Bootloader分析

- 熟悉BootLoader的实现原理
- 认识Bootloader的主要任务
- 熟悉BootLoader的结构框架
- U-boot使用

引言

本章详细地介绍了基于嵌入式系统中的 OS 启动加载程序——Boot Loader 的概念、软件设计的主要任务以及结构框架等内容 。

一个嵌入式 Linux 系统从软件的角度看通常可以分为四个层次：

- 1. 引导加载程序。包括固化在固件(firmware)中的 boot 代码(可选)，和 Boot Loader 两大部分。
- 2. Linux 内核。特定于嵌入式板子的定制内核以及内核的启动参数。
- 3. 文件系统。包括根文件系统和建立于 Flash 内存设备之上文件系统。通常用 ram disk 来作为 root fs。
- 4. 用户应用程序。特定于用户的应用程序。有时在用户应用程序和内核层之间可能还会包括一个嵌入式图形用户界面。常用的嵌入式 GUI 有：MicroWindows 和 MiniGUI 。

- 引导加载程序是系统加电后运行的第一段软件代码。回忆一下 PC 的体系结构我们可以知道，PC 机中的引导加载程序由 BIOS(其本质就是一段固件程序)和位于硬盘 MBR 中的 OS Boot Loader（比如，LILO 和 GRUB 等）一起组成。

- BIOS 在完成硬件检测和资源分配后，将硬盘 MBR 中的 Boot Loader 读到系统的 RAM 中，然后将控制权交给 OS Boot Loader。Boot Loader 的主要运行任务就是将内核映象从硬盘上读到 RAM 中，然后跳转到内核的入口点去运行，也即开始启动操作系统。

  而在嵌入式系统中，通常并没有像 BIOS 那样的固件程序（注，有的嵌入式 CPU 也会内嵌一段短小的启动程序），因此整个系统的加载启动任务就完全由 Boot Loader 来完成。

  - **比如在一个基于 ARM7TDMI core 的嵌入式系统中，系统在上电或复位时通常都从地址 0x00000000 处开始执行，而在这个地址处安排的通常就是系统的 Boot Loader 程序。**

- 系统的启动通常有两种方式，一种是可以直接从Flash启动，另一种是可以将压缩的内存映像文件从Flash（为节省Flash资源、提高速度）中复制、解压到RAM，再从RAM启动。

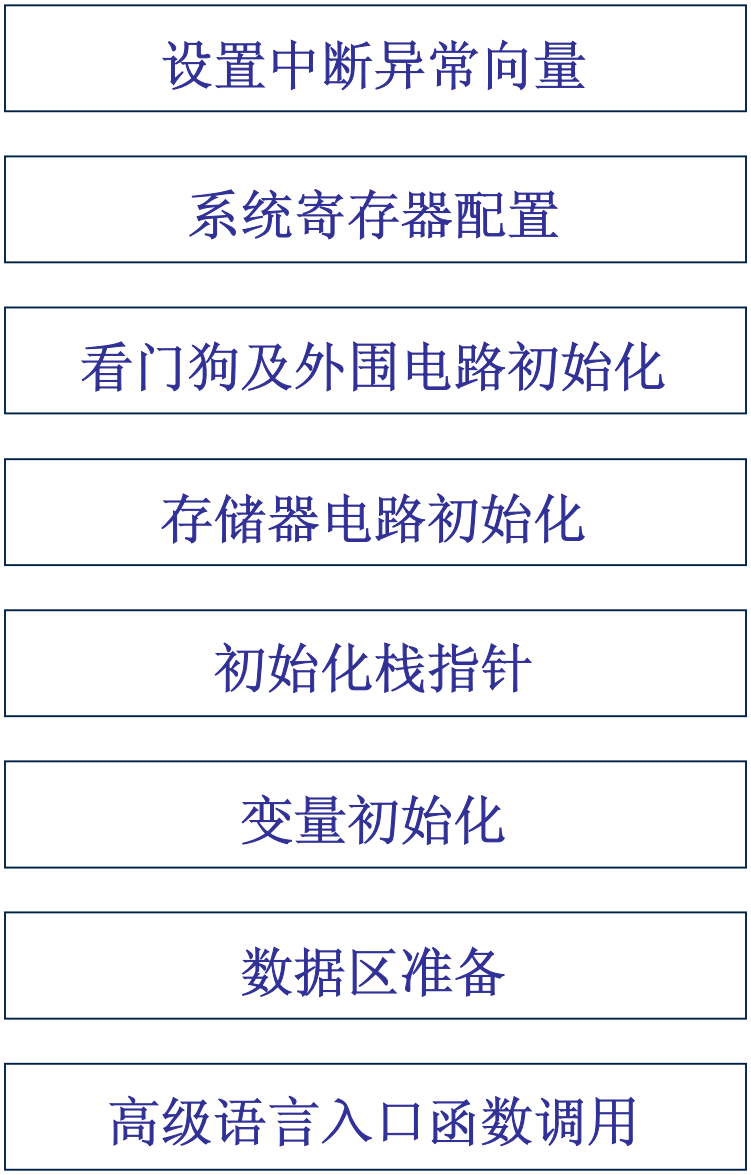
- 当电源打开时，一般的系统会去执行ROM（应用较多的是Flash）里面的启动代码。这些代码是用汇编语言编写的，其主要作用在于初始化CPU和板上的必备硬件如内存、中断控制器等。有时候用户还必须根据自己板子的硬件资源情况做适当的调整与修改。

