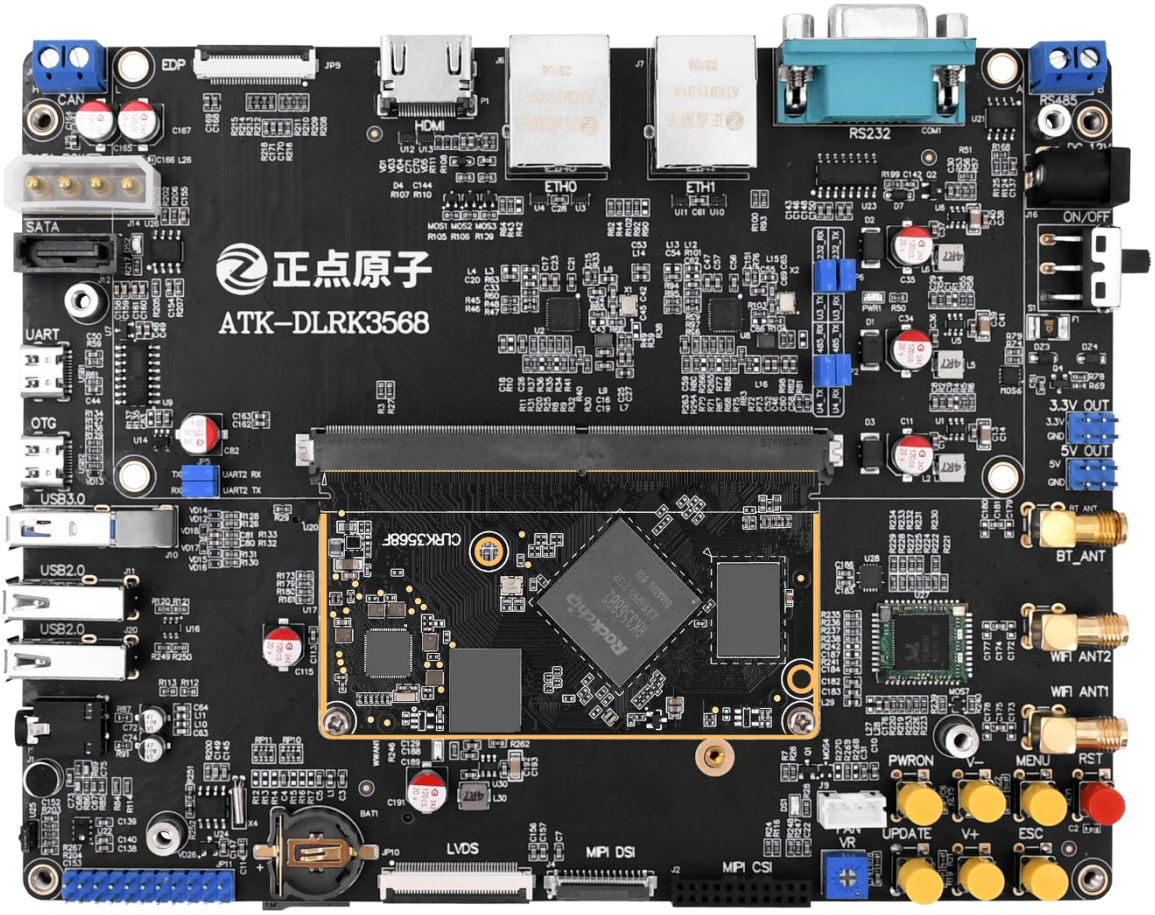
ATK-DLRK3568

Android System User Manual

V1.2



ALIENTEK

1. Shopping:

TMALL: https://zhengdianyuanzi.tmall.com

TAOBAO: https://openedv.taobao.com

2. Download

Address: http://www.openedv.com/docs/index.html

3. FAE

Website : www.alientek.com

Forum : http://www.openedv.com/forum.php

Videos : www.yuanzige.com

Fax : +86 - 20 - 36773971

Phone : +86 - 20 - 38271790

Disclaimer

The product specifications and instructions mentioned in this document are for reference only and subject to update without prior notice; Unless otherwise agreed, this document is intended as a product guide only, and none of the representations made herein constitutes a warranty of any kind. The copyright of this document belongs to Guangzhou Xingyi Electronic Technology Co., LTD. Without the written permission of the company, any unit or individual shall not be used for profit-making purposes in any way of dissemination.

In order to get the latest version of product information, please regularly visit the download center or contact the customer service of Taobao ALIENTEK flagship store. Thank you for your tolerance and support.

Revision History:

Version	Version Update Notes	Responsible person	Proofreading	Date
V1.0	release officially	ALIENTEK	ALIENTEK	2023.06.25
V1.1	① Add a document explaining Android 11 in the preface section; ② In the 4.1.1 subsection, use "\ " for line breaks when specifying the installation command; ③ In the 6.2.2 subsection, modify the kernel source code configuration command;	ALIENTEK	ALIENTEK	2023.08.30
V1.2	Add: 6.1.3 Section, 6.2.2 Section	ALIENTEK	ALIENTEK	2023.11.01

Catalogue

Brief 1
Overview of Embedded Development Environment 2
Chapter 1. Install the Ubuntu system 4
1.1 Installing VMware Virtual Machine Software 4
1.1.1 Download the VMware software 4
1.1.2 Installing VMware software 7
1.2 Installing the Ubuntu system 13
1.2.1 Download the Ubuntu system image 13
1.2.2 Create a virtual machine 15
1.2.3 Install Ubuntu system 26
Chapter 2. Development environment setup 41
2.1 Ubuntu System Settings 41
2.1.1 Set the root user password 41
2.1.2 Change the software download source 42
2.1.3 Disable automatic updates 46
2.2 File Transfer Between Ubuntu and Windows 47
2.2.1 Setting up an FTP Server on Ubuntu 47
2.2.2 Installing FTP client on Windows 47
2.2.3 How to use FileZilla 52
2.3 Setting up the tftp server in the Ubuntu system 57
2.4 Set up an NFS server on the Ubuntu system 57
2.5 Set up an SSH server on the Ubuntu system 57
2.6 Installation of CH340 serial port driver 58
2.7 Installation of MobaXterm software 60
2.7.1 Download of MobaXterm software 60
2.7.2 Installation of MobaXterm software 61
2.7.3 Usage of MobaXterm software 65
2.8 Installation of Rockchip USB Driver 72
2.9 Usage of Mirror Image Burning Tools 74
2.9.1 Introduction to Burning Modes 74
2.9.2 Windows flashing tool RKDevTool 76
2.9.3 Method for burning update.img image 82
2.9.4 Erase operation 84
2.9.5 Usage of the Linux_Upgrade_Tool tool under Ubuntu 85
2.10 Installation of ADB tool 85
Chapter 3. Introduction to the ATK-DLRK3568 Platform 87
3.1 Introduction to RK3568 87
3.2 Introduction to the Hardware Resources of the ATK-DLRK3568 Development Board 87
Chapter 4. RK3568 Android SDK software package 88
4.1 Installing the RK3568 Android SDK 88
4.1.1 Install dependent software packages 88

4.1.2 Install repo (skip) 89
4.1.3 Git configuration 89
4.1.4 Install SDK 90
4.1.5 SDK update 92
4.1.6 SDK Issue Feedback 92
4.2 Introduction to SDK Software Architecture 92
4.2.1 Introduction to the Android SDK Project Directory 92
4.2.2 Software Block Diagram of the SDK 94
4.2.3 SDK Version Query 97
4.3 SDK Automatic Compilation 98
4.4 Image Introduction 105
4.5 Independent Compilation 107
4.5.1 Independent Compilation of U-Boot 107
4.5.2 Independently compiling Kernel 108
4.5.3 Independently compiling Android source code 111
4.5.4 Package into update.img image 119
4.6 SDK cleanup 121
Chapter 5. SDK image burning 122
5.1 Flash Mode Introduction 123
5.2 Flashing on Windows System 123
5.2.1 Introduction to Partition Table parameter.txt 124
5.2.2 Configuration 126
5.2.3 Flashing 128
5.2.4 Booting the System 130
5.2.5 Flashing update.img 130
5.3 Flashing in Ubuntu system 131
5.3.1 Connect the USB device to Ubuntu 132
5.3.2 Burning with the upgrade_tool tool 133
5.3.3 Flashing update.img 135
5.3.4 Erase Operation 136
Chapter 6. Android SDK Development 137
6.1 U-Boot development 138
6.1.1 Device Tree of U-Boot 138
6.1.2 U-Boot Compilation 138
6.1.3 Configuration of U-Boot 154
6.1.4 The defconfig configuration file 155
6.1.5 Shortcuts 158
6.1.6 HW-ID DTB 159
6.2 kernel development 162
6.2.1 Kernel compilation 162
6.2.2 Kernel Configuration 172
6.2.3 Device tree and defconfig configuration file 173
6.2.4 IO Power Domain 178

6.2.5 Replace the logo....186
6.2.6 Summary of Kernel Module Development Documents....187
6.3 Android System Development....189
6.3.1 Android System Architecture....189
6.3.2 Android product configuration....192
6.3.3 Android Source Code Compilation....206
6.3.4 Shutting down SELinux....256
6.3.4.1 Introduction to SELinux/SEAndroid....256
6.3.4.2 Shutting down SELinux....257
6.3.5 Application Pre-installation Function....258
6.3.6 Modify default language and time zone information....260
6.3.7 Set the system to default not to lock the screen or enter sleep mode....261
6.3.8 Modify the default backlight brightness....261
6.3.9 Modify the Boot Logo....262
6.3.10 Modify Boot Animation....263
6.3.11 Modify the main screen / secondary screen....267
6.3.12 Change screen orientation....273
6.3.12.1 Changing the display direction of the main screen....274
6.3.12.2 Modify the display direction of the secondary screen....277
6.3.12.3 Rotation of the screen in recovery mode....278
6.3.13 Adding Custom Modules....279
6.3.14 Android 11 obtains root access....281
6.3.15 Multi-screen Different Display and Touch....282
6.3.16 Multi-screen Different Sound....282
6.3.17 Three-screen Display....282
6.4 Android Debugging....282
6.4.1 ADB....282
6.4.2 logcat....283
6.5 Android Application Development....285
6.6 Secure Boot/Safety Mechanism Development....285
6.7 A/B System Development....286
6.8 System Upgrade Development....286

Brief

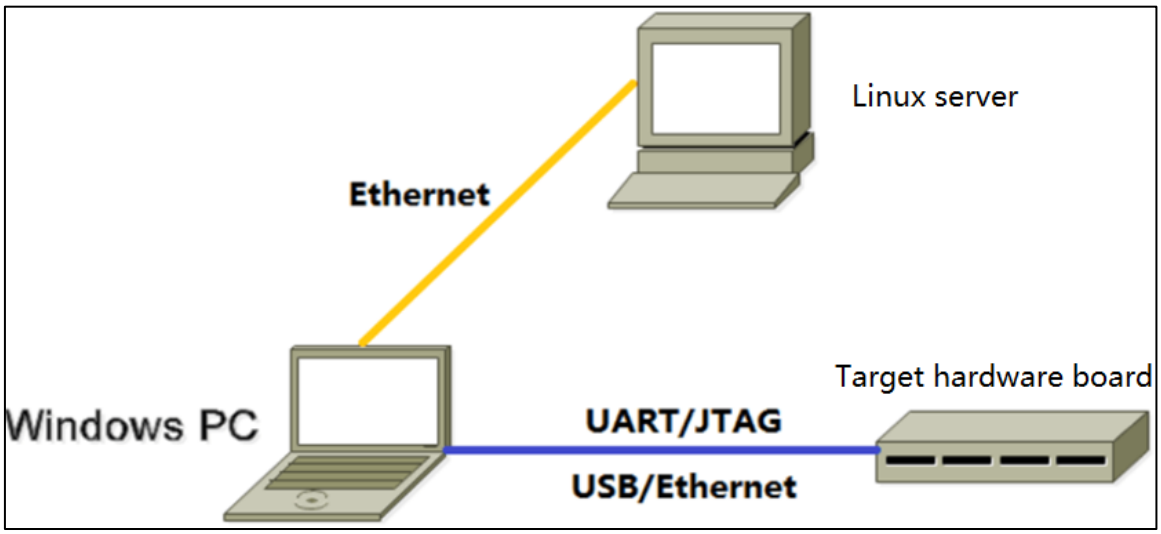
This document will provide users with information on Android system development for the ATK-DLRK3568 development platform, including the following sections:

1. Installing the Ubuntu operating system;
2. Setting up the development environment;
3. Introduction to the ALIENTEK ATK-DLRK3568 development platform;
4. RK3568 Android SDK software package;
5. Burning the SDK image;
6. SDK development.

This document is written based on the RK3568 platform and Android 11 system! There might be some differences on Android 12 system! However, the majority of the content remains the same.

Overview of Embedded Development Environment

A typical embedded development environment usually consists of a Linux server, a Windows PC and a target hardware board. The typical development environment is shown in the following figure:



- Establish a cross-compilation environment on the Linux server to provide code update download and cross-compilation services for software development.
- Install remote terminal software on the Windows PC, such as Putty, SecureCRT or MobaXterm, and remotely log in to the Linux server via the network (using ssh remote login) to perform cross-compilation (copying the generated image file through the network to the Windows PC), as well as code development and debugging.
- The Windows PC can be connected to the target hardware board via the serial port and USB. The compiled image file can be burned onto the target hardware board (usually through USB or JTAG burner for burning), and the system or application can be debugged.

◆ **Virtual machine Linux system + Windows PC + target hardware board**

For many developers, there might not be a Linux server or the company might not provide a Linux server specifically for cross-compilation of code. You only have a Windows computer. In such cases, you can install virtual machine software (such as VMware, Virtual Box or Virtual PC) on the Windows PC. Use the virtual machine software to create a virtual machine on the Windows PC, install the Linux operating system on the virtual machine. Use the virtual machine Linux system to replace the Linux server.

Nowadays, many chip platforms support directly writing image files to target hardware devices in the Linux system environment, such as the Rockchip platform. Using the virtual machine Linux system is more convenient because the virtual machine Linux system and the Windows system are on the same physical machine and share the hardware resources of the physical machine. The target hardware device is connected to the PC via serial port and USB. The Windows system can detect the device connection, and the virtual machine Linux system can also detect the device connection (usually you can choose whether to connect the device to the Windows system or the virtual machine Linux system). Therefore, you can directly write the image file to the target hardware device in the virtual machine Linux system

environment without copying the image file from the Linux system to the Windows system and then burning it by the Windows system, which greatly facilitates development!

Chapter 1. Install the Ubuntu system

Embedded development is generally carried out in a dual system development environment of Windows and Ubuntu. This chapter will introduce to you how to install the Ubuntu operating system on a virtual machine.

This chapter will be divided into the following sections:

1.1 Installing VMware virtual machine software

1.2 Installing the Ubuntu system

1.1 Installing VMware Virtual Machine Software

VMware (Vmware Workstation) is a highly powerful virtual machine software. With this software, we can create one or more virtual machines in the Windows system. A virtual machine (Virtual Machine) refers to a computer system simulated by software, which has the complete hardware system functions of a physical computer, such as memory, hard disk, and processor. Various operating systems supported by physical computers can be installed in virtual machines, such as Windows operating systems, DOS, and Linux operating systems (Ubuntu, Debian, CentOS, etc.). Any work that can be done on a physical computer can also be achieved in a virtual machine, and virtual machines can be operated just like physical computers.

Currently popular virtual machine software includes VMware (Vmware Workstation), Virtual Box, and Virtual PC, etc. This document will guide users on how to create a virtual machine and install the Ubuntu operating system through VMware software.

1.1.1 Download the VMware software

The development board data package has already provided the installation package file of VMware virtual machine software for everyone. The path is: Development Board CD-ROM A Drive - Basic Materials→4_softwares→VMware-workstation-full-16.2.1-18811642.exe. If users do not want to waste time downloading, they can directly use this installation package file and proceed to the next 1.1.2 section.

Users can also download the current latest version of the VMware Workstation installation package file through the link address: https://www.vmware.com/products/workstation-pro/workstation-pro-evaluation.html

Figure 1.1-1 Download of the latest version installation package for VMware virtual machine software

The above link address can only download the current latest version of the installation package file. If users need to download the older version of the installation package file, they can do so through the link address: https://customerconnect.vmware.com/cn/downloads/#all_products.

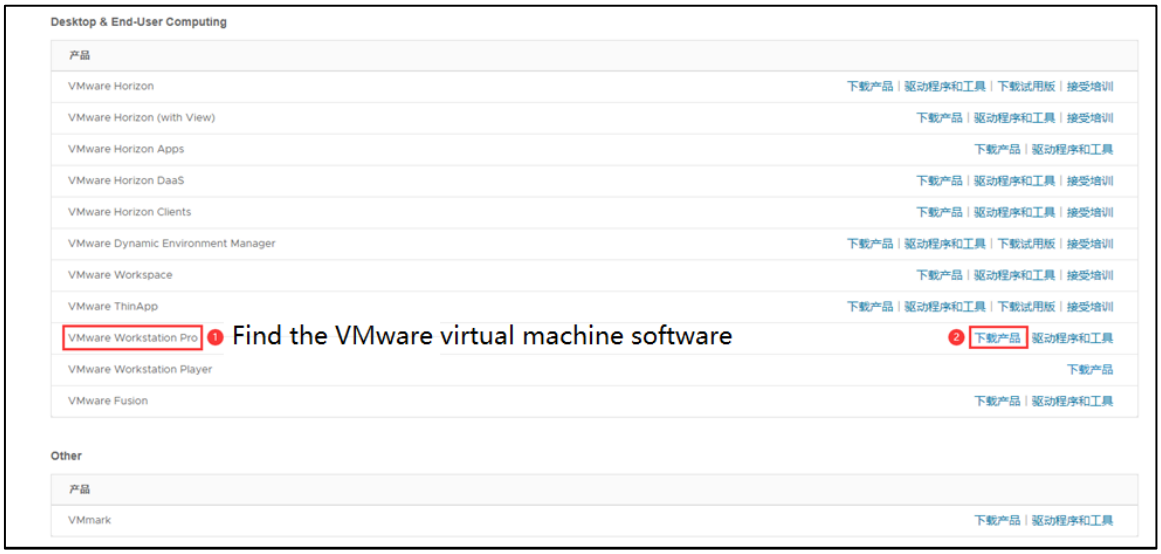


Figure 1.1-2 Download of the old version VMware virtual machine software installation package 1

Figure 1.1-3 Download of the old version VMware virtual machine software installation package 2

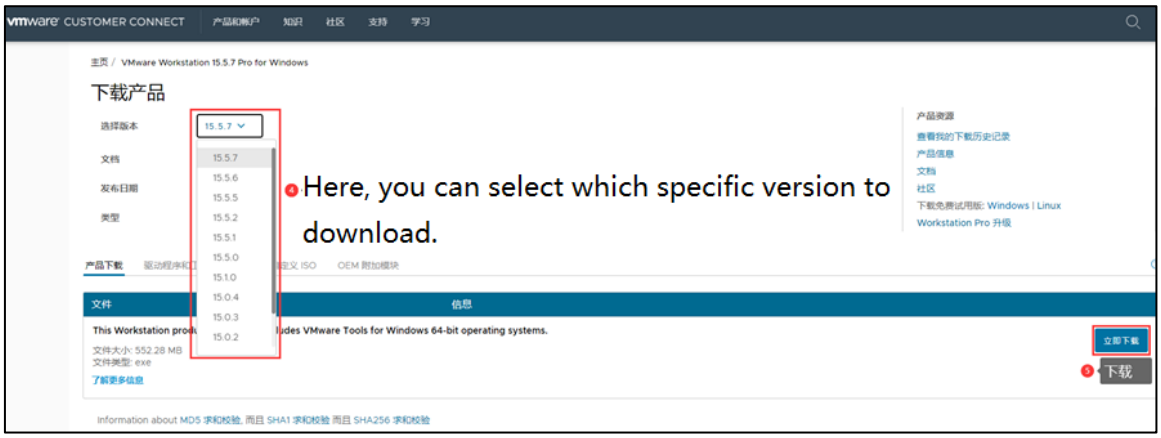


Figure 1.1-4 Download of the old version VMware virtual machine software installation package 3

The download process requires users to log in to their VMware account. If you don't have a VMware account, you can register through the VMware website at https://www.vmware.com/cn.html.

There are no specific requirements for the version of the VMware virtual machine software. The latest version is fine, and older versions (such as 15.X.X or 14.X.X) are also acceptable. This document will use version 16.2.1 as an example, and the corresponding installation package file name is VMware-workstation-full-16.2.1-18811642.exe.

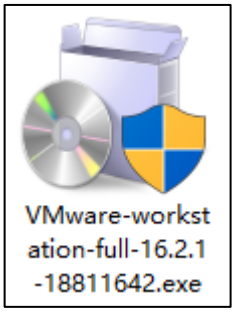


Figure 1.1-5 The installation package file of the VMware virtual machine software version

1.1.2 Installing VMware software

Installing VMware software on Windows is very simple. Just double-click to run the installation package file VMware-workstation-full-16.2.1-18811642.exe, and follow the operation steps shown in Figures 1.1.2.1 to 1.1.2.8 to install the VMware virtual machine software:

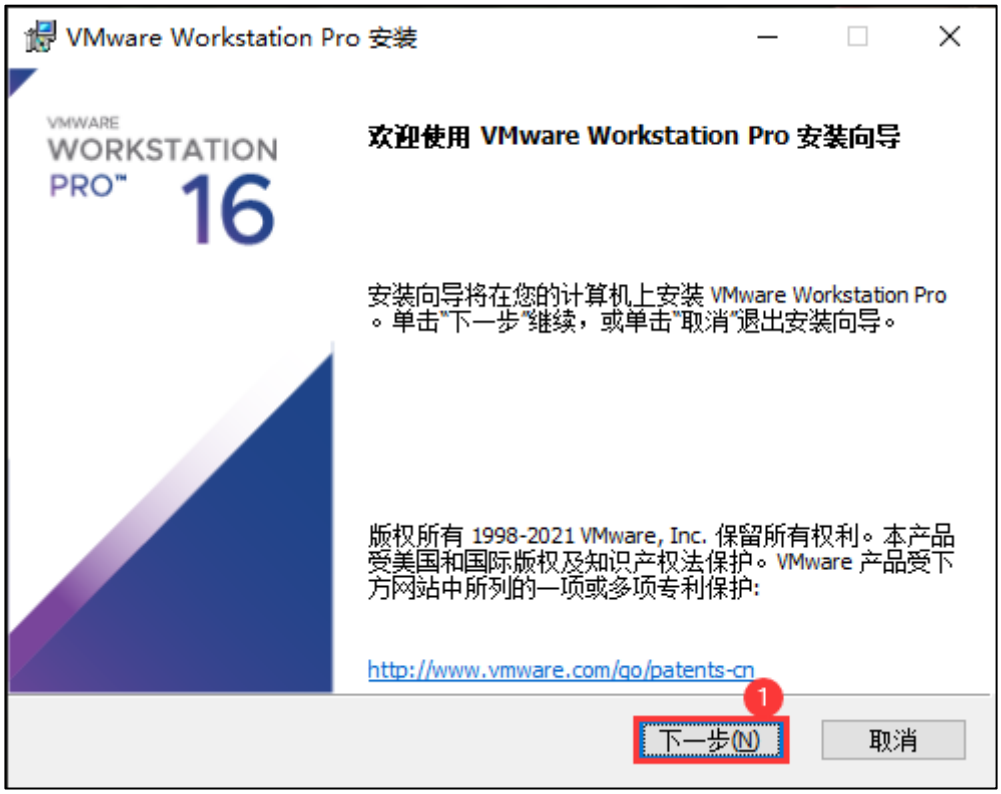


Figure 1.1.2.1 Install VMware software 1

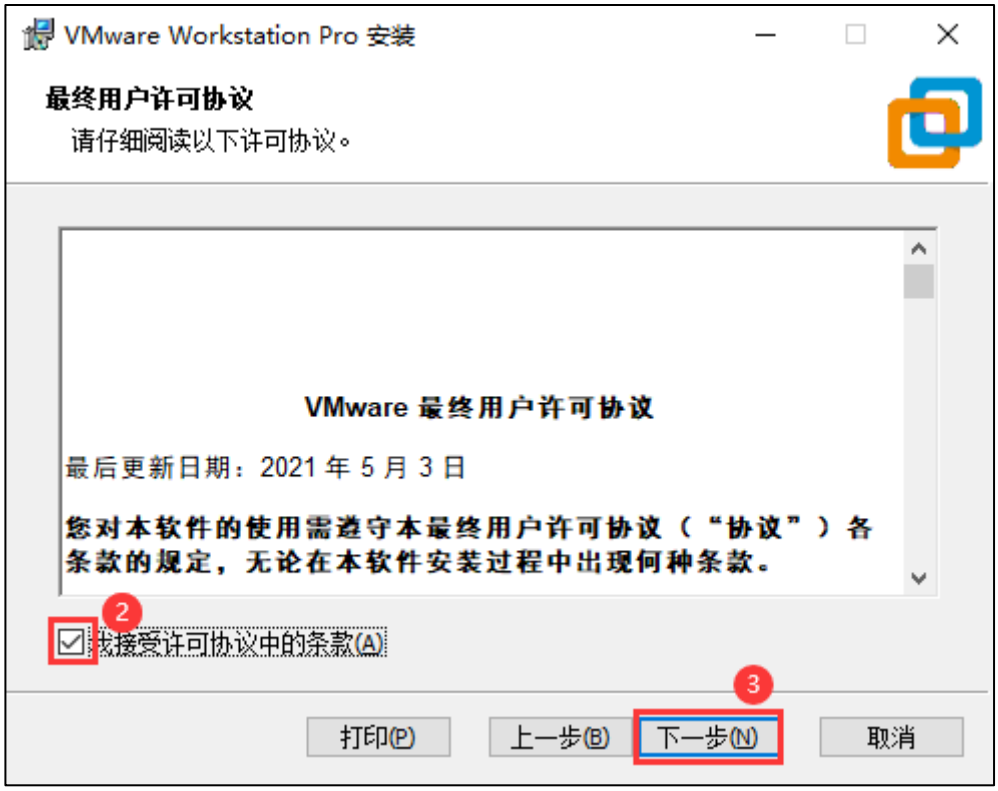


Figure 1.1.2.2 Install VMware software 2

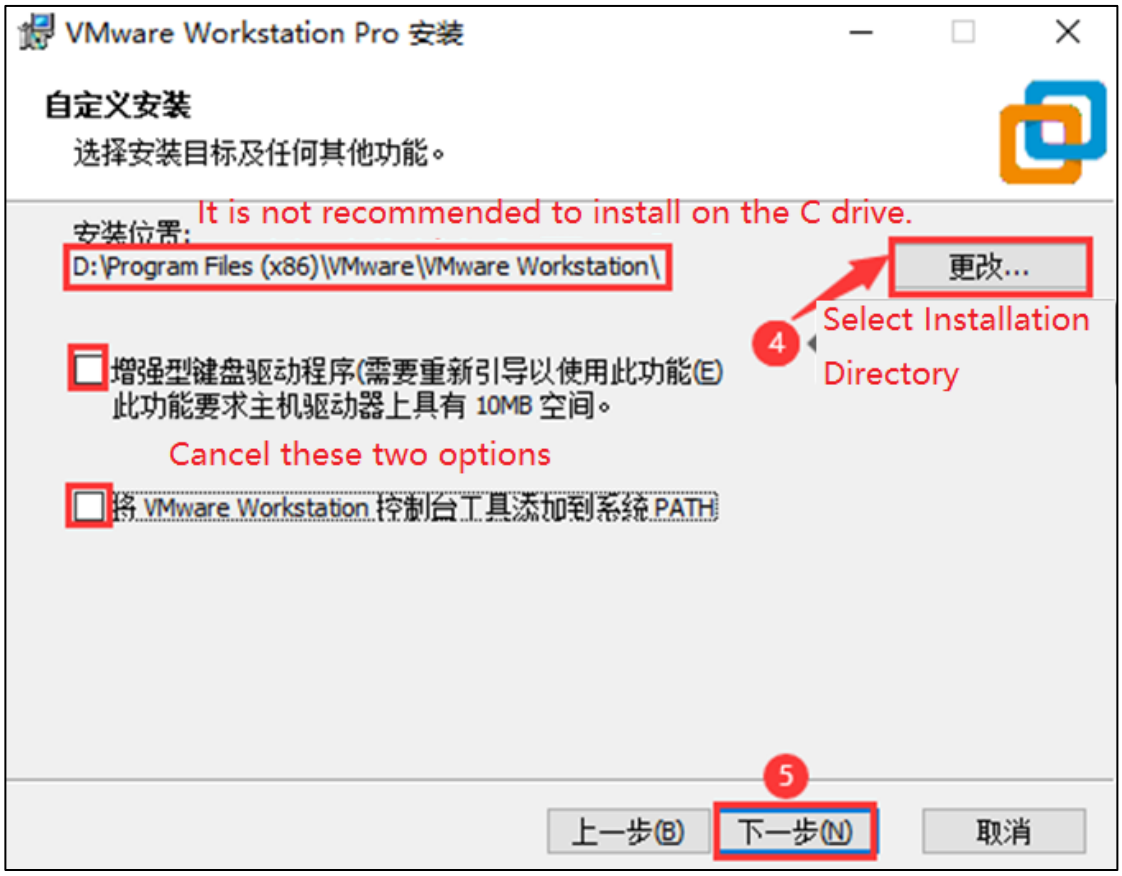


Figure 1.1.2.3 Install VMware software 3

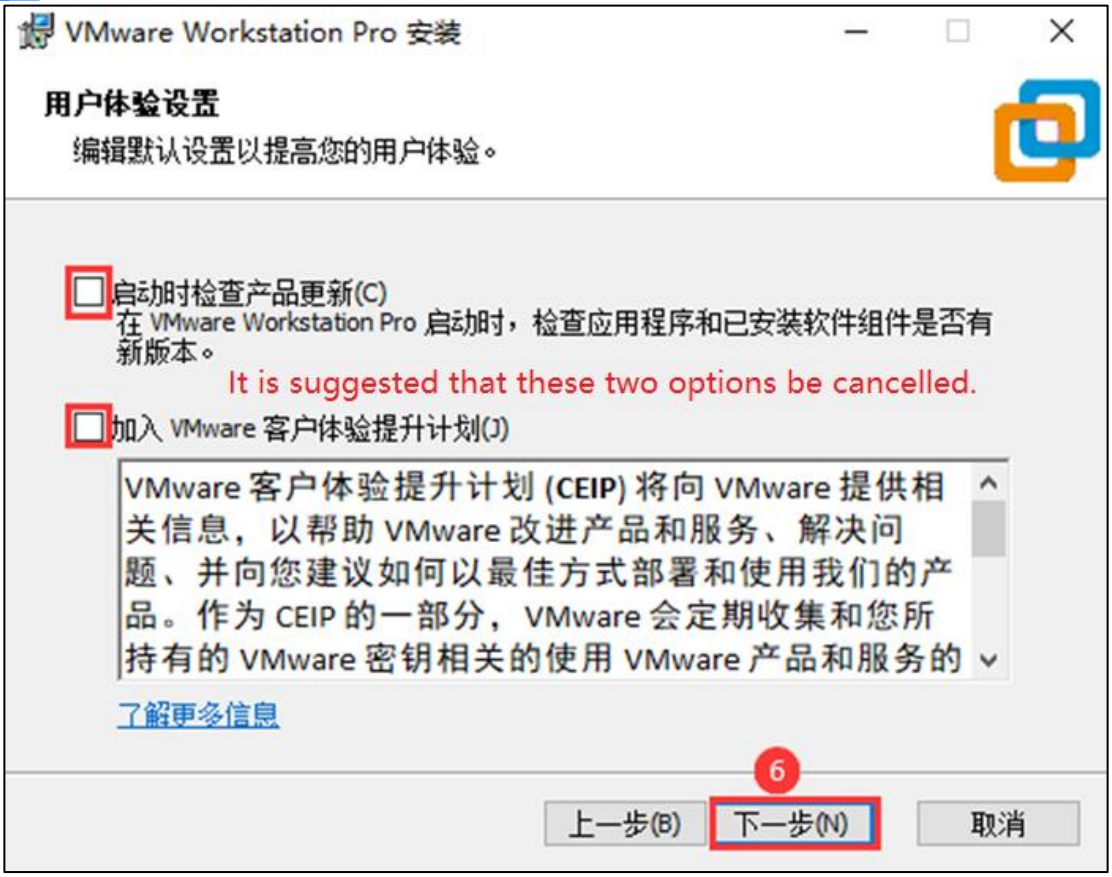


Figure 1.1.2.4 Install VMware software 4

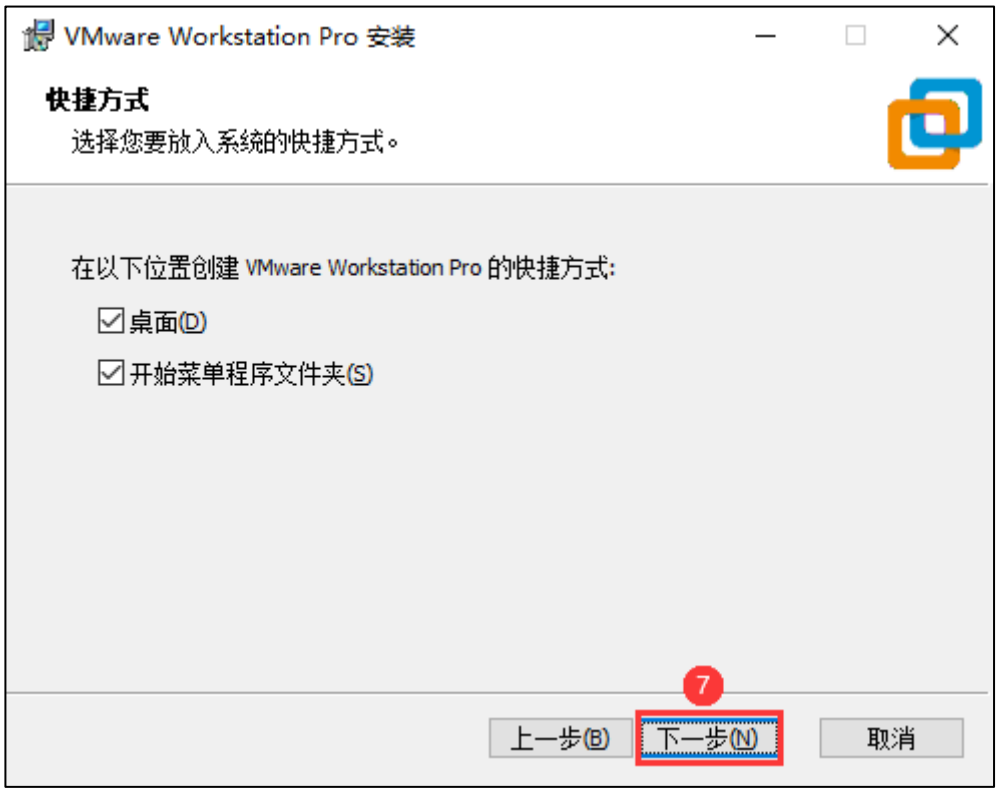


Figure 1.1.2.5 Install VMware software 5

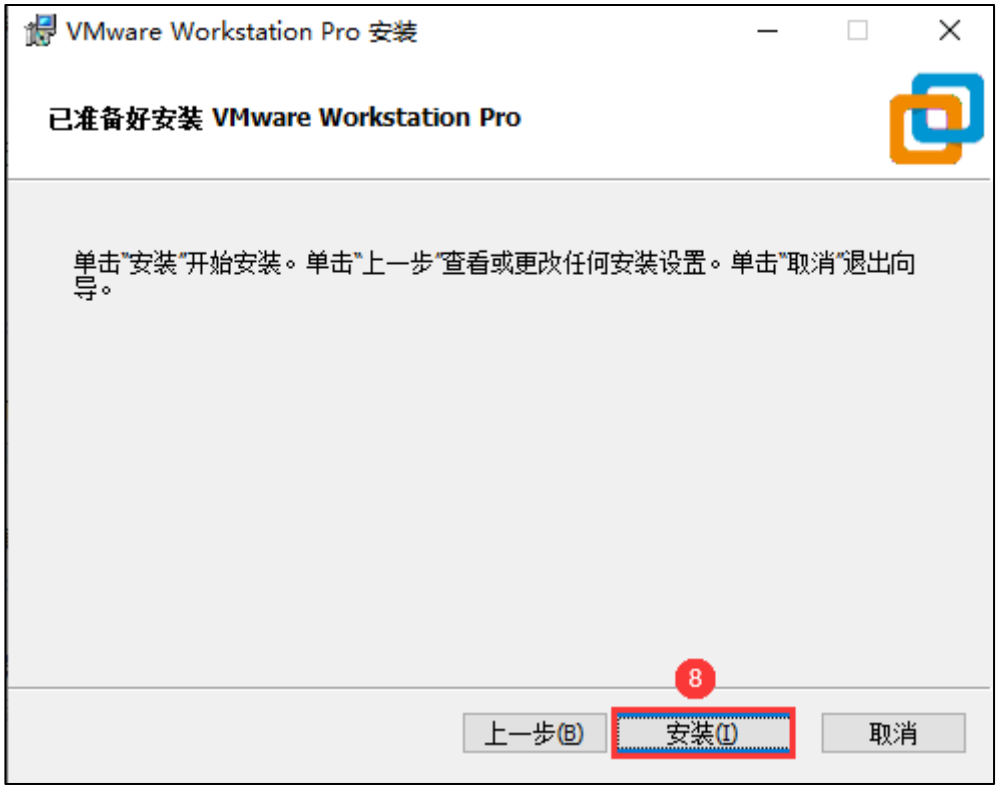


Figure 1.1.2.6 Install VMware software 6

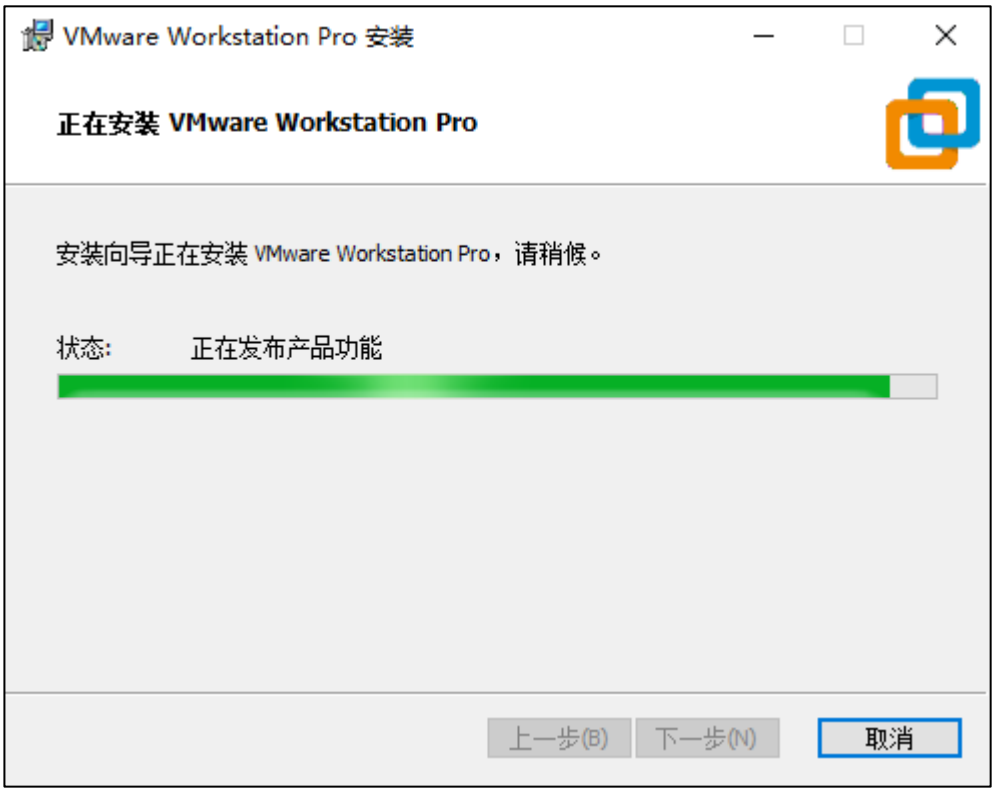


Figure 1.1.2.7 Install VMware software 7

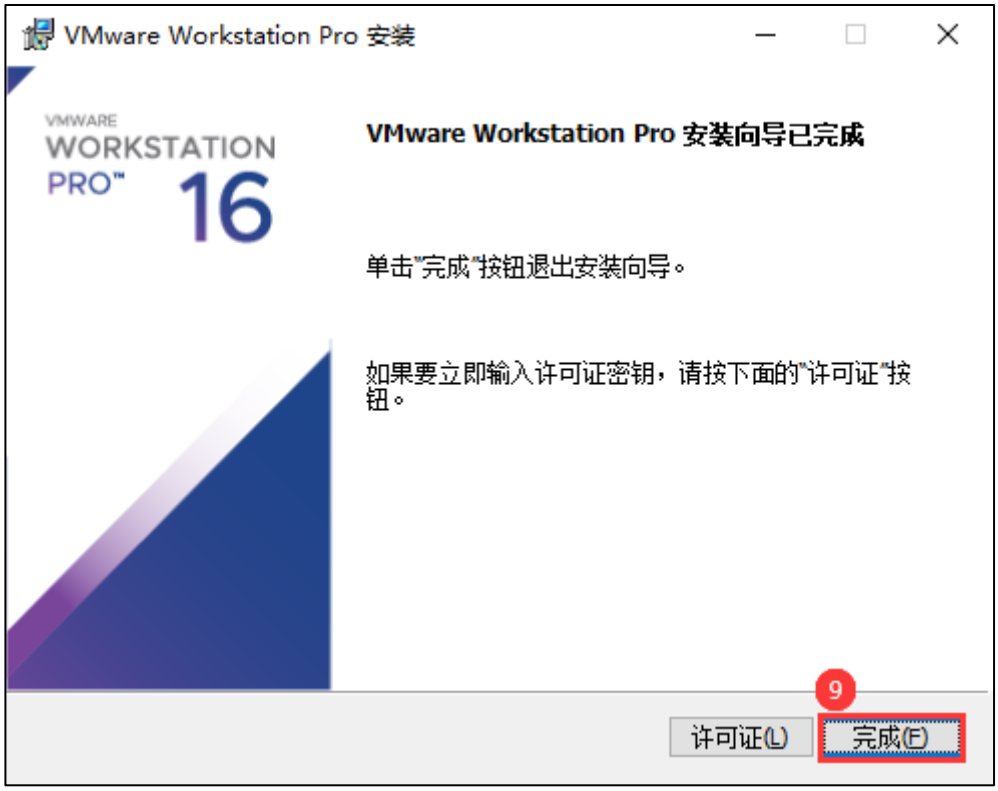


Figure 1.1.2.8 Install VMware software 8

Click the "Finish" button. At this point, the VMware virtual machine software installation is complete.

After the installation is finished, a shortcut icon of the VMware virtual machine software will be generated on the Windows system desktop, as shown in Figure 1.1.2.9:

Figure 1.1.2.9 VMware software desktop shortcut icon

Double-click the icon to open the VMware virtual machine software. When opening the software for the first time, it will prompt the user to enter the license key, as shown in Figure 1.1.2.10:

Figure 1.1.2.10 Enter the license key

VMware virtual machine software is a paid software and requires users to purchase it before using; if a user purchases the VMware software license, they will receive a series of license keys. Fill in the license key in the corresponding input box labeled "**I have the VMware Workstation 16 license key (H)**" to activate it. Once activation is successful, the VMware software can be used normally.

If the user has not purchased the software license, they can choose the option "**I wish to use VMware Workstation 16 for 30 days (W)**". In this way, they can obtain a 30-day trial period for the VMware software. Click the "Continue" button:

Figure 1.1.2.11 Welcome to use VMware software.

Click the "Finish" button, then you will enter the main interface of VMware virtual machine software, as shown in Figure 1.1.2.12:

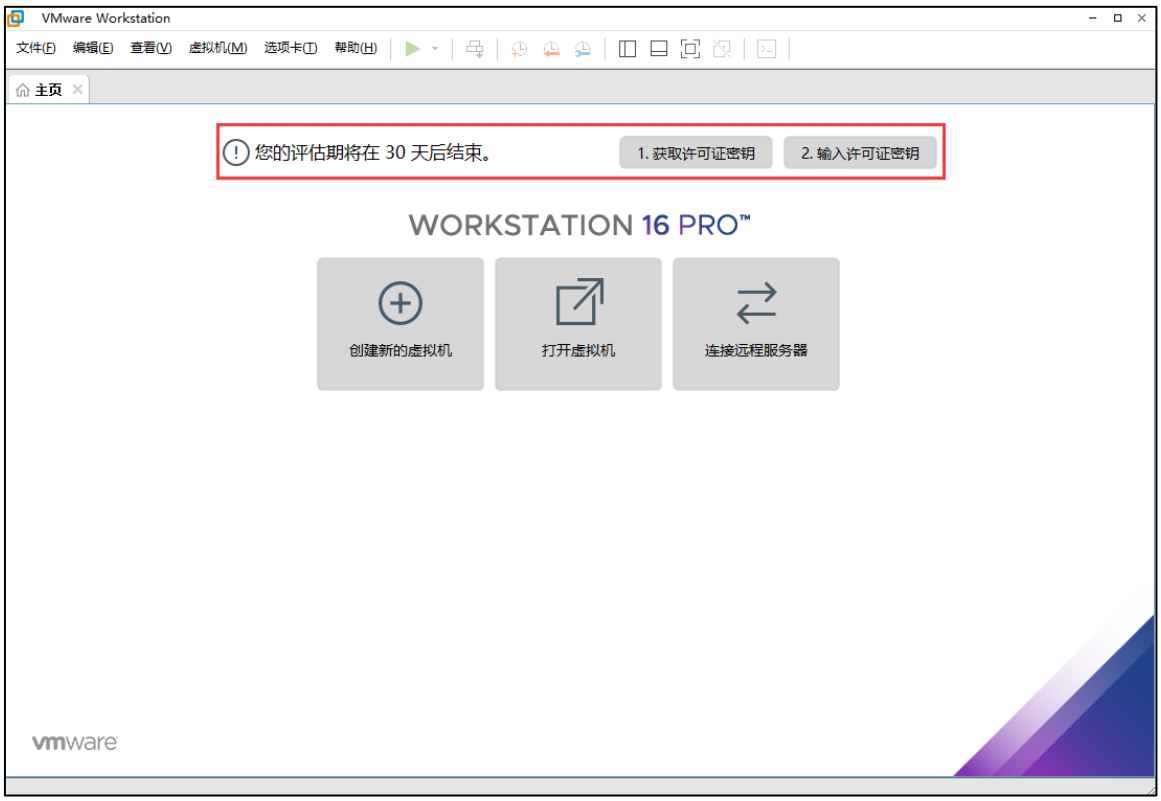


Figure 1.1.2.12 VMware virtual machine software main interface

1.2 Installing the Ubuntu system

This section introduces to users how to install the Ubuntu operating system in the VMware virtual machine. We recommend that users use Ubuntu 20.04 (**we strongly suggest that everyone use version 20.04. The documents, videos we provide, including those used by developers during the development process, are all in version 20.04**) or Ubuntu 18.04 (**not recommended!**). This document will be introduced using version 20.04 of Ubuntu. For users using Ubuntu 18.04, they can refer to this document for comparison.

1.2.1 Download the Ubuntu system image

The development board package has already provided users with Ubuntu 18.04 and Ubuntu 20.04 system image files. The paths are: Development Board CD-ROM A Disk - Basic Materials→4_softwares → ubuntu-18.04.6-desktop-amd64.iso and Development Board CD-ROM A Disk - Basic Materials → 04_softwares → ubuntu-20.04.2-desktop-amd64.iso. If users do not want to waste time downloading, they can directly use the Ubuntu system images provided in the package and proceed to the next 1.2.2 section.

Users can also download the Ubuntu system image through the link address: https://old-releases.ubuntu.com/releases/, (taking Ubuntu 20.04.2 version as an example):

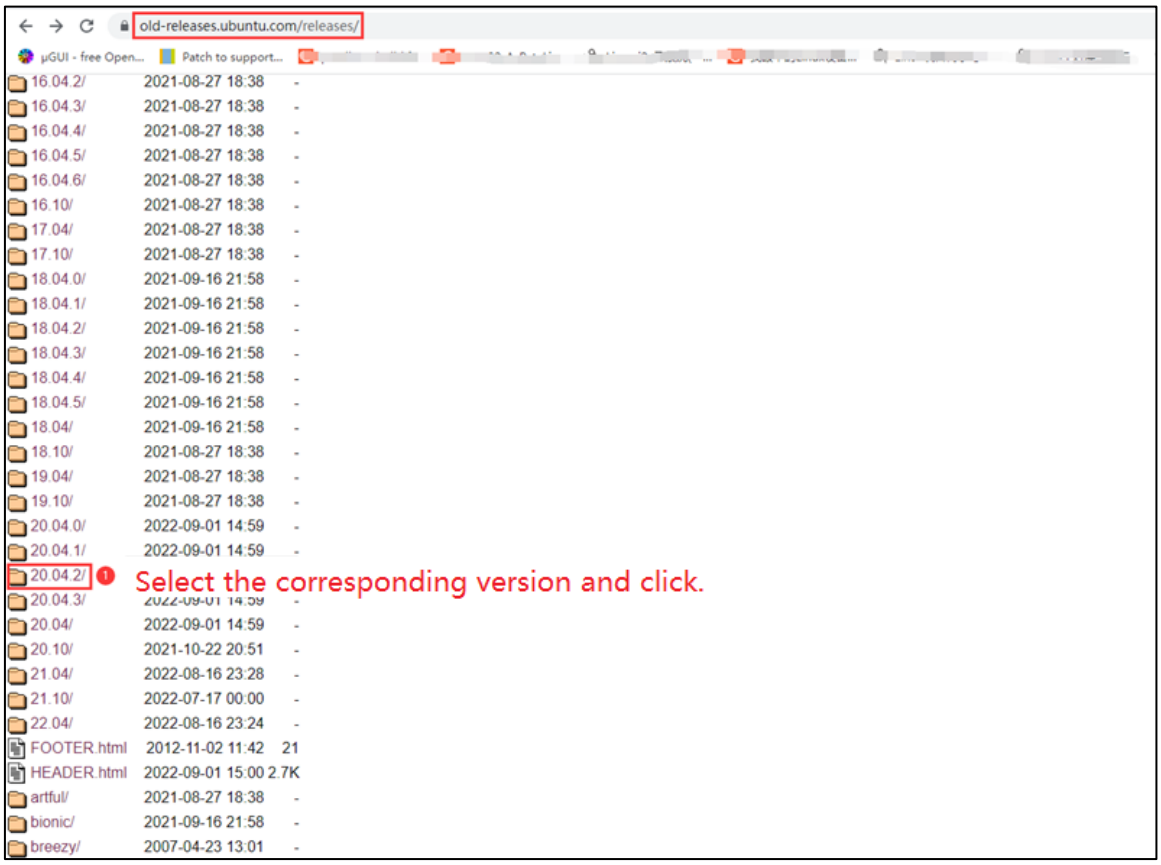


Figure 1.2.1.1 Download the Ubuntu system image 1

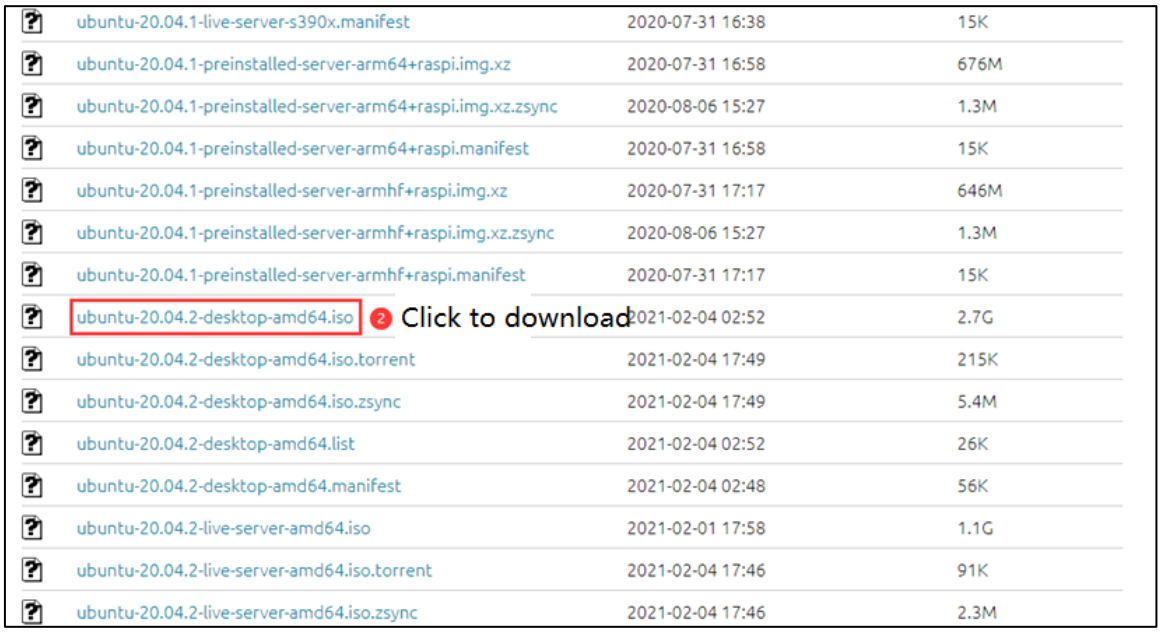


Figure 1.2.1.2 Download the Ubuntu system image 2

After the download is completed, you will obtain the Ubuntu 20.04.2 system image file: ubuntu-20.04.2-desktop-amd64.iso

Figure 1.2.1.3 Ubuntu 20.04.2 system image file

1.2.2 Create a virtual machine

Open the VMware virtual machine software, as shown in the figure:

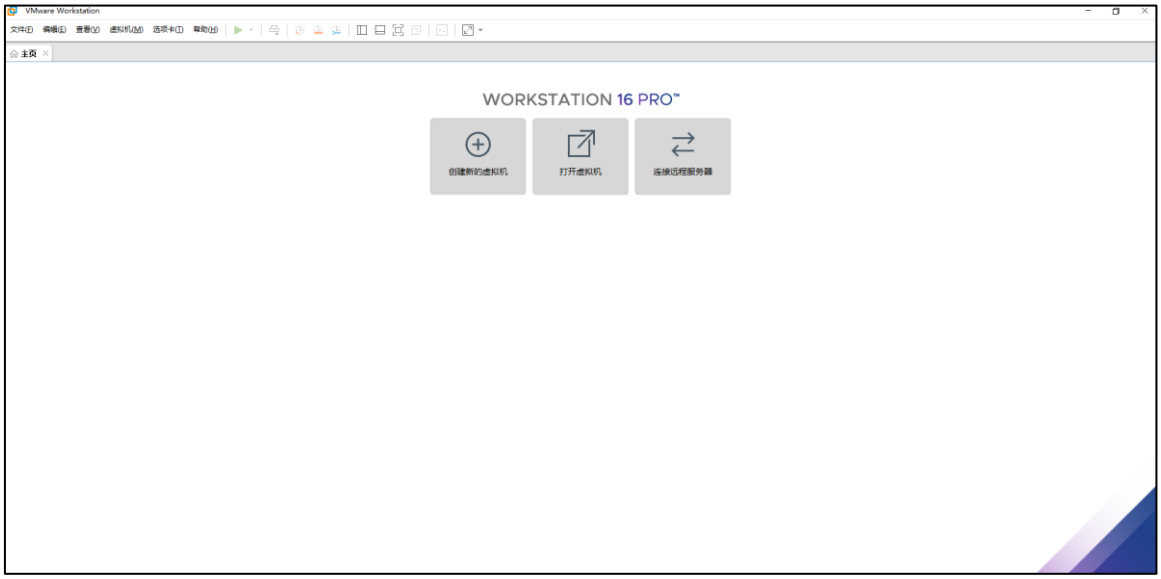


Figure 1.2.2.1 VMware virtual machine software

Then, follow the operation steps shown in Figures 1.2.2.2 to 1.2.2.17 to create a new virtual machine:

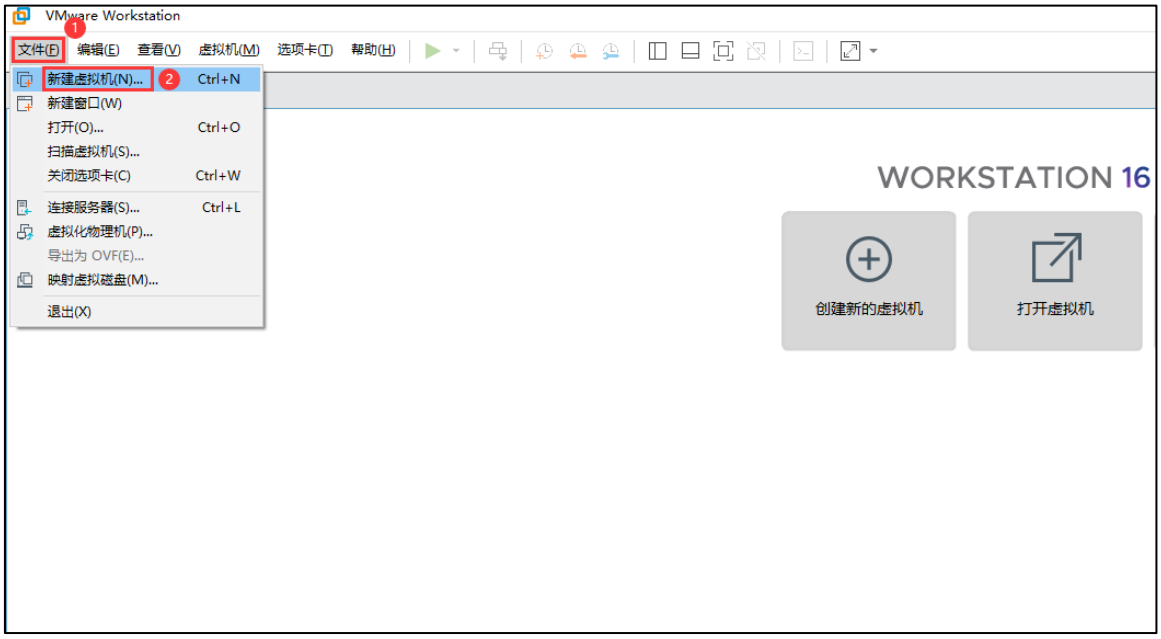


Figure 1.2.2.2 Create a virtual machine (1)

Figure 1.2.2.3 Create a virtual machine (2)

Figure 1.2.2.4 Create a virtual machine (3)

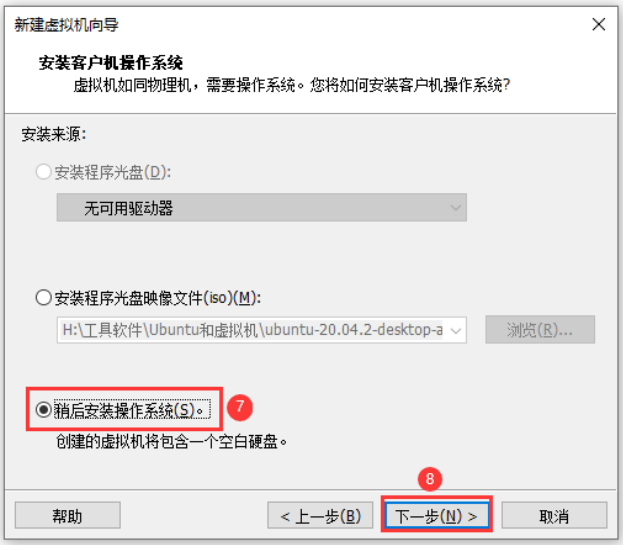


Figure 1.2.2.5 Create a virtual machine (4)

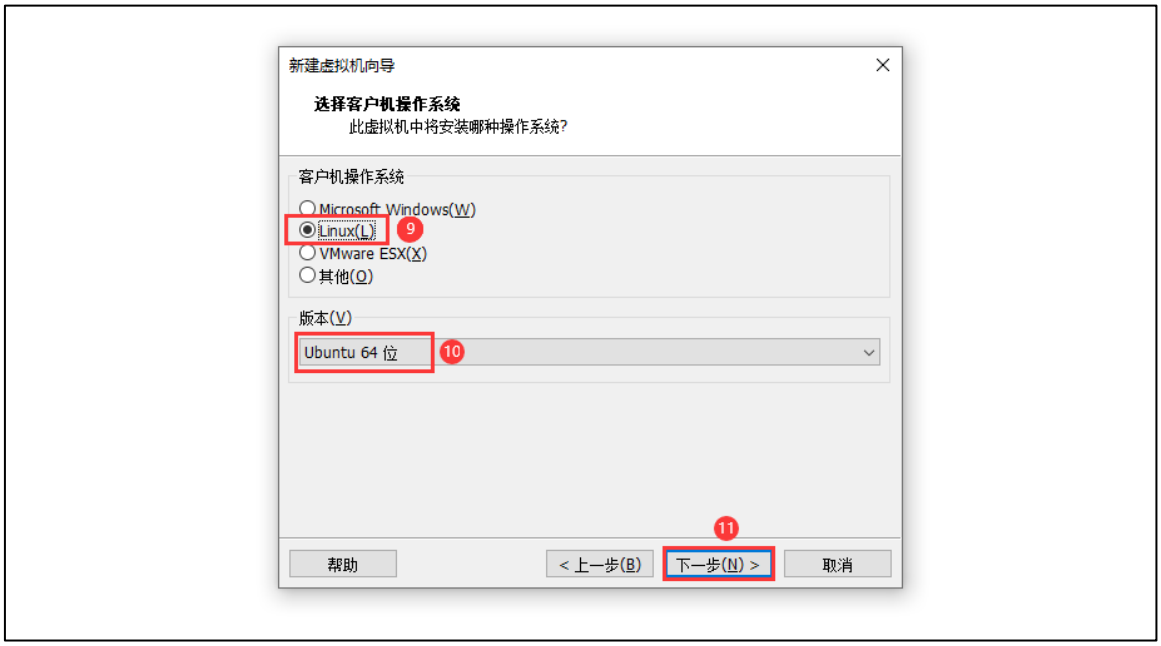


Figure 1.2.2.6 Create a virtual machine (5)

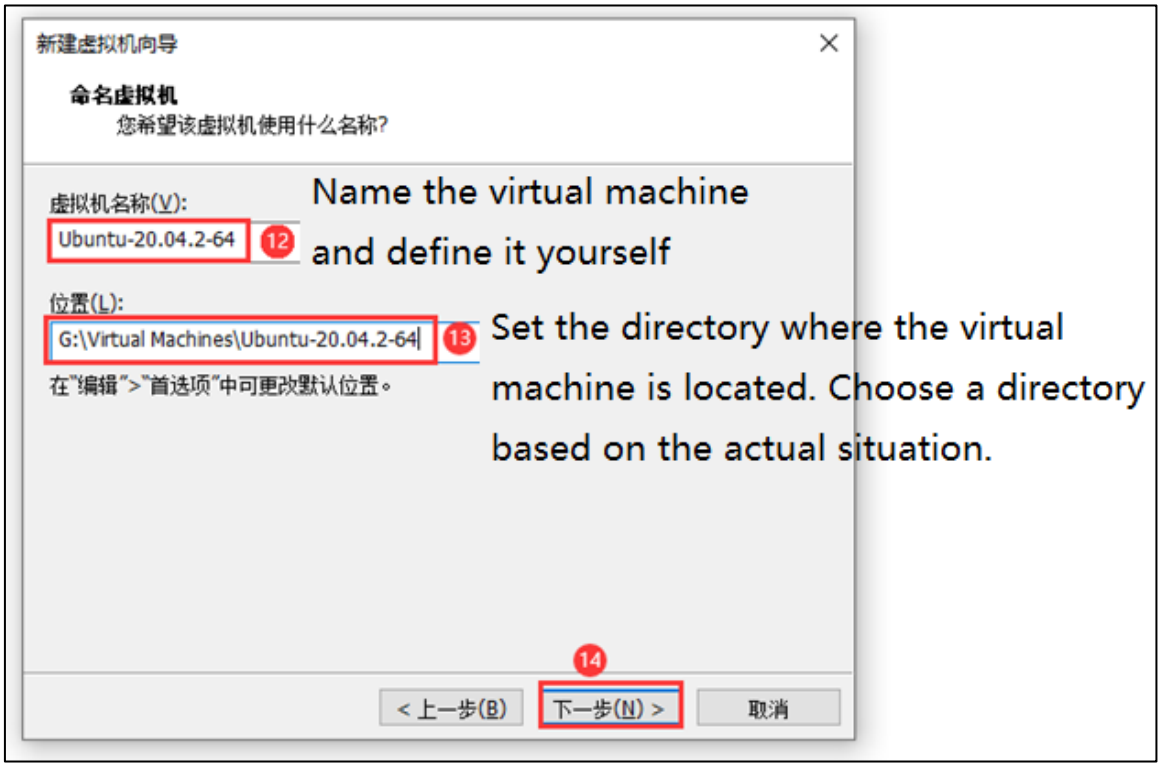


Figure 1.2.2.7 Create a virtual machine (6)

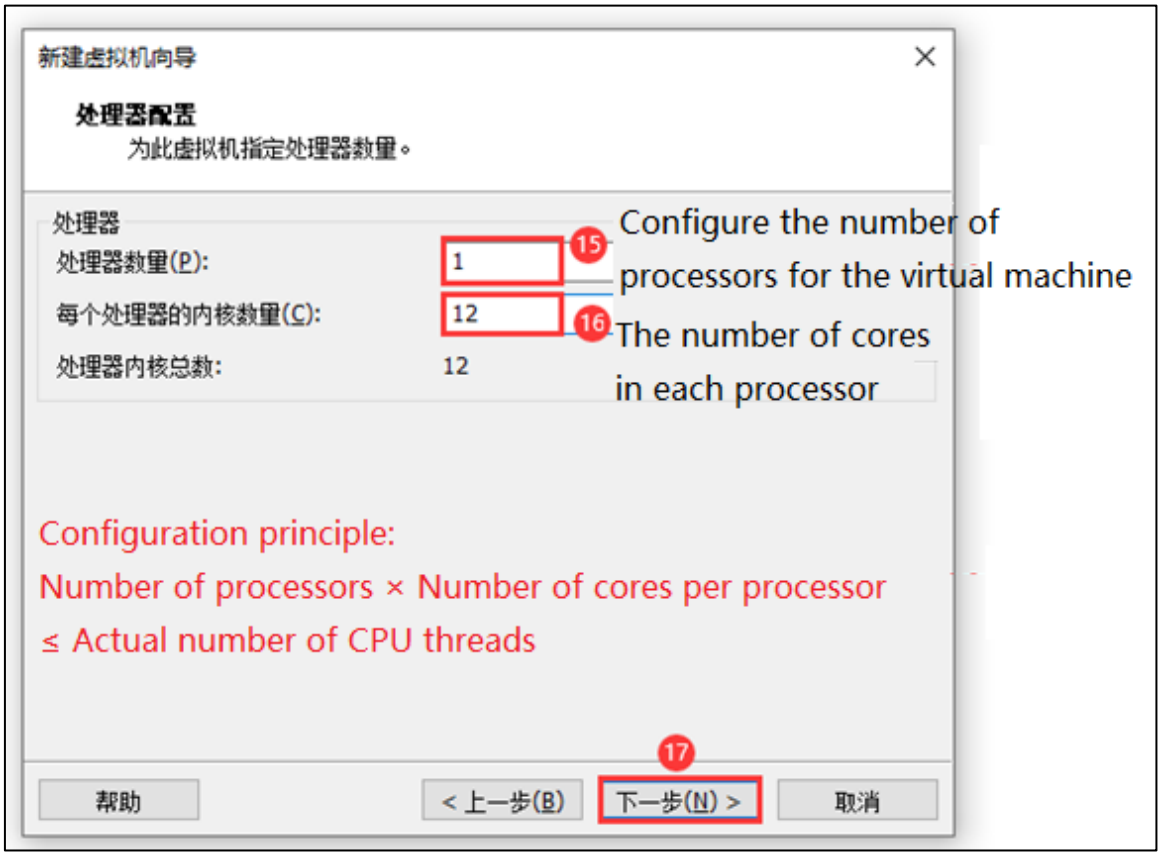


Figure 1.2.2.8 Create Virtual Machine (7)

Configure the number of processors and the number of cores per processor for the virtual machine:

Total number of processor cores = Number of processors * Number of cores per processor;

Users need to allocate the number of processors and the number of cores per processor for the virtual machine based on their computer configuration (CPU configuration). They only need to meet the following requirements:

The total number of processor cores of the virtual machine is less than or equal to (<=) the actual number of CPU threads;

The CPU configuration of the computer can be queried through the Windows Task Manager, as shown in Figure 1.2.2.9:

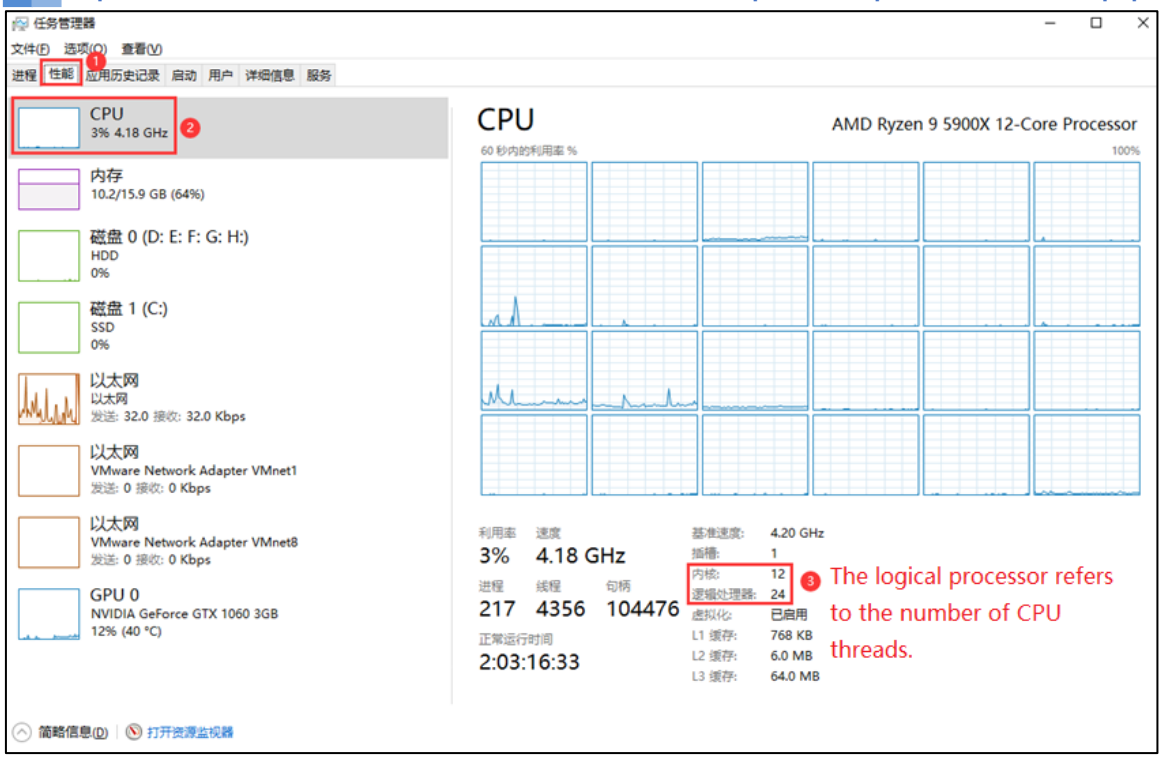


Figure 1.2.2.9 Query the CPU configuration

The number of logical processors in the figure represents the actual number of threads of the CPU. Therefore, the actual number of CPU threads in my computer is 24. According to the configuration requirements, the total number of processor cores for the virtual machine should be less than or equal to (<=) the actual number of CPU threads, which is 24. Then, there are various configuration options available, such as:

Processor quantity = 1, each processor's core quantity = 12;

Processor quantity = 1, each processor's core quantity = 16;

Processor quantity = 1, each processor's core quantity = 18;

Processor quantity = 4, each processor's core quantity = 4;

Processor quantity = 2, each processor's core quantity = 6;

Processor quantity = 6, each processor's core quantity = 3;

Processor quantity = 8, each processor's core quantity = 2; Wait a moment...

In summary, the specific configuration methods are diverse, and as long as the requirements are met, any method can be used!

Of course, when conditions permit (for example, without affecting the normal use of the physical machine's Windows system), it is advisable to make the total number of processor cores in the virtual machine slightly larger. This way, when compiling source code in the Ubuntu system of the virtual machine, we can enable more threads for compilation, and naturally the compilation speed will be faster!

Figure 1.2.2.10 Creating Virtual Machine (8)

The memory size of the virtual machine can be adjusted by dragging the blue slider on the left, or by directly entering a number in the input box on the right to specify the memory size, with the unit being MB. Similarly, the memory size of the virtual machine also needs to be set according to the user's computer configuration (memory configuration). For example, if my computer has a memory size of 32GB, I can allocate 16GB of memory for the virtual machine to use.

Of course, the memory capacity allocated to the virtual machine cannot be too small. If the memory is too small, it may cause errors when compiling source code under the Ubuntu system of the virtual machine (errors due to insufficient memory); for Android system development on the RK3568 platform, it is recommended to allocate a memory size of at least 16GB or even larger to the virtual machine.

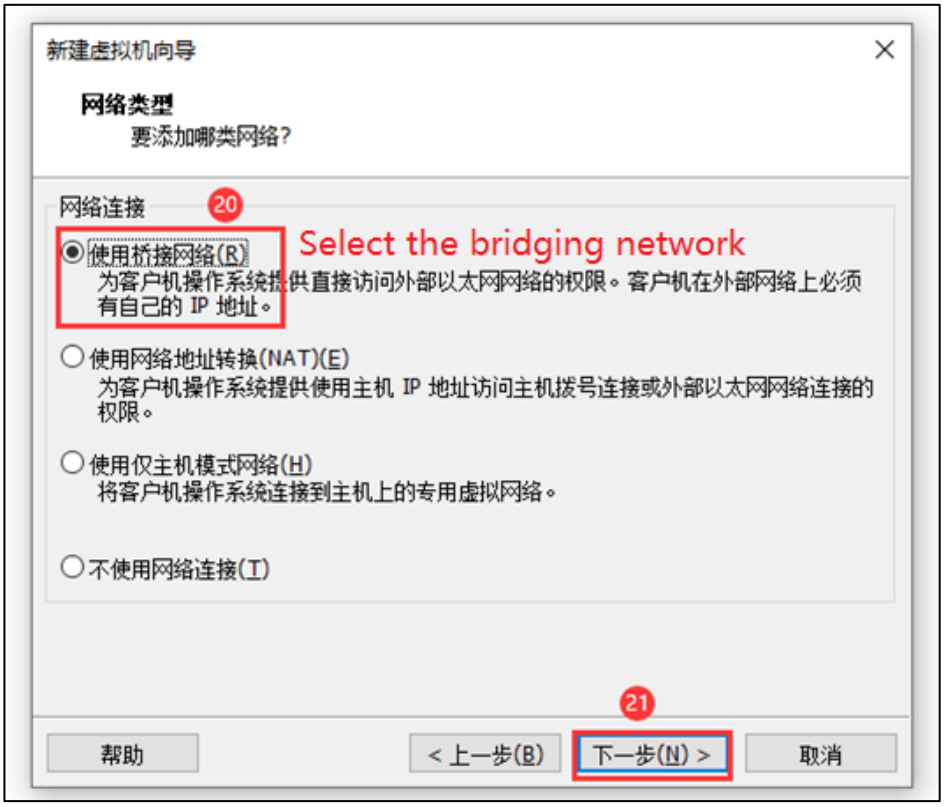


Figure 1.2.2.11 Create virtual machine (9)

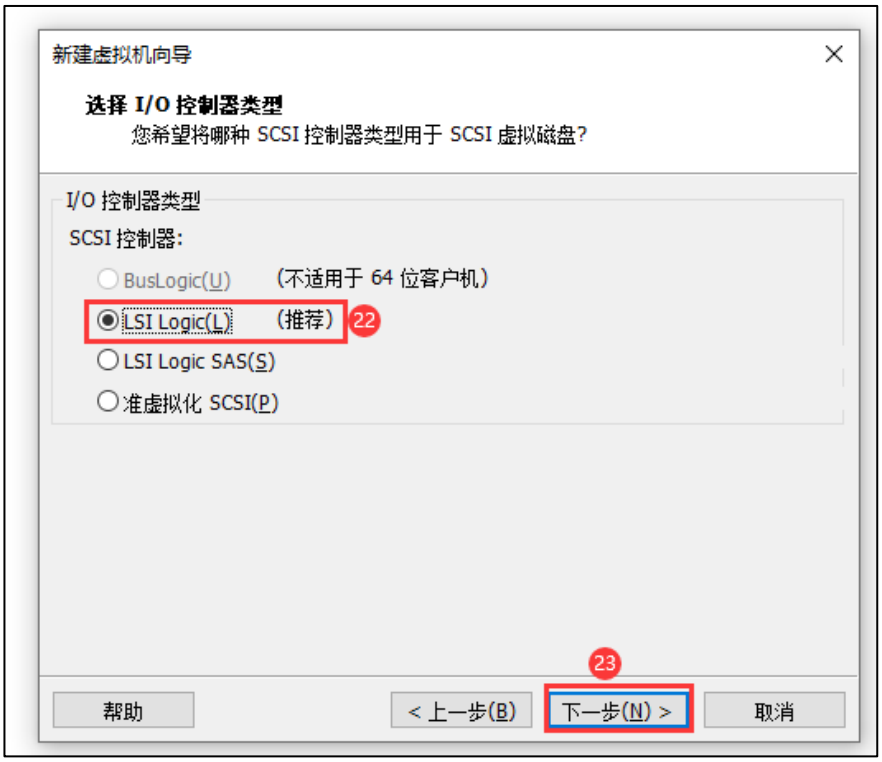


Figure 1.2.2.12 Create virtual machine (10)

Figure 1.2.2.13 Create virtual machine (11)

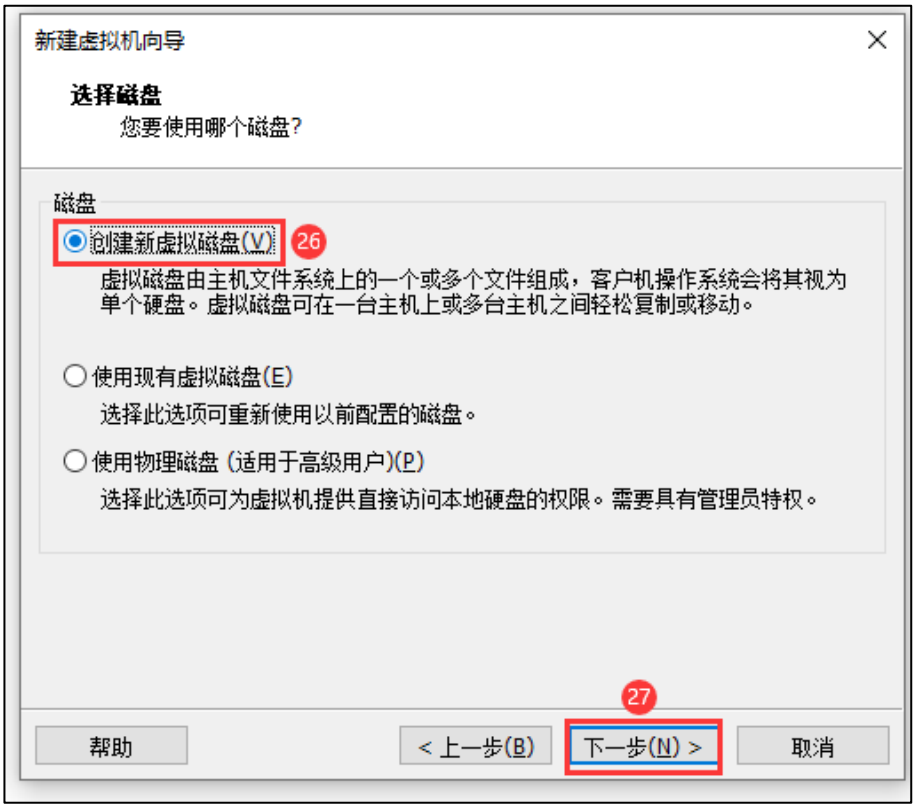


Figure 1.2.2.14 Create virtual machine (12)

Figure 1.2.2.15 Create virtual machine (13)

Similarly, the virtual machine disk capacity also needs to be set according to the user's computer configuration (disk configuration). It is recommended to set it to at least 300 to 500 GB.

Figure 1.2.2.16 Create virtual machine (14)

Figure 1.2.2.17 Create virtual machine (15)

At this point, the virtual machine has been successfully created, as shown in Figure 1.2.2.18:

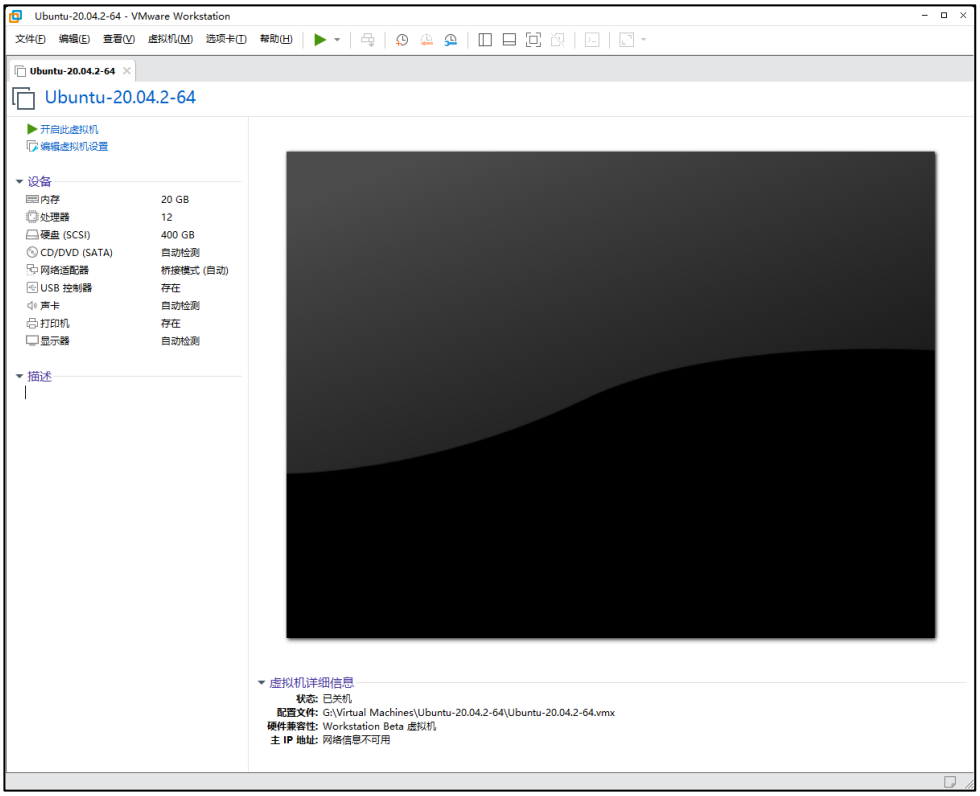


Figure 1.2.2.18 The virtual machine has been created successfully.

1.2.3 Install Ubuntu system

Next, we will install the Ubuntu 20.04 operating system on the virtual machine. On the created virtual machine, click "**Virtual Machine (M) →Settings**" to open the virtual machine settings interface:

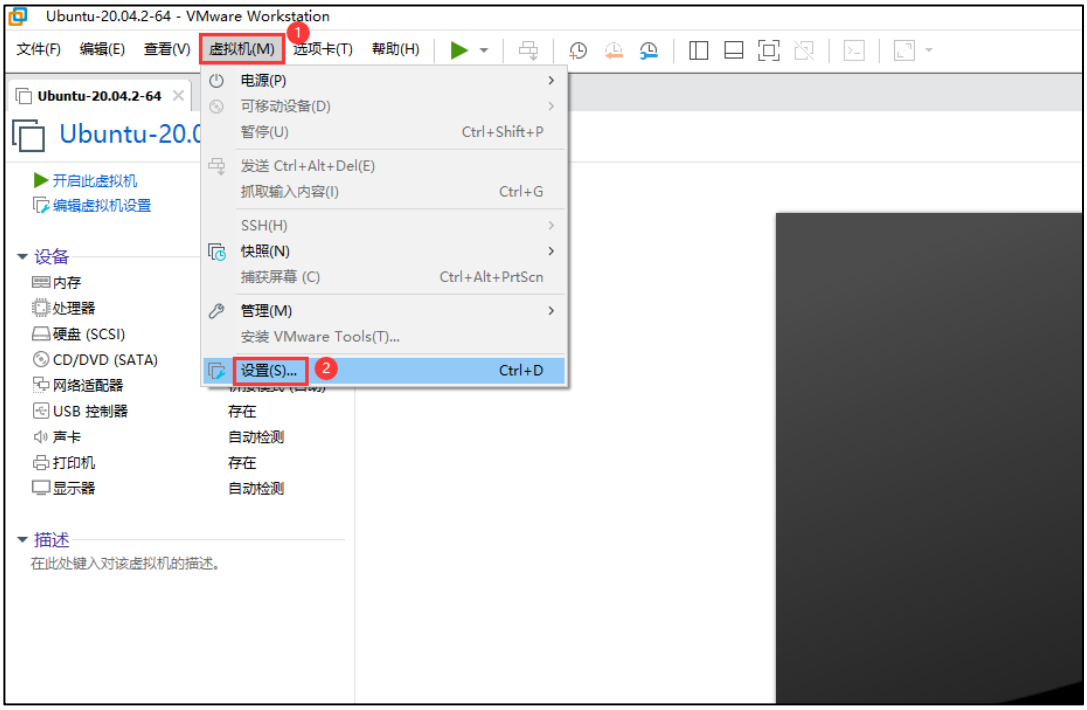


Figure 1.2.3.1 Open the virtual machine settings interface

First, specify the path where the Ubuntu 20.04 system image file "ubuntu-20.04.2-desktop-amd64.iso" is located:

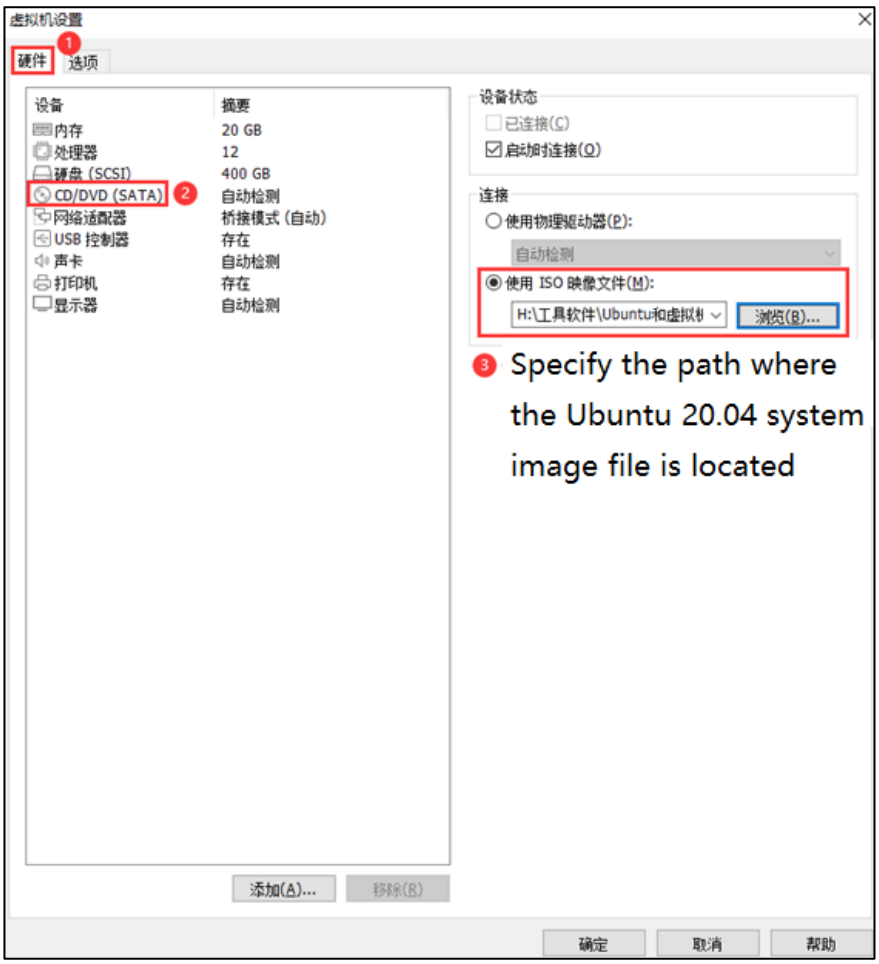


Figure 1.2.3.2 Specify the Ubuntu system image

Click the "Browse (B)" button, and select the Ubuntu 20.04 system image file: ubuntu-20.04.2-desktop-amd64.iso.

Next, set up the USB controller of the virtual machine:

Figure 1.2.3.3 Set USB controller compatibility

The USB compatibility of the USB controller is set to USB 2.0 by default. When we use USB 3.0 devices, the Ubuntu system may not be able to recognize them. Therefore, we need to adjust the USB compatibility to USB 3.1. Finally, click the "OK" button to exit the window.

Next, start installing the Ubuntu operating system. Follow the steps shown in Figures 1.2.3.4 to 1.2.3.13 to install the Ubuntu system:

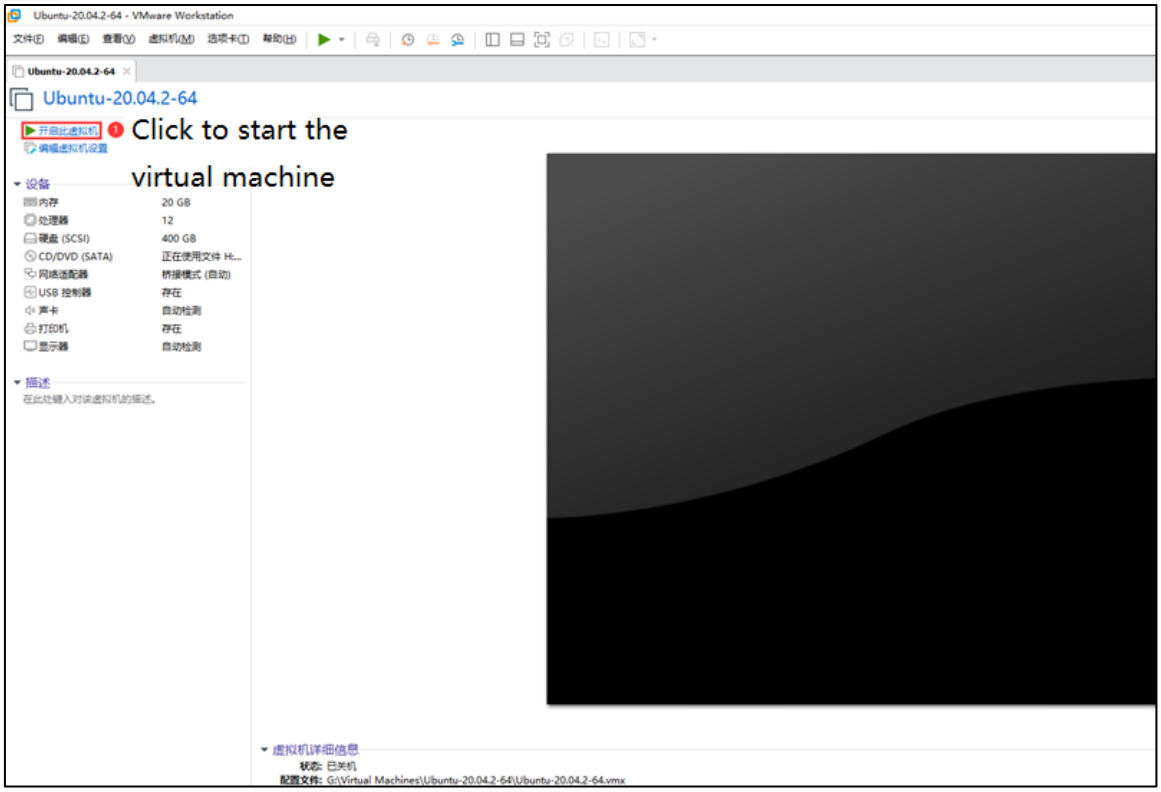


Figure 1.2.3.4 Install the Ubuntu operating system (1)

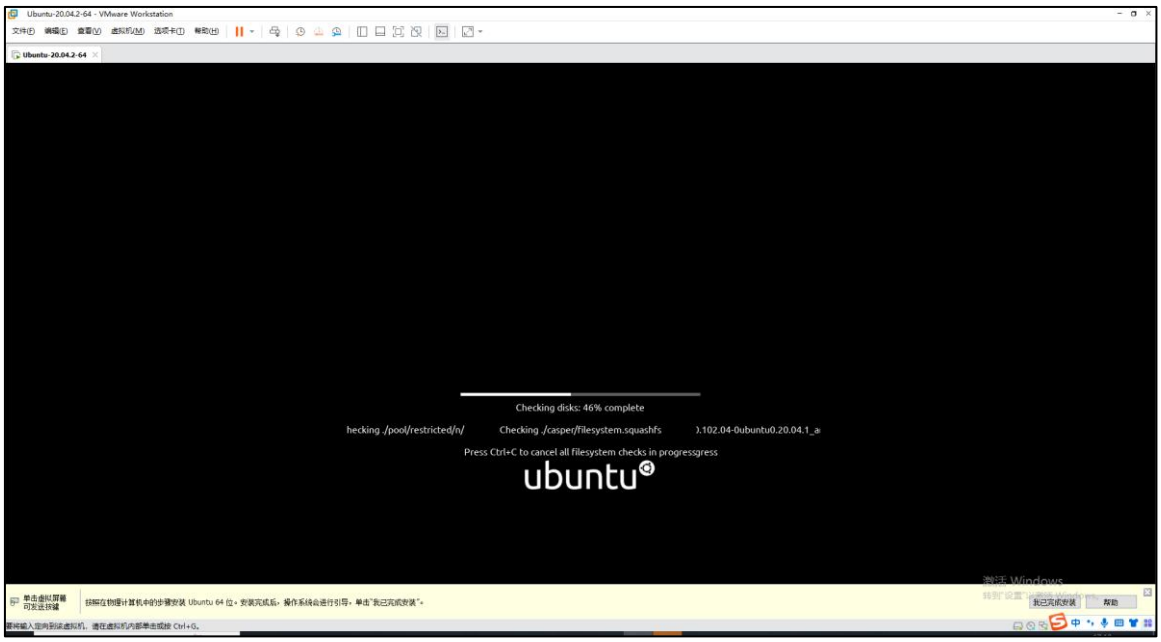


Figure 1.2.3.5 Install the Ubuntu operating system (2)

Figure 1.2.3.6 Install the Ubuntu operating system (3)

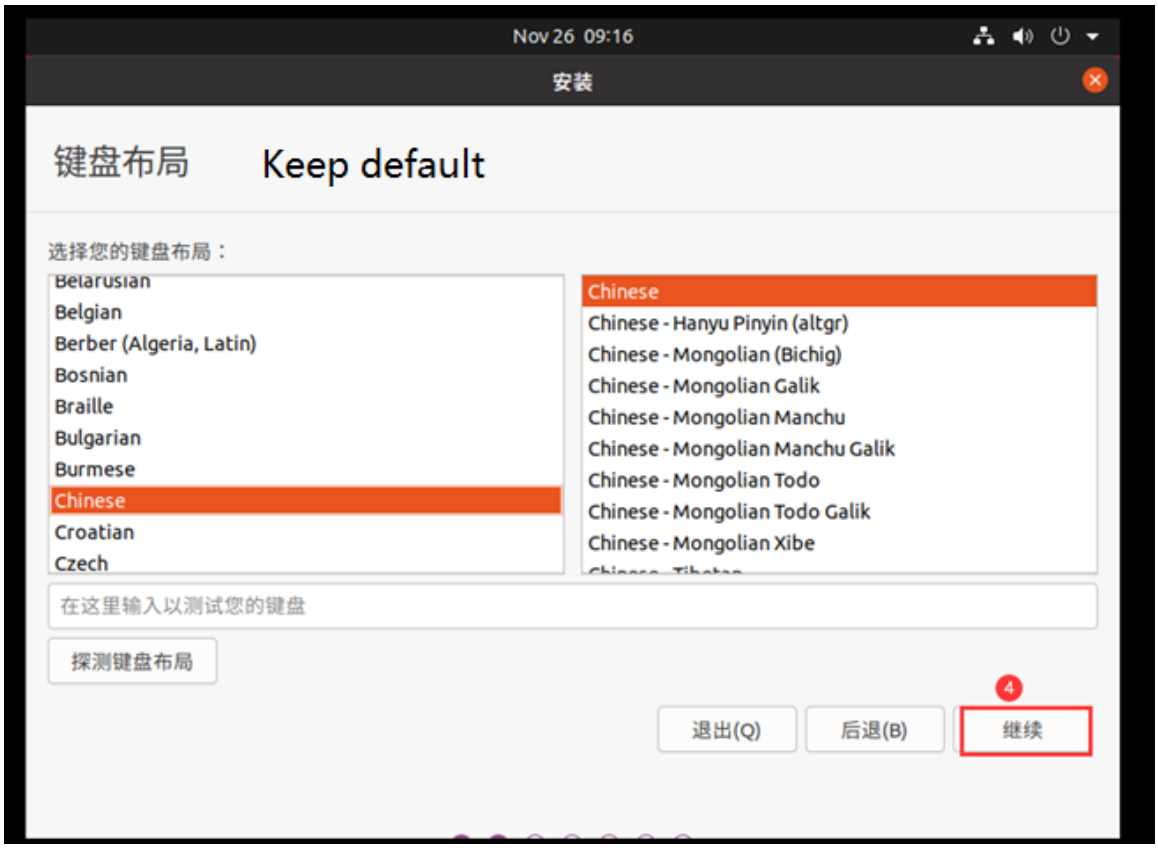


Figure 1.2.3.7 Installing Ubuntu Operating System (4)

If during the installation process, due to display issues, the user cannot see the following three buttons (Exit, Back, and Continue), and thus is unable to click on the buttons; at this time, you can hold down the Win key (the one between the Ctrl key and the Alt key), and then use the left mouse button to drag the window.

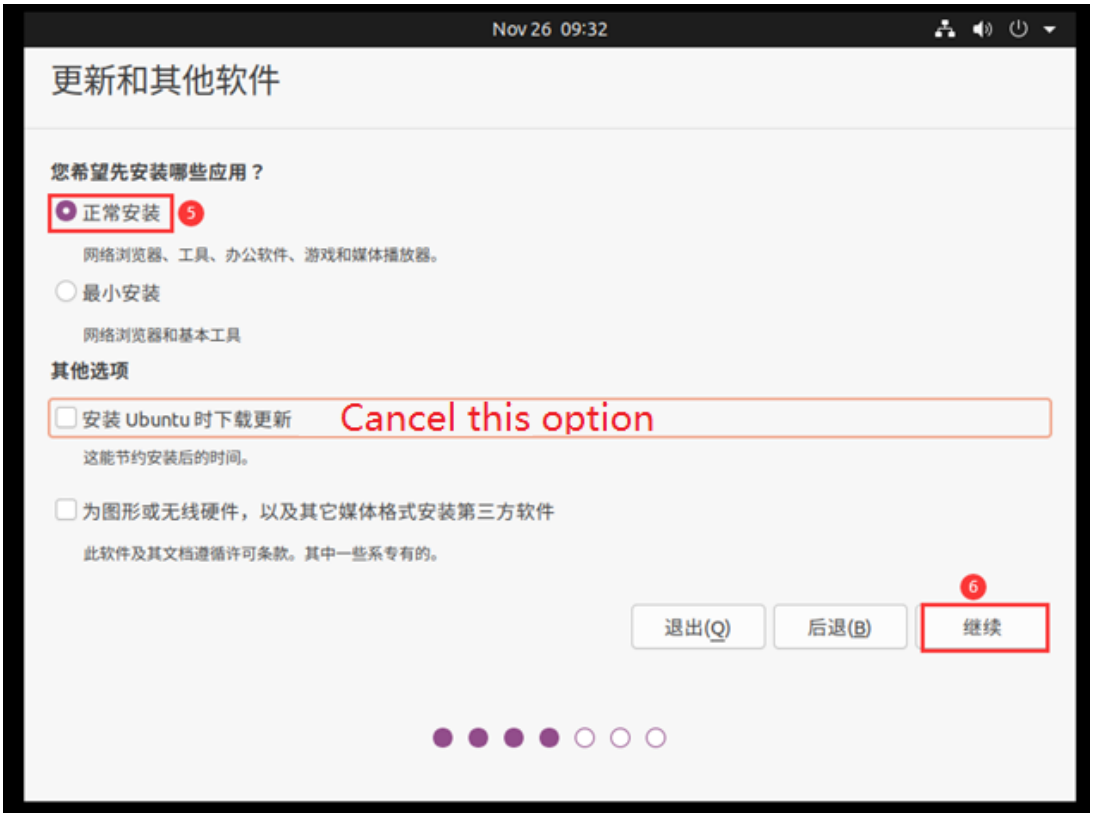


Figure 1.2.3.8 Install the Ubuntu operating system (5)

Figure 1.2.3.9 Install the Ubuntu operating system (6)

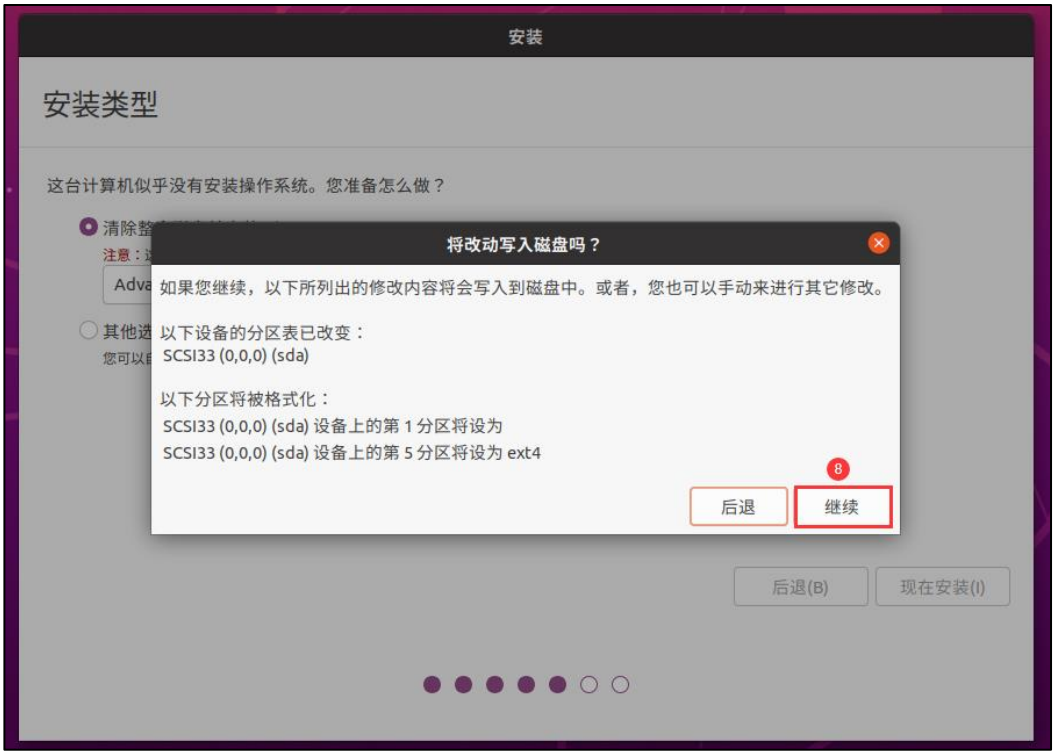


Figure 1.2.3.10 Install the Ubuntu operating system (7)

Figure 1.2.3.11 Install the Ubuntu operating system (8)

Enter the name of the city where the user is located. This information is mainly used to determine the time zone where the user is situated.

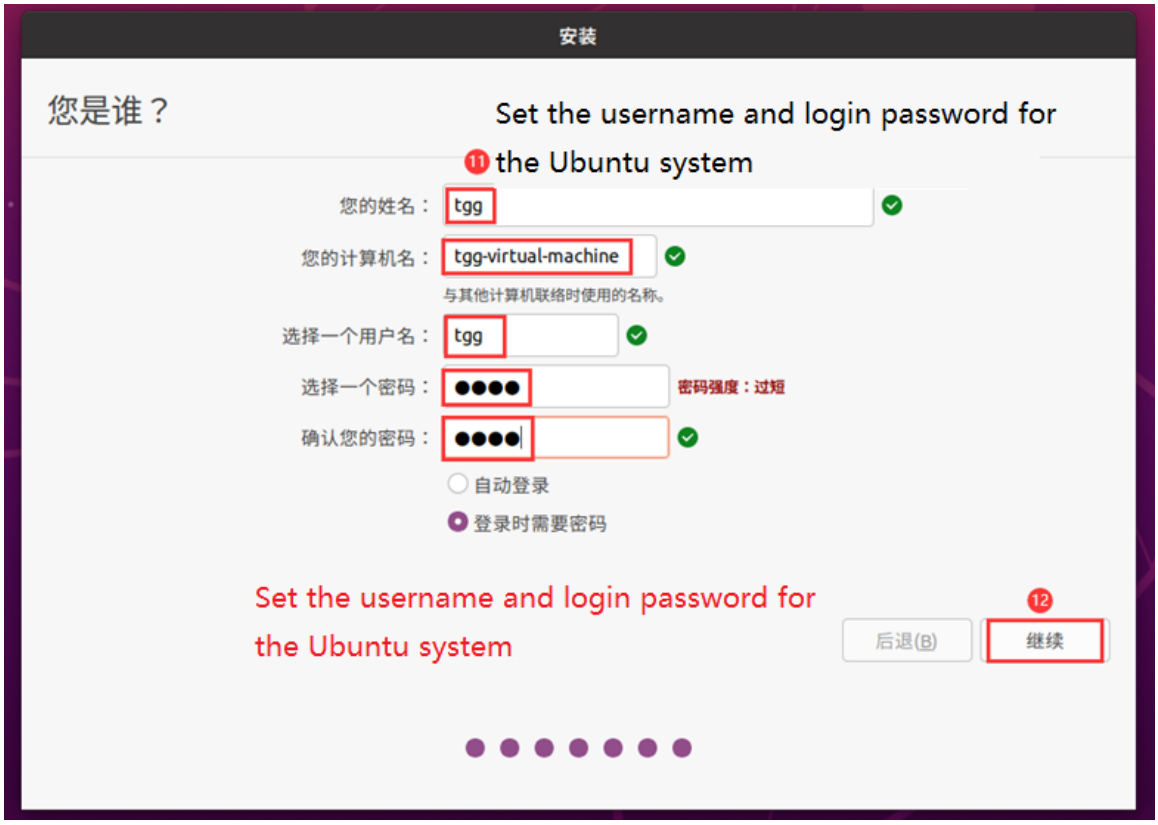


Figure 1.2.3.12 Install the Ubuntu operating system (9)

After clicking the "Continue" button, the actual installation process begins:

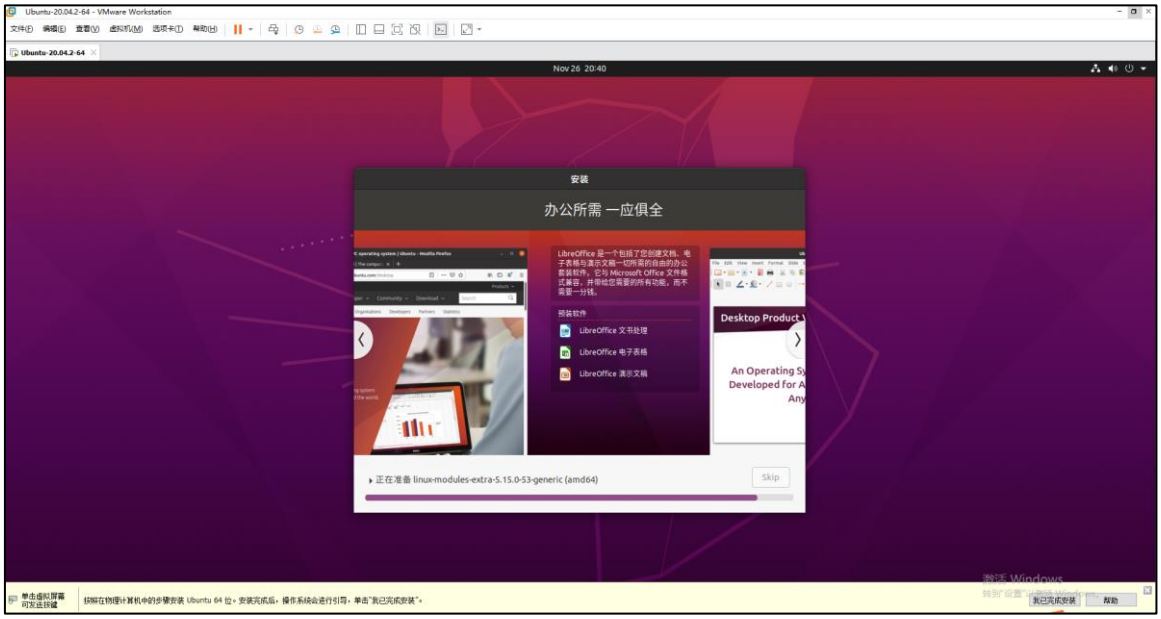


Figure 1.2.3.13 Install the Ubuntu operating system (10)

Then wait for the system installation to complete. After the installation is finished, the system will prompt the user to restart, as shown in Figure 1.2.3.14:

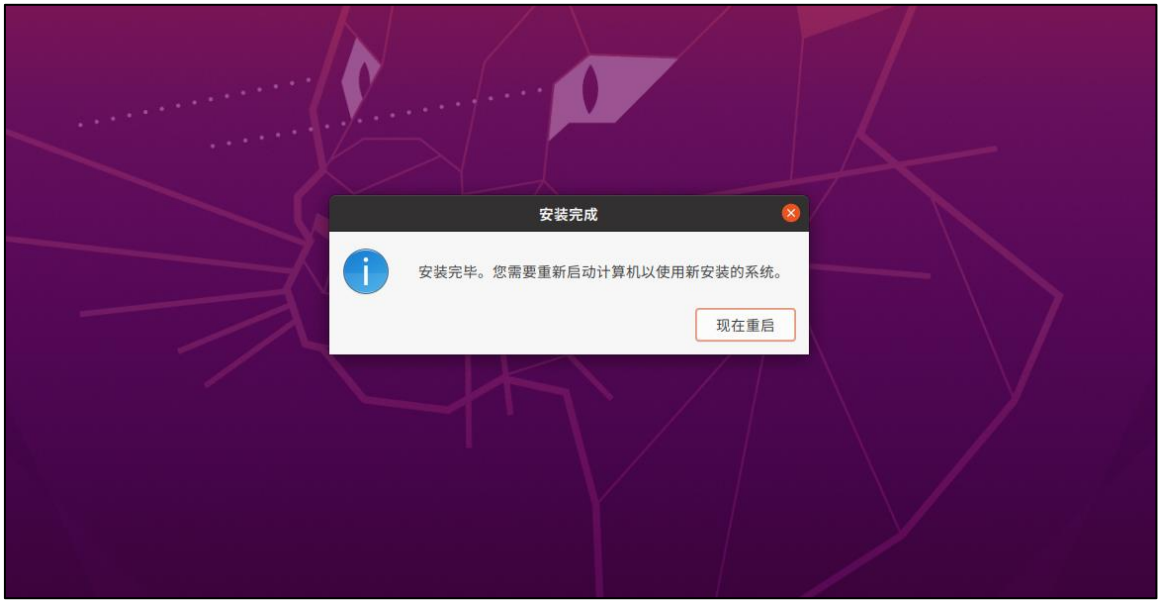


Figure 1.2.3.14 Ubuntu system installation is complete.

