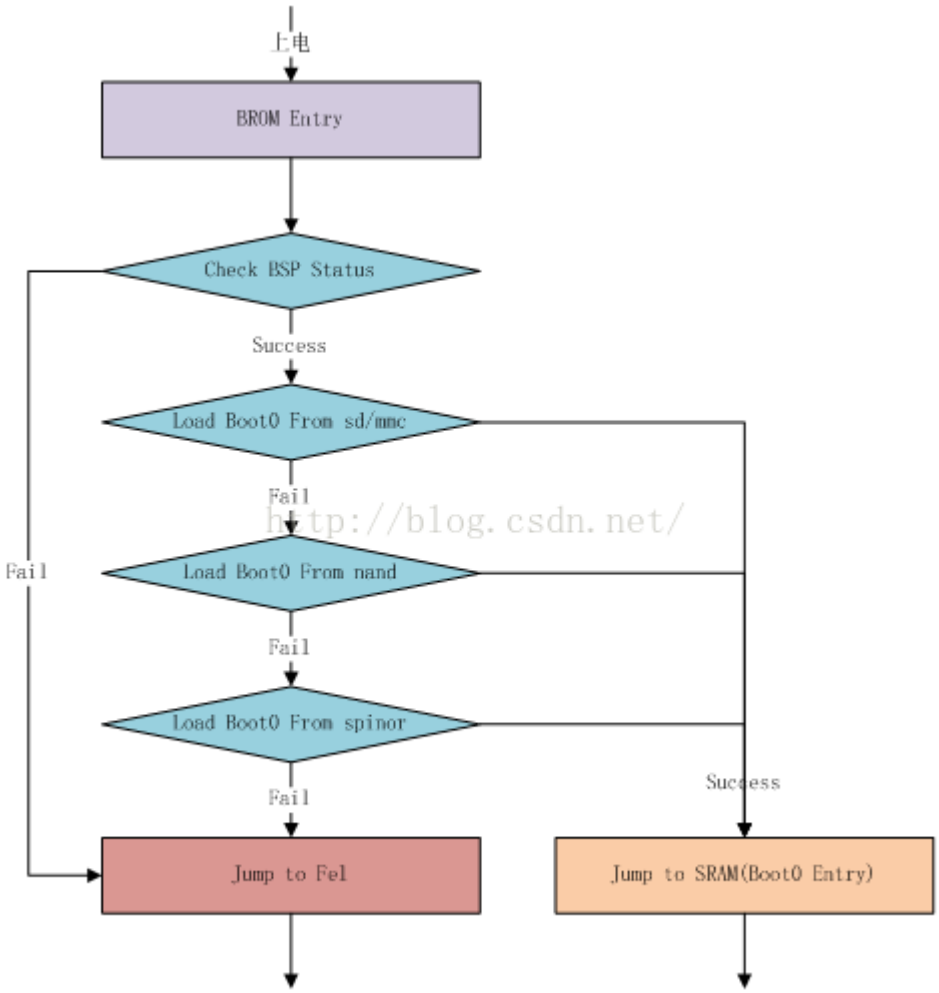
全志平台 linux 启动流程分析

一、BROM 阶段

机器上电之后会执行固化在 BROM 里面的一段引导程序，这个程序会依次遍历所有支持的启动介质，直到找到第一个支持的。目前支持的启动介质是 sd/mmc 卡、nand 和 spinor。当程序初始化启动介质成功后，就从固定位置读入 Bootloader 的 Boot0 到 SRAM，然后跳到 SRAM 执行。

