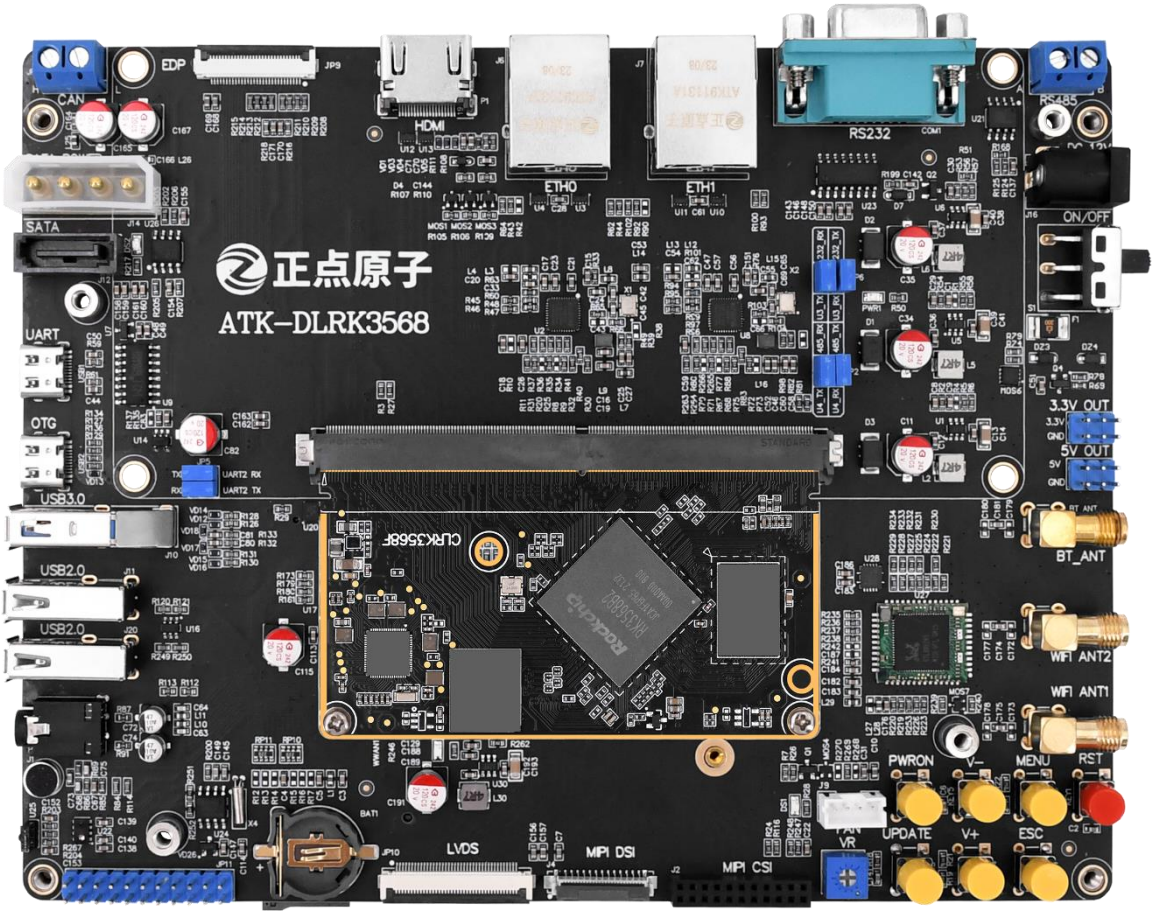
ATK-DLRK3568

Linux 系统开发手册

V1.0

-正点原子 ATK-DLRK3568 开发板文档



修订历史:

版本	日期	修改内容
V1.0	2023/06/20	第一次发布

正点原子公司名称 ：广州市星翼电子科技有限公司
原子哥在线教学平台：www.yuanzige.com
开源电子网 / 论坛 ：www.openedv.com
正点原子官方网站 ：www.alientek.com
正点原子淘宝店铺 ：https://openedv.taobao.com
正点原子 B 站视频 ：https://space.bilibili.com/394620890

电话：020-38271790 传真：020-36773971

请下载原子哥 APP，数千讲视频免费学习，更快更流畅。
请关注正点原子公众号，资料发布更新我们会通知。

扫码下载“原子哥”APP

扫码关注正点原子公众号

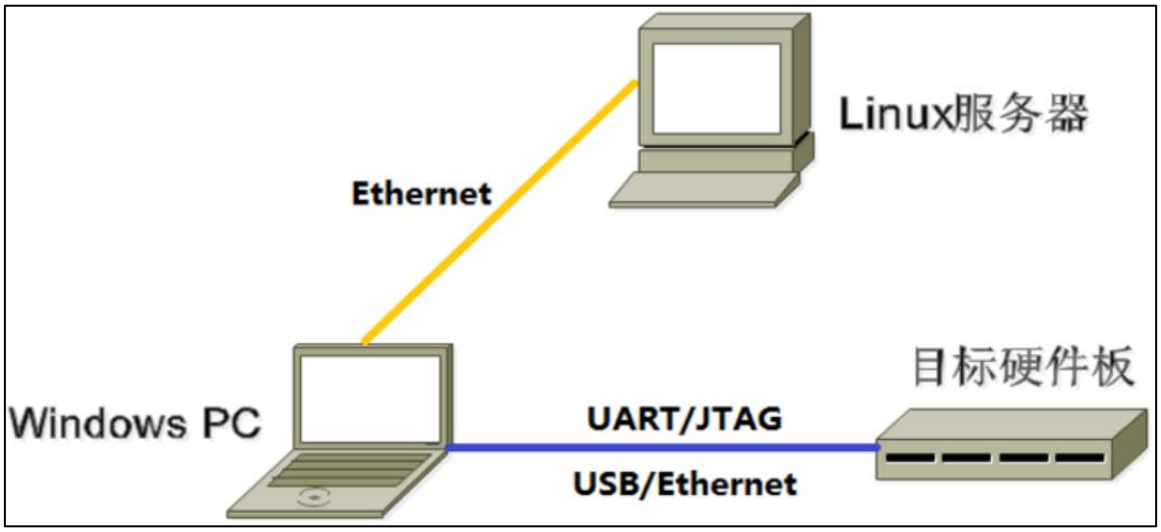
前言

本文档将向用户介绍 RK3568 平台嵌入式 Linux 系统开发相关内容，包括如下几个章节：

1， 安装 Ubuntu 操作系统，
2， 搭建开发环境，
3， 正点原子 RK3568 开发平台简介，
4， RK3568 Linux SDK 软件包，

嵌入式开发环境概述

一个典型的嵌入式开发环境通常包括 Linux 服务器、Windows PC 和目标硬件板，典型开发环境如下图所示：



- Linux 服务器上建立交叉编译环境，为软件开发提供代码更新下载、代码交叉编译服务。
- Windows PC 安装远程终端软件，譬如 Putty、SecureCRT 或 MobaXterm 等，通过网络远程登录（ssh 远程登录）到 Linux 服务器，进行交叉编译（将编译后生成的镜像文件通过网络拷贝到 Windows PC），及代码的开发调试。
- Windows PC 通过串口和 USB 与目标硬件板连接，可将编译后生成的镜像文件烧写到目标硬件板（一般都是通过 USB 或者 JTAG 烧录器进行烧录），并调试系统或应用程序。

◆ **虚拟机 Linux 系统+Windows PC+目标硬件板**

对于很多开发者来说，可能并没有 Linux 服务器或者公司并未提供专用于代码交叉编译的 Linux 服务器，你只有一台 Windows 电脑；那么这个时候可以在 Windows PC 上安装虚拟机软件（譬如 VMware、Virtual Box 或 Virtual PC），使用虚拟机软件在 Windows PC 上创建虚拟机，虚拟机安装 Linux 操作系统；使用虚拟机 Linux 系统来代替 Linux 服务器。

现在很多芯片平台都支持直接在 Linux 系统环境下烧写镜像文件至目标硬件设备，譬如 Rockchip 平台；使用虚拟机 Linux 系统会更加方便，因为虚拟机 Linux 系统与 Windows 系统本就处于同一台实体机上，共享实体机的硬件资源；目标硬件设备通过串口和 USB 连接到 PC，Windows 系统能检测到设备接入，同样虚拟机 Linux 系统也能检测到设备接入（通常可以选择设备连接到 Windows 系统还是虚拟机 Linux 系统），所以可以直接在虚拟机 Linux 系统环境下烧写镜像文件至目标硬件设备，而不用将镜像文件从 Linux 系统拷贝至 Windows 系统、再由 Windows 系统烧写镜像文件，极大方便开发！

第一章 安装 Ubuntu 系统

嵌入式 linux 开发一般都是基于 Windows+Ubuntu 双系统开发环境，本章向大家介绍如何在虚拟机上安装 Ubuntu 操作系统。

本章将分为如下几个小节：

1.1 安装 VMware 虚拟机软件

1.2 安装 Ubuntu 系统

1.1 安装 VMware 虚拟机软件

VMware（Vmware Workstation）是一款功能非常强大的虚拟机软件，通过该软件我们可以在 Windows 系统中创建一个或多个虚拟机。所谓虚拟机（Virtual Machine），它是指通过软件模拟出来的具有实体计算机完整硬件系统功能（譬如内存、硬盘、处理器等）的计算机系统。可以在虚拟机中安装实体机所支持的各种操作系统，譬如 Windows 操作系统、DOS、Linux 操作系统（Ubuntu、Debian、CentOS 等），在实体计算机中能够完成的工作在虚拟机中都能够实现，可以像使用实体机一样对虚拟机进行操作。

