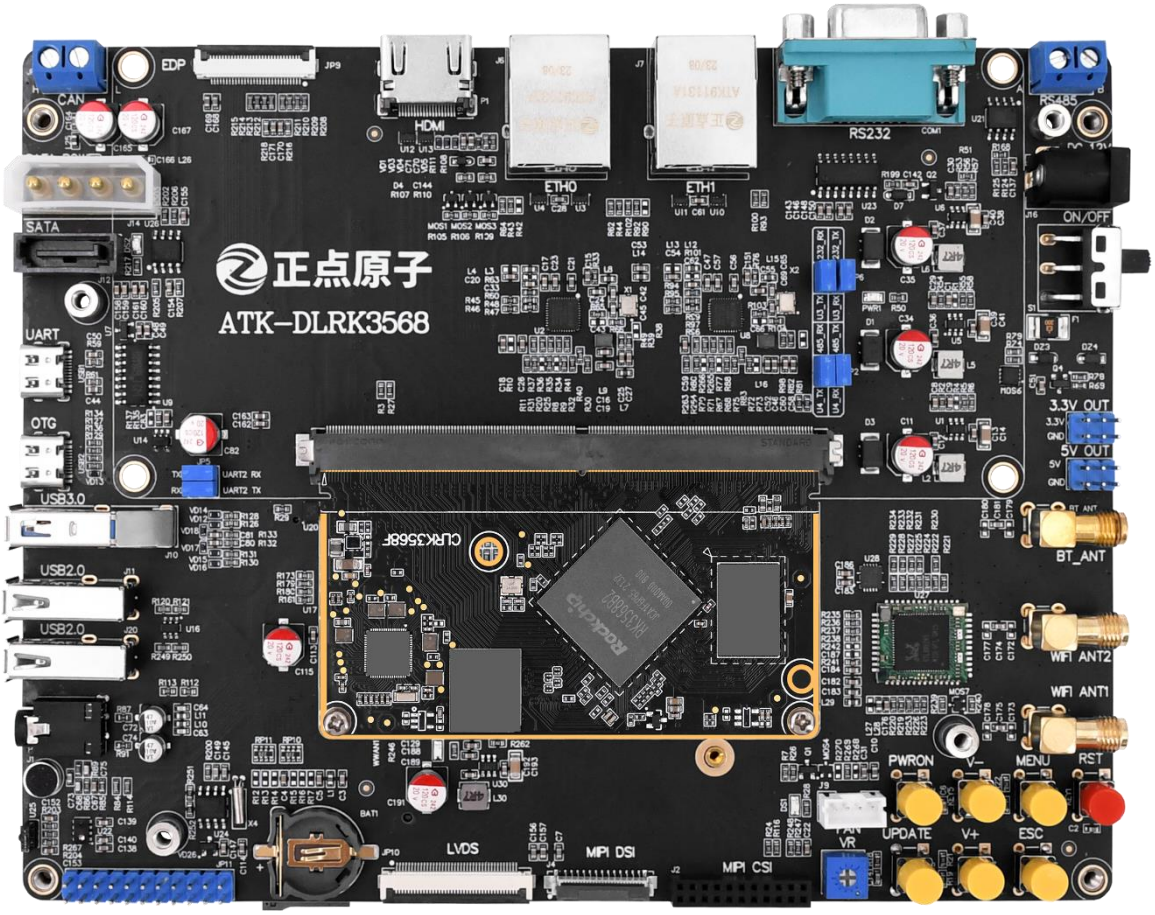
ATK-DLRK3568

Embedded Linux System Development Manual

V1.3



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	2023.06.20
V1.1	4.1.1 Section: The installation command uses "\ " for line breaks;	ALIENTEK	ALIENTEK	2023.08.30
V1.2	Add: 6.1.3 section, 6.2.2 section, 6.3.4 section and 6.4.2 section	ALIENTEK	ALIENTEK	2023.11.01
V1.3	Modify Section 4.3 and add a compilation description for the compilation configuration items. Since users often choose the card computer configuration to compile the ATK-DLRK3568 board.	ALIENTEK	ALIENTEK	2024.07.10

Catalogue

Brief 1
Overview of Embedded Development Environment 2
Chapter 1. Install the Ubuntu system 4
1.1 Install VMware virtual machine software 4
1.1.1 Download VMware software 4
1.1.2 Installing VMware software 7
1.2 Installing the Ubuntu system 13
1.2.1 Downloading the Ubuntu system image 13
1.2.2 Create a virtual machine 15
1.2.3 Installing the Ubuntu System 28
Chapter 2. Development environment setup 44
2.1 Ubuntu System Settings 44
2.1.1 Set the root user password 44
2.1.2 Change the software download source 45
2.1.3 Disable automatic updates 50
2.2 File transfer between Ubuntu and Windows 50
2.2.1 Set up an FTP server on the Ubuntu system 50
2.2.2 Installing FTP client on Windows 51
2.2.3 How to use FileZilla 55
2.3 Setting up the tftp server in the Ubuntu system 61
2.4 Set up an NFS server on the Ubuntu system 61
2.5 Set up an SSH server on the Ubuntu system 61
2.6 Installation of CH340 serial port driver 62
2.7 Installation of MobaXterm software 64
2.7.1 Download of MobaXterm software 64
2.7.2 Installation of MobaXterm software 66
2.7.3 Usage of MobaXterm software 69
2.8 Rockchip USB Driver Installation 77
2.9 Usage of Mirror Image Burning Tools 79
2.9.1 Introduction to Burning Modes 79
2.9.2 Using the RKDevTool Tool on Windows 81
2.9.3 The method for burning the update.img image 88
2.9.4 Erase operation 89
2.9.5 Usage of the Linux_Upgrade_Tool tool under Ubuntu 91
2.10 Installation of ADB tools 91
Chapter 3. Introduction to the ALIENTEK ATK-DLRK3568 Platform 92
3.1 Introduction to RK3568 92
3.2 Introduction to the Hardware Resources of ALIENTEK ATK-RK3568 Development Board 92
Chapter 4. RK3568 Linux SDK software package 93
4.1 Install RK3568 Linux SDK 93

4.1.1 Install dependent software packages 93
4.1.2 Install repo (skip) 94
4.1.3 Git Configuration 94
4.1.4 Install SDK....95
4.1.5 SDK Update 96
4.1.6 SDK Issue Feedback 96
4.1.7 SDK Optimization....97
4.2 Introduction to SDK Software Architecture....97
4.2.1 Introduction to SDK Project Directory Structure....97
4.2.2 SDK Software Block Diagram....98
4.2.3 Query of SDK Version....99
4.3 SDK Automatic Compilation....99
4.4 Independently compiling....107
4.4.1 Compile U-Boot separately....107
4.4.2 Compile Kernel Independently 108
4.4.3 Independently compiling rootfs 109
4.4.4 Build recovery separately....111
4.4.5 Package into update.img image....112
4.5 SDK cleanup 113
4.6 Mirror Introduction 113
Chapter 5. SDK image burning....115
5.1 Flash Writing Mode Introduction....116
5.2 Flash Writing under Windows System....116
5.2.1 Introduction to Partition Table parameter.txt....117
5.2.2 Configuration 119
5.2.3 Flashing....120
5.2.4 Booting the System 122
5.2.5 Flashing update.img....122
5.3 Flashing in Ubuntu system....123
5.3.1 Connect the development board to Ubuntu....123
5.3.2 Using the upgrade_tool tool for burning....125
5.3.3 Flashing update.img....127
5.3.4 Erase Operation....127
5.3.5 Writing using the rkflash.sh script 128
Chapter 6. SDK Development 130
6.1 SDK Board Configuration File 131
6.2 U-Boot Development 134
6.2.1 The device tree of U-Boot....135
6.2.2 U-Boot Compilation....135
6.2.3 Configuration of U-Boot....152
6.2.4 The defconfig configuration file 153
6.2.5 Shortcuts 155
6.2.6 HW-ID DTB 156

6.3 kernel development 159
6.3.1 Kernel compilation 159
6.3.2 Kernel Configuration 168
6.3.3 Kernel Device Tree and defconfig Configuration File 169
6.3.4 IO Power Domain 173
6.3.5 Replace the logo 180
6.3.6 Summary of Kernel Module Development Documents 181
6.3.7 Three-screen Display 184
6.4 Buildroot Development 184
6.4.1 Introduction to the buildroot directory structure 184
6.4.2 Common Compilation Commands 186
6.4.3 Compile the root file system image rootfs.img for RK3568 187
6.4.4 Configuration of rootfs 190
6.4.5 Output Directory Introduction 190
6.4.6 Package compilation 191
6.4.7 Add package 193
6.4.8 Customization of rootfs 198
6.5 Recovery development 198
6.5.1 Compile recovery 198
6.5.2 Configuration of recovery 203
6.5.3 Reference Documents 203
6.6 OEM and Userdata 203
6.7 The parameter.txt partition table file 205
6.8 misc.img image 205
6.9 Qt Application Development 210
6.10 Other application development 210
6.11 Yocto Development 212
6.12 NPU Development 212
6.13 Debian Development 213
6.14 Multimedia Development 213
6.15 Graphics Development 213
6.16 Development of Security Mechanisms 214
6.17 A/B System Development 214
6.18 System Upgrade Development 214

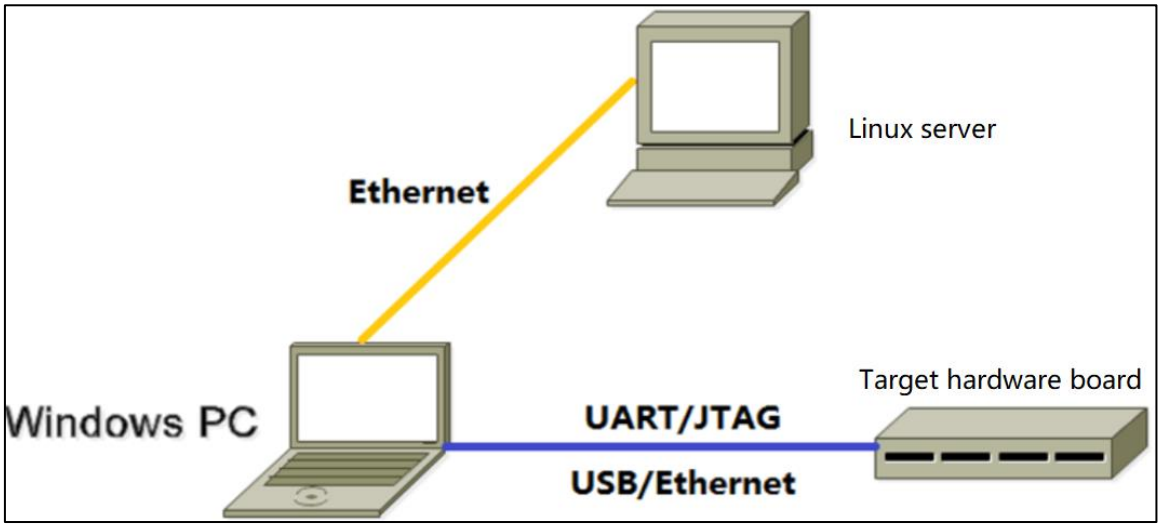
Brief

This document will provide users with information on the development of embedded Linux systems on the RK3568 platform, covering the following sections:

1. Installing the Ubuntu operating system,
2. Setting up the development environment,
3. Introduction to the ALIENTEK RK3568 development platform,
4. RK3568 Linux SDK software package.

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 downloading 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 Linux development is usually carried out in a dual-system development environment of Windows and Ubuntu. This chapter introduces 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 Install VMware virtual machine software

VMware (Vmware Workstation) is a very 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 that simulates the complete hardware system functions (such as memory, hard disk, processor, etc.) of an actual computer through software. Various operating systems supported by the actual machine, such as Windows operating system, DOS, Linux operating system (Ubuntu, Debian, CentOS, etc.), can be installed in the virtual machine. The tasks that can be accomplished in the actual computer can also be realized in the virtual machine, and the virtual machine can be operated just like an actual machine.

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

1.1.1 Download 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 optical disc 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 1.1.1.1 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.

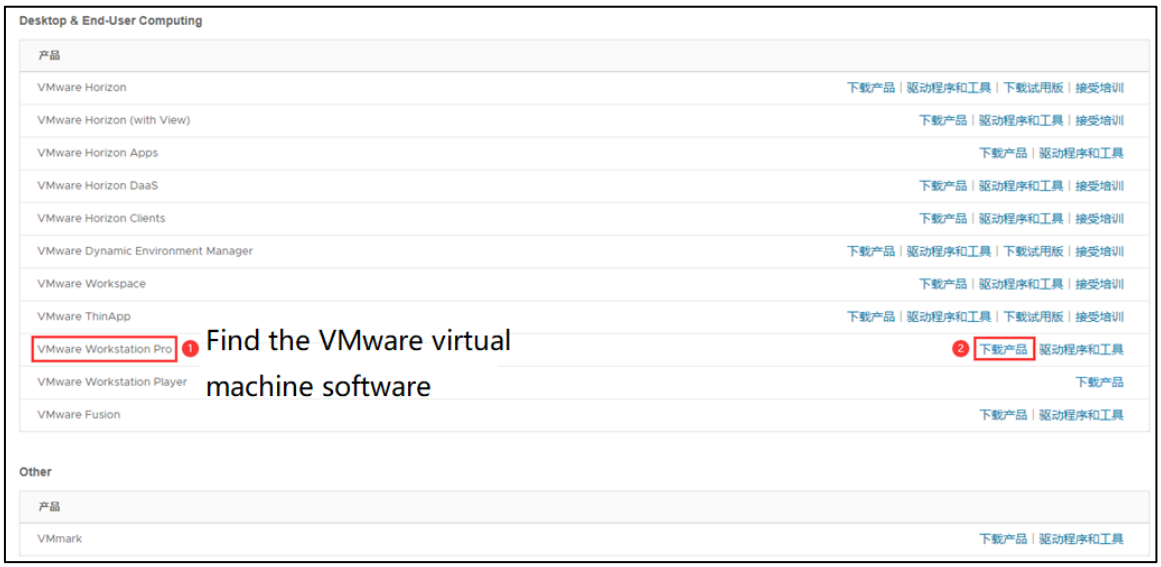


Figure 1.1.1.2 Download of the old version VMware virtual machine software installation package 1

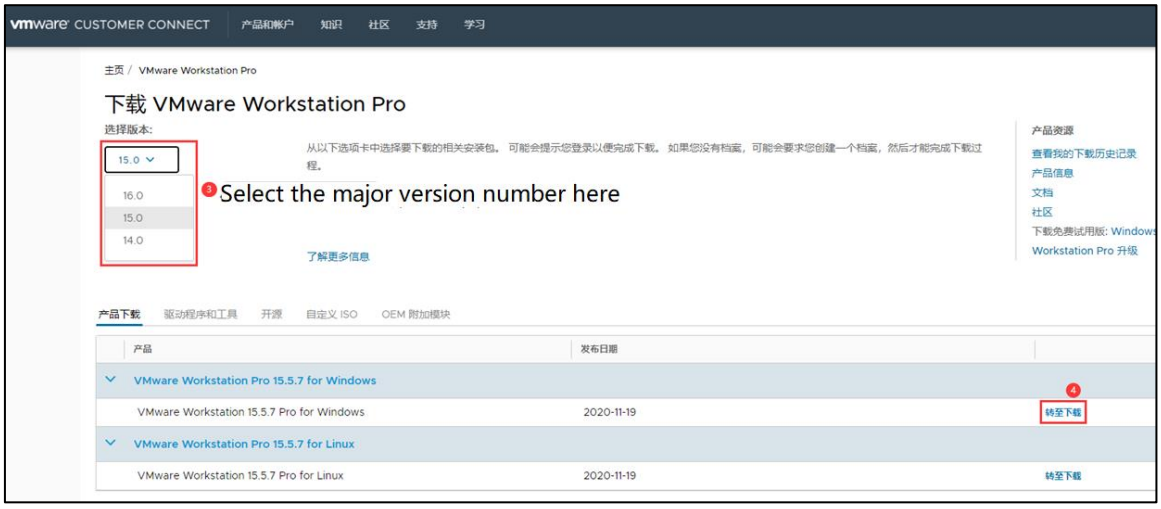


Figure 1.1.1.3 Download of the old version VMware virtual machine software installation package 2

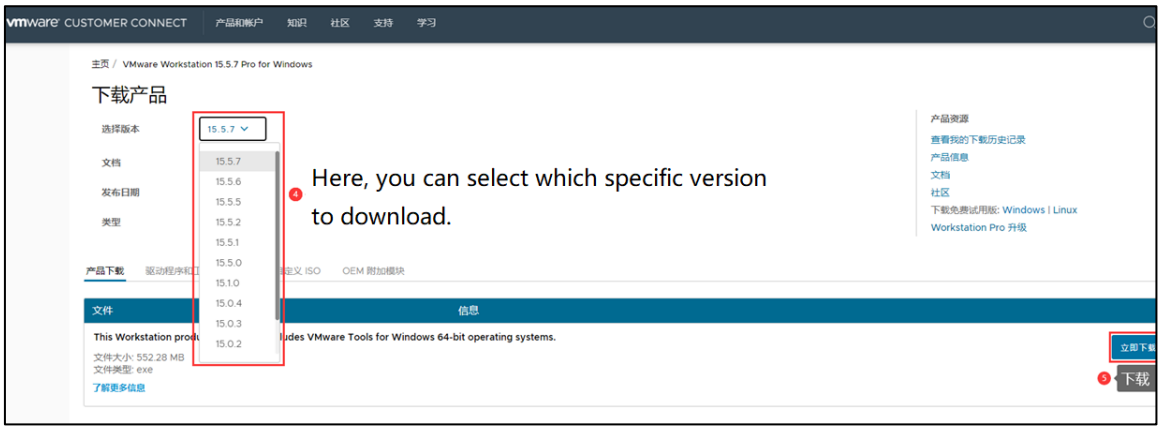


Figure 1.1.1.4 Download of the old version VMware virtual machine software installation package 3

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.

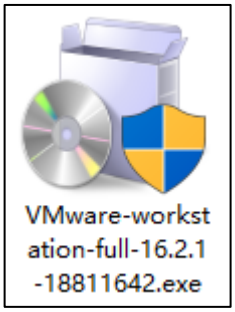


Figure 1.1.1.5 16.2.1 version of VMware virtual machine software installation package file

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:

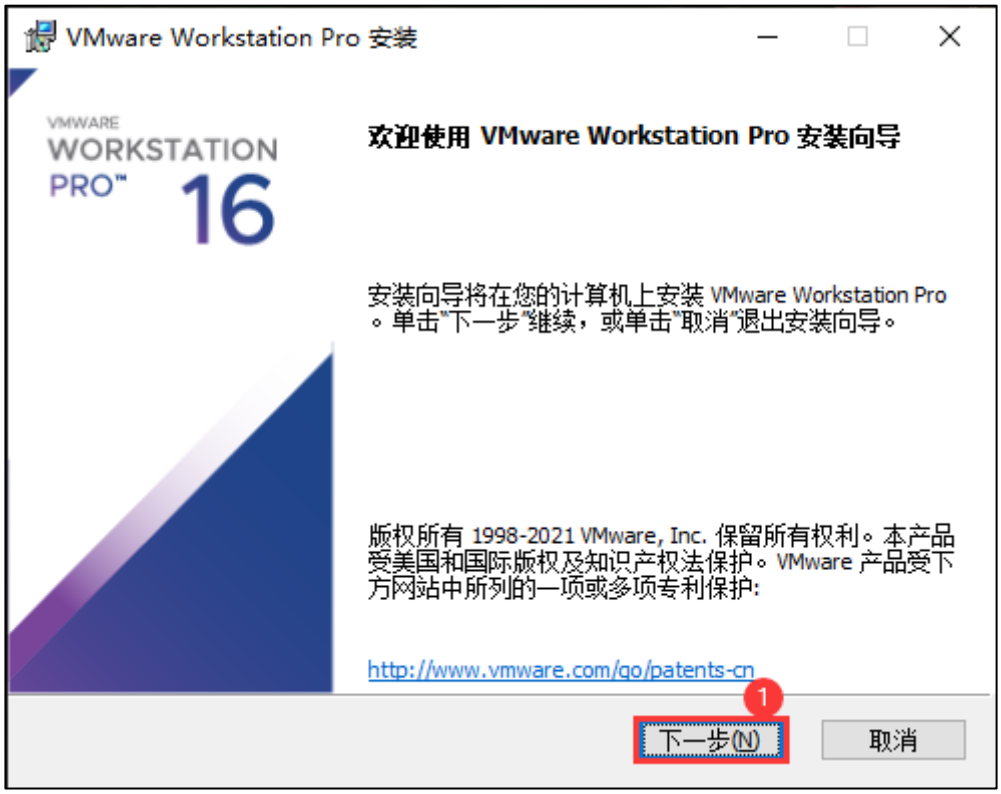


Figure 1.1.2.1 Install VMware software 1

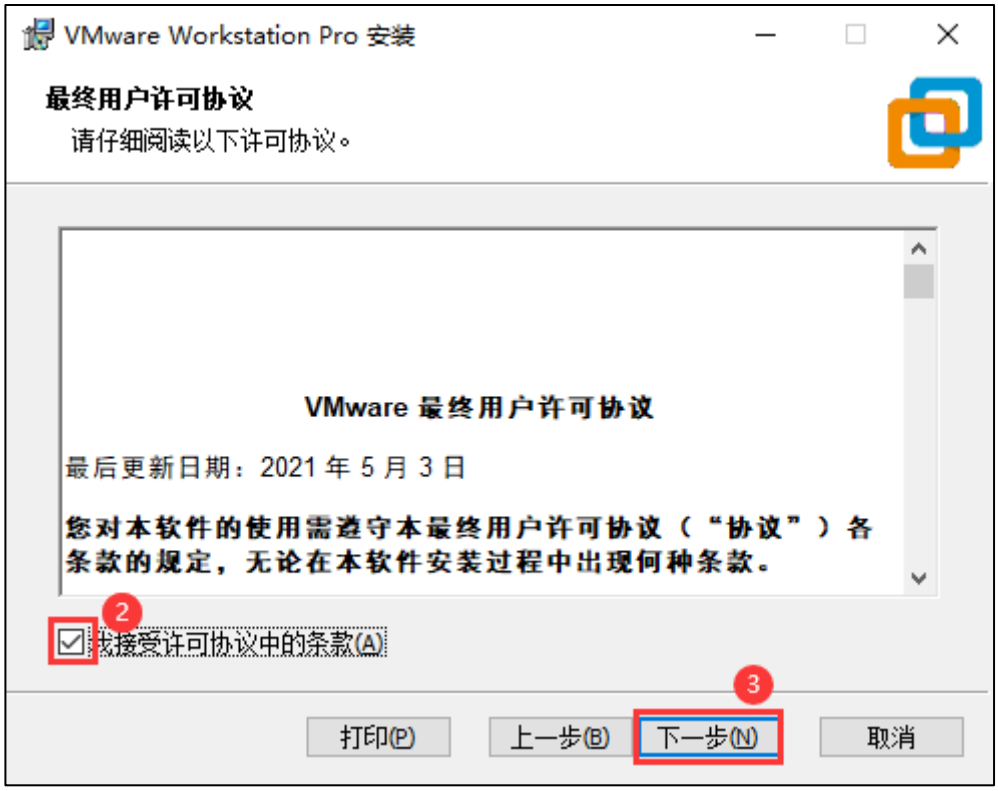


Figure 1.1.2.2 Install VMware software 2

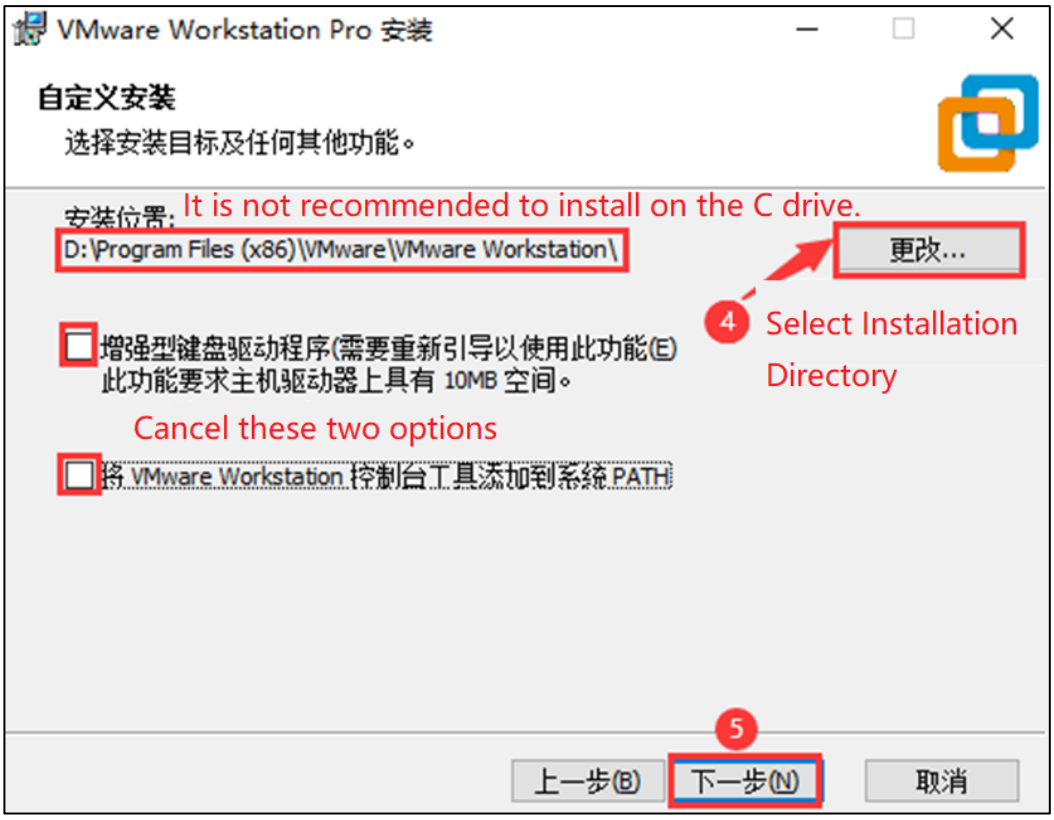


Figure 1.1.2.3 Install VMware software 3

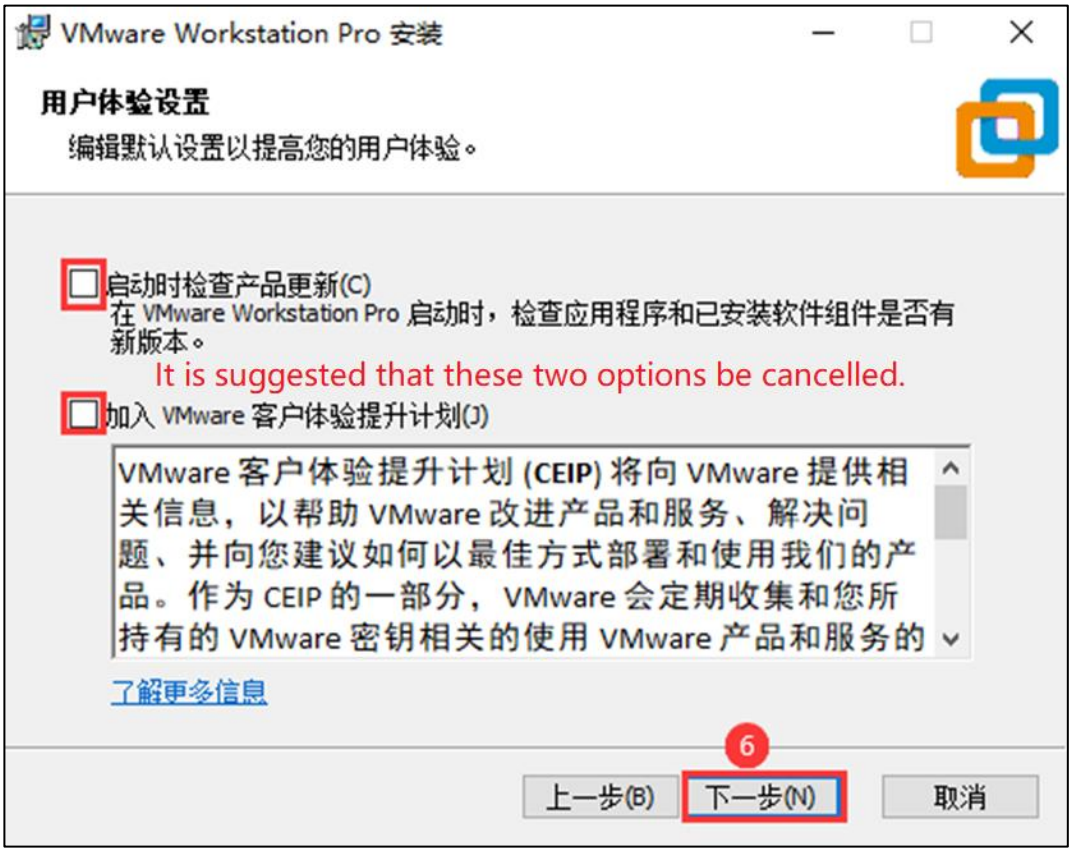


Figure 1.1.2.4 Install VMware software 4

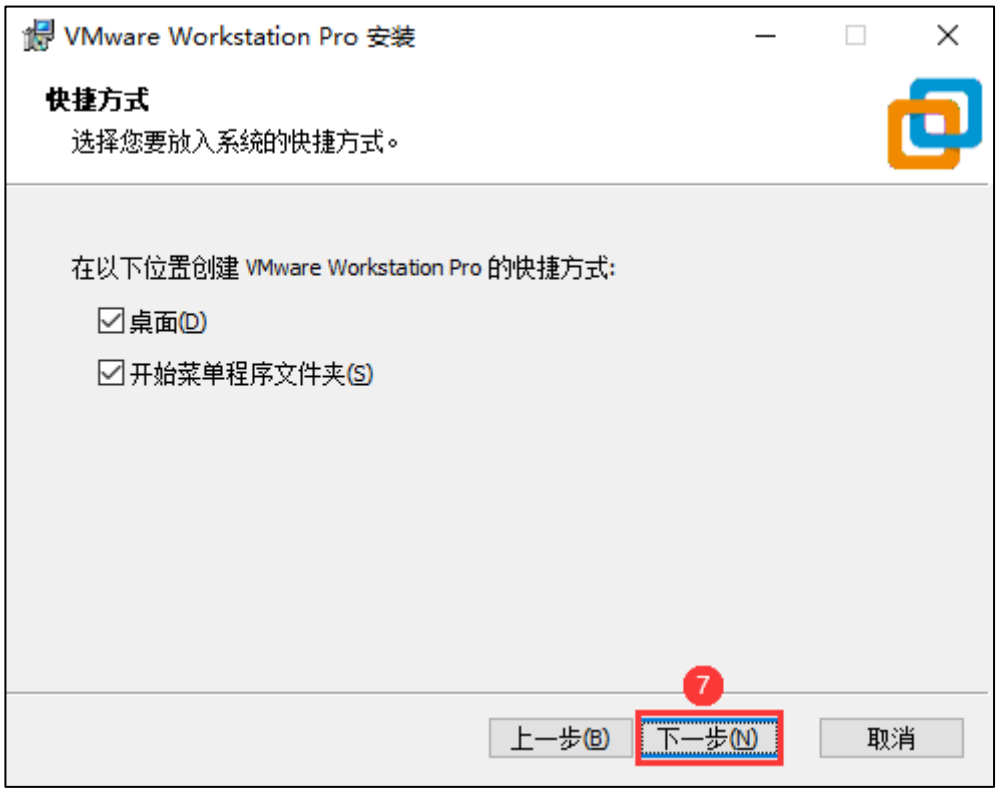


Figure 1.1.2.5 Install VMware software 5

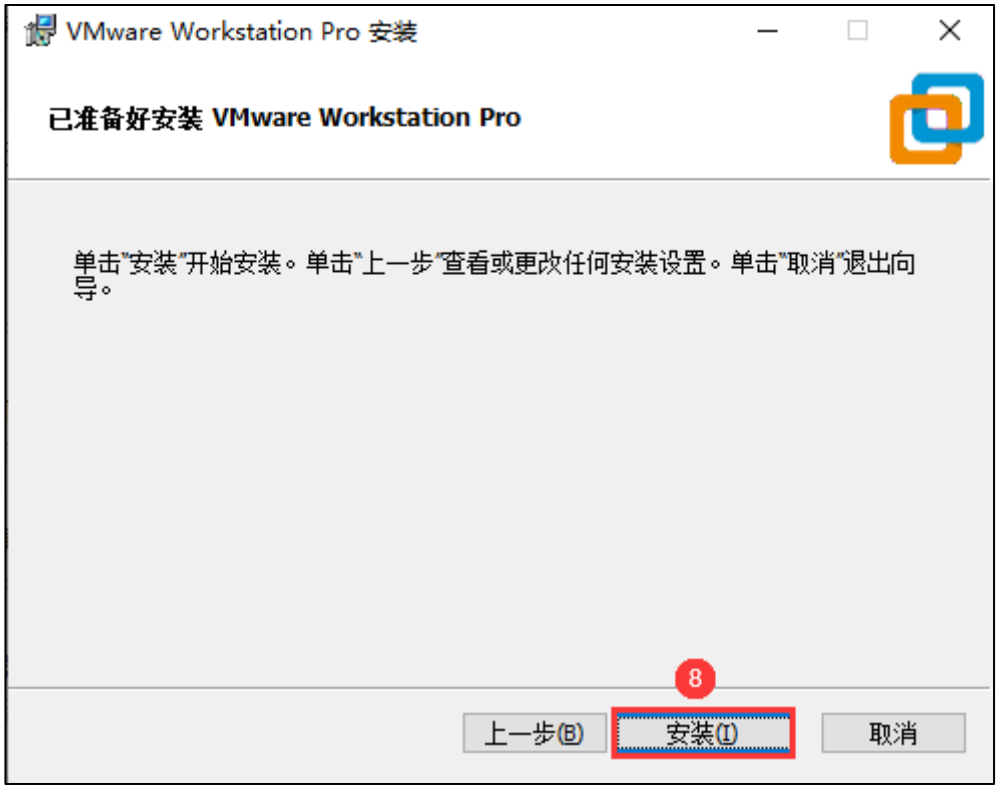


Figure 1.1.2.6 Install VMware software 6

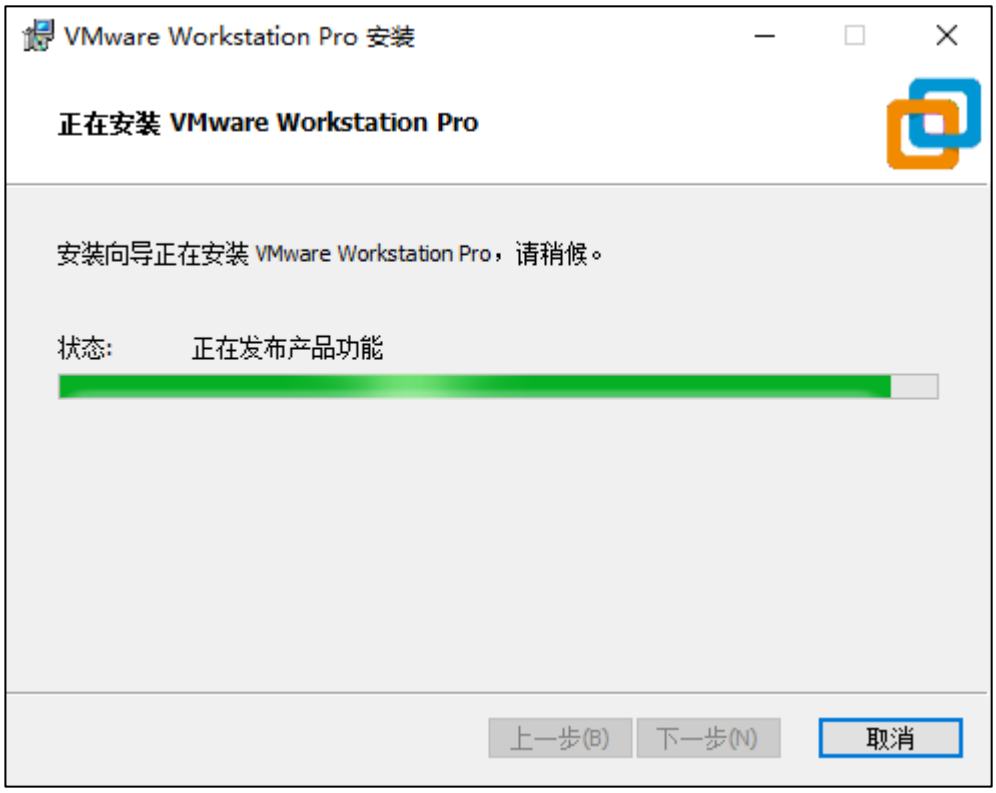


Figure 1.1.2.7 Install VMware software 7

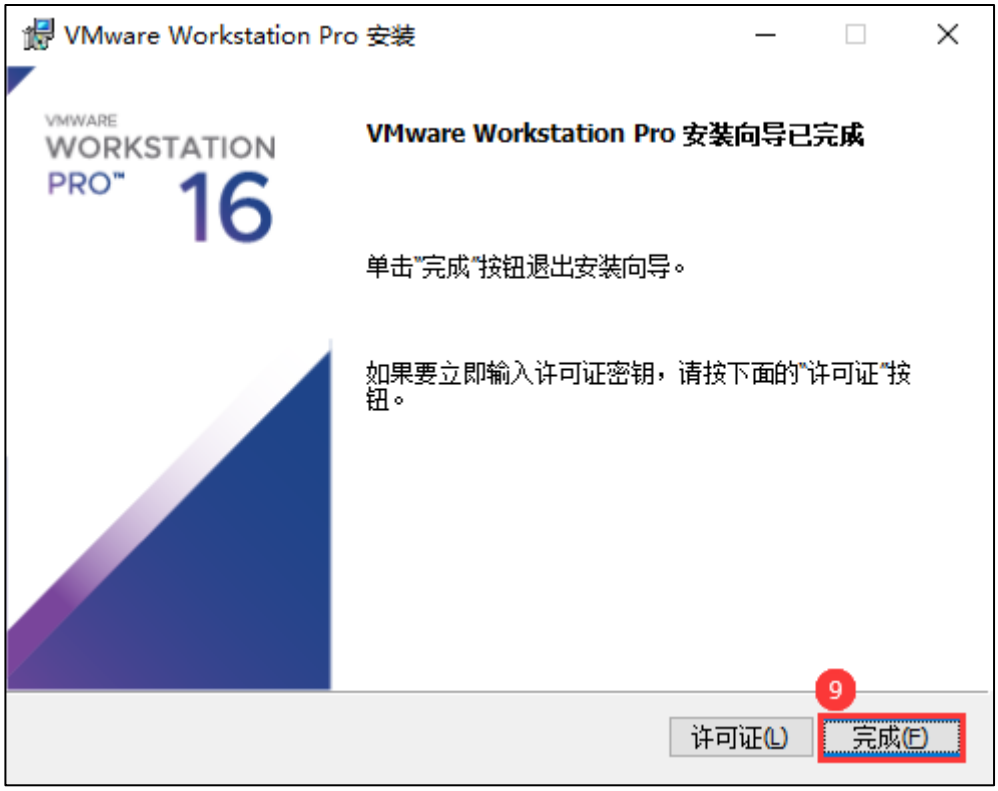


Figure 1.1.2.8 Install VMware software 8

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

After the installation is completed, a shortcut icon for 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 has purchased the VMware software usage 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 usage 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:

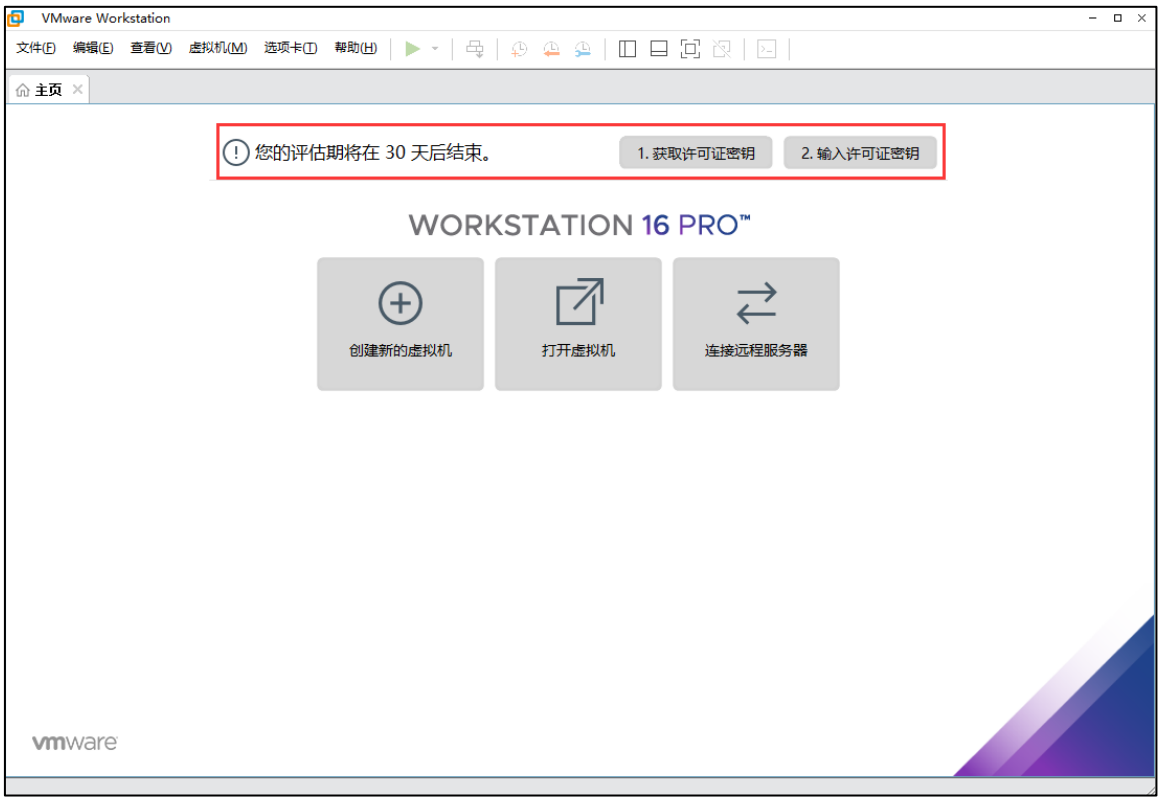


Figure 1.1.2.12 VMware virtual machine software main interface

1.2 Installing the Ubuntu system

This section introduces to users how to install the Ubuntu operating system in the VMware virtual machine. It is recommended 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 as an example. For users using Ubuntu 18.04, they can refer to this document for comparison.

1.2.1 Downloading the Ubuntu system image

The development board data 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 04, Software ubuntu-18.04.6-desktop-amd64.iso and Development Board CD-ROM A Disk - **Basic Materials→4_softwares→ubuntu-20.04.2-desktop-amd64.iso**. If users do not want to waste time downloading, they can directly use the Ubuntu system image provided in the data package and proceed to the next 1.2.2 subsection.

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

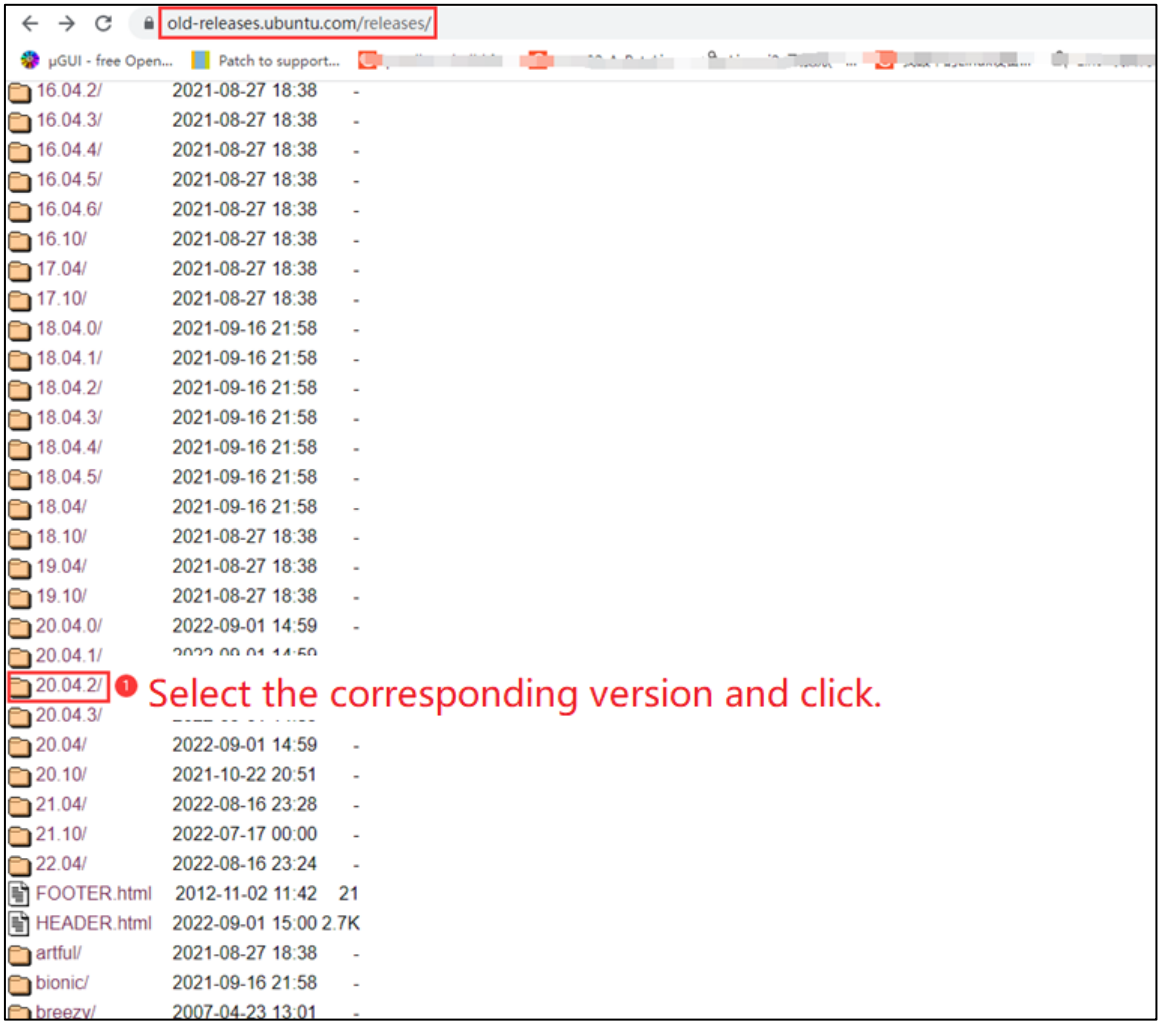


Figure 1.2.1.1 Download the Ubuntu system image 1

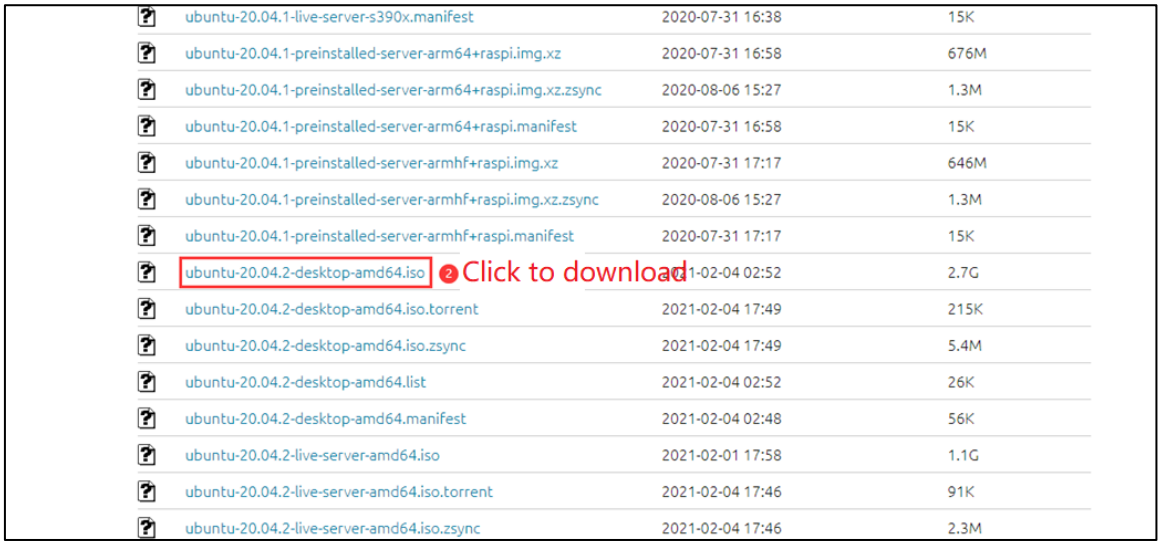


Figure 1.2.1.2 Download the Ubuntu system image 2

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:

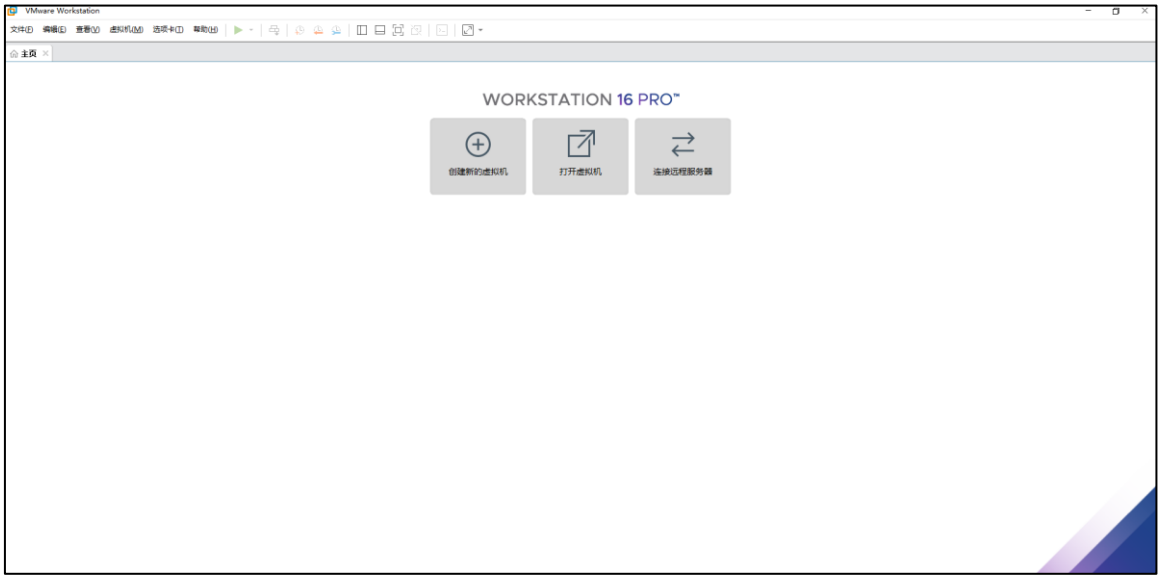


Figure 1.2.2.1 VMware virtual machine software

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

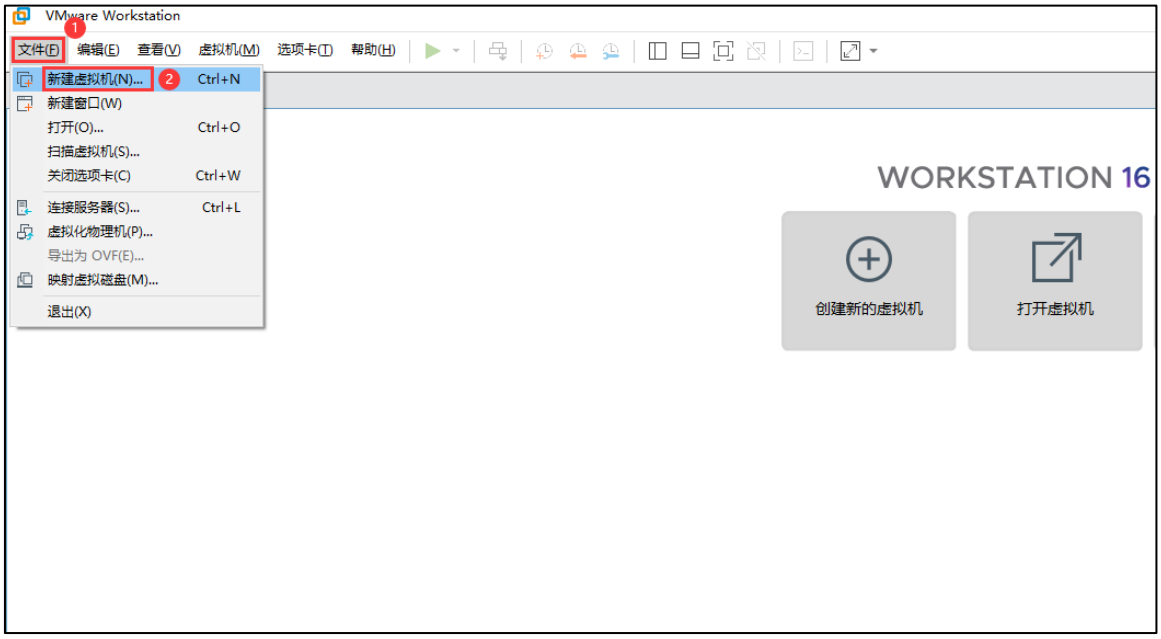


Figure 1.2.2.2 Create a virtual machine (1)

Figure 1.2.2.3 Create a virtual machine (2)

Figure 1.2.2.4 Create a virtual machine (3)

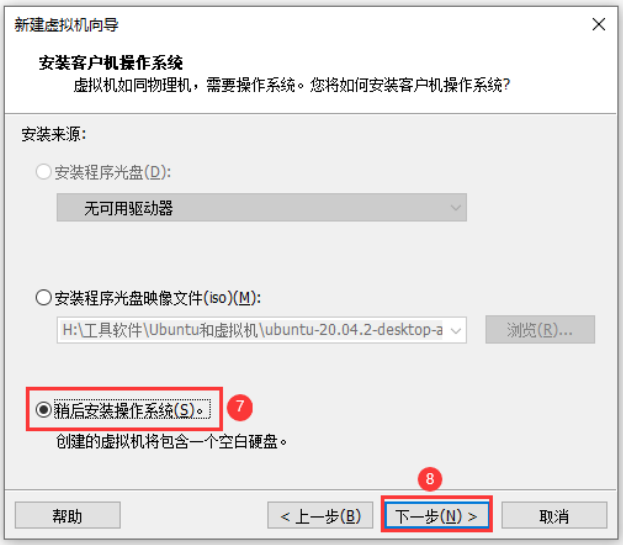


Figure 1.2.2.5 Create a virtual machine (4)

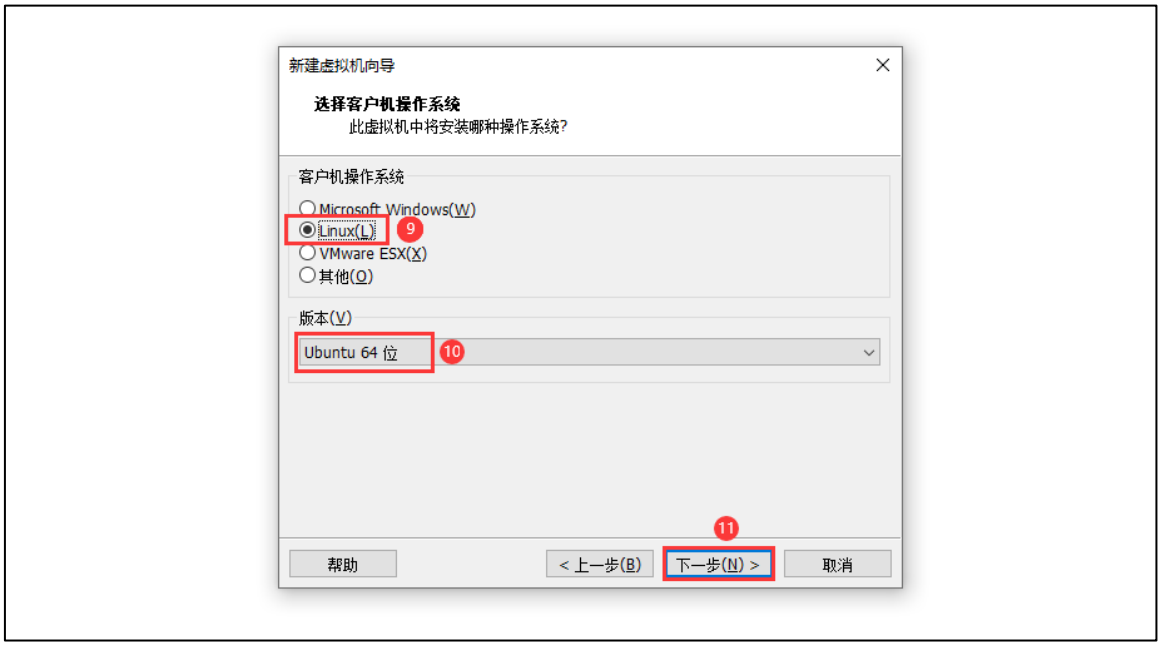


Figure 1.2.2.6 Create a virtual machine (5)

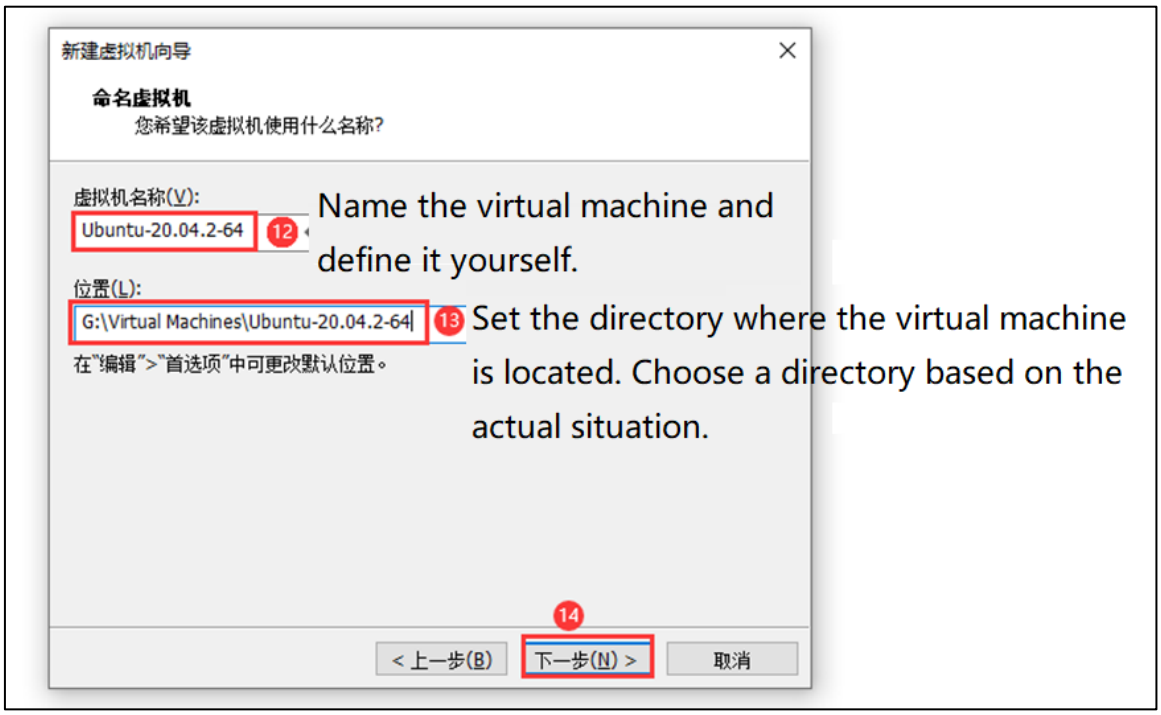


Figure 1.2.2.7 Create a virtual machine (6)

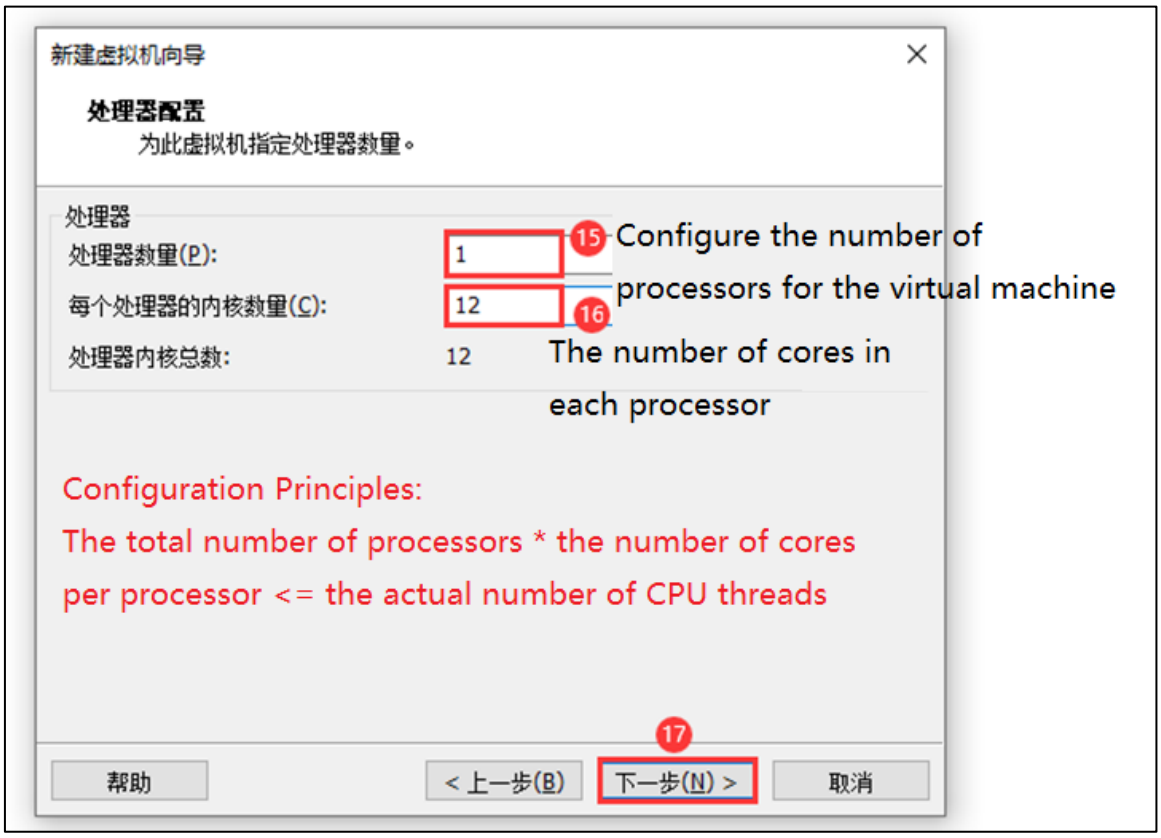


Figure 1.2.2.8 Create a virtual machine (7)

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:

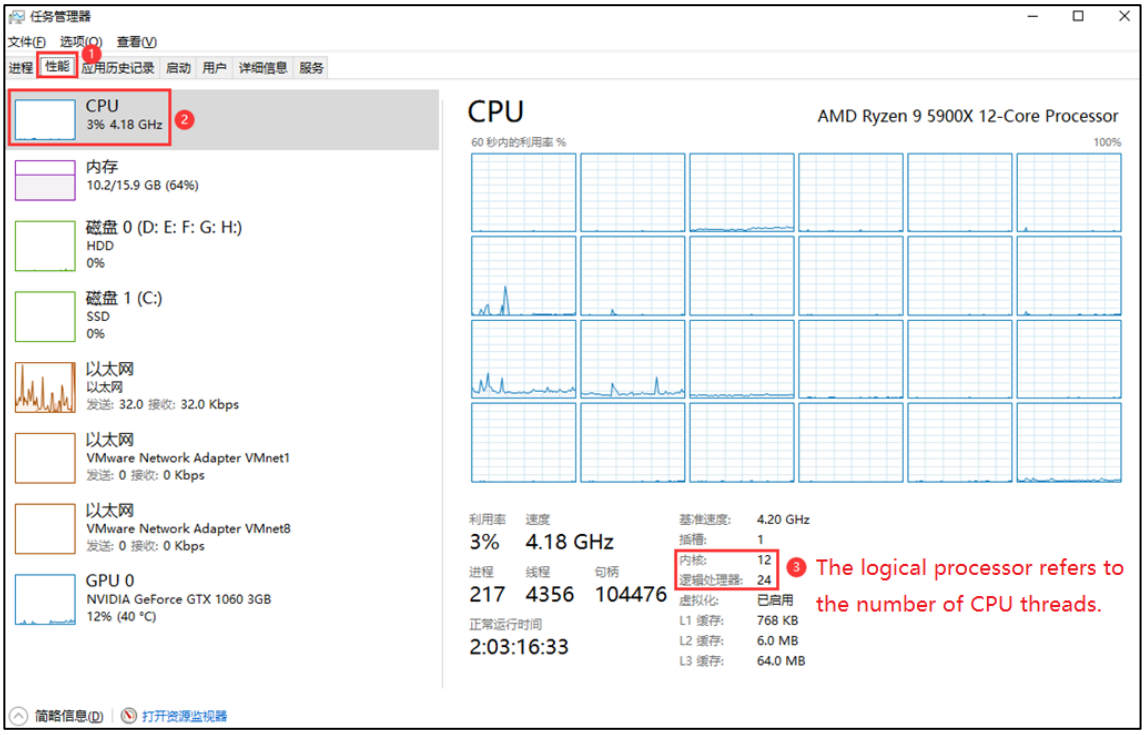


Figure 1.2.2.9 Query the CPU configuration

The number of logical processors shown in the figure is the actual number of threads of the CPU. Therefore, the actual number of threads of my computer's CPU is 24. According to the configuration requirements, the total number of processor cores of the virtual machine should be less than or equal to (<=) the actual number of threads of the CPU, which is 24. Then, we can have various different configuration options. For example:

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;

And so on...

In summary, the specific configuration options are diverse. As long as the requirements are met, it is fine!

Of course, if conditions permit (for example, not affecting the normal use of the physical machine's Windows system), it is advisable to make the total number of processor cores of 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!

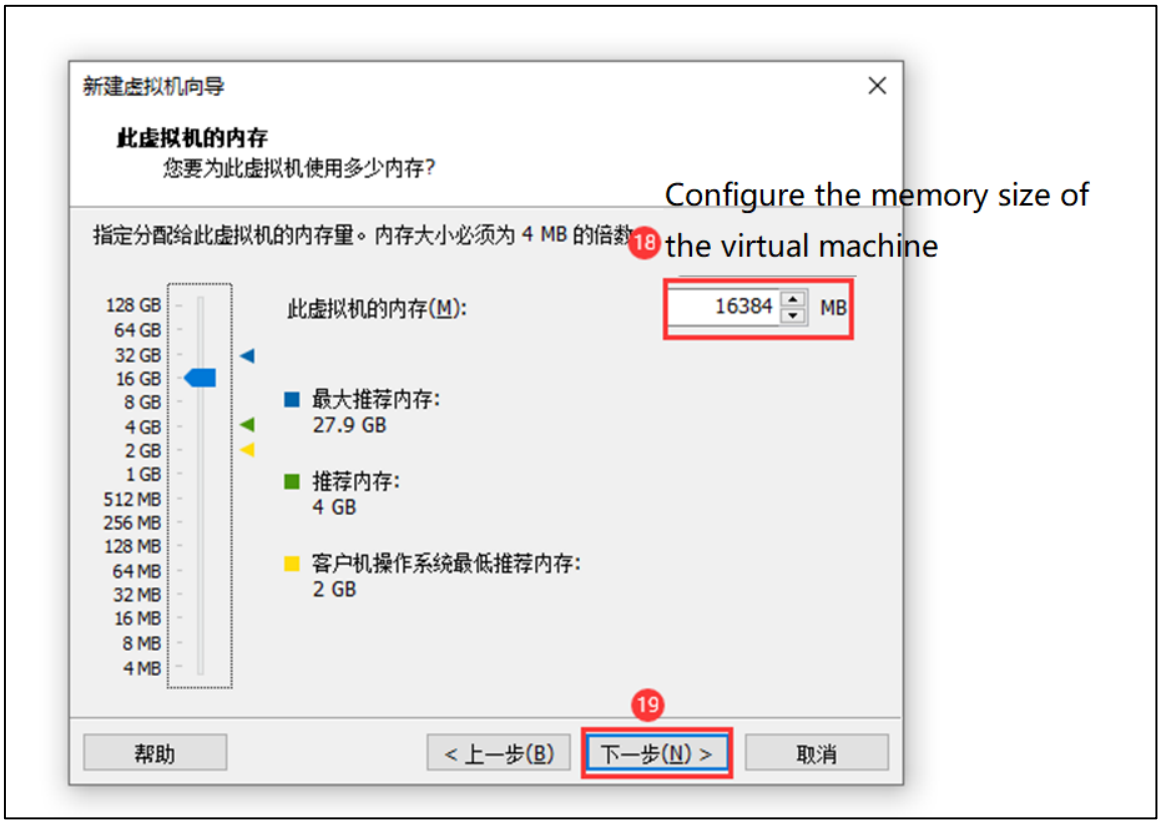


Figure 1.2.2.10 Creating Virtual Machine (8)

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. The unit is 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 the embedded Linux 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.

