Rockchip Android14 GKI开发指南

文件标识：RK-KF-YF-775

发布版本：V1.1.0

日期：2024-04-15

文件密级：□绝密 □秘密 □内部资料 ■公开

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

概述

本文介绍Android14的GKI开发的流程和注意点。

读者对象

本文档（本指南）主要适用于以下工程师：

技术支持工程师

软件开发工程师

修订记录

版本号	作者	修改日期	修改说明
V1.0.0	吴良清	2023-11-15	初始版本
V1.1.0	吴良清	2024-04-15	修改GKI编译方式，不再预编译KO

目录

Rockchip Android14 GKI开发指南

1 GKI介绍
- 1.1 什么是GKI
- 1.2 什么产品需要使用GKI
- 1.3 GKI和非GKI的差别

2 Rockchip Android14 GKI的适配情况

3 Google upstream kernel下载及编译

4 Rockchip SDK中GKI相关目录介绍

5 GKI编译环境要求

6 Rockchip GKI编译
- 6.1 代码修改
- 6.2 编译
- 6.3 固件烧写

7 KO编译及修改
- 7.1 添加新的模块驱动的方法

8 开机log确认
- 8.1 uboot阶段
- 8.2 Android阶段
- 8.3 KO加载
- 8.4 KO加载报错
- 7.5 bootcmdline解析出错
- 8.6 Mali KO加载失败
- 8.7 编译kernel报错

9 调试技巧
- 9.1 打印更多KO加载的log
- 9.2 在RK的kernel打包中编译GKI使用的boot.img
- 9.3 查看google发布的内核接口

10 如何提交内核接口到upstream

11 如何更新AOSP发布的boot.img

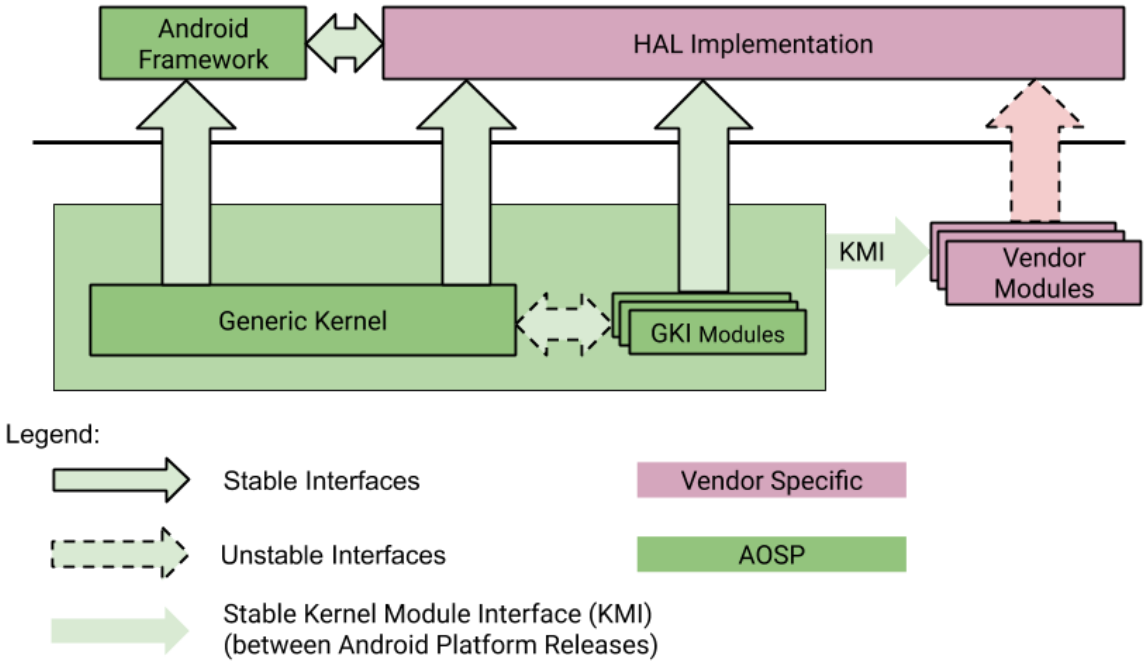
12 如何单独打包vendor_boot.img
- 步骤一：在kernel中编译KO
- 步骤二：拷贝KO文件到mkcombinedroot目录下
- 步骤三：拷贝vendor_boot.img到mkcombinedroot目录下
- 步骤四：进到mkcombinedroot目录下执行mkgki4.sh脚本更新ko并编译到vendor_boot.img中
- 步骤五： 烧写new_vendor_boot.img到机器中

1 GKI介绍

1.1 什么是GKI

GKI：Generic Kernel Image 通用内核映像。
Android14 GMS和EDLA认证的一个难点是google强制要求要支持GKI。GKI通用内核映像，是google为了解决内核碎片化的问题，而设计的通过提供统一核心内核并将SoC和板级驱动从核心内核移至可加载模块中。核心内核为驱动模块提供了稳定的内核模块接口，模块驱动和核心内核可以独立进行更新。内

核接口可以通过upstream的方式进行扩展。 Soc和板级厂商在驱动开发时需要使用已经定义的内核接口，如果要新加核心内核接口需要提交给google，这个周期会比较长，所以要提前做好开发准备。



1.2 什么产品需要使用GKI

- 使用Android14且需要过GMS认证和EDLA认证的产品
- 使用Android13且需要过GMS认证和EDLA认证的产品
- 使用Android12 的RK3588和RK3588S的需要过GMS认证和EDLA认证的产品
- 不过GMS认证和EDLA认证的产品不强制要求使用GKI

