此文章已于 16:56:30 2016/12/26 重新发布到 MADHEX

arm环境u-boot功能和linux启动过程

# U-BOOT概况总结

1. 启动代码为start.S
2. 由于u-boot为裸板上第一个程序，所以要初始化各个外设，由于u-boot已经初始化了，

包括：中断禁用、分配动态内存、初始化BBS区域、初始化页目录、打开缓存等任务。

所以linux内核不需要全部再次初始化，所以可以理解bootloader和kernel其实是两兄弟。

例:

/\*start code on reset\*/

Reset:

bl set\_cpu\_mode /\*设置特权模式\*/

bl turn\_off\_watchdog /\*关闭看门狗\*/

bl mask\_irqs /\*关闭中断\*/

bl set\_clock /\*设置时钟\*/

bl disable\_id\_caches /\*关闭caches\*/

bl init\_memory /\*初始化sram\*/

bl init\_stack /\*初始化栈\*/

bl clean\_bss /\*初始化bss\*/

bl nand\_init /\*初始化nand\*/

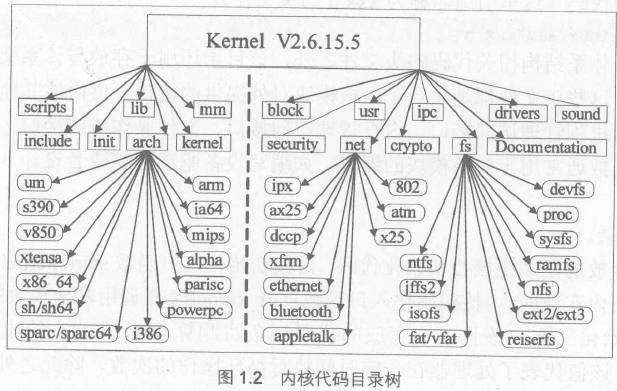
bl copy\_to\_ram /\*复制自身到ram\*/

ldr pc, =arm\_main /\*执行main函数\*/

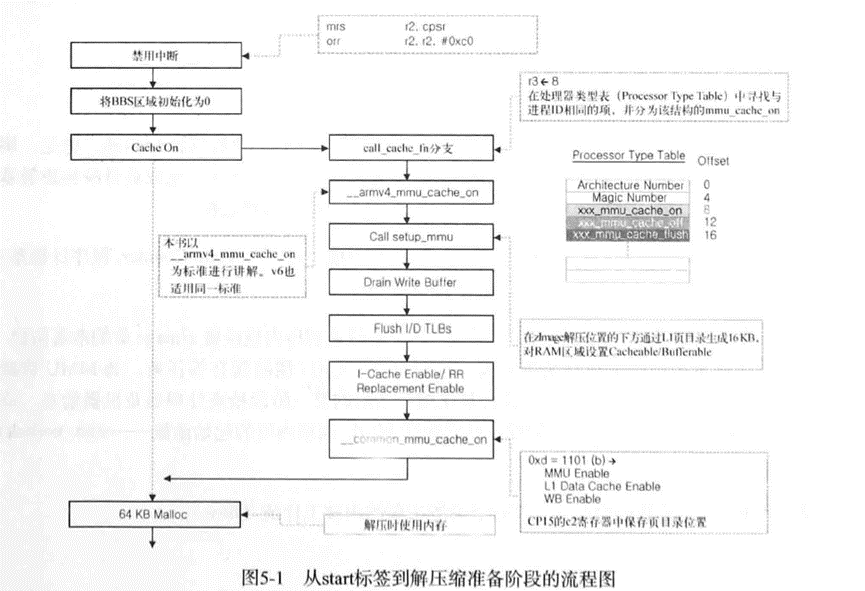
1. U-boot引导内核第一步要做的的是把内核下到ram中，然后跳到内核的start函数。

四、bootloader必须提供5种功能：RAM初始化、串行端口初始化、查找机器类别、构建tagged list内核、将控制移交到内核镜像。

# linux内核初始化流程



u-boot跳到linux内核的arch/arm/boot/compressed/head.S中start后



内核大致可以理解为三部曲：

第一步 解压内核并跳到start\_kernel()。

1.arch/arm/boot/compressed/head.S(start)