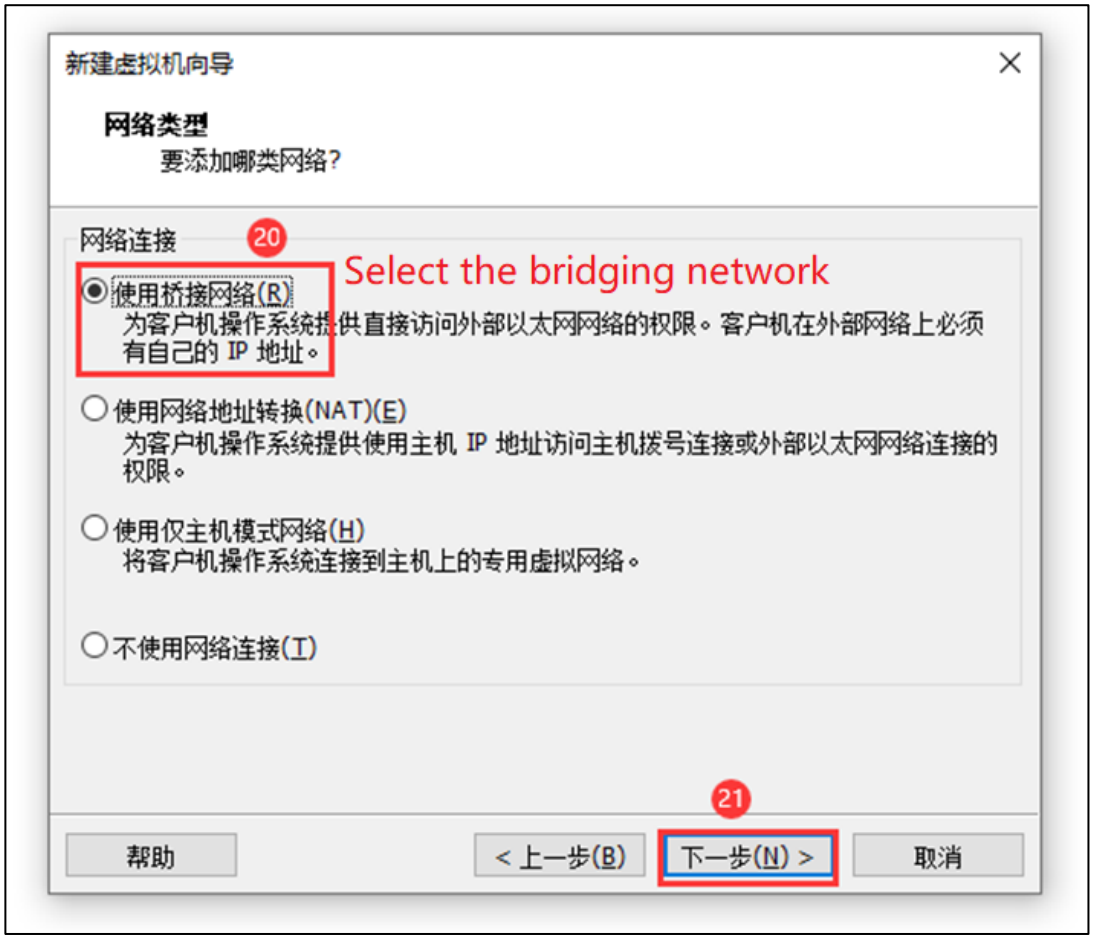


Figure 1.2.2.11 Create virtual machine (9)

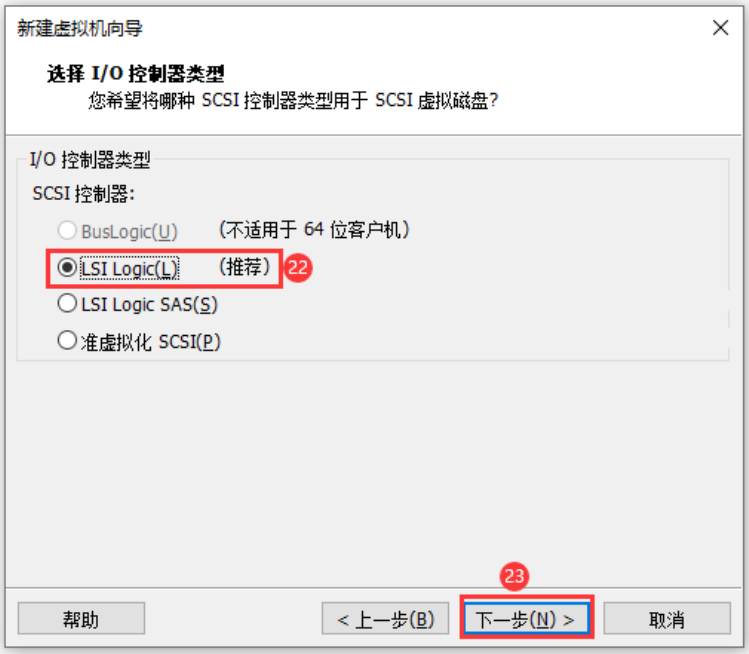


Figure 1.2.2.12 Create virtual machine (10)

Figure 1.2.2.13 Create virtual machine (11)

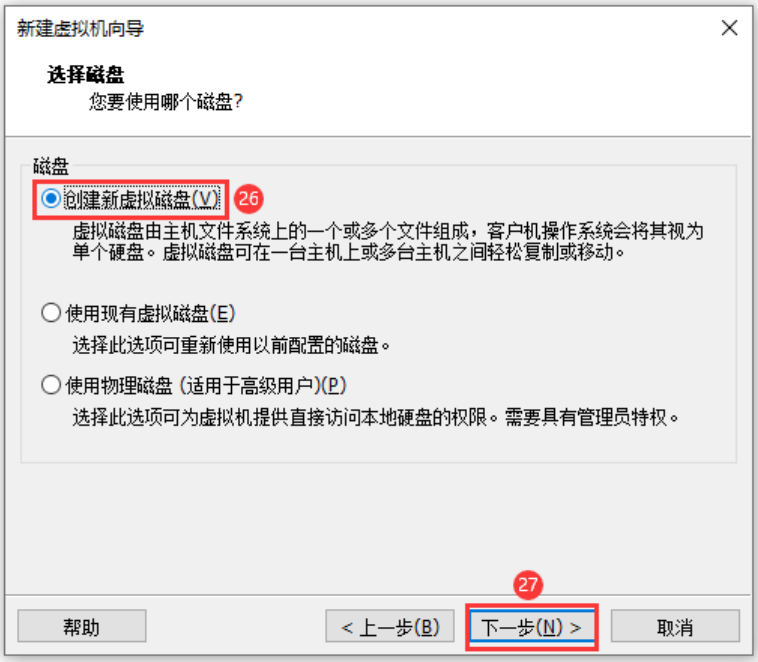


Figure 1.2.2.14 Create virtual machine (12)

Figure 1.2.2.15 Create virtual machine (13)

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

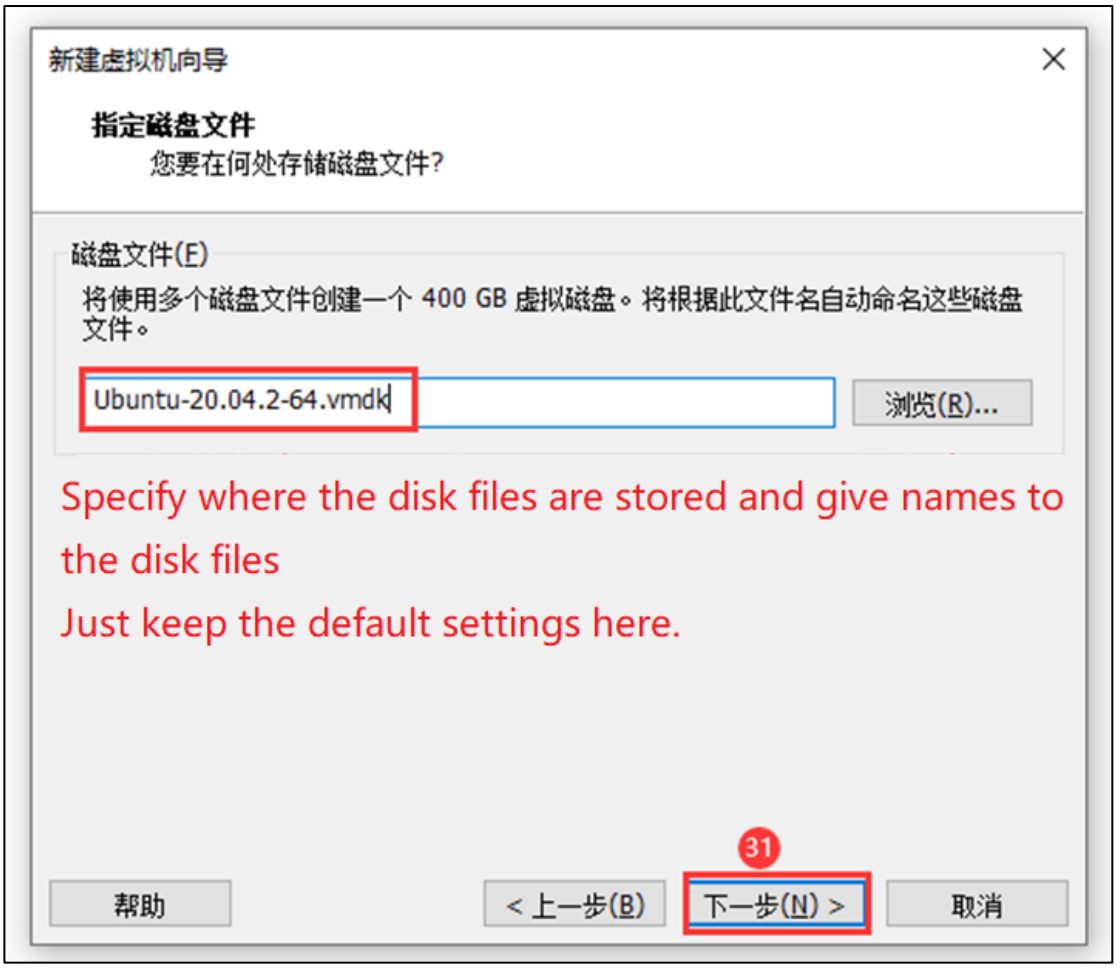


Figure 1.2.2.16 Create virtual machine (14)

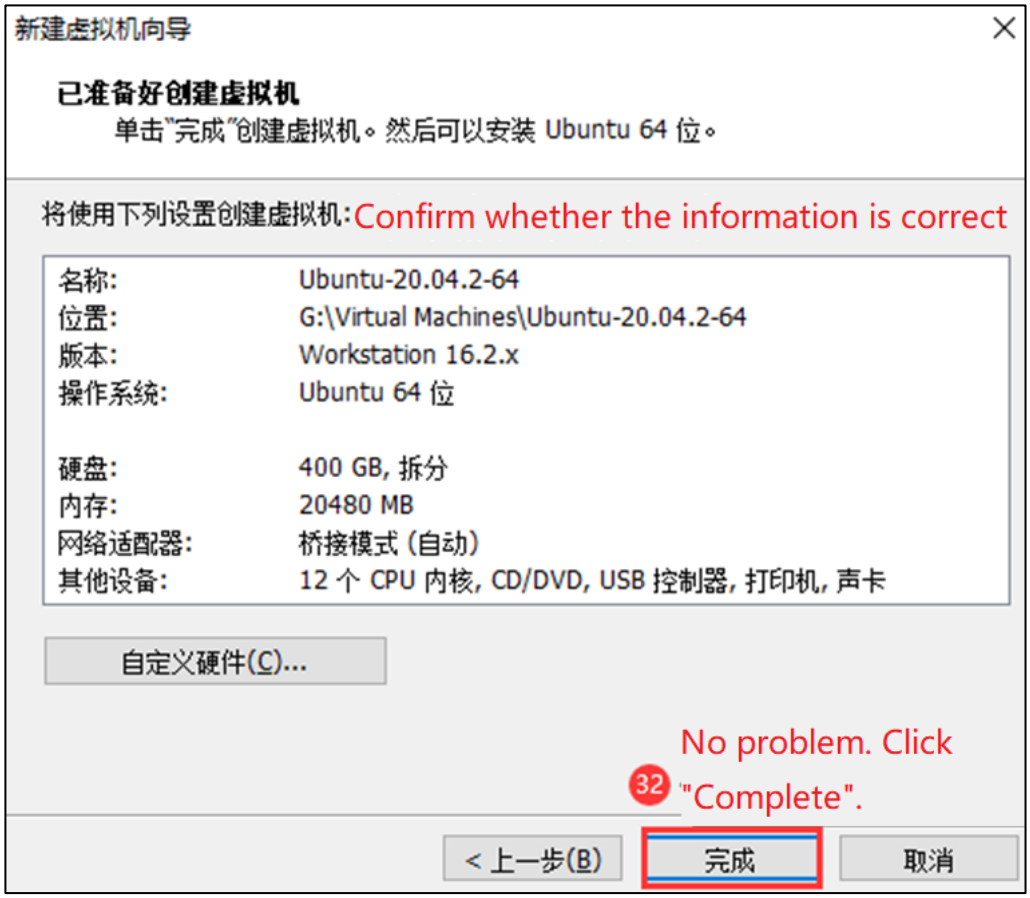


Figure 1.2.2.17 Create the virtual machine (15)

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

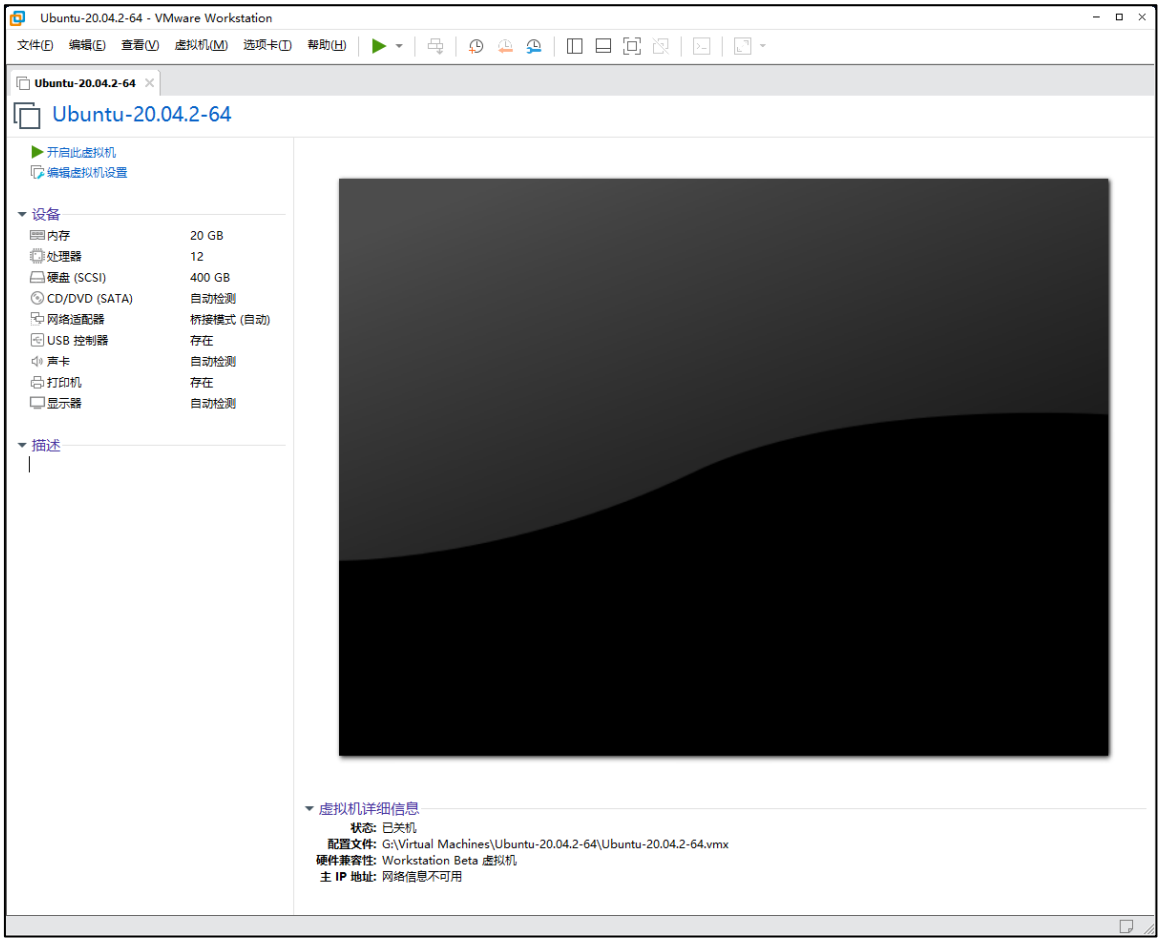


Figure 1.2.2.18 The virtual machine has been created successfully.

1.2.3 Installing the 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:

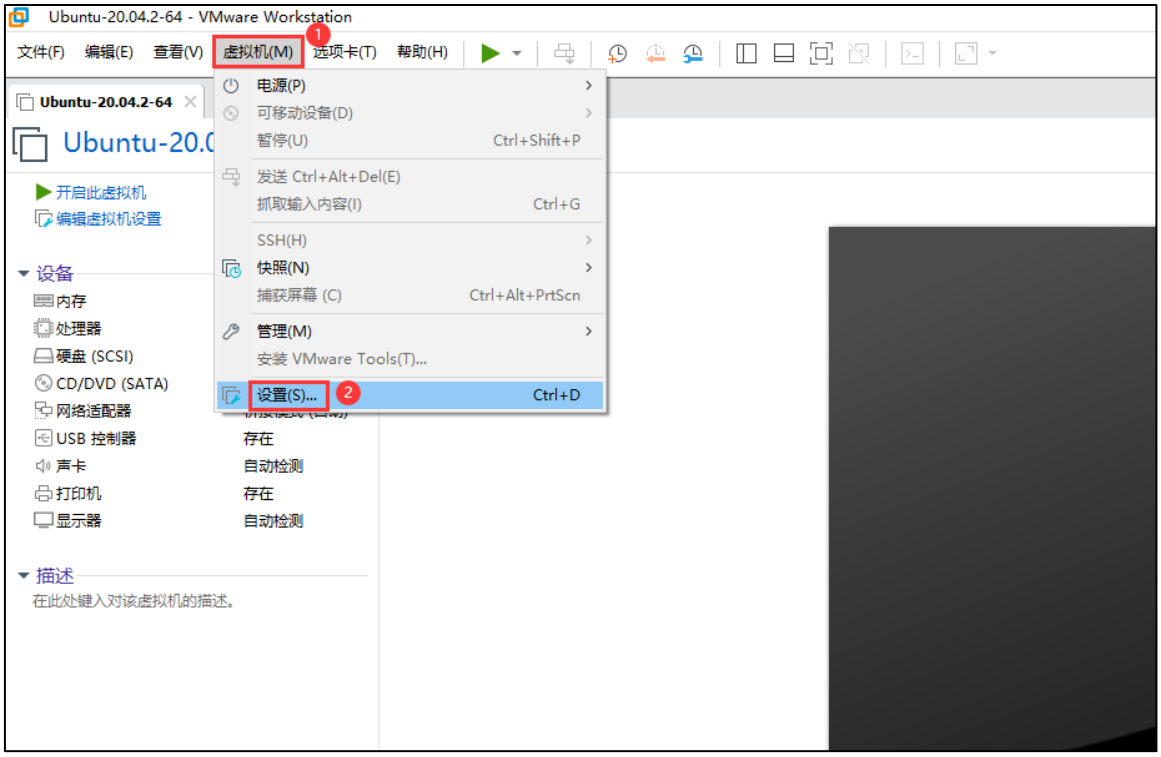


Figure 1.2.3.1 Open the virtual machine settings interface.

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

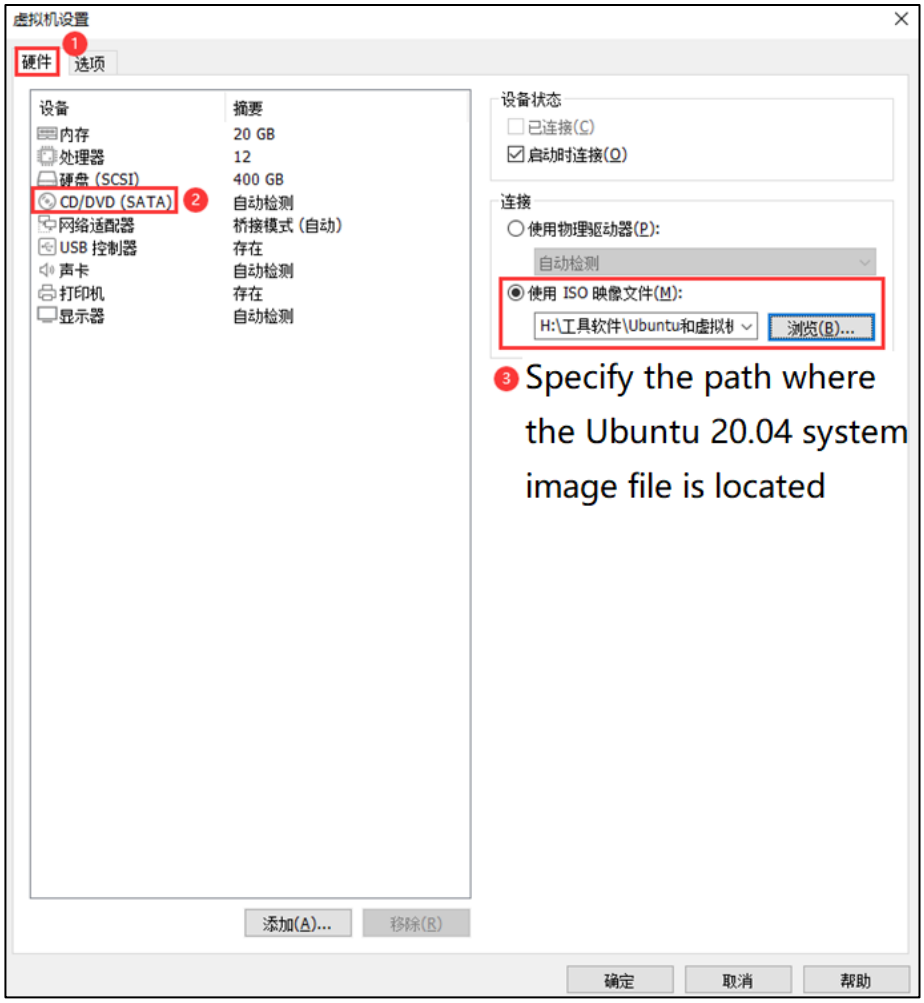


Figure 1.2.3.2 Specify the Ubuntu system image

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:

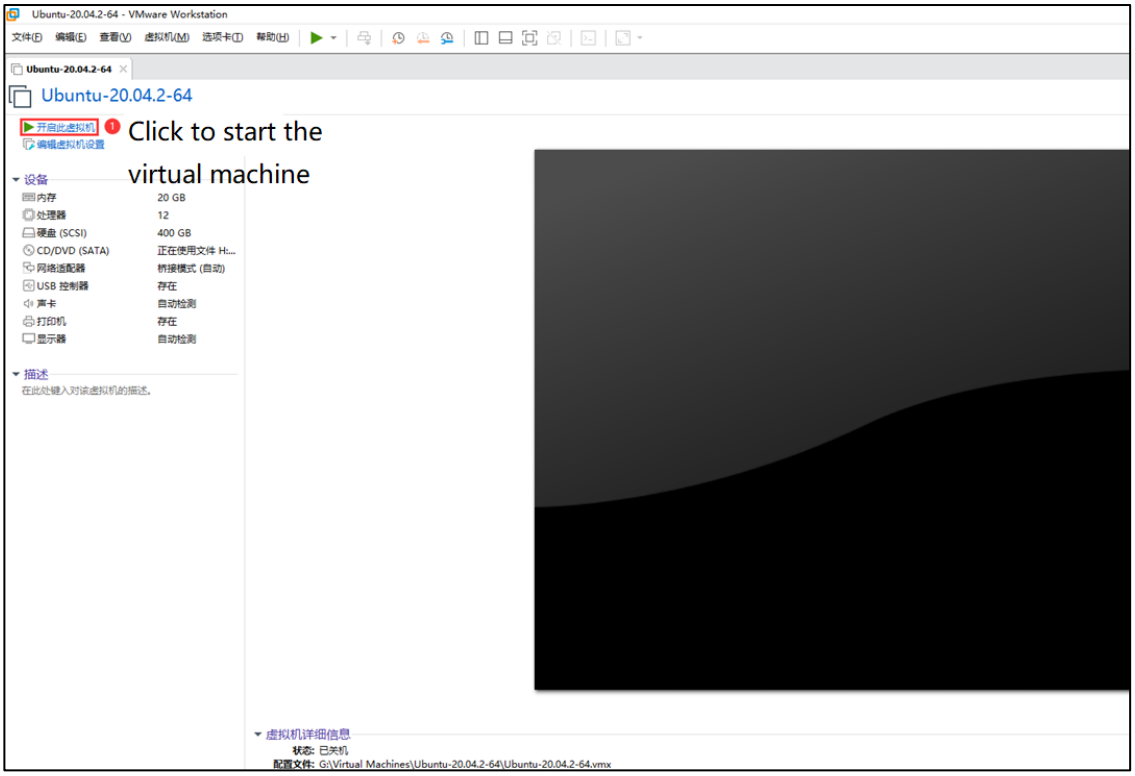


Figure 1.2.3.4 Install the Ubuntu operating system (1)

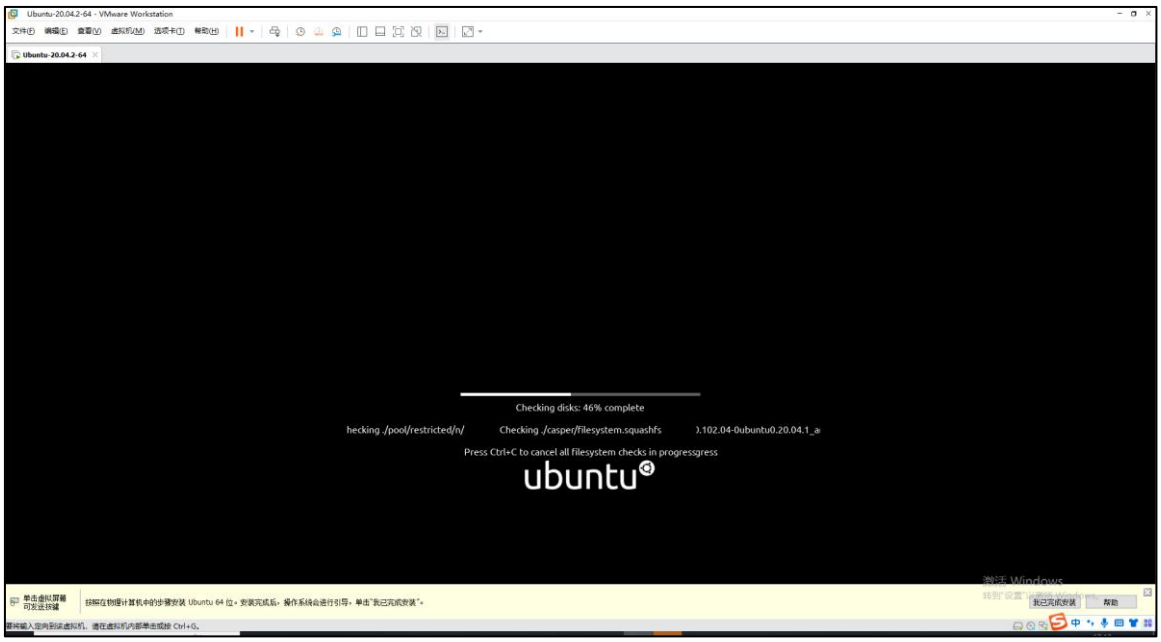


Figure 1.2.3.5 Install the Ubuntu operating system (2)

Figure 1.2.3.6 Install the Ubuntu operating system (3)

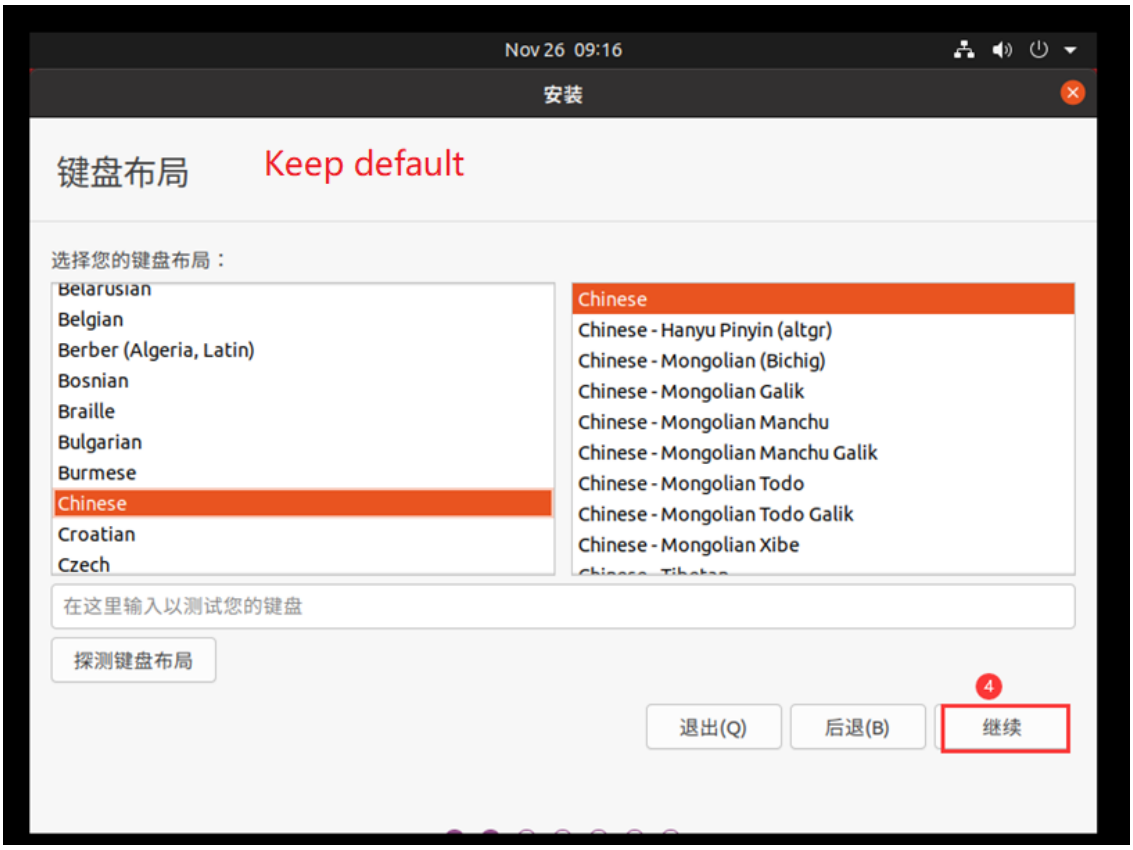


Figure 1.2.3.7 Installing Ubuntu Operating System (4)

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 them; 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.

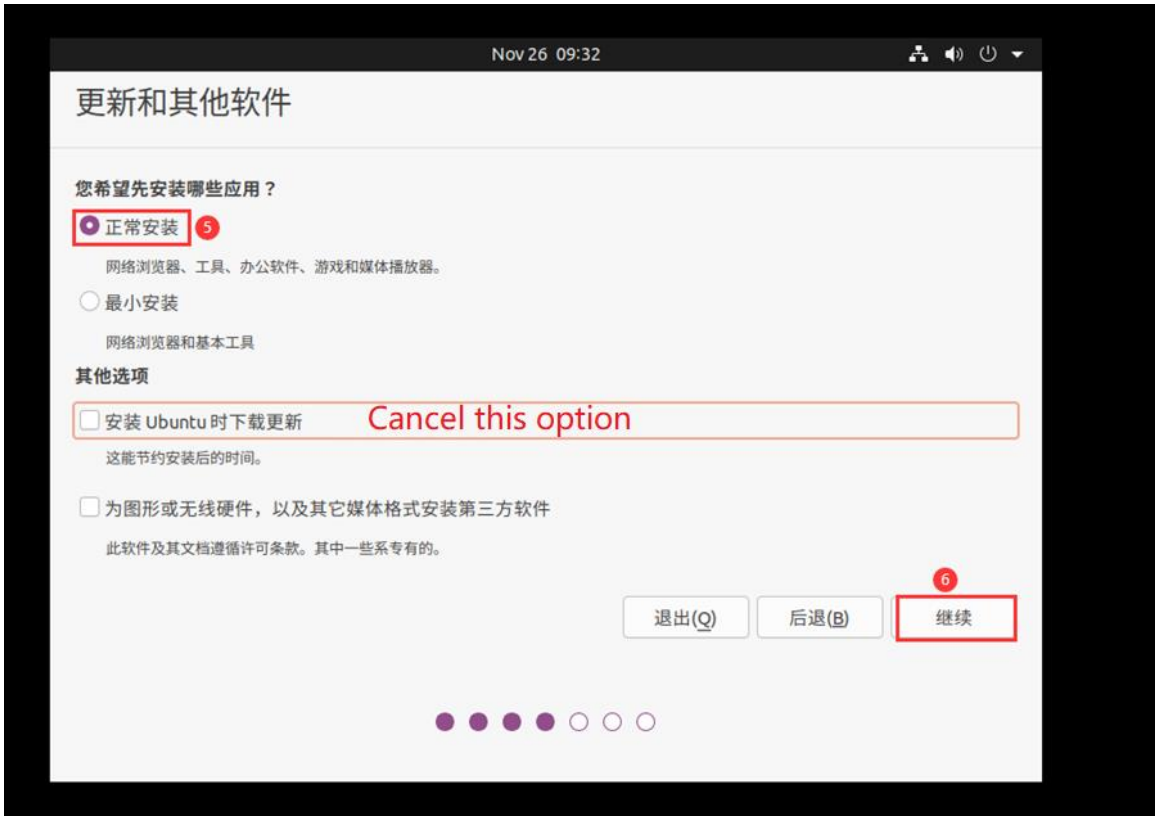


Figure 1.2.3.8 Install the Ubuntu operating system (5)

Figure 1.2.3.9 Install the Ubuntu operating system (6)

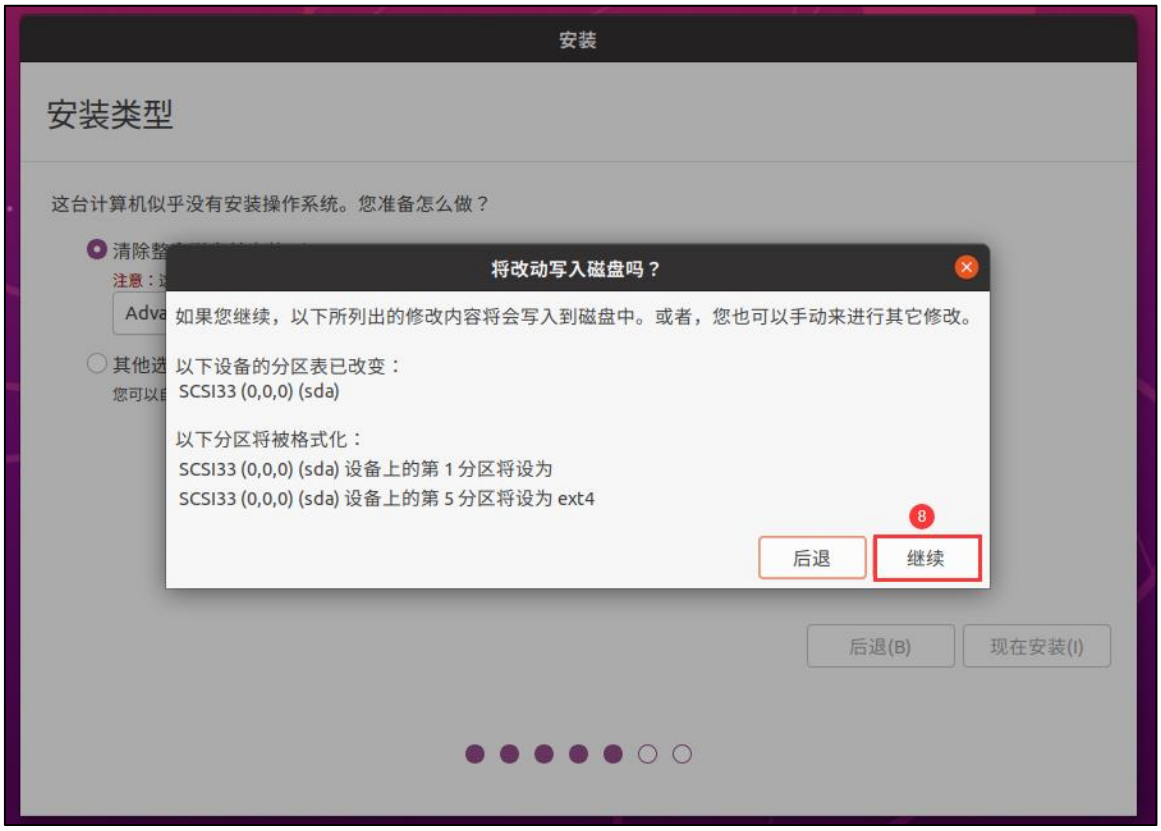


Figure 1.2.3.10 Install the Ubuntu operating system (7)

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.

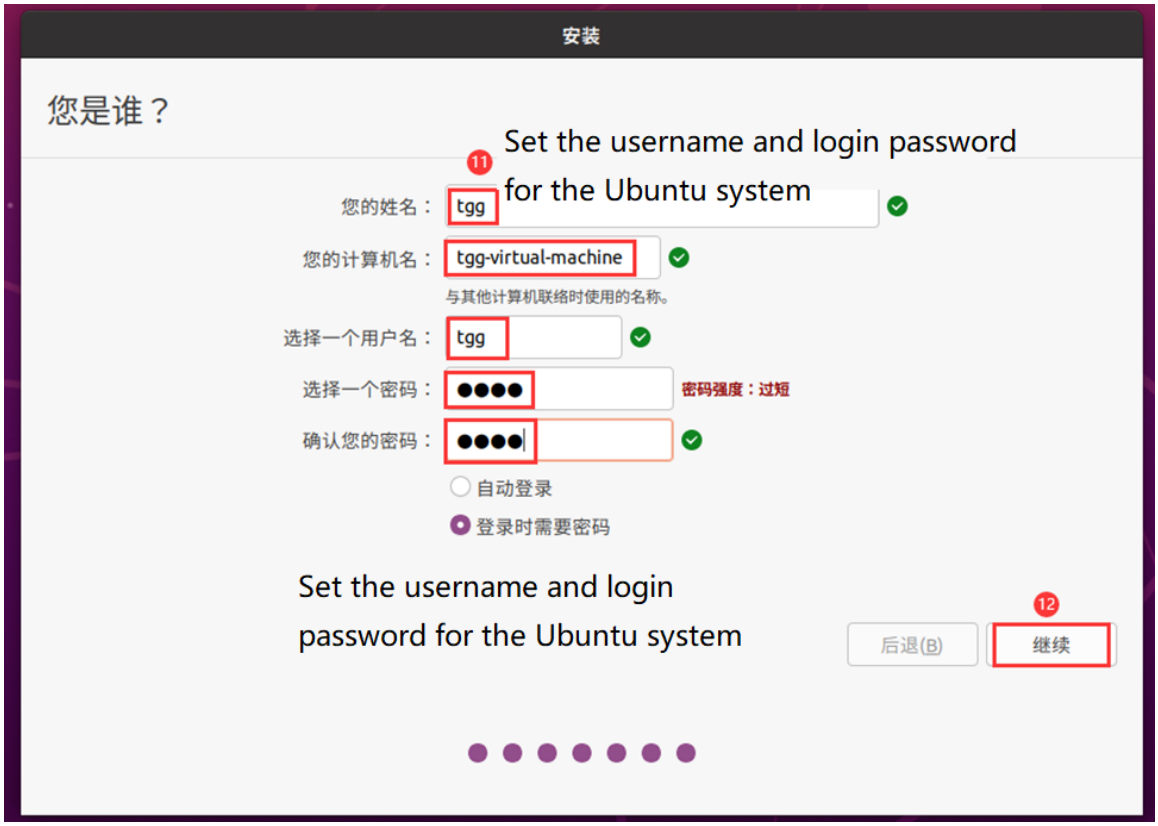


Figure 1.2.3.12 Install the Ubuntu operating system (9)

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

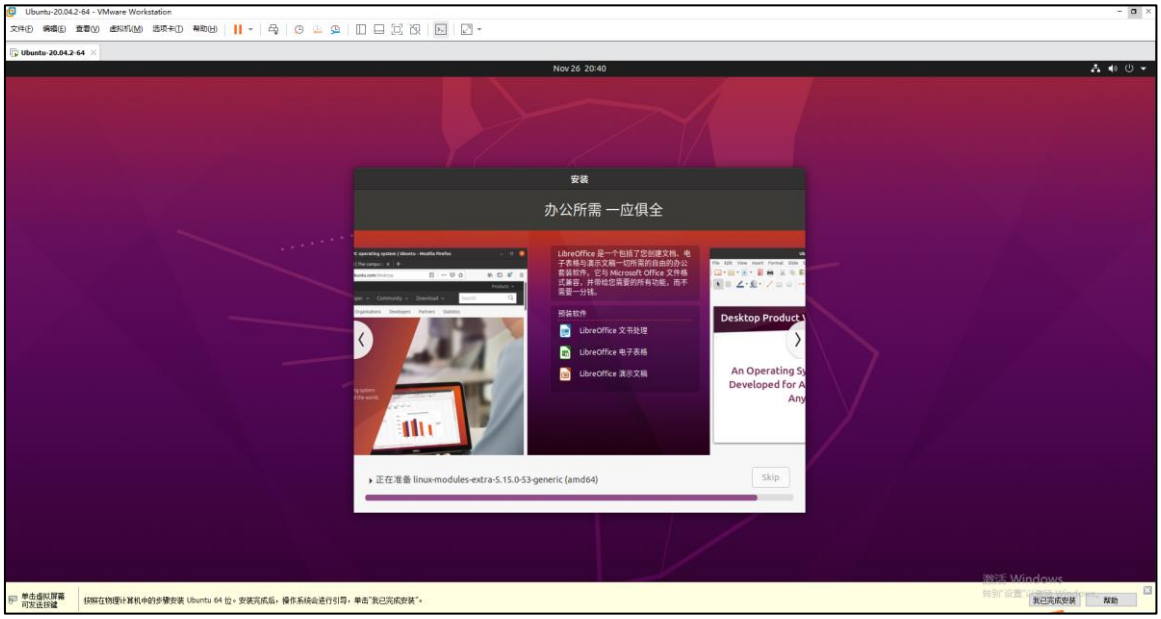


Figure 1.2.3.13 Install the Ubuntu operating system (10)

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:

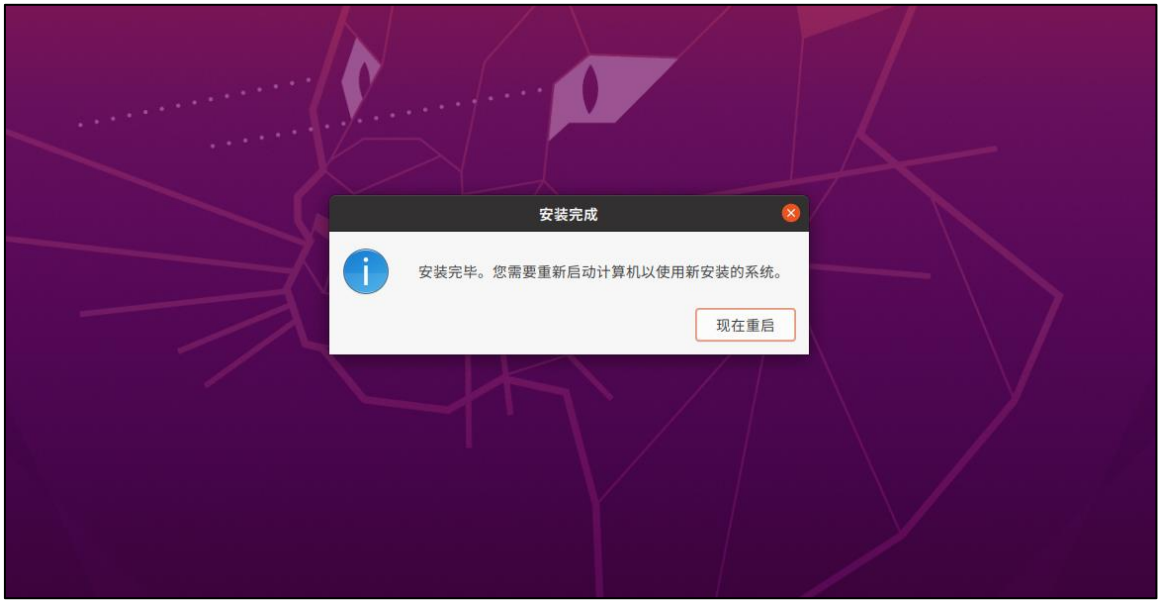


Figure 1.2.3.14 Ubuntu system installation is complete.

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:

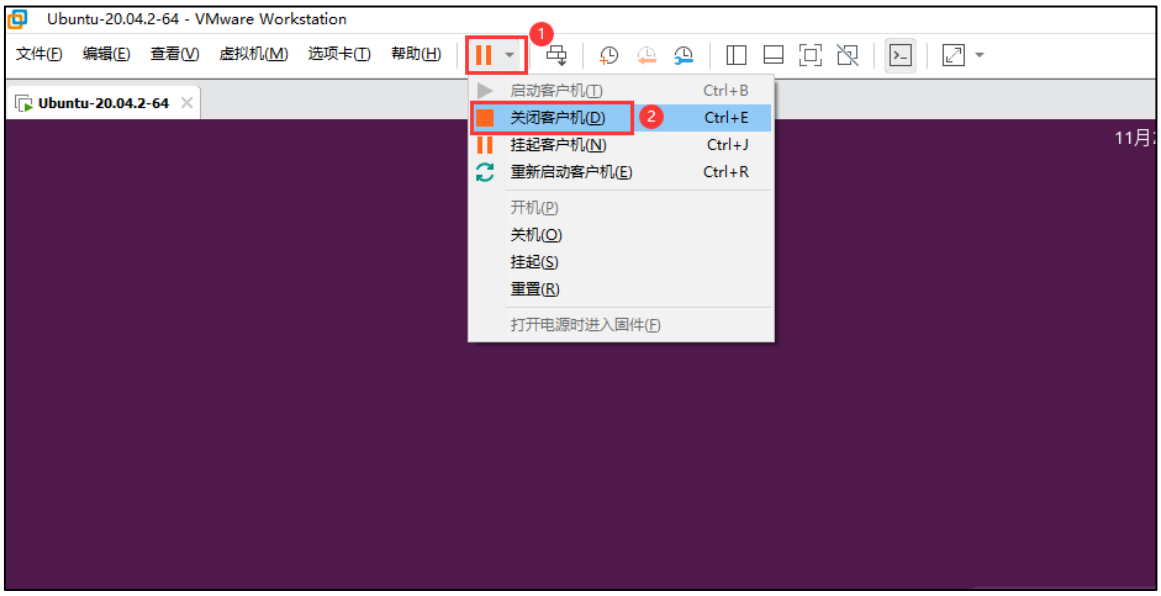


Figure 1.2.3.15 Shutdown the virtual machine (1)

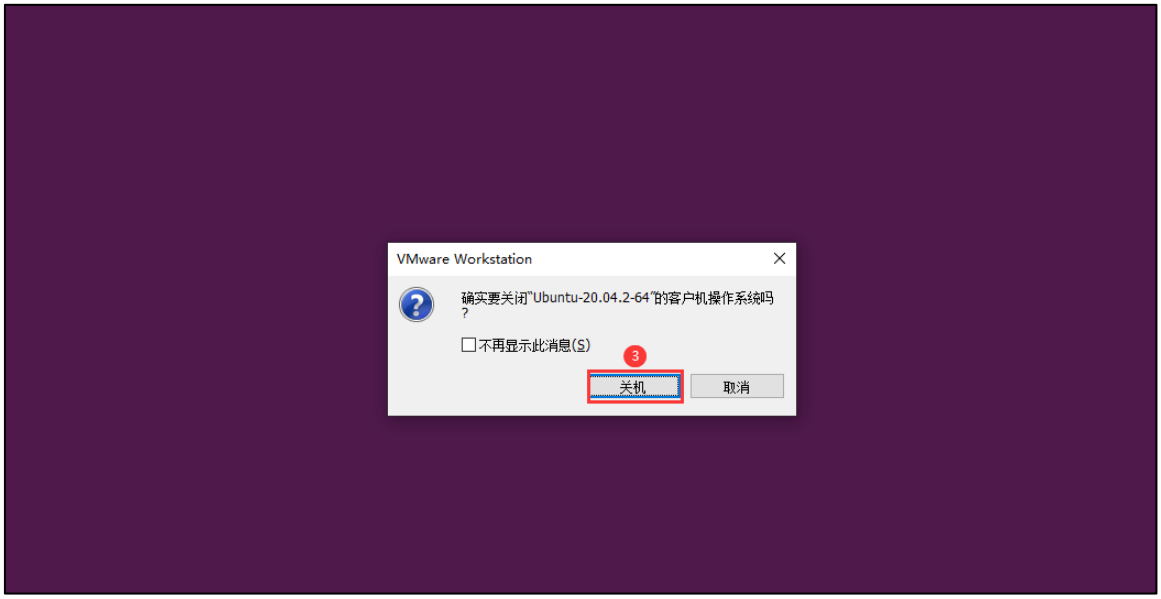


Figure 1.2.3.16 Shutdown the virtual machine (2)

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

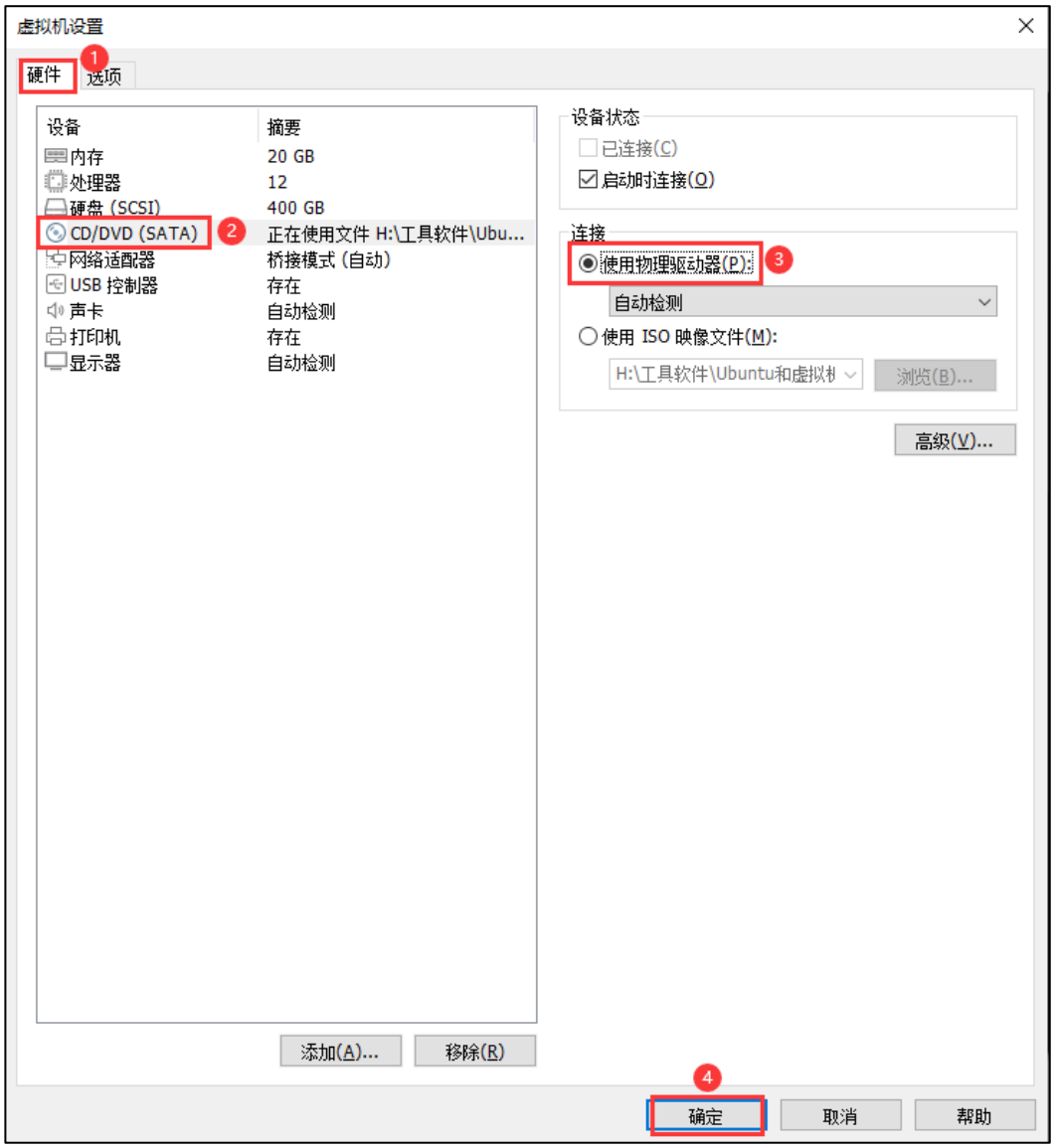


Figure 1.2.3.17 Remove the Ubuntu system image file.

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):

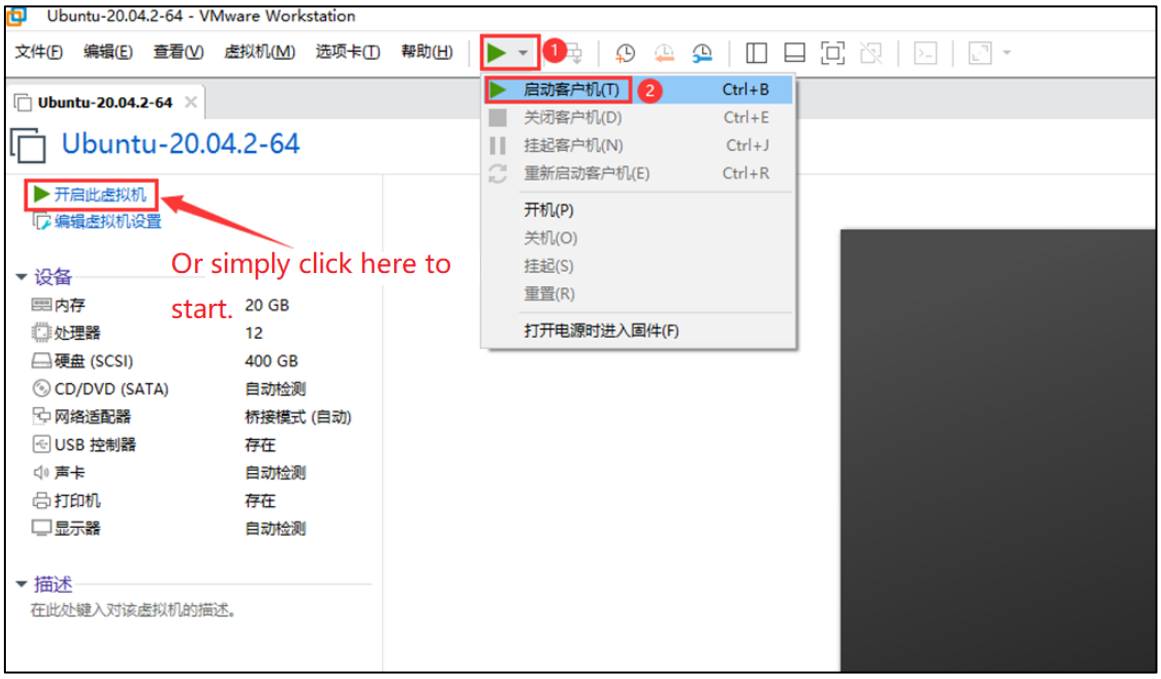


Figure 1.2.3.18 Start the virtual machine

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

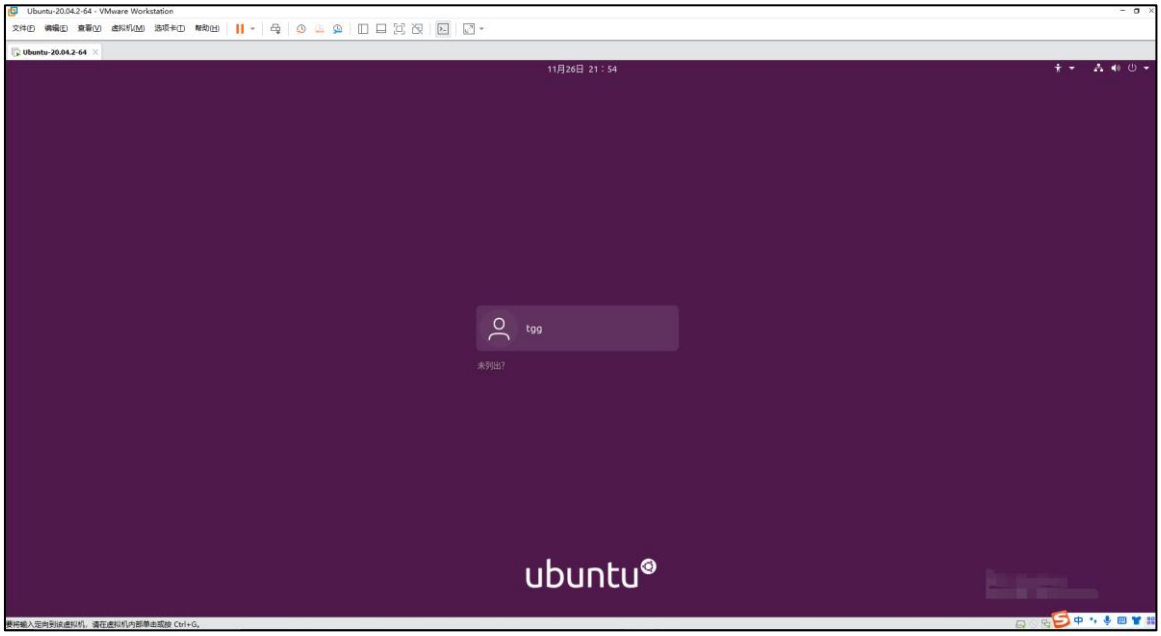


Figure 1.2.3.19 User login interface

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:

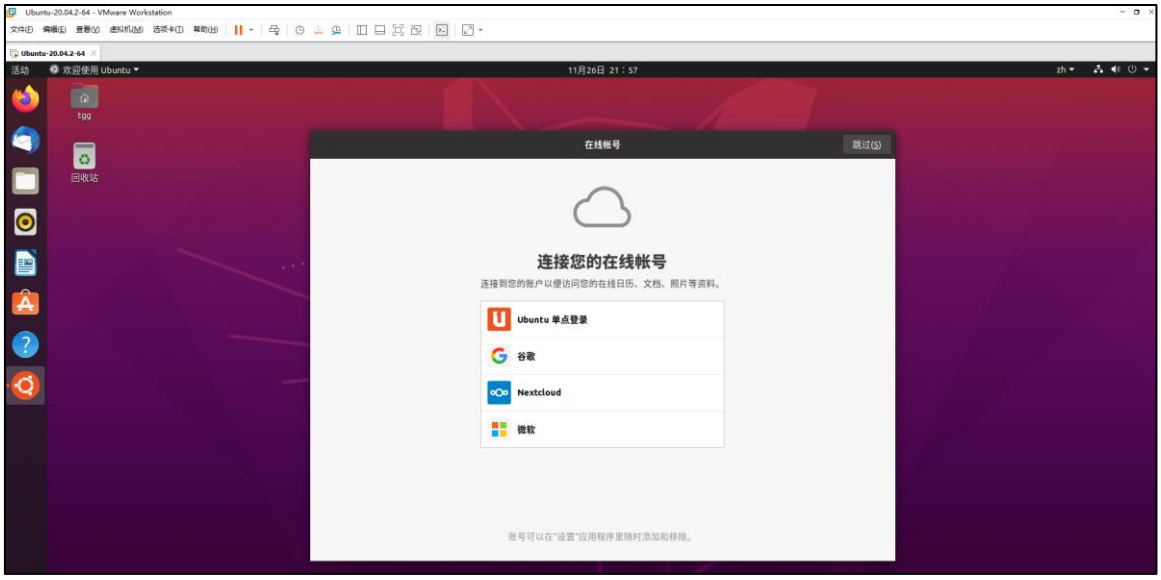


Figure 1.2.3.20 Ubuntu system main interface

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:

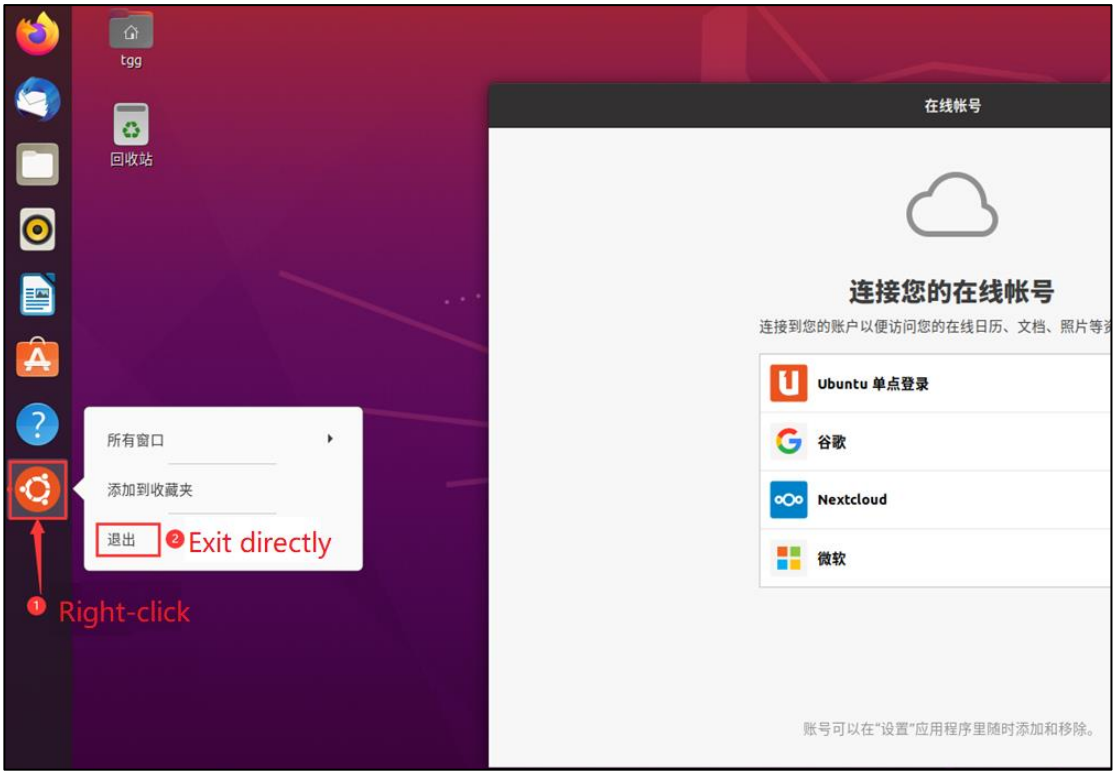


Figure 1.2.3.21 Exit the usage guidance window

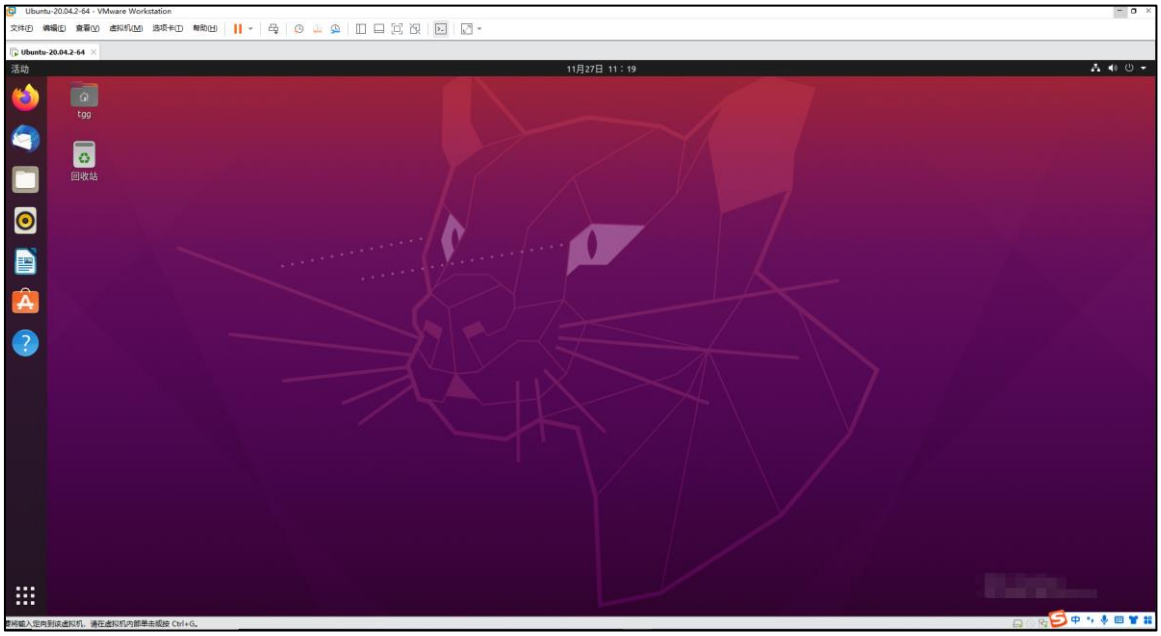


Figure 1.2.3.22 Ubuntu desktop

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

Chapter 2. Development environment setup

Before conducting the embedded Linux development of RK3568, we need to first set up the development environment; including the environment setup for both Windows and Ubuntu operating systems, as embedded Linux 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
```

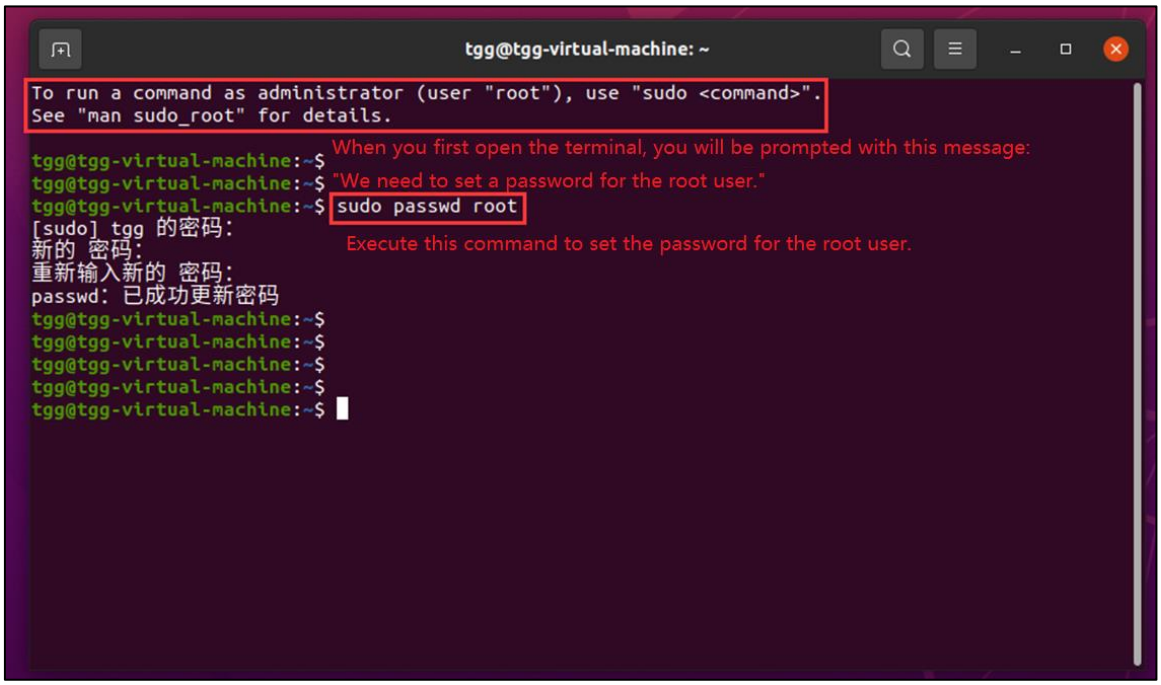


Figure 2.1.1.1 Set the password for the root user

2.1.2 Change the software download source

The default software download source of the Ubuntu system is often slow in domestic areas due to server issues. In such cases, 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 Figures 2.1.2.1 to 2.1.2.7 to make the change:

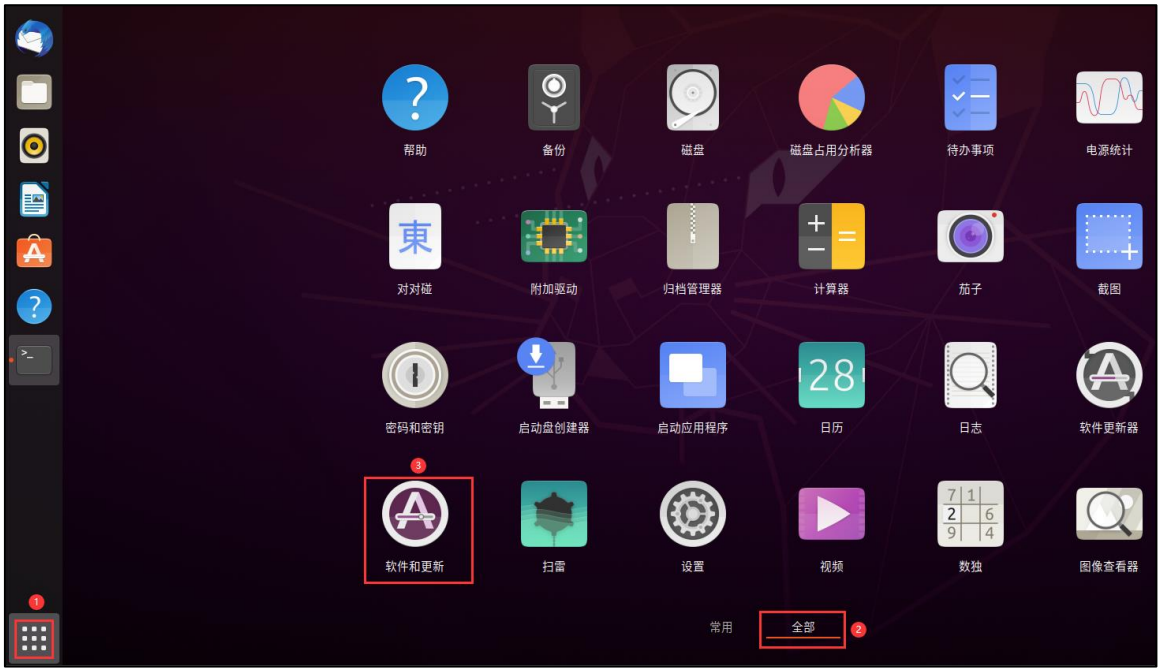


Figure 2.1.2.1 Change the software download source (1)

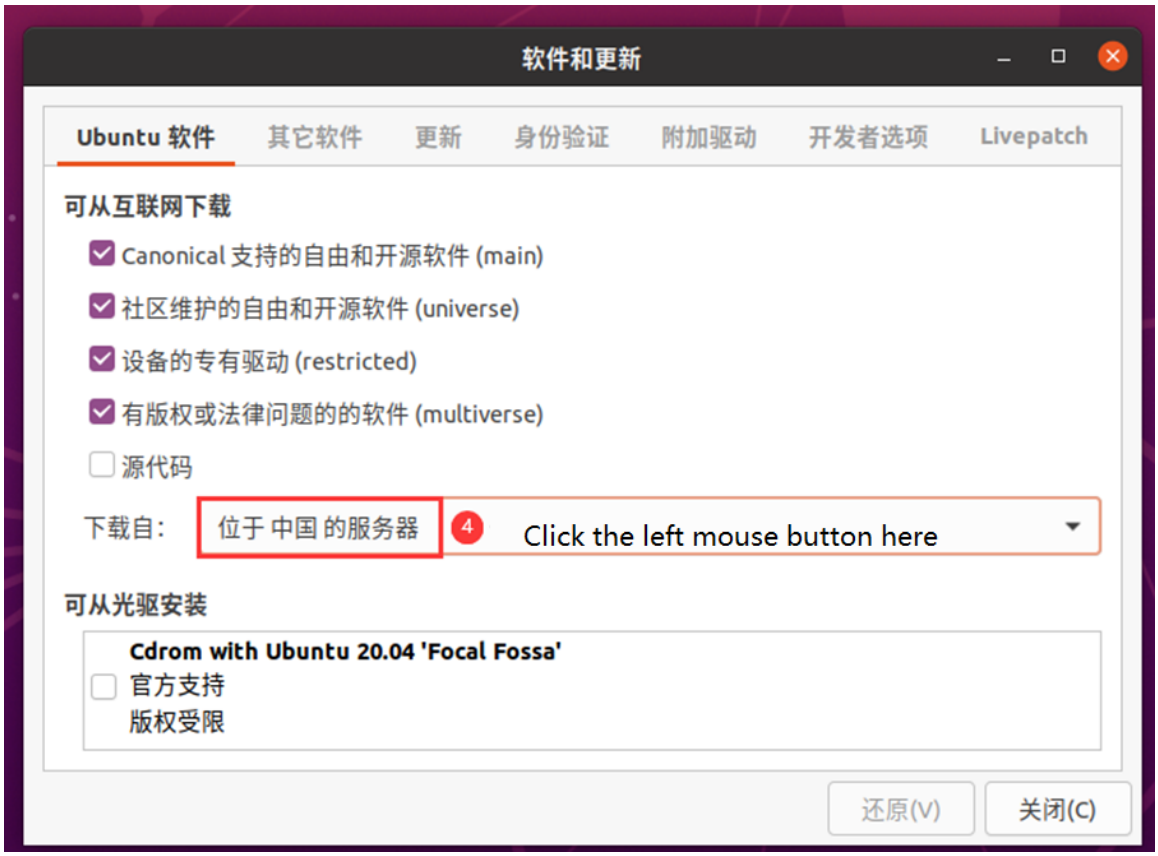


Figure 2.1.2.2 Change the software download source (2)

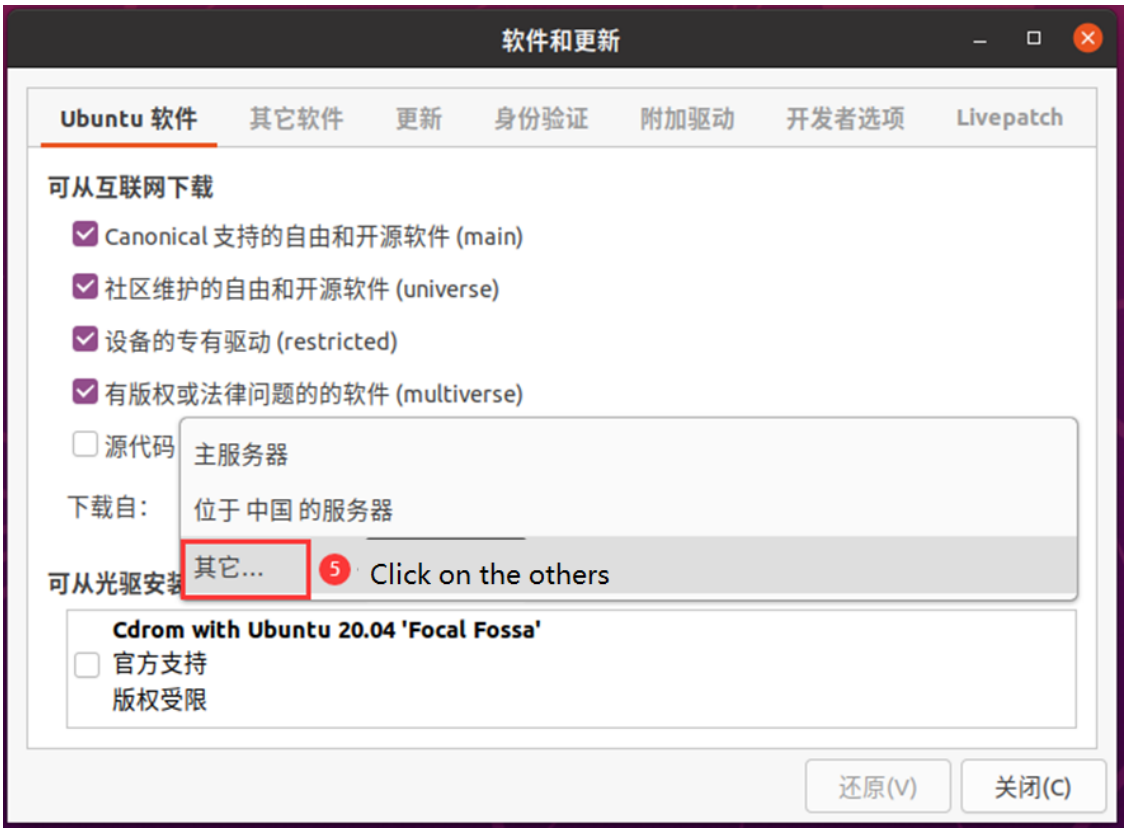


Figure 2.1.2.3 Change the software download source (3)

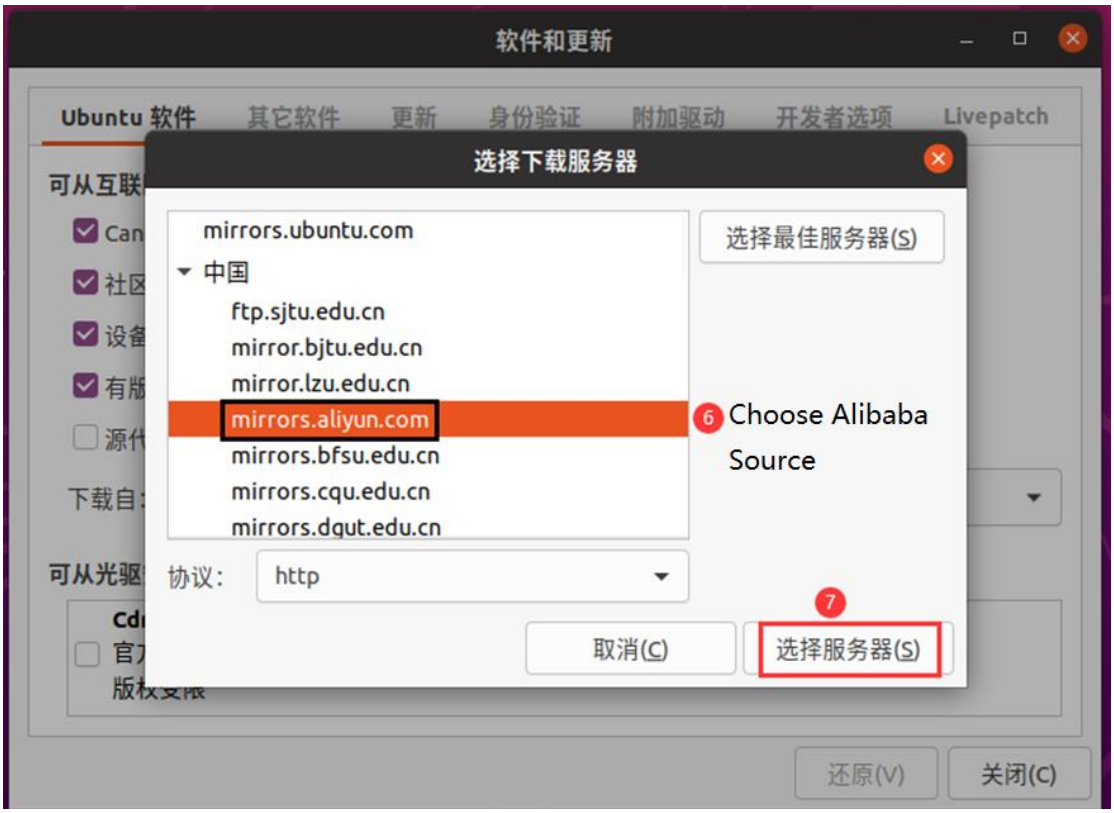


Figure 2.1.2.4 Change the software download source (4)

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

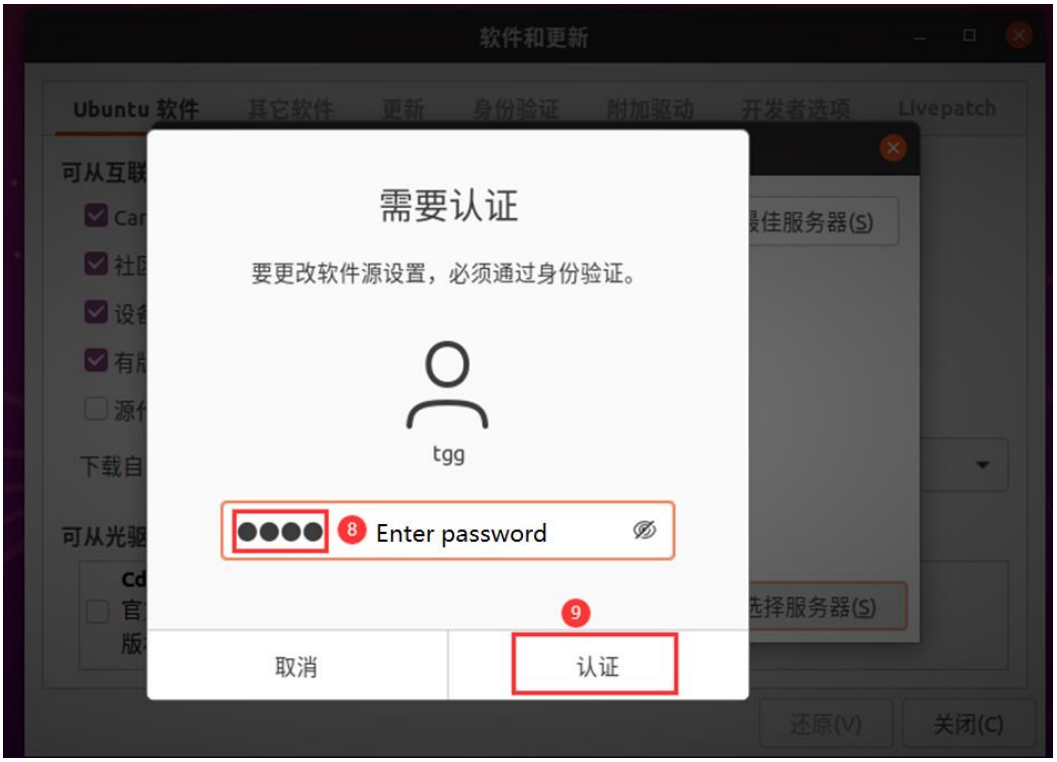


Figure 2.1.2.5 Change the software download source (5)

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:

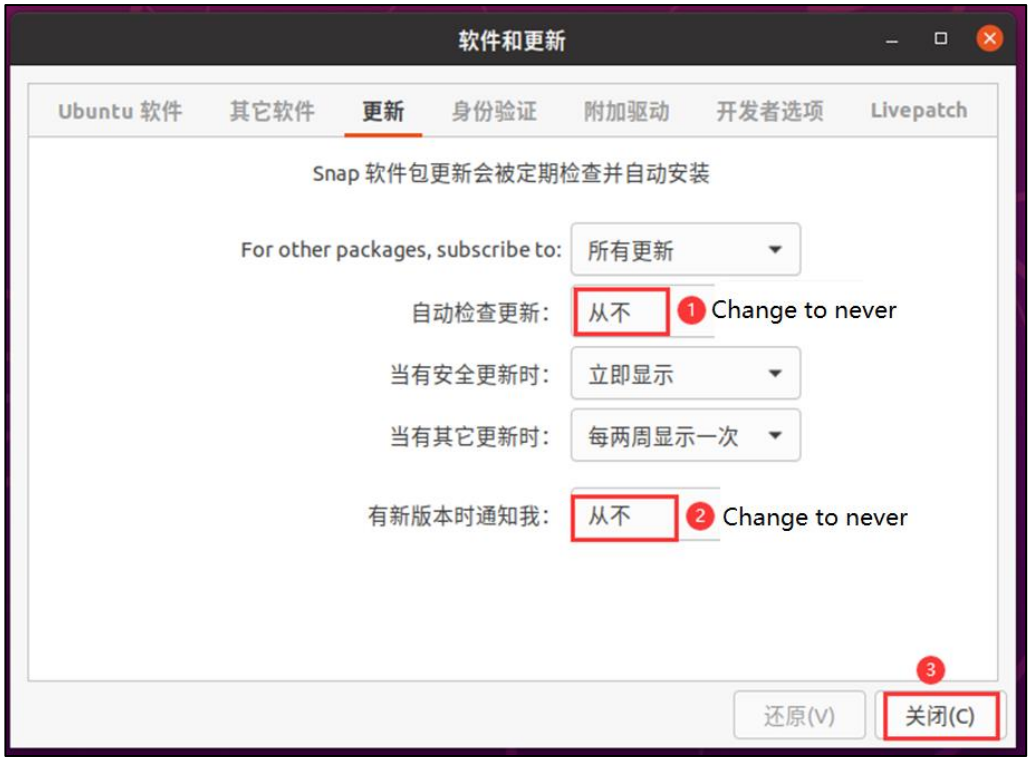


Figure 2.1.3.1 Disable automatic update function

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 sharing folders, Samba, FTP, scp, etc. This section introduces to users the file transfer method via FTP.

To transfer files between Windows and Ubuntu via FTP, the following two things need to be done.

2.2.1 Set up an FTP server on the Ubuntu system

Open the terminal on 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 is installed, use the vi command to open the /etc/vsftpd.conf configuration file. If the vim software is not installed, then you need to install the vim editor 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. It 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.

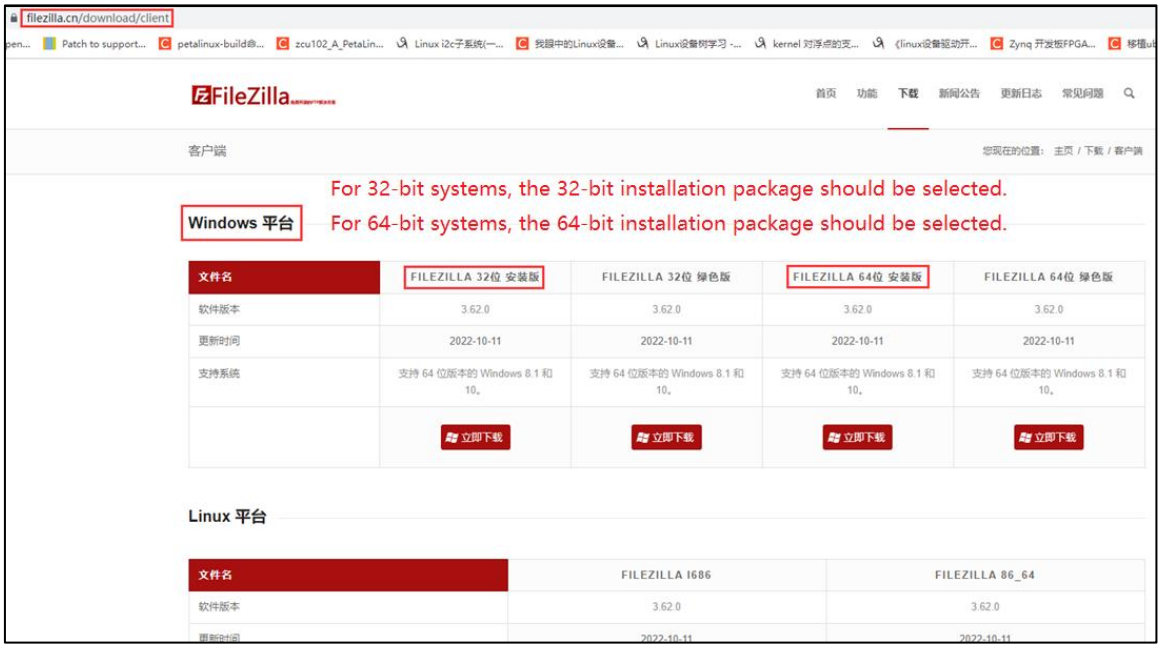


Figure 2.2.2.1 Download the installation package of FileZilla software

After the download is complete, it will be 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:

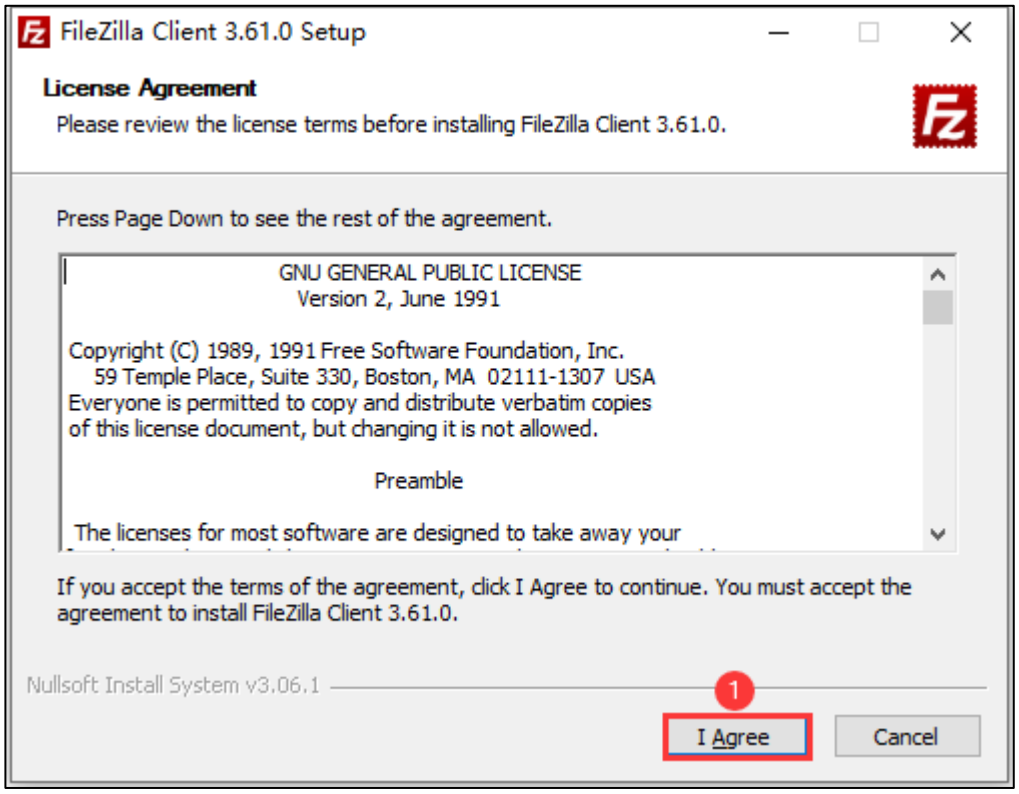


Figure 2.2.2.3 Install the FileZilla software (1)

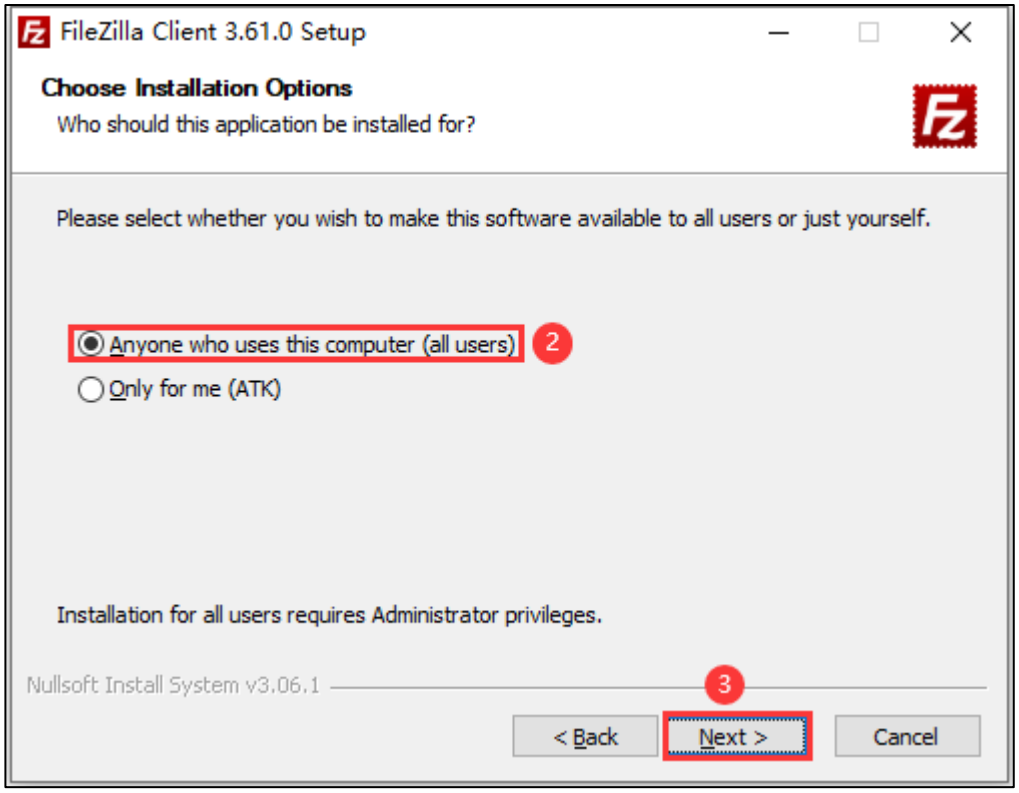


Figure 2.2.2.4 Install the FileZilla software (2)

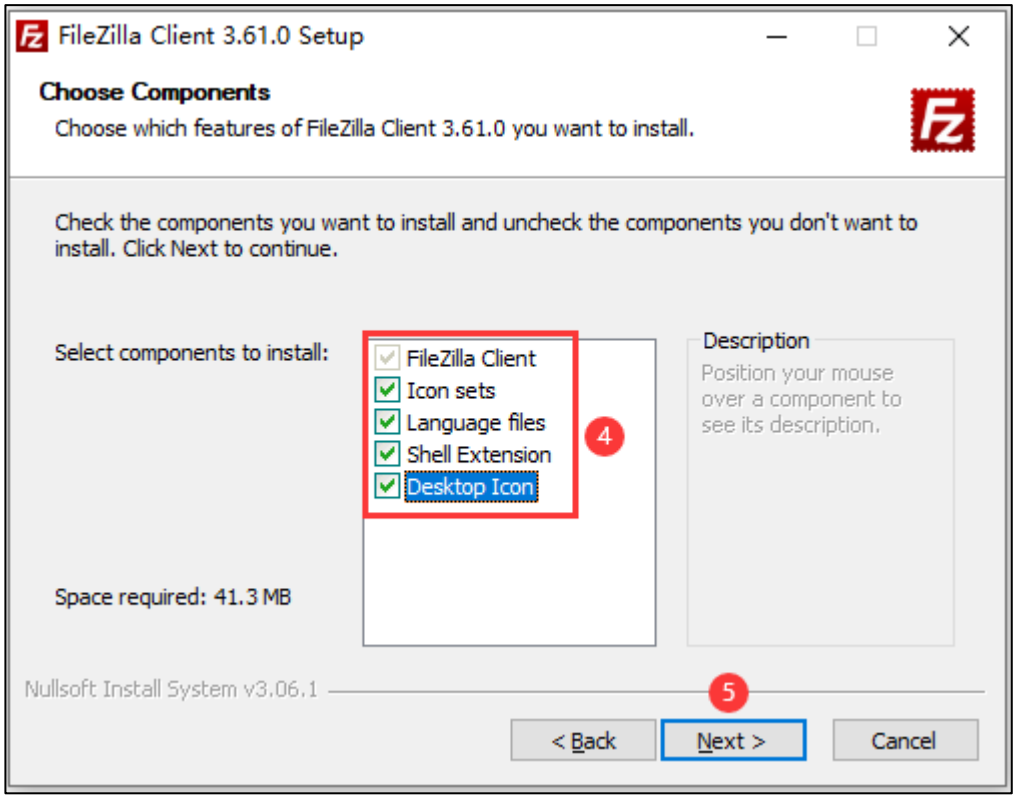


Figure 2.2.2.5 Install the FileZilla software (3)

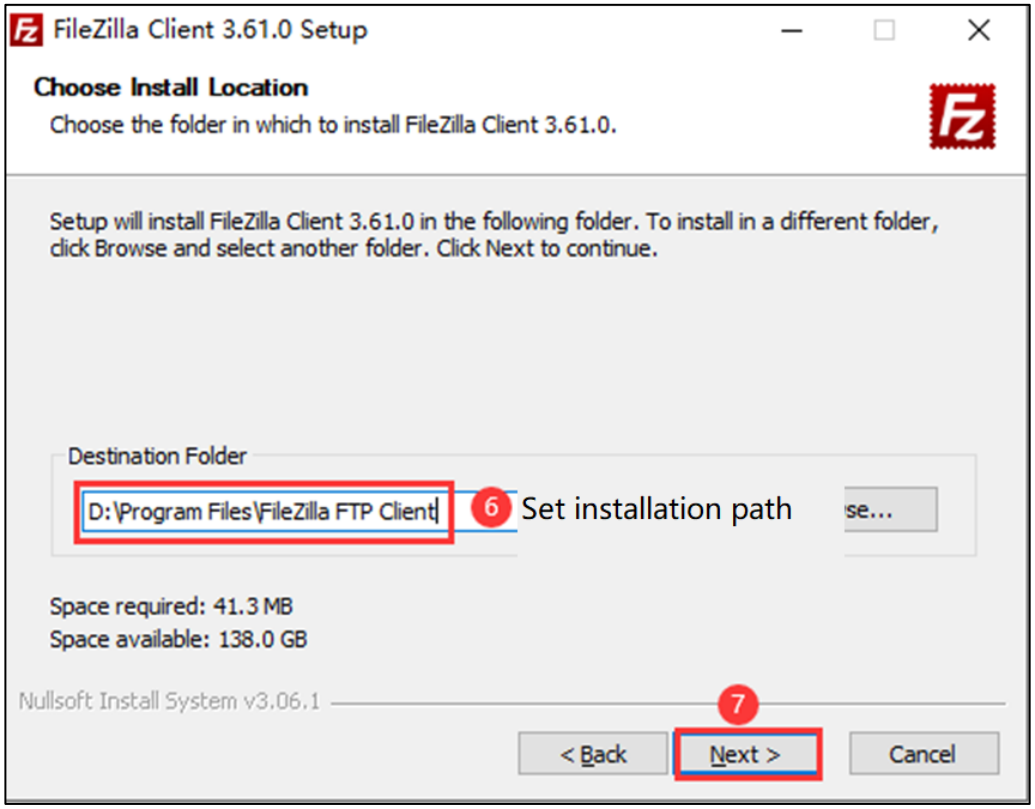


Figure 2.2.2.6 Install the FileZilla software (4)

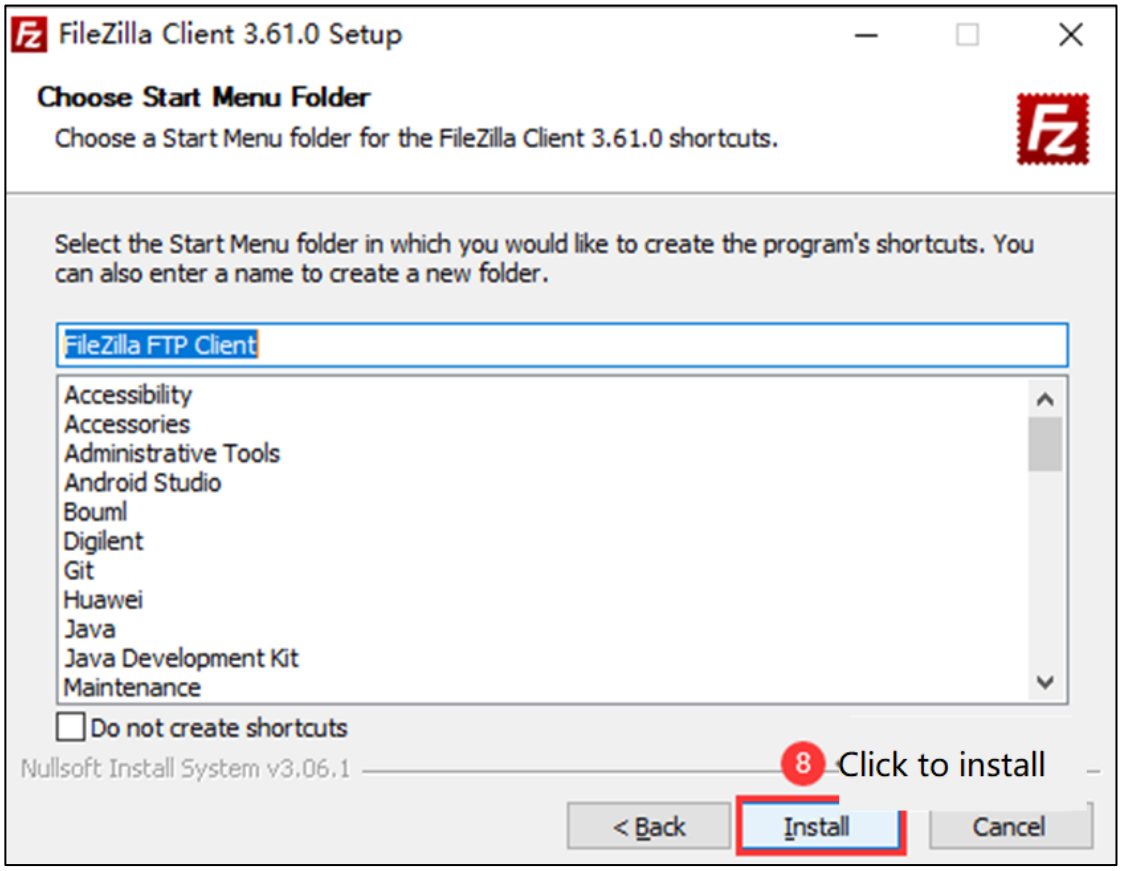


Figure 2.2.2.7 Install the FileZilla software (5)

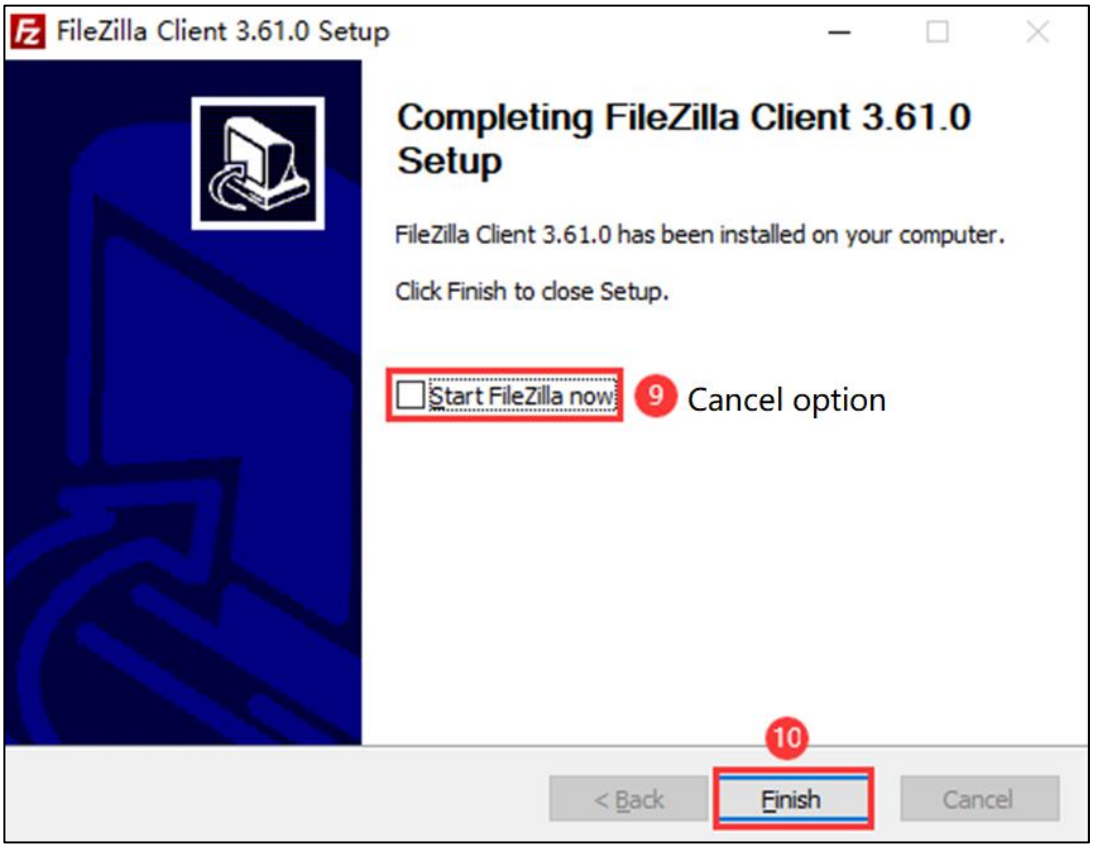


Figure 2.2.2.8 Install the FileZilla software (6)

At this point, the software installation is complete. A corresponding shortcut will be generated on the desktop:

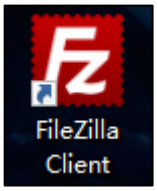


Figure 2.2.2.9 FileZilla desktop icon

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:

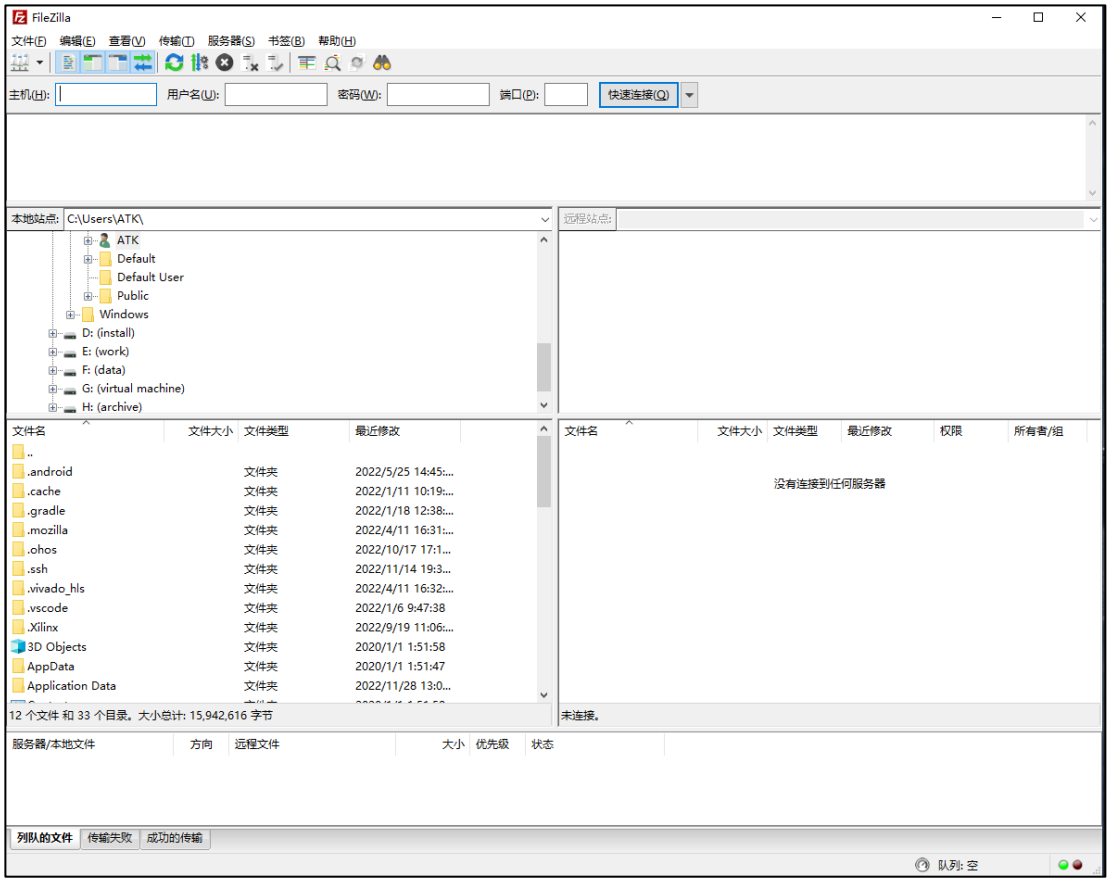


Figure 2.2.3.1 FileZilla software interface

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)

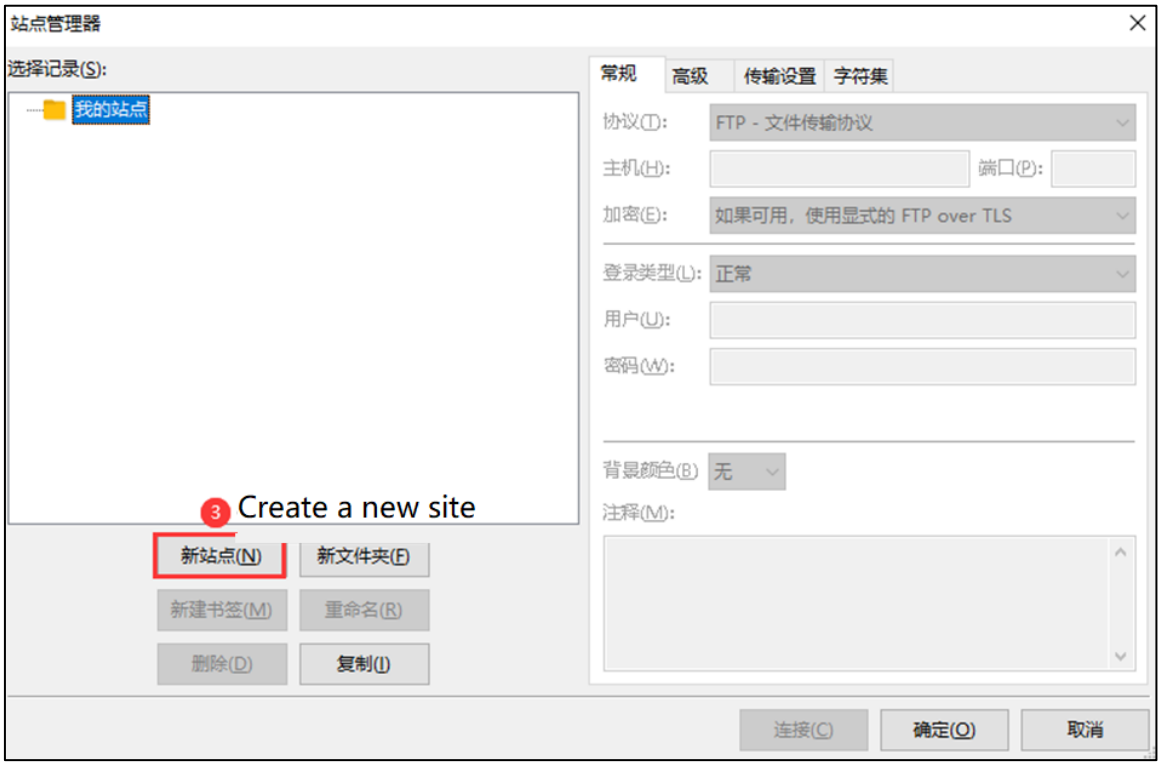


Figure 2.2.3.3 FTP client connects to the FTP server (2)

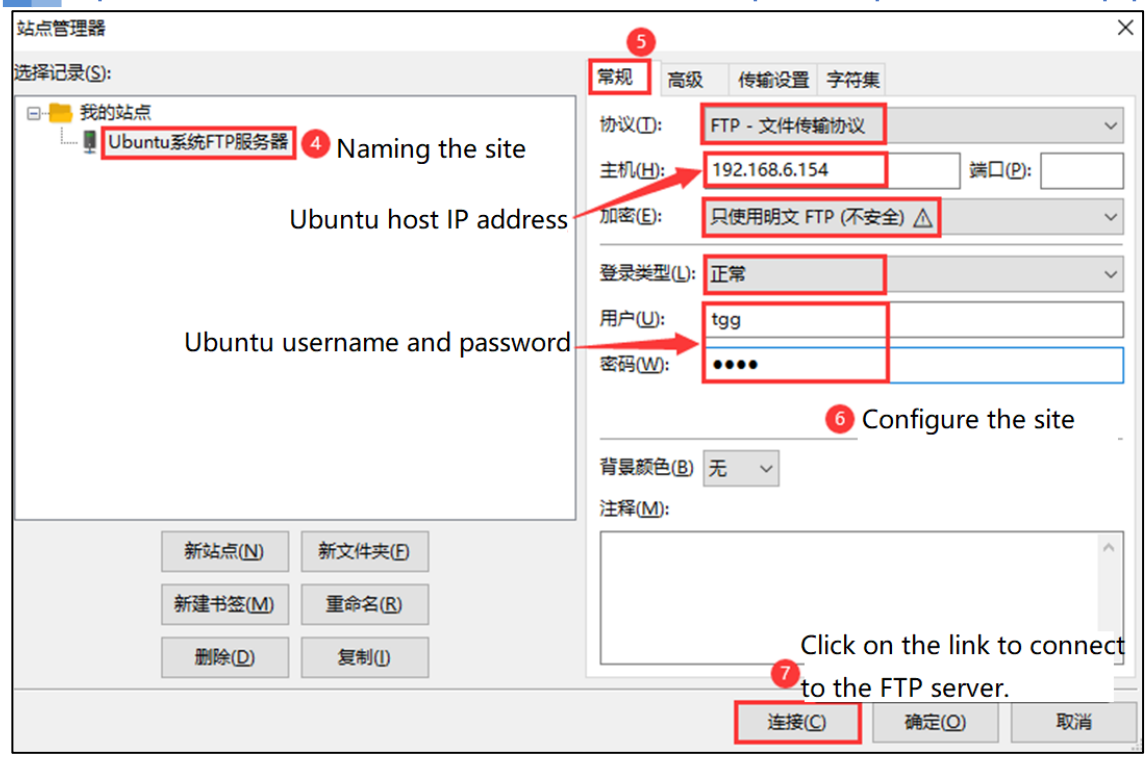


Figure 2.2.3.4 The FTP client connects to the FTP server (3)

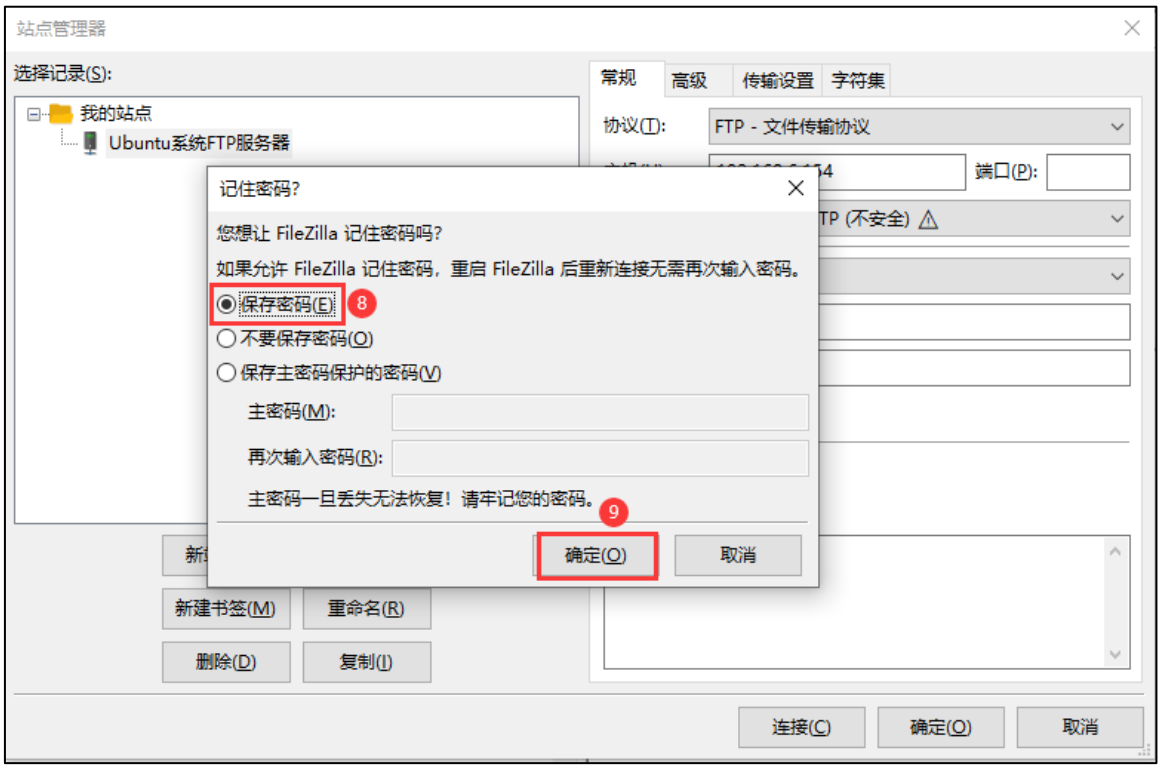


Figure 2.2.3.5 The FTP client connects to the FTP server (4)

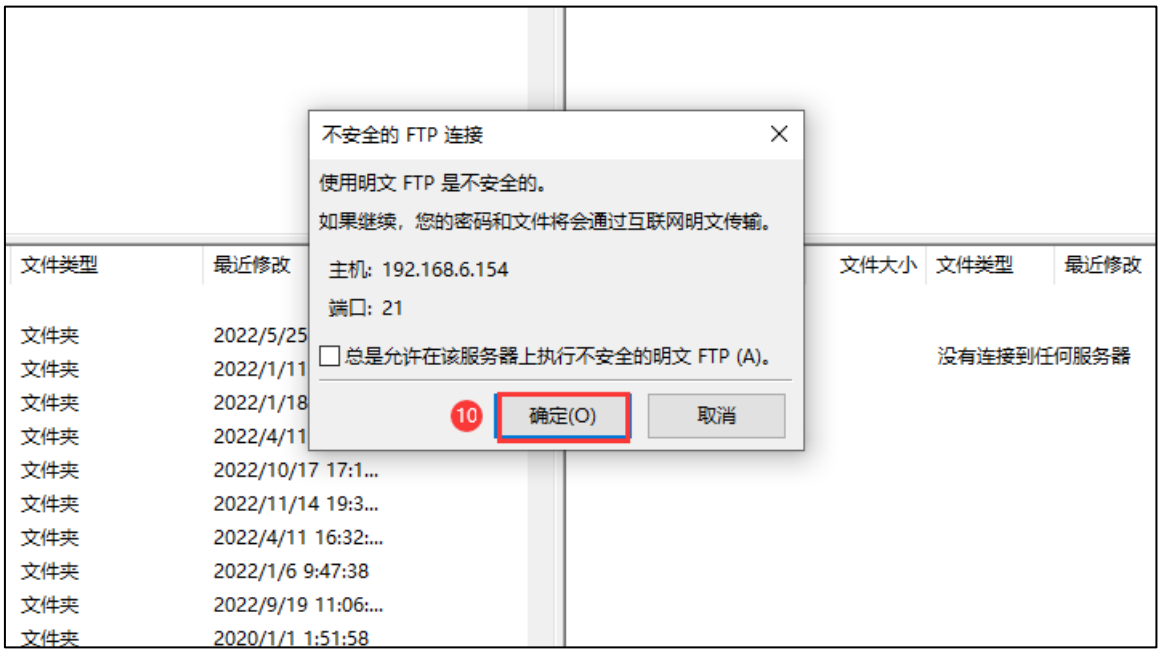


Figure 2.2.3.6 The FTP client connects to the FTP server (5)

The connection was successful as shown in Figure 2.2.3.7:

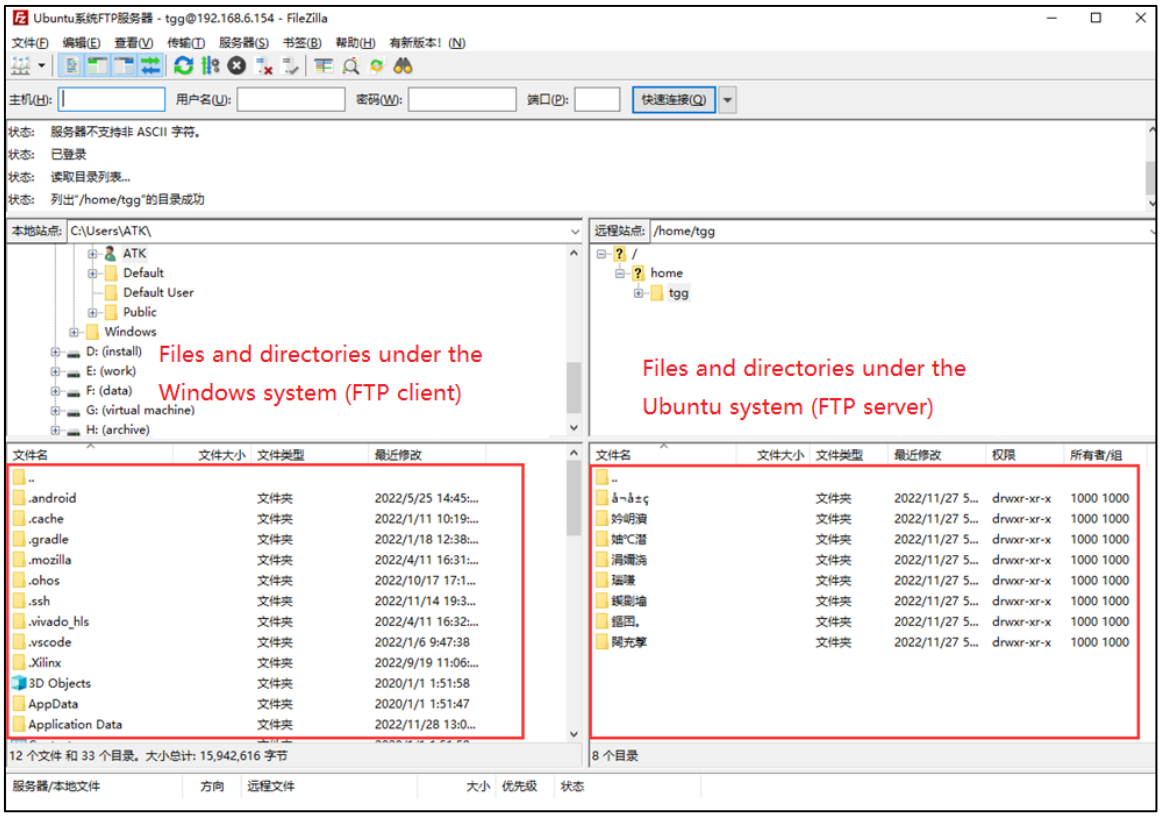


Figure 2.2.3.7 Connection successful

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 messy code (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:

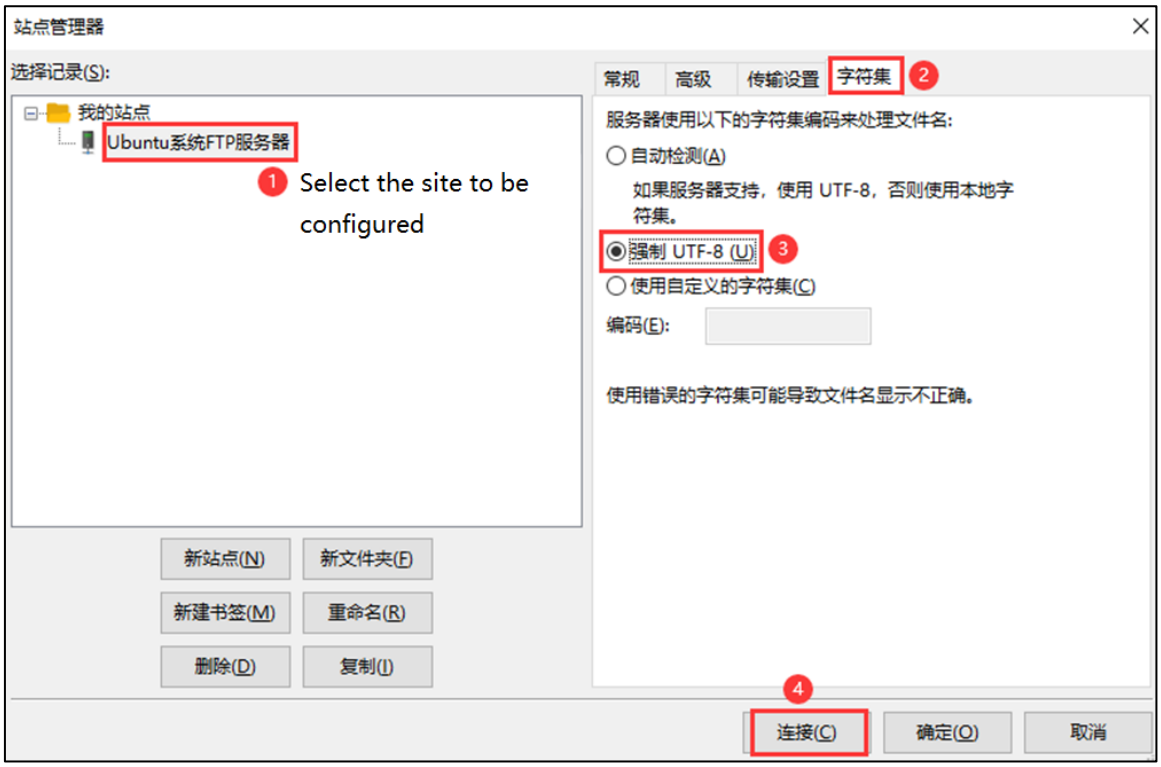


Figure 2.2.3.8 Set character encoding method (1)

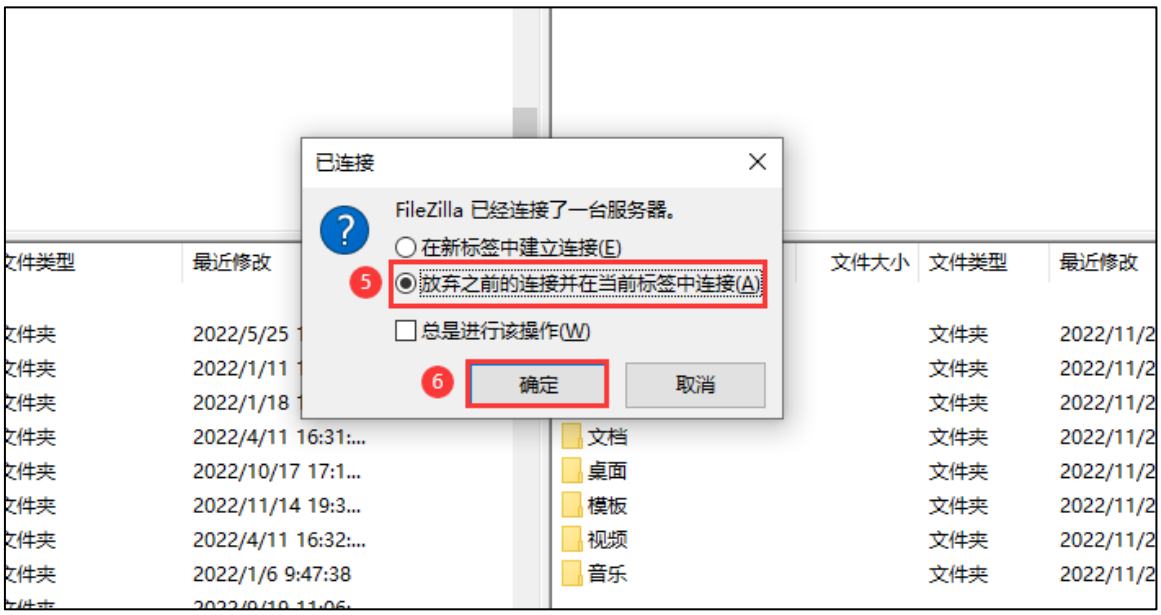


Figure 2.2.3.9 Set character encoding method (2)

After reconnecting again, you will find that the file name is displayed normally, as shown in Figure 2.2.3.10:

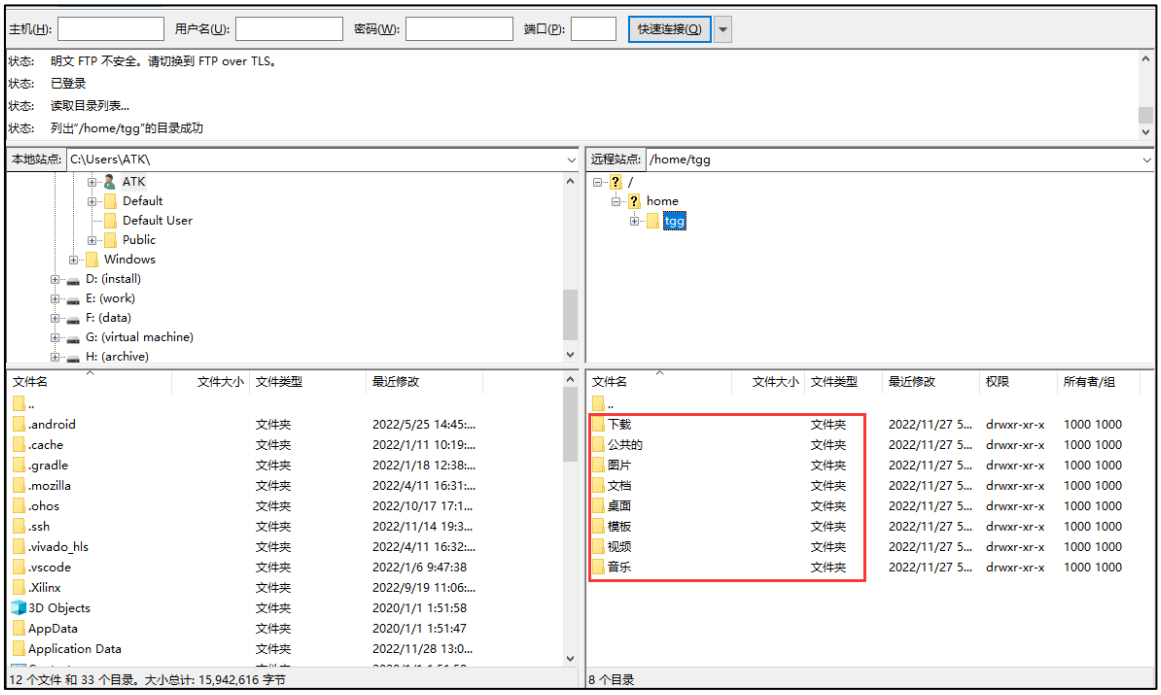


Figure 2.2.3.10 The file name is displayed normally.

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
```

```
tgg@tgg-virtual-machine:~$
tgg@tgg-virtual-machine:~$ ps -aux | grep ssh | grep -v grep
tgg         1793  0.0  0.0   6040   452 ?        Ss   09:53   0:00 /usr/bin/ssh-agent /usr/bin/im-launch env GNOME_SHELL_SE
root        7414  0.0  0.0  12184  6972 ?        Ss   15:22   0:00 sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups
tgg@tgg-virtual-machine:~$
```

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 executable file and follow the steps shown in Figures 2.6.1 to 2.6.2 to install the CH340 driver (do not connect the debugging serial port of the development board before installation):

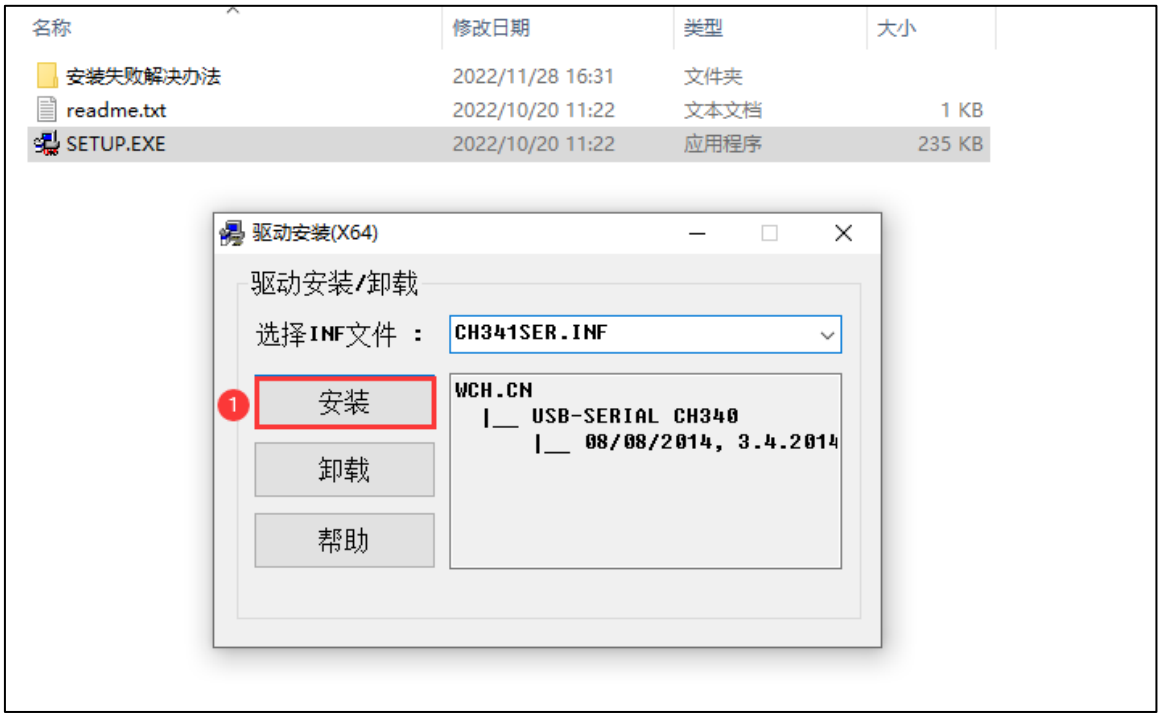


Figure 2.6-1 Install CH340 driver (1)

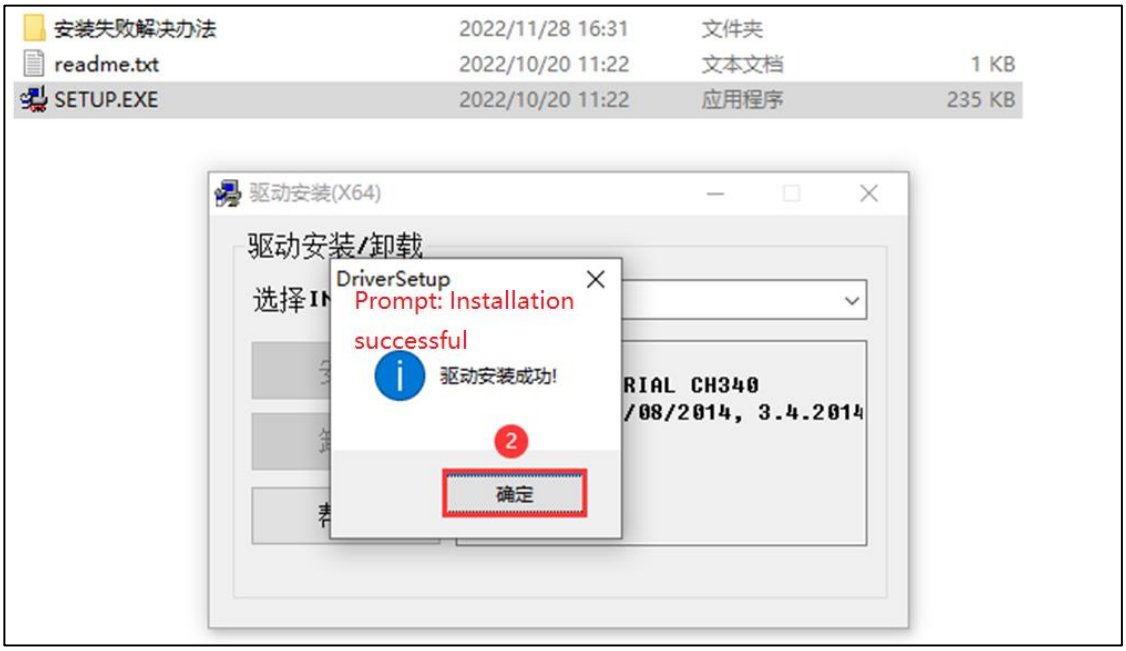


Figure 2.6-2 Install the CH340 driver (2)

After the installation is successful, 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:

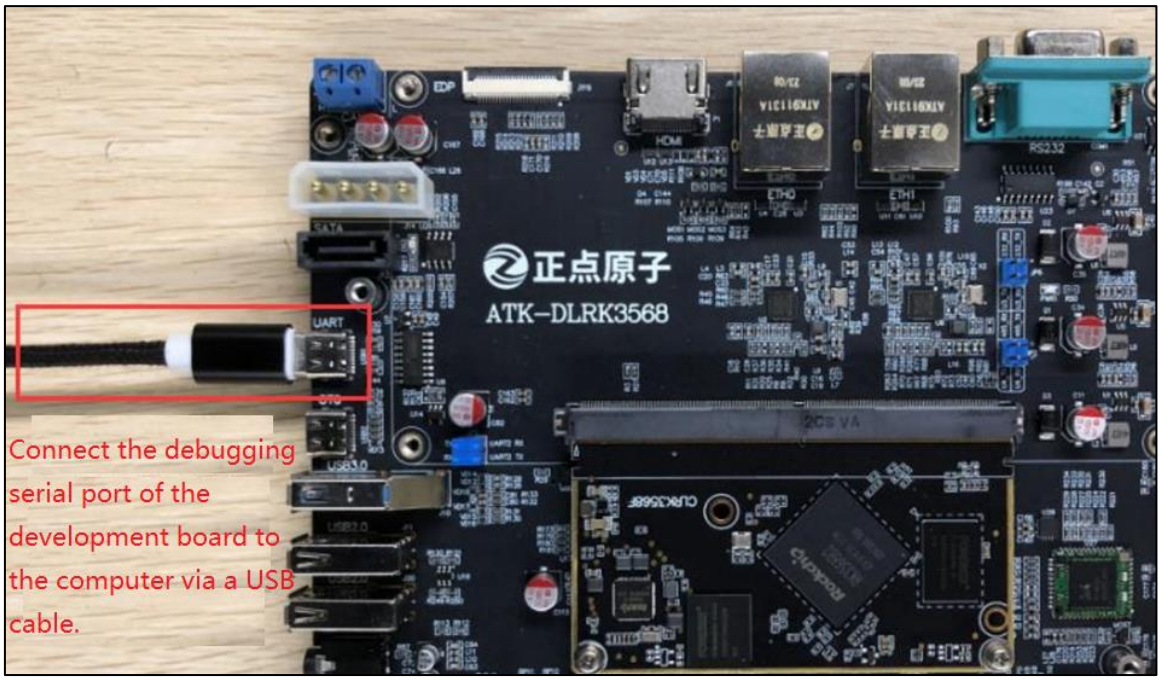


Figure 2.6-3 The development board is debugging the serial port and is connected to the computer.