1.3 GKI和非GKI的差别

- 通用内核boot.img

GKI	非GKI
由google定期发布boot.img，代码不能自己修改	由RK提供内核源码编译，可以自由修改

- 驱动模块

GKI	非GKI
以KO的形式加载，调用的内核接口必需是google发布的boot.img里面包含的	内嵌在boot中，由RK提供内核源码编译，可以自由修改和添加内核接口

- kernel代码

GKI	非GKI
RK发布的kernel源码仅用于编译驱动模块的KO	RK发布的kernel源码用于完整的内核和模块驱动的编译，模块以.o的形式内嵌编译

- uboot支持head4
- 分区差异
  GKI增加vendor_boot、init_boot、resource分区

- 启用AB分区

2 Rockchip Android14 GKI的适配情况

kernel版本是6.1

芯片	是否完成适配
RK3562	已适配
RK3568	已适配
RK3566	已适配
RK3588	已适配
RK3588S	已适配
RK3326	已适配
PX30	已适配
RK3399	已适配
RK3576	已适配

3 Google upstream kernel下载及编译

Google的boot.img是定期发布，时间间隔比较长。 我们可以下载google的upstream的kernel源码自己编译boot.img进行验证和debug。

Google Upstream kernel下载链接：

```
repo init -u https://android.googlesource.com/kernel/manifest -b common-
android14-6.1
```

需要链接google服务器下载

编译

```
tools/bazel run //common:kernel_aarch64_dist -- --dist_dir=out
```

生成boot.img

```
out/boot.img
```

4 Rockchip SDK中GKI相关目录介绍

- kernel KO文件路径

```
mkcombinedroot/vendor_ramdisk/lib/modules/
```

- Google boot.img路径

```
mkcombinedroot/prebuilts/boot-6.1.img
```

- Android AOSP 发布的受保护的KO文件路径

```
kernel/prebuilts/6.1/arm64/
```

- Kernel-6.1源码编译的KO文件kernel启动加载的顺序配置文件

```
mkcombinedroot/res/vendor_ramdisk_modules.load
```

- Android Init阶段加载的KO文件加载顺序配置文件
  ...
  mkcombinedroot/res/vendor_modules.load
  ...

5 GKI编译环境要求

- Ubuntu版本需要20.04及以上版本
- pahole版本需要1.25版本
-

6 Rockchip GKI编译

6.1 代码修改

- Android的device产品目录下配置GKI选项

```
~/a2_Android14_sdk/device/rockchip/rk3562$ git diff
diff --git a/rk3562_u/BoardConfig.mk b/rk3562_u/BoardConfig.mk
old mode 100644
new mode 100755
index 50da541..06da5f3
--- a/rk3562_u/BoardConfig.mk
+++ b/rk3562_u/BoardConfig.mk
@@ -15,10 +15,21 @@
 #
 include device/rockchip/rk3562/BoardConfig.mk
 BUILD_WITH_GO_OPT := false
-BOARD_BUILD_GKI := fasle
+BOARD_BUILD_GKI := true
```

注：RK3562_UGO的配置已经默认打开GKI的配置，不需要额外配置

如果单独编译uboot代码需要修改config文件打开AB配置，如果是用build.sh脚本完整编译则不需要修改，编译的时候会自动添加AB的宏配置。

- uboot需要打开AB配置

```
~/a2_Android13_sdk/u-boot$ git diff
diff --git a/configs/rk3568_defconfig b/configs/rk3568_defconfig
index fbd9820acc..e23e438792 100644
--- a/configs/rk3588_defconfig
+++ b/configs/rk3588_defconfig
```

```
@@ -207,6 +207,7 @@ CONFIG_RSA_N_SIZE=0x200
CONFIG_RSA_E_SIZE=0x10
CONFIG_RSA_C_SIZE=0x20
CONFIG_SHA512=y
CONFIG_LZ4=y
CONFIG_LZMA=y
CONFIG_SPL_GZIP=y
@@ -220,3 +221,4 @@ CONFIG_RK_AVB_LIBAVB_USER=y
CONFIG_OPTEE_CLIENT=y
CONFIG_OPTEE_V2=y
CONFIG_OPTEE_ALWAYS_USE_SECURITY_PARTITION=y
+CONFIG_ANDROID_AB=y
```

6.2 编译

完整编译方式与非GKI的一样

```
source  build/envsetup.sh
lunch rk3562_ugo-userdebug
./build.sh -ACUKup
```

注意：这里编译的kernel只是为了编译出resource.img，kernel源码会编译成ko文件打包成vendor_boot.img。内核部分使用的是google发布的boot.img，具体路径在mkcombinedroot/prebuilts/boot-6.1.img

编译完可以直接烧写 `rockdev/Image-rk3562_ugo/update.img`

在调试阶段也支持单独编译vendor_boot.img
编译命令：

```
make installclean;make vendorbootimage -j12
```

