第十章 U-boot 使用

在移植 U-Boot 之前，我们肯定要先使用一下 U-Boot，得先体验一下 U-Boot 是个什么东西。STM32MP157 开发板光盘资料里面已经提供了一个正点原子团队已经移植好的 U-Boot，本章我们就直接编译这个移植好的 U-Boot，然后烧写到 EMMC 里面启动，启动 U-Boot 以后就可以学习使用 U-Boot 的命令。

10.1 U-Boot 简介

Linux 系统要启动需要通过 bootloader 程序引导，也就说芯片上电以后先运行一段 bootloader 程序。这段 bootloader 程序会先初始化 DDR 等外设，然后将 Linux 内核从 flash(NAND，NOR FLASH，SD，EMMC 等)拷贝到 DDR 中，最后启动 Linux 内核。当然了，bootloader 的实际工作要复杂的多，但是它最主要的工作就是启动 Linux 内核，bootloader 和 Linux 内核的关系就跟 PC 上的 BIOS 和 Windows 的关系一样，bootloader 就相当于 BIOS。所以我们要先搞定 bootloader，很庆幸，有很多现成的 bootloader 软件可以使用，比如 U-Boot、vivi、RedBoot 等等，其中以 U-Boot 使用最为广泛，为了方便书写，本书会将 U-Boot 写为 uboot。

uboot 的全称是 Universal Boot Loader，uboot 是一个遵循 GPL 协议的开源软件，uboot 是一个裸机代码，可以看作是一个裸机综合例程。现在的 uboot 已经支持液晶屏、网络、USB 等高级功能。uboot 官网为 http://www.denx.de/wiki/U-Boot/，如图 10.1.1 所示：

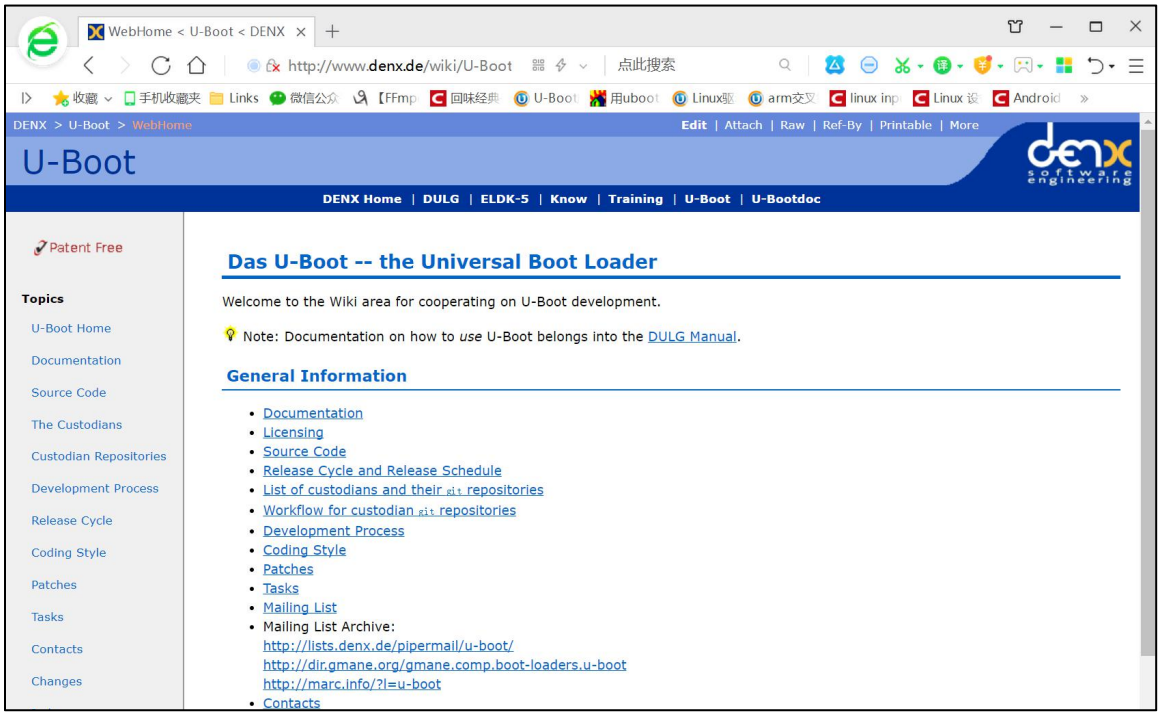

图 10.1.1 uboot 官网

我们可以在 uboot 官网下载 uboot 源码，点击图 10.1.1 中左侧 Topics 中的“Source Code”，打开以后如图 10.1.2 所示：

图 10.1.2 uboot 源码界面

点击图 10.1.2 中的“FTP”，进入其 FTP 服务器即可看到 uboot 源码，如图 10.1.3 所示：

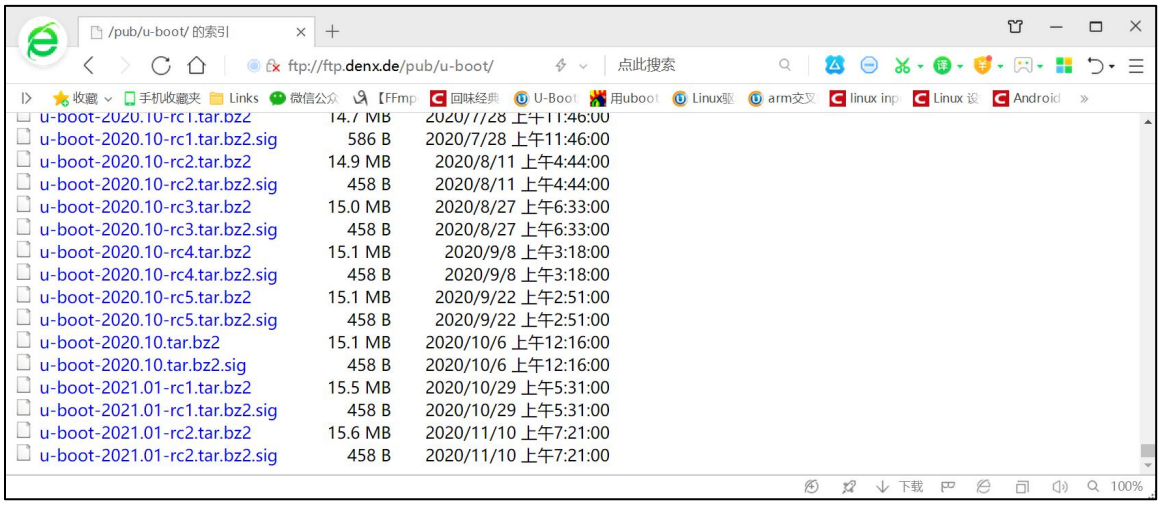

图 10.1.3 uboot 源码

图 10.1.3 中就是 uboot 原汁原味的源码文件，目前最新的版本是 2020.10。但是我们一般不会直接用 uboot 官方的 U-Boot 源码的。uboot 官方的 uboot 源码是给半导体厂商准备的，半导体厂商会下载 uboot 官方的 uboot 源码，然后将自家相应的芯片移植进去。也就是说半导体厂商会自己维护一个版本的 uboot，这个版本的 uboot 相当于是他们定制的。既然是定制的，那么肯定对自家的芯片支持会很全，虽然 uboot 官网的源码中一般也会支持他们的芯片，但是绝对是没有半导体厂商自己维护的 uboot 全面。ST 提供了 2020.01 版本的 uboot，在 6.1.1 小节获取 ST 官方系统源码中我们已经得到了 ST 官方 uboot 源码， 进入到如下目录：

/stm32mp1-openstlinux-5.4-dunfell-mp1-20-06-24/sources/arm-ostl-linux-gnueabi

uboot 源码如图 10.1.4 所示：

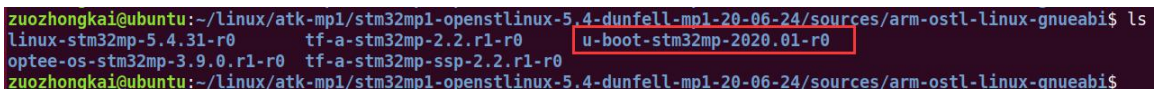

图 10.1.4 ST 官方 uboot 源码

图 10.1.4 中的“u-boot-stm32mp-2020.01-r0”就是 ST 官方 uboot 源码包，它支持了 STM32MP1 家族全系列芯片(后续 ST 也会一直更新，添加新的 SOC 进去)，而且支持各种启动

方式，比如 EMMC、NAND 等等，这些都是 uboot 官方所不支持的。但是图 10.1.4 中的 uboot 是针对 ST 自家评估板的，如果要在我们自己的板子上跑，那么就需要对其进行修改，使其支持我们自己做的板子，正点原子的 STM32MP157 开发板就是自己做的板子，虽然大部分都参考了 ST 官方的 STM32MP157 EVK 开发板，但是还是有很多不同的地方，所以需要修改 ST 官方的 uboot，使其适配正点原子的 STM32MP157 开发板。所以当我们拿到开发板以后，是有三种 uboot 的，这三种 uboot 的区别如表 10.1.1 所示：

种类	描述
uboot 官方的 uboot 代码	由 uboot 官方维护开发的 uboot 版本，版本更新快，基本包含所有常用的芯片。
半导体厂商的 uboot 代码	半导体厂商维护的一个 uboot，专门针对自家的芯片，在对自家芯片支持上要比 uboot 官方的好。
开发板厂商的 uboot 代码	开发板厂商在半导体厂商提供的 uboot 基础上加入了对自家开发板的支持。

表 10.1.1 三种 uboot 的区别

那么这三种 uboot 该如何选择呢？首先 uboot 官方的基本是不会用的，因为支持太弱了。最常用的就是半导体厂商或者开发板厂商的 uboot，如果你用的半导体厂商的评估板，那么就使用半导体厂商的 uboot，如果你是购买的第三方开发板，比如正点原子的 STM32MP157 开发板，那么就使用正点原子提供的 uboot 源码（也是在半导体厂商的 uboot 上修改的）。当然了，你也可以在购买了第三方开发板以后使用半导体厂商提供的 uboot，只不过有些外设驱动可能不支持，需要自己移植，这个就是我们常说的 uboot 移植。本节是 uboot 的使用，所以就直接使用正点原子已经移植好的 uboot，这个已经放到了开发板光盘中了，路径为：开发板光盘→1、程序源码→1、正点原子 Linux 出厂系统源码→u-boot-stm32mp-2020.01-gdb8d2374-v1.0.tar.bz2。

10.2 U-Boot 初次编译

10.2.1 编译

首先需要在 Ubuntu 中安装一些库，否则编译 uboot 会报错，安装命令如下：

```
sudo apt-get install libncurses5-dev bison flex
```

在 Ubuntu 中创建存放 uboot 的目录，比如我这里新建了一个名为“alientek_uboot”的文件夹用于存放正点原子提供的 uboot 源码。alientek_uboot 文件夹创建成功以后使用 FileZilla 软件将正点原子提供的 uboot 源码拷贝到此目录中，正点原子提供的 uboot 源码已经放到了开发板光盘中，路径为：开发板光盘→1、程序源码→1、正点原子 Linux 出厂系统源码→u-boot-stm32mp-2020.01-xxxxxxxx-v1.0.tar.bz2（“xxxxxxxx”为 uboot 打包时候的版本号，每次打包其版本号都不同！所以大家不要纠结于开发板光盘中的 uboot 源码打包版本号是否和教程里面的一致）。将 stm32mp-2020.01-xxxxxxxx-v1.0.tar.bz2 拷贝到前面新建的 alientek_uboot 文件夹下，完成以后如图 10.2.1.1 所示：

```
zuozhongkai@ubuntu:~/linux/atk-mp1/uboot/alientek_uboot$ ls
u-boot-stm32mp-2020.01-gdb8d2374-v1.0.tar.bz2
zuozhongkai@ubuntu:~/linux/atk-mp1/uboot/alientek_uboot$
```

图 10.2.1.1 正点原子出厂 uboot 源码

使用如下命令对其进行解压缩:

```
tar -vxf u-boot-stm32mp-2020.01-gdb8d2374-v1.0.tar.bz2
```

解压完成以后如图 10.2.1.2 所示：

```
zuozhongkai@ubuntu:~/linux/atk-mp1/uboot/alientek_uboot$ ls
api          configs          env       lib          README
arch         CONTRIBUTING.md  examples  Licenses     scripts
board        disk             fs        MAINTAINERS  test
cmd          doc              include   Makefile     tools
common       drivers          Kbuild    net          u-boot-stm32mp-2020.01-gdb8d2374-v1.0.tar.bz2
config.mk    dts              Kconfig   post
```

图 10.2.1.2 解压后的 Uboot

图 10.2.1.2 中除了 u-boot-stm32mp-2020.01-gdb8d2374-v1.0.tar.bz2 这个正点原子提供 uboot 源码压缩包以外，其他的文件和文件夹都是解压出来的 uboot 源码。执行以下命令，编译正点原子提供的 uboot。

```
make distclean
make ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabihf- stm32mp157d_atk_def config
make V=1 ARCH=arm CROSS_COMPILE=arm-none-linux-gnueabihf- DEVICE_TREE= stm32mp157d-atk all
```

上面命令每次编译的时候都要指定 ARCH、CROSS_COMPILE 和 DEVICE_TREE，这三个含义如下：

ARCH：指定所使用的平台架构，这里肯定是 arm。

CROSS_COMPILE：所使用的交叉编译器前缀，本教程使用的是交叉编译器前缀为 arm-none-linux-gnueabihf-。

DEVICE_TREE：设备树文件，uboot 也支持设备树，所以在编译的时候需要指定设备树文件，不同的硬件其设备树文件肯定不同，这里为 stm32mp157d_atk，也就是正点原子的 STM32MP157 开发板对应的设备树。

