

# IPC Linux SDK 快速开发指南

---

文档标识：RK-JC-YF-920

发布版本：V1.4.0

日期：2023-08-18

文件密级：☐绝密 ☐秘密 ☐内部资料 ☒公开

## 免责声明

本文档按“现状”提供，瑞芯微电子股份有限公司（“本公司”，下同）不对本文档的任何陈述、信息和内容的准确性、可靠性、完整性、适销性、特定目的性和非侵权性提供任何明示或暗示的声明或保证。本文档仅作为使用指导的参考。

由于产品版本升级或其他原因，本文档将可能在未经任何通知的情况下，不定期进行更新或修改。

## 商标声明

“Rockchip”、“瑞芯微”、“瑞芯”均为本公司的注册商标，归本公司所有。

本文档可能提及的其他所有注册商标或商标，由其各自拥有者所有。

## 版权所有 © 2023 瑞芯微电子股份有限公司

超越合理使用范畴，非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

瑞芯微电子股份有限公司

Rockchip Electronics Co., Ltd.

地址：福建省福州市铜盘路软件园A区18号

网址：[www.rock-chips.com](http://www.rock-chips.com)

客户服务电话：+86-4007-700-590

客户服务传真：+86-591-83951833

客户服务邮箱：[fae@rock-chips.com](mailto:fae@rock-chips.com)

**前言**

**概述**

本文主要描述了IPC Linux SDK的基本使用方法，旨在帮助开发者快速了解并使用IPC SDK开发包。

**产品版本**

芯片名称	内核版本
RK3588	Linux 5.10
RV1106/RV1103	Linux 5.10
RV1126/RV1109	Linux 4.19

**读者对象**

本文档（本指南）主要适用于以下工程师：

- 技术支持工程师
- 软件开发工程师

**修订记录**

版本号	作者	修改日期	修改说明
V1.0.0	CWW	2021-12-17	初始版本
V1.0.1	CWW	2022-01-01	1. 更新文档 2. 增加Q&A
V1.0.2	GZC	2022-01-13	1. 更新Q&A： <a href="#">如何使用Recovery和插入SD卡检测不到设备</a>
V1.0.3	CWW	2022-02-06	1. 更新 <a href="#">交叉工具链下载以及安装</a> 2. 增加 <a href="#">第三方案序集成说明</a> 3. <a href="#">更新添加新APP到project编译说明</a>
V1.0.4	Ruby Zhang	2022-02-15	更新一些语言描述
V1.0.5	CWW	2022-02-21	1. 更新 <a href="#">交叉工具链下载以及安装</a> 2. 增加 <a href="#">安全启动相关代码以及文档说明</a>
V1.0.6	GZC, CWW	2022-03-02	1. 更新Q&A： a. <a href="#">分区表说明</a> b. <a href="#">如何在U-Boot终端下使用tftp进行分区升级</a> c. <a href="#">如何在U-Boot终端下使用SD卡进行分区升级</a> 2. 更新 <a href="#">烧录工具说明</a> 3. 增加 <a href="#">Sysdrv目录说明</a> 和增加 <a href="#">Media目录说明</a> 4. 增加 <a href="#">BoardConfig.mk的配置项说明</a>
V1.0.7	CWW	2022-03-26	1. 更新SocToolKit 2. 增加 <a href="#">RV1106 IPC SDK在线下载</a> 3. 增加 <a href="#">RV1126 IPC SDK在线下载</a>
V1.0.8	GZC	2022-04-02	1. 更新Q&A： a. <a href="#">oem分区挂载说明</a> b. <a href="#">如何下载NPU模型转换工具以及runtime库</a> c. <a href="#">压力测试使用方法</a> 2. 更新 <a href="#">SDK目录结构说明</a>
V1.0.9	CWW	2022-04-12	1. 更新 <a href="#">spi nor分区表</a> 2. 更新拼写错误
V1.1.0	CWW	2022-04-15	1. 增加 <a href="#">通过Telnet调试</a> 2. Add <a href="#">env.img 格式说明</a> 3. 增加 <a href="#">内核驱动insmod说明</a> 4. 增加 <a href="#">编译内核驱动</a>
V1.1.1	CWW	2022-05-07	1. 更新 <a href="#">开发烧写工具</a> 2. 更新 <a href="#">Media目录说明</a> 3. <a href="#">BoardConfig.mk的配置项说明</a> 4. 更新 <a href="#">内核驱动insmod说明</a>
V1.1.2	CWW	2022-05-09	1. 更新 <a href="#">如何在U-Boot终端下使用tftp进行分区升级</a> 2. 更新 <a href="#">文档说明</a>

版本号	作者	修改日期	修改说明
V1.1.3	CWW	2022-05-20	<ol style="list-style-type: none"> <li>1. 增加<a href="#">通过串口调试</a></li> <li>2. 增加<a href="#">如何使用coredump功能</a></li> <li>3. 更新<a href="#">BoardConfig.mk</a>的配置项说明</li> <li>4. 更新<a href="#">内核驱动insmod说明</a></li> <li>5. 增加<a href="#">RV1106和RV1103平台相关库文件和驱动文件的信息</a></li> <li>5. 增加<a href="#">如何使用NFS文件系统</a></li> <li>6. 增加<a href="#">如何增加新用户并设置登陆密码</a></li> <li>7. 增加<a href="#">获取摄像头支持列表</a></li> </ol>
V1.1.4	GZC	2022-05-30	<ol style="list-style-type: none"> <li>1. 更新<a href="#">SD升级启动制作工具</a></li> <li>2. 更新<a href="#">如何在U-Boot终端下使用SD卡进行分区升级</a></li> <li>3. 增加<a href="#">获取Flash支持列表</a></li> <li>4. 更新<a href="#">update.img相关工具</a></li> <li>5. 更新<a href="#">工厂固件说明</a></li> <li>6. 增加<a href="#">如何在板端修改系统CMA大小</a></li> </ol>
V1.1.5	GZC	2022-07-12	<ol style="list-style-type: none"> <li>1. 更新<a href="#">量产升级工具</a></li> <li>2. 更新<a href="#">服务器环境搭建</a></li> </ol>
V1.1.6	CWW	2022-08-02	<ol style="list-style-type: none"> <li>1. 增加<a href="#">如何使用rmdis功能</a></li> </ol>
V1.1.7	CWW	2022-10-25	<ol style="list-style-type: none"> <li>1. 更新<a href="#">BoardConfig.mk</a>的配置项说明</li> <li>2. 增加<a href="#">App目录说明</a></li> <li>3. 更新<a href="#">如何在板端修改系统CMA大小</a></li> <li>4. 更新<a href="#">打包env.img</a></li> </ol>
V1.1.8	GZC	2022-11-17	<ol style="list-style-type: none"> <li>1. 增加<a href="#">A/B系统使用方法</a></li> <li>2. 更新Q&amp;A：<a href="#">如何使用Recovery</a></li> <li>3. 增加<a href="#">如何优化spi nor启动速度</a></li> <li>4. 更新<a href="#">获取SDK包</a></li> </ol>
V1.2.0	CWW GZC	2022-12-08	<ol style="list-style-type: none"> <li>1. 增加<a href="#">如何增加非root用户登陆</a></li> <li>2. 更新<a href="#">如何添加第三方库到sysdrv目录编译</a></li> <li>3. 更新<a href="#">内核驱动insmod说明</a></li> <li>4. 增加<a href="#">如何添加新的Camera sensor配置</a></li> <li>5. <a href="#">如何添加reboot命令进U-Boot终端</a></li> <li>6. 更新<a href="#">BoardConfig.mk</a>的配置项说明</li> <li>7. 更新<a href="#">服务器环境搭建</a></li> <li>8. 更新<a href="#">安全启动相关代码以及文档说明</a></li> <li>9. 更新<a href="#">如何使用Recovery</a></li> <li>10. 增加<a href="#">如何在 U-Boot 里支持 USB 大容量存储功能</a></li> </ol>
V1.2.1	GZC	2023-01-16	<ol style="list-style-type: none"> <li>1. 更新<a href="#">安全启动相关代码以及文档说明</a></li> <li>2. 更新<a href="#">下载repo工具以及使用</a></li> </ol>
V1.2.2	GZC	2023-03-16	<ol style="list-style-type: none"> <li>1. 更新<a href="#">量产升级工具</a></li> <li>2. 更新<a href="#">固件烧录</a></li> <li>3. 增加<a href="#">如何在U-Boot阶段通过按键触发SD卡升级功能</a></li> <li>4. 增加<a href="#">如何通过指令修改GPIO寄存器配置</a></li> </ol>

版本号	作者	修改日期	修改说明
V1.3.0	CWW GZC	2023-06-02	<ol style="list-style-type: none"><li>1. 更新<a href="#">工厂固件说明</a></li><li>2. 更新<a href="#">如何在板端修改系统CMA大小</a></li><li>3. 增加<a href="#">如何使用UVC预览</a></li><li>4. 更新<a href="#">如何使用Recovery</a></li><li>5. 更新<a href="#">A/B系统使用方法</a></li><li>6. 更新<a href="#">如何使用rndis功能</a></li><li>7. 增加<a href="#">如何开启OP-TEE功能</a></li><li>8. 更新<a href="#">SD升级启动制作工具</a></li><li>9. 更新<a href="#">安全启动相关代码以及文档说明</a></li></ol>
V1.4.0	CWW	2023-08-18	<ol style="list-style-type: none"><li>1. 增加<a href="#">休眠唤醒功能开发</a></li></ol>

# 目录

## IPC Linux SDK 快速开发指南

1. 服务器环境搭建
  - 1.1 下载repo工具以及使用
  - 1.2 获取SDK包
  - 1.3 更新SDK代码
  - 1.4 交叉工具链下载以及安装
2. SDK 使用说明
  - 2.1 BoardConfig.mk的配置项说明
  - 2.2 查看SDK版本以及编译配置
  - 2.3 一键自动编译
  - 2.4 编译U-Boot
  - 2.5 编译kernel
  - 2.6 编译rootfs
  - 2.7 编译media
  - 2.8 编译参考应用
  - 2.9 编译内核驱动
  - 2.10 打包env.img
  - 2.11 固件打包
  - 2.12 SDK目录结构说明
    - 2.12.1 Sysdrv目录说明
    - 2.12.2 Media目录说明
    - 2.12.3 App目录说明
  - 2.13 镜像存放目录说明
  - 2.14 调试工具
    - 2.14.1 通过网络tftp传输文件
    - 2.14.2 通过网络ADB调试
    - 2.14.3 通过Telnet调试
    - 2.14.4 通过串口调试
3. 文档说明
4. 工具说明
  - 4.1 驱动安装工具
  - 4.2 固件烧录
  - 4.3 update.img相关工具
    - 4.3.1 打包
    - 4.3.2 解包
    - 4.3.3 update.img烧写
  - 4.4 SD升级启动制作工具
  - 4.5 工厂固件说明
  - 4.6 量产升级工具
5. IPC Linux SDK Q&A
  - 5.1 如何修改分区表和增加自定义分区以及分区可读写说明
    - 5.1.1 存储介质以及文件系统类型说明
    - 5.1.2 分区表说明
    - 5.1.3 增加自定义分区
    - 5.1.4 oem分区挂载说明
  - 5.2 如何使用Recovery
  - 5.3 如何在U-Boot终端下使用tftp进行分区升级
  - 5.4 如何在U-Boot终端下使用SD卡进行分区升级
  - 5.5 如何在U-Boot阶段通过按键触发SD卡升级功能
  - 5.6 插入SD卡检测不到设备
  - 5.7 如何添加第三方库到sysdrv目录编译
  - 5.8 如何在project/app里增加新应用程序
  - 5.9 如何下载NPU模型转换工具以及runtime库
  - 5.10 如何在板端修改系统CMA大小
  - 5.11 如何使用coredump功能

- 5.12 内核驱动insmod说明
- 5.13 RV1106和RV1103平台相关库文件和驱动文件的信息
- 5.14 如何使用NFS文件系统
- 5.15 如何增加新用户并设置登陆密码
- 5.16 A/B系统使用方法
  - 5.16.1 启动方案介绍
  - 5.16.2 配置
  - 5.16.3 OTA升级工具
  - 5.16.4 A/B系统切换
  - 5.16.5 A/B系统升级
- 5.17 获取摄像头支持列表
- 5.18 获取Flash支持列表
- 5.19 压力测试使用方法
  - 5.19.1 memtester test
  - 5.19.2 stressapptest
  - 5.19.3 cpufreq test
  - 5.19.4 flash stress test
  - 5.19.5 reboot test
- 5.20 安全启动相关代码以及文档说明
  - 5.20.1 Key
  - 5.20.2 U-Boot配置
  - 5.20.3 固件签名
- 5.21 如何使用rndis功能
- 5.22 如何优化SPI NOR启动速度
- 5.23 如何增加非root用户登陆
- 5.24 如何添加新的Camera sensor配置
- 5.25 如何添加reboot命令进U-Boot终端
- 5.26 如何在 U-Boot 里支持 USB 大容量存储功能
- 5.27 如何通过指令修改GPIO寄存器配置
  - 5.27.1 U-Boot终端
  - 5.27.2 根文件系统
- 5.28 如何使用UVC预览
- 5.29 如何开启OP-TEE功能
  - 5.29.1 OP-TEE介绍
  - 5.29.2 打开U-Boot和kernel的OP-TEE配置
  - 5.29.3 配置U-Boot OP-TEE固件
  - 5.29.4 打开kernel OP-TEE驱动
  - 5.29.5 在RV1106 SDK中编译TEE库文件
  - 5.29.6 测试验证
- 5.30 休眠唤醒功能开发
  - 5.30.1 如何开启休眠唤醒功能
  - 5.30.2 休眠唤醒方案简介
  - 5.30.3 调试
  - 5.30.4 硬件参考设计
  - 5.30.5 唤醒按键配置举例
- 6. 注意事项

# 1. 服务器环境搭建

本 SDK 开发环境是在 Ubuntu 系统上开发测试。我们推荐使用 Ubuntu 18.04 的系统进行编译。其他的 Linux 版本可能需要对软件包做相应调整。除了系统要求外，还有其他软硬件方面的要求。

硬件要求：64 位系统，硬盘空间大于 20G。如果您进行多个构建，将需要更大的硬盘空间。

软件要求：Ubuntu 18.04 系统

编译 SDK 环境搭建所依赖的软件包安装命令如下：

```
sudo apt-get install repo git ssh make gcc \
gcc-multilib g++-multilib module-assistant \
expect g++ gawk texinfo libssl-dev \
bison flex fakeroot cmake unzip gperf autoconf \
device-tree-compiler libncurses5-dev
```

建议使用 Ubuntu 18.04 系统或更高版本开发，若编译遇到报错，可以视报错信息，安装对应的软件包。

## 1.1 下载repo工具以及使用

```
mkdir -p $HOME/repo-tool
git clone ssh://git@www.rockchip.com.cn/repo/rk/tools/repo $HOME/repo-tool

export PATH="$HOME/repo-tool:$PATH"
# 测试命令
repo version
```

## 1.2 获取SDK包

SDK 下载有 2 种方式，分别是 **在线下载方式** 和 **离线包方式**

注：不同平台的 SDK 需要开通对应的下载权限。

### 1. 在线下载方式

请联系业务或 FAE 获取对应芯片的 SDK Release 文档。

比如：RV1106/RV1103 Linux IPC SDK Release 文档

Rockchip\_RV1106\_RV1103\_Linux\_IPC\_SDK\_Release\_V1.0.0\_20220530\_CN.pdf

### 2. SDK 离线包

SDK 离线包可以从 FAE 窗口获取。

```
# 以RK3588_IPC_LINUX_SDK_V1.0.0_XXX.tar.bz2为例
mkdir rk3588_ipc_linux_sdk
tar xf RK3588_IPC_LINUX_SDK_V1.0.0_XXX.tar.bz2 -C rk3588_ipc_linux_sdk
cd rk3588_ipc_linux_sdk

# 检出本地代码
.repo/repo/repo sync -l
```



## 1.3 更新SDK代码

更新SDK代码前，需要备份本地修改。

```
# 更新SDK代码命令
.repo/repo/repo sync -c --no-tags
# 如果有仓库下载不了，可以加上强制更新的参数 --force-sync
.repo/repo/repo sync -c --no-tags --force-sync
# 更新SDK代码后，需要进行clean操作
./build.sh clean
```

## 1.4 交叉工具链下载以及安装

交叉工具链可以从SDK目录下tools/linux/toolchain/获取。

芯片名称	交叉工具链	测试命令
RK3588	gcc-arm-10.3-2021.07-x86_64 -aarch64-none-linux-gnu	aarch64-rockchip1031-linux-gnu-gcc --version
RV1106	arm-rockchip830-linux -uclibcgnueabihf	arm-rockchip830-linux-uclibcgnueabihf-gcc --version
RV1126 /RV1109	gcc-arm-8.3-2019.03-x86_64 -arm-linux-gnueabihf/	arm-rockchip830-linux-gnueabihf-gcc --version

```
cd tools/linux/toolchain/aarch64-rockchip1031-linux-gnu
source env_install_toolchain.sh
# or install toolchain to the dirname
# source env_install_toolchain.sh dirname
```

交叉工具链在SDK目录 tools/linux/toolchain 。

## 2. SDK 使用说明

### 2.1 BoardConfig.mk的配置项说明

- 板级配置文件BoardConfig.mk说明

SDK的板级配置在 project/cfg/ 目录下，BoardConfig.mk文件是SDK编译的重要文件。

project/cfg-all-items-introduction.txt 会记录最新配置项说明。

配置项	说明
RK_ARCH	arm或arm64 定义编译32位或64位程序
RK_CHIP	不可修改 不同的芯片对应不同的SDK
RK_TOOLCHAIN_CROSS	不可修改 定义交叉工具链
RK_BOOT_MEDIUM	emmc或spi_nor或spi_nand 定义板子存储类型
RK_UBOOT_DEFCONFIG	U-Boot defconfig文件名 文件目录sysdrv/source/uboot/u-boot/configs
RK_UBOOT_DEFCONFIG_FRAGMENT	U-Boot config文件名（可选） 文件目录sysdrv/source/uboot/u-boot/configs 对RK_UBOOT_DEFCONFIG定义的defconfig进行覆盖
RK_KERNEL_DEFCONFIG	内核defconfig文件名 文件目录sysdrv/source/kernel/arch/\$RK_ARCH/configs
RK_KERNEL_DEFCONFIG_FRAGMENT	内核defconfig文件名（可选） 文件目录sysdrv/source/kernel/arch/\$RK_ARCH/configs 对RK_KERNEL_DEFCONFIG定义的defconfig进行覆盖
RK_KERNEL_DTS	内核dts文件名 RK_ARCH=arm目录： sysdrv/source/kernel/arch/arm/boot/dts RK_ARCH=arm64目录： sysdrv/source/kernel/arch/arm64/boot/dts/rockchip
RK_MISC	如果打开recovery功能，系统启动时读取标志选择进recovery系统或应用系统（没有recovery时，可以去掉）
RK_CAMERA_SENSOR_IQFILES	Camera Sensor的IQ配置文件 文件目录media/isp/camera_engine_rkaiq/iqfiles或media/isp/camera_engine_rkaiq/rkaiq/iqfiles 多个IQ文件用空格隔开，例如 RK_CAMERA_SENSOR_IQFILES="iqfile_1 iqfile_2"
RK_PARTITION_CMD_IN_ENV	配置分区表（重要） 分区表格式：<partdef>[,<partdef>] <partdef>格式：<size>[@<offset>](part-name) 详细配置参考 <a href="#">分区表说明章节</a>
RK_PARTITION_FS_TYPE_CFG	配置分区文件系统类型以及挂载点（重要） 格式说明： 分区名称@分区挂载点@分区文件系统类型 注：根文件系统的分区挂载点默认值是IGNORE（不可修改）
RK_SQUASHFS_COMP	配置squashfs镜像压缩算法（可选） 支持：lz4/lzo/lzma/xz/gzip (default xz)