编译完可以直接烧写

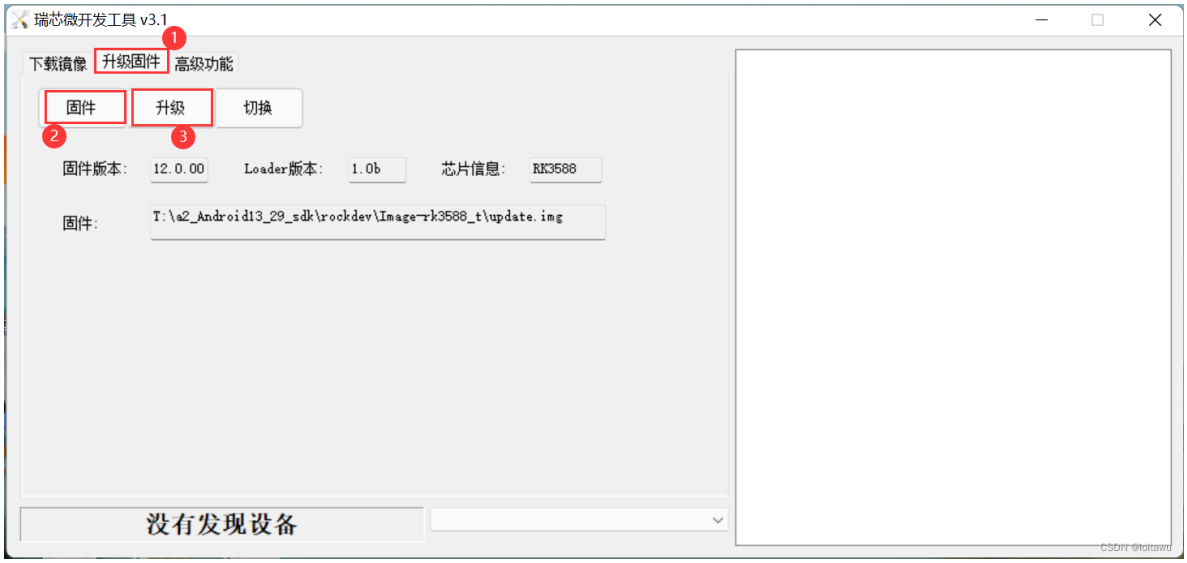
```
out/target/product/rk3562_ugo/vendor_boot.img
```

6.3 固件烧写

固件烧写分2中方式：

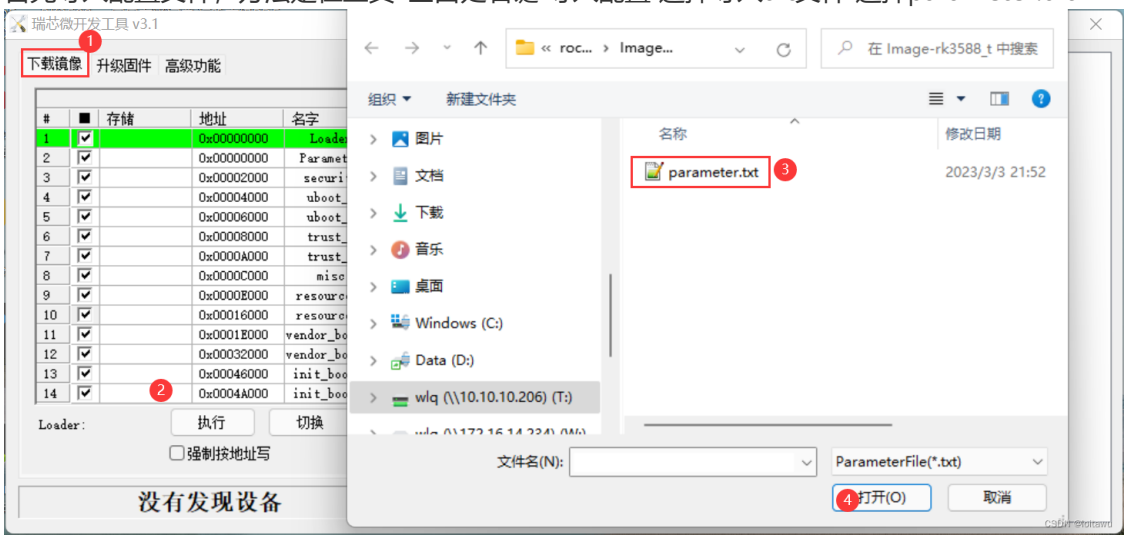
- 完整包update.img
  固件路径

```
rockdev/Image-rk3562_ugo/update.img
```