目前流行的虚拟机软件有 VMware(Vmware Workstation)、Virtual Box 和 Virtual PC 等，本文档将向用户介绍如何通过 VMware 软件创建虚拟机并安装 Ubuntu 操作系统。

1.1.1 下载 VMware 软件

开发板资料包中已经给大家提供了 VMware 虚拟机软件的安装包文件，路径为：**开发板光盘 A 盘-基础资料→04、软件→VMware-workstation-full-16.2.1-18811642.exe**，如果用户不想浪费时间下载，那么可以直接使用该安装包文件，进入下一步 1.1.2 小节。

用户也可以通过链接地址：https://www.vmware.com/products/workstation-pro/workstation-pro-evaluation.html，下载当前最新版本的 VMware Workstation 安装包文件：

图 1.1.1.1 VMware 虚拟机软件最新版本安装包下载

上述链接地址只能下载当前最新版本的安装包文件，如果用户需要下载旧版本的安装包文件，可以通过链接地址：https://customerconnect.vmware.com/cn/downloads/#all_products，进行下载：

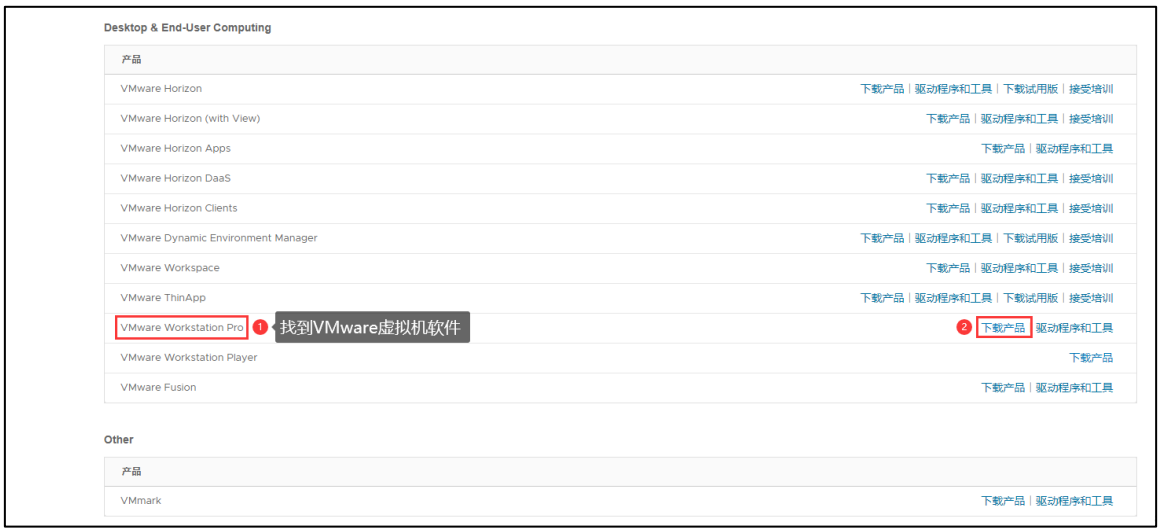


图 1.1.1.2 旧版本 VMware 虚拟机软件安装包下载 1

图 1.1.1.3 旧版本 VMware 虚拟机软件安装包下载 2

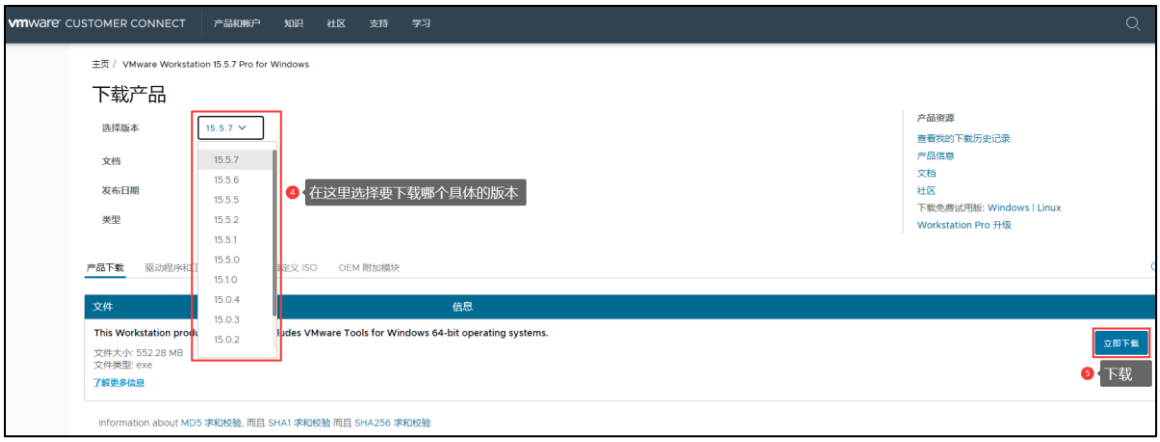


图 1.1.1.4 旧版本 VMware 虚拟机软件安装包下载 3

下载过程需要用户登录 VMware 账号，如果没有 VMware 账号、可以通过 VMware 官网 https://www.vmware.com/cn.html 进行注册。

对 VMware 虚拟机软件的版本没有什么要求，最新版本也行、旧的版本（譬如 15.X.X 或 14.X.X）也可以，本文档将以 16.2.1 版本为例，其对应的安装包文件名为 VMware-workstation-full-16.2.1-18811642.exe。

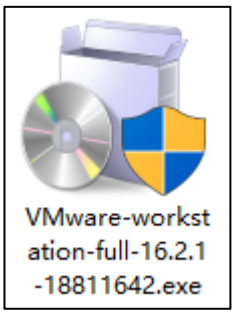


图 1.1.1.5 16.2.1 版本的 VMware 虚拟机软件安装包文件

1.1.2 安装 VMware 软件

Windows 下安装 VMware 软件非常简单，直接双击运行安装包文件 VMware-workstation-full-16.2.1-18811642.exe，按照图 1.1.2.1~1.1.2.8 所示操作步骤安装 VMware 虚拟机软件：