编译的时候每次都输入 ARCH 和 CROSS_COMPILE 比较麻烦，为了方便起见，我们可以直接修改 uboot 的 Makefile 文件，在里面直接对 ARCH 和 CROSS_COMPILE 进行赋值，也就是直接将 ARCH 设置为 arm，CROSS_COMPILE 设置为 arm-none-linux-gnueabihf-，修改完成以后如图 10.2.1.3 所示：

```
261 # set default to nothing for native builds
262 ifeq ($(HOSTARCH),$(ARCH))
263 CROSS_COMPILE ?=
264 endif
265
266 ARCH = arm
267 CROSS_COMPILE = arm-none-linux-gnueabihf-
268
```

设置ARCH和CROSS_COMPILE的值

图 10.2.1.3 设置 ARCH 和 CROSS_COMPILE 值

注意！不能在 Makefile 里面对 DEVICE_TREE 进行复制，因为没用，必须在编译的时候手动输入！

设置好 Makefile 里面的 ARCH 和 CROSS_COMPILE 以后就可以将编译命令简化为如下所示：

```
make distclean                                  //清除
make stm32mp157d_atk_defconfig                  //配置 uboot
make V=1 DEVICE_TREE=stm32mp157d-atk all        //编译
```

上述命令和前面的相比就要简洁很多，最后的“make V=1”是真正的编译命令，V=1 表示编译 uboot 的时候输出详细的编译过程，方便我们观察 uboot 编译过程。直接输入“make”命令的话默认使用单线程编译，编译速度会比较慢，可以通过添加“-j”选项来使用多线程编译，比如使用 8 线程编译，最后的编译命令就是：

```
make V=1 DEVICE_TREE=stm32mp157d-atk all -j8          //8 线程编译
```

uboot 编译完成如图 10.2.1.4 所示：

```
  cat u-boot-nodtb.bin dts/dt.dtb > u-boot-dtb.bin
  cp u-boot-dtb.bin u-boot.bin
  ./tools/mkimage -T stm32image -a 0xC0100000 -e 0xC0100000 -d u-boot.bin u-boot.stm32 >u-boot.s
tm32.log  && cat u-boot.stm32.log
Image Type   : STMicroelectronics STM32 V1.0
Image Size   : 876085 bytes
Image Load   : 0xc0100000
Entry Point  : 0xc0100000
Checksum     : 0x04ae3f76
Option      : 0x00000001
BinaryType : 0x00000000
  ./scripts/check-config.sh u-boot.cfg ./scripts/config_whitelist.txt .
zuozhongkai@ubuntu:~/linux/atk-mp1/uboot/alientek uboot$
```

图 10.2.1.4 uboot 编译成功

编译完成以后 就会在 uboot 源码目录下生成相应的镜像文件，如图 10.2.1.5 所示：

```
zuozhongkai@ubuntu:~/linux/atk-mp1/uboot/alientek_uboot$ ls
api          CONTRIBUTING.md  fs          Makefile     tools              u-boot.lds
arch         disk             include     net          u-boot             u-boot.map
board        doc              Kbuild      post         u-boot.bin         u-boot-nodtb.bin
cmd          drivers          Kconfig     README       u-boot.cfg         u-boot.srec
common       dts              lib         scripts      u-boot.cfg.configs u-boot.stm32
config.mk    env              Licenses    System.map   u-boot.dtb         u-boot.stm32.log
configs      examples         MAINTAINERS test         u-boot-dtb.bin     u-boot.sym
zuozhongkai@ubuntu:~/linux/atk-mp1/uboot/alientek uboot$
```

图 10.2.1.5 编译后生成的 uboot 可执行文件

可以看出，编译完成以后 uboot 源码多了一些文件，重点是 u-boot.bin 和 u-boot.stm32 这两个文件。u-boot.bin 是 uboot 的二进制可执行文件，u-boot.stm32 是在 u-boot.bin 前面添加了 256 个字节头部信息。STM32MP1 内部 ROM 代码和 TF-A 在运行 uboot 的时候要求前面添加头部信息，所以这就是为什么 uboot 也有这个头部信息的原因。

10.2.2 烧写

使用 STM32CubeProgrammer 将上面编译出来的 u-boot.stm32 镜像烧写到开发板的 EMMC 里面，修改前面创建的 tf-a.tsv 文件，添加 uboot 烧写指令(其实在 9.3.2 小节已经讲过了)，在最后面添加下面这行：

示例代码 10.2.2.1 uboot 烧写指令

```
P       0x06     ssbl         Binary  mmc1     0x00080000  u-boot.stm32
```

修改以后的 tf-a.tsv 如图 10.2.2.1 所示：

```
tf-a.tsv
1  #Opt    Id      Name        Type    Device    Offset      Binary
2  -       0x01    fsbl1-boot  Binary  none      0x0         tf-a-stm32mp157d-atk-serialboot.stm32
3  -       0x03    ssbl-boot   Binary  none      0x0         u-boot.stm32
4  P       0x04    fsbl1       Binary  mmc1      boot1       tf-a-stm32mp157d-atk-trusted.stm32
5  P       0x05    fsbl2       Binary  mmc1      boot2       tf-a-stm32mp157d-atk-trusted.stm32
6  P       0x06    ssbl        Binary  mmc1      0x00080000  u-boot.stm32   <- uboot镜像烧写命令
```

图 10.2.2.1 修改后的 tf-a.tsv

最后将上一小节编译出来的 u-boot.stm32，拷贝到前面创建的 images 目录下（在做 TF-A 实验的就有 u-boot.stm32 这个文件，我们只要替换就行）。

一切准备就绪以后就可以使用 STM32CubeProgrammer 软件通过 USB OTG 将 uboot 烧写到开发板上的 EMMC 里面，等到烧写完成。完成以后设置开发板上的拨码开关，设置从 EMMC 启动，然后用 USB Type-C 线将开发板上的 USB_TTL 接口与电脑连接起来，因为我们要在串口终端里面输入命令来操作 uboot。

打开 MobaXterm，设置好串口参数，最后复位开发板。在 MobaXterm 上出现“Hit any key tostop autoboot: ”倒计时的时候按下键盘上的回车键，默认是 1 秒倒计时，在 1 秒倒

计时结束以后如果没有按下回车键的话 uboot 就会使用默认参数来启动 Linux 内核了（如果内核存在的话，如果 Linux 内核不存在那么就会进入到 uboot 的命令行模式）。如果在 1 秒倒计时结束之前按下回车键，那么就会进入 uboot 的命令行模式，如图 10.2.2.2 所示：

```
U-Boot 2020.01-stm32mp-r1 (Nov 24 2020 - 17:17:20 +0800)

CPU: STM32MP157DAA Rev.Z
Model: STMicroelectronics STM32MP157D eval daughter
Board: stm32mp1 in trusted mode (st,stm32mp157d-atk)
DRAM:  1 GiB
Clocks:
- MPU : 800 MHz
- MCU : 208.878 MHz
- AXI : 266.500 MHz
- PER : 24 MHz
- DDR : 533 MHz
WDT:   Started with servicing (32s timeout)
NAND:  0 MiB
MMC:   STM32 SD/MMC: 0, STM32 SD/MMC: 1
Loading Environment from MMC... OK
In:    serial
Out:   serial
Err:   serial
invalid MAC address in OTP 00:00:00:00:00:00
Net:
Error: ethernet@5800a000 address not set.
No ethernet found.

Hit any key to stop autoboot:  0
STM32MP>
```

图 10.2.2.2 uboot 启动 log 信息

从图 10.2.2.2 可以看出，当进入到 uboot 的命令行模式以后，左侧会出现一个“STM32MP=>”标志。uboot 启动的时候会输出一些信息，这些信息如下所示：

示例代码 10.2.2.2 uboot 启动 log 信息

```
1  U-Boot 2020.01-stm32mp-r1 (Nov 24 2020 - 17:17:20 +0800)
2
3  CPU: STM32MP157DAA Rev.Z
4  Model: STMicroelectronics STM32MP157D eval daughter
5  Board: stm32mp1 in trusted mode (st,stm32mp157d-atk)
6  DRAM:  1 GiB
7  Clocks:
8  - MPU : 800 MHz
9  - MCU : 208.878 MHz
10 - AXI : 266.500 MHz
11 - PER : 24 MHz
12 - DDR : 533 MHz
13 WDT:   Started with servicing (32s timeout)
14 NAND:  0 MiB
15 MMC:   STM32 SD/MMC: 0, STM32 SD/MMC: 1
16 Loading Environment from MMC... OK
17 In:    serial
18 Out:   serial
19 Err:   serial
20 invalid MAC address in OTP 00:00:00:00:00:00
```

```
21 Net:
22 Error: ethernet@5800a000 address not set.
23 No ethernet found.
24
25 Hit any key to stop autoboot:  0
26 STM32MP>
```

简单讲解一下 uboot 启动过程的 log 信息：

第 1 行是 uboot 版本号和编译时间，可以看出，当前的 uboot 版本号是 2020.01，编译时间是 2020 年 11 月 24 日 17: 17。

第 3 行是 CPU 的信息，可以看出 CPU 型号为 STM32MP157DAA。

第 4 行是板子信息，当前板子是 ST 公司的 STM32MP157D eval 开发板，这个信息是可以改的，因为正点原子是直接参考 ST 公司的 EVK 开发板移植的 uboot，所以这部分信息也就没改。

第 5 行是板子的一些信息，比如工作在 trusted 模式下。

第 6 行是 DDR 的大小为 1GB。

第 7~12 行它们的频率分别为，MPU 频率、MCU 频率、AXI 总线频率、PER 的频率、DDR 频率。

第 13 行是看门狗信息，喂狗时间为 32s。

第 14 行是 NAND 的大小，因为正点原子的 STM32MP157 开发板没有 NAND，所以这里就是 0MB。

第 15 行是板子上 MMC 设备，一共有两个，SD/MMC0 (SD 卡)和 SD/MMC1 (EMMC)。

第 16 行是从 MMC 里获取环境变量。

第 17~19 行是标准输入、标准输出和标准错误所使用的终端，这里都使用串口(serial)作为终端。

第 20~23 行是网络相关信息，网络的 MAC 地址从 OTP 里获取，因为我们的 OTP 没有设置 MAC 地址，所以就获取失败。这里的网络是可以用的，只是因为没有 MAC 地址所以提示没有找到网络，可以自行添加相关环境变量来设置 MAC 地址，后面会讲如何设置。

第 25 行是倒计时提示，默认倒计时 1 秒，倒计时结束之前按下回车键就会进入 Linux 命令行模式。如果在倒计时结束以后没有按下回车键，那么 Linux 内核就会启动，Linux 内核一旦启动，uboot 就会寿终正寝。

uboot 的主要作用是引导 kernel，我们现在已经进入 uboot 的命令行模式了，进入命令行模式以后就可以给 uboot 发号施令了。当然了，不能随便发号施令，得看看 uboot 支持哪些命令，然后使用这些 uboot 所支持的命令来做一些工作，下一节就讲解 uboot 命令的使用。

10.3 U-Boot 命令使用

进入 uboot 的命令行模式以后输入“help”或者“？”，然后按下回车即可查看当前 uboot 所支持的命令，如图 10.3.1 所示：

```
STM32MP> help
?          - alias for 'help'
adc        - ADC sub-system
base       - print or set address offset
bdinfo     - print Board Info structure
blkcache   - block cache diagnostics and control
bmp        - manipulate BMP image data
bootcount  - bootcount
bootefi    - Boots an EFI payload from memory
bootm      - boot application image from memory
bootp      - boot image via network using BOOTP/TFTP protocol
bootstage  - Boot stage command
bootz      - boot Linux zImage image from memory
chpart     - change active partition
clk        - CLK sub-system
cls        - clear screen
cmp        - memory compare
coninfo    - print console devices and information
cp         - memory copy
crc32      - checksum calculation
date       - get/set/reset date & time
dcache     - enable or disable data cache
dfu        - Device Firmware Upgrade
dhcp       - boot image via network using DHCP/TFTP protocol
dm         - Driver model low level access
dtimg      - manipulate dtb/dtbo Android image
```

图 10.3.1 uboot 的命令列表(部分)

图 10.3.1 中只是 uboot 的一部分命令，具体的命令列表以实际为准。图 10.3.1 中的命令并不是 uboot 所支持的所有命令，说过 uboot 是可配置的，需要什么命令就使能什么命令。所以图 10.3.1 中的命令是正点原子提供的 uboot 中使能的命令，uboot 支持的命令还有很多，而且也可以在 uboot 中自定义命令。这些命令后面都跟有命令说明，用于描述此命令的作用，但是命令具体怎么用呢？我们输入“help(或?) 命令名”既可以查看命令的详细用法，以“bootz”这个命令为例，我们输入如下命令即可查看“bootz”这个命令的用法：

```
? bootz 或 help bootz
```

结果如图 10.3.2 所示：

```
STM32MP> ? bootz
bootz - boot Linux zImage image from memory

Usage:
bootz [addr [initrd[:size]] [fdt]]
    - boot Linux zImage stored in memory
        The argument 'initrd' is optional and specifies the address
        of the initrd in memory. The optional argument ':size' allows
        specifying the size of RAW initrd.
        When booting a Linux kernel which requires a flat device-tree
        a third argument is required which is the address of the
        device-tree blob. To boot that kernel without an initrd image,
        use a '-' for the second argument. If you do not pass a third
        a bd_info struct will be passed instead

STM32MP>
```

