init＝”启动参数执行用户指定的命令，或者执行以下几个进程之一：

execve("/sbin/init",argv_init,envp_init)

execve("/etc/init",argv_init,envp_init)

execve("/bin/init",argv_init,envp_init)

execve("/bin/sh",argv_init,envp_init)

当所有的初始化工作结束后，cpu_idle()函数会被调用来使系统处于闲置（idle）状态并等待用户程序的执行。至此，整个 Linux 内核启动完毕。

Linux 内核是一个非常庞大的工程，经过十多年的发展，它已从从最初的几百 KB 大小发展到现在的几百兆。清晰的了解它执行的每一个过程是件非常困难的事。但是在嵌入式开发过程中，我们并不需要十分清楚 Linux 的内部工作机制，只要适当修改 Linux 内核中那些与硬件相关的部分，就可以将 Linux 移植到其它目标平台上。通过对 Linux 的启动过程的分析，我们可以看出哪些是和硬件相关的，哪些是 Linux 内核内部已实现的功能，这样在移植 Linux 的过程中便有所针对。而 Linux 内核的分层设计将使 Linux 的移植变得更加容易。

1.3 Linux 内核移植

1.3.1 移植内核和根文件系统准备工作

移植内核前，保证你已经装上了 Linux 系统，建立好了交叉编译环境，我用的是虚拟机，装的 Redhat9.0 系统，交叉编译工具用的是友善之臂的 arm-linux-gcc-4.3.2。

开始移植 Linux 内核了，下面是我我使用的内核和文件系统，以及所用到的工具及获取方法：

1、Linux 系统

我是在虚拟机上安装的 Redhat9.0。XP 系统下虚拟机设置的共享目录是 D:\share，对应的 Linux 系统的目录是/mnt/hgfs/share。我将下面准备的压缩包和文件都统一放到该目录下。

2、Linux 内核

到 www.kernel.org/主页，进入该网站中链接 FTP ftp://ftp.kernel.org/pub/，在里面进入文件夹“linux->kernel->v2.6”，会出现很多版本的内核压缩包和补丁，选中 Linux-2.6.29.1.tar.bz2 下载。

3、交叉编译工具链

使用友善之臂提供的 arm-linux-4.3.2 工具链,没有的到 http://www.arm9.net/下载。工具链也可以自己做，可以参考《构建嵌入式 Linux 系统》一书或其它资料。

4、yaffs2 代码

进入 http://www.aleph1.co.uk/cgi-bin/viewcvs.cgi/，点击“Download GNU tarball”，下载后出现 cvs-root.tar.gz 压缩包。

5、busybox-1.13.3

从 http://www.busybox.net/downloads/下载 busybox，这里下载的是 busy busybox-1.13.3.tar.gz。

6、根文件系统制作工具

到友善之臂 http://www.arm9.net/ 网站下载根文件系统制作工具 mkyaffs2image.tgz。

7、友善之臂的根文件系统

在制作根文件系统时，需要用到链接库，为保证链接库能用直接用友善之臂的根文件系统 root_qtopia 中的链接库 lib，到友善之臂网站 http://www.arm9.net/ 下载 root_qtopia.tgz。

这些文件都下载到 D:\share 中，通过虚拟机进入 Redhat9.0 系统，进入/mnt/hgfs/share 目录便可访问这些与 XP 共享的文件。

8、硬件平台

友善之臂的 mini2440

1.3.2 修改 Linux 源码中参数

1、解压内核源码

mkdir /opt/studyarm

cd /mnt/hgfs/share

tar –jxvf linux-2.6.29.1.tar.bz2 –C /opt/studyarm

2、 进入内核目录，修改 makefile，并对内核进行默认配置进行修改

193 行，修改

ARCH ?=arm

CROSS_COMPILE ?=arm-linux-

3、 修改平台输入时钟

找到内核源码 arch/arm/mach-s3c2440/mach-smdk2440.c 文件，在函数 static void __init smdk2440_map_io(void)中，修改成 s3c24xx_init_clocks(12000000)。

4、 修改 machine 名称(可以不改)

修改文件 arch/arm/mach-s3c2440/mach-smdk2440.c，在文件中找到 MACHINE_START( ),修改为 MACHINE_START(S3C2440, “Study-S3C2440”)。

5、 修改 Nand flash 分区信息

修改文件/arch/arm/plat-s3c24xx/common-smdk.c。

第一，修改分区信息：

```
static struct mtd_partition smdk_default_nand_part[] = {
        [0] = {
        .name = "bootloader",
        .offset = 0x00000000,
        .size = 0x00030000,
        },
        [1] = {
        .name = "kernel",
        .offset = 0x00050000,
        .size = 0x00200000,
        },
        [2] = {
        .name = "root",
        .offset = 0x00250000,
        .size = 0x03dac000,
        }
};
```

第二，再修改 s3c2410_platform_nand_smdk_nand_info smdk_nand_info = {

…

.tacls = 0,

.twrph0 = 30,

.twrph1=0,

…

};

6、 修改 LCD 背光

修改文件/arch/arm/mach-s3c2440/mach-smdk2440.c，因为友善的 3.5 寸液晶的背光控制是由 S3C2440 的 GPG4 引脚来控制的，故下面的改动将开启背光。

```
static void __init smdk2440_machine_init(void)
{
        s3c24xx_fb_set_platdata(&smdk2440_fb_info);
        platform_add_devices();
        s3c2410_gpio_cfgpin(S3C2410_GPG4,S3C2410_GPG4_OUTP);
        s3c2410_gpio_setpin(S3C2410_GPG4,1);
 smdk_machine_init();
}
```

