密级状态：绝密( ) 秘密( ) 内部( ) 公开(√)

Rockchip Android 11.0 SDK开发指南

文件状态:
[ ] 草稿
[√] 正式发布
[ ] 正在修改

文件标识:	RK-KF-YF-279
当前版本:	V1.1.4
作者:	吴良清
完成日期:	2021-07-12
审核:	陈海燕
审核日期:	2021-07-12

版本号	作者	修改日期	修改说明	备注
V0.0.1	吴良清	2020-12-25	发布RK3566/RK3568 Alpha版本	
V0.0.2	卞金晨	2021-01-06	发布PX30/RK3326 Beta版本	
V1.0.0	吴良清	2021-01-29	增加RK3566/RK3568 EVB板编译方法	
V1.1.0	吴良清	2021-02-23	发布RK3399 Alpha版本	
V1.1.1	吴良清	2021-03-09	修改单独编译kernel的说明	
V1.1.2	吴良清	2021-05-12	支持RK3288W芯片平台	
V1.1.3	吴良清	2021-05-23	增加常见问题说明	
V1.1.4	吴良清	2021-07-12	支持RK3566 BOX产品形态，支持RK3328 BOX产品形态，增加repo服务器搭建及常见问题说明	

文档问题反馈：wlq@rock-chips.com

免责声明

商标声明

版权所有 © 2020 瑞芯微电子股份有限公司

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
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

Rockchip Android 11.0 SDK支持芯片

芯片平台	是否支持	SDK版本
RK3566	支持	RKR1
RK3568	支持	RKR1
PX30/RK3326	支持	RKR1
RK3399	支持	RKR5
RK3288W	支持	RKR7
RK366-BOX	支持	RKR9
RK3328	支持	RKR9

Rockchip Android 11.0 SDK代码下载编译

代码下载

下载地址

```
repo init --repo-url=ssh://git@www.rockchip.com.cn:2222/repo-
release/tools/repo.git  -u
ssh://git@www.rockchip.com.cn:2222/Android_R/manifests.git -m Android11.xml
```

服务器镜像下载

```
repo init --repo-url=ssh://git@www.rockchip.com.cn:2222/repo-
release/tools/repo.git  -u
ssh://git@www.rockchip.com.cn:2222/Android_R/manifests.git -m Android11.xml --
mirror
```

注，repo是google用Python脚本写的调用git的一个脚本，主要是用来下载、管理Android项目的软件仓库，其下载地址如下：

```
git clone ssh://git@www.rockchip.com.cn:2222/repo-release/tools/repo
```

为方便客户快速获取SDK源码，瑞芯微技术窗口通常会提供对应版本的SDK初始压缩包。以`ROCKCHIP_ANDROID11.0_SDK_RELEASE.tar.gz.*`为例，拷贝到该初始化包后，通过如下命令可检出源码：

```
mkdir ROCKCHIP_ANDROID11.0_SDK_RELEASE
cat ROCKCHIP_ANDROID11.0_SDK_RELEASE.tar.gz* | tar -zx -C
ROCKCHIP_ANDROID11.0_SDK_RELEASE
cd ROCKCHIP_ANDROID11.0_SDK_RELEASE
.repo/repo/repo sync -l
.repo/repo/repo sync -c
```

搭建自己的repo代码服务器

环境

安装`openssh-server`用于远程登录，`git`用于管理工程，`keychain`用于公私钥管理工具

```
sudo apt-get install openssh-server git keychain
```

gitolite搭建

服务器端操作

（以服务器地址：10.10.10.206为例进行说明）

1. 创建git账户：

```
sudo adduser --system --shell /bin/bash --group git
sudo passwd git
```

2. 以“git”账户登录服务器
3. 确保“~/.ssh/authorized_keys”为空或者不存在
4. 拷贝服务器管理员的公钥到“~/YourName.pub”
5. 下载gitolite源码

```
git clone https://github.com/sitaramc/gitolite.git
```

6. 在git用户目录下创建bin目录

```
mkdir -p ~/bin
```

7. 执行下列命令安装gitolite，不同版本安装方法不同，请参考源码中的文档：

```
gitolite/install -to ~/bin
```

8. 设置管理员

```
~/bin/gitolite setup -pk YourName.pub
```

客户端操作

1. 克隆服务器的gitolite管理仓库：

```
git clone ssh://git@10.10.10.206/gitolite-admin.git
```

2. 添加用户公钥到gitolite目录下

```
cp username.pub keydir/username.pub
```

3. 添加管理员用户

```
vi conf/gitolite.conf
@admin = admin1 admin2 admin3
repo gitolite-admin
RW+     =   @admin
```

repo镜像搭建

服务器端操作

1. 用git账号登入服务器
2. 在根目录下载repo工具

```
git clone ssh://git@www.rockchip.com.cn:2222/repo-release/tools/repo
```

3. 新建RK_Android11_mirror目录

```
mkdir RK_Android11_mirror
```

4. 进入 RK_Android11_mirror目录

```
cd RK_Android11_mirror
```

5. 下载RK Android11 SDK镜像

```
~/repo/repo init --repo-url=ssh://git@www.rockchip.com.cn:2222/repo-
release/tools/repo.git  -u
ssh://git@www.rockchip.com.cn:2222/Android_R/manifests.git -m Android11.xml --
mirror
```

6. 创建仓库组权限

```
.repo/repo/repo list -n > android_r.conf
sed -i 's/^/@android_r = RK_Android11_mirror\/&/g' android_r.conf
```

客户端操作

1. 将服务器端的android_r.conf拷贝到客户端的·gitolite-admin/conf/·下
2. 添加组权限

```
vi conf/android_r.conf
@usergroup = user1 user2 user3
repo @android_r
R   = @usergroup
RW+ = @admin
```

```
vi conf/gitolite.conf
include "android_r.conf"
```

5. 新建自己的manifests仓库

```
vi conf/android_r.conf
@android_r = Android_R/manifests_xxx
```

客户端操作

1. 在客户端下载manifests_xxx仓库
   在其他客户端电脑上下载manifests_xxx.git仓库

```
git clone ssh://git@10.10.10.206/Android_R/manifests_xxx.git
```

2. 在客户端下载原始manifests仓库

```
git clone ssh://git@10.10.10.206/Android_R/manifests.git
```

3. 提交manifest.xml文件到新建的manifest_xxx仓库中
   将原始manifests下面的文件拷贝到的manifests_xxx内

```
cd manifests_xxx
cp -rf manifests/*.xml manifests_xxx/
```

查看拷贝文件

```
git status

 Android11.xml
 Android11_Express.xml
 default.xml
 include/partner_gms_express.xml
 include/partner_modules_express.xml
 include/rk3288_repository.xml
 include/rk3326_repository.xml
 include/rk3399_repository.xml
 include/rk356x_repository.xml
 include/rk_checkout_from_aosp.xml
 include/rk_modules_repository.xml
 remote.xml
 remove_r.xml
```

本地提交

```
 git add -A
 git commit -m "init xxx"
```

push到远程分支

```
 git push origin master:master
```

7. 创建自己的代码下载链接
   在根目录下下载repo工具

```
git clone ssh://git@www.rockchip.com.cn:2222/repo-release/tools/repo
```

按以上步骤操作后，自己的代码下载链接如下

```
mkdir Android11
cd Android11
~/repo/repo init -u ssh://git@10.10.10.206/Android_R/manifests_xxx.git -m
Android11.xml
```

其中：
`//10.10.10.206` 是你的服务器端地址
通过以上步骤就可以完成自己的repo服务器搭建了，可以把自己的代码服务器链接分享给同事们一起工作了。

代码管理

通过以上步骤搭建代码服务器后大部分代码仓库都使用RK默认的分支，如果有仓库需要修改自己的代码，可以参考下面的步骤进行操作。

切换自己的代码分支

1. 进入需要修改的代码仓库，以kernel目录为例进行说明

```
cd kernel
```

2. 切换一个本地分支

```
git checkout remotes/m/master -b xxx_branch
```

3. push xxx_branch分支到远程服务器

```
git push rk29 xxx_branch:xxx_branch
```

其中`rk29`是remote 可以直接tab键自动补全

4. 进入.repo/manifests目录修改manifest里面指定的分支
   进入.repo/manifests目录通过grep kernel可以找到kernel仓库对应的manifest的位置

```
cd .repo/manifests
```

```
--- a/include/rk_modules_repository.xml
+++ b/include/rk_modules_repository.xml
@@ -10,7 +10,7 @@
   <project path="hardware/rockchip/libgraphicpolicy"
name="rk/hardware/rk29/libgraphicpolicy" remote="rk"
revision="refs/tags/android-11.0-mid-rkr8" />
   <project path="hardware/rockchip/libhwjpeg" name="rk/hardware/rk29/libhwjpeg"
remote="rk" revision="refs/tags/android-11.0-mid-rkr8"/>
   <project path="u-boot" name="rk/u-boot" remote="rk"
revision="refs/tags/android-11.0-mid-rkr8"/>
-  <project path="kernel" name="rk/kernel" remote="rk29"
revision="refs/tags/android-11.0-mid-rkr8"/>
+  <project path="kernel" name="rk/kernel" remote="rk29" revision="xxx_branch"/>

   <project path="bootable/recovery/rkupdate"
name="platform/bootable/recovery/rk_update" remote="rk"
revision="refs/tags/android-11.0-mid-rkr8"/>
   <project path="bootable/recovery/rkutility"
name="platform/bootable/recovery/rk_utility" remote="rk"
revision="refs/tags/android-11.0-mid-rkr8"/>
```

5. 提交修改的manifest到远程分支

```
git add include/rk_modules_repository.xml
git commit -m "change kernel branch on xxx_branch"
git push origin default:master
```

提交manifests仓库后，其他同事就可以同步到你们自己的分支的kernel代码了。

代码修改提交

按上面步骤切换完分支后就可以在自己分支上提交自己的修改了， 提交直接push到xxx_branch分支上面。

同步RK的代码

1. 同步RK代码需要在服务器端进行sync操作

```
cd RK_Android11_mirror
```

```
.repo/repo/repo sync -c
```

2. 客户端合并RK对manifests的修改

- 下载RK原始manifests仓库

```
git clone //10.10.10.206/wlq/test/manifests.git
```

使用对比工具对比manifests（RK原始）和manifests_xxx(自己的)，将RK修改的差异部分合并到自己的仓库中（主要修改tag，增加删除仓库等）。

- 对比确认后将修改push到manifests_xxx上。

这步也可以确认自己修改了哪些仓库，在下一步中将进行修改仓库的合并。

3. 有自己切分支的目录需要手动把RK的修改merge到自己的分支上面
   以kernel为例：

- 查看当前指向的远程分支

```
wlq@wlq:~/home1/test2/kernel$ git branch -av
* android-11.0-mid-rkr7   0bde59fad73a ARM: configs: rockchip_defconfig enable
ION_CMA_HEAP
  xxx_branch              0bde59fad73a ARM: configs: rockchip_defconfig enable
ION_CMA_HEAP
  remotes/m/master        -> rk29/xxx_branch
  remotes/rk29/xxx_branch 0bde59fad73a ARM: configs: rockchip_defconfig enable
ION_CMA_HEAP
```

可以看到当前指向的是：`remotes/m/master -> rk29/xxx_branch`

- 创建本地分支（从自己的远程分支上切）

```
git checkout remotes/m/xxx_branch -b local_xxx_branch
```

- 确认当前RK发布的最新TAG

```
wlq@wlq:~/home1/test2/kernel$ git tag | grep rkr
android-10.0-mid-rkr1
android-10.0-mid-rkr10
android-10.0-mid-rkr11
android-10.0-mid-rkr13
android-10.0-mid-rkr2
android-10.0-mid-rkr3
android-10.0-mid-rkr4
android-10.0-mid-rkr5
android-10.0-mid-rkr6
android-10.0-mid-rkr7
android-10.0-mid-rkr8
android-10.0-mid-rkr9
android-11.0-ebook-rkr1
android-11.0-ebook-rkr2
android-11.0-ebook-rkr3
android-11.0-ebook-rkr4
android-11.0-ebook-rkr5
android-11.0-ebook-rkr6
android-11.0-mid-rkr1
android-11.0-mid-rkr2
android-11.0-mid-rkr3
android-11.0-mid-rkr4
android-11.0-mid-rkr4.1
android-11.0-mid-rkr5
android-11.0-mid-rkr6
android-11.0-mid-rkr7
android-11.0-mid-rkr7-prev
android-11.0-mid-rkr8
```

可以看到当前最新的Android11的tag是`android-11.0-mid-rkr8`

- 合并`android-11.0-mid-rkr8`到本地分支

```
git merge android-11.0-mid-rkr8
```

查看是否有冲突，如果有冲突先解决冲突，没有冲突在执行下一步

- push合并完的代码到远程分支

```
git push rk29 local_xxx_branch:xxx_branch
```

- 其他切分的目录都按这个方式进行合并提交即可

代码编译

一键编译命令

```
./build.sh -UKAup
（ WHERE: -U = build uboot
       -C = build kernel with Clang
       -K = build kernel
       -A = build android
       -p = will build packaging in IMAGE
       -o = build OTA package
       -u = build update.img
       -v = build android with 'user' or 'userdebug'
       -d = huild kernel dts name
       -V = build version
       -J = build jobs
   ------------大家可以按需使用，不用记录uboot/kernel编译命令了------------------
 ）

============================================================
请注意使用一键编译命令之前需要设置环境变量，选择好自己需要编译的平台，举例：
source build/envsetup.sh
lunch rk3566_rgo-userdebug
============================================================
```

各个平台编译命令汇总

Soc	类型	机型	Android	一键编译	kernel编译	uboot编译
RK3566	平板	样机	build/envsetup.sh;lunch rk3566_rgo-userdebug	./build.sh -AUCKu	make ARCH=arm64 rockchip_defconfig android-11.config;make ARCH=arm64 rk3566-rk817-tablet.img -j24	./make.sh rk3566
RK3568	开发板	EVB1-DDR4-V10	build/envsetup.sh;lunch rk3568_r-userdebug	./build.sh -AUCKu	make ARCH=arm64 rockchip_defconfig rk356x_evb.config android-11.config;make ARCH=arm64 rk3568-evb1-ddr4-v10.img -j24	./make.sh rk3568
RK3568	开发板	EVB2-LPDDR4X-V10	build/envsetup.sh;lunch rk3568_r-userdebug	./build.sh -AUCKu	make ARCH=arm64 rockchip_defconfig rk356x_evb.config android-11.config;make ARCH=arm64 rk3568-evb2-lp4x-v10.img -j24	./make.sh rk3568
RK3568	开发板	EVB4-LPDDR3-V10	build/envsetup.sh;lunch rk3568_r-userdebug	./build.sh -AUCKu	make ARCH=arm64 rockchip_defconfig rk356x_evb.config android-11.config;make ARCH=arm64 rk3568-evb4-lp3-v10.img -j24	./make.sh rk3568
RK3568	开发板	EVB5-DDR4-V10	build/envsetup.sh;lunch rk3568_r-userdebug	./build.sh -AUCKu	make ARCH=arm64 rockchip_defconfig rk356x_evb.config android-11.config;make ARCH=arm64 rk3568-evb5-ddr4-v10.img -j24	./make.sh rk3568
RK3568	开发板	EVB6-DDR3-V10	build/envsetup.sh;lunch rk3568_r-userdebug	./build.sh -AUCKu	make ARCH=arm64 rockchip_defconfig rk356x_evb.config android-11.config;make ARCH=arm64 rk3568-evb6-ddr3-v10.img -j24	./make.sh rk3568
RK3568	开发板	EVB7-DDR4-V10	build/envsetup.sh;lunch rk3568_r-userdebug	./build.sh -AUCKu	make ARCH=arm64 rockchip_defconfig rk356x_evb.config android-11.config;make ARCH=arm64 rk3568-evb7-ddr4-v10.img -j24	./make.sh rk3568
RK3566	开发板	EVB1-DDR4-V10	build/envsetup.sh;lunch rk3566_r-userdebug	./build.sh -AUCKu -d rk3566-evb2-lp4x-v10	make ARCH=arm64 rockchip_defconfig rk356x_evb.config android-11.config;make ARCH=arm64 rk3566-evb1-ddr4-v10.img -j24	./make.sh rk3566
RK3566	开发板	EVB2-LP4X-V10	build/envsetup.sh;lunch rk3566_r-userdebug	./build.sh -AUCKu -d rk3566-evb2-lp4x-v10	make ARCH=arm64 rockchip_defconfig rk356x_evb.config android-11.config;make ARCH=arm64 rk3566-evb2-lp4x-v10.img -j24	./make.sh rk3566

