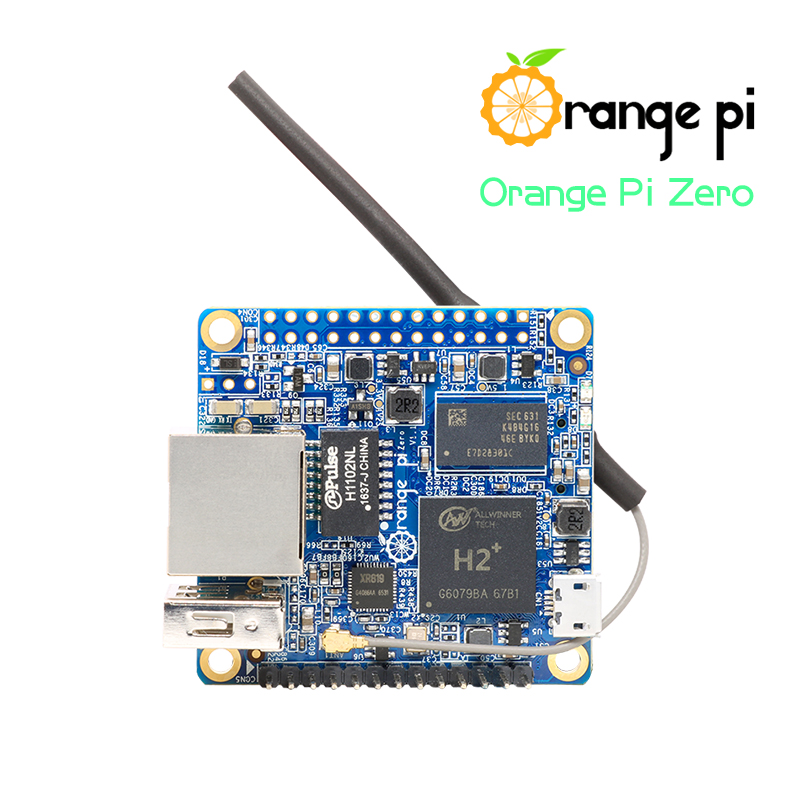
嵌入式

于耀文 著

未经允许请勿传播



纯净版Linux系统

基于OrangePi Zero制作

**前言**

本书仅作为实践参考使用，对于技术细节不作讲解，需要读者有一定的嵌入式Linux基础，比如：设备驱动模型、设备树、Linux系统组成等，不适合初学者。

书中尽量使用官方开源代码，尽量不删、改官方源代码，仅对设备树进行必要的“添加”操作。所使用的工具、功能库等也是开源代码，除了制作镜像需要重复多次没有营养的操作写成的一个脚本以外，没有手写任何工具，大部分内容是在配置内核选项、为设备树添加设备节点。

写下本书目的是为了抛砖引玉，启示更多的读者：开源项目已经为我们做了很多事情，不要再做重复的工作。

目录

[第一章 点亮开发板 1](#_Toc39935280)

[一、交叉编译工具链 1](#_Toc39935281)

[1.1 下载交叉编译工具链 1](#_Toc39935282)

[1.2 配置交叉编译工具到环境变量 1](#_Toc39935283)

[二、BootLoader选用U-boot 1](#_Toc39935284)

[2.1 从官网下载u-boot源码 1](#_Toc39935285)

[2.2 编译U-Boot 2](#_Toc39935286)

[2.3 烧写U-Boot到TF卡并测试 2](#_Toc39935287)

[三、Linux内核 3](#_Toc39935288)

[3.1 从官网下载Linux内核源码 3](#_Toc39935289)

[3.2 编译Linux内核 3](#_Toc39935290)

[3.3测试Linux内核 3](#_Toc39935291)

[四、根文件系统选用Busybox制作 4](#_Toc39935292)

[4.1 从官网下载Busybox源码 4](#_Toc39935293)

[4.2 编译Busybox 4](#_Toc39935294)

[第二章 让系统自动启动 6](#_Toc39935295)

[一、固化U-Boot启动参数 6](#_Toc39935296)

[1.1 配置启动参数与启动命令 6](#_Toc39935297)

[1.2 烧写TF卡并测试 8](#_Toc39935298)

[二、优化U-Boot启动速度 8](#_Toc39935299)

[2.1 关闭U-Boot的USB支持 8](#_Toc39935300)

[2.2 去掉自动启动前的等待 8](#_Toc39935301)

