IPC Linux SDK Quick Start

ID: RK-JC-YF-920

Release Version: V1.2.1

Release Date: 2023-01-16

Security Level: □Top-Secret □Secret □Internal ■Public

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

The document presents the basic usage of Rockchip IPC Linux SDK, aiming to help developers get started with IPC Linux SDK faster.

Chipset and System Support

Chip Name	Kernel Version
RK3588	Linux 5.10
RV1106/RV1103	Linux 5.10
RV1126/RV1109	Linux 4.19

Intended Audience

This document (this guide) is mainly intended for:

Technical support engineers

Software development engineers

Revision History

Version	Author	Date	Revision History
V1.0.0	CWW	2021-12-17	Initial version
V1.0.1	CWW	2022-01-01	1. Update document 2. Add Q&A
V1.0.2	GZC	2022-01-13	1. Update Q&A: How to use Recovery and Cannot Detect Device When SD Card Inserted
V1.0.3	CWW	2022-02-06	1. Update Cross toolchain download and installation 2. Add Third-Party program integration instructions 3. Update Compilation instructions of adding new APPs to project
V1.0.4	Ruby Zhang	2022-02-15	Update some languages description
V1.0.5	CWW	2022-02-21	1. Update Cross Toolchain Download and Installation 2. Add The code and document of secure boot
V1.0.6	GZC, CWW	2022-03-02	1. Update Q&A: a. Partition Table Introduction b. How to Use TFTP Upgrade in U-Boot Terminal c. How to Use SDcard Upgrade in U-Boot Terminal 2. Update firmware upgrade tool 3. Add Sysdrv Directory Introduction and Media Directory Introduction 4. Add BoardConfig.mk Introduction
V1.0.7	CWW	2022-03-26	1. Update SocToolKit 2. Add Download RV1106 IPC SDK Online 3. Add Download RV1126/RV1109 IPC SDK Online
V1.0.8	GZC	2022-04-02	1. Update Q&A: a. OEM Partition Mounting Introduction b. How to Download the NPU Model Transformation Tool and Runtime Library c. The Usage of the Stress Test 2. Update SDK Directory Structure
V1.0.9	CWW	2022-04-12	1. Update spi nor partition 2. Fix some typos
V1.1.0	CWW	2022-04-15	1. Add Debug via Telnet 2. Add The env.img format introduction 3. Add Kernel Driver Insmod Introduction 4. Add Build kernel's driver
V1.1.1	CWW	2022-05-07	1. Update Development Flashing Tools 2. Update Media Directory Introduction 3. Update BoardConfig.mk Introduction 4. Update Kernel Driver Insmod Introduction

Version	Author	Date	Revision History
V1.1.2	CWW	2022-05-09	1. Update How to Use TFTP Upgrade in U-Boot Terminal 2. Update Documents Introduction
V1.1.3	CWW	2022-05-20	1. Add Debug via Serial Port 2. Add How to Use the coredump 3. Update BoardConfig.mk Introduction 4. Update Kernel Driver Insmod Introduction 5. Add Information About Libraries And Driver For RV1106 And RV1103 5. Add How to Use NFS 6. Add How to Add A New User and Set Password For Login 7. Add Get the Camera Support Lists
V1.1.4	GZC	2022-05-30	1. Update SD Card Upgrade and Booting Making Tool 2. Update How to Use SDcard Upgrade in U-Boot Terminal 3. Add Get the Flash Support Lists 4. Update Update.img Related Tool 5. Update Factory Firmware Introduction 6. Add How to Change the CMA Size on Board
V1.1.5	GZC	2022-07-12	1. Update Mass Production Upgrade Tool 2. Update Set up a Development Environment
V1.1.6	CWW	2022-08-02	1. Add How to Use rndis
V1.1.7	CWW	2022-10-25	1. Update BoardConfig.mk Introduction 2. Add App Directory Introduction 3. Update How to Change the CMA Size on Board 4. Update Packaging env.img
V1.1.8	GZC	2022-11-17	1. Add The Usage of A/B Systems 2. Update Q&A: How to use Recovery 3. Add How to Optimize the Booting Time of SPI NOR 4. Update SDK Obtaining
V1.2.0	CWW GZC	2022-12-08	1. Add How to Add Non-root User Login 2. Update How to Add Third-Party Libraries to the sysdrv Directory 3. Update Kernel Driver Insmod Introduction 4. Add How to Add New Camera Sensor 5. Add How to Reboot into The Terminal of U-Boot 6. Update BoardConfig.mk Introduction 7. Update Set up a Development Environment 8. Update The Code and Document of Secure Boot 9. Update How to use Recovery 10. Add How to Support USB Mass Storage in U-Boot
V1.2.1	GZC	2023-01-16	1. Update The Code and Document of Secure Boot 2. Update Download the Repo Tool and Usage

Contents

1. Set up a Development Environment

This SDK is developed and tested on Ubuntu system, so it is recommended to use Ubuntu18.04 for compilation. Other Linux versions may should adjust the software package accordingly. In addition to the system requirements, there are other hardware and software requirements.
Hardware requirements: 64-bit system, hard disk space should be greater than 20G. If you do multiple builds, you will need more hard drive space

Software requirements: Ubuntu 18.04 system:

Please install software packages with below commands to setup SDK compiling environment:

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

It is recommended to use Ubuntu 18.04 system or higher version for development. If you encounter an error during compilation, you can check the error message and install the corresponding software packages accordingly.

1.1 Download the Repo Tool and Usage

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

export PATH="$HOME/repo-tool:$PATH"
# Test command
repo version
```

1.2 SDK Obtaining

There are two ways to obtain the SDK: **download online** and **Offline SDK Package**.
Note: SDKs of different platforms require corresponding download permissions.

1. **Download online**

Please contact RK business or FAE to get the SDK Release document of related chip.
For example: The Linux IPC SDK Release document of RV1106/RV1103 is Rockchip_RV1106_RV1103_Linux_IPC_SDK_Release_V1.0.0_20220530_CN.pdf

2. **Offline SDK Package**

The offline SDK package can be obtained from RK FAE window.

```
# Take RK3588_IPC_LINUX_SDK_V1.0.0_XXX.tar.bz2 as an example
mkdir rk3588_ipc_linux_sdk
tar xf RK3588_IPC_LINUX_SDK_V1.0.0_XXX.tar.bz2 -C rk3588_ipc_linux_sdk
cd rk3588_ipc_linux_sdk

# Check out the local code
.repo/repo/repo sync -l
```

1.3 Update SDK Code

Before update SDK code, please make a local backup.

```
# The command of update SDK code
.repo/repo/repo sync -c --no-tags
# If some projects fail to download, add the --force-sync parameter
.repo/repo/repo sync -c --no-tags --force-sync
# After update SDK code, clean the SDK.
./build.sh clean
```

1.4 Cross Toolchain Download and Installation

The cross toolchain can be obtained from tools/linux/toolchain/ in the SDK directory.

Chip Name	Cross Toolchain	Test Commands
RK3588	gcc-arm-10.3-2021.07-x86_64 -aarch64-none-linux-gnu	aarch64-rockchip1031-linux-gnu-gcc --version
RV1106	arm-rockchip830-linux -uclibcgnueabihf	arm-rockchip830-linux-uclibcgnueabihf-gcc --version
RV1126 /RV1109	gcc-arm-8.3-2019.03-x86_64 -arm-linux-gnueabihf/	arm-rockchip830-linux-gnueabihf-gcc --version

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

The cross toolchain can be obtained from tools/linux/toolchain/ in the SDK directory.

2. SDK Usage Introduction

2.1 BoardConfig.mk Introduction

- BoardConfig.mk Introduction

SDK's board config file is in the directory of `project/cfg/`, BoardConfig.mk file is an important file for SDK compilation.
The `project/cfg-all-items-introduction.txt` file will record the latest config options introduction.

Config Options	Introduction
RK_ARCH	arm or arm64 Define compiling 32 bit or 64 bit program
RK_CHIP	non-modifiable Different chips correspond to different SDKs
RK_TOOLCHAIN_CROSS	non-modifiable Define cross-compilation toolchain
RK_BOOT_MEDIUM	emmc/spi_nor/spi_nand Define the storage medium of board
RK_UBOOT_DEFCONFIG	U-Boot defconfig file name Located in sysdrv/source/uboot/u-boot/configs
RK_UBOOT_DEFCONFIG_FRAGMENT	U-Boot config file name (optional) Located in sysdrv/source/uboot/u-boot/configs Overlays the defconfig defined by RK_UBOOT_DEFCONFIG
RK_KERNEL_DEFCONFIG	Kernel defconfig file name Located in sysdrv/source/kernel/arch/$RK_ARCH/configs
RK_KERNEL_DEFCONFIG_FRAGMENT	Kernel defconfig file neme (Optional) Located in sysdrv/source/kernel/arch/$RK_ARCH/configs Overlays the defconfig defined by RK_KERNEL_DEFCONFIG
RK_KERNEL_DTS	Kernel dts file name RK_ARCH=arm Located in sysdrv/source/kernel/arch/arm/boot/dts RK_ARCH=arm64 Located in sysdrv/source/kernel/arch/arm64/boot/dts/rockchip
RK_MISC	If recovery is enabled, Read the flag at system startup and select it into the recovery system or application system (if there is no recovery, it can be removed)
RK_CAMERA_SENSOR_IQFILES	Camera Sensor IQ file name Located in media/isp/camera_engine_rkaiq/iqfiles or media/isp/camera_engine_rkaiq/rkaiq/iqfiles If there are multi IQ files, separated by spaces. e.g. RK_CAMERA_SENSOR_IQFILES="iqfile_1 iqfile_2"
RK_PARTITION_CMD_IN_ENV	Configure partition table (IMPORTANT) The Format of Partition Table:<partdef>[,<partdef>] <partdef> format: <size>[@<offset>](part-name) For detailed configuration, please refer to Partition Table Introduction chapter

Config Options	Introduction
RK_PARTITION_FS_TYPE_CFG	Configure filesystem type of partition and mount point (IMPORTANT) Format Introduction: "partition name"@"the mount point of partition"@"filesystem type of partition" Notice: the mount point of root filesystem is IGNORE default (non-modifiable)
RK_SQUASHFS_COMP	Config squashfs image compression algorithm (Optional) options: lz4/lzo/lzma/xz/gzip (default xz)
RK_UBIFS_COMP	Config ubifs image compression algorithm options: lzo/zlib (default lzo)
RK_APP_TYPE	Config the reference app compilation type (Optional) run `./build.sh info` to get the options of app type
RK_APP_IPCWEB_BACKEND	Config whether to build WebServer app (Optional) y: enable
RK_BUILD_APP_TO_OEM_PARTITION	Config whether to install app to oem partition (Optional) y: enable
RK_ENABLE_RECOVERY	Config whether to enable recovery compilation (Optional) y: enable n: disable
RK_ENABLE_FASTBOOT	Config whether to enable fastboot (Optional) y: enable Needs to be combined with U-Boot and kernel modifications, refer to SDK BoardConfig-*-TB.mk
RK_ENABLE_GDB	Config whether to enable gdb compilation (Optional) y: enable n: disable
RK_ENABLE_ADBD	Config whether to enable adb (Optional) y: enable n: disable Notice: needs to be enable USB defconfig in kernel
RK_BOOTARGS_CMA_SIZE	Config CMA size (Optional)
RK_POST_BUILD_SCRIPT	The script to execute configuration before packaging rootfs.img (Located in BoardConfig dir, optional)
RK_PRE_BUILD_OEM_SCRIPT	The script to execute configuration before packaging oem.img (Located in BoardConfig dir, optional)
RK_BUILD_APP_TO_OEM_PARTITION	Config whether to enable install application to oem partition (Optional) :y: enable
RK_ENABLE_RNDIS	Config whether to enable rndis (Optional)y: enable n: disable

Config Options	Introduction
RK_META_PARAM	Conifg the parameter of meta partition (Optional, which is used for battery IPC product)

- The command to select BoardConfig

`./build.sh lunch`

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

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

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

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

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

Enter the corresponding number to select the corresponding reference board level.

2.2 Check the SDK Version and Build Configuration

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

The command to check the sdk version `sdkinfo`

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

2.3 One-click Automatic Compilation