- 系统启动代码完成基本软硬件环境初始化后，对于有操作系统的情况下，启动操作系统、启动内存管理、任务调度、加载驱动程序等，最后执行应用程序或等待用户命令；对于没有操作系统的系统直接执行应用程序或等待用户命令。

- 启动代码是用来初始化电路以及用来为高级语言写的软件做好运行前准备的一小段汇编语言，在商业实时操作系统中，启动代码部分一般被称为板级支持包，英文缩写为BSP。

- 它的主要功能就是：电路初始化和为高级语言编写的软件运行做准备。

  - 右图:嵌入式系统启动流程图



Bootloader概述

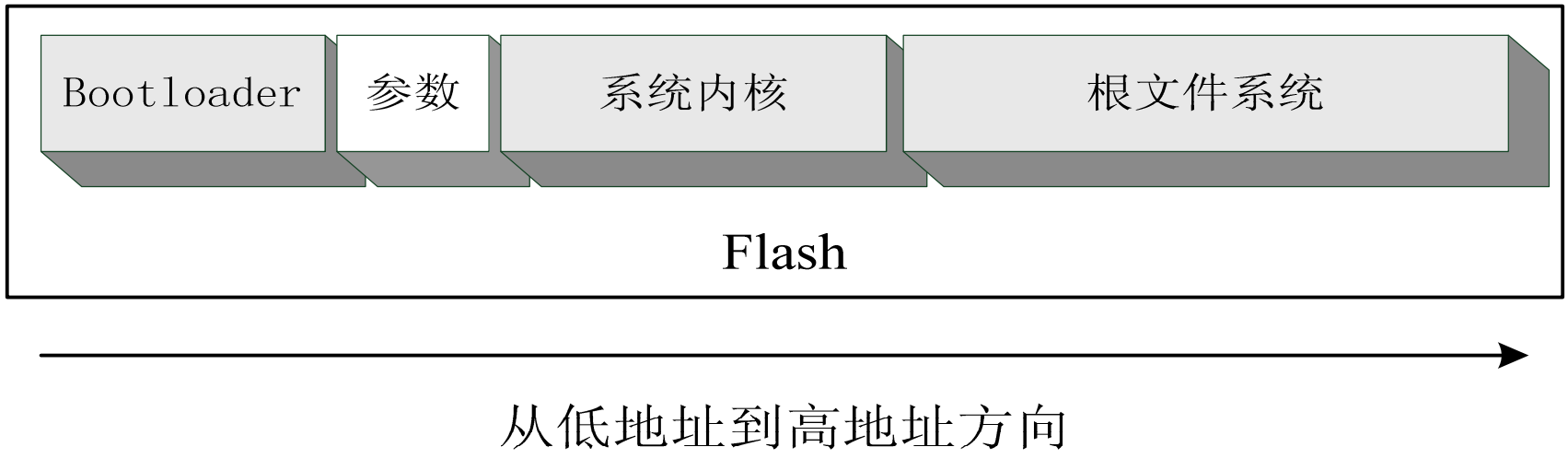
- 简单地说，Boot Loader 就是在操作系统内核运行之前运行的一段小程序。通过这段小程序，我们可以初始化硬件设备、建立内存空间的映射图，从而将系统的软硬件环境带到一个合适的状态，以便为最终调用操作系统内核准备好正确的环境。
- 通常，Boot Loader 是严重地依赖于硬件而实现的，特别是在嵌入式世界。因此，在嵌入式世界里建立一个通用的 Boot Loader 几乎是不可能的。尽管如此，我们仍然可以对 Boot Loader 归纳出一些通用的概念来，以指导用户特定的 Boot Loader 设计与实现。