[三、完善根文件系统的基本内容 9](#_Toc39935302)

[3.1 创建必要的以及常用的文件夹 9](#_Toc39935303)

[3.2 创建/etc/inittab文件 9](#_Toc39935304)

[3.3 创建/etc/fstab文件 9](#_Toc39935305)

[3.4 创建/etc/init.d/rcS文件 9](#_Toc39935306)

[四、制作镜像 10](#_Toc39935307)

[4.1 制作固定大小的镜像 10](#_Toc39935308)

[4.2 写入TF卡测试镜像 10](#_Toc39935309)

[4.3 整理制作镜像的命令为脚本文件 11](#_Toc39935310)

[五、调整根分区大小填充整个TF卡 12](#_Toc39935311)

[5.1 使用fdisk命令修改根分区大小 12](#_Toc39935312)

[5.2 使用脚本自动调整分区大小 13](#_Toc39935313)

[六、自动启动有线网卡 14](#_Toc39935314)

[6.1 开机启动if接口 14](#_Toc39935315)

[6.2 自动获取ip地址 14](#_Toc39935316)

[七、自动校准系统时间 14](#_Toc39935317)

[7.1 使用ntpd命令自动同步网络时间 14](#_Toc39935318)

[第三章 点亮板载外设 15](#_Toc39935319)

[一、两个LED指示灯 15](#_Toc39935320)

[1.1 配置LED驱动 15](#_Toc39935321)

[1.2 配置LED设备 15](#_Toc39935322)

[二、去掉/dev/pty\* 16](#_Toc39935323)

[三、加入SPI-Flash驱动 16](#_Toc39935324)

[3.1 设备树中加入节点 16](#_Toc39935325)

[3.2 配置驱动 17](#_Toc39935326)

[3.3 验证SPI-Nor-Flash 17](#_Toc39935327)

[四、加入WiFi驱动 17](#_Toc39935328)

[4.1 下载驱动源码 17](#_Toc39935329)

[4.2 配置驱动 18](#_Toc39935330)

[4.3 连接WLAN 21](#_Toc39935331)

[五、加入声卡驱动 22](#_Toc39935332)

[5.1 设备树中修改节点 22](#_Toc39935333)

[5.2 配置驱动 23](#_Toc39935334)

[5.3 播放声音 23](#_Toc39935335)

[六、加入外部LCD驱动（SPI-ILI9488） 24](#_Toc39935336)

[6.1 寻找驱动源码，根据Datasheet修改参数 24](#_Toc39935337)

[6.2 配置设备树 24](#_Toc39935338)

[6.3 背光控制 25](#_Toc39935339)

[6.4 触摸屏 25](#_Toc39935340)

[七、开发板模拟优盘 26](#_Toc39935341)

[7.1 配置驱动 26](#_Toc39935342)

[7.2 加载驱动 26](#_Toc39935343)

[八、添加UVC摄像头驱动 27](#_Toc39935344)

[九、添加U盘、移动硬盘支持 27](#_Toc39935345)

# 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。

### 1.2 烧写TF卡并测试

烧写TF卡的方法同第一章，给开发板上电时，不要手动输入任何内容，观察串口输出即可。

当串口输出类似以下内容则代表成功：

Hit any key to stop autoboot: 0

4364560 bytes read in 196 ms (21.2 MiB/s)

21734 bytes read in 3 ms (6.9 MiB/s)

## Flattened Device Tree blob at 45000000

Booting using the fdt blob at 0x45000000

EHCI failed to shut down host controller.

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

Starting kernel ...