```
./build.sh lunch    # Select reference board level
./build.sh          # One-click automatic compilation
```

2.4 Build U-Boot

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

# The detailed uboot compilation command
# ./build.sh info
```

Generate image files: output/image/download.bin output/image/idblock.img output/image/uboot.img

2.5 Build kernel

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

# The detailed kernel compilation command
# ./build.sh info
```

Generate image file: output/image/boot.img

2.6 Build rootfs

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

Make rootfs.img firmware by the command of `./build.sh firmware`
Generate image file: output/image/rootfs.img

2.7 Build media

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

The storage directory of the generated files: output/out/media_out

2.8 Build Reference Applications

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

The storage directory of the generated file: output/out/app_out
Note: app depends on media

2.9 Build Kernel's Driver

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

The storage directory of the generated file: output/out/sysdrv_out/kernel_drv_ko/

2.10 Packaging env.img

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

Use uboot's mkenvimage tool to package env.img
The command format: `mkenvimage -s $env_partition_size -p 0x0 -o env.img env.txt`
NOTICE: The different storage medium, **$env_partition_size** is different，please refer to Partition Table Introduction for details.

To see the details of env.img: `strings env.img`

```
# for example, the content of eMMC's env.txt
blkdevparts=mmcblk0:32K(env),512K@32K(idblock),256K(uboot),32M(boot),2G(rootfs),1
G(oem),2G(userdata),-(media)
```

NOTICE: The different storage medium, env.img is different. Use `strings env.img` to see the details.

The blkdevparts will be transferred from the uboot to kernel and rewrite kernel's bootargs parameter.

2.11 Firmware Packaging

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

The path of the generated files: output/image

2.12 SDK Directory Structure

Directory Path	Introduction
build.sh	SDK compilation script soft-link to project/build.sh
media	Multimedia codec, ISP, etc. Algorithm related
sysdrv	U-Boot, kernel, rootfs directory
project	Reference applications, build configuration and script directory
docs	SDK Documents directory
tools	Image packaging tools and Burning tools
output	The directory where the image files are stored after SDK compilation
output/image	Firmware images directory
output/out	Files generated after compilation
output/out/app_out	Files generated after reference applications compilation
output/out/media_out	Files generated after media files compilation
output/out/sysdrv_out	Files generated after sysdrv files compilation
output/out/sysdrv_out/kernel_drv_ko	Ko files for peripherals and multimedia
output/out/rootfs_xxx	Rootfs source
output/out/S20linkmount	Partition mount script
output/out/userdata	userdata

Notice: media and sysdrv can be compiled independently of the SDK.

2.12.1 Sysdrv Directory Introduction

sysdrv can be built independently of the SDK and includes U-Boot, kernel, rootfs, and some image packaging tools.

Build Command:

```
# Build all default
make all

# Build U-Boot
make uboot_clean
make uboot

# Build kernel
make kernel_clean
make kernel

# Build rootfs
```

```
make rootfs_clean
make rootfs

# Clean
make clean

# Clean and remove out directory
make distclean

# Get the information of Build configuration, e.g. uboot or kernel compilation
command
make info
```

sysdrv subdir	Introduction
cfg	config kernel and U-Boot
out	sysdrv compilation files output directory
out/bin/board_glibc_xxx	programs which run on the board
out/bin/pc	programs which run on PC
out/bin/image_glibc_xxx	the output directory of generated firmware images
out/bin/rootfs_glibc_xxx	rootfs source files
source/busybox	busybox compilation directory, the source is located in sysdrv/tools/board/busybox
source/kernel	the directory of linux kernel source
source/uboot	the directory of U-Boot and rkbin (ddr init)
tools/board	the directory of tools source of board
tools/pc	Tools used on PC

2.12.2 Media Directory Introduction

The media directorycan be compiled independently of the SDK and includes multi-media encode and decode, ISP image algorithm.

Compiling Command:

```
# Compile all default
make

# Clean
make clean

# Get the information of compilation configure
make info
```

media subdir	Introduction
cfg	config whether to build the module
alsa-lib	Advanced Linux Sound Architecture (ALSA) library
avs	Any View Stitching (RK3588 ONLY)
common_algorithm	audio 3A algorithm, move detect, occlusion detect
isp	Image Signal Processing
iva	Intelligence Video Analysis (RV1106/RV1103/RK3588 ONLY)
ive	Intelligent Video Engine (RV1106/RV1103 ONLY)
libdrm	Direct Rendering Manager
libv4l	video4linux2 library
mali	GPU firmware and library (Notice: RK3588 ONLY, mali_csffw.bin MUST is located in /lib/firmware)
mpp	encode and decode interface, used for rkmedia and rockit, Not Recommended to call MPP directly
rga	Raster Graphic Acceleration Unit
rkmedia	Interface of multi-media (use for RV1126/RV1109)
rockit	Interface of multi-media (Recommend)
sysutils	Peripheral interface (ADC/GPIO/TIME/WATCHDOG)
samples	Test samples
out	The output directory of media compilation

2.12.3 App Directory Introduction

App directory: project/app

project/app subdir	Introduction
rkadk	rkadk has packaged basic commonly used interfaces, such as video recording, photography, playback, preview, etc., which simplifies the difficulty of application development.
rkfsmk_release	Optimize storage-related libraries (include FAT32 format, FAT32 filesystem repair, MP4 file repair)

2.13 The Output Directory of Images

The images after compiling SDK are located in output/image directory.

Image name	Introduction
download.bin	Will Only be downloaded to the DDR of the board
env.img	include partiton table and boot parameter(The env partiton is located at address 0 in the SDK by default)
idblock.img	loader image (include DDR init), used to load U-Boot image
uboot.img	uboot image
boot.img	Linux kernel image
rootfs.img	rootfs image
oem.img	oem image (Optional)
userdata.img	userdata image (Optional)

2.14 Debugging Tools

The images compiled by the SDK support adb and tftp tools for PC and board file transfer.

2.14.1 Transfer Files via Network tftp

```
### Get the IP address 192.168.1.159 of the PC
### Download files from the tftp server on the PC to the board
cd /tmp
tftp 192.168.1.159 -g -r test-file

### Upload files from the board to the tftp server on the PC
tftp 192.168.1.159 -p -l test-file
```

NOTICE: tftp server configuration refers to How to Use TFTP Upgrade in U-Boot Terminal

2.14.2 Debug via ADB

```
### Get the IP address 192.168.1.159 of the EVB board
adb connect 192.168.1.159

adb devices
List of devices attached
192.168.1.159:5555      device

### adb login EVB board to debug
adb -s 192.168.1.159:5555 shell
```

```
### Upload the file test-file from the PC to the directory /userdata of the EVB
board
adb -s 192.168.1.159:5555 push test-file /userdata/

### Download the file /userdata/test-file on the EVB board to the PC
adb -s 192.168.1.159:5555 pull /userdata/test-file test-file
```

2.14.3 Debug via Telnet

```
### Set the IP address
udhcpc -i eth0

### Run telnetd on board
telnetd
```

```
### Get the IP address 192.168.1.159 of the EVB board
### Run telnet on PC
telnet 192.168.1.159
### username: root
### password: rockchip
```

2.14.4 Debug via Serial Port

Chip Name	Serial Port Configure
RV1106/RV1103	Baudrate:115200, Data Bits:8, Parity:None, Stop Bits:1, Flow Type:None
RV1126/RV1109	Baudrate:1500000, Data Bits:8, Parity:None, Stop Bits:1, Flow Type:None
RK3588	Baudrate:1500000, Data Bits:8, Parity:None, Stop Bits:1, Flow Type:None

3. Documents Introduction

```
docs/
├── zh ---------------------------- SDK documents in Chinese
│ ├── bsp
│ ├── isp
│ ├── iva
│ ├── media
│ ├── security
│ └── ipc/Rockchip_Quick_Start_Linux_IPC_SDK_CN.pdf --- SDK quick start in
Chinese
│
└── en ---------------------------- SDK documents in English
  ├── bsp
  ├── isp
  ├── iva
```