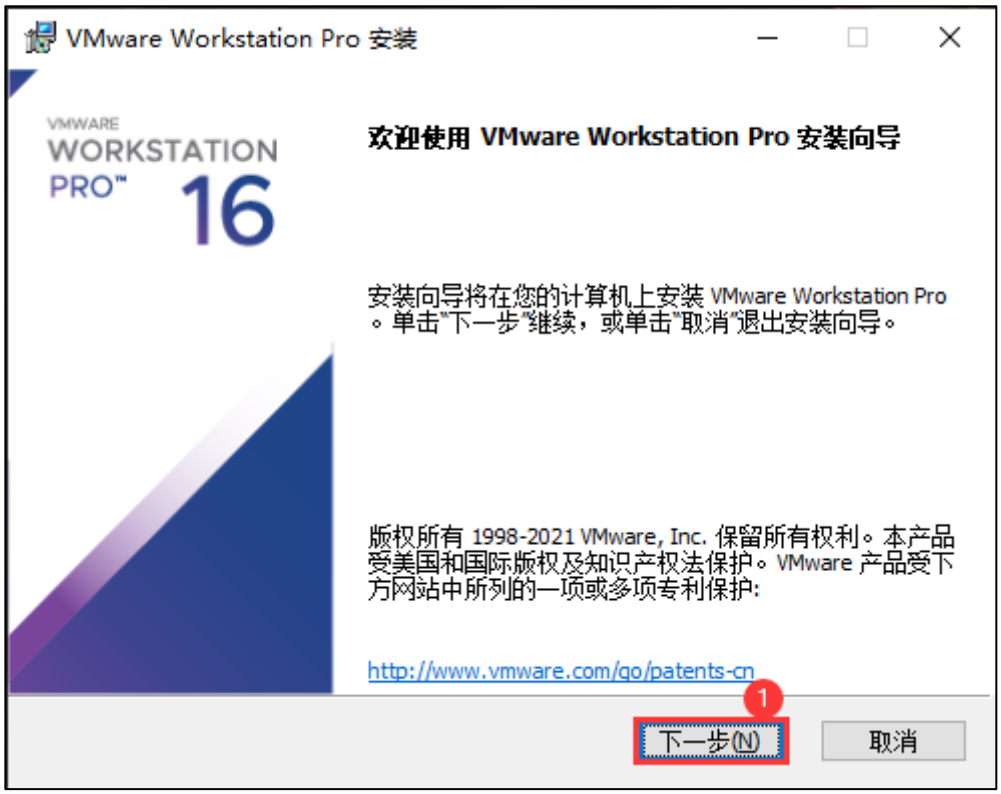


图 1.1.2.1 安装 VMware 软件 1

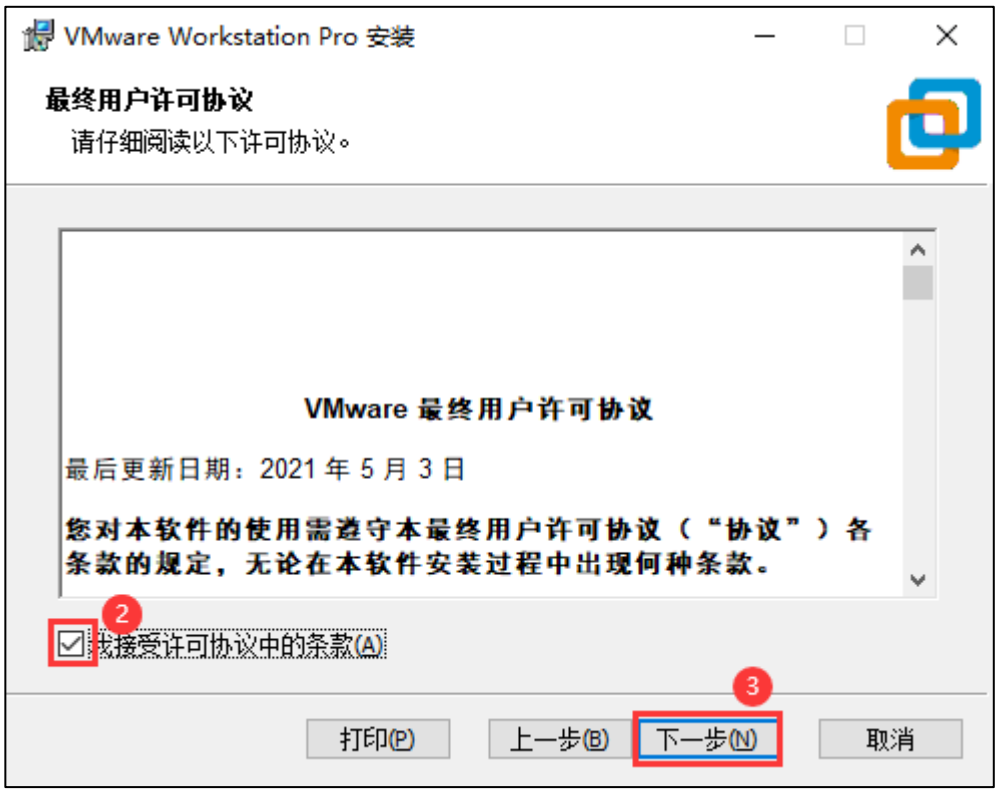


图 1.1.2.2 安装 VMware 软件 2

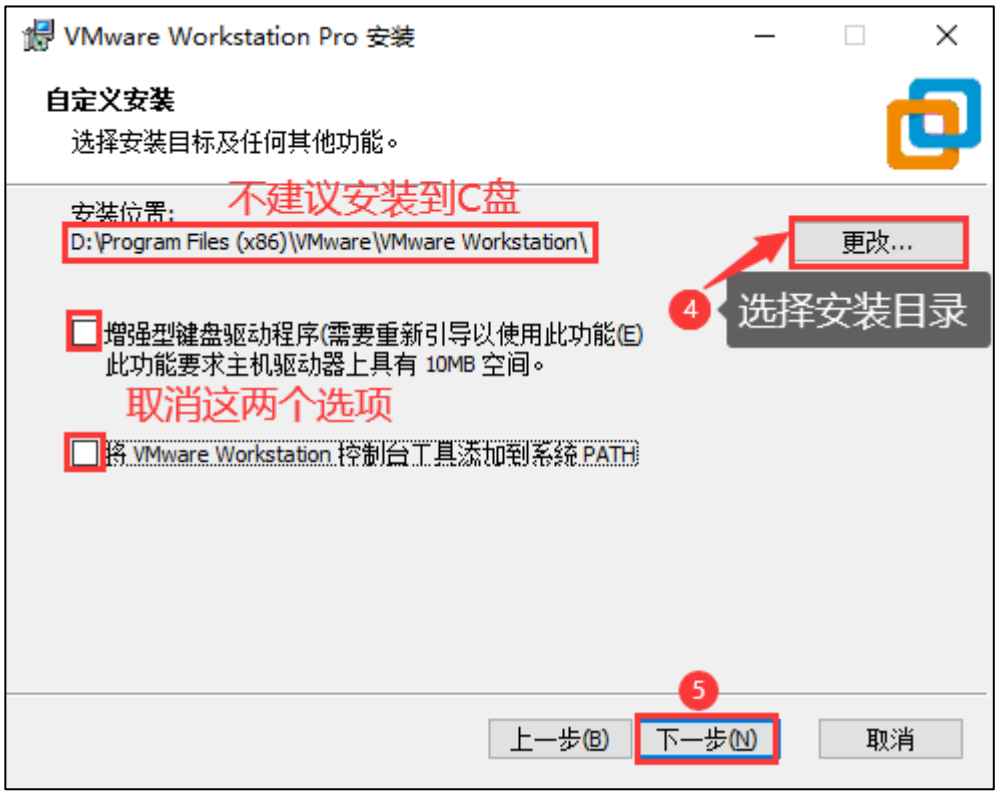


图 1.1.2.3 安装 VMware 软件 3

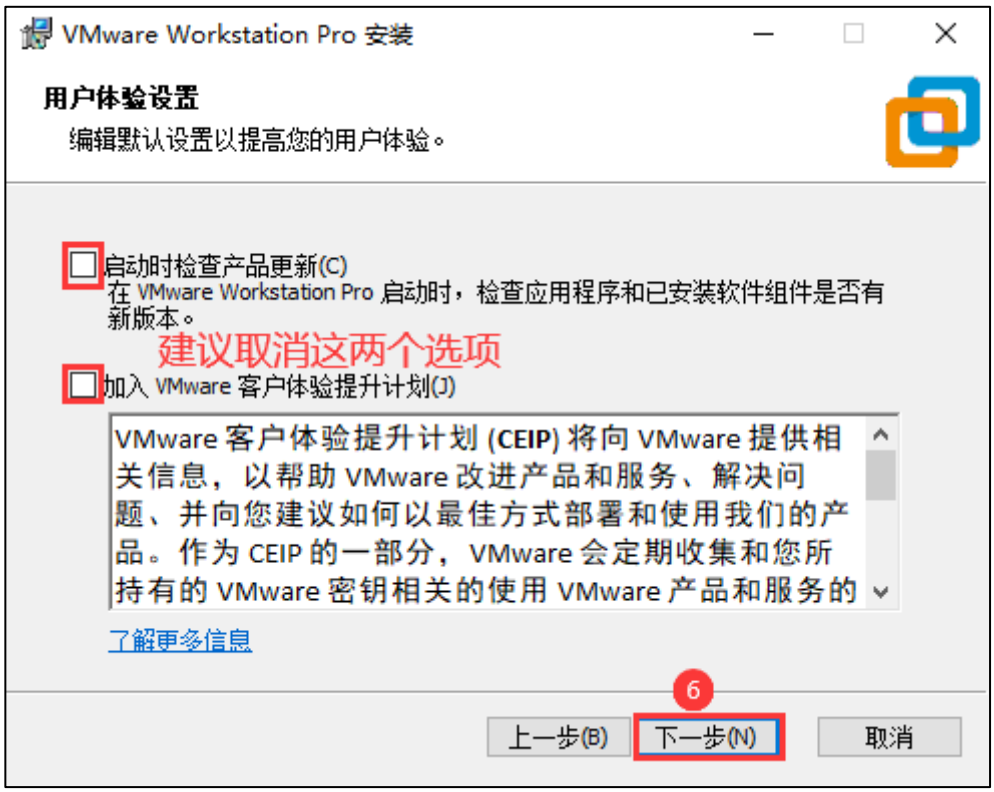


图 1.1.2.4 安装 VMware 软件 4

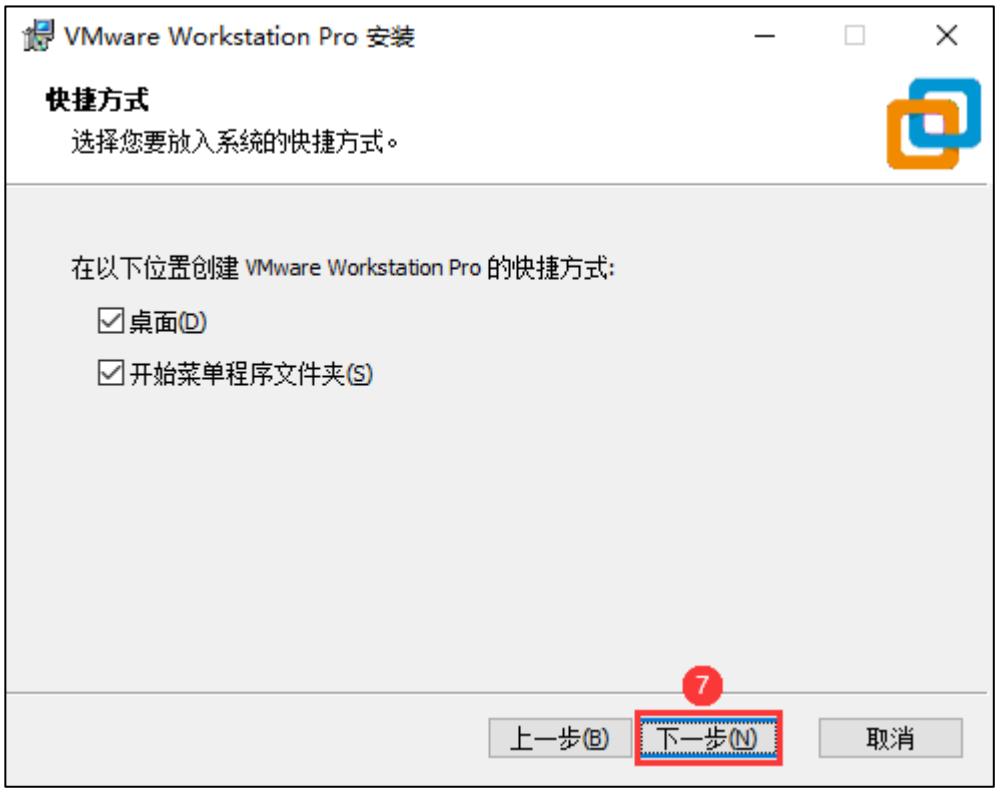


图 1.1.2.5 安装 VMware 软件 5

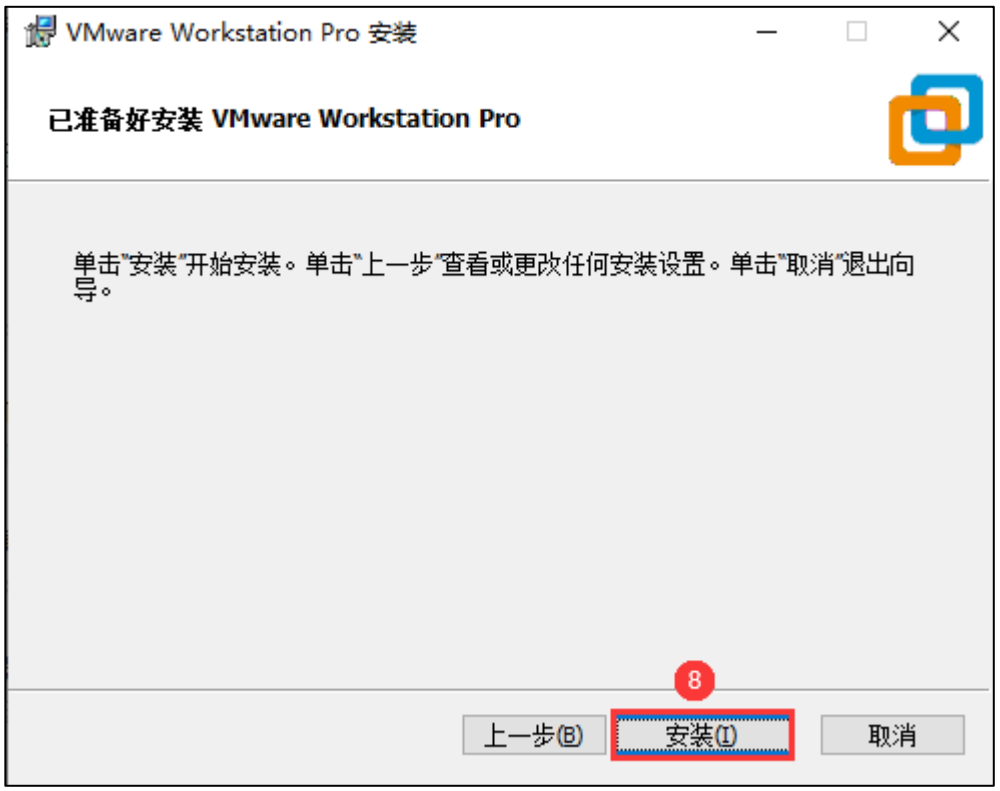


图 1.1.2.6 安装 VMware 软件 6

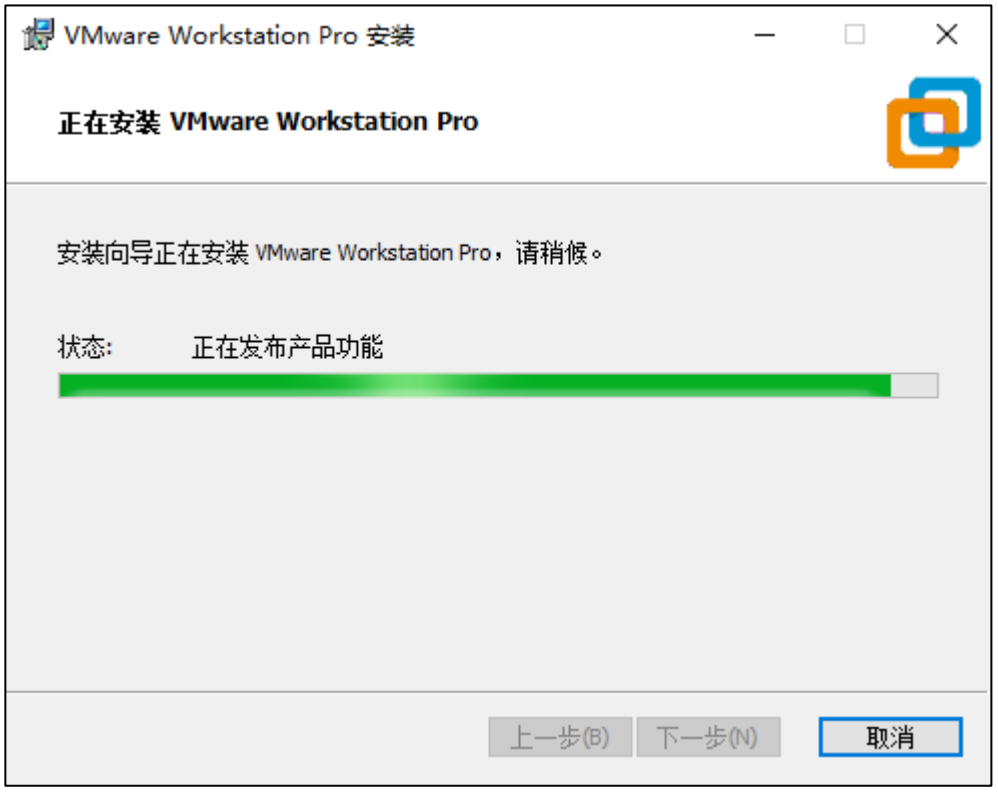


图 1.1.2.7 安装 VMware 软件 7