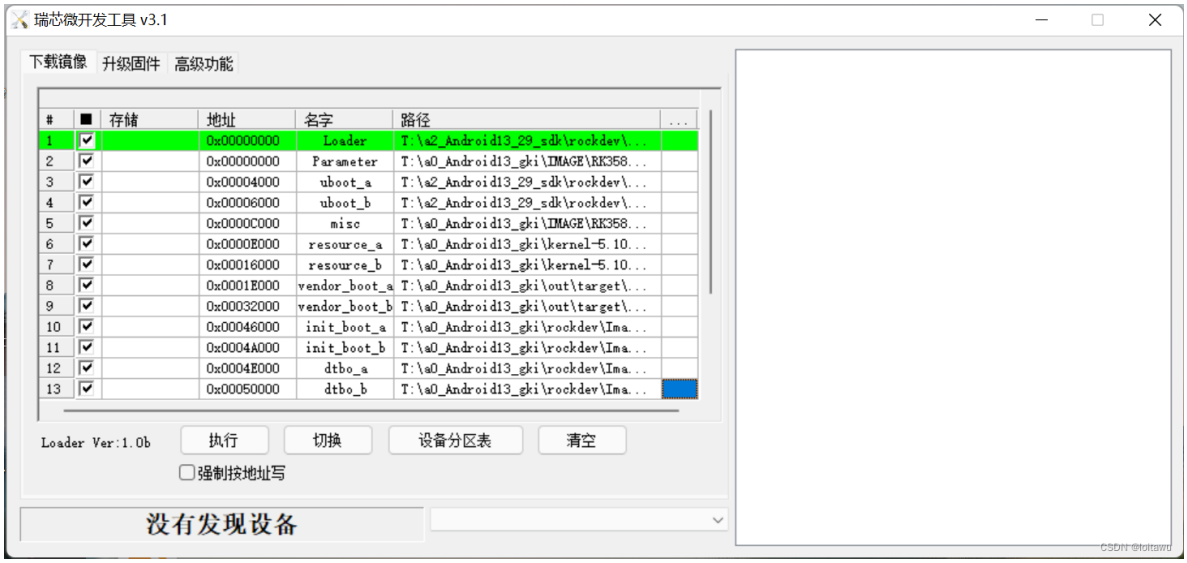
可以通过瑞芯微开发工具烧写

- 散包烧写
  首先导入配置文件，方法是在工具 空白处右键-导入配置-选择导入txt文件-选择parameter.txt



然后依次选择rockdev/Image-rk3588_t/下对应的img文件进行烧写，分区A和B导入的固件是同一个

```
rockdev/Image-rk3562_ugo
├── baseparameter.img
├── boot.img
├── dtbo.img
├── init_boot.img
├── MiniLoaderAll.bin
├── misc.img
├── parameter.txt
├── resource.img
├── super.img
├── uboot.img
├── update.img
├── vbmeta.img
└── vendor_boot.img
```



7 KO编译及修改

7.1 添加新的模块驱动的方法

7.1.1 将驱动代码放到kernel-6.1对应的目录下，这里以新加触摸屏驱动gt1x为例进行说明。

将gt1x的驱动放在`drivers/input/touchscreen/`下面，并添加对应的`Makefile`和`Kconfig`，这里按kernel的标准方式进行操作；

7.1.2 增加一个自己的config文件， 在`arch/arm64/configs/`下新建一个`xxx_gki.config`，并将`CONFIG_TOUCHSCREEN_GT1X=m` (m表示编译为ko)添加到`xxx_gki.config`中；

7.1.3 将ko文件名添加到mkcombinedroot/res/vendor_ramdisk_modules.load或者mkcombinedroot/res/vendor_modules.load

`.load`文件名称	对应分区	makefile解析	加载时间
vendor_ramdisk_modules.load	vendor_boot	vendor_ramdisk_gki.mk	ramdisk init阶段
vendor_modules.load	vendor	vendor_gki.mk	android启动时
recovery_modules.load	recovery	recovery_gki.mk	recovery阶段

如果驱动对加载时间没有要求的话可以放在Android阶段去加载，比如触摸屏的驱动、sensor驱动等等，具体修改如下：

- 进到mkcombinedroot目录

```
cd mkcombinedroot
```

- 在res/vendor_modules.load中添加需要编译到vendor的ko名字，如xxx_tp.ko

```
diff --git a/res/vendor_modules.load b/res/vendor_modules.load
index e69de29..a53449f 100644
--- a/res/vendor_modules.load
+++ b/res/vendor_modules.load
@@ -0,0 +1,4 @@
pcie-dw-rockchip.ko
cfg80211.ko
+xxx_tp.ko
```

- 添加完成后编译烧写super.img即可。
  添加到res/vendor_modules.load中的ko编译的时候会编译到vendor_dlkm/lib/modules/目录下，所以在调试的时候可以直接将编译好的ko文件push到机器的vendor_dlkm/lib/modules/目录下，然后重启机器即可在开机的时候自动加载。
  注意： mkcombinedroot/res/vendor_ramdisk_modules.load文件关乎驱动的加载顺序，请不要修改原有顺序，否则可能会导致系统无法启动！！！新加的ko文件非必要不建议放在这里面，vendor_ramdisk_modules.load加载ko越多系统启动的速度越慢。 建议新加的ko文件都放在mkcombinedroot/res/vendor_modules.load中，这个是在android启动阶段加载的，对开机时间影响比较小。

7.1.4 编译
kernel的修改建议使用build.sh -K脚本进行编译，因为build.sh -K 中编译kernel的时候会自动将ko文件拷贝到mkcombinedroot/vendor_ramdisk/lib/modules临时目录下，在完整编译的会从这个目录打包ko文件到vendor_boot.img或者super.img中。
如果是在kernel目录下单独编译kernel，编译生产的ko文件不会自动拷贝到mkcombinedroot/vendor_ramdisk/lib/modules，需要手动进入mkcombinedroot目录执行./copy_modules.sh脚本进行拷贝。

7.1.5 编译vendor_boot.img
在工程根目录下编译vendor_boot.img，命令如下。 这一步是将KO文件打包到vendor_boot.img，在将vendor_boot.img烧写到机器中。

```
make installclean;make vendorbootimage -j12
```

单独烧写vendor_boot.img，编译完的vendor_boot.img路径如下：

```
out/target/product/rk3562_ugo/vendor_boot.img
```

注意：如果是编译到vendor的ko需要完整编译super.img并烧写super.img
7.1.6 验证

- 烧写`out/target/product/rk3562_ugo/vendor_boot.img`文件到机器中开机验证
- 如果是放在vendor分区的ko可以在系统起来后直接push到机器内的vendor分区中，手动挂载进行验证
- 如果有涉及到dts的修改，需要烧写kernel-6.1下的`resource.img`

附：AOSP定义的各类ko加载阶段

Boot mode	Storage	Display	Keypad	Battery	PMIC	TP	NFC/Wi-Fi/BT	Sensors	Camera
Recovery	Y	Y	Y	Y	Y	N	N	N	N
Charger	Y	Y	Y	Y	Y	N	N	N	N
Android	Y	Y	Y	Y	Y	Y	Y	Y	Y

8 开机log确认

8.1 uboot阶段

内容	header版本
vendor_ramdisk(v-ramdisk)	V3+
bootconfig	V4+

```
## Booting Android Image at 0x003ff000 ...
Kernel: 0x00400000 - 0x03088ffc (45604 KiB)
v-ramdisk:  0x0a200000 - 0x0a6944c8 (4690 KiB)
ramdisk:    0x0a6944c8 - 0x0a7e54df (1349 KiB)
bootconfig: 0x0a7e54df - 0x0a7e559c (1 KiB)
bootparams: 0x0a7e559c - 0x0a7e759c
```

