嵌入式

Administrator

[公司名称]

纯净版Linux系统

基于OrangePi Zero制作

前言

目录

[第一章 点亮开发板 1](#_Toc37256934)

[一、交叉编译工具链 1](#_Toc37256935)

[1.1 下载交叉编译工具链 1](#_Toc37256936)

[1.2 配置交叉编译工具到环境变量 1](#_Toc37256937)

[2、将交叉编译工具链的bin目录配置到宿主机的PATH环境变量 1](#_Toc37256938)

[二、BootLoader选用U-boot 1](#_Toc37256939)

[2.1 从官网下载u-boot源码 1](#_Toc37256940)

[2.2 编译U-Boot 2](#_Toc37256941)

[2.3 烧写U-Boot到TF卡并测试 2](#_Toc37256942)

[三、Linux内核 3](#_Toc37256943)

[3.1 从官网下载Linux内核源码 3](#_Toc37256944)

[3.2 编译Linux内核 3](#_Toc37256945)

[3.3测试Linux内核 3](#_Toc37256946)

[四、根文件系统选用Busybox制作 4](#_Toc37256947)

[4.1 从官网下载Busybox源码 4](#_Toc37256948)

[4.2 编译Busybox 4](#_Toc37256949)

# 第一章 点亮开发板

## 一、交叉编译工具链

### 1.1 下载交叉编译工具链

使用Linaro（<https://www.linaro.org/>）提供的arm-linux-gnueabi-hf-工具链，下载地址：

<https://releases.linaro.org/components/toolchain/binaries/>

这里使用的是7.4版本，由于宿主机使用的是64位Linux，所以，选用x86\_64的，具体下载链接如下：

<https://releases.linaro.org/components/toolchain/binaries/7.4-2019.02/arm-linux-gnueabihf/gcc-linaro-7.4.1-2019.02-x86_64_arm-linux-gnueabihf.tar.xz>

其他版本也可以自行尝试。

### 1.2 配置交叉编译工具到环境变量

#### 1、选择合适的地方放置下载的压缩包，并且解压

解压命令 tar –xvJf gcc-linaro-7.4.1-2019.02-x86\_64\_arm-linux-gnueabihf.tar.xz

### 2、将交叉编译工具链的bin目录配置到宿主机的PATH环境变量

也就是这个目录gcc-linaro-7.4.1-2019.02-i686\_arm-linux-gnueabihf/bin

配置环境变量的位置有几个：

/etc/profile（这个是全局的，不建议使用）

/etc/rc.local（这个也是全局的，并且执行的更早，不建议使用）

~/.profile（这个是当前用户的，建议使用）

或者临时配置，直接执行：

export PATH=”/xxx/gcc-linaro-7…\_arm-linux-gnueabihf/bin:$PATH”

回车即可，但是每次重新登录终端，都要配置，不建议使用。

## 二、BootLoader选用U-boot

### 2.1 从官网下载u-boot源码

官网地址： <http://www.denx.de/> （可能需要科学上网）

建议到FTP站点上去下，并且选择2019.10（含）以前的版本，更新的版本不能直接支持OrangePi Zero的Linux内核启动，具体地址如下：

<ftp://ftp.denx.de/pub/u-boot/>

这里选择u-boot-2019.10.tar.bz2作为实验，选择一个地方，下载U-Boot源码：

wget 'ftp://ftp.denx.de/pub/u-boot/u-boot-201 9.10.tar.bz2'

解压：

tar -xvjf u-boot-2019.10.tar.bz2

### 2.2 编译U-Boot

直接编译：

make orangepi\_zero\_defconfig

make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabihf- -j32 V=s

编译成功后，会在u-boot源码的根目录生成u-boot-sunxi-with-spl.bin文件

否则，请检查编译环境，此版本的源码本身没有问题，可以编译成功。

### 2.3 烧写U-Boot到TF卡并测试

#### 1、准备TF卡

超过256MB就好了。用读卡器把TF卡插入到Linux主机上，假设Linux读到的设备是/dev/sdb

#### 2、uboot刷到TF卡8192字节处

dd if=u-boot-sunxi-with-spl.bin of=/dev/sdb bs=1024 seek=8

得到类似以下结果即可插卡到开发板上测试了：

记录了441+1 的读入

记录了441+1 的写出

452024 bytes (452 kB, 441 KiB) copied, 0.361404 s, 1.3 MB/s

通过串口看开发板的输出，得到类似以下的结果，即说明U-Boot已经成功的运行了：

U-Boot SPL 2019.10 (Apr 07 2020 - 15:23:00 +0800)

DRAM: 512 MiB

Trying to boot from MMC1

U-Boot 2019.10 (Apr 07 2020 - 15:23:00 +0800) Allwinner Technology

CPU: Allwinner H3 (SUN8I 1680)

Model: Xunlong Orange Pi Zero

DRAM: 512 MiB

----- 略 -----

## 三、Linux内核

### 3.1 从官网下载Linux内核源码

官网地址：

<https://www.kernel.org/> （可能需要科学上网）

这里选择linux-5.6.2.tar.xz作为实验，选择一个地方，下载U-Boot源码：

wget 'https://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.6.2.tar.xz'

