Firefly-RK3288 用户指南

V1.1

2015/01/15

版本历史

日期	版本	描述
2014-12-29	V1.0	初始版本
2015-01-15	V1.1	同步官网 wiki，优化排版

序言

Firefly-RK3288 是一个高性能平台，它拥有强大的多线程运算能力、图形处理能力以及硬件解码能力，支持 Android 4.4 和 Ubuntu 双系统，也是一台强大的微型电脑。

Firefly-RK3288 除了可以当作电视盒子，微型电脑使用，还适用于多种的领域。Firefly-RK3288 板级支持红外、蓝牙 4.0、双频 WIFI、4K*2K 高清输出，及丰富的外围扩展接口，可以满足一些高性能场合，如 DIY 出智能客厅，智能厨房，背景音乐及智能家居中控等。Firefly-RK3288 支持 4K 高清输出及 H.265 硬解码，可以打造高清室内室外广告机，还可以打造车载智能娱乐影音中心等等。除些之处，由于 Firefly-RK3288 配置高，有足够多的外围接口，可以跑双系统，因些不仅是初学者学习 Linux 及安卓软件开发的极佳平台，也可以让爱好者（极客）扩展出多种玩法，具有无限可能。

欢迎访问我们的官网：www.t-firefly.com

欢迎到我们的开发者社区交流：http://developer.t-firefly.com

目录

第 1 章 上手教程

1.1 入手指南

1.1.1 配件

Firefly-RK3288 的标准套装包含以下配件：

- Firefly-RK3288 主板一块
- USB 接口的电源线一根
- WiFi 天线
- 亚克力架子（保护主板）

另外可以选购的配件有：

- 5V/2.5A 的直流电源适配器（支持 220V 和 110V 输入）
- Firefly 串口模块
- 散热风扇套件（5V/0.12A）
- 散热片套装
- 10.1 寸 LVDS 显示屏
- OV13850 摄像头模组（1300W 像素）

另外，在使用过程中，你可能需要以下配件：

- 显示设备
- 带 VGA 接口的显示器或电视，及 VGA 连接线
- 带 HDMI 接口的显示器或电视，及 HDMI 连接线
- 网络
- 100M/1000M 以太网线缆，及有线路由器
- WiFi 路由器
- 输入设备
- USB 无线/有线的鼠标/键盘
- 红外遥控器
- 升级固件，调试
- Micro USB 连接线

1.1.2 开机

确认主板配件连接无误后，将电源适配器插入带电的插座上，电源线接口插入开发板，开发板第一次加电会自动开机。

在 Android 系统选择关机后，维持开发板供电，此时 Firefly-RK3288 有两种开机方式：

1. 长按电源键三秒
2. 按红外遥控器上的开机按钮

开机时，蓝色的电源指示灯会亮起。显示设备上会出现 Linux 的四只企鹅标志，代表四核 CPU。

1.1.3 常见问题

1.1.3.1 Android 和 Ubuntu 双系统切换方法

Firefly-RK3288 开发板默认安装了双系统固件，下面告诉你如何进行 Android 和 Ubuntu 系统的切换。

1.1.3.2 从 Android 切换到 Ubuntu

进入 Android 系统后，点击状态栏的【关机按钮】会弹出关机对话框。假如你使用的是双系统固件，关机对话框会出现【Switch System】选项，只要点击它就可以切换到 Ubuntu 系统，如图 1-1-1 所示。

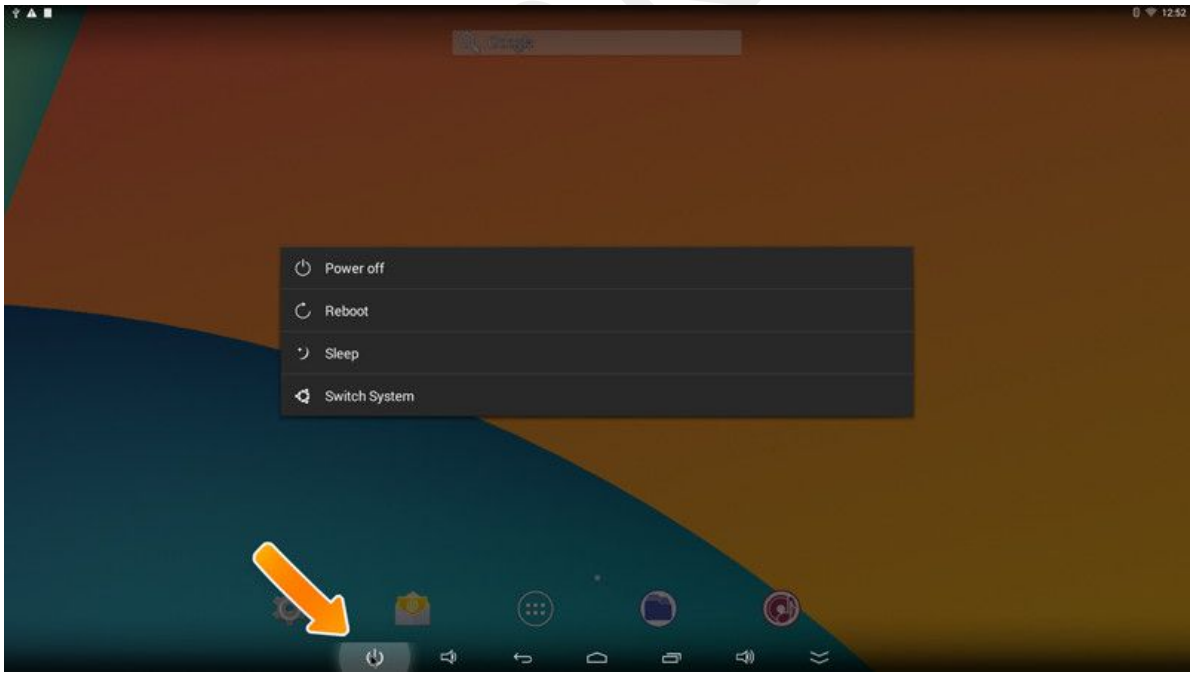


图 1-1-1 Android 切换到 Ubuntu

1.1.3.3 从 Ubuntu 切换到 Android

在 Ubuntu 的桌面有一个【Boot Android】的图标，只要你点击它并按【确认】，就可

以切换到 Android 系统，如图 1-1-2 所示。

图 1-1-2 Ubuntu 切换到 Android

1.1.3.4 如何切换板载麦克风和耳麦

Firefly-RK3288 可以使用板子麦克风或耳麦来进行音频输入，而系统默认为【自动切换模式】，你可以进入【Setting】->【Sound】->【ES8323 Microphone Manager】选择切换方式，如图 1-1-3 所示。

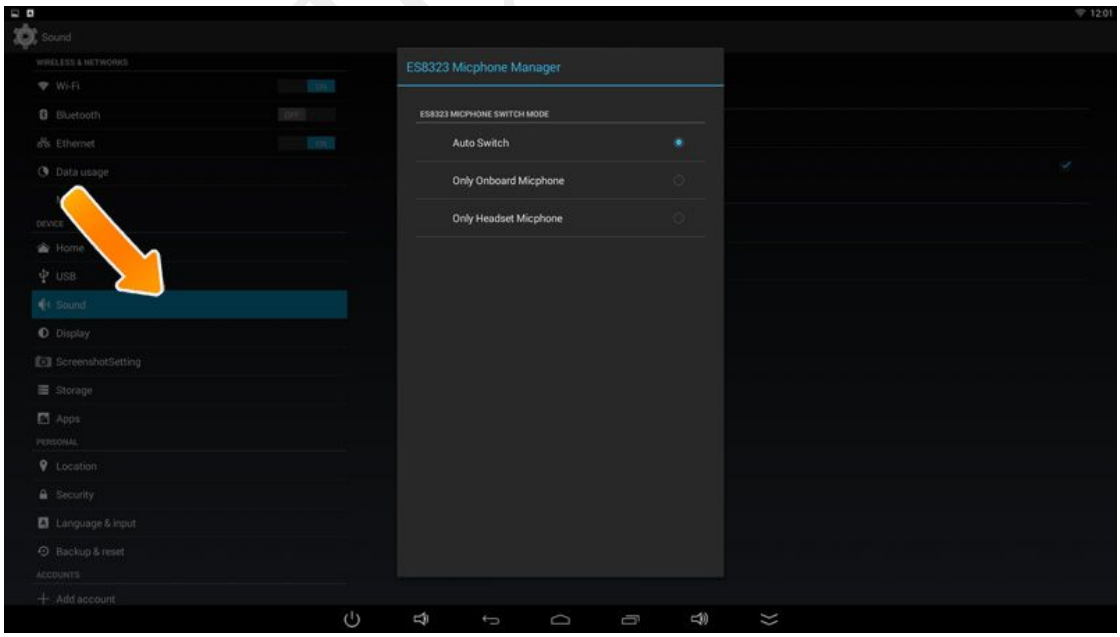


图 1-1-3 选择切换板载麦克风和耳麦方式

1.1.3.5 关于 VGA 的自动识别

Firefly-RK3288 的 VGA 能自动识别显示的分辨率。假如无法读取显示器的 EDID (Extended Display Identification Data, 扩展显示器标识数据)，VGA 会默认设置为 1080P 的分辨率。你也可以进入系统设置对 VGA 分辨率进行手动调整，如图 1-1-4 所示。

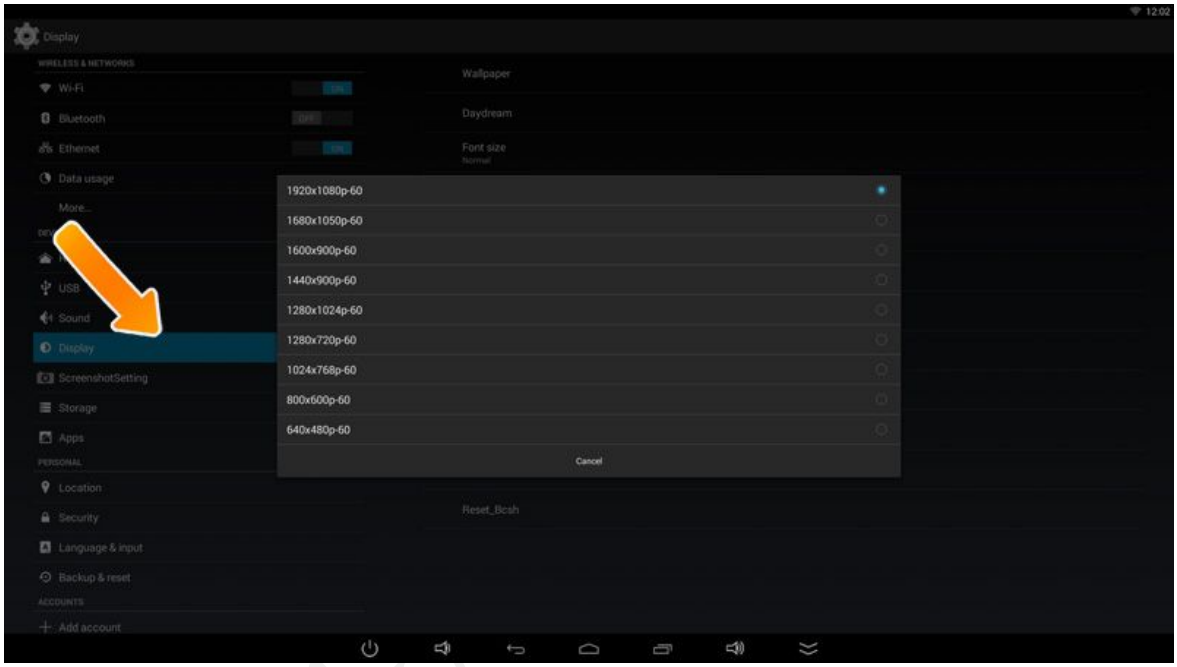


图 1-1-4 设置 VGA 分辨率

1.1.3.6 关于电源

Firefly-rk3288 开发板正常工作需要电源 5V/2.5A，电流低于 2.5A 可能会因电流过小而异常重启，为了保证开发板的正常工作，请使用电压为 5V，电流为 2.5A~3A 的电源，推荐使用 Firefly-rk3288 官网的电源配件。

1.1.3.7 关于散热风扇

Firefly-rk3288 官网中的散热风扇工作电压为 5V，在 Firefly-rk3288 开发板中有对应的接口，标记为：FAN+ FAN-，风扇的黑色电源线对应 FAN-，红色电源线对应 FAN+，本端口直接与开发板的电源模块连接，不能通过软件控制，具体连接如图 1-1-5 所示。

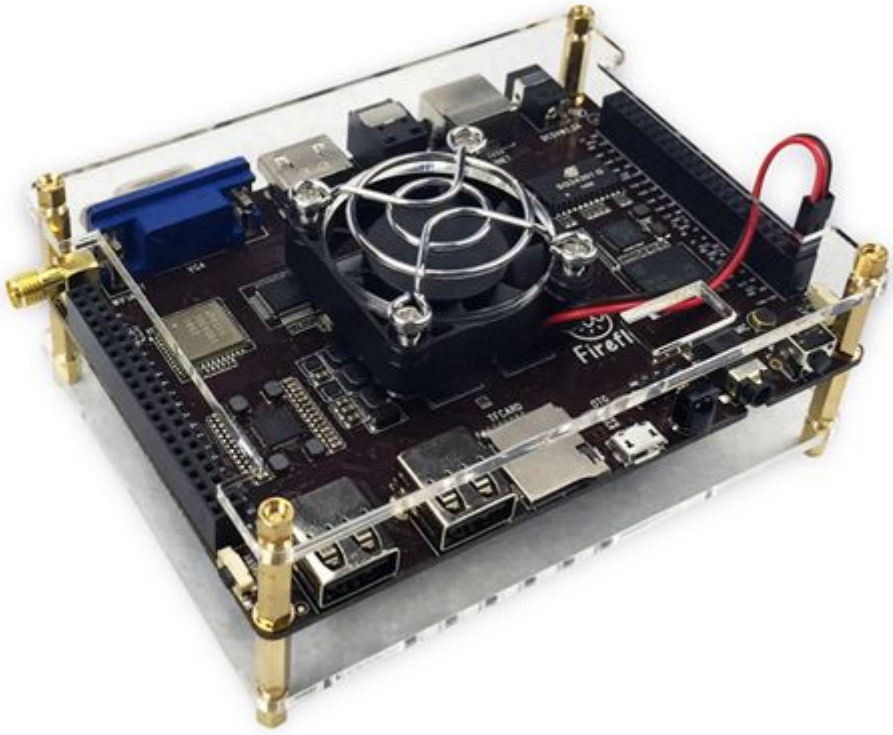


图 1-1-5 散热风扇连接图

1.2 固件升级

1.2.1 前言

本文介绍了如何将主机上的固件文件，通过 Micro USB OTG 线，烧录到开发板的闪存中。升级时，需要根据主机操作系统和固件类型来选择合适的升级方式。

1.2.2 准备工作

- Firefly RK3288 开发板
- 固件
- 主机
- 良好的 Micro USB OTG 线

固件文件一般有两种：

- 单个统一固件 update.img, 将启动加载器、参数和所有分区镜像都打包到一起，用于固件发布。
- 多个分区镜像,如 kernel.img, boot.img, recovery.img 等，在开发阶段生成。

这里可以找到已编译好的统一固件，下载后解压。也可以参考编译固件的说明自行编译。

主机操作系统支持：

- Windows XP （32/64 位）
- Windows 7 (32/64 位)
- Windows 8 (32/64 位)
- Linux (32/64 位)

1.2.3 Windows

之前烧写 RK 的固件，需要用到以下两种工具：

- 量产工具 RKBatchTool，用于烧写统一固件（update.img）
- 开发者工具 RKDevelopTool，可单独烧写分区固件

后来 RK 发布了 AndroidTool 工具，在 RKDevelopTool 的基础上增加了统一固件（update.img）的烧写支持，因此现在仅需要这个工具即可。

使用烧写工具前需要安装 RK USB 驱动。如果驱动已经安装好，可以跳过这步。

1.2.4 安装 RK USB 驱动

下载 Release_DriverAssistant.zip ，解压，然后运行里面的 DriverInstall.exe 。
为了所有设备都使用更新的驱动，请先选择"驱动卸载"，然后再选择"驱动安装"，如图 1-2-1 所示。

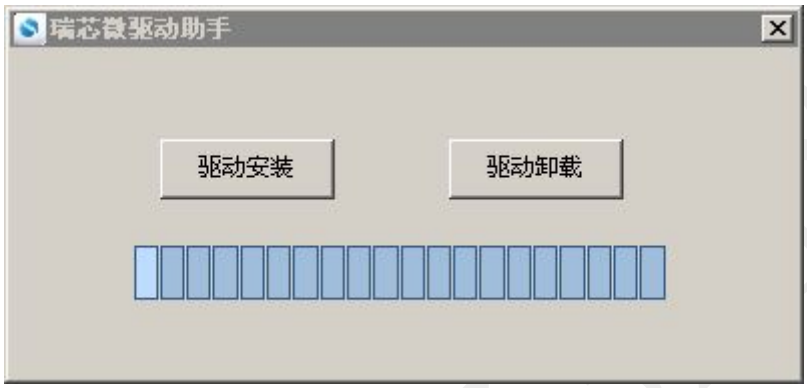


图 1-2-1 安装 RK USB 驱动

1.2.5 连接设备

1．确保设备连接好电源适配器并处于通电状态。
2．用 Micro USB OTG 线连接好设备和主机。
3．按住设备上的 RECOVERY （恢复）键并保持。
4．短按一下 RRESET（复位）键。
5．大约两秒钟后，松开 RECOVERY 键。
6．注意：如果发现按了 RESET 键后还是没有发现设备，请在保持 RECOVERY 键按下的同时，长按一下 PWRKEY 键，然后才松开 RECOVERY 键。

主机应该会提示发现新硬件并配置驱动。打开设备管理器，会见到新设备"Rockusb Device" 出现，如图 1-2-2 所示。如果没有，则需要返回上一步重新安装驱动。

