

Rockchip Android 14 GKI Developer Guide

ID: RK-KF-YF-778

Release Version: V1.1.0

Release Date: 2024-04-16

Security Level: ☐Top-Secret ☐Secret ☐Internal ☒Public

DISCLAIMER

THIS DOCUMENT IS PROVIDED "AS IS". ROCKCHIP ELECTRONICS CO., LTD.("ROCKCHIP")DOES NOT PROVIDE ANY WARRANTY OF ANY KIND, EXPRESSED, IMPLIED OR OTHERWISE, WITH RESPECT TO THE ACCURACY, RELIABILITY, COMPLETENESS, MERCHANTABILITY, FITNESS FOR ANY PARTICULAR PURPOSE OR NON-INFRINGEMENT OF ANY REPRESENTATION, INFORMATION AND CONTENT IN THIS DOCUMENT. THIS DOCUMENT IS FOR REFERENCE ONLY. THIS DOCUMENT MAY BE UPDATED OR CHANGED WITHOUT ANY NOTICE AT ANY TIME DUE TO THE UPGRADES OF THE PRODUCT OR ANY OTHER REASONS.

Trademark Statement

"Rockchip", "瑞芯微", "瑞芯" shall be Rockchip's registered trademarks and owned by Rockchip. All the other trademarks or registered trademarks mentioned in this document shall be owned by their respective owners.

All rights reserved. ©2023. Rockchip Electronics Co., Ltd.

Beyond the scope of fair use, neither any entity nor individual shall extract, copy, or distribute this document in any form in whole or in part without the written approval of Rockchip.

Rockchip Electronics Co., Ltd.

No.18 Building, A District, No.89, software Boulevard Fuzhou, Fujian, PRC

Website: www.rock-chips.com

Customer service Tel: +86-4007-700-590

Customer service Fax: +86-591-83951833

Customer service e-Mail: fae@rock-chips.com

Preface

Overview

This document introduces the development process and attention points of Android 14 GKI.

Intended Audience

This document (this guide) is mainly intended for:

Technical support engineers

Software development engineers

Revision History

Version	Author	Date	Change Description
V1.0.0	Wu Liangqing	2023- 11-16	Initial version
V1.1.0	Wu Liangqing	2024- 04-15	Modify the GKI compilation method to no longer pre-compile kernel modules