At this point, don't rush to click the "Restart Now" button. We need to first shut down the virtual machine and remove the Ubuntu 20.04 system image file. Follow the steps shown in Figures 1.2.3.15 to 1.2.3.16 to shut down the virtual machine:

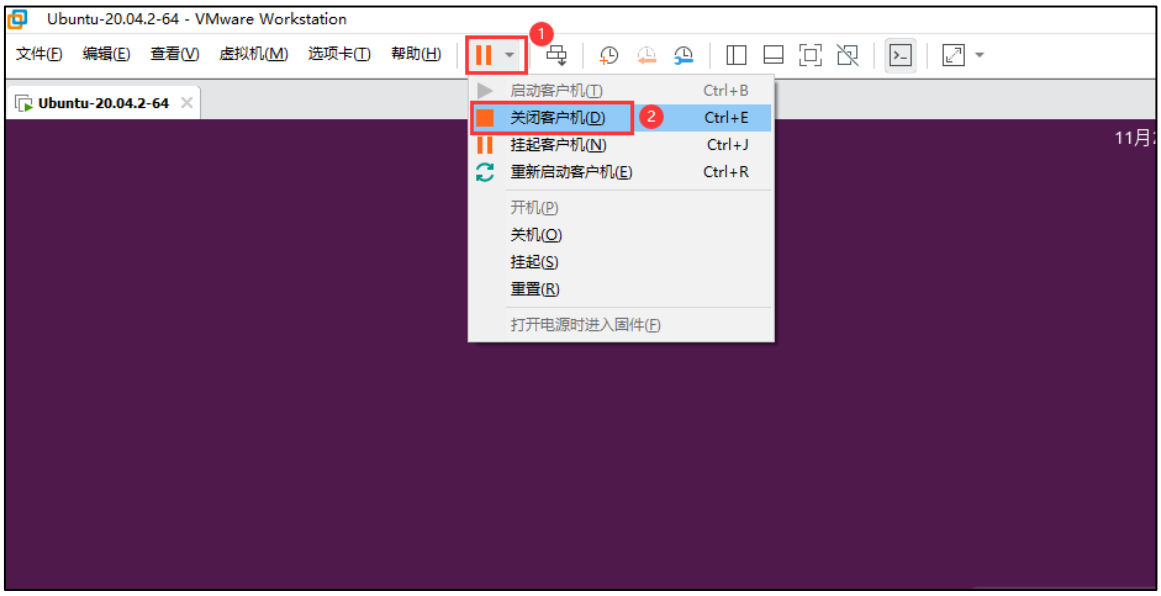


Figure 1.2.3.15 Shutdown the virtual machine (1)

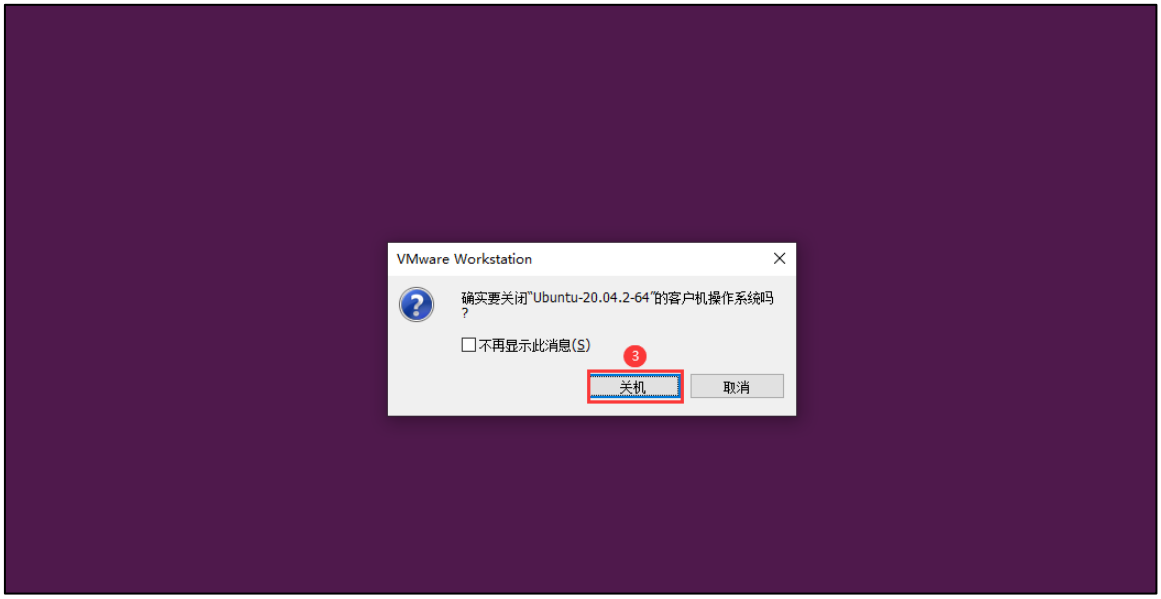


Figure 1.2.3.16 Shutdown the virtual machine (2)

After closing the virtual machine, click on the "**Virtual Machine (M) →Settings (S)**" option in the menu bar to enter the virtual machine settings interface. Remove the Ubuntu system image file as shown in Figure 1.2.3.17:

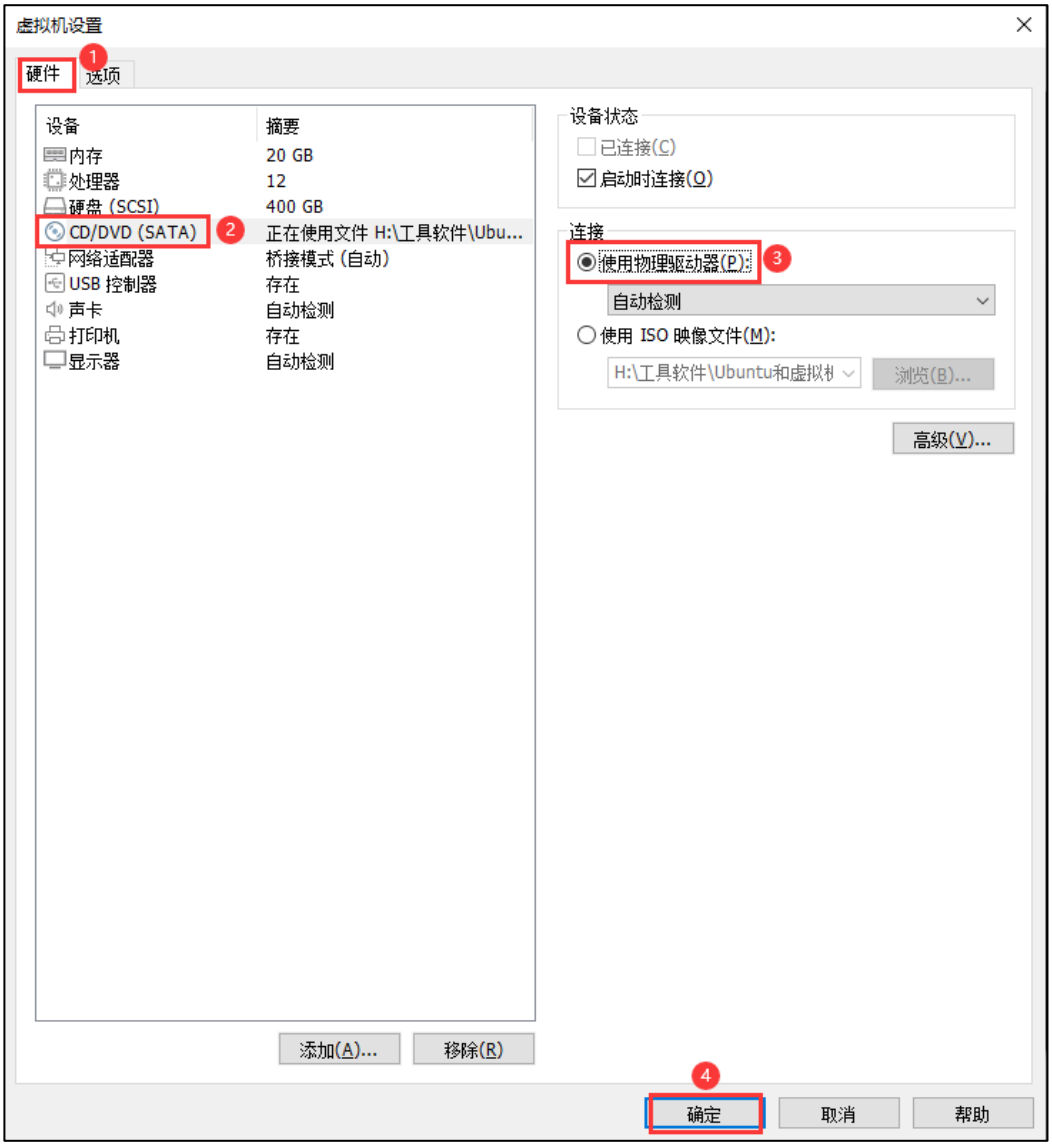


Figure 1.2.3.17 Remove the Ubuntu system image file.

After clicking the "OK" button, exit the settings window. Then you can start the virtual machine. Follow the steps shown in Figure 1.2.3.18 to start the virtual machine (that is, to boot the Ubuntu system):

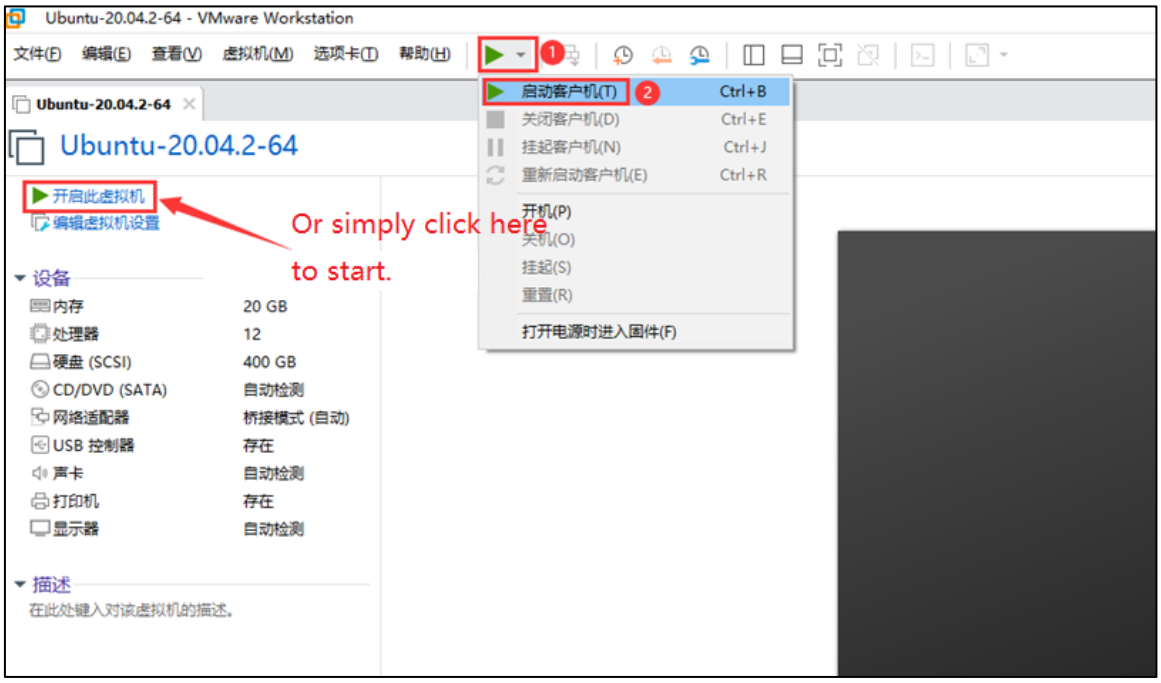


Figure 1.2.3.18 Start the virtual machine

After the system boots up, as shown in Figure 1.2.3.19:

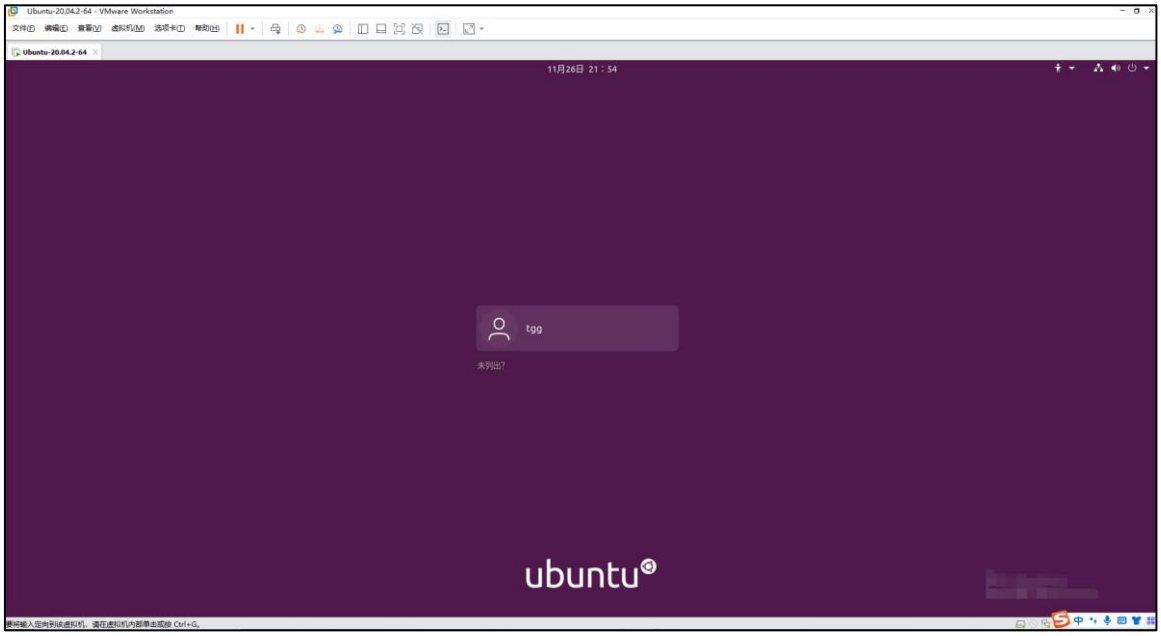


Figure 1.2.3.19 User login interface

Click on the "Next" button, enter the username, input the password, and press the Enter key to log in to the system and enter the main interface of the Ubuntu system:

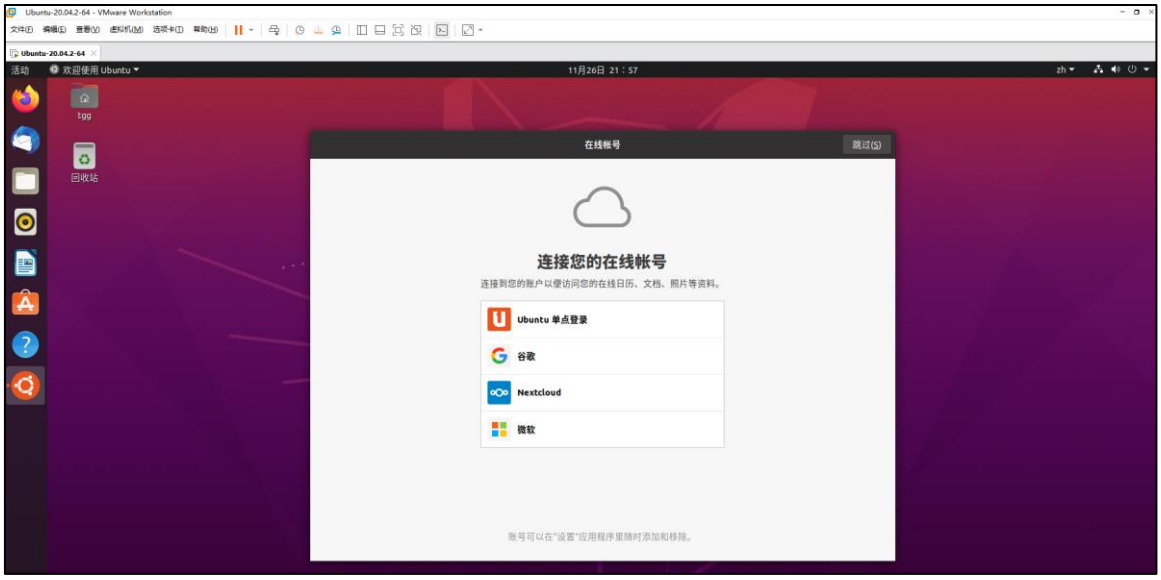


Figure 1.2.3.20 Ubuntu system main interface

When you first enter the Ubuntu 20.04 system desktop, an "Online Account" window (which is a simple setup guide) will pop up, as shown in Figure 1.2.3.20; you can follow the instructions step by step or simply exit, as shown in Figure 1.2.3.21:

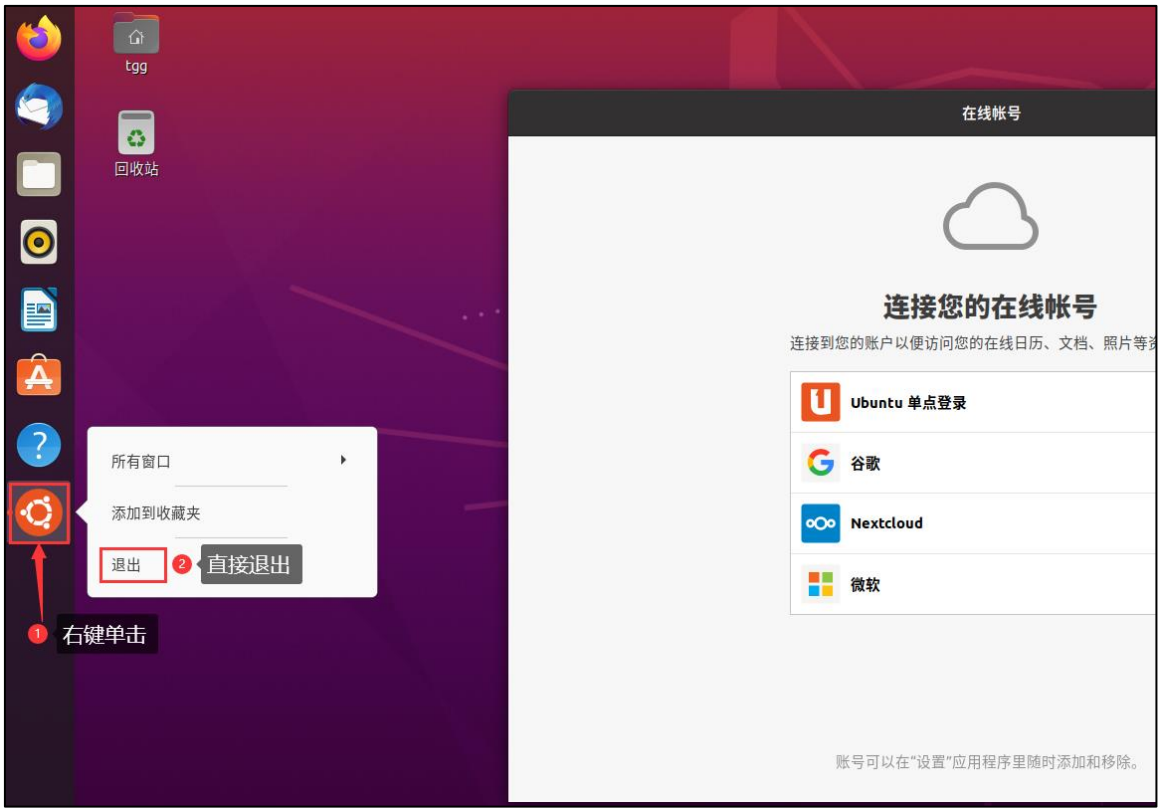


Figure 1.2.3.21 Exit the usage guidance window

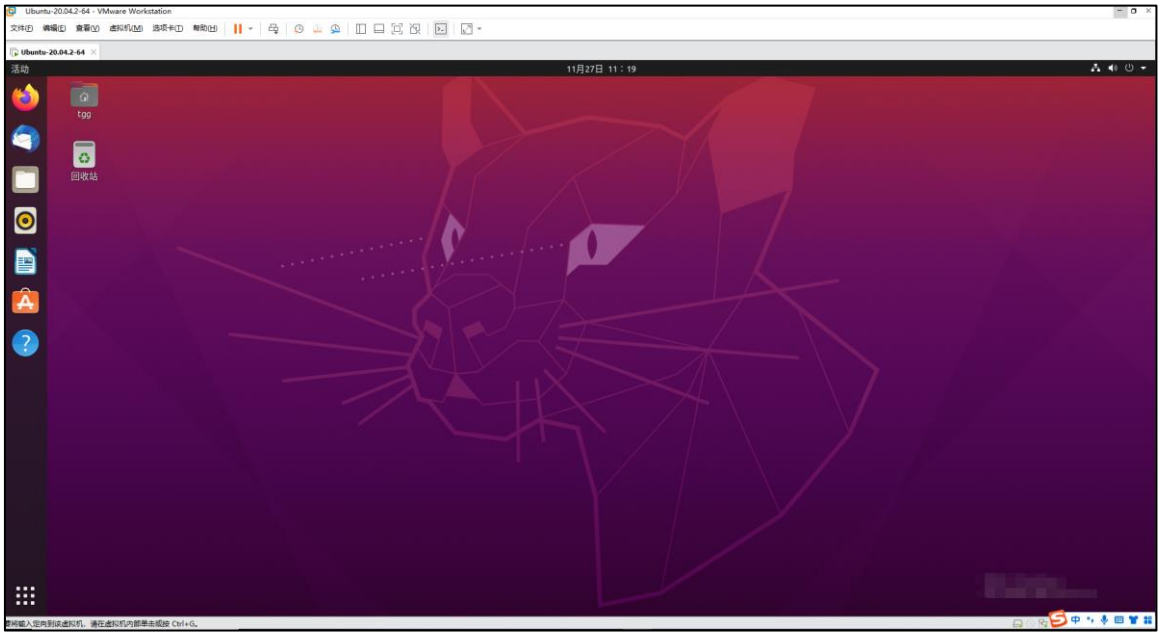


Figure 1.2.3.22 Ubuntu desktop

At this point, the Ubuntu 20.04 operating system has been installed successfully!

Chapter 2. Development environment setup

Before developing the RK3568 Android system, we need to set up the development environment first; including the environment setup for both Windows and Ubuntu operating systems, as embedded development is usually based on a dual-system development environment of Windows + Ubuntu.

This chapter will be divided into the following sections:

2.1 Ubuntu System Settings
2.2 File Transfer Between Ubuntu and Windows
2.3 Setting up the tftp server in the Ubuntu system
2.4 Setting up the nfs server in the Ubuntu system
2.5 Setting up the ssh server in the Ubuntu system
2.6 Installation of CH340 Serial Port Driver
2.7 Installation of MobaXterm Software
2.8 Installation of Rockchip USB Driver
2.9 Installation of ADB Tools

2.1 Ubuntu System Settings

This section provides a simple setup for the Ubuntu system.

2.1.1 Set the root user password

The newly installed Ubuntu system does not have a root user password set. We need to open the terminal (Ctrl + T) and execute the following command to set the root user password:

```
sudo passwd root
```

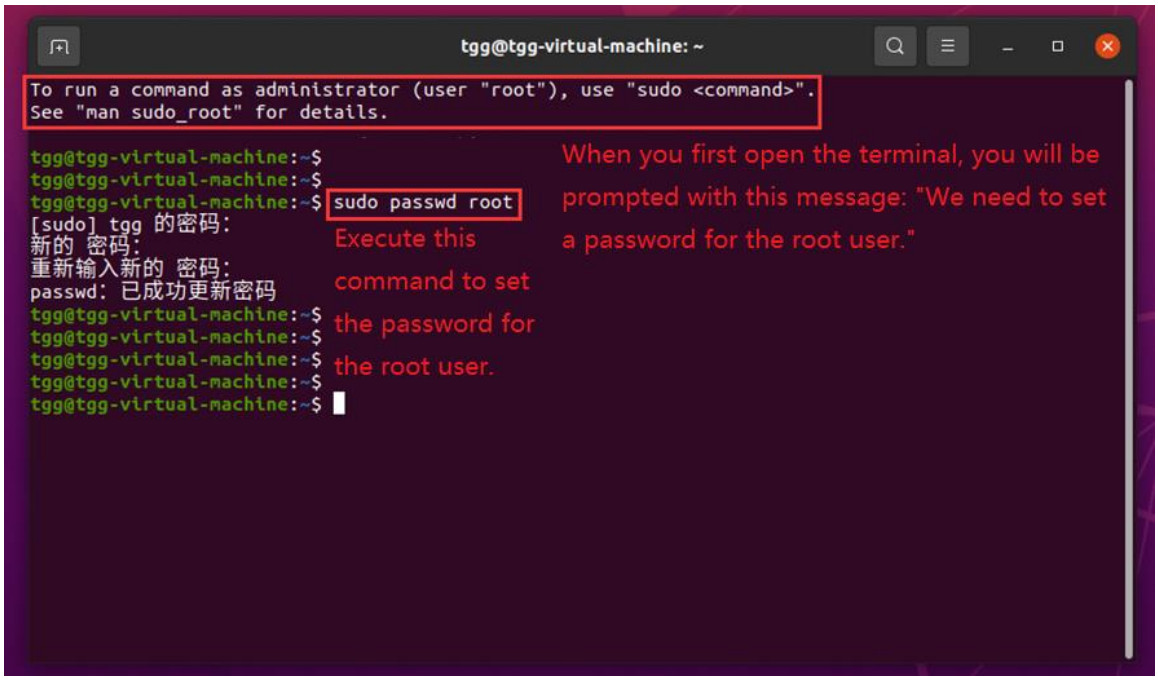


Figure 2.1.1.1 Set the password for the root user

2.1.2 Change the software download source

The default software download source of the Ubuntu system is due to server issues, and the download speed in China is often relatively slow. In this case, we can change the software download source of the Ubuntu system to a domestic one, such as Alibaba source, University of Science and Technology of China source, Tsinghua source, etc. The download speed will be much faster compared to the official software source of Ubuntu!

Follow the steps shown in Figure 2.1.2.1 to 2.1.2.7 to make the change:

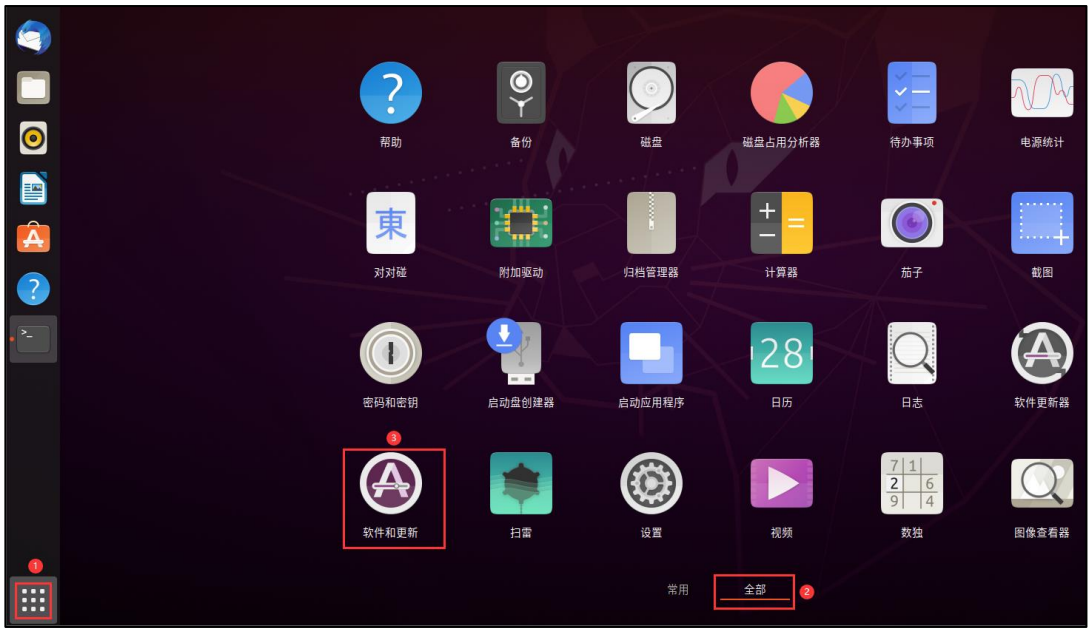


Figure 2.1.2.1 Change the software download source (1)

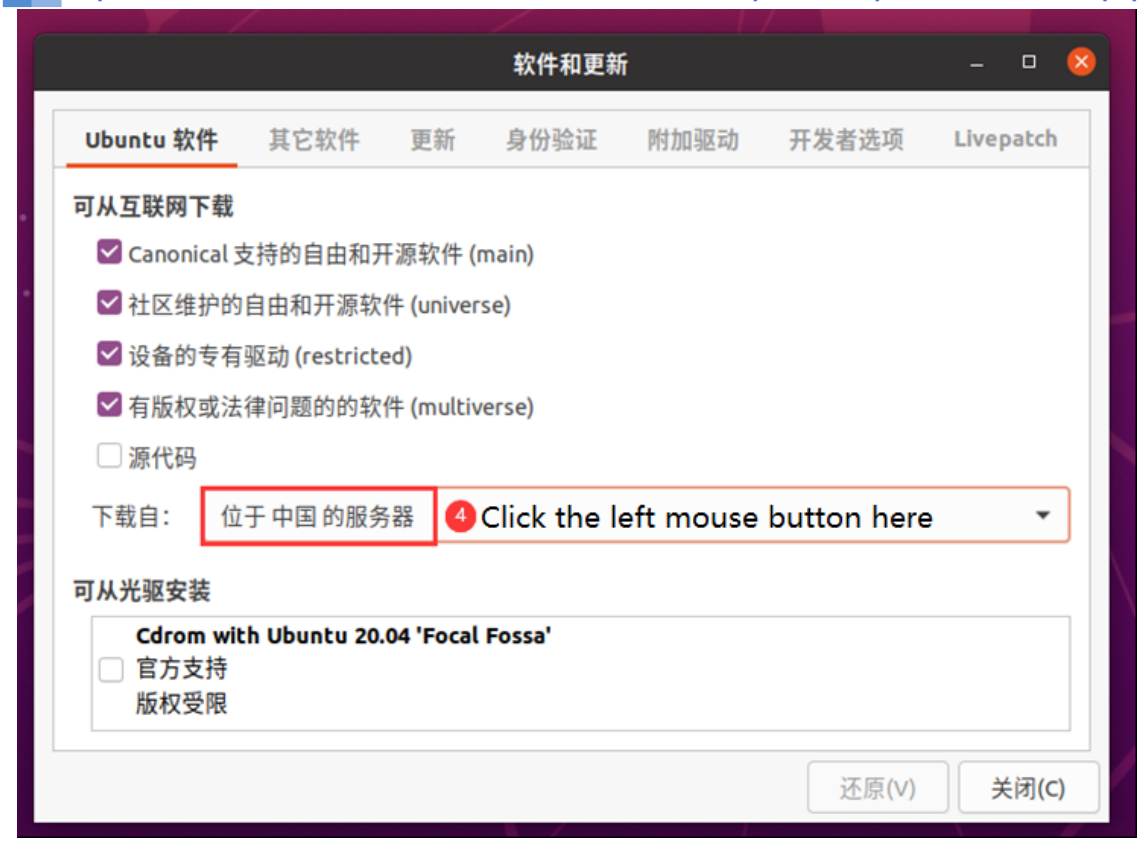


Figure 2.1.2.2 Change the software download source (2)

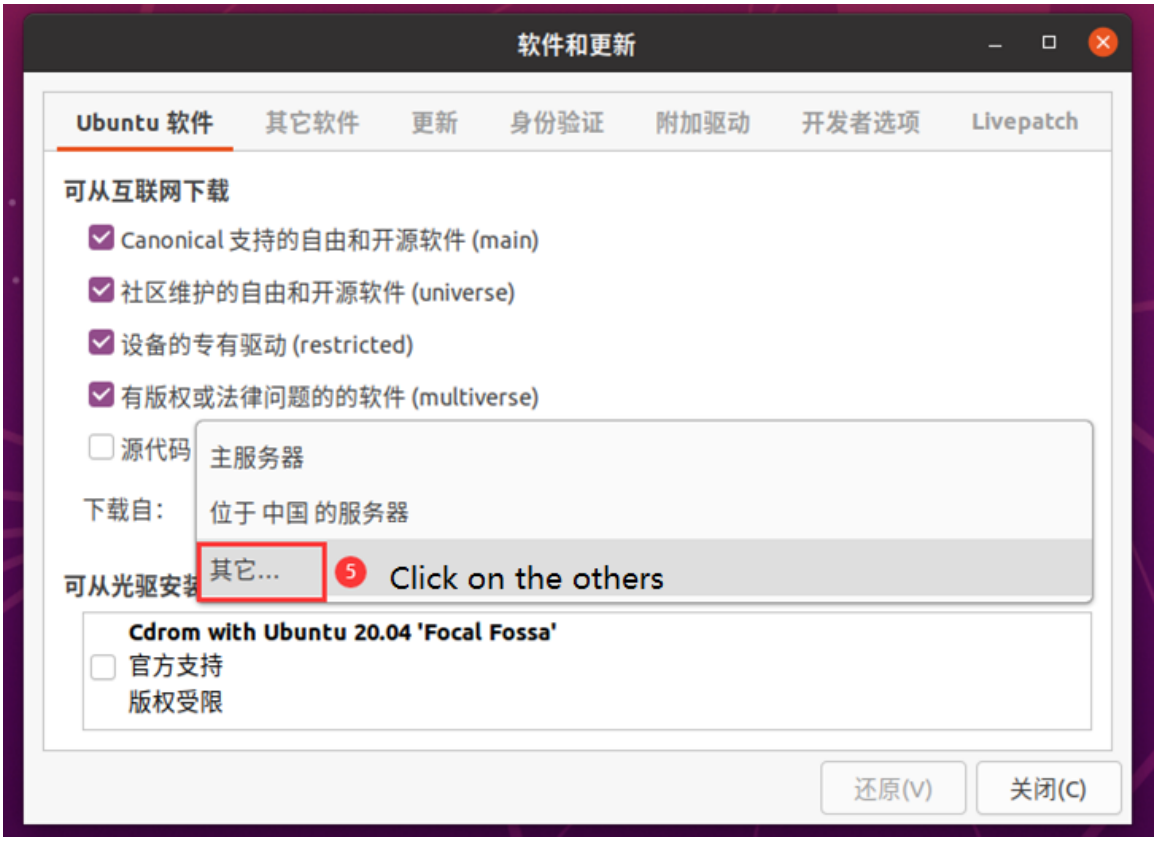


Figure 2.1.2.3 Change the software download source (3)

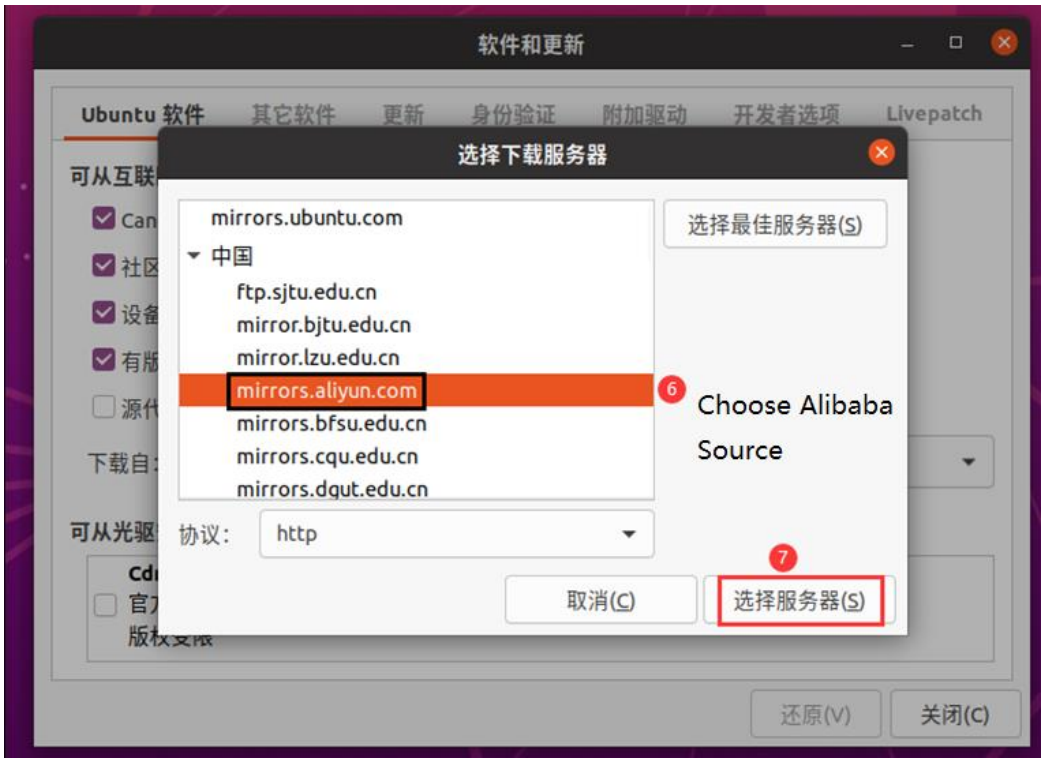


Figure 2.1.2.4 Change the software download source (4)

From the list, it can be seen that there are many domestic download sources for us to choose from. Here, we take the Alibaba source mirrors.aliyun.com as an example. Of course, other download sources can also be selected.

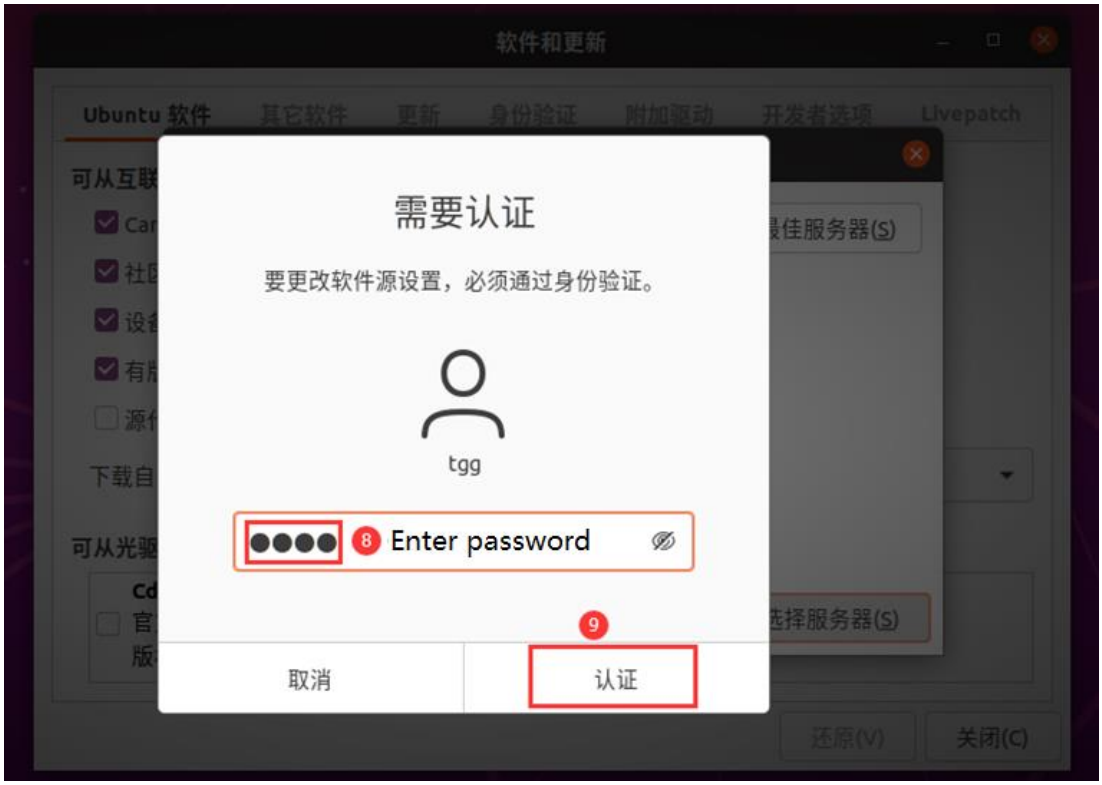


Figure 2.1.2.5 Change the software download source (5)

Figure 2.1.2.6 Change the software download source (6)

Figure 2.1.2.7 Change the software download source (7)

Figure 2.1.2.8 Change the software download source (8)

2.1.3 Disable automatic updates

Similarly, open the "**Software & Updates**" window in the Ubuntu system, and follow the steps shown in Figure 2.1.3.1 to disable the automatic update function:

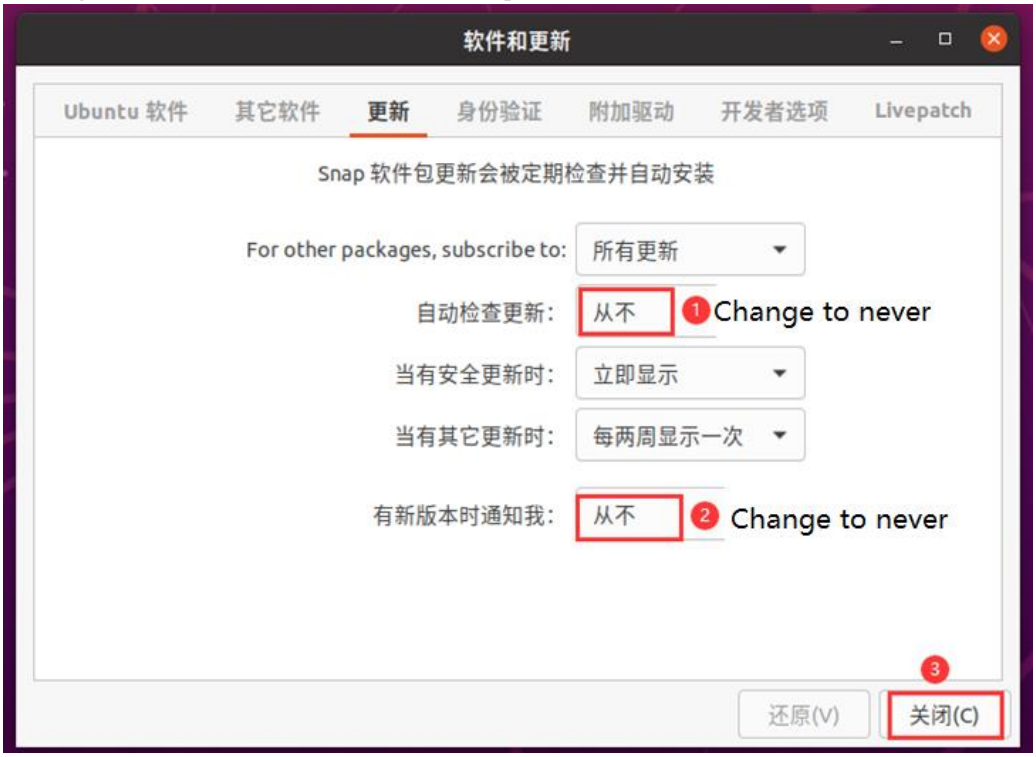


Figure 2.1.3.1 Disable the automatic update function

2.2 File Transfer Between Ubuntu and Windows

During the development process, we often need to transfer files between the Windows system and the Ubuntu system. There are many ways to transfer files, such as shared folders, Samba, FTP, scp, etc. This section introduces to users how to transfer files via FTP.

To transfer files between Windows and Ubuntu via FTP (File Transfer Protocol), two things need to be done.

2.2.1 Setting up an FTP Server on Ubuntu

Open the terminal in the Ubuntu system and execute the following command to install the FTP service:

```
sudo apt-get update
sudo apt-get install vsftpd
```

After the vsftpd installation is completed, use the vi command to open the /etc/vsftpd.conf configuration file. If the vim software is not installed, you need to install the vim editor first by using the following command:

```
sudo apt-get install vim
```

Then execute the following command to open the /etc/vsftpd.conf configuration file:

```
sudo vi /etc/vsftpd.conf
```

After opening the configuration file, locate the following two lines. Make sure there is no "#" symbol in front of them (the "#" symbol indicates a comment, and we need to remove the comments):

```
#
# Uncomment this to allow local users to log in.
local_enable=YES
#
# Uncomment this to enable any form of FTP write command.
write_enable=YES
#
```

Figure 2.2.1.1 Modify the /etc/vsftpd.conf configuration file

By default, the line "write_enable=YES" starts with a "#" symbol. We need to remove this symbol to enable the configuration. After the modification, save and exit the file, and then execute the following command to restart the FTP service:

```
sudo /etc/init.d/vsftpd restart
```

You can confirm whether the FTP service is enabled by using the following command:

```
ps -aux | grep vsftpd | grep -v grep
```

```
tgg@tgg-virtual-machine:~$
tgg@tgg-virtual-machine:~$ ps -aux | grep vsftpd | grep -v grep
root         5469  0.0  0.0   6816  3076 ?        Ss   11:15   0:00 /usr/sbin/vsftpd /etc/vsftpd.conf
tgg@tgg-virtual-machine:~$
```

Figure 2.2.1.2 Confirm if the FTP service is enabled

2.2.2 Installing FTP client on Windows

We need to install an FTP client software on the Windows system. Here, we choose FileZilla as the FTP client software. This is a free FTP client software.

The FileZilla software installation package has been provided to users in the ATK-RK3568 development board data package. The path is: Development board optical disc A drive - Basic materials 4_softwares→FileZilla_3.61.0_win64-setup.exe. Users can also download and install the package file through the link address: https://www.filezilla.cn/download/client.

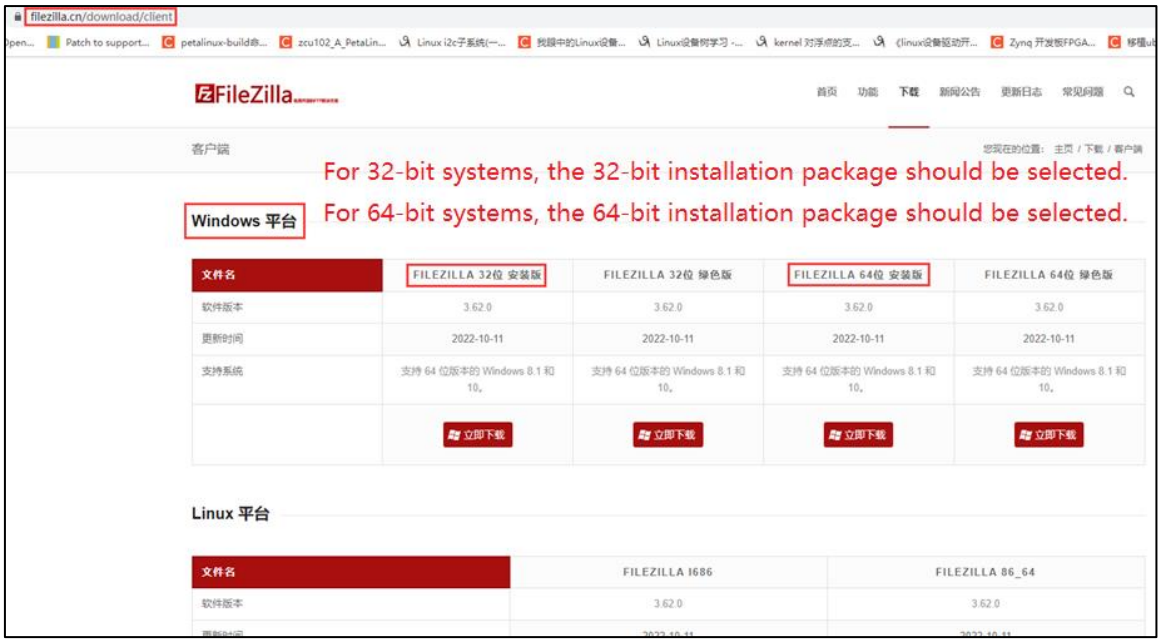


Figure 2.2.2.1 Download the installation package of FileZilla software

After the download is complete, it is as shown in Figure 2.2.2.2:

Figure 2.2.2.2 FileZilla software installation package

Next, double-click the installation package file FileZilla_3.61.0_win64-setup.exe, and follow the steps shown in Figures 2.2.2.3 to 2.2.2.8 to complete the installation:

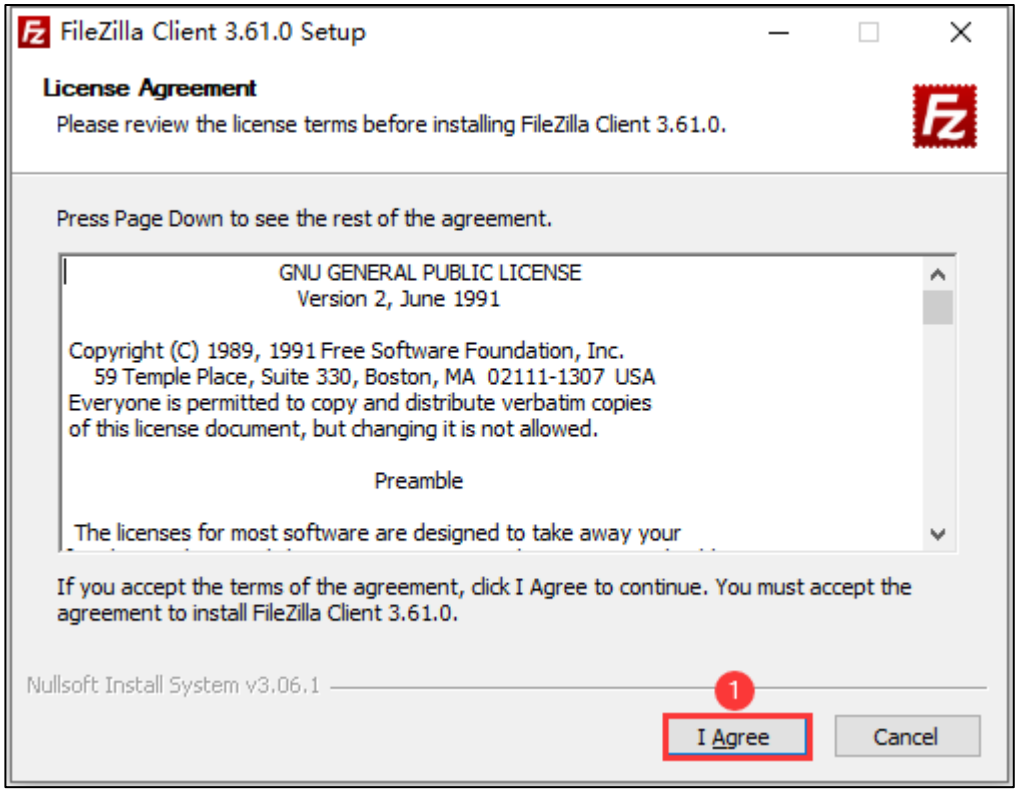


Figure 2.2.2.3 Install the FileZilla software (1)

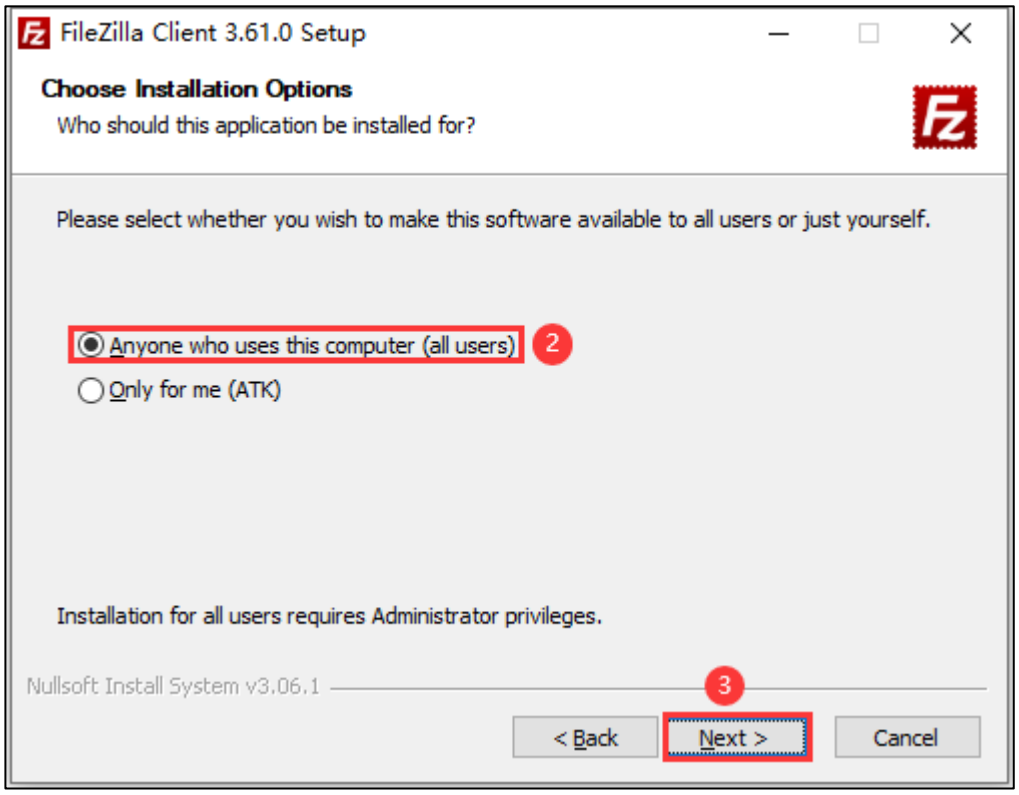


Figure 2.2.2.4 Install the FileZilla software (2)

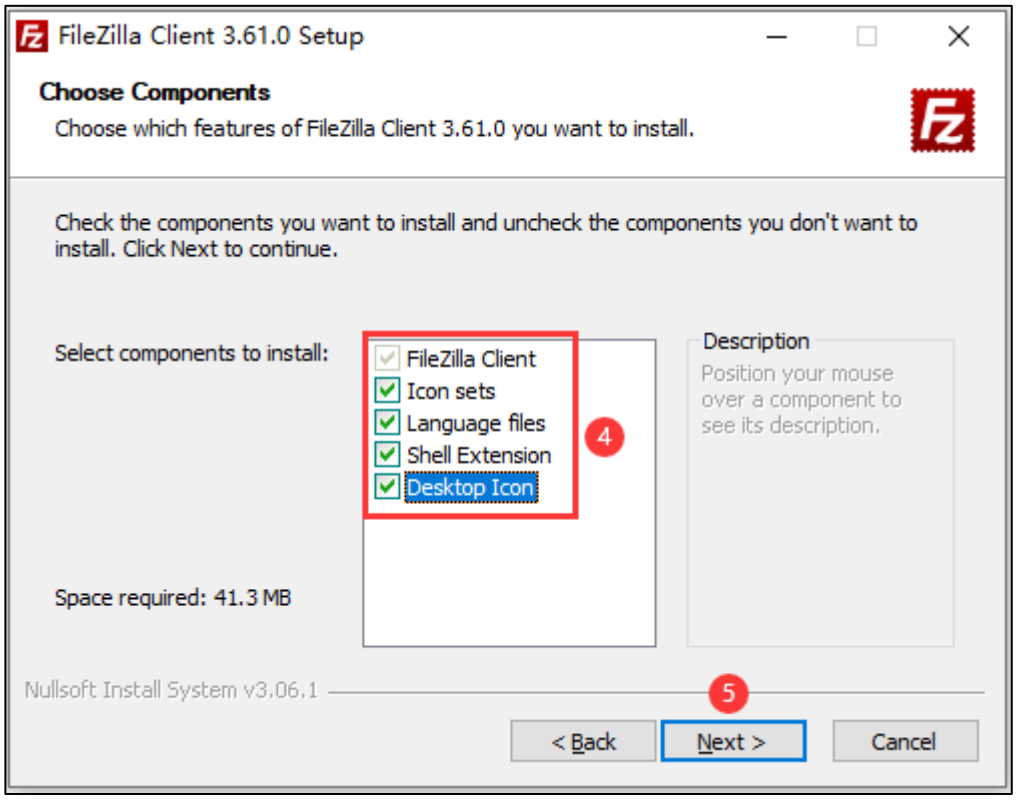


Figure 2.2.2.5 Install the FileZilla software (3)

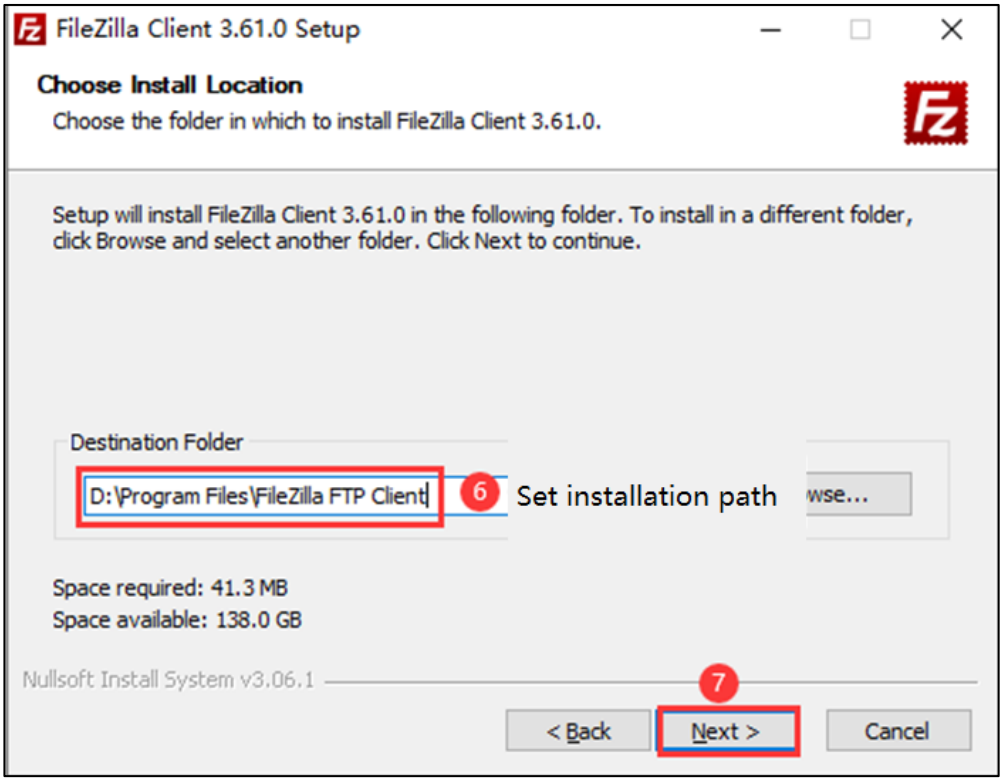


Figure 2.2.2.6 Install the FileZilla software (4)

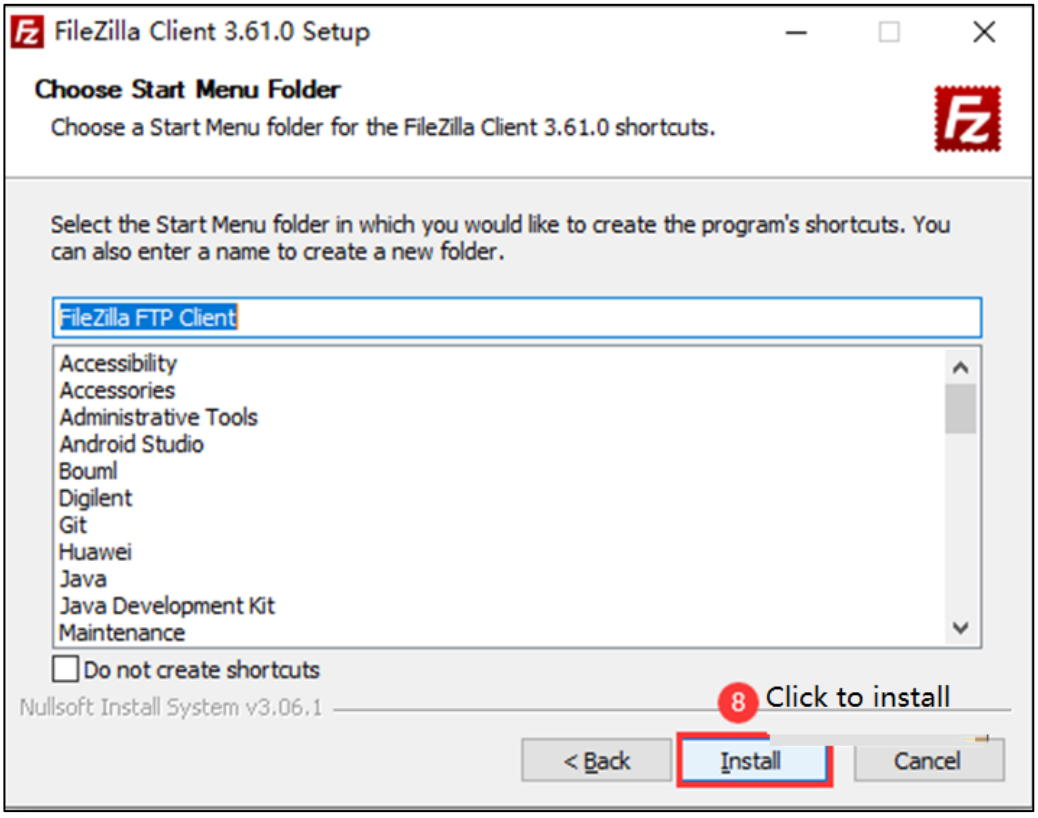


Figure 2.2.2.7 Install the FileZilla software (5)

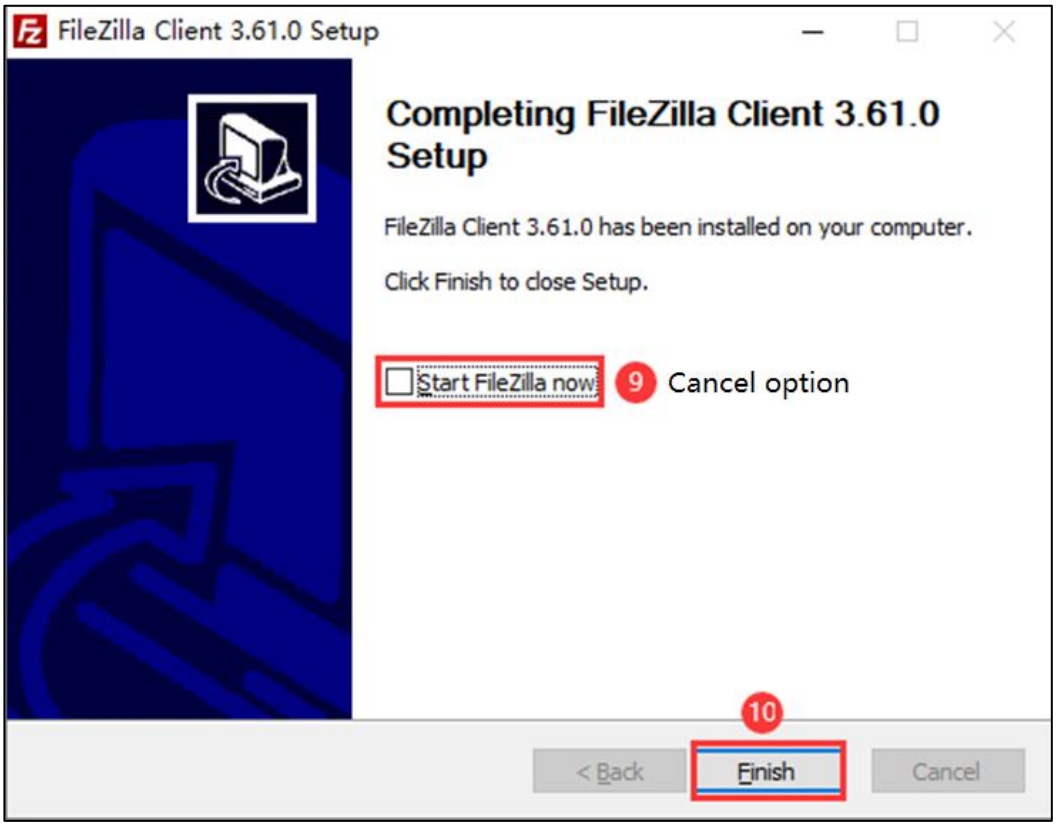


Figure 2.2.2.8 Install the FileZilla software (6)

At this point, the software installation is complete and a corresponding shortcut will be created on the desktop.

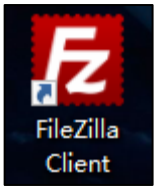


Figure 2.2.2.9 FileZilla desktop icon

2.2.3 How to use FileZilla

Now let's introduce the usage method of FileZilla. First, double-click the FileZilla desktop icon to open the software:

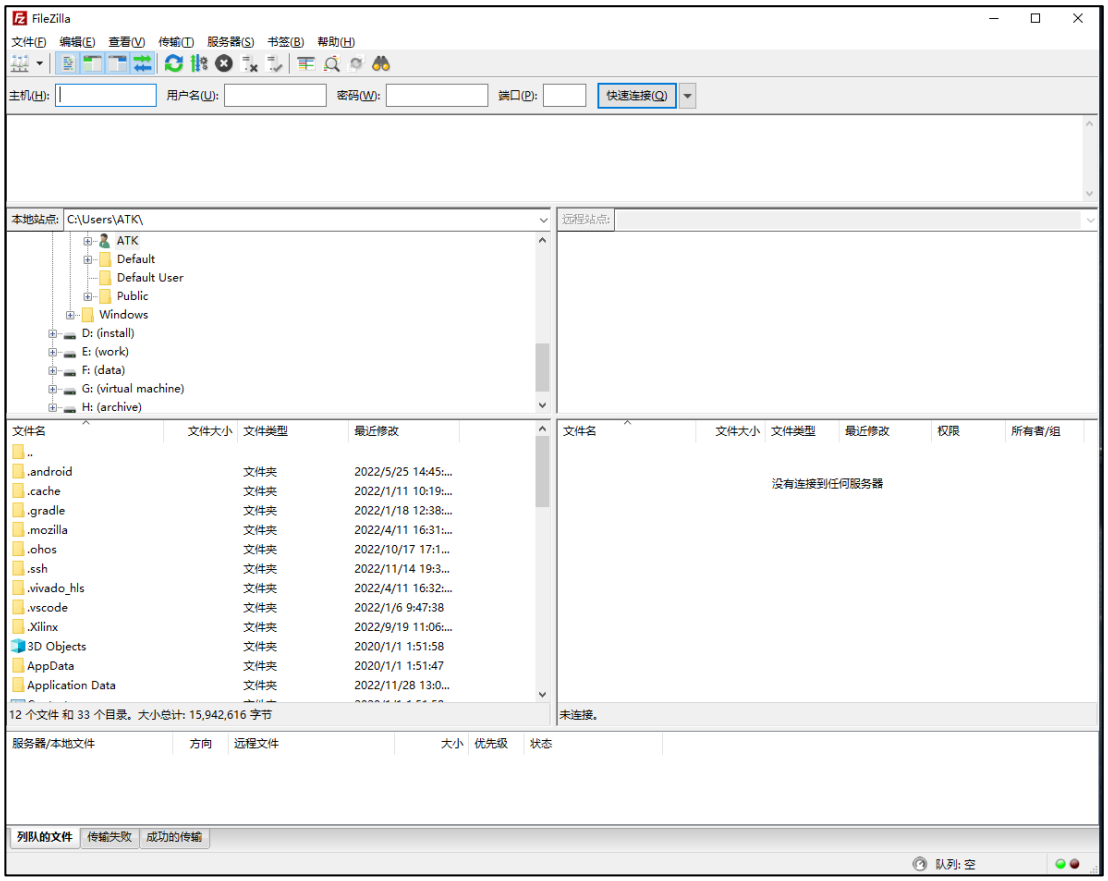


Figure 2.2.3.1 FileZilla software interface

Ubuntu acts as the FTP server, while Windows serves as the FTP client. Before conducting file transfer, the client needs to connect to the server first. Follow the steps shown in Figures 2.2.3.2 to 2.2.3.6 to connect to the FTP server:

Figure 2.2.3.2 FTP client connects to the FTP server (1)

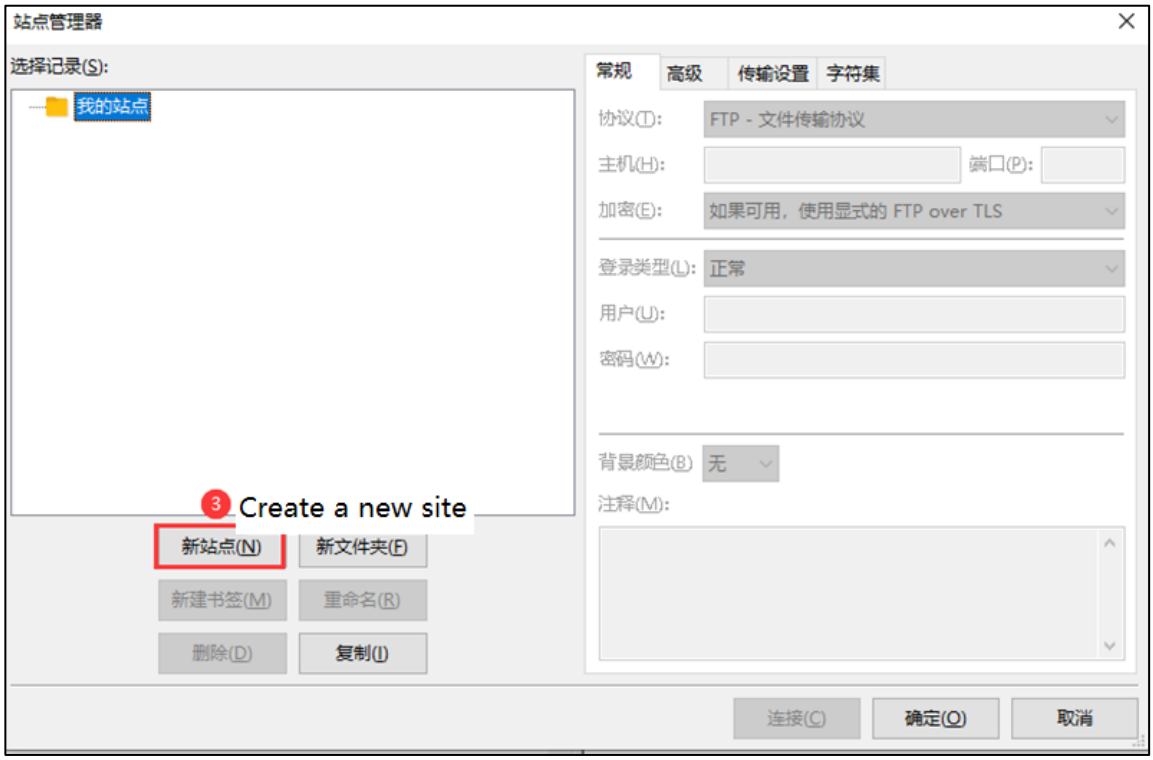


Figure 2.2.3.3 FTP client connects to the FTP server (2)

Figure 2.2.3.4 FTP client connects to the FTP server (3)

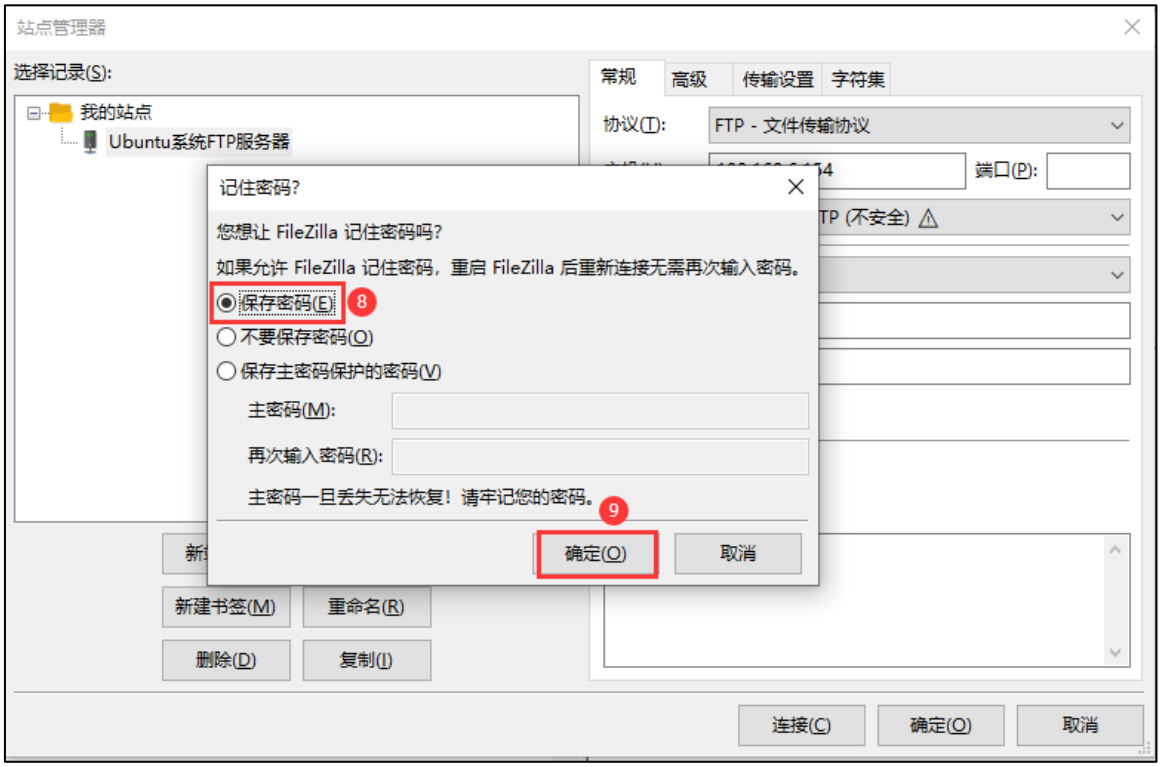


Figure 2.2.3.5 FTP client connects to the FTP server (4)

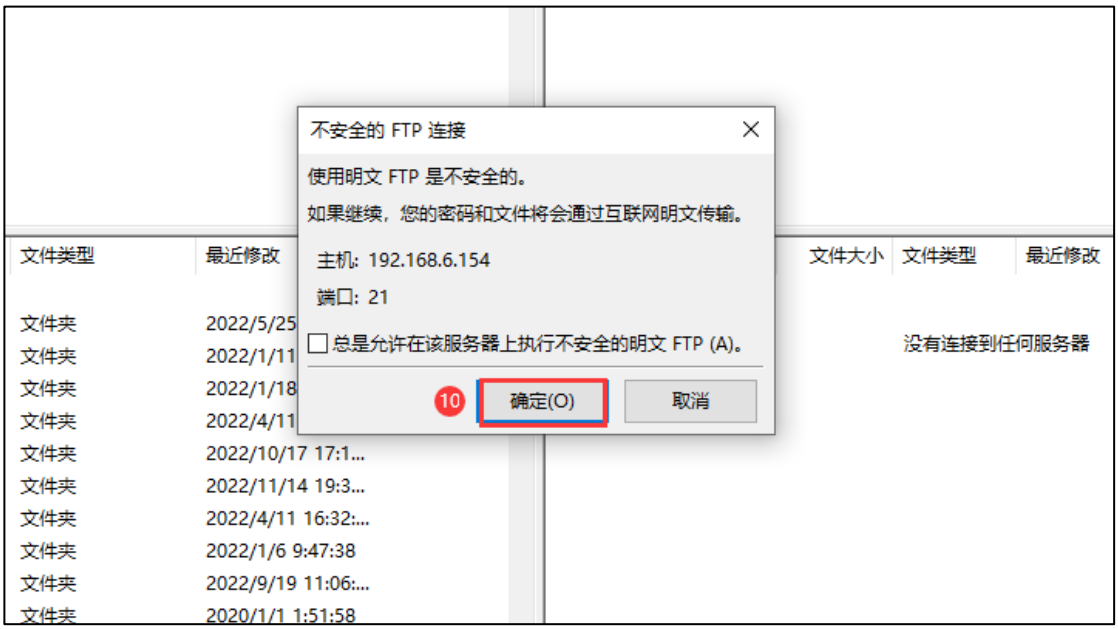


Figure 2.2.3.6 FTP client connects to the FTP server (5)

The connection was successful as shown in Figure 2.2.3.7:

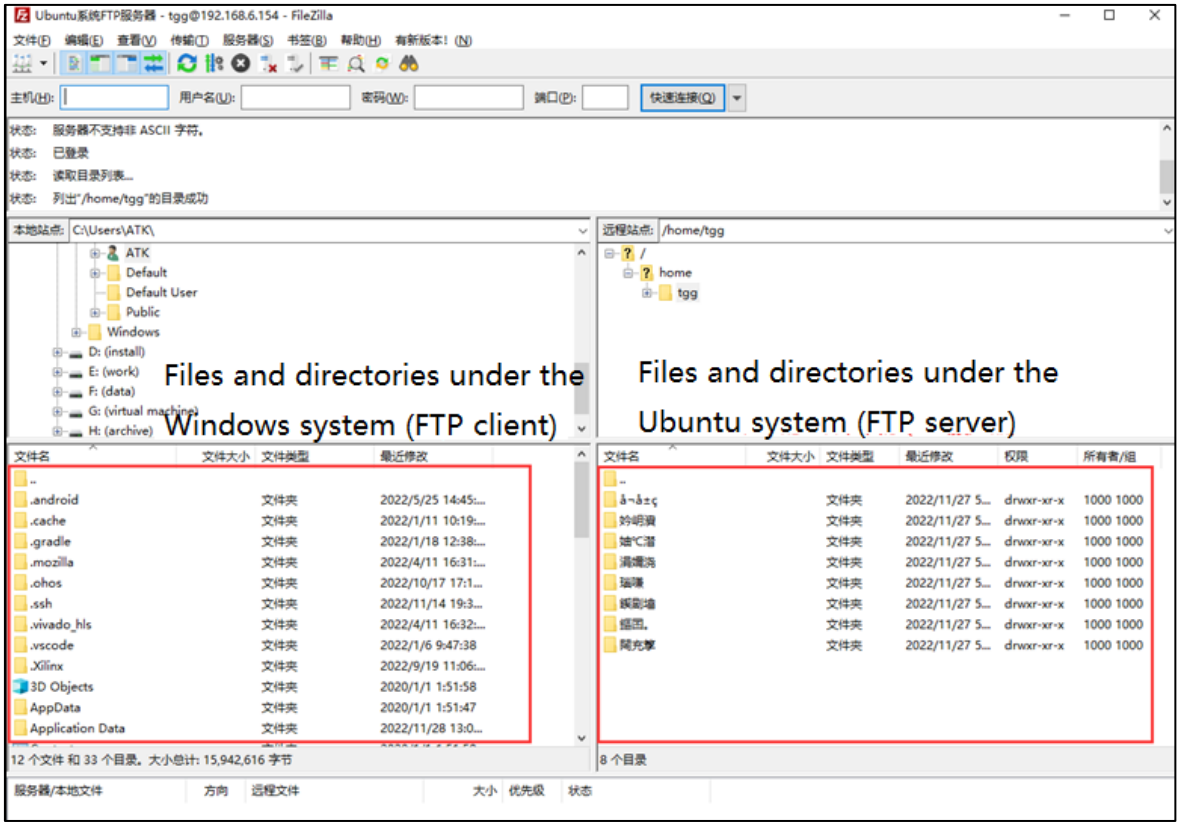


Figure 2.2.3.7 Connection successful

On the left is the file and directory under the Windows system (FTP client), and on the right is the file and directory under the Ubuntu system (FTP server, it will default to the user's home directory, such as "/home/tgg"). From the above picture, it can be seen that the names of the file list under the Ubuntu system are all in garbled, which is caused by the encoding problem. We need to modify the encoding

method and reopen the "Site Manager", and follow the steps shown in Figure 2.2.3.8 to set the encoding method：

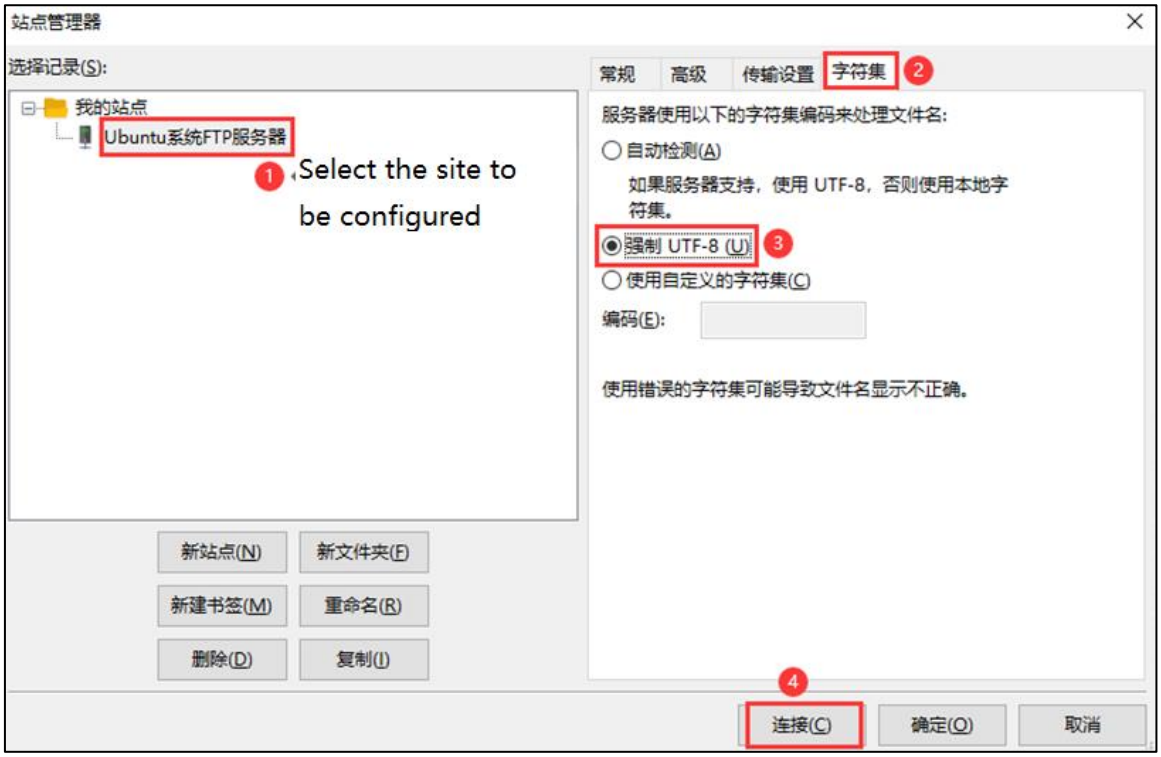


Figure 2.2.3.8 Set character encoding method (1)

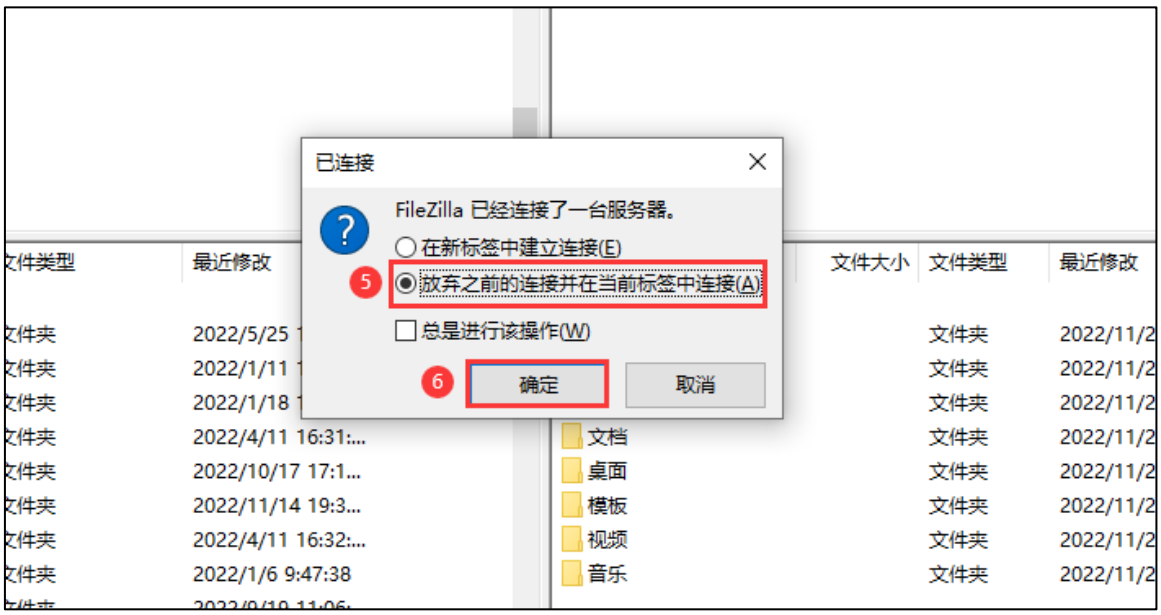


Figure 2.2.3.9 Set character encoding method (2)

After reconnecting, you will find that the file names are displayed normally, as shown in Figure 2.2.3.10:

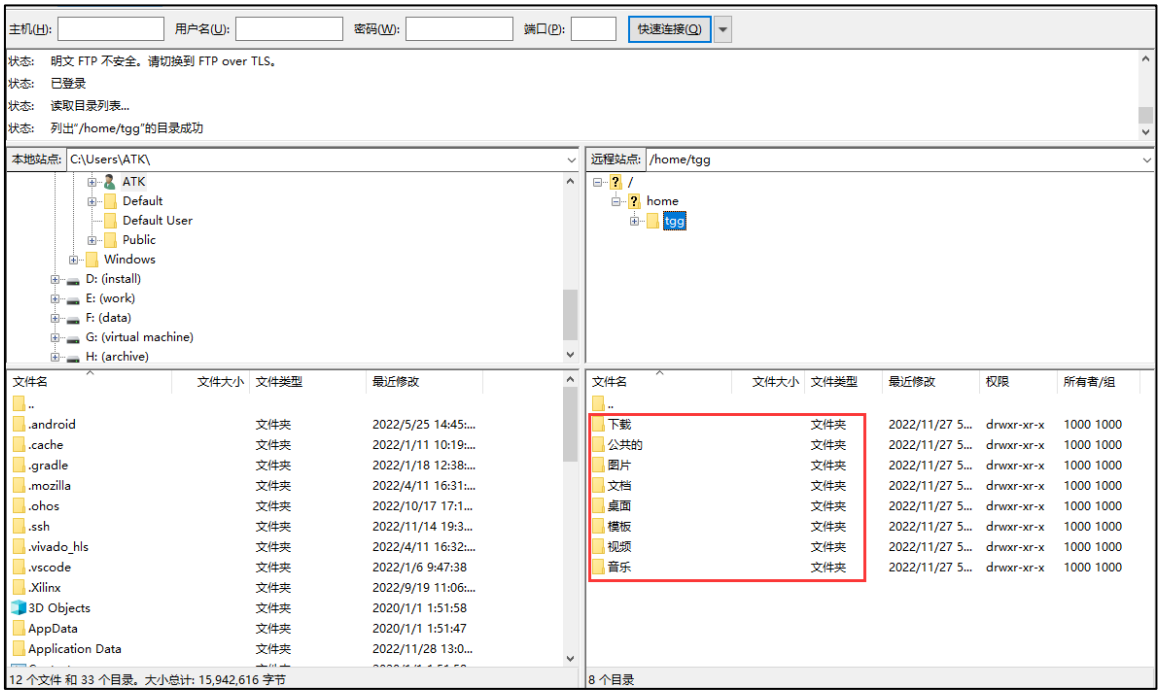


Figure 2.2.3.10 The file name is displayed normally.

After the client successfully connects to the server and is authenticated, file transfer can be carried out. The transfer method is very simple. Select the file to be transferred and directly drag it using the left mouse button to the Windows directory area or the Ubuntu system directory area! For example, to copy the test.txt file under the Windows system to the Ubuntu system, first find the file test.txt in the left Windows directory area, then use the left mouse button to drag it to the right Ubuntu directory area and release it; similarly, copying files from the Ubuntu system to the Windows system is also the same!

2.3 Setting up the tftp server in the Ubuntu system

Consider adding it later if there is a need!

2.4 Set up an NFS server on the Ubuntu system

If there is a need in the future, we can decide whether to add it!

2.5 Set up an SSH server on the Ubuntu system

During the development process, sometimes it is necessary to remotely connect and log in to the Ubuntu system via SSH. To achieve this function, we need to enable the SSH service on the Ubuntu system. Execute the following command to install openssh-server:

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

After installing the openssh-server, the ssh service will be automatically started. You can confirm whether the ssh service is running by using the following command:

```
ps -aux | grep ssh | grep -v grep
```

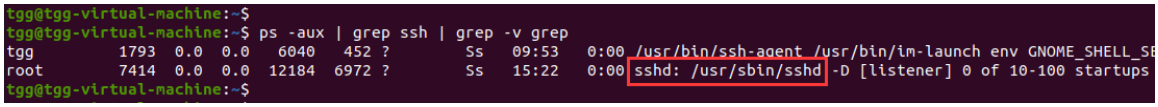


Figure 2.5-1 Confirm whether the SSH service is enabled

2.6 Installation of CH340 serial port driver

The ALIENTEK ATK-DLRK3568 development board uses the domestic chip CH340 to achieve USB to serial port function. It can directly connect the debugging serial port of the development board to the computer through a USB cable, without the need for a USB to serial port adapter, making it convenient for users to use; however, the CH340 driver needs to be installed under Windows in order to recognize the debugging serial port of the development board.

The development board package has already provided the installation file of the CH340 driver for users. The path is: Development board optical disc A drive - Basic materials→4_softwares→CH340 Driver (USB Serial Port Driver) _ XP_WIN7 Common→SETUP.EXE. Double-click the SETUP.EXE

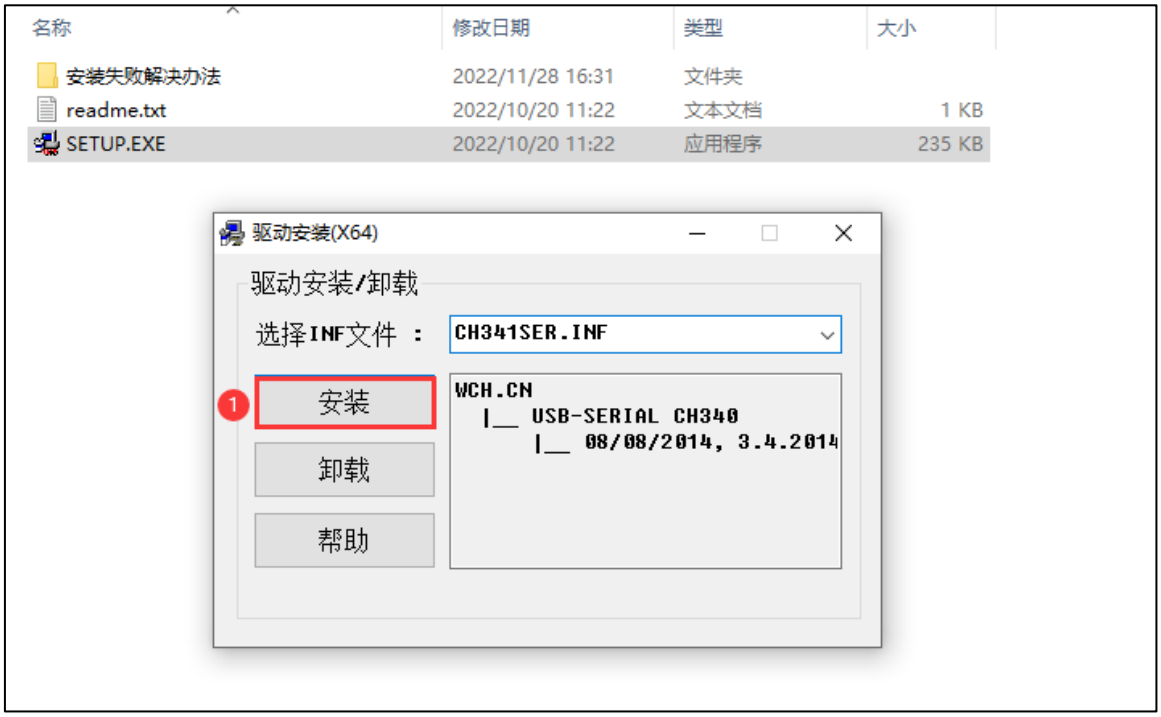


Figure 2.6-1 Install CH340 driver (1)

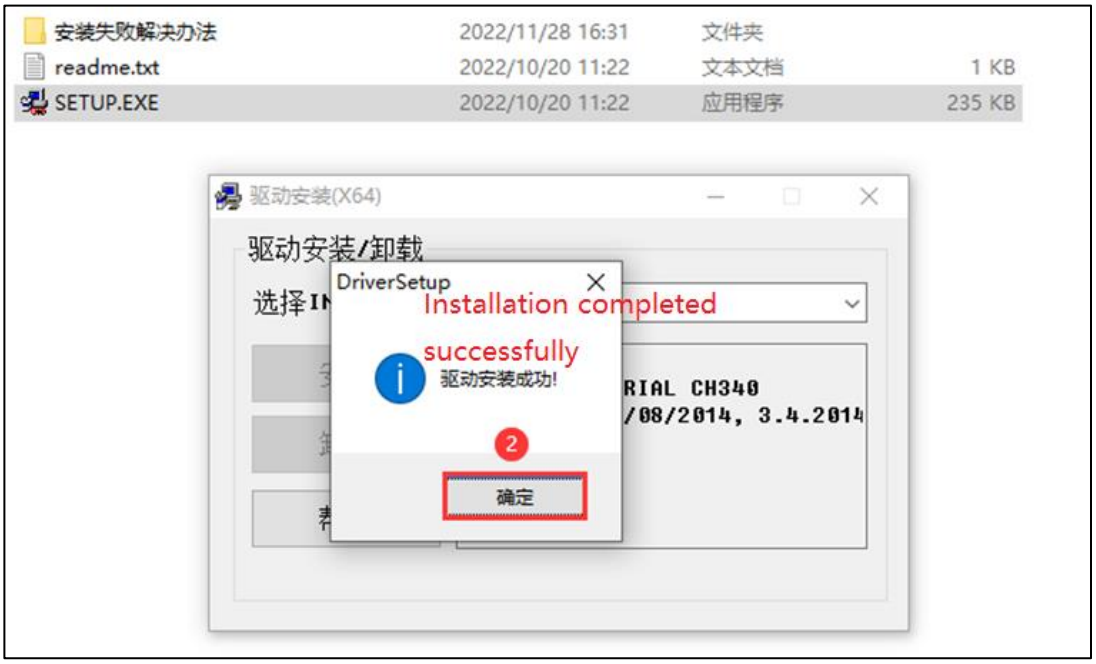


Figure 2.6-2 Install CH340 driver (2)

After the installation is completed, connect the debugging serial port of the development board to the computer via a USB cable. The connection method is shown in Figure 2.6.3:

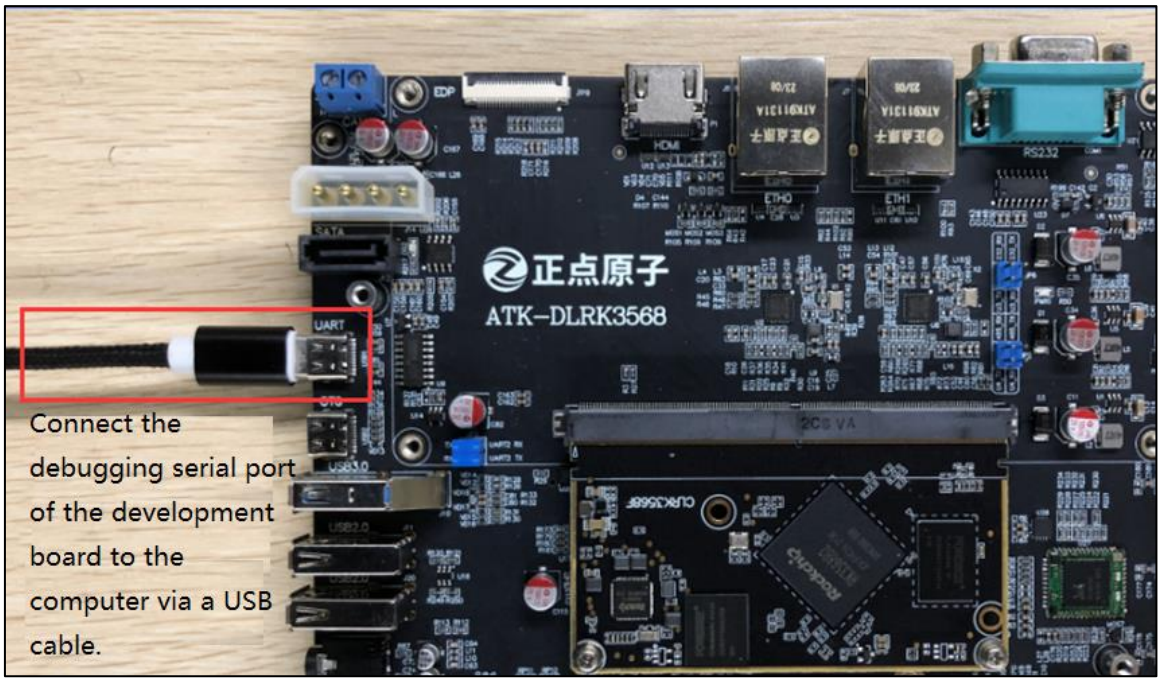


Figure 2.6-3 The development board is connected to the computer via the serial port for debugging.

After the connection is successful, at this point, the computer will detect a USB device and open the Windows Device Manager:

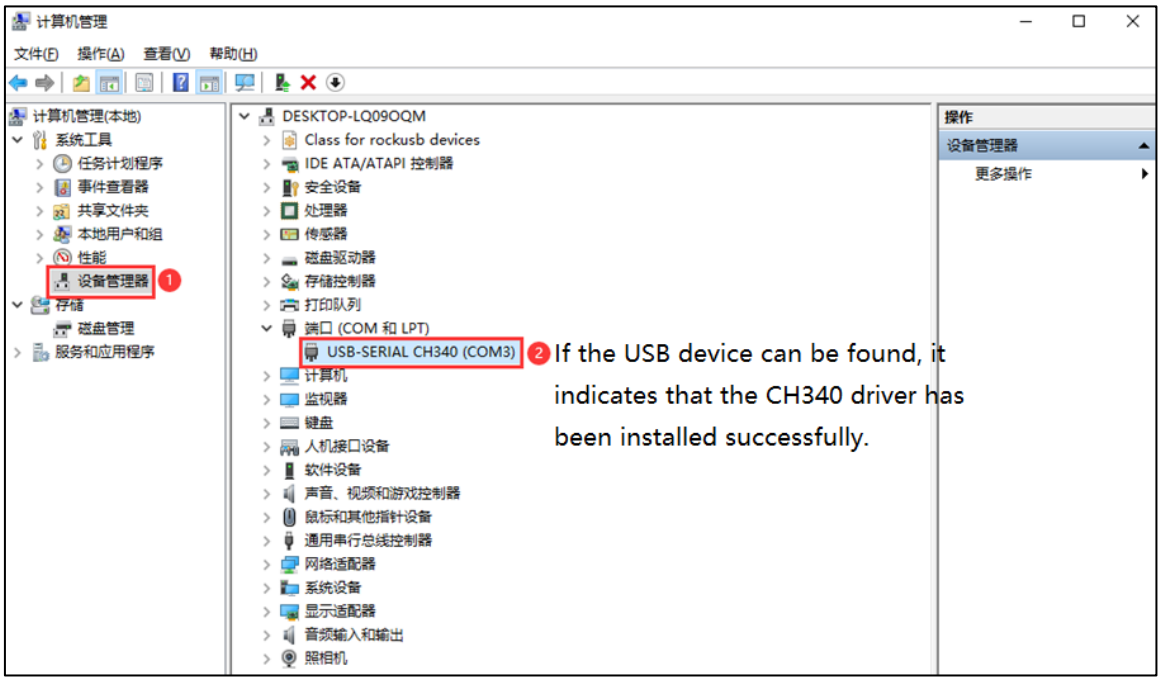


Figure 2.6-4 Windows Device Manager

If a device named "**USB-SERIAL→CH340**" can be found under "**Device Manager Ports (COM and LPT)**", it indicates that the CH340 driver installation was successful! If the device named "**USB-SERIAL CH340**" cannot be found, please have the user check the hardware connection for any errors! You can unplug the USB cable, reconnect it, or try connecting to another USB port on the computer.

2.7 Installation of MobaXterm software

MobaXterm is a multi-functional remote terminal software with powerful functions and is free (there are also paid versions). It supports creating various remote connection functions such as SSH, Telnet, Rsh, Xdmc, RDP, VNC, FTP, SFTP, serial port (Serial COM), etc. MobaXterm provides a user-friendly interface and is very powerful in function, so it is recommended that users use this terminal software.

2.7.1 Download of MobaXterm software

The development board data package has already provided the installation package of MobaXterm to users. The path is: the A drive of the development board CD-ROM - Basic Materials → 4_softwares MobaXterm_Installer_v12.3.zip; Users can also download it themselves through the link address: https://mobaxterm.mobatek.net/download.html

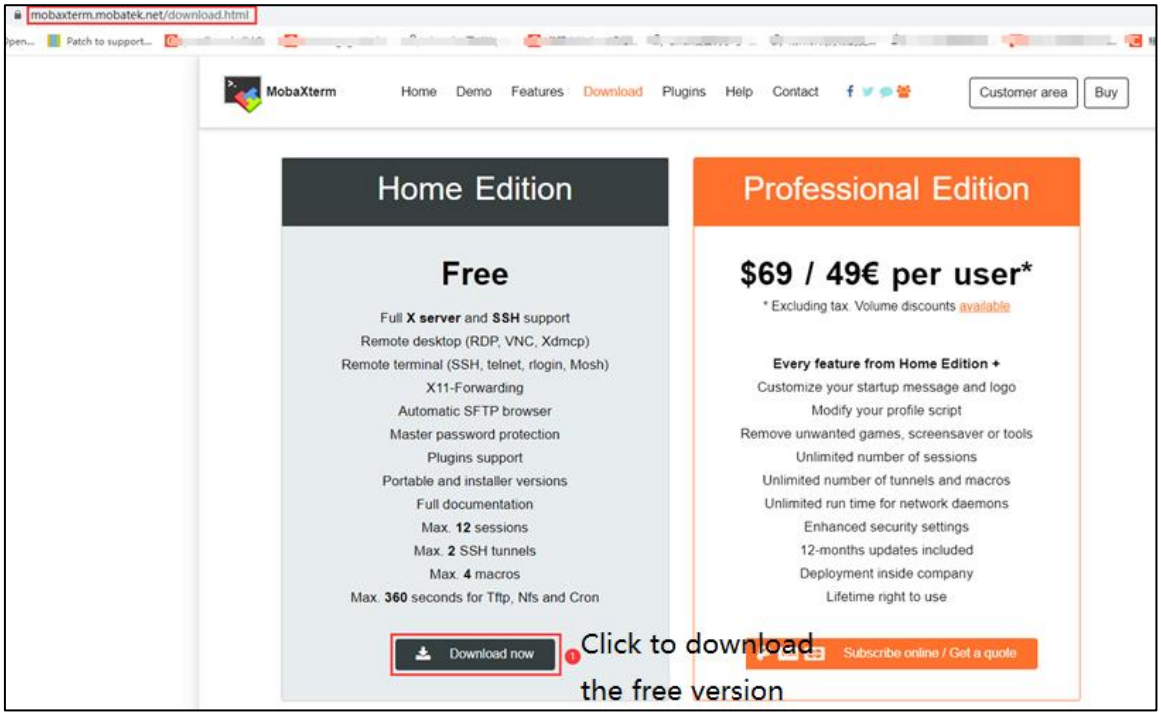


Figure 2.7.1.1 Download the MobaXterm software (1)

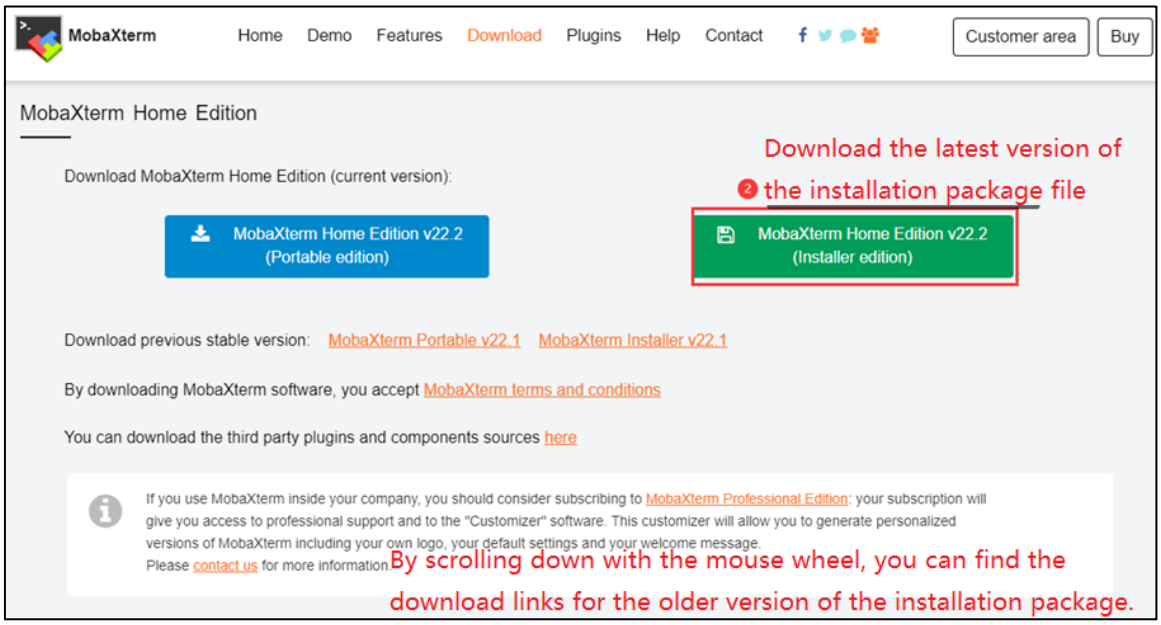


Figure 2.7.1.2 Download the MobaXterm software (2)

After the download is complete, a compressed file named "MobaXterm_Installer_vxxx.zip" (where xxx represents the version number) will be obtained. Currently, the latest version is 22.2. The version of the installation package provided for users in the data package is 12.3. You can use the new version or the old one, it doesn't matter. Here, we take the 12.3 version as an example.