准备解压内核

2.arch/arm/boot/compressed/head.S

通过decompress\_kernel 解压zImage

通过call\_kernel调用已解压后内核vmlinux

3.arch/arm/kernel/head.S

通过ENTRY(stext)->

处理器信息搜寻——\_\_look\_processor\_type

搜寻我的机型——\_\_lookup\_machine\_type

通过\_\_mmap\_switched调用start\_kernel

第二步 跳到真正第一个内核代码init/main.c(start\_kernel())。注:在linux 0.11中为init/main.c main()。

其中明星函数（排名不分先后）：

1.初始化console(此处为平台相关需要hack的地方<1>)

start\_kernel->console\_init->serial\_pxa\_console\_init

2.处理与架构相关的一系列事物

start\_kernel->setup-arch()[/arch/arm/kernel/setup.c]

赋值内核结构体machine\_desc，在不同时期被调用：

1）init\_machine 在 arch/arm/kernel/setup.c 中被 customize\_machine 调用，放在 arch\_initcall() 段里面，会自动按顺序被调用。

2）init\_irq在start\_kernel() --> init\_IRQ() --> init\_arch\_irq()中被调用

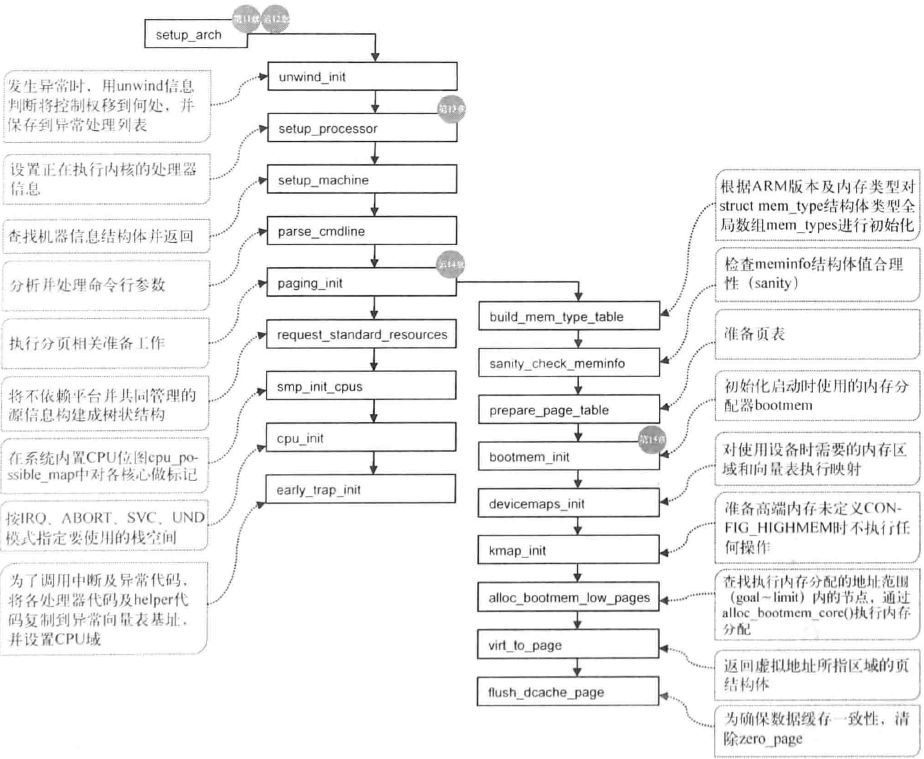
3）map\_io 在 setup\_arch() --> paging\_init() --> devicemaps\_init()中被调用

4）timer是定义系统时钟，定义TIMER4为系统时钟，在arch/arm/plat-s3c/time.c中体现。在start\_kernel() --> time\_init()中被调用。