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

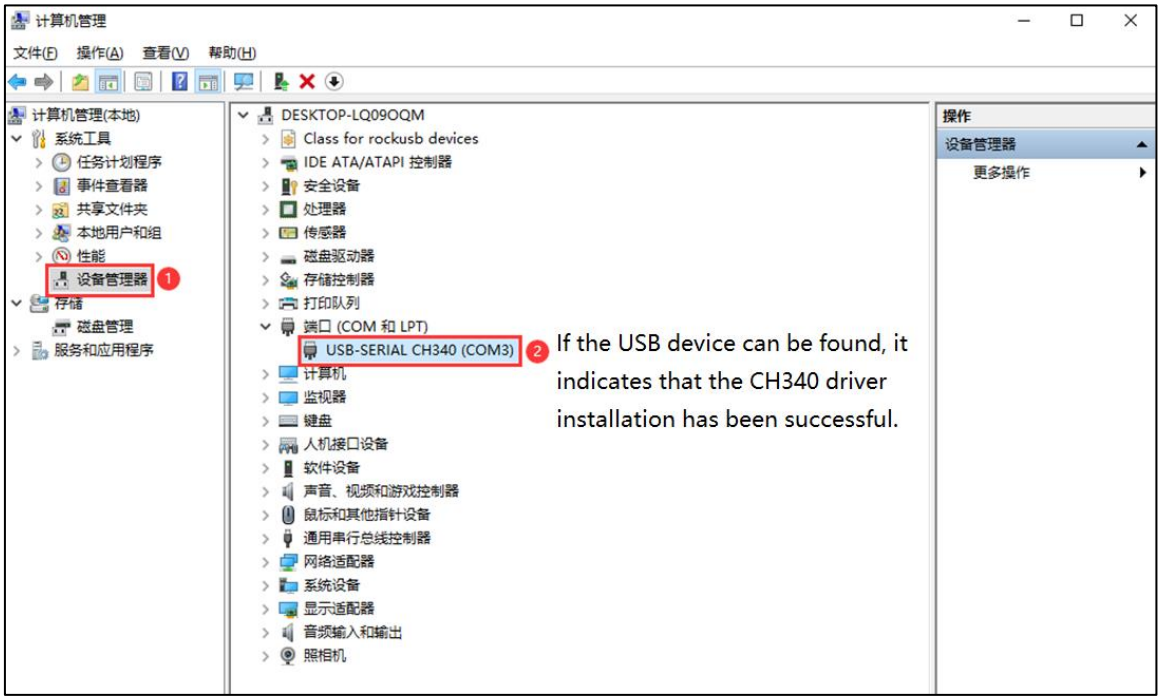


Figure 2.6-4 Windows Device Manager

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 functionality, 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

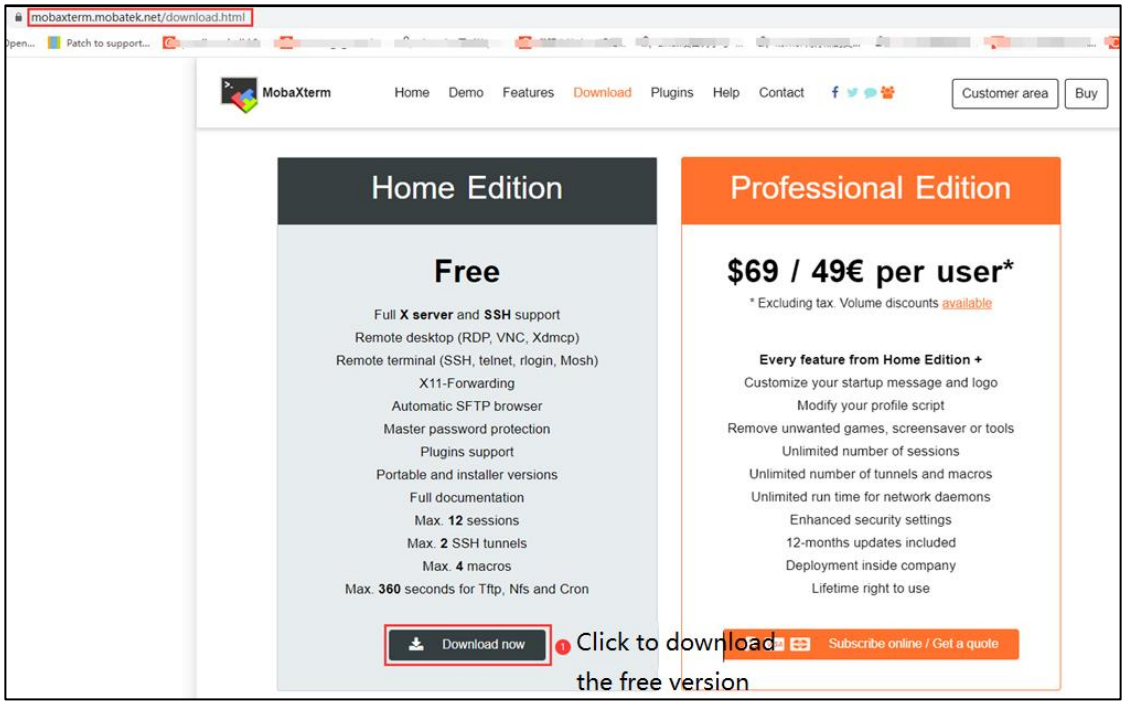


Figure 2.7.1.1 Download the MobaXterm software (1)

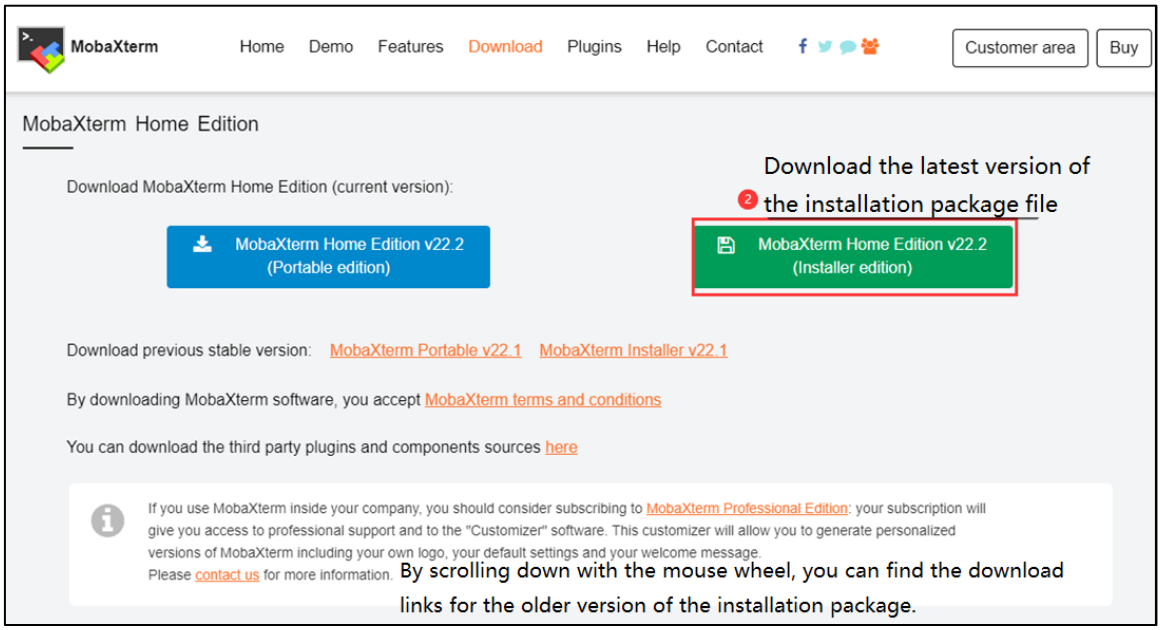


Figure 2.7.1.2 Download the MobaXterm software (2)

After the download is completed, a compressed file named MobaXterm_Installer_vxxx.zip (where xxx represents the version number) will be obtained. The current latest version is 22.2. The version of the installation package provided for users in the data package is 12.3. Using the new version is fine, or the old version is also acceptable. It doesn't matter which one you choose. Here, we will take version 12.3 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:

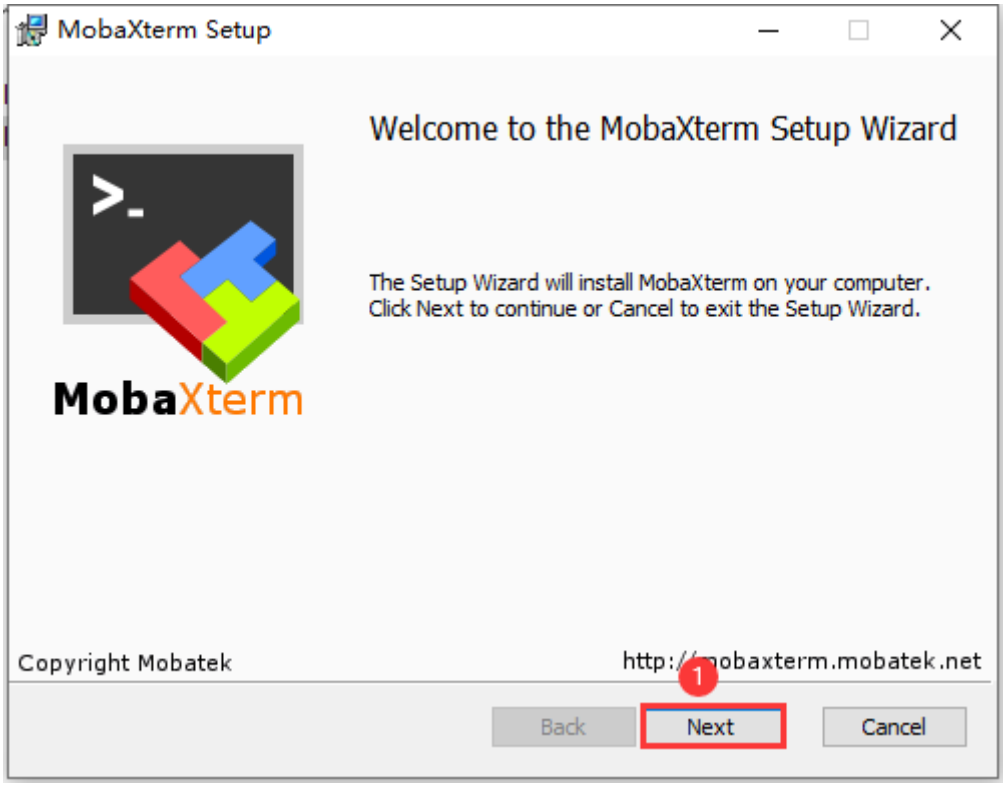


Figure 2.7.2.2 Install the MobaXterm software (1)

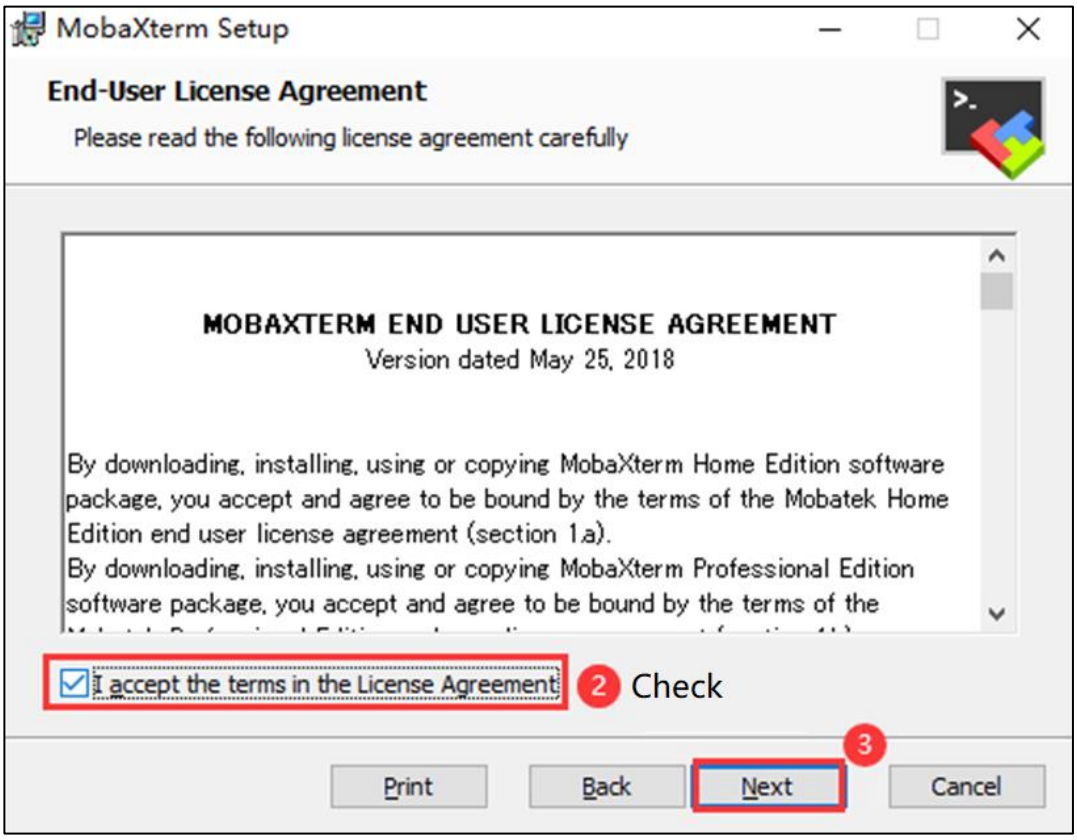


Figure 2.7.2.3 Install the MobaXterm software (2)

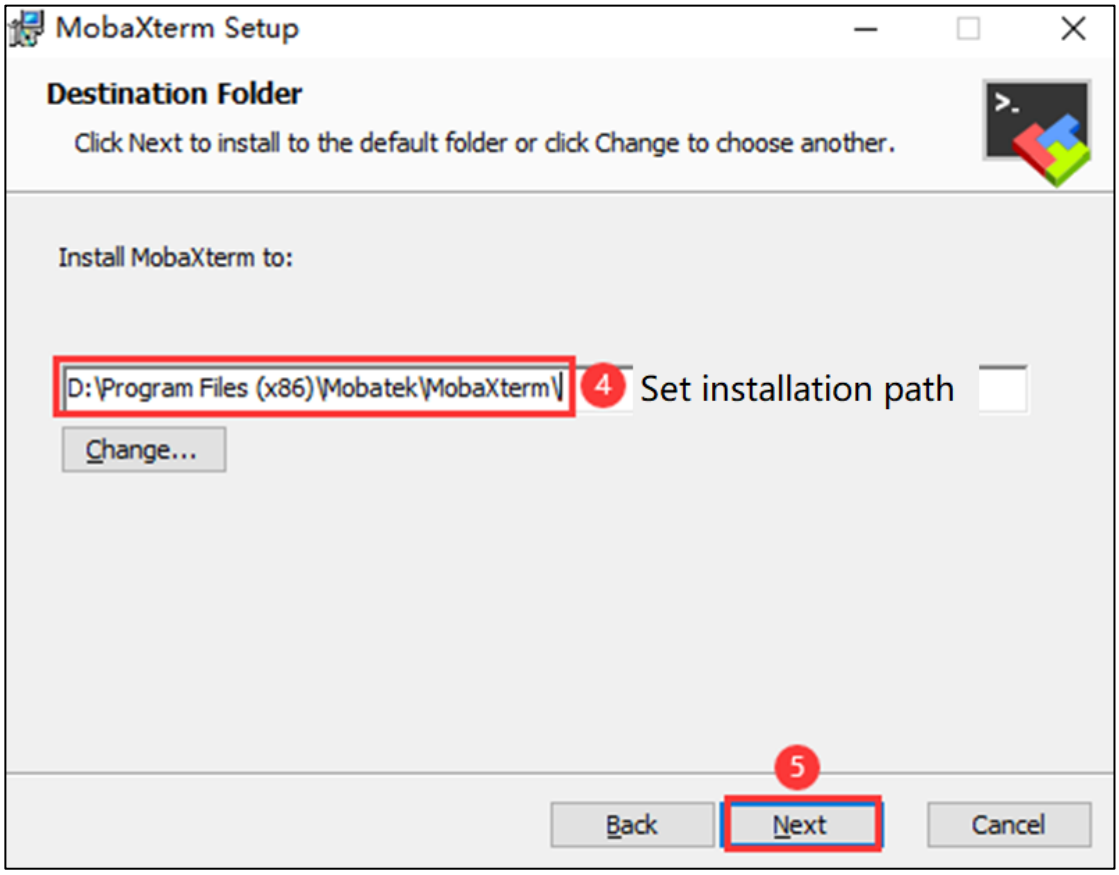


Figure 2.7.2.4 Install the MobaXterm software (3)

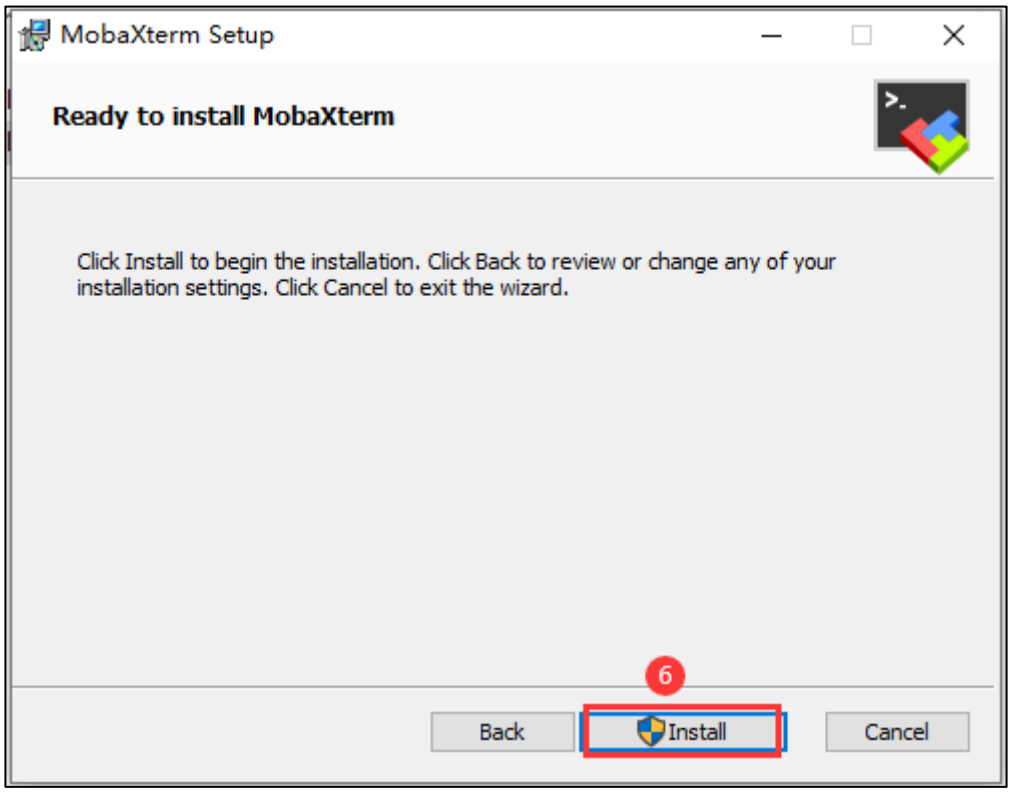


Figure 2.7.2.5 Install the MobaXterm software (4)

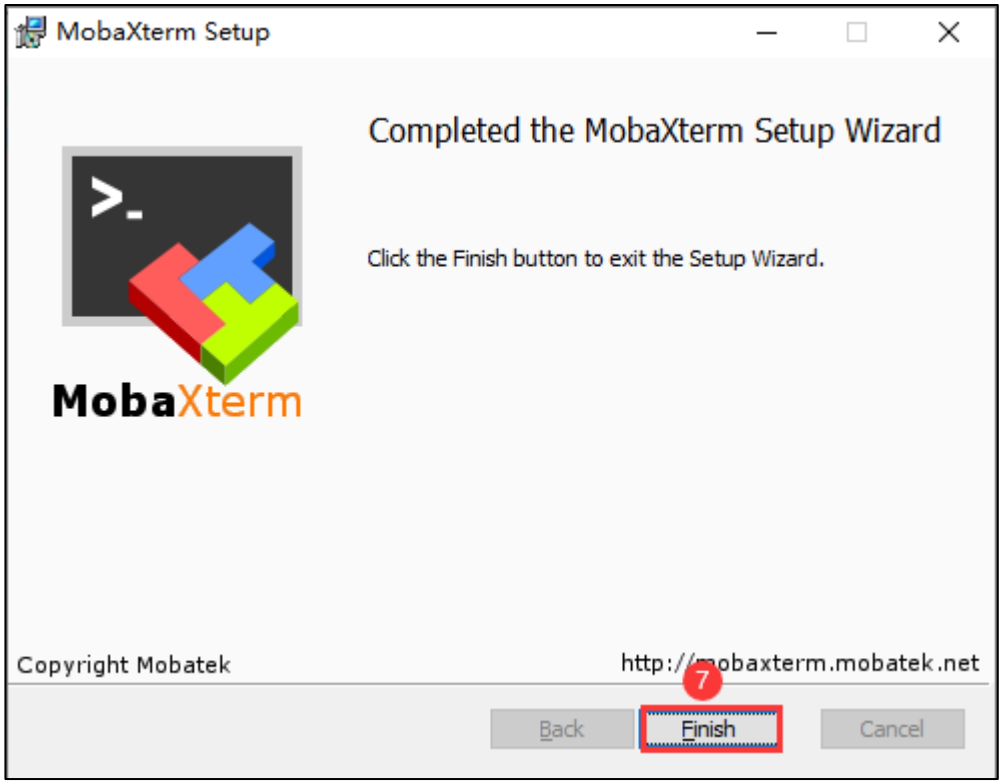


Figure 2.7.2.6 Install the MobaXterm software (5)

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

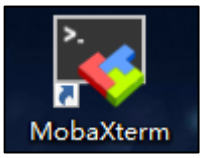


Figure 2.7.2.7 MobaXterm software desktop icon

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:

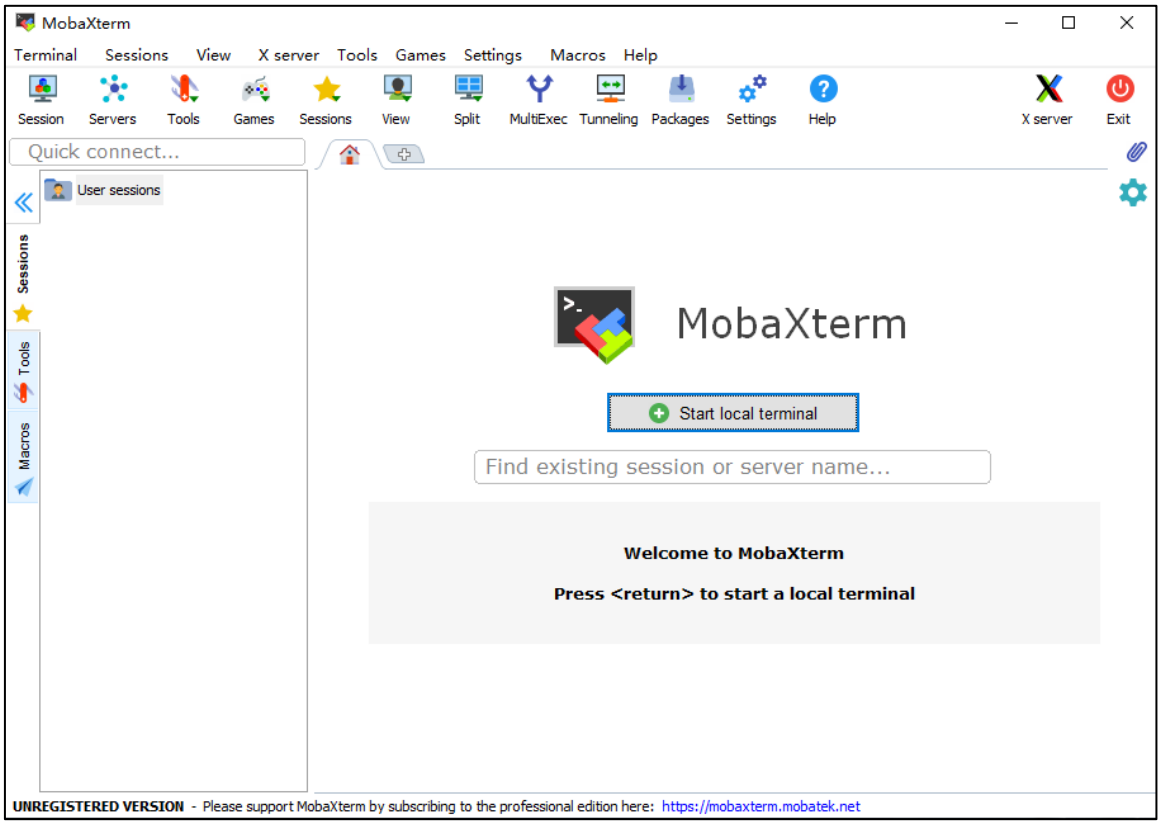


Figure 2.7.3.1 The main interface of MobaXterm software

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):

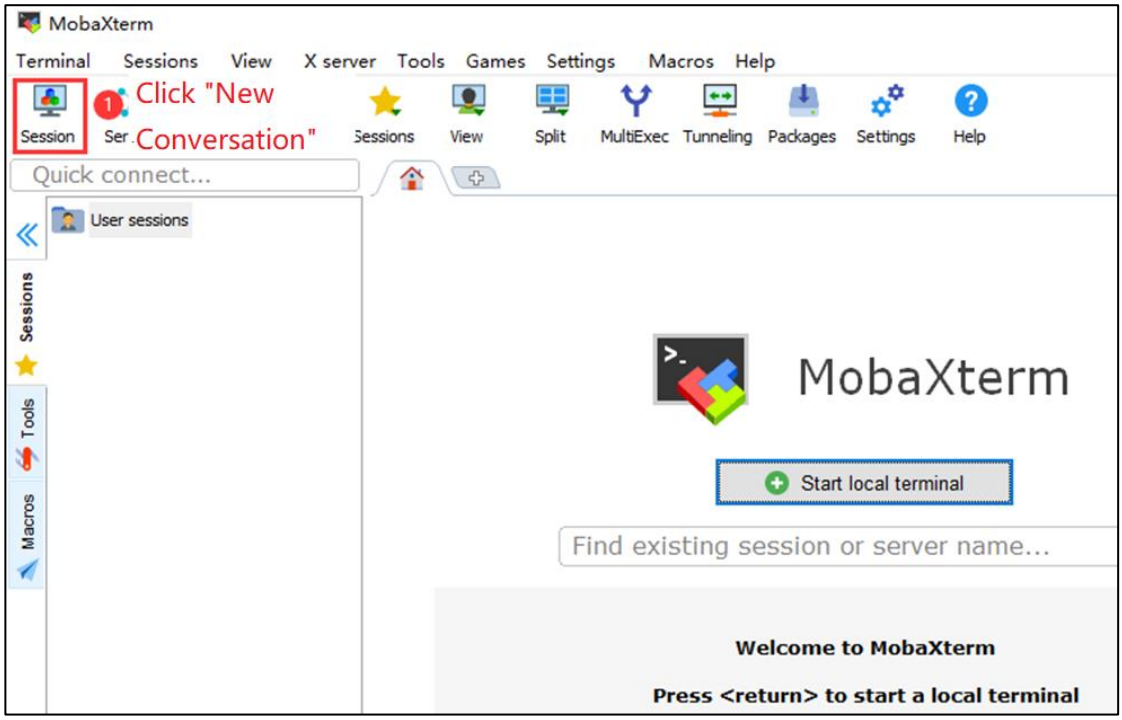


Figure 2.7.3.2 Establish serial port connection (1)

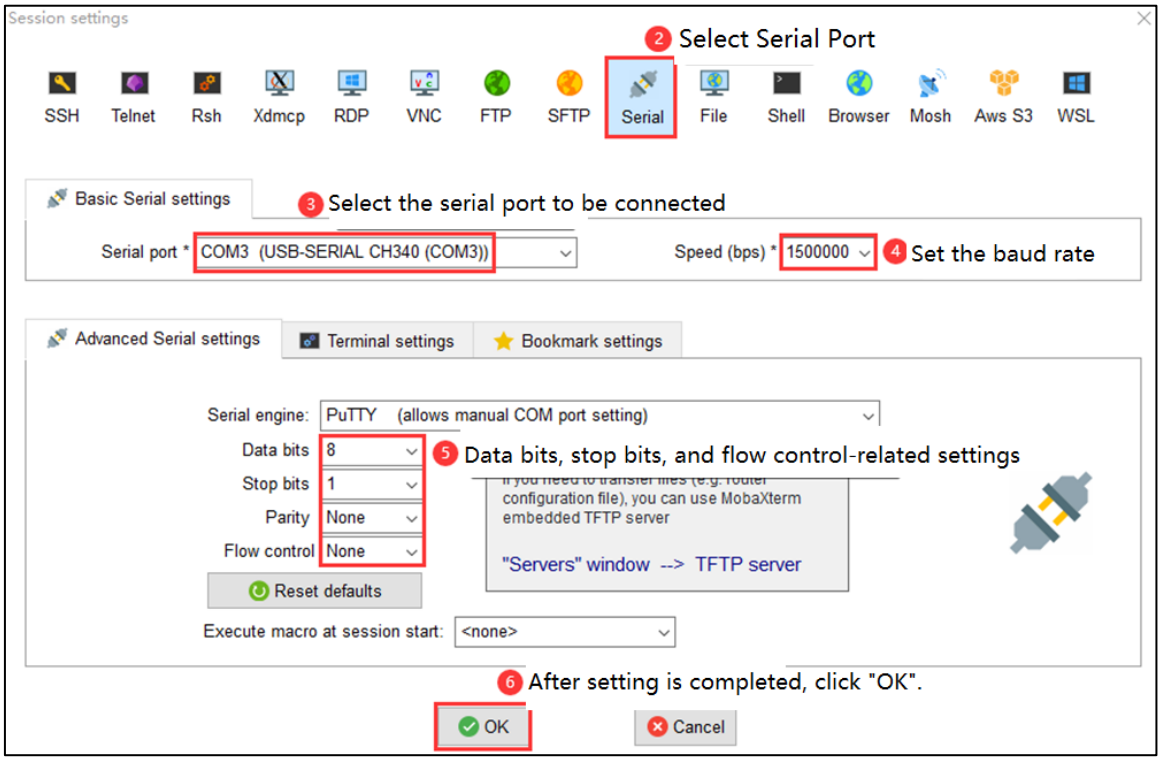


Figure 2.7.3.3 Establishing Serial Port Connection (2)

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 port communication baud rate. Set it according to the actual situation. For the debugging serial port of the RK3568 platform, its default baud rate is 1500000 (1.5M). Then set the data bits, stop bits, and flow control, and after setting, click the "**OK**" button.

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

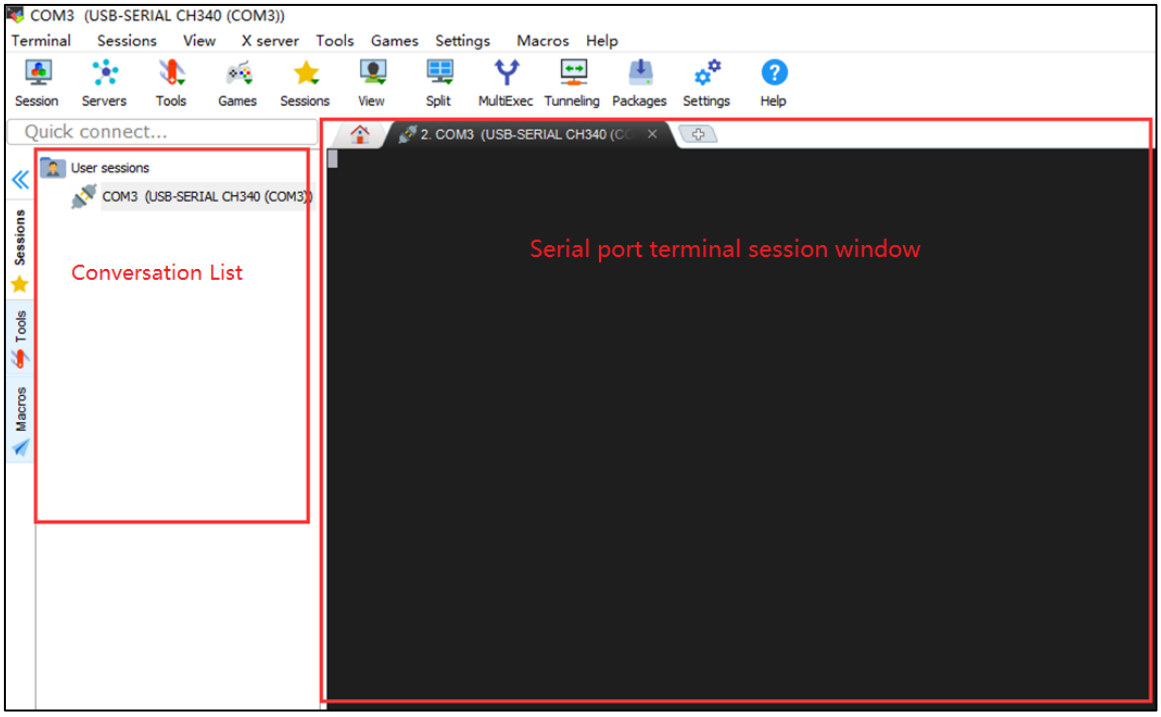


Figure 2.7.3.4 Establish serial port connection (3)

When the development board is powered on and starts the Linux system or Android system, the log information during the system startup process 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):

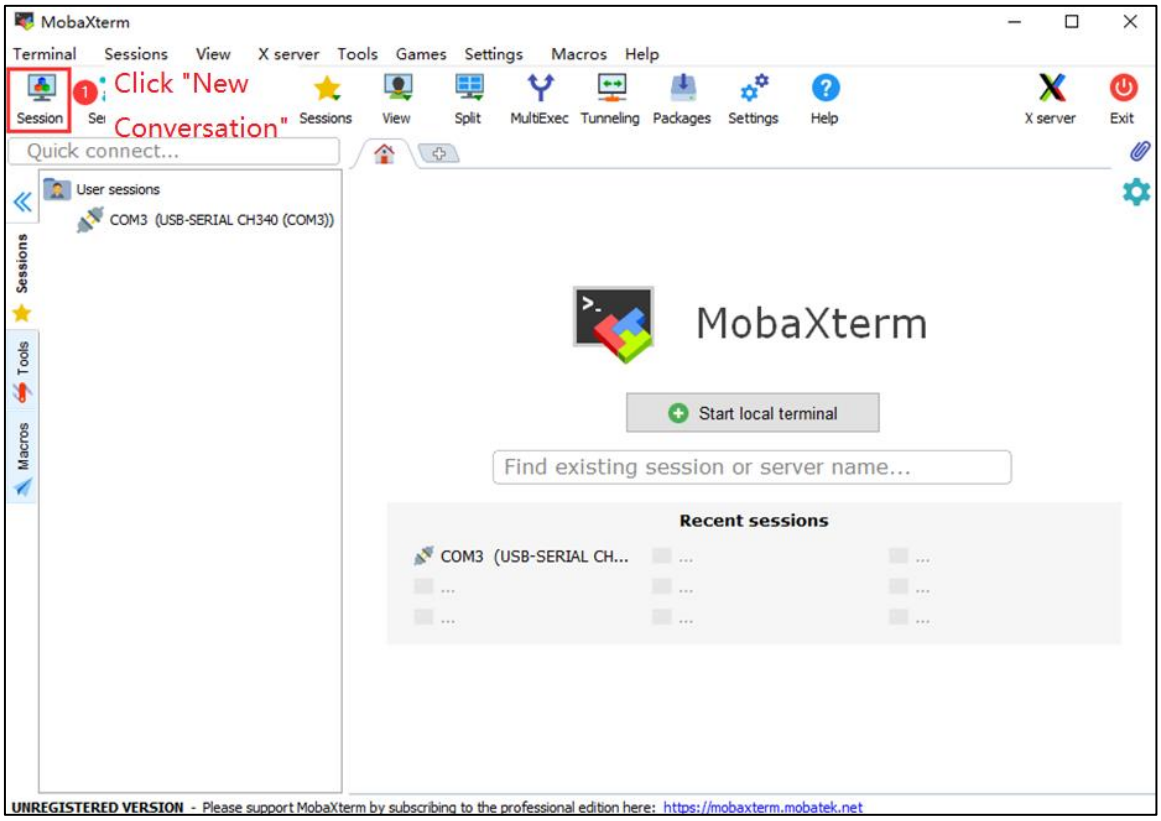


Figure 2.7.3.6 Establish SSH remote connection (1)

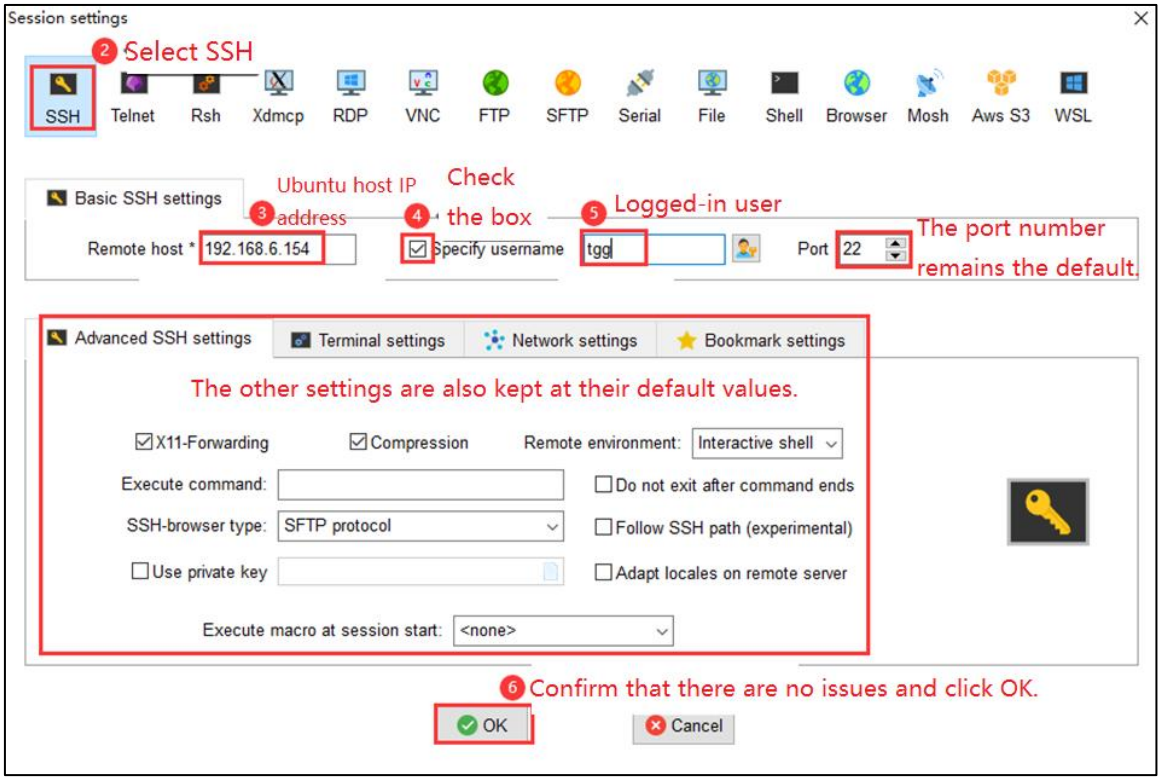


Figure 2.7.3.7 Establish SSH remote connection (2)

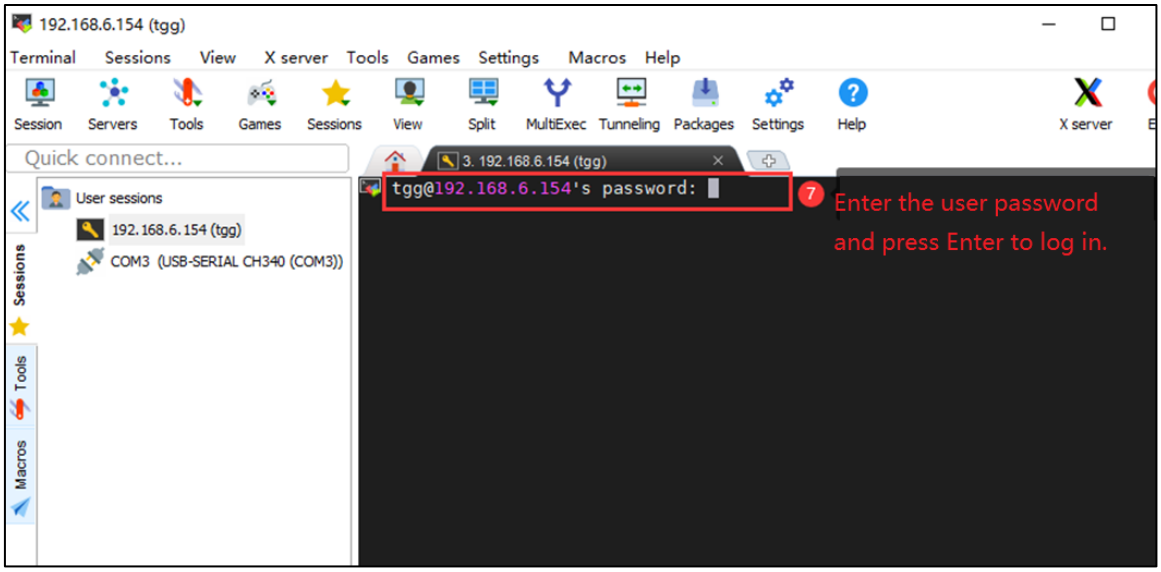


Figure 2.7.3.8 Establishing an SSH remote connection (3)

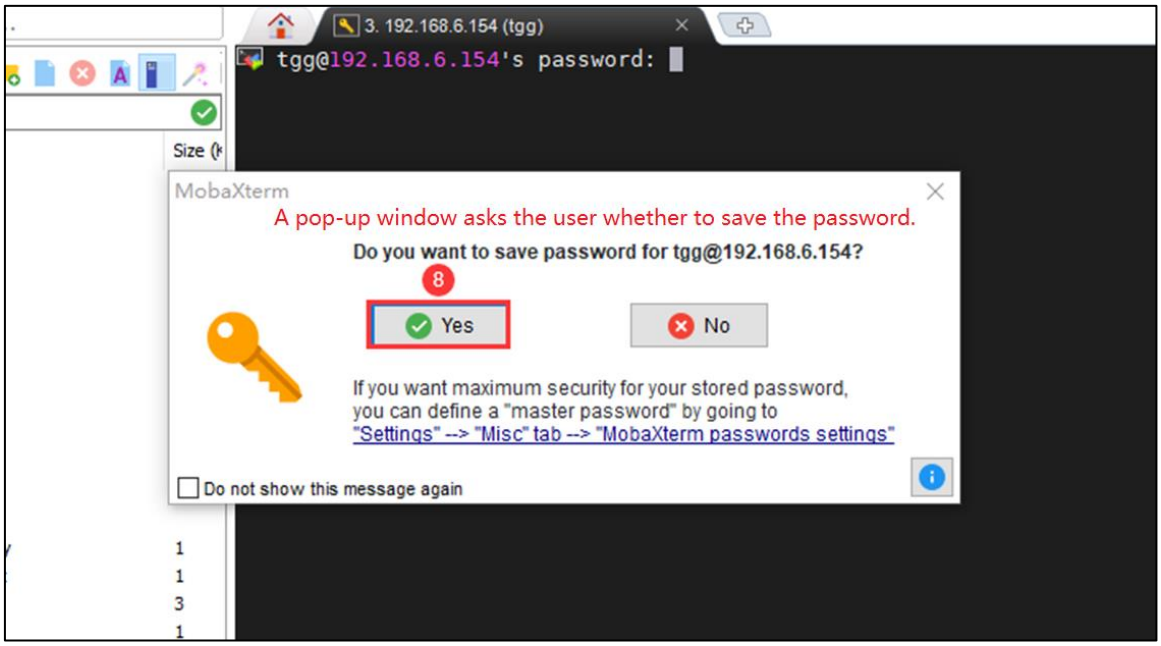


Figure 2.7.3.9 Establishing an SSH remote connection (4)

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:

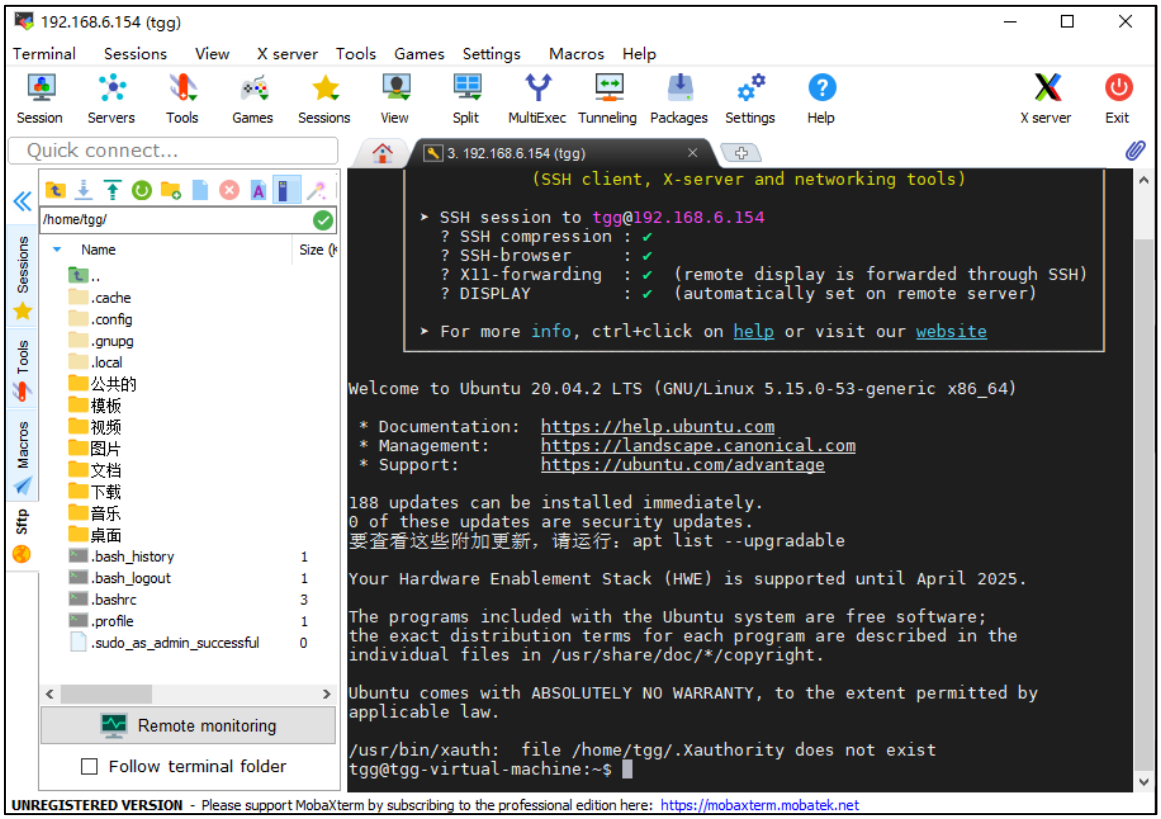


Figure 2.7.3.10 SSH remote login was successful.

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:

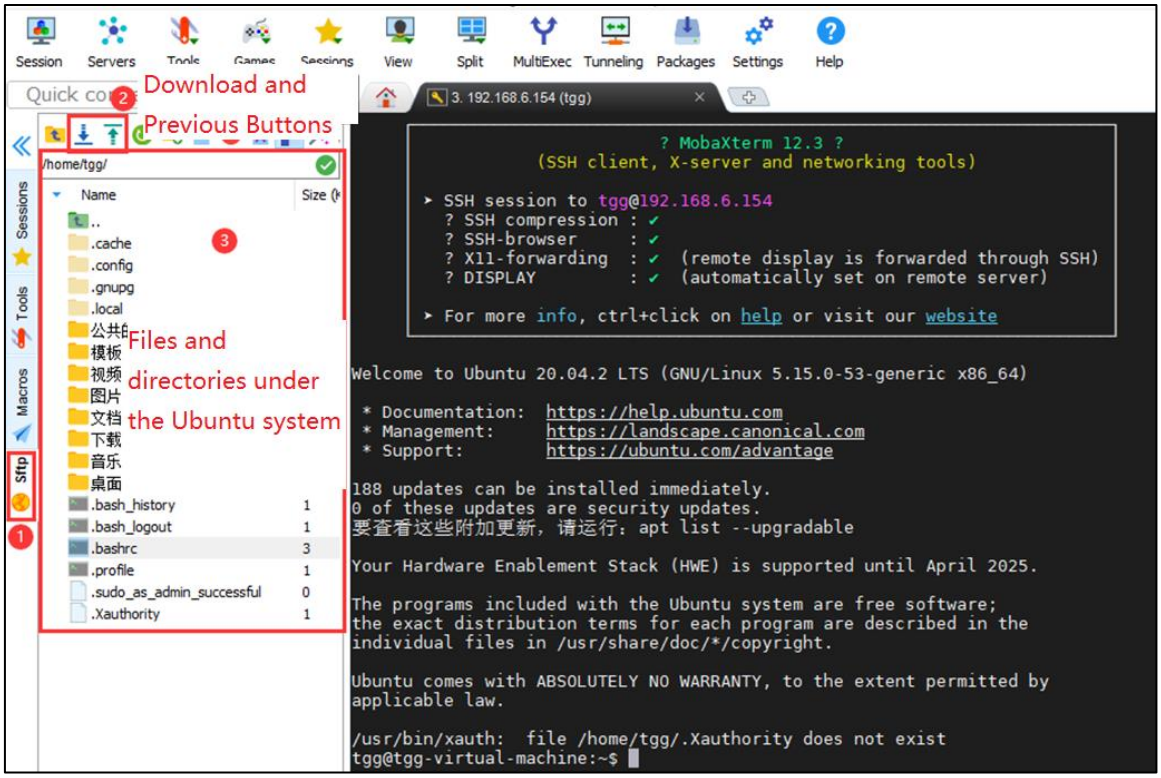


Figure 2.7.3.11 MobaXterm enables file transfer between Windows and Ubuntu

First, select the "**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 Rockchip USB Driver Installation

During the development and debugging stage, it is often necessary to switch the device to **Loader** mode or **Maskrom** mode. For example, when using the Rockchip development tools (RKDevTool) to burn the image (connect a USB cable is required), the device needs to operate in Loader mode or Maskrom mode, and only then can the burning process be carried out; to be able to recognize the device, the Rockchip USB driver needs to be installed on Windows.

The Rockchip USB driver installation package has been provided to users in the development board package. 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, the file and directory list as shown in Figure 2.8.1 will be obtained:

Figure 2.8-1 Unzip DriverAssistant_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:

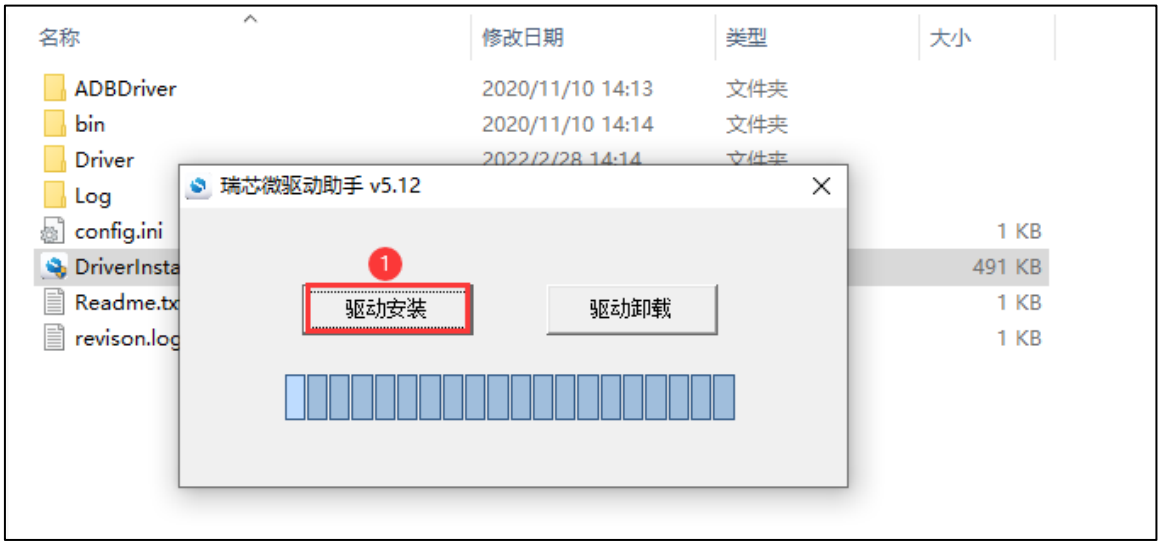


Figure 2.8-2 Install Rockchip USB driver (1)

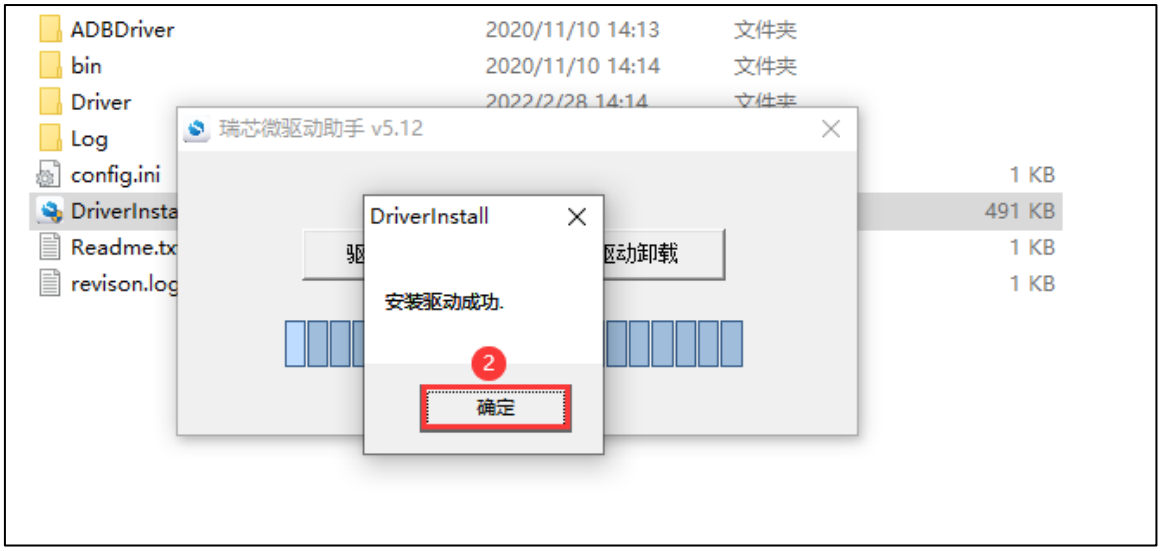


Figure 2.8-3 Install Rockchip USB driver (2)

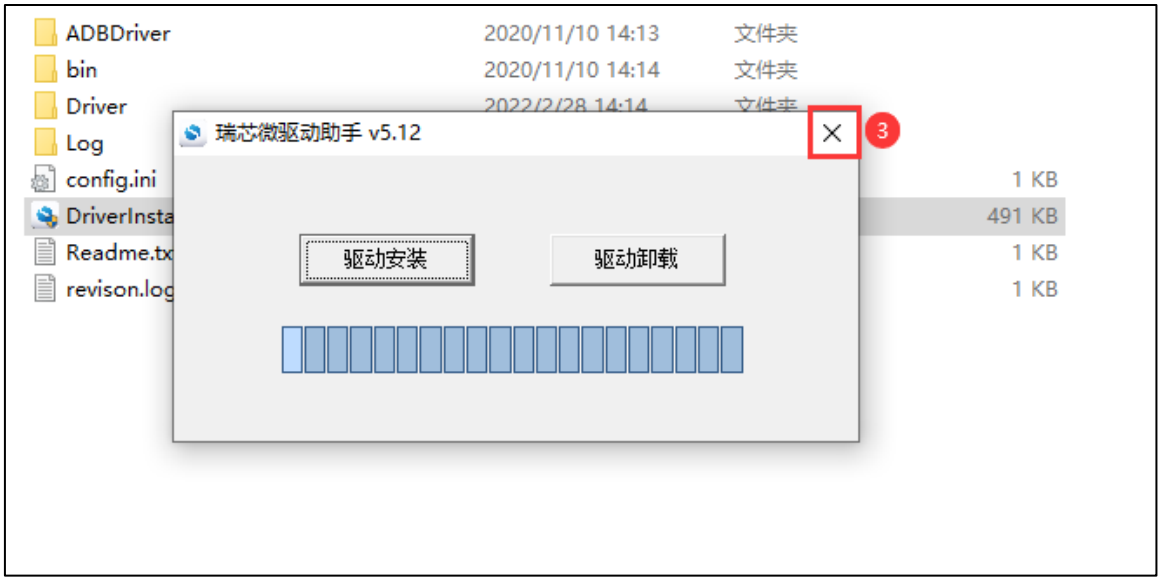


Figure 2.8-4 Install Rockchip USB driver (3)

2.9 Usage of Mirror Image Burning Tools

The Rockchip platform offers various 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; in 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 possible to burn a specific partition image file separately using the burning tool, which is convenient for debugging.

Recovery	Not supported	By booting into the recovery mode with the recovery image, the main function of the recovery mode is to perform operations such as upgrading and restoring factory settings.
Normal Boot	Not supported	When the system starts normally and enters this mode, it is started by booting the rootfs, loading the root file system, and most of the development is carried out in this mode.

Table 2.9.1.1 Rockchip Platform Operating Mode Description

The method to enter the Maskrom programming mode: (Connect the power adapter and OTG first)

- 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 is as follows: (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 Linux 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.

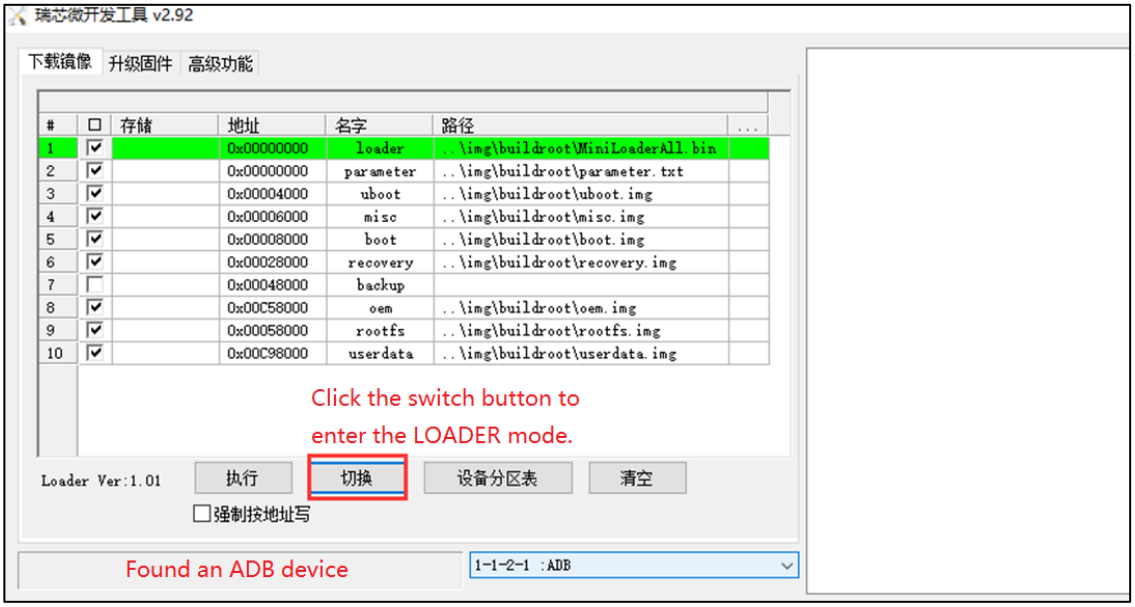


Figure 2.9.1.1 Enter the Loader mode through the development tools of Rockchip Microelectronics.

The above has introduced how the development board can enter the Maskrom or Loader programming mode. Only after entering the Maskrom mode or Loader mode can the programming be carried out.

2.9.2 Using the RKDevTool Tool on Windows

When using the RKDevTool tool to burn the image on Windows, it is also referred to as the Rockchip Development Tool in the documentation of Rockchip. This tool is not only used for burning the image, but also has other functions, which are the most commonly used tools in our development and debugging process.

The ATK-DLRK3568 development board data package provided by ALIENTEK already includes this tool. 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 Rockchip 3568 Linux SDK.

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

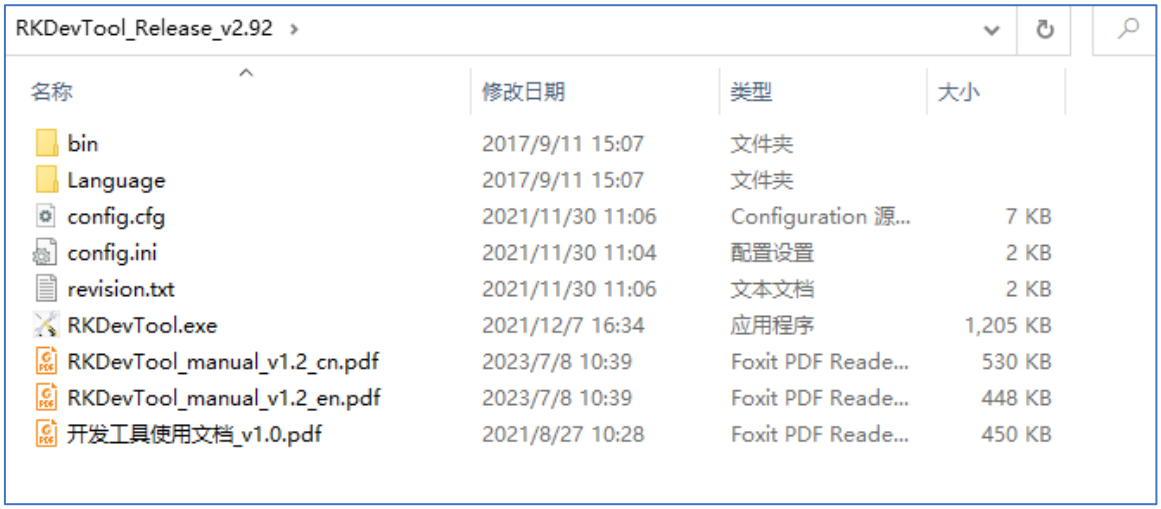


Figure 2.9.2.1 RKDevTool Directory

There are two documents in this directory: "**Developer Tools Usage Document_v1.0.pdf**" and "**RKDevTool_manual_v1.2_cn.pdf**". These two documents provide users with instructions on how to use the RKDevTool tool (Rockchip Development Tool), which is provided by RK officially; For detailed usage methods of the RKDevTool tool, please refer to these two documents. This document only gives a simple introduction.

Double-click the **RKDevTool.exe** executable file to open the Rockchip Development Tool, as shown in the following figure:

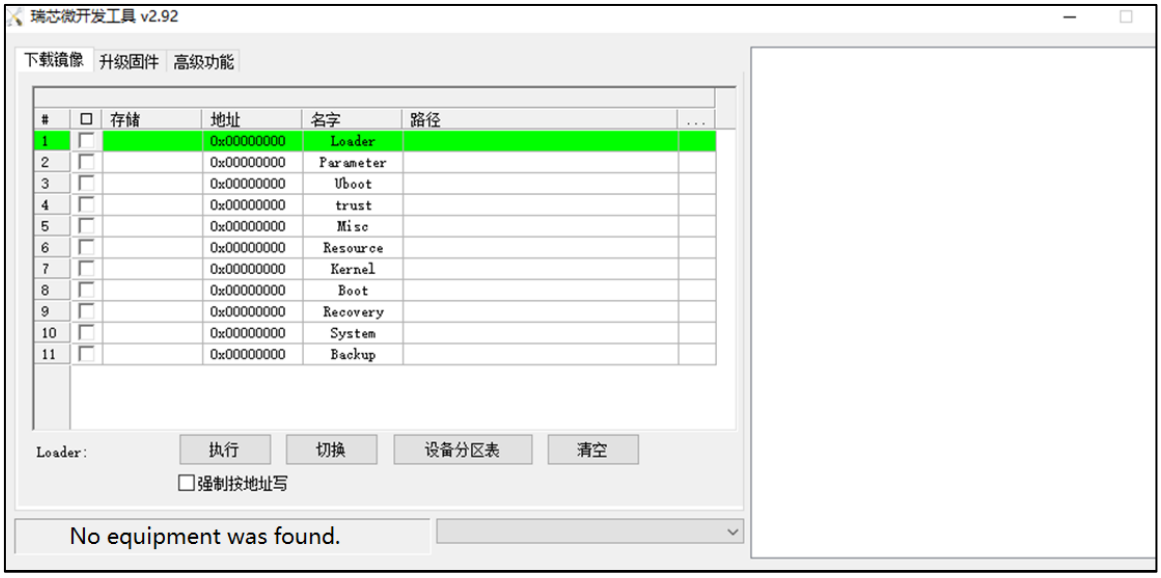


Figure 2.9.2.4 The interface of the Renesas Micro development tool

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

- **Burn each image separately.** After the Linux SDK of RK3568 is compiled successfully, multiple image files will be generated, including: boot.img, MiniLoaderAll.bin, uboot.img, misc.img, rootfs.img, recovery.img, userdata.img, and OEM.img, totaling 8 image files, among which there is also a partition table file named 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 boot.img, MiniLoaderAll.bin, uboot.img, misc.img, rootfs.img, and recovery.img. These individual images are combined to form the update.img firmware package. This can be done using the tools provided by RK.

Next, let's talk about how to burn each individual image. 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 image burning address, the path where the image is located, etc. Of course, you can also manually configure these information, and we will explain it later):

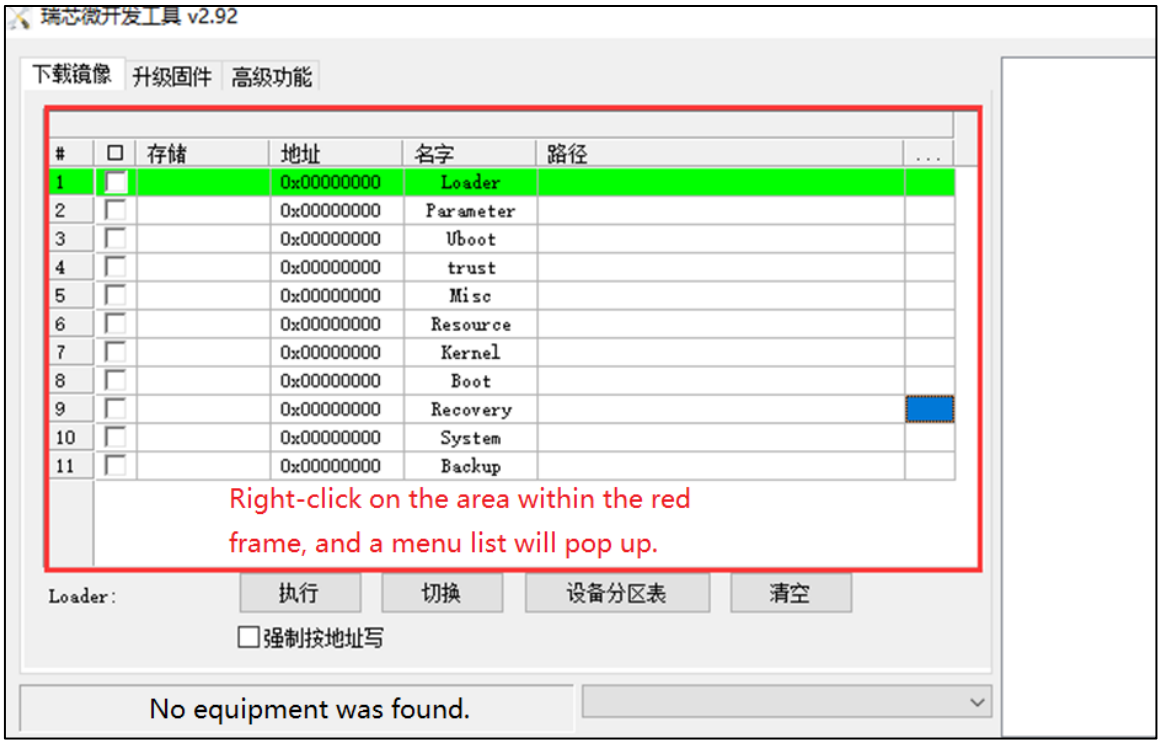


Figure 2.9.2.5 Import configuration file (1)

Figure 2.9.2.6 Import configuration file (2)

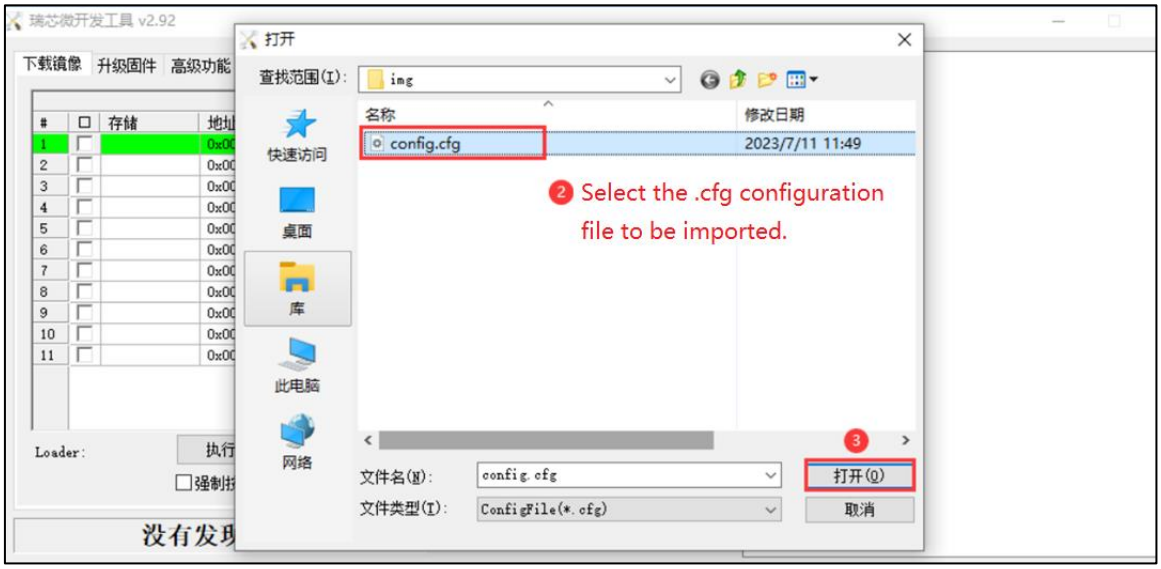


Figure 2.9.2.7 Import configuration file (3)

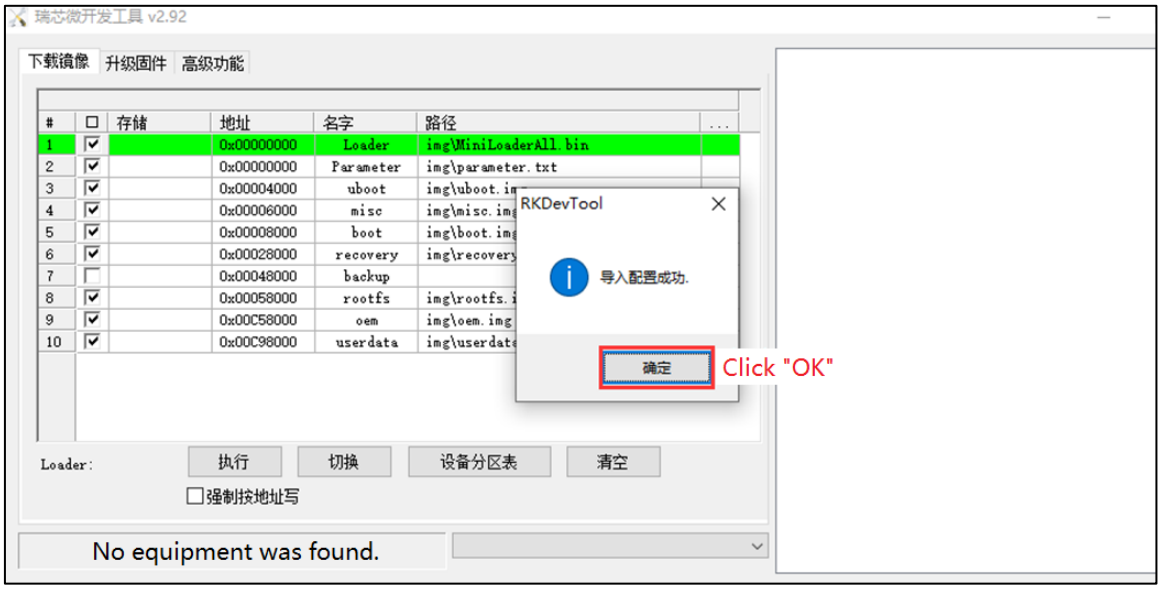


Figure 2.9.2.8 Importing configuration file (4)

At this point, we have imported a burning configuration file (importing configuration file).