Soc	类型	机型	Android	一键编译	kernel编译	uboot编译
RK3566	开发板	EVB3-DDR3-V10	build/envsetup.sh;lunch rk3566_r-userdebug	./build.sh -AUCKu -d rk3566-evb2-lp4x-v10	make ARCH=arm64 rockchip_defconfig rk356x_evb.config android-11.config;make ARCH=arm64 rk3566-evb3-ddr3-v10.img -j24	./make.sh rk3566
RK3566	开发板	EVB5-LPDDR4X-V10	build/envsetup.sh;lunch rk3566_r-userdebug	./build.sh -AUCKu -d rk3566-evb2-lp4x-v10	make ARCH=arm64 rockchip_defconfig rk356x_evb.config android-11.config;make ARCH=arm64 rk3566-evb5-lp4x-v10.img -j24	./make.sh rk3566
RK3566	BOX-DEMO板	rk3566-box-demo-v10	build/envsetup.sh;lunch rk356x_box-userdebug	./build.sh -AUCKu -d rk3566-box-demo-v10	make ARCH=arm64 rockchip_defconfig rk356x_evb.config android-11.config;make ARCH=arm64 rk3566-box-demo-v10.img -j24	./make.sh rk3566
RK3326	平板	863样机	build/envsetup.sh;lunch rk3326_rgo-userdebug	./build.sh -AUCKu	make ARCH=arm64 rockchip_defconfig android-11-go.config;make ARCH=arm64 rk3326-863-lp3-v10-rksip1.img -j24	./make.sh rk3326
PX30	开发板	px30-evb-ddr3-v10-avb	build/envsetup.sh;lunch PX30_Android11-userdebug	./build.sh -AUCKu	make ARCH=arm64 rockchip_defconfig android-11.config;make ARCH=arm64 px30-evb-ddr3-v10-avb.img -j24	./make.sh px30
RK3399	挖掘机开发板	rk3399-sapphire-excavator-edp-avb	build/envsetup.sh;lunch rk3399_Android11-userdebug	./build.sh -AUCKu -d rk3399-sapphire-excavator-edp-avb	make ARCH=arm64 rockchip_defconfig android-11.config;make ARCH=arm64 rk3399-sapphire-excavator-edp-avb.img -j24	./make.sh rk3399
RK3399	IND行业开发板	rk3399-evb-ind-lpddr4-android-avb	build/envsetup.sh;lunch rk3399_Android11-userdebug	./build.sh -AUCKu	make ARCH=arm64 rockchip_defconfig android-11.config;make ARCH=arm64 rk3399-evb-ind-lpddr4-android-avb.img -j24	./make.sh rk3399
RK3288	开发板	rk3288-evb-android-rk808-edp-avb	build/envsetup.sh;lunch rk3288_Android11-userdebug	./build.sh -AUCKu	make ARCH=arm rockchip_defconfig android-11.config;make ARCH=arm rk3288-evb-android-rk808-edp-avb.img -j24	./make.sh rk3288
RK3328	box开发板	rk3328-box-liantong-avb	build/envsetup.sh;lunch rk3328_box-userdebug	./build.sh -AUCKu	make ARCH=arm rockchip_defconfig android-11.config;make ARCH=arm rk3328-box-liantong-avb.img -j24	./make.sh rk3328

其他编译说明

Android11.0不能直接烧写kernel.img和resource.img

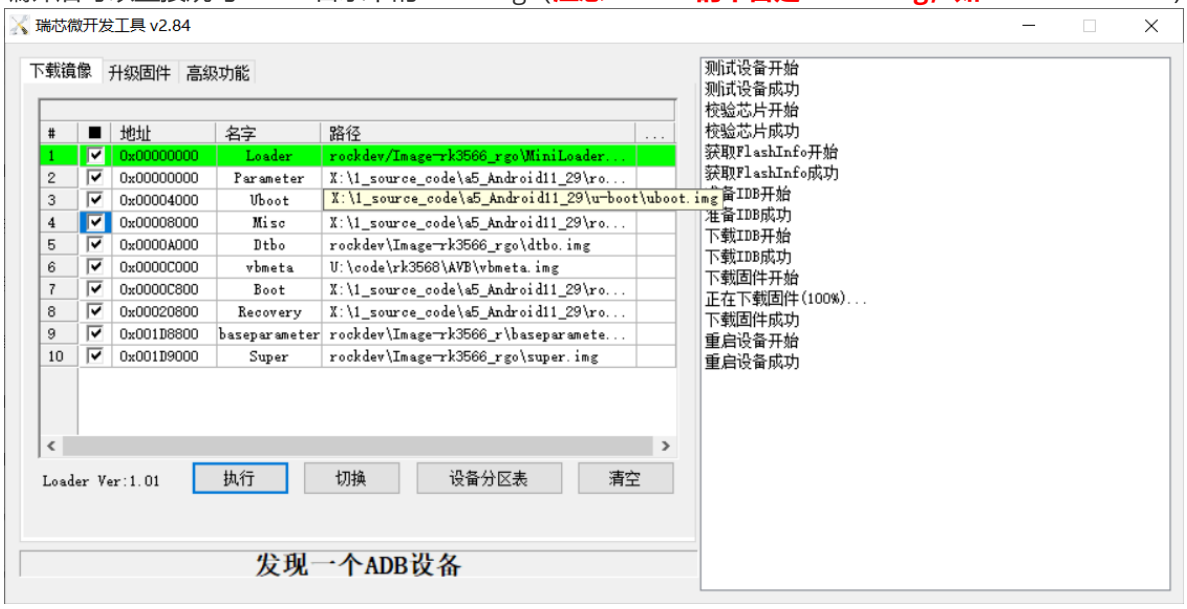
Android11.0的kernel.img和resource.img包含在boot.img中，需要使用build.sh -K 命令来编译kernel。编译后烧写rockdev下面的boot.img。也可以使用如下方法单独编译kernel。

单独编译kernel生成boot.img

编译的原理：在kernel目录下将编译生成的`kernel.img`和`resource.img`替换到旧的`boot.img`中。
以`RK3566`样机为例，编译时替换对应的boot.img及dts：
其中`BOOT_IMG=../rockdev/Image-rk3566_r/boot.img`这里指定的是旧的boot.img的路径，命令如下：

```
cd kernel
make ARCH=arm64 rockchip_defconfig android-11.config
make ARCH=arm64 BOOT_IMG=../rockdev/Image-rk3566_r/boot.img rk3566-rk817-
tablet.img -j24
```

编译后可以直接烧写kernel目录下的boot.img（**注意：32bit的平台是zboot.img，如3126c/rk3288w**）



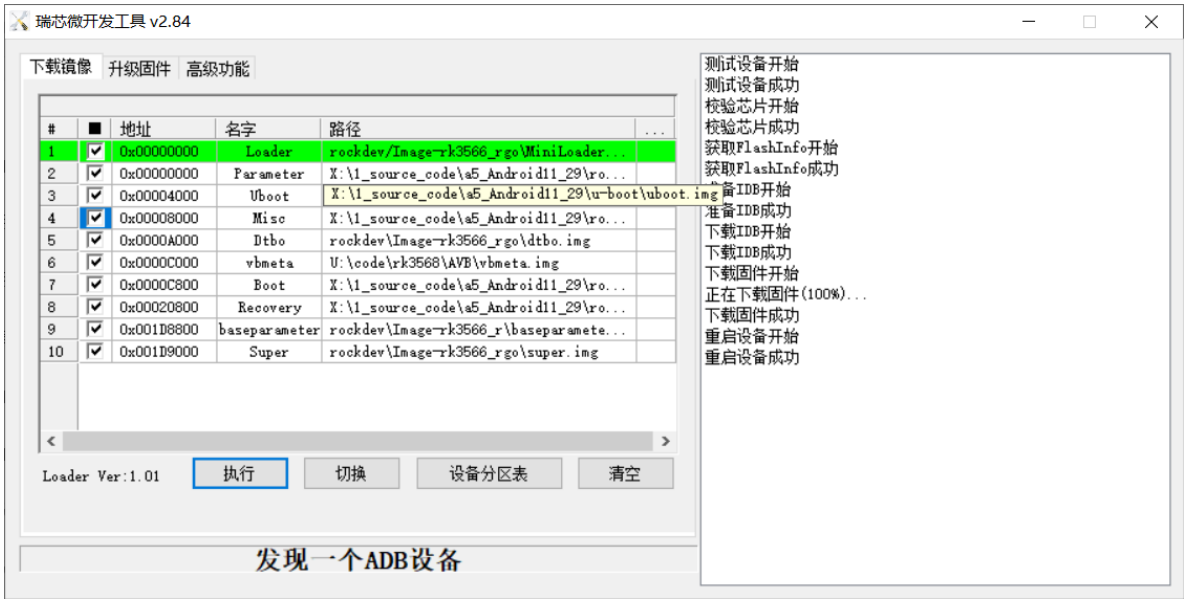
到机器的boot位置，烧写时**请先加载分区表（parameter.txt）**，以免烧写位置错误。

固件烧写

固件烧写工具

Android11的USB驱动DriverAssitant需要更新到V5.1.1版本，可以参考下面的工具章节进行更新。
Windows烧写工具：（工具是时刻更新，请及时同步更新）

```
RKTools/windows/AndroidTool/AndroidTool_Release_v2.84
```




```
RKTools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool_v1.56
```

在下文工具说明章节有详细说明

固件说明

完整编译后会生成如下文件： （以RK3566为例，这里lunch的是rk3566_rgo-userdebug）

```
rockdev/Image-rk3566_rgo/
├── boot-debug.img
├── boot.img
├── config.cfg
├── dtbo.img
├── MiniLoaderAll.bin
├── misc.img
├── parameter.txt
├── pcba_small_misc.img
├── pcba_whole_misc.img
├── recovery.img
├── resource.img
├── super.img
├── uboot.img
├── update.img
└── vbmeta.img
```

工具烧写如下文件即可：(RK3566/RK3568不需要烧写trust.img)

```
rockdev/Image-rk3566_rgo/
├── boot.img
├── dtbo.img
├── MiniLoaderAll.bin
├── misc.img
├── parameter.txt
├── recovery.img
├── super.img
├── uboot.img
└── vbmeta.img
```

也可以直接烧写`update.img`

固件说明

固件	说明
boot.img	包含ramdis、kernel、dtb
boot-debug.img	与boot.img的差别是user固件可以烧写这个boot.img进行root权限操作
dtbo.img	Device Tree Overlays 参考下面的dtbo章节说明
config.cfg	烧写工具的配置文件，可以直接导入烧写工具显示需要烧写的选项
MiniLoaderAll.bin	包含一级loader
misc.img	包含recovery-wipe开机标识信息，烧写后会进行recovery
parameter.txt	包含分区信息
pcba_small_misc.img	包含pcba开机标识信息，烧写后会进入简易版pcba模式
pcba_whole_misc.img	包含pcba开机标识信息，烧写后会进入完整版pcba模式
recovery.img	包含recovery-ramdis、kernel、dtb
super.img	包含odm、product、vendor、system、system_ext分区内容
trust.img	包含BL31、BL32 RK3566/RK3568没有生成这个固件，不需要烧写
uboot.img	包含uboot固件
vbmeta.img	包含avb校验信息，用于AVB校验
update.img	包含以上需要烧写的img文件，可以用于工具直接烧写整个固件包

fastboot烧写动态分区

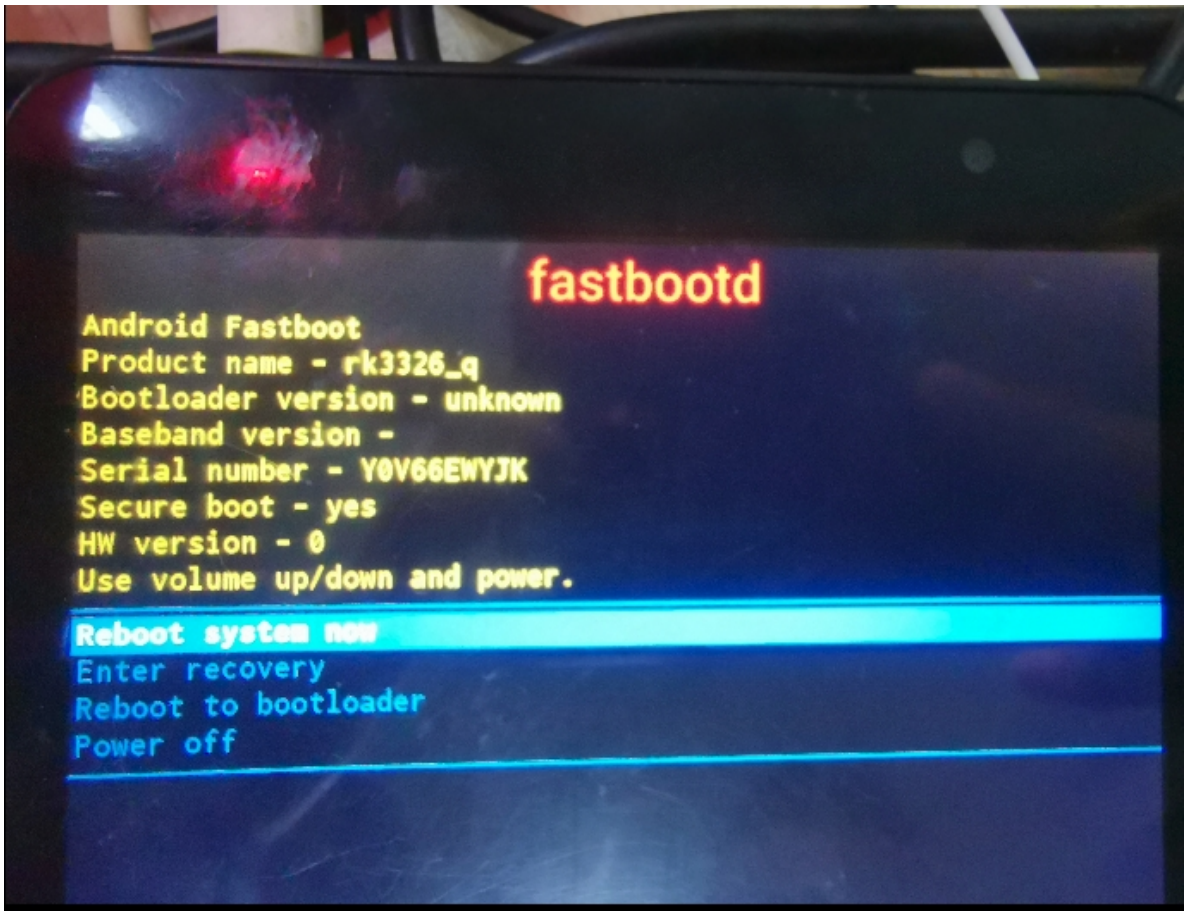
R的新设备支持动态分区，已经移除了system/vendor/odm/product/system_ext分区，请烧写super.img，单独烧写system/vendor/odm等（可以在out下面找到对应img文件）可以用`fastbootd`，要求adb和fastboot版本均为最新，SDK提供了编译好的工具包：

```
RKTools/linux/Linux_adb_fastboot (Linux_x86版本)
RKTools/windows/adb_fastboot (Windows_x86版本)
```