[ 0.001377] /cpus/cpu@0 missing clock-frequency property

[ 0.001401] /cpus/cpu@1 missing clock-frequency property

[ 0.001417] /cpus/cpu@2 missing clock-frequency property

[ 0.001432] /cpus/cpu@3 missing clock-frequency property

## 二、优化U-Boot启动速度

### 2.1 关闭U-Boot的USB支持

执行命令make ARCH=arm menuconfig

找到以下路径：

Device Drivers --->

-\*- USB support --->

去掉该路径内的所有选项。

### 2.2 去掉自动启动前的等待

执行命令make ARCH=arm menuconfig

找到：(2) delay in seconds before automatically booting

将括号内的2改为0

## 三、完善根文件系统的基本内容

### 3.1 创建必要的以及常用的文件夹

以编译好的Busybox中的\_install为基础，创建这些文件夹：

bin boot dev etc lib linuxrc mnt proc root run sbin share sys tmp usr var

在etc中创建init.d文件夹

### 3.2 创建/etc/inittab文件

内容如下：

# /etc/inittab

::sysinit:/etc/init.d/rcS

ttyS0::sysinit:/bin/ash

::ctrlaltdel:/sbin/reboot

::shutdown:/bin/umount -a -r

注：此文件用来控制文件系统挂载后，系统的一些动作，例如，第一个执行的脚本为rcS、终端默认使用ttyS0、关机命令发出时执行umount -a –r等

### 3.3 创建/etc/fstab文件

内容如下：

# device mount-point type options dump fsck order

proc /proc proc defaults 0 0

tmpfs /tmp tmpfs defaults 0 0

sysfs /sys sysfs defaults 0 0

debugfs /sys/kernel/debug debugfs defaults 0 0

注：此文件用来指导文件系统的挂载动作

### 3.4 创建/etc/init.d/rcS文件

内容如下：

#! /bin/sh

mount -a

mount -o remount,rw /

mount /dev/mmcblk0p1 /boot

dmesg -n 1

注：此文件为挂载文件系统完成后第一个被执行的脚本，必须有x权限

## 四、制作镜像

### 4.1 制作固定大小的镜像

* 创建一个128MB空文件：OrangePiZero.img

dd if=/dev/zero of= OrangePiZero.img bs=1M count=128

* 将空文件看做磁盘，进行分区，一个8MB的FAT分区，剩下的是EXT4分区

echo 'n\np\n1\n2048\n18432\nt\nb\nn\np\n2\n18433\n\nw\n' |fdisk OrangePiZero.img

* 将镜像文件映射到系统的loop设备上，大约需要等待100ms左右

sudo kpartx -av OrangePiZero.img

* 分别格式化2个分区

sudo mkfs.vfat /dev/mapper/loop0p1

sudo mkfs.ext4 /dev/mapper/loop0p2

* 分别设置两个分区的卷标（非必须，卷名随意起）

sudo dosfslabel /dev/mapper/loop0p1 "BOOT"

sudo e2label /dev/mapper/loop0p2 "Linux"

* 在系统上创建临时挂载点（位置和名称随意），用于挂载镜像的两个分区

sudo mkdir -p /mnt/BOOT

sudo mkdir -p /mnt/Linux

* 分别挂载两个分区

sudo mount -o loop -t vfat /dev/mapper/loop0p1 /mnt/BOOT

sudo mount -o loop -t ext4 /dev/mapper/loop0p2 /mnt/Linux

* 将准备好的文件放入对应的分区：zImage和dtb放到FAT分区，根文件系统放到EXT4分区

sudo cp -v zImage sun8i-h2-plus-orangepi-zero.dtb /mnt/BOOT

sudo cp -v -r rootfs/\* /mnt/Linux/

sync # 同步一下磁盘，保证物理写入

* 卸载分区

sudo umount -l /mnt/BOOT

sudo umount -l /mnt/Linux

* 删除临时挂载点

sudo rm -rf /mnt/BOOT

sudo rm -rf /mnt/Linux

* 将U-Boot写入镜像文件

sudo dd if=u-boot-sunxi-with-spl.bin of=/dev/loop0 bs=1024 seek=8

* 取消镜像文件在系统上映射的loop设备

sudo kpartx -d OrangePiZero.img

### 4.2 写入TF卡测试镜像

先使用umount卸载已挂载的TF卡分区，假设TF卡是sdb

sudo dd if=OrangePiZero.img of=/dev/sdb bs=4M

执行完上面命令后，即可测试是否成功。

### 4.3 整理制作镜像的命令为脚本文件

#! /bin/sh

IMG\_NAME="OrangePiZero.img"

# Create a new disk image: 128MB

dd if=/dev/zero of=$IMG\_NAME bs=1M count=128

# Partition the image, part1: 8MB@FAT32, part2:@EXT4

echo 'n\np\n1\n2048\n18432\nt\nb\nn\np\n2\n18433\n\nw\n' |fdisk $IMG\_NAME > /dev/null

fdisk -l $IMG\_NAME |tail -n 3

# Mapping the image to host system

sudo kpartx -av $IMG\_NAME

[ $? -ne 0 ] && echo "Please use right root password." && exit 1

sleep 0.1s

# Format the partion: part1@FAT32

sudo mkfs.vfat /dev/mapper/loop0p1

# Format the partion: part2@EXT4

sudo mkfs.ext4 /dev/mapper/loop0p2

sleep 0.1s

# Change the label of partion: part1@BOOT

sudo dosfslabel /dev/mapper/loop0p1 "BOOT"