Contents

Rockchip Android 14 GKI Developer Guide

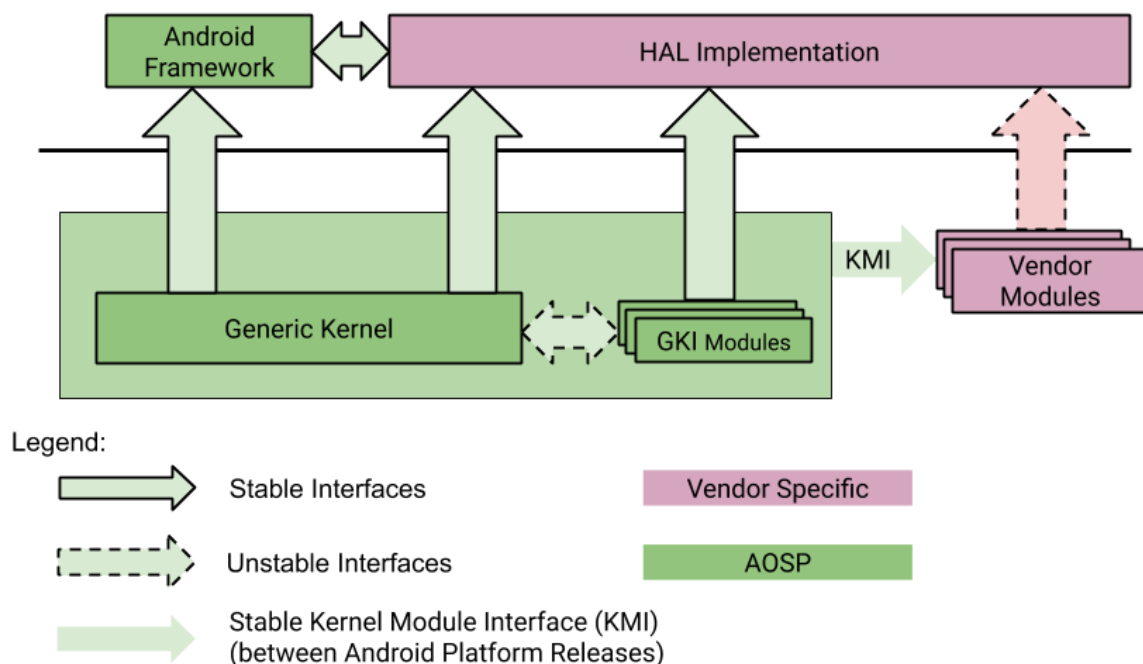
- 1 GKI Introduction
 - 1.1 What is GKI
 - 1.2 What products need GKI
 - 1.3 The difference of GKI and non-GKI
 - 2 Rockchip Android14 GKI Adaptation
 - 3 Google upstream kernel download and compile
 - 4 Introduction to GKI related directories of Rockchip SDK
 - 5 Requirements for GKI Compilation Environment
 - 6 Rockchip GKI Compile
 - 6.1 Code Modify
 - 6.2 Compile
 - 6.3 Firmware Flash
 - 7 KO Compile and Modify
 - 7.1 Method of adding new module drivers
 - 8 Uboot log verification
 - 8.1 uboot stage
 - 8.2 Android stage
 - 8.3 KO loading
 - 8.4 KO loading error
 - 8.5 bootcmdline parsing error
 - 8.6 Fail to load Mali KO
 - 8.7 kernel compiling error
 - 9 GKI compilation environment requirements
 - 10 Debugging skills
 - 10.1 Print more logs loaded by KO
 - 10.2 Compile GKI boot.img in RK kernel package
 - 10.3 Check the kernel interface published by google
 - 11 How to submit the kernel interface to upstream
 - 12 How to update boot.img published by AOSP
 - 13 How to pack vendor_boot.img solely
 - Step1: compile ko in kernel
 - Step2: copy ko file into mkcombinedroot directory
 - Step3: copy vendor_boot.img into mkcombinedroot directory
 - Step4: enter the mkcombinedroot directory and execute mkgki4.sh script, then update ko and compile it to vendor_boot.img
 - Step5: flash vendor_boot.img to the device
-

1 GKI Introduction

1.1 What is GKI

GKI: Generic Kernel Image

One of the difficulties of Android 14 GMS and EDLA authentication is that google mandates to support GKI. GKI is designed by google for solving the problem of kernel fragmentation by providing an unified core kernel and moving SOC and board-level drivers from core kernel into the loadable modules. The core kernel provides a stable kernel module interface for the driver module, and the driver and the kernel can be updated independently. The kernel interface can be extended by upstream. SOC and board-level vendors need use the kernel interfaces defined when developing, if you want to add core kernel interfaces, you need to submit to google, which will be a long time, so you need to make preparation in advance.



1.2 What products need GKI

- The products that use Android14 and require GMS and EDLA certifications
- The products that use Android13 and require GMS and EDLA certifications
- The products that use Android12 RK3588/RK3588S and require GMS and EDLA certifications
- The products without GMS and EDLA certifications are not forced to use GKI

1.3 The difference of GKI and non-GKI

- Generic Kernel boot.img

GKI	non-GKI
Google releases boot.img regularly, you can not modify the code	RK provides the kernel sources to compile, you can modify by yourselves freely

- Driver Module

GKI	non-GKI
Load in the form of KO, and the kernel interface invoked must be included in boot.img published by google	Embedded in boot, RK provides the kernel sources to compile, you can modify and add the kernel interfaces freely

- kernel code

GKI	non-GKI
Kernel sources published by RK only use for compiling KO of driver module	Kernel sources published by RK use for compiling the whole kernel and driver module, the module is embedded and compiled in the form of .o

- uboot supports head4
- Partition difference
GKI adds vendor_boot、init_boot、 resource partitions
- Enable AB partition

2 Rockchip Android14 GKI Adaptation

The kernel version is 6.1.

Chipset	Whether the adaptation is complete
RK3562	Yes
RK3568	Yes
RK3566	Yes
RK3588	Yes
RK3588S	Yes
RK3326	Yes
PX30	Yes
RK3399	Yes
RK3576	Yes

3 Google upstream kernel download and compile

The boot.img provided by google publishes regularly, and the time interval is relatively long. We can download the google upstream kernel to compile boot.img by ourselves to verify and debug.