2.7.2 Installation of MobaXterm software

Extract the MobaXterm_Installer_v12.3.zip compressed file. After the extraction, it will look like this:

Figure 2.7.2.1 The files after extracting MobaXterm_Installer_v12.3.zip

Then double-click to run the MobaXterm_Installer_v12.3.msi file and follow the steps shown in Figures 2.7.2.2 to 2.7.2.6 to install the MobaXterm software:

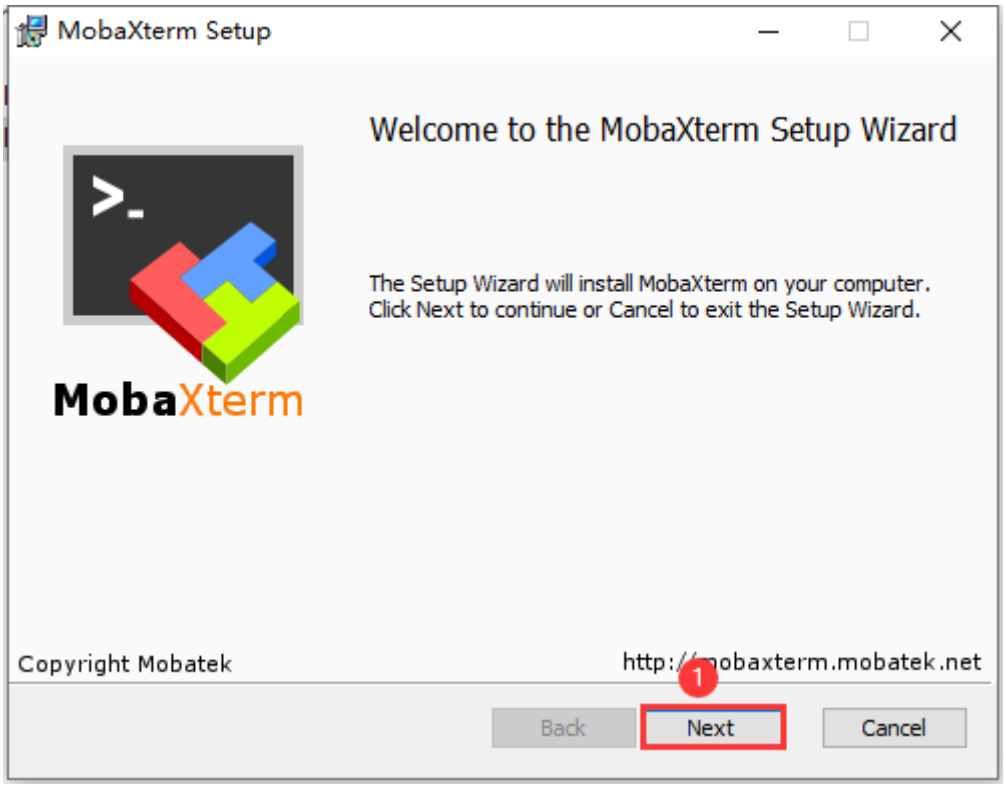


Figure 2.7.2.2 Install the MobaXterm software (1)

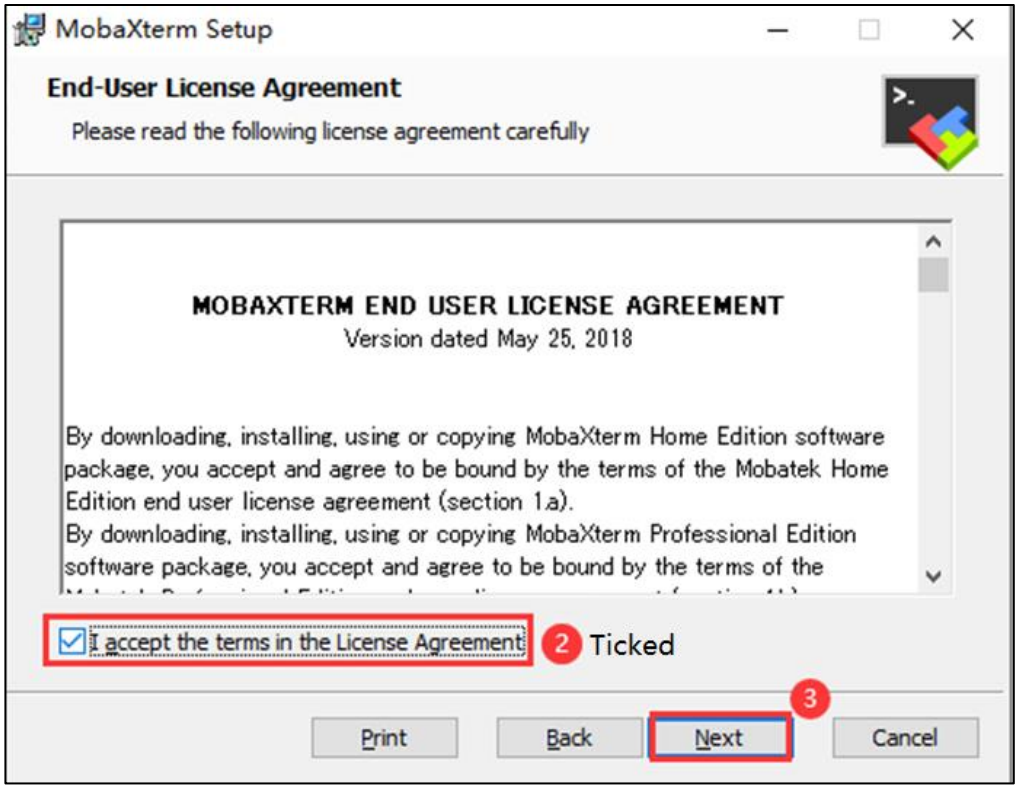


Figure 2.7.2.3 Install the MobaXterm software (2)

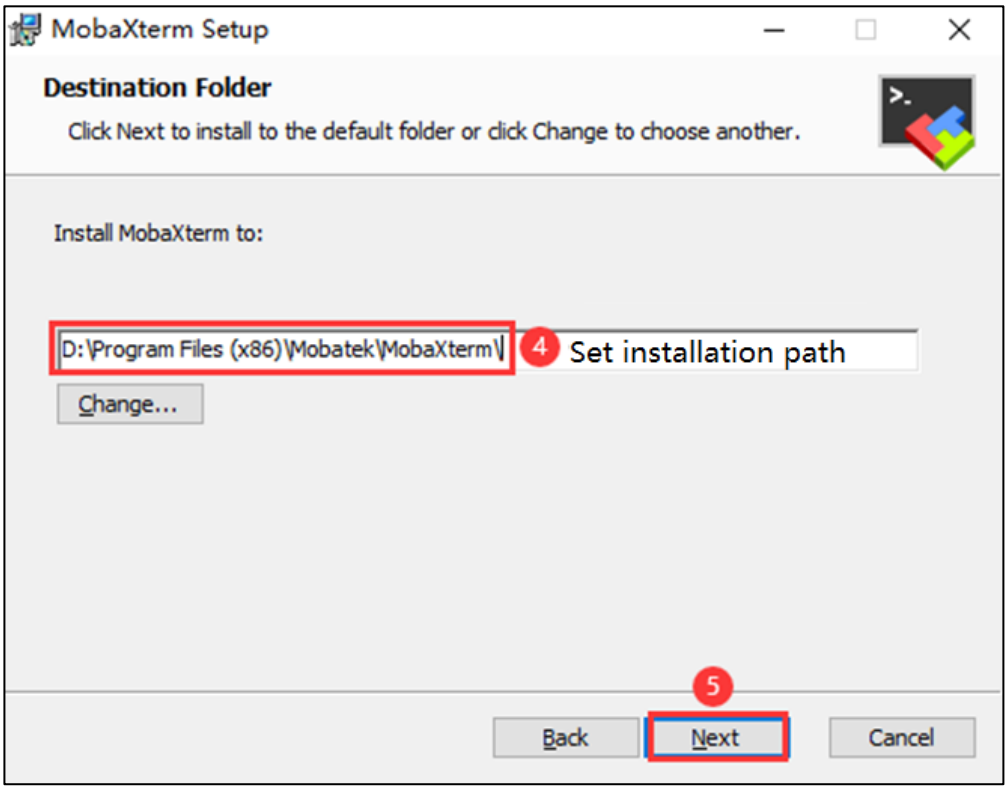


Figure 2.7.2.4 Install the MobaXterm software (3)

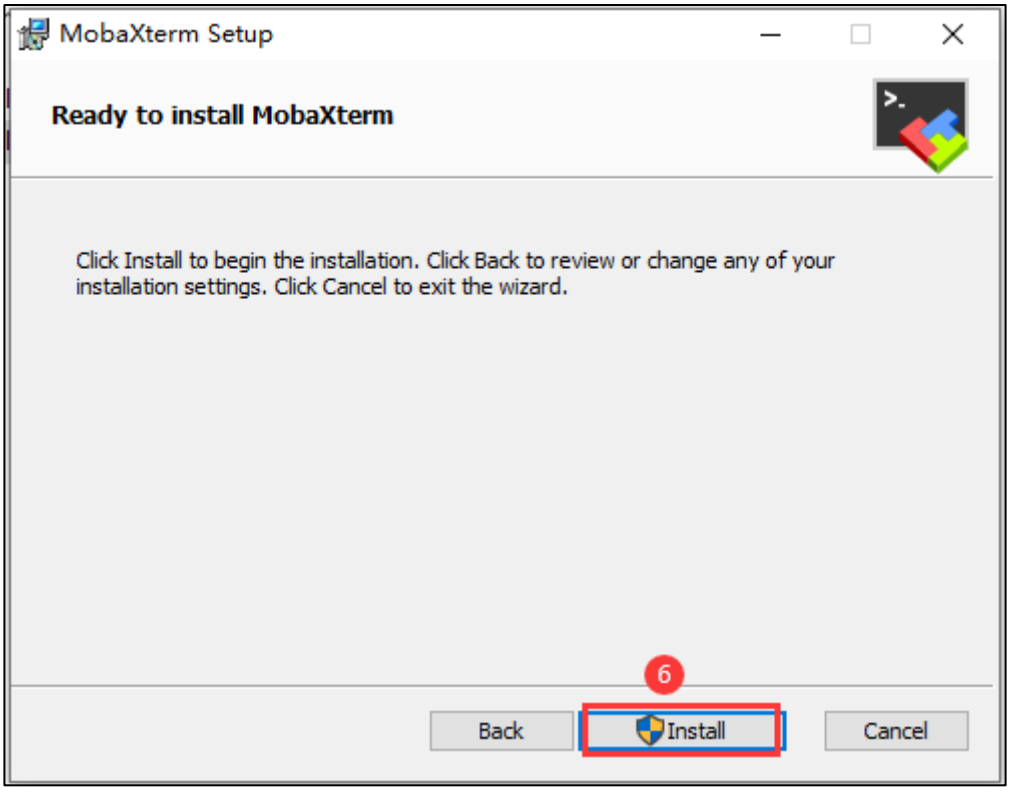


Figure 2.7.2.5 Install the MobaXterm software (4)

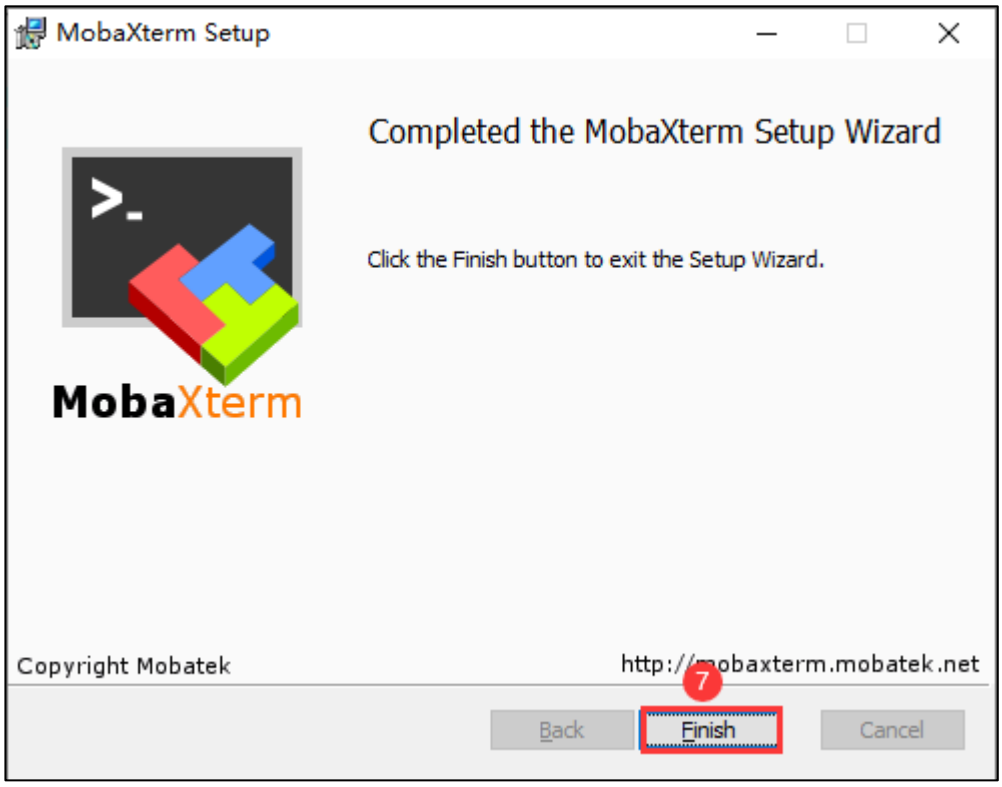


Figure 2.7.2.6 Install the MobaXterm software (5)

At this point, the software installation is complete. The desktop will automatically generate a shortcut icon for the MobaXterm software.

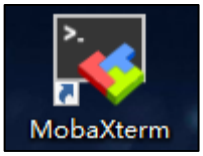


Figure 2.7.2.7 MobaXterm software desktop icon

2.7.3 Usage of MobaXterm software

Double-click the MobaXterm desktop icon to open the software, as shown in Figure 2.7.3.1:

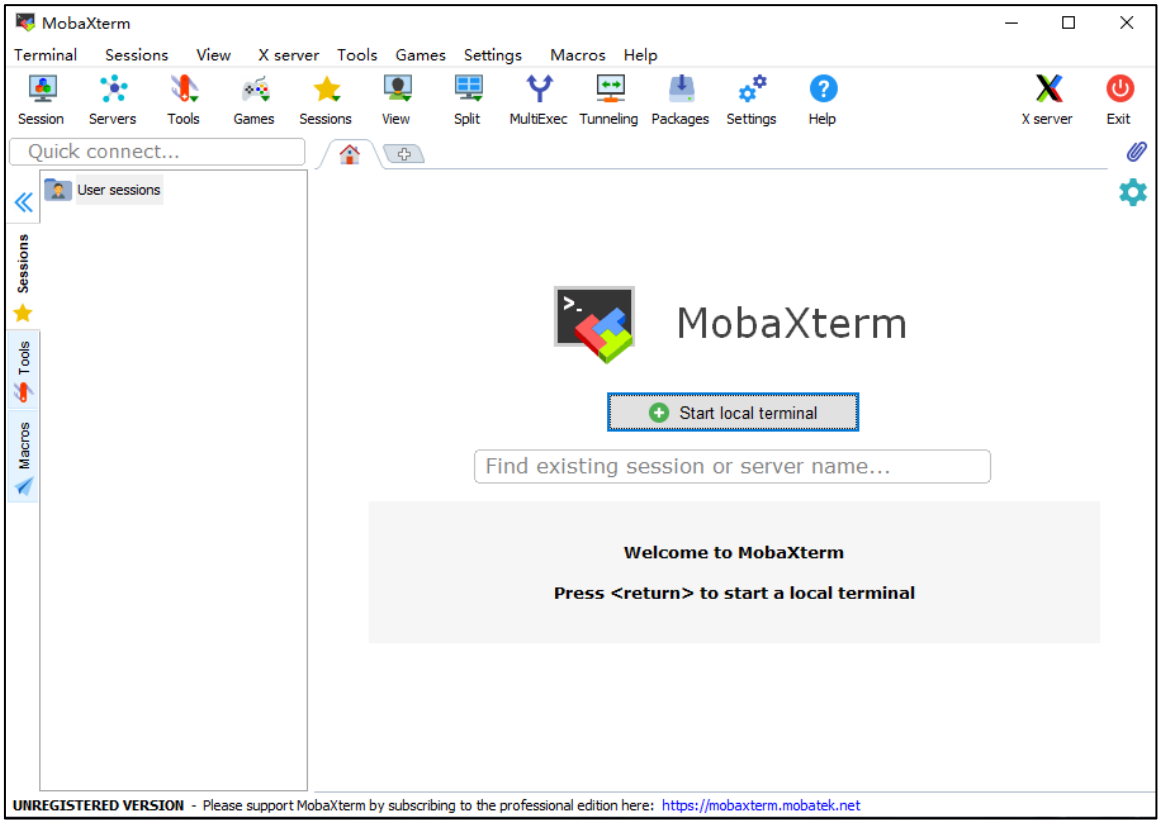


Figure 2.7.3.1 The main interface of MobaXterm software

Next, I will introduce how to establish a Serial (serial port) connection and an SSH remote connection.

1. Serial connection

There is a debugging serial port on the ATK-RK3568 development board, which can be directly connected to the computer via a USB cable. The serial port is one of the most common communication interfaces for embedded devices. It not only enables data transmission between the computer and the embedded device, but also allows the computer to control the embedded device. In the embedded development process, it is usually used as a debugging interface (serial port debugging) to debug the embedded device.

Follow the operation steps shown in Figures 2.7.3.2 to 2.7.3.3 to establish a Serial (serial port) connection (before establishing the connection, the debugging serial port of the development board needs to be connected to the computer via a USB cable, and the CH340 driver has been installed):

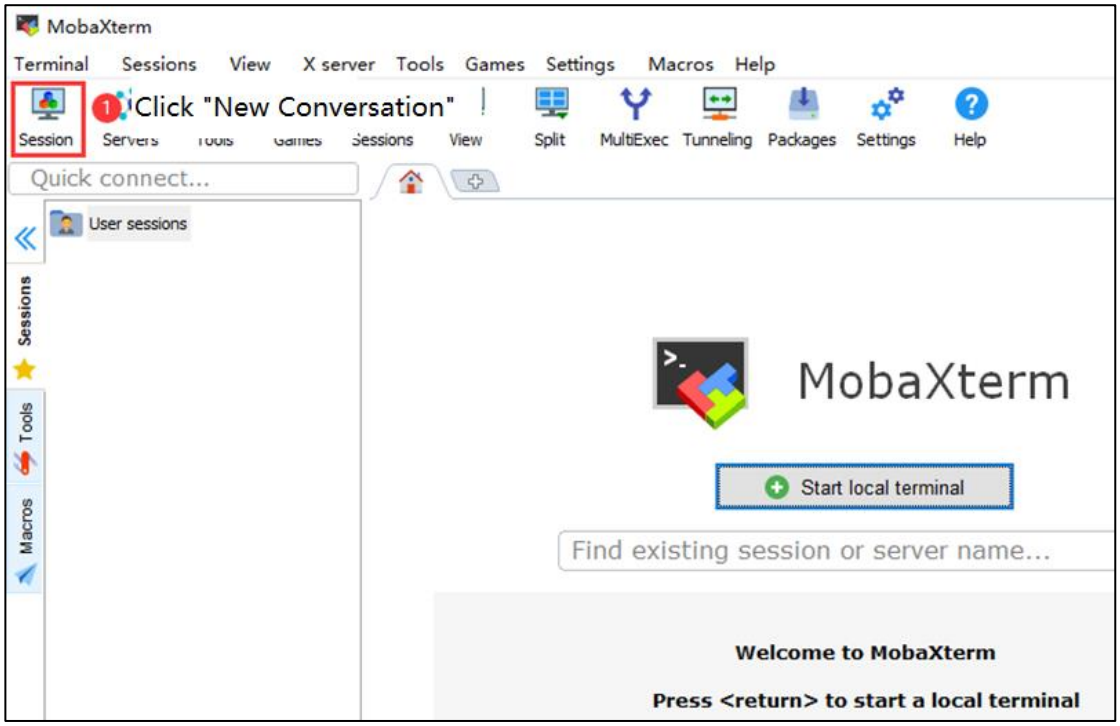


Figure 2.7.3.2 Establish serial port connection (1)

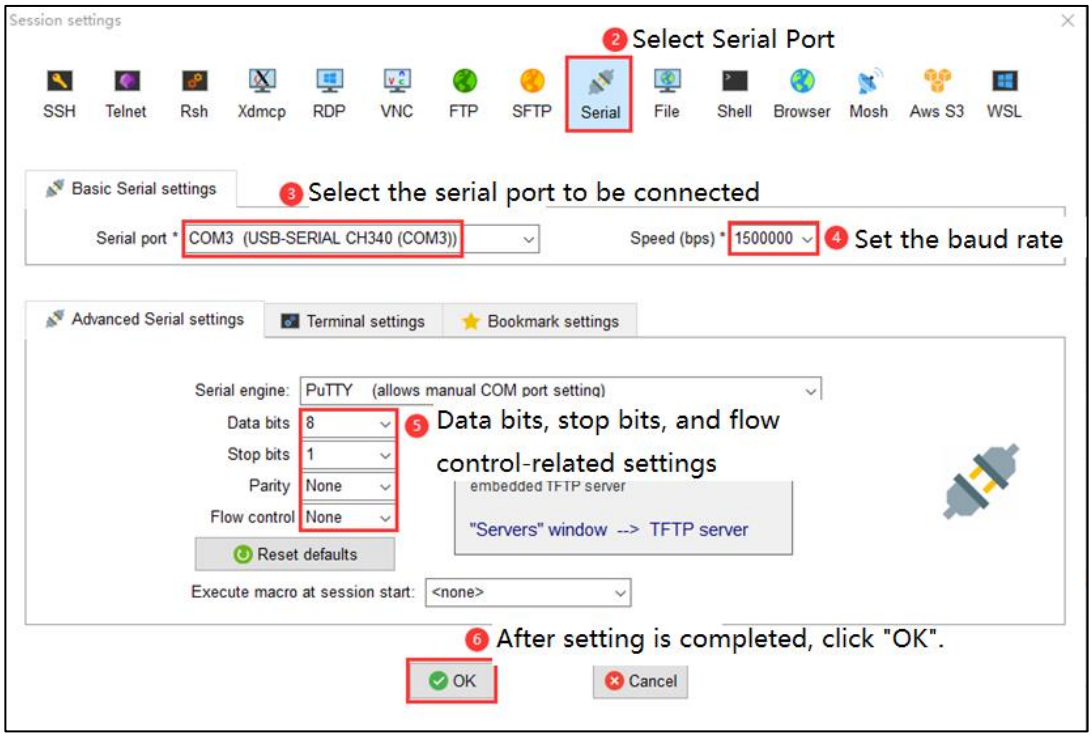


Figure 2.7.3.3 Establishing Serial Port Connection (2)

First, select the serial port that needs to be connected. Ensure that the debugging serial port of the development board is connected to the computer via a USB cable, and that the CH340 driver has been successfully installed ("**Device Manager→Ports (COM and LPT)**") and can be found under a device named "**USB-SERIAL CH340**". Only then can the MobaXterm software recognize the debugging

serial port of the development board. We can then find the corresponding serial port of the development board (**USB-SERIAL CH340**) in the "Serial port" drop-down list, and select it!

Next, set the serial communication baud rate. Adjust it according to the actual situation. For the debugging serial port on the RK3568 platform, the default baud rate is 1500000 (1.5M). Then set the data bits, stop bits, flow control, etc. After setting is completed, click the "OK" button.

The serial connection has been successfully established, as shown in Figure 2.7.3.4:

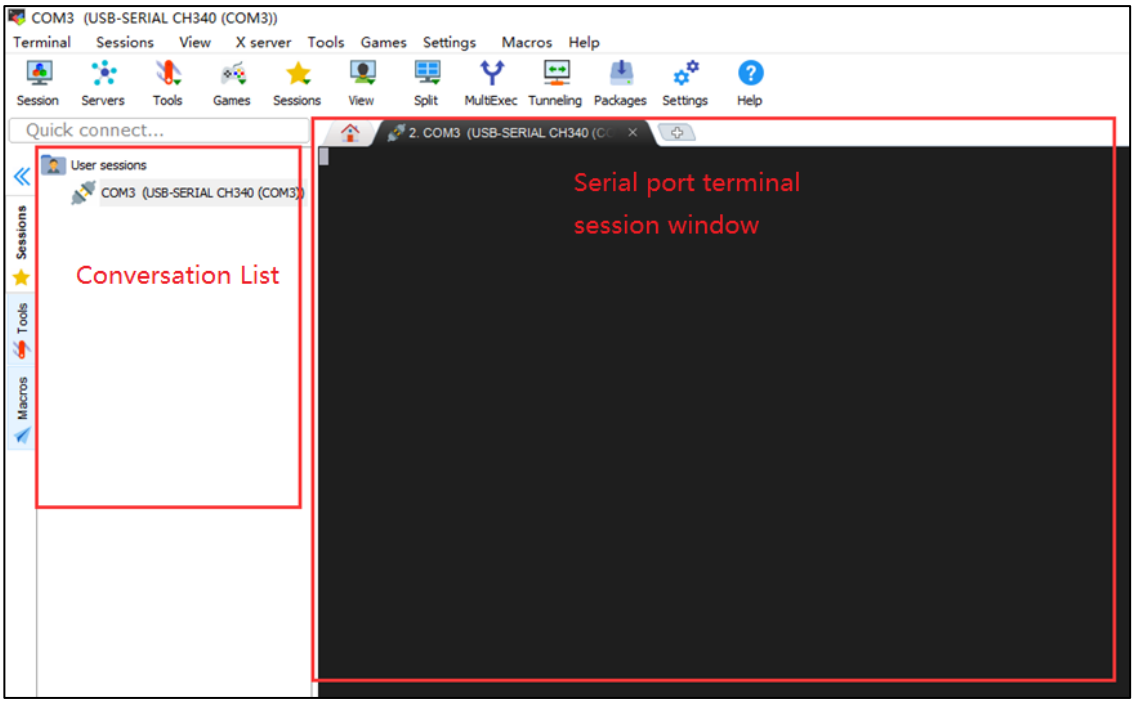


Figure 2.7.3.4 Establishing Serial Port Connection (3)

When the development board is powered on, it starts the Linux system or the Android system. During the system startup process, the log information will be displayed in the serial port terminal session window, as shown in Figure 2.7.3.5.

```
vp1 adjust cursor plane from 0 to 1
vp0, plane_mask:0x2a, primary-id:5, curser-id:1
vp1 adjust cursor plane from 1 to 0
vp1, plane_mask:0x15, primary-id:4, curser-id:0
vp2, plane_mask:0x0, primary-id:0, curser-id:-1
Adding bank: 0x00200000 - 0x08400000 (size: 0x08200000)
Adding bank: 0x09400000 - 0x80000000 (size: 0x76c00000)
Total: 1255.933 ms

Starting kernel ...

[    0.000000] Booting Linux on physical CPU 0x0000000000 [0x412fd050]
[    0.000000] Linux version 4.19.232 (alientek@android) (gcc version 6.3.1 20170404 (Linaro GCC 6.3-2017.05), GNU ld (Linaro_Binutils-201
[    0.000000] Machine model: Rockchip RK3568 ATK EVB1 DDR4 V10 Board
[    0.000000] earlycon: uart8250 at MMIO32 0x00000000fe660000 (options '')
[    0.000000] bootconsole [uart8250] enabled
[    0.000000] cma: Reserved 16 MiB at 0x000000007ec00000
[    0.000000] psci: probing for conduit method from DT.
[    0.000000] psci: PSCIv1.1 detected in firmware.
[    0.000000] psci: Using standard PSCI v0.2 function IDs
[    0.000000] psci: Trusted OS migration not required
[    0.000000] psci: SMC Calling Convention v1.2
[    0.000000] percpu: Embedded 23 pages/cpu s54184 r8192 d31832 u94208
[    0.000000] Detected VIPT I-cache on CPU0
[    0.000000] CPU features: detected: Virtualization Host Extensions
[    0.000000] CPU features: detected: Speculative Store Bypassing Safe (SSBS)
[    0.000000] Built 1 zonelists, mobility grouping on.  Total pages: 511496
[    0.000000] Kernel command line: storagemedia=emmc androidboot.storagemedia=emmc androidboot.mode=normal  androidboot.verifiedbootstate
000
[    0.000000] Dentry cache hash table entries: 262144 (order: 9, 2097152 bytes)
[    0.000000] Inode-cache hash table entries: 131072 (order: 8, 1048576 bytes)
[    0.000000] mem auto-init: stack:off, heap alloc:off, heap free:off
[    0.000000] Memory: 1993376K/2078720K available (13566K kernel code, 1970K rwdata, 4840K rodata, 1536K init, 514K bss, 68960K reserved,
[    0.000000] SLUB: HWalign=64, Order=0-3, MinObjects=0, CPUs=4, Nodes=1
[    0.000000] ftrace: allocating 49744 entries in 195 pages
[    0.000000] rcu: Hierarchical RCU implementation.
[    0.000000] rcu:     RCU event tracing is enabled.
[    0.000000] rcu:     RCU restricting CPUs from NR_CPUS=8 to nr_cpu_ids=4.
[    0.000000] rcu: Adjusting geometry for rcu_fanout_leaf=16, nr_cpu_ids=4
[    0.000000] NR_IRQS: 64, nr_irqs: 64, preallocated irqs: 0
[    0.000000] GICv3: GIC: Using split EOI/Deactivate mode
[    0.000000] GICv3: Distributor has no Range Selector support
[    0.000000] GICv3: no VLPI support, no direct LPI support
[    0.000000] ITS [mem 0xfd440000-0xfd45ffff]
[    0.000000] ITS@0x00000000fd440000: allocated 8192 Devices @7e4f0000 (indirect, esz 8, psz 64K, shr 0)
[    0.000000] ITS@0x00000000fd440000: allocated 32768 Interrupt Collections @7e500000 (flat, esz 2, psz 64K, shr 0)
[    0.000000] ITS: using cache flushing for cmd queue
[    0.000000] GIC: using LPI property table @0x000000007e510000
```

Figure 2.7.3.5 System startup log information

After the system starts successfully, users can execute commands through the serial port terminal, and the execution results of the commands will also be displayed on the serial port terminal.

2. SSH Connection

Next, we will introduce how to establish an SSH remote connection. Through SSH, remote login can be achieved, such as remote login to the Ubuntu system or the Linux system of the development board. According to the operation steps shown in Figures 2.7.3.5 to 2.7.3.6, establish the SSH remote connection and perform the remote login operation (taking the remote login to the Ubuntu system as an example):

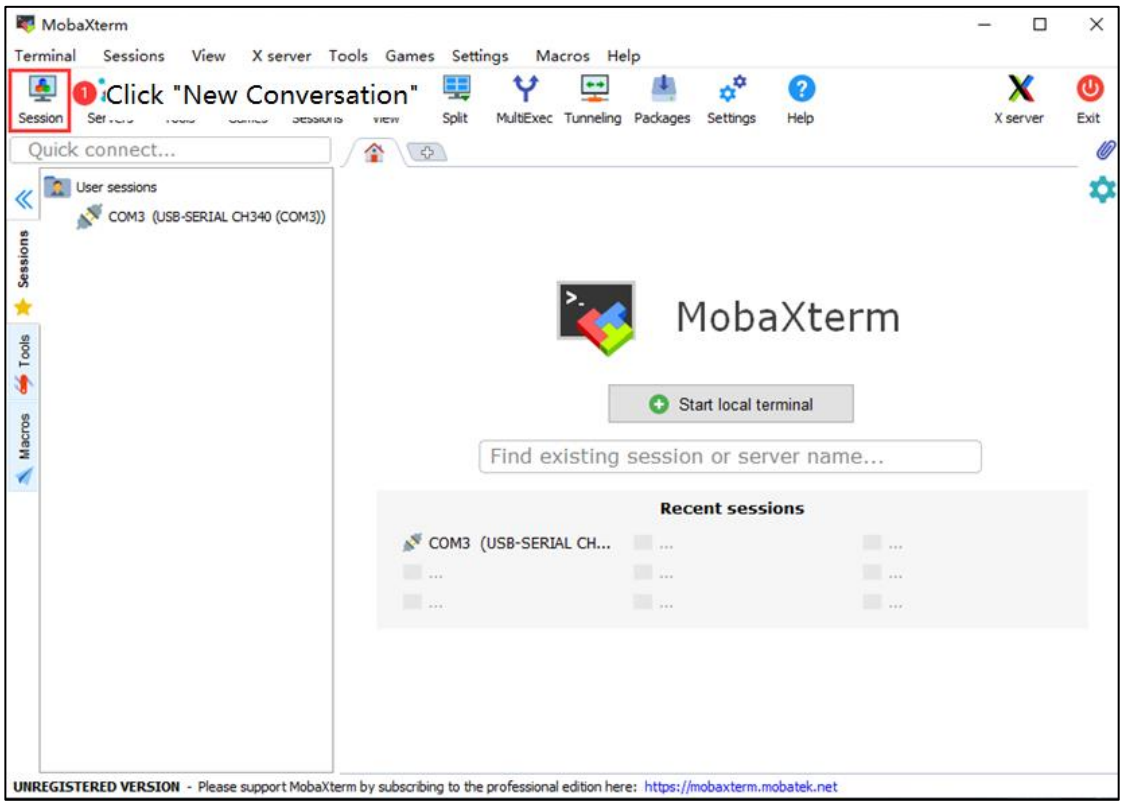


Figure 2.7.3.6 Establish SSH remote connection (1)

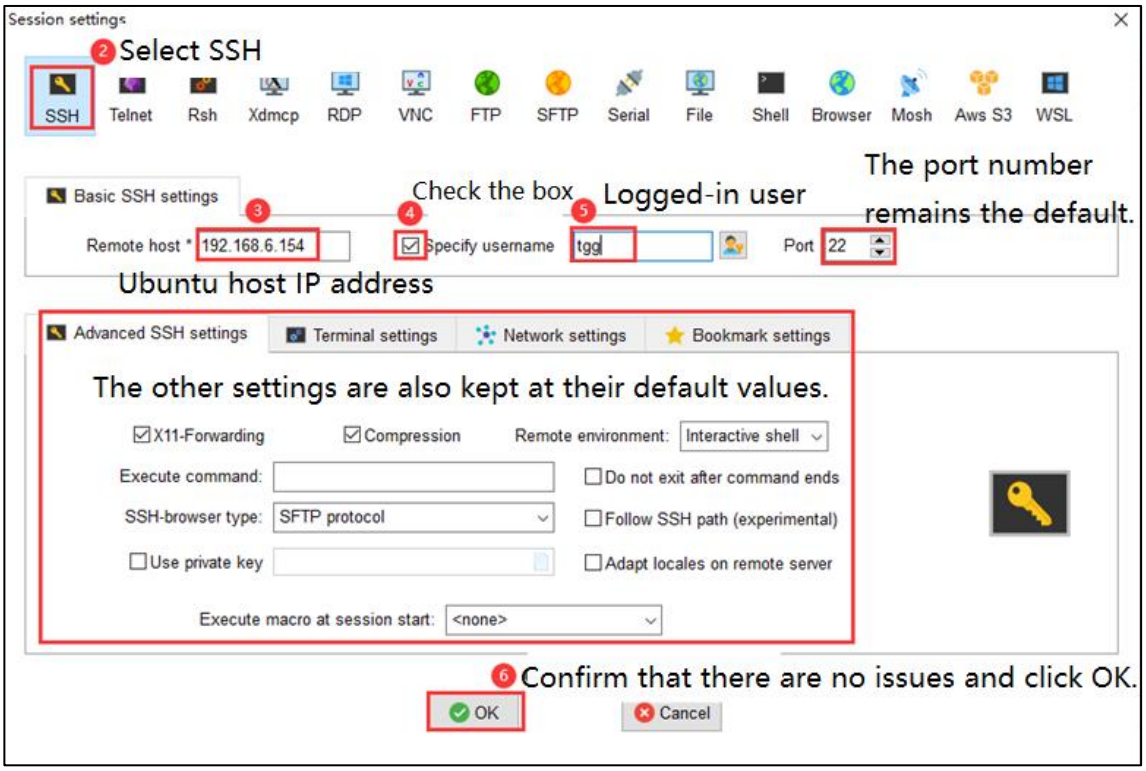


Figure 2.7.3.7 Establish SSH remote connection (2)

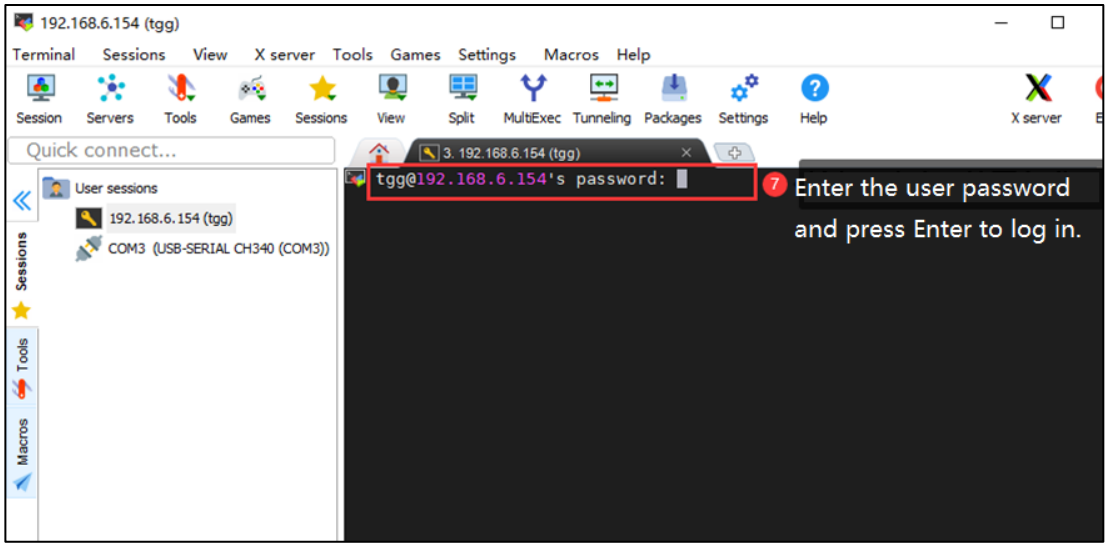


Figure 2.7.3.8 Establishing an SSH remote connection (3)

Figure 2.7.3.9 Establishing an SSH remote connection (4)

A pop-up window asks the user if they need to save the password. We can choose "**Yes**" to save the password or "**No**" to not save it, depending on personal preference. At this point, the SSH remote connection has been successfully created, as shown in Figure 2.7.3.9:

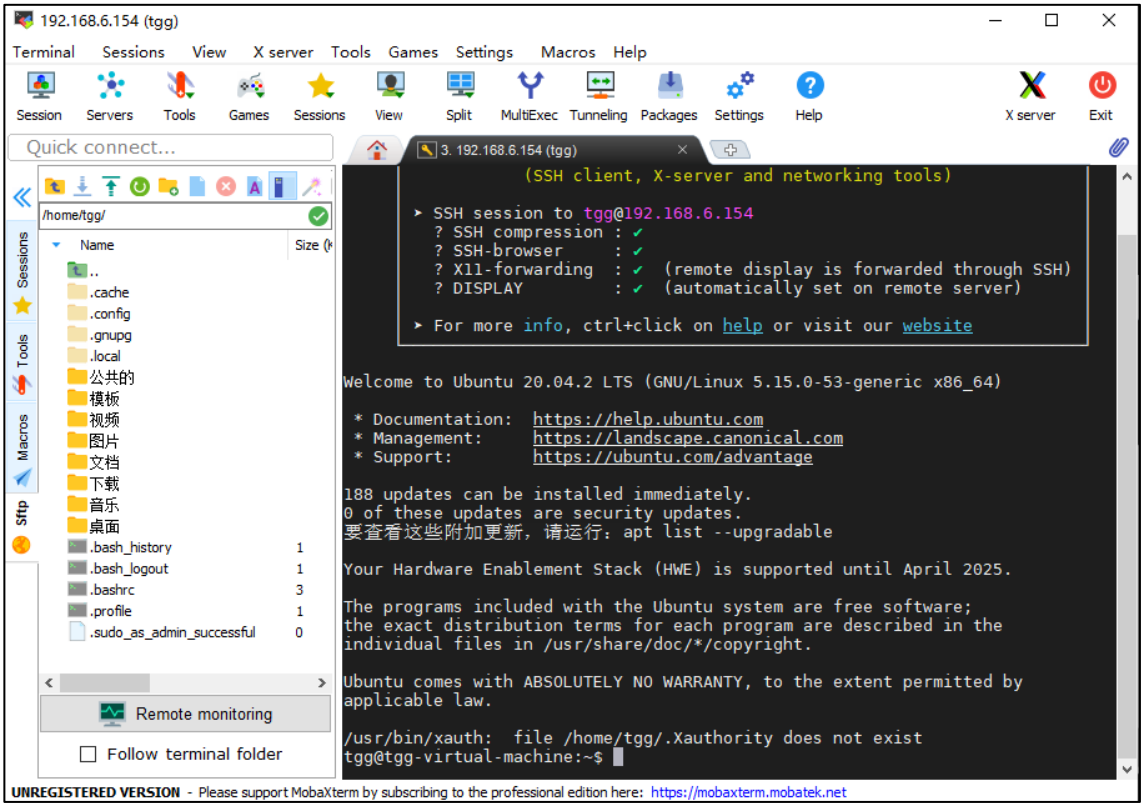


Figure 2.7.3.10 SSH remote login was successful.

When conducting SSH remote login, we can also use the MobaXterm software to achieve file transfer between Windows and Ubuntu (using the SFTP protocol for file transmission). The specific method is shown in Figure 2.7.3.10:

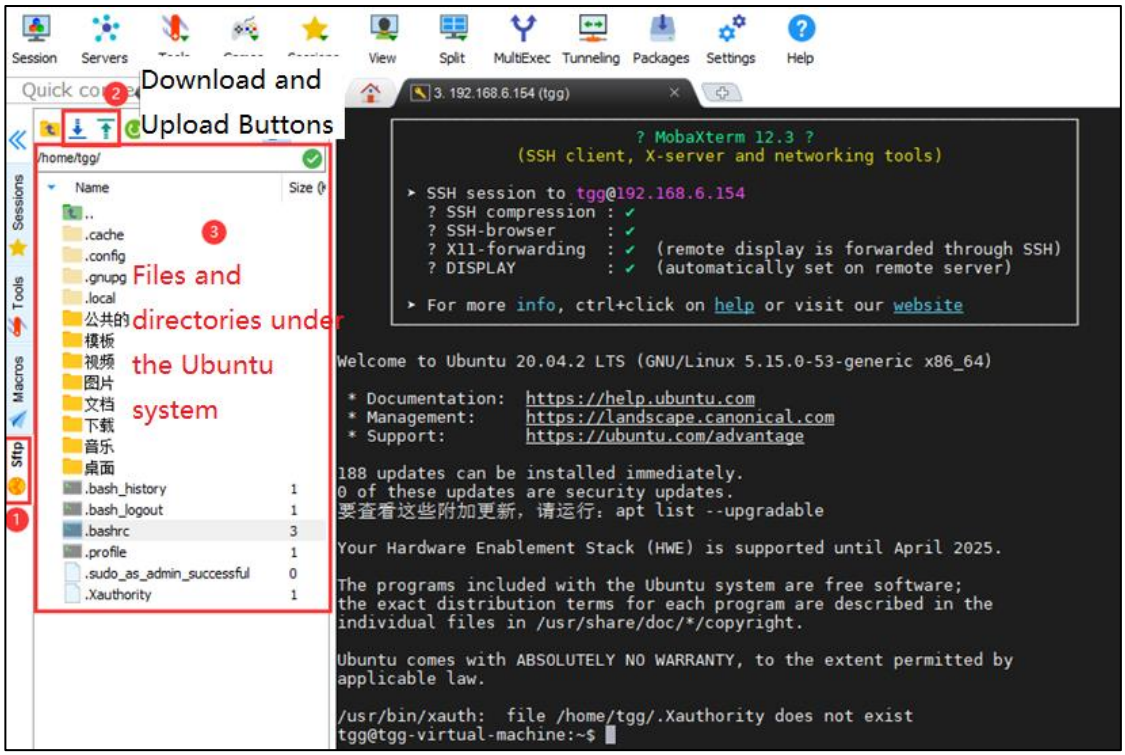


Figure 2.7.3.11 MobaXterm enables file transfer between Windows and Ubuntu

First, select the ①" **Sftp**" tab. By clicking the "**Download** " button, files on the Ubuntu system can be copied to the Windows system, and by clicking the "**Upload** " button, files on the Windows system can be copied to the Ubuntu system. You can also directly drag and drop using the left mouse button. The area shown in ③ represents the files and directories on the Ubuntu system. You can directly drag files from the Ubuntu system to the Windows system using the left mouse button, or drag files from the Windows system to the Ubuntu system.

2.8 Installation of Rockchip USB Driver

During the development and debugging phase, it is often necessary to switch the device to the Loader mode or Maskrom mode. For example, when using the RKDevTool developed by Rockchip to burn the image (requiring a USB connection), the device needs to operate in the Loader mode or Maskrom mode, and only then can the burning process be carried out; in order to be able to recognize the device, the Rockchip USB driver needs to be installed under Windows.

The development board package has already provided the Rockchip USB driver installation package for users. The path is: Development Board CD-ROM A Drive - Basic Materials→5_tools→RKTools→Windows→DriverAssitant_v5.12.zip. It supports operating systems such as xp, win7_32, win7_64, win10_32, and win10_64; Decompress the DriverAssitant_v5.12.zip compressed file, and after decompression, you will get the file and directory list as shown in Figure 2.8.1:

名称	修改日期	类型	大小
ADBDriver	2020/11/10 14:13	文件夹	
bin	2020/11/10 14:14	文件夹	
Driver	2022/2/28 14:14	文件夹	
config.ini	2014/6/3 15:38	配置设置	1 KB
DriverInstall.exe	2022/2/28 14:11	应用程序	491 KB
Readme.txt	2018/1/31 17:44	文本文档	1 KB
revison.log	2022/2/28 14:14	文本文档	1 KB

Figure 2.8-1 Unzip DriverAssitant_v5.12.zip

Simply double-click the DriverInstall.exe file and follow the steps shown in Figures 2.8.2 to 2.8.4 to install the Rockchip USB driver:

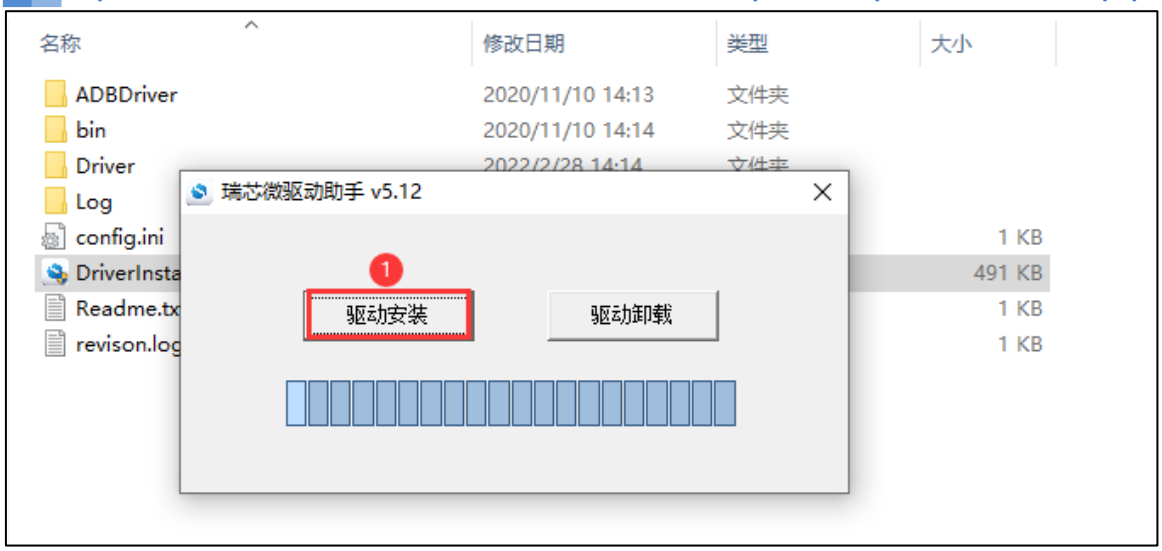


Figure 2.8-2 Install Rockchip USB driver (1)

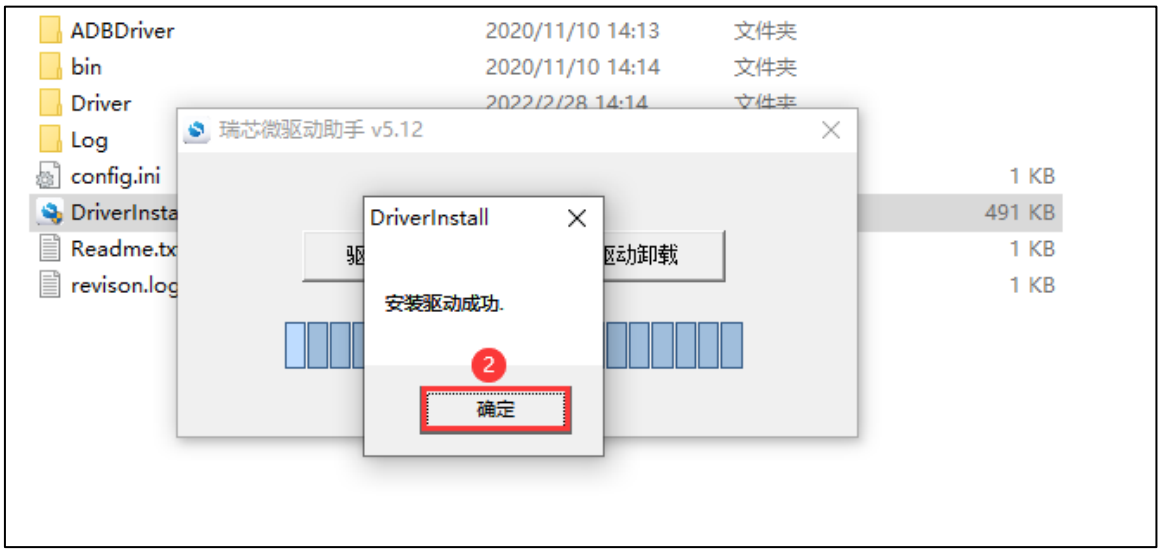


Figure 2.8-3 Install Rockchip USB driver (2)

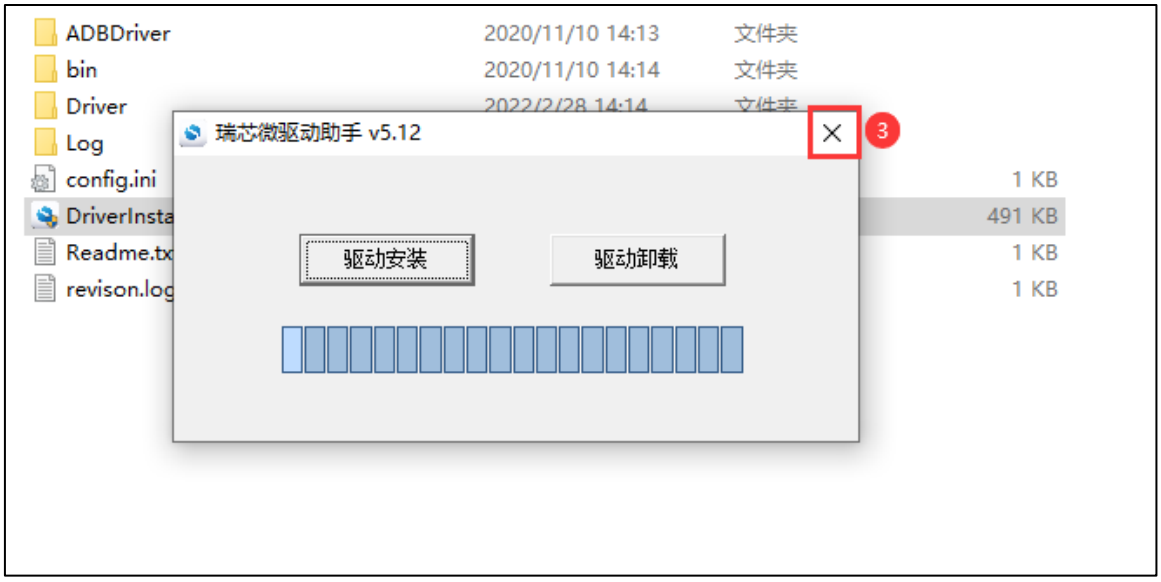


Figure 2.8-4 Install Rockchip USB driver (3)

2.9 Usage of Mirror Image Burning Tools

The Rockchip platform offers multiple methods for burning images, such as burning using the RKDevTool (Renee Micro Development Tool) in Windows, through an SD card, using the FactoryTool tool for batch burning (mass production burning tool, supporting USB one-to-many burning), and through the Linux_Upgrade_Tool tool in the Ubuntu system, among others. In summary, there are many ways to burn images, and users can choose the appropriate burning method to burn the image file onto the development board.

This section briefly introduces the burning tool RKDevTool for Windows and the burning tool Linux_Upgrade_Tool for Linux. During our development and debugging process, these two should be the most frequently used burning tools.

2.9.1 Introduction to Burning Modes

The several modes in which Rockchip platform hardware devices operate are as follows in the table. Only when the device is in the Maskrom and Loader mode (it is necessary to connect the OTG port of the development board to the computer via a USB cable), can the image be burned or the image on the board be updated.

Mode	Do you support burning	Description
Maskrom	Support	When Flash is not burned with an image, the chip will enter the Maskrom mode, allowing for the initial burning of the image. During the development and debugging process, if it is impossible to enter the Loader mode normally, the Maskrom mode can also be used to burn the image.
Loader	Support	In Loader mode, image burning, update and upgrade can be performed. It is also possible to burn a specific partition image file separately using the burning tool, which is convenient for debugging.

Recovery	Not supported	The system initiates the "recovery" mode, which mainly serves for operations such as upgrading and restoring factory settings.
Normal Boot	Not supported	The system starts normally and enters the mode after startup. It is initiated by booting the rootfs and loads the rootfs. Most of the development is carried out in this mode for debugging.

Table 2.9.1.1 Rockchip Platform Operating Mode Explanation

The method to enter the Maskrom programming mode: (First, connect the power adapter and the OTG)

- If the development board has not burned the image, it will enter the Maskrom mode after power-on;
- If the development board has burned the image, press the **UPDATE button** on the development board, then power-on or reset the development board, and the system will enter the Maskrom mode;
- If the development board has burned the image, press the **V- button** (volume-) on the development board, then power-on or reset the development board, and the system will enter the Maskrom mode;
- If the development board has burned the image, execute the "**rbrom**" command in the U-Boot command line to enter the Maskrom mode.

The method to enter the Loader burning mode: (First connect the power adapter and OTG)

- After the development board has burned the image, press the **V+ button** (volume+) and then power on or reset the development board. The system will enter the Loader mode;
- After the development board has burned the image, execute the "**download**" command in the U-Boot command line to enter the Loader mode;
- After the development board has burned the image, in the Android system, execute the command "**reboot loader**" via serial port or ADB to restart and enter the Loader mode;
- After the development board has burned the image, after normal startup of the development board after power-on or reset, the Rockchip development tool (also known as RKDevTool tool) will display "Found an ADB device", then click the "**Switch**" button to enter the Loader mode, as shown in the following figure.

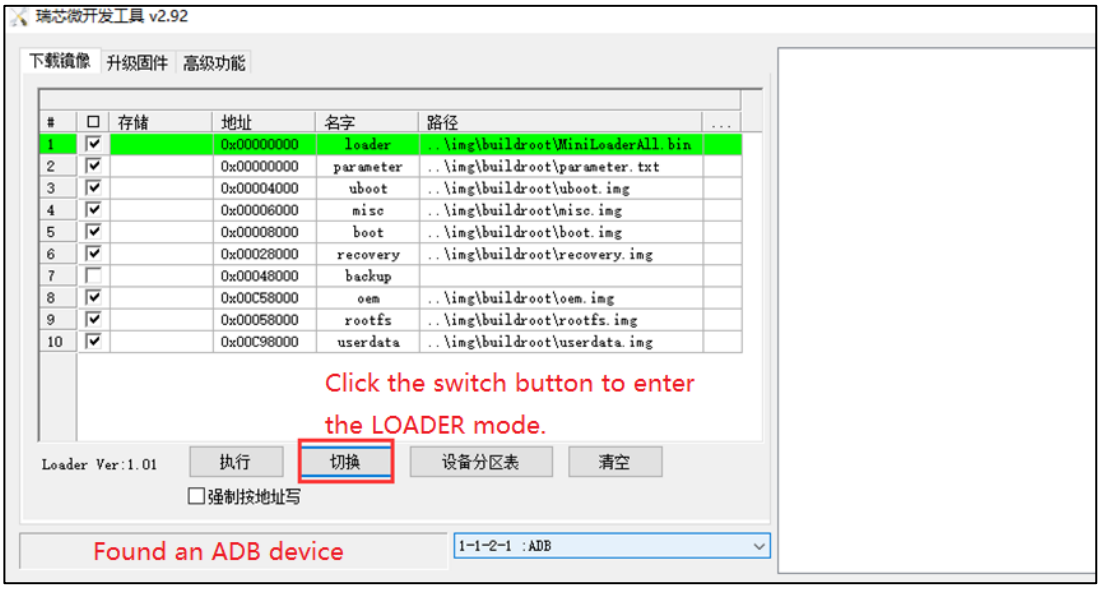


Figure 2.9.1.1 Enter the Loader mode through the development tools of Rockchip Microelectronics.

The above has introduced to you how the development board can enter the Maskrom and Loader two types of burning modes. Only after entering the Maskrom mode or the Loader mode can the burning process be carried out.

2.9.2 Windows flashing tool RKDevTool

To flash the image under Windows, use the RKDevTool tool. In the documentation of RK, it is generally referred to as the Rockchip Development Tool. This tool is not only used for flashing the image, but also has other functions, and it is the most commonly used tool in our development and debugging process.

The ATK-DLRK3568 development board data package from ALIENTEK has already provided this tool to users. The path is: Development board optical disc A disk - Basic materials→5_tools→RKTools→Windows→RKDevTool_Release_v2.92.zip. Of course, this tool can also be found in the RK3568 Android SDK.

First, decompress the RKDevTool_Release_v2.92.zip compressed file. The content after decompression is as follows:

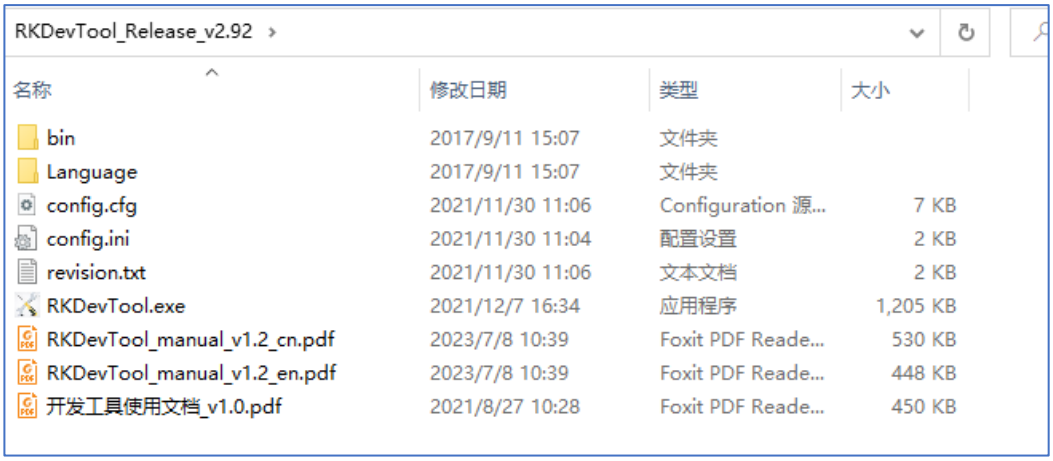


Figure 2.9.2.1 RKDevTool_Release_v2.92 directory

There are two official documents provided by Rockchip Microelectronics in this directory: "Developer Tools Usage Document_v1.0.pdf" and "RKDevTool_manual_v1.2_xx.pdf". These documents provide users with detailed instructions on how to use the RKDevTool tool (Rockchip Microelectronics development tool). You can refer to these two documents for the detailed usage methods of the tool. We are just giving a brief introduction here.

Double-click the RKDevTool.exe executable file to open the Rockchip Microelectronics development tool, as shown in the following figure:

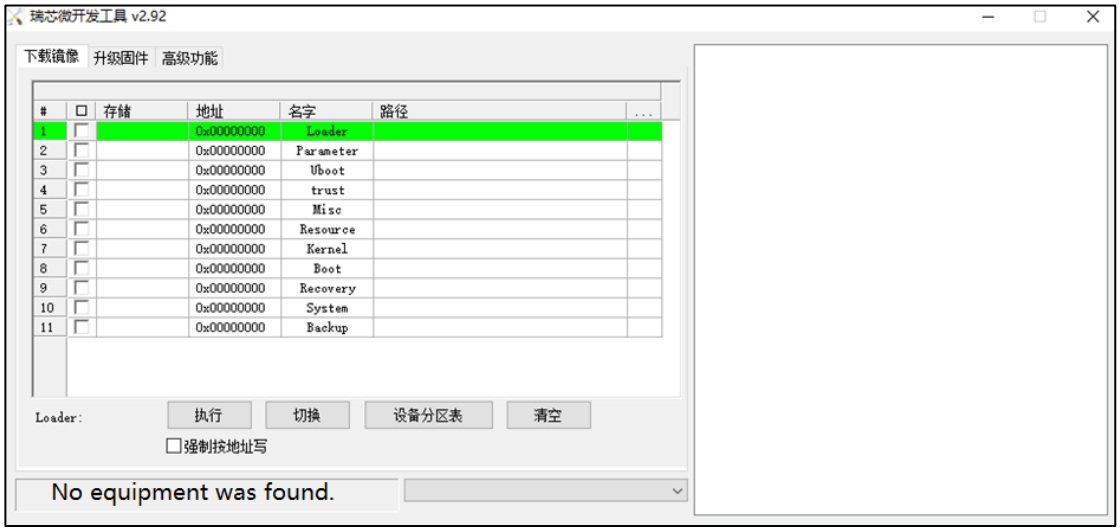


Figure 2.9.2.4 The interface of the Renesas Micro development tool

As shown in the above picture, this is a main interface of the Renesas Micro development tool. The RKDevTool tool has two methods for image burning:

- **Burn each individual image separately.** After the successful compilation of the RK3568 Android11 SDK, multiple image files will be generated, including boot.img, baseparameter.img, MiniLoaderAll.bin, misc.img, dtbo.img, recovery.img, super.img, uboot.img, vbmeta.img, totaling 9 images, among which there is also a partition table file parameter.txt. Using the RKDevTool tool, these images can be burned separately.
- Burn the complete update.img firmware. The update.img is not compiled; it is actually a collection of multiple images, including uboot.img, boot.img, dtbo.img, misc.img, vbmeta.img, recovery.img, super.img, and MiniLoaderAll.bin, etc. These individual images are combined to form the update.img; using the tools provided by RK, each individual image can be packaged into one update.img.

Next, let's talk about how to perform the burning in the form of separate mirrors. In Section 2.9.3, we will introduce to users how to burn the update.img firmware.

Follow the operation steps shown in Figures 2.9.2.5 to 2.9.2.8, and import a.cfg configuration file (the configuration file contains the configuration information during the burning process, including the burning address, the path of the image, etc. Of course, you can also manually configure these information, and we will explain it later):

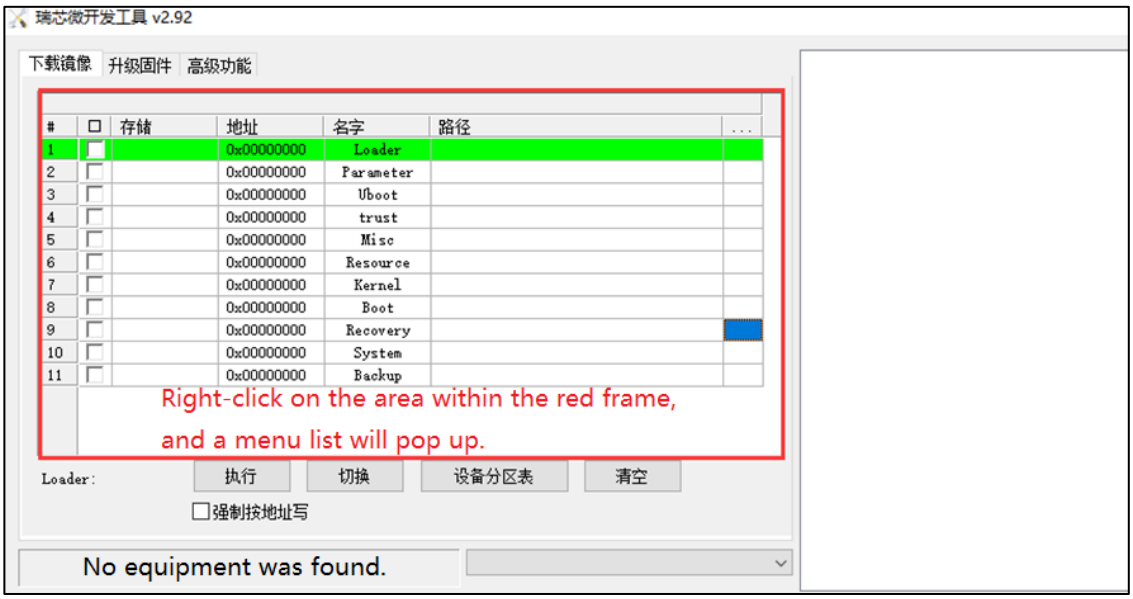


Figure 2.9.2.5 Import configuration file (1)

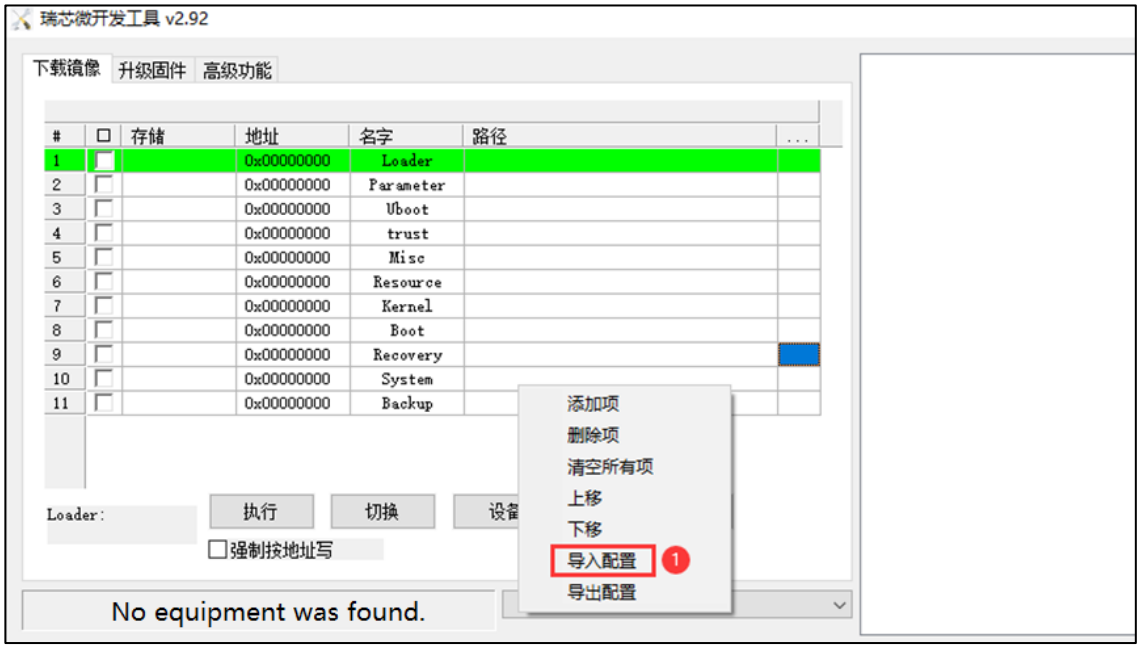


Figure 2.9.2.6 Import configuration file (2)

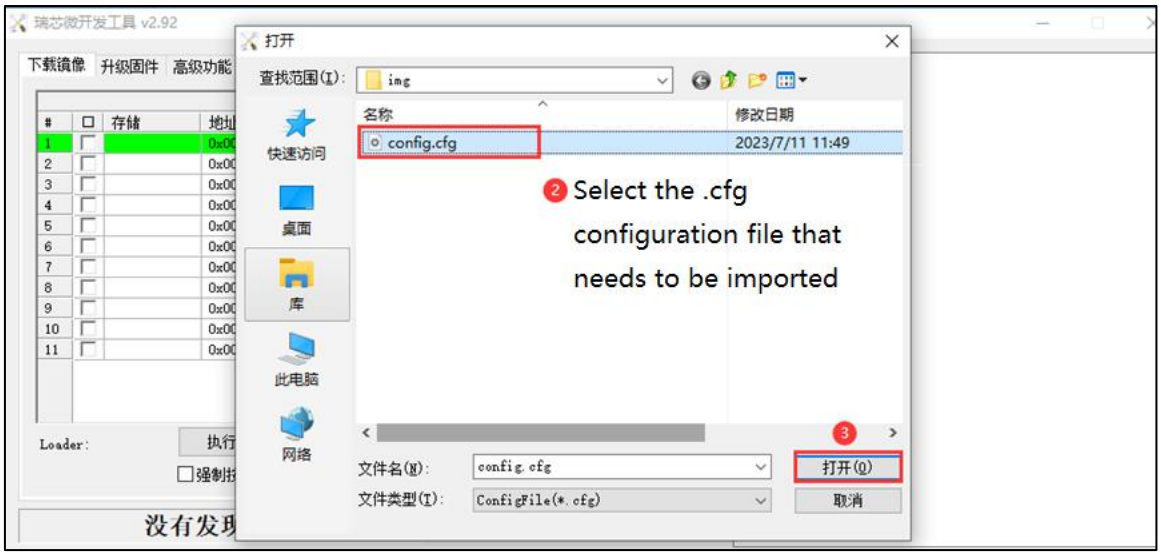


Figure 2.9.2.7 Import configuration file (3)

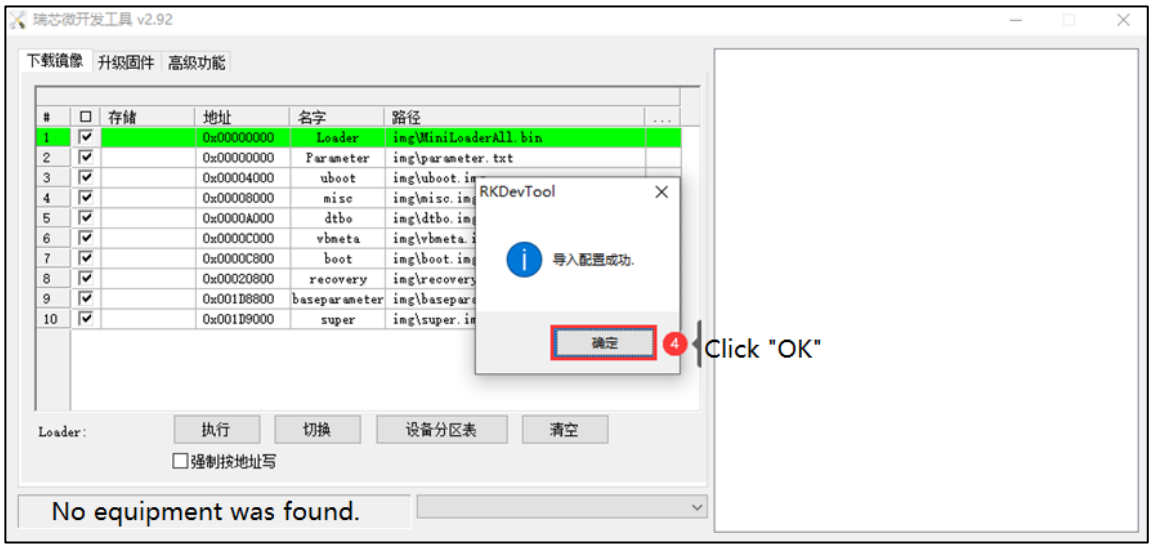


Figure 2.9.2.8 Import configuration file (4)

At this point, we have imported a burn-in configuration file:

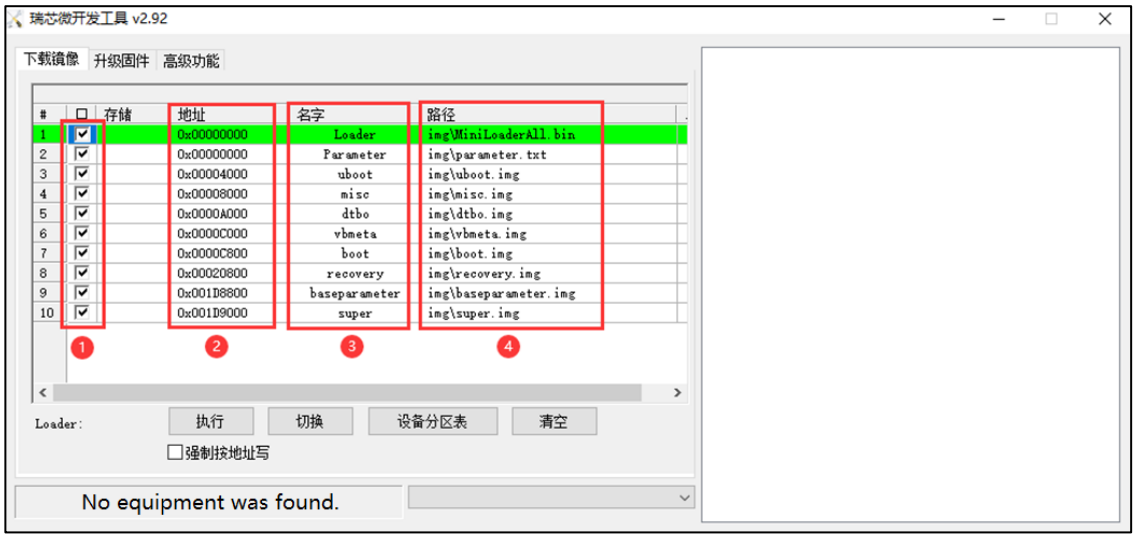


Figure 2.9.2.9 Configuration Information Division Explanation

① Some are used to control whether to burn the corresponding image. Checking indicates that it needs to be burned, and unchecking indicates that it does not need to be burned;

② Some represent the address for burning (that is, the burning address of the image);

③ Some represent the name of this burning item;

④ Some are used to specify the path where the image is located.

These configuration information can all be manually modified. After modification, they can be re-exported to the configuration file for saving, which is convenient for importing this configuration next time.

After importing the configuration, you can then proceed with the burning process. Before performing the burning operation, the development board must be in Maskrom mode or Loader mode. The development board that customers receive is usually already burned with the image. You can enter Loader mode, or you can enter Maskrom mode. First, the development board needs to be connected to the power adapter and the OTG port as shown in the following figure:

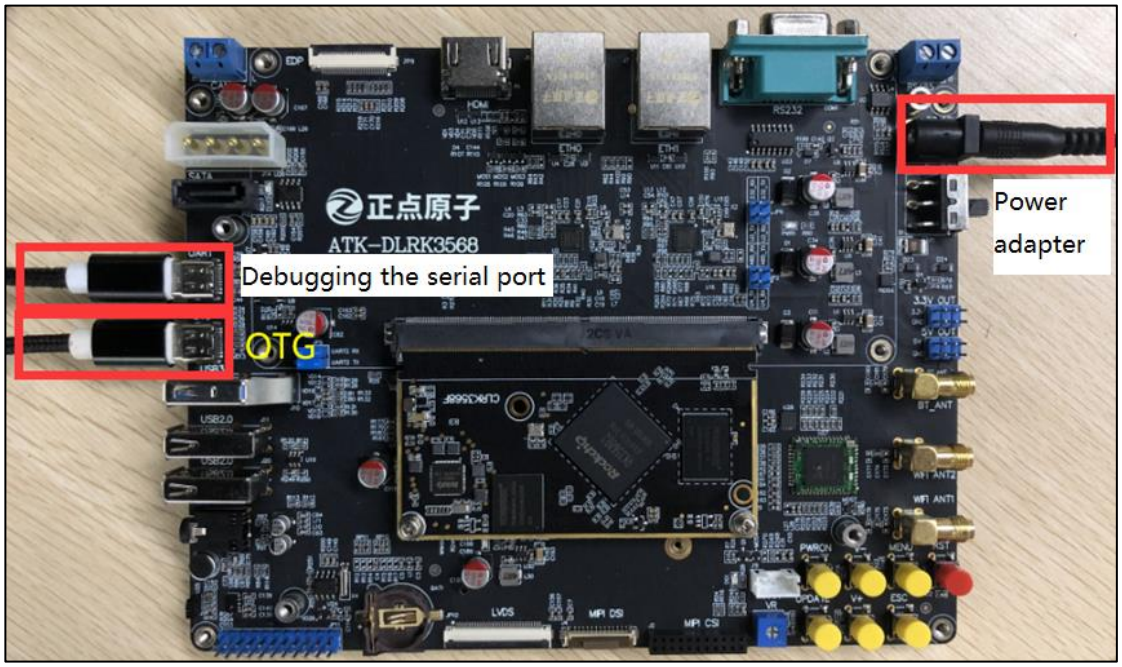


Figure 2.9.2.10 Hardware connection diagram of the development board

The power adapter uses the 12V-2.5A power adapter provided by the ATK-DLRK3568 development board; connect the debugging serial port UART of the development board to the computer via a USB cable (install the CH340 driver first), and also connect the OTG interface of the development board to the computer using a USB cable.

After the hardware connection is completed, press the **V+ button** (volume+) on the development board, then power on or reset the development board. The system will enter the Loader mode; at this time, the Rockchip development tools will display "**Found a LOADER device**":

Figure 2.9.2.11 The development tools of Rockchip Microelectronics detected the loader device.

This indicates that our development board is currently in Loader mode. Simply click the "**Execute**" button to burn the image, as shown below:

Figure 2.9.2.12 Perform the burning operation

Of course, it is also possible to perform the burning operation in Maskrom mode. In Section 2.9.1, we have already introduced how to enter the Maskrom mode. After entering the Maskrom mode, the burning operation is carried out by clicking the "**Execute**" button as well.

If there are no unexpected issues, the burning operation will be successful, which means that these images have been successfully burned into the Flash memory of the development board (the ATK-DLRK3568 development board uses eMMC as the onboard memory, so by default, the images will be burned into the eMMC of the development board, and then the development board can be started through eMMC; of course, the images can also be burned onto an SD card and the development board can be started through the SD card). If unfortunately, an error occurs during the burning process, causing the burning operation to fail, then you can refer to the official document of RK provided by RK, "**RKDevTool_manual_v1.2_cn.pdf**", which contains explanations for several common errors.

2.9.3 Method for burning update.img image

update.img is a collection of multiple images (formed by packaging and merging multiple images). Using the tools provided by RK, the individual separate images (uboot.img, boot.img, MiniLoaderAll.bin, parameter.txt, misc.img, super.img, dtbo.img, vbmeta.img, recovery.img, etc.) can be packaged into an update.img firmware, which is convenient for users to burn and upgrade.

This section introduces how to use the RK development tools to burn update.img, first open the RK development tools and select "**Upgrade Firmware**":

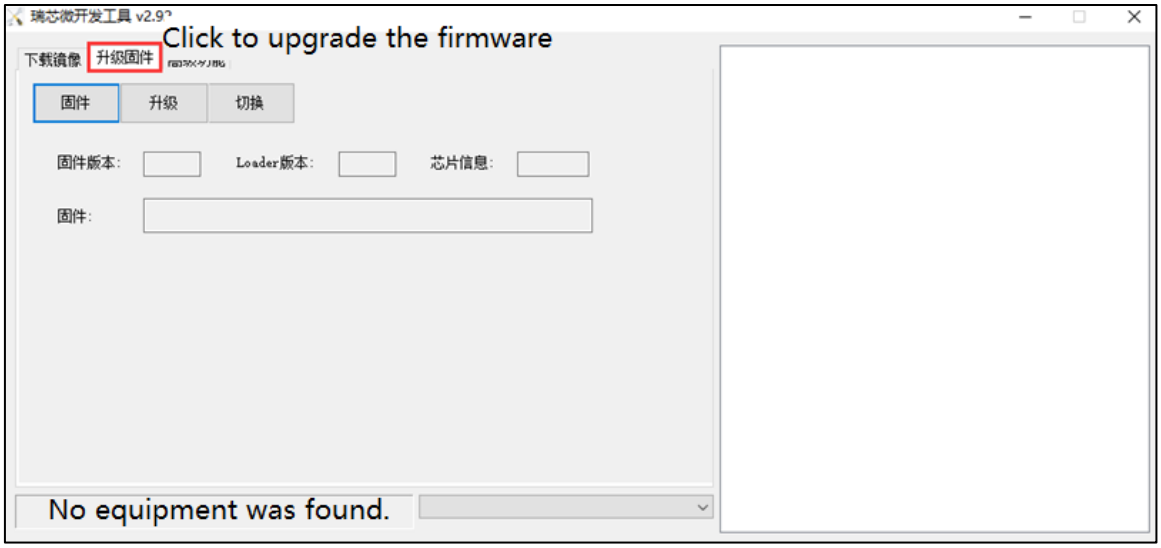


Figure 2.9.3.1 Upgrade firmware option

Then click the "**Firmware**" button to select the update.img firmware that we need to upgrade or burn (the development board materials do not provide users with the update.img image. This is just to demonstrate the method of burning the update.img image to users. Users can practice the operation in their subsequent development process). After loading the image, some information about the update.img image will be displayed:

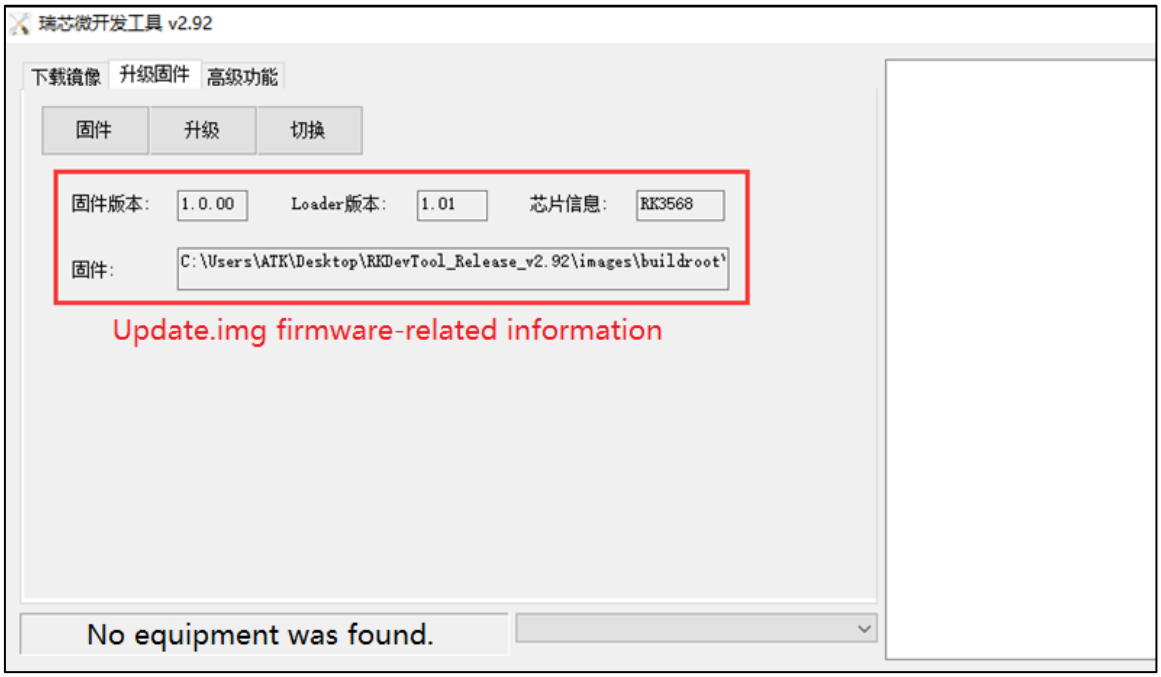


Figure 2.9.3.2 Import the update.img image

First, make the device enter the Maskrom or Loader mode, then click the "Upgrade" button to perform firmware upgrade and update:

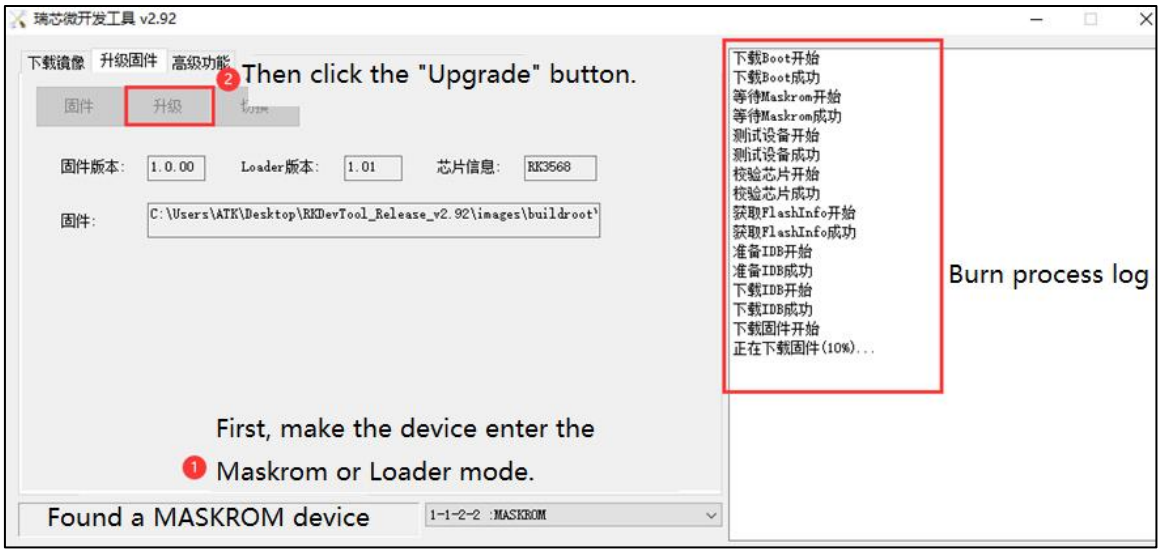


Figure 2.9.3.3 update.img upgrade

2.9.4 Erase operation

The Flash can be erased using the development tools of Rockchip. This operation involves erasing the image that has been burned into the Flash. Open the Rockchip development tools and click on the "**Advanced Functions**" tab, as shown below:

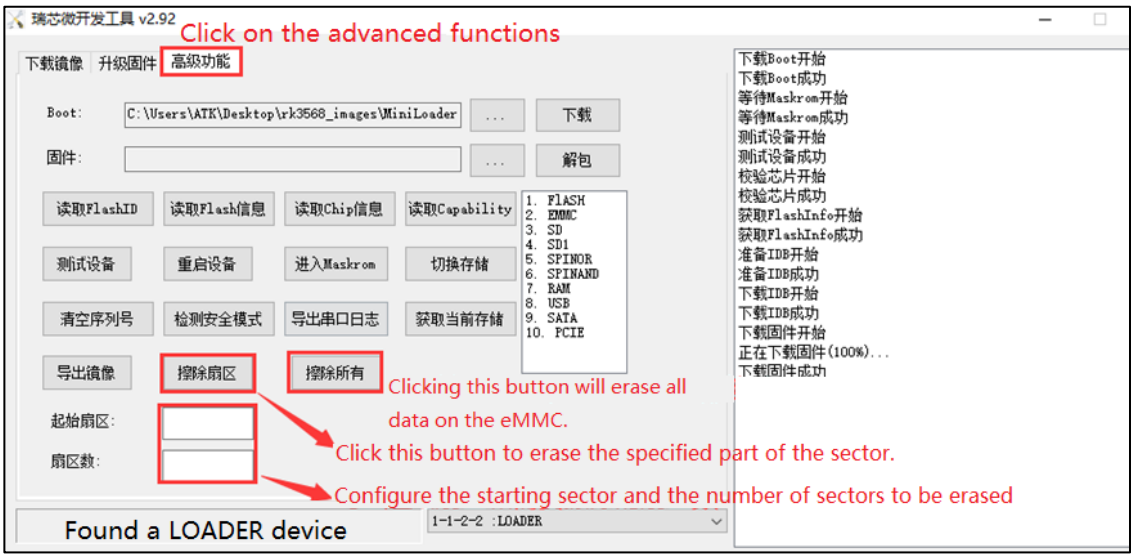


Figure 2.9.4.1 Advanced Function Page

It should be noted that the erasure operation needs to be performed in Loader mode (Maskrom mode can also perform the erasure operation, but it is necessary to first download MiniLoaderAll.bin. First, select the MiniLoaderAll.bin image, then click the "Download" button in the above picture to perform the download operation. For details, please refer to the <Development Tools User Guide_v1.0.pdf> document). Clicking the "**Erase All**" button can erase all the data in the eMMC of the development board; clicking the "**Erase Sector**" button can erase the specified part of the sectors, fill the starting sector of the erasure into the "**Starting Sector**" input box, and fill the total number of sectors to be erased into the "**Sector Number**" input box.

For example, clicking the "**Erase All**" button to erase all the data in the eMMC of the development board:

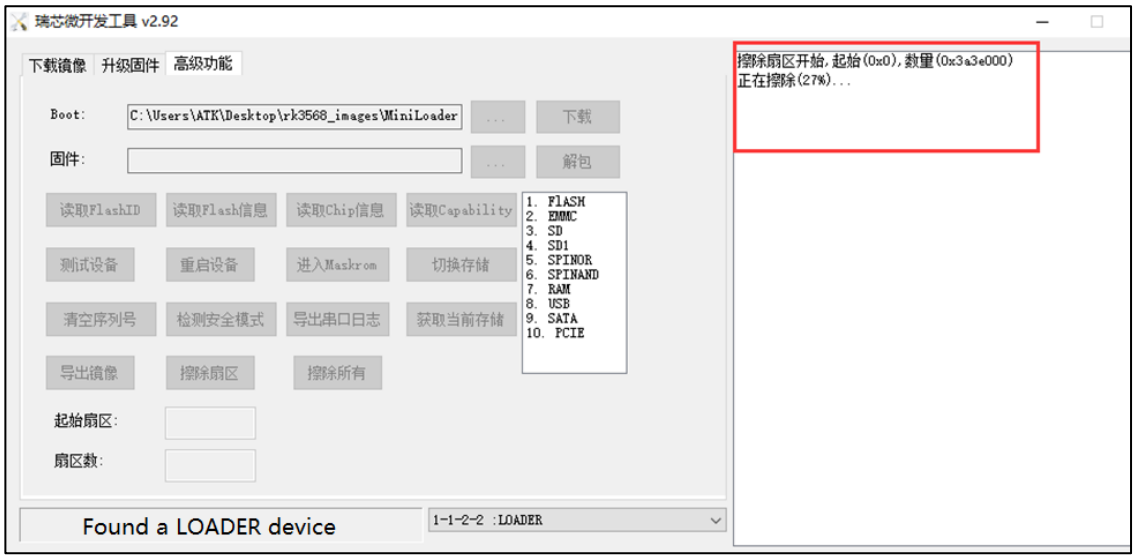


Figure 2.9.4.2 Erasure of all operations

2.9.5 Usage of the Linux_Upgrade_Tool tool under Ubuntu

During the development and debugging phases, we usually compile the source code under Ubuntu, and the compiled images are also located in the Ubuntu system directory. Before burning the images using the RKDevTool development tool, it is necessary to first copy these image files to the Windows system, and then perform the burning process. Therefore, it is indeed somewhat troublesome.

In addition to being able to burn the images under Windows, we can also burn the images under Ubuntu. This avoids the process of copying the images. Under the Ubuntu system, the Linux_Upgrade_Tool tool provided by RK can be used for burning.

This tool is included in the SDK, the ATK-DLRK3568 development board SDK. The burning tool is provided by default in the fifth chapter for everyone to be introduced.

2.10 Installation of ADB tool

ADB stands for Android Debug Bridge, which literally means Android Debug Bridge. It is a universal command-line tool. As a bridge between Android devices and the PC side, ADB enables developers to communicate with Android devices via network or USB, thus serving as a debugging tool for Android devices. In Android development, ADB is an indispensable tool.

Users can perform a series of operations on the Android emulator or real Android device on the computer using ADB, such as installing, uninstalling and debugging applications (apk), copying and pushing files (copying files between the computer and the Android device), viewing device information, capturing logs, etc. Its main functions include:

- Running the device's shell (command line);
- Managing the port mapping of the emulator or device;
- File upload/download between the computer and the device;
- Installing local apk software on the emulator or Android device.

Regarding the installation and usage methods of the ADB tool, this document does not provide any introduction. Users can refer to another document provided by SparkFun, titled "adb Tool Usage Instructions.pdf", located at: Development Board CD-ROM A Disk - Basic Materials -> 10_user_manual -> 03, Supplementary Documents -> [ALIENTEK] adb Tool Usage Instructions.pdf. In this document, users will be instructed on how to install the adb tool on Windows systems and Ubuntu systems, and also how to use the adb tool.

Chapter 3. Introduction to the ATK-DLRK3568 Platform

Reference Document:

"ATK-DLRK3568 Hardware Reference Manual V1.1"

3.1 Introduction to RK3568

Reference Document:

"ATK-DLRK3568 Hardware Reference Manual V1.1" Chapter one

3.2 Introduction to the Hardware Resources of the ATK-DLRK3568 Development Board

Reference Document:

"ATK-DLRK3568 Hardware Reference Manual V1.1" Chapter two

Chapter 4. RK3568 Android SDK software package

This chapter introduces to users the installation and usage methods of the Android SDK software package for the ALIENTEK ATK-RK3568 hardware platform. This SDK is applicable to the ALIENTEK ATK-RK3568 development board and all Android system products developed based on this board. Based on this SDK, system customization and application porting development can be effectively achieved, helping users to quickly develop and improve their development efficiency!

The RK3568 Android SDK integrates Android source code, Linux Kernel source code, U-Boot source code, and various development tools provided by RK. The Linux kernel version is 4.19 and the U-Boot version is 2017.09.

This chapter will be divided into the following sections:

4.1 Installation of RK3568 Android SDK

4.2 Introduction to the SDK software architecture

4.3 Full automatic compilation of the SDK

4.4 Individual compilation

4.5 Cleaning of the SDK

4.6 Introduction to the image

The content of this chapter and the subsequent chapters of this document is based on the official documentation provided by RK:

<SDK>/RKDocs/android/Rockchip_Developer_Guide_Android11_SDK_V1.1.6_CN.pdf

Or in the development board data package:

Development board CD A disc - Basic materials -> 08_RK_official_documents -> Android -> android -> Rock-chip_Developer_Guide_Android11_SDK_V1.1.6_CN.pdf

It is highly recommended that everyone take a look!

4.1 Installing the RK3568 Android SDK

This section provides instructions for users on how to install the Android SDK package for the ALIENTEK ATK-RK3568 development board on the Ubuntu system.

4.1.1 Install dependent software packages

First, you need to install the software packages required by the SDK compilation environment on the Ubuntu system. Execute the following command to install the software packages:

```
sudo apt-get update
sudo apt-get install git-core gnupg flex bison gperf build-essential zip curl zlib1g-dev gcc-multilib \
g++-multilib libc6-dev-i386 lib32ncurses5-dev libx11-dev lib32z-dev ccache libgl1-mesa-dev \
libxml2-utils xsltproc unzip python-pyelftools python3-pyelftools device-tree-compiler libfdt-dev \
libfdt1 libssl-dev liblz4-tool python-dev libncurses5 make
```

During the installation process, ensure that the Ubuntu system has a normal network connection. The installation process will take some time. Please be patient and wait!

Set Python 2 as the default Python version for the system:

```
sudo rm -rf /usr/bin/python
sudo ln -s /usr/bin/python2 /usr/bin/python
```

4.1.2 Install repo (skip)

Repo is a multi-repository management tool built on Git, which can organize the upload and download of multiple repositories and is used for managing multiple Git storage repositories. The code and related documentation of the RK3568 Android SDK have been divided into several git repositories for version management (the SDK contains several git repositories), and developers can use the repo tool to perform unified operations such as downloading, committing, and switching branches on these git repositories.

Installing the SDK requires the use of the repo tool, so it is necessary to install repo first.

First, create a bin folder in the user's home directory and export it to the PATH environment variable:

```
mkdir ~/bin
export PATH=~/bin:$PATH
```

If you have access to Google, you can download the repo using the following command and grant it execution permissions:

```
curl https://storage.googleapis.com/git-repo-downloads/repo > ~/bin/repo
chmod a+x ~/bin/repo
```

In the domestic network environment, if after executing the above command you find that the ~/bin/repo file is empty, at this point you can access the domestic website to download the repo tool and grant it execution permissions:

```
curl https://mirrors.tuna.tsinghua.edu.cn/git/git-repo -o ~/bin/repo
chmod a+x ~/bin/repo
```

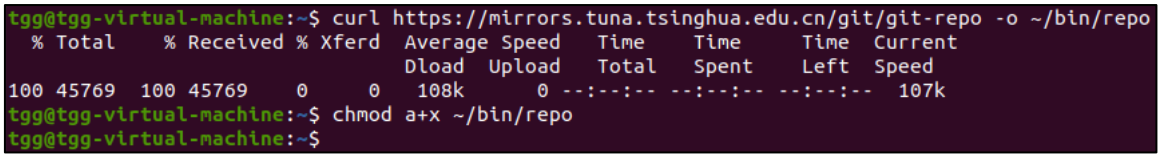


Figure 4.1.2.1 Download the repo tool

The repo tool is actually just a Python script.

4.1.3 Git configuration

Before using repo, users need to configure their own git information; otherwise, there may be obstacles when performing subsequent operations due to hook checks:

```
git config --global user.name "your name"
git config --global user.email "your email"
```

Please configure according to the actual situation. You can view the git configuration information by using the following command:

```
git config --list
```

```
tgg@tgg-virtual-machine:~$
tgg@tgg-virtual-machine:~$ git config --list
user.name=[illegible]
user.email=[illegible]om
tgg@tgg-virtual-machine:~$
```

Figure 4.1.3.1 View git configuration information

4.1.4 Install SDK

Now let's start installing the RK3568 Android SDK. The development board package has already provided the SDK compressed file for users. The path is: Development board CD B drive – Development environment and SDK→02_ATK-DLRK3568_development_board_SDK→atk-rk3568_androidR_release_v1.0_20230623.tgz. With the update of the version, the name of the SDK compressed file will also change, but it will always be named as atk-rk3568_androidR_release_ version _ release date.tgz. When a new version of the SDK is released, the old version of the SDK in the network drive will be replaced. If users need to obtain the old version of the SDK, please contact the ALIENTEK Linux technical support for it!

Copy the tgz compressed file to the home directory of the Ubuntu system, and execute the following command to decompress it to the rk3568_android_sdk directory in the home directory:

```
mkdir ~/rk3568_android_sdk
tar xf atk-rk3568_androidR_release_v1.0_20230623.tgz -C ~/rk3568_android_sdk
```

```
tgg@tgg-virtual-machine:~$
tgg@tgg-virtual-machine:~$ ls
公共的  模板  视频  图片  文档  下载  音乐  桌面  atk-rk3568_androidR_release_v1.0_20230623.tgz  bin  to
tgg@tgg-virtual-machine:~$
tgg@tgg-virtual-machine:~$ mkdir ~/rk3568_android_sdk
tgg@tgg-virtual-machine:~$ tar xf atk-rk3568_androidR_release_v1.0_20230623.tgz -C ~/rk3568_android_sdk
tgg@tgg-virtual-machine:~$
```

Figure 4.1.4.1 Unzipping the SDK package

The unzipping process will take a long time. Please be patient! After the unzipping is complete, a .repo folder will exist in the ~/rk3568_android_sdk/ directory, as shown in Figure 4.1.4.2:

```
tgg@tgg-virtual-machine:~$
tgg@tgg-virtual-machine:~$ ls -lha ~/rk3568_android_sdk/
总用量 12K
drwxrwxr-x  3 tgg tgg 4.0K 4月  22 13:34 .
drwxr-xr-x 22 tgg tgg 4.0K 4月  22 14:30 ..
drwxrwxr-x  7 tgg tgg 4.0K 4月  21 20:00 .repo
tgg@tgg-virtual-machine:~$
```

Figure 4.1.4.2 The .repo folder

To check out the source code, execute the following command:

```
cd ~/rk3568_android_sdk/
.repo/repo/repo sync -l -j16
```

```
tgg@tgg-virtual-machine:~$ cd ~/rk3568_android_sdk/
tgg@tgg-virtual-machine:~/rk3568_android_sdk$
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ .repo/repo/repo sync -l -j16
正在更新文件: 100% (257/257), 完成.
正在更新文件: 100% (539/539), 完成.
正在更新文件: 100% (396/396), 完成.
正在更新文件: 100% (602/602), 完成.
正在更新文件: 100% (1647/1647), 完成.
正在更新文件: 100% (273/273), 完成.
正在更新文件: 100% (119/119), 完成.
正在更新文件: 100% (497/497), 完成.
正在更新文件: 100% (189/189), 完成.
正在更新文件: 100% (192/192), 完成.
正在更新文件: 100% (1007/1007), 完成.
正在更新文件: 100% (171/171), 完成.
正在更新文件: 100% (707/707), 完成.
正在更新文件: 100% (3083/3083), 完成.
正在更新文件: 100% (172/172), 完成.
正在更新文件: 100% (492/492), 完成.
正在更新文件: 100% (73162/73162), 完成.
正在更新文件: 100% (394/394), 完成.onfigs正在更新文件:  49% (355/723)
正在更新文件: 100% (758/758), 完成.
正在更新文件: 100% (723/723), 完成.
正在更新文件:  59% (126/213))
```

```
正在更新文件: 100% (9261/9261), 完成.
正在更新文件: 100% (122/122), 完成.
正在更新文件: 100% (423/423), 完成.generic/opengl-transport正在更新文件:  8% (827/9260)
正在更新文件: 100% (9260/9260), 完成.
正在更新文件: 100% (269/269), 完成.
正在更新文件: 100% (17/17), 完成.
正在更新文件: 100% (5667/5667), 完成.
正在更新文件: 100% (358/358), 完成.
正在更新文件: 100% (7855/7855), 完成.
正在更新文件: 100% (1151/1151), 完成.
正在更新文件: 100% (1898/1898), 完成.
正在更新文件: 100% (2096/2096), 完成.
正在更新文件: 100% (366/366), 完成.
正在更新文件: 100% (33/33), 完成.
正在更新文件: 100% (2075/2075), 完成.
正在更新文件: 100% (43442/43442), 完成.
正在更新文件: 100% (948/948), 完成.
正在更新文件: 100% (34285/34285), 完成.stem/libfmq正在更新文件:  35% (12050/34285)
正在更新文件: 100% (952/952), 完成.
正在更新文件: 100% (683/683), 完成.
正在更新文件: 100% (763/763), 完成.
正在更新文件: 100% (16294/16294), 完成.
正在更新文件: 100% (44081/44081), 完成.
正在更新文件: 100% (601/601), 完成.m/prebuilts/clang/host/linux-x86正在更新文件:  60% (366/601)
正在更新文件: 100% (2036/2036), 完成.
Checking out: 100% (780/780), done in 1h33m32.433s
repo sync has finished successfully.
tgg@tgg-virtual-machine:~/rk3568_android_sdk$
```

Figure 4.1.4.3 Check out the source code

The entire process will take a relatively long time. Please be patient and wait until the synchronization is completed.