# Change the label of partion: part2@Linux

sudo e2label /dev/mapper/loop0p2 "Linux"

# Create mountpoints

sudo mkdir -p /mnt/BOOT

sudo mkdir -p /mnt/Linux

# Mount partions

sudo mount -o loop -t vfat /dev/mapper/loop0p1 /mnt/BOOT

sudo mount -o loop -t ext4 /dev/mapper/loop0p2 /mnt/Linux

# Copy files to image partions

sudo cp -v zImage sun8i-h2-plus-orangepi-zero.dtb /mnt/BOOT

sudo cp -v -r rootfs/\* /mnt/Linux/

sync

# Unmount partions

sudo umount -l /mnt/BOOT

sudo umount -l /mnt/Linux

# Delete mountpoints

sudo rm -rf /mnt/BOOT

sudo rm -rf /mnt/Linux

# Install u-boot to image

sudo dd if=u-boot-sunxi-with-spl.bin of=/dev/loop0 bs=1024 seek=8

# Unmap the image from host system

sudo kpartx -d $IMG\_NAME

[ $? -ne 0 ] && echo "Please use right root password." && exit 1

## 五、调整根分区大小填充整个TF卡

### 5.1 使用fdisk命令修改根分区大小

#### 1、查看当前TF卡分区情况

# fdisk –l

Disk /dev/mmcblk0: 125 GB, 134217728000 bytes, 262144000 sectors

4096000 cylinders, 4 heads, 16 sectors/track

Units: sectors of 1 \* 512 = 512 bytes

Device Boot StartCHS EndCHS StartLBA EndLBA Sectors Size Id Type

/dev/mmcblk0p1 0,32,33 1,37,37 2048 18432 16385 8192K b Win95 FAT32

/dev/mmcblk0p2 1,37,38 16,81,1 18433 262143 243711 118M 83 Linux

由上述输出可以看出，第二个分区只有118MB，TF卡总大小为125GB。只需要将第二个分区的范围默认为TF卡剩余的空间即可。记住第二个分区的StartLBA为18433即可。

#### 2、重新分区

# fdisk /dev/mmcblk0

依次输入下列命令：

d<回车>

2<回车>

n<回车>

p<回车>

2<回车>

18433<回车>

<回车>

w<回车>

注：18433后面是两个回车

命令执行完毕后，再使用fdisk –l命令查看分区情况：

Disk /dev/mmcblk0: 125 GB, 134217728000 bytes, 262144000 sectors

4096000 cylinders, 4 heads, 16 sectors/track

Units: sectors of 1 \* 512 = 512 bytes

Device Boot StartCHS EndCHS StartLBA EndLBA Sectors Size Id Type

/dev/mmcblk0p1 0,32,33 1,37,37 2048 18432 16385 8192K b Win95 FAT32

/dev/mmcblk0p2 288,0,2 1023,3,16 18433 262143999 262125567 124G 83 Linux

#### 3、重新读入分区表

使用fdisk分区后，并不能直接生效，需要**重启**系统**后**使用resize2fs命令才能生效。

# resize2fs /dev/mmcblk0p2

但是，Busybox内没有resize2fs命令需要移植。

* 移植resize2fs命令所在的e2fsprogs工具包，在kernel.org网站上有源码：

<https://mirrors.edge.kernel.org/pub/linux/kernel/people/tytso/e2fsprogs/>

这里使用新版本1.45.6：

<https://mirrors.edge.kernel.org/pub/linux/kernel/people/tytso/e2fsprogs/v1.45.6/e2fsprogs-1.45.6.tar.xz>

