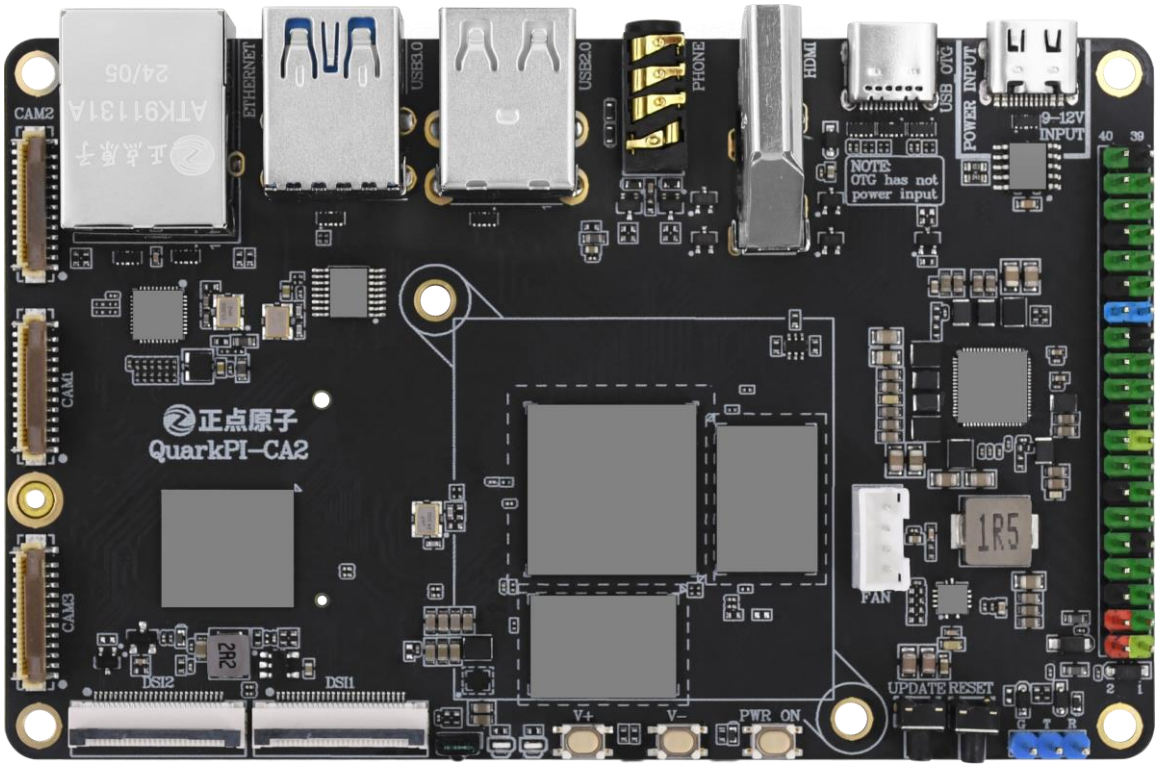
QuarkPi-CA2 Buildroot

SDK Development manual

V1.0



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	2025.03.12

Catalogue

Brief 1
Chapter 1. Overview of Embedded Development Environment 2
Chapter 2. Installing the Ubuntu System 3
2.1 Installing VMware Virtual Machine Software 4
2.1.1 Download VMware software 4
2.1.2 Installing VMware software 6
2.2 Installing the Ubuntu system 12
2.2.1 Download Ubuntu System Image 12
2.2.2 Creating a virtual machine 14
2.2.3 Installing the Ubuntu System 24
Chapter 3. Development Environment Setup 39
3.1 Ubuntu System Settings 40
3.1.1 Set the root user password 40
3.1.2 Change the software download source 40
3.1.3 Disable Automatic Updates 44
3.2 File transfer between Ubuntu and Windows 45
3.2.1 Setting up an FTP server on the Ubuntu system 45
3.2.2 Installing FTP client on Windows 46
3.2.3 Using FileZilla 50
3.3 Setting up the ssh server in the Ubuntu system 56
3.4 Installation of CH343 serial port driver 56
3.5 Installation of MobaXterm Software 59
3.5.1 Download of MobaXterm Software 59
3.5.2 Installation of MobaXterm software 60
3.5.3 Usage of MobaXterm software 63
3.6 Installation of Rockchip USB Driver 71
3.7 Use of Mirror Image Burning Tools 73
3.7.1 Flashing Mode Introduction 73
3.7.2 Usage of RKDevTool tool under Windows 74
3.7.3 Method for Burning the update.img Image 81
3.7.4 Erase operation 82
3.7.5 Usage of the Linux_Upgrade_Tool tool under Ubuntu 84
3.8 Installation of ADB Tools 84
Chapter 4. QuarkPi-CA2 Linux SDK Software Package 85
4.1 Installing the QuarkPi-CA2 Linux SDK 86
4.1.1 Installing dependent software packages 86
4.1.2 Install repo (skip) 86
4.1.3 Git configuration 86
4.1.4 Install SDK 87
4.1.5 SDK issue feedback 88
4.1.6 SDK Optimization 88

4.2 Introduction to the SDK Software Architecture....88
4.2.1 Introduction to SDK Project Directory Structure....89
4.2.2 SDK Software Block Diagram....89
4.2.3 Query of SDK Version....90
4.3 SDK Automatic Compilation....91
4.4 Independent Compilation....97
4.4.1 Independent Compilation of U-Boot....97
4.4.2 Independently compiling the Linux kernel....99
4.4.3 Independently compiling rootfs....101
4.4.4 Independently compiling recovery....102
4.4.5 Creating the update.img firmware....104
4.5 SDK Cleanup....105
4.6 Image Introduction....105
Chapter 5. Mirror Image Burning....108
5.1 Flashing Mode Introduction....109
5.2 Flashing in Windows System....109
5.2.1 Introduction to Partition Table parameter.txt....110
5.2.2 Configuration....112
5.2.3 Flashing....114
5.2.4 Booting the System....115
5.2.5 Burn update.img....116
5.3 Writing on the Ubuntu System....116
5.3.1 Connect the development board to Ubuntu....117
5.3.2 Using the upgrade_tool tool for flashing (in LOADER mode)....117
5.3.3 Flash update.img....120
5.3.4 Erase Operation....120
5.3.5 Writing using the rkflash.sh script....121
Chapter 6. SDK Development....124
6.1 Board-level Configuration of SDK....125
6.2 U-Boot Development....125
6.2.1 U-Boot's Device Tree....126
6.2.2 U-Boot Compilation....126
6.2.3 Configuration of U-Boot....127
6.2.4 Shortcuts....128
6.3 Kernel Development....128
6.3.1 Compilation of the Linux kernel....129
6.3.2 Kernel Configuration....129
6.3.3 Kernel Device Tree and defconfig Configuration File....129
6.3.4 Replacing the logo....131
6.3.5 Summary of Kernel Module Development Documentation....132
6.4 Buildroot Development....135
6.4.1 Introduction to the buildroot Directory Structure....135
6.4.2 Common Compilation Commands....136

6.4.3 Compiling the root file system....137
6.4.4 Configuration of rootfs....139
6.4.5 Introduction to the output directory....140
6.4.6 Compiling the package....141
6.4.7 Adding a package....142
6.4.8 Customization of rootfs....146
6.5 Recovery Development....147
6.5.1 Compilation of recovery....147
6.5.2 Configuration of recovery....147
6.5.3 Reference Documents....148
6.6 misc.img Image....148

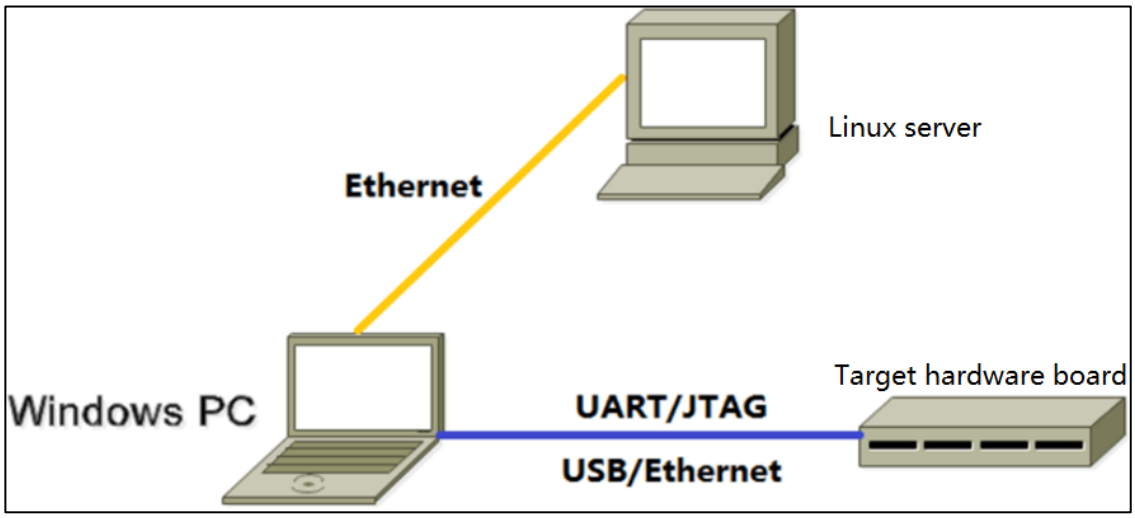
Brief

This document mainly records the relevant contents of the Linux system development for the ALIENTEK QuarkPi-CA2 platform, including the following sections:

1. Installing the Ubuntu operating system;
2. Setting up the development environment;
3. Introduction to the ALIENTEK QuarkPi-CA2 platform;
4. Installation and usage of the QuarkPi-CA2 Linux SDK;
5. Image burning;
6. SDK development.

Chapter 1. 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-compiling 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 your Windows PC. Use the virtual machine software to create a virtual machine on your Windows PC, and install the Linux operating system (such as Ubuntu) on the virtual machine. Use the virtual machine with the Linux system to replace the Linux server.

Chapter 2. Installing the Ubuntu System

Embedded Linux development is usually based on a Windows + Ubuntu dual-system development environment. This chapter will introduce to you how to install the Ubuntu operating system on a virtual machine.

This chapter will be divided into the following sections:

1. Installing VMware virtual machine software;
2. Installing the Ubuntu system.

2.1 Installing VMware Virtual Machine Software

VMware(Vmware Workstation) is a highly powerful virtual machine software. With this software, we can create one or more virtual machine on a Windows PC. A virtual machine (Virtual Machine) refers to a computer system that is simulated by software and has the complete hardware system functions of an actual computer (such as memory, hard disk, processor, etc.). 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 on the actual computer can also be accomplished in the virtual machine, and the virtual machine can be used just like an actual machine.

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

2.1.1 Download VMware software

The development board data disk has already provided the installation package file of VMware virtual machine software for everyone. The path is: Development Board CD-ROM A Drive - Basic Materials -> 03_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 2.1-1 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.

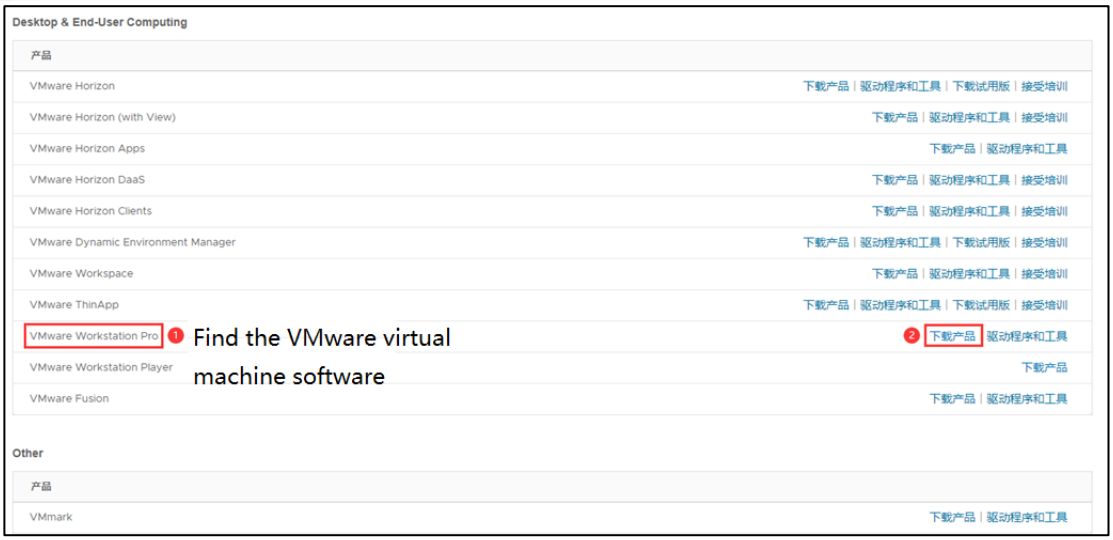


Figure 2.1-2 Download of the old version VMware virtual machine software installation package 1

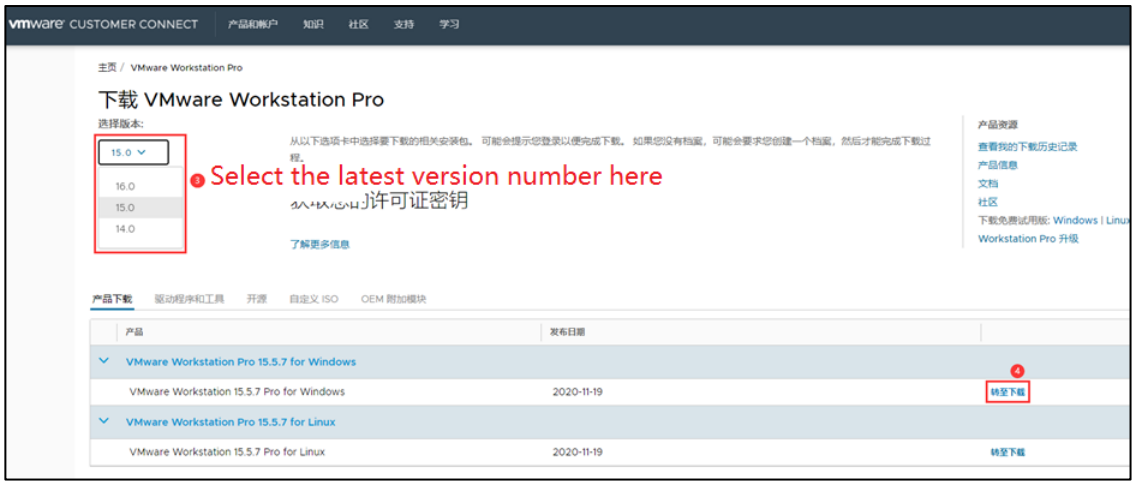


Figure 2.1-3 Download of the old version VMware virtual machine software installation package 2

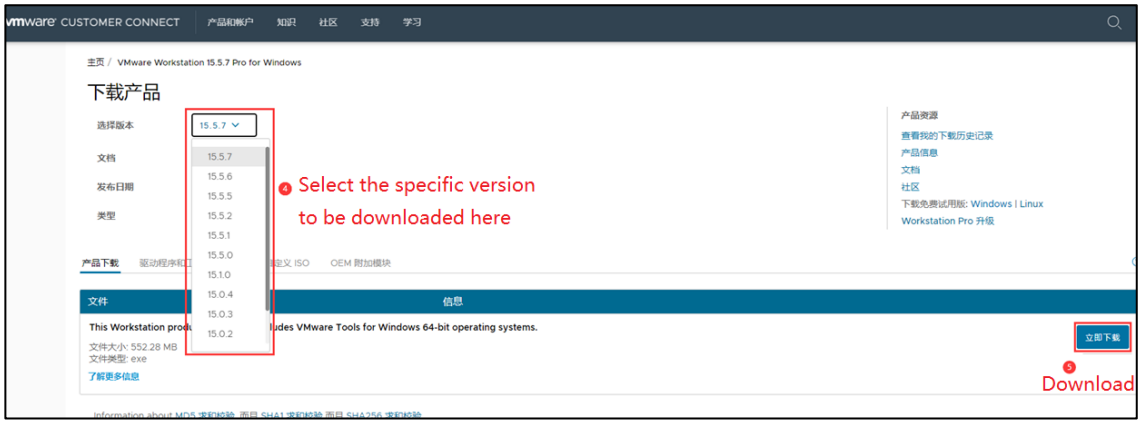


Figure 2.1-4 Download of the old version VMware virtual machine software installation package 3

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 is no specific requirement for the version of 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

take version 16.2.1 as an example, and the corresponding installation package file name is VMware-workstation-full-16.2.1-18811642.exe.

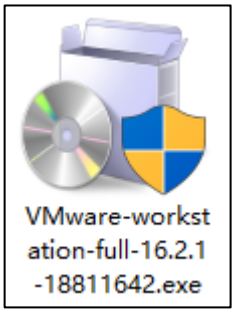


Figure 2.1-5 16.2.1 version of VMware virtual machine software installation package file

2.1.2 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:

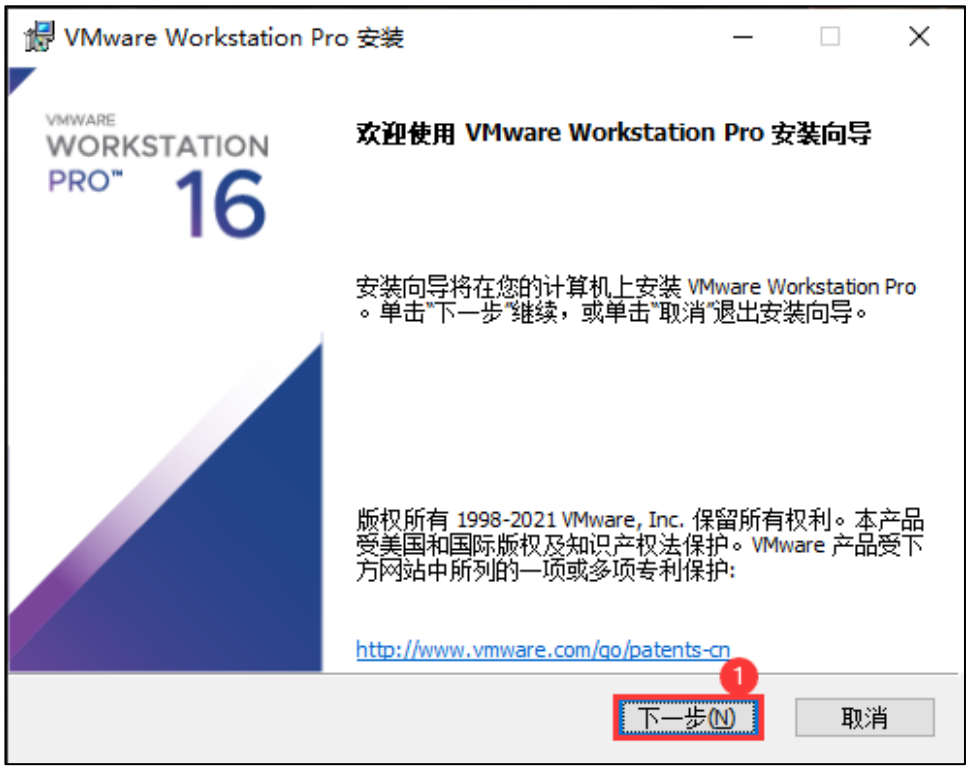


Figure 2.1-6 Install VMware software 1

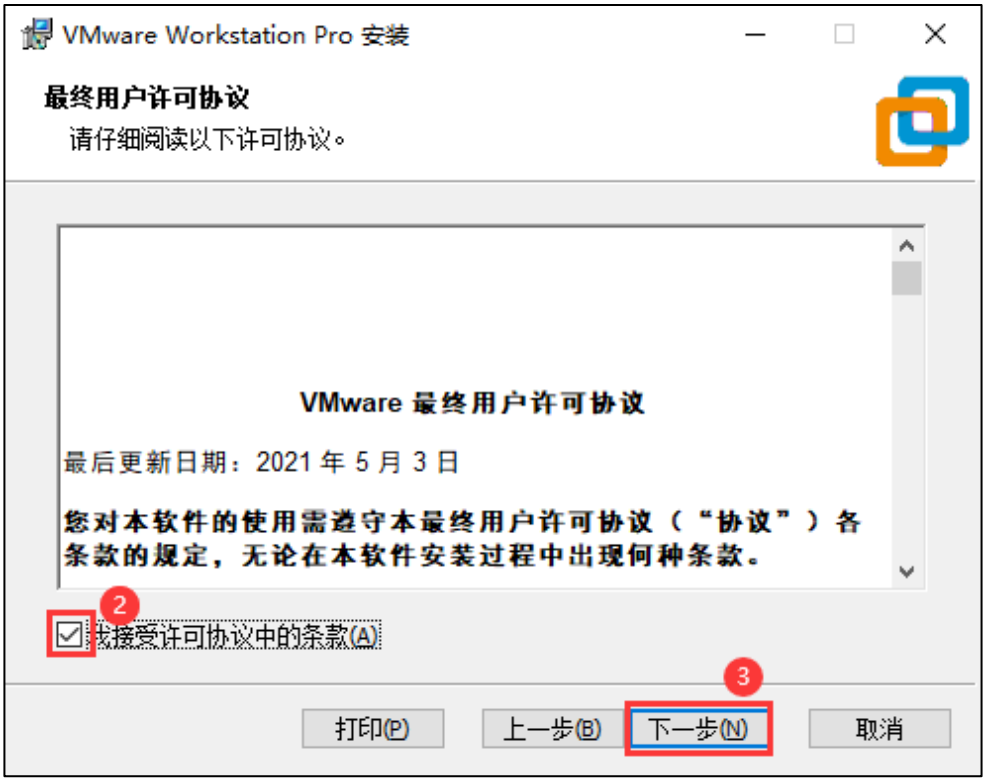


Figure 2.1-7 Install VMware software 2

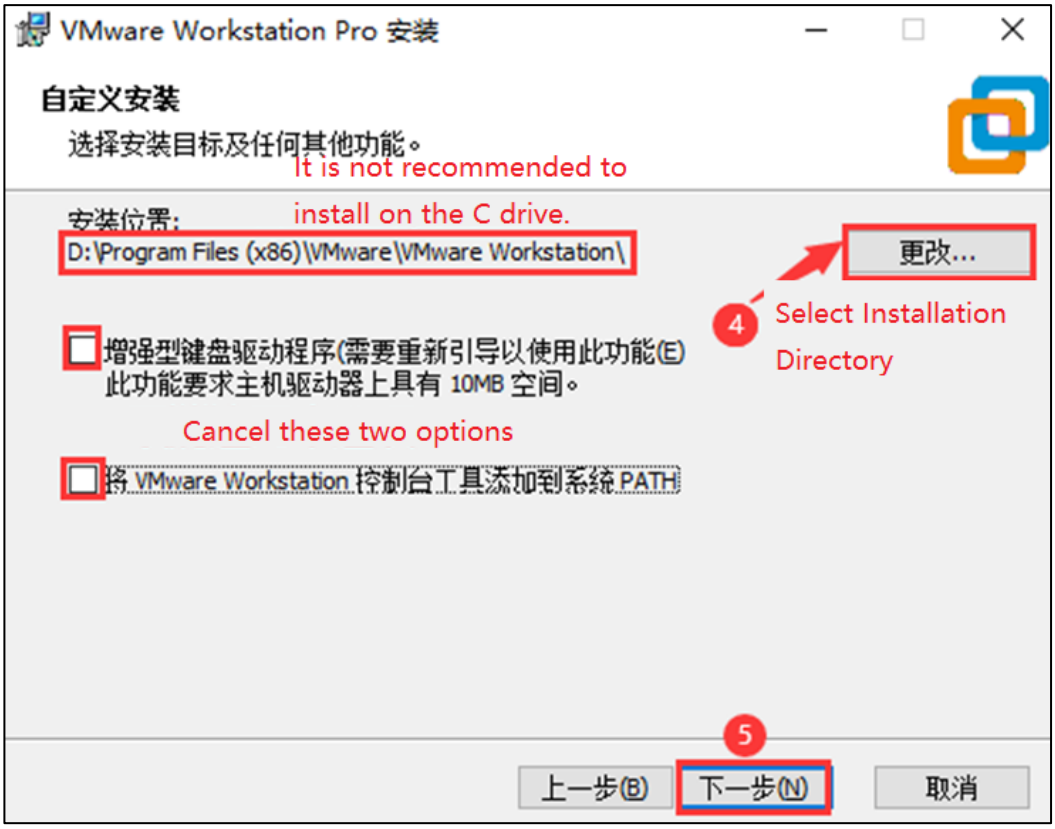


Figure 2.1-8 Install VMware software 3

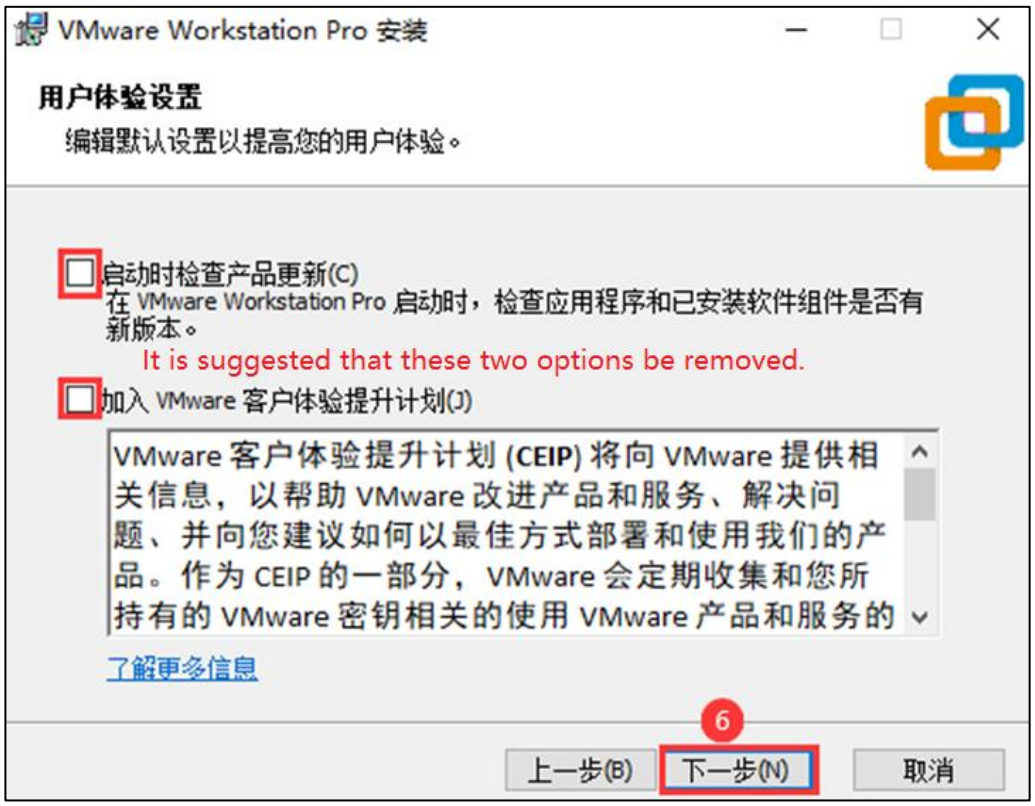


Figure 2.1-9 Install VMware software 4

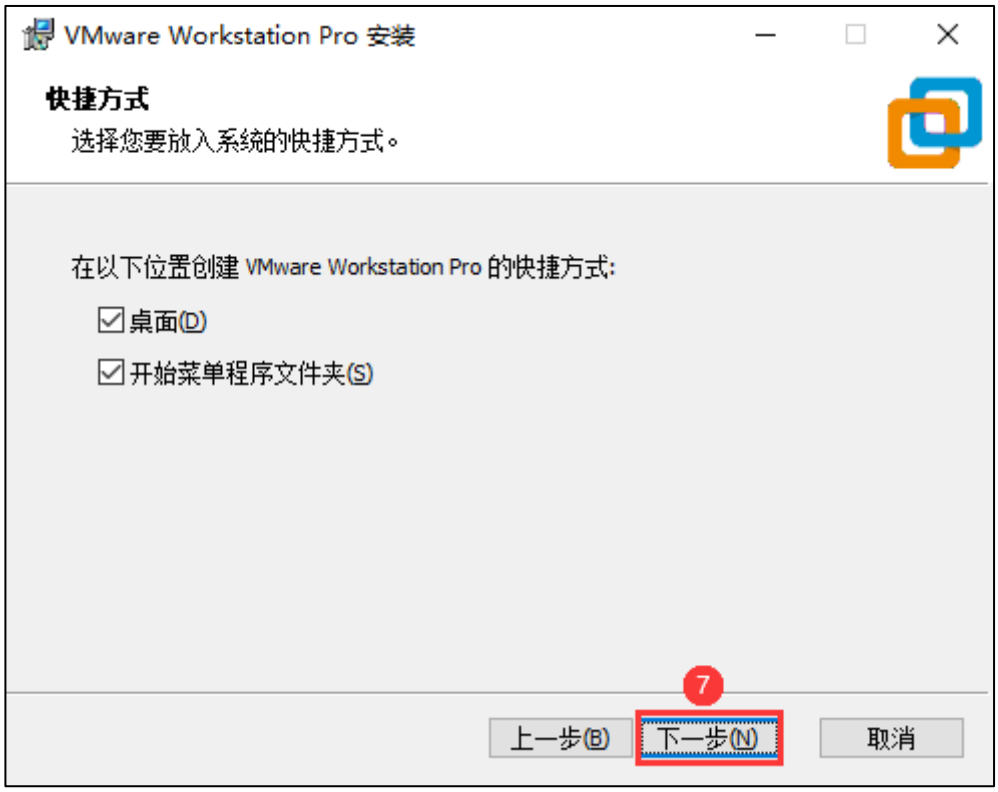


Figure 2.1-10 Install VMware software 5

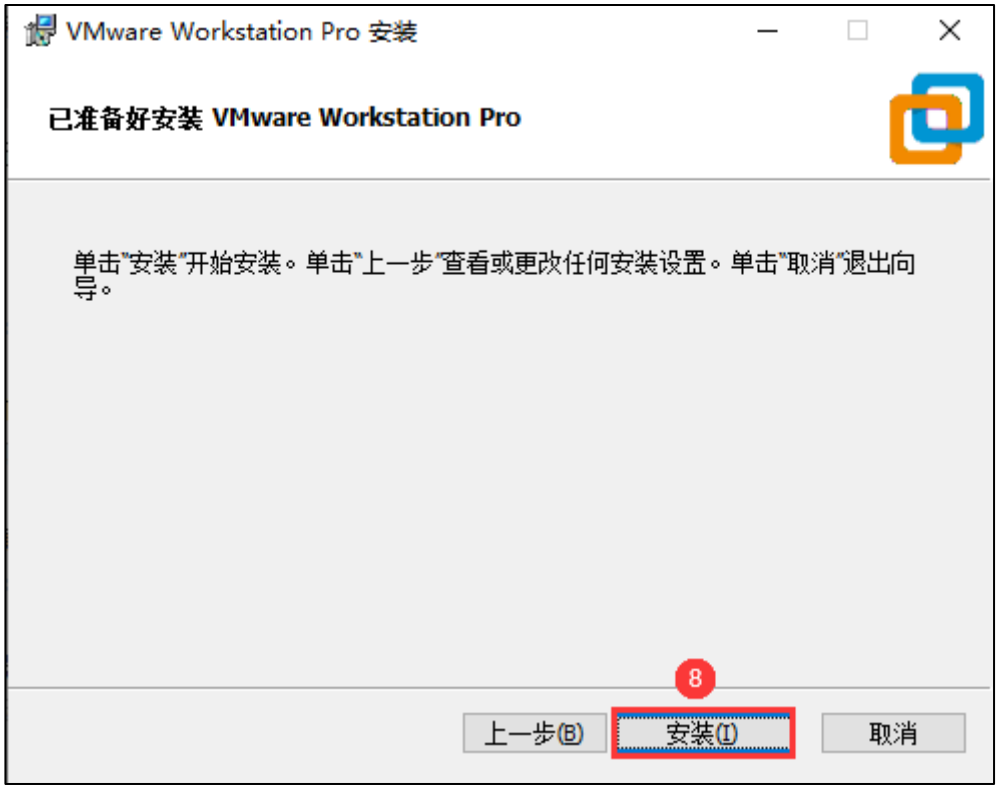


Figure 2.1-11 Install VMware software 6

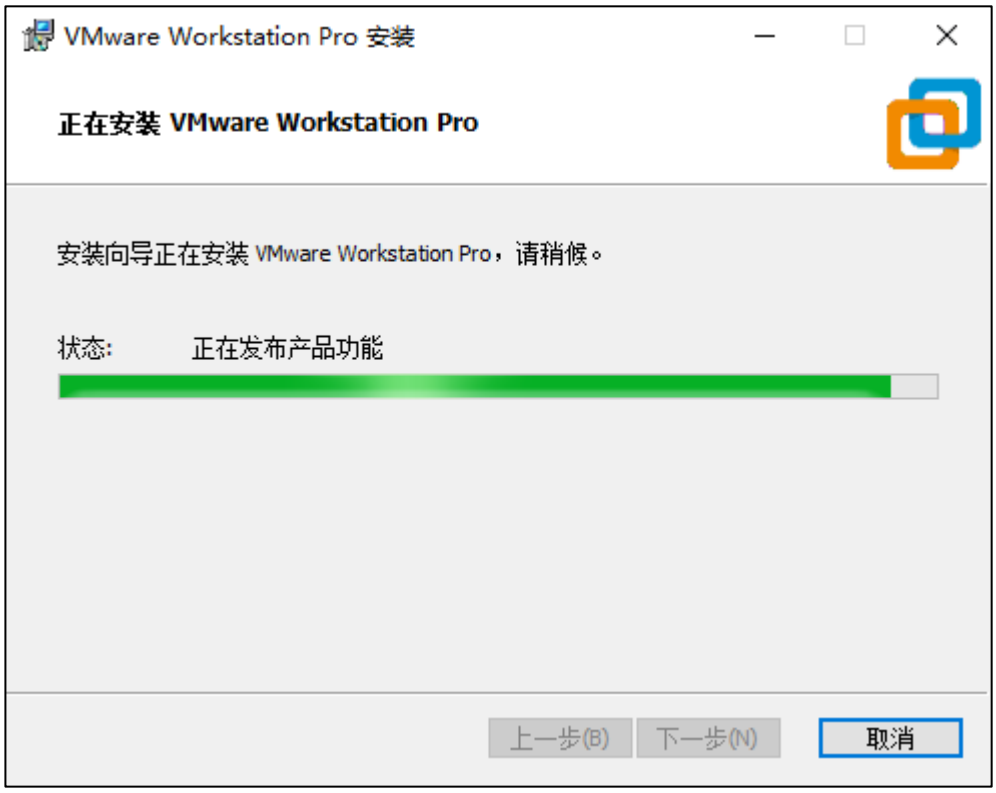


Figure 2.1-12 Install VMware software 7

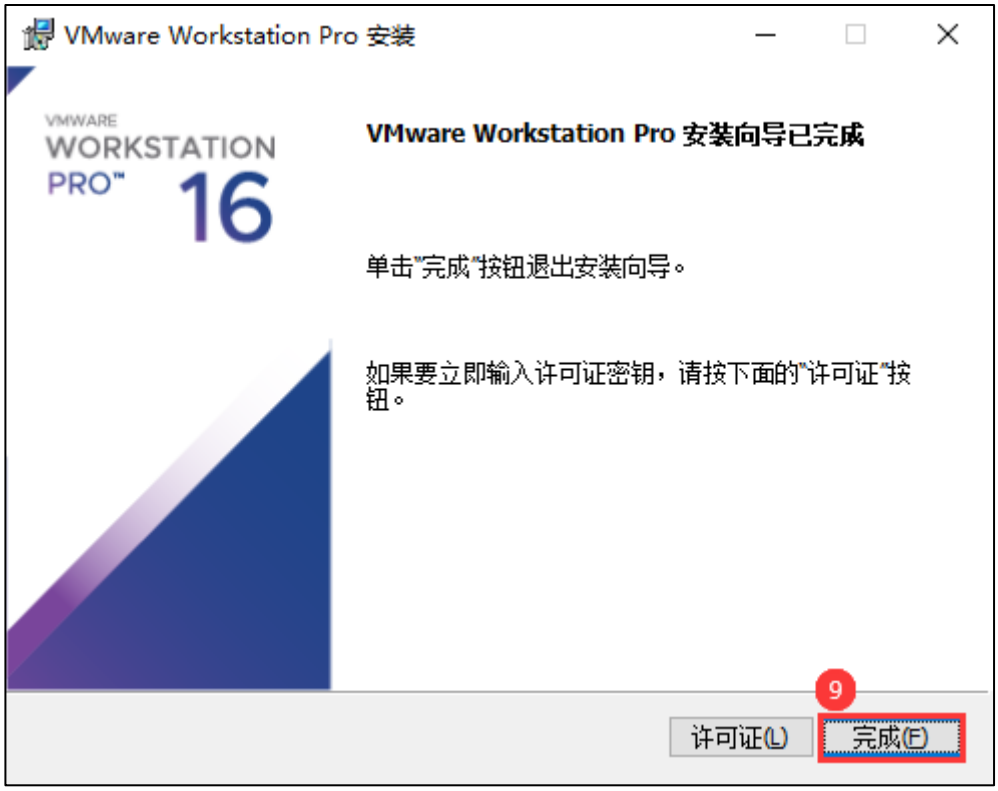


Figure 2.1-13 Install VMware software version 8

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

After the installation is finished, a shortcut icon of the VMware virtual machine software will be generated on the Windows system desktop, as shown in Figure 2.1.14:

Figure 2.1-14 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 2.1.15:

Figure 2.1-15 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 license, they will receive a series of license keys. Fill in the license key in the corresponding input box labeled "I have the license key (H) for VMware Workstation 16" to activate it. Once activation is successful, the VMware software can be used normally.

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

Figure 2.1-16 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 2.1.17:

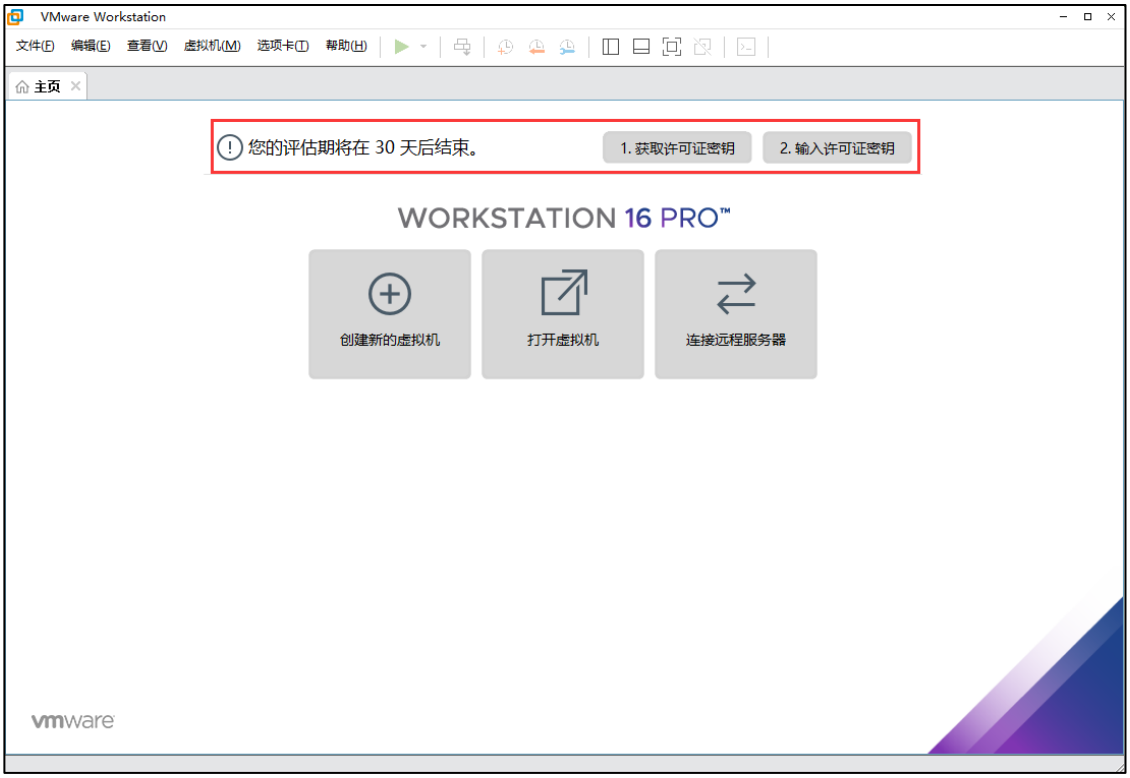


Figure 2.1-17 VMware virtual machine software main interface

2.2 Installing the Ubuntu system

This section introduces to users how to install the Ubuntu operating system in the VMware virtual machine. We recommend that users use Ubuntu 20.04 (we strongly suggest that everyone use version 20.04. The documents, videos we provide, including those used by developers during the development process, are all in Ubuntu 20.04 version). Do not use other Ubuntu versions. For users using other Ubuntu versions, you can refer to this document for comparison. If there are any problems, please solve them by yourself!

2.2.1 Download Ubuntu System Image

The Ubuntu 20.04 system image file has been provided to users in the development board data disk. The path is: Development Board Optical Disk - Disk A - Basic Materials -> 03_softwares -> ubuntu-20.04.2-desktop-amd64.iso. If the user does not want to waste time downloading, they can directly use the Ubuntu system image provided in the data disk and proceed to the next section 2.2.2.

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

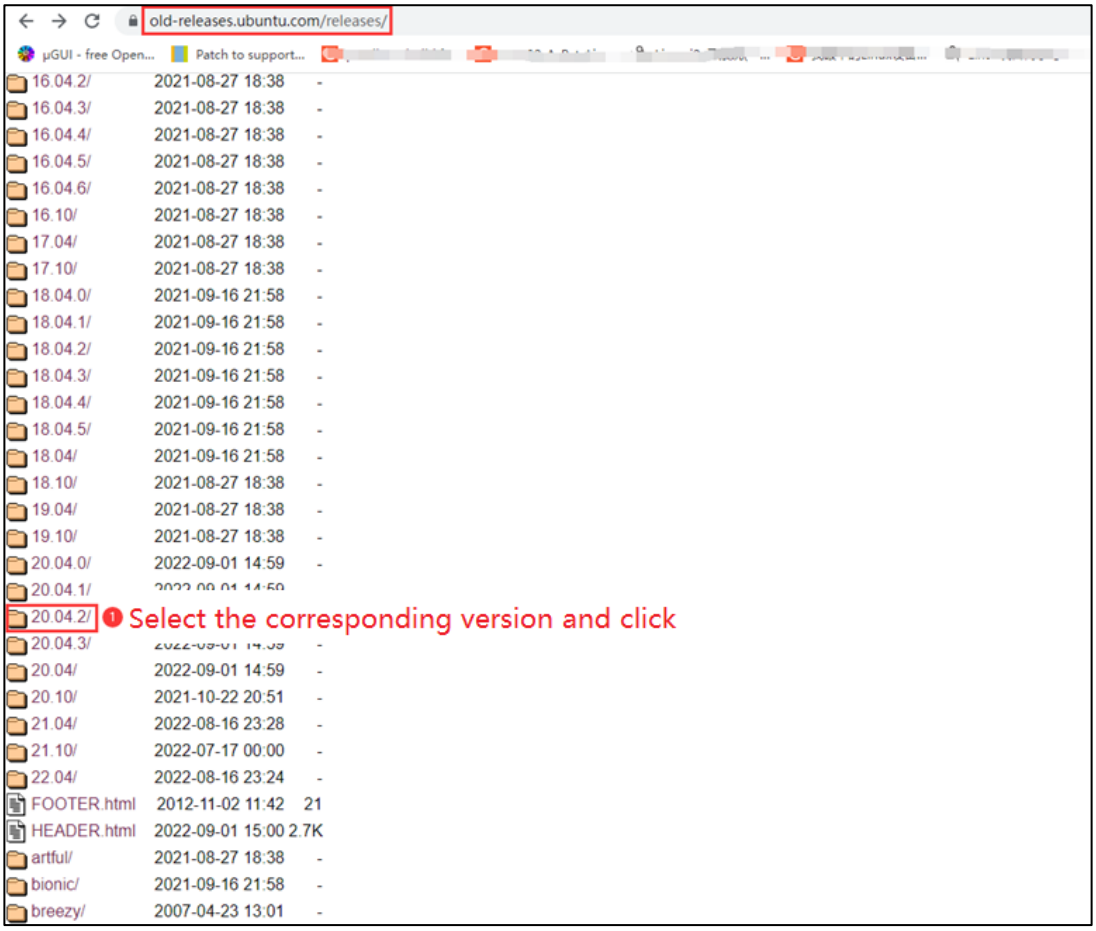


Figure 2.2-1 Download the Ubuntu system image 1

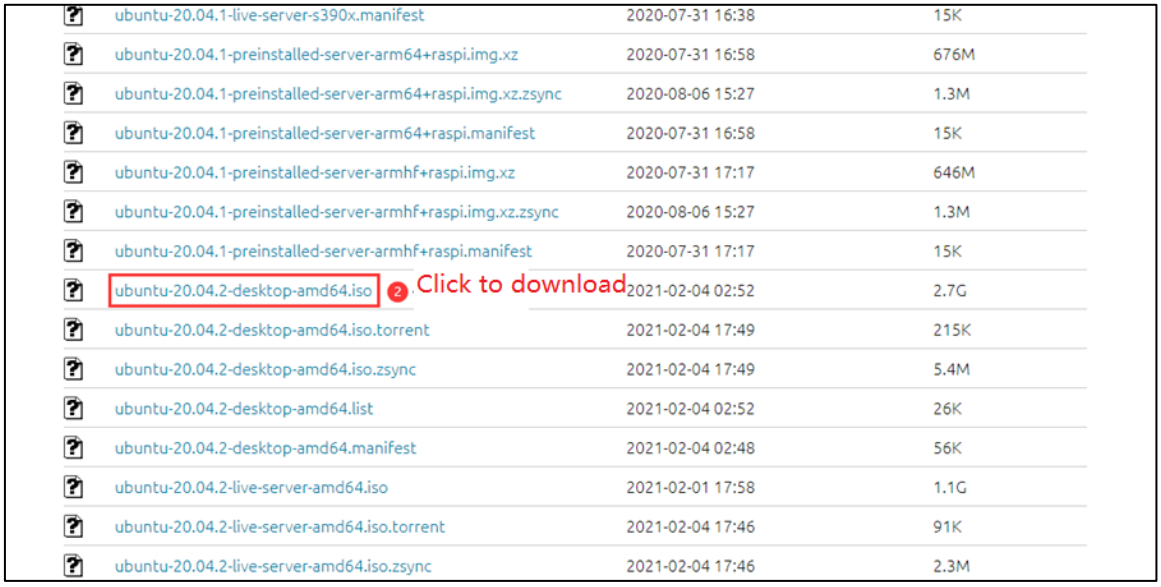


Figure 2.2-2 Download the Ubuntu system image 2

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

Figure 2.2-3 Ubuntu 20.04.2 system image file

2.2.2 Creating a virtual machine

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

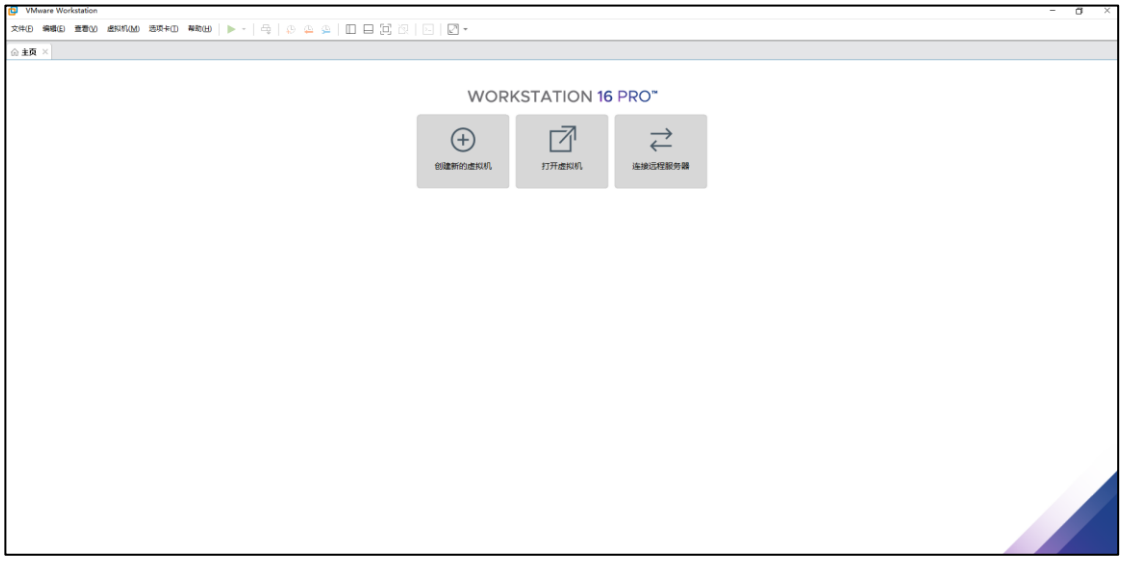


Figure 2.2-4 VMware virtual machine software

Then, follow the operation steps shown in Figures 2.2.5 to 2.2.20 to create a new virtual machine:

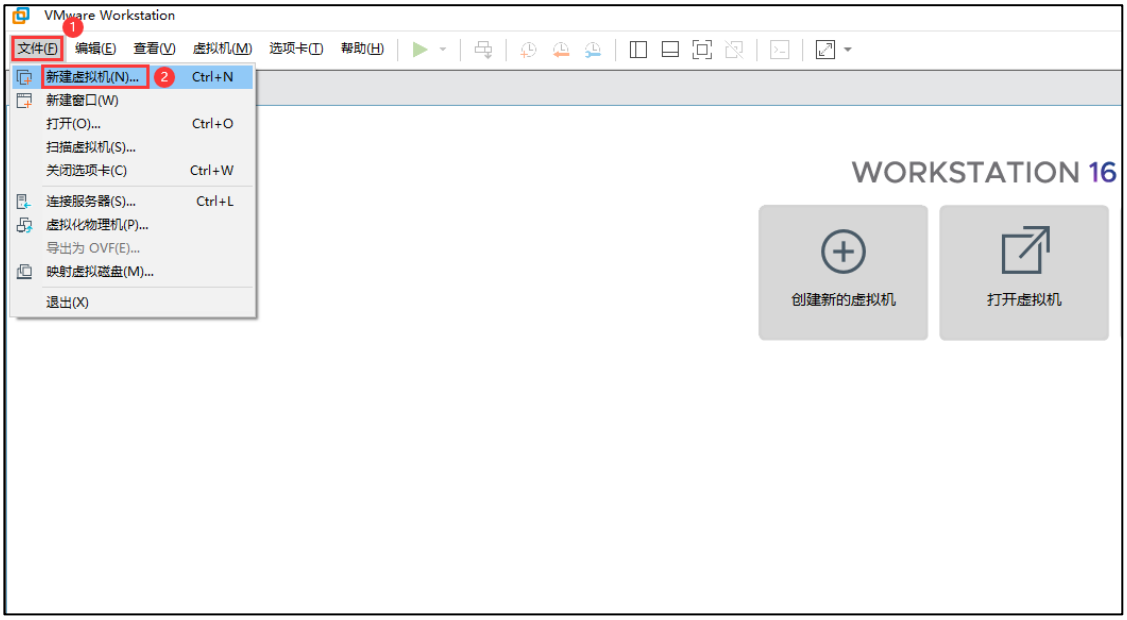


Figure 2.2-5 Create a virtual machine (1)

Figure 2.2-6 Create a virtual machine (2)

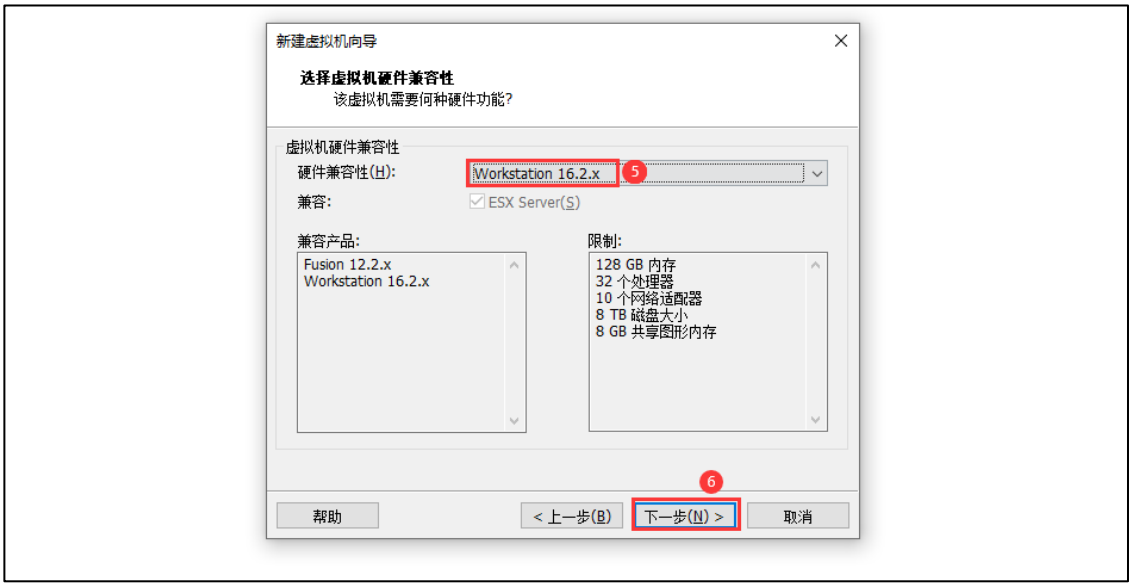


Figure 2.2-7 Create a virtual machine (3)

Figure 2.2-8 Create a virtual machine (4)

Figure 2.2-9 Create a virtual machine (5)

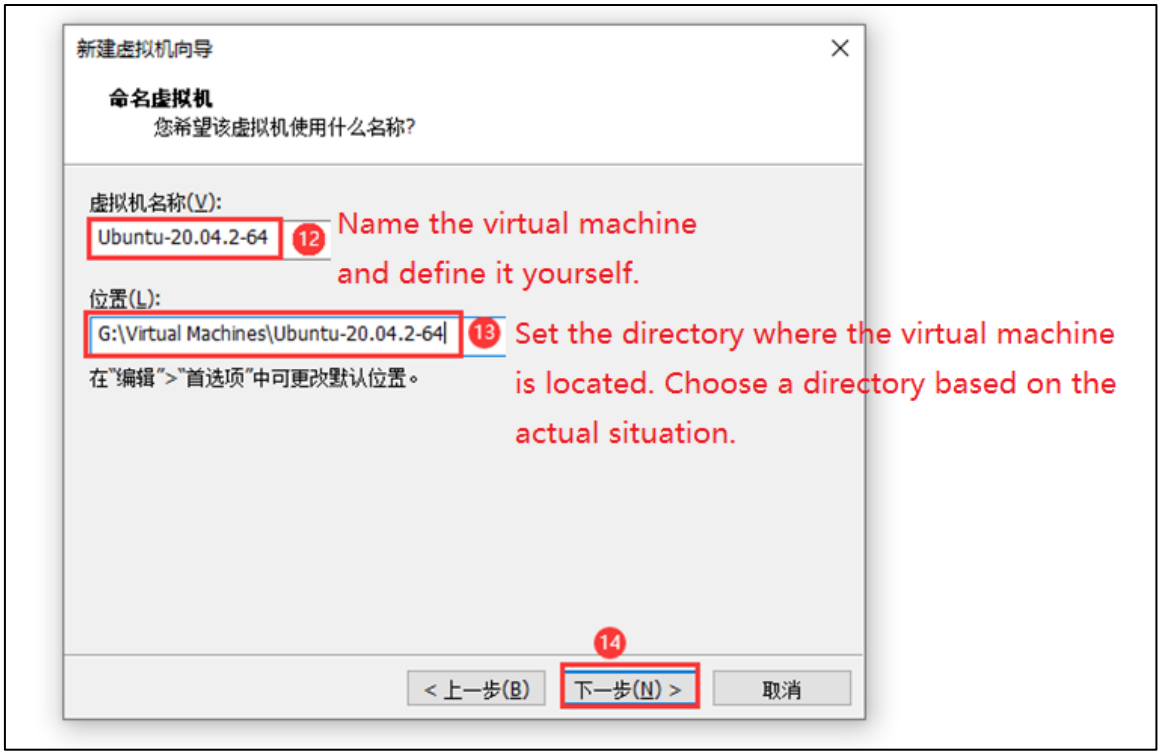


Figure 2.2-10 Create a virtual machine (6)

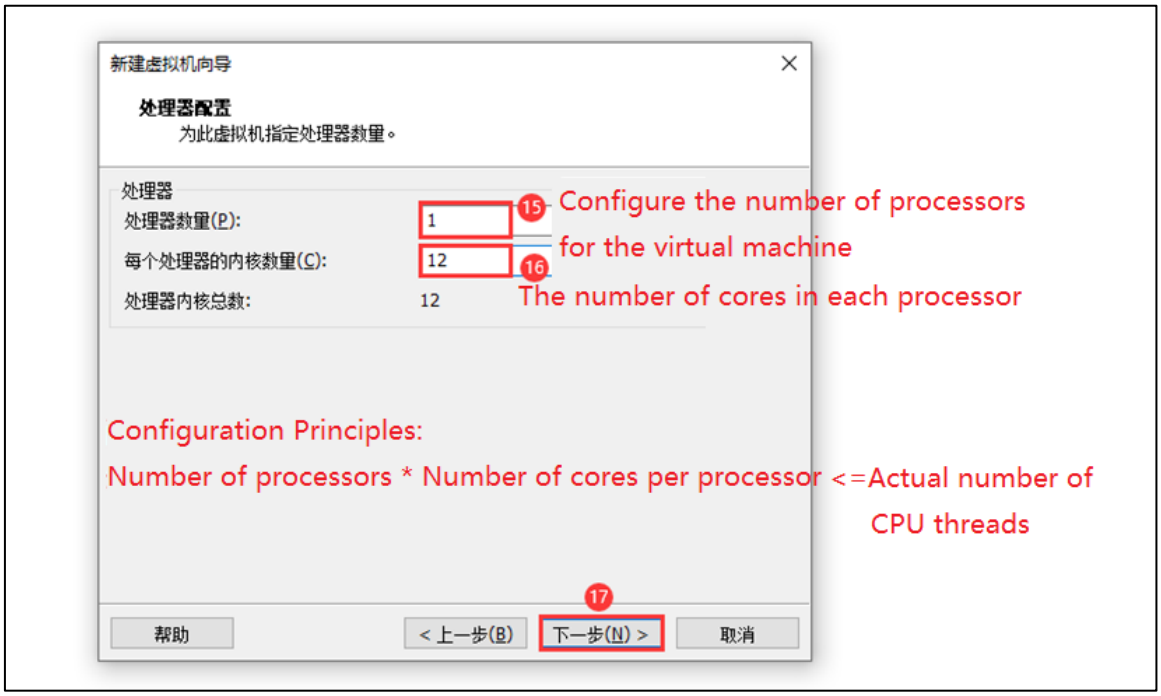


Figure 2.2-11 Create a virtual machine (7)

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 2.2.12:

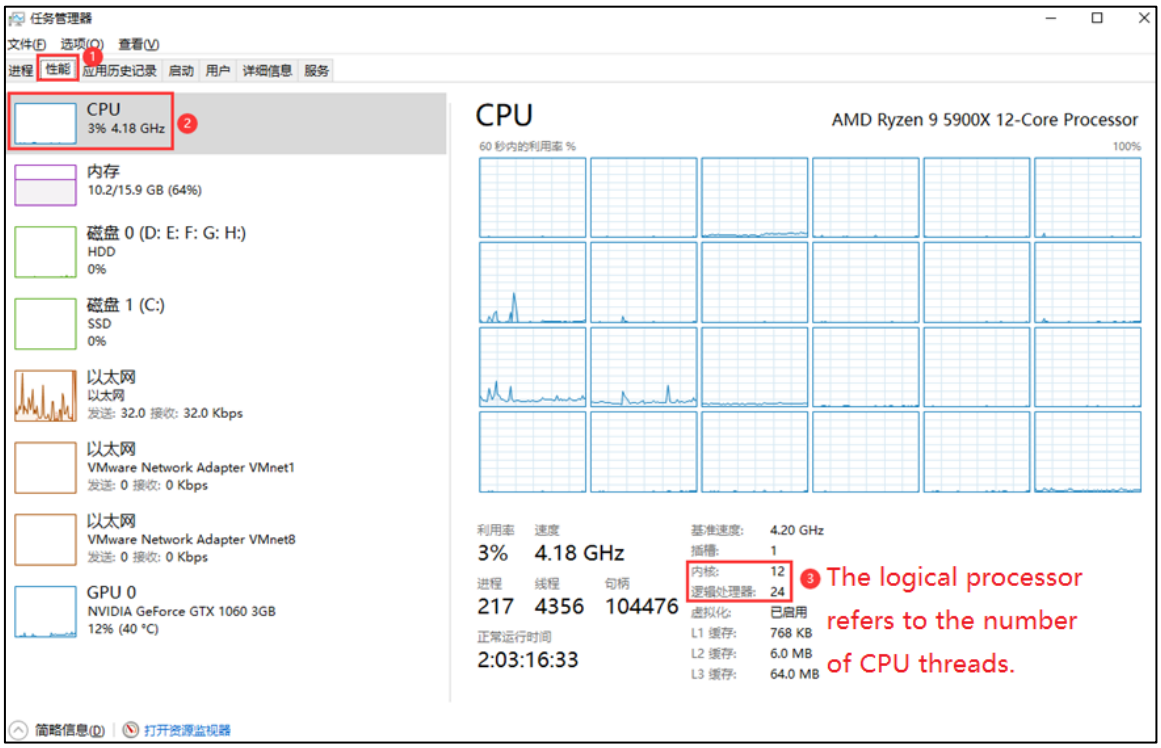


Figure 2.2-12 Query the CPU configuration

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

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

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

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

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

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

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

Processor quantity = 8, each processor's core quantity = 2;

And so on...