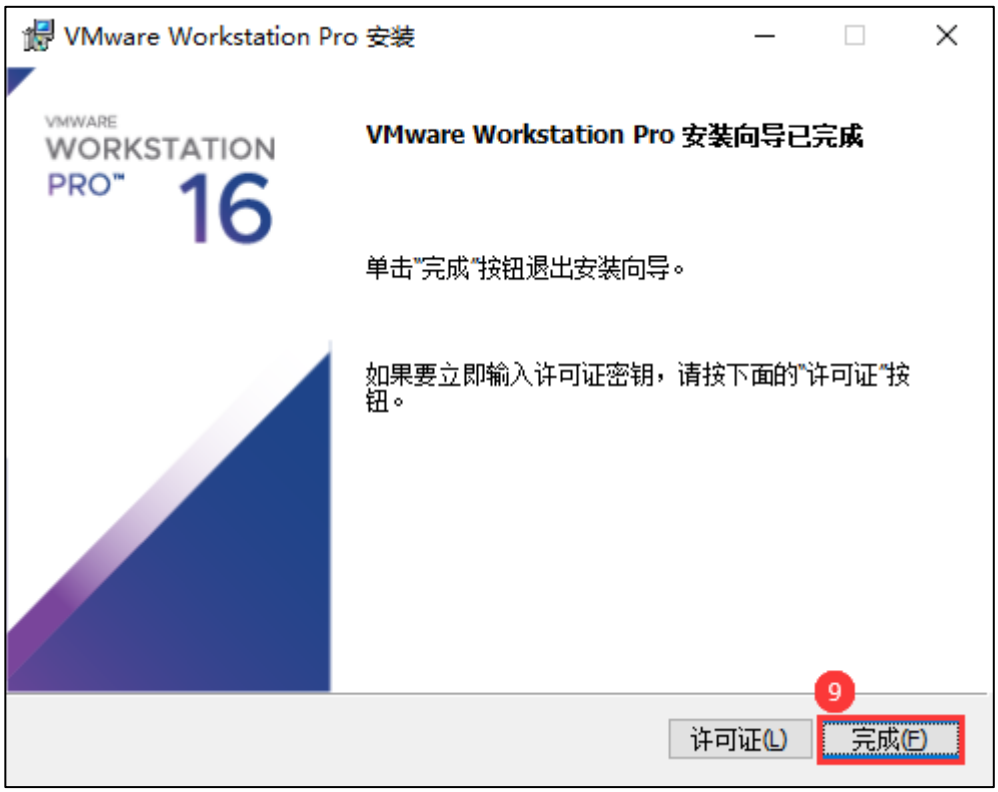


图 1.1.2.8 安装 VMware 软件 8

点击“完成”按钮，至此，VMware 虚拟机软件安装完成。

安装完成后，会在 Windows 系统桌面生成 VMware 虚拟机软件快捷方式图标，如图 1.1.2.9 所示：

图 1.1.2.9 VMware 软件桌面快捷方式图标

双击图标打开 VMware 虚拟机软件，首次打开软件会提示用户输入许可证密钥，如图 1.1.2.10 所示：

图 1.1.2.10 输入许可证密钥

VMware 虚拟机软件是付费软件，需要用户购买才能使用；如果用户购买了 VMware 软件使用许可，那么将会得到一串许可证密钥，将许可证密钥填写至“**我有 VMware Workstation 16 的许可证密钥(H)**”所对应的输入框中进行激活，激活成功后便可以正常使用 VMware 软件了。

如果用户没有购买软件使用许可，那么可以选择“**我希望使用 VMware Workstation 16 30 天(W)**”选项，这样便可获得 VMware 软件的 30 天试用期，点击“继续”按钮：