8.2 Android阶段

GKI版本：`Linux version 5.10.117-android13-9-00037-gbc08447eb7bd`

```
[    0.000000][    T0] Booting Linux on physical CPU 0x0000000000 [0x412fd050]
[    0.000000][    T0] Linux version 5.10.117-android12-9-00037-gbc08447eb7bd
(build-user@build-host) (Android (7284624, based on r416183b) clang version
12.0.5 (https://android.googlesource.com/toolchain/llvm-project
c935d99d7cf2016289302412d708641d52d2f7ee), LLD 12.0.5
(/buildbot/src/android/llvm-toolchai
n/out/llvm-project/lld c935d99d7cf2016289302412d708641d52d2f7ee)) #1 SMP PREEMPT
Thu Aug 25 15:24:20 UTC 2022
```

Kernel command line： Header V4中不能存在androidboot.xxx这一类的命令行参数，这类参数全部在bootconfig中。此类参数可以通过`cat /proc/bootconfig`确认。

```
[    0.000000][    T0] Kernel command line: stack_depot_disable=on
kasan.stacktrace=off kvm-arm.mode=protected cgroup_disable=pressure
cgroup.memory=nokme
m storagemedia=emmc console=ttyFIQ0 firmware_class.path=/vendor/etc/firmware
init=/init rootwait ro loop.max_part=7 bootconfig buildvariant=userdebug earl
ycon=uart8250,mmio32,0xfeb50000 irqchip.gicv3_pseudo_nmi=0
```

8.3 KO加载

开始加载ko，可以看到log：

```
[    1.034730][    T1] Run /init as init process
[    1.036190][    T1] init: init first stage started!
[    1.040534][    T1] init: Loading module /lib/modules/io-domain.ko with args
''
[    1.042038][    T1] init: Loaded kernel module /lib/modules/io-domain.ko
```

8.4 KO加载报错

使用了未导出的符号，报错重启：

```
[    0.805736][    T1] cryptodev: Unknown symbol crypto_ahash_final (err -2)
[    0.806383][    T1] cryptodev: Unknown symbol sg_nents (err -2)
[    0.806972][    T1] cryptodev: Unknown symbol crypto_alloc_akcipher (err -2)
[    0.819768][    T1] Kernel panic - not syncing: Attempted to kill init!
exitcode=0x00007f00
```

这个log表示crypto_ahash_final、sg_nents、crypto_alloc_akcipher这些符号在boot中没有导出，无法使用。
解决方法：

- 更换其它接口，boot里面有导出的接口可以在kernel-6.1/android目录下搜索
- 一定要使用该接口的，需要添加对应的接口并提交给google，具体方法可以参考本文中的"如何提交内核接口到upstream"章节。

```
[    1.825070][    T1] init: Failed to insmod '/lib/modules/gc05a2.ko' with args
'': Exec format error
[    1.825094][    T1] init: LoadWithAliases was unable to load gc05a2
[    1.826118][    T1] init: Failed to load kernel modules
[    1.827032][    T1] init: InitFatalReboot: signal 6
[    1.836535][    T1] init:   #00 pc 0000000000031fe7c  /init
(unwindstack::AndroidLocalUnwinder::InternalUnwind(std::__1::optional<int>,
unwindstack::AndroidUnwinderData&)+92) (BuildId:
aec5ebfd90cec4000c74ac83c809ae14)
[    1.836603][    T1] init:   #01 pc 0000000000031444c  /init
(android::init::InitFatalReboot(int)+204) (BuildId:
aec5ebfd90cec4000c74ac83c809ae14)
[    1.836638][    T1] init:   #02 pc 00000000000314b00  /init
(android::init::InstallRebootSignalHandlers()::$_0::__invoke(int)+32) (BuildId:
aec5ebfd90cec4000c74ac83c809ae14)
[    1.836658][    T1] init:   #03 pc 0000000000000860  [vdso]
[    1.836685][    T1] init:   #04 pc 00000000004ddfb0  /init (abort+176)
(BuildId: aec5ebfd90cec4000c74ac83c809ae14)
[    1.836719][    T1] init:   #05 pc 00000000000319dac  /init
(android::init::InitAborter(char const*)+44) (BuildId:
aec5ebfd90cec4000c74ac83c809ae14)
[    1.836760][    T1] init:   #06 pc 0000000000049eed0  /init
(android::base::SetAborter(std::__1::function<void (char
const*)>&&)::$_0::__invoke(char const*)+80) (BuildId:
aec5ebfd90cec4000c74ac83c809ae14)
[    1.836793][    T1] init:   #07 pc 0000000000049e664  /init
(android::base::LogMessage::~LogMessage()+356) (BuildId:
aec5ebfd90cec4000c74ac83c809ae14)
[    1.836824][    T1] init:   #08 pc 00000000030adc8  /init
(android::init::FirstStageMain(int, char**)+8696) (BuildId:
aec5ebfd90cec4000c74ac83c809ae14)
[    1.836870][    T1] init:   #09 pc 00000000004dbd9c  /init
(__real_libc_init(void*, void (*)(), int (*)(int, char**, char**),
structors_array_t const*, bionic_tcb*)+716) (BuildId:
aec5ebfd90cec4000c74ac83c809ae14)
[    1.836887][    T1] init: Reboot ending, jumping to kernel
```

init: Failed to insmod '/lib/modules/gc05a2.ko' with args '': Exec format error 这个错误是编译ko的环境不匹配导致，一般是pahole的版本不对的原因，要求pahole的版本是V1.25，可以通过如下命令查询：

```
@sys2206:~/pahole$ pahole --version
v1.25
```

