密级状态：绝密( ) 秘密( ) 内部( ) 公开(√)

Rockchip Android 10.0 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-218
当前版本:	V1.2.6
作者:	吴良清
完成日期:	2020-08-06
审核:	陈海燕
审核日期:	2020-08-06

版本号	作者	修改日期	修改说明	备注
V1.0	吴良清	2019-12-13	发布初稿	
V1.2	吴良清	2019-12-31	发布RK3399	
V1.2.1	吴良清	2020-03-24	增加常见问题说明	
V1.2.2	吴良清	2020-04-16	发布RK3368；增加常见问题说明	
V1.2.3	吴良清	2020-05-06	发布RK3288	
V1.2.4	吴良清	2020-06-18	增加常见问题说明，修正图片显示异常	
V1.2.5	吴良清	2020-06-24	发布RK3368A	
V1.2.6	吴良清	2020-08-06	增加常见问题	

文档问题反馈：wlq@rock-chips.com

Rockchip Android 10.0 SDK支持芯片

芯片平台	是否支持	SDK版本
RK3326	支持	RKR1
PX30	支持	RKR1
RK3126C	支持	RKR3
RK3399	支持	RKR5
RK3368	支持	RKR7
RK3288W	支持	RKR8
RK3688A	支持	RKR9

Rockchip Android 10.0 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_Qt/manifests.git -m Android10.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_Qt/manifests.git -m Android10.xml --
mirror
```

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_Android10.0_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 Rockcip_Android10.0_SDK
cat Rockchip_Android10.0_SDK_Release.tar.gz* | tar -zx -C
Rockcip_Android10.0_SDK
cd Rockcip_Android10.0_SDK
.repo/repo/repo sync -l
.repo/repo/repo sync -c
```

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 rk3328_box-userdebug
============================================================
```

编译命令说明

芯片平台	Uboot编译	kernel编译	Android编译
RK3326	cd uboot ./make.sh rk3326	cd kernel make ARCH=arm64 rockchip_defconfig android-10.config rk3326.config make ARCH=arm64 rk3326-863-lp3-v10-rkisp1.img -j24	source build/envsetup.sh lunch rk3326_q-userdebug make -j24 ./mkimage.sh
PX30	cd uboot ./make.sh px30	cd kernel make ARCH=arm64 rockchip_defconfig android-10.config rk3326.config make ARCH=arm64 px30-evb-ddr3-v10-avb.img -j24	source build/envsetup.sh lunch PX30_Android10-userdebug make -j24 ./mkimage.sh
RK3368	cd uboot ./make.sh rk3368	cd kernel make ARCH=arm64 rockchip_defconfig android-10.config make ARCH=arm64 rk3368-808-evb.img -j20	source build/envsetup.sh lunch rk3368_Android10-userdebug make -j24 ./mkimage.sh
RK3368A	cd uboot ./make.sh rk3368	cd kernel make ARCH=arm64 rockchip_defconfig android-10.config make ARCH=arm64 rk3368a-817-tablet.img -j20	source build/envsetup.sh lunch rk3368a_qgo-userdebug make -j24 ./mkimage.sh
RK3399	cd uboot ./make.sh rk3399	cd kernel make ARCH=arm64 rockchip_defconfig android-10.config rk3399.config 挖掘机 make ARCH=arm64 rk3399-sapphire-excavator-edp-avb.img -j24 RK3399 LPDDR4+RK809 IND开发板 make ARCH=arm64 rk3399-evb-ind-lpddr4-android-avb.img -j24	source build/envsetup.sh lunch rk3399_Android10-userdebug make -j24 ./mkimage.sh
RK3399pro	cd uboot ./make.sh rk3399pro	cd kernel make ARCH=arm64 rockchip_defconfig android-10.config rk3399.config 挖掘机 make ARCH=arm64 rk3399pro-evb-v11.img -j24	source build/envsetup.sh lunch rk3399pro_Android10-userdebug make -j24 ./mkimage.sh

芯片平台	Uboot编译	kernel编译	Android编译
RK3126C	cd uboot ./make.sh rk3126	cd kernel make ARCH=arm rockchip_defconfig android-10.config make ARCH=arm rk3126-m88.img -j24	source build/envsetup.sh lunch rk3126c_qgo-userdebug make -j24 ./mkimage.sh
RK3328W	cd uboot ./make.sh rk3328	cd kernel make ARCH=arm64 rockchip_defconfig android-10.config ## 联通样机BOX编译命令: make ARCH=arm64 rk3328-box-liantong-avb.img -j24 ## EVB 板编译命令: make ARCH=arm64 rk3328-evb-android-avb.img -j24	source build/envsetup.sh ## 联通样机BOX编译命令: lunch rk3328_box-userdebug ## EVB 板 ATV编译命令: lunch rk3328_atv-userdebug make -j24 ./mkimage.sh
RK3229	cd uboot ./make.sh rk322x	cd kernel make ARCH=arm rockchip_defconfig android-10.config ## EVB 板编译命令: make ARCH=arm rk3229-evb-android-avb.img -j24	source build/envsetup.sh ## EVB 板 BOX编译命令: lunch rk3328_box-userdebug make -j24 ./mkimage.sh
RK3288	cd uboot ./make.sh rk3288	cd kernel make ARCH=arm rockchip_defconfig android-10.config ## EVB 板编译命令: make ARCH=arm rk3288-evb-android-rk808-edp-avb.img -j24	source build/envsetup.sh ## EVB 板: lunch rk3288_Android10-userdebug make -j24 ./mkimage.sh

其他编译说明

Android10.0不能直接烧写kernel.img和resource.img

Android10.0的kernel.img和resource.img包含在boot.img中，更新编译kernel后需要在android根目录下执行./mkimage.sh重新打包boot.img。打包后烧写rockdev下面的boot.img，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，所以编译的时候需要用`BOOT_IMG=xxx`参数指定boot.img的路径，命令如下：
以`RK3399`样机为例，编译时替换对应的boot.img及dts：

```
cd kernel
make ARCH=arm64 rockchip_defconfig android-10.config rk3399.config
make ARCH=arm64 BOOT_IMG=../rockdev/Image_rk3368_Android10/boot.img rk3399-evb-ind-lpddr4-android-avb.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**）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。

支持从P升级到Q版本:

以RK3326为例

- 编译命令:

```
source build/envsetup.sh
lunch rk3326_pie-userdebug
make -j24
./mkimage.sh
```

