https://www.vxworks.net

搞嵌入式的，为啥要有 uboot

为什么要有 uboot

1.1、计算机系统的主要部件

(1)计算机系统就是以 CPU 为核心来运行的系统。典型的计算机系统有：PC 机（台式机+笔记本）、嵌入式设备（手机、平板电脑、游戏机）、单片机（家用电器像电饭锅、空调）

(2)计算机系统的组成部件非常多，不同的计算机系统组成部件也不同。但是所有的计算机系统运行时需要的主要核心部件都是 3 个东西：CPU+外部存储器（Flash/硬盘）+内部存储器（DDR SDRAM/SDRAM/SRAM）

1.2、PC 机的启动过程

(1)部署：典型的 PC 机的 BIOS 程序部署在 PC 机主板上（随主板出厂时已经预制了），操作系统部署在硬盘上，内存在掉电时无作用，CPU 在掉电时不工作。

(2)启动过程：PC 上电后先执行 BIOS 程序（实际上 PC 的 BIOS 就是 NorFlash），BIOS 程序负责初始化 DDR 内存，负责初始化硬盘，然后从硬盘上将 OS 镜像读取到 DDR 中，然后跳转到 DDR 中去执行 OS 直到启动（OS 启动后 BIOS 就无用了）

1.3、典型嵌入式 linux 系统启动过程

(1)典型嵌入式系统的部署：uboot 程序部署在 Flash（能作为启动设备的 Flash）上、OS 部署在 FLash（嵌入式系统中用 Flash 代替了硬盘）上、内存在掉电时无作用，CPU 在掉电时不工作。

(2)启动过程：嵌入式系统上电后先执行 uboot、然后 uboot 负责初始化 DDR，初始化 Flash，然后将 OS 从 Flash 中读取到 DDR 中，然后启动 OS（OS 启动后 uboot 就无用了）

总结：嵌入式系统和 PC 机的启动过程几乎没有两样，只是 BIOS 成了 uboot，硬盘成了 Flash。

1.4、android 系统启动过程

(1)Android 系统的启动和 Linux 系统（前面讲的典型的嵌入式系统启动）几乎一样。几乎一样意思就是前面完全一样，只是在内核启动后加载根文件系统后不同了。

(2)可以认为启动分为 2 个阶段：第一个阶段是 uboot 到 OS 启动；第二个阶段是 OS 启动后到 rootfs 加载到命令行执行；现在我们主要研究第一个阶段，android 的启动和 linux 的差别在第二阶段。

1.5、总结：uboot 到底是干嘛的

https://www.vxworks.net

(1) uboot 主要作用是用来启动操作系统内核。

(2) uboot 还要负责部署整个计算机系统。

(3) uboot 中还有操作 Flash 等板子上硬盘的驱动。

(4) uboot 还得提供一个命令行界面供人来操作。

为什么是 uboot

2.1、uboot 从哪里来的？(1)uboot 是 SourceForge 上的开源项目(2)uboot 项目的作者：一个德国人最早发起的项目(3)uboot 就是由一个人发起，然后由整个网络上所有感兴趣的人共同维护发展而来的一个 bootloader。

2.2、uboot 的发展历程

(1)自己使用的小开源项目。

(2)被更多人认可使用

(3)被 SoC 厂商默认支持。

总结：uboot 经过多年发展，已经成为事实上的业内 bootloader 标准。现在大部分的嵌入式设备都会默认使用 uboot 来做为 bootloader。

2.3、uboot 的版本号问题

(1)早期的 uboot 的版本号类似于这样：uboot1.3.4。后来版本号便成了类似于 uboot-2010.06。
(2)uboot 的核心部分几乎没怎么变化，越新的版本支持的开发板越多而已，对于一个老版本的芯片来说，新旧版本的 uboot 并没有差异。

2.4、uboot 的可移植性的正确理解

(1)uboot 就是 universal bootloader（通用的启动代码），通用的意思就是在各种地方都可以用。所以说 uboot 具有可移植性。

(2)uboot 具有可移植性并不是说 uboot 在哪个开发板都可以随便用，而是说 uboot 具有在源代码级别的移植能力，可以针对多个开发板进行移植，移植后就可以在这个开发板上使用了。

uboot 必须解决哪些问题

3.1、自身可开机直接启动

(1)一般的 SoC 都支持多种启动方式，譬如 SD 卡启动、NorFlash 启动、NandFlash 启动等……uboot 要能够开机启动，必须根据具体的 SoC 的启动设计来设计 uboot。