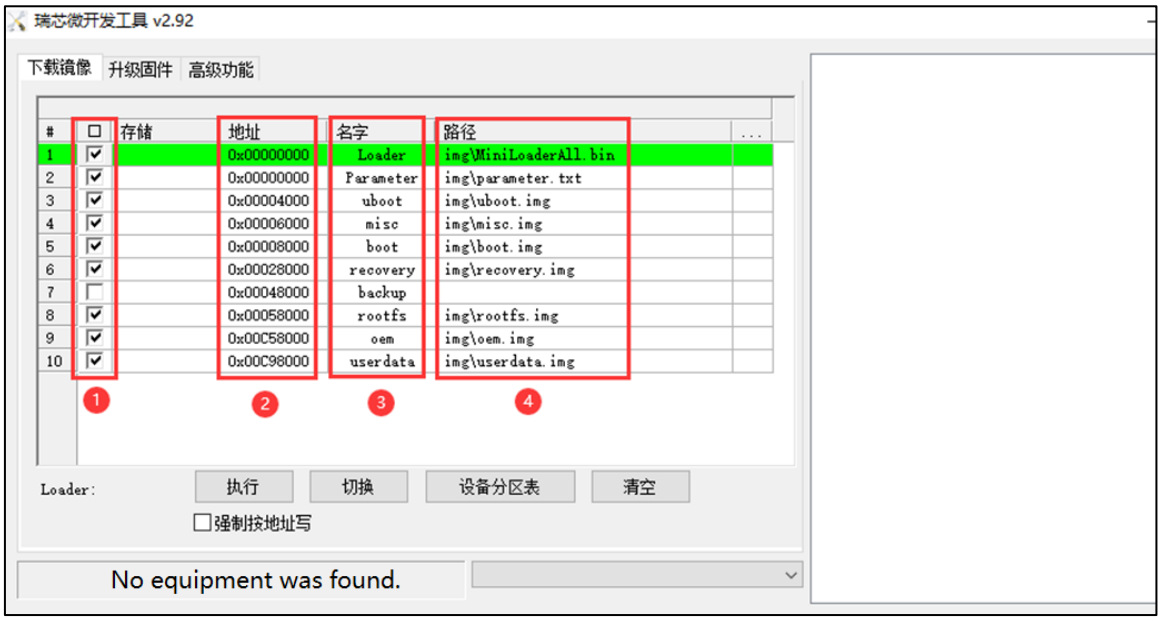


Figure 2.9.2.9 Configuration Information Division Explanation

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 should not 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 boards that customers have obtained are usually already burned with the image, and they can enter Loader mode, or they 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:

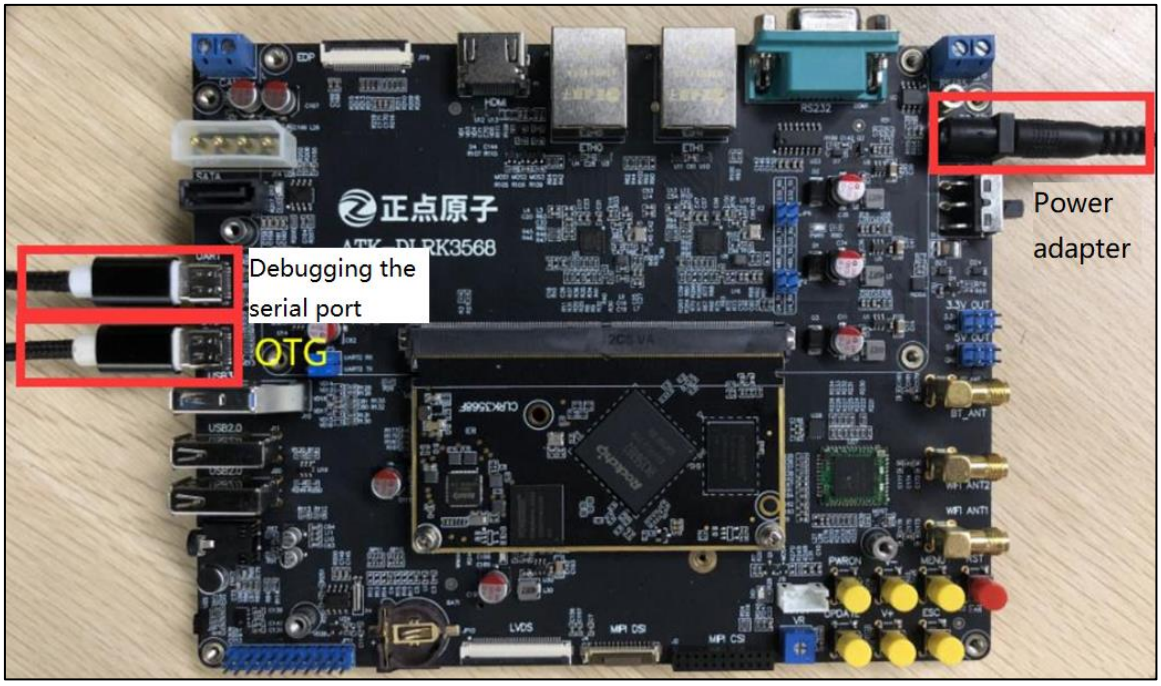


Figure 2.9.2.10 Hardware connection diagram for the development board

It is recommended to use the 12-2.5A power adapter provided with the ATK-DLRK3568 development board; connect the UART debugging serial port of the development board to the computer via a USB cable (one end is a USB interface, the other end is a Type-C interface), and also connect the OTG port of the development board to the computer using a USB cable (one end is a USB interface, the other end is a Type-C interface).

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

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

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

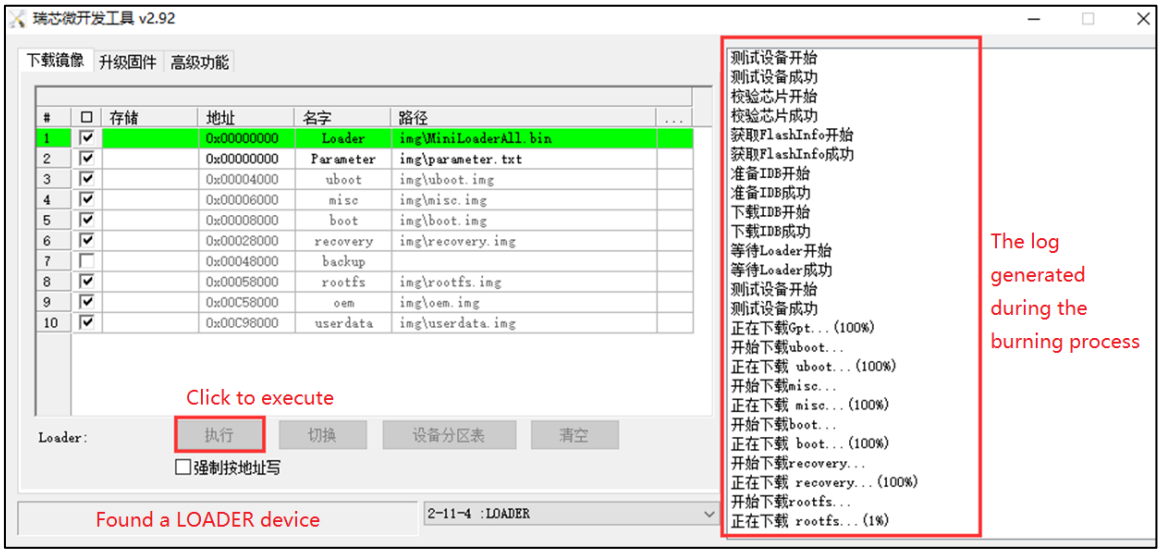


Figure 2.9.2.12 Perform the burning operation.

Of course, it can also be done in Maskrom mode. In Section 2.9.1, we have already introduced how to enter the Maskrom mode. After entering the Maskrom mode, the same "**Execute**" button should be clicked to perform the burning.

If there are no unexpected issues, the burning 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 started through the SD card**). If unfortunately, an error occurs during the burning process, causing the burning

to fail, then you can refer to the official RK document "**RKDevTool_manual_v1.2_cn.pdf**", which contains explanations for several common errors.

After the burning is completed, the development board will automatically restart. Connect the debugging serial port of the development board through the MobaXterm software, and you can view the log information during the system startup process.

2.9.3 The method for burning the update.img image

The update.img image is a collection of multiple images (formed by packaging and merging multiple images). Using the tools provided by RK, each separate image (such as uboot.img, boot.img, MiniLoaderAll.bin, parameter.txt, misc.img, rootfs.img, OEM.img, userdata.img, recovery.img) can be packaged into a single update.img firmware, making it convenient for users to burn and upgrade.

This section explains how to use the Rockchip development tools to burn the "update.img". First, open the Rockchip development tools and select "**Upgrade Firmware**".

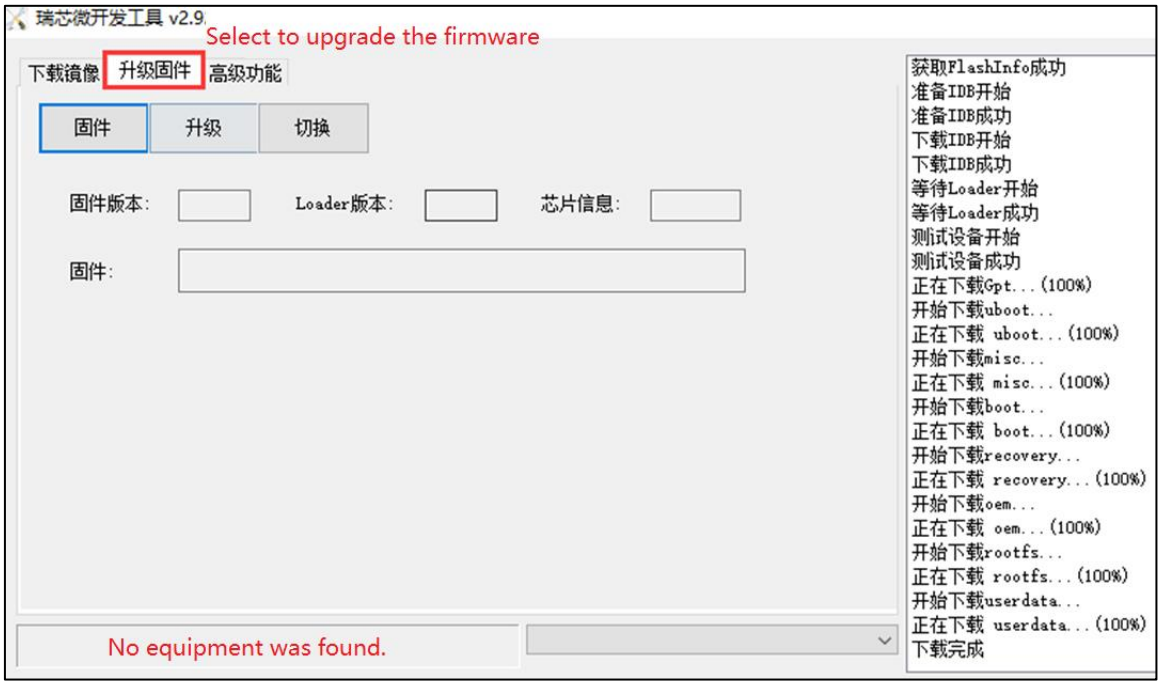


Figure 2.9.3.1 Upgrade Firmware Tab

Then click the "**Firmware**" button to select the update.img firmware that we need to upgrade and burn (the update.img firmware is not provided to users in the development board package. Here, we are only demonstrating the method of burning the update.img firmware to show users how it works. Later, during the development process, users can practice and operate it themselves). After importing the update.img firmware, some information about this firmware will be displayed:

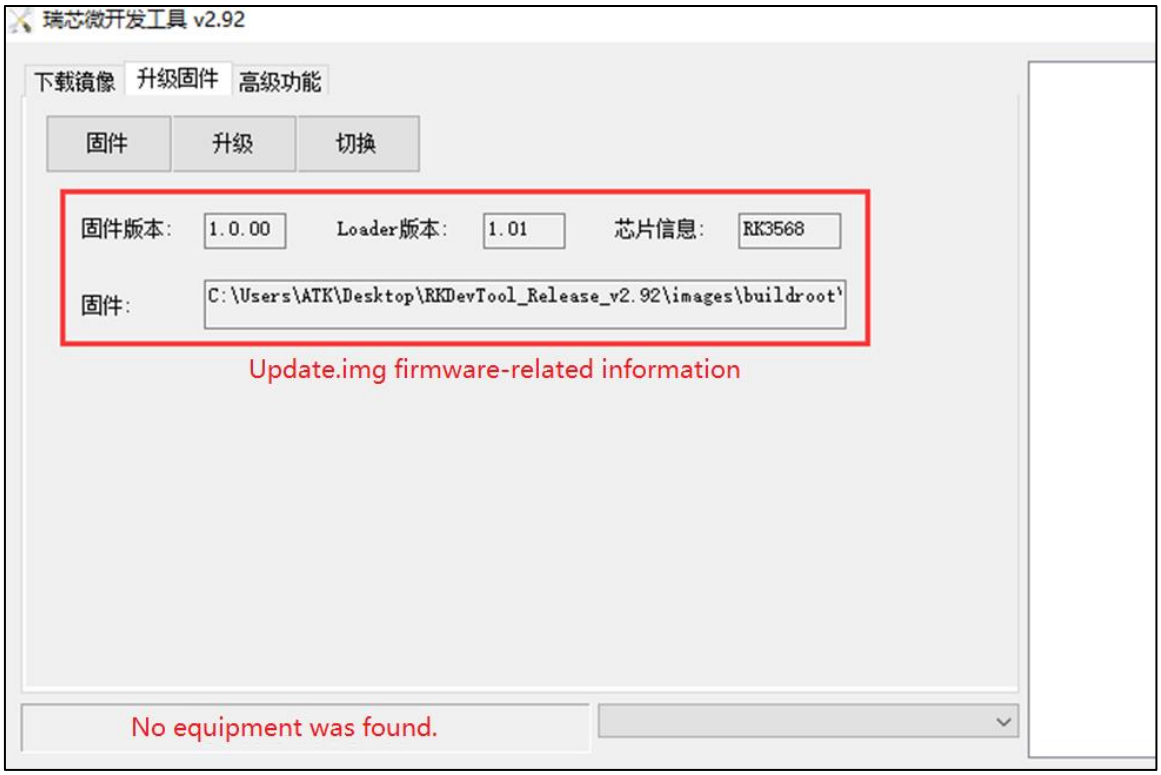


Figure 2.9.3.2 Import the update.img image

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

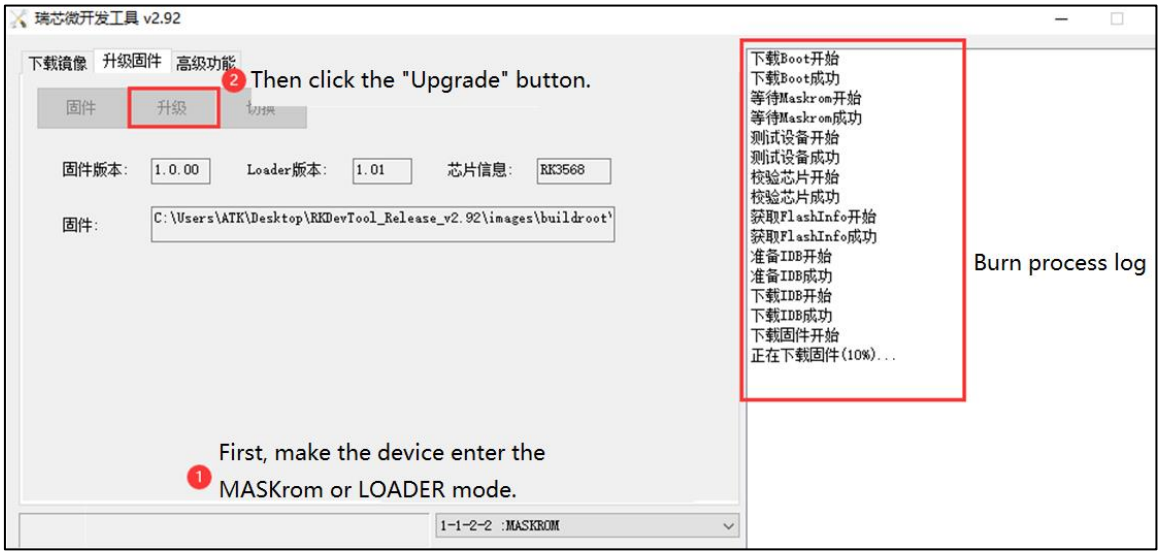


Figure 2.9.3.3 update.img upgrade

After the firmware burning is completed, the development board will automatically restart.

2.9.4 Erase operation

The Flash can be erased using the Rockchip development tools, which will erase the image that has been burned into the Flash. Open the Rockchip development tools, click on the "**Advanced Function**" 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 stages, the source code is usually compiled under Ubuntu, and the compiled images are also present in the Ubuntu system. Before burning the images using the RKDevTool development tool, these image files need to be copied to the Windows system first, and then the burning process can be carried out. 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 in the SDK. The SDK comes with this burning tool, and it will be introduced to you in **Chapter 5**.

2.10 Installation of ADB tools

The full name of ADB is 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, ADB enables developers to communicate with Android devices via the network or USB, 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 through 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 port mapping of the emulator or device;
- File upload/download between the computer and the device;
- Installing local apk software to the emulator or Android device.

It should be noted that although ADB is a debugging tool for Android devices, with the widespread use of ADB, not only Android devices but also many Linux devices in embedded development also support ADB debugging, such as the Rockchip platform.

This document does not provide an introduction to the installation and usage methods of the ADB tool. Users can refer to another document provided by ALIENTEK, "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**. This document will guide users on how to install the ADB tool in Windows systems and Ubuntu systems, as well as how to use the ADB tool.

Chapter 3. Introduction to the ALIENTEK ATK-DLRK3568 Platform

Reference Document:

"[ALIENTEK] ATK-DLRK3568 Development Board Hardware Reference Manual.pdf"

3.1 Introduction to RK3568

Reference Document:

"[ALIENTEK] ATK-DLRK3568 Development Board Hardware Reference Manual.pdf"
Chapter 1

3.2 Introduction to the Hardware Resources of ALIENTEK ATK-RK3568 Development Board

Reference Document:

"[ALIENTEK] ATK-DLRK3568 Development Board Hardware Reference Manual.pdf"
Chapter 2

Chapter 4. RK3568 Linux SDK software package

This chapter introduces to users the installation and usage methods of the Linux SDK software package for the ALIENTEK ATK-DLRK3568 hardware platform. This SDK is applicable to the ALIENTEK ATK-DLRK3568 development board and all Linux products developed based on this board. Based on this SDK, it is possible to effectively achieve system customization and application porting development, helping users to quickly develop and improve their development efficiency!

The Linux SDK supports three root file systems: Buildroot, Yocto, and Debian. 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 Linux SDK

4.2 Introduction to the SDK software architecture

4.3 Full-automatic compilation of the SDK

4.4 Independent compilation

4.5 Cleaning of the SDK

4.6 Introduction to the image

Official reference document of RK:

```
<SDK>/docs/Rockchip_Developer_Guide_Linux_Software_CN.pdf
```

Development board CD-ROM A disc - **Basic Information →8_RK official_documents→Linux Rock-chip_Developer_Guide_Linux_Software_CN.pdf**

It is highly recommended that everyone take a look at this!

4.1 Install RK3568 Linux SDK

This section provides users with instructions on how to install the ATK-DLRK3568 development board SDK software package for the ALIENTEK 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 curl python2.7 python-pyelftools git ssh make gcc libssl-dev liblz4-tool expect \
g++ patchelf chrpath gawk texinfo chrpath diffstat binfmt-support qemu-user-static live-build bison \
flex fakeroot cmake gcc-multilib g++-multilib unzip device-tree-compiler python3-pip \
libncurses-dev python3-pyelftools vim mtd-utils
```

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 in the RK3568 Linux SDK are 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 site 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
```

```
tgg@tgg-virtual-machine:~$ curl https://mirrors.tuna.tsinghua.edu.cn/git/git-repo -o ~/bin/repo
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 45769  100 45769    0     0   108k      0 --:--:-- --:--:-- --:--:--  107k
tgg@tgg-virtual-machine:~$ chmod a+x ~/bin/repo
tgg@tgg-virtual-machine:~$
```

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]
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 Linux 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_linux_release_v1.0_20230620.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_linux_release_ version _ release date.tgz**. Each time a new version of the SDK is released, the old version SDK in the network drive will be replaced. If users need to obtain the old version SDK, please contact the ALIENTEK Linux technical support for it!

Copy the tgz compressed file to the home directory of the Ubuntu system user, and execute the following command to decompress it into the rk3568_linux_sdk directory in the home directory:

```
mkdir ~/rk3568_linux_sdk
tar xvf atk-rk3568_linux_release_v1.0_20230620.tgz -C ~/rk3568_linux_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_linux_release_v1.0_20230620.tgz  bin
tgg@tgg-virtual-machine:~$
tgg@tgg-virtual-machine:~$ mkdir ~/rk3568_linux_sdk
tgg@tgg-virtual-machine:~$ tar xvf atk-rk3568_linux_release_v1.0_20230620.tgz -C ~/rk3568_linux_sdk
```

Figure 4.1.4.1 Unzipping the SDK package

After the unzipping is completed, a .repo folder will exist in the ~/rk3568_linux_sdk/ directory, as shown in Figure 4.1.4.2:

```
tgg@tgg-virtual-machine:~$
tgg@tgg-virtual-machine:~$ ls -lha ~/rk3568_linux_sdk/
总用量 12K
drwxrwxr-x  3 tgg tgg 4.0K 12月  1 17:12 .
drwxr-xr-x 16 tgg tgg 4.0K 12月  1 17:12 ..
drwxrwxr-x  7 tgg tgg 4.0K 11月 30 21:03 .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_linux_sdk/
.repo/repo/repo sync -l -j10
```

```
tgg@tgg-virtual-machine:~$ cd ~/rk3568_linux_sdk/
tgg@tgg-virtual-machine:~/rk3568_linux_sdk$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk$ .repo/repo/repo sync -l -j10
正在更新文件: 100% (17078/17078), 完成.更新文件:   2% (400/17078)
正在更新文件: 100% (127/127), 完成.
正在更新文件: 100% (119/119), 完成.
正在更新文件: 100% (1037/1037), 完成.
正在更新文件: 100% (368/368), 完成.
正在更新文件: 100% (33/33), 完成.
正在更新文件: 100% (782/782), 完成.
正在更新文件: 100% (72653/72653), 完成.
正在更新文件: 100% (1176/1176), 完成.
正在更新文件: 100% (5939/5939), 完成.
正在更新文件: 100% (7165/7165), 完成.
正在更新文件: 100% (5722/5722), 完成.
正在更新文件: 100% (7085/7085), 完成.
正在更新文件: 100% (403/403), 完成.
正在更新文件: 100% (236/236), 完成.
正在更新文件: 100% (272/272), 完成.ls正在更新文件:  81% (221/272)
正在更新文件: 100% (275/275), 完成.
正在更新文件: 100% (1493/1493), 完成.
正在更新文件: 100% (188/188), 完成.
Checking out: 100% (72/72), done in 4m11.232s
repo sync has finished successfully.
tgg@tgg-virtual-machine:~/rk3568_linux_sdk$
```

Figure 4.1.4.3 Check out the source code

After the synchronization is completed, the contents in the ~/rk3568_linux_sdk/ directory are as shown in Figure 4.1.4.4:

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk$ ls
app        debian   envsetup.sh  Makefile      rkbin       u-boot
buildroot  device   external     mkfirmware.sh rkflash.sh  yocto
build.sh   docs     kernel       prebuilts     tools
tgg@tgg-virtual-machine:~/rk3568_linux_sdk$
```

Figure 4.1.4.4 SDK Project Directory

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 package file. The naming format of the compressed package file is **atk-rk3568_linux_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 version of the 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!

Each SDK version update will be recorded in the **SDK/.repo/manifests/rk356x_linux/ATK-RK3568_Linux_SDK_Note.md** document. If you want to know the SDK version update record, 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, if users encounter any issues with the SDK, software bugs, specific technical problems, technical inquiries, etc., they can contact the ALIENTEK Linux

technical support through the QQ group or Taobao and explain the problem to them. Subsequently, developers will handle and track the issue accordingly.

4.1.7 SDK Optimization

If the disk space of the Ubuntu system is tight, the SDK can be optimized by deleting the .repo (a hidden folder that can be seen using ls -a) folder under the root directory of the SDK source code; note that the .repo folder stores the git commit information of all git repositories of the SDK. Once deleted, it will be impossible to view the git commit information of any repository in the future. Therefore, it is not recommended to delete it.

4.2 Introduction to SDK Software Architecture

This section provides an introduction to the SDK software architecture for users.

4.2.1 Introduction to SDK Project Directory Structure

The root directory of the SDK source code contains multiple directories such as app, buildroot, debian, device, external, tools, u-boot, and yocto. Each directory or its subdirectories corresponds to a git project; since the code and related documentation of the SDK are divided into several git repositories for version management (thus the SDK actually contains several git repositories), they are organized into different directories according to their functions and modules.

- app: Stores upper-level application apps, including Qt applications and other C/C++ applications.
- .buildroot: Root file system developed based on buildroot.
- .debian: Root file system developed based on Debian.
- .device/rockchip: Stores board-level configuration files and Parameter partition table files for each chip, as well as some scripts and preparatory files for compiling and packaging firmware.
- .docs: Stores development guidance documents for chip modules, platform support lists, chip platform-related documents, Linux development guides, etc.
- .external: Stores required third-party libraries, including audio, video, network, recovery, etc.
- .kernel: Source code of the Linux 4.19 version kernel.
- .prebuilts: Stores cross-compilation toolchains.
- .rkbin: Stores Rockchip-related Binary and tools.
- .rockdev: Stores compiled firmware output, and this folder will be generated after compiling the SDK.
- .tools: Stores commonly used tools in Linux and Windows operating systems environments, including image burning tools, SD card upgrade boot production tools, batch burning tools, etc. For example, the RKDevTool tool and the Linux_Upgrade_Tool tool introduced earlier are all stored in this directory.
- .u-boot: Uboot source code developed based on v2017.09 version.
- .yocto: Root file system developed based on Yocto.

4.2.2 SDK Software Block Diagram

The SDK software block diagram is shown in Figure 4.2.2.1. It is divided from bottom to top into four levels: Bootloader, Linux Kernel, Libraries, and Applications. The contents of each level are as follows:

- The BootLoader layer mainly provides underlying system support packages, such as BootLoader, U-Boot, and ATF-related support.
- The Kernel layer mainly provides the standard implementation of the Linux Kernel. Linux is also an open operating system. The Linux core of the Rockchip platform is the standard Linux 4.4/4.19/5.10 kernel, providing basic support such as security, memory management, process management, and network protocol stack; mainly through the Linux kernel to manage device hardware resources, such as CPU scheduling, cache, memory, I/O, etc.
- The Libraries layer corresponds to general embedded systems and serves as the middleware layer. It includes various system basic libraries and third-party open-source program libraries, providing API interfaces for the application layer. System customizers and application developers can develop new applications based on the API of the Libraries layer.
- The Applications layer mainly implements specific product functions and interaction logic. It requires some system basic libraries and third-party program libraries for support. Developers can develop their own applications and provide various system capabilities to end users.

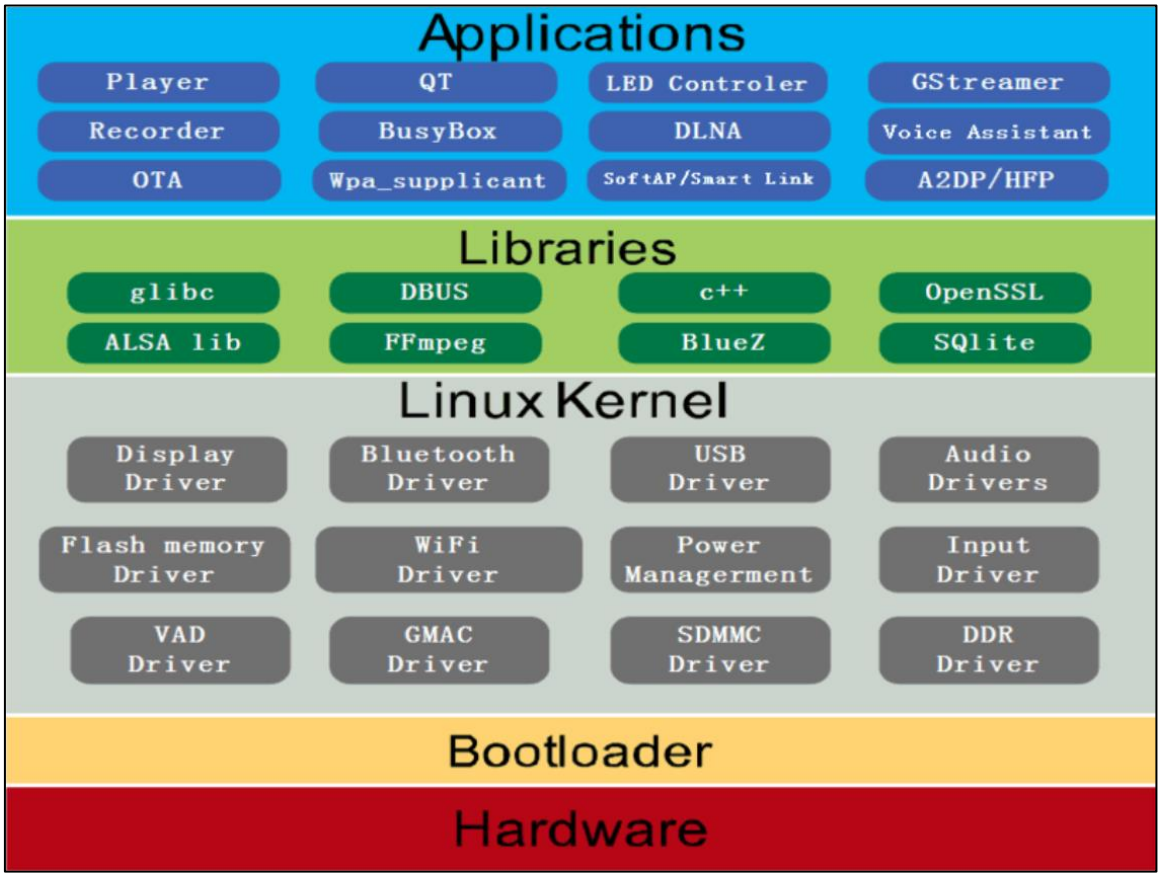


Figure 4.2.2.1 SDK Software Block Diagram

4.2.3 Query of SDK Version

The version of the ALIENTEK RK3568 Linux SDK can be divided into RK version and ATK version; the RK version indicates that this SDK is a secondary development based on a certain version of the official RK Linux SDK from Rockchip (ALIENTEK); 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 that the ALIENTEK Linux technical team defines the internal version number of the RK3568 Linux SDK. Each released SDK will have a corresponding version number.

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

```
realpath .repo/manifests/rk3568_linux_release.xml
```

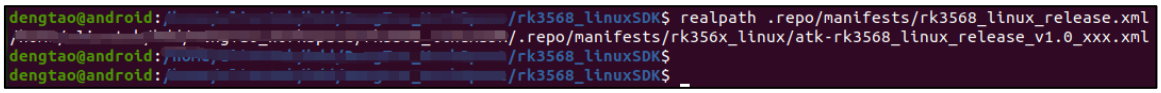


Figure 4.2.3.1 Query the ATK version of the SDK

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

```
ls .repo/manifests/rk356x_linux/rk356x_linux_release*
```

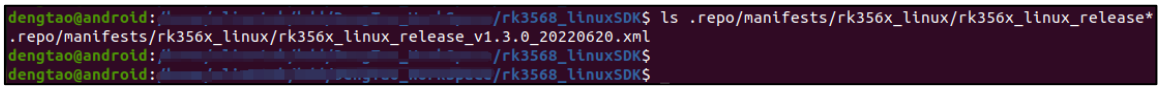


Figure 4.2.3.2 Query the RK version of the SDK

As can be seen from the figure, the current RK version of the SDK is V1.3.0, and the release date is June 20, 2022.

4.3 SDK Automatic Compilation

This section introduces to users how to compile the RK3568 Linux SDK. First, enter the root directory of the SDK source code. Before compiling, execute the following command to specify the board configuration file of the SDK

```
./build.sh lunch
Or ./build.sh BoardConfig-rk3568-atk-evb1-ddr4-v10.mk    (Suggested to use)
```

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk$ ./build.sh lunch
processing option: lunch

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

0. default BoardConfig.mk
1. BoardConfig-rk3566-evb2-lp4x-v10-32bit.mk
2. BoardConfig-rk3566-evb2-lp4x-v10.mk
3. BoardConfig-rk3568-atk-evb1-ddr4-v10.mk
4. BoardConfig-rk3568-evb1-ddr4-v10-32bit.mk
5. BoardConfig-rk3568-evb1-ddr4-v10-spi-nor-64M.mk
6. BoardConfig-rk3568-evb1-ddr4-v10.mk
7. BoardConfig-rk3568-nvr-spi-nand.mk
8. BoardConfig-rk3568-nvr.mk
9. BoardConfig-rk3568-uvc-evb1-ddr4-v10.mk
10. BoardConfig.mk
Which would you like? [0]: 3
```

The board-level configuration file corresponding to the ATK-DLRK3568 development board

The corresponding serial number of the board-level configuration file

Figure 4.3-1 Execute "build.sh lunch" to select the board-level configuration file.

Special note! Due to the SDK update, the number of the configuration named "**BoardConfig-rk3568-atk-evb1-ddr4-v10.mk" might not be number 3!!!!!!** Therefore, it is recommended to use "./build.sh BoardConfig-rk3568-atk-evb1-ddr4-v10.mk". This way, you don't have to worry about the sequence number!

Enter the corresponding sequence number of the board-level configuration file and press Enter to confirm:

```
Which would you like? [0]: 3
switching to board: /home/tgg/rk3568_linux_sdk/device/rockchip/rk356x/BoardConfig-rk3568-atk-evb1-ddr4-v10.mk
tgg@tgg-virtual-machine:~/rk3568_linux_sdk$
```

Figure 4.3-2 Confirm the selection of the corresponding board configuration file

build.sh is a compilation script provided by RK. By using this script, users can conveniently quickly build various image files and perform packaging operations on the images. It can either automatically compile the entire SDK in one step or separately compile U-Boot, Linux Kernel, buildroot, etc., which is very convenient!

The build.sh script is actually a symbolic link file, and it actually points to the device/rockchip/common/build.sh file, as shown in the following figure:

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk$ ls -l build.sh
lrwxrwxrwx 1 tgg tgg 31 4月   6 15:32 build.sh -> device/rockchip/common/build.sh
tgg@tgg-virtual-machine:~/rk3568_linux_sdk$
```

Figure 4.3-3 The build.sh symbolic link file

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

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

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk$ ./build.sh -h
Usage: build.sh [OPTIONS]
Available options:
BoardConfig*.mk    -switch to specified board config
lunch              -list current SDK boards and switch to specified board config
uboot              -build uboot
uefi             -build uefi
spl                -build spl
loader             -build loader
kernel             -build kernel
modules            -build kernel modules
toolchain          -build toolchain
rootfs             -build default rootfs, currently build buildroot as default
buildroot          -build buildroot rootfs
ramboot            -build ramboot image
multi-npu_boot     -build boot image for multi-npu board
yocto              -build yocto rootfs
debian             -build debian rootfs
pcba               -build pcba
recovery           -build recovery
all                -build uboot, kernel, rootfs, recovery image
cleanall           -clean uboot, kernel, rootfs, recovery
firmware           -pack all the image we need to boot up system
updateimg          -pack update image
otapackage         -pack ab update otapackage image (update_ota.img)
sdpackage          -pack update sdcard package image (update_sdcard.img)
save               -save images, patches, commands used to debug
allsave            -build all & firmware & updateimg & save
check              -check the environment of building
info               -see the current board building information
app/<pkg>          -build packages in the dir of app/*
external/<pkg>     -build packages in the dir of external/*

createkeys         -create secureboot root keys
security_rootfs    -build rootfs and some relevant images with security paramter (just for dm-v)
security_boot      -build boot with security paramter
security_uboot     -build uboot with security paramter
security_recovery  -build recovery with security paramter
security_check     -check security paramter if it's good

Default option is 'allsave'.
tgg@tgg-virtual-machine:~/rk3568_linux_sdk$
```

Figure 4.3-4 View the help information of the build.sh script

As can be seen from the figure, the build.sh script supports a large number of parameters. However, the ones commonly used during the development process are only a few, as shown in the following table:

build.sh Script parameters	Explanation	Example
lunch	Select board configuration file	./build.sh lunch
uboot	Compile u-boot	./build.sh uboot
kernel	Compile kernel	./build.sh kernel
modules	Compile kernel module	./build.sh modules
rootfs	Compile root file system	./build.sh rootfs
buildroot	Compile buildroot root file system	./build.sh buildroot
debian	Compile Debian root file system	./build.sh debian
recovery	Compile recovery	./build.sh recovery
all	Compile the entire SDK, including u-boot, kernel, rootfs, and recovery	./build.sh all
cleanall	Clean the entire SDK	./build.sh cleanall

firmware	Package the image into the rockdev directory	./build.sh firmware
updateimg	Package all images into a single update.img firmware	./build.sh updateimg

Table 4.3.1 Common Commands of build.sh

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

After selecting the board-level configuration file, you can proceed with the compilation. The most time-consuming part of the entire SDK compilation process is the compilation of the root file system. During the compilation of the root file system, many third-party library files will be downloaded via the network. Firstly, the download process will consume a lot of time, causing the compilation time to be prolonged. Secondly, if the user's network environment is unstable or the download source of the third-party library files changes, it is very likely to result in a download failure, thereby causing errors in the compilation of the root file system. Therefore, in order to speed up the compilation process of the root file system and reduce the probability of problems occurring during the compilation of the root file system, we can pre-copy the third-party library files required for compiling the root file system to the SDK.

The required third-party library files have been provided to users in the ALIENTEK ATK-DLRK3568 development board data package. The specific path is: Development board optical disc B drive - **Development environment and SDK→02, ATK-DLRK3568 development board SDK→dl.tgz**. First, copy this compressed file to the home directory of the Ubuntu system, as shown below:

```
tgg@tgg-virtual-machine:~$
tgg@tgg-virtual-machine:~$ ls -l dl.tgz
-rw-rw-r-- 1 tgg tgg 1205119367 4月   6 17:34 dl.tgz
tgg@tgg-virtual-machine:~$
tgg@tgg-virtual-machine:~$
```

Figure 4.3-5 The dl.tgz compressed file

Execute the following command to extract it to the <SDK>/buildroot directory:

```
tar -xzf dl.tgz -C ~/rk3568_linux_sdk/buildroot/
```

```
tgg@tgg-virtual-machine:~$
tgg@tgg-virtual-machine:~$ tar -xzf dl.tgz -C ~/rk3568_linux_sdk/buildroot/
tgg@tgg-virtual-machine:~$
```

Figure 4.3-6 Extract dl.tgz

After the extraction is completed, you can enter the <SDK>/buildroot directory. There will be a dl directory in this directory, and the files stored in this directory are the third-party library files, as shown below:

```
tgg@tgg-virtual-machine:~$ cd ~/rk3568_linux_sdk/buildroot/
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$ cd dl/
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot/dl$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot/dl$ ls
acl-2.2.52.src.tar.gz                                          iputils-20210722.tar.gz
alsa-lib-1.1.5.tar.bz2                                         iw-4.9.tar.xz
alsa-plugins-1.1.5.tar.bz2                                     jpegsrc.v9b.tar.gz
alsa-ucm-conf-v1.2.6.3.tar.gz                                  keyutils-1.5.10.tar.bz2
alsa-utils-1.1.5.tar.bz2                                       kmod-26.tar.xz
android-tools_4.2.2+git20130218-3ubuntu41.debian.tar.gz        libbsd-0.8.7.tar.xz
android-tools_4.2.2+git20130218.orig.tar.xz                    libdrm-2.4.109.tar.xz
attr-2.4.47.src.tar.gz                                         liberation-fonts-ttf-2.00.1.tar.gz
autoconf-2.69.tar.xz                                           libevdev-1.5.8.tar.xz
automake-1.15.1.tar.xz                                         libevent-2.1.8-stable.tar.gz
bash-5.1.16.tar.gz                                             libffi-3.2.1.tar.gz
binutils-2.36.1.tar.xz                                         libgudev-230.tar.xz
bison-3.0.4.tar.xz                                             libical-1.0.1.tar.gz
bluez-5.50.tar.xz                                              libinput-1.19.3.tar.xz
bluez-alsa-22ffb1965a0b79fbb28af5751b98814f94f6f81d.tar.gz     libjpeg-turbo-2.0.2.tar.gz
busybox-1.34.1.tar.bz2                                         liblockfile_1.09-6.debian.tar.bz2
ca-certificates_20170717.tar.xz                                liblockfile_1.09.orig.tar.gz
cairo-1.16.0.tar.xz                                            libmad-0.15.1b.tar.gz
cantarell-fonts-0.0.25.tar.xz                                  libmpeg2-0.5.1.tar.gz
can-utils-c3305fdd515464153d20199db232b6124bc962c0.tar.gz      libnl-3.4.0.tar.gz
coreutils-8.30.tar.xz                                          libogg-1.3.3.tar.xz
curl-7.75.0.tar.xz                                             libpcap-1.8.1.tar.gz
dbus-1.12.2.tar.gz                                             libpng-1.6.37.tar.xz
dejavu-fonts-ttf-2.37.tar.bz2                                  libpthread-stubs-0.4.tar.bz2
dhcpcd-6.11.5.tar.xz                                           libsndfile-1.0.28.tar.gz
dhry-c                                                         libsoup-2.56.1.tar.xz
```

Figure 4.3-7 The library files under the dl directory

After the preparations are completed, the SDK can be compiled next. Enter the root directory of the SDK source code and execute the following command to compile the entire SDK:

```
./build.sh all
```

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk$ ./build.sh all
processing option: all
============================================
TARGET_ARCH=arm64
TARGET_PLATFORM=rk356x
TARGET_UBOOT_CONFIG=rk3568
TARGET_SPL_CONFIG=
TARGET_KERNEL_CONFIG=rockchip_linux_defconfig
TARGET_KERNEL_DTS=rk3568-atk-evb1-ddr4-v10-linux
TARGET_TOOLCHAIN_CONFIG=
TARGET_BUILDROOT_CONFIG=rockchip_rk3568
TARGET_RECOVERY_CONFIG=rockchip_rk356x_recovery
TARGET_PCBA_CONFIG=
TARGET_RAMBOOT_CONFIG=
============================================
============Start building uboot============
TARGET_UBOOT_CONFIG=rk3568
========================================
grep: .config: No such file or directory
## 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 between 1 an
  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 4104) in
   36 |   return __builtin___sprintf_chk (__s, __USE_FORTIFY_LEVEL - 1,
      |          ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   37 |        __bos (__s), __fmt, __va_arg_pack ());
```

Figure 4.3-8 One-click automatic compilation of the entire SDK

The entire compilation process will take a very long time. The duration of the compilation process depends on the configuration of your personal computer. It could take as little as one hour or as long as three or four hours, or even longer.

If the compilation fails and similar error messages appear, it indicates that your Ubuntu system has insufficient memory:

```
ead -Wall -U_FORTIFY_SOURCE -D_FORTIFY_SOURCE=2 -Wno-unused-local-typedefs -Wno-maybe-uninitialized -Wno-deprecated-declarations -
-dangling-else -Wno-missing-field-initializers -Wno-unused-parameter -O2 -fno-ident -fdata-sections -ffunction-sections -fno-omit-
oot/output/rockchip_rk3568/host/aarch64-buildroot-linux-gnu/sysroot/usr/include/nss -I/home/tgg/rk3568_linux_sdk/buildroot/output/
 -I/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/host/aarch64-buildroot-linux-gnu/sysroot/usr/include/dbus-1.0 -I/ho
ldroot-linux-gnu/sysroot/usr/lib/dbus-1.0/include -std=gnu++14 -Wno-narrowing -Wno-class-memaccess -Wno-attributes -Wno-class-mema
deprecated-copy -fno-exceptions -fno-rtti --sysroot=../../../../../host/aarch64-buildroot-linux-gnu/sysroot -fvisibility-inlines-h
wser/frame_host/navigation_entry_impl.cc -o obj/content/browser/browser/navigation_entry_impl.o
2023-04-06T19:14:25 aarch64-buildroot-linux-gnu-g++.br_real: fatal error: Killed signal terminated program cc1plus
2023-04-06T19:14:25 compilation terminated.
[17798/20421] CXX obj/content/browser/browser/mixed_content_navigation_throttle.o
2023-04-06T19:21:11 ninja: build stopped: subcommand failed.
2023-04-06T19:21:20 make[5]: *** [Makefile.gn_run:444: run_ninja] Error 1
2023-04-06T19:21:20 make[4]: *** [Makefile:82: sub-gn_run-pro-make_first] Error 2
2023-04-06T19:21:20 make[3]: *** [Makefile:79: sub-core-make_first] Error 2
2023-04-06T19:21:20 make[2]: *** [Makefile:49: sub-src-make_first] Error 2
2023-04-06T19:21:20 make[1]: *** [package/pkg-generic.mk:231: /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/build/qt
2023-04-06T19:21:33 make: *** [/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/Makefile:16: _all] Error 2
Command exited with non-zero status 1
you take 1:27:49 to build builroot
ERROR: Running build_buildroot failed!
ERROR: exit code 1 from line 717:
    /usr/bin/time -f "you take %E to build builroot" $COMMON_DIR/mk-buildroot.sh $BOARD_CONFIG
```

The occurrence of this error indicates insufficient memory.

Figure 4.3-9 Insufficient memory leads to compilation failure.

In such cases, there are two methods that can be attempted to solve it; first, if the hardware conditions permit, directly expand the memory capacity of the Ubuntu system. This is the most direct and effective method; if the hardware conditions do not allow and there is no additional physical memory to allocate to the Ubuntu system, in this situation, we can increase the swap interaction space of the Ubuntu system. By increasing the swap space, we can try to solve the compilation failure caused by insufficient memory.

It is recommended that everyone use the first method to solve this problem, that is, to increase the memory capacity of Ubuntu; if that doesn't work, then try to solve it by increasing the swap space; the default size of the Ubuntu system's swap space is 2G. For example, it can be configured as 8G, 16G, 32G, etc. Regarding how to configure and increase the swap space, this document does not provide explanations. You can search on Baidu to solve it yourself, or you can refer to the document provided by ALIENTEK, titled "**Ubuntu System Expansion of Swap Space.pdf**", located at: Development Board CD-ROM A Disk - **Basic Materials -> 10_user_manual -> 03, Auxiliary Documents -> [ALIENTEK] Ubuntu System Expansion of Swap Space.pdf.**

During the compilation process, in addition to possibly encountering this problem, there may also be other problems. After all, each person's development environment may have some differences to some extent. Therefore, everyone should try to operate according to the instructions in this document, especially the software packages required for installation in the 4.1.1 section.

If no unexpected situation occurs, the entire SDK compilation will be successful, as shown in Figure 4.3.10:

```
  Load Address: 0xffffff02
  Entry Point:  unavailable
  Hash algo:    sha256
  Hash value:   c0443b5c577c14109ad3a52c6c581bd26c0f2421af45d8c29b910b1ec9016e22
 Image 3 (resource)
  Description:  unavailable
  Created:      Thu Apr  6 23:11:26 2023
  Type:         Multi-File Image
  Compression:  uncompressed
  Data Size:    458240 Bytes = 447.50 KiB = 0.44 MiB
  Hash algo:    sha256
  Hash value:   57ea171509199dbfaa27deb1e506cb02fe924d6343ac225e85b32c1460839119
 Default Configuration: 'conf'
 Configuration 0 (conf)
  Description:  unavailable
  Kernel:       kernel
  Init Ramdisk: ramdisk
  FDT:          fdt
done.
you take 25:06.37 to build recovery
Running build_recovery succeeded.
Skipping build_ramboot for missing configs:  RK_CFG_RAMBOOT.
Running build_all succeeded.
tgg@tgg-virtual-machine:~/rk3568_linux_sdk$
```

The entire SDK has been compiled successfully.

Figure 4.3-10 The SDK compilation was successful.

If the string "**Running build_all succeeded**" appears, it indicates that the SDK compilation was successful. So, congratulations everyone!

After the compilation is completed, various images will be generated, including boot.img, uboot.img, MiniLoaderAll.bin, rootfs.img, recovery.img, etc. However, these image files are scattered in their respective source code directories and are not convenient for users to search for. At this time, we can execute the following command to package them into the SDK/rockdev directory:

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

Or simply execute the ./mkfirmware.sh script located in the root directory of the SDK source code (the ./build.sh firmware command actually executes the mkfirmware.sh script):

```
./mkfirmware.sh
```

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk$ ./build.sh firmware
processing option: firmware
/usr/bin/fakeroot
 Source buildroot/build/envsetup.sh
Top of tree: /home/tgg/rk3568_linux_sdk
=========================================

#TARGET_BOARD=rk3568
#OUTPUT_DIR=output/rockchip_rk3568
#CONFIG=rockchip_rk3568_defconfig

=========================================
make: Entering directory '/home/tgg/rk3568_linux_sdk/buildroot'
  GEN     /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/Makefile
/home/tgg/rk3568_linux_sdk/buildroot/build/defconfig_hook.py -m /home/tgg/rk3568_linux_sdk/buildroot/configs/rockchip_rk3
pconfig
BR2_DEFCONFIG='' KCONFIG_AUTOCONFIG=/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/build/buildroot-config/au
3568/build/buildroot-config/autoconf.h KCONFIG_TRISTATE=/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/build
output/rockchip_rk3568/.config HOST_GCC_VERSION="9" BUILD_DIR=/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568
kchip_rk3568_defconfig /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/build/buildroot-config/conf --defconfi
ig.in
/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/.rockchipconfig:256:warning: override: reassigning to symbol
/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/.rockchipconfig:257:warning: override: reassigning to symbol
/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/.rockchipconfig:259:warning: override: reassigning to symbol
/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/.rockchipconfig:284:warning: override: reassigning to symbol
/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/.rockchipconfig:285:warning: override: reassigning to symbol
/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/.rockchipconfig:286:warning: override: reassigning to symbol
/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/.rockchipconfig:292:warning: override: reassigning to symbol
/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/.rockchipconfig:293:warning: override: reassigning to symbol
/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/.rockchipconfig:294:warning: override: reassigning to symbol
/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/.rockchipconfig:297:warning: override: reassigning to symbol
/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/.rockchipconfig:316:warning: override: reassigning to symbol
/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/.rockchipconfig:319:warning: override: reassigning to symbol
#
# configuration written to /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/.config
#
make: Leaving directory '/home/tgg/rk3568_linux_sdk/buildroot'
 Linking parameter.txt from /home/tgg/rk3568_linux_sdk/device/rockchip/rk356x/parameter-buildroot-fit.txt...
 Done linking parameter.txt
 Linking uboot.img from /home/tgg/rk3568_linux_sdk/u-boot/uboot.img...
 Done linking uboot.img
 Linking MiniLoaderAll.bin from /home/tgg/rk3568_linux_sdk/u-boot/rk356x_spl_loader_v1.13.112.bin...
 Done linking MiniLoaderAll.bin
 Linking boot.img from /home/tgg/rk3568_linux_sdk/kernel/boot.img...
 Done linking boot.img
 Linking recovery.img from /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk356x_recovery/images/recovery.img...
```

Figure 4.3-11 Package the image into the "rockdev" directory.

After executing the command, enter the "rockdev" directory as shown below:

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk$ cd rockdev/
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/rockdev$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/rockdev$ ls
boot.img  MiniLoaderAll.bin  misc.img  oem.img  parameter.txt  recovery.img  rootfs.ext4  rootfs.img  uboot.img  userdata.img
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/rockdev$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/rockdev$ ls -l
总用量 12556
lrwxrwxrwx 1 tgg tgg       18 4月   7 09:23 boot.img -> ../kernel/boot.img
lrwxrwxrwx 1 tgg tgg       41 4月   7 09:23 MiniLoaderAll.bin -> ../u-boot/rk356x_spl_loader_v1.13.112.bin
lrwxrwxrwx 1 tgg tgg       44 4月   7 09:23 misc.img -> ../device/rockchip/rockimg/wipe_all-misc.img
-rw-rw-r-- 1 tgg tgg 17457152 4月   7 09:23 oem.img
lrwxrwxrwx 1 tgg tgg       53 4月   7 09:23 parameter.txt -> ../device/rockchip/rk356x/parameter-buildroot-fit.txt
lrwxrwxrwx 1 tgg tgg       64 4月   7 09:23 recovery.img -> ../buildroot/output/rockchip_rk356x_recovery/images/recovery.img
lrwxrwxrwx 1 tgg tgg       54 4月   6 22:46 rootfs.ext4 -> ../buildroot/output/rockchip_rk3568/images/rootfs.ext2
lrwxrwxrwx 1 tgg tgg       54 4月   7 09:23 rootfs.img -> ../buildroot/output/rockchip_rk3568/images/rootfs.ext2
lrwxrwxrwx 1 tgg tgg       19 4月   7 09:23 uboot.img -> ../u-boot/uboot.img
-rw-rw-r-- 1 tgg tgg  4472832 4月   7 09:23 userdata.img
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/rockdev$
```

Figure 4.3-12 The files in the "rockdev" directory

The files in this directory are mostly soft links, which link to the actual image files.

Apart from using the command "**./build.sh all**", we can also directly execute the script "**./build.sh**" (without any parameters) to compile the entire SDK; running the command "**./build.sh**" will add the following operations on the basis of the command "**./build.sh all**":

- Execute "./mkfirmware.sh" to package all the images into the "rockdev" directory
- Package all the images in the "rockdev" directory into a "update.img" firmware
- Copy the images in the "rockdev" directory to the "IMAGE/***_RELEASE_TEST/IMAGES" directory (*** represents the compilation date)
- Save the patches of each module to the "IMAGE/***_RELEASE_TEST/PATCHES" directory

- Note: The commands "./build.sh" and "./build.sh all" are the same

4.4 Independently compiling

The previous section introduced to users how to automatically compile the entire SDK to obtain all the images in one step. This section introduces to users how to independently compile each part of the source code to obtain the corresponding images.

4.4.1 Compile U-Boot separately

To compile U-Boot separately using the build.sh script, execute the following command in the root directory of the SDK source code:

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

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk$ ./build.sh uboot
processing option: uboot
============Start building uboot============
TARGET_UBOOT_CONFIG=rk3568
========================================
## make rk3568_defconfig -j24
#
# configuration written to .config
#
scripts/kconfig/conf  --silentoldconfig Kconfig
  CHK     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     tpl/include/autoconf.mk
  GEN     spl/include/autoconf.mk
  CHK     include/config/uboot.release
  CHK     include/generated/timestamp_autogenerated.h
  UPD     include/generated/timestamp_autogenerated.h
  CHK     include/config.h
  CFG     u-boot.cfg
  CHK     include/generated/version_autogenerated.h
  CHK     include/generated/generic-asm-offsets.h
  CHK     include/generated/asm-offsets.h
  HOSTCC  tools/mkenvimage.o
  HOSTCC  tools/fit_image.o
  HOSTCC  tools/image-host.o
  HOSTCC  tools/dumpimage.o
  HOSTCC  tools/mkimage.o
  HOSTCC  tools/rockchip/boot_merger.o
  HOSTCC  tools/rockchip/loaderimage.o
  HOSTLD  tools/mkenvimage
  HOSTLD  tools/loaderimage
  HOSTLD  tools/dumpimage
  HOSTLD  tools/mkimage
```

Figure 4.4.1.1 Compile U-Boot separately (1)

```
 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_linux_sdk/rkbin/RKBOOT/RK3568MINIALL.ini
/home/tgg/rk3568_linux_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_linux_sdk/rkbin/RKTRUST/RK3568TRUST.ini

Platform RK3568 is build OK, with new .config(make rk3568_defconfig -j24)
/home/tgg/rk3568_linux_sdk/prebuilts/gcc/linux-x86/aarch64/gcc-linaro-6.3.1-2017.05-x86_64_aarch64-linux-gn
Fri Apr  7 10:02:27 CST 2023
Running build_uboot succeeded.
tgg@tgg-virtual-machine:~/rk3568_linux_sdk$
```

Figure 4.4.1.2 Compile U-Boot separately (2)

After the compilation is successful, two images will be generated as follows:

<SDK>/uboot/uboot.img

<SDK>/uboot/rk356x_spl_loader_v1.13.112.bin

rk356x_spl_loader_v1.13.112.bin is actually MiniLoaderAll.bin, just renamed.

4.4.2 Compile Kernel Independently

By using the build.sh script, compile the Linux Kernel independently. Execute the following command in the root directory of the SDK source code:

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

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk$ ./build.sh kernel
processing option: kernel
============Start building kernel============
TARGET_ARCH          =arm64
TARGET_KERNEL_CONFIG =rockchip_linux_defconfig
TARGET_KERNEL_DTS    =rk3568-atk-evb1-ddr4-v10-linux
TARGET_KERNEL_CONFIG_FRAGMENT =
=========================================
  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
#
  WRAP    arch/arm64/include/generated/uapi/asm/errno.h
  WRAP    arch/arm64/include/generated/uapi/asm/ioctl.h
  WRAP    arch/arm64/include/generated/uapi/asm/ioctls.h
  WRAP    arch/arm64/include/generated/uapi/asm/ipcbuf.h
  WRAP    arch/arm64/include/generated/uapi/asm/kvm_para.h
  WRAP    arch/arm64/include/generated/uapi/asm/mman.h
  WRAP    arch/arm64/include/generated/uapi/asm/msgbuf.h
  WRAP    arch/arm64/include/generated/uapi/asm/poll.h
  WRAP    arch/arm64/include/generated/uapi/asm/resource.h
  WRAP    arch/arm64/include/generated/uapi/asm/sembuf.h
  WRAP    arch/arm64/include/generated/uapi/asm/shmbuf.h
  WRAP    arch/arm64/include/generated/uapi/asm/socket.h
  WRAP    arch/arm64/include/generated/uapi/asm/sockios.h
  WRAP    arch/arm64/include/generated/uapi/asm/swab.h
  WRAP    arch/arm64/include/generated/uapi/asm/termbits.h
```

Figure 4.4.2.1 Compile Kernel separately (1)

```
  Compression:  uncompressed
  Data Size:    458240 Bytes = 447.50 KiB = 0.44 MiB
  Hash algo:    sha256
  Hash value:   57ea171509199dbfaa27deb1e506cb02fe924d6343ac225e85b32c1460839119
 Default Configuration: 'conf'
 Configuration 0 (conf)
  Description:  unavailable
  Kernel:       kernel
  FDT:          fdt
grep: exceeded PCRE's backtracking limit
grep: exceeded PCRE's backtracking limit
grep: exceeded PCRE's backtracking limit
grep: exceeded PCRE's backtracking limit
grep: exceeded PCRE's backtracking limit
grep: exceeded PCRE's backtracking limit
grep: exceeded PCRE's backtracking limit
 PLEASE CHECK BOARD GPIO POWER DOMAIN CONFIGURATION !!!!!
 <<< ESPECIALLY Wi-Fi/Flash/Ethernet IO power domain >>> !!!!!
 Check Node [pmu_io_domains] in the file: /home/tgg/rk3568_linux_sdk/kernel/arch/arm64/boot/dts/rockchip/rk3

 请再次确认板级的电源域配置！！！！！！
 <<< 特别是Wi-Fi, FLASH, 以太网这几路IO电源的配置 >>> ！！！！！
 检查内核文件 /home/tgg/rk3568_linux_sdk/kernel/arch/arm64/boot/dts/rockchip/rk3568-atk-evb1-ddr4-v10-linux.d
Running build_kernel succeeded.
tgg@tgg-virtual-machine:~/rk3568_linux_sdk$
```

Figure 4.4.2.2 Build Kernel Separately (2)

After successful compilation, a **boot.img** image will be generated, and its path is: **<SDK>/kernel/boot.img**.

Executing the command "**./build.sh kernel**" will compile the Linux kernel source code, including the kernel device tree and kernel modules. If we need to compile the kernel modules separately, we can execute the following command for compilation:

```
./build.sh modules
```

Of course, before compiling the kernel module, the kernel source code needs to be compiled first.

4.4.3 Independently compiling rootfs

The rootfs is the root file system. The RK3568 Linux SDK supports various root file systems, including Buildroot, Yocto, and Debian. This section mainly introduces how to compile the Buildroot root file system. It is also recommended that users use Buildroot, and Yocto is not recommended. **The RK official itself does not recommend that users use Yocto either.**

By executing the build.sh script independently to compile the Buildroot root file system, execute the following commands in the root directory of the SDK source code:

```
./build.sh buildroot
```

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk$ ./build.sh buildroot
processing option: buildroot
==========Start building buildroot==========
TARGET_BUILDROOT_CONFIG=rockchip_rk3568
=========================================
Top of tree: /home/tgg/rk3568_linux_sdk
=========================================

#TARGET_BOARD=rk3568
#OUTPUT_DIR=output/rockchip_rk3568
#CONFIG=rockchip_rk3568_defconfig

=========================================
make: Entering directory '/home/tgg/rk3568_linux_sdk/buildroot'
  GEN     /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/Makefile
/home/tgg/rk3568_linux_sdk/buildroot/build/defconfig_hook.py -m /home/tgg/rk3568_linux_sdk/buildroot/conf
pconfig
BR2_DEFCONFIG='' KCONFIG_AUTOCONFIG=/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/build/bui
3568/build/buildroot-config/autoconf.h KCONFIG_TRISTATE=/home/tgg/rk3568_linux_sdk/buildroot/output/rockc
output/rockchip_rk3568/.config HOST_GCC_VERSION="9" BUILD_DIR=/home/tgg/rk3568_linux_sdk/buildroot/output,
kchip_rk3568_defconfig /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/build/buildroot-config,
ig.in
/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/.rockchipconfig:256:warning: override: reassig
/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/.rockchipconfig:257:warning: override: reassig
/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/.rockchipconfig:259:warning: override: reassig
/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/.rockchipconfig:284:warning: override: reassig
/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/.rockchipconfig:285:warning: override: reassig
/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/.rockchipconfig:286:warning: override: reassig
/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/.rockchipconfig:292:warning: override: reassig
/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/.rockchipconfig:293:warning: override: reassig
/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/.rockchipconfig:294:warning: override: reassig
/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/.rockchipconfig:297:warning: override: reassig
/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/.rockchipconfig:316:warning: override: reassig
/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/.rockchipconfig:319:warning: override: reassig
#
# configuration written to /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/.config
#
```

Figure 4.4.3.1 Buildroot compilation in isolation (1)

```
make: Leaving directory '/home/tgg/rk3568_linux_sdk/buildroot'
2023-04-07T10:34:27 >>>   Finalizing target directory
2023-04-07T10:34:28 >>>   Sanitizing RPATH in target tree
2023-04-07T10:34:36 >>>   Copying overlay board/rockchip/common/base
2023-04-07T10:34:36 >>>   Copying overlay board/rockchip/common/powermanager
2023-04-07T10:34:36 >>>   Copying overlay board/rockchip/rk356x/fs-overlay/
2023-04-07T10:34:36 >>>   Copying overlay board/rockchip/common/wifi
2023-04-07T10:34:36 >>>   Executing post-build script ../device/rockchip/common/post-build.sh
2023-04-07T10:34:36 >>>   Generating root filesystem image rootfs.cpio
2023-04-07T10:35:38 >>>   Generating root filesystem image rootfs.ext2
2023-04-07T10:35:44 >>>   Generating root filesystem image rootfs.squashfs
2023-04-07T10:35:50 >>>   Generating root filesystem image rootfs.tar
Done in 1min 39s
log saved on /home/tgg/rk3568_linux_sdk/br.log. pack buildroot image at: /home/tgg/rk3568_linux_sdk/buildroo
you take 1:40.36 to build builroot
Running build_buildroot succeeded.
Running build_rootfs succeeded.
```

Figure 4.4.3.2 Buildroot compilation in isolation (2)

After successful compilation, a buildroot root file system image will be generated. The image will be output in the directory "buildroot/output/rockchip_rk3568/images/" as follows:

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk$ cd buildroot/output/rockchip_rk3568/images/
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/images$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/images$ ls
rootfs.cpio  rootfs.cpio.gz  rootfs.ext2  rootfs.ext4  rootfs.squashfs  rootfs.tar
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/images$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/images$ ls -l
总用量 1765640
-rw-r--r-- 1 tgg tgg 426384384 4月   7 13:40 rootfs.cpio
-rw-r--r-- 1 tgg tgg 217841529 4月   7 13:41 rootfs.cpio.gz
-rw-r--r-- 1 tgg tgg 667996160 4月   7 13:41 rootfs.ext2
lrwxrwxrwx 1 tgg tgg        11 4月   7 13:41 rootfs.ext4 -> rootfs.ext2
-rw-r--r-- 1 tgg tgg 218415104 4月   7 13:41 rootfs.squashfs
-rw-r--r-- 1 tgg tgg 431226880 4月   7 13:41 rootfs.tar
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/images$
```

Figure 4.4.3.3 The buildroot root file system image

Multiple different formats of rootfs image files were compiled and generated. For the RK3568 platform, the ext4 format image file **rootfs.ext2** is used, and it is usually renamed as rootfs.img.

In addition, the buildroot root file system can also be compiled by using the command "**./build.sh rootfs**". This command is used for compiling the root file system, but it is not limited to buildroot; it can also compile Yocto and Debian. By default, buildroot is compiled. The rootfs to be compiled (Yocto/Debian/buildroot) can be specified through the environment variable **RK_ROOTFS_SYSTEM**.

```
# Specify the compilation of the buildroot root file system
export RK_ROOTFS_SYSTEM=buildroot
./build.sh rootfs
```

It should be noted that: Before compiling the root file system, the Linux kernel needs to be compiled in advance; because during the process of compiling the root file system, some driver modules that are not integrated in the kernel source code will also be compiled (such as the Bluetooth driver module hci_uart.ko), and the .ko driver modules compiled in the kernel source code directory will also be copied to the root file system (such as the WiFi driver module 8852bs.ko). Therefore, the kernel must be compiled first.

4.4.4 Build recovery separately

To build the recovery separately using the build.sh script, execute the following command in the root directory of the SDK source code:

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

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk$ ./build.sh recovery
processing option: recovery
==========Start building recovery==========
TARGET_RECOVERY_CONFIG=rockchip_rk356x_recovery
========================================
config is rockchip_rk356x_recovery
found kernel image
Top of tree: /home/tgg/rk3568_linux_sdk
============================================

#TARGET_BOARD=rk356x
#OUTPUT_DIR=output/rockchip_rk356x_recovery
#CONFIG=rockchip_rk356x_recovery_defconfig

============================================
make: Entering directory '/home/tgg/rk3568_linux_sdk/buildroot'
  GEN     /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk356x_recovery/Makefile
/home/tgg/rk3568_linux_sdk/buildroot/build/defconfig_hook.py -m /home/tgg/rk3568_linux_sdk/buildroot/confi
_recovery/.rockchipconfig
BR2_DEFCONFIG='' KCONFIG_AUTOCONFIG=/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk356x_recovery/b
ckchip_rk356x_recovery/build/buildroot-config/autoconf.h KCONFIG_TRISTATE=/home/tgg/rk3568_linux_sdk/build
rk3568_linux_sdk/buildroot/output/rockchip_rk356x_recovery/.config HOST_GCC_VERSION="9" BUILD_DIR=/home/tg
me/tgg/rk3568_linux_sdk/buildroot/configs/rockchip_rk356x_recovery_defconfig /home/tgg/rk3568_linux_sdk/bu
linux_sdk/buildroot/output/rockchip_rk356x_recovery/.rockchipconfig Config.in
/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk356x_recovery/.rockchipconfig:62:warning: override:
/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk356x_recovery/.rockchipconfig:63:warning: override:
#
# configuration written to /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk356x_recovery/.config
#
make: Leaving directory '/home/tgg/rk3568_linux_sdk/buildroot'
```

Figure 4.4.4.1 Compile recovery(1) separately

```
  Created:      Fri Apr  7 15:32:45 2023
  Type:         RAMDisk Image
  Compression:  uncompressed
  Data Size:    7805954 Bytes = 7623.00 KiB = 7.44 MiB
  Architecture: AArch64
  OS:           Linux
  Load Address: 0xffffff02
  Entry Point:  unavailable
  Hash algo:    sha256
  Hash value:   6f5fecf54934f16141bbf9769a58cc8405fbff5f8b9275830681a616ba7d64f7
 Image 3 (resource)
  Description:  unavailable
  Created:      Fri Apr  7 15:32:45 2023
  Type:         Multi-File Image
  Compression:  uncompressed
  Data Size:    458240 Bytes = 447.50 KiB = 0.44 MiB
  Hash algo:    sha256
  Hash value:   57ea171509199dbfaa27deb1e506cb02fe924d6343ac225e85b32c1460839119
 Default Configuration: 'conf'
 Configuration 0 (conf)
  Description:  unavailable
  Kernel:       kernel
  Init Ramdisk: ramdisk
  FDT:          fdt
done.
you take 0:10.17 to build recovery
Running build_recovery succeeded.
tgg@tgg-virtual-machine:~/rk3568_linux_sdk$
```

Figure 4.4.4.2 Build recovery (2)

The "recovery.img" is used to enter the recovery mode. This image will be burned onto the recovery partition of the development board.

The "recovery.img" is formed by merging multiple images, which include the ramdisk (the root file system mounted in the recovery mode), the kernel image, the kernel DTB, and the resource image "resource.img". Therefore, before compiling the recovery, the Linux kernel must also be compiled in advance.