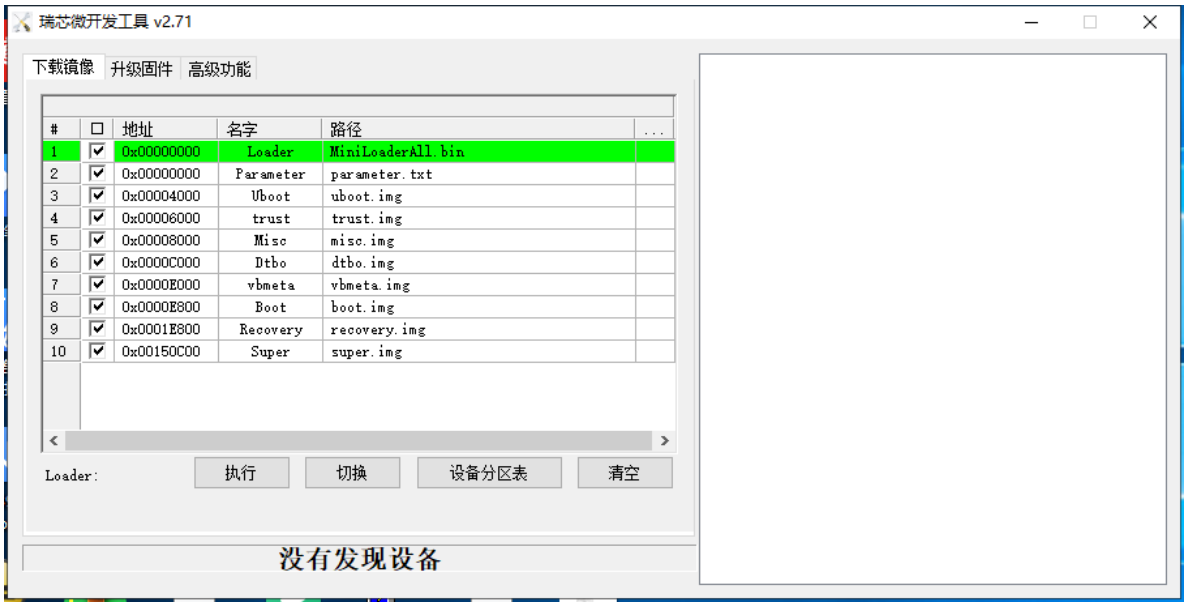
- 请把odm.img烧写到oem的位置，附烧写工具配置：

固件烧写

固件烧写工具

Windows烧写工具：

```
RKTools/windows/AndroidTool/AndroidTool_Release_v2.71
```



Linux工具：

```
RKTools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool_v1.43
```

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RK3326为例，这里lunch的是rk3326_qgo-userdebug）

```
rockdev/Image-rk3326_qgo/
├── boot.img
├── config.cfg
├── dtbo.img
├── kernel.img
├── MiniLoaderAll.bin
├── misc.img
├── odm.img
├── parameter.txt
├── pcba_small_misc.img
├── pcba_whole_misc.img
├── recovery.img
├── resource.img
├── super.img
├── system.img
├── trust.img
├── uboot.img
├── update.img
├── vbmeta.img
└── vendor.img
```

工具烧写如下文件即可：

```
rockdev/Image-rk3326_qgo/
├── boot.img
├── dtbo.img
├── MiniLoaderAll.bin
├── misc.img
├── parameter.txt
├── recovery.img
├── super.img
├── trust.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
dtbo.img	Device Tree Overlays 参考下面的dtbo章节说明
kernel.img	包含kernel，目前无法单独烧写，需要打包到boot.img内烧写
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
odm.img	包含android odm，包含在super.img分区内,单独烧写需要用fastboot烧写
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
resource.img	包含dtb，kernel和uboot阶段的log及uboot充电logo，目前无法单独烧写，需要打包到boot.img内烧写
super.img	包含odm、vendor、system分区内容
system.img	包含android system，包含在super.img分区内,单独烧写需要同fastboot烧写
trust.img	包含BL31、BL32
uboot.img	包含uboot固件
vbmeta.img	包含avb校验信息，用于AVB校验
vendor.img	包含android vendor，包含在super.img分区内,单独烧写需要同fastboot烧写
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

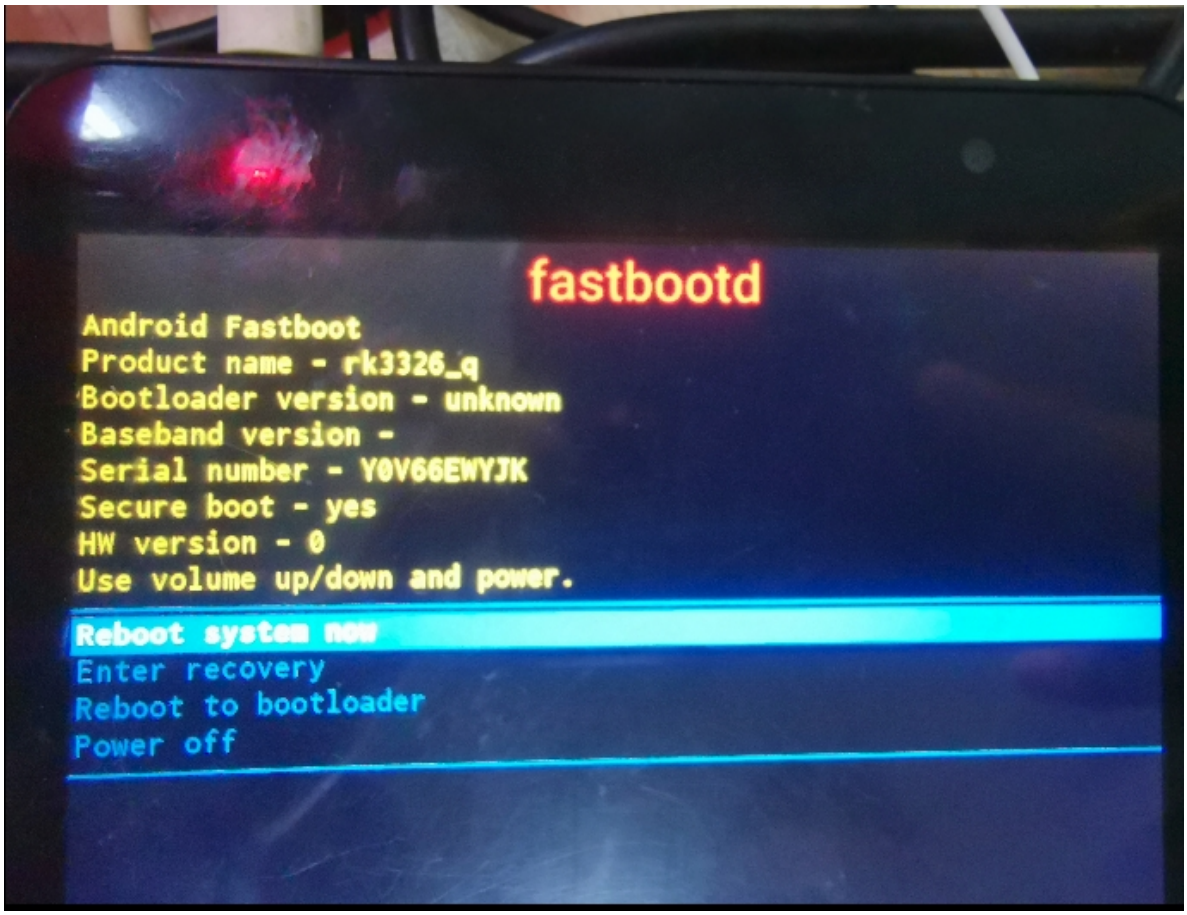
Q的新设备支持动态分区，已经移除了system/vendor/odm分区，请烧写super.img，单独烧写system/vendor/odm可以用`fastbootd`，要求adb和fastboot版本均为最新, SDK提供了编译好的工具包:

