# 引导加载

嵌入式系统是我们的研究对象。本系列实验中, 我们以BeagleBone Black和树莓派3B作为典型案例分析。在宿主机—目标机开发模式里, 它们是目标机, 或叫做“目标系统”, 在实验环境下, 又习惯地把它们叫做“开发板”。

## 引导加载的作用

PC机中的引导加载(Bootloader)程序由BIOS和位于硬盘的主引导记录MBR(Master Boot Recorder)中的OS Bootloader共同组成。BIOS在完成硬件检测和资源分配后, 将硬盘MBR 中的Bootloader读到系统的RAM中, 然后将控制权交给OS Bootloader。Bootloader的主要运行任务就是将内核映像文件从硬盘上读到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。

嵌入式系统中, 通常并没有像BIOS那样的固件程序, 因此整个系统的加载启动任务完全由Bootloader来完成。用于引导嵌入式操作系统的Bootloader有U-Boot、vivi、RedBoot等等。Bootloader的主要作用是:

1. 初始化硬件设备;
2. 建立内存空间的映射图;
3. 完成内核的加载, 为内核设置启动参数。

嵌入式系统中的Bootloader的实现完全依赖于CPU的体系结构, 因此大多数Bootloader都分两个阶段。依赖于CPU体系结构的代码, 比如设备初始化代码等, 通常都放在阶段一中, 且通常都用汇编语言来实现, 以便完成一些高级语言不能完成的工作。阶段一通常包括以下步骤:

1. 硬件设备初始化;
2. 拷贝Bootloader的程序到RAM空间中;
3. 设置堆栈;
4. 跳转到阶段二的C程序入口点。

阶段二则通常用C语言来实现, 这样可以实现一些复杂的功能, 而且代码会具有更好的可读性和可移植性。这一阶段主要包括以下步骤:

1. 初始化本阶段要使用到的硬件设备;
2. 系统内存映射(Memory Map);
3. 将内核映像和根文件系统映像读到RAM空间中;
4. 为内核设置启动参数;
5. 跳转到内核入口点, 内核启动。

Bootloader也是嵌入式Linux系统开发过程中差异最大的地方。以下分别介绍BeagleBone Black和树莓派的引导启动过程。

## BeagleBone Black

### BeagleBoard简介

BeagleBone Black是基于TI的嵌入式处理器Sitara AM335X设计的单板计算机, Cortex-A8架构, 主频1GHz, 板载512M DDR3L内存和2GiB/4GiB eMMC FLASH存储器, 3D图形引擎SGX530, 采用6层板工艺设计。元器件布局见图1.1, 板载接口及功能列表如下:

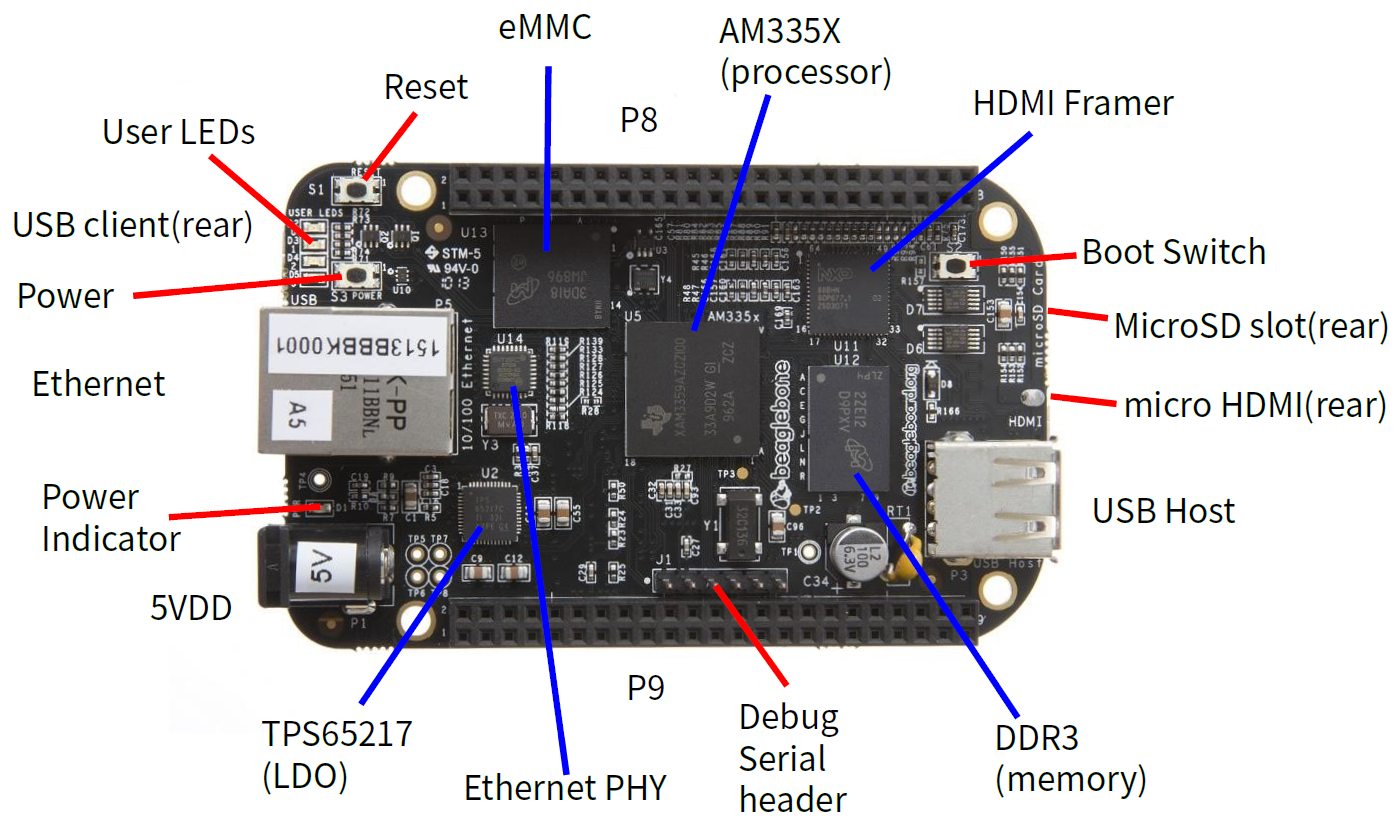
* 有线以太网:10/100M, RJ45, LAN8720
* microSD卡接口(3.3V)
* 电源管理:TPS65217PMIC
* USB Client(USB0, mini-USB),
* USB HOST(USB1)
* HDMI高清视频输出接口(Max. 1920x1080@24Hz)
* 音频输出:通过HDMI
* 3D高性能图形加速SGX530
* UART0调试接口
* 两组2×23扩展接口:McASP, SPI, I2C, LCD, MMC, GPIO, ADC等

BeagleBoard出厂时已安装了Bootloader(U-Boot)和Linux操作系统。如果需要从外接microSD卡上启动系统, 或者重新安装Bootloader, 就需要编译U-Boot了。