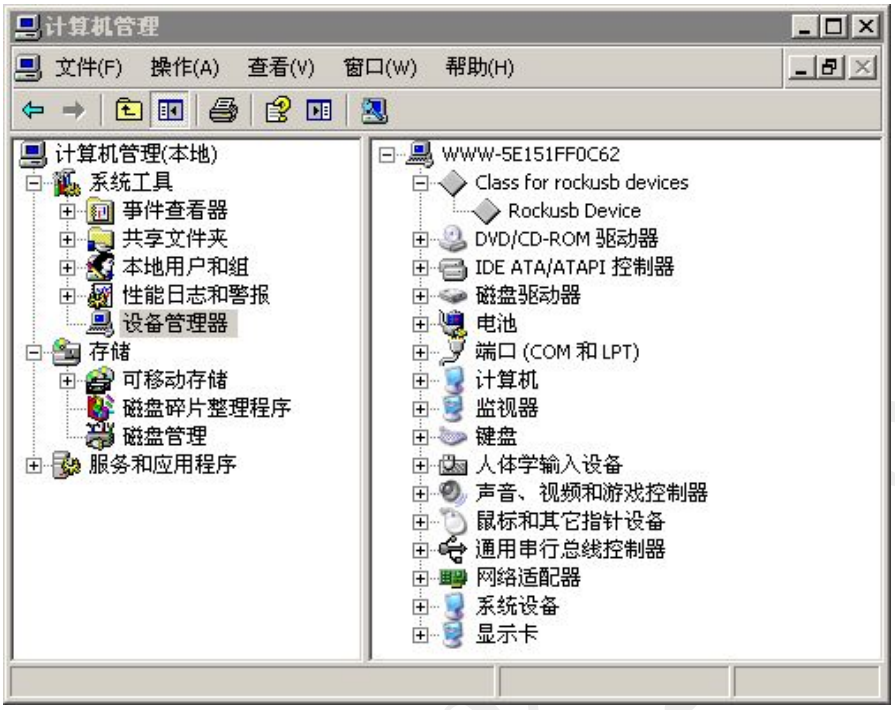


图 1-2-2 查看 Rockusb Device

1.2.6 烧写固件

下载 AndroidTool_Release_v2.3.rar，解压，运行 AndroidTool_Release_v2.3 目录里面的 AndroidTool.exe（注意，如果是 Windows 7/8,需要按鼠标右键，选择以管理员身份运行），如图 1-2-3 所示：

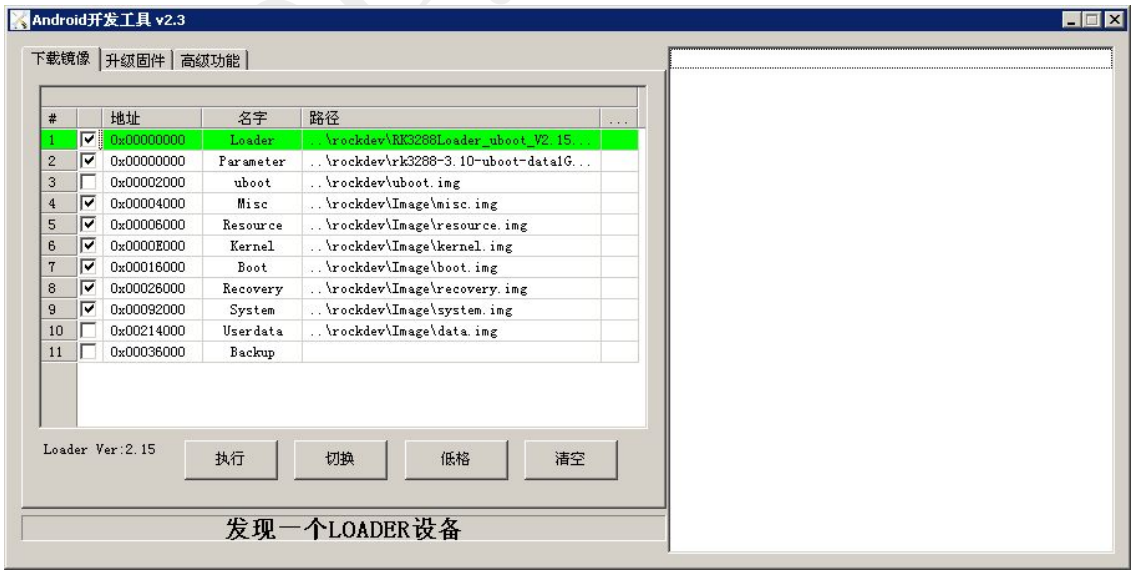


图 1-2-3 AndroidTool

1.2.7 烧写统一固件 update.img

烧写统一固件 update.img 的步骤如下:

1. 切换至"升级固件"页，如图 1-2-4 所示。
2. 按"固件"按钮，打开要升级的固件文件。升级工具会显示详细的固件信息。
3. 按"升级"按钮开始升级。
4. 如果升级失败，可以尝试先按"擦除 Flash"按钮来擦除 Flash，然后再升级。
5. 注意：如果你烧写的固件 laoder 版本与原来的机器的不一致，请在升级固件前先执行"擦除 Flash"。

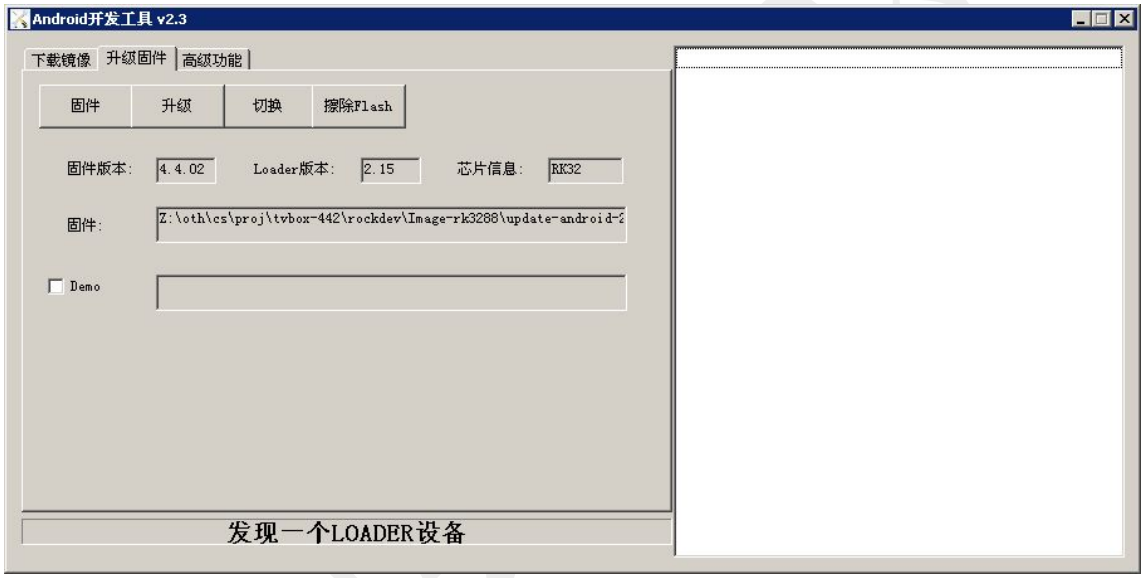


图 1-2-4 烧写统一固件

1.2.8 烧写分区映像

烧写分区映像的步骤如下：

1. 切换至"下载镜像"页，如图 1-2-5 所示。
2. 勾选需要烧录的分区，可以多选。
3. 确保映像文件的路径正确，需要的话，点路径右边的空白表格单元格来重新选择。
4. 点击"执行"按钮开始升级，升级结束后设备会自动重启。

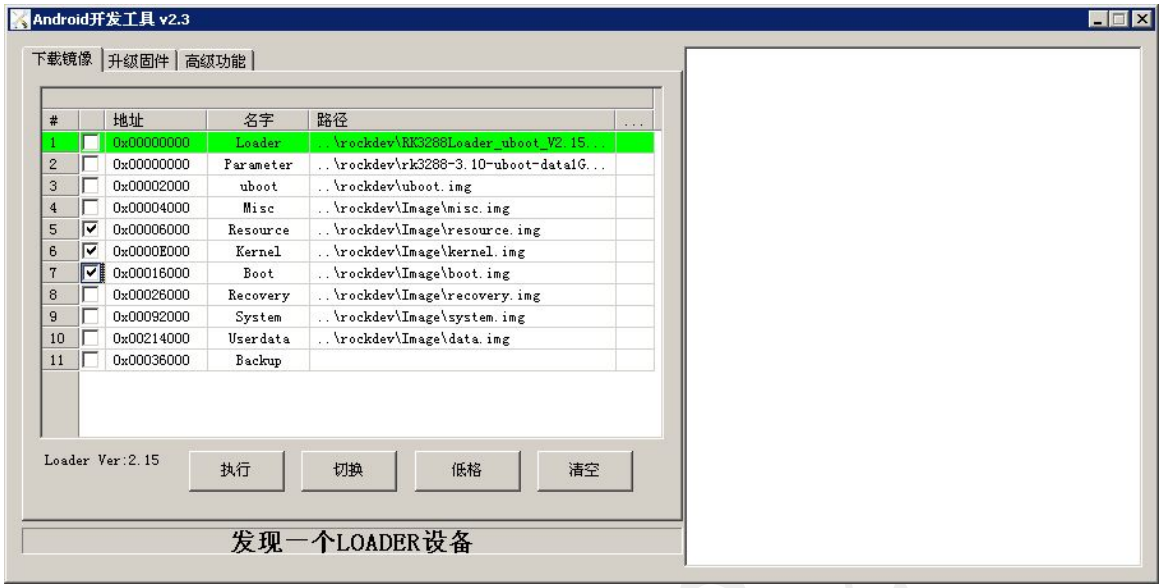


图 1-2-5 烧写分区映像

1.2.9 Linux

RK 提供了一个 Linux 下的命令行工具 upgrade_tool，支持统一固件 update.img 和分区镜像的烧写。

开源工具则有两个选择:

- rkflashtool https://github.com/Galland/rkflashtool_rk3066
- rkflashkit https://github.com/linuxerwang/rkflashkit

它们都仅支持分区映像烧写，不支持统一固件。rkflashtool 是命令行工具，rkflashkit 有图形界面，后加了命令行支持，更是好用。以下仅对 rkflashkit 做介绍。

Linux 下无须安装设备驱动，参照 Windows 章节连接设备则可。

1.2.10 upgrade_tool

下载 Linux_Upgrade_Tool_v1.2.tar.gz, 并按以下方法安装到系统中，方便调用：

```
tar xf Linux_UpgradeTool_v1.2.tar.gz
cd Linux_UpgradeTool_v1.2
sudo mv upgrade_tool /usr/local/bin
sudo chown root:root /usr/local/bin/upgrade_tool
```

烧写统一固件 update.img：

```
sudo upgrade_tool uf update.img
```

烧写分区镜像：

```
upgrade_tool di -b /path/to/boot.img
```

upgrade_tool di -k /path/to/kernel.img
upgrade_tool di -s /path/to/system.img
upgrade_tool di -r /path/to/recovery.img
upgrade_tool di -m /path/to/misc.img
upgrade_tool di -p paramater #烧写 parameter
upgrade_tool ul bootloader.bin # 烧写 bootloader:

如果因 flash 问题导致升级时出错，可以尝试低级格式化、擦除 nand flash：：

upgrade_tool lf # 低级格式化
upgrade_tool ef # 擦除

1.2.11 rkflashkit

1. 安装：

sudo apt-get install build-essential fakeroot
git clone https://github.com/linuxerwang/rkflashkit
cd rkflashkit
./waf debian
sudo apt-get install python-gtk2
sudo dpkg -i rkflashkit_0.1.2_all.deb

2. 图形界面：

sudo rkflashkit

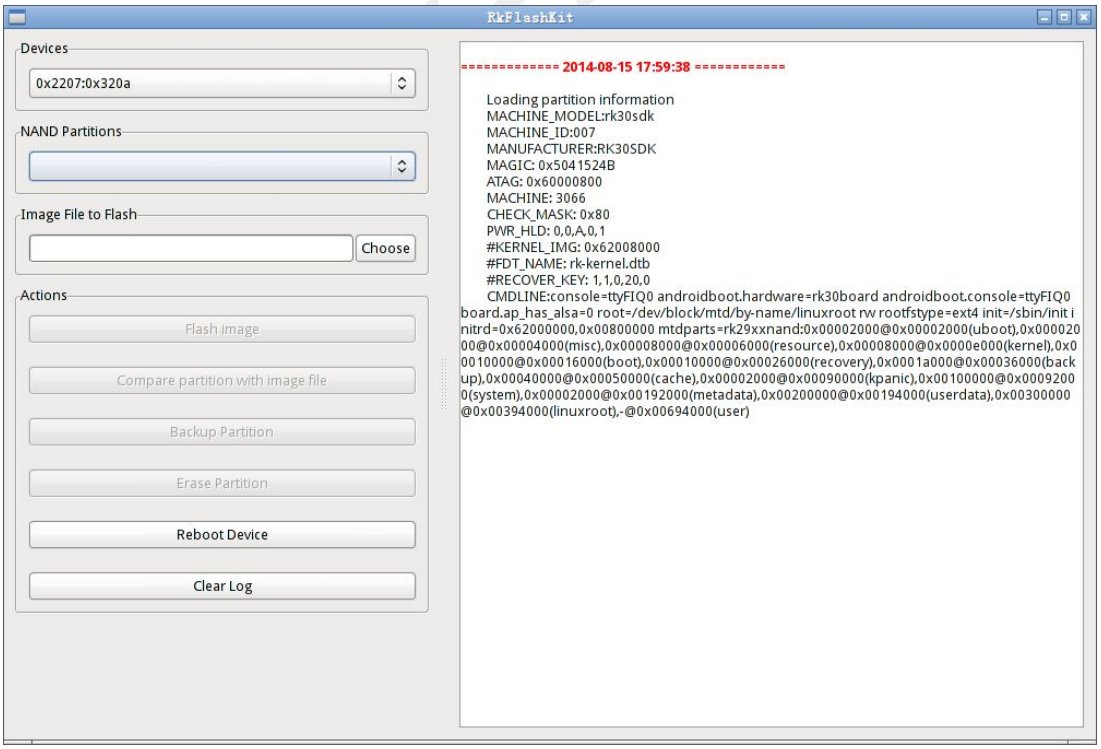


图 1-2-6 rkflashkit 图形化界面

3. 命令行：

```
$ rkflashkit --help
Usage: <cmd> [args] [<cmd> [args]...]

part                                    List partition
flash @<PARTITION> <IMAGE FILE>     Flash partition with image file
cmp @<PARTITION> <IMAGE FILE>       Compare partition with image file
backup @<PARTITION> <IMAGE FILE>    Backup partition to image file
erase   @<PARTITION>                    Erase partition
reboot                                  Reboot device

For example, flash device with boot.img and kernel.img, then reboot:

  sudo rkflashkit flash @boot boot.img @kernel.img kernel.img reboot
```

帮助信息里有使用示例，可以看出，一条命令就可以烧写多个映像文件并重启设备，对需要经常编译和烧写内核的开发者来说，是一大福音。

1.3 串口调试

1.3.1 选购适配器

网店上有许多 USB 转串口的适配器，按芯片来分，有以下几种：

- PL2303
- CH340

一般来说，采用 CH340 芯片的适配器，性能比较稳定，价格上贵一些。

1.3.2 硬件连接

串口转 USB 适配器，有四根不同颜色的连接线：

- 红色：3.3V 电源，不需要连接
- 黑色：GND，串口的地线，接开发板串口的 GND 针
- 白色：TXD，串口的输出线，接开发板串口的 TX 针
- 绿色：RXD，串口的输入线，接开发板串口的 RX 针

1.3.3 连接参数

Firefly-RK3288 使用以下串口参数：

- 波特率：115200
- 数据位：8
- 停止位：1
- 奇偶校验：无
- 流控：无

1.3.4 Windows 上使用串口调试

1.3.4.1 安装驱动

下载驱动并安装:

- CH340
- PL2303

如果在 Win8 上不能正常使用 PL2303，参考这篇文章， 采用 3.3.5.122 或更老版本的旧驱动即可。

插入适配器后，系统会提示发现新硬件，并初始化，之后可以在设备管理器找到对应的

COM 口，如图 1-3-1 所示。

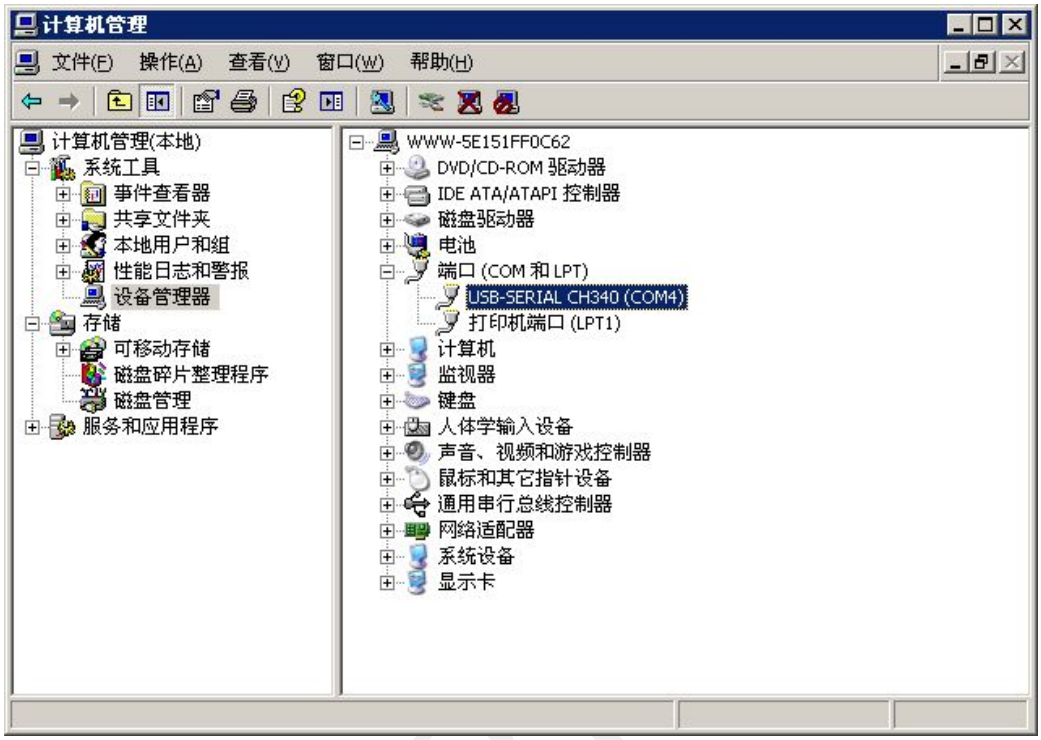


图 1-3-1 查看 COM 口

1.3.4.2 安装软件

Windows 上一般用 putty 或 SecureCRT。其中 putty 是开源软件，在这里介绍一下，SecureCRT 的使用方法与之类似。

- 到这里 下载 putty，建议下载 putty.zip，它包含了其它有用的工具。
- 解压后运行 PUTTY.exe，选择 Connection type（连接类型）为 Serial（串口），将 Serial line（串口线）设置成设备管理器所看到的 COM 口，并将 Speed（波特率）设置为 115200，按 Open（打开）即可，如图 1-3-2 所示。