Google Upstream kernel download link:

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

Need to link google server to download.

Compile:

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

Generate boot.img

```
out/boot.img
```

4 Introduction to GKI related directories of Rockchip SDK

- kernel KO file path

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

- Google boot.img path

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

- The protected KO file path published by Android AOSP

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

- KO loading sequence configuration file compiled by Kernel-6.1 source

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

- KO loading sequence configuration file loaded during Android Init stage
...
mkcombinedroot/res/vendor_modules.load
...

5 Requirements for GKI Compilation Environment

- Ubuntu version needs to be 20.04 or higher.
- pahole version needs to be 1.25 or higher.
-

6 Rockchip GKI Compile

6.1 Code Modify

Configure the GKI options in the device products directory of Android.

```
~/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 := false
+BOARD_BUILD_GKI := true
```

NOTE: the configuration of RK3562 UGO is enabling GKI by default, you needn't configure additionally.

If you compile uboot solely, you need to modify config to open AB configuration. If you compile fully by build.sh, then no need to modify, which will add AB macro configuration automatically during compiling.

- uboot need open AB configuration

```
~/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 Compile

Full compilation mode is the same to that of non-GKI.

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

Note: The kernel compiled here is only for generating the resource.img. The kernel source code will be compiled into KO files and packaged into vendor_boot.img. The kernel part uses the boot.img released by Google, and the specific path is in mkcombinedroot/prebuilts/boot-6.1.img.

Flash directly after compiling: `rockdev/Image-rk3562_ugo/update.img`

Compiling vendor_boot.img solely is also supported during debugging.
Compile command:

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

Flash directly after compiling:

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

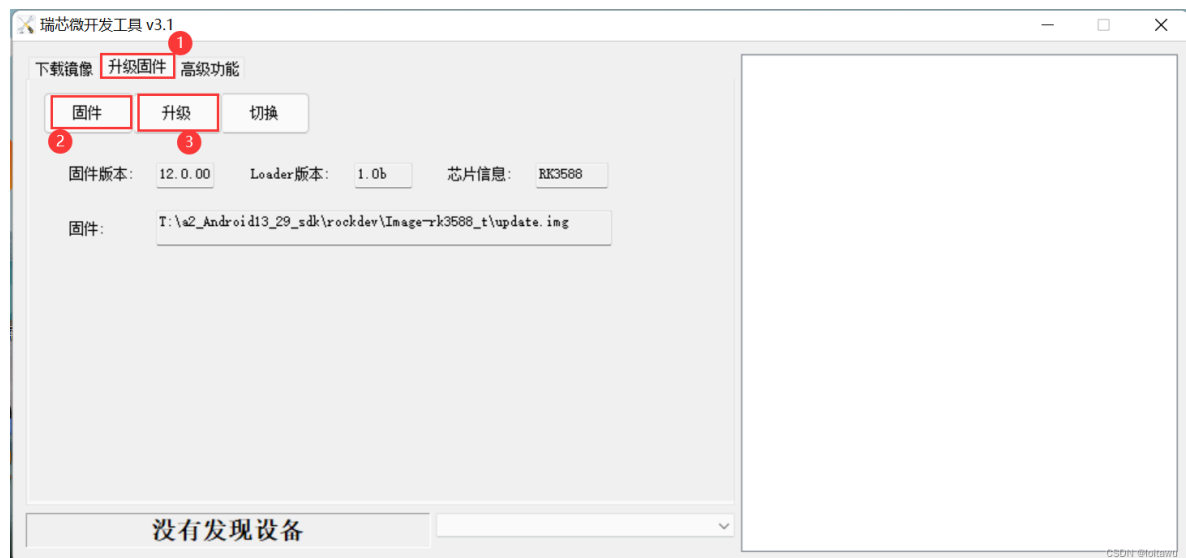
6.3 Firmware Flash

There are 2 ways to flash the firmware:

- Complete package update.img

Firmware path:

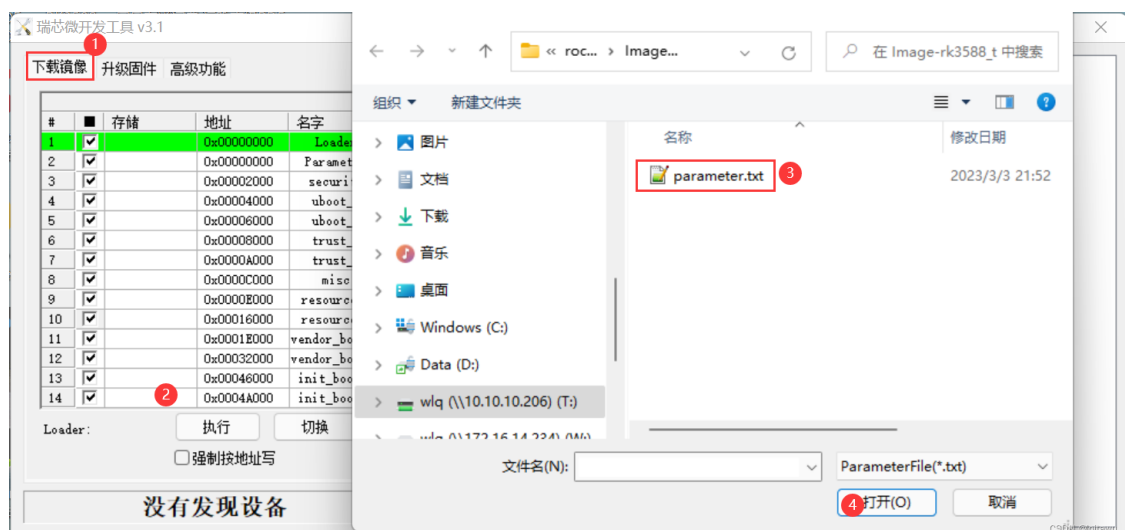
```
rockdev/Image-rk3562_ugo/update.img
```



You can flash through RK tools.

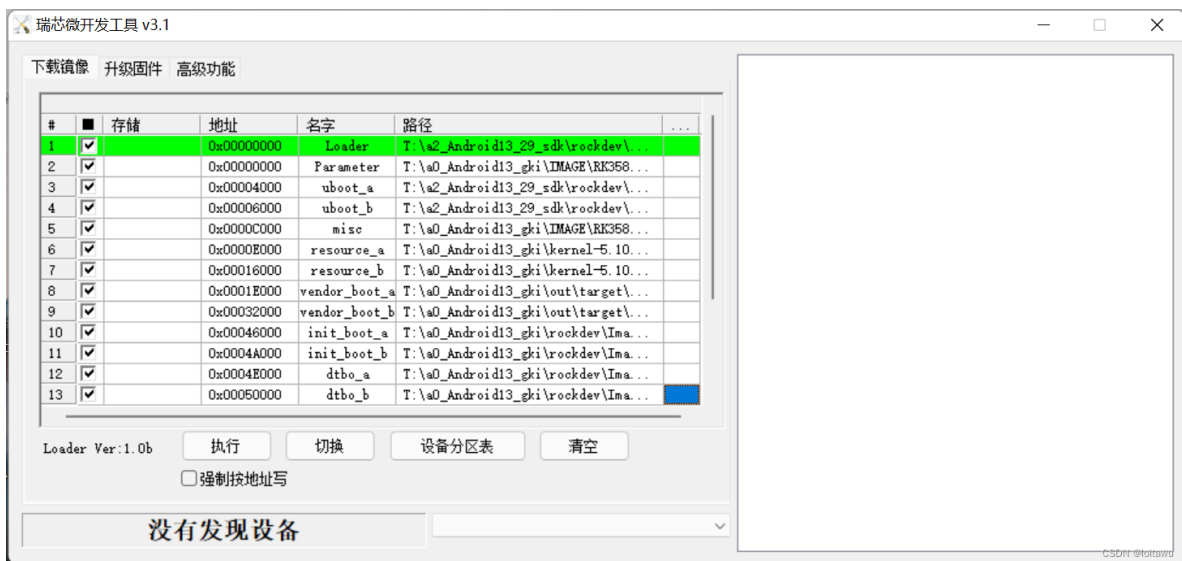
- Dispersive package to flash

First, import the configuration files, method: right click on the space of the tool-import configuration-select to import txt file-select parameter.txt.



Then select the img files corresponded to rockdev/Image-rk3588_t/ one by one to flash, the firmwares imported by partition A and B are the same.