* 下载并解压，然后进入文件夹进行配置：

./configure --prefix=${PWD}/\_install --host=arm-linux-gnueabihf

make -j32

make install

* 将生成的\_install/sbin/resize2fs复制到镜像中根文件系统/sbin即可

### 5.2 使用脚本自动调整分区大小

#### 1、编写脚本resize.sh放在镜像根文件系统/root/位置

此脚本完成fdisk分区操作，部署下次开机启动命令，完成重启工作。

#! /bin/sh

cat << EOF > /tmp/partargs.txt

d

2

n

p

2

18433

w

EOF

#echo 'd\n2\nn\np\n2\n18433\n\nw\n' |fdisk /dev/mmcblk0 > /dev/null

fdisk /dev/mmcblk0 < /tmp/partargs.txt

sed -i '/#Remove this line/d' /etc/init.d/rcS

sync

echo "resize2fs /dev/mmcblk0p2 #Remove this line" >> /etc/init.d/rcS

echo "rm $0 #Remove this line" >> /etc/init.d/rcS

echo "sed -i '/#Remove this line/d' /etc/init.d/rcS" >> /etc/init.d/rcS

sync

reboot -f

#### 2、修改/etc/init.d/rcS文件

在末尾加入

/root/resize.sh #Remove this line

rm /root/resize.sh #Remove this line

## 六、自动启动有线网卡

### 6.1 开机启动if接口

编辑镜像根文件系统中的/etc/init.d/rcS文件，追加命令：

ifconfig eth0 up

### 6.2 自动获取ip地址

#### 1、使用dhcpc命令自动获取并设置ip地址

添加开机启动命令，在/etc/init.d/rcS文件追加命令：

udhcpc &

注意：此命令应该在启用if口之后。

#### 2、配置udhcpc

从Busybox源码目录中找到busybox-1.31.1/examples/udhcp/simple.script，

复制到镜像文件系统中的/usr/share/udhcpc/中，

改名为default.script

## 七、自动校准系统时间

### 7.1 使用ntpd命令自动同步网络时间

#### 1、配置开机启动进程

在/etc/init.d/rcS文件追加命令：

ntpd

注意：此命令应该在自动获取ip地址之后执行

#### 2、编辑配置文件

在/etc/ntp.conf 文件中写入：

server cn.pool.ntp.org prefer

#### 3、设置时区

复制PC机的/usr/share/zoneinfo/Asia/Shanghai，

放到/etc/

重命名为localtime

# 第三章 点亮板载外设

## 一、两个LED指示灯

### 1.1 配置LED驱动

板载LED是直接接在GPIO引脚上的，所以Linux内核应该配置“LEDS\_GPIO”

Device Drivers --->

[\*] LED Support --->

<\*> LED Class Support

<\*> LED Support for GPIO connected LEDs

[\*] LED Trigger support --->

（可以全选）

### 1.2 配置LED设备

由原开发板电路理图可知，两个LED分别接在PA17、PL10引脚上。

修改设备树linux-5.6.2/arch/arm/boot/dts/ sun8i-h2-plus-orangepi-zero.dts内容：

注：

leds {

compatible = "gpio-leds";

pwr\_led {

label = "orangepi:green:status";

gpios = <&r\_pio 0 10 GPIO\_ACTIVE\_HIGH>;

default-state = "on";

linux,default-trigger ="mmc0";

};

status\_led {

label = "orangepi:red:system";

gpios = <&pio 0 17 GPIO\_ACTIVE\_HIGH>;

default-state = "on";

linux,default-trigger ="heartbeat";

};

};

GPL使用r\_pio控制器，其他IO使用pio控制器

gpios中的内容为：<IO控制器 组号 IO偏移量 电平>

刷机测试成功结果：

1. 红色LED心跳闪烁、绿色LED随TF卡读写闪烁；
2. /sys/class/leds/有两个文件夹：orangepi:green:status、orangepi:red:status

## 二、去掉/dev/pty\*

Device Drivers --->

Character devices --->

[ ] Legacy (BSD) PTY support

/dev/ttyxx数量也有点多，改少点include/uapi/linux/vt.h：

#define MAX\_NR\_CONSOLES 63

主要是看着太多，烦。

## 三、加入SPI-Flash驱动

### 3.1 设备树中加入节点

修改设备树linux-5.6.2/arch/arm/boot/dts/ sun8i-h2-plus-orangepi-zero.dts内容：

### 3.2 配置驱动

&spi0 {

status = "okay";

flash@0 {

#address-cells = <1>;

#size-cells = <1>;

compatible = "mxicy,mx25l1606e", "winbond,w25q128";

reg = <0>;

spi-max-frequency = <40000000>;

};

};