6、 LCD 参数修改

这里用的是 NEC3.5 英寸屏液晶屏，大小为 320x240，需要修改修改文件 arch/arm/mach-s3c2440/mach-smdk2440.c。

```
static struct s3c2410fb_display smdk2440_lcd_cfg __initdata =
{
        …
        .right_margin = 37,
        .hsync_len = 6,
        .upper_margin =2,
        .lower_margin = 6,
        .vsync_len =2,
};
static struct s3c2410fb_mach_info smdk2440_fb_info   __initdata   ={
…
.default_display =0
        .gpccon         = 0xaa955699,
        .gpccon_mask    = 0xffc003cc,
        .gpcup          = 0x0000ffff,
        .gpcup_mask = 0xffffffff,
        .gpdcon         = 0xaa95aaa1,
        .gpdcon_mask    = 0xffc0fff0,
        .gpdup          = 0x0000faff,
        .gpdup_mask = 0xffffffff,
.lpcsel = 0xf82,
};
```

增减设备

修改文件/arch/arm/plat-s3c24xx/devs.c　添加内容如下：

```
#include <linux/dm9000.h>
/* DM9000 */
static struct resource s3c_dm9k_resource[] = {
     [0] = {
          .start = S3C2410_CS4,
          .end   = S3C2410_CS4 + 3,
          .flags = IORESOURCE_MEM,
```

```
        },
        [1] = {
                .start = S3C2410_CS4 + 4,
                .end      = S3C2410_CS4 + 4 + 3,
                .flags = IORESOURCE_MEM,
        },
        [2] = {
                .start = IRQ_EINT7,
                .end      = IRQ_EINT7,
                .flags = IORESOURCE_IRQ | IRQF_TRIGGER_RISING,
        }
};
/* CS8900 netif. */
static struct resource cs8900_resource[] = {
 [0] = {
         .start = 0x19000300,
         .end      = 0x19000310,
         .flags = IORESOURCE_MEM,
 },
 [1] = {
         .start = IRQ_EINT9,
         .end      = IRQ_EINT9,
         .flags = IORESOURCE_IRQ,
 }
};
struct platform_device net_device_cs8900 = {
 .name = "cs8900",
 .id = -1,
 .num_resources = ARRAY_SIZE(cs8900_resource),
 .resource      = cs8900_resource,
};
EXPORT_SYMBOL(net_device_cs8900);
/* for the moment we limit ourselves to 16bit IO until some
  * better IO routines can be written and tested
*/
static struct dm9000_plat_data s3c_dm9k_platdata = {
        .flags            = DM9000_PLATF_16BITONLY,
};
struct platform_device s3c_device_dm9k = {
        .name             = "dm9000",
        .id         = 0,
        .num_resources    = ARRAY_SIZE(s3c_dm9k_resource),
        .resource      = s3c_dm9k_resource,
        .dev              = {
```

```
        .platform_data = &s3c_dm9k_platdata,
    }
};
EXPORT_SYMBOL(s3c_device_dm9k);
```

7、 给内核打 yaffs2 文件系统的补丁

```
cd /mnt/hgfs/share
tar –zxvf /mnt/hgfs/share/cvs-root.tar.gz –C /opt/studyarm
    cd /opt/stdudyarm/cvs/yaffs2/
./patch-ker.sh c /opt/studyarm/linux-2.6.29.1/
```

上面命令完成下面三件事情：

(1) 修改内核 fs/Kconfig

增加一行:source "fs/yaffs2/Kconfig"

(2) 修改内核 fs/Kconfig

增加一行:ojb-$(CONFIG_YAFFS_FS) +=yaffs2/

(3) 在内核 fs/目录下创建 yaffs2 目录

将 yaffs2 源码目录下面的 Makefile.kernel 文件复制为内核 fs/yaffs2/Makefie;

将 yaffs2 源码目录的 Kconfig 文件复制到内核 fs/yaffs2 目录下;

将 yaffs2 源码目录下的*.c *.h 文件复制到内核 fs/yaffs2 目录下.

8、修改 S3C2440 的机器号

由于 Bootloader 传递给 Linux 内核的机器号为 782，为与 Bootloader 传递参数一致，修改 arch/arm/tools/math-types 文件。

```
s3c2440             ARCH_S3C2440        S3C2440                 362
```

修改为：

```
s3c2440              ARCH_S3C2440        S3C2440                  782
```

另外，还可以不修改内核中的 S3C2440 机器号，只需修改修改 Bootloader 传递给内核的参数中的机器号就可以了。在 VIVI 中菜单中，按 s，再按 s，输入 mach_type，回车，输入 362，w,保存。

配置网卡

对比友善的 mach-mini2440.c 文件,发现我们的 mach-smdk2440.c 中的 smdk2440_devices[]数组并没有 &s3c_device_dm9k 这个结构,加上,追踪发现该数据结构在 arch/arm/plat-s3c24xx/devs.c 中,我们的 devs.c 中没有该数据结构的定义,加上

```
#include <linux/dm9000.h>
static struct resource s3c_dm9k_resource[] = {
[0] = {
.start = S3C2410_CS4,
.end = S3C2410_CS4 + 3,
.flags = IORESOURCE_MEM,
},
[1] = {
.start = S3C2410_CS4 + 4,
.end = S3C2410_CS4 + 4 + 3,
.flags = IORESOURCE_MEM,
},
[2] = {
.start = IRQ_EINT7,
```

```
.end = IRQ_EINT7,
.flags = IORESOURCE_IRQ | IRQF_TRIGGER_RISING,
},
};
static struct dm9000_plat_data s3c_dm9k_platdata = {
.flags = DM9000_PLATF_16BITONLY,
};
struct platform_device s3c_device_dm9k = {
.name = “dm9000”,
.id = 0;
.num_resources = ARRAY_SIZE(s3c_dm9k_resource),
.resource = s3c_dm9k_resource,
.dev = {
.platform_data = &s3c_dm9k_platdata,
}
};
EXPORT_SYMBOL(s3c_device_dm9k);
```