下面展示了 BROM 的执行流程



二、Bootloader 阶段

Bootloader 是全志平台上从小系统一直沿用下来的内核加载器，在这里的主要职责是加载 U-Boot 到 DRAM。

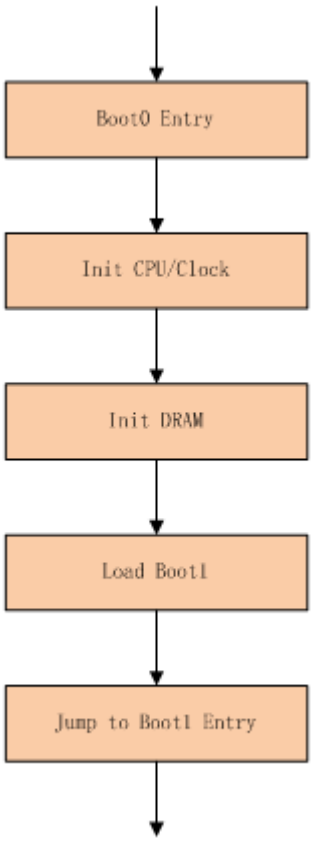
Bootloader 分为两个部分，分别是 Boot0 和 Boot1。

Boot0：初始化 DRAM，加载 Boot1 到 DRAM；

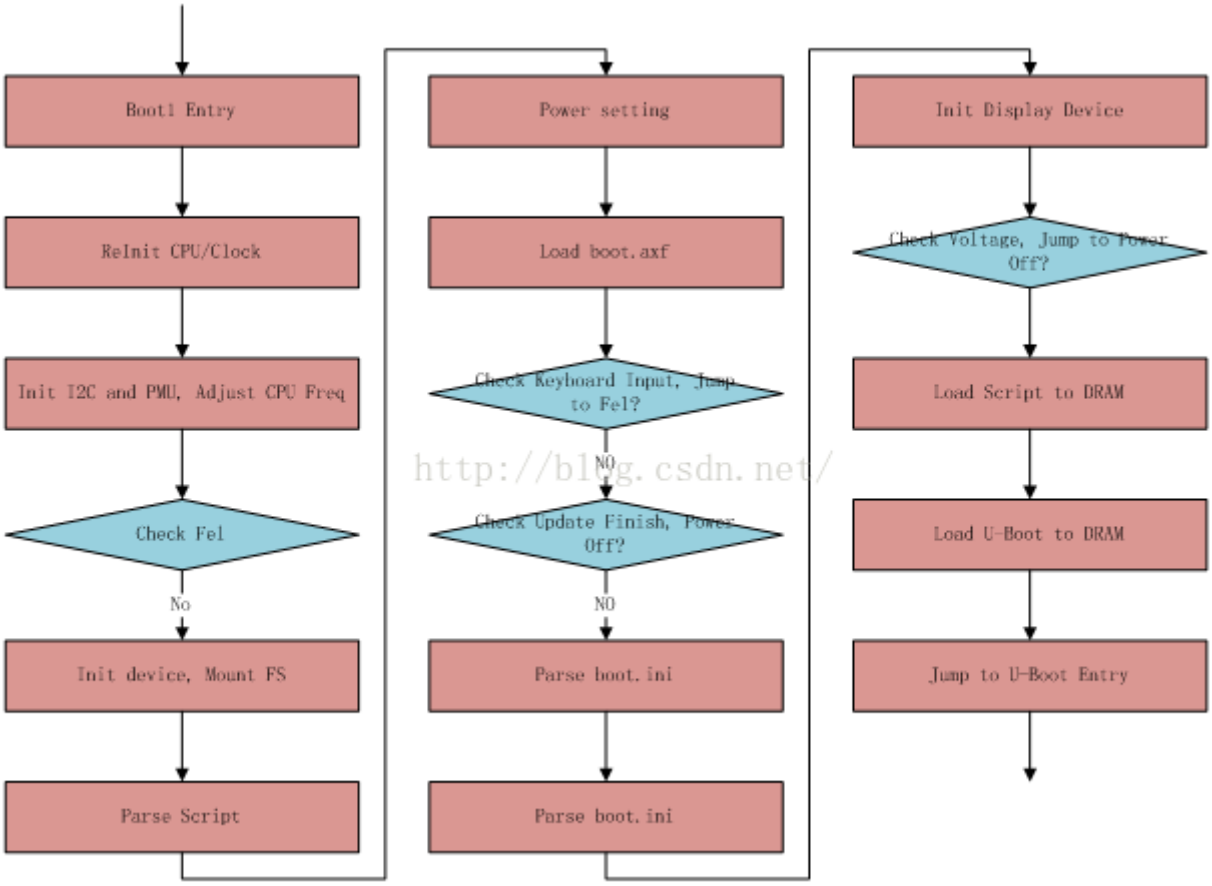
Boot1：调频，加载 U-Boot 到 DRAM；

为什么 Bootloader 要划分成 Boot0 和 Boot1 两个部分？因为在 Bootloader 阶段，使用的 SRAM 大小是 32KB，除去 C 运行环境需要的栈空间，可用的空间在 24KB 左右，这点不足以载入整个 Bootloader。因此，需要将 Bootloader 划分成两个部分，尽可能将繁重的任务放在 Boot1 执行，这个情况类似于 Linux 系统中断执行环境的上半部和下半部。