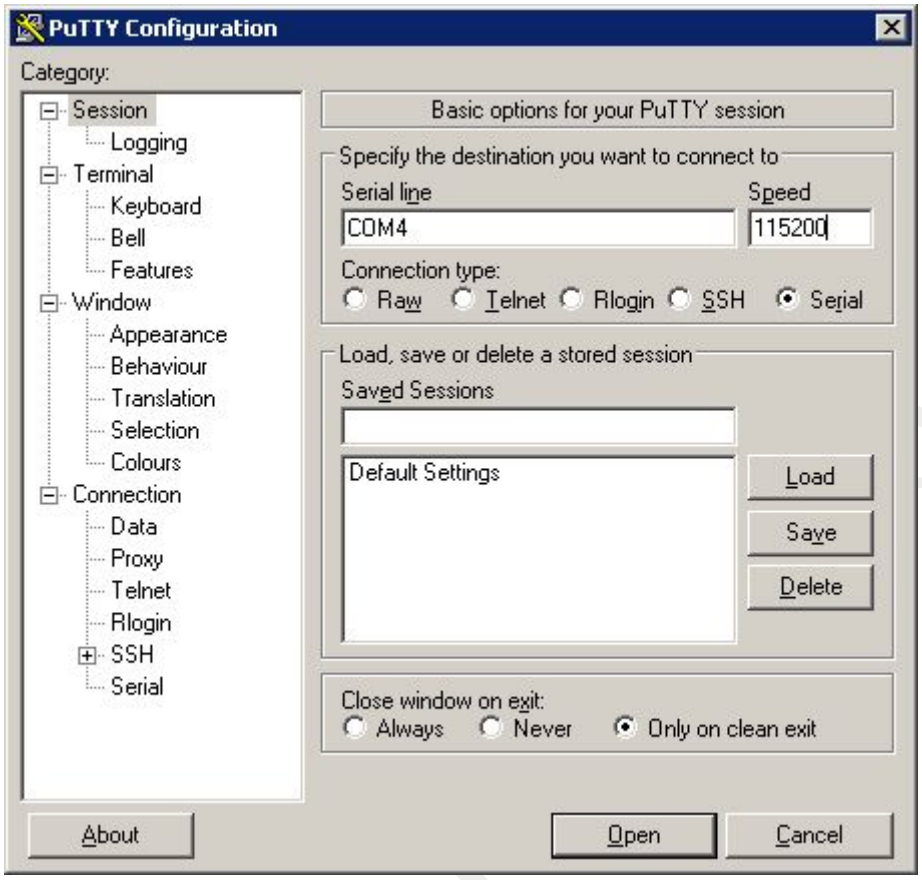


图 1-3-2 设置 PUTTY

1.3.5 Ubuntu 上使用串口调试

在 Ubuntu 上可以有多种选择：

- picocom
- minicom
- kermit

picocom 的使用比较简单，以下就介绍 picocom，其它软件也是类似的。

安装：

sudo apt-get install picocom

连接好串口线的，看一下串口设备文件是什么，下面示例是 /dev/ttyUSB0

$ ls /dev/ttyUSB*

/dev/ttyUSB0

运行：

$ picocom -b 115200 /dev/ttyUSB0

picocom v1.7

port is : /dev/ttyUSB0

```
flowcontrol       : none
baudrate is       : 115200
parity is         : none
databits are      : 8
escape is         : C-a
local echo is     : no
noinit is         : no
noreset is        : no
nolock is         : no
send_cmd is       : sz -vv
receive_cmd is : rz -vv
imap is           :
omap is           :
emap is           : crcrlf,delbs,

Terminal ready
```

以上提示 Ctrl-a 是转义键，按 Ctrl-a Ctrl-q 就可以退出终端。除了 Ctrl-q 外，还有几个比较常用的控制命令：

Ctrl-u ： 提高波特率
Ctrl-d ： 降低波特率
Ctrl-f ： 切换流控设置（硬件流控 RTS/CTS, 软件流控 XON/XOFF, 无 none）
Ctrl-y ： 切换奇偶校验 （偶 even, 奇 odd, 无 none）
Ctrl-b : 切换数据位 （5, 6, 7, 8）
Ctrl-c ： 切换本地回显（local-echo）开关
Ctrl-v ： 显示当前串口参数和状态

1.4 启动模式说明

1.4.1 前言

RK3288 有灵活的启动方式。一般情况下，除非硬件损坏，Firefly-RK3288 开发板是不会变砖的。如果在升级过程中出现意外，bootloader 损坏，导致无法重新升级，此时仍可以进入 MaskRom 模式来修复。

1.4.2 加载方式

RK3288 有 20KB 的 BootRom 和 100KB 的内部 SRAM，支持从以下设备加载系统：
8 位 Async Nand Flash
8 位 toggle Nand Flash
SPI 接口
eMMC 接口
SDMMC 接口
也就是说，除了支持从 Nand Flash、SPI Flash、eMMC Flash 启动外，还支持 SD 卡启动。另外 RK3288 支持从 USB OTG 接口下载系统代码。

1.4.3 启动次序

启动的次序是这样的：
1. 主控上电初始化
2. BootRom 代码在 SRAM 上运行，校验存储设备里的 bootloader
3. 校验通过，加载并运行 bootloader 引导代码
4. bootloader 引导代码负责初始化 DDR 内存，加载 bootloader 完整代码到 DDR 内存中并运行
5. bootloader 加载存储设备上的 Linux 内核，并将执行权交给 Linux 内核

1.4.4 启动模式

RK3288 有三种启动模式：
- Normal 模式
- Loader 模式
- MaskRom 模式

1.4.4.1 Normal 模式

Normal 模式就是正常的启动过程，各个组件依次加载，正常进入系统。

1.4.4.2 Loader 模式

在 Loader 模式下，bootloader 会进入升级状态，等待主机命令，用于固件升级等。要进入 Loader 模式，必须让 bootloader 在启动时检测到 RECOVERY（恢复）键按下，且 USB 处于连接状态：
确保设备连接好电源适配器并处于通电状态。
Micro USB OTG 线连接好设备和主机。
按住设备上的 RECOVERY （恢复）键并保持。
短按一下 RRESET（复位）键。
松开 RECOVERY 键。
注意：如果发现按了 RESET 键后还是没有发现设备，请在保持 RECOVERY 键按下的同时，长按一下 PWRKEY 键，然后才松开 RECOVERY 键。

1.4.4.3 MaskRom 模式

MaskRom 模式用于 bootloader 损坏时的系统修复。一般情况下是不用进入 MaskRom 模式的，只有在 bootloader 校验失败（读取不了 IDR 块，或 bootloader 损坏） 的情况下，BootRom 代码 就会进入 MaskRom 模式。此时 BootRom 代码等待主机通过 USB 接口传送 bootloader 代码，加载并运行之。

第 2 章 Android 开发

2.1 编译 Android 固件

2.1.1 准备工作

编译 Android 对机器的配置要求较高：

- 64 位 CPU
- 16GB 物理内存+交换内存
- 30GB 空闲的磁盘空间用于构建，源码树另外占用大约 8GB

官方推荐 Ubuntu 12.04 操作系统，实际上也可以采用更新的操作系统版本，只需要满足 http://source.android.com/source/building.html 里的软硬件配置即可。

编译环境的初始化可参考 http://source.android.com/source/initializing.html 。

安装 JDK 6：

```
sudo add-apt-repository ppa:webupd8team/java
sudo apt-get update
sudo apt-get install oracle-java6-installer
```

Ubuntu 12.04 软件包安装：

```
sudo apt-get install git gnupg flex bison gperf build-essential \
 zip curl libc6-dev libncurses5-dev:i386 x11proto-core-dev \
 libx11-dev:i386 libreadline6-dev:i386 libgl1-mesa-glx:i386 \
 g++-multilib mingw32 tofrodos gcc-multilib ia32-libs\
 python-markdown libxml2-utils xsltproc zlib1g-dev:i386
```

Ubuntu 13.10/14.04 软件包安装：

```
sudo apt-get install git-core gnupg flex bison gperf libsdl1.2-dev \
 libesd0-dev libwxgtk2.8-dev squashfs-tools build-essential zip curl \
 libncurses5-dev zlib1g-dev pngcrush schedtool libxml2 libxml2-utils \
 xsltproc lzop libc6-dev schedtool g++-multilib lib32z1-dev lib32ncurses5-dev \
 lib32readline-gplv2-dev gcc-multilib libswitch-perl
```

安装 ARM 交叉编译工具链和编译内核相关软件包：

```
sudo apt-get install gcc-arm-linux-gnueabihf \
 lzop libncurses5-dev \
 libssl1.0.0 libssl-dev
```

2.1.2 下载 默认版 Android SDK

注意：如果你在 2014-12-11 之前下载过源码，请重新到云盘下载并更新。由于 SDK 比较大，请选择以下云盘之一下载 firefly-rk3288_android4.4_git_20141211.tar.gz ：

百度云盘

Google Drive

下载完成后先验证一下 MD5 码：

$ md5sum /path/to/firefly-rk3288_android4.4_git_20141211.tar.gz

8fe99f519d487ff40c8bc7b5ded62887 firefly-rk3288_android4.4_git_20141211.tar.gz

确认无误后，就可以解压：

mkdir -p ~/proj/firefly-rk3288

cd ~/proj/firefly-rk3288

tar xf /path/to/firefly-rk3288_android4.4_git_20141211.tar.gz

git reset --hard

git remote add bitbucket https://bitbucket.org/T-Firefly/firefly-rk3288.git

以后就可以直接从 bitbucket 处更新：

git pull bitbucket master:master

也可以到 https://bitbucket.org/T-Firefly/firefly-rk3288/commits/branch/master 在线浏览源码。

另外，linux-rockchip 社区搭建了 git 镜像服务器，详见这里。 如果要下载源码，请用以下命令（可选其它镜像服务器）：

git clone -b firefly/master git://git.us.linux-rockchip.org/rk3288_r-box_android4.4.2_sdk.git

2.1.3 下载 PAD 版 Android SDK

PAD 版 SDK 是具有 PAD 特性的 SDK，可用于调屏，TP，支持横竖屏显示等。

由于 SDK 比较大，请选择以下云盘之一下载

firefly-rk3288_pad_android4.4_git_20141218.tar.gz

百度网盘

Google Drive

下载完成后先验证一下 MD5 码：

$ md5sum /path/to/firefly-rk3288_pad_android4.4_git_20141218.tar.gz

4ba44765fa649bc5cddadd8b349aa8af firefly-rk3288_pad_android4.4_git_20141218.tar.gz

确认无误后，就可以解压： mkdir -p ~/proj/firefly-rk3288_pad

cd ~/proj/firefly-rk3288_pad

tar xf /path/to/firefly-rk3288_pad_android4.4_git_20141218.tar.gz

git reset --hard

git remote add bitbucket https://bitbucket.org/T-Firefly/firefly-rk3288.git

以后就可以直接从 bitbucket 处更新：

git pull bitbucket pad:pad

也可以到 https://bitbucket.org/T-Firefly/firefly-rk3288/commits/branch/pad 在线浏览源码。

2.1.4 编译内核

编译正式版(0930)开发板的内核：
cd ~/proj/firefly-rk3288/kernel
make firefly-rk3288_defconfig
make -j8 firefly-rk3288.img
编译公测版(0809)开发板的内核：
cd ~/proj/firefly-rk3288/kernel
make firefly-rk3288_beta_defconfig
make -j8 firefly-rk3288_beta.img

2.1.5 编译 Android

编译 Android：
cd ~/proj/firefly-rk3288
make -j8
./mkimage.sh

2.1.6 烧写分区映像

上一步骤的 ./mkimage.sh 会重新打包 boot.img 和 system.img, 并将其它相关的映像文件拷贝到目录 rockdev/Image-rk3288/ 中。以下列出一般固件用到的映像文件：
boot.img ： Android 的初始文件映像，负责初始化并加载 system 分区。
kernel.img ： 内核映像。
misc.img ： misc 分区映像，负责启动模式切换和急救模式的参数传递。
recovery.img ： 急救模式映像。
resource.img ： 资源映像，内含开机图片和内核的设备树信息。
system.img ： Android 的 system 分区映像，ext4 文件系统格式。
请参照 如何升级固件 一文来烧写分区映像文件。

如果使用的是 Windows 系统，将上述映像文件拷贝到 AndroidTool （Windows 下的固件升级工具）的 rockdev\Image 目录中，之后参照升级文档烧写分区映像即可，这样的好处是使用默认配置即可，不用修改文件的路径。

2.1.7 打包成统一固件 update.img

在 Windows 下打包统一固件 update.img 很简单，按上一步骤将文件拷贝到 AndroidTool 的 rockdev\Image 目录中，然后运行 rockdev 目录下的 mkupdate.bat 批处理文件

即可创建 update.img 并存放到 rockdev\Image 目录里。

update.img 方便固件的发布，供终端用户升级系统使用。一般开发时使用分区映像比较方便。

2.2 定制 Android 固件

2.2.1 前言

定制 Android 固件，有两种方法：

1. 改源码，然后编译生成固件。
2. 在现有固件的基础上进行裁剪。

前一种方法，可以从各个层面去定制 Android，自由度大，但对编译环境和技术要求比较高，参见编译 Android 固件一文。

现在介绍后一种方法，分为解包、定制和打包三个阶段。主机操作系统为 Linux，采用的工具为开源软件。

2.2.2 固件格式

统一固件 release_update.img，内含启动加载器 loader.img 和真正的固件数据 update.img

```
release_update.img
|- loader.img
`- update.img
```

update.img 是个复合文件，内含多个文件，由 package-file 描述。一个典型的 package-file 为：

```
# NAME          Relative path
package-file    package-file
bootloader      RK3288Loader_uboot_Apr212014_134842.bin
parameterrk3288-3.10-uboot-data1G.parameter.txt
misc        Image/misc.img
kernel          Image/kernel.img
resource            Image/resource.img
boot            Image/boot.img
recovery    Image/recovery.img
system          Image/system.img
backup          RESERVED
update-script   update-script
recover-script  recover-script
```

package-file

update.img 的打包说明文件，update.img 里也含有一份 package-file。

RK3288Loader_uboot_Apr212014_134842.bin
启动加载器，即 bootloader。
rk3288-3.10-uboot-data1G.parameter.txt
参数文件，可以设定内核启动参数，里面有重要的分区信息。
Image/misc.img
misc 分区的映像，用来控制 Android 是正常启动，还是进入急救模式（Recovery Mode）。
Image/kernel.img
Android 内核。
Image/resource.img
资源映像，内有内核开机图片和内核设备树信息（Device Tree Blob)。
Image/boot.img
Android 内核的内存启动盘（initrd)，是内核启动后最先加载的根文件系统，包含重要的初始化动作，一般不需要改动。
Image/recovery.img
Android 急救模式的映像，内含内核和急救模式的根文件系统。
Image/system.img
对应于 Android 的 /system 分区，是以下的定制对象。
解包，就是提取 release_update.img 里的 update.img, 然后解压出内含 package-file 所声明的多个文件。
打包，则是个逆过程，将 package-file 将所列的多个文件合成 update.img，加进 loader.img，最终生成 release_update.img 。

2.2.3 工具准备

```
git clone https://github.com/TeeFirefly/rk2918_tools.git
cd rk2918_tools
make
sudo cp afptool img_unpack img_maker mkkrnlimg /usr/local/bin
```

2.2.4 解包

解压 release_update.img

```
$ cd /path/to/your/firmware/dir
$ img_unpack Firefly-RK3288_Android4.4_20140818.img img
rom version: 4.4.2
build time: 2014-08-18 14:25:57
chip: 80
checking md5sum....OK
```

解压 update.img

```
$ cd img
$ afptool -unpack update.img update
Check file...OK
------- UNPACK -------
package-file    0x00000800    0x00000285
RK3288Loader_uboot_Apr212014_134842.bin    0x00001000    0x0004694E
rk3288-3.10-uboot-data1G.parameter.txt      0x00048000    0x000005A1
Image/misc.img      0x00048800    0x0000C000
Image/kernel.img    0x00055000    0x00578E3C
Image/resource.img0x005CE000    0x0001C400
Image/boot.img      0x005EA800    0x0011F6CF
Image/recovery.img      0x0070A000    0x0040F6AE
Image/system.img    0x00B19800    0x180EF000
RESERVED    0x00000000    0x00000000
update-script    0x18C09000    0x000003A5
recover-script  0x18C09800    0x0000010A
UnPack OK!
```

查看 update 目录下的文件树

```
$ cd update/
$ tree
.
├── Image
│   ├── boot.img
│   ├── kernel.img
│   ├── misc.img
│   ├── recovery.img
│   ├── resource.img
│   └── system.img
├── package-file
├── recover-script
├── RESERVED
├── rk3288-3.10-uboot-data1G.parameter.txt
├── RK3288Loader_uboot_Apr212014_134842.bin
└── update-script
```

这样，固件就解包成功了，下面就开始定制吧。

2.2.5 定制

定制 system.img

system.img 是个 ext4 文件系统格式的映像文件，可以直接挂载到系统进行修改：

```
sudo mkdir -p /mnt/system
sudo mount -o loop Image/system.img /mnt/system
cd /mnt/system
# 修改里面的东西，注意剩余空间，不能添加太多的 APK
# 修改完毕，要卸载
cd /
sudo umount /mnt/system
```

注意，该 system.img 的剩余空间基本为 0, 如果不是删除文件，就需要对 system.img 进行扩容，并根据最后的文件大小，相应地调整 parameter 文件里的分区设置。

以下是如何扩展空间的示例，在扩展前，先运行 mount 来查看系统挂载情况，确保 system.img 已经卸载：

```
# 增加 128M 的空间
dd if=/dev/zero bs=1M count=128 >> Image/system.img
# 扩展文件系统信息
e2fsck -f Image/system.img
resize2fs Image/system.img
```

2.2.6 打包

首先要检查一下 system.img 的大小，对照 parameter 文件的分区情况(可参考文档 Parameter 文件格式)，作必要的大小调整。例如，rk3288-3.10-uboot-data1G.parameter.txt 文件里的 system 分区大小，可以找到 CMDLINE 一行，然后找到 system 字符串：

```
0x00180000@0x00092000(system)
```

@ 前面就是分区的大小，单位是 512 字节，这样该 system 分区的大小就是：

```
$ echo $(( 0x00180000 * 512 / 1024 / 1024))M
768M
```