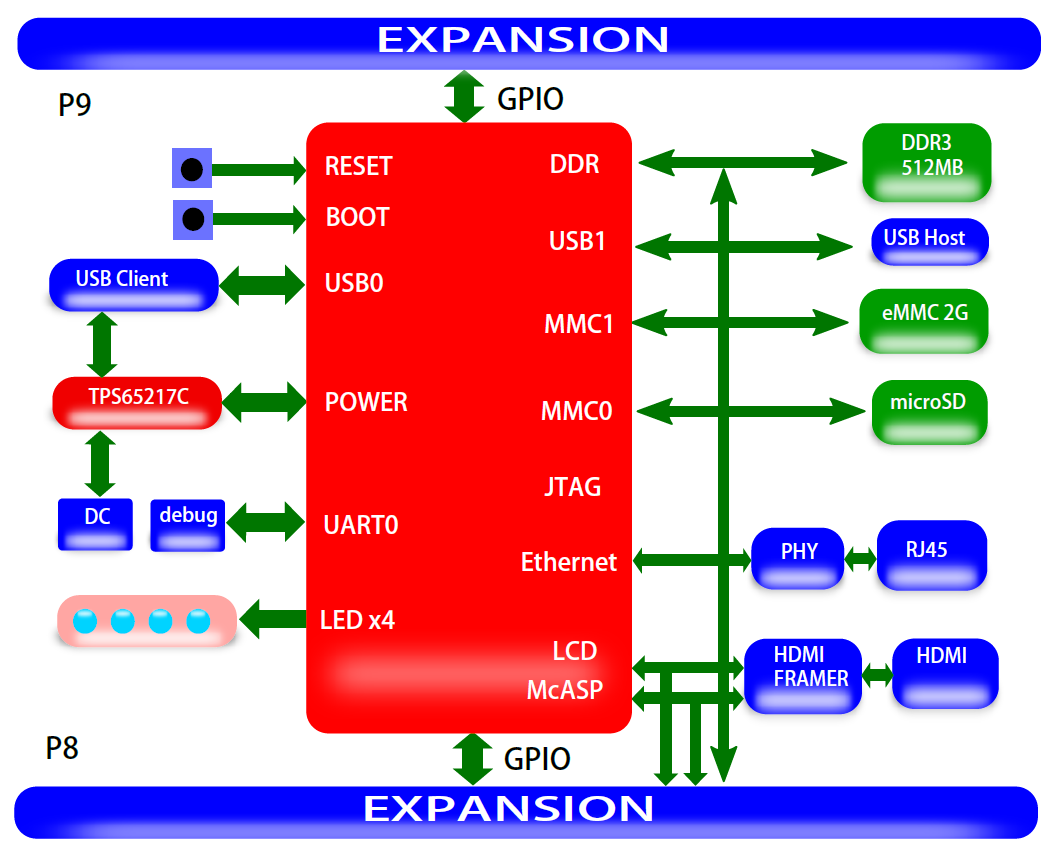
### 编译U-Boot

很多嵌入式Linux使用U-Boot引导。U-Boot源码在<https://gitlab.denx.de/u-boot>, 使用下面的命令克隆项目:

$ git clone <https://gitlab.denx.de/u-boot/u-boot.git>



BeagleBone Black布局



BeagleBone Black框图

BeagleBone Black使用am335x\_evm\_defconfig的缺省配置文件。在下载的U-boot目录中, 执行make am335x\_evm\_defconfig, 即完成针对BeagleBone Black的缺省Bootloader配置。在此基础上, 还可以用make menuconfig或make gconfig进入配置界面, 对Bootloader的单个选项进行合理的取舍。前者为字符菜单界面(图1.3a), 后者是图形配置界面(图1.3b)。完成配置后, 用下面的命令编译:

$ make ARCH=arm CROSS\_COMPILE=arm-linux-

编译完成后, 生成MLO和u-boot.img。

这里用到了交叉编译器arm-linux-gcc。交叉编译器将在下一章详细讨论。



1. make menuconfig



1. make gconfig

U-Boot配置界面

### 制作microSD卡

将前面生成的MLO和u-boot.img用下面的命令写入microSD卡:

# dd if=MLO of=/dev/mmcblk0 count =1 seek=1 bs=128k

# dd if=u-boot.img of=/dev/mmcblk0 count =2 seek=1 bs=384k

设备名/dev/mmcblkX因时而异。如果通过USB转接器转接, 设备名也可能是/dev/sdbX, 请

认清操作对象。错误的操作可能会破坏重要数据。BeagleBone Black上电时按住S2(图1.1 BootSwitch), 系统首先选择MMC0(microSD卡插槽)启动。

再在该卡上创建两个文件系统分区。一个引导分区, 用于存放启动内核相关的固件及内核映像, 用VFAT格式化;一个Linux分区, 用于准备作为根文件系统, 用inode型文件系统格式化(Ext4FS、BtrFS、ReiserFS等等均可, 要求内核支持)。内核映像文件一般不超过10MB, RAM Disk映像文件、设备树文件等等也在10MB以内, 可以以此估算VFAT分区的大小。剩余空间作为根文件系统, 按Ext4分区格式化。图1.4是在一个2GB microSD卡上规划的分区方案。

命令(输入m获取帮助):p

Disk/dev/sdb:1.9GiB, 2032664576字节, 3970048个扇区

单元:扇区/1\*512=512字节

扇区大小(逻辑/物理):512字节/512字节

I/O大小(最小/最佳):512字节/512字节

磁盘标签类型:dos

磁盘标识符:0xc66ac9af

设备 启动 起点 末尾 扇区 大小 Id 类型

/dev/mmcblk0p1 2048 133119 131072 64M c W95 FAT32 (LBA)

/dev/mmcblk0p2 133120 3970047 3836928 1.8G 83 Linux

2GB microSD卡的分区方案(使用fdisk/dev/sdb命令)