1. Boot Loader 所支持的 CPU 和嵌入式板

- 每种不同的 CPU 体系结构都有不同的 Boot Loader。有些 Boot Loader 也支持多种体系结构的 CPU，比如 U-Boot 就同时支持 ARM 体系结构和MIPS 体系结构。

- 除了依赖于 CPU 的体系结构外，Boot Loader 实际上也依赖于具体的嵌入式板级设备的配置。这也就是说，对于两块不同的嵌入式板而言，即使它们是基于同一种 CPU 而构建的，要想让运行在一块板子上的 Boot Loader 程序也能运行在另一块板子上，通常也都需要修改 Boot Loader 的源程序。

- 2. Boot Loader 的安装媒介（Installation Medium）
  - 系统加电或复位后，所有的 CPU 通常都从某个由 CPU 制造商预先安排的地址上取指令。。而基于 CPU 构建的嵌入式系统通常都有某种类型的固态存储设备(比如：ROM、EEPROM 或 FLASH 等)被映射到这个预先安排的地址上。因此在系统加电后，CPU 将首先执行 Boot Loader 程序。
  - 下图就是一个同时装有 Boot Loader、内核的启动参数、内核映像和根文件系统映像的固态存储设备的典型空间分配结构图。



- 3. 用来控制 Boot Loader 的设备或机制
  - 主机和目标机之间一般通过串口建立连接，Boot Loader 软件在执行时通常会通过串口来进行 I/O，比如：输出打印信息到串口，从串口读取用户控制字符等。

- 4. Boot Loader 的启动过程是单阶段（Single Stage）还是多阶段（Multi-Stage）
  - 通常多阶段的 Boot Loader 能提供更为复杂的功能，以及更好的可移植性。从固态存储设备上启动的 Boot Loader 大多都是 2 阶段的启动过程，也即启动过程可以分为 stage 1 和 stage 2 两部分。而至于在 stage 1 和 stage 2 具体完成哪些任务将在下面讨论。

- 5. Boot Loader 的操作模式（Operation Mode）
  - 大多数 Boot Loader 都包含两种不同的操作模式：“启动加载”模式和“下载”模式，这种区别仅对于开发人员才有意义。但从最终用户的角度看，Boot Loader 的作用就是用来加载操作系统，而并不存在所谓的启动加载模式与下载工作模式的区别。
  - 启动加载（Boot loading）模式：
  - 这种模式也称为“自主”（Autonomous）模式。也即 Boot Loader 从目标机上的某个固态存储设备上将操作系统加载到 RAM 中运行，整个过程并没有用户的介入。
  - 下载（Downloading）模式：
  - 在这种模式下，目标机上的 Boot Loader 将通过串口连接或网络连接等通信手段从主机（Host）下载文件，比如：下载内核映像和根文件系统映像等。

- 6. BootLoader 与主机之间进行文件传输所用的通信设备及协议

  - 最常见的情况就是，目标机上的 Boot Loader 通过串口与主机之间进行文件传输，传输协议通常是 xmodem / ymodem / zmodem 协议中的一种。
  - 但是，串口传输的速度是有限的，因此通过以太网连接并借助 TFTP 协议来下载文件是个更好的选择。在通过以太网连接和 TFTP 协议来下载文件时，主机方必须有一个软件用来的提供 TFTP 服务。

BootLoader的结构框架

- 假定内核映像与根文件系统映像都被加载到 RAM 中运行.

- 另外，由于 Boot Loader 的实现依赖于 CPU 的体系结构，因此大多数 Boot Loader 都分为 stage1 和 stage2 两大部分。
  - 依赖于 CPU 体系结构的代码，比如设备初始化代码等，通常都放在 stage1 中，而且通常都用汇编语言来实现，以达到短小精悍的目的。
  - 而 stage2 则通常用C语言来实现，这样可以实现给复杂的功能，而且代码会具有更好的可读性和可移植性。

- Bootloader的stage1

  - Boot Loader 的 stage1 通常包括以下步骤(以执行的先后顺序)：
  - （1）硬件设备初始化。
  - （2）为加载 Boot Loader 的 stage2 准备 RAM 空间。
  - （3）拷贝 Boot Loader 的 stage2 到 RAM 空间中。
  - （4）设置好堆栈。
  - （5）跳转到 stage2 的 C 入口点。

- Bootloader 的 stage2 （1/2）

- 通常包括以下步骤(以执行的先后顺序)：
  - 初始化本阶段要使用到的硬件设备。
  - 检测系统内存映射(memory map)。
  - 将 kernel 映像和根文件系统映像从 flash 上读到 RAM 空间中。
  - 为内核设置启动参数。
  - 调用内核。