配置项	说明
RK_UBIFS_COMP	配置ubifs镜像压缩算法（可选） 支持：lzo/zlib (default lzo)
RK_APP_TYPE	配置编译参考的应用（可选） 运行./build.sh info 可以查看支持的参考应用
RK_APP_IPCWEB_BACKEND	配置是否编译web应用（可选） y:使能
RK_BUILD_APP_TO_OEM_PARTITION	配置是否将应用安装到oem分区（可选） y:使能
RK_ENABLE_RECOVERY	配置是否使能recovery功能（可选） y:使能 n:关闭
RK_ENABLE_FASTBOOT	配置是否快速启动功能（可选） y:使能 需要配合U-Boot和内核修改，可以参考SDK提供的BoardConfig-*-TB.mk
RK_ENABLE_GDB	配置是否编译gdb（可选） y:使能 n:关闭
RK_ENABLE_ADBD	配置是否支持adb功能（可选） y:使能 n:关闭 注：需要内核打开对应USB配置
RK_BOOTARGS_CMA_SIZE	配置内核CMA大小（可选）
RK_POST_BUILD_SCRIPT	配置的脚本将会在打包rootfs.img前执行（脚本放在BoardConfig对应目录下）（可选）
RK_PRE_BUILD_OEM_SCRIPT	配置的脚本将会在打包oem.img前执行（脚本放在BoardConfig对应目录下）（可选）
RK_BUILD_APP_TO_OEM_PARTITION	配置是否将应用安装到oem分区（可选） y:使能
RK_ENABLE_RNDIS	配置是否打开rndis功能（可选） y:使能 n:关闭
RK_META_PARAM	配置meta分区参数（可选，用于电池IPC类产品）

- 选择BoardConfig的命令

```
./build.sh lunch
```

```
You're building on Linux
Lunch menu...pick a combo:

BoardConfig-*.mk naming rules:
BoardConfig-"启动介质"-"电源方案"-"硬件版本"-"应用场景".mk
BoardConfig-"boot medium"-"power solution"-"hardware version"-"application".mk

-----
```

```
0. BoardConfig-EMMC-2xRK806-HW_V10-IPC_MULTI_SENSOR.mk
    boot medium(启动介质): EMMC
    power solution(电源方案): 2xRK806
    hardware version(硬件版本): HW_V10
    application(应用场景): IPC_MULTI_SENSOR

-----

1. BoardConfig-EMMC-RK806-HW_V10-IPC_SINGLE_SENSOR.mk
    boot medium(启动介质): EMMC
    power solution(电源方案): RK806
    hardware version(硬件版本): HW_V10
    application(应用场景): IPC_SINGLE_SENSOR

-----

Which would you like? [0]:
```

输入对应的序号选择对应的参考板级。

## 2.2 查看SDK版本以及编译配置

```
./build.sh info
```

板端查看SDK版本命令 `sdkinfo`

```
# sdkinfo
Build Time: 2022-03-02-20:26:13
SDK Version: rk3588_ipc_linux_v0.0.5_20220221.xml
```

## 2.3 一键自动编译

```
./build.sh lunch    # 选择参考板级
./build.sh          # 一键自动编译
```

## 2.4 编译U-Boot

```
./build.sh clean uboot
./build.sh uboot

# ./build.sh info 可以查看uboot详细的编译命令格式
```

生成镜像文件：output/image/download.bin、output/image/idblock.img 和 output/image/uboot.img

## 2.5 编译kernel

```
./build.sh clean kernel
./build.sh kernel

# ./build.sh info 可以查看kernel详细的编译命令格式
```

生成镜像文件：output/image/boot.img

## 2.6 编译rootfs

```
./build.sh clean rootfs
./build.sh rootfs
```

编译后使用 `./build.sh firmware` 命令打包成rootfs.img

生成镜像文件：output/image/rootfs.img

## 2.7 编译media

```
./build.sh clean media
./build.sh media
```

生成文件的存放目录：output/out/media\_out

## 2.8 编译参考应用

```
./build.sh clean app
./build.sh app
```

生成文件的存放目录：output/out/app\_out

注：app依赖media

## 2.9 编译内核驱动

```
./build.sh clean driver
./build.sh driver
```

生成文件的存放目录：output/out/sysdrv\_out/kernel\_drv\_ko/

## 2.10 打包env.img

```
./build.sh env
```

env.img是用uboot的mkenvimage工具进行打包。

env.img打包命令格式：`mkenvimage -s $env_partition_size -p 0x0 -o env.img env.txt`

注意：不同的存储介质，`$env_partition_size` 不一样，具体查看[分区表说明](#)

查看env.img内容：`strings env.img`

# 例如eMMC的env.txt内容

```
blkdevparts=mmcblk0:32K(env),512K@32K(idblock),256K(uboot),32M(boot),2G(rootfs),1G(oem),2G(userdata),-(media)
```

注意：不同的存储介质，env.img内容会不一样，可以用 `strings env.img` 查看。

blkdevparts会被uboot传递给内核，并覆盖内核对应bootargs的参数。

## 2.11 固件打包

```
./build.sh firmware
```

生成文件的存放目录：output/image

## 2.12 SDK目录结构说明

Directory Path	Introduction
build.sh	SDK编译脚本 软链接到project/build.sh
media	多媒体编解码、ISP等算法相关
sysdrv	U-Boot、kernel、rootfs目录
project	参考应用、编译配置以及脚本目录
docs	SDK文档目录
tools	烧录镜像打包工具以及烧录工具
output	SDK编译后镜像文件存放目录
output/image	烧录镜像输出目录
output/out	编译生成的文件
output/out/app_out	参考应用编译后的文件
output/out/media_out	media相关编译后的文件
output/out/sysdrv_out	sysdrv编译后的文件
output/out/sysdrv_out/kernel_drv_ko	外设和多媒体的ko文件
output/out/rootfs_XXX	文件系统打包目录
output/out/S20linkmount	分区挂载脚本
output/out/userdata	userdata

**注：**media和sysdrv可以独立SDK编译。

### 2.12.1 Sysdrv目录说明

sysdrv可以独立于SDK进行编译，包含U-Boot、kernel、rootfs以及一些镜像打包工具。

编译命令：

```
# 默认全部编译
make all

# 编译U-Boot
make uboot_clean
make uboot

# 编译内核
make kernel_clean
make kernel

# 编译rootfs
make rootfs_clean
make rootfs
```

```
# 清除编译
make clean

# 清除编译并删掉out目录
make distclean

# 查看编译配置，比如uboot、kernel详细的编译命令
make info
```

sysdrv子目录	说明
cfg	内核和U-Boot编译相关的配置
out	sysdrv编译输出目录
out/bin/board_glibc_xxx	运行在板端的程序
out/bin/pc	运行在PC端的程序
out/bin/image_glibc_xxx	生成的烧录镜像输出目录
out/bin/rootfs_glibc_xxx	根文件系统目录
source/busybox	busybox编译目录，源码在sysdrv/tools/board/busybox
source/kernel	内核源码目录
source/uboot	U-Boot源码目录以及rkbin（ddr初始化预编译镜像）
tools/board	板端程序源码
tools/pc	PC端打包镜像的工具

### 2.12.2 Media目录说明

media可以独立于SDK进行编译，包含多媒体编解码、ISP等算法相关。

编译命令：

```
# 默认全编译
make

# 清除编译文件
make clean

# 查看编译配置
make info
```



media子目录	说明
cfg	配置模块是否编译
alsa-lib	Advanced Linux Sound Architecture (ALSA) library
avs	全景拼接（只支持RK3588）
common_algorithm	音频3A算法、移动检测、遮挡检测
isp	isp图像处理算法
iva	智能视频分析算法（只支持RV1106/RV1103/RK3588）
ive	智能视频分析硬件加速引擎（只支持RV1106/RV1103）
libdrm	Direct Rendering Manager
libv4l	video4linux2设备用户层接口
mali	GPU firmware以及库文件（注：只支持RK3588，mali_csffw.bin必须放在/lib/firmware目录）
mpp	编解码接口，给rkmedia和rockit调用，不建议直接调用mpp
rga	RGA是一个独立的2D硬件加速器
rkmedia	多媒体接口（适用RV1126/RV1109平台）
rockit	多媒体接口（推荐）
sysutils	外设参考接口（ADC/GPIO/TIME/WATCHDOG）
samples	测试例程
out	media 编译输出目录

### 2.12.3 App目录说明

App目录project/app

project/app子目录	说明
rkadk	rkadk封装基础通用组件，如录像、拍照、播放、预览等，简化了应用开发难度
rkfsmk_release	优化存储相关的库（包含FAT32格式化，FAT32文件系统修复，MP4文件修复接口）

## 2.13 镜像存放目录说明

编译后的烧录镜像在output/image目录下

固件镜像名称	说明
download.bin	烧录工具升级通讯的设备端程序，只会下载到板子内存
env.img	包含分区表和启动参数（SDK默认的env分区放在0地址）
idblock.img	loader镜像（包含DDR初始化），负责加载U-Boot
uboot.img	uboot镜像
boot.img	Linux内核镜像
rootfs.img	根文件系统镜像
oem.img	oem镜像（可选）
userdata.img	userdata镜像（可选）

## 2.14 调试工具

SDK支持adb和tftp工具用于PC和单板端文件传输。

### 2.14.1 通过网络tftp传输文件

```
### PC端的IP地址192.168.1.159
### 从PC端tftp服务器下载文件到单板
cd /tmp
tftp 192.168.1.159 -g -r test-file

### 从单板上传文件到PC端tftp服务器
tftp 192.168.1.159 -p -l test-file
```

注：tftp服务器配置请参考[如何在U-Boot终端下使用tftp进行分区升级](#)

### 2.14.2 通过网络ADB调试

```
### 获取EVB板的IP地址192.168.1.159
adb connect 192.168.1.159

adb devices
List of devices attached
192.168.1.159:5555      device

### adb登陆EVB板子调试
adb -s 192.168.1.159:5555 shell

### 从PC端上传文件test-file到EVB板的目录/userdata
```

```
adb -s 192.168.1.159:5555 push test-file /userdata/

### 下载EVB板上的文件/userdata/test-file到PC端
adb -s 192.168.1.159:5555 pull /userdata/test-file test-file
```

### 2.14.3 通过Telnet调试

```
### 设置IP地址
udhcpd -i eth0

### 在板端运行telnetd
telnetd
```

```
### 获取EVB板的IP地址192.168.1.159
### 在PC端运行telnet
telnet 192.168.1.159
### username: root
### password: rockchip
```

### 2.14.4 通过串口调试

芯片	PC端串口配置说明
RV1106/RV1103	波特率：115200，数据位：8，奇偶校验：无，停止位：1，流控制：无
RV1126/RV1109	波特率：1500000，数据位：8，奇偶校验：无，停止位：1，流控制：无
RK3588	波特率：1500000，数据位：8，奇偶校验：无，停止位：1，流控制：无

## 3. 文档说明

```
docs/
├── zh ----- SDK的中文文档
│   ├── bsp
│   ├── isp
│   ├── iva
│   ├── media
│   ├── security
│   └── ipc/Rockchip_Quick_Start_Linux_IPC_SDK_CN.pdf --- SDK中文Quick Start文档
└── en ----- SDK的英文文档
    ├── bsp
    ├── isp
    ├── iva
    ├── media
    ├── security
    └── ipc/Rockchip_Quick_Start_Linux_IPC_SDK_EN.pdf --- SDK英文Quick Start文档
```

## 4. 工具说明

随 Rockchip Linux IPC SDK 发布的工具，用于开发调试阶段及量产阶段。工具版本会随SDK更新不断更新，如有工具上的疑问及需求，请联系我们的 FAE 窗口[fae@rock-chips.com](mailto:fae@rock-chips.com)。

Rockchip Linux IPC SDK 中在 tools 目录下附带了linux（Linux操作系统环境下使用工具）、windows（Windows操作系统环境下使用工具）2个版本。

- Windows工具

工具说明文档：tools/windows/ToolsRelease.txt

工具名称	工具用途
SocToolKit	固件升级工具
DriverAssitant	驱动安装工具

- Linux工具

工具说明文档：tools/linux/ToolsRelease.txt

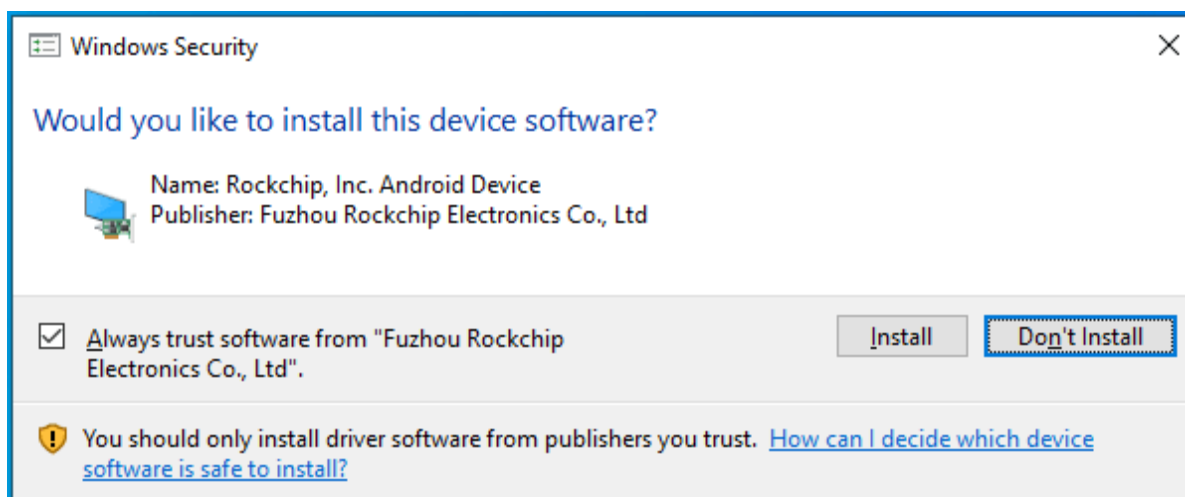
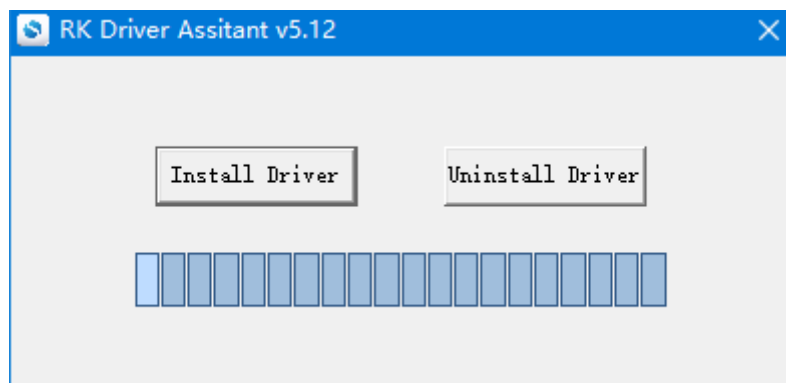
工具名称	工具用途
SocToolKit	固件升级工具
Linux_Upgrade_Tool	命令行烧录工具（只支持USB）

### 4.1 驱动安装工具

Rockchip USB 驱动安装助手存放在 `<SDK>/tools/windows/DriverAssitant_<版本>.zip`。支持 win7\_64, win10\_64等操作系统。

安装步骤如下：





## 4.2 固件烧录

- 切换烧录模式的方法

按住按键“Update”不放并按下复位键“RESET”后松手，就能进入Maskrom模式。

按住按键“Recovery”不放并按下复位键“RESET”后松手，就能进入Loader模式。

**注：RV1106/RV1103不支持Loader模式。**

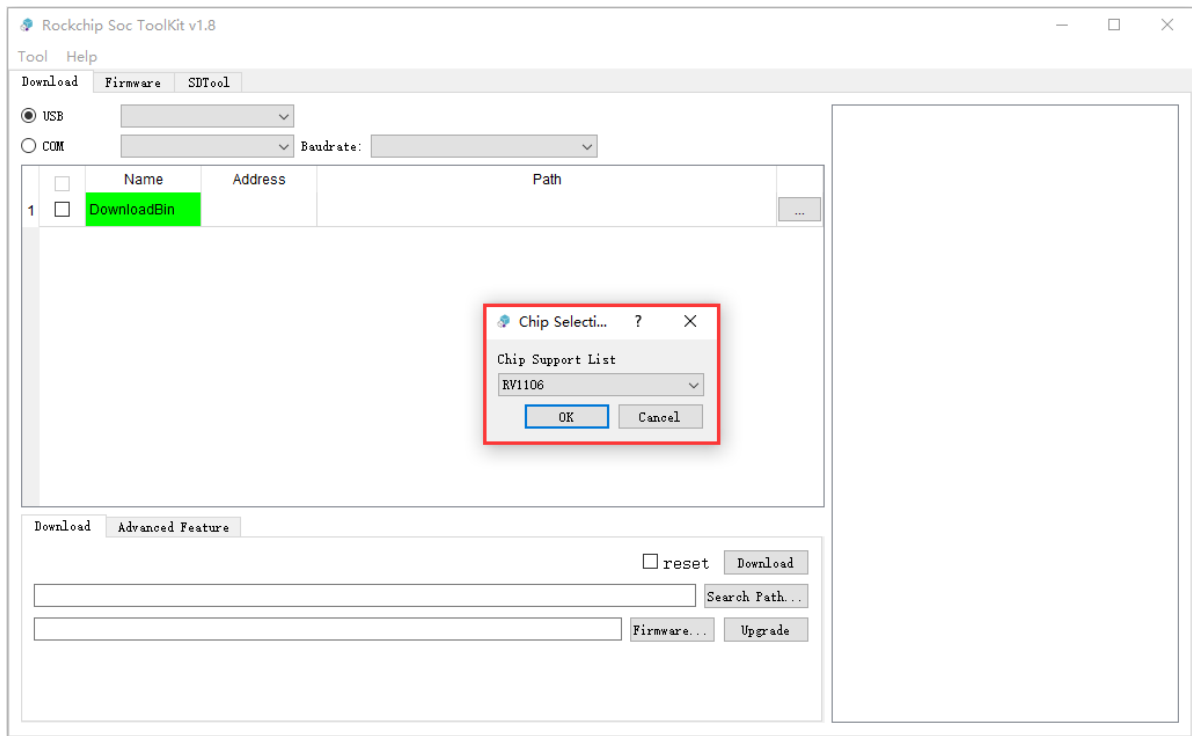
- SDK 提供 Windows 烧写工具，工具位于工程根目录。

如果单板已烧录过固件，可以进U-Boot进行升级固件。详细操作请参考以下章节：

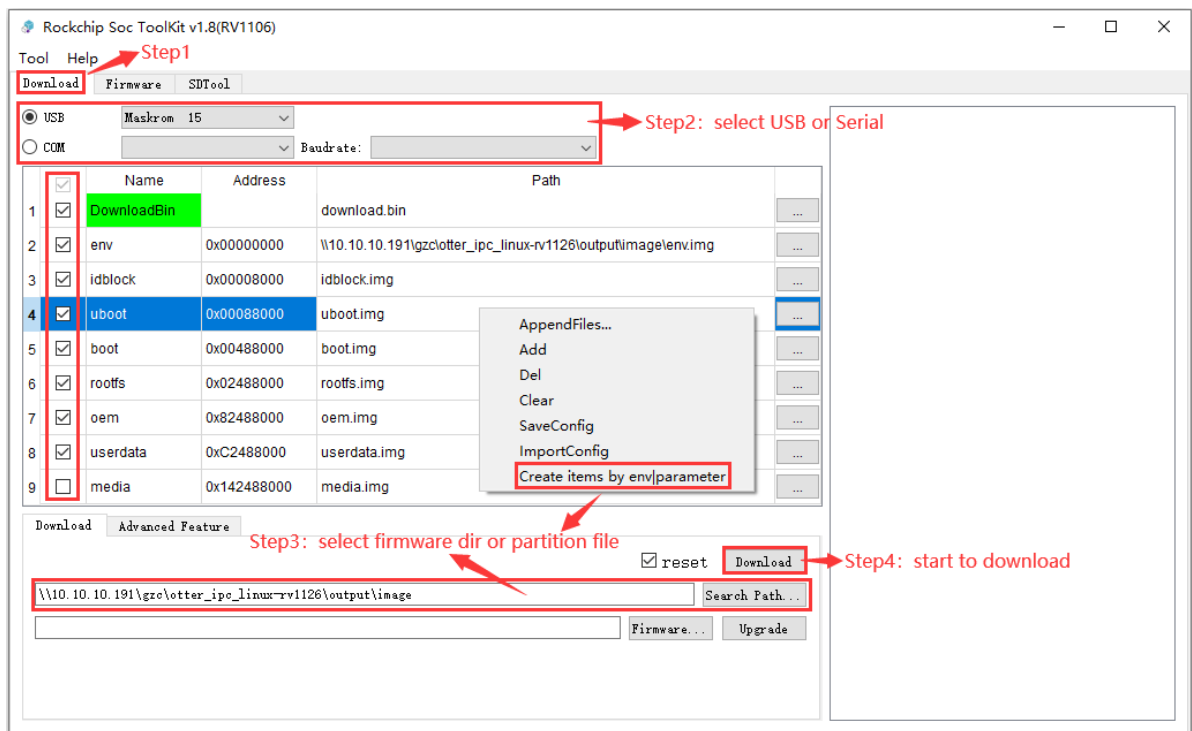
[如何在U-Boot终端下使用tftp进行分区升级](#)