解压：

tar -xvJf linux-5.6.2.tar.xz

### 3.2 编译Linux内核

直接编译：

make ARCH=arm sunxi\_defconfig

make ARCH=arm CROSS\_COMPILE=arm-linux-gnueabihf- -j32 V=s

编译成功后，会生成两个目标文件：

zImage (arch/arm/boot/zImage)

sun8i-h2-plus-orangepi-zero.dtb (arch/arm/boot/dts/sun8i-h2-plus-orangepi-zero.dtb)

否则，请检查编译环境，此版本的源码本身没有问题，可以编译成功。

### 3.3测试Linux内核

#### 1、准备TF卡

保证TF卡的第一个分区为FAT文件系统，可以格式化，不影响之前刷写的U-Boot。

然后将编译出来的两个文件（zImage和sun8i-h2-plus-orangepi-zero.dtb）复制到TF卡的第一个FAT分区中。

将TF卡插入开发板，接串口，上电，等待箭头符号出现（就是这个 =>）。

#### 2、启动Linux内核

以下操作均属于操作U-Boot，暂时使用手动操作，后续章节搞自动启动。

1. 设置启动参数：

setenv bootargs 'console=ttyS0,115200n8 earlyprintk'

1. 从TF卡的第一个FAT分区读入zImage到内存的0x46000000地址处

fatload mmc 0 0x46000000 zImage

输出类似以下内容则表示读取成功，否则请检查TF卡分区的文件系统，必须是FAT，其他格式（如exFAT）不可以，如果TF卡太大，格式化的时候没有FAT选项，可以进行分区，分出来一个比较小的区（8MB左右），然后进行格式化处理即可。

4364560 bytes read in 406 ms (10.3 MiB/s)

1. 从TF卡的第一个FAT分区读入设备树到内存的0x49000000地址处，

此地址根据zImage的大小进行推算，保证设备树与内核代码不会重合，并且不要偏移太多，以免超出内存范围。

fatload mmc 0 0x49000000 sun8i-h2-plus-orangepi-zero.dtb

输出类似以下内容则表示读取成功，否则请检查TF卡分区的文件系统。

21734 bytes read in 4 ms (5.2 MiB/s)

1. 启动Linux内核，第一个地址为zImage起始地址，第二个为设备树起始地址

bootz 0x46000000 - 0x49000000

输出类似以下内容则表示Linux内核已启动

## Flattened Device Tree blob at 49000000

Booting using the fdt blob at 0x49000000

EHCI failed to shut down host controller.

Loading Device Tree to 49ff7000, end 49fff4e5 ... OK

Starting kernel ...

[ 0.000000] Booting Linux on physical CPU 0x0

[ 0.000000] Linux version 5.6.2

----- 略 -----

## 四、根文件系统选用Busybox制作

### 4.1 从官网下载Busybox源码

官网地址：

<https://busybox.net/>

这里选择busybox-1.31.1.tar.bz2作为实验，选择一个地方，下载U-Boot源码：

wget 'https://busybox.net/downloads/busybox-1.31.1.tar.bz2'

解压：

tar -xvjf busybox-1.31.1.tar.bz2

### 4.2 编译Busybox

#### 1、配置Busybox

执行命令：

make menuconfig

#### 2、设置静态编译、交叉编译工具链名称、安装路径

下面是具体配置位置，更早的版本可能没有Settings这一层。

Settings --->

[\*] Build static binary (no shared libs)

(arm-linux-gnueabihf-) Cross compiler prefix

(./\_install) Destination path for 'make install' (NEW)

#### 3、编译并安装Busybox

A、执行编译命令：

make –j32

make install

编译成功后，会在Busybox源码根目录生成名为 \_install的文件夹，并且 \_install/bin/中有busybox可执行文件，以及其他很多指向busybox的软连接。

若没有，则检查编译环境，此版本代码经过验证没有问题。

B、给TF卡分区：

确保TF卡第一个分区为FAT文件系统，并且容量可以存下Linux内核和设备树；

TF卡第二个分区为ext4文件系统；

分区工具可以使用Linux下的fdisk命令。

C、部署跟文件系统：

将编译好的内容（\_install文件夹中的内容）复制到TF卡的ext4分区中即可。

#### 4、启动测试跟文件系统

A、将TF卡插入开发板，接串口，上电；

B、按照“启动Linux内核”的步骤进行操作，但是需要将设置启动参数改为：

setenv bootargs 'console=ttyS0,115200n8 earlyprintk root=/dev/mmcblk0p2 rootwait

C、操作之后，串口显示类似以下信息，说明启动成功：

[ 1.754985] EXT4-fs (mmcblk0p2): mounted filesystem with ordered data mode. Opts: (null)

[ 1.763172] VFS: Mounted root (ext4 filesystem) readonly on device 179:2.