图 10.3.2 bootz 命令使用说明

图 10.3.2 列出了“bootz”这个命令的详细说明，其它的命令也可以使用此方法查询具体的使用方法。接下来我们学习一下一些常用的 uboot 命令。

10.3.1 查询命令

常用的和信息查询有关的命令有 3 个：bdinfo、printenv 和 version。先来看一下 bdinfo 命令，此命令用于查看板子信息，直接输入“bdinfo”即可，结果如图 10.3.1.1 示：

```
STM32MP> bdinfo
arch_number = 0x00000000
boot_params = 0xc0000100
DRAM bank   = 0x00000000
-> start    = 0xc0000000
-> size     = 0x40000000
baudrate    = 115200 bps
TLB addr    = 0xf5ff0000
relocaddr   = 0xf5c46000
reloc off   = 0x35b46000
irq_sp      = 0xf3aedd50
sp start    = 0xf3aedd40
FB base     = 0x00000000
Early malloc usage: 13f4 / 3000
fdt_blob    = 0xf3aedfe0
STM32MP>
```

图 10.3.1.1 bdinfo 命令

从图 10.3.1.1 中可以看出 DRAM 的起始地址和大小、BOOT 参数保存起始地址、波特率、sp(堆栈指针)起始地址等信息。命令“printenv”用于输出环境变量信息，uboot 也支持 TAB 键自动补全功能，输入“print”然后按下 TAB 键就会自动补全命令。直接输入“print”也可以，因为整个 uboot 命令中只有 printenv 的前缀是“print”，所以当输入 print 以后就只有 printenv 命令了。输入“print”，然后按下回车键，环境变量如图 10.3.1.2 所示：

```
STM32MP> print
altbootcmd=run bootcmd
android_mmc_boot=mmc dev ${devnum};run android_mmc_splash;run android_mmc_fdt;run android_mmc_kernel;bootm ${kerne
l_addr_r} - ${fdt_addr_r};
android_mmc_fdt=if part start mmc ${devnum} dt_${suffix} dt_start &&part size mmc ${devnum} dt_${suffix} dt_size;t
hen mmc read ${dtimg_addr} ${dt_start} ${dt_size};dtimg getindex ${dtimg_addr} ${board_id} ${board_rev} dt_index;d
timg start ${dtimg_addr} ${dt_index} fdt_addr_r;fi
android_mmc_kernel=if part start mmc ${devnum} boot_${suffix} boot_start &&part size mmc ${devnum} boot_${suffix}
boot_size;then mmc read ${kernel_addr_r} ${boot_start} ${boot_size};part nb mmc ${devnum} system_${suffix} rootpar
t_nb;env set bootargsroot=/dev/mmcblk${devnum}p${rootpart_nb} androidboot.serialno=${serial#} androidboot.slot_suf
fix=_${suffix};fi
android_mmc_splash=if part start mmc ${devnum} splash splash_start && part size mmc ${devnum} splash splash_size;t
hen mmc read ${splashimage} ${splash_start} ${splash_size};cls; bmp display ${splashimage} m m;fi
arch=arm
autoload=no
baudrate=115200
board=stm32mp1
board_name=stm32mp157d-atk
boot_a_script=load ${devtype} ${devnum}:${distro_bootpart} ${scriptaddr} ${prefix}${script}; source ${scriptaddr}
boot_device=mmc
boot_efi_binary=if fdt addr ${fdt_addr_r}; then bootefi bootmgr ${fdt_addr_r};else bootefi bootmgr ${fdtcontroladd
r};fi;load ${devtype} ${devnum}:${distro_bootpart} ${kernel_addr_r} efi/boot/bootarm.efi; if fdt addr ${fdt_addr_r
}; then bootefi ${kernel_addr_r} ${fdt_addr_r};else bootefi ${kernel_addr_r} ${fdtcontroladdr};fi
boot_extlinux=sysboot ${devtype} ${devnum}:${distro_bootpart} any ${scriptaddr} ${prefix}${boot_syslinux_conf}
```

图 10.3.1.2 printenv 命令部分结果

图 10.3.1.2 只是 printenv 命令的部分内容，STM32MP1 系列的环境变量有很多，比如 baudrate、board、board_name、boot_device、bootcmd、bootdelay 等等。uboot 中的环境变量都是字符串，既然叫做环境变量，那么它的作用就和“变量”一样。比如 bootdelay 这个环境变量就表示 uboot 启动延时时间，默认 bootdelay=1，也就默认延时 1 秒。前面说的 1 秒倒计时就是由 bootdelay 定义的，如果将 bootdelay 改为 5 的话就会倒计时 5s 了。uboot 中的环境变量是可以修改的，有专门的命令来修改环境变量的值，稍后我们会讲解。

命令 version 用于查看 uboot 的版本号，输入“version”，uboot 版本号如图 10.3.1.3 所示：

```
STM32MP> version
U-Boot 2020.01-stm32mp-r1 (Nov 24 2020 - 17:17:20 +0800)

arm-none-linux-gnueabihf-gcc (GNU Toolchain for the A-profile Architecture 9.2-2019.12 (arm-9.10)) 9.2.1 20191025
GNU ld (GNU Toolchain for the A-profile Architecture 9.2-2019.12 (arm-9.10)) 2.33.1.20191209
STM32MP>
```

图 10.3.1.3 version 命令结果

10.3.2 环境变量操作命令

1、修改环境变量

环境变量的操作涉及到两个命令：setenv 和 saveenv，setenv 命令用于设置或者修改环境变量的值。命令 saveenv 用于保存修改后的环境变量，一般环境变量存放在外部 flash 中，uboot 启动的时候会将环境变量从 flash 读取到 DRAM 中。所以使用命令 setenv 修改的是 DRAM 中的环境变量值，修改以后要使用 saveenv 命令将修改后的环境变量保存到 flash 中，否则 uboot 下一次重启会继续使用以前的环境变量值。

命令 saveenv 使用起来很简单，格式为：

```
saveenv
```

比如我们要将环境变量 bootdelay 改为 5，就可以使用如下所示命令：

```
setenv bootdelay 5
saveenv
```

上述命令执行过程如图 10.3.2.1 所示：

```
STM32MP> setenv bootdelay 5
STM32MP> saveenv
Saving Environment to MMC... Writing to redundant MMC(1)... OK
STM32MP>
```

图 10.3.2.1 环境变量修改

在图 10.3.2.1 中，当我们使用命令 saveenv 保存修改后的环境变量会有保存过程提示信息，根据提示可以看出环境变量保存到了 MMC(1)中，也就是 EMMC 中。因为我用 EMMC 启动的，所以会保存到 MMC(1)中。修改 bootdelay 以后，重启开发板，uboot 就是变为 5 秒倒计时，如图 10.3.2.2 所示：

```
U-Boot 2020.01-stm32mp-r1 (Nov 24 2020 - 17:17:20 +0800)

CPU: STM32MP157DAA Rev.Z
Model: STMicroelectronics STM32MP157D eval daughter
Board: stm32mp1 in trusted mode (st,stm32mp157d-atk)
DRAM:  1 GiB
Clocks:
- MPU : 800 MHz
- MCU : 208.878 MHz
- AXI : 266.500 MHz
- PER : 24 MHz
- DDR : 533 MHz
WDT:   Started with servicing (32s timeout)
NAND:  0 MiB
MMC:   STM32 SD/MMC: 0, STM32 SD/MMC: 1
Loading Environment from MMC... OK
In:    serial
Out:   serial
Err:   serial
invalid MAC address in OTP 00:00:00:00:00:00
Net:
Error: ethernet@5800a000 address not set.
No ethernet found.

Hit any key to stop autoboot:  5
```

图 10.3.2.2 5 秒倒计时

从图 10.3.2.2 可以看出，此时 uboot 启动倒计时变为了 5 秒。

2、新建环境变量

命令 setenv 也可以用于新建命令，用法和修改环境变量一样，比如我们新建一个环境变量 author，author 的值为‘console=ttySTM0,11520 root=/dev/mmcblk2p2 rootwait rw’，那么就可以使用如下命令：

```
setenv author 'console=ttySTM0,11520 root=/dev/mmcblk2p2 rootwait rw '
saveenv
```

上面命令设置 author 的值为“console=ttySTM0,11520 root=/dev/mmcblk2p2 rootwait rw”，其中“console=ttySTM0,115200”、“root=/dev/mmcblk2p2”、“rootwait”和“rw”相当于四组“值”，
这四组“值”之间用空格隔开，所以需要使用单引号‘’将其括起来，表示这四组“值”都属
于环境变量 author。

author 命令创建完成以后重启 uboot，然后使用命令 printenv 查看当前环境变量，如图 10.3.2.3 所示：

```
android_mmc_splash=if part start mmc ${devnum} splash splash_start && part size mmc ${devnum} splash splash_size;t
hen mmc read ${splashimage} ${splash_start} ${splash_size};cls; bmp display ${splashimage} m m;fi
arch=arm
author=console=ttySTM0,115200 root=/dev/mmcblk2p2 rootwait rw
autoload=no
baudrate=115200
```

图 10.3.2.3 新建的 author 环境变量值

3、删除环境变量

既然可以新建环境变量，那么就可以删除环境变量，删除环境变量也是使用命令 setenv，要删除一个环境变量只要给这个环境变量赋空值即可，比如我们删除掉上面新建的 author 环境变量，命令如下：

```
setenv author
saveenv
```

上面命令中通过 setenv 给 author 赋空值，也就是什么都不写来删除环境变量 author。重启 uboot 就会发现环境变量 author 没有了。

10.3.3 内存操作命令

内存操作命令就是用于直接对 DRAM 进行读写操作的，常用的内存操作命令有 md、nm、
mm、mw、cp 和 cmp，我们依次来看一下这些命令都是做什么的。

1、md 命令

md 命令用于显示内存值，格式如下：

```
md[.b, .w, .l] address [# of objects]
```

命令中的[.b .w .l]对应 byte、word 和 long，也就是分别以 1 个字节、2 个字节、4 个字节来显示内存值。address 就是要查看的内存起始地址，[# of objects]表示要查看的数据长度，这个数据长度单位不是字节，而是跟你所选择的显示格式有关。比如你设置要查看的内存长度为 20(十六进制为 0x14)，如果显示格式为.b 的话那就表示 20 个字节；如果显示格式为.w 的话就表示 20 个 word，也就是 20*2=40 个字节；如果显示格式为.l 的话就表示 20 个 long，也就是 20*4=80 个字节，另外要注意：

uboot 命令中的数字都是十六进制的！不是十进制的！

比如你想查看 0XC0100000 开始的 20 个字节的内存值，显示格式为.b 的话，应该使用如下所示命令：

```
md.b C0100000 14
```

而不是：

```
md.b C0100000 20
```

上面说了，uboot 命令里面的数字都是十六进制的，所以可以不用写“0x”前缀，十进

制的 20 对应的十六进制为 0x14，所以命令 md 后面的个数应该是 14，如果写成 20 的话就表示查看 32(十六进制为 0x20)个字节的数据。分析下面三个命令的区别：

```
md.b C0100000 10
md.w C0100000 10
md.l C0100000 10
```

上面这三个命令都是查看以 0XC0100000 为起始地址的内存数据，第一个命令以.b 格式显示，长度为 0x10，也就是 16 个字节；第二个命令以.w 格式显示，长度为 0x10，也就是 16*2=32 个字节；最后一个命令以.l 格式显示，长度也是 0x10，也就是 16*4=64 个字节。这三个命令的执行结果如图 10.3.3.1 所示：

```
STM32MP> md.b c0100000 10
c0100000: b8 00 00 ea 14 f0 9f e5 14 f0 9f e5 14 f0 9f e5    ................
STM32MP> md.w c0100000 10
c0100000: 00b8 ea00 f014 e59f f014 e59f f014 e59f    ................
c0100010: f014 e59f f014 e59f f014 e59f f014 e59f    ................
STM32MP> md.l c0100000 10
c0100000: ea0000b8 e59ff014 e59ff014 e59ff014    ................
c0100010: e59ff014 e59ff014 e59ff014 e59ff014    ................
c0100020: c0100060 c01000c0 c0100120 c0100180    `....... .......
c0100030: c01001e0 c0100240 c01002a0 deadbeef    ....@...........
STM32MP>
```

图 10.3.3.1 md 命令使用示例

2、nm 命令

nm 命令用于修改指定地址的内存值，命令格式如下：

```
nm [.b, .w, .l] address
```

nm 命令同样可以以.b、.w 和.l 来指定操作格式，比如现在以.l 格式修改 0XC0100000 地址的数据为 0x12345678。输入命令：

```
nm.l C0100000
```

输入上述命令以后如图 10.3.3.2 所示：

```
STM32MP> nm.l c0100000
c0100000: ea0000b8 ?
```

图 10.3.3.2 nm 命令

在图 10.3.3.2 中，C0100000 表示现在要修改的内存地址，ea0000b8 表示地址 0xc0100000 现在的数据，‘？’后面就可以输入要修改后的数据 0x12345678，输入完成以后按下回车，然后再输入‘q’即可退出，如图 10.3.3.3 所示：

```
STM32MP> nm.l c0100000
c0100000: ea0000b8 ? 0x12345678
c0100000: 12345678 ? q
STM32MP>
```

图 10.3.3.3 修改内存数据

修改完成以后再使用 md 命令来查看一下有没有修改成功，如图 10.3.3.4 所示：

```
STM32MP> md.l c0100000 1
c0100000: 12345678                                  xV4.
STM32MP>
```

图 10.3.3.4 查看修改后的值

从图 10.3.3.4 可以看出，此时地址 0XC0100000 的值变为了 0X12345678。

3、mm 命令

mm 命令也是修改指定地址内存值的，使用 mm 修改内存值的时候地址会自增，而使用 nm 命令的话地址不会自增。比如以.l 格式修改从地址 0XC0100000 开始的连续 3 个内存块

(3*4=12 个字节)的数据为 0X05050505，操作如图 10.3.3.5 所示：

```
STM32MP> mm.l c0100000
c0100000: 12345678 ? 05050505
c0100004: e59ff014 ? 05050505
c0100008: e59ff014 ? 05050505
c010000c: e59ff014 ? q
STM32MP>
```