```
├── media
├── security
└── ipc/Rockchip_Quick_Start_Linux_IPC_SDK_EN.pdf ---SDK quick start in English
```

4. Tools Introduction

Tools which are used for debugging and mass production are released with Rockchip Linux IPC SDKs for development. Tools version will be continuously updated with the SDK update. If you have any questions or requirements on the tools, please contact our FAE window fae@rock-chips.com.
There are two versions of tools: linux (tools used in Linux operating system) and windows (tools used in Windows operating system in the tools directory of Rockchip Linux IPC SDK.

- Windows Tools

Tools documents: tools/windows/ToolsRelease.txt

The Name of Tools	Usage of Tools
SocToolKit	Firmware update and used for entire update
DriverAssitant	Driver installation tool

- Linux tools

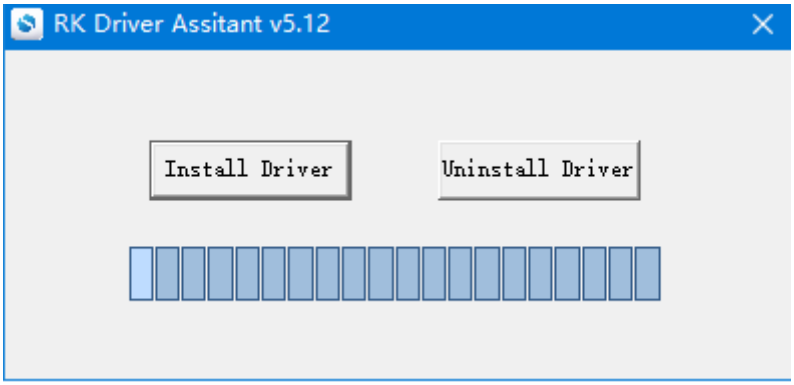
Tools related document: tools/linux/ToolsRelease.txt

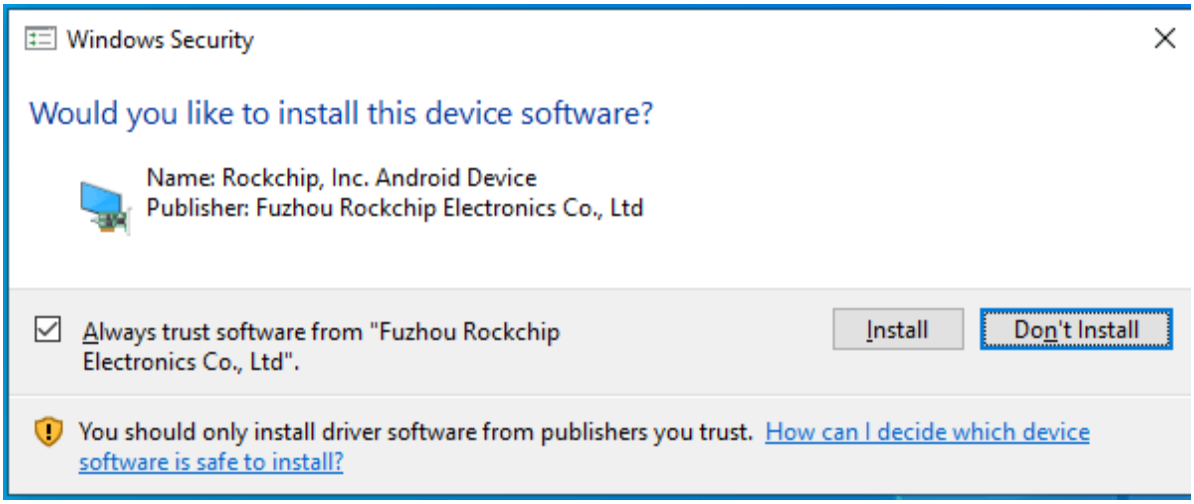
Tool Name	Tool Purpose
SocToolKit	Firmware update and used for entire update
Linux_Upgrade_Tool	Firmware update for command-line interface (only support USB)

4.1 Driver Installation Tool

Rockchip USB driver installation assistant is stored in `<SDK>/tools/windows/DriverAssitant_<version>.zip`. support win7_64, win10_64 and other operating systems.

The installation steps are as follows:



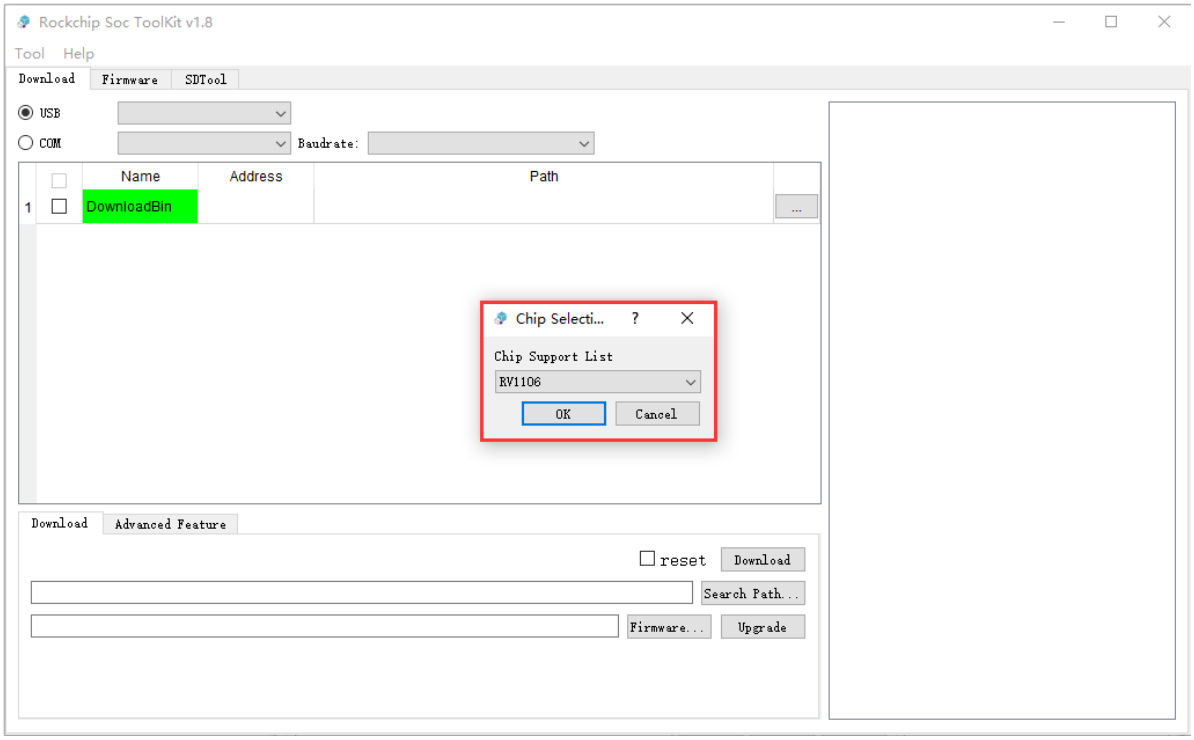


4.2 Development Flashing Tools

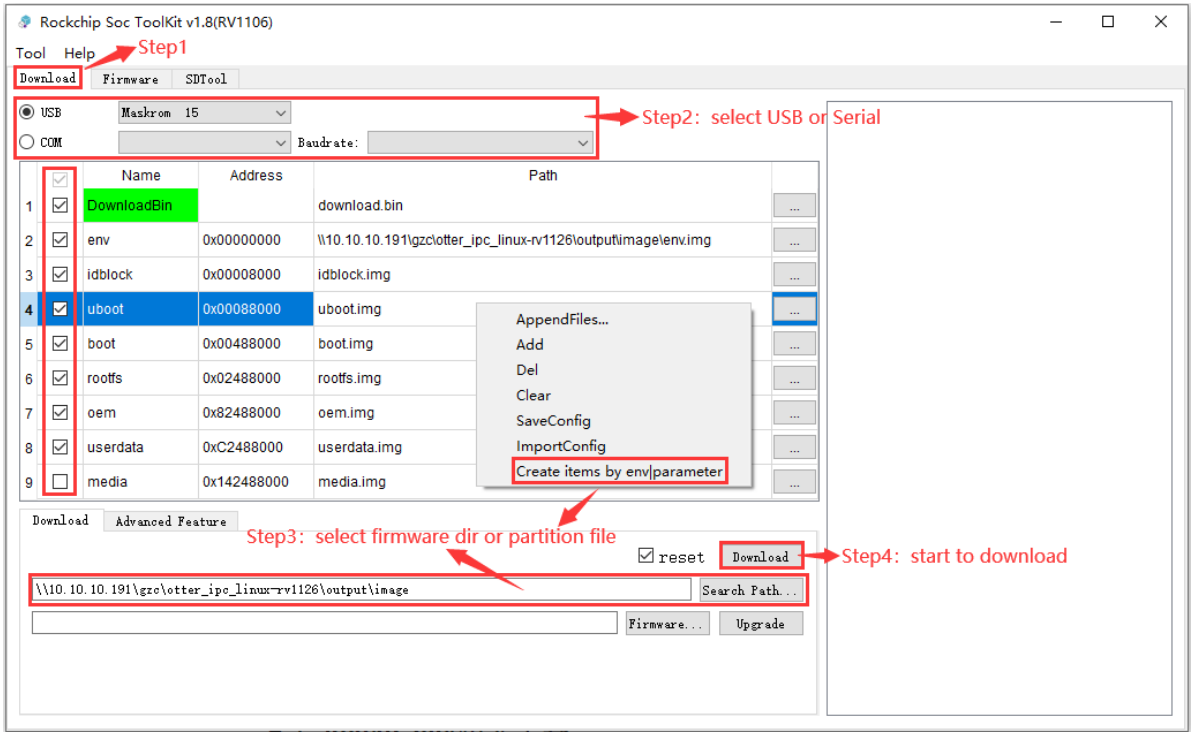
- The SDK provides Windows flashing tools , the tool is located in the project root directory:

  If the board has flashed firmware, we can enter U-Boot to upgrade firmware. For detailed instructions, please refer to the following chapters:
  How to Use TFTP Upgrade in U-Boot Terminal
  How to Use SDcard Upgrade in U-Boot Terminal

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



NOTICE：The funciotn of upgrading firmware by serial port is ONLY supported by RV1106/RV1103.



4.3 Update.img Related Tool

4.3.1 Packaging

When the SDK automatically compiles (`./build.sh`) with one click, the firmware to be flashed will be packaged into update.img and stored in the `<SDK>/output/image` directory. Alternatively, you can also run the following command to manually package the firmware in the preceding directory:

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

To customize the firmware directory, you can manually run the package script. View the help (-h) or enter the following options:

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

4.3.2 Unpacking

This function needs to be run manually. You can unpack the `<SDK>/output/image/update.img` into discrete firmware and save it in the `<SDK>/output/image/unpack` directory. The unpacking command is as follows:

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

To customize the firmware path, you can manually run the unpacking script. View the help (-h), or enter the following options:

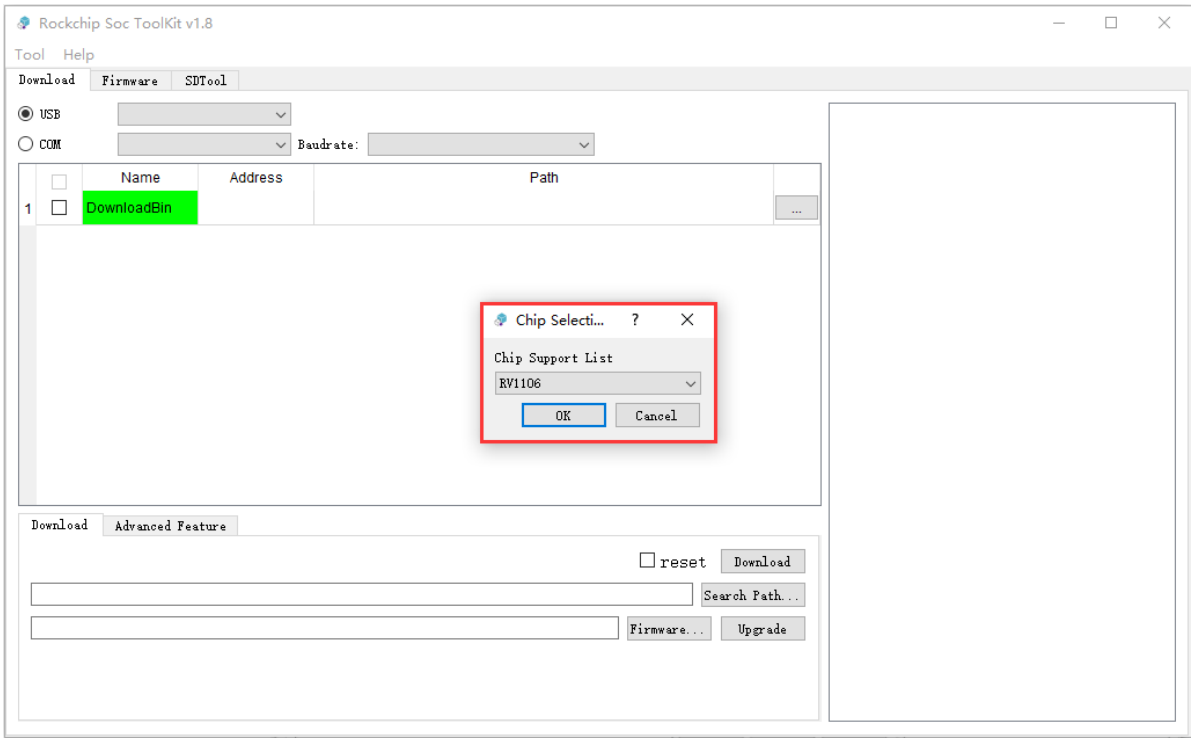
```
<SDK>/tools/linux/Linux_Pack_Firmware/mk-update_unpack.sh -i <IMAGE_PATH> -o
<UNPACK_DIR>
```

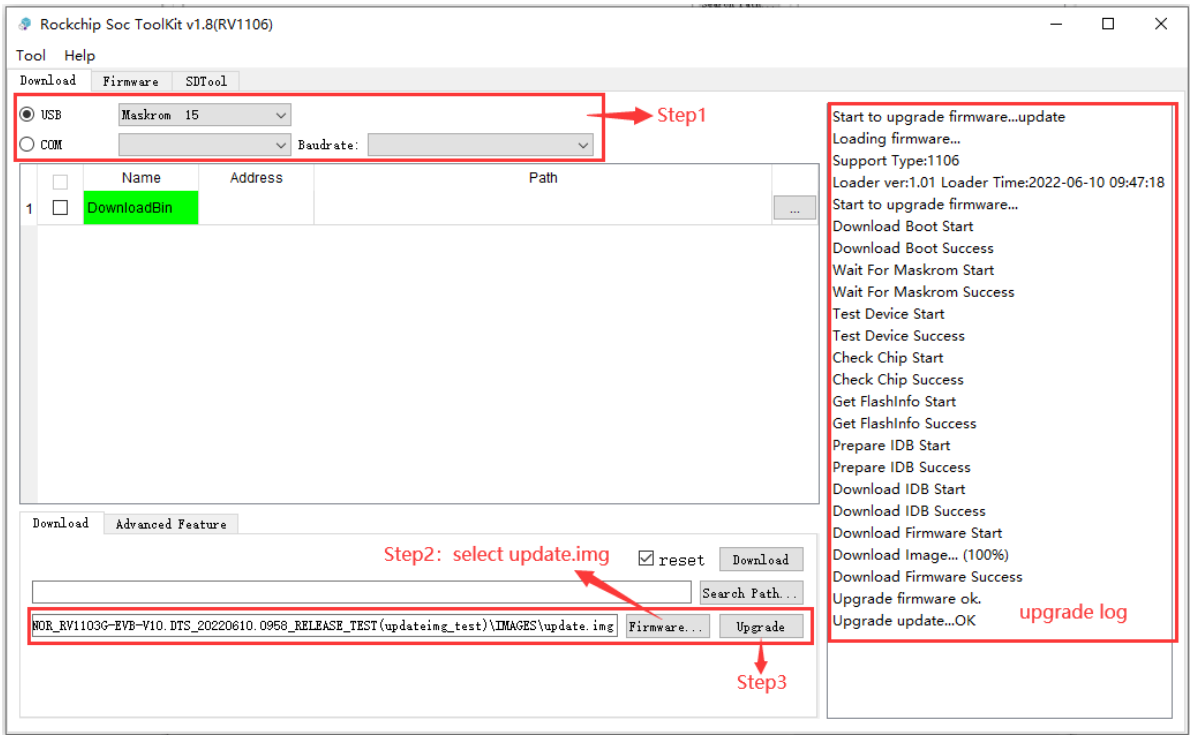
4.3.3 Update.img Flashing

- The SDK provides Windows flashing tools , the tool is located in the project root directory:

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

NOTICE: SocToolKit 1.8 or later is required to support the update.img flashing function.



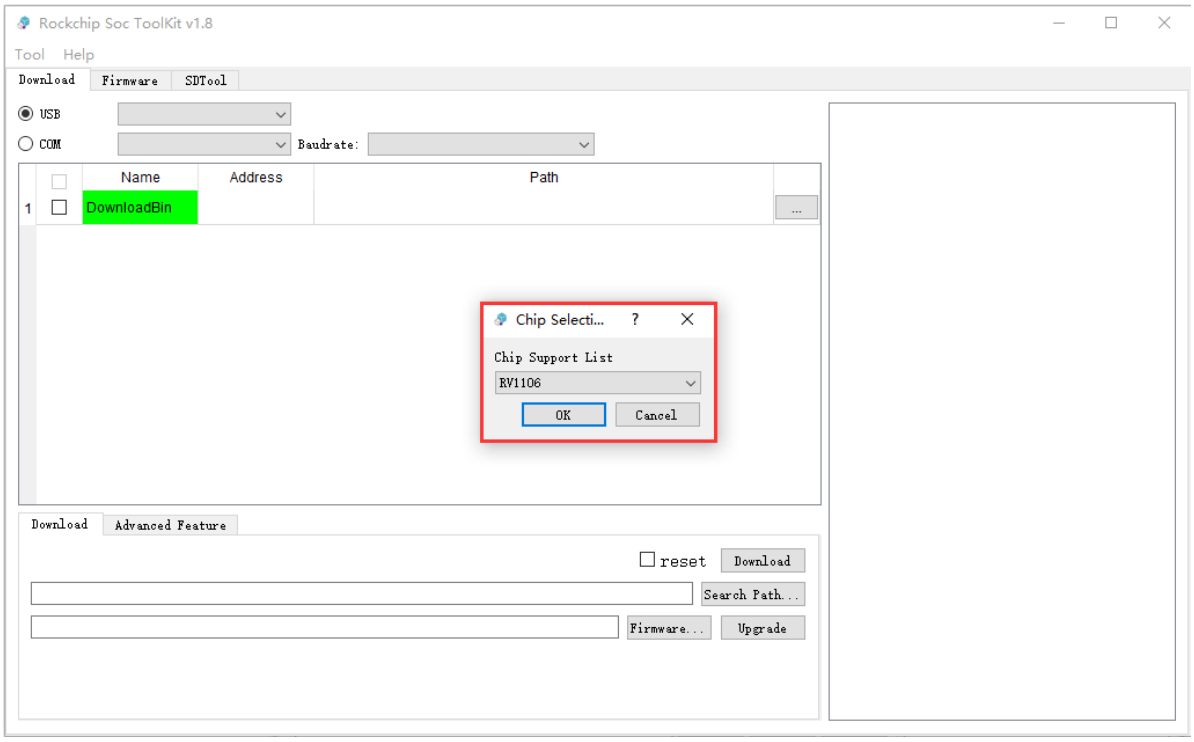


4.4 SD Card Upgrade and Booting Making Tool

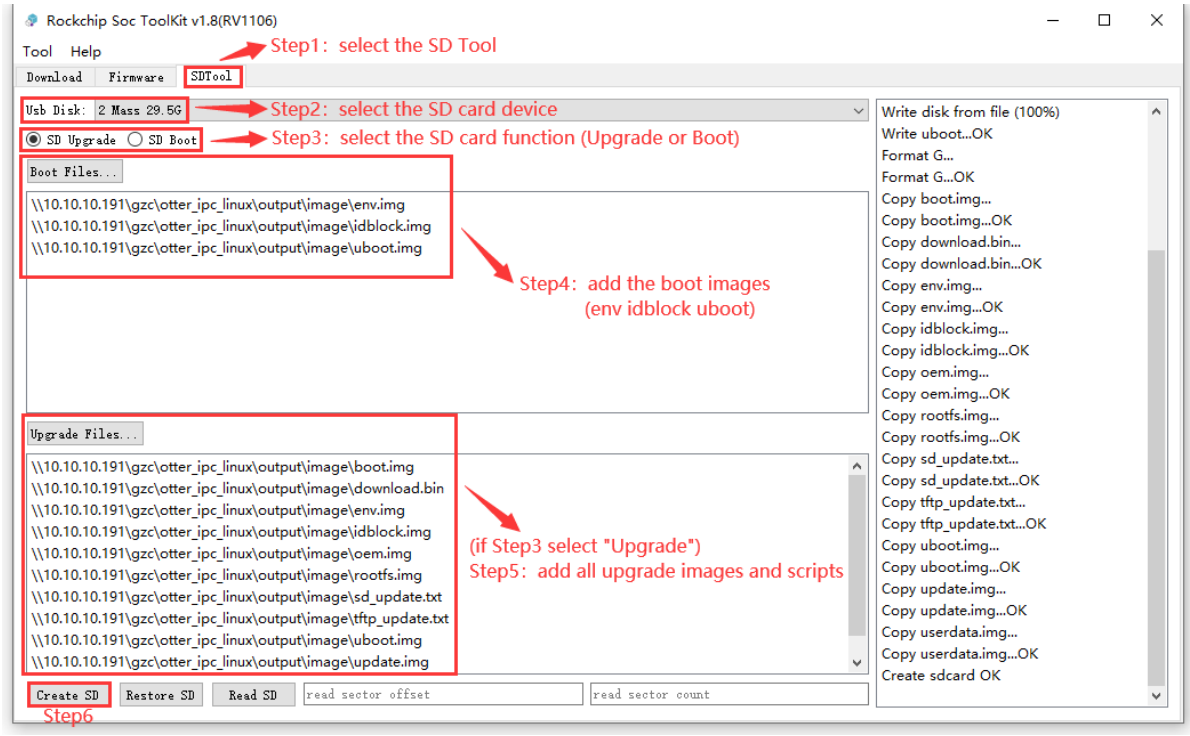
- The SDK provides Windows SD card upgrade and booting making tool, the tool is located in the project root directory.

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

NOTICE-1: SocToolKit 1.7 or later is required to support the SD card upgrade and booting function.



NOTICE-2: This function requires the user to run SocToolKit.exe as an administrator (asked by default when the tool is started).



- After inserting the SD card and restarting the device, the device preferentially accesses the U-boot terminal in the SD card.
- If the SD card has the upgrade function, the device is automatically upgraded.
- After the upgrade, remove the SD card and restart the device to access the device system.

4.5 Factory Firmware Introduction