(2)uboot 必须进行和硬件相对应的代码级别的更改和移植，才能够保证可以从相应的启动介质启动。uboot 中第一阶段的 start.S 文件中具体处理了这一块。

3.2、能够引导操作系统内核启动并给内核传参

(1)uboot 的终极目标就是启动内核。

(2)linux 内核在设计的时候，设计为可以被传参。也就是说我们可以在 uboot 中事先给 linux 内核准备一些启动参数放在内存中特定位置然后传给内核，内核启动后会到这个特定位置去取 uboot 传给他的参数，然后在内核中解析这些参数，这些参数将被用来指导 linux 内核的启动过程。

3.3、能提供系统部署功能

(1)uboot 必须能够被人借助而完成整个系统（包括 uboot、kernel、rootfs 等的镜像）在 Flash 上的烧录下载工作。

(2)裸机教程中刷机（ARM 裸机第三部分）就是利用 uboot 中的 fastboot 功能将各种镜像烧录到 iNand 中，然后从 iNand 启动。

3.4 能进行 soc 级和板级硬件管理

(1)uboot 中实现了一部分硬件的控制能力（uboot 中初始化了一部分硬件），因为 uboot 为了完成一些任务必须让这些硬件工作。譬如 uboot 要实现刷机必须能驱动 iNand，譬如 uboot 要在刷机时 LCD 上显示进度条就必须能驱动 LCD，譬如 uboot 能够通过串口提供操作界面就必须驱动串口。譬如 uboot 要实现网络功能就必须驱动网卡芯片。

(2)SoC 级（譬如串口）就是 SoC 内部外设，板级就是 SoC 外面开发板上面的硬件（譬如网卡、iNand）

3.5、uboot 的“生命周期”

(1)uboot 的生命周期就是指：uboot 什么时候开始运行，什么时候结束运行。

(2)uboot 本质上是一个裸机程序（不是操作系统），一旦 uboot 开始 SoC 就会单纯运行 uboot（意思就是 uboot 运行的时候别的程序是不可能同时运行的），一旦 uboot 结束运行则无法再回到 uboot（所以 uboot 启动了内核后 uboot 自己本身就死了，要想再次看到 uboot 界面只能重启系统。重启并不是复活了刚才的 uboot，重启只是 uboot 的另一生）

(3)uboot 的入口和出口。uboot 的入口就是开机自动启动，uboot 的唯一出口就是启动内核。uboot 还可以执行很多别的任务（譬如烧录系统），但是其他任务执行完后都可以回到 uboot 的命令行继续执行 uboot 命令，而启动内核命令一旦执行就回不来了。

总结：一切都是为了启动内核

uboot 的工作方式

4.1、从裸机程序镜像 uboot.bin 说起

(1)uboot 的本质就是一个裸机程序，和我们裸机全集中写的那些裸机程序 xx.bin 并没有本质区别。如果非说要有区别，那就是：我们写的大部分小于 16KB，而 uboot 大于 16KB（一般 uboot 在 180k-400k 之间）

(2)uboot 本身是一个开源项目，由若干个.c 文件和.h 文件组成，配置编译之后会生成一个 uboot.bin，这就是 uboot 这个裸机程序的镜像文件。然后这个镜像文件被合理的烧录到启动介质中拿给 SoC 去启动。也就是说 uboot 在没有运行时表现为 uboot.bin，一般躺在启动介质中。

(3)uboot 运行时会被加载到内存中然后一条指令一条指令的拿给 CPU 去运行。

4.2、uboot 的命令式 shell 界面

(1)普通的裸机程序运行起来就直接执行了，执行时效果和代码有关。

(2)有些程序需要和人进行交互，于是乎程序中就实现了一个 shell（shell 就是提供人机交互的一个界面，回想 ARM 裸机全集第十六部分），uboot 就实现了一个 shell。注意：shell 并不是操作系统，和操作系统一点关系都没有。linux 中打开一个终端后就得到了一个 shell，可以输入命令回车执行。uboot 中的 shell 工作方式和 linux 中的终端 shell 非常像（其实几乎是一样的，只是命令集不一样。譬如 linux 中可以 ls，uboot 中 ls 就不识别）