可以按下面步骤更新pahole版本

- 下载最新版本pahole
  `git clone https://git.kernel.org/pub/scm/devel/pahole/pahole.git`
- 编译pahole

安装编译依赖库

```
sudo apt-get install cmake
sudo apt-get install libdw-dev
```

如果之前有安装过pahole，需要先卸载

```
sudo apt-get --purge remove dwarves
```

- 开始编译
  ahole目录下执行

```
mkdir build
cd build/
cmake -D__LIB=lib -DBUILD_SHARED_LIBS=OFF ..
sudo make install
```

pahole --version 查看版本确认是否安装成功

如果以上安装编译失败可以直接下载我们编译好的pahole，并替换到 /usr/local/bin/pahole
链接：https://pan.baidu.com/s/1JP1F0EjzSn25ZVUsbO89Zg
提取码：zy6z

7.5 bootcmdline解析出错

错误log

```
Failed to parse bootconfig: Value is redefined at 416.
```

现象：无法开机或者开机进到recovery
原因：cmdline中的字段重复了，导致解析cmdline出错，可以在开机到uboot的时候在串口按住crtl+p就会打印所有的cmdline信息，从打印的cmdline信息中检查哪个字段重复了，然后去代码里面找对应的定义的位置删除对应的字段即可。cmdline是在device和kernel的dts中定义，可以在这两个目录下搜索该重复的字段即可。

8.6 Mali KO加载失败

Mali KO加载失败表现为无法开机界面显示卡在'Rockchip kernel'的logo，logcat可以看到surfaceflinger crash。

```
04-27 22:45:27.653   366   366 F DEBUG   : *** *** *** *** *** *** *** *** ***
*** *** *** *** *** *** ***
```

```
04-27 22:45:27.653   366   366 F DEBUG   : Build fingerprint:
'rockchip/rk3562_t/rk3562_t:13/TQ2A.230305.008.F1/eng.wlq.20230427.101925:userde
bug/release-keys'
04-27 22:45:27.653   366   366 F DEBUG   : Revision: '0'
04-27 22:45:27.653   366   366 F DEBUG   : ABI: 'arm64'
04-27 22:45:27.653   366   366 F DEBUG   : Timestamp: 2023-04-27
22:45:27.509738048+0000
04-27 22:45:27.653   366   366 F DEBUG   : Process uptime: 2s
04-27 22:45:27.653   366   366 F DEBUG   : Cmdline: /system/bin/surfaceflinger
04-27 22:45:27.653   366   366 F DEBUG   : pid: 335, tid: 360, name:
surfaceflinger  >>> /system/bin/surfaceflinger <<<
04-27 22:45:27.653   366   366 F DEBUG   : uid: 1000
04-27 22:45:27.653   366   366 F DEBUG   : tagged_addr_ctrl: 0000000000000001
(PR_TAGGED_ADDR_ENABLE)
04-27 22:45:27.653   366   366 F DEBUG   : signal 6 (SIGABRT), code -1
(SI_QUEUE), fault addr --------
04-27 22:45:27.653   366   366 F DEBUG   : Abort message: 'no suitable EGLConfig
found, giving up'
04-27 22:45:27.653   366   366 F DEBUG   :     x0  0000000000000000  x1
0000000000000168  x2  0000000000000006  x3  0000000710899d340
04-27 22:45:27.654   366   366 F DEBUG   :     x4  7568661f2b636d74  x5
7568661f2b636d74  x6  7568661f2b636d74  x7  7f7f7f7f7f7f7f7f
04-27 22:45:27.654   366   366 F DEBUG   :     x8  00000000000000f0  x9
0000000739bcbda00  x10 0000000000000001  x11 0000000739bcff6a0
04-27 22:45:27.654   366   366 F DEBUG   :     x12 0000000710899d310  x13
0000000000000027  x14 0000000710899d4e0  x15 00000000197b1a4f
04-27 22:45:27.654   366   366 F DEBUG   :     x16 0000000739bd6dd58  x17
0000000739bd48770  x18 0000007108812000  x19 00000000000000ac
04-27 22:45:27.654   366   366 F DEBUG   :     x20 00000000000000b2  x21
000000000000014f  x22 0000000000000168  x23 00000000ffffffff
04-27 22:45:27.654   366   366 F DEBUG   :     x24 b4000071bbca60b0  x25
0000000710899dcb0  x26 0000000710899dff8  x27 00000000000fe000
04-27 22:45:27.654   366   366 F DEBUG   :     x28 0000000710899daf0  x29
0000000710899d3c0
04-27 22:45:27.654   366   366 F DEBUG   :     lr  0000000739bcef3f4  sp
00000071089ndroid.runtime/lib64/bionic/libc.so (__pthread_start(void*)+208)
(BuildId: e2429c64ab29f2d0ffc5a8f42c0c1b80)
04-27 22:45:27.655   366   366 F DEBUG   :       #09 pc 0000000000054c50
/apex/com.android.runtime/lib64/bionic/libc.so (__start_thread+64) (BuildId:
e2429c64ab29f2d0ffc5a8f42c0c1b80)
```