In summary, the specific configuration methods are diverse. As long as the requirements are met, any method is acceptable!

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

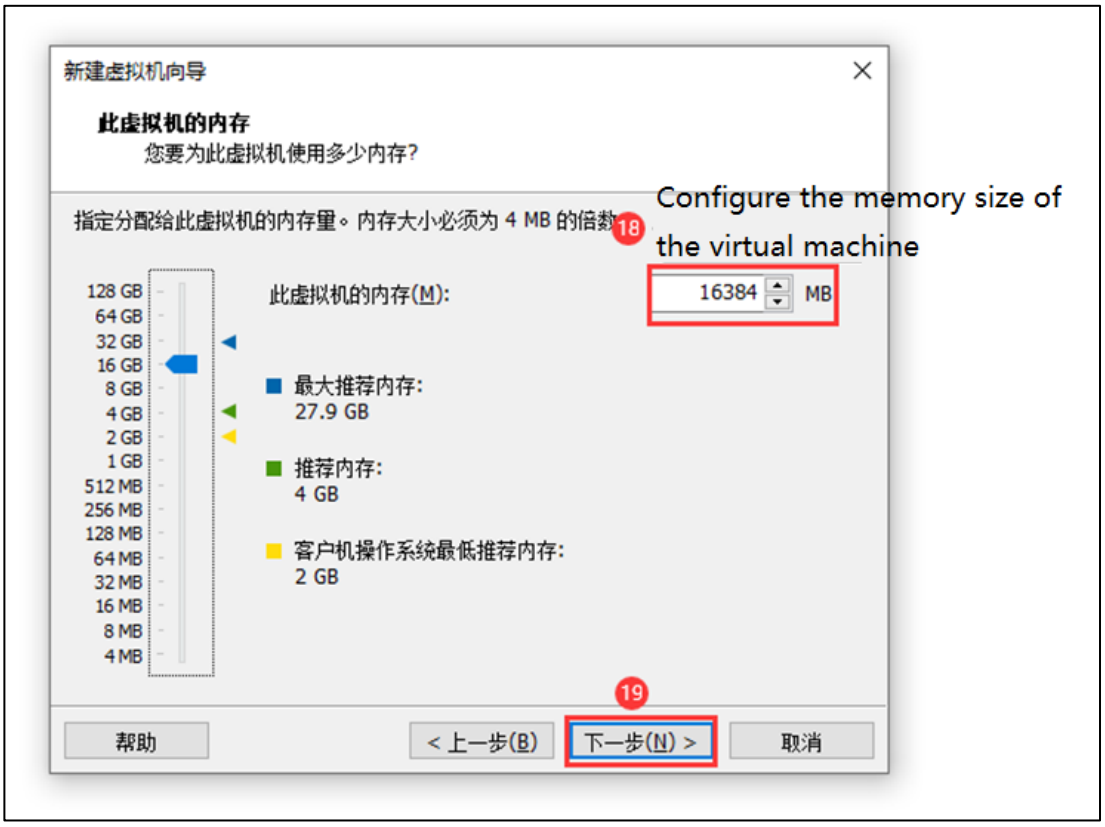


Figure 2.2-13 Creating Virtual Machine (8)

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 development of QuarkPi-CA2 embedded Linux system, it is recommended to allocate a memory size to the virtual machine that is at least greater than or equal to 16GB, or even larger.

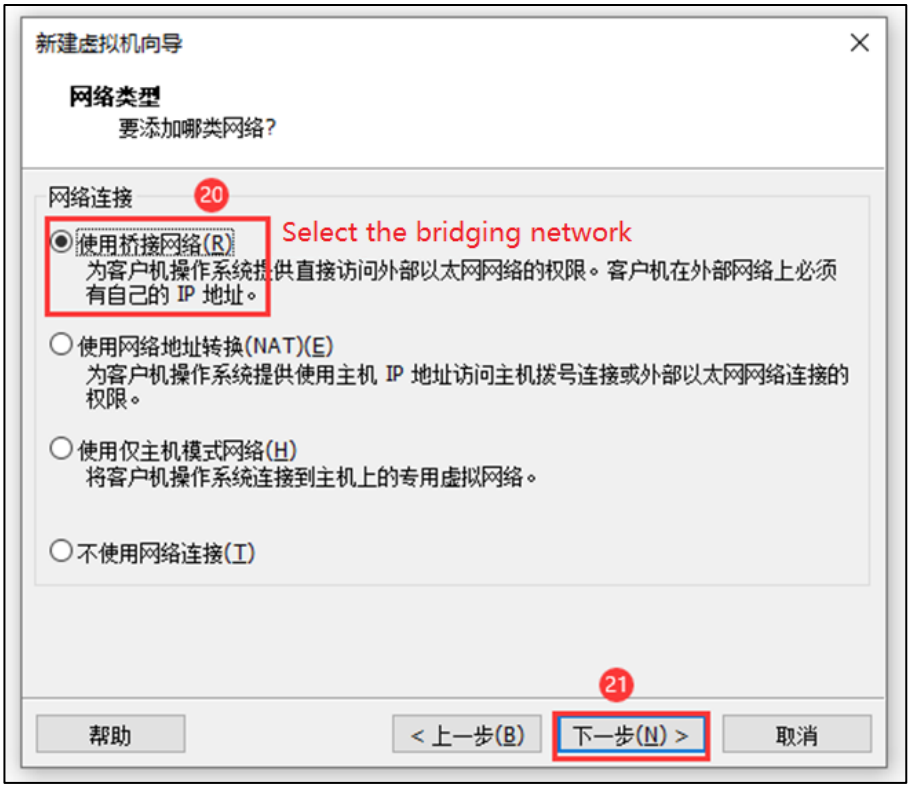


Figure 2.2-14 Create virtual machine (9)

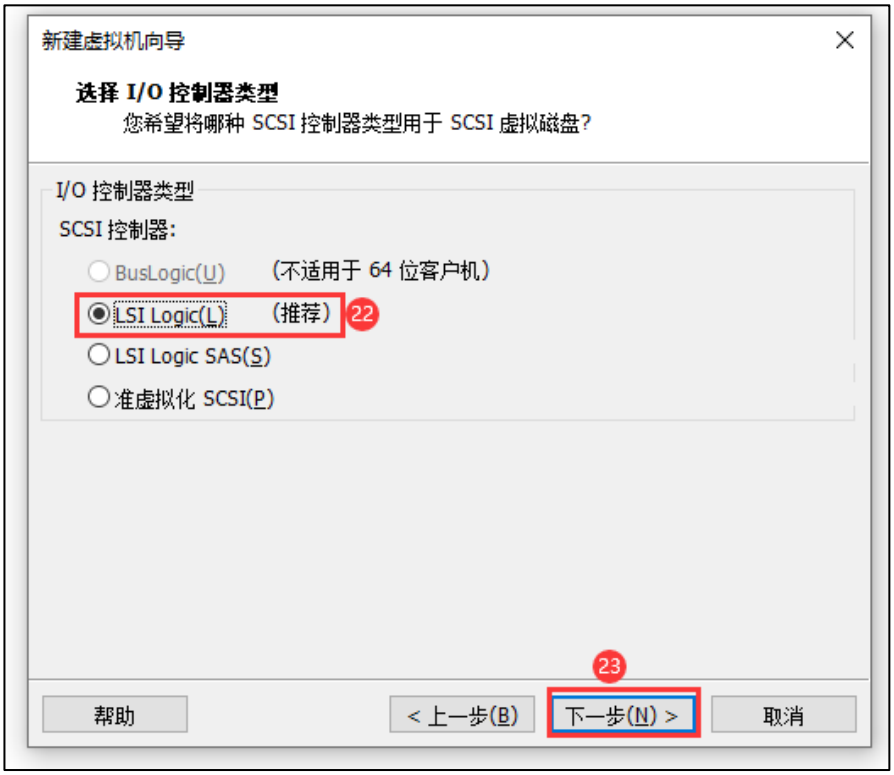


Figure 2.2-15 Create virtual machine (10)

Figure 2.2-16 Create virtual machine (11)

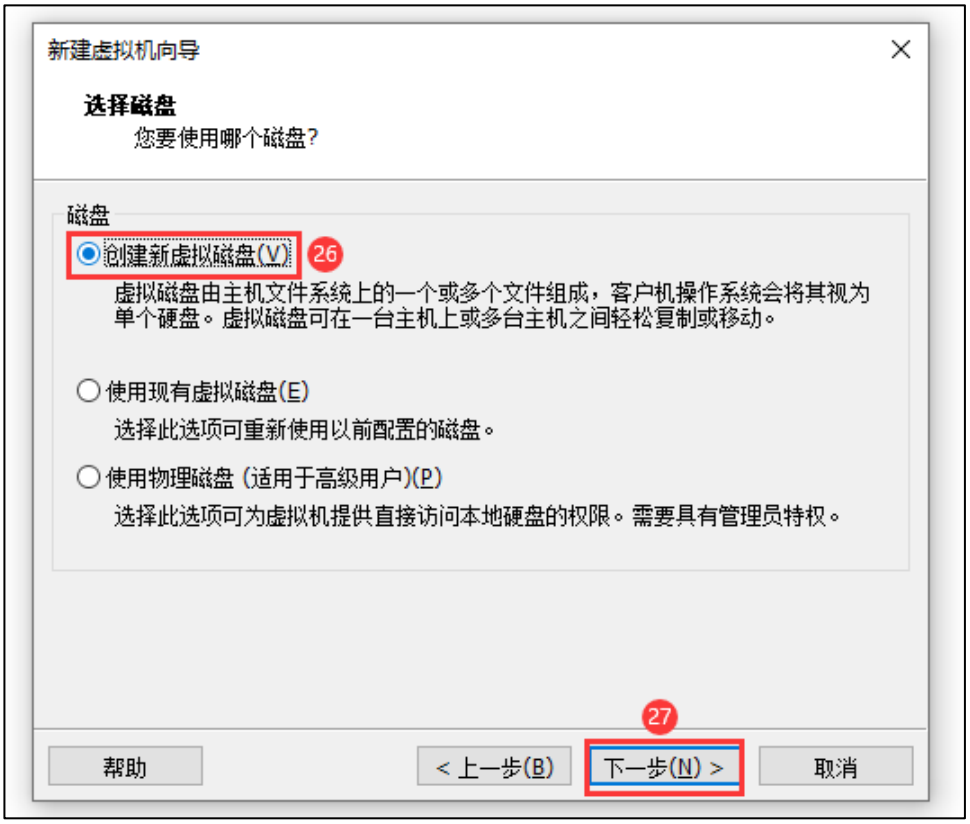


Figure 2.2-17 Create virtual machine (12)

Figure 2.2-18 Create virtual machine (13)

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

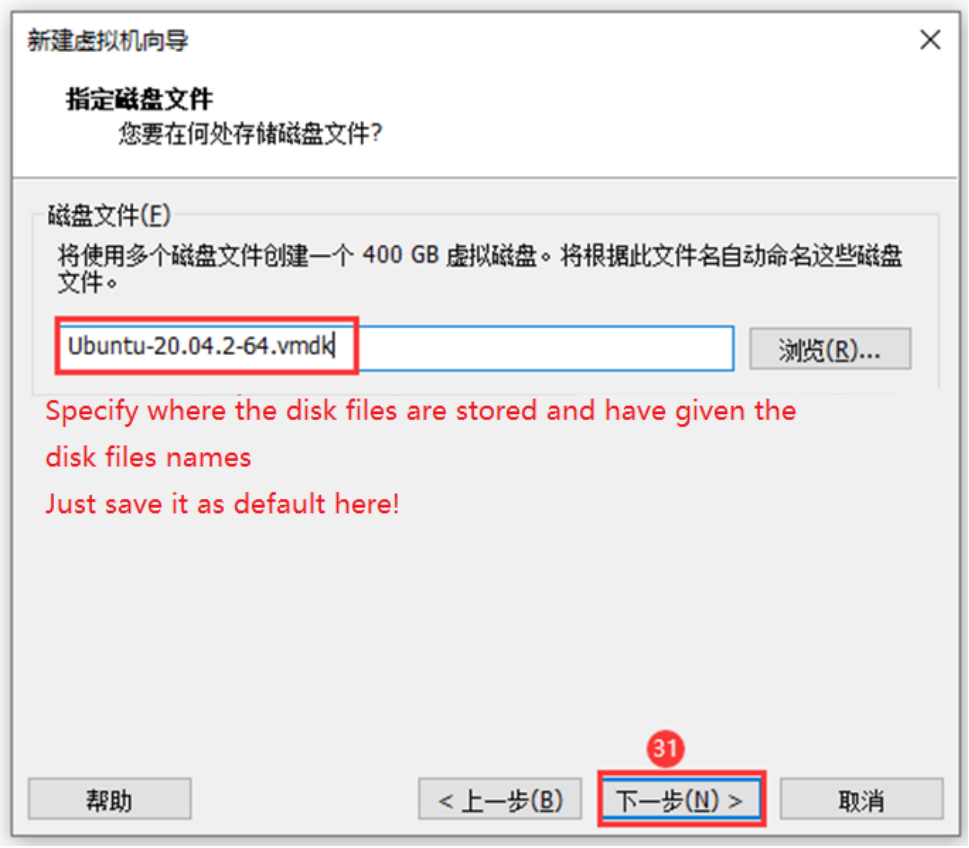


Figure 2.2-19 Create virtual machine (14)

Figure 2.2-20 Create virtual machine (15)

At this point, the virtual machine has been successfully created, as shown in Figure 2.2.21:

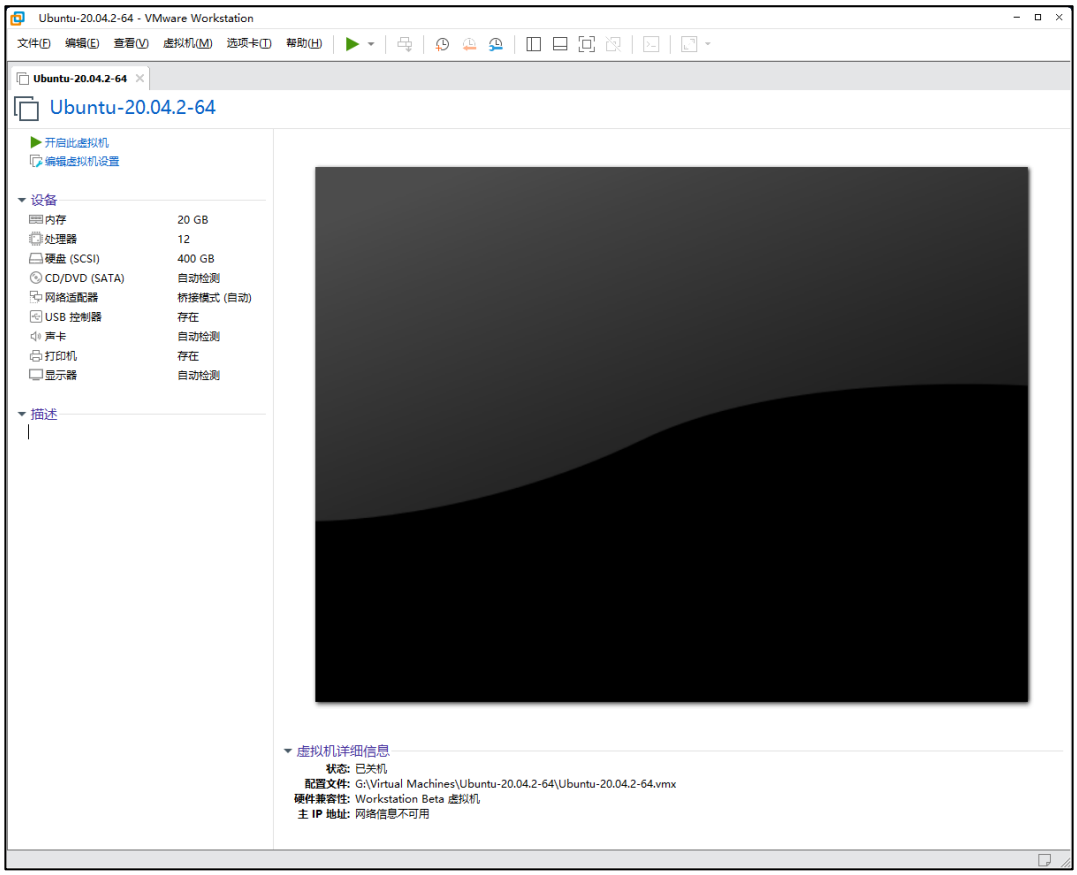


Figure 2.2-21 The virtual machine has been created successfully.

2.2.3 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:

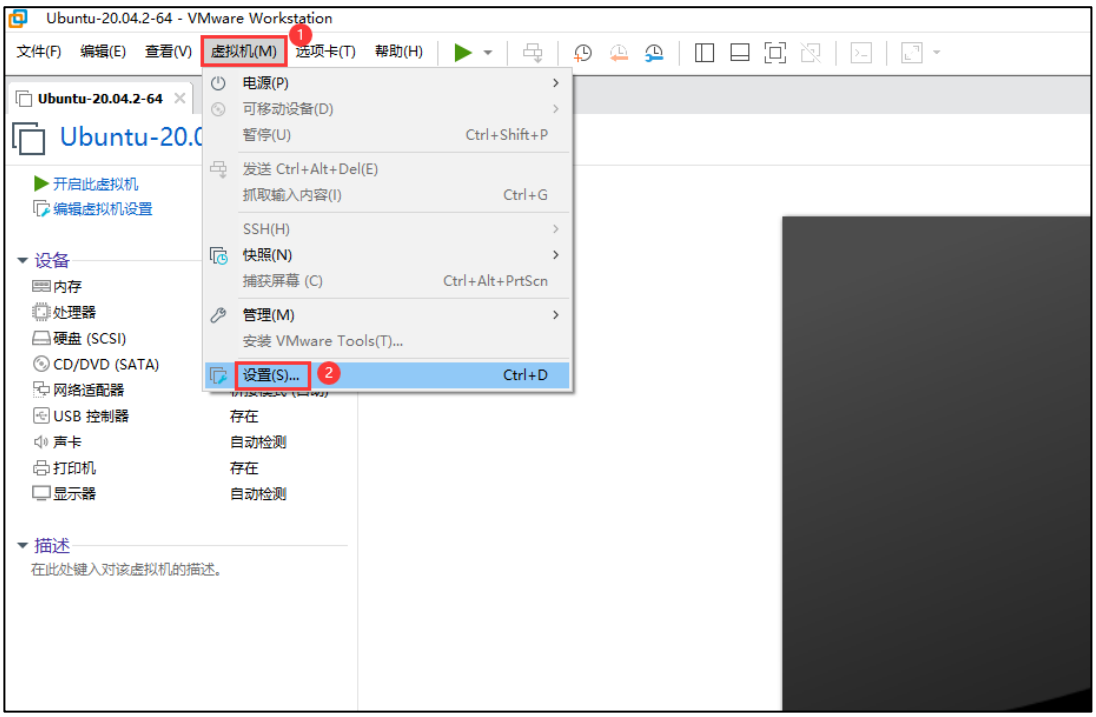


Figure 2.2-22 Open the virtual machine settings interface

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

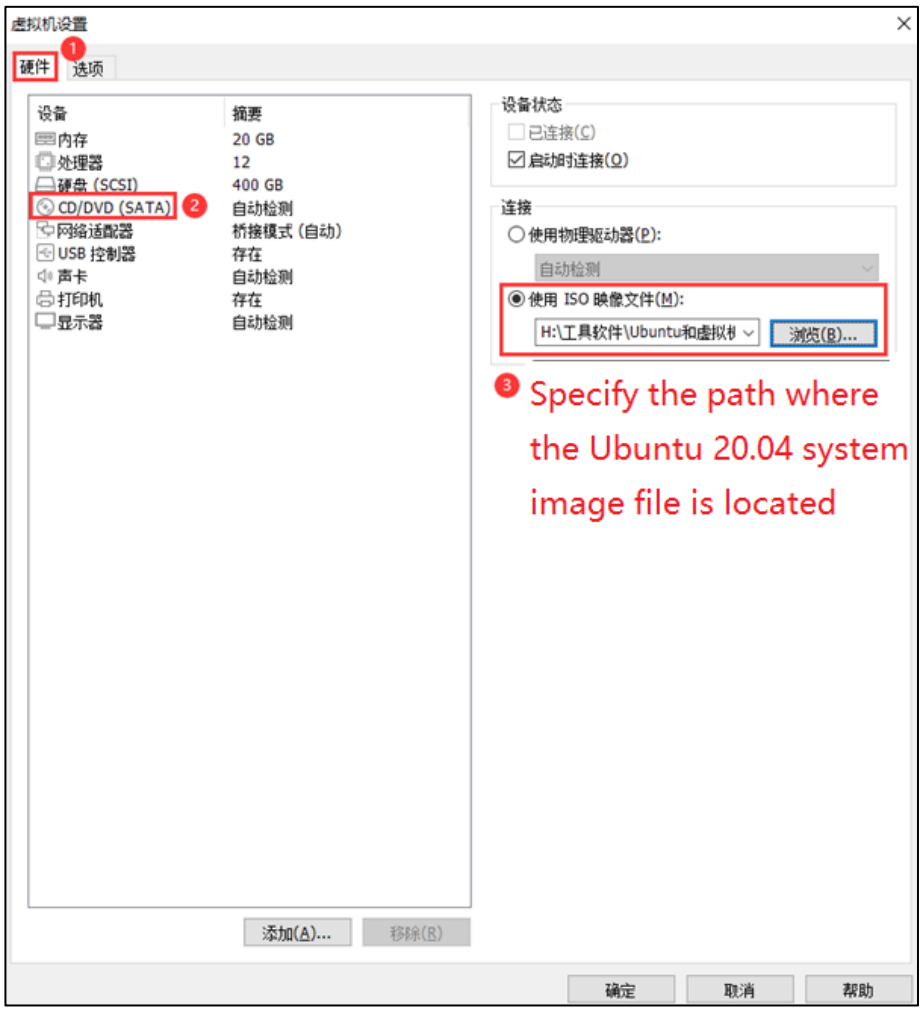


Figure 2.2-23 Specify the Ubuntu system image

Click the "Browse (B)" button, 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 2.2-24 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 2.2.25 to 2.2.35 to install the Ubuntu system:

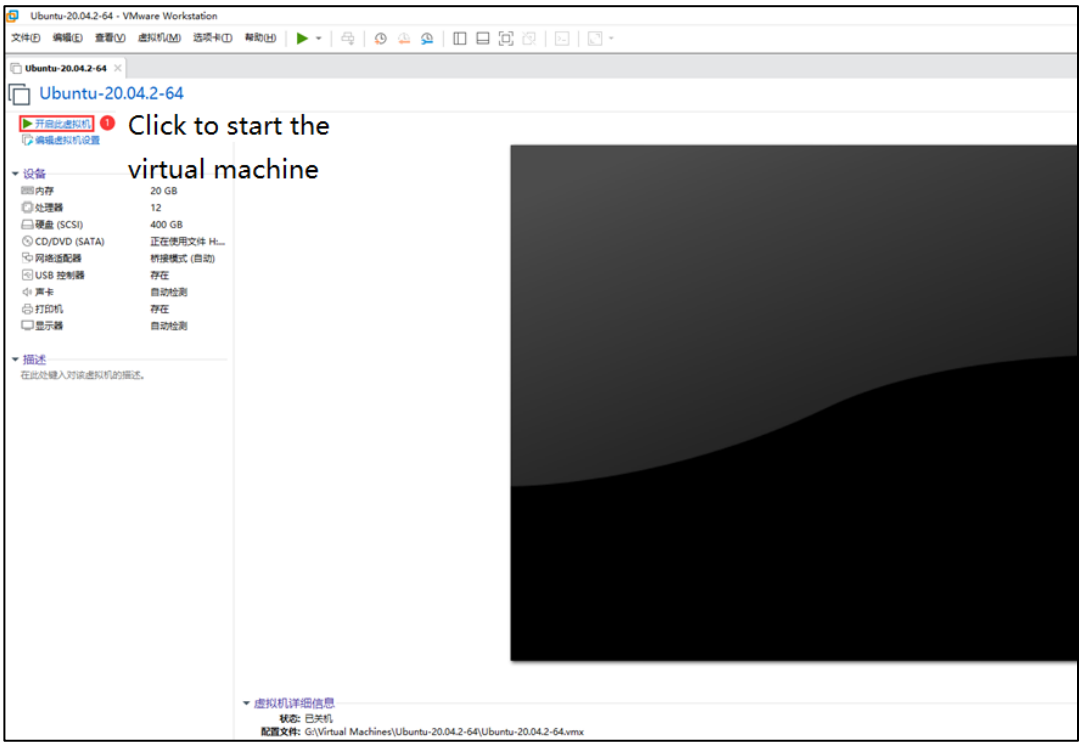


Figure 2.2-25 Install the Ubuntu operating system (1)

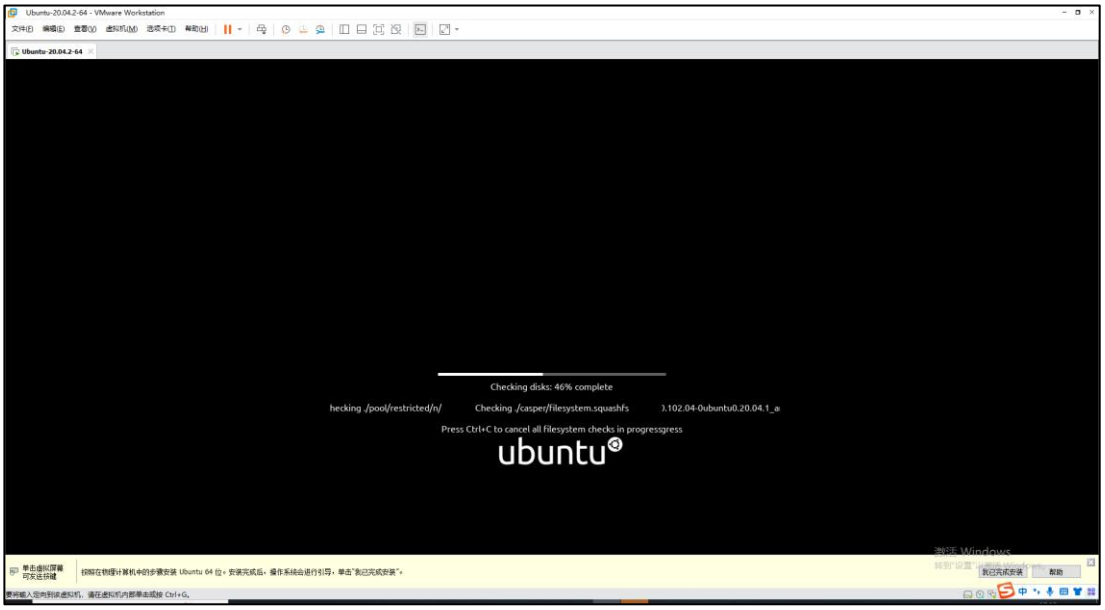


Figure 2.2-26 Install the Ubuntu operating system (2)

Nov 26 09:13
安装
欢迎
ગુજરાતી
தமிழ்
తెలుగు
ಕನ್ನಡ
മലയാളം
සිංහල
ภาษาไทย
ลาว
ᨅᨔᨘᨁᨗ
ᨕᨘᨁᨗᨕᨘᨁᨗᨕᨘᨁᨗ

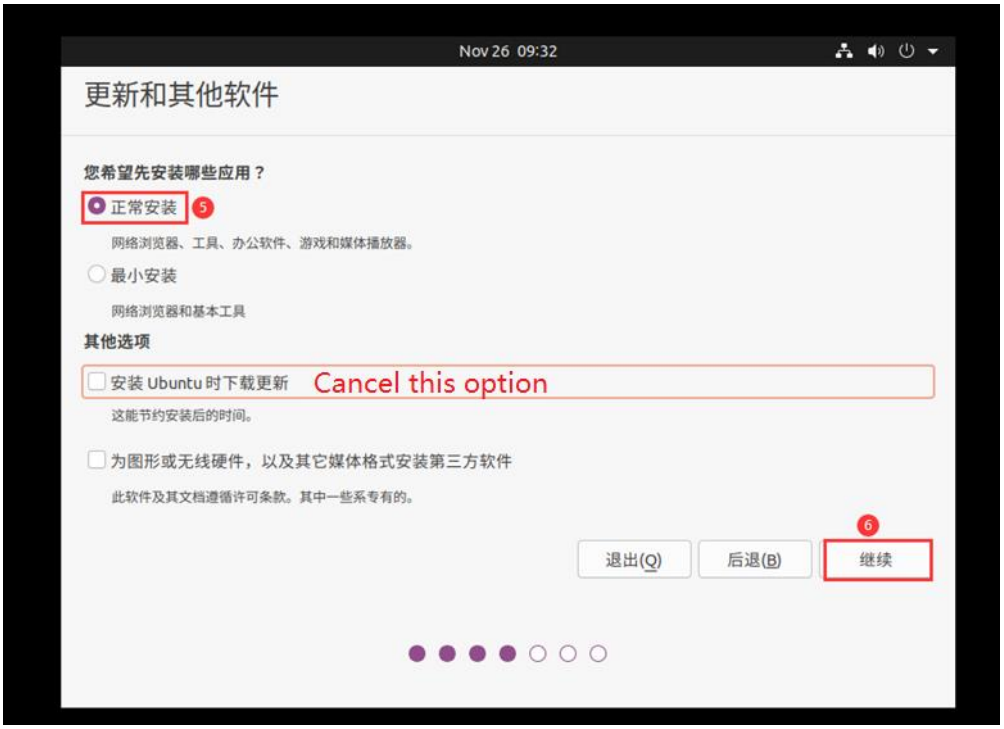


Figure 2.2-29 Install the Ubuntu operating system (5)

Figure 2.2-30 Install the Ubuntu operating system (6)

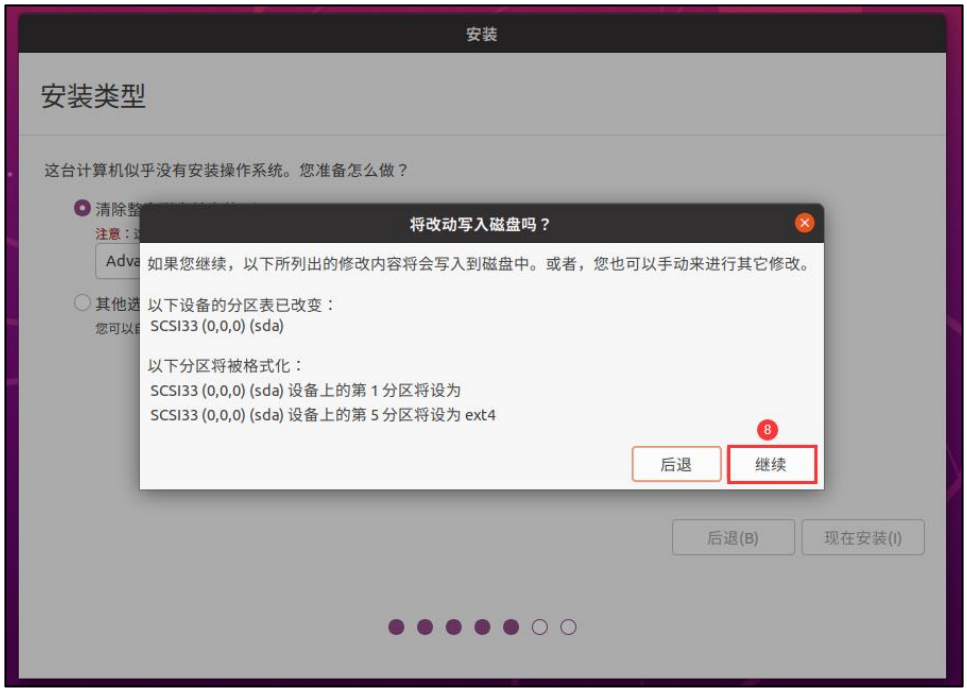


Figure 2.2-31 Install the Ubuntu operating system (7)

Figure 2.2-32 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.

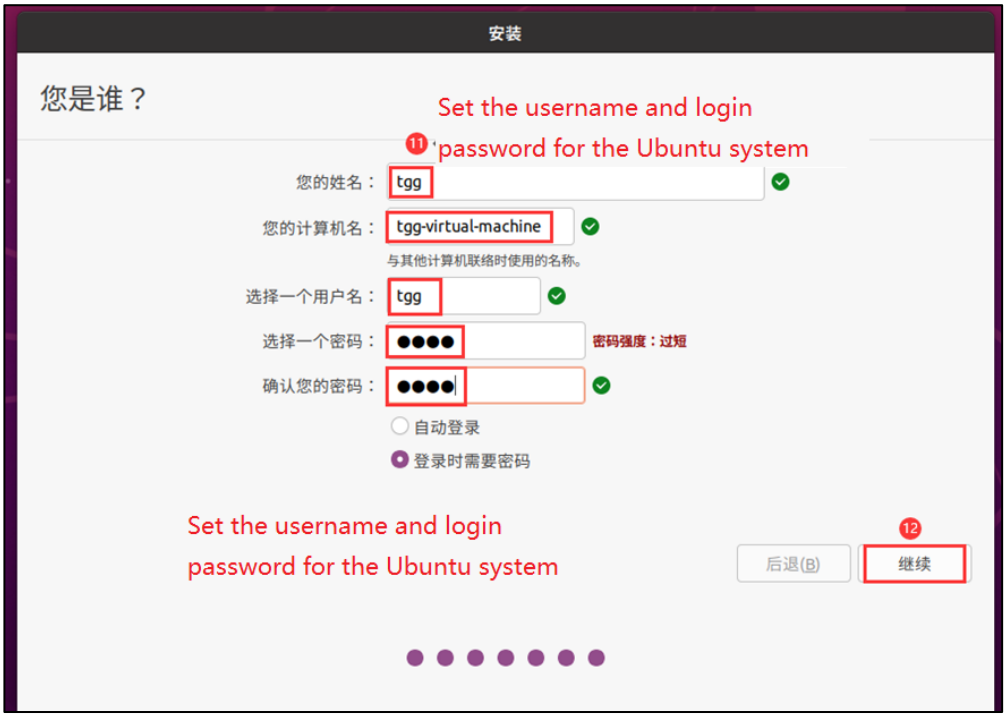


Figure 2.2-33 Install the Ubuntu operating system (9)

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

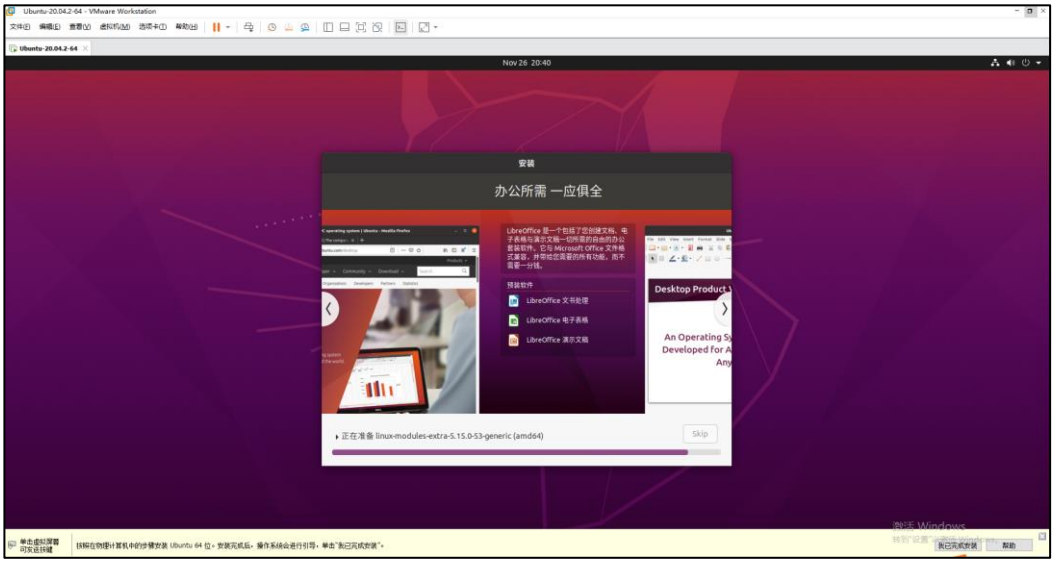


Figure 2.2-34 Install the Ubuntu operating system (10)

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

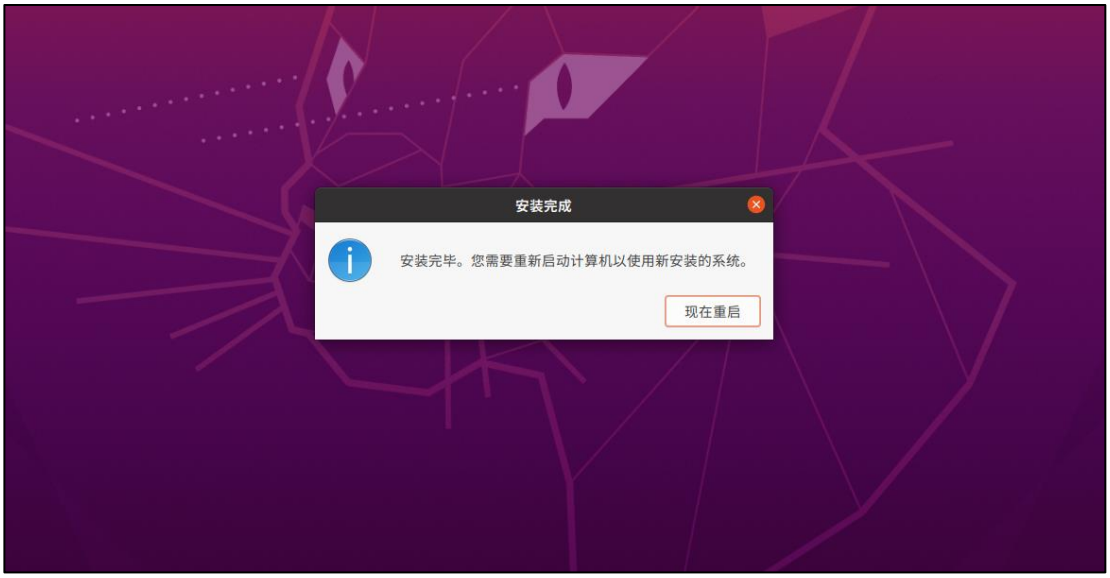


Figure 2.2-35 Ubuntu system installation is complete.

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 2.2.36 to 2.2.37 to shut down the virtual machine:

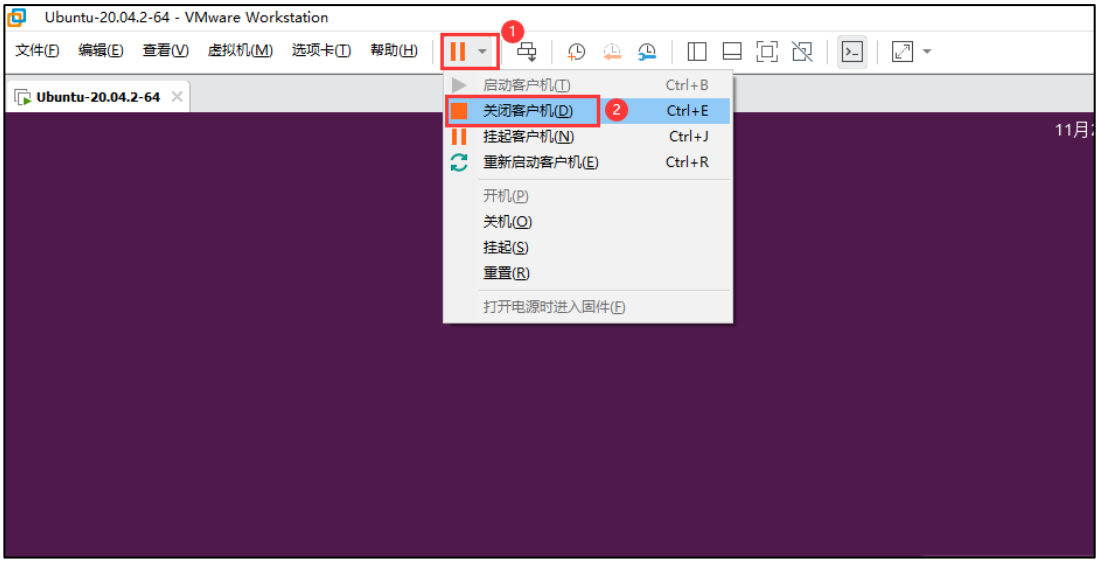


Figure 2.2-36 Shutdown the virtual machine (1)

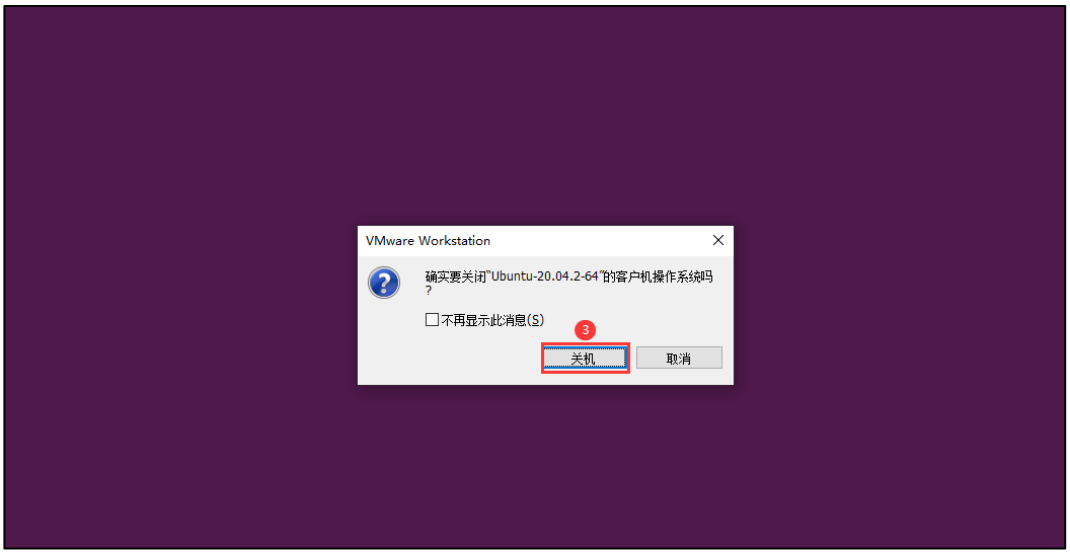


Figure 2.2-37 Shutdown the virtual machine (1)

After closing the virtual machine, click on the "Virtual Machine (M) -> Settings (S)" option in the menu bar to enter the virtual machine settings interface. Remove the Ubuntu system image file as shown in Figure 2.2.38:

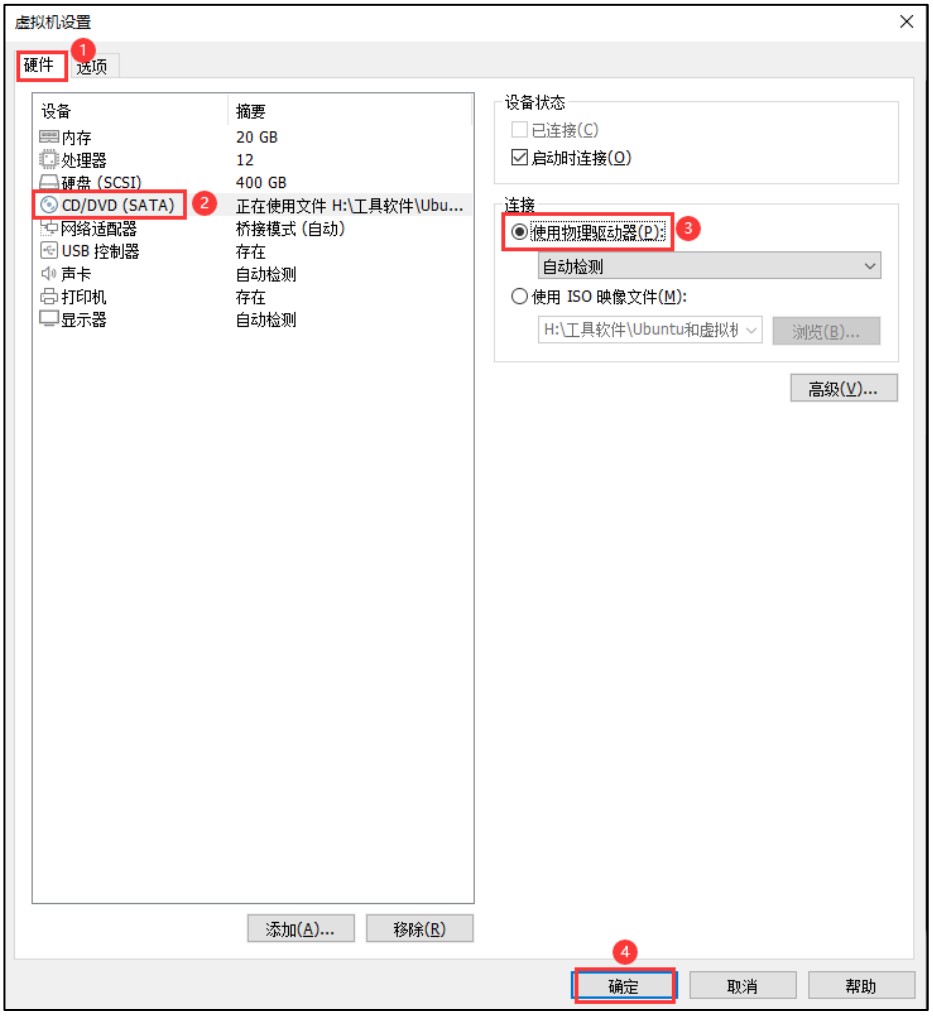


Figure 2.2-38 Remove the Ubuntu system image file.