```
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:



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;

Android 10.0特性

编译系统

SDK版本号变更

首先，在PDK代码中取消了`PLATFORM_VERSION`这个宏，并且在后续的AOSP中，此宏会变更为10.0，导致编译系统判断变量出错，因此统一修改为`PLATFORM_SDK_VERSION`，如果你发现某个功能不能用或崩溃了，到源码目录去查看有没有引用到这个宏。
两者的对应关系如下：

PLATFORM_VERSION	PLATFORM_SDK_VERSION
7.0	24
7.1	25
8.0	26
8.1	27
9.0	28
10.0	29
…	…

mk文件重复加载

这个问题早期版本就一直存在，只是影响不大，会少许影响编译速度。新版本中引入了`readonly`机制，指定为只读的变量不可以再赋值，所以在重复加载时会编译报错。对此，推荐以下结构避免被重复加载：

```
# First lunching is Q, api_level is 29
PRODUCT_SHIPPING_API_LEVEL := 29
PRODUCT_FSTAB_TEMPLATE := $(LOCAL_PATH)/fstab.in
PRODUCT_DTBO_TEMPLATE := $(LOCAL_PATH)/dt-overlay.in
PRODUCT_BOOT_DEVICE := ff390000.dwmmc,ff3b0000.nandc

include device/rockchip/common/build/rockchip/DynamicPartitions.mk
include device/rockchip/common/BoardConfig.mk
include device/rockchip/rk3326/rk3326_q/BoardConfig.mk
$(call inherit-product, device/rockchip/rk3326/device-common.mk)
$(call inherit-product, device/rockchip/common/device.mk)
$(call inherit-product, frameworks/native/build/tablet-10in-xhdpi-2048-dalvik-
heap.mk)

PRODUCT_CHARACTERISTICS := tablet

PRODUCT_NAME := rk3326_q
PRODUCT_DEVICE := rk3326_q
PRODUCT_BRAND := rockchip
PRODUCT_MODEL := rk3326_q
PRODUCT_MANUFACTURER := rockchip
PRODUCT_AAPT_PREF_CONFIG := tvdpi
```

```
#
## add Rockchip properties
#
PRODUCT_PROPERTY_OVERRIDES += ro.sf.lcd_density=160
```

> 相比旧版本，主要是调了下顺序，删了点include
> {.is-info}

mk文件copy机制

在mk中，`PRODUCT_COPY`这个宏只能够执行一次，因此，拷贝的目标文件为同名时，只有前面被声明的命令会执行。常见问题，bringup后`adb`无法使用。

fstab动态生成器

由于Android中存在大量的可配置项，SDK又将用于行业应用，配置的冲突在所难免。增加这个脚本以减少修改配置时，需要额外修改的内容。使用方法:
新增模板文件，并在mk中配置好即可：

`PRODUCT_FSTAB_TEMPLATE := $(LOCAL_PATH)/fstab.in`

```
# Android fstab file.
#<src>                                                <mnt_point>         <type>
<mnt_flags and options>                              <fs_mgr_flags>
# The filesystem that contains the filesystem checker binary (typically /system)
cannot
# specify MF_CHECK, and must come before any filesystems that do specify
MF_CHECK
${_block_prefix}system /system ext4 ro,barrier=1
${_flags_vbmeta},first_stage_mount
${_block_prefix}vendor /vendor ext4 ro,barrier=1 ${_flags},first_stage_mount
${_block_prefix}odm    /odm    ext4 ro,barrier=1 ${_flags},first_stage_mount
/dev/block/by-name/metadata /metadata ext4 nodev,noatime,nosuid,discard,sync
wait,formattable,first_stage_mount
/dev/block/by-name/misc         /misc               emmc      defaults
defaults
/dev/block/by-name/cache          /cache              ext4
noatime,nodiratime,nosuid,nodev,noauto_da_alloc,discard
wait,check

/devices/platform/*usb*   auto vfat defaults      voldmanaged=usb:auto

/dev/block/zram0                                   none                swap
defaults                                                zramsize=50%
# For sdmmc
/devices/platform/ff370000.dwmmc/mmc_host*        auto  auto    defaults
voldmanaged=sdcard1:auto,encryptable=userdata
#  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,fsync_mode=nobarrier
latemount,wait,check,fileencryption=software,quota,formattable,reservedsize=128M
,checkpoint=fs
```

dtbo.img动态生成器

Android 10.0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。
增加这个功能理由同上，可以兼容其它功能，如 `SD Boot`，该功能需要修改 `boot_devices` 为 `sdmmc` 的地址，使用方法：
新增模板文件，并在mk中配置好即可，**注意，使用dtbo时一定要确保dts中存在alias，否则无法成功overlay**：

```
PRODUCT_DTBO_TEMPLATE := $(LOCAL_PATH)/dt-overlay.in
PRODUCT_BOOT_DEVICE := ff390000.dwmmc,ff3b0000.nandc //修改为sdmmc时可以从sd卡启动
```

```
/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>;
};
```

sepolicy的ioctl控制机制

libsync库访问内核节点变更由 `/dev/sw_sync` 变更为 `/sys/kernel/debug/sync/sw_sync`，此项修改会引出大量的sepolicy问题，对此，需要说明sepolicy的ioctl控制机制。

- 在rules中添加ioctl后，还需要声明具体的iocmd，必须结合源代码添加，例如：

```
allow hal_graphics_composer_default debugfs_sw_sync:file { ioctl };
allowxperm hal_graphics_composer_default debugfs_sw_sync:file ioctl {
SYNC_IOC_MERGE SW_SYNC_IOC_INC SW_SYNC_IOC_CREATE_FENCE };
```