图 10.3.3.5 命令 mm

从图 10.3.3.5 可以看出，修改了地址 0XC0100000、0XC0100004 和 0XC0100008 的内容为 0x05050505。使用命令 md 查看修改后的值，结果如图 10.3.3.6 所示：

```
STM32MP> md.l c0100000 3
c0100000: 05050505 05050505 05050505                ............
STM32MP>
```

图 10.3.3.6 查看修改后的内存数据

从图 10.3.3.6 可以看出内存数据修改成功。

4、mw 命令

命令 mw 用于使用一个指定的数据填充一段内存，命令格式如下：

```
mw [.b, .w, .l] address value [count]
```

mw 命令同样以.b、.w 和.l 来指定操作格式，address 表示要填充的内存起始地址，value 为要填充的数据，count 是填充的长度。比如使用.l 格式将以 0XC0100000 为起始地址的 0x10 个内存块(0x10 * 4=64 字节)填充为 0X0A0A0A0A，命令如下：

```
mw.l C0100000 0A0A0A0A 10
```

然后使用命令 md 来查看，如图 10.3.3.7 所示：

```
STM32MP> mw.l c0100000 0A0A0A0A 10
STM32MP> md.l c0100000 10
c0100000: 0a0a0a0a 0a0a0a0a 0a0a0a0a 0a0a0a0a    ................
c0100010: 0a0a0a0a 0a0a0a0a 0a0a0a0a 0a0a0a0a    ................
c0100020: 0a0a0a0a 0a0a0a0a 0a0a0a0a 0a0a0a0a    ................
c0100030: 0a0a0a0a 0a0a0a0a 0a0a0a0a 0a0a0a0a    ................
STM32MP>
```

图 10.3.3.7 查看修改后的内存数据

从图 10.3.3.7 以看出内存数据修改成功。

5、cp 命令

cp 是数据拷贝命令，用于将 DRAM 中的数据从一段内存拷贝到另一段内存中，或者把 NorFlash 中的数据拷贝到 DRAM 中。命令格式如下：

```
cp [.b, .w, .l] source target count
```

cp 命令同样以.b、.w 和.l 来指定操作格式，source 为源地址，target 为目的地址，count 为拷贝的长度。我们使用.l 格式将 0xC0100000 处的地址拷贝到 0xC0100100 处，长度为 0x10 个内存块(0x10 * 4=64 个字节)，命令如下所示：

```
cp.l c0100000 c0100100 10
```

结果如图 10.3.3.8 所示：

```
STM32MP> md.l c0100000 10
c0100000: 0a0a0a0a 0a0a0a0a 0a0a0a0a 0a0a0a0a    ................
c0100010: 0a0a0a0a 0a0a0a0a 0a0a0a0a 0a0a0a0a    ................
c0100020: 0a0a0a0a 0a0a0a0a 0a0a0a0a 0a0a0a0a    ................
c0100030: 0a0a0a0a 0a0a0a0a 0a0a0a0a 0a0a0a0a    ................
STM32MP> md.l c0100100 10
c0100100: e1a0000d fa000622 e320f000 e320f000    ....".... ... .
c0100110: e320f000 e320f000 e320f000 e320f000    .. ... ... ... .
c0100120: e51fd0e8 e58de000 e14fe000 e58de004    .........O.....
c0100130: e3a0d013 e169f00d e1a0e00f e1b0f00e    ......i.........
STM32MP> cp.l c0100000 c0100100 10
STM32MP> md.l c0100000 10
c0100000: 0a0a0a0a 0a0a0a0a 0a0a0a0a 0a0a0a0a    ................
c0100010: 0a0a0a0a 0a0a0a0a 0a0a0a0a 0a0a0a0a    ................
c0100020: 0a0a0a0a 0a0a0a0a 0a0a0a0a 0a0a0a0a    ................
c0100030: 0a0a0a0a 0a0a0a0a 0a0a0a0a 0a0a0a0a    ................
STM32MP>
```

图 10.3.3.8 cp 命令操作结果

在图 10.3.3.8 中，先使用 md.l 命令打印出地址 0xC0100000 和 0xC0100100 处的数据，然后使用命令 cp.l 将 0xC0100000 处的数据拷贝到 0xC0100100 处。最后使用命 md.l 查看 0xC0100100 处的数据有没有变化，检查拷贝是否成功。

6、cmp 命令

cmp 是比较命令，用于比较两段内存的数据是否相等，命令格式如下：

```
cmp [.b, .w, .l] addr1 addr2 count
```

cmp 命令同样以.b、.w 和.l 来指定操作格式，addr1 为第一段内存首地址，addr2 为第二段内存首地址，count 为要比较的长度。我们使用.l 格式来比较 0xC0100000 和 0xC0100100 这两个地址数据是否相等，比较长度为 0x10 个内存块(16 * 4=64 个字节)，命令如下所示：

```
cmp.l c0100000 c0100100 10
```

结果如图 10.3.3.9 所示：

```
STM32MP> cmp.l c0100000 c0100100 10
Total of 16 word(s) were the same
STM32MP>
```

图 10.3.3.9 cmp 命令比较结果

从图 10.3.3.9 可以看出两段内存的数据相等。我们再随便挑两段内存比较一下，比如地址 0xC0100000 和 0xC0100200，长度为 0X10，比较结果如图 10.3.3.10 所示：

```
STM32MP> cmp.l c0100000 c0100200 10
word at 0xc0100000 (0xa0a0a0a) != word at 0xc0100200 (0xe24dd048)
Total of 0 word(s) were the same
STM32MP>
```

图 10.3.3.10 cmp 命令比较结果

从图 10.3.3.10 可以看出，0xC0100000 处的数据和 0xC0100200 处的数据就不一样。

10.3.4 网络操作命令

uboot 是支持网络的，我们在移植 uboot 的时候一般都要调通网络功能，因为在移植 linux kernel 的时候需要使用到 uboot 的网络功能做调试。uboot 支持大量的网络相关命令，比如 dhcp、ping、nfs 和 tftpboot，我们接下来依次学习一下这几个和网络有关的命令。

建议开发板和主机 PC 都连接到同一个路由器上！最后设置表 10.3.4.1 所示的几个环境变量：

环境变量	描述
ipaddr	开发板 ip 地址，可以不设置，使用 dhcp 命令来从路由器获取 IP 地址。
ethaddr	开发板的 MAC 地址，一定要设置。

gatewayip	网关地址。
netmask	子网掩码。
serverip	服务器 IP 地址，也就是 Ubuntu 主机 IP 地址，用于调试代码。

表 10.3.4.1 网络相关环境变量

表 10.3.4.1 中环境变量设置命令如下所示：

```
setenv ipaddr 192.168.1.250
setenv ethaddr 00:04:9f:04:d2:35
setenv gatewayip 192.168.1.1
setenv netmask 255.255.255.0
setenv serverip 192.168.1.249
saveenv
```

注意，网络地址环境变量的设置要根据自己的实际情况，确保 Ubuntu 主机和开发板的 IP 地址在同一个网段内，比如我现在的开发板和电脑都在 192.168.1.0 这个网段内，所以设置开发板的 IP 地址为 192.168.1.250，我的 Ubuntu 主机的地址为 192.168.1.249，因此 serverip 就是 192.168.1.249。ethaddr 为网络 MAC 地址，是一个 48bit 的地址，如果在同一个网段内有多个开发板的话一定要保证每个开发板的 ethaddr 是不同的，否则通信会有问题！设置好网络相关的环境变量以后就可以使用网络相关命令了。

1、ping 命令

开发板的网络能否使用，是否可以和服务器(Ubuntu 主机)进行通信，通过 ping 命令就可以验证，直接 ping 服务器的 IP 地址即可，比如我的服务器 IP 地址为 192.168.1.249，命令如下：

```
ping 192.168.1.249
```

结果如图 10.3.4.1 所示：

```
STM32MP> ping 192.168.1.249
ethernet@5800a000 Waiting for PHY auto negotiation to complete.............. done
Using ethernet@5800a000 device
host 192.168.1.249 is alive
STM32MP>
```

图 10.3.4.1 ping 命令

从图 10.3.4.1 可以看出，192.168.1.249 这个主机存在，说明 ping 成功，uboot 的网络工作正常。

注意！只能在 uboot 中 ping 其他的机器，其他机器不能 ping uboot，因为 uboot 没有对 ping 命令做处理，如果用其他的机器 ping uboot 的话会失败！

2、dhcp 命令

dhcp 用于从路由器获取 IP 地址，前提是开发板得连接到路由器上的，如果开发板是和电脑直连的，那么 dhcp 命令就会失效。直接输入 dhcp 命令即可通过路由器获取到 IP 地址，如图 10.3.4.2 所示：

```
STM32MP> dhcp
BOOTP broadcast 1
BOOTP broadcast 2
BOOTP broadcast 3
DHCP client bound to address 192.168.1.7 (787 ms)
STM32MP>
```

图 10.3.4.2 dhcp 命令

从图 10.3.4.2 可以看出，开发板通过 dhcp 获取到的 IP 地址为 192.168.1.7。DHCP 不单

单是获取 IP 地址，其还会通过 TFTP 来启动 linux 内核，输入“? dhcp”即可查看 dhcp 命令详细的信息，如图 10.3.4.3 所示：

```
STM32MP> ? dhcp
dhcp - boot image via network using DHCP/TFTP protocol

Usage:
dhcp [loadAddress] [[hostIPaddr:]bootfilename]
STM32MP>
```

图 10.3.4.3 dhcp 命令使用查询

3、nfs 命令

nfs(Network File System)网络文件系统，通过 nfs 可以在计算机之间通过网络来分享资源，比如我们将 linux 镜像和设备树文件放到 Ubuntu 中，然后在 uboot 中使用 nfs 命令将 Ubuntu 中的 linux 镜像和设备树下载到开发板的 DRAM 中。这样做的目的是为了方便调试 linux 镜像和设备树，也就是网络调试，网络调试是 Linux 开发中最常用的调试方法。原因是嵌入式 linux 开发不像单片机开发，可以直接通过 JLINK 或 STLink 等仿真器将代码直接烧写到单片机内部的 flash 中，嵌入式 Linux 通常是烧写到 EMMC、NAND Flash、SPI Flash 等外置 flash 中，但是嵌入式 Linux 开发也没有 MDK，IAR 这样的 IDE，更没有烧写算法，因此不可能通过点击一个“download”按钮就将固件烧写到外部 flash 中。虽然半导体厂商一般都会提供一个烧写固件的软件，但是这个软件使用起来比较复杂，这个烧写软件一般用于量产的。其远没有 MDK、IAR 的一键下载方便，在 Linux 内核调试阶段，如果用这个烧写软件的话将会非常浪费时间，而这个时候网络调试的优势就显现出来了，可以通过网络将编译好的 linux 镜像和设备树文件下载到 DRAM 中，然后就可以直接运行。

我们一般使用 uboot 中的 nfs 命令将 Ubuntu 中的文件下载到开发板的 DRAM 中，在使用之前需要开启 Ubuntu 主机的 NFS 服务，并且要新建一个 NFS 使用的目录，以后所有要通过 NFS 访问的文件都需要放到这个 NFS 目录中。Ubuntu 的 NFS 服务开启我们在 4.2.1 小节已经详细讲解过了，包括 NFS 文件目录的创建，如果忘记的话可以去查看一下 4.2.1 小节。我设置的 NFS 文件目录为：/home/zuozhongkai/linux/nfs，uboot 中的 nfs 命令格式如下所示：

```
nfs [loadAddress] [[hostIPaddr:]bootfilename]
```

loadAddress 是要保存的 DRAM 地址，[[hostIPaddr:]bootfilename]是要下载的文件地址。这里我们将正点原子官方编译出来的 Linux 镜像文件 uImage 下载到开发板 DRAM 的 0xC2000000 这个地址处(ST 官方指定的 Linux 内核加载地址)。正点原子编译出来的 uImage 文件已经放到了开发板光盘中，路径为：开发板光盘 A→8、系统镜像→1、教程系统镜像→linux-mp1-5.4.31-g032f9dcc0-v1.0→uImage。将整个文件通过 FileZilla 发送到 Ubuntu 中的 NFS 目录下：/home/zuozhongkai/linux/nfs，完成以后的 NFS 目录如图 10.3.4.4 所示：

```
zuozhongkai@ubuntu:~/linux/nfs$ ls
uImage
zuozhongkai@ubuntu:~/linux/nfs$
```

图 10.3.4.4 nfs 目录

准备好以后就可以使用 nfs 命令来将 uImage 下载到开发板 DRAM 的 0XC2000000 地址处，命令如下：

```
nfs C2000000 192.168.1.249:/home/zuozhongkai/linux/nfs/uImage
```

命令中的“C2000000”表示 uImage 在 DDR 中的首地址，“192.168.1.249:/home/zuozhongkai/linux/nfs/uImage”表示 uImage 在 192.168.1.249 这个主机中，路径为/home/zuozhongkai/linux/nfs/uImage。下载过程如图 10.3.4.5 所示：

```
STM32MP> nfs 0xc2000000 192.168.1.249:/home/zuozhongkai/linux/nfs/uImage
Using ethernet@5800a000 device
File transfer via NFS from server 192.168.1.249; our IP address is 192.168.1.250
Filename '/home/zuozhongkai/linux/nfs/uImage'.
Load address: 0xc2000000
Loading: #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         ######################T ##########################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         ################################################################
done
Bytes transferred = 7313888 (6f99e0 hex)
STM32MP>
```

图 10.3.4.5 nfs 命令下载 uImage 过程

图 10.3.4.5 中通过网络下载 uImage 的时候会打印出‘#’表示正在下载，如果网络环境不好，下载有干扰的话就会打印出一个‘T’，这个没事的，只要它能正常下载成功，最终打印出“done”即可。