图 1.1.2.11 欢迎使用 VMware 软件

点击“完成”按钮，接着进入到 VMware 虚拟机软件主界面，如图 1.1.2.12 所示：

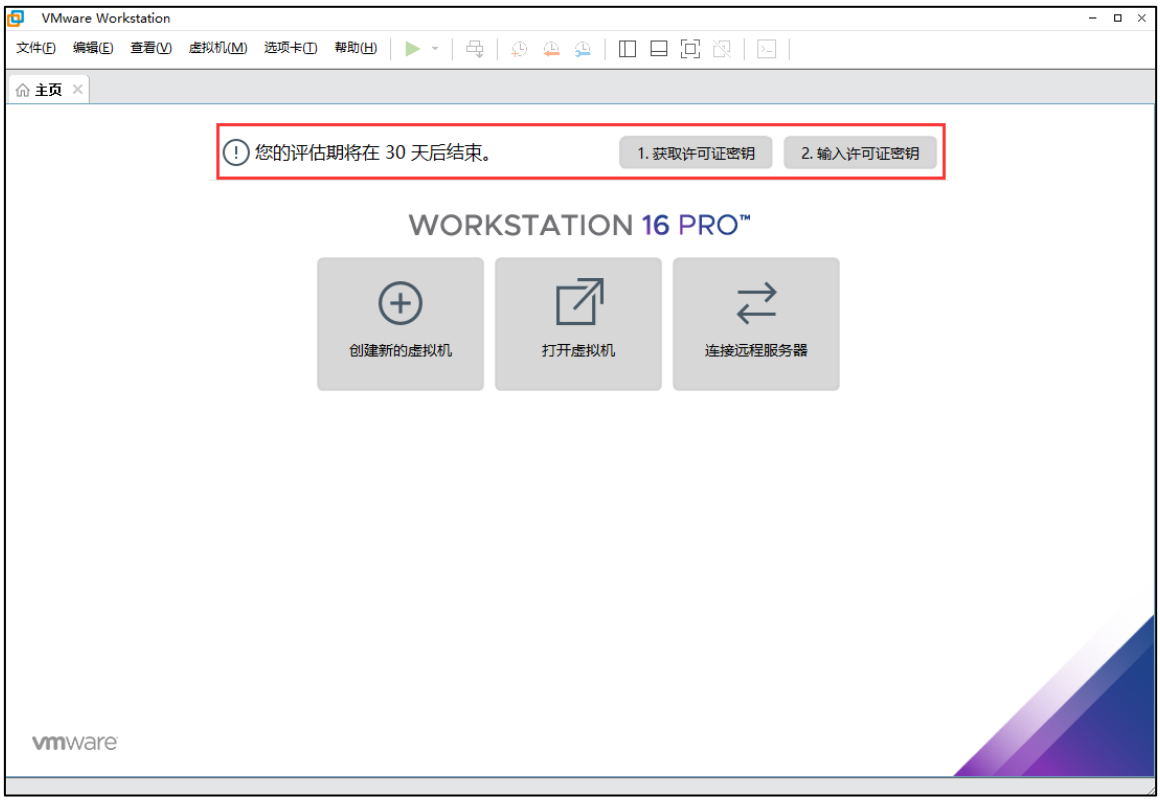


图 1.1.2.12 VMware 虚拟机软件主界面

1.2 安装 Ubuntu 系统

本小节向用户介绍如何在 VMware 虚拟机中安装 Ubuntu 操作系统。推荐用户使用 Ubuntu 20.04（**强烈建议大家使用 20.04 版本，我们提供的文档、视频，包括开发人员在开发过程中所使用的都是 Ubuntu 20.04 版本**）或者是 Ubuntu 18.04（**不推荐!**），本文档将以 Ubuntu 20.04 版本为例进行介绍，对于使用 Ubuntu 18.04 的用户，可对比参考本文档。

1.2.1 下载 Ubuntu 系统镜像

开发板资料包中已经给用户提供了 Ubuntu 18.04 以及 Ubuntu 20.04 系统镜像文件，路径为：**开发板光盘 A 盘-基础资料→04、软件→ubuntu-18.04.6-desktop-amd64.iso** 和**开发板光盘 A 盘-基础资料→04、软件→ubuntu-20.04.2-desktop-amd64.iso**。如果用户不想浪费时间下载，可以直接使用资料包中提供的 Ubuntu 系统镜像，进入下一步 1.2.2 小节。

用户也可以通过链接地址：https://old-releases.ubuntu.com/releases/，下载 Ubuntu 系统镜像（以 Ubuntu 20.04.2 版本为例）：