- Bootloader 的 stage2确 （2/2）
  - **stage2** 的代码通常用 **C** 语言来实现，以便于实现更复杂的功能和取得更好的代码可读性和可移植性。
  - 与普通 **C** 语言应用程序不同的是，在编译和链接 **Bootloader** 这样的程序时，我们不能使用 **glibc** 库中的任何支持函数。

- Bootloaer功能
  - 系统配置、中断接管、引导
  - 装载内核、根文件系统、参数传递、内核调试、内核和根文件系统的下载等等
- 常见的uClinux(Linux)的Bootloader：
  - Redboot
  - Blob
  - Vivi
  - Uboot
  - armBoot…

U-Boot 的常用命令的用法

- 进入U-Boot 控制界面后，可以运行各种命令，比如下载文件到内存，擦除、读写Flash，运行内存、NOR Flash、NAND Flash 中的程序，查看、修改、比较内存中的数据等。
- 使用各种命令时，可以使用其开头的若干个字母代替它。比如tftpboot命令，可以使用t、tf、tft、tftp 等字母代替，只要其他命令不以这些字母开头即可。
- 当运行一个命令之后，如果它是可重复执行的（代码中使用U_BOOT_CMD定义这个命令时，第3 个参数是1），若想再次运行可以直接输入回车。U-Boot 接收的数据都是十六进制，输入时可以省略前缀0x、0X。

（1）帮助命令help

- 运行help 命令可以看到U-Boot 中所有命令的作用，如果要查看某个命令的使用方法，运行“help 命令名”，比如“help bootm”。
  可以使用“?”来代替“help”，比如直接输入“?”、“? bootm”。

（2）下载命令

- U-Boot 支持串口下载、网络下载，相关命令有：loadb、loads、loadx、loady 和tftpboot、nfs。
- 前几个串口下载命令使用方法相似，以loadx 命令为例，它的用法为“loadx [off][baud]”。“[]”表示里面的参数可以省略，off 表示文件下载后存放的内存地址，baud 表示使用的波特率。如果baud 参数省略，则使用当前的波特率；如果off参数省略，存放的地址为配置文件中定义的宏CFG_LOAD_ADDR。

- tftpboot 命令使用TFTP 协议从服务器下载文件，服务器的IP 地址为环境变量serverip。
  用法为“tftpboot [loadAddress] [bootfilename]”，loadAddress 表示文件下载后存放的内存地址，bootfilename 表示要下载的文件的名称。如果loadAddress 省略，存放的地址为配置文件中定义的宏CFG_LOAD_ADDR；如果bootfilename 省略，则使用开发板的IP 地址构造一个文件名，比如开发板IP 为192.168.1.17，则默认的文件名为C0A80711.img。
- nfs 命令使用NFS 协议下载文件，用法为“nfs [loadAddress] [host ip addr:bootfilename]”。
  “loadAddress、bootfilename”的意义与tftpboot 命令一样，“host ip addr”表示服务器的IP 地址，默认为环境变量serverip。下载文件成功后，U-Boot 会自动创建或更新环境变量filesize，它表示下载的文件的长度，可以在后续命令中使用“$(filesize)”来引用它。

（3）内存操作命令

- 常用的命令有：查看内存命令md、修改内存命令mm、填充内存命令mw、复制命令cp。

- 这些命令都可以带上后缀“.b”、“.w”或“.l”，表示以字节、字（2个字节）、双字（4 个字节）为单位进行操作。比如“cp.l 30000000 31000000 2”将从开始地址0x30000000 处，复制2 个双字到开始地址为0x31000000 的地方。

- md 命令用法为“md[.b, .w, .l] address [count]”，表示以字节、字或双字（默认为双字）为单位，显示从地址address 开始的内存数据，显示的数据个数为count。
- mm 命令用法为“mm[.b, .w, .l] address”，表示以字节、字或双字（默认为双字）为单位，从地址address 开始修改内存数据。执行mm 命令后，输入新数据后回车，地址会自动增加，按“Ctrl+C”键退出。
- mw 命令用法为“mw[.b, .w, .l] address value [count]”，表示以字节、字或双字（默认为双字）为单位，往开始地址为address 的内存中填充count 个数据，数据值为value。
- cp 命令用法为“cp[.b, .w, .l] source target count”，表示以字节、字或双字（默认为双字）为单位，从源地址source 的内存复制count 个数据到目的地址的内存。