After clicking the "OK" button, exit the settings window. Then you can start the virtual machine. Follow the steps shown in Figure 2.2.39 to start the virtual machine (that is, to boot the Ubuntu system):

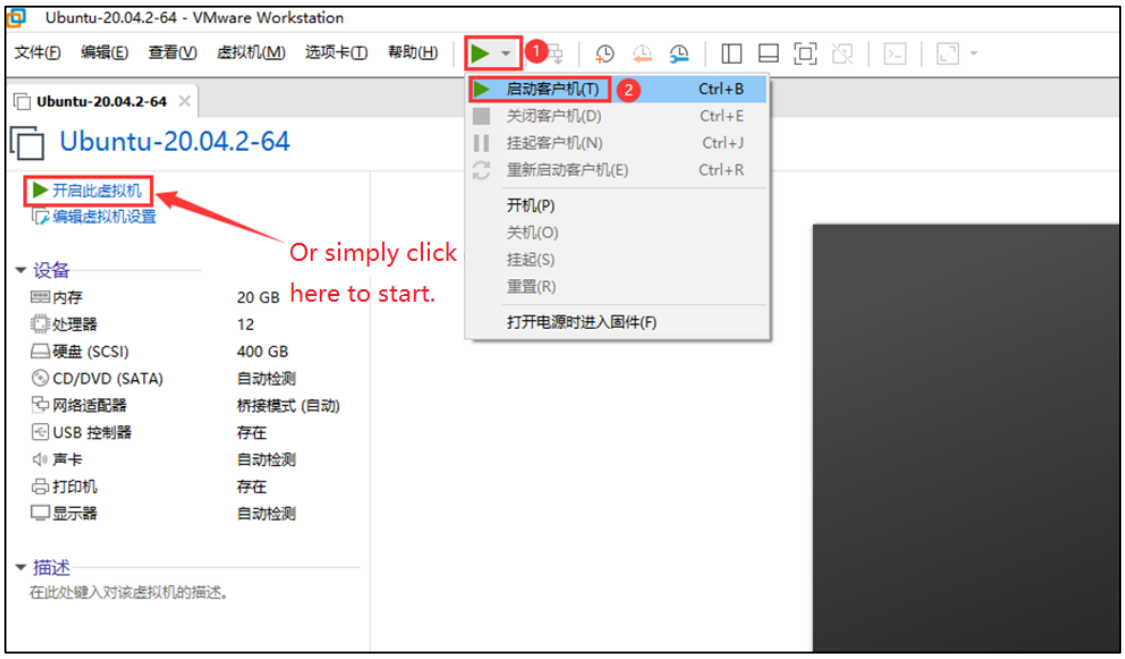


Figure 2.2-39 Start the virtual machine

After the system boots up, it will look like Figure 2.2.40:

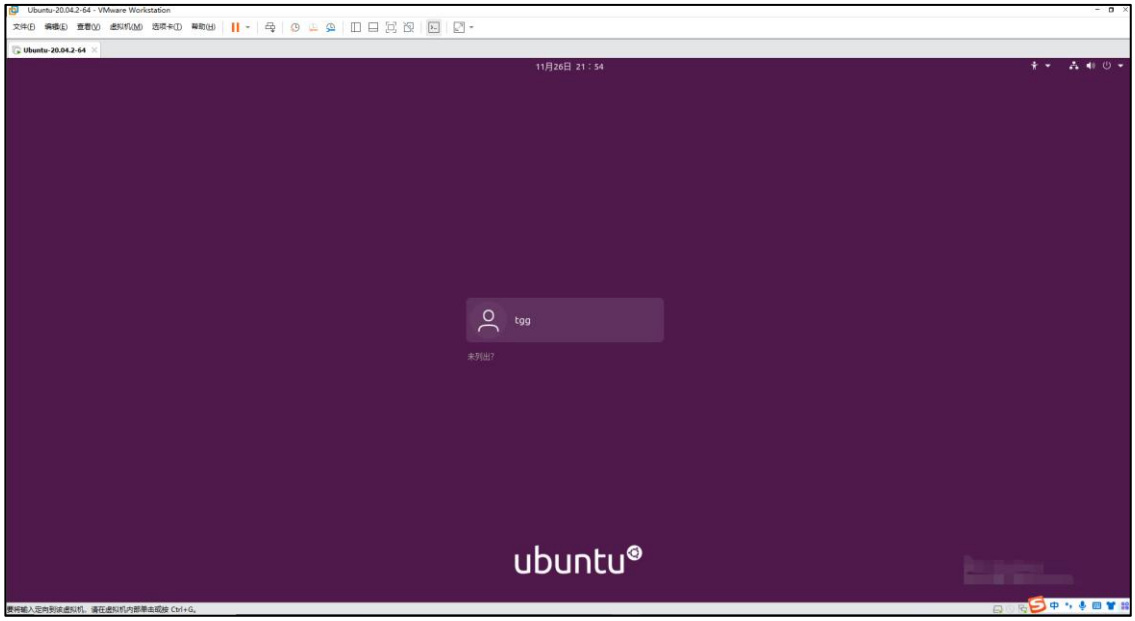


Figure 2.2-40 User login interface

Then click on the username, enter the password and press the Enter key to log in to the system and enter the main interface of the Ubuntu system:

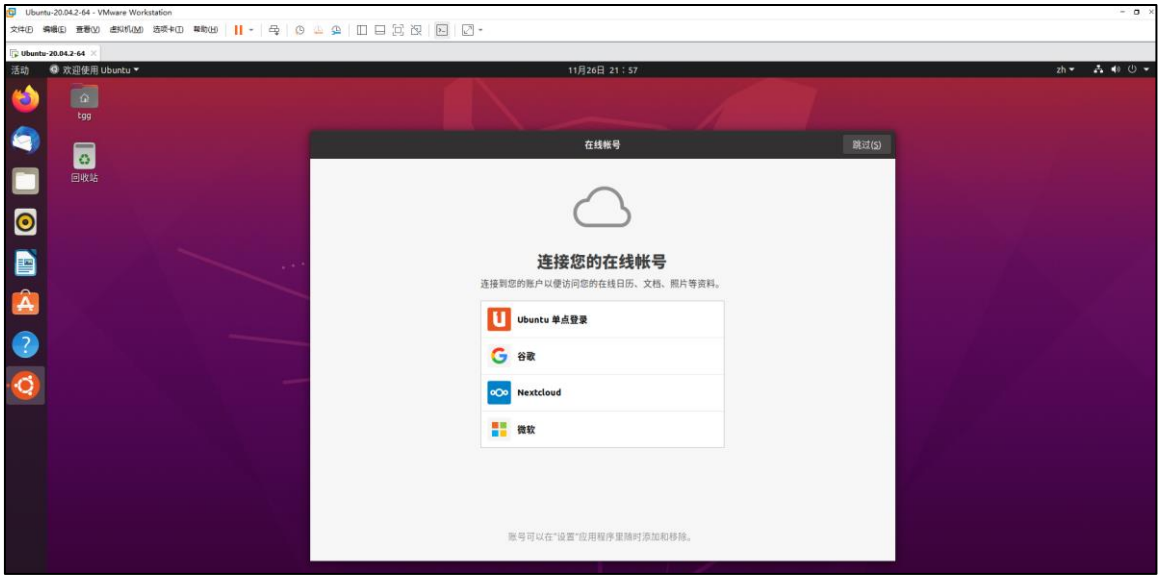


Figure 2.2-41 Ubuntu system main interface

When you first enter the Ubuntu 20.04 system, a "Online Account" window (which is a simple setup guide) will pop up on the desktop, as shown in Figure 2.2.41; you can follow the instructions step by step or simply exit, as shown in Figure 2.2.42:

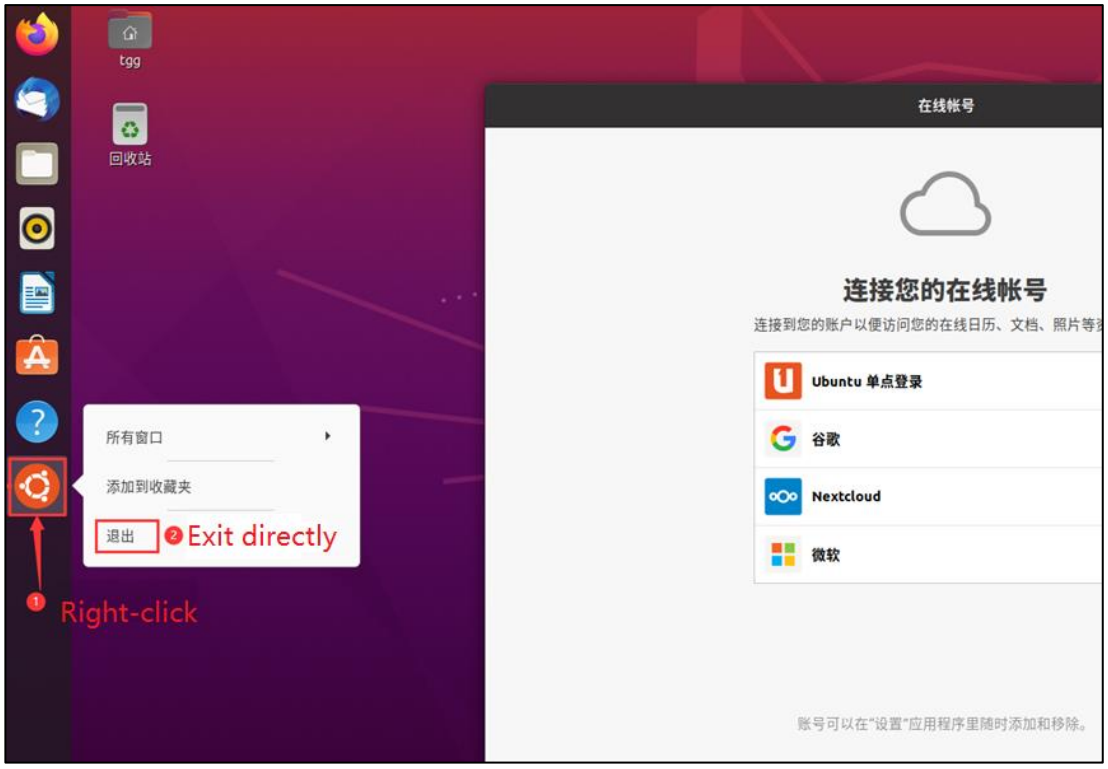


Figure 2.2-42 Exit the usage guidance window

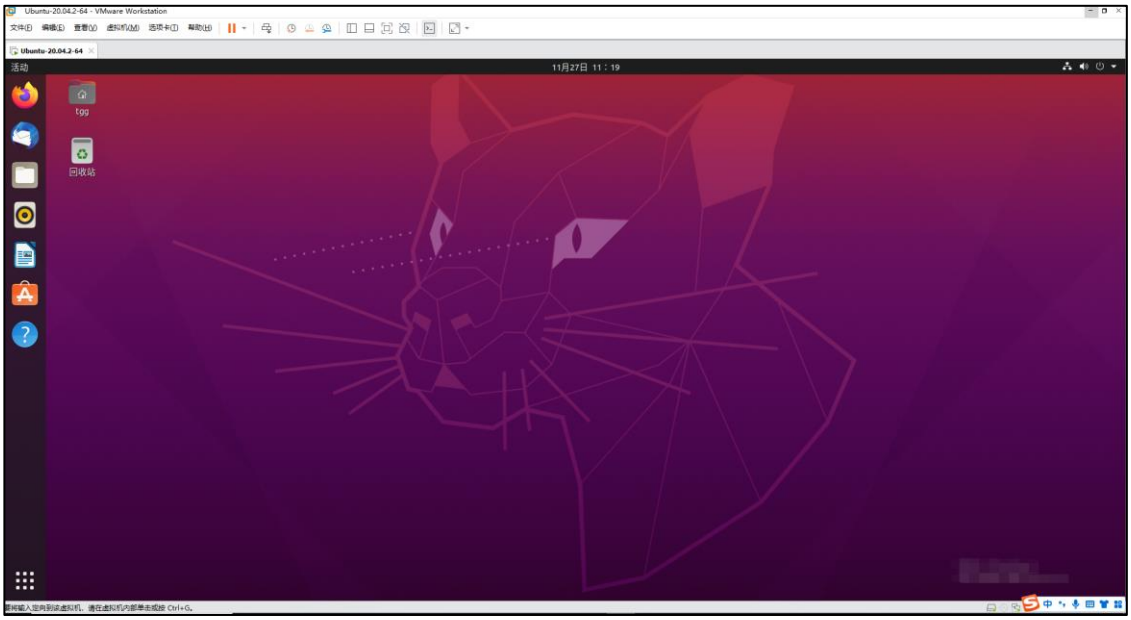


Figure 2.2-43 Ubuntu desktop

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

Chapter 3. Development Environment Setup

Before conducting the QuarkPi-CA2 embedded Linux development, we need to first set up the development environment; this includes the setup of both Windows and Ubuntu operating systems, as embedded Linux development is typically carried out in a dual-system development environment of Windows + Ubuntu.

This chapter will be divided into the following sections:

1. Ubuntu System Settings;
2. File Transfer Between Ubuntu and Windows;
3. Setting up the TFTP Server in the Ubuntu System;
4. Setting up the NFS Server in the Ubuntu System;
5. Setting up the SSH Server in the Ubuntu System;
6. Installation of the CH343 Serial Port Driver;
7. Installation of the MobaXterm Software;
8. Installation of the Rockchip USB Driver;
9. Use of the Image Burning Tool;
10. Installation of the adb Tool.

3.1 Ubuntu System Settings

This section provides a simple setup for the Ubuntu system.

3.1.1 Set the root user password

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

```
sudo passwd root
```

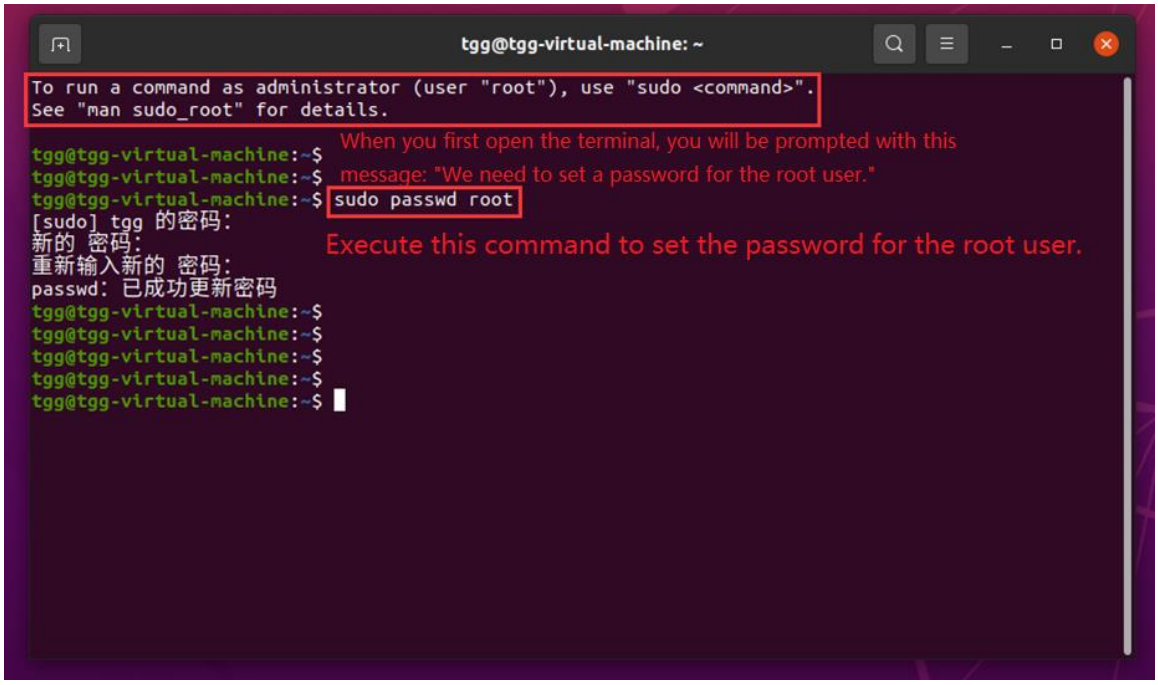


Figure 3.1-1 Set the password for the root user

3.1.2 Change the software download source

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

Follow the steps shown in Figures 3.1.2 to 3.1.9 to make the change:

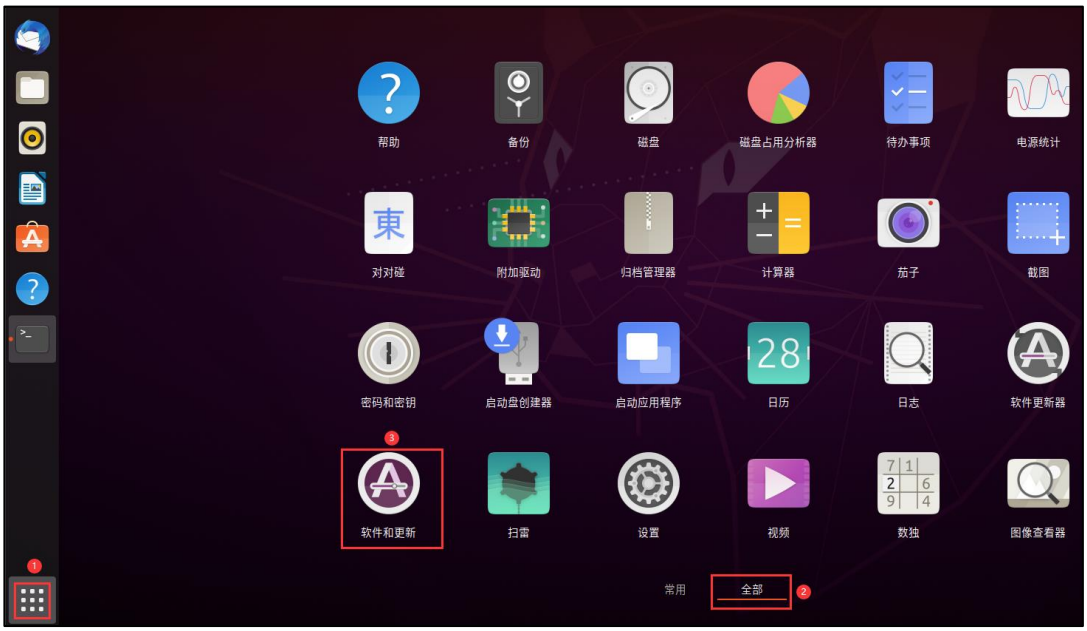


Figure 3.1-2 Change the software download source (1)

Figure 3.1-3 Change the software download source (2)

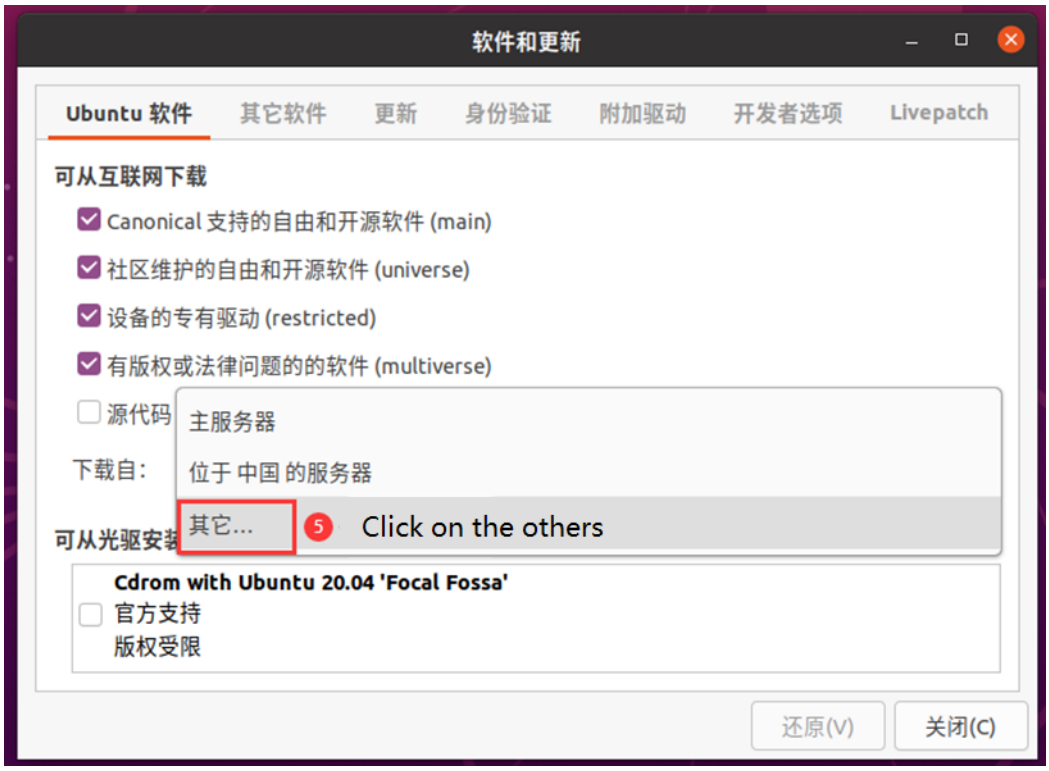


Figure 3.1-4 Change the software download source (3)

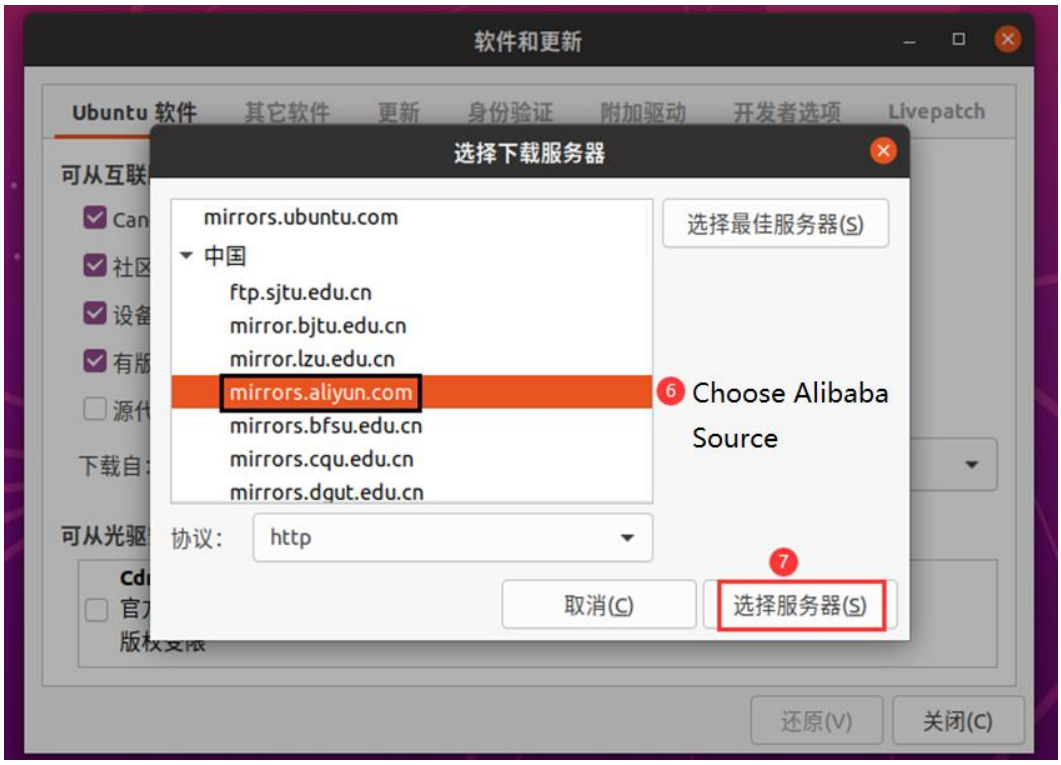


Figure 3.1-5 Change the software download source (4)

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

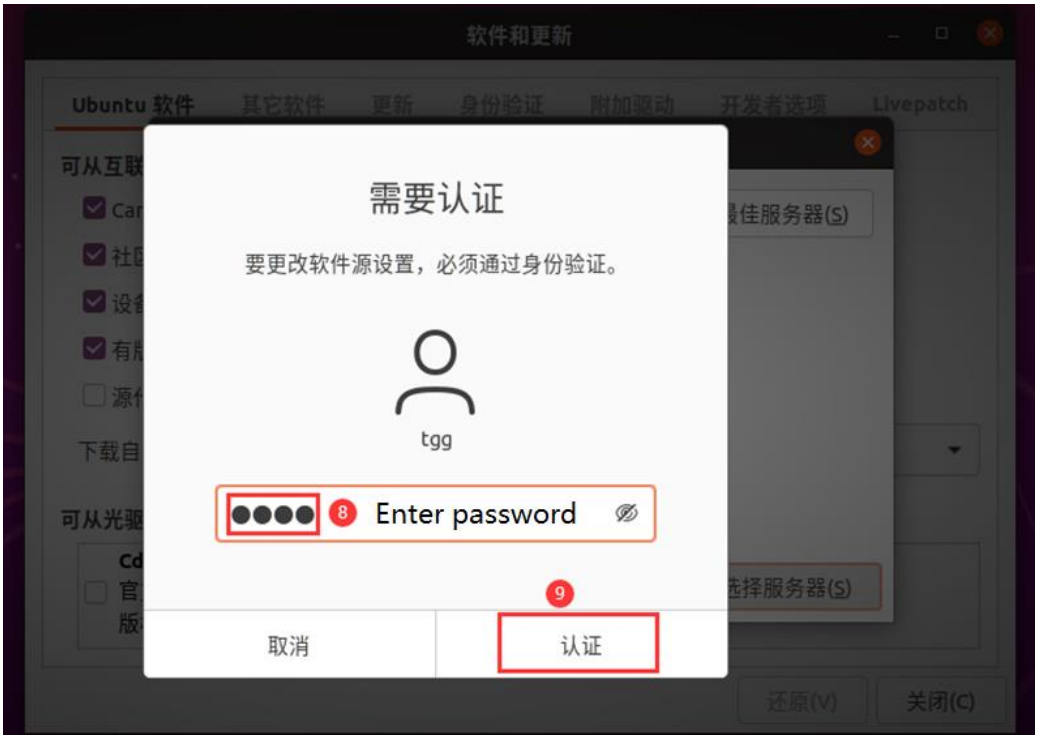


Figure 3.1-6 Change the software download source (5)

Figure 3.1-7 Change the software download source (6)

Figure 3.1-8 Change the software download source (7)

Figure 3.1-9 Change the software download source (8)

3.1.3 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:

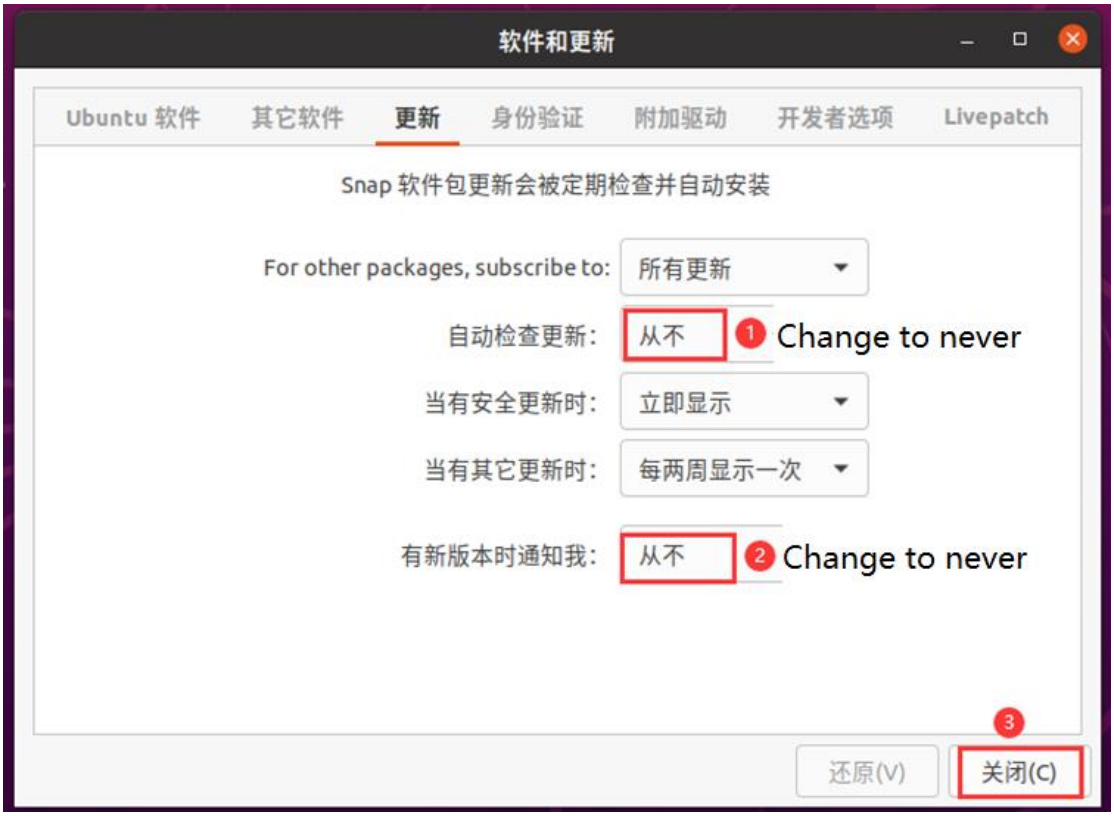


Figure 3.1-10 Disable automatic update function

3.2 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 subsection 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.

3.2.1 Setting 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 3.2-1 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 3.2-2 Confirm if the FTP service is enabled

3.2.2 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 QuarkPi-CA2 development board data disk has already provided the FileZilla software installation package for users. The path is: Development Board CD-ROM A Drive - Basic Materials -> 03_softwares -> FileZilla_3.61.0_win64-setup.exe. Users can also download and install the package file by visiting the link address: https://www.filezilla.cn/download/client.

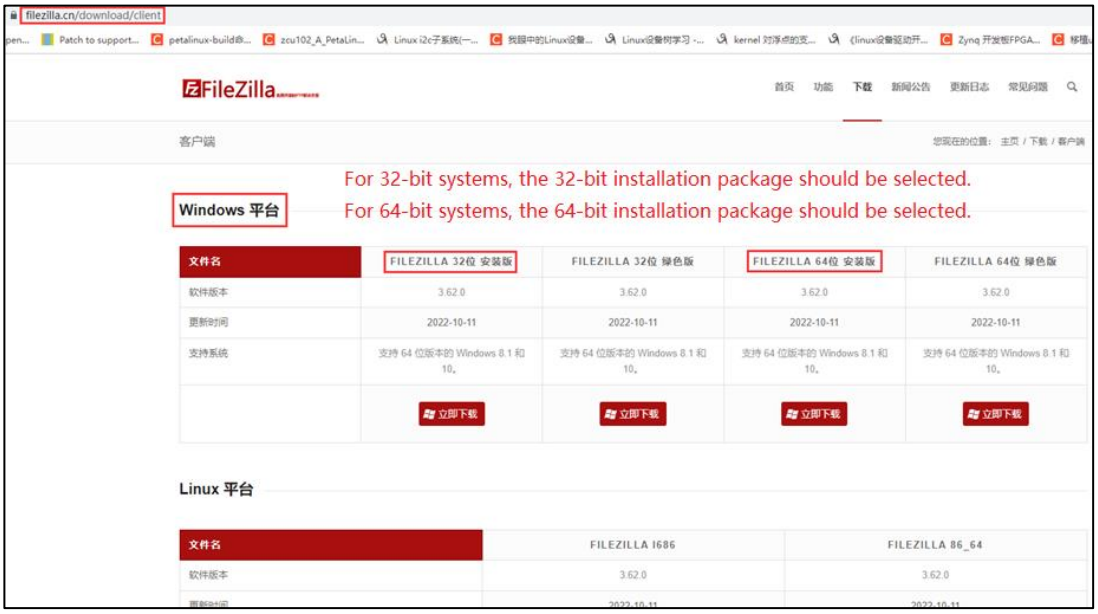


Figure 3.2-3 Download the installation package of FileZilla software

After the download is complete, it is as shown in Figure 3.2.4:

Figure 3.2-4 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 3.2.5 to 3.2.10 to complete the installation:

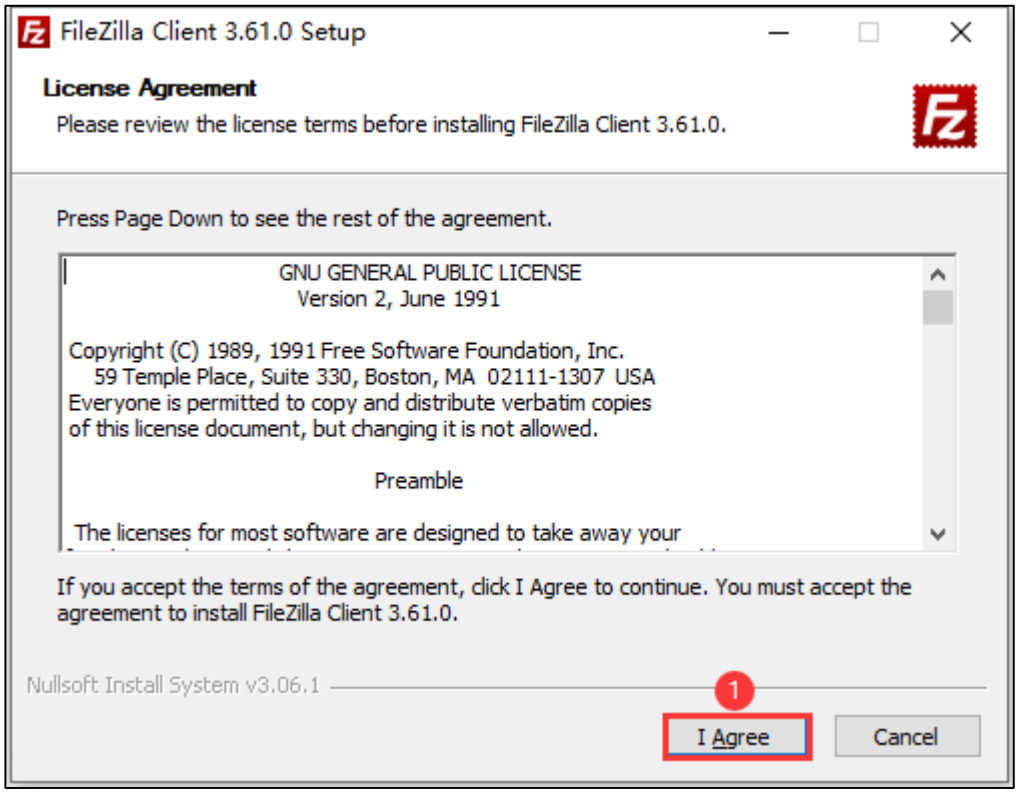


Figure 3.2-5 Install the FileZilla software (1)

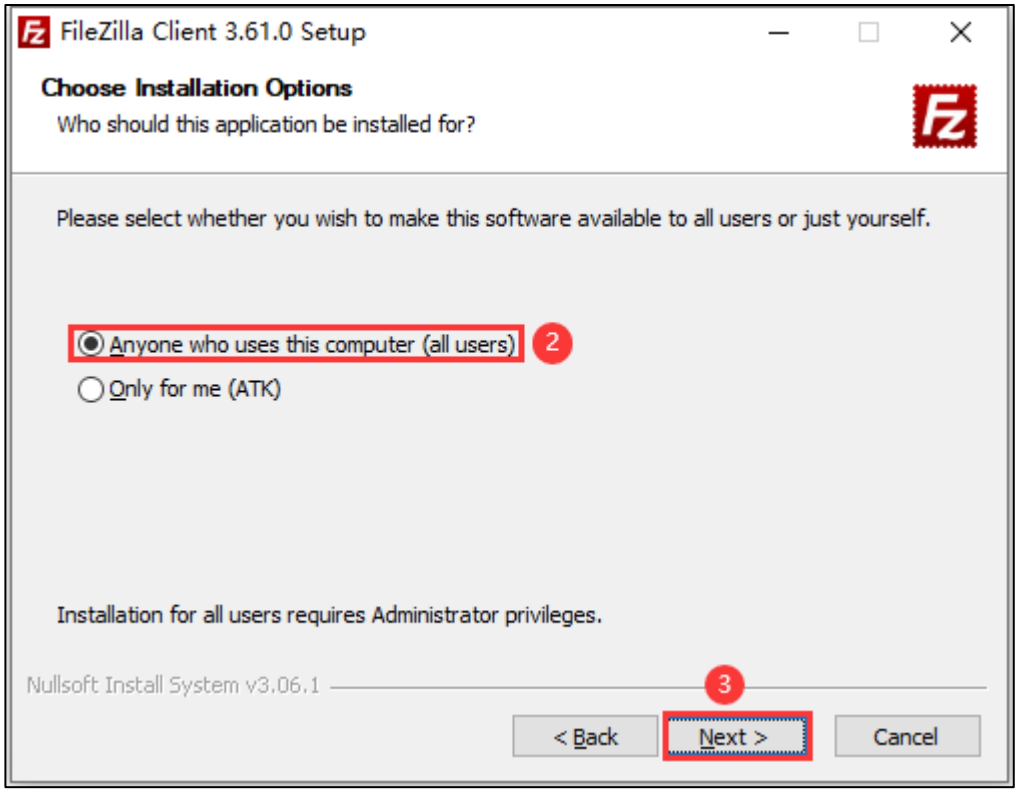


Figure 3.2-6 Install the FileZilla software (2)

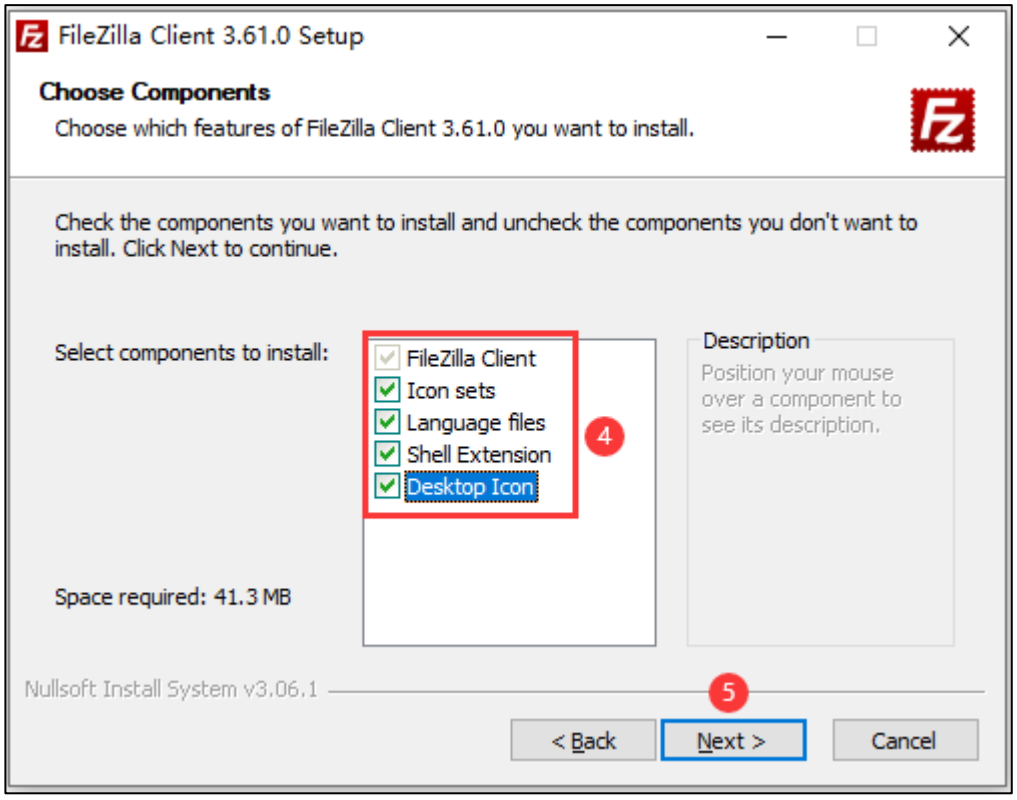


Figure 3.2-7 Install the FileZilla software (3)

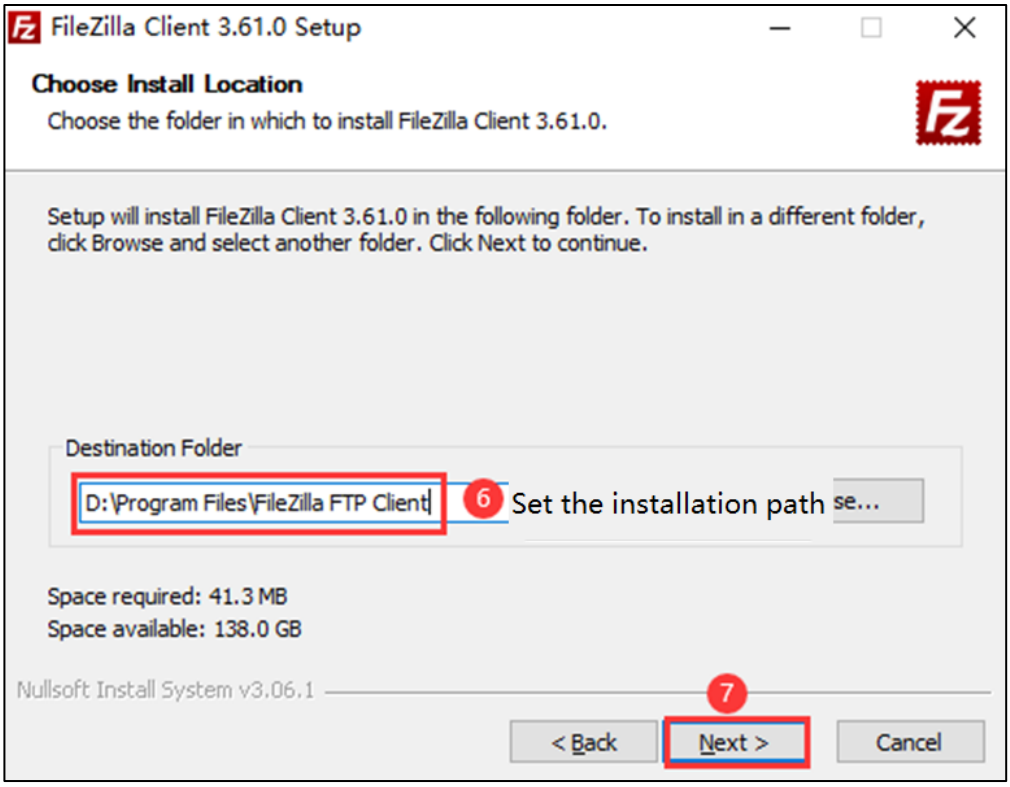


Figure 3.2-8 Install the FileZilla software (4)

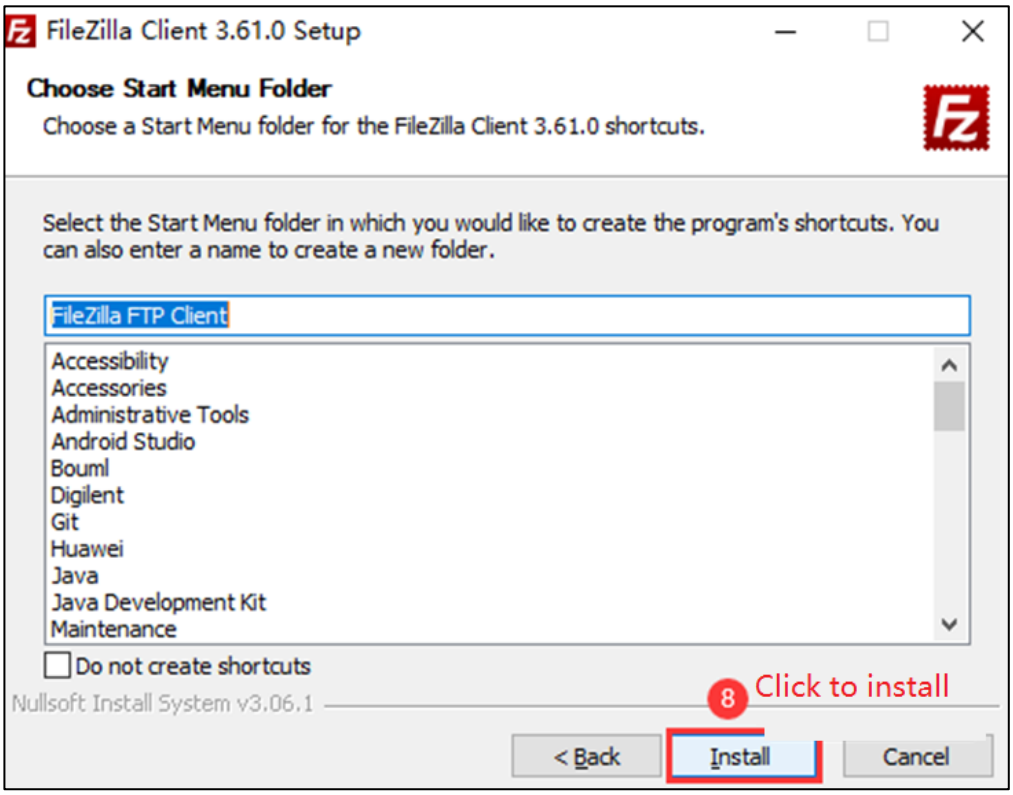


Figure 3.2-9 Install the FileZilla software (5)

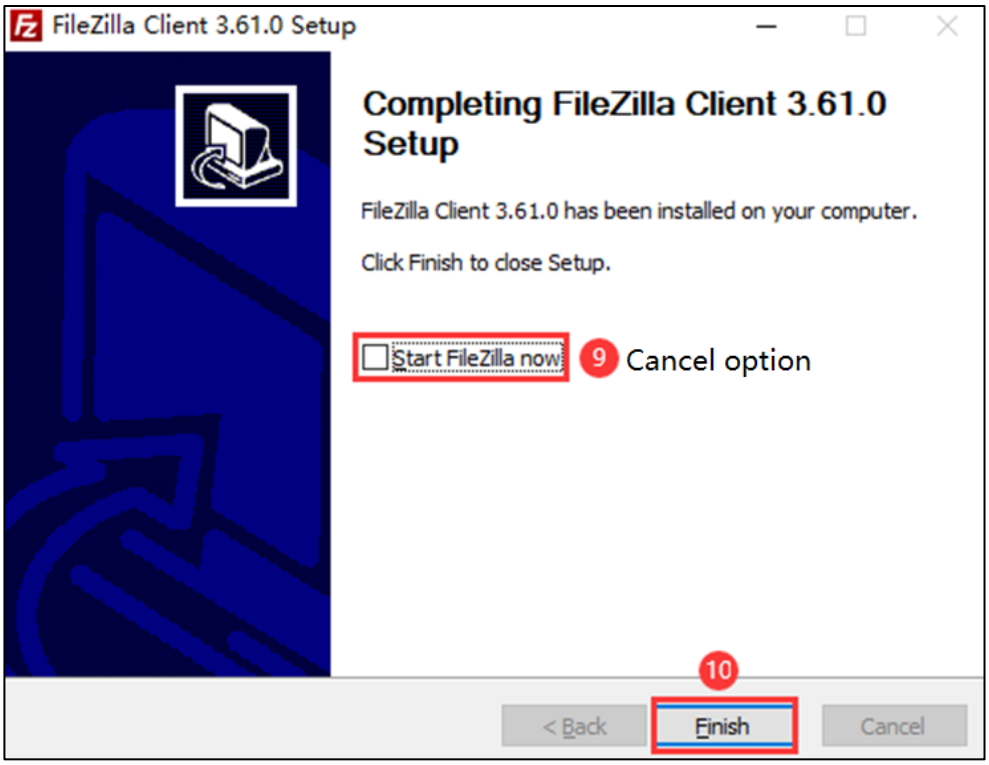


Figure 3.2-10 Install the FileZilla software (6)

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

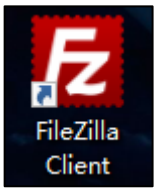


Figure 3.2-11 FileZilla desktop icon

3.2.3 Using FileZilla

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

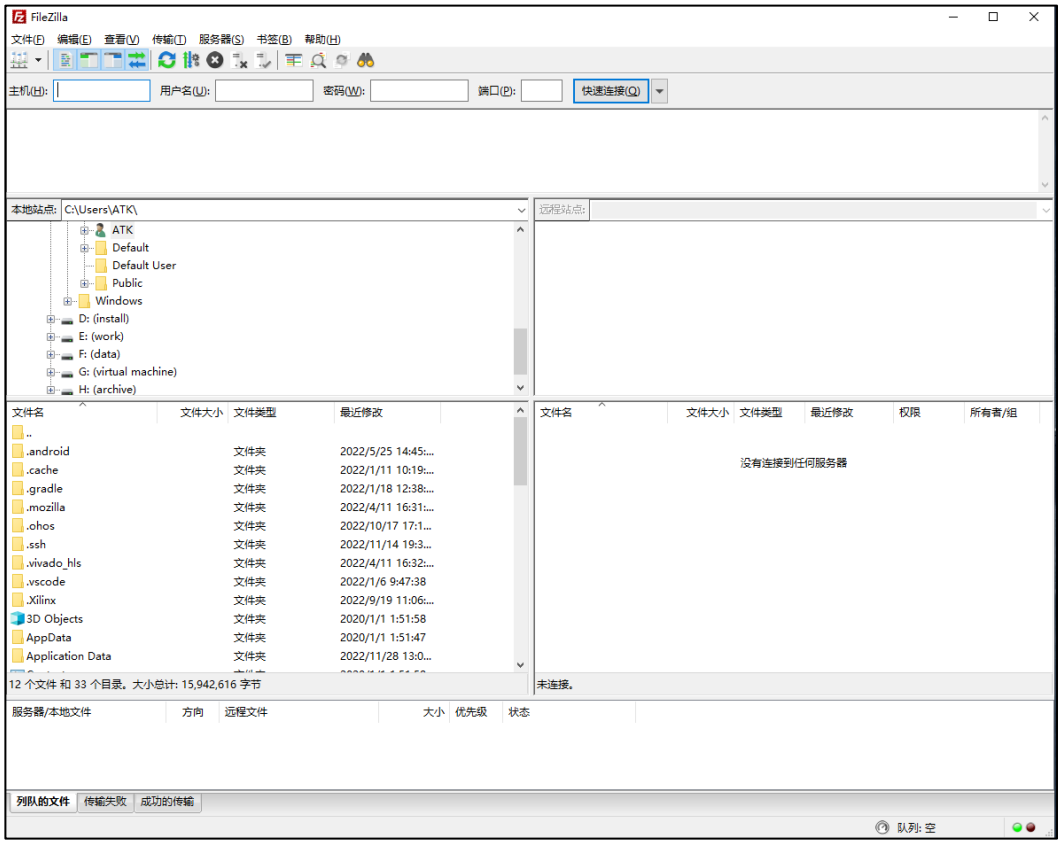


Figure 3.2-12 FileZilla software interface

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 3.2-13 FTP client connects to the FTP server (1)

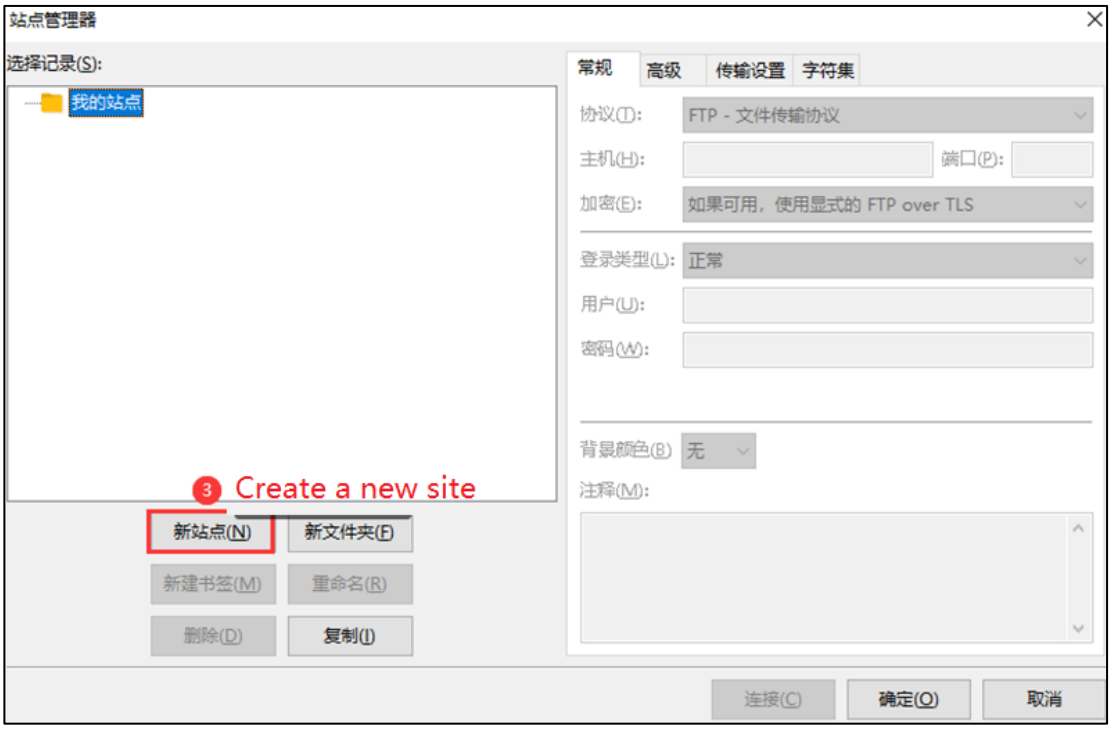


Figure 3.2-14 FTP client connects to the FTP server (2)

Figure 3.2-15 FTP client connects to the FTP server (3)

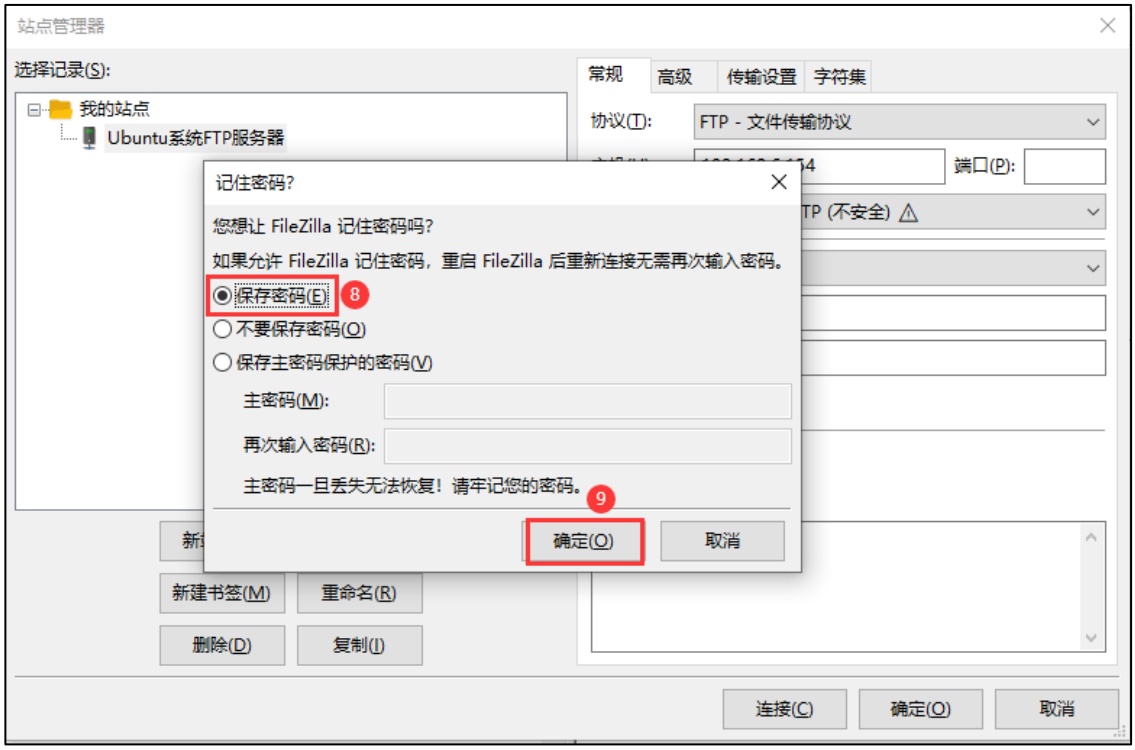


Figure 3.2-16 FTP client connects to the FTP server (4)