5）boot\_params是bootloader向内核传递的参数的位置，这要和bootloader中参数的定义要一致。

例: start\_kernel()--->setup\_arch()--->do\_initcalls()--->customize\_machine()--->mini6410\_machine\_init()

http://blog.csdn.net/boyemachao/article/details/45394363



3.初始化驱动平台platform

start\_kernel->kernel\_init()→do\_basic\_setup()→driver\_init()→platform\_bus\_init()→bus\_register(&platform\_bus\_type)

4.初始化soc(initcall参考下面的init.h)

start\_kernel->kernel\_init->依次调用do\_basic\_setup()->do\_initcalls()->do\_one\_initcall()



第三步 就为愉快的init的代码了也就是第一个进程，在linux 0.11中集成在内核中。在之后版本由用户态提供。

当然我已经升级为systemd啦，哈哈。

# linux初始化代码框架

哪部分是只和CPU相关的部分，

哪部分是SOC平台相关的部分，

也再次标明LINUX下大致逻辑。

## 第一部分代码arch/arm/boot/

简介：也就是自我解压部分，可以理解为和cpu相关但和SOC无关。

移植：移植时选择对应的cpu。

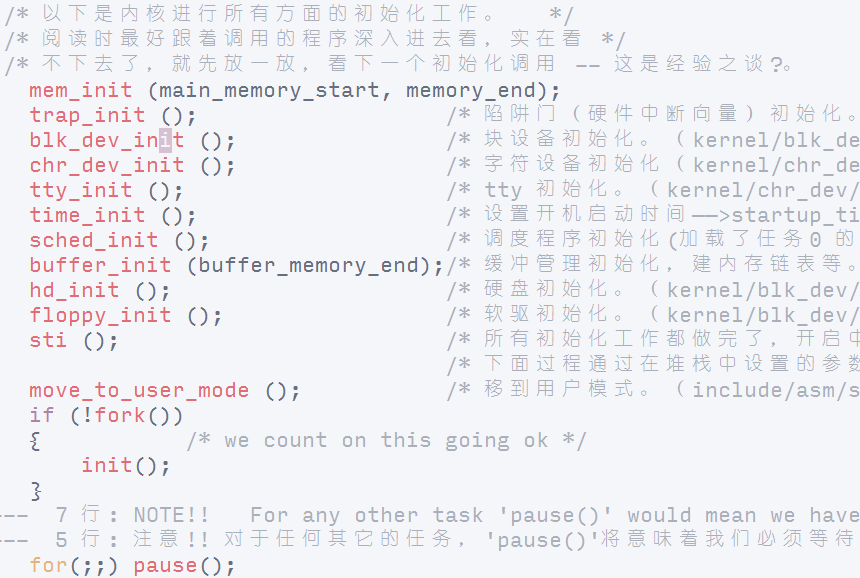
功能：arch/arm/boot/compressed/head.S自我解压前准备、解压缩。跳到start\_kernel。

## 第二部分代码init/main.c

简介：也就是start\_kernel（）或0.11中的main（） 可以理解内核真正的第一代码。为平台无关代码。

移植：移植时不用管。

功能：此代码虽说简单的大几十行，但功能太多，我还是给个linux 0.11中main()的靓照吧~\_~！



## 第三部分代码 arch/arm/PLAT- arch/arm/MACH-

简介：为了完成start\_kernel()中初始化的SOC相关部分。

移植：移植时需要考虑对应的SOC。没有对应soc支持时，才是各路玩家大显身手的时候。

功能：看下面的介绍吧。

# Mini2440的plat和mach

plat-s3c24xx

mach-s3c2440

mach-s3c2410

======================

1. 三星这样分层的理由是s3c系列的soc具有一定的共通性, plat-实现了一些较通用的封装, 这些封装的具体参数一般是宏, 这些宏如寄存器地址可能是在mach-里面被定义;

linux/arch/arm/plat-s3c24xx/common-smdk.c

static struct s3c24xx\_led\_platdata smdk\_pdata\_led5 = {

.gpio = S3C2410\_GPF5,

.flags = S3C24XX\_LEDF\_ACTLOW | S3C24XX\_LEDF\_TRISTATE,

.name = "led5",

.def\_trigger = "nand-disk",

};

linux/include/asm-arm/arch-s3c2410/regs-gpio.h

#define S3C2410\_GPF5 S3C2410\_GPIONO(S3C2410\_GPIO\_BANKF, 5)