The `programmer_image_tool` is required to make factory firmware. The tool is located in the `<SDK>/tools/linux/SocToolKit/bin/linux` directory. The input image is update.img. For details, see Update.img Related Tool.

The SDK supports building part of the flash type factory firmware. The generated firmware is stored in the `<SDK>/output/image/factory` directory. **For other flash types of firmware, see the document `Rockchip_Developer_Guide_Linux_Nand_Flash_Open_Source_Solution_CN.pdf` and use the tool `programmer_image_tool` to generate them.**

- SDK compilation command:

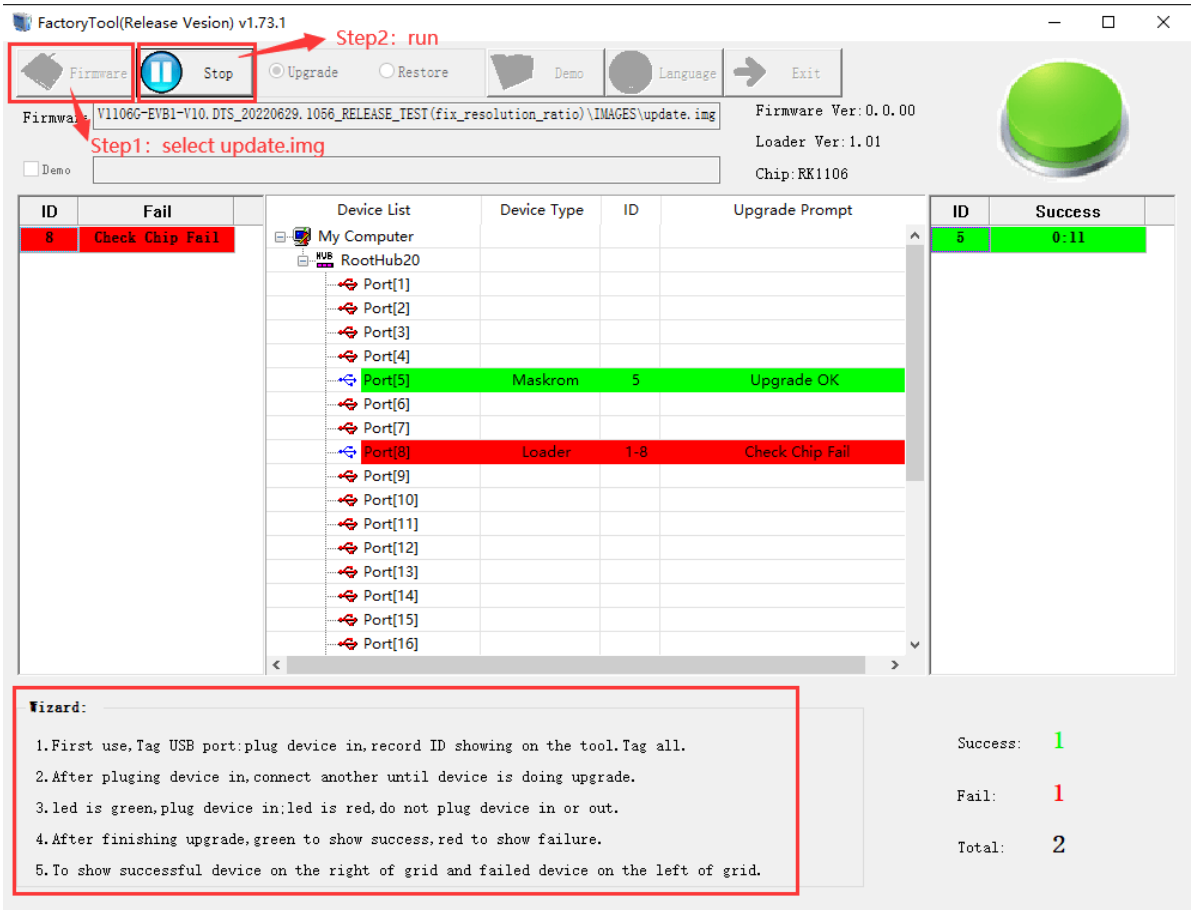
```
./build.sh factory
```

- The following flash types are supported by the compilation command:

flash type	block size	page size	oob size
emmc	-	-	-
spi nor	-	-	-
spi nand	128KB	2KB	-
slc nand	128KB	2KB	128B

4.6 Mass Production Upgrade Tool

Mass production upgrading needs the `FactoryTool`. The tool is located in the `<SDK>/tools/windows` directory. The input image is update.img. For details, see Update.img Related Tool.



After the tool runs, the upgrade automatically starts as long as the serial port is connected to the Maskrom or Loader device. No further operations are required.

Other content can be found in the wizard below the tool.

5. IPC Linux SDK Q&A

5.1 How to Modify Partition Table, Add Customized Partition and Partition Read/Write

5.1.1 Storage Medium and File System Type

Storage Medium	Supported Readable and Writable File System Formats	Supported Read-Only File System Formats
eMMC	ext4	squashfs
spi nand or slc nand	ubifs	squashfs
spi nor	jffs2	squashfs

File System Format	Script to Create Burning Image Files
ext4	output/out/sysdrv_out/pc/mkfs_ext4.sh
jffs2	output/out/sysdrv_out/pc/mkfs_jffs2.sh
ubifs	output/out/sysdrv_out/pc/mkfs_ubi.sh
squashfs	output/out/sysdrv_out/pc/mkfs_squashfs.sh

Note: Nand Flash hardware has different page size and block size, and requires to burn the corresponding image file, so mkfs_ubi.sh will package the burning image with different page size and block size by default.
For detailed Nand Flash instructions, please refer to the document Rockchip_Developer_Guide_Linux_Nand_Flash_Open_Source_Solution_CN.pdf

5.1.2 Partition Table Introduction

The SDK uses env partition to set partition table. Partition table information is configured by the parameter `RK_PARTITION_CMD_IN_ENV` in `<SDK>/project/cfg/BoardConfig*.mk`.

The partition table is stored as a string in the configuration. Here is each medium partition table.

Medium	Partition Table
eMMC	RK_PARTITION_CMD_IN_ENV= "32K(env),512K@32K(idblock),4M(uboot),32M(boot),2G(rootfs),-(userdata)"
spi nand 或slc nand	RK_PARTITION_CMD_IN_ENV= "256K(env),256K@256K(idblock),1M(uboot),8M(boot),32M(rootfs),-(userdata)"
spi nor	RK_PARTITION_CMD_IN_ENV= "64K(env),128K@64K(idblock),128K(uboot),3M(boot),6M(rootfs),-(userdata)"

The format for each partition is `<size>[@<offset>](part-name)`. The partition size and partition name are required, and the offset depends (see the Note 3 below).

There are the following points to note when configuring a partition table:

1. Partitions are separated by characters ",".
2. The unit of the partition size is K, M, G, T, P, and E, case insensitive. If there is no unit, the default unit is byte. "-" indicates that the partition size is the remaining capacity.
3. If the first partition starts at address 0x0, no offset is added. Otherwise, an offset must be added. The offsets of subsequent partitions can be added or not.
4. **idblock** partition offset and size is fixed. **non-modifiable**

5. It is not advised to change the env partition name. (If you want to modify env's partition offset and size, The defconfig CONFIG_ENV_OFFSET and CONFIG_ENV_SIZE of U-Boot must be modified, re-build firmware, erase board's flash, and update firmware finally)

5.1.3 Add Customized Partition

The following is an example of eMMC adding a readable and writable partition custom_part with a size of 64MB.

- Modify partition table parameters in the boardconfig using the preceding method [Partition Table Introduction](). For example: 64M(custom_part)
- Add a partition naming "custom_part" in `RK_PARTITION_CMD_IN_ENV` and modify the `RK_PARTITION_FS_TYPE_CFG` partition mount configuration in `<SDK>/project/cfg/BoardConfig*.mk`

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

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

- Make custom_part partition image