编译,出错,

error : s3c_device_dm9k undeclared here

找不到结构?~ 打开 mach-smdk2440.c 看看有什么头文件

比较显眼的就是<plat/s3c2410.h><plat/s3c2440.h><plat/devs.h>,我们刚才编辑的文件是 devs.c 呢么 devs.h 的可能性较高,通过搜索,这个 devs.h 在/arch/arm/plat-s3c/include/plat/中,打开,BINGO,里面都是 devs.c 的数据定义

加上我们的 s3c_device_dm9k

extern struct platform_device s3c_device_dm9k;

1.3.3 配置 Linux 内核

1、 进入 Linux-2.6.29.1 内核主目录，通过以下命令将 2410 的默认配置文件写到当前目录下的.config。S3C2410 的配置和 S3C2440 差不多，，在这基础上进行修改。

make s3c2410_defconfig

2、 配置内核模块的功能，有几种方式可以进行界面选择：

make menuconfig（文本选单的配置方式，在有字符终端下才能使用）

make xconfig（图形窗口模式的配置方式，图形窗口的配置比较直观，必须支持 Xwindow 下才能使用）

make oldconfig（文本配置方式，在原内核配置的基础修改时使用）

这里使用 make menuconfig 命令。

3、[*]Enable loadable module support--->

[*]Forced module loading

[*]Module unloading

4、System Type--->

S3C2410 Machines--->

[*]SMDK2410/A9M2410 选上 其余不选 在 input device--->touchscreen 中才会出现 Samsung 2410 的选项。

S3C2440 Machines--->

[*]SMDK2440

[*]SMDK2440 with S3C2440 CPU module，其余不选

其余的 Machines 下选项全部不选（如 2400，2412，2442，2443）

5、Kernel Features--->

[*]Use the ARM EABI to compile the kernel

注：由于所使用的的交叉编译 arm-linux-gcc-4.3.2 是符合 EABI 标准交叉编译器，对于浮点运行会预设硬浮点运算 FPA(Float Point Architecture)，而没有 FPA 的 CPU，比如 SAMSUNG S3C2410/S3C2440，会使用 FPE(Float Point Emulation 即软浮点)，这样在速度上就会遇到极大的限制，使用 EABI(Embedded Application Binary Interface)则可以对此改善处理，ARM EABI 有许多革新之处，其中最突出的改进就是 Float Point Performance，它使用 Vector Float Point(矢量浮点)，因此可以极大提高涉及到浮点运算的程序。

参考：http://www.hotchn.cn/bbs/viewthread.php?tid=130&extra=page%3D1

6、Boot options-?

noinitrd root="/dev/mtdblock2" init="/linuxrc" console=ttySAC0

7、Userspace binary formats--->

[*]Kernel support for ELF binaries

其它的可以全部不选。

8、 选择支持 yaffs2 文件系统

Filesystem--->

Miscellaneous filesystems--->

<*>YAFFS2 file system support

Native language support

<*> Codepage 437 (United States,Canada)

<*>Simplified Chinese charset(GB2312)

<*>Traditional Chinese charset(Big5)

<*>NLS ISO 8859-1(Latin1:Western European Languages)

<*>NLS UTF-8

9、Device Drivers--->

Graphics support--->

<*>Support for frame buffer devices--->

[*]Enable firmware EDID

[*]Enable Video Mode Handling Helpers

<*>S3C2410 LCD framebuffer support

Console display driver support--->

<*>Framebuffer Console support

[*]Select compiled-in fonts

[*] VGA8x8 font

[*]VGA8x16 font

[*]Bootup logo--->

[*]Standard black and white Linux logo

[*]Standard 16-color Linux logo

[*]Standard 224-color Linux logo

在 Bootup logo--->选择的那几项，将会在系统启动时在液晶上显示开机 logo。

1.3.4、编译内核

编译内核需要遵守以下步骤：

1、make dep

make dep 的意思就是说：如果你使用程序 A（比如支持特殊设备），而 A 需用到 B(比如 B 是 A 的一 个模块/子程序）。而你在做 make config 的时候将一个设备的驱动 由内核支持改为 module,或取消支持，这将可能影响到 B 的一个参数的设置，需重新编译 B，重新编译或连接 A....如果程序数量非常多， 你是很难手工完全做好此工作的。make dep 实际上读取配置过程生成的配置文件，来创建对应于配置的依赖关系树，从而决定哪些需要编译而那些不需要编译。所以，你要 make dep。

2、make clean

清除一些以前留下的文件，比如以前编译生成的目标文件，这一步必须要进行。否则，即使内核配置改动过，编译内核时还是将原来生成的目标文件进行连接，而不生成改动后的文件。

3、make zImage

Linux 内核有两种映像：一种是非压缩内核，叫 Image，另一种是它的压缩版本，叫 zImage。根据内核映像的不同，Linux 内核的启动在开始阶段也有所不同。zImage 是 Image 经过压缩形成的，所以它的大小比 Image 小。但为了能使用 zImage，必须在它的开头加上解压缩的代码，将 zImage 解压缩之后才能执行，因此它的执行速度比 Image 要慢。但考虑到嵌入式系统的存储空容量一般比较小，采用 zImage 可以占用较少的存储空间，因此牺牲一点性能上的代价也是值得的，所以一般的嵌入式系统均采用压缩内核的方式。

编译完成后，会在内核目录 arch/arm/boot/下生成 zImage 内核映像文件。

第二章 制作根文件系统

2.1 根文件系统预备知识

嵌入式 Linux 中都需要构建根文件系统，构建根文件系统的规则在 FHS(Filesystem Hierarchy Standard)文档中，下面是根文件系统顶层目录。