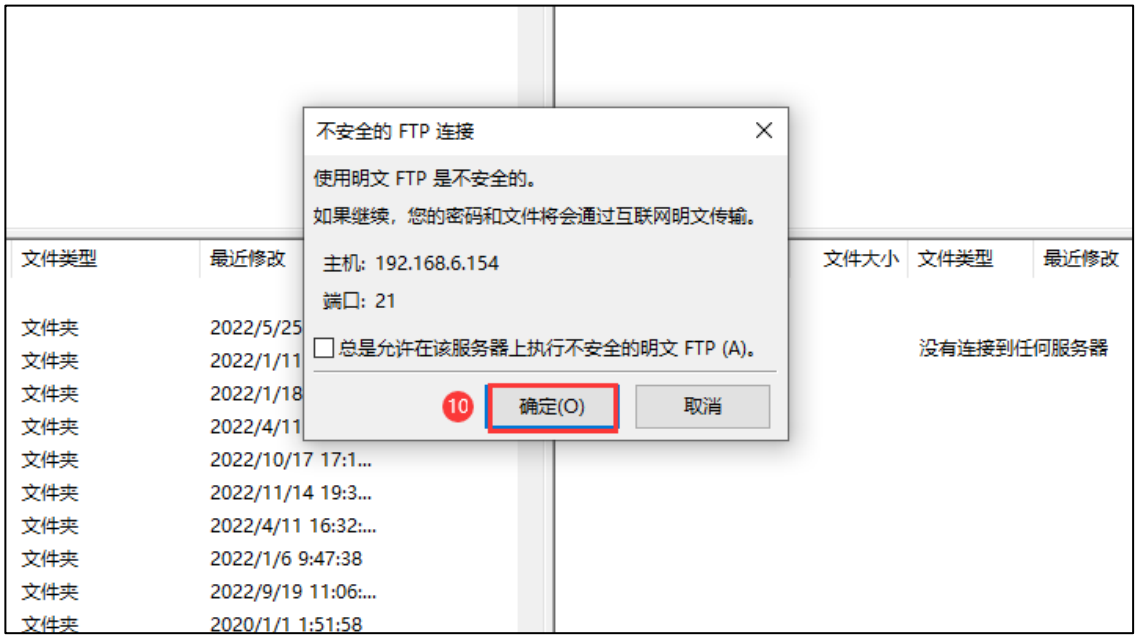


Figure 3.2-17 FTP client connects to the FTP server (5)

The connection was successful as shown in Figure 3.2.18:

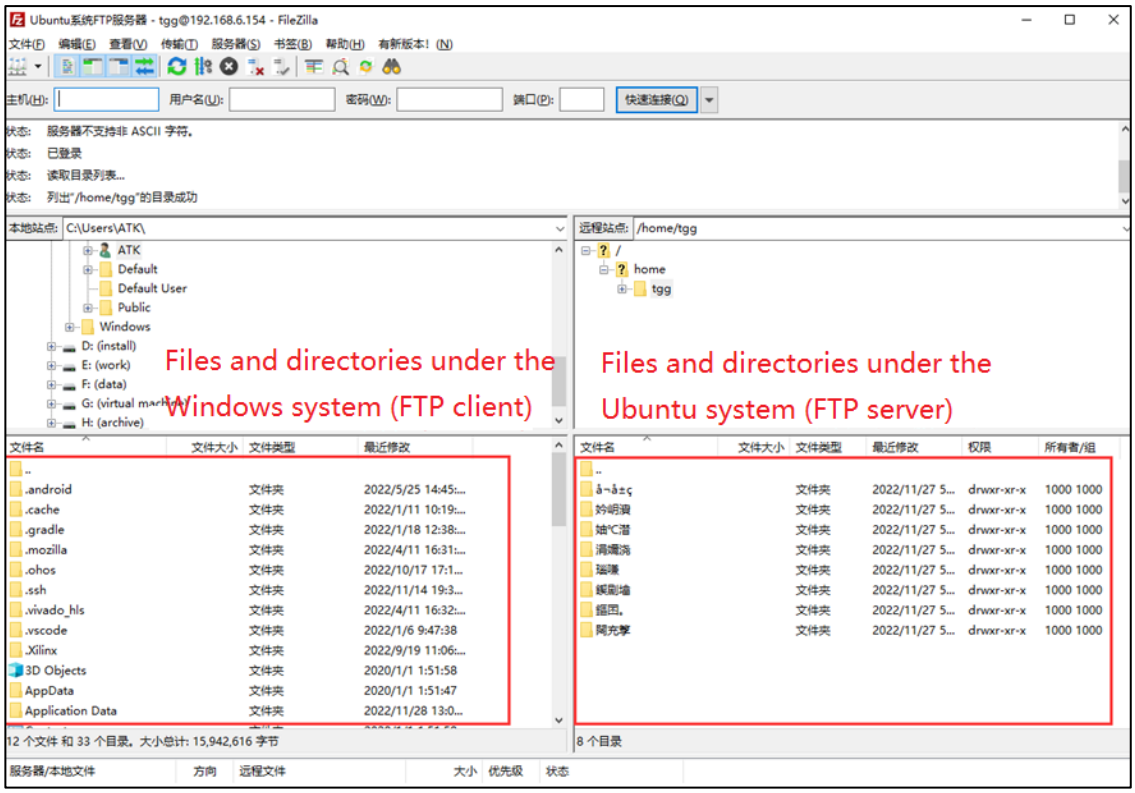


Figure 3.2-18 Connection successful

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

method and reopen the "Site Manager", following the steps shown in Figure 3.2.19 to set the encoding method：

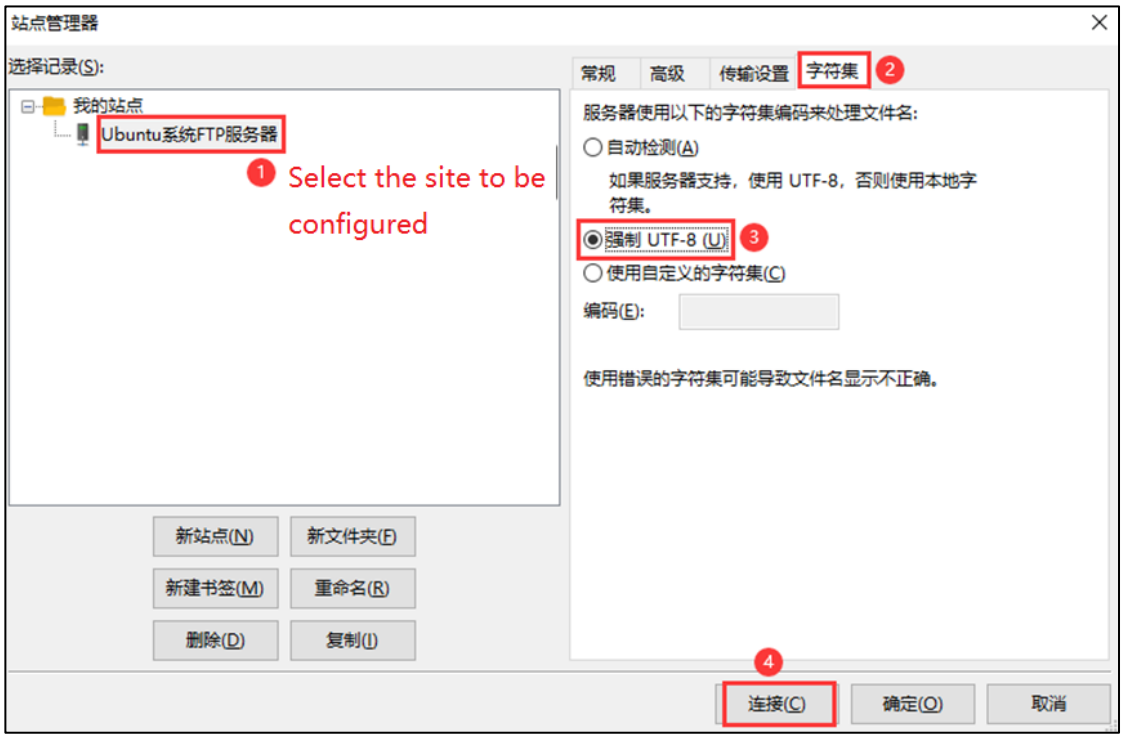


Figure 3.2-19 Set character encoding method (1)

Figure 3.2-20 Set character encoding method (2)

After reconnecting again, you will find that the file name is displayed normally, as shown in Figure 3.2.21:

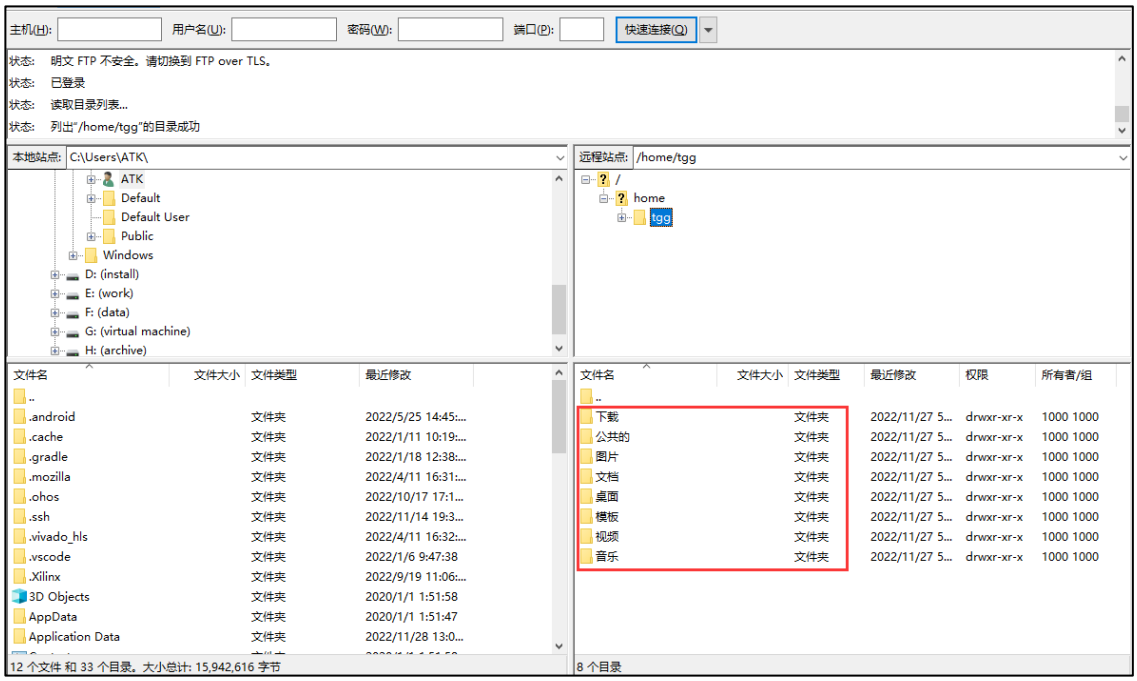


Figure 3.2-21 The file name is displayed normally.

After the client successfully connects to the server, file transfer can be carried out. The transfer method is very simple. Select the file to be transferred and directly use the left mouse button to drag it 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 this 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!

3.3 Setting up the ssh server in 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 start the ssh service in the Ubuntu system. Execute the following command to install openssh-server:

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

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

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

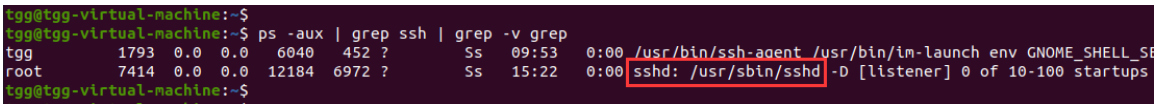


Figure 3.3-1 Confirm whether the SSH service is enabled

3.4 Installation of CH343 serial port driver

The QuarkPi-CA2 card computer of ALIENTEK Corporation has reserved serial port pins and needs to be used in conjunction with a serial port module. Here, we take the official serial port module of ALIENTEK Corporation as an example. The serial port module uses the domestic chip CH343 to

achieve USB to serial port functionality. However, the development board's debugging serial port needs to be recognized by installing the CH343 driver on Windows.

The CH343 driver installation file has been provided to users in the development board data disk. The path is: Development board optical disc A drive - Basic data -> 03_softwares -> CH343SER.EXE. Double-click the executable file CH343SER.EXE, and follow the steps shown in Figures 3.4.1 to 3.4.3 to install the CH343 driver (do not connect the development board's debugging serial port before installation):

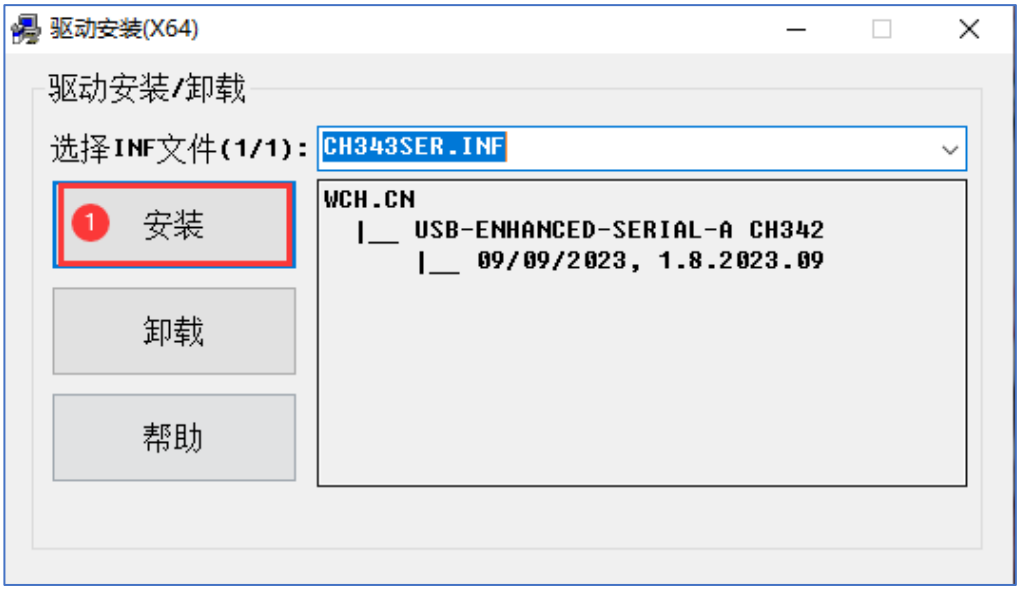


Figure 3.4-1 Install CH343 driver (1)

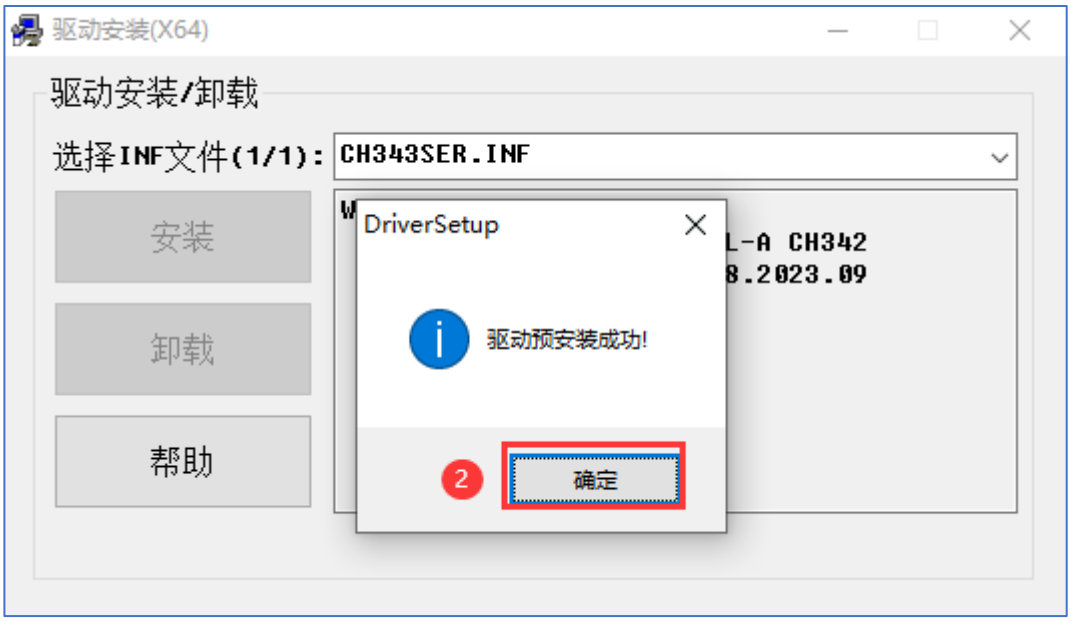


Figure 3.4-2 Install CH343 driver (2)

After the installation is completed, the serial port module is connected to the computer and the card computer. The connection method is shown in Figure 3.4.3:

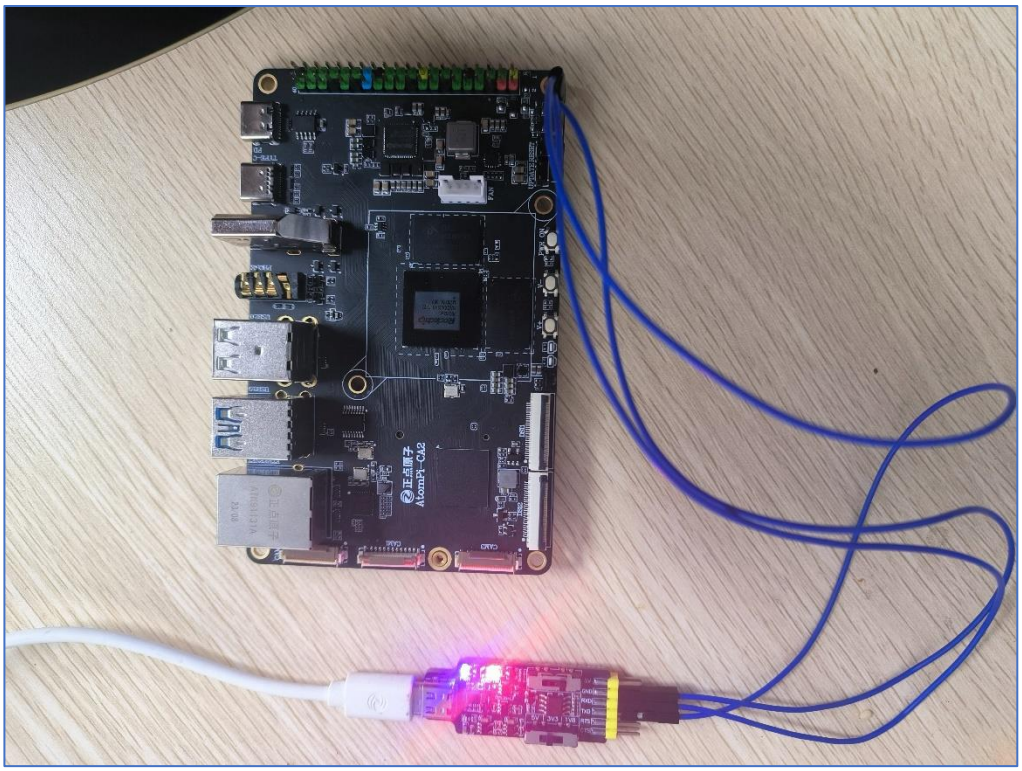


Figure 3.4-3 The development board is debugging the serial port and is connected to the computer.

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

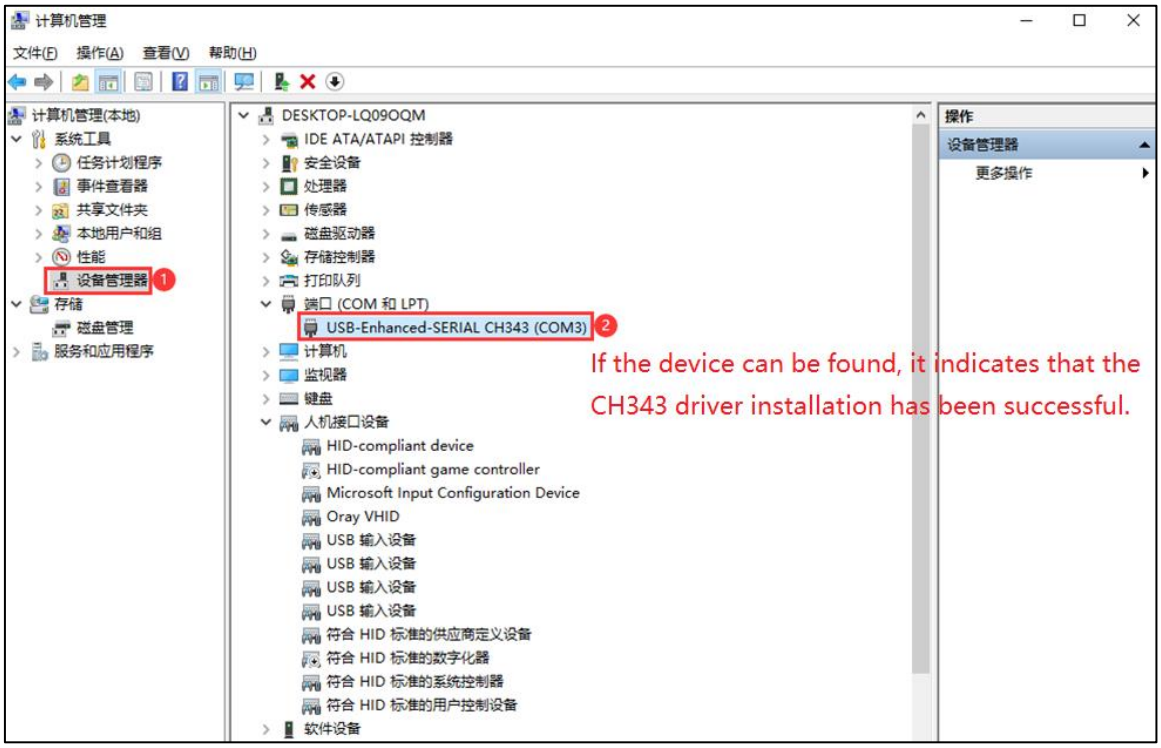


Figure 3.4-4 Windows Device Manager

If you can find a device named "USB-SERIAL CH343" under "Device Manager -> Ports (COM & LPT)", it indicates that the CH343 driver has been installed successfully! If you cannot find the device

named "USB-SERIAL CH343", please have the user check if the hardware connection is incorrect! You can unplug the USB cable, reconnect it, or try connecting to another USB port on the computer.

3.5 Installation of MobaXterm Software

MobaXterm is a versatile remote terminal software with powerful functions and is free (there are also paid versions). It supports numerous remote connection features such as SSH, Telnet, Rsh, Xdmc, RDP, VNC, FTP, SFTP, serial port (Serial COM), etc. MobaXterm offers a user-friendly interface and is extremely powerful, so it is recommended that users use this terminal software.

3.5.1 Download of MobaXterm Software

The development board data disk has already provided the installation package of MobaXterm to users. The path is: Development Board CD-ROM A Drive - Basic Materials -> 04_softwares -> MobaXterm_Installer_v12.3.zip. Users can also download it themselves through the link address: https://mobaxterm.mobatek.net/download.html

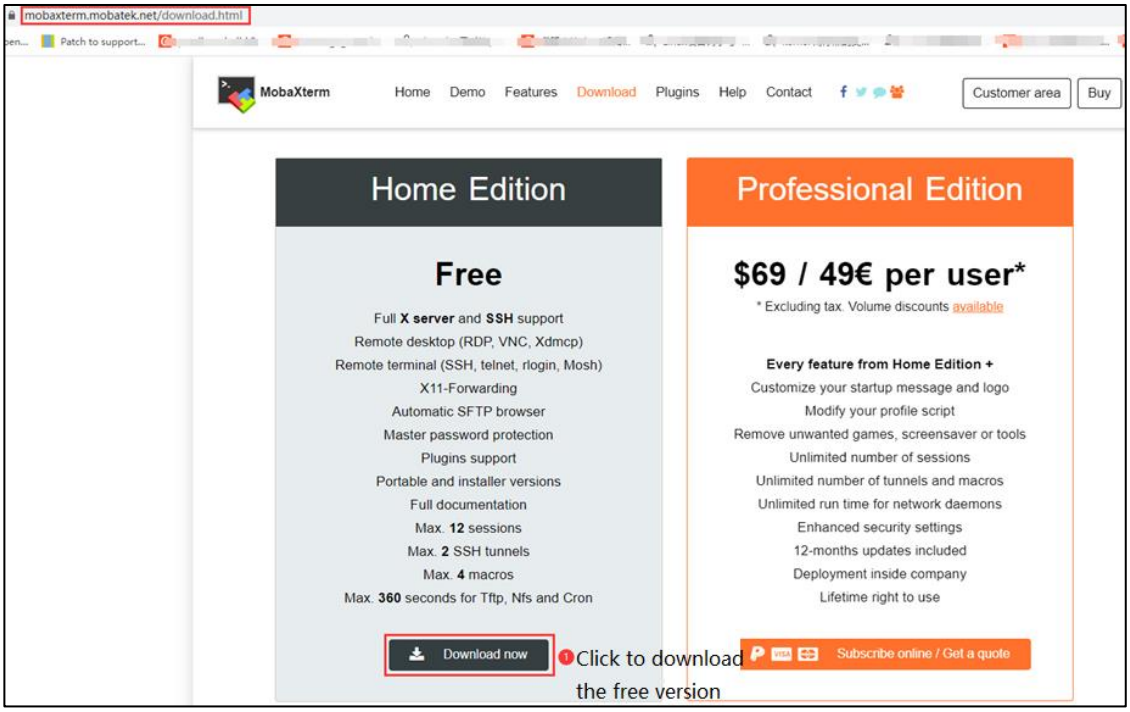


Figure 3.5-1 Download the MobaXterm software (1)

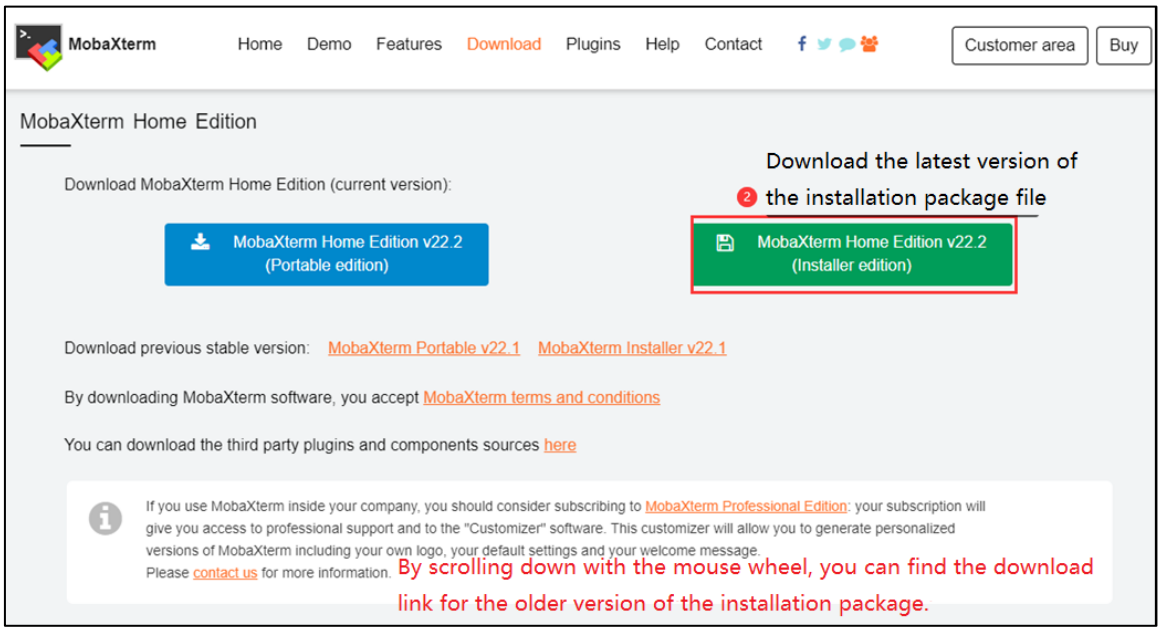


Figure 3.5-2 Download the MobaXterm software (2)

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

3.5.2 Installation of MobaXterm software

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

Figure 3.5-3 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 3.5.4 to 3.5.8 to install the MobaXterm software:

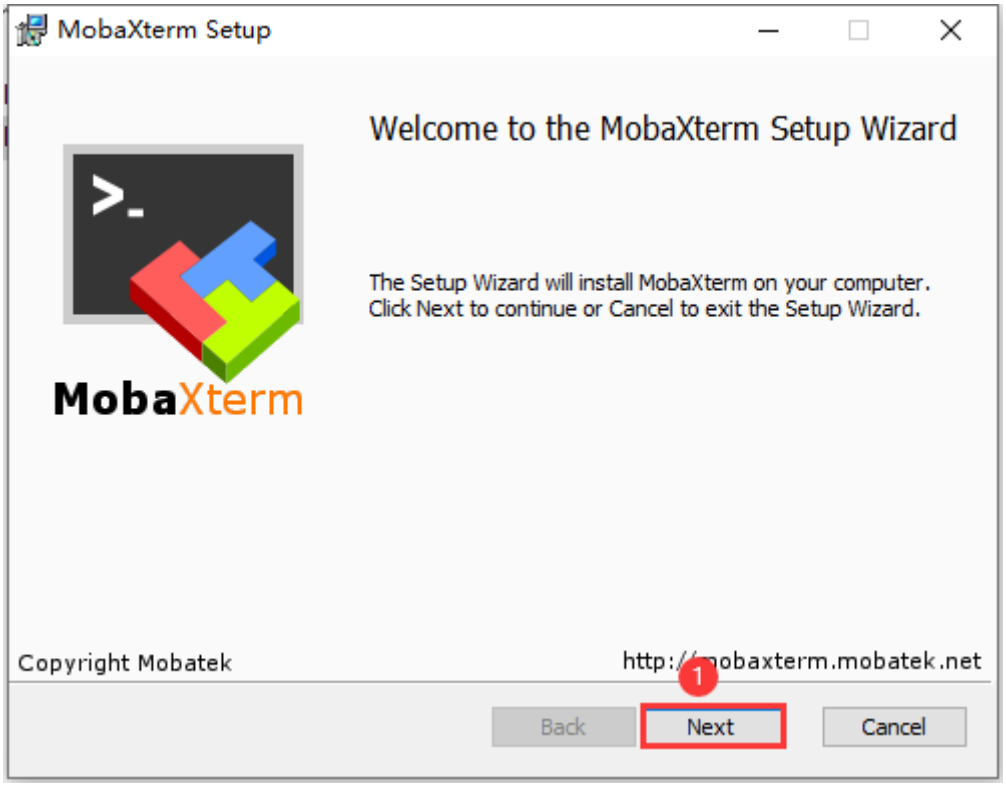


Figure 3.5-4 Install the MobaXterm software (1)

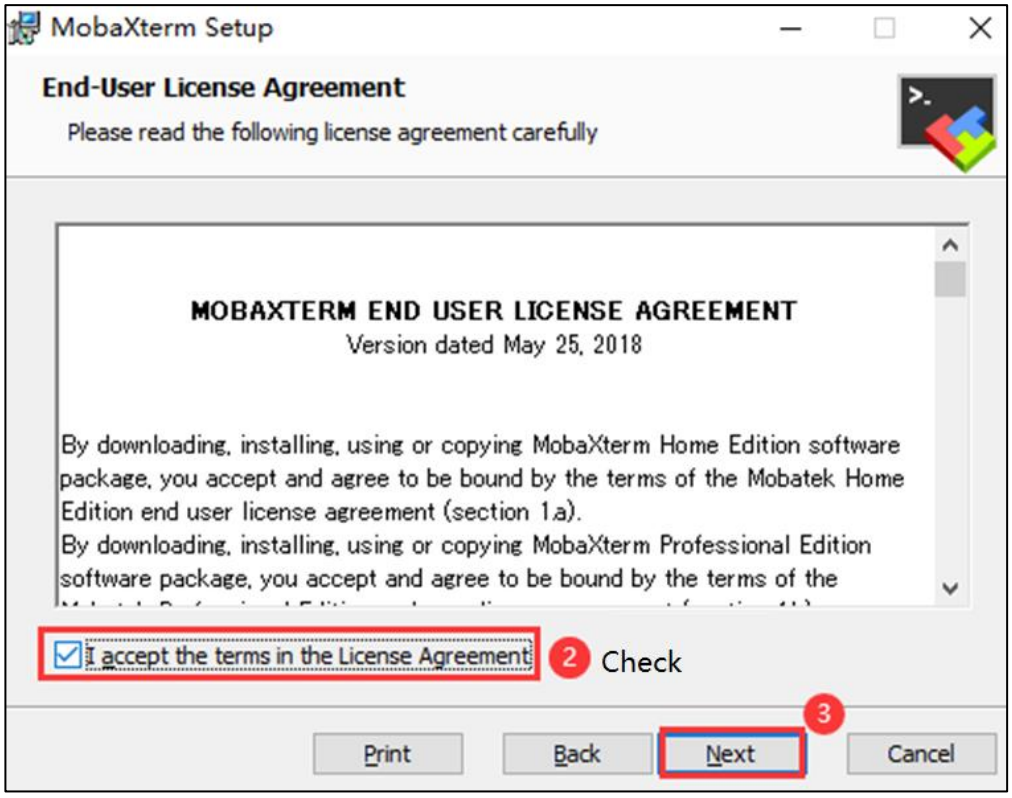


Figure 3.5-5 Install the MobaXterm software (2)

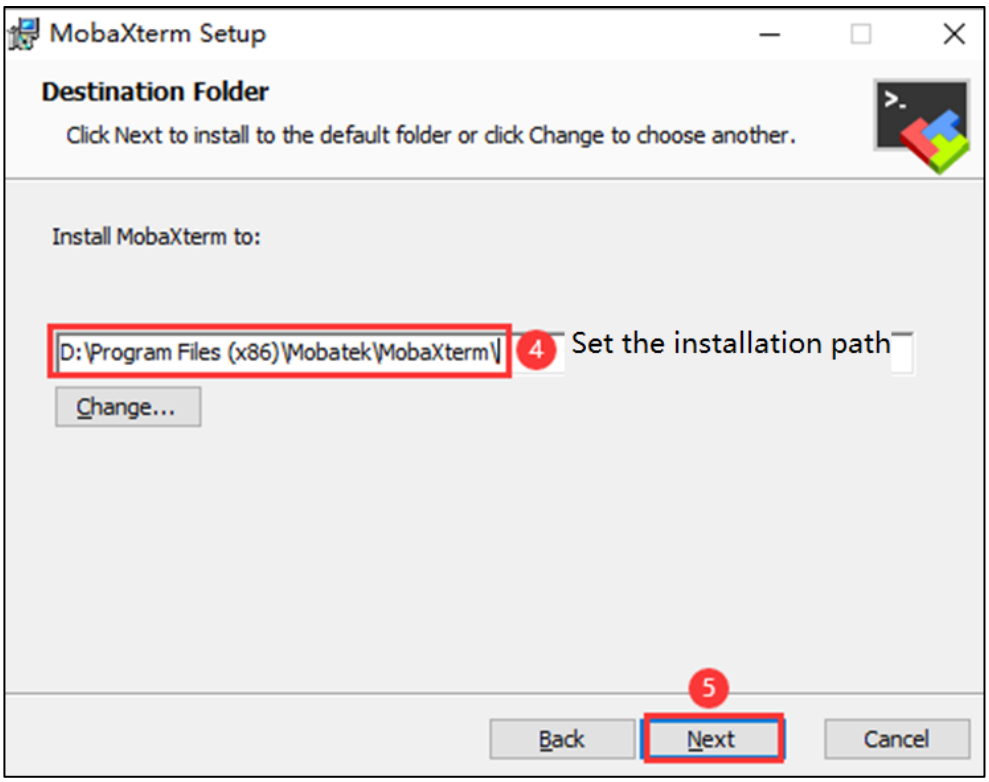


Figure 3.5-6 Install the MobaXterm software (3)

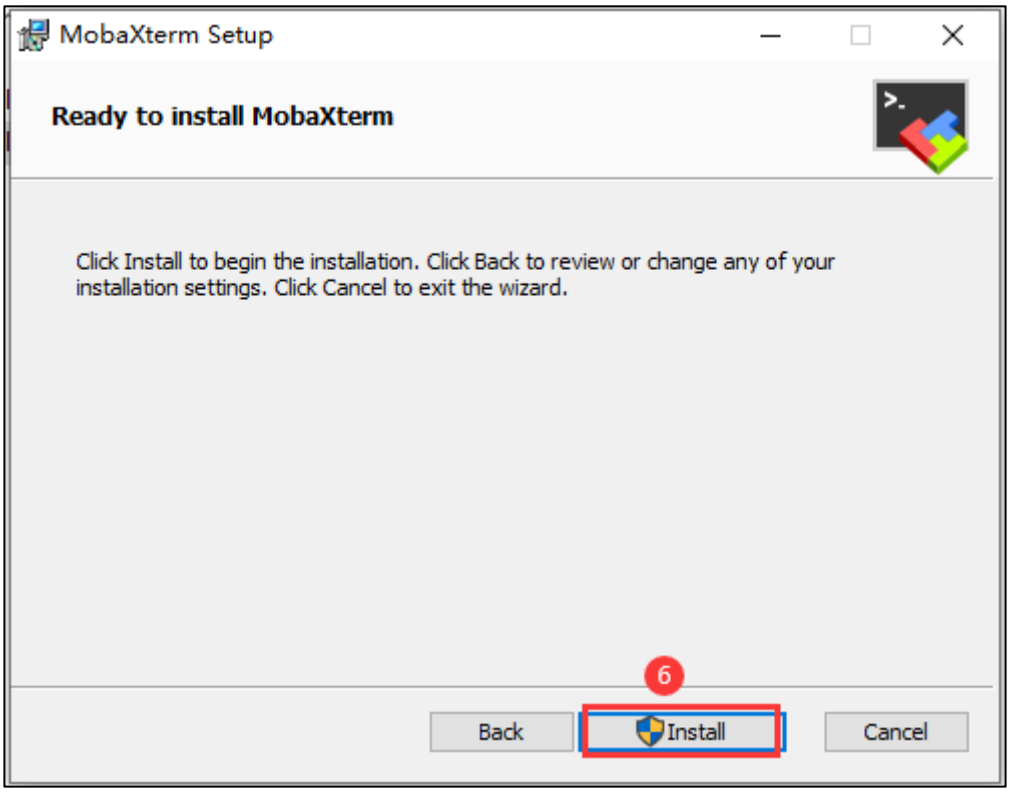


Figure 3.5-7 Install the MobaXterm software (4)

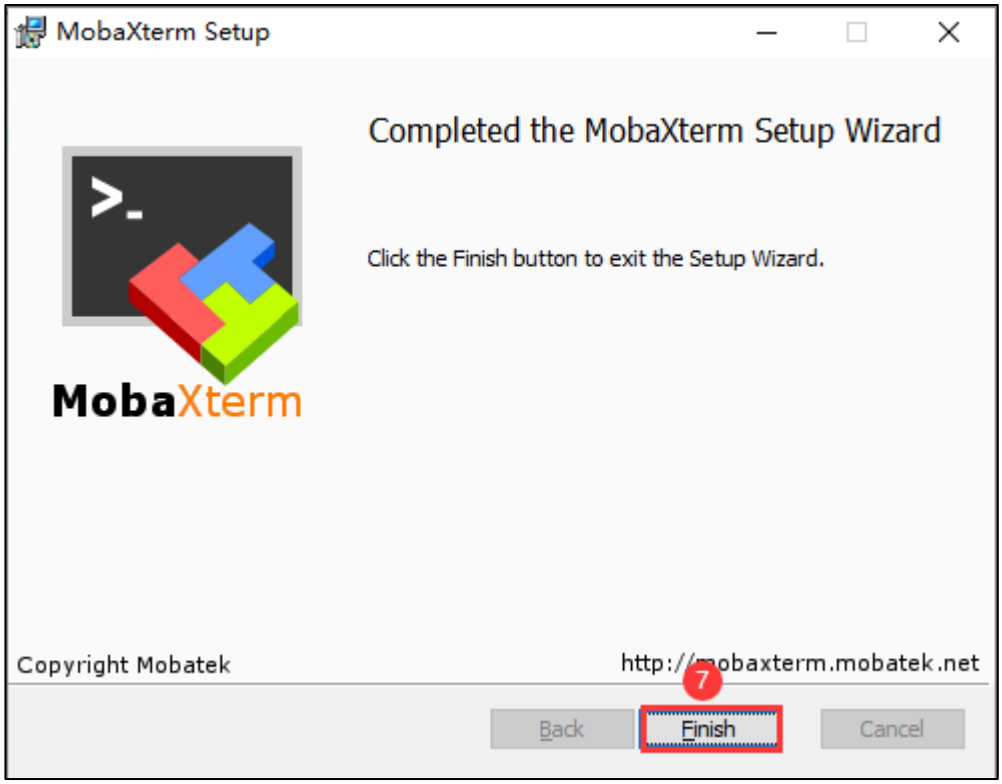


Figure 3.5-8 Install the MobaXterm software (5)

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

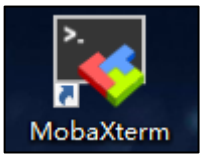


Figure 3.5-9 MobaXterm software desktop icon

3.5.3 Usage of MobaXterm software

Double-click the MobaXterm desktop icon to open the software, as shown in Figure 3.5.10:

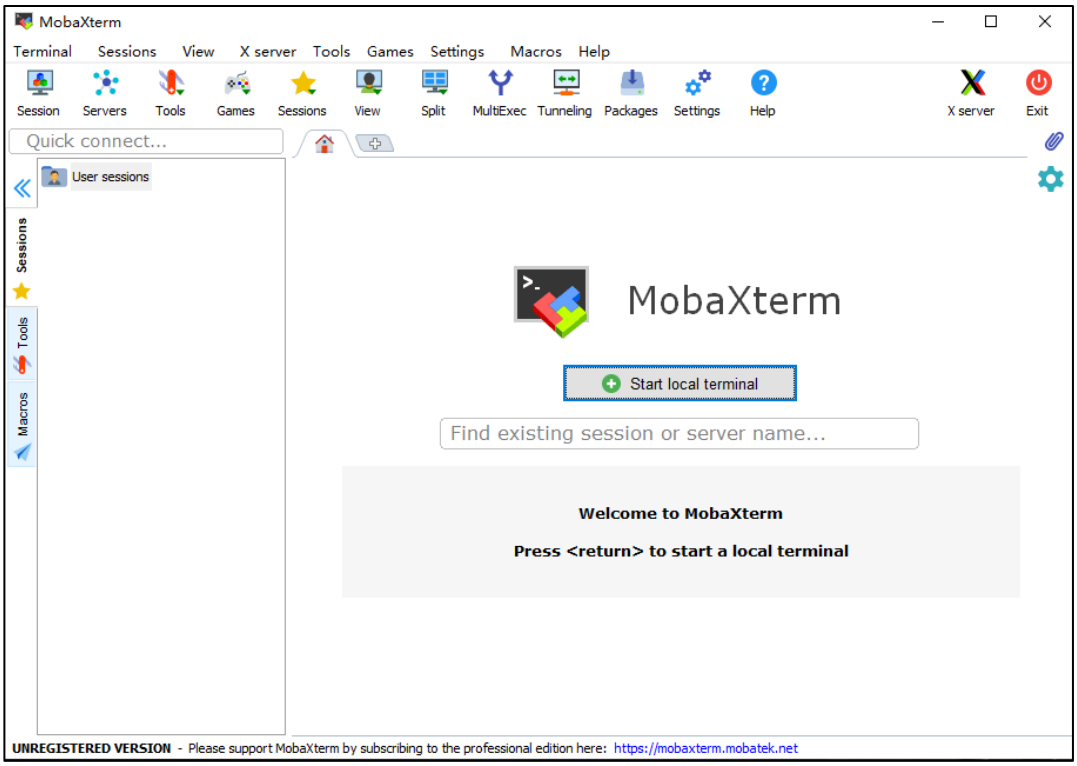


Figure 3.5-10 The main interface of MobaXterm software

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

1. Serial port connection

The QuarkPi-CA2 card computer has reserved serial port pins, which can be directly connected to the computer through the serial port module. 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 3.5.11 to 3.5.12 to establish a Serial (serial port) connection (before establishing the connection, you need to connect the debugging serial port of the development board to the computer via a USB cable, and the CH343 driver has been installed):

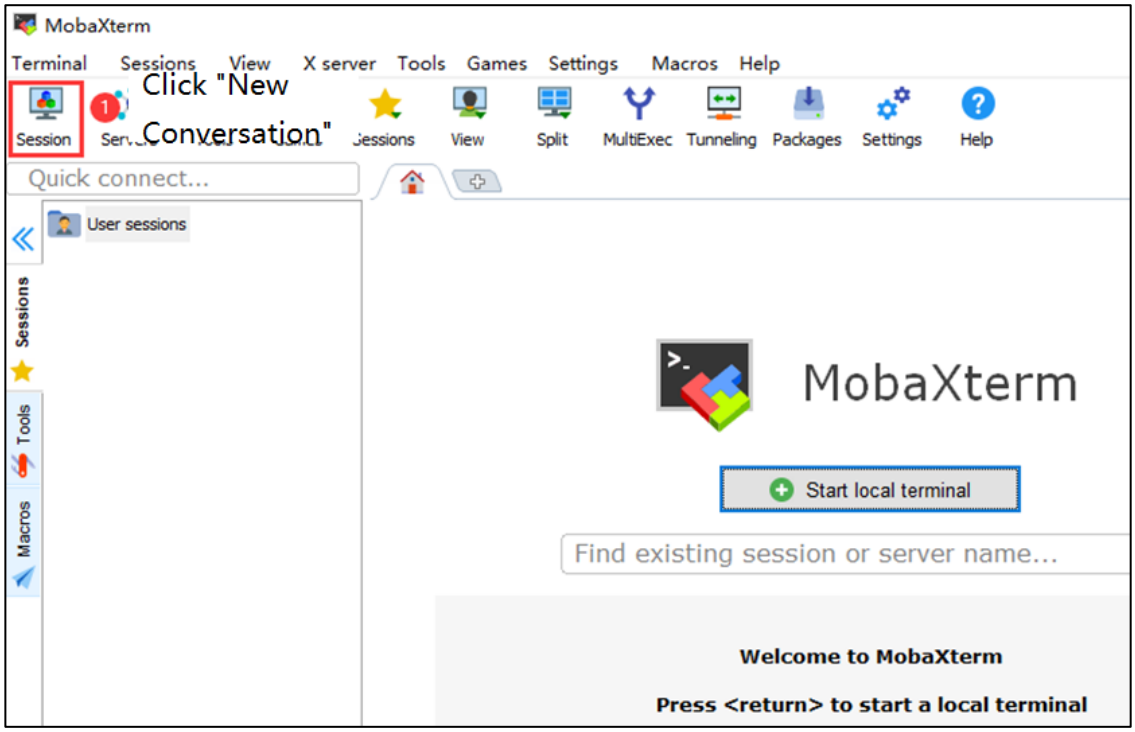


Figure 3.5-11 Establish serial port connection (1)

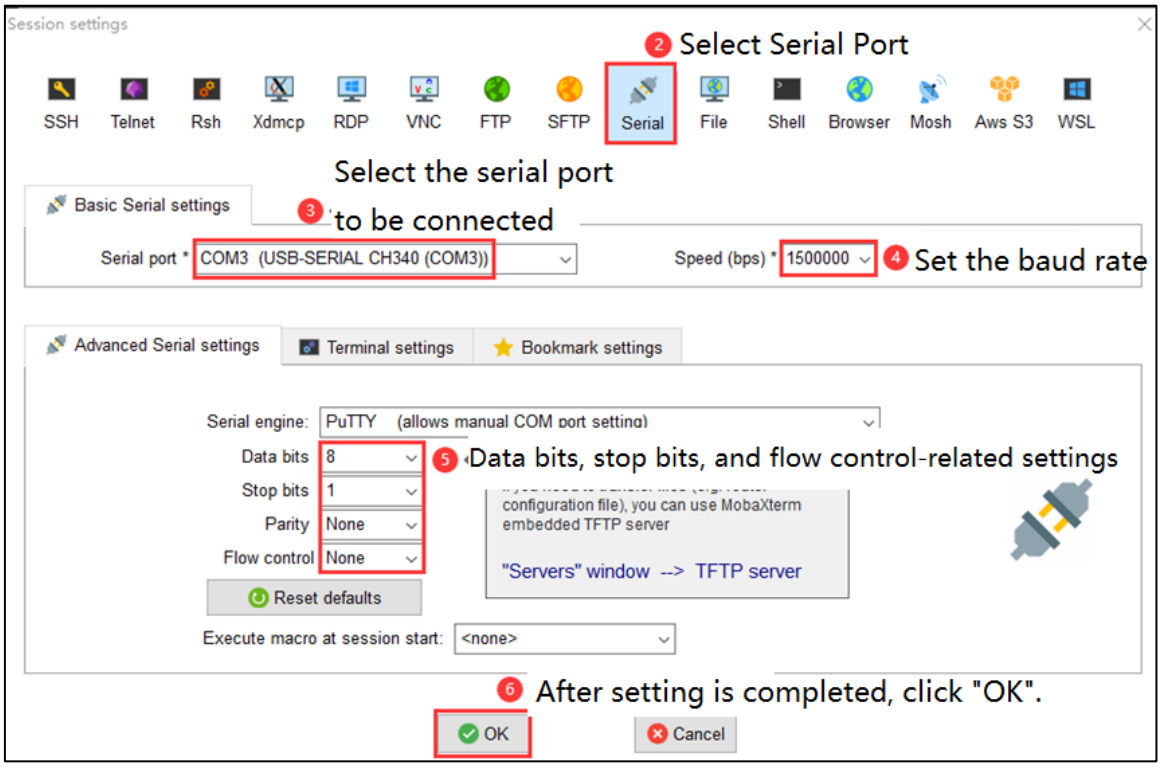


Figure 3.5-12 Establish serial port connection (2)

First, select the serial port that needs to be connected. Make sure that the debugging serial port of the development board is already connected to the computer via a USB cable, and that the CH343 driver has been successfully installed ("Device Manager Ports (COM and LPT)") and can be found under a device named "USB-SERIAL CH343". Only then can the MobaXterm software recognize the

debugging serial port of the development board. We can find the corresponding serial port of the development board (USB-SERIAL CH343) in the "Serial port" drop-down list, and then select it!

Next, set the serial communication baud rate. This should be done according to the actual situation. The debugging serial port of the QuarkPi-CA2 platform has a default baud rate of 1500000. Then, set the data bits, stop bits, and flow control, etc. After the settings are completed, click the "OK" button.

