U-Boot 启动过程分析

刘通平 Homepage: http://www.cs.umass.edu/~tonyliu/

4.4.1 U-Boot 启动位置

U-Boot 具有两种启动位置，既可以在 Flash 上启动也可以在内存中启动。但一般常见的启动位置是 U-Boot 在 Flash 上启动。
U-Boot 在内存中启动一般有两种情况：
（1） 先使用调试器对内存进行初始化，当内存可用时，再把 U-Boot 映像下载到内存中进行启动。这种方式通常适用于调试阶段。为了让内存等设备工作正常，必须注意调试器脚本中对内存的设置必须和 U-Boot 中对内存的初始化一致，否则容易发生 U-Boot 加载后内存没法正常工作的问题。

（2）热启动模式下 U-Boot 是在内存中进行启动的。在 U-Boot 中有些错误可能导致热启动，比如”bootm A B”，假如在 A，B 地址找不到 U-Boot 所能辨识的映像，这时就会发生热启动。或者看门狗定时器超时也会发生热启动。

而 U-Boot 在 Flash 中启动时，往往在处理器默认的启动地址放置 U-Boot 启动代码。对于不同的 CPU 而言，这些默认的启动地址是不同的。比如对于 MPC860 而言，处理器默认的启动地址为 0x00000100，因此需要把 U-Boot 映像放在这个地址。而对于 OMAP1610 而言，处理器默认的启动地址为 0x0 地址处，因此需要把 U-Boot 映像放在 0 地址处。

4.4.2 启动过程

此处阐述 U-Boot 中的启动过程，也许这里的阐述会和第 5 章的阐述有所重复，但此处更像是一种概述，和具体的 CPU 和目标板没有关系。
（1）设置 CPU 的控制寄存器，比如设置 CPU 的工作模式，控制寄存器的起始地址等。
（2）进行设备相关的初始化，比如内存，Flash，cache 等设备的初始化。通常，在开发时由于使用了调试器进行这些初始化的工作，因此这步也可以由调试器实现，而在具体的代码中跳过这段代码。比如，在 OMAP 平台下，提供了一个宏 CONFIG_INIT_CRITICAL 用于控制是否执行设备初始化代码 cpu_init_crit。
（3）准备堆栈，由于 U-Boot 还需要管理板级信息，内核命令行等信息，因此需要给这些信息预留内存空间，并初始化 U-Boot 相关的管理结构体。
（4）代码的拷贝。通常 U-Boot 代码保存在 Flash 设备上，因此需要把 U-Boot 代码拷贝到 RAM 中运行，因为 RAM 设备的速度较快。
（5）当代码拷贝和重定位完成后，会在内存中进行其它的一些初始化，主要指 U-Boot 管理结构体和其它外部设备的初始化。U-Boot 管理结构体的初始化，比如 malloc 空间，环境变量的初始化等。而其它外部设备是指除 SDRAM，Flash 外的其它设备，比如串口的初始化，定时器的初始化等。

（6）初始化完成后会进入 main_loop()函数，比如在预留一定的延时后，将执行默认的启动命令序列（即”bootcmd”中的相关设置）。当命令序列为有返回的指令序列或者没有预设的执行序列时，将等待界面发出的命令。当收到”bootm”或者”go”指令时，将设置相关的命令行参数和寄存器，然后启动内核并不再返回。