- 使用命令烧写动态分区：

```
adb reboot fastboot
fastboot flash vendor vendor.img
fastboot flash system system.img
fastboot flash odm odm.img
```

注：进入fastbootd模式后，屏幕上会显示相关设备信息，如图所示：



注：非动态分区使用fastboot，请进入bootloader:

```
adb reboot bootloader
```

烧写GSI的方法：

- 确认机器解锁后，进入fastbootd，只需要烧写GSI中的system.img及固件中的misc.img，烧写后会进入recovery进行恢复出厂设置。下面附上整个烧写流程：

1. 重启至bootloader，未解锁->机器解锁：

```
adb reboot bootloader
fastboot oem at-unlock-vboot ## 对于烧写过avb公钥的客户，请参考对应的文档解锁。
```

2. 恢复出厂设置，重启至fastbootd：

```
fastboot flash misc misc.img
fastboot reboot fastboot  ## 此时将进入fastbootd
```

3. 开始烧写GSI

```
fastboot delete-logical-partition product  ## (可选)对于分区空间紧张的设备，可以先执行本条命令删除product分区后再烧写GSI
fastboot flash system system.img
fastboot reboot  ## 烧写成功后，重启
```

- 注：也可以使用DSU(Dynamic System Updates)烧写GSI，目前Rockchip平台已经默认支持DSU。由于该功能需要消耗大量内存，不建议1G DDR及以下的设备使用，有关DSU的说明和使

用，请参考Android官网：
https://source.android.com/devices/tech/ota/dynamic-system-updates

- 注1：VTS测试时，需要同时烧写编译出的boot-debug.img到boot分区；
- 注2：CTS-ON-GSI测试时则不需要烧boot-debug.img；
- 注3：测试时请使用Google官方发布的，带有-signed结尾的GSI镜像；

使用DTBO功能

Android 10.0及以上支持Device Tree Overlays功能，开发过程体现在需要烧写dtbo.img，用于多个产品间的兼容等。
修改方法：

1. 找到(或指定)模板文件：

```
get_build_var PRODUCT_DTBO_TEMPLATE
```

例如：

```
PRODUCT_DTBO_TEMPLATE := $(LOCAL_PATH)/dt-
overlay.in(device/rockchip/rk356x/rk3566_r/dt-overlay.in)
```

2. 添加或修改需要的节点：
   例如：

```
/dts-v1/;
/plugin/;

&chosen {
    bootargs_ext = "androidboot.boot_devices=${_boot_device}";
};

&firmware_android {
    vbmeta {
        status = "disabled";
    };
    fstab {
        status = "disabled";
    };
};

&reboot_mode {
    mode-bootloader = <0x5242C309>;
    mode-charge = <0x5242C30B>;
    mode-fastboot = <0x5242C303>;
    mode-loader = <0x5242C301>;
    mode-normal = <0x5242C300>;
    mode-recovery = <0x5242C303>;
};
```

注: 使用dtbo时一定要确保dts中存在alias，否则无法成功overlay

修改fstab文件

1. 找到(或指定)模板文件：

```
get_build_var PRODUCT_FSTAB_TEMPLATE
```

例如：

```
PRODUCT_FSTAB_TEMPLATE := device/rockchip/rk356x/rk3566_eink/fstab_eink.in
```

2. 修改：添加分区挂载、修改swap_zram参数，修改data分区格式等

修改parameter.txt

Android 11添加了生成parameter.txt的工具，支持根据配置参数编译出parameter.txt。如果没有配置模板文件，则会寻找添加修改好的parameter.txt文件。

1. 找到(或指定)模板文件：

```
get_build_var PRODUCT_PARAMETER_TEMPLATE
```

例如：

```
PRODUCT_PARAMETER_TEMPLATE :=
device/rockchip/common/scripts/parameter_tools/parameter.in
```

2. 修改配置分区大小(例如)：

```
BOARD_SUPER_PARTITION_SIZE := 2688548864
BOARD_DTBOIMG_PARTITION_SIZE := xxxx
BOARD_BOOTIMAGE_PARTITION_SIZE := xxxxx
BOARD_CACHEIMAGE_PARTITION_SIZE := xxxx
```

3. 不使用parameter生成工具：
   添加一个parameter.txt文件到你的device目录下即可：
   例如：device/rockchip/rk3326/rk3326_q/parameter.txt
4. 仅使用工具生成parameter.txt(例如)：

```
parameter_tools --input
device/rockchip/common/scripts/parameter_tools/parameter.in --firmware-version
11.0 --machine-model rk3326 --manufacturer rockchip --machine rk3326_r --
partition-list
uboot_a:4096K,trust_a:4M,misc:4M,dtbo_a:4M,vbmeta_a:4M,boot_a:33554432,backup:30
0M,security:4M,cache:300M,metadata:4096,frp:512K,super:2G --output
parameter_new.txt
```

注: 如果需要大版本OTA升级，请直接使用之前版本的parameter.txt

5. 新加一个分区
   以新建baseparameter分区为例进行说明：

- 在产品的BoardConfig.mk中定义：BOARD_WITH_SPECIAL_PARTITIONS
  like: BOARD_WITH_SPECIAL_PARTITIONS := baseparameter:1M,logo:16M

```
device/rockchip/rk356x/rk3566_r/BoardConfig.mk
+++ b/BoardConfig.mk
@@ -494,4 +494,11 @@ ifeq ($(strip $(BOARD_TWRP_ENABLE)), true)
+BOARD_WITH_SPECIAL_PARTITIONS := baseparameter:1M
```

- 在RebuildParameter.mk中添加BOARD_WITH_SPECIAL_PARTITIONS

```
device/rockchip/common/build/rockchip/RebuildParameter.mk
+ifneq ($(strip $(BOARD_WITH_SPECIAL_PARTITIONS)), )
+partition_list := $(partition_list),$(BOARD_WITH_SPECIAL_PARTITIONS)
+endif
```

Android常用配置

新建产品lunch

以RK356x平台新建rk3568_r产品为例，分以下步骤：
1）修改device/rockchip/rk356x/AndroidProducts.mk增加rk3568_r的lunch

```
--- a/AndroidProducts.mk
+++ b/AndroidProducts.mk
@@ -17,10 +17,14 @@
 PRODUCT_MAKEFILES := \
         $(LOCAL_DIR)/rk3566_rgo/rk3566_rgo.mk \
         $(LOCAL_DIR)/rk3566_r/rk3566_r.mk \
+        $(LOCAL_DIR)/rk3568_r/rk3568_r.mk \

 COMMON_LUNCH_CHOICES := \
     rk3566_rgo-userdebug \
     rk3566_rgo-user \
     rk3566_r-userdebug \
     rk3566_r-user \
+    rk3568_r-userdebug \
+    rk3568_r-user \
```

2）在device/rockchip/rk356x目录下新建rk3568_r目录
参考device/rockchip/rk356x下已有的rk3566_r产品目录新建，可以先直接拷贝rk3566_r为rk3568_r，然后将rk3568_r目录下的所有`rk3566_r`字符改为`rk3568_r`

Kernel dts说明

新建产品dts

产品新建dts可以根据下表的配置选择对应的dts作为参考。

Soc	PMIC	DDR	开发板类型	机型	DTS
RK3566	RK817	LPDDR4	平板	样机	rk3566-rk817-tablet
RK3566	RK809	LPDDR4	开发板	RK3566 EVB2	rk3566-evb2-lp4x-v10
RK3566	分立	DDR4	开发板	RK3566 BOX DEMO	rk3566-box-demo-v10
RK3568	RK809	DDR4	开发板	RK3568 EVB1	rk3566-evb1-ddr4-v10
RK3326	RK817	DDR3	开发板	evb	rk3326-evb-lp3-v10-avb
PX30	RK809	DDR3	开发板	evb	px30-evb-ddr3-v10-avb
RK3326	RK817	DDR3	平板	样机	rk3326-863-lp3-v10-rkisp1
RK3326	RK809	DDR3	人工智能语音	EVB	rk3326-evb-ai-va-v12
RK3399	RK808	LPDDR3	开发板	挖掘机	rk3399-sapphire-excavator-edp-avb
RK3399	RK809	LPDDR4	开发板	IND开发板	rk3399-evb-ind-lpddr4-android-avb
RK3399	RK808	LPDDR4	平板	BQ25703双节电池	rk3399-tve1030g-avb
RK3399	RK818	LPDDR3	平板	edp屏	rk3399-mid-818-android
RK3288W	RK808	LPDDR3	开发板	edp屏	rk3288-evb-android-rk808-edp-avb
RK3328	分立	DDR3	开发板	RK3328 开发板	rk3328-box-liantong-avb

文档说明

外设支持列表

DDR/EMMC/NAND FLASH/WIFI/3G/CAMERA的支持列表实时更新在redmine上，链接如下：

```
https://redmine.rockchip.com.cn/projects/fae/documents
```

Android文档

```
RKDocs\android
```

Android_SELinux(Sepolicy)开发指南

```
RKDocs/android/Rockchip_Developer_Guide_Android_SELinux(Sepolicy)_CN.pdf
```

Wi-Fi文档

```
RKDocs/android/wifi/
├── Rockchip_Introduction_Android10.0_WIFI_Configuration_CN&EN.pdf
└── Rockchip_Introduction_REALTEK_WIFI_Driver_Porting_CN&EN.pdf
```

3G/4G模块说明文档

```
RKDocs/common/mobile-net/
├── Rockchip_Introduction_3G_Data_Card_USB_File_Conversion_CN.pdf
├── Rockchip_Introduction_3G_Dongle_Configuration_CN.pdf
└── Rockchip_Introduction_4G_Module_Configuration_CN&EN.pdf
```

Kernel文档

```
RKDocs\common
```

DDR相关文档

```
RKDocs/common/DDR/
├── Rockchip-Developer-Guide-DDR-CN.pdf
├── Rockchip-Developer-Guide-DDR-EN.pdf
├── Rockchip-Developer-Guide-DDR-Problem-Solution-CN.pdf
├── Rockchip-Developer-Guide-DDR-Problem-Solution-EN.pdf
└── Rockchip-Developer-Guide-DDR-Verification-Process-CN.pdf
```

Audio模块文档

```
RKDocs/common/Audio/
├──
Rockchip_Developer_Guide_Audio_Call_3A_Algorithm_Integration_and_Parameter_Debug
ging_CN.pdf
├── Rockchip_Developer_Guide_Linux4.4_Audio_CN.pdf
└── Rockchip_Developer_Guide_RK817_RK809_Codec_CN.pdf
```

CRU模块文档

```
RKDocs/common/CRU/
├── Rockchip_Developer_Guide_Linux3.10_Clock_CN.pdf
└── Rockchip_RK3399_Developer_Guide_Linux4.4_Clock_CN.pdf
```

GMAC模块文档

```
RKDocs/common/GMAC/
└── Rockchip_Developer_Guide_Ethernet_CN.pdf
```

PCie模块文档

```
RKDocs/common/PCie/
└── Rockchip-Developer-Guide-linux4.4-PCIe.pdf
```

I2C模块文档

```
RKDocs/common/I2C/
└── Rockchip_Developer_Guide_I2C_CN.pdf
```

PIN-Ctrl GPIO模块文档

```
RKDocs/common/PIN-Ctrl/
└── Rockchip-Developer-Guide-Linux-Pin-Ctrl-CN.pdf
```

SPI模块文档

```
RKDocs/common/SPI/
└── Rockchip-Developer-Guide-linux4.4-SPI.pdf
```

Sensor模块文档

```
RKDocs/common/Sensors/
└── Rockchip_Developer_Guide_Sensors_CN.pdf
```

IO-Domain模块文档

```
RKDocs/common/IO-Domain/
└── Rockchip_Developer_Guide_Linux_IO_DOMAIN_CN.pdf
```

Leds模块文档

```
RKDocs/common/Leds/
└── Rockchip_Introduction_Leds_GPIO_Configuration_for_Linux4.4_CN.pdf
```

Thermal温控模块文档

```
RKDocs/common/Thermal/
├── Rockchip-Developer-Guide-Linux4.4-Thermal-CN.pdf
└── Rockchip-Developer-Guide-Linux4.4-Thermal-EN.pdf
```

PMIC电源管理模块文档

```
RKDocs/common/PMIC/
├── Archive.zip
├── Rockchip_Developer_Guide_Power_Discrete_DCDC_EN.pdf
├── Rockchip-Developer-Guide-Power-Discrete-DCDC-Linux4.4.pdf
├── Rockchip-Developer-Guide-RK805.pdf
├── Rockchip_Developer_Guide_RK817_RK809_Fuel_Gauge_CN.pdf
├── Rockchip_RK805_Developer_Guide_CN.pdf
└── Rockchip_RK818_RK816_Introduction_Fuel_Gauge_Log_CN.pdf
```

MCU模块文档

```
RKDocs/common/MCU/
└── Rockchip_Developer_Guide_MCU_EN.pdf
```

功耗与休眠模块文档

```
RKDocs/common/power/
├── Rockchip_Developer_Guide_Power_Analysis_EN.pdf
└── Rockchip_Developer_Guide_Sleep_and_Resume_CN.pdf
```

UART模块文档

```
RKDocs/common/UART/
├── Rockchip-Developer-Guide-linux4.4-UART.pdf
└── Rockchip-Developer-Guide-RT-Thread-UART.pdf
```

DVFS CPU/GPU/DDR变频相关文档

```
RKDocs/common/DVFS/
├── Rockchip_Developer_Guide_CPUFreq_CN.pdf
├── Rockchip_Developer_Guide_CPUFreq_EN.pdf
├── Rockchip_Developer_Guide_Devfreq_CN.pdf
├── Rockchip_Developer_Guide_Linux4.4_CPUFreq_CN.pdf
└── Rockchip_Developer_Guide_Linux4.4_Devfreq_CN.pdf
```

EMMC/SDMMC/SDIO模块文档

```
RKDocs/common/MMC
└── Rockchip-Developer-Guide-linux4.4-SDMMC-SDIO-eMMC.pdf
```

PWM模块文档

```
RKDocs/common/PWM/
├── Rockchip-Developer-Guide-Linux-PWM-CN.pdf
└── Rockchip_Developer_Guide_PWM_IR_CN.pdf
```

USB模块文档

```
RKDocs/common/usb/
├── putty20190213_162833_1.log
├── Rockchip-Developer-Guide-Linux4.4-RK3399-USB-DTS-CN.pdf
├── Rockchip-Developer-Guide-Linux4.4-USB-CN.pdf
├── Rockchip-Developer-Guide-Linux4.4-USB-FFS-Test-Demo-CN.pdf
├── Rockchip-Developer-Guide-Linux4.4-USB-Gadget-UAC-CN.pdf
├── Rockchip-Developer-Guide-USB-Initialization-Log-Analysis-CN.pdf
├── Rockchip-Developer-Guide-USB-Performance-Analysis-CN.pdf
├── Rockchip-Developer-Guide-USB-PHY-CN.pdf
└── Rockchip-Developer-Guide-USB-SQ-Test-CN.pdf
```