After the synchronization is finished, the contents in the ~/rk3568_android_sdk/ directory are as shown in Figure 4.1.4.4:

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk$
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ ls
Android.bp      build           developers   hardware         Makefile        pdk                 RKDocs   test
art             build.sh        development  javaenv.sh       mkcombinedroot  platform_testing    rkst     toolchain
bionic          compatibility   device       kernel           mkimage_ab.sh   prebuilts           RKTools  tools
bootable        cts             external     libcore          mkimage.sh      restore_patches.sh  sdk      u-boot
bootstrap.bash  dalvik          frameworks   libnativehelper  packages        rkbin               system   vendor
tgg@tgg-virtual-machine:~/rk3568_android_sdk$
```

Figure 4.1.4.4 Android SDK project directory

At this point, the RK3568 Android SDK has been installed successfully.

4.1.5 SDK update

The ALIENTEK Technology team will update and upgrade the SDK software and provide it to users in the form of a compressed file. The naming format of the compressed file is **atk-rk3568_androidR_release_version_release_date.tgz**. When a new version of the SDK is released, the old version of the SDK in the data cloud will be replaced. Therefore, the SDK in the cloud is the current latest SDK. To obtain the latest version of the SDK, you can only download it from the ALIENTEK Data Cloud. If you need to obtain an older version of the SDK, please contact the Linux technical support of ALIENTEK for assistance!

Each SDK version update will be recorded in the **<SDK>/.repo/manifests/ATK-RK3568_Android11_SDK_Note.md** document. If you want to know the version update record of the SDK, you can view this document (it is recommended to open it with a Markdown document reader).

4.1.6 SDK Issue Feedback

During the usage and development process, users may discover some issues, software bugs, specific technical problems, technical inquiries, etc. of the SDK. They can contact the ALIENTEK Linux technical support through the QQ group or Taobao and explain the problems. Subsequently, developers will be arranged to handle and track the issues accordingly.

4.2 Introduction to SDK Software Architecture

This section introduces the Android SDK software architecture to users.

4.2.1 Introduction to the Android SDK Project Directory

The directory structure of the RK3568 Android SDK is actually the same as that of the Android source code, including folders such as art, bionic, bootable, build, compatibility, cts, dalvik, external, frameworks, hardware, etc. Besides the native directories of the Android source code, it also includes directories provided by Rockchip, such as rkbin, RKDocs, rkst, RKTools, u-boot, etc.

Each directory or its subdirectories in the SDK corresponds to a git project; because the Android source code and related documents, tools are divided into several git repositories for version management, and they are organized into different directories according to their functions and modules.

Next, we will introduce the source code directory of the RK3568 Android SDK to users:

Top-level directory name	Explanation
art	Android Runtime (Android operating environment) - Starting from Android 5.0， the Java virtual machine has been replaced by ART； unlike the traditional Dalvik virtual machine， it is designed to enhance the smoothness of the Android system.
bionic	The source code of the basic C library, the Android-converted C/C++ library, is more streamlined than the traditional glibc! It is not subject to the GPL restrictions and supports pthread_cancel(), but does not support C++ exceptions and C++ STL template programming.
bootable	It includes a "recovery" directory, which contains the source code of the recovery program (for factory reset or upgrade).

build	The Android compilation system defines various compilation rules and includes general product configurations.
compatibility	Including Android CDD (Android Compatibility Definition Document)
cts	Compatibility Test Suite, namely Android Compatibility Test Suite (CTS), is an automated testing suite that includes both the PC side and the device side (the device being tested). It tests the hardware, software, interfaces, and performance of mobile phones. Only Android phone systems that pass the CTS test will receive a license from Google to ensure the compatibility of Android systems among different manufacturers.
dalvik	Contents related to the Android Dalvik virtual machine. After Android 5.0, the Dalvik virtual machine was no longer used and ART was adopted as the Android Java virtual machine. However, this directory still exists in the Android source code directory. This directory provides some tools.
developers	Developer Directory, providing some reference documents and sample demo codes for Android developers.
development	Provide Android application development infrastructure, including development tools, application examples, monkey commands, shell scripts and python scripts
device	The various devices and their configurations supported by Android, which constitutes the device product customization catalogue
external	The third-party open-source libraries introduced by Android. Google will incorporate some of the more outstanding open-source libraries into the Android source code, such as ppp, wpa_supplicant, libz, libcurl, etc.
frameworks	The core layer code of the Android application framework, written in Java and C++, including various system resources, System Services, SystemUI, etc., is a directory that is frequently dealt with in framework development.
hardware	Hardware abstraction layer code, such as GPS, WiFi, etc.
kernel	Linux Kernel source code directory, but Android does not provide this by default. Manufacturers (such as Rockchip) can integrate the kernel source code into this directory.
libcore	Android's Java core class libraries, such as java.lang package, libart library, JSON library, etc.
libnativehelper	The implementation code of various interfaces called during JNI programming, such as findClass, which will eventually be compiled into libnativehelper.so
packages	The source code of built-in Android apps, such as settings, input method, wallpaper, screen saver, etc.
pdk	The test codes and tools related to the Platform Development Kit (PDK) are aimed at assisting chip suppliers (such as Ralink Microelectronics, which is a semiconductor manufacturer) and original equipment manufacturers (OEMs) in migrating to the new version. Starting from Android 4.1, Google releases the PDK to some contracted OEMs and semiconductor manufacturers before releasing new Android versions. The purpose of this is to allow OEMs and

	semiconductor manufacturers to start adapting to the new Android version in advance. Therefore, the content of the PDK is not made public.
platform_testing	Android platform test program
prebuilts	Precompiled related content, including precompiled resources for x86 and arm architectures, such as NDK toolkits, cross-compilers, etc.
sdk	Android's Java layer SDK, which includes development tools provided for application developers. Application developers can quickly develop applications by leveraging the APIs in the SDK.
system	Android's underlying file system library, applications and components, such as busybox, init processes, etc.
test	Android vendor testing framework, such as vts (vendor test suite), mts (mainline test suite), and mlts. CTS ensures that there is a consistent call interface (API) between the App and the Android Framework, which enables App developers to write the same application that can run on different system versions (forward compatibility), different hardware platforms, and different devices manufactured by different manufacturers; VTS is similar to CTS. By testing the vendor interface, it ensures that the same version of the Android framework can run on different HALs, or that different Android frameworks can run on the same HAL. Mainline Test Suite (MTS) consists of a set of test frameworks and test cases, aiming to enhance the robustness, reliability and compliance of the mainline branch code. MTS is the Android machine learning test suite.
toolchain	Nothing much, just some Python test scripts
tools	Some special tools, such as the acloud command for connecting to Google Cloud Workstation, and apkzlib for APK packaging
vendor	Customized by the manufacturer, containing the codes related to the manufacturer
mkcombinedroot	Rockchip provides
rkbin	Rockchip provides, storing Binary files and tools related to Rockchip
RKDocs	Rockchip provides, storing development guidelines for chip modules, platform support list, chip platform-related documents, Android SDK development guide, etc. These documents are reference materials provided by RK for customers
rkst	Ricoh Micro provides
RKTools	Ricoh Micro provides, storing common tools for Linux and Windows operating system environments, such as image burning tools, etc.
u-boot	Ricoh Micro provides, storing the U-Boot source code developed by RK based on version v2017.09

Table 4.2.1.1 Introduction to the Source Code Directory of RK3568 Android SDK

4.2.2 Software Block Diagram of the SDK

The architecture diagram of the Android system is as follows:

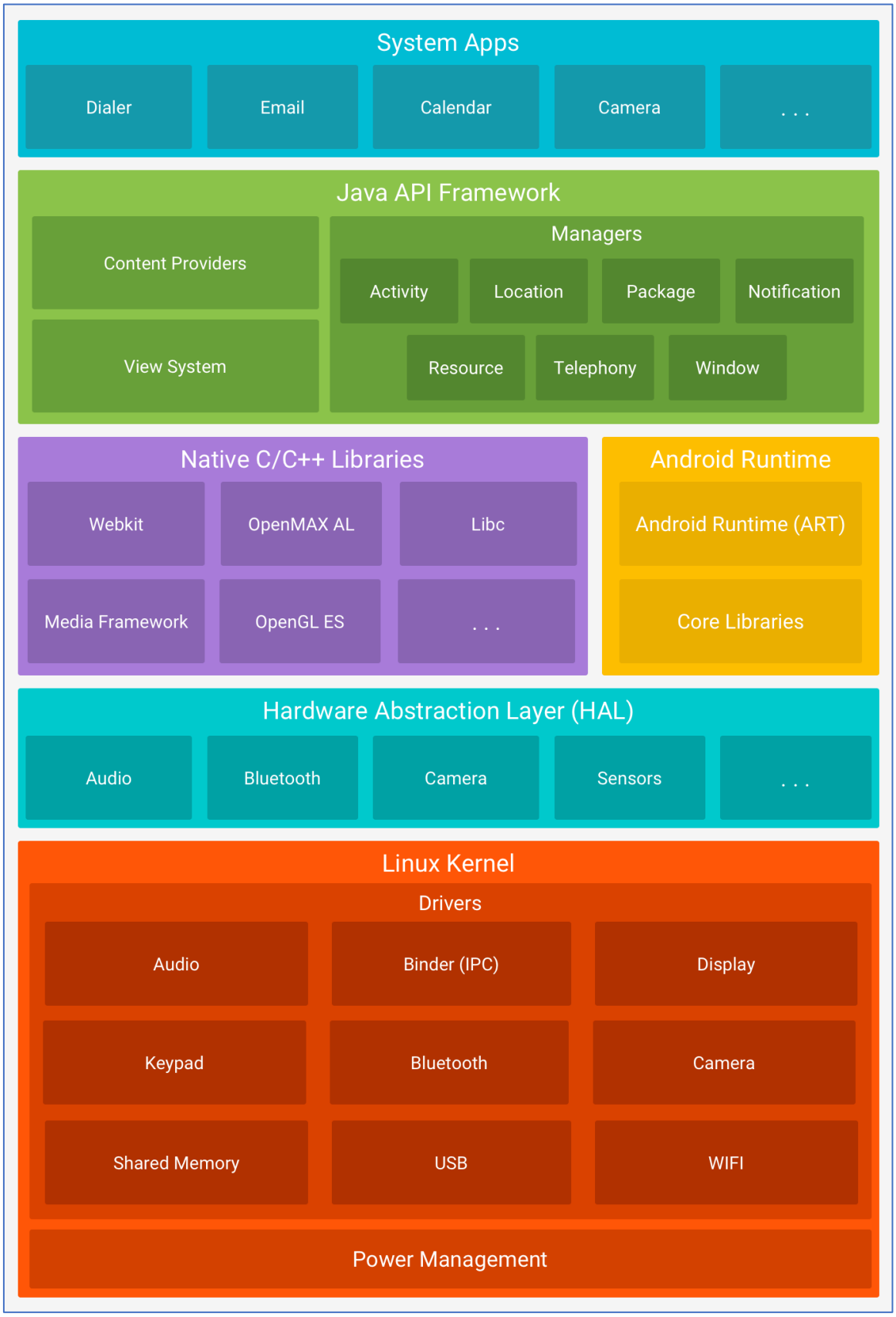


Figure 4.2.2.1 Android system architecture

The Android system architecture is divided into five layers, from top to bottom: the application layer (System Apps), the application framework layer (Java API Framework), the system runtime library layer (Android Runtime & Native C/C++ libraries), the hardware abstraction layer (HAL), and the Linux kernel layer (Linux Kernel).

① Application Layer

The application layer is the topmost layer of the entire Android system. It includes core system applications, input methods, and third-party applications, etc. Both built-in and non-built-in applications of the Android system belong to the application layer, which is responsible for direct interaction with users and is usually developed in Java.

② Application Framework Layer (Java API Framework)

The application framework layer provides various APIs required for developing Android applications. Application developers call the APIs provided by the framework layer to develop applications, including system applications. The framework layer is written in Java code, so it is also called Java Framework.

In addition, the framework layer also provides a series of important system services for applications. Application developers are allowed to use these services in their applications:

Service name	Explanation
Activity Manager	Manage the lifecycle of various applications and provide a universal navigation back function.
Location Manager	Provide location and positioning services.
Package Manager	Manage all applications installed on the Android system, including installation, uninstallation, and updates.
Notification Manager	Enable all applications to display notification information in the status bar.
Resource Manager	Manage a series of resources other than code (non-code resources), such as local strings, images, layout files, color files, etc.
Telephony Manager	Manage all mobile device functions.
Window Manager	Manage all programs with open windows.
Content Provider	Enable data sharing between different applications.
View System	Build basic components of applications, such as lists, grids, text boxes, buttons, etc.

Table 4.2.2.1 System Service Overview

③. System Runtime Library Layer (Native)

This layer is divided into two parts: C/C++ program libraries and Android runtime libraries.

C/C++ Program Libraries: These are provided for use by various components of the Android system. These functions are provided to developers through the application framework layer.

Android Runtime Library (Android Runtime): The runtime libraries are divided into core libraries and ART (after Android 5.0, the Dalvik virtual machine was replaced by ART). The core libraries integrate most of the functions of the Java language core library, allowing developers to write Android applications using the Java language. Under the ART mechanism, the system will perform a pre-compilation when installing applications, converting bytecode into machine code and storing it

locally. This way, when the application starts each time, there is no need to execute the compilation again, significantly improving efficiency!

④. Hardware Abstraction Layer (HAL/HIDL)

"HAL" is the abbreviation of "Hardware Abstraction Layer". It is located between the Framework and the Linux Kernel. The code of the HAL layer is generally composed of C/C++. The purpose of the Hardware Abstraction Layer is to abstract the underlying hardware and provide unified and standardized API interfaces to the upper layer; thus, the HAL layer is a software layer that shields the upper-layer applications from the specific workings of the underlying hardware. The upper-layer applications do not need to be concerned about how the underlying hardware works; they only need to call the unified interfaces provided by the HAL layer! Generally, when doing Android low-level development, the main focus is to modify the HAL layer to adapt to specific hardware.

On the other hand, the significance of the existence of the HAL layer is to protect intellectual property rights. Hardware manufacturers, considering their interests, do not want to disclose the implementation details and parameters related to the hardware devices. Therefore, hardware manufacturers or individuals can place the code they do not wish to open source (such as some core algorithms) in the HAL layer in the form of a dynamic link library, and the HAL layer is located in the user space (Linux user space), which does not belong to the Linux kernel, and does not need to follow the GPL open-source protocol.

⑤. Linux Kernel

The foundation of the Android system is the Linux kernel. The core system services of Android are based on the Linux kernel, and some Android-specific drivers have been added on top of it. The system's security, memory management, process management, network protocol stack, and driver model all rely on the Linux kernel.

4.2.3 SDK Version Query

The Android SDK version of the ALIENTEK RK3568 can be divided into RK version and ATK version. The RK version indicates that this SDK is a secondary development based on the official Android SDK of RK (Rockchip, Ralink Technology). The secondary development is carried out on the basis of the original RK SDK to adapt to our ALIENTEK ATK-DLRK3568 development platform and make some corresponding extensions.

The ATK version indicates the internal version number defined by the ALIENTEK Linux Technology Team for the RK3568 Android SDK. Each SDK release will have a corresponding version number.

In the Android SDK source code directory, execute the following command to query the ATK version of the current SDK:

```
realpath .repo/manifests/rk3568_android11_release.xml
```

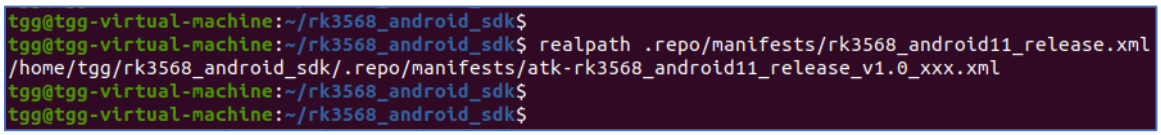


Figure 4.2.3.1 Query the ATK version of the SDK

As can be seen from the figure, the current ATK version of the SDK is V1.0, and the release date is xxx.

In the SDK source code directory, execute the following command to query the RK version of the current SDK:

```
vi .repo/manifests/include/rk_checkout_from_aosp.xml
```

```
<?xml version="1.0" encoding="UTF-8"?>
<manifest>
  <project groups="pdk" name="platform/build" path="build/make" remote="rk" revision="refs/tags/android-11.0-mid-rkr12">
    <copyfile dest="Makefile" src="core/root.mk"/>
    <linkfile dest="build/CleanSpec.mk" src="CleanSpec.mk"/>
    <linkfile dest="build/buildspec.mk.default" src="buildspec.mk.default"/>
```

Figure 4.2.3.2 Query the RK version of the SDK

As can be seen from the figure, the current RK version of the SDK is **android-11.0-mid-rkr12**.

4.3 SDK Automatic Compilation

This section introduces to users how to automatically compile the entire RK3568 Android SDK in one step. First, enter the top-level directory of the SDK. Before compiling, execute the following command to initialize the compilation environment:

```
source build/envsetup.sh
```

Note: Before opening a new terminal to compile the Android source code, you need to execute the "source build/envsetup.sh" command to initialize the compilation environment.

Then execute the "lunch" command to select a product (that is, to choose a compilation target and compilation items):

```
lunch
```

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ lunch

You're building on Linux

Lunch menu... pick a combo:
    1. PX30_Android11-user
    2. PX30_Android11-userdebug
    3. aosp_arm-eng
    4. aosp_arm64-eng
    5. aosp_blueline_car-userdebug
    6. aosp_bonito_car-userdebug
    7. aosp_car_arm-userdebug
    8. aosp_car_arm64-userdebug
    9. aosp_car_x86-userdebug
    10. aosp_car_x86_64-userdebug
    11. aosp_coral_car-userdebug
    12. aosp_crosshatch_car-userdebug
    13. aosp_flame_car-userdebug
    14. aosp_x86-eng
    15. aosp_x86_64-eng
    16. car_x86_64-userdebug
    17. qemu_trusty_arm64-userdebug
    18. rk3288_Android10-user
    19. rk3288_Android10-userdebug
```

```
20. rk3288_Android11-user
21. rk3288_Android11-userdebug
22. rk3326_pie-user
23. rk3326_pie-userdebug
24. rk3326_q-user
25. rk3326_q-userdebug
26. rk3326_r-user
27. rk3326_r-userdebug
28. rk3326_rgo-user
29. rk3326_rgo-userdebug
30. rk3328_atv-user
31. rk3328_atv-userdebug
32. rk3328_box-user
33. rk3328_box-userdebug
34. rk3328_box_32-user
35. rk3328_box_32-userdebug
36. rk3399_Android10-user
37. rk3399_Android10-userdebug
38. rk3399_Android11-user
39. rk3399_Android11-userdebug
40. rk3399_atv-user
41. rk3399_atv-userdebug
42. rk3399_mid-user
43. rk3399_mid-userdebug
44. rk3566_32bit-user
45. rk3566_32bit-userdebug
46. rk3566_eink-user
47. rk3566_eink-userdebug
48. rk3566_einkw6-user
49. rk3566_einkw6-userdebug
50. rk3566_r-user
51. rk3566_r-userdebug
52. rk3566_rgo-user
53. rk3566_rgo-userdebug
54. rk3568_r-user
55. rk3568_r-userdebug
56. rk356x_box-user
57. rk356x_box-userdebug
58. rk356x_box_32-user
59. rk356x_box_32-userdebug
60. uml-userdebug
```

After executing the "lunch" command (without any parameters), it will list all the products (compilation targets). For the rk3568 platform, we can choose either **rk3568_r-user** or **rk3568_r-**

userdebug; rk3568_r-user will compile the user version of the Android system image, while **rk3568_r-userdebug** will compile the userdebug version of the Android system image.

Here, taking the userdebug version as an example, input "**rk3568_r-userdebug**" or the corresponding number **55**:

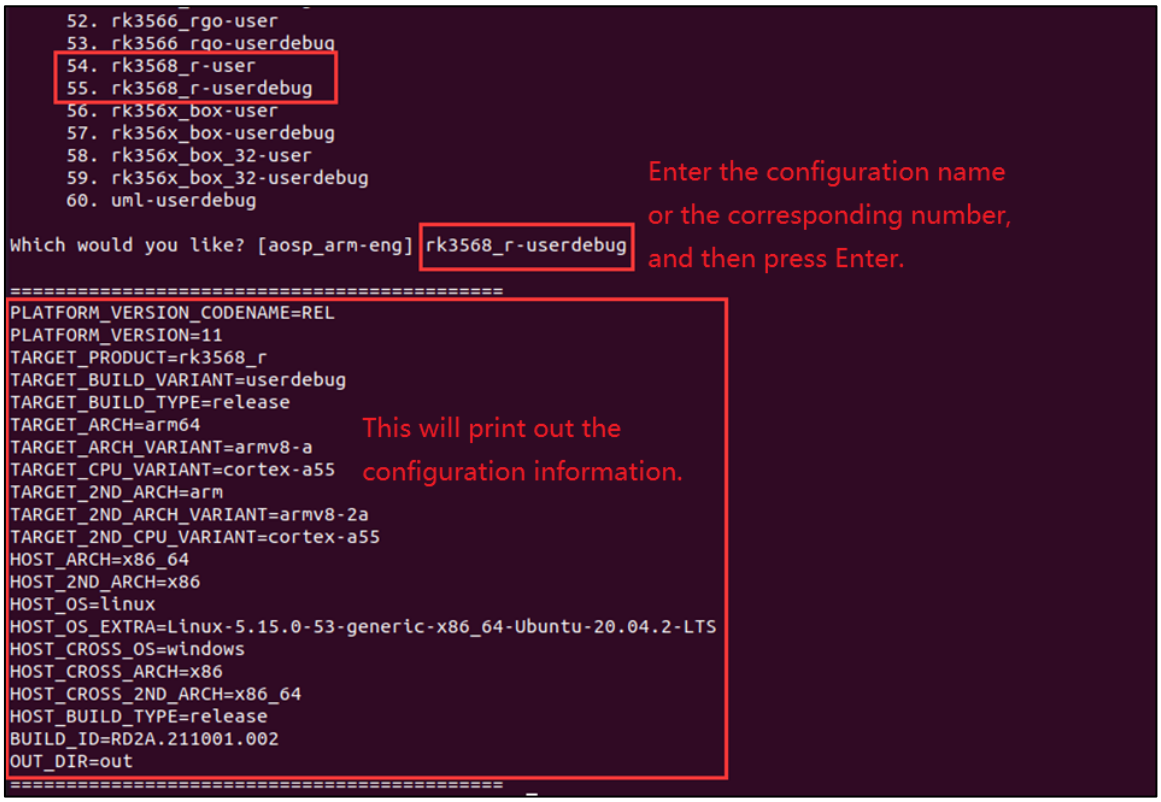


Figure 4.3-1 Selecting products for lunch

Alternatively, you can directly specify the product name when executing the "lunch" command, as follows:

```
lunch rk3568_r-userdebug
```

Now we can proceed with the compilation. Use the compiled script build.sh provided by RK for compilation.

build.sh is a compiled script packaged by RK. By using this script, users can conveniently quickly build the entire Android system image file and perform packaging operations on the image. It can automatically compile the entire Android SDK with one click, and also separately compile U-Boot, Linux Kernel, Android source code, etc. It is very convenient!

You can view the usage method of the build.sh script by executing the following command:

```
./build.sh -h
```

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ ./build.sh -h
./build.sh: 非法选项 -- h
USAGE: [-U] [-CK] [-A] [-p] [-o] [-u] [-v VERSION_NAME]
No ARGS means use default build option
WHERE: -U = build uboot
       -C = build kernel with Clang
       -K = build kernel
       -A = build android
       -p = will build packaging in IMAGE
       -o = build OTA package
       -u = build update.img
       -v = build android with 'user' or 'userdebug'
       -d = huild kernel dts name
       -V = build version
       -J = build jobs
tgg@tgg-virtual-machine:~/rk3568_android_sdk$
```

Figure 4.3-2 View the help information of the build.sh script

As shown in the table:

Parameters of the build.sh script	Explanation	Example
-U	Compile the U-Boot source code	./build.sh -U
-C	Compile the Linux Kernel source code using the clang compiler; if GMS certification is required, Google mandates the use of the clang compiler to compile the kernel source code	./build.sh -C
-K	Compile the Linux kernel source code	./build.sh -K
-A	Compile the Android source code	./build.sh -A
-p	Package the compiled images into the IMAGE directory	./build.sh -p
-o	Compile the OTA upgrade package	./build.sh -o
-u	Package all the images into update.img	./build.sh -u
-d	When compiling the Linux Kernel, specify the kernel device tree file	./build.sh -d
-J	Specify the number of compilation threads	./build.sh -J24

Table 4.3.1 Common Commands of build.sh

We will introduce the usage methods of these parameters to you later. For now, just have a basic understanding of them!

Under the top-level directory of the Android SDK, execute the following command to compile the entire Android SDK:

```
./build.sh -UKAp -J16
```

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk$
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ ./build.sh -UKAp -J16
will build u-boot
will build kernel
will build android
will build packaging in IMAGE
-------------------KERNEL_DTS:rk3568-atk-evb1-ddr4-v10

============================================
PLATFORM_VERSION_CODENAME=REL
PLATFORM_VERSION=11
TARGET_PRODUCT=rk3568_r
TARGET_BUILD_VARIANT=userdebug
TARGET_BUILD_TYPE=release
TARGET_ARCH=arm64
TARGET_ARCH_VARIANT=armv8-a
TARGET_CPU_VARIANT=cortex-a55
TARGET_2ND_ARCH=arm
TARGET_2ND_ARCH_VARIANT=armv8-2a
TARGET_2ND_CPU_VARIANT=cortex-a55
HOST_ARCH=x86_64
HOST_2ND_ARCH=x86
HOST_OS=linux
HOST_OS_EXTRA=Linux-5.15.0-53-generic-x86_64-Ubuntu-20.04.2-LTS
HOST_CROSS_OS=windows
HOST_CROSS_ARCH=x86
HOST_CROSS_2ND_ARCH=x86_64
HOST_BUILD_TYPE=release
BUILD_ID=RD2A.211001.002
OUT_DIR=out
============================================
start build uboot

#### build completed successfully (2 seconds) ####


#### build completed successfully  ####


#### build completed successfully (1 seconds) ####

grep: .config: 没有那个文件或目录
## make rk3568_defconfig -j24
  HOSTCC  scripts/basic/fixdep
  HOSTCC  scripts/kconfig/conf.o
```

Figure 4.3-3 Figure 4.3.3 One-click full-automatic compilation of Android SDK

-U: Indicates compiling U-Boot;

-K: Indicates compiling Linux Kernel;

-A: Indicates compiling Android;

-p: Indicates packaging the compiled image into the <SDK>/IMAGE directory;

-J16: Specifies the number of compilation threads as 16.

The entire compilation process will take a considerable amount of time, approximately 2 to 3 hours or longer, depending on the performance of each individual's computer.

During the compilation process, some problems may arise, causing the compilation to fail! After all, everyone's development environment has some differences to some extent, so everyone should try to follow the instructions in this document, especially the dependent software packages required in Section 4.1.1. In addition, ensure that the memory capacity of Ubuntu is at least **greater than or equal to 16G**. If the compilation fails due to insufficient memory, you can increase the memory capacity again and try compiling; if the computer hardware conditions do not allow it, it is no longer possible to further expand the memory capacity of the virtual Ubuntu system. At this time, we can try to expand the swap space of the Ubuntu system. The expansion method can be referred to as: Development Board CD-ROM A Disk - **Basic Materials -> 10_user_manual -> 03. Supplementary Documents -> [ALIENTEK] Ubuntu Expand Swap Space.pdf**. In addition, the disk space of the Ubuntu system should be guaranteed to have at least 400 to 500G.

If no unexpected situation occurs, the compilation will be successful, as shown in Figure 4.3.4:

```
lpmake I 04-22 00:09:37 770826 770826 builder.cpp:1031] [liblp]Partition system_ext will resize from 0 bytes t
lpmake I 04-22 00:09:37 770826 770826 builder.cpp:1031] [liblp]Partition vendor will resize from 0 bytes to 30
lpmake I 04-22 00:09:37 770826 770826 builder.cpp:1031] [liblp]Partition product will resize from 0 bytes to 2
lpmake I 04-22 00:09:37 770826 770826 builder.cpp:1031] [liblp]Partition odm will resize from 0 bytes to 62668
2023-04-22 00:09:51 - build_super_image.py - INFO    : Done writing image out/target/product/rk3568_r/super.im

#### build completed successfully (02:10:28 (hh:mm:ss)) ####

Build android ok!
make and copy android images
TARGET_PRODUCT=rk3568_r
TARGET_BASE_PARAMETER_IMAGE==device/rockchip/common/baseparameter/v2.0/baseparameter.img
HIGH_RELIABLE_RECOVERY_OTA=
BOARD_AVB_ENABLE=false
system filesysystem is ext4
create dtbo.img...
done.
create resource.img...
done.
create boot.img...
done.
create boot-debug.img...
done.
skip copy images: vendor_boot.img
skip copy images: vendor_boot-debug.img
create recovery.img...
done.
create super.img...
done.
create vbmeta.img...
BOARD_AVB_ENABLE is false, use default vbmeta.img
create misc.img.... done.
create uboot.img...
u-boot/trust.img not fount! Please make it from u-boot first!
create loader...
create config.cfg...
create baseparameter...done.
Make image ok!
make and copy packaging in IMAGE
Generate patch for hardware/rockchip/libhwjpeg done
Saved manifest to out/commit_id.xml
```

Compilation completed

Figure 4.3-4 The Android SDK compilation is complete.

If the string "**#### build completed successfully (compilation time (hh:mm:ss)) ####**" appears, it indicates that the compilation was successful. So, congratulations everyone!

After the compilation is completed, the generated image files will be packaged into the <SDK>/IMAGE directory (because when running the build.sh script for compilation, the **-p** parameter was added). This includes boot.img, uboot.img, misc.img, dtbo.img, recovery.img, super.img, vbmeta.img, etc., making it convenient for users to search.

There is a directory named "**RK3568_R_USERDEBUG_RK3568-ATK-EVB1-DDR4-V10__compilation time**" in the <SDK>/IMAGE directory, as shown below:

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk$
tgg@tgg-virtual-machine:~/rk3568_android_sdk$
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ cd IMAGE/
tgg@tgg-virtual-machine:~/rk3568_android_sdk/IMAGE$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/IMAGE$ ls
RK3568_R_USERDEBUG_RK3568-ATK-EVB1-DDR4-V10__20230423.1608
tgg@tgg-virtual-machine:~/rk3568_android_sdk/IMAGE$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/IMAGE$
```

Figure 4.3-5 The contents under the IMAGE directory

The naming convention for this directory is: **Product Name_Kernel Device Tree Name__Compilation Time**. Therefore, the directory names generated due to different compilation times are also different (all in capital letters). The image is packaged into the **RK3568_R_USERDEBUG_RK3568-ATK-EVB1-DDR4-V10/IMAGES** directory, as shown below:

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/IMAGE$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/IMAGE$ ls -lh RK3568_R_USERDEBUG_RK3568-ATK-EVB1-DDR4-V10__20230423.1608/IMAGES/
总用量 1.7G
-rwxrwxr-x 1 tgg tgg 1.0M 4月  23 19:31 baseparameter.img
-rw-rw-r-- 1 tgg tgg  33M 4月  23 19:31 boot-debug.img
-rw-rw-r-- 1 tgg tgg  32M 4月  23 19:31 boot.img
-rwxrwxr-x 1 tgg tgg 6.0K 4月  23 19:31 config.cfg
-rw-rw-r-- 1 tgg tgg  623 4月  23 19:31 dtbo.img
-rw-rw-r-- 1 tgg tgg 455K 4月  23 19:31 MiniLoaderAll.bin
-rw-rw-r-- 1 tgg tgg  48K 4月  23 19:31 misc.img
-rw-rw-r-- 1 tgg tgg  630 4月  23 19:31 parameter.txt
-rw-rw-r-- 1 tgg tgg  48K 4月  23 19:31 pcba_small_misc.img
-rw-rw-r-- 1 tgg tgg  49K 4月  23 19:31 pcba_whole_misc.img
-rw-rw-r-- 1 tgg tgg  66M 4月  23 19:31 recovery.img
-rw-rw-r-- 1 tgg tgg 497K 4月  23 19:31 resource.img
-rw-r--r-- 1 tgg tgg 1.6G 4月  23 19:32 super.img
-rw-rw-r-- 1 tgg tgg 4.0M 4月  23 19:32 uboot.img
-rwxrwxr-x 1 tgg tgg 4.0K 4月  23 19:32 vbmeta.img
tgg@tgg-virtual-machine:~/rk3568_android_sdk/IMAGE$
```

Figure 4.3-6 Mirror file

In addition, the build.sh script will also package the image into the <SDK>/rockdev directory; every time the build.sh script is run for compilation, regardless of whether the -p parameter is added or not, the compiled image will be packaged into the **<SDK>/rockdev** directory.

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ cd rockdev/
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rockdev$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rockdev$ ls
Image-rk3568_r
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rockdev$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rockdev$ cd Image-rk3568_r/
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rockdev/Image-rk3568_r$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rockdev/Image-rk3568_r$ ls
baseparameter.img  boot.img    dtbo.img          misc.img       pcba_small_misc.img  recovery.img  super.img  vbmeta.img
boot-debug.img     config.cfg  MiniLoaderAll.bin  parameter.txt  pcba_whole_misc.img  resource.img  uboot.img
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rockdev/Image-rk3568_r$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rockdev/Image-rk3568_r$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rockdev/Image-rk3568_r$
```

Figure 4.3-7 The contents under the "rockdev" directory

So, the image can also be found in the <SDK>/rockdev/Image-rk3568_r directory. Now, let's make a brief summary:

<SDK>/rockdev: This directory only contains one Android image. The newly compiled image will replace the old one.

<SDK>/IMAGE: This directory can store multiple Android images, similar to a backup. Every time a build (run the build.sh script for compilation, add the -p parameter) is performed, a directory named in the format of "**product name_kernel device tree name__compilation time**" will be created in <SDK>/IMAGE directory, and the compiled image will be packaged into the **IMAGES** subdirectory of that directory.

When compiling the Android source code, an "out" directory will be created under the <SDK> directory. As the name suggests, this directory is used to store the intermediate files and the final image generated during the compilation process of the Android source code, including jar files, .so dynamic libraries, img images, etc. The image files generated by compiling the Android source code are output to the **<SDK>/out/target/product/rk3568_r** (this directory is named after the product name) directory:

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk$
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ cd out/target/product/rk3568_r/
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r$ ls
android-info.txt        installed-files-odm.json            installed-files-vendor.txt  ramdisk-recovery.img
apex                    installed-files-odm.txt             kernel                      rebuild-dtbo.img
appcompat               installed-files-product.json        misc_info.txt               recovery
boot-debug.img          installed-files-product.txt         module-info.json            recovery.img
boot.img                installed-files-ramdisk-debug.json  module-info.json.rsp        root
build_fingerprint.txt   installed-files-ramdisk-debug.txt   obj                         super_empty.img
build_thumbprint.txt    installed-files-ramdisk.json        obj_arm                     super.img
cache                   installed-files-ramdisk.txt         odm                         symbols
cache.img               installed-files-recovery.json       odm.img                     system
clean_steps.mk          installed-files-recovery.txt        parameter.txt               system_ext
data                    installed-files-root.json           previous_build_config.mk    system_ext.img
debug_ramdisk           installed-files-root.txt            product                     system.img
dtb.img                 installed-files-system_ext.json     product.img                 testcases
fake_packages           installed-files-system_ext.txt      ramdisk                     vendor
gen                     installed-files.txt                 ramdisk-debug.img           vendor.img
installed-files.json    installed-files-vendor.json         ramdisk.img
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r$ ls -lh *.img
-rw-rw-r-- 1 tgg tgg  33M 4月  23 17:46 boot-debug.img
-rw-rw-r-- 1 tgg tgg  32M 4月  23 17:46 boot.img
-rw-rw-r-- 1 tgg tgg  57K 4月  23 17:44 cache.img
-rw-rw-r-- 1 tgg tgg 137K 4月  23 16:27 dtb.img
-rw-rw-r-- 1 tgg tgg 609K 4月  23 17:44 odm.img
-rw-rw-r-- 1 tgg tgg 203M 4月  23 19:19 product.img
-rw-rw-r-- 1 tgg tgg 947K 4月  23 17:46 ramdisk-debug.img
-rw-rw-r-- 1 tgg tgg 804K 4月  23 17:46 ramdisk.img
-rw-rw-r-- 1 tgg tgg  35M 4月  23 19:06 ramdisk-recovery.img
-rw-rw-r-- 1 tgg tgg  623 4月  23 17:19 rebuild-dtbo.img
-rw-rw-r-- 1 tgg tgg  66M 4月  23 19:06 recovery.img
-rw-r--r-- 1 tgg tgg 4.6K 4月  23 17:44 super_empty.img
-rw-r--r-- 1 tgg tgg 1.6G 4月  23 19:25 super.img
-rw-rw-r-- 1 tgg tgg 118M 4月  23 19:19 system_ext.img
-rw-rw-r-- 1 tgg tgg 933M 4月  23 19:25 system.img
-rw-rw-r-- 1 tgg tgg 293M 4月  23 19:21 vendor.img
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r$
```

The img image generated by compiling the Android source code is output to this directory.

However, it does not include uboot.img and MiniLoaderAll.bin. These two image files are output in the uboot source code directory.

Figure 4.3-8 out directory

Therefore, it can be known that some of the images in the <SDK>/IMAGE and <SDK>/rockdev directories actually originated from the <SDK>/out/target/product/rk3568_r directory and were copied from there. RK did this to facilitate users in finding these images, so it bundled these images together into the <SDK>/IMAGE or <SDK>/rockdev directories.

4.4 Image Introduction

In the previous section, we compiled the RK3568 Android SDK through the build.sh script and compiled the Android system images, including boot.img, dtbo.img, MiniLoaderAll.bin, misc.img, recovery.img, super.img, and uboot.img, etc. This section will briefly introduce these image files to you.

Image	Explanation
boot.img	Android format boot image, which is composed of multiple images, including ramdisk root file system, kernel image Image, kernel DTB, resource image resource.img; boot.img will eventually be burned onto the boot partition of the development board.
boot-debug.img	The difference from boot.img lies in: the user version firmware can burn this boot-debug.img for root permission operations.
baseparameter.img	It is used to store configuration information such as display resolution and display effect adjustment for the Rockchip platform, ensuring that the effect remains the same after power off and restart, and ensuring

	consistent display effect throughout the boot process; this image will be burned onto the baseparameter partition.
dtbo.img	For Android versions 10.0 and above, the Device Tree Overlays function is supported, used for compatibility among multiple products; the dtbo.img image needs to be burned to the dtbo partition.
MiniLoaderAll.bin	A loader code running before the RK3568 platform U-Boot (a stage earlier than U-Boot), MiniLoaderAll.bin consists of two parts: TPL for initializing DDR and running in SRAM; while SPL runs in DDR and mainly responsible for loading and booting uboot.img.
misc.img	It contains Android BCB (Bootloader Control Block) information and A/B system information; this image will be burned onto the misc partition. The misc partition is an important partition, including BCB (used to control the system boot process, Android system and BootLoader, and recovery interaction mechanism) and A/B system data blocks. It contains recovery-wipe boot identification information, which will perform recovery after burning; commonly used.
pcba_small_misc.img	It contains PCBA boot identification information, which will enter the simple version PCBA mode after burning; not commonly used
pcba_whole_misc.img	It contains PCBA boot identification information, which will enter the complete version PCBA mode after burning; not commonly used
parameter.txt	A txt text file, which is the partition table for the RK3568 platform (records partition names and the starting and ending addresses of each partition); when burning the image, the parameter.txt file does not need to be burned to the Flash device (eMMC or NAND), and its information will be read to define the partitions.
recovery.img	This image is composed of multiple images, including recovery-ramdisk root file system, kernel image Image, DTB, DTBO, and resource.img; recovery.img is used to enter the recovery mode, which is a mode used for repairing and upgrading the device. Recovery.img will eventually be burned onto the development board recovery partition.
resource.img	The RK-defined resource image includes logo and DTB; resource.img is not burned alone, but is packaged in boot.img and recovery.img.
super.img	Super image is formed by merging multiple images, including odm.img, product.img, system.img, system_ext.img, vendor.img, and super.img. It will eventually be burned into the super partition of the development board.
uboot.img	It is formed by merging multiple images, including the trust image (ARM Trusted Firmware + OP-TEE OS), u-boot image, and u-boot dtb. During the compilation of U-Boot, these images will be packaged into an uboot.img. The uboot.img will eventually be burned into the uboot partition of the development board.

vbmeta.img	It contains AVB (Android Verified Boot) verification information for AVB verification; AVB is a firmware verification process designed by Google.

Table 4.4.1 Introduction to RK3568 Android System Image

4.5 Independent Compilation

In the previous section, we introduced to users how to perform a one-click automatic compilation of the entire Android SDK. This section will guide users on how to independently compile individual source codes, such as independently compiling the U-Boot source code, independently compiling the Linux Kernel source code, and independently compiling the Android source code.

4.5.1 Independent Compilation of U-Boot

By using the build.sh script, independently compile the U-Boot source code. Execute the following command in the top-level directory of the SDK:

```
./build.sh -U -J16
```

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ ./build.sh -U -J16
will build u-boot
-------------------KERNEL_DTS:rk3568-atk-evb1-ddr4-v10

============================================
PLATFORM_VERSION_CODENAME=REL
PLATFORM_VERSION=11
TARGET_PRODUCT=rk3568_r
TARGET_BUILD_VARIANT=userdebug
TARGET_BUILD_TYPE=release
TARGET_ARCH=arm64
TARGET_ARCH_VARIANT=armv8-a
TARGET_CPU_VARIANT=cortex-a55
TARGET_2ND_ARCH=arm
TARGET_2ND_ARCH_VARIANT=armv8-2a
TARGET_2ND_CPU_VARIANT=cortex-a55
HOST_ARCH=x86_64
HOST_2ND_ARCH=x86
HOST_OS=linux
HOST_OS_EXTRA=Linux-5.15.0-53-generic-x86_64-Ubuntu-20.04.2-LTS
HOST_CROSS_OS=windows
HOST_CROSS_ARCH=x86
HOST_CROSS_2ND_ARCH=x86_64
HOST_BUILD_TYPE=release
BUILD_ID=RD2A.211001.002
OUT_DIR=out
============================================
start build uboot
  CLEAN   dts/../arch/arm/dts
  CLEAN   dts
  CLEAN   examples/standalone
  CLEAN   tools
  CLEAN   tools/lib tools/common
  CLEAN   spl/arch spl/board spl/cmd spl/common spl/disk spl/drivers spl/dts spl/env spl/fs spl/lib spl/u-boot
pl.lds spl/u-boot-spl.map spl/u-boot-spl-nodtb.bin spl/u-boot-spl.sym tpl/arch tpl/board tpl/common tpl/disk t
 tpl/u-boot-tpl.map tpl/u-boot-tpl-nodtb.bin tpl/u-boot-tpl.sym
  CLEAN   u-boot-dtb.bin u-boot.lds u-boot.dtb u-boot.map u-boot-nodtb.bin u-boot.srec u-boot.bin u-boot u-boo
0000.bin bl31_0xfdcce000.bin bl31_0x0006a000.bin u-boot-nodtb.bin bl31_0xfdcc1000.bin tee.bin bl31_0x00068000.

#### build completed successfully  ####
```

Figure 4.5.1.1 Use the build.sh script to compile U-Boot (1)

```
Build uboot ok!
package resoure.img with charger images

Pack ./tools/images/ & ../kernel/resource.img to resource.img ...
Unpacking old image(../kernel/resource.img):
rk-kernel.dtb rk3568-atk-evb1-mipi-dsi-1080p#_saradc_ch2=341.dtb rk3568-atk-evb1-mipi-dsi-720p#_saradc_ch
nel.bmp battery_0.bmp battery_1.bmp 12
Pack to resource.img successed!

Packed resources:
rk-kernel.dtb rk3568-atk-evb1-mipi-dsi-1080p#_saradc_ch2=341.dtb rk3568-atk-evb1-mipi-dsi-720p#_saradc_ch
nel.bmp battery_0.bmp battery_1.bmp 12

./resource.img with battery images is ready
/home/tgg/rk3568_android_sdk
make and copy android images
TARGET_PRODUCT=rk3568_r
TARGET_BASE_PARAMETER_IMAGE==device/rockchip/common/baseparameter/v2.0/baseparameter.img
HIGH_RELIABLE_RECOVERY_OTA=
BOARD_AVB_ENABLE=false
system filesysystem is ext4
create dtbo.img...
done.
create resource.img...
done.
create boot.img...
done.
create boot-debug.img...
done.
skip copy images: vendor_boot.img
skip copy images: vendor_boot-debug.img
create recovery.img...
done.
create super.img...
done.
create vbmeta.img...
BOARD_AVB_ENABLE is false, use default vbmeta.img
create misc.img.... done.
create uboot.img...
u-boot/trust.img not fount! Please make it from u-boot first!
create loader...
create config.cfg...
create baseparameter...done.
Make image ok!
tgg@tgg-virtual-machine:~/rk3568_android_sdk$
```

Figure 4.5.1.2 Use the build.sh script to compile U-Boot (version 2).

The compiled image files are output to the **<SDK>/u-boot** directory, including **uboot.img** (which is formed by merging multiple images, including ARM Trusted Firmware + OP-TEE OS, u-boot image, and u-boot dtb) and **rk356x_spl_loader_v1.13.112.bin** (also known as MiniLoaderAll.bin).

The build.sh script will package these images into the **<SDK>/rockdev/Image-ProductName/** directory:

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ cd rockdev/Image-rk3568_r/
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rockdev/Image-rk3568_r$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rockdev/Image-rk3568_r$ ls -l uboot.img MiniLoaderAll.bin
-rw-rw-r-- 1 tgg tgg  465344 4月  24 12:06 MiniLoaderAll.bin
-rw-rw-r-- 1 tgg tgg 4194304 4月  24 12:06 uboot.img
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rockdev/Image-rk3568_r$
```

Figure 4.5.1.3 Mirror

Among them, MiniLoaderAll.bin is copied from <SDK>/u-boot/rk356x_spl_loader_v1.13.112.bin.

4.5.2 Independently compiling Kernel

By using the build.sh script, independently compile the source code of the Linux Kernel. Execute the following commands in the top-level directory of the SDK:

```
./build.sh -K -J16
```

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ ./build.sh -K -J16
will build kernel
-------------------KERNEL_DTS:rk3568-atk-evb1-ddr4-v10

============================================
PLATFORM_VERSION_CODENAME=REL
PLATFORM_VERSION=11
TARGET_PRODUCT=rk3568_r
TARGET_BUILD_VARIANT=userdebug
TARGET_BUILD_TYPE=release
TARGET_ARCH=arm64
TARGET_ARCH_VARIANT=armv8-a
TARGET_CPU_VARIANT=cortex-a55
TARGET_2ND_ARCH=arm
TARGET_2ND_ARCH_VARIANT=armv8-2a
TARGET_2ND_CPU_VARIANT=cortex-a55
HOST_ARCH=x86_64
HOST_2ND_ARCH=x86
HOST_OS=linux
HOST_OS_EXTRA=Linux-5.15.0-53-generic-x86_64-Ubuntu-20.04.2-LTS
HOST_CROSS_OS=windows
HOST_CROSS_ARCH=x86
HOST_CROSS_2ND_ARCH=x86_64
HOST_BUILD_TYPE=release
BUILD_ID=RD2A.211001.002
OUT_DIR=out
============================================
Start build kernel
  CLEAN   .
  CLEAN   certs
pwd: /home/tgg/rk3568_android_sdk/kernel
pwd: /home/tgg/rk3568_android_sdk/kernel
  CLEAN   drivers/scsi
  CLEAN   fs/unicode
  CLEAN   kernel
  CLEAN   lib
  CLEAN   net/wireless
  CLEAN   security/selinux
  CLEAN   usr
  CLEAN   .tmp_versions

#### build completed successfully (18 seconds) ####
```

Figure 4.5.2.1 Compile the Kernel using the build.sh script (1)

```
/home/tgg/rk3568_android_sdk
Build kernel ok!
package resource.img with charger images

Pack ./tools/images/ & ../kernel/resource.img to resource.img ...
Unpacking old image(../kernel/resource.img):
rk-kernel.dtb rk3568-atk-evb1-mipi-dsi-720p#_saradc_ch2=0.dtb rk3568-atk-evb1-mipi-dsi-1080p#_saradc_ch2
Pack to resource.img successed!

Packed resources:
rk-kernel.dtb rk3568-atk-evb1-mipi-dsi-1080p#_saradc_ch2=341.dtb rk3568-atk-evb1-mipi-dsi-720p#_saradc_c
nel.bmp battery_0.bmp battery_1.bmp 12

./resource.img with battery images is ready
/home/tgg/rk3568_android_sdk
make and copy android images
TARGET_PRODUCT=rk3568_r
TARGET_BASE_PARAMETER_IMAGE==device/rockchip/common/baseparameter/v2.0/baseparameter.img
HIGH_RELIABLE_RECOVERY_OTA=
BOARD_AVB_ENABLE=false
system filesysystem is ext4
create dtbo.img...
done.
create resource.img...
done.
create boot.img...
done.
create boot-debug.img...
done.
skip copy images: vendor_boot.img
skip copy images: vendor_boot-debug.img
create recovery.img...
done.
create super.img...
done.
create vbmeta.img...
BOARD_AVB_ENABLE is false, use default vbmeta.img
create misc.img.... done.
create uboot.img...
u-boot/trust.img not fount! Please make it from u-boot first!
create loader...
create config.cfg...
create baseparameter...done.
Make image ok!
```

Figure 4.5.2.2 Compile the Kernel using the build.sh script (2)

After the compilation is completed, two image files, namely boot.img and resource.img, will be generated in the <SDK>/kernel directory.

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk$
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ cd kernel/
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel$ ls -lh boot.img resource.img
-rw-rw-r-- 1 tgg tgg  31M 4月  24 13:05 boot.img
-rw-rw-r-- 1 tgg tgg 497K 4月  24 13:05 resource.img
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel$
```

Figure 4.5.2.3 Mirror

boot.img is an Android format boot image (which can be viewed using the "file" command). It is formed by merging multiple images. For the RK3568 platform, the boot.img burned to the boot partition of the development board contains four parts: the kernel image Image, the kernel DTB, the ramdisk, and the resource.img.

However, it should be noted that the boot.img generated in the kernel source code directory after compiling with the command "./build.sh -K -J16" is not the final boot.img to be burned to the boot partition of the development board. Because this boot.img is not complete, it only contains the kernel image Image and the resource.img, and does not include the ramdisk (ramdisk.img is generated when compiling the Android source code). You can unpack this boot.img using the "unpack_bootimg" tool to find out more.

So, how to generate a complete version of boot.img (including the kernel image Image, the kernel DTB, the ramdisk, and the resource.img)? Simply execute the following command in the top-level directory of the SDK:

```
make bootimage
```

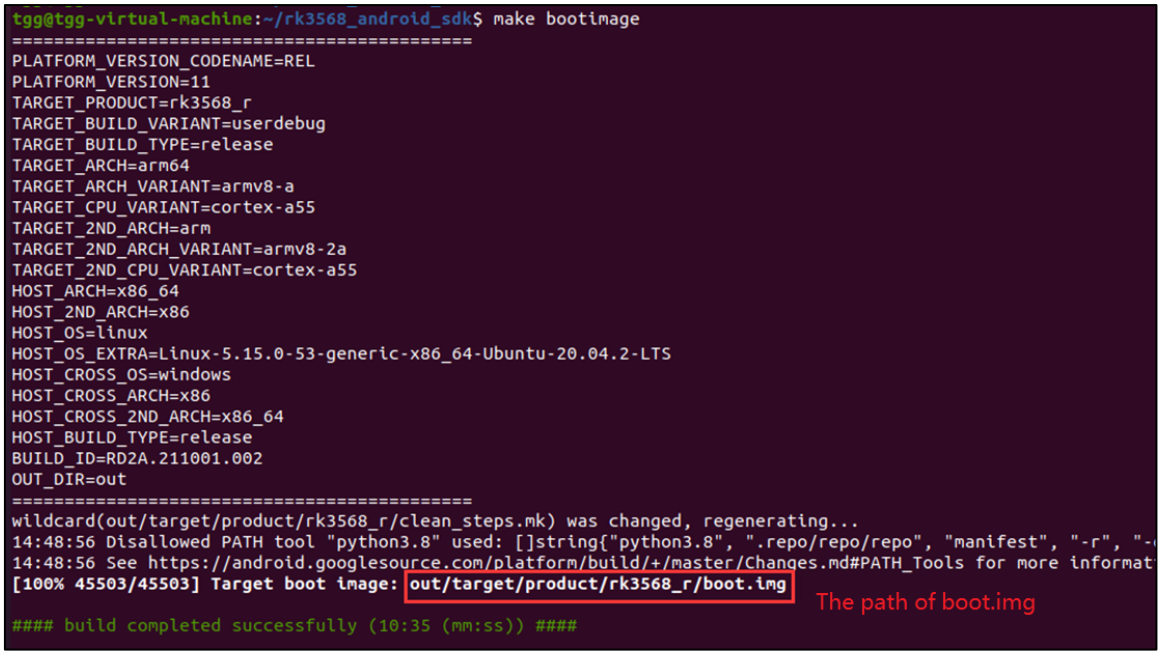


Figure 4.5.2.4 Generate boot.img

Execute the "make bootimage" command in the <SDK> directory to compile the Android source code and generate boot.img. During the compilation process, ramdisk.img will be generated. Then, ramdisk.img, the kernel image Image, the kernel DTB, and resource.img will be packaged into a boot.img. This boot.img will be output in the <SDK>/out/target/product/rk3568_r/ directory.

Therefore, out/target/product/rk3568_r/boot.img is the final boot.img that is burned onto the boot partition of the development board, not the boot.img in the kernel source code directory. This must be noted! **And the resource.img image is not burned separately, but is packaged into the boot.img.**

4.5.3 Independently compiling Android source code

The RK3568 Android SDK package integrates Android 11 source code, Linux Kernel source code, and u-boot source code. Execute the following command in the top-level directory of the SDK to compile the Android source code:

```
./build.sh -A -J16
```

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ ./build.sh -A -J16
will build android
-------------------KERNEL_DTS:rk3568-atk-evb1-ddr4-v10

============================================
PLATFORM_VERSION_CODENAME=REL
PLATFORM_VERSION=11
TARGET_PRODUCT=rk3568_r
TARGET_BUILD_VARIANT=userdebug
TARGET_BUILD_TYPE=release
TARGET_ARCH=arm64
TARGET_ARCH_VARIANT=armv8-a
TARGET_CPU_VARIANT=cortex-a55
TARGET_2ND_ARCH=arm
TARGET_2ND_ARCH_VARIANT=armv8-2a
TARGET_2ND_CPU_VARIANT=cortex-a55
HOST_ARCH=x86_64
HOST_2ND_ARCH=x86
HOST_OS=linux
HOST_OS_EXTRA=Linux-5.15.0-53-generic-x86_64-Ubuntu-20.04.2-LTS
HOST_CROSS_OS=windows
HOST_CROSS_ARCH=x86
HOST_CROSS_2ND_ARCH=x86_64
HOST_BUILD_TYPE=release
BUILD_ID=RD2A.211001.002
OUT_DIR=out
============================================
package resoure.img with charger images

Pack ./tools/images/ & ../kernel/resource.img to resource.img ...
Unpacking old image(../kernel/resource.img):
rk-kernel.dtb rk3568-atk-evb1-mipi-dsi-1080p#_saradc_ch2=341.dtb rk3568-atk-evb1-mipi-dsi-720p#_saradc
nel.bmp battery_0.bmp battery_1.bmp 12
Pack to resource.img successed!

Packed resources:
rk-kernel.dtb rk3568-atk-evb1-mipi-dsi-1080p#_saradc_ch2=341.dtb rk3568-atk-evb1-mipi-dsi-720p#_saradc
nel.bmp battery_0.bmp battery_1.bmp 12

./resource.img with battery images is ready
```

Figure 4.5.3.1 Use the build.sh script to compile the Android source code (1)

```
#### build completed successfully (09:42 (mm:ss)) ####

Build android ok!
make and copy android images
TARGET_PRODUCT=rk3568_r
TARGET_BASE_PARAMETER_IMAGE==device/rockchip/common/baseparameter/v2.0/baseparameter.img
HIGH_RELIABLE_RECOVERY_OTA=
BOARD_AVB_ENABLE=false
system filesysystem is ext4
create dtbo.img...
done.
create resource.img...
done.
create boot.img...
done.
create boot-debug.img...
done.
skip copy images: vendor_boot.img
skip copy images: vendor_boot-debug.img
create recovery.img...
done.
create super.img...
done.
create vbmeta.img...
BOARD_AVB_ENABLE is false, use default vbmeta.img
create misc.img.... done.
create uboot.img...
u-boot/trust.img not fount! Please make it from u-boot first!
create loader...
create config.cfg...
create baseparameter...done.
Make image ok!
```

Figure 4.5.3.2 Use the build.sh script to compile the Android source code (2)

After the compilation is successful, the generated image is stored in the <SDK>/out/target/product/product_name/directory.

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r$ ls -l *.img
-rw-rw-r-- 1 tgg tgg   33654784 4月  24 16:49 boot-debug.img
-rw-rw-r-- 1 tgg tgg   33507328 4月  24 16:49 boot.img
-rw-rw-r-- 1 tgg tgg      57492 4月  24 16:50 cache.img
-rw-rw-r-- 1 tgg tgg     139706 4月  24 16:49 dtb.img
-rw-rw-r-- 1 tgg tgg     622644 4月  24 16:50 odm.img
-rw-rw-r-- 1 tgg tgg  212701272 4月  24 16:50 product.img
-rw-rw-r-- 1 tgg tgg     969257 4月  24 16:49 ramdisk-debug.img
-rw-rw-r-- 1 tgg tgg     822327 4月  24 16:49 ramdisk.img
-rw-rw-r-- 1 tgg tgg   36503448 4月  24 16:49 ramdisk-recovery.img
-rw-rw-r-- 1 tgg tgg        623 4月  24 16:49 rebuild-dtbo.img
-rw-rw-r-- 1 tgg tgg   69189632 4月  24 16:49 recovery.img
-rw-r--r-- 1 tgg tgg       4644 4月  24 16:49 super_empty.img
-rw-r--r-- 1 tgg tgg 1610862136 4月  24 16:51 super.img
-rw-rw-r-- 1 tgg tgg  123277376 4月  24 16:50 system_ext.img
-rw-rw-r-- 1 tgg tgg  977600756 4月  24 16:51 system.img
-rw-rw-r-- 1 tgg tgg  306614396 4月  24 16:50 vendor.img
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r$
```

Figure 4.5.3.3 out directory

This directory generates many img images, but not all of them need to be burned onto the development board. Some images are just intermediate products and are not the final required images, such as odm.img, product.img, system.img, vendor.img, system_ext.img. These images will eventually be packaged into a super.img image, and only the super.img needs to be burned onto the development board! For the RK3568 platform, the images that need to be burned onto the development board are as follows:

boot.img (Or boot-debug.img, which is not commonly used)

recovery.img

rebuild-dtbo.img

super.img

parameter.txt (The partition table file will not be burned onto the development board. It is needed when burning the image.)

The build.sh script will copy (package) these images to the **<SDK>/rockdev/Image-Product Name/** directory.

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rockdev/Image-rk3568_r$ ls -l
总用量 1711612
-rwxrwxr-x 1 tgg tgg    1048576 4月  22 15:50 baseparameter.img
-rw-rw-r-- 1 tgg tgg   33654784 4月  24 16:49 boot-debug.img
-rw-rw-r-- 1 tgg tgg   33507328 4月  24 16:49 boot.img
-rwxrwxr-x 1 tgg tgg       6129 4月  22 15:50 config.cfg
-rw-rw-r-- 1 tgg tgg        623 4月  24 16:49 dtbo.img   Copied from rebuild-dtbo.img
-rw-rw-r-- 1 tgg tgg     465344 4月  24 12:06 MiniLoaderAll.bin
-rw-rw-r-- 1 tgg tgg      49152 4月  22 16:31 misc.img
-rw-rw-r-- 1 tgg tgg        630 4月  24 16:49 parameter.txt
-rw-rw-r-- 1 tgg tgg      49152 4月  22 16:31 pcba_small_misc.img
-rw-rw-r-- 1 tgg tgg      49153 4月  22 16:31 pcba_whole_misc.img
-rw-rw-r-- 1 tgg tgg   69189632 4月  24 16:49 recovery.img
-rw-rw-r-- 1 tgg tgg     508928 4月  24 16:49 resource.img
-rw-r--r-- 1 tgg tgg 1610862136 4月  24 16:51 super.img
-rw-rw-r-- 1 tgg tgg    4194304 4月  24 12:06 uboot.img
-rwxrwxr-x 1 tgg tgg       4096 4月  22 15:50 vbmeta.img
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rockdev/Image-rk3568_r$
```

Figure 4.5.3.4 "rockdev directory"

In addition to using the build.sh script for compilation, you can also directly use make for compilation. To compile the Android source code, execute the following command:

```
make -j16
```

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ make -j16
============================================
PLATFORM_VERSION_CODENAME=REL
PLATFORM_VERSION=11
TARGET_PRODUCT=rk3568_r
TARGET_BUILD_VARIANT=userdebug
TARGET_BUILD_TYPE=release
TARGET_ARCH=arm64
TARGET_ARCH_VARIANT=armv8-a
TARGET_CPU_VARIANT=cortex-a55
TARGET_2ND_ARCH=arm
TARGET_2ND_ARCH_VARIANT=armv8-2a
TARGET_2ND_CPU_VARIANT=cortex-a55
HOST_ARCH=x86_64
HOST_2ND_ARCH=x86
HOST_OS=linux
HOST_OS_EXTRA=Linux-5.15.0-53-generic-x86_64-Ubuntu-20.04.2-LTS
HOST_CROSS_OS=windows
HOST_CROSS_ARCH=x86
HOST_CROSS_2ND_ARCH=x86_64
HOST_BUILD_TYPE=release
BUILD_ID=RD2A.211001.002
OUT_DIR=out
============================================
19:36:57 Disallowed PATH tool "python3.8" used: []string{"python3.8", ".repo/repo/repo", "manifest", "-r", "-o", "
19:36:57 See https://android.googlesource.com/platform/build/+/master/Changes.md#PATH_Tools for more information.
[100% 77/77] Target super fs image for debug: out/target/product/rk3568_r/super.img
2023-04-24 19:37:04 - build_super_image.py - INFO    : Building super image from info dict...
2023-04-24 19:37:04 - sparse_img.py - INFO    : Total of 239401 4096-byte output blocks in 18 input chunks.
2023-04-24 19:37:04 - sparse_img.py - INFO    : Total of 30192 4096-byte output blocks in 3 input chunks.
2023-04-24 19:37:04 - sparse_img.py - INFO    : Total of 75087 4096-byte output blocks in 8 input chunks.
2023-04-24 19:37:04 - sparse_img.py - INFO    : Total of 52087 4096-byte output blocks in 5 input chunks.
2023-04-24 19:37:04 - sparse_img.py - INFO    : Total of 153 4096-byte output blocks in 2 input chunks.
2023-04-24 19:37:04 - common.py - INFO    :   Running: "lpmake --metadata-size 65536 --super-name super --metadata
tion system:readonly:980586496:rockchip_dynamic_partitions --image system=out/target/product/rk3568_r/system.img
ut/target/product/rk3568_r/system_ext.img --partition vendor:readonly:307556352:rockchip_dynamic_partitions --imag
ckchip_dynamic_partitions --image product=out/target/product/rk3568_r/product.img --partition odm:readonly:626688
tput out/target/product/rk3568_r/super.img"
2023-04-24 19:37:34 - common.py - INFO    : lpmake I 04-24 19:37:04   900   900 builder.cpp:1031] [liblp]Partition
lpmake I 04-24 19:37:04   900   900 builder.cpp:1031] [liblp]Partition system_ext will resize from 0 bytes to 123
lpmake I 04-24 19:37:04   900   900 builder.cpp:1031] [liblp]Partition vendor will resize from 0 bytes to 30755635
lpmake I 04-24 19:37:04   900   900 builder.cpp:1031] [liblp]Partition product will resize from 0 bytes to 213348
lpmake I 04-24 19:37:04   900   900 builder.cpp:1031] [liblp]Partition odm will resize from 0 bytes to 626688 byte
2023-04-24 19:37:34 - build_super_image.py - INFO    : Done writing image out/target/product/rk3568_r/super.img

#### build completed successfully (38 seconds) ####
```

Figure 4.5.3.5 Use make for compilation

The compiled image generated is also stored in the **out/target/product/product_name/** directory, but it will not package the image into the <SDK>/rockdev directory.

Before compiling the Android source code by executing the commands "./build.sh -A -J16" or "make -j16", it is necessary to pre-compile the Linux Kernel. There are two reasons for this:

①, when creating boot.img and recovery.img, kernel images (Image, kernel DTB, and resource.img) need to be provided, which will be packaged into boot.img and recovery.img. These images need to be obtained after compiling the Linux Kernel;

②, the .ko modules (such as WiFi driver modules) generated by compiling the kernel source code will be packaged into vendor.img.

1. Generate boot.img separately

Executing the commands "./build.sh -A -J16" or "make -j16" is used to compile the entire Android source code. After the compilation is completed, boot.img, recovery.img, dtbo.img, odm.img, system.img, and vendor.img are obtained. These images can also be generated separately using the make command.

The following command can be executed to generate boot.img:

```
make bootimage -j16
```

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ make bootimage -j16
============================================
PLATFORM_VERSION_CODENAME=REL
PLATFORM_VERSION=11
TARGET_PRODUCT=rk3568_r
TARGET_BUILD_VARIANT=userdebug
TARGET_BUILD_TYPE=release
TARGET_ARCH=arm64
TARGET_ARCH_VARIANT=armv8-a
TARGET_CPU_VARIANT=cortex-a55
TARGET_2ND_ARCH=arm
TARGET_2ND_ARCH_VARIANT=armv8-2a
TARGET_2ND_CPU_VARIANT=cortex-a55
HOST_ARCH=x86_64
HOST_2ND_ARCH=x86
HOST_OS=linux
HOST_OS_EXTRA=Linux-5.15.0-53-generic-x86_64-Ubuntu-20.04.2-LTS
HOST_CROSS_OS=windows
HOST_CROSS_ARCH=x86
HOST_CROSS_2ND_ARCH=x86_64
HOST_BUILD_TYPE=release
BUILD_ID=RD2A.211001.002
OUT_DIR=out
============================================
21:16:14 Disallowed PATH tool "python3.8" used: []string{"python3.8", ".repo/repo/repo", "manifest", "-
21:16:14 See https://android.googlesource.com/platform/build/+/master/Changes.md#PATH_Tools for more in
[100% 1/1] Target boot image: out/target/product/rk3568_r/boot.img

#### build completed successfully (4 seconds) ####

tgg@tgg-virtual-machine:~/rk3568_android_sdk$
```

Figure 4.5.3.6 Generate boot.img

It is necessary to pre-compile the kernel source code to obtain the kernel image Image, kernel DTB, and resource.img.

2. Independently compile and generate odm.img

Execute the following command to generate odm.img:

```
make odmimage -j16
```

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ make odmimage -j16
============================================
PLATFORM_VERSION_CODENAME=REL
PLATFORM_VERSION=11
TARGET_PRODUCT=rk3568_r
TARGET_BUILD_VARIANT=userdebug
TARGET_BUILD_TYPE=release
TARGET_ARCH=arm64
TARGET_ARCH_VARIANT=armv8-a
TARGET_CPU_VARIANT=cortex-a55
TARGET_2ND_ARCH=arm
TARGET_2ND_ARCH_VARIANT=armv8-2a
TARGET_2ND_CPU_VARIANT=cortex-a55
HOST_ARCH=x86_64
HOST_2ND_ARCH=x86
HOST_OS=linux
HOST_OS_EXTRA=Linux-5.15.0-53-generic-x86_64-Ubuntu-20.04.2-LTS
HOST_CROSS_OS=windows
HOST_CROSS_ARCH=x86
HOST_CROSS_2ND_ARCH=x86_64
HOST_BUILD_TYPE=release
BUILD_ID=RD2A.211001.002
OUT_DIR=out
============================================
21:19:24 Disallowed PATH tool "python3.8" used: []string{"python3.8", ".repo/repo/repo", "manifest", "-r",
21:19:24 See https://android.googlesource.com/platform/build/+/master/Changes.md#PATH_Tools for more infor
[100% 1/1] Target odm fs image: out/target/product/rk3568_r/odm.img

#### build completed successfully (4 seconds) ####
```

Figure 4.5.3.7 Generate odm.img

Instead of burning it onto the development board separately, it is packaged into super.img.

3. Independently compile to generate product.img

Execute the following command to generate product.img:

```
make productimage -j16
```

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ make productimage -j16
============================================
PLATFORM_VERSION_CODENAME=REL
PLATFORM_VERSION=11
TARGET_PRODUCT=rk3568_r
TARGET_BUILD_VARIANT=userdebug
TARGET_BUILD_TYPE=release
TARGET_ARCH=arm64
TARGET_ARCH_VARIANT=armv8-a
TARGET_CPU_VARIANT=cortex-a55
TARGET_2ND_ARCH=arm
TARGET_2ND_ARCH_VARIANT=armv8-2a
TARGET_2ND_CPU_VARIANT=cortex-a55
HOST_ARCH=x86_64
HOST_2ND_ARCH=x86
HOST_OS=linux
HOST_OS_EXTRA=Linux-5.15.0-53-generic-x86_64-Ubuntu-20.04.2-LTS
HOST_CROSS_OS=windows
HOST_CROSS_ARCH=x86
HOST_CROSS_2ND_ARCH=x86_64
HOST_BUILD_TYPE=release
BUILD_ID=RD2A.211001.002
OUT_DIR=out
============================================
21:31:44 Disallowed PATH tool "python3.8" used: []string{"python3.8", ".repo/repo/repo", "manifest", "-r
21:31:44 See https://android.googlesource.com/platform/build/+/master/Changes.md#PATH_Tools for more inf
[100% 1/1] Target product fs image: out/target/product/rk3568_r/product.img

#### build completed successfully (6 seconds) ####
```

Figure 4.5.3.8 Generate product.img

Instead of burning it onto the development board separately, it is packaged into super.img.

4. Independently compile to generate recovery.img

Execute the following command to generate recovery.img:

```
make recoveryimage -j16
```

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ make recoveryimage -j16
============================================
PLATFORM_VERSION_CODENAME=REL
PLATFORM_VERSION=11
TARGET_PRODUCT=rk3568_r
TARGET_BUILD_VARIANT=userdebug
TARGET_BUILD_TYPE=release
TARGET_ARCH=arm64
TARGET_ARCH_VARIANT=armv8-a
TARGET_CPU_VARIANT=cortex-a55
TARGET_2ND_ARCH=arm
TARGET_2ND_ARCH_VARIANT=armv8-2a
TARGET_2ND_CPU_VARIANT=cortex-a55
HOST_ARCH=x86_64
HOST_2ND_ARCH=x86
HOST_OS=linux
HOST_OS_EXTRA=Linux-5.15.0-53-generic-x86_64-Ubuntu-20.04.2-LTS
HOST_CROSS_OS=windows
HOST_CROSS_ARCH=x86
HOST_CROSS_2ND_ARCH=x86_64
HOST_BUILD_TYPE=release
BUILD_ID=RD2A.211001.002
OUT_DIR=out
============================================
15:07:18 Disallowed PATH tool "python3.8" used: []string{"python3.8", ".repo/repo/repo", "manifest", "-
15:07:18 See https://android.googlesource.com/platform/build/+/master/Changes.md#PATH_Tools for more in
[100% 1/1] build out/target/product/rk3568_r/recovery.img

#### build completed successfully (13 seconds) ####
```

Figure 4.5.3.9 Generate recovery.img

It is necessary to pre-compile the kernel source code to obtain the kernel image Image, kernel DTB, and resource.img.

5. Independently compile to generate system.img

Execute the following command to generate system.img:

```
make systemimage -j16
```

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ make systemimage -j16
============================================
PLATFORM_VERSION_CODENAME=REL
PLATFORM_VERSION=11
TARGET_PRODUCT=rk3568_r
TARGET_BUILD_VARIANT=userdebug
TARGET_BUILD_TYPE=release
TARGET_ARCH=arm64
TARGET_ARCH_VARIANT=armv8-a
TARGET_CPU_VARIANT=cortex-a55
TARGET_2ND_ARCH=arm
TARGET_2ND_ARCH_VARIANT=armv8-2a
TARGET_2ND_CPU_VARIANT=cortex-a55
HOST_ARCH=x86_64
HOST_2ND_ARCH=x86
HOST_OS=linux
HOST_OS_EXTRA=Linux-5.15.0-53-generic-x86_64-Ubuntu-20.04.2-LTS
HOST_CROSS_OS=windows
HOST_CROSS_ARCH=x86
HOST_CROSS_2ND_ARCH=x86_64
HOST_BUILD_TYPE=release
BUILD_ID=RD2A.211001.002
OUT_DIR=out
============================================
15:16:54 Disallowed PATH tool "python3.8" used: []string{"python3.8", ".repo/repo/repo", "manifest", "-r",
15:16:54 See https://android.googlesource.com/platform/build/+/master/Changes.md#PATH_Tools for more inforr
[100% 1/1] Install system fs image: out/target/product/rk3568_r/system.img

#### build completed successfully (6 seconds) ####
```

Figure 4.5.3.10 Generate system.img

Instead of burning it separately onto the development board, it is packaged into super.img.

6. Independently compile to generate vendor.img

Execute the following command to generate vendor.img:

```
make vendorimage -j16
```

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ make vendorimage -j16
============================================
PLATFORM_VERSION_CODENAME=REL
PLATFORM_VERSION=11
TARGET_PRODUCT=rk3568_r
TARGET_BUILD_VARIANT=userdebug
TARGET_BUILD_TYPE=release
TARGET_ARCH=arm64
TARGET_ARCH_VARIANT=armv8-a
TARGET_CPU_VARIANT=cortex-a55
TARGET_2ND_ARCH=arm
TARGET_2ND_ARCH_VARIANT=armv8-2a
TARGET_2ND_CPU_VARIANT=cortex-a55
HOST_ARCH=x86_64
HOST_2ND_ARCH=x86
HOST_OS=linux
HOST_OS_EXTRA=Linux-5.15.0-53-generic-x86_64-Ubuntu-20.04.2-LTS
HOST_CROSS_OS=windows
HOST_CROSS_ARCH=x86
HOST_CROSS_2ND_ARCH=x86_64
HOST_BUILD_TYPE=release
BUILD_ID=RD2A.211001.002
OUT_DIR=out
============================================
15:18:23 Disallowed PATH tool "python3.8" used: []string{"python3.8", ".repo/repo/repo", "manifest", "-r", "-
15:18:23 See https://android.googlesource.com/platform/build/+/master/Changes.md#PATH_Tools for more informat
[100% 303/303] Target vendor fs image: out/target/product/rk3568_r/vendor.img

#### build completed successfully (49 seconds) ####
```

Figure 4.5.3.11 Generate vendor.img

Instead of burning it separately onto the development board, it is packaged into super.img. The Linux kernel needs to be pre-compiled and the .ko modules need to be compiled as well, because these .ko modules will be included in vendor.img (such as the WiFi driver module).

7. Compile and generate super.img separately

Execute the following command to generate super.img:

```
make superimage -j16
```

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ make superimage -j16
============================================
PLATFORM_VERSION_CODENAME=REL
PLATFORM_VERSION=11
TARGET_PRODUCT=rk3568_r
TARGET_BUILD_VARIANT=userdebug
TARGET_BUILD_TYPE=release
TARGET_ARCH=arm64
TARGET_ARCH_VARIANT=armv8-a
TARGET_CPU_VARIANT=cortex-a55
TARGET_2ND_ARCH=arm
TARGET_2ND_ARCH_VARIANT=armv8-2a
TARGET_2ND_CPU_VARIANT=cortex-a55
HOST_ARCH=x86_64
HOST_2ND_ARCH=x86
HOST_OS=linux
HOST_OS_EXTRA=Linux-5.15.0-53-generic-x86_64-Ubuntu-20.04.2-LTS
HOST_CROSS_OS=windows
HOST_CROSS_ARCH=x86
HOST_CROSS_2ND_ARCH=x86_64
HOST_BUILD_TYPE=release
BUILD_ID=RD2A.211001.002
OUT_DIR=out
============================================
15:20:36 Disallowed PATH tool "python3.8" used: []string{"python3.8", ".repo/repo/repo", "manifest", "-r", "-o",
15:20:36 See https://android.googlesource.com/platform/build/+/master/Changes.md#PATH_Tools for more information.
[100% 91/91] Target super fs image for debug: out/target/product/rk3568_r/super.img
2023-04-25 15:21:24 - build_super_image.py - INFO    : Building super image from info dict...
2023-04-25 15:21:24 - sparse_img.py - INFO    : Total of 239401 4096-byte output blocks in 18 input chunks.
2023-04-25 15:21:24 - sparse_img.py - INFO    : Total of 30192 4096-byte output blocks in 3 input chunks.
2023-04-25 15:21:24 - sparse_img.py - INFO    : Total of 75087 4096-byte output blocks in 8 input chunks.
2023-04-25 15:21:24 - sparse_img.py - INFO    : Total of 52087 4096-byte output blocks in 5 input chunks.
2023-04-25 15:21:24 - sparse_img.py - INFO    : Total of 153 4096-byte output blocks in 2 input chunks.
2023-04-25 15:21:24 - common.py - INFO    :   Running: "lpmake --metadata-size 65536 --super-name super --metadat
tion system:readonly:980586496:rockchip_dynamic_partitions --image system=out/target/product/rk3568_r/system.img
ut/target/product/rk3568_r/system_ext.img --partition vendor:readonly:307556352:rockchip_dynamic_partitions --ima
ckchip_dynamic_partitions --image product=out/target/product/rk3568_r/product.img --partition odm:readonly:626688
tput out/target/product/rk3568_r/super.img"
2023-04-25 15:22:09 - common.py - INFO    : lpmake I 04-25 15:21:24  1097  1097 builder.cpp:1031] [liblp]Partitio
lpmake I 04-25 15:21:24  1097  1097 builder.cpp:1031] [liblp]Partition system_ext will resize from 0 bytes to 123
lpmake I 04-25 15:21:24  1097  1097 builder.cpp:1031] [liblp]Partition vendor will resize from 0 bytes to 3075563
lpmake I 04-25 15:21:24  1097  1097 builder.cpp:1031] [liblp]Partition product will resize from 0 bytes to 213348
lpmake I 04-25 15:21:24  1097  1097 builder.cpp:1031] [liblp]Partition odm will resize from 0 bytes to 626688 byt
2023-04-25 15:22:09 - build_super_image.py - INFO    : Done writing image out/target/product/rk3568_r/super.img
```

Figure 4.5.3.12 Generate super.img

super.img is formed by merging odm.img, vendor.img, system.img, system_ext.img and product.img. Eventually, super.img will be burned into the super partition of the development board.

4.5.4 Package into update.img image

Using the build.sh script, the image in the **<SDK>/rockdev/Image-ProductName**/ directory can be packaged into an update.img. The build.sh script will call the packaging tool for packaging, and the packaging tool is provided by RK. Packaging all images into one update.img is more convenient for users to burn, update and upgrade.

Execute the following command in the top-level directory of the SDK to package the image in the **<SDK>/rockdev/Image-ProductName/** directory into an update.img:

```
./build.sh -u
```

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ ./build.sh -u
will build update.img
-------------------KERNEL_DTS:rk3568-atk-evb1-ddr4-v10

============================================
PLATFORM_VERSION_CODENAME=REL
PLATFORM_VERSION=11
TARGET_PRODUCT=rk3568_r
TARGET_BUILD_VARIANT=userdebug
TARGET_BUILD_TYPE=release
TARGET_ARCH=arm64
TARGET_ARCH_VARIANT=armv8-a
TARGET_CPU_VARIANT=cortex-a55
TARGET_2ND_ARCH=arm
TARGET_2ND_ARCH_VARIANT=armv8-2a
TARGET_2ND_CPU_VARIANT=cortex-a55
HOST_ARCH=x86_64
HOST_2ND_ARCH=x86
HOST_OS=linux
HOST_OS_EXTRA=Linux-5.15.0-53-generic-x86_64-Ubuntu-20.04.2-LTS
HOST_CROSS_OS=windows
HOST_CROSS_ARCH=x86
HOST_CROSS_2ND_ARCH=x86_64
HOST_BUILD_TYPE=release
BUILD_ID=RD2A.211001.002
OUT_DIR=out
============================================
package resoure.img with charger images

Pack ./tools/images/ & ../kernel/resource.img to resource.img ...
Unpacking old image(../kernel/resource.img):
rk-kernel.dtb rk3568-atk-evb1-mipi-dsi-1080p#_saradc_ch2=341.dtb rk3568-atk-evb1-mipi-dsi-720p#_saradc_c
nel.bmp battery_0.bmp battery_1.bmp 12
Pack to resource.img successed!

Packed resources:
rk-kernel.dtb rk3568-atk-evb1-mipi-dsi-1080p#_saradc_ch2=341.dtb rk3568-atk-evb1-mipi-dsi-720p#_saradc_c
nel.bmp battery_0.bmp battery_1.bmp 12
```

```
------ PACKAGE ------
Add file: ./package-file
package-file,Add file: ./package-file done,offset=0x800,size=0x2ba,userspace=0x1
Add file: ./Image/MiniLoaderAll.bin
bootloader,Add file: ./Image/MiniLoaderAll.bin done,offset=0x1000,size=0x719c0,userspace=0xe4
Add file: ./Image/parameter.txt
parameter,Add file: ./Image/parameter.txt done,offset=0x73000,size=0x282,userspace=0x1
Add file: ./Image/uboot.img
uboot,Add file: ./Image/uboot.img done,offset=0x73800,size=0x400000,userspace=0x800
Add file: ./Image/misc.img
misc,Add file: ./Image/misc.img done,offset=0x473800,size=0xc000,userspace=0x18
Add file: ./Image/boot.img
boot,Add file: ./Image/boot.img done,offset=0x47f800,size=0x1ff4800,userspace=0x3fe9
Add file: ./Image/dtbo.img
dtbo,Add file: ./Image/dtbo.img done,offset=0x2474000,size=0x26f,userspace=0x1
Add file: ./Image/vbmeta.img
vbmeta,Add file: ./Image/vbmeta.img done,offset=0x2474800,size=0x1000,userspace=0x2
Add file: ./Image/recovery.img
recovery,Add file: ./Image/recovery.img done,offset=0x2475800,size=0x41fc000,userspace=0x83f8
Add file: ./Image/baseparameter.img
baseparameter,Add file: ./Image/baseparameter.img done,offset=0x6671800,size=0x100000,userspace=0x200
Add file: ./Image/super.img
super,Add file: ./Image/super.img done,offset=0x6771800,size=0x6003ce38,userspace=0xc007a
Add CRC...
Make firmware OK!
------ OK ------
********rkImageMaker ver 2.1********
Generating new image, please wait...
Writing head info...
Writing boot file...
Writing firmware...
Generating MD5 data...
MD5 data generated successfully!
New image generated successfully!
Making update.img OK.
Make update image ok!
/home/tgg/rk3568_android_sdk
```

Figure 4.5.4.1 Generate update.img

After the command is executed successfully, an update.img image will be generated in the **<SDK>/rockdev/Image-Product Name**/ directory, as shown below:

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk$
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ ls -l rockdev/Image-rk3568_r/update.img
-rw-rw-r-- 1 tgg tgg 1719796298 4月  25 15:42 rockdev/Image-rk3568_r/update.img
tgg@tgg-virtual-machine:~/rk3568_android_sdk$
```

Figure 4.5.4.2 update.img image

When using the burning tool to burn the image, users only need to provide the update.img, which is very convenient!

4.6 SDK cleanup

The build.sh script does not have a cleanup function. Users need to perform the cleanup manually, including Android cleanup, U-Boot cleanup, and Linux kernel cleanup. Execute the following command to clean U-Boot (enter the U-Boot source code directory):

```
cd u-boot/
make distclean     #Or make clean
```

Execute the following command to clean the Linux kernel (please enter the kernel source code directory):

```
cd kernel/
make distclean     #Or make clean
```

Execute the following command in the top-level directory of the SDK to clean the Android source code (**Be cautious!!**) :

```
make clean
```

Executing "make clean" will completely clean up everything, so use it with caution! Actually, you can also execute the command "make clean-<module>" to clean a specific module. We will introduce this later.

Chapter 5. SDK image burning

This chapter provides users with instructions on how to burn and run the compiled image files (such as uboot.img, boot.img, dtbo.img, misc.img, recovery.img, super.img, etc.) on the ATK-DLRK3568 development board.

The Rockchip platform offers various methods for image burning. These methods were introduced to you in Section 2.9. For example, image burning can be done using the RKDevTool tool under Windows, through an SD card, through the FactoryTool tool for batch burning (a production line burning tool that supports USB multi-device burning), and through the Linux_Upgrade_Tool tool under the Ubuntu system, etc. In summary, there are many ways to burn images. Users can choose the appropriate burning method to burn the image file onto the development board.

5.1 Flash Mode Introduction

In Section 2.9.1, the issues related to the flash mode of the development board have been elaborated in detail. If you still have any doubts about this, please proceed to read in that section!

5.2 Flashing on Windows System

On Windows, the image can be flashed using the RKDevTool provided by Rockchip. Before flashing, first copy the image file obtained by compiling the Android SDK in Chapter 4 (<SDK>/rockdev/Image-rk3568_r/ directory) from the Ubuntu system to the Windows system. For example, copy it to the rk3568_images directory on the Windows desktop, including baseparameter.img, boot.img, dtbo.img, MiniLoaderAll.bin, misc.img, parameter.txt, recovery.img, super.img, uboot.img, and vbmeta.img. As shown below:

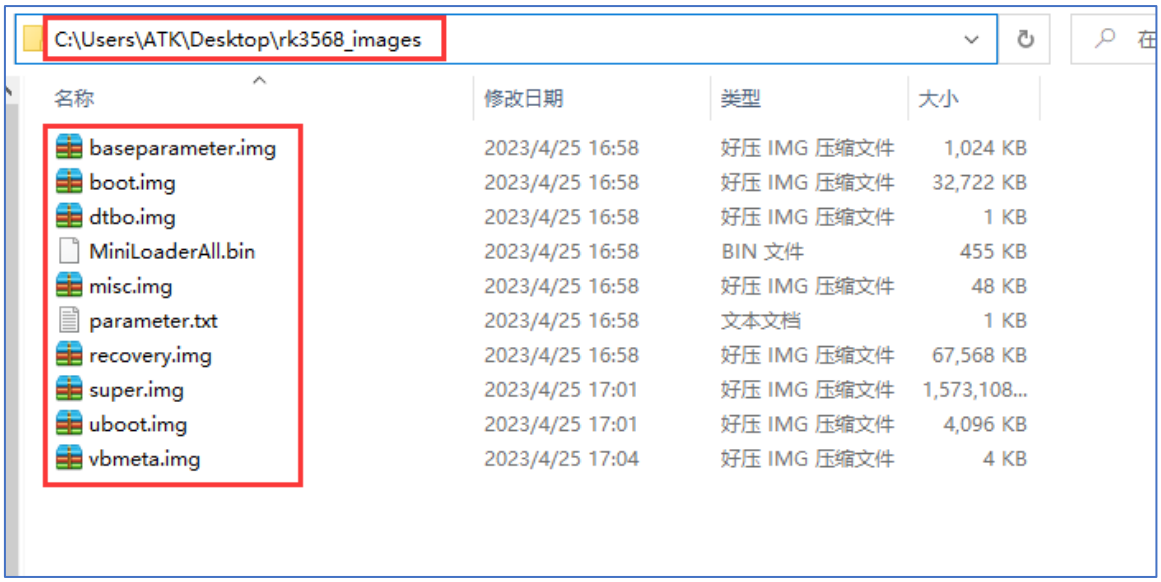


Figure 5.2-1 The images in the "rk3568_images" directory

Open the development tools of Rockchip Microelectronics:

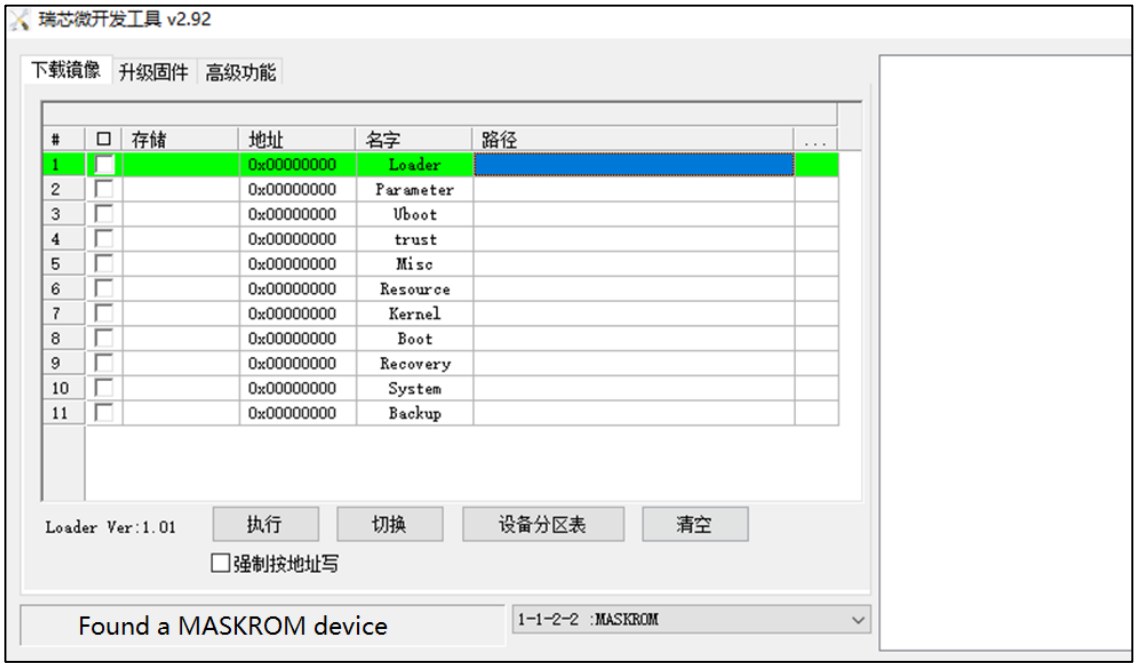


Figure 5.2-2 Ricoh Micro Development Tools

5.2.1 Introduction to Partition Table parameter.txt

In the 2.9.2 section, the user was introduced to how to use the Ricoh Micro Development Tools (RKDevTool) to burn the image, by directly importing a.cfg configuration file and then clicking the "**Execute**" button to perform the burning process; no user configuration is required throughout the process. However, for developers, they definitely need to know how to configure these information to facilitate their own development and debugging.

This section will explain the partition table file parameter.txt. This file is a txt text file. As previously mentioned, parameter.txt is the partition table file for the RK3568 platform, and this file describes the partition table of the development board and the starting and ending addresses of each partition, etc. Let's take a look at its content:

```
FIRMWARE_VER: 11.0
MACHINE_MODEL: rk3568_r
MACHINE_ID: 007
MANUFACTURER: rockchip
MAGIC: 0x5041524B
ATAG: 0x00200800
MACHINE: rk3568_r
CHECK_MASK: 0x80
PWR_HLD: 0,0,A,0,1
TYPE: GPT
CMDLINE:mtdparts=rk29xxnand:0x00002000@0x00002000(security),0x00002000@0x00004000(uboot),0x00002000@0x00006000(trust),0x00002000@0x00008000(misc),0x00002000@0x0000a000(dtbo),0x00000800@0x0000c000(vbmeta),0x00014000@0x0000c800(boot),0x00030000@0x00020800(recovery),0x000c0000@0x00050800(backup),0x000c0000@0x00110800(cache),0x00008000
```

@0x001d0800(metadata),0x00000800@0x001d8800(baseparameter),0x00614000@0x001d9000(super),-@0x007ed000(userdata:grow)

RK has provided the corresponding documentation to introduce various identifiers in the parameter.txt file to users. This document can be found in the SDK, and its path is: **<SDK>/RKDocs/common/NVM/Rockchip_Introduction_Partition_CN.pdf.** Here, we are only presenting the partition information defined by **mtdparts** to you.

The information defined by mtdparts is as follows:

rk29xxnand:0x00002000@0x00002000(security),0x00002000@0x00004000(uboot),0x00002000@0x00006000(trust),0x00002000@0x00008000(misc),0x00002000@0x0000a000(dtbo),0x00000800@0x0000c000(vbmeta),0x00014000@0x0000c800(boot),0x00030000@0x00020800(recovery),0x000c0000@0x00050800(backup),0x000c0000@0x00110800(cache),0x00008000@0x001d0800(metadata),0x00000800@0x001d8800(baseparameter),0x00614000@0x001d9000(super),-@0x007ed000(userdata:grow)

rk29xxnand is a designation. For compatibility, the Rockchip platform always uses rk29xxnand as the designation.

Information such as 0x00002000@0x00002000 (security), 0x00002000@0x00004000 (uboot), 0x00002000@0x00008000 (misc), etc. are used to define partitions. The value before the @ symbol is the partition size, the value after the @ symbol is the starting position of the partition, and the characters in the parentheses are the name of the partition. All values are in sectors (sectors), and 1 sector is 512 bytes. Therefore, it can be known that the starting position of the uboot partition is at the 0x4000 sectors position, with a size of 0x2000 sectors (4MB); the starting position of the misc partition is at the 0x8000 sectors position, and the size is also 0x2000 sectors.

For performance, the starting address of each partition needs to be aligned by 32KB (64 sectors), and the size also needs to be an integer multiple of 32KB.

The last partition needs to specify the grow parameter, indicating that all the remaining storage space is allocated to this partition:

-@0x007ed000(userdata:grow)

Although the userdata partition has specified the starting position, it has not specified the partition size. Instead, a "-" is used to represent it, and then "**:grow**" is added after the partition description. This indicates that all the remaining space will be allocated to this partition.

The functions of each partition are as shown in the table below:

Partition name	Explanation
security	Security partition is used to store encrypted secure data, required for AVB secure boot
uboot	The uboot.img image will be burned into this partition, and the uboot.img image will be burned onto this partition
trust	The trust.img image will be burned into this partition. The RK3568 does not use it. Its Trust image has been packaged into the uboot.img
misc	The misc.img image will be burned into this partition. The misc partition is a very important partition, mainly used for communication between Android, U-Boot and recovery
dtbo	The dtbo.img image will be burned into this partition

vbmeta	The vbmeta.img image will be burned into this partition, used for AVB verification
boot	The boot.img image will be burned into this partition
recovery	The recovery.img image will be burned into this partition, used to enter the recovery mode
backup	Backup partition, when the device fails to boot into the Android system, users can guide the device to enter recovery mode. Select the "Recovery System" option in the recovery menu to perform system recovery operations. This function will use the backup partition
cache	Buffer partition, during the system upgrade process, the /cache partition will be used. The system upgrade package will be downloaded to this partition, mainly used for buffering the system upgrade OTA package /cache/recovery/command /cache/recovery/command /cache/recovery/log /cache/recovery/intent
metadata	This partition is used to store the metadata encryption key when the device uses metadata encryption.
baseparameter	The Baseparameter.img image will be burned onto this partition.
super	The super.img will also be burned onto this partition. Since Android 10, the Android system supports dynamic partitions (dynamic partition), which combines multiple system read-only partitions (including system, product, vendor, odm, or other manufacturer-defined partitions) into one super partition.
userdata	The partition for user use, this partition contains the applications and data installed by the user, including custom data.

Table 5.2.1.1 Introduction of Each Partition's Function

5.2.2 Configuration

From the parameter.txt partition table, we know that a total of 14 partitions have been defined, along with the starting position and size of each partition. Next, we need to manually configure the Rockchip development tools. After the configuration is completed, it will be as shown in the following figure:

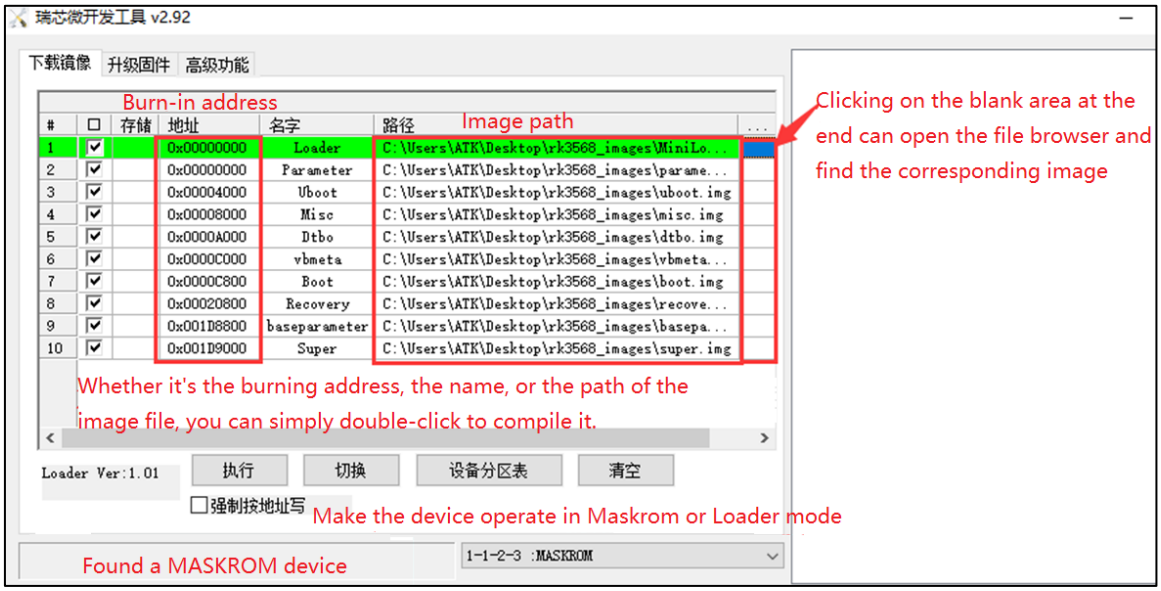


Figure 5.2.2.1 Manually configure the Rockchip development tools

Here are a few points to note:

1. The unit of the address values in the Rockchip development tools is also sectors (a sector equals 512 bytes);

2. The first item corresponds to the MiniLoaderAll.bin image. **Its address does not need to be configured; simply use 0x0. Because the MiniLoaderAll.bin image has a dedicated burning address, no user configuration is required.** Moreover, its name is usually Loader (or in lowercase, loader), and do not modify it;

3. In the above picture, the second item corresponds to the partition table parameter. **Similarly, its address does not need to be configured; simply use 0x0. Because the parameter.txt file will not be burned to the Flash, but the partition information defined in this file will be extracted;** also, its name is set as parameter (or in uppercase, Parameter) and do not modify it, because the underlying system needs to identify the partition table file through this "parameter" name.

4. Apart from **MiniLoaderAll.bin** and **parameter.txt** which are slightly special, the other images can be configured directly based on the starting positions defined in the partition table. The names should be as close as possible to the partition names specified in the parameter.txt file.

After the configuration is completed, we can also export and save these configuration information to a.cfg file for easy import next time. The export method is quite simple. The steps are as shown in Figures 5.2.2.2 to 5.2.2.3:

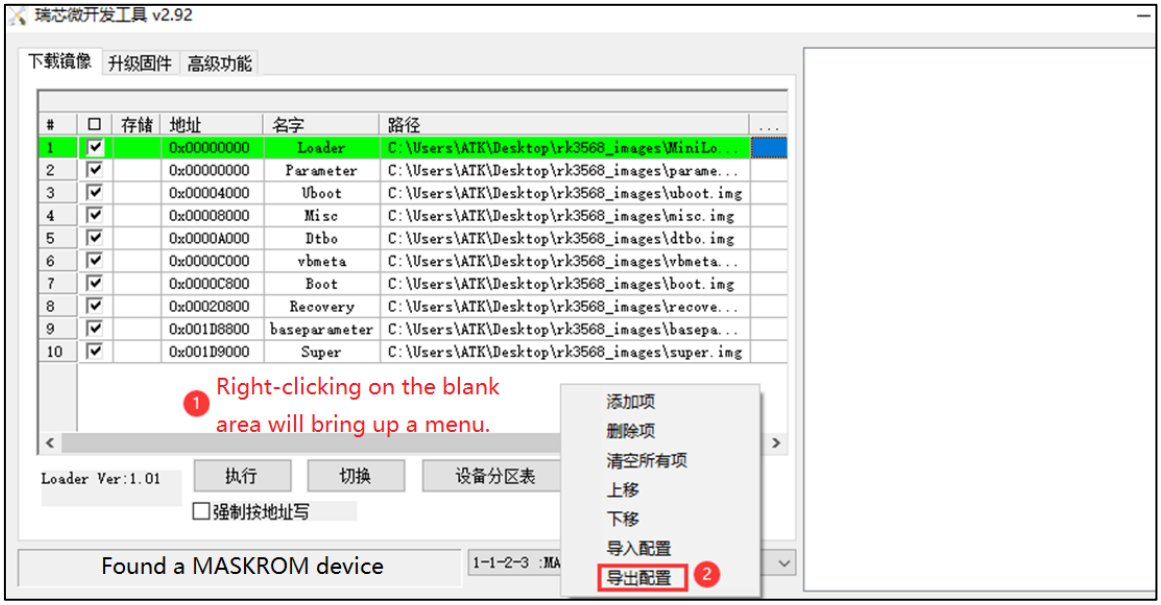


Figure 5.2.2.2 Export configuration (1)

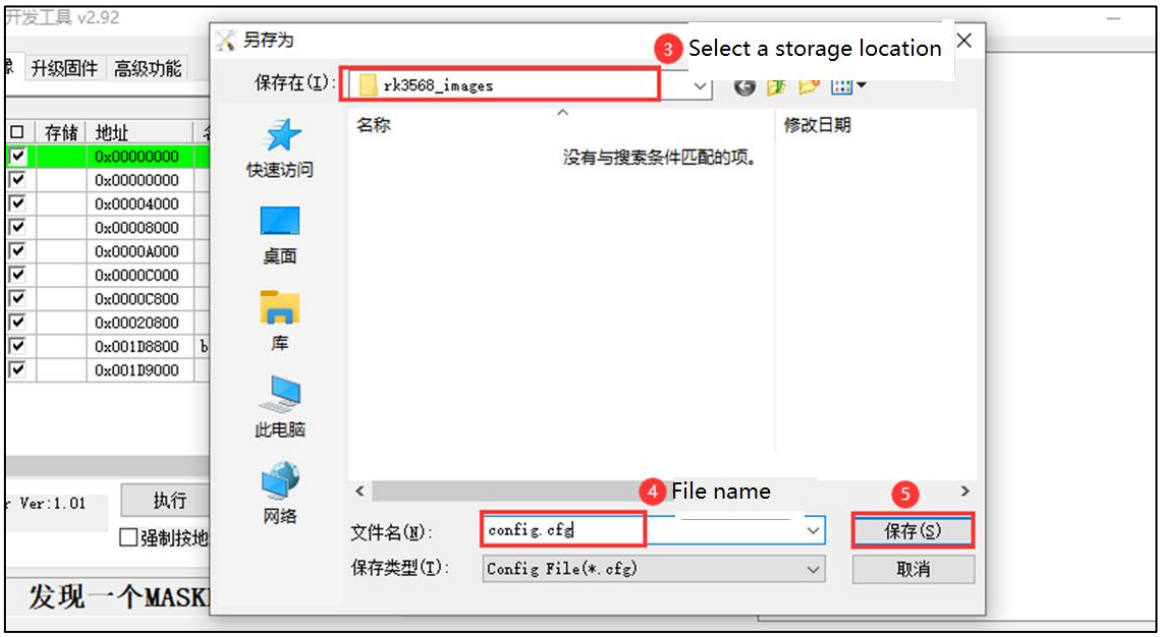


Figure 5.2.2.3 Export configuration (2)

Export and save the configuration information to the config.cfg file, which makes it convenient to directly import this configuration information in the next time, without having to configure manually each time.

5.2.3 Flashing

First, connect the power adapter and OTG to the development board. The specific details can be referred to as shown in Figure 2.9.2.10. No need to repeat here! Before flashing, make the development board enter the Maskrom or Loader mode. The method for the development board to enter the Maskrom or Loader mode has been explained in the previous chapters.

For example, by using the Maskrom mode to write the image, press the **UPDATE** button on the development board, then power on or reset the development board. At this point, the device will enter the Maskrom mode (the Rockchip development tools will prompt the user "**A MASKROM device has been detected**"), and then click the "**Execute**" button to burn the image:

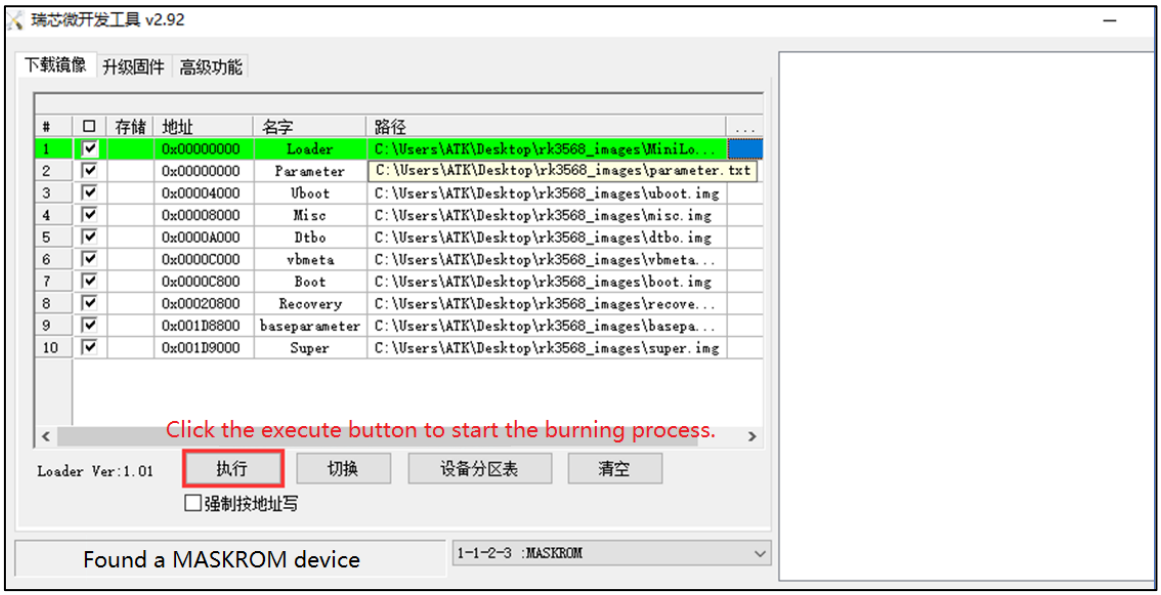


Figure 5.2.3.1 Perform the burn operation

During the burn process, log information will be output in the blank area on the right, as follows:

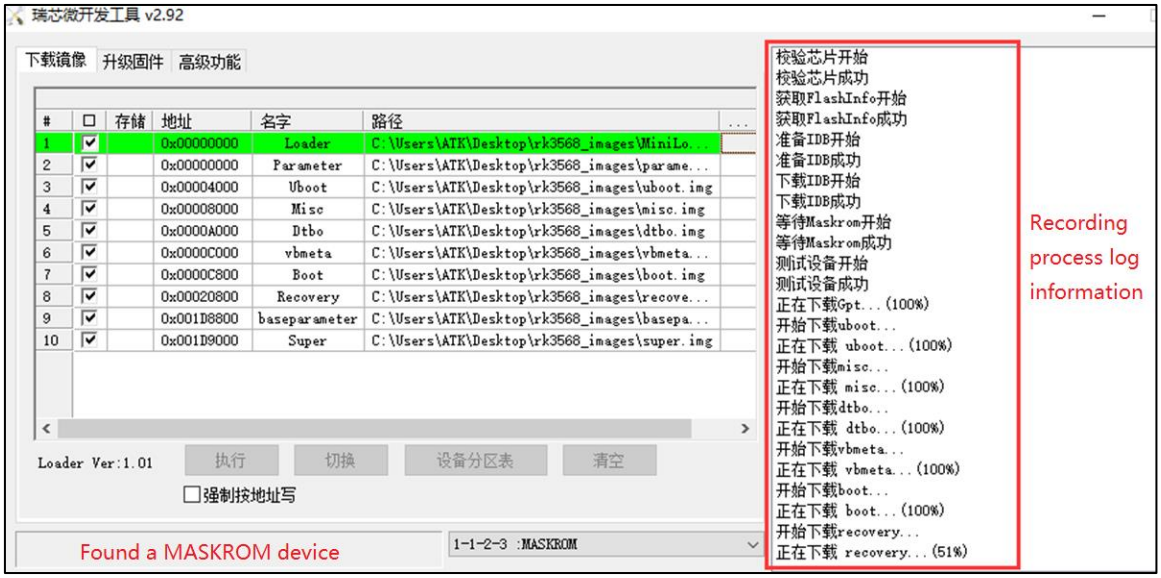
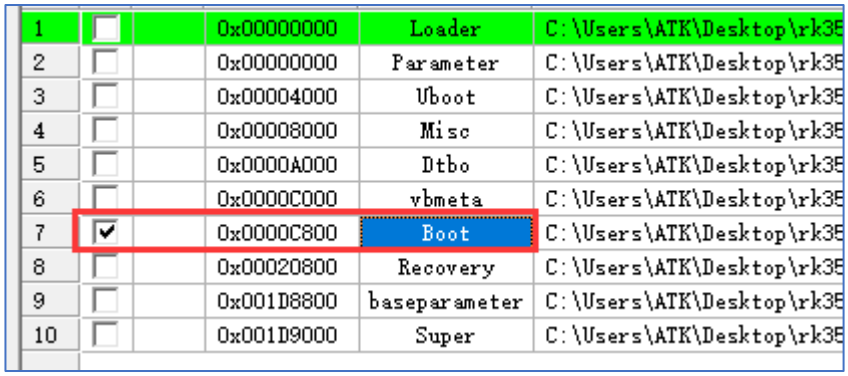


Figure 5.2.3.2 Recording process log information

It is also possible to burn a specific partition separately. For example, you can burn the "boot.img" file to the "boot" partition alone. Just check the corresponding option as shown below:



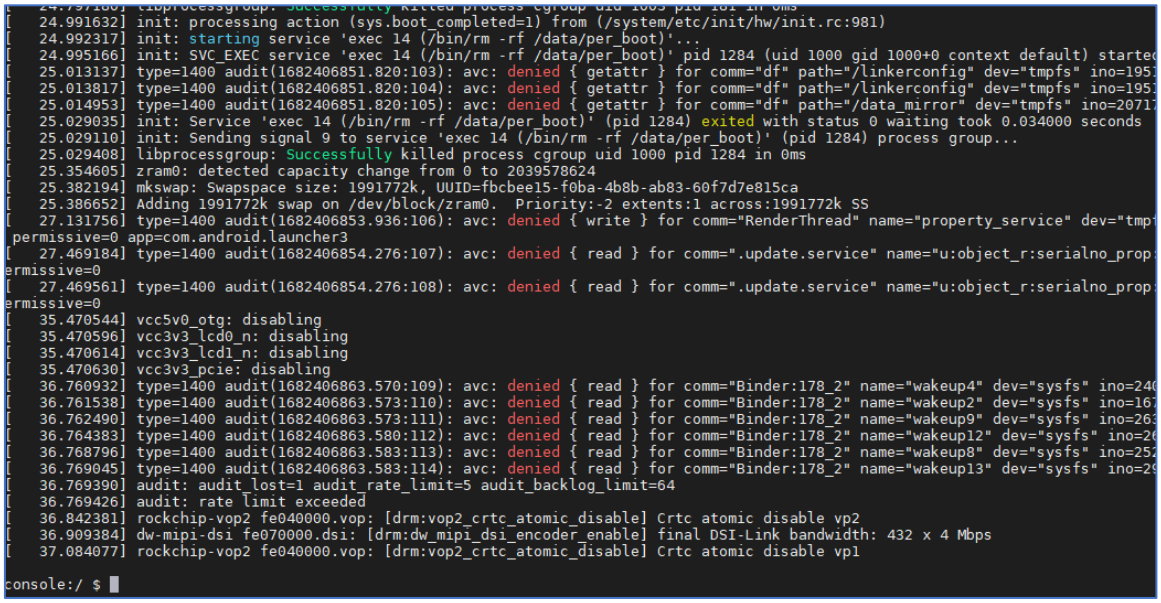
1	☐	0x00000000	Loader	C:\Users\ATK\Desktop\rk35
2	☐	0x00000000	Parameter	C:\Users\ATK\Desktop\rk35
3	☐	0x00004000	Uboot	C:\Users\ATK\Desktop\rk35
4	☐	0x00008000	Misc	C:\Users\ATK\Desktop\rk35
5	☐	0x0000A000	Dtbo	C:\Users\ATK\Desktop\rk35
6	☐	0x0000C000	vbmeta	C:\Users\ATK\Desktop\rk35
7	☑	0x0000C800	Boot	C:\Users\ATK\Desktop\rk35
8	☐	0x00020800	Recovery	C:\Users\ATK\Desktop\rk35
9	☐	0x001D8800	baseparameter	C:\Users\ATK\Desktop\rk35
10	☐	0x001D9000	Super	C:\Users\ATK\Desktop\rk35

Figure 5.2.3.3 Burn the boot.img file separately.

Then click the "**Execute**" button to perform the burning process. This is very useful during the development and debugging process, because sometimes we only need to update a certain partition and simply burn the image to that partition to replace the old image, without having to re-burn the entire system.

5.2.4 Booting the System

After the burning is completed, the system will automatically start and enter the system. The serial port terminal will output the following information:



```
[   24.991632] init: processing action (sys.boot_completed=1) from (/system/etc/init/hw/init.rc:981)
[   24.992317] init: starting service 'exec 14 (/bin/rm -rf /data/per_boot)'...
[   24.995166] init: SVC_EXEC service 'exec 14 (/bin/rm -rf /data/per_boot)' pid 1284 (uid 1000 gid 1000+0 context default) started
[   25.013137] type=1400 audit(1682406851.820:103): avc: denied { getattr } for comm="df" path="/linkerconfig" dev="tmpfs" ino=1951
[   25.013817] type=1400 audit(1682406851.820:104): avc: denied { getattr } for comm="df" path="/linkerconfig" dev="tmpfs" ino=1951
[   25.014953] type=1400 audit(1682406851.820:105): avc: denied { getattr } for comm="df" path="/data_mirror" dev="tmpfs" ino=2071
[   25.029035] init: Service 'exec 14 (/bin/rm -rf /data/per_boot)' (pid 1284) exited with status 0 waiting took 0.034000 seconds
[   25.029110] init: Sending signal 9 to service 'exec 14 (/bin/rm -rf /data/per_boot)' (pid 1284) process group...
[   25.029408] libprocessgroup: Successfully killed process cgroup uid 1000 pid 1284 in 0ms
[   25.354605] zram0: detected capacity change from 0 to 2039578624
[   25.382194] mkswap: Swapspace size: 1991772k, UUID=fbcbee15-f0ba-4b8b-ab83-60f7d7e815ca
[   25.386652] Adding 1991772k swap on /dev/block/zram0.  Priority:-2 extents:1 across:1991772k SS
[   27.131756] type=1400 audit(1682406853.936:106): avc: denied { write } for comm="RenderThread" name="property_service" dev="tmpf
permissive=0 app=com.android.launcher3
[   27.469184] type=1400 audit(1682406854.276:107): avc: denied { read } for comm=".update.service" name="u:object_r:serialno_prop
ermissive=0
[   27.469561] type=1400 audit(1682406854.276:108): avc: denied { read } for comm=".update.service" name="u:object_r:serialno_prop
ermissive=0
[   35.470544] vcc5v0_otg: disabling
[   35.470596] vcc3v3_lcd0_n: disabling
[   35.470614] vcc3v3_lcd1_n: disabling
[   35.470630] vcc3v3_pcie: disabling
[   36.760932] type=1400 audit(1682406863.570:109): avc: denied { read } for comm="Binder:178_2" name="wakeup4" dev="sysfs" ino=240
[   36.761538] type=1400 audit(1682406863.573:110): avc: denied { read } for comm="Binder:178_2" name="wakeup2" dev="sysfs" ino=167
[   36.762490] type=1400 audit(1682406863.573:111): avc: denied { read } for comm="Binder:178_2" name="wakeup9" dev="sysfs" ino=263
[   36.764383] type=1400 audit(1682406863.580:112): avc: denied { read } for comm="Binder:178_2" name="wakeup12" dev="sysfs" ino=26
[   36.768796] type=1400 audit(1682406863.583:113): avc: denied { read } for comm="Binder:178_2" name="wakeup8" dev="sysfs" ino=252
[   36.769045] type=1400 audit(1682406863.583:114): avc: denied { read } for comm="Binder:178_2" name="wakeup13" dev="sysfs" ino=29
[   36.769390] audit: audit_lost=1 audit_rate_limit=5 audit_backlog_limit=64
[   36.769426] audit: rate limit exceeded
[   36.842381] rockchip-vop2 fe040000.vop: [drm:vop2_crtc_atomic_disable] Crtc atomic disable vp2
[   36.909384] dw-mipi-dsi fe070000.dsi: [drm:dw_mipi_dsi_encoder_enable] final DSI-Link bandwidth: 432 x 4 Mbps
[   37.084077] rockchip-vop2 fe040000.vop: [drm:vop2_crtc_atomic_disable] Crtc atomic disable vp1

console:/ $
```

Figure 5.2.4.1 Enter the system

5.2.5 Flashing update.img

You can use the Renesas Microelectronics development tools to flash the update.img package. First, copy the update.img generated in Section 4.5.4 to the Windows environment, then use the Renesas Microelectronics development tools to flash it onto the development board. The flashing method can be referred to in Section 2.9.3.

5.3 Flashing in Ubuntu system

This section introduces to users how to flash the image in Ubuntu. In Windows, flashing requires first copying the image from the Ubuntu system to the Windows system, so that it can be flashed. If flashing in Ubuntu, then there is no need to copy the image.

As we have mentioned earlier, in Ubuntu, the Linux_Upgrade_Tool tool can be used for flashing, which is in the SDK, and the path is **<SDK>/RKTools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool_v2.4.zip**, as shown below:

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ cd RKTools/linux/Linux_Upgrade_Tool/
tgg@tgg-virtual-machine:~/rk3568_android_sdk/RKTools/linux/Linux_Upgrade_Tool$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/RKTools/linux/Linux_Upgrade_Tool$ ls
Linux_Upgrade_Tool_v2.4.zip
tgg@tgg-virtual-machine:~/rk3568_android_sdk/RKTools/linux/Linux_Upgrade_Tool$
```

Figure 5.3-1 Linux_Upgrade_Tool directory

First, extract the Linux_Upgrade_Tool_v2.4.zip compressed file to the current directory, and then execute the following command:

```
unzip -O GBK Linux_Upgrade_Tool_v2.4.zip
```

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/RKTools/linux/Linux_Upgrade_Tool$ ls
Linux_Upgrade_Tool_v2.4.zip
tgg@tgg-virtual-machine:~/rk3568_android_sdk/RKTools/linux/Linux_Upgrade_Tool$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/RKTools/linux/Linux_Upgrade_Tool$ unzip -O GBK Linux_Upgrade_Tool_v2.4.zip
Archive:  Linux_Upgrade_Tool_v2.4.zip
   creating: Linux_Upgrade_Tool_v2.4/
  inflating: Linux_Upgrade_Tool_v2.4/revision.txt
  inflating: Linux_Upgrade_Tool_v2.4/upgrade_tool
  inflating: Linux_Upgrade_Tool_v2.4/命令行开发工具使用文档.pdf
tgg@tgg-virtual-machine:~/rk3568_android_sdk/RKTools/linux/Linux_Upgrade_Tool$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/RKTools/linux/Linux_Upgrade_Tool$ cd Linux_Upgrade_Tool_v2.4/
tgg@tgg-virtual-machine:~/rk3568_android_sdk/RKTools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool_v2.4$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/RKTools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool_v2.4$ ls
命令行开发工具使用文档.pdf  revision.txt  upgrade_tool
tgg@tgg-virtual-machine:~/rk3568_android_sdk/RKTools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool_v2.4$
```

Figure 5.3-2 Unzip the file Linux_Upgrade_Tool_v2.4.zip

After executing the unzip command, a folder named Linux_Upgrade_Tool_v2.4 is generated. Inside this folder, there is a user manual provided by RK, which provides detailed instructions on how to use the upgrade_tool tool. Therefore, for the detailed usage method, please refer to this document: "**Command Line Development Tool Usage Document.pdf**".

We need to add the executable permission to upgrade_tool. Execute the following command:

```
chmod a+x upgrade_tool
```

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/RKTools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool_v2.4$ ls
命令行开发工具使用文档.pdf  revision.txt  upgrade_tool
tgg@tgg-virtual-machine:~/rk3568_android_sdk/RKTools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool_v2.4$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/RKTools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool_v2.4$ chmod a+x upgrade_tool
tgg@tgg-virtual-machine:~/rk3568_android_sdk/RKTools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool_v2.4$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/RKTools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool_v2.4$ ls -l upgrade_tool
-rwxrwxr-x 1 tgg tgg 3027128 3月  21  2022 upgrade_tool
tgg@tgg-virtual-machine:~/rk3568_android_sdk/RKTools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool_v2.4$
```

Figure 5.3-3 Grant execute permission to the upgrade_tool

The upgrade_tool has two running modes: **command line mode** and **tool mode**. Simply running the upgrade_tool command without any parameters will enter the tool mode. The tool mode is actually an interactive mode. In this mode, users can input different instructions and press enter to execute them. Then, the corresponding operation corresponding to the instruction will be triggered, such as downloading the image, erasing the image, reading device information, etc.