Device Drivers --->

<\*> Memory Technology Device (MTD) support --->

<\*> Caching block device access to MTD devices

<\*> SPI-NOR device support --->

[\*] Use small 4096 B erase sectors

### 3.3 验证SPI-Nor-Flash

在/dev/下出现

mtd0 mtd0ro mtdblock0 三个设备节点

mtd0是对应的字符设备

mtd0ro是对应的只读字符设备

mtdblock0 是对应的块设备，同硬盘一样使用

将u-boot刷入Flash中测试：

# dd if=u-boot-sunxi-with-spl.bin of=/dev/mtdblock0

断电，取下TF卡，上电，串口有u-boot的输出内容则成功。

## 四、加入WiFi驱动

### 4.1 下载驱动源码

#### 1、源码地址：

<https://github.com/Yuyaowen/xradio.git>

#### 2、固件地址：

<https://github.com/armbian/firmware/tree/master/xr819>

### 4.2 配置驱动

#### 1、下载的驱动放入内核源码树：

linux-5.6.2/drivers/net/wireless/xradio

编辑linux-5.6.2/drivers/net/wireless/Kconfig文件，

加入语句source "drivers/net/wireless/xradio/Kconfig"

编辑linux-5.6.2/drivers/net/wireless/Makefile文件，

加入语句obj-$(CONFIG\_XRADIO) += xradio/

#### 2、配置驱动并编译

[\*] Networking support --->

-\*- Wireless --->

<\*> cfg80211 - wireless configuration API

[\*] cfg80211 wireless extensions compatibility

<\*> Generic IEEE 802.11 Networking Stack (mac80211)

Device Drivers --->

[\*] Network device support --->

[\*] Wireless LAN --->

<M> XRADIO WLAN support

[\*] Platform supports non-power-of-two SDIO transfer

[\*] 5GHz band support

[\*] WAPI support

[\*] Extensions for WFD and PS mode

注：

必须以模块形式编译，因为需要加载文件系统中的固件文件。

直接编进内核，在加载的时候，文件系统还没准备就绪，所以会找不到固件文件。

#### 3、 设备树修改

修改文件linux-5.6.2/arch/arm/boot/dts/ sun8i-h2-plus-orangepi-zero.dts

&mmc1 {

pinctrl-names = "default";

vmmc-supply = <&reg\_vcc3v3>;

vqmmc-supply = <&vdd\_wifi>;

mmc-pwrseq = <&pwrseq\_wifi>;

bus-width = <4>;

max-frequency = <16000000>;

non-removable;

status = "okay";

/\*

\* Explicitly define the sdio device, so that we can add an ethernet

\* alias for it (which e.g. makes u-boot set a mac-address).

\*/

xr819: sdio\_wifi@1 {

reg = <1>;

compatible = "xradio,xr819";

pinctrl-names = "default";

pinctrl-0 = <&wifi\_wake>;

interrupt-parent = <&pio>;

interrupts = <6 10 IRQ\_TYPE\_EDGE\_RISING>;

interrupt-names = "host-wake";

local-mac-address = [dc 44 6d c0 ff ee];

};

};

&pio {

wifi\_wake: wifi\_wake {

pins = "PG10";

function = "gpio\_in";

};

};

&r\_pio {

wifi\_rst: wifi\_rst {

pins = "PL7";

function = "gpio\_out";

};

};

#### 4、放入固件

vdd\_wifi: vdd\_wifi {

compatible = "regulator-fixed";

regulator-name = "wifi";

regulator-min-microvolt = <1800000>;

regulator-max-microvolt = <1800000>;

gpio = <&pio 0 20 GPIO\_ACTIVE\_HIGH>;

startup-delay-us = <70000>;

enable-active-high;

};

reg\_vdd\_cpux: vdd-cpux-regulator {

compatible = "regulator-gpio";

regulator-name = "vdd-cpux";

regulator-type = "voltage";

regulator-boot-on;

regulator-always-on;

regulator-min-microvolt = <1100000>;

regulator-max-microvolt = <1300000>;

regulator-ramp-delay = <50>; /\* 4ms \*/

gpios = <&r\_pio 0 6 GPIO\_ACTIVE\_HIGH>; /\* PL6 \*/

enable-active-high;

gpios-states = <1>;

states = <1100000 0 1300000 1>;

};

pwrseq\_wifi: pwrseq\_wifi@0 {

compatible = "mmc-pwrseq-simple";

pinctrl-names = "default";

pinctrl-0 = <&wifi\_rst>;

reset-gpios = <&r\_pio 0 7 GPIO\_ACTIVE\_LOW>;

post-power-on-delay-ms = <50>;

};