目录	内容
bin	存放所有用户都可以使用的、基本的命令。
sbin	存放的是基本的系统命令，它们用于启动系统、修复系统等。
usr	里面存放的是共享、只读的程序和数据。
proc	这是个空目录，常作为 proc 文件系统的挂载点。
dev	该目录存放设备文件和其它特殊文件。
etc	存放系统配置文件，包括启动文件。
lib	存放共享库和可加载块(即驱动程序)，共享库用于启动系统、运行根文件系统中的可执行程序。
boot	引导加载程序使用的静态文件
home	用户主目录，包括供服务账号锁使用的主目录，如 FTP
mnt	用于临时挂接某个文件系统的挂接点，通常是空目录。也可以在里面创建空的子目录。
opt	给主机额外安装软件所摆放的目录。
root	root 用户的主目录
tmp	存放临时文件，通常是空目录。
var	存放可变的数据。

2.2、构建根文件按系统

2.2.1、建立根文件系统目录

进入到/opt/studyarm 目录，新建建立根文件系统目录的脚本文件 create_rootfs_bash，使用命令 chmod +x create_rootfs_bash 改变文件的可执行权限，./create_rootfs_bash 运行脚本，就完成了根文件系统目录的创建。

```
#!/bin/sh
echo "------Create rootfs directons start...--------"
mkdir rootfs
cd      rootfs
echo "--------Create root,dev....----------"
mkdir root dev etc boot tmp var sys proc lib mnt home
mkdir etc/init.d etc/rc.d etc/sysconfig
mkdir usr/sbin usr/bin usr/lib usr/modules
echo "make node in dev/console dev/null"
mknod -m 600 dev/console c 5 1
mknod -m 600 dev/null        c 1 3
mknod ttySAC0 c 204 64
mknod mtdblock2 b 31 2
mkdir mnt/etc mnt/jffs2 mnt/yaffs mnt/data mnt/temp
mkdir var/lib var/lock var/run var/tmp
chmod 1777 tmp
chmod 1777 var/tmp
echo "-------make direction done---------"
```

改变了 tmp 目录的使用权，让它开启 sticky 位，为 tmp 目录的使用权开启此位，可确保 tmp 目录底下建立的文件，只有建立它的用户有权删除。尽管嵌入式系统多半是单用户，不过有些嵌入式应用不一定用 root 的权限来执行，因此需要遵照根文件系统权限位的基本规定来设计。

2.2.2、建立动态链接库

动态链接库直接用友善之臂的，先解压友善之臂的根文件包，拷贝 lib 的内容到新建的根文件目录 lib 内。

```
cd /mnt/hgfs/share
tar –zxvf root_qtopia.tgz –C /opt/studyarm
cp –rfd /opt/studyarm/root_qtopia/lib/* /opt/studyarm/rootfs/lib/*
```

2.2.3 交叉编译 Bosybox

Bosybox 是一个遵循 GPL v2 协议的开源项目，它在编写过程总对文件大小进行优化，并考虑了系统资源有限(比如内存等)的情况，使用 Busybox 可以自动生成根文件系统所需的 bin、sbin、usr 目录和 linuxrc 文件。

1、解压 busybox

```
cd /mnt/hgfs/share
tar –zxvf busybox-1.13.3.tar.gz –C /opt/studyarm
```

2、进入源码，修改 Makefile 文件：

```
cd /opt/studyarm/busybox-1.13.3
```

修改：

```
CROSS_COMPILE     ?=arm-linux-          //第 164 行
ARCH              ?=arm                 //第 189 行
```

3、配置 busybox

输入 make menuconfig 进行配置

(1)、Busybox Settings--->

General Configuration--->

[*] Show verbose applet usage messages

[*] Store applet usage messages in compressed form
[*] Support –install [-s] to install applet links at runtime
[*] Enable locale support(system needs locale for this to work)
[*] Support for –long-options
[*] Use the devpts filesystem for unix98 PTYs
[*] Support writing pidfiles
[*] Runtime SUID/SGID configuration via /etc/busybox.config
[*] Suppress warning message if /etc/busybox.conf is not readable
Build Options--->
[*] Build BusyBox as a static binary(no shared libs)
[*] Build with Large File Support(for accessing files>2GB)
Installation Options->
[]Don't use /usr
Applets links (as soft-links) --->
(./_install) BusyBox installation prefix
Busybox Library Tuning --->
(6)Minimum password legth
(2)MD5:Trade Bytes for Speed
[*]Fsater /proc scanning code(+100bytes)
[*]Command line editing
(1024)Maximum length of input
[*] vi-style line editing commands
(15) History size
[*] History saving
[*] Tab completion
[*]Fancy shell prompts
(4) Copy buffer size ,in kilobytes
[*]Use ioctl names rather than hex values in error messages
[*]Support infiniband HW
(2)、Linux Module Utilities--->
(/lib/modules)Default directory containing modules
(modules.dep)Default name of modules.dep
[*] insmod
[*] rmmod
[*] lsmod
[*] modprobe
-----options common to multiple modutils
[ ] support version 2.2/2.4 Linux kernels
[*]Support tainted module checking with new kernels
[*]Support for module .aliases file
[*] support for modules.symbols file
(3)、在 busybox 中配置对 dev 下设备类型的支持
dev 的创建有三种方法：
手动创建：在制作根文件系统的时候，就在 dev 目录下创建好要使用的设备文件，系统挂接根文件系