After successful compilation, the "recovery.img" will be generated. This image will be output in the "buildroot/output/rockchip_rk356x_recovery/images/" directory, as shown below:

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk$ cd buildroot/output/rockchip_rk356x_recovery/images/
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot/output/rockchip_rk356x_recovery/images$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot/output/rockchip_rk356x_recovery/images$ ls
recovery.img  rootfs.cpio  rootfs.cpio.gz  rootfs.ext2  rootfs.ext4  rootfs.squashfs  rootfs.tar
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot/output/rockchip_rk356x_recovery/images$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot/output/rockchip_rk356x_recovery/images$ ls -l
总用量 168364
-rw-rw-r-- 1 tgg tgg 31011840 4月   7 15:32 recovery.img
-rw-r--r-- 1 tgg tgg 18861568 4月   7 15:32 rootfs.cpio
-rw-r--r-- 1 tgg tgg  7805954 4月   7 15:32 rootfs.cpio.gz
-rw-r--r-- 1 tgg tgg 89628672 4月   7 15:32 rootfs.ext2
lrwxrwxrwx 1 tgg tgg       11 4月   7 15:32 rootfs.ext4 -> rootfs.ext2
-rw-r--r-- 1 tgg tgg  7835648 4月   7 15:32 rootfs.squashfs
-rw-r--r-- 1 tgg tgg 19220480 4月   7 15:32 rootfs.tar
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot/output/rockchip_rk356x_recovery/images$
```

Figure 4.4.4.3 Recovery image file

4.4.5 Package into update.img image

update.img is a collection of multiple images (formed by packaging and merging multiple images). Using the tools provided by RK, each separate image (such as uboot.img, boot.img, MiniLoaderAll.bin, parameter.txt, misc.img, rootfs.img, oem.img, userdata.img, recovery.img, etc.) can be packaged into a single update.img firmware, making it convenient for users to burn and upgrade.

We can package each separate image in the rockdev directory into an update.img firmware using the following command. Using the update.img firmware is more convenient for burning and updating!

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

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk$ ./build.sh updateimg
processing option: updateimg
Make update.img
start to make update.img...
Android Firmware Package Tool v2.0
------ PACKAGE ------
Add file: ./package-file
package-file,Add file: ./package-file done,offset=0x800,size=0x28b,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=0x1f4,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=0x1621600,userspace=0x2c43
Add file: ./Image/recovery.img
recovery,Add file: ./Image/recovery.img done,offset=0x1aa1000,size=0x1d93400,userspace=0x3b27
Add file: ./Image/rootfs.img
rootfs,Add file: ./Image/rootfs.img done,offset=0x3834800,size=0x27d11000,userspace=0x4fa22
Add file: ./Image/oem.img
oem,Add file: ./Image/oem.img done,offset=0x2b545800,size=0x10a6000,userspace=0x214c
Add file: ./Image/userdata.img
userdata,Add file: ./Image/userdata.img done,offset=0x2c5eb800,size=0x444000,userspace=0x888
Add CRC...
Make firmware OK!
------ OK ------
********rkImageMaker ver 2.0********
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 ./Image/update.img OK.
Running build_updateimg succeeded.
tgg@tgg-virtual-machine:~/rk3568_linux_sdk$
```

Figure 4.4.5.1 Package into update.img

After the package is successful, a firmware named update.img will be generated in the rockdev directory, as shown below:

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk$ cd rockdev/
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/rockdev$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/rockdev$ ls
boot.img  MiniLoaderAll.bin  misc.img  oem.img  parameter.txt  recovery.img  rootfs.ext4  rootfs.img  uboot.img  update.img  u
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/rockdev$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/rockdev$ ls -l update.img
-rw-rw-r-- 1 tgg tgg 749343306 4月   7 16:20 update.img
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/rockdev$
```

Figure 4.4.5.2 update.img firmware

4.5 SDK cleanup

The cleanup operation can be performed in the root directory of the SDK source code by running the build.sh script. Execute the following command:

```
./build.sh cleanall
```

The cleanup operation can be performed in the root directory of the SDK source code by running the build.sh script. Execute the following command:

4.6 Mirror Introduction

The first two sections introduced to users how to compile the SDK. After compilation, multiple mirror image files will be generated, as shown in the following table:

Mirror name	Function
uboot.img	uboot.img is a FIT format image, which is formed by merging multiple images. These include the trust image (ARM Trusted Firmware + OP-TEE OS), the u-boot image, and the u-boot dtb. When compiling U-Boot, these images are packaged into a single uboot.img.

	The uboot.img will be burned into the uboot partition of the development board.
boot.img	boot.img is also a FIT format image, and it is composed of multiple images combined together, including the kernel image, the kernel DTB, and the resource image resource.img. boot.img will be burned onto the boot partition of the development board.
MiniLoaderAll.bin	This image is a segment of Loader code that ran before U-Boot on the RK3568 platform (that is, it is a Loader at a stage earlier than U-Boot). MiniLoaderAll.bin consists of two parts: TPL and SPL. TPL is used to initialize DDR and runs in SRAM; while SPL runs in DDR and mainly responsible for loading and booting uboot.img.
misc.img	The image contains BCB (Bootloader Control Block) information and will be written to the misc partition of the development board. The misc partition is a very important partition, where the BCB data block is stored and is mainly used for communication between the Android/Linux system, U-Boot, and recovery.
oem.img	For use by the manufacturer, it is used to store the manufacturer's APP or data. This image will be burned onto the OEM partition of the development board. After the system starts up, it will be mounted to the /oem directory.
parameter.txt	A txt text file is a partition table file for the RK3568 platform (which records the partition names and the starting and ending addresses of each partition); when burning the image, there is no need to burn the parameter.txt file to the Flash; instead, its information will be read to define the partitions.
recovery.img	The recovery mode image, recovery.img, is used to enter the recovery mode. This recovery.img is burned onto the recovery partition of the development board. The recovery mode is a mode used for repairing and upgrading the device. recovery.img is also a FIT format image, which is formed by merging multiple images, including the ramdisk (mounted as the root file system when entering the recovery mode), the kernel image (booted when entering the recovery mode), the kernel DTB, and resource.img.
rootfs.img	The corresponding root file system image in the normal boot mode contains a large number of library files, executable files, etc. The rootfs.img will be burned onto the rootfs partition of the development board.
userdata.img	For user use, it can be used to store the user's apps or data; this image will be written to the userdata partition of the development board. After the system starts, it will be mounted to the /userdata directory.

Table 4.6.1 Introduction to Each Image of RK3568

This is a brief introduction to these images.

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, rootfs.img, recovery.img, etc.) on the ATK-DLRK3568 development board.

The Rockchip platform offers various methods for burning images, such as burning them using the RISC-V development tools in Windows, through the SD card method, through the FactoryTool tool for batch burning (mass production burning tool, supporting USB one-to-many burning), and through the Linux_Upgrade_Tool tool in Ubuntu, etc. In summary, there are many ways to burn images, and users can choose the appropriate method to burn the image files onto the development board.

5.1 Flash Writing Mode Introduction

Section 2.9.1 has already provided a detailed explanation of the flash writing mode related issues of the development board. If you still have any doubts about this, please proceed to read this section!

5.2 Flash Writing under Windows System

Under Windows, the image can be flashed using the RKDevTool provided by Rockchip. Before flashing, first copy the image file obtained by compiling the SDK in Chapter 4 (<SDK>/rockdev directory) from the Ubuntu system to the Windows system. For example, copy these images to the rk3568_images directory on the Windows desktop, including boot.img, MiniLoaderAll.bin, misc.img, OEM.img, parameter.txt (partition table file, not the image), recovery.img, rootfs.img, uboot.img, and userdata.img. As shown below:

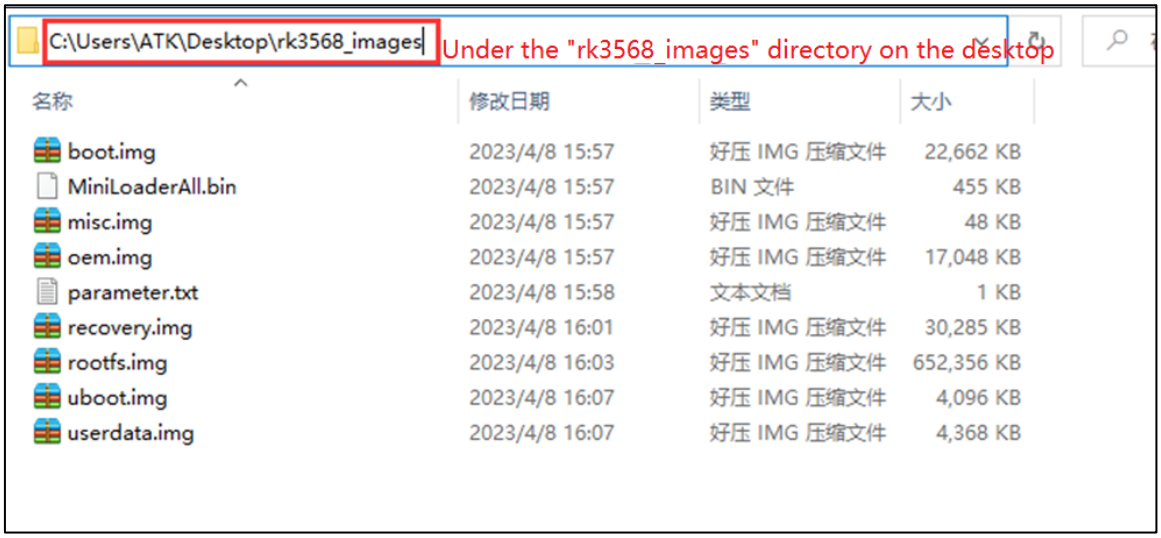


Figure 5.2-1 The files in the "rk3568_images" directory of the desktop

Open the Ralink Microelectronics development tools:

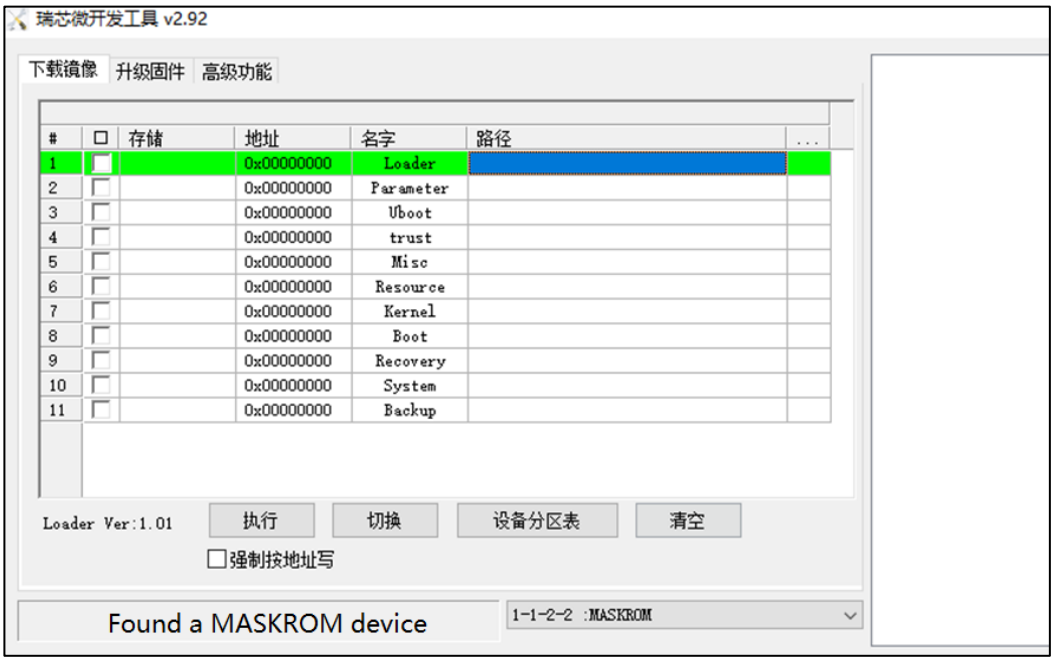


Figure 5.2-2 Rockchip Development Tools

5.2.1 Introduction to Partition Table parameter.txt

This section will explain the partition table file parameter.txt for the RK platform. This file is a txt text file. The parameter.txt file describes the partition table information of the development board, including the names of each partition, the starting addresses of the partitions, and the sizes of the partitions, etc. Let's take a look at its content:

```
FIRMWARE_VER: 1.0
MACHINE_MODEL: RK3568
MACHINE_ID: 007
MANUFACTURER: RK3568
MAGIC: 0x5041524B
ATAG: 0x00200800
MACHINE: 0xffffffff
CHECK_MASK: 0x80
PWR_HLD: 0,0,A,0,1
TYPE: GPT
CMDLINE:
mtdparts=rk29xxnand:0x00002000@0x00004000(uboot),0x00002000@0x00006000(misc),0x00020000@0x00008000(boot),0x00020000@0x00028000(recovery),0x00010000@0x00048000(backup),0x00c00000@0x00058000(rootfs),0x00040000@0x00c58000(oem),-@0x00c98000(userdata:grow)
uuid:rootfs=614e0000-0000-4b53-8000-1d28000054a9
```

The parameter.txt file not only contains partition table information, but also other identifiers such as FIRMWARE_VER, MACHINE_MODEL, MACHINE_ID, MAGIC, etc. This article does not provide an introduction to these, for details, please refer to the official documentation of RK: **<SDK>/docs/Common/TOOL/Rockchip_Introduction_Partition_CN.pdf**.

Here, we will only introduce the partition table information defined by the **mtdparts** identifier.

The information defined by mtdparts is as follows:

```
rk29xxnand:0x00002000@0x00004000(uboot),0x00002000@0x00006000(misc),0x00020000@0x00008000(boot),0x00020000@0x00028000(recovery),0x00010000@0x00048000(backup),0x00c00000@0x00058000(rootfs),0x00040000@0x00c58000(oem),-@0x00c98000(userdata:grow)
```

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

Information such as 0x00002000@0x00004000 (uboot), 0x00002000@0x00006000 (misc) is 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 (512 bytes per sector). Therefore, it can be known that the starting position of the uboot partition is at the 0x4000 sector position, with a size of 0x2000 sectors (4MB); the starting position of the misc partition is at the 0x6000 sector 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:

```
-@0x00c98000(userdata:grow)
```

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

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

Partition name	Explanation
uboot	The partition is used to store uboot.img. The uboot.img image will be burned into this partition.
misc	The misc partition is a very important partition, which includes BCB data blocks and is mainly used for communication between Android/Linux systems, U-Boot, and recovery. The misc.img image will be burned into this partition.
boot	It is used to store boot.img. The boot.img image will be burned into this partition.
recovery	For the recovery mode, the recovery.img will be burned into this partition. The system will boot the recovery.img to enter the recovery mode. The recovery mode is a mode used for repairing and upgrading the device.
backup	The reserved partition is not currently in use.
rootfs	The root file system partition is used to store rootfs.img. The rootfs.img image of the normal boot mode will be burned into this partition.
oem	This is a partition for the manufacturer's use, which stores the manufacturer's apps or data. The oem.img image will be burned into this partition; after the system starts, this partition will be mounted to the root file system/oem directory.
userdata	This is the partition for end users, which stores the users' apps or data. After the system starts, this partition will be mounted to the root file system/userdata directory.

Table 5.2.1.1 Introduction of Each Partition's Function

5.2.2 Configuration

From the analysis in the previous section, it can be seen that the parameter.txt file defines a total of 8 partitions, providing the names, starting addresses, and partition sizes 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:

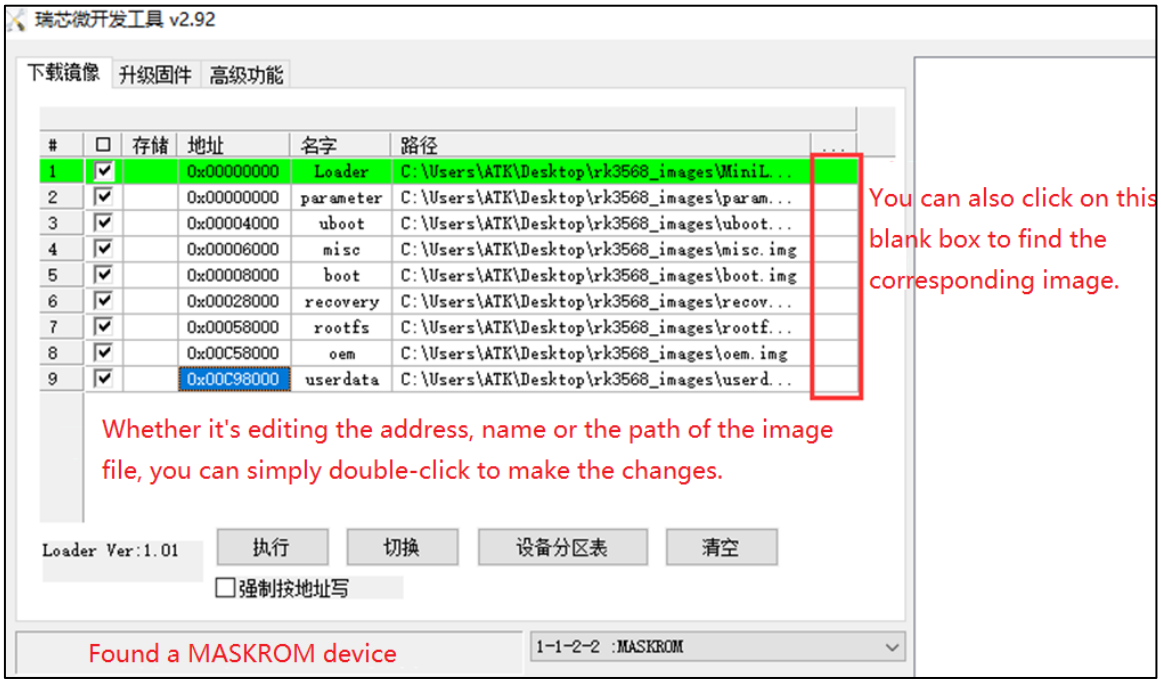


Figure 5.2.2.1 Manually configure the Rockchip development tools

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 burning address does not need to be configured; simply use 0x0, because the MiniLoaderAll.bin image has a dedicated burning address and does not require user configuration. Also, its name is usually Loader (or in lowercase, loader), do not modify it;

3. In the above figure, the second item corresponds to the partition table parameter.txt. 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 will read the partitions defined in this file and initialize the physical partitions of the Flash; likewise, its name is parameter (or in uppercase, Parameter), do not modify it, because the underlying system needs to identify the partition table file through this "**parameter**" name;

4. Except for **MiniLoaderAll.bin** and **parameter.txt** which are a little special, the other images can be configured directly according to the starting address defined in the parameter.txt partition table. Try to use the partition names defined in the parameter.txt file as much as possible.

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 very simple, the steps are as shown in Figures 5.2.2.2 to 5.2.2.3:

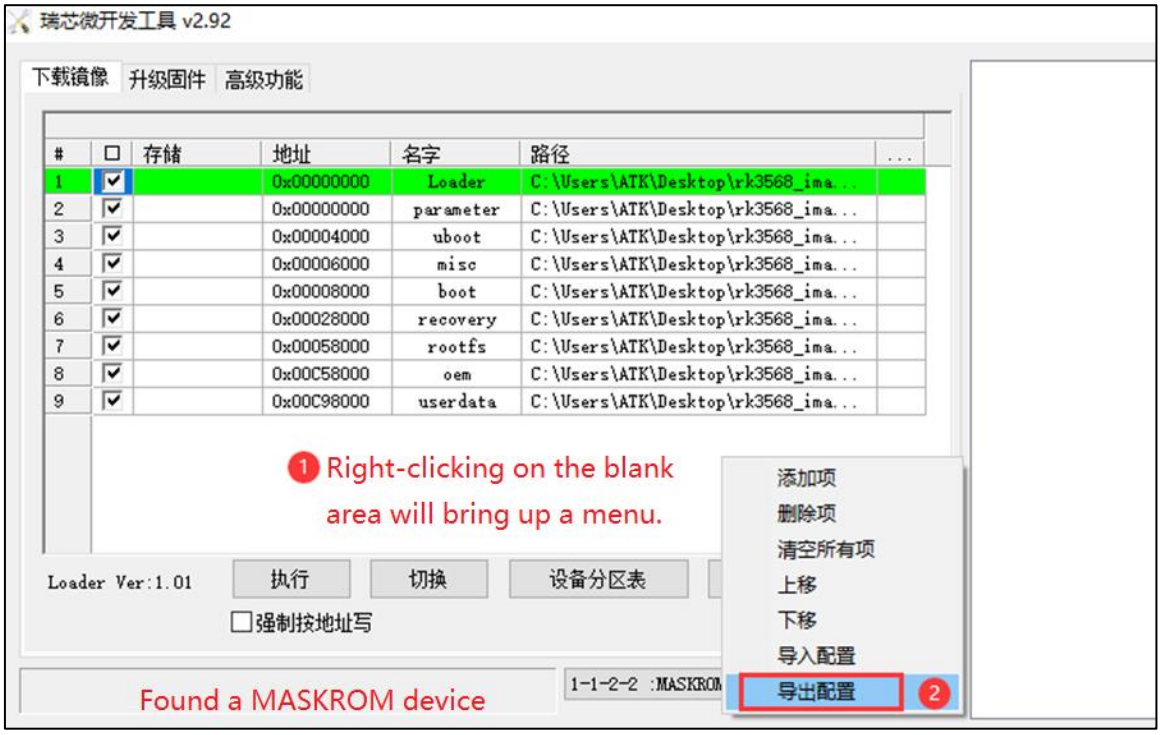


Figure 5.2.2.2 Export configuration (1)

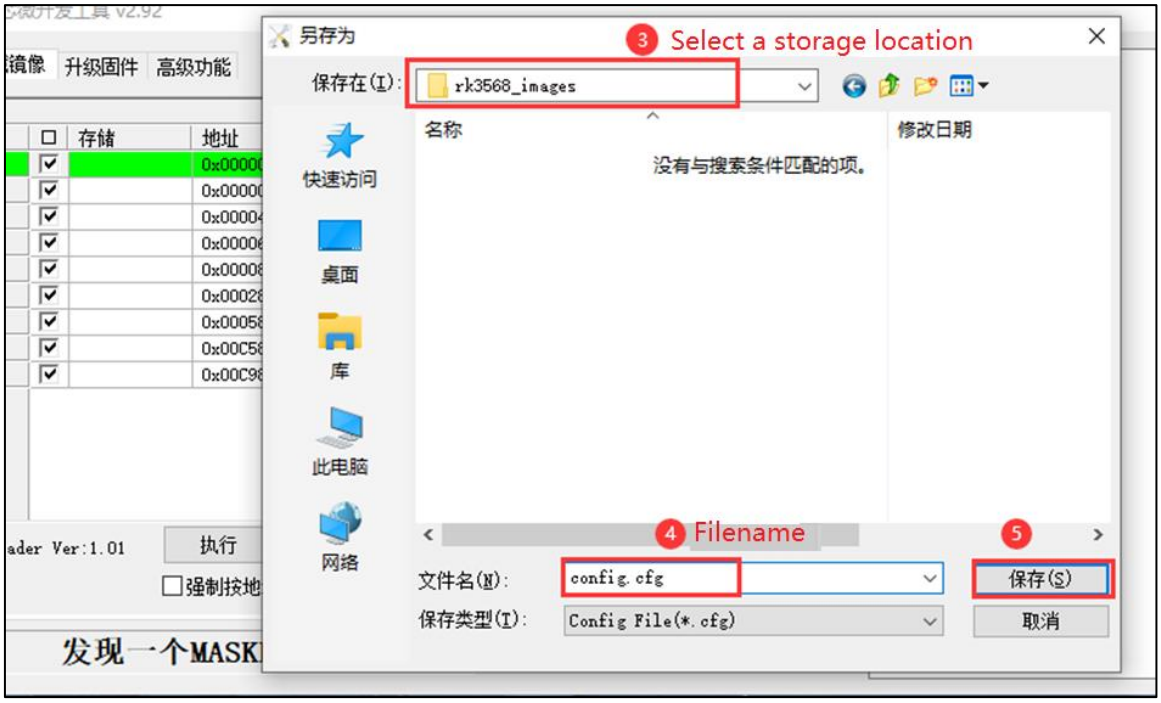


Figure 5.2.2.3 Export Configuration (2)

Export and save the configuration information to the config.cfg file for easy import of this configuration information in the future.

5.2.3 Flashing

First, connect the power adapter and OTG to the development board. Before flashing, make the development board enter the Maskrom or Loader mode.

For example, to flash the image in Maskrom mode, press the **UPDATE** button on the development board, then power on or reset the development board. At this time, 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 flash 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:

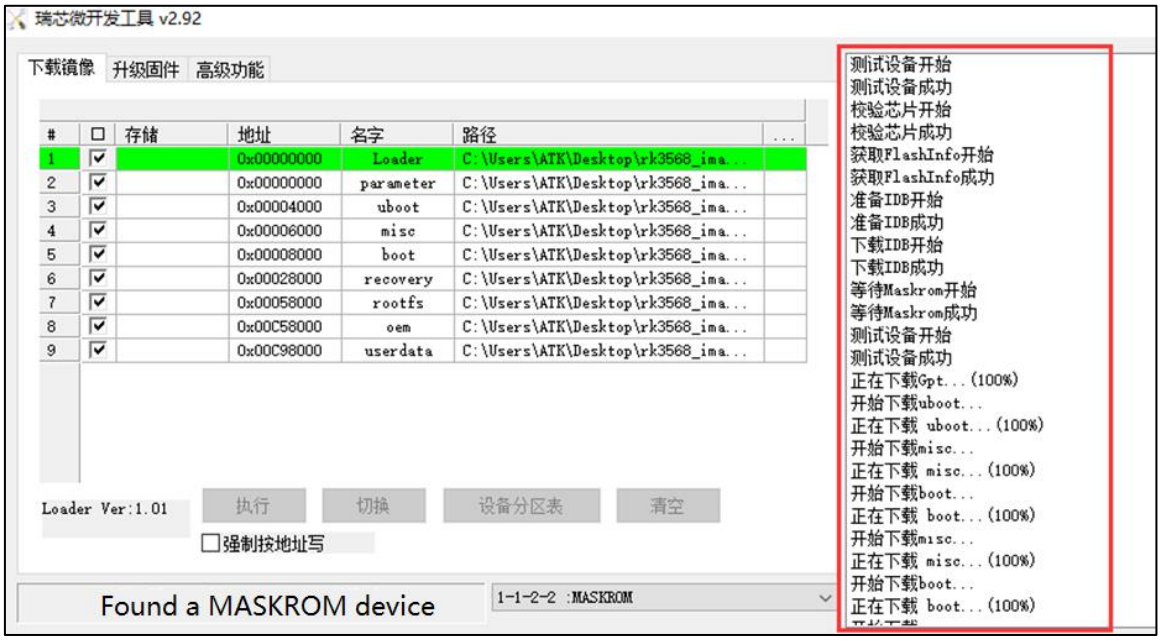
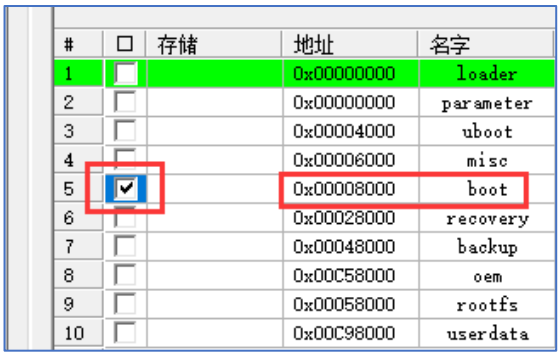


Figure 5.2.3.2 Log information of the burning process

It is also possible to burn a specific image alone, for example, to burn the "boot.img" to the "boot" partition alone. Just check the corresponding option as shown below:



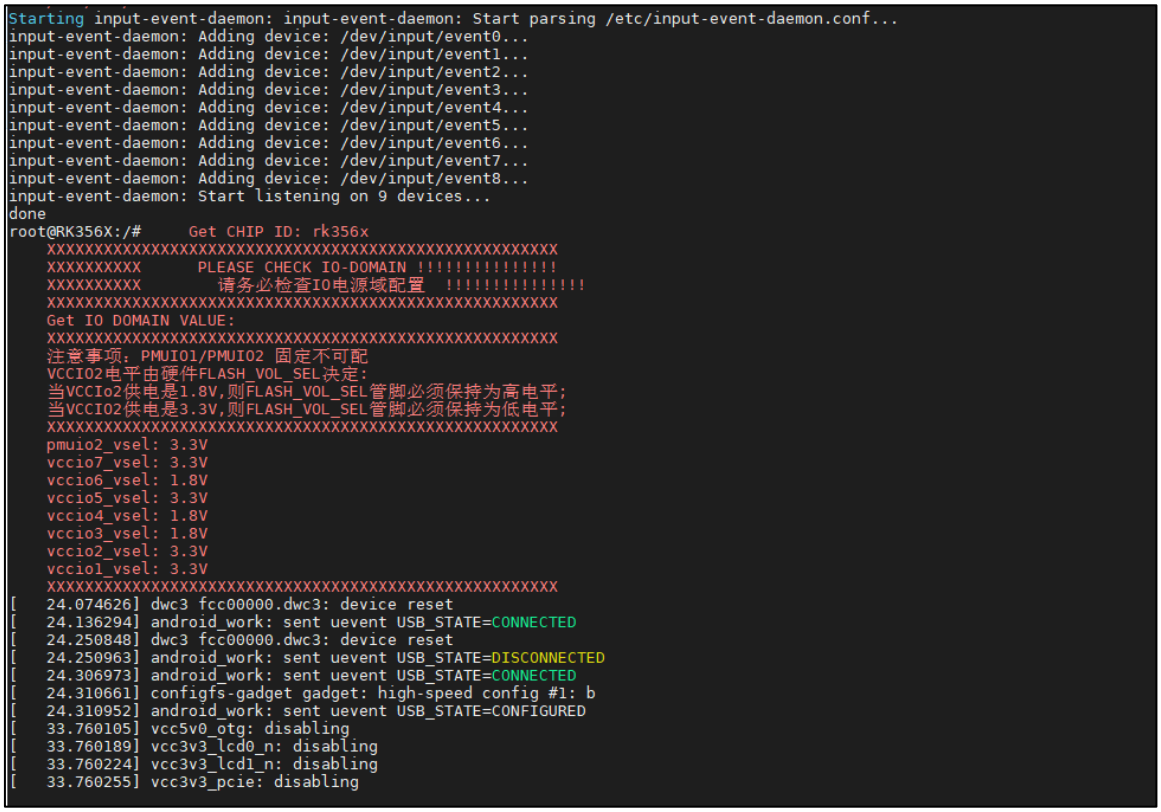
#	☐	存储	地址	名字
1	☐		0x00000000	loader
2	☐		0x00000000	parameter
3	☐		0x00004000	uboot
4	☐		0x00006000	misc
5	☑		0x00008000	boot
6	☐		0x00028000	recovery
7	☐		0x00048000	backup
8	☐		0x00C58000	oem
9	☐		0x00058000	rootfs
10	☐		0x00C98000	userdata

Figure 5.2.3.3 Burn the boot.img separately

Then click the "**Execute**" button to perform the burning. 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 development board will automatically restart and enter the Linux buildroot system. The serial port terminal will output information:



```
Starting input-event-daemon: input-event-daemon: Start parsing /etc/input-event-daemon.conf...
input-event-daemon: Adding device: /dev/input/event0...
input-event-daemon: Adding device: /dev/input/event1...
input-event-daemon: Adding device: /dev/input/event2...
input-event-daemon: Adding device: /dev/input/event3...
input-event-daemon: Adding device: /dev/input/event4...
input-event-daemon: Adding device: /dev/input/event5...
input-event-daemon: Adding device: /dev/input/event6...
input-event-daemon: Adding device: /dev/input/event7...
input-event-daemon: Adding device: /dev/input/event8...
input-event-daemon: Start listening on 9 devices...
done
root@RK356X:/#     Get CHIP ID: rk356x
    XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
    XXXXXXXXXX      PLEASE CHECK IO-DOMAIN !!!!!!!!!!!!!!!
    XXXXXXXXXX        请务必检查IO电源域配置  !!!!!!!!!!!!!!!
    XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
    Get IO DOMAIN VALUE:
    XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
    注意事项：PMUIO1/PMUIO2 固定不可配
    VCCIO2电平由硬件FLASH_VOL_SEL决定:
    当VCCIo2供电是1.8V,则FLASH_VOL_SEL管脚必须保持为高电平;
    当VCCIO2供电是3.3V,则FLASH_VOL_SEL管脚必须保持为低电平;
    XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
    pmuio2_vsel: 3.3V
    vccio7_vsel: 3.3V
    vccio6_vsel: 1.8V
    vccio5_vsel: 3.3V
    vccio4_vsel: 1.8V
    vccio3_vsel: 1.8V
    vccio2_vsel: 3.3V
    vccio1_vsel: 3.3V
    XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
[   24.074626] dwc3 fcc00000.dwc3: device reset
[   24.136294] android_work: sent uevent USB_STATE=CONNECTED
[   24.250848] dwc3 fcc00000.dwc3: device reset
[   24.250963] android_work: sent uevent USB_STATE=DISCONNECTED
[   24.306973] android_work: sent uevent USB_STATE=CONNECTED
[   24.310661] configfs-gadget gadget: high-speed config #1: b
[   24.310952] android_work: sent uevent USB_STATE=CONFIGURED
[   33.760105] vcc5v0_otg: disabling
[   33.760189] vcc3v3_lcd0_n: disabling
[   33.760224] vcc3v3_lcd1_n: disabling
[   33.760255] vcc3v3_pcie: disabling
```

Figure 5.2.4.1 Enter the system

5.2.5 Flashing update.img

You can use the Rockchip development tools to flash the update.img firmware. First, copy the update.img firmware generated in Section 4.4.5 to the Windows environment. Then, use the Rockchip

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 integrated in the SDK. The path is:

<SDK>/tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool, as shown below:

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk$ cd tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool$ ls
命令行开发工具使用文档.pdf  config.ini  Linux开发工具使用手册_v1.32.pdf  revision.txt  upgrade_tool
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool$
```

Figure 5.3-1 Linux_Upgrade_Tool Directory

This directory contains two usage instructions provided by RK: "**Command Line Development Tools Usage Document.pdf**" and "**Linux Development Tools User Manual_v1.32.pdf**". For detailed usage methods of the upgrade_tool tool, please refer to these two documents.

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 instructions and press enter to execute the instructions. Then, the corresponding operation triggered by the instruction will be executed, such as downloading the image, erasing the image, reading device information, etc. It is somewhat like a command line terminal.

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' command, there is no interaction with the user. The parameters passed in are used to tell the 'upgrade_tool' 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 development board to Ubuntu

Before using the 'upgrade_tool' tool to burn the image in Ubuntu, the development board needs to be connected to the Ubuntu system. First, connect the hardware (connect the power adapter and the OTG port), and put the development board in Maskrom or Loader mode; in the lower right corner of the Ubuntu system, there is a device named "**Fuzhou Rockchip Rockusb Device**" (it will only be displayed when the mouse is moved to the corresponding icon, as shown in the following figure):

Figure 5.3.1.1 Rockchip Rockusb Device

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

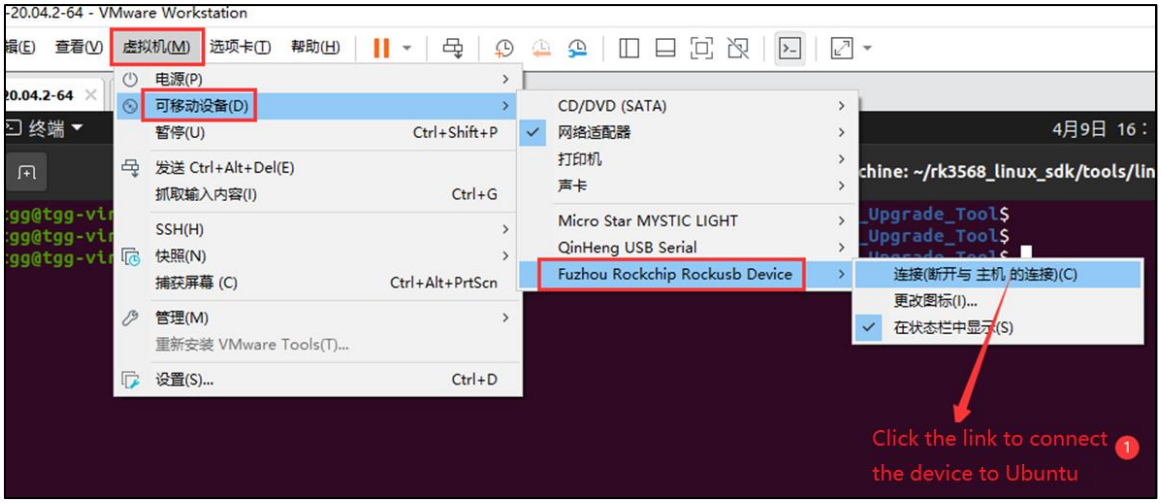


Figure 5.3.1.2 Connect the device to Ubuntu (1)

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

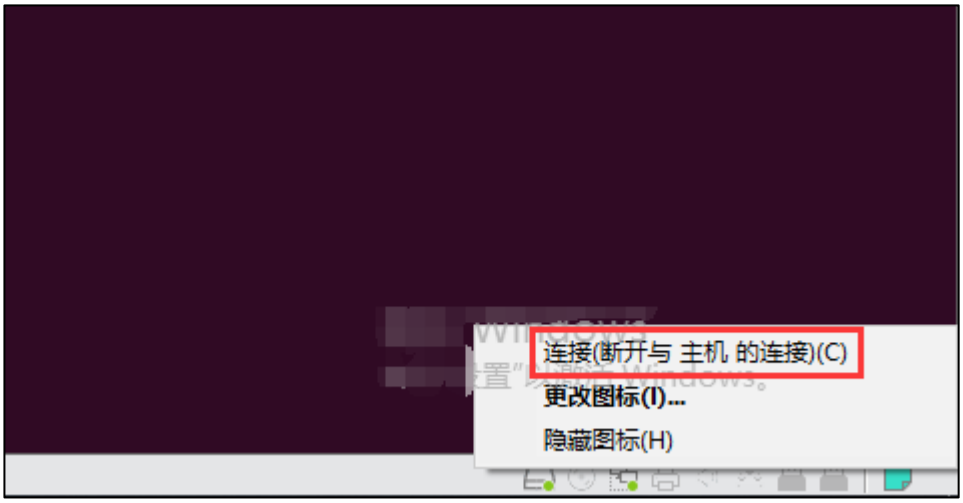


Figure 5.3.1.3 Connect the device to Ubuntu (2)

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

5.3.2 Using the upgrade_tool tool for burning

This section introduces to users how to use the upgrade_tool tool to burn the image (only the command-line mode burning is described). For ease of operation, we directly enter the <SDK>/rockdev directory. Before burning, the development board needs to be in the Maskrom or Loader mode.

The upgrade_tool tool supports many commands. Different commands can perform different operations, such as CD, LD, SD, UF, DI, DB, TD, RD, etc. (**case insensitive**), some commands require parameters while others do not. For details, please refer to <**Command Line Development Tools User Guide.pdf**> and <**Linux Development Tools User Manual_v1.32.pdf**>.

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

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

```
sudo ../tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/upgrade_tool UL MiniLoaderAll.bin -noreset
```

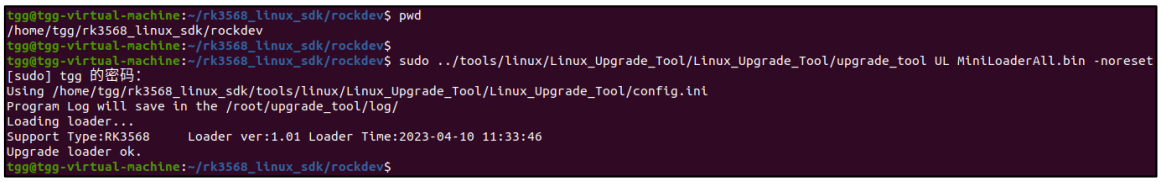


Figure 5.3.2.1 Flash "MiniLoaderAll.bin"

The "upgrade_tool" command is followed by three parameters. The first parameter indicates the instruction to be executed, the UL instruction is used to flash "MiniLoaderAll.bin"; the second parameter is used to specify the path where "MiniLoaderAll.bin" is located; the third parameter -noreset indicates that the development board should not be reset (do not reset the device) after flashing "MiniLoaderAll.bin".

After flashing "MiniLoaderAll.bin", the next step is to download the "parameter.txt" partition table through the DI instruction:

```
sudo ../tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/upgrade_tool DI -p parameter.txt
```

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/rockdev$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/rockdev$ sudo ../tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/upgrade_tool DI -p parameter.txt
Using /home/tgg/rk3568_linux_sdk/tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/config.ini
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_linux_sdk/rockdev$
```

Figure 5.3.2.2 Download the partition table

Next, burn the other images:

```
sudo ../tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/upgrade_tool DI -uboot uboot.img
sudo ../tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/upgrade_tool DI -misc misc.img
sudo ../tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/upgrade_tool DI -boot boot.img
sudo ../tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/upgrade_tool DI -recovery recovery.img
sudo ../tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/upgrade_tool DI -oem oem.img
sudo ../tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/upgrade_tool DI -rootfs rootfs.img
sudo ../tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/upgrade_tool DI -userdata userdata.img
```

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/rockdev$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/rockdev$ sudo ../tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/upgrade_tool DI -uboot uboot.img
Using /home/tgg/rk3568_linux_sdk/tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/config.ini
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_linux_sdk/rockdev$
```

Figure 5.3.2.3 Burn uboot.img

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/rockdev$ sudo ../tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/upgrade_tool DI -misc misc.img
Using /home/tgg/rk3568_linux_sdk/tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/config.ini
Program Log will save in the /root/upgrade_tool/log/
directlba=1,first4access=1,gpt=1
Download misc start...(0x00006000)
Download image ok.
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/rockdev$
```

Figure 5.3.2.4 Burn misc.img

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/rockdev$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/rockdev$ sudo ../tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/upgrade_tool DI -boot boot.img
Using /home/tgg/rk3568_linux_sdk/tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/config.ini
Program Log will save in the /root/upgrade_tool/log/
directlba=1,first4access=1,gpt=1
Download boot start...(0x00008000)
Download image ok.
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/rockdev$
```

Figure 5.3.2.5 Burn boot.img

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/rockdev$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/rockdev$ sudo ../tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/upgrade_tool DI -recovery recovery.img
Using /home/tgg/rk3568_linux_sdk/tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/config.ini
Program Log will save in the /root/upgrade_tool/log/
directlba=1,first4access=1,gpt=1
Download recovery start...(0x00028000)
Download image ok.
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/rockdev$
```

Figure 5.3.2.6 Burn recovery.img

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/rockdev$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/rockdev$ sudo ../tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/upgrade_tool DI -oem oem.img
Using /home/tgg/rk3568_linux_sdk/tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/config.ini
Program Log will save in the /root/upgrade_tool/log/
directlba=1,first4access=1,gpt=1
Download oem start...(0x00c58000)
Download image ok.
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/rockdev$
```

Figure 5.3.2.7 Burn oem.img

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/rockdev$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/rockdev$ sudo ../tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/upgrade_tool DI -rootfs rootfs.img
Using /home/tgg/rk3568_linux_sdk/tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/config.ini
Program Log will save in the /root/upgrade_tool/log/
directlba=1,first4access=1,gpt=1
Download rootfs start...(0x00058000)
Download image ok.
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/rockdev$
```

Figure 5.3.2.8 Burn rootfs.img

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/rockdev$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/rockdev$ sudo ../tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/upgrade_tool DI -userdata userdata.img
Using /home/tgg/rk3568_linux_sdk/tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/config.ini
Program Log will save in the /root/upgrade_tool/log/
directlba=1,first4access=1,gpt=1
Download userdata start...(0x00c98000)
Download image ok.
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/rockdev$
```

Figure 5.3.2.9 Burn userdata.img

The "DI -<partition_name>" part, where <partition_name> represents the partition name, such as -boot is the boot partition and -oem is the OEM partition; using the upgrade_tool tool to burn the image does not require the user to specify the burning address. The parameter.txt file has already defined the starting positions of each partition, which is also the burning address of the image.

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

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

```
sudo ../tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/upgrade_tool RD
```

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/rockdev$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/rockdev$ sudo ../tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/upgrade_tool RD
[sudo] tgg 的密码:
Using /home/tgg/rk3568_linux_sdk/tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/config.ini
Program Log will save in the /root/upgrade_tool/log/
Reset Device OK.
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/rockdev$
```

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 firmware.

The development board is in Maskrom or Loader mode. Execute the following command to flash the update.img firmware (using the UF command to flash update.img):

```
sudo ../tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/upgrade_tool UF update.img
```

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/rockdev$ sudo ../tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/upgrade_tool UF update.img
Using /home/tgg/rk3568_linux_sdk/tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/config.ini
Program Log will save in the /root/upgrade_tool/log/
Loading firmware...
Support Type:RK3568     FW Ver:1.0.00   FW Time:2023-04-07 16:20:07
Loader ver:1.01 Loader Time:2023-04-07 14:18:36
Upgrade firmware ok.
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/rockdev$
```

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 explains how to use the "upgrade_tool" tool to erase the Flash of the development board.

When the development board is in Maskrom or Loader mode, the following command can be executed to erase all the data in the Flash (using the EF instruction):

sudo ../tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/upgrade_tool EF MiniLoaderAll.bin

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/rockdev$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/rockdev$ sudo ../tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/upgrade_tool EF MiniLoaderAll.bin
Using /home/tgg/rk3568_linux_sdk/tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/config.ini
Program Log will save in the /root/upgrade_tool/log/
Loading loader...
Erase flash ok.
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/rockdev$
```

Figure 5.3.4.1 Erasure of all data

Of course, sector erasure by address can also be performed, where the user specifies the starting position and size of the erasure (using the EL instruction).

5.3.5 Writing using the rkflash.sh script

The <SDK>/rkflash.sh script provided by RK is a burning script that we can directly use to perform the burning; of course, this script also calls the upgrade_tool tool to execute the burning operation.

The usage is very simple. First, let the development board be in Maskrom or Loader mode, and directly run the rkflash.sh script to burn the image in the <SDK>/rockdev directory onto the development board (**you also need to add sudo, otherwise the operation will fail!**) ：

sudo ./rkflash.sh

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk$ sudo ./rkflash.sh
flash all images as default
Using /home/tgg/rk3568_linux_sdk/tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/config.ini
Program Log will save in the /root/upgrade_tool/log/
Loading loader...
Support Type:RK3568     Loader ver:1.01 Loader Time:2023-04-10 11:33:46
Upgrade loader ok.
Using /home/tgg/rk3568_linux_sdk/tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/config.ini
Program Log will save in the /root/upgrade_tool/log/
directlba=1,first4access=1,gpt=1
Write gpt...
Write gpt ok.
Using /home/tgg/rk3568_linux_sdk/tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/config.ini
Program Log will save in the /root/upgrade_tool/log/
directlba=1,first4access=1,gpt=1
Download uboot start...(0x00004000)
Download image ok.
Using /home/tgg/rk3568_linux_sdk/tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/config.ini
Program Log will save in the /root/upgrade_tool/log/
check download item failed!
Using /home/tgg/rk3568_linux_sdk/tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/config.ini
Program Log will save in the /root/upgrade_tool/log/
directlba=1,first4access=1,gpt=1
Download boot start...(0x00008000)
Download image ok.
Using /home/tgg/rk3568_linux_sdk/tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/config.ini
Program Log will save in the /root/upgrade_tool/log/
directlba=1,first4access=1,gpt=1
Download recovery start...(0x00028000)
Download image ok.
Using /home/tgg/rk3568_linux_sdk/tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/config.ini
Program Log will save in the /root/upgrade_tool/log/
directlba=1,first4access=1,gpt=1
Download misc start...(0x00006000)
Download image ok.
Using /home/tgg/rk3568_linux_sdk/tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/config.ini
Program Log will save in the /root/upgrade_tool/log/
directlba=1,first4access=1,gpt=1
Download oem start...(0x00c58000)
Download image ok.
Using /home/tgg/rk3568_linux_sdk/tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/config.ini
Program Log will save in the /root/upgrade_tool/log/
directlba=1,first4access=1,gpt=1
Download userdata start...(0x00c98000)
Download image ok.
Using /home/tgg/rk3568_linux_sdk/tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/config.ini
```

Figure 5.3.5.1 Execute rkflash.sh to burn all the images.

Executing the above command will burn the boot.img, MiniLoaderAll.bin, misc.img, OEM.img, recovery.img, rootfs.img, uboot.img and userdata.img in the rockdev directory to the corresponding partitions of the development board.

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

In addition, it is also possible to burn a specific image separately, as shown in the table below:

Command	Function
sudo ./rkflash.sh all	Burn all images
sudo ./rkflash.sh	Burn all images, equivalent to the above
sudo ./rkflash.sh loader	Write Loader (which is MiniLoaderAll.bin)
sudo ./rkflash.sh parameter	Download partition table parameter.txt
sudo ./rkflash.sh uboot	Burn uboot.img
sudo ./rkflash.sh boot	Burn boot.img
sudo ./rkflash.sh recovery	Burn recovery.img
sudo ./rkflash.sh misc	Burn misc.img
sudo ./rkflash.sh oem	Burn OEM.img
sudo ./rkflash.sh userdata	Burn userdata.img
sudo ./rkflash.sh rootfs	Burn root file system image rootfs.img
sudo ./rkflash.sh update	Burn update.img image package
sudo ./rkflash.sh erase	Erase all data in the Flash storage

Table 5.3.5.1 Usage of the rkflash.sh script

Chapter 6. SDK Development

This chapter provides users with instructions on how to develop with the RK3568 Linux SDK, covering u-boot development, Linux kernel development, buildroot root file system development, Qt application development, C/C++ application development, etc. Users can conduct secondary development and software customization based on this SDK to adapt it to their own Linux products. Based on this SDK, system customization and application porting development can be effectively achieved, helping users to quickly develop and improve their development efficiency!

6.1 SDK Board Configuration File

This section introduces the board configuration file of the SDK to users. The SDK board configuration file provides some necessary configuration information. For the RK3568 platform, its board configuration file is located in the **<SDK>/device/rockchip/rk356x/** directory, as shown below:

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/device/rockchip/rk356x$ pwd
/home/tgg/rk3568_linux_sdk/device/rockchip/rk356x
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/device/rockchip/rk356x$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/device/rockchip/rk356x$ ls -lh
总用量 64K
lrwxrwxrwx 1 tgg tgg   35 4月   6 15:32 BoardConfig.mk -> BoardConfig-rk3568-evb1-ddr4-v10.mk
-rw-rw-r-- 1 tgg tgg 1.9K 4月   6 15:32 BoardConfig-rk3566-evb2-lp4x-v10-32bit.mk
-rw-rw-r-- 1 tgg tgg 1.9K 4月   6 15:32 BoardConfig-rk3566-evb2-lp4x-v10.mk
-rw-rw-r-- 1 tgg tgg 1.9K 4月   6 15:32 BoardConfig-rk3568-atk-evb1-ddr4-v10.mk
-rw-rw-r-- 1 tgg tgg 1.9K 4月   6 15:32 BoardConfig-rk3568-evb1-ddr4-v10-32bit.mk
-rw-rw-r-- 1 tgg tgg 1.9K 4月   6 15:32 BoardConfig-rk3568-evb1-ddr4-v10.mk
-rwxrwxr-x 1 tgg tgg 2.1K 4月   6 15:32 BoardConfig-rk3568-evb1-ddr4-v10-spi-nor-64M.mk
-rw-rw-r-- 1 tgg tgg 2.0K 4月   6 15:32 BoardConfig-rk3568-nvr.mk
-rw-rw-r-- 1 tgg tgg 2.7K 4月   6 15:32 BoardConfig-rk3568-nvr-spi-nand.mk
-rw-rw-r-- 1 tgg tgg 1.9K 4月   6 15:32 BoardConfig-rk3568-uvc-evb1-ddr4-v10.mk
-rw-rw-r-- 1 tgg tgg 1.7K 4月   6 15:32 boot4recovery.its
-rw-rw-r-- 1 tgg tgg 1.5K 4月   6 15:32 boot.its
-rw-rw-r-- 1 tgg tgg  488 4月   6 15:32 parameter-buildroot-fit.txt
-rw-rw-r-- 1 tgg tgg  458 4月   6 15:32 parameter-buildroot-NVR-128M.txt
-rw-rw-r-- 1 tgg tgg  399 4月   6 15:32 parameter-buildroot-NVR-spi-nand-128M.txt
-rw-rw-r-- 1 tgg tgg  373 4月   6 15:32 parameter-buildroot-spi-nor-64M.txt
-rw-rw-r-- 1 tgg tgg 1.5K 4月   6 15:32 zboot.its
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/device/rockchip/rk356x$
```

Figure 6.1-1 Board-level configuration files

There are multiple BoardConfig-xxxx.mk files in this directory. These.mk files are the board-level configuration files. Among them, **BoardConfig-rk3568-atk-evb1-ddr4-v10.mk** is the board-level configuration file used by our ATK-DLRK3568 development board; the files listed when we execute "**./build.sh lunch**" in the SDK root directory are from the <SDK>/device/rockchip/rk356x/ directory, as shown below:

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk$ ./build.sh lunch
processing option: lunch

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

0. default BoardConfig.mk
1. BoardConfig-rk3566-evb2-lp4x-v10-32bit.mk
2. BoardConfig-rk3566-evb2-lp4x-v10.mk
3. BoardConfig-rk3568-atk-evb1-ddr4-v10.mk
4. BoardConfig-rk3568-evb1-ddr4-v10-32bit.mk
5. BoardConfig-rk3568-evb1-ddr4-v10-spi-nor-64M.mk
6. BoardConfig-rk3568-evb1-ddr4-v10.mk
7. BoardConfig-rk3568-nvr-spi-nand.mk
8. BoardConfig-rk3568-nvr.mk
9. BoardConfig-rk3568-uvc-evb1-ddr4-v10.mk
10. BoardConfig.mk
Which would you like? [0]:
```

Figure 6.1-2 List the configuration files

These .mk files are actually shell script files. Let's open the BoardConfig-rk3568-atk-evb1-ddr4-v10.mk configuration file to take a look at its contents:

```
#!/bin/bash


# Target arch
export RK_ARCH=arm64
# Uboot defconfig
```

```
export RK_UBOOT_DEFCONFIG=rk3568
# Uboot image format type: fit(flattened image tree)
export RK_UBOOT_FORMAT_TYPE=fit
# Kernel defconfig
export RK_KERNEL_DEFCONFIG=rockchip_linux_defconfig
# Kernel defconfig fragment
export RK_KERNEL_DEFCONFIG_FRAGMENT=
# Kernel dts
export RK_KERNEL_DTS=rk3568-atk-evb1-ddr4-v10-linux
# boot image type
export RK_BOOT_IMG=boot.img
# kernel image path
export RK_KERNEL_IMG=kernel/arch/arm64/boot/Image
# kernel image format type: fit(flattened image tree)
export RK_KERNEL_FIT_ITS=boot.its
# parameter for GPT table
export RK_PARAMETER=parameter-buildroot-fit.txt
# Buildroot config
export RK_CFG_BUILDROOT=rockchip_rk3568
# Recovery config
export RK_CFG_RECOVERY=rockchip_rk356x_recovery
# Recovery image format type: fit(flattened image tree)
export RK_RECOVERY_FIT_ITS=boot4recovery.its
# ramboot config
export RK_CFG_RAMBOOT=
# Pcba config
export RK_CFG_PCBA=
# Build jobs
export RK_JOBS=24
# target chip
export RK_TARGET_PRODUCT=rk356x
# Set rootfs type, including ext2 ext4 squashfs
export RK_ROOTFS_TYPE=ext4
# Set debian version (debian10: buster, debian11: bullseye)
export RK_DEBIAN_VERSION=buster
# yocto machine
export RK_YOCTO_MACHINE=rockchip-rk3568-evb
# rootfs image path
export RK_ROOTFS_IMG=rockdev/rootfs.${RK_ROOTFS_TYPE}
# Set ramboot image type
export RK_RAMBOOT_TYPE=
# Set oem partition type, including ext2 squashfs
export RK_OEM_FS_TYPE=ext2
```

```
# Set userdata partition type, including ext2, fat
export RK_USERDATA_FS_TYPE=ext2
#OEM config
export RK_OEM_DIR=oem_normal
# OEM build on buildroot
#export RK_OEM_BUILDIN_BUILDROOT=YES
#userdata config
export RK_USERDATA_DIR=userdata_normal
#misc image
export RK_MISC=wipe_all-misc.img
#choose enable distro module
export RK_DISTRO_MODULE=
# Define pre-build script for this board
export RK_BOARD_PRE_BUILD_SCRIPT=app-build.sh
```

In this script, some environment variables are exported as follows:

RK_ARCH: used to specify the target architecture, rk3568 corresponds to arm64;

RK_UBOOT_DEFCONFIG: used to specify the defconfig configuration file for U-Boot; rk3568_defconfig.

RK_UBOOT_FORMAT_TYPE: used to specify the format of the uboot.img image, the default format for the rk3568 platform is fit (flattened image tree);

RK_KERNEL_DEFCONFIG: used to specify the defconfig configuration file for the Linux Kernel (kernel); rockchip_linux_defconfig.

RK_KERNEL_DEFCONFIG_FRAGMENT: used to specify the defconfig fragment of the Linux kernel, for rk3568 it is empty;

RK_KERNEL_DTS: used to specify the kernel device tree file; rk3568-atk-evb1-ddr4-v10-linux.dts.

RK_BOOT_IMG: set it to boot.img.

RK_KERNEL_IMG: Specifies the path of the kernel image; kernel/arch/arm64/boot/Image.

RK_KERNEL_FIT_ITS: The boot.img used for the Linux system on the rk3568 platform is also a FIT format image. FIT uses its files (image source files) to describe the information of the image. RK_KERNEL_FIT_ITS is used to specify this its file. This its file must be stored in the <SDK>/device/rockchip/rk356x/ directory; boot.its.

RK_PARAMETER: Specifies the partition table file, which is the parameter.txt file introduced earlier.

RK_CFG_BUILDROOT: Specifies the defconfig configuration file of the buildroot root file system (in normal mode); rockchip_rk3568_defconfig.

RK_CFG_RECOVERY: Specifies the defconfig configuration file of the root file system in recovery mode (using the ramdisk root file system in recovery mode); rockchip_rk356x_recovery_defconfig.

RK_RECOVERY_FIT_ITS: Specifies the its file corresponding to the recovery.img image. The recovery.img is also a FIT format image. It is necessary to use the its file to describe the information of the image; boot4recovery.its.

RK_CFG_RAMBOOT: Is left blank by default, but its meaning is unknown.

RK_CFG_PCBA: Specifies the defconfig configuration file of the PCBA;

RK_JOBS: Used to specify the number of threads for the make compilation, for example: make -j24;

RK_TARGET_PRODUCT: Used to specify the target product name. For the rk3568 platform, it is set to rk356x by default;

RK_ROOTFS_TYPE: Used to specify the type of the root file system, such as ext2, ext4;

RK_DEBIAN_VERSION: Used to specify the Debian version, debian10: buster, debian11: bullseye. The default is Debian 10, do not modify it;

RK_YOCTO_MACHINE: Used for specifying the machine when compiling yocto;

RK_ROOTFS_IMG: Used to specify the path of the root file system image;

RK_RAMBOOT_TYPE: The default is empty, but its meaning is unknown;

RK_OEM_FS_TYPE: Used to specify the type of the OEM partition, keep it as the default;

RK_USERDATA_FS_TYPE: Used to specify the type of the userdata partition, keep it as the default;

RK_OEM_DIR: Used to specify the folder corresponding to OEM, <SDK>/device/rockchip/oem directory has multiple OEM_xxx folders, these folders store the manufacturer's APP or data, and will eventually be compiled into the OEM.img image;

RK_USERDATA_DIR: Used to specify the folder corresponding to userdata, <SDK>/device/rockchip/userdata directory has multiple userdata_xxx folders, these folders store the final user's APP or data, and will eventually be compiled into the userdata.img image;

RK_MISC: Used to specify the misc image. After compiling the SDK, the <SDK>/rockdev/misc.img image generated is actually the file specified by RK_MISC, just renamed; The misc image specified by RK_MISC must be stored in the <SDK>/device/rockchip/rockimg directory;

RK_DISTRO_MODULE: Not used;

RK_BOARD_PRE_BUILD_SCRIPT:

That's all about this board-level configuration file. Users can add their own board-level configuration files in the **<SDK>/device/rockchip/rk356x** directory. They can modify the variables in the configuration file according to the actual situation or add new variables.

6.2 U-Boot Development

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

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/u-boot$ pwd
/home/tgg/rk3568_linux_sdk/u-boot
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/u-boot$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/u-boot$ ls
api        common     Documentation  fit      lib          net            snapshot.commit
arch       config.mk  drivers        fs       Licenses     post           test
bl31.elf   configs    dts            include  MAINTAINERS  PREUPLOAD.cfg  tools
board      disk       env            Kbuild   Makefile     README
cmd        doc        examples       Kconfig  make.sh      scripts
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/u-boot$
```

Figure 6.2-1 U-Boot source code directory

RK has provided a document that details the knowledge points and technical points related to U-Boot on the Rockchip platform for users, including an introduction to the basics of U-Boot on the

Rockchip platform, the U-Boot architecture, the U-Boot startup process, the 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 it. The path of this document is: **<SDK>/docs/Common/UBOOT/Rockchip_Developer_Guide_UBoot_Nextdev_CN.pdf**. Similarly, the development board package ATK-DLRK3568 also provides this document, with the path being: Development board CD-ROM A disk - **Basic materials→8_RK_official_documents→Linux Common→U-BOOT→Rockchip_Developer_Guide_UBoot_Nextdev_CN.pdf.**

6.2.1 The device tree of U-Boot

In U-Boot, the device tree file for RK3568 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 purpose of this design is mainly 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, the storage and debug serial port initialization operations are first completed using the U-Boot device tree, and then the kernel's device tree is loaded from storage and the device tree is used to continue initializing the remaining peripherals.

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

6.2.2 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_linux_sdk/u-boot$ ./make.sh rk3568
## make rk3568_defconfig -j24
#
# configuration written to .config
#
scripts/kconfig/conf  --silentoldconfig Kconfig
  CHK     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     tpl/include/autoconf.mk
  GEN     spl/include/autoconf.mk
  CHK     include/config/uboot.release
  CHK     include/generated/timestamp_autogenerated.h
  UPD     include/generated/timestamp_autogenerated.h
  CHK     include/config.h
  CFG     u-boot.cfg
  CHK     include/generated/version_autogenerated.h
  CHK     include/generated/generic-asm-offsets.h
  CHK     include/generated/asm-offsets.h
  HOSTCC  tools/mkenvimage.o
  HOSTCC  tools/fit_image.o
  HOSTCC  tools/image-host.o
  HOSTCC  tools/dumpimage.o
  HOSTCC  tools/mkimage.o
  HOSTCC  tools/rockchip/boot_merger.o
  HOSTCC  tools/rockchip/loaderimage.o
  HOSTLD  tools/mkenvimage
  HOSTLD  tools/loaderimage
  HOSTLD  tools/dumpimage
  HOSTLD  tools/mkimage
tools/rockchip/boot_merger.c: In function ‘main’:
tools/rockchip/boot_merger.c:895:11: warning: array subscript 20 is outside array bounds of ‘char[20]’ [-Warray-bounds]
  895 |  str[len] = 0;
      |  ~~~~~~~~~~^~~
tools/rockchip/boot_merger.c:933:7: note: while referencing ‘name’
  933 |  char name[MAX_NAME_LEN];
      |       ^~~~
```

Figure 6.2.2.1 Compile the U-Boot source code (1)

```
 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_linux_sdk/rkbin/RKBOOT/RK3568MINIALL.ini
/home/tgg/rk3568_linux_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_linux_sdk/rkbin/RKTRUST/RK3568TRUST.ini

Platform RK3568 is build OK, with new .config(make rk3568_defconfig -j24)
/home/tgg/rk3568_linux_sdk/prebuilts/gcc/linux-x86/aarch64/gcc-linaro-6.3.1-2017.05-x86_64_aarch64-linux-gnu/bin/aarch64-linux-gnu-
2023年 04月 10日 星期一 18:23:23 CST
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/u-boot$
```

Figure 6.2.2.2 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_linux_sdk/u-boot$ ls
api                  config.mk      Kbuild                        scripts              u-boot.dtb
arch                 configs        Kconfig                       snapshot.commit      u-boot-dtb.bin
bl31_0x00040000.bin  disk           lib                           spl                  uboot.img
bl31_0x00068000.bin  doc            Licenses                      System.map           u-boot.lds
bl31_0x0006a000.bin  Documentation  MAINTAINERS                   tee.bin              u-boot.map
bl31_0xfdcc1000.bin  drivers        Makefile                      test                 u-boot-nodtb.bin
bl31_0xfdcce000.bin  dts            make.sh                       tools                u-boot.srec
bl31_0xfdcd0000.bin  env            net                           tpl                  u-boot.sym
bl31.elf             examples       post                          u-boot
board                fit            PREUPLOAD.cfg                 u-boot.bin
cmd                  fs             README                        u-boot.cfg
common               include        rk356x_spl_loader_v1.13.112.bin u-boot.cfg.configs
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/u-boot$
```

Figure 6.2.2.3 The image file generated after U-Boot compilation

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

1. uboot.img image

As mentioned earlier, uboot.img is a combination of multiple images, including the u-boot image, u-boot dtb, and the trust image (ARM Trusted Firmware + OP-TEE). uboot.img is a FIT (**flattened image tree**) format image and supports the packaging and verification of any number of images. To view uboot.img using the file command, it is as follows:

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/u-boot$
tgg@tgg-virtual-machine:~/rk3568_linux_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_linux_sdk/u-boot$
tgg@tgg-virtual-machine:~/rk3568_linux_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 generally 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_linux_sdk/u-boot$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/u-boot$ ls -l *.bin *.dtb
-rw-rw-r-- 1 tgg tgg  163840 4月  11 10:21 bl31_0x00040000.bin
-rw-rw-r-- 1 tgg tgg    7732 4月  11 10:21 bl31_0x00068000.bin
-rw-rw-r-- 1 tgg tgg   20267 4月  11 10:21 bl31_0x0006a000.bin
-rw-rw-r-- 1 tgg tgg   40960 4月  11 10:21 bl31_0xfdcc1000.bin
-rw-rw-r-- 1 tgg tgg    8192 4月  11 10:21 bl31_0xfdcce000.bin
-rw-rw-r-- 1 tgg tgg    8192 4月  11 10:21 bl31_0xfdcd0000.bin
-rw-rw-r-- 1 tgg tgg  465344 4月  11 10:21 rk356x_spl_loader_v1.13.112.bin
-rw-rw-r-- 1 tgg tgg  457112 4月  11 10:21 tee.bin
-rw-rw-r-- 1 tgg tgg 1282096 4月  11 10:21 u-boot.bin
-rw-rw-r-- 1 tgg tgg   14377 4月  11 10:21 u-boot.dtb
-rw-rw-r-- 1 tgg tgg 1282089 4月  11 10:21 u-boot-dtb.bin
-rwxrwxr-x 1 tgg tgg 1267712 4月  11 10:21 u-boot-nodtb.bin
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/u-boot$
```

Figure 6.2.2.4 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
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, but does not include u-boot dtb.
bl31_0x00040000.bin	All 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 file are both in the <U-Boot>/fit directory:

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/u-boot$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/u-boot$ ls -l fit/*
-rw-rw-r-- 1 tgg tgg 1994752 4月  11 11:07 fit/uboot.itb
-rw-rw-r-- 1 tgg tgg    2805 4月  11 11:07 fit/u-boot.its
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/u-boot$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/u-boot$
```

Figure 6.2.2.5 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_linux_sdk/u-boot$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/u-boot$ ls tee.bin bl31.elf -lh
-rwxrwxr-x 1 tgg tgg 393K 4月  11 11:07 bl31.elf
-rw-rw-r-- 1 tgg tgg 447K 4月  11 11:07 tee.bin
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/u-boot$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/u-boot$
```

Figure 6.2.2.6 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_v1.33.elf** and **<SDK>/rkbin/bin/rk35/rk3568_bl32_v2.08.bin**. During the packaging of uboot.itb, <SDK>/rkbin/bin/rk35/rk3568_bl31_v1.33.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_linux_sdk/u-boot$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/u-boot$ ls -l bl31_0x*
-rw-rw-r-- 1 tgg tgg 163840 4月  11 11:07 bl31_0x00040000.bin
-rw-rw-r-- 1 tgg tgg   7732 4月  11 11:07 bl31_0x00068000.bin
-rw-rw-r-- 1 tgg tgg  20267 4月  11 11:07 bl31_0x0006a000.bin
-rw-rw-r-- 1 tgg tgg  40960 4月  11 11:07 bl31_0xfdcc1000.bin
-rw-rw-r-- 1 tgg tgg   8192 4月  11 11:07 bl31_0xfdcce000.bin
-rw-rw-r-- 1 tgg tgg   8192 4月  11 11:07 bl31_0xfdcd0000.bin
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/u-boot$
```

Figure 6.2.2.7 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 mentioned earlier, MiniLoaderAll.bin is a Loader code that runs before U-Boot on the RK3568 platform (it is a Loader code at a stage earlier than U-Boot).

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

TPL runs in SRAM (internal memory) and is started by the Maskrom (BootROM) code inside the rk3568 chip; its function is to complete the initialization of DRAM and start SPL; SPL runs in DDR and its function is to complete the low-level initialization of the system and 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 offer their 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 the binary files are available.

Open source version

The tpl and spl images can also be compiled from the U-Boot source code. Since it is part of the U-Boot source code, it is 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_linux_sdk/u-boot$ cd spl/
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/u-boot/spl$
tgg@tgg-virtual-machine:~/rk3568_linux_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_linux_sdk/u-boot/spl$
```

Figure 6.2.2.8 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 dtb
u-boot-spl-dtb.bin	Spl image, including spl dtb, is the same as u-boot-spl.bin
u-boot-spl-nodtb.bin	Spl image, without including spl dtb

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_linux_sdk/u-boot/tpl$
tgg@tgg-virtual-machine:~/rk3568_linux_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_linux_sdk/u-boot/tpl$
```

Figure 6.2.2.9 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_linux_sdk/rkbin/RKBOOT$ pwd
/home/tgg/rk3568_linux_sdk/rkbin/RKBOOT
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/rkbin/RKBOOT$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/rkbin/RKBOOT$ ls
PX30MINIALL.ini              RK3126MINIALL_SLC.ini          RK3566MINIALL_NAND.ini
PX30MINIALL_SLC.ini          RK3126MINIALL_WO_FTL.ini       RK3566MINIALL_ULTRA.ini
PX30MINIALL_WO_FTL.ini       RK3128.ini                     RK3568MINIALL.ini
PX3SEMINIALL.ini             RK3128MINIALL.ini              RK3568MINIALL_NAND.ini
PX3SEMINIALL_SLC.ini         RK3128MINIALL_SLC.ini          RK3568MINIALL_RAMBOOT.ini
PX5KERNEL4.4MINIALL.ini      RK3128XMINIALL.ini             RK3568MINIALL_SPI_NAND.ini
PX5MINIALL.ini               RK3188MINIALL.ini              RK3588MINIALL.ini
RK1806MINIALL.ini            RK322XATMINIALL.ini            RK3588MINIALL_IPC.ini
RK1808MINIALL.ini            RK322XHMINIALL.ini             RK3588MINIALL_RAMBOOT.ini
RK1808MINIALL_WO_FTL.ini     RK322XMINIALL.ini              RKCAYMAN.ini
RK302A.ini                   RK3288.ini                     RKCROWN.ini
RK302AMINIALL.ini            RK3288MINIALL.ini              RKNANO.ini
RK302AMINI.ini               RK3308MINIALL.ini              RKNPULIONMINIALL.ini
RK3032MINIALL.ini            RK3308MINIALL_SPI_NAND.ini     RKPANDA.ini
RK3032MINIALL_SLC.ini        RK3308MINIALL_UART4.ini        RKSMART.ini
RK3036_ECHOMINIALL.ini       RK3308MINIALL_WO_FTL.ini       RV1106MINIALL_EMMC_TB.ini
RK3036.ini                   RK3326AARCH32MINIALL.ini       RV1106MINIALL.ini
RK3036MINIALL.ini            RK3326AARCH32MINIALL_SLC.ini   RV1106MINIALL_SPI_NOR_TB.ini
RK3036MINIALL_SLC.ini        RK3326MINIALL.ini              RV110XMINIALL.ini
RK30B.ini                    RK3326MINIALL_SLC.ini          RV1126MINIALL_EMMC_TB.ini
RK30BMINIALL.ini             RK3328MINIALL.ini              RV1126MINIALL_FTL.ini
RK30BMINI.ini                RK3366MINIALL.ini              RV1126MINIALL.ini
RK30.ini                     RK3368BOXMINIALL.ini           RV1126MINIALL_IPC.ini
RK30MINIALL.ini              RK3368HMINIALL.ini             RV1126MINIALL_LP3_EMMC_TB.ini
RK30MINI.ini                 RK3368.ini                     RV1126MINIALL_LP4_EMMC_TB.ini
RK310B.ini                   RK3368MINIALL.ini              RV1126MINIALL_RAMBOOT.ini
RK310BMINIALL.ini            RK3399MINIALL.ini              RV1126MINIALL_SPI_NOR_TB.ini
RK310BMINI.ini               RK3399MINIALL_SPINOR.ini       RV1126MINIALL_SPI_NOR_TINY.ini
RK3126.ini                   RK3399PROMINIALL.ini           SDBOOT.ini
RK3126MINIALL.ini            RK3566MINIALL.ini
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/rkbin/RKBOOT$ _
```

Figure 6.2.2.10 .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 TPL part of the U-Boot 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 6.2.2.11 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_linux_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_linux_sdk/u-boot$
```

Figure 6.2.2.12 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 **ARM Trusted Firmware + OP-TEE** combination 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 secure boot process is defined as: BL1, BL2, BL31, BL32, 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, also called BootROM, 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 → rootfs.img

It can also be further divided into:

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

This startup process can be analyzed simply by printing the information. Here 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 6.2.2.13 TPL (DDR BIN) has been run.