下载完成以后查看 0xC2000000 地址处的数据，使用命令 md.b 来查看前 0x100(256)个字节的数据，如图 10.3.4.6 所示：

```
STM32MP> md.b c2000000 100
c2000000: 27 05 19 56 77 46 15 2c 5f a3 c3 21 00 6f 99 a0    '..VwF.,_..!.o..
c2000010: c2 00 00 40 c2 00 00 40 14 80 31 03 05 02 02 00    ...@...@..1.....
c2000020: 4c 69 6e 75 78 2d 35 2e 34 2e 33 31 2d 67 30 33    Linux-5.4.31-g03
c2000030: 32 66 39 64 63 63 30 00 00 00 00 00 00 00 00 00    2f9dcc0.........
c2000040: 00 00 a0 e1 00 00 a0 e1 00 00 a0 e1 00 00 a0 e1    ................
c2000050: 00 00 a0 e1 00 00 a0 e1 00 00 a0 e1 00 00 a0 e1    ................
c2000060: 05 00 00 ea 18 28 6f 01 00 00 00 00 a0 99 6f 00    .....(o.......o.
c2000070: 01 02 03 04 45 45 45 45 a8 66 00 00 00 90 0f e1    ....EEEE.f......
c2000080: fe 0d 00 eb 01 70 a0 e1 02 80 a0 e1 00 20 0f e1    .....p....... ..
c2000090: 03 00 12 e3 01 00 00 1a 17 00 a0 e3 56 34 12 ef    ............V4..
c20000a0: 00 00 0f e1 1a 00 20 e2 1f 00 10 e3 1f 00 c0 e3    ...... .........
c20000b0: d3 00 80 e3 04 00 00 1a 01 0c 80 e3 0c e0 8f e2    ................
c20000c0: 00 f0 6f e1 0e f3 2e e1 6e 00 60 e1 00 f0 21 e1    ..o.....n.`...!.
c20000d0: 09 f0 6f e1 00 00 00 00 00 00 00 00 00 00 00 00    ..o.............
c20000e0: 0f 40 a0 e1 3e 43 04 e2 02 49 84 e2 0f 00 a0 e1    .@..>C...I......
c20000f0: 04 00 50 e1 6c 02 9f 35 0f 00 80 30 00 00 54 31    ..P.l..5...0..T1
STM32MP>
```

图 10.3.4.6 下载的数据

再使用 winhex 软件来查看刚刚下载的 uImage，检查一下前面的数据是否和图 10.3.4.6 中的一致，结果如图 10.3.4.7 所示：

uImage

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00000000	27	05	19	56	77	46	15	2C	5F	A3	C3	21	00	6F	99	A0
00000010	C2	00	00	40	C2	00	00	40	14	80	31	03	05	02	02	00
00000020	4C	69	6E	75	78	2D	35	2E	34	2E	33	31	2D	67	30	33
00000030	32	66	39	64	63	63	30	00	00	00	00	00	00	00	00	00
00000040	00	00	A0	E1	00	00	A0	E1	00	00	A0	E1	00	00	A0	E1
00000050	00	00	A0	E1	00	00	A0	E1	00	00	A0	E1	00	00	A0	E1
00000060	05	00	00	EA	18	28	6F	01	00	00	00	00	A0	99	6F	00
00000070	01	02	03	04	45	45	45	45	A8	66	00	00	00	90	0F	E1
00000080	FE	0D	00	EB	01	70	A0	E1	02	80	A0	E1	00	20	0F	E1
00000090	03	00	12	E3	01	00	00	1A	17	00	A0	E3	56	34	12	EF
000000A0	00	00	0F	E1	1A	00	20	E2	1F	00	10	E3	1F	00	C0	E3
000000B0	D3	00	80	E3	04	00	00	1A	01	0C	80	E3	0C	E0	8F	E2
000000C0	00	F0	6F	E1	0E	F3	2E	E1	6E	00	60	E1	00	F0	21	E1
000000D0	09	F0	6F	E1	00	00	00	00	00	00	00	00	00	00	00	00
000000E0	0F	40	A0	E1	3E	43	04	E2	02	49	84	E2	0F	00	A0	E1
000000F0	04	00	50	E1	6C	02	9F	35	0F	00	80	30	00	00	54	31
00000100	01	40	84	33	9D	00	00	2B	8E	0F	8F	E2	4E	1C	90	E8

图 10.3.4.7 winhex 查看 uImage

可以看出图 10.3.4.和图 10.3.4.7 中的前 0x100 个字节的数据一致，说明 nfs 命令下载到的 uImage 是正确的。

4、tftp 命令

tftp 命令的作用和 nfs 命令一样，都是用于通过网络下载东西到 DRAM 中，只是 tftp 命令使用的是 TFTP 协议，Ubuntu 主机作为 TFTP 服务器。因此需要在 Ubuntu 上搭建 TFTP 服务器，需要安装 tftp-hpa 和 tftpd-hpa，命令如下：

```
sudo apt-get install tftp-hpa tftpd-hpa
sudo apt-get install xinetd
```

和 NFS 一样，TFTP 也需要一个文件夹来存放文件，在用户目录下新建一个目录，命令如下：

```
mkdir /home/zuozhongkai/linux/tftpboot
chmod 777 /home/zuozhongkai/linux/tftpboot
```

这样我就在我的电脑上创建了一个名为 tftpboot 的目录(文件夹)，路径为/home/zuozhongkai /linux/tftpboot。注意！我们要给 tftpboot 文件夹权限，否则的话 uboot 不能从 tftpboot 文件夹里面下载文件。

最后配置 tftp，新建文件/etc/xinetd.d/tftp，如果没有/etc/xinetd.d 目录的话自行创建，然后在里面输入如下内容：

示例代码 10.3.4.1 /etc/xinetd.d/tftp 文件内容

```
1  server tftp
2  {
3      socket_type     = dgram
4      protocol        = udp
5      wait            = yes
6      user            = root
7      server          = /usr/sbin/in.tftpd
8      server_args     = -s /home/zuozhongkai/linux/tftpboot/
9      disable         = no
10     per_source      = 11
11     cps             = 100 2
```

```
12     flags           = IPv4
13 }
```

完了以后启动 tftp 服务，命令如下：

```
sudo service tftpd-hpa start
```

打开/etc/default/tftpd-hpa 文件，将其修改为如下所示内容：

示例代码 10.3.4.2 /etc/default/tftpd-hpa 文件内容

```
1 # /etc/default/tftpd-hpa
2
3 TFTP_USERNAME="tftp"
4 TFTP_DIRECTORY="/home/liangwencong/linux/tftpboot"
5 TFTP_ADDRESS=":69"
6 TFTP_OPTIONS="-l -c -s"
```

TFTP_DIRECTORY 就是我们上面创建的 tftp 文件夹目录，以后我们就将所有需要通过 TFTP 传输的文件都放到这个文件夹里面，并且要给予这些文件相应的权限。

最后输入如下命令， 重启 tftp 服务器：

```
sudo service tftpd-hpa restart
```

tftp 服务器已经搭建好了，接下来就是使用了。将 uImage 镜像文件拷贝到 tftpboot 文件夹中，并且给予 uImage 相应的权限，命令如下：

```
cp uImage /home/zuozhongkai/linux/tftpboot/
cd /home/zuozhongkai/linux/tftpboot/
chmod 777 uImage
```

万事俱备，只剩验证了，uboot 中的 tftp 命令格式如下：

```
tftpboot [loadAddress] [[hostIPaddr:]bootfilename]
```

看起来和 nfs 命令格式一样的， loadAddress 是文件在 DRAM 中的存放地址，[[hostIPaddr:]bootfilename]是要从 Ubuntu 中下载的文件。但是和 nfs 命令的区别在于，tftp 命令不需要输入文件在 Ubuntu 中的完整路径，只需要输入文件名即可。比如我们现在将 tftpboot 文件夹里面的 uImage 文件下载到开发板 DRAM 的 0XC2000000 地址处，命令如下：

```
tftp C2000000 uImage
```

下载过程如图 10.3.4.8 所示：

```
STM32MP> tftp c2000000 uImage
Using ethernet@5800a000 device
TFTP from server 192.168.1.249; our IP address is 192.168.1.250
Filename 'uImage'.
Load address: 0xc2000000
Loading: #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         ##########################################
         2.3 MiB/s
done
Bytes transferred = 7313888 (6f99e0 hex)
STM32MP>
```

图 10.3.4.8 tftp 命令下载过程

从图 10.3.4.8 可以看出，uImage 下载成功了，网速为 2.3MibB/s，文件大小为 7312880 字节。同样的，可以使用 md.b 命令来查看前 0x100 个字节的数据是否和图 10.3.4.7 中的相等。有时候使用 tftp 命令从 Ubuntu 中下载文件的时候会出现如图 10.3.4.9 所示的错误提示：

```
STM32MP> tftp c2000000 uImage
Using ethernet@5800a000 device
TFTP from server 192.168.1.249; our IP address is 192.168.1.250
Filename 'uImage'.
Load address: 0xc2000000
Loading: *
TFTP error: 'Permission denied' (0)
Starting again

STM32MP>
```

图 10.3.4.9 tftp 下载出错

在图 10.3.4.9 中可以看到“TFTP error: 'Permission denied' (0)”这样的错误提示，提示没有权限，出现这个错误一般有两个原因：

①、在 Ubuntu 中创建 tftpboot 目录的时候没有给予 tftboot 相应的权限。

②、tftpboot 目录中要下载的文件没有给予相应的权限。

针对上述两个问题，使用命令“chmod 777 xxx”来给予权限，其中“xxx”就是要给予权限的文件或文件夹。

好了，uboot 中关于网络的命令就讲解到这里，我们最常用的就是 ping、nfs 和 tftp 这三个命令。使用 ping 命令来查看网络的连接状态，使用 nfs 和 tftp 命令来从 Ubuntu 主机中下载文件。

10.3.5 EMMC 和 SD 卡操作命令

uboot 支持 EMMC 和 SD 卡，因此也要提供 EMMC 和 SD 卡的操作命令。一般认为 EMMC 和 SD 卡是同一个东西，所以没有特殊说明，本教程统一使用 MMC 来代指 EMMC 和 SD 卡。uboot 中常用于操作 MMC 设备的命令为“mmc”。

mmc 是一系列的命令，其后可以跟不同的参数，输入“？mmc”即可查看 mmc 有关的命令，如图 10.3.5.1 所示：

```
STM32MP> ? mmc
mmc - MMC sub system

Usage:
mmc info - display info of the current MMC device
mmc read addr blk# cnt
mmc write addr blk# cnt
mmc erase blk# cnt
mmc rescan
mmc part - lists available partition on current mmc device
mmc dev [dev] [part] - show or set current mmc device [partition]
mmc list - lists available devices
mmc hwpartition [args...] - does hardware partitioning
  arguments (sizes in 512-byte blocks):
    [user [enh start cnt] [wrrel {on|off}]] - sets user data area attributes
    [gp1|gp2|gp3|gp4 cnt [enh] [wrrel {on|off}]] - general purpose partition
    [check|set|complete] - mode, complete set partitioning completed
  WARNING: Partitioning is a write-once setting once it is set to complete.
  Power cycling is required to initialize partitions after set to complete.
mmc bootbus dev boot_bus_width reset_boot_bus_width boot_mode
 - Set the BOOT_BUS_WIDTH field of the specified device
mmc bootpart-resize <dev> <boot part size MB> <RPMB part size MB>
 - Change sizes of boot and RPMB partitions of specified device
mmc partconf dev [boot_ack boot_partition partition_access]
 - Show or change the bits of the PARTITION_CONFIG field of the specified device
mmc rst-function dev value
 - Change the RST_n_FUNCTION field of the specified device
   WARNING: This is a write-once field and 0 / 1 / 2 are the only valid values.
mmc setdsr <value> - set DSR register value

STM32MP>
```

图 10.3.5.1 mmc 命令

从图 10.3.5.1 可以看出，mmc 后面跟不同的参数可以实现不同的功能，如表 10.3.5.1 所

示：

命令	描述
mmc info	输出 MMC 设备信息
mmc read	读取 MMC 中的数据。
mmc wirte	向 MMC 设备写入数据。
mmc rescan	扫描 MMC 设备。
mmc part	列出 MMC 设备的分区。
mmc dev	切换 MMC 设备。
mmc list	列出当前有效的所有 MMC 设备。
mmc hwpartition	设置 MMC 设备的分区。
mmc bootbus……	设置指定 MMC 设备的 BOOT_BUS_WIDTH 域的值。
mmc bootpart……	设置指定 MMC 设备的 boot 和 RPMB 分区的大小。
mmc partconf……	设置指定 MMC 设备的 PARTITION_CONFG 域的值。
mmc rst	复位 MMC 设备
mmc setdsr	设置 DSR 寄存器的值。

表 10.3.5.1 mmc 命令

1、mmc info 命令

mmc info 命令用于输出当前选中的 mmc info 设备的信息，输入命令“mmc info”即可，如图 10.3.5.2 所示：

```
STM32MP> mmc info
Device: STM32 SD/MMC
Manufacturer ID: 15
OEM: 100
Name: 8GTF4
Bus Speed: 52000000
Mode: MMC High Speed (52MHz)
Rd Block Len: 512
MMC version 5.1
High Capacity: Yes
Capacity: 7.3 GiB
Bus Width: 8-bit
Erase Group Size: 512 KiB
HC WP Group Size: 8 MiB
User Capacity: 7.3 GiB WRREL
Boot Capacity: 4 MiB ENH
RPMB Capacity: 512 KiB ENH
STM32MP>
```

图 10.3.5.2 mmc info 命令

从图 10.3.5.2 可以看出，当前选中的 MMC 设备是 EMMC，EMMC 芯片版本为 5.1，容量为 7.3GiB(EMMC 为 8GB)，速度为 52000000Hz=52MHz，8 位宽的总线。还有一个与 mmc info 命令相同功能的命令：mmcinfo，“mmc”和“info”之间没有空格。