HDMI-IN功能文档

```
RKDocs/common/hdmi-in/
└── Rockchip_Developer_Guide_HDMI_IN_CN.pdf
```

安全模块文档

```
RKDocs/common/security/
├── Efuse process explain .pdf
├── RK3399_Efuse_Operation_Instructions_V1.00_20190214_EN.pdf
├── Rockchip_Developer_Guide_Secure_Boot_V1.1_20190603_CN.pdf
├── Rockchip_Developer_Guide_TEE_Secure_SDK_CN.pdf
├── Rockchip_RK3399_Introduction_Efuse_Operation_EN.pdf
├── Rockchip-Secure-Boot2.0.pdf
├── Rockchip-Secure-Boot-Application-Note-V1.9.pdf
└── Rockchip Vendor Storage Application Note.pdf
```

uboot介绍文档

```
RKDocs\common\u-boot\Rockchip-Developer-Guide-UBoot-nextdev-CN.pdf
```

Trust介绍文档

```
RKDocs/common/TRUST/
├── Rockchip_Developer_Guide_Trust_CN.pdf
└── Rockchip_Developer_Guide_Trust_EN.pdf
```

Camera文档

```
RKDocs\common\camera\HAL3\
```

Camera IQ Tool文档

```
external/camera_engine_rkaiq/rkisp2x_tuner/doc/
├── Rockchip_Color_Optimization_Guide_ISP2x_CN_v2.0.0.pdf
├── Rockchip_IQ_Tools_Guide_ISP2x_CN_v2.0.0.pdf
└── Rockchip_Tuning_Guide_ISP21_CN_v2.0.0.pdf
```

工具文档

```
RKDocs\common\RKTools manuals
```

PCBA开发使用文档

```
RKDocs\android\Rockchip_Developer_Guide_PCBA_Test_Tool_CN&EN.pdf
```

显示屏驱动调试指南

```
RKDocs\common\display\Rockchip_Developer_Guide_DRM_Panel_Porting_CN.pdf
```

HDMI调试指南

```
RKDocs\common\display\Rockchip_Developer_Guide_HDMI_Based_on_DRM_Framework_CN.pd
f
```

图像显示DRM Hardware Composer（HWC）问题分析排查

```
RKDocs\common\display\Rockchip FAQ DRM Hardware Composer V1.00-20181213.pdf
```

DRM显示开发指南

```
RKDocs\common\display\Rockchip DRM Display Driver Development Guide V1.0.pdf
```

RGA相关问题分析排查

```
RKDocs\common\display\Rockchip_RGA_FAQ.pdf
```

图形显示框架常见问题分析

包括frameworks、GPU.Gralloc、GUI、HWComposer、HWUI、RGA

```
RKDocs\common\display\Rockchip_Trouble_Shooting_Graphics
```

工具使用

StressTest

设备上使用Stresstest 工具，对待测设备的各项功能进行压力测试，确保整个系统运行的稳定性。SDK通过打开计算器应用，输入“83991906=”暗码，可启动StressTest应用，进行各功能压力测试。
Stresstest 测试工具测试的内容主要包括：

模块相关

- Camera 压力测试：包括Camera 打开关闭，Camera 拍照以及Camera 切换。
- Bluetooth 压力测试：包括Bluetooth 打开关闭。
- Wi-Fi 压力测试： 包括Wi-Fi 打开关闭，（ ping 测试以及iperf 测试待加入）。

非模块相关

- 飞行模式开关测试
- 休眠唤醒拷机测试
- 视频拷机测试
- 重启拷机测试
- 恢复出厂设置拷机测试
- Arm 变频测试
- Gpu 变频测试
- DDR 变频测试

PCBA测试工具

PCBA 测试工具用于帮助在量产的过程中快速地甄别产品功能的好坏，提高生产效率。目前包括屏幕（LCD）、无线（Wi-Fi）、蓝牙（bluetooth）、DDR/EMMC 存储、SD 卡（sdcard）、USB HOST、按键（KEY），喇叭耳机（Codec）测试项目。
这些测试项目包括自动测试项和手动测试项，无线网络、DDR/EMMC、以太网为自动测试项，按键、SD卡、USB HOST、Codec、为手动测试项目。
具体PCBA功能配置及使用说明，请参考：

```
RKDocs\android\Rockchip_Developer_Guide_PCBA_Test_Tool_CN&EN.pdf_V1.1_20171222.pdf。
```

DeviceTest

DeviceTest 用于工厂整机测试，主要测试装成整机以后外围器件是否正常。SDK 通过打开计算器，输入暗码“000.=”进入 DeviceTest，如下所示：

在产线可以根据这个界面进行对应外设的测试，测试时点击“TEST CHECKED”对所测项目逐项进行测试，测试如果成功点击 pass，失败点击 failed，最终结果会显示在界面上，如下图所示，红色为 failed 项，其余为通过项，工厂可根据测试结果进行相应的维修。另外，如果客户需要对该工具进行定制，请联系 FAE 窗口申请对应的源码。

USB驱动

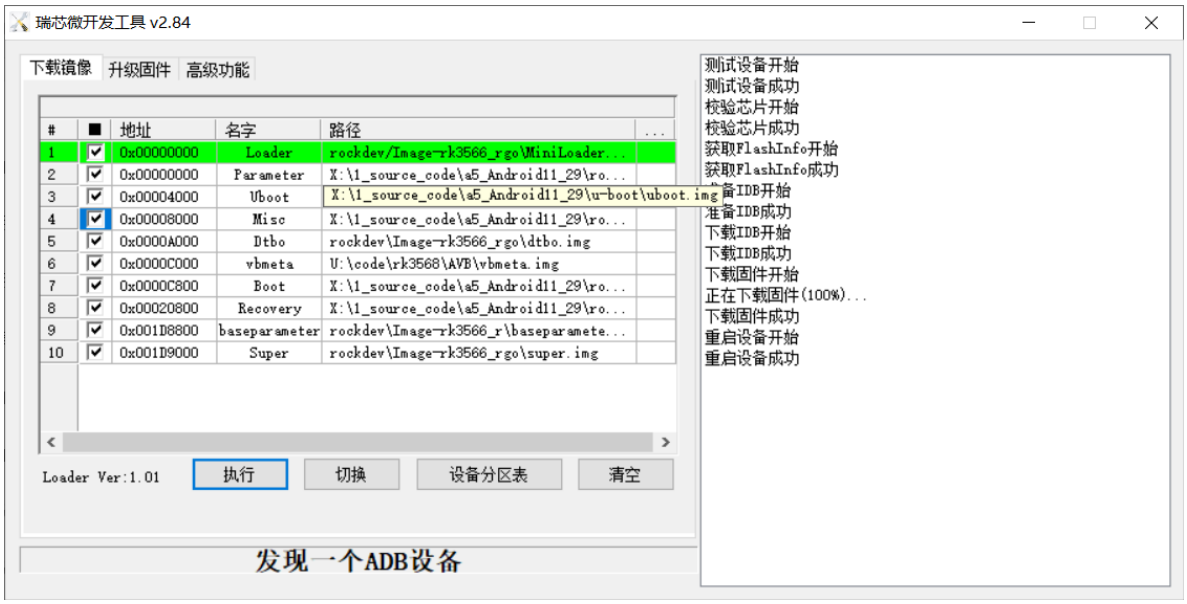
Rockchip USB驱动安装包，包括ADB、固件烧写驱动

```
RKTools\windows\DriverAssitant_v5.1.1.zip
```

开发烧写工具

Windows版本

RKTools/windows/AndroidTool/AndroidTool_Release_v2.84.zip，工具版本会时刻更新，请及时同步更新



Linux版本

RKTools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool_v1.56.zip

```
Linux_Upgrade_Tool_v1.56$ sudo ./upgrade_tool -h
Program Data in /home/wlq/.config/upgrade_tool

----------------------Tool Usage ----------------------
Help:              H
Quit:              Q
Version:           V
Clear Screen:      CS
------------------Upgrade Command ------------------
ChooseDevice:            CD
ListDevice:                  LD
SwitchDevice:            SD
UpgradeFirmware:         UF <Firmware> [-noreset]
UpgradeLoader:           UL <Loader> [-noreset]
DownloadImage:           DI <-p|-b|-k|-s|-r|-m|-u|-t|-re image>
DownloadBoot:            DB <Loader>
EraseFlash:              EF <Loader|firmware> [DirectLBA]
PartitionList:           PL
WriteSN:                 SN <serial number>
ReadSN:               RSN
----------------Professional Command -----------------
TestDevice:              TD
ResetDevice:             RD [subcode]
ResetPipe:               RP [pipe]
ReadCapability:          RCB
ReadFlashID:             RID
ReadFlashInfo:           RFI
ReadChipInfo:            RCI
ReadSector:              RS  <BeginSec> <SectorLen> [-decode] [File]
WriteSector:             WS  <BeginSec> <File>
ReadLBA:                 RL  <BeginSec> <SectorLen> [File]
WriteLBA:                WL  <BeginSec> <File>
EraseLBA:                EL  <BeginSec> <EraseCount>
```

```
EraseBlock:             EB <CS> <BeginBlock> <BlokcLen> [--Force]
-------------------------------------------------------
```

SD升级启动制作工具

用于制作SD卡升级、SD卡启动、SD卡PCBA测试

```
RKTools\windows\SDDiskTool_v1.59.zip
```

写号工具

RKTools\windows\RKDevInfoWriteTool_Setup_V1.0.3.rar
解压RKDevInfoWriteTool_Setup_V1.0.3.rar后安装
以管理员权限打开软件

工具说明请参考：

```
RKDocs\common\RKTools manuals\RKDevInfoWriteTool_User_Guide_V1.0.3.pdf
```

DDR焊接测试工具

用于测试DDR的硬件连接，排查虚焊等硬件问题

```
RKTools\windows\Rockchip_Platform_DDR_Test_Tool_V1.38_Release_Annoucement_CN.7z
RKTools\windows\Rockchip_Platform_DDR_Test_Tool_V1.38_Release_Annoucement_EN.7z
```

efuse烧写工具

用于efuse的烧写，适用于RK3288W/RK3368/RK3399平台

```
RKTools\windows\efuse_v1.37.rar
```

efuse/otp签名工具

用于固件的efuse/otp签名

```
RKTools\windows\SecureBootTool_v1.94.zip
```

工厂生产固件烧写工具

用于工厂批量烧写固件

```
RKTools\windows\FactoryTool_1.66.zip
```

固件修改工具

用于修改update.img固件

```
RKTools\windows\FWFactoryTool_V5.52.rar
```

userdata分区数据预置工具

用于制作userdata分区预置数据包的工具

```
RKTools\windows\OemTool_v1.3.rar
```

Camera IQ Tool

用于调试ISP图像效果

```
external/camera_engine_rkaiq/rkisp2x_tuner
```

系统调试

ADB工具

概述

ADB（Android Debug Bridge）是 Android SDK里的一个工具，用这个工具可以操作管理 Android 模拟器或真实的 Android 设备。主要功能有：

- 运行设备的 shell（命令行）
- 管理模拟器或设备的端口映射
- 计算机和设备之间上传/下载文件
- 将本地 apk软件安装至模拟器或 Android 设备
  ADB 是一个“客户端 - 服务器端”程序，其中客户端主要是指PC，服务器端是Android 设备的实体机器或者虚拟机。根据PC连接Android设备的方式不同，ADB 可以分为两类：
- 网络 ADB：主机通过有线/无线网络（同一局域网）连接到STB设备
- USB ADB：主机通过 USB 线连接到STB设备

USB adb使用说明

USB adb 使用有以下限制：

- 只支持 USB OTG 口
- 不支持多个客户端同时使用（如 cmd 窗口，eclipse等）
- 只支持主机连接一个设备，不支持连接多个设备
  连接步骤如下：
  1、Android设备已经运行 Android 系统，设置->开发者选项->已连接到计算机 打开，usb调试开关打开。
  2、PC主机只通过 USB 线连接到机器 USB otg 口，然后电脑通过如下命令与Android设备相连。

```
adb shell
```

3、测试是否连接成功，运行”adb devices”命令，如果显示机器的序列号，表示连接成功。

网络adb使用要求

adb早期版本只能通过USB来对设备调试，从adb v1.0.25开始，增加了对通过tcp/ip调试Android设备的功能。
如果你需要使用网络adb来调试设备，必须要满足如下条件：
1、设备上面首先要有网口，或者通过WiFi连接网络。
2、设备和研发机（PC机）已经接入局域网，并且设备设有局域网的IP地址。
3、要确保研发机和设备能够相互ping得通。
4、研发机已经安装了adb。
5、确保Android设备中adbd进程（adb的后台进程）已经运行。adbd进程将会监听端口5555来进行adb连接调试。

SDK网络adb端口配置

SDK默认未开启网络adb，需要手动在开发者选项中打开。
Setting-System-Advanced-Developer options-Open net adb

网络adb使用

本节假设设备的ip为192.168.1.5，下文将会用这个ip建立adb连接，并调试设备。
1、首先Android设备需要先启动，如果可以的话，可以确认adbd是否启动(ps命令查看)。
2、在PC机的cmd中，输入：

```
adb connect 192.168.1.5:5555
```

如果连接成功会进行相关的提示，如果失败的话，可以先kill-server命令，然后重试连接。

```
adb kill-server
```

3、如果连接已经建立，在研发机中，可以输入adb相关的命令进行调试了。比如adb shell，将会通过tcp/ip连接设备上面。和USB调试是一样的。
4、调试完成之后，在研发机上面输入如下的命令断开连接：

```
adb disconnect 192.168.1.5:5555
```

手动修改网络adb端口号

若SDK未加入adb端口号配置，或是想修改adb端口号，可通过如下方式修改：
1、首先还是正常地通过USB连接目标机，在windows cmd下执行adb shell进入。
2、设置adb监听端口：

```
#setprop service.adb.tcp.port 5555
```

3、通过ps命令查找adbd的pid
4、重启adbd
#kill -9，这个pid就是上一步找到那个pid
杀死adbd之后，android的init进程后自动重启adbd。adbd重启后，发现设置了service.adb.tcp.port，就会自动改为监听网络请求。

ADB常用命令详解

（1）查看设备情况
查看连接到计算机的 Android 设备或者模拟器：

```
adb devices
```

返回的结果为连接至开发机的 Android 设备的序列号或是IP和端口号（Port）、状态。
（2）安装apk
将指定的 apk 文件安装到设备上：

```
adb install <apk文件路径>
```

示例如下：

```
adb install “F:\WishTV\WishTV.apk”
```

重新安装应用：

```
adb install –r “F:\WishTV\WishTV.apk”
```

（3）卸载apk
完全卸载：

```
adb uninstall <package>
```

示例如下：

```
adb uninstall com.wishtv
```

（4）使用 rm移除 apk 文件：

```
adb shell rm <filepath>
```

示例如下：

```
adb shell rm “system/app/WishTV.apk”
```

示例说明：移除“system/app”目录下的“WishTV.apk”文件。
（5）进入设备和模拟器的shell
进入设备或模拟器的 shell 环境：

```
adb shell
```

（6）从电脑上传文件到设备
用 push 命令可以把本机电脑上的任意文件或者文件夹上传到设备。本地路径一般指本机电脑；远程路径一般指 adb 连接的单板设备。
adb push <本地路径> <远程路径>
示例如下：

```
adb push “F:\WishTV\WishTV.apk” “system/app”
```