注意在分区时不要覆盖之前写入microSD卡的裸数据所在扇区(即第一个分区的起点不要太低)。保存分区表、退出, 用下面的命令格式化:

# mkfs.vfat /dev/mmcblk0p1

# mkfs.ext4 /dev/mmcblk0p2

如果要求系统能自动引导操作系统, 还需要在引导分区上建立U-Boot的脚本文件。默认的脚本文件名是uEnv.txt。它至少应包含下面的内容:

* 将指定的内核映像文件加载到内存的指定位置;
* 设定向内核传递的启动参数, 特别重要的是根文件系统的类型和位置;
* 跳转到内核解压的内存地址开始运行, 向内核交权。

一个典型的uEnv.txt内容如下:

loadkernel=fatload mmc 0 0x82000000 zImage

loadfdt=fatload mmc 0 0x88000000 /dtbs/am335x -boneblack.dtb

rootfs=root=/dev/mmcblk0p2 rw rootfstype=ext4

loadfiles=run loadkernel; run loadfdt

mmcargs=setenv bootargs console=ttyO0 , 115200 n8 ${rootfs}

uenvcmd=run loadfiles; run mmcargs; bootz 0x82000000 - 0x88000000

以上表示把eMMC设备0分区的zImage文件读入内存0x82000000、dtbs/am335xboneblack.dtb 读入0x88000000, 指定根文件系统/dev/mmcblk0p2, 设置启动参数bootargs, 最后通过boot或bootz命令启动内核。bootz可带三个参数, 依次是内核地址、根文件系统地址、设备树地址。当某项参数缺失时, 用符号“-”填充。此处bootz的第二个参数由bootargs的root参数指定。

如果需要把Bootloader刷到板载eMMC FLASH, 可在Beagleboard Linux系统启动后

(外扩MMC0启动或板载eMMC启动), 使用dd命令把编译U-Boot生成的文件MLO和

u-boot.img写入eMMC设备。

### U-Boot用法

U-Boot的主要任务是引导操作系统。但在开发阶段, 连接串口监视、系统上电后的短时间内, 在minicom窗口通过人工干预, 也可以进入U-Boot的交互环境, 出现“U-Boot#”提示符, 在这个提示符下通过人工操作尝试系统启动。当确定能够完成正确的启动后, 再把这些操作写入uEnv.txt文件, 让U-Boot上电后自动执行。

以下列出本实验常用的命令:

* setenv:设置环境变量。主要的环境变量有:
* ipaddr: 本机IP 地址, serverip: TFTP服务器IP 地址, gatewayip: 网关。如果本机和服务器在同一个子网, 可以不设gatewayip。

U-Boot# set ipaddr 192.168.2.123

U-Boot# set serverip 192.168.2.23

U-Boot# set gatewayip 192.168.2.1

U-Boot的命令具有自动补全功能, 即如果开头的几个字母能指向一个唯一存在的命令,敲入制表符键TAB, 这条命令的后面几个字母就自动填补上了; 也可以不用制表符键补全, 直接用开头几个字母作为命令的替代(但需要确定其唯一性)。这里的“set” 即表示“setenv”。

* bootargs: 启动参数, 一般包括监控端口、内核启动参数、加载文件系统等, 如:

U-Boot# set bootargs console=ttyO0,115200n8 root=/dev/mmcblk0p2 rw →rootfstype=ext4 init=/linuxrc

参数格式“arg=val”, 参数项之间用空格分隔。上面表示用串口设备ttyO0作为终端, 波特率115200bps, 无校验位, 8位数据位, 根文件系统在板载eMMC FLASH的第二分区, 以读写允许方式挂载, ext4文件系统, 内核启动后执行根目录下的linuxrc命令。

* bootcmd, 启动命令, 上电或者执行boot命令时调用。如

U-Boot# set bootcmd "fatload mmc 0 0x82000000 zImage;bootz 0x82000000"

表示将eMMC 卡的第一分区(FAT 文件系统) 内核映像文件zImage 读到内存0x82000000起始的地址中, 然后从0x82000000处开始运行。

* bootdelay: 启动延迟秒数, 在上电后的这段时间里可以通过键盘干预, 打断正常引导过程。
* tftpboot: tftp文件传输, 如:

U-Boot# tftp 0x82000000 zImage

将TFTP服务器目录中的文件zImage通过TFTP协议读入内存0x82000000起始处。