2、mmc rescan 命令

mmc rescan 命令用于扫描当前开发板上所有的 MMC 设备，包括 EMMC 和 SD 卡，输入“mmc rescan”即可。

3、mmc list 命令

mmc list 命令用于来查看当前开发板一共有几个 MMC 设备，输入“mmc list”，结果如图 10.3.5.3 所示：

```
STM32MP> mmc list
STM32 SD/MMC: 0
STM32 SD/MMC: 1 (eMMC)
STM32MP>
```

图 10.3.5.3 扫描 MMC 设备

可以看出当前开发板有两个 MMC 设备：STM32 SD/MMC:0 和 STM32 SD/MMC:1 (eMMC)，这是因为我现在用的是 EMMC 版本的核心板，加上 SD 卡一共有两个 MMC 设备，STM32 SD/MMC:0 是 SD 卡，STM32 SD/MMC:1 (eMMC)是 EMMC。默认会将 EMMC 设置为当前 MMC 设备，这就是为什么输入“mmc info”查询到的是 EMMC 设备信息，而不是 SD 卡。要想查看 SD 卡信息，就要使用命令“mmc dev”来将 SD 卡设置为当前的 MMC 设备。

4、mmc dev 命令

mmc dev 命令用于切换当前 MMC 设备，命令格式如下：

```
mmc dev [dev] [part]
```

[dev]用来设置要切换的 MMC 设备号，[part]是分区号，如果不写分区号的话默认为分区 0。使用如下命令切换到 SD 卡：

```
mmc dev 0          //切换到 SD 卡，0 为 SD 卡，1 为 eMMC
```

结果如图 10.3.5.4 所示：

```
STM32MP> mmc dev 0
switch to partitions #0, OK
mmc0 is current device
STM32MP>
```

图 10.3.5.4 切换到 SD 卡

从图 30.4.5.4 可以看出，切换到 SD 卡成功，mmc0 为当前的 MMC 设备，输入命令“mmc info”即可查看 SD 卡的信息(要插入 SD 卡)，结果如图 10.3.5.5 所示：

```
STM32MP> mmc info
Device: STM32 SD/MMC
Manufacturer ID: 3
OEM: 5344
Name: SC16G
Bus Speed: 50000000
Mode: SD High Speed (50MHz)
Rd Block Len: 512
SD version 3.0
High Capacity: Yes
Capacity: 14.8 GiB
Bus Width: 4-bit
Erase Group Size: 512 Bytes
STM32MP>
```

图 10.3.5.5 SD 信息

从图 10.3.5.5 可以看出当前 SD 卡版本号为 3.0，容量为 14.8GiB(16GB 的 SD 卡)，4 位宽的总线。

5、mmc part 命令

有时候 SD 卡或者 EMMC 会有多个分区，可以使用命令“mmc part”来查看其分区，比如查看 EMMC 的分区情况，输入如下命令：

```
mmc dev 1      //切换到 EMMC
mmc part       //查看 EMMC 分区
```

结果如图 10.3.5.6 所示：

```
STM32MP> mmc dev 1
switch to partitions #0, OK
mmc1(part 0) is current device
STM32MP> mmc part

Partition Map for MMC device 1  --   Partition Type: EFI

Part    Start LBA       End LBA         Name
        Attributes
        Type GUID
        Partition GUID
  1     0x00000400      0x00e8fbff      "ssbl"
        attrs:  0x0000000000000000
        type:   8da63339-0007-60c0-c436-083ac8230908
        guid:   c6b6280e-7e2d-4f06-9606-438916e8b7d9
STM32MP>
```

图 10.3.5.6 查看 EMMC 分区

从图 10.3.5.6 中可以看出，此时 EMMC 是 EFI 类型并且有 1 个分区，如果你的核心板 EMMC 烧写过系统的话会有 3 个分区，正点原子出厂开发板都烧写系统，所以一般是 3 个分区，如图 10.3.5.7 所示：

```
Hit any key to stop autoboot:  0
STM32MP> mmc dev 1
switch to partitions #0, OK
mmc1(part 0) is current device
STM32MP> mmc part

Partition Map for MMC device 1  --   Partition Type: EFI

Part    Start LBA       End LBA         Name
        Attributes
        Type GUID
        Partition GUID
  1     0x00000400      0x000013ff      "ssbl"
        attrs:  0x0000000000000000
        type:   8da63339-0007-60c0-c436-083ac8230908
        guid:   3df85fdb-3108-41af-9a8c-ba44821e09b3
  2     0x00001400      0x000213ff      "boot"
        attrs:  0x0000000000000004
        type:   0fc63daf-8483-4772-8e79-3d69d8477de4
        type:   linux
        guid:   4a8d95b5-d425-4583-bb7d-e9c41c7b8382
  3     0x00021400      0x00e8fbff      "rootfs"
        attrs:  0x0000000000000000
        type:   0fc63daf-8483-4772-8e79-3d69d8477de4
        type:   linux
        guid:   491f6117-415d-4f53-88c9-6e0de54deac6
STM32MP>
```

图 10.3.5.7 烧写过系统的分区

从图 10.3.5.7 中可以看出，有 3 个分区，第一个分区为名字为“ssbl”，用来存放 uboot 镜像，范围为：扇区 0x400~0x13ff。第二个分区名字为“boot”，用来存放 linux 内核镜像，范围为：扇区 0x1400~0x213ff。第三个分区名字为“rootfs”，这个是根文件系统分区，占用了剩余的所有扇区，也就是扇区 0x21400~0xe8fbff。

如果要将 EMMC 的分区 2 设置为当前 MMC 设置分区，可以使用如下命令：

```
mmc dev 1 2
```

结果如图 10.3.5.8 所示：

```
STM32MP> mmc dev 1 2
switch to partitions #2, OK
mmc1(part 2) is current device
STM32MP>
```

图 10.3.5.8 设置 EMMC 分区 2 为当前设备

6、mmc read 命令

mmc read 命令用于读取 mmc 设备的数据，命令格式如下：

```
mmc read addr blk# cnt
```

addr 是数据读取到 DRAM 中的地址，blk 是要读取的块起始地址(十六进制)，一个块是 512 字节，这里的块和扇区是一个意思，在 MMC 设备中我们通常说扇区，cnt 是要读取的块数量(十六进制)。比如从 EMMC 的第 1024(0x400)个块开始，读取 16(0x10)个块的数据到 DRAM 的 0XC0000000 地址处，命令如下：

```
mmc dev 1                    //切换到 EMMC
mmc read c0000000 400 10     //读取数据
```

结果如图 10.3.5.9 所示：

```
STM32MP> mmc dev 1
switch to partitions #0, OK
mmc1(part 0) is current device
STM32MP> mmc read c0000000 400 10

MMC read: dev # 1, block # 1024, count 16 ... 16 blocks read: OK
STM32MP>
```

图 10.3.5.9 mmc read 命令

这里我们还看不出来读取是否正确，通过 md.b 命令查看 0xc0000000 处的数据就行了，查看 16*512=8192(0x2000)个字节的数据，命令如下：

```
md.b c0000000 2000
```

结果如图 10.3.5.10 所示：

```
STM32MP> md.b c0000000 2000
c0000000: 53 54 4d 32 00 00 00 00 00 00 00 00 00 00 00 00    STM2............
c0000010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
c0000020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
c0000030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
c0000040: 00 00 00 00 aa 5a aa 04 00 00 01 00 f4 36 0d 00    .....Z.......6..
c0000050: 00 00 10 c0 00 00 00 00 00 00 10 c0 00 00 00 00    ................
c0000060: 00 00 00 00 01 00 00 00 01 00 00 00 00 00 00 00    ................
c0000070: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
c0000080: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
c0000090: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
c00000a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
c00000b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
c00000c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
c00000d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
c00000e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
c00000f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
c0000100: b8 00 00 ea 14 f0 9f e5 14 f0 9f e5 14 f0 9f e5    ................
c0000110: 14 f0 9f e5 14 f0 9f e5 14 f0 9f e5 14 f0 9f e5    ................
c0000120: 60 00 10 c0 c0 00 10 c0 20 01 10 c0 80 01 10 c0    `....... .......
```

图 10.3.5.10 读取到的数据(部分截图)

从图 30.4.5.9 可以看出，前面的 265 个字节就是 STM32MP1 的头部信息。

10.3.6 EXT 格式文件系统操作命令

uboot 有 ext2 和 ext4 这两种格式的文件系统的操作命令， STM32MP1 的系统镜像都是 ext4 格式的，所以我们重点讲解一下和 ext4 有关的三个命令：ext4ls、ext4ls 和 ext4write。注意，由于只有 linux 内核、设备树和根文件系统是以 ext4 格式存放在 EMMC 中的，因此在测试 ext4 相关命令的时候要先确保 EMMC 里面烧写了完整的出厂系统！

1、ext4ls

ext4ls 命令用于查询 EXT4 格式设备的目录和文件信息，命令格式如下：

```
ext4ls <interface> [<dev[:part]>] [directory]
```

interface 是要查询的接口，比如 mmc，dev 是要查询的设备号，part 是要查询的分区，directory 是要查询的目录。比如查询 EMMC 分区 2 中的所有的目录和文件，输入命令：

```
ext4ls mmc 1:2
```

结果如图 10.3.6.1 所示：

```
STM32MP> ext4ls mmc 1:2
<DIR>       1024 .
<DIR>       1024 ..
            2943 boot.scr.uimg
<DIR>       1024 lost+found
<DIR>       1024 mmc0_extlinux
<DIR>       1024 mmc1_extlinux
           92670 splash.bmp
           74594 stm32mp157d-atk.dtb
           74014 stm32mp157d-atk-hdmi.dtb
           74510 stm32mp157d-atk-spdif.dtb
         8125872 uImage
         3632241 uInitrd
STM32MP>
```

图 10.3.6.1　EMMC 分区 2 文件查询

从上图可以看出，emmc 的分区 2 中存放了 10 个文件，其中比较重要的就是三个.dtb 设备树文件和 Linux 内核的 uImage 镜像文件。

2、ext4load 命令

extload 命令用于将指定的文件读取到 DRAM 中，命令格式如下：

```
fatload <interface> [<dev[:part]> [<addr> [<filename> [bytes [pos]]]]]
```

interface 为接口，比如 mmc，dev 是设备号，part 是分区，addr 是保存在 DRAM 中的起始地址，filename 是要读取的文件名字。bytes 表示读取多少字节的数据，如果 bytes 为 0 或者省略的话表示读取整个文件。pos 是要读的文件相对于文件首地址的偏移，如果为 0 或者省略的话表示从文件首地址开始读取。我们将图 10.3.6.1 中 EMMC 分区 2 中的 uImage 文件读取到 DRAM 中的 0XC2000000 地址处，命令如下：

```
ext4load mmc 1:2 C2000000 uImage
```

操作过程如图 10.3.6.2 所示：

```
STM32MP> ext4load mmc 1:2 c2000000 uImage
8125872 bytes read in 207 ms (37.4 MiB/s)
STM32MP>
```

图 10.3.6.2 读取过程

从上图可以看出在 207ms 内读取了 8125872 个字节的数据，速度为 37.4MiB/s，速度是非常快的，因为这是从 EMMC 里面读取的，而 EMMC 是 8 位的，速度肯定会很快。

3、ext4write

ext4wirte 命令用于将 DRAM 中的数据写入到 MMC 设备中，命令格式如下：

```
ext4write <interface> <dev[:part]> <addr> <absolute filename path> [sizebytes] [file offset]
```

interface 为接口，比如 mmc；dev 是设备号；part 是分区；addr 是要写入的数据在 DRAM 中的起始地址；absolute filename path 是写入的数据文件名字，注意是要带有绝对路径，以‘/’开始；sizebytes 表示要写入多少字节的数据；file offset 为文件偏移。我们可以通过 fatwrite 命令在 uboot 中更新 linux 镜像文件和设备树。我们以更新 linux 镜像文件 uImage 为例，首先将正点原子 STM32MP157 开发板提供的 uImage 镜像文件拷贝到 Ubuntu 中的 tftpboot 目录下，使用命令 tftp 将 uImage 下载到 DRAM 的 0XC0000000 地址处，命令如下：

```
tftp C0000000 uImage
```

下载过程如图 10.3.6.3 所示：

```
STM32MP> tftp c0000000 uImage
Using ethernet@5800a000 device
TFTP from server 192.168.1.249; our IP address is 192.168.1.250
Filename 'uImage'.
Load address: 0xc0000000
Loading: #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         #################################################################
         ##########################################
         2.4 MiB/s
done
Bytes transferred = 7313888 (6f99e0 hex)
STM32MP>
```

图 10.3.6.3 uImage 下载过程

uImage 大小为 7313888(0X6F99E0)个字节，接下来使用命令 ext4write 将其写入到 EMMC 的分区 2 中。为了和原有的 uImage 文件区分，我们将要写入的文件命名为 test_uImage，命令如下：

```
ext4write mmc 1:2 c0000000 /test_uImage 0x6f99e0
```

结果如图 10.4.6.4 所示：

```
STM32MP> ext4write mmc 1:2 c0000000 /test_uImage 0x6f99e0
File System is consistent
update journal finished
7313888 bytes written in 335 ms (20.8 MiB/s)
STM32MP>
```

图 10.3.6.4 将 uImage 烧写到 EMMC 扇区 2 中

完成以后使用“ext4ls”命令查看一下 EMMC 分区 2 里面的文件，结果如图 10.3.6.5 所示：

```
STM32MP> ext4ls mmc 1:2
<DIR>       1024 .
<DIR>       1024 ..
            2943 boot.scr.uimg
<DIR>       1024 lost+found
<DIR>       1024 mmc0_extlinux
<DIR>       1024 mmc1_extlinux
           92670 splash.bmp
           74594 stm32mp157d-atk.dtb
           74014 stm32mp157d-atk-hdmi.dtb
           74510 stm32mp157d-atk-spdif.dtb
         8125872 uImage
         3632241 uInitrd
         7313888 test_uImage
STM32MP>
```