[如何在U-Boot终端下使用SD卡进行分区升级](#)

```
<SDK>/Tools/windows/SocToolKit/SocToolKit.exe
```



注：目前只有RV1106、RV1103支持串口烧录。



## 4.3 update.img相关工具

### 4.3.1 打包

SDK在一键自动编译（`./build.sh`）时，会自动将需要烧录的固件打包成update.img，存放在 `<SDK>/output/image` 目录。同时，也可以运行以下命令，手动打包上述目录中的固件：

```
./build.sh updateimg
```

如需自定义固件目录等，可以手动运行打包脚本，查看帮助（-h），或输入相应选项：

```
<SDK>/tools/linux/Linux_Pack_Firmware/mk-update_pack.sh -id <RK_CHIP> -i  
<IMAGE_DIR>
```

### 4.3.2 解包

该功能需手动运行，可将 `<SDK>/output/image/update.img` 解包为分立固件，存放在 `<SDK>/output/image/unpack` 目录下。解包命令如下：

```
./build.sh unpackimg
```

如需自定义固件路径等，可以手动运行解包脚本，查看帮助（-h），或输入相应选项：

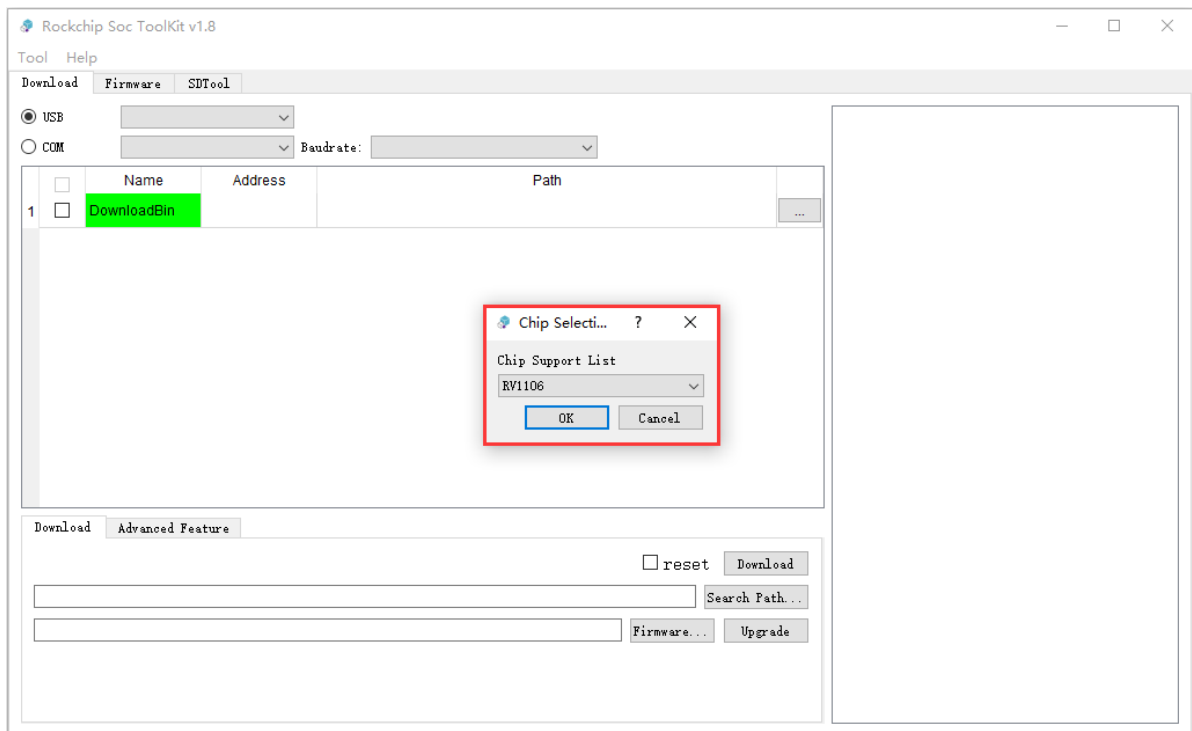
```
<SDK>/tools/linux/Linux_Pack_Firmware/mk-update_unpack.sh -i <IMAGE_PATH> -o  
<UNPACK_DIR>
```

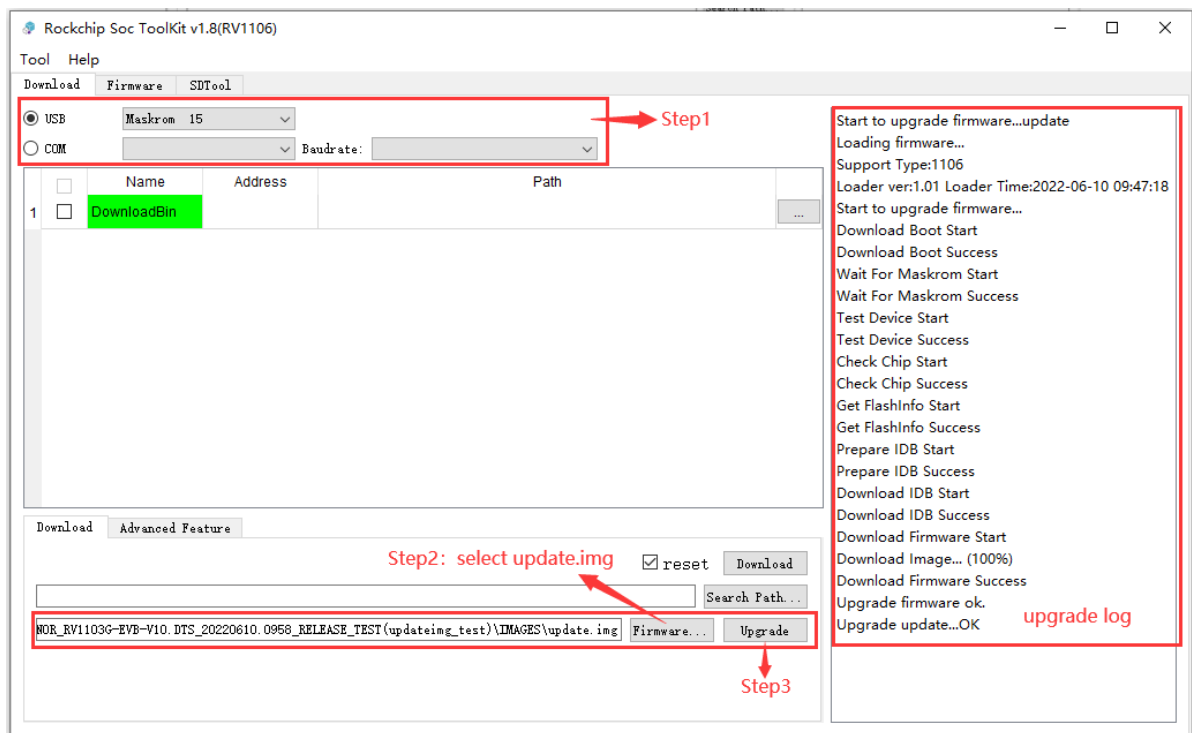
### 4.3.3 update.img烧写

- SDK 提供 Windows 烧写工具，工具位于工程根目录。

```
<SDK>/Tools/windows/SocToolKit/SocToolKit.exe
```

**注：需要1.8或更高版本的SocToolKit工具才能支持update.img烧写功能。**





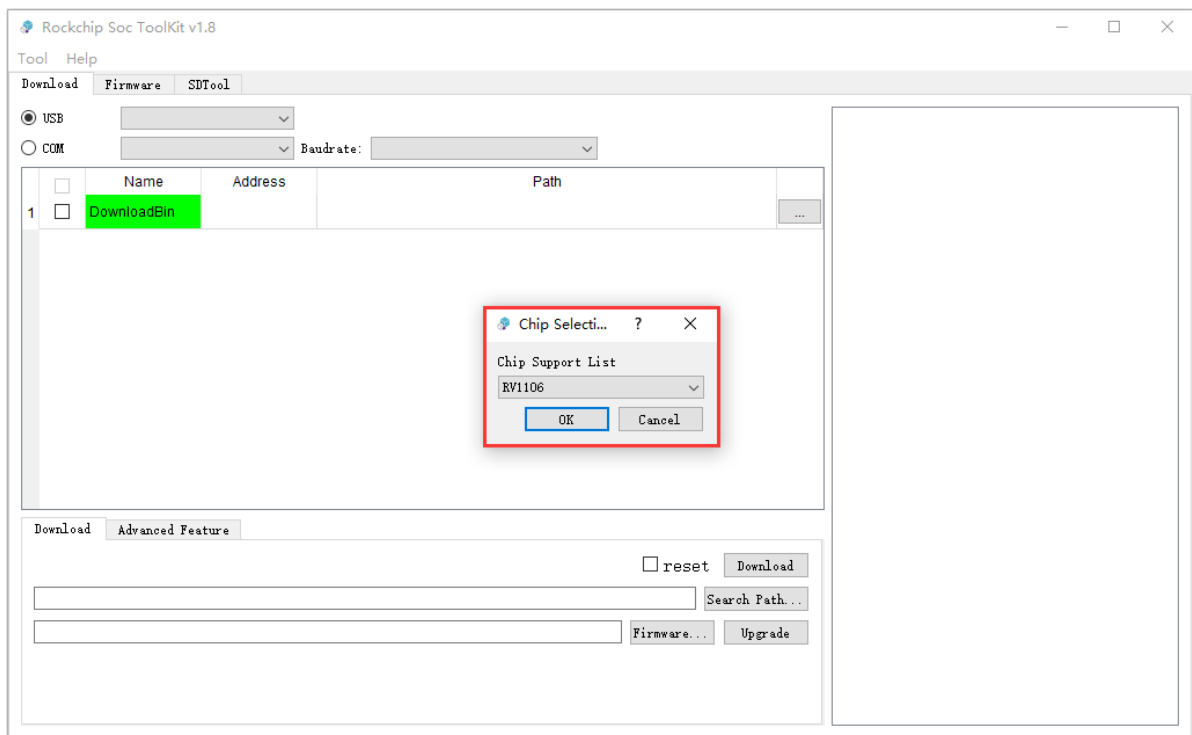
## 4.4 SD升级启动制作工具

- SDK 提供 Windows SD卡升级启动制作工具，工具位于工程根目录。

注：SD启动卡只支持用eMMC固件。

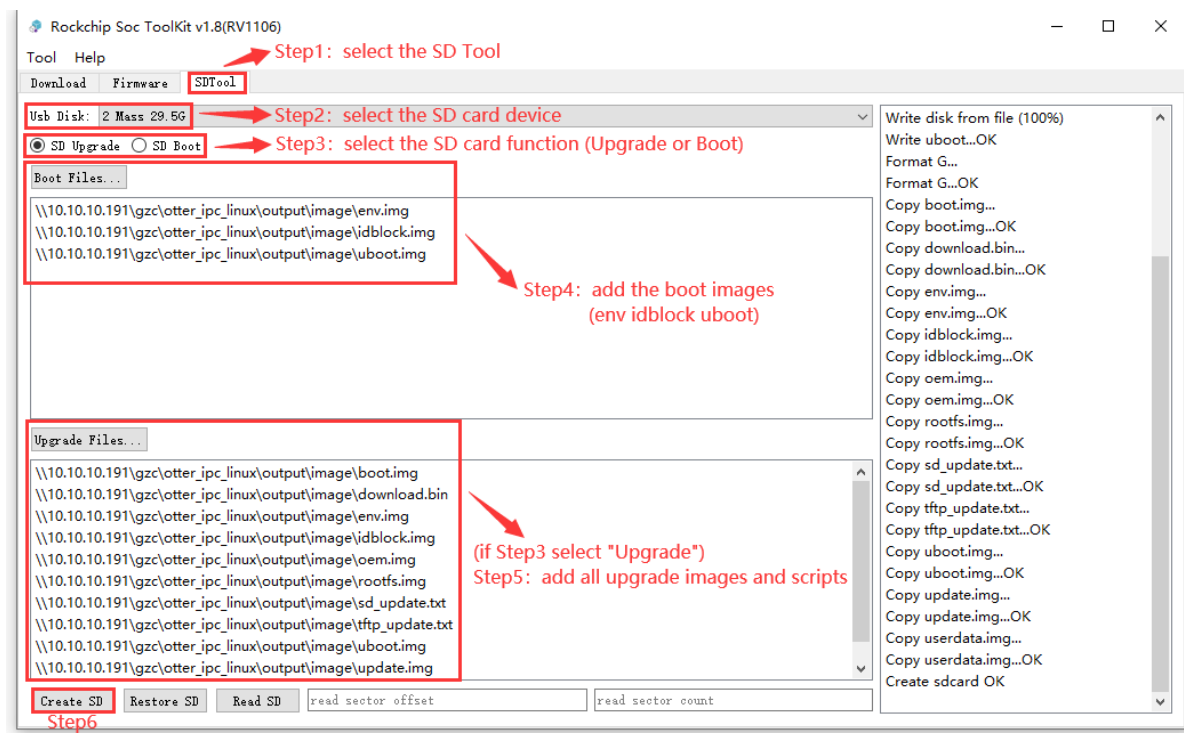
```
<SDK>/Tools/windows/SocToolKit/SocToolKit.exe
```

注1：需要1.7或更高版本的SocToolKit工具才能支持SD卡升级启动功能。



注2：此功能需要用户以管理员身份运行SocToolKit.exe（开启工具时会默认询问）。





- 将制作成功的SD卡插入设备后重启，设备将优先进入SD卡中的U-Boot终端。
- 若SD卡有升级功能，则会自动升级设备。
- 升级完成后，需要拔掉SD卡，再重启设备，方能进入设备系统中。

## 4.5 工厂固件说明

制作工厂固件需要使用programmer\_image\_tool工具，该工具位于

`<SDK>/tools/linux/SocToolKit/bin/linux` 目录。输入的镜像为update.img，详细内容见[update.img 相关工具](#)。

programmer\_image\_tool的文档在

`<SDK>/tools/linux/SocToolKit/bin/linux/programmer_image_tool.pdf`。

SDK支持编译部分flash类型的工厂固件，生成的固件存放在 `<SDK>/output/image/factory` 目录下。

其他flash类型的固件可以参考文档

**Rockchip\_Developer\_Guide\_Linux\_Nand\_Flash\_Open\_Source\_Solution\_CN.pdf**，使用工具 **programmer\_image\_tool**自行生成。

- 对于SPI NAND和SLC NAND的flash，需要配置block size和page size：

在对应的BoardConfig文件里配置如下：

```
export RK_NAND_BLOCK_SIZE=0x20000
export RK_NAND_PAGE_SIZE=2048

# For SLC NAND
export RK_NAND_OOB_SIZE=128
```

- SDK编译命令：

```
# 先编译代码
./build.sh all

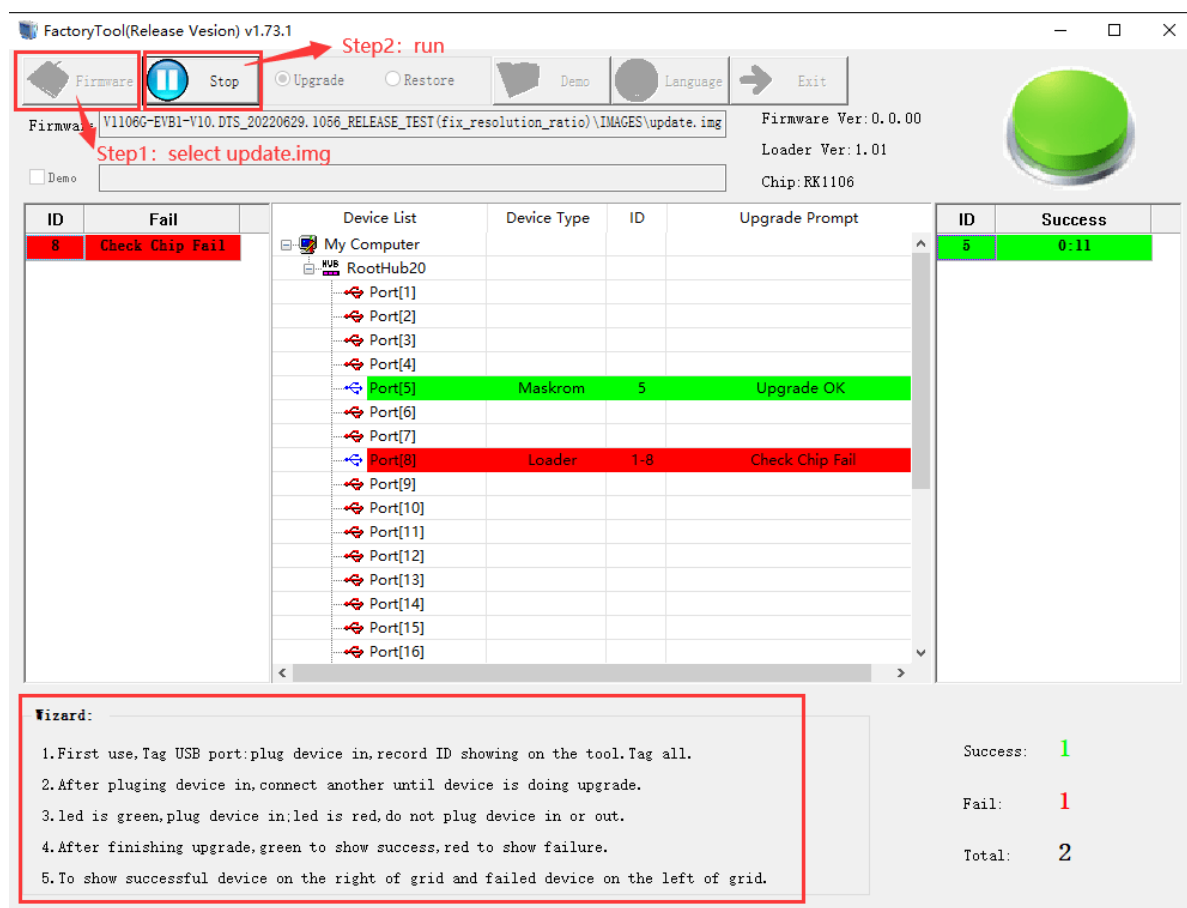
# 打包 update.img
./build.sh updateimg

# 生成工厂固件: output/image/factory
./build.sh factory
```

更多存储相关问题调试及处理请参考: <https://redmine.rock-chips.com/documents/108>

## 4.6 量产升级工具

量产升级需要使用FactoryTool工具, 该工具位于 `<SDK>/tools/windows` 目录。输入的镜像为 update.img, 详细内容见[update.img相关工具](#)。



开始运行升级后, 只要电脑和设备通过USB连接, 并且设备进入Maskrom或Loader模式, 就能自动开始升级, 不需要其他操作。

其他内容可见工具下方的友情提醒。

## 5. IPC Linux SDK Q&A

## 5.1 如何修改分区表和增加自定义分区以及分区可读写说明

### 5.1.1 存储介质以及文件系统类型说明

存储介质	支持的可读可写文件系统格式	支持的只读文件系统格式
eMMC	ext4	squashfs
spi nand 或slc nand	ubifs	squashfs
spi nor	jffs2	squashfs

文件系统格式	制作烧录镜像文件的脚本
ext4	output/out/sysdrv_out/pc/mkfs_ext4.sh
jffs2	output/out/sysdrv_out/pc/mkfs_jffs2.sh
ubifs	output/out/sysdrv_out/pc/mkfs_ubi.sh
squashfs	output/out/sysdrv_out/pc/mkfs_squashfs.sh