4.3、掌握 uboot 使用的 2 个关键点：命令和环境变量

(1)uboot 启动后大部分时间和工作都是在 shell 下完成的（譬如 uboot 要部署系统要在 shell 下输命令、要设置环境变量也得在命令行地下，要启动内核也要在命令行底下敲命令）

(2)命令就是 uboot 的 shell 中可以识别的各种命令。uboot 中有几十个命令，其中有一些常用另一些不常用（我们还可以自己给 uboot 添加命令），后面会用几节课时间来依次学习 uboot 中常用命令。
(3)uboot 的环境变量和操作系统的环境变量工作原理和方式几乎完全相同。uboot 在设计时借助了操作系统的设计理念（命令行工作方式借鉴了 linux 终端命令行，环境变量借鉴了操作系统的环境变量，uboot 的驱动管理几乎完全照抄了 linux 的驱动框架）。

(4)环境变量可以被认为是系统的全局变量，环境变量名都是系统内置的（认识就认识，不认识就不认识，这部分是系统自带的默认的环境变量，譬如 PATH；但是也有一部分环境变量是自己添加

的，自己添加的系统就不认识但是我们自己认识）。系统或者我们自己的程序在运行时可以通过读取环境变量来指导程序的运行。这样设计的好处就是灵活，譬如我们要让一个程序更改运行方法，不用去重新修改程序代码再重新编译运行，而只要修改相应的环境变量就可以了。

(5)环境变量就是运行时的配置属性。

uboot 的常用命令

5.1、类似 linux 终端的行缓冲命令行

(1) 行缓冲的意思就是：当我们向终端命令行输入命令的时候，这些命令没有立即被系统识别，而是被缓冲到一个缓存区（也就是系统认为我们还没有输入完），当我们按下回车键（换行）后系统就认为我们输入完了，然后将缓冲区中所有刚才输入的作为命令拿去分析处理。

(2) Linux 终端设计有 3 种缓冲机制：无缓冲、行缓冲、全缓冲

(3) 有些命令有简化的别名，譬如 printenv 命令可以简化为 print，譬如 setenv 可以简化为 set

(4) 有些命令会带参数（注意格式是固定的），uboot 的每个命令都有事先规定好的各种格式。有些命令就是不带参数的，譬如 printenv/print 命令；有些命令带可选的参数（可以带也可以不带，当然带不带参数的执行结果是不同的）；有些命令带必须的参数（譬如 setenv/set 命令）

(5) 采用“help+命令名”来查询命令的详细信息，只输入 help 时，则打印出命令列表。

5.2、命令中的特殊符号（譬如单引号）

(1)uboot 的有些命令带的参数非常长，为了告诉 uboot 这个非常长而且中间有好多个空格的东西是给他的一整个参数，所以用单引号将这个很长且中间有空格隔开的参数引起来。

(2)别的符号也许也有，而且有特定的意义。当碰到 uboot 的命令行有特殊符号时要注意不是弄错了，而是可能有特别的含义。

5.3、有些命令是一个命令族（譬如 movi）

(1)命令族意思就是好多个命令开头都是用同一个命令关键字的，但是后面的参数不一样，这些命令的功能和作用也不同。这就叫一个命令族。

(2)同一个命令族中所有的命令都有极大的关联，譬如 movi 开头的命令族都和 moviNand（EMMC、iNand）操作有关。

https://www.vxworks.net

5.4、第一个命令：printenv/print

(1)print 命令不用带参数，作用是打印出系统中所有的环境变量。

(2)环境变量就好像程序的全局变量一样。程序中任何地方都可以根据需要去调用或者更改环境变量（一般都是调用），环境变量和全局变量不同之处在于：全局变量的生命周期是在程序的一次运行当中，开始运行时诞生程序结束时死亡，下次运行程序时从头开始；但是环境变量被存储在 Flash 的另一块专门区域（Flash 上有一个环境变量分区），一旦我们在程序中保存了该环境变量，那么下次开机时该环境变量的值将维持上一次更改保存后的值。

uboot 的常用命令 2

1、设置（添加/更改）环境变量：setenv/set 用法：set name value

2、保存环境变量的更改：saveenv/savesaveenv/save 命令不带参数，直接执行，作用是将内存中的环境变量的值同步保存到 Flash 中环境变量的分区。注意：环境变量的保存是整体的覆盖保存，也就是说内存中所有的环境变量都会整体的将 Flash 中环境变量分区中原来的内容整体覆盖。总结：彻底更改一个环境变量的值，需要 2 步：第一步：set 命令来更改内存中的环境变量第二步：用 save 命令将其同步到 Flash 中环境变量的分区。有时候我们只是想测试下这个环境变量，不希望影响到下一次开机，那就只 set 不 save，这样 set 后当前本次运行的 uboot 已经起效果了，只不过没 save 下一次开机还是会恢复到原来的状况。