统后，就可以使用 dev 目录下的设备文件了。

使用 devfs 文件系统：这种方法已经过时，具有不确定的设备映射、没有足够的主/次设备号、devfs 消耗大量的内存。

udev：它是个用户程序，能根据系统中硬件设备的状态动态的更新设备文件，包括设备文件的创建、删除等。它的操作相对复杂，但灵活性很高

mdev 是 busybox 自带的一个简化版的 udev，适合于嵌入式的应用場合。其具有使用简单的特点。它的作用，就是在系统启动和热插拔或动态加载驱动程序时，自动产生驱动程序所需的节点文件。在以 busybox 为基础构建嵌入式 linux 的根文件系统时，使用它是最优的选择。下面的选项将增加对 mdev 的支持。

Linux System Utilities ---->

[*]Support /etc/mdev.conf

[*]Support command execution at device addition/removal

4、 编译 busybox

编译 busybox 到指定目录：

cd /opt/studyarm/busybox-1.13.3

make CONFIG_PREFIX=/opt/studyarm/rootfs install

在 rootfs 目录下会生成目录 bin、sbin、usr 和文件 linuxrc 的内容。

2.2.4 建立 etc 目录下的配置文件

1、etc/mdev.conf 文件，内容为空。

2、拷贝主机 etc 目录下的 passwd、group、shadow 文件到 rootfs/etc 目录下。

3、etc/sysconfig 目录下新建文件 HOSTNAME，内容为”MrFeng”。

4、etc/inittab 文件：

```
#etc/inittab
::sysinit:/etc/init.d/rcS
ttySAC0::askfirst:-/bin/sh
::ctrlaltdel:/sbin/reboot
::shutdown:/bin/umount -a –r
```

5、etc/init.d/rcS 文件：

```
#!/bin/sh
PATH=/sbin:/bin:/usr/sbin:/usr/bin
runlevel=S
prevlevel=N
umask 022
export PATH runlevel prevlevel
echo "----------munt all----------"
mount -a
echo /sbin/mdev>/proc/sys/kernel/hotplug
mdev -s
echo "************************************************"
echo "*****************Studying ARM**********************"
echo "Kernel version:linux-2.6.29.1"
echo "Student:Feng dong rui"
echo "Date:2009.07.15"
echo "************************************************"
/bin/hostname -F /etc/sysconfig/HOSTNAME
```

使用以下命令改变 rcS 的执行权限：

Chmod +x rcS

6、etc/fstab 文件：

#device	mount-point	type	option	dump	fsck	order
proc	/proc	proc	defaults	0	0	
none	/tmp	ramfs	defaults	0	0	
sysfs	/sys	sysfs	defaults	0	0	
mdev	/dev	ramfs	defaults	0	0	

7、 etc/profile 文件：

```
#Ash profile
#vim:syntax=sh
#No core file by defaults
#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
```

2.2.5 制作根文件系统映像文件

使用以下命令安装好 yaffs 文件系统制作工具：

cd /mnt/hgfs/share

tar –zxvf mkyaffs2image.tgz –C /

在/opt/studyarm 目录下，使用命令 mkyaffs2image rootfs rootfs.img 生成根文件系统映像文件。

第三章 启动系统

将前面两章生成的内核映像文件和根文件系统映像文件下载到 mini2440 开发板，查看启动信息。我成功移植启动信息如下：

```
VIVI version 0.1.4 (root@capcross) (gcc version 2.95.3 20010315 (release)) #0.1.4 Mon Oct 27 10:18:15 CST 2008
MMU table base address = 0x33DFC000
Succeed memory mapping.
DIVN_UPLL0
MPLLVal [M:7fh,P:2h,S:1h]
CLKDIVN:5h
+-------------------------------------------+
| S3C2440A USB Downloader ver R0.03 2004 Jan  |
+-------------------------------------------+
USB: IN_ENDPOINT:1 OUT_ENDPOINT:3
FORMAT: <ADDR(DATA):4>+<SIZE(n+10):4>+<DATA:n>+<CS:2>
NOTE: Power off/on or press the reset button for 1 sec
      in order to get a valid USB device address.
NAND device: Manufacture ID: 0xec, Chip ID: 0x76 (Samsung K9D1208V0M)
Found saved vivi parameters.
```

Press Return to start the LINUX/Wince now, any other key for vivi
Copy linux kernel from 0x00050000 to 0x30008000, size = 0x00200000 ... done
zImage magic = 0x016f2818
Setup linux parameters at 0x30000100
linux command line is: "noinitrd root="/dev/mtdblock2" init="/linuxrc" console="ttySAC0""
MACH_TYPE = 362
NOW, Booting Linux.....
Uncompressing Linux..... done, booting the kernel.
Linux version 2.6.29.1 (root@localhost.localdomain) (gcc version 4.3.2 (Sourcery G++ Lite 2008q3-72)) #8 Sat Jul 18 10:37:22 CST 2009
CPU: ARM920T [41129200] revision 0 (ARMv4T), cr="c0007177"
CPU: VIVT data cache, VIVT instruction cache
Machine: Study-S3C2440
ATAG_INITRD is deprecated; please update your bootloader.
Memory policy: ECC disabled, Data cache writeback
CPU S3C2440A (id 0x32440001)
S3C24XX Clocks, (c) 2004 Simtec Electronics
S3C244X: core 405.000 MHz, memory 101.250 MHz, peripheral 50.625 MHz
CLOCK: Slow mode (1.500 MHz), fast, MPLL on, UPLL on
Built 1 zonelists in Zone order, mobility grouping on. Total pages: 16256
Kernel command line: noinitrd root="/dev/mtdblock2" init="/linuxrc" console="ttySAC0"
irq: clearing pending status 02000000
irq: clearing subpending status 00000002
PID hash table entries: 256 (order: 8, 1024 bytes)
Console: colour dummy device 80x30
console [ttySAC0] enabled
Dentry cache hash table entries: 8192 (order: 3, 32768 bytes)
Inode-cache hash table entries: 4096 (order: 2, 16384 bytes)
Memory: 64MB = 64MB total
Memory: 60876KB available (3536K code, 293K data, 136K init)
Calibrating delay loop... 201.93 BogoMIPS (lpj=504832)
Mount-cache hash table entries: 512
CPU: Testing write buffer coherency: ok
net_namespace: 296 bytes
NET: Registered protocol family 16
S3C2410 Power Management, (c) 2004 Simtec Electronics
S3C2440: Initialising architecture
S3C2440: IRQ Support
S3C24XX DMA Driver, (c) 2003-2004,2006 Simtec Electronics
DMA channel 0 at c4808000, irq 33
DMA channel 1 at c4808040, irq 34
DMA channel 2 at c4808080, irq 35
DMA channel 3 at c48080c0, irq 36