```
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 Compile and Modify

7.1 Method of adding new module drivers

7.1.1 Put the driver codes into the corresponding directory of kernel-6.1, take adding touchscreen driver gt1x as an example:

Put gt1x driver into `drivers/input/touchscreen/`, and add corresponding `makefile` and `kconfig`. Here we follow kernel standard method to do.

7.1.2 Add an own config file, and new-create a `xxx_gki.config` under `arch/arm64/configs/`, and add `CONFIG_TOUCHSCREEN_GT1X=m` (m means being compiled into ko) to `xxx_gki.config`.

7.1.3 Add the KO file names to `mkcombinedroot/res/vendor_ramdisk_modules.load` or `mkcombinedroot/res/vendor_modules.load`.

.load file name	Corresponding partition	makefile analysis	Load time
vendor_ramdisk_modules.load	vendor_boot	vendor_ramdisk_gki.mk	ramdisk init stage
vendor_modules.load	vendor	vendor_gki.mk	android startup
recovery_modules.load	recovery	recovery_gki.mk	recovery stage

If the driver has no requirement for load time, it can be loaded in android stage, such as touchscreen driver, sensor driver and so on, detailed modification is followed:

- Enter mkcombinedroot directory

```
cd mkcombinedroot
```

- Add the ko name which needs to be compiled into vendor to res/vendor_modules.load, such as 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
```

- Compile and flash super.img after finishing addition.

When adding KO files to `res/vendor_modules.load`, they will be compiled into the `vendor_d1km/lib/modules/` directory. Therefore, during debugging, you can directly push the compiled KO files to the `vendor_d1km/lib/modules/` directory on the device. Afterward, reboot the device, and the KO files will be automatically loaded during boot.

Note : The mkcombinedroot/res/vendor_ramdisk_modules.load file is crucial for the loading order of drivers. Please do not modify the original order, as it may cause the system to fail to boot!!! New KO files are not recommended to be placed here unless absolutely necessary, as loading more KO files through vendor_ramdisk_modules.load will slow down the system startup. It is recommended to add new KO files to mkcombinedroot/res/vendor_modules.load, which is loaded during the Android boot stage and has a relatively minor impact on boot time.

7.1.4 Compile

For kernel modifications, it's advisable to use the build.sh -K script for compilation. When using build.sh -K, the script automatically copies KO files to the temporary directory mkcombinedroot/vendor_ramdisk/lib/modules during kernel compilation. In a complete compilation, the KO files from this directory will be packaged into vendor_boot.img or super.img. If you compile the kernel separately in the kernel directory, the generated KO files will not be automatically copied to mkcombinedroot/vendor_ramdisk/lib/modules. In this case, you will need

to manually navigate to the mkcombinedroot directory and execute the ./copy_modules.sh script to copy the KO files.

7.1.5 Compile vendor_boot.img in the project root directory, the command is as followed. This step is to pack KO file to vendor_boot.img, then flash it into the device.

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

```
Flash vendor_boot.img solely, and the vendor_boot.img path after finishing  
compiling is followed:  
```bash  
out/target/product/rk3562_ugo/vendor_boot.img
```

**NOTE: If the ko is compiled to vendor, then you need to compile super.img completely and flash super.img.**

#### 7.1.6 Verify

- Flash out/target/product/rk3562\_ugo/vendor\_boot.img file to the device for boot verifying.
- If the ko is put in vendor partition, then it can be push into the vendor partition of device directly after system boot, and mount manually to verify.
- If it's related to the modification of dts, then you need flash resource.img under kernel-6.1.

Attach: [Various ko loading stages defined by AOSP](#)

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 Uboot log verification

### 8.1 uboot stage

Content	header version
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 stage

GKI version: 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: Command line parameter such as androidboot.xxx can not exist in Header V4, this kind of parameters should be in bootconfig, which can be verified by `cat /proc/bootconfig`.

```
[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=ttyFIQO 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 loading

When starting to load ko, you can see the 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 loading error

Use an unexported symbol, and restart with errors:

```
[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
```

NOTE: Normally, this problem doesn't occur, please refer to `Noun explanation phase - ABI`.

## 8.5 bootcmdline parsing error

Error log

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

Phenomenon: unable to boot or boot into recovery.

Reason: The fields in cmdline are duplicated, resulting in a parsing cmdline error. You can press `ctrl+p` in the serial port when booting to uboot, and all cmdline information will be printed, and check which field is duplicated from the printed cmdline information. Then find the

corresponding definition in the codes and delete the corresponding field. cmdline is defined in the dts of device and kernel, so you can search for the duplicated field in both directories.

## 8.6 Fail to load Mali KO

The performance of Mali KO loading failure is unable to boot and the boot screen locking in the logo of 'Rockchip kernel'. You can find the surfaceflinger crash in logcat.