3、网络测试指令：ping(1)命令用法： ping ip 地址注意：ping 是测试开发板和主机之间的网络链接，注意以下步骤：1)首先要插上网线。2)先试图 ping 通主机 windows。注意 Windows 中有线网卡的地址设置（设置本地连接）。设置主机 windows 的本地连接 IPv4 地址为 192.168.1.103)第三步确认开发板中 uboot 里几个网络相关的环境变量的值对不对。最重要的是 ipaddr（这个环境变量表示当前开发板的 IP 地址），这个地址必须和主机 windows 的 IP 地址在同一个网段。网段的概念：一个 IP 地址分为 2 部分，一部分是网段地址，另一部分是网段内的主机地址（由子网掩码来区分哪一部分是网段地址，哪一部分是 IP 地址）。在子网掩码是 255.255.255.0 的情况下，192.168.1.10 这个 IP 地址的前三部分（192.168.1.）属于网段地址，第 4 部分（10）属于主机地址。

开发板和主机的 ping 通

上节课最后的结果是：uboot 中的 ipaddr 和主机 windows 本地连接地址已经设置到一个网段，但是实际还 ping 不通。还发现了这样的现象：

1、我把 2 个的网段都从 192.168.1.x 改到 192.168.0.x 时会 ping 通一次，第二次开始就 ping 不通了
2、有同学说 ping 不通可能是因为 uboot 中 gatewayip 没设置，我就实际测试设置网管为同网段.1，再次测试结论是第一次 ping 通了，第二次开始又不通了。

7.1、开发板运行 linux 下和主机 Windows 的 ping 通

(1)先将开发板刷机成 linux+QT 镜像（刷机见裸机教程第三部分），然后启动进入 linux 命令行终端下。
(2)在 linux 下使用 ifconfig 命令将开发板中 linux 系统的 IP 地址设置为和主机 windows 同一网段（为了上课方便，以后就固定：主机 windows 地址 192.168.1.10，开发板 uboot 或 linux 的地址为 192.168.1.20，虚拟机 ubuntu 地址为 192.168.1.141）
(3)此时开发板端 ping windows 通的。
(4)windows 中 ping 开发板也是通的。说明：首先开发板和主机的网络部分硬件都是好的，网络连接也是好的，主机 windows 中的网络软件设置是好的。

7.2、开发板运行 linux 下和虚拟机 ubuntu 的 ping 通

(1)在 linux 基础课中讲过：虚拟机的网卡设置可以选择好几种方式，常用的就是 NAT 和桥接（bridged）。
(2)虚拟机要和开发板进行网络通信，只能通过桥接方式连接。
(3)虚拟机要想被开发板 ping 通，设置步骤如下：第一步：虚拟机设置成桥接方式。第二步：虚拟机的菜单中有个“虚拟网络编辑器”，这里面要设置为桥接到有线网卡。（默认是自动的，自动的一般会影响 ping 通。因为电脑现在一般都有 2 个网卡：一个有线的一个无线的。如果选了自动，那么虚拟机会自动桥接到无线网卡上，但是我们却是通过有线网卡来连接开发板的，自然 ping 不通）第三步：在虚拟机 ubuntu 中设置 IP 地址为 192.168.1.141（可以通过/etc/network/interfaces 文件来设置静态的然后重启；也可以直接命令行 ifconfig 去设置）
(4)此时开发板 ping 虚拟机 ubuntu 应该就通了。
(5)此时虚拟机 ubuntu 中 ping 开发板也是通的。

7.3、开发板运行 uboot 下和主机 Windows 的 ping 通

(1)刚才开发板运行 linux 时和主机 windows、虚拟机 ubuntu 都 ping 通了，说明硬件和连接和主机设置没错。
(2)此时开发板重启进入 uboot，设置好 ipaddr、gatewayip，然后去 ping windows 发现还是不通。 怀