S3C244X: Clock Support, DVS off
bio: create slab <bio-0> at 0
SCSI subsystem initialized
usbcore: registered new interface driver usbfs
usbcore: registered new interface driver hub
usbcore: registered new device driver usb
NET: Registered protocol family 2
IP route cache hash table entries: 1024 (order: 0, 4096 bytes)
TCP established hash table entries: 2048 (order: 2, 16384 bytes)
TCP bind hash table entries: 2048 (order: 1, 8192 bytes)
TCP: Hash tables configured (established 2048 bind 2048)
TCP reno registered
NET: Registered protocol family 1
NTFS driver 2.1.29 [Flags: R/O].
yaffs Jul 18 2009 10:31:41 Installing.
msgmni has been set to 119
io scheduler noop registered
io scheduler anticipatory registered (default)
io scheduler deadline registered
io scheduler cfq registered
Console: switching to colour frame buffer device 30x40
fb0: s3c2410fb frame buffer device
lp: driver loaded but no devices found
ppdev: user-space parallel port driver
Serial: 8250/16550 driver, 4 ports, IRQ sharing enabled
s3c2440-uart.0: s3c2410_serial0 at MMIO 0x50000000 (irq = 70) is a S3C2440
s3c2440-uart.1: s3c2410_serial1 at MMIO 0x50004000 (irq = 73) is a S3C2440
s3c2440-uart.2: s3c2410_serial2 at MMIO 0x50008000 (irq = 76) is a S3C2440
brd: module loaded
loop: module loaded
dm9000 Ethernet Driver, V1.31
Uniform Multi-Platform E-IDE driver
ide-gd driver 1.18
ide-cd driver 5.00
Driver 'sd' needs updating - please use bus_type methods
S3C24XX NAND Driver, (c) 2004 Simtec Electronics
s3c2440-nand s3c2440-nand: Tacls="1", 9ns Twrph0=4 39ns, Twrph1=1 9ns
NAND device: Manufacturer ID: 0xec, Chip ID: 0x76 (Samsung NAND 64MiB 3,3V 8-bit)
Scanning device for bad blocks
Creating 3 MTD partitions on "NAND 64MiB 3,3V 8-bit":
0x000000000000-0x000000030000 : "boot"
0x000000050000-0x000000250000 : "kernel"
0x000000250000-0x000003ffc000 : "kernel"
usbmon: debugfs is not available

```
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 42, io mem 0x49000000
usb usb1: New USB device found, idVendor="1d6b", idProduct="0001"
usb usb1: New USB device strings: Mfr="3", Product="2", SerialNumber="1"
usb usb1: Product: S3C24XX OHCI
usb usb1: Manufacturer: Linux 2.6.29.1 ohci_hcd
usb usb1: SerialNumber: s3c24xx
usb usb1: configuration #1 chosen from 1 choice
hub 1-0:1.0: USB hub found
hub 1-0:1.0: 2 ports detected
usbcore: registered new interface driver libusual
usbcore: registered new interface driver usbserial
USB Serial support registered for generic
usbcore: registered new interface driver usbserial_generic
usbserial: USB Serial Driver core
USB Serial support registered for FTDI USB Serial Device
usbcore: registered new interface driver ftdi_sio
ftdi_sio: v1.4.3:USB FTDI Serial Converters Driver
USB Serial support registered for pl2303
usbcore: registered new interface driver pl2303
pl2303: Prolific PL2303 USB to serial adaptor driver
s3c2410_udc: debugfs dir creation failed -19
mice: PS/2 mouse device common for all mice
i2c /dev entries driver
s3c2440-i2c s3c2440-i2c: slave address 0x10
s3c2440-i2c s3c2440-i2c: bus frequency set to 98 KHz
s3c2440-i2c s3c2440-i2c: i2c-0: S3C I2C adapter
S3C2410 Watchdog Timer, (c) 2004 Simtec Electronics
s3c2410-wdt s3c2410-wdt: watchdog inactive, reset disabled, irq enabled
TCP cubic registered
RPC: Registered udp transport module.
RPC: Registered tcp transport module.
yaffs: dev is 32505858 name is "mtdblock2"
yaffs: passed flags ""
yaffs: Attempting MTD mount on 31.2, "mtdblock2"
yaffs_read_super: isCheckpointed 0
VFS: Mounted root (yaffs filesystem) on device 31:2.
Freeing init memory: 136K
----------munt all----------------

************************************************
*****************Studying ARM**********************
```

Kernel version:linux-2.6.29.1
Student:Feng dong rui
Date:2009.07.15
****
Please press Enter to activate this console.
[@MrFeng=W]#ls
bin etc linuxrc proc sys var
boot home lost+found root tmp www
dev lib mnt sbin usr
[@MrFeng=W]#

第四章 2.6.31 内核移植问题

设备未编译，导致编译内核失败，添加编译规则即可，打开 linux2.6.31/arch/arm/mach-s3c2440/Kconfig；在 92 行后添加如下内容：

