U-BOOT 内存布局及启动过程浅析

作　　者：cugfeng
完成日期：2009-2-7
更新日期：2009-2-12
当前版本：v1.3
电子邮箱：cugfeng@gmail.com
个人博客：http://hi.baidu.com/cugfeng/

本文以 ARC600 平台的某一实现为例，对 U-BOOT 的内存布局和启动方式进行简要的分析。

【内存布局】
在 ARC600 平台，U-BOOT 的内存布局图 1 所示：

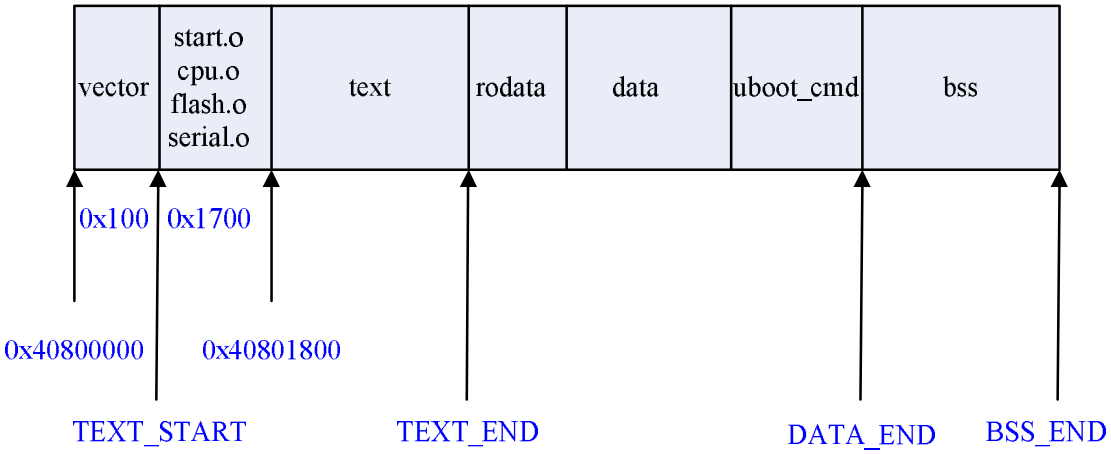

图 1 U-BOOT 内存布局

该布局由 board/arc600/u-boot.lds 文件定义，在链接的时候生成相应的二进制映像。首先，定义起始地址为 0x40800000，接下来是中断向量表，大小为 256 字节，按每个中断向量占用 4 个字节的跳转地址算，最多可以有 64 个中断向量；第二部分是一些基础性的代码段，它为下一步加载 boot 或者 kernel 做准备，其大小为 0x1700 字节；第三部分是代码段的后半部分，代码段的大部分代码都在这里；第四部分只读数据区；第五部分为可读写数据区；第六部分为 U-BOOT 命令区；最后一部分为未初始化数据段。

U-BOOT 命令的代码区仍然在 text 段，uboot_cmd 段存放的是一个数组，该数组中每个元素为一个结构体，里面存放着 U-BOOT 命令的信息，如名称、最多参数个数、入口函数等，在 include/command.h 中有它的定义。之所以单独设置一个 uboot_cmd 段的原因有二，一是可以使结构清晰，二是方便添加删除命令，只用宏 U_BOOT_CMD 就可以控制是否添加新命令。

【启动过程】
众所周知，U-BOOT 是存放在 FLASH 上的。系统启动时，CPU 会映射 FLASH 到它的内存

空间（映射一部分、还是全部 FLASH 空间？），然后执行 FLASH 上的代码。首先，进入 cpu/arc600/start.S 中的入口_start，进行内存初始化，接着把 U-BOOT 的前 0x1800 字节从 FLASH 复制到内存的 0x40800000 处，也就是链接时的地址；然后对 bss 段进行清零，设置堆栈指针，为运行 C 函数做准备；下一步，运行 C 函数检测在规定时间内是否有按键发生，如有则加载 boot 的后半部分(0x40801800——DATA_END)并启动 boot，无则加载 kernel 并启动 kernel。U-BOOT 启动的前半部分流程如图 2 所示：