When running the upgrade_tool command with parameters (such as UF, UL, DI, DB, etc.), it will enter the command line mode. The command line mode means that after running the upgrade_tool, there is no interaction with the user. When running upgrade_tool with parameters, these parameters are

used to tell the upgrade_tool what operation to perform this time, such as burning the image, erasing, etc. After completing the task, it will exit.

5.3.1 Connect the USB device to Ubuntu

Now, let's introduce how to use the upgrade_tool tool. First, connect the hardware (connect the power adapter and the OTG interface of the development board); since the image is burned in the Ubuntu system of the VMware virtual machine, the development board (USB device) needs to be connected to the Ubuntu system. Then, you can see a device named "**Fuzhou Rockchip Rockusb Device**" in the lower right corner of the Ubuntu system (it will be displayed only when the mouse is moved over the icon), as shown in the following figure:

Figure 5.3.1.1 Rockchip Rockusb Device

If your development board is running the Linux system, then this name becomes "Fuzhou Rockchip Android ADB interface", because at this time the development board exists as an ADB device, and it only becomes a Rockusb device in Loader or Maskrom mode.

Similarly, this device can also be found under "**Virtual Machine→ Mobile Device**", as shown in the following figure:

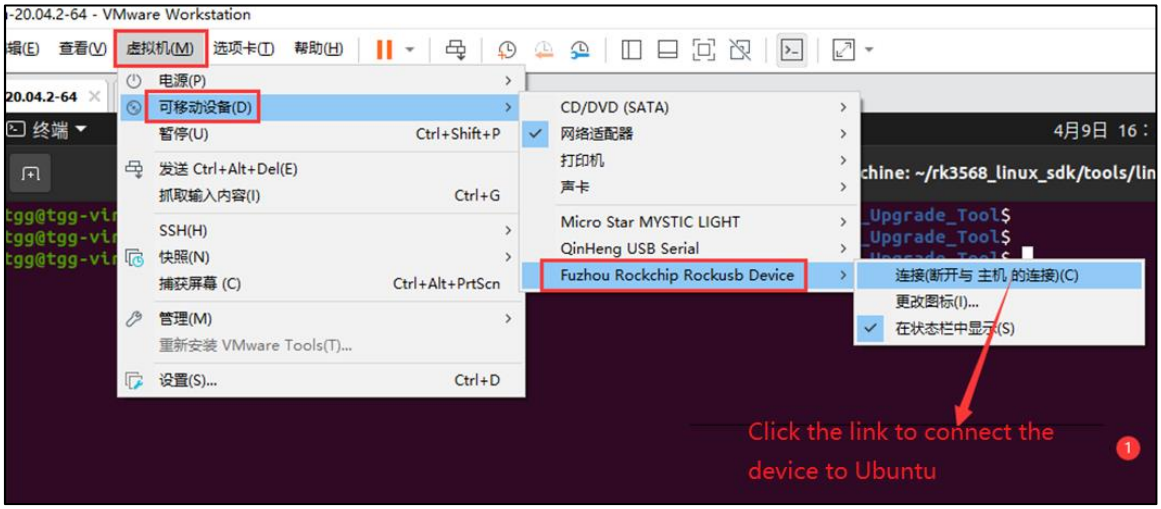


Figure 5.3.1.2 Connect the device to Ubuntu (1)

You can also right-click on the device icon at the bottom right corner and then select "**Connect**":

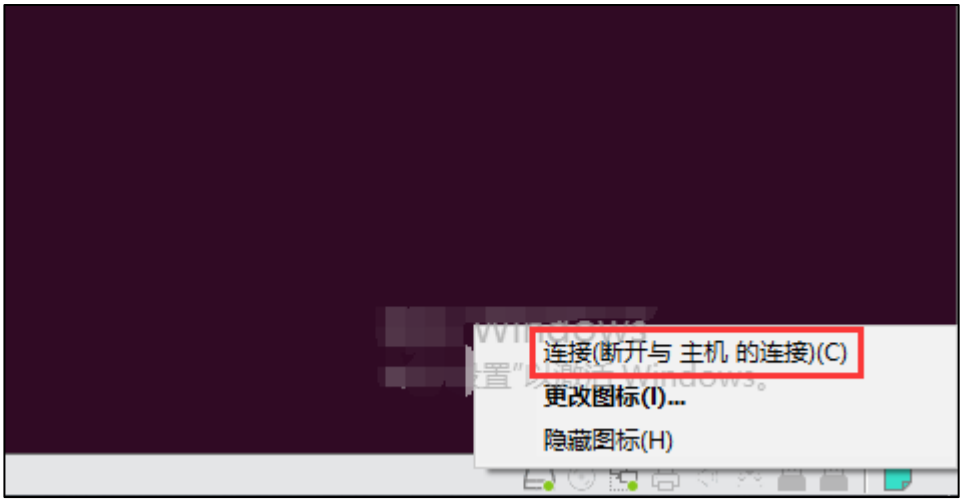


Figure 5.3.1.3 Connect the device to Ubuntu (2)

After performing this operation, the development board can be connected to Ubuntu.

5.3.2 Burning with the upgrade_tool tool

This section introduces to users how to burn images using the upgrade_tool. Here, only the command-line mode burning is covered. For ease of operation, we directly enter the <SDK>/rockdev/Image-rk3568_r/ directory (where the images are stored). Before burning, the development board needs to be in Maskrom or Loader mode.

The upgrade_tool supports many commands. Different commands perform different operations, such as CD, LD, SD, UF, DI, DB, TD, RD, etc. (**case-insensitive**). Some commands require parameters while others do not. The functions of these commands are all introduced in the usage documentation provided by RK.

To burn the MiniLoaderAll.bin image, use the UL command. To burn other images (uboot.img, boot.img, oem.img, userdata.img, rootfs.img, misc.img, etc.) and the partition table file parameter.txt, use the DI command.

First, burn the MiniLoaderAll.bin image. Use the UL command to burn the MiniLoaderAll.bin image (**when executing upgrade_tool, add sudo to obtain root user privileges; otherwise, the operation will fail!**). ：

```
sudo ../../RKTools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool_v2.4/upgrade_tool UL MiniLoaderAll.bin -noreset
```

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rockdev/Image-rk3568_r$ pwd
/home/tgg/rk3568_android_sdk/rockdev/Image-rk3568_r
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rockdev/Image-rk3568_r$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rockdev/Image-rk3568_r$ sudo ../../RKTools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool_v2.4/
upgrade_tool UL MiniLoaderAll.bin -noreset
Program Log will save in the /root/upgrade_tool/log/
Loading loader...
Support Type:RK3568     Loader ver:1.01 Loader Time:2023-04-24 12:06:23
Upgrade loader ok.
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rockdev/Image-rk3568_r$
```

Figure 5.3.2.1 Burn MiniLoaderAll.bin

The "upgrade_tool" command is followed by three parameters. The first parameter indicates the instruction to be executed, and the "UL" instruction is used for burning the Loader; the second parameter represents the path of the MiniLoaderAll.bin image; the third parameter -noreset indicates that the development board should not be reset (do not reset the device) after the Loader is burned.

Generally, MiniLoaderAll.bin is burned first, and then the parameter.txt partition table needs to be downloaded through the DI instruction next:

```
sudo ../../RKTools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool_v2.4/upgrade_tool DI -p parameter.txt
```

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rockdev/Image-rk3568_r$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rockdev/Image-rk3568_r$ sudo ../../RKTools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool_v2.4/
upgrade_tool DI -p parameter.txt
[sudo] tgg 的密码:
Program Log will save in the /root/upgrade_tool/log/
directlba=1,first4access=1,gpt=1
Write gpt...
Write gpt ok.
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rockdev/Image-rk3568_r$
```

Figure 5.3.2.2 Download the partition table

Next, burn the other images:

```
sudo ../../RKTools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool_v2.4/upgrade_tool DI -uboot uboot.img
sudo ../../RKTools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool_v2.4/upgrade_tool DI -misc misc.img
sudo ../../RKTools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool_v2.4/upgrade_tool DI -dtbo dtbo.img
sudo ../../RKTools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool_v2.4/upgrade_tool DI -vbmeta vbmeta.img
sudo ../../RKTools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool_v2.4/upgrade_tool DI -boot boot.img
sudo ../../RKTools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool_v2.4/upgrade_tool DI -recovery recovery.img
sudo ../../RKTools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool_v2.4/upgrade_tool DI -baseparameter baseparameter.img
sudo ../../RKTools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool_v2.4/upgrade_tool DI -super super.img
```

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rockdev/Image-rk3568_r$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rockdev/Image-rk3568_r$ sudo ../../RKTools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool_v2.4/
upgrade_tool DI -uboot uboot.img
Program Log will save in the /root/upgrade_tool/log/
directlba=1,first4access=1,gpt=1
Download uboot start...(0x00004000)
Download image ok.
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rockdev/Image-rk3568_r$
```

Figure 5.3.2.3 Burn uboot.img

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rockdev/Image-rk3568_r$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rockdev/Image-rk3568_r$ sudo ../../RKTools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool_v2.4/
upgrade_tool DI -misc misc.img
Program Log will save in the /root/upgrade_tool/log/
directlba=1,first4access=1,gpt=1
Download misc start...(0x00008000)
Download image ok.
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rockdev/Image-rk3568_r$
```

Figure 5.3.2.4 Burn misc.img

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rockdev/Image-rk3568_r$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rockdev/Image-rk3568_r$ sudo ../../RKTools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool_v2.4/
upgrade_tool DI -dtbo dtbo.img
Program Log will save in the /root/upgrade_tool/log/
directlba=1,first4access=1,gpt=1
Download dtbo start...(0x0000a000)
Download image ok.
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rockdev/Image-rk3568_r$
```

Figure 5.3.2.5 Burn dtbo.img

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rockdev/Image-rk3568_r$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rockdev/Image-rk3568_r$ sudo ../../RKTools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool_v2.4/
upgrade_tool DI -vbmeta vbmeta.img
Program Log will save in the /root/upgrade_tool/log/
directlba=1,first4access=1,gpt=1
Download vbmeta start...(0x0000c000)
Download image ok.
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rockdev/Image-rk3568_r$
```

Figure 5.3.2.6 Burn vbmeta.img

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rockdev/Image-rk3568_r$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rockdev/Image-rk3568_r$ sudo ../../RKTools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool_v2.4/
upgrade_tool DI -boot boot.img
Program Log will save in the /root/upgrade_tool/log/
directlba=1,first4access=1,gpt=1
Download boot start...(0x0000c800)
Download image ok.
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rockdev/Image-rk3568_r$
```

Figure 5.3.2.7 Burn boot.img

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rockdev/Image-rk3568_r$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rockdev/Image-rk3568_r$ sudo ../../RKTools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool_v2.4/
upgrade_tool DI -recovery recovery.img
Program Log will save in the /root/upgrade_tool/log/
directlba=1,first4access=1,gpt=1
Download recovery start...(0x00020800)
Download image ok.
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rockdev/Image-rk3568_r$
```

Figure 5.3.2.8 Burn recovery.img

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rockdev/Image-rk3568_r$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rockdev/Image-rk3568_r$ sudo ../../RKTools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool_v2.4/
upgrade_tool DI -baseparameter baseparameter.img
Program Log will save in the /root/upgrade_tool/log/
directlba=1,first4access=1,gpt=1
Download baseparameter start...(0x001d8800)
Download image ok.
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rockdev/Image-rk3568_r$
```

Figure 5.3.2.9 Burn baseparameter.img

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rockdev/Image-rk3568_r$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rockdev/Image-rk3568_r$ sudo ../../RKTools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool_v2.4/
upgrade_tool DI -super super.img
Program Log will save in the /root/upgrade_tool/log/
directlba=1,first4access=1,gpt=1
Download super start...(0x001d9000)
Download image ok.
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rockdev/Image-rk3568_r$
```

Figure 5.3.2.10 Burn super.img

In "DI -xxx", xxx represents the partition name. For example, -boot is the boot partition and -misc is the misc partition. Using the upgrade_tool tool to burn the image does not require the user to specify the burning address. The starting positions of each partition, which are the burning addresses of the image, have been defined in the parameter.txt partition table.

If the command execution fails, you can try resetting and restarting the development board, and then perform the operation again.

Finally, when all the images have been burned completely, we can execute the following command to reset the development board and start the system:

```
sudo ../../RKTools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool_v2.4/upgrade_tool RD
```

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rockdev/Image-rk3568_r$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rockdev/Image-rk3568_r$ sudo ../../RKTools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool_v2.4/
upgrade_tool RD
[sudo] tgg 的密码:
Program Log will save in the /root/upgrade_tool/log/
Reset Device OK.
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rockdev/Image-rk3568_r$
```

Figure 5.3.2.10 Reset the development board

5.3.3 Flashing update.img

This section explains how to use the upgrade_tool tool to flash the update.img image package. Similarly, the development board must be in Maskrom or Loader mode before flashing.

Execute the following command to flash the update.img image (using the UF command to flash update.img):

```
sudo ../../RKTools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool_v2.4/upgrade_tool UF update.img
```

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rockdev/Image-rk3568_r$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rockdev/Image-rk3568_r$ sudo ../../RKTools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool_v2.4/
upgrade_tool UF update.img
Program Log will save in the /root/upgrade_tool/log/
Loading firmware...
Support Type:RK3568     FW Ver:b.0.00   FW Time:2023-04-25 15:40:56
Loader ver:1.01 Loader Time:2023-04-24 12:06:23
Upgrade firmware ok.
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rockdev/Image-rk3568_r$
```

Figure 5.3.3.1 Burn "update.img"

After the burning process is completed, the development board will automatically reset.

5.3.4 Erase Operation

This section describes how to use the upgrade_tool tool to erase the Flash (such as the eMMC device) of the development board. Similarly, the development board must be in Maskrom or Loader mode before the operation.

Executing the following command can erase all the data in the Flash storage (using the EF instruction):

```
sudo ../../RKTools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool_v2.4/upgrade_tool EF MiniLoaderAll.bin
```

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rockdev/Image-rk3568_r$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rockdev/Image-rk3568_r$ sudo ../../RKTools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool_v2.4/
upgrade_tool EF MiniLoaderAll.bin
Program Log will save in the /root/upgrade_tool/log/
Loading loader...
Erase flash ok.
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rockdev/Image-rk3568_r$
```

Figure 5.3.4.1 Erasure of all data

Of course, it is also possible to perform sector erasure by address, with the user specifying the starting position and size of the erasure (using the EL instruction).

Chapter 6. Android SDK Development

This chapter provides users with instructions on how to develop with the RK3568 Android SDK, covering u-boot development, Linux kernel development, Android system development, and Android application development. Although this SDK is currently applicable to the ALIENTEK ATK-DLRK3568 development board, users can conduct secondary development and customization based on this SDK to adapt it to their own Android products. Based on this SDK, system customization and application migration development can be effectively achieved, helping users to quickly develop and improve their development efficiency!

6.1 U-Boot development

The U-Boot source code is located in the **<SDK>/u-boot** directory, as shown below:

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot$ pwd
/home/tgg/rk3568_android_sdk/u-boot
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot$ ls
api     cmd        configs  Documentation  env       include  lib          Makefile  post           scripts          tools
arch    common     disk     drivers        examples  Kbuild   Licenses     make.sh   PREUPLOAD.cfg  snapshot.commit
board   config.mk  doc      dts            fs        Kconfig  MAINTAINERS  net       README         test
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot$
```

Figure 6.1-1 U-Boot source code directory

RK has provided a document that details the knowledge points and technical points related to Rockchip platform U-Boot for users, including an introduction to the basics of Rockchip platform U-Boot, U-Boot architecture, U-Boot startup process, U-Boot system modules, driver modules, Kernel-DTB, AB system, AVB secure boot, TPL, SPL, U-Boot shortcut keys, etc. The content is very extensive. Users who are not familiar with U-Boot (Rockchip platform U-Boot) are recommended to take a look at this document; its path is: **<SDK>/RKDocs/common/u-boot/Rockchip_Developer_Guide_UBoot_Nextdev_CN.pdf.** Similarly, the ATK-DLRK3568 development board data package also provides it, with the path being: Development board CD-ROM A disk - **Basic materials→08_RK_official_documents→Android→common→u-boot Rockchip_Developer_Guide_UBoot_Nextdev_CN.pdf**.

6.1.1 Device Tree of U-Boot

In U-Boot, the device tree file used by the RK3568 platform is <U-Boot>/arch/arm/dts/rk3568-evb.dts. This device tree file includes rk3568.dtsi and rk3568-u-boot.dtsi. The inclusion relationship is as follows:

```
rk3568-evb.dts
    rk3568.dtsi
    rk3568-u-boot.dtsi
```

The native U-Boot only supports its own DTB. On the RK platform, the native U-Boot has added support for the kernel DTB mechanism, meaning that U-Boot will use the kernel DTB to initialize the peripherals. The main purpose of this design is to accommodate the differences in peripheral board levels, such as power, clock, display, etc.

The U-Boot device tree is responsible for initializing basic peripherals such as storage and debug serial ports; while the kernel device tree initializes peripherals other than storage and debug serial ports, such as LCD display, Gigabit network, etc. When executing U-Boot code, it first uses the U-Boot device tree to complete the initialization of storage and debug serial ports, and then loads the kernel device tree from storage and continues to initialize the remaining peripherals using this device tree.

Therefore, users generally do not need to modify the U-Boot device tree file (unless they change the debug serial port).

6.1.2 U-Boot Compilation

There is a compilation script named make.sh in the U-Boot source code directory. You can directly use this script to compile the U-Boot source code. For example, to compile U-Boot in the U-Boot source code directory, execute the following command:

./make.sh rk3568

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot$ ./make.sh rk3568
grep: .config: 没有那个文件或目录
## make rk3568_defconfig -j24
  HOSTCC  scripts/basic/fixdep
  HOSTCC  scripts/kconfig/conf.o
  SHIPPED scripts/kconfig/zconf.tab.c
  SHIPPED scripts/kconfig/zconf.lex.c
  SHIPPED scripts/kconfig/zconf.hash.c
  HOSTCC  scripts/kconfig/zconf.tab.o
In file included from scripts/kconfig/zconf.tab.c:2468:
scripts/kconfig/confdata.c: In function ‘conf_write’:
scripts/kconfig/confdata.c:771:19: warning: ‘%s’ directive writing likely 7 or more bytes into a region of size b
  771 |   sprintf(newname, "%s%s", dirname, basename);
      |                    ^~~~~~
scripts/kconfig/confdata.c:771:19: note: assuming directive output of 7 bytes
In file included from /usr/include/stdio.h:867,
                 from scripts/kconfig/zconf.tab.c:82:
/usr/include/x86_64-linux-gnu/bits/stdio2.h:36:10: note: ‘__builtin___sprintf_chk’ output 1 or more bytes (assumi
   36 |   return __builtin___sprintf_chk (__s, __USE_FORTIFY_LEVEL - 1,
      |          ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   37 |        __bos (__s), __fmt, __va_arg_pack ());
      |        ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
In file included from scripts/kconfig/zconf.tab.c:2468:
scripts/kconfig/confdata.c:774:20: warning: ‘.tmpconfig.’ directive writing 11 bytes into a region of size betwee
  774 |   sprintf(tmpname, "%s.tmpconfig.%d", dirname, (int)getpid());
      |                    ^~~~~~~~~~~~~~~~~
In file included from /usr/include/stdio.h:867,
                 from scripts/kconfig/zconf.tab.c:82:
/usr/include/x86_64-linux-gnu/bits/stdio2.h:36:10: note: ‘__builtin___sprintf_chk’ output between 13 and 4119 byt
   36 |   return __builtin___sprintf_chk (__s, __USE_FORTIFY_LEVEL - 1,
      |          ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   37 |        __bos (__s), __fmt, __va_arg_pack ());
      |        ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  HOSTLD  scripts/kconfig/conf
#
# configuration written to .config
#
scripts/kconfig/conf  --silentoldconfig Kconfig
  CHK     include/config.h
  UPD     include/config.h
  CFG     u-boot.cfg
```

Figure 6.1.2.1 Compile the U-Boot source code (1)

```
  Hash algo:    sha256
  Hash value:   66bbd173528d12e9739c336926e33ee1ac1f4c7078fcac5712eeb8747d02163e
 Image 8 (fdt)
  Description:  U-Boot dtb
  Created:      Thu Apr 27 14:22:06 2023
  Type:         Flat Device Tree
  Compression:  uncompressed
  Data Size:    14377 Bytes = 14.04 KiB = 0.01 MiB
  Architecture: AArch64
  Hash algo:    sha256
  Hash value:   56cff76c011fcefff4f52549b1ed4a1ba3f5cccd48430161b0d24da4ae89918a
 Default Configuration: 'conf'
 Configuration 0 (conf)
  Description:  rk3568-evb
  Kernel:       unavailable
  Firmware:     atf-1
  FDT:          fdt
  Loadables:    uboot
                atf-2
                atf-3
                atf-4
                atf-5
                atf-6
                optee
********boot_merger ver 1.2********
Info:Pack loader ok.
pack loader okay! Input: /home/tgg/rk3568_android_sdk/rkbin/RKBOOT/RK3568MINIALL.ini
/home/tgg/rk3568_android_sdk/u-boot

Image(no-signed, version=0): uboot.img (FIT with uboot, trust...) is ready
Image(no-signed): rk356x_spl_loader_v1.13.112.bin (with spl, ddr...) is ready
pack uboot.img okay! Input: /home/tgg/rk3568_android_sdk/rkbin/RKTRUST/RK3568TRUST.ini

Platform RK3568 is build OK, with new .config(make rk3568_defconfig -j24)
/home/tgg/rk3568_android_sdk/prebuilts/gcc/linux-x86/aarch64/gcc-linaro-6.3.1-2017.05-x86_64_aarch64-linux-gn
2023年 04月 27日 星期四 14:22:06 CST
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot$
```

Figure 6.1.2.2 Compile the U-Boot source code (2)

After the compilation is completed, two image files, **uboot.img** and **rk356x_spl_loader_v1.13.112.bin**, will be generated. As shown below:

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot$ ls
api                  cmd            examples     make.sh                        tee.bin            uboot.img
arch                 common         fit          net                            test               u-boot.lds
bl31_0x00040000.bin  config.mk      fs           post                           tools              u-boot.map
bl31_0x00068000.bin  configs        include      PREUPLOAD.cfg                  tpl                u-boot-nodtb.bin
bl31_0x0006a000.bin  disk           Kbuild       README                         u-boot             u-boot.srec
bl31_0xfdcc1000.bin  doc            Kconfig      rk356x_spl_loader_v1.13.112.bin u-boot.bin        u-boot.sym
bl31_0xfdcce000.bin  Documentation  lib          scripts                        u-boot.cfg
bl31_0xfdcd0000.bin  drivers        Licenses     snapshot.commit                u-boot.cfg.configs
bl31.elf             dts            MAINTAINERS  spl                            u-boot.dtb
board                env            Makefile     System.map                     u-boot-dtb.bin
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot$
```

Figure 6.1.2.3 The image generated after compiling U-Boot

Of course, apart from these two images, many other.bin images are also generated, such as bl31_xxx.bin, tee.bin, u-boot.bin, u-boot-dtb.bin, u-boot-nodtb.bin, etc. These images are all intermediate products. Only uboot.img and rk356x_spl_loader_v1.13.112.bin are burned onto the development board in the end. Next, let's talk about these two image files.

1. uboot.img image

As mentioned earlier, uboot.img is formed by merging multiple images, including the u-boot image, u-boot dtb, and trust image (ARM Trusted Firmware + OP-TEE). uboot.img is a FIT (**flattened image tree)** format image and supports packaging and verification of any number of images. Use the file command to view uboot.img, as shown below:

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot$ file uboot.img
uboot.img: Device Tree Blob version 17, size=3584, boot CPU=0, string block size=197, DT structure block size=2824
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot$
```

FIT uses its (**image source file**) files to describe the information of the image. Finally, through the mkimage tool, an itb (**flattened image tree blob**) image is generated. This itb image is actually the uboot.img image. The uboot.img image usually contains multiple itb images, as shown below:

uboot.img = uboot.itb * N (N is usually 2)

This design is also to ensure that if the first mirror image fails to start, it is still possible to attempt to start using the second mirror image.

After U-Boot is successfully compiled, many.bin images and.dtb images will be generated in the U-Boot source code directory:

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot$ ls -l *.bin *.dtb
-rw-rw-r-- 1 tgg tgg  163840 4月  27 14:22 bl31_0x00040000.bin
-rw-rw-r-- 1 tgg tgg    7732 4月  27 14:22 bl31_0x00068000.bin
-rw-rw-r-- 1 tgg tgg   20267 4月  27 14:22 bl31_0x0006a000.bin
-rw-rw-r-- 1 tgg tgg   40960 4月  27 14:22 bl31_0xfdcc1000.bin
-rw-rw-r-- 1 tgg tgg    8192 4月  27 14:22 bl31_0xfdcce000.bin
-rw-rw-r-- 1 tgg tgg    8192 4月  27 14:22 bl31_0xfdcd0000.bin
-rw-rw-r-- 1 tgg tgg  465344 4月  27 14:22 rk356x_spl_loader_v1.13.112.bin
-rw-rw-r-- 1 tgg tgg  457112 4月  27 14:22 tee.bin
-rw-rw-r-- 1 tgg tgg 1282016 4月  27 14:22 u-boot.bin
-rw-rw-r-- 1 tgg tgg   14377 4月  27 14:22 u-boot.dtb
-rw-rw-r-- 1 tgg tgg 1282009 4月  27 14:22 u-boot-dtb.bin
-rwxrwxr-x 1 tgg tgg 1267632 4月  27 14:22 u-boot-nodtb.bin
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot$
```

Figure 6.1.2.4 The bin images in the U-Boot directory

Let's organize them as follows:

Mirror image	Explanation
u-boot.bin	The u-boot image itself includes the u-boot dtb.
u-boot.dtb	u-boot dtb
u-boot-dtb.bin	The u-boot itself image includes u-boot dtb and is equivalent and identical to u-boot.bin.
u-boot-nodtb.bin	The u-boot itself image does not include u-boot dtb.
bl31_0x00040000.bin	Both are ARM Trusted Firmware (ATF) firmware.
bl31_0x00068000.bin	
bl31_0x0006a000.bin	
bl31_0xfdcc1000.bin	
bl31_0xfdcce000.bin	
bl31_0xfdcd0000.bin	
tee.bin	OP-TEE firmware

Table 6.1.2.1 <U-Boot> Directory Image Introduction

The uboot.img image is ultimately formed by merging **u-boot-nodtb.bin**, **u-boot.dtb**, **bl31_0x00040000.bin**, **bl31_0x00068000.bin**, **bl31_0x0006a000.bin**, **bl31_0xfdcc1000.bin**, **bl31_0xfdcce000.bin**, **bl31_0xfdcd0000.bin,** and **tee.bin**.

uboot.img is a FIT format image. Its information is described using its file, and it is ultimately generated as an itb image through the **<U-Boot>/tools/mkimage** tool; the its file and the generated itb image are both in the <U-Boot>/fit directory:

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot$ ls -l fit/*
-rw-rw-r-- 1 tgg tgg 1994752 4月  27 14:22 fit/uboot.itb
-rw-rw-r-- 1 tgg tgg    2805 4月  27 14:22 fit/u-boot.its
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot$
```

Figure 6.1.2.5 The files in the "fit" directory

The grammar rules of the u-boot.its file are the same as those of the device tree DTS, and they are very flexible!

The u-boot.its file describes which images will be merged into uboot.itb, as well as the paths and other information of these images:

```
/dts-v1/;

/ {
        description = "FIT Image with ATF/OP-TEE/U-Boot/MCU";
        #address-cells = <1>;

        images {

                uboot {
                        description = "U-Boot";
                        data = /incbin/("u-boot-nodtb.bin");
                        type = "standalone";
                        arch = "arm64";
                        os = "U-Boot";
                        compression = "none";
                        load = <0x00a00000>;
                        hash {
                                algo = "sha256";
                        };
                };
                atf-1 {
                        description = "ARM Trusted Firmware";
                        data = /incbin/("./bl31_0x00040000.bin");
                        type = "firmware";
                        arch = "arm64";
                        os = "arm-trusted-firmware";
                        compression = "none";
                        load = <0x00040000>;
                        hash {
                                algo = "sha256";
                        };
                };
                atf-2 {
                        description = "ARM Trusted Firmware";
```

```
                        data = /incbin/("./bl31_0xfdcc1000.bin");
                        type = "firmware";
                        arch = "arm64";
                        os = "arm-trusted-firmware";
                        compression = "none";
                        load = <0xfdcc1000>;
                        hash {
                                algo = "sha256";
                        };
                };
                atf-3 {
                        description = "ARM Trusted Firmware";
                        data = /incbin/("./bl31_0x0006a000.bin");
                        type = "firmware";
                        arch = "arm64";
                        os = "arm-trusted-firmware";
                        compression = "none";
                        load = <0x0006a000>;
                        hash {
                                algo = "sha256";
                        };
                };
                atf-4 {
                        description = "ARM Trusted Firmware";
                        data = /incbin/("./bl31_0xfdcd0000.bin");
                        type = "firmware";
                        arch = "arm64";
                        os = "arm-trusted-firmware";
                        compression = "none";
                        load = <0xfdcd0000>;
                        hash {
                                algo = "sha256";
                        };
                };
                atf-5 {
                        description = "ARM Trusted Firmware";
                        data = /incbin/("./bl31_0xfdcce000.bin");
                        type = "firmware";
                        arch = "arm64";
                        os = "arm-trusted-firmware";
                        compression = "none";
                        load = <0xfdcce000>;
                        hash {
```

```
                        algo = "sha256";
                };
        };
        atf-6 {
                description = "ARM Trusted Firmware";
                data = /incbin/("./bl31_0x00068000.bin");
                type = "firmware";
                arch = "arm64";
                os = "arm-trusted-firmware";
                compression = "none";
                load = <0x00068000>;
                hash {
                        algo = "sha256";
                };
        };
        optee {
                description = "OP-TEE";
                data = /incbin/("tee.bin");
                type = "firmware";
                arch = "arm64";
                os = "op-tee";
                compression = "none";

                load = <0x8400000>;
                hash {
                        algo = "sha256";
                };
        };
        fdt {
                description = "U-Boot dtb";
                data = /incbin/("./u-boot.dtb");
                type = "flat_dt";
                arch = "arm64";
                compression = "none";
                hash {
                        algo = "sha256";
                };
        };
};

configurations {
        default = "conf";
        conf {
```

```
                                description = "rk3568-evb";
                                rollback-index = <0x0>;
                                firmware = "atf-1";
                                loadables = "uboot", "atf-2", "atf-3", "atf-4", "atf-5", "atf-6",
"optee";

                                fdt = "fdt";
                                signature {
                                        algo = "sha256,rsa2048";

                                        key-name-hint = "dev";
                                        sign-images = "fdt", "firmware", "loadables";
                                };
                        };
                };
        };
```

From this, it can be seen that the grammar rules of the "its" file are exactly the same as those of DTS, with no differences at all! For details, please refer to the documents in the **<U-Boot>/doc/uImage.FIT/** directory.

For ARM Trusted Firmware and OP-TEE, they are closed-source. RK only provides binary image files and does not provide source code. The ARM Trusted Firmware firmware corresponds to **<U-Boot>/bl31.elf,** and the OP-TEE firmware corresponds to **<U-Boot>/tee.bin**, as shown below:

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot$ ls bl31.elf tee.bin -lh
-rwxrwxr-x 1 tgg tgg 393K 4月  27 14:22 bl31.elf
-rw-rw-r-- 1 tgg tgg 447K 4月  27 14:22 tee.bin
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot$
```

Figure 6.1.2.6 ATF firmware and OP-TEE firmware

Before compiling the U-Boot source code, these two images do not exist; they will appear after the compilation. In fact, they come from the two image files **<SDK>/rkbin/bin/rk35/rk3568_bl31_v1.34.elf** and **<SDK>/rkbin/bin/rk35/rk3568_bl32_v2.08.bin**. During the packaging of uboot.itb, <SDK>/rkbin/bin/rk35/rk3568_bl31_v1.34.elf will be copied to <U-Boot>/bl31.elf, and <SDK>/rkbin/bin/rk35/rk3568_bl32_v2.08.bin will be copied to <U-Boot>/tee.bin.

The bl31.elf firmware is not directly packaged into uboot.itb; instead, bl31.elf is decomposed into multiple bl31_xxx.bin files, and finally these bl31_xxx.bin images are packaged into uboot.itb.

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot$ ls -l bl31_0x*
-rw-rw-r-- 1 tgg tgg 163840 4月  27 14:22 bl31_0x00040000.bin
-rw-rw-r-- 1 tgg tgg   7732 4月  27 14:22 bl31_0x00068000.bin
-rw-rw-r-- 1 tgg tgg  20267 4月  27 14:22 bl31_0x0006a000.bin
-rw-rw-r-- 1 tgg tgg  40960 4月  27 14:22 bl31_0xfdcc1000.bin
-rw-rw-r-- 1 tgg tgg   8192 4月  27 14:22 bl31_0xfdcce000.bin
-rw-rw-r-- 1 tgg tgg   8192 4月  27 14:22 bl31_0xfdcd0000.bin
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot$
```

Figure 6.1.2.7 bl31_xxx.bin firmware

2. rk356x_spl_loader_v1.13.112.bin image

Let's talk about rk356x_spl_loader_v1.13.112.bin again. This image file is the same as the MiniLoaderAll.bin image (renamed from rk356x_spl_loader_v1.13.112.bin). As we mentioned earlier, MiniLoaderAll.bin is a Loader code that runs before U-Boot on the RK3568 platform (it is a Loader at a stage earlier than U-Boot).

MiniLoaderAll.bin consists of two parts: **TPL (Tiny Program Loader)** + **SPL (Secondary Program Loader)**.

TPL operates in SRAM (internal memory) and is initially started by the Maskrom code embedded in the rk3568 chip; its function is to complete the initialization of DRAM and to start SPL. SPL operates in DDR, and its role is to complete the low-level initialization of the system, load and boot uboot.img.

TPL and SPL have two implementation schemes: **open-source version** and **closed-source version**.

Closed-source version

RK provides users with binary image files of tpl and spl, but does not provide their corresponding source codes. The paths are as follows:

<SDK>/rkbin/bin/rk35/rk3568_ddr_1560MHz_v1.13.bin ----------> tpl

<SDK>/rkbin/bin/rk35/rk356x_spl_v1.12.bin ----------> spl

In the <SDK>/rkbin/bin directory, there are many binary files provided by RK. These binary files are not open-source, so only binary files are available.

Open-source version

The tpl and spl images can also be compiled from the U-Boot source code. Since they are part of the U-Boot source code, they are naturally open-source. After compiling the U-Boot source code using the command "./make.sh rk3568", in addition to generating the u-boot image, the spl and tpl images will also be compiled and generated. The spl image is located in **<U-Boot>/spl** directory:

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot$ cd spl/
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot/spl$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot/spl$ ls
arch   cmd     disk     dts  fs       lib         u-boot-spl      u-boot-spl.dtb      u-boot-spl.lds  u-boot-spl-nodtb.bin
board  common  drivers  env  include  u-boot.cfg  u-boot-spl.bin  u-boot-spl-dtb.bin  u-boot-spl.map  u-boot-spl.sym
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot/spl$
```

Figure 6.1.2.8 The files in the "spl" directory

There are also many image files in this directory:

Mirror image	Explanation
u-boot-spl	Elf format spl image
u-boot-spl.bin	Spl image, including spl dtb spl dtb
u-boot-spl.dtb	Spl image, including spl dtb, is the same as u-boot-spl.bin
u-boot-spl-dtb.bin	Spl image, without including spl dtb
u-boot-spl-nodtb.bin	Explanation

Table 6.1.2.2 Introduction to Spl Directory Image

The tpl image is located in the **<U-Boot>/tpl** directory.

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot$ cd tpl/
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot/tpl$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot/tpl$ ls
arch    common  drivers  fs       u-boot.cfg      u-boot-tpl      u-boot-tpl.map        u-boot-tpl.sym
board   disk    dts      include  u-boot-spl.lds  u-boot-tpl.bin  u-boot-tpl-nodtb.bin
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot/tpl$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot/tpl$
```

Figure 6.1.2.9 The files in the tpl directory

In the RK documentation, the closed-source TPL image <SDK>/rkbin/bin/rk35/rk3568_ddr_1560MHz_v1.13.bin provided by RK is called **ddr bin**, and the closed-source SPL image <SDK>/rkbin/bin/rk35/rk356x_spl_v1.12.bin is called **miniloader**.

The image file rk356x_spl_loader_v1.13.112.bin is made by the **<U-Boot>/tools/boot_merger** tool, which is provided by RK. When making the image file rk356x_spl_loader_v1.13.112.bin using the boot_merger tool, an.ini file needs to be provided. The.ini file is used to describe the information of the image; there are many.ini files in the **<SDK>/rkbin/RKBOOT** directory, as shown in the following figure:

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rkbin/RKBOOT$ pwd
/home/tgg/rk3568_android_sdk/rkbin/RKBOOT
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rkbin/RKBOOT$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rkbin/RKBOOT$ ls
PX30MINIALL.ini            RK30MINIALL.ini            RK3326AARCH32MINIALL.ini      RKCAYMAN.ini
PX30MINIALL_SLC.ini        RK30MINI.ini               RK3326AARCH32MINIALL_SLC.ini  RKCROWN.ini
PX30MINIALL_WO_FTL.ini     RK310B.ini                 RK3326MINIALL.ini             RKNANO.ini
PX3SEMINIALL.ini           RK310BMINIALL.ini          RK3326MINIALL_SLC.ini         RKNPULIONMINIALL.ini
PX3SEMINIALL_SLC.ini       RK310BMINI.ini             RK3328MINIALL.ini             RKPANDA.ini
PX5KERNEL4.4MINIALL.ini    RK3126.ini                 RK3366MINIALL.ini             RKSMART.ini
PX5MINIALL.ini             RK3126MINIALL.ini          RK3368BOXMINIALL.ini          RV1106MINIALL_EMMC_TB.ini
RK1806MINIALL.ini          RK3126MINIALL_SLC.ini      RK3368HMINIALL.ini            RV1106MINIALL.ini
RK1808MINIALL.ini          RK3126MINIALL_WO_FTL.ini   RK3368.ini                    RV1106MINIALL_SPI_NOR_TB.ini
RK1808MINIALL_WO_FTL.ini   RK3128.ini                 RK3368MINIALL.ini             RV110XMINIALL.ini
RK302A.ini                 RK3128MINIALL.ini          RK3399MINIALL.ini             RV1126MINIALL_EMMC_TB.ini
RK302AMINIALL.ini          RK3128MINIALL_SLC.ini      RK3399MINIALL_SPINOR.ini      RV1126MINIALL_FTL.ini
RK302AMINI.ini             RK3128XMINIALL.ini         RK3399PROMINIALL.ini          RV1126MINIALL.ini
RK3032MINIALL.ini          RK3188MINIALL.ini          RK3566MINIALL.ini             RV1126MINIALL_IPC.ini
RK3032MINIALL_SLC.ini      RK322XATMINIALL.ini        RK3566MINIALL_NAND.ini        RV1126MINIALL_LP3_EMMC_TB.ini
RK3036_ECHOMINIALL.ini     RK322XHMINIALL.ini         RK3566MINIALL_ULTRA.ini       RV1126MINIALL_LP4_EMMC_TB.ini
RK3036.ini                 RK322XMINIALL.ini          RK3568MINIALL.ini             RV1126MINIALL_RAMBOOT.ini
RK3036MINIALL.ini          RK3288.ini                 RK3568MINIALL_NAND.ini        RV1126MINIALL_SPI_NOR_TB.ini
RK3036MINIALL_SLC.ini      RK3288MINIALL.ini          RK3568MINIALL_RAMBOOT.ini     RV1126MINIALL_SPI_NOR_TINY.ini
RK30B.ini                  RK3308MINIALL.ini          RK3568MINIALL_SPI_NAND.ini    SDBOOT.ini
RK30BMINIALL.ini           RK3308MINIALL_SPI_NAND.ini RK3588MINIALL.ini
RK30BMINI.ini              RK3308MINIALL_UART4.ini    RK3588MINIALL_IPC.ini
RK30.ini                   RK3308MINIALL_WO_FTL.ini   RK3588MINIALL_RAMBOOT.ini
tgg@tgg-virtual-machine:~/rk3568_android_sdk/rkbin/RKBOOT$
```

Figure 6.1.2.10 .ini file

For the RK3568 platform, it defaults to using **RK3568MINIALL.ini**. The file content is as follows:

```
[CHIP_NAME]
NAME=RK3568
[VERSION]
MAJOR=1
MINOR=1
[CODE471_OPTION]
NUM=1
Path1=bin/rk35/rk3568_ddr_1560MHz_v1.13.bin
Sleep=1
[CODE472_OPTION]
NUM=1
Path1=bin/rk35/rk356x_usbplug_v1.14.bin
[LOADER_OPTION]
```

```
NUM=2
LOADER1=FlashData
LOADER2=FlashBoot
FlashData=bin/rk35/rk3568_ddr_1560MHz_v1.13.bin
FlashBoot=bin/rk35/rk356x_spl_v1.12.bin
[OUTPUT]
PATH=rk356x_spl_loader_v1.13.112.bin
[SYSTEM]
NEWIDB=true
[FLAG]
471_RC4_OFF=true
RC4_OFF=true
```

FlashData specifies the path of the tpl image; FlashBoot specifies the path of the spl image. Therefore, by default, when executing the command "./make.sh rk3568", the rk356x_spl_loader_v1.13.112.bin generated uses the RK closed-source ddr bin (<SDK>/rkbin/bin/rk35/rk3568_ddr_1560MHz_v1.13.bin) and the RK closed-source miniloader (<SDK>/rkbin/bin/rk35/rk356x_spl_v1.12.bin).

In addition to using the RK closed-source miniloader, an open-source spl compiled by U-Boot can also be used (**note that the tpl image compiled by U-Boot cannot be used; I have tested and found that it cannot be directly started, probably because the U-Boot TPL part of the code is not fully supported**). By executing the following command, the spl image generated by U-Boot (<U-Boot>/spl/u-boot-spl.bin) can be used to replace the RK closed-source miniloader to create rk356x_spl_loader_v1.13.112.bin:

```
./make.sh --spl
```

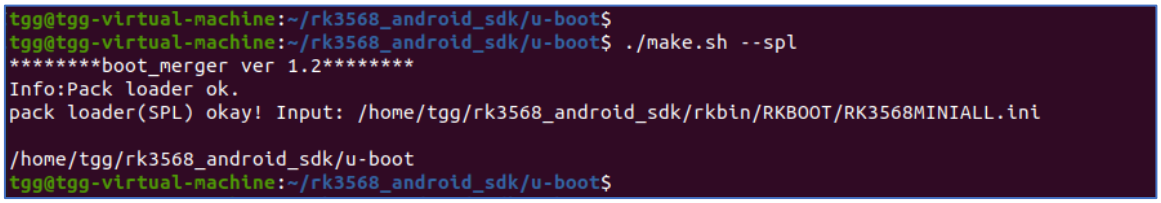


Figure 6.1.2.11 Generate MiniLoaderAll.bin using open-source SPL

Default setting: MiniLoaderAll.bin = Closed-source ddr bin + Closed-source miniloader;

After executing "./make.sh --spl": MiniLoaderAll.bin = Closed-source ddr bin + Open-source SPL.

The make.sh script also supports many other parameters. To view its usage help information, execute "**./make.sh -h**". Please experiment on your own.

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot$ ./make.sh -h

Usage:
        ./make.sh [board|sub-command]

         - board:        board name of defconfig
         - sub-command:  elf*|loader|trust|uboot|--spl|--tpl|itb|map|sym|<addr>
         - ini:          ini file to pack trust/loader

Output:
         When board built okay, there are uboot/trust/loader images in current directory

Example:

1. Build:
        ./make.sh evb-rk3399                --- build for evb-rk3399_defconfig
        ./make.sh firefly-rk3288            --- build for firefly-rk3288_defconfig
        ./make.sh EXT_DTB=rk-kernel.dtb     --- build with exist .config and external dtb
        ./make.sh                           --- build with exist .config
        ./make.sh env                       --- build envtools

2. Pack:
        ./make.sh uboot                     --- pack uboot.img
        ./make.sh trust                     --- pack trust.img
        ./make.sh trust <ini>               --- pack trust img with assigned ini file
        ./make.sh loader                    --- pack loader bin
        ./make.sh loader <ini>              --- pack loader img with assigned ini file
        ./make.sh --spl                     --- pack loader with u-boot-spl.bin
        ./make.sh --tpl                     --- pack loader with u-boot-tpl.bin
        ./make.sh --tpl --spl               --- pack loader with u-boot-tpl.bin and u-boot-spl.bin

3. Debug:
        ./make.sh elf                       --- dump elf file with -D(default)
        ./make.sh elf-S                     --- dump elf file with -S
        ./make.sh elf-d                     --- dump elf file with -d
        ./make.sh elf-*                     --- dump elf file with -*
        ./make.sh <no reloc_addr>           --- unwind address(no relocated)
        ./make.sh <reloc_addr-reloc_off>    --- unwind address(relocated)
        ./make.sh map                       --- cat u-boot.map
        ./make.sh sym                       --- cat u-boot.sym
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot$
```

Figure 6.1.2.12 Help information for the make.sh script

3. Boot sequence of the image

Here, we will explain the boot sequence of the image for the RK3568 platform. Regarding Trust, currently, on Rockchip's 64-bit SoC platforms, the combination of **ARM Trusted Firmware + OP-TEE** is used to implement Trust, while on 32-bit SoC platforms, OP-TEE is used.

The architecture of ARM Trusted Firmware divides the entire system into four security levels: EL0, EL1, EL2, and EL3. The entire security boot process is defined as: BL1, BL2, BL31, BL32, and BL33. The source code of ARM Trusted Firmware provides the functions of BL1, BL2, and BL31. On the Rockchip platform, only the function of BL31 is used; for BL1 and BL2, RK has its own implementation scheme. Therefore, on the Rockchip platform, we can generally "default" that ARM Trusted Firmware refers to BL31, and BL32 uses OP-TEE.

If the above stage definitions are mapped to the firmware levels of the Rockchip platform, the corresponding relationship is: Maskrom (the boot code internally solidified in the RK chip, BL1), MiniLoaderAll.bin (BL2), Trust (BL31: ARM Trusted Firmware + BL32: OP-TEE), U-Boot (BL33).

So, the boot sequence of the Linux system image is:

Maskrom → MiniLoaderAll.bin → uboot.img → boot.img → Android

It can also be further divided into:

Maskrom → TPL(ddr bin) → SPL(miniloader) → Trust(ATF + OP-TEE) → u-boot → kernel → Android

This startup process can be analyzed simply by printing the information. The following is the printed information when the ATK-DLRK3568 development board is powered on and starts up:

 # Execute ddr bin, to initialize the DDR of the development board

DDR Version V1.13 20220218

```
In
ddrconfig:15
DDR4, 324MHz
BW=32 Col=10 Bk=4 BG=2 CS0 Row=16 CS=1 Die BW=16 Size=2048MB
tdqss: cs0 dqs0: 24ps, dqs1: -48ps, dqs2: -96ps, dqs3: -96ps,

change to: 324MHz
clk skew:0x87
PHY drv:clk:37,ca:37,DQ:37,odt:0
vrefinner:50%, vrefout:50%
dram drv:34,odt:0

change to: 528MHz
clk skew:0x87
PHY drv:clk:37,ca:37,DQ:37,odt:139
vrefinner:50%, vrefout:61%
dram drv:34,odt:120

change to: 780MHz
clk skew:0x87
PHY drv:clk:37,ca:37,DQ:37,odt:139
vrefinner:50%, vrefout:61%
dram drv:34,odt:120

change to: 1560MHz(final freq)
clk skew:0x87
PHY drv:clk:37,ca:37,DQ:37,odt:139
vrefinner:50%, vrefout:61%
dram drv:34,odt:120
cs 0:
the read training result:
DQS0:0x32, DQS1:0x2c, DQS2:0x27, DQS3:0x30,
min    : 0xf    0xe    0xd    0x8    0x3    0x2    0x5    0x5 , 0x7    0x5    0x1    0x3    0x5    0x7    0x9
0x9 ,
               0x5    0x8    0x9    0x7    0x5    0x3    0x2    0x5 , 0xa    0x6    0x5    0x2    0x7    0xb
0x7    0x9 ,
mid    :0x28 0x27 0x27 0x21 0x1b 0x1b 0x1e 0x1c ,0x1f 0x1d 0x19 0x1c 0x1e 0x1f 0x21 0x1e ,
               0x1d 0x20 0x20 0x1e 0x1d 0x1c 0x1a 0x1d ,0x23 0x1e 0x1e 0x1a 0x20 0x23 0x20 0x21 ,
max    :0x41 0x41 0x41 0x3b 0x34 0x35 0x37 0x34 ,0x38 0x36 0x32 0x36 0x37 0x37 0x39 0x34 ,
               0x36 0x39 0x37 0x36 0x36 0x35 0x33 0x35 ,0x3c 0x37 0x38 0x32 0x39 0x3c 0x39 0x39 ,
range:0x32 0x33 0x34 0x33 0x31 0x33 0x32 0x2f ,0x31 0x31 0x31 0x33 0x32 0x30 0x30 0x2b ,
               0x31 0x31 0x2e 0x2f 0x31 0x32 0x31 0x30 ,0x32 0x31 0x33 0x30 0x32 0x31 0x32 0x30 ,
the write training result:
```

```
DQS0:0x8b, DQS1:0x7e, DQS2:0x74, DQS3:0x74,
min    :0x76 0x76 0x7c 0x77 0x6b 0x6d 0x6f 0x6f 0x78 ,0x6d 0x6e 0x69 0x6e 0x6c 0x6e 0x74 0x6d 0x69 ,
        0x67 0x63 0x6a 0x69 0x64 0x5e 0x65 0x64 0x60 ,0x63 0x5e 0x63 0x5d 0x63 0x5f 0x61 0x67 0x61 ,
mid    :0x8f 0x8e 0x93 0x8e 0x81 0x83 0x85 0x87 0x91 ,0x84 0x85 0x7d 0x84 0x83 0x85 0x88 0x84 0x7e ,
        0x7c 0x7d 0x7f 0x7d 0x7a 0x76 0x78 0x7a 0x78 ,0x79 0x74 0x78 0x71 0x79 0x78 0x78 0x7c 0x79 ,
max    :0xa9 0xa7 0xab 0xa6 0x97 0x9a 0x9c 0xa0 0xaa ,0x9c 0x9c 0x92 0x9b 0x9b 0x9c 0x9c 0x9c 0x93 ,
        0x91 0x97 0x94 0x91 0x91 0x8e 0x8c 0x91 0x90 ,0x90 0x8b 0x8e 0x86 0x8f 0x91 0x90 0x91 0x91 ,
range:0x33 0x31 0x2f 0x2f 0x2c 0x2d 0x2d 0x31 0x32 ,0x2f 0x2e 0x29 0x2d 0x2f 0x2e 0x28 0x2f 0x2a ,
        0x2a 0x34 0x2a 0x28 0x2d 0x30 0x27 0x2d 0x30 ,0x2d 0x2d 0x2b 0x29 0x2c 0x32 0x2f 0x2a 0x30 ,
out
# DDR bin operation completed. Starting to run SPL (mini-loader) code.
U-Boot SPL board init
U-Boot SPL 2017.09-gaaca6ffec1-211203 #zzz (Dec 03 2021 - 18:42:16)
unknown raw ID phN
unrecognized JEDEC id bytes: 00, 00, 00
Trying to boot from MMC2
MMC error: The cmd index is 1, ret is -110
Card did not respond to voltage select!
mmc_init: -95, time 10
spl: mmc init failed with error: -95
Trying to boot from MMC1
SPL: A/B-slot: _a, successful: 0, tries-remain: 7
Trying fit image at 0x4000 sector
## Verified-boot: 0
## Checking atf-1 0x00040000 ... sha256(6204b6f381...) + OK
## Checking uboot 0x00a00000 ... sha256(08dca575f2...) + OK
## Checking fdt 0x00b35800 ... sha256(56cff76c01...) + OK
## Checking atf-2 0xfdcc1000 ... sha256(5563d929da...) + OK
## Checking atf-3 0x0006a000 ... sha256(b04372ab0f...) + OK
## Checking atf-4 0xfdcd0000 ... sha256(b46eaa95b8...) + OK
## Checking atf-5 0xfdcce000 ... sha256(2f8839c803...) + OK
## Checking atf-6 0x00068000 ... sha256(6e9d32ba23...) + OK
## Checking optee 0x08400000 ... sha256(66bbd17352...) + OK
Jumping to U-Boot(0x00a00000) via ARM Trusted Firmware(0x00040000)
Total: 236.326 ms
```

```
# SPL operation is completed (MiniLoaderAll.bin operation is over), and the program has jumped to ATF and OP-TEE.
INFO:    Preloader serial: 2
NOTICE:  BL31: v2.3():v2.3-365-gae7c295ca:derrick.huang
NOTICE:  BL31: Built : 15:37:13, May 17 2022
INFO:    GICv3 without legacy support detected.
INFO:    ARM GICv3 driver initialized in EL3
INFO:    pmu v1 is valid 220114
INFO:    dfs DDR fsp_param[0].freq_mhz= 1560MHz
INFO:    dfs DDR fsp_param[1].freq_mhz= 324MHz
INFO:    dfs DDR fsp_param[2].freq_mhz= 528MHz
INFO:    dfs DDR fsp_param[3].freq_mhz= 780MHz
INFO:    Using opteed sec cpu_context!
INFO:    boot cpu mask: 0
INFO:    BL31: Initializing runtime services
INFO:    BL31: Initializing BL32
I/TC:
I/TC: OP-TEE version: 3.13.0-641-g4167319d3 #hisping.lin (gcc version 10.2.1 20201103 (GNU Toolchain for the A-profile Architecture 10.2-2020.11 (arm-10.16))) #8 Wed Mar 16 15:14:56 CST 2022 aarch64
I/TC: Primary CPU initializing
I/TC: Primary CPU switching to normal world boot
INFO:    BL31: Preparing for EL3 exit to normal world
INFO:    Entry point address = 0xa00000
INFO:    SPSR = 0x3c9

# Enter U-Boot and start executing the U-Boot code.
U-Boot 2017.09-gc2d9f80c05-220621 #tgg (Apr 10 2023 - 11:31:38 +0800)

Model: Rockchip RK3568 Evaluation Board
PreSerial: 2, raw, 0xfe660000
DRAM:  2 GiB
Sysmem: init
Relocation Offset: 7d343000
Relocation fdt: 7b9f84e0 - 7b9fece0
CR: M/C/I
Using default environment

dwmmc@fe2b0000: 1, dwmmc@fe2c0000: 2, sdhci@fe310000: 0
Bootdev(atags): mmc 0
MMC0: HS200, 200Mhz
PartType: EFI
```

DM: v1

boot mode: None

……

……

```
DDR Version V1.13 20220218
In
ddrconfig:15
DDR4, 324MHz
BW=32 Col=10 Bk=4 BG=2 CS0 Row=16 CS=1 Die BW=16 Size=2048MB
tdqss: cs0 dqs0: 24ps, dqs1: -48ps, dqs2: -96ps, dqs3: -96ps,

change to: 324MHz
clk skew:0x87
PHY drv:clk:37,ca:37,DQ:37,odt:0
vrefinner:50%, vrefout:50%
dram drv:34,odt:0

change to: 528MHz
clk skew:0x87
PHY drv:clk:37,ca:37,DQ:37,odt:139
vrefinner:50%, vrefout:61%
dram drv:34,odt:120

change to: 780MHz
clk skew:0x87
PHY drv:clk:37,ca:37,DQ:37,odt:139
vrefinner:50%, vrefout:61%
dram drv:34,odt:120

change to: 1560MHz(final freq)
clk skew:0x87
PHY drv:clk:37,ca:37,DQ:37,odt:139
vrefinner:50%, vrefout:61%
dram drv:34,odt:120
cs 0:
the read training result:
DQS0:0x32, DQS1:0x2c, DQS2:0x27, DQS3:0x30,
min  : 0xf  0xe  0xd  0x8  0x3  0x2  0x5  0x5 , 0x7  0x5  0x1  0x3  0x5  0x7  0x9  0x9 ,
       0x5  0x8  0x9  0x7  0x5  0x3  0x2  0x5 , 0xa  0x6  0x5  0x2  0x7  0xb  0x7  0x9 ,
mid  :0x28 0x27 0x27 0x21 0x1b 0x1b 0x1e 0x1c ,0x1f 0x1d 0x19 0x1c 0x1e 0x1f 0x21 0x1e ,
      0x1d 0x20 0x20 0x1e 0x1d 0x1c 0x1a 0x1d ,0x23 0x1e 0x1e 0x1a 0x20 0x23 0x20 0x21 ,
max  :0x41 0x41 0x41 0x3b 0x34 0x35 0x37 0x34 ,0x38 0x36 0x32 0x36 0x37 0x37 0x39 0x34 ,
      0x36 0x39 0x37 0x36 0x36 0x35 0x33 0x35 ,0x3c 0x37 0x38 0x32 0x39 0x3c 0x39 0x39 ,
range:0x32 0x33 0x34 0x33 0x31 0x33 0x32 0x2f ,0x31 0x31 0x31 0x33 0x32 0x30 0x30 0x2b ,
      0x31 0x31 0x2e 0x2f 0x31 0x32 0x31 0x30 ,0x32 0x31 0x33 0x30 0x32 0x31 0x32 0x30 ,
the write training result:
DQS0:0x8b, DQS1:0x7e, DQS2:0x74, DQS3:0x74,
min  :0x76 0x76 0x7c 0x77 0x6b 0x6d 0x6f 0x6f 0x78 ,0x6d 0x6e 0x69 0x6e 0x6c 0x6e 0x74 0x6d 0x69 ,
      0x67 0x63 0x6a 0x69 0x64 0x5e 0x65 0x64 0x60 ,0x63 0x5e 0x63 0x5d 0x63 0x5f 0x61 0x67 0x61 ,
mid  :0x8f 0x8e 0x93 0x8e 0x81 0x83 0x85 0x87 0x91 ,0x84 0x85 0x7d 0x84 0x83 0x85 0x88 0x84 0x7e ,
      0x7c 0x7d 0x7f 0x7d 0x7a 0x76 0x78 0x7a 0x78 ,0x79 0x74 0x78 0x71 0x79 0x78 0x78 0x7c 0x79 ,
max  :0xa9 0xa7 0xab 0xa6 0x97 0x9a 0x9c 0xa0 0xaa ,0x9c 0x9c 0x92 0x9b 0x9b 0x9c 0x9c 0x9c 0x93 ,
      0x91 0x97 0x94 0x91 0x91 0x8e 0x8c 0x91 0x90 ,0x90 0x8b 0x8e 0x86 0x8f 0x91 0x90 0x91 0x91 ,
range:0x33 0x31 0x2f 0x2f 0x2c 0x2d 0x2d 0x31 0x32 ,0x2f 0x2e 0x29 0x2d 0x2f 0x2e 0x28 0x2f 0x2a ,
      0x2a 0x34 0x2a 0x28 0x2d 0x30 0x27 0x2d 0x30 ,0x2d 0x2d 0x2b 0x29 0x2c 0x32 0x2f 0x2a 0x30 ,
out
```

Figure 6.1.2.13 The operation process of TPL (DDR BIN)

```
U-Boot SPL board init
U-Boot SPL 2017.09-gaaca6ffec1-211203 #zzz (Dec 03 2021 - 18:42:16)
unknown raw ID phN
unrecognized JEDEC id bytes: 00, 00, 00
Trying to boot from MMC2
MMC error: The cmd index is 1, ret is -110
Card did not respond to voltage select!
mmc_init: -95, time 10
spl: mmc init failed with error: -95
Trying to boot from MMC1
SPL: A/B-slot: _a, successful: 0, tries-remain: 7
Trying fit image at 0x4000 sector
## Verified-boot: 0
## Checking atf-1 0x00040000 ... sha256(6204b6f381...) + OK
## Checking uboot 0x00a00000 ... sha256(08dca575f2...) + OK
## Checking fdt 0x00b35800 ... sha256(56cff76c01...) + OK
## Checking atf-2 0xfdcc1000 ... sha256(5563d929da...) + OK
## Checking atf-3 0x0006a000 ... sha256(b04372ab0f...) + OK
## Checking atf-4 0xfdcd0000 ... sha256(b46eaa95b8...) + OK
## Checking atf-5 0xfdcce000 ... sha256(2f8839c803...) + OK
## Checking atf-6 0x00068000 ... sha256(6e9d32ba23...) + OK
## Checking optee 0x08400000 ... sha256(66bbd17352...) + OK
Jumping to U-Boot(0x00a00000) via ARM Trusted Firmware(0x00040000)
Total: 236.326 ms
```

Figure 6.1.2.14 The operation process of SPL

```
INFO:    Preloader serial: 2
NOTICE:  BL31: v2.3():v2.3-365-gae7c295ca:derrick.huang
NOTICE:  BL31: Built : 15:37:13, May 17 2022
INFO:    GICv3 without legacy support detected.
INFO:    ARM GICv3 driver initialized in EL3
INFO:    pmu v1 is valid 220114
INFO:    dfs DDR fsp_param[0].freq_mhz= 1560MHz
INFO:    dfs DDR fsp_param[1].freq_mhz= 324MHz
INFO:    dfs DDR fsp_param[2].freq_mhz= 528MHz
INFO:    dfs DDR fsp_param[3].freq_mhz= 780MHz
INFO:    Using opteed sec cpu_context!
INFO:    boot cpu mask: 0
INFO:    BL31: Initializing runtime services
INFO:    BL31: Initializing BL32
I/TC:
I/TC: OP-TEE version: 3.13.0-641-g4167319d3 #hisping.lin (gcc version 10.2.1 20201103 (GNU Toolchain for the A-p
I/TC: Primary CPU initializing
I/TC: Primary CPU switching to normal world boot
INFO:    BL31: Preparing for EL3 exit to normal world
INFO:    Entry point address = 0xa00000
INFO:    SPSR = 0x3c9
```

Figure 6.1.2.15 The process of Trust operation

```
U-Boot 2017.09-gc2d9f80c05-220621 #tgg (Apr 10 2023 - 11:31:38 +0800)

Model: Rockchip RK3568 Evaluation Board
PreSerial: 2, raw, 0xfe660000
DRAM:  2 GiB
Sysmem: init
Relocation Offset: 7d343000
Relocation fdt: 7b9f84e0 - 7b9fece0
CR: M/C/I
Using default environment

dwmmc@fe2b0000: 1, dwmmc@fe2c0000: 2, sdhci@fe310000: 0
Bootdev(atags): mmc 0
MMC0: HS200, 200Mhz
PartType: EFI
DM: v1
boot mode: None
FIT: no signed, no conf required
Error: find duplicate(3) dtbs
DTB: rk3568-atk-evb1-mipi-dsi-1080p#_saradc_ch2=341.dtb
HASH(c): OK
I2c0 speed: 100000Hz
PMIC:  RK8090 (on=0x40, off=0x00)
vsel-gpios- not found! Error: -2
vdd_cpu init 900000 uV
vdd_logic init 900000 uV
vdd_gpu init 900000 uV
vdd_npu init 900000 uV
io-domain: OK
Could not find baseparameter partition
Model: Rockchip RK3568 ATK EVB1 DDR4 V10 Board
Rockchip UBOOT DRM driver version: v1.0.1
VOP have 3 active VP
vp0 have layer nr:2[1 5 ], primary plane: 5
vp1 have layer nr:3[0 2 4 ], primary plane: 4
vp2 have layer nr:1[3 ], primary plane: 3
Using display timing dts
dsi@fe070000:  detailed mode clock 75000 kHz, flags[8000000a]
    H: 1080 1128 1136 1188
    V: 1920 1936 1942 1957
```

Figure 6.1.2.16 The operation process of U-Boot

6.1.3 Configuration of U-Boot

If the user wants to configure U-Boot themselves, they can execute the following commands in the U-Boot source code directory to do so:

```
make rk3568_defconfig      // Select configuration file
make menuconfig            //Open the graphical configuration interface
```

The running results are as follows:

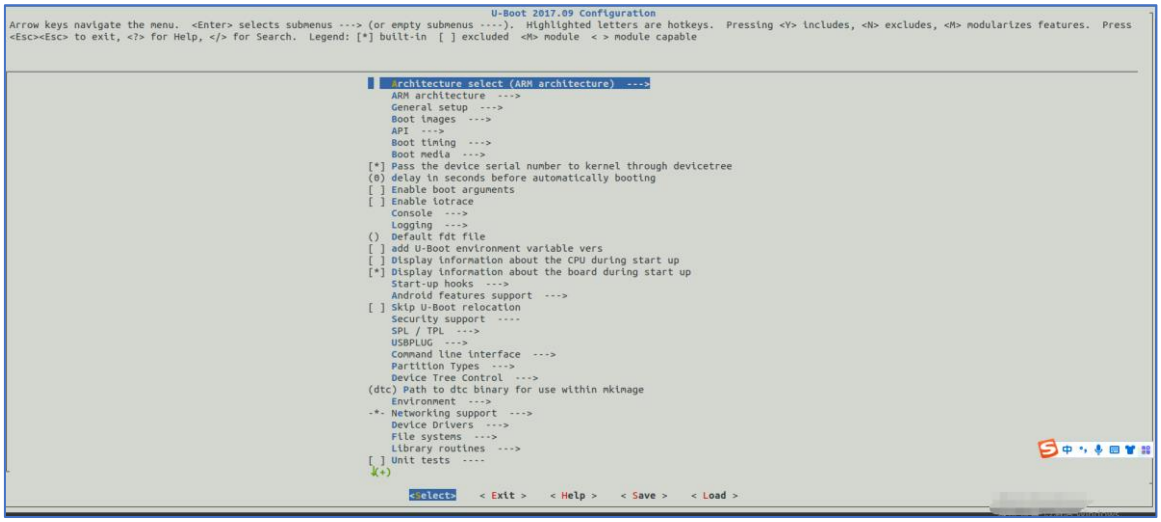


Figure 6.1.3.1 menuconfig Graphical configuration

Through the menuconfig graphical interface, users can configure U-Boot by themselves. After the configuration is completed, run the following command to save the configuration:

```
make savedefconfig                    // Save the configuration information to the defconfig file.
cp defconfig configs/rk3568_defconfig // Replace rk3568_defconfig with the defconfig file
```

After saving the configuration, we can directly execute the make.sh script in the U-Boot source code directory to compile the U-Boot source code, or we can return to the SDK root directory and execute the build.sh script for compilation:

```
./make.sh rk3568
// Execute the make.sh script in the U-Boot source code directory for compilation.
cd ../ && ./build.sh -U
//Execute the build.sh script in the SDK root directory for compilation
```

6.1.4 The defconfig configuration file

In U-Boot, the defconfig configuration file used for the RK3568 platform is: **<U-Boot>/configs/rk3568_defconfig**. Developers generally do not need to modify it. In fact, for the entire U-Boot, if your product has no special requirements, you basically do not need to touch the U-Boot source code.

If you do not use the make.sh script, directly configure U-Boot using the make command and then compile U-Boot, you can do it like this:

```
make rk3568_defconfig -j24
make PYTHON=python2 CROSS_COMPILE=<SDK>/prebuilts/gcc/linux-x86/aarch64/gcc-linaro-6.3.1-2017.05-x86_64_aarch64-linux-gnu/bin/aarch64-linux-gnu- all --jobs=24
```

<SDK> represents the directory of the SDK, please modify it according to your own actual path; this cross-compiler **<SDK>/prebuilts/gcc/linux-x86/aarch64/gcc-linaro-6.3.1-2017.05-x86_64_aarch64-linux-gnu/bin/aarch64-linux-gnu-** is used for compiling the kernel and U-Boot.

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot$ make rk3568_defconfig -j24
  HOSTCC  scripts/basic/fixdep
  HOSTCC  scripts/kconfig/conf.o
  SHIPPED scripts/kconfig/zconf.tab.c
  SHIPPED scripts/kconfig/zconf.lex.c
  SHIPPED scripts/kconfig/zconf.hash.c
  HOSTCC  scripts/kconfig/zconf.tab.o
In file included from scripts/kconfig/zconf.tab.c:2468:
scripts/kconfig/confdata.c: In function ‘conf_write’:
scripts/kconfig/confdata.c:771:19: warning: ‘%s’ directive writing likely 7 or more bytes into a region of size betwe
  771 |  sprintf(newname, "%s%s", dirname, basename);
      |                   ^~~~~~
scripts/kconfig/confdata.c:771:19: note: assuming directive output of 7 bytes
In file included from /usr/include/stdio.h:867,
                 from scripts/kconfig/zconf.tab.c:82:
/usr/include/x86_64-linux-gnu/bits/stdio2.h:36:10: note: ‘__builtin___sprintf_chk’ output 1 or more bytes (assuming 4
   36 |   return __builtin___sprintf_chk (__s, __USE_FORTIFY_LEVEL - 1,
      |          ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   37 |        __bos (__s), __fmt, __va_arg_pack ());
      |        ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
In file included from scripts/kconfig/zconf.tab.c:2468:
scripts/kconfig/confdata.c:774:20: warning: ‘.tmpconfig.’ directive writing 11 bytes into a region of size between 1
  774 |   sprintf(tmpname, "%s.tmpconfig.%d", dirname, (int)getpid());
      |                    ^~~~~~~~~~~~~~~~
In file included from /usr/include/stdio.h:867,
                 from scripts/kconfig/zconf.tab.c:82:
/usr/include/x86_64-linux-gnu/bits/stdio2.h:36:10: note: ‘__builtin___sprintf_chk’ output between 13 and 4119 bytes i
   36 |   return __builtin___sprintf_chk (__s, __USE_FORTIFY_LEVEL - 1,
      |          ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   37 |        __bos (__s), __fmt, __va_arg_pack ());
      |        ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  HOSTLD  scripts/kconfig/conf
#
# configuration written to .config
#

#### build completed successfully (1 seconds) ####
```

Figure 6.1.4.1 Configure U-Boot

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot$ make PYTHON=python2 CROSS_COMPILE=/home/tgg/rk3568_android_sdk/prebuilt
s/gcc/linux-x86/aarch64/gcc-linaro-6.3.1-2017.05-x86_64_aarch64-linux-gnu/bin/aarch64-linux-gnu- all --jobs=24
scripts/kconfig/conf  --silentoldconfig Kconfig
  CHK     include/config.h
  UPD     include/config.h
  CFG     u-boot.cfg
  GEN     include/autoconf.mk.dep
  CFG     spl/u-boot.cfg
  CFG     tpl/u-boot.cfg
  GEN     include/autoconf.mk
  GEN     spl/include/autoconf.mk
  GEN     tpl/include/autoconf.mk
  CHK     include/config/uboot.release
  CHK     include/generated/timestamp_autogenerated.h
  UPD     include/generated/timestamp_autogenerated.h
  HOSTCC  scripts/dtc/dtc.o
  HOSTCC  scripts/dtc/flattree.o
  HOSTCC  scripts/dtc/fstree.o
  HOSTCC  scripts/dtc/data.o
  HOSTCC  scripts/dtc/livetree.o
  HOSTCC  scripts/dtc/treesource.o
  HOSTCC  scripts/dtc/srcpos.o
  HOSTCC  scripts/dtc/checks.o
  HOSTCC  scripts/dtc/util.o
  SHIPPED scripts/dtc/dtc-lexer.lex.c
  SHIPPED scripts/dtc/dtc-parser.tab.h
```

```
  CC      spl/lib/hexdump.o
  CC      spl/drivers/mtd/nand/spi/etron.o
  CC      spl/drivers/mtd/nand/spi/jsc.o
  CC      spl/lib/uuid.o
  CC      spl/drivers/mtd/nand/spi/silicongo.o
  CC      spl/drivers/mtd/nand/spi/unim.o
  CC      spl/lib/tiny-printf.o
  CC      spl/lib/panic.o
  CC      spl/drivers/mtd/nand/spi/core.o
  CC      spl/lib/strto.o
  LD      spl/lib/built-in.o
  LD      spl/drivers/pinctrl/built-in.o
  LD      spl/drivers/mtd/nand/spi/spinand.o
  LD      spl/drivers/mtd/nand/spi/built-in.o
  LD      spl/drivers/mtd/nand/built-in.o
  LD      spl/drivers/mtd/built-in.o
  LD      spl/drivers/built-in.o
  LD      spl/u-boot-spl
  OBJCOPY spl/u-boot-spl-nodtb.bin
  CAT     spl/u-boot-spl-dtb.bin
  COPY    spl/u-boot-spl.bin
  CFGCHK  u-boot.cfg

#### build completed successfully (8 seconds) ####
```

Figure 6.1.4.2 Compile U-Boot

After the compilation is completed, uboot.img and rk356x_spl_loader_v1.13.112.bin were not generated because these two images are pack-compiled images that need to be generated through specialized tools or scripts. The make.sh script has already added the packaging operation, so after compiling with the make.sh script, uboot.img and rk356x_spl_loader_v1.13.112.bin will be generated.

Of course, we can execute the following commands to package and generate these two images:

```
./scripts/fit.sh --ini-loader ../rkbin/RKBOOT/RK3568MINIALL.ini --chip RK3568
```

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot$ ./scripts/fit.sh --ini-loader ../rkbin/RKBOOT/RK3568MINIALL.ini --chip RK3568
SEC=1
pack u-boot.itb okay! Input: /home/tgg/rk3568_android_sdk/rkbin/RKTRUST/RK3568TRUST.ini

FIT description: FIT Image with ATF/OP-TEE/U-Boot/MCU
Created:         Thu Apr 27 15:16:36 2023
 Image 0 (uboot)
  Description:  U-Boot
  Created:      Thu Apr 27 15:16:36 2023
  Type:         Standalone Program
  Compression:  uncompressed
  Data Size:    1267632 Bytes = 1237.92 KiB = 1.21 MiB
  Architecture: AArch64
  Load Address: 0x00a00000
  Entry Point:  unavailable
  Hash algo:    sha256
  Hash value:   4e930f9dac4bd8ddb0d091a2b8bc99264c554c6c8bdb15472cb4929a9db90588
 Image 1 (atf-1)
  Description:  ARM Trusted Firmware
  Created:      Thu Apr 27 15:16:36 2023
  Type:         Firmware
  Compression:  uncompressed
  Data Size:    163840 Bytes = 160.00 KiB = 0.16 MiB
  Architecture: AArch64
  Load Address: 0x00040000
  Hash algo:    sha256
  Hash value:   32fbfe613959f3a5a194b87d05b07db2b5b4fefa83791951f7f6d2659abc6c6c
 Image 2 (atf-2)
  Description:  ARM Trusted Firmware
  Created:      Thu Apr 27 15:16:36 2023
  Type:         Firmware
  Compression:  uncompressed
  Data Size:    40960 Bytes = 40.00 KiB = 0.04 MiB
  Architecture: AArch64
  Load Address: 0xfdcc1000
  Hash algo:    sha256
  Hash value:   9662321b37d2cd36d477550952c8c3c74144d95bee5a22d7fe54fbe154eb5a59
 Image 3 (atf-3)
  Description:  ARM Trusted Firmware
  Created:      Thu Apr 27 15:16:36 2023
  Type:         Firmware
  Compression:  uncompressed
  Data Size:    20267 Bytes = 19.79 KiB = 0.02 MiB
  Architecture: AArch64
```

Figure 6.1.4.3 Perform the packaging operation (1)

```
  Compression:  uncompressed
  Data Size:    14377 Bytes = 14.04 KiB = 0.01 MiB
  Architecture: AArch64
  Hash algo:    sha256
  Hash value:   56cff76c011fcefff4f52549b1ed4a1ba3f5cccd48430161b0d24da4ae89918a
 Default Configuration: 'conf'
 Configuration 0 (conf)
  Description:  rk3568-evb
  Kernel:       unavailable
  Firmware:     atf-1
  FDT:          fdt
  Loadables:    uboot
                atf-2
                atf-3
                atf-4
                atf-5
                atf-6
                optee
********boot_merger ver 1.2********
Info:Pack loader ok.
pack loader okay! Input: ../rkbin/RKBOOT/RK3568MINIALL.ini
/home/tgg/rk3568_android_sdk/u-boot

Image(no-signed, version=0): uboot.img (FIT with uboot, trust...) is ready
Image(no-signed): rk356x_spl_loader_v1.13.112.bin (with spl, ddr...) is ready
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot$
```

Figure 6.1.4.4 Perform the packaging operation (2)

If the operation is successful, two image files will be generated:

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot$ ls -lh uboot.img rk356x_spl_loader_v1.13.112.bin
-rw-rw-r-- 1 tgg tgg 455K 4月  27 15:16 rk356x_spl_loader_v1.13.112.bin
-rw-rw-r-- 1 tgg tgg 4.0M 4月  27 15:16 uboot.img
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot$
```

Figure 6.1.4.5 uboot.img and MiniLoaderAll.bin

The scripts/fit.sh script will invoke the **tools/mkimage** tool to generate uboot.img, and invoke the **tools/boot_merger** tool to generate rk356x_spl_loader_v1.13.112.bin. It also supports various other usages. You can explore them on your own.

If you need to clean up the U-Boot source code project, execute the "**make distclean**" command for the cleanup:

```
make distclean
```

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot$ make distclean
  CLEAN   dts/../arch/arm/dts
  CLEAN   dts
  CLEAN   examples/standalone
  CLEAN   tools
  CLEAN   tools/lib tools/common
  CLEAN   spl/arch spl/board spl/cmd spl/common spl/disk spl/drivers spl/dts spl/env spl/fs spl/lib spl/u-boot.cfg
pl.lds spl/u-boot-spl.map spl/u-boot-spl-nodtb.bin spl/u-boot-spl.sym tpl/arch tpl/board tpl/common tpl/disk tpl/d
 tpl/u-boot-tpl.map tpl/u-boot-tpl-nodtb.bin tpl/u-boot-tpl.sym
  CLEAN   u-boot-dtb.bin u-boot.lds u-boot.dtb u-boot.map u-boot-nodtb.bin u-boot.srec u-boot.bin u-boot u-boot.cf
 bl31_0xfdcd0000.bin bl31_0x00040000.bin bl31_0xfdcce000.bin bl31_0x0006a000.bin u-boot-nodtb.bin bl31_0xfdcc1000.
  CLEAN   scripts/basic
  CLEAN   scripts/dtc
  CLEAN   scripts/kconfig
  CLEAN   include/config include/generated spl tpl
  CLEAN   .config include/autoconf.mk include/autoconf.mk.dep include/config.h

#### build completed successfully (1 seconds) ####
```

Figure 6.1.4.6 Cleaning U-Boot

6.1.5 Shortcuts

On the RK platform, U-Boot supports triggering certain events through combination keys on the serial port for operations such as flashing and debugging. After RK U-Boot starts, there is no startup countdown (the countdown for booting the kernel), and it directly starts the kernel, skipping the countdown.

Of course, you can interrupt its kernel startup by pressing the serial port combination key "Ctrl + C", thereby entering the U-Boot command line mode. RK U-Boot supports the following serial port combination keys (**pressed for a long time during startup**):

- Ctrl + C: Enter U-Boot command line mode.
- Ctrl + D: Enter Loader flashing mode; in addition to the method introduced in Section 2.9.1 for entering the Loader mode, this is also a way to enter the Loader flashing mode.
- Ctrl + B: Enter Maskrom flashing mode; in addition to the method introduced in Section 2.9.1 for entering the Maskrom mode, this is also a way to enter the Maskrom flashing mode.
- Ctrl + F: Enter fastboot mode.
- Ctrl + M: Print bidram/system information.
- Ctrl + I: Enable kernel initcall_debug.
- Ctrl + P: Print cmdline information.
- Ctrl + S: Enter U-Boot command line after "Starting kernel...".

You can test it yourself. More details are not provided here.

6.1.6 HW-ID DTB

The RK platform U-Boot supports detecting the hardware status of GPIO or ADC to achieve dynamic loading of different kernels DTB. That is to say, it can find the DTB that matches the current hardware status from multiple Kernel DTB files based on the hardware status of GPIO or ADC; for example, a fixed ADC value or a fixed GPIO level. The RK calls this function **HW-ID DTB**.

For instance, the MIPI screen interface of the ALIENTEK ATK-DLRK3568 development board supports a 5.5-inch 720p MIPI screen (ALIENTEK 5.5-inch MIPI screen) and also a 5.5-inch 1080p MIPI screen (ALIENTEK 5.5-inch MIPI screen). However, the timing parameters and initialization parameters of these two screens are different. Therefore, two DTS files are needed to configure the two MIPI screens separately, which leads to a situation where: **when using the 720p MIPI screen, the 720p MIPI screen corresponding DTB needs to be burned, and when using the 1080p MIPI screen, the 1080p MIPI screen corresponding DTB needs to be burned.** This is very troublesome!

At this time, the HW-ID DTB can solve this trouble. First, we need to package the DTB files of the 720p MIPI screen and the 1080p MIPI screen into the same resource.img image. When U-Boot boots the kernel, it will detect the hardware status (ADC or GPIO), find the DTB that matches the current hardware status (a fixed ADC value or a fixed GPIO level) from the resource.img image, and load it.

1.Hardware Design Reference

Currently, two hardware states are supported for detection: ADC and GPIO.

On the MIPI screen interface of the ALIENTEK ATK-DLRK3568 development board, a SARADC_VIN2_LCD_ID pin has been designed:

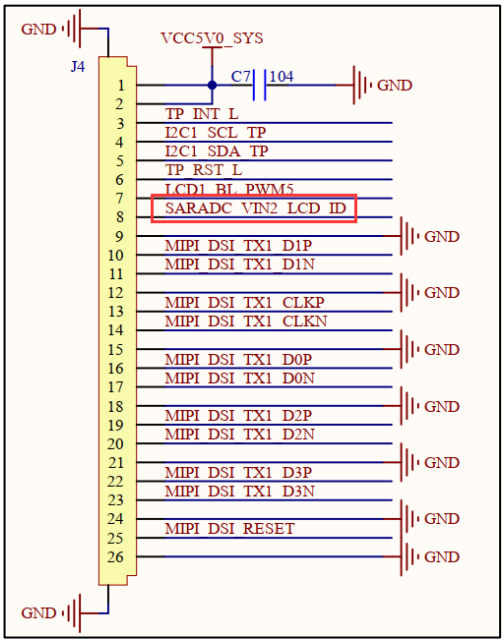
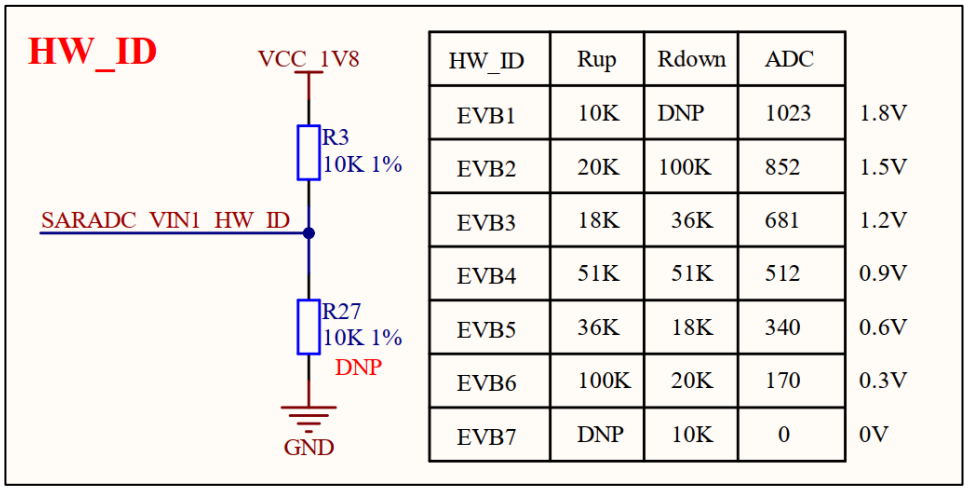


Figure 6.1.6.1 MIPI screen interface

SARADC_VIN2_LCD_ID is an ADC pin (using ADC channel 2), allowing users to connect either a 720p MIPI screen or a 1080p MIPI screen. The corresponding ADC values are different (the two screens use different voltage divider resistors in the hardware design). Before loading the kernel DTB by U-Boot, it reads this ADC value and finds the matching DTB based on the read ADC value for loading. Thus, DTB can be dynamically loaded according to the MIPI screen connected by the user (provided that the DTB files of both screens need to be fully packaged into resource.img).

In addition, on the ATK-DLRK3568 motherboard, a board-level HW_ID is reserved (for backup purposes):



HW_ID	Rup	Rdown	ADC	
EVB1	10K	DNP	1023	1.8V
EVB2	20K	100K	852	1.5V
EVB3	18K	36K	681	1.2V
EVB4	51K	51K	512	0.9V
EVB5	36K	18K	340	0.6V
EVB6	100K	20K	170	0.3V
EVB7	DNP	10K	0	0V

Figure 6.1.6.2 The HW_ID on the motherboard

On the baseboard, by welding different voltage divider resistors (using ADC 1 channel), it can be used to determine different hardware versions. Users can provide corresponding DTB files (one version corresponds to one DTB) on the software for different hardware versions. When users use different versions of hardware, the current version's corresponding DTB can be automatically loaded.

Currently, there is no hardware reference design for GPIO. Users can customize it themselves.

2. DTB Command Rules

Users need to include the unique hardware values of ADC/GPIO in the kernel DTB file name. The naming rule for kernel DTB is as follows:

ADC as HW_ID DTB

- The file name ends with.dtb;
- The file name is equal to "**custom part + #_[controller]_ch[channel]=[adcval]**", "#_[controller]_ch[channel]=[adcval]" is a complete unit, describing the unique hardware value information of the ADC:
  [controller]: The name of the ADC controller in the kernel device tree, usually saradc;
  [channel]: ADC channel, such as 0, 1, 2, 3, etc.;
  [adcval]: The theoretical calculated value of ADC, the actual valid range is adcval+-30.
- Each complete unit (#_[controller]_ch[channel]=[adcval]) must use lowercase letters and cannot have spaces inside;
- Multiple complete units are separated by a # sign, and the maximum number of units supported is 10. If a DTB needs to meet multiple unique hardware values of ADC to be matched, then the file name of this DTB will have multiple units.

Example:

rk3568-evb1-ddr4-v10#_saradc_ch2=111.dtb
rk3568-evb1-ddr4-v10#_saradc_ch2=111#_saradc_ch1=810.dtb

The first DTB file name indicates that when the ADC value read from ADC channel 2 is 111+-30, it will match this DTB; the second one indicates that when the ADC value read from ADC channel 2 is 111+-30 and the ADC value read from ADC channel 1 is 810+-30, it will match this DTB (condition 1 && condition 2).

GPIO as HW_ID DTB

- The file name ends with.dtb;
- The file name is equal to "**custom part + #_gpio[pin]=[level]**", "#_gpio[pin]=[level]" is a complete unit used to describe the unique value information of the GPIO hardware:
  [pin]: GPIO pin, for example 0a2 represents GPIO0_A2;
  [level]: the level of the GPIO pin, 0 or 1;
- Each complete unit (#_gpio[pin]=[level]) must use lowercase letters and cannot have spaces inside;
- Multiple complete units are separated by a # symbol, and the maximum number of units supported is 10. If a DTB needs to meet multiple unique values of the GPIO hardware to be matched, then the naming of this DTB file will have multiple units.

Example:

rk3568-evb1-ddr4-v10#_gpio0c1=1.dtb
rk3568-evb1-ddr4-v10#_gpio0c1=1#_gpio2b3=0.dtb

The first DTB file name indicates that when GPIO0_C1 is read as a high voltage, it will match this DTB; the second DTB file name indicates that it will match this DTB only when GPIO0_C1 is read as a high voltage and GPIO2_B3 is read as a low voltage (condition 1 && condition 2).

3. DTB Packaging

The <SDK>/kernel/scripts/mkmultidtb.py (script located in the kernel source code directory under scripts/mkmultidtb.py) can package multiple kernel DTBs into the same resource.img. For specific situations, please refer to Section 6.2.1.

4. Function Startup

The RK U-Boot does not enable the HW-ID DTB function by default, but ALIENTEK has enabled it. This is because the ATK-DLRK3568 development board hardware design requires the use of this function to adapt to two MIPI screens.

The HW-ID DTB can be enabled by setting the macro CONFIG_ROCKCHIP_HWID_DTB to 'y'. If the user does not need to use this function in their own product design, they can disable it. Open the <U-Boot>/configs/rk3568_defconfig file, find the following line, and comment it out! CONFIG_ROCKCHIP_HWID_DTB=y

As shown below:

```
CONFIG_ROCKCHIP_VENDOR_PARTITION=y
CONFIG_ROCKCHIP_FIT_IMAGE_PACK=y
CONFIG_ROCKCHIP_NEW_IDB=y
# CONFIG_ROCKCHIP_HWID_DTB=y
CONFIG_SPL_SERIAL_SUPPORT=y
CONFIG_SPL_DRIVERS_MISC_SUPPORT=y
CONFIG_TARGET_EVB_RK3568=y
CONFIG_SPL_LIBDISK_SUPPORT=y
```

Figure 6.1.6.3 Disable HW-ID DTB

6.2 kernel development

The Linux kernel source code is located in the **<SDK>/kernel** directory:

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel$ pwd
/home/tgg/rk3568_android_sdk/kernel
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel$ ls
android                          build.config.gki.aarch64          configs        Kconfig          README
arch                             build.config.gki-debug.aarch64    COPYING        kernel           samples
block                            build.config.gki-debug.x86_64     CREDITS        lib              scripts
boot.its                         build.config.gki_kasan            crypto         LICENSES         security
build.config.aarch64             build.config.gki_kasan.aarch64    Documentation  logo.bmp         sound
build.config.allmodconfig        build.config.gki_kasan.x86_64     drivers        logo_kernel.bmp  tests
build.config.allmodconfig.aarch64 build.config.gki_kprobes         firmware       MAINTAINERS      tools
build.config.allmodconfig.arm    build.config.gki_kprobes.aarch64  fs             Makefile         usr
build.config.allmodconfig.x86_64 build.config.gki_kprobes.x86_64   include        make.sh          virt
build.config.arm                 build.config.gki.x86_64           init           mm
build.config.common              build.config.x86_64               ipc            net
build.config.gki                 certs                             Kbuild         OWNERS
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel$
```

Figure 6.2-1 Kernel source code directory

This section introduces the content related to kernel development.

6.2.1 Kernel compilation

Before compiling, we first execute "**make distclean**" to clean up. In the kernel source code directory, there is a make.sh script file. You can directly use this script file to compile the kernel source code and run it directly as follows:

```
./make.sh
```

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel$ ./make.sh
  HOSTCC  scripts/basic/fixdep
  HOSTCC  scripts/kconfig/conf.o
  YACC    scripts/kconfig/zconf.tab.c
  LEX     scripts/kconfig/zconf.lex.c
  HOSTCC  scripts/kconfig/zconf.tab.o
  HOSTLD  scripts/kconfig/conf
#
# configuration written to .config
#
Using .config as base
Merging ./kernel/configs/android-11.config
Value of CONFIG_CIFS is redefined by fragment ./kernel/configs/android-11.config:
Previous value: CONFIG_CIFS=y
New value: # CONFIG_CIFS is not set

Value of CONFIG_CIFS_XATTR is redefined by fragment ./kernel/configs/android-11.config:
Previous value: CONFIG_CIFS_XATTR=y
New value: # CONFIG_CIFS_XATTR is not set

Value of CONFIG_CIFS_POSIX is redefined by fragment ./kernel/configs/android-11.config:
Previous value: CONFIG_CIFS_POSIX=y
New value: # CONFIG_CIFS_POSIX is not set

Value of CONFIG_CRYPTO_MD4 is redefined by fragment ./kernel/configs/android-11.config:
Previous value: CONFIG_CRYPTO_MD4=y
New value: # CONFIG_CRYPTO_MD4 is not set

Value of CONFIG_DEVMEM is redefined by fragment ./kernel/configs/android-11.config:
Previous value: CONFIG_DEVMEM=y
New value: # CONFIG_DEVMEM is not set

Value of CONFIG_SDCARD_FS is redefined by fragment ./kernel/configs/android-11.config:
Previous value: CONFIG_SDCARD_FS=y
New value: # CONFIG_SDCARD_FS is not set

Value of CONFIG_ANDROID_BINDERFS is redefined by fragment ./kernel/configs/android-11.config:
Previous value: # CONFIG_ANDROID_BINDERFS is not set
New value: CONFIG_ANDROID_BINDERFS=y
```

Figure 6.2.1.1 Compile the kernel (1)

```
  CC [M]  drivers/net/wireless/rockchip_wlan/rtl8852bs/phl/hal_g6/phy/rf/halrf_8852b/halrf_dpk_8852b.o
  CC [M]  drivers/net/wireless/rockchip_wlan/rtl8852bs/phl/hal_g6/phy/rf/halrf_8852b/halrf_set_pwr_table_8852b.o
  CC [M]  drivers/net/wireless/rockchip_wlan/rtl8852bs/phl/hal_g6/phy/rf/halrf_8852b/halrf_efuse_8852b.o
  CC [M]  drivers/net/wireless/rockchip_wlan/rtl8852bs/phl/hal_g6/phy/rf/halrf_8852b/halrf_tssi_8852b.o
  CC [M]  drivers/net/wireless/rockchip_wlan/rtl8852bs/phl/hal_g6/phy/rf/halrf_8852b/halrf_kfree_8852b.o
  CC [M]  drivers/net/wireless/rockchip_wlan/rtl8852bs/phl/hal_g6/phy/rf/halrf_8852b/halrf_psd_8852b.o
  LD [M]  drivers/net/wireless/rockchip_wlan/rtl8852bs/8852bs.o
  Building modules, stage 2.
  MODPOST 3 modules
  CC      drivers/media/usb/gspca/gspca_main.mod.o
  CC      drivers/net/wireless/rockchip_wlan/rkwifi/bcmdhd/bcmdhd.mod.o
  CC      drivers/net/wireless/rockchip_wlan/rtl8852bs/8852bs.mod.o
  LD [M]  drivers/media/usb/gspca/gspca_main.ko
  LD [M]  drivers/net/wireless/rockchip_wlan/rkwifi/bcmdhd/bcmdhd.ko
  LD [M]  drivers/net/wireless/rockchip_wlan/rtl8852bs/8852bs.ko
found ./arch/arm64/boot/dts/rockchip/.rk3568-atk-evb1-mipi-dsi-720p.dtb.dts.tmp
found ./arch/arm64/boot/dts/rockchip/.rk3568-atk-evb1-mipi-dsi-720p.dtb.dts.tmp
found ./arch/arm64/boot/dts/rockchip/.rk3568-atk-evb1-mipi-dsi-720p.dtb.dts.tmp
found ./arch/arm64/boot/dts/rockchip/.rk3568-atk-evb1-mipi-dsi-720p.dtb.dts.tmp
found ./arch/arm64/boot/dts/rockchip/.rk3568-atk-evb1-mipi-dsi-720p.dtb.dts.tmp
found ./arch/arm64/boot/dts/rockchip/.rk3568-atk-evb1-mipi-dsi-720p.dtb.dts.tmp
found ./arch/arm64/boot/dts/rockchip/.rk3568-atk-evb1-mipi-dsi-720p.dtb.dts.tmp
0
  Image:  resource.img (with rk3568-atk-evb1-mipi-dsi-720p.dtb logo.bmp logo_kernel.bmp) is ready
  Image:  boot.img (with Image  resource.img) is ready
  Image:  zboot.img (with Image.lz4  resource.img) is ready
##################################################
  DTC     arch/arm64/boot/dts/rockchip/rk3568-atk-evb1-ddr4-v10.dtb
  DTC     arch/arm64/boot/dts/rockchip/rk3568-atk-evb1-mipi-dsi-1080p.dtb
rk-kernel.dtb  rk3568-atk-evb1-mipi-dsi-720p#_saradc_ch2=0.dtb rk3568-atk-evb1-mipi-dsi-1080p#_saradc_ch2=341.dtb
Pack to resource.img successed!
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel$
```

Figure 6.2.1.2 Compile the kernel (2)

After the compilation is completed, two files, boot.img and resource.img, will be generated, as shown below:

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel$ ls -l boot.img resource.img
-rw-rw-r-- 1 tgg tgg 22786048 4月  12 08:46 boot.img
-rw-rw-r-- 1 tgg tgg   458240 4月  12 08:46 resource.img
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel$
```

Figure 6.2.1.3 The boot.img image and the resource.img image

Regarding these two images, I would also like to explain them to you.

1. boot.img image

The boot.img is an Android format boot image. It is composed of multiple images. For the RK3568 platform, the boot.img burned to the boot partition of the development board contains four parts: the kernel image Image, the kernel DTB, the ramdisk, and the resource.img. You can use the file command to view the boot.img file, as shown below:

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel$ file boot.img
boot.img: Android bootimg, kernel (0x10008000), second stage (0x10f00000), page size: 2048
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel$
```

Figure 6.2.1.4 View "boot.img" in the file.

"**Android bootimg**" indicates that it is an Android format boot image. However, it should be noted that the "boot.img" generated in the kernel source code directory after compiling through the make.sh script is not the final boot.img that is burned onto the boot partition of the development board. This is because this "boot.img" is not complete; it only contains the kernel image "Image" and "resource.img", and does not include the ramdisk. We can unpack it using the "scripts/unpack_bootimg" tool as follows:

```
mkdir out
./scripts/unpack_bootimg --boot_img ./boot.img --out ./out/
```

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel$ mkdir out
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel$ ./scripts/unpack_bootimg --boot_img ./boot.img --out ./out/
boot_magic: ANDROID!
kernel_size: 32094216
kernel load address: 0x10008000
ramdisk size: 0
ramdisk load address: 0x0
second bootloader size: 177664
second bootloader load address: 0x10f00000
kernel tags load address: 0x10000100
page size: 2048
os version: 0.0.0
os patch level: 2000-00
boot image header version: 0
product name:
command line args:
additional command line args:
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel$ ls -l out/
总用量 31520
-rw-rw-r-- 1 tgg tgg 32094216 4月  27 16:26 kernel
-rw-rw-r-- 1 tgg tgg        0 4月  27 16:26 ramdisk
-rw-rw-r-- 1 tgg tgg   177664 4月  27 16:26 second
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel$
```

Figure 6.2.1.5 Unpacking boot.img

The kernel is actually the kernel image arch/arm64/boot/Image.

Second is the resource image located in the kernel source code directory, named resource.img.

From the figure, it can be seen that the size of ramdisk is 0, indicating that this boot.img does not contain ramdisk (**ramdisk image is generated during the compilation of Android**).

2. resource.img image

resource.img is a type of image designed by RK, used to store some resources, such as u-boot logo, kernel logo, battery power images, and device tree image DTB. resource.img is not directly burned onto the development board, but is packaged into boot.img.

The <kernel>scripts/resource_tool tool can be used to generate the resource.img image. It can package one or more resources (DTB, image resources, etc.) into the resource.img image, as shown below:

```
mkdir out
cd out/
cp ../arch/arm64/boot/dts/rockchip/rk3568-atk-evb1-ddr4-v10.dtb ./
cp ../logo.bmp ../logo_kernel.bmp ./
../scripts/resource_tool logo.bmp logo_kernel.bmp rk3568-atk-evb1-ddr4-v10.dtb
```

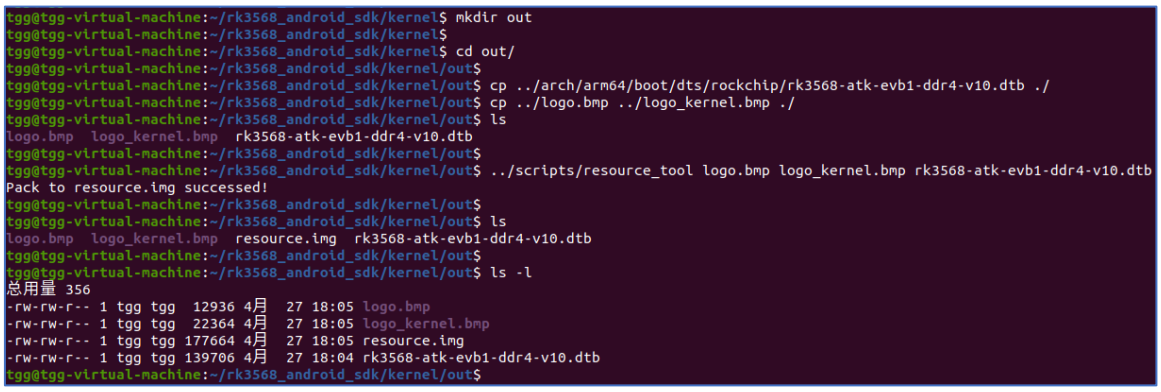


Figure 6.2.1.8 Create resource.img

In the example, we have included the u-boot logo image logo.bmp, the kernel logo image logo_kernel.bmp, and the kernel DTB into the resource.img image. The resource files carried after the resource_tool command cannot have paths, so the kernel DTB file and the logo image need to be copied to the current directory.

The scripts/resource_tool tool can also be used to unpack resource.img. Execute the following command to perform the unpacking operation:

```
rm -rf logo.bmp logo_kernel.bmp rk3568-atk-evb1-ddr4-v10.dtb
../scripts/resource_tool --unpack
```

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel/out$ pwd
/home/tgg/rk3568_android_sdk/kernel/out
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel/out$ ls
logo.bmp  logo_kernel.bmp  resource.img  rk3568-atk-evb1-ddr4-v10.dtb
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel/out$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel/out$ rm -rf logo.bmp logo_kernel.bmp rk3568-atk-evb1-ddr4-v10.dtb
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel/out$ ls
resource.img
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel/out$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel/out$ ../scripts/resource_tool --unpack
Dump header:
partition version:0.0
header size:1
index tbl:
        offset:1        entry size:1    entry num:3
Dump Index table:
entry(0):
        path:rk-kernel.dtb
        offset:4        size:139706
entry(1):
        path:logo_kernel.bmp
        offset:277      size:22364
entry(2):
        path:logo.bmp
        offset:321      size:12936
Unack resource.img to out successed!
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel/out$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel/out$ ls -l
总用量 180
drwxr-xr-x 2 tgg tgg   4096 4月  27 18:21 out
-rw-rw-r-- 1 tgg tgg 177664 4月  27 18:19 resource.img
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel/out$ ls -l out/
总用量 180
-rw-rw-r-- 1 tgg tgg  12936 4月  27 18:21 logo.bmp
-rw-rw-r-- 1 tgg tgg  22364 4月  27 18:21 logo_kernel.bmp
-rw-rw-r-- 1 tgg tgg 139706 4月  27 18:21 rk-kernel.dtb
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel/out$
```

Figure 6.2.1.9 Unpacking resource.img

After unpacking, we obtain the u-boot logo image logo.bmp, the kernel logo image logo_kernel.bmp, and a DTB file; because when creating resource.img, the DTB file is renamed to rk-kernel.dtb.