```
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
00000000000000168 x2 0000000000000006 x3 000000710899d340
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
000000739bcbda00 x10 0000000000000001 x11 000000739bcef6a0
04-27 22:45:27.654 366 366 F DEBUG : x12 000000710899d310 x13
0000000000000027 x14 000000710899d4e0 x15 00000000197b1a4f
04-27 22:45:27.654 366 366 F DEBUG : x16 000000739bd6dd58 x17
000000739bd48770 x18 0000007108812000 x19 00000000000000ac
04-27 22:45:27.654 366 366 F DEBUG : x20 00000000000000b2 x21
0000000000000014f x22 00000000000000168 x23 00000000ffffffff
04-27 22:45:27.654 366 366 F DEBUG : x24 b4000071bbca60b0 x25
000000710899dcb0 x26 000000710899dff8 x27 000000000000fe00
04-27 22:45:27.654 366 366 F DEBUG : x28 000000710899daf0 x29
000000710899d3c0
04-27 22:45:27.654 366 366 F DEBUG : 1r 000000739bcef3f4 sp
000000710899d000 (/system/bin/surfaceflinger) (BuildId: e2429c64ab29f2d0ffc5a8f42c0c1b80)
04-27 22:45:27.655 366 366 F DEBUG : #09 pc 00000000000054c50
/system/bin/surfaceflinger (BuildId: e2429c64ab29f2d0ffc5a8f42c0c1b80)
```

This is because the ko of GPU is not match, you need recompile the ko file of GPU, and copy to the corresponding directory under vendor/rockchip/common/gpu, the detail is as followed:

Modify kernel config in the product directory of device: PRODUCT\_KERNEL\_CONFIG := gki\_defconfig rockchip\_gki.config. And add the GPU configuration corresponding to the chip:

```

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 should be modified as followed:
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

```

Then execute `./build.sh -CK` to compile kernel.

After finishing compiling, copy the corresponding mali ko to the vendor, please refer to above sections for specific path.

## 8.7 kernel compiling error

Compiling error 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)

```

Resolution:

- Update latest pahole  
`git clone https://git.kernel.org/pub/scm/devel/pahole/pahole.git`
- Compile pahole

Install and compile dependency library.

```
sudo apt-get install cmake
```

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

If you have installed pahole before, you need to uninstall it first.

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

- Begin to compile

Execute in the ahole directory.

```
mkdir build
```

```
cd build/
```

```
cmake -D__LIB=lib -DBUILD_SHARED_LIBS=OFF ..
```

Configure static compilation

```
sudo make install
```

`pahole --version` to check the version to make sure that it is installed successfully.

## 9 GKI compilation environment requirements

- Ubuntu version requires 20.04 and above
- pahole version requires 1.25

## 10 Debugging skills

### 10.1 Print more logs loaded by KO

Modify the value of `ratelimit` to print more logs of init, in order to debug the problems. If init informations are few, the error informations loaded by ko will be hided.

```
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
```

### 10.2 Compile GKI boot.img in RK kernel package

Compile kernel according to normal steps first, then generate `arch/arm64/boot/Image`.

Pack boot.img by following commands:

```
mkbootimg --kernel arch/arm64/boot/Image --header_version 4 --output
```

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

### 10.3 Check the kernel interface published by google

The standard kernel interface definition is in the android directory:

```
:~/a5_google_kernel/common$ tree a
android/ arch/
w1q@sys2_206:~/a5_google_kernel/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_d1km_modules
```

## 11 How to submit the kernel interface to upstream

If you need to add a new kernel interface, you can generate the corresponding patch, and submit the patch to rockchip redmine to be examined and then submit it to google uniformly.

```
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
```

## 12 How to update boot.img published by AOSP

Android AOSP updates boot.img and corresponding protected KO file regularly, the Android release link is as followed:

<https://source.android.com/docs/core/architecture/kernel/gki-android14-6.1-release-builds>