新Feature或变更

屏幕旋转

Android 10.0取消了ConfigStoreHAL，问题多且内存占用高，更换回 `prop` 的方式，对此，SDK已经相应地做了修改，使用方法：
mk文件中指定旋转方向：

```
# set screen rotation: 0/90/180/270
SF_PRIMARY_DISPLAY_ORIENTATION := 0
```

`manifest`升级

在引入`treble`技术后(于`Android Oreo`引入)，所有的hal都需要从manifest中声明，并且在新的Android版本中都需要更新`target-level`，所以如果你的manifest.xml文件没有修改，会编译不过。参考：

```
device/rockchip/rk3326/manifest.xml
```

`boot header 2`引入

在原先的`header 1`的基础上，新增了`dtb`的支持，需要在编译时将`dtb`（复数）打包到`boot.img`中，对此，新的product需要指定dtb的目录：

```
BOARD_INCLUDE_DTB_IN_BOOTIMG := true
BOARD_PREBUILT_DTBIMAGE_DIR := kernel/arch/arm64/boot/dts/rockchip
```

对于升级到Q的设备，则要置空：

```
# No need to place dtb into boot.img for the device upgrading to Q.
BOARD_INCLUDE_DTB_IN_BOOTIMG :=
BOARD_PREBUILT_DTBIMAGE_DIR :=
```

`early mount`引入

最新的Android 10中，ramdisk再次回归boot.img，所以分区的mount fstab配置又需要到fstab文件中配置了，但和8.1不同的是，ramdisk中的fstab还可以支持earlymount，而8.1的earlymount只能在dts文件中配置，**而10.0中需要保证节点为空或disabled状态。**因此，需要同时拷贝fstab文件到`vendor/etc`及`ramdisk`下，这里推荐使用`fstab动态生成器`，会自动生成支持earlymount的fstab文件并拷贝到对应的目录，如果不使用，则通过以下的宏拷贝：

```
ifndef PRODUCT_FSTAB_TEMPLATE
$(warning Please add fstab.in with PRODUCT_FSTAB_TEMPLATE in your product.mk)
# To use fstab auto generator, define fstab.in in your product.mk,
# Then include the device/rockchip/common/build/rockchip/RebuildFstab.mk in your
AndroidBoard.mk

PRODUCT_COPY_FILES += \

$(TARGET_DEVICE_DIR)/fstab.rk30board:$(TARGET_COPY_OUT_VENDOR)/etc/fstab.rk30boa
rd \

$(TARGET_DEVICE_DIR)/fstab.rk30board:$(TARGET_COPY_OUT_RAMDISK)/fstab.rk30board
endif # Use PRODUCT_FSTAB_TEMPLATE
```

新的`kernel config`要求

在新的Android 10.0以及后续需要通过的GMS认证中，新增了大量的config，所以在新的编译系统中，建议使用新的内核编译命令：`make ARCH=arm64 rockchip_defconfig android-10.config`

动态分区

开启动态分区后，将不会存在system/vendor/product/odm等物理分区，他们将会合并到super分区中作为动态分区，开启后修改super中的动态分区会更加灵活，后续也能更容易大版本升级。即开启动态分区，要先修改分区表，把system/vendor/product/odm移除，添加一个名为super的分区。具体分区表的修改方法，请参阅`RKDocs/common/RKTools manuals/Rockchip-Parameter-File-Format-Version1.4-CN.pdf`
该功能中相对独立，以模块化的方式处理。修改分区表后，在mk文件中包含此文件即可使能动态分区：

```
include device/rockchip/common/build/rockchip/DynamicPartitions.mk
```

如果觉得这几个分区（system/vendor/odm）预留的太小，push大文件时，可以修改该文件：

```
BOARD_SYSTEMIMAGE_PARTITION_RESERVED_SIZE := 52428800
BOARD_VENDORIMAGE_PARTITION_RESERVED_SIZE := 52428800
BOARD_ODMIMAGE_PARTITION_RESERVED_SIZE := 52428800
```

fastbootd

该功能需要使能动态分区后才能用，配合修改的地方有两处，**注意这里不要修改vendorId，目前使用的是google的bootloader驱动，Windows平台也能正常识别和烧写固件**：

- init.recovery.rk30board.rc文件

```
on early-fs
    setprop sys.usb.controller "ff300000.usb" //这里要修改为对应平台的controller
    setprop sys.usb.configfs 1

on fs && property:sys.usb.configfs=1
    write /config/usb_gadget/g1/bcdDevice 0x0310
    write /config/usb_gadget/g1/bcdUSB 0x0200
    write /config/usb_gadget/g1/os_desc/b_vendor_code 0x1
    write /config/usb_gadget/g1/os_desc/qw_sign "MSFT100"
    write /config/usb_gadget/g1/configs/b.1/MaxPower 500
    symlink /config/usb_gadget/g1/configs/b.1 /config/usb_gadget/g1/os_desc/b.1
```

- reboot mode支持fastboot，前面的`dtbo动态生成器`中便是支持reboot mode的模板。

文件加密

该功能并不是新功能，只是rockchip的sdk中引入的新功能，也是android 10的强制要求功能。
使能文件加密，修改fstab文件，在`fstab动态生成器`中便是支持文件加密的模板：

```
/dev/block/by-name/userdata       /data               f2fs
noatime,nodiratime,nosuid,nodev,discard,inline_xattr
wait,check,notrim,fileencryption=software,quota,reservedsize=128M
```

除了fstab文件的修改，还支持分阶段挂载分区，以提高开机速度，**注意，非文件加密不能使用分阶段挂载，磁盘加密请务必检查一下，否则将无法启动**:
分阶段挂载示例：

```
# do mount_all early can imporve boot time when FBE
```

```
# is enabled
on fs
    mount_all /vendor/etc/fstab.${ro.hardware} --early
on late-fs
    # Start services for bootanim
    start servicemanager
    start hwcomposer-2-1
    start gralloc-2-0
    start surfaceflinger
    start bootanim

    # Mount RW partitions which need run fsck
    mount_all /vendor/etc/fstab.${ro.hardware} --late
```

不分阶段加载示例：

```
on fs
    mount_all /vendor/etc/fstab.${ro.hardware}
```

用户数据检查点 (UDC)

该功能需要data分区为f2fs，修改fstab文件以支持该功能，在`fstab动态生成器`中便是支持UDC的模板：

```
/dev/block/by-name/userdata /data f2fs
noatime,nosuid,nodev,discard,reserve_root=32768,resgid=1065,fsync_mode=nobarrier
latemount,wait,check,fileencryption=software,quota,formattable,reservedsize=128M
,checkpoint=fs
```

avb中vbmeta增加公钥元数据

更高级的avb锁，需要证书才能解锁，详情请参考uboot的security部分文档，SDK中指定以下宏开启编译元数据功能：

```
BOARD_AVB_ENABLE := true
BOARD_AVB_METADATA_BIN_PATH := \
   external/avb/test/data/atx_metadata.bin
```