图 10.3.6.5 写入的 test_uImage 文件

从图 10.3.6.5 可以看出，test_uImage 写入成功。

10.3.7 BOOT 操作命令

uboot 的本质工作是引导 Linux，所以 uboot 肯定有相关的 boot(引导)命令来启动 Linux。常用的跟 boot 有关的命令有：bootm、bootz 和 boot。

1、bootm 命令

要启动 Linux，需要先将 Linux 镜像文件拷贝到 DRAM 中，如果使用到设备树的话也需要将设备树拷贝到 DRAM 中。可以从 EMMC 或者 NAND 等存储设备中将 Linux 镜像和设

备树文件拷贝到 DRAM，也可以通过 nfs 或者 tftp 将 Linux 镜像文件和设备树文件下载到 DRAM 中。不管用那种方法，只要能将 Linux 镜像和设备树文件存到 DRAM 中就行，然后使用 bootm 命令来启动，bootm 命令用于启动 uImage 镜像文件，bootm 命令格式如下：

```
bootm [addr [arg ...]]
```

命令 bootm 主要有三个参数，addr 是 Linux 镜像文件在 DRAM 中的位置，后面的“arg…”表示其他可选的参数，比如要指定 initrd 的话，第二个参数就是 initrd 在 DRAM 中的地址。如果 Linux 内核使用设备树的话还需要第三个参数，用来指定设备树在 DRAM 中的地址，如果不需要 initrd 的话第二个参数就用‘-’来代替。

现在我们使用网络和 EMMC 两种方法来启动 Linux 系统，首先将 STM32MP157 开发板的 Linux 镜像和设备树发送到 Ubuntu 主机中的 tftpboot 文件夹下。Linux 镜像文件前面已经放到了 tftpboot 文件夹中，现在把设备树文件放到 tftpboot 文件夹里面。以 EMMC 核心板为例，将开发板光盘→8、系统镜像→1、教程系统镜像→linux-mp1-5.4.31-g032f9dcc0-v1.0→stm32mp157d-atk.dtb 发送到 Ubuntu 主机中的 tftpboot 文件夹里面，完成以后的 tftpboot 文件夹如图 10.3.7.1 所示：

```
zuozhongkai@ubuntu:~/linux/tftpboot$ ls
stm32mp157d-atk.dtb  uImage
zuozhongkai@ubuntu:~/linux/tftpboot$
```

图 10.3.7.1 tftpboot 文件夹

现在 Linux 镜像文件和设备树都准备好了，我们先学习如何通过网络启动 Linux，使用 tftp 命令将 uImage 下载到 DRAM 的 0XC2000000 地址处，然后将设备树 stm32mp157d-atk.dtb 下载到 DRAM 中的 0XC400000 地址处，最后使用命令 bootm 启动，命令如下：

```
tftp c2000000 uImage
tftp c2000000 stm32mp157d-atk.dtb
bootm c2000000 - c4000000
```

命令运行结果如图 10.3.7.2 所示：

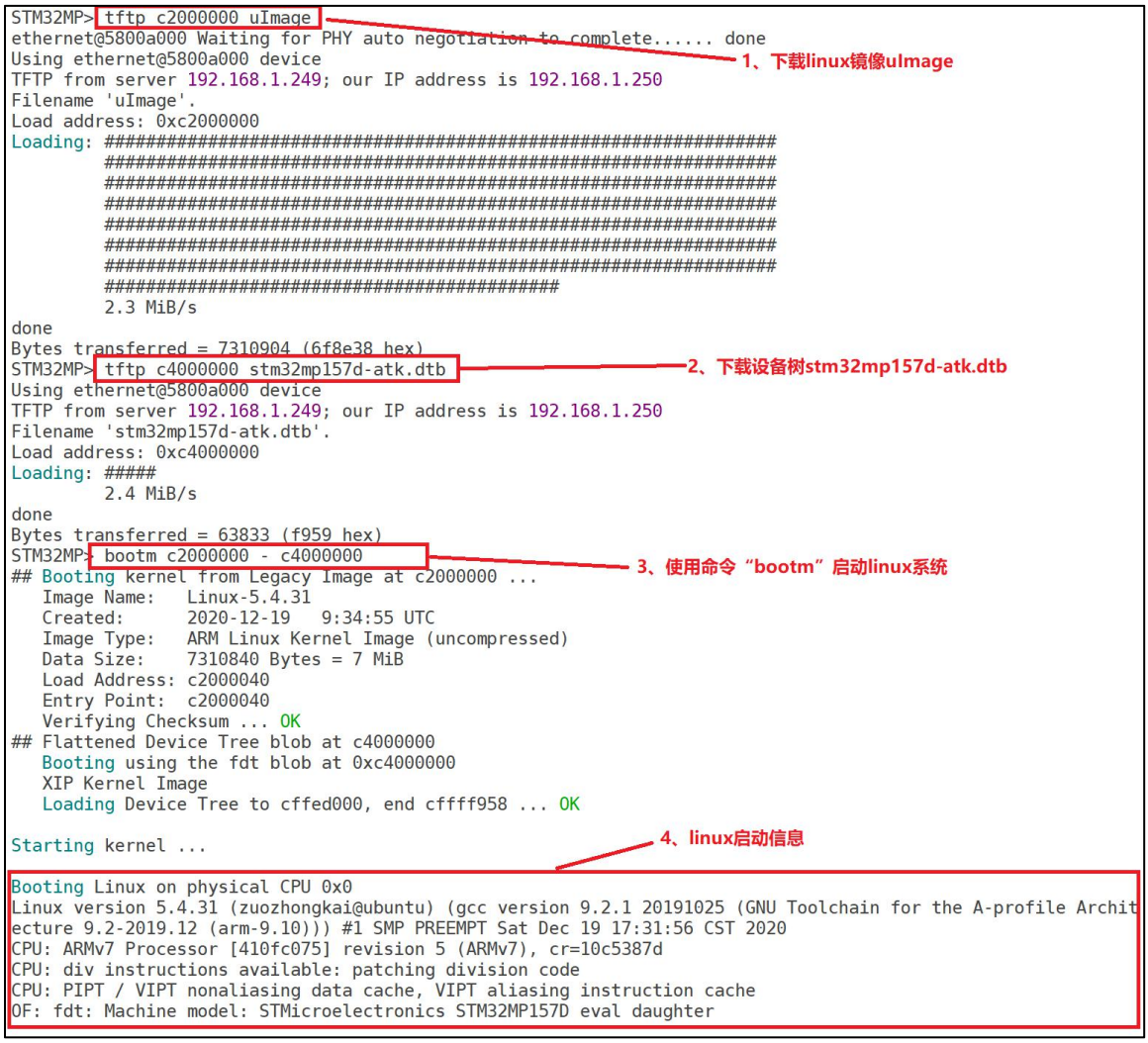

图 10.3.7.2 通过网络启动 Linux

从图 10.3.7.2 可以看出，此时 Linux 内核已经启动成功，相关的 log 信息已经打印出来了。

注意！只要打印出 Linux 内核启动信息就说明 Linux 系统启动成功，由于没有给定 bootargs 参数，所以 Linux 系统启动最终会失败，因为找不到根文件系统！

如果我们要从 EMMC 中启动 Linux 系统的话只需要使用命令 ext4load 将 uImage 和 stm32mp157d-atk.dtb 从 EMMC 的分区 2 中拷贝到 DRAM 中，然后使用命令 bootm 启动即可。先使用命令 ext4ls 查看要下 EMMC 的分区 2 中有没有 uImage 和 stm32mp157d-atk.dtb，如果没有的话请先使用 STM32CubeProgrammer 将正点原子出厂系统烧写到开发板中。

使用命令 ext4load 将 uImage 和 stm32mp157d-atk.dtb 文件拷贝到 DRAM 中，地址分别为 0XC2000000 和 0XC4000000，最后使用 bootm 启动，命令如下：

```
ext4load mmc 1:2 c2000000 uImage
ext4load mmc 1:2 c4000000 stm32mp157d-atk.dtb
bootm c2000000 - c4000000
```

命令运行结果如图 10.3.7.5 所示：

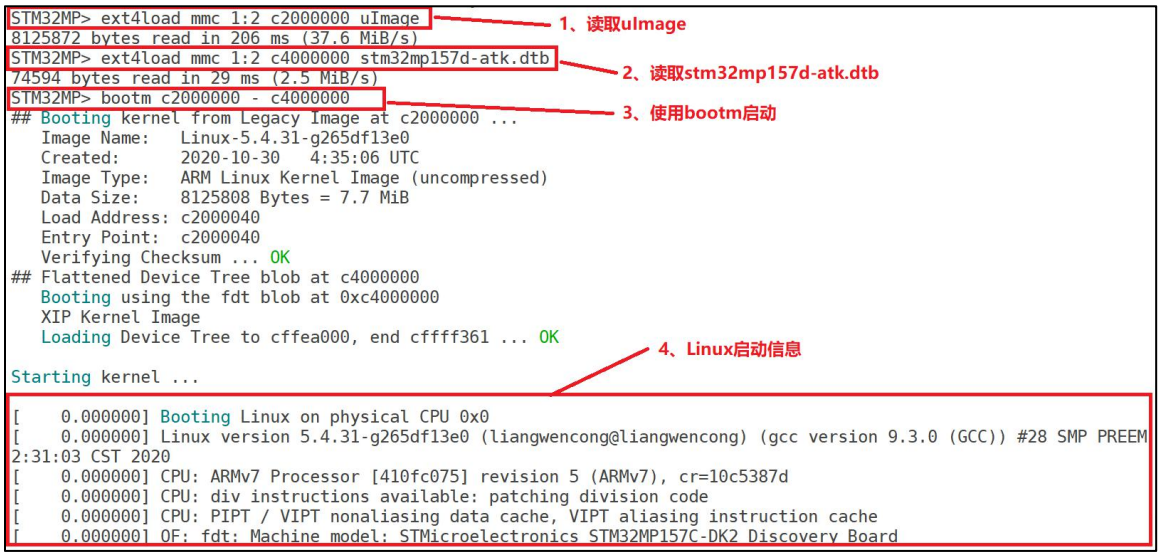

图 10.3.7.5 从 EMMC 中启动 Linux

2、bootz 命令

bootz 和 bootm 功能类似，但是 bootz 用于启动 zImage 镜像文件，bootz 命令格式如下：

```
bootz [addr [initrd[:size]] [fdt]]
```

命令 bootz 有三个参数，addr 是 Linux 镜像文件在 DRAM 中的位置，initrd 是 initrd 文件在 DRAM 中的地址，如果不使用 initrd 的话使用‘-’代替即可，fdt 就是设备树文件在 DRAM 中的地址，使用方法和 bootm 一模一样，只是所引导的 Linux 镜像格式不同，NXP 的 I.MX6ULL 就是使用 bootz 命令来引导 Linux 内核的。

3、boot 和 bootd 命令

注意！ST 官方 uboot 并没有使能 boot 和 bootd 这两个命令，需要自行配置 uboot 来启动这两个命令，正点原子出厂系统已经使能了这两个命令，后面 uboot 移植章节会给大家讲解如何使能这两个命令。

如果你使用别的厂商的开发板，在学习本小节的时候发现其 uboot 下没有使能 boot 和 bootd 命令，请联系对应的厂商，让其提供使能了 boot 和 bootd 命令的 uboot 内核和相应的 uboot 镜像文件，然后使用新的 uboot 镜像完成本实验。

boot 和 bootd 其实是一个命令，它们最终执行的是同一个函数。为了方便起见，后面就统一使用 boot 命令，此命令也是用来启动 Linux 系统的，只是 boot 会读取环境变量 bootcmd 来启动 Linux 系统，bootcmd 是一个很重要的环境变量！其名字分为“boot”和“cmd”，也就是“引导”和“命令”，说明这个环境变量保存着引导命令，其实就是多条启动命令的集合，具体的引导命令内容是可以修改的。比如我们要想使用 tftp 命令从网络启动 Linux 那么就可以设置 bootcmd 为“tftp c2000000 uImage;tftp c4000000 stm32mp157d-atk.dtb;bootm c2000000 - c4000000”，然后使用 saveenv 将 bootcmd 保存起来。然后直接输入 boot 命令即可从网络启动 Linux 系统，命令如下：

```
setenv bootcmd 'tftp c2000000 uImage;tftp c4000000 stm32mp157d-atk.dtb;bootm c2000000 -c4000000'
saveenv
boot
```