将 boot\_xr819.bin fw\_xr819.bin sdd\_xr819.bin 三个文件放入文件系统的

/lib/firmware/xr819/ 中

### 4.3 连接WLAN

使用wpa\_supplicant作为连接WLAN的工具，wireless-tools作为管理工具。

#### 1、移植wpa\_supplicant

源码下载地址 <http://w1.fi/wpa_supplicant/>

进入源码，复制defconfig为.config

依赖库：

#### 2、libnl 3.2

下载地址 <http://www.linuxfromscratch.org/blfs/view/7.7/basicnet/libnl.html>

编译此库需要宿主机有bison flex两个命令，安装：

# sudo apt-get install bison flex

配置与交叉编译：

# ./configure --host=arm-linux-gnueabihf --prefix=$(pwd)/\_install --enable-shared --enable-static

# make -j32

# make install

在wpa\_supplicant的.config中加入

CFLAGS += -I../../libnl-3.2.25/\_install/include/libnl3

LIBS += -L../../libnl-3.2.25/\_install/lib

#### 3、OpenSSL

下载地址 <https://www.openssl.org/source/>

配置与交叉编译：

# ./config no-asm shared no-async --prefix=$(pwd)/\_install --cross-compile-prefix=arm-linux-gnueabihf-

编辑Makefile文件，去掉其中所有的 -m64和-m32

# make –j32

# make install

在wpa\_supplicant的.config中加入

CFLAGS += -I../../openssl-1.1.1f/\_install/include

LIBS += -L../../openssl-1.1.1f/\_install/lib

#### 4、配置与编译wpa\_supplicant

配置环境变量（包含libnl-3.0.pc的路径，可以不配置）

# export PKG\_CONFIG\_PATH=~/xxx/libnl-3.2.25/\_install/lib/pkgconfig

去掉DBUS相关配置：

CONFIG\_CTRL\_IFACE\_DBUS\_NEW=y

CONFIG\_CTRL\_IFACE\_DBUS\_INTRO=y

# make –j32

涉及到的动态库：

libnl-3.so.200 libnl-3.so.200.20.0 libnl-genl-3.so.200 libnl-genl-3.so.200.20.0

libssl.so.1.1 libcrypto.so.1.1

#### 5、运行时配置文件

/etc/wpa\_supplicant.conf

network={

ssid="xxx"

psk="12345678"

}

连接：

 wpa\_supplicant -Dnl80211 -iwlan0 -c/etc/wpa\_supplicant.conf -B

获取IP

udhcpc -i wlan0

## 五、加入声卡驱动

### 5.1 设备树中修改节点

修改设备树linux-5.6.2/arch/arm/boot/dts/ sun8i-h2-plus-orangepi-zero.dts内容

&codec {

status = "okay";

allwinner,audio-routing =

"Line Out", "LINEOUT",

"MIC1", "Mic",

"Mic", "MBIAS";

};

### 5.2 配置驱动

Device Drivers --->

<\*> Sound card support --->

<\*> Advanced Linux Sound Architecture --->

<\*> ALSA for SoC audio support --->

Allwinner SoC Audio support --->

<\*> Allwinner A10 Codec Support

<\*> Allwinner sun8i Codec Analog Controls Support

编译替换zImage和dtb，可以看到/dev/snd/中有以下几个文件即可：

controlC0 pcmC0D0c pcmC0D0p timer

### 5.3 播放声音

用到的软件和库有：

alsa-utils、alsa-lib、ncurses 、madplay、libid3tag、libmad、zlib

命令alsamixer在alsa-utils中，用于调整系统音量及静音（按m键切换静音）；

命令madplay用于播放音频文件，支持mp3

#### 1、alsa-lib

官方下载地址 <https://www.alsa-project.org/wiki/Download>

编译配置：

# ./configure --prefix=$(pwd)/\_install --host=arm-linux-gnueabihf

# make –j32

# make install

#### 2、ncurses

官方下载地址 <http://ftp.gnu.org/gnu/ncurses/>

编译配置：

# ./configure --prefix=$(pwd)/\_install --host=arm-linux-gnueabihf --enable-widec

# make –j32

# make install（这步没成功）

#### 3、alsa-utils

官方下载地址

<https://www.alsa-project.org/wiki/Download>

编译配置：

# ./configure --prefix=$(pwd)/\_install --host=arm-linux-gnueabihf CPPFLAGS="-I$(pwd)/../alsa-lib-1.2.2/\_install/include -I$(pwd)/../ncurses-6.2/include -I$(pwd)/../ncurses-6.2/\_install/include -I$(pwd)/../ncurses-6.2/\_install/include/ncursesw" LDFLAGS="-L$(pwd)/../alsa-lib-1.2.2/\_install/lib -L$(pwd)/../ncurses-6.2/lib"