Android常用配置

新建产品lunch

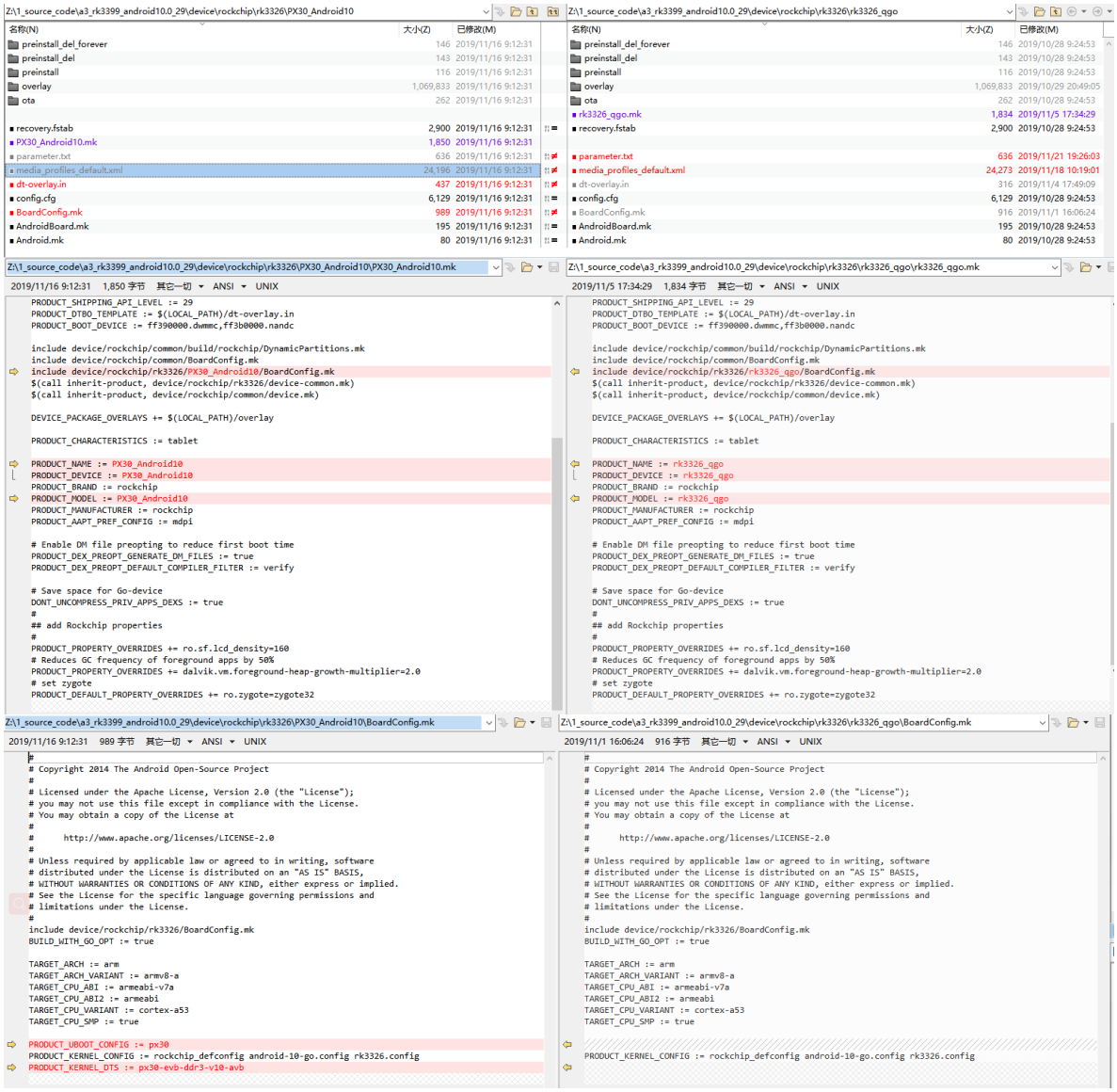
以RK3326平台新建PX30_Android10产品为例，分以下步骤：
1，修改device/rockchip/rk3326/AndroidProducts.mk增加PX30_Android10的lunch

```
diff --git a/AndroidProducts.mk b/AndroidProducts.mk
index 44d8475..e2f7c02 100755
--- a/AndroidProducts.mk
+++ b/AndroidProducts.mk
@@ -15,11 +15,14 @@
 #
```

```
 PRODUCT_MAKEFILES := \
+        $(LOCAL_DIR)/PX30_Android10/PX30_Android10.mk \
         $(LOCAL_DIR)/rk3326_pie/rk3326_pie.mk \
         $(LOCAL_DIR)/rk3326_q/rk3326_q.mk \
         $(LOCAL_DIR)/rk3326_qgo/rk3326_qgo.mk

 COMMON_LUNCH_CHOICES := \
+        PX30_Android10-userdebug \
+        PX30_Android10-user \
         rk3326_q-userdebug \
         rk3326_q-user \
         rk3326_qgo-userdebug \
```

2，在device/rockchip/rk3326目录下新建PX30_Android10目录
参考device/rockchip/下已有的rk3326_qgo产品目录新建，可以先直接拷贝rk3326_qgo为PX30_Android10，然后将PX30_Android10目录下的所有 `rk3326_qgo` 字符改为 `PX30_Android10`
具体修改如下图：



GMS认证

在Android 10.0中，如果需要通过GMS认证，需要将GMS包下载放到vendor目录下。

1. 通过RK的manifest (**推荐**, 一键部署, 包含RK修改, 减少GTS各种问题)

- 获取GMS包
  此时需要先联系FAE获取GMS包权限(**需要MADA**)，开通后，使用以下命令切换并重新同步工程：

```
.repo/repo/repo init -m Android10_Express.xml
.repo/repo/repo sync -c
```

- 打开配置GMS

```
device/rockchip/rkxxxx/BoardConfig.mk中配置:
# Google Service and frp overlay


BUILD_WITH_GOOGLE_MARKET := true
BUILD_WITH_GOOGLE_FRP := true
BUILD_WITH_GOOGLE_GMS_EXPRESS := false (此项做Express的客户设置为true, funding的客户请参考GMS的文档修改)
```

2. 自行从实验室或其他渠道获取GMS包 (**不推荐**)
   获取后，放置于vendor目录，此时vendor下的结构如下：

```
vendor$ ls
partner_gms  partner_modules (原有目录: rockchip  widevine )
```