```
mkdir -p custom_part
./output/out/sysdrv_out/pc/mkfs_ext4.sh custom_part custom_part.img 64*0x100000
# Note: When using the default mount script of the SDK, the partition image file
name should be named with the partition name
# For example: the partition name is custom_part, and the partition image name is
custom_part.img
```

- Create a mount directory for the custom_part partition in the filesystem

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

- Repackage root filesystem (rootfs.img) `./build.sh firmware`
- Burn rootfs.img env.img and custom_part.img

5.1.4 OEM Partition Mounting Introduction

To mount OEM partition, the following configurations need to be modified:

- Add OEM partition to partition table (see Partition Table Introduction for details), for example:

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

- Add OEM configurations to file system type configuration (see Add Customized Partition for details), for example:

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

- Enable the following configuration:

```
# Enable app installation in OEM partition
export RK_BUILD_APP_TO_OEM_PARTITION=y
```

5.2 How to Use Recovery

There is one more recovery partition on the device in recovery mode, which consists of kernel+resource+ramdisk and is mainly used for upgrade operations. u-boot will determine whether the system to be booted is the Normal system or the recovery system according to the fields stored in the misc partition. Due to the independence of the system, the recovery mode can ensure the integrity of the upgrade, that is, if the upgrade process is interrupted, such as an abnormal power failure, the upgrade can still continue.

This chapter mainly introduces the upgrade process and technical details through the SD card local upgrade program Recovery. The following is the way to use recovery.

- The defconfig corresponding to the kernel requires to enable the configuration that supports INITRD.

```
CONFIG_BLK_DEV_INITRD=y
```

- Add the following configuration to `<SDK>/project/cfg/BoardConfig*.mk`:

```
#misc image
export RK_MISC=recovery-misc.img

# enable build recovery
export RK_ENABLE_RECOVERY=y

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

Note: If RK_OTA_RESOURCE is not enabled, uboot.img, boot.img and rootfs.img are packaged by default.

- Modify the partition table

Add two partitions: misc and recovery to the partition table. The size and order of the partitions can be adjusted within a reasonable range based on actual requirements.

- Build recovery

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

- Build the firmware that required to be upgraded

Manually build the firmware included in RK_OTA_RESOURCE. If it is not enabled, build the default partition firmware (please refer to the previous section for details of firmware compilation).

- Package OTA upgrade package

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

- Copy the OTA upgrade package

Copy the generated OTA upgrade package (`<SDK>/output/image/update_ota.tar`) to the root directory of the SD card.

- Insert the SD card to the device
- Enter the Recovery system on the device side to start the upgrade

```
reboot recovery
# Note: This command requires using busybox in the SDK or the corresponding patch
```

Enter the above code on the device side (board side) to enter the recovery system to start the upgrade.

Note: After the upgrade is complete, the device will restart and enter the normal system. If it fails, the device will stay in the recovery system and print log. If no SD card or upgrade package is found, the system will also restart to normal system.

5.3 How to Use TFTP Upgrade in U-Boot Terminal

The TFTP upgrade file is compiled in the `<SDK>/output/image/` directory along with the firmware. The file name is `tftp_update.txt`. The usage method is as follows:

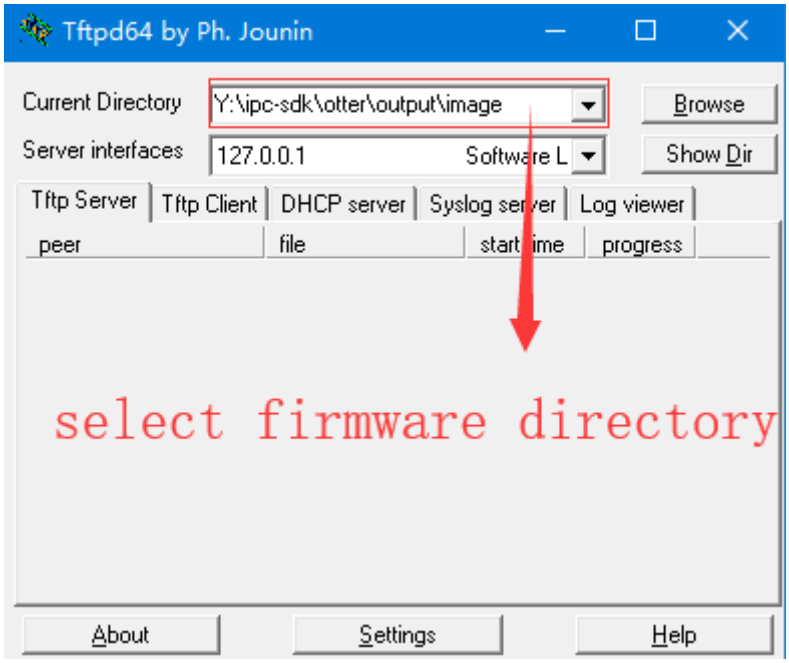
- Configure the TFTP Server

Tftpd64 download address https://pjo2.github.io/tftpd64

NOTICE:

1. Customers should follow relevant open source agreements when using Tftpd64

2. Customers shall take all legal risks and consequences from using Tftpd64



- Save the upgrade file `tftp_update.txt` and all firmware files with the file name extension `.img` to the specified directory on the server

(Note: SLC Nand does not support upgrading `idblock` partition by downloading firmware.)

- Set the IP address on the U-Boot terminal

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

The preceding IP addresses are for reference only. Set them based on actual conditions and ensure that the client and server reside on the same network segment.

- Run the `tftp_update` command on the U-Boot terminal

```
=>  tftp_update
ethernet@ffc40000 Waiting for PHY auto negotiation to complete. done
Using ethernet@ffc40000 device
TFTP from server 192.168.1.100; our IP address is 192.168.1.111
Filename 'tftp_update.txt'.
Load address: 0x3be24c00
Loading: *•#
     203.1 KiB/s
done
Bytes transferred = 1250 (4e2 hex)
...
```

5.4 How to Use SDcard Upgrade in U-Boot Terminal

The SD card upgrade file is compiled in the `<SDK>/output/image/` directory along with the firmware. The file name is `sd_update.txt`. The usage method is as follows:

- Save the upgrade file `sd_update.txt` and all firmware files with the file name extension `.img` to the specified directory in the root directory of the SD card

(Note: 1. SLC Nand does not support upgrading `idblock` partition by downloading firmware. 2. The SD card only supports FAT file systems.)