```
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 6.2.2.14 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 6.2.2.15 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 6.2.2.16 The operation process of U-Boot

6.2.3 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:

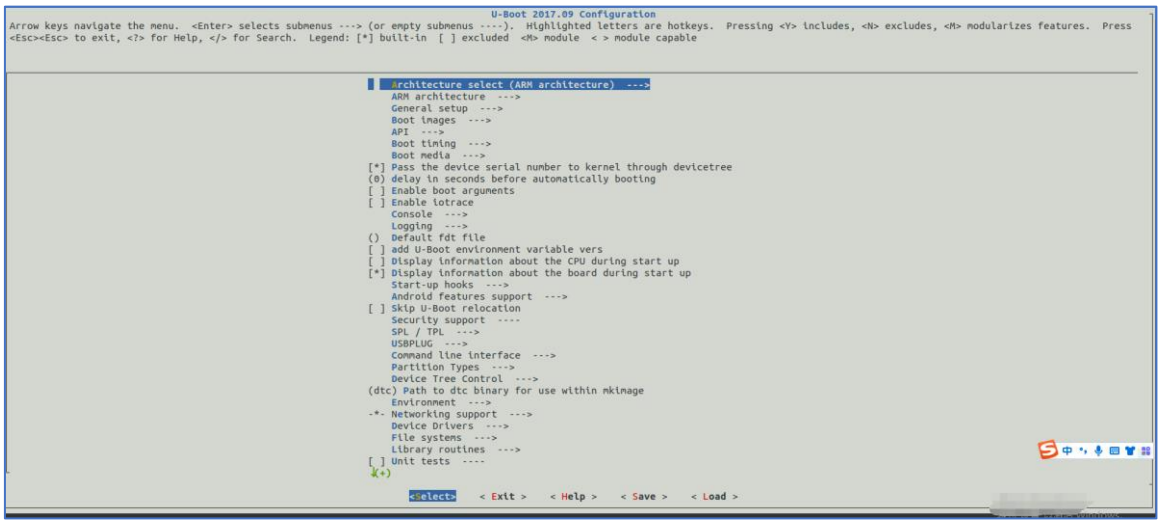


Figure 6.2.3.1 Menuconfig graphical configuration

Through the menuconfig graphical interface, users can configure U-Boot 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 uboot
// Execute the "build.sh" script in the root directory of the SDK for compilation.
```

6.2.4 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 (or build.sh) script, and 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 root 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_linux_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 between 1 a
  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 4104) i
   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 and 40
  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 into a
   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
#
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/u-boot$
```

Figure 6.2.4.1 Configure U-Boot

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/u-boot$ make PYTHON=python2 CROSS_COMPILE=/home/tgg/rk3568_linux_sdk/prebuilts/gcc/l
inux-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
  SHIPPED scripts/dtc/dtc-parser.tab.c
  HOSTCC  scripts/dtc/dtc-parser.tab.o
  HOSTCC  scripts/dtc/dtc-lexer.lex.o
  UPD     include/config/uboot.release
  CHK     include/generated/version_autogenerated.h
  UPD     include/generated/version_autogenerated.h
  CC      lib/asm-offsets.s
  CC      arch/arm/lib/asm-offsets.s
```

Figure 6.2.4.2 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_linux_sdk/u-boot$
tgg@tgg-virtual-machine:~/rk3568_linux_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_linux_sdk/rkbin/RKTRUST/RK3568TRUST.ini

FIT description: FIT Image with ATF/OP-TEE/U-Boot/MCU
Created:         Tue Apr 11 17:50:31 2023
 Image 0 (uboot)
  Description:  U-Boot
  Created:      Tue Apr 11 17:50:31 2023
  Type:         Standalone Program
  Compression:  uncompressed
  Data Size:    1267712 Bytes = 1238.00 KiB = 1.21 MiB
  Architecture: AArch64
  Load Address: 0x00a00000
  Entry Point:  unavailable
  Hash algo:    sha256
  Hash value:   b2a3d3cc74e7a4b9119e04e4f398b5ac62d8691de72ed9d211cb43c3144e454f
 Image 1 (atf-1)
  Description:  ARM Trusted Firmware
  Created:      Tue Apr 11 17:50:31 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:   6204b6f381d2ad6175ffa52cc82abe8c934eb23aaf8a00c7db2dda0367449fdc
 Image 2 (atf-2)
  Description:  ARM Trusted Firmware
  Created:      Tue Apr 11 17:50:31 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:   5563d929dab73f22d2229dd7933b58de3dd6553334fc653502e97b1bd561365f
 Image 3 (atf-3)
  Description:  ARM Trusted Firmware
  Created:      Tue Apr 11 17:50:31 2023
  Type:         Firmware
  Compression:  uncompressed
  Data Size:    20267 Bytes = 19.79 KiB = 0.02 MiB
  Architecture: AArch64
```

Figure 6.2.4.3 Perform the packaging operation (1)

```
  Hash value:   66bbd173528d12e9739c336926e33ee1ac1f4c7078fcac5712eeb8747d02163e
 Image 8 (fdt)
  Description:  U-Boot dtb
  Created:      Tue Apr 11 17:50:31 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: ../rkbin/RKBOOT/RK3568MINIALL.ini
/home/tgg/rk3568_linux_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_linux_sdk/u-boot$
```

Figure 6.2.4.4 Perform the packaging operation (2)

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

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/u-boot$ ls -l rk356x_spl_loader_v1.13.112.bin uboot.img
-rw-rw-r-- 1 tgg tgg  465344 4月  11 17:50 rk356x_spl_loader_v1.13.112.bin
-rw-rw-r-- 1 tgg tgg 4194304 4月  11 17:50 uboot.img
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/u-boot$
```

Figure 6.2.4.5 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_linux_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/
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/
 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
 bl31_0xfdcd0000.bin bl31_0x00040000.bin bl31_0xfdcce000.bin bl31_0x0006a000.bin u-boot-nodtb.bin bl31_0
  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
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/u-boot$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/u-boot$
```

Figure 6.2.4.6 Cleaning U-Boot

6.2.5 Shortcuts

On the RK platform, U-Boot supports triggering some 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.

6.2.6 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 5.5-inch 720p MIPI screens (ALIENTEK 5.5-inch MIPI screen) and also supports 5.5-inch 1080p MIPI screens (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 corresponding DTB for the 720p MIPI screen needs to be burned, and when using the 1080p MIPI screen, the corresponding DTB for the 1080p MIPI screen needs to be burned**. This is very troublesome!

At this time, the HW-ID DTB can solve this problem. 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, it supports the detection of two hardware states: ADC and GPIO.

The MIPI screen interface of the ALIENTEK ATK-DLRK3568 development board is designed with a SARADC_VIN2_LCD_ID pin:

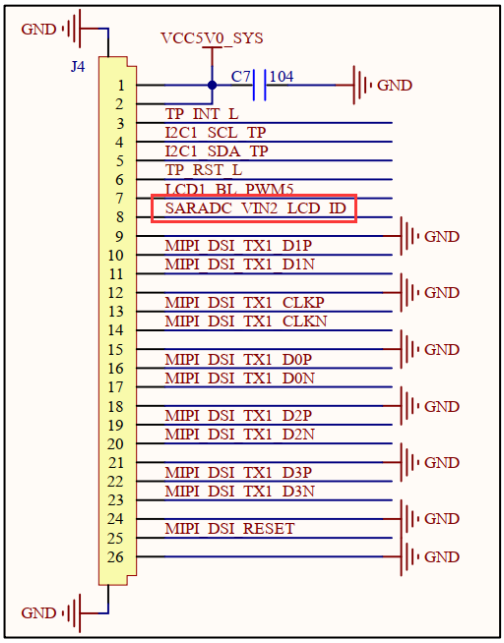
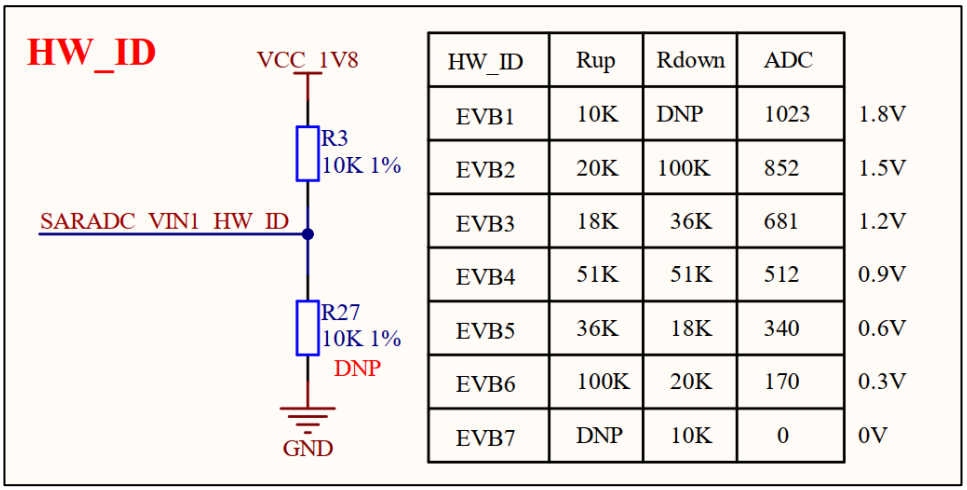


Figure 6.2.6.1 The MIPI screen interface

SARADC_VIN2_LCD_ID is an ADC pin (using ADC channel 2). When a user connects a 720p MIPI screen or a 1080p MIPI screen, the corresponding ADC value is different (as different voltage divider resistors are used in the hardware design for the two screens). Before U-Boot loads the kernel DTB, it reads this ADC value and loads the matching DTB based on the read ADC value. Thus, the DTB can be dynamically loaded according to the MIPI screen connected by the user (provided that the DTB files for both screens are all packaged into resource.img).

In addition, the ATK-DLRK3568 motherboard also reserves a board-level HW_ID (for backup):



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 6.2.6.2 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 the 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 the kernel DTB is as follows:

ADC is used as the 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 called a complete unit, describing the unique information of the ADC hardware:

[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 effective range is adcval+-30.

- Each complete unit (#_[controller]_ch[channel]=[adcval]) must use lowercase letters, and there must be no spaces inside;
- Multiple complete units are separated by # symbol, and the maximum number of units supported is 10. If a DTB needs to meet multiple unique values of ADC hardware to be matched, then the naming of this DTB file 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 within the range of 111 ± 30, it will match this DTB; the second one indicates that it will match this DTB only when the ADC value read from ADC channel 2 is within the range of 111 ± 30 and the ADC value read from ADC channel 1 is within the range of 810 ± 30 (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]**", where "#_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 within it;
- Multiple complete units are separated by the "#" symbol, and the maximum number of units supported is 10. If a DTB needs to meet multiple unique values of 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 at a high level, it will match this DTB; the second DTB file name indicates that it will only match this DTB when GPIO0_C1 is at a high level and GPIO2_B3 is at a low level (condition 1 && condition 2).

3. DTB Packaging

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

4. Function Activation

The RK U-Boot default does not enable the HW-ID DTB function. However, ALIENTEK Microelectronics has enabled it as the ATK-DLRK3568 development board's 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 6.2.6.3 Disable HW-ID DTB

6.3 kernel development

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

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel$ pwd
/home/tgg/rk3568_linux_sdk/kernel
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel$ ls
android                          build.config.gki                build.config.x86_64  ipc              mm
arch                             build.config.gki.aarch64        certs                Kbuild           net
block                            build.config.gki-debug.aarch64  COPYING              Kconfig          OWNERS
boot.its                         build.config.gki-debug.x86_64   CREDITS              kernel           README
build.config.aarch64             build.config.gki_kasan          crypto               lib              samples
build.config.allmodconfig        build.config.gki_kasan.aarch64  Documentation        LICENSES         scripts
build.config.allmodconfig.aarch64 build.config.gki_kasan.x86_64  drivers              logo.bmp         security
build.config.allmodconfig.arm    build.config.gki_kprobes        firmware             logo_kernel.bmp  sound
build.config.allmodconfig.x86_64 build.config.gki_kprobes.aarch64 fs                  MAINTAINERS      tools
build.config.arm                 build.config.gki_kprobes.x86_64 include              Makefile         usr
build.config.common              build.config.gki.x86_64         init                 make.sh          virt
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel$
```

Figure 6.3-1 Kernel source code directory

This section introduces the content related to kernel development.

6.3.1 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_linux_sdk/kernel$
tgg@tgg-virtual-machine:~/rk3568_linux_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
#
  WRAP    arch/arm64/include/generated/uapi/asm/errno.h
  WRAP    arch/arm64/include/generated/uapi/asm/ioctl.h
  WRAP    arch/arm64/include/generated/uapi/asm/ioctls.h
  WRAP    arch/arm64/include/generated/uapi/asm/ipcbuf.h
  WRAP    arch/arm64/include/generated/uapi/asm/kvm_para.h
  WRAP    arch/arm64/include/generated/uapi/asm/mman.h
  WRAP    arch/arm64/include/generated/uapi/asm/msgbuf.h
  WRAP    arch/arm64/include/generated/uapi/asm/poll.h
  WRAP    arch/arm64/include/generated/uapi/asm/resource.h
  WRAP    arch/arm64/include/generated/uapi/asm/sembuf.h
  WRAP    arch/arm64/include/generated/uapi/asm/shmbuf.h
  WRAP    arch/arm64/include/generated/uapi/asm/socket.h
  WRAP    arch/arm64/include/generated/uapi/asm/sockios.h
  WRAP    arch/arm64/include/generated/uapi/asm/swab.h
  WRAP    arch/arm64/include/generated/uapi/asm/termbits.h
  WRAP    arch/arm64/include/generated/uapi/asm/termios.h
  WRAP    arch/arm64/include/generated/uapi/asm/types.h
  UPD     include/generated/uapi/linux/version.h
  UPD     include/config/kernel.release
  WRAP    arch/arm64/include/generated/asm/bugs.h
  WRAP    arch/arm64/include/generated/asm/delay.h
  WRAP    arch/arm64/include/generated/asm/div64.h
  WRAP    arch/arm64/include/generated/asm/dma.h
  WRAP    arch/arm64/include/generated/asm/dma-contiguous.h
```

Figure 6.3.1.1 Compile the kernel (1)

```
  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 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/rkwifi/bcmdhd/bcmdhd.ko
  LD [M]  drivers/net/wireless/rockchip_wlan/rtl8852bs/8852bs.ko
  LD [M]  drivers/net/wireless/marvell/mwifiex/mwifiex_sdio.ko
  LD [M]  drivers/net/wireless/marvell/mwifiex/mwifiex.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-linux.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_linux_sdk/kernel$
```

Figure 6.3.1.2 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 6.3.1.3 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 image used by the Linux system on the RK3568 platform is a FIT format image. It is composed of multiple images. For the RK3568 platform, the boot.img burned to the boot partition of the development board contains the kernel image Image, the kernel DTB, and the resource.img. You can use the file command to view the boot.img file, as shown below:

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel$ file boot.img
boot.img: Android bootimg, kernel (0x10008000), second stage (0x10f00000), page size: 2048
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel$
```

Figure 6.3.1.5 View the "boot.img" file

It can be observed that this file is an Android format boot image (Android bootimg), not in FIT format; what's going on here? **In fact, the boot.img generated by compiling through the make.sh script is indeed an Android format image; the Android format boot.img is used for the Android system on the RK3568 platform, while the FIT format boot.img is used for the Linux system on the RK3568 platform.** This indicates that this boot.img is not the final boot.img to be burned onto the boot partition of the development board (Linux system).

So, the boot.img directly generated by compiling through the make.sh script is not the final used boot.img. The following command can be used to generate the FIT format boot.img. First, enter the top-level directory of <SDK>, and execute the command:

```
device/rockchip/common/mk-fitimage.sh kernel/boot.img device/rockchip/rk356x/boot.its
```

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel$ cd ..
tgg@tgg-virtual-machine:~/rk3568_linux_sdk$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk$ device/rockchip/common/mk-fitimage.sh kernel/boot.img device/rockchip/rk356x/boot.its
fdt {
kernel {
resource {
FIT description: U-Boot FIT source file for arm
Created:         Wed Apr 12 12:14:47 2023
 Image 0 (fdt)
  Description:  unavailable
  Created:      Wed Apr 12 12:14:47 2023
  Type:         Flat Device Tree
  Compression:  uncompressed
  Data Size:    139531 Bytes = 136.26 KiB = 0.13 MiB
  Architecture: AArch64
  Load Address: 0xffffff00
  Hash algo:    sha256
  Hash value:   c42c094101b6f7d220fd826f2b61d69ed9948afff318a6166b81f2eeada511bb
 Image 1 (kernel)
  Description:  unavailable
  Created:      Wed Apr 12 12:14:47 2023
  Type:         Kernel Image
  Compression:  uncompressed
  Data Size:    22603784 Bytes = 22074.01 KiB = 21.56 MiB
  Architecture: AArch64
  OS:           Linux
  Load Address: 0xffffff01
  Entry Point:  0xffffff01
  Hash algo:    sha256
  Hash value:   5b22f84597afe2ee4bc42cb7ec5446f5d378f1d2dce08114638dc0c19c0729e1
 Image 2 (resource)
  Description:  unavailable
  Created:      Wed Apr 12 12:14:47 2023
  Type:         Multi-File Image
  Compression:  uncompressed
  Data Size:    458240 Bytes = 447.50 KiB = 0.44 MiB
  Hash algo:    sha256
  Hash value:   57ea171509199dbfaa27deb1e506cb02fe924d6343ac225e85b32c1460839119
 Default Configuration: 'conf'
 Configuration 0 (conf)
  Description:  unavailable
  Kernel:       kernel
  FDT:          fdt
tgg@tgg-virtual-machine:~/rk3568_linux_sdk$
```

Figure 6.3.1.6 Generate FIT format boot.img

Use the <SDK>/device/rockchip/common/mk-fitimage.sh script to generate the FIT format boot.img image.

The first parameter: Indicates the output path of the output file "boot.img". If using a relative address, it must be relative to the root directory of the SDK source code;

The second parameter: Specifies the path of the its source file. If using a relative address, it must be relative to the root directory of the SDK source code; As mentioned earlier, the FIT format requires the use of an its file to describe the information of the image, so an its source file needs to be specified here.

Executing the above command will package the kernel image, kernel DTB, and resource image resource.img into a FIT format boot.img; in the command, the paths of the kernel image, kernel DTB, and resource.img are not specified because they all have default values (as can be known from the mk-fitimage.sh script), as follows:

Kernel image: The image specified by the RK_KERNEL_IMG variable in the board-level configuration file

Kernel DTB: kernel/arch/arm64/boot/dts/rockchip/$RK_KERNEL_DTS.dtb

Resource image: resource.img (kernel)

RK_KERNEL_IMG and RK_KERNEL_DTS are both variables defined in the board-level configuration files.

```
13 # Kernel dts
14 export RK_KERNEL_DTS=rk3568-atk-evb1-ddr4-v10-linux
15 # boot image type
16 export RK_BOOT_IMG=boot.img
17 # kernel image path
18 export RK_KERNEL_IMG=kernel/arch/arm64/boot/Image
19 # kernel image format type: fit(flattened image tree)
20 export RK_KERNEL_FIT_ITS=boot.its
```

At this point, use the "file" command to check the "**kernel/boot.img" file in <SDK>**:

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk$ cd kernel/
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel$ file boot.img
boot.img: Device Tree Blob version 17, size=1536, boot CPU=0, string block size=190, DT structure block size=1004
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel$
```

Figure 6.3.1.7 View the "boot.img" file.

"**Device Tree Blob**" indicates that this file is a FIT format image. The "boot.img" generated by the mk-fitimage.sh script is the actual boot.img that is ultimately burned onto the boot partition (Linux system) of the development board.

When using the mk-fitimage.sh script to generate the boot.img image, it will search for the kernel image Image in the <Kernel>/arch/arm64/boot/ directory, the kernel DTB in the <Kernel>/arch/arm64/boot/dts/rockchip/ directory, and the resource.img in the <Kernel> directory:

<Kernel>/arch/arm64/boot/Image: Kernel image (specified by the RK_KERNEL_IMG variable in the board-level configuration file).

<Kernel>/arch/arm64/boot/dts/rk3568-atk-evb1-ddr4-v10-linux.dtb: Kernel DTB (determined by the RK_KERNEL_DTS variable in the board-level configuration file!) .

<Kernel>/resource.img: Resource image (a custom image type defined by RK, which packages resources such as logos and dtb into resource.img).

Therefore, when running the script, it is necessary to ensure that these image files exist; otherwise, the execution will fail!

From the above, it can be seen that boot.img is composed of the kernel DTB, the kernel image, and resource.img.

Filename	Explanation
rk3568-atk-evb1-ddr4-v10-linux.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.
Image	Kernel image
resource.img	RK platform resource image, the "resource.img" contains the kernel DTB and logo images. The RK U-Boot will read the kernel DTB from "resource.img" instead of directly packaging this DTB into the "boot.img". This point must be clearly understood!

If the device tree file is modified, it is necessary to recompile the device tree to obtain a new Kernel DTB. Then, it should be packaged into resource.img. After that, resource.img should be packaged into boot.img. Finally, the new boot.img should be burned into the boot partition of the development board to complete the replacement. Although it is quite troublesome, this must be done.

2. resource.img image

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

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

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

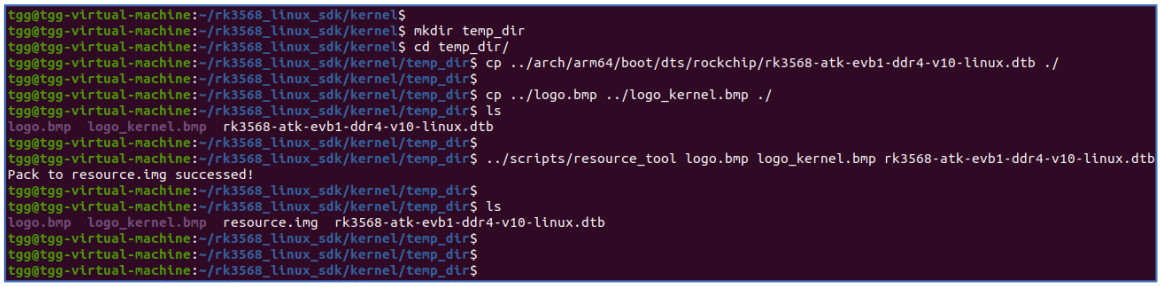


Figure 6.3.2.8 Create resource.img

In the example, we have combined the u-boot logo image logo.bmp, the kernel logo image logo_kernel.bmp, and the kernel DTB into a 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 the resource.img. To do this, execute the following command:

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

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel/temp_dir$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel/temp_dir$ ls
logo.bmp  logo_kernel.bmp  resource.img  rk3568-atk-evb1-ddr4-v10-linux.dtb
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel/temp_dir$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel/temp_dir$ rm -rf logo.bmp logo_kernel.bmp rk3568-atk-evb1-ddr4-v10-linux.dtb
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel/temp_dir$ ls
resource.img
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel/temp_dir$ ../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:141844
entry(1):
        path:logo_kernel.bmp
        offset:282      size:177764
entry(2):
        path:logo.bmp
        offset:630      size:177700
Unack resource.img to out successed!
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel/temp_dir$ ls
out  resource.img
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel/temp_dir$ ls out/
logo.bmp  logo_kernel.bmp  rk-kernel.dtb
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel/temp_dir$
```

Figure 6.3.2.9 Unpacking resource.img

After unpacking, we obtain the u-boot logo image file logo.bmp, the kernel logo image file 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. This is to adapt to the two MIPI screens of the ALIENTEK project (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 based on the MIPI screen connected by the user; if the connected MIPI screen is a 720p one, then load rk3568-atk-evb1-mipi-dsi-720p.dtb; if it is a 1080p MIPI screen, then load rk3568-atk-evb1-mipi-dsi-1080p.dtb.

When creating a resource.img that contains multiple kernel DTBs, the scripts/mkmultidtb.py script can be used to do it, which is more convenient than directly using the scripts/resource_tool tool; the scripts/mkmultidtb.py script still creates resource.img through the scripts/resource_tool tool, but before that, it will regularize the DTB file names, which is required by U-Boot HW-ID DTB, as mentioned in Section 6.1.5 earlier.

Before using the scripts/mkmultidtb.py script to create resource.img, developers need to configure this script in advance. Open this 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 6.3.2.10 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 search 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

The processed DTB name must 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_linux_sdk/kernel$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel$ rm -rf resource.img
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel$
tgg@tgg-virtual-machine:~/rk3568_linux_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_linux_sdk/kernel$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel$ ls -l resource.img
-rw-rw-r-- 1 tgg tgg 458240 4月  12 16:27 resource.img
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel$
```

Figure 6.3.2.11 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_linux_sdk/kernel$ mkdir temp
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel$ cp resource.img ./temp/
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel$ cd temp/
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel/temp$ ls
resource.img
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel/temp$ ../scripts/resource_tool --unpack
Dump header:
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:139535
entry(1):
        path:rk3568-atk-evb1-mipi-dsi-720p#_saradc_ch2=0.dtb
        offset:279      size:139535
entry(2):
        path:rk3568-atk-evb1-mipi-dsi-1080p#_saradc_ch2=341.dtb
        offset:552      size:139531
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_linux_sdk/kernel/temp$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel/temp$ ls
out  resource.img
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel/temp$ 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_linux_sdk/kernel/temp$
```

Figure 6.3.2.12 Unpacking resource.img

After unpacking, you will find three DTB files:

- **rk-kernel.dtb**: A default DTB, which will be used as the default DTB if none of the DTBs match successfully. The mkmultidtb.py script in the packaging script 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 channel 2 is 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 channel 2 is 341+-30.

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

3. Compiling 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 must 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 6.3.2.13 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 6.3.2.14 Generate .ko files

On the Linux system, the kernel modules ultimately need to be copied to the corresponding directories of the root file system, such as /system/lib/modules, /usr/lib/modules, etc. For the 8852bs.ko (the driver module for the WiFi RTL8852 on the ATK-DLRK3568 development board), during the compilation of the buildroot root file system, it will automatically be copied to the root file system's /system/lib/modules directory, and finally packaged to generate the root file system image (rootfs.img).

When compiling the buildroot root file system, if the user wants to package their own .ko module into the rootfs.img image, they need to add the corresponding configuration (Config.in, xxx.mk) in buildroot. (During compilation) Let buildroot copy the user-specified .ko module to the root file system and package it together to generate rootfs.img.

6.3.2 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_linux_defconfig        // Load configuration file
make ARCH=arm64 menuconfig                      // Open the graphical configuration interface
```

The running results are as follows:

```
                    Linux/arm64 4.19.232 Kernel Configuration
Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).  Highlighted letters are hotkeys.  Pressing <Y>
includes, <N> excludes, <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ]
excluded  <M> module  < > module capable

                    *** Compiler: aarch64-linux-gnu-gcc (Linaro GCC 6.3-2017.05) 6.3.1 20170404 ***
                        General setup  --->
                    [*] Support DMA32 zone
                        Platform selection  --->
                        Bus support  --->
                        Kernel Features  --->
                        Boot options  --->
                    [*] Kernel support for 32-bit EL0
                    [*]   Enable kuser helpers page for 32 bit applications.
                        Power management options  --->
                        CPU Power Management  --->
                        Firmware Drivers  --->
                    [ ] Virtualization  ----
                    [*] ARM64 Accelerated Cryptographic Algorithms  --->
                        General architecture-dependent options  --->
                    [*] Enable loadable module support  --->
                    [*] Enable the block layer  --->
                    [ ] Hidden DRM configs needed for GKI
                    [ ] Hidden Regmap configs needed for GKI
                    [ ] Hidden CRYPTO configs needed for GKI
                    [ ] Hidden SND configs needed for GKI
                    [ ] Hidden SND_SOC configs needed for GKI
                    [ ] Hidden GPIO configs needed for GKI
                    [ ] Hidden Virtual configs needed for GKI
                    [ ] Hidden wireless extension configs needed for GKI
                    [ ] Hidden SOC Power Management configs for GKI
                    [ ] Hidden v4l2 configs for GKI
                    [ ] Hiddel USB configurations needed for GKI
                    [ ] Hidden SoC bus configuration needed for GKI
                    [ ] Hidden GPU configuration needed for GKI
                    [ ] Hidden IRQ configuration needed for GKI
                    [ ] GKI Dummy config options
                    v(+)

                          <Select>    < Exit >    < Help >    < Save >    < Load >
```

Figure 6.3.2.1 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_linux_defconfig
```

After saving the configuration, next we can 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 kernel         // Execute the build.sh script to compile the kernel
```

In addition, you can also compile the kernel source code by executing the make.sh script (execute it in the kernel source code directory):

```
make clean                // First, carry out the cleaning work.
./make.sh                 // Execute the script for compilation
../device/rockchip/common/mk-fitimage.sh kernel/boot.img device/rockchip/rk356x/boot.its
```

The last command must also be executed. The reason has been explained earlier!

6.3.3 Kernel Device Tree and defconfig Configuration File

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

All the device tree files of the Rockchip platform are stored in the **<Kernel>arch/arm64/boot/dts/rockchip/** directory. For the ATK-DLRK3568 development board, the used device tree file is: **rk3568-atk-evb1-ddr4-v10-linux.dts**.

rk3568-atk-evb1-ddr4-v10-linux.dts contains multiple.dtsi device trees, as follows:

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

- rk3568.dtsi: This device tree file is for the RK3568 platform-level device tree. 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: The RK3568 board-level universal device tree file, usually included in the board-level device tree files;
- rk3568-linux.dtsi: This device tree contains the configuration information specific to the Linux part (distinguished from Android);
- rk3568-atk-evb1-ddr4-v10.dtsi: Board-level device tree file for the ATK-DLRK3568 development board. This device tree file includes the board-level common device tree file rk3568-evb.dtsi and the platform-level device tree file rk3568.dtsi for the RK3568 platform;
- 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 to enable the corresponding screen, for example, the ATK_LCD_TYPE_HDMI macro indicates enabling HDMI display, and not defining this macro means 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 peripherals associated with the screen based on the user-defined ATK_LCD_TYPE_XXX macro. For example, the touch screen (only enabled when MIPI or LVDS screen is enabled), HDMI audio (only enabled when HDMI is enabled), and backlight (only enabled when MIPI or LVDS is enabled).

The rk3568-atk-evb1-ddr4-v10-linux.dts device tree is designed for user use. When compiling the kernel source code, users only need to include this device tree in the compilation process!

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

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

The rk3568-atk-evb1-ddr4-v10-linux.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. 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 6.3.3.1 rk3568-screen_choose.dtsi device tree file

By default, only MIPI screen and HDMI are enabled.

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

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

The first step is to execute the defconfig configuration:

make ARCH=arm64 rockchip_linux_defconfig

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-linux.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-linux.dtb).

Execute the following command to package the Kernel DTB and the logo image into resource.img:

```
cp arch/arm64/boot/dts/rockchip/rk3568-atk-evb1-ddr4-v10-linux.dtb ./
./scripts/resource_tool logo.bmp logo_kernel.bmp rk3568-atk-evb1-ddr4-v10-linux.dtb
rm -rf ./rk3568-atk-evb1-ddr4-v10-linux.dtb
```

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel$ cp arch/arm64/boot/dts/rockchip/rk3568-atk-evb1-ddr4-v10-linux.dtb ./
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel$ ./scripts/resource_tool logo.bmp logo_kernel.bmp rk3568-atk-evb1-ddr4-v10-linux.dtb
Pack to resource.img successed!
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel$ ls -l resource.img
-rw-rw-r-- 1 tgg tgg 500736 5月  19 18:07 resource.img
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel$ rm -rf ./rk3568-atk-evb1-ddr4-v10-linux.dtb
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel$
```

Figure 6.3.3.2 Package resource.img

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

```
../device/rockchip/common/mk-fitimage.sh kernel/boot.img device/rockchip/rk356x/boot.its
```

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel$ ../device/rockchip/common/mk-fitimage.sh kernel/boot.img device/rockchip/rk356x/boot.its
fdt {
kernel {
resource {
FIT description: U-Boot FIT source file for arm
Created:         Fri May 19 18:12:05 2023
 Image 0 (fdt)
  Description:  unavailable
  Created:      Fri May 19 18:12:05 2023
  Type:         Flat Device Tree
  Compression:  uncompressed
  Data Size:    141844 Bytes = 138.52 KiB = 0.14 MiB
  Architecture: AArch64
  Load Address: 0xffffff00
  Hash algo:    sha256
  Hash value:   c5d23103e3f014f3dc90c6da2f2b0f8d5f4e1b9aed0e22eda3c269bee09f1ce3
 Image 1 (kernel)
  Description:  unavailable
  Created:      Fri May 19 18:12:05 2023
  Type:         Kernel Image
  Compression:  uncompressed
  Data Size:    22603784 Bytes = 22074.01 KiB = 21.56 MiB
  Architecture: AArch64
  OS:           Linux
  Load Address: 0xffffff01
  Entry Point:  0xffffff01
  Hash algo:    sha256
  Hash value:   1232f076260ad21e6a6227a797c8ae0beffeb616cdceb78659a03f1aa85b3edb
 Image 2 (resource)
  Description:  unavailable
  Created:      Fri May 19 18:12:05 2023
  Type:         Multi-File Image
  Compression:  uncompressed
  Data Size:    500736 Bytes = 489.00 KiB = 0.48 MiB
  Hash algo:    sha256
  Hash value:   5ec580be88129e5a219cf00e3cc68898e9963dd5db1ab720c5b4f527947db472
 Default Configuration: 'conf'
 Configuration 0 (conf)
  Description:  unavailable
  Kernel:       kernel
  FDT:          fdt
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel$ ls -l boot.img
-rw-rw-r-- 1 tgg tgg 23250432 5月  19 18:12 boot.img
```

Figure 6.3.3.3 Generate boot.img

The <Kernel>/boot.img can be directly generated by running the mk-fitimage.sh script.

2. Modify make.sh script

After obtaining the SDK, users need to modify the make.sh script in the root directory of the kernel source code; open the <kernel>/make.sh script file and make the modifications. The modified content is as follows:

```
#!/bin/bash

# Default
RK_ARCH=arm64
RK_DEFCONFIG=rockchip_linux_defconfig
RK_DEFCONFIG_FRAGMENT=
RK_DTS=rk3568-atk-evb1-ddr4-v10-linux
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=$arg
            fi
            shift 1
            ;;
        defconfig_fragment=*)
            RK_DEFCONFIG_FRAGMENT=${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 ARCH=$RK_ARCH $RK_DEFCONFIG $RK_DEFCONFIG_FRAGMENT
    # Compile
    make ARCH=$RK_ARCH $RK_DTS.img -j$RK_JOBS
```

By default, the generated resource.img image compiled using the make.sh script does not contain rk3568-atk-evb1-ddr4-v10-linux.dtb, which results in the rk3568-atk-evb1-ddr4-v10-linux.dts device tree configuration not being effective. Therefore, it needs to be modified.

6.3.4 IO Power Domain

This section content is referenced from the document:
<SDK>/docs/Common/IO-DOMAIN/Rockchip_Developer_Guide_Linux_IO_DOMAIN_CN.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 power supply is 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.

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:

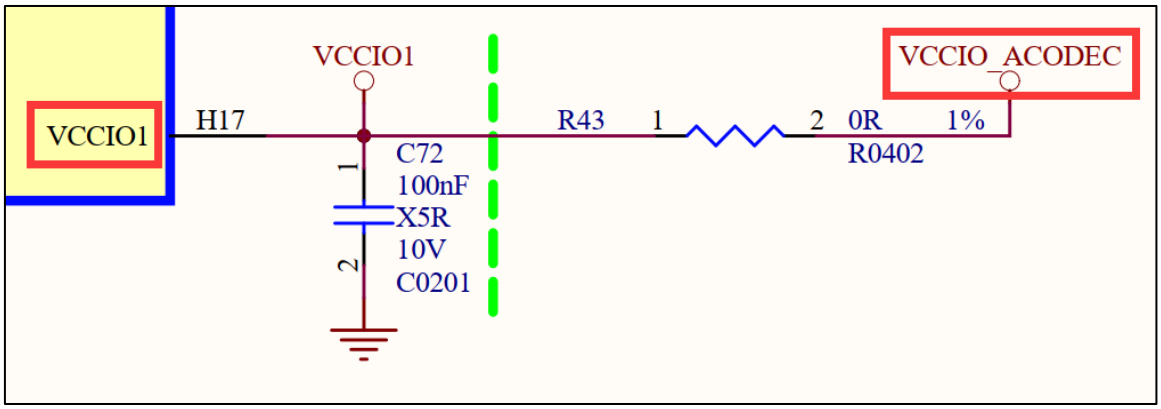


Figure 6.3.4.1 VCCIO1 power domain

As can be seen from the schematic, the power supply for VCCIO1 is VCCIO_ACODEC. Searching for "**VCCIO_ACODEC**" in the core board schematic:

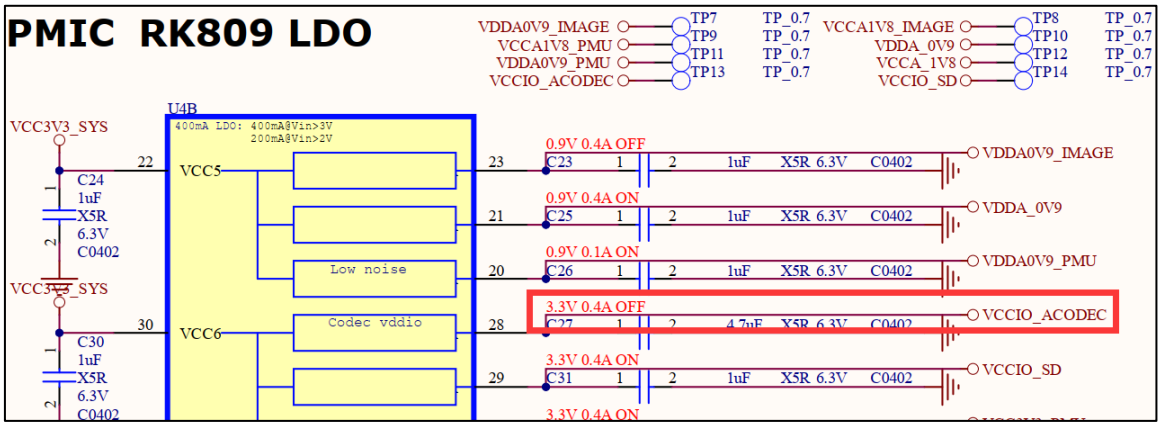


Figure 6.3.4.2 VCCIO_ACODEC power supply

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 6.3.4.3 vccio_acodec node

In the device tree, the VCCIO_ACODEC power supply is configured as 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.

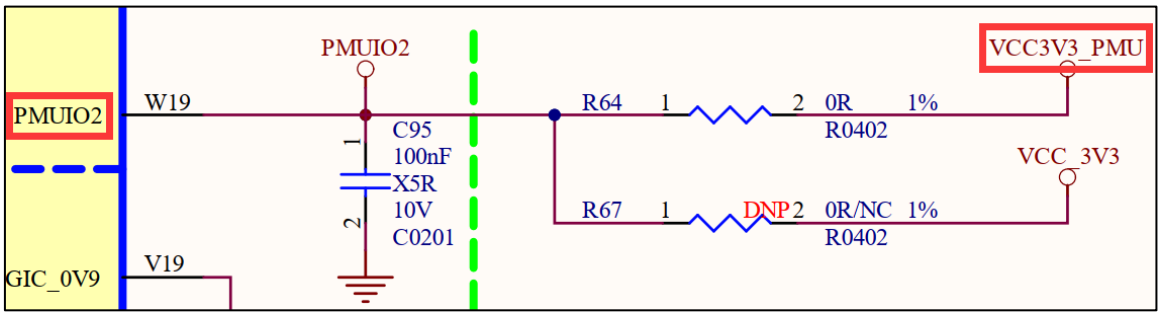


Figure 6.3.4.4 PMUIO2 power domain

As can be seen from the schematic, the power supply for PMUIO2 is VCC3V3_PMU. Search for "**VCC3V3_PMU**" in the schematic.

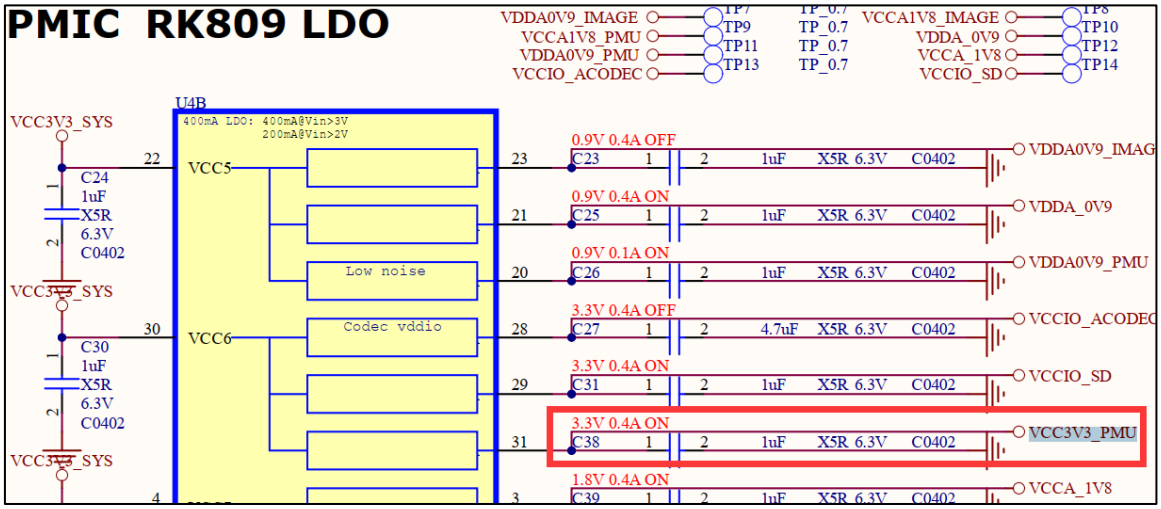


Figure 6.3.4.5 VCC3V3_PMU power supply

Therefore, it can be concluded that VCC3V3_PMU is powered by 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 6.3.4.6 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 no further explanation is needed 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 high; 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 low.** Search for "**VCCIO2**" in the schematic diagram.

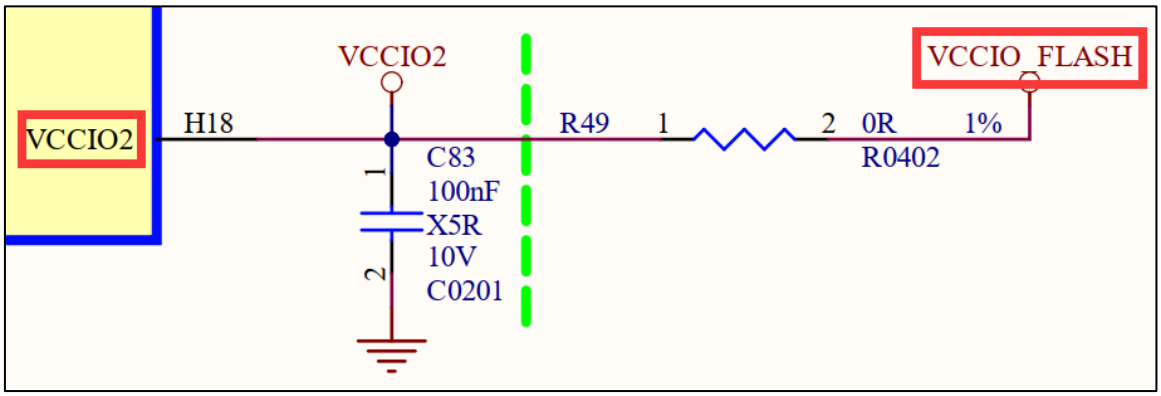


Figure 6.3.4.7 VCCIO2 power domain

As can be seen from the schematic, the power supply for VCCIO2 is VCCIO_FLASH. Search for "**VCCIO_FLASH**" in the schematic:

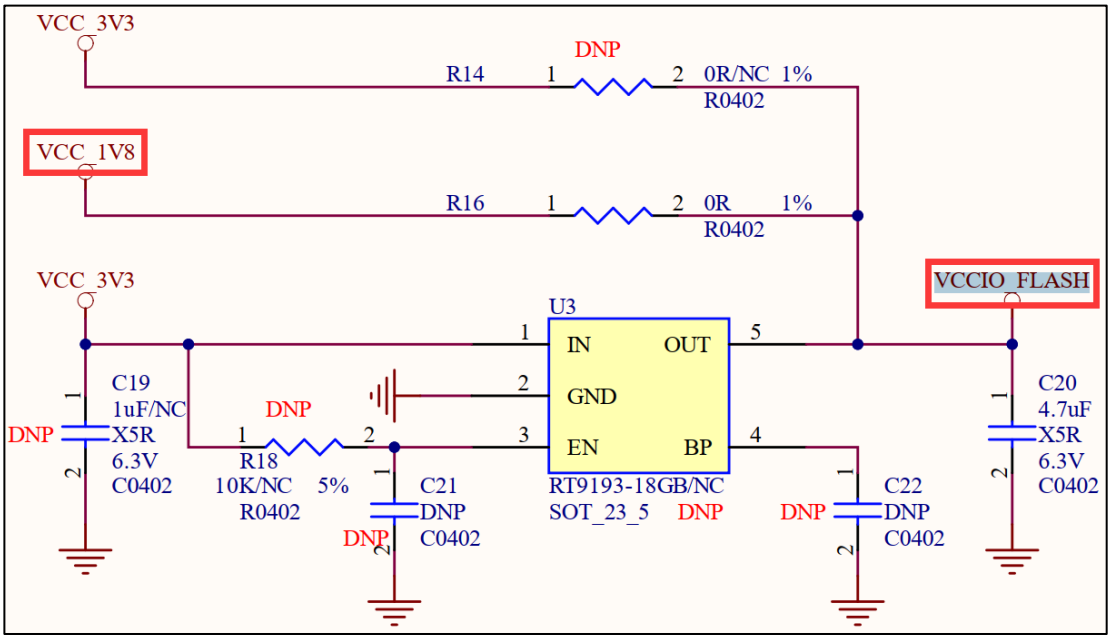


Figure 6.3.4.8 VCCIO_FLASH power supply

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:

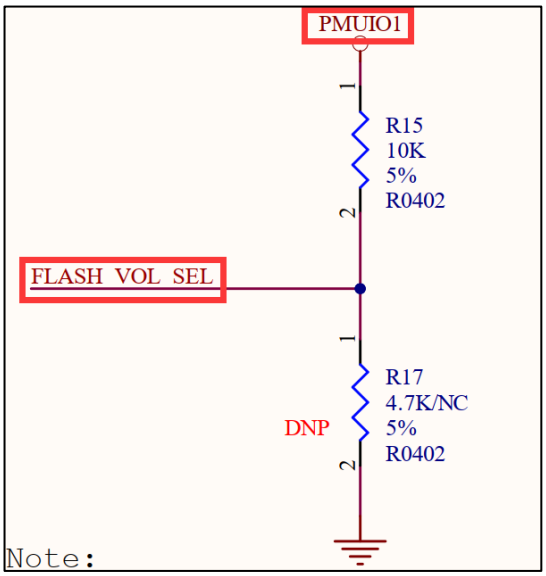


Figure 6.3.4.9 The voltage of the FLASH_VOL_SEL pin

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 6.3.4.10 The definition of the pmu_io_domains node

We cannot directly modify the rk3568.dtsi file. Instead, we modify it by referencing it (**&pmu_io_domains**). By default, RK configures it in the rk3568-evb.dtsi device tree. Open the rk3568-evb.dtsi file and find the following location:

```
&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 6.3.4.11 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 6.3.4.12 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 during kernel compilation

During the compilation of the RK kernel source code, an IO-Domains confirmation dialog box will pop up, as shown in the following figure:

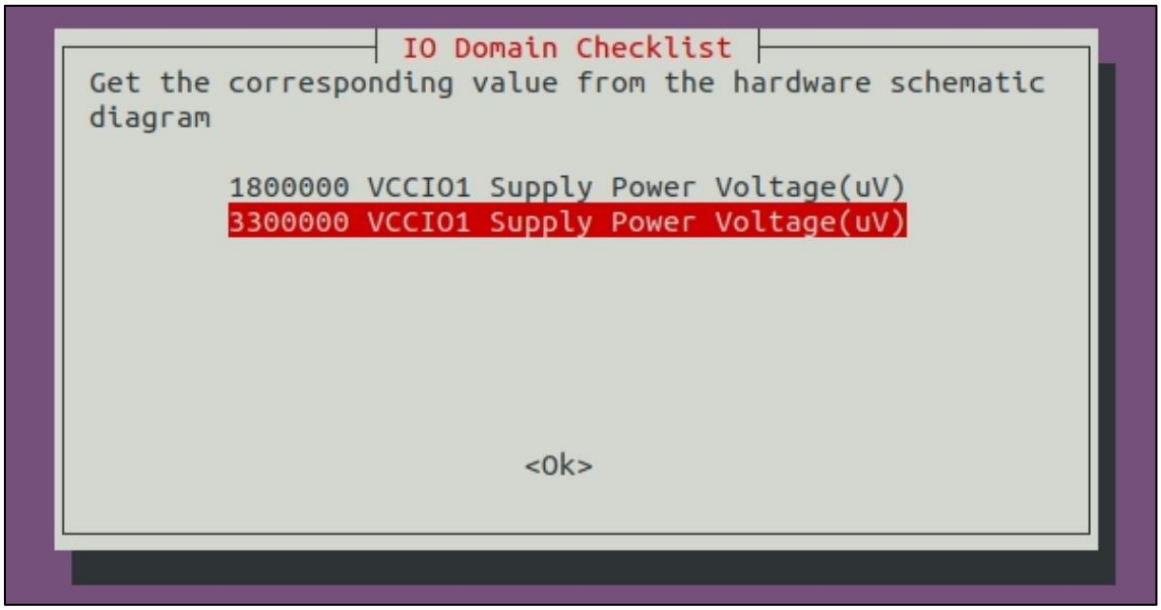


Figure 6.3.4.13 IO-Domains Confirmation Dialog Box

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 (it will only compile successfully if the selected voltage value is the same as the configuration of the pmu_io_domains node in the device tree; otherwise, it will fail to compile!). 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 configuration of the IO power domain after compiling the kernel

After the kernel is successfully compiled, a file starting with "." (i.e., 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_linux_sdk/kernel/arch/arm64/boot/dts/rockchip$ ls -l .*.tmp.domain
-rw-rw-r-- 1 tgg tgg 259 4月   6 17:49 .rk3568-atk-evb1-ddr4-v10-linux.dtb.dts.tmp.domain
-rw-rw-r-- 1 tgg tgg 259 4月  12 08:46 .rk3568-atk-evb1-mipi-dsi-720p.dtb.dts.tmp.domain
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel/arch/arm64/boot/dts/rockchip$
```

Figure 6.3.4.14 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_linux_sdk/kernel/arch/arm64/boot/dts/rockchip$ cat .rk3568-atk-evb1-ddr4-v10-linux.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_linux_sdk/kernel/arch/arm64/boot/dts/rockchip$
```

Figure 6.3.4.15 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.

6.3.5 Replace the logo

Including the logo at the u-boot stage and the logo at the kernel stage. These two logo images are all located in the kernel source code directory:

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel$ pwd
/home/tgg/rk3568_linux_sdk/kernel
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel$ ls -l *.bmp
-rw-rw-r-- 1 tgg tgg 177700 4月  13 09:23 logo.bmp
-rw-rw-r-- 1 tgg tgg 177764 4月  13 09:23 logo_kernel.bmp
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel$
```

Figure 6.3.5.1 Logo image file

The U-Boot logo image is named "logo.bmp". The default U-Boot logo is as follows:

Figure 6.3.5.2 U-Boot logo

The kernel logo image is named "logo_kernel.bmp". The default kernel logo is as follows:

Figure 6.3.5.3 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 root directory of the kernel source code 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.

6.3.6 Summary of Kernel Module Development Documents

The **<SDK>/docs/Common** directory contains many kernel module development documents provided by RK. These documents are divided and placed in different directories according to functional modules to facilitate users' 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
Camera	CAMERA	The document is stored in the ISP2X and ISP3X subdirectories.
CAN	CAN	Rockchip_Develop_Guide_Can_CN.pdf Rockchip_Develop_Guide_CAN_FD_CN.pdf
Clock configuration	CLK	Rockchip_Developer_Guide_Linux4.4_4.19_Clock_CN.pdf Rockchip_Develop_Guide_Gpio_Output_Clocks_CN.pdf Rockchip_Develop_Guide_Pll_Ssmod_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_Trouble_Shooting_DDR_CN.pdf ……
Common debugging tools such as JTAG/GDB	DEBUG	Rockchip_Developer_Guide_DS5_CN.pdf Rockchip_Developer_Guide_GDB_Over_ADB_CN.pdf Rockchip_Developer_Guide_OpenOCD_CN.pdf Rockchip_User_Guide_J-Link_CN.pdf
Display	DISPLAY	Rockchip_Developer_Guide_DRM_Display_Driver_CN.pdf Rockchip_Developer_Guide_HDMI-CEC_CN.pdf Rockchip_Developer_Guide_HDMI_CN.pdf Rockchip_Developer_Guide_HDMI-PHY-PLL_Config_CN.pdf Rockchip_DRM_Display_Driver_Development_Guide_V1.0.pdf Rockchip_DRM_Panel_Porting_Guide_V1.6_20190228.pdf ……
Frequency and voltage adjustment of CPU/GPU, etc.	DVFS	Rockchip_Developer_Guide_CPUFreq_CN.pdf Rockchip_Developer_Guide_Devfreq_CN.pdf
Ethernet configuration	GMAC	Rockchip_Developer_Guide_Linux_GMAC_CN.pdf Rockchip_Developer_Guide_Linux_GMAC_RGMII_Delayline_CN.pdf Rockchip_Developer_Guide_Linux_GMAC_Mode_Configuration_CN.pdf Rockchip_Developer_Guide_Linux_MAC_TO_MAC_CN.pdf
I2C communication	I2C	Rockchip_Developer_Guide_I2C_CN.pdf
GPIO power domain	IO-DOMAIN	Rockchip_Developer_Guide_Linux_IO_DOMAIN_CN.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
Storage	NVM	Rockchip_Application_Notes_Storage_CN.pdf Rockchip_Developer_Guide_SATA_CN.pdf Rockchip_Developer_Guide_OTP_CN.pdf

		Rockchip_Developer_FAQ_Storage_CN.pdf Rockchip_Developer_Guide_Secure_Boot_for_UBoot_Next_Dev_CN.pdf
PCIe	PCIe	Rockchip_Developer_Guide_PCIe_CN.pdf Rockchip_Developer_Guide_PCIe_Performance_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_Pinctrl_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_CN.pdf
PWM	PWM	Rockchip_Developer_Guide_Linux_PWM_CN.pdf
ADC	SARADC	Rockchip_Developer_Guide_Linux_SARADC_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_RK3308_Developer_Guide_System_Suspend_CN.pdf Rockchip_RK3399_Developer_Guide_System_Suspend_CN.pdf
Serial port	UART	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 details should be based on the **<SDK>/docs/Common** directory!

6.3.7 Three-screen Display

Reference Document: Development Board CD-ROM A Disk - **Basic Materials -> 10_user_manual -> 03. Supplementary Documents -> [ALIENTEK] ATK-DLRK3568 Development Board Three-Screen Display Reference Manual.pdf**.

6.4 Buildroot Development

The Buildroot source code is located in the <SDK>/buildroot directory:

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$ pwd
/home/tgg/rk3568_linux_sdk/buildroot
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$ ls
arch    build      Config.in.legacy  DEVELOPERS  fs        Makefile.legacy  README   toolchain
board   CHANGES    configs           dl          linux     output           support  utils
boot    Config.in  COPYING           docs        Makefile  package          system
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$
```

Figure 6.4-1 Buildroot project directory

It is recommended that users use Buildroot to build the root file system. Yocto is not recommended!

RK has already configured environment variables, BSP configuration, and development of each module, making it convenient for users to develop and customize based on Buildroot for RK.

The introduction and usage methods of Buildroot are not covered in this document. You can refer to the following documents:

Development Board CD-ROM A Disk - **Basic Materials→10_user Manual →03. Supplementary Documents→ [ALIENTEK] Buildroot User Manual in Chinese Version (Translated by ALIENTEK)_V1.0.pdf**

Development Board CD-ROM A Disk - **Basic Materials →10_user Manual→03. Supplementary Documents→ [ALIENTEK] Buildroot Quick User Manual V1.0.pdf**

During the process of compiling the RK3568 Linux SDK, you need to compile buildroot twice:

- Compile buildroot to obtain the root file system image named rootfs.img. This rootfs.img will be burned onto the rootfs partition of the development board.
- Compile buildroot to obtain the ramdisk root file system image (this ramdisk root file system is mounted when entering the recovery mode, while the rootfs.img is the root file system that is mounted during normal system startup. Make sure to understand this point!) 。 The ramdisk will eventually be packaged into the recovery.img image, and the recovery.img will be burned onto the recovery partition of the development board.

6.4.1 Introduction to the buildroot directory structure

The directory structure of the buildroot source code is shown in the following figure:

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$ ls
arch    build       Config.in.legacy  DEVELOPERS  fs        Makefile.legacy  README   toolchain
board   CHANGES     configs           dl          linux     output           support  utils
boot    Config.in   COPYING           docs        Makefile  package          system
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$
```

Next, let's briefly introduce the folders located in the buildroot root directory:

Top-level directory name	Explanation
arch	Store the configuration files and build scripts related to all CPU architectures supported by Buildroot
board	Store the files specific to the target platform, such as kernel configurations or patch files, rootfs override files, etc.
boot	Store the patches, verification files, build scripts, configuration options, etc. related to BootLoader supported by Buildroot
build	Components related to the Buildroot compilation system
configs	Stores all the defconfig configuration files for the target platforms
dl	The abbreviation of "download", this directory is used to store various open-source software packages downloaded, such as the alsa-lib library, bluez library, bzip2, curl tool, etc.; during the compilation process, buildroot will download the required software packages from the network and place them in the "dl/" directory; if buildroot fails to download a certain software package or cannot download successfully, at this time, we can also manually download the software package and copy it to the "dl" directory. All software packages only need to be downloaded once, not every compilation needs to download it again (unless the "dl" directory is deleted), as long as the software package exists in the "dl" directory, there is no need to download it again; therefore, the first compilation is often slower because the download process will occupy a lot of time.
docs	Store relevant reference documents
fs	Store the source code of various file systems
linux	Store the build scripts and configuration options of Linux
output	This folder will appear after the buildroot compilation. The "output" directory is used to store various files generated during the compilation process, including various intermediate target files, executable files, library files, and the final rootfs image that is burned onto the development board.
package	All the build scripts and configuration options for the packages are stored here; in each package directory (package/<package_name>/), there is a Config.in file and a <package_name>.mk file (which is actually a Makefile file); if a new package needs to be added, the package/ directory needs to be modified.
support	Store some scripts and configuration files that provide functional support for Buildroot
toolchain	Store the build scripts and related files for creating various cross-compiler toolchains, such as binutils, gcc, gdb, kernel-header, and uClibc
utils	Store some practical scripts and tools for Buildroot

6.4.2 Common Compilation Commands

There is a Makefile file in the root directory of the buildroot source code, as shown below:

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$ ls
arch    build      Config.in.legacy  DEVELOPERS  fs        Makefile.legacy  README   toolchain
board   CHANGES    configs           dl          linux     output           support  utils
boot    Config.in  COPYING           docs        Makefile  package          system
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$
```

This is the Makefile file of the Buildroot source code project. Therefore, we can compile Buildroot using the make command, which can either compile the entire root file system or compile a specific package separately.

① **Compiling the root file system**

Before compiling, we need to configure it first; enter the buildroot directory and execute the following command for configuration (taking the rk3568 platform as an example):

```
source build/envsetup.sh rockchip_rk3568
```

The last parameter in the command (rockchip_rk3568) is used to specify the defconfig configuration file for the target platform (without the _defconfig suffix). All the defconfig configuration files for the target platforms are stored in the **<Buildroot>/configs** directory. In this directory, you can find the configuration file for rk3568, which is named **rockchip_rk3568_defconfig**, as shown below:

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$ ls configs/rockchip_rk3568* -l
-rw-rw-r-- 1 tgg tgg  604 4月   6 15:32 configs/rockchip_rk3568_32_defconfig
-rw-rw-r-- 1 tgg tgg  740 4月   6 15:32 configs/rockchip_rk3568_defconfig
-rw-rw-r-- 1 tgg tgg 1206 4月   6 15:32 configs/rockchip_rk3568_uvc_defconfig
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$
```

After the configuration is completed, simply execute the "**make**" or "**make all**" command to compile the root file system.

```
make
```

Or

```
make all
```

Using multi-threaded compilation (-jN):

```
make -j12
```

Or

```
make all -j12
```

② **Compiling the package**

Before compiling, perform the configuration first. Enter the buildroot directory and execute the following command for configuration (taking the rk3568 platform as an example):

```
source build/envsetup.sh rockchip_rk3568
```

After the configuration is completed, execute the following command to compile the specified package:

```
make <package_name>
```

Parameter <package_name> is used to specify the name of the package, for example, rkwifibt:

```
make rkwifibt
```

③ **buildroot clean command**

By executing the following command, all the files generated during the build process (including build, host, staging and target trees, images, and toolchain) can be deleted:

```
make clean
```

Executing the following command will delete all the files generated during the build process as well as the related configurations (a complete clearance, equivalent to deleting the "output" directory, and it will also delete the "dl" folder. Use with caution!!!) ：

```
make distclean
```

Executing the following command will delete the build directory of the specified package:

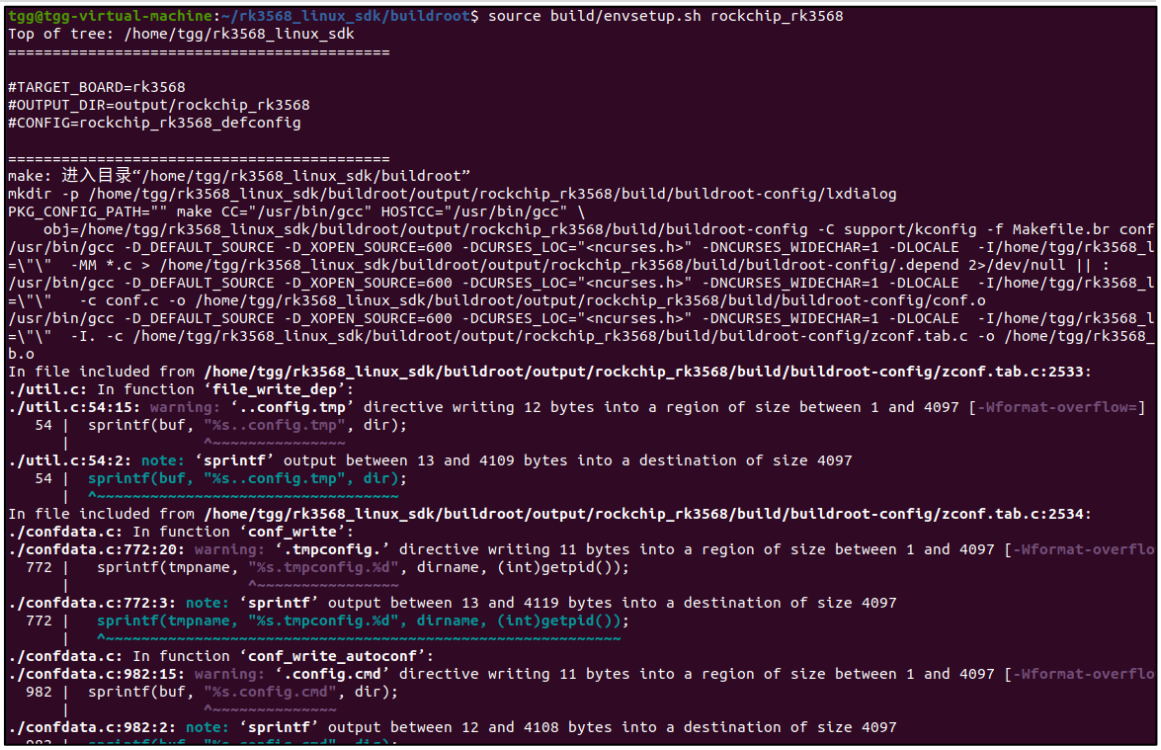
```
make <package_name>-dirclean
```

6.4.3 Compile the root file system image rootfs.img for RK3568

This section introduces the process of compiling the root file system image rootfs.img for the RK3568 platform using Buildroot. Section 6.4 explains how to compile the ramdisk root file system image using Buildroot.

Before compiling, perform the defconfig configuration first. Execute the following command in the buildroot directory:

```
source build/envsetup.sh rockchip_rk3568
```



```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$ source build/envsetup.sh rockchip_rk3568
Top of tree: /home/tgg/rk3568_linux_sdk
========================================

#TARGET_BOARD=rk3568
#OUTPUT_DIR=output/rockchip_rk3568
#CONFIG=rockchip_rk3568_defconfig

========================================
make: 进入目录“/home/tgg/rk3568_linux_sdk/buildroot”
mkdir -p /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/build/buildroot-config/lxdialog
PKG_CONFIG_PATH="" make CC="/usr/bin/gcc" HOSTCC="/usr/bin/gcc" \
    obj=/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/build/buildroot-config -C support/kconfig -f Makefile.br conf
/usr/bin/gcc -D_DEFAULT_SOURCE -D_XOPEN_SOURCE=600 -DCURSES_LOC="<ncurses.h>" -DNCURSES_WIDECHAR=1 -DLOCALE  -I/home/tgg/rk3568_l
=\"\"  -MM *.c > /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/build/buildroot-config/.depend 2>/dev/null || :
/usr/bin/gcc -D_DEFAULT_SOURCE -D_XOPEN_SOURCE=600 -DCURSES_LOC="<ncurses.h>" -DNCURSES_WIDECHAR=1 -DLOCALE  -I/home/tgg/rk3568_l
=\"\"   -c conf.c -o /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/build/buildroot-config/conf.o
/usr/bin/gcc -D_DEFAULT_SOURCE -D_XOPEN_SOURCE=600 -DCURSES_LOC="<ncurses.h>" -DNCURSES_WIDECHAR=1 -DLOCALE  -I/home/tgg/rk3568_l
=\"\"  -I. -c /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/build/buildroot-config/zconf.tab.c -o /home/tgg/rk3568_
b.o
In file included from /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/build/buildroot-config/zconf.tab.c:2533:
./util.c: In function ‘file_write_dep’:
./util.c:54:15: warning: ‘..config.tmp’ directive writing 12 bytes into a region of size between 1 and 4097 [-Wformat-overflow=]
   54 |  sprintf(buf, "%s..config.tmp", dir);
      |               ^~~~~~~~~~~~~~~
./util.c:54:2: note: ‘sprintf’ output between 13 and 4109 bytes into a destination of size 4097
   54 |  sprintf(buf, "%s..config.tmp", dir);
      |  ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
In file included from /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/build/buildroot-config/zconf.tab.c:2534:
./confdata.c: In function ‘conf_write’:
./confdata.c:772:20: warning: ‘.tmpconfig.’ directive writing 11 bytes into a region of size between 1 and 4097 [-Wformat-overflo
  772 |   sprintf(tmpname, "%s.tmpconfig.%d", dirname, (int)getpid());
      |                    ^~~~~~~~~~~~~~~~
./confdata.c:772:3: note: ‘sprintf’ output between 13 and 4119 bytes into a destination of size 4097
  772 |   sprintf(tmpname, "%s.tmpconfig.%d", dirname, (int)getpid());
      |   ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
./confdata.c: In function ‘conf_write_autoconf’:
./confdata.c:982:15: warning: ‘.config.cmd’ directive writing 11 bytes into a region of size between 1 and 4097 [-Wformat-overflo
  982 |  sprintf(buf, "%s.config.cmd", dir);
      |               ^~~~~~~~~~~~~~
./confdata.c:982:2: note: ‘sprintf’ output between 12 and 4108 bytes into a destination of size 4097
```

Figure 6.4.3.1 Execute the defconfig configuration (1)

```
  GEN     /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/Makefile
/home/tgg/rk3568_linux_sdk/buildroot/build/defconfig_hook.py -m /home/tgg/rk3568_linux_sdk/buildroot/configs/rockchip_rk3568_def
pconfig
BR2_DEFCONFIG='' KCONFIG_AUTOCONFIG=/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/build/buildroot-config/auto.conf
3568/build/buildroot-config/autoconf.h KCONFIG_TRISTATE=/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/build/buildr
output/rockchip_rk3568/.config HOST_GCC_VERSION="9" BUILD_DIR=/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/build
kchip_rk3568_defconfig /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/build/buildroot-config/conf --defconfig=/home
ig.in
/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/.rockchipconfig:256:warning: override: reassigning to symbol BR2_PAC
/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/.rockchipconfig:257:warning: override: reassigning to symbol BR2_PAC
/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/.rockchipconfig:259:warning: override: reassigning to symbol BR2_PAC
/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/.rockchipconfig:284:warning: override: reassigning to symbol BR2_PAC
/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/.rockchipconfig:285:warning: override: reassigning to symbol BR2_PAC
/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/.rockchipconfig:286:warning: override: reassigning to symbol BR2_PAC
/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/.rockchipconfig:292:warning: override: reassigning to symbol BR2_PAC
/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/.rockchipconfig:293:warning: override: reassigning to symbol BR2_PAC
/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/.rockchipconfig:294:warning: override: reassigning to symbol BR2_PAC
/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/.rockchipconfig:297:warning: override: reassigning to symbol BR2_PAC
/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/.rockchipconfig:316:warning: override: reassigning to symbol BR2_PAC
/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/.rockchipconfig:319:warning: override: reassigning to symbol BR2_PAC
#
# configuration written to /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/.config
#
make: 离开目录"/home/tgg/rk3568_linux_sdk/buildroot"
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$
```

Figure 6.4.3.2 Execute the defconfig configuration (2)

The last parameter in the command (rockchip_rk3568) is used to specify the defconfig configuration file for the target platform (without the _defconfig suffix). All the defconfig configuration files for the target platforms are stored in the **<Buildroot>/configs** directory. In this directory, you can find the configuration file for rk3568, which is named **rockchip_rk3568_defconfig**, as shown below:

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$ ls configs/rockchip_rk3568* -l
-rw-rw-r-- 1 tgg tgg  604 4月   6 15:32 configs/rockchip_rk3568_32_defconfig
-rw-rw-r-- 1 tgg tgg  740 4月   6 15:32 configs/rockchip_rk3568_defconfig
-rw-rw-r-- 1 tgg tgg 1206 4月   6 15:32 configs/rockchip_rk3568_uvc_defconfig
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$
```

Figure 6.4.3.3 rockchip_rk3568_defconfig configuration file

Then execute the "make" command or the "make all" command to compile the root file system:

```
make -j16
```