Users can package multiple kernel DTBs into the same resource.img image. For example, the resource.img used in the factory system of the ATK-DLRK3568 development board contains two kernel DTBs, to adapt to the two MIPI screens of the ALIENTEK (5.5-inch 720p and 5.5-inch 1080p). These two DTBs are: arch/arm64/boot/dts/rockchip/rk3568-atk-evb1-mipi-dsi-720p.dtb and arch/arm64/boot/dts/rockchip/rk3568-atk-evb1-mipi-dsi-1080p.dtb.

Because the RK platform U-Boot supports the HW-ID DTB function, in U-Boot, the corresponding kernel DTB can be dynamically loaded according to the MIPI screen connected by the user; if the connected MIPI screen is 720p, then load rk3568-atk-evb1-mipi-dsi-720p.dtb; if the connected MIPI screen is 1080p, then load rk3568-atk-evb1-mipi-dsi-1080p.dtb.

When creating a resource.img that contains multiple cores, the scripts/mkmultidtb.py script can be used instead of directly using the scripts/resource_tool tool, which is more convenient; the scripts/mkmultidtb.py script still creates the resource.img through the scripts/resource_tool tool, but before that, it will regularize the DTB file names, which is required by the U-Boot HW-ID DTB. This has been explained in Section 6.1.5 earlier.

Before using the scripts/mkmultidtb.py script to create the resource.img, developers need to configure this script in advance. Open the file:

```
DTBS['PX30-EVB'] = OrderedDict([('px30-evb-ddr3-v10', '#_saradc_ch0=1024'),
                                ('px30-evb-ddr3-lvds-v10', '#_saradc_ch0=512')])

DTBS['RK3308-EVB'] = OrderedDict([('rk3308-evb-dmic-i2s-v10', '#_saradc_ch3=288'),
                                  ('rk3308-evb-dmic-pdm-v10', '#_saradc_ch3=1024'),
                                  ('rk3308-evb-amic-v10', '#_saradc_ch3=407')])

DTBS['RK3568-ATK-EVB1'] = OrderedDict([('rk3568-atk-evb1-mipi-dsi-720p', '#_saradc_ch2=0'),
                                       ('rk3568-atk-evb1-mipi-dsi-1080p', '#_saradc_ch2=341')])
```

Figure 6.2.1.10 The mkmultidtb.py script

This file has already made three configurations. Users need to write the packaged DTB file into the DTBS dictionary. DTBS['RK3568-ATK-EVB1'] is the configuration made by ALIENTEK for the ATK-DLRK3568 development board. **If users need to define it themselves, they just need to follow the rules to configure**. RK3568-ATK-EVB1 represents the configuration name, while rk3568-atk-evb1-mipi-dsi-720p and rk3568-atk-evb1-mipi-dsi-1080p represent the names of the DTB files to be packaged (**without the .dtb suffix**). It will look for this DTB file in the arch/arm64/boot/dts/rockchip/ directory.

After processing by the mkmultidtb.py script:

rk3568-atk-evb1-mipi-dsi-720p.dtb---> rk3568-atk-evb1-mipi-dsi-720p#_saradc_ch2=0.dtb

rk3568-atk-evb1-mipi-dsi-1080p.dtb--->rk3568-atk-evb1-mipi-dsi-1080p#_saradc_ch2=341.dtb

Only after the DTB name is processed will it comply with the naming rules for DTB as specified by U-Boot HW-ID. Execute the following command to generate resource.img:

```
scripts/mkmultidtb.py RK3568-ATK-EVB1
```

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel$ scripts/mkmultidtb.py RK3568-ATK-EVB1
rk-kernel.dtb  rk3568-atk-evb1-mipi-dsi-720p#_saradc_ch2=0.dtb rk3568-atk-evb1-mipi-dsi-1080p#_saradc_ch2=341.dtb
Pack to resource.img successed!
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel$ ls -l resource.img
-rw-rw-r-- 1 tgg tgg 458240 4月  27 19:03 resource.img
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel$
```

Figure 6.2.1.11 Use mkmultidtb.py to create resource.img

We can unpack it using the scripts/resource_tool tool to have a look:

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel$ mkdir out
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel$ cd out/
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel/out$ cp ../resource.img ./
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel/out$ ls
resource.img
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel/out$ ../scripts/resource_tool --unpack
Dump header:                                   Unpack the "resource.img" file in the current directory.
partition version:0.0
header size:1
index tbl:
        offset:1        entry size:1    entry num:5
Dump Index table:
entry(0):
        path:rk-kernel.dtb
        offset:6        size:139710
entry(1):
        path:rk3568-atk-evb1-mipi-dsi-720p#_saradc_ch2=0.dtb
        offset:279      size:139710
entry(2):
        path:rk3568-atk-evb1-mipi-dsi-1080p#_saradc_ch2=341.dtb
        offset:552      size:139706
entry(3):
        path:logo_kernel.bmp
        offset:825      size:22364
entry(4):
        path:logo.bmp
        offset:869      size:12936
Unack resource.img to out successed!
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel/out$ ls
out  resource.img
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel/out$ ls out/*
 out/logo.bmp          'out/rk3568-atk-evb1-mipi-dsi-1080p#_saradc_ch2=341.dtb'   out/rk-kernel.dtb
 out/logo_kernel.bmp   'out/rk3568-atk-evb1-mipi-dsi-720p#_saradc_ch2=0.dtb'
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel/out$
```

Figure 6.2.1.12 Unpacking resource.img

After unpacking, it will be found that there are 3 DTB files:

- **rk-kernel.dtb:** A default DTB, which will be used as the default DTB when no DTB matches successfully. The mkmultidtb.py script in the packaging process will use the first DTB in the DTBS dictionary as the default DTB;
- **rk3568-atk-evb1-mipi-dsi-720p#_saradc_ch2=0.dtb:** A rk3568 DTB file containing ADC information. This DTB will be loaded only when the ADC value read from ADC channel 2 is within the range of 0+-30;
- **rk3568-atk-evb1-mipi-dsi-1080p#_saradc_ch2=341.dtb:** Another rk3568 DTB file containing ADC information. This DTB will be loaded only when the ADC value read from ADC channel 2 is within the range of 341+-30.

U-Boot must enable the HW-ID DTB function; otherwise, U-Boot will only load rk-kernel.dtb.

3. Generate the complete boot.img

After compiling through the make.sh script and generating the boot.img in the kernel source directory, this boot.img is not the final one that is burned onto the boot partition of the development board. This is because the images contained in this boot.img are not complete. As explained in Section 6.2.1.1, this has been clarified; for the RK3568 platform, the boot.img burned onto the development board's boot partition consists of four parts: ramdisk, kernel image Image, kernel DTB, and resource.img. It is formed by merging these four images.

In addition, the resource image file generated by compiling through the make.sh script, namely resource.img, does not contain all the resource files (missing the battery display image). For the RK3568 platform, resource.img should include the logo images (U-Boot startup logo, kernel startup logo), the kernel DTB, and the battery display image. Of course, it can also include other resources. Customers can package the required resources into resource.img according to their actual project needs.

The battery display images provided by RK are stored in the U-Boot source code directory, at the path: **<SDK>/u-boot/tools/images/**. As shown below:

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot$ pwd
/home/tgg/rk3568_android_sdk/u-boot
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot$ cd tools/images/
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot/tools/images$ ls
battery_0.bmp  battery_1.bmp  battery_2.bmp  battery_3.bmp  battery_4.bmp  battery_5.bmp  battery_fail.bmp
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot/tools/images$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot/tools/images$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot/tools/images$
```

Figure 6.2.1.13 Battery level images

So before packaging the resource.img into the boot.img, these battery level images need to be packaged into the resource.img; First, run the make.sh script to compile the kernel. After successful compilation, boot.img and resource.img will be generated in the kernel source code directory. Then, execute the following command to re-package the resource.img and also include the battery level images in the resource.img:

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

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel$ pwd
/home/tgg/rk3568_android_sdk/kernel
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel$ cd ../u-boot/
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot$ ./scripts/pack_resource.sh ../kernel/resource.img

Pack ./tools/images/ & ../kernel/resource.img to resource.img ...
Unpacking old image(../kernel/resource.img):
rk-kernel.dtb rk3568-atk-evb1-mipi-dsi-720p#_saradc_ch2=0.dtb rk3568-atk-evb1-mipi-dsi-1080p#_saradc_ch2=341.dtb logo_kern
el.bmp logo.bmp 5
Pack to resource.img successed!

Packed resources:
rk-kernel.dtb rk3568-atk-evb1-mipi-dsi-1080p#_saradc_ch2=341.dtb rk3568-atk-evb1-mipi-dsi-720p#_saradc_ch2=0.dtb battery_2
.bmp battery_3.bmp battery_4.bmp battery_5.bmp battery_fail.bmp logo.bmp logo_kernel.bmp battery_0.bmp battery_1.bmp 12

./resource.img with battery images is ready
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot$ cp resource.img ../kernel/
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/u-boot$ cd -
/home/tgg/rk3568_android_sdk/kernel
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel$ ls -l resource.img
-rw-rw-r-- 1 tgg tgg 508928 4月  27 20:23 resource.img
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel$
```

Figure 6.2.1.14 Re-package resource.img

Enter the U-Boot source code directory and execute the scripts/pack_resource.sh script to re-package resource.img; the pack_resource.sh script will unpack the specified resource.img, then re-package it, and pack all the original resources and all.bmp image resources in the <u-boot>/tools/images directory into a new resource.img. Finally, copy this newly re-packaged resource.img to the kernel source code directory to replace the old resource.img.

Note: In the above picture, the "Packed resources:" item shows the resource files contained in the newly re-packaged resource.img.

Then, return to the top-level directory of the Android SDK and execute the following command to generate the complete version of the boot.img:

```
make bootimage -j16
```

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel$ cd ..
tgg@tgg-virtual-machine:~/rk3568_android_sdk$
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ make bootimage -j16
============================================
PLATFORM_VERSION_CODENAME=REL
PLATFORM_VERSION=11
TARGET_PRODUCT=rk3568_r
TARGET_BUILD_VARIANT=userdebug
TARGET_BUILD_TYPE=release
TARGET_ARCH=arm64
TARGET_ARCH_VARIANT=armv8-a
TARGET_CPU_VARIANT=cortex-a55
TARGET_2ND_ARCH=arm
TARGET_2ND_ARCH_VARIANT=armv8-2a
TARGET_2ND_CPU_VARIANT=cortex-a55
HOST_ARCH=x86_64
HOST_2ND_ARCH=x86
HOST_OS=linux
HOST_OS_EXTRA=Linux-5.15.0-53-generic-x86_64-Ubuntu-20.04.2-LTS
HOST_CROSS_OS=windows
HOST_CROSS_ARCH=x86
HOST_CROSS_2ND_ARCH=x86_64
HOST_BUILD_TYPE=release
BUILD_ID=RD2A.211001.002
OUT_DIR=out
============================================
20:46:46 Disallowed PATH tool "python3.8" used: []string{"python3.8", ".repo/repo/repo", "manifest", "-r", "-o", "out/
20:46:46 See https://android.googlesource.com/platform/build/+/master/Changes.md#PATH_Tools for more information.
[100% 3/3] Target boot image: out/target/product/rk3568_r/boot.img

#### build completed successfully (3 seconds) ####
```

Figure 6.2.1.15 Generate boot.img

Execute the command "make bootimage -j16" in the <SDK> directory to compile Android and generate boot.img. During the compilation process, ramdisk.img will be generated. Then, ramdisk.img, kernel image Image, kernel DTB, and resource.img will be packaged into one boot.img. This boot.img will be output in the <SDK>/out/target/product/rk3568_r/ directory. Therefore, out/target/product/rk3568_r/boot.img is the final boot.img that will be burned onto the boot partition of the development board, not the boot.img in the kernel source code directory. This point must be noted!

We can use the unpack_bootimg tool to unpack this boot.img as follows:

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk$
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ cd out/target/product/rk3568_r/
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r$ ls -l boot.img
-rw-rw-r-- 1 tgg tgg 33572864 4月  27 20:46 boot.img
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r$ mkdir temp
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r$ unpack_bootimg --boot_img ./boot.img --out ./temp/
boot_magic: ANDROID!
kernel_size: 32094216
kernel load address: 0x10008000
ramdisk size: 822327
ramdisk load address: 0x11000000
second bootloader size: 508928
second bootloader load address: 0x10f00000
kernel tags load address: 0x10000100
page size: 2048
os version: 11.0.0
os patch level: 2021-10
boot image header version: 2
product name:
command line args: console=ttyFIQ0 androidboot.baseband=N/A androidboot.wificountrycode=CN androidboot.veritymode=enforcing
bootstate=orange firmware_class.path=/vendor/etc/firmware init=/init rootwait ro loop.max_part=7 buildvariant=userdebug
additional command line args:
recovery dtbo size: 0
recovery dtbo offset: 0x0
boot header size: 1660
dtb size: 139706
dtb address: 0x11f00000
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r$ cd te
bash: cd: te: 没有那个文件或目录
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r$ cd temp/
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp$ ls -l
总用量 32788
-rw-rw-r-- 1 tgg tgg   139706 4月  27 22:09 dtb
-rw-rw-r-- 1 tgg tgg 32094216 4月  27 22:09 kernel
-rw-rw-r-- 1 tgg tgg   822327 4月  27 22:09 ramdisk
-rw-rw-r-- 1 tgg tgg   508928 4月  27 22:09 second
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp$
```

This "second" refers to resources.img

Figure 6.2.1.16 Unpack the boot.img file

Filename	Explain

dtb	Kernel DTB. For the RK platform, this DTB is invalid. It is merely a decoration. The RK platform does not use this DTB. Instead, the DTB that actually takes effect is the one in the resource.img. Because the RK U-Boot directly reads the kernel DTB from the resource.img and loads it into memory.
kernel	Kernel image
ramdisk	ramdisk.img
second	This "second" file is actually the resource image named "resource.img". This "resource.img" contains the kernel DTB, logo image, and battery level image. The RK U-Boot will read the kernel DTB from "resource.img", rather than directly including this DTB in the "boot.img". This point must be clearly understood!

If the device tree file is modified, you need to recompile the device tree to obtain a new Kernel DTB. Then, package it into resource.img. After that, package resource.img into boot.img. Finally, burn the new boot.img into the boot partition of the development board. Although it is quite troublesome, this must be done! Unless you can modify the U-Boot code yourself and change this design.

boot.img is actually created by the mkbootimg tool. This tool is automatically generated when compiling the Android source code. You can also compile the mkbootimg tool by executing the following command:

```
make mkbootimg -j16
```

The boot.img unpacking tool, "unpack_bootimg", is also automatically generated during the compilation of the Android source code. Similarly, the "unpack_bootimg" tool can be compiled by executing the following command:

```
make unpack_bootimg -j16
```

After the compilation is successful, mkbootimg and unpack_bootimg will be output to the <SDK>/out/host/linux-x86/bin/ directory, as shown below:

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ cd out/host/linux-x86/bin/
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/host/linux-x86/bin$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/host/linux-x86/bin$ ls -l mkbootimg unpack_bootimg
-rwxrwxr-x 1 tgg tgg 31928699 4月  28 13:21 mkbootimg
-rwxrwxr-x 1 tgg tgg 31927699 4月  28 14:04 unpack_bootimg
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/host/linux-x86/bin$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/host/linux-x86/bin$
```

When compiling the Android source code, the <SDK>/out/host/linux-x86/bin/ directory will be added to the PATH environment variable of the Ubuntu system. Therefore, when using these two tools, there is no need to specify the full path.

In addition, there are also mkbootimg and unpack_bootimg tools in the kernel source code directory. These two tools are actually Python scripts, and their paths are: <kernel>/scripts/mkbootimg and <kernel>/scripts/unpack_bootimg. While the mkbootimg and unpack_bootimg compiled from the Android source code are ELF binary executable files, I'm not quite sure about the specific differences between them!

3. Compile kernel modules separately

Running the make.sh script will compile the entire kernel source code, including kernel modules. Of course, we can also compile kernel modules separately, which will be very useful when debugging

kernel modules! Before compiling kernel modules separately, the kernel source code needs to be compiled first. You can execute the following command to compile the kernel module:

```
make ARCH=arm64 modules -j24
```

```
CALL    scripts/checksyscalls.sh
CHK     include/generated/compile.h
LZ4C    arch/arm64/boot/Image.lz4
Image:  kernel.img is ready
CALL    scripts/checksyscalls.sh
CC [M]  drivers/net/wireless/marvell/mwifiex/main.o
CC [M]  drivers/net/wireless/marvell/mwifiex/init.o
CC [M]  drivers/net/wireless/marvell/mwifiex/cfp.o
CC [M]  drivers/net/wireless/marvell/mwifiex/cmdevt.o
CC [M]  drivers/net/wireless/rockchip_wlan/rkwifi/bcmdhd/aiutils.o
CC [M]  drivers/net/wireless/marvell/mwifiex/util.o
CC [M]  drivers/net/wireless/marvell/mwifiex/txrx.o
CC [M]  drivers/net/wireless/rockchip_wlan/rkwifi/bcmdhd/siutils.o
CC [M]  drivers/net/wireless/marvell/mwifiex/wmm.o
CC [M]  drivers/net/wireless/marvell/mwifiex/11n.o
CC [M]  drivers/net/wireless/rockchip_wlan/rkwifi/bcmdhd/sbutils.o
CC [M]  drivers/net/wireless/rockchip_wlan/rkwifi/bcmdhd/bcmutils.o
CC [M]  drivers/net/wireless/marvell/mwifiex/11ac.o
CC [M]  drivers/net/wireless/marvell/mwifiex/11n_aggr.o
CC [M]  drivers/net/wireless/marvell/mwifiex/11n_rxreorder.o
CC [M]  drivers/net/wireless/marvell/mwifiex/scan.o
CC [M]  drivers/net/wireless/marvell/mwifiex/join.o
CC [M]  drivers/net/wireless/rockchip_wlan/rtl8852bs/platform/platform_ARM_RK_sdio.o
CC [M]  drivers/net/wireless/marvell/mwifiex/sta_ioctl.o
CC [M]  drivers/net/wireless/marvell/mwifiex/sta_cmd.o
CC [M]  drivers/net/wireless/marvell/mwifiex/uap_cmd.o
CC [M]  drivers/net/wireless/marvell/mwifiex/ie.o
CC [M]  drivers/net/wireless/marvell/mwifiex/sta_cmdresp.o
CC [M]  drivers/net/wireless/marvell/mwifiex/sta_event.o
CC [M]  drivers/net/wireless/rockchip_wlan/rkwifi/bcmdhd/bcmwifi_channels.o
CC [M]  drivers/net/wireless/rockchip_wlan/rkwifi/bcmdhd/dhd_linux.o
CC [M]  drivers/net/wireless/rockchip_wlan/rkwifi/bcmdhd/dhd_linux_platdev.o
CC [M]  drivers/net/wireless/rockchip_wlan/rtl8852bs/os_dep/osdep_service.o
CC [M]  drivers/net/wireless/rockchip_wlan/rtl8852bs/os_dep/osdep_service_linux.o
CC [M]  drivers/net/wireless/rockchip_wlan/rtl8852bs/os_dep/linux/rtw_cfg.o
CC [M]  drivers/net/wireless/rockchip_wlan/rtl8852bs/os_dep/linux/os_intfs.o
CC [M]  drivers/net/wireless/rockchip_wlan/rkwifi/bcmdhd/dhd_linux_sched.o
CC [M]  drivers/net/wireless/rockchip_wlan/rkwifi/bcmdhd/dhd_pno.o
CC [M]  drivers/net/wireless/rockchip_wlan/rkwifi/bcmdhd/dhd_common.o
CC [M]  drivers/net/wireless/rockchip_wlan/rkwifi/bcmdhd/dhd_ip.o
CC [M]  drivers/net/wireless/rockchip_wlan/rkwifi/bcmdhd/dhd_linux_wq.o
CC [M]  drivers/net/wireless/marvell/mwifiex/uap_event.o
```

Figure 6.2.1.13 Compile the kernel module

Finally, a.ko file will be generated:

```
Building modules, stage 2.
MODPOST 4 modules
CC      drivers/net/wireless/marvell/mwifiex/mwifiex.mod.o
CC      drivers/net/wireless/marvell/mwifiex/mwifiex_sdio.mod.o
CC      drivers/net/wireless/rockchip_wlan/rkwifi/bcmdhd/bcmdhd.mod.o
CC      drivers/net/wireless/rockchip_wlan/rtl8852bs/8852bs.mod.o
LD [M]  drivers/net/wireless/rockchip_wlan/rtl8852bs/8852bs.ko
LD [M]  drivers/net/wireless/marvell/mwifiex/mwifiex.ko
LD [M]  drivers/net/wireless/rockchip_wlan/rkwifi/bcmdhd/bcmdhd.ko
LD [M]  drivers/net/wireless/marvell/mwifiex/mwifiex_sdio.ko
```

Figure 6.2.1.14 Generate .ko files

For the Android system on the RK3568 platform, the Linux kernel modules will be packaged into the vendor.img file, and the vendor.img file is not burned onto the development board alone; instead, it is packaged into the super.img file, and finally, the super.img file is burned onto the super partition of the development board.

6.2.2 Kernel Configuration

If the user wants to configure the kernel source code, they can execute the following command to open the menuconfig graphical configuration interface (execute it in the kernel source code directory):

```
make ARCH=arm64 rockchip_defconfig android-11.config        // Load configuration file
make ARCH=arm64 menuconfig
// Open the graphical configuration interface
```

The running results are as follows:

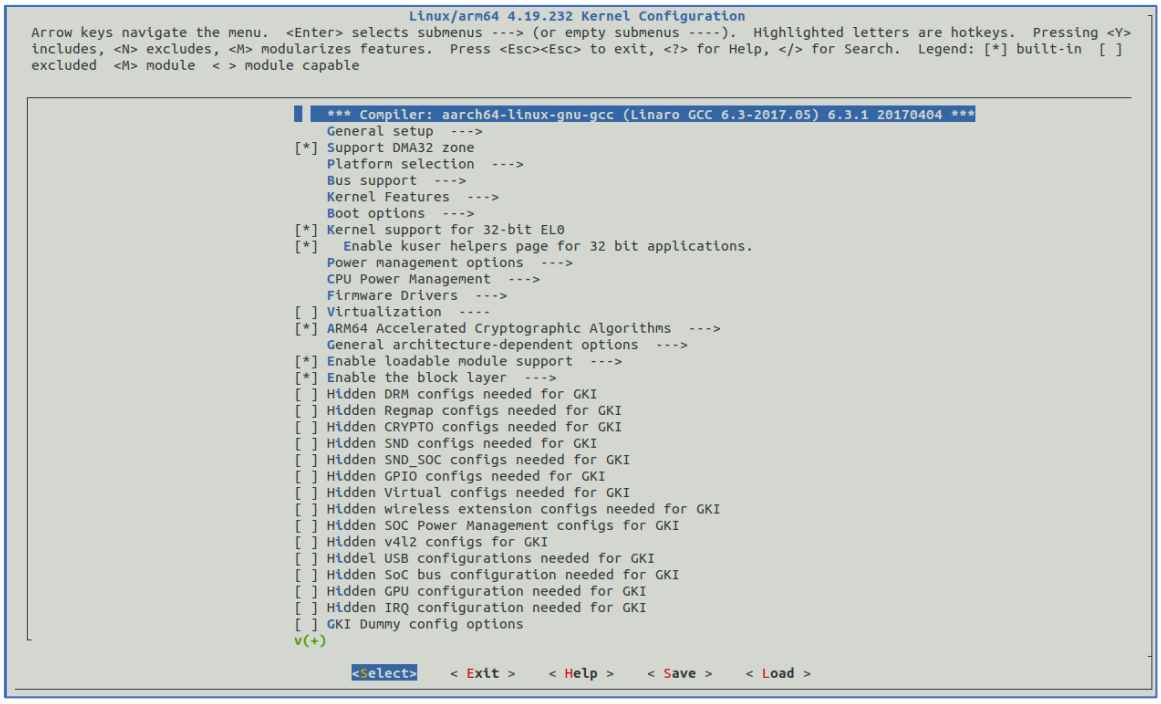


Figure 6.2.2.1 The menuconfig graphical configuration interface

Through the menuconfig graphical interface, users can configure the kernel source code. After the configuration is completed, the following command can be executed to save the configuration:

```
make ARCH=arm64 savedefconfig
cp defconfig arch/arm64/configs/rockchip_defconfig
```

After saving the configuration, we can now return to the SDK root directory and execute the build.sh script to compile the kernel source code:

```
cd ../                  // Return to the root directory of the SDK (currently located in the kernel source code directory)
./build.sh -K           // Execute the build.sh script to compile the kernel
make bootimage          // Package and generate boot.img. The generated image file will be saved in the product output directory.
```

6.2.3 Device tree and defconfig configuration file

The defconfig configuration file used by the ALIENTEK ATK-DLRK3568 development board is **<kernel>/arch/arm64/configs/rockchip_defconfig**. Users can modify this file, enable or disable kernel modules according to their own needs.

All device tree files for the Rockchip platform are stored in the **<kernel>arch/arm64/boot/dts/rockchip/** directory. For the ATK-DLRK3568 development board, there are 3 top-level device tree files: **rk3568-atk-evb1-ddr4-v10.dts**, **rk3568-atk-evb1-mipi-dsi-1080p.dts**, and **rk3568-atk-evb1-mipi-dsi-720p.dts**.

Of course, these three top-level device trees all contain multiple.dtsi device trees, as shown below.

rk3568-atk-evb1-ddr4-v10.dts
- rk3568-atk-evb1-ddr4-v10.dtsi
  - rk3568.dtsi
  - rk3568-evb.dtsi
- rk3568-android.dtsi
- rk3568-screen_choose.dtsi
- rk3568-lcds.dtsi

rk3568-atk-evb1-mipi-dsi-720p.dts
- rk3568-atk-evb1-ddr4-v10.dtsi
  - rk3568.dtsi
  - rk3568-evb.dtsi
- rk3568-android.dtsi
- rk3568-lcds.dtsi

rk3568-atk-evb1-mipi-dsi-1080p.dts
- rk3568-atk-evb1-ddr4-v10.dtsi
  - rk3568.dtsi
  - rk3568-evb.dtsi
- rk3568-android.dtsi
- rk3568-lcds.dtsi

- rk3568.dtsi: This device tree file is the platform-level device tree file for the RK3568 platform. It is independent of the specific development board hardware and is a pure SoC-level device tree file provided by RK. Developers do not need to modify this file!
- rk3568-evb.dtsi: This board-level general device tree file is usually included in the board-level device tree files;
- rk3568-android.dtsi: This device tree contains the unique configuration information specific to Android. If it is a Linux platform, then use rk3568-linux.dtsi;
- rk3568-atk-evb1-ddr4-v10.dtsi: This board-level device tree file. This device tree file includes the board-level general device tree rk3568-evb.dtsi and the platform-level device tree rk3568.dtsi;
- rk3568-screen_choose.dtsi: This device tree is used to select the LCD screens that need to be enabled, such as HDMI, MIPI, LVDS, VGA, supporting single display, dual display, and triple display. Users can define the ATK_LCD_TYPE_XXX macro in this file (for detailed information, please refer to the file) to enable the corresponding screen, for example, the ATK_LCD_TYPE_HDMI macro indicates enabling HDMI, and not defining this macro indicates disabling;
- rk3568-lcds.dtsi: This device tree implements the logic for enabling LCD screens controlled by the ATK_LCD_TYPE_XXX macro. The rk3568-lcds.dtsi device tree enables the corresponding display interface module and the related peripherals related to this screen based on the user-defined ATK_LCD_TYPE_XXX macro. For example, the touch screen

(only enabled when MIPI or LVDS is enabled), HDMI audio (only enabled when HDMI is enabled), and backlight (only enabled when MIPI or LVDS is enabled).

The files rk3568-atk-evb1-mipi-dsi-720p.dts and rk3568-atk-evb1-mipi-dsi-1080p.dts are the device trees used by ALIENTEK to compile the factory system image. Since the factory system of the ATK-DLRK3568 development board needs to be compatible with two different MIPI screens (ALIENTEK 5.5-inch 720p MIPI screen and 1080p MIPI screen), the initialization parameters and timing parameters of these two MIPI screens are different. Therefore, they need to be configured separately: rk3568-atk-evb1-mipi-dsi-720p.dts contains the parameter configuration for the 720p MIPI screen, and rk3568-atk-evb1-mipi-dsi-1080p.dts contains the parameter configuration for the 1080p MIPI screen. When compiling the factory system image, these two device trees will be compiled to obtain two DTB files, and both of these DTBs will be packaged into resource.img. The U-Boot code will determine whether the user is accessing a 720p MIPI screen or a 1080p MIPI screen (the unique value of the ADC hardware, refer to Section 6.1.5.1), and then read the corresponding DTB from resource.img and load it into memory.

The device tree rk3568-atk-evb1-ddr4-v10.dts is provided for customers and users only need to compile this device tree when compiling the kernel source code!

1. rk3568-atk-evb1-ddr4-v10.dts

rk3568-atk-evb1-ddr4-v10.dts is the device tree file corresponding to the ATK-DLRK3568 development board.

The rk3568-atk-evb1-ddr4-v10.dts device tree includes rk3568-screen_choose.dtsi. Users can modify the rk3568-screen_choose.dtsi file themselves to select the LCD screens that need to be enabled, including HDMI, MIPI, LVDS, and eDP_VGA. The content of this file is as follows:

```
/*
 * 屏幕选择
 * ATK_LCD_TYPE_MIPI_720P:   正点原子5.5寸 720*1280 MIPI屏
 * ATK_LCD_TYPE_MIPI_1080P:  正点原子5.5寸 1080*1920 MIPI屏
 * ATK_LCD_TYPE_LVDS:        正点原子10.1寸 1280*800 LVDS屏
 * ATK_LCD_TYPE_HDMI:        HDMI显示器
 * ATK_LCD_TYPE_EDP_VGA:     eDP屏或者VGA显示器（硬件默认使能的是VGA接口，若用户需要使用eDP屏，则需修改硬件
 *                           具体情况可以看正点原子RK3568底板原理图！）
 */
/*
 * RK3568可支持三屏显示，也就是三路显示 VP0 VP1 VP2
 * 但是三屏显示需要注意一些问题，具体情况可以看正点原子提供的文档<RK3568三屏显示参考手册>!
 */

/*
 * ATK_LCD_TYPE_MIPI_720P 和 ATK_LCD_TYPE_MIPI_1080P 二选一
 */
#define ATK_LCD_TYPE_MIPI_720P          // 从VP1输入
//#define ATK_LCD_TYPE_MIPI_1080P       // 从VP1输入

/*
 * ATK_LCD_TYPE_HDMI 和 ATK_LCD_TYPE_EDP_VGA 二选一
 */
#define ATK_LCD_TYPE_HDMI               // 从VP0输入
//#define ATK_LCD_TYPE_EDP_VGA          // 从VP0输入

//#define ATK_LCD_TYPE_LVDS             // 从VP2输入
```

Figure 6.2.3.1 rk3568-screen_choose.dtsi device tree

By default, only MIPI screen and HDMI are enabled.

If the make.sh script is not used, how to compile the Linux kernel using the make command? As shown below:

```
make ARCH=arm64 rockchip_defconfig android-11.config
make ARCH=arm64 Image -j24
```

```
make ARCH=arm64 rockchip/rk3568-atk-evb1-ddr4-v10.dtb -j24
```

The first step is to execute the defconfig configuration： make ARCH=arm64 rockchip_defconfig android-11.config

Step 2: Compile the kernel Image：make ARCH=arm64 Image -j24

Step 3: Compile the device tree： make ARCH=arm64 rockchip/rk3568-atk-evb1-ddr4-v10.dtb -j24

After successfully executing the above commands for compilation, you will obtain the kernel image Image (arch/arm64/boot/Image) and the Kernel DTB (arch/arm64/boot/dts/rockchip/rk3568-atk-evb1-ddr4-v10.dtb).

Executing the following command will package the Kernel DTB, logo image, and battery display image into resource.img:

```
cp arch/arm64/boot/dts/rockchip/rk3568-atk-evb1-ddr4-v10.dtb ./
./scripts/resource_tool logo.bmp logo_kernel.bmp rk3568-atk-evb1-ddr4-v10.dtb
cd ../u-boot/
./scripts/pack_resource.sh ../kernel/resource.img
cp resource.img ../kernel/
cd ../kernel/
rm -rf ./rk3568-atk-evb1-ddr4-v10.dtb
```

Figure 6.2.3.2 Package resource.img

The final step is to package Kernel DTB, resource.img (which contains Kernel DTB), the kernel image Image, and ramdisk.img into a single file named boot.img, as shown below:

```
cd ../
make bootimage -j16
```

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel$ pwd
/home/tgg/rk3568_android_sdk/kernel
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel$ cd ../
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ make bootimage -j16
============================================
PLATFORM_VERSION_CODENAME=REL
PLATFORM_VERSION=11
TARGET_PRODUCT=rk3568_r
TARGET_BUILD_VARIANT=userdebug
TARGET_BUILD_TYPE=release
TARGET_ARCH=arm64
TARGET_ARCH_VARIANT=armv8-a
TARGET_CPU_VARIANT=cortex-a55
TARGET_2ND_ARCH=arm
TARGET_2ND_ARCH_VARIANT=armv8-2a
TARGET_2ND_CPU_VARIANT=cortex-a55
HOST_ARCH=x86_64
HOST_2ND_ARCH=x86
HOST_OS=linux
HOST_OS_EXTRA=Linux-5.15.0-53-generic-x86_64-Ubuntu-20.04.2-LTS
HOST_CROSS_OS=windows
HOST_CROSS_ARCH=x86
HOST_CROSS_2ND_ARCH=x86_64
HOST_BUILD_TYPE=release
BUILD_ID=RD2A.211001.002
OUT_DIR=out
============================================
16:59:36 Disallowed PATH tool "python3.8" used: []string{"python3.8", ".repo/repo/repo", "manifest", "-r", "-o",
16:59:36 See https://android.googlesource.com/platform/build/+/master/Changes.md#PATH_Tools for more information.
[100% 1/1] Target boot image: out/target/product/rk3568_r/boot.img

#### build completed successfully (6 seconds) ####
```

Figure 6.2.3.3 Generate boot.img

First, return to the <SDK> directory, and then execute the command "make bootimage -j16".

2. Modify the make.sh script

Open the <kernel>/make.sh script file, and then modify it. The modified content is as follows:

```
#!/bin/bash

# Default
RK_ARCH=arm64
RK_DEFCONFIG="rockchip_defconfig android-11.config"
ADDON_ARGS=
RK_DTS=rk3568-atk-evb1-ddr4-v10
RK_JOBS=24

while [ $# -gt 0 ]; do
    case $1 in
        arch=*)
            arg=${1#*=}
            if [ -n "$arg" ]; then
                RK_ARCH=$arg
            fi
            shift 1
            ;;
        defconfig=*)
            arg=${1#*=}
            if [ -n "$arg" ]; then
                RK_DEFCONFIG=$arg
```

```
                fi
                shift 1
                ;;
            addon_arg=*)
                ADDON_ARGS=${1#*=}
                shift 1
                ;;
            dts=*)
                arg=${1#*=}
                if [ -n "$arg" ]; then
                    RK_DTS=$arg
                fi
                shift 1
                ;;
            jobs=*)
                arg=${1#*=}
                if [ -n "$arg" ]; then
                    RK_JOBS=$arg
                fi
                shift 1
                ;;
            *)
                shift 1
                ;;
        esac
    done

    # Configuration
    make $ADDON_ARGS ARCH=$RK_ARCH $RK_DEFCONFIG
    # Compile
    make $ADDON_ARGS ARCH=$RK_ARCH $RK_DTS.img -j$RK_JOBS
```

By default, when using the make.sh script to compile the Linux kernel, the generated resource image resource.img will not contain rk3568-atk-evb1-ddr4-v10.dtb. Therefore, the rk3568-atk-evb1-ddr4-v10.dts device tree configuration will not take effect.

6.2.4 IO Power Domain

This section content is referenced from the document:
<SDK>/RKDocs/common/IODomain/Rockchip_Developer_Guide_Linux_IO_DOMAIN_CN.pdf
<SDK>/RKDocs/common/IODomain/Rockchip_RK356X_Introduction_IO_Power_Domains_Configuration.pdf

The RK3568 has a total of 10 independent IO power domains, namely PMUIO[0:2] and VCCIO[1:7]. Here are the details:

- PMUIO0 and PMUIO1 are fixed-level power domains and cannot be configured;
- The power domains of PMUIO2, VCCIO1, and VCCIO[3:7] all require the hardware supply voltage to match the software configuration:

(1) When the hardware IO level is 1.8V, the software voltage configuration should also be set to 1.8V;

(2) When the hardware IO level is 3.3V, the software voltage configuration should also be set to 3.3V;

- For the VCCIO2 power domain, the software does not need to be configured, but the hardware supply voltage must be consistent with the FLASH_VOL_SEL state:

(1) When VCCIO2 is powered by 1.8V, the FLASH_VOL_SEL pin must remain at a high level;

(2) When the VCCIO2 power supply is 3.3V, the FLASH_VOL_SEL pin must remain at a low level.

If the user fails to configure as per the above requirements:

- When the software is set to 1.8V and the hardware supplies 3.3V, the IO will be in an overvoltage state, and prolonged operation will cause damage to the IO;
- When the software is set to 3.3V and the hardware supplies 1.8V, the IO function will be abnormal.

This section will explain how to configure the IO power domain, which is divided into 4 steps:

Step 1: Review the hardware schematic and confirm the supply voltage for each IO power domain;

Step 2: Modify the IO power domain configuration node pmu_io_domains in the kernel DTS;

Step 3: During kernel compilation, verify the IO power domain;

Step 4: After compiling the kernel, check the configuration of the IO power domain.

1. Review the hardware schematic and confirm the supply voltage for each IO power domain.

Take the ALIENTEK ATK-DLRK3568 development board as an example. First, locate the hardware schematic of the ATK-DLRK3568 development board, open the core board schematic, and the path of the core board schematic is: Development Board CD-ROM A Disk - Basic Materials -> 2_sch -> 2_Core Board Schematic -> ATK-CLRK3568F V1.2 (Core Board Schematic).pdf.

According to the previous introduction, among the 10 IO power domains, PMUIO0 and PMUIO1 are fixed-level power domains and do not require configuration; for the power domains such as PMUIO2 and VCCIO1 as well as VCCIO[3:7], users need to make configurations.

On the core board schematic, search for "**VCCIO1**" to find the supply voltage of this power domain, as shown in the following figure:

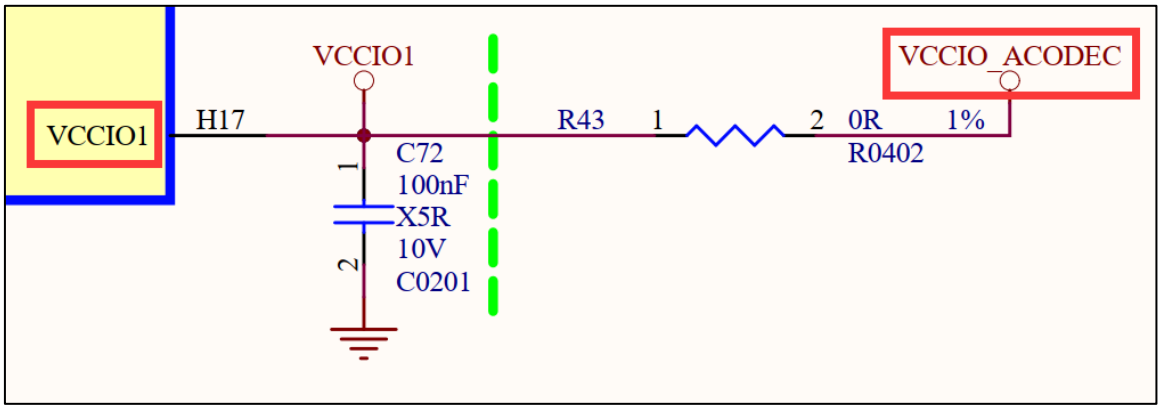


Figure 6.2.4.1 VCCIO1 power domain

As can be seen from the schematic, the power supply for VCCIO1 is VCCIO_ACODEC. Searching for "**VCCIO_ACODEC**" in the core board schematic:

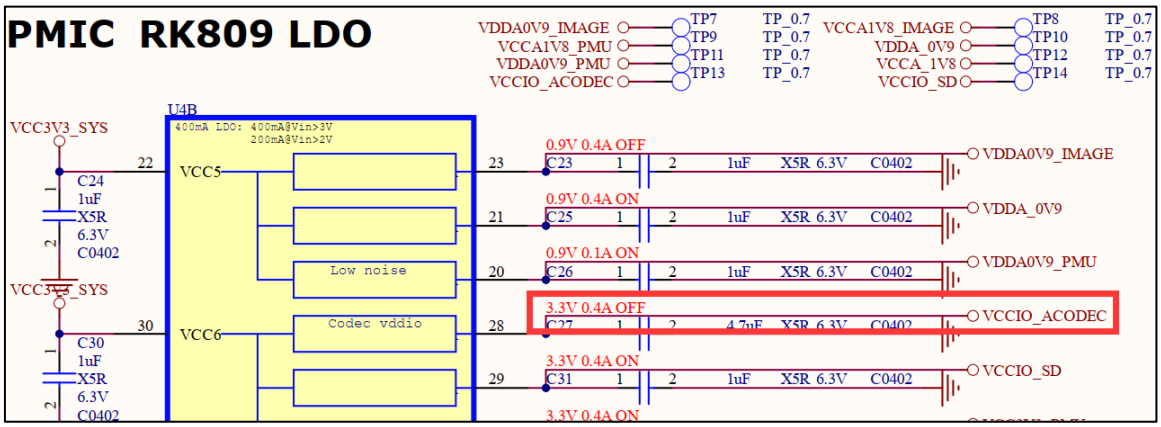


Figure 6.2.4.2 VCCIO_ACODEC power supply

Therefore, it can be concluded that VCCIO_ACODEC is powered by the LDO4 of RK809. The schematic diagram indicates that this power supply is 3.3V. Whether the actual voltage is 3.3V or not, this needs to be determined by the configuration of the device tree. Open the device tree file arch/arm64/boot/dts/rockchip/rk3568-evb.dtsi, and find the vccio_acodec node in the RK809 configuration information:

```
vccio_acodec: LDO_REG4 {
        regulator-always-on;
        regulator-boot-on;
        regulator-min-microvolt = <3300000>;
        regulator-max-microvolt = <3300000>;
        regulator-name = "vccio_acodec";
        regulator-state-mem {
                regulator-off-in-suspend;
        };
};
```

Figure 6.2.4.3 vccio_acodec node

In the device tree, the VCCIO_ACODEC power supply is configured to 3.3V, so the supply voltage of the VCCIO1 power domain is also 3.3V.

Now let's look for the PMUIO2 power domain. Search for "**PMUIO2**" on the schematic.

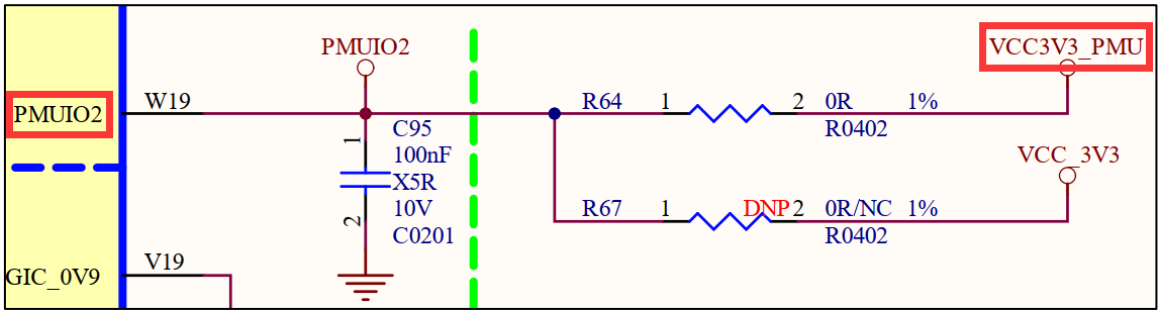


Figure 6.2.4.4 PMUIO2 power domain

As can be seen from the schematic, the power supply for PMUIO2 is VCC3V3_PMU. Searching for "**VCC3V3_PMU**" in the schematic:

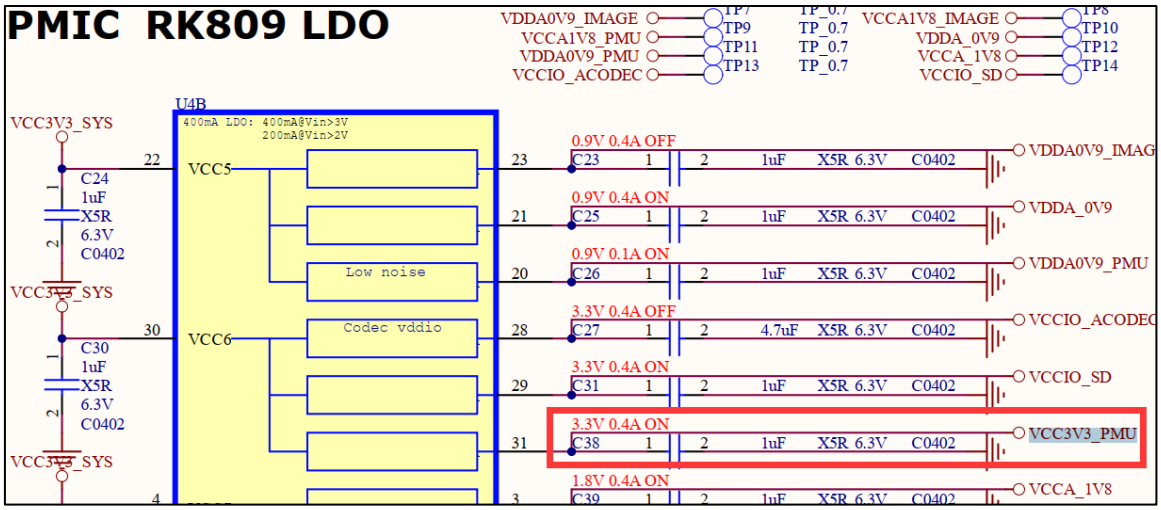


Figure 6.2.4.5 VCC3V3_PMU power supply

Therefore, it can be concluded that VCC3V3_PMU is powered by the LDO6 of RK809. Open the device tree file arch/arm64/boot/dts/rockchip/rk3568-evb.dtsi, and find the vcc3v3_pmu node in the RK809 configuration information:

```
vcc3v3_pmu: LDO_REG6 {
        regulator-always-on;
        regulator-boot-on;
        regulator-min-microvolt = <3300000>;
        regulator-max-microvolt = <3300000>;
        regulator-name = "vcc3v3_pmu";
        regulator-state-mem {
                regulator-on-in-suspend;
                regulator-suspend-microvolt = <3300000>;
        };
};
```

Figure 6.2.4.6 vcc3v3_pmu node

In the device tree, the VCC3V3_PMU power supply is configured as 3.3V, so the supply voltage of the PMUIO2 power domain is also 3.3V.

The supply voltage of other IO power domains is checked in the same way, so there is no need to elaborate further here!

Let's focus on the VCCIO2 power domain. On the software side, no configuration is required for the VCCIO2 power domain. However, there is a requirement in the hardware design: **when the supply voltage of the VCCIO2 power domain is 1.8V, the FLASH_VOL_SEL pin must remain at a high level and pulled up; when the supply voltage of the VCCIO2 power domain is 3.3V, the FLASH_VOL_SEL pin must remain at a low level and pulled down**. Search for "**VCCIO2**" in the schematic diagram.

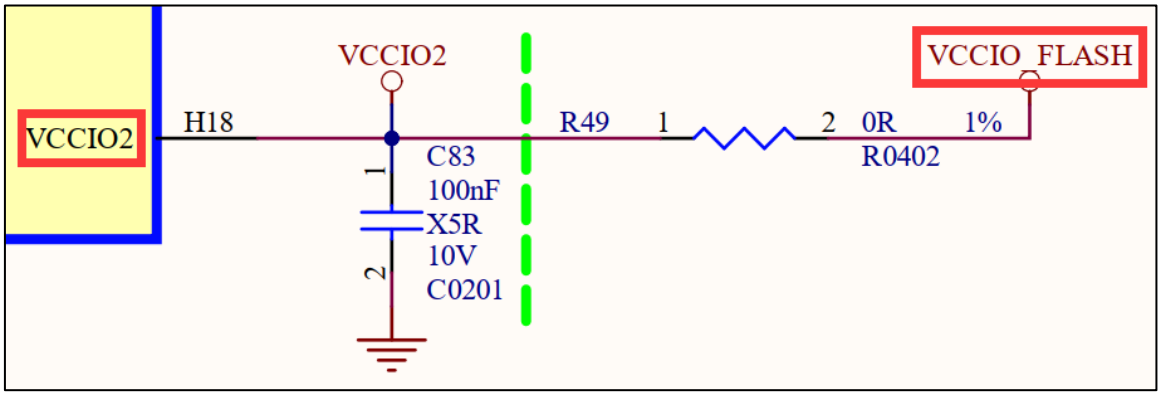


Figure 6.2.4.7 VCCIO2 power domain

As can be seen from the schematic, the power supply for VCCIO2 is VCCIO_FLASH. Search for "**VCCIO_FLASH**" in the schematic:

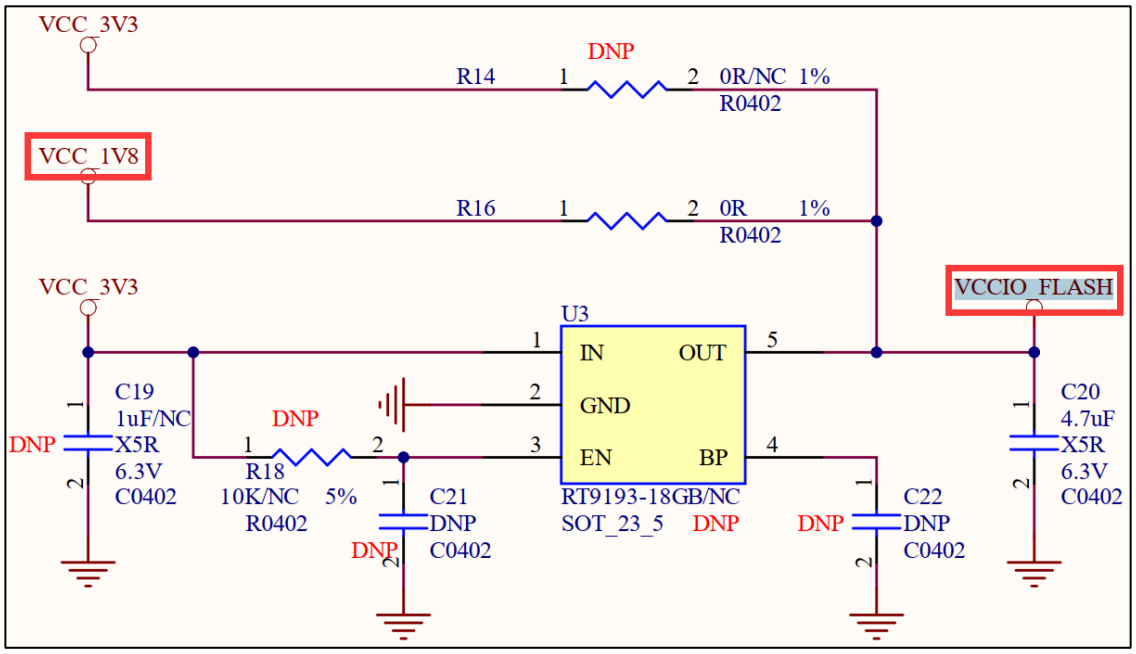


Figure 6.2.4.8 VCCIO_FLASH power supply

From the schematic diagram, it can be seen that the VCCIO_FLASH power supply comes from VCC_1V8, so it is known that the VCCIO_FLASH power supply is 1.8V. Finally, it can be determined that the supply voltage of the VCCIO2 power domain is 1.8V. Since this is the case, according to the design requirements, the FLASH_VOL_SEL pin must remain at a high level and must be pulled high. On the schematic diagram, find FLASH_VOL_SEL:

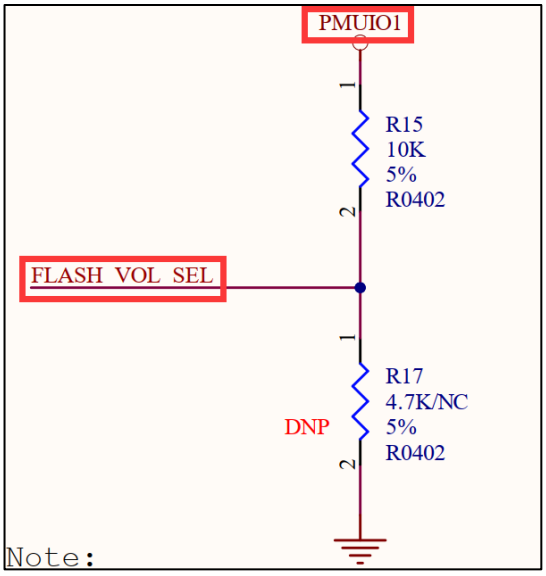


Figure 6.2.4.9 The voltage of the FLASH_VOL_SEL pin

From the schematic diagram, it can be seen that the FLASH_VOL_SEL pin has indeed been pulled up (PMUIO1 is 3.3V) and remains in a high-level state, meeting the aforementioned hardware design requirements.

2. Modify the IO power domain configuration node pmu_io_domains in the kernel DTS

After knowing the supply voltage of each IO power domain from the schematic diagram, the next step is to configure the device tree. The IO power domains are configured through the pmu_io_domains node in the device tree. First, find the IO power domain configuration node **pmu_io_domains** in the kernel device tree, and then modify it.

The pmu_io_domains node is defined in the rk3568.dtsi device tree:

```
pmugrf: syscon@fdc20000 {
        compatible = "rockchip,rk3568-pmugrf", "syscon", "simple-mfd";
        reg = <0x0 0xfdc20000 0x0 0x10000>;

        pmu_io_domains: io-domains {
                compatible = "rockchip,rk3568-pmu-io-voltage-domain";
                status = "disabled";
        };

        reboot_mode: reboot-mode {
                compatible = "syscon-reboot-mode";
                offset = <0x200>;
                mode-bootloader = <BOOT_BL_DOWNLOAD>;
                mode-charge = <BOOT_CHARGING>;
                mode-fastboot = <BOOT_FASTBOOT>;
                mode-loader = <BOOT_BL_DOWNLOAD>;
                mode-normal = <BOOT_NORMAL>;
                mode-recovery = <BOOT_RECOVERY>;
                mode-ums = <BOOT_UMS>;
                mode-panic = <BOOT_PANIC>;
                mode-watchdog = <BOOT_WATCHDOG>;
        };
};
```

Figure 6.2.4.10 The definition of the pmu_io_domains node

We cannot directly modify the rk3568.dtsi file. Instead, we need to make the modification through referencing (**&pmu_io_domains**). By default, RK configures it in the rk3568-evb.dtsi device tree. Open the rk3568-evb.dtsi file and locate the following position:

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

```

Figure 6.2.4.11 Configure pmu_io_domains

This is the default configuration information for RK. It is configured based on the hardware design of RK's own EVB board. There are some differences in power supply design compared to the power supply design of the ALIENTEK ATK-DLRK3568 development board. The main differences are **VCCIO4** and **VCCIO6**:

The power supply voltage of the VCCIO4 power domain on the ATK-DLRK3568 development board is 1.8V, coming from the power vcc_1v8; while the power supply voltage of the RK official board is 3.3V. The power supply voltage of the VCCIO6 power domain on the ATK-DLRK3568 development board is 1.8V, coming from the power vcc_1v8; while the power supply voltage of the RK official board is 3.3V.

These are some minor differences in the hardware design of the IO power domains between the ATK-DLRK3568 development board and the RK official board. Therefore, for the ATK-DLRK3568 development board, the pmu_io_domains node needs to be modified. It is not recommended to directly modify the rk3568-evb.dtsi file (as we have previously mentioned, rk3568-evb.dtsi is a board-level generic device tree file). We can modify it in the specific board-level configuration file, such as rk3568-atk-evb1-ddr4-v10.dtsi (ALIENTEK has already configured it):

```
&pmu_io_domains {
        vccio4-supply = <&vcc_1v8>;
        vccio6-supply = <&vcc_1v8>;
};

```

Figure 6.2.4.12 Configure pmu_io_domains

Configure the power supply of VCCIO4 as vcc_1v8, and configure the power supply of VCCIO6 as vcc_1v8.

The contents that need to be modified have already been configured by ALIENTEK. Here, we are just explaining how to configure the IO power domain. When users need to develop their own products and design their own hardware, they can configure these things by themselves.

3. Confirm the IO power domain when compiling the kernel

During the compilation of the RK kernel source code, an IO-Domains confirmation dialog box will pop up, as shown in the following figure:

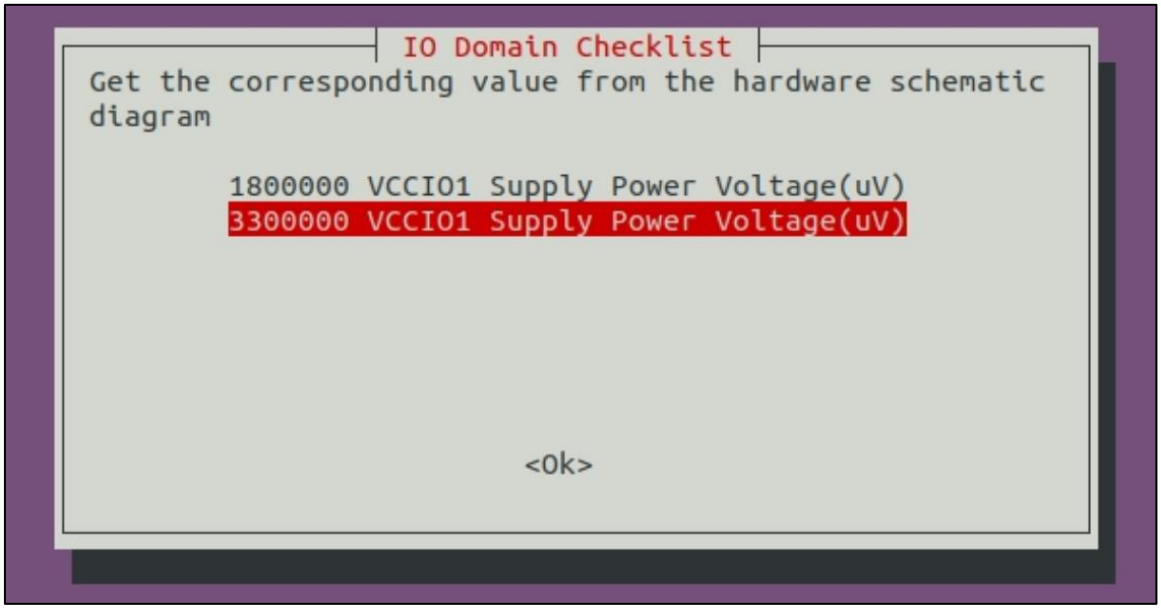


Figure 6.2.4.13 IO-Domains Confirmation Dialog Box

The purpose of this dialog box is to require developers to reconfirm whether the hardware schematic and the software DTS's IO voltage are matched. Although the pmu_io_domains node has been configured in the device tree, this dialog box will pop up when compiling the kernel, asking you to reconfirm. This is very important, please be sure to carefully confirm! If the IO voltage configuration is incorrect, it will cause the chip's IO to be burned.

After you complete the confirmation through this dialog box, the next compilation will not occur (compilation will pass only when the selected voltage value is the same as the configuration of the pmu_io_domains node in the device tree; otherwise, it will fail!). If the device tree name or the configuration of the pmu_io_domains node in the device tree has changed, the next compilation will continue to pop up this dialog box for reconfirmation.

4. Check the IO Power Domain Configuration after Compiling the Kernel

After the kernel is compiled successfully, a file starting with "." (a hidden file in the Linux system) and ending with "dtb.dts.tmp.domain" will be generated in the **<Kernel>/arch/arm64/boot/dts/rockchip/** directory, as shown in the figure:

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel/arch/arm64/boot/dts/rockchip$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel/arch/arm64/boot/dts/rockchip$ ls -l .*.tmp.domain
-rw-rw-r-- 1 tgg tgg 259 4月  23 16:13 .rk3568-atk-evb1-ddr4-v10.dtb.dts.tmp.domain
-rw-rw-r-- 1 tgg tgg 259 4月  27 16:12 .rk3568-atk-evb1-mipi-dsi-720p.dtb.dts.tmp.domain
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel/arch/arm64/boot/dts/rockchip$
```

Figure 6.2.4.14 The dtb.dts.tmp.domain file

The content of this file is the configuration of the IO voltage (the result selected by the user through the IO-Domains dialog box, and summarized), as shown in the following figure:

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel/arch/arm64/boot/dts/rockchip$ cat .rk3568-atk-evb1-ddr4-v10.dtb.dts.tmp.domain
PMUIO2 Supply Power Voltage1:3300000
VCCIO1 Supply Power Voltage1:3300000
VCCIO3 Supply Power Voltage1:3300000
VCCIO4 Supply Power Voltage1:1800000
VCCIO5 Supply Power Voltage1:3300000
VCCIO6 Supply Power Voltage1:1800000
VCCIO7 Supply Power Voltage1:3300000
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel/arch/arm64/boot/dts/rockchip$
```

Figure 6.2.4.15 The content of the dtb.dts.tmp.domain file

If this kind of dtb.dts.tmp.domain file is deleted, then when compiling the kernel next time, the IO-Domains dialog box will pop up again.

6.2.5 Replace the logo

Including the logo at the u-boot stage and the logo at the kernel stage. These two logo files are all located in the kernel source code directory:

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel$ pwd
/home/tgg/rk3568_android_sdk/kernel
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel$ ls -l *.bmp
-rw-rw-r-- 1 tgg tgg 177700 4月  28 15:24 logo.bmp
-rw-rw-r-- 1 tgg tgg 177764 4月  28 15:24 logo_kernel.bmp
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel$
```

Figure 6.2.5.1 Logo image file

The default logo is as follows:

Figure 6.2.5.2 U-Boot logo

Figure 6.2.5.3 Kernel logo

If the user needs to replace it with their own logo, they first need to rename your U-Boot logo image to "logo.bmp" and your kernel logo image to "logo_kernel.bmp". Then, copy these two bmp image files to the kernel source code directory and replace the default logo images in the kernel source code.

When compiling the kernel, logo.bmp and logo_kernel.bmp will be packaged into resource.img, and then resource.img will be packaged into boot.img.

When U-Boot starts, these two bmp files will be loaded into memory. logo.bmp will start to be displayed during the U-Boot stage, and the address of logo_kernel.bmp in memory will be passed to the Linux Kernel by U-Boot during the initialization stage of the DRM driver in the Linux Kernel.

6.2.6 Summary of Kernel Module Development Documents

In the **<SDK>/RKDocs/common** directory, there are many kernel module development documents provided by RK. These documents are divided according to functional modules and placed in different directories for easy user search. If you encounter some problems during the development process, you can check the development documents provided by RK. I believe they will be helpful to you!

The following table summarizes these documents:

Module Function	Subdirectory	Corresponding document
Audio	Audio	Rockchip_Developer_Guide_Audio_CN.pdf Rockchip_Developer_Guide_RK817_RK809_Codec_CN.pdf Rockchip_Developer_Guide_Audio_Call_3A_Algorithm_Integration_and_Parameter_Debugging_CN.pdf
Camera	camera	The document is stored in the subdirectories of common, HAL1, and HAL3.
CAN	Can	Rockchip_Develop_Guide_CAN_FD_CN.pdf
Clock configuration	CLK	Rockchip_Developer_Guide_Clock_CN.pdf Rockchip_Develop_Guide_Pll_Ssmod_Clock_CN.pdf Rockchip_RK3399_Developer_Guide_Clock_CN.pdf
DDR	DDR	Rockchip-Developer-Guide-DDR-CN.pdf Rockchip-Developer-Guide-DDR-Problem-Solution-CN.pdf Rockchip-Developer-Guide-DDR-Verification-Process-CN.pdf Rockchip-User-Guide-DDR-DQ-Eye-Tool-CN.pdf
Common debugging tools such as JTAG/GDB	debug	Rockchip_Developer_Guide_DS5_CN.pdf Rockchip_Quick_Start_Linux_Perf.pdf Rockchip_Quick_Start_Linux_Streamline.pdf Rockchip_Quick_Start_Linux_Systrace.pdf Rockchip_User_Guide_J-Link_CN.pdf Rockchip_RK3399_JTAG_Configuration_CN.pdf
Display	display	Rockchip_Developer_Guide_DRM_Display_Driver_CN.pdf Rockchip_Developer_Guide_DRM_Panel_Porting_CN&EN.pdf Rockchip_Developer_Guide_DRM_Panel_Porting_CN.pdf Rockchip_Developer_Guide_HDMI_Based_on_DRM_Framework_CN&EN.pdf Rockchip_Developer_Guide_HDMI-CEC_CN.pdf Rockchip_Developer_Guide_HDMI_CN.pdf
Frequency and voltage	DVFS	Rockchip_Developer_Guide_CPUFreq_CN.pdf Rockchip_Developer_Guide_Devfreq_CN.pdf

adjustment of CPU/GPU, etc.		Rockchip_Developer_Guide_Linux4.4_CPUFreq_CN.pdf
Ethernet configuration	GMAC	Rockchip_Developer_Guide_Ethernet_CN.pdf Rockchip_Developer_Guide_Linux_GMAC_CN.pdf Rockchip_Developer_Guide_Linux_GMAC_Mode_Configuration_CN.pdf Rockchip_Developer_Guide_Linux_GMAC_RGMII_Delayline_CN.pdf Rockchip_Developer_Guide_Linux_GMAC_RGMII_Delayline_EN.pdf
I2C communication	I2C	Rockchip_Developer_Guide_I2C_CN.pdf
GPIO power domain	IO-Domain	Rockchip_Developer_Guide_Linux_IO_DOMAIN_CN.pdf Rockchip_RK356X_Introduction_IO_Power_Domains_Configuration.pdf
IOMMU	IOMMU	Rockchip_Developer_Guide_Linux_IOMMU_CN.pdf
MMC	MMC	Rockchip_Developer_Guide_SD_Boot_CN.pdf Rockchip_Developer_Guide_SDMMC_SDIO_eMMC_CN.pdf
LED	Leds	Rockchip_Introduction_Leds_GPIO_Configuration_for_Linux4.4_CN.pdf
Mobile network 4G/5G	mobile-net	Rockchip_Introduction_3G_Data_Card_USB_File_Conversion_CN.pdf Rockchip_Introduction_3G_Dongle_Configuration_CN&EN.pdf Rockchip_Introduction_4G_Module_Configuration_CN&EN.pdf
Multimedia framework MPP	MPP	Rockchip_Developer_Guide_MPP_CN.pdf
Storage	NVM	Rockchip_Application_Notes_Storage_CN.pdf Rockchip_Developer_FAQ_Storage_CN.pdf Rockchip_Developer_Guide_OTP_CN.pdf Rockchip_Developer_Guide_SATA_CN.pdf Rockchip_Introduction_Partition_CN.pdf Rockchip_RK356X_Developer_Guide_SATA_CN.pdf
PCIe	PCie	Rockchip-Developer-Guide-linux4.4-PCIe.pdf Rockchip_Developer_Guide_PCIe_CN.pdf Rockchip_PCIe_Virtualization_Developer_Guide_CN.pdf Rockchip_RK3399_Developer_Guide_PCIe_CN.pdf
Pin-Ctrl Configuration	PIN-Ctrl	Rockchip-Developer-Guide-Linux-Pin-Ctrl-CN.pdf
PMIC Voltage Meter, DCDC	PMIC	Rockchip_Developer_Guide_Power_Discrete_DCDC_EN.pdf Rockchip_RK805_Developer_Guide_CN.pdf Rockchip_RK806_Developer_Guide_CN.pdf Rockchip_RK808_Developer_Guide_CN.pdf Rockchip_RK809_Developer_Guide_CN.pdf Rockchip_RK816_Developer_Guide_CN.pdf Rockchip_RK817_Developer_Guide_CN.pdf Rockchip_RK818_Developer_Guide_CN.pdf Rockchip_RK818_RK816_Developer_Guide_Fuel_Gauge_CN.pdf Rockchip_RK818_RK816_Introduction_Fuel_Gauge_Log_CN.pdf

Power Supply, Power Consumption	power	Rockchip_Developer_Guide_Power_Analysis_EN.pdf Rockchip_Developer_Guide_Sleep_and_Resume_CN.pdf
PWM	PWM	Rockchip_Developer_Guide_Linux_PWM_CN.pdf Rockchip_Developer_Guide_PWM_IR_CN.pdf
ADC	SARADC	Rockchip_Developer_Guide_Linux_SARADC_CN.pdf
Sensor	Sensors	Rockchip_Developer_Guide_Sensors_CN.pdf
SPI	SPI	Rockchip_Developer_Guide_Linux_SPI_CN.pdf
Temperature control	Thermal	Rockchip_Developer_Guide_Thermal_CN.pdf
TRUST	TRUST	Rockchip_Developer_Guide_Trust_CN.pdf Rockchip_Developer_Guide_Trust_EN.pdf
Serial port	UART	Rockchip-Developer-Guide-RT-Thread-UART.pdf Rockchip_Developer_Guide_UART_CN.pdf Rockchip_Developer_Guide_UART_FAQ_CN.pdf
USB	usb	Rockchip_Developer_Guide_Linux_USB_PHY_CN.pdf Rockchip_Developer_Guide_USB2_Compliance_Test_CN.pdf Rockchip_Developer_Guide_USB_CN.pdf Rockchip_Developer_Guide_USB_FFS_Test_Demo_CN.pdf Rockchip_Developer_Guide_USB_Gadget_UAC_CN.pdf Rockchip_Developer_Guide_USB_SQ_Test_CN.pdf Rockchip_Introduction_USB_SQ_Tool_CN.pdf Rockchip_RK3399_Developer_Guide_USB_DTS_CN.pdf Rockchip_RK356x_Developer_Guide_USB_CN.pdf Rockchip_Trouble_Shooting_Linux4.19_USB_Gadget_UVC_CN.pdf
Watchdog	watchdog	Rockchip_Developer_Guide_Linux_WDT_CN.pdf

Table 6.2.6.1 Summary of Kernel Module Development Documents

The table only lists some of the documents, not all. All should be based on the documents in the **<SDK>/RKDocs/common/** directory!

6.3 Android System Development

This subsection covers the relevant content of Android system development.

6.3.1 Android System Architecture

The architecture diagram of the Android system is as follows:

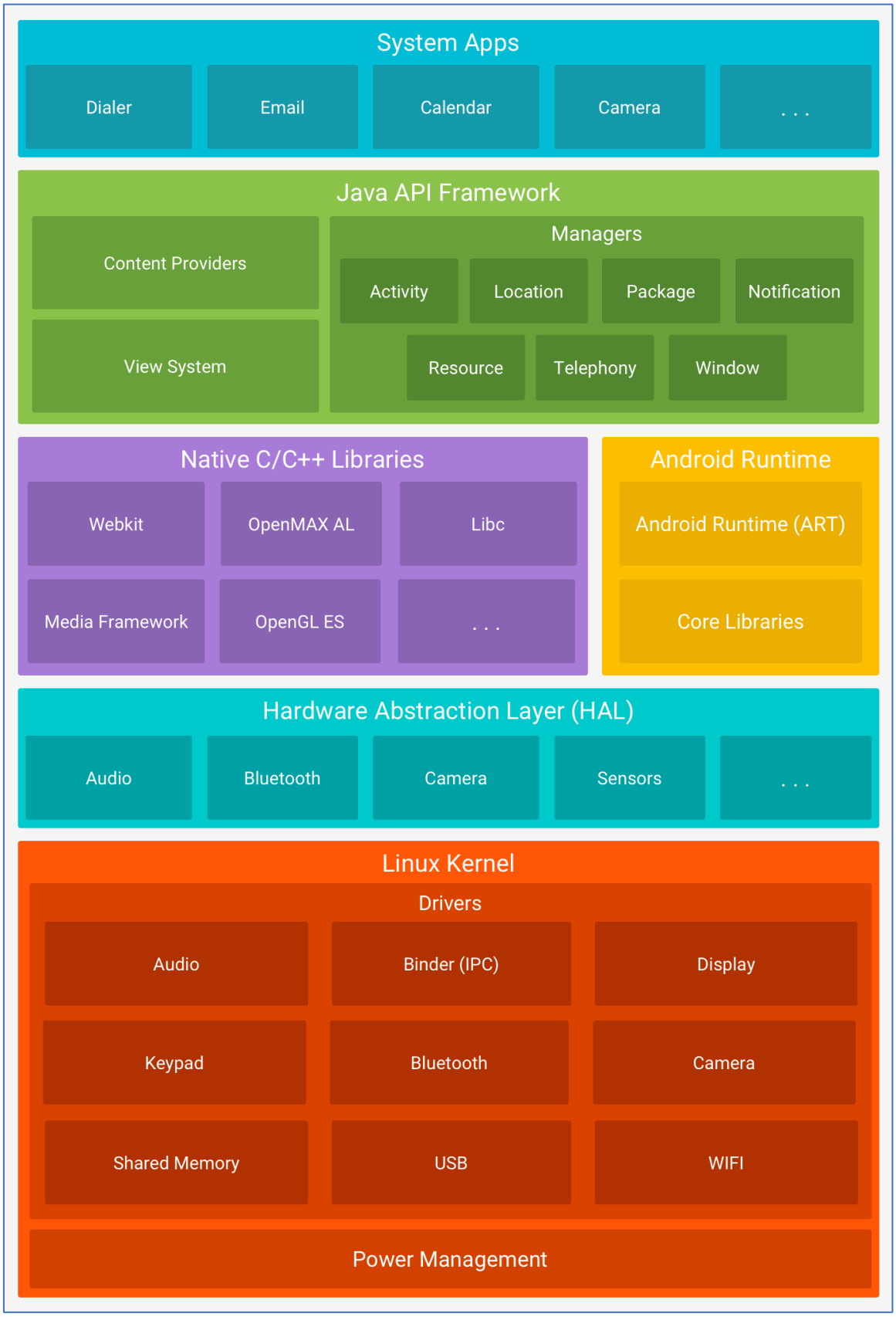


Figure 6.3.1.1 Android System Architecture

The Android system architecture is divided into five layers, from top to bottom: the Application Layer (System Apps), the Application Framework Layer (Java API Framework), the System Runtime Layer (Android Runtime & Native C/C++ Libraries), the Hardware Abstraction Layer (HAL), and the Linux Kernel Layer (Linux Kernel).

① Application Layer

The Application Layer is the topmost layer of the entire Android system. This layer includes core system applications, input methods, and third-party applications, etc. The built-in applications of the Android system and non-built-in system applications all belong to the Application Layer and are responsible for interacting directly with users. They are usually developed using Java.

② Application Framework Layer (Java API Framework)

The application framework layer provides various APIs required for developing Android applications. Application developers call the APIs provided by the framework layer to develop applications, including system applications. The framework layer is written in Java code and is therefore also called the Java Framework.

In addition, the framework layer also provides a series of important system services for applications. Application developers are allowed to use these services in their applications:

Service name	Explanation
Activity Manager	Manage the life cycle of each application and provide a universal navigation back function
Location Manager	Offer geographical location and positioning service
Package Manager	Manage all applications installed on the Android system, including installation, uninstallation, and update
Notification Manager	Enable all applications to display notification information in the status bar
Resource Manager	Manage a series of resources (non-code resources) other than code, such as local strings, images, layout files, color files, etc.
Telephony Manager	Manage all mobile device functions
Window Manager	Manage all programs with open windows
Content Provider	Enable data sharing between different applications
View System	Build basic components of the application, such as lists, grids, text boxes, buttons, etc.

Table 6.3.1.1 System Service Overview

③. System Runtime Library Layer (Native)

This layer is divided into two parts: C/C++ program libraries and Android runtime libraries.

C/C++ Program Libraries: These are provided for use by various components of the Android system. These functions are provided to developers through the application framework layer.

Android Runtime Library (Android Runtime): The runtime libraries are divided into core libraries and ART (after Android 5.0, the Dalvik virtual machine was replaced by ART). The core libraries integrate most of the functions of the Java language core library, allowing developers to write Android applications using the Java language. Under the ART mechanism, the system will perform a pre-compilation when installing applications, converting bytecode into machine code and storing it

locally. This way, when the application starts each time, there is no need to execute the compilation again, significantly improving efficiency!

④. Hardware Abstraction Layer (HAL/HIDL)

"HAL" is the abbreviation of "hardware abstraction layer", and it is located between the Framework and the Linux Kernel. The code of the HAL layer is generally composed of C/C++. The purpose of the Hardware Abstraction Layer is to abstract the underlying hardware and provide unified and standardized API interfaces to the upper layer; therefore, the HAL layer is a software layer that shields the operations of the underlying hardware for upper-layer applications. Upper-layer applications do not need to concern how the underlying hardware works; they only need to call the unified interfaces provided by the HAL layer! Generally, the main task of Android's low-level development is to modify the HAL layer to adapt to specific hardware.

On the other hand, the significance of the HAL layer is to protect intellectual property rights. Hardware manufacturers, considering their interests, do not want to disclose the implementation details and parameters related to the hardware devices. Therefore, hardware manufacturers or individuals can place the code they do not want to open source (such as some core algorithms) in the HAL layer in the form of a dynamic link library, and the HAL layer is located in the user space (Linux user space), not belonging to the Linux kernel, and does not need to follow the GPL open-source protocol.

⑤. Linux Kernel

The foundation of the Android system is the Linux kernel. The core system services of Android are based on the Linux kernel and add some Android-specific drivers on top of it. The security, memory management, process management, network protocol stack, and driver model of the system all rely on the Linux kernel.

6.3.2 Android product configuration

Executing the "lunch" command in the <SDK> directory (without any parameters) will list all the Android configurations. Before compiling the Android source code, users need to select one of these products, such as rk3568_r-userdebug. How can we add our own product?

This section introduces Android product configuration to users, including the Android product configuration architecture, Android product configuration related files, and how to configure a new product. Before formally introducing product configuration, let's first understand the following concepts:

- Target product: Specifically refers to the Android device that the end user buys, such as Xiaomi phones: Xiaomi 13, Xiaomi 13 Pro, Xiaomi 12, Xiaomi 12 Pro, Oppo phones: Find X6, Find X5, Find N2, etc.
- Product series: After a product is developed by an Android product developer, they often continue to develop new products based on it. These new products often upgrade the hardware or software of the old product. These products (new and old products) form a product series, such as Oppo's Find X series, Find N series, etc.
- Target device: The target device can be understood as the mobile phone motherboard, which is the hardware motherboard of the Android product. It refers to the collection of hardware configuration information of the Android device. Each Android product has its own hardware configuration, and one hardware configuration may be used by different products. The same phone often has high-end and low-end versions, such as 8GB+64GB, 12GB+128GB, 12GB+256GB, etc.

In the Android compilation system, each compilation item (also called a product compilation item) will correspond to a target product, and each target product will correspond to a target device. A product series can contain multiple different products, and a target device can be used by multiple products.

1. Android product configuration architecture

Product configuration is mainly carried out in the <SDK>/device directory. The structure of the device directory is generally as shown in the following figure:

```
device-|
      |-<company_name>-|
                  |-<board_name>-|
                              |-AndroidBoard.mk
                              |-BoardConfig.mk
                  |-products-|
                          |-AndroidProducts.mk
                          |-<first_product_name>.mk
                          |-<second_product_name>.mk
```

Figure 6.3.2.1 Device directory structure (1)

①Generally, a folder named after the manufacturer's name will be created under the device directory, and the name can be defined by yourself, such as rockchip (Rockchip Microelectronics), alientek (ALIENTEK) etc.;

②Then, in the manufacturer's name folder, create a "products" directory. This directory is used to store all the product configuration files (product Makefile configuration files) of this manufacturer, the AndroidProducts.mk file (used to define the list of all products of this manufacturer), and other required files.

③Under the manufacturer name folder, one or more directories named after the motherboard (or device) will be created. These directories are used to store files such as BoardConfig.mk (the motherboard Makefile configuration file) and AndroidBoard.mk.

In addition to the above directory structure, the following structure is also commonly used, as shown in the following figure:

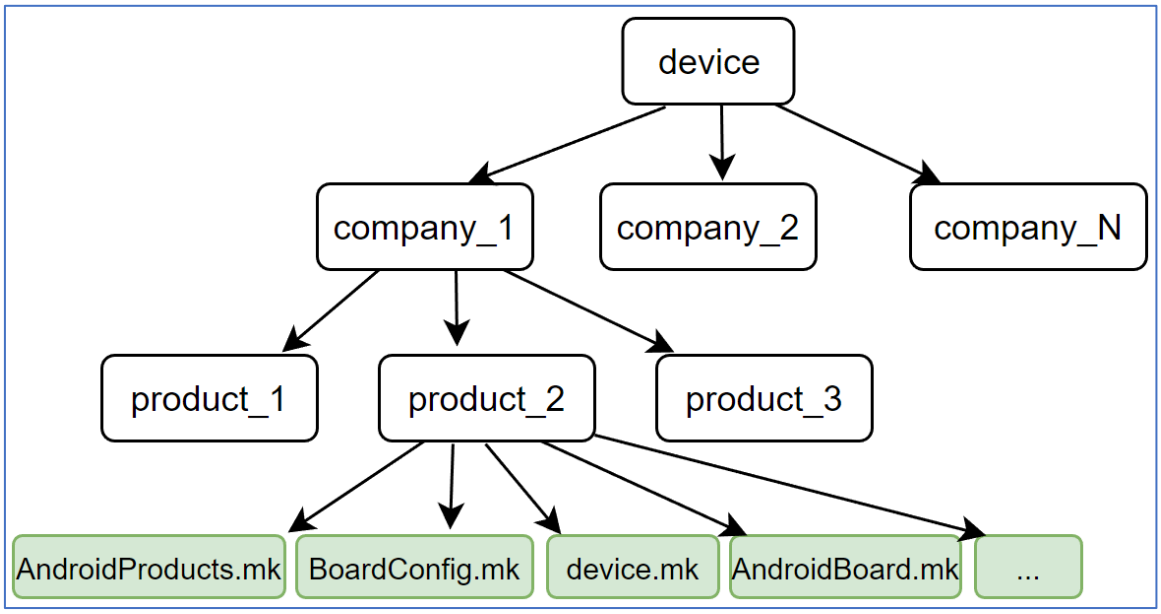


Figure 6.3.2.2 Device directory structure (2)

①. In the device directory, folders named after the manufacturer's name will be created. The names can be defined by yourself, such as rockchip (Rockchip Microelectronics), alientek (ALIENTEK) etc.;

②. Then, in the manufacturer's name folder, a folder named after the product will be created. The name can also be defined by yourself, such as device/alientek/atk_dlrk3568. If there are multiple products, multiple folders named after the products will be created, indicating that the manufacturer has multiple products;

③. Some mk configuration files such as AndroidProducts.mk, BoardConfig.mk, device.mk, AndroidBoard.mk, <product_name>.mk, etc. will be added in the product directory.

The above has introduced two types of device directory structures to you. However, it is not necessary for users to add or configure a new product in accordance with this structure.

The contents of the "device" directory in the RK3568 Android SDK are as follows:

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ ls -l device/
总用量 24
drwxrwxr-x  3 tgg tgg 4096 4月  22 16:20 common
drwxrwxr-x 18 tgg tgg 4096 4月  22 16:21 generic
drwxrwxr-x  3 tgg tgg 4096 4月  22 15:50 google
drwxrwxr-x  9 tgg tgg 4096 4月  22 15:50 google_car
drwxrwxr-x  9 tgg tgg 4096 4月  29 21:00 rockchip
drwxrwxr-x  9 tgg tgg 4096 4月  22 15:50 sample
tgg@tgg-virtual-machine:~/rk3568_android_sdk$
```

Figure 6.3.2.3 Device directory

Our main focus is on the rockchip directory, which contains the product configuration information of Rockchip. Let's take a look at the structure of this directory, as follows:

```
rockchip
 ├── common
 ├── rk3288
 ├── rk3326
 ├── rk3328
 ├── rk3399
```

```
├── rk356x
│   ├── rk3566_32bit
│   │   ├── AndroidBoard.mk
│   │   ├── Android.mk
│   │   ├── BoardConfig.mk
│   │   └── rk3566_32bit.mk
│   ├── rk3566_eink
│   │   ├── AndroidBoard.mk
│   │   ├── Android.mk
│   │   ├── BoardConfig.mk
│   │   └── rk3566_eink.mk
│   ├── rk3566_einkw6
│   │   ├── AndroidBoard.mk
│   │   ├── Android.mk
│   │   ├── BoardConfig.mk
│   │   └── rk3566_einkw6.mk
│   ├── rk3566_r
│   │   ├── AndroidBoard.mk
│   │   ├── Android.mk
│   │   ├── BoardConfig.mk
│   │   └── rk3566_r.mk
│   ├── rk3566_rgo
│   │   ├── AndroidBoard.mk
│   │   ├── Android.mk
│   │   ├── BoardConfig.mk
│   │   └── rk3566_rgo.mk
│   └── rk3568_r
│       ├── AndroidBoard.mk
│       ├── Android.mk
│       ├── BoardConfig.mk
│       └── rk3568_r.mk
└── rk356x_box
```

Directory name	Explanation
common	Including common files for the RK platform, such as common Makefile files, shell scripts, Python scripts, XML files, rc files, etc.
rk3288	Including product configuration information for the Resin Micro RK3288 platform
rk3326	Including product configuration information for the Resin Micro RK3326 platform
rk3328	Including product configuration information for the Resin Micro RK3328 platform
rk3399	Including product configuration information for the Resin Micro RK3399 platform
rk356x	Including product configuration information for the Resin Micro RK3566 and RK3568 platforms

rk356x_box	Including Android tv box product configuration information for the Resin Micro RK3568 and RK3566 platforms (Android TV boxes)

Table 6.3.2.1 Rockchip Directory Description

The product configuration information for the RK3568 platform is stored in the rk356x directory:

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/device/rockchip/rk356x$ ls
Android.mk          init.recovery.rk30board.rc  public.libraries.txt  rk3566_rgo            sepolicy_ebook_system
AndroidProducts.mk  init.rk356x.rc              rk3566_32bit          rk3568_r              sepolicy_vendor
bluetooth           ota                         rk3566_eink           rk356x.prop           wake_lock_filter.xml
BoardConfig.mk      overlay                     rk3566_einkw6         sepolicy_ebook        wifi_bt.mk
device.mk           package_performance.xml     rk3566_r              sepolicy_ebook_public
tgg@tgg-virtual-machine:~/rk3568_android_sdk/device/rockchip/rk356x$
```

Figure 6.3.2.4 rk356x directory

This directory contains 6 product configurations: rk3566_32bit, rk3566_eink, rk3566_einkw6, rk3566_r, rk3566_rgo, and rk3568_r. Each directory has a product Makefile configuration file, namely rk3566_32bit.mk, rk3566_eink.mk, rk3566_einkw6.mk, rk3566_r.mk, rk3566_rgo.mk, and rk3568_r.mk. In the rk356x directory, there is also an AndroidProducts.mk file, which is used to define the list of all products under the rk356x directory.

2. Configuration related to Android products.mk file

I. AndroidProducts.mk file (required)

The AndroidProducts.mk file is used to define the list of all products of the current manufacturer. The content of this file is very simple and usually only defines two variables: **PRODUCT_MAKEFILES** and **COMMON_LUNCH_CHOICES**. For example:

```
PRODUCT_MAKEFILES := \
        $(LOCAL_DIR)/rk3566_rgo/rk3566_rgo.mk \
        $(LOCAL_DIR)/rk3566_32bit/rk3566_32bit.mk \
        $(LOCAL_DIR)/rk3566_r/rk3566_r.mk \
        $(LOCAL_DIR)/rk3568_r/rk3568_r.mk \
        $(LOCAL_DIR)/rk3566_eink/rk3566_eink.mk \
        $(LOCAL_DIR)/rk3566_einkw6/rk3566_einkw6.mk

COMMON_LUNCH_CHOICES := \
    rk3566_32bit-userdebug \
    rk3566_32bit-user \
    rk3566_rgo-userdebug \
    rk3566_rgo-user \
    rk3566_r-userdebug \
    rk3566_r-user \
    rk3568_r-userdebug \
    rk3568_r-user \
    rk3566_eink-userdebug \
    rk3566_eink-user \
    rk3566_einkw6-userdebug \
    rk3566_einkw6-user
```

The PRODUCT_MAKEFILES variable is used to record the Makefile configuration files (.mk files) of the products. Each product has a corresponding configuration file, which is also known as the product configuration file. This product configuration file is actually a Makefile file and is usually

named <product_name>.mk, where <product_name> represents the product name. The path of the product configuration file needs to be added to the PRODUCT_MAKEFILES variable for saving. In the above example, $(LOCAL_DIR) represents the current directory, which is the directory where AndroidProducts.mk file is located.

The COMMON_LUNCH_CHOICES variable is used to add the product (or product compilation item) to the lunch selection list (the list displayed when executing the lunch command). The name of the product compilation item consists of two parts: **<product name> - <version type>**, and <product name> and <version type> are separated by the "-" symbol. There are three types of version types: **user (user version), userdebug (user debugging version),** and **eng (engineering version)**. The differences between these three are as follows:

Version type	Explanation
user	Intended as the final release version ● Install the modules with the "user" tag ● In addition to the modules with the tag, corresponding files will also be installed according to the product definition file ● ro.secure=1 ● ro.debuggable=0 ● adb is default disabled
userdebug	Similar to the user, except for the following points: ● Modules with debug tags will also be installed ● ro.debuggable = 1 ● adb is enabled by default
eng	● Install the modules with the "eng" or "debug" tags ● In addition to the modules with the tags, the corresponding modules will be installed according to the product definition file ● ro.secure=0 ● ro.debuggable=1 ● ro.kernel.android.checkjni=1 ● adb is enabled by default

Table 6.3.2.2 The differences among user, userdebug and eng

Notes:

①. The AndroidProducts.mk file is usually stored in the <SDK>/device/Company Name/Product Name/ directory or <SDK>/device/Company Name/Products/ directory. Of course, it is not necessary to place the AndroidProducts.mk file in this directory. It can also be placed in other directories according to one's own planning and requirements, and there is no problem with that. Because when executing the lunch command, it will search for the AndroidProducts.mk file in the device directory and find all files named AndroidProducts.mk and load them, so it does not search for the AndroidProducts.mk file in certain specific directories, but directly searches the entire device directory.

②. The naming convention for product configuration files is generally <product_name>.mk. It is a Makefile file, where product_name represents the product name. The Android compilation system will search for the product configuration file based on the product name (product_name). It will look for whether <product_name>.mk exists in the PRODUCT_MAKEFILES list. Of course, the name of

the product configuration file can also be different from <product_name>.mk. In this case, it needs to be specified through the PRODUCT_MAKEFILES variable as follows:

```
<product_name>:<product_makefile>
```

<Product Name> + <Colon> + <Product Makefile File Path>, for example:

```
PRODUCT_MAKEFILES := \
    aosp_cf_arm64_phone:$(LOCAL_DIR)/vsoc_arm64/phone/device.mk \
    aosp_cf_x86_64_phone:$(LOCAL_DIR)/vsoc_x86_64/phone/device.mk \
    aosp_cf_x86_auto:$(LOCAL_DIR)/vsoc_x86/auto/device.mk \
    aosp_cf_x86_pasan:$(LOCAL_DIR)/vsoc_x86/pasan/device.mk \
    aosp_cf_x86_phone:$(LOCAL_DIR)/vsoc_x86/phone/device.mk \
    aosp_cf_x86_go_phone:$(LOCAL_DIR)/vsoc_x86/go_phone/device.mk \
    aosp_cf_x86_gsi:$(LOCAL_DIR)/vsoc_x86/gsi/aosp_cf_x86_gsi.mk \
    aosp_cf_x86_tv:$(LOCAL_DIR)/vsoc_x86/tv/device.mk

COMMON_LUNCH_CHOICES := \
    aosp_cf_arm64_phone-userdebug \
    aosp_cf_x86_64_phone-userdebug \
    aosp_cf_x86_auto-userdebug \
    aosp_cf_x86_phone-userdebug \
    aosp_cf_x86_tv-userdebug
```

2. Product Configuration File/Product Makefile File: <product_name>.mk (Required)

The product Makefile file <product_name>.mk, in this file, some information about the product is defined: product name, device name, product manufacturer, product model, etc. For example:

```
PRODUCT_NAME := rk3568_r
PRODUCT_DEVICE := rk3568_r
PRODUCT_BRAND := rockchip
PRODUCT_MODEL := rk3568_r
PRODUCT_MANUFACTURER := rockchip
```

Descriptions are as follows:

Variable name	Effect	Explanation
PRODUCT_NAME	The product name will be displayed in "**Settings→ About Phone**"	The product name should be consistent with the product name in the product compilation item, which is the product compilation item defined by the COMMON_LUNCH_CHOICES variable: $(PRODUCT_NAME)-<version type>
PRODUCT_DEVICE	The corresponding device name of the product, which is the name of the	The compilation system will look for the motherboard configuration file BoardConfig.mk based on this name. This name should be consistent with the directory name where the BoardConfig.mk file is located.

	motherboard or the name of the board;	The output directory of the product is also created based on this name as out/target/product/$(PRODUCT_DEVICE)
PRODUCT_BRAND	For customizing the software, the brand (if any) is targeted, such as the operator.	
PRODUCT_MODEL	The final user-visible name of the end product, or called the model name.	
PRODUCT_MANUFACTURER	The name of the product manufacturer.	It will be assigned to the ro.product.manufacturer attribute

Table 6.3.2.3 Explanation of Common Variables in Product Makefile Files

Once again, the compilation system will use PRODUCT_DEVICE to search for the motherboard configuration file BoardConfig.mk. This name must be consistent with the directory name of the BoardConfig.mk file, and ensure that this directory name is unique within the device directory structure; because the compilation system will search for the directory named $(PRODUCT_DEVICE) in the device directory (with a maximum path depth of 4 levels) and load the BoardConfig.mk file in this directory.

3. BoardConfig.mk File (Required)

The motherboard configuration file BoardConfig.mk, which is also a Makefile file, contains specific motherboard configuration information and is used to define and define the underlying characteristics and variables related to the hardware. For example:

```
TARGET_ARCH := arm64
TARGET_ARCH_VARIANT := armv8-a
TARGET_CPU_ABI := arm64-v8a
TARGET_CPU_ABI2 :=
TARGET_CPU_VARIANT := cortex-a55
TARGET_CPU_SMP := true

TARGET_2ND_ARCH := arm
TARGET_2ND_ARCH_VARIANT := armv8-2a
TARGET_2ND_CPU_ABI := armeabi-v7a
TARGET_2ND_CPU_ABI2 := armeabi
TARGET_2ND_CPU_VARIANT := cortex-a55
```

```
TARGET_PREBUILT_KERNEL := kernel/arch/arm64/boot/Image
BOARD_PREBUILT_DTBIMAGE_DIR := kernel/arch/arm64/boot/dts/rockchip

PRODUCT_UBOOT_CONFIG ?= rk3568
PRODUCT_KERNEL_ARCH ?= arm64
PRODUCT_KERNEL_DTS ?= rk3568-evb1-ddr4-v10
PRODUCT_KERNEL_CONFIG ?= rockchip_defconfig
BOARD_NFC_SUPPORT := false
BOARD_HAS_GPS := false

BOARD_GRAVITY_SENSOR_SUPPORT := true
BOARD_COMPASS_SENSOR_SUPPORT := false
BOARD_GYROSCOPE_SENSOR_SUPPORT := true
BOARD_PROXIMITY_SENSOR_SUPPORT := true
BOARD_LIGHT_SENSOR_SUPPORT := true
BOARD_PRESSURE_SENSOR_SUPPORT := false
BOARD_TEMPERATURE_SENSOR_SUPPORT := false
BOARD_USB_HOST_SUPPORT := true
```

The maximum depth of the path where the BoardConfig.mk file is located should not exceed 4 levels; otherwise, the compilation system will not be able to find the file. Moreover, the directory name where the BoardConfig.mk file is located must be equal to $(PRODUCT_DEVICE).

4. device.mk file (optional)

It is usually inherited by various product Makefile files <product_name>.mk (product configuration files) as a general Makefile file. For example:

```
rk356x/rk3566_einkw6/rk3566_einkw6.mk:30:$(call inherit-product, device/rockchip/rk356x/device.mk)
rk356x/rk3566_einkw6/rk3566_einkw6.mk:31:$(call inherit-product, device/rockchip/common/device.mk)
rk356x/rk3566_rgo/rk3566_rgo.mk:25:$(call inherit-product, device/rockchip/rk356x/device.mk)
rk356x/rk3566_rgo/rk3566_rgo.mk:26:$(call inherit-product, device/rockchip/common/device.mk)
rk356x/rk3568_r/rk3568_r.mk:25:$(call inherit-product, device/rockchip/rk356x/device.mk)
rk356x/rk3568_r/rk3568_r.mk:26:$(call inherit-product, device/rockchip/common/device.mk)
rk356x/rk3566_32bit/rk3566_32bit.mk:25:$(call inherit-product, device/rockchip/rk356x/device.mk)
rk356x/rk3566_32bit/rk3566_32bit.mk:26:$(call inherit-product, device/rockchip/common/device.mk)
```

Figure 6.3-1 device.mk file

5. AndroidBoard.mk file (optional)

This file will be included in the core code of the compilation system:

```
build/make/target/board/Android.mk:7:-include $(TARGET_DEVICE_DIR)/AndroidBoard.mk
```

```
1 #
2 # Set up product-global definitions and include product-specific rules.
3 #
4
5 LOCAL_PATH := $(call my-dir)
6
7 -include $(TARGET_DEVICE_DIR)/AndroidBoard.mk
8
```

Figure 6.3-2 Contains AndroidBoard.mk

The AndroidBoard.mk file will be included by **<SDK>/build/make/target/board/Android.mk.** Unlike the common "**include**", here "-include" is used; using "include" to include a Makefile file, if the

included file does not exist, then make will report an error and exit; while using "-include", if the included file does not exist, make will not exit; it will continue to execute (the "-" means to tell make to ignore this error and continue execution).

Therefore, the AndroidBoard.mk file is not necessary and can be present or absent; if this file is provided, it must be placed in the directory where the board configuration file BoardConfig.mk is located, that is, to place AndroidBoard.mk and BoardConfig.mk in the same directory.

For example (device/rockchip/rk356x/rk3568_r/AndroidBoard.mk):

```
# generate fstab file for device
-include device/rockchip/common/build/rockchip/RebuildFstab.mk


# generate dtbo image for device
-include device/rockchip/common/build/rockchip/RebuildDtboImg.mk
# generate parameter.txt for device
-include device/rockchip/common/build/rockchip/RebuildParameter.mk
```

3.Configure a new product

This section will explain how to add/create a new product.

① Copy the reference template

First, enter the <SDK>/device/rockchip/rk356x directory, take <SDK>/device/rockchip/rk356x/rk3568_r as the template, and copy it to <SDK>/device/rockchip/rk356x/my_rk3568 (the directory name can be defined by yourself), as shown below:

```
cp -r rk3568_r my_rk3568
```

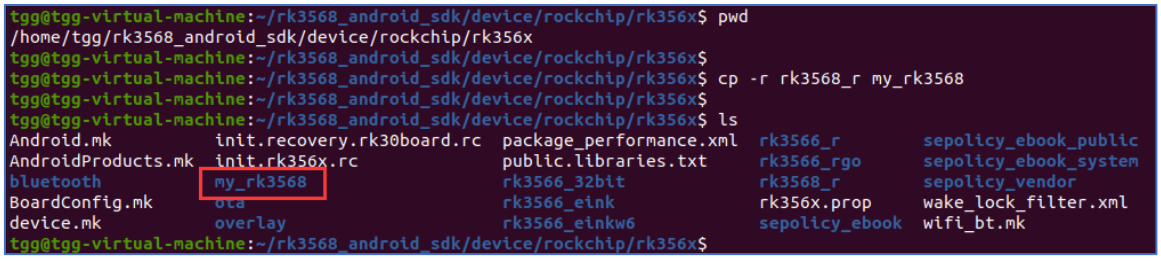


Figure 6.3.2.7 my_rk3568

We generally do not start from scratch to configure a new product because the workload would be too large; instead, we use the existing RK3568 platform product rk3568_r as a template to implement a new product.

② , modify the rk356x/AndroidProducts.mk file

As we have already explained to you, the AndroidProducts.mk file defines a list of products. We need to add our product my_rk3568 to this product list, as shown below:

```
PRODUCT_MAKEFILES := \
        $(LOCAL_DIR)/my_rk3568/my_rk3568.mk \
        $(LOCAL_DIR)/rk3566_rgo/rk3566_rgo.mk \
        $(LOCAL_DIR)/rk3566_32bit/rk3566_32bit.mk \
        $(LOCAL_DIR)/rk3566_r/rk3566_r.mk \
        $(LOCAL_DIR)/rk3568_r/rk3568_r.mk \
        $(LOCAL_DIR)/rk3566_eink/rk3566_eink.mk \
```

```
        $(LOCAL_DIR)/rk3566_einkw6/rk3566_einkw6.mk

COMMON_LUNCH_CHOICES := \
    my_rk3568-userdebug \
    my_rk3568-user \
    rk3566_32bit-userdebug \
    rk3566_32bit-user \
    rk3566_rgo-userdebug \
    rk3566_rgo-user \
    rk3566_r-userdebug \
    rk3566_r-user \
    rk3568_r-userdebug \
    rk3568_r-user \
    rk3566_eink-userdebug \
    rk3566_eink-user \
    rk3566_einkw6-userdebug \
    rk3566_einkw6-user
```

The bolded part is the content we have added. The meanings represented by these two variables have been introduced in the previous text, so we won't elaborate on them here!

③ . Modify my_rk3568.mk

First, enter the directory rk356x/my_rk3568, rename rk3568_r.mk to my_rk3568.mk as follows:

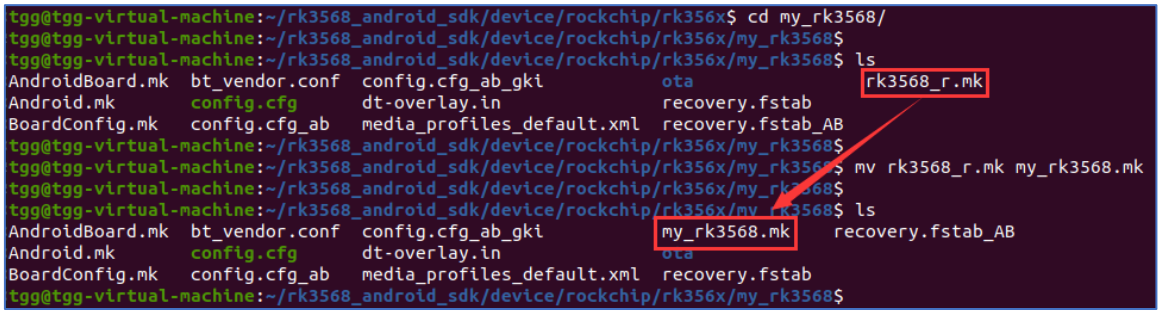


Figure 6.3.2.8 Rename rk3568_r.mk to my_rk3568.mk

my_rk3568.mk is the Makefile configuration file for the product, and modify it as follows:

```
PRODUCT_NAME := my_rk3568
PRODUCT_DEVICE := my_rk3568
PRODUCT_BRAND := rockchip
PRODUCT_MODEL := my_rk3568
PRODUCT_MANUFACTURER := rockchip
```

```
17 # First lunching is R, api_level is 30
18 PRODUCT_SHIPPING_API_LEVEL := 30
19 PRODUCT_DTBO_TEMPLATE := $(LOCAL_PATH)/dt-overlay.in
20 PRODUCT_SDMMC_DEVICE := fe2b0000.dwmmc
21
22 include device/rockchip/common/build/rockchip/DynamicPartitions.mk
23 include device/rockchip/rk356x/rk3568_r/BoardConfig.mk
24 include device/rockchip/common/BoardConfig.mk
25 $(call inherit-product, device/rockchip/rk356x/device.mk)
26 $(call inherit-product, device/rockchip/common/device.mk)
27 $(call inherit-product, frameworks/native/build/tablet-10in-xhdpi-2048-dalvik-heap.mk)
28
29 DEVICE_PACKAGE_OVERLAYS += $(LOCAL_PATH)/../overlay
30
31 PRODUCT_CHARACTERISTICS := tablet
32
33 PRODUCT_NAME := my_rk3568
34 PRODUCT_DEVICE := my_rk3568
35 PRODUCT_BRAND := rockchip
36 PRODUCT_MODEL := my_rk3568
37 PRODUCT_MANUFACTURER := rockchip
38 PRODUCT_AAPT_PREF_CONFIG := mdpi
39 #
40 ## add Rockchip properties
41 #
42 PRODUCT_PROPERTY_OVERRIDES += ro.sf.lcd_density=320
43 PRODUCT_PROPERTY_OVERRIDES += ro.wifi.sleep.power.down=true
44 PRODUCT_PROPERTY_OVERRIDES += persist.wifi.sleep.delay.ms=0
45 PRODUCT_PROPERTY_OVERRIDES += persist.bt.power.down=true
```

Figure 6.3.2.9 The content of my_rk3568.mk file

These variables have been discussed before. The key points to note are PRODUCT_NAME and PRODUCT_DEVICE. PRODUCT_NAME is the product name, and it needs to be consistent with the product compilation item in the product configuration file, which is the product compilation item defined by the COMMON_LUNCH_CHOICES variable:

$(PRODUCT_NAME)-<version type>

In addition, the name of the product Makefile configuration file must also be $(PRODUCT_NAME).mk, unless specified in the PRODUCT_MAKEFILES variable.

PRODUCT_DEVICE represents the device name, which is also the motherboard name. The compilation system will use PRODUCT_DEVICE to search for the motherboard configuration file BoardConfig.mk. PRODUCT_DEVICE must be consistent with the directory name of the BoardConfig.mk file.