- 打开配置GMS

```
自行配置, 需要编译mainline modules及GMS包app, RK提供整合包, 自行根据整合包修改GMS包进行适配:
RKDocs/android/patches/gms/rockchip_gms_*.tar.gz
```

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
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
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
RK3126C	RK816	DDR3	平板	样机	rk3126-m88
RK3368	RK808	LPDDR3	开发板	evb	rk3368-808-evb
RK3368	RK818	LPDDR3	平板	样机	rk3368-tablet
RK3368A	RK817	LPDDR3	平板	样机	rk3368a-817-tablet
RK3288W	RK808	LPDDR3	开发板	evb	rk3288-evb-android-rk808-edp-avb
RK3229		DDR3	开发板	evb	rk3229-evb-android-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\
```

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.pdf
```

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
```

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，确保各项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。
- Wifi 压力测试： 包括Wifi 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

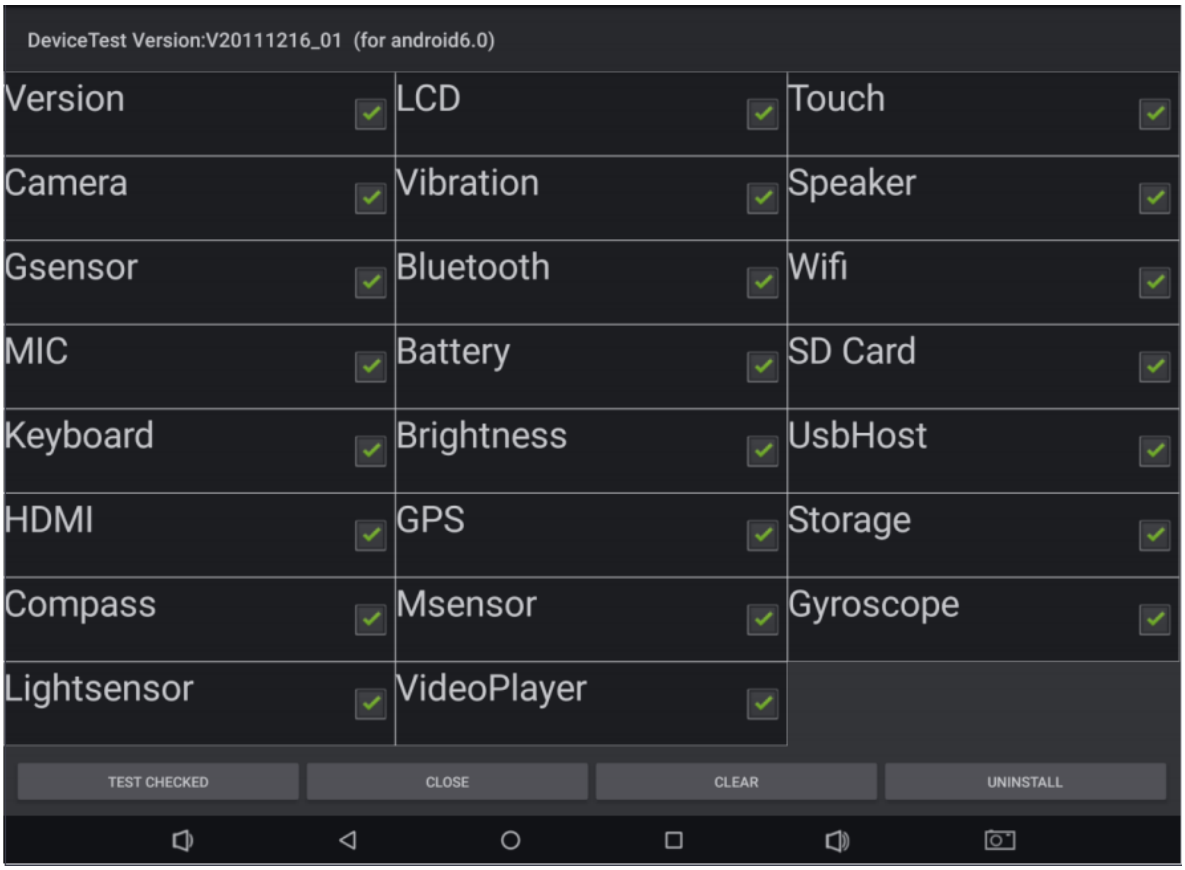
PCBA测试工具

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

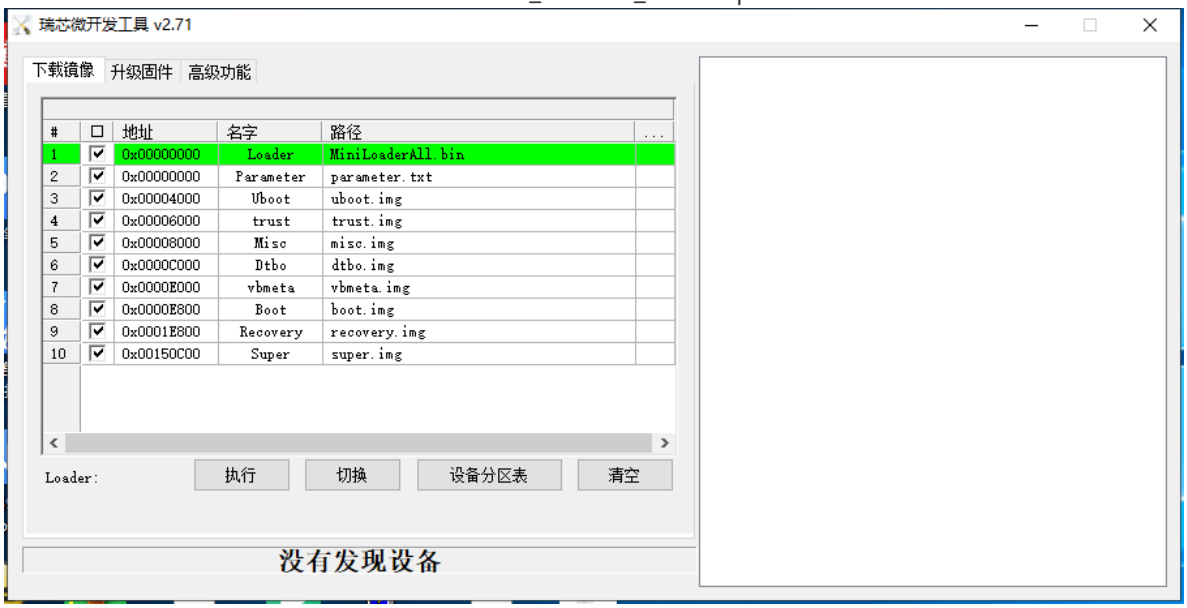
Rockchip USB驱动安装包，包括ADB、固件烧写驱动

```
RKTools\windows\DriverAssitant_v4.5.zip
```

开发烧写工具

Windows版本

RKTools/windows/AndroidTool/AndroidTool_Release_v2.71.zip



Linux版本

RKTools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool_v1.43.zip

```
Linux_Upgrade_Tool_v1.43$ 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
```

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?