- Insert the SD card to the device
- Run the `sd_update` command on the U-Boot terminal

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

- After the upgrade, restart the device

5.5 Cannot Detect Device When SD Card Inserted

The defconfig corresponding to the kernel requires to enable the configuration that supports SD card.

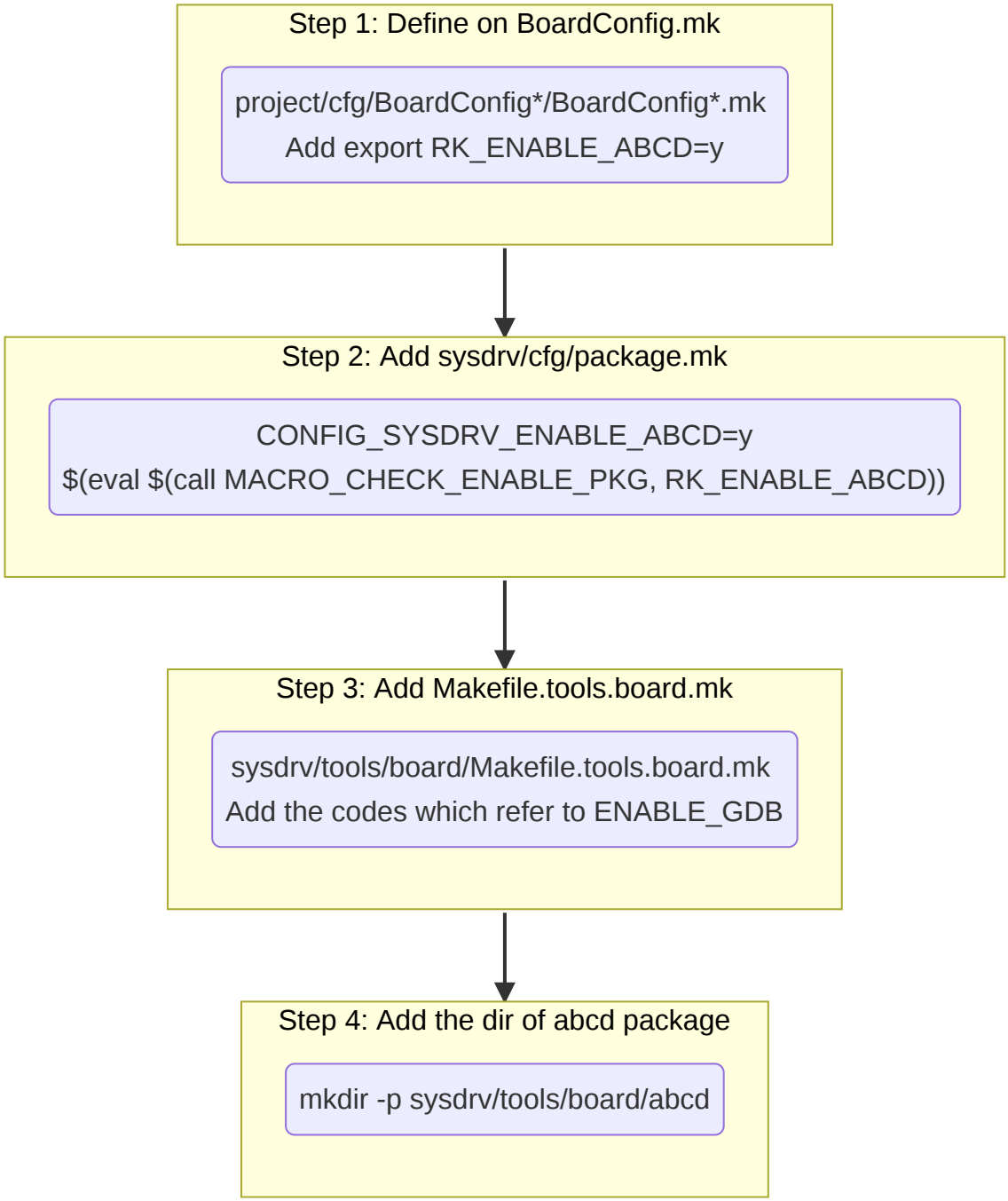
```
CONFIG_MMC_DW=y
CONFIG_MMC_DW_PLTFM=y
CONFIG_MMC_DW_ROCKCHIP=y
```

Add vfat filesystem support.

```
CONFIG_VFAT_FS=y
CONFIG_MSDOS_PARTITION=y
```

5.6 How to Add Third-Party Libraries to the sysdrv Directory

- sysdrv/tools/board is the program running on the board, now is going to introduce how to add a third-party program (for example, the third-party program is named ABCD).



- `touch sysdrv/tools/board/abcd/Makefile`

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

export LC_ALL=C
SHELL:=/bin/bash

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

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

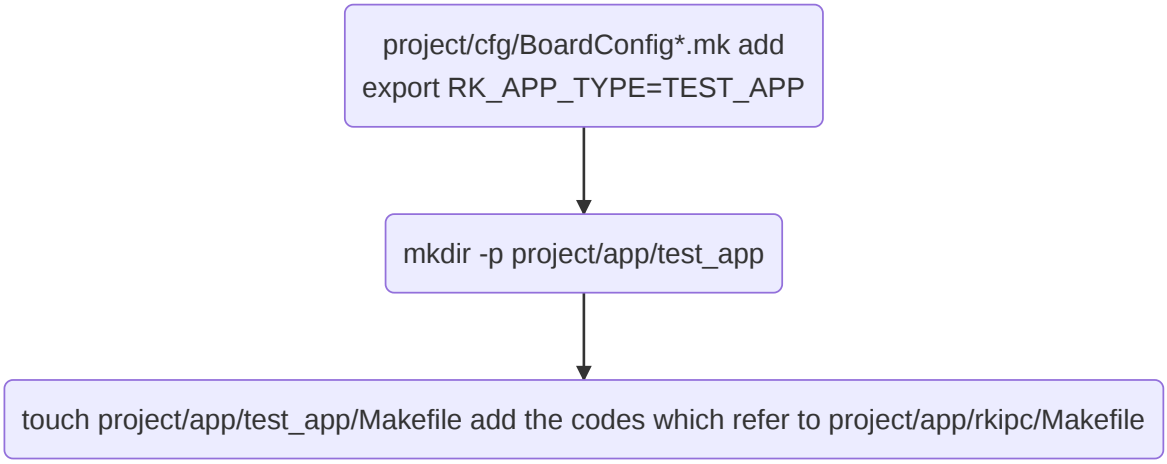
```
    tar -xf $(PKG_TARBALL); \
    mkdir -p $(CURRENT_DIR)/$(PKG_NAME)/$(PKG_BIN); \
    mkdir -p $(CURRENT_DIR)/$(PKG_BIN); \
    pushd $(CURRENT_DIR)/$(PKG_NAME)/; \
        ./configure --host=$(SYSDRV_CROSS) \
        --target=$(SYSDRV_CROSS) CFLAGS="$(SYSDRV_CROSS_CFLAGS)" \
        LDFLAGS="$(SYSDRV_CROSS_CFLAGS)"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

clean: distclean

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

- Test and build

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

5.7 How to Add New Applications in Project/app

- Take adding application test_app as an example




```
# project/app/test_app/Makefile reference code

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

export LC_ALL=C
SHELL:=/bin/bash
```

```
CURRENT_DIR := $(shell pwd)

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

RK_APP_CFLAGS = -I $(RK_APP_MEDIA_INCLUDE_PATH)

RK_APP_LDFLAGS = -L $(RK_APP_MEDIA_LIBS_PATH)

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

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

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

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

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

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

distclean: clean
```

- Test and build

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

5.8 How to Download the NPU Model Transformation Tool and Runtime Library

There are no NPU model transformation tool or NPU runtime library in the SDK, and it can be download from github.

The NPU model transformation tool download address:

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

NPU runtime library download address:

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

5.9 How to Change the CMA Size on Board

- Check the CMA size on board

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

- Enter the U-Boot terminal

  After the device is restarted, press `Ctrl+C` always. Until `=> <INTERRUPT>` is displayed, it is indicating that the U-boot terminal is entered.

- Check the CMA size in the U-Boot environment variables

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

  `rk_dma_heap_cma` is the CMA size.

  NOTICE: This environment variable is named `sys_bootargs`, and `rk_dma_heap_cma` is only one of its parameters. When modifying this environment variable, all contents after `sys_bootargs` should be considered as one.

- Modify environment variables, save environment variables, and restart the device

  Using the above environment variable as an example, change the CMA from 24M to 32M.

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

- Check the CMA size on board again

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

Notice: If security boot is enabled, sys_bootargs's parameter must be written into the bootargs parameter of kernel's dts file.

5.10 How to Use the coredump

- The defconfig corresponding to the kernel requires to enable the configuration that supports coredump.

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

- Set the size of coredump on board

```
ulimit -c unlimited
```

- Modify the coredump path on board

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

Notice: If write coredump file to the NFS or VFAT filesystem, we MUST run thess codes on board. If not, the size of coredump file is 0.

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

- List the stack of coredump

Copy the coredump to (e.g. /data/core-279-rkipc_get_nn_up) the directory of SDK and run these commands: For example, the toolchain of RV1106/RV1103 is **arm-rockchip830-linux-uclibcgnueabihf-gdb**

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

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

5.11 Kernel Driver Insmod Introduction

This chapter is used for RV1106/RV1103.
Please refer to `sysdrv/drv_ko/insmod_ko.sh`

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

insmod rk_dvbm.ko

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

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

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

insmod rga3.ko

insmod mpp_vcodec.ko
insmod rockit.ko

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

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

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

5.12 Information About Libraries and Driver for RV1106 And RV1103

- Libraries

Name	Size	Usage	Is it necessary
ld-uClibc-1.0.31.so	32K	Toolchain Library	YES
libatomic.so	16K	Toolchain Library	YES
libgcc_s.so	4.0K	Toolchain Library	YES
libgcc_s.so.1	124K	Toolchain Library	YES
libstdc++.so	992K	Toolchain Library	YES
libuClibc-1.0.31.so	420K	Toolchain Library	YES
libitm.so	52K	Toolchain Library	YES
librga.so	96K	Raster Graphic Acceleration Unit	YES
librkaiq.so	1.1M	Rockchip Auto Image Quality	YES
librockchip_mpp.so	169K	Encode And Decode Library	YES
librockit.so	812K	Interface of Multi-media Library	YES
libaec_bf_process.so	380K	Audio 3A Algorithm Library	NO
librkaudio_detect.so	148K	Audio Detect Library	NO
librockiva.so	760K	NPU Algorithm Library	NO
librknnmrt.so	84K	librockiva.so needed	NO
librve.so	96K	Intelligent Video Engine	NO
librkfsmk.so	68K	Optimize Storage-related Library	NO
librkmuxer.so	552K	Media File Stream Library	NO
libdrm_rockchip.so	8.0K	Direct Rendering Manager For Rockchip	NO
libdrm.so	48K	Direct Rendering Manager	NO
libcgicc.so	96K	Common Gateway Interface C++ Library	NO
libfcgi.so	32K	Fast Common Gateway Interface Library	NO
libfcgi++.so	16K	Fast Common Gateway Interface C++ Library	NO
libiconv.so	236K	Character Encoding Library	NO
libkmod.so	48K	udevadm Needed	NO
libblkid.so	180K	udevadm Needed	NO
libpcre.so	92K	Perl Regular Expression Library	NO
libwpa_client.so	28K	WiFi Tool Needed Library	NO
libz.so	76K	Compress Library	NO

- Drivers

Some of Kernel Driver	Size	Usage	Is it necessary
mpp_vcodec.ko	462K	Video Encode Driver	YES
phy-rockchip-csi2-dphy-hw.ko	14K	Mipi dphy rx Physics Driver	YES
phy-rockchip-csi2-dphy.ko	14K	Mipi dphy rx Logic Driver	YES
video_rkcif.ko	140K	Rockchip CIF Driver	YES
video_rkisp.ko	172K	Rockchip ISP Driver	YES
rockit.ko	109K	Interface of Multi-media	YES
rga3.ko	104K	Raster Graphic Acceleration Unit	YES
os04a10.ko	24K	os04a10 Sensor Driver	NO
sc3336.ko	16K	sc3336 Sensor Driver	NO
sc4336.ko	16K	sc4336 Sensor Driver	NO
sc530ai.ko	20K	sc530ai Sensor Driver	NO
rknpu.ko	32K	NPU Driver	NO
rve.ko	36K	Intelligent Video Engine	NO

5.13 How to Use NFS

- Enable NFS for kernel defconfig

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

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

- Config NFS Server On PC