1. boot0 执行过程



2. boot1 的执行过程



Boot1 会进行一次系统调频，将 CPU 的频率调到用户在 sys_config1.fex target 段配置的 boot_clock。

如何在 Boot1 让机器进入升级模式？

（1）按住 power 键，再按任意键 3 下；

（2）接上串口启动，进入 Boot1 后在键盘输入 2；

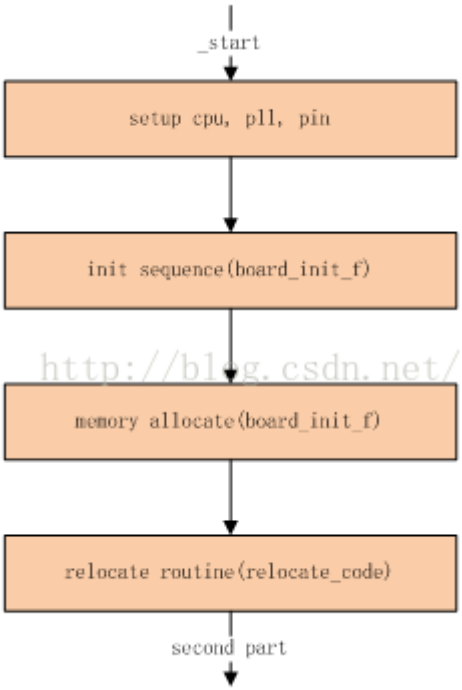
如何替换 Bootloader 分区的内容？

接上串口启动，进入 Boot1 后在键盘输入 1，USB 会挂载 Bootloader 分区到 PC 上，卷标是“Volume”，替换掉相关的文件之后重启机器即可生效。Boot1 会检测低电关机，以及插入火牛开机的情况进入关机程序。后者需要在 sys_config1.fex 里配置。

三、u-boot 阶段

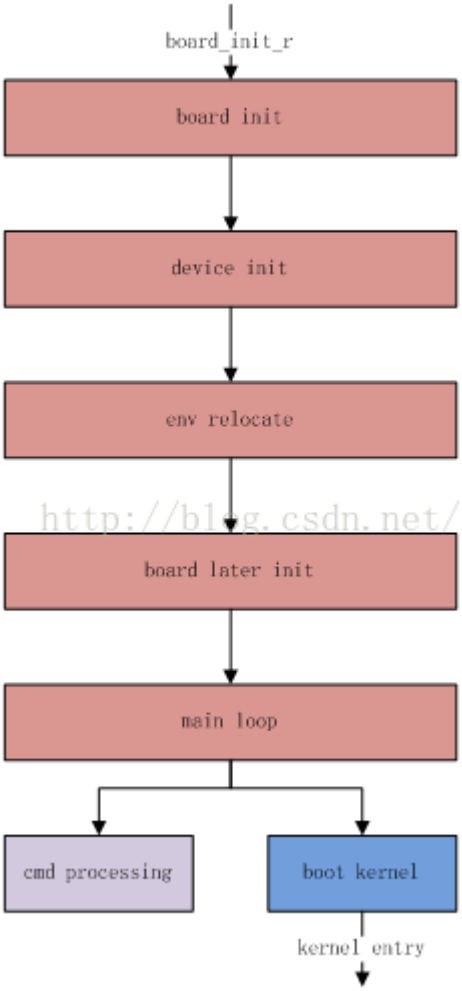
概括地说，U-Boot 引导内核分为两个阶段，第一阶段负责重定位 U-Boot 到最高地址，第二阶段才是真正的引导内核。