示例说明：将本地“WishTV.apk”文件上传到 Android 系统的“system/app”目录下。
（7）从设备下载文件到电脑
pull 命令可以把设备上的文件或者文件夹下载到本机电脑中。

```
adb pull <远程路径> <本地路径>
```

示例如下：

```
adb pull system/app/Contacts.apk F:\
```

示例说明：将 Android 系统“system/app”目录下的文件或文件夹下载到本地“F:\”目录下。
（8）查看 bug报告
需要查看系统生成的所有错误消息报告，可以运行 adb bugreport指令来实现，该指令会将 Android 系统的dumpsys、dumpstate 与 logcat 信息都显示出来。
（9）查看设备的系统信息
在 adb shell下查看设备系统信息的具体命令。

```
adb shell getprop
```

Logcat工具

Android 日志系统提供了记录和查看系统调试信息的功能。日志都是从各种软件和一些系统的缓冲区中记录下来的，缓冲区可以通过 Logcat 来查看和使用。Logcat 是调试程序用的最多的功能。该功能主要是通过打印日志来显示程序的运行情况。由于要打印的日志量非常大，需要对其进行过滤等操作。

Logcat命令使用

用 logcat 命令来查看系统日志缓冲区的内容：
基本格式：

```
[adb] logcat [<option>] [<filter-spec>]
```

示例如下：

```
adb shell
logcat
```

常用的日志过滤方式

控制日志输出的几种方式：

- 控制日志输出优先级
  示例如下：

```
adb shell
logcat *:W
```

示例说明：显示优先级为 warning 或更高的日志信息。

- 控制日志标签和输出优先级
  示例如下：

```
adb shell
logcat ActivityManager:I MyApp:D *:S
```

示例说明：支持所有的日志信息，除了那些标签为“ActivityManager”和优先级为“Info”以上的、标签为“MyApp”和优先级为“Debug”以上的。

- 只输出特定标签的日志
  示例如下：

```
adb shell
logcat WishTV:* *:S
```

或者

```
adb shell
logcat –s WishTV
```

示例说明：只输出标签为 WishTV的日志。

- 只输出指定优先级和标签的日志
  示例如下：

```
adb shell
logcat WishTV:I *:S
```

示例说明：只输出优先级为 I，标签为 WishTV的日志。

Procrank工具

Procrank 是 Android 自带的一款调试工具，运行在设备侧的 shell 环境下，用来输出进程的内存快照，便于有效的观察进程的内存占用情况。
包括如下内存信息：

- VSS：Virtual Set Size 虚拟耗用内存大小（包含共享库占用的内存）
- RSS：Resident Set Size 实际使用物理内存大小（包含共享库占用的内存）
- PSS：Proportional Set Size 实际使用的物理内存大小（比例分配共享库占用的内存）
- USS：Unique Set Size 进程独自占用的物理内存大小（不包含共享库占用的内存）
  注意：
- USS 大小代表只属于本进程正在使用的内存大小，进程被杀死后会被完整回收；
- VSS/RSS 包含了共享库使用的内存，对查看单一进程内存状态没有参考价值；
- PSS 是按照比例将共享内存分割后，某单一进程对共享内存区的占用情况。

使用procrank

执行procrank前需要先让终端获取到root权限
su
命令格式：

```
procrank [ -w ] [ -v | -r | -p | -u | -h ]
```

常用指令说明：
-v：按照 VSS 排序
-r：按照 RSS 排序
-p：按照 PSS 排序
-u：按照 USS 排序
-R：转换为递增[递减]方式排序
-w：只显示 working set 的统计计数
-W：重置 working set 的统计计数
-h：帮助
示例：
输出内存快照：

```
procrank
```

按照 VSS 降序排列输出内存快照：

```
procrank -v
```

默认procrank输出是通过PSS排序。

检索指定内容信息

查看指定进程的内存占用状态，命令格式如下：

```
procrank | grep [cmdline | PID]
```

其中 cmdline 表示需要查找的应用程序名，PID 表示需要查找的应用进程。
输出 systemUI 进程的内存占用状态：

```
procrank | grep “com.android.systemui”
```

或者：

```
procrank  | grep 3396
```

跟踪进程内存状态

通过跟踪内存的占用状态，进而分析进程中是否存在内存泄露场景。使用编写脚本的方式，连续输出进程的内存快照，通过对比 USS 段，可以了解到此进程是否有内存泄露。
示例：输出进程名为 com.android.systemui 的应用内存占用状态，查看是否有泄露：
1、编写脚本 test.sh

```
#!/bin/bash
while true;do
adb shell procrank | grep “com.android.systemui”
sleep 1
done
```

2、通过 adb 工具连接到设备后，运行此脚本：./test.sh

Dumpsys工具

Dumpsys 工具是 Android系统中自带的一款调试工具，运行在设备侧的 shell 环境下，提供系统中正在运行的服务状态信息功能。正在运行的服务是指 Android binder机制中的服务端进程。
dumpsys 输出打印的条件：
1、只能打印已经加载到 ServiceManager中的服务；
2、如果服务端代码中的 dump 函数没有被实现，则没有信息输出。

使用Dumpsys

- 查看Dumpsys帮助
  作用：输出dumpsys帮助信息。

```
  dumpsys  -help
```

- 查看Dumpsys包含服务列表
  作用：输出dumpsys所有可打印服务信息，开发者可以关注需要调试服务的名称。

```
  dumpsys  -l
```

- 输出指定服务的信息
  作用：输出指定的服务的 dump 信息。
  格式：dumpsys [servicename]
  示例：输出服务 SurfaceFlinger的信息，可执行命令：

```
dumpsys SurfaceFlinger
```

- 输出指定服务和应有进程的信息
  作用：输出指定服务指定应用进程信息。
  格式：dumpsys [servicename] [应用名]
  示例：输出服务名为 meminfo，进程名为 com.android.systemui 的内存信息，执行命令：

```
dumpsys meminfo com.android.systemui
```

注意：服务名称是大小写敏感的，并且必须输入完整服务名称。

Last log 开启

- 在dts文件里面添加下面两个节点

```
ramoops_mem: ramoops_mem {
    reg = <0x0 0x110000 0x0 0xf0000>;
    reg-names = "ramoops_mem";
};

ramoops {
    compatible = "ramoops";
    record-size = <0x0 0x20000>;
    console-size = <0x0 0x80000>;
    ftrace-size = <0x0 0x00000>;
    pmsg-size = <0x0 0x50000>;
    memory-region = <&ramoops_mem>;
};
```

```
- 在机器中查看last log
130|root@rk3399:/sys/fs/pstore # ls
dmesg-ramoops-0    上次内核panic后保存的log。
pmsg-ramoops-0     上次用户空间的log，android的log。
ftrace-ramoops-0     打印某个时间段内的function trace。
console-ramoops-0  last_log 上次启动的kernel log，但只保存了优先级比默认log level 高的
log。
```

- 使用方法：

```
cat dmesg-ramoops-0
cat console-ramoops-0
logcat -L （pmsg-ramoops-0） 通过logcat 取出来并解析pull out by logcat and parse
cat ftrace-ramoops-0
```

FIQ模式

当设备死机或者卡住的时候可以在串口输入fiq命令查看系统的状态，具体命令如下：

```
127|console:/ $ fiq
debug>  help
FIQ Debugger commands:
 pc            PC status
 regs          Register dump
 allregs       Extended Register dump
 bt            Stack trace
 reboot [<c>]  Reboot with command <c>
 reset [<c>]   Hard reset with command <c>
 irqs          Interupt status
 kmsg          Kernel log
 version       Kernel version
```

```
sleep          Allow sleep while in FIQ
nosleep        Disable sleep while in FIQ
console        Switch terminal to console
cpu            Current CPU
cpu <number>   Switch to CPU<number>
ps             Process list
sysrq          sysrq options
sysrq <param>  Execute sysrq with <param>
```

log自动收集

- 收集的内容

  ```
  android：android log
  kernel ：kernel log
  ```

- 打开方式
- 开启Developer options
- Setting-System-Advanced-Developer options-Android bug collector
- log保存路径

  ```
  data/vendor/logs/
  ```

常见问题

当前kernel和u-boot版本?

```
Android11.0 对应的kernel版本为：develop-4.19，u-boot的分支为next-dev分支
```

如何获取当前SDK对应的RK release版本

Rockchip Android11.0 SDK包括AOSP原始代码和RK修改的代码两部分，其中RK修改的仓库包含在`.repo/manifests/include`目录下面的xml中，AOSP默认的仓库在`.repo/manifests/default.xml`。
版本确认：

- RK修改部分

```
vim .repo/manifests/include/rk_checkout_from_aosp.xml
<project groups="pdk" name="platform/build" path="build/make" remote="rk"
revision="refs/tags/android-11.0-mid-rkr1">
```

说明RK的版本是android-11.0-mid-rkr1

- AOSP部分

```
vim .repo/manifests/default.xml
<default revision="refs/tags/android-10.0.0_r14"...>
```

说明AOSP的版本是android-10.0.0_r14
当需要提供版本信息的时候提供以上两个版本信息即可。
单个仓库可以直接通过如下命令获取tag信息

```
kernel$ git tag
android-11.0-mid-rkr1
develop-4.4-20190201
```

RK的版本是以android-11.0-mid-rkrxx的格式递增的，所以当前的最新tag是android-11.0-mid-rkr1

如何确认本地SDK已经完整更新RK发布的SDK状态

RK发布SDK版本时会在.repo/manifests/commit/目录下对应提交该版本的commit信息，客户可以通过对比这个commit信息来确认是否有完整更新SDK，具体操作如下：

- 按"如何获取当前SDK对应的RK release版本"的说明先确认SDK的RK版本，下面以RK版本是RKR6为例进行说明；
- 用如下命令保存本地的commit信息

```
.repo/repo/repo manifest -r -o release_manifest_rkr6_local.xml
```

- 通过比较.repo/manifests/commit/commit_release_rkr6.xml和release_manifest_rkr6_local.xml，即可确认SDK代码是否更新完整，
  其中.repo/manifests/commit/commit_release_rkr6.xml为RK版本RKR6发布的commit信息。

uboot和kernel阶段logo图片替换

uboot和kernel阶段的logo分别为开机显示的第一张和第二张logo图片，可以根据产品需求进行修改替换。
uboot logo源文件：`kernel/logo.bmp`
kernel logo源文件：`kernel/logo_kernel.bmp`
如果需要更换某一张，只需用同名的bmp替换掉，重新编译内核即可，编译后的文件在boot.img中。
说明：Logo图片大小目前只支持到8M以内大小的bmp格式图片，支持8、16、24、32位的bmp。

RK3566/RK3568 NAND FLASH配置

RK3566和RK3568的NAND和EMMC的固件无法兼容兼容，现在代码默认编译的是EMMC，如果需要使用NAND FLASH则需要修改编译编译方式，差异在UBOOT的编译

- RK3566

```
./make.sh rk3566-nand
```

- RK3568

```
./make.sh rk3568-nand
```

可以修改默认编译的配置，以RK3566为例：

```
cd device/rockchip/rk356x
diff --git a/rk3566_rgo/BoardConfig.mk b/rk3566_rgo/BoardConfig.mk
index be9020a..306a72d 100644
--- a/rk3566_rgo/BoardConfig.mk
+++ b/rk3566_rgo/BoardConfig.mk
@@ -15,7 +15,7 @@
 #
 include device/rockchip/rk356x/BoardConfig.mk
 BUILD_WITH_GO_OPT := true
-PRODUCT_UBOOT_CONFIG := rk3566
+PRODUCT_UBOOT_CONFIG := rk3566-nand
 PRODUCT_KERNEL_DTS := rk3566-rk817-tablet
 CAMERA_SUPPORT_AUTOFOCUS := true
```

关机充电和低电预充

关机充电和低电预充可以在dts中配置，具体如下：

```
charge-animation {
                compatible = "rockchip,uboot-charge";
                rockchip,uboot-charge-on = <1>;
                rockchip,android-charge-on = <0>;
                rockchip,uboot-low-power-voltage = <3400>;
                rockchip,screen-on-voltage = <3500>;
                status = "okay";
        };
```

其中：
rockchip,uboot-charge-on： uboot关机充电，与android关机充电互斥
rockchip,android-charge-on： android关机充电，与uboot关机充电互斥
rockchip,uboot-low-power-voltage： 配置低电预充到开机的电压，可以根据实际需求进行配置
rockchip,screen-on-voltage：配置低电预充到亮屏的电压，可以根据实际需求进行配置

Box 机器待机和假关机功能

```
BOX 最新的版本已经默认支持低功耗真待机功能，短按power键可验证功能；

如果需要假关机(唤醒重启)功能，要注意的是此功能必须要 PD_PMU 的pwm才可以支持，例如：
&pwm3 {
     status = "okay";

     compatible = "rockchip,remotectl-pwm";
     remote_pwm_id = <3>;
     handle_cpu_id = <1>;
     remote_support_psci = <0>;
     pinctrl-names = "default";
     pinctrl-0 = <&pwm3_pins>;
     ................
}

diff --git a/arch/arm64/boot/dts/rockchip/rk3566-box.dtsi
b/arch/arm64/boot/dts/rockchip/rk3566-box.dtsi
index 7818060..58ffe6b 100644
--- a/arch/arm64/boot/dts/rockchip/rk3566-box.dtsi
```

```
+++ b/arch/arm64/boot/dts/rockchip/rk3566-box.dtsi
@@ -309,6 +309,7 @@
                | RKPM_SLP_32K_PVTM
                )
        >;
+    rockchip,virtual-poweroff = <1>;
        rockchip,wakeup-config = <
                (0
                | RKPM_PWM0_WKUP_EN
                ..............
                >
```

Uboot阶段充电图片打包和替换

充电图片路径，可以直接替换同名文件，格式要求与原文件一样。

```
u-boot/tools/images/
├── battery_0.bmp
├── battery_1.bmp
├── battery_2.bmp
├── battery_3.bmp
├── battery_4.bmp
├── battery_5.bmp
└── battery_fail.bmp
```

如果打开uboot充电，但是没有显示充电图片，可能是图片没有打包到resource.img中，可以按如下命令打包

```
cd u-boot
./scripts/pack_resource.sh ../kernel/resource.img
cp resource.img ../kernel/resource.img
```

执行以上命令后uboot充电图片会打包到kernel目录的resource.img中，此时需要再将resource.img打包到boot.img中，可以在android根目录执行./mkiamge.sh，然后烧写rockdev/下面的boot.img即可。

RM310 4G配置

4G功能SDK默认是关闭的，如需打开，按以下操作：

```
vim device/rockchip/common/BoardConfig.mk
#for rk 4g modem
-BOARD_HAS_RK_4G_MODEM ?= false
+BOARD_HAS_RK_4G_MODEM ?= true
```

WIFI休眠策略配置

wifi默认休眠策略是休眠一直保持连接，如需休眠断开，按以下操作：

```
---
a/rk3566_rgo/overlay/frameworks/base/packages/SettingsProvider/res/values/defaul
ts.xml
+++
b/rk3566_rgo/overlay/frameworks/base/packages/SettingsProvider/res/values/defaul
ts.xml
@@ -24,5 +24,5 @@
        You can configure persist.wifi.sleep.delay.ms to delay closing wifi.
        The default is 15 minutes, 0 means that the wifi is turned off
        immediately after the screen is off. -->
-    <integer name="def_wifi_sleep_policy">2</integer>
+    <integer name="def_wifi_sleep_policy">0</integer>
 </resources>
```