注：Nand Flash硬件有不同的page size和block size，需要烧录对应的镜像文件，所以mkfs\_ubi.sh默认会打包出不同page size和block size的烧录镜像。

详细的Nand Flash说明可以参考文档

Rockchip\_Developer\_Guide\_Linux\_Nand\_Flash\_Open\_Source\_Solution\_CN.pdf

### 5.1.2 分区表说明

SDK使用env分区设置分区表，分区表信息配置在 `<SDK>/project/cfg/BoardConfig*.mk` 里的 `RK_PARTITION_CMD_IN_ENV` 参数中。

分区表以字符串的形式保存在配置中，以下是各存储介质的分区表例子。

存储介质	分区表
eMMC	<code>RK_PARTITION_CMD_IN_ENV="32K(env),512K@32K(idblock),4M(uboot),32M(boot),2G(rootfs),-(userdata)"</code>
spi nand 或slc nand	<code>RK_PARTITION_CMD_IN_ENV="256K(env),256K@256K(idblock),1M(uboot),8M(boot),32M(rootfs),-(userdata)"</code>
spi nor	<code>RK_PARTITION_CMD_IN_ENV="64K(env),128K@64K(idblock),128K(uboot),3M(boot),6M(rootfs),-(userdata)"</code>

每个分区的格式为：`<size>[<offset>] (part-name)`，其中，分区大小和分区名是必须的，偏移量则因情况而定（见以下注意事项第三点）。

在配置分区表时有以下几点注意事项：

1. 分区之间用英文逗号","隔开。
2. 分区大小的单位有：K/M/G/T/P/E，不区分大小写，无单位则默认为byte；"-"表示该分区大小为剩余容量。

3. 第一个分区若从0x0地址开始时，不加偏移量，反之，则必须添加偏移量。后续分区任意选择是否添加偏移量。
4. idblock分区的偏移固定的，**请勿修改**。
5. 不建议修改env分区名。（如果要修改env地址和大小，需要修改对应U-Boot的defconfig 配置 CONFIG\_ENV\_OFFSET和CONFIG\_ENV\_SIZE，重新生成固件，擦除板端0地址的env数据，再烧录新的固件）

### 5.1.3 增加自定义分区

以下是eMMC增加一个大小为64MB的可读写分区custom\_part的例子。

- 使用上述方法（[分区表说明](#)）修改对应板级配置中的分区表参数，以此例为：64M(custom\_part)
- 修改 <SDK>/project/cfg/BoardConfig\*.mk 里配置增加分区 `RK_PARTITION_CMD_IN_ENV` 和 `RK_PARTITION_FS_TYPE_CFG` 分区挂载

```
# config partition's filesystem type (squashfs is readonly)
# emmc:      squashfs/ext4
# nand:      squashfs/ubifs
# spi nor:   squashfs/jffs2
# RK_PARTITION_FS_TYPE_CFG format:
#      AAAA@/BBBB/CCCC@DDDD
#      AAAA -----> partition name
#      /BBBB/CCCC ----> partition mount point
#      DDDD -----> partition filesystem type
(squashfs/ext4/ubifs/jffs2)
export
RK_PARTITION_FS_TYPE_CFG=rootfs@IGNORE@ext4,custom_part@/opt/custom_part@ext4

# config partition in environment
# RK_PARTITION_CMD_IN_ENV format:
#      <partdef>[,<partdef>]
#      <partdef> := <size>[@<offset>] (part-name)
#
export
RK_PARTITION_CMD_IN_ENV="32K(env),512K@32K(idblock),4M(uboot),32M(boot),2G(rootfs),64M(custom_part),-(userdata)"
```

- 制作custom\_part分区镜像

```
mkdir -p custom_part
./output/out/sysdrv_out/pc/mkfs_ext4.sh custom_part custom_part.img 64*0x100000
# 注意：使用SDK默认的挂载脚本时，分区镜像文件名需要以分区名称命名
#      例如：分区名称是custom_part，分区镜像名称custom_part.img
```

- 在文件系统创建custom\_part分区的挂载目录

```
mkdir -p output/out/rootfs_glibc_rk3588/opt/custom_part
```

- 重新打包根文件系统（rootfs.img） `./build.sh firmware`
- 烧录rootfs.img、env.img以及custom\_part.img

## 5.1.4 oem分区挂载说明

挂载oem分区需要进行如下配置：

- 在分区表中添加oem分区（详见[分区表说明](#)），例如：

```
RK_PARTITION_CMD_IN_ENV="32K(env),512K@32K(idblock),4M(uboot),32M(boot),2G(rootfs),64M(oem),-(userdata)"
```

- 在文件系统类型的配置里添加oem相关配置（详见[增加自定义分区](#)），例如：

```
RK_PARTITION_FS_TYPE_CFG=rootfs@IGNORE@ext4,oem@/oem@ext4
```

- 开启如下配置：

```
# enable oem partition to install app
export RK_BUILD_APP_TO_OEM_PARTITION=y
```

## 5.2 如何使用Recovery

Recovery模式是在设备上多一个Recovery分区，该分区由kernel+resource+ramdisk组成，主要用于升级操作。u-boot会根据misc分区存放的字段来判断将要引导的系统是Normal系统还是Recovery系统。由于系统的独立性，所以Recovery模式能保证升级的完整性，即升级过程被中断，如异常掉电，升级仍然能继续执行。

本章主要介绍了通过userdata分区或SD卡执行Recovery本地升级程序的流程及技术细节，下面是使用Recovery的方法。

- 在 `<SDK>/project/cfg/BoardConfig*.mk` 中添加如下配置：

```
# Kernel defconfig fragment
# export RK_KERNEL_DEFCONFIG_FRAGMENT="rv1106-evb.config"
export RK_KERNEL_DEFCONFIG_FRAGMENT="rv1106-evb.config rv1106-recovery.config"

#misc image
export RK_MISC=recovery-misc.img

# enable build recovery
export RK_ENABLE_RECOVERY=y

# select image to update
# export RK_OTA_RESOURCE="uboot.img boot.img rootfs.img userdata.img"
```

注：不开启RK\_OTA\_RESOURCE则默认打包uboot.img、boot.img、rootfs.img。

- 修改分区表

在分区表中添加 `misc` 和 `recovery` 两个分区，分区大小和顺序可根据实际需求在合理范围内进行调整。

- 编译Recovery

```
./build.sh recovery
```

- 编译需要升级的固件

手动编译RK\_OTA\_RESOURCE包含的固件，若未开启则编译默认分区固件（固件编译方式见前文）。

- 打包OTA升级包

```
./build.sh ota
```

- 拷贝OTA升级包

将生成的OTA升级包（<SDK>/output/image/update\_ota.tar）拷贝至SD卡根目录下。

**注：若使用程序下载OTA升级包至板端userdata分区，则不需要这一步。**

- 设备端插上SD卡

**注：若使用程序下载OTA升级包至板端userdata分区，则不需要这一步。**

- 设备端进入Recovery系统开始升级

```
reboot recovery  
# 注：该命令需要使用SDK里的busybox或打上对应补丁
```

设备端（板端）输入上述代码进入Recovery系统，即可开始升级。

**注：升级完成后设备将会重启进入Normal系统，失败则停留在Recovery系统并打印log。未检索到SD卡或升级包也将会重启进入Normal系统。**

## 5.3 如何在U-Boot终端下使用tftp进行分区升级

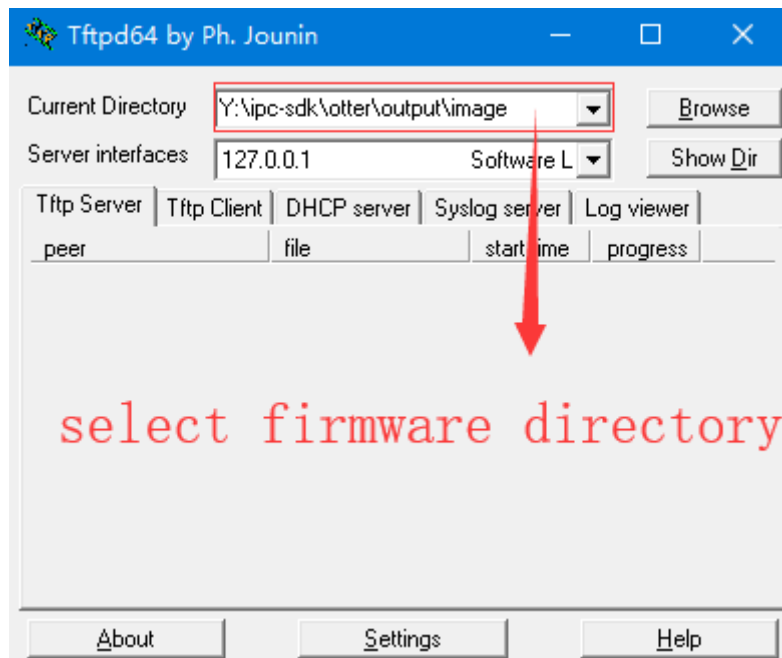
tftp升级文件会随固件一同编译在<SDK>/output/image/目录下，文件名为tftp\_update.txt，使用方法如下：

- 配置tftp服务器

Tftpd64下载地址 <https://pjo2.github.io/tftpd64>

**注意：**

- 1、使用Tftpd64软件需要遵守相关的开源协议
- 2、使用Tftpd64带来的所有的法律风险以及后果全部由客户自己承担



- 将升级文件 `tftp_update.txt` 和所有后缀名为 `.img` 的固件放进服务器指定的目录下

(注：slc nand暂时不支持以下载固件的方式升级 `idblock` 分区。)

- U-Boot终端下设置IP地址

```
=> setenv ipaddr 192.168.1.111
=> setenv serverip 192.168.1.100
=> saveenv
Saving Environment to envf...
=>
```

以上IP地址仅供参考，请根据实际情况自行设置，保证客户端与服务器在同一网段即可。

- U-Boot终端下运行升级指令 `tftp_update`

```
=> tftp_update
ethernet@fffc40000 Waiting for PHY auto negotiation to complete. done
Using ethernet@fffc40000 device
TFTP from server 192.168.1.100; our IP address is 192.168.1.111
Filename 'tftp_update.txt'.
Load address: 0x3be24c00
Loading: *.*#
      203.1 KiB/s
done
Bytes transferred = 1250 (4e2 hex)
...
```

## 5.4 如何在U-Boot终端下使用SD卡进行分区升级

SD卡升级文件会随固件一同编译在 `<SDK>/output/image/` 目录下，文件名为 `sd_update.txt`，使用方法如下：

- 将升级文件 `sd_update.txt` 和所有后缀名为 `.img` 的固件放进SD卡根目录下

(注：1. slc nand暂时不支持以下载固件的方式升级 `idblock` 分区；2. SD卡仅支持FAT格式的文件系统。)

- 设备端插上SD卡
- U-Boot终端下运行升级指令 `sd_update`

```
=> sd_update
PartType: ENV
reading sd_update.txt
1511 bytes read in 2 ms (737.3 KiB/s)
...
```

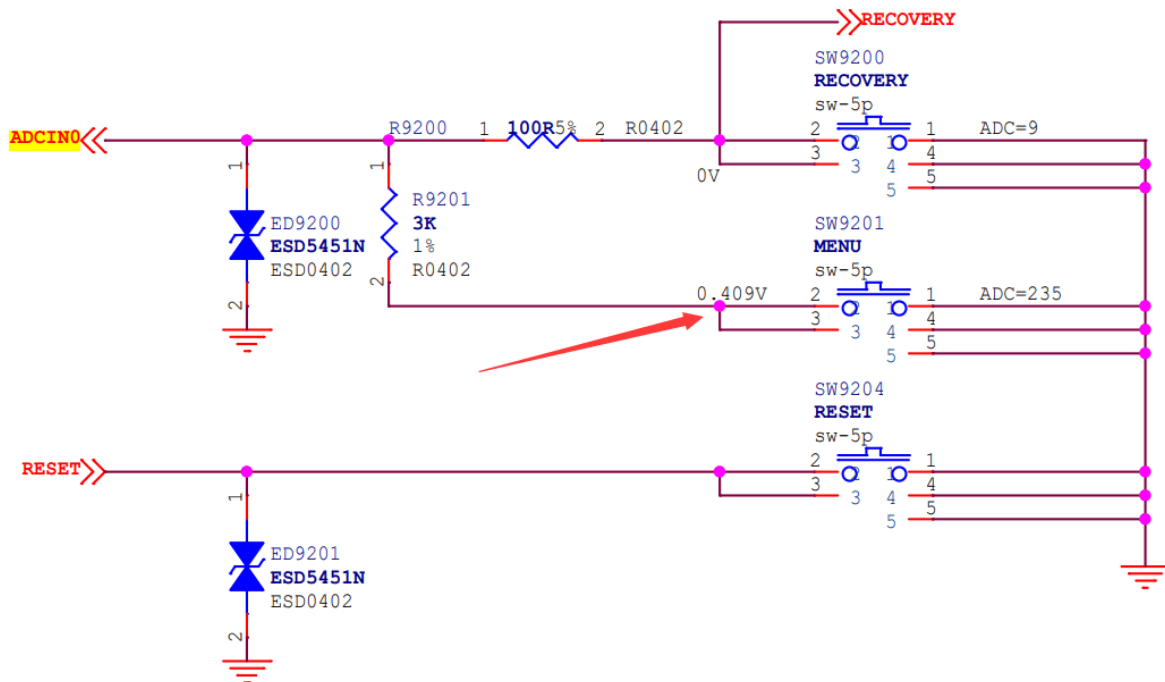
- 升级完成后重启设备

## 5.5 如何在U-Boot阶段通过按键触发SD卡升级功能

该功能需要ADC功能。U-Boot配置好相应dts与defconfig后，通过按住按键并重启的方式，在U-Boot阶段触发 `sd_update` 升级设备分区。具体配置流程如下所示：

- 根据硬件电路图，获取按键电压值，用于ADC配置

以RV1106 EVB1 V10为例，根据电路图得到选定key的电压值为0.409V。



- 配置相应dts的adc-keys节点

以RV1106 EVB1 V10为例，修改rv1106-evb.dts中的adc-keys。

```
diff --git a/arch/arm/dts/rv1106-evb.dts b/arch/arm/dts/rv1106-evb.dts
index 344558f8f9..10caf5b0da 100644
--- a/arch/arm/dts/rv1106-evb.dts
+++ b/arch/arm/dts/rv1106-evb.dts
@@ -27,6 +27,13 @@
+
+         label = "volume up";
+         press-threshold-microvolt = <1750>;
+     };
+
+     volumedown-key {
+         u-boot,dm-pre-reloc;
+         linux,code = <KEY_VOLUMEDOWN>;
```



```

+             label = "volume down";
+             press-threshold-microvolt = <409000>;
+             };
+     };
};

```

`KEY_VOLUMEDOWN` 为头文件 `<u-boot>/include/linux/input.h` 中定义的键值。键值可任意选择，但需要与defconfig中的键值参数对应（见之后配置defconfig的步骤）。

`press-threshold-microvolt` 为电路图中key的电压值，以uV为单位。

- 配置相应defconfig

以RV1106 EVB1 V10为例，添加配置 `CONFIG_ROCKCHIP_CMD`。

```

diff --git a/configs/rv1106_defconfig b/configs/rv1106_defconfig
index e797553435..ca4baacac8 100644
--- a/configs/rv1106_defconfig
+++ b/configs/rv1106_defconfig
@@ -10,6 +10,8 @@ CONFIG_ROCKCHIP_SPL_RESERVE_IRAM=0x0
 CONFIG_ROCKCHIP_FIT_IMAGE=y
 CONFIG_USING_KERNEL_DTB_V2=y
 CONFIG_ROCKCHIP_FIT_IMAGE_PACK=y
+CONFIG_ROCKCHIP_CMD="sd_update 114"
 CONFIG_SPL_SERIAL_SUPPORT=y
 CONFIG_SPL_DRIVERS_MISC_SUPPORT=y
 CONFIG_TARGET_EVB_RV1106=y

```

`sd_update` 为按键触发的指令，这里设置为SD卡升级分区指令，该功能下无需修改。

`114` 为键值 `KEY_VOLUMEDOWN` 对应的参数，如果键值变更了这里需要随之修改。

## 5.6 插入SD卡检测不到设备

对应内核的defconfig需要打开支持SD卡的配置。

```

CONFIG_MMC_DW=y
CONFIG_MMC_DW_PLTFM=y
CONFIG_MMC_DW_ROCKCHIP=y

```

增加vfat文件系统支持

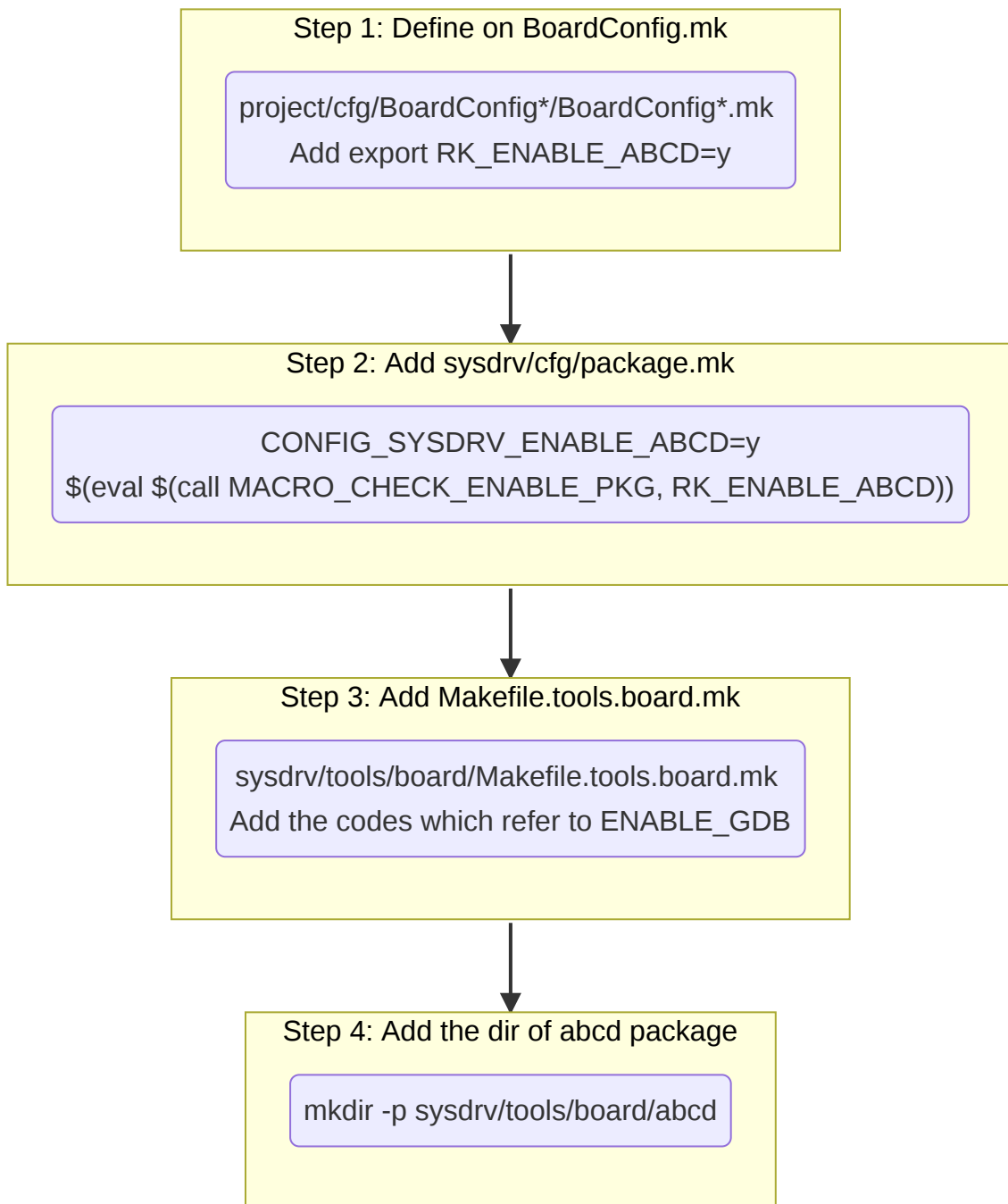
```

CONFIG_VFAT_FS=y
CONFIG_MSDFS_PARTITION=y