疑 uboot 本身网络驱动有问题。
(3)然后同样情况下尝试去 ping 通虚拟机 ubuntu，理论分析应该也不通，但是实际发现是通的。

7.4、开发板运行 uboot 下和虚拟机 ubuntu 的 ping 通 uboot 和虚拟机 ubuntu 互相 ping 通（前提是虚拟机 ubuntu 设置为桥接，且桥接到有线网卡，且 ip 地址设置正确的情况下）结论：开发板中运行的 uboot 有点小 bug，ping windows 就不通，ping 虚拟机 ubuntu 就通。

uboot 常用命令 3

8.1、tftp 下载指令：tftp

(1)uboot 本身主要目标是启动内核，为了完成启动内核必须要能够部署内核，uboot 为了部署内核就需要将内核镜像从主机中下载过来然后烧录到本地 flash 中。uboot 如何从主机（windows 或者虚拟机 ubuntu）下载镜像到开发板上？有很多种方式，主流方式是：fastboot 和 tftp。fastboot 的方式是通过 USB 线进行数据传输。tftp 的方式是通过有线网络的。典型的方式就是通过网络，fastboot 是近些年才新发展的。

(2)tftp 方式下载时实际上 uboot 扮演的是 tftp 客户端程序角色，主机 windows 或虚拟机 ubuntu 中必须有一个 tftp 服务器，然后将要下载的镜像文件放在服务器的下载目录中，然后开发板中使用 uboot 的 tftp 命令去下载即可。

(3)有些人习惯在 windows 中搭建 tftp 服务器，一般是用一些软件来搭建（譬如 tftpd32，使用起来比较简单）；有些人习惯在 linux 下搭建 tftp 服务器，可以参考网盘中的虚拟机下载目录下的一个教程《嵌入式开发环境搭建-基于 14.04.pdf》，这里面有 ubuntu 中搭建 tftp 服务器的教程，也可以自己上网搜索教程尝试。（如果你直接就用我的虚拟机，那就已经搭建好了，不用再搭建了；如果是自己新装的那就参考文档搭建；如果你的版本和我的不一样那搭建过程可能不一样）

(4)我的虚拟机搭建的时候设置的 tftp 下载目录是/tftpboot，将要被下载的镜像复制到这个目录下。
(5)检查开发板 uboot 的环境变量，注意 serverip 必须设置为虚拟机 ubuntu 的 ip 地址。（serverip 这个环境变量的意义就是主机 tftp 服务器的 ip 地址）

(6)然后在开发板的 uboot 下先 ping 通虚拟机 ubuntu，然后再尝试下载：tftp 0x30000000 zImage-qt（意思是将服务器上名为 zImage-qt 的文件下载到开发板内存的 0x30000000 地址处。）

(7)镜像下载到开发板的 DDR 中后，uboot 就可以用 movi 指令进行镜像的烧写了。注意：如果你是用的 windows 下的 tftp 服务器，那 uboot 的 serverip 就要设置为和 windwos 下 tftp 服务器的 ip 地址一样（windows 下的 tftp 服务器软件设置的时候就有个步骤是让你设置服务器的 ip 地址，这个 ip 地址和主机 windows 必须在一个网段）。

uboot 的常用命令 4

9.1、SD 卡/iNand 操作指令 movi

(1)开发板如果用 SD 卡/EMMC/iNand 等作为 Flash，则在 uboot 中操作 flash 的指令为 movi（或 mmc）

(2)movi 指令是一个命令集，有很多子命令，具体用法可以 help movi 查看。

(3)movi 的指令都是 movi read 和 movi write 一组的，movi read 用来读取 iNand 到 DDR 上，movi write 用来将 DDR 中的内容写入 iNand 中。理解这些指令时一定要注意涉及到的 2 个硬件：iNand 和 DDR 内存。

(4)movi read ｛u-boot | kernel｝{addr} 这个命令使用了一种通用型的描述方法来描述：movi 和 read 外面没有任何标记说明每一次使用这个指令都是必选的；一对大括号{}括起来的部分必选 1 个；大括号中的竖线表是多选一；中括号[]表示可选参数

(5)指令有多种用法，譬如 movi read u-boot 0x30000000，意思就是把 iNand 中的 u-boot 分区读出到 DDR 的 0x30000000 起始的位置处。（uboot 代码中将 iNand 分成了很多个分区，每个分区有地址范围和分区名，uboot 程序操作中可以使用直接地址来操作 iNand 分区，也可以使用分区名来操作分区。）；注意这里的 0x30000000 也可以直接写作 30000000，意思是一样的（uboot 的命令行中所有数字都被默认当作十六进制处理，不管你加不加 0x 都一样）。