Recovery旋转配置

支持Recovery旋转0/90/180/270度，默认不旋转（即旋转0度），旋转配置说明如下：

```
vim device/rockchip/common/BoardConfig.mk
#0:   ROTATION_NONE 旋转0度
#90:  ROTATION_RIGHT 旋转90度
#180: ROTATION_DOWN 旋转180度
#270: ROTATION_LEFT 旋转270度
# For Recovery Rotation
TARGET_RECOVERY_DEFAULT_ROTATION ?= ROTATION_NONE
```

Android Surface旋转

Android系统显示旋转，可以修改如下配置，配置参数为0/90/180/270

```
# For Surface Flinger Rotation
SF_PRIMARY_DISPLAY_ORIENTATION ?= 0
```

替换 AOSP 部分源代码的 remote

客户下载RK的release代码时速度较慢，可以将AOSP的remote修改为国内镜像源，国外的客户可以修改为Google的镜像源。这样可以提高下载速度。具体操作方法如下：
执行repo init（或者解压base包）后，修改.repo/manifests/remote.xml，把其中的 AOSP 这个 remote 的 fetch 从

```
< remote name="aosp" fetch="./" review="https://10.10.10.29" />
```

改为
国内客户：（国内以清华大学镜像源为例，可以根据需要修改为其他国内镜像源）

```
< remote name="aosp" fetch="https://aosp.tuna.tsinghua.edu.cn" />;
```

国外的客户：（Google镜像源）

```
< remote name="aosp" fetch="https://android.googlesource.com" />
```

userdata区文件系统换为EXT4

默认data分区的文件系统为f2fs，建议不带电池的产品可以将data区的文件系统改为ext4，可以减小异常掉电后数据丢失的概率。修改方法如下：
以rk3566_r为例说明：

```
device/rockchip/common$ git diff
diff --git a/scripts/fstab_tools/fstab.in b/scripts/fstab_tools/fstab.in
index 6e78b00..a658332 100755
--- a/scripts/fstab_tools/fstab.in
+++ b/scripts/fstab_tools/fstab.in
@@ -20,6 +20,6 @@ ${_block_prefix}system_ext /system_ext  ext4 ro,barrier=1
${_flags},first_stage_
 # For sdmmc
 /devices/platform/${_sdmmc_device}/mmc_host*        auto  auto    defaults
   voldmanaged=sdcard1:auto
 #  Full disk encryption has less effect on rk3326, so default to enable this.
-/dev/block/by-name/userdata /data f2fs
noatime,nosuid,nodev,discard,reserve_root=32768,resgid=1065
latemount,wait,check,fileencryption=aes-256-xts:aes-256-
cts:v2+inlinecrypt_optimized,quota,formattable,reservedsize=128M,checkpoint=fs
+#/dev/block/by-name/userdata /data f2fs
noatime,nosuid,nodev,discard,reserve_root=32768,resgid=1065
latemount,wait,check,fileencryption=aes-256-xts:aes-256-
cts:v2+inlinecrypt_optimized,quota,formattable,reservedsize=128M,checkpoint=fs
 # for ext4
-#/dev/block/by-name/userdata    /data      ext4
discard,noatime,nosuid,nodev,noauto_da_alloc,data=ordered,user_xattr,barrier=1
latemount,wait,formattable,check,fileencryption=software,quota,reservedsize=128M
,checkpoint=block
+/dev/block/by-name/userdata    /data      ext4
discard,noatime,nosuid,nodev,noauto_da_alloc,data=ordered,user_xattr,barrier=1
latemount,wait,formattable,check,fileencryption=software,quota,reservedsize=128M
,checkpoint=block
```

```
device/rockchip/rk356x$ git diff
diff --git a/rk3566_r/recovery.fstab b/rk3566_r/recovery.fstab
index 7532217..cf789ac 100755
--- a/rk3566_r/recovery.fstab
+++ b/rk3566_r/recovery.fstab
@@ -7,7 +7,7 @@
 /dev/block/by-name/odm                  /odm                ext4
defaults                  defaults
 /dev/block/by-name/cache                /cache              ext4
defaults                  defaults
 /dev/block/by-name/metadata             /metadata           ext4
defaults                  defaults
-/dev/block/by-name/userdata             /data               f2fs
defaults                  defaults
+/dev/block/by-name/userdata             /data               ext4
defaults                  defaults
 /dev/block/by-name/cust                 /cust               ext4
defaults                  defaults
 /dev/block/by-name/custom               /custom             ext4
defaults                  defaults
```

```
 /dev/block/by-name/radical_update        /radical_update      ext4
defaults                   defaults
```

修改开关机动画和开关机铃声

参考文档：

```
RKDocs\android\Rockchip_Introduction_Android_Power_On_Off_Animation_and_Tone_Customization_CN&EN.pdf
```

APP设置性能模式

device/rockchip/rk3xxx/下配置文件：package_performance.xml，在其中的节点中加入需要使用性能模式的包名：（使用 aapt dump badging (file_path.apk)获取包名）

```
< app package="包名" mode="是否启用加速，启用为 1，关闭为 0"/>
```

例如针对安兔兔的参考如下：

```
< app package="com.antutu.ABenchMark"mode="1"/>
< app package="com.antutu.benchmark.full"mode="1"\/>
< app package="com.antutu.benchmark.full"mode="1"\/>
```

编译时会将文件打包进固件。

GPU相关问题排查方法

参考下面文档，可以做初步的问题排查

```
RKDocs\android\Rockchip_User_Guide_Dr.G_CN&EN.pdf
```

OTP和efuse说明

OTP支持芯片

- RK3326
- PX30
- RK3566
- RK3568
  EFUSE支持芯片
- RK3288
- RK3368
- RK3399
  固件签名和otp/efuse烧写参考文档

```
RKDocs\common\security\Rockchip-Secure-Boot-Application-Note-V1.9.pdf
```

代码中如何判断设备的OTP/EFUSE是否已经烧写

OTP/EFUSE的状态会通过kernel的cmdline进行传递，cmdline中的fuse.programmed用来标识OTP/EFUSE状态，具体如下：

- "fuse.programmed=1"：软件固件包已经进行了secure-boot签名，硬件设备的efuse/otp已经被烧写。
- "fuse.programmed=0"：软件固件包已经进行了secure-boot签名，硬件设备的efuse/otp没有被烧写。
- cmdline中没有fuse.programmed：软件固件包没有进行secure-boot签名（Miniloader不传递），或者Miniloader太旧没有支持传递。

开关selinux

如下修改，false为关闭，true为打开

```
device/rockchip/common$
--- a/BoardConfig.mk
+++ b/BoardConfig.mk
@@ -67,7 +67,7 @@ endif

 # Enable android verified boot 2.0
 BOARD_AVB_ENABLE ?= false
-BOARD_SELINUX_ENFORCING ?= false
+BOARD_SELINUX_ENFORCING ?= true
```

开机弹出"Android系统出现问题"警告

出现警告框的原因有两种：

1. 固件不匹配，system/boot/vendor三个fingerprint不一致，不是同一套固件。
2. 机器打开支持了IO调试功能的config，编译时，使用文档前面所说的内核编译命令即可关闭。
3. 对于需要使用IO调试功能的项目，可以直接不管上述两种原因，直接合入frameworks/base下的patch去掉弹窗：

```
diff --git
a/services/core/java/com/android/server/wm/ActivityTaskManagerService.java
b/services/core/java/com/android/server/wm/ActivityTaskManagerService.java
index 595c340..d4e495a 100644
--- a/services/core/java/com/android/server/wm/ActivityTaskManagerService.java
+++ b/services/core/java/com/android/server/wm/ActivityTaskManagerService.java
@@ -6555,7 +6555,7 @@ public class ActivityTaskManagerService extends
IActivityTaskManager.Stub {
                 } catch (RemoteException e) {
                 }

-                if (!Build.isBuildConsistent()) {
+                if (0 && !Build.isBuildConsistent()) {
                     Slog.e(TAG, "Build fingerprint is not consistent, warning
user");
                     mUiHandler.post(() -> {
                         if (mShowDialogs) {
```

如何打开设置中以太网的设置项

系统设置中默认没有以太网设置的选项，如果项目中需要以太网可以按如下配置打开：

```
--- a/BoardConfig.mk
+++ b/BoardConfig.mk
@@ -146,3 +146,6 @@ endif
 ifeq ($(strip $(BOARD_USES_AB_IMAGE)), true)
 DEVICE_MANIFEST_FILE :=
device/rockchip/$(TARGET_BOARD_PLATFORM)/manifest_ab.xml
 endif

+# for ethernet
+BOARD_HS_ETHERNET := true
```

关于AVB和security boot的操作

AVB和security boot的操作参考文档

```
RKDocs/common/security/RK356X_SecurityBoot_And_AVB_instructions_CN.pdf
```

IO命令无法使用

IO命令需要依赖DEVMEM，而DEVMEM默认是关闭的，所以导致IO默认无法使用，如果调试需要使用IO命令可以按如下修改：

```
wlq@ubuntu:~/1_source_code/a3_rk3399_android10.0_29/kernel$ vim
kernel/configs/android-11.config
```

如果是GO的产品则需要修改：

```
wlq@ubuntu:~/1_source_code/a3_rk3399_android10.0_29/kernel$ vim
kernel/configs/android-11-go.config
```

删除掉下面这行：

```
# CONFIG_DEVMEM is not set
```

```
wlq@ubuntu:~/1_source_code/a3_rk3399_android10.0_29/kernel$ vim
configs/r/android-4.19/android-base.config
```

删掉下面这行：

```
# CONFIG_DEVMEM is not set
```

以上两个需要同时删掉

SN号的命令规则

SN号必须以字母开头，长度14个字节以内。

RK3288编译报LZ4的错误

RK3288编译kernel的时候报如下错误：

```
  SORTEX  vmlinux
  SYSMAP  System.map
  OBJCOPY arch/arm/boot/Image
  Kernel: arch/arm/boot/Image is ready
  SHIPPED arch/arm/boot/compressed/hyp-stub.S
  SHIPPED arch/arm/boot/compressed/fdt_rw.c
  SHIPPED arch/arm/boot/compressed/fdt.h
  SHIPPED arch/arm/boot/compressed/libfdt.h
  SHIPPED arch/arm/boot/compressed/libfdt_internal.h
  SHIPPED arch/arm/boot/compressed/fdt_ro.c
  SHIPPED arch/arm/boot/compressed/fdt_wip.c
  SHIPPED arch/arm/boot/compressed/fdt.c
  SHIPPED arch/arm/boot/compressed/lib1funcs.S
  SHIPPED arch/arm/boot/compressed/ashldi3.S
  SHIPPED arch/arm/boot/compressed/bswapsdi2.S
  LDS     arch/arm/boot/compressed/vmlinux.lds
  AS      arch/arm/boot/compressed/head.o
  LZ4     arch/arm/boot/compressed/piggy_data
Incorrect parameters
Usage :
      lz4 [arg] [input] [output]

input   : a filename
          with no FILE, or when FILE is - or stdin, read standard input
Arguments :
 -1     : Fast compression (default)
 -9     : High compression
 -d     : decompression (default for .lz4 extension)
 -z     : force compression
 -f     : overwrite output without prompting
 -h/-H  : display help/long help and exit
arch/arm/boot/compressed/Makefile:191: recipe for target 'arch/arm/boot/compressed/piggy_data' failed
make[2]: *** [arch/arm/boot/compressed/piggy_data] Error 1
arch/arm/boot/Makefile:71: recipe for target 'arch/arm/boot/compressed/vmlinux' failed
make[1]: *** [arch/arm/boot/compressed/vmlinux] Error 2
arch/arm/Makefile:351: recipe for target 'zImage' failed
make: *** [zImage] Error 2
```

问题原因：
系统自带的lz4版本太低，要求1.8.3及以上版本

```
wlq@ubuntu:~$ lz4 -v
*** LZ4 command line interface 64-bits v1.8.3, by Yann Collet ***
refusing to read from a console
```

解决方法：
直接拷贝android编译出来的lz4覆盖系统的lz4

```
sudo cp out/host/linux-x86/bin/lz4 /usr/bin/lz4
```

RKR7以前的版本更新到RKR7及以上版本后第一次烧写（或者ota升级）后无法开机、带电池的机器reboot loader等命令无法使用等问题

问题原因：
RKR7以前版本（小于或等于RKR6）存在概率性无法关机问题，RKR7版本kernel下面有如下提交解决该问题：

```
commit 65a3eeea668ca570009caa1423f9780d44efebbf
Author: Wu Liangqing <wlq@rock-chips.com>
Date:   Fri Apr 23 17:11:25 2021 +0800

    arm64: dts: rockchip: add not-save-power-en to rk356x board

    fix reboot block as follows log:
    [   15.874382] binder: release 247:268 transaction 4234 in, still active
    [   15.874418] binder: send failed reply for transaction 4234 to 395:455
    [   15.959849] binder: undelivered TRANSACTION_ERROR: 29189
    [   16.085993] binder: 147:147 transaction failed 29189/-22, size 100-0 line
3059
    [   16.128154] android_work: sent uevent USB_STATE=DISCONNECTED
```

```
    [   16.145570] logd.klogd: 24 output lines suppressed due to ratelimiting
    [   16.690141] cpu cpu0: min=816000, max=816000
    [   16.696558] rk808 0-0020: reboot: force RK817_RST_FUNC_REG ok!
    [   33.769778] vcc5v0_otg: disabling
    [   33.770099] vcc3v3_lcd0_n: disabling
    [   33.770424] vcc3v3_lcd1_n: disabling
    [   37.699768] rcu: INFO: rcu_preempt detected stalls on CPUs/tasks:
    [   37.700342] rcu:     3-...0: (0 ticks this GP)
idle=b52/1/0x4000000000000000 softirq=4194/4194 fqs=2012
    [   37.701150] rcu:     (detected by 0, t=6302 jiffies, g=3301, q=16)
    [   37.701684] Task dump for CPU 3:
    [   37.701981] init            R  running task        0     1      0
0x0400000a
    [   37.702609] Call trace:
    [   37.702851]  __switch_to+0xe4/0x138
    [   37.703166]  lock_timer_base+0x5c/0xa0
    [   37.703502]  try_to_del_timer_sync+0x30/0x98
    [   37.703883]  del_timer_sync+0x50/0x60
    [   37.704220]  schedule_timeout+0x19c/0x478
    [   37.704582]  clk_gate_endisable+0x2c/0xc8
    [  100.716421] rcu: INFO: rcu_preempt detected stalls on CPUs/tasks:
    [  100.716991] rcu:     3-...0: (0 ticks this GP)
idle=b52/1/0x4000000000000000 softirq=4194/4194 fqs=7921
    [  100.717799] rcu:     (detected by 0, t=25207 jiffies, g=3301, q=19)
    [  100.718334] Task dump for CPU 3:
    [  100.718632] init            R  running task        0     1      0
0x0400000a
    [  100.719260] Call trace:
    [  100.719500]  __switch_to+0xe4/0x138
    [  100.719816]  lock_timer_base+0x5c/0xa0
    [  100.720152]  try_to_del_timer_sync+0x30/0x98
    [  100.720533]  del_timer_sync+0x50/0x60
    [  100.720870]  schedule_timeout+0x19c/0x478
    [  100.721231]  clk_gate_endisable+0x2c/0xc8
    [  163.733075] rcu: INFO: rcu_preempt detected stalls on CPUs/tasks:
    [  163.733643] rcu:     3-...0: (0 ticks this GP)
idle=b52/1/0x4000000000000000 softirq=4194/4194 fqs=13833
    [  163.734462] rcu:     (detected by 0, t=44112 jiffies, g=3301, q=20)
    [  163.734997] Task dump for CPU 3:
    [  163.735294] init            R  running task        0     1      0
0x0400000a
    [  163.735921] Call trace:
    [  163.736160]  __switch_to+0xe4/0x138
    [  163.736475]  lock_timer_base+0x5c/0xa0
    [  163.736811]  try_to_del_timer_sync+0x30/0x98
    [  163.737192]  del_timer_sync+0x50/0x60
    [  163.737528]  schedule_timeout+0x19c/0x478
    [  163.737889]  clk_gate_endisable+0x2c/0xc8

    Change-Id: I663b6b3e0b081ad17bf2845629b34e2ec9d2d76d
    Signed-off-by: Wu Liangqing <wlq@rock-chips.com>
```