只要 system.img 的大小不超过 768M，parameter 文件就不用更改。

如果分区不用更改，可以直接用烧写工具将新的 system.img 烧写到开发板的 system 分区上做试验。否则，需要制作新固件并烧写后再行测试。

以下是打包成统一固件 update.img 所需要的步骤：

合成 update.img ：

```
# 当前的目录仍然为 update/ ，内有 package-file, package-file 所列的文件均存在
# 将参数文件拷贝一份到 paramter, 因为 afptool 默认要用到
$ cp rk3288-3.10-uboot-data1G.parameter.txt parameter
$ afptool -pack . ../update_new.img
```

```
------ PACKAGE ------
Add file: ./package-file
Add file: ./RK3288Loader_uboot_Apr212014_134842.bin
Add file: ./rk3288-3.10-uboot-data1G.parameter.txt
Add file: ./Image/misc.img
Add file: ./Image/kernel.img
Add file: ./Image/resource.img
Add file: ./Image/boot.img
Add file: ./Image/recovery.img
Add file: ./Image/system.img
Add file: ./RESERVED
Add file: ./update-script
Add file: ./recover-script
Add CRC...
------ OK ------
Pack OK!
```

合成 release_update.img ：

```
$ img_maker -rk32 loader.img update_new.img release_update_new.img
generate image...
append md5sum...
success!
```

release_update_new.img 即为最终生成的可烧写的统一固件文件。

2.2.7 常见问题

2.2.7.1 固件的版本在哪设置

在 parameter 文件中找到下行并修改即可，注意版本号为数字，中间两个点号不能省略。

```
FIRMWARE_VER:4.4.2
```

2.3 ADB 使用

2.3.1 前言

adb，全称 Android Debug Bridge，是 Android 的命令行调试工具，可以完成多种功能，如跟踪系统日志，上传下载文件，安装应用等。

2.3.2 准备连接

在开发板上进入选项->开发人员选项，勾上 "USB 调试" 选项。
用 Micro USB OTG 线连接设备和主机。

2.3.3 Windows 下的 ADB 安装

首先参照安装 RK USB 驱动一节安装好驱动。然后
到 http://adbshell.com/download/download-adb-for-windows.html 下载 adb.zip，解压
到 C:\adb 以方便调用。
打开命令行窗口，输入：
cd C:\adb
adb shell
如果一切正常，就可以进入 adb shell，在设备上面运行命令。

2.3.4 Ubuntu 下的 ADB 安装

安装 adb 工具：
sudo apt-get install android-tools-adb
加入设备标识：
mkdir -p ~/.android
vi ~/.android/adb_usb.ini
添加以下一行
0x2207
加入 udev 规则：
sudo vi /etc/udev/rules.d/51-android.rules
# 添加以下一行：
SUBSYSTEM=="usb", ATTR{idVendor}=="2207", MODE="0666"

重新插拔 USB 线，或运行以下命令，让 udev 规则生效：
sudo udevadm control --reload-rules
sudo udevadm trigger
重新启动 adb 服务器
sudo adb kill-server
adb start-server

2.3.5 常用 ADB 命令

2.3.5.1 连接管理

列出所有连接设备及其序列号
adb devices
如果有多个连接设备，则需要使用序列号来区分：
export ANDROID_SERIAL=<设备序列号>
adb shell ls
可以通过网络来连接 adb：
让设备端的 adbd 重启，并在 TCP 端口 5555 处监听
adb tcpip 5555
此时可以断开 USB 连接
远程连接设备，设备的 IP 地址是 192.168.1.100
adb connect 192.168.1.100:5555
断开连接
adb disconnect 192.168.1.100:5555

2.3.5.2 调试

2.3.5.2.1 获取系统日志 adb logcat

用法：
adb logcat [选项] [应用标签]
示例：
查看全部日志
adb logcat
仅查看部分日志
adb logcat -s WifiStateMachine StateMachine

2.3.5.2.2 运行命令 adb shell

2.3.5.2.3 获取详细运行信息 adb bugreport

adb bugreport 用于错误报告，里面包含大量有用的信息。
示例：
adb bugreport
保存到本地，方便用编辑器查看
adb bugreport >bugreport.txt

2.3.5.3 应用管理

2.3.5.3.1 安装应用 adb install

用法：
adb install [选项] 应用包.apk
选项包括：
-l forward-lock
-r 重新安装应用，保留原先数据
-s 安装到 SD 卡上，而不是内部存储
示例：
安装 facebook.apk
adb install facebook.apk
升级 twitter.apk
adb install -r twitter.apk
如果安装成功，工具会返回成功提示 "Success"；失败的话，一般是以下几种情况
INSTALL_FAILED_ALREADY_EXISTS: 此时需要用 -r 参数来重新安装。
INSTALL_FAILED_SIGNATURE_ERROR: 应用的签名不一致，可能是发布版和调试版签名不同所致。如果确认 APK 文件签名正常，可以用 adb uninstall 命令先卸载旧的应用，然后再安装。
INSTALL_FAILED_INSUFFICIENT_STORAGE: 存储空间不足，需要检查设备存储情况。

2.3.5.3.2 卸载应用 adb uninstall

用法：
adb uninstall 应用包名称
示例：

adb uninstall com.android.chrome
应用包名称可以用以下命令列出：
adb shell pm list packages -f
运行结果是：
...
package:/system/app/Bluetooth.apk=com.android.bluetooth
...
前面是 apk 文件，后面则是对应的包名称。

2.3.5.4 命令行帮助信息 adb help

Android Debug Bridge version 1.0.31

```
 -a                            - directs adb to listen on all interfaces for a connection
 -d                            - directs command to the only connected USB device
                                 returns an error if more than one USB device is present.
 -e                            - directs command to the only running emulator.
                                 returns an error if more than one emulator is running.
 -s <specific device>          - directs command to the device or emulator with the given
                                 serial number or qualifier. Overrides ANDROID_SERIAL
                                 environment variable.
 -p <product name or path>     - simple product name like 'sooner', or
                                 a relative/absolute path to a product
                                 out directory like 'out/target/product/sooner'.
                                 If -p is not specified, the ANDROID_PRODUCT_OUT
                                 environment variable is used, which must
                                 be an absolute path.
 -H                             - Name of adb server host (default: localhost)
 -P                             - Port of adb server (default: 5037)
 devices [-l]                   - list all connected devices('-l' will also list device qualifiers)
 connect <host>[:<port>]        - connect to a device via TCP/IP
                                 Port 5555 is used by default if no port number is specified.
 disconnect [<host>[:<port>]]   - disconnect from a TCP/IP device.
                                 Port 5555 is used by default if no port number is specified.
                                 Using this command with no additional arguments
                                 will disconnect from all connected TCP/IP devices.

device commands:
  adb push [-p] <local> <remote>
                                - copy file/dir to device ('-p' to display the transfer progress)
  adb pull [-p] [-a] <remote> [<local>]
```

```
                                - copy file/dir from device
                                  ('-p' to display the transfer progress)
                                  ('-a' means copy timestamp and mode)
  adb sync [ <directory> ]      - copy host->device only if changed
                                  (-l means list but don't copy)
                                  (see 'adb help all')
  adb shell                     - run remote shell interactively
  adb shell <command>           - run remote shell command
  adb emu <command>             - run emulator console command
  adb logcat [ <filter-spec> ]  - View device log
  adb forward --list            - list all forward socket connections.
                                  the format is a list of lines with the following format:
                                  <serial> " " <local> " " <remote> "\n"
  adb forward <local> <remote>  - forward socket connections
                                      forward specs are one of:
                                      tcp:<port>
                                      localabstract:<unix domain socket name>
                                      localreserved:<unix domain socket name>
                                      localfilesystem:<unix domain socket name>
                                      dev:<character device name>
                                      jdwp:<process pid> (remote only)
  adb forward --no-rebind <local> <remote>
                                - same as 'adb forward <local> <remote>' but fails
                                  if <local> is already forwarded
  adb forward --remove <local>  - remove a specific forward socket connection
  adb forward --remove-all      - remove all forward socket connections
  adb jdwp                      - list PIDs of processes hosting a JDWP transport
  adb install [-l] [-r] [-d] [-s] [--algo <algorithm name> --key <hex-encoded key> --iv
<hex-encoded iv>] <file>        - push this package file to the device and install it
                                  ('-l' means forward-lock the app)
                                  ('-r' means reinstall the app, keeping its data)
                                  ('-d' means allow version code downgrade)
                                  ('-s' means install on SD card instead of internal storage)
                                ('--algo', '--key', and '--iv' mean the file is encrypted already)
  adb uninstall [-k] <package>  - remove this app package from the device
                                  ('-k' means keep the data and cache directories)
  adb bugreport                 - return all information from the device
                                  that should be included in a bug report.

  adb backup [-f <file>] [-apk|-noapk] [-obb|-noobb] [-shared|-noshared] [-all]
[-system|-nosystem] [<packages...>]
```

- write an archive of the device's data to <file>.
If no -f option is supplied then the data is written
to "backup.ab" in the current directory.
(-apk|-noapk enable/disable backup of the .apks
themselves in the archive; the default is noapk.)
(-obb|-noobb enable/disable backup of any installed apk
expansion (aka .obb) files associated with each
application; the default is noobb.)
(-shared|-noshared enable/disable backup of the device's
shared storage / SD card contents; the default is noshared.)
(-all means to back up all installed applications)
(-system|-nosystem toggles whether -all automatically
includes system applications; the default is to include
system apps)
(<packages...> is the list of applications to be backed up.
If he -all or -shared flags are passed, then the package
list is optional. Applications explicitly given on the
command line will be included even if -nosystem
would ordinarily cause them to be omitted.)

adb restore <file> - restore device contents from the <file> backup archive

adb help - show this help message
adb version - show version num

scripting:
adb wait-for-device - block until device is online
adb start-server - ensure that there is a server running
adb kill-server - kill the server if it is running
adb get-state - prints: offline | bootloader | device
adb get-serialno - prints: <serial-number>
adb get-devpath - prints: <device-path>
adb status-window - continuously print device status for a specified device
adb remount - remounts the /system partition on the device read-write
adb reboot [bootloader|recovery]
- reboots the device, optionally into the bootloader or recovery
program
adb reboot-bootloader - reboots the device into the bootloader
adb root - restarts the adbd daemon with root permissions
adb usb - restarts the adbd daemon listening on USB
adb tcpip <port> - restarts the adbd daemon listening on TCP on the specified port

networking:
adb ppp <tty> [parameters] - Run PPP over USB.
Note: you should not automatically start a PPP connection.
<tty> refers to the tty for PPP stream. Eg. dev:/dev/omap_csmi_tty1
[parameters] - Eg. defaultroute debug dump local notty usepeerdns

adb sync notes: adb sync [<directory>]
<localdir> can be interpreted in several ways:

- If <directory> is not specified, both /system and /data partitions will be updated.

- If it is "system" or "data", only the corresponding partition is updated.

environmental variables:

ADB_TRACE	- Print debug information. A comma separated list of the following values 1 or all, adb, sockets, packets, rwx, usb, sync, sysdeps, transport, jdwp
ANDROID_SERIAL	- The serial number to connect to. -s takes priority over this if given.
ANDROID_LOG_TAGS	- When used with the logcat option, only these debug tags are printed.

2.4 SD 卡启动 Android

2.4.1 所需工具

- Firefly 的 SD 卡启动 Android 固件包
- 瑞芯微创建升级磁盘工具
- TF 卡（空间 4G 及以上，读写速度：C10）
- TF 卡读卡器
- windows PC

2.4.2 步骤

1) 下载 Firefly 的 SD 卡启动 Android 固件包及瑞芯微创建升级磁盘工具，下载地址为：百度网盘 或 Google drive
2) 解压下载的文件，提取出 Firefly 的 SD 卡启动 Android 固件包和瑞芯微创建升级磁盘工具。
3) 把 TF 卡插入读卡器，把读卡器插入 windows 系统的 PC。
4) 运行 SD_Firmware_Tool.exe，启动瑞芯微创建升级磁盘工具。
5) 根据工具的提示：选中 TF 卡，选择“SD 启动”，选中刚下载的 Android 固件包，点“开始创建”（如下图 2-4-1 所示）。
6) 开始创建时，工具会提醒你 TF 卡原来的数据会丢失，请选择“是”（如下图 2-4-2 所示）。
7) 经过一段时间的等待，当显示创建完成时，退出 TF 卡，并把 TF 卡插进 Firefly 开发板 TF 卡槽。
8) 给 Firefly 开发板接上电源，此时启动的就是 TF 卡上的 Android 系统。

图 2-4-1 瑞芯微创建升级磁盘工具

图 2-4-2 瑞芯微创建升级磁盘工具清除数据提醒

第 3 章 Linux 开发

3.1 编译内核

3.1.1 准备工作

3.1.1.1 安装开发包

安装开发包：
sudo apt-get install build-essential lzop libncurses5-dev libssl-dev
# 如果使用的是 64 位的 Ubuntu，还需要安装：
sudo apt-get install libc6:i386

3.1.1.2 安装 mkbootimg 工具

git clone https://github.com/neo-technologies/rockchip-mkbootimg.git
cd rockchip-mkbootimg
make && sudo make install

3.1.1.3 获取内核源码和安装交叉编译工具链

如果已经下载 Firefly-RK3288 Android SDK，内核源码和交叉编译工具链分别在 SDK/kernel 和 SDK/prebuilts 目录里，无需额外下载，请跳到下一步。

如果没有下载 SDK，则需要下载内核源码及 Android 的 arm-eabi-4.6 交叉编译工具链。

下载内核源码：
git clone https://bitbucket.org/T-Firefly/firefly-rk3288-kernel.git
注意：这其实就是 SDK 里的内核源码，为方便仅需要下载内核的用户，特意提取出来成为独立的源码仓库。
Android 的 arm-eabi-4.6 交叉编译工具链，可以看其它 Android SDK 的 prebuilts/gcc/linux-x86/arm/arm-eabi-4.6 目录是否存在，有则可以重用，没有则需要到这里下载，并解压。

3.1.2 编译内核

3.1.2.1 编译内核映像

如果不是在 SDK 里编译内核，则需要先指定 ARCH 和 CROSS_COMPILE：
export ARCH=arm
export CROSS_COMPILE=/path/to/prebuilts/gcc/linux-x86/arm/arm-eabi-4.6/bin/arm-eabi-
在内核源码目录里执行：
make firefly-rk3288-linux_defconfig
make -j8 firefly-rk3288.img
注意，如果是 beta 版的开发板，请将 firefly-rk3288.img 替换成 firefly-rk3288_beta.img 。

3.1.2.2 编译内核模块

在内核源码目录里执行：
make modules
mkdir modules_insatll
make INSTALL_MOD_PATH=./modules_install modules_install
内核模块是需要拷到根文件系统中即可：
rsync -av ./modules_install/ /path/to/your/rfs/lib/modules/
也可以远程拷贝到开发板的根文件系统中，这需要开发板可以通过 ssh 远程连接：
rsync -av ./modules_install/ root@开发板 IP:/lib/modules/
最后清理一下模块安装目录（该目录含有链接，会影响 SDK 的编译）：
rm -rf ./modules_install

3.1.3 创建 boot.img

3.1.3.1 创建内存盘

内核启动时会加载内存盘作为初始的根文件系统，再加载实际的根存储设备，最后切换过去。因为开发板使用的是 eMMC 存储，不需要特别的驱动，因此实际上可以跳过此步。但内存盘可以做得非常灵活和强大，例如可以做多系统启动。
git clone https://github.com/TeeFirefly/initrd.git
make -C initrd

3.1.3.2 打包内核和内存盘

将 kernel 和 initrd 打包成 boot.img：
mkbootimg --kernel arch/arm/boot/zImage --ramdisk initrd.img -o boot.img

3.1.4 修改 parameter 文件

Linux 的根文件系统（RFS）可能在不同的分区或存储设备上（eMMC、TF 卡或 U 盘），

所以需要在内核的参数中指定。修改 parameter 文件中的 CMDLINE 行：

```
CMDLINE:console=ttyFIQ0 ...
mtdparts=rk29xxnand:0x00002000@0x00002000(uboot),...,-@0x00394000(user)
```

根据实际情况加入以下之一（# 后是注释，不需要加入）：

```
root=/dev/block/mtd/by-name/linuxroot          # 名为 "linuxroot" 的 nand 分区
root=/dev/mmcblk0p1                # TF 卡的第一个分区
root=/dev/sda1                     # U 盘或 USB 硬盘的第一个分区
root=LABEL=linuxroot               # 卷标为 "linuxroot" 的分区，可以是任一存储设备
```

3.1.5 烧写到设备

参考《升级固件》，选择生成的 boot.img 和修改过的 parameter 文件，分别烧写到 "boot" 和 "parameter" 分区，则可完成内核的更新。如果还没有烧写根文件系统的，可以下载预先做好的镜像，或定制自己的根文件系统，并烧写到 parameter 文件指定的根分区中。

3.2 创建 Ubuntu 根文件系统

3.2.1 使用 miniroot 来创建并引导系统

miniroot 的主页是：http://androtab.info/miniroot/
miniroot 是个非常小巧的 shell 环境，用来安装和引导其它根文件系统，例如 Ubuntu, Gentoo, Arch Linux 等，这些系统可以在内核支持的存储设备的根或子目录上。这意味着我们能够从开发板的 eMMC Flash, 外置 TF 卡或 U 盘上安装多个系统，而且方便地切换系统，而不用修改并烧写 parameter 文件。

miniroot 需要使用串口线来调试，参见[串口调试]一文。另外在下载系统映像时需要使用以太网，当然，也可以预先下载到移动存储设备上。

3.2.1.1 准备

请先备份好开发板及相关存储设备上的数据，以免操作失误或其它不可预见的因素带来的数据丢失。

首先确保开发板已经烧写了可以正常工作的固件，然后下载以下映像文件：
misc.img
linux-boot-miniroot.img 0930 0809 beta （根据板子的版本选择）
如果开发板安装的是 Android 或双系统固件，则将 linux-boot-miniroot.img 写到 recovery 分区，misc.img 写到 misc 分区。
如果开发板安装的是 Linux 固件，则将 linux-boot-miniroot.img 写到 boot 分区。
miniroot 初次启动后，会进入 shell，在串口终端上可以见到提示符：
miniroot#
然后开始配置网络，如果是 DHCP 网络：
miniroot# udhcpc
否则就要手工配置网络参数（将 192.168.1.* 替换成实际使用的网络地址）：
miniroot# ip addr add 192.168.1.2/24 broadcast + dev eth0
miniroot# ip link set dev eth0 up
miniroot# ip route add default via 192.168.1.1
miniroot# echo nameserver 192.168.1.1 > /etc/resolv.conf
miniroot 支持从目录里启动，这就意味着根文件系统的放置位置很灵活，而且可以方便地支持多种 Linux 发行版启动。
下面用 TF 卡第一分区作为系统存储，创建 ext4 文件系统并挂载到 /mnt，ubuntu 将解压到 /mnt/ubuntu 下：
miniroot# mkfs.ext4 -E nodiscard /dev/mmcblk0p1
miniroot# mount /dev/mmcblk0p1 /mnt
一般需要保证此分区有 4G 以上的剩余空间。