2. 原则上是把所有s3c系列共同的东西放在 plat-里面去, 具体的io或者比较有mach-特色的部分放到mach-里面;

改板时, 实际上大多是直接在mach-里面增删自己的功能. (不按三星预设方案的改动除外)

plat里面需要动的相对更少, 不过在linux/arch/arm/plat-s3c24xx/common-smdk.c里面, 我们可以根据实际情形来分配nand的分区(修改static struct mtd\_partition smdk\_default\_nand\_part[] );

3. 编译时,一般只会选中一个特定的mach-, mach-会调用plat-的功能具体实现平台的资源和设备初始化.

## MACHINE\_START

以mach-s3c2440/mach-smdk2440.c 为例：

MACHINE\_START(S3C2440, "SMDK2440")

/\* Maintainer: Ben Dooks <ben@fluff.org> \*/

.phys\_io = S3C2410\_PA\_UART,

.io\_pg\_offst = (((u32)S3C24XX\_VA\_UART) >> 18) & 0xfffc,

.boot\_params = S3C2410\_SDRAM\_PA + 0x100,

.init\_irq = s3c24xx\_init\_irq,

.map\_io = smdk2440\_map\_io,

.init\_machine = smdk2440\_machine\_init,

.timer = &s3c24xx\_timer,

MACHINE\_END

start\_kernel里setup\_arch：

mdesc = setup\_machine(machine\_arch\_type);

init\_arch\_irq = mdesc->init\_irq;

system\_timer = mdesc->timer;

init\_machine = mdesc->init\_machine;

mdesc 即是我们定义的machine type，这个结构体里我们定义的借口调用顺序如下：

mdesc->fixup()； //setup\_arch调用

mdesc->map\_io()； //setup\_arch-》paging\_init-》devicemaps\_init

init\_arch\_irq； //start\_kernel-》init\_IRQ

system\_timer->init(); //start\_kernel-》time\_init

init\_machine; //arch\_initcall

# 5.内核移植

在一般普通的产品公司，大部分都是在soc上添加自己需要的外设做成产品。

并且常见的soc都已经有了成熟的soc代码支持。

现在讨论内核移植重点在板级支持，这属于BSP工程师的范围。并非属于芯片工程。

BSP工程师的能力就是要在各种板子上移植我们万众期待的linux内核。

总的来说，我们所做的就是要找到外设的区别，熟悉内核的框架，

修改外设的驱动，使内核完美的运行起来就好了。

所涉及到的驱动：GPIO、协处理器、中断、时钟、MMU、存储控制器、NAND、UART、IIC等。

我们可以找出最基本的，启动linux所依赖的。并根据依赖的顺序一一攻破。

我暂时认为，学习此部分好的方法是先在一个成熟板子上写一个裸板程序。

这样可以熟悉板子硬件，也可熟悉一个小的系统是如何跑起来的。

有了这个基础我们就能把我哪些是最为基础的驱动。和依赖的顺序等。

通过本文可以体会到，对于内核，移植它改造它有难度呀。

会裸板驱动、会写bootloader只是基础。

还要熟悉内核代码，真是上知系统、应用，下知硬件、驱动。

参考资料：

[console] early printk实现流程

<http://blog.csdn.net/ooonebook/article/details/52654120>

linux2.6中的console\_init初始化的研究

<http://blog.csdn.net/breeze_vickie/article/details/5563375>

linux下tty，控制台，虚拟终端，串口，console（控制台终端）详解

<http://blog.csdn.net/liaoxinmeng/article/details/5004743>

【原创】s3c2440内核启动时如何注册串口为终端设备

<http://blog.sina.com.cn/s/blog_70ef2ee90100zc4z.html>