The serial connection has been successfully established, as shown in Figure 3.5.13:

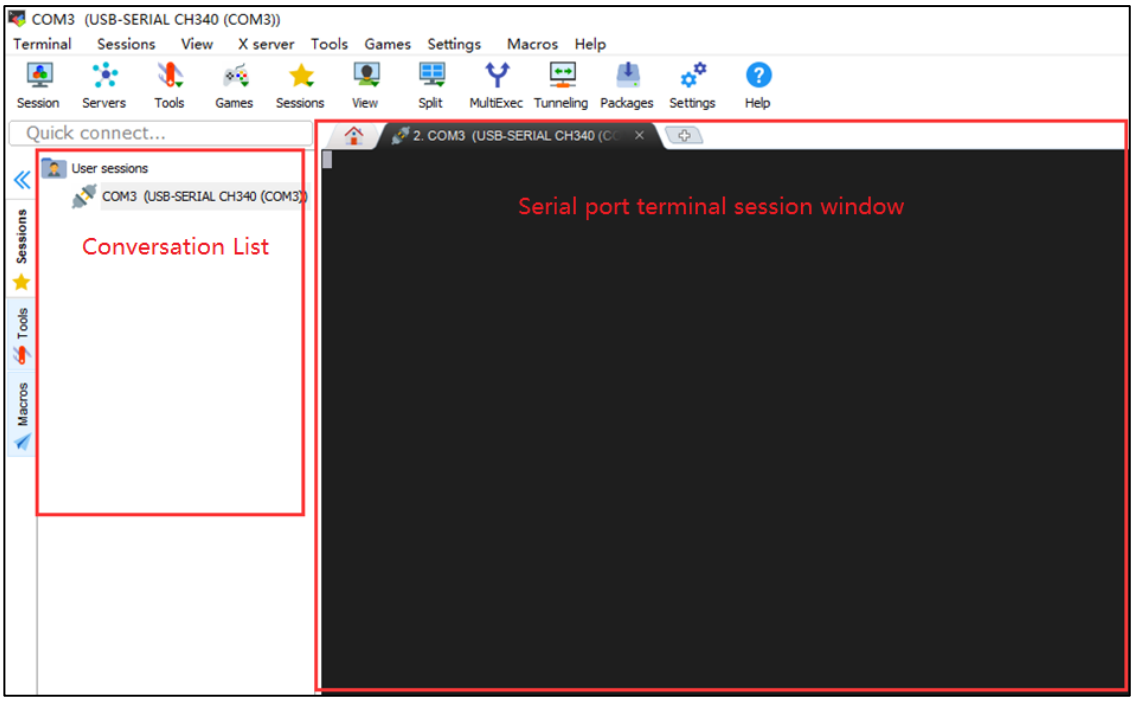


Figure 3.5-13 Establish serial port connection (3)

When the development board is powered on, it starts the Linux system or the Android system. During the system startup process, the log information will be displayed in the serial port terminal session window, as shown in Figure 3.5.14.

```
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 3.5-14 System startup log information

After the system starts successfully, users can execute commands through the serial port terminal and the execution results will also be displayed through 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, remote login to the development board Linux system; follow the operation steps shown in Figures 3.5.15 to 3.5.18 to establish the SSH remote connection and perform the remote login operation (taking remote login to the Ubuntu system as an example):

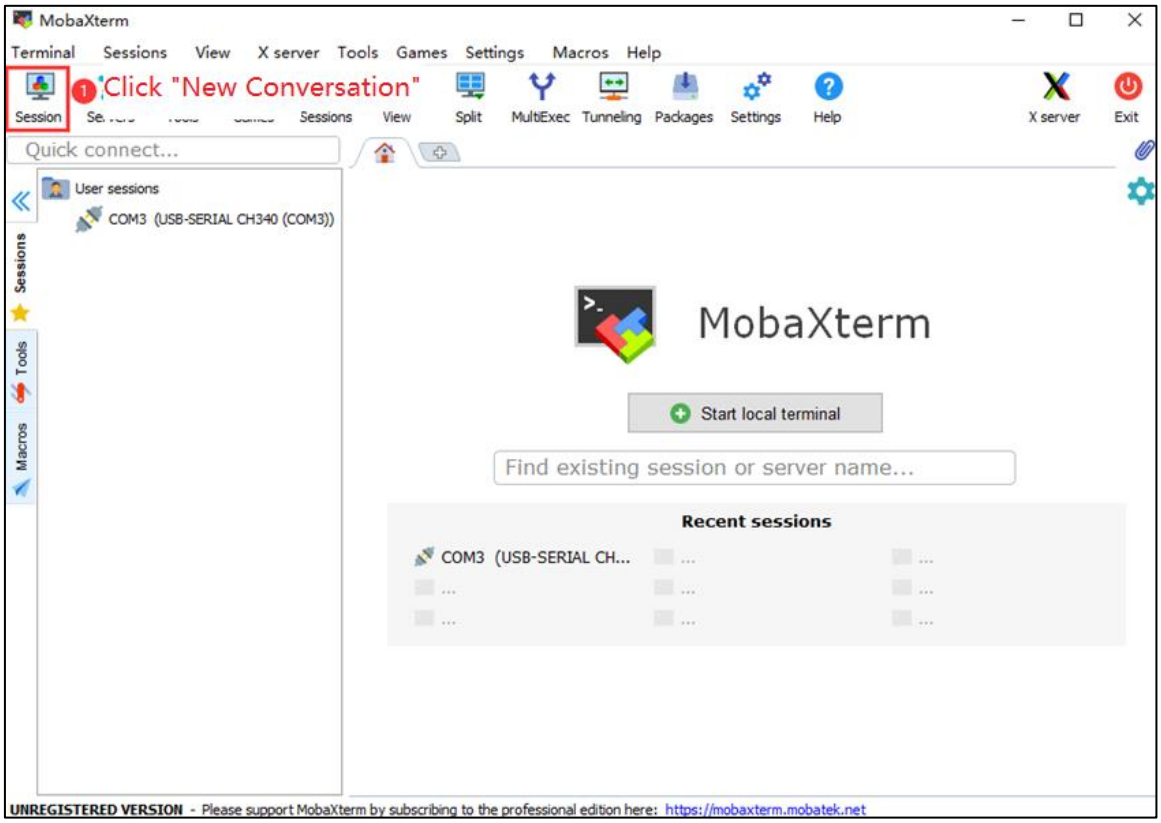


Figure 3.5-15 Establish SSH remote connection (1)

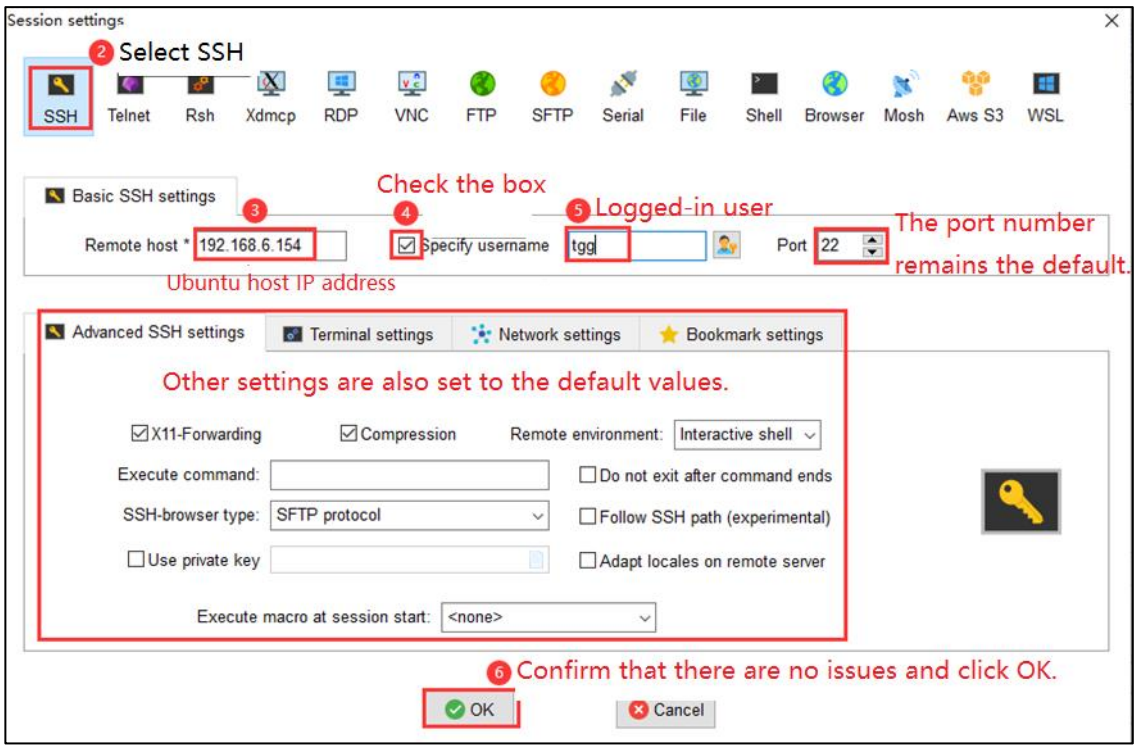


Figure 3.5-16 Establish SSH remote connection (2)

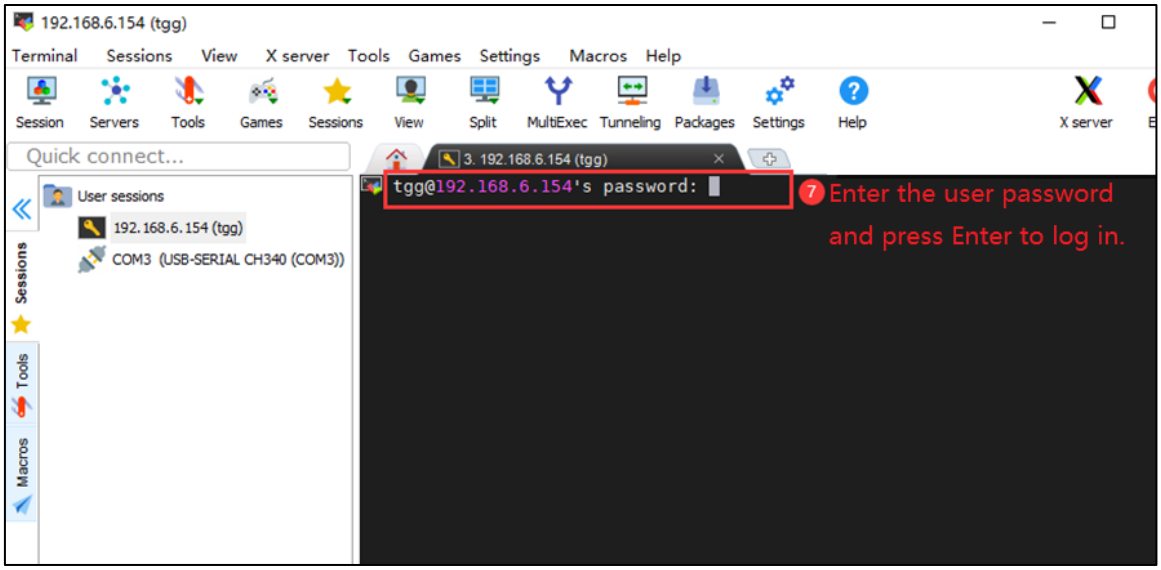


Figure 3.5-17 Establish SSH remote connection (3)

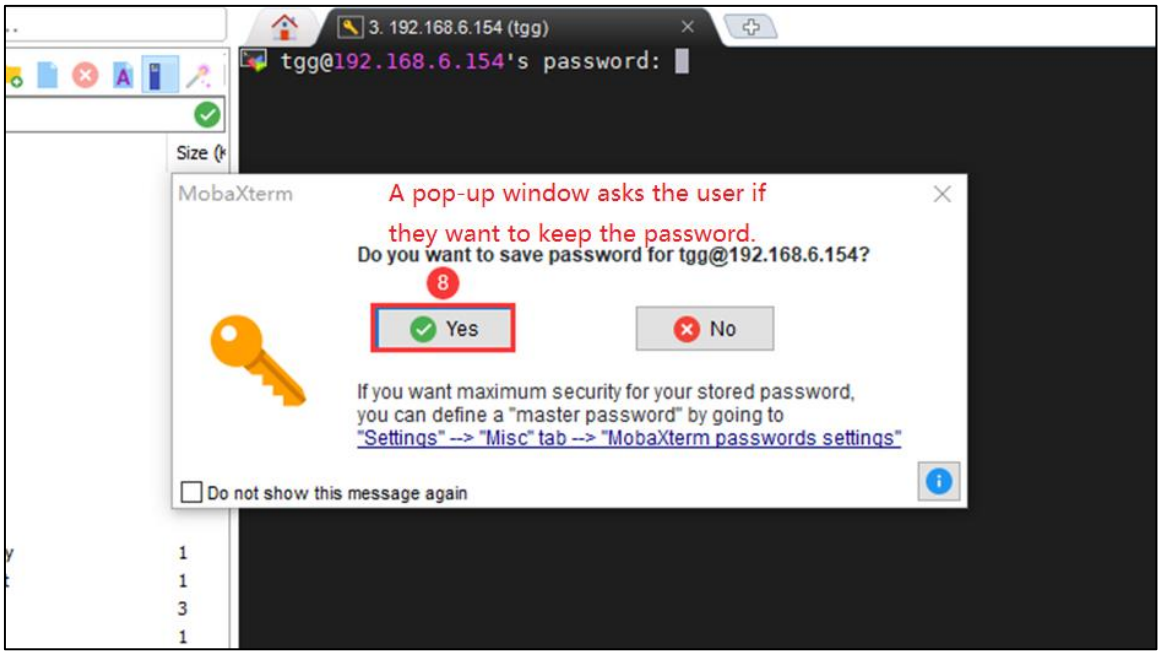


Figure 3.5-18 Establish SSH remote connection (4)

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 3.5.19:

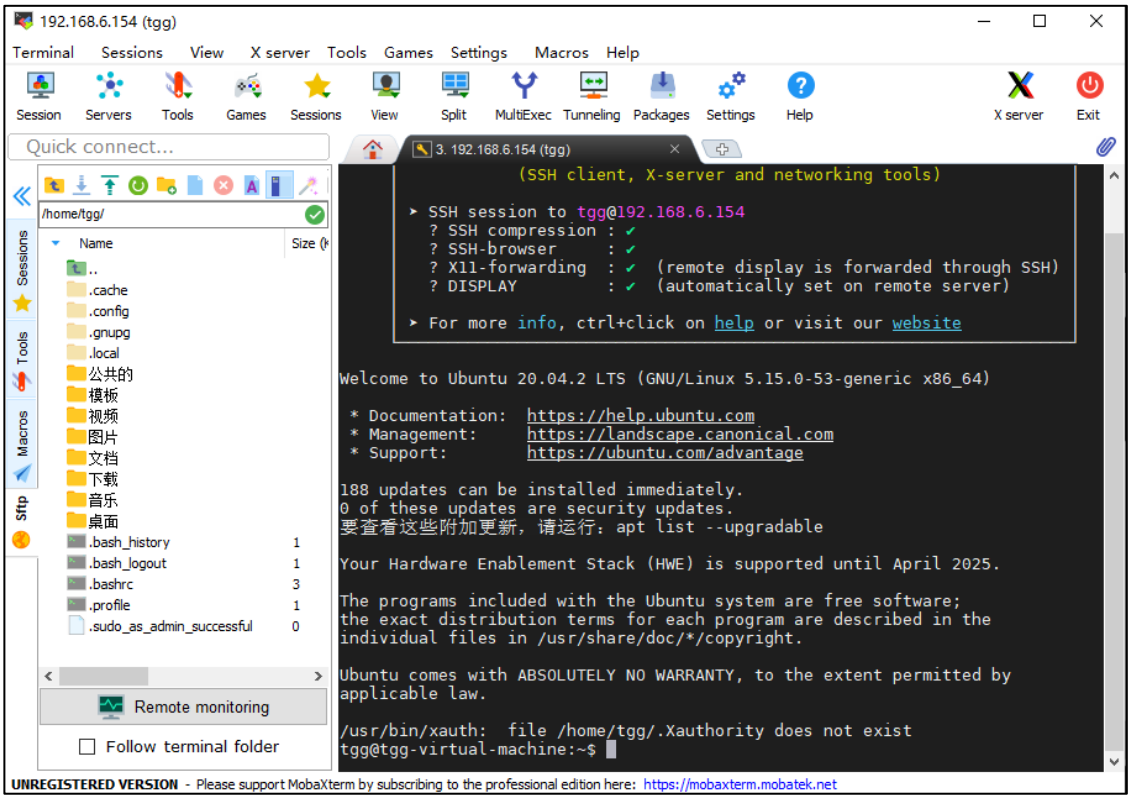


Figure 3.5-19 SSH remote login was successful.

When using SSH for 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 3.5.20:

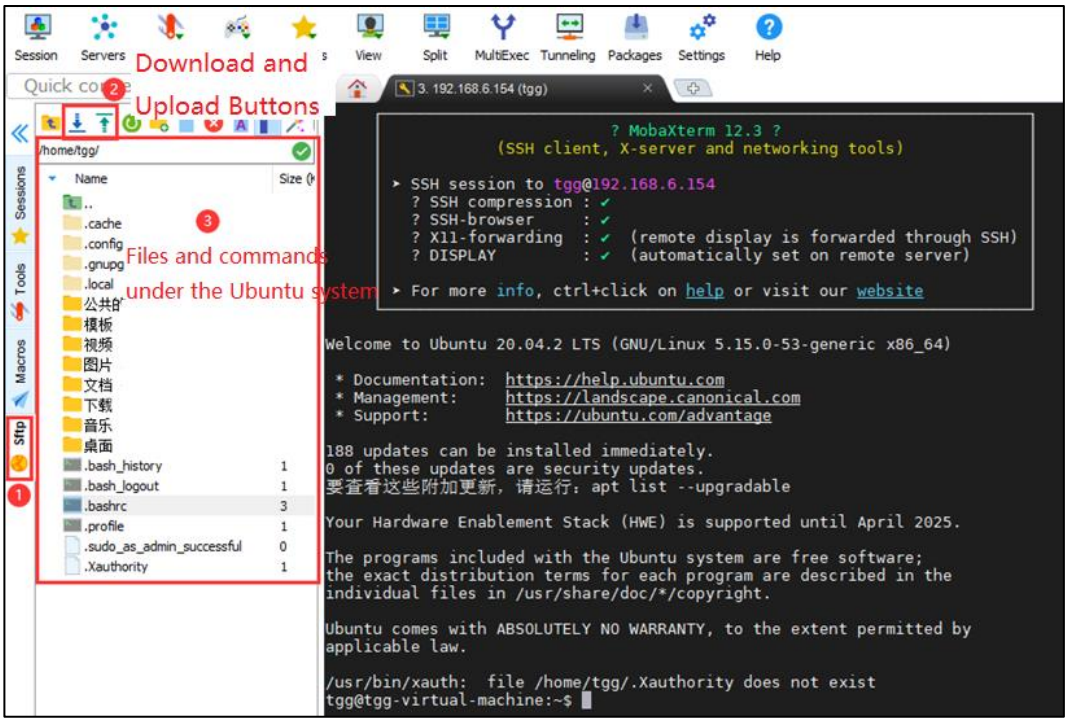


Figure 3.5-20 MobaXterm enables file transfer between Windows and Ubuntu

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.

3.6 Installation of Rockchip USB Driver

During the development and debugging phase, 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; the Rockchip USB driver needs to be installed on Windows to be able to recognize the device.

The Rockchip USB driver installation package has been provided to users in the development board data disk. The path is: Development Board CD-ROM A Drive - Basic Materials -> 04_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 3.6.1 will be obtained:

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

Figure 3.6-1 Unzip DriverAssitant_v5.12.zip

Double-click the DriverInstall.exe file directly and install the Rockchip USB driver following the steps shown in Figures 3.6.2 to 3.6.4:

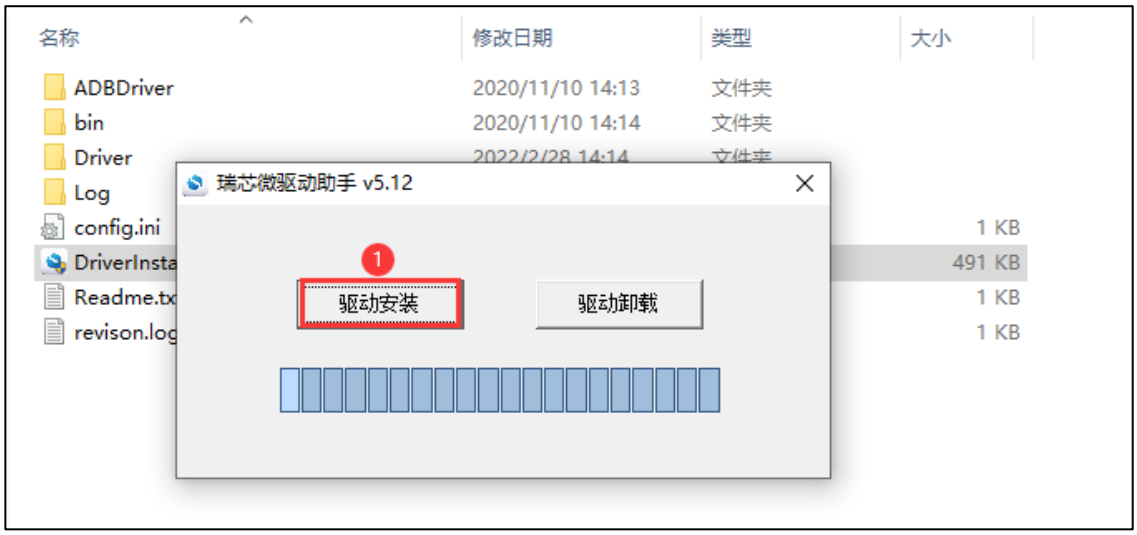


Figure 3.6-2 Install Rockchip USB Driver (1)

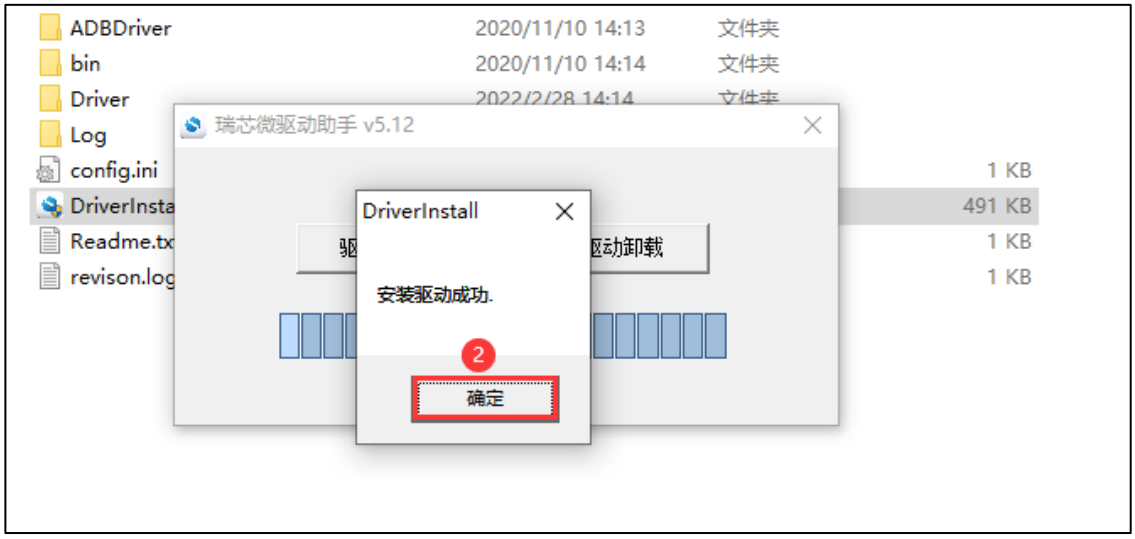


Figure 3.6-3 Install Rockchip USB Driver (2)

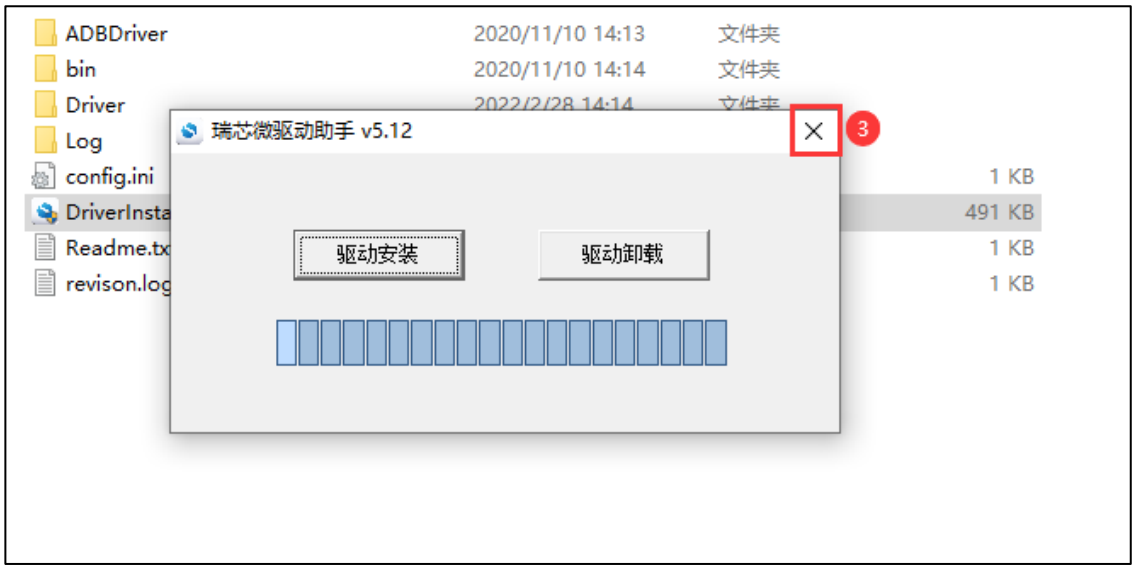


Figure 3.6-4 Install Rockchip USB Driver (3)

3.7 Use 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, through the FactoryTool tool for batch burning (mass production burning tool, supporting USB multi-device burning), and through the Linux_Upgrade_Tool tool in the Ubuntu system, etc. In summary, there are many ways to burn the image, 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.

3.7.1 Flashing Mode Introduction

The various operating modes of the Rockchip platform hardware devices are as shown in the table below. Only when the device is in the Maskrom and Loader mode (and the Type-C port of the development board must be connected to the computer via a USB cable), can the image be flashed or the image on the board be updated.

Pattern	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 there is an issue where it is unable to enter the Loader mode normally, it can also enter the Maskrom mode to burn the image.
Loader	Support	In the Loader mode, image burning, update and upgrade can be performed. Images of specific partitions can be burned separately using the burning tool, facilitating debugging.
Recovery	Not supported	The system initiates recovery startup, mainly for operations such as upgrading and restoring factory settings.
Normal Boot	Not supported	After the system is normally started, it enters a certain mode. It starts by booting the rootfs, loading the rootfs, and most of the development is done in this mode for debugging.

Table 3.7.1 Rockchip Platform Operating Mode Explanation

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

- If the card computer has not burned the image before, it will enter the Maskrom mode after power-on;

- If the card computer has burned the image before, press the UPDATE button on the card computer, then power-on or reset the card computer, and the system will enter the Maskrom mode;
- If the card computer has burned the image before, execute the "rbrom" command in the U-Boot command line to enter the Maskrom mode.

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

- If the card computer has burned the image, press the V+ button (volume+) and then power on or reset the card computer. The system will enter the Loader mode;
- If the card computer has burned the image, execute the "download" command in the U-Boot command line to enter the Loader mode;
- If the card computer 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;
- If the card computer has burned the image, after powering on or resetting the development board, it will normally start and enter the system. The Rockchip development tool (also known as RKDevTool tool) will display "**An ADB device has been found**", then click the "Switch" button to enter the Loader mode, as shown in the following figure.

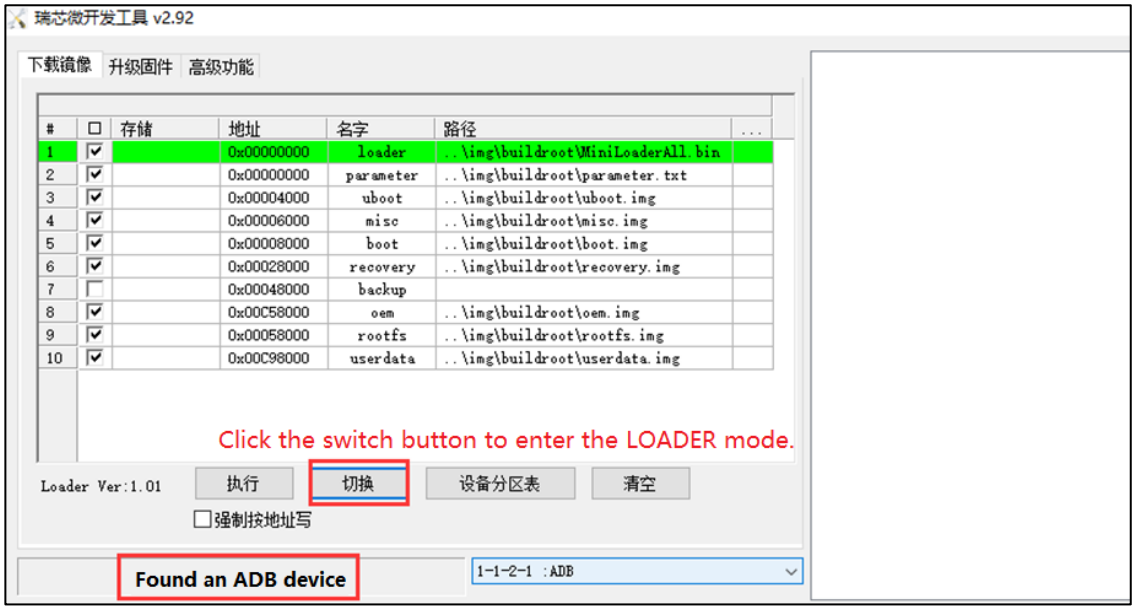


Figure 3.7-1 Enter the Loader mode through the development tools of Rockchip Microelectronics.

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

3.7.2 Usage of RKDevTool tool under Windows

When burning the image on Windows, use the RKDevTool tool. In the documentation of RK, it is generally referred to as the Rockchip Development Tool. This tool is not only used for burning the image, but also has other functions. It is the most commonly used tool during our development and debugging process.

The QuarkPi-CA2 card computer data disk of ALIENTEK has already provided this tool to users. The path is: Development Board CD-ROM A Disk - Basic Materials -> 05_tools -> RKTools ->

Windows -> RKDevTool_Release_v2.92.zip. Of course, this tool can also be found in the RK3588S Buildroot SDK.

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

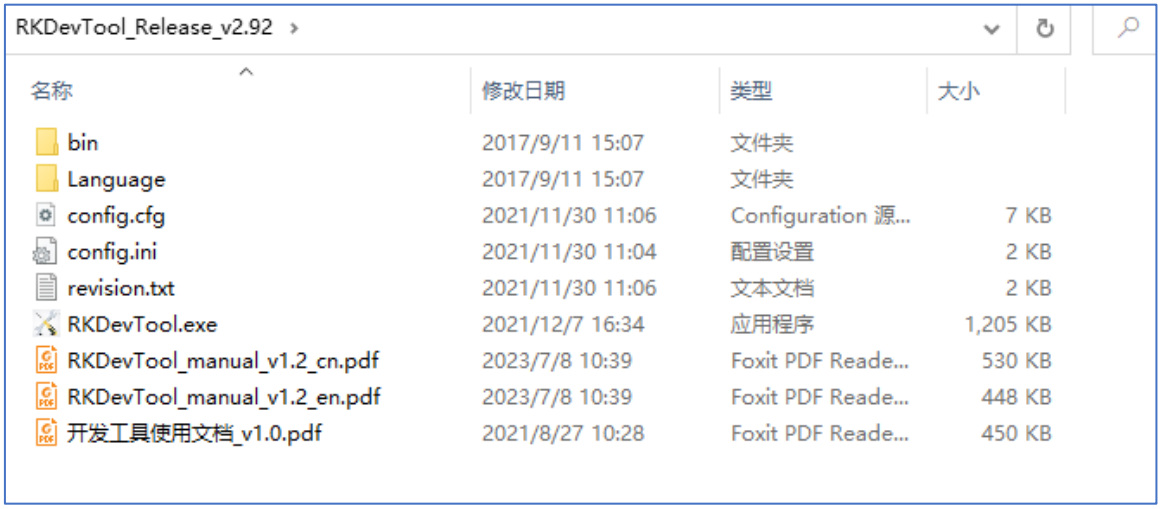


Figure 3.7-2 RKDevTool directory

There are two official documents provided by Rockchip Microelectronics in this directory: "Developer Tools Usage Documentation_v1.0.pdf" and "RKDevTool_manual_v1.2_cn.pdf". These documents provide users with detailed instructions on how to use the RKDevTool tool (Rockchip Microelectronics development tool). You can refer to these two documents for the detailed usage methods of the tool. Here, we are just giving a brief introduction.

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

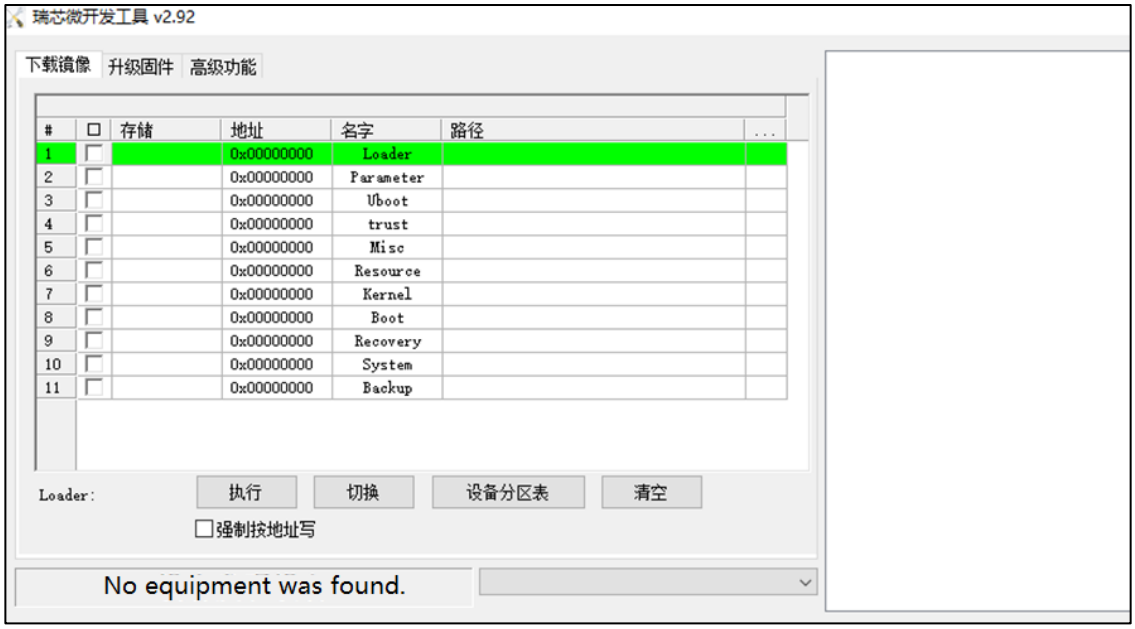


Figure 3.7-3 The interface of the RISC-V development tool

As shown in the above picture, this is a main interface of the RISC-V development tool, RKDevTool. This tool has two methods for image burning:

- Burn individual separate images. After the Linux SDK is compiled, multiple image files will be generated, including: boot.img, MiniLoaderAll.bin, uboot.img, misc.img, rootfs.img, recovery.img, userdata.img, oem.img, and several other image files, among which there is also a partition table file parameter.txt. Using the RKDevTool tool, these images can be burned separately.
- Burn the complete update.img firmware. update.img is not compiled; it is actually a package of multiple images, including boot.img, MiniLoaderAll.bin, uboot.img, misc.img, rootfs.img, recovery.img, etc., which are the individual images; through the tools provided by RK, each individual image can be packaged into an update.img.

Next, let's talk about how to burn in the form of individual images. In Section 3.7.3, users will be introduced to how to burn the update.img firmware.

According to the operation steps shown in Figures 3.7.4 to 3.7.7, import a.cfg configuration file (this configuration file contains the configuration information for the burning process, including the burning address, the path of the image, etc. Of course, you can also manually configure these information; this will be explained later):

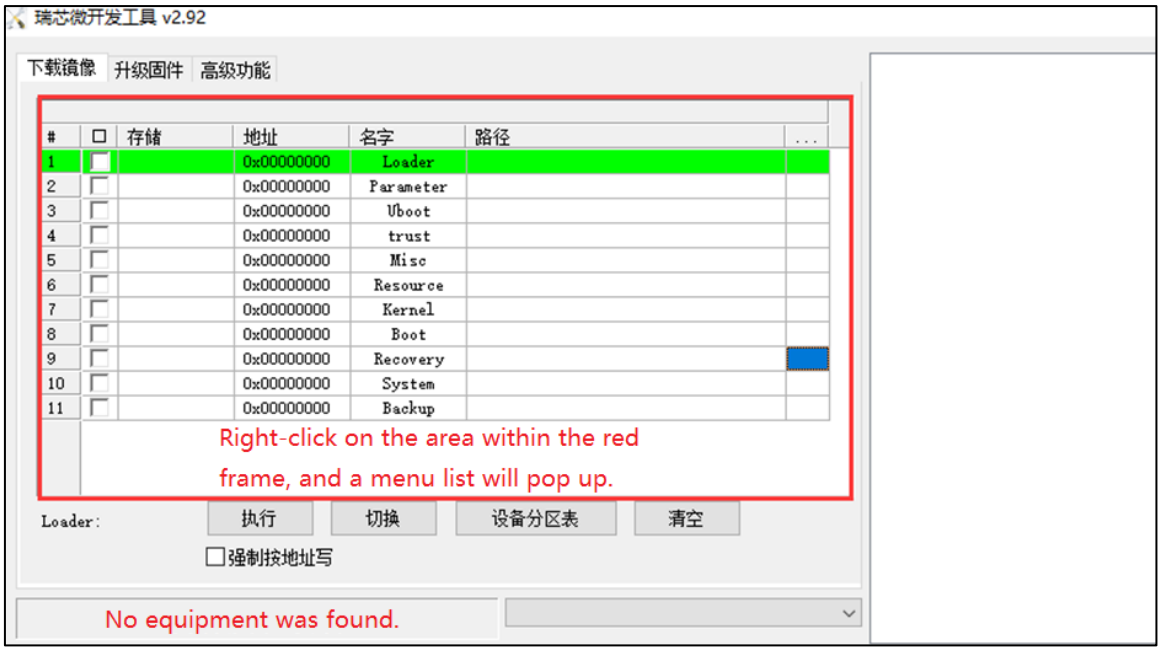


Figure 3.7-4 Import configuration file (1)

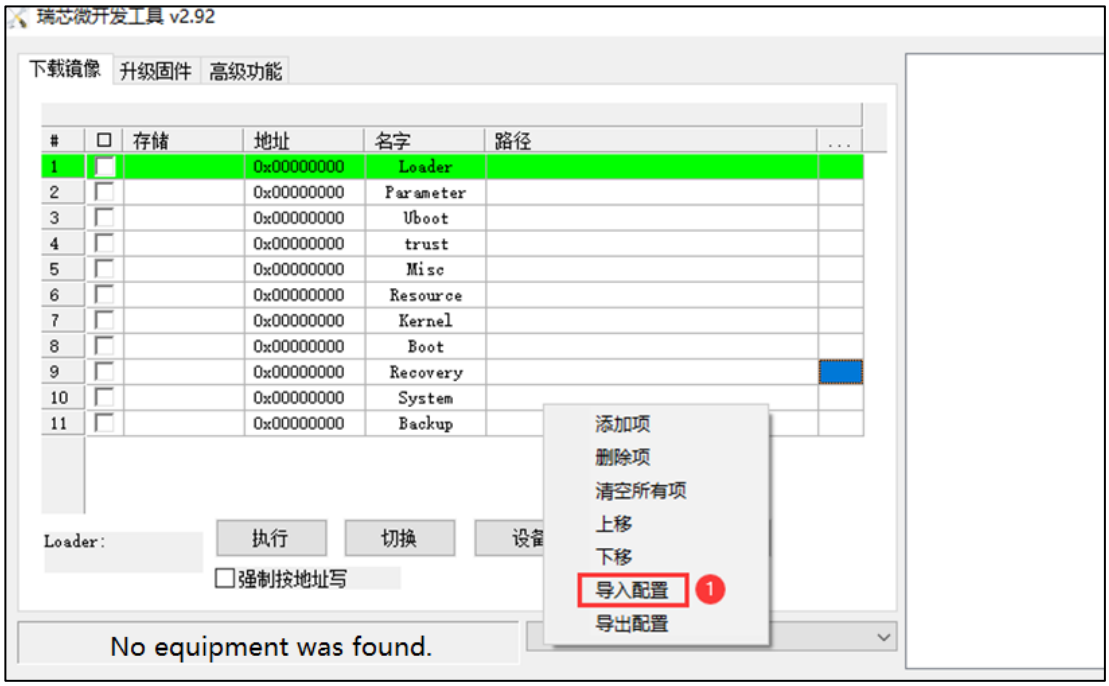


Figure 3.7-5 Import configuration file (2)

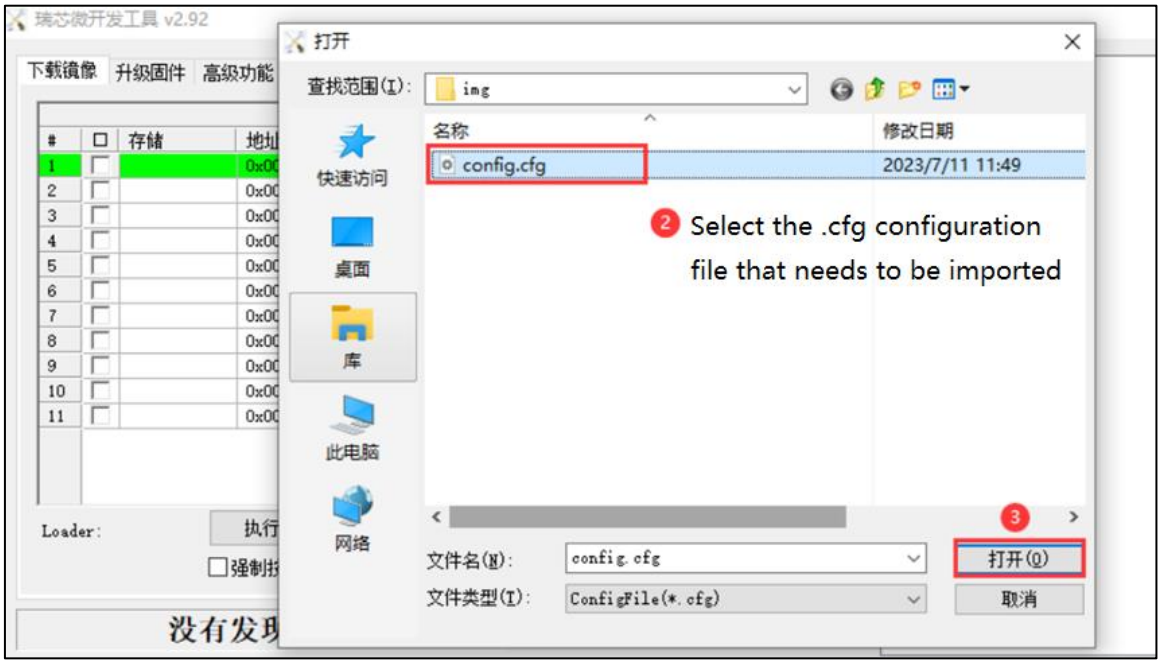


Figure 3.7-6 Import configuration file (3)

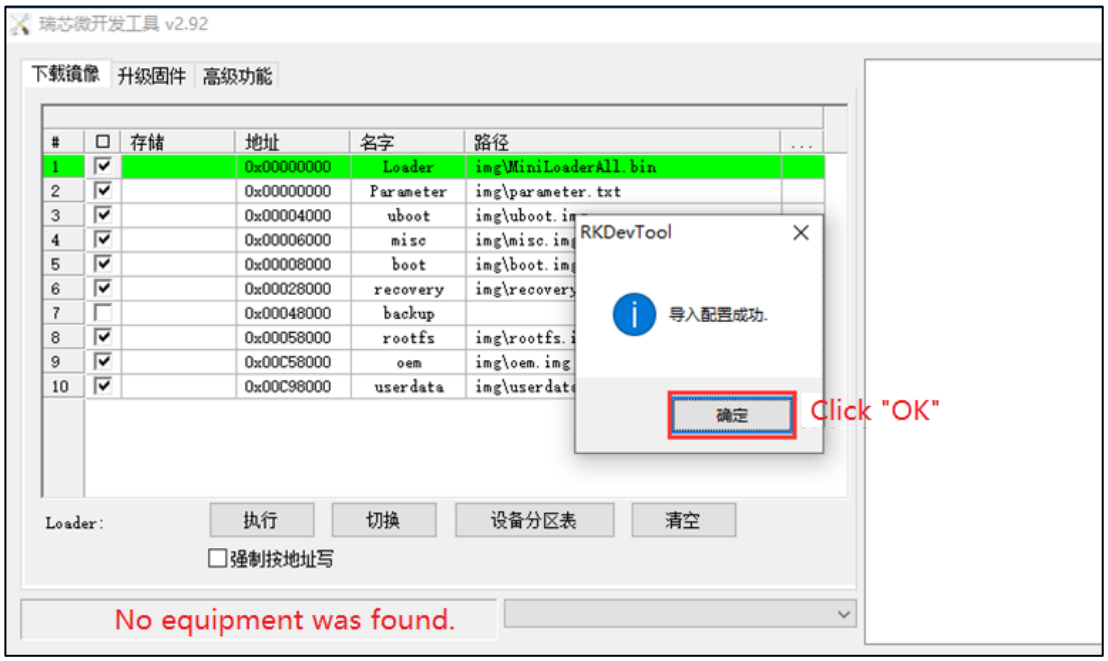


Figure 3.7-7 Import configuration file (4)

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

Figure 3.7-8 Configuration Information Division Explanation

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

② Part represents the burning address (that is, the burning address of the image);

③ Part represents the name of this burning item;

④ Part is used to specify the path where the image is located.

All these configuration information can 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, the next step is to perform the burning operation. Before executing the burning process, the card computer must be in the Maskrom mode or Loader mode. The cards computers that customers receive are usually already burned with the image. They can enter the Loader mode, or they can also enter the Maskrom mode. First, the development board needs to be connected to the power adapter and the TYPE-C0 port, as shown in the following figure:

Figure 3.7-9 Hardware connection diagram of the development board

The power adapter uses a 12V-2.5A power adapter; connect the debugging serial port pins of the card computer to the computer via a USB cable and the serial port module (install the CH343 driver first), and also connect the TYPE-C interface of the development board to the computer using a USB cable.

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

Figure 3.7-10 The development tool of Rockchip Microelectronics detected the loader device.

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

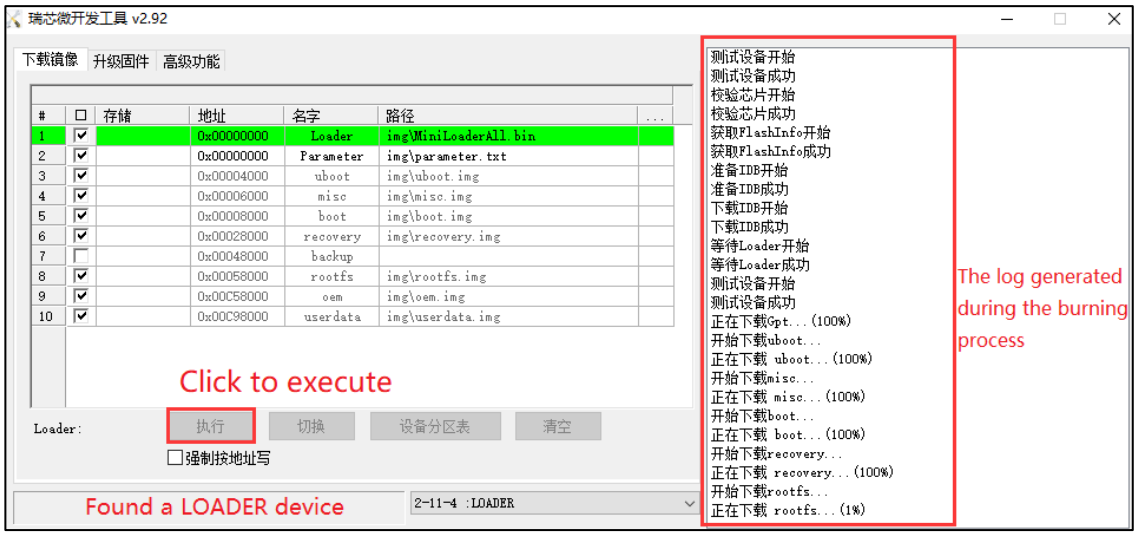


Figure 3.7-11 Perform the burning operation

Of course, it is also possible to perform the burning in Maskrom mode. In Section 3.7.1, we have already introduced how to enter the Maskrom mode. After entering the Maskrom mode, the burning operation is carried out by clicking the "Execute" button.

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 QuarkPi-CA2 card computer 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 documentation "RKDevTool_manual_v1.2_cn.pdf", which contains explanations for several common errors.

3.7.3 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 (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.

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

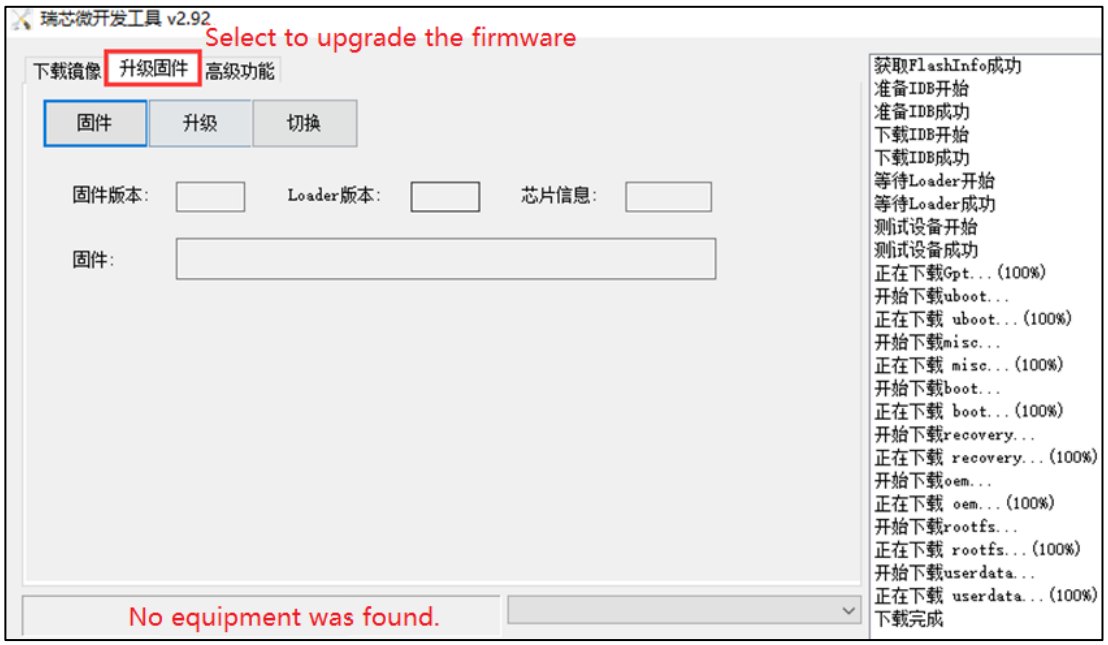


Figure 3.7-12 Upgrade Firmware Tab

Then click the "Firmware" button to select the update.img firmware that we need to upgrade or burn (the update.img firmware is not provided for users in the development board data disk. This is just a demonstration of how to burn the update.img firmware.) After loading the image, some information about this firmware will be displayed:

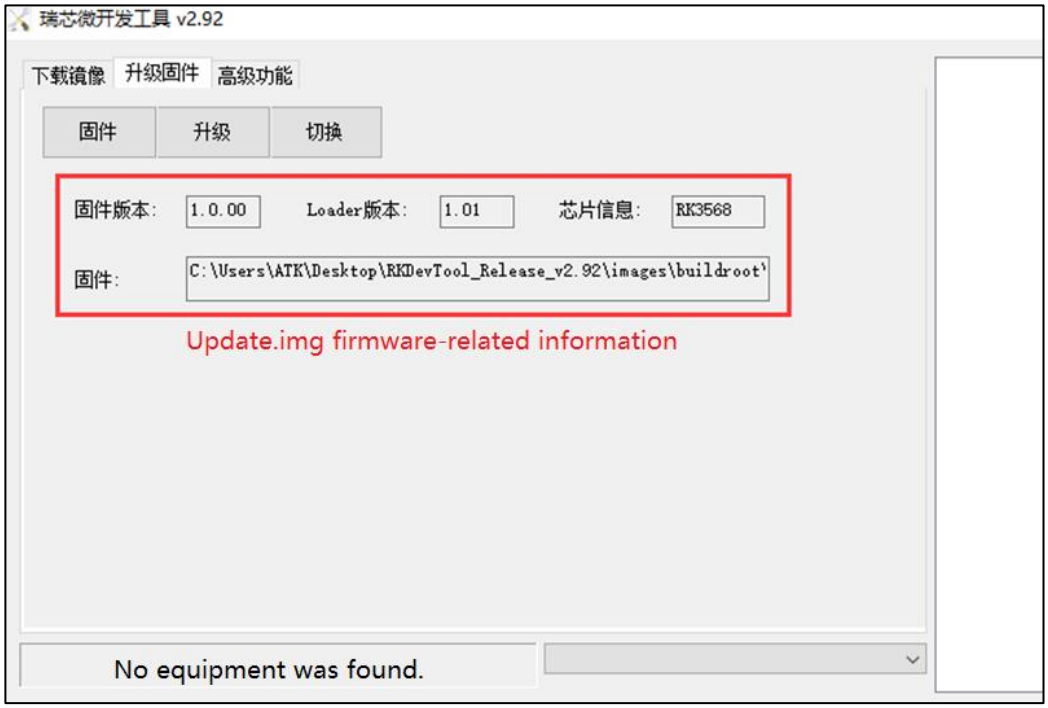


Figure 3.7-13 Import the update.img image

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

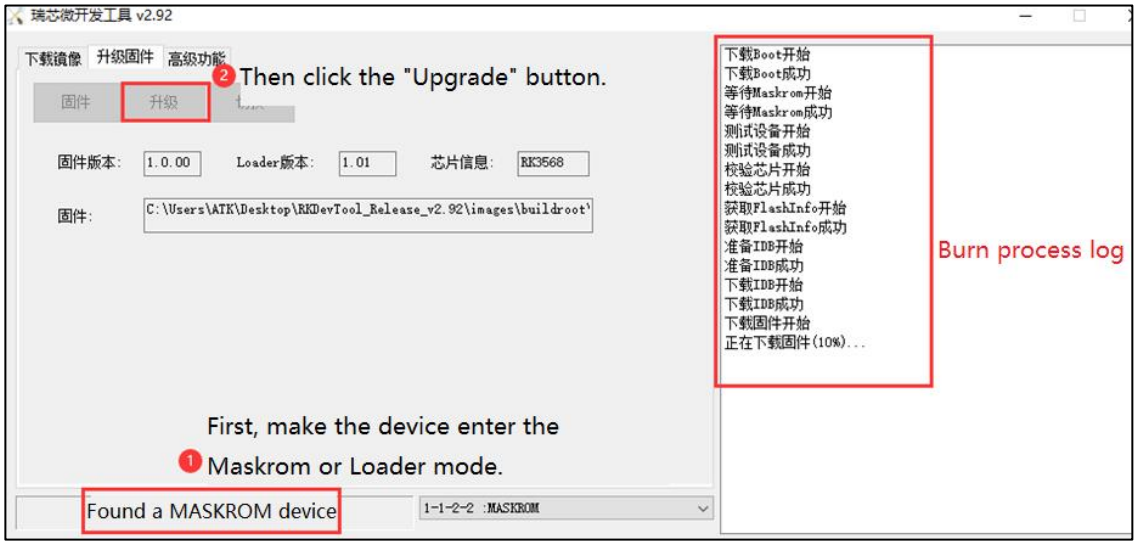


Figure 3.7-14 update.img upgrade

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

3.7.4 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 3.7-15 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:

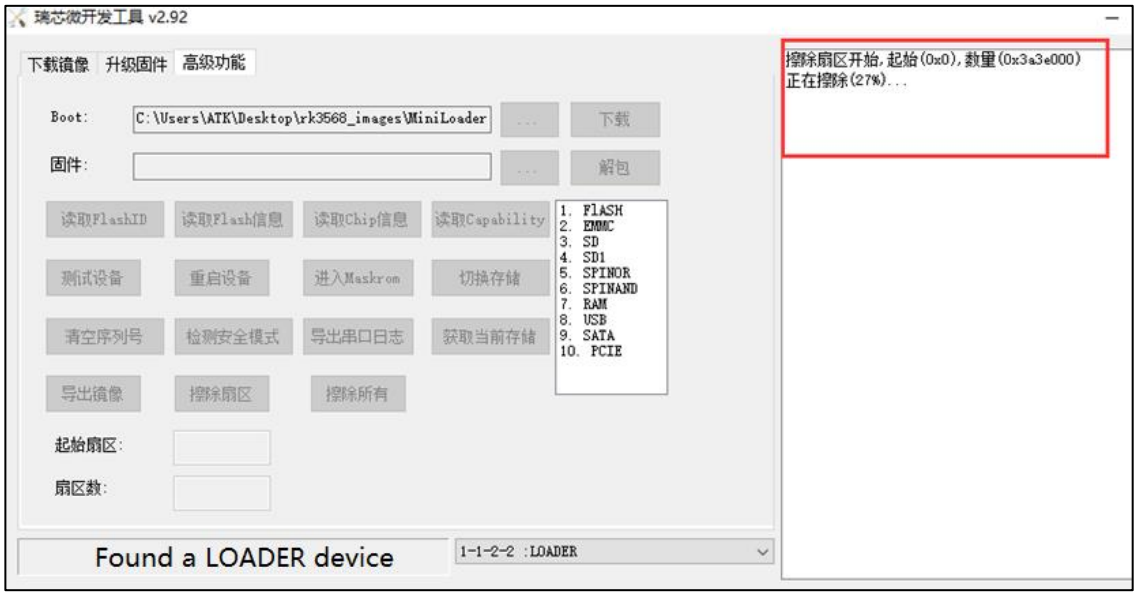


Figure 3.7-16 Erasing all operations

3.7.5 Usage of the Linux_Upgrade_Tool tool under Ubuntu

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

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

This tool is in the SDK. The Linux SDK comes with this burning tool by default. We will introduce it to you later.

3.8 Installation of ADB Tools

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

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

- Running the device's shell (command line);
- Managing port mapping of the emulator or device;
- File upload/download between the computer and the device;
- Installing local apk software onto 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. In this document, users will be introduced how to install the ADB tool on Windows systems and Ubuntu systems, and how to use the ADB tool.

Chapter 4. QuarkPi-CA2 Linux SDK Software Package

This chapter introduces to users the installation and usage methods of the Buildroot SDK software package for the ALIENTEK QuarkPi-CA2 hardware platform. This SDK is applicable to the ALIENTEK QuarkPi-CA2 card computer and all Linux products developed based on this development board; based on this SDK, system customization and application porting development can be effectively achieved, helping users to quickly develop and improve development efficiency!

The QuarkPi-CA2 Linux SDK integrates the Linux Kernel source code, U-Boot source code, and various development tools and documents provided by RK. This SDK supports three root file systems: Buildroot, Yocto, and Debian. The Linux kernel version is 5.10, and the U-Boot version is 2017.09.

This chapter will be divided into the following sections:

1. Installation of QuarkPi-CA2 Linux SDK;
2. Introduction to the SDK software architecture;
3. Full automatic compilation of the SDK;
4. Manual compilation;
5. Cleaning of the SDK;
6. Introduction to the image.

Official reference document of RK:

<SDK>/docs/cn/Rockchip_Developer_Guide_Linux_Software_CN.pdf

4.1 Installing the QuarkPi-CA2 Linux SDK

This section provides instructions for users on how to install the Linux SDK software package for the ALIENTEK QuarkPi-CA2 development board on the Ubuntu system.

4.1.1 Installing 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 packages:

```
sudo apt-get update && sudo apt-get install git ssh make gcc libssl-dev \
liblz4-tool expect expect-dev 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 ncurses-dev \
bzip2 expat gpgv2 cpp-aarch64-linux-gnu libgmp-dev \
libmpc-dev bc python-is-python3 python2
```

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 python2 as the default Python version for the system.

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

Execute the following command to enable DNS support for kgithub.com:

```
sudo sed -i '$a 43.154.68.204\tkgithub.com' /etc/hosts
sudo sed -i '$a 43.155.83.75\traw.kgithub.com objects.githubusercontent.kgithub.com' /etc/hosts
```

4.1.2 Install repo (skip)

Skip this step.

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
```

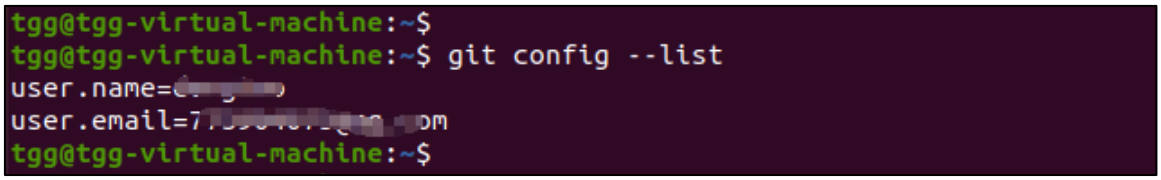


Figure 4.1-1 View git configuration information

4.1.4 Install SDK

Now let's start installing the QuarkPi-CA2 Linux SDK. The SDK compressed file has already been provided to users in the development board data disk. The path is: Development board optical disc B drive - Development environment and SDK -> 02_QuarkPi_CA2_card_computer_SDK -> atk-quarkpi_ca2_linux_release_v1.0_20241217.tar.gz. With the update of the version, the name of the SDK compressed file will also change, but it will always be named as atk-quarkpi_ca2_linux_release_ version _ release date.tar.gz. Each time a new version of the SDK is released, the old version SDK in the network disk 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, and execute the following command to extract it to the atk_quarkpi_ca2_buildroot directory in the home directory:

```
mkdir ~/atk_quarkpi_ca2_buildroot
tar xvf atk-quarkpi_ca2_linux_release_v1.0_20241217.tar.gz -C ~/atk_quarkpi_ca2_buildroot/
```

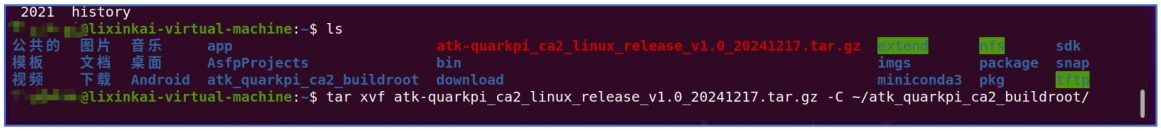


Figure 4.1-2 Unzipping the SDK package

After the unzipping process is completed, a .repo folder will exist in the ~/atk_quarkpi_ca2_buildroot/ directory, as shown in Figure 4.1.3:

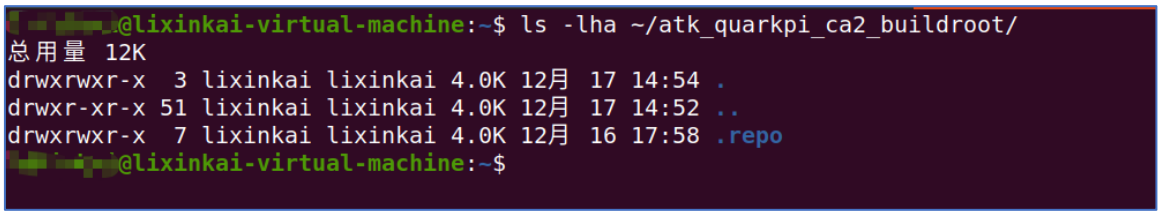


Figure 4.1-3.The .repo folder

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

```
cd ~/atk_quarkpi_ca2_buildroot/
.repo/repo/repo sync -l -j16
```

```
lixinkai@lixinkai-virtual-machine:~$ cd ~/atk_quarkpi_ca2_buildroot/
lixinkai@lixinkai-virtual-machine:~/atk_quarkpi_ca2_buildroot$ .repo/repo/repo sync -l -j16
正在更新文件: 100% (27/27), 完成.
正在更新文件: 100% (13805/13805), 完成.
正在更新文件: 100% (952/952), 完成.
正在更新文件: 100% (235/235), 完成.
正在更新文件: 100% (109/109), 完成.
正在更新文件: 100% (18768/18768), 完成.
正在更新文件: 100% (36099/36099), 完成.
正在更新文件: 100% (6217/6217), 完成.
正在更新文件: 100% (263/263), 完成.
正在更新文件: 100% (5872/5872), 完成.
正在更新文件: 100% (740/740), 完成.o/poky正在更新文件:   7% (83/1178)
正在更新文件: 100% (959/959), 完成.
正在更新文件: 100% (1178/1178), 完成.
正在更新文件: 100% (6242/6242), 完成.
正在更新文件: 100% (2131/2131), 完成.
正在更新文件: 100% (236/236), 完成.
正在更新文件: 100% (22/22), 完成.
正在更新文件: 100% (2010/2010), 完成.
正在更新文件: 100% (98/98), 完成.
正在更新文件: 100% (17/17), 完成.
正在更新文件: 100% (3666/3666), 完成.
正在更新文件: 100% (275/275), 完成.
正在更新文件: 100% (164/164), 完成.
正在更新文件: 100% (5268/5268), 完成.
正在更新文件: 100% (742/742), 完成.
正在更新文件: 100% (81899/81899), 完成.
正在更新文件: 100% (5939/5939), 完成.
正在更新文件: 100% (5722/5722), 完成.
正在更新文件: 100% (6910/6910), 完成.
正在更新文件: 100% (166/166), 完成.
正在更新文件: 100% (37171/37171), 完成.ga正在更新文件:  64% (24069/37171)
正在更新文件: 100% (1492/1492), 完成.
正在更新文件: 100% (264/264), 完成.
Checking out: 100% (73/73), done in 7m26.846s
repo sync has finished successfully.
```

Figure 4.1-4 Check out the source code

The entire process will take a relatively long time. Please be patient and wait until the synchronization is completed. After the synchronization is finished, the contents in the ~/atk_quarkpi_ca2_buildroot/ directory will be as shown in Figure 4.1.5:

```
lixinkai@lixinkai-virtual-machine:~/atk_quarkpi_ca2_buildroot$ ls
app        build.sh  device   envsetup.sh  kernel    prebuilts  rkflash.sh  tools    uefi
buildroot  debian    docs     external     Makefile  rkbin      rtos        u-boot   yocto
lixinkai@lixinkai-virtual-machine:~/atk_quarkpi_ca2_buildroot$
```

Figure 4.1-5 Linux SDK project directory

At this point, the QuarkPi-CA2 Linux SDK has been successfully installed.

4.1.5 SDK issue feedback

During the use and development process, if users find any problems with the SDK, software bugs, specific technical issues, 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 arrange to handle and track the issue accordingly.

4.1.6 SDK Optimization

If the disk space on the Ubuntu system is limited, the SDK can be optimized by deleting the .repo folder (a hidden folder that can be viewed using the command "ls -a") in the SDK root directory, which can free up some disk space. Please note that the .repo folder stores the git commit records of all 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 for users to delete it.

4.2 Introduction to the SDK Software Architecture

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

4.2.1 Introduction to SDK Project Directory Structure

The root directory of the SDK 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 consists of 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 Linux 5.10 kernel.
- .prebuilts: Stores cross-compiler 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 4.2.1. It is divided from bottom to top into four layers: Bootloader, Linux Kernel, Libraries, and Applications. The contents of each layer are as follows:

- The BootLoader layer mainly provides the 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; it mainly manages device hardware resources through the Linux kernel, such as CPU scheduling, cache, memory, I/O, etc.
- The Libraries layer corresponds to general embedded systems and is equivalent to 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 capabilities of the system to the end users.

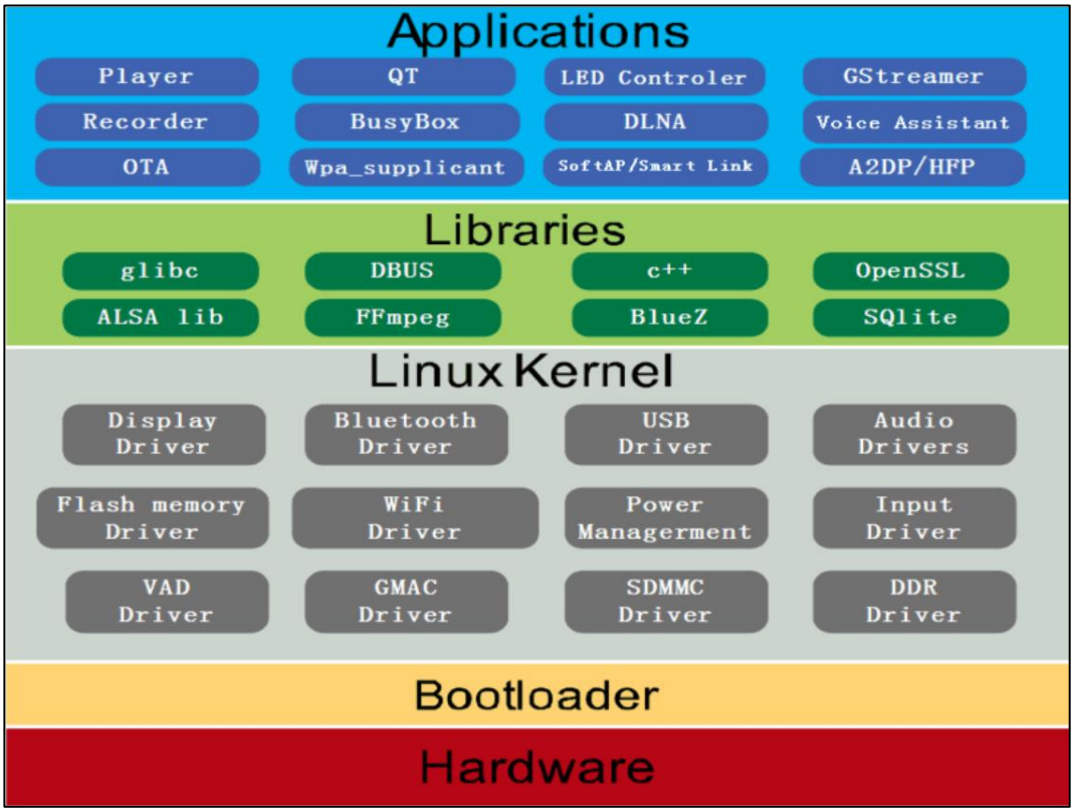


Figure 4.2-1 SDK Software Block Diagram

4.2.3 Query of SDK Version

The version of the QuarkPi-CA2 Linux SDK provided by ALIENTEK can be divided into RK version and ATK version. The RK version refers to the version number defined by Rockchip (Ricehu) when they released the SDK; while the ATK version refers to the version number defined by the Linux technology team of ALIENTEK when they released the SDK. Each time a new version of the SDK is released, there will be a corresponding version number.

From the file name of the SDK compressed package provided by us, you can know the corresponding ATK version and release date; you can also execute the following command in the root directory of the Linux SDK source code to query the ATK version of the SDK:

```
realpath .repo/manifests/atk-quarkpi_ca2_linux5.10_release.xml
```

```
lixinkai@lixinkai-virtual-machine:~/atk_quarkpi_ca2_buildroot$ realpath .repo/manifests/atk-quarkpi_ca2_linux5.10_release.xml
/home/lixinkai/atk_quarkpi_ca2_buildroot/.repo/manifests/rk3588_linux/atk-quarkpi_ca2_linux_release_v1.0.0_20241216.xml
lixinkai@lixinkai-virtual-machine:~/atk_quarkpi_ca2_buildroot$
```

Figure 4.2-2 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 2024-12-16.

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/rk3588_linux/rk3588_linux_release*
```

```
lixinkai@lixinkai-virtual-machine:~/atk_quarkpi_ca2_buildroot$ ls .repo/manifests/rk3588_linux/rk3588_linux_release*
.repo/manifests/rk3588_linux/rk3588_linux_release_v1.5.0_20240620.xml
lixinkai@lixinkai-virtual-machine:~/atk_quarkpi_ca2_buildroot$
```

Figure 4.2-3 Query the RK version of the SDK

As can be seen from the figure, the RK version of the SDK is V1.5.0, and the release date is June 20, 2024.

4.3 SDK Automatic Compilation

This section provides users with instructions on how to automatically compile the entire QuarkPi-CA2 Linux SDK in one step. First, enter the root directory of the SDK. Before compiling, execute the following command to specify the board-level configuration file:

```
./build.sh lunch
```

```
lixinkai@lixinkai-virtual-machine:~/atk_quarkpi_ca2_buildroot$ ./build.sh lunch

############### Rockchip Linux SDK ###############

Manifest: atk-quarkpi_ca2_linux_release_v1.0.0_20241216.xml
Version: linux-5.10-gen-rkr8

Log colors: message notice warning error fatal

Parsing supported commands...

Log saved at /home/lixinkai/atk_quarkpi_ca2_buildroot/output/sessions/2024-12-17_15-41-23
Pick a defconfig:

1. rockchip_defconfig
2. alientek_quarkpi_ca2_defconfig        The configuration file corresponding to the card computer
3. alientek_rk3588_defconfig
4. rockchip_rk3588_evb1_lp4_v10_defconfig
5. rockchip_rk3588_evb7_v11_defconfig
6. rockchip_rk3588_ipc_evb1_v10_defconfig
7. rockchip_rk3588_multi_ipc_evb1_v10_defconfig
8. rockchip_rk3588s_evb1_lp4x_v10_defconfig
Which would you like? [1]: 2    Enter the corresponding number
```

Figure 4.3-1 Select the board configuration file

Enter the corresponding number for "alientek_quarkpi_ca2_defconfig" (the sequence number may change in subsequent updates, use the file name of the configuration file as the reference), and then press the Enter key to confirm:

```
Log colors: message notice warning error fatal

Parsing supported commands...

Log saved at /home/lixinkai/atk_quarkpi_ca2_buildroot/output/sessions/2024-12-17_15-41-23
Pick a defconfig:

1. rockchip_defconfig
2. alientek_quarkpi_ca2_defconfig
3. alientek_rk3588_defconfig
4. rockchip_rk3588_evb1_lp4_v10_defconfig
5. rockchip_rk3588_evb7_v11_defconfig
6. rockchip_rk3588_ipc_evb1_v10_defconfig
7. rockchip_rk3588_multi_ipc_evb1_v10_defconfig
8. rockchip_rk3588s_evb1_lp4x_v10_defconfig
Which would you like? [1]: 2
Switching to defconfig: /home/lixinkai/atk_quarkpi_ca2_buildroot/device/rockchip/.chip/alientek_quarkpi_ca2_defconfig
mkdir -p /home/lixinkai/atk_quarkpi_ca2_buildroot/output/kconf/lxdialog
make CC="gcc" HOSTCC="gcc" \
    obj=/home/lixinkai/atk_quarkpi_ca2_buildroot/output/kconf -C /home/lixinkai/atk_quarkpi_ca2_buildroot/device/rockchip/common/kco
nfig -f Makefile.br conf
make[1]: Entering directory '/home/lixinkai/atk_quarkpi_ca2_buildroot/device/rockchip/common/kconfig'
gcc -D_DEFAULT_SOURCE -D_XOPEN_SOURCE=600 -DCURSES_LOC="<ncurses.h>" -DNCURSES_WIDECHAR=1 -DLOCALE  -I/home/lixinkai/atk_quarkpi_ca2
_buildroot/output/kconf -DCONFIG_=\"\"  -MM *.c > /home/lixinkai/atk_quarkpi_ca2_buildroot/output/kconf/.depend 2>/dev/null || :
gcc -D_DEFAULT_SOURCE -D_XOPEN_SOURCE=600 -DCURSES_LOC="<ncurses.h>" -DNCURSES_WIDECHAR=1 -DLOCALE  -I/home/lixinkai/atk_quarkpi_ca2
_buildroot/output/kconf -DCONFIG_=\"\"   -c conf.c -o /home/lixinkai/atk_quarkpi_ca2_buildroot/output/kconf/conf.o
gcc -D_DEFAULT_SOURCE -D_XOPEN_SOURCE=600 -DCURSES_LOC="<ncurses.h>" -DNCURSES_WIDECHAR=1 -DLOCALE  -I/home/lixinkai/atk_quarkpi_ca2
_buildroot/output/kconf -DCONFIG_=\"\"  -I. -c /home/lixinkai/atk_quarkpi_ca2_buildroot/output/kconf/zconf.tab.c -o /home/lixinkai/a
tk_quarkpi_ca2_buildroot/output/kconf/zconf.tab.o
gcc -D_DEFAULT_SOURCE -D_XOPEN_SOURCE=600 -DCURSES_LOC="<ncurses.h>" -DNCURSES_WIDECHAR=1 -DLOCALE  -I/home/lixinkai/atk_quarkpi_ca2
_buildroot/output/kconf -DCONFIG_=\"\"   /home/lixinkai/atk_quarkpi_ca2_buildroot/output/kconf/conf.o /home/lixinkai/atk_quarkpi_ca2
_buildroot/output/kconf/zconf.tab.o  -o /home/lixinkai/atk_quarkpi_ca2_buildroot/output/kconf/conf
rm /home/lixinkai/atk_quarkpi_ca2_buildroot/output/kconf/zconf.tab.c
make[1]: Leaving directory '/home/lixinkai/atk_quarkpi_ca2_buildroot/device/rockchip/common/kconfig'
#
# configuration written to /home/lixinkai/atk_quarkpi_ca2_buildroot/output/.config
#
Using current kernel version(5.10)
lixinkai@lixinkai-virtual-machine:~/atk_quarkpi_ca2_buildroot$
```

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

Tips: The above operation process can be simplified as the following command:

```
./build.sh alientek_quarkpi_ca2_defconfig
```

"build.sh" is a compiled script packaged by RK. By using this script, users can easily and quickly build system image files and perform packaging operations on the images. It can not only automatically compile the entire Linux SDK in one step, but also separately compile U-Boot, Linux Kernel, buildroot, etc., which is very convenient!

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

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

```
lixinkai@lixinkai-virtual-machine:~/atk_quarkpi_ca2_buildroot$ ./build.sh -h

############### Rockchip Linux SDK ###############

Manifest: atk-quarkpi_ca2_linux_release_v1.0.0_20241216.xml
Version: linux-5.10-gen-rkr8

Log colors: message notice warning error fatal

Usage: build.sh [OPTIONS]
Available options:
chip[:<chip>[:<config>]]                choose chip
defconfig[:<config>]                    choose defconfig
 *_defconfig                            switch to specified defconfig
    available defconfigs:
        alientek_quarkpi_ca2_defconfig
        alientek_rk3588_defconfig
        rockchip_defconfig
        rockchip_rk3588_evb1_lp4_v10_defconfig
        rockchip_rk3588_evb7_v11_defconfig
        rockchip_rk3588_ipc_evb1_v10_defconfig
        rockchip_rk3588_multi_ipc_evb1_v10_defconfig
        rockchip_rk3588s_evb1_lp4x_v10_defconfig
 olddefconfig                           resolve any unresolved symbols in .config
 savedefconfig                          save current config to defconfig
 menuconfig                             interactive curses-based configurator
config                                  modify SDK defconfig
print-parts                             print partitions
list-parts                              alias of print-parts
mod-parts                               interactive partition table modify
edit-parts                              edit raw partitions
new-parts:<offset>:<name>:<size>...     re-create partitions
insert-part:<idx>:<name>[:<size>]       insert partition
del-part:(<idx>|<name>)                 delete partition
move-part:(<idx>|<name>):<idx>          move partition
rename-part:(<idx>|<name>):<name>       rename partition
resize-part:(<idx>|<name>):<size>       resize partition
misc                                    pack misc image
kernel[:dry-run]                        build kernel
```

```
debian                                  build debian rootfs
recovery                                build recovery
security-createkeys                     create keys for security
security-misc                           build misc with system encryption key
security-ramboot                        build security ramboot
security-system                         build security system
loader[:dry-run]                        build loader (u-boot)
uboot[:dry-run]                         build u-boot
u-boot[:dry-run]                        alias of uboot
uefi[:dry-run]                          build uefi
extra-parts                             pack extra partition images
firmware                                pack and check firmwares
edit-package-file                       edit package-file
edit-ota-package-file                   edit package-file for OTA
updateimg                               build update image
ota-updateimg                           build update image for OTA
all                                     build images
release                                 release images and build info
all-release                             build and release images
shell                                   setup a shell for developing
cleanall                                cleanup
clean[:module[:module]]...              cleanup modules
    available modules:
        all
        amp
        config
        extra-parts
        firmware
        kernel
        loader
        misc
        recovery
        rootfs
        security
        updateimg
post-rootfs <rootfs dir>                trigger post-rootfs hook scripts
help                                    usage

Default option is 'all'.
lixinkai@lixinkai-virtual-machine:~/atk_quarkpi_ca2_buildroot$
```

Figure 4.3-3 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:

Parameters of the build.sh script	Explain	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 in 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 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 open-source libraries 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 address of the third-party open-source libraries 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 copy the required third-party open-source libraries to the SDK in advance.

The required third-party open-source libraries have already been provided to users in the user's data disk of the ALIENTEK QuarkPi-CA2 card computer. The specific path is: Development board optical disc B drive - Development environment and SDK -> 02, QuarkPi_CA2 card computer SDK -> 02, buildroot_sdk -> dl.tgz. First, copy this compressed file to the home directory of the Ubuntu system, as shown below:

```
lixinkai@lixinkai-virtual-machine:~$ ls -l dl.tar.gz
-rw------- 1 lixinkai lixinkai 1506176005 12月 17 15:52 dl.tar.gz
lixinkai@lixinkai-virtual-machine:~$
lixinkai@lixinkai-virtual-machine:~$
```

Figure 4.3-4 The dl.tgz compressed file

Execute the following command to decompress it to the buildroot directory under the SDK root directory:

```
tar xvf dl.tar.gz -C ./atk_quarkpi_ca2_buildroot/buildroot/
```

```
lixinkai@lixinkai-virtual-machine:~$
lixinkai@lixinkai-virtual-machine:~$ tar xvf dl.tar.gz -C ./atk_quarkpi_ca2_buildroot/buildroot/
dl/
dl/numactl/
```

Figure 4.3-5 Extract dl.tgz

After the extraction is completed, you can enter the directory ~/atk_quarkpi-ca2_buildroot/buildroot/. In this directory, there will be a dl directory. The contents stored in the dl directory are the third-party open-source libraries, as shown below:

```
lixinkai@lixinkai-virtual-machine:~$
lixinkai@lixinkai-virtual-machine:~$ cd ~/atk_quarkpi_ca2_buildroot/buildroot/dl/
lixinkai@lixinkai-virtual-machine:~/atk_quarkpi_ca2_buildroot/buildroot/dl$
lixinkai@lixinkai-virtual-machine:~/atk_quarkpi_ca2_buildroot/buildroot/dl$ ls
acl                flac               libevdev           meson              python-protobuf          sdl2
alsa-lib           flex               libevent           minicom            python-psutil            sdl2_gfx
alsa-plugins       fluidsynth         libffi             mjpegtools         python-ruamel-yaml       sdl2_image
alsa-ucm-conf      font-awesome       libfribidi         mmc-utils          python-ruamel-yaml-clib  sdl2_mixer
alsa-utils         fontconfig         libfuse3           mpc                python-setuptools        sdl2_net
android-tools      freetype           libglib2           mpdecimal          python-six               sdl2_ttf
at-spi2-core       gawk               libgudev           mpfr               qemu                     source-han-sans-cn
attr               gcc                libinput           mpg123             qextserialport           sox
autoconf           gdb                libjpeg            mtdev              qjson                    sqlite
autoconf-archive   gdk-pixbuf         liblockfile        ncurses            qt53d                    squashfs
automake           gesftpserver       libmad             netstat-nat        qt5base                  strace
bash               gettext-gnu        libmpeg2           nettle             qt5charts                stress
binutils           gettext-tiny       libnl              nfs-utils          qt5connectivity          stressapptest
bison              ghostscript-fonts  libnspr            nginx              qt5declarative           stress-ng
bitstream-vera     glibc              libnss             nginx-rtmp         qt5enginio               tiff
bluez5_utils       glmark2            libogg             ninja              qt5graphicaleffects      unixbench
bluez-alsa         gmp                libopenssl         noto-sans-sc       qt5imageformats          usbmount
busybox            gnutls             libpng             ntfs-3g            qt5location              util-linux
bzip2              gperf              libpthread-stubs   ntp                qt5lottie                valgrind
cairo              gst1-plugins-bad   libsamplerate      numactl            qt5mqtt                  vim
cantarell          gst1-plugins-base  libsndfile         opencv4            qt5multimedia            vsftpd
```

Figure 4.3-6 The third-party open-source libraries under the dl directory

Next, enter the root directory of the SDK and execute the following command to compile the entire Linux SDK:

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

```
lixinkai@lixinkai-virtual-machine:~/atk_quarkpi_ca2_buildroot$ ./build.sh all

############### Rockchip Linux SDK ###############

Manifest: atk-quarkpi_ca2_linux_release_v1.0.0_20241216.xml
Version: linux-5.10-gen-rkr8

Log colors: message notice warning error fatal

Log saved at /home/lixinkai/atk_quarkpi_ca2_buildroot/output/sessions/2024-12-17_15-55-41
Using last kernel version(5.10)

========================================
           Final configs
========================================
RK_BOOT_FIT_ITS=/home/lixinkai/atk_quarkpi_ca2_buildroot/device/rockchip/.chip/boot.its
RK_BOOT_IMG=boot.img
RK_BUILDROOT=y
RK_BUILDROOT_CFG=rockchip_atk_quarkpi_ca2
RK_CHIP=rk3588
RK_CHIP_FAMILY=rk3588
RK_DEBIAN=y
RK_DEBIAN_ARCH=arm64
RK_DEBIAN_BULLSEYE=y
RK_DEBIAN_MIRROR=mirrors.ustc.edu.cn
RK_DEBIAN_VERSION=bullseye
RK_DEFCONFIG=alientek_quarkpi_ca2_defconfig
RK_EXTRA_PARTITION_NUM=2
RK_EXTRA_PARTITION_STR=oem:oem:/oem:ext4:defaults:normal:auto:@userdata:userdata:/userdata:ext4:defaults:normal:auto:@@@
RK_KERNEL=y
RK_KERNEL_ARCH=arm64
RK_KERNEL_CFG=rockchip_linux_defconfig
RK_KERNEL_DTS=kernel/arch/arm64/boot/dts/rockchip/rk3588s-atk-quarkpi-ca2.dts
RK_KERNEL_IMG=kernel/arch/arm64/boot/Image
RK_KERNEL_VERSION=5.10
```

Figure 4.3-7 Automatically compile the entire SDK

The entire compilation process will take a very long time. The duration of the compilation 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.

During the compilation process, some problems may arise, causing the compilation to fail! After all, everyone's development environment has some differences, so everyone should try to follow the instructions in this document, including the selection of the Ubuntu version, and the dependent software packages required in Section 4.1.1. In addition, ensure that the memory of Ubuntu is at least 16GB or more. If the compilation fails due to insufficient memory, you can expand the memory again and try compiling; if the computer hardware conditions do not allow it, and it is no longer possible to further expand the memory of the virtual machine Ubuntu, at this time, we can try to expand the swap space of Ubuntu. The expansion method can be referred to: Development Board CD-ROM A Disk - Basic Materials -> 10_user_manual -> 03, Auxiliary Documents -> [ALIENTEK] Ubuntu Expansion of Swap Partition.pdf. In addition, the disk space of Ubuntu should be guaranteed to have at least 150 to 200G.

If no unexpected situations occur, the compilation will be successful, as shown in Figure 4.3.8:

```
==========================================
          Start packing linux-headers-aarch64
==========================================
+ mkdir -p /home/lixinkai/atk_quarkpi-ca2_buildroot/output/linux-headers
+ rm -rf /home/lixinkai/atk_quarkpi-ca2_buildroot/output/linux-headers/linux-kbuild-aarch64 /home/lixinkai/atk_quarkpi-ca2_buildroot/output/lin
linux-headers-aarch64.tar.gz
+ ln -rsf /home/lixinkai/atk_quarkpi-ca2_buildroot/device/rockchip/common/linux-kbuild/aarch64/linux-kbuild-5.10 /home/lixinkai/atk_quarkpi-ca2
output/linux-headers/linux-kbuild-aarch64
+ cd /home/lixinkai/atk_quarkpi-ca2_buildroot/kernel
{
        # Based on kernel/scripts/package/builddeb
        find . arch/arm64 -maxdepth 1 -name Makefile\*
        find include -type f -o -type l
        find arch/arm64 -name module.lds -o -name Kbuild.platforms -o -name Platform
        find $(find arch/arm64 -name include -o -name scripts -type d) -type f
        find arch/arm64/include Module.symvers -type f
        echo .config
} | tar --no-recursion --ignore-failed-read -T -        -cf "/home/lixinkai/atk_quarkpi-ca2_buildroot/output/linux-headers/linux-headers-aarch64

        # Pack kbuild
        tar -uf "/home/lixinkai/atk_quarkpi-ca2_buildroot/output/linux-headers/linux-headers-aarch64.tar" -C "/home/lixinkai/atk_quarkpi-ca2_bu
put/linux-headers/linux-kbuild-aarch64" scripts/ tools/
+ cd /home/lixinkai/atk_quarkpi-ca2_buildroot
Unpacking /home/lixinkai/atk_quarkpi-ca2_buildroot/output/linux-headers/linux-headers-aarch64.tar ...
Debian control file:
Package: linux-headers-5.10-arm64
Source: linux-rockchip (5.10)
Version: 5.10-rockchip
Architecture: aarch64
Section: kernel
Priority: optional
Multi-Arch: foreign
Maintainer: Tao Huang <huangtao@rock-chips.com>
Homepage: https://www.kernel.org/
Description: Kbuild and headers for Rockchip Linux 5.10 arm64 configuration
Packing linux-headers-5.10-arm64_aarch64.deb...
Running mk-kernel.sh - linux-headers-aarch64 succeeded.
Running mk-kernel.sh - linux-headers succeeded.
Running 99-all.sh - build_all succeeded.
lixinkai@lixinkai-virtual-machine:~/atk_quarkpi-ca2_buildroot$
```

Figure 4.3-8 The SDK compilation was successful.

The print message "Running 99-all.sh - build_all succeeded." indicates that the compilation was successful.

After the compilation is completed, the generated image is stored in the "output/firmware/" directory, as shown in the following figure:

```
lixinkai@lixinkai-virtual-machine:~/atk_quarkpi-ca2_buildroot$ ls output/firmware/ -l
总用量 4
lrwxrwxrwx 1 lixinkai lixinkai 21 3月  11 10:05 boot.img -> ../../kernel/boot.img
lrwxrwxrwx 1 lixinkai lixinkai 44 3月  11 10:04 MiniLoaderAll.bin -> ../../u-boot/rk3588_spl_loader_v1.17.113.bin
lrwxrwxrwx 1 lixinkai lixinkai 11 3月  11 10:04 misc.img -> ../misc.img
lrwxrwxrwx 1 lixinkai lixinkai 22 3月  11 10:10 oem.img -> ../extra-parts/oem.img
lrwxrwxrwx 1 lixinkai lixinkai 49 3月  11 10:10 parameter.txt -> ../../device/rockchip/.chips/rk3588/parameter.txt
lrwxrwxrwx 1 lixinkai lixinkai 23 3月  11 10:10 recovery.img -> ../recovery/ramboot.img
lrwxrwxrwx 1 lixinkai lixinkai 66 3月  11 10:10 rootfs.img -> ../../buildroot/output/rockchip_atk_quarkpi_ca2/images/rootfs.ext2
lrwxrwxrwx 1 lixinkai lixinkai 22 3月  11 10:04 uboot.img -> ../../u-boot/uboot.img
lrwxrwxrwx 1 lixinkai lixinkai 26 3月  11 10:10 update.img -> ../update/Image/update.img
lrwxrwxrwx 1 lixinkai lixinkai 27 3月  11 10:10 userdata.img -> ../extra-parts/userdata.img
lixinkai@lixinkai-virtual-machine:~/atk_quarkpi-ca2_buildroot$
```

There are multiple soft links in this directory, which will link to the actual image files. Section 4.6 will provide an introduction to them.

4.4 Independent Compilation

The QuarkPi-CA2 Linux SDK integrates the Linux kernel source code, U-Boot source code, and rootfs (root file system), etc. This section will introduce to users how to independently compile U-Boot, the Linux kernel, and the rootfs.

4.4.1 Independent Compilation of U-Boot

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

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

```
lixinkai@lixinkai-virtual-machine:~/atk_quarkpi_ca2_buildroot$ ./build.sh uboot

############### Rockchip Linux SDK ###############

Manifest: atk-quarkpi_ca2_linux_release_v1.0.0_20241216.xml
Version: linux-5.10-gen-rkr8

Log colors: message notice warning error fatal

Log saved at /home/lixinkai/atk_quarkpi_ca2_buildroot/output/sessions/2024-12-18_09-58-13
Using last kernel version(5.10)

=========================================
          Final configs
=========================================
RK_BOOT_FIT_ITS=/home/lixinkai/atk_quarkpi_ca2_buildroot/device/rockchip/.chip/boot.its
RK_BOOT_IMG=boot.img
RK_BUILDROOT=y
RK_BUILDROOT_CFG=rockchip_atk_quarkpi_ca2
RK_CHIP=rk3588
RK_CHIP_FAMILY=rk3588
RK_DEBIAN=y
RK_DEBIAN_ARCH=arm64
RK_DEBIAN_BULLSEYE=y
RK_DEBIAN_MIRROR=mirrors.ustc.edu.cn
RK_DEBIAN_VERSION=bullseye
RK_DEFCONFIG=alientek_quarkpi_ca2_defconfig
RK_EXTRA_PARTITION_NUM=2
RK_EXTRA_PARTITION_STR=oem:oem:/oem:ext4:defaults:normal:auto:@userdata:userdata:/userdata:ext4:defaults:normal:auto:@@@
RK_KERNEL=y
RK_KERNEL_ARCH=arm64
RK_KERNEL_CFG=rockchip_linux_defconfig
RK_KERNEL_DTS=kernel/arch/arm64/boot/dts/rockchip/rk3588s-atk-quarkpi-ca2.dts
RK_KERNEL_IMG=kernel/arch/arm64/boot/Image
RK_KERNEL_VERSION=5.10
RK_KERNEL_VERSION_RAW=5.10
RK_LOADER=y
RK_MISC=y
RK_MISC_BLANK=y
RK_OVERLAY=y
```

```
  Type:         Firmware
  Compression:  gzip compressed
  Data Size:    228595 Bytes = 223.24 KiB = 0.22 MiB
  Architecture: AArch64
  Load Address: 0x08400000
  Hash algo:    sha256
  Hash value:   d5fbc5c32e6e5cb621f8fa64002bd3cadab94a0a5246d274ad9cf8e3517cff51
 Image 5 (fdt)
  Description:  U-Boot dtb
  Created:      Wed Dec 18 09:58:41 2024
  Type:         Flat Device Tree
  Compression:  uncompressed
  Data Size:    8867 Bytes = 8.66 KiB = 0.01 MiB
  Architecture: AArch64
  Hash algo:    sha256
  Hash value:   a434b1c4fe1fe989156b3b53572daa76e0342fbcf2503fd4807ad30f8da2ce71
 Default Configuration: 'conf'
 Configuration 0 (conf)
  Description:  rk3588-evb
  Kernel:       unavailable
  Firmware:     atf-1
  FDT:          fdt
  Loadables:    uboot
                atf-2
                atf-3
                optee
********boot_merger ver 1.35********
Info:Pack loader ok.
pack loader okay! Input: /home/lixinkai/atk_quarkpi_ca2_buildroot/rkbin/RKBOOT/RK3588MINIALL.ini
/home/lixinkai/atk_quarkpi_ca2_buildroot/u-boot

Image(no-signed, version=0): uboot.img (FIT with uboot, trust...) is ready
Image(no-signed): rk3588_spl_loader_v1.17.113.bin (with spl, ddr...) is ready
pack uboot.img okay! Input: /home/lixinkai/atk_quarkpi_ca2_buildroot/rkbin/RKTRUST/RK3588TRUST.ini

Platform RK3588 is build OK, with new .config(make rk3588_defconfig -j32)
/home/lixinkai/atk_quarkpi_ca2_buildroot/prebuilts/gcc/linux-x86/aarch64/gcc-arm-10.3-2021.07-x86_64-aarch64-none-linux-gnu/bin/a
Wed Dec 18 09:58:42 CST 2024
+ cd ..
Running 70-loader.sh - build uboot succeeded.
lixinkai@lixinkai-virtual-machine:~/atk_quarkpi_ca2_buildroot$
```

Figure 4.4-1 Compile U-Boot

After the compilation is completed, two image files will be generated: uboot.img and MiniLoaderAll.bin. The images are stored in the directory of output/firmware/.

```
lixinkai@lixinkai-virtual-machine:~/atk_quarkpi-ca2_buildroot$ ls output/firmware/ -l
总用量 0
lrwxrwxrwx 1 lixinkai lixinkai 44 3月  11 10:17 MiniLoaderAll.bin -> ../../u-boot/rk3588_spl_loader_v1.17.113.bin
lrwxrwxrwx 1 lixinkai lixinkai 22 3月  11 10:17 uboot.img -> ../../u-boot/uboot.img
lixinkai@lixinkai-virtual-machine:~/atk_quarkpi-ca2_buildroot$
```

If the user has modified the U-Boot source code, they need to regenerate uboot.img by following the above command.

4.4.2 Independently compiling the Linux kernel

By using the build.sh script, independently compile the Linux kernel. Execute the following command in the SDK root directory:

```
./build.sh kernel          # Compile the kernel
./build.sh recovery        # Compile recovery
```

```
lixinkai@lixinkai-virtual-machine:~/atk_quarkpi_ca2_buildroot$ ./build.sh kernel

############### Rockchip Linux SDK ###############

Manifest: atk-quarkpi_ca2_linux_release_v1.0.0_20241216.xml
Version: linux-5.10-gen-rkr8

Log colors: message notice warning error fatal

Log saved at /home/lixinkai/atk_quarkpi_ca2_buildroot/output/sessions/2024-12-18_10-00-07
Using last kernel version(5.10)

=========================================
          Final configs
=========================================
RK_BOOT_FIT_ITS=/home/lixinkai/atk_quarkpi_ca2_buildroot/device/rockchip/.chip/boot.its
RK_BOOT_IMG=boot.img
RK_BUILDROOT=y
RK_BUILDROOT_CFG=rockchip_atk_quarkpi_ca2
RK_CHIP=rk3588
RK_CHIP_FAMILY=rk3588
RK_DEBIAN=y
RK_DEBIAN_ARCH=arm64
RK_DEBIAN_BULLSEYE=y
RK_DEBIAN_MIRROR=mirrors.ustc.edu.cn
RK_DEBIAN_VERSION=bullseye
RK_DEFCONFIG=alientek_quarkpi_ca2_defconfig
RK_EXTRA_PARTITION_NUM=2
RK_EXTRA_PARTITION_STR=oem:oem:/oem:ext4:defaults:normal:auto:@userdata:userdata:/userdata:ext4:defaults:normal:auto:@@@
RK_KERNEL=y
RK_KERNEL_ARCH=arm64
RK_KERNEL_CFG=rockchip_linux_defconfig
RK_KERNEL_DTS=kernel/arch/arm64/boot/dts/rockchip/rk3588s-atk-quarkpi-ca2.dts
RK_KERNEL_IMG=kernel/arch/arm64/boot/Image
RK_KERNEL_VERSION=5.10
RK_KERNEL_VERSION_RAW=5.10
RK_LOADER=y
RK_MISC=y
RK_MISC_BLANK=y
RK_OVERLAY=y
```

```
FIT description: U-Boot FIT source file for arm
Created:         Wed Dec 18 10:01:04 2024
 Image 0 (fdt)
  Description:  unavailable
  Created:      Wed Dec 18 10:01:04 2024
  Type:         Flat Device Tree
  Compression:  uncompressed
  Data Size:    249926 Bytes = 244.07 KiB = 0.24 MiB
  Architecture: AArch64
  Load Address: 0xffffff00
  Hash algo:    sha256
  Hash value:   ddf5a47df00d325a720120c6d98f1f49316caec39408a5b0e2ec5c4be319d1d6
 Image 1 (kernel)
  Description:  unavailable
  Created:      Wed Dec 18 10:01:04 2024
  Type:         Kernel Image
  Compression:  uncompressed
  Data Size:    36272640 Bytes = 35422.50 KiB = 34.59 MiB
  Architecture: AArch64
  OS:           Linux
  Load Address: 0xffffff01
  Entry Point:  0xffffff01
  Hash algo:    sha256
  Hash value:   e479c9cbecf6dfd39ad7efd9c71edd426887bce75443741ca6e419d25c97ac0e
 Image 2 (resource)
  Description:  unavailable
  Created:      Wed Dec 18 10:01:04 2024
  Type:         Multi-File Image
  Compression:  uncompressed
  Data Size:    608768 Bytes = 594.50 KiB = 0.58 MiB
  Hash algo:    sha256
  Hash value:   b1bcfecdbe38deef626b21f4d40b4a25242095f20497920d6cd1863fa44815be
 Default Configuration: 'conf'
 Configuration 0 (conf)
  Description:  unavailable
  Kernel:       kernel
  FDT:          fdt
+ ln -rsf kernel/boot.img /home/lixinkai/atk_quarkpi_ca2_buildroot/output/firmware/boot.img
Not Found io-domains in kernel/arch/arm64/boot/dts/rockchip/rk3588s-atk-quarkpi-ca2.dts
Running 10-kernel.sh - build_kernel succeeded.
lixinkai@lixinkai-virtual-machine:~/atk_quarkpi_ca2_buildroot$
```

Figure 4.4-2 Compile the Linux kernel

```
lixinkai@lixinkai-virtual-machine:~/atk_quarkpi_ca2_buildroot$ ./build.sh recovery

############### Rockchip Linux SDK ###############

Manifest: atk-quarkpi_ca2_linux_release_v1.0.0_20241216.xml
Version: linux-5.10-gen-rkr8

Log colors: message notice warning error fatal

Log saved at /home/lixinkai/atk_quarkpi_ca2_buildroot/output/sessions/2024-12-18_10-04-24
Using last kernel version(5.10)

=========================================
          Final configs
=========================================
RK_BOOT_FIT_ITS=/home/lixinkai/atk_quarkpi_ca2_buildroot/device/rockchip/.chip/boot.its
RK_BOOT_IMG=boot.img
RK_BUILDROOT=y
RK_BUILDROOT_CFG=rockchip_atk_quarkpi_ca2
RK_CHIP=rk3588
RK_CHIP_FAMILY=rk3588
RK_DEBIAN=y
RK_DEBIAN_ARCH=arm64
RK_DEBIAN_BULLSEYE=y
RK_DEBIAN_MIRROR=mirrors.ustc.edu.cn
RK_DEBIAN_VERSION=bullseye
RK_DEFCONFIG=alientek_quarkpi_ca2_defconfig
RK_EXTRA_PARTITION_NUM=2
RK_EXTRA_PARTITION_STR=oem:oem:/oem:ext4:defaults:normal:auto:@userdata:userdata:/userdata:ext4:defaults:normal:auto:@@@
RK_KERNEL=y
RK_KERNEL_ARCH=arm64
RK_KERNEL_CFG=rockchip_linux_defconfig
RK_KERNEL_DTS=kernel/arch/arm64/boot/dts/rockchip/rk3588s-atk-quarkpi-ca2.dts
RK_KERNEL_IMG=kernel/arch/arm64/boot/Image
RK_KERNEL_VERSION=5.10
RK_KERNEL_VERSION_RAW=5.10
RK_LOADER=y
RK_MISC=y
RK_MISC_BLANK=y
```

```
  Hash value:   ddf5a47df00d325a720120c6d98f1f49316caec39408a5b0e2ec5c4be319d1d6
 Image 1 (kernel)
  Description:  unavailable
  Created:      Wed Dec 18 10:05:08 2024
  Type:         Kernel Image
  Compression:  uncompressed
  Data Size:    36272640 Bytes = 35422.50 KiB = 34.59 MiB
  Architecture: AArch64
  OS:           Linux
  Load Address: 0xffffff01
  Entry Point:  0xffffff01
  Hash algo:    sha256
  Hash value:   e479c9cbecf6dfd39ad7efd9c71edd426887bce75443741ca6e419d25c97ac0e
 Image 2 (ramdisk)
  Description:  unavailable
  Created:      Wed Dec 18 10:05:08 2024
  Type:         RAMDisk Image
  Compression:  uncompressed
  Data Size:    6795275 Bytes = 6636.01 KiB = 6.48 MiB
  Architecture: AArch64
  OS:           Linux
  Load Address: 0xffffff02
  Entry Point:  unavailable
  Hash algo:    sha256
  Hash value:   f9efe63a623890e250f960d115da704f68195866285f702dffb0d3aa21f4171c
 Image 3 (resource)
  Description:  unavailable
  Created:      Wed Dec 18 10:05:08 2024
  Type:         Multi-File Image
  Compression:  uncompressed
  Data Size:    608768 Bytes = 594.50 KiB = 0.58 MiB
  Hash algo:    sha256
  Hash value:   b1bcfecdbe38deef626b21f4d40b4a25242095f20497920d6cd1863fa44815be
 Default Configuration: 'conf'
 Configuration 0 (conf)
  Description:  unavailable
  Kernel:       kernel
  Init Ramdisk: ramdisk
  FDT:          fdt
Running 40-recovery.sh - build_recovery succeeded.
lixinkai@lixinkai-virtual-machine:~/atk_quarkpi_ca2_buildroot$
```

Figure 4.4-3 Compile recovery

After compilation is completed, boot.img and recovery.img are generated. The images are stored in the directory output/firmware/.