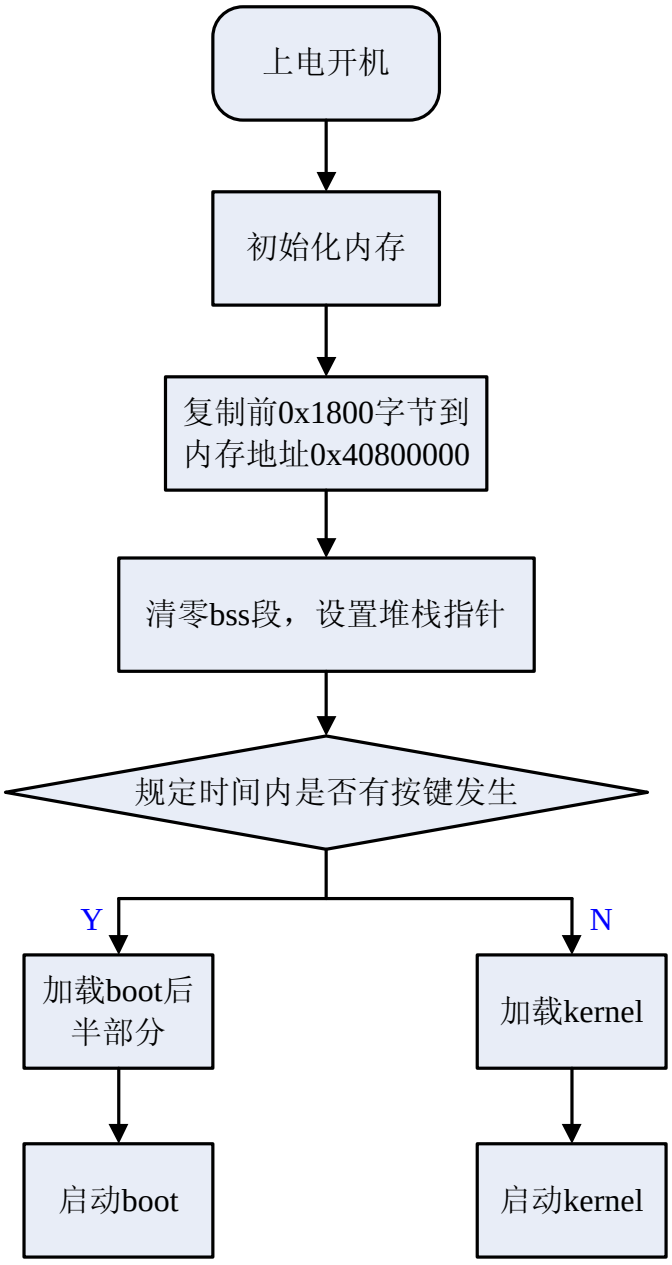

图 2 U-BOOT 启动流程（前半部分）

U-BOOT 启动的后半部分，会进行 heap、环境变量（env）的初始化，PHY 驱动的加载等工作，然后进入一个无限循环开始 shell 的运行，shell 运行过程中的内存示意如图 3 所示。其中，heap 和 stack 依次排列在 bss 段的后面，图中所示的 free area 则为 U-BOOT 未用到的内存。

图 3 中，heap 区域为 malloc()提供内存。在 uClib 库中，malloc()是通过 sbrk()或者 mmap()实现的，而 sbrk()和 mmap()是在内核中实现的。U-BOOT 作为系统最早运行的程序，没有内核的支持。为了实现 malloc()，它定义一个 32K 的 heap 区域，在此区域的基础上实现了简化版的 sbrk()。

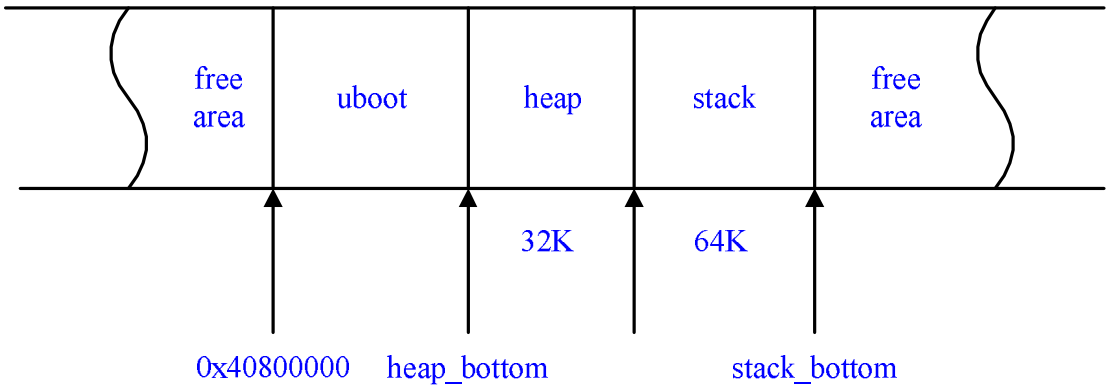

图 3 shell 运行时的内存示意

图 3 中，stack 区域是在 U-BOOT 启动的前半部分中第三步设置的。它首先根据 BSS_END、heap 大小和 stack 大小算出 stack_bottom 的值，然后设置堆栈指针 SP 和帧指针 FP 为 stack_bottom – 4。

由于水平有限，文中不免会出现疏漏。如发现问题，请发邮件告知我，谢谢！