```

## 5.7 如何添加第三方库到sysdrv目录编译

- sysdrv/tools/board运行在板子上的程序，这里介绍如何添加一个第三方案程序（例如第三方案程序名为ABCD）。



- touch sysdrv/tools/board/abcd/Makefile

```
# sysdrv/tools/board/abcd/Makefile reference code
ifeq ($(SYSDRV_PARAM), )
SYSDRV_PARAM:=../../../../../Makefile.param
include $(SYSDRV_PARAM)
endif

export LC_ALL=C
SHELL:=/bin/bash

CURRENT_DIR := $(shell pwd)
PKG_TARBALL := abcd.tar.xz
PKG_NAME := abcd
PKG_BIN := out

all:
rm -rf $(CURRENT_DIR)/$(PKG_NAME); \
```

```

tar -xf $(PKG_TARBALL); \
mkdir -p $(CURRENT_DIR)/$(PKG_NAME)/$(PKG_BIN); \
mkdir -p $(CURRENT_DIR)/$(PKG_BIN); \
pushd $(CURRENT_DIR)/$(PKG_NAME)/; \
    ./configure --host=$(SYSDRV_CROSS) \
    --target=$(SYSDRV_CROSS) CFLAGS="$(SYSDRV_CROSS_CFLAGS)" \
    LDFLAGS="$(SYSDRV_CROSS_CFLAGS)" \
    --prefix=$(CURRENT_DIR)/$(PKG_NAME)/$(PKG_BIN); \
    make -j$(SYSDRV_JOBS) > /dev/null || exit -1; \
    make install > /dev/null; \
popd; )
$(call MAROC_COPY_PKG_TO_SYSDRV_OUTPUT, $(SYSDRV_DIR_OUT_ROOTFS),
$(PKG_BIN))

clean: distclean

distclean:
    -rm -rf $(PKG_NAME) $(PKG_BIN)

```

- Test and build

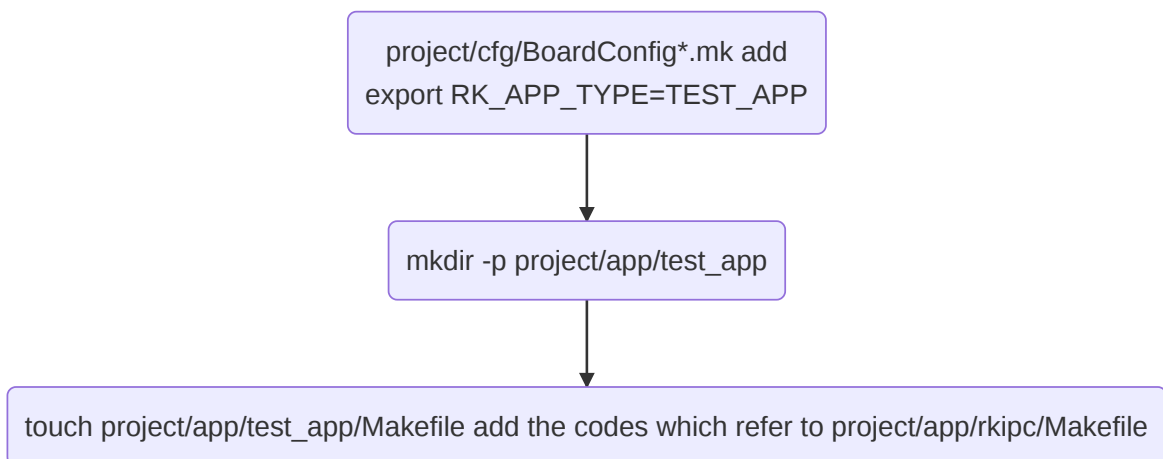
```

cd sysdrv/tools/board/abcd/
make
cd ../../..
make

```

## 5.8 如何在project/app里增加新应用程序

- 以增加应用程序test\_app为例



```

# project/app/test_app/Makefile reference code

ifeq ($(APP_PARAM), )
APP_PARAM:=../Makefile.param
include $(APP_PARAM)
endif

export IC_ALL=C

```

```

SHELL:=/bin/bash

CURRENT_DIR := $(shell pwd)

PKG_NAME := test_app
PKG_BIN ?= out
PKG_BUILD ?= build

RK_APP_CFLAGS = -I $(RK_APP_MEDIA_INCLUDE_PATH)

RK_APP_LDFLAGS = -L $(RK_APP_MEDIA_LIBS_PATH)

RK_APP_OPTS += -Wl,-rpath-
link,$(RK_APP_MEDIA_LIBS_PATH):$(RK_APP_PATH_LIB_INCLUDE)/root/usr/lib
PKG_CONF_OPTS += -DCMAKE_C_FLAGS="$(RK_APP_CFLAGS) $(RK_APP_LDFLAGS)
$(RK_APP_OPTS)" \
    -DCMAKE_CXX_FLAGS="$(RK_APP_CFLAGS) $(RK_APP_LDFLAGS)
$(RK_APP_OPTS)"

# define project/cfg/BoardConfig*.mk
ifneq ($(findstring $(RK_APP_TYPE),TEST_APP),)
PKG_TARGET := test_app_build
endif

ifeq ($(PKG_BIN),)
$(error ### $(CURRENT_DIR): PKG_BIN is NULL, Please Check !!!)
endif

all: $(PKG_TARGET)
    @echo "build $(PKG_NAME) done"

test_app_build:
    rm -rf $(PKG_BIN) $(PKG_BUILD); \
    mkdir -p $(PKG_BIN); \
    mkdir -p $(PKG_BUILD); \
    pushd $(PKG_BUILD)/; \
        rm -rf CMakeCache.txt; \
        cmake $(CURRENT_DIR)/$(PKG_NAME) / \
        -DCMAKE_C_COMPILER=$(RK_APP_CROSS)-gcc \
        -DCMAKE_CXX_COMPILER=$(RK_APP_CROSS)-g++ \
        -DCMAKE_INSTALL_PREFIX="$(CURRENT_DIR)/$(PKG_BIN)" \
        $(PKG_CONF_OPTS) ;\
        make -j$(RK_APP_JOBS) || exit -1; \
        make install; \
    popd;
    $(call MAROC_COPY_PKG_TO_APP_OUTPUT, $(RK_APP_OUTPUT), $(PKG_BIN))

clean:
    @rm -rf $(PKG_BIN) $(PKG_BUILD)

distclean: clean

```

- Test and build

```

./build.sh clean app
./build.sh app

```

## 5.9 如何下载NPU模型转换工具以及runtime库

SDK没有带NPU模型转换工具以及runtime，需要从以下github地址下载。

NPU模型转换工具下载地址：

```
https://github.com/rockchip-linux/rknn-toolkit2
```

NPU runtime库下载地址：

```
https://github.com/rockchip-linux/rknpu2
```

## 5.10 如何在板端修改系统CMA大小

- 板端查看CMA大小

```
# grep -i cma /proc/meminfo
CmaTotal:          24576 kB
CmaFree:           0 kB
```

- 进入U-Boot终端

重启设备后按住Ctrl+C，直到出现 => <INTERRUPT> 字段，表示已进入U-Boot终端。

- 查看U-Boot环境变量中的CMA大小

```
=> printenv
...
sys_bootargs=root=/dev/mtdblock4 rk_dma_heap_cma=24M rootfstype=squashfs
...
```

rk\_dma\_heap\_cma即为CMA大小。

**注：**此环境变量名为sys\_bootargs，rk\_dma\_heap\_cma仅为其参数之一，修改此环境变量时需要将sys\_bootargs后的所有内容看作一体。

- 修改环境变量、保存环境变量、重启设备

以上述环境变量为例，将CMA从24M修改为32M。

```
# setenv <name> <vars>
# saveenv
# reset
=> setenv sys_bootargs root=/dev/mtdblock4 rk_dma_heap_cma=32M
rootfstype=squashfs
=> saveenv
Saving Environment to envf...
=> reset
```

- 再次查看板端CMA大小

```
# grep -i cma /proc/meminfo
CmaTotal:          32768 kB
CmaFree:           0 kB
```

- 查看板端CMA使用情况

```
# cat /proc/rk_dma_heap/alloc_bitmap
Total: 67584 KiB
Used: 56136 KiB

FFFFFFFF FFFFFFFFFF FFFFFFFFFF FFFFFFFFFF FFFFFFFFFF FFFFFFFFFF FFFFFFFFFF FFFFFFFFFF
FFFFFFFF FFFFFFFFFF FFFFFFFFFF FFFFFFFFFF FFFFFFFFFF FFFFFFFFFF FFFFFFFFFF FFFFFFFFFF
...
FFFFFFFF FFFFFFFFFF FFFFFFFFFF FFFFFFFFFF FFFFFFFFFF FFFFFFFFFF FFFFFFFFFF FFFFFFFFFF
FFFFFFFF FFFFFFFFFF FFFFFFFFFF FFFFFFFFFF FFFFFFFFFF FFFFFFFFFF 0003FFFF 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

注：1个bit表示4KiB。

- 查看各个模块具体CMA的使用情况

```
# cat /proc/rk_dma_heap/dma_heap_info |grep Alloc
Alloc by (vmpi          ) [0x074c2000-0x074d1fff] 0x00010000 (64 KiB)
Alloc by (vmpi          ) [0x074b2000-0x074c1fff] 0x00010000 (64 KiB)
Alloc by (vmpi          ) [0x074b0000-0x074b1fff] 0x00002000 (8 KiB)
Alloc by (vmpi          ) [0x074ae000-0x074affff] 0x00002000 (8 KiB)
Alloc by (vmpi          ) [0x07391000-0x074adfff] 0x0011d000 (1140
KiB)
Alloc by (vmpi          ) [0x07390000-0x0739ffff] 0x00001000 (4 KiB)
Alloc by (vmpi          ) [0x07310000-0x0738ffff] 0x00080000 (512
KiB)
Alloc by (vmpi          ) [0x0730e000-0x0730ffff] 0x00002000 (8 KiB)
Alloc by (vmpi          ) [0x0730c000-0x0730dfff] 0x00002000 (8 KiB)
Alloc by (vmpi          ) [0x0730a000-0x0730bfff] 0x00002000 (8 KiB)
Alloc by (vmpi          ) [0x07308000-0x07309fff] 0x00002000 (8 KiB)
Alloc by (vmpi          ) [0x07306000-0x07307fff] 0x00002000 (8 KiB)
Alloc by (vmpi          ) [0x07246000-0x07305fff] 0x000c0000 (768
KiB)
Alloc by (vmpi          ) [0x07186000-0x07245fff] 0x000c0000 (768
KiB)
Alloc by (vmpi          ) [0x070c6000-0x07185fff] 0x000c0000 (768
KiB)
Alloc by (vmpi          ) [0x06a1d000-0x070c5fff] 0x006a9000 (6820
KiB)
Alloc by (vmpi          ) [0x0691d000-0x06a1cfff] 0x00100000 (1024
KiB)
Alloc by (vmpi          ) [0x068eb000-0x0691cfff] 0x00032000 (200
KiB)
Alloc by (vmpi          ) [0x068ea000-0x068eaffff] 0x00001000 (4 KiB)
Alloc by (vmpi          ) [0x068e2000-0x068e9fff] 0x00008000 (32 KiB)
Alloc by (vmpi          ) [0x068e1000-0x068e1fff] 0x00001000 (4 KiB)
Alloc by (vmpi          ) [0x068d9000-0x068e0fff] 0x00008000 (32 KiB)
Alloc by (vmpi          ) [0x0680f000-0x068d8fff] 0x000ca000 (808
KiB)
```

Alloc by (vmpi KiB)	) [0x0677a000-0x0680efff]	0x00095000	(596
Alloc by (vmpi KiB)	) [0x066e5000-0x06779fff]	0x00095000	(596
Alloc by (vmpi	) [0x066d3000-0x066e4fff]	0x00012000	(72 KiB)
Alloc by (vmpi	) [0x066c1000-0x066d2fff]	0x00012000	(72 KiB)
Alloc by (vmpi KiB)	) [0x06488000-0x066c0fff]	0x00239000	(2276
Alloc by (vmpi	) [0x06486000-0x06487fff]	0x00002000	(8 KiB)
Alloc by (vmpi KiB)	) [0x0643c000-0x06485fff]	0x0004a000	(296
Alloc by (vmpi	) [0x0643a000-0x0643bfff]	0x00002000	(8 KiB)
Alloc by (vmpi KiB)	) [0x063f0000-0x06439fff]	0x0004a000	(296
Alloc by (vmpi KiB)	) [0x05c34000-0x063effff]	0x007bc000	(7920
Alloc by (vmpi KiB)	) [0x05587000-0x05c2ffff]	0x006a9000	(6820
Alloc by (vmpi KiB)	) [0x04ede000-0x05586fff]	0x006a9000	(6820
Alloc by ((null) KiB)	) [0x04633000-0x046f0fff]	0x000be000	(760
Alloc by (ff660000.npu KiB)	) [0x044e3000-0x04632fff]	0x00150000	(1344
Alloc by (ff660000.npu	) [0x044d5000-0x044e2fff]	0x0000e000	(56 KiB)
Alloc by (ff660000.npu	) [0x044c7000-0x044d4fff]	0x0000e000	(56 KiB)
Alloc by (ff660000.npu KiB)	) [0x044ab000-0x044c6fff]	0x0001c000	(112
Alloc by (ff660000.npu KiB)	) [0x04473000-0x044aafff]	0x00038000	(224
Alloc by (ff660000.npu KiB)	) [0x0443b000-0x04472fff]	0x00038000	(224
Alloc by (ff660000.npu KiB)	) [0x043cb000-0x0443afff]	0x00070000	(448
Alloc by (ff660000.npu KiB)	) [0x03e03000-0x043cafff]	0x005c8000	(5920
Alloc by (vmpi	) [0x03e00000-0x03e02fff]	0x00003000	(12 KiB)
Alloc by (ffa00000.rkisp	) [0x046f1000-0x04705fff]	0x00015000	(84 KiB)
Alloc by (ffa00000.rkisp	) [0x04706000-0x04706fff]	0x00001000	(4 KiB)
Alloc by (ffa00000.rkisp	) [0x04707000-0x04707fff]	0x00001000	(4 KiB)
Alloc by (ffa00000.rkisp KiB)	) [0x04708000-0x04e3efff]	0x00737000	(7388
Alloc by (ffa00000.rkisp KiB)	) [0x04e3f000-0x04ecdfff]	0x0008f000	(572
Alloc by (rkisp-vir0	) [0x04ece000-0x04ed1fff]	0x00004000	(16 KiB)
Alloc by (rkisp-vir0	) [0x04ed2000-0x04ed5fff]	0x00004000	(16 KiB)
Alloc by (rkisp-vir0	) [0x04ed6000-0x04ed9fff]	0x00004000	(16 KiB)
Alloc by (rkisp-vir0	) [0x04eda000-0x04eddfff]	0x00004000	(16 KiB)
Alloc by (ffa00000.rkisp	) [0x05c30000-0x05c33fff]	0x00004000	(16 KiB)

注意：如果是安全启动，需要把sys\_bootargs的参数写到内核dts的bootargs。

## 5.11 如何使用coredump功能

- 内核打开对应defconfig

```
CONFIG_ELF_CORE=y
CONFIG_CORE_DUMP_DEFAULT_ELF_HEADERS=y
```

- 板端设置coredump文件大小

```
ulimit -c unlimited
```

- 板端设置coredump文件位置

```
echo "/data/core-%p-%e" > /proc/sys/kernel/core_pattern
```

注：如果要把coredump文件直接写到NFS或VFAT文件系统，开机后需要执行如下代码，否则在生成的coredump文件大小一直为0

```
# dump core file to /mnt/sdcard (which mount on vfat)
ulimit -c unlimited
echo "| /bin/coredump.sh %p %e" > /proc/sys/kernel/core_pattern
cat > /bin/coredump.sh << EOF
#!/bin/sh
exec cat - > "/mnt/sdcard/core-%$1-%$2"
EOF
chmod a+x /bin/coredump.sh
```

- 查看coredump里的堆栈

把板端生成的coredump文件（例如：/data/core-279-rkipc\_get\_nn\_up）拷贝到SDK根目录，然后运行如下命令：

例如RV1106、RV1103平台工具链是arm-rockchip830-linux-uclibcgnueabi-hf-gdb

```
arm-rockchip830-linux-uclibcgnueabi-hf-gdb ./output/out/app_out/bin/rkipc
./core-279-rkipc_get_nn_up
# ...
(gdb) set solib-search-path output/out/media_out/lib/
Reading symbols from /home/rk/ipc-
sdk/output/out/media_out/lib/librkaiq.so...done.
Reading symbols from /home/rk/ipc-sdk/output/out/media_out/lib/librockiva.so...
(no debugging symbols found)...done.
Reading symbols from /home/rk/ipc-
sdk/output/out/media_out/lib/librockchip_mpp.so.0...(no debugging symbols
found)...done.
Reading symbols from /home/rk/ipc-
sdk/output/out/media_out/lib/libaec_bf_process.so...(no debugging symbols
found)...done.
Reading symbols from /home/rk/ipc-
sdk/output/out/media_out/lib/librkaudio_detect.so...(no debugging symbols
found)...done.
Reading symbols from /home/rk/ipc-sdk/output/out/media_out/lib/librga.so...(no
debugging symbols found)...done.
```



```

Reading symbols from /home/rk/ipc-sdk/output/out/media_out/lib/librknnmrt.so...
(no debugging symbols found)...done.
(gdb) bt
#0  0x00042080 in xx_list_pop ()
#1  0x0004229c in rkipc_xx_object_get ()
#2  0x0001a3c4 in rkipc_xx_update_osd ()
#3  0xa6c05390 in start_thread ()
    from /home/rk/ipc-sdk/tools/linux/toolchain/arm-rockchip830-linux-
uclibcgnueabi/hf/arm-rockchip830-linux-uclibcgnueabi/hf/sysroot/lib/libc.so.0
#4  0xa6bb8764 in clone ()
    from /home/rk/ipc-sdk/tools/linux/toolchain/arm-rockchip830-linux-
uclibcgnueabi/hf/arm-rockchip830-linux-uclibcgnueabi/hf/sysroot/lib/libc.so.0
Backtrace stopped: previous frame identical to this frame (corrupt stack?)

```

## 5.12 内核驱动insmod说明

本章节适用于RV1106、RV1103。

可以参考 `sysdrv/drv_ko/insmod_ko.sh`