注：上方配置命令无换行。

# make –j32

# make install（这步没成功）

## 六、加入外部LCD驱动（SPI-ILI9488）

所使用的的3.5寸LCD为SPI接口，根据Datasheet可知，该控制方式仅支持两种模式：

RGB 1:1:1

RGB 6:6:6: （不是656）

### 6.1 寻找驱动源码，根据Datasheet修改参数

略

### 6.2 配置设备树

修改文件linux-5.6.2/arch/arm/boot/dts/ sun8i-h2-plus-orangepi-zero.dts

&spi1 {

status = "okay";

tft-lcd@0 {

compatible = "ilitek,ili9488";

reg = <0>;

spi-max-frequency = <74999000>;

reset-gpios = <&pio 0 3 GPIO\_ACTIVE\_HIGH>;

dc-gpios = <&pio 0 7 GPIO\_ACTIVE\_HIGH>;

rotate = <270>;

fps = <60>;

buswidth = <8>;

regwidth = <8>;

width = <320>;

height = <480>;

bpp = <16>;

};

};

### 6.3 背光控制

开发板上SoC实际只有1个PWM引脚并且与UART0\_RX复用，所以就没有合适的背光调节引脚了。

#### 1、配置设备树

backlight {

compatible = "gpio-backlight";

gpios = <&pio 0 6 GPIO\_ACTIVE\_HIGH>;

default-on;

};

#### 2、配置驱动

Device Drivers --->

Graphics support --->

Backlight & LCD device support --->

<\*> Generic GPIO based Backlight Driver

### 6.4 触摸屏

此触摸屏控制器使用的是XPT2046，兼容ADS7846、TSC2046

开发板硬件带的SPI不支持扩展CS引脚，只能用GPIO再模拟一个SPI接口

#### 1、配置驱动

Device Drivers --->

Input device support --->

[\*] Touchscreens --->

<\*> ADS7846/TSC2046/AD7873 and AD(S)7843 based touchscreens

Device Drivers --->

[\*] SPI support --->

<\*> GPIO-based bitbanging SPI Master

#### 2、配置设备树

spi2: spi-gpio {

compatible = "spi-gpio";

#address-cells = <0x1>;

#size-cells = <0x0>;

ranges;

sck-gpios = <&pio 0 18 GPIO\_ACTIVE\_HIGH>;

miso-gpios = <&pio 0 19 GPIO\_ACTIVE\_HIGH>;

mosi-gpios = <&pio 0 1 GPIO\_ACTIVE\_HIGH>;

cs-gpios = <&pio 0 10 GPIO\_ACTIVE\_HIGH>;

num-chipselects = <1>;

touchscreen@1 {

compatible = "ti,ads7846";

reg = <1>;

spi-max-frequency = <500000>;

interrupt-parent = <&pio>;

interrupts = <0 2 2>;

pendown-gpio = <&pio 0 2 0>;

wakeup-source;

};

};

## 七、开发板模拟优盘

### 7.1 配置驱动

Device Drivers --->

[\*] USB support --->

<\*> USB Gadget Support --->

<\*> USB Gadget functions configurable through configfs

[\*] Mass storage

<M> USB Gadget precomposed configurations

<M> Mass Storage Gadget

### 7.2 加载驱动

在内核源码目录中找到linux-5.6.2/drivers/usb/gadget/legacy/g\_mass\_storage.ko放入开发板中

在开发板中寻找一个文件（磁盘文件、虚拟磁盘镜像等，例如：/dev/mmcblk0）

执行命令：

# insmod g\_mass\_storage.ko file=/dev/mmcblk0p1 removable=1

使用microUSB线连接PC和开发板

注意：

在PC机上操作优盘之后，开发板中需要重新挂载才能正常使用。

## 八、添加UVC摄像头驱动

Device Drivers --->

<\*> Multimedia support --->

[\*] Cameras/video grabbers support

[\*] Media USB Adapters --->

<\*> USB Video Class (UVC)

[\*] UVC input events device support

更新内核，插入USB摄像头后，可以找到/dev/videox

## 九、添加U盘、移动硬盘支持

Device Drivers --->

[\*] USB support --->

<\*> USB Mass Storage support

该选项下的子选项都选上即可

更新内核，插入U盘或者移动硬盘，可以找到/dev/sdxx