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$ make -j16
/usr/bin/make -j1 O=/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568 HOSTCC="/usr/bin/gcc" HOSTCXX="/usr/bin/g++" silentoldconfig
make[2]: warning: -j1 forced in submake: resetting jobserver mode.
  GEN     /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/Makefile
>>> host-tar 1.29 Extracting
mkdir -p /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/build/host-tar-1.29
cd /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/build/host-tar-1.29 && gzip -d -c /home/tgg/rk3568_linux_sdk/buildroot/dl/tar
31631 块
mv /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/build/host-tar-1.29/tar-1.29/* /home/tgg/rk3568_linux_sdk/buildroot/output/ro
rmdir /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/build/host-tar-1.29/tar-1.29
>>> host-tar 1.29 Patching
>>> host-tar 1.29 Updating config.sub and config.guess
for file in config.guess config.sub; do for i in $(find /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/build/host-tar-1.29 -nam
>>> host-tar 1.29 Patching libtool
>>> host-tar 1.29 Configuring
(cd /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/build/host-tar-1.29/ && rm -rf config.cache; PATH="/home/tgg/rk3568_linux_sd
nux_sdk/buildroot/output/rockchip_rk3568/host/sbin:/home/tgg/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/us
FIG="/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/host/bin/pkg-config" PKG_CONFIG_SYSROOT_DIR="/" PKG_CONFIG_ALLOW_SYSTEM_CFL
gg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/host/lib/pkgconfig:/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/host/sha
" NM="/usr/bin/nm" CC="/usr/bin/gcc" GCC="/usr/bin/gcc" CXX="/usr/bin/g++" CPP="/usr/bin/cpp" OBJCOPY="/usr/bin/objcopy" RANLIB="/usr/bin/ra
ockchip_rk3568/host/include" CFLAGS="-O2 -I/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/host/include" CXXFLAGS="-O2 -I/home/t
de" LDFLAGS="-L/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/host/lib -Wl,-rpath,/home/tgg/rk3568_linux_sdk/buildroot/output/r
2 -I/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/host/include" LDFLAGS="-L/home/tgg/rk3568_linux_sdk/buildroot/output/rockchi
ot/output/rockchip_rk3568/host/lib"  CONFIG_SITE=/dev/null ./configure --prefix="/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568
t/rockchip_rk3568/host/etc" --localstatedir="/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/host/var" --enable-shared --disable
--disable-docs --disable-documentation --disable-debug --with-xmlto=no --with-fop=no --disable-dependency-tracking  --without-selinux )
configure: WARNING: unrecognized options: --enable-shared, --disable-static, --disable-gtk-doc, --disable-gtk-doc-html, --disable-doc, --dis
to, --with-fop
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for a thread-safe mkdir -p... /usr/bin/mkdir -p
checking for gawk... gawk
checking whether make sets $(MAKE)... yes
checking whether make supports nested variables... yes
checking whether UID '1000' is supported by ustar format... yes
checking whether GID '1000' is supported by ustar format... yes
checking how to create a ustar tar archive... gnutar
checking whether make supports nested variables... (cached) yes
```

Figure 6.4.3.4 Compile the root file system

The entire compilation process will take a very long time and requires patience to wait!

After the compilation is completed, as shown in the following figure:

```
Number of uids 3
        root (0)
        tgg (1000)
        www-data (33)
Number of gids 3
        root (0)
        tgg (1000)
        www-data (33)
/usr/bin/install -m 0644 support/misc/target-dir-warning.txt /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/target/
>>>   Generating root filesystem image rootfs.tar
rm -rf /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/build/buildroot-fs
mkdir -p /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/build/buildroot-fs
echo '#!/bin/sh' > /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/build/buildroot-fs/fakeroot.fs
echo "set -e" >> /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/build/buildroot-fs/fakeroot.fs
echo "chown -h -R 0:0 /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/target" >> /home/tgg/rk3568_linux_sdk/buildroo
printf '        dbus -1 dbus -1 * /var/run/dbus - dbus DBus messagebus user\n   - - input -1 * - - - Input device group\n\n' >>
-fs/users_table.txt
PATH="/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/host/bin:/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_
r/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:/home/tgg/tools/platform-tools" /home/tgg/rk3568_linux_sdk/buildroot/supp
3568/build/buildroot-fs/users_table.txt /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/target >> /home/tgg/rk3568_l
cat system/device_table.txt > /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/build/buildroot-fs/device_table.txt
printf '        /bin/busybox                    f 4755 0  0 - - - - -\n        /usr/libexec/dbus-daemon-launch-helper f 4755 0
lockdiff    f 4755 0 0 - - - - -\n        /bin/ping             f 4755 0 0 - - - - -\n    /usr/bin/traceroute6  f 4755 0 0 - - -
 1 - - -\n\n' >> /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/build/buildroot-fs/device_table.txt
echo "/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/host/bin/makedevs -d /home/tgg/rk3568_linux_sdk/buildroot/outp
ux_sdk/buildroot/output/rockchip_rk3568/target" >> /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/build/buildroot-f
printf '        (cd /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/target; find -print0 | LC_ALL=C sort -z | tar
s.tar --null --no-recursion -T - --numeric-owner)\n' >> /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/build/buildr
chmod a+x /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/build/buildroot-fs/fakeroot.fs
rm -f /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/target/THIS_IS_NOT_YOUR_ROOT_FILESYSTEM
PATH="/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/host/bin:/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_
r/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:/home/tgg/tools/platform-tools" /home/tgg/rk3568_linux_sdk/buildroot/outp
t/output/rockchip_rk3568/build/buildroot-fs/fakeroot.fs
rootdir=/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/target
table='/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/build/buildroot-fs/device_table.txt'
/usr/bin/install -m 0644 support/misc/target-dir-warning.txt /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/target/
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$
```

Figure 6.4.3.5 The root file system has been compiled successfully.

The compiled root file system image is saved in the "output/rockchip_rk3568/images" directory.

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$ cd output/rockchip_rk3568/images/
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/images$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/images$ ls
rootfs.cpio  rootfs.cpio.gz  rootfs.ext2  rootfs.ext4  rootfs.squashfs  rootfs.tar
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/images$
```

Figure 6.4.3.6 Rootfs image

In this directory, multiple different formats of root file system images have been generated. On the RK3568 platform, the one used is **rootfs.ext4** (ext4 format, determined by the **RK_ROOTFS_TYPE** variable in the SDK board-level configuration file). rootfs.ext4 is a symbolic link file, and the actual image it points to is rootfs.ext2. Use the file command to view rootfs.ext2, as shown below:

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/images$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/images$ ls
rootfs.cpio  rootfs.cpio.gz  rootfs.ext2  rootfs.ext4  rootfs.squashfs  rootfs.tar
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/images$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/images$ file rootfs.ext2
rootfs.ext2: Linux rev 1.0 ext4 filesystem data, UUID=0bb88009-18b9-4f5a-ab25-4ac943ca2fbf (extents) (large files) (huge files)
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/images$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/images$
```

Figure 6.4.3.7 The "file" command is used to view the contents of "rootfs.ext2".

It can be known that rootfs.ext2 is the image of the ext4 format root file system. Renaming it to **rootfs.img**, the rootfs.img file will eventually be burned onto the rootfs partition of the development board, serving as the root file system that is mounted when the system is started normally.

Use the "make" command for compilation. During the compilation process, a large amount of output information will be generated, which looks very messy and is quite annoying! If you want to filter out these numerous useless output messages during the compilation process, you can use the following command for compilation:

```
utils/brmake -j16
```

"brmake" is a shell script. By using this script for compilation, it can filter out most of the insignificant output information and retain the key information, making the output content on the terminal more concise.

6.4.4 Configuration of rootfs

In the source code directory of Buildroot, execute the following command to configure the Buildroot root file system:

```
source build/envsetup.sh rockchip_rk3568        // Load the buildroot configuration file
make menuconfig
// Open the menuconfig graphical configuration interface
```

The running results are as follows:

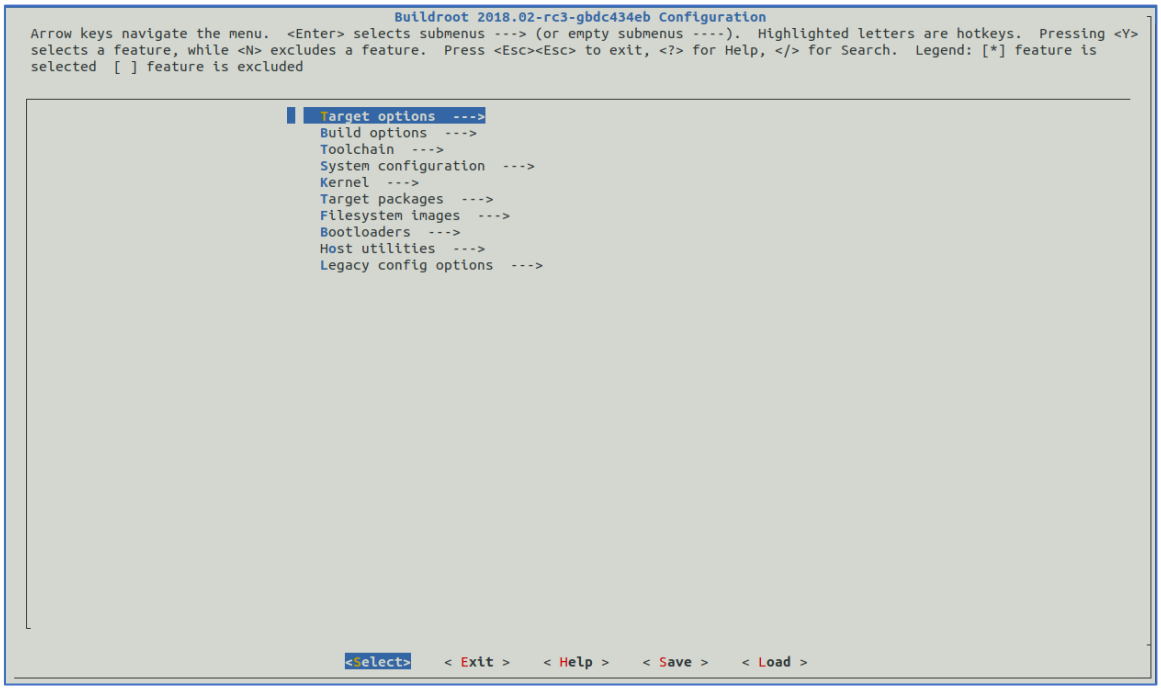


Figure 6.4.4.1 The menuconfig graphical configuration interface

Through the menuconfig graphical interface, the buildroot can be configured. After the configuration is completed, the following command can be executed to save the settings:

```
make savedefconfig
```

Next, we can return to the SDK root directory and execute the build.sh script to compile the buildroot (or we can follow the steps in Section 6.3.3 for the compilation process):

```
cd ../                    // Return to the root directory of the SDK (currently located in the buildroot source code directory)
./build.sh buildroot      // Execute the build.sh script to compile buildroot
```

6.4.5 Output Directory Introduction

The output directory is used to store various files generated during the compilation process, including intermediate target files, executable files, lib libraries, and the final rootfs image burned onto the development board, etc.

The structure of the output directory is as follows:

```
output/
├── rockchip_rk3568
│   ├── build        # It includes all the built software packages, including the tools required by buildroot on the host machine as well as the software packages compiled for the target platform.
│   ├── host         # It includes the tools for building the Ubuntu host (the host machine) as well as the target toolchain.
│   ├── images       # The image for storing the final compiled output
│   ├── Makefile
│   ├── staging      # A symbolic link pointing to the target toolchain sysroot in the "host/" directory
│   └── target       # The root file system system directory is used to create the root file system image. The content of the rootfs.img image is exactly the contents of this directory.
```

6.4.6 Package compilation

This section will explain how to compile the specified software package.

First, perform the configuration before compilation; no further explanation is needed here. Then, execute the following command to compile the specified package:

```
make <package_name>
```

Or

```
utils/brmake <package_name>
```

The parameter "package_name" is used to specify the name of the package, for example, "rkwifibt:".

```
make rkwifibt
```

Or

```
utils/brmake rkwifibt
```

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$ utils/brmake rkwifibt
2023-07-13T14:47:16 >>> rkwifibt 1.0.0 Syncing from source dir /home/tgg/rk3568_linux_sdk/buildroot/../external/rkwifibt
2023-07-13T14:47:16 >>> rkwifibt 1.0.0 Configuring
2023-07-13T14:47:17 >>> rkwifibt 1.0.0 Building
2023-07-13T14:47:25 >>> rkwifibt 1.0.0 Installing to target
Done in 12s
```

The key issue lies in how to determine/get the parameter "package_name"?

In fact, the parameter "package_name" corresponds to the name of a certain directory within the buildroot/package/folder (or the name of a certain subdirectory), and this directory contains the Config.in configuration file (used to define package configuration options) and the <package_name>.mk file (Makefile file, used to define the build logic of the package). For example:

```
rkwifibt  →  package/rockchip/rkwifibt/
rknpu     →  package/rockchip/rknpu/
rkupdate  →  package/rockchip/rkupdate/
```

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$ ls package/rockchip/rkwifibt/
Config.in  rkwifibt.hash  rkwifibt.mk
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$ ls package/rockchip/rknpu/
Config.in  rknpu.mk
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$ ls package/rockchip/rkupdate/
Config.in  rkupdate.mk
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$
```

The parameter "package_name" refers to the name of the directory where the "Config.in" and "<package_name>.mk" files are located.

The construction process of a package can be broken down into: configure, build, and install. The execution sequence is: **configuration → compilation → installation**. Each step can be executed independently. The following commands can be used for execution:

```
make <package_name>-configure      # Execute configuration commands
make <package_name>-build          # Execute the compilation command
make <package_name>-install        # Execute the installation command
```

Or

```
utils/brmake <package_name>-configure      # Execute configuration commands
utils/brmake <package_name>-build          # Execute the compilation command
utils/brmake <package_name>-install        # Execute the installation command
```

Take rkwifibt as an example:

```
utils/brmake rkwifibt-configure
utils/brmake rkwifibt-build
utils/brmake rkwifibt-install
```

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$ utils/brmake rkwifibt-configure
2023-07-13T15:53:43 >>> rkwifibt 1.0.0 Syncing from source dir /home/tgg/rk3568_linux_sdk/buildroot/../external/rkwifibt
2023-07-13T15:53:43 >>> rkwifibt 1.0.0 Configuring
Done in 3s
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$ utils/brmake rkwifibt-build
2023-07-13T15:53:49 >>> rkwifibt 1.0.0 Building
Done in 11s
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$ utils/brmake rkwifibt-install
2023-07-13T15:54:02 >>> rkwifibt 1.0.0 Installing to target
Done in 3s
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$
```

Corresponding to this are: reconfigure (reconfigure), rebuild (recompile), reinstall (reinstall).

```
make <package_name>-reconfigure    # Starting from the configuration stage, restart the package's build process
make <package_name>-rebuild        # Starting from the compilation stage, restart the building process of the package
make <package_name>-reinstall      # Starting from the installation stage, restart the package's build process
```

Or

```
utils/brmake <package_name>-reconfigure  # Starting from the configuration stage, restart the package's build process
```

utils/brmake <package_name>-rebuild # Starting from the compilation stage, restart the building process of the package

utils/brmake <package_name>-reinstall # Starting from the installation stage, restart the package's build process

Take rkwifibt as an example:

```
utils/brmake rkwifibt-reconfigure
utils/brmake rkwifibt-rebuild
utils/brmake rkwifibt-reinstall
```

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$ utils/brmake rkwifibt-reconfigure
2023-07-13T15:56:16 >>> rkwifibt 1.0.0 Syncing from source dir /home/tgg/rk3568_linux_sdk/buildroot/../external/rkwifibt
2023-07-13T15:56:16 >>> rkwifibt 1.0.0 Configuring
2023-07-13T15:56:16 >>> rkwifibt 1.0.0 Building                    Re-run the configuration -> Compile -> Install
2023-07-13T15:56:23 >>> rkwifibt 1.0.0 Installing to target
Done in 11s
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$ utils/brmake rkwifibt-rebuild
2023-07-13T15:56:29 >>> rkwifibt 1.0.0 Syncing from source dir /home/tgg/rk3568_linux_sdk/buildroot/../external/rkwifibt
2023-07-13T15:56:29 >>> rkwifibt 1.0.0 Building
2023-07-13T15:56:36 >>> rkwifibt 1.0.0 Installing to target        Re-run the compilation -> Install
Done in 11s
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$ utils/brmake rkwifibt-reinstall
2023-07-13T15:56:42 >>> rkwifibt 1.0.0 Syncing from source dir /home/tgg/rk3568_linux_sdk/buildroot/../external/rkwifibt
2023-07-13T15:56:42 >>> rkwifibt 1.0.0 Installing to target
Done in 4s                                                         Re-run the installation process
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$
```

If you need to clean the package build directory, you can execute the following command:

```
make <package_name>-dirclean
```

Take rkwifibt as an example:

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$ utils/brmake rkwifibt-dirclean
Done in 3s
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$
```

6.4.7 Add package

Here, we will introduce to users how to add their own package (software package) in the buildroot/package directory through a simple example.

1. Develop source code project

First, enter the <SDK>/app directory, and create a folder named "mypackage" in this directory, as shown below:

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk$ cd app/
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/app$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/app$ mkdir mypackage
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/app$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/app$ cd mypackage/
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/app/mypackage$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/app/mypackage$ ls
```

Figure 6.4.7.1 Create the "mypackage" folder

Create a ".c" source file named "main.c" and a "Makefile" file in the "mypackage" directory:

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/app/mypackage$ touch main.c Makefile
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/app/mypackage$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/app/mypackage$ ls
main.c  Makefile
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/app/mypackage$
```

Figure 6.4.7.2 Create the source file and Makefile

You can write a simple test code in the main.c source file yourself, such as printing "Hello World". The content of the Makefile is as follows:

```
mypackage: main.o
$(CC) -o mypackage main.o

%.o: %.c
$(CC) -c $< -o $@
```

The purpose is to compile the main.c source file into an executable file named "mypackage".

2. Add package

Enter the <Buildroot>/package directory, and create a directory named "mypackage" in this directory:

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk$ cd buildroot/package/
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot/package$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot/package$ mkdir mypackage
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot/package$ cd mypackage/
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot/package/mypackage$ ls
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot/package/mypackage$
```

Figure 6.4.7.3 Create the "mypackage" directory

Create two files, "Config.in" and "mypackage.mk", within the "mypackage" directory:

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot/package/mypackage$ touch Config.in
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot/package/mypackage$ touch mypackage.mk
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot/package/mypackage$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot/package/mypackage$ ls -l
总用量 0
-rw-rw-r-- 1 tgg tgg 0 4月  15 08:41 Config.in
-rw-rw-r-- 1 tgg tgg 0 4月  15 08:41 mypackage.mk
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot/package/mypackage$
```

Figure 6.4.7.4 Create Config.in and mypackage.mk

The content of the Config.in file is as follows:

```
config BR2_PACKAGE_MYPACKAGE
    bool "my package"
    help
      this configuration is used to enable or disable mypackage.
```

The grammar rules of the Config.in file are the same as those of the Kconfig files in the Linux Kernel and U-Boot.

The content of the mypackage.mk file is as follows:

```
##############################################################################
#
# mypackage
```

```
#
##############################################################################
##

# Define a version number for your software package
MYPACKAGE_VERSION = 1.0

# The directory where your software package is located
MYPACKAGE_SITE = $(TOPDIR)/../app/mypackage

# The way to obtain the software package: "local" indicates obtaining it locally. Some packages may need to be downloaded via the network, such as projects in the git repository.
MYPACKAGE_SITE_METHOD = local

# List the configuration operations that need to be performed before compiling the software package
define MYPACKAGE_CONFIGURE_CMDS
endef

# The operations that need to be performed when listing the compilation software packages
define MYPACKAGE_BUILD_CMDS
	$(MAKE) -C $(@D) CC=$(TARGET_CC)
endef

# List the operations that need to be performed when installing the software package to the target directory (<Buildroot>/output/rockchip_rk3568/target)
define MYPACKAGE_INSTALL_TARGET_CMDS
	$(INSTALL) -D -m 0755 $(@D)/mypackage $(TARGET_DIR)/usr/bin/mypackage
endef

# Indicates that the current software package is a general-purpose software package infrastructure
$(eval $(generic-package))
```

Note: This file defines some variables and macros. All of these variables and macros start with the prefix "MYPACKAGE_", and you must not deviate from this. They must be equal to the name of the directory where Config.in and mypackage.mk files are located (converted from lowercase to uppercase).

The functions of these variables and macros have already been explained. Besides these variables and macros, many other variables or macros can be defined in the.mk file. Each variable or macro has its own meaning. For detailed usage methods, please refer to the document: Development Board CD-ROM A Disk - **Basic Materials -> 10_user_manual -> 03. Auxiliary Documents -> [ALIENTEK] Buildroot User Manual Chinese Version (ALIENTEK Translation) - V1.0.pdf**.

$(MAKE): represents the make command;

$(@D): represents the directory where the software package is located. **Note that this directory is not <SDK>/app/mypackage, but the corresponding folder of this software package in the**

output/rockchip_rk3568/build directory; Before compiling the software package, Buildroot will copy <SDK>/app/mypackage to <Buildroot>/output/rockchip_rk3568/build directory and rename it as mypackage-1.0 (1.0 is the version number). Therefore, this "$(@D)" refers to the directory output/rockchip_rk3568/build/mypackage-1.0.

$(TOPDIR): represents the top-level directory of the buildroot, which is the <SDK>/buildroot directory.

$(TARGET_CC): represents the cross compiler. On the RK platform, the default cross compiler used is the buildroot cross compiler, and its location is: **<Buildroot>/output/rockchip_rk3568/host/bin/aarch64-buildroot-linux-gnu-gcc**.

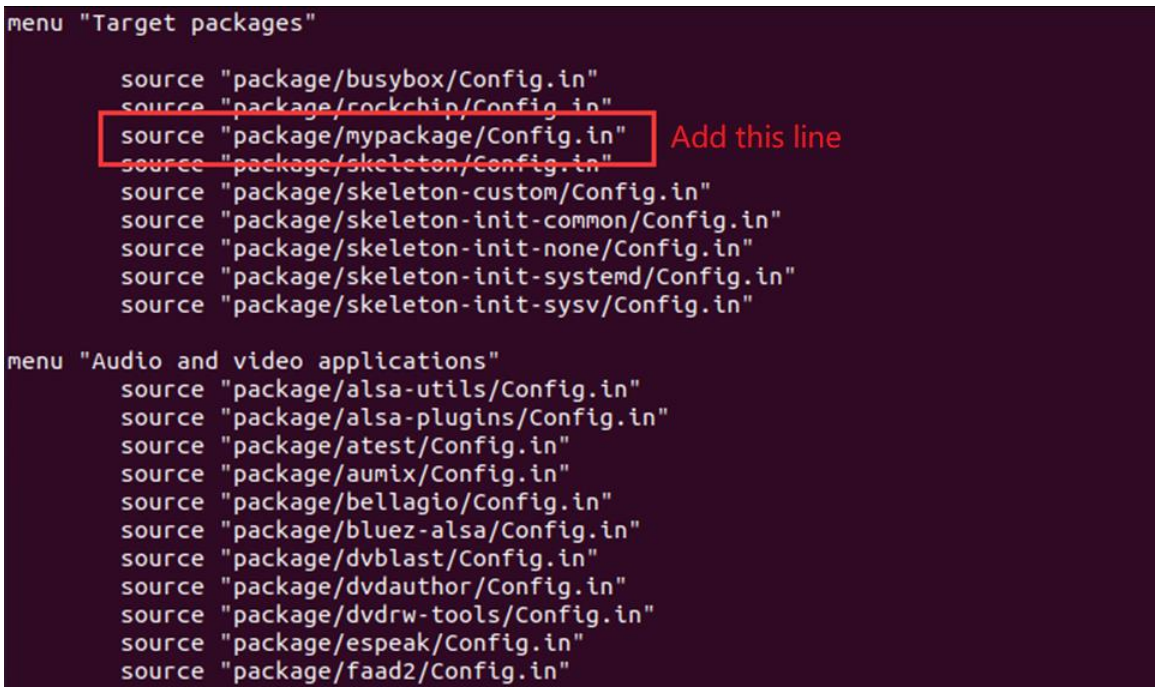
$(INSTALL): represents the install command.

$(TARGET_DIR): represents the target directory <Buildroot>/output/rockchip_rk3568/target.

That's all about the content of the "mypackage.mk" file. For more details, please refer to the document "**[ALIENTEK] Buildroot User Manual Chinese Version (Translation by ALIENTEK)_V1.0.pdf**".

Next, open the "package/Config.in" file and add the following line of content to this file:

```
source "package/mypackage/Config.in"
```



After adding, save and exit.

3. Enable and compile the package

Execute "make menuconfig" to open the graphical configuration interface, find the package we added, and then enable it:

```
Target packages    --->
    my package
```

```
-*- BusyBox
[ ]   Enable static
(package/busybox/busybox.config) BusyBox configuration file to use?
(board/rockchip/common/base/busybox.fragment) Additional BusyBox configuration fra
[*]   Show packages that are also provided by busybox
[ ]   Individual binaries
[ ]   Install the watchdog daemon startup script
[*] Rockchip BSP packages  --->
[*] my package
    Audio and video applications  --->
    Compressors and decompressors  --->
    Debugging, profiling and benchmark  --->
    Development tools  --->
    Filesystem and flash utilities  --->
    Fonts, cursors, icons, sounds and themes  --->
    Games  --->
    Graphic libraries and applications (graphic/text)  --->
    Hardware handling  --->
    Interpreter languages and scripting  --->
    Libraries  --->
    Mail  --->
    Miscellaneous  --->
    Networking applications  --->
```

Input Y to enable it.

Figure 6.4.7.5 Enable "mypackage"

Then save the configuration and exit the graphical configuration interface.

Execute the following command to compile this software package, as shown below:

```
make mypackage-rebuild
```

Or

```
utils/brmake mypackage-rebuild
```

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$ make mypackage-rebuild
rm -f /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/build/mypackage-1.0/.stamp_rsynced
rm -f /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/build/mypackage-1.0/.stamp_staging_installed
rm -f /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/build/mypackage-1.0/.stamp_target_installed
rm -f /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/build/mypackage-1.0/.stamp_images_installed
rm -f /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/build/mypackage-1.0/.stamp_host_installed
touch /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/build/mypackage-1.0/.stamp_built || true
touch: 无法创建 '/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/build/mypackage-1.0/.stamp_built': 没有那个
rm -f /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/build/mypackage-1.0/.stamp_built
>>> mypackage 1.0 Syncing from source dir /home/tgg/rk3568_linux_sdk/buildroot/../app/mypackage
rsync -au --chmod=u=rwX,go=rX --exclude .svn --exclude .git --exclude .hg --exclude .bzr --exclude CVS /home/tgg/rk3568
ockchip_rk3568/build/mypackage-1.0
>>> mypackage 1.0 Configuring
>>> mypackage 1.0 Building
/usr/bin/make -j13 -C /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/build/mypackage-1.0 CC=/home/tgg/rk35
cc
/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/host/bin/aarch64-buildroot-linux-gnu-gcc -c main.c -o main.
/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/host/bin/aarch64-buildroot-linux-gnu-gcc -o mypackage main.
>>> mypackage 1.0 Installing to target
/usr/bin/install -D -m 0755 /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk3568/build/mypackage-1.0/mypackage /
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$
```

Figure 6.4.7.6 Compile "mypackage"

The executable file "mypackage" will be generated and its location will be: output/rockchip_rk3568/target/usr/bin/mypackage.

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$ file output/rockchip_rk3568/target/usr/bin/mypackage
output/rockchip_rk3568/target/usr/bin/mypackage: ELF 64-bit LSB executable, ARM aarch64, version 1 (SYSV), d
ynamically linked, interpreter /lib/ld-linux-aarch64.so.1, for GNU/Linux 3.7.0, not stripped
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$
```

Figure 6.4.7.7 My package executable file

6.4.8 Customization of rootfs

Users can customize the rootfs for their own Linux products, so as to minimize the size of the rootfs while still meeting the functional requirements of the product.

6.5 Recovery development

The recovery.img is mainly used to bring the device into the recovery mode. The main purpose of the device entering the recovery mode is to erase user data (such as the data in the userdata partition) and perform system upgrades or other operations, etc.

The recovery.img is burned onto the recovery partition of the development board. The recovery.img is also a FIT format image, formed by merging multiple images, including the ramdisk (mounted as the root file system when entering the recovery mode), the kernel DTB, the kernel image, and resource.img. 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 that the recovery mode needs to be entered, then U-Boot will read the resource image resource.img from the recovery.img (stored in the recovery partition), and then find the kernel DTB in resource.img and load it into the memory (the kernel DTB is packed into resource.img, and resource.img is then packed into recovery.img). Then, the kernel image contained in the recovery.img will be loaded and booted, and finally, the ramdisk root file system will be mounted after entering the system.

Therefore, the recovery.img is prepared for the system to enter the recovery mode.

6.5.1 Compile recovery

recovery.img is composed of the kernel image Image, kernel DTB, resource.img, and ramdisk root file system. Therefore, to create recovery.img, the following three steps are required:

①, Compile the Linux Kernel, obtaining the kernel image Image, kernel DTB, and resource.img;

②, Compile buildroot to obtain the ramdisk root file system image;

③, Finally, package it into recovery.img.

1. Compile Linux Kernel

Enter the kernel source code directory <SDK>/kernel, directly run the make.sh script to compile the kernel. After compilation is complete, you will obtain the kernel image arch/arm64/boot/Image and the resource image resource.img.

2. Compile buildroot to obtain the ramdisk root file system image

Now we need to compile a root file system in recovery mode using buildroot. In the recovery mode, a ramdisk root file system (a root file system simulated by memory) is used, so this root file system is relatively small and has a small size.

Before compiling, we need to configure it. Enter the buildroot directory and execute the following command to perform defconfig configuration.

```
source build/envsetup.sh rockchip_rk356x_recovery
```

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$ source build/envsetup.sh rockchip_rk356x_recovery
Top of tree: /home/tgg/rk3568_linux_sdk
=========================================

#TARGET_BOARD=rk356x
#OUTPUT_DIR=output/rockchip_rk356x_recovery
#CONFIG=rockchip_rk356x_recovery_defconfig

=========================================
make: 进入目录"/home/tgg/rk3568_linux_sdk/buildroot"
mkdir -p /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk356x_recovery/build/buildroot-config/lxdialog
PKG_CONFIG_PATH="" make CC="/usr/bin/gcc" HOSTCC="/usr/bin/gcc" \
    obj=/home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk356x_recovery/build/buildroot-config -C support/kconfig
/usr/bin/gcc -D_DEFAULT_SOURCE -D_XOPEN_SOURCE=600 -DCURSES_LOC="<ncurses.h>" -DNCURSES_WIDECHAR=1 -DLOCALE  -I/home/t
-DCONFIG_=\"\"  -MM *.c > /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk356x_recovery/build/buildroot-config/
/usr/bin/gcc -D_DEFAULT_SOURCE -D_XOPEN_SOURCE=600 -DCURSES_LOC="<ncurses.h>" -DNCURSES_WIDECHAR=1 -DLOCALE  -I/home/t
-DCONFIG_=\"\"   -c conf.c -o /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk356x_recovery/build/buildroot-con
/usr/bin/gcc -D_DEFAULT_SOURCE -D_XOPEN_SOURCE=600 -DCURSES_LOC="<ncurses.h>" -DNCURSES_WIDECHAR=1 -DLOCALE  -I/home/t
-DCONFIG_=\"\"  -I. -c /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk356x_recovery/build/buildroot-config/zco
d/buildroot-config/zconf.tab.o
In file included from /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk356x_recovery/build/buildroot-config/zcon
./util.c: In function 'file_write_dep':
./util.c:54:15: warning: '..config.tmp' directive writing 12 bytes into a region of size between 1 and 4097 [-Wformat-
   54 |  sprintf(buf, "%s..config.tmp", dir);
      |               ^~~~~~~~~~~~~~~
./util.c:54:2: note: 'sprintf' output between 13 and 4109 bytes into a destination of size 4097
   54 |  sprintf(buf, "%s..config.tmp", dir);
      |  ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
In file included from /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_rk356x_recovery/build/buildroot-config/zcon
./confdata.c: In function 'conf_write':
./confdata.c:772:20: warning: '.tmpconfig.' directive writing 11 bytes into a region of size between 1 and 4097 [-Wfor
  772 |   sprintf(tmpname, "%s.tmpconfig.%d", dirname, (int)getpid());
      |                    ^~~~~~~~~~~~~~~
./confdata.c:772:3: note: 'sprintf' output between 13 and 4119 bytes into a destination of size 4097
  772 |   sprintf(tmpname, "%s.tmpconfig.%d", dirname, (int)getpid());
      |   ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
./confdata.c: In function 'conf_write_autoconf':
./confdata.c:982:15: warning: '.config.cmd' directive writing 11 bytes into a region of size between 1 and 4097 [-Wfor
  982 |  sprintf(buf, "%s.config.cmd", dir);
      |               ^~~~~~~~~~~~~~
```

Figure 6.5.1 Execute the defconfig configuration

The last parameter in the command (rockchip_rk356x_recovery) is used to specify the defconfig configuration file for the target platform (without the _defconfig suffix). The defconfig configuration file corresponding to the ramdisk root file system of the RK3568 platform is configs/rockchip_rk356x_recovery_defconfig; In the 6.3.3 section, the root file system compiled based on rockchip_rk3568_defconfig is the root file system that is mounted when the system of the RK3568 development board is started normally, which is the root file system in the Normal mode. This point needs to be clearly understood by everyone!

rockchip_rk356x_recovery_defconfig # Used for compiling the ramdisk root file system (in recovery mode)
rockchip_rk3568_defconfig # Used for mounting the root file system that is loaded during the normal startup of the system

Then, proceed with the compilation. Execute the following command:

```
./utils/brmake -j16
```

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$ ./utils/brmake -j16
2023-04-15T21:56:59 >>> host-tar 1.29 Extracting
2023-04-15T21:56:59 >>> host-tar 1.29 Patching
2023-04-15T21:56:59 >>> host-tar 1.29 Updating config.sub and config.guess
2023-04-15T21:56:59 >>> host-tar 1.29 Patching libtool
2023-04-15T21:56:59 >>> host-tar 1.29 Configuring
2023-04-15T21:57:20 >>> host-tar 1.29 Building
2023-04-15T21:57:22 >>> host-tar 1.29 Installing to host directory
2023-04-15T21:57:22 >>> host-lzip 1.19 Extracting
2023-04-15T21:57:22 >>> host-lzip 1.19 Patching
2023-04-15T21:57:22 >>> host-lzip 1.19 Configuring
2023-04-15T21:57:22 >>> host-lzip 1.19 Building
2023-04-15T21:57:23 >>> host-lzip 1.19 Installing to host directory
2023-04-15T21:57:23 >>> host-m4 1.4.19 Extracting
2023-04-15T21:57:23 >>> host-m4 1.4.19 Patching
2023-04-15T21:57:23 >>> host-m4 1.4.19 Updating config.sub and config.guess
2023-04-15T21:57:23 >>> host-m4 1.4.19 Patching libtool
2023-04-15T21:57:23 >>> host-m4 1.4.19 Configuring
2023-04-15T21:57:39 >>> host-m4 1.4.19 Building
2023-04-15T21:57:42 >>> host-m4 1.4.19 Installing to host directory
2023-04-15T21:57:42 >>> host-bison 3.0.4 Extracting
2023-04-15T21:57:42 >>> host-bison 3.0.4 Patching
2023-04-15T21:57:42 >>> host-bison 3.0.4 Updating config.sub and config.guess
2023-04-15T21:57:42 >>> host-bison 3.0.4 Patching libtool
2023-04-15T21:57:42 >>> host-bison 3.0.4 Configuring
2023-04-15T21:57:59 >>> host-bison 3.0.4 Building
2023-04-15T21:58:00 >>> host-bison 3.0.4 Installing to host directory
2023-04-15T21:58:00 >>> host-gawk 4.1.4 Extracting
2023-04-15T21:58:00 >>> host-gawk 4.1.4 Patching
2023-04-15T21:58:00 >>> host-gawk 4.1.4 Updating config.sub and config.guess
2023-04-15T21:58:00 >>> host-gawk 4.1.4 Patching libtool
2023-04-15T21:58:00 >>> host-gawk 4.1.4 Configuring
2023-04-15T21:58:06 >>> host-gawk 4.1.4 Building
2023-04-15T21:58:08 >>> host-gawk 4.1.4 Installing to host directory
2023-04-15T21:58:09 >>> host-binutils 2.36.1 Extracting
2023-04-15T21:58:11 >>> host-binutils 2.36.1 Patching
2023-04-15T21:58:11 >>> host-binutils 2.36.1 Updating config.sub and config.guess
2023-04-15T21:58:11 >>> host-binutils 2.36.1 Patching libtool
2023-04-15T21:58:11 >>> host-binutils 2.36.1 Configuring
2023-04-15T21:58:12 >>> host-binutils 2.36.1 Building
2023-04-15T21:58:33 >>> host-binutils 2.36.1 Installing to host directory
```

Figure 6.5-2 Compile buildroot (1)

```
2023-04-15T22:20:26 >>> host-xz 5.2.3 Building
2023-04-15T22:20:27 >>> host-xz 5.2.3 Installing to host directory
2023-04-15T22:20:29 >>> host-squashfs 3de1687d7432ea9b302c2db9521996f506c140a3 Extracting
2023-04-15T22:20:29 >>> host-squashfs 3de1687d7432ea9b302c2db9521996f506c140a3 Patching
2023-04-15T22:20:29 >>> host-squashfs 3de1687d7432ea9b302c2db9521996f506c140a3 Configuring
2023-04-15T22:20:29 >>> host-squashfs 3de1687d7432ea9b302c2db9521996f506c140a3 Building
2023-04-15T22:20:30 >>> host-squashfs 3de1687d7432ea9b302c2db9521996f506c140a3 Installing to host directory
2023-04-15T22:20:30 >>>   Finalizing target directory
2023-04-15T22:20:31 >>>   Sanitizing RPATH in target tree
2023-04-15T22:20:31 >>>   Copying overlay board/rockchip/common/base
2023-04-15T22:20:31 >>>   Copying overlay board/rockchip/common/recovery
2023-04-15T22:20:31 >>>   Copying overlay board/rockchip/rk356x/fs-overlay/
2023-04-15T22:20:31 >>>   Executing post-build script ../device/rockchip/common/post-build.sh
2023-04-15T22:20:31 >>>   Generating root filesystem image rootfs.cpio
2023-04-15T22:20:34 >>>   Generating root filesystem image rootfs.ext2
2023-04-15T22:20:34 >>>   Generating root filesystem image rootfs.squashfs
2023-04-15T22:20:34 >>>   Generating root filesystem image rootfs.tar
Done in 23min 40s
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$
```

Figure 6.5-3 Compile buildroot (2)

The compiled root file system image is stored in the directory: output/rockchip_rk356x_recovery/images/

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$ cd output/rockchip_rk356x_recovery/images/
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot/output/rockchip_rk356x_recovery/images$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot/output/rockchip_rk356x_recovery/images$ ls -l
总用量 138080
-rw-r--r-- 1 tgg tgg 18861568 4月  15 22:20 rootfs.cpio
-rw-r--r-- 1 tgg tgg  7806916 4月  15 22:20 rootfs.cpio.gz
-rw-r--r-- 1 tgg tgg 89628672 4月  15 22:20 rootfs.ext2
lrwxrwxrwx 1 tgg tgg       11 4月  15 22:20 rootfs.ext4 -> rootfs.ext2
-rw-r--r-- 1 tgg tgg  7835648 4月  15 22:20 rootfs.squashfs
-rw-r--r-- 1 tgg tgg 19220480 4月  15 22:20 rootfs.tar
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot/output/rockchip_rk356x_recovery/images$
```

Figure 6.5.4 Root file system image

In this directory, multiple different formats of root file system images have been generated. The **rootfs.cpio.gz** is the ramdisk root file system image we need.

3. Create recovery.img

Next, we need to create recovery.img. Package the kernel image Image, kernel DTB, resource.img, and ramdisk root file system image rootfs.cpio.gz into one file called recovery.img. Enter the buildroot directory and execute the following command to package it (the command is quite long; it is actually one command, <SDK> represents the top-level directory of the SDK, and users need to fill in according to their own actual situation):

```
../device/rockchip/common/mk-fitimage.sh
buildroot/output/rockchip_rk356x_recovery/images/recovery.img
device/rockchip/rk356x/boot4recovery.its
<SDK>/buildroot/output/rockchip_rk356x_recovery/images/rootfs.cpio.gz
```

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$ ../device/rockchip/common/mk-fitimage.sh buildroot/output/rockchip_rk356
x_recovery/images/recovery.img device/rockchip/rk356x/boot4recovery.its /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_r
k356x_recovery/images/rootfs.cpio.gz
fdt {
kernel {
ramdisk {
resource {
FIT description: U-Boot FIT source file for arm
Created:         Sat Apr 15 23:15:27 2023
 Image 0 (fdt)
  Description:  unavailable
  Created:      Sat Apr 15 23:15:27 2023
  Type:         Flat Device Tree
  Compression:  uncompressed
  Data Size:    142051 Bytes = 138.72 KiB = 0.14 MiB
  Architecture: AArch64
  Load Address: 0xffffff00
  Hash algo:    sha256
  Hash value:   9e918b39233b06917557a36d2e8d5abfddb35f970eac89be44a57923924c0a5b
 Image 1 (kernel)
  Description:  unavailable
  Created:      Sat Apr 15 23:15:27 2023
  Type:         Kernel Image
  Compression:  uncompressed
  Data Size:    22603784 Bytes = 22074.01 KiB = 21.56 MiB
  Architecture: AArch64
  OS:           Linux
  Load Address: 0xffffff01
  Entry Point:  0xffffff01
  Hash algo:    sha256
  Hash value:   638f5b1ff43672df49ccdfb5b37393dfadd3a9a17cfcb318b62807df30db3d87
 Image 2 (ramdisk)
```

```
  Type:         RAMDisk Image
  Compression:  uncompressed
  Data Size:    7805956 Bytes = 7623.00 KiB = 7.44 MiB
  Architecture: AArch64
  OS:           Linux
  Load Address: 0xffffff02
  Entry Point:  unavailable
  Hash algo:    sha256
  Hash value:   4f0e62b81e22ce1257f26bb597c587ebc9fcbebc4887b43101d22065a7389365
 Image 3 (resource)
  Description:  unavailable
  Created:      Sat Apr 15 23:15:27 2023
  Type:         Multi-File Image
  Compression:  uncompressed
  Data Size:    786432 Bytes = 768.00 KiB = 0.75 MiB
  Hash algo:    sha256
  Hash value:   96a6fec15606489cd301fdebe95e782f881be7af3bd5154ae8d7d802737ac730
 Default Configuration: 'conf'
 Configuration 0 (conf)
  Description:  unavailable
  Kernel:       kernel
  Init Ramdisk: ramdisk
  FDT:          fdt
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$
```

The script used for creating the boot.img, namely device/rockchip/common/mk-fitimage.sh, is the same. However, the ITS files they use are different.

boot.img ----> <SDK>/device/rockchip/rk356x/boot.its

recovery.img -----> <SDK>/device/rockchip/rk356x/boot4recovery.its

The first parameter in the command indicates the output path of the file "recovery.img". If using a relative address, it must be relative to the root directory of the SDK.

The second parameter in the command represents the path of the "its" file. If using a relative address, it must also be relative to the root directory of the SDK.

The third parameter in the command indicates the path of the ramdisk root file system image. Here, an absolute path is used. You need to modify it according to your specific situation.

In the above command, only the path of the ramdisk is specified, while the paths of the kernel image (the fourth parameter can be used to specify the path of the kernel image), the kernel DTB, and resource.img are not specified. This is because they all have default values:

Kernel image Image ---> kernel/arch/arm64/boot/Image（The image specified by the RK_KERNEL_IMG variable in the board-level configuration file）

Kernel DTB ---> kernel/arch/arm64/boot/dts/rockchip/rk3568-atk-evb1-ddr4-v10-linux.dtb（It is determined by the RK_KERNEL_DTS in the board-level configuration file.）

Resource.img ---> kernel/resource.img

The final generated recovery.img is stored in the <Buildroot>/output/rockchip_rk356x_recovery/images/ directory.

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$ cd output/rockchip_rk356x_recovery/images/
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot/output/rockchip_rk356x_recovery/images$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot/output/rockchip_rk356x_recovery/images$ ls -l
总用量 168688
-rw-rw-r-- 1 tgg tgg 31342592 4月  15 23:15 recovery.img
-rw-r--r-- 1 tgg tgg 18861568 4月  15 22:44 rootfs.cpio
-rw-r--r-- 1 tgg tgg  7805956 4月  15 22:44 rootfs.cpio.gz
-rw-r--r-- 1 tgg tgg 89628672 4月  15 22:44 rootfs.ext2
lrwxrwxrwx 1 tgg tgg       11 4月  15 22:44 rootfs.ext4 -> rootfs.ext2
-rw-r--r-- 1 tgg tgg  7835648 4月  15 22:44 rootfs.squashfs
-rw-r--r-- 1 tgg tgg 19220480 4月  15 22:44 rootfs.tar
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot/output/rockchip_rk356x_recovery/images$
```

6.5.2 Configuration of recovery

In the buildroot source code directory, execute the following command to configure the recovery root file system:

```
source build/envsetup.sh rockchip_rk356x_recovery    // Load the recovery configuration file
make menuconfig    // Open the menuconfig graphical configuration interface
```

The running results are as follows:

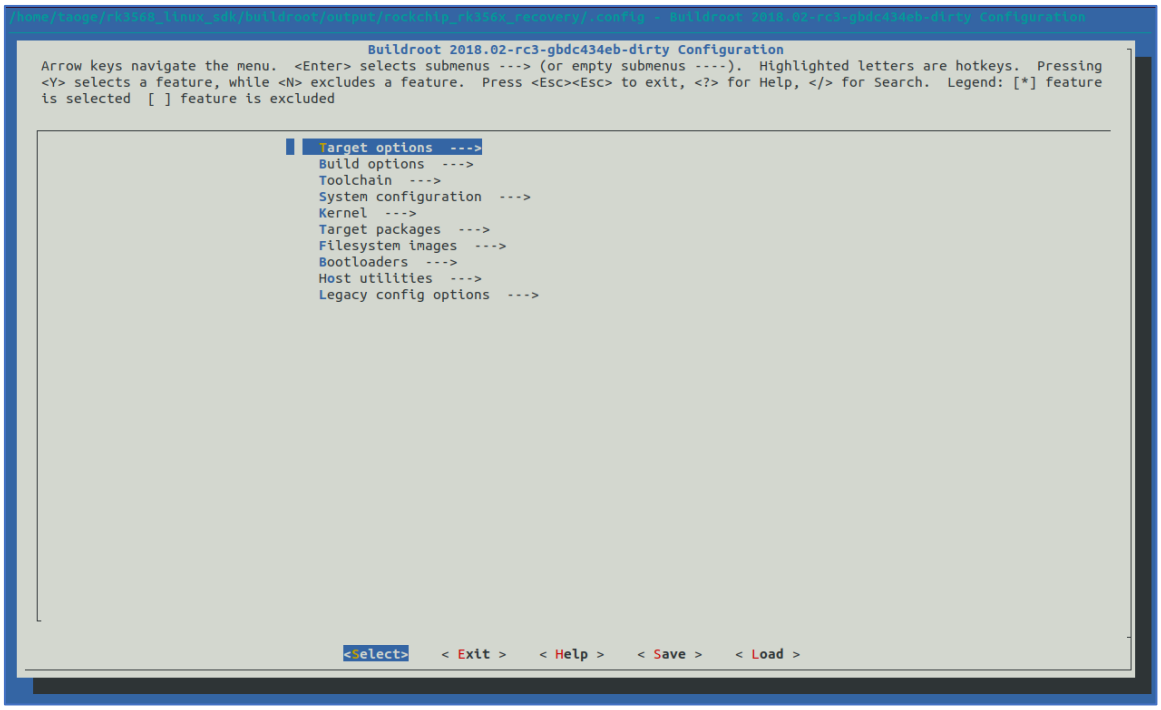


Figure 6.5.2.1 The menuconfig graphical configuration interface

Through the menuconfig graphical interface, the buildroot can be configured. After the configuration is completed, the following command can be executed to save the settings:

```
make savedefconfig
```

Next, we can go back to the SDK root directory and execute the build.sh script to compile the recovery (or we can follow the steps in Section 6.4.1 for the compilation process):

```
cd ../    // Return to the root directory of the SDK (currently located in the buildroot source code directory)
./build.sh recovery    // Execute the build.sh script to compile the recovery.
```

6.5.3 Reference Documents

Recovery-related reference documents are stored in the <SDK>/docs/Linux/Recovery directory.

6.6 OEM and Userdata

The OEM.img contains the manufacturer's apps or data. This image will be burned onto the OEM partition of the development board, and will be mounted to the root file system directory /oem after

system startup. The userdata.img contains the final user's apps or data. This image will be burned onto the userdata partition of the development board, and will be mounted to the root file system directory /userdata after system startup.

After compiling the Linux SDK (you need to execute "**./build.sh firmware**" or "**./mkfirmware.sh**"), two images, namely "oem.img" and "userdata.img", will be generated in the "rockdev/" directory. Use the "file" command to view these two images as follows:

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/rockdev$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/rockdev$ file oem.img userdata.img
oem.img:      Linux rev 1.0 ext2 filesystem data, UUID=c2c39065-01d5-46b0-b4a5-4121ada9f129 (large files)
userdata.img: Linux rev 1.0 ext2 filesystem data, UUID=f26bd67f-5898-4dcc-a450-79e0ab128157 (large files)
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/rockdev$
```

From this, it can be seen that both of these images are ext2 format file system images (determined by the **RK_USERDATA_FS_TYPE** and **RK_OEM_FS_TYPE** variables in the SDK board-level configuration file), and can be directly mounted using the mount command onto the Ubuntu system for viewing.

Both of these images are created by the mkfirmware.sh script (this script will eventually call the device/rockchip/common/mk-image.sh script, for more details, please read the mk-image.sh script content). The sizes of the oem.img and userdata.img are usually automatic and do not require configuration; if there are special requirements, these variables **RK_OEM_PARTITION_SIZE** or **RK_USERDATA_PARTITION_SIZE** can be defined in the SDK board-level configuration file to manually specify the image size. If these variables are not defined, the script will automatically determine the size.

oem.img is created from the directory device/rockchip/oem/oem_xxx (determined by the RK_OEM_DIR in the board-level configuration file); userdata.img is created from the directory device/rockchip/userdata/userdata_xxx (determined by the RK_USERDATA_DIR in the board-level configuration file); as shown below:

```
#OEM config
export RK_OEM_DIR=oem_normal
# OEM build on buildroot
#export RK_OEM_BUILDIN_BUILDROOT=YES
#userdata config
export RK_USERDATA_DIR=userdata_normal
```

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/device/rockchip/oem$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/device/rockchip/oem$ ls
oem_battery_ipc  oem_cvr  oem_empty  oem_facial_gate  oem_ipc  oem_normal  oem_nvr  oem_sample  oem_uvcc
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/device/rockchip/oem$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/device/rockchip/oem$ cd ../userdata/
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/device/rockchip/userdata$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/device/rockchip/userdata$ ls
userdata_empty  userdata_normal  userdata_sl
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/device/rockchip/userdata$
```

From this, it can be seen that the content of the oem.img image is the content in the device/rockchip/oem/oem_normal directory; the content of the userdata.img image is the content in the device/rockchip/userdata/userdata_normal directory.

If the content in the oem_normal or userdata_normal directories has changed, then the oem.img or userdata.img needs to be re-packaged; first, delete the oem.img and userdata.img in the rockdev directory, and then execute the ./mkfirmware.sh script again to create.

6.7 The parameter.txt partition table file

parameter.txt is a partition table file for the RK platform, and it 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.

The board-level configuration file of the SDK (device/rockchip/rk356x/BoardConfig-rk3568-atk-evb1-ddr4-v10.mk) defines the partition table file used by the ATK-DLRK3568 Linux system, which is determined by the RK_PARAMETER variable, as follows:

```
export RK_KERNEL_FIT_ITS=boot.its
# parameter for GPT table
export RK_PARAMETER=parameter-buildroot-fit.txt
# Buildroot config
export RK_CFG_BUILDROOT=rockchip_rk3568
# Recovery config
```

The file pointed to by this variable is device/rockchip/rk356x/parameter-buildroot-fit.txt. Therefore, after compiling the Linux SDK, the parameter.txt partition table file generated in the rockdev/ directory is actually a copy of this file (parameter-buildroot-fit.txt).

6.8 misc.img image

The misc.img image will be burned to the misc partition of the development board.

The concept of the misc partition originates from the Android system. In the Android system, the misc partition is an important partition, mainly used for communication between the Android system and the BootLoader. The misc partition contains the BCB (BootLoader Control Block) data block, which is a mechanism designed by Android to interact with the BootLoader for controlling the startup process.

The BCB data structure is as shown in the following figure:

```
struct android_bootloader_message {
    char command[32];
    char status[32];
    char recovery[768];

    /* The 'recovery' field used to be 1024 bytes.  It has only ever
     * been used to store the recovery command line, so 768 bytes
     * should be plenty.  We carve off the last 256 bytes to store the
     * stage string (for multistage packages) and possible future
     * expansion. */
    char stage[32];

    /* The 'reserved' field used to be 224 bytes when it was initially
     * carved off from the 1024-byte recovery field. Bump it up to
     * 1184-byte so that the entire bootloader_message struct rounds up
     * to 2048-byte. */
    char reserved[1184];
};
```

- command: Startup command. Currently, the following three commands are supported:

Parameters	Function
bootonce-bootloader	Start and enter U-Boot fastboot
boot-recovery	Start and enter recovery
boot-fastboot	Start and enter recovery fastboot

- recovery: This is an optional command for entering the recovery mode. It starts with "recovery\n", and can be followed by multiple parameters, which start with "--" and end with "\n". For example, "recovery\n--wipe_ab\n--wipe_package_size=345\n--reason=wipePackage\n":

Parameters	Function
update_package	OTA upgrade
retry_count	Increase the number of recovery upgrades. For example, if the device unexpectedly loses power during the upgrade process, re-enter the recovery upgrade based on this value.
wipe_data	erase user data (and cache), then reboot
wipe_cache	wipe cache (but not user data), then reboot
show_text	show the recovery text menu, used by some bootloader
sideload	
sideload_auto_reboot	an option only available in user-debug build, reboot the device without waiting
just_exit	do nothing, exit and reboot
locale	save the locale to cache, then recovery will load locale from cache when reboot
shutdown_after	return shutdown
wipe_all	Erase the entire userdata partition
wipe_ab	wipe the current A/B device, with a secure wipe of all the partitions in RECOVERY_WIPE
wipe_package_size	wipe package size
prompt_and_wipe_data	prompt the user that data is corrupt, with their consent erase user data (and cache), then reboot
fw_update	SD card firmware upgrade
factory_mode	The factory mode is mainly used for conducting some equipment tests, such as PCBA tests.
pcba_test	Start the PCBA testing process
resize_partition	Reconfigure the partition size and support for dynamic partitions in Android Q
rk_fwupdate	Specify rk SD/USB firmware upgrade, the scope is limited to U-Boot only

In the Linux system, it is often used during system upgrades or when restoring factory settings. During the U-Boot stage, the BCB data in the misc partition is read. Based on the content of the command in the BCB data, it is determined whether to enter the normal system or the recovery mode (if the command is "boot-recovery", it enters the recovery mode; if the command is empty, it enters the normal system).

When the device enters the recovery mode, it reads the content of the recovery section in the misc partition, thereby performing different actions, such as upgrading the partition firmware, erasing the user partition data, or other operations, etc.

The misc.img images are all pre-compiled. RK provides multiple misc.img images for users to choose from. These images are stored in the directory device/rockchip/rockimg/, as shown below:

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk$ cd device/rockchip/rockimg/
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/device/rockchip/rockimg$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/device/rockchip/rockimg$ ls -lh *misc*
-rwxrwxr-x 1 tgg tgg  48K 7月  12 09:25 blank-misc.img
-rwxrwxr-x 1 tgg tgg 3.0K 7月  12 09:25 dfu_misc_a.img
-rwxrwxr-x 1 tgg tgg 3.0K 7月  12 09:25 dfu_misc_b.img
-rwxrwxr-x 1 tgg tgg  48K 7月  12 09:25 pcba_small_misc.img
-rwxrwxr-x 1 tgg tgg  48K 7月  12 09:25 pcba_whole_misc.img
-rw-rw-r-- 1 tgg tgg  20K 7月  12 09:25 sdupdate-ab-misc.img
-rwxrwxr-x 1 tgg tgg  48K 7月  12 09:25 wipe_all-misc.img
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/device/rockchip/rockimg$
```

Among these misc.img images, the commonly used ones are blank-misc.img and wipe_all-misc.img, while the others are less frequently used. The difference between them lies in the commands and recovery identifiers in the BCB data block. We can use tools like WinHex or UltraEdit to open the misc.img image file in binary format and locate the position of the BCB data block (at the offset of 16KB, 0x4000), as shown below:

pcba_small_misc.img | wipe_all-misc.img | blank-misc.img

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	ANSI ASCII
00003FE0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00003FF0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00004000	62	6F	6F	74	2D	72	65	63	6F	76	65	72	79	00	00	00	boot-recovery
00004010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00004020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00004030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00004040	72	65	63	6F	76	65	72	79	0A	2D	2D	66	61	63	74	6F	recovery --facto
00004050	72	79	5F	6D	6F	64	65	3D	73	6D	61	6C	6C	00	00	00	ry_mode=small
00004060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00004070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00004080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00004090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000040A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000040B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000040C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

Figure 6.8-1 pcba_small_misc.img image file

pcba_small_misc.img | wipe_all-misc.img | blank-misc.img

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	ANSI ASCII
00003FE0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00003FF0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00004000	62	6F	6F	74	2D	72	65	63	6F	76	65	72	79	00	00	00	boot-recovery
00004010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00004020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00004030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00004040	72	65	63	6F	76	65	72	79	0A	2D	2D	77	69	70	65	5F	recovery --wipe_
00004050	61	6C	6C	00	00	00	00	00	00	00	00	00	00	00	00	00	all
00004060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00004070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00004080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00004090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000040A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000040B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000040C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000040D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000040E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

Figure 6.8-2 wipe_all-misc.img image

pcba_small_misc.img | wipe_all-misc.img | blank-misc.img

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	ANSI ASCII
00003FE0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00003FF0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00004000	20	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00004010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00004020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00004030	00	20	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00004040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00004050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00004060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00004070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00004080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00004090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000040A0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000040B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000040C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000040D0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000040E0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000040F0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

Figure 6.8-3 blank-misc.img image

Regarding these identifiers, the previous text has already provided an introduction (in comparison with the BCB data structure).

The current used misc.img image is determined by the RK_MISC variable in the SDK board-level configuration file (device/rockchip/rk356x/BoardConfig-rk3568-atk-evb1-ddr4-v10.mk), as follows:

```
export RK_USERDATA_DIR=userdata_normal
#misc image
export RK_MISC=blank-misc.img
#choose enable distro module
export RK_DISTRO_MODULE=
```

The file specified by this variable is blank-misc.img located in the rockimg directory. Therefore, after compiling the Linux SDK, the misc.img image generated in the rockdev/ directory is actually a copy of this file (blank-misc.img).

6.9 Qt Application Development

Reference documents:

Development board CD-ROM A disc - **Basic materials -> 10_user_manual -> 03. Supplementary documents -> [ALIENTEK] ATK-DLRK3568 Qt Development Environment Setup.pdf**.

Development board CD-ROM A disc - **Basic materials ->10_user_manual -> 02. Development documents -> [ALIENTEK] ATK-DLRK3568 Embedded Qt Development Practical Experience**.

6.10 Other application development

Application Note reference documents: <SDK>/docs/Linux/ApplicationNote

```
├── Rockchip_Developer_Guide_Debian_CN.pdf
├── Rockchip_Developer_Guide_Debian_Docker_EN.pdf
├── Rockchip_Developer_Guide_Debian_EN.pdf
├── Rockchip_Developer_Guide_Linux_Docker_Deploy_CN.pdf
├── Rockchip_Developer_Guide_Linux_Flash_Open_Source_Solution_CN.pdf
├── Rockchip_Developer_Guide_Linux_Flash_Open_Source_Solution_EN.pdf
├── Rockchip_Developer_Guide_Linux_PCBA_CN.pdf
├── Rockchip_Developer_Guide_Linux_PCBA_EN.pdf
├── Rockchip_Developer_Guide_Third_Party_System_Adaptation_CN.pdf
├── Rockchip_Developer_Guide_Third_Party_System_Adaptation_EN.pdf
├── Rockchip_Instruction_Linux_ROS2_CN.pdf
├── Rockchip_Instruction_Linux_ROS_CN.pdf
├── Rockchip_Instruction_Linux_ROS_EN.pdf
├── Rockchip_Quick_Start_Linux_USB_Gadget_CN.pdf
├── Rockchip_Quick_Start_Linux_USB_Gadget_EN.pdf
├── Rockchip_Use_Guide_Linux_RetroArch_CN.pdf
├── Rockchip_User_Guide_Linux_RetroArch_EN.pdf
└── Rockchip_User_Guide_SDK_Docker_CN.pdf
```

Camera development-related documents:

<SDK>/docs/Linux/Camera

```
├── Rockchip_Developer_Guide_Linux4.4_Camera_CN.pdf
├── Rockchip_Developer_Guide_Linux4.4_Camera_EN.pdf
├── Rockchip_Developer_Guide_Linux_RMSL_CN.pdf
├── Rockchip_Developer_Guide_Linux_RMSL_EN.pdf
├── Rockchip_Trouble_Shooting_Linux4.4_Camera_CN.pdf
```

```
└──── Rockchip_Trouble_Shooting_Linux4.4_Camera_EN.pdf
```

Documents related to WIFI/Bluetooth development:

```
<SDK>/docs/Linux/Wifibt
├──── AP Module RF Test Document
│     ├──── BT RF Test Commands for Linux-v05.pdf
│     └──── Wi-Fi RF Test Commands for Linux-v03.pdf
├──── REALTEK Module RF Test Document
│     ├──── 00014010-WS-170731-RTL8723D_COB_MP_FLOW_R04.pdf
│     ├──── MP tool user guide for linux20180319.pdf
│     └──── Quick_Start_Guide_V6.txt
├──── RK Platform - RTL8723DS_AIRKISS Wiring Instructions.pdf
├──── Rockchip_Developer_Guide_DeviceIo_Bluetooth_CN.pdf
├──── Rockchip_Developer_Guide_DeviceIo_Bluetooth_EN.pdf
├──── Rockchip_Developer_Guide_Linux_WIFI_BT_CN.pdf
├──── Rockchip_Developer_Guide_Linux_WIFI_BT_EN.pdf
├──── Rockchip_Developer_Guide_Network_Config_CN.pdf
├──── Rockchip_Developer_Guide_Network_Config_EN.pdf
├──── Rockchip_Introduction_RK3308_DeviceIo_WIFI_CN.pdf
├──── Rockchip_Introduction_RK3308_DeviceIo_WIFI_EN.pdf

<SDK>/external/deviceio_release/doc
├──── Rockchip_Developer_Guide_DeviceIo_Bluetooth_CN.pdf
├──── Rockchip_Developer_Guide_Linux_WIFI_BT_CN.pdf
├──── Rockchip_Developer_Guide_Network_Config_CN.pdf
└──── Rockchip_Introduction_RK3308_DeviceIo_WIFI_CN.pdf
```

Summary of documents in the <SDK>/external directory:

```
<SDK>/external
./rkfacial/doc/Rockchip_Instruction_Rkfacial_CN.pdf
./uvc_app/doc/zh-cn/Rockchip_Introduction_Linux_UVCApp_CN.pdf
./rockit/mpi/doc/Rockchip_FAQ_MPI_CN.pdf
./rockit/mpi/doc/Rockchip_Developer_Guide_MPI_CN.pdf
./rockit/tgi/doc/Rockchip_Developer_Guide_Linux_Rockit_CN.pdf
./uac_app/doc/zh-cn/Rockchip_Developer_Guide_Microphone_Array_TEST_CN.pdf
./uac_app/doc/zh-cn/Rockchip_Developer_Guide_Linux_UACApp_CN.pdf
./uac_app/doc/zh-cn/Rockchip_Quick_Start_Linux_UAC_CN.pdf
./deviceio_release/doc/Rockchip_Developer_Guide_DeviceIo_Bluetooth_CN.pdf
./deviceio_release/doc/Rockchip_Introduction_RK3308_DeviceIo_WIFI_CN.pdf
./deviceio_release/doc/Rockchip_Developer_Guide_Network_Config_CN.pdf
./deviceio_release/doc/Rockchip_Developer_Guide_Linux_WIFI_BT_CN.pdf
./camera_engine_rkaiq/rkisp2x_tuner/Doc/Rockchip_IQ_Tools_Guide_ISP2x_CN_v2.0.2.pdf
./rknn-toolkit2/doc/Rockchip_Quick_Start_RKNN_Toolkit2_CN-1.3.0.pdf
```

./rknn-toolkit2/doc/Rockchip_Quick_Start_RKNN_Toolkit2_EN-1.3.0.pdf
./rknn-toolkit2/doc/Rockchip_User_Guide_RKNN_Toolkit2_EN-1.3.0.pdf
./rknn-toolkit2/doc/Rockchip_User_Guide_RKNN_Toolkit2_CN-1.3.0.pdf
./rknn-toolkit2/rknn_toolkit_lite2/docs/Rockchip_User_Guide_RKNN_Toolkit_Lite2_V1.3.0_EN.pdf
./rknn-toolkit2/rknn_toolkit_lite2/docs/Rockchip_User_Guide_RKNN_Toolkit_Lite2_V1.3.0_CN.pdf
./rknpu2/doc/Rockchip_RKNPU_User_Guide_RKNN_API_V1.3.0_EN.pdf
./rknpu2/doc/Rockchip_RKNPU_User_Guide_RKNN_API_V1.3.0_CN.pdf
./rknpu2/doc/Rockchip_RV1106_Quick_Start_RKNN_SDK_V1.3.0b0_CN.pdf
./rknpu2/doc/Rockchip_Quick_Start_RKNN_SDK_V1.3.0_CN.pdf

Linux C Application Programming Reference Manual (provided by ALIENTEK):

Development Board CD-ROM A Disk - **Basic Materials →10_user_manual→ 02. Development Documents→ [ALIENTEK] Linux_C Application Programming Reference Manual V1.0.pdf**

6.11 Yocto Development

It is not recommended for users to use Yocto! The Linux Technology Team of ALIENTEK will not provide maintenance for it.

For more information, please refer to: http://opensource.rock-chips.com/wiki_Yocto

6.12 NPU Development

RK Reference Document List:

```
external/rknn-toolkit2/doc
├── Rockchip_Quick_Start_RKNN_Toolkit2_CN-1.3.0.pdf
├── Rockchip_Quick_Start_RKNN_Toolkit2_EN-1.3.0.pdf
├── Rockchip_User_Guide_RKNN_Toolkit2_CN-1.3.0.pdf
├── Rockchip_User_Guide_RKNN_Toolkit2_EN-1.3.0.pdf

external/rknn-toolkit2/rknn_toolkit_lite2/docs
├── Rockchip_User_Guide_RKNN_Toolkit_Lite2_V1.3.0_CN.pdf
└── Rockchip_User_Guide_RKNN_Toolkit_Lite2_V1.3.0_EN.pdf

external/rknpu2/doc
├── Rockchip_Quick_Start_RKNN_SDK_V1.3.0_CN.pdf
├── Rockchip_RKNPU_User_Guide_RKNN_API_V1.3.0_CN.pdf
├── Rockchip_RKNPU_User_Guide_RKNN_API_V1.3.0_EN.pdf
└── Rockchip_RV1106_Quick_Start_RKNN_SDK_V1.3.0b0_CN.pdf
```

6.13 Debian Development

This SDK provides support for Debian 10, based on the X11 display architecture.

The system is based on the Linaro version and adds support for graphics and video acceleration, including packages such as libmali, xserver, gstreamer rockchip, etc.

Reference for Debian Development Documentation (provided by ALIENTEK):

Development board CD-ROM A disc - **Basic Information →10_user_manual→02. Development Documents→ [ALIENTEK] ATK-DLRK3568_Debian System Development Manual V1.0.pdf**

Debian Development Documentation Reference (Provided by RK):

<SDK>/docs/Linux/ApplicationNote/Rockchip_Developer_Guide_Debian_Docker_EN.pdf

<SDK>/docs/Linux/ApplicationNote/Rockchip_Developer_Guide_Debian_CN.pdf

6.14 Multimedia Development

RK Reference Document:

<SDK>/docs/Linux/Multimedia

```
├── Rockchip_Developer_Guide_Linux_RGA_CN.pdf
├── Rockchip_Developer_Guide_Linux_RGA_FAQ_CN.pdf
├── Rockchip_Developer_Guide_Linux_RKADK_CN.pdf
├── Rockchip_Developer_Guide_MPP_CN.pdf
├── Rockchip_Developer_Guide_MPP_EN.pdf
├── Rockchip_Introduction_Linux_Audio_3A_Algorithm_CN.pdf
├── Rockchip_Introduction_Linux_Audio_3A_Algorithm_EN.pdf
├── Rockchip_User_Guide_Linux_Gstreamer_CN.pdf
└── Rockchip_User_Guide_Linux_Gstreamer_EN.pdf
```

<SDK>/external/rockit/mpi/doc

```
├── Rockchip_Developer_Guide_MPI_CN.pdf
└── Rockchip_FAQ_MPI_CN.pdf
```

6.15 Graphics Development

RK Reference Documentation:

<SDK>/docs/Linux/Graphics

```
├── Rockchip_Developer_Guide_Buildroot_Weston_CN.pdf
├── Rockchip_Developer_Guide_Buildroot_Weston_EN.pdf
├── Rockchip_Developer_Guide_Linux_Graphics_CN.pdf
└── Rockchip_Developer_Guide_Linux_Graphics_EN.pdf
```

6.16 Development of Security Mechanisms

RK Reference Document:
U-Boot Documentation
docs/Common/UBOOT/Rockchip_Developer_Guide_UBoot_Nextdev_CN.pdf

<SDK>/docs/Linux/Security

```
├── Rockchip_Developer_Guide_Crypto_HWRNG_CN.pdf
├── Rockchip_Developer_Guide_Linux_Secure_Boot_CN.pdf
├── Rockchip_Developer_Guide_Linux_Secure_Boot_EN.pdf
└── Rockchip_Developer_Guide_TEE_SDK_CN.pdf
```

6.17 A/B System Development

RK Reference Document:
U-Boot Documentation
docs/Common/UBOOT/Rockchip_Developer_Guide_UBoot_Nextdev_CN.pdf
docs/Common/UBOOT/Rockchip_Developer_Guide_Linux_AB_System_CN.pdf

6.18 System Upgrade Development

System Upgrade Plan, RK Reference Document:
<SDK>/docs/Linux/Recovery

```
├── Rockchip_Developer_Guide_Linux_DFU_Upgrade_CN.pdf
├── Rockchip_Developer_Guide_Linux_Recovery_CN.pdf
├── Rockchip_Developer_Guide_Linux_Recovery_EN.pdf
├── Rockchip_Developer_Guide_Linux_Upgrade_CN.pdf
├── Rockchip_Developer_Guide_Linux_Upgrade_EN.pdf
└── Rockchip_Introduction_Smart_Screen_OTA_CN.pdf
```