由于上面这个提交生效的前提是开机时pmic的某些寄存器配置恢复默认状态，但是由于这些寄存器配置在不断电的情况下会保留上一次的值，所以上面这个补丁在工具以及ota升级情况下是无效的，也就是升级完的第一次reboot操作仍然会出现卡死问题，解决办法:
断电一次，使pmic重新复位
带电池的机器可以在串口或者ADB输入如下命令来清空PMIC

```
i2cset  -yf 0 0x20 0x99 0x0 b
i2cset  -yf 0 0x20 0xa4 0x0 b
```

RK356X IO-Domain GPIO电压配置确认，GPIO电压没配置对会导致芯片GPIO烧坏

RK3566/RK3568的IO-Domain GPIO电压默认都配置为3.3V，实际需要根据硬件原理图来配置，以RK3566/RK3568 EVB板子为例：
IO-Domain的默认配置：

```
arch/arm64/boot/dts/rockchip/rk3568-evb.dtsi
&pmu_io_domains {
         status = "okay";
         pmuio2-supply = <&vcc3v3_pmu>;
         vccio1-supply = <&vccio_acodec>;
         vccio3-supply = <&vccio_sd>;
         vccio4-supply = <&vcc_3v3>;
         vccio5-supply = <&vcc_3v3>;
         vccio6-supply = <&vcc_3v3>;
         vccio7-supply = <&vcc_3v3>;
    };
```

根据实际硬件原理图应该改为：

```
&pmu_io_domains {
         status = "okay";
         pmuio2-supply = <&vcc3v3_pmu>;
         vccio1-supply = <&vccio_acodec>;
         vccio3-supply = <&vccio_sd>;
         vccio4-supply = <&vcc_1v8>;
         vccio5-supply = <&vcc_3v3>;
         vccio6-supply = <&vcc_1v8>;
         vccio7-supply = <&vcc_3v3>;
    };
```

IO-Domain配置流程如下：
第一步：获取硬件原理图并确认硬件电源的设计方案
本文以RK_EVB1_RK3568_DDR4P216SD6_V10_20200911 EVB板为例进行介绍。
硬件原理图：RK_EVB1_RK3568_DDR4P216SD6_V10_20200911.pdf
电源方案：从硬件原理图分析，EVB板RK_EVB1_RK3568_DDR4P216SD6_V10_20200911是带PMU（RK809-5）方案
第二步：查找对应的内核dts配置文件
由第一步可知，该EVB板的硬件电源设计是带PMU方案的，对应的内核dts配置文件位于：
arch/arm64/boot/dts/rockchip/rk3568-evb.dtsi （本文讨论的方案）
第三步：修改内核dts的电源域配置节点pmu_io_domains

```
&pmu_io_domains {
        status = "okay";
        pmuio2-supply = <&vcc3v3_pmu>;
        vccio1-supply = <&vccio_acodec>;
        vccio3-supply = <&vccio_sd>;
        vccio4-supply = <&vcc_1v8>;
        vccio5-supply = <&vcc_3v3>;
        vccio6-supply = <&vcc_1v8>;
        vccio7-supply = <&vcc_3v3>;
    };
```

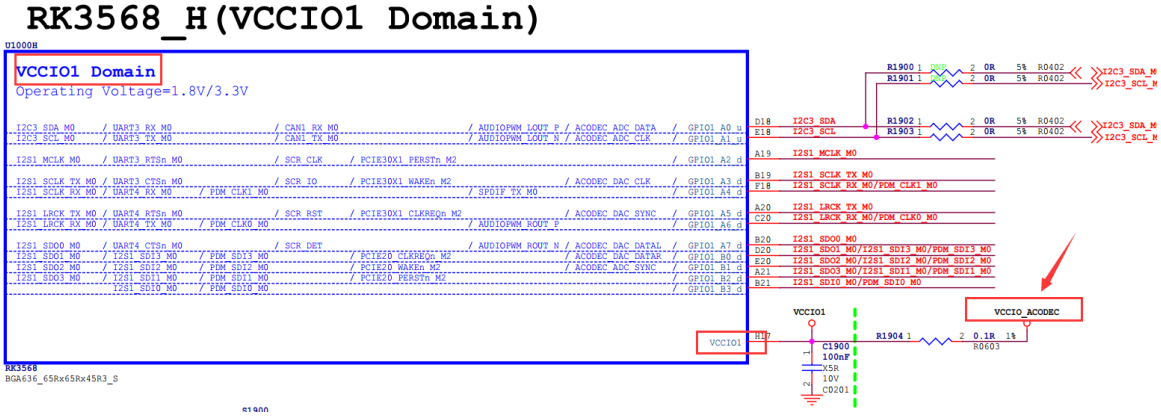
vccio1-supply为例，首先查看硬件原理图确认vccio1电源域（VCCIO1）的配置如图所示。

IO Power Domain Map

Updates must be Revision accordingly!

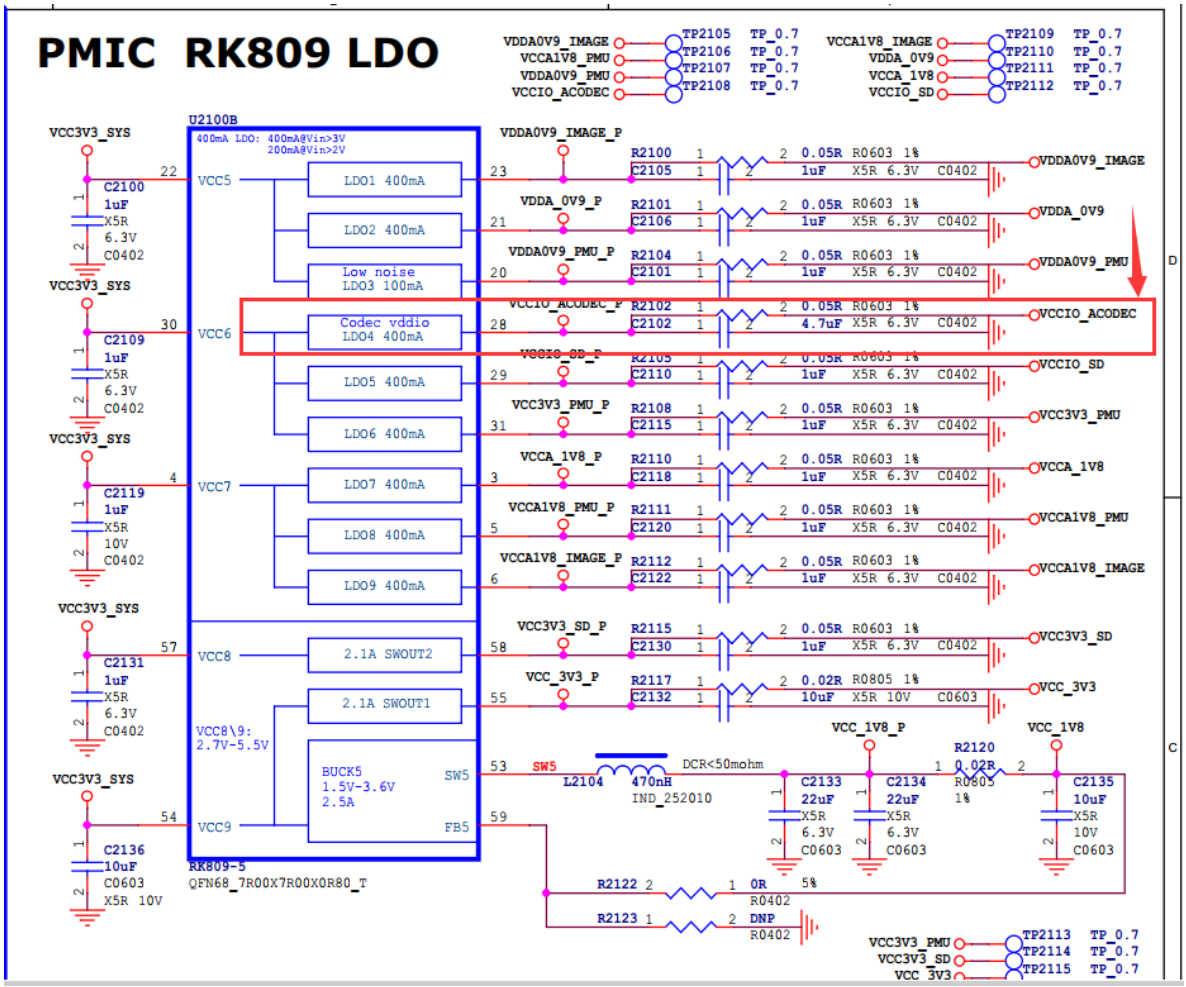
IO Domain	Pin Num	Support IO Voltage		Actual assigned IO Domain Voltage			Notes
		3.3V	1.8V	Supply Power Net Name	Power Source	Voltage	
PMUIO1	Pin Y20	✓	✗	VCC3V3_PMU	VCC3V3_PMU	3.3V	
PMUIO2	Pin W19	✓	✓	VCC3V3_PMU	VCC3V3_PMU	3.3V	
VCCIO1	Pin H17	✓	✓	VCCIO_ACODEC	VCCIO_ACODEC	3.3V	
VCCIO2	Pin H18	✓	✓	VCCIO_FLASH	VCC_1V8	1.8V	PIN "FLASH_VOL_SEL" must be logic High if VCCIO_FLASH=3.3V,FLASH_VOL_SEL must be logic lo
VCCIO3	Pin L22	✓	✓	VCCIO_SD	VCCIO_SD	3.3V	
VCCIO4	Pin J21	✓	✓	VCCIO4	VCC_1V8	1.8V	
VCCIO5	Pin V10 Pin V11	✓	✓	VCCIO5	VCC_3V3	3.3V	
VCCIO6	Pin R9 Pin U9	✓	✓	VCCIO6	VCC_1V8	1.8V	
VCCIO7	Pin V12	✓	✓	VCCIO7	VCC_3V3	3.3V	

在硬件原理图上面搜索`VCCIO1`，如下



从上图找到`VCCIO1`的电源是`vccio_acodec`。

在原理图上搜索`vccio_acodec`，可以找到如下图。



从上图找到`vccio_acodec`是由RK809的LDO4供电。
从软件的DTS里面找到LDO_REG4（LDO4）的配置信息,如下

```
vccio_acodec: LDO_REG4 {

                regulator-always-on;
        regulator-boot-on;
        regulator-min-microvolt = <3300000>;
        regulator-max-microvolt = <3300000>;
        regulator-name = "vccio_acodec";
        regulator-state-mem {
                regulator-off-in-suspend;
        };
};
```

将上面的`vccio_acodec`配置到pmu_io_domains节点中的vccio1-supply = <&vccio_acodec>;即可完成vccio1的电压配置

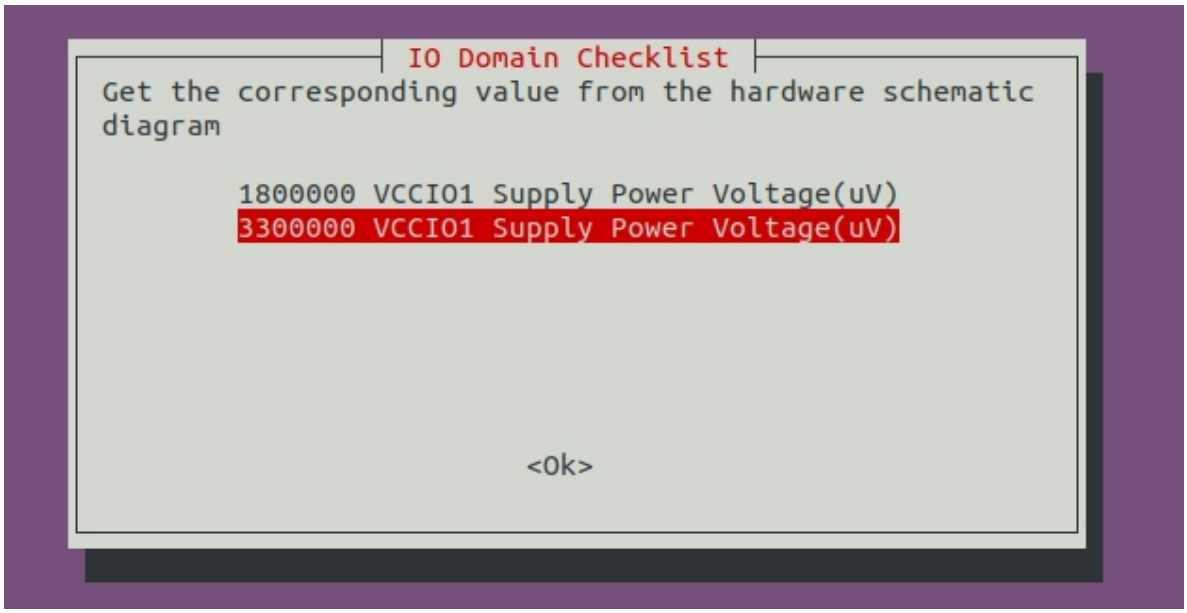
```
&pmu_io_domains {
        status = "okay";
        pmuio2-supply = <&vcc3v3_pmu>;
        vccio1-supply = <&vccio_acodec>;
        vccio3-supply = <&vccio_sd>;
        vccio4-supply = <&vcc_1v8>;
        vccio5-supply = <&vcc_3v3>;
        vccio6-supply = <&vcc_1v8>;
        vccio7-supply = <&vcc_3v3>;
    };
```

其他几路按照上面的方式配置即可，其中vccio2是硬件固定电压不需要配置

- pmuio2
- vccio1
- vccio3
- vccio5
- vccio6
- vccio7
  GPIO电压按照上面的方式配置即可

RK356X kernel编译弹出IO-Domain确认对话框

编译kernel时弹出如下对话框：



弹出这个对话框目的是检查实际硬件原理图和软件dts的GPIO电压是否匹配， 需要根据硬件原理图的实际设计电压来选择（对话框中选择的值不会保存到dts中，dts需要手动去修改）。如果你是软件工程师请与你们的硬件工程师一起核对确认，这个很重要。如果GPIO电压配置不对会导致芯片GPIO烧坏。
当你正确确认GPIO电压后这个对话框就不会在弹出（输入值和dts配置的值相同），如果dts名字或者dts里面的io-domian发生变化，则会在继续弹出进行确认。