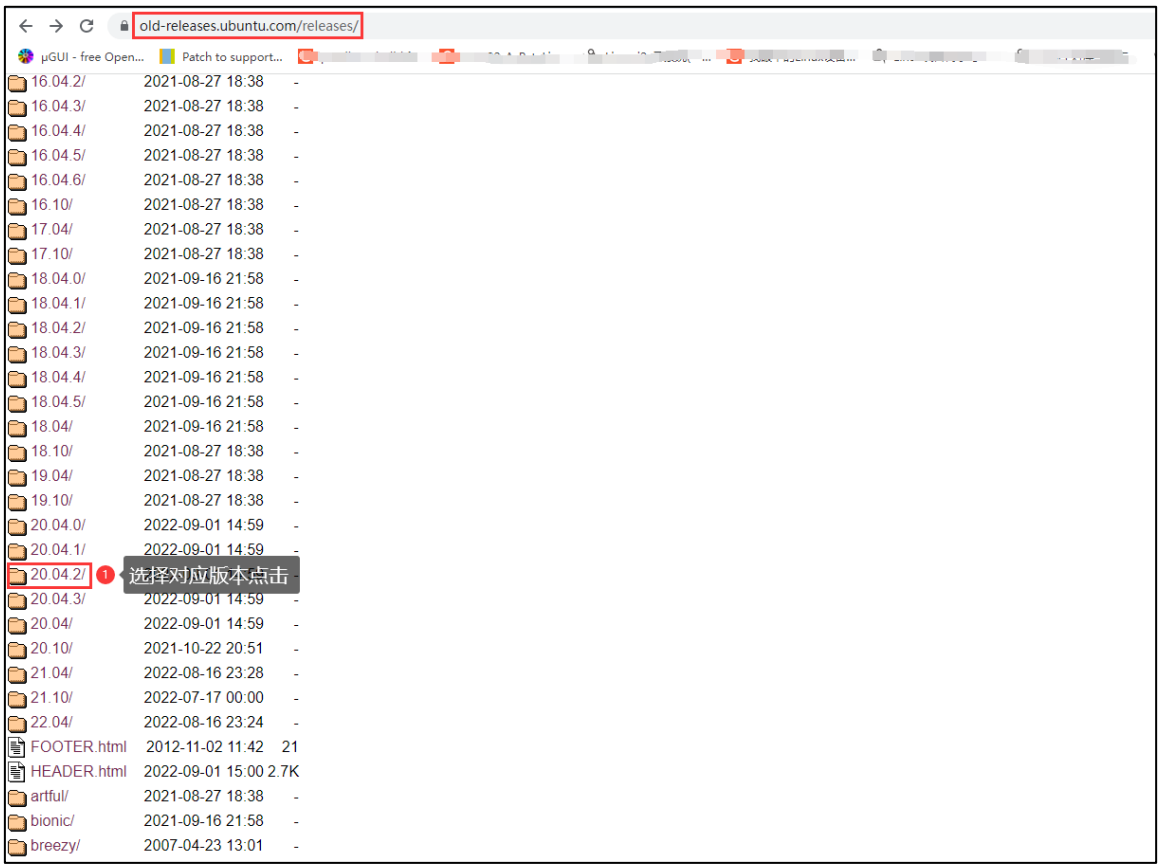


图 1.2.1.1 下载 Ubuntu 系统镜像 1

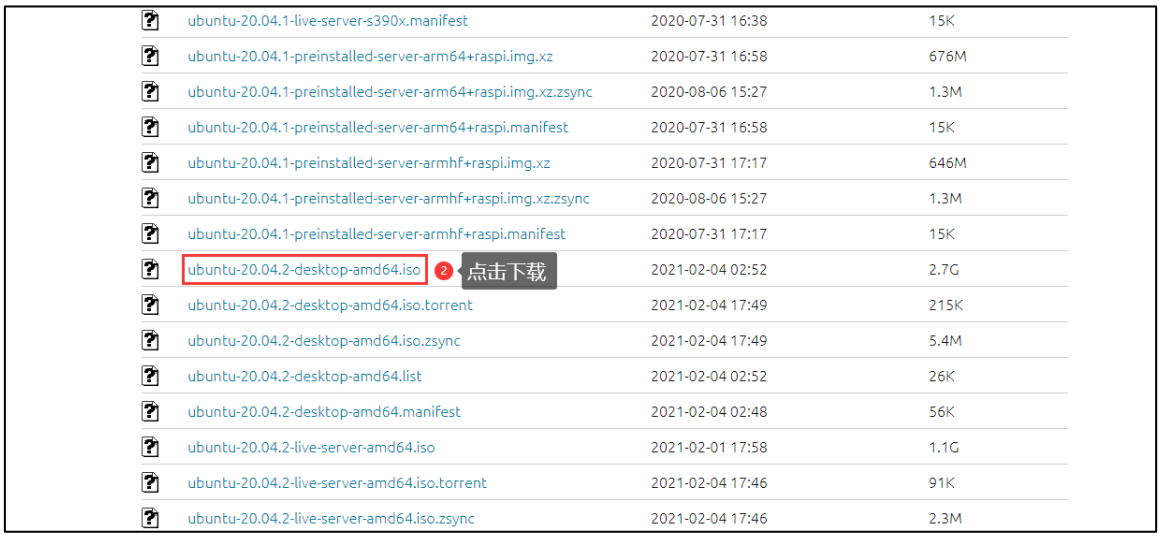


图 1.2.1.2 下载 Ubuntu 系统镜像 2

下载完成之后便会得到 Ubuntu 20.04.2 系统镜像文件 ubuntu-20.04.2-desktop-amd64.iso：

图 1.2.1.3 Ubuntu 20.04.2 系统镜像文件

1.2.2 创建虚拟机

打开 VMware 虚拟机软件，如图所示：

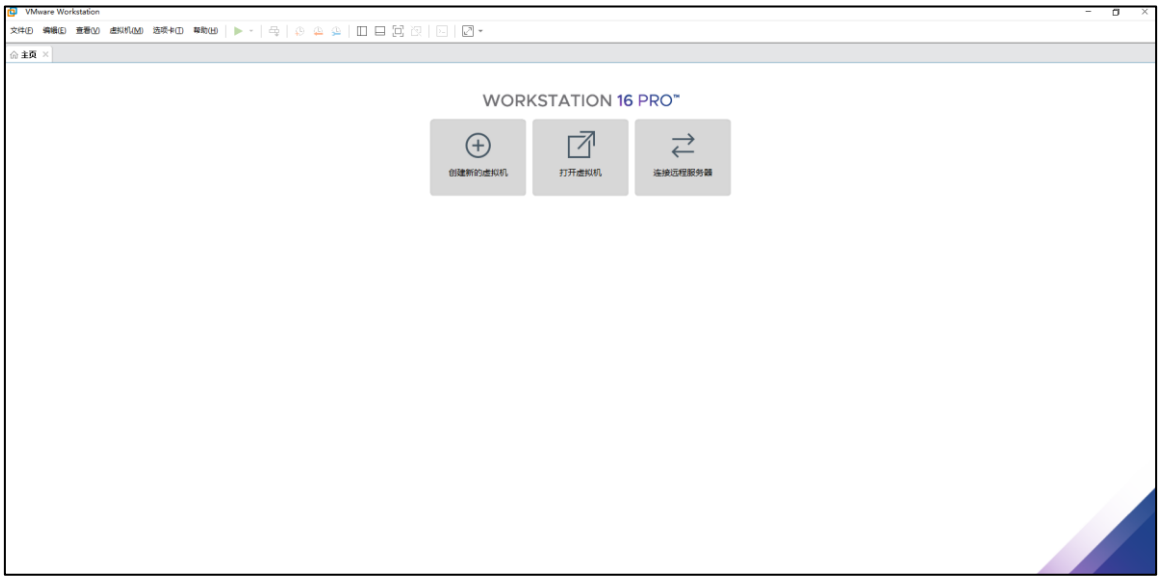


图 1.2.2.1 VMware 虚拟机软件

然后按照图 1.2.2.2~1.2.2.17 所示操作步骤创建一个新的虚拟机：

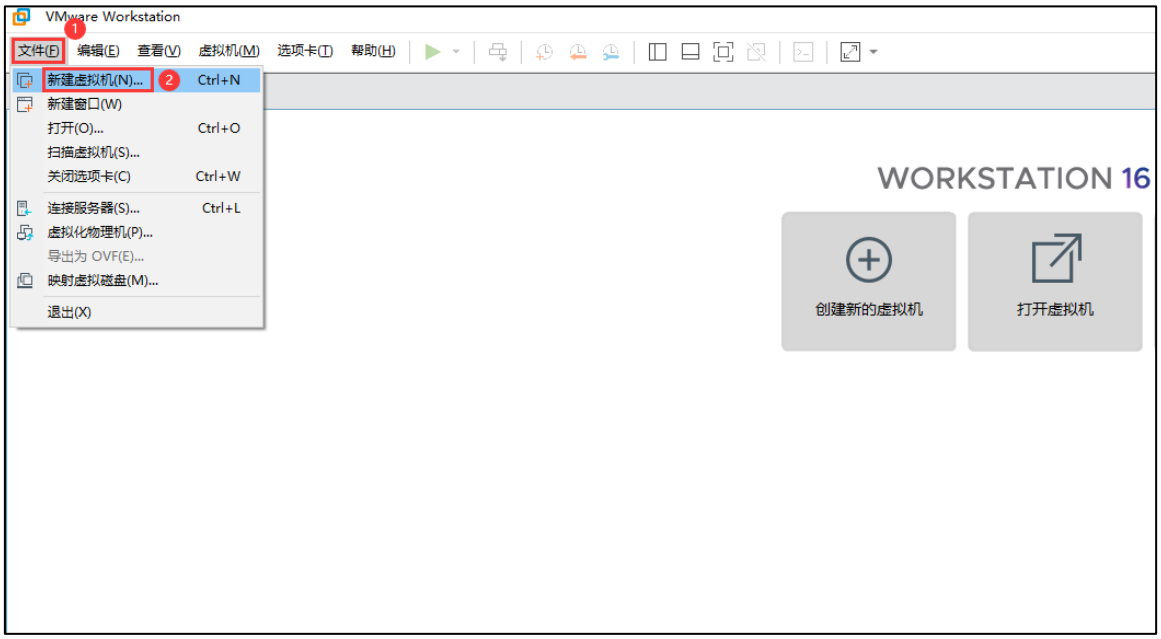


图 1.2.2.2 创建虚拟机(1)

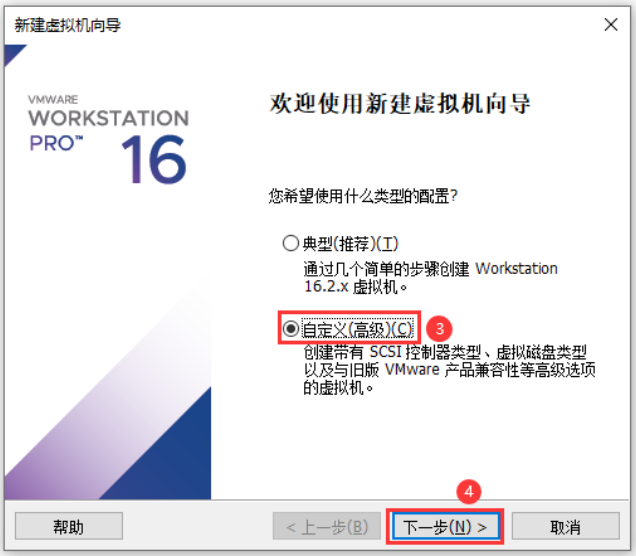


图 1.2.2.3 创建虚拟机(2)

图 1.2.2.4 创建虚拟机(3)

图 1.2.2.5 创建虚拟机(4)

图 1.2.2.6 创建虚拟机(5)

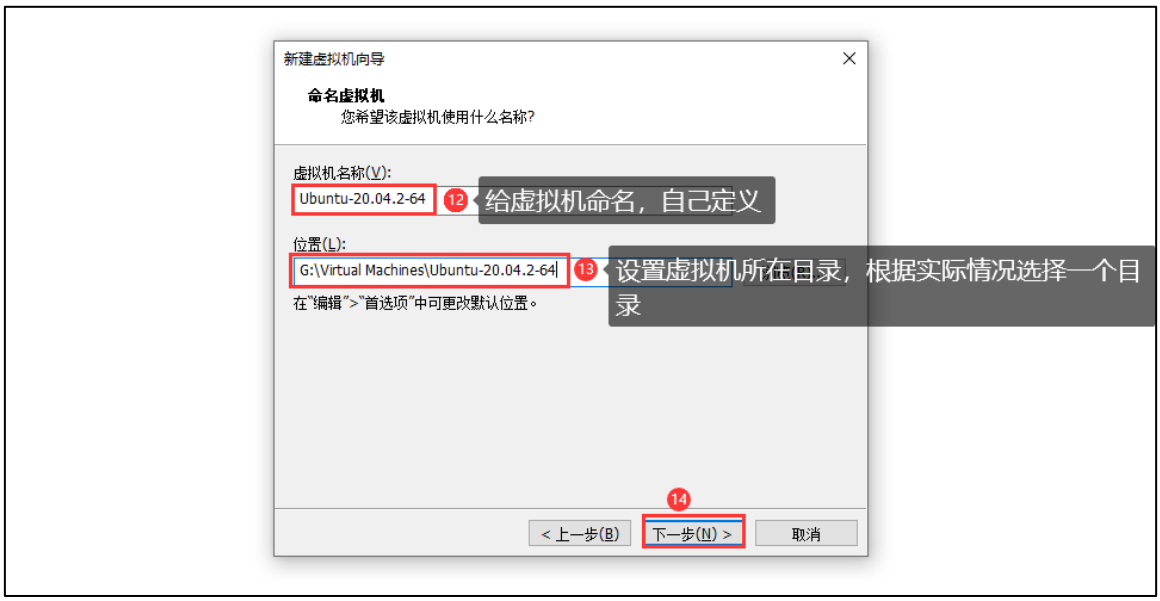


图 1.2.2.7 创建虚拟机(6)

图 1.2.2.8 创建虚拟机(7)