```

#!/bin/sh
# if not install udevadm, ignore 'udevadm control'
udevadm control --stop-exec-queue

insmod rk_dvbm.ko

insmod video_rkcif.ko
insmod video_rkisp.ko
insmod phy-rockchip-csi2-dphy-hw.ko
insmod phy-rockchip-csi2-dphy.ko

insmod os04a10.ko
insmod sc4336.ko
insmod sc3336.ko
insmod sc530ai.ko

echo 1 > /sys/module/video_rkcif/parameters/clr_unready_dev
echo 1 > /sys/module/video_rkisp/parameters/clr_unready_dev

insmod rga3.ko

insmod mpp_vcodec.ko
insmod rockit.ko

insmod rknpu.ko
insmod rve.ko
insmod snd-soc-rv1106.ko

# $sensor_height is the height of the camera sensor (e.g. os04a0/sc4336/sc3336
and so on)
insmod rockit.ko mcu_fw_path="./hpmcu_wrap.bin" mcu_fw_addr=0xff6ff000
isp_max_h=$sensor_height

udevadm control --start-exec-queue

```

## 5.13 RV1106和RV1103平台相关库文件和驱动文件的信息

- 库文件

名称	大小	用途	是否必须
ld-uClibc-1.0.31.so	32K	toolchain标准库	是
libatomic.so	16K	toolchain标准库	是
libgcc_s.so	4.0K	toolchain标准库	是
libgcc_s.so.1	124K	toolchain标准库	是
libstdc++.so	992K	toolchain标准库	是
libuClibc-1.0.31.so	420K	toolchain标准库	是
libitm.so	52K	toolchain标准库	是
librga.so	96K	2D图形加速库	是
librkaiq.so	1.1M	瑞芯微自动图像算法库	是
librockchip_mpp.so	272K	通用媒体处理软件平台	是
librockit.so	812K	通用多媒体接口	是
libaec_bf_process.so	380K	音频算法库	否
librkaudio_detect.so	148K	音频检测库	否
librockiva.so	760K	NPU分析算法库	否
librknnmrt.so	84K	NPU分析算法依赖库	否
librve.so	96K	智能视频分析硬件加速引擎库	否
librkfsmk.so	68K	存储优化相关	否
librkmuxer.so	552K	多媒体文件封装库	否
libdrm_rockchip.so	8.0K	显示驱动框架（Rockchip）	否
libdrm.so	48K	显示驱动框架	否
libcgicc.so	96K	公共网关接口c++库	否
libfcgi.so	32K	快速公共网关接口库	否
libfcgi++.so	16K	快速公共网关接口c++库	否
libiconv.so	236K	转换字符编码库	否
libkmod.so	48K	udevadm依赖库	否
libblkid.so	180K	udevadm依赖库	否
libpcre.so	92K	perl兼容的正则表达式库	否
libwpa_client.so	28K	WiFi工具依赖库	否
libz.so	76K	压缩库	否

- 驱动文件

部分内核驱动文件	大小	用途	是否必须
mpp_vcodec.ko	462K	视频编码器驱动	是
phy-rockchip-csi2-dphy-hw.ko	14K	mipi dphy rx物理驱动	是
phy-rockchip-csi2-dphy.ko	14K	mipi dphy rx逻辑驱动	是
video_rkcif.ko	140K	CIF驱动	是
video_rkisp.ko	172K	图像信号处理驱动	是
rockit.ko	109K	多媒体框架驱动	是
rga3.ko	104K	2D图像处理模块驱动	是
os04a10.ko	24K	os04a10 sensor 驱动	否
sc3336.ko	16K	sc3336 sensor 驱动	否
sc4336.ko	16K	sc4336 sensor 驱动	否
sc530ai.ko	20K	sc530ai sensor 驱动	否
rknpu.ko	32K	NPU驱动	否
rve.ko	36K	智能视频分析硬件加速引擎	否

## 5.14 如何使用NFS文件系统

- 内核打开NFS配置

```

CONFIG_EXPORTFS_BLOCK_OPS=y
CONFIG_FILE_LOCKING=y
CONFIG_KEYS=y
CONFIG_NETWORK_FILESYSTEMS=y
CONFIG_ASSOCIATIVE_ARRAY=y
CONFIG_DNS_RESOLVER=y
# CONFIG_ECRYPT_FS is not set
# CONFIG_ENCRYPTED_KEYS is not set
CONFIG_FS_POSIX_ACL=y
CONFIG_GRACE_PERIOD=y
CONFIG_LOCKD=y
CONFIG_LOCKD_V4=y
CONFIG_MANDATORY_FILE_LOCKING=y
CONFIG_NFS_ACL_SUPPORT=y
CONFIG_NFS_COMMON=y
CONFIG_NFS_DISABLE_UDP_SUPPORT=y
CONFIG_NFS_FS=y
CONFIG_NFS_USE_KERNEL_DNS=y
# CONFIG_NFS_USE_LEGACY_DNS is not set
CONFIG_NFS_V2=y
CONFIG_NFS_V3=y
CONFIG_NFS_V3_ACL=y
CONFIG_NFS_V4=y
CONFIG_OID_REGISTRY=y
# CONFIG_PERSISTENT_KEYRINGS is not set

```

```
CONFIG_SUNRPC=y
CONFIG_SUNRPC_GSS=y
```

- 配置PC端NFS服务器

```
# Ubuntu 16.04 install NFS server
sudo apt-get install nfs-kernel-server
# Create /opt/rootfs
mkdir /opt/rootfs
# Enable nobody mount /opt/rootfs
chmod O+w -R /opt/rootfs
# Add the directory to exports
sudo echo "/opt/rootfs *(rw,sync,root_squash)" >> /etc/exports
# Update NFS configure
sudo exportfs -r
# Test NFS server
sudo mount -t nfs localhost:/opt/rootfs /mnt
# Ubuntu 16.04 disable firewall
sudo ufw disable
```

- 单板端NFS挂载命令

```
# Get the IP address 192.168.1.123 of the PC
mount -t nfs -o nolock 192.168.1.123:/opt/rootfs /opt
```

## 5.15 如何增加新用户并设置登陆密码

以创建新用户（testNewUser）为例。

- 查看当前用户列表，获取新用户的UID和GID

文件格式：

**Username:Password:User ID(UID):Group ID(GID):User ID Info (GECOS):Home directory:Login shell**

修改单板上/etc/passwd，增加如下

```
testNewUser:x:1000:1000:testNewUser:/home:/bin/sh
```

- 在PC端用mkpasswd命令生成密码

```
sudo apt install whois
mkpasswd -m "md5" "test123" # --> $1$kprQOoLU$kOU2H.ecXkAw1ZJ0oplu/.
```

- 修改单板上/etc/shadow，增加如下

```
testNewUser:$1$kprQOoLU$kOU2H.ecXkAw1ZJ0oplu/..:0:0:99999:7:::
```

- 修改单板上/etc/group，增加如下

```
testNewUser:x:1000:
```

- 修改单板上/etc/inittab，修改如下

```
#::respawn:-/bin/sh
::sysinit:/etc/init.d/rcS

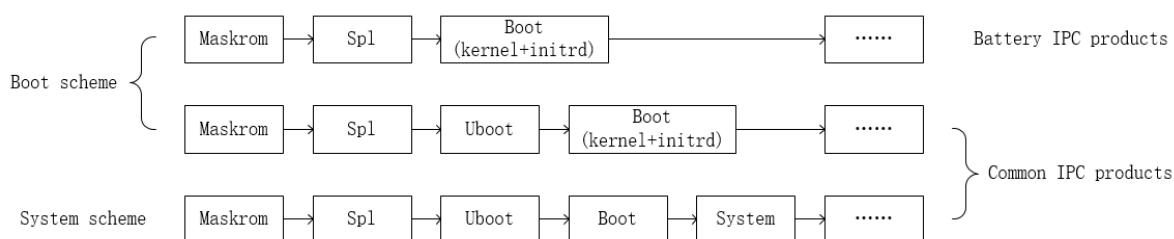
# Put a getty on the serial port
ttyFIQ0::respawn:/sbin/getty -L ttyFIQ0 0 vt100 # GENERIC_SERIAL
```

## 5.16 A/B系统使用方法

### 5.16.1 启动方案介绍

A/B系统支持两种启动方案：boot方案——boot分区兼根文件系统（kernel+initrd）、system方案——独立根文件系统（boot+system）。

目前，这两种启动方案，通用IPC类产品均支持；电池IPC类产品仅支持boot方案。



### 5.16.2 配置

- Uboot

通用IPC类产品和电池IPC类产品需要开启的uboot配置不同，对应defconfig开启配置如下所示：

```
# 通用IPC类产品
CONFIG_ANDROID_AB=y
CONFIG_SPL_MTD_WRITE=y
CONFIG_SPL_AB=y
CONFIG_EFI_PARTITION=y
CONFIG_SPL_EFI_PARTITION=y
CONFIG_AVB_LIBAVB=y
CONFIG_AVB_LIBAVB_AB=y
CONFIG_AVB_LIBAVB_USER=y
CONFIG_RK_AVB_LIBAVB_USER=y
```

```
# 电池IPC类产品
CONFIG_SPL_MTD_WRITE=y
CONFIG_SPL_AB=y
CONFIG_EFI_PARTITION=y
CONFIG_SPL_EFI_PARTITION=y
```

- Kernel

仅在boot方案使用initramfs文件系统时，才需要开启以下配置，其余情况不需要kernel开启额外配置。

```
# initramfs文件系统kernel配置
CONFIG_BLK_DEV_INITRD=y
```

- Root File System

当使用SPI Nand存储介质时，需要保证文件系统里有以下shell工具：

工具名称	工具功能
flash_erase	擦除指定mtd设备的块
nandwrite	写入指定的mtd设备
nanddump	转储nand mtd分区的内容
md5sum	计算和检查md5消息摘要
dd	转换和复制一个文件

- BoardConfig

对应BoardConfig板级配置中使能OTA，以编译OTA升级工具和打包相应镜像，用于A/B系统切换和升级。

```
# Enable OTA tool
export RK_ENABLE_OTA=y
# OTA package
export RK_OTA_RESOURCE="uboot.img boot.img system.img"
```

修改分区表RK\_PARTITION\_CMD\_IN\_ENV、文件系统类型RK\_PARTITION\_FS\_TYPE\_CFG这两个参数。

以下提供了boot、system两种方案的例子，仅供参考，实际参数可根据需求自行修改。

boot方案：

```
# 分区表（添加misc分区，去掉rootfs分区，boot分区改为boot_a、boot_b分区）
export
RK_PARTITION_CMD_IN_ENV="64K(env),256K@64K(idblock),256K(uboot),64K(misc),384K(meta),6M(boot_a),6M(boot_b),1M(userdata)"

# 文件系统类型（电池IPC类产品默认使用erofs，通用IPC类产品默认使用initramfs）
export RK_PARTITION_FS_TYPE_CFG=boot_a@IGNORE@erofs,userdata@/userdata@jffs2
```

system方案：

```
# 分区表（添加misc分区，去掉rootfs分区，uboot分区改为uboot_a、uboot_b分区，boot分区改为boot_a、boot_b分区，添加system_a、system_b分区）
export
RK_PARTITION_CMD_IN_ENV="256K(env),256K@256K(idblock),256K(uboot_a),256K(uboot_b),256K(misc),4M(boot_a),4M(boot_b),16M(system_a),16M(system_b),32M(oem),32M(userdata),-(media)"

# 文件系统类型
export
RK_PARTITION_FS_TYPE_CFG=system_a@IGNORE@ubifs,oem@/oem@ubifs,userdata@/userdat
a@ubifs
```

### 5.16.3 OTA升级工具

AB系统默认使用RETRY\_BOOT模式启动。若需要使用SUCCESSFUL\_BOOT模式，则修改配置如下所示：

**注：两种模式的区别，可见U-Boot文档第6.3.2章节：  
Rockchip\_Developer\_Guide\_UBoot\_Nextdev\_CN.pdf**

```
diff --git a/tools/board/rk_ota/Makefile b/tools/board/rk_ota/Makefile
index 8bf67f5..21cf5e0 100644
--- a/tools/board/rk_ota/Makefile
+++ b/tools/board/rk_ota/Makefile
@@ -10,7 +10,7 @@ CURRENT_DIR := $(shell pwd)
 PKG_BIN := out
 TARGET_BIN := rk_ota
 PKG_SRCS := src/mtutils.c src/md5sum.cpp src/bootloader.cpp src/main.cpp
-CONFIG_IS_SUCCESSFUL_BOOT := n
+CONFIG_IS_SUCCESSFUL_BOOT := y

PKG_DEP_OPENSSL := $(CURRENT_DIR)/../toolkits/openssl
PKG_DEP_ZLIB := $(CURRENT_DIR)/../toolkits/zlib
```

BoardConfig中添加配置 `RK_ENABLE_OTA` 后，SDK会将OTA升级工具编译到固件文件系统的 `/usr/bin` 下，指令名为 `rk_ota`，同时打包相应的镜像至 `<SDK>/output/image/update_ota.tar` 中。

```
# rk_ota --help
[I/]RECOVERY *** rk_ota: Version V1.0.0 ***.
[I/]RECOVERY --misc=now                               Linux A/B mode: Setting the
current partition to bootable.
[I/]RECOVERY --misc=other                             Linux A/B mode: Setting
another partition to bootable.
[I/]RECOVERY --misc=update                             Linux A/B mode: Setting the
partition to be upgraded.
[I/]RECOVERY --misc=display                             Display misc info.
[I/]RECOVERY --tar_path=<path>                         Set upgrade firmware path.
[I/]RECOVERY --save_dir=<path>                         Set the path for saving the
image.
[I/]RECOVERY --partition=<uboot/boot/system/all>       Set the partition to be
upgraded. ('all' means 'uboot', 'boot' and 'system' are included.)
[I/]RECOVERY --extra_part=<name>                       Set the extra partition to be
upgraded.
[I/]RECOVERY --reboot                                 Restart the machine at the
end of the program.
```

### 5.16.4 A/B系统切换

切换A/B系统的指令为 `rk_ota --misc=other`，运行情况如下：

```
# rk_ota --misc=other --reboot
[I/]RECOVERY *** rk_ota: Version V1.0.0 ***.
[I/]RECOVERY Now is MTD.
A/B-slot: B, successful: 0, tries-remain: 6
[I/]RECOVERY Now is MTD.
```



`--reboot`：重启设备。

进入另一个系统，若该系统成功启动，需要运行指令 `rk_ota --misc=now`，将该系统设置为“最后启动的系统”。

```
# rk_ota --misc=now
[I/]RECOVERY *** rk_ota: Version V1.0.0 ***.
[I/]RECOVERY Now is MTD.
A/B-slot: A, successful: 0, tries-remain: 6
info.mafic is 0
info.mafic is 41
info.mafic is 42
info.mafic is 30
[I/]RECOVERY Now is MTD.
```

### 5.16.5 A/B系统升级

升级A/B系统的相关选项有：

`--misc=update`：选择“升级”模式；

`--tar_path=<path>`：设置OTA升级包的路径（存放镜像的tar包）；

`--save_dir=<path>`：（可选）设置解包文件夹，未设置则默认为 `/mnt/sdcard/rk_update/`；

`--partition=<uboot/boot/system/all>`：（可选）设置需要升级的分区，“all”表示“uboot”、“boot”和“system”都升级，未设置则默认为“all”。（当源文件不存在时，会跳过相应分区的升级。）

`--extra_part=<name>`：（可选）设置一个需要升级的自定义分区，未设置则默认忽略该选项。

**注：目前只能升级uboot、boot、system分区和一个自定义分区；**

示例：

```
# rk_ota --misc=update --tar_path=/mnt/sdcard/update_ota.tar --
save_dir=/mnt/sdcard/ --partition=all --reboot
# 写boot
[I/]RECOVERY *** rk_ota: Version V1.0.0 ***.
[I/]RECOVERY tar path = /mnt/sdcard/update_ota.tar
[I/]RECOVERY save path = /mnt/sdcard/
[I/]RECOVERY Now is MTD.
A/B-slot: A, successful: 0, tries-remain: 5
[I/]RECOVERY Now is MTD.
[I/]RECOVERY mtd_write src=/mnt/sdcard//boot.img dest=/dev/block/by-
name/boot_b.
Erasing 128 Kibyte @ 3e0000 -- 100 % complete
Writing data to block 0 at offset 0x0
Writing data to block 1 at offset 0x20000
...
2846720+0 records in
2846720+0 records out

[I/]RECOVERY Now is MTD.
[I/]RECOVERY [checkdata_mtd:30] offset [0] checksize [2846720]
ECC failed: 0
ECC corrected: 0
Number of bad blocks: 0
```

```

Number of bbt blocks: 0
Block size 131072, page size 2048, OOB size 128
Dumping data starting at 0x00000000 and ending at 0x002b7000...

read new md5: [6c60afda3ab31a49acd6d5d65e86a2e6]
new md5:6c60afda3ab31a49acd6d5d65e86a2e6
[I/]RECOVERY MD5Check is ok of /dev/block/by-name/boot_b
new md5:6c60afda3ab31a49acd6d5d65e86a2e6
[I/]RECOVERY MD5Check is ok for /mnt/sdcard//boot.img
[I/]RECOVERY check /dev/block/by-name/boot_b ok.
[I/]RECOVERY Write /mnt/sdcard//boot.img into /dev/block/by-name/boot_b
successfully.

# 写system
[I/]RECOVERY Now is MTD.
[I/]RECOVERY mtd_write src=/mnt/sdcard//system.img dest=/dev/block/by-
name/system_b.
Erasing 128 Kibyte @ fe0000 -- 100 % complete
Writing data to block 0 at offset 0x0
Writing data to block 1 at offset 0x20000
...
8126464+0 records in
8126464+0 records out

Writing data to block 62 at offset 0x7c0000
[I/]RECOVERY Now is MTD.
[I/]RECOVERY [checkdata_mtd:30] offset [0] checksize [8126464]
ECC failed: 0
ECC corrected: 0
Number of bad blocks: 0
Number of bbt blocks: 0
Block size 131072, page size 2048, OOB size 128
Dumping data starting at 0x00000000 and ending at 0x007c0000...
ECC: 1 corrected bitflip(s) at offset 0x004be000

read new md5: [9f9fad6f08cbdd210488ff544e14af25]
new md5:9f9fad6f08cbdd210488ff544e14af25
[I/]RECOVERY MD5Check is ok of /dev/block/by-name/system_b
new md5:9f9fad6f08cbdd210488ff544e14af25
[I/]RECOVERY MD5Check is ok for /mnt/sdcard//system.img
[I/]RECOVERY check /dev/block/by-name/system_b ok.
[I/]RECOVERY Write /mnt/sdcard//system.img into /dev/block/by-name/system_b
successfully.

[I/]RECOVERY Now is MTD.
[I/]RECOVERY Now is MTD.
mtd: successfully wrote block at 395a80000000
mtd: successfully wrote block at 395a800020000
reboot

```

## 5.17 获取摄像头支持列表

可以在Redmine上获取<https://redmine.rock-chips.com/documents/53>

## 5.18 获取Flash支持列表

可以在Redmine上获取<https://redmine.rock-chips.com/documents/46>

## 5.19 压力测试使用方法

压力测试需要开启如下配置：

```
# enable rockchip test
export RK_ENABLE_ROCKCHIP_TEST=y
```

目前已支持的压力测试列表：

```
# ./rockchip_test/rockchip_test.sh
*****
***                                     ***
***          *****                  ***
***      *ROCKCHIPS TEST TOOLS*      ***
***          *                      ***
***          *****                  ***
***                                     ***
*****
*****
ddr test :           1 (memtester & stressapptest)
cpufreq test:        2 (cpufreq stresstest)
flash stress test:   3
auto reboot test:    4
*****
please input your test moudle:
```

### 5.19.1 memtester test

- 打开压力测试列表

```
sh rockchip_test/rockchip_test.sh
```

- 开始测试（压力测试列表内选择测试项对应序号1）
- 再选择 `memtester test` 对应序号（默认使用空闲内存的一半容量进行测试）

### 5.19.2 stressapptest

- 打开压力测试列表

```
sh rockchip_test/rockchip_test.sh
```

- 开始测试（压力测试列表内选择测试项对应序号1）
- 再选择 `stressapptest` 对应序号（默认使用空闲内存的一半容量测试48小时）

### 5.19.3 cpufreq test

- 打开压力测试列表

```
sh rockchip_test/rockchip_test.sh
```

- 开始测试（压力测试列表内选择测试项对应序号2）
- 再选择 `cpu freq stress test` 或 `cpu freq test:(with out stress test)` 对应序号（前者默认使用空闲内存的一半容量测试24小时；后者默认一秒变频一次）

### 5.19.4 flash stress test

- 打开压力测试列表

```
sh rockchip_test/rockchip_test.sh
```

- 开始测试（压力测试列表内选择测试项对应序号3）

### 5.19.5 reboot test

- 打开压力测试列表

```
sh rockchip_test/rockchip_test.sh
```

- 开始测试（压力测试列表内选择测试项对应序号4）
- 退出测试（默认重启10000次后停止测试；若需要提前退出，可输入以下指令）

```
echo off > /data/cfg/rockchip_test/reboot_cnt
```

## 5.20 安全启动相关代码以及文档说明

相关加解密代码路径：`media/security`

相关安全文档路径：`docs/zh/security`

U-Boot签名文档路径（FIT章节）：

`docs/zh/bsp/Rockchip_Developer_Guide_UBoot_Nextdev_CN.pdf`

### 5.20.1 Key

U-Boot工程下执行如下三条命令可以生成签名用的RSA密钥对。通常情况下只需要生成一次，此后都用这对密钥对（`dev.key`、`dev.pubkey`）和自签名证书（`dev.crt`）来签名和验证固件，请妥善保管。

```
# 1. 到rkbin/tools目录下进行相应操作
cd ./sysdrv/source/uboot/rkbin/tools

# 2. 使用RK的"rk_sign_tool"工具生成RSA2048的私钥privateKey.pem和publicKey.pem
```

```
./rk_sign_tool kk --bits 2048 --out .