3.2.1.2 下载和解压 ubuntu-core

ubuntu-core 是最小的根文件系统，在安装之后根据需要再设置桌面或服务器环境。
下载并解压到 /mnt :

```
miniroot# cd /mnt
miniroot# wget -P /mnt
http://cdimage.ubuntu.com/ubuntu-core/releases/14.04/release/ubuntu-core-14.04-core-armhf.tar.gz
miniroot# mkdir /mnt/ubuntu
miniroot# tar -xpzf /mnt/ubuntu-core-14.04-core-armhf.tar.gz -C /mnt/ubuntu
```

3.2.1.3 启动 Ubuntu

设置主机名称

```
miniroot# echo firefly > /mnt/ubuntu/etc/hostname
miniroot# sed -e 's/miniroot/firefly/' < /etc/hosts > /mnt/ubuntu/etc/hosts
```

设置串口控制台，自动以 root 用户登录：

```
miniroot# sed -e 's/tty1/ttyFIQ0/g' -e '/^exec/c exec /sbin/getty -a root -L 115200 ttyFIQ0 vt100' \
        < /mnt/ubuntu/etc/init/tty1.conf > /mnt/ubuntu/etc/init/ttyFIQ0.conf
```

如果不能使用串口控制台，可以新增用户帐户（帐户和密码均是 "ubuntu"）：

```
miniroot# chroot /mnt/ubuntu useradd -G sudo -m -s /bin/bash ubuntu
miniroot# echo ubuntu:ubuntu | chroot /mnt/ubuntu chpasswd
```

启动 Ubuntu

```
miniroot# boot /mnt:/ubuntu
```

提示：如果根设备没有挂载，可以将冒号前的挂载目录替换成根设备文件，miniroot 会自动挂载：

```
miniroot# boot /dev/mmcblk0p1:/ubuntu
```

3.2.1.4 初始配置

设置网络（DHCP）

```
root@ubuntu:~# echo auto eth0 > /etc/network/interfaces.d/eth0
root@ubuntu:~# echo iface eth0 inet dhcp >> /etc/network/interfaces.d/eth0
root@ubuntu:~# ln -fs ../run/resolvconf/resolv.conf /etc/resolv.conf
root@ubuntu:~# ifup eth0
```

更新软件包

```
root@ubuntu:~# apt-get update
root@ubuntu:~# apt-get dist-upgrade
```

重启

```
root@ubuntu:~# reboot
```

进入 miniroot，编辑环境变量，加入 ubuntu 的启动参数：

miniroot# editenv
boot=/dev/mmcblk0p1:/ubuntu
init=/sbin/init
autoboot=1
保存环境变量并重启
miniroot# saveenv
miniroot# reboot -f

3.2.1.5 安装软件包

安装 Lubuntu （LXDE）桌面环境：
root@ubuntu:~# apt-get install lubuntu-desktop

3.2.1.6 固化系统

将 TF 卡拔出，插入到主机系统，挂载到 /mnt 目录上。
查看根文件系统所需空间的大小：
sudo du -hs /mnt/ubuntu
视情况对 /mnt/ubuntu 目录进行清理，特别是一些日志目录、临时目录等。
生成空白磁盘映像文件，以生成 1G 大小的根文件系统磁盘映像文件为例：
cd /new/firmware/work/dir/
dd if=/dev/zero of=linuxroot.img bs=1M count=1024
格式化成 ext4 文件系统格式，卷标为 linuxroot
mkfs.ext4 -F -L linuxroot linuxroot.img
挂载，拷贝数据，然后卸载：
mount -o loop linuxroot.img /opt
cp -a /mnt/ubuntu/ /opt/
umount /opt
这样 linuxroot.img 就是最终的根文件系统映像文件了。

3.2.1.7 常见问题

3.2.1.7.1 如何恢复正常启动

往 misc 分区烧写 misc.img 后，开发板就会从 recovery 分区启动系统，要恢复回 boot 分区启动，有两种方法：

- 下载 misc_zero.img，然后烧写到 misc 分区
- 在开发板的 Linux shell 下运行：

sudo dd if=/dev/zero of=/dev/block/mtd/by-name/misc bs=16K count=count=3
sudo sync
sudo reboot

3.2.1.8 参考链接

Ubuntu 14.04 LTS with miniroot

3.3 Uboot 使用

3.3.1 前言

RK Uboot 是基于开源的 Uboot 进行开发的，UBoot 的工作模式有启动加载模式和下载模式。启动加载模式是 Uboot 的正常工作模式，嵌入式产品发布时，Uboot 都工作在此模式下，主要用于开机时把 FLASH 中的内核加载到 SDRAM 中，启动操作系统；下载模式主要用于将固件下载到 FLASH，开机时长按 recovery 键可进入下载模式。本文简单说明 Uboot 的使用，更多 Uboot 相关文档请看 SDK 下面的 RKDocs/common/uboot/RockChip_Uboot 开发文档 V1.0.pdf。

3.3.2 编译

编译 Uboot 与编译内核类似，编译前把默认配置写入.config，执行：make rk3288_config
如果需要修改相关选项，也可以用 make menuconfig
编译执行：make
编译后生成：u-boot/RK3288Loader_uboot_Vx.xx.xx.bin

3.3.3 烧录

打开烧录工具，板子接好 OTG 线，接通电源时按住 recovery 键，使用开发板进入 Uboot 的下载模式，在烧录工具中选择编译好的 Loader 文件，点击执行即可，如图 3 -3-1 所示。

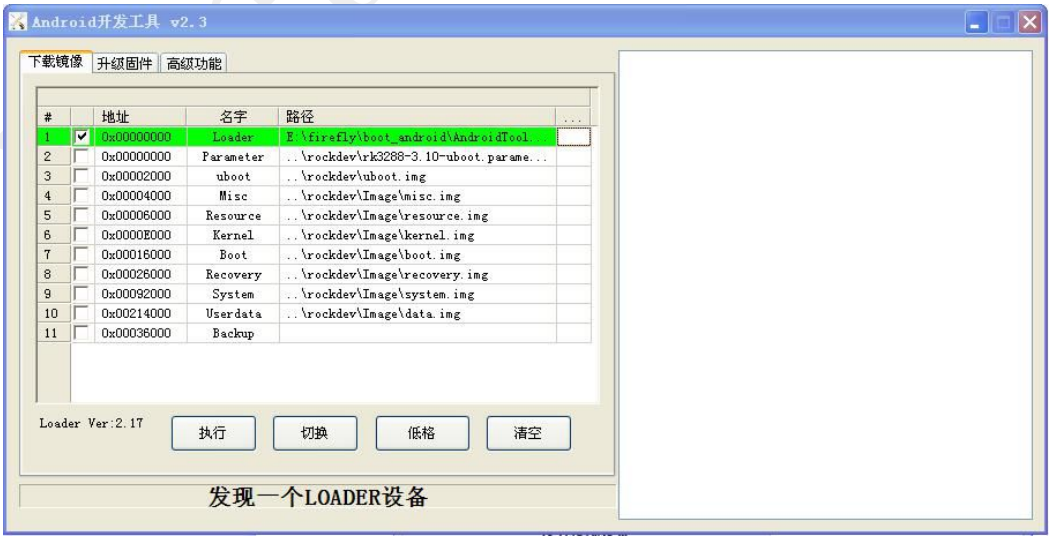


图 3-3-1 Uboot 烧写

3.3.4 确认是否正确烧写新的 Loader

如果你已经成功烧写你最新编译的 Loader，在开机的串口输出 log 中可以看到类似如下信息：#Boot ver: 2014-12-10#2.17
如果打印的时间及版本与你编译的一致，说明你成功更新了 Loader。

3.3.5 进入 Uboot 命令行模式

由于 Firefly 产品主要用于开发，所以我们默认设置开机时有 1 秒的倒计时，如果这时候在串口输入任意键即可进入 u-boot 命令行模式。 发布的产品是不需要进入 u-boot 命令行模式的，如果需要设置 u-boot 默认不进入命令行模式的，可以做如下修改：
在文件 U-boot/include/configs/rk32plat.h
/* mod it to enable console commands. */
#define CONFIG_BOOTDELAY 0
把宏 CONFIG_BOOTDELAY 改为 0 即默认不进入命令行模式。

3.3.6 二级 Loader

由于 Firefly 开发板没有用 nand flash，所以默认不用二级 Loader，这里只是简单说明二级 Loader。
RK3288Loader_uboot_Vx.xx.xx.bin 是一级 Loader 模式，只支持 emmc。
RK3288Loader_uboot_Vx.xx.xx.bin 和 uboot.img 组合是二级 loader 模式，同时支持 emmc 和 nand flash，二级 Loader 模式需要在 rk32xx.h 配置文件中添加以下定义：
#define CONFIG_SECOND_LEVEL_BOOTLOADER
添加后重新编译 Uboot，可以生成：
RK3288Loader_uboot_Vx.xx.xx.bin 和 uboot.img
把 RK3288Loader_uboot_Vx.xx.xx.bin 和 uboot.img 烧入板子即可。

第 4 章 驱动开发

4.1 ADC 使用

4.1.1 前言

Firefly-RK3288 开发板上的 AD 接口分为：高速 ADC 流接口 (High-speed ADC Stream Interface)、温度传感器 (Temperature Sensor)、逐次逼近 ADC (Successive Approximation Register)。本文主要介绍 ADC 的基本配置方法。

内核采用工业 I/O 子系统来控制 ADC，该子系统主要为 AD 转换或者 DA 转换的传感器设计。其相关数据结构以及配置方法如下：

4.1.2 数据结构

4.1.2.1 iio_channel 结构体

```
struct iio_channel {
    struct iio_dev *indio_dev;//工业 I/O 设备
    const struct iio_chan_spec *channel;//I/O 通道
    void *data;};
```

4.1.2.2 iio_dev 结构体

该结构体主要用于描述 IO 口所属的设备，其具体定义如下：

```
struct iio_dev {
    int                     id;
    int                     modes;
    int                     currentmode;
    struct device           dev;
    struct iio_event_interface    *event_interface;
    struct iio_buffer       *buffer;
    struct list_head        buffer_list;
    int                     scan_bytes;
    struct mutex            mlock;
    const unsigned long     *available_scan_masks;
    unsigned                masklength;
    const unsigned long     *active_scan_mask;
```

```
bool                    scan_timestamp;
unsigned                scan_index_timestamp;
struct iio_trigger      *trig;
struct iio_poll_func        *pollfunc;
struct iio_chan_spec const  *channels;
int                 num_channels;
struct list_head            channel_attr_list;
struct attribute_group      chan_attr_group;
const char              *name;
const struct iio_info       *info;
struct mutex                info_exist_lock;
const struct iio_buffer_setup_ops   *setup_ops;
struct cdev             chrdev;#define IIO_MAX_GROUPS 6
const struct attribute_group    *groups[IIO_MAX_GROUPS + 1];
int                 groupcounter;
unsigned long               flags;#if defined(CONFIG_DEBUG_FS)
struct dentry               *debugfs_dentry;
unsigned                cached_reg_addr;#endif};
```

4.1.2.3 iio_chan_spec 结构体

该结构体主要用于描述单个通道的属性，具体定义如下：

```
struct iio_chan_spec {
    enum iio_chan_type  type; //描述通道类型
    int             channel; //通道号
    int             channel2; //通道号
    unsigned long       address; //通道地址
    int             scan_index;
    struct {
        char    sign;
        u8  realbits;
        u8  storagebits;
        u8  shift;
        enum iio_endian endianness;
    } scan_type;
    long                info_mask;
    long                info_mask_separate;
    long                info_mask_shared_by_type;
    long                event_mask;
    const struct iio_chan_spec_ext_info *ext_info;
    const char          *extend_name;
```

```
    const char      *datasheet_name;
    unsigned        modified:1;
    unsigned        indexed:1;
    unsigned        output:1;
    unsigned        differential:1;};
```

4.1.3 配置步骤

4.1.3.1 配置 DTS 节点

Firefly ADC 的 DTS 节点在 kernel/arch/arm/boot/dts/rk3288.dtsi 文件中定义，如下所示：

```
 adc: adc@ff100000 {
        compatible = "rockchip,saradc";
        reg = <0xff100000 0x100>;
        interrupts = <GIC_SPI 36 IRQ_TYPE_LEVEL_HIGH>;
        #io-channel-cells = <1>;
        io-channel-ranges;
        rockchip,adc-vref = <1800>;
        clock-frequency = <1000000>;
        clocks = <&clk_saradc>, <&clk_gates7 1>;
        clock-names = "saradc", "pclk_saradc";
        status = "disabled";};
```

用户只需在 firefly-rk3288.dts 文件中添加通道定义，并将其 status 改为 "okay" 即可：

```
&adc {
    status = "okay";
adc_test{
        compatible = "rockchip,adc_test";
        io-channels = <&adc 0>;
    };};
```

4.1.3.2 在驱动文件中匹配 DTS 节点

在驱动文件中定义 of_device_id 结构体数组：

```
static const struct of_device_id of_XXX_match[] = {
    { .compatible = "rockchip,adc_test" },
    { /* Sentinel */ }};
```

将该结构体数组填充到要使用 ADC 的 platform_driver 中。

```
static struct platform_driver XXX_driver = {
    .probe          = ...,
    .remove         = ...,
    .driver         = {
```

```
        .name     = "..",
        .owner    = THIS_MODULE,#ifdef CONFIG_OF
        .of_match_table = of_XXX_match,#endif
    },  };
```

4.1.3.3 获取 AD 通道

struct iio_channel *chan; //定义 IIO 通道结构体
chan = iio_channel_get(&pdev->dev, NULL); //获取 IIO 通道结构体
注：iio_channel_get 通过 probe 函数传进来的参数 pdev 获取 IIO 通道结构体，probe 函数如下：
static int XXX_probe(struct platform_device *pdev);

4.1.3.4 读取 AD 采集到的原始数据

int val,ret;
ret = iio_read_channel_raw(chan, &val);
调用 iio_read_channel_raw 函数读取 AD 采集的原始数据并存入 val 中。

4.1.3.5 计算采集到的电压

使用标准电压将 AD 转换的值转换为用户所需要的电压值。其计算公式如下：
Vref / (2^n-1) = Vresult / raw
注：
Vref 为标准电压
n 为 AD 转换的位数
Vresult 为用户所需要的采集电压
raw 为 AD 采集的原始数据
例如，标准电压为 1.8V，AD 采集位数为 10 位，AD 采集到的原始数据为 568，则：
Vresult = (1800mv * 568) / 1023;

4.1.3.6 ADC 常用函数接口

struct iio_channel *iio_channel_get(struct device *dev, const char *consumer_channel);
功能：获取 iio 通道描述
参数：
dev: 使用该通道的设备描述指针
consumer_channel: 该设备所使用的 IIO 通道描述指针

void iio_channel_release(struct iio_channel *chan);
功能：释放 iio_channel_get 函数获取到的通道

参数：
chan：要被释放的通道描述指针

int iio_read_channel_raw(struct iio_channel *chan, int *val);
功能：读取 chan 通道 AD 采集的原始数据。
参数：
chan：要读取的采集通道指针
val：存放读取结果的指针

4.2 GPIO 使用

4.2.1 简介

GPIO，全称 General-Purpose Input/Output（通用输入输出），是一种软件运行期间能够动态配置和控制的通用引脚。

RK3288 有 9 组 GPIO bank： GPIO0，GPIO1, ..., GPIO8。每组又以 A0~A7, B0~B7, C0~C7, D0~D7 作为编号区分（不是所有 bank 都有全部编号，例如 GPIO5 就只有 B0~B7, C0~C3)。每个 GPIO 口除了通用输入输出功能外，还可能有其它复用功能，例如 GPIO5_B4，可以复用成以下功能之一：

spi0_clk
ts0_data4
uart4exp_ctsn

每个 GPIO 口的驱动电流、上下拉和重置后的初始状态都不尽相同，详细情况请参考《RK3288 规格书》中的 "RK3288 function IO description" 一章。

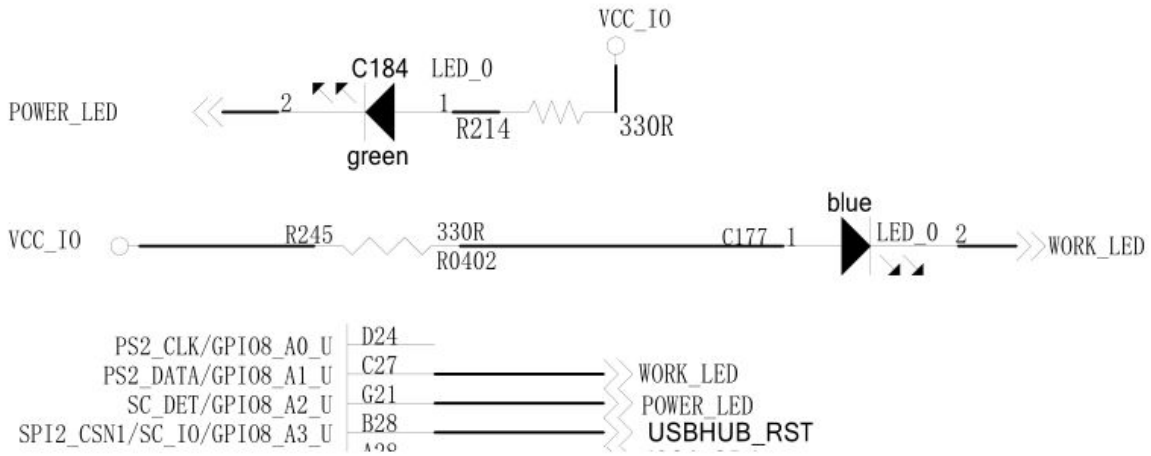
RK3288 的 GPIO 驱动是在以下 pinctrl 文件中实现的：

kernel/drivers/pinctrl/pinctrl-rockchip.c

其核心是填充 GPIO bank 的方法和参数，并调用 gpiochip_add 注册到内核中。

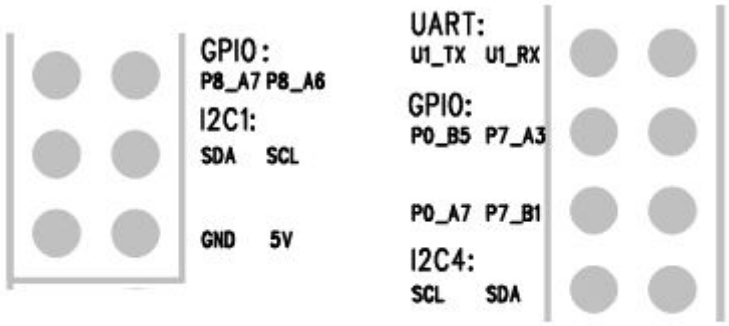
4.2.2 使用

开发板有两个电源 LED 灯是 GPIO 口控制的，分别是：



从电路图上看，GPIO 口输出低电平时灯亮，高电平时灯灭。

另外，扩展槽上引出了几个空闲的 GPIO 口，分别是：



J6 J10

这几个 GPIO 口可以自定义作输入、输出使用。

4.2.2.1 输入输出

下面以电源 LED 灯的驱动为例，讲述如何在内核编写代码控制 GPIO 口的输出。首先需要在 dts (Device Tree) 文件 firefly-rk3288.dts (0930 版) 或 firefly-rk3288_beta.dts (0809 版) 中增加驱动的资源描述：

```
firefly-led{
        compatible = "firefly,led";
        led-work = <&gpio8 GPIO_A2 GPIO_ACTIVE_LOW>;
        led-power = <&gpio8 GPIO_A1 GPIO_ACTIVE_LOW>;
        status = "okay";
};
```