复制代码 select S3C_DEV_USB_HOST

初始化代码不对，这个我也不是很清楚原因，待知道的兄弟解释一下，我是从 http://code.google.com/p/mini2440 的代码库里得到的解决方法。解决办法就是打开 linux2.6.31/arch/arm/mach-s3c2440/mach-mini2440.c，将除了下面这一句之外的所有的 __init 去掉即可

复制代码 static char mini2440_features_str[12] __initdata = "0tb";

另外提一下，上面这一句是设置 mini2440 初始化参数的。

结构体缺失的问题，应该不影响启动，但是为了代码规范还是注意点比较好。解决方法，同样是打开 linux2.6.31/arch/arm/mach-s3c2440/mach-mini2440.c，将

复制代码
```
static struct s3c2410_platform_nand mini2440_nand_info __initdata = {
        .tacls          = 0,
        .twrph0         = 25,
        .twrph1         = 15,
        .nr_sets        = ARRAY_SIZE(mini2440_nand_sets),
        .sets           = mini2440_nand_sets,
        .ignore_unset_ecc = 1,
};
```

修改为：

复制代码
```
static struct s3c2410_platform_nand mini2440_nand_info __initdata = {
        .tacls          = 0,
        .twrph0         = 25,
        .twrph1         = 15,
        .nr_sets        = ARRAY_SIZE(mini2440_nand_sets),
        .sets           = mini2440_nand_sets,
        .ignore_unset_ecc = 1,
       .select_chip     = NULL
};
```

再将

复制代码
```
static struct s3c24xx_led_platdata mini2440_led_backlight_pdata __initdata = {
```

```
        .name             = "backlight",
        .gpio             = S3C2410_GPG(4),
        .def_trigger      = "backlight",
};
```

修改为：

复制代码

```
static struct s3c24xx_led_platdata mini2440_led_backlight_pdata __initdata = {
        .name             = "backlight",
        .gpio             = S3C2410_GPG(4),
       .flags             = S3C24XX_LEDF_STARTON,
        .def_trigger      = "backlight",
};
```

即加上复制代码 `.flags  = S3C24XX_LEDF_STARTON,`

这一行。并且打开 arch/arm/mach-s3c2410/include/mach/leds-gpio.h，在 18 行后添加下面一行：

复制代码

```
#define S3C24XX_LEDF_STARTON      (1<<2)          /* Initialise 'on' */
```

以定义从开机一直保持液晶常亮的宏参数。

背光灯的问题，有两种解决方法，一是在启动参数中加上复制代码 mini2440=0t

参数；而是修改上面的

复制代码

```
static char mini2440_features_str[12] __initdata = "0tb";
```

为

复制代码

```
static char mini2440_features_str[12] __initdata = "0t";
```

即可，这里实际上是 mini2440 移植的时候作的一个参数化的处理。具体可以参考 linux2.6.31/arch/arm/mach-s3c2440/mach-mini2440.c 的 584 行到 625 行这一段。

触摸屏支持

一.

（1）

修改 arch/arm/mach-s3c2410/mach-smdk2410.c，在文件中添加:

```
#include <mach/ts.h>
static struct s3c2410ts_mach_info s3c2410_tscfg __initdata = {
        .delay = 10000,
        .presc = 49,
        .oversampling_shift = 2, };
```

（2）修改

```
static struct platform_device *smdk2410_devices[] __initdata = {

        &s3c_device_usb,
        &s3c_device_lcd,
        &s3c_device_wdt,
        &s3c_device_i2c,
        &s3c_device_iis,
        &s3c_device_ts,                 //添加此行  };
```

（3）在 static void _init mini2440_init(void)中加入：

```
        s3c24xx_ts_set_platdata(&s3c2410_tscfg);
```

二.

(1)在 2.6.31.3 内核中：
./arch/arm/plat-s3c/include/plat/devs.h 中加入

```
extern struct platform_device s3c_device_ts;
```

(2)在 driver/input/touchscreen/下把 s3c2410-ts.c 文件拷贝过去
(3) 在 arch/arm/mach-s3c2410/include/mach/下新建 ts.h 文件， ts.h 文件内容如下：

```
#ifndef __ASM_ARM_S3C2410_TS_H
#define __ASM_ARM_S3C2410_TS_H
struct s3c2410ts_mach_info {
int delay;
int presc;
    int oversampling_shift;
};
extern    void __init s3c24xx_ts_set_platdata(struct s3c2410ts_mach_info *);
#endif
```

(如果编译时这行出错，改为:

```
extern void    s3c24xx_ts_set_platdata(struct s3c2410ts_mach_info *);

        /* _ASM_ARM_S3C2410_TS_H */
```

三.
（1）修改 arch/arm/plat-s3c24xx/devs.c，至少要在 s3c_adc_resource 定义之后,加入:

```
#include <mach/ts.h>
/* Touch Screen    Controller  触摸屏*/
struct platform_device s3c_device_ts = {
        .name           = "s3c2410-ts",
        .id             = -1,
};
EXPORT_SYMBOL(s3c_device_ts);
void __init s3c24xx_ts_set_platdata(struct s3c2410ts_mach_info *pd) {
      struct s3c2410ts_mach_info *npd;
      npd = kmalloc(sizeof(*npd), GFP_KERNEL);
      if (npd) {
          memcpy(npd, pd, sizeof(*npd));
              s3c_device_ts.dev.platform_data = npd;          }
      else      {
          printk(KERN_ERR "no memory for TS platform data");
      } }
```

四.
将 s3c2410-ts.c 文件拷入 drivers/input/touchscreen/目录下。
在 s3c2410-ts.c 文件中：
（1）修改头文件：

```
//#include <asm/plat-s3c/regs-adc.h>
//#include <asm/arch/regs-gpio.h>
//#include <asm/arch/ts.h>
```

添加头文件：

```
#include <plat/regs-adc.h>
#include <mach/regs-gpio.h>
#include <mach/ts.h>
#include <mach/gpio-fns.h>
```

（2）添加宏定义

```
#define S3C2410_GPG12           S3C2410_GPIONO(S3C2410_GPIO_BANKG, 12)
#define S3C2410_GPG13           S3C2410_GPIONO(S3C2410_GPIO_BANKG, 13)
#define S3C2410_GPG14           S3C2410_GPIONO(S3C2410_GPIO_BANKG, 14)
#define S3C2410_GPG15           S3C2410_GPIONO(S3C2410_GPIO_BANKG, 15)
```

五.