After opening the link, find the latest release version of Android14-6.1, and then click corresponding boot-6.1.img to download, as the picture shows:

<a href="#">Release build</a> Debug build			
Release date	Tag / Source / Licenses	Kernel artifacts 	Certified GKI
2023-10-31	<a href="#">android14-6.1-2023-10_r1</a> SHA1: 835b6458fa548d62264f <a href="#">LICENSES</a>	<a href="#">kernel</a>	<a href="#">boot-6.1.img</a> <a href="#">boot-6.1-gz.img</a> <a href="#">boot-6.1-lz4.img</a>
2023-10-31	<a href="#">android14-6.1-2023-10_r2</a> SHA1: fffe3966fa735fa5b94b <a href="#">LICENSES</a>	<a href="#">kernel</a>	<a href="#">boot-6.1.img</a> <a href="#">boot-6.1-gz.img</a> <a href="#">boot-6.1-lz4.img</a>

After downloading the boot-6.1.img, copy it to:

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

From Android14, AOSP publishes protected ko module synchronously, which needs to be download together, and this ko file need to match with boot.img to load correctly.

Click the kernel link in the following picture to download system\_dkml\_staging\_archive.tar.gz.

<a href="#">Release build</a> Debug build			
Release date	Tag / Source / Licenses	Kernel artifacts 	Certified GKI
2023-10-31	<a href="#">android14-6.1-2023-10_r1</a> SHA1: 835b6458fa548d62264f <a href="#">LICENSES</a>	<a href="#">kernel</a>	<a href="#">boot-6.1.img</a> <a href="#">boot-6.1-gz.img</a> <a href="#">boot-6.1-lz4.img</a>
2023-10-31	<a href="#">android14-6.1-2023-10_r2</a> SHA1: fffe3966fa735fa5b94b <a href="#">LICENSES</a>	<a href="#">kernel</a>	<a href="#">boot-6.1.img</a> <a href="#">boot-6.1-gz.img</a> <a href="#">boot-6.1-lz4.img</a>

Download and unzip system\_dkml\_staging\_archive.tar.gz, and copy the unzipped ko file in flatten\lib\modules\ to:

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

## 13 How to pack vendor\_boot.img solely

- Step1: compile the corresponding ko file in kernel.
- Step2: copy ko file into mkcombinedroot directory.



- Step3: copy vendor\_boot.img into mkcombinedroot directory.
- Step4: enter the mkcombinedroot directory and execute mkgki4.sh script, then update ko and compile it to vendor\_boot.img.
- Step5: flash vendor\_boot.img to the device.

Now we introduce each steps in the following:

## Step1: compile ko in kernel

- Enter kernel directory  
Android14 + kernel6.1

```
cd kernel-6.1
```

- Export clang to the environment

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

- Compile 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
```

## Step2: copy ko file into mkcombinedroot directory

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

## Step3: copy vendor\_boot.img into mkcombinedroot directory

Compiling vendor\_boot.img solely needs to copy a vendor\_boot.img base pack, just like that compiling boot.img solely needs a boot\_sample.img. This vendor\_boot.img needs to be the same with the vendor\_boot.img in the device you prepare to update, which can be copied from GKI firmware.

## Step4: enter the mkcombinedroot directory and execute mkgki4.sh script, then update ko and compile it to vendor\_boot.img

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

Compile vendor\_boot.img, thereinto:

- DTS=board-level dts name, dts needs to use the load name defined in res/board/.

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

After compiling, there will be `new_vendor_boot.img` generated in the `mkcombinedroot` root directory.

## Step5: flash vendor\_boot.img to the device

- Flash `mkcombinedroot/new_vendor_boot.img` file to the device and boot to verify.  
Generally, GKI firmware is AB firmware, so when flashing `new_vendor_boot.img`, you need update `vendor_boot_a` and `vendor_boot_b` partitions at the same time. If the device has been in fastboot mode due to multiple abnormal restarts before flashing, you need to flash `misc.img` at the same time. Only the mark in misc partition is cleaned, can the device boot, the reference commands of flash tools in ubuntu are as followed:

```
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
```

- If the ko is put in vendor partition, then it can be pushed to the vendor partition of device directly after system enabled, and mount manually to verify.
- If it's related to dts modification, you need to flash `resource.img` in kernel-6.1.