```
Android10.0 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 Android10.0 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-10.0-mid-rkr2">
```

说明RK的版本是android-10.0-mid-rkr2

- AOSP部分

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

说明OASP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-10.0-mid-rkr1
android-10.0-mid-rkr2
develop-4.4-20190201
```

RK的版本是以android-10.0-mid-rkrxx的格式递增的，所以当前的最新tag是android-10.0-mid-rkr2

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。

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

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
```

A/B 系统配置

A/B系统就是设备上有A和B两套可以工作的系统，可以理解为一套系统分区，另外一套为备份分区。
A/B默认关闭，如需打开，在对应芯片的BoardConfig.mk中按如下配置打开（以RK3326为例）：

```
vim device/rockchip/rk3326_q/BoardConfig.mk
#AB image definition
-BOARD_USES_AB_IMAGE := false
+BOARD_USES_AB_IMAGE := true
```

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
```

RK3368 使能sd卡功能

串口与sd卡的GPIO复用，SDK默认enable fiq_debugger，disable sdmmc，若要是使用sd卡功能需要修改dts。对于RK3368 Android10.0 之前版本，只需要 disable fiq_debugger，enable sdmmc，但是对于RK3368 Android10.0系统该方法已经不适用，若disable fiq_debugger会导致系统无法启动。

代码修改

```
diff --git a/arch/arm64/boot/dts/rockchip/rk3368-808-evb.dts
b/arch/arm64/boot/dts/rockchip/rk3368-808-evb.dts
index e859624..a07f17e 100644
--- a/arch/arm64/boot/dts/rockchip/rk3368-808-evb.dts
+++ b/arch/arm64/boot/dts/rockchip/rk3368-808-evb.dts
@@ -36,6 +36,8 @@
 &fiq_debugger {
        status = "okay";
+       pinctrl-names = "";
+       pinctrl-0 = "";
 };

 &cif {
diff --git a/arch/arm64/boot/dts/rockchip/rk3368-808.dtsi
b/arch/arm64/boot/dts/rockchip/rk3368-808.dtsi
index 8a79416..44ead6e 100644
--- a/arch/arm64/boot/dts/rockchip/rk3368-808.dtsi
+++ b/arch/arm64/boot/dts/rockchip/rk3368-808.dtsi
@@ -409,7 +409,7 @@
        num-slots = <1>;
        pinctrl-names = "default";
        pinctrl-0 = <&sdmmc_clk &sdmmc_cmd &sdmmc_cd &sdmmc_bus4>;
-       status = "disabled";
+       status = "okay";
 };
```

SD卡启动功能

前提

- 机器没有EMMC或者NAND等存储设备，如果有则需要先擦除flash。
- SD卡功能需要打开，如果SD卡默认关闭的需要先在dts中配置打开，如RK3326。

代码修改

以RK3399为例

Android部分

```
device/rockchip/rk3399$
diff --git a/rk3399_Android10/rk3399_Android10.mk
b/rk3399_Android10/rk3399_Android10.mk
index d00e0a3..c031ecb 100755
--- a/rk3399_Android10/rk3399_Android10.mk
+++ b/rk3399_Android10/rk3399_Android10.mk
@@ -17,7 +17,7 @@
 # First lunching is Q, api_level is 29
 PRODUCT_SHIPPING_API_LEVEL := 29
 PRODUCT_DTBO_TEMPLATE := $(LOCAL_PATH)/dt-overlay.in
-PRODUCT_BOOT_DEVICE := fe330000.sdhci
+PRODUCT_BOOT_DEVICE := fe330000.sdhci,fe320000.dwmmc
 include device/rockchip/common/build/rockchip/DynamicPartitions.mk
 include device/rockchip/common/BoardConfig.mk
 $(call inherit-product, $(SRC_TARGET_DIR)/product/full_base.mk)
```

Kernel部分

dts的sdmmc节点加入supports-emmc字段，如下：

```
&sdmmc {
        clock-frequency = <150000000>;
        clock-freq-min-max = <100000 150000000>;
        supports-sd;
        bus-width = <4>;
        cap-mmc-highspeed;
        cap-sd-highspeed;
        disable-wp;
        supports-emmc;
        num-slots = <1>;
        sd-uhs-sdr104;
        vmmc-supply = <&vcc_sd>;
        vqmmc-supply = <&vccio_sd>;
        pinctrl-names = "default";
        pinctrl-0 = <&sdmmc_clk &sdmmc_cmd &sdmmc_cd &sdmmc_bus4>;
        status = "okay";
};
```

制作工具

```
SDDiskTool_v1.59
```

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RK3399_Android10为例说明:

```
device/rockchip/common$ git diff
diff --git a/scripts/fstab_tools/fstab.in b/scripts/fstab_tools/fstab.in
index 266531a..52453ea 100755
--- a/scripts/fstab_tools/fstab.in
+++ b/scripts/fstab_tools/fstab.in
@@ -16,6 +16,6 @@ ${_block_prefix}product /product  ext4 ro,barrier=1
${_flags},first_stage_mount
 # For sdmmc
 /devices/platform/${_sdmmc_device}/mmc_host*        auto  auto    defaults
   voldmanaged=sdcard1:auto,encryptable=userdata
 #  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,fsync_mode=nobarrier
latemount,wait,check,fileencryption=software,quota,formattable,reservedsize=128M
,checkpoint=fs
+#/dev/block/by-name/userdata /data f2fs
noatime,nosuid,nodev,discard,reserve_root=32768,resgid=1065,fsync_mode=nobarrier
latemount,wait,check,fileencryption=software,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
wait,formattable,check,fileencryption=software,quota,reservedsize=128M
+/dev/block/by-name/userdata    /data      ext4
discard,noatime,nosuid,nodev,noauto_da_alloc,data=ordered,user_xattr,barrier=1
wait,formattable,check,fileencryption=software,quota,reservedsize=128M
```

```
device/rockchip/rk3399$ git diff
--- a/device.mk
+++ b/device.mk
@@ -27,7 +27,7 @@ PRODUCT_PACKAGES += \
```

```
     libion

 #enable this for support f2fs with data partion
-BOARD_USERDATAIMAGE_FILE_SYSTEM_TYPE := f2fs
+BOARD_USERDATAIMAGE_FILE_SYSTEM_TYPE := ext4

 # used for fstab_generator, sdmmc controller address
 PRODUCT_SDMMC_DEVICE := fe320000.dwmmc
diff --git a/rk3399_Android10/recovery.fstab b/rk3399_Android10/recovery.fstab
index 7532217..cf789ac 100755
--- a/rk3399_Android10/recovery.fstab
+++ b/rk3399_Android10/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
```

root功能

root功能的补丁：

```
RKDocs/android/patches/box/rootservice_for_android10.rar
```

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。

从Android 9.0 OTA升级到Android 10.0

Android10.0 SDK 支持从Android 9.0版本通过OTA的方式升级到Android10.0，具体可以参考文档：

```
RKDocs\android\Rockchip_Introduction_OTA_from_Android9.0_to_Android10.0_CN&EN.pdf
```