RK356x PCIE模块相关问题

PCIE3.0口如果没有提供外部时钟给芯片但是dts中使能了该接口，会导致开机卡死问题。

PCIE3.0因为需要外部提供时钟，所以如果在dts中使能了pcie3.0，但是外部又没有提供时钟给rk356x pcie3.0模块（比如没有贴外部时钟模块，或者时钟模块工作异常），则会导致系统卡死问题，问题log如下所示：

```
[   21.449927] rcu: INFO: rcu_preempt detected stalls on CPUs/tasks:
[   21.449935] rcu: INFO: rcu_sched detected stalls on CPUs/tasks:
[   21.449960] rcu:     1-...0: (1 GPs behind) idle=486/1/0x4000000000000000
softirq=29/29 fqs=2057
[   21.449989] rcu:     1-...0: (87 ticks this GP) idle=486/1/0x4000000000000000
softirq=7/29 fqs=2100
[   21.450003] rcu:     2-...0: (100 ticks this GP)
idle=412/1/0x4000000000000000 softirq=7/52 fqs=2100
[   21.450010] rcu:     (detected by 0, t=6302 jiffies, g=-1183, q=1)
[   21.450023] Task dump for CPU 1:
```

```
[   21.450031] rk-pcie         R  running task        0    53      2 0x0000002a
[   21.450048] Call trace:
[   21.450065] rcu:     2-...0: (15 ticks this GP) idle=412/1/0x4000000000000000
softirq=37/52 fqs=2057
[   21.450071] rcu:     (detected by 3, t=6302 jiffies, g=-1147, q=720)
[   21.450084] Task dump for CPU 1:
[   21.450092] rk-pcie         R  running task        0    53      2 0x0000002a
[   21.450125]  __switch_to+0xe4/0x138
[   21.450139]  kthread+0x12c/0x158
[   21.450150]  ret_from_fork+0x10/0x18
[   21.450156] Task dump for CPU 2:
[   21.450163] kworker/u8:1    R  running task        0    32      2 0x0000002a
[   21.450178] Call trace:
[   21.450199] Workqueue: events_unbound enable_ptr_key_workfn
[   21.450216]  __switch_to+0xe4/0x138
[   21.450224] Call trace:
[   21.450239]  kthread+0x12c/0x158
[   21.450249]  ret_from_fork+0x10/0x18
[   21.450256] Task dump for CPU 2:
[   21.450270]  __switch_to+0xe4/0x138
[   21.450283]  bp_hardening_data+0x0/0x10
[   21.450300] kworker/u8:1    R  running task        0    32      2 0x0000002a
[   21.450327] Workqueue: events_unbound enable_ptr_key_workfn
[   21.450340] Call trace:
[   21.450356]  __switch_to+0xe4/0x138
[   21.450372]  bp_hardening_data+0x0/0x10
[   48.026601] watchdog: BUG: soft lockup - CPU#3 stuck for 22s! [swapper/0:1]
[   48.026638] Modules linked in:
[   48.026664] CPU: 3 PID: 1 Comm: swapper/0 Not tainted 4.19.172 #10
[   48.026676] Hardware name: Rockchip RK3568 EVB1 DDR4 V10 Board (DT)
[   48.026692] pstate: 80c00009 (Nzcv daif +PAN +UAO)
[   48.026716] pc : smp_call_function_many+0x314/0x350
[   48.026753] lr : smp_call_function_many+0x2d4/0x350
  ... ...
[   48.032654] Call trace:
[   48.032678]  smp_call_function_many+0x314/0x350
[   48.032700]  smp_call_function+0x38/0x68
[   48.032729]  on_each_cpu+0x30/0x80
[   48.032759]  clock_was_set+0x1c/0x28
[   48.032787]  do_settimeofday64+0x130/0x180
[   48.032819]  rtc_hctosys+0x78/0x118
[   48.032846]  __rtc_register_device+0xd4/0x160
[   48.032874]  rk808_rtc_probe+0x174/0x2c0
[   48.032904]  platform_drv_probe+0x50/0xa8
[   48.032922]  really_probe+0x228/0x2a0
[   48.032940]  driver_probe_device+0x58/0x100
[   48.032969]  __device_attach_driver+0x90/0xc0
[   48.032995]  bus_for_each_drv+0x70/0xc8
[   48.033023]  __device_attach+0xec/0x148
[   48.033051]  device_initial_probe+0x10/0x18
[   48.033077]  bus_probe_device+0x94/0xa0
[   48.033103]  device_add+0x384/0x698
[   48.033130]  platform_device_add+0x108/0x248
[   48.033160]  mfd_add_device+0x2d8/0x358
[   48.033177]  mfd_add_devices+0xb0/0x170
[   48.033211]  devm_mfd_add_devices+0x78/0xe0
[   48.033238]  rk808_probe+0x724/0x780
[   48.033264]  i2c_device_probe+0x200/0x278
```

```
[   48.033296]  really_probe+0x228/0x2a0
[   48.033323]  driver_probe_device+0x58/0x100
[   48.033350]  __device_attach_driver+0x90/0xc0
[   48.033377]  bus_for_each_drv+0x70/0xc8
[   48.033404]  __device_attach+0xec/0x148
[   48.033422]  device_initial_probe+0x10/0x18
[   48.033438]  bus_probe_device+0x94/0xa0
[   48.033463]  device_add+0x384/0x698
[   48.033488]  device_register+0x1c/0x28
[   48.033516]  i2c_new_device+0x1c0/0x398
[   48.033546]  of_i2c_register_devices+0x134/0x160
[   48.033573]  i2c_register_adapter+0x150/0x408
[   48.033601]  __i2c_add_numbered_adapter+0x5c/0xa8
[   48.033628]  i2c_add_adapter+0xa4/0xd8
[   48.033655]  rk3x_i2c_probe+0x324/0x428
[   48.033674]  platform_drv_probe+0x50/0xa8
[   48.033708]  really_probe+0x228/0x2a0
[   48.033736]  driver_probe_device+0x58/0x100
[   48.033764]  device_driver_attach+0x6c/0x78
[   48.033791]  __driver_attach+0xb0/0xf0
[   48.033818]  bus_for_each_dev+0x68/0xc8
[   48.033844]  driver_attach+0x20/0x28
[   48.033870]  bus_add_driver+0xf8/0x1f0
[   48.033898]  driver_register+0x60/0x110
[   48.033917]  __platform_driver_register+0x40/0x48
[   48.033938]  rk3x_i2c_driver_init+0x18/0x20
[   48.033966]  do_one_initcall+0x48/0x240
[   48.033997]  kernel_init_freeable+0x210/0x37c
[   48.034024]  kernel_init+0x10/0x108
[   48.034052]  ret_from_fork+0x10/0x18
```

出现该log时，如果确实有贴外部时钟模块，则检查模块工作是否正常；如果没有贴，则说明不需要使用PCIE3.0口，需要在dts中disable如下几项：

```
&pcie3x1 {
   status = "disabled";
};

&pcie3x2 {
   status = "disabled";
};
```

rk356x pcie2x1控制器和sata2控制器不可同时开启

由于rk356x pcie2x1控制器和sata2控制器是复用的，不能同时开启，否则会导致相互干扰产生异常。

Android Samba功能

参考文档

```
RKDocs/android/Rockchip_Introduction_Android_Samba_CN.pdf
```

NFS启动

参考文档及补丁：

```
RKDocs/android/patches/customized_functions/nfs_boot_patch_v1.1.0.zip
```

附录 A 编译开发环境搭建 Compiling and development environment setup

Initializing a Build Environment

This section describes how to set up your local work environment to build the Android source files. You must use Linux or Mac OS; building under Windows is not currently supported.
For an overview of the entire code-review and code-update process, see Life of a Patch.
Note: All commands in this site are preceded by a dollar sign ($) to differentiate them from output or entries within files. You may use the Click to copy feature at the top right of each command box to copy all lines without the dollar signs or triple-click each line to copy it individually without the dollar sign.

Choosing a Branch

Some requirements for the build environment are determined by the version of the source code you plan to compile. For a full list of available branches, see Build Numbers. You can also choose to download and build the latest source code (called master), in which case you will simply omit the branch specification when you initialize the repository.
After you have selected a branch, follow the appropriate instructions below to set up your build environment.

Setting up a Linux build environment

These instructions apply to all branches, including master.
The Android build is routinely tested in house on recent versions of Ubuntu LTS (14.04) and Debian testing. Most other distributions should have the required build tools available.
For Gingerbread (2.3.x) and newer versions, including the master branch, a 64-bit environment is required. Older versions can be compiled on 32-bit systems.
Note: See Requirements for the complete list of hardware and software requirements, then follow the detailed instructions for Ubuntu and Mac OS below.

Installing the JDK

The master branch of Android in the Android Open Source Project (AOSP) comes with prebuilt versions of OpenJDK below prebuilts/jdk/ so no additional installation is required.
Older versions of Android require a separate installation of the JDK. On Ubuntu, use OpenJDK.
See JDK Requirements for precise versions and the sections below for instructions.
For Ubuntu >= 15.04
Run the following:

```
sudo apt-get update
sudo apt-get install openjdk-8-jdk
```

For Ubuntu LTS 14.04
There are no available supported OpenJDK 8 packages for Ubuntu 14.04. The Ubuntu 15.04 OpenJDK 8 packages have been used successfully with Ubuntu 14.04. Newer package versions (e.g. those for 15.10, 16.04) were found not to work on 14.04 using the instructions below.

1. Download the .deb packages for 64-bit architecture from old-releases.ubuntu.com:

```
openjdk-8-jre-headless_8u45-b14-1_amd64.deb with SHA256
0f5aba8db39088283b51e00054813063173a4d8809f70033976f83e214ab56c0
openjdk-8-jre_8u45-b14-1_amd64.deb with SHA256
9ef76c4562d39432b69baf6c18f199707c5c56a5b4566847df908b7d74e15849
openjdk-8-jdk_8u45-b14-1_amd64.deb with SHA256
6e47215cf6205aa829e6a0a64985075bd29d1f428a4006a80c9db371c2fc3c4c
```

2. Optionally, confirm the checksums of the downloaded files against the SHA256 string listed with each package above. For example, with the sha256sum tool:

```
    sha256sum {downloaded.deb file}
```

3. Install the packages:

```
    sudo apt-get update
```

Run dpkg for each of the .deb files you downloaded. It may produce errors due to missing dependencies:

```
    sudo dpkg -i {downloaded.deb file}
```

To fix missing dependencies:

```
sudo apt-get -f install
```

Update the default Java version - optional
Optionally, for the Ubuntu versions above update the default Java version by running:

```
sudo update-alternatives --config javasudo update-alternatives --config javac
```

Note: If, during a build, you encounter version errors for Java, see Wrong Java version for likely causes and solutions.
Installing required packages (Ubuntu 14.04)
You will need a 64-bit version of Ubuntu. Ubuntu 14.04 is recommended.

```
sudo apt-get install git-core gnupg flex bison gperf build-essential zip curl
zlib1g-dev gcc-multilib g++-multilib libc6-dev-i386 lib32ncurses5-dev x11proto-
core-dev libx11-dev lib32z-dev ccache libgl1-mesa-dev libxml2-utils xsltproc
unzip python-pyelftools python3-pyelftools device-tree-compiler libfdt-dev
libfdt1 libssl-dev liblz4-tool python-dev
```

Note: To use SELinux tools for policy analysis, also install the python-networkx package. Note: If you are using LDAP and want to run ART host tests, also install the libnss-sss:i386 package.

Configuring USB Access

Under GNU/linux systems (and specifically under Ubuntu systems), regular users can't directly access USB devices by default. The system needs to be configured to allow such access.
The recommended approach is to create a file /etc/udev/rules.d/51-android.rules (as the root user) and to copy the following lines in it. must be replaced by the actual username of the user who is authorized to access the phones over USB.

```
# adb protocol on passion (Rockchip products)
SUBSYSTEM=="usb", ATTR{idVendor}=="2207", ATTR{idProduct}=="0010", MODE="0600",
OWNER="<username>"
```

Those new rules take effect the next time a device is plugged in. It might therefore be necessary to unplug the device and plug it back into the computer.
This is known to work on both Ubuntu Hardy Heron (8.04.x LTS) and Lucid Lynx (10.04.x LTS).
Other versions of Ubuntu or other variants of GNU/linux might require different configurations.
References : http://source.android.com/source/initializing.html

附录 B SSH公钥操作说明 SSH public key operation instruction

附录 B-1 SSH公钥生成 SSH public key generation

使用如下命令生成：
Use the following command to generate:

```
ssh-keygen -t rsa -C "user@host"
```

请将user@host替换成您的邮箱地址。
Please replace user@host with your email address.

命令运行完成会在你的目录下生成key文件。
It will generate the key file in your directory after the command is executed successfully.

请妥善保存生成的私钥文件id_rsa和密码，并将id_rsa.pub发邮件给SDK发布服务器的管理员。
Please keep carefully the generated private key file id_rsa and password, and send id_rsa.pub to SDK release server admin through email.

附录 B-2 使用key-chain管理密钥 Use key-chain to manage the key

推荐您使用比较简易的工具keychain管理密钥。
Recommend you use the simple tool keychain to manage the key.
具体使用方法如下：
The detailed usage is as follows:

1. 安装keychain软件包：
   Install keychain software package:

```
$sudo aptitude install keychain
```

2. 配置使用密钥：
   Configure to use the key:

```
$vim ~/.bashrc
```

增加下面这行：
Add the following command:

```
eval `keychain --eval ~/.ssh/id_rsa`
```

其中，id_rsa是私钥文件名称。
Among which, id_rsa is the file name of the private key.
以上配置以后，重新登录控制台，会提示输入密码，只需输入生成密钥时使用的密码即可，若无密码可不输入。
Log in the console again after configuring as above, and it will prompt to input the password. Only need to input the password used for generating the key if there is one.
另外，请尽量不要使用sudo或root用户，除非您知道如何处理，否则将导致权限以及密钥管理混乱。
Besides, please avoid using sudo or root user unless you know clearly how to deal with, otherwise it will cause the authority and key management problems.

附录 B-3 多台机器使用相同ssh公钥 Multiple devices use the same ssh public key

在不同机器使用，可以将你的ssh私钥文件id_rsa拷贝到要使用的机器的“~/.ssh/id_rsa”即可。
In order to use on different devices, you can copy ssh private key file id_rsa to the target device “~/.ssh/id_rsa”.
在使用错误的私钥会出现如下提示，请注意替换成正确的私钥。
Below hint will show up if using the wrong private key. Please replace with the correct private key.

添加正确的私钥后，就可以使用git 克隆代码，如下图。
After adding the correct private key, you can use git to clone code, shown as below picture:

添加ssh私钥可能出现如下提示错误。
Below error may occur when adding ssh private key:

```
Agent admitted failture to sign using the key
```

在console输入如下命令即可解决。
Input the following command at console can fix it.

```
ssh-add ~/.ssh/id_rsa
```

附录 B-4 一台机器切换不同ssh公钥 Switch different ssh public keys on one device

可以参考ssh_config文档配置ssh。
You can refer to ssh_config document to configure ssh.

```
~$ man ssh_config
```

通过如下命令，配置当前用户的ssh配置。
Use the following commands to configure ssh for current user.

```
~$ cp /etc/ssh/ssh_config ~/.ssh/config
~$ vi .ssh/config
```

如图，将ssh使用另一个目录的文件"~/.ssh1/id_rsa"作为认证私钥。通过这种方法，可以切换不同的密钥。
As below picture, identify another directory ssh file "~/.ssh1/id_rsa" as certificate private key. In this way, you can switch different keys.

附录 B-5 密钥权限管理 Key authority management

服务器可以实时监控某个key的下载次数、IP等信息，如果发现异常将禁用相应的key的下载权限。
The server can real-time monitor for the specific key the download times, IP and other information. If any abnormal case is found, it will prohibit the download authority of the corresponding key.
请妥善保管私钥文件。并不要二次授权与第三方使用。
Please keep carefully the private key file. DO NOT re-authorize it to the third party.

附录 B-6 Git权限申请说明 Git authority application instruction

参考上述章节，生成公钥文件，发邮件至 fae@rock-chips.com，申请开通SDK代码下载权限。
Refer to above chapters, generate the public key file, and send email to fae@rock-chips.com applying for SDK code download authority.