```
lixinkai@lixinkai-virtual-machine:~/atk_quarkpi-ca2_buildroot$ ls output/firmware/ -l
总用量 0
lrwxrwxrwx 1 lixinkai lixinkai 21 3月  11 10:20 boot.img -> ../../kernel/boot.img
lrwxrwxrwx 1 lixinkai lixinkai 23 3月  11 10:21 recovery.img -> ../recovery/ramboot.img
lixinkai@lixinkai-virtual-machine:~/atk_quarkpi-ca2_buildroot$
```

Because the Linux kernel image (including the kernel DTB) is not burned separately, but is packaged into boot.img and recovery.img (both boot.img and recovery.img images contain the kernel image and kernel DTB). Therefore, when users modify the kernel source code or device tree configuration, they need to regenerate boot.img and recovery.img according to the above commands.

To compile the Linux kernel module independently, the following command can be executed for compilation:

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

4.4.3 Independently compiling rootfs

The RK3588S Linux SDK supports various root file systems (rootfs), 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 official RK team does not recommend users to use Yocto either.

The Buildroot root file system can be compiled independently through the build.sh script. Execute the following commands in the SDK root directory:

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

```
lixinkai@lixinkai-virtual-machine:~/atk_quarkpi_ca2_buildroot$ ./build.sh buildroot

############### Rockchip Linux SDK ###############

Manifest: atk-quarkpi_ca2_linux_release_v1.0.0_20241216.xml
Version: linux-5.10-gen-rkr8

Log colors: message notice warning error fatal

Log saved at /home/lixinkai/atk_quarkpi_ca2_buildroot/output/sessions/2024-12-18_10-06-05
Using last kernel version(5.10)
Using rootfs system(buildroot) from cmdline

=========================================
          Final configs
=========================================
RK_BOOT_FIT_ITS=/home/lixinkai/atk_quarkpi_ca2_buildroot/device/rockchip/.chip/boot.its
RK_BOOT_IMG=boot.img
RK_BUILDROOT=y
RK_BUILDROOT_CFG=rockchip_atk_quarkpi_ca2
RK_CHIP=rk3588
RK_CHIP_FAMILY=rk3588
RK_DEBIAN=y
RK_DEBIAN_ARCH=arm64
RK_DEBIAN_BULLSEYE=y
RK_DEBIAN_MIRROR=mirrors.ustc.edu.cn
RK_DEBIAN_VERSION=bullseye
RK_DEFCONFIG=alientek_quarkpi_ca2_defconfig
RK_EXTRA_PARTITION_NUM=2
RK_EXTRA_PARTITION_STR=oem:oem:/oem:ext4:defaults:normal:auto:@userdata:userdata:/userdata:ext4:defaults:normal:auto:@@@
RK_KERNEL=y
RK_KERNEL_ARCH=arm64
RK_KERNEL_CFG=rockchip_linux_defconfig
RK_KERNEL_DTS=kernel/arch/arm64/boot/dts/rockchip/rk3588s-atk-quarkpi-ca2.dts
RK_KERNEL_IMG=kernel/arch/arm64/boot/Image
RK_KERNEL_VERSION=5.10
RK_KERNEL_VERSION_RAW=5.10
RK_LOADER=y
RK_MISC=y
```

```
copy from `/home/lixinkai/atk_quarkpi_ca2_buildroot/buildroot/output/rockchip_atk_quarkpi_ca2/target/usr/lib/modules/5.10.160/rtw88_8822b.ko' [elf64-littl
eaarch64] to `/home/lixinkai/atk_quarkpi_ca2_buildroot/buildroot/output/rockchip_atk_quarkpi_ca2/target/usr/lib/modules/5.10.160/stDL8rgp' [elf64-littleaa
rch64]
copy from `/home/lixinkai/atk_quarkpi_ca2_buildroot/buildroot/output/rockchip_atk_quarkpi_ca2/target/usr/lib/modules/RTL8189FU.ko' [elf64-littleaarch64] t
o `/home/lixinkai/atk_quarkpi_ca2_buildroot/buildroot/output/rockchip_atk_quarkpi_ca2/target/usr/lib/modules/stFflPQE' [elf64-littleaarch64]
copy from `/home/lixinkai/atk_quarkpi_ca2_buildroot/buildroot/output/rockchip_atk_quarkpi_ca2/target/usr/lib/modules/RTL8822CS.ko' [elf64-littleaarch64] t
o `/home/lixinkai/atk_quarkpi_ca2_buildroot/buildroot/output/rockchip_atk_quarkpi_ca2/target/usr/lib/modules/sthUps7R' [elf64-littleaarch64]
copy from `/home/lixinkai/atk_quarkpi_ca2_buildroot/buildroot/output/rockchip_atk_quarkpi_ca2/target/usr/lib/modules/RTL8189FS.ko' [elf64-littleaarch64] t
o `/home/lixinkai/atk_quarkpi_ca2_buildroot/buildroot/output/rockchip_atk_quarkpi_ca2/target/usr/lib/modules/stfkXbIh' [elf64-littleaarch64]
copy from `/home/lixinkai/atk_quarkpi_ca2_buildroot/buildroot/output/rockchip_atk_quarkpi_ca2/target/usr/lib/modules/hci_uart.ko' [elf64-littleaarch64] to
 `/home/lixinkai/atk_quarkpi_ca2_buildroot/buildroot/output/rockchip_atk_quarkpi_ca2/target/usr/lib/modules/stKPAKMi' [elf64-littleaarch64]
Running 95-partitions.sh for /home/lixinkai/atk_quarkpi_ca2_buildroot/buildroot/output/rockchip_atk_quarkpi_ca2/target (buildroot init=busybox)...
Preparing extra partitions...
Running 97-ldcache.sh for /home/lixinkai/atk_quarkpi_ca2_buildroot/buildroot/output/rockchip_atk_quarkpi_ca2/target (buildroot init=busybox)...
Keep original /etc/ld.so.cache
Running 99-owner.sh for /home/lixinkai/atk_quarkpi_ca2_buildroot/buildroot/output/rockchip_atk_quarkpi_ca2/target (buildroot init=busybox)...
Running 30-rootfs.sh - build_buildroot /home/lixinkai/atk_quarkpi_ca2_buildroot/output/buildroot/images succeeded.
Running 30-rootfs.sh - build_rootfs buildroot succeeded.
lixinkai@lixinkai-virtual-machine:~/atk_quarkpi_ca2_buildroot$
```

Figure 4.4-4 Compile buildroot

After the compilation is completed, a buildroot root file system image named "rootfs.img" is generated. The image is stored in the "output/firmware/" directory, as shown below:

```
lixinkai@lixinkai-virtual-machine:~/atk_quarkpi-ca2_buildroot$ ls output/firmware/ -l
总用量 4
lrwxrwxrwx 1 lixinkai lixinkai 66 3月  11 10:34 rootfs.img -> ../../buildroot/output/rockchip_atk_quarkpi_ca2/images/rootfs.ext2
lixinkai@lixinkai-virtual-machine:~/atk_quarkpi-ca2_buildroot$
```

Figure 4.4-5 Root file system image

4.4.4 Independently compiling recovery

By using the build.sh script, independently compile the recovery. Execute the following command in the root directory of the SDK:

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

```
lixinkai@lixinkai-virtual-machine:~/atk_quarkpi_ca2_buildroot$ ./build.sh recovery

############### Rockchip Linux SDK ###############

Manifest: atk-quarkpi_ca2_linux_release_v1.0.0_20241216.xml
Version: linux-5.10-gen-rkr8

Log colors: message notice warning error fatal

Log saved at /home/lixinkai/atk_quarkpi_ca2_buildroot/output/sessions/2024-12-18_10-04-24
Using last kernel version(5.10)

========================================
          Final configs
========================================
RK_BOOT_FIT_ITS=/home/lixinkai/atk_quarkpi_ca2_buildroot/device/rockchip/.chip/boot.its
RK_BOOT_IMG=boot.img
RK_BUILDROOT=y
RK_BUILDROOT_CFG=rockchip_atk_quarkpi_ca2
RK_CHIP=rk3588
RK_CHIP_FAMILY=rk3588
RK_DEBIAN=y
RK_DEBIAN_ARCH=arm64
RK_DEBIAN_BULLSEYE=y
RK_DEBIAN_MIRROR=mirrors.ustc.edu.cn
RK_DEBIAN_VERSION=bullseye
RK_DEFCONFIG=alientek_quarkpi_ca2_defconfig
RK_EXTRA_PARTITION_NUM=2
RK_EXTRA_PARTITION_STR=oem:oem:/oem:ext4:defaults:normal:auto:@userdata:userdata:/userdata:ext4:defaults:normal:auto:@@@
RK_KERNEL=y
RK_KERNEL_ARCH=arm64
RK_KERNEL_CFG=rockchip_linux_defconfig
RK_KERNEL_DTS=kernel/arch/arm64/boot/dts/rockchip/rk3588s-atk-quarkpi-ca2.dts
RK_KERNEL_IMG=kernel/arch/arm64/boot/Image
RK_KERNEL_VERSION=5.10
RK_KERNEL_VERSION_RAW=5.10
RK_LOADER=y
RK_MISC=y
RK_MISC_BLANK=y
```

```
  Hash value:   ddf5a47df00d325a720120c6d98f1f49316caec39408a5b0e2ec5c4be319d1d6
 Image 1 (kernel)
  Description:  unavailable
  Created:      Wed Dec 18 10:05:08 2024
  Type:         Kernel Image
  Compression:  uncompressed
  Data Size:    36272640 Bytes = 35422.50 KiB = 34.59 MiB
  Architecture: AArch64
  OS:           Linux
  Load Address: 0xffffff01
  Entry Point:  0xffffff01
  Hash algo:    sha256
  Hash value:   e479c9cbecf6dfd39ad7efd9c71edd426887bce75443741ca6e419d25c97ac0e
 Image 2 (ramdisk)
  Description:  unavailable
  Created:      Wed Dec 18 10:05:08 2024
  Type:         RAMDisk Image
  Compression:  uncompressed
  Data Size:    6795275 Bytes = 6636.01 KiB = 6.48 MiB
  Architecture: AArch64
  OS:           Linux
  Load Address: 0xffffff02
  Entry Point:  unavailable
  Hash algo:    sha256
  Hash value:   f9efe63a623890e250f960d115da704f68195866285f702dffb0d3aa21f4171c
 Image 3 (resource)
  Description:  unavailable
  Created:      Wed Dec 18 10:05:08 2024
  Type:         Multi-File Image
  Compression:  uncompressed
  Data Size:    608768 Bytes = 594.50 KiB = 0.58 MiB
  Hash algo:    sha256
  Hash value:   b1bcfecdbe38deef626b21f4d40b4a25242095f20497920d6cd1863fa44815be
 Default Configuration: 'conf'
 Configuration 0 (conf)
  Description:  unavailable
  Kernel:       kernel
  Init Ramdisk: ramdisk
  FDT:          fdt
Running 40-recovery.sh - build recovery succeeded.
lixinkai@lixinkai-virtual-machine:~/atk_quarkpi_ca2_buildroot$
```

Figure 4.4-6 Compile recovery

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, including 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 needs to be pre-compiled.

After compilation is complete, the "recovery.img" is generated and is stored in the "output/firmware/" directory, as shown below:

```
lixinkai@lixinkai-virtual-machine:~/atk_quarkpi-ca2_buildroot$ ls output/firmware/ -l
总用量 0
lrwxrwxrwx 1 lixinkai lixinkai 23 3月  11 10:36 recovery.img -> ../recovery/ramboot.img
lixinkai@lixinkai-virtual-machine:~/atk_quarkpi-ca2_buildroot$
```

Figure 4.4-7 recovery.img

4.4.5 Creating the update.img firmware

The update.img firmware is a collection of multiple images, which are packaged together from multiple individual images. Using the build.sh script, the separate images in the output/firmware/ directory can be packaged into a single update.img firmware, making it convenient for users to burn, update, and upgrade.

Execute the following command in the SDK root directory to create the update.img:

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

```
lixinkai@lixinkai-virtual-machine:~/atk_quarkpi_ca2_buildroot$ ./build.sh updateimg

############### Rockchip Linux SDK ###############

Manifest: atk-quarkpi_ca2_linux_release_v1.0.0_20241216.xml
Version: linux-5.10-gen-rkr8

Log colors: message notice warning error fatal

Log saved at /home/lixinkai/atk_quarkpi_ca2_buildroot/output/sessions/2024-12-18_10-13-34
Using last kernel version(5.10)

========================================
          Final configs
========================================
RK_BOOT_FIT_ITS=/home/lixinkai/atk_quarkpi_ca2_buildroot/device/rockchip/.chip/boot.its
RK_BOOT_IMG=boot.img
RK_BUILDROOT=y
RK_BUILDROOT_CFG=rockchip_atk_quarkpi_ca2
RK_CHIP=rk3588
RK_CHIP_FAMILY=rk3588
RK_DEBIAN=y
RK_DEBIAN_ARCH=arm64
RK_DEBIAN_BULLSEYE=y
RK_DEBIAN_MIRROR=mirrors.ustc.edu.cn
RK_DEBIAN_VERSION=bullseye
RK_DEFCONFIG=alientek_quarkpi_ca2_defconfig
RK_EXTRA_PARTITION_NUM=2
RK_EXTRA_PARTITION_STR=oem:oem:/oem:ext4:defaults:normal:auto:@userdata:userdata:/userdata:ext4:defaults:normal:auto:@@@
RK_KERNEL=y
RK_KERNEL_ARCH=arm64
RK_KERNEL_CFG=rockchip_linux_defconfig
RK_KERNEL_DTS=kernel/arch/arm64/boot/dts/rockchip/rk3588s-atk-quarkpi-ca2.dts
RK_KERNEL_IMG=kernel/arch/arm64/boot/Image
RK_KERNEL_VERSION=5.10
RK_KERNEL_VERSION_RAW=5.10
RK_LOADER=y
RK_MISC=y
RK_MISC_BLANK=y
RK_OVERLAY=y
```

```
oem     oem.img
userdata        userdata.img
Packing /home/lixinkai/atk_quarkpi_ca2_buildroot/output/firmware/update.img for update...
Android Firmware Package Tool v2.27
------ PACKAGE ------
Add file: ./package-file
package-file,Add file: ./package-file done,offset=0x800,size=0xe1,userspace=0x1
Add file: ./parameter.txt
parameter,Add file: ./parameter.txt done,offset=0x1000,size=0x227,userspace=0x1,flash_address=0x00000000
Add file: ./MiniLoaderAll.bin
bootloader,Add file: ./MiniLoaderAll.bin done,offset=0x1800,size=0x761c0,userspace=0xed
Add file: ./uboot.img
uboot,Add file: ./uboot.img done,offset=0x78000,size=0x400000,userspace=0x800,flash_address=0x00004000
Add file: ./misc.img
misc,Add file: ./misc.img done,offset=0x478000,size=0xc000,userspace=0x18,flash_address=0x00006000
Add file: ./boot.img
boot,Add file: ./boot.img done,offset=0x484000,size=0x236a200,userspace=0x46d5,flash_address=0x00008000
Add file: ./recovery.img
recovery,Add file: ./recovery.img done,offset=0x27ee800,size=0x29e5400,userspace=0x53cb,flash_address=0x00028000
Add file: ./rootfs.img
rootfs,Add file: ./rootfs.img done,offset=0x51d4000,size=0x5b200000,userspace=0xb6400,flash_address=0x00078000
Add file: ./oem.img
oem,Add file: ./oem.img done,offset=0x603d4000,size=0x11ef000,userspace=0x23de,flash_address=0x01c78000
Add file: ./userdata.img
userdata,Add file: ./userdata.img done,offset=0x615c3000,size=0x44e000,userspace=0x89c,flash_address=0x01cb8000
Add CRC...
Make firmware OK!
------ OK ------
********rkImageMaker ver 2.23********
Generating new image, please wait...
Writing head info...
Writing boot file...
Writing firmware...
Generating MD5 data...
MD5 data generated successfully!
New image generated successfully!

Run 'make edit-package-file' if you want to change the package-file.

Running 90-updateimg.sh - build_updateimg succeeded.
lixinkai@lixinkai-virtual-machine:~/atk_quarkpi_ca2_buildroot$
```

Figure 4.4-8 Create the update.img firmware

After the packaging is successful, the update.img firmware will be generated in the output/firmware/ directory, as shown below:

```
lixinkai@lixinkai-virtual-machine:~/atk_quarkpi-ca2_buildroot$ ls output/firmware/update.img -l
lrwxrwxrwx 1 lixinkai lixinkai 26 3月  11 10:46 output/firmware/update.img -> ../update/Image/update.img
lixinkai@lixinkai-virtual-machine:~/atk_quarkpi-ca2_buildroot$
```

Figure 4.4-9 update.img firmware

After the compilation of "./build.sh all" is completed, all the separate images will be automatically packaged into update.img.

4.5 SDK Cleanup

The following command can be executed in the root directory of the SDK to clean up the entire SDK:

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

You can also execute the following commands to clean up each module:

```
./build.sh clean:loader        # Clean up U-Boot
./build.sh clean:kernel        # Clean up the Linux kernel
./build.sh clean:recovery      # Clean up "recovery"
./build.sh clean:rootfs        # Clean the rootfs
```

4.6 Image Introduction

This section will provide a brief introduction to the image files located in the output/firmware/ directory, as shown in the table below:

Mirror image	Function

uboot.img	uboot.img is a FIT format image, which is formed by merging multiple images, including the trust image (ARM Trusted Firmware + OP-TEE OS), u-boot image, and u-boot dtb. When compiling U-Boot, these images will be packaged into one uboot.img; ultimately, uboot.img will be burned into the uboot partition of the development board.
boot.img	The boot.img is also a FIT format image, which is formed by merging multiple images, including the kernel image Image, the kernel DTB, and the resource image resource.img. Eventually, boot.img will be burned onto the boot partition of the development board. resource.img is the resource image for the RK platform, including DTB, boot logo and other resources.
MiniLoaderAll.bin	A section of Loader code that ran before U-Boot on the RK3588S platform (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 is mainly responsible for loading and booting uboot.img.
misc.img	The misc.img image will be burned onto 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 a very important partition, mainly used for communication between the Android system, BootLoader, and recovery; the misc partition contains the BCB (BootLoader Control Block) data block, which is a mechanism designed by Android to control the boot process and interact with the BootLoader.
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 the partition table for the RK3588S platform (which records the partition names and the starting and ending addresses of each partition); the parameter.txt partition table file will not be burned onto the Flash storage device. The burning tool will create the partition table based on the defined partitions inside it.
recovery.img	This image is formed by merging multiple images, including the ramdisk (the root file system mounted when entering the recovery mode), the kernel image Image, the kernel DTB, and the resource image resource.img. recovery.img is used to enter the recovery mode, which is a mode used for repairing and upgrading the device; recovery.img will eventually be burned onto the recovery partition of the development board.
rootfs.img	The root file system corresponding to the normal boot mode (the root file system mounted during normal boot mode) contains a large number of library files, executable files, etc.

	The rootfs.img file will eventually be burned onto the rootfs partition of the development board.
userdata.img	For end users' use, it can be used to store users' apps or data; this image will be burned into the userdata partition of the development board. After the system starts, it will be mounted to the /userdata directory.
update.img	The firmware package formed by merging the separate mirror images is convenient for users to burn and update.

Table 4.6.1 Introduction of Each Image on the RK3588S Platform

Chapter 5. Mirror Image Burning

This chapter provides users with instructions on how to burn the mirror image onto the QuarkPi-CA2 card computer and run it.

The Rockchip platform offers multiple methods for burning the image. In Section 3.7.1, we have introduced these methods to you. For example, burning the image under Windows using the RKDevTool tool, burning it via an SD card, batch burning (mass production burning tool, supporting USB one-to-many burning) using the FactoryTool tool, and burning it under the Ubuntu system using the Linux_Upgrade_Tool tool, etc. In summary, there are many ways to burn the image, and users can choose the appropriate method to burn the image onto the development board.

5.1 Flashing Mode Introduction

In Section 3.7.1, the issues related to the flashing mode of the development board have been elaborated in detail. If you still have any doubts about this, please proceed to read this section!

5.2 Flashing in Windows System

In Windows, use the RKDevTool provided by Rockchip to flash the image. Before flashing, first copy the compiled image (the image in the output/firmware/ directory) from the Ubuntu system to the Windows system. For example, copy it to the rk3588s_images directory on the Windows desktop, including boot.img, MiniLoaderAll.bin, misc.img, OEM.img, parameter.txt (partition table file), recovery.img, rootfs.img, uboot.img, and userdata.img. As shown below (note the issue of soft links, copy the image pointed to by the soft link, not the link file itself):

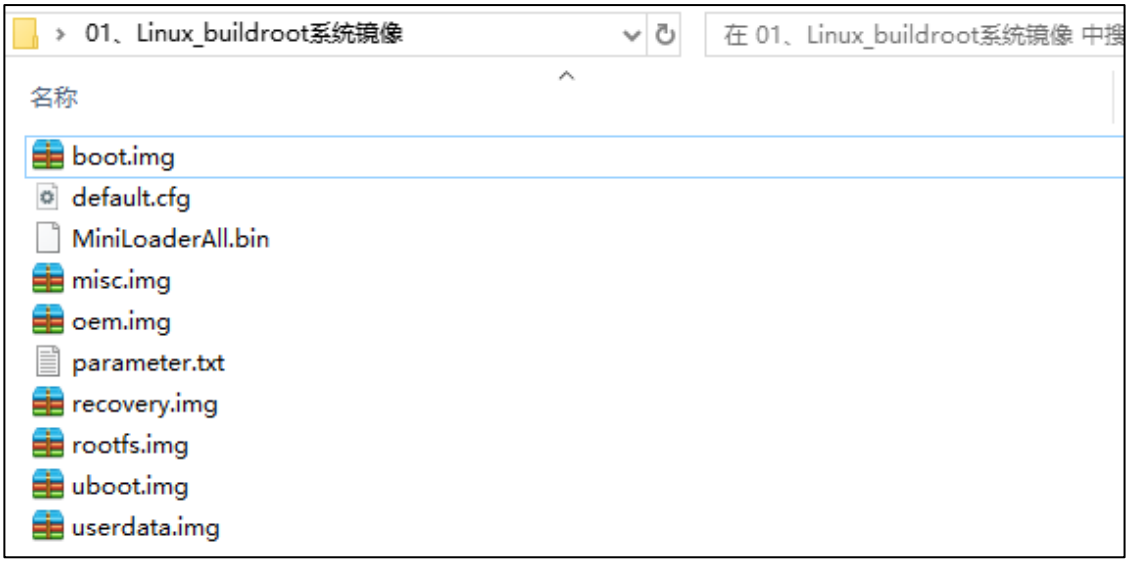


Figure 5.2-1 The files in the "rk3588_images" directory of the desktop

Open the development tools of Rockchip Microelectronics:

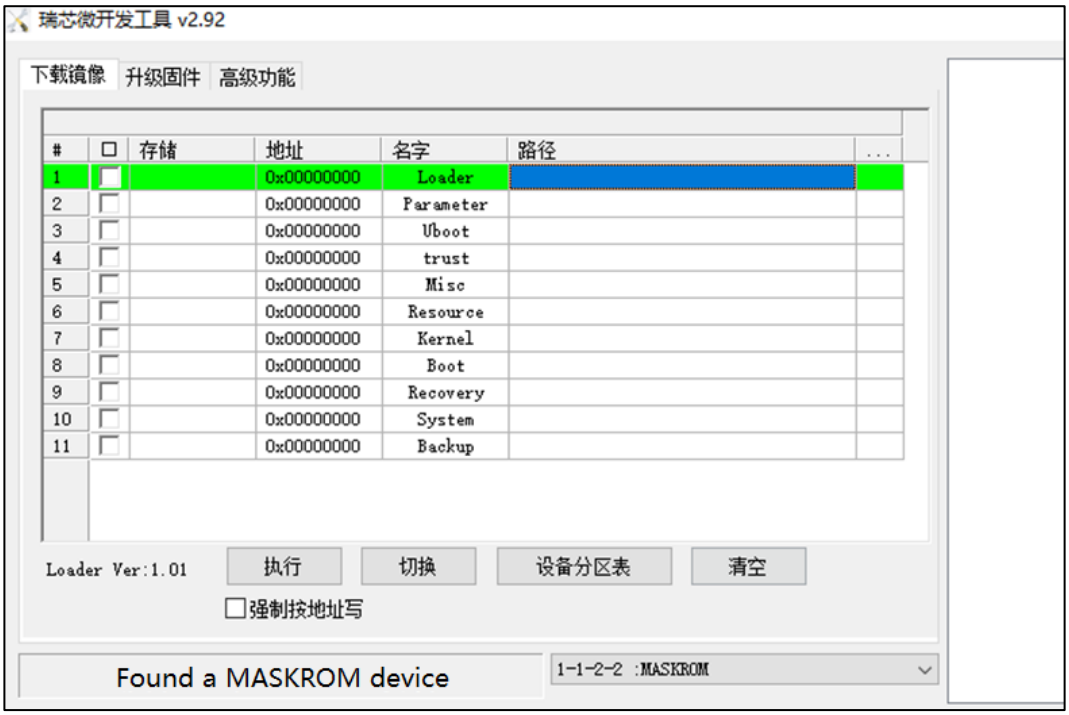


Figure 5.2-2 Ricoh Micro Development Tools

5.2.1 Introduction to Partition Table parameter.txt

This section will explain the partition table file parameter.txt. This file is a txt text file. As we have already mentioned earlier, parameter.txt is the partition table file for the Rockchip platform. This file records information such as the starting address and size of each partition. Let's take a look at its content:

```
FIRMWARE_VER: 1.0
MACHINE_MODEL: RK3588
MACHINE_ID: 007
MANUFACTURER: RK3588
MAGIC: 0x5041524B
ATAG: 0x00200800
MACHINE: 0xffffffff
CHECK_MASK: 0x80
PWR_HLD: 0,0,A,0,1
TYPE: GPT
GROW_ALIGN: 0
CMDLINE:
mtdparts=:0x00002000@0x00004000(uboot),0x00002000@0x00006000(misc),0x00020000@0x00008000(boot),0x00040000@0x00028000(recovery),0x00010000@0x00068000(backup),0x01c00000@0x00078000(rootfs),0x00040000@0x01c78000(oem),-@0x01cb8000(userdata:grow)
uuid:rootfs=614e0000-0000-4b53-8000-1d28000054a9
uuid:boot=7A3F0000-0000-446A-8000-702F00006273
```

RK has provided the corresponding documentation that describes various identifiers in the parameter.txt file. The location of this document is:

<SDK>/docs/cn/Common/TOOL/Rockchip_Introduction_Partition_CN.pdf. Here, we are merely presenting the partition information defined by mtdparts to you.

The information defined by mtdparts is as follows:

mtdparts=:0x00002000@0x00004000(uboot),0x00002000@0x00006000(misc),0x00020000@0x00008000(boot),0x00040000@0x00028000(recovery),0x00010000@0x00068000(backup),0x01c00000@0x00078000(rootfs),0x00040000@0x01c78000(oem),-@0x01cb8000(userdata:grow)

rk29xxnand is a designation. For compatibility, the Rockchip platform 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, and the value after the @ symbol is the starting position of the partition. The content inside the parentheses corresponds to 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 0x4000 sectors, with a size of 0x2000 sectors (4MB); the starting position of the Misc partition is 0x6000 sectors, 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:

-@0x01cb8000(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. This indicates that all the remaining space will be allocated to this partition.

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

Partition name	Explanation
uboot	Store the uboot.img image, and the uboot.img image will be burned into this partition.
misc	The misc.img image will be burned into this partition. The concept of the misc partition comes from the Android system. In the Android system, the misc partition is an important partition, mainly used for communication between Android, u-boot, and recovery.
boot	Store the boot.img, and the boot.img image will be burned into this partition.
recovery	Store the recovery.img, and the recovery.img image will be burned into this partition. By booting the recovery.img to enter the recovery mode, the recovery mode is a mode used for repairing and upgrading the device.
backup	Reserved partition, not yet in use
rootfs	Root file system partition, used to store rootfs.img (the rootfs.img image will be burned into this partition), by booting the rootfs.img to enter the normal boot mode
oem	A partition for the manufacturer to use, storing the manufacturer's APP 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	Partition for end users, storing the users' APP or data; after the system starts, this partition will be mounted to the root file system/userdata directory

5.2.2 Configuration

From the parameter.txt partition table file, we know that it defines a total of 8 partitions, along with their names, starting positions, and sizes. Based on this information, we can manually configure the Rockchip development tools. After the configuration is completed, it will look as shown in the following figure:

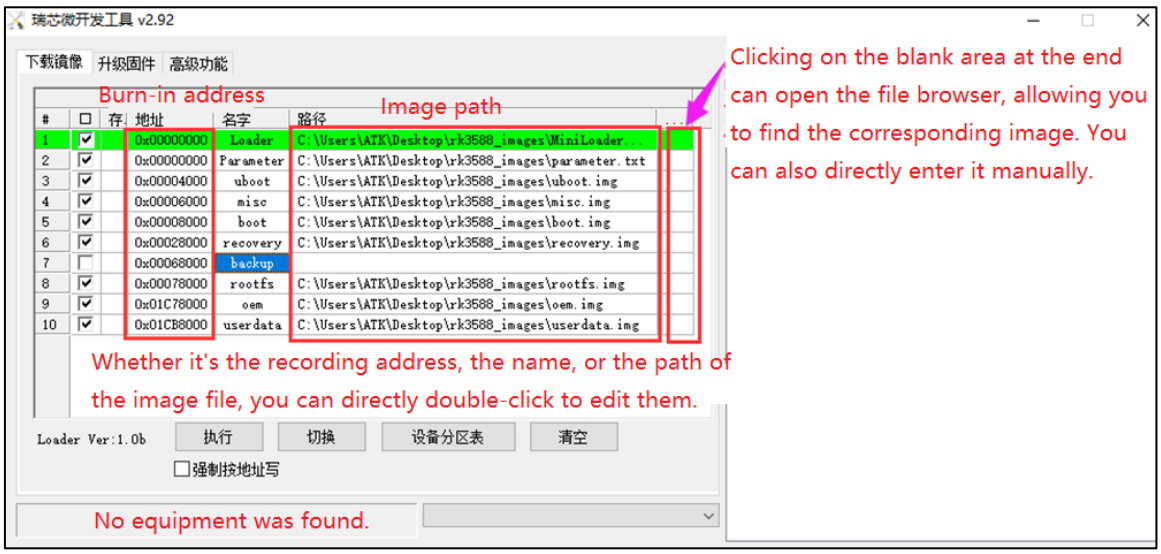


Figure 5.2-3 Manually configure the Rockchip development tools

Here are a few points to note：

1. The unit of the burning address is also sectors (a sector is equal to 512 bytes);

2. The first item corresponds to the MiniLoaderAll.bin image. Its address does not need to be configured and can be directly set to 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 picture, the second item corresponds to the partition table parameter.txt. Similarly，its address does not need to be configured, and can be directly set to 0x0, because the parameter.txt partition table file will not be burned to the Flash storage device. The burning tool will create the partition table based on the partitions defined in it; likewise, its name is set to parameter (or in uppercase Parameter), do not modify it, because the burning tool 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 rest of the images can be configured directly according to the partition table. Try to use the partition names defined in the parameter.txt file (actually, you can directly import the parameter.txt file, it will be automatically generated, we just need to configure the corresponding path of the image).

After the configuration is completed, we can also export these configuration information and save it to a.cfg file for easy re-import next time. The export method is quite simple. The steps are as shown in Figures 5.2.4 to 5.2.5:

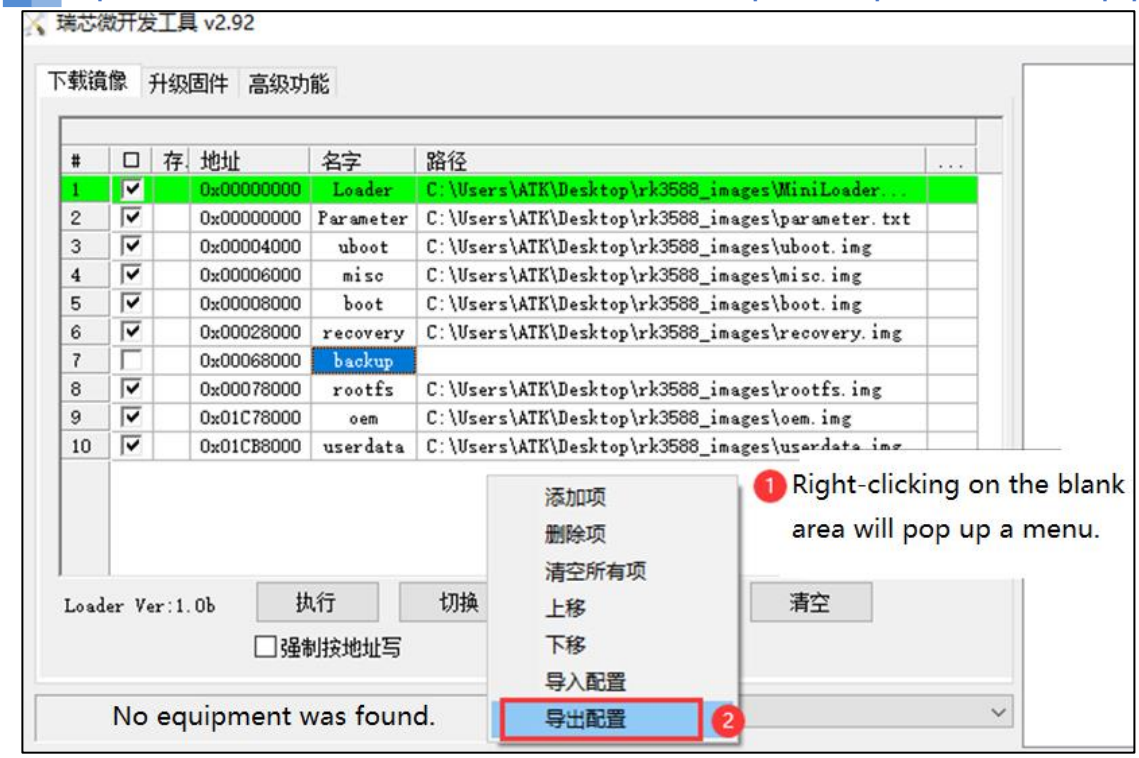


Figure 5.2-4 Export configuration (1)

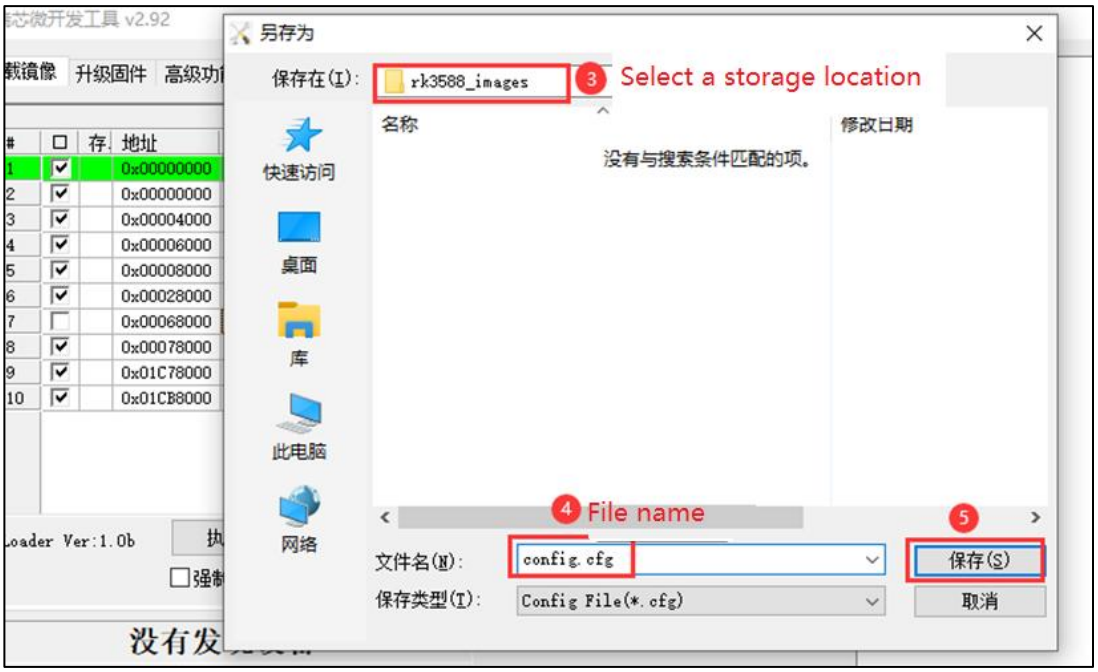


Figure 5.2-5 Export configuration (2)

Export and save the configuration information to the "config.cfg" file, which makes it convenient to directly import this configuration information in the next time. There is no need to manually configure it every time.

5.2.3 Flashing

First, connect the power adapter and the TYPE-C0 interface to the development board. The specific connection method can be referred to as shown in Figure 3.7.9. There is no need to repeat it here! Before flashing, make the development board enter the Maskrom or Loader mode. The method for the development board to enter the Maskrom or loader mode has been explained in the previous sections.

For example, to flash the image through the Loader mode, first press the V+ button on the development board, then power on or reset the development board. At this time, the device will enter the Loader mode (the Rockchip development tools will prompt "A LOADER device has been detected"), and then click the "Execute" button to flash the image:

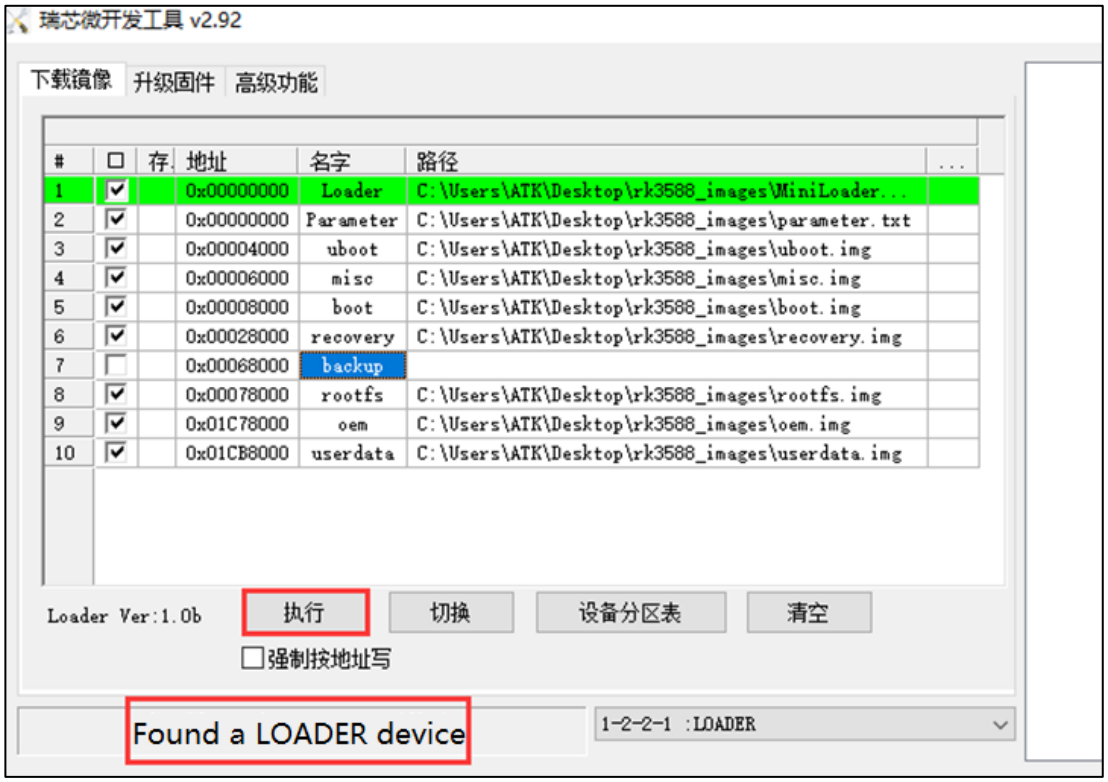


Figure 5.2-6 Perform the burn operation

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

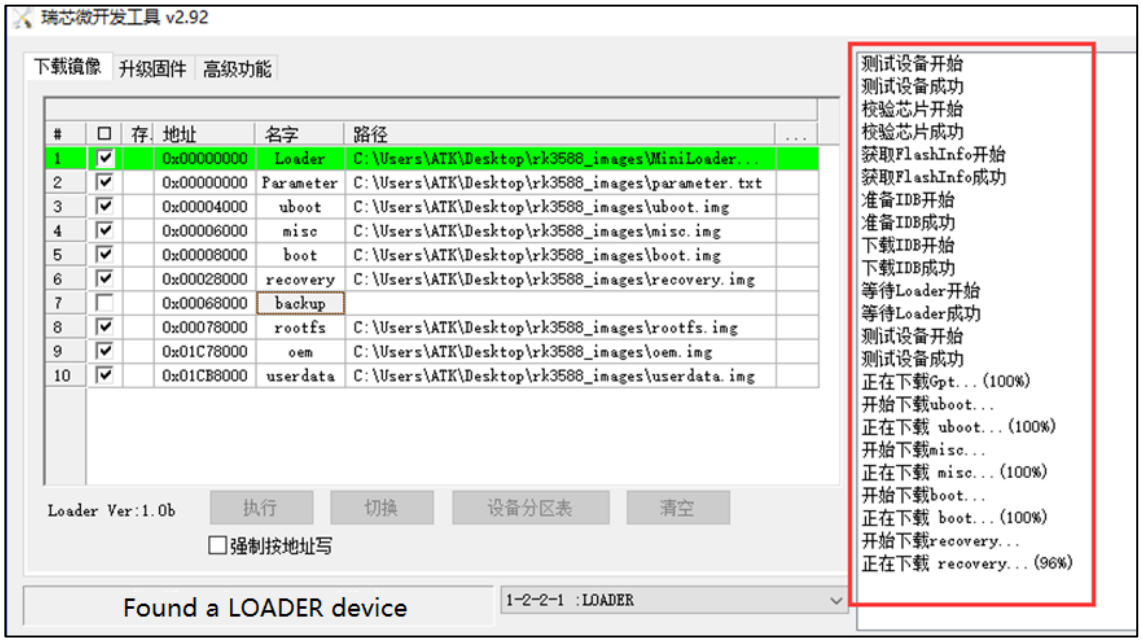


Figure 5.2-7 Log information of the burning process

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

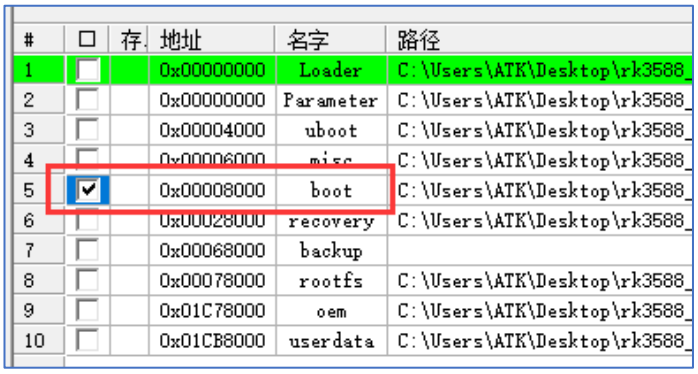


Figure 5.2-8 Burn the boot.img separately

Then click the "Execute" button to perform the burning. This is very useful during the development and debugging process, as in some cases we only need to update a certain partition without having to re-burn the entire system. It should be noted that when updating a single partition in Maskrom mode, it is necessary to check and burn the Loader and Parameter together.

5.2.4 Booting the System

After the burning is completed, the development board will automatically restart, and the serial port terminal will output information:

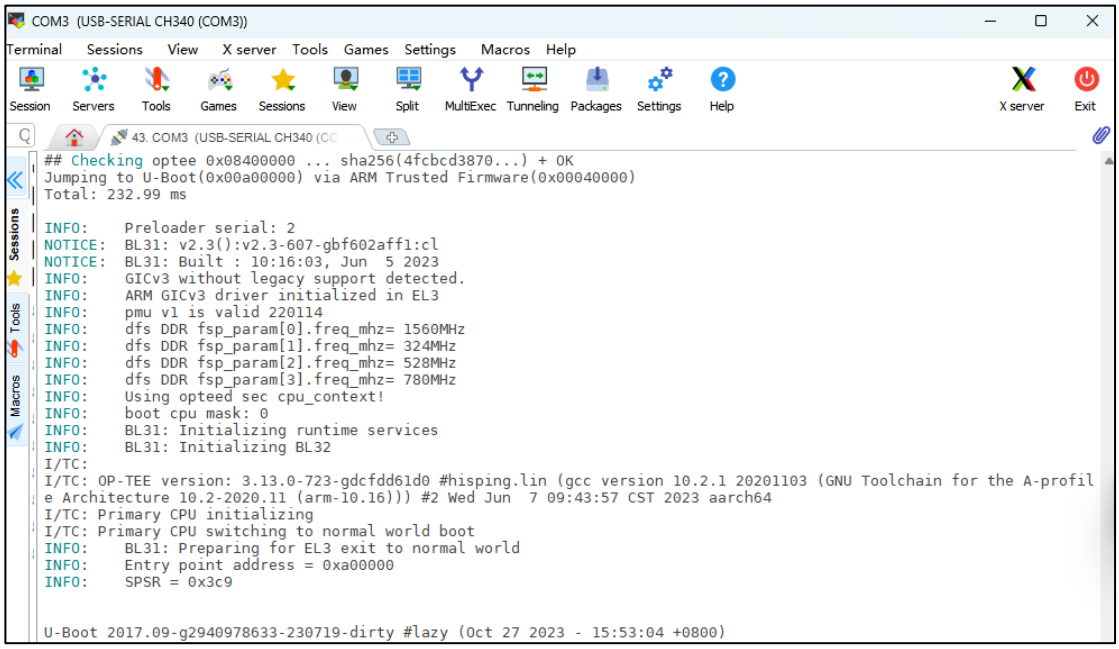


Figure 5.2-9 Enter the system

5.2.5 Burn update.img

You can use the Rockchip development tools to burn the update.img firmware. First, copy the update.img generated in Section 4.4.5 to the Windows environment, then use the Rockchip development tools to burn it onto the development board. The burning method can be referred to in Section 3.7.3.

5.3 Writing on the Ubuntu System

This section will guide users on how to burn the image on the Ubuntu system. When burning on Windows, it is necessary to first copy the image from the Ubuntu system to the Windows system before proceeding with the burning process. If burning on the Ubuntu system, there is no need to copy the image.

As we mentioned earlier, on the Ubuntu system, the Linux_Upgrade_Tool tool can be used for burning. The Linux SDK comes with the Upgrade_Tool burning tool, and its location is: <SDK>/tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool. As shown below:

```
atk@atk-virtual-machine:~/rk3588_linux_sdk$
atk@atk-virtual-machine:~/rk3588_linux_sdk$ cd tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/
atk@atk-virtual-machine:~/rk3588_linux_sdk/tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool$
atk@atk-virtual-machine:~/rk3588_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  revision.txt  upgrade_tool
atk@atk-virtual-machine:~/rk3588_linux_sdk/tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool$
```

Figure 5.3-1 Linux_Upgrade_Tool Directory

This directory contains a user manual provided by RK, which details how to use the upgrade_tool tool. So, for detailed usage instructions, please refer to this document: "Command Line Development Tools User Manual.pdf".

upgrade_tool has two running modes: command line mode and tool mode. Simply running the upgrade_tool command without any parameters will enter the tool mode. The tool mode is actually an interactive mode. In this mode, users can input different instructions and press enter to execute them.

Then, the corresponding operation triggered by the instruction will be executed, such as downloading the image, erasing the image, reading device information, etc.

When running the upgrade_tool command with parameters (such as UF, UL, DI, DB, etc.), it will enter the command line mode. The command line mode refers to the non-interactive mode. By adding parameters when executing upgrade_tool, these parameters 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

First, connect the hardware (the power adapter and the TYPE-C0 interface of the development board). The development board will enter the Loader or Maskrom mode. Open the Ubuntu system and search for a device named "Fuzhou Rockchip Rockusb Device" under "Virtual Machine -> Removable Devices", as shown in the following figure:

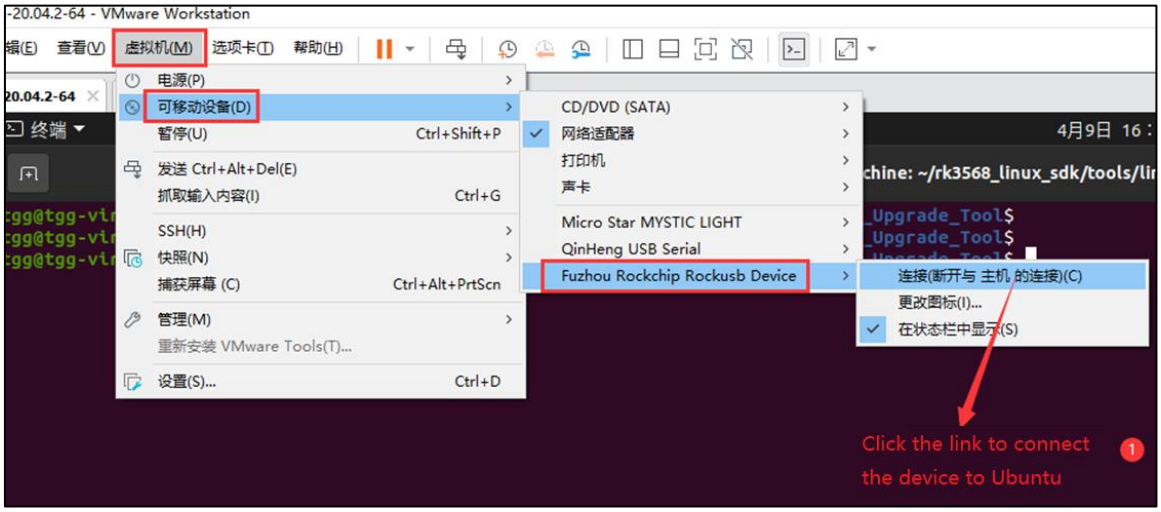


Figure 5.3-2 Connect the device to Ubuntu

If the device does not exist, please check if the hardware connection is incorrect, if the development board is in Maskrom or Loader mode, and try to re-plug the USB cable or change the USB port of the computer. Click "Connect (Disconnect from Host) (C)" with the mouse to connect the development board to Ubuntu.

5.3.2 Using the upgrade_tool tool for flashing (in LOADER mode)

This section provides users with instructions on how to use the upgrade_tool tool to flash the image. Here, we only cover the command-line mode flashing. For ease of operation, we directly enter the output/firmware/ directory (where the image is stored).

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

To flash the MiniLoaderAll.bin image using the UL command, and flash other images (uboot.img, boot.img, OEM.img, userdata.img, rootfs.img, misc.img, etc.) and the partition table file parameter.txt.

First, flash the MiniLoaderAll.bin image using the UL command for flashing (when using upgrade_tool, 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
```

```
atk@atk-virtual-machine:~/rk3588_linux_sdk/output/firmware$ sudo ../../tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/upgrade_tool UL MiniLoaderAll.bin -noreset
[sudo] atk 的密码:
Using /home/atk/rk3588_linux_sdk/tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/config.ini
Loading loader...
Support Type:RK3588     Loader ver:1.0b Loader Time:2024-04-08 16:40:36
Start to upgrade loader...
Download Boot Start
Download Boot Success
Wait For Maskrom Start
Wait For Maskrom Success
Test Device Start
Test Device Success
Check Chip Start
Check Chip Success
Get FlashInfo Start
Get FlashInfo Success
Prepare IDB Start
Prepare IDB Success
Download IDB Start
Download IDB Success
Upgrade loader ok.
atk@atk-virtual-machine:~/rk3588_linux_sdk/output/firmware$
```

Figure 5.3-3 Flash MiniLoaderAll.bin

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

Next, the parameter.txt partition table file is downloaded through the DI instruction:

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

```
atk@atk-virtual-machine:~/rk3588_linux_sdk/output/firmware$ sudo ../../tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/upgrade_tool DI -p parameter.txt
Using /home/atk/rk3588_linux_sdk/tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/config.ini
directlba=1,first4access=1,gpt=1
Write gpt...
Write gpt ok.
atk@atk-virtual-machine:~/rk3588_linux_sdk/output/firmware$
```

Figure 5.3-4 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
```

```
atk@atk-virtual-machine:~/rk3588_linux_sdk/output/firmware$ sudo ../../tools/linux/Linux_Upgrade_Tool/Linux_U
pgrade_Tool/upgrade_tool DI -uboot uboot.img
Using /home/atk/rk3588_linux_sdk/tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/config.ini
directlba=1,first4access=1,gpt=1
Download uboot start...(0x00004000)
Write file...(100%)
Download image ok.
atk@atk-virtual-machine:~/rk3588_linux_sdk/output/firmware$
```

Figure 5.3-5 Burn the uboot.img file

```
atk@atk-virtual-machine:~/rk3588_linux_sdk/output/firmware$ sudo ../../tools/linux/Linux_Upgrade_Tool/Linux_U
pgrade_Tool/upgrade_tool DI -misc misc.img
Using /home/atk/rk3588_linux_sdk/tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/config.ini
directlba=1,first4access=1,gpt=1
Download misc start...(0x00006000)
Write file...(100%)
Download image ok.
atk@atk-virtual-machine:~/rk3588_linux_sdk/output/firmware$
```

Figure 5.3-6 Burn misc.img

```
atk@atk-virtual-machine:~/rk3588_linux_sdk/output/firmware$ sudo ../../tools/linux/Linux_Upgrade_Tool/Linux_U
pgrade_Tool/upgrade_tool DI -boot boot.img
Using /home/atk/rk3588_linux_sdk/tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/config.ini
directlba=1,first4access=1,gpt=1
Download boot start...(0x00008000)
Write file...(100%)
Download image ok.
atk@atk-virtual-machine:~/rk3588_linux_sdk/output/firmware$
```

Figure 5.3-7 Burn boot.img

```
atk@atk-virtual-machine:~/rk3588_linux_sdk/output/firmware$ sudo ../../tools/linux/Linux_Upgrade_Tool/Linux_U
pgrade_Tool/upgrade_tool DI -recovery recovery.img
Using /home/atk/rk3588_linux_sdk/tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/config.ini
directlba=1,first4access=1,gpt=1
Download recovery start...(0x00028000)
Write file...(100%)
Download image ok.
atk@atk-virtual-machine:~/rk3588_linux_sdk/output/firmware$
```

Figure 5.3-8 Burn recovery.img

```
atk@atk-virtual-machine:~/rk3588_linux_sdk/output/firmware$ sudo ../../tools/linux/Linux_Upgrade_Tool/Linux_U
pgrade_Tool/upgrade_tool DI -oem oem.img
Using /home/atk/rk3588_linux_sdk/tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/config.ini
directlba=1,first4access=1,gpt=1
Download oem start...(0x01c78000)
Write file...(100%)
Download image ok.
atk@atk-virtual-machine:~/rk3588_linux_sdk/output/firmware$
```

Figure 5.3-9 Burn oem.img

```
atk@atk-virtual-machine:~/rk3588_linux_sdk/output/firmware$ sudo ../../tools/linux/Linux_Upgrade_Tool/Linux_U
pgrade_Tool/upgrade_tool DI -rootfs rootfs.img
Using /home/atk/rk3588_linux_sdk/tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/config.ini
directlba=1,first4access=1,gpt=1
Download rootfs start...(0x00078000)
Write file...(100%)
Download image ok.
atk@atk-virtual-machine:~/rk3588_linux_sdk/output/firmware$
```

Figure 5.3-10 Burn rootfs.img

```
atk@atk-virtual-machine:~/rk3588_linux_sdk/output/firmware$ sudo ../../tools/linux/Linux_Upgrade_Tool/Linux_U
pgrade_Tool/upgrade_tool DI -userdata userdata.img
Using /home/atk/rk3588_linux_sdk/tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/config.ini
directlba=1,first4access=1,gpt=1
Download userdata start...(0x01cb8000)
Write file...(100%)
Download image ok.
atk@atk-virtual-machine:~/rk3588_linux_sdk/output/firmware$
```

Figure 5.3-11 Burn userdata.img

In "DI -uboot", "uboot" refers to the uboot partition. For example, "-boot" indicates that the image should be burned to the boot partition, and "-misc" indicates that the image should be burned to the misc partition.

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

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

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

```
atk@atk-virtual-machine:~/rk3588_linux_sdk/output/firmware$
atk@atk-virtual-machine:~/rk3588_linux_sdk/output/firmware$ sudo ../../tools/linux/Linux_Upgrade_Tool/Linux_U
pgrade_Tool/upgrade_tool RD
Using /home/atk/rk3588_linux_sdk/tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/config.ini
Reset Device OK.
atk@atk-virtual-machine:~/rk3588_linux_sdk/output/firmware$
atk@atk-virtual-machine:~/rk3588_linux_sdk/output/firmware$
```

Figure 5.3-12 Reset the development board

5.3.3 Flash update.img

First, execute the command "./build.sh firmware" in the root directory of the SDK to package the update.img firmware (the command "./build.sh all" will also generate update.img by default). The generated update.img is saved in the directory "output/firmware/", then enter this directory and execute the following command to flash the update.img firmware (using the UF command for flashing):

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

```
atk@atk-virtual-machine:~/rk3588_linux_sdk/output/firmware$
atk@atk-virtual-machine:~/rk3588_linux_sdk/output/firmware$ sudo ../../tools/linux/Linux_Upgrade_Tool/Linux_U
pgrade_Tool/upgrade_tool UF update.img
Using /home/atk/rk3588_linux_sdk/tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/config.ini
Loading firmware...
Support Type:3588       FW Ver:1.0.00   FW Time:2024-04-06 13:27:07
Loader ver:1.11 Loader Time:2024-04-06 12:01:15
Start to upgrade firmware...
Download Boot Start
Download Boot Success
Wait For Maskrom Start
Wait For Maskrom Success
Test Device Start
Test Device Success
Check Chip Start
Check Chip Success
Get FlashInfo Start
Get FlashInfo Success
Prepare IDB Start
Prepare IDB Success
Download IDB Start
Download IDB Success
Download Firmware Start
Download Image... (100%)
Download Firmware Success
Upgrade firmware ok.
atk@atk-virtual-machine:~/rk3588_linux_sdk/output/firmware$
```

Figure 5.3-13 Burn update.img

After the burning is completed, the development board will automatically reset and start the system.

5.3.4 Erase Operation

This section explains how to use the upgrade_tool tool to perform an erase operation on the Flash storage device (such as eMMC) on the development board. Similarly, before the operation, the development board must be in Maskrom or Loader mode and be connected to Ubuntu.

Executing the following command can erase all data on the eMMC (using the EF instruction):

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

```
atk@atk-virtual-machine:~/rk3588_linux_sdk/output/firmware$ sudo ../../tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/upgrade_tool EF MiniLoaderAll.bin
Using /home/atk/rk3588_linux_sdk/tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/config.ini
Loading loader...
Start to erase flash...
Download Boot Start
Download Boot Success
Wait For Maskrom Start
Wait For Maskrom Success
Test Device Start
Test Device Success
Get FlashInfo Start
Get FlashInfo Success
Erase Flash Start
Erase Flash... (100%)
Erase Flash Success
Reset Device Start
Erase flash ok.
atk@atk-virtual-machine:~/rk3588_linux_sdk/output/firmware$
```

Figure 5.3-14 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

rkflash.sh is a burning script provided by RK. We can directly use this rkflash.sh script for burning; of course, this script also calls the upgrade_tool tool to perform the burning operation.

The usage is very simple. First, let the development board be in the Loader mode, connect the development board to Ubuntu, and then directly run the rkflash.sh script in the SDK root directory to burn the image in the output/firmware/ directory to the development board (also need to add sudo, otherwise the operation will fail!) :

```
sudo ./rkflash.sh
```

```
atk@atk-virtual-machine:~/rk3588_linux_sdk$ sudo ./rkflash.sh
flash all images as default
Using /home/atk/rk3588_linux_sdk/tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/config.ini
Loading loader...
Support Type:RK3588     Loader ver:1.0b Loader Time:2024-04-08 16:40:36
Start to upgrade loader...
Download Boot Start
Download Boot Success
Wait For Maskrom Start
Wait For Maskrom Success
Test Device Start
Test Device Success
Check Chip Start
Check Chip Success
Get FlashInfo Start
Get FlashInfo Success
Prepare IDB Start
Prepare IDB Success
Download IDB Start
Download IDB Success
Upgrade loader ok.
Using /home/atk/rk3588_linux_sdk/tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/config.ini
directlba=1,first4access=1,gpt=1
Write gpt...
Write gpt ok.
Using /home/atk/rk3588_linux_sdk/tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/config.ini
directlba=1,first4access=1,gpt=1
Download uboot start...(0x00004000)
Write file...(100%)
Download image ok.
Using /home/atk/rk3588_linux_sdk/tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/config.ini
check download item failed!
Using /home/atk/rk3588_linux_sdk/tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/config.ini
directlba=1,first4access=1,gpt=1
Download boot start...(0x00008000)
Write file...(100%)
Download image ok.
Using /home/atk/rk3588_linux_sdk/tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/config.ini
directlba=1,first4access=1,gpt=1
Download recovery start...(0x00028000)
Write file...(100%)
Download image ok.
```

```
Using /home/atk/rk3588_linux_sdk/tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/config.ini
directlba=1,first4access=1,gpt=1
Download misc start...(0x00006000)
Write file...(100%)
Download image ok.
Using /home/atk/rk3588_linux_sdk/tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/config.ini
directlba=1,first4access=1,gpt=1
Download oem start...(0x01c78000)
Write file...(100%)
Download image ok.
Using /home/atk/rk3588_linux_sdk/tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/config.ini
directlba=1,first4access=1,gpt=1
Download userdata start...(0x01cb8000)
Write file...(100%)
Download image ok.
Using /home/atk/rk3588_linux_sdk/tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/config.ini
directlba=1,first4access=1,gpt=1
Download rootfs start...(0x00078000)
Write file...(100%)
Download image ok.
Using /home/atk/rk3588_linux_sdk/tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/config.ini
Reset Device OK.
atk@atk-virtual-machine:~/rk3588_linux_sdk$
```

Figure 5.3-15 Execute rkflash.sh to burn all the images.

Executing the above command will burn all the individual images (boot.img, MiniLoaderAll.bin, misc.img, OEM.img, recovery.img, rootfs.img, uboot.img, userdata.img) in the output/firmware/ directory onto the development board.

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

In addition, you can also burn a specific designated 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 (that 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 firmware package
sudo ./rkflash.sh erase	Erase all data in the Flash storage device

Figure 5.3.16 Usage of the rkflash.sh script

Chapter 6. SDK Development

This chapter provides users with instructions on how to develop the QuarkPi-CA2 Linux SDK, including 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 QuarkPi-CA2 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 Board-level Configuration of SDK

The board-level configuration file of SDK is stored in the directory <SDK_PATH>/device/rockchip/rk3588/, as shown below:

```
lixinkai@lixinkai-virtual-machine:~/atk_quarkpi_ca2_buildroot/device/rockchip/rk3588$ ls
alientek_quarkpi_ca2_defconfig  boot.its                                rockchip_rk3588_evb7_v11_defconfig
alientek_rk3588_defconfig       parameter-ab.txt                        rockchip_rk3588_ipc_evb1_v10_defconfig
amp_linux.its                   parameter.txt                           rockchip_rk3588_multi_ipc_evb1_v10_defconfig
amp_mcu.its                     rockchip_defconfig                      rockchip_rk3588s_evb1_lp4x_v10_defconfig
boot4recovery.its               rockchip_rk3588_evb1_lp4_v10_defconfig  zboot.its
lixinkai@lixinkai-virtual-machine:~/atk_quarkpi_ca2_buildroot/device/rockchip/rk3588$
```

There are multiple *_defconfig configuration files in this directory, corresponding to the configuration list printed by the "./build.sh lunch" command. The configuration file for the ALIENTEK QuarkPi-CA2 card computer is alientek_quarkpi_ca2_defconfig. The content is as follows:

```
lixinkai@lixinkai-virtual-machine:~/atk_quarkpi_ca2_buildroot/device/rockchip/rk3588$ cat alientek_quarkpi_ca2_defconfig
RK_YOCTO_CFG="rockchip-rk3588-evb"
RK_KERNEL_DTS_NAME="rk3588s-atk-quarkpi-ca2"
RK_USE_FIT_IMG=y
RK_TARGET_BOARD="quarkpi-ca2"
RK_BUILD_JOBS="24"
RK_BUILDROOT_BASE_CFG="atk_quarkpi_ca2"
RK_ROOTFS_HOSTNAME_CUSTOM=y
RK_ROOTFS_HOSTNAME="quarkpi-ca2"
lixinkai@lixinkai-virtual-machine:~/atk_quarkpi_ca2_buildroot/device/rockchip/rk3588$
```

This file defines the kernel device tree file and other configuration information used by the QuarkPi-CA2 platform; only a small portion of the configuration information is defined in this file, while the majority of the configuration information remains in its default state.

We can also execute the "make menuconfig" command in the root directory of the SDK to open the graphical configuration interface and make changes to the configuration:

```
Rockchip Linux SDK Configuration
Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).  Highlighted letters are hotkeys.  Pressing <Y> selects a
feature, while <N> excludes a feature.  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] feature is selected  [ ] feature is
excluded

(quarkpi-ca2) Target board name
(alientek_quarkpi_ca2_defconfig) Name of defconfig to save
[*] Rootfs (Buildroot|Debian|Yocto)  --->
[*] Loader (U-Boot)  --->
[ ] AMP (Asymmetric Multi-Processing System)
[*] Kernel (Embedded in an Android-style boot image)  --->
    Boot (Android-style boot image)  --->
[*] Recovery (based on Buildroot)  --->
[ ] Security feature (secureboot, encryption, verity, etc.)
    Extra partitions (oem, userdata, etc.)  --->
    Firmware (partition table, misc image, etc.)  --->
[*] Update (Rockchip update image)  --->
    Others configurations  --->

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

Users can modify the SDK configuration by themselves. After the modification, execute the "make savedefconfig" command to save the configuration.