运行结果如图 10.3.7.6 所示：

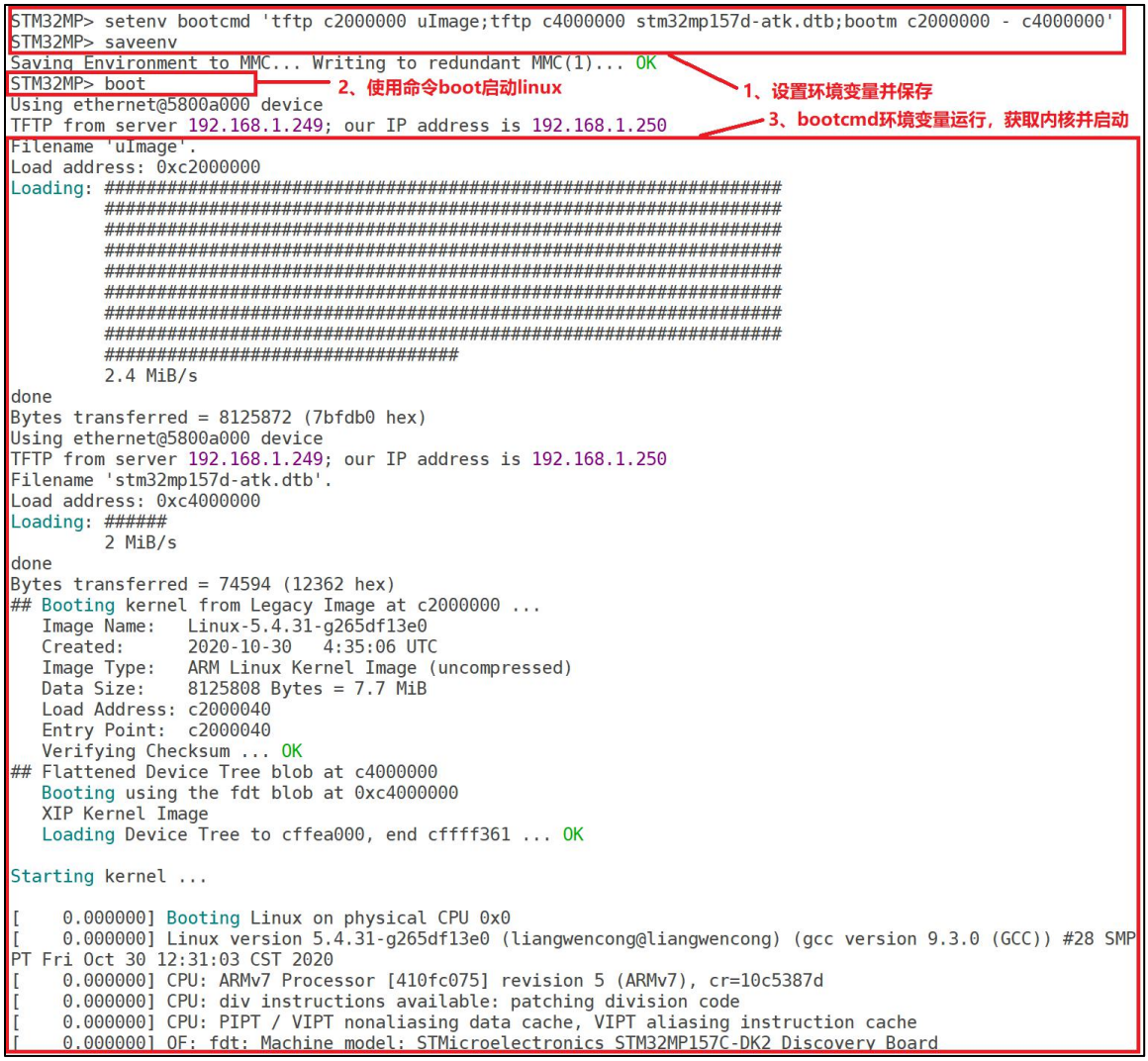

图 10.3.7.6 设置 bootcmd 从网络启动 Linux

前面说过 uboot 倒计时结束以后就会启动 Linux 系统，其实就是执行的 bootcmd 中的启动命令。只要不修改 bootcmd 中的内容，以后每次开机 uboot 倒计时结束以后都会使用 tftp 命令从网络下载 uImage 和 stm32mp157d-atk.dtb，然后启动 Linux 内核。

如果想从 EMMC 启动系统，那就设置 bootcmd 环境变量为“ext4load mmc 1:2 c2000000 uImage;ext4load mmc 1:2 c4000000 stm32mp157d-atk.dtb;bootm c2000000 - c4000000”，然后使用 boot 命令启动即可，命令如下：

```
setenv bootcmd 'ext4load mmc 1:2 c2000000 uImage;ext4load mmc 1:2 c4000000 stm32mp157d-atk.dtb;bootm c2000000 - c4000000'
saveenv
boot
```

运行结果如图 10.3.7.7 所示：

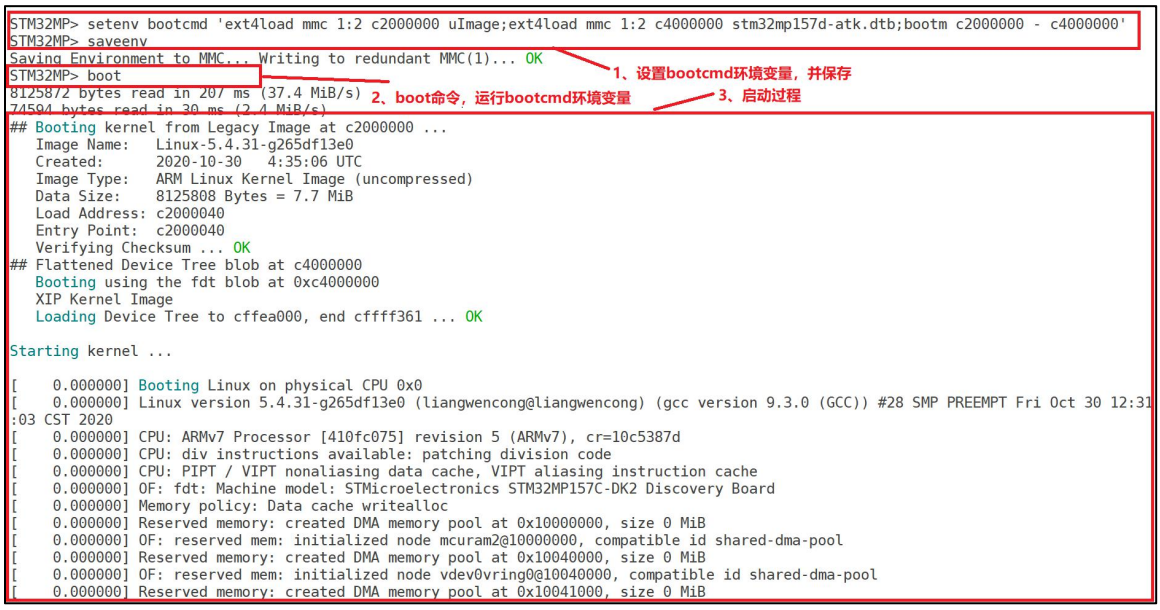

图 10.3.7.7 设置 bootcmd 从 EMMC 启动 Linux

如果不修改 bootcmd 的话，每次开机 uboot 倒计时结束以后都会自动从 EMMC 里面读取 uImage 和 stm32mp157d-atk.dtb，然后启动 Linux。

在启动 Linux 内核的时候可能会遇到如下错误：

“ ---[end Kernel panic - not syncing: VFS: Unable to mount root fs on unknown-block(0,0)]---”

如图 10.3.7.8 所示：

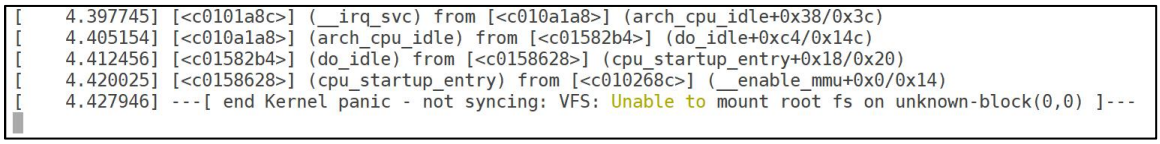

图 10.3.7.8 Linux 内核启动失败

这个错误的原因是 linux 内核没有找到根文件系统，这个很正常，因为没有在 bootargs 环境变量中指定根文件系统路径，关于 bootargs 环境变量后面会讲解！此处我们重点是验证 boot 命令，linux 内核已经成功启动了，说明 boot 命令工作正常。

10.3.8 UMS 命令

在 uboot 下我们可以将开发板虚拟成一个 U 盘，我们可以选择使用哪个 Flash 作为这个 U 盘的存储器，比如将正点原子 STM32MP157 开发板上的 EMMC 或者 SD 卡虚拟成 U 盘。当我们将 EMMC 虚拟成 U 盘以后就可以直接在电脑上向开发板拷贝文件了，比如我们在产品开发阶段，就可以直接在 uboot 下将某个文件拷贝到开发板的根文件系统中，这样就不需要进入系统或者通过网络来替换文件。

uboot 提供的 ums 命令就是来完成此功能的，ums 命令格式如下：

```
ums <USB_controller> [<devtype>] <dev[:part]>
```

其中 USB_controller 是 usb 接口索引，有的开发板有多个 USB SLAVE 接口，具体要使用哪个就可以通过 USB_controller 参数指定。正点原子 STM32MP157 开发板的只有一个 USB_OTG 口可以作为 USB SLAVE，对应的索引为 0。Devtype 是要挂载的设备，默认为 mmc，dev[:part]是要挂载的 Flash 设备，part 是要挂载的分区。

这里我们将开发板的 EMMC 挂载到电脑上，首先使用 USB Type-C 线将开发板的

USB_OTG 口与电脑连接起来，然后用以下命令启动。

```
ums 0 mmc 1
```

运行结果如图 10.3.8.1 所示：

```
STM32MP> ums 0 mmc 1
UMS: LUN 0, dev 1, hwpart 0, sector 0x0, count 0xe90000
\
```

图 10.3.8.1 ums 命令运行结果

挂载成功以后就会在电脑上有多个 U 盘，U 盘数量取决于你当前开发板 EMMC 分区数量，如果烧写过出厂系统的话默认是 3 个 U 盘，如图 10.3.8.2 所示：

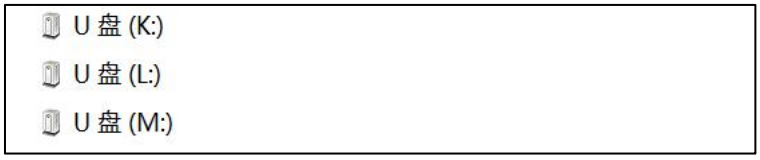

图 10.3.8.1 U 盘

注意，在 Windows 下这三个 U 盘是无法操作的，因为这三个 U 盘是 ext4 格式的，而 Windows 是不支持 ext4 格式！所以大家在操作的时候发现 Windows 报出 U 盘识别有问题，让格式化的，千万不要格式化！负责开发板整个 Linux 系统都会被格式化掉！

既然 Windows 识别不了 ext4 格式，那么我们可以将其挂载到 Ubuntu 下，这样就可以正常操作这三个 U 盘了。

如果要结束挂载，在终端下运行先按住 CTRL+C 键就能结束这个挂载。

10.3.9 其他常用命令

uboot 中还有其他一些常用的命令，比如 reset、go、run 和 mtest 等。

1、reset 命令

reset 命令顾名思义就是复位的，输入“reset”即可复位重启，如图 10.3.9.1 所示：

```
STM32MP> reset
resetting ...
INFO:    PSCI Power Domain Map:
INFO:      Domain Node : Level 1, parent_node -1, State ON (0x0)
INFO:      Domain Node : Level 0, parent_node 0, State ON (0x0)
INFO:      CPU Node : MPID 0x0, parent_node 0, State ON (0x0)
INFO:      CPU Node : MPID 0xffffffff, parent_node 0, State OFF (0x2)
NOTICE:  CPU: STM32MP157DAA Rev.Z
NOTICE:  Model: STMicroelectronics STM32MP157D eval daughter
INFO:    Reset reason (0x54):
INFO:      System reset generated by MPU (MPSYSRST)
INFO:    Using EMMC
INFO:      Instance 2
INFO:    Boot used partition fsbl1
NOTICE:  BL2: v2.2-r1.0(debug):463d4d8
NOTICE:  BL2: Built : 02:07:54, Oct 19 2020
INFO:    Using crypto library 'stm32_crypto_lib'
INFO:    BL2: Doing platform setup
INFO:    RAM: DDR3-DDR3L 32bits 533000Khz
```

reset命令

复位以后TF-A重新运行

图 10.3.9.1 reset 命令运行结果

2、go 命令

go 命令用于跳到指定的地址处执行应用，命令格式如下：

```
go addr [arg ...]
```

addr 是应用在 DRAM 中的首地址。

3、run 命令

run 命令用于运行环境变量中定义的命令，比如可以通过“run bootcmd”来运行 bootcmd 中的启动命令，但是 run 命令最大的作用在于运行我们自定义的环境变量。在后面调试 Linux 系统的时候常常要在网络启动和 EMMC 启动之间来回切换，而 bootcmd 只能保存一种启动方式，如果要换另外一种启动方式的话就得重写 bootcmd，会很麻烦。这里我们就可以通过自定义环境变量来实现不同的启动方式，比如定义环境变量 mybootemmc 表示从 emmc 启动，定义 mybootnet 表示从网络启动。如果要切换启动方式的话只需要运行“run mybootxxx(xxx 为 emmc 或 net)”即可。

说干就干，创建环境变量 mybootemmc 和 mybootnet，命令如下：

```
setenv mybootemmc 'ext4load mmc 1:2 c2000000 uImage;ext4load mmc 1:2 c4000000(有空格) stm32mp157d-atk.dtb;bootm c2000000 - c4000000'
setenv mybootnet 'tftp c2000000 uImage;tftp c4000000 stm32mp157d-atk.dtb;bootm c2000000 - (有空格) c4000000'
saveenv
```

创建环境变量成功以后就可以使用 run 命令来运行 mybootemmc、mybootnet 或 mybootnand 来实现不同的启动：

```
run mybootemmc
```

或

```
run mybootnet
```

4、mtest 命令

mtest 命令是一个简单的内存读写测试命令，可以用来测试自己开发板上的 DDR，命令格式如下：

```
mtest [start [end [pattern [iterations]]]]
```

start 是要测试的 DRAM 开始地址，end 是结束地址，比如我们测试 0XC0000000~0XC0001000 这段内存，输入“mtest C0000000 C0001000”，结果如图 10.3.9.2 所示：

```
  mtd mtdparts mtest
STM32MP> mtest c0000000 c0001000
Testing c0000000 ... c0001000:
Pattern 00000000  Writing..  Reading...Iteration:    527
```

图 10.3.9.2 mtest 命令运行结果

从图 10.3.9.2 可以看出，测试范围为 0XC0000000~0XC0001000，已经测试了 527 次，如果要结束测试就按下键盘上的“Ctrl+C”键。

至此，uboot 常用的命令就讲解完了，如果要使用 uboot 的其他命令，可以查看 uboot 中的帮助信息，或者上网查询一下相应的资料。