9.2、NandFlash 操作指令 nand 理解方法和操作方法完全类似于 movi 指令

9.3、内存操作指令：mm、mw、md

(1)DDR 中是没有分区的（只听说过对硬盘、Flash 进行分区，没听说过对内存进行分区…），但是内存使用时要注意，千万不能越界踩到别人了。因为 uboot 是一个裸机程序，不像操作系统会由系统整体管理所有内存，系统负责分配和管理，系统会保证内存不会随便越界。然后裸机程序中 uboot 并不管理所有内存，内存是散的随便用的，所以如果程序员（使用 uboot 的人）自己不注意就可能出现自己把自己的数据给覆盖了。（所以你思考下我们为什么把 uboot 放在 23E00000 地址处）

(2)md 就是 memory display，用来显示内存中的内容。

(3)mw 就是 memory write，将内容写到内存中

(4)mm 就是 memory modify，修改内存中的某一块，说白了还是写内存（如果需要批量的逐个单元的修改内存，用 mm 最合适）

9.4、启动内核指令：bootm、go

https://www.vxworks.net

(1)uboot 的终极目标就是启动内核，启动内核在 uboot 中表现为一个指令，uboot 命令行中调用这个指令就会启动内核（不管成功与否，所以这个指令是一条死路）。

(2)差别：bootm 启动内核同时给内核传参，而 Go 命令启动内核不传参。bootm 其实才是正宗的启动内核的命令，一般情况下都用这个；go 命令本来不是专为启动内核设计的，go 命令内部其实就是一个函数指针指向一个内存地址然后直接调用那个函数，go 命令的实质就是 PC 直接跳转到一个内存地址去运行而已。go 命令可以用来在 uboot 中执行任何的裸机程序（有一种调试裸机程序的方法就是事先启动 uboot，然后在 uboot 中去下载裸机程序，用 go 命令去执行裸机程序）

uboot 的常用环境变量 1

10.1、环境变量如何参与程序运行

(1)环境变量有 2 份，一份在 Flash 中，另一份在 DDR 中。uboot 开机时一次性从 Flash 中读取全部环境变量到 DDR 中作为环境变量的初始化值，然后使用过程中都是用 DDR 中这一份，用户可以用 saveenv 指令将 DDR 中的环境变量重新写入 Flash 中去更新 Flash 中环境变量。下次开机时又会从 Flash 中再读一次。

(2)环境变量在 uboot 中是用字符串表示的，也就是说 uboot 是按照字符匹配的方式来区分各个环境变量的。因此用的时候一定要注意不要打错字了。

1、自动运行倒数时间：bootdelay

2、网络设置：ipaddr serverip

(1)ipaddr 是开发板的本地 IP 地址

(2)serverip 是开发板通过 tftp 指令去 tftp 服务器下载东西时，tftp 服务器的 IP 地址。

(3)gatewayip 是开发板的本地网关地址

(4)netmask 是子网掩码

(5)ethaddr 是开发板的本地网卡的 MAC 地址。

uboot 的常用环境变量 2

11.1、自动运行命令设置：bootcmd

(1)uboot 启动后会开机自动倒数 bootdelay 秒，如果没有人按下回车打断启动，则 uboot 会自动执行启动命令来启动内核。

(2)uboot 开机自动启动时实际就是在内部执行了 bootcmd 这个环境变量的值所对应的命令集：bootcmd=movi read kernel 30008000; bootm 30008000　意思是：将 iNand 的 kernel 分区读取到 DDR 内存的 0x30008000 地址处，然后使用 bootm 启动命令从内存 0x30008000 处去启动内核。

(3)set bootcmd printenv，然后 saveenv；然后重启则会看到启动倒数后自动执行 printenv 命令打印出环境变量。这个小实验说明开机自动执行了 bootcmd。

(4)设置 bootcmd 环境变量：set bootcmd 'movi read kernel 30008000; bootm 30008000'

11.2、uboot 给 kernel 传参：bootargs

(1)linux 内核启动时可以接收 uboot 给他传递的启动参数，这些启动参数是 uboot 和内核约定好的形式、内容，linux 内核在这些启动参数的指导下完成启动过程。这样的设计是为了灵活，为了内核在不重新编译的情况下可以用不同的方式启动。