[ 1.773492] devtmpfs: error mounting -2

[ 1.778433] Freeing unused kernel memory: 1024K

[ 1.783174] Run /sbin/init as init process

[ 1.790196] mmc1: new high speed SDIO card at address 0001

can't run '/etc/init.d/rcS': No such file or directory

can't open /dev/tty2: No such file or directory

can't open /dev/tty3: No such file or directory

can't open /dev/tty4: No such file or directory

can't open /dev/tty2: No such file or directory

# 第二章 让系统自动启动

## 一、固化U-Boot启动参数

### 1.1 配置启动参数与启动命令

在U-Boot源码根目录执行命令（进入配置菜单）：

make orangepi\_zero\_defconfig

make ARCH=arm menuconfig

以下操作都在此菜单内完成。

#### 1、配置启动参数（bootargs）：

[\*] Enable boot arguments

() Boot arguments

在() Boot arguments中填入：

console=ttyS0,115200n8 earlyprintk root=/dev/mmcblk0p2 rootwait panic=10 no\_console\_suspend quiet

注释：

* console=ttyS0,115200n8

设置控制台参数，使用串口0（ttyS0）作为控制台，波特率115200，8个数据位，1个停止位

* earlyprintk

允许Linux内核在启动初期就输出打印信息（不是启动必须的，但有助于调试）

* root=/dev/mmcblk0p2

设置根文件系统的位置，在mmc0（TF卡）的第2个分区（数字从1开始）。起初TF卡分两个区，第一个分区为FAT，放置内核跟设备树，第二个分区就是根文件系统的分区

* rootwait

等待根文件系统所在的设备初始化完成后再挂载根文件系统。若不使用此参数，当TF卡还没初始化完成，就去挂载根文件系统，则会报错

* panic=10

当出现kernel panic，等待10秒后自动重启系统（不是启动必须的）

* no\_console\_suspend

加此选项后，当系统进入休眠时，控制台依然会输出信息（不是启动必须的，调试低功耗时使用）

* quiet

减少启动时的打印信息，有助于加速启动（不是启动必须的）

备注：

详细的U-Boot操作文档请点击

<http://www.denx.de/wiki/DULG/Manual>

#### 2、配置启动命令（bootcmd）

[\*] Enable a default value for bootcmd

(run distro\_bootcmd) bootcmd value

将括号中的run distro\_bootcmd修改为下面内容（实际输入时不要换行，下面前两条末尾有分号，最后一条不需要加分号）：

fatload mmc 0 0x40000000 zImage;

fatload mmc 0 0x45000000 sun8i-h2-plus-orangepi-zero.dtb;

bootz 0x40000000 - 0x45000000

注释：

* fatload mmc 0 0x40000000 zImage;

从FAT分区装载装载源设备为mmc第0个分区装到内存的这个地址处被装载的文件名

连起来解释为：从mmc（TF卡）的第0个FAT分区读取zImage文件，并从内存的0x40000000地址处开始放置此文件的内容，最后的分号表示本条指令结束，与下一条指令分开

* fatload mmc 0 0x45000000 sun8i-h2-plus-orangepi-zero.dtb;

意思同上，上条命令为加载Linux内核，本条命令加载设备树

* bootz 0x40000000 - 0x45000000

启动 内核地址 分隔符 设备树地址

本条命令会设置CPU的PC指针，完成跳转动作，执行后，会启动内核。

备注：

1. fatload命令仅用于FAT文件系统，其他文件系统请参考官方文档：<http://www.denx.de/wiki/DULG/Manual>
2. mmc指的是TF卡、SD卡、eMMC等存储器，与NandFlash、NorFlash使用的命令不同
3. 内存地址的来历：

* 根据Allwinner\_H3\_Datasheet\_V1.0.pdf中的85页靠下的位置可知，该SOC的内存地址是从0x40000000开始的，如截图所示：



* 根据orange\_pi-zero-v1\_11.pdf中的第5页上方位置可知，开发板上只有一块内存芯片，内存芯片的片选信号（CS#）接在主控的SCS0引脚上，内存芯片的CS1#引脚接在主控的SCS0引脚上。并且地址线没有空位，可知RAM的地址没有偏移，直接从主控规定的RAM起始地址开始。又因为此内存芯片为512MB，则此开发板的RAM地址空间范围是0x40000000 ~ 0x60000000
* 根据U-Boot的配置菜单（menuconfig）中

Boot images --->

(0x4a000000) Text Base

可知，U-Boot的运行地址是从0x4a000000开始的，大约450KB，所以，内核地址、设备树地址只要在开发板的RAM空间内，并且与U-Boot所在的空间不重合即可。

1. 开发板启动顺序：

由Allwinner\_H3\_Datasheet\_V1.0.pdf中的56、86页可知，主控芯片上电后执行芯片内部BootROM中的启动程序，然后启动程序去TF卡中执行U-Boot开头一部分代码，随后，U-Boot开头的一段代码将U-Boot自身完整的代码搬运到RAM中的0x4a000000位置继续执行。接着根据Bootcmd命令与Bootargs内容去TF卡中加载并执行Linux内核。