1. 第一阶段



在第一阶段会关闭 I/D cache 和 MMU，因此，整个 U-Boot 是直接运行在物理地址上。

2. 第二阶段



U-Boot 第二阶段有一个完整、宽松的 C 环境，不再受制于栈空间，各个平台可以在这个阶段完成一些复杂的操作，以求达到定制的目的。

a. board init

执行平台相关的初始化，这些比较复杂，不适宜在第一阶段完成。这部分的功能在 board/allwinner 目录下实现。

b. device init

主要是根据用户的编译配置选择初始化对应的存储设备。这个地方可以改进，比如根据用户的配置文件

sys_config1.fex 选择初始化对应的存储设备。这样可以做到一份 u-boot.bin 适应不同的存储设备。

c. env relocate

初始化环境变量。

d. board later init

初始化 fastboot 和 android recovery，可能修改 bootcmd，影响引导流程。

e. main loop

U-Boot 的主程序，负责引导内核以及处理用户的命令请求。这部分的功能在 common 目录下实现。

四、内核启动

在我们平台上使用的是非压缩的内核（bImage），因此内核的启动省去了自解压的过程。内核的链接依赖链接脚本 vmlinux.lds，我们平台使用了 ARM 的内核，对应的链接脚本是 arch/arm/kernel/vmlinux.lds。

1. 调用__lookup_processor_type@head-common.S 查找处理器信息

2. 调用__create_page_tables@head.S 为内核自身创建页表

3. 调用处理器底层初始化函数 __v7_setup@arch/arm/mm/proc-v7.S，初始化 MMU，Cache，TLB

4. 调用__enable_mmu@head.S 使能 MMU

5. 调用__mmap_switched@head-common.S 重定位数据段，清零 BSS，然后跳转到 C 函数入口

start_kernel@init/main.c，start_kernel()函数是内核初始化 C 语言部分的主体。这个函数完成系统底层基本机

制，包括处理器、存储管理系统、进程管理系统、中断机制、定时机制等的初始化工作。由于这个函数过于复

杂，这里仅阐述关键点。

6. 调用 setup_arch 完成架构相关的初始化

7. 调用 pidhash_init 初始化 pid hast 机制

8. 调用 sched_init 初始化调度器

9. 调用 init_IRQ 初始化中断机制

10. 调用 softirq_init 完成软中断初始化

11. 调用 local_irq_enable 打开中断

12. 调用 console_init 完成控制台初始化

13. 调用 rest_init

由上可知，rest_init 函数创建了两个内核线程，分别是 kernel_init 和 kthreadd。kernel_init 函数将完成设备驱动

程序的初始化，并调用 init_post 函数启动用户空间的 init 进程。kthreadd 的作用是管理调度其他的内核线程。

14. 调用 do_basic_setup 函数初始化设备，完成外设及其驱动程序（直接编译进内核的模块）的加载和初始化。

a. cpuset_init_smp@init/cpuset.c

创建 cpuset 工作队列，它的作用是控制每个程序在哪个核心执行，对于多核的 CPU。

b. usermodehelper_init@init/kmod.c

创建 khelper 工作队列，它的作用是指定用户空间的程序路径和环境变量，最终运行 user space 的程序。

c. init_tmpfs@mm/shmem.c

初始化 tmpfs。

d. driver_init@driver/base/init.c

初始化设备模型。

e. init_irq_proc@kernel/irq/proc.c

初始化/proc/irq。

f. do_ctors@init/main.c

调用所有构造函数。

g. do_initcalls

初始化子系统。注意，.initcall.init 节所保存的函数地址有一定的优先级，越前面的函数优先级越高，会被先调

用。include/linux/init.h 定义若干的宏协助内核模块添加它们的初始化函数到.initcall.init 节。如需控制同一优先

级的初始化函数执行顺序，可以通过修改模块的 Makefile 调动编译链接顺序。

15. 调用 init_post 函数启动用户空间的 init 进程。在 Android 系统中，init 进程在 system/core/init 目录下实现。

init_post 函数会调用 run_init_process 执行 ramdisk_execute_command，在 run_init_process 中，

kernel_execve 负责创建用户空间的 init 进程。