```
# Ubuntu 16.04 install NFS server
sudo apt-get install nfs-kernel-server
# Create /opt/rootfs
mkdir /opt/rootfs
# Enable nobody mount /opt/rootfs
chmod 0+w -R /opt/rootfs
# Add the directory to exports
sudo echo "/opt/rootfs *(rw,sync,root_squash)" >> /etc/exports
# Update NFS configure
sudo exportfs -r
# Test NFS server
sudo mount -t nfs localhost:/opt/rootfs /mnt
# Ubuntu 16.04 disable firewall
sudo ufw disable
```

- Run NFS Mount Command On Board

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

5.14 How to Add a New User and Set Password for Login

For example, add the new user named testNewUser.

- Checking current users and get UID/GID

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

Add /etc/passwd on board

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

- Use the command of mkpasswd to generate password on PC

```
sudo apt install whois
mkpasswd -m "md5" "test123" # --> $1$kprQOoLU$kOU2H.ecXkAwlZJ0oplu/.
```

- Add /etc/shadow on board

```
testNewUser:$1$kprQOoLU$kOU2H.ecXkAwlZJ0oplu/.:0:0:99999:7:::
```

- Add /etc/group on board

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

- Modify /etc/inittab on board

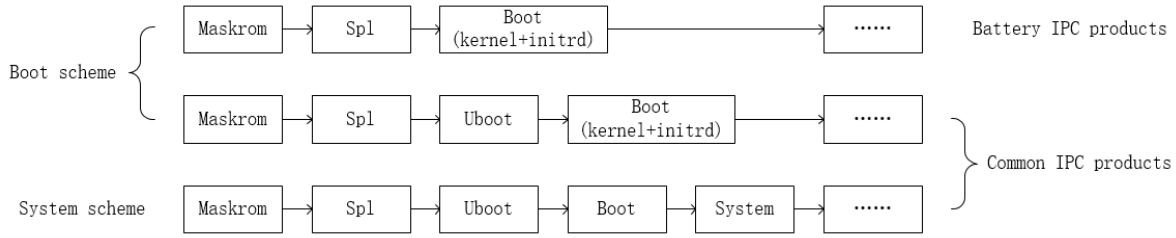
```
#::respawn:-/bin/sh
::sysinit:/etc/init.d/rcS

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

5.15 The Usage of A/B Systems

5.15.1 Introduction to Startup Schemes

The A/B systems support two startup schemes. The 'boot' scheme means that the boot partition contains the root file system (kernel+initrd). The 'system' scheme means that the root file system is separated (boot+system).

Currently, both startup schemes are supported in the common IPC products. Only the 'boot' scheme is supported in the battery IPC products.



5.15.2 Configuration

- Uboot

The Uboot configurations that need to be enabled for common IPC products and battery IPC products are different. The defconfig configurations are shown as follows respectively:

```
# Common IPC products
CONFIG_ANDROID_AB=y
CONFIG_SPL_MTD_WRITE=y
CONFIG_SPL_AB=y
CONFIG_EFI_PARTITION=y
CONFIG_SPL_EFI_PARTITION=y
CONFIG_AVB_LIBAVB_AB=y
CONFIG_AVB_LIBAVB_USER=y
CONFIG_RK_AVB_LIBAVB_USER=y
```

```
# Battery IPC products
CONFIG_SPL_MTD_WRITE=y
CONFIG_SPL_AB=y
CONFIG_EFI_PARTITION=y
CONFIG_SPL_EFI_PARTITION=y
```

- Kernel

The following configuration should be enabled **only when the 'boot' scheme uses the initramfs file system**. In other cases, no additional configuration is required in kernel.

```
# The kernel configuration of initramfs file system
CONFIG_BLK_DEV_INITRD=y
```

- Root File System

When **using SPI NAND as storage media**, ensure that the following shell tools are in the file system:

Tool Name	Tool Functions
flash_erase	Erase blocks of the specified MTD device.
nandwrite	Writes to the specified MTD device.
nanddump	Dumps the contents of a nand mtd partition.
md5sum	Compute and check MD5 message digest.
dd	Convert and copy a file.

- BoardConfig

Enable OTA in BoardConfig to compile OTA upgrade tool and package images for A/B systems switching and upgrading.

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

Modify the **partition Table** `RK_PARTITION_CMD_IN_ENV` and the **file system type** `RK_PARTITION_FS_TYPE_CFG`.

The followings provide examples of ‘boot’ and ‘system’ schemes for reference only, and parameters can be modified as required actually.

The 'boot' scheme:

```
# Partition table(Add the misc partition, remove the rootfs partition, and change
the boot partition to boot_a and boot_b partitions.)
export
RK_PARTITION_CMD_IN_ENV="64K(env),256K@64K(idblock),256K(uboot),64K(misc),384K(me
ta),6M(boot_a),6M(boot_b),1M(userdata)"

# File system type(By default, erofs is used for battery IPC products and
initramfs is used for common IPC products.)
export RK_PARTITION_FS_TYPE_CFG=boot_a@IGNORE@erofs,userdata@/userdata@jffs2
```

The 'system' scheme:

```
# Partition table(Add the misc partition, remove the rootfs partition, change the
uboot partition to uboot_a and uboot_b partitions, change the boot partition to
boot_a and boot_b partitions, and add system_a and system_b partitions.)
export
RK_PARTITION_CMD_IN_ENV="256K(env),256K@256K(idblock),256K(uboot_a),256K(uboot_b)
,256K(misc),4M(boot_a),4M(boot_b),16M(system_a),16M(system_b),32M(oem),32M(userda
ta),-(media)"

# File system type
export
RK_PARTITION_FS_TYPE_CFG=system_a@IGNORE@ubifs,oem@/oem@ubifs,userdata@/userdata@
ubifs
```

5.15.3 OTA Upgrade Tool

After adding the configuration `RK_OTA_RESOURCE` to BoardConfig, the SDK will compile the OTA upgrade tool into the `/usr/bin` of the firmware file system. The command is called `rk_ota`. At the same time, the corresponding images are packaged to the `<SDK>/output/image/update_ota.tar`.

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

5.15.4 A/B Systems Switching

The command for switching A/B systems is `rk_ota --misc=other`. The running result is as follows:

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

`--reboot`: Restart the device.

Enter the other system. If the system boots successfully, you can run the command `rk_ota --misc=now` to set the system as "the last boot system".

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

5.15.5 A/B Systems Upgrading

Options for upgrading A/B systems are as follows:

`--misc=update`: Select 'Upgrade' mode;

`--tar_path=<path>`: Set the path of the OTA upgrading package (tar package that stores the images);

`--save_dir=<path>`: (Optional) Set the unpacking folder. If it is not set, the default folder is `/mnt/sdcard/rk_update/`;

`--partition=<uboot/boot/system/all>`: (Optional) Set the partition to be upgraded. 'all' indicates that 'uboot', 'boot' and 'system' are upgraded. If it is not set, 'all' is the default value. (When there is no source file, the upgrade of the corresponding partition will be skipped.)

`--extra_part=<name>`: (Optional) Set a user-defined partition to be upgraded. If it is not set, this parameter is ignored by default.

Note : Currently, only the 'uboot', 'boot' and 'system' partitions and a user-defined partition can be upgraded.

For example:

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

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

```
Block size 131072, page size 2048, OOB size 128
Dumping data starting at 0x00000000 and ending at 0x002b7000...

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

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

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

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

[I/]RECOVERY Now is MTD.
[I/]RECOVERY Now is MTD.
mtd: successfully wrote block at 395a800000000
mtd: successfully wrote block at 395a800020000
reboot
```

5.16 Get the Camera Support Lists

The support list of camera sensor can be got from the Redmine https://redmine.rock-chips.com/documents/53

5.17 Get the Flash Support Lists

The support list of flash can be got from the Redmine https://redmine.rock-chips.com/documents/46

5.18 The Usage of the Stress Test

For the stress test function, the configuration needs to be enabled as follow:

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

The list of currently supported stress tests:

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

5.18.1 memtester Test

- Open the stress test list.

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

- Start the test. (Select the test item No. 1 in the stress test list.)
- Select the `memtester test` number. (By default, the test uses half of the free memory.)

5.18.2 stressapptest

- Open the stress test list.

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

- Start the test. (Select the test item No. 1 in the stress test list.)
- Select the `stressapptest` number. (By default, the test uses half of the free memory for 48 hours.)

5.18.3 cpufreq Test

- Open the stress test list.

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

- Start the test. (Select the test item No. 2 in the stress test list.)
- Select the `cpu freq stress test` or `cpu freq test:(with out stress test)` number. (By default, the former uses half of the free memory for 24 hours. The latter default frequency conversion once a second.)

5.18.4 Flash Stress Test

- Open the stress test list.

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

- Start the test. (Select the test item No. 3 in the stress test list.)

5.18.5 Reboot Test

- Open the stress test list.

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

- Start the test. (Select the test item No. 4 in the stress test list.)
- Quit the test. (By default, the test is stopped after restarting for 10000 times. To exit early, enter the following command.)

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

5.19 The Code and Document of Secure Boot

The path of encryption and decryption code: `media/security`

The path of security document: `docs/zh/security`

The path of U-Boot signature document (FIT section): `docs/zh/bsp/Rockchip_Developer_Guide_UBoot_Nextdev_CN.pdf`

5.19.1 Key

Run the following three commands in the U-Boot project to generate the RSA key pair for signature. Generally, the keys need to be generated only once. After that, **the key pair (dev.key, dev.pubkey) and self-signed certificate (dev.crt)** are used to sign and verify the firmware. Please keep it properly.

```
```

```
# 1. Perform operations in the 'rkbin/tools' directory
cd ./sysdrv/source/uboot/rkbin/tools

# 2. Use the "rk_sign_tool" of RK tools to generate the keys of RSA2048 named
'privateKey.pem' and 'publicKey.pem'.
./rk_sign_tool kk --bits 2048 --out .

# 3. Perform subsequent operations in the 'U-Boot' directory
cd ../../u-boot

# 4. The directory where the key is placed: keys
mkdir -p keys

# 5. Rename the keys to 'dev.key' and 'dev.pubkey' and store them in the 'keys'
directory.
cp ../rkbin/tools/private_key.pem keys/dev.key
cp ../rkbin/tools/public_key.pem keys/dev.pubkey

# 6. Generate a self-signed certificate using -x509 and the private key:
keys/dev.crt （The effect is essentially equivalent to the public key）
openssl req -batch -new -x509 -key keys/dev.key -out keys/dev.crt
```

`ls keys/` View the result:

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

Note: The names of "keys", "dev.key", "dev.crt" and "dev.pubkey" above are immutable. Because these names are already statically defined in the its file, packaging fails if changed.