这里定义了两颗 LED 灯的 GPIO 设置：
led-work GPIO8_A2 GPIO_ACTIVE_LOW
led-power GPIO8_A1 GPIO_ACTIVE_LOW
GPIO_ACTIVE_LOW 表示低电平有效（灯亮），如果是高电平有效，需要替换为 GPIO_ACTIVE_HIGH 。
之后在驱动程序中加入对 GPIO 口的申请和控制则可：

```
#ifdef CONFIG_OF#include <linux/of.h>#include <linux/of_gpio.h>#endif
 static int firefly_led_probe(struct platform_device *pdev){
        int ret = -1;
        int gpio, flag;
        struct device_node *led_node = pdev->dev.of_node;

        gpio = of_get_named_gpio_flags(led_node, "led-power", 0, &flag);
        if (!gpio_is_valid(gpio)){
                printk("invalid led-power: %d\n",gpio);
                return -1;
```

```
    }
    if (gpio_request(gpio, "led_power")) {
        printk("gpio %d request failed!\n",gpio);
        return ret;
    }
    led_info.power_gpio = gpio;
    led_info.power_enable_value = (flag == OF_GPIO_ACTIVE_LOW) ? 0 : 1;
    gpio_direction_output(led_info.power_gpio, !(led_info.power_enable_value));
...on_error:
    gpio_free(gpio);}
```

of_get_named_gpio_flags 从设备树中读取 led-power 的 GPIO 配置编号和标志，gpio_is_valid 判断该 GPIO 编号是否有效，gpio_request 则申请占用该 GPIO。如果初始化过程出错，需要调用 gpio_free 来释放之前申请过且成功的 GPIO 。

调用 gpio_direction_output 就可以设置输出高还是低电平，因为是 GPIO_ACTIVE_LOW ，如果要灯亮，需要写入 0 。

实际中如果要读出 GPIO，需要先设置成输入模式，然后再读取值：

```
int val;
gpio_direction_input(your_gpio);
val = gpio_get_value(your_gpio);
```

下面是常用的 GPIO API 定义：

```
#include <linux/gpio.h>#include <linux/of_gpio.h>
 enum of_gpio_flags {
      OF_GPIO_ACTIVE_LOW = 0x1,};
 int of_get_named_gpio_flags(struct device_node *np, const char *propname,
                    int index, enum of_gpio_flags *flags);
 int gpio_is_valid(int gpio);
 int gpio_request(unsigned gpio, const char *label);
 void gpio_free(unsigned gpio);
 int gpio_direction_input(int gpio);
 int gpio_direction_output(int gpio, int v)
```

4.2.2.2 复用

如何定义 GPIO 有哪些功能可以复用，在运行时又如何切换功能呢？以 I2C4 为例作简单的介绍。

查规格表可知，I2C4_SDA 与 GPIO7C1 的功能定义如表 4-2-1 所示。

表 4-2-1 I2C4_SDA 与 GPIO7C1 引脚功能定义

Pad#	func0	func1
I2C4_SDA/GPIO7_C1	gpio7c1	i2c4tp_sda
I2C4_SCL/GPIO7_C2	gpio7c2	i2c4tp_scl

在 /kernel/arch/arm/boot/dts/rk3288.dtsi 里有：

```
i2c4: i2c@ff160000 {
        compatible = "rockchip,rk30-i2c";
        reg = <0xff160000 0x1000>;
        interrupts = <GIC_SPI 64 IRQ_TYPE_LEVEL_HIGH>;
        #address-cells = <1>;
        #size-cells = <0>;
        pinctrl-names = "default", "gpio";
        pinctrl-0 = <&i2c4_sda &i2c4_scl>;
        pinctrl-1 = <&i2c4_gpio>;
        gpios = <&gpio7 GPIO_C1 GPIO_ACTIVE_LOW>, <&gpio7 GPIO_C2
GPIO_ACTIVE_LOW>;
        clocks = <&clk_gates6 15>;
        rockchip,check-idle = <1>;
        status = "disabled";
};
```

此处，跟复用控制相关的是 pinctrl- 开头的属性：

pinctrl-names 定义了状态名称列表： default (i2c 功能) 和 gpio 两种状态。

pinctrl-0 定义了状态 0 (即 default）时需要设置的 pinctrl: i2c4_sda 和 i2c4_scl

pinctrl-1 定义了状态 1 (即 gpio)时需要设置的 pinctrl: i2c4_gpio

这些 pinctrl 在 /kernel/arch/arm/boot/dts/rk3288-pinctrl.dtsi 中定义：

```
/ {
    pinctrl: pinctrl@ff770000 {
        compatible = "rockchip,rk3288-pinctrl";
        ...
        gpio7_i2c4 {
            i2c4_sda:i2c4-sda {
                rockchip,pins = <I2C4TP_SDA>;
                rockchip,pull = <VALUE_PULL_DISABLE>;
                rockchip,drive = <VALUE_DRV_DEFAULT>;
                //rockchip,tristate = <VALUE_TRI_DEFAULT>;
            };

            i2c4_scl:i2c4-scl {
                rockchip,pins = <I2C4TP_SCL>;
                rockchip,pull = <VALUE_PULL_DISABLE>;
                rockchip,drive = <VALUE_DRV_DEFAULT>;
                //rockchip,tristate = <VALUE_TRI_DEFAULT>;
            };

            i2c4_gpio: i2c4-gpio {
```

```
                    rockchip,pins = <FUNC_TO_GPIO(I2C4TP_SDA)>,
<FUNC_TO_GPIO(I2C4TP_SCL)>;
                    rockchip,drive = <VALUE_DRV_DEFAULT>;
                };
            };
            ...
        }
    }
```

I2C4TP_SDA, I2C4TP_SCL 的定义在
/kernel/arch/arm/boot/dts/include/dt-bindings/pinctrl/rockchip-rk3288.h 中：

```
#define GPIO7_C1 0x7c10#define I2C4TP_SDA 0x7c11
 #define GPIO7_C2 0x7c20#define I2C4TP_SCL 0x7c21
```

FUN_TO_GPIO 的定义在 /kernel/arch/arm/boot/dts/include/dt-bindings/pinctrl/rockchip.h 中：

```
#define FUNC_TO_GPIO(m)             ((m) & 0xfff0)
```

也就是说 FUNC_TO_GPIO(I2C4TP_SDA) == GPIO7_C1, FUNC_TO_GPIO(I2C4TP_SCL) == GPIO7_C2 。
像 0x7c11 这样的值是有编码规则的：

```
7 c1 1
| |   `- func
| `---- offset
`------ bank
```

0x7c11 就表示 GPIO7_C1 func1, 即 i2c4tp_sda 。
在复用时，如果选择了 "default" （即 i2c 功能),系统会应用 i2c4_sda 和 i2c4_scl 这两个 pinctrl，最终得将 GPIO7_C1 和 GPIO7_C2 两个针脚切换成对应的 i2c 功能；而如果选择了 "gpio" ，系统会应用 i2c4_gpio 这个 pinctrl，将 GPIO7_C1 和 GPIO7_C2 两个针脚还原为 GPIO 功能。
我们看看 i2c 的驱动程序 /kernel/drivers/i2c/busses/i2c-rockchip.c 是如何切换复用功能的：

```
static int rockchip_i2c_probe(struct platform_device *pdev){
    struct rockchip_i2c *i2c = NULL;
    struct resource *res;
    struct device_node *np = pdev->dev.of_node;
    int ret;// ...
        i2c->sda_gpio = of_get_gpio(np, 0);
        if (!gpio_is_valid(i2c->sda_gpio)) {
            dev_err(&pdev->dev, "sda gpio is invalid\n");
            return -EINVAL;
        }
        ret = devm_gpio_request(&pdev->dev, i2c->sda_gpio, dev_name(&i2c->adap.dev));
        if (ret) {
            dev_err(&pdev->dev, "failed to request sda gpio\n");
            return ret;
```

```
        }
        i2c->scl_gpio = of_get_gpio(np, 1);
        if (!gpio_is_valid(i2c->scl_gpio)) {
            dev_err(&pdev->dev, "scl gpio is invalid\n");
            return -EINVAL;
        }
        ret = devm_gpio_request(&pdev->dev, i2c->scl_gpio, dev_name(&i2c->adap.dev));
        if (ret) {
            dev_err(&pdev->dev, "failed to request scl gpio\n");
            return ret;
        }
        i2c->gpio_state = pinctrl_lookup_state(i2c->dev->pins->p, "gpio");
        if (IS_ERR(i2c->gpio_state)) {
            dev_err(&pdev->dev, "no gpio pinctrl state\n");
            return PTR_ERR(i2c->gpio_state);
        }
        pinctrl_select_state(i2c->dev->pins->p, i2c->gpio_state);
        gpio_direction_input(i2c->sda_gpio);
        gpio_direction_input(i2c->scl_gpio);
        pinctrl_select_state(i2c->dev->pins->p, i2c->dev->pins->default_state);// ...}
```

首先是调用 of_get_gpio 取出设备树中 i2c4 结点的 gpios 属于所定义的两个 gpio:

```
gpios = <&gpio7 GPIO_C1 GPIO_ACTIVE_LOW>, <&gpio7 GPIO_C2
GPIO_ACTIVE_LOW>;
```

然后是调用 devm_gpio_request 来申请 gpio，接着是调用 pinctrl_lookup_state 来查找“gpio” 状态，而默认状态 "default" 已经由框架保存到 i2c->dev-pins->default_state 中了。最后调用 pinctrl_select_state 来选择是 "default" 还是 "gpio" 功能。

下面是常用的复用 API 定义：

```
#include <linux/pinctrl/consumer.h>
 struct device {//...#ifdef CONFIG_PINCTRL
     struct dev_pin_info *pins;#endif//...};
 struct dev_pin_info {
       struct pinctrl *p;
       struct pinctrl_state *default_state;#ifdef CONFIG_PM
       struct pinctrl_state *sleep_state;
       struct pinctrl_state *idle_state;#endif};
 struct pinctrl_state * pinctrl_lookup_state(struct pinctrl *p, const char *name);
 int pinctrl_select_state(struct pinctrl *p, struct pinctrl_state *s);
```

4.3 I2C 使用

4.3.1 前言

Firefly-RK3288 开发板上有 6 个片上 I2C 控制器。本文主要描述如何在该开发板上配置 I2C。配置 I2C 可分为两大步骤：

- 定义和注册 I2C 设备
- 定义和注册 I2C 驱动

下面以配置 lt8641ex 为例。

4.3.2 定义和注册 I2C 设备

在注册 I2C 设备时，需要结构体 i2c_client 来描述 I2C 设备。然而在标准 Linux 中，用户只需要提供相应的 I2C 设备信息，Linux 就会根据所提供的信息构造 i2c_client 结构体。

用户所提供的 I2C 设备信息以节点的形式写到 dts 文件中，如下所示：

```
&i2c1 {
        status = "okay";
        lt8641ex@3f {
                compatible = "firefly,lt8641ex";
                gpio-sw = <&gpio7 GPIO_B2 GPIO_ACTIVE_LOW>;
                reg = <0x3f>;
        };
        rtc@51 {
                compatible = "nxp,pcf8563";
                reg = <0x51>;
        };
};
```

4.3.3 定义和注册 I2C 驱动

4.3.3.1 定义 I2C 驱动

在定义 I2C 驱动之前，用户首先要定义变量 of_device_id 和 i2c_device_id 。of_device_id 用于在驱动中调用 dts 文件中定义的设备信息，其定义如下所示：

```
static const struct of_device_id of_rk_lt8641ex_match[] = {
```

```
    { .compatible = "firefly,lt8641ex" },
    { /* Sentinel */ }};
```

定义变量 i2c_device_id：

```
static const struct i2c_device_id lt8641ex_id[] = {
    { lt8641ex, 0 },
    { }};
MODULE_DEVICE_TABLE(i2c, lt8641ex_id);
```

i2c_driver 如下所示：

```
static struct i2c_driver lt8641ex_device_driver = {
    .driver        = {
        .name     = "lt8641ex",
        .owner    = THIS_MODULE,
        .of_match_table = of_rk_lt8641ex_match,
    },
    .probe              = lt8641ex_probe,
    .remove             = lt8641ex_remove,
    .suspend            = lt8641ex_suspend,
    .resume             = lt8641ex_resume,
    .id_table           = lt8641ex_id,};
```

注：变量 id_table 指示该驱动所支持的设备。

4.3.3.2 注册 I2C 驱动

使用 i2c_add_driver 函数注册 I2C 驱动。

```
    i2c_add_driver(&lt8641ex_device_driver);
```

在调用 i2c_add_driver 注册 I2C 驱动时，会遍历 I2C 设备，如果该驱动支持所遍历到的设备，则会调用该驱动的 probe 函数。

4.3.3.3 通过 I2C 收发数据

在注册好 I2 C 驱动后，即可进行 I2C 通讯。

向从机发送信息

```
static int i2c_master_reg8_send(const struct i2c_client *client, const char reg, const char *buf, int count, int scl_rate){
    struct i2c_adapter *adap=client->adapter;
    struct i2c_msg msg;
    int ret;
    char *tx_buf = (char *)kzalloc(count + 1, GFP_KERNEL);
    if(!tx_buf)
        return -ENOMEM;
    tx_buf[0] = reg;
```

```
    memcpy(tx_buf+1, buf, count);

    msg.addr = client->addr;
    msg.flags = client->flags;
    msg.len = count + 1;
    msg.buf = (char *)tx_buf;
    msg.scl_rate = scl_rate;

    ret = i2c_transfer(adap, &msg, 1);
    kfree(tx_buf);
    return (ret == 1) ? count : ret;
}
```

向从机读取信息

```
static int i2c_master_reg8_recv(const struct i2c_client *client, const char reg, char *buf, int count,
int scl_rate){
    struct i2c_adapter *adap=client->adapter;
    struct i2c_msg msgs[2];
    int ret;
    char reg_buf = reg;

    msgs[0].addr = client->addr;
    msgs[0].flags = client->flags;
    msgs[0].len = 1;
    msgs[0].buf = &reg_buf;
    msgs[0].scl_rate = scl_rate;

    msgs[1].addr = client->addr;
    msgs[1].flags = client->flags | I2C_M_RD;
    msgs[1].len = count;
    msgs[1].buf = (char *)buf;
    msgs[1].scl_rate = scl_rate;

    ret = i2c_transfer(adap, msgs, 2);

    return (ret == 2)? count : ret;}
```

注:

msgs[0] 是要向从机发送的信息，告诉从机主机要读取信息。

msgs[1] 是主机向从机读取到的信息。

至此，主机可以使用函数 i2c_master_reg8_send 和 i2c_master_reg8_recv 和从机进行通讯。

实际通讯示例

例如主机和 LT8641EX 通讯，主机向 LT8641EX 发送信息，设置 LT8641EX 使用通道 1：

```
int channel=1;
i2c_master_reg8_send(g_lt8641ex->client, 0x00, &channel,1, 100000);
```

注：通道寄存器的地址为 0x00。

主机向从机 LT8641EX 读取当前使用的通道：

```
u8 ch = 0xfe;
i2c_master_reg8_recv(g_lt8641ex->client, 0x00, &ch,1, 100000);
```

注：ch 用于保存读取到的信息。

4.4 IR 使用

4.4.1 红外遥控配置

Firefly-RK3288 开发板上使用红外收发传感器 IR (在 USB OTG 接口和音频接口之间)实现遥控功能。本文主要描述在开发板上如何配置红外遥控器。

其配置步骤可分为两个部分：

- 修改内核驱动：内核空间修改，Linux 和 Android 都要修改这部分的内容。
- 修改键值映射：用户空间修改（仅限 Android 系统）。

4.4.2 内核驱动

在 Linux 内核中，IR 驱动仅支持 NEC 编码格式。以下是在内核中配置红外遥控的方法。所涉及到的文件：kernel/drivers/input/remotectl/rk_pwm_remotectl.c

4.4.2.1 定义相关数据结构

以下是定义数据结构的步骤：

添加键值表结构体数组：

```
static struct rkxx_remote_key_table remote_key_table_r66[12] = {
    {0xeb, KEY_POWER},          // Power
    // Control
    {0xa3, 250},                // Settings
    {0xec, KEY_MENU},           // Menu
    {0xfc, KEY_UP},             // Up
    {0xfd, KEY_DOWN},           // Down
    {0xf1, KEY_LEFT},           // Left
    {0xe5, KEY_RIGHT},          // Right
    {0xf8, KEY_REPLY},          // Ok
    {0xb7, KEY_HOME},           // Home
    {0xfe, KEY_BACK},           // Back
    // Vol
    {0xa7, KEY_VOLUMEDOWN},     // Vol-
    {0xf4, KEY_VOLUMEUP},       // Vol+};
```

注：第一列为键值，第二列为要响应的按键码。

添加按键结构体数组

```
static struct rkxx_remotectl_button remotectl_button[] ={
```

```
    //...
    {
        .usercode = 0xff00,/* need to get the usercode in next step */
        .nbuttons =    12,/* number of buttons */
        .key_table = &remote_key_table_r66[0],/* key table */
    },
    // ...};
```

注：
usercode 是用户码，每个 IR 都有一个对应的用户码；
nbuttons 是遥控按键个数；
key_table 是在第一步中添加的键值表结构体数组的地址。

4.4.2.2 如何获取用户码和 IR 键值

在 remotectl_do_something 函数中获取用户码和键值：

```
  case RMC_USERCODE:
        {
            //ddata->scanData <<= 1;
            //ddata->count ++;
            if ((RK_PWM_TIME_BIT1_MIN < ddata->period) && (ddata->period < RK_PWM_TIME_BIT1_MAX)){
                ddata->scanData |= (0x01<<ddata->count);;
            }
            ddata->count ++;
            if (ddata->count == 0x10){//16 bit user code
                DBG_CODE("GET USERCODE=0x%x\n",((ddata->scanData) & 0xffff));
                if (remotectl_keybdNum_lookup(ddata)){
                    ddata->state = RMC_GETDATA;
                    ddata->scanData = 0;
                    ddata->count = 0;
                }else{                          //user code error
                    ddata->state = RMC_PRELOAD;
                }
            }
        }
```