# 3. 到U-Boot目录下进行后续操作
cd ../../u-boot

# 4. 放key的目录: keys
mkdir -p keys

# 5. 将密钥分别更名存放为: keys/dev.key和keys/dev.pubkey
cp ../rkbin/tools/private_key.pem keys/dev.key
cp ../rkbin/tools/public_key.pem keys/dev.pubkey

# 6. 使用-x509和私钥生成一个自签名证书: keys/dev.crt (效果本质等同于公钥)
openssl req -batch -new -x509 -key keys/dev.key -out keys/dev.crt
```

ls keys/ 查看结果:

```
dev.crt dev.key dev.pubkey
```

**注:** 上述的"keys"、"dev.key"、"dev.crt"、"dev.pubkey"名字都不可变。因为这些名字已经在its文件中静态定义, 如果改变则会打包失败。

## 5.20.2 U-Boot配置

U-Boot的defconfig需要打开如下配置:

```
# 必选项
CONFIG_FIT_SIGNATURE=y
CONFIG_SPL_FIT_SIGNATURE=y
CONFIG_CMD_HASH=y
CONFIG_DM_CRYPT=y
CONFIG_ROCKCHIP_CIPHER=y
CONFIG_SPL_ROCKCHIP_CIPHER=y
CONFIG_ROCKCHIP_CRYPT_V2=y
CONFIG_SPL_ROCKCHIP_SECURE_OTP=y
CONFIG_RSA_N_SIZE=0x200
CONFIG_RSA_E_SIZE=0x10
CONFIG_RSA_C_SIZE=0x20

# 可选项 (U-Boot读写otp)
CONFIG_ROCKCHIP_SECURE_OTP=y      # 使能U-Boot阶段读写otp
CONFIG_MISC=y                     # 相关读写函数的编译配置

# 可选项 (防回滚)
CONFIG_FIT_ROLLBACK_PROTECT=y     # boot.img防回滚
CONFIG_SPL_FIT_ROLLBACK_PROTECT=y # uboot.img防回滚
```

## 5.20.3 固件签名

固件签名需要用到U-Boot目录下的make.sh脚本, make.sh追加参数的含义如下:

追加参数	含义
--spl-new	传递此参数，表示使用当前编译的spl文件打包loader，否则使用rkbin工程里的spl文件
--boot_img	签名boot.img
--recovery_img	签名recovery.img
CROSS_COMPILE=xxx	确定工具链（不同芯片工具链见 <a href="#">交叉工具链下载以及安装</a> ）
--rollback-index-uboot	uboot防回滚（开启配置则不需要追加该参数）
--rollback-index-boot	boot防回滚（开启配置则不需要追加该参数）
--rollback-index-recovery	recovery防回滚
--burn-key-hash	要求SPL阶段把公钥hash烧写到OTP中
[ini_path]	（电池IPC类必选） rkbin/RKBOOT 中的ini文件路径（具体可见uboot固件编译命令）

签名固件前需要先编译出相应固件，编译方法可见[SDK 使用说明](#)的编译章节。下面是签名步骤：

```
# 1. 拷贝相应固件到U-Boot目录下（以boot.img为例，recovery.img类似）
cp ./output/image/boot.img ./sysdrv/source/uboot/u-boot

# 2. 到U-Boot目录下进行相应操作
cd ./sysdrv/source/uboot/u-boot

# 3. 签名固件，指令格式如下：
# ./make.sh --spl-new [ini_path] [--boot_img <boot镜像名称>] [--recovery_img
<recovery镜像名称>] CROSS_COMPILE=<交叉工具链> --burn-key-hash
# 例如：
./make.sh --spl-new --boot_img boot.img CROSS_COMPILE=arm-rockchip830-linux-
uclibcgnueabi-hf- --burn-key-hash
```

- 注1：使用make.sh脚本生成固件，应根据实际需要，追加相应参数。
- 注2：添加参数 `--burn-key-hash` 会将公钥hash写入OTP并使能安全启动。如果需要自行写入hash以及使能安全启动，则不要追加此参数。
- 注3：签名后的固件在U-Boot目录下，固件名分别为xxx\_download\_xxx.bin、xxx\_idblock\_xxx.img、uboot.img、boot.img（如果有）、recovery.img（如果有）。

## 5.21 如何使用rndis功能

确认内核dts配置

```
&u2phy {
    status = "okay";
};

&u2phy_otg {
    status = "okay";
```

```
};

&usbdrd {
    status = "okay";
};

&usbdrd_dwc3 {
    extcon = <&u2phy>;
    dr_mode = "peripheral";
    status = "okay";
};
```

确认内核defconfig配置 `rv1106-rndis.config`。

确认usb的vbus 有供电3.3V。

修改对应的BoardConfig.mk, 增加 `export RK_ENABLE_RNDIS=y`, 然后按如下命令编译:

```
./build.sh sysdrv
./build.sh firmware
```

烧录固件后, 开机运行命令: `rndis.sh`

详细log如下:

```
# rndis.sh
serialnumber is f5bc7ed083b85dcf
config usb0 IP...
# ifconfig
usb0      Link encap:Ethernet  HWaddr C2:44:18:6D:9A:05
          inet addr:192.168.1.100  Bcast:192.168.1.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:133 errors:0 dropped:69 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:11360 (11.0 KiB)  TX bytes:0 (0.0 B)
```

注: 如果需要修改默认IP地址, 可以修改SDK文件"sysdrv/tools/board/rndis/rndis.sh", 然后重新编译烧录固件。

## 5.22 如何优化SPI NOR启动速度

系统启动流程: 上电 --> Maskrom --> idblock --> uboot --> kernel

Maskrom会读取idblock镜像里的bootconfig参数配置, 然后使用4线模式启动SPI NOR (默认是1线模式)。

Maskrom配置SPI NOR使用4线方式启动的方法有2种:

1. 内核defconfig打开 `CONFIG_MTD_SPI_NOR_MISC=y`, 系统开机后运行 `idb_bootconfig /dev/mtdblock1`  
(参数 `/dev/mtdblock1` 是idblock分区对应的mtd设备节点, `idb_bootconfig`代码在: `sysdrv/tools/board/idb_bootconfig/idb_bootconfig.c`)
2. 使用Rockchip的USB烧录工具进行烧录idblock.img固件。

注: 目前方法1只适用于RV1106/RV1103平台。

## 5.23 如何增加非root用户登陆

这里介绍使用busybox时，增加普通用户登陆终端的方法。

### 1. 修改/etc/passwd

```
test123:x:1010:1011:test123:/opt:/bin/sh
```

x表示用户设有密码

1010表示用户id

1011表示用户组id

/opt表示test123用户的HOME目录

注：/etc/passwd详细说明参考<https://www.man7.org/linux/man-pages/man5/passwd.5.html>

### 2. 修改/etc/shadow

```
test123:$1$x.YlInZQ$N.kbNTIEOkBjmlfftSVFs0:10933:0:99999:7:::
```

`$1$x.YlInZQ$N.kbNTIEOkBjmlfftSVFs0:10933` 是用命令 `mkpasswd -m "md5" "rockchip123"` 生成。

注：/etc/shadow详细说明参考<https://www.man7.org/linux/man-pages/man5/shadow.5.html>

### 3. 修改/etc/group

```
test123:x:1011:test123
```

注：/etc/group详细说明参考<https://www.man7.org/linux/man-pages/man5/group.5.html>

### 4. 修改/etc/inittab

```
#::respawn:~/bin/sh
ttyFIQ0::respawn:/sbin/getty -L ttyFIQ0 0 vt100 # GENERIC_SERIAL
```

## 5.24 如何添加新的Camera sensor配置

以增加 sc530ai sensor 为例：

- 在 `sysdrv/source/kernel/drivers/media/i2c` 目录添加 sensor 驱动（sdk 已包含大部分 sensor 驱动，如果没有，可以参考 sc530ai 驱动添加）

```
sysdrv/source/kernel/drivers/media/i2c$ ls sc*

sc031gs.c sc200ai.c sc2232.c sc2310.c sc401ai.c sc430cs.c sc500ai.c sc132gs.c
sc210iot.c sc2239.c sc3336.c sc4238.c sc4336.c sc530ai.c
```

- 在 Kconfig 和 Makefile 添加对应配置

```
sysdrv/source/kernel/drivers/media/i2c$ vi Kconfig

config VIDEO_SC530AI
    tristate "SmartSens SC530AI sensor support"
```



```

depends on I2C && VIDEO_V4L2
select MEDIA_CONTROLLER
select VIDEO_V4L2_SUBDEV_API
select V4L2_FWNODE
help
    This is a Video4Linux2 sensor driver for the SmartSens
    SC530AI camera.

sysdrv/source/kernel/drivers/media/i2c$ vi Makefile

obj-$(CONFIG_VIDEO_SC530AI) += sc530ai.o

```

- 在对应的defconfig中添加如下配置，编译 ko 文件

```

# 例如: rv1106-uvic-spi-nor.config
sysdrv/source/kernel$ vi arch/arm/configs/rv1106-uvic-spi-nor.config

CONFIG_VIDEO_SC530AI=m

```

- dts中添加如下配置

```

sysdrv/source/kernel$ vi arch/arm/boot/dts/rv1106-evb-cam.dtsi

&csi2_dphy_hw {
    status = "okay";
};

&csi2_dphy0 {
    status = "okay";

    ports {

        #address-cells = <1>;
        #size-cells = <0>;

        port@0 {
            reg = <0>;
            #address-cells = <1>;
            #size-cells = <0>;

            csi_dphy_input0: endpoint@0 {
                reg = <0>;
                remote-endpoint = <&sc530ai_out>;
                data-lanes = <1 2>;          //注意: 如果是4lane则为data-lanes = <1 2 3
4>;
            };
        };

        port@1 {
            reg = <1>;
            #address-cells = <1>;
            #size-cells = <0>;

            csi_dphy_output: endpoint@0 {
                reg = <0>;
                remote-endpoint = <&mipi_csi2_input>;
            };
        };
    };
};

```

```

    };

};

};

&i2c4 {
    status = "okay";
    clock-frequency = <400000>;
    pinctrl-names = "default";
    pinctrl-0 = <&i2c4m2_xfer>;

    sc530ai: sc530ai@30 {                                     //30和reg的0x30 代表sensor的i2c
地址
        compatible = "smartsens,sc530ai";                   //需要和驱动中sc530ai_of_match
字段匹配
        status = "okay";
        reg = <0x30>;                                         //i2c地址
        clocks = <&cru MCLK_REF_MIPI0>;
        clock-names = "xvclk";
        reset-gpios = <&gpio3 RK_PC5 GPIO_ACTIVE_HIGH>;
        pwn-gpios = <&gpio3 RK_PD2 GPIO_ACTIVE_HIGH>;
        pinctrl-names = "default";
        pinctrl-0 = <&mipi_refclk_out0>;
        rockchip, camera-module-index = <0>;
        rockchip, camera-module-facing = "back";
        rockchip, camera-module-name = "CMK-OT2115-PC1";    //模组名字
        rockchip, camera-module-lens-name = "30IRC-F16";    //模组规格      名字和规格
用于匹配json文件名字
        port {
            sc530ai_out: endpoint {
                remote-endpoint = <&csi_dphy_input0>;
                data-lanes = <1 2>;      //注意：如果是4lane则为data-lanes = <1 2 3
4>;
            };
        };
    };
};

&mipi0_csi2 {
    status = "okay";

    ports {
        #address-cells = <1>;
        #size-cells = <0>;

        port@0 {
            reg = <0>;
            #address-cells = <1>;
            #size-cells = <0>;

            mipi_csi2_input: endpoint@1 {
                reg = <1>;
                remote-endpoint = <&csi_dphy_output>;
            };
        };

        port@1 {

```

```

        reg = <1>;
        #address-cells = <1>;
        #size-cells = <0>;

        mipi_csi2_output: endpoint@0 {
            reg = <0>;
            remote-endpoint = <&cif_mipi_in>;
        };
    };
};

&rkCIF {
    status = "okay";
};

&rkCIF_mipi_lvds {
    status = "okay";

    pinctrl-names = "default";
    pinctrl-0 = <&mipi_pins>;
    port {
        /* MIPI CSI-2 endpoint */
        cif_mipi_in: endpoint {
            remote-endpoint = <&mipi_csi2_output>;
        };
    };
};

&rkCIF_mipi_lvds_sditf {
    status = "okay";

    port {
        /* MIPI CSI-2 endpoint */
        mipi_lvds_sditf: endpoint {
            remote-endpoint = <&isp_in>;
        };
    };
};

&rkISP {
    status = "okay";
};

&rkISP_vir0 {
    status = "okay";

    port@0 {
        isp_in: endpoint {
            remote-endpoint = <&mipi_lvds_sditf>;
        };
    };
};
};

```

- 开机自动加载 ko

一般在 `sysdrv/drv_ko/insmod_ko.sh` 里添加 `insmod` 命令。

如果 `BoardConfig.mk` 里有配置 `RK_POST_BUILD_SCRIPT`，则在对应的脚本里添加 `insmod` 命令。

```
# 例如: BoardConfig_SmartDoor
project/cfg/BoardConfig_SmartDoor$ vi rv1106-tb-post.sh

insmod /oem/usr/ko/sc530ai.ko
```

- 在对应的 `BoardConfig.mk` 添加配置拷贝对应的 Sensor IQ 效果文件

```
export RK_CAMERA_SENSOR_IQFILES="sc530ai_CMK-OT2115-PC1_30IRC-F16.json"
```

- Sensor IQ 效果文件放在目录 `media/isp/release_camera_engine_rkaiq*`

```
media/isp/release_camera_engine_rkaiq_rv1106_arm-rockchip830-linux-
uclibcgnueabi/hf/isp_iqfiles/$ ls

sc530ai_CMK-OT2115-PC1_30IRC-F16.json
```

注:

a. Sensor IQ 效果文件可以用 `j2s4b` 工具把 json 格式转成 bin 格式

(`media/isp/release_camera_engine_rkaiq */host/j2s4b`)

b. Camera sensor 驱动开发和 ISP 效果调试详情可参考 SDK 文档

```
docs/zh/isp$ ls -l
Rockchip_Color_Optimization_Guide_ISP32_CN_v3.1.0.pdf
Rockchip_Development_Guide_ISP32_CN_v0.1.0.pdf
Rockchip_Driver_Guide_VI_CN_v1.1.1.pdf
Rockchip_Tuning_Guide_ISP32_CN_v0.1.0.pdf
```

## 5.25 如何添加 reboot 命令进 U-Boot 终端

以 RV1106 平台为例。

目录: `sysdrv/source/kernel`

```
diff --git a/arch/arm/boot/dts/rv1106.dtsi b/arch/arm/boot/dts/rv1106.dtsi
index 4564909db8b2..99228b4b80cf 100644
--- a/arch/arm/boot/dts/rv1106.dtsi
+++ b/arch/arm/boot/dts/rv1106.dtsi
@@ -359,6 +359,7 @@ reboot_mode: reboot-mode {
                mode-ums = <BOOT_UMS>;
                mode-panic = <BOOT_PANIC>;
                mode-watchdog = <BOOT_WATCHDOG>;
+               mode-uboot = <BOOT_TO_UBOOT>;
            };

            rgb: rgb {
diff --git a/include/dt-bindings/soc/rockchip,boot-mode.h b/include/dt-
bindings/soc/rockchip,boot-mode.h
index 1436e1d32619..cc5a421aef26 100644
--- a/include/dt-bindings/soc/rockchip,boot-mode.h
```

```

+++ b/include/dt-bindings/soc/rockchip,boot-mode.h
@@ -21,4 +21,6 @@
/* enter usb mass storage mode */
#define BOOT_UMS                (REBOOT_FLAG + 12)

+#define BOOT_TO_UBOOT          (REBOOT_FLAG + 14)
+
#endif

```

目录：sysdrv/source/uboot/u-boot

```

diff --git a/arch/arm/include/asm/arch-rockchip/boot_mode.h
b/arch/arm/include/asm/arch-rockchip/boot_mode.h
index bcl395ee2c..d68099a94e 100644
--- a/arch/arm/include/asm/arch-rockchip/boot_mode.h
+++ b/arch/arm/include/asm/arch-rockchip/boot_mode.h
@@ -26,6 +26,8 @@
/* enter bootrom download mode */
#define BOOT_BROM_DOWNLOAD      0xEF08A53C

+#define BOOT_TO_UBOOT          (REBOOT_FLAG + 14)
+
#ifndef __ASSEMBLY__
int setup_boot_mode(void);
#endif
diff --git a/arch/arm/mach-rockchip/boot_mode.c b/arch/arm/mach-
rockchip/boot_mode.c
index 61f0e85c1c..0d555314e2 100644
--- a/arch/arm/mach-rockchip/boot_mode.c
+++ b/arch/arm/mach-rockchip/boot_mode.c
@@ -189,6 +189,11 @@ int rockchip_get_boot_mode(void)
        boot_mode[PL] = BOOT_MODE_UMS;
        clear_boot_reg = 1;
        break;
+
+       case BOOT_TO_UBOOT:
+               printf("boot mode: uboot\n");
+               boot_mode[PL] = BOOT_MODE_UBOOT_TERMINAL;
+               clear_boot_reg = 1;
+               break;
+
        case BOOT_CHARGING:
                printf("boot mode: charging\n");
                boot_mode[PL] = BOOT_MODE_CHARGING;
@@ -227,6 +232,8 @@ int setup_boot_mode(void)
{
    char env_preboot[256] = {0};

+   env_set("cli", NULL); /* removed by default */
+
    switch (rockchip_get_boot_mode()) {
    case BOOT_MODE_BOOTLOADER:
        printf("enter fastboot!\n");
@@ -259,6 +266,10 @@ int setup_boot_mode(void)
        printf("enter charging!\n");
        env_set("preboot", "setenv preboot; charge");
        break;
+
+       case BOOT_MODE_UBOOT_TERMINAL:
+               printf("enter uboot!\n");

```

```

+         env_set("cli", "yes");
+         break;
    }

    return 0;
diff --git a/common/autoboot.c b/common/autoboot.c
index c64d566d1c..9a6679aca9 100644
--- a/common/autoboot.c
+++ b/common/autoboot.c
@@ -220,7 +220,7 @@ static int __abortboot(int bootdelay)
#endif

#ifdef CONFIG_ARCH_ROCKCHIP
-    if (!IS_ENABLED(CONFIG_CONSOLE_DISABLE_CLI) && ctrlc()) {          /* we
press ctrl+c ? */
+    if ((!IS_ENABLED(CONFIG_CONSOLE_DISABLE_CLI) && ctrlc()) ||
env_get("cli")) { /* we press ctrl+c ? */
        #else
        /*
         * Check if key already pressed
diff --git a/include/boot_rkimg.h b/include/boot_rkimg.h
index cb5781850e..d8ef3e6127 100644
--- a/include/boot_rkimg.h
+++ b/include/boot_rkimg.h
@@ -19,6 +19,7 @@ enum _boot_mode {
    BOOT_MODE_PANIC,
    BOOT_MODE_WATCHDOG,
    BOOT_MODE_DFU,
+    BOOT_MODE_UBOOT_TERMINAL,
    BOOT_MODE_UNDEFINE,
};

```

注：**BOOT\_TO\_UBOOT**的值在内核和U-Boot要一样。

重新编译内核和U-Boot，`./build.sh kernel; ./build.sh uboot`，然后更新boot.img、uboot.img。

开机后输入`reboot uboot`，单板重启后会进U-Boot的终端。

## 5.26 如何在 U-Boot 里支持 USB 大容量存储功能

在对应BoardConfig.mk的RK\_UBOOT\_DEFCONFIG\_FRAGMENT里加上rv1106-usb.config。（其他平台也可以参考）

```
diff --git a/BoardConfig_IPC/BoardConfig-SPI_NAND-NONE-RV1106_EVB1_V11-IPC.mk
b/BoardConfig_IPC/BoardConfig-SPI_NAND-NONE-RV1106_EVB1_V11-IPC.mk
index 558cd57..3abc1cd 100644
--- a/BoardConfig_IPC/BoardConfig-SPI_NAND-NONE-RV1106_EVB1_V11-IPC.mk
+++ b/BoardConfig_IPC/BoardConfig-SPI_NAND-NONE-RV1106_EVB1_V11-IPC.mk
@@ -16,7 +16,7 @@ export RK_BOOT_MEDIUM=spi_nand
 export RK_UBOOT_DEFCONFIG=rv1106_defconfig