这个是因为GPU的ko不匹配，需要重新编译GPU的ko文件，并拷贝到vendor/rockchip/common/gpu下面对应的目录中，具体编译方法如下：
修改device下面的产品目录中kernel config配置：PRODUCT_KERNEL_CONFIG := gki_defconfig rockchip_gki.config增加对应芯片的gpu配置，具体如下

```
RK3588：
PRODUCT_KERNEL_CONFIG := gki_defconfig rockchip_gki.config
RK356X/RK3562：
PRODUCT_KERNEL_CONFIG := gki_defconfig rockchip_gki.config rk356x.config
RK3326/RK3326-S：
PX30/PX30-S：
PRODUCT_KERNEL_CONFIG := gki_defconfig rockchip_gki.config rk3326.config
RK3399：
PRODUCT_KERNEL_CONFIG := gki_defconfig rockchip_gki.config rk3399.config

rk3399.config需要按如下修改：
```

```
wlq@sys2_206:~/a0_Android13_gki/mkcombinedroot$ git diff configs/
diff --git a/configs/rk3399.config b/configs/rk3399.config
old mode 100644
new mode 100755
index 0d66674..a003ba5
--- a/configs/rk3399.config
+++ b/configs/rk3399.config
@@ -1,4 +1,11 @@
-CONFIG_MALI_MIDGARD=y
+CONFIG_MALI_MIDGARD=m
+CONFIG_MALI_PLATFORM_THIRDPARTY_NAME="rk"
+CONFIG_MALI_PLATFORM_THIRDPARTY=y
+CONFIG_MALI_DEBUG=y
+CONFIG_MALI_DEVFREQ=y
+CONFIG_MALI_DT=y
+CONFIG_MALI_EXPERT=y
+CONFIG_MALI_SHARED_INTERRUPTS=y
```

然后执行./build.sh -CK 编译kernel
编译完成后拷贝对应的mali ko到vendor下面，具体路径可以看上面的章节。

8.7 编译kernel报错

编译错误log：

```
BTF     .btf.vmlinux.bin.o
Segmentation fault (core dumped)
LD      .tmp_vmlinux.kallsyms1
KSYMS   .tmp_vmlinux.kallsyms1.S
AS      .tmp_vmlinux.kallsyms1.S
LD      .tmp_vmlinux.kallsyms2
KSYMS   .tmp_vmlinux.kallsyms2.S
AS      .tmp_vmlinux.kallsyms2.S
LD      vmlinux
BTFIDS  vmlinux
FAILED: load BTF from vmlinux: Unknown error -22Makefile:1293: recipe for target
'vmlinux' failed
make[1]: *** [vmlinux] Error 255
arch/arm64/Makefile:214: recipe for target 'rk3588-evb1-lp4-v10.img' failed
make: *** [rk3588-evb1-lp4-v10.img] Error 2
failed to build some targets (21 seconds)
```

可以按下面步骤更新pahole版本

- 下载最新版本pahole
  `git clone https://git.kernel.org/pub/scm/devel/pahole/pahole.git`
- 编译pahole

安装编译依赖库

```
sudo apt-get install cmake
sudo apt-get install libdw-dev
```

如果之前有安装过pahole，需要先卸载

```
sudo apt-get --purge remove dwarves
```

- 开始编译
  ahole目录下执行

```
mkdir build
cd build/
cmake -D__LIB=lib -DBUILD_SHARED_LIBS=OFF ..
sudo make install
```

pahole --version 查看版本确认是否安装成功

如果以上安装编译失败可以直接下载我们编译好的pahole，并替换到 /usr/local/bin/pahole
链接：https://pan.baidu.com/s/1JP1F0EjzSn25ZVUsbO89Zg
提取码：zy6z

9 调试技巧

9.1 打印更多KO加载的log

修改ratelimit的值，可以打印更多init的log，方便查问题，init信息太少会把ko加载的报错信息隐藏掉。

```
xxx@sys2_206:~/a0_Android13_gki/device/rockchip/common$ vim BoardConfig.mk
xxx@sys2_206:~/a0_Android13_gki/device/rockchip/common$ git diff
diff --git a/BoardConfig.mk b/BoardConfig.mk
index 0d1c886..1761ed0 100755
--- a/BoardConfig.mk
+++ b/BoardConfig.mk
@@ -392,3 +392,5 @@ ifeq ($(strip $(BOARD_BASEPARAMETER_SUPPORT)), true)
     endif
         BOARD_WITH_SPECIAL_PARTITIONS := baseparameter:1M
 endif
+
+BOARD_KERNEL_CMDLINE += printk.devkmsg=on
```

9.2 在RK的kernel打包中编译GKI使用的boot.img

先按正常编译步骤编译kernel，生成arch/arm64/boot/Image
用如下命令打包boot.img
mkbootimg --kernel arch/arm64/boot/Image --header_version 4 --output ../mkcombinedroot/prebuilts/boot-6.1.img

9.3 查看google发布的内核接口

标准的内核接口定义在android目录下：

```
:~/a5_google_kenrel/common$ tree a
android/ arch/
wlq@sys2_206:~/a5_google_kenrel/common$ tree android/
android/
├── abi_gki_aarch64
├── abi_gki_aarch64_core
```

```
├── abi_gki_aarch64_db845c
├── abi_gki_aarch64_exynos
├── abi_gki_aarch64_fips140
├── abi_gki_aarch64_galaxy
├── abi_gki_aarch64_generic
├── abi_gki_aarch64_hikey960
├── abi_gki_aarch64_rockchip
├── abi_gki_aarch64_type_visibility
├── abi_gki_aarch64_virtual_device
├── abi_gki_aarch64.xml
├── abi_gki_modules_exports
├── abi_gki_modules_protected
├── gki_aarch64_fips140_modules
├── gki_aarch64_modules
└── gki_system_dlkm_modules
```