(2)我们要做的事情就是：在 uboot 的环境变量中设置 bootargs，然后 bootm 命令启动内核时会自动将 bootargs 传给内核。

(3)环境变量 bootargs=console=ttySAC2,115200 root=/dev/mmcblk0p2 rw init=/linuxrc rootfstype=ext3 意义解释：console=ttySAC2,115200 控制台使用串口 2，波特率 115200.root=/dev/mmcblk0p2 rw 根文件系统在 SD 卡端口 0 设备（iNand）第 2 分区，根文件系统是可读可写的 init=/linuxrc linux 的进程 1（init 进程）的路径 rootfstype=ext3 根文件系统的类型是 ext3(4)内核传参非常重要。在内核移植的时候，新手经常因为忘记给内核传参，或者给内核传递的参数不对，造成内核启动不起来。

11.3、新建、更改、删除一个环境变量的方法

(1)新建一个环境变量，使用 set var value

(2)更改一个环境变量，使用 set var value

(3)删除一个环境变量，使用 set var 注意：修改完成环境变量后一定要保存，否则下次开机更改就又没了。

uboot 中对 Flash 和 DDR 的管理

12.1、uboot 阶段 Flash 的分区

(1)所谓分区，就是说对 Flash 进行分块管理。

(2)PC 机等产品中，因为大家都是在操作系统下使用硬盘的，整个硬盘由操作系统统一管理，操作系统会使用文件系统帮我们管理硬盘空间。（管理保证了文件之间不会互相堆叠），于是乎使用者不用自己太过在意分区问题。

(3)在 uboot 中是没有操作系统的，因此我们对 Flash（相当于硬盘）的管理必须事先使用分区界定（实际上在 uboot 中和 kernel 中都有个分区表，分区表就是我们在做系统移植时对 Flash 的整体管理分配方法）。有了这个界定后，我们在部署系统时按照分区界定方法来部署，uboot 和 kernel 的软件中也是按照这个分区界定来工作，就不会错。

(4)分区方法不是一定的，不是固定的，是可以变动的。但是在一个移植中必须事先设计好定死，一般在设计系统移植时就会定好，定的标准是：uboot:uboot 必须从 Flash 起始地址开始存放（也许是扇区 0，也许是扇区 1，也许是其他，取决于 SoC 的启动设计），uboot 分区的大小必须保证 uboot 肯定能放下，一般设计为 512KB 或者 1MB（因为一般 uboot 肯定不足 512KB，给再大其实也可以工作，但是浪费）；环境变量：环境变量分区一般紧贴着 uboot 来存放，大小为 32KB 或者更多一点。kernel：kernel 可以紧贴环境变量存放，大小一般为 3MB 或 5MB 或其他。rootfs：……剩下的就是自由分区，一般 kernel 启动后将自由分区挂载到 rootfs 下使用

总结：一般规律如下：

(1)各分区彼此相连，前面一个分区的结尾就是后一个分区的开头。

(2)整个 flash 充分利用，从开头到结尾。

(3)uboot 必须在 Flash 开头，其他分区相对位置是可变的。

(4)各分区的大小由系统移植工程师自己来定，一般定为合适大小（不能太小，太小了容易溢出；不能太大，太大了浪费空间）

(5)分区在系统移植前确定好，在 uboot 中和 kernel 中使用同一个分区表。将来在系统部署时和系统代码中的分区方法也必须一样。

12.2、uboot 阶段 DDR 的分区

(1)DDR 的分区和 Flash 的分区不同，主要是因为 Flash 是掉电存在的，而 DDR 是掉电消失，因此可以说 DDR 是每次系统运行时才开始部署使用的。

(2)内存的分区主要是在 linux 内核启动起来之前，linux 内核启动后内核的内存管理模块会接管整个内存空间，那时候就不用我们来管了。

(3)注意内存分区关键就在于内存中哪一块用来干什么必须分配好，以避免各个不同功能使用了同一块内存造成的互相踩踏。譬如说我们 tftp 0x23E00000 zImage 去下载 zImage 到内存的 0x23E00000 处就会出错，因为这个内存处实际是 uboot 的镜像所在。这样下载会导致下载的 zImage 把内存中的 uboot 给冲掉。