6.2 U-Boot Development

The U-Boot source code is stored in the "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

The path where the U-Boot development reference documentation (provided by Rockchip) is located is: <SDK>/docs/cn/Common/UBOOT/Rockchip_Developer_Guide_UBoot_Nextdev_CN.pdf. You can take a look at it.

6.2.1 U-Boot's Device Tree

For U-Boot, the device tree file used by the QuarkPi-CA2 platform is arch/arm/dts/rk3588-evb.dts.

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, that is, U-Boot will use the kernel DTB to initialize the peripherals. The purpose of this design is mainly to accommodate the differences in board-level peripherals, 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 initialization operation of basic peripherals such as storage and debug serial ports is 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 changing the debug serial port, etc.).

6.2.2 U-Boot Compilation

Execute the following command in the U-Boot source code directory to compile U-Boot (SDK_PATH refers to the root directory of the SDK, please modify it to your own SDK root directory, using an absolute path):

```
./make.sh CROSS_COMPILE=<SDK_PATH>/prebuilts/gcc/linux-x86/aarch64/gcc-arm-10.3-2021.07-x86_64-aarch64-none-linux-gnu/bin/aarch64-none-linux-gnu- rk3588
```

```
lixinkai@lixinkai-virtual-machine:~/atk_quarkpi_ca2_buildroot/u-boot$ ./make.sh CROSS_COMPILE=/home/lixinkai/atk_quarkpi_ca2_buildroot/prebuilts/gcc/linux
-x86/aarch64/gcc-arm-10.3-2021.07-x86_64-aarch64-none-linux-gnu/bin/aarch64-none-linux-gnu- rk3588
## make rk3588_defconfig -j32
  HOSTCC  scripts/basic/fixdep
  HOSTCC  scripts/kconfig/conf.o
  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 and 4097 [-Wformat-overflow=]
  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) into a destination of size 4097
   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:
```

Figure 6.2-2 Compile U-Boot (1)

```
  Firmware:     atf-1
  FDT:          fdt
  Loadables:    uboot
                atf-2
                atf-3
                optee
********boot_merger ver 1.35********
Info:Pack loader ok.
pack loader okay! Input: /home/lixinkai/atk_quarkpi_ca2_buildroot/rkbin/RKBOOT/RK3588MINIALL.ini
/home/lixinkai/atk_quarkpi_ca2_buildroot/u-boot

Image(no-signed, version=0): uboot.img (FIT with uboot, trust...) is ready
Image(no-signed): rk3588_spl_loader_v1.17.113.bin (with spl, ddr...) is ready
pack uboot.img okay! Input: /home/lixinkai/atk_quarkpi_ca2_buildroot/rkbin/RKTRUST/RK3588TRUST.ini

Platform RK3588 is build OK, with new .config(make rk3588_defconfig -j32)
/home/lixinkai/atk_quarkpi_ca2_buildroot/prebuilts/gcc/linux-x86/aarch64/gcc-arm-10.3-2021.07-x86_64-aarch64-none-linux-gnu/bin/aarch64-none-linux-gnu-
2024年 12月 18日 星期三 10:26:53 CST
```

Figure 6.2-3 Compiling U-Boot source code (2)

After the compilation is completed, two image files, uboot.img and rk3588_spl_loader_v1.13.112.bin, are generated as follows:

```
lixinkai@lixinkai-virtual-machine:~/atk_quarkpi_ca2_buildroot/u-boot$ ls
api                          cmd            env       MAINTAINERS                  snapshot.commit  u-boot              u-boot-nodtb.bin
arch                         common         examples  Makefile                     spl              u-boot.bin          u-boot-nodtb.bin.digest
bl31_0x00040000.bin          config.mk      fit       make.sh                      System.map       u-boot.cfg          u-boot-nodtb.bin.gz
bl31_0x00040000.bin.digest   configs        fs        net                          tee.bin          u-boot.cfg.configs  u-boot.srec
bl31_0x00040000.bin.gz       disk           include   post                         tee.bin.digest   u-boot.dtb          u-boot.sym
bl31_0x000f0000.bin          doc            Kbuild    PREUPLOAD.cfg                tee.bin.gz       u-boot-dtb.bin      usb_update.txt
bl31_0xff100000.bin          Documentation  Kconfig   README                       test             uboot.img
bl31.elf                     drivers        lib       rk3588_spl_loader_v1.17.113.bin tools         u-boot.lds
board                        dts            Licenses  scripts                      tpl              u-boot.map
lixinkai@lixinkai-virtual-machine:~/atk_quarkpi_ca2_buildroot/u-boot$
```

Figure 6.2-4 The image generated after compiling U-Boot

<SDK_PATH>/output/firmware/uboot.img (soft link) points to uboot.img in the U-Boot source code directory; <SDK_PATH>/output/firmware/MiniLoaderAll.bin (soft link) points to rk3588_spl_loader_v1.17.113.bin in the U-Boot source code directory.

6.2.3 Configuration of U-Boot

Execute the following command in the U-Boot source code directory to open the menuconfig configuration interface:

```
make rk3588_defconfig        // Select configuration file
make menuconfig              // Open the graphical configuration interface
```

The running results are as follows:

```
                                U-Boot 2017.09 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

                        Architecture select (ARM architecture)  --->
                        ARM architecture  --->
                        General setup  --->
                        Boot images  --->
                        API  --->
                        Boot timing  --->
                        Boot media  --->
                    [*] Pass the device serial number to kernel through devicetree
                    (0) delay in seconds before automatically booting
                    [ ] Enable boot arguments
                    [ ] Enable lotrace
                        Console  --->
                        Logging  --->
                    ()  Default fdt file
                    [ ] add U-Boot environment variable vers
                    [ ] Display information about the CPU during start up
                    [*] Display information about the board during start up
                        Start-up hooks  --->
                        Androld features support  --->
                    [ ] Skip U-Boot relocation
                        Security support  ----
                        SPL / TPL  --->
                        USBPLUG  --->
                        Command line interface  --->
                        Partition Types  --->
                        Device Tree Control  --->
                    (dtc) Path to dtc binary for use within mkimage
                        Environment  --->
                    -*- Networking support  --->
                        Device Drivers  --->
                        File systems  --->
                        Library routines  --->
                    [ ] Unit tests  ----
                        <Select>    < Exit >    < Help >    < Save >    < Load >
```

Figure 6.2-5 Menuconfig graphical configuration

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/rk3588_defconfig // Replace rk3588_defconfig with the defconfig file
```

After saving the configuration, execute the following command for compilation:

```
# Compile the U-Boot source code directory using the make.sh script.
./make.sh CROSS_COMPILE=<SDK_PATH>/prebuilts/gcc/linux-x86/aarch64/gcc-arm-10.3-2021.07-x86_64-aarch64-none-linux-gnu/bin/aarch64-none-linux-gnu- rk3588

# Or, you can compile it in the root directory of the SDK using the build.sh script.
cd ../; ./build.sh uboot
```

6.2.4 Shortcuts

The RK platform U-Boot supports triggering certain events through serial port key combinations for flashing and debugging. When running on the RK U-Boot, there is no kernel startup countdown. It skips the countdown and directly starts the Linux kernel. We can interrupt its kernel startup by pressing the serial port key combination "Ctrl + C", thereby entering the U-Boot command line mode.

In addition to the Ctrl + C key combination, the following multiple key combinations are also supported (press multiple times when powering on or resetting to avoid triggering failure):

- Ctrl + C: Enter U-Boot command line mode.
- Ctrl + D: Enter Loader flashing mode.
- Ctrl + B: Enter 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: After U-Boot prints "Starting kernel...", it enters the U-Boot command line mode.

You can test it yourself. No more details will be provided here.

6.3 Kernel Development

The Linux kernel source code is stored in the <SDK_PATH>/kernel/ directory:

```
atk@atk-virtual-machine:~/rk3588_linux_sdk$
atk@atk-virtual-machine:~/rk3588_linux_sdk$ cd kernel/
atk@atk-virtual-machine:~/rk3588_linux_sdk/kernel$
atk@atk-virtual-machine:~/rk3588_linux_sdk/kernel$ ls
android                                    build.config.gki-debug.aarch64    Documentation        Makefile
arch                                       build.config.gki-debug.x86_64     drivers              make.sh
block                                      build.config.gki_kasan            fs                   mm
boot.its                                   build.config.gki_kasan.aarch64    include              net
build.config.aarch64                       build.config.gki_kasan.x86_64     init                 OWNERS
build.config.allmodconfig                  build.config.gki_kprobes          io_uring             README
build.config.allmodconfig.aarch64          build.config.gki_kprobes.aarch64  ipc                  README.md
build.config.allmodconfig.arm              build.config.gki_kprobes.x86_64   Kbuild               samples
build.config.allmodconfig.x86_64           build.config.gki.x86_64           Kconfig              scripts
build.config.amlogic                       build.config.hikey960             kernel               security
build.config.arm                           build.config.khwasan              kernel_build_atk.sh  sound
build.config.common                        build.config.rockchip             kernel_build.sh      tools
build.config.db845c                        build.config.x86_64               lib                  usr
build.config.gki                           certs                             LICENSES             virt
build.config.gki.aarch64                   COPYING                           logo.bmp
build.config.gki.aarch64.fips140           CREDITS                           logo_kernel.bmp
build.config.gki.aarch64.fips140_eval_testing  crypto                        MAINTAINERS
atk@atk-virtual-machine:~/rk3588_linux_sdk/kernel$
```

Figure 6.3-1 Kernel source code directory

6.3.1 Compilation of the Linux kernel

Refer to Section 4.4.2

6.3.2 Kernel Configuration

In the kernel source code directory, execute the following command to open the graphical configuration interface of menuconfig:

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

The running results are as follows:

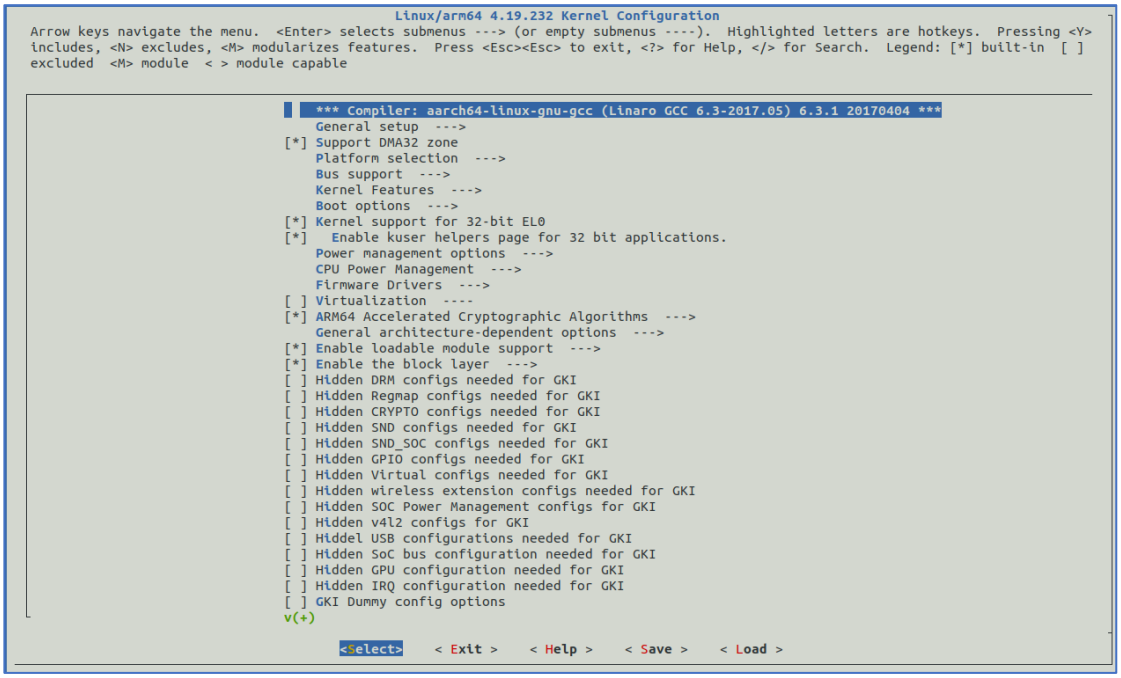


Figure 6.3-2 The menuconfig graphical configuration interface

Users can configure the Linux kernel by themselves. 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, execute the following command to compile the kernel:

```
cd ../ && ./build.sh kernel      // Return to the root directory of the SDK, and call the build.sh script to compile the kernel.
```

6.3.3 Kernel Device Tree and defconfig Configuration File

For the Linux kernel, the defconfig configuration file used by the ALIENTEK QuarkPi-CA2 platform is: arch/arm64/configs/rockchip_linux_defconfig.

The device tree files used by the Rockchip platform are all stored in the directory arch/arm64/boot/dts/rockchip/. The corresponding device tree file for the ALIENTEK QuarkPi-CA2 platform is arch/arm64/boot/dts/rockchip/rk3588s-atk-quarkpi-ca2.dts.

rk3588s-atk-quarkpi-ca2.dts.dts is the top-level device tree file, and it contains multiple.dtsi device trees. Its structure is as follows:

```
rk3588s-atk-quarkpi-ca2.dts
    rk3588s-atk-quarkpi-ca2.dtsi
        rk3588s.dtsi
        rk3588s-evb.dtsi
        rk3588-rk806-single.dtsi
    rk3588s-atk-camera.dtsi
    rk3588s-atk-screen_choose.dtsi
    rk3588s-atk-lcds.dtsi
    rk3588-linux.dtsi
```

- rk3588s.dtsi: This device tree file is for the RK3588S platform-level device tree. It is independent of specific board-level hardware and is a pure SoC-level device tree file provided by RK. Developers do not need to modify this file!
- rk3588s-evb.dtsi: This is the general board-level device tree file for the RK3588S platform, usually included in the board-level device tree files;
- rk3588-rk806-single.dtsi: Configuration information related to rk806 (PMIC);
- rk3588-linux.dtsi: This device tree contains the configuration information specific to the Linux part (distinguishing it from Android);
- rk3588s-atk-quarkpi-ca2.dtsi: Board-level device tree file. This device tree file includes the general board-level device tree file rk3588s-evb.dtsi and the platform-level device tree file rk3588s.dtsi of the RK3588S platform;
- rk3588s-atk-screen_choose.dtsi: This device tree is used to select the LCD screens that need to be enabled, such as HDMI, MIPI, DP, etc.;
- rk3588s-atk-lcds.dtsi: This device tree contains configuration information related to LCDs, such as touch screens, backlight, LCD timing parameters, display interfaces, etc.;
- rk3588s-atk-cameras.dtsi: This device tree contains the configuration information related to the cameras of the development board.

The rk3588s-atk-screen_choose.dtsi file defines some control macros. These macros are used to control whether to enable the corresponding display interface. Users can modify this file themselves to select the LCD screens that need to be enabled. The content of this file is as follows:

```
/*
 * RK3588s平台支持4路独立的Video Port，可实现4路独立的显示输出
 * 显示通路默认配置情况如下：
 * ----------------------------------------
 * |  VP0 -----------------> HDMI_TX0      |
 * |  VP1 -----------------> DP_TX0        |
 * |  VP2 -----------------> DSI_TX0       |
 * |  VP3 -----------------> DSI_TX1       |
 * ----------------------------------------
 * 如需修改显示通路，配置本文件即可！
 *
 *
 * 通过宏定义使能相应的显示接口：
 * ATK_LCD_TYPE_MIPI_TX0_5P5_720X1280：使能DSI_TX0、支持正点原子5.5寸720x1280 MIPI屏
 * ATK_LCD_TYPE_MIPI_TX0_5P5_1080X1920：使能DSI_TX0、支持正点原子5.5寸1080x1920 MIPI屏
 * ATK_LCD_TYPE_MIPI_TX0_10P1_800X1280：使能DSI_TX0、支持正点原子10.1寸800x1280 MIPI屏
 * ATK_LCD_TYPE_HDMI_TX0：使能HDMI_TX0
 * ATK_LCD_TYPE_DP_TX0：使能DP_TX0
 *
 */

#define ATK_LCD_TYPE_MIPI_TX0_5P5_720X1280
#define ATK_LCD_TYPE_MIPI_TX1_5P5_720X1280
// #define ATK_LCD_TYPE_MIPI_TX0_5P5_1080X1920
// #define ATK_LCD_TYPE_MIPI_TX1_5P5_1080X1920
// #define ATK_LCD_TYPE_MIPI_TX0_10P1_800X1280
// #define ATK_LCD_TYPE_MIPI_TX1_10P1_800X1280
#define ATK_LCD_TYPE_HDMI_TX0
#define ATK_LCD_TYPE_DP_TX0
```

As can be seen from the figure, the default settings have enabled MIPI_TX0 (which supports a 5.5-inch 720x1280 MIPI screen), MIPI_TX1 (also supporting a 5.5-inch 720x1280 MIPI screen), HDMI_TX0 (for connecting an HDMI display), and DP_TX0 (corresponding to TYPE-C0 on the development board, for connecting a DP display).

6.3.4 Replacing the logo

The boot logo includes the u-boot stage logo and the kernel stage logo. Both of these logo files are located in the kernel source code directory:

```
atk@atk-virtual-machine:~/rk3588_linux_sdk/kernel$
atk@atk-virtual-machine:~/rk3588_linux_sdk/kernel$
atk@atk-virtual-machine:~/rk3588_linux_sdk/kernel$ ls -l *.bmp
-rw-rw-r-- 1 atk atk 177760 4月   3 15:33 logo.bmp
-rw-rw-r-- 1 atk atk 177760 4月   3 15:33 logo_kernel.bmp
atk@atk-virtual-machine:~/rk3588_linux_sdk/kernel$
```

Figure 6.3-3 Logo image file

The default logo is as follows:

Figure 6.3-4 U-Boot logo

Figure 6.3-5 Kernel logo

If the user needs to change the startup logo, they should first rename their U-Boot logo file (a bmp image) to "logo.bmp", and rename their kernel logo file (a bmp image) to "logo_kernel.bmp". Then, copy these two logos to the kernel source code directory and replace the default logo!

When compiling the Linux kernel, the "logo.bmp" and "logo_kernel.bmp" files will be packaged into the "resource.img" resource image.

During U-Boot operation, these two bmp files will be loaded into memory. The logo.bmp will start to display at the beginning of the U-Boot stage, and the address of logo_kernel.bmp in memory will be passed by U-Boot to the Linux Kernel, which will then be displayed during the initialization stage of the DRM driver in the Linux Kernel.

6.3.5 Summary of Kernel Module Development Documentation

The docs/cn/Common directory of the SDK contains many kernel module development documents provided by RK. These documents are placed in different subdirectories according to functional modules, making it convenient for users to search for them.

The following table provides a simple summary of 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 power 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
串口	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

Figure 6.3.5 Summary of Kernel Module Development Documents

The table only lists some of the documents, not all. Users can browse the contents by themselves in the <SDK>/docs/cn/Common directory!

6.4 Buildroot Development

The source code of Buildroot is stored in the <SDK_PATH>/buildroot directory:

```
atk@atk-virtual-machine:~/rk3588_linux_sdk$
atk@atk-virtual-machine:~/rk3588_linux_sdk$ cd buildroot/
atk@atk-virtual-machine:~/rk3588_linux_sdk/buildroot$
atk@atk-virtual-machine:~/rk3588_linux_sdk/buildroot$ ls
arch      board  build    Config.in         configs  DEVELOPERS  docs         fs     Makefile         output   README          support  toolchain
archives  boot   CHANGES  Config.in.legacy  COPYING  dl          envsetup.sh  linux  Makefile.legacy  package  README.rockchip system   utils
atk@atk-virtual-machine:~/rk3588_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 RK Buildroot.

The reference documentation for Buildroot development provided by RK:

<SDK_PATH>/docs/cn/Linux/System/Rockchip_Developer_Guide_Buildroot_CN.pdf

During the process of compiling the RK3588S Linux SDK, two builds of buildroot need to be carried out:

- Compile buildroot to obtain rootfs.img. This rootfs.img will be burned onto the rootfs partition of the development board and serves as the root file system that is mounted during normal boot mode;
- Compile buildroot to obtain ramdisk (this is also the root file system. It is the small-sized root file system that is mounted when entering the recovery mode, while rootfs.img is the root file system that is mounted during normal boot mode. This point needs to be clarified!) 。 The ramdisk will eventually be packaged into the recovery.img image file, 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:

```
atk@atk-virtual-machine:~/rk3588_linux_sdk/buildroot$
atk@atk-virtual-machine:~/rk3588_linux_sdk/buildroot$ ls
arch      boot     Config.in         COPYING     docs         linux            output   README.rockchip  toolchain
archives  build    Config.in.legacy  DEVELOPERS  envsetup.sh  Makefile         package  support          utils
board     CHANGES  configs           dl          fs           Makefile.legacy  README   system
atk@atk-virtual-machine:~/rk3588_linux_sdk/buildroot$
```

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

Top-level directory name	Explanation
arch	Stores all the configuration files and build scripts related to the CPU architectures supported by Buildroot
board	Stores files specific to the target platform, such as kernel configurations or patch files, rootfs override files, etc.
boot	Stores patches, verification files, build scripts, configuration options 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 third-party open-source software packages downloaded, such as the alsa-lib library, the bluez library, bzip2, the 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 it 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 time the compilation is performed (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 consume 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 for 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	Stores the build scripts and related files for various cross-compilation toolchains, including binutils, gcc, gdb, kernel-header and uClibc.
utils	Store some useful 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:

```
atk@atk-virtual-machine:~/rk3588_linux_sdk/buildroot$
atk@atk-virtual-machine:~/rk3588_linux_sdk/buildroot$ ls
arch       build              configs     docs         Makefile         README           toolchain
archives   CHANGES            COPYING     envsetup.sh  Makefile.legacy  README.rockchip  utils
board      Config.in          DEVELOPERS  fs           output           support
boot       Config.in.legacy   dl          linux        package          system
atk@atk-virtual-machine:~/rk3588_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 Compile the root file system

Before compiling, set the environment variables. Execute the following command in the Buildroot source code directory (you only need to execute this command once in the same terminal)

```
source ./envsetup.sh rockchip_atk_quarkpi_ca2
```

The last parameter in the command (rockchip_atk_quarkpi_ca2) is used to specify the target platform for compilation. The defconfig configuration files for all platforms are stored in the "configs" directory. In this directory, you can find the configuration file rockchip_atk_quarkpi_ca2_defconfig corresponding to the ALIENTEK QuarkPi-CA2 platform, as shown below:

```
lixinkai@lixinkai-virtual-machine:~/atk_quarkpi_ca2_buildroot/buildroot$
lixinkai@lixinkai-virtual-machine:~/atk_quarkpi_ca2_buildroot/buildroot$ ls configs/rockchip_atk_quarkpi_ca2_defconfig -l
-rw-rw-r-- 1 lixinkai lixinkai 793 12月 17 15:01 configs/rockchip_atk_quarkpi_ca2_defconfig
lixinkai@lixinkai-virtual-machine:~/atk_quarkpi_ca2_buildroot/buildroot$
```

Execute the "make" or "make all" command to compile the entire root file system:

```
make
```

Or

```
make all
```

Using multi-task simultaneous compilation (-jN):

```
make -j12
```

Or

```
make all -j12
```

② Compile package (compile a specific software package)

Execute the following command in the buildroot source code directory to set the environment variables (this can be done only once in the same terminal):

```
source ./envsetup.sh rockchip_atk_quarkpi_ca2
```

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 Clear Command

Executing the following command can remove all the files generated during the build process (including build, host, staging and target trees, images, and toolchain):

```
make clean
```

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

```
make distclean
```

Executing the following command can clear the build directory of the specified package:

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

6.4.3 Compiling the root file system

Execute the following command in the buildroot source code directory to set the environment variables (if the command has been executed in the current terminal, there is no need to repeat the execution):

```
source ./envsetup.sh rockchip_atk_quarkpi_ca2
```

```
lixinkai@lixinkai-virtual-machine:~/atk_quarkpi_ca2_buildroot/buildroot$ source ./envsetup.sh rockchip_atk_quarkpi_ca2

############## Rockchip Buildroot SDK ##############

make: 进入目录“/home/lixinkai/atk_quarkpi_ca2_buildroot/buildroot”
  GEN     /home/lixinkai/atk_quarkpi_ca2_buildroot/buildroot/output/rockchip_atk_quarkpi_ca2/Makefile
Parsing defconfig: /home/lixinkai/atk_quarkpi_ca2_buildroot/buildroot/configs/rockchip_atk_quarkpi_ca2_defconfig
Using configs/rockchip/base/kernel.config as base
Merging configs/rockchip/fs/e2fs.config
Merging configs/rockchip/base/common.config
Merging configs/rockchip/base/base.config
Merging configs/rockchip/chips/rk3588.config
Value of BR2_ROOTFS_OVERLAY is redefined by configs/rockchip/chips/rk3588.config:
Previous value: BR2_ROOTFS_OVERLAY="board/rockchip/common/base"
Modify value:   BR2_ROOTFS_OVERLAY+="board/rockchip/rk3588/fs-overlay/"
New value:      BR2_ROOTFS_OVERLAY="board/rockchip/common/base board/rockchip/rk3588/fs-overlay/"

Merging configs/rockchip/chips/rk3588_aarch64.config
Merging configs/rockchip/font/font.config
Merging configs/rockchip/font/chinese.config
Merging configs/rockchip/fs/exfat.config
Merging configs/rockchip/fs/ntfs.config
Merging configs/rockchip/fs/vfat.config
Merging configs/rockchip/gpu/gpu.config
Merging configs/rockchip/multimedia/audio.config
Merging configs/rockchip/multimedia/camera.config
Merging configs/rockchip/multimedia/gst/audio.config
Merging configs/rockchip/multimedia/gst/camera.config
Merging configs/rockchip/multimedia/gst/rtsp.config
```

Figure 6.4-2 Execute the defconfig configuration (1)

```
Merging /home/lixinkai/atk_quarkpi_ca2_buildroot/buildroot/configs/rockchip_atk_quarkpi_ca2_defconfig
#
# merged configuration written to /home/lixinkai/atk_quarkpi_ca2_buildroot/buildroot/output/rockchip_atk_quarkpi_ca2/.config.in (needs make)
#
/home/lixinkai/atk_quarkpi_ca2_buildroot/buildroot/output/rockchip_atk_quarkpi_ca2/.config.in:240:warning: override: reassigning to symbol BR2_PACKAGE_QT5
/home/lixinkai/atk_quarkpi_ca2_buildroot/buildroot/output/rockchip_atk_quarkpi_ca2/.config.in:241:warning: override: reassigning to symbol BR2_PACKAGE_QT5
CONNECTIVITY
/home/lixinkai/atk_quarkpi_ca2_buildroot/buildroot/output/rockchip_atk_quarkpi_ca2/.config.in:242:warning: override: reassigning to symbol BR2_PACKAGE_QT5
MULTIMEDIA
/home/lixinkai/atk_quarkpi_ca2_buildroot/buildroot/output/rockchip_atk_quarkpi_ca2/.config.in:243:warning: override: reassigning to symbol BR2_PACKAGE_QT5
QUICKCONTROLS
/home/lixinkai/atk_quarkpi_ca2_buildroot/buildroot/output/rockchip_atk_quarkpi_ca2/.config.in:244:warning: override: reassigning to symbol BR2_PACKAGE_QT5
BASE_JPEG
/home/lixinkai/atk_quarkpi_ca2_buildroot/buildroot/output/rockchip_atk_quarkpi_ca2/.config.in:245:warning: override: reassigning to symbol BR2_PACKAGE_QT5
BASE_PNG
/home/lixinkai/atk_quarkpi_ca2_buildroot/buildroot/output/rockchip_atk_quarkpi_ca2/.config.in:246:warning: override: reassigning to symbol BR2_PACKAGE_QT5
SVG
/home/lixinkai/atk_quarkpi_ca2_buildroot/buildroot/output/rockchip_atk_quarkpi_ca2/.config.in:247:warning: override: reassigning to symbol BR2_PACKAGE_QT5
VIRTUALKEYBOARD
/home/lixinkai/atk_quarkpi_ca2_buildroot/buildroot/output/rockchip_atk_quarkpi_ca2/.config.in:265:warning: override: reassigning to symbol BR2_PACKAGE_QT5
BASE_FONTCONFIG
#
# configuration written to /home/lixinkai/atk_quarkpi_ca2_buildroot/buildroot/output/rockchip_atk_quarkpi_ca2/.config
#
make: 离开目录“/home/lixinkai/atk_quarkpi_ca2_buildroot/buildroot”
```

Figure 6.4-3 Execute the defconfig configuration (2)

Specify the target compilation platform as rockchip_atk_quarkpi_ca2 (the ALIENTEK ATK Quarkpi CA2 platform).

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

```
make -j16
```

```
lixinkai@lixinkai-virtual-machine:~/atk_quarkpi-ca2_buildroot/buildroot$ make -j32
RKPARTYBOX_CONF_OPTS is
/usr/bin/make -j1 O=/home/lixinkai/atk_quarkpi-ca2_buildroot/buildroot/output/rockchip_atk_quarkpi_ca2 HOSTCC="/usr/bin/gcc" HOSTCXX="/usr
nfig
make[2]: warning: -j1 forced in submake: resetting jobserver mode.
  GEN     /home/lixinkai/atk_quarkpi-ca2_buildroot/buildroot/output/rockchip_atk_quarkpi_ca2/Makefile
```

Figure 6.4-4 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 picture:

```
Parallel mksquashfs: Using 17 processors
Creating 4.0 filesystem on /home/lixinkai/atk_quarkpi-ca2_buildroot/buildroot/output/rockchip_atk_quarkpi_ca2/images/rootfs.squashfs, block size 131072.
[=============================================================================================================================\] 19668/19668 100%

Exportable Squashfs 4.0 filesystem, gzip compressed, data block size 131072
        compressed data, compressed metadata, compressed fragments,
        compressed xattrs, compressed ids
        duplicates are removed
Filesystem size 600668.82 Kbytes (586.59 Mbytes)
        48.86% of uncompressed filesystem size (1229381.70 Kbytes)
Inode table size 160883 bytes (157.11 Kbytes)
        32.30% of uncompressed inode table size (498163 bytes)
Directory table size 136066 bytes (132.88 Kbytes)
        43.28% of uncompressed directory table size (314362 bytes)
Number of duplicate files found 198
Number of inodes 13980
Number of files 11604
Number of fragments 1383
Number of symbolic links  1159
Number of device nodes 0
Number of fifo nodes 0
Number of socket nodes 0
Number of directories 1217
Number of ids (unique uids + gids) 6
Number of uids 5
        root (0)
        unknown (1003)
        lixinkai (1000)
        unknown (1002)
        www-data (33)
Number of gids 5
        root (0)
        unknown (1006)
        lixinkai (1000)
        unknown (1002)
        www-data (33)
lixinkai@lixinkai-virtual-machine:~/atk_quarkpi-ca2_buildroot/buildroot$
```

Figure 6.4-5 The root file system has been compiled successfully.

After the compilation is completed, a root file system image is generated and the image is stored in the directory "output/rockchip_atk_quarkpi_ca2/images/":

```
lixinkai@lixinkai-virtual-machine:~/atk_quarkpi-ca2_buildroot/buildroot$ ls output/rockchip_atk_quarkpi_ca2/images/ -l
总用量 3722052
-rw-r--r-- 1 lixinkai lixinkai 1260171264 3月  11 18:06 rootfs.cpio
-rw-r--r-- 1 lixinkai lixinkai  612038866 3月  11 18:08 rootfs.cpio.gz
-rw-r--r-- 1 lixinkai lixinkai 1518338048 3月  11 18:08 rootfs.ext2
lrwxrwxrwx 1 lixinkai lixinkai         11 3月  11 18:08 rootfs.ext4 -> rootfs.ext2
-rw-r--r-- 1 lixinkai lixinkai  615088128 3月  11 18:09 rootfs.squashfs
lixinkai@lixinkai-virtual-machine:~/atk_quarkpi-ca2_buildroot/buildroot$
```

Figure 6.4-6 Rootfs image

The file <SDK_PATH>/output/firmware/rootfs.img (soft link) points to the rootfs.ext2 in the above picture.

To compile directly using the make command, the compilation process will generate a large amount of output information, which looks very messy and is quite annoying! If you want to filter out these large amounts of useless output information 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 unimportant output information and retain the key information, making the terminal output content more concise.

6.4.4 Configuration of rootfs

In the buildroot source code directory, execute the following command to open the menuconfig graphical configuration interface:

```
source ./envsetup.sh rockchip_atk_quarkpi_ca2      // Set environment variables
make menuconfig                                    // Open the menuconfig graphical configuration interface
```

The running results are as follows:

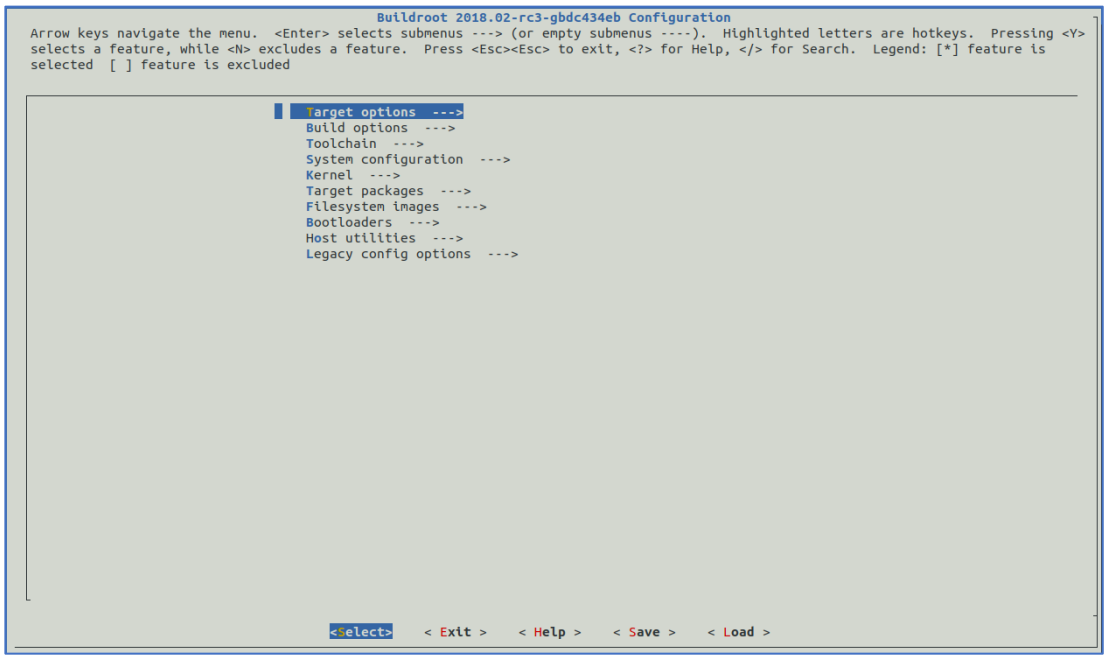


Figure 6.4-7 The menuconfig graphical configuration interface

Users can configure the buildroot themselves. After the configuration is completed, execute the following command to save the configuration:

```
make savedefconfig
```

Next, go back to the SDK root directory and run the build.sh script to compile the buildroot (you can also follow the steps in Section 4.4.3 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 Introduction to the output directory

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.

The structure of the output directory is as follows:

```
output/
├── rockchip_atk_dlrk3588
│   ├── 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 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 Compiling the package

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

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
```

```
atk@atk-virtual-machine:~/rk3588_linux_sdk/buildroot$
atk@atk-virtual-machine:~/rk3588_linux_sdk/buildroot$ utils/brmake rkwifibt
2024-04-10T09:15:08 >>> rkwifibt 1.0.0 Syncing from source dir /home/atk/rk3588_linux_sdk/buildroot/../external/rkwifibt
2024-04-10T09:15:11 >>> rkwifibt 1.0.0 Configuring
2024-04-10T09:15:12 >>> rkwifibt 1.0.0 Building
2024-04-10T09:15:12 >>> rkwifibt 1.0.0 Installing to target
Done in 8s
atk@atk-virtual-machine:~/rk3588_linux_sdk/buildroot$
```

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, as shown below:

```
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
```

```
atk@atk-virtual-machine:~/rk3588_linux_sdk/buildroot$ utils/brmake rkwifibt-configure
2024-04-10T09:18:35 >>> rkwifibt 1.0.0 Syncing from source dir /home/atk/rk3588_linux_sdk/buildroot/../external/rkwifibt
2024-04-10T09:18:38 >>> rkwifibt 1.0.0 Configuring
Done in 8s
atk@atk-virtual-machine:~/rk3588_linux_sdk/buildroot$
atk@atk-virtual-machine:~/rk3588_linux_sdk/buildroot$ utils/brmake rkwifibt-build
2024-04-10T09:18:44 >>> rkwifibt 1.0.0 Building
Done in 5s
atk@atk-virtual-machine:~/rk3588_linux_sdk/buildroot$
atk@atk-virtual-machine:~/rk3588_linux_sdk/buildroot$ utils/brmake rkwifibt-install
2024-04-10T09:18:53 >>> rkwifibt 1.0.0 Installing to target
Done in 4s
atk@atk-virtual-machine:~/rk3588_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

```
atk@atk-virtual-machine:~/rk3588_linux_sdk/buildroot$
atk@atk-virtual-machine:~/rk3588_linux_sdk/buildroot$ utils/brmake rkwifibt-reconfigure
2024-04-10T09:20:02 >>> rkwifibt 1.0.0 Syncing from source dir /home/atk/rk3588_linux_sdk/buildroot/../external/rkwifibt
2024-04-10T09:20:02 >>> rkwifibt 1.0.0 Configuring
2024-04-10T09:20:02 >>> rkwifibt 1.0.0 Building                    Re-run the configuration -> Compile -> Install
2024-04-10T09:20:03 >>> rkwifibt 1.0.0 Installing to target
Done in 5s
atk@atk-virtual-machine:~/rk3588_linux_sdk/buildroot$
atk@atk-virtual-machine:~/rk3588_linux_sdk/buildroot$ utils/brmake rkwifibt-rebuild
2024-04-10T09:21:16 >>> rkwifibt 1.0.0 Syncing from source dir /home/atk/rk3588_linux_sdk/buildroot/../external/rkwifibt
2024-04-10T09:21:16 >>> rkwifibt 1.0.0 Building
2024-04-10T09:21:16 >>> rkwifibt 1.0.0 Installing to target        Re-run the compilation -> Install
Done in 5s
atk@atk-virtual-machine:~/rk3588_linux_sdk/buildroot$
atk@atk-virtual-machine:~/rk3588_linux_sdk/buildroot$ utils/brmake rkwifibt-reinstall
2024-04-10T09:21:26 >>> rkwifibt 1.0.0 Syncing from source dir /home/atk/rk3588_linux_sdk/buildroot/../external/rkwifibt
2024-04-10T09:21:26 >>> rkwifibt 1.0.0 Installing to target
Done in 5s                                                          Re-run the installation process
atk@atk-virtual-machine:~/rk3588_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:

```
atk@atk-virtual-machine:~/rk3588_linux_sdk/buildroot$
atk@atk-virtual-machine:~/rk3588_linux_sdk/buildroot$ utils/brmake rkwifibt-dirclean
Done in 4s
atk@atk-virtual-machine:~/rk3588_linux_sdk/buildroot$
atk@atk-virtual-machine:~/rk3588_linux_sdk/buildroot$
```

6.4.7 Adding a package

The following provides a simple example to guide users on how to add their own package (software package) in the buildroot/package directory.

1. Developing the source code project

First, enter the app directory under the SDK root directory. 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 6.4-8 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 6.4-9 Create the source file and Makefile

Write a simple test code in the main.c source file, such as printing "Hello World". The content of the Makefile file 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 6.4-10 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 6.4-11 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 packagesdefine 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
define MYPACKAGE_INSTALL_TARGET_CMDS
	$(INSTALL) -D -m 0755 $(@D)/mypackage $(TARGET_DIR)/usr/bin/mypackage
endef

# Indicating that the current software package is a basic framework for a general-purpose software package
$(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 tamper with them. They must be equal to the directory

name where the Config.in and mypackage.mk files are located (mypackage, with lowercase letters converted to uppercase).

The functions of these variables and macros have already been explained. Besides these variables and macros,.mk files can also define other variables or macros. 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 -> 08_user_manual -> 03, Auxiliary Documents -> [ALIENTEK] Buildroot User Manual in Chinese (ALIENTEK Translation)_V1.0.pdf.

$(MAKE): represents the make command;

$(@D): Represents the build directory of the software package. To avoid contaminating the source code project of the software package, before compilation, buildroot will copy <SDK_PATH>/app/mypackage to <Buildroot>/output/rockchip_atk_dlrk3588/build/ directory for building and compiling, and rename it as mypackage-1.0 (1.0 is the version number). Therefore, this "$(@D)" refers to the directory <output/rockchip_atk_dlrk3588>/build/mypackage-1.0.

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

$(TARGET_CC): Represents the cross compiler. On the RK platform, the default cross compiler used is the buildroot cross compiler, and the path of the cross compiler is: <Buildroot>/output/rockchip_atk_dlrk3588/host/bin/aarch64-buildroot-linux-gnu-gcc.

$(INSTALL): Represents the install command.

$(TARGET_DIR): Represents the target directory <Buildroot>/output/rockchip_atk_dlrk3588/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 (ALIENTEK Translation)_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"
```

```
menu "Target packages"

menu "Hardware Platforms"
        source "package/rockchip/Config.in"
        source "package/alientek/Config.in"
endmenu

        source "package/busybox/Config.in"
        source "package/skeleton/Config.in"
        source "package/skeleton-custom/Config.in"
        source "package/skeleton-init-common/Config.in"
        source "package/skeleton-init-none/Config.in"
        source "package/skeleton-init-openrc/Config.in"
        source "package/skeleton-init-systemd/Config.in"
        source "package/skeleton-init-sysv/Config.in"
        source "package/mypackage/Config.in"

menu "Audio and video applications"
        source "package/alsa-utils/Config.in"
        source "package/atest/Config.in"
        source "package/aumix/Config.in"
```

Figure 6.4-12 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 fr
[*]   Show packages that are also provided by busybox
[ ]   Individual binaries
[ ]   Install the watchdog daemon startup script
[*] Rockchip BSP packages  --->
[*] my package   Input Y to enable it.
    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  --->
```

Figure 6.4-13 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 6.4-14 Compile mypackage

Compile to generate an executable file named "mypackage", and its location is: output/rockchip_atk_dlrk3588/target/usr/bin/mypackage.

6.4.8 Customization of rootfs

Users can customize the rootfs for their own Linux products. This enables reducing the size of the rootfs while still meeting the product's functionality requirements. It also avoids installing unnecessary executable programs and libraries that the product does not use, thereby improving the system's stability!

6.5 Recovery Development

The recovery.img file 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 file will be burned onto the recovery partition of the development board. The recovery.img file is also a FIT format image, which is composed of multiple images merged together, including the ramdisk (a small-root file system compiled by Buildroot, which is mounted when entering the recovery mode), the kernel DTB, the kernel image Image, and the resource image 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 necessary to enter the recovery mode, then U-Boot will load and boot the recovery.img to enter the recovery mode (start the kernel image contained in the recovery.img and eventually mount the ramdisk root file system contained in the recovery.img).

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

6.5.1 Compilation of 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 to obtain the kernel image Image, kernel DTB, and resource.img;

② Compile buildroot to obtain the ramdisk root file system;

④ Finally, package them into recovery.img.

The compilation process can be referred to in Sections 4.4.2, 4.4.3, and 4.4.4 respectively.

6.5.2 Configuration of recovery

In the buildroot source code directory, execute the following command to configure the ramdisk root file system (normally, we do not need to configure the ramdisk; keeping it in the default state is sufficient):

```
source build/envsetup.sh rockchip_rk3588_recovery          //
make menuconfig          // Open the menuconfig graphical configuration interface
```

The running results are as follows:

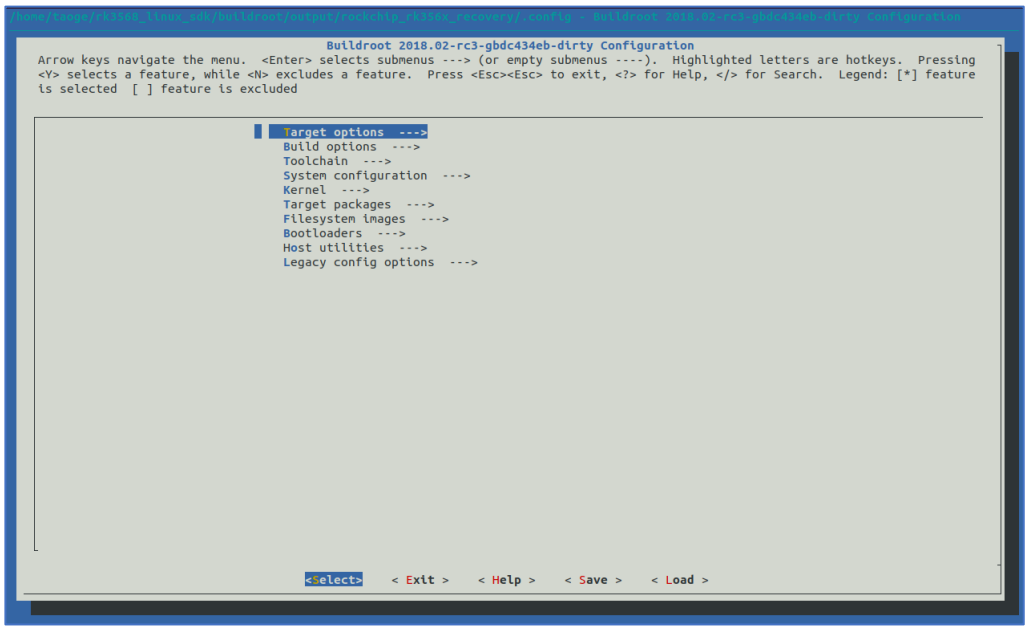


Figure 6.5-1 The menuconfig graphical configuration interface

Users can configure the ramdisk root file system by themselves (usually no configuration is required), and after the configuration is completed, the following command can be executed to save the configuration:

```
make savedefconfig
```

Return to the root directory of the SDK and run the "build.sh" script to compile the recovery (you can also compile it in the manner described in the subsections):

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

6.5.3 Reference Documents

Recovery-related reference documents are stored in the <SDK_PATH>/docs/cn/Linux/Recovery directory. Users can refer to them at their own discretion:

6.6 misc.img Image

The misc.img image will be burned onto 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 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 to enter recovery mode
boot-fastboot	Start and enter recovery mode via 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	Factory mode, mainly used for some equipment testing, such as PCBA testing
pcba_test	Enter PCBA testing
resize_partition	Re-plan the partition size, support for dynamic partitions in Android Q
rk_fwupdate	Specify rk SD/USB firmware upgrade, the scope is limited to U-Boot

In the Linux system, it is often used during system upgrades or when restoring factory settings. At the U-Boot stage, the BCB data in the misc partition is read. Based on the command content 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.