10 如何提交内核接口到upstream

如果需要添加新的内核接口，可以生成对应的patch，再将patch通过瑞芯微的redmine系统提交给瑞芯微审核然后再统一提交给google

```
diff --git a/android/abi_gki_aarch64_rockchip b/android/abi_gki_aarch64_rockchip
index 85bd8bc134cf..3344cf064e06 100644
--- a/android/abi_gki_aarch64_rockchip
+++ b/android/abi_gki_aarch64_rockchip
@@ -2144,6 +2144,15 @@
   mmc_pwrseq_register
   mmc_pwrseq_unregister

+# required by r8168.ko
+  pci_set_mwi
+  pci_clear_mwi
+  proc_get_parent_data
+  skb_checksum_help
+  __skb_gso_segment
+  remove_proc_subtree
+  pci_choose_state
+
 # required by reboot-mode.ko
   devres_release
   kernel_kobj
```

11 如何更新AOSP发布的boot.img

Android AOSP会定期更新最新的boot.img及对应的受保护的KO文件，Android release的链接如下：
https://source.android.com/docs/core/architecture/kernel/gki-android14-6_1-release-builds
打开链接后找到Android14-6.1的最新release版本，然后点对应版本的boot-6.1.img下载，如图

Release build | Debug build

Release date	Tag / Source / Licenses	Kernel artifacts	Certified GKI
2023-10-31	android14-6.1-2023-10_r1 SHA1: 835b6458fa548d62264f LICENSES	kernel	boot-6.1.img boot-6.1-gz.img boot-6.1-lz4.img
2023-10-31	android14-6.1-2023-10_r2 SHA1: fffe3966fa735fa5b94b LICENSES	kernel	boot-6.1.img boot-6.1-gz.img boot-6.1-lz4.img

boot-6.1.img下载后拷贝到

```
mkcombinedroot/prebuilts/boot-6.1.img
```

Android14开始AOSP会同步发布受保护的ko模块，需要一起下载，这个ko文件需要核boot.img匹配才能正确加载。点击如下图的kernel链接下载system_dlkm_staging_archive.tar.gz

Release build | Debug build

Release date	Tag / Source / Licenses	Kernel artifacts	Certified GKI
2023-10-31	android14-6.1-2023-10_r1 SHA1: 835b6458fa548d62264f LICENSES	kernel	boot-6.1.img boot-6.1-gz.img boot-6.1-lz4.img
2023-10-31	android14-6.1-2023-10_r2 SHA1: fffe3966fa735fa5b94b LICENSES	kernel	boot-6.1.img boot-6.1-gz.img boot-6.1-lz4.img

下载后解压system_dlkm_staging_archive.tar.gz，并将解压后的flatten\lib\modules\下的ko文件拷贝到

```
kernel/prebuilts/6.1/arm64/
```

12 如何单独打包vendor_boot.img

- 步骤一：在kernel下面编译对应的ko文件
- 步骤二：拷贝KO文件到mkcombinedroot目录下
- 步骤三：拷贝vendor_boot.img到mkcombinedroot目录下
- 步骤四：进到mkcombinedroot目录下执行mkgki4.sh脚本更新ko并编译到vendor_boot.img中

- 步骤五： 烧写vendor_boot.img到机器中
  下面详细介绍各个步骤

步骤一：在kernel中编译KO

- 进入kernel目录
  Android14 + kernel6.1

```
cd kernel-6.1
```

- 导clang到环境

```
export PATH=../prebuilts/clang/host/linux-x86/clang-r487747c/bin:$PATH
```

- 编译KO

```
make CROSS_COMPILE=aarch64-linux-gnu- LLVM=1 LLVM_IAS=1 ARCH=arm64 gki_defconfig
rockchip_gki.config && make CROSS_COMPILE=aarch64-linux-gnu- LLVM=1 LLVM_IAS=1
ARCH=arm64 rk3562-rk817-tablet-v10.img -j32
```

步骤二：拷贝KO文件到mkcombinedroot目录下

```
llvm-objcopy --strip-debug  drivers/xxx.ko
../mkcombinedroot/vendor_ramdisk/lib/modules/xxx.ko
```

步骤三：拷贝vendor_boot.img到mkcombinedroot目录下

单独编译vendor_boot.img需要拷贝一个vendor_boot.img的基础包，类似单独编译boot.img需要一个boot_sample.img, 这个vendor_boot.img需要和你准备更新的机器里面的vendor_boot.img一样，可以从GKI的固件里面拷贝。

步骤四：进到mkcombinedroot目录下执行mkgki4.sh脚本更新ko并编译到vendor_boot.img中

```
cd ../mkcombinedroot/
```

编译vendor_boot.img，其中：

- DTS=板级dts名称，dts需要使用res/board/下有定义的load名

```
  ./mkgki4.sh DTS=rk3568-evb1-ddr4-v10
```

编译后会在mkcombinedroot根目录下生成new_vendor_boot.img

步骤五： 烧写new_vendor_boot.img到机器中

- 烧写`mkcombinedroot/new_vendor_boot.img`文件到机器中开机验证
  一般GKI的固件都是AB固件，所以烧写new_vendor_boot.img的时候需要同时更新vendor_boot_a和vendor_boot_b分区，如果机器烧写之前由于异常重启多次导致进入fastboot模式，此时需要一起烧写misc.img，清除misc分区中的标记才能正常启动， ubuntu下面的烧写工具的参考命令如下：

```
sudo ./upgrade_tool di -vendor_boot_a
mkcombinedroot/new_vendor_boot.img/vendor_boot.img
sudo ./upgrade_tool di -vendor_boot_b
mkcombinedroot/new_vendor_boot.img/vendor_boot.img;
sudo ./upgrade_tool rd
```

- 如果是放在vendor分区的ko可以在系统起来后直接push到机器内的vendor分区中，手动挂载进行验证
- 如果有涉及到dts的修改，需要烧写kernel-6.1下的`resource.img`