为虚拟机配置处理器数量以及每个处理器的内核数量：

处理器内核总数 = 处理器数量 * 每个处理器的内核数量；

用户需根据自己的电脑配置（CPU 配置）情况来为虚拟机分配处理器数量以及每个处理器的内核数量，只需满足如下要求：

虚拟机处理器内核总数小于或等于（<=）CPU 实际线程数；

可通过 Windows 任务管理器来查询电脑的 CPU 配置情况，如图 1.2.2.9 所示：

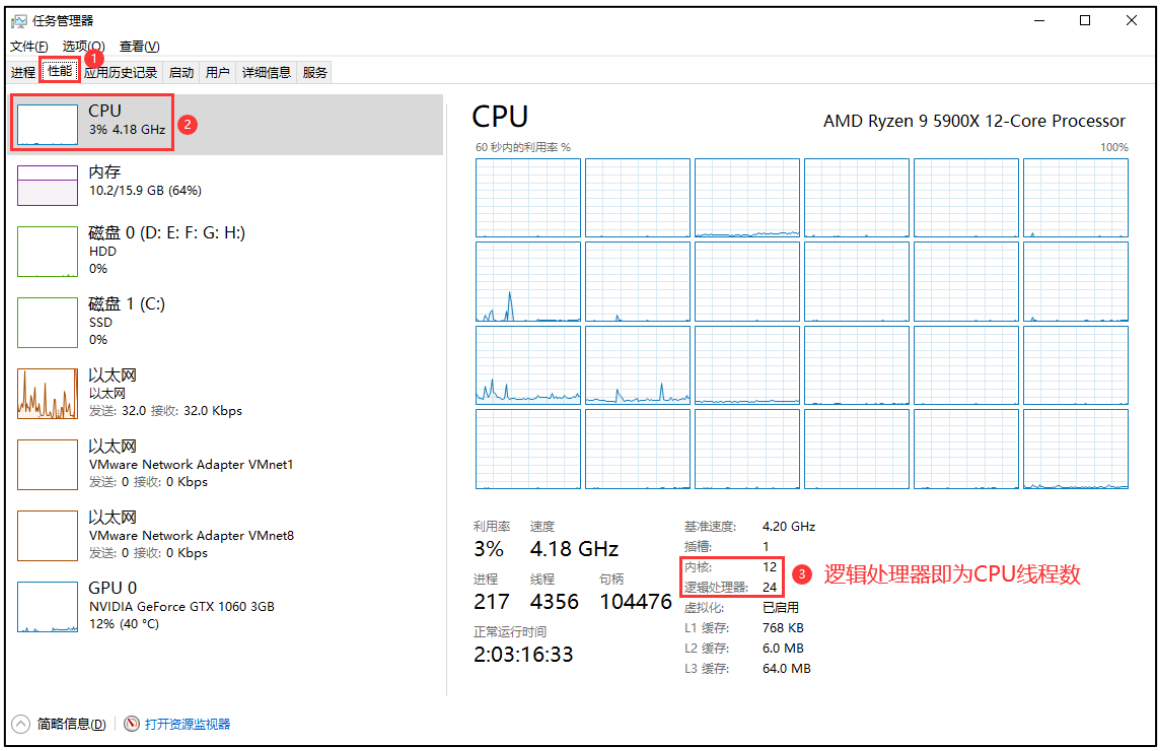


图 1.2.2.9 查询 CPU 配置情况

图中逻辑处理器数量即为 CPU 实际线程数，所以我的电脑 CPU 实际线程数为 24，按照配置要求，虚拟机的处理器内核总数需小于或等于（<=）CPU 实际线程数 24，那么我们可以有多种不同的配置方式，譬如：

处理器数量=1、每个处理器的内核数量=12；
处理器数量=1、每个处理器的内核数量=16；
处理器数量=1、每个处理器的内核数量=18；
处理器数量=4、每个处理器的内核数量=4；
处理器数量=2、每个处理器的内核数量=6；
处理器数量=6、每个处理器的内核数量=3；
处理器数量=8、每个处理器的内核数量=2；
等等……

总之具体的配置方式多种多样，只需满足要求即可！

当然，在条件允许的情况下（譬如不影响实体机 Windows 系统的正常使用），尽可能使得虚拟机的处理器内核总数稍多一些，这样我们在虚拟机 Ubuntu 系统下编译源码时可以开启更多的线程进行编译，自然编译速度也会更快！

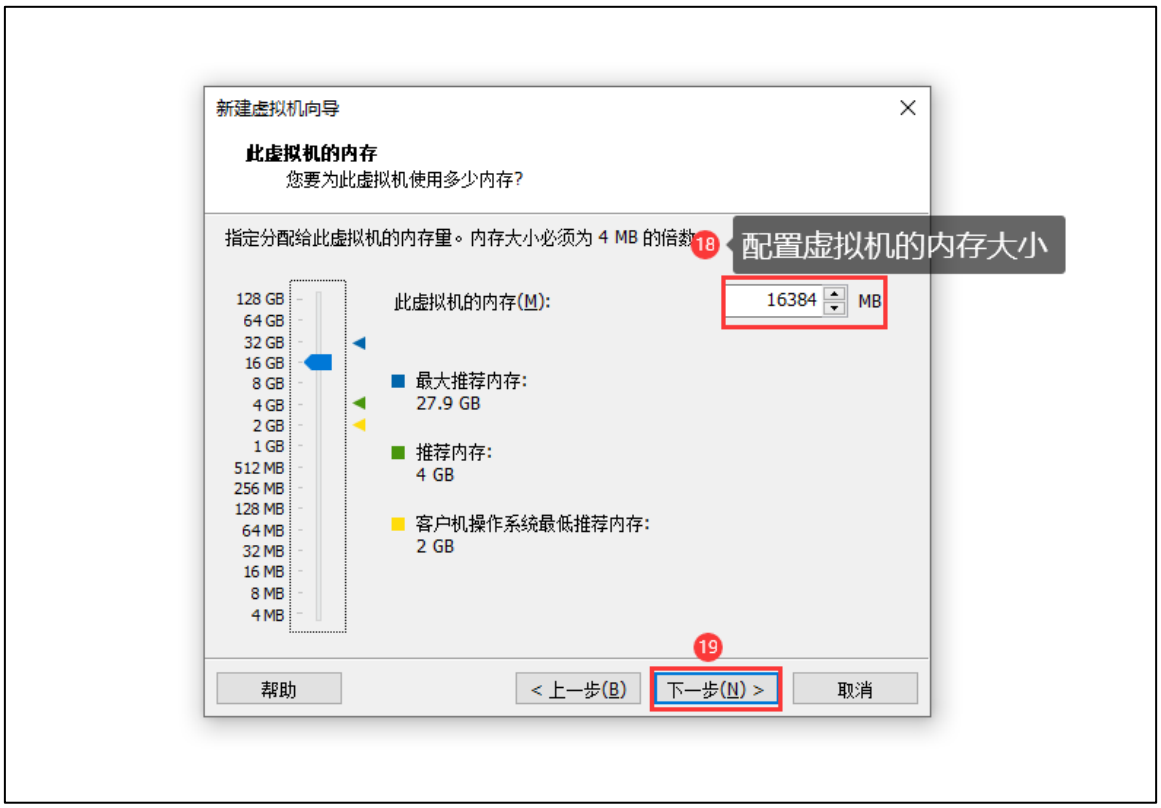


图 1.2.2.10 创建虚拟机(8)

可以通过上下拖动左边的蓝色滑块来调整虚拟机的内存大小，也可以直接在右边的输入框中填写数字来指定虚拟机的内存大小，单位是 MB。同样，虚拟机的内存大小也需要根据用户的电脑配置（内存配置）情况来设置，比如我的电脑内存大小为 32GB，因此我可以将 16GB 内存分配给虚拟机使用。

当然，分配给虚拟机的内存容量不能太小，如果内存太小，可能会导致虚拟机 Ubuntu 系统下编译源码时报错（由于内存不足而出错）；对于 RK3568 平台嵌入式 linux 系统开发来说，建议分配给虚拟机的内存量至少大于或等于 16GB、甚至更大。

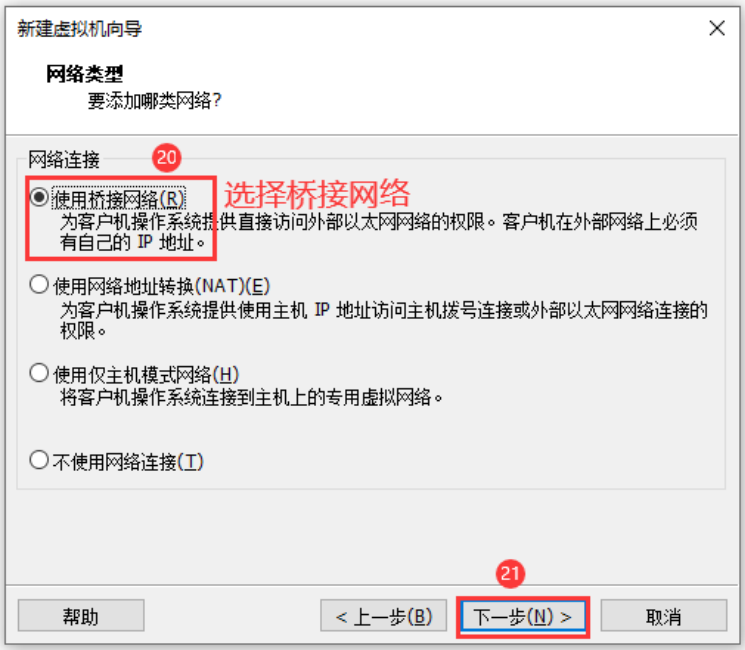


图 1.2.2.11 创建虚拟机(9)