（1）修改 drivers/input/touchscreen/Makefile 加入：

（参照文件中的格式，注意 Tob 键和空格键，注意注释时要用“#”而不能用“//”

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

（2）修改 drivers/input/touchscreen/Kconfig，在 if INPUT_TOUCHSCREEN 下加入（参照文件中的格式，注意 Tob 键和空格键，注意注释时要用“#”而不能用“//”）

```
config TOUCHSCREEN_S3C2410
 tristate        “s3c2410 touchscreen”
            depends on ARCH_SMDK2410
            default y
            help
            This is used for supporting s3c2410 touchscreen.
```

六.

修改 drivers/char/Makefile

```
obj-$(CONFIG_JS_RTC)            += js-rtc.o
js-rtc-y = rtc.o
```

加入下面一句

```
obj-$(CONFIG_MINI2440_ADC)      += mini2440_adc.o
```

修改 drivers/char/Kconfig

```
config DEVKMEM
      bool "/dev/kmem virtual device support"
      default y
      help
        Say Y here if you want to support the /dev/kmem device. The
        /dev/kmem device is rarely used, but can be used for certain
        kind of kernel debugging operations.
        When in doubt, say "N".
```

加入下面

```
config MINI2440_ADC
      bool "ADC driver for FriendlyARM Mini2440/QQ2440 development boards"
      default y if MACH_FRIENDLY_ARM_MINI2440
```

help

this is ADC driver for FriendlyARM Mini2440/QQ2440 development boards

Notes: the touch-screen-driver required this option

在 drivers/char 放入 友善的 s3c24xx-adc.h mini2440_adc.c

在 drivers/input/touchscreen/ 放入 友善的 s3c2410_ts.c

make menuconfig

选中

Device drivers -> Input device support -> Touchscreen -> Samsung S3C2410 touchscreen input driver

Device drivers -> character devices -> ADC driver for FriendlyARM Mini2440/QQ2440 development boards

设备未编译，导致编译内核失败，添加编译规则即可，打开 linux2.6.31/arch/arm/mach-s3c2440/Kconfig；在 92 行后添加如下内容：

复制代码 select S3C_DEV_USB_HOST

初始化代码不对，这个我也不是很清楚原因，待知道的兄弟解释一下，我是从 http://code.google.com/p/mini2440 的代码库里得到的解决方法。解决办法就是打开 linux2.6.31/arch/arm/mach-s3c2440/mach-mini2440.c，将除了下面这一句之外的所有的 __init 去掉即可

复制代码 static char mini2440_features_str[12] __initdata = "0tb";

另外提一下，上面这一句是设置 mini2440 初始化参数的。

结构体缺失的问题，应该不影响启动，但是为了代码规范还是注意点比较好。解决方法，同样是打开 linux2.6.31/arch/arm/mach-s3c2440/mach-mini2440.c，将

复制代码
```
static struct s3c2410_platform_nand mini2440_nand_info __initdata = {
        .tacls          = 0,
        .twrph0         = 25,
        .twrph1         = 15,
        .nr_sets        = ARRAY_SIZE(mini2440_nand_sets),
        .sets           = mini2440_nand_sets,
        .ignore_unset_ecc = 1,
};
```

修改为：

复制代码
```
static struct s3c2410_platform_nand mini2440_nand_info __initdata = {
        .tacls          = 0,
        .twrph0         = 25,
        .twrph1         = 15,
        .nr_sets        = ARRAY_SIZE(mini2440_nand_sets),
        .sets           = mini2440_nand_sets,
        .ignore_unset_ecc = 1,
       .select_chip     = NULL
};
```

再将

复制代码
```
static struct s3c24xx_led_platdata mini2440_led_backlight_pdata __initdata = {
```

```
        .name             = "backlight",
        .gpio             = S3C2410_GPG(4),
        .def_trigger      = "backlight",
};
```

修改为：

复制代码

```
static struct s3c24xx_led_platdata mini2440_led_backlight_pdata __initdata = {
        .name             = "backlight",
        .gpio             = S3C2410_GPG(4),
        .flags            = S3C24XX_LEDF_STARTON,
        .def_trigger      = "backlight",
};
```

即加上复制代码.flags = S3C24XX_LEDF_STARTON,这一行。并且打开 arch/arm/mach-s3c2410/include/mach/leds-gpio.h，在 18 行后添加下面一行：

复制代码

```
#define S3C24XX_LEDF_STARTON      (1<<2)          /* Initialise 'on' */
```

以定义从开机一直保持液晶常亮的宏参数。背光灯的问题，有两种解决方法，一是在启动参数中加上复制代码 mini2440=0t 参数；二是修改上面的复制代码

```
static char mini2440_features_str[12] __initdata = "0tb";
```

为复制代码

```
static char mini2440_features_str[12] __initdata = "0t";
```

即可，这里实际上是 mini2440 移植的时候作的一个参数化的处理。具体可以参考 linux2.6.31/arch/arm/mach-s3c2440/mach-mini2440.c 的 584 行到 625 行这一段。