# Uboot defconfig fragment
-export RK_UBOOT_DEFCONFIG_FRAGMENT=rk-sfc.config
+export RK_UBOOT_DEFCONFIG_FRAGMENT="rk-sfc.config rv1106-usb.config"

# Kernel defconfig
export RK_KERNEL_DEFCONFIG=rv1106_defconfig
```

**rv1106-usb.config**配置如下：

```
CONFIG_DM_REGULATOR=y
CONFIG_DM_REGULATOR_FIXED=y
CONFIG_DM_REGULATOR_GPIO=y
CONFIG_CMD_USB=y
CONFIG_USB=y
CONFIG_USB_XHCI_HCD=y
CONFIG_USB_DWC3=y
CONFIG_USB_DWC3_GENERIC=y
CONFIG_USB_STORAGE=y
CONFIG_PHY_ROCKCHIP_INNO_USB2=y
```

## 5.27 如何通过指令修改GPIO寄存器配置

GPIO全部寄存器与配置指令可见文档

`<SDK>/docs/zh/bsp/Rockchip_XXXX_User_Manual_GPIO.pdf`，指令在U-Boot终端和根文件系统中有所不同，下面将分别阐述。

**注：**使用引脚复用功能时，需提前确认该引脚是否已被某项功能占用。

## RV1106 GPIO

Name	Config	Description	Register	Position	Value	IO Command
GPIO0A0	IOMUX	UART0_RX_M0	0xFF388000	[0,2]	1	io -4 0xFF388000 0x00070001
		CLK_32K	0xFF388000	[0,2]	2	io -4 0xFF388000 0x00070002
		CLK_REFOUT	0xFF388000	[0,2]	3	io -4 0xFF388000 0x00070003
		RTC_CLKO	0xFF388000	[0,2]	4	io -4 0xFF388000 0x00070004
		GPIO0_A0	0xFF388000	[0,2]	0	io -4 0xFF388000 0x00070000
	Pull	normal	0xFF388038	[0,1]	0	io -4 0xFF388038 0x00030000
		pullup	0xFF388038	[0,1]	1	io -4 0xFF388038 0x00030001
		pulldown	0xFF388038	[0,1]	2	io -4 0xFF388038 0x00030002
	I/O Mode	input	0xFF380008	[0,0]	0	io -4 0xFF380008 0x00010000
		output	0xFF380008	[0,0]	1	io -4 0xFF380008 0x00010001
	Output	low	0xFF380000	[0,0]	0	io -4 0xFF380000 0x00010000
		high	0xFF380000	[0,0]	1	io -4 0xFF380000 0x00010001
	Input Ctrl	disable	0xFF388030	[0,0]	0	io -4 0xFF388030 0x00010000
		enable	0xFF388030	[0,0]	1	io -4 0xFF388030 0x00010001
	Schmitt	disable	0xFF388058	[0,0]	0	io -4 0xFF388058 0x00010000
		enable	0xFF388058	[0,0]	1	io -4 0xFF388058 0x00010001
	Drive Strength	level0	0xFF388010	[0,5]	1	io -4 0xFF388010 0x003F0001
		level1	0xFF388010	[0,5]	3	io -4 0xFF388010 0x003F0003
		level2	0xFF388010	[0,5]	7	io -4 0xFF388010 0x003F0007
		level3	0xFF388010	[0,5]	15	io -4 0xFF388010 0x003F000F
		level4	0xFF388010	[0,5]	31	io -4 0xFF388010 0x003F001F
		level5	0xFF388010	[0,5]	63	io -4 0xFF388010 0x003F003F
	ExtPort	read gpio data in input mode	0xFF380070	[0,0]		io -4 0xFF380070

### 5.27.1 U-Boot终端

U-Boot终端使用 `md` 指令显示寄存器值，使用 `mw` 指令写入寄存器值。

- 以设置GPIO0A0输出高电平为例：

```
# 复用模式：GPIO (GPIO0A0->IOMUX->GPIO0_A0)
=> mw.l 0xFF388000 0x00070000
# I/O模式：输出 (GPIO0A0->I/O Mode->output)
=> mw.l 0xFF380008 0x00010001
# 输出电平：高电平 (GPIO0A0->Output->high)
=> mw.l 0xFF380000 0x00010001
```

- 以读取GPIO0A0输入数据为例：

```
# 复用模式：GPIO (GPIO0A0->IOMUX->GPIO0_A0)
=> mw.l 0xFF388000 0x00070000
# I/O模式：输入 (GPIO0A0->I/O Mode->input)
=> mw.l 0xFF380008 0x00010000
# 读取数据：16字节 (GPIO0A0->ExtPort) (4个long，即4*4=16字节)
=> md.l 0xFF380070 0x4
ff380070: 00000008 00000000 0101157c 00000000
```

### 5.27.2 根文件系统

根文件系统使用 `io` 指令读写寄存器值。

- 以设置GPIO0A0输出高电平为例：



```
# 复用模式：GPIO (GPIO0A0->IOmux->GPIO0_A0)
io -4 0xFF388000 0x00070000
# I/O模式：输出 (GPIO0A0->I/O Mode->output)
io -4 0xFF380008 0x00010001
# 输出电平：高电平 (GPIO0A0->Output->high)
io -4 0xFF380000 0x00010001
```

- 以读取GPIO0A0输入数据为例：

```
# 复用模式：GPIO (GPIO0A0->IOmux->GPIO0_A0)
io -4 0xFF388000 0x00070000
# I/O模式：输入 (GPIO0A0->I/O Mode->input)
io -4 0xFF380008 0x00010000
# 读取数据：16字节 (GPIO0A0->ExtPort) (以4个字节为单位)
io -4 0xFF380070
ff380070:  00000008
io -4 0xFF380074
ff380074:  00000000
io -4 0xFF380078
ff380078:  0101157c
io -4 0xFF38007c
ff38007c:  00000000
```

## 5.28 如何使用UVC预览

SDK提供UVC功能，厂线上可使用UVC功能对摄像头进行预览调焦。

- 打开UVC编译

以project/cfg/BoardConfig\_IPC/BoardConfig-SPI\_NAND-NONE-RV1106\_EVB1\_V11-IPC.mk为例：

1. RK\_APP\_TYPE增加UVC\_TINY重新编译
2. 使能RK\_ENABLE\_RNDIS，打开内核驱动
3. 安装rk\_mpi\_uvc和usb\_config.sh

```
# app config
-export RK_APP_TYPE=RKIPC_RV1106
+export RK_APP_TYPE="RKIPC_RV1106 UVC_TINY"

+ # enable rndis
+ export RK_ENABLE_RNDIS=y
```

编译命令

```
./build.sh all
```

- 板端测试命令

```
RkLunch-stop.sh

# 配置，2304/1296是摄像头的宽高
usb_config.sh 2304 1296

rk_mpi_uvc &
```

- PC端预览

OTG连接开发板与PC，PC端使用UVC预览工具开USB设备进行预览。

## 5.29 如何开启OP-TEE功能

### 5.29.1 OP-TEE介绍

OP-TEE介绍请先阅读参考文档：

```
docs/zh/security/Rockchip_Developer_Guide_TEE_SDK_CN.pdf
```

### 5.29.2 打开U-Boot和kernel的OP-TEE配置

以project/cfg/BoardConfig\_IPC/BoardConfig-SPI\_NAND-NONE-RV1106\_EVB1\_V11-IPC.mk为例

- 在RK\_UBOOT\_DEFCONFIG中添加rv1106-optee.config
- 在RK\_KERNEL\_DEFCONFIG\_FRAGMENT中添加rv1106-tee.config
- RK\_PARTITION\_CMD\_IN\_ENV调整uboot分区大小为512K。对于SPI NAND/SPI NOR/SLC NAND还需要增加1M大小的security分区。

```
diff --git a/BoardConfig_IPC/BoardConfig-SPI_NAND-NONE-RV1106_EVB1_V11-IPC.mk
b/BoardConfig_IPC/BoardConfig-SPI_NAND-NONE-RV1106_EVB1_V11-IPC.mk
index ddda257..9ea99f1 100644
--- a/BoardConfig_IPC/BoardConfig-SPI_NAND-NONE-RV1106_EVB1_V11-IPC.mk
+++ b/BoardConfig_IPC/BoardConfig-SPI_NAND-NONE-RV1106_EVB1_V11-IPC.mk
@@ -16,13 +16,13 @@ export RK_BOOT_MEDIUM=spi_nand
 export RK_UBOOT_DEFCONFIG=rv1106_defconfig

# Uboot defconfig fragment
-export RK_UBOOT_DEFCONFIG_FRAGMENT=rk-sfc.config
+export RK_UBOOT_DEFCONFIG_FRAGMENT="rk-sfc.config rv1106-optee.config"

# Kernel defconfig
export RK_KERNEL_DEFCONFIG=rv1106_defconfig

# Kernel defconfig fragment
-export RK_KERNEL_DEFCONFIG_FRAGMENT=rv1106-evb.config
+export RK_KERNEL_DEFCONFIG_FRAGMENT="rv1106-evb.config rv1106-tee.config"

# Kernel dts
export RK_KERNEL_DTS=rv1106g-evb1-v11.dts
@@ -48,7 +48,7 @@ export RK_BOOTARGS_CMA_SIZE="66M"
# <partdef> := <size>[<offset>] (part-name)

# Note:
```

```
# If the first partition offset is not 0x0, it must be added. Otherwise, it
needn't adding.
-export
RK_PARTITION_CMD_IN_ENV="256K(env),256K@256K(idblock),256K(uboot),8M(boot),32M(
rootfs),48M(oem),32M(userdata),-(media)"
+export
RK_PARTITION_CMD_IN_ENV="256K(env),256K@256K(idblock),512K(uboot),1M(security),
8M(boot),32M(rootfs),48M(oem),32M(userdata),-(media)"
```

注意：如果使用的是emmc，uboot还需要在sysdrv/source/uboot/u-boot/rv1106-optee.config中添加

```
diff --git a/configs/rv1106-optee.config b/configs/rv1106-optee.config
index dc76adc6ff..5425810c21 100644
--- a/configs/rv1106-optee.config
+++ b/configs/rv1106-optee.config
@@ -3,3 +3,4 @@ CONFIG_OPTEE_CLIENT=y
CONFIG_OPTEE_V2=y
CONFIG_SPL_FIT_IMAGE_KB=512
CONFIG_SPL_OPTEE=y
+CONFIG_SUPPORT_EMMC_RPMB=y
```

### 5.29.3 配置U-Boot OP-TEE固件

cd sysdrv/source/uboot/rkbin

```
diff --git a/RKTRUST/RV1106TOS.ini b/RKTRUST/RV1106TOS.ini
index 0169208..677cc0c 100644
--- a/RKTRUST/RV1106TOS.ini
+++ b/RKTRUST/RV1106TOS.ini
@@ -4,4 +4,4 @@ COMPRESSION=lzma
TOSTA=bin/rv11/rv1106_tee_ta_v1.00.bin
ADDR=0x03000000
[MCU]
-MCU0=bin/rv11/rv1106_hpmcu_wrap_v1.56.bin,0xff6ff000,disabled
+MCU0=bin/rv11/rv1106_hpmcu_wrap_v1.56.bin,0xff6ff000,okay
```

注：**ADDR**是TEE运行地址，不可修改。**TOSTA**需要根据具体版本指定对应的bin文件。

### 5.29.4 打开kernel OP-TEE驱动

cd sysdrv/source/kernel/

```
diff --git a/arch/arm/boot/dts/rv1106.dtsi b/arch/arm/boot/dts/rv1106.dtsi
index f877096e8060..0e8244465ab9 100644
--- a/arch/arm/boot/dts/rv1106.dtsi
+++ b/arch/arm/boot/dts/rv1106.dtsi
@@ -235,7 +235,7 @@ firmware {
        optee: optee {
                compatible = "linaro,optee-tz";
                method = "smc";
-               status = "disabled";
+               status = "okay";
        };
};
```

cd sysdrv/drv\_ko

```
diff --git a/insmod_ko.sh b/insmod_ko.sh
index 35ecb02..d489aab 100644
--- a/insmod_ko.sh
+++ b/insmod_ko.sh
@@ -74,7 +74,8 @@ __insmod motor.ko

__chk_camera_sensor_height

-__insmod rokit.ko mcu_fw_path="./hpmcu_wrap.bin" mcu_fw_addr=0xff6ff000
isp_max_h=$sensor_height
+__insmod rokit.ko isp_max_h=$sensor_height

udevadm control --start-exec-queue
```

确认驱动开启：

若出现 /dev/opteearmtz00 节点，说明optee v1的TEE linux kernel驱动已开启；

若出现 /dev/tee0 和 /dev/teepriv0 节点，说明optee v2的TEE linux kernel驱动已开启。

### 5.29.5 在RV1106 SDK中编译TEE库文件

TEE环境相关组件在linux工程目录 media/security/bin 下（包含V1与V2版本，需根据不同平台采用不同版本文件）：

1. lib：包含32bit与64bit平台编译出来的tee-supplciant、libteec.so以及其他CA相关库文件。
2. ta：存放编译好的TA文件。

RV1106是32bit使用V2版本，编译：

```
# 全部编译
./build.sh all

# 编译TEE
cd media/security/rk_tee_user/v2
./build.sh 3232
```

编译后push相关文件到开发板验证：

```
# 在SDK根目录执行如下命令
cp ./media/security/bin/optee_v2/uclibc_lib/arm/libteeec.so*
output/out/rootfs_uclibc_rv1106/lib/ -fv
cp ./media/security/bin/optee_v2/uclibc_lib/arm/tee-supplciant
./media/security/rk_tee_user/v2/out/rk_test/rktest
output/out/rootfs_uclibc_rv1106/bin/ -fv
mkdir output/out/rootfs_uclibc_rv1106/lib/optee_armtz -p
cp ./media/security/rk_tee_user/v2/out/ta/rk_test/ldb57234-dacd-462d-9bb1-
ae79de44e2a5.ta output/out/rootfs_uclibc_rv1106/lib/optee_armtz
./build.sh firmware
```

## 5.29.6 测试验证

- uboot中验证OP-TEE

开机按住ctrl+c进入uboot cmd模式，运行指令 `mmc testsecurestorage`：

```
=> <INTERRUPT>
=> mmc testsecurestorage
optee api revision: 2.0
test_secure_storage_default success! 1/100
test_secure_storage_security_partition success! 1/100
test_secure_storage_default success! 2/100
test_secure_storage_security_partition success! 2/100
test_secure_storage_default success! 3/100
```

- 内核中验证OP-TEE

开机进入Linux Console之后，运行测试TEE环境，运行CA，调用TA，测试TEE相关功能：

```
$ tee-supplciant &
$ rktest transfer_data

INF [827] TEES:tee_supp_rk_fs_init:123: get_rkss_version version=2

INF [827] TEES:process_one_request:650: rk secure storage is available. We
prefer to use it.
I/TA: Hello!
I/TA: Get and update the params[0].
I/TA: Get the params[1]: Transfer data test.
I/TA: Copy params[1] to params[2].
Done.
test value : Pass!
I/TA: Goodbye!
test buffer : Pass!
Test OK.
#
```

## 5.30 休眠唤醒功能开发

### 5.30.1 如何开启休眠唤醒功能

以project/cfg/BoardConfig\_IPC/BoardConfig-SPI\_NAND-NONE-RV1106\_EVB1\_V11-IPC.mk为例：

在RK\_KERNEL\_DEFCONFIG\_FRAGMENT中添加rv1106-pm.config配置后重新编译：

```
diff --git a/BoardConfig_IPC/BoardConfig-SPI_NAND-NONE-RV1106_EVB1_V11-IPC.mk
b/BoardConfig_IPC/BoardConfig-SPI_NAND-NONE-RV1106_EVB1_V11-IPC.mk
index ddda257..8177170 100644
--- a/BoardConfig_IPC/BoardConfig-SPI_NAND-NONE-RV1106_EVB1_V11-IPC.mk
+++ b/BoardConfig_IPC/BoardConfig-SPI_NAND-NONE-RV1106_EVB1_V11-IPC.mk
@@ -22,7 +22,7 @@ export RK_UBOOT_DEFCONFIG_FRAGMENT=rk-sfc.config
 export RK_KERNEL_DEFCONFIG=rv1106_defconfig

# Kernel defconfig fragment
-export RK_KERNEL_DEFCONFIG_FRAGMENT=rv1106-evb.config
+export RK_KERNEL_DEFCONFIG_FRAGMENT="rv1106-evb.config rv1106-pm.config"

# Kernel dts
export RK_KERNEL_DTS=rv1106g-evb1-v11.dts
```

### 5.30.2 休眠唤醒方案简介

SDK的休眠唤醒采用的方案为：

- 进入休眠状态描述：

用户层收到触发事件后，暂停业务流程并主动调用 `echo mem > /sys/power/state` 命令使系统进入休眠状态。在此期间，除了CPU内部PMU和DDR自刷新之外，其他模块包括LOGIC&ARM都会关掉以节省电量。休眠流程为：按键OR其他自定义休眠事件--->应用层收到事件--->应用、系统服务休眠前的准备工作--->进入休眠

- 唤醒系统描述：

由于在休眠状态下所有模块都已经断电，因此为了保证唤醒的成功，必须要在进入休眠前保存好一些重要的状态信息，例如设备的配置参数等。同时，在唤醒时也要及时恢复这些状态信息，确保设备能够正确地回到休眠前的工作状态。SDK已实现了通过PMUIO唤醒，之后用户层应实现休眠前终止掉的业务流程。

这里可用的唤醒源为Group GPIO0\_A,客户根据实际原理图设计。

唤醒流程：硬件中断唤醒--->PMUIO--->SRAM--->BOOTROM、ddr.bin--->KERNEL--->应用从之前中断的位置继续执行--->应用读取唤醒原因--->应用恢复

### 5.30.3 调试

开启休眠唤醒相关的打印

```
echo 1 > /sys/power/pm_print_times
```

强制系统进入休眠

```
echo mem > /sys/power/state
```

### 5.30.4 硬件参考设计

硬件设计时需要参考RV1106 CVR硬件参考设计。参考设计获取方法如下：

请通过技术服务窗口获取参考设计<<RV1106G\_CVR\_REF\_V10>>、<<RV1103G\_CVR\_REF\_V10>> RV1103也可以PMU单独供电，但由于RV1103的pmuio电源域只引出了一个GPIO，如果接了就没有IO来控制SOC PWREN，无法给ARM和LOGIC上电，因此需要其他GPIO来做唤醒源。

建议RV1103使用外置MCU，用MCU的IO来控制RV1103的电源使能。

### 5.30.5 唤醒按键配置举例

这里配置gpio0 RK\_PA1为唤醒源：

注：RV1106\_EVB1板子的gpio0 RK\_PA1默认接SD卡，需要修改硬件才可以做GPIO唤醒。

```
diff --git a/arch/arm/boot/dts/rv1106g-evb1-v11.dts
b/arch/arm/boot/dts/rv1106g-evb1-v11.dts
index 8f820d60fecc..f1daa263401c 100644
--- a/arch/arm/boot/dts/rv1106g-evb1-v11.dts
+++ b/arch/arm/boot/dts/rv1106g-evb1-v11.dts
@@ -12,6 +12,29 @@ / {
    compatible = "rockchip,rv1106g-evb1-v11", "rockchip,rv1106";

    /delete-node/ vcc5v0-usb;

+
+   gpio-keys {
+       compatible = "gpio-keys";
+       autorepeat;
+       pinctrl-names = "default";
+       pinctrl-0 = <&pwr_key>;
+       power-key {
+           gpios = <&gpio0 RK_PA1 GPIO_ACTIVE_HIGH>;
+           linux,code = <KEY_POWER>;
+           label = "GPIO Key Power";
+           debounce-interval = <100>;
+           wakeup-source;
+       };
+   };
+};
+
+&pinctrl {
+   buttons {
+       pwr_key:pwr-key {
+           rockchip,pins = <0 RK_PA1 RK_FUNC_GPIO
&pcfg_pull_none>;
+       };
+   };
+};
+};
```

## 6. 注意事项

---

在windows下复制源码包时，linux下的可执行文件可能变为非可执行文件，或者软连接失效导致无法编译使用。

因此使用时请注意不要在windows下复制源代码包。