④ , Modifying path information

We created a new product based on the template rk3568_r. We need to modify the paths related to the rk3568_r directory to my_rk3568 as follows:

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/device/rockchip/rk356x/my_rk3568$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/device/rockchip/rk356x/my_rk3568$ grep -nR "device/rockchip/rk356x/rk3568_r"
my_rk3568.mk:23:include device/rockchip/rk356x/rk3568_r/BoardConfig.mk
BoardConfig.mk:25:    TARGET_RECOVERY_FSTAB := device/rockchip/rk356x/rk3568_r/recovery.fstab_AB
tgg@tgg-virtual-machine:~/rk3568_android_sdk/device/rockchip/rk356x/my_rk3568$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/device/rockchip/rk356x/my_rk3568$
```

Figure 6.3.2.10 Files related to the "rk3568_r" directory

Only the my_rk3568.mk and BoardConfig.mk files need to be modified.

The my_rk3568.mk file is modified as follows:

```
# First lunching is R, api_level is 30
PRODUCT_SHIPPING_API_LEVEL := 30
PRODUCT_DTBO_TEMPLATE := $(LOCAL_PATH)/dt-overlay.in
PRODUCT_SDMMC_DEVICE := fe2b0000.dwmmc

include device/rockchip/common/build/rockchip/DynamicPartitions.mk
include device/rockchip/rk356x/my_rk3568/BoardConfig.mk
include device/rockchip/common/BoardConfig.mk
$(call inherit-product, device/rockchip/rk356x/device.mk)
$(call inherit-product, device/rockchip/common/device.mk)
$(call inherit-product, frameworks/native/build/tablet-10in-xhdpi-2048-dalvik-heap.mk)

DEVICE_PACKAGE_OVERLAYS += $(LOCAL_PATH)/../overlay

```

Figure 6.3.2.11 Modify the my_rk3568.mk file

The BoardConfig.mk file has been modified as follows:

```
include device/rockchip/rk356x/BoardConfig.mk
BUILD_WITH_GO_OPT := false

# AB image definition
BOARD_USES_AB_IMAGE := false
BOARD_ROCKCHIP_VIRTUAL_AB_ENABLE := false

ifeq ($(strip $(BOARD_USES_AB_IMAGE)), true)
    include device/rockchip/common/BoardConfig_AB.mk
    TARGET_RECOVERY_FSTAB := device/rockchip/rk356x/my_rk3568/recovery.fstab_AB
endif

PRODUCT_UBOOT_CONFIG := rk3568
PRODUCT_KERNEL_DTS := rk3568-atk-evb1-ddr4-v10
BOARD_GSENSOR_MXC6655XA_SUPPORT := true
BOARD_CAMERA_SUPPORT_EXT := true
BOARD_HS_ETHERNET := true
```

Figure 6.3.2.12 Modify the BoardConfig.mk file

At this point, we have added a new product called my_rk3568.

⑤ . Select the new product my_rk3568 for lunch

Return to the top-level directory of <SDK>, execute the lunch command, and at this time you can see the product we added, my_rk3568, as shown below:

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ pwd
/home/tgg/rk3568_android_sdk
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ lunch

You're building on Linux

Lunch menu... pick a combo:
     1. PX30_Android11-user
     2. PX30_Android11-userdebug
     3. aosp_arm-eng
     4. aosp_arm64-eng
     5. aosp_blueline_car-userdebug
     6. aosp_bonito_car-userdebug
     7. aosp_car_arm-userdebug
     8. aosp_car_arm64-userdebug
     9. aosp_car_x86-userdebug
     10. aosp_car_x86_64-userdebug
     11. aosp_coral_car-userdebug
     12. aosp_crosshatch_car-userdebug
     13. aosp_flame_car-userdebug
     14. aosp_x86-eng
     15. aosp_x86_64-eng
     16. car_x86_64-userdebug
     17. my_rk3568-user
     18. my_rk3568-userdebug
     19. qemu_trusty_arm64-userdebug
     20. rk3288_Android10-user
     21. rk3288_Android10-userdebug
     22. rk3288_Android11-user
     23. rk3288_Android11-userdebug
     24. rk3326_pie-user
     25. rk3326_pie-userdebug
     26. rk3326_q-user
     27. rk3326_q-userdebug
     28. rk3326_r-user
     29. rk3326_r-userdebug
     30. rk3326_rgo-user
     31. rk3326_rgo-userdebug
     32. rk3328_atv-user
     33. rk3328_atv-userdebug
```

Product configuration items corresponding to the new product

Figure 6.3.2.13 Lunch Select Product (1)

Enter the corresponding number code and press Enter to select the product:

```
Which would you like? [aosp_arm-eng] 18

============================================
PLATFORM_VERSION_CODENAME=REL
PLATFORM_VERSION=11
TARGET_PRODUCT=my_rk3568
TARGET_BUILD_VARIANT=userdebug
TARGET_BUILD_TYPE=release
TARGET_ARCH=arm64
TARGET_ARCH_VARIANT=armv8-a
TARGET_CPU_VARIANT=cortex-a55
TARGET_2ND_ARCH=arm
TARGET_2ND_ARCH_VARIANT=armv8-2a
TARGET_2ND_CPU_VARIANT=cortex-a55
HOST_ARCH=x86_64
HOST_2ND_ARCH=x86
HOST_OS=linux
HOST_OS_EXTRA=Linux-5.15.0-53-generic-x86_64-Ubuntu-20.04.2-LTS
HOST_CROSS_OS=windows
HOST_CROSS_ARCH=x86
HOST_CROSS_2ND_ARCH=x86_64
HOST_BUILD_TYPE=release
BUILD_ID=RD2A.211001.002
OUT_DIR=out
============================================
tgg@tgg-virtual-machine:~/rk3568_android_sdk$
```

Figure 6.3.2.14 lunch selection of products (2)

4. Various variables involved in the configuration of Android products

There are many variables involved in the configuration of Android products. Some are common variables, such as PRODUCT_NAME, PRODUCT_DEVICE, PRODUCT_BRAND, PRODUCT_MODEL, and PRODUCT_MANUFACTURER; some are custom variables, and some are

functional variables (indicating that the variable has special functions), such as PRODUCT_COPY_FILES, PRODUCT_PROPERTY_OVERRIDES, and PRODUCT_PACKAGES, as shown below:

Variable name	Function
PRODUCT_COPY_FILES	Used to complete the copy operation, copying files from the source path to the target path
PRODUCT_PROPERTY_OVERRIDES	Used to set system properties (overwriting)
PRODUCT_PACKAGES	List of APKs and modules to be installed

Table 6.3.2.4 Description of Common Function Variables

PRODUCT_COPY_FILES example:

```
PRODUCT_COPY_FILES += vendor/rockchip/common/phone/etc/spn-conf.xml:system/etc/spn-conf.xml
```

Indicates that the file "vendor/rockchip/common/phone/etc/spn-conf.xml" is copied to the product's output directory (out/target/product/$(PRODUCT_DEVICE)) and saved as "system/etc/spn-conf.xml".

Example of PRODUCT_PROPERTY_OVERRIDES:

```
PRODUCT_PROPERTY_OVERRIDES += persist.bt.power.down=true
```

Define the attribute "persist.bt.power.down" as "true".

PRODUCT_PACKAGES example:

```
PRODUCT_PACKAGES += \
        rkaiq_tool_server \
        rkaiq_3A_server
```

Compile and install the rkaiq_tool_server module and the rkaiq_3A_server module.

6.3.3 Android Source Code Compilation

You can either use the built-in build.sh script provided by RK to compile the Android source code, or directly use the make command. This section will introduce this to the users. Before compiling, the following two tasks need to be completed:

①Initialize the compilation environment: source build/envsetup.sh

②Select the product: lunch rk3568_r-userdebug or lunch rk3568_r-user

1. Common Compilation Commands

In the Android source code directory, there is a Makefile file as follows:

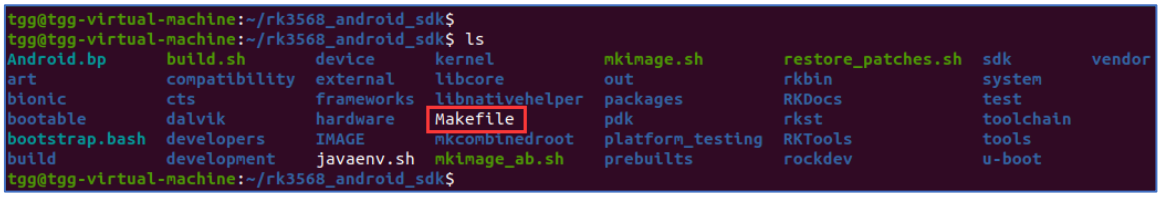


Figure 6.3.3.1 Android source code Makefile file

This Makefile file is the Makefile for the Android source code. Therefore, we can compile the Android source code using the make command. We can either compile the entire Android source code or compile a specific module separately.

① , Image compilation command

Compile command	Function
make	Without any parameters, it indicates compiling the entire Android source code. The intermediate files generated during the compilation process as well as the final image will be output to the out/target/product/$(PRODUCT_DEVICE) directory. We call this directory the product output directory.
make bootimage	Compile the boot.img image
make systemimage	Compile the system.img image
make vendorimage	Compile the vendor.img image
make productimage	Compile the product.img image
make odmimage	Compile the odm.img image
make systemextimage	Compile the system_ext.img image
make recoveryimage	Compile the recovery.img image
make cacheimage	Compile the cache.img image
make userdataimage	Compile the userdata.img image
make vbmetaimage	Compile the vbmeta.img image
make superimage	Compile the super.img image

Table 6.3.3.1 Make command for image compilation

② , Module compilation command

Compilation instructions for several major modules:

Module	"make" command
init	make init
zygote	make app_process
system_server	make services
java framework	make framework
Framework resource	make framework-res
jni framework	make libandroid_runtime
binder	make libbinder
Settings	make Settings
……	make xxx

Table 6.3.3.2 Compilation Commands for the Make Module

Each module can be compiled independently by executing the command "make <module_name>".

③ , Clearing Commands

There are several clearing commands:

Make command	Explanation
make clean	Completely remove, which is equivalent to deleting all contents in the out/ directory.
make clobber	The effect is the same as "make clean", also deleting all contents in the out directory.
make installclean	Delete all img image files and all.zip files in out/target/product/$(PRODUCT_DEVICE)/, delete the system,

	vendor, product, data, recovery, odm, ramdisk, root, cache directories, as well as some other files. It's impossible to list them all here. This command will be introduced separately later.
make dataclean	It is equivalent to performing the following operations: rm -rf out/target/product/$(PRODUCT_DEVICE)/data/*
make clean-<module>	Clear a single specific module

Table 6.3.3.3 make Clear Command

④, Introduction of commands provided by the build/envsetup.sh script

Before compiling the Android source code, you need to execute the "**source build/envsetup.sh**" command to initialize the compilation environment. This command needs to be executed every time a new terminal is opened. Then, execute the lunch command to select a product. Here, it should be noted that this lunch command is not provided by the Ubuntu system, but it appears only after executing the "**source build/envsetup.sh**" command. In other words, the lunch command is provided by the build/envsetup.sh script, not by the Ubuntu system itself.

If the "**source build/envsetup.sh**" command is not executed, the lunch command will not appear.

The build/envsetup.sh script provides not only the lunch command but also many other commands, such as m, mm, mmm, mma, mmma, hmm, and other commands. You can execute the "**hmm**" command to view the help information, as shown below:

```
hmm
```

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ hmm

Run "m help" for help with the build system itself.

Invoke ". build/envsetup.sh" from your shell to add the following functions to your environment:
- lunch:      lunch <product_name>-<build_variant>
              Selects <product_name> as the product to build, and <build_variant> as the variant to
              build, and stores those selections in the environment to be read by subsequent
              invocations of 'm' etc.
- tapas:      tapas [<App1> <App2> ...] [arm|x86|mips|arm64|x86_64|mips64] [eng|userdebug|user]
- croot:      Changes directory to the top of the tree, or a subdirectory thereof.
- m:          Makes from the top of the tree.
- mm:         Builds and installs all of the modules in the current directory, and their
              dependencies.
- mmm:        Builds and installs all of the modules in the supplied directories, and their
              dependencies.
              To limit the modules being built use the syntax: mmm dir/:target1,target2.
- mma:        Same as 'mm'
- mmma:       Same as 'mmm'
- provision:  Flash device with all required partitions. Options will be passed on to fastboot.
- cgrep:      Greps on all local C/C++ files.
- ggrep:      Greps on all local Gradle files.
- gogrep:     Greps on all local Go files.
- jgrep:      Greps on all local Java files.
- resgrep:    Greps on all local res/*.xml files.
- mangrep:    Greps on all local AndroidManifest.xml files.
- mgrep:      Greps on all local Makefiles and *.bp files.
- owngrep:    Greps on all local OWNERS files.
- sepgrep:    Greps on all local sepolicy files.
- sgrep:      Greps on all local source files.
- godir:      Go to the directory containing a file.
- allmod:     List all modules.
- gomod:      Go to the directory containing a module.
- pathmod:    Get the directory containing a module.
- refreshmod: Refresh list of modules for allmod/gomod.

Environment options:
- SANITIZE_HOST: Set to 'address' to use ASAN for all host modules.
- ANDROID_QUIET_BUILD: set to 'true' to display only the essential messages.

Look at the source to view more functions. The complete list is:
addcompletions add_lunch_combo allmod build_build_var_cache cgrep check_product check_type check_variant choosecombo choosepr
able coredump_setup cproj _croot croot destroy_build_var_cache enable_zsh_completion get_abs_build_var getbugreports get_buil
rdpath gettargetarch gettop ggrep godir gogrep gomod hmm is isviewserverstarted jgrep key_back key_home key_menu _lunch lunch
nch_menu provision qpid rcgrep refreshmod resgrep runhat runtest sepgrep setpaths set_sequence_number set_stuff_for_environme
opviewserver systemstack tapas tracedmdump treegrep _trigger_build validate_current_shell _wrap_build
tgg@tgg-virtual-machine:~/rk3568_android_sdk$
```

Figure 6.3.3.2 Execute "hmm" to view the help information.

Please provide the text for translation.

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ hmm
```

```
Run "m help" for help with the build system itself.

Invoke ". build/envsetup.sh" from your shell to add the following functions to your environment:
- lunch:          lunch <product_name>-<build_variant>
                    Selects <product_name> as the product to build, and <build_variant> as the variant to
                    build, and stores those selections in the environment to be read by subsequent
                    invocations of 'm' etc.
- tapas:                tapas [<App1> <App2> ...] [arm|x86|mips|arm64|x86_64|mips64] [eng|userdebug|user]
- croot:          Changes directory to the top of the tree, or a subdirectory thereof.
- m:              Makes from the top of the tree.
- mm:             Builds and installs all of the modules in the current directory, and their
                  dependencies.
- mmm:            Builds and installs all of the modules in the supplied directories, and their
                  dependencies.
                  To limit the modules being built use the syntax: mmm dir/:target1,target2.
- mma:            Same as 'mm'
- mmma:           Same as 'mmm'
- provision:      Flash device with all required partitions. Options will be passed on to fastboot.
- cgrep:          Greps on all local C/C++ files.
- ggrep:          Greps on all local Gradle files.
- gogrep:         Greps on all local Go files.
- jgrep:          Greps on all local Java files.
- resgrep:        Greps on all local res/*.xml files.
- mangrep:        Greps on all local AndroidManifest.xml files.
- mgrep:          Greps on all local Makefiles and *.bp files.
- owngrep:        Greps on all local OWNERS files.
- sepgrep:        Greps on all local sepolicy files.
- sgrep:          Greps on all local source files.
- godir:          Go to the directory containing a file.
- allmod:         List all modules.
- gomod:          Go to the directory containing a module.
- pathmod:        Get the directory containing a module.
- refreshmod: Refresh list of modules for allmod/gomod.

Environment options:
- SANITIZE_HOST: Set to 'address' to use ASAN for all host modules.
- ANDROID_QUIET_BUILD: set to 'true' to display only the essential messages.

Look at the source to view more functions. The complete list is:
```

addcompletions add_lunch_combo allmod build_build_var_cache cgrep check_product check_type check_variant choosecombo chooseproduct choosetype choosevariant _complete_android_module_names core coredump_enable coredump_setup cproj _croot croot destroy_build_var_cache enable_zsh_completion get_abs_build_var getbugreports get_build_var getlastscreenshot get_make_command getprebuilt getscreenshotpath getsdcardpath gettargetarch gettop ggrep godir gogrep gomod hmm is isviewserverstarted jgrep key_back key_home key_menu _lunch lunch m make mangrep mgrep mm mma mmm mmma owngrep pathmod pez printconfig print_lunch_menu provision qpid rcgrep refreshmod resgrep runhat runtest sepgrep setpaths set_sequence_number set_stuff_for_environment sgrep should_add_completion smoketest source_vendorsetup startviewserver stopviewserver systemstack tapas tracedmdump treegrep _trigger_build validate_current_shell _wrap_build

From this, we can see that the build/envsetup.sh script provides many commands. Let's briefly introduce them next.

Compilation instructions:

Command	Explanation
m	Perform the compilation operation at the top-level directory of the Android source code, that is, execute the "make" command in the top-level directory of the Android source code. For example: m -j16 m bootimage -j16 m systemimage -j16 However, there is no need to execute the "m" command at the top-level directory of the Android source code. It can be executed in any subdirectory of the Android source code, which is equivalent to: croot; make
mm	Compile all modules in the current directory, excluding dependencies.
mmm <module_path>	Compile all modules in the current directory, excluding dependencies.
mma	Compile all modules in the current directory, including dependencies.
mmma <module_name>	Compile all modules in the current directory, excluding dependencies.

Table 6.3.3.4 m/mm/mmm Compilation Command

The Android source code is extremely large. Using grep/ag to search for code directly would consume a lot of time. Based on specific requirements, one can choose appropriate search commands to effectively save the time for code search and improve accuracy.

Search command:

Command	Explanation
cgrep	Perform grep search operations on all C/C++ files
jgrep	Perform grep search operations on all Java files
ggrep	Perform grep search operations on all Gradle files
mgrep	Perform grep search operations on all Makefile files and *.bp files
sepgrep	Perform grep search operations on all sepolicy files
resgrep	Perform grep search operations on all res/*.xml files
gogrep	Perform grep search operations on all go files

mangrep	Perform grep search operations on all AndroidManifest.xml files
…	…

Table 6.3.3.4 Grep Search Command

Other commands:

Command	Explanation
croot	Executing the "croot" command allows you to switch to the root directory (top-level directory) of the Android source code. This command needs to be invoked in the root directory of the Android source code or any of its subdirectories.
hmm	Viewing help information
check_product	Checking if the product is valid
get_abs_build_var	Obtaining the value of a certain compilation variable
get_build_var	Obtaining the value of a certain compilation variable
printconfig	Printing the configuration information corresponding to the currently selected product
…	…

Table 6.3.3.5 Other Commands

2. Compile the entire Android source code

Execute the "make" command in the root directory of the Android source code, without any parameters (-jX parameter excluded, which indicates the number of compilation threads), and it will compile the entire Android source code:

```
make -j16
```

Or execute the "m" command (which can be executed in any subdirectory of the Android source code):

```
m -j16
```

All the files generated during the Android compilation process are separated from the source code. All the intermediate files and the final results generated during the compilation are stored in the "out" directory. The structure of the "out" directory is as follows:

```
out/
├── Android.mk
├── build_date.txt
├── build_error
├── casecheck.txt
├── CaseCheck.txt
├── CleanSpec.mk
├── ninja_build
├── soong
├── soong_ui
├── commit_id.xml
├── dumpvars-build_error
├── host          # The tools and libraries currently required for the compiling host (an Ubuntu host) are mostly of the x86 architecture.
│   ├── common
```

```
│   └── linux-x86
├── target      # Store all kinds of binary files, library files, images, etc. that are required for the operation of the target machine.
│   ├── common
│   └── product
└── ...
```

The "host" and "target" are the two subdirectories we need to pay attention to. The "host" directory stores various files that the current compiling host (that is, our Ubuntu host) needs, such as the tools and libraries that the compiling host uses; the "target" directory stores various files generated for the target machine, including various intermediate files, binary files, Android APKs, Java libraries, .so libraries, configuration files, and the final img image, etc.

The structure of the "host" directory is as follows:

```
host/
├── common
│   └── obj      # Including JAVA libraries
└── linux-x86
    ├── bin        # Binary programs, such as mkbootimg and adb, etc.
    ├── com.android.i18n
    ├── com.android.tzdata
    ├── fake_packages
    ├── framework          # JAVA library, .jar file
    ├── lib                #.so DLL
    ├── lib64              #64-bit.so dynamic link library
    ├── obj                # The intermediate files generated during compilation (object files, 64-bit)
    ├── obj32              # The intermediate files generated during compilation (object file, 32-bit)
    └── usr
```

The directory structure of the target is as follows:

```
target/
├── common
│   ├── obj
│   └── R
└── product
    └── rk3568_r      # The name of this directory is determined by the variable PRODUCT_DEVICE.
```

The key focus is on the "rk3568_r" directory. This directory is the output directory for the product.

```
tree -L 1 out/target/product/rk3568_r/
out/target/product/rk3568_r/
├── boot-debug.img          # Supports root-level operations and debug capabilities for the boot.img image.
├── boot.img                # "boot.img" will be written to the boot partition of the development board.
├── cache                   # Used to create cache.img
├── cache.img               # It will be written to the cache partition of the development board and mounted to the /cache directory of the Android root file system. The RK platform does not use this.
├── clean_steps.mk
```

```
    ├── data                    # This file is used to create userdata.img, which is mounted to the Android root
file system's /data directory. The RK platform does not use it.
    ├── debug_ramdisk           # Used to create ramdisk-debug.img
    ├── dtb.img                 # Kernel DTB
    ├── fake_packages
    ├── gen
    ├── kernel                  # Kernel Image
    ├── obj                     # This directory is used to store the target files (or intermediate files), in 64-bit
format.
    ├── obj_arm                 # This directory is used to store the target files (or intermediate files), 32-bit
    ├── odm                     # Used to create odm.img
    ├── odm.img                 # odm.img has been mounted to the root file system of Android in the /odm
directory.
    ├── parameter.txt           # RK platform partition table file
    ├── previous_build_config.mk
    ├── product                 # Used to create product.img
    ├── product.img             # "product.img" has been mounted to the root file system of Android in the
"/product" directory.
    ├── ramdisk                 # Used to create ramdisk.img
    ├── ramdisk-debug.img       # RAMdisk.img supporting the debug debugging function
    ├── ramdisk.img             # In the normal mode, the ramdisk.img will be packaged into the boot.img.
    ├── ramdisk-recovery.img    # In the recovery mode, the ramdisk.img will be packaged into the recovery.img.
    ├── rebuild-dtbo.img        #dtbo.img
    ├── recovery                # Used to create ramdisk-recovery.img
    ├── recovery.img            # "recovery.img" will be written to the "recovery" partition of the development
board.
    ├── root                    # Android root file system directory
    ├── super_empty.img
    ├── super.img               # "super.img", the sparse image format, will be written to the "super" partition
of the development board.
    ├── system                  # Used to create system.img
    ├── system_ext              # Used to create system_ext.img
    ├── system_ext.img          # Mounted to the /system_ext directory of the Android root file system
    ├── system.img              # Mounted to the /system directory of the Android root file system
    ├── testcases
    ├── vendor                  # Used to create vendor.img
    └── vendor.img              # Mounted to the /vendor directory of the Android root file system
```

The "obj" and "obj_arm" directories are used to store the target files, which are the intermediate files. They are divided into 64-bit platform and 32-bit platform. The 64-bit platform targets are stored in the "obj" directory, while the 32-bit platform targets are stored in the "obj_arm" directory. The directory structure is as follows:

```
    obj/
```

```
    ├── APPS                    # Store Android applications, various APK files, and the intermediate files generated during compilation.
    ├── CONFIG
    ├── ETC                     # The intermediate compiled files of etc usually end up being generated in /system/etc or /vendor/etc.
    ├── EXECUTABLES             # Various executable files, as well as the intermediate files generated during the compilation process, such as the.o files
    ├── FAKE
    ├── JAVA_LIBRARIES          # Java jar library files and the intermediate files generated during the compilation process
    ├── NATIVE_TESTS
    ├── NOTICE_FILES
    ├── NOTICE_ODM.txt
    ├── NOTICE_ODM.xml
    ├── NOTICE_ODM.xml.gz
    ├── NOTICE_PRODUCT.txt
    ├── NOTICE_PRODUCT.xml
    ├── NOTICE_PRODUCT.xml.gz
    ├── NOTICE_SYSTEM_EXT.txt
    ├── NOTICE_SYSTEM_EXT.xml
    ├── NOTICE_SYSTEM_EXT.xml.gz
    ├── NOTICE.txt
    ├── NOTICE_VENDOR.txt
    ├── NOTICE_VENDOR.xml
    ├── NOTICE_VENDOR.xml.gz
    ├── NOTICE.xml
    ├── NOTICE.xml.gz
    ├── PACKAGING               # Intermediate files during mirror packaging
    ├── RENDERSCRIPT_BITCODE
    ├── SHARED_LIBRARIES        #. So, the dynamic library files and the intermediate files generated during the compilation process
    └── STATIC_LIBRARIES        #.a static library file and the intermediate files generated during the compilation process
```

```
    obj_arm/
    ├── EXECUTABLES             # Executable files as well as the intermediate files generated during the compilation process, such as .o files
    ├── NATIVE_TESTS
    ├── RENDERSCRIPT_BITCODE
    ├── SHARED_LIBRARIES        #. So, the dynamic library files as well as the intermediate files generated during the compilation process
    └── STATIC_LIBRARIES        #.a static library file and the intermediate files generated during the compilation process
```

That's all about the "out" directory!

3. Compile boot.img

Execute the following command to compile the boot.img image (please execute it in the root directory of the Android source code):

```
make bootimage -j16
```

Or compile boot.img using the "m" command (which can be executed in any subdirectory of the Android source code):

```
m bootimage -j16
```

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ make bootimage -j16
============================================
PLATFORM_VERSION_CODENAME=REL
PLATFORM_VERSION=11
TARGET_PRODUCT=rk3568_r
TARGET_BUILD_VARIANT=userdebug
TARGET_BUILD_TYPE=release
TARGET_ARCH=arm64
TARGET_ARCH_VARIANT=armv8-a
TARGET_CPU_VARIANT=cortex-a55
TARGET_2ND_ARCH=arm
TARGET_2ND_ARCH_VARIANT=armv8-2a
TARGET_2ND_CPU_VARIANT=cortex-a55
HOST_ARCH=x86_64
HOST_2ND_ARCH=x86
HOST_OS=linux
HOST_OS_EXTRA=Linux-5.15.0-53-generic-x86_64-Ubuntu-20.04.2-LTS
HOST_CROSS_OS=windows
HOST_CROSS_ARCH=x86
HOST_CROSS_2ND_ARCH=x86_64
HOST_BUILD_TYPE=release
BUILD_ID=RD2A.211001.002
OUT_DIR=out
============================================
09:55:13 Disallowed PATH tool "python3.8" used: []string{"python3.8", ".repo/repo/repo", "manifest", "-r", "-o", "
09:55:13 See https://android.googlesource.com/platform/build/+/master/Changes.md#PATH_Tools for more information.
[100% 1/1] Target boot image: out/target/product/rk3568_r/boot.img

#### build completed successfully (3 seconds) ####
```

Figure 6.3.3.3 Compile boot.img

The path where the boot.img image file is located is: out/target/product/rk3568_r/boot.img.

The boot.img will be burned into the boot partition of the development board. boot.img is an Android format boot image, which is formed by merging multiple images. For the RK3568 platform, boot.img includes four parts: kernel image Image, kernel DTB, ramdisk, and resource.img. You can use the unpack_bootimg tool to unpack it, as shown below:

Note: If the "unpack_bootimg" tool is not available, it needs to be compiled using the command "make unpack_bootimg -j16".

```
cd out/target/product/rk3568_r/
mkdir temp_dir
cd temp_dir/
cp ../boot.img ./
unpack_bootimg --boot_img boot.img
ls out/
```

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk$
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ cd out/target/product/rk3568_r/
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r$ mkdir temp_dir
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r$ cd temp_dir/
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$ cp ../boot.img ./
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$ ls
boot.img
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$ unpack_bootimg --boot_img boot.img
boot_magic: ANDROID!
kernel_size: 32028680
kernel load address: 0x10008000
ramdisk size: 822327
ramdisk load address: 0x11000000
second bootloader size: 837120
second bootloader load address: 0x10f00000
kernel tags load address: 0x10000100
page size: 2048
os version: 11.0.0
os patch level: 2021-10
boot image header version: 2
product name:
command line args: console=ttyFIQ0 androidboot.baseband=N/A androidboot.wificountrycode=CN androidboot.veritymode=enforc
bootstate=orange firmware_class.path=/vendor/etc/firmware init=/init rootwait ro loop.max_part=7 buildvariant=userdebug
additional command line args:
recovery dtbo size: 0
recovery dtbo offset: 0x0
boot header size: 1660
dtb size: 142226
dtb address: 0x11f00000
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$ ls
boot.img  out
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$ ls out/
dtb  kernel  ramdisk  second
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$
```

Nnpack

Figure 6.3.3.4 Unpacking boot.img

The specific situation is as shown in the following table:

Mirror image	Explanation
dtb	This is the kernel device tree image. However, this dtb is just a decoration and will not take effect in the system. The actual effective one is the dtb contained in the resource.img file.
kernel	This dtb is actually the out/target/product/rk3568_r/dtb.img file.
ramdisk	The kernel image is actually the out/target/product/rk3568_r/kernel file.
second	The ramdisk image is actually the out/target/product/rk3568_r/ramdisk.img file.

Table 6.3.3.6 Detailed Explanation of Components of boot.img

Before compiling the boot.img image, it is necessary to first compile the Linux Kernel, obtaining the kernel image, kernel DTB, and resource.img. When compiling the boot.img, the kernel image (kernel/arch/arm64/boot/Image) will be copied to out/target/product/rk3568_r/kernel, and the kernel DTB (for example: kernel/arch/arm64/boot/dts/rockchip/rk3568-atk-evb1-ddr4-v10.dtb) will be copied to out/target/product/rk3568_r/dtb.img. The resource.img will not be copied over; instead, it will directly search for the file in the kernel/ directory.

These mirrors are defined by the product board-level configuration file BoardConfig.mk:

BOARD_PREBUILT_DTBIMAGE_DIR variable

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/device/rockchip/rk356x$ grep -nR BOARD_PREBUILT_DTBIMAGE_DIR
BoardConfig.mk:21:BOARD_PREBUILT_DTBIMAGE_DIR := kernel/arch/arm64/boot/dts/rockchip
tgg@tgg-virtual-machine:~/rk3568_android_sdk/device/rockchip/rk356x$
```

Figure 6.3.3.5 Definition of DTB image path

PRODUCT_KERNEL_DTS variable

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/device/rockchip/rk356x$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/device/rockchip/rk356x$ grep -nR PRODUCT_KERNEL_DTS
rk3566_eink/BoardConfig.mk:31:PRODUCT_KERNEL_DTS := rk3566-rk817-eink-w103
rk3566_r/BoardConfig.mk:28:PRODUCT_KERNEL_DTS := rk3566-evb2-lp4x-v10
rk3566_32bit/BoardConfig.mk:19:PRODUCT_KERNEL_DTS := rk3566-rk817-tablet
rk3568_r/BoardConfig.mk:29:PRODUCT_KERNEL_DTS := rk3568-atk-evb1-ddr4-v10   rk3568_r
BoardConfig.mk:25:PRODUCT_KERNEL_DTS ?= rk3568-evb1-ddr4-v10
rk3566_rgo/BoardConfig.mk:19:PRODUCT_KERNEL_DTS := rk3566-rk817-tablet
rk3566_einkw6/BoardConfig.mk:32:PRODUCT_KERNEL_DTS := rk3566-rk817-eink-w6
tgg@tgg-virtual-machine:~/rk3568_android_sdk/device/rockchip/rk356x$
```

Figure 6.3.3.6 DTB file name definition

TARGET_PREBUILT_KERNEL variable

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/device/rockchip/rk356x$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/device/rockchip/rk356x$ grep -nR TARGET_PREBUILT_KERNEL
BoardConfig.mk:20:TARGET_PREBUILT_KERNEL := kernel/arch/arm64/boot/Image
tgg@tgg-virtual-machine:~/rk3568_android_sdk/device/rockchip/rk356x$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/device/rockchip/rk356x$
```

Figure 6.3.3.7 Kernel image definition

TARGET_PREBUILT_RESOURCE variable

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/device/rockchip$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/device/rockchip$ grep -nR TARGET_PREBUILT_RESOURCE
rk3399/rk3399_mid/BoardConfig.mk:13:    BOARD_RECOVERY_MKBOOTIMG_ARGS := --second $(TARGET_PREBUILT_RESOURCE) --
common/BoardConfig.mk:66:TARGET_PREBUILT_RESOURCE ?= kernel/resource.img
common/BoardConfig.mk:102:BOARD_MKBOOTIMG_ARGS += --second $(TARGET_PREBUILT_RESOURCE)
common/BoardConfig.mk:111:BOARD_RECOVERY_MKBOOTIMG_ARGS ?= --second $(TARGET_PREBUILT_RESOURCE) --header_version
rk3326/rk3326_pie/BoardConfig.mk:32:    BOARD_RECOVERY_MKBOOTIMG_ARGS := --second $(TARGET_PREBUILT_RESOURCE) --
tgg@tgg-virtual-machine:~/rk3568_android_sdk/device/rockchip$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/device/rockchip$
```

Figure 6.3.3.8 Definition of the resource.img image

Definition of kernel DTB and kernel image Image copy:

device/rockchip/common/device.mk:94

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/device/rockchip$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/device/rockchip$ grep -nR "\$(TARGET_PREBUILT_KERNEL):kernel"
common/device.mk:94:    $(TARGET_PREBUILT_KERNEL):kernel
tgg@tgg-virtual-machine:~/rk3568_android_sdk/device/rockchip$
```

```
90  ##########################################################
91  # Kernel
92  ##########################################################
93  PRODUCT_COPY_FILES += \
94      $(TARGET_PREBUILT_KERNEL):kernel    Kernel image copy
95
96  ##########################################################
97  # dtb
98  ##########################################################
99  PRODUCT_COPY_FILES += \                                   Kernel DTB copy
100     $(BOARD_PREBUILT_DTBIMAGE_DIR)/$(PRODUCT_KERNEL_DTS).dtb:dtb.img
101
```

Figure 6.3.3.9 Definition of Mirror Copy

The boot.img file is ultimately created by the mkbootimg tool (located at out/host/linux-x86/bin/mkbootimg). The creation command is as follows:

```
[100% 277/277] -e \"<<
```

```
out/host/linux-x86/bin/mkbootimg --kernel out/target/product/rk3568_r/kernel --ramdisk out/target/product/rk3568_r/ramdisk.img
--dtb out/target/product/rk3568_r/dtb.img --cmdline "console=ttyFIQ0 androidboot.baseband=N/A androidboot.wificountrycode=CN a
ndroidboot.veritymode=enforcing androidboot.hardware=rk30board androidboot.console=ttyFIQ0 androidboot.verifiedbootstate=orange
firmware_class.path=/vendor/etc/firmware init=/init rootwait ro loop.max_part=7 buildvariant=userdebug" --os_version 11 --os_p
atch_level 2021-10-01 --second kernel/resource.img --header_version 2 --output out/target/product/rk3568_r/boot.img
```

Note-taking:

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/build/make$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/build/make$ pwd
/home/tgg/rk3568_android_sdk/build/make
tgg@tgg-virtual-machine:~/rk3568_android_sdk/build/make$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/build/make$ grep -nR "bootimage:"
core/main.mk:1416:vendorbootimage: $(INSTALLED_VENDOR_BOOTIMAGE_TARGET)
core/main.mk:1437:bootimage: $(INSTALLED_BOOTIMAGE_TARGET)
tgg@tgg-virtual-machine:~/rk3568_android_sdk/build/make$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/build/make$
```

To learn about the boot.img image creation process, you can start from build/make/core/main.mk:1437!

Key variables: BUILDING_BOOT_IMAGE, BOARD_BOOTIMAGE_PARTITION_SIZE

4. Compile system.img

system.img will provide the commands needed by the Android system, built-in apps, running dynamic libraries, and system configuration files, etc.

Execute the following command to compile the system.img image (execute it in the root directory of the Android source code):

```
make systemimage -j16
```

Or compile system.img using the "m" command (this can be done in any subdirectory of the Android source code):

```
m systemimage -j16
```

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk$
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ make systemimage -j16
============================================
PLATFORM_VERSION_CODENAME=REL
PLATFORM_VERSION=11
TARGET_PRODUCT=rk3568_r
TARGET_BUILD_VARIANT=userdebug
TARGET_BUILD_TYPE=release
TARGET_ARCH=arm64
TARGET_ARCH_VARIANT=armv8-a
TARGET_CPU_VARIANT=cortex-a55
TARGET_2ND_ARCH=arm
TARGET_2ND_ARCH_VARIANT=armv8-2a
TARGET_2ND_CPU_VARIANT=cortex-a55
HOST_ARCH=x86_64
HOST_2ND_ARCH=x86
HOST_OS=linux
HOST_OS_EXTRA=Linux-5.15.0-53-generic-x86_64-Ubuntu-20.04.2-LTS
HOST_CROSS_OS=windows
HOST_CROSS_ARCH=x86
HOST_CROSS_2ND_ARCH=x86_64
HOST_BUILD_TYPE=release
BUILD_ID=RD2A.211001.002
OUT_DIR=out
============================================
12:29:30 Disallowed PATH tool "python3.8" used: []string{"python3.8", ".repo/repo/repo", "manifest", "-r", "-o", "o
12:29:30 See https://android.googlesource.com/platform/build/+/master/Changes.md#PATH_Tools for more information.
[100% 1/1] Install system fs image: out/target/product/rk3568_r/system.img

#### build completed successfully (4 seconds) ####
```

Figure 6.3.3.11 Compile system.img

The path where the system.img image file is located is: out/target/product/rk3568_r/system.img.

The system.img is not burned onto the development board alone, but is packaged into super.img.

Starting from Android 10, the **system-as-root** partition layout is used. ramdisk.img is no longer used as the root directory (/), but is merged into system.img, and then system.img is mounted as the root file system (rootfs); therefore, ramdisk.img is no longer used at present. To ensure compatibility with devices upgraded to Android 10, ramdisk.img is still retained.

In Android 11, system.img is a mirror file in the Android sparse image format. To view system.img using the file command, it is as follows:

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk$
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ file out/target/product/rk3568_r/system.img
out/target/product/rk3568_r/system.img: Android sparse image, version: 1.0, Total of 239401 4096-byte output blocks in 18 input chunks.
tgg@tgg-virtual-machine:~/rk3568_android_sdk$
```

Figure 6.3.3.12 The "file" command is used to view the "system.img".

The "simg2img" tool can be used to convert the sparse image into the original format.

```
rm -rfd temp_dir
mkdir temp_dir
cd temp_dir/
cp ../system.img ./
simg2img system.img system_raw.img
file system_raw.img
```

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r$ pwd
/home/tgg/rk3568_android_sdk/out/target/product/rk3568_r
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r$ rm -rfd temp_dir
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r$ mkdir temp_dir
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r$ cd temp_dir/
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$ cp ../system.img ./
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$ ls
system.img
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$ simg2img system.img system_raw.img
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$ ls
system.img  system_raw.img
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$ file system_raw.img
system_raw.img: Linux rev 1.0 ext2 filesystem data, UUID=b699ca26-19f0-4619-968a-1a749ae5de32 (extents) (large files) (huge files)
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$
```

Figure 6.3.3.13 Convert the sparse image back to its original format

Note: If the simg2img tool is not available, it needs to be compiled using the command "make simg2img -j16".

The system_raw.img obtained through the simg2img tool is an ext4 format file, and it can be directly mounted to the Ubuntu system using the mount command:

```
mkdir root
sudo mount system_raw.img root/
```

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$ mkdir root
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$ sudo mount system_raw.img root/
[sudo] tgg 的密码:
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$ ls ./root/
acct  bin        cache  d     data_mirror   default.prop  etc   init.environ.rc  lost+found  mnt  oem   product  sdcard   sys     system_ext
apex  bugreports config data  debug_ramdisk dev           init  linkerconfig     metadata    odm  proc  res      storage  system  vendor
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$
```

Figure 6.3.3.14 Mount the "system_raw.img"

This is actually the directory structure of the Android root file system. This directory structure is the same as that of "/home/tgg/rk3568_android_sdk/out/target/product/rk3568_r/root". As mentioned earlier, starting from Android 10, system-as-root is used, and the "system.img" is mounted as the root file system (rootfs), instead of using "ramdisk.img" as the root directory (Android versions before 10 did not support the system-as-root partition layout and mounted "ramdisk.img" as the root file system). The content of "system.img" is equivalent to:

```
/home/tgg/rk3568_android_sdk/out/target/product/rk3568_r/root
+
/home/tgg/rk3568_android_sdk/out/target/product/rk3568_r/system
```

Table of Contents is used to replace /home/tgg/rk3568_android_sdk/out/target/product/rk3568_r/syste to /home/tgg/rk3568_android_sdk/out/target/product/rk3568_r/root/system.

映像	ramdisk（Android 9 之前）	system-as-root（Android 9 之后）
boot.img	包含内核和 ramdisk.img: ramdisk.img -/ - init.rc - init - etc -> /system/etc - system/ (mount point) - vendor/ (mount point) - odm/ (mount point) ...	仅包含正常启动内核。
recovery.img	包含恢复内核和恢复 ramdisk.img。	
system.img	包含以下内容: system.img -/ - bin/ - etc - vendor -> /vendor - ...	包含原始 system.img 和 ramdisk.img 的合并内容: system.img -/ - init.rc - init - etc -> /system/etc - system/ - bin/ - etc/ - vendor -> /vendor - ... - vendor/ (mount point) - odm/ (mount point) ...

Figure 6.3.3.15 System-as-root legend explanation

Reference link:

https://source.android.google.cn/docs/core/architecture/bootloader/system-as-root?hl=zh-cn

The img2simg tool can be used to convert the system_raw.img file back into the sparse image format.

```
sudo umount root        # unload
rm -rf system.img
img2simg system_raw.img system.img
file system.img
```

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$ sudo umount root
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$ ls
root  system.img  system_raw.img
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$ rm -rf system.img
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$ ls
root  system_raw.img
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$ img2simg system_raw.img system.img
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$ file system.img
system.img: Android sparse image, version: 1.0, Total of 239401 4096-byte output blocks in 2647 input chunks.
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$
```

Note: If the "img2simg" tool is not available, it needs to be compiled using the command "make img2simg -j16".

To learn about the process of creating the system.img image file, you can start by referring to line 2801 of the build/make/core/Makefile!

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/build/make$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/build/make$ grep -nR "systemimage:"
core/Makefile:1042:systemimage:
core/Makefile:2801:systemimage: $(INSTALLED_SYSTEMIMAGE_TARGET)
core/Makefile:5368:systemimage: $(INSTALLED_QEMU_SYSTEMIMAGE)
core/Makefile:5429:systemimage: $(QEMU_VERIFIED_BOOT_PARAMS)
tgg@tgg-virtual-machine:~/rk3568_android_sdk/build/make$
```

Key variables: BUILDING_SYSTEM_IMAGE, PRODUCT_BUILD_SYSTEM_IMAGE, INSTALLED_SYSTEMIMAGE_TARGET, INSTALLED_SYSTEMIMAGE, BUILT_SYSTEMIMAGE, BOARD_SYSTEMIMAGE_PARTITION_SIZE

5. Compile vendor.img

After system startup, the vendor.img image will be mounted under the Android root file system's /vendor directory. The vendor.img provides manufacturer-specific executable programs, libraries, system services, and apps, etc. Some custom features of the manufacturer can be placed in the vendor.img.

Execute the following command to compile the vendor.img image (it needs to be executed in the root directory of the Android source code):

```
make vendorimage -j16
```

Or compile vendor.img using the "m" command (this can be done in any subdirectory of the Android source code):

```
m vendorimage -j16
```

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ make vendorimage -j16
============================================
PLATFORM_VERSION_CODENAME=REL
PLATFORM_VERSION=11
TARGET_PRODUCT=rk3568_r
TARGET_BUILD_VARIANT=userdebug
TARGET_BUILD_TYPE=release
TARGET_ARCH=arm64
TARGET_ARCH_VARIANT=armv8-a
TARGET_CPU_VARIANT=cortex-a55
TARGET_2ND_ARCH=arm
TARGET_2ND_ARCH_VARIANT=armv8-2a
TARGET_2ND_CPU_VARIANT=cortex-a55
HOST_ARCH=x86_64
HOST_2ND_ARCH=x86
HOST_OS=linux
HOST_OS_EXTRA=Linux-5.15.0-53-generic-x86_64-Ubuntu-20.04.2-LTS
HOST_CROSS_OS=windows
HOST_CROSS_ARCH=x86
HOST_CROSS_2ND_ARCH=x86_64
HOST_BUILD_TYPE=release
BUILD_ID=RD2A.211001.002
OUT_DIR=out
============================================
17:24:39 Disallowed PATH tool "python3.8" used: []string{"python3.8", ".repo/repo/repo", "manifest", "-r", "-o", "out
17:24:39 See https://android.googlesource.com/platform/build/+/master/Changes.md#PATH_Tools for more information.
[100% 75/75] Target vendor fs image: out/target/product/rk3568_r/vendor.img

#### build completed successfully (9 seconds) ####
```

Figure 6.3.3.15 Compile vendor.img

The path where the vendor.img is located is: out/target/product/rk3568_r/vendor.img.

The vendor.img is not burned onto the development board alone, but is packaged into super.img.

The vendor.img is created from the out/target/product/rk3568_r/vendor folder, and the content of vendor.img is the contents of the vendor directory. View vendor.img using the file command as follows:

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r$ pwd
/home/tgg/rk3568_android_sdk/out/target/product/rk3568_r
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r$ file vendor.img
vendor.img: Android sparse image, version: 1.0, Total of 75087 4096-byte output blocks in 8 input chunks.
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r$
```

Figure 6.3.3.16 The "file" command is used to view the contents of "vendor.img".

From this, it can be seen that vendor.img is also a sparse image file. It can be converted to the original format using the simg2img command. The original format of vendor.img is ext4. It can be mounted using the mount command to view its contents.

It should be noted that before compiling vendor.img, the Linux Kernel needs to be compiled first, and the kernel ko modules need to be compiled as well. For example, the WiFi driver module. On the RK platform, the WiFi driver module is usually packaged into vendor.img (vendor/lib/modules/).

The Android compilation system does not default to installing kernel modules. These need to be specified through the BOARD_XXX_KERNEL_MODULES variable. For example, the BOARD_VENDOR_KERNEL_MODULES variable indicates that the kernel modules will be installed to the vendor partition (vendor/lib/modules). The compilation system will run depmod on these modules and copy the depmod output files to the same location in the vendor partition. The BOARD_ODM_KERNEL_MODULES variable indicates that the kernel modules will be installed to the odm partition (odm/lib/modules). The compilation system will run depmod on these modules and copy the depmod output files to the same location in the odm partition.

Go to the directory vendor/rockchip/common and search for the BOARD_VENDOR_KERNEL_MODULES macro:

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/vendor/rockchip/common$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/vendor/rockchip/common$ grep -nR BOARD_VENDOR_KERNEL_MODULES
modular_kernel/update_makefiles.sh:17:VENDOR_MK_PREFIX="BOARD_VENDOR_KERNEL_MODULES"
gpu/MaliG52.mk:12:BOARD_VENDOR_KERNEL_MODULES += \
gpu/MaliG52.mk:15:BOARD_VENDOR_KERNEL_MODULES += \
gpu/MaliTDVx.mk:13:BOARD_VENDOR_KERNEL_MODULES += \
gpu/MaliTDVx.mk:16:BOARD_VENDOR_KERNEL_MODULES += \
gpu/G6110.mk:10:BOARD_VENDOR_KERNEL_MODULES += \
gpu/G6110.mk:79:BOARD_VENDOR_KERNEL_MODULES += \
gpu/MaliT860.mk:11:BOARD_VENDOR_KERNEL_MODULES += \
gpu/MaliT860.mk:14:BOARD_VENDOR_KERNEL_MODULES += \
hdcp2/hdcp2.mk:4:BOARD_VENDOR_KERNEL_MODULES += \
wifi/wifi.mk:3:BOARD_VENDOR_KERNEL_MODULES += \
tgg@tgg-virtual-machine:~/rk3568_android_sdk/vendor/rockchip/common$
```

Open the "wifi/wifi.mk" file, as shown below:

```
WIFI_KO_FILES := $(shell find $(TOPDIR)kernel/drivers/net/wireless/rockchip_wlan -name "*.ko" -type f)

BOARD_VENDOR_KERNEL_MODULES += \
        $(foreach file, $(WIFI_KO_FILES), $(file))

PRODUCT_COPY_FILES += \
        $(CUR_PATH)/wifi/iwconfig:$(TARGET_COPY_OUT_VENDOR)/bin/iwconfig \
        $(CUR_PATH)/wifi/iwlist:$(TARGET_COPY_OUT_VENDOR)/bin/iwlist

WifiFirmwareFile := $(shell ls $(CUR_PATH)/wifi/firmware)
PRODUCT_COPY_FILES += \
        $(foreach file, $(WifiFirmwareFile), $(CUR_PATH)/wifi/firmware/$(file):$(TARGET_COPY_OUT_VENDOR)/etc/firmware/$(file))
```

Install all the .ko modules in the directory "drivers/net/wireless/rockchip_wlan" of the kernel to the vendor partition.

Therefore, it can be concluded that if a kernel module needs to be installed in the vendor partition, the path of the module should be added to the BOARD_VENDOR_KERNEL_MODULES variable. In this way, the system will install it in the vendor partition during compilation. Of course, we can

configure the BOARD_VENDOR_KERNEL_MODULES variable in the board-level configuration file BoardConfig.mk of the product. If it needs to be installed in the odm partition, the path of the module should be added to the BOARD_ODM_KERNEL_MODULES variable.

Relevant variables:

BOARD_RECOVERY_KERNEL_MODULES
BOARD_VENDOR_RAMDISK_KERNEL_MODULES
BOARD_VENDOR_KERNEL_MODULES
BOARD_ODM_KERNEL_MODULES
BOARD_GENERIC_RAMDISK_KERNEL_MODULES

To learn about the vendor.img image creation process, you can start from build/make/core/main.mk: 1413!

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/build/make$ grep -nR "vendorimage:"
tools/releasetools/ota_from_target_files.py:542:  if board_uses_vendorimage:
tools/releasetools/make_recovery_patch.py:53:  if board_uses_vendorimage:
tools/releasetools/validate_target_files.py:165:  if board_uses_vendorimage:
tools/releasetools/ota_from_target_files:542:  if board_uses_vendorimage:
tools/releasetools/common.py:3032:  if board_uses_vendorimage:
core/Makefile:5375:vendorimage: $(INSTALLED_QEMU_VENDORIMAGE)
core/main.mk:1413:vendorimage: $(INSTALLED_VENDORIMAGE_TARGET)
tgg@tgg-virtual-machine:~/rk3568_android_sdk/build/make$
```

Key variables: BUILDING_VENDOR_IMAGE, PRODUCT_BUILD_VENDOR_IMAGE, BOARD_VENDORIMAGE_FILE_SYSTEM_TYPE, BUILDING_VENDOR_IMAGE, INSTALLED_VENDORIMAGE_TARGET

6. Compile product.img

After the system starts, the product.img image will be mounted to the /product directory of the Android root file system. The product.img image provides product/OEM (original equipment manufacturer) private libraries, apps, and configuration files, etc.

```
make productimage -j16
```

Or compile product.img using the "m" command (this can be done in any subdirectory of the Android source code):

```
m productimage -j16
```

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ make productimage -j16
============================================
PLATFORM_VERSION_CODENAME=REL
PLATFORM_VERSION=11
TARGET_PRODUCT=rk3568_r
TARGET_BUILD_VARIANT=userdebug
TARGET_BUILD_TYPE=release
TARGET_ARCH=arm64
TARGET_ARCH_VARIANT=armv8-a
TARGET_CPU_VARIANT=cortex-a55
TARGET_2ND_ARCH=arm
TARGET_2ND_ARCH_VARIANT=armv8-2a
TARGET_2ND_CPU_VARIANT=cortex-a55
HOST_ARCH=x86_64
HOST_2ND_ARCH=x86
HOST_OS=linux
HOST_OS_EXTRA=Linux-5.15.0-53-generic-x86_64-Ubuntu-20.04.2-LTS
HOST_CROSS_OS=windows
HOST_CROSS_ARCH=x86
HOST_CROSS_2ND_ARCH=x86_64
HOST_BUILD_TYPE=release
BUILD_ID=RD2A.211001.002
OUT_DIR=out
============================================
18:06:56 Disallowed PATH tool "python3.8" used: []string{"python3.8", ".repo/repo/repo", "manifest", "-r", "-o", "
18:06:56 See https://android.googlesource.com/platform/build/+/master/Changes.md#PATH_Tools for more information.
[100% 1/1] Target product fs image: out/target/product/rk3568_r/product.img

#### build completed successfully (5 seconds) ####
```

Figure 6.3.3.17 Compile product.img

The path where the product.img image file is located is: out/target/product/rk3568_r/product.img.

The product.img is not burned onto the development board alone, but is packaged into super.img.

The product.img is created from the out/target/product/rk3568_r/product folder, and its content is the contents of the product directory. View product.img using the file command as follows:

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ file out/target/product/rk3568_r/product.img
out/target/product/rk3568_r/product.img: Android sparse image, version: 1.0, Total of 52087 4096-byte output blocks in 5 input chunks.
tgg@tgg-virtual-machine:~/rk3568_android_sdk$
tgg@tgg-virtual-machine:~/rk3568_android_sdk$
```

Figure 6.3.3.18 The "file" command is used to view the contents of "product.img".

From this, it can be seen that product.img is also a spare image format mirror. It can be converted to the original format using the simg2img tool. The original format of product.img is ext4, and it can be directly mounted using the mount command to view its contents.

To learn about the production process of the product.img image, you can start from line 1422 of build/make/core/main.mk!

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/build/make$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/build/make$ grep -nR "productimage:"
core/Makefile:5398:productimage: $(INSTALLED_QEMU_PRODUCTIMAGE)
core/main.mk:1422:productimage: $(INSTALLED_PRODUCTIMAGE_TARGET)
tgg@tgg-virtual-machine:~/rk3568_android_sdk/build/make$
```

Key variables: PRODUCT_BUILD_PRODUCT_IMAGE, BUILDING_PRODUCT_IMAGE, BOARD_PRODUCTIMAGE_FILE_SYSTEM_TYPE, INSTALLED_PRODUCTIMAGE_TARGET, BUILT_PRODUCTIMAGE_TARGET

7. Compile odm.img

After system startup, the odm.img image will be mounted under the Android root file system directory /odm. The odm.img provides ODM (branding manufacturer) related customizations, including libraries, bin, etc., and the jar file.

Execute the following command to compile the odm.img image (execute it in the root directory of the Android source code):

```
make odmimage -j16
```

Or compile odm.img using the m command (it can be executed in any subdirectory of the Android source code):

```
m odmimage -j16
```

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ make odmimage -j16
============================================
PLATFORM_VERSION_CODENAME=REL
PLATFORM_VERSION=11
TARGET_PRODUCT=rk3568_r
TARGET_BUILD_VARIANT=userdebug
TARGET_BUILD_TYPE=release
TARGET_ARCH=arm64
TARGET_ARCH_VARIANT=armv8-a
TARGET_CPU_VARIANT=cortex-a55
TARGET_2ND_ARCH=arm
TARGET_2ND_ARCH_VARIANT=armv8-2a
TARGET_2ND_CPU_VARIANT=cortex-a55
HOST_ARCH=x86_64
HOST_2ND_ARCH=x86
HOST_OS=linux
HOST_OS_EXTRA=Linux-5.15.0-53-generic-x86_64-Ubuntu-20.04.2-LTS
HOST_CROSS_OS=windows
HOST_CROSS_ARCH=x86
HOST_CROSS_2ND_ARCH=x86_64
HOST_BUILD_TYPE=release
BUILD_ID=RD2A.211001.002
OUT_DIR=out
============================================
18:19:33 Disallowed PATH tool "python3.8" used: []string{"python3.8", ".repo/repo/repo", "manifest", "-r", "-o", "o
18:19:33 See https://android.googlesource.com/platform/build/+/master/Changes.md#PATH_Tools for more information.
[100% 1/1] Target odm fs image: out/target/product/rk3568_r/odm.img

#### build completed successfully (4 seconds) ####
```

Figure 6.3.3.19 Compile odm.img

The path where the odm.img image file is located is: out/target/product/rk3568_r/odm.img.

The odm.img is not burned onto the development board alone, but is packaged into super.img.

The odm.img is created from the out/target/product/rk3568_r/odm folder, and its content is the contents of the odm directory. View odm.img using the file command as follows:

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk$
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ file out/target/product/rk3568_r/odm.img
out/target/product/rk3568_r/odm.img: Android sparse image, version: 1.0, Total of 153 4096-byte output blocks in 2 input chunks.
tgg@tgg-virtual-machine:~/rk3568_android_sdk$
```

Figure 6.3.3.20 The "file" command is used to view the "odm.img" file.

From this, it can be seen that odm.img is also a spare image format mirror. It can be converted to the original format using the simg2img command. The original format is ext4, and it can be mounted using the mount command to view its contents.

To learn about the ODM.img image creation process, you can start from line 1428 of build/make/core/main.mk!

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/build/make$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/build/make$ grep -nR "odmimage:"
core/Makefile:5416:odmimage: $(INSTALLED_QEMU_ODMIMAGE)
core/main.mk:1428:odmimage: $(INSTALLED_ODMIMAGE_TARGET)
tgg@tgg-virtual-machine:~/rk3568_android_sdk/build/make$
```

Key variables: BUILDING_ODM_IMAGE, PRODUCT_BUILD_ODM_IMAGE, BOARD_ODMIMAGE_FILE_SYSTEM_TYPE, INSTALLED_ODMIMAGE_TARGET

8. Compile system_ext.img

In Android 11, a new feature called SSI was used, introducing the system_ext partition as an optional partition. This partition is the location for non-AOSP components that are tightly coupled with the components defined by AOSP in the /system partition. The system_ext partition is regarded as an extension of the /system partition specifically for the OEM and no interfaces are defined between the two partitions. Components in the system_ext partition can make private API calls to the /system partition, and vice versa, components in the /system partition can also make private API calls to the system_ext partition.

After the system is started, the system_ext.img image will be mounted to the /system_ext directory of the Android root file system. The system_ext.img image provides some apps, etc., and library files, etc.

Execute the following command to compile the system_ext.img image (it needs to be executed in the root directory of the Android source code):

```
make systemextimage -j16
```

Or, you can compile system_ext.img using the "m" command (this can be done in any subdirectory of the Android source code):

```
m systemextimage -j16
```

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk$
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ m systemextimage -j16
============================================
PLATFORM_VERSION_CODENAME=REL
PLATFORM_VERSION=11
TARGET_PRODUCT=rk3568_r
TARGET_BUILD_VARIANT=userdebug
TARGET_BUILD_TYPE=release
TARGET_ARCH=arm64
TARGET_ARCH_VARIANT=armv8-a
TARGET_CPU_VARIANT=cortex-a55
TARGET_2ND_ARCH=arm
TARGET_2ND_ARCH_VARIANT=armv8-2a
TARGET_2ND_CPU_VARIANT=cortex-a55
HOST_ARCH=x86_64
HOST_2ND_ARCH=x86
HOST_OS=linux
HOST_OS_EXTRA=Linux-5.15.0-53-generic-x86_64-Ubuntu-20.04.2-LTS
HOST_CROSS_OS=windows
HOST_CROSS_ARCH=x86
HOST_CROSS_2ND_ARCH=x86_64
HOST_BUILD_TYPE=release
BUILD_ID=RD2A.211001.002
OUT_DIR=out
============================================
19:07:33 Disallowed PATH tool "python3.8" used: []string{"python3.8", ".repo/repo/repo", "manifest", "-r", "-o", "
19:07:33 See https://android.googlesource.com/platform/build/+/master/Changes.md#PATH_Tools for more information.
[100% 1/1] Target system_ext fs image: out/target/product/rk3568_r/system_ext.img

#### build completed successfully (5 seconds) ####
```

Figure 6.3.3.21 Compile system_ext.img

The path where the system_ext.img image file is located is: out/target/product/rk3568_r/system_ext.img.

The system_ext.img is not burned onto the development board alone; instead, it is packaged into super.img.

The system_ext.img is created from the out/target/product/rk3568_r/system_ext folder, and the content of system_ext.img is the contents of the system_ext directory. View system_ext.img using the file command as follows:

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ file out/target/product/rk3568_r/system_ext.img
out/target/product/rk3568_r/system_ext.img: Android sparse image, version: 1.0, Total of 30192 4096-byte output blocks in 3 input chunks.
tgg@tgg-virtual-machine:~/rk3568_android_sdk$
tgg@tgg-virtual-machine:~/rk3568_android_sdk$
```

Figure 6.3.3.22 The "file" command is used to view the "system_ext.img" file.

From this, it can be seen that system_ext.img is also a spare image format image. It can be converted to the original format using the simg2img command. The original format is ext4, and it can be mounted using the mount command to view its contents.

To learn about the process of creating the system_ext.img image, you can start by referring to line 1428 of build/make/core/main.mk!

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/build/make$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/build/make$ grep -nR "systemextimage:"
core/Makefile:5407:systemextimage: $(INSTALLED_QEMU_SYSTEM_EXTIMAGE)
core/main.mk:1425:systemextimage: $(INSTALLED_SYSTEM_EXTIMAGE_TARGET)
tgg@tgg-virtual-machine:~/rk3568_android_sdk/build/make$
```

Key variables: BUILDING_SYSTEM_EXT_IMAGE、
PRODUCT_BUILD_SYSTEM_EXT_IMAGE、
BOARD_SYSTEM_EXTIMAGE_FILE_SYSTEM_TYPE、
INSTALLED_SYSTEM_EXTIMAGE_TARGET

9.Compile recovery.img

recovery.img is used to enter the recovery mode. Recovery is typically used for system restoration and upgrade. U-Boot will determine whether to boot the system into the normal startup mode (Normal mode) or the recovery mode based on the BCB data block stored in the misc partition.

Execute the following command to compile the recovery.img image (this needs to be executed in the root directory of the Android source code):

```
Make recoveryimage -j16
```

Or compile recovery.img using the "m" command (this can be done in any subdirectory of the Android source code):

```
m recoveryimage -j16
```

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk$
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ make recoveryimage -j16
============================================
PLATFORM_VERSION_CODENAME=REL
PLATFORM_VERSION=11
TARGET_PRODUCT=rk3568_r
TARGET_BUILD_VARIANT=userdebug
TARGET_BUILD_TYPE=release
TARGET_ARCH=arm64
TARGET_ARCH_VARIANT=armv8-a
TARGET_CPU_VARIANT=cortex-a55
TARGET_2ND_ARCH=arm
TARGET_2ND_ARCH_VARIANT=armv8-2a
TARGET_2ND_CPU_VARIANT=cortex-a55
HOST_ARCH=x86_64
HOST_2ND_ARCH=x86
HOST_OS=linux
HOST_OS_EXTRA=Linux-5.15.0-53-generic-x86_64-Ubuntu-20.04.2-LTS
HOST_CROSS_OS=windows
HOST_CROSS_ARCH=x86
HOST_CROSS_2ND_ARCH=x86_64
HOST_BUILD_TYPE=release
BUILD_ID=RD2A.211001.002
OUT_DIR=out
============================================
09:07:10 Disallowed PATH tool "python3.8" used: []string{"python3.8", ".repo/repo/repo", "manifest", "-r", "-o", "
09:07:10 See https://android.googlesource.com/platform/build/+/master/Changes.md#PATH_Tools for more information.
[100% 1/1] build out/target/product/rk3568_r/recovery.img

#### build completed successfully (4 seconds) ####
```

Figure 6.3.3.23 Compile recovery.img

The path where the recovery.img image file is located is: out/target/product/rk3568_r/recovery.img.

The recovery.img will be burned into the recovery partition of the development board. recovery.img is also an Android format image. Use the file command to view recovery.img as follows:

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk$
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ file out/target/product/rk3568_r/recovery.img
out/target/product/rk3568_r/recovery.img: Android bootimg, kernel (0x10008000), ramdisk (0x11000000), second stage (0x10f00000
), page size: 2048, cmdline (console=ttyFIQ0 androidboot.baseband=N/A androidboot.wificountrycode=CN androidboot.veritymode=)
tgg@tgg-virtual-machine:~/rk3568_android_sdk$
```

Figure 6.3.3.24 The "file" command is used to view the "recovery.img".

It is formed by merging multiple images. For the RK3568 platform, the "recovery.img" contains five parts: the kernel image "Image", the kernel DTB, the ramdisk (for recovery, the ramdisk in the recovery mode), the "dtbo.img", and the resource image "resource.img". The "unpack_bootimg" tool can be used to unpack it as follows:

```
rm -rfd temp_dir
mkdir temp_dir
cd temp_dir/
cp ../recovery.img ./
unpack_bootimg --boot_img ./recovery.img
ls out/
```

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r$ pwd
/home/tgg/rk3568_android_sdk/out/target/product/rk3568_r
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r$ rm -rfd temp_dir
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r$ mkdir temp_dir
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r$ cd temp_dir/
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$ cp ../recovery.img ./
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$ ls
recovery.img
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$ unpack_bootimg --boot_img ./recovery.img
boot_magic: ANDROID!                                                    Unpacking recovery.img
kernel_size: 32028680
kernel load address: 0x10008000
ramdisk size: 36503449
ramdisk load address: 0x11000000
second bootloader size: 837120
second bootloader load address: 0x10f00000
kernel tags load address: 0x10000100
page size: 2048
os version: 11.0.0
os patch level: 2021-10
boot image header version: 2
product name:
command line args: console=ttyFIQ0 androidboot.baseband=N/A androidboot.wificountrycode=CN androidboot.veritymode=enforcing a
bootstate=orange firmware_class.path=/vendor/etc/firmware init=/init rootwait ro loop.max_part=7 buildvariant=userdebug
additional command line args:
recovery dtbo size: 623
recovery dtbo offset: 0x4229000
boot header size: 1660
dtb size: 142226
dtb address: 0x11f00000
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$ ls
out  recovery.img
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$ ls out/
dtb  kernel  ramdisk  recovery_dtbo  second   The unpacked image file
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$
```

Figure 6.3.3.25 Unpacking recovery.img

The specific situation is as shown in the table below:

Mirror image	Explanation
dtb	The kernel device tree image used in the Recovery mode, but this dtb is just a decoration and will not take effect in the system. The actual effective one is the dtb contained in the resource.img. This dtb is actually the file out/target/product/rk3568_r/dtb.img
Kernel	The kernel image is actually the file out/target/product/rk3568_r/kernel

ramdisk	The ramdisk image used in the Recovery mode is actually the file out/target/product/rk3568_r/ramdisk-recovery.img. ramdisk-recovery.img is the ramdisk in the Recovery mode. As the root file system in the Recovery mode, after the system starts (starts into the Recovery mode), it will mount ramdisk-recovery.img as the root file system (rootfs).
second	For the RK platform, this is actually the resource image resource.img (<SDK>/kernel/resource.img), which contains the kernel DTB, logo images, charging images.
recovery-dtbo	The dtbo image is actually the file out/target/product/rk3568_r/rebuild-dtbo.img

Table 6.3.3.7 Detailed Explanation of Components of recovery.img

Before compiling the recovery.img image, it is necessary to first compile the Linux Kernel, obtaining the kernel image, kernel DTB, and resource.img. When compiling the recovery.img, the kernel image (kernel/arch/arm64/boot/Image) will be copied to out/target/product/rk3568_r/kernel, and the kernel DTB (for example: kernel/arch/arm64/boot/dts/rockchip/rk3568-atk-evb1-ddr4-v10.dtb) will be copied to out/target/product/rk3568_r/dtb.img. resource.img will not be copied over; instead, it will directly be searched for in the kernel/ directory.

U-Boot will determine whether to boot the system into the normal startup mode (Normal mode) or the recovery mode based on the BCB data block stored in the misc partition. If it is determined to enter the recovery mode, U-Boot will find the kernel DTB from the resource.img contained in recovery.img (which will be burned to the recovery partition) and load it into memory (the kernel DTB is packed into resource.img, and resource.img is then packed into recovery.img), and load the dtbo.img contained in recovery.img into memory for overlay operation (device tree overlay). Then it will load and boot the kernel image contained in recovery.img, and finally enter the system and mount the ramdisk contained in recovery.img.

Therefore, recovery.img is prepared for the system to enter the recovery mode.

Use the "file" command to view the "ramdisk-recovery.img", as shown below:

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r$ pwd
/home/tgg/rk3568_android_sdk/out/target/product/rk3568_r
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r$ file ramdisk-recovery.img
ramdisk-recovery.img: gzip compressed data, from Unix, original size modulo 2^32 76981248
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r$
```

From this, it can be known that ramdisk-recovery.img is a gzip-compressed file. It can be decompressed using gunzip, as shown below:

```
rm -rfd temp_dir
mkdir temp_dir
cd temp_dir/
cp ../ramdisk-recovery.img ./
mv ramdisk-recovery.img ramdisk-recovery.img.gz
gunzip ramdisk-recovery.img.gz
file ramdisk-recovery.img
```

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r$ rm -rfd temp_dir
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r$ mkdir temp_dir
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r$ cd temp_dir/
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$ cp ../ramdisk-recovery.img ./
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$ ls
ramdisk-recovery.img
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$ mv ramdisk-recovery.img ramdisk-recovery.img.gz
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$ ls
ramdisk-recovery.img.gz
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$ gunzip ramdisk-recovery.img.gz
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$ ls
ramdisk-recovery.img
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$ file ramdisk-recovery.img
ramdisk-recovery.img: ASCII cpio archive (SVR4 with no CRC)
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$
```

After being decompressed using the gunzip tool, it becomes a cpio archive file. The contents inside can be obtained by using cpio for decompression:

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$ ls
ramdisk-recovery.img
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$ mkdir root
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$ cd root/
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir/root$ cpio -i -F ../ramdisk-recovery.img
150354 块
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir/root$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir/root$ ls
acct          debug_ramdisk              odm                     product_file_contexts      system
apex          default.prop               odm_file_contexts       product_property_contexts  system_ext
bin           dev                        odm_property_contexts   prop.default               system_ext_file_contexts
bugreports    etc                        oem                     res                        system_ext_property_contexts
cache         init                       pcba                    sbin                       tmp
config        init.recovery.rk30board.rc plat_file_contexts      sdcard                     vendor
d             linkerconfig               plat_property_contexts  sepolicy                   vendor_file_contexts
data          metadata                   proc                    storage                    vendor_property_contexts
data_mirror   mnt                        product                 sys
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir/root$
```

The method of re-packaging the contents of the root directory into the ramdisk-recovery.img file:

```
rm -rf ../ramdisk-recovery.img
find . | cpio -o -H newc > ../ramdisk_tmp.img
cd ..
gzip -c ramdisk_tmp.img > ramdisk-recovery.img
file ramdisk-recovery.img
```

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir/root$ rm -rf ../ramdisk-recovery.img
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir/root$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir/root$ find . | cpio -o -H newc > ../ramdisk_tmp.img
150355 块
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir/root$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir/root$ cd ..
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$ ls
ramdisk_tmp.img  root
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$ gzip -c ramdisk_tmp.img > ramdisk-recovery.img
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$ ls
ramdisk-recovery.img  ramdisk_tmp.img  root
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$ file ramdisk-recovery.img
ramdisk-recovery.img: gzip compressed data, was "ramdisk_tmp.img", last modified: Fri May 12 02:38:28 2023, from Unix, original size modulo 2^32
 76981760
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$
```

The recovery.img is ultimately created by the mkbootimg tool (located at out/host/linux-x86/bin/mkbootimg).

To learn about the process of creating the recovery.img image, you can start from line 2327 of build/make/core/Makefile!

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/build/make$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/build/make$ grep -nR "recoveryimage:"
core/Makefile:2327:recoveryimage: $(INSTALLED_RECOVERYIMAGE_TARGET) $(RECOVERY_RESOURCE_ZIP)
tgg@tgg-virtual-machine:~/rk3568_android_sdk/build/make$
```

Key variables: BUILDING_RECOVERY_IMAGE, PRODUCT_BUILD_RECOVERY_IMAGE, BOARD_USES_RECOVERY_AS_BOOT, BOARD_RECOVERYIMAGE_PARTITION_SIZE

10. Compile cache.img

The cache.img file will be burned onto the cache partition of the development board. After the system starts up, it will be mounted to the /cache directory of the Android root file system. The cache partition is a buffer partition. It will be used during the system upgrade process. The upgrade package will be downloaded to this partition, and it is mainly used to buffer the system upgrade OTA package. Double cleaning refers to the cleaning of the data partition and the cache partition.

There is a "recovery" folder in this directory, as shown below:

/cache/recovery/command
/cache/recovery/log
/cache/recovery/intent

Regarding communication with recovery, no further details will be provided!

It should be noted that the RK platform does not use cache.img by default (this image is not mandatory). Although cache.img was generated after compiling the Android source code, the RK did not burn it into the cache partition of the development board.

Execute the following command to compile the cache.img image (please execute it in the root directory of the Android source code):

```
make cacheimage -j16
```

Or compile cache.img using the "m" command (it can be executed in any subdirectory of the Android source code):

```
m cacheimage -j16
```

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ make cacheimage -j16
============================================
PLATFORM_VERSION_CODENAME=REL
PLATFORM_VERSION=11
TARGET_PRODUCT=rk3568_r
TARGET_BUILD_VARIANT=userdebug
TARGET_BUILD_TYPE=release
TARGET_ARCH=arm64
TARGET_ARCH_VARIANT=armv8-a
TARGET_CPU_VARIANT=cortex-a55
TARGET_2ND_ARCH=arm
TARGET_2ND_ARCH_VARIANT=armv8-2a
TARGET_2ND_CPU_VARIANT=cortex-a55
HOST_ARCH=x86_64
HOST_2ND_ARCH=x86
HOST_OS=linux
HOST_OS_EXTRA=Linux-5.15.0-53-generic-x86_64-Ubuntu-20.04.2-LTS
HOST_CROSS_OS=windows
HOST_CROSS_ARCH=x86
HOST_CROSS_2ND_ARCH=x86_64
HOST_BUILD_TYPE=release
BUILD_ID=RD2A.211001.002
OUT_DIR=out
============================================
10:15:07 Disallowed PATH tool "python3.8" used: []string{"python3.8", ".repo/repo/repo", "manifest", "-r", "-o", '
10:15:07 See https://android.googlesource.com/platform/build/+/master/Changes.md#PATH_Tools for more information.
[100% 1/1] Target cache fs image: out/target/product/rk3568_r/cache.img

#### build completed successfully (3 seconds) ####
```

Figure 6.3.3.26 Compile cache.img

The path where the cache.img image file is located is: out/target/product/rk3568_r/cache.img.

The cache.img was created from the out/target/product/rk3568_r/cache folder. The content of cache.img is the contents of the cache directory.

The cache.img will be burned onto the cache partition of the development board (if it needs to be used), and the cache.img is also in the sparse image format. Use the file command to view as follows:

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk$
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ file out/target/product/rk3568_r/cache.img
out/target/product/rk3568_r/cache.img: Android sparse image, version: 1.0, Total of 98304 4096-byte output blocks in 10 input chunks.
tgg@tgg-virtual-machine:~/rk3568_android_sdk$
```

Figure 6.3.3.27 The "file" command can be used to view the "cache.img".

It can be converted to the original format using the "simg2img" command. The original format is "ext4" and the ext4 image can be directly mounted using the "mount" command to view its contents.

To learn about the process of creating the cache.img image, you can start from line 1407 of build/make/core/main.mk!

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/build/make$ grep -nR "cacheimage:"
core/main.mk:1407:cacheimage: $(INSTALLED_CACHEIMAGE_TARGET)
tgg@tgg-virtual-machine:~/rk3568_android_sdk/build/make$
```

Key Variables: BUILDING_CACHE_IMAGE, PRODUCT_BUILD_CACHE_IMAGE, BOARD_CACHEIMAGE_FILE_SYSTEM_TYPE, INSTALLED_CACHEIMAGE_TARGET

11. Compile userdata.img

userdata.img will be burned onto the userdata partition of the development board. userdata.img can contain some user and application-related data, but it is not mandatory; after system startup, userdata.img will be mounted to the Android root file system's /data directory.

The userdata partition is the user storage space of an Android device. Generally, the partition of a newly purchased Android phone is almost empty, and the installed apps and user data are placed in the userdata partition (Android root file system's /data directory).

The RK platform does not use userdata.img by default; by default, compiling the RK3568 Android source code will not generate userdata.img because the rk3568_r product configuration does not enable the compilation of userdata.img.

12. Compile vbmeta.img

The vbmeta.img image is used for AVB verification. AVB (Android Verified Boot) is a security feature introduced in Android 4.4. Its function is to verify the image during the Android system startup process to achieve secure boot. AVB will check whether the image has been tampered with or modified. If the verification fails, the system cannot start normally, and there will be relevant text and graphic prompts.

AVB is a very important security feature of Android. It mainly prevents the image from being tampered with and enhances the overall anti-attack capability of the system.

During the entire AVB verification process, vbmeta.img and vbmeta partition are needed. vbmeta.img will be burned into the vbmeta partition of the development board. The size of the vbmeta.img image compiled is 4K, which stores the verification data, signatures, and other information of various images (such as boot.img, recovery.img, dtbo.img, system.img, vendor.img, odm.img, product.img, system_ext.img).

Executing the following command will compile the vbmeta.img image (please execute it in the root directory of the Android source code):

```
make vbmetaimage -j16
```

Or compile vbmeta.img using the "m" command (this can be executed in any subdirectory of the Android source code):

```
m vbmetaimage -j16
```

After the above command is executed successfully, a vbmeta.img image will be generated in the directory out/target/product/rk3568_r/.

Since the AVB function is not enabled by default on the RK3568 platform, by default, executing the command "make vbmetaimage -j16" or "m vbmetaimage -j16" will not generate the vbmeta.img image in out/target/product/rk3568_r/vbmeta.img.

The AVB function can be enabled by setting the BOARD_AVB_ENABLE variable (which is defined in the BoardConfig.mk file of the board-level configuration). Just set this variable to true, and it will be default false. Therefore, the Android system on the RK3568 platform is not enabled with AVB by default.

The compilation can be done using the compilation script provided by RK <SDK>/build.sh. After the compilation is completed, the image will be packaged and placed in the directory rockdev/Image-rk3568_r/. You can see that there is vbmeta.img in this directory. Note that this vbmeta.img is not obtained by compiling the Android source code, but is directly copied from device/rockchip/common/vbmeta.img (pre-compiled, provided by RK). Open the <SDK>/mkimage.sh script as follows:

```
echo "create vbmeta.img..."
if [ "$BOARD_AVB_ENABLE" = "true" ]; then
cp -a $OUT/vbmeta.img $IMAGE_PATH/vbmeta.img
else
echo "BOARD_AVB_ENABLE is false, use default vbmeta.img"
cp -a device/rockchip/common/vbmeta.img $IMAGE_PATH/vbmeta.img
fi
```

According to the mkimage.sh script, if the AVB function is enabled, the vbmeta.img (obtained by compiling the Android source code) in the product output directory will be copied to the rockdev/Image-rk3568_r/ directory; if the AVB function is not enabled, after compiling the Android source code, vbmeta.img will not be generated in the product output directory; instead, it will directly copy device/rockchip/common/vbmeta.img to the rockdev/Image-rk3568_r/ directory.

device/rockchip/common/vbmeta.img is the default vbmeta.img image provided by RK. This image is actually an empty image and does not contain any valid verification data; if AVB is enabled, during the Android system startup process, AVB verification will be performed, and the AVB verification process requires the assistance of vbmeta.img (containing verification data, signatures, etc.) and the vbmeta partition.

AVB requires the cooperation of BootLoader (for RK platform, U-Boot is used). The AVB verification is divided into two stages: the BootLoader stage and the init stage. The contents of verification in these two stages are different.

① , BootLoader stage

In the BootLoader stage, the boot.img, dtbo.img, and recovery.img (if entering the recovery mode, then the recovery.img will be verified) will undergo secure verification (performed according to AVB rules). If the verification fails, the next-level firmware will not be started and the system will reset.

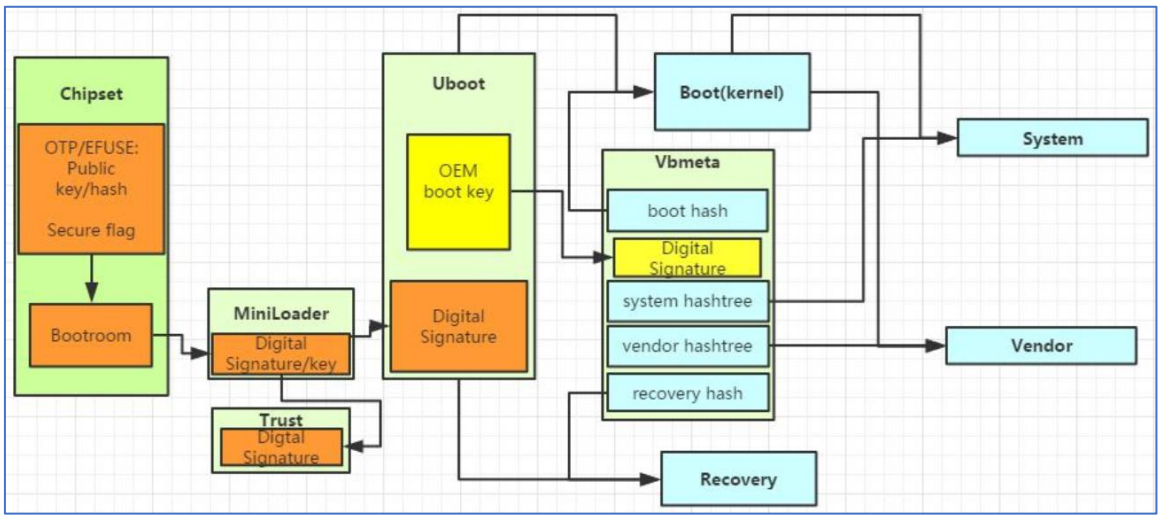
②, init stage

Entering the kernel, the kernel will run the init process. In the init stage, the system.img, vendor.img, product.img, odm.img, and system_ext.img images will undergo secure verification (executing DM-Verity verification). If the verification is successful, the system will start normally and enter the Android system; otherwise, the system will not start normally and there will be corresponding text and graphic prompts.

From the above introduction, it can be seen that AVB does not include the verification of all images, but only the verification of boot.img, dtbo.img, recovery.img, and the subsequent images (system.img, product.img, odm.img, etc.). It does not verify u-boot.img or MiniLoaderAll.bin. If a truly secure boot is to be achieved, the boot process must also include the secure verification of u-boot.img and MiniLoaderAll.bin images.

Because Google has no standard for the security verification of U-Boot and the loader images running before U-Boot (such as MiniLoaderAll.bin on the RK platform), each chip manufacturer can customize it. Therefore, RK has implemented a verification scheme for u-boot.img and MiniLoaderAll.bin images by itself (based on the hardware protection mechanism provided by the RK chip).

Therefore, the secure boot on the RK Android 11 platform consists of two main parts: the secure boot of the bootloader (including uboot.img and MiniLoaderAll.bin) and the AVB (dtbo.img, boot.img/recovery.img, and the subsequent image system.img, etc.). After the secure boot is enabled, the BootRom code, as read-only code BL1, is trusted and does not require signature verification. BL1 will verify MiniLoaderAll.bin, and MiniLoaderAll.bin will verify uboot.img. Subsequently, a step-by-step verification method is adopted to verify whether each loaded image has been tampered with! The basic flowchart is as follows:



To learn about the process of creating the vbmeta.img image, you can start by referring to line 1446 of the build/make/core/main.mk file!

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/build/make$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/build/make$ grep -nR "vbmetaimage:"
core/main.mk:1446:vbmetaimage: $(INSTALLED_VBMETAIMAGE_TARGET)
tgg@tgg-virtual-machine:~/rk3568_android_sdk/build/make$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/build/make$
```

Key variables: BOARD_AVB_ENABLE, BUILT_VBMETAIMAGE_TARGET, BUILDING_VBMETA_IMAGE, INSTALLED_VBMETAIMAGE_TARGET, PRODUCT_BUILD_VBMETA_IMAGE

13. Compile super.img

Since the Android 10 version, Android system has introduced dynamic partition, which combines multiple system read-only partitions (including system, product, vendor, odm, system_ext or other custom partitions of the manufacturer) into one super partition. As shown in the figure below:

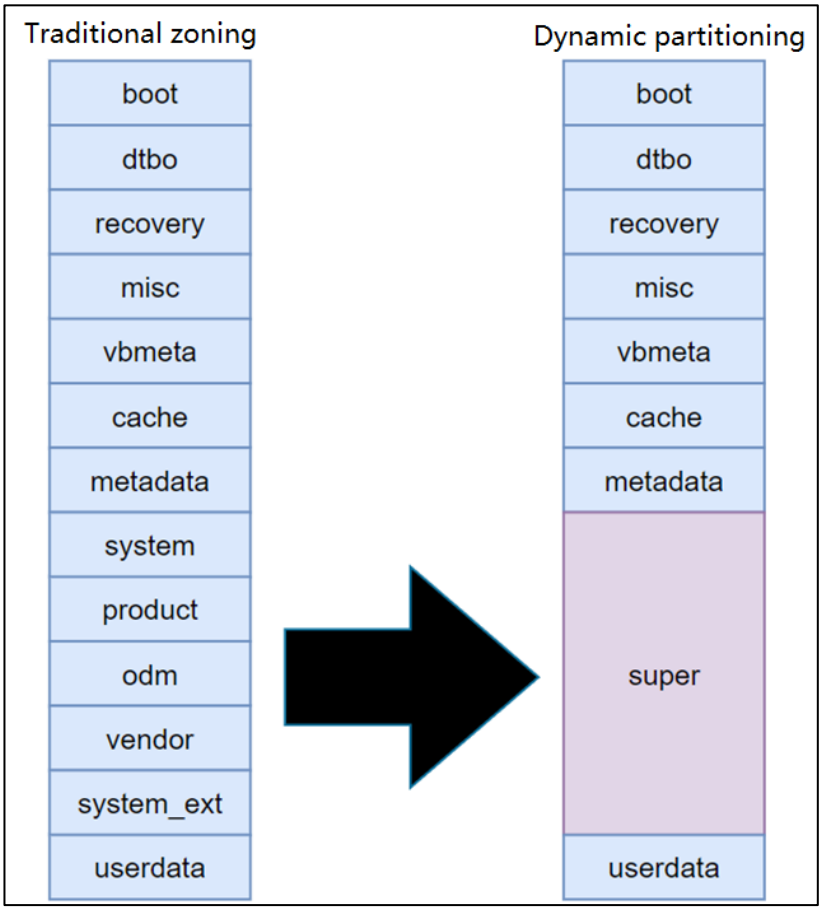


Figure 6.3.3.28 Dynamic Partition Principle

After introducing dynamic partitions, the physical partitions only have the concept of the super partition, but no partitions such as system, product, vendor, odm, etc. During the creation of the image, these images (system.img, vendor.img, odm.img, product.img, system_ext.img) will be merged into a super.img image, and finally, the super.img will be burned onto the super partition of the development board.

Since dynamic partitions are implemented in the user space, the partitions required by the boot loader cannot be dynamic. For example, the boot loader will read boot, dtbo, and vbmeta. Therefore, these partitions must still remain as physical partitions.

With dynamic partitions, suppliers do not need to worry about the sizes of each partition (such as system, product, vendor, odm, system_ext, etc.). Although they have been combined into a super partition, the various sub-partitions within the super partition can be dynamically adjusted in size. Of

course, it is necessary to ensure that the total size of each sub-partition does not exceed the size of the super partition.

To enable dynamic partitions, the following variables need to be defined in the product Makefile configuration file:

```
PRODUCT_USE_DYNAMIC_PARTITIONS := true
```

The RK3568 platform has already enabled dynamic partitioning by default. This variable is defined in the file device/rockchip/common/build/rockchip/DynamicPartitions.mk, as shown below:

```
# Enable dynamic partitions, Android Q must set this by default.
# Use the non-open-source parts, if they're present
# Android Q -> api_level 29, Pie or earlier should not include this makefile

PRODUCT_USE_DYNAMIC_PARTITIONS := true

# Add standalone odm partition configrations
PRODUCT_BUILD_PRODUCT_IMAGE := true
TARGET_COPY_OUT_PRODUCT := product
BOARD_PRODUCTIMAGE_FILE_SYSTEM_TYPE ?= ext4

```

Figure 6.3.3.29 Define the PRODUCT_USE_DYNAMIC_PARTITIONS variable

The DynamicPartitions.mk file will eventually be included in the device/rockchip/rk356x/rk3568_r/rk3568_r.mk file.

Execute the following command to compile the super.img image (please execute it in the root directory of the Android source code):

```
make superimage -j16
```

Or compile super.img using the "m" command (which can be executed in any subdirectory of the Android source code):

```
m superimage -j16
```

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk$
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ m superimage -j16
============================================
PLATFORM_VERSION_CODENAME=REL
PLATFORM_VERSION=11
TARGET_PRODUCT=rk3568_r
TARGET_BUILD_VARIANT=userdebug
TARGET_BUILD_TYPE=release
TARGET_ARCH=arm64
TARGET_ARCH_VARIANT=armv8-a
TARGET_CPU_VARIANT=cortex-a55
TARGET_2ND_ARCH=arm
TARGET_2ND_ARCH_VARIANT=armv8-2a
TARGET_2ND_CPU_VARIANT=cortex-a55
HOST_ARCH=x86_64
HOST_2ND_ARCH=x86
HOST_OS=linux
HOST_OS_EXTRA=Linux-5.15.0-53-generic-x86_64-Ubuntu-20.04.2-LTS
HOST_CROSS_OS=windows
HOST_CROSS_ARCH=x86
HOST_CROSS_2ND_ARCH=x86_64
HOST_BUILD_TYPE=release
BUILD_ID=RD2A.211001.002
OUT_DIR=out
============================================
17:01:19 Disallowed PATH tool "python3.8" used: []string{"python3.8", ".repo/repo/repo", "manifest", "-r", "-o", "out/com
17:01:19 See https://android.googlesource.com/platform/build/+/master/Changes.md#PATH_Tools for more information.
[100% 76/76] Target super fs image for debug: out/target/product/rk3568_r/super.img
2023-05-12 17:01:29 - build_super_image.py - INFO    : Building super image from info dict...
2023-05-12 17:01:29 - sparse_img.py - INFO    : Total of 239419 4096-byte output blocks in 18 input chunks.
2023-05-12 17:01:29 - sparse_img.py - INFO    : Total of 30192 4096-byte output blocks in 3 input chunks.
2023-05-12 17:01:29 - sparse_img.py - INFO    : Total of 75087 4096-byte output blocks in 8 input chunks.
2023-05-12 17:01:29 - sparse_img.py - INFO    : Total of 52087 4096-byte output blocks in 5 input chunks.
2023-05-12 17:01:29 - sparse_img.py - INFO    : Total of 153 4096-byte output blocks in 2 input chunks.
2023-05-12 17:01:29 - common.py - INFO    :   Running: "lpmake --metadata-size 65536 --super-name super --metadata-slots 
tion system:readonly:980660224:rockchip_dynamic_partitions --image system=out/target/product/rk3568_r/system.img --partit
ut/target/product/rk3568_r/system_ext.img --partition vendor:readonly:307556352:rockchip_dynamic_partitions --image vendo
ckchip_dynamic_partitions --image product=out/target/product/rk3568_r/product.img --partition odm:readonly:626688:rockchi
tput out/target/product/rk3568_r/super.img"
2023-05-12 17:01:45 - common.py - INFO    : lpmake I 05-12 17:01:29   883   883 builder.cpp:1031] [liblp]Partition system
lpmake I 05-12 17:01:29   883   883 builder.cpp:1031] [liblp]Partition system_ext will resize from 0 bytes to 123666432 b
lpmake I 05-12 17:01:29   883   883 builder.cpp:1031] [liblp]Partition vendor will resize from 0 bytes to 307556352 bytes
lpmake I 05-12 17:01:29   883   883 builder.cpp:1031] [liblp]Partition product will resize from 0 bytes to 213348352 byte
lpmake I 05-12 17:01:29   883   883 builder.cpp:1031] [liblp]Partition odm will resize from 0 bytes to 626688 bytes
2023-05-12 17:01:45 - build_super_image.py - INFO    : Done writing image out/target/product/rk3568_r/super.img

#### build completed successfully (30 seconds) ####
```

Figure 6.3.3.30 Compile super.img

The path where the super.img image file is located is: out/target/product/rk3568_r/super.img.

The super.img will eventually be burned into the super partition of the development board. The super.img contains system.img, vendor.img, product.img, odm.img, and system_ext.img.

Use the file command to view the super.img, as follows:

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk$
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ file out/target/product/rk3568_r/super.img
out/target/product/rk3568_r/super.img: Android sparse image, version: 1.0, Total of 796672 4096-byte output blocks in 3478 input chunks
tgg@tgg-virtual-machine:~/rk3568_android_sdk$
```

Figure 6.3.3.31 The "file" command is used to view the contents of "super.img".

From this, it can be seen that super.img is also a sparse image format image. Next, we will unpack the super.img image. First, we will use the simg2img tool for conversion:

```
rm -rfd temp_dir
mkdir temp_dir
cd temp_dir/
cp ../super.img ./
simg2img super.img super_raw.img
file super_raw.img
```

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r$ pwd
/home/tgg/rk3568_android_sdk/out/target/product/rk3568_r
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r$ rm -rfd temp_dir
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r$ mkdir temp_dir
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r$ cd temp_dir/
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$ cp ../super.img ./
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$ ls
super.img
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$ simg2img super.img super_raw.img
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$ ls
super.img  super_raw.img
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$ file super_raw.img
super_raw.img: data
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$
```

Figure 6.3.3.32 Use simg2img to convert the super.img image

Next, we need to unpack it using the lpunpack tool, as shown below:

```
lpunpack super_raw.img
```

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$ ls
super.img  super_raw.img
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$ lpunpack super_raw.img
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$ ls
odm.img  product.img  super.img  super_raw.img  system_ext.img  system.img  vendor.img
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$
```

Figure 6.3.3.33 Use the lpunpack tool to unpack super_raw.img

Note: If the simg2img tool is not available, it needs to be compiled through the "make lpunpack -j16" command.

After successful unpacking, five img images are obtained: odm.img, product.img, system_ext.img, system.img, and vendor.img. Use the file command to view these img images, as shown below:

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$ file *.img
odm.img:        Linux rev 1.0 ext2 filesystem data, UUID=80c1ea02-172f-4bba-ae2a-1c30f74507a3, volume name "odm" (extents) (large files) (huge files)
product.img:    Linux rev 1.0 ext2 filesystem data, UUID=37e9f6c1-286b-400f-88db-a3750970d119, volume name "product" (extents) (large files) (huge files)
super.img:      Android sparse image, version: 1.0, Total of 796672 4096-byte output blocks in 3478 input chunks.
super_raw.img:  data
system_ext.img: Linux rev 1.0 ext2 filesystem data, UUID=6c55339c-00e3-4a66-a06c-b68f5ccc584a, volume name "system_ext" (extents) (large files) (huge files)
system.img:     Linux rev 1.0 ext2 filesystem data, UUID=0626a3f8-96f3-4415-8b71-35c66d1ad78f (extents) (large files) (huge files)
vendor.img:     Linux rev 1.0 ext2 filesystem data, UUID=35b852dd-ea91-4e24-9f79-435bc44bf9d0, volume name "vendor" (extents) (large files) (huge files)
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/temp_dir$
```

Figure 6.3.3.34 The "file" command is used to view the contents of the "img" image.

If all the files are in the ext4 format, then you can directly use the "mount" command to mount and view them. You can also use the "lpmake" tool to repackage them into "super.img".

Use the "get_build_var" command to view the "BOARD_SUPER_PARTITION_SIZE" variable to obtain the current size of the super partition, as shown below:

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/device/rockchip$ get_build_var BOARD_SUPER_PARTITION_SIZE
3263168512
tgg@tgg-virtual-machine:~/rk3568_android_sdk/device/rockchip$
```

Figure 6.3.3.35 Check the size of the super partition

This variable is defined in the product board-level configuration file BoardConfig.mk, as shown below:

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/device/rockchip$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/device/rockchip$ grep -nR BOARD_SUPER_PARTITION_SIZE
common/BoardConfig_AB.mk:30:            BOARD_SUPER_PARTITION_SIZE := $(shell expr $(BOARD_SUPER_PARTITION_SYSTEM_DEVICE_SIZE)
DEVICE_SIZE))
common/BoardConfig_AB.mk:31:            BOARD_ROCKCHIP_DYNAMIC_PARTITIONS_SIZE := $(shell expr $(BOARD_SUPER_PARTITION_SIZE) -
common/BoardConfig_AB.mk:33:            BOARD_SUPER_PARTITION_SIZE := $(shell python device/rockchip/common/get_partition_size
common/BoardConfig_AB.mk:35:                BOARD_ROCKCHIP_DYNAMIC_PARTITIONS_SIZE := $(shell expr $(BOARD_SUPER_PARTITION_SI
common/BoardConfig_AB.mk:37:                BOARD_ROCKCHIP_DYNAMIC_PARTITIONS_SIZE := $(shell expr $(BOARD_SUPER_PARTITION_SI
common/BoardConfig_AB.mk:59:                        BOARD_SUPER_PARTITION_SIZE := 2390753280
common/BoardConfig_AB.mk:61:                        BOARD_SUPER_PARTITION_SIZE := 1971322880
common/BoardConfig_AB.mk:64:                    BOARD_SUPER_PARTITION_SIZE := 3263168512
common/BoardConfig_AB.mk:66:                BOARD_ROCKCHIP_DYNAMIC_PARTITIONS_SIZE := $(shell expr $(BOARD_SUPER_PARTITION_SI
common/BoardConfig_AB.mk:68:                BOARD_SUPER_PARTITION_SIZE := 5372903424
common/BoardConfig_AB.mk:69:                BOARD_ROCKCHIP_DYNAMIC_PARTITIONS_SIZE := $(shell expr $(BOARD_SUPER_PARTITION_SI
common/build/rockchip/RebuildParameter.mk:36:partition_list := $(partition_list),super:$(BOARD_SUPER_PARTITION_SIZE)
common/BoardConfig.mk:149:    BOARD_SUPER_PARTITION_SIZE := $(shell python device/rockchip/common/get_partition_size.py $(TARG
common/BoardConfig.mk:150:    BOARD_ROCKCHIP_DYNAMIC_PARTITIONS_SIZE := $(shell expr $(BOARD_SUPER_PARTITION_SIZE) - 4194304)
common/BoardConfig.mk:168:      BOARD_SUPER_PARTITION_SIZE ?= 2516582400
common/BoardConfig.mk:170:      BOARD_SUPER_PARTITION_SIZE ?= 3263168512   What takes effect is this one.
common/BoardConfig.mk:172:    BOARD_ROCKCHIP_DYNAMIC_PARTITIONS_SIZE ?= $(shell expr $(BOARD_SUPER_PARTITION_SIZE) - 4194304)
tgg@tgg-virtual-machine:~/rk3568_android_sdk/device/rockchip$
```

Users can modify this variable to set the size of the super partition.

The BOARD_ROCKCHIP_DYNAMIC_PARTITIONS_PARTITION_LIST variable can be used to specify the partition list that needs to be merged into the super dynamic partition. On the RK3568 platform, by default, the system, vendor, product, odm, and system_ext partitions will be merged into the super partition, as shown below:

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/device/rockchip$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/device/rockchip$ grep -nR BOARD_ROCKCHIP_DYNAMIC_PARTITIONS_PARTITION_LIST
common/BoardConfig_AB_retrofit.mk:22:BOARD_ROCKCHIP_DYNAMIC_PARTITIONS_PARTITION_LIST := system vendor odm
common/build/rockchip/DynamicPartitions.mk:18:BOARD_ROCKCHIP_DYNAMIC_PARTITIONS_PARTITION_LIST := system system_ext vendor product odm
tgg@tgg-virtual-machine:~/rk3568_android_sdk/device/rockchip$
```

Open the DynamicPartitions.mk file:

```
16 BOARD_BUILD_SUPER_IMAGE_BY_DEFAULT := true
17 BOARD_SUPER_PARTITION_GROUPS := rockchip_dynamic_partitions
18 BOARD_ROCKCHIP_DYNAMIC_PARTITIONS_PARTITION_LIST := system system_ext vendor product odm
~
```

The compilation system will combine the partitions defined by the variable BOARD_$(BOARD_SUPER_PARTITION_GROUPS)_PARTITION_LIST into a single super dynamic partition. After the variable is replaced, it becomes:

BOARD_rockchip_dynamic_partitions_PARTITION_LIST

Converting lowercase letters to uppercase letters results in the third variable:

BOARD_ROCKCHIP_DYNAMIC_PARTITIONS_PARTITION_LIST

To learn about the process of creating the super.img image, you can start by referring to line 5282 of the build/make/core/Makefile!

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/build/make$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/build/make$ grep -nR "superimage:"
core/Makefile:5282:superimage: $(INSTALLED_SUPERIMAGE_TARGET)
tgg@tgg-virtual-machine:~/rk3568_android_sdk/build/make$
```

Key variables: PRODUCT_BUILD_SUPER_PARTITION, BOARD_SUPER_PARTITION_SIZE, PRODUCT_USE_DYNAMIC_PARTITIONS, BOARD_SUPER_PARTITION_GROUPS, BOARD_ROCKCHIP_DYNAMIC_PARTITIONS_PARTITION_LIST

14. Compile dtbo.img

Starting from Android 9, DTO (Device Tree Overlay) was introduced. DTO refers to Device Tree Overlay. The concept of device tree overlay means covering one device tree (referred to as the overlay DT) on top of another device tree (the main DT), thereby allowing the addition of attributes/nodes to the main DT or the modification of attributes in the main DT. An example is as follows:

Superposition

If the application target attribute exists in the main DT, it will be superimposed after the DTO. Otherwise, the system will add it. For example:

main.dts

```
[my_main_dt.dts]

/dts-v1/;

/ {
  compatible = "corp,foo";

  my_node: node@0 {
    status = "disabled";
  };
};
```

overlay.dts

```
[my_overlay_dt.dts]

/dts-v1/;
/plugin/;

&my_node {
  status = "okay";
};
```

合并结果

```
/dts-v1/;

/ {
  compatible = "corp,foo";

  ...

  node@0 {
    linux,phandle = <0x1>;
    phandle = <0x1>;
    status = "okay";
  };
};
```

The "status" attribute has changed from "disable" to "okay".

Additional

If the target attribute of the application does not exist in the main DT, it will be appended after the DTO. For example:

main.dts

```
[my_main_dt.dts]

/dts-v1/;

/ {
  compatible = "corp,foo";

  my_node: node@0 {
    status = "okay";
  };
};
```

overlay.dts

```
[my_overlay_dt.dts]

/dts-v1/;
/plugin/;

&my_node {
  new_prop = "bar";
};
```

合并结果

```
/dts-v1/;

/ {
  compatible = "corp,foo";

  ...

  node@0 {
    linux,phandle = <0x1>;
    phandle = <0x1>;
    status = "okay";
    new_prop = "bar";
  };
};
```

Add an attribute "new_prop = 'bar'" to node@0 node.

The device tree overlay operation is completed by the BootLoader (for the RK platform, it is U-Boot). The example diagram of the overlay operation is as follows:

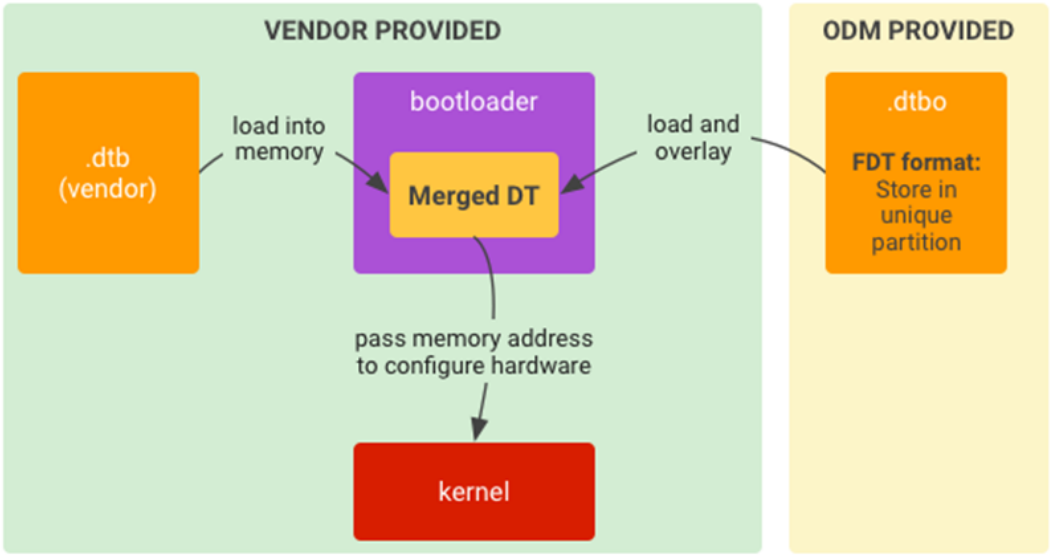


Figure 5. Typical runtime implementation of the device tree overlay layer in the bootloader

1. Load the.dtb from the storage space into the memory.
2. Load.dtbo from the storage space into the memory.
3. Combine .dtb and .dtbo files to form a merged DT.
4. Start the kernel (with the given memory address for the merged DT)

"dtbo.img" refers to the previously mentioned superimposed DT mirror image - dtbo (DTB Overlay). dtbo.img will be burned into the dtbo partition of the development board. For the RK platform, the main DT mirror image is the DTB contained in the resource.img.

The original command for compiling dtbo.img from the Android source code is as follows (it needs to be executed in the root directory of the Android source code):

```
make dtboimage -j16
```

Or compile dtbo.img using the "m" command (which can be executed in any subdirectory of the Android source code):

```
m dtboimage -j16
```

However, it should be noted that executing the above command in the RK Android source code will result in an error, indicating that the target (dtboimage) does not exist. The reason is that the original dtbo.img compilation rule in the RK Android source code has been modified. Instead of using the "make dtboimage" or "m dtboimage" commands for compilation, it is now compiled using the following command:

```
make out/target/product/rk3568_r/rebuild-dtbo.img -j16
```

Or, you can compile rebuild-dtbo.img using the m command (this can be done in any subdirectory of the Android source code):

```
m out/target/product/rk3568_r/rebuild-dtbo.img -j16
```

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk$
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ m out/target/product/rk3568_r/rebuild-dtbo.img -j16
============================================
PLATFORM_VERSION_CODENAME=REL
PLATFORM_VERSION=11
TARGET_PRODUCT=rk3568_r
TARGET_BUILD_VARIANT=userdebug
TARGET_BUILD_TYPE=release
TARGET_ARCH=arm64
TARGET_ARCH_VARIANT=armv8-a
TARGET_CPU_VARIANT=cortex-a55
TARGET_2ND_ARCH=arm
TARGET_2ND_ARCH_VARIANT=armv8-2a
TARGET_2ND_CPU_VARIANT=cortex-a55
HOST_ARCH=x86_64
HOST_2ND_ARCH=x86
HOST_OS=linux
HOST_OS_EXTRA=Linux-5.15.0-53-generic-x86_64-Ubuntu-20.04.2-LTS
HOST_CROSS_OS=windows
HOST_CROSS_ARCH=x86
HOST_CROSS_2ND_ARCH=x86_64
HOST_BUILD_TYPE=release
BUILD_ID=RD2A.211001.002
OUT_DIR=out
============================================
09:02:20 Disallowed PATH tool "python3.8" used: []string{"python3.8", ".repo/repo/repo", "manifest", "-r", "-o", "
09:02:20 See https://android.googlesource.com/platform/build/+/master/Changes.md#PATH_Tools for more information.
[100% 1/1] build out/target/product/rk3568_r/rebuild-dtbo.img

#### build completed successfully (3 seconds) ####
```

Figure 6.3.3.36 Compile rebuild-dtbo.img

The path of the image is: out/target/product/rk3568_r/rebuild-dtbo.img (rename it to dtbo.img).

The dtbo.img will eventually be burned onto the dtbo partition of the development board (for Normal boot mode). Besides, the recovery.img image also contains a copy of the dtbo.img image for the recovery mode.

The compilation rules for the rebuild-dtbo.img image are implemented by the Makefile provided by RK. The specific file path is: device/rockchip/common/build/rockchip/RebuildDtboImg.mk:

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/device/rockchip$ grep -nR rebuild-dtbo.img
common/mkimage.sh:88:BOARD_DTBO_IMG=$OUT/rebuild-dtbo.img
common/build/rockchip/RebuildDtboImg.mk:12:rebuild_dtbo_img := $(intermediates)/rebuild-dtbo.img
common/mkimage_ab.sh:98:BOARD_DTBO_IMG=$OUT/rebuild-dtbo.img
tgg@tgg-virtual-machine:~/rk3568_android_sdk/device/rockchip$
```

Open this file:

```
1 ifdef PRODUCT_DTBO_TEMPLATE
2
3 $(info rebuilding dtbo image with $(PRODUCT_DTBO_TEMPLATE)....)
4 intermediates := $(call intermediates-dir-for,FAKE,rockchip_dtbo)
5
6 AOSP_DTC_TOOL := $(SOONG_HOST_OUT_EXECUTABLES)/dtc
7 AOSP_MKDTIMG_TOOL := $(SOONG_HOST_OUT_EXECUTABLES)/mkdtimg
8 ROCKCHIP_FSTAB_TOOLS := $(SOONG_HOST_OUT_EXECUTABLES)/fstab_tools
9
10 rebuild_dts := $(intermediates)/device-tree-overlay.dts
11 rebuild_dtbo_dtb := $(intermediates)/device-dtbo.dtb
12 rebuild_dtbo_img := $(intermediates)/rebuild-dtbo.img
13
14 dtbo_flags := wait
15 ifeq ($(strip $(BOARD_USES_AB_IMAGE)), true)
16     dtbo_flags := $(dtbo_flags),slotselect
17 endif # BOARD_USES_AB_IMAGE
18
19 ifeq ($(strip $(BOARD_AVB_ENABLE)), true)
20     dtbo_flags := $(dtbo_flags),avb
21 endif # BOARD_AVB_ENABLE
22
23 dtbo_boot_device := none
24 ifdef PRODUCT_BOOT_DEVICE
25     dtbo_boot_device := $(PRODUCT_BOOT_DEVICE)
26 endif
27 $(rebuild_dts) : $(ROCKCHIP_FSTAB_TOOLS) $(PRODUCT_DTBO_TEMPLATE)
28         @echo "Building dts file $@."
29         $(ROCKCHIP_FSTAB_TOOLS) -I dts \
30         -i $(PRODUCT_DTBO_TEMPLATE) \
31         -p $(dtbo_boot_device) \
32         -f $(dtbo_flags) \
33         -o $(rebuild_dts)
34
35 $(rebuild_dtbo_img) : $(rebuild_dts) $(AOSP_DTC_TOOL) $(AOSP_MKDTIMG_TOOL)
36         @echo "Building dtbo img file $@."
37         $(AOSP_DTC_TOOL) -@ -O dtb -o $(rebuild_dtbo_dtb) $(rebuild_dts)
38         $(AOSP_MKDTIMG_TOOL) create $(rebuild_dtbo_img) $(rebuild_dtbo_dtb)
39
40 INSTALLED_RK_DTBO_IMAGE := $(PRODUCT_OUT)/$(notdir $(rebuild_dtbo_img))
41 $(INSTALLED_RK_DTBO_IMAGE) : $(rebuild_dtbo_img)
42         $(call copy-file-to-new-target-with-cp)
43
44 ALL_DEFAULT_INSTALLED_MODULES += $(INSTALLED_RK_DTBO_IMAGE)
45 BOARD_PREBUILT_DTBOIMAGE := $(INSTALLED_RK_DTBO_IMAGE)
46 endif
```

Obtain the overlay.dts source file through rebuild_dts

Obtain overlay.dtb through rebuild_dtbo_img

The $(rebuild_dts) target will result in the overlay.dts source file (overlay DTS). The generation of the overlay.dts source file depends on the $(PRODUCT_DTBO_TEMPLATE) variable, which is defined in the device/rockchip/rk356x/rk3568_r/rk3568_r.mk file. This variable specifies the following file:

device/rockchip/rk356x/rk3568_r/dt-overlay.in

The input file is "dt-overlay.in" and the output file is "overlay.dts". The content of the "dt-overlay.in" file is as follows:

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

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

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

After obtaining overlay.dts, first use the $(AOSP_DTC_TOOL) tool to compile the source file overlay.dts into overlay.dtb, then use the $(AOSP_MKDTIMG_TOOL) tool for format conversion, and finally obtain overlay.img.