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
  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。

分区修改

修改分区大小

分区大小在parameter文件中定义，具体位置在产品的目录下，如rk3326_qgo产品的parameter文件在如下路径：

```
device/rockchip/rk3326/rk3326_qgo/parameter.txt
```

parameter文件的介绍请参考文档

```
RKDocs/common/RKTools manuals/Rockchip-Parameter-File-Format-Version1.4-CN.pdf
```

增加分区

新加分区的步骤：

- parameter文件中加入分区大小和地址信息，参考上面的说明；
- fstab中加入分区挂在信息，fstab文件默认路径（有些产品在产品目录下）：

  ```
  device/rockchip/common/scripts/fstab_tools/fstab.in
  ```

- recovery.fstab中增加分区信息，recovery.fstab文件路径在具体的产品目录下，如rk3326_qgo:

  ```
  device/rockchip/rk3326/rk3326_qgo/recovery.fstab
  ```

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
```

PX30使用build.sh脚本编译update.img无法烧写问题

问题描述：PX30使用build.sh脚本编译update.img无法烧写，烧写工具提示检测芯片失败。
解决办法：

```
cd RKTools
diff --git a/linux/Linux_Pack_Firmware/rockdev/mkupdate_rk3326.sh
b/linux/Linux_Pack_Firmware/rockdev/mkupdate_rk3326.sh
index 7df28a9..722cd9d 100755
--- a/linux/Linux_Pack_Firmware/rockdev/mkupdate_rk3326.sh
+++ b/linux/Linux_Pack_Firmware/rockdev/mkupdate_rk3326.sh
@@ -15,7 +15,7 @@ if [ ! -f "package-file" ]; then
 #      pause
 fi
 ./afptool -pack ./ Image/update.img || pause
-./rkImageMaker -RK3326 Image/MiniLoaderAll.bin Image/update.img update.img -
os_type:androidos || pause
+./rkImageMaker -RKPX30 Image/MiniLoaderAll.bin Image/update.img update.img -
os_type:androidos || pause
```

```
 echo "Making update.img OK."
 #echo "Press any key to quit:"
 #read -n1 -s key
```

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

AVB的相关说明和配置可以参考文档

```
RKDocs/common/u-boot/Rockchip_Developer_Guide_UBoot_Nextdev_CN.pdf
```

如何关闭userdata分区的磁盘加密

磁盘加密的配置在fstab.in中，去掉对应的配置就可以，不同芯片可能fstab.in文件不一样，需要先确定当前使用的fstab，下面以3326的fstab为例说明：

```
diff --git a/rk3326_q/fstab.in b/rk3326_q/fstab.in
index 4d2828e..0e4ca04 100755
--- a/rk3326_q/fstab.in
+++ b/rk3326_q/fstab.in
@@ -15,4 +15,4 @@ ${_block_prefix}odm    /odm    ext4 ro,barrier=1
${_flags},first_stage_mount
 # For sdmmc
 /devices/platform/ff370000.dwmmc/mmc_host*        auto  auto    defaults
voldmanaged=sdcard1:auto,encryptable=userdata
 #  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,fsync_mode=nobarrier
latemount,wait,check,fileencryption=software,quota,formattable,reservedsize=128M
,checkpoint=fs
+/dev/block/by-name/userdata /data f2fs
noatime,nosuid,nodev,discard,reserve_root=32768,resgid=1065,fsync_mode=nobarrier
latemount,wait,check,quota,formattable,reservedsize=128M
```

RK3326/PX30使能SD卡功能

RK3326/PX30的SD卡功能由于与fiq的log打印功能GPIO冲突了，所以默认是配置为打开FIQ和关闭SD卡功能，如果要使能SD卡功能，可以参考如下修改：

- fiq-debugger节点设置为disabled

```
--- a/arch/arm64/boot/dts/rockchip/px30-android.dtsi
+++ b/arch/arm64/boot/dts/rockchip/px30-android.dtsi
@@ -19,7 +19,7 @@
                interrupts = <GIC_SPI 127 IRQ_TYPE_LEVEL_LOW>;
                pinctrl-names = "default";
                pinctrl-0 = <&uart2m0_xfer>;
-               status = "okay";
+               status = "disabled";
        };

        firmware {
```

- &sdmmc节点设置为okay

```
diff --git a/arch/arm64/boot/dts/rockchip/rk3326-863-lp3-v10.dtsi
b/arch/arm64/boot/dts/rockchip/rk3326-863-lp3-v10.dtsi
index b43d9c0..f1edca3 100644
--- a/arch/arm64/boot/dts/rockchip/rk3326-863-lp3-v10.dtsi
+++ b/arch/arm64/boot/dts/rockchip/rk3326-863-lp3-v10.dtsi
@@ -715,7 +715,7 @@
        sd-uhs-sdr104;
        vqmmc-supply = <&vccio_sd>;
        vmmc-supply = <&vcc_sd>;
-       status = "disabled";
+       status = "okay";
 };
```

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-10.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-10-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/q/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

配置显示只有HDMI或者DP，没有非热插拔屏（mipi或者edp等），系统无法开机问题

解决办法：
内核注册非热插拔的显示设备，如edp，同时配置hwc 主显设备为hdmi，即：
(1) dts里面把edp通路打开，包括edp、panel、以及panel中可能依赖的backlight；
(2) 关闭edp uboot 显示，避免uboot中对edp 通路的操作；
(3) 配置vendor.hwc.device.primary 为HDMI-A；
加上以上修改后，只是保证驱动加载过程中有一个确认可用的分辨率用于内核框架注册fb设备，保证hwc正常加载，同时由于配置了hwc的primary 属性，可以保证系统起来后只有HDMI 显示，同时不会产生任何额外的功耗及性能开销；

SN号的命令规则

SN号必须以字母开头，长度14个字节以内。