注：用户可以使用 DBG_CODE() 函数打印用户码。

向 remotectl_button 数组添加用户码和键值：

```
 case RMC_GETDATA:
        {
            //ddata->count ++;
            //ddata->scanData <<= 1;
```

```
                #ifdef CONFIG_FIREFLY_POWER_LED
                mod_timer(&timer_led,jiffies + msecs_to_jiffies(50));
                remotectl_led_ctrl(0);
                #endif
                if ((RK_PWM_TIME_BIT1_MIN < ddata->period) && (ddata->period < RK_PWM_TIME_BIT1_MAX)){
                    ddata->scanData |= (0x01<<ddata->count);
                }
                ddata->count ++;
                if (ddata->count == 0x10){
                    DBG_CODE("RMC_GETDATA=%x\n",(ddata->scanData>>8));
                    if ((ddata->scanData&0x0ff) == ((~ddata->scanData >> 8)&0x0ff)){
                        if (remotectl_keycode_lookup(ddata)){
                            ddata->press = 1;
                            ...
                        }
                        ...
                    }
                    ...
                }
            }
```

注：用户可以使用 DBG_CODE() 函数打印键值。

4.4.2.3 将 IR 驱动编译进内核

将 IR 驱动编译进内核的步骤如下所示：

(1)、向配置文件 drivers/input/remotectl/Kconfig 中添加如下配置：

```
config RK_REMOTECTL_PWM
    bool "rkxx remoctrl pwm0 capture"
default n
```

(2)、修改 drivers/input/remotectl 路径下的 Makefile,添加如下编译选项：

```
obj-$(CONFIG_RK_REMOTECTL_PWM)          += rk_pwm_remotectl.o
```

(3)、在 kernel 路径下使用 make menuconfig ，按照如下方法将 IR 驱动选中。

```
Device Drivers
  --->Input device support
  ----->   [*]    rkxx remotectl
  ------->[*]    rkxx remoctrl pwm0 capture.
```

保存后，执行 make 命令即可将该驱动编进内核。

4.4.3 Android 键值映射

文件 /system/usr/keylayout/rkxx-remotectl.kl 用于将 Linux 层获取的键值映射到 Android 上对应的键值。用户可以添加或者修改该文件的内容以实现不同的键值映射。
该文件内容如下所示：

```
key 28     ENTER
key 116    POWER              WAKE
key 158    BACK
key 139    MENU
key 217    SEARCH
key 232    DPAD_CENTER
key 108    DPAD_DOWN
key 103    DPAD_UP
key 102    HOME
key 105    DPAD_LEFT
key 106    DPAD_RIGHT
key 115    VOLUME_UP
key 114    VOLUME_DOWN
key 143    NOTIFICATION       WAKE
key 113    VOLUME_MUTE
key 388    TV_KEYMOUSE_MODE_SWITCH
key 400    TV_MEDIA_MULT_BACKWARD
key 401    TV_MEDIA_MULT_FORWARD
key 402    TV_MEDIA_PLAY_PAUSE
key 64     TV_MEDIA_PLAY
key 65     TV_MEDIA_PAUSE
key 66     TV_MEDIA_STOP
```

注：通过 adb 修改该文件重启后即可生效。

4.5 LED 使用

4.5.1 前言

Firefly-RK3288 开发板上有 2 个 LED 灯，如表 4-5-1 所示。

表 4-5-1 LED 引脚

LED	GPIO ref.	GPIO number
Blue	GPIO8_A1	257
Yellow	GPIO8_A2	258

可通过使用 LED 设备子系统或者直接操作 GPIO 控制该 LED。

4.5.2 以设备的方式控制 LED

标准的 Linux 专门为 LED 设备定义了 LED 子系统。 在 Firefly-RK3288 开发板中的两个 LED 均以设备的形式被定义。用户可以通过 /sys/class/leds/ 目录控制这两个 LED。更详细的说明请参考 leds-class.txt 。
开发板上的 LED 的默认状态为：
Blue: 系统上电时打开
Yellow：用户自定义
用户可以通过 echo 向其 trigger 属性输入命令控制每一个 LED：

```
root@firefly:~ # echo none >/sys/class/leds/firefly:blue:power/trigger
root@firefly:~ # echo default-on >/sys/class/leds/firefly:blue:power/trigger
```

用户还可以使用 cat 命令获取 trigger 的可用值：

```
root@firefly:~ # cat /sys/class/leds/firefly:blue:power/trigger
none [ir-power-click] test_ac-online test_battery-charging-or-full test_battery-charging
test_battery-full test_battery-charging-blink-full-solid test_usb-online mmc0 mmc1 mmc2
backlight default-on rfkill0 rfkill1 rfkill2
```

4.5.3 在内核中操作 LED

在内核中操作 LED 的步骤如下：
1、在 dts 文件中定义 LED 节点“leds”
在 kernel/arch/arm/boot/dts/firefly-rk3288.dts 文件中定义 LED 节点，具体定义如下：

```
 leds {
```

```
        compatible = "gpio-leds";
        power {
            label = "firefly:blue:power";
            linux,default-trigger = "ir-power-click";
            default-state = "on";
            gpios = <&gpio8 GPIO_A1 GPIO_ACTIVE_LOW>;
        };
        user{
            label = "firefly:yellow:user";
            linux,default-trigger = "ir-user-click";
            default-state = "off";
            gpios = <&gpio8 GPIO_A2 GPIO_ACTIVE_LOW>;
        };
  };
```

注意：compatible 的值要跟 drivers/leds/leds-gpio.c 中的 .compatible 的值要保持一致。

2、在驱动文件包含头文件

```
#include <linux/leds.h>
```

3、在驱动文件中控制 LED。

（1）、定义 LED 触发器

```
DEFINE_LED_TRIGGER(ledtrig_ir_click);
```

（2）、注册该触发器

```
led_trigger_register_simple("ir-power-click", &ledtrig_ir_click);
```

（3）、控制 LED 的亮灭。

```
led_trigger_event(ledtrig_ir_click, LED_FULL); //亮
led_trigger_event(ledtrig_ir_click, LED_OFF); //灭
```

4.6 PWM 使用

4.6.1 前言

Firefly-RK3288 开发板上有 4 路 PWM 输出，分别为 PWM0～PWM3。本章主要描述如何配置 PWM。

RK3288 的 PWM 驱动为：kernel/drivers/pwm/pwm-rockchip.c

4.6.2 数据结构

4.6.2.1 pwm_device 结构体

```
struct pwm_device {
    const char          *label;
    unsigned long           flags;
    unsigned int            hwpwm;
    unsigned int            pwm;//pwm 通道
    struct pwm_chip         *chip;
    void                    *chip_data;
    unsigned int            period; /* in nanoseconds */};
```

4.6.2.2 pwm_chip 结构体

该结构体是抽象的 PWM 控制器。

```
struct pwm_chip {
    struct device           *dev; //提供 PWM 的设备
    struct list_head        list; //内部使用的节点列表
    const struct pwm_ops        *ops; //该 PWM 控制器的回调函数
    int             base; //该设备所控制的第一个 PWM 的号码
    unsigned int            npwm; //该设备所控制的 PWM 数
    struct pwm_device       *pwms;
    struct pwm_device * (*of_xlate)(struct pwm_chip *pc,
                                    const struct of_phandle_args *args);
    unsigned int            of_pwm_n_cells;
    bool                    can_sleep;};
```

4.6.3 配置步骤

配置 PWM 主要有以下三大步骤：配置 PWM DTS 节点、配置 PWM 内核驱动、控制 PWM 设备。

4.6.3.1 配置 PWM DTS 节点

在 DTS 源文件 kernel/arch/arm/boot/dts/rk3288.dtsi 添加 PWM DTS 节点，如下所示：

```
pwm1: pwm@ff680010 {
        compatible = "rockchip,rk-pwm";
        reg = <0xff680010 0x10>;
        #pwm-cells = <2>;
        pinctrl-names = "default";
        pinctrl-0 = <&pwm1_pin>;
        clocks = <&clk_gates11 11>;
        clock-names = "pclk_pwm";
        status = "okay";
    };
```

注：ff680010 为 PWM1 寄存器的地址。

4.6.3.2 配置 PWM 内核驱动

PWM 驱动位于文件 kernel/drivers/pwm/pwm-rockchip.c。

修改该文件的如下代码：

```
static const struct of_device_id rk_pwm_of_match[] = {
    { .compatible = "rockchip,pwm",          .data = &rk_pwm_data_v1,},
    { .compatible =  "rockchip,rk-pwm",      .data = &rk_pwm_data_v2,},
    { .compatible =  "rockchip,vop-pwm",     .data = &rk_pwm_data_v3,},
    { }};
```

把之前步骤中的 compatible = "rockchip,rk-pwm" 添加到如上代码中。

4.6.3.3 控制 PWM 设备

用户可在其它驱动文件中使用以上步骤生成的 PWM 节点。具体方法如下：

(1)、在要使用 PWM 控制的设备驱动文件中包含以下头文件：

```
#include <linux/pwm.h>
```

该头文件主要包含 PWM 的函数接口。

(2)、申请 PWM

使用

```
struct pwm_device *pwm_request(int pwm_id, const char *label);
```

函数申请 PWM。例如：

struct pwm_device * pwm0 = NULL;

pwm0 = pwm_request(0, “backlight-pwm”);

参数 pwm_id 表示要申请 PWM 的通道，label 为该 PWM 所取的标签。

(3)、配置 PWM

使用

int pwm_config(struct pwm_device *pwm, int duty_ns, int period_ns);

配置 PWM 的占空比，例如：

pwm_config(pwm0, 500000, 1000000)；

参数 pwm 为前一步骤申请的 pwm_device。duty_ns 为占空比激活的时长，单位为 ns。

period_ns 为 PWM 周期，单位为 ns。

(4)、使能 PWM

函数

int pwm_enable(struct pwm_device *pwm);

用于使能 PWM，例如：

pwm_enable(pwm0);

参数 pwm 为要使能的 pwm_device。

控制 PWM 输出主要使用以下接口函数：

struct pwm_device *pwm_request(int pwm_id, const char *label);

功能：用于申请 pwm

参数：

pwm_id：要申请的 pwm 通道。

label: 为该申请的 pwm 所取的标签。

void pwm_free(struct pwm_device *pwm);

功能：用于释放所申请的 pwm

参数：

pwm：所要释放的 pwm 结构体

int pwm_config(struct pwm_device *pwm, int duty_ns, int period_ns);

功能：用于配置 pwm 的占空比

参数：

pwm: 所要配置的 pwm

duty_ns：pwm 的占空比激活的时长，单位 ns

period_ns：pwm 占空比周期，单位 ns

int pwm_enable(struct pwm_device *pwm);

功能：使能 pwm

参数：

pwm：要使能的 pwm

void pwm_disable(struct pwm_device *pwm);

功能：禁止 pwm

参数：

pwm：要禁止的 pwm

4.7 UART 使用

4.7.1 板载资源介绍

Firefly-RK3288 开发板内置 5 路 UART，分别为 uart0，uart1，uart2，uart3，uart4。
uart0 为 uart_bt，用于蓝牙传输。
uart2 为 uart_dbg，用做调试串口。
uart 1、uart3、uart4 可做外部串口使用，开发板已将其引脚连接至 J10 处，其中 uart4 和 SPI0 引脚复用。
拥有 64 字节的 FIFO 收发缓冲区，支持 5 位、6 位、7 位、8 位数据收发和 DMA 操作。

4.7.2 配置步骤

以下以配置 uart3 为例。

4.7.2.1 配置 DTS 节点

文件 kernel/arch/arm/boot/dts/rk3288.dtsi 中已经有 uart 相关节点定义，如下所示：

```
uart_gps: serial@ff1b0000 {
        compatible = "rockchip,serial";
        reg = <0xff1b0000 0x100>;
        interrupts = <GIC_SPI 58 IRQ_TYPE_LEVEL_HIGH>;
        clock-frequency = <24000000>;
        clocks = <&clk_uart3>, <&clk_gates6 11>;
        clock-names = "sclk_uart", "pclk_uart";
        current-speed = <115200>;
        reg-shift = <2>;
        reg-io-width = <4>;
        dmas = <&pdma1 7>, <&pdma1 8>;
        #dma-cells = <2>;
        pinctrl-names = "default";
        pinctrl-0 = <&uart3_xfer &uart3_cts &uart3_rts>;
        status = "disabled";
};
```

注：uart_gps 在该文件的 aliases 节点中被定义为：serial3 = &uart_gps;
用户只需在 kernel/arch/arm/boot/dts/firefly-rk3288.dts 文件中打开所要使用的节点即可，如下所示：

```
&uart_gps {
        status = "okay";
        dma-names = "!tx", "!rx";
        pinctrl-0 = <&uart3_xfer &uart3_cts>;};
```

4.7.2.2 编译并烧写内核

将串口驱动编译到内核中，在 kernel 目录下执行如下命令：
make firefly-rk3288.img
把 kernel 目录下生成的 kernel.img 和 resource.img 烧录到开发板中即可。

4.7.2.3 串口通讯

配置好串口后，用户可以通过主机的 USB 转串口适配器向开发板的串口收发数据，步骤如下：
(1) 连接硬件
将开发板 uart3 的 TX、RX、GND 引脚分别和主机串口适配器的 RX、TX、GND 引脚相连。
(2) 打开主机的串口终端
在终端打开 kermit,并设置波特率：

```
$ sudo kermit
C-Kermit> set line /dev/ttyUSB0
C-Kermit> set speed 115200
C-Kermit> set flow-control none
C-Kermit> connect
```

/dev/ttyUSB0 为 USB 转串口适配器的设备文件
波特率与配置 DTS 节点中的 current-speed 属性相同
(3) 发送数据
uart3 的设备文件为 /dev/ttyS3。在设备上运行下列命令：

```
echo firefly uart3 test... > /dev/ttyS3
```

主机中的串口终端即可接收到字符串“firefly uart3 test...”
(4) 接收数据
首先在设备上运行下列命令：

```
cat /dev/ttyS3
```

然后在主机的串口终端输入字符串 “Firefly uart3 test...”，设备端即可见到相同的字符串。

4.8 Camera 使用

4.8.1 板载资源

Firefly-RK3288 开发板带有一个 MIPI 摄像头接口，图像处理能力达到 4416x3312 像素，支持 4K 视频录制。此外，开发板还支持 USB 摄像头。

本文以 OV13850 摄像头为例，讲解在该开发板上的配置过程。

4.8.2 相关代码目录

与摄像头相关的代码目录如下：

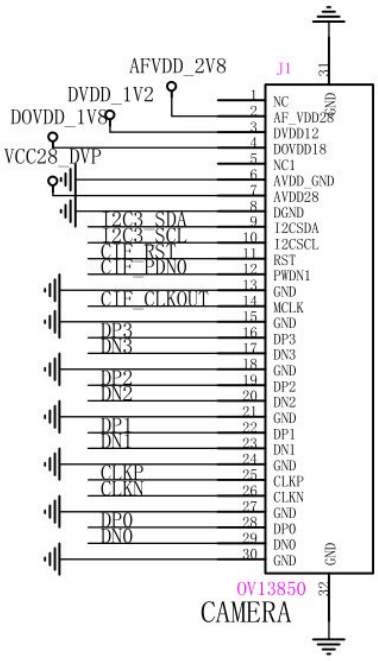
```
Android：
`- hardware/rk29/camera
    |- Config
    |   `- cam_board.xml          // 摄像头的参数设置
    |- CameraHal                   // 摄像头的 HAL 源码
    `- SiliconImage               // ISP 库，包括所有支持模组的驱动源码
        `- isi/drv/OV13850        // OV13850 模组的驱动源码
            `- calib/OV13850.xml // OV13850 模组的调校参数

Kernel：
|- kernel/drivers/media/video/rk_camsys    // CamSys 驱动源码
`- kernel/include/media/camsys_head.h
```

4.8.3 配置原理

设置摄像头相关的引脚和时钟，即可完成配置过程。

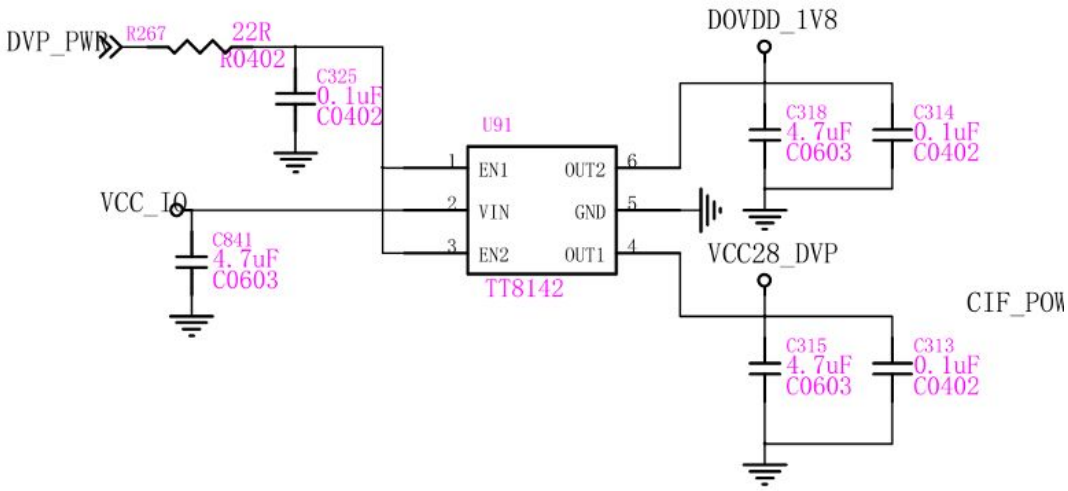
从以下摄像头接口原理图可知，需要配置的引脚有：AF_VDD28、DOVDD18、AVDD28、DVDD12、PWDN1、RST 和 MCLK。