AOSP_DTC_TOOL	---------->	out/soong/host/linux-x86/bin/dtc
AOSP_MKDTIMG_TOOL	---------->	out/soong/host/linux-x86/bin/mkdtimg

15. Compile misc.img

The misc.img image will be burned onto the misc partition of the development board. The misc partition is a very important partition. The misc partition is mainly used for communication between the Android system and the BootLoader. The misc partition contains the Android BCB (BootLoader Control Block) data. BCB is a mechanism designed by Android to interact with the BootLoader and control the system startup process.

U-Boot will read the BCB data block to determine whether to boot the system into the normal startup mode (Android system, Normal mode) or the recovery mode (recovery system). When the device enters the recovery mode, it will read the recovery part of the misc partition, thereby performing different actions, such as upgrading the partition firmware, erasing user data, or other operations, etc.

The misc.img image file for the RK3568 platform does not need to be obtained through compilation. Instead, RK directly provides the img image file. Open the mkimage.sh script located in the root directory of the Android source code:

```
echo -n "create misc.img.... "
cp -a rkst/Image/misc.img $IMAGE_PATH/misc.img
cp -a rkst/Image/pcba_small_misc.img $IMAGE_PATH/pcba_small_misc.img
cp -a rkst/Image/pcba_whole_misc.img $IMAGE_PATH/pcba_whole_misc.img
echo "done."
```

IMAGE_PATH refers to the "rockdev/Image-rk3568_r" directory located at the root of the Android source code. Therefore, through the "mkimage.sh" script, the "misc.img", "pcba_small_misc.img", and "pcba_whole_misc.img" files in the "rkst/Image" directory will be copied to the "rockdev/Image-rk3568_r" directory.

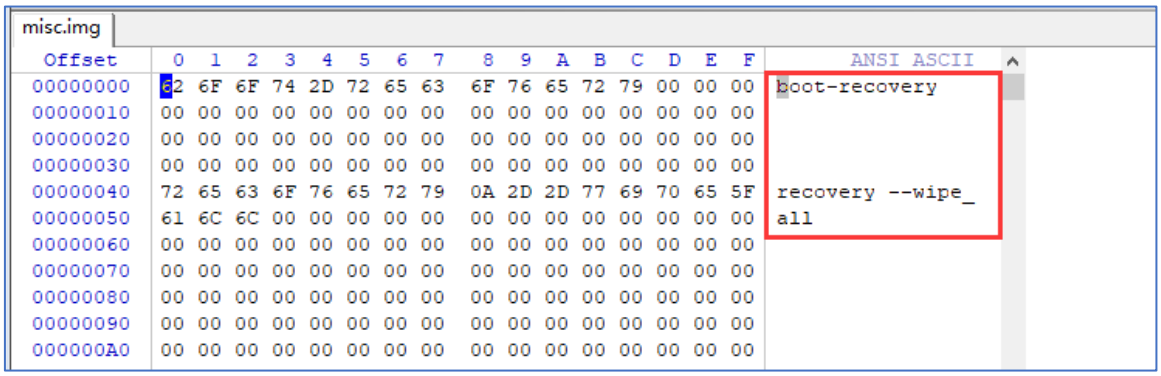
Among these three images, "misc.img" is the most commonly used, while the other two are less frequently used. The difference between them lies in the "recovery" identifier in the BCB data block.

We can use tools such as WinHex or UltraEdit to open the "misc.img" file in binary format, and then we can find the following content (at the 0 offset address):

Compatibility of misc

The purpose of misc.img is to serve as the boot interaction between U-Boot and Android. Its main content is BCB (Boot Loader Control Block).

Due to historical reasons of the RK platform, for SDK versions equal to or greater than Android-10.0, the BCB in misc.img must be stored at offset 0 of the misc partition; for versions lower than Android-10.0, the BCB must be stored at offset 16KB of the misc partition.



The "boot-recovery" identifier indicates entering the recovery mode;

While "recovery --wipe_all" is an additional identifier for the recovery mode, --wipe_all indicates erasing the entire userdata partition. Therefore, the "misc.img" was burned, and after the burn, the system restarted into the recovery mode and executed the command to erase the entire userdata partition. After the erasure, the system restarted again (before the restart, the "boot-recovery" identifier in the BCB data block of the misc partition was erased). Only then would this restart enter the normal Android system.

If you want to restart after burning and directly enter the Android system instead of the recovery mode, you can remove the "boot-recovery" identifier and directly edit it using tools like WinHex or UltraEdit!

Apart from the --wipe_all identifier, many other identifiers are also supported. For specific details, you can refer to the RK U-Boot documentation: RKDocs/common/u-boot/Rockchip_Developer_Guide_UBoot_Nextdev_CN.pdf

16. Generation of parameter.txt

parameter.txt is a partition table file for the RK platform and is a TXT text file. When burning the image, it is not necessary to burn the parameter.txt file to the Flash device (such as eMMC), but its information will be read to define the physical partitions of the memory. The partition names, starting addresses, and sizes of each partition are all defined by the parameter.txt file.

After compiling the RK Android source code, a parameter.txt file will be generated in the product output directory. This file is also dynamically generated during the compilation process and does not exist originally. The following command can be executed to generate the parameter.txt file (it needs to be executed in the root directory of the Android source code):

```
make out/target/product/rk3568_r/parameter.txt -j16
```

Or generate parameter.txt using the m command (which can be executed in any subdirectory of the Android source code):

```
m out/target/product/rk3568_r/parameter.txt -j16
```

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ make out/target/product/rk3568_r/parameter.txt -j16
============================================
PLATFORM_VERSION_CODENAME=REL
PLATFORM_VERSION=11
TARGET_PRODUCT=rk3568_r
TARGET_BUILD_VARIANT=userdebug
TARGET_BUILD_TYPE=release
TARGET_ARCH=arm64
TARGET_ARCH_VARIANT=armv8-a
TARGET_CPU_VARIANT=cortex-a55
TARGET_2ND_ARCH=arm
TARGET_2ND_ARCH_VARIANT=armv8-2a
TARGET_2ND_CPU_VARIANT=cortex-a55
HOST_ARCH=x86_64
HOST_2ND_ARCH=x86
HOST_OS=linux
HOST_OS_EXTRA=Linux-5.15.0-53-generic-x86_64-Ubuntu-20.04.2-LTS
HOST_CROSS_OS=windows
HOST_CROSS_ARCH=x86
HOST_CROSS_2ND_ARCH=x86_64
HOST_BUILD_TYPE=release
BUILD_ID=RD2A.211001.002
OUT_DIR=out
============================================
12:10:17 Disallowed PATH tool "python3.8" used: []string{"python3.8", ".repo/repo/repo", "manifest", "-r", "-o",
12:10:17 See https://android.googlesource.com/platform/build/+/master/Changes.md#PATH_Tools for more information.
[100% 1/1] build out/target/product/rk3568_r/parameter.txt

#### build completed successfully (3 seconds) ####
```

Figure 6.3.3.37 Generate the "parameter.txt" file.

The path of the "parameter.txt" file is: out/target/product/rk3568_r/parameter.txt.

The "parameter.txt" file in rockdev/Image-rk3568_r/ was copied from the product output directory.

The parameter.txt file is dynamically generated by the Makefile provided by RK. The specific file path is: device/rockchip/common/build/rockchip/RebuildParameter.mk, as shown below:

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/device/rockchip$ grep -nR parameter.txt | grep RebuildParameter.mk
common/build/rockchip/RebuildParameter.mk:3:$(info build parameter.txt with $(PRODUCT_PARAMETER_TEMPLATE)....)
common/build/rockchip/RebuildParameter.mk:40:rebuild_parameter := $(intermediates)/parameter.txt
common/build/rockchip/RebuildParameter.mk:45:   @echo "Building parameter.txt $@."
tgg@tgg-virtual-machine:~/rk3568_android_sdk/device/rockchip$
```

Open this file:

```
ifdef PRODUCT_PARAMETER_TEMPLATE

$(info build parameter.txt with $(PRODUCT_PARAMETER_TEMPLATE)....)

ifeq ($(strip $(BOARD_USES_AB_IMAGE)), true)
partition_list := security:4M,uboot_a:4M,trust_a:4M,misc:4M
else
partition_list := security:4M,uboot:4M,trust:4M,misc:4M
endif # BOARD_USES_AB_IMAGE

ifeq ($(strip $(BOARD_USES_AB_IMAGE)), true)
# Header V3, add vendor_boot and resource.
ifeq (1,$(strip $(shell expr $(BOARD_BOOT_HEADER_VERSION) \>= 3)))
```

```
    partition_list := $(partition_list),resource_a:16M,vendor_boot_a:$(BOARD_VENDOR_BOOTIMAGE_PARTITION_SIZE)
    endif # Header V3
    partition_list := $(partition_list),dtbo_a:$(BOARD_DTBOIMG_PARTITION_SIZE),vbmeta_a:1M,boot_a:$(BOARD_BOOTIMAGE_PARTITION_SIZE)
    else # None-A/B
    # Header V3, add vendor_boot and resource.
    ifeq (1,$(strip $(shell expr $(BOARD_BOOT_HEADER_VERSION) \>= 3)))
    partition_list := $(partition_list),resource:16M,vendor_boot:$(BOARD_VENDOR_BOOTIMAGE_PARTITION_SIZE)
    endif # Header V3
    partition_list := $(partition_list),dtbo:$(BOARD_DTBOIMG_PARTITION_SIZE),vbmeta:1M,boot:$(BOARD_BOOTIMAGE_PARTITION_SIZE),recovery:$(BOARD_RECOVERYIMAGE_PARTITION_SIZE)
    endif # BOARD_USES_AB_IMAGE

    partition_list := $(partition_list),backup:384M,cache:$(BOARD_CACHEIMAGE_PARTITION_SIZE),metadata:16M

    ifeq ($(strip $(BUILD_WITH_GOOGLE_FRP)), true)
    partition_list := $(partition_list),frp:512K
    endif

    ifneq ($(strip $(BOARD_WITH_SPECIAL_PARTITIONS)), )
    partition_list := $(partition_list),$(BOARD_WITH_SPECIAL_PARTITIONS)
    endif

    ifeq ($(strip $(BOARD_SUPER_PARTITION_GROUPS)),rockchip_dynamic_partitions)
    partition_list := $(partition_list),super:$(BOARD_SUPER_PARTITION_SIZE)
    endif # BOARD_USE_DYNAMIC_PARTITIONS

    intermediates := $(call intermediates-dir-for,FAKE,rockchip_parameter)
    rebuild_parameter := $(intermediates)/parameter.txt

    ROCKCHIP_PARAMETER_TOOLS := $(SOONG_HOST_OUT_EXECUTABLES)/parameter_tools

    $(rebuild_parameter) : $(PRODUCT_PARAMETER_TEMPLATE) $(ROCKCHIP_PARAMETER_TOOLS)
```

```
        @echo "Building parameter.txt $@."
        $(ROCKCHIP_PARAMETER_TOOLS) --input $(PRODUCT_PARAMETER_TEMPLATE) \
        --start-offset 8192 \
        --firmware-version $(BOARD_PLATFORM_VERSION) \
        --machine-model $(PRODUCT_MODEL) \
        --manufacturer $(PRODUCT_MANUFACTURER) \
        --machine $(PRODUCT_DEVICE) \
        --partition-list $(partition_list) \
        --output $(rebuild_parameter)

INSTALLED_RK_PARAMETER := $(PRODUCT_OUT)/$(notdir $(rebuild_parameter))
$(INSTALLED_RK_PARAMETER) : $(rebuild_parameter)
        $(call copy-file-to-new-target-with-cp)

ALL_DEFAULT_INSTALLED_MODULES += $(INSTALLED_RK_PARAMETER)
endif
```

When generating the parameter.txt partition table file, a partition table template file $(PRODUCT_PARAMETER_TEMPLATE) needs to be provided. This variable refers to the file device/rockchip/common/scripts/parameter_tools/parameter.in, and its content is as follows:

```
FIRMWARE_VER: ${_firmware_version}
MACHINE_MODEL: ${_machine_model}
MACHINE_ID: ${_machine_id}
MANUFACTURER: ${_manufacturer}
MAGIC: ${_magic}
ATAG: ${_atag}
MACHINE: ${_machine}
CHECK_MASK: ${_check_mask}
PWR_HLD: ${_pwr_hld}
TYPE: ${_type}
CMDLINE:mtdparts=rk29xxnand:${_partition_list}
~
```

RebuildParameter.mk will use this file as a template and finally generate parameter.txt.

The most important information in the partition table of parameter.txt is the partition size and the starting address of each partition; from this Makefile, we can see that the sizes of each partition are as follows:

Partition name	Partition size
security	Fixed at 4M size
uboot	Fixed at 4M size
trust	Fixed at 4M size
misc	Fixed at 4M size
dtbo	Dependent on the BOARD_DTBOIMG_PARTITION_SIZE variable
vbmeta	Fixed at 1M size
boot	Dependent on the BOARD_BOOTIMAGE_PARTITION_SIZE variable

recovery	Dependent on the BOARD_RECOVERYIMAGE_PARTITION_SIZE variable
backup	Fixed at 384M size
cache	Dependent on the BOARD_CACHEIMAGE_PARTITION_SIZE variable
metadata	Fixed at 16M size
super	Dependent on the BOARD_SUPER_PARTITION_SIZE variable

The size of the userdata partition has not been defined because, apart from these partitions, all the remaining storage space belongs to the userdata partition, so it does not need to be defined with a size.

The sizes of these partitions can be changed and adjusted according to actual needs. In summary, the partition size must be able to accommodate the corresponding image, and there must not be a situation where the image is larger than the partition. Otherwise, it will overflow and occupy space in other partitions, resulting in the corruption of other partitions' images.

17. Compilation Module

The Android source code is very large! It contains many modules, and each module can be compiled through the corresponding command. The module compilation commands provided by the make command or the build/envsetup.sh script (mm/mma/mmm/mmma) can be used to compile the specified module.

① , Introduction to Android.mk and Android.bp files

Android uses Android.mk or Android.bp files for modular compilation of Android source code. Each module directory contains an Android.mk or Android.bp file.

In the early versions of Android, Android.mk was used to compile the source code. Android.mk is a Makefile file, and the syntax of Android.mk follows the Makefile. However, as Android versions updated and the projects grew larger with more modules included, due to the inefficiency and slow incremental compilation speed of make, Google introduced the Soong build system after Android 7.0, aiming to replace make; it utilizes the Kati and Ninja build system components to accelerate the compilation of Android source code.

And it is hoped to replace Android.mk with Android.bp, because the Android.bp file is simple. They do not contain any conditional statements or control flow statements, so the Android.bp file is more convenient for generating ninja files (The Soong build system will convert all Android.mk and Android.bp files into ninja files, and then the Ninja tool will parse the ninja files to execute the final compilation).

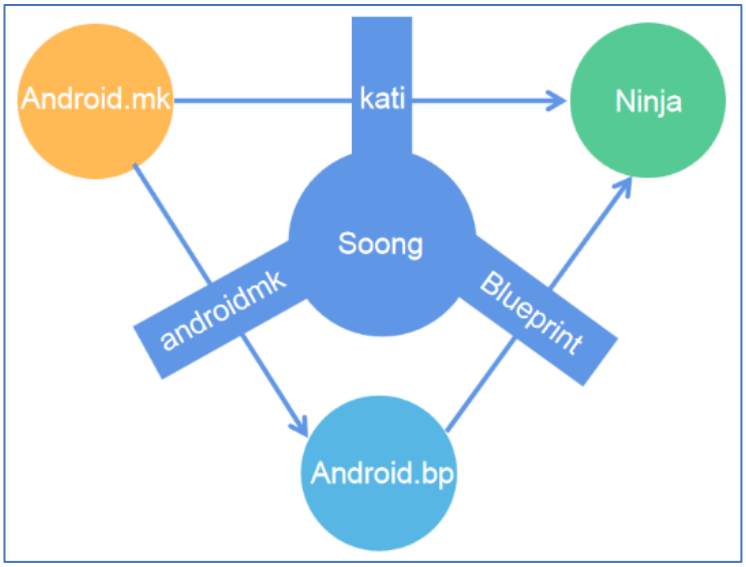


Figure 6.3.3.8 Relationship diagram

- Kati: It can convert the Android.mk file into a ninja file;
- androidmk: It can convert the Android.mk file into an Android.bp file;
- Blueprint: It can convert the Android.bp file into a ninja file.

In the out directory of the Android source code, one can indeed see many.ninja files, as shown below:

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk$
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ ls -l out/*.ninja
-rw-rw-r-- 1 tgg tgg       160 5月  12 16:44 out/build-rk3568_r-cleanspec.ninja
-rw-rw-r-- 1 tgg tgg 513877912 5月  12 16:45 out/build-rk3568_r.ninja
-rw-rw-r-- 1 tgg tgg    160717 5月  12 16:45 out/build-rk3568_r-package.ninja
-rw-rw-r-- 1 tgg tgg       160 5月  12 12:20 out/build-rk3568_r-userdataimage-cleanspec.ninja
-rw-rw-r-- 1 tgg tgg 513904159 5月  12 12:20 out/build-rk3568_r-userdataimage.ninja
-rw-rw-r-- 1 tgg tgg    160770 5月  12 12:20 out/build-rk3568_r-userdataimage-package.ninja
-rw-rw-r-- 1 tgg tgg       185 5月  13 13:51 out/combined-rk3568_r.ninja
-rw-rw-r-- 1 tgg tgg       213 5月  12 12:23 out/combined-rk3568_r-userdataimage.ninja
tgg@tgg-virtual-machine:~/rk3568_android_sdk$
tgg@tgg-virtual-machine:~/rk3568_android_sdk$
```

Figure 6.3.3.39 The ninja files under the out directory

From the above text, it can be seen that the conversion of Android.bp to ninja files is much more efficient than that of Android.mk files. Therefore, Android.mk files will gradually be replaced by Android.bp files; however, because before the introduction of Android.bp, there were already a large number of Android.mk files in the Android source code, completely replacing Android.mk is not something that can be achieved overnight. It cannot be completed all at once, so it results in the presence of both Android.mk and Android.bp files in the Android source code.

If you need to add your own module to the Android source code, it is recommended that you use the Android.bp file to write the compilation rules for the module.

②，Module compilation commands

To compile a specified module using the make command, the command format is as follows:

```
make <module_name>
```

"module_name" represents the name of the module that needs to be compiled; it can also be compiled using the compilation command provided by the "build/envsetup.sh" script (mm/mma/mmm/mmma), for example, "mmm":

mmm <module_path>

The "module_path" represents the path where the module is located.

The following lists several compilation commands for the main modules:

Module	Make command	Mmm command
init	make init	mmm system/core/init
zygtoe	make app_process	mmm frameworks/base/cmds/app_process
system_server	make services	mmm frameworks/base/services
java framework	make framework	mmm frameworks/base
Framework resource	make framework-res	mmm frameworks/base/core/res
jni framework	make libandroid_runtime	mmm frameworks/base/core/jni
binder	make libbinder	mmm frameworks/native/libs/binder
Settings	make Settings	mmm packages/apps/Settings

How is the module_name in the "**make <module_name>**" command determined? This module name is actually defined in the Android.mk or Android.bp file. Therefore, to confirm the module name, you need to check the Android.mk or Android.bp file in the module directory.

For example, taking the zygtoe module as an example, this module uses the Android.bp file to define the compilation rules. Open the Android.bp file in frameworks/base/cmds/app_process, as shown below:

```
cc_binary {
    name: "app_process",

    srcs: ["app_main.cpp"],

    multilib: {
        lib32: {
            version_script: ":art_sigchain_version_script32.txt",
            suffix: "32",
        },
        lib64: {
            version_script: ":art_sigchain_version_script64.txt",
            suffix: "64",
        },
    },

    ldflags: ["-Wl,--export-dynamic"],

    shared_libs: [
        "libandroid_runtime",
        "libbinder",
        "libcutils",
        "libdl",
        "libhidlbase",
        "liblog",
        "libnativeloader",
        "libutils",

        // This is a list of libraries that need to be included in order to avoid
        // bad apps. This prevents a library from having a mismatch when resolving
        // new/delete from an app shared library.
        // See b/21032018 for more details.
        "libwilhelm",
    ],

    whole_static_libs: ["libsigchain"],

    compile_multilib: "both",

    cflags: [
        "-Wall",
        "-Werror",
```

The "name" attribute defines the module name.

"cc_binary" indicates that the current module is an executable program module, meaning that an executable file will eventually be generated through compilation.

Figure 6.3.3.40 Viewing the module name in Android.bp

From this, we can see that the name of this module is "app_process", and executing the command "make app_process" can compile this module.

Note that the module name must be globally unique and there should not be two modules with the same name; otherwise, the compilation will fail. The "cc_binary" indicates that the current module will be compiled into an executable program. Regarding the grammar rules of the Android.bp file, this article does not introduce them. There are plenty of online resources!

Let's take a look at an example of Android.mk. For instance, the module "RKVRPlayer" in vendor/rockchip/common/apps, this module uses the Android.mk file to define the compilation rules. Open the Android.mk file in this directory as follows:

```
1 ##############################################################################
2 # Rk4kVideoPlayer
3 LOCAL_PATH := $(call my-dir)
4 include $(CLEAR_VARS)
5 LOCAL_MODULE := RKVRPlayer
6 LOCAL_MODULE_CLASS := APPS
7 LOCAL_MODULE_TAGS := optional
8 LOCAL_BUILT_MODULE_STEM := package.apk
9 LOCAL_MODULE_SUFFIX := $(COMMON_ANDROID_PACKAGE_SUFFIX)
10 #LOCAL_PRIVILEGED_MODULE :=
11 LOCAL_CERTIFICATE := PRESIGNED
12 #LOCAL_OVERRIDES_PACKAGES :=
13 LOCAL_SRC_FILES := $(LOCAL_MODULE).apk
14 #LOCAL_REQUIRED_MODULES :=
15 #LOCAL_PREBUILT_JNI_LIBS :=
16 include $(BUILD_PREBUILT)
```

LOCAL_MODULE is used to define the module name.

Figure 6.3.3.41 Viewing the module name in Android.mk

The LOCAL_MODULE variable in the Android.mk file defines the module name of the current module (having the same meaning as the name attribute in the Android.bp file). Therefore, executing the "make RKVRPlayer" command can compile this module. Similarly, regarding the grammar rules of the Android.mk file, this article does not provide an introduction. There are plenty of resources available online!

There are some differences between using "**make <module_name>**" and "**mmm <module_path>**" to compile a module. The "make <module_name>" command is only used to compile the specified module, while "mmm <module_path>" can compile all modules under the specified path (including those in its subdirectories), because an Android.mk or Android.bp file can define the compilation rules for multiple modules, not just one! This needs to be noted.

18. Clearing Commands

The following table lists several common clearing commands:

Make command	Explanation
make clean	Completely remove, which is equivalent to deleting all contents in the "out/" directory.
make clobber	The effect is the same as "make clean", which also deletes all contents in the "out" directory.
make installclean	See below.
make dataclean	It is equivalent to performing the following operation: rm -rf out/target/product/$(PRODUCT_DEVICE)/data/*
make clean-<module_name>	Clean the specified module. The parameter "module_name" represents the module name.

The following will be introduced one by one:

① **make clean**

This completely removes everything in the out/ directory, equivalent to deleting all contents of the out/ directory.

② **make clobber**

The effect is the same as make clean, also deleting all contents in the out/ directory. Generally, these commands are not easily executed because they will delete all contents in the out/ directory,

including all files built during compilation, all intermediate files, and the final image. Once deleted, it means having to recompile from scratch, and the process will take a very long time.

③ make installclean

This command will delete all installed files, including all *.img files in the product output directory, all .zip compressed files, the kernel image kernel, and other files. However, intermediate files will be retained. The specific details are shown in the following figure (build/soong/ui/build/cleanbuild.go):

```
        removeGlobs(ctx,
                hostOut("apex"),
                hostOut("obj/NOTICE_FILES"),
                hostOut("obj/PACKAGING"),
                hostOut("coverage"),
                hostOut("cts"),
                hostOut("nativetest*"),
                hostOut("sdk"),
                hostOut("sdk_addon"),
                hostOut("testcases"),
                hostOut("vts"),
                hostOut("vts10"),
                hostOut("vts-core"),
                productOut("*.img"),
                productOut("*.zip"),
                productOut("android-info.txt"),
                productOut("apex"),
                productOut("kernel"),
                productOut("data"),
                productOut("skin"),
                productOut("obj/NOTICE_FILES"),
                productOut("obj/PACKAGING"),
                productOut("ramdisk"),
                productOut("debug_ramdisk"),
                productOut("vendor-ramdisk"),
                productOut("vendor-ramdisk-debug.cpio.gz"),
                productOut("vendor_debug_ramdisk"),
                productOut("test_harness_ramdisk"),
                productOut("recovery"),
                productOut("root"),
                productOut("system"),
                productOut("system_other"),
                productOut("vendor"),
                productOut("product"),
                productOut("system_ext"),
                productOut("oem"),
                productOut("obj/FAKE"),
                productOut("breakpad"),
                productOut("cache"),
                productOut("coverage"),
                productOut("installer"),
                productOut("odm"),
                productOut("sysloader"),
                productOut("testcases"))
}
```

Figure 6.3.3.42 make installclean clears the file list

This list is defined in the build/soong/ui/build/cleanbuild.go file of the installClean function, indicating the files that need to be deleted when executing the "make installclean" command. hostOut represents the out/host/linux-x86/ directory, so hostOut("cts") means the out/host/linux-x86/cts directory; productOut represents the product output directory out/target/product/rk3568_r/, so productOut("*.img") means all *.img image files in the product output directory.

After the entire Android source code compilation is completed, if there are code updates, it is usually executed "make installclean" first before executing "make".

④ make dataclean

This is equivalent to deleting all contents in the data/ directory of the product output directory.

⑤ make clean<module_name>

This deletes all intermediate files (i.e., object target files) generated by the compilation of the specified module, as well as the final files compiled, such as executable programs, libraries, etc.

For example, to clear the init module:

```
make clean-init
```

6.3.4 Shutting down SELinux

6.3.4.1 Introduction to SELinux/SEAndroid

SELinux is the full name of Security-Enhanced Linux. On the Android system, it is generally referred to as SEAndroid. SEAndroid is a set of system security mechanisms based on SELinux that was officially launched by Google in Android 4.4.

The main function of SELinux is to enhance the security of the system, protect system resources from unauthorized access and attacks.

SElinux aims to address security issues. In simple terms, under the SELinux security mechanism, the specified process can only access the specific resources and perform specific operations that are authorized by the policy rules, while resources outside the rules cannot be accessed or operated; through the control of process permissions, it avoids unauthorized operations by the process, thereby meeting the security requirements of the system!

Compared with the traditional Linux discretionary access control (**Discretionary Access control, abbreviated as DAC) mechanism**, SELinux is a mandatory access control **(Mandatory Access control, abbreviated as MAC) mechanism.** SELinux provides more fine-grained and flexible security policies to protect the security of the system and data, and can be customized according to different scenarios and requirements to meet various security needs.

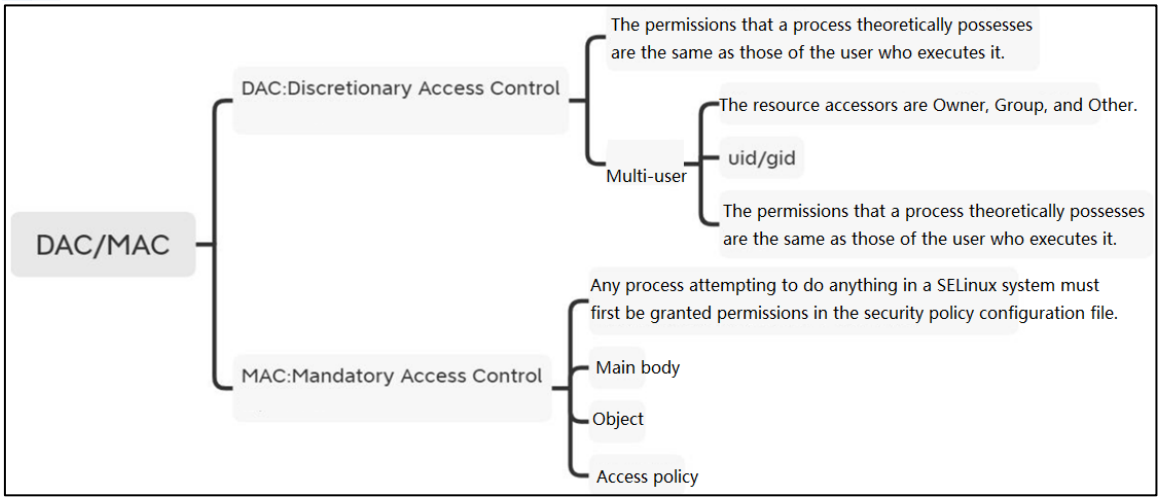
Before SElinux, Linux used the DAC (Discretionary Access Control) mechanism. The basic concept was: The system only provided basic permission verification, and the complete access control was controlled by the developers themselves (for example, permissions could be modified directly through chmod). Based on users and groups, the permissions of a process depend on the permissions of the user who invokes this process. Users are divided into three categories: Owner, Group, and Other, and permissions are defined according to these three categories.

- Resource Accessors: Various processes running in the system;
- The resources being accessed are usually files;
- The access permissions of resources are divided into three types: read, write, and execute.

If a process is started by the root superuser, then this process has the root user permissions, meaning that this process can do anything in the system. This is very dangerous and extremely insecure!

SElinux employs the MAC (Mandatory Access Control) permission control mechanism, which requires the system to strictly limit and verify every access of any process. The specific restrictions and verification policies (policies) are provided by the developers, that is, they need to clearly define which resources (in the Android system, the types of resources are not limited to files, but also include network ports, attributes, etc.) a certain process can access, and which operations can be performed on these resources.

MAC can significantly overcome the shortcomings of DAC. For example, it can limit root privileges. Even if you have root privileges, if you cannot pass the MAC verification, then you cannot perform the related operations. Additionally, the granularity of DAC's permission control is very large and not fine enough or flexible enough; while SElinux MAC provides more fine-grained and flexible security policies to protect the security of the system and data. It can be customized according to different scenarios and requirements to meet various security needs.



The Android system has not abandoned the DAC permission checking mechanism in the traditional Linux. Instead, it has added the MAC (SElinux is an implementation of MAC) permission checking mechanism on top of DAC. In the Android system, when a process accesses a certain resource, it first performs a DAC check. If the DAC permission check fails, the operation will directly fail; after passing the DAC check, the SElinux MAC permission check will be executed!

6.3.4.2 Shutting down SELinux

This section introduces how to shut down SELinux. There are several methods to shut down SELinux:

① To disable it, use the "setenforce" command.

To disable SELinux using the Android system command line terminal, execute the following command:

```
setenforce 0
```

Execute the following command to enable SELinux:

```
setenforce 1
```

This method is only for temporary modifications. It becomes invalid after a system restart or reset!

② Control the enabling or disabling of SELinux through the CONFIG_SECURITY_SELINUX variable.

Open the file device/rockchip/rk356x/rk3568_r/BoardConfig.mk, and add the following line:

```
BOARD_SELINUX_ENFORCING := false
```

The setting of the BOARD_SELINUX_ENFORCING variable to true indicates the activation of SELinux, while setting it to false indicates the deactivation of SELinux.

```
ifeq ($(strip $(BOARD_USES_AB_IMAGE)), true)
    include device/rockchip/common/BoardConfig_AB.mk
    TARGET_RECOVERY_FSTAB := device/rockchip/rk356x/rk3568_r/recovery.fstab_AB
endif

PRODUCT_UBOOT_CONFIG := rk3568
PRODUCT_KERNEL_DTS := rk3568-atk-evb1-ddr4-v10
BOARD_GSENSOR_MXC6655XA_SUPPORT := true
BOARD_CAMERA_SUPPORT_EXT := true
BOARD_HS_ETHERNET := true

BOARD_SELINUX_ENFORCING := false
```

After the modification, save and exit. Recompile the Android source code (it is recommended to first execute "make installclean" before making the compilation), and burn the compiled image onto the development board.

6.3.5 Application Pre-installation Function

The application pre-installation function on the Android system mainly refers to the situation when the product is configured, according to the requirements of the manufacturer, to pre-install third-party applications into the Android system in advance.

Pre-installation is divided into:

1. Installing uninstallable applications;

2. Installing applications that can be permanently uninstalled;

3. Installing applications that will automatically restore after uninstallation and restoring to factory settings.

① Configuration and Usage

First, execute "**source build/envsetup.sh**" to initialize the compilation environment, then execute the "lunch" command to select the product.

After selecting the product through the "lunch" command, by executing the "**get_build_var TARGET_DEVICE_DIR**" command, you can find the corresponding target folder (for example, device/rockchip/rk356x/rk3568_r/):

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk$
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ get_build_var TARGET_DEVICE_DIR
device/rockchip/rk356x/rk3568_r
tgg@tgg-virtual-machine:~/rk3568_android_sdk$
```

Enter this directory. There are three folders named "preinstall", "preinstall_del" and "preinstall_del_forever" within this directory, as shown in the following figure (if they do not exist, create them yourself):

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ cd device/rockchip/rk356x/rk3568_r
tgg@tgg-virtual-machine:~/rk3568_android_sdk/device/rockchip/rk356x/rk3568_r$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/device/rockchip/rk356x/rk3568_r$ ls -l
总用量 100
-rw-rw-r-- 1 tgg tgg   298 4月  22 15:50 AndroidBoard.mk
-rw-rw-r-- 1 tgg tgg    80 4月  22 15:50 Android.mk
-rw-rw-r-- 1 tgg tgg  1129 5月  12 12:10 BoardConfig.mk
-rw-rw-r-- 1 tgg tgg   152 4月  22 15:50 bt_vendor.conf
-rwxrwxr-x 1 tgg tgg  6129 4月  22 15:50 config.cfg
-rw-rw-r-- 1 tgg tgg  7959 4月  22 15:50 config.cfg_ab
-rw-rw-r-- 1 tgg tgg 11009 4月  22 15:50 config.cfg_ab_gki
-rw-rw-r-- 1 tgg tgg   316 4月  22 15:50 dt-overlay.in
-rw-rw-r-- 1 tgg tgg 24273 4月  22 15:50 media_profiles_default.xml
drwxrwxr-x 3 tgg tgg  4096 4月  22 15:50 ota
drwxrwxr-x 2 tgg tgg  4096 5月  17 10:44 preinstall
drwxrwxr-x 2 tgg tgg  4096 5月  17 10:44 preinstall_del
drwxrwxr-x 2 tgg tgg  4096 5月  17 10:44 preinstall_del_forever
-rw-rw-r-- 1 tgg tgg  3249 4月  22 15:50 recovery.fstab
-rw-rw-r-- 1 tgg tgg  1824 4月  22 15:50 recovery.fstab_AB
-rw-rw-r-- 1 tgg tgg  1656 5月  12 13:02 rk3568_r.mk
tgg@tgg-virtual-machine:~/rk3568_android_sdk/device/rockchip/rk356x/rk3568_r$
```

These three folders are as follows:

preinstall: Stores applications that cannot be uninstalled

preinstall_del_forever: Stores applications that can be permanently uninstalled

preinstall_del: Stores applications that can be automatically restored to factory settings after uninstallation

Just place the pre-installed APK applications in the corresponding folders. Note that the APK file names should be in English as much as possible and avoid using spaces. For example, if you want to pre-install the Sogou Input Method application in the Android system, the application needs to be uninstallable and should be able to automatically restore after uninstallation and factory reset. Therefore, the Sogou Input Method APK should be placed in the "preinstall_del/" directory, as shown below:

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/device/rockchip/rk356x/rk3568_r$ cd preinstall_del
tgg@tgg-virtual-machine:~/rk3568_android_sdk/device/rockchip/rk356x/rk3568_r/preinstall_del$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/device/rockchip/rk356x/rk3568_r/preinstall_del$ ls
SogouInput_11.18_android_sweb.apk
tgg@tgg-virtual-machine:~/rk3568_android_sdk/device/rockchip/rk356x/rk3568_r/preinstall_del$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/device/rockchip/rk356x/rk3568_r/preinstall_del$
```

② Compilation Results

Recompile the Android source code. After the compilation is completed, a folder named "bundled_uninstall_back-app" will be generated in the $OUT/odm directory, as shown below (OUT refers to the product output directory):

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/odm$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/odm$ pwd
/home/tgg/rk3568_android_sdk/out/target/product/rk3568_r/odm
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/odm$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/odm$ ls
bundled_uninstall_back-app  etc
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/odm$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/out/target/product/rk3568_r/odm$
```

After uninstallation and restoring to factory settings, applications that can be automatically restored will be installed to the odm/bundled_uninstall_back-app directory. The odm folder will be created as odm.img (which is merged into super.img, and super.img is burned onto the development board's super partition. After system startup, odm.img will be mounted to the Android root file system/odm directory).

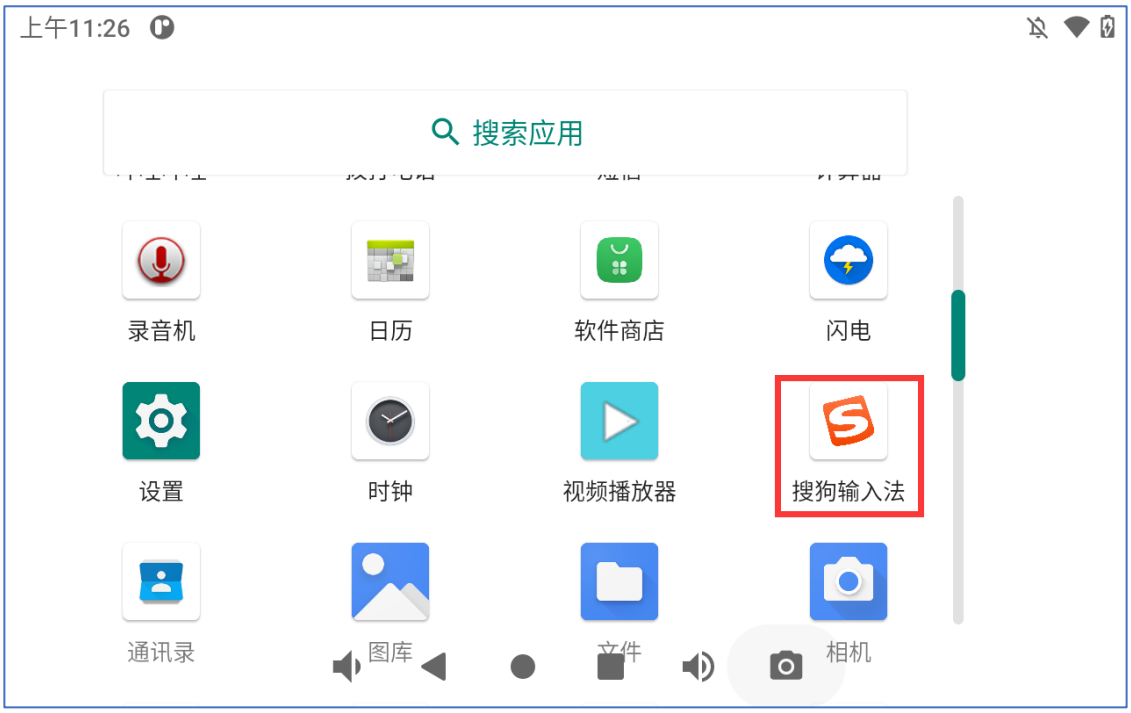
In addition to the bundled_uninstall_back-app directory, there are the following two directories:

bundled_uninstall_gone-app: Applications that can be permanently uninstalled will be installed to this directory;

bundled_persist-app: Applications that cannot be uninstalled will be installed to this directory.

③ **Burning the image file, starting the Android system**

Reburn the image file and then start the Android system. After entering the system, you will find the pre-installed Sogou input method application as shown below:



6.3.6 Modify default language and time zone information

Modify the default language and time zone information of the system. The main changes involve modifying the following two attributes:

```
persist.sys.locale
persist.sys.timezone
```

The first attribute "persist.sys.locale" represents the current system language, and the second attribute "persist.sys.timezone" represents the current system time zone.

Open the "device/rockchip/rk356x/rk356x.prop" attribute configuration file and add the following two lines of content:

```
persist.sys.locale=zh-Hans-CN
persist.sys.timezone=Asia/Shanghai
```

```
ro.vendor.rk_sdk=1

persist.sys.locale=zh-Hans-CN
persist.sys.timezone=Asia/Shanghai
~
~
```

zh-Hans-CN: Represents simplified Chinese, for example, en-US represents American English, ja-JP represents Japanese, ko-KR represents Korean, de-DE represents German, ru-RU represents Russian, etc.

Asia/Shanghai: Represents Shanghai, China (the time zone it belongs to), for example, America/New_York represents New York, USA, Europe/London represents London, Europe, Asia/Jerusalem represents Jerusalem, Asia/Dubai represents Dubai, Asia/Hong_Kong represents Hong Kong, China, Asia/Tokyo represents Tokyo, Japan, Asia/Seoul represents Seoul, South Korea, etc.

After the modification, save and exit. Recompile the Android source code (it is best to first execute "make installclean" before making the compilation), and burn the compiled image onto the development board.

6.3.7 Set the system to default not to lock the screen or enter sleep mode.

Modify the file:

device/rockchip/rk356x/overlay/frameworks/base/packages/SettingsProvider/res/values/defaults.xml

Open this file and add the following information:

```
integer name="def_screen_off_timeout">0x7fffffff</integer>
<bool name="def_lockscreen_disabled">true</bool>
```

The "def_screen_off_timeout" parameter represents the screen-off time, measured in milliseconds (ms). By setting it to the maximum value of 0x7fffffff, it indicates that the screen will never go off.

The "def_lockscreen_disabled" parameter is used to control whether to lock the screen. If set to true, it means no lock will be enforced; if set to false, it indicates that the screen will be locked.

```
18 -->
19 <resources>
20     <integer name="def_screen_off_timeout">0x7fffffff</integer>
21     <bool name="def_lockscreen_disabled">true</bool>
22     <!-- Initial value for the Settings.Secure.IMMERSIVE_MODE_CONFIRMATIONS setting,
23          which is a comma separated list of packages that no longer need confirmation
24          for immersive mode.
25          Override to disable immersive mode confirmation for certain packages. -->
```

After the modification is completed, save and exit. Recompile the Android source code (it is best to first execute "make installclean" before making the compilation), and burn the compiled image onto the development board.

6.3.8 Modify the default backlight brightness

Modify the file:

device/rockchip/common/overlay/frameworks/base/packages/SettingsProvider/res/values/defaults.xml

Find the "def_screen_brightness" parameter in this file, as shown below:

```
18 -->
19 <resources>
20     <!-- Default screen brightness, translate to 85 in kernel -->
21     <integer name="def_screen_brightness">240</integer>
22     <bool name="def_bluetooth_on">false</bool>
23 </resources>
~
```

The def_screen_brightness parameter represents the screen brightness, with values ranging from 0 to 255. The default value is 240. Users can modify it according to their own needs.

After making the modifications, save and exit. Recompile the Android source code (it is recommended to first execute "make installclean" before making the compilation), and burn the compiled image onto the development board.

6.3.9 Modify the Boot Logo

The boot logo includes the logo during the u-boot stage and the logo during the kernel stage. These two logo files are all located in the kernel source code directory:

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel$ pwd
/home/tgg/rk3568_android_sdk/kernel
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel$ ls -l *.bmp
-rw-rw-r-- 1 tgg tgg 177700 4月  28 15:24 logo.bmp
-rw-rw-r-- 1 tgg tgg 177764 4月  28 15:24 logo_kernel.bmp
tgg@tgg-virtual-machine:~/rk3568_android_sdk/kernel$
```

logo.bmp is the logo image for the U-Boot stage, and logo_kernel.bmp is the logo image for the kernel stage. Both are bmp images. The default startup logo of the ALIENTEK factory system is as follows:

U-Boot logo

Kernel logo

If you need to replace the default logo, first rename the U-Boot logo image to "logo.bmp" and the kernel logo image to "logo_kernel.bmp". Then copy these two bmp image files to the kernel source code directory to complete the replacement. After that, recompile the kernel.

When compiling the kernel, the "logo.bmp" and "logo_kernel.bmp" will be packaged into "resource.img", and then "resource.img" will be packaged into "boot.img".

When U-Boot starts, these two bmp files will be loaded into memory. The "logo.bmp" will start to be displayed during the U-Boot stage, and the address of "logo_kernel.bmp" in the memory will be passed to the Linux Kernel by U-Boot during the initialization of the DRM driver in the Linux Kernel.

6.3.10 Modify Boot Animation

If the user needs to modify the default boot animation of the Android system, they can refer to the content of this section for modification.

The Android boot animation is an animated form composed of a series of consecutive PNG images as frames, not a single GIF image. Each frame of the PNG images should be saved in a compressed format (the compression method should be storage compression), and the saved file name should be named "bootanimation.zip". This "bootanimation.zip" compressed file is the boot animation file.

① Creating the bootanimation.zip file for the boot animation

First, prepare a set of consecutive PNG images (you can export each frame of the animation video to generate a set of consecutive PNG images, for example, through the Format Factory software; of course, you can also manually create the corresponding PNG images for each frame). Use this set of images to create the boot animation.

Name this series of consecutive PNG images in numerical order according to their display sequence. If there are 100 PNG images, then this series of images can be named as 0001.png to 0100.png. Of course, it can also be named as 001.png to 100.png. Just make sure the number of digits in the file names is the same!

Create a folder named "bootanimation" in Windows, and create one or more directories named "partN" (where N represents a number, such as 0, 1, 2, 3, etc.) in this directory. The boot animation can be divided into multiple stages. Each "partN" directory is used to store the PNG images to be played in different stages. Each "partN" directory can contain a portion of images (one or more images):

The first-stage animation images are stored in the "part0" directory;

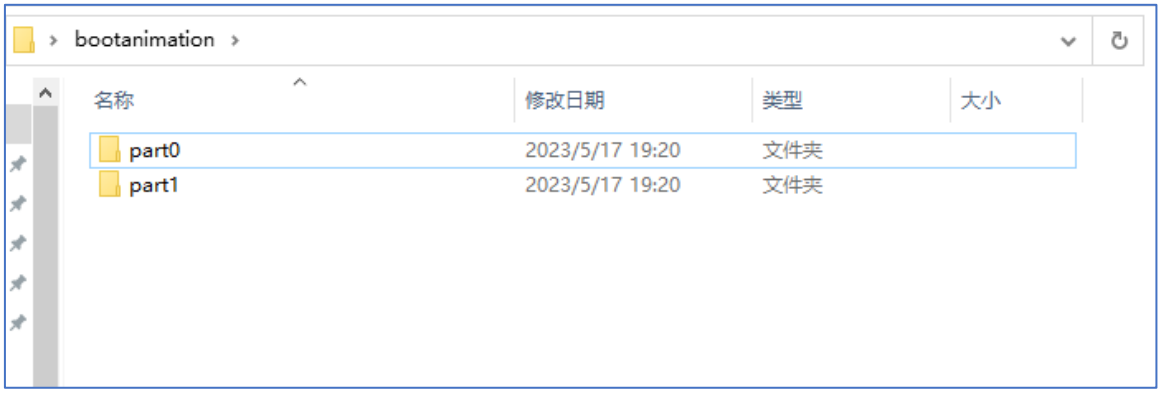
The second-stage animation images are stored in the "part1" directory;

The third-stage animation images are stored in the "part2" directory;

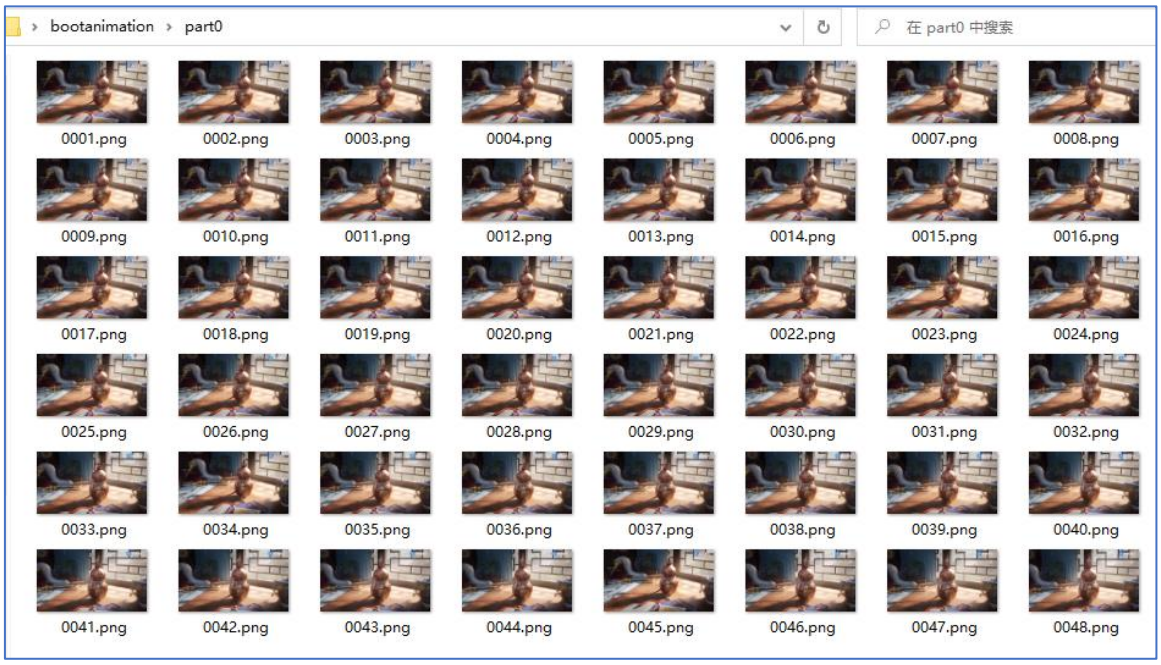
And so on!

If you just want to play all the images in a loop, only create the "part0" directory; if you want to play some images first and then loop another set of images, that is, there are two stages; then you need to create two folders, "part0" and "part1", where the "part0" directory stores the images to be played in the first stage, and the "part1" directory is used to store the images to be played in the second stage.

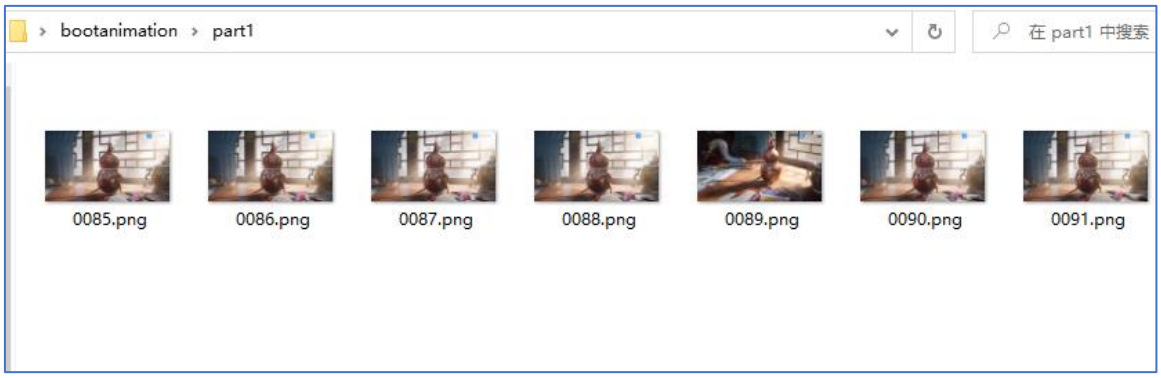
Suppose we want to create an opening animation with two stages. First, create two directories named "part0" and "part1", as shown below:



Place the pictures that need to be played in the first stage into the "part0" directory, as follows:



In this example, there are a total of 84 PNG images placed in the "part0" directory; then the images needed to be played in the second stage are placed in the "part1" directory, as follows:



In this example, there are a total of 7 pictures placed in the "part1" directory.

Next, we need to create a file named "desc.txt", which is referred to as the animation attribute description file. The desc.txt file is used to set information such as the pixel size of the animation, the number of frames, the number of loops, and the folders corresponding to each stage. The content of the desc.txt file is as follows:

```
480 427 20
p 1 0 part0
p 0 10 part1
```

480 and 427 represent the width and height of the image pixels;

20 indicates the number of frames, which is the number of images played per second; 20 also means playing 20 images per second;

p 1 0 part0: In this line, the first letter p indicates the meaning of playing, the second number 1 indicates the number of times this stage is played (this stage is played once), which is the number of times this stage is looped; the third number 0 indicates the interval time, for example, in the case of looped playback, after one round of playback, how much time is needed to play the next round, the unit is the duration of one frame, for example, 20 frames, then the duration of one frame is 1 second / 20 = 50ms. The last parameter part0 indicates that the images played in this stage are stored in the port0 directory.

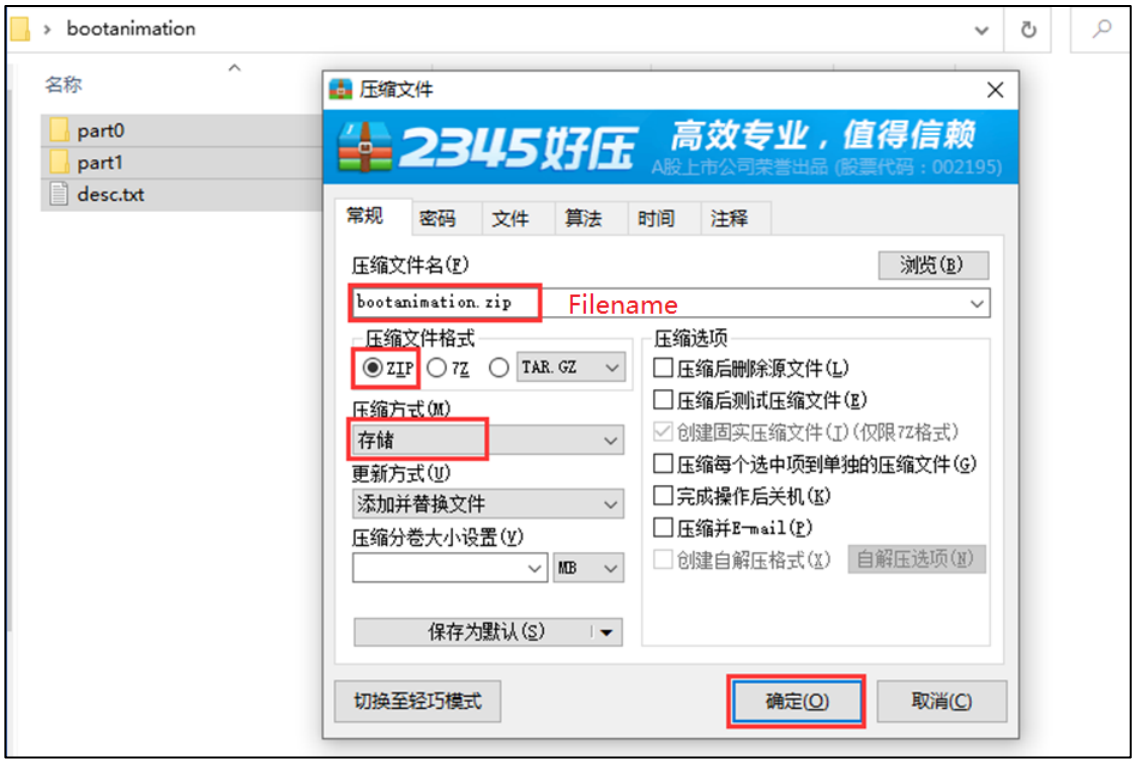
p 0 10 part1: The meaning of this line is the same as the second line. The first letter p indicates the meaning of playing, the second number 0 indicates the number of times this stage is looped (0 indicates infinite loop in this stage); the third number 10 indicates the interval time; the last parameter part1 indicates that the images played in this stage are stored in the port1 directory.

Let's take a look at the meaning of the entire animation attribute description file desc.txt:

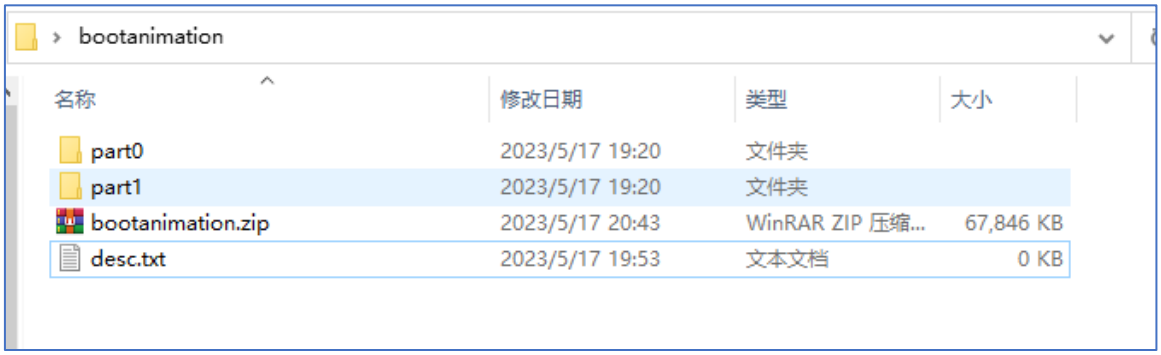
First, play all the images in the part0 directory at a frame rate of 20 frames per second, and then play all the images in the part1 directory in an infinite loop at a frame rate of 20 frames per second.

Now let's start packing:

At the same time, select part0, part1 and desc.txt, then right-click with the mouse and choose "**Add to Archive**". The compression method should be selected as "**Save**", as shown below:



In this way, the animation file "bootanimation.zip" is successfully created.



② Product Configuration

Copy the generated bootanimation.zip file to the directory device/rockchip/common/bootshutdown/ in the RK3568 Android source code, as shown below:

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/device/rockchip/common/bootshutdown$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/device/rockchip/common/bootshutdown$ pwd
/home/tgg/rk3568_android_sdk/device/rockchip/common/bootshutdown
tgg@tgg-virtual-machine:~/rk3568_android_sdk/device/rockchip/common/bootshutdown$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/device/rockchip/common/bootshutdown$ ls
bootanimation.zip  bootshutdown.mk
tgg@tgg-virtual-machine:~/rk3568_android_sdk/device/rockchip/common/bootshutdown$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/device/rockchip/common/bootshutdown$
```

And set the permissions of the bootanimation.zip file to 755 as follows:

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/device/rockchip/common/bootshutdown$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/device/rockchip/common/bootshutdown$ chmod 755 bootanimation.zip
tgg@tgg-virtual-machine:~/rk3568_android_sdk/device/rockchip/common/bootshutdown$ ls -l bootanimation.zip
-rwxr-xr-x 1 tgg tgg 5843557 5月  17 16:50 bootanimation.zip
tgg@tgg-virtual-machine:~/rk3568_android_sdk/device/rockchip/common/bootshutdown$
```

Next, open the board configuration file device/rockchip/rk356x/rk3568_r/BoardConfig.mk and add the following line of content:

```
BOOT_SHUTDOWN_ANIMATION_RINGING := true
```

```
#
# Copyright 2014 The Android Open-Source Project
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#      http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#
include device/rockchip/rk356x/BoardConfig.mk
BUILD_WITH_GO_OPT := false

# AB image definition
BOARD_USES_AB_IMAGE := false
BOARD_ROCKCHIP_VIRTUAL_AB_ENABLE := false

BOOT_SHUTDOWN_ANIMATION_RINGING := true

ifeq ($(strip $(BOARD_USES_AB_IMAGE)), true)
    include device/rockchip/common/BoardConfig_AB.mk
    TARGET_RECOVERY_FSTAB := device/rockchip/rk356x/rk3568_r/recovery.fstab_AB
endif

PRODUCT_UBOOT_CONFIG := rk3568
PRODUCT_KERNEL_DTS := rk3568-atk-evb1-ddr4-v10
BOARD_GSENSOR_MXC6655XA_SUPPORT := true
BOARD_CAMERA_SUPPORT_EXT := true
BOARD_HS_ETHERNET := true
```

After making the modifications, save and exit the file. Then recompile the Android source code (it is recommended to first execute the "make installclean" command before making the compilation), and then burn the compiled image onto the development board.

③ Compiling Android source code and burning the image

After burning the compiled image onto the development board, restart the development board, enter the Android system, and notice that the startup animation has changed.

Reference document of RK:

RKDocs/android/Rockchip_Introduction_Android_Power_On_Off_Animation_and_Tone_Customization_CN&EN.pdf

6.3.11 Modify the main screen / secondary screen

RK Reference Document:

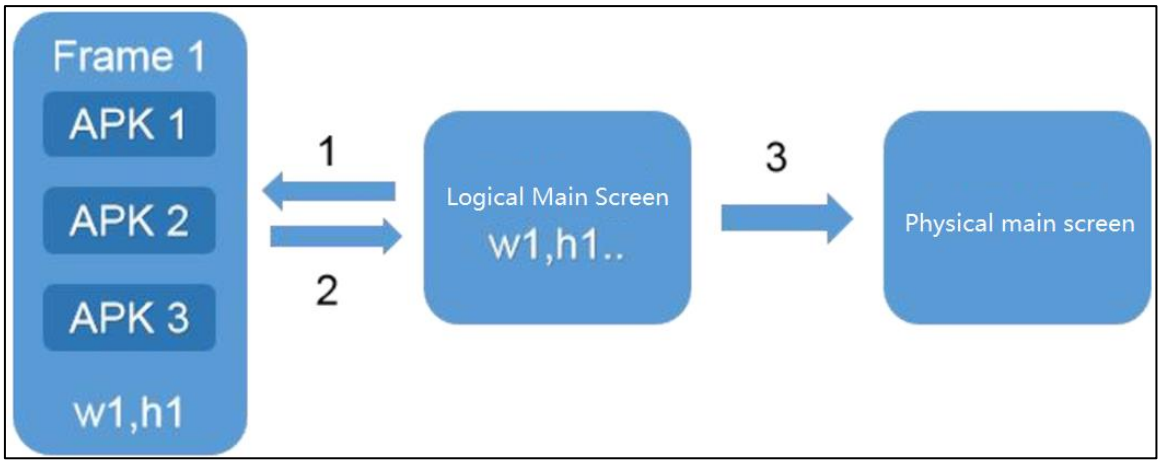
<SDK>/RKDocs/common/display/Rockchip_Developer_Guide_Dual_Display_Rotation_Direction_Debugging_CN.pdf

The logical screens of the Android system are divided into the main screen, the secondary screen, and the virtual screen (this does not consider the virtual screen for now), corresponding to the actual

physical screens; there is only one main screen, while there can be multiple secondary screens. The attributes of the logical screens in the system and the display paths will vary according to the display requirements, such as multi-screen simultaneous display and multi-screen different display.

① There is only one logical screen

If the system has only one logical screen, that is, the logical main screen, then the display path is as follows:



The APK obtains the attribute information (width/height/fps) of the main screen and uses it as the rendering layout for the APK.

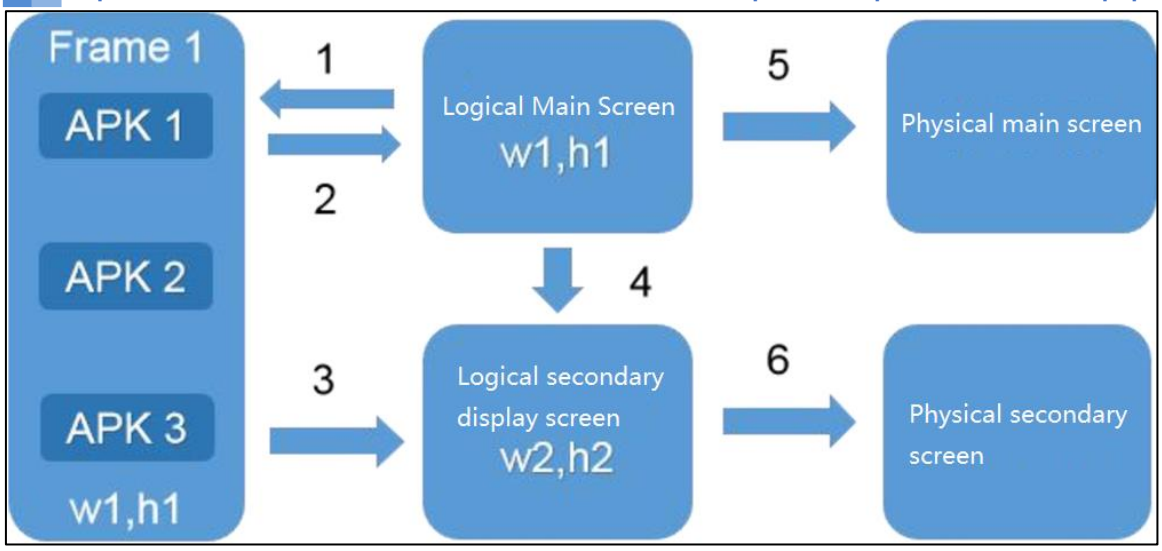
2: Multiple APKs will display the request submission on the logical main screen, and the main screen will mix them based on the hierarchical structure.

3: The logical main screen will display the combined result on the actual physical main screen. By default, the attribute information of the logical main screen is the same as that of the physical main screen.

Note: Since the APK performs rendering and drawing based on the attribute information of the main screen, in the case of having only one logical main screen, the display effect is normal and there are no issues such as stretching.

② Multi-screen Display

Multi-screen display, for example, the RK3568 platform supports dual-screen display and triple-screen display. In fact, the secondary logical screen sends the display content of the primary logical screen to the physical secondary screen. The display path is as follows:



1.The APK obtains the attribute information (width/height/fps) of the main screen and uses it as the rendering layout for the APK.

2: Multiple APKs will display the request submission on the logical main screen, and the main screen will mix them based on the hierarchical structure.

3: Multiple APKs will display the request submission to the logical secondary screen, and the secondary screen will mix them according to the hierarchical structure.

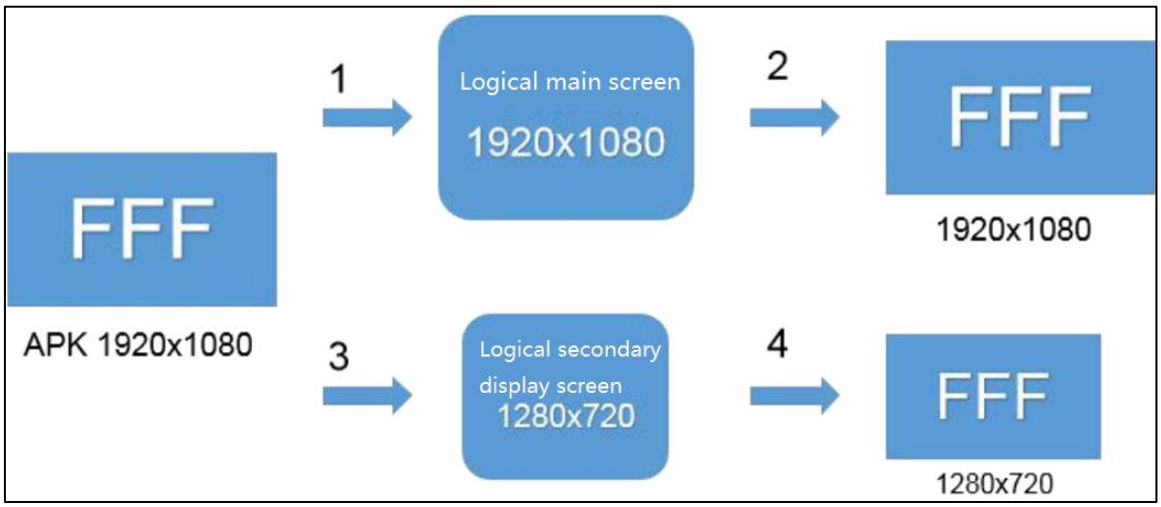
Step 4: 3 can actually be equivalent to the situation where the main logic screen submits the display content to the secondary logic screen.

5: Logical Main Screen The combined result is displayed on the actual physical main screen. By default, the attribute information of the logical main screen is the same as that of the physical main screen.

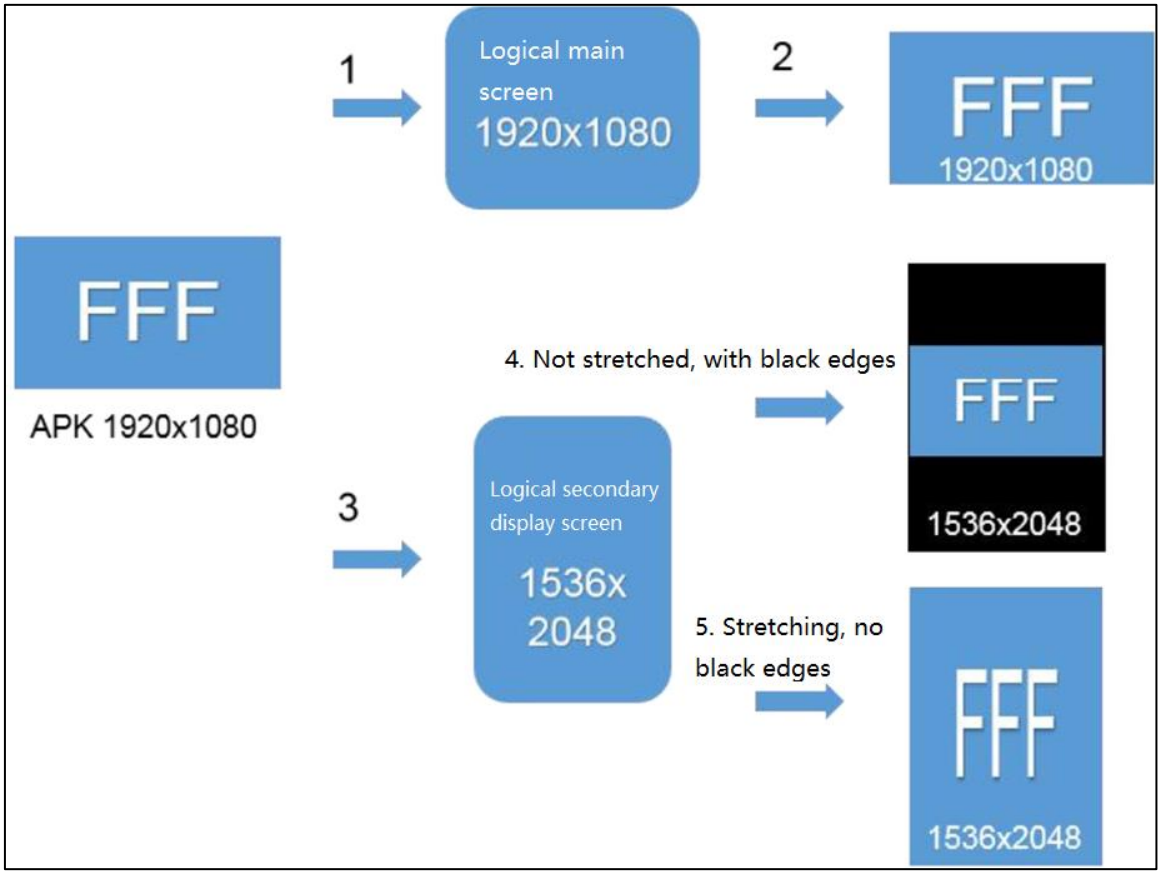
6: Logical secondary display screen Transmits the combined result to the actual physical secondary display screen

When using multi-screen display, the logical secondary screen may have black borders or be stretched. The specific display scenarios are as follows:

- The pixel width and height ratios of the main and secondary screens are the same: that is, w1/h1 = w2/h2:

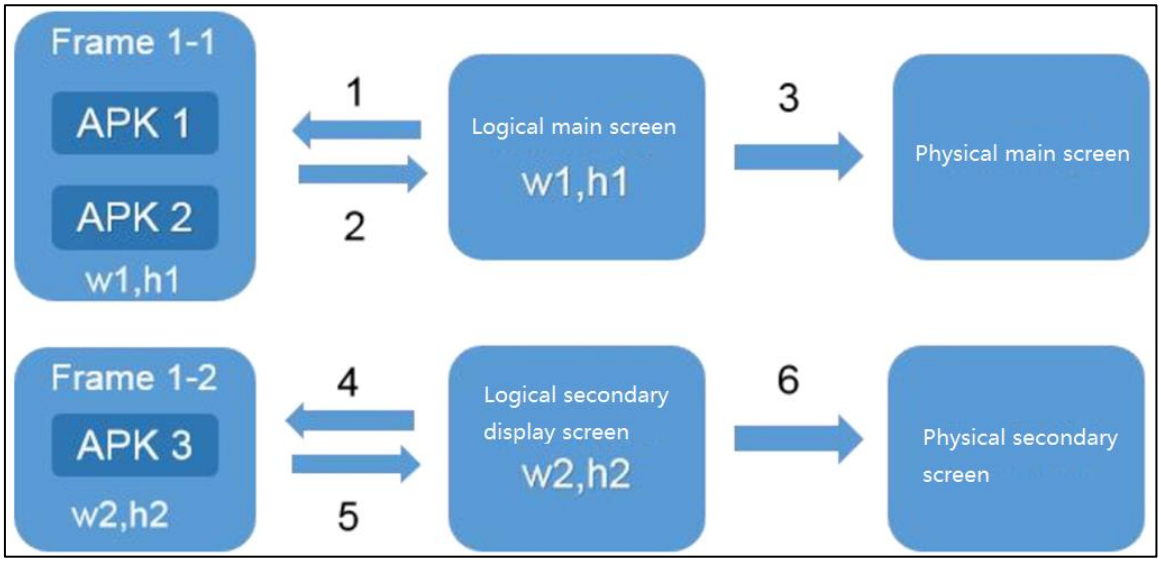


- If the pixel aspect ratios of the main and secondary screens are different: that is, w1/h1 is not equal to w2/h2:



③ Multi-screen Different Display

Multi-screen Different Display refers to the fact that the APK independently draws based on the attribute information of each logical screen and then independently sends the display. This is different from multi-screen same display:



1.APK1 and APK2 obtain the attribute information (width/height/fps) of the main screen of the logical device, which is then used as the rendering layout for APK1 and APK2.

2: APK1 and APK2 will display the request submission to the logical main screen, and the main screen will perform a mixture based on the hierarchical structure.

3: Logical Main Screen The combined result is displayed on the actual physical main screen. By default, the attribute information of the logical main screen is the same as that of the physical main screen.

4: APK3 acquires the attribute information (width/height/fps) of the secondary display screen and uses it as the rendering layout for APK3.

5: APK3 will display the request submission to the logical secondary screen, and the secondary screen will perform the mixing based on the hierarchical structure.

6: Logical secondary display Sends the combined result to the actual physical secondary display. By default, the attribute information of the logical secondary display is the same as that of the physical secondary display.

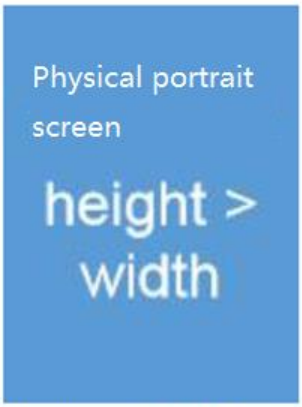
Multi-screen heterogeneous display: Since the display content sent by the APK is independently drawn, the rendering layout of the APK is always consistent with the physical screen attributes that need to be displayed. Therefore, there is no stretching or black borders, and all screens display normally. The displayed content depends on each respective APK.

④ Physical landscape screen and physical portrait screen

- Physical landscape view

That is, screens with an actual width greater than height, such as 1920*1080, 1280*720, etc.

- Physical portrait screen



That is, screens with an actual height greater than the width, such as 1536*2048, 1080*1920, etc.

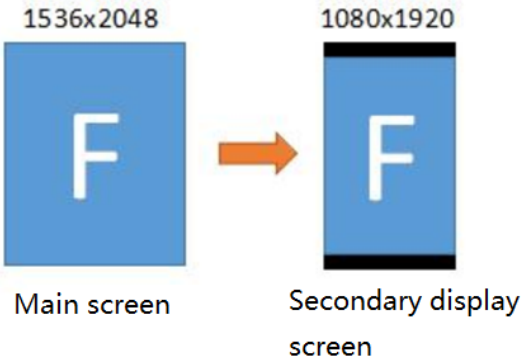
⑤. Attribute Introduction

Next, four attributes will be introduced: **persist.sys.rotation.efull-1, persist.sys.rotation.einit-1, vendor.hwc.device.primary, and vendor.hwc.device.extend**. Users can define these attributes in the device/rockchip/rk356x/rk356x.prop configuration file according to their actual project requirements.

- The persist.sys.rotation.efull-1 attribute is used to control whether the secondary screen displays in full screen mode. Its values can be false or true:

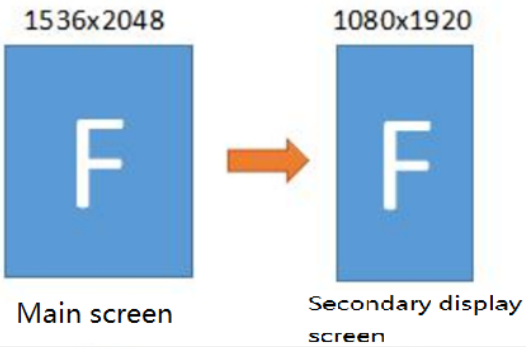
false:

The secondary screen image is scaled according to the aspect ratio of the main screen. If the aspect ratios are not consistent, black borders will appear.

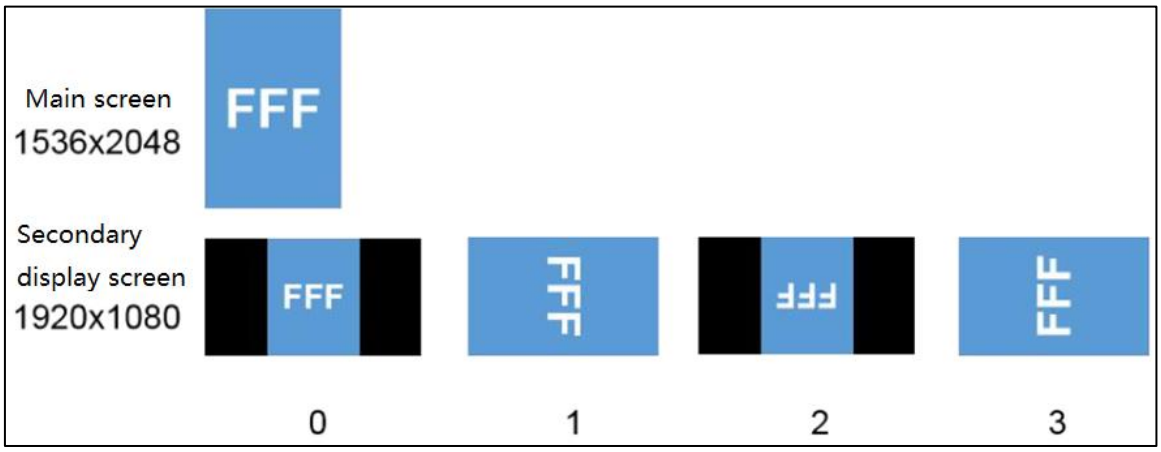


true:

The secondary screen image is forced to display in full screen, highlighting the stretching effect.



- The "persist.sys.rotation.einit-1" attribute is used to set the display orientation of the secondary screen. Its possible values are 0/1/2/3, which respectively represent rotating the display orientation of the secondary screen (clockwise) by 0 degrees, 90 degrees, 180 degrees, and 270 degrees.



- The "vendor.hwc.device.primary" attribute is used to set the primary screen, for example:

vendor.hwc.device.primary=DSI	# Set the MIPI screen as the main screen
vendor.hwc.device.primary=HDMI-A	# Set the HDMI screen as the main screen
vendor.hwc.device.primary=LVDS	# Set the LVDS screen as the main screen
vendor.hwc.device.primary=eDP	# Set the eDP screen as the main screen

- The "vendor.hwc.device.extend" attribute is used to set the secondary display, for example:

vendor.hwc.device.extend=DSI	# Set the MIPI screen as the secondary display screen
vendor.hwc.device.extend=HDMI-A	# Set the HDMI screen as the secondary display screen
vendor.hwc.device.extend=LVDS	# Set the LVDS screen as the secondary display screen
vendor.hwc.device.extend=eDP	# Set the eDP screen as the secondary display screen

6.3.12 Change screen orientation

The ALIENTEK ATK-DLRK3568 development board supports various LCD screens, including 5.5-inch MIPI screens (720p and 1080p), 10.1-inch LVDS screens, eDP screens, and HDMI displays. The 5.5-inch MIPI screen of Zongdian is a physically vertical screen (pixel width is less than pixel height), the 10.1-inch LVDS screen is a physically horizontal screen (pixel width is greater than pixel height), and HDMI displays are usually horizontal screens.

The Android system supports both landscape and portrait display modes. During the development process, the screen we used might be a physically horizontal or vertical screen. If we want to change a physically horizontal screen to a vertical display or a physically vertical screen to a horizontal display, for example, converting a MIPI screen to a vertical display or a LVDS screen to a horizontal display, then this situation requires configuration and modification of the screen orientation in software, rotating the display direction by 90 degrees or 270 degrees.

This section takes the 5.5-inch MIPI screen as an example to introduce to users how to change it to a portrait display, divided into the following two parts:

1. How to rotate the display direction when the MIPI screen is the main screen;
2. How to rotate the display direction when the MIPI screen is the secondary screen;

6.3.12.1 Changing the display direction of the main screen

① **Select LCD screen**

Enter the kernel source code directory, open the device tree file arch/arm64/boot/dts/rockchip/rk3568-screen_choose.dtsi, select the LCD screen, as shown below:

```
/*
 * 屏幕选择
 * ATK_LCD_TYPE_MIPI_720P:   正点原子5.5寸 720*1280 MIPI屏
 * ATK_LCD_TYPE_MIPI_1080P:  正点原子5.5寸 1080*1920 MIPI屏
 * ATK_LCD_TYPE_LVDS:        正点原子10.1寸 1280*800 LVDS屏
 * ATK_LCD_TYPE_HDMI:        HDMI显示器
 * ATK_LCD_TYPE_EDP_VGA:     eDP屏或者VGA显示器（硬件默认使能的是VGA接口，若用户需要使用eDP屏，则需修改硬件
 *                           具体情况可以看正点原子RK3568底板原理图！）
 */
/*
 * RK3568可支持三屏显示，也就是三路显示 VP0 VP1 VP2
 * 但是三屏显示需要注意一些问题，具体情况可以看正点原子提供的文档<RK3568三屏显示参考手册>!
 */

/*
 * ATK_LCD_TYPE_MIPI_720P 和 ATK_LCD_TYPE_MIPI_1080P 二选一
 */
#define ATK_LCD_TYPE_MIPI_720P          // 从VP1输入
//#define ATK_LCD_TYPE_MIPI_1080P       // 从VP1输入

/*
 * ATK_LCD_TYPE_HDMI 和 ATK_LCD_TYPE_EDP_VGA 二选一
 */
//#define ATK_LCD_TYPE_HDMI             // 从VP0输入
//#define ATK_LCD_TYPE_EDP_VGA          // 从VP0输入

//#define ATK_LCD_TYPE_LVDS             // 从VP2输入
```

Let's take the 5.5-inch 720p MIPI screen as an example. Just define the macro ATK_LCD_TYPE_MIPI_720P! After the modification, save and exit.

② **Configure the display orientation of the main screen of the Android system**

Return to the root directory of the Android source code, open the board-level configuration file device/rockchip/rk356x/BoardConfig.mk, find the variable SF_PRIMARY_DISPLAY_ORIENTATION, and modify it to 90, as shown below:

```
SF_PRIMARY_DISPLAY_ORIENTATION := 90
```

```
23 PRODUCT_UBOOT_CONFIG ?= rk3568
24 PRODUCT_KERNEL_ARCH ?= arm64
25 PRODUCT_KERNEL_DTS ?= rk3568-evb1-ddr4-v10
26 PRODUCT_KERNEL_CONFIG ?= rockchip_defconfig
27
28 # BOARD_AVB_ENABLE := true
29 SF_PRIMARY_DISPLAY_ORIENTATION := 90
30
31 # Disable emulator for "make dist" until there is a 64-bit qemu kernel
32 BUILD_EMULATOR := false
33 TARGET_BOARD_PLATFORM := rk356x
34 TARGET_BOARD_PLATFORM_GPU := mali-G52
35 TARGET_RK_GRALLOC_VERSION := 4
36 BOARD_USE_DRM := true
```

This variable is used to control the display direction of the main screen.

This variable can take values of 0/90/180/270, respectively representing a rotation (clockwise) of 0 degrees, 90 degrees, 180 degrees, and 270 degrees of the main screen display direction.

After the modification is completed, save and exit.

③ Set the MIPI screen as the main screen

Open the device/rockchip/rk356x/rk356x.prop property configuration file and add the following content:

```
vendor.hwc.device.primary=DSI
vendor.hwc.device.extend=HDMI-A
```

```
ro.vendor.rk_sdk=1
vendor.hwc.device.primary=DSI
vendor.hwc.device.extend=HDMI-A
~
~
```

vendor.hwc.device.primary=DSI indicates that the MIPI screen is set as the primary screen.

vendor.hwc.device.extend=HDMI-A indicates that the HDMI display is used as the secondary screen.

This step is not necessary to perform. If only one screen (the MIPI screen) is enabled, it will be automatically set as the primary screen by default.

After the modification, save and exit.

④ Configure touch direction

Enter the kernel source code directory, open the device tree file arch/arm64/boot/dts/rockchip/rk3568-atk-evb1-ddr4-v10.dtsi, find the touch screen device node, as follows:

```
&i2c1 {
        dsi_touch: gt911@14 {
                compatible = "goodix,gt9xx";
                reg = <0x14>;
                pinctrl-names = "default";
                pinctrl-0 = <&touch_gpio>;
                touch-gpio = <&gpio0 RK_PB5 IRQ_TYPE_LEVEL_LOW>;
                reset-gpio = <&gpio0 RK_PB6 GPIO_ACTIVE_HIGH>;
                max-x = <1080>;
                max-y = <1920>;
                tp-size = <911>;
                tp-supply = <&vcc5v0_sys>;
                wakeup-source;
                status = "okay";
        };
};
```

This is the touch device corresponding to the MIPI screen.

```
&i2c3 {
        status = "okay";

        lvds_touch: gt911@14 {
                compatible = "goodix,gt9xx";
                reg = <0x14>;
                pinctrl-names = "default";
                pinctrl-0 = <&lvds_touch_gpio>;
                touch-gpio = <&gpio0 RK_PC5 IRQ_TYPE_LEVEL_LOW>;
                reset-gpio = <&gpio3 RK_PC4 GPIO_ACTIVE_HIGH>;
                max-x = <1280>;
                max-y = <800>;
                tp-size = <911>;
                tp-supply = <&vcc5v0_sys>;
                wakeup-source;
                status = "okay";
        };
};
```

This is the touch device corresponding to the LVDS screen.

Add the following two attributes to the corresponding touch device node of the MIPI screen:

```
touchscreen-inverted-x;
touchscreen-swapped-x-y;
```

```
&i2c1 {
        dsi_touch: gt911@14 {
                compatible = "goodix,gt9xx";
                reg = <0x14>;
                pinctrl-names = "default";
                pinctrl-0 = <&touch_gpio>;
                touch-gpio = <&gpio0 RK_PB5 IRQ_TYPE_LEVEL_LOW>;
                reset-gpio = <&gpio0 RK_PB6 GPIO_ACTIVE_HIGH>;
                max-x = <1080>;
                max-y = <1920>;
                tp-size = <911>;
                tp-supply = <&vcc5v0_sys>;
                wakeup-source;
                touchscreen-inverted-x;
                touchscreen-swapped-x-y;
                status = "okay";
        };
};
```

The first attribute "touchscreen-inverted-x" indicates that the X-axis coordinates are processed in terms of direction. If the Y-axis coordinates need to be processed in terms of direction, then the attribute "touchscreen-inverted-y" must be defined.

The second attribute "touchscreen-swapped-x-y" indicates that the X-axis and Y-axis are swapped.

For different rotation angles, the processing methods of the X-axis and Y-axis are different. You need to understand it yourself and imagine it with your hands to get a general idea. There's nothing more to say.

After the modification is completed, save and exit. Ensure that the content of the make.sh script file in the kernel source code directory has been modified according to the requirements of the section "<6.2.2.2 - Modify make.sh script>".

⑤ Modify the display direction of the logo

Enter the kernel source code directory, open the device tree file "arch/arm64/boot/dts/rockchip/rk3568-atk-evb1-ddr4-v10.dtsi", find the "route_dsi1" node, as follows:

```
&route_dsi1 {
        status = "okay";
        connect = <&vp1_out_dsi1>;
};
```

Add the following attribute to this node：

```
logo,rotate = <1>;
```

```
&route_dsi1 {
        status = "okay";
        connect = <&vp1_out_dsi1>;
        logo,rotate = <1>;
};
```

The "logo,rotate" attribute is used to set the display direction of the logo. Its values can be 0, 1, 2, or 3, representing a rotation of 0 degrees, 90 degrees, 180 degrees, and 270 degrees respectively.

After the modification is completed, save and exit!

⑥ **Compile and Burn the Image**

Switch to the root directory of the Android source code, and execute the following command to compile the Android source code and the kernel source code:

```
make installclean -j16
./build.sh -KAu -J16
```

After the compilation is completed, the image in the "rockdev/Image-rk3568_r/" directory will be burned onto the development board.

6.3.12.2 Modify the display direction of the secondary screen

The configuration is also carried out following steps ① to ⑤.

① **Select LCD screen**

Enter the kernel source code directory, open the device tree file "arch/arm64/boot/dts/rockchip/rk3568-screen_choose.dtsi", select the LCD screen, as shown below:

```
/*
 * 屏幕选择
 * ATK_LCD_TYPE_MIPI_720P:   正点原子5.5寸 720*1280 MIPI屏
 * ATK_LCD_TYPE_MIPI_1080P:  正点原子5.5寸 1080*1920 MIPI屏
 * ATK_LCD_TYPE_LVDS:        正点原子10.1寸 1280*800 LVDS屏
 * ATK_LCD_TYPE_HDMI:        HDMI显示器
 * ATK_LCD_TYPE_EDP_VGA:     eDP屏或者VGA显示器（硬件默认使能的是VGA接口，若用户需要使用eDP屏，则需修改硬件
 *                           具体情况可以看正点原子RK3568底板原理图！）
 */
/*
 * RK3568可支持三屏显示，也就是三路显示 VP0 VP1 VP2
 * 但是三屏显示需要注意一些问题，具体情况可以看正点原子提供的文档<RK3568三屏显示参考手册>!
 */

/*
 * ATK_LCD_TYPE_MIPI_720P 和 ATK_LCD_TYPE_MIPI_1080P 二选一
 */
#define ATK_LCD_TYPE_MIPI_720P          // 从VP1输入
//#define ATK_LCD_TYPE_MIPI_1080P       // 从VP1输入

/*
 * ATK_LCD_TYPE_HDMI 和 ATK_LCD_TYPE_EDP_VGA 二选一
 */
//#define ATK_LCD_TYPE_HDMI             // 从VP0输入
//#define ATK_LCD_TYPE_EDP_VGA          // 从VP0输入

#define ATK_LCD_TYPE_LVDS               // 从VP2输入
```

Here, we take the LVDS screen and the MIPI screen as examples. The LVDS screen is used as the main screen, and the 5.5-inch 720p MIPI screen is used as the secondary screen. Two macros, ATK_LCD_TYPE_MIPI_720P and ATK_LCD_TYPE_LVDS, need to be defined! After the modification, save and exit.

② **Configure the display direction of the secondary screen in the Android system**

Return to the root directory of the Android source code, open the property configuration file of device/rockchip/rk356x/rk356x.prop, and add the following content:

```
persist.sys.rotation.efull-1=false
persist.sys.rotation.einit-1=1
```

```
ro.vendor.rk_sdk=1
persist.sys.rotation.efull-1=false
persist.sys.rotation.einit-1=1
~
```

The persist.sys.rotation.einit-1 attribute is used to set the display orientation of the secondary screen. Its possible values are 0, 1, 2, and 3, which respectively represent rotating the secondary screen display direction (clockwise) by 0 degrees, 90 degrees, 180 degrees, and 270 degrees.

The persist.sys.rotation.efull-1 attribute is described in the <6.3.11 - Modifying the Main Screen/Secondary Screen> section. No further explanation is needed here!

After the modification is completed, save and exit.

③ Set the MIPI screen as the secondary screen

Open the device/rockchip/rk356x/rk356x.prop file again and add the following content:

```
vendor.hwc.device.primary=LVDS
vendor.hwc.device.extend=DSI
```

```
ro.vendor.rk_sdk=1
persist.sys.rotation.efull-1=false
persist.sys.rotation.einit-1=1
vendor.hwc.device.primary=LVDS
vendor.hwc.device.extend=DSI
~
```

vendor.hwc.device.primary=LVDS indicates that the LVDS screen is set as the primary screen;

vendor.hwc.device.extend=DSI indicates that the DSI (MIPI screen) is set as the secondary screen.

After the modification, save and exit.

④ Configure touch direction

Refer to point ④ in Section 6.3.12.1.

⑤ Modify logo display direction

Refer to point ⑤ in Section 6.3.12.1. For the LVDS screen, the route_lvds node needs to be modified.

⑥Compile and burn the image

Refer to point ⑤ in Section 6.3.12.1.

6.3.12.3 Rotation of the screen in recovery mode

If you need to modify the screen direction in the recovery mode, you can open the file device/rockchip/common/BoardConfig.mk, find the TARGET_RECOVERY_DEFAULT_ROTATION variable, as follows:

```
#rotate screen to 0, 90, 180, 270
#0:   ROTATION_NONE      ORIENTATION_0   : 0
#90:  ROTATION_RIGHT     ORIENTATION_90  : 90
#180: ROTATION_DOWN    ORIENTATION_180: 180
#270: ROTATION_LEFT    ORIENTATION_270: 270
# For Recovery Rotation
TARGET_RECOVERY_DEFAULT_ROTATION ?= ROTATION_NONE
# For Surface Flinger Rotation
SF_PRIMARY_DISPLAY_ORIENTATION ?= 0
```

The display direction of the recovery mode screen can be specified through the TARGET_RECOVERY_DEFAULT_ROTATION variable. The default value is ROTATION_NONE, which indicates no rotation of the display direction. The possible values are: ROTATION_NONE,

ROTATION_RIGHT, ROTATION_DOWN, and ROTATION_LEFT, representing a rotation of 0, 90, 180, and 270 degrees respectively.

After the modification is completed, save and exit, and recompile the Android source code.

6.3.13 Adding Custom Modules

The Android source code is very large! It contains a large number of modules. This section introduces to users how to add a custom module in the Android source code. This example takes a C/C++ executable program module as an example.

① **Adding Modules**

Let's take the vendor directory as an example. First, enter the vendor directory under the root directory of the Android source code, and create a folder with the manufacturer's name in this directory, such as "alientek", as shown below:

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ cd vendor/
tgg@tgg-virtual-machine:~/rk3568_android_sdk/vendor$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/vendor$ ls
rockchip  widevine
tgg@tgg-virtual-machine:~/rk3568_android_sdk/vendor$ mkdir alientek
tgg@tgg-virtual-machine:~/rk3568_android_sdk/vendor$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/vendor$ ls
alientek  rockchip  widevine
tgg@tgg-virtual-machine:~/rk3568_android_sdk/vendor$
```

Enter the "alientek" directory and create a C or C++ source file as follows:

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/vendor$ cd alientek/
tgg@tgg-virtual-machine:~/rk3568_android_sdk/vendor/alientek$ ls
tgg@tgg-virtual-machine:~/rk3568_android_sdk/vendor/alientek$ touch my_test.c
tgg@tgg-virtual-machine:~/rk3568_android_sdk/vendor/alientek$ ls
my_test.c
tgg@tgg-virtual-machine:~/rk3568_android_sdk/vendor/alientek$
```

And open the "my_test.c" source file, and write a test code. The test code for this example is as follows:

```
#include <stdio.h>

int main(void)
{
    printf("Hello World!\n");
    return 0;
}
```

Next, an Android.bp or Android.mk file needs to be created in the current directory. It is recommended to use Android.bp.

Android uses Android.mk or Android.bp files to modularize the compilation of Android source code. As introduced in the <**6.3.3.17 - Module Compilation**> section, this has already been explained, so no further elaboration is necessary here.

To create the Android.bp file, do it as follows:

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/vendor/alientek$ ls
my_test.c
tgg@tgg-virtual-machine:~/rk3568_android_sdk/vendor/alientek$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/vendor/alientek$ touch Android.bp
tgg@tgg-virtual-machine:~/rk3568_android_sdk/vendor/alientek$ ls
Android.bp  my_test.c
tgg@tgg-virtual-machine:~/rk3568_android_sdk/vendor/alientek$
```

Open the Android.bp file and add the following content:

```
cc_binary {
    name: "my_c_module",
    srcs: ["my_test.c"],
}
```

```
cc_binary {
    name: "my_c_module",
    srcs: ["my_test.c"],
}
~
~
```

"cc_binary" indicates that this module is a C/C++ executable program module, and it will eventually be compiled into a binary executable program. The "name" attribute specifies the name of the module as "my_c_module". The "srcs" attribute is used to specify the source files required for compiling this module. The content of this file is very simple. Of course, apart from these two attributes, the Android.bp file also supports many other attributes. Regarding the grammar rules of the Android.bp file, this article does not introduce them. Please search on Baidu by yourself!

After completing the writing, save and exit. Then return to the root directory of the Android source code (you can switch to the root directory by directly executing the "croot" command):

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk/vendor/alientek$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/vendor/alientek$ ls
Android.bp  my_test.c
tgg@tgg-virtual-machine:~/rk3568_android_sdk/vendor/alientek$
tgg@tgg-virtual-machine:~/rk3568_android_sdk/vendor/alientek$ croot
tgg@tgg-virtual-machine:~/rk3568_android_sdk$
tgg@tgg-virtual-machine:~/rk3568_android_sdk$
```

② Compilation Module

Switch to the root directory of the Android source code, and then compile our module using the command "make my_c_module":

```
make my_c_module -j12
```

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk$
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ make my_c_module -j12
============================================
PLATFORM_VERSION_CODENAME=REL
PLATFORM_VERSION=11
TARGET_PRODUCT=rk3568_r
TARGET_BUILD_VARIANT=userdebug
TARGET_BUILD_TYPE=release
TARGET_ARCH=arm64
TARGET_ARCH_VARIANT=armv8-a
TARGET_CPU_VARIANT=cortex-a55
TARGET_2ND_ARCH=arm
TARGET_2ND_ARCH_VARIANT=armv8-2a
TARGET_2ND_CPU_VARIANT=cortex-a55
HOST_ARCH=x86_64
HOST_2ND_ARCH=x86
HOST_OS=linux
HOST_OS_EXTRA=Linux-5.15.0-53-generic-x86_64-Ubuntu-20.04.2-LTS
HOST_CROSS_OS=windows
HOST_CROSS_ARCH=x86
HOST_CROSS_2ND_ARCH=x86_64
HOST_BUILD_TYPE=release
BUILD_ID=RD2A.211001.002
OUT_DIR=out
============================================
10:33:14 Disallowed PATH tool "python3.8" used: []string{"python3.8", ".repo/repo/repo", "manifest", "-r", "-o",
10:33:14 See https://android.googlesource.com/platform/build/+/master/Changes.md#PATH_Tools for more information.
[100% 2/2] Install: out/target/product/rk3568_r/system/bin/my_c_module

#### build completed successfully (3 seconds) ####
```

The parameter "my_c_module" after "make" is the name of the module; the compiled executable program will eventually be installed at "$OUT/system/bin/my_c_module", as shown below:

```
tgg@tgg-virtual-machine:~/rk3568_android_sdk$
tgg@tgg-virtual-machine:~/rk3568_android_sdk$ file out/target/product/rk3568_r/system/bin/my_c_module
out/target/product/rk3568_r/system/bin/my_c_module: ELF 64-bit LSB shared object, ARM aarch64, version 1 (SYSV),
dynamically linked, interpreter /system/bin/linker64, BuildID[md5/uuid]=f4feb3a70cfb230cb5d70b2392c9aee8, stripped
tgg@tgg-virtual-machine:~/rk3568_android_sdk$
```

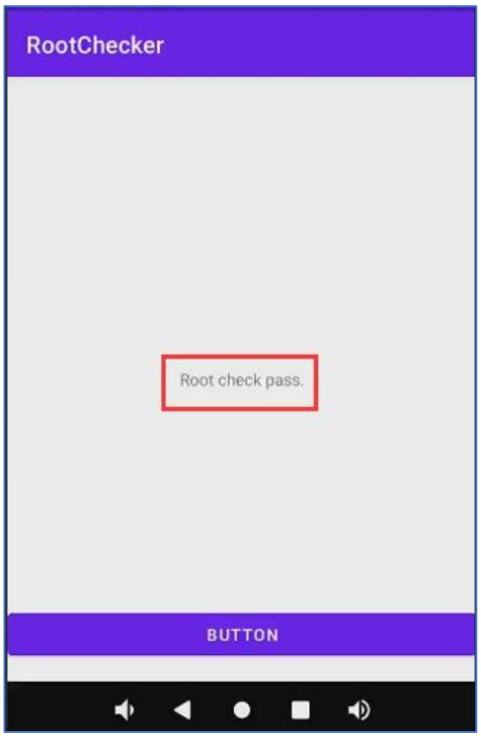
6.3.14 Android 11 obtains root access

Reference: The document provided by RK: **RKDocs/android/patches/root/android11_root.pdf.**

According to the "android11_root.pdf" document, recompile the Android source code and burn the image onto the development board.

Start the development board and enter the Android system. Use adb to install the root check APK: RKDocs/android/patches/root/RootChecker.apk.

After the APK is installed successfully, open the app. If it prompts "root check pass", as shown in the following picture, it indicates that the root check has been passed!



6.3.15 Multi-screen Different Display and Touch

Reference document: **RKDocs/android/patches/customized_functions/Android11 Multi-display Development Instructions.zip.**

6.3.16 Multi-screen Different Sound

Reference document: **RKDocs/android/patches/customized_functions/Dual_Audio_v1.0.zip**.

6.3.17 Three-screen Display

Reference document: Development board CD-ROM A Disk - **Basic Materials -> 10_user_manual -> 03, Auxiliary Documents -> [ALIENTEK] ATK-DLRK3568 Development Board Three-screen Display Reference Manual.pdf**.

6.4 Android Debugging

Introduction to Android system debugging tools.

6.4.1 ADB

Reference document: Development board CD-ROM A Disk - **Basic Materials -> 10_user_manual -> 03, Auxiliary Documents -> [ALIENTEK] adb Tool Usage Instructions.pdf.**

6.4.2 logcat

The Android logging system provides the function of recording and viewing system debugging information. Logs are recorded from various software and some system buffers. Buffers can be viewed and used through the logcat command-line tool. Logcat is the most commonly used tool during the debugging program stage. This tool mainly displays the running status of the program by printing logs. Due to the large amount of logs to be printed, filtering and other operations are usually required.

① Usage of the logcat command

To view the contents of the system log buffer using the logcat command, the basic usage is as follows:

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

If you want to run logcat via adb shell, then do it as follows:

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

Here is an example:

```
adb shell
logcat
```

Running logcat directly without any parameters will print all the log information in the default log buffer (main/system/crash), which is very large! This is usually not done and filtering is often required.

② Common Log Filtering Methods

- Control the priority of log output, for example:

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

Example description: Display log information with a priority level of warning or higher. The priority levels are one of the following letters (arranged in the order from lowest to highest priority):

- V : Detailed (with the lowest priority)
- D : Debugging
- I : Information
- W : Warning
- E : Error
- F : Serious mistake
- S : Silence (highest priority, will never output any content)

- Control log labels and output priorities. For example:

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

Example explanation: This expression only outputs log information with the tag "ActivityManager" and a priority level no lower than "Information (I)", as well as log information with the tag "MyApp" and a priority level no lower than "Debug (D)". All other types of information except these two will not be output.

- Only output the log information with specific tags. For example:

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

Or

```
adb shell
logcat –s WishTV
```

Example description: Only log information with the tag "WishTV" will be output.

- Only output the log information of the specified priority and tag. For example:

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

Example description: Output only the log messages with the label "WishTV" and a priority level no lower than I.

- View the log information of the specified buffer

The Android log information buffer is divided into the default log buffer and the backup log buffer. The default log buffer includes: system, main, and crash. In addition to the default buffer, there are also several backup buffers.

- `radio`: Check the buffer containing messages related to wireless devices/telephones
- `events`: View the buffered binary system event messages that have been interpreted
- `main`: View the main log buffer (default), excluding system and crash log messages
- `system`: View the system log buffer (default)
- `crash`: View crash log buffer (default)
- `all`: View all buffers
- `default`: Report the main, system and crash buffers

The logcat command can be used to view the log information in the specified buffer (and multiple -b options can be added to specify multiple buffers). An example is as follows:

```
adb shell
logcat -b radio
```

Example description: Only view the log buffer that contains information related to wireless devices and telephones (radio).

```
adb shell
```

```
logcat -b system *:W
```

Example explanation: Only view all log information in the system log buffer (system) with a priority not lower than W.

③ **Clear log information**

The logcat command uses the -c parameter to clear the log buffer, which means clearing the log information. Here is an example:

```
adb shell
logcat -c
```

Clear the default buffer (system/main/crash).

```
adb shell
logcat -b all -c
```

Clear all buffers.

```
adb shell
logcat -b system -c
```

Example explanation: Clear the system log buffer of the system.

6.5 Android Application Development

Reference document: Development board CD-ROM A disc - Basic materials →10_user_manual →02. Development documents→ [ALIENTEK] ATK-DLRK3568 Android Application Development.

6.6 Secure Boot/Safety Mechanism Development

RK Reference Document:
RKDocs/common/NVM/Rockchip_Developer_Guide_OTP_CN.pdf
RKDocs/common/u-boot/Rockchip_Developer_Guide_UBoot_Nextdev_CN.pdf

<SDK>/RKDocs/common/security

```
├── RK3399_Efuse_Operation_Instructions_V1.00_EN.pdf
├── RK356X_SecurityBoot_And_AVB_instructions_CN.pdf
├── RK356X_SecurityBoot_And_AVB_instructions_EN.pdf
├── Rockchip_Developer_Guide_Crypto_HWRNG_CN.pdf
├── Rockchip_Developer_Guide_Secure_Boot_Application_Note_EN.pdf
├── Rockchip_Developer_Guide_Secure_Boot_for_UBoot_Next_Dev_CN.pdf
├── Rockchip_Developer_Guide_Secure_Boot_for_UBoot_Next_Dev_EN.pdf
├── Rockchip_Developer_Guide_TEE_SDK_CN.pdf
└── Rockchip Vendor Storage Application Note.pdf
```

<SDK>/RKDocs/android

```
├── Rockchip_Introduction_Android10.0_Safety_Boot_Solution_CN.pdf
├── Rockchip_Introduction_Android_Verify_Boot_CN&EN.pdf
```

6.7 A/B System Development

RK Reference Document:

<SDK>/RKDocs/common/u-boot

```
├── Rockchip_Developer_Guide_UBoot_Nextdev_CN.pdf
├── Rockchip-Developer-Guide-Linux-AB-System.pdf
```

<SDK>/RKDocs/common/security

```
├── RK356X_SecurityBoot_And_AVB_instructions_CN.pdf
├── RK356X_SecurityBoot_And_AVB_instructions_EN.pdf
└── Rockchip Vendor Storage Application Note.pdf
```

<SDK>/RKDocs/android

```
├── Rockchip_Introduction_Android_AB_System_Upgrading_CN&EN.pdf
```

6.8 System Upgrade Development

System Upgrade Plan, RK Reference Document:

<SDK>/RKDocs/common/NVM

```
├── Rockchip_Application_Notes_Storage_CN.pdf
```

<SDK>/RKDocs/android

```
├── Rockchip_User_Guide_Recovery_CN&EN.pdf
```