- （4）NOR Flash 操作命令
  常用的命令有查看Flash 信息的flinfo 命令、加/解写保护命令protect、擦除命令erase。
- 由于NOR Flash 的接口与一般内存相似，所以一些内存命令可以在NOR Flash 上使用，比如读NOR Flash 时可以使用md、cp 命令，写NOR Flash 时可以使用cp 命令（cp 根据地址分辨出是NOR Flash，从而调用NOR Flash 驱动完成写操作）。
- 直接运行“flinfo”即可看到NOR Flash 的信息，有NOR Flash 的型号、容量、各扇区的开始地址、是否只读等信息。
  对于只读的扇区，在擦除、烧写它之前，要先解除写保护。最简单的命令为“protect off all”，解除所有NOR Flash 的写保护。
- erase 命令常用的格式为“erase start end”，擦除的地址范围为start～end；“erase start +len”，擦除的地址范围为start～（star+len），“erase all”，表示擦除所有NOR Flash。

（5）NAND Flash 操作命令

NAND Flash 操作命令只有一个：nand，它根据不同的参数进行不同操作，比如擦除、读取、烧写等。

- “nand info”查看NAND Flash 信息。
- “nand erase [clean] [off size]”擦除NAND Flash。加上“clean”时，表示在每个块的第一个扇区的OOB 区加写入清除标记；off、size表示要擦除的开始偏移地址的长度，如果省略off 和size，表示要擦除整个NAND Flash。
- “nand read[.jffs2] addr off size”从NAND Flash 偏移地址off 处读出size 个字节的数据存放到开始地址为addr中。是否加后缀“.jffs”的差别只是读操作时的ECC 校验方法不同。

- “nand write[.jffs2] addr off size”把开始地址为addr 的内存中的size 个字节数据写到NAND Flash 的偏移地址off 处。是否加后缀“.jffs”的差别只是写操作时的ECC 校验方法不同。
- “nand read.yaffs addr off size”从NAND Flash 偏移地址off 处读出size 个字节的数据（包括OOB 区域），存放到开始地址为addr 的内存中。
- “nand write.yaffs addr off size”把开始地址为addr 的内存中的size 个字节数据（其中有要写入OOB 区域的数据）写到NAND Flash 的偏移地址off 处。
- “nand dump off”将NAND Flash 偏移地址off 的一个扇区的数据打印出来，包括OOB 数据。

- （6）环境变量命令。

- “printenv”命令打印全部环境变量，“printenv name1 name2?”打印名字为name1、name2、?的环境变量。

- “setenv name value”设置名字为name 的环境变量的值为value。

- “setenv name”删除名字为name 的环境变量。

- 上面的设置、删除操作只是在内存中进行，saveenv将更改后的所有环境变量写入NOR Flash 中。

- U-boot# printenv
  - bootargs=root=ubi0:FriendlyARM-root ubi.mtd=2 rootfstype=ubifs init=/linuxrc console=ttySAC0,115200
  - bootcmd=nand read.i c0008000 80000 500000;bootm c0008000
  - bootdelay=1
  - baudrate=115200
  - ethaddr=08:90:90:90:90:90
  - ipaddr=192.168.1.230
  - serverip=192.168.1.88
  - gatewayip=192.168.1.1
  - netmask=255.255.255.0
  - stdin=serial
  - stdout=serial
  - stderr=serial

- （7）启动命令
  不带参数的“boot”、“bootm”命令都是执行环境变量bootcmd 所指定的命令。

- “bootm [addr [arg…]]”命令启动存放在地址addr 处的U-Boot 格式的映象文件（使用U-Boot 目录tools 下的mkimage 工具制作得到），[arg…]表示参数。如果addr 参数省略，映象文件所在地址为配置文件中定义的宏CFG_LOAD_ADDR。

- “go addr [arg…]”与bootm 命令类似，启动存放在地址addr 处的二进制文件，[arg...]表示参数。

- “nboot [[[loadAddr] dev] offset]”命令将NAND Flash 设备dev 上偏移地址off 处的映象文件复制到内存loadAddr 处，然后，如果环境变量autostart 的值为“yes”，就启动这个映象。

- 如果loadAddr 参数省略，存放地址为配置文件中定义的宏CFG_LOAD_ADDR；如果dev 参数省略，则它的取值为环境变量bootdevice的值；如果offset 参数省略，则默认为0.

配置编译U-boot

- 配置编译支持NAND 启动的U-boot

- 说明：根据开发板不同的内存(DDR RAM)容量，需要使用不同的U-boot配置项。
- 要编译适合于 128M 内存的U-boot，请按照以下步骤：
- 进入 U-boot 源代码目录，执行：
- #cd /opt/u-boot-mini6410
- #make mini6410_nand_config-ram128;make
- 将会在当前目录配置并编译生成支持Nand 启动的U-boot.bin，使用SD卡或者USB下载到Nand Flash 即可使用.


END
Embedded +software Development
Tony ,vopo123@126.com