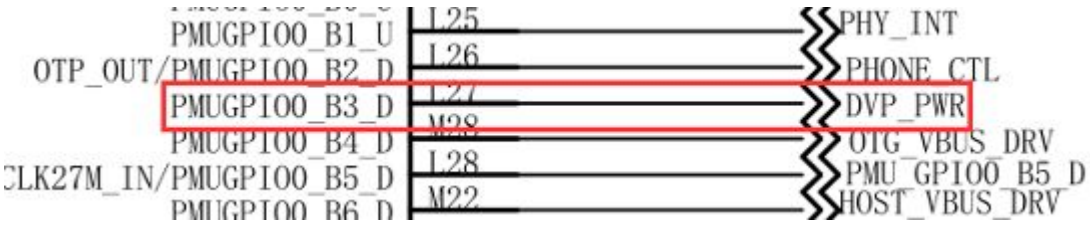
AF_VDD28 可不做配置。

DOVDD18、AVDD28

DOVDD18、AVDD28 由 DVP_PWR 控制：

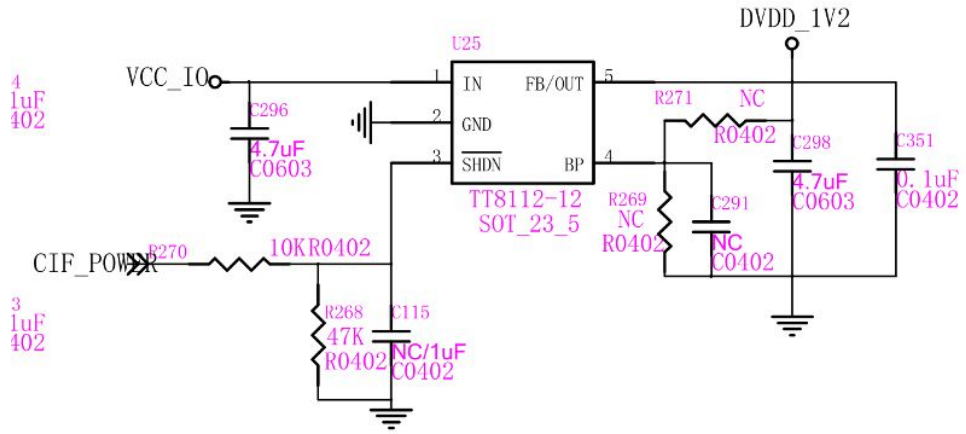


DVP_PWR 对应 RK3288 的 GPIO0_B3：

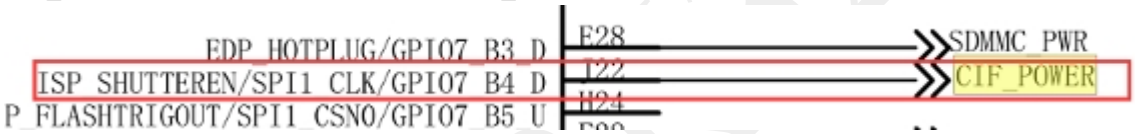


DVDD12

DVDD12 由 CIF_POWER 引脚控制：

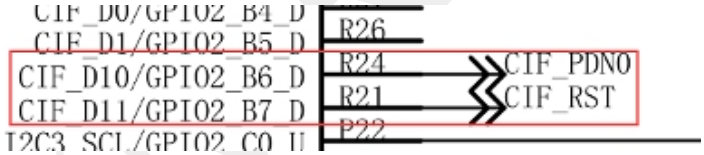


CIF_POWER 对应 RK3288 上的 GPIO7_B4 引脚：



PWDN1、RST

PWDN1、RST 对应 GPIO2_B6、GPIO2_B7 引脚：



在开发板中，除了 DVDD12 (CIF_POWER) 要在 DTS 和驱动中设置，其它引脚都是在 cam_board.xml 中设置。

4.8.4 配置步骤

4.8.4.1 配置 Android

修改 hardware/rk29/camera/Config/cam_board.xml 来注册摄像头：

```
<?xml version="1.0" ?><BoardFile>
        <BoardXmlVersion version="v0.7.0">
        </BoardXmlVersion>
        <CamDevie>
            <HardWareInfo>
                <Sensor>
                    <SensorName name="OV13850" ></SensorName>
                    <SensorDevID
IDname="CAMSYS_DEVID_SENSOR_1B"></SensorDevID>
```

```
                    <SensorHostDevID
busnum="CAMSYS_DEVID_MARVIN" ></SensorHostDevID>
                    <SensorI2cBusNum busnum="3"></SensorI2cBusNum>
                    <SensorI2cAddrByte byte="2"></SensorI2cAddrByte>
                    <SensorI2cRate rate="100000"></SensorI2cRate>
                    <SensorMclk mclk="24000000"></SensorMclk>
                    <SensorAvdd name="NC" min="0" max="0"></SensorAvdd>
                    <SensorDovdd name="NC" min="18000000"
max="18000000"></SensorDovdd>
                    <SensorDvdd name="NC" min="0" max="0"></SensorDvdd>
                    <SensorGpioPwdn ioname="RK30_PIN2_PB6"
active="0"></SensorGpioPwdn>
                    <SensorGpioRst ioname="RK30_PIN2_PB7"
active="0"></SensorGpioRst>
                    <SensorGpioPwen ioname="RK30_PIN0_PB3"
active="1"></SensorGpioPwen>
                    <SensorFacing facing="front"></SensorFacing>
                    <SensorInterface interface="MIPI"></SensorInterface>
                    <SensorMirrorFlip mirror="0"></SensorMirrorFlip>
                    <SensorOrientation orientation="0"></SensorOrientation>
                    <SensorPowerupSequence seq="1234"></SensorPowerupSequence>
                    <SensorFovParemeter h="60.0" v="60.0"></SensorFovParemeter>
                    <SensorAWB_Frame_Skip fps="15"></SensorAWB_Frame_Skip>
                    <SensorPhy phyMode="CamSys_Phy_Mipi" lane="2"    phyIndex="1"
sensorFmt="CamSys_Fmt_Raw_10b"></SensorPhy>
                </Sensor>
 <VCM>
                    <VCMDrvName name="BuiltInSensor"></VCMDrvName>
                    <VCMName name="NC"></VCMName>
                    <VCMI2cBusNum busnum="3"></VCMI2cBusNum>
                    <VCMI2cAddrByte byte="0"></VCMI2cAddrByte>
                    <VCMI2cRate rate="0"></VCMI2cRate>
                    <VCMVdd name="NC" min="0" max="0"></VCMVdd>
                    <VCMGpioPwdn ioname="NC" active="0"></VCMGpioPwdn>
                    <VCMGpioPower ioname="NC" active="0"></VCMGpioPower>
                    <VCMCurrent start="20" rated="80" vcmmax="100" stepmode="13"
drivermax="100"></VCMCurrent>
                </VCM>
 <Flash>
        <FlashName name="Internal"></FlashName>
        <FlashI2cBusNum busnum="0"></FlashI2cBusNum>
```

```
        <FlashI2cAddrByte byte="0"></FlashI2cAddrByte>
        <FlashI2cRate rate="0"></FlashI2cRate>
        <FlashTrigger ioname="NC" active="0"></FlashTrigger>
        <FlashEn ioname="NC" active="0"></FlashEn>
        <FlashModeType mode="1"></FlashModeType>
        <FlashLuminance luminance="0"></FlashLuminance>
        <FlashColorTemp colortemp="0"></FlashColorTemp>
    </Flash></HardWareInfo>
<SoftWareInfo>
        <AWB>
                <AWB_Auto support="1"></AWB_Auto>
                <AWB_Incandescent support="1"></AWB_Incandescent>
                <AWB_Fluorescent support="1"></AWB_Fluorescent>
                 <AWB_Warm_Fluorescent support="1"></AWB_Warm_Fluorescent>
                 <AWB_Daylight support="1"></AWB_Daylight>
                 <AWB_Cloudy_Daylight support="1"></AWB_Cloudy_Daylight>
                 <AWB_Twilight support="1"></AWB_Twilight>
                 <AWB_Shade support="1"></AWB_Shade>
         </AWB>
         <Sence>
                 <Sence_Mode_Auto support="1"></Sence_Mode_Auto>
                 <Sence_Mode_Action support="1"></Sence_Mode_Action>
                 <Sence_Mode_Portrait support="1"></Sence_Mode_Portrait>
                 <Sence_Mode_Landscape support="1"></Sence_Mode_Landscape>
                 <Sence_Mode_Night support="1"></Sence_Mode_Night>
                 <Sence_Mode_Night_Portrait support="1"></Sence_Mode_Night_Portrait>
                 <Sence_Mode_Theatre support="1"></Sence_Mode_Theatre>
                 <Sence_Mode_Beach support="1"></Sence_Mode_Beach>
                 <Sence_Mode_Snow support="1"></Sence_Mode_Snow>
                 <Sence_Mode_Sunset support="1"></Sence_Mode_Sunset>
                 <Sence_Mode_Steayphoto support="1"></Sence_Mode_Steayphoto>
                 <Sence_Mode_Pireworks support="1"></Sence_Mode_Pireworks>
                 <Sence_Mode_Sports support="1"></Sence_Mode_Sports>
                 <Sence_Mode_Party support="1"></Sence_Mode_Party>
                 <Sence_Mode_Candlelight support="1"></Sence_Mode_Candlelight>
                 <Sence_Mode_Barcode support="1"></Sence_Mode_Barcode>
                 <Sence_Mode_HDR support="1"></Sence_Mode_HDR>
         </Sence>
         <Effect>
                 <Effect_None support="1"></Effect_None>
                 <Effect_Mono support="1"></Effect_Mono>
```

```
                <Effect_Solarize support="1"></Effect_Solarize>
                <Effect_Negative support="1"></Effect_Negative>
                <Effect_Sepia support="1"></Effect_Sepia>
                <Effect_Posterize support="1"></Effect_Posterize>
                <Effect_Whiteboard support="1"></Effect_Whiteboard>
                <Effect_Blackboard support="1"></Effect_Blackboard>
                <Effect_Aqua support="1"></Effect_Aqua>
        </Effect>
                <FocusMode>
                    <Focus_Mode_Auto support="1"></Focus_Mode_Auto>
                    <Focus_Mode_Infinity support="1"></Focus_Mode_Infinity>
                    <Focus_Mode_Marco support="1"></Focus_Mode_Marco>
                    <Focus_Mode_Fixed support="1"></Focus_Mode_Fixed>
                    <Focus_Mode_Edof support="1"></Focus_Mode_Edof>
            <Focus_Mode_Continuous_Video support="0"></Focus_Mode_Continuous_Video>
            <Focus_Mode_Continuous_Picture support="1"></Focus_Mode_Continuous_Picture>
                </FocusMode>
                <FlashMode>
                    <Flash_Mode_Off support="1"></Flash_Mode_Off>
                    <Flash_Mode_On support="1"></Flash_Mode_On>
                    <Flash_Mode_Torch support="1"></Flash_Mode_Torch>
                    <Flash_Mode_Auto support="1"></Flash_Mode_Auto>
                    <Flash_Mode_Red_Eye support="1"></Flash_Mode_Red_Eye>
                </FlashMode>
                <AntiBanding>
                    <Anti_Banding_Auto support="1"></Anti_Banding_Auto>
                    <Anti_Banding_50HZ support="1"></Anti_Banding_50HZ>
                    <Anti_Banding_60HZ support="1"></Anti_Banding_60HZ>
                    <Anti_Banding_Off support="1"></Anti_Banding_Off>
                </AntiBanding>
                <HDR support="1"></HDR>
                <ZSL support="1"></ZSL>
                <DigitalZoom support="1"></DigitalZoom>
                <Continue_SnapShot support="1"></Continue_SnapShot>
                <PreviewSize width="800" height="600"></PreviewSize>
                <DV><DV_QCIF name="qcif" width="176" height="144" fps="10"
support="1"></DV_QCIF><DV_QVGA name="qvga" width="320" height="240" fps="10"
support="1"></DV_QVGA><DV_CIF name="cif" width="352" height="288" fps="10"
support="1"></DV_CIF><DV_VGA name="480p" width="640" height="480" fps="10"
support="0"></DV_VGA><DV_480P name="480p" width="720" height="480" fps="10"
support="0"></DV_480P><DV_720P name="720p" width="1280" height="720" fps="10"
```

support="1"></DV_720P><DV_1080P name="1080p" width="1920" height="1080" fps="10" support="1"></DV_1080P>
</DV>
</SoftWareInfo>
</CamDevie></BoardFile>

主要修改的内容如下：

Sensor 名称

<SensorName name="OV13850" ></SensorName>

该名字必须与 Sensor 驱动的名字一致,目前提供的 Sensor 驱动格式如下：

libisp_isi_drv_OV13850.so

用户可在编译 Android 完成后在目录 out/target/product/rk3288/system/lib/hw/ 下找到该摄像头驱动文件。

Sensor 软件标识

<SensorDevID IDname="CAMSYS_DEVID_SENSOR_1A"></SensorDevID>

注册标识不一致即可,可填写以下值：

CAMSYS_DEVID_SENSOR_1A

CAMSYS_DEVID_SENSOR_1B

CAMSYS_DEVID_SENSOR_2

采集控制器名称

<SensorHostDevID busnum="CAMSYS_DEVID_MARVIN" ></SensorHostDevID>

目前只支持：

CAMSYS_DEVID_MARVIN

Sensor 所连接的主控 I2C 通道号

<SensorI2cBusNum busnum="3"></SensorI2cBusNum>

具体通道号请参考摄像头原理图连接主控的 I2C 通道号。

Sensor 寄存器地址长度,单位：字节

<SensorI2cAddrByte byte="2"></SensorI2cAddrByte>

Sensor 的 I2C 频率,单位：Hz，用于设置 I2C 的频率。

<SensorI2cRate rate="100000"></SensorI2cRate>

Sensor 输入时钟频率, 单位：Hz，用于设置摄像头的时钟。

<SensorMclk mclk="24000000"></SensorMclk>

Sensor AVDD 的 PMU LDO 名称。如果不是连接到 PMU，那么只需填写 NC。

<SensorAvdd name="NC" min="0" max="0"></SensorAvdd>

Sensor DOVDD 的 PMU LDO 名称。

<SensorDovdd name="NC" min="18000000" max="18000000"></SensorDovdd>

如果不是连接到 PMU，那么只需填写 NC。注意 min 以及 max 值必须填写，这决定了 Sensor 的 IO 电压。

Sensor DVDD 的 PMU LDO 名称。

<SensorDvdd name="NC" min="0" max="0"></SensorDvdd>

如果不是连接到 PMU，那么只需填写 NC。

Sensor PowerDown 引脚。
<SensorGpioPwdn ioname="RK30_PIN2_PB6" active="0"></SensorGpioPwdn>
直接填写名称即可，active 填写休眠的有效电平。
Sensor Reset 引脚。
<SensorGpioRst ioname="RK30_PIN2_PB7" active="0"></SensorGpioRst>
直接填写名称即可，active 填写复位的有效电平。
Sensor Power 引脚。
<SensorGpioPwen ioname="RK30_PIN0_PB3" active="1"></SensorGpioPwen>
直接填写名称即可, active 填写电源有效电平。
选择 Sensor 作为前置还是后置。
<SensorFacing facing="front"></SensorFacing>
可填写 "front" 或 "back"。
Sensor 的接口方式
<SensorInterface mode="MIPI"></SensorInterface>
可填写如下值：
CCIR601
CCIR656
MIPI
SMIA
Sensor 的镜像方式
<SensorMirrorFlip mirror="0"></SensorMirrorFlip>
目前暂不支持。
Sensor 的角度信息
<SensorOrientation orientation="0"></SensorOrientation>
物理接口设置
MIPI
<SensorPhy phyMode="CamSys_Phy_Mipi" lane="2" phyIndex="1" sensorFmt="CamSys_Fmt_Raw_10b"> </SensorPhy>
hyMode：Sensor 接口硬件连接方式，对 MIPI Sensor 来说，该值取 "CamSys_Phy_Mipi"
Lane：Sensor mipi 接口数据通道数
Phyindex：Sensor mipi 连接的主控 mipi phy 编号
sensorFmt：Sensor 输出数据格式,目前仅支持 CamSys_Fmt_Raw_10b
DVP
<SensorPhy phyMode="CamSys_Phy_Cif" sensor_d0_to_cif_d ="2" cif_num="0" sensorFmt="CamSys_Fmt_Raw_10b"></SensorPhy>
phyMode： Sensor 接口硬件连接方式，DVP Sensor 接口则为：CamSys_Phy_Cif
sensor_d0_to_cif_d：Sensor DVP 输出数据位 D0 对应连接的主控 DVP 接口的数据位号码
cif_num：Sensor DVP 连接到主控 DVP 接口编号
sensorFmt：Sensor 输出的数据格式,目前版本仅支持填写 CamSys_Fmt_Raw_10b

4.8.4.2 配置内核

在配置原理中提到，GPIO7_B4 需要在 DTS 和驱动中配置。其配置方法如下：

(1). DTS 文件添加 GPIO7_B4 配置属性

在 kernel/arch/arm/boot/dts/rk3288.dtsi 文件中添加 gpios-cifpower 属性，如下所示：

```
isp： isp@ff910000{
        compatible = "firefly,isp";
       ...
        gpios-cifpower = <&gpio7 GPIO_B4 GPIO_ACTIVE_HIGH>;
    ...
        status = "okay";
};
```

(2). 驱动中配置 CIF_POWER

在 kernel/drivers/media/video/rk_camsys/camsys_drv.c 中读取 gpios-cifpower ，并设置该引脚，使能 CIF_POWER，在 probe 函数 camsys_platform_probe()中添加如下所示：

```
enum of_gpio_flags flags;
int cifpower_io;
int io_ret;
cifpower_io = of_get_named_gpio_flags(dev->of_node, "gpios-cifpower", 0, &flags);
camsys_trace(1, "1-gpios-cifpower： gpio=%d", cifpower_io);
if(gpio_is_valid(cifpower_io)){
        cifpower_io = of_get_named_gpio_flags(dev->of_node, "gpios-cifpower", 0,
&flags);
        camsys_trace(1, "gpios-cifpower： gpio_request");
        io_ret = gpio_request(cifpower_io,"cifpower");
        camsys_trace(1, "1-gpios-cifpower： gpio_request=%d", io_ret);
        if(io_ret < 0){
          camsys_err("Request %s(%d) failed","cifpower", cifpower_io);
        }
        else{
              gpio_direction_output(cifpower_io, 1);
              gpio_set_value(cifpower_io, 1);
             camsys_trace(1, "gpios-cifpower： %d high", cifpower_io);
          }
}
```

(3). 编译内核

需将 drivers\media\video\rk_camsys 驱动源码编进内核，其配置方法如下：

在内核源码目录下执行命令：

make menuconfig

然后将以下配置项打开：

Device Drivers --->

<*>Multimedia support --->
<*>camsys driver
RockChip camera system driver --->
<*> camsys driver for marvin isp
< > camsys driver for cif

最后执行：

make firefly-rk3288.img

即可完成内核的编译。

第 5 章 附录

5.1 硬件资料

原理图：0809 (Beta) 0930
贴片图：0809 (Beta) 0930
器件规格书

5.2 工具文档

RK3288 规格书
AndroidTool 升级工具使用手册
Linux 升级工具使用手册
驱动助手说明文档