5.19.2 U-Boot Configuration

The U-Boot defconfig configurations are shown as follows respectively:

```
# Mandatory option
CONFIG_FIT_SIGNATURE=y
CONFIG_SPL_FIT_SIGNATURE=y
CONFIG_ROCKCHIP_CIPHER=y
CONFIG_SPL_ROCKCHIP_CIPHER=y
CONFIG_CMD_HASH=y
CONFIG_SPL_ROCKCHIP_SECURE_OTP=y

# Optional (Read and write otp in U-Boot)
CONFIG_ROCKCHIP_SECURE_OTP=y    # Enable the U-Boot to read and write otp
CONFIG_MISC=y           # Compile configuration of related read and write
functions
# Optional (Rollback protect)
CONFIG_FIT_ROLLBACK_PROTECT=y       # boot.img rollback protect
CONFIG_SPL_FIT_ROLLBACK_PROTECT=y   # uboot.img rollback protect
```

5.19.3 Firmware Signature

The make.sh script in the U-Boot directory needs to be used for the firmware signature. The meanings of the supplementary parameters are as follows:

supplementary parameter	**meaning**
--spl-new	Pass this parameter to use the currently compiled spl file to package loader. Otherwise use the spl file in the rkbin project
--boot_img	Sign boot.img
--recovery_img	Sign recovery.img
CROSS_COMPILE=xxxx	Select the toolchain (For toolchain of different chips,please refers to Cross Toolchain Download and Installation)
--rollback-index-uboot	Uboot rollback protect (If the configuration is enabled, this parameter does not need to be added)
--rollback-index-boot	Boot rollback protect (If the configuration is enabled, this parameter does not need to be added)
--rollback-index-recovery	Recovery rollback protect
--burn-key-hash	Require the SPL phase to burn the public key hash to the OTP
[ini_path]	(battery IPC) The path of ini file in `rkbin/RKBOOT` (For details, see the uboot firmware compilation command.)

The firmware must be compiled before signing the firmware. For details about how to compile the firmware, see the compiling chapter in SDK Usage Introduction. Here are the signing steps:

```
# 1. Copy the firmware to the U-Boot directory (for example, 'boot.img', similar
to 'recovery.img')
cp ./output/image/boot.img ./sysdrv/source/uboot/u-boot

# 2. Perform operations in the U-Boot directory
cd ./sysdrv/source/uboot/u-boot

# 3. Sign the firmware. The command format is as follows:
# ./make.sh --spl-new [--boot_img <boot image name>] [--recovery_img <recovery
image name>] CROSS_COMPILE=<cross toolchain> --burn-key-hash
# For example:
./make.sh --spl-new --boot_img boot.img CROSS_COMPILE=arm-rockchip830-linux-
uclibcgnueabihf- --burn-key-hash
```

Note 1: If the 'make.sh' script is used to generate the firmware, add parameters as required.

Note 2: When adding the `--burn-key-hash` parameter, it will automatically write the public key hash to OTP and enable secure boot. Do not add this parameter if you want to write the hash to OTP and enable secure boot by yourself.

Note 3: The signed firmwares are stored in the U-Boot directory. And the firmware names are 'xxx_download_xxx.bin', 'xxx_idblock_xxx.img', 'uboot.img', 'boot.img' (if any) and 'recovery.img' (if any).

5.20 How to Use rndis

Modify BoardConfig.mk, add `export RK_ENABLE_RNDIS=y`, the command building as follow:

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

Flash firmware and run this command on board: `rndis.sh`
The details as follow:

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

Notice: If want to modify IP address default, we can modify the file of SDK named "sysdrv/tools/board/rndis/rndis.sh", and compiling and flash firmware again.

5.21 How to Optimize the Booting Time of SPI NOR

The process of system boot-up: Power On --> Maskrom --> idblock --> uboot --> kernel
The Maskrom will read the bootconfig parameter which is in idblock image, and then boot-up with Quad SPI mode (default is Single SPI mode).
There are two methods to configure Maskrom working in Quad SPI mode:

1. Enable `CONFIG_MTD_SPI_NOR_MISC=y` in the defconfig of kernel，Run the command of `idb_bootconfig /dev/mtdblock1` after system booting.
   (The parameter of `/dev/mtdblock1` is the mtd device node of idblock partition, idb_bootconfig code is located in `sysdrv/tools/board/idb_bootconfig/idb_bootconfig.c`)
2. Use the Rockchip's USB firmware flashing tool to flash idblock.img.

Notice: The idb_bootconfig is ONLY applicable to RV1106/RV1103 platform at present.

5.22 How to Add Non-root User Login

This section is going to introduce how to add non-root users to login in to a terminal when using busybox.

1. Modify /etc/passwd

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

x represents user has set a password
1010 represents user's id
1011 represents user's group id
/opt represents the HOME directory of test123
Note: For detailed /etc/passwd instructions, please refer to https://www.man7.org/linux/man-pages/man5/passwd.5.html

2. Modify /etc/shadow

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

`$1$x.YlInZQ$N.kbNTIEOkBjmlfftSVFs0:10933` is generated by the command of `mkpasswd -m "md5" "rockchip123"`.
Note: For detailed /etc/shadow instructions, please refer to https://www.man7.org/linux/man-pages/man5/shadow.5.html

3. Modify /etc/group

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

Note: For detailed /etc/group instructions, please refer to https://www.man7.org/linux/man-pages/man5/group.5.html

4. Modify /etc/inittab

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

5.23 How to Add New Camera Sensor

For example, add sc530ai sensor:

- Add sensor codes in the dir of sysdrv/source/kernel/drivers/media/i2c

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

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

- Add the codes for Kconfig and Makefile

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

```
config VIDEO_SC530AI
        tristate "SmartSens SC530AI sensor support"
        depends on I2C && VIDEO_V4L2
        select MEDIA_CONTROLLER
        select VIDEO_V4L2_SUBDEV_API
        select V4L2_FWNODE
        help
          This is a Video4Linux2 sensor driver for the SmartSens
          SC530AI camera.

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

obj-$(CONFIG_VIDEO_SC530AI) += sc530ai.o
```

- Enable building sensor in the defconfig to build the file of driver ko

```
# e.g. rv1106-uvc-spi-nor.config
sysdrv/source/kernel$ vi arch/arm/configs/rv1106-uvc-spi-nor.config

CONFIG_VIDEO_SC530AI=m
```

- Add camera config for dts file

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

&csi2_dphy_hw {
    status = "okay";
};

&csi2_dphy0 {
    status = "okay";

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

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

            csi_dphy_input0: endpoint@0 {
                reg = <0>;
                remote-endpoint = <&sc530ai_out>;
                data-lanes = <1 2>;     //Notice: if use 4 lane, data-lanes = <1
2 3 4>;
            };
        };

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

            csi_dphy_output: endpoint@0 {
                reg = <0>;
```

```
                remote-endpoint = <&mipi_csi2_input>;
            };
        };
    };

};

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

    sc530ai: sc530ai@30 {                       //30 and reg = <0x30> stand for
the i2c address of sensor
        compatible = "smartsens,sc530ai";       //match with the field of
driver's sc530ai_of_match
        status = "okay";
        reg = <0x30>;                           //i2c address
        clocks = <&cru MCLK_REF_MIPI0>;
        clock-names = "xvclk";
        reset-gpios = <&gpio3 RK_PC5 GPIO_ACTIVE_HIGH>;
        pwdn-gpios = <&gpio3 RK_PD2 GPIO_ACTIVE_HIGH>;
        pinctrl-names = "default";
        pinctrl-0 = <&mipi_refclk_out0>;
        rockchip,camera-module-index = <0>;
        rockchip,camera-module-facing = "back";
        rockchip,camera-module-name = "CMK-OT2115-PC1";   //module name
        rockchip,camera-module-lens-name = "30IRC-F16";  //module lens,
<sensor_name>_<module_name>_<module_lens>.json --> sc530ai_CMK-OT2115-PC1_30IRC-
F16.json
        port {
            sc530ai_out: endpoint {
                remote-endpoint = <&csi_dphy_input0>;
                data-lanes = <1 2>;     //Notice: if use 4 lane, data-lanes = <1
2 3 4>;
            };
        };
    };

};

&mipi0_csi2 {
    status = "okay";

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

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

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

```
        };

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

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

&rkcif {
    status = "okay";
};

&rkcif_mipi_lvds {
    status = "okay";

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

&rkcif_mipi_lvds_sditf {
    status = "okay";

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

&rkisp {
    status = "okay";
};

&rkisp_vir0 {
    status = "okay";

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

- Add auto load driver ko while boot-up

Add the insmod command in the file of `sysdrv/drv_ko/insmod_ko.sh`.
If config the `RK_POST_BUILD_SCRIPT` in the BoardConfig.mk, and add the insmod command in RK_POST_BUILD_SCRIPT.

```
# e.g. BoardConfig_SmartDoor
project/cfg/BoardConfig_SmartDoor$ vi rv1106-tb-post.sh

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

- Config Sensor IQ file in the BoardConfig.mk

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

Sensor IQ file is in the dir of `media/isp/release_camera_engine_rkaiq*`

```
media/isp/release_camera_engine_rkaiq_rv1106_arm-rockchip830-linux-
uclibcgnueabihf/isp_iqfiles/$ ls

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

Notice:
a. The json format of Sensor IQ file can be converted to the bin format by the tool named j2s4b (media/isp/release_camera_engine_rkaiq_*/host/j2s4b).
b. The document for Camera sensor is the dir of /docs/zh/isp

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

5.24 How to Reboot into The Terminal of U-Boot

Take RV1106 fox example.

Directory: sysdrv/source/kernel

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

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

```
index 1436e1d32619..cc5a421aef26 100644
--- a/include/dt-bindings/soc/rockchip,boot-mode.h
+++ b/include/dt-bindings/soc/rockchip,boot-mode.h
@@ -21,4 +21,6 @@
 /* enter usb mass storage mode */
 #define BOOT_UMS                (REBOOT_FLAG + 12)

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

Directory: sysdrv/source/uboot/u-boot

```
diff --git a/arch/arm/include/asm/arch-rockchip/boot_mode.h
b/arch/arm/include/asm/arch-rockchip/boot_mode.h
index bc1395ee2c..d68099a94e 100644
--- a/arch/arm/include/asm/arch-rockchip/boot_mode.h
+++ b/arch/arm/include/asm/arch-rockchip/boot_mode.h
@@ -26,6 +26,8 @@
 /* enter bootrom download mode */
 #define BOOT_BROM_DOWNLOAD      0xEF08A53C

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

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

```
+       case BOOT_MODE_UBOOT_TERMINAL:
+               printf("enter uboot!\n");
+               env_set("cli", "yes");
+               break;
        }

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

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

Notice: The value of **BOOT_TO_UBOOT** in the kernel MUST be matched with it in the U-Boot.

5.25 How to Support USB Mass Storage in U-Boot

Add **rv1106-usb.config** to **RK_UBOOT_DEFCONFIG_FRAGMENT** in the **BoardConfig.mk**.(Other platforms can also be referenced)

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

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

 # Kernel defconfig
 export RK_KERNEL_DEFCONFIG=rv1106_defconfig
```

The configs of **rv1106-usb.config**:

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

6. Notices

When copying the source code package under Windows, the executable file under Linux may become a non-executable file, or the soft link fails and cannot be compiled and used.
Therefore, please be careful not to copy the source code package under Windows.