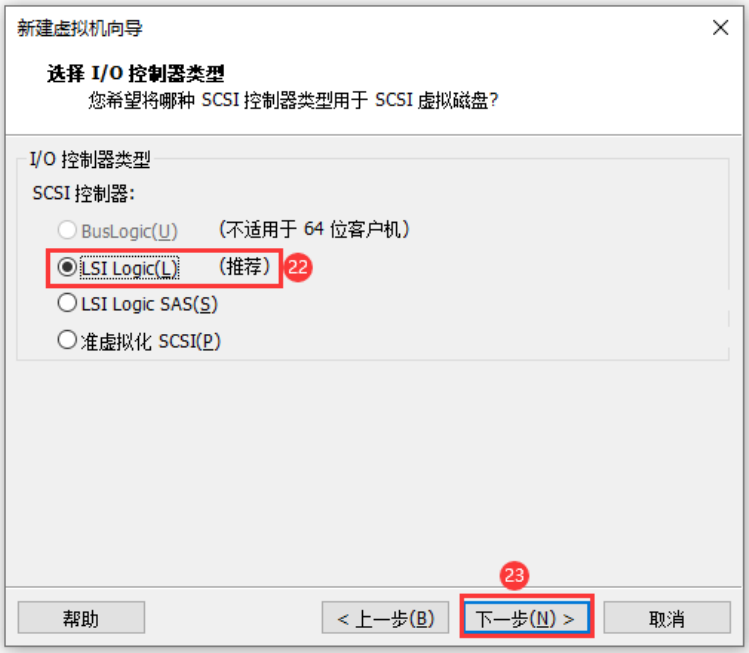


图 1.2.2.12 创建虚拟机(10)

图 1.2.2.13 创建虚拟机(11)

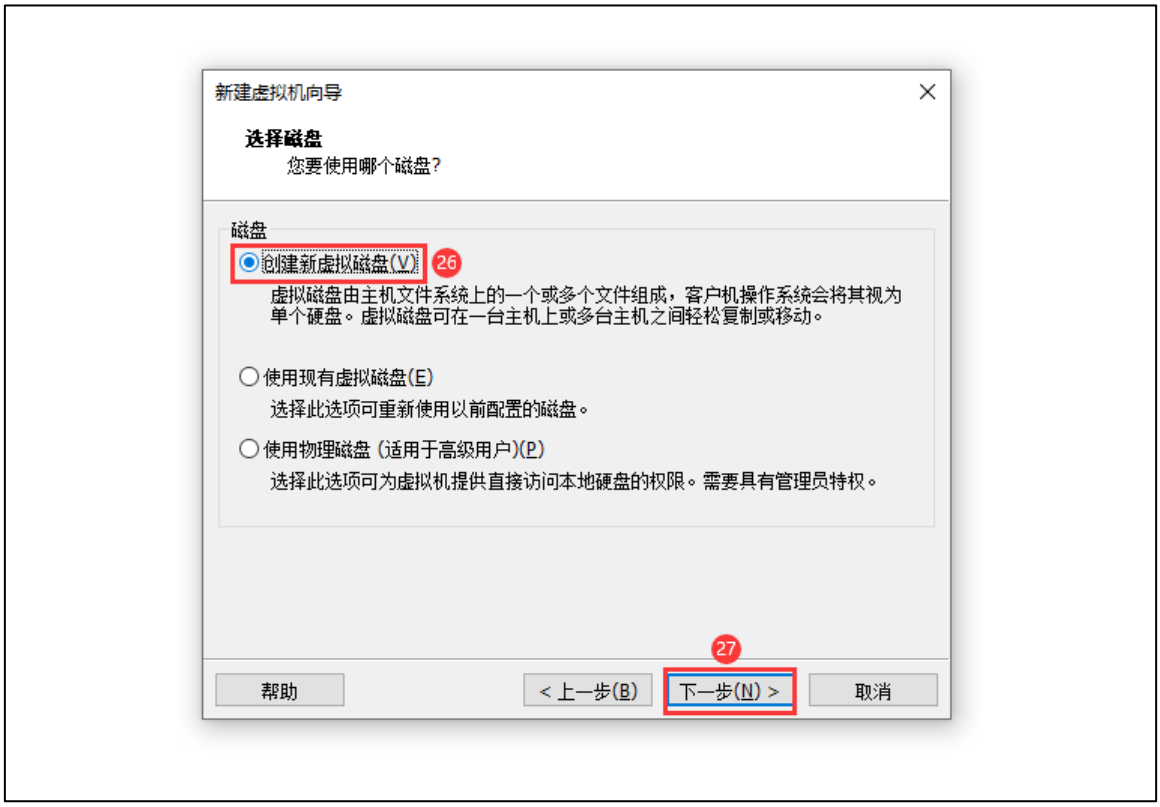


图 1.2.2.14 创建虚拟机(12)

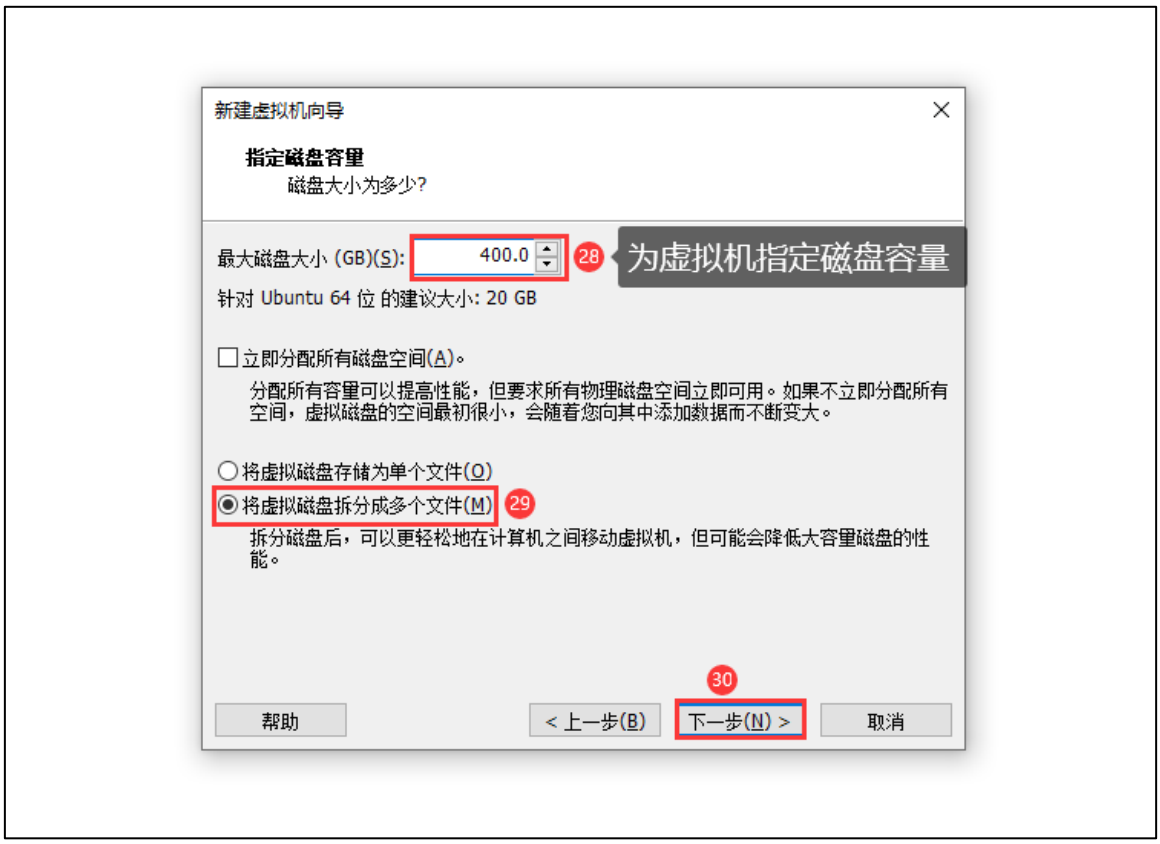


图 1.2.2.15 创建虚拟机(13)

同样，虚拟机磁盘容量也需要根据用户的电脑配置（磁盘配置）情况来设置，建议至少 300GB。

图 1.2.2.16 创建虚拟机(14)

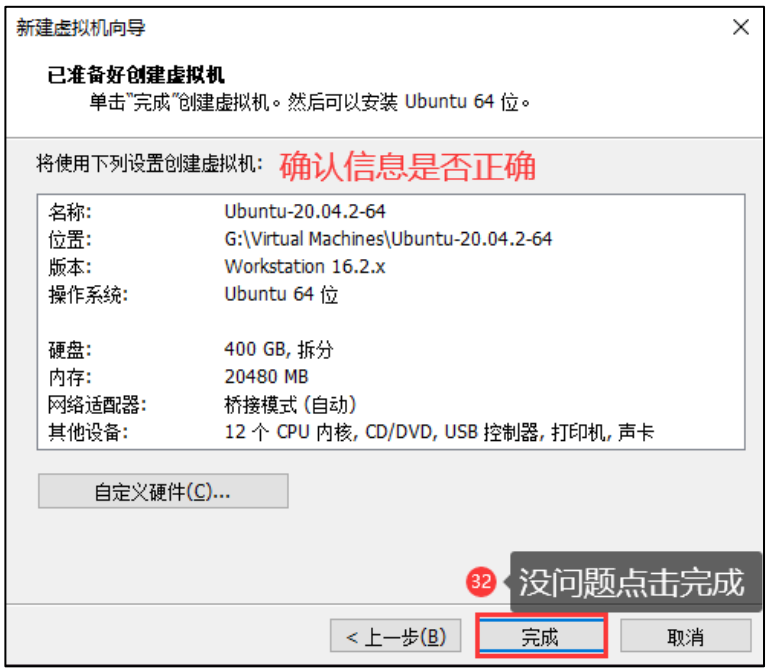


图 1.2.2.17 创建虚拟机(15)

至此，虚拟机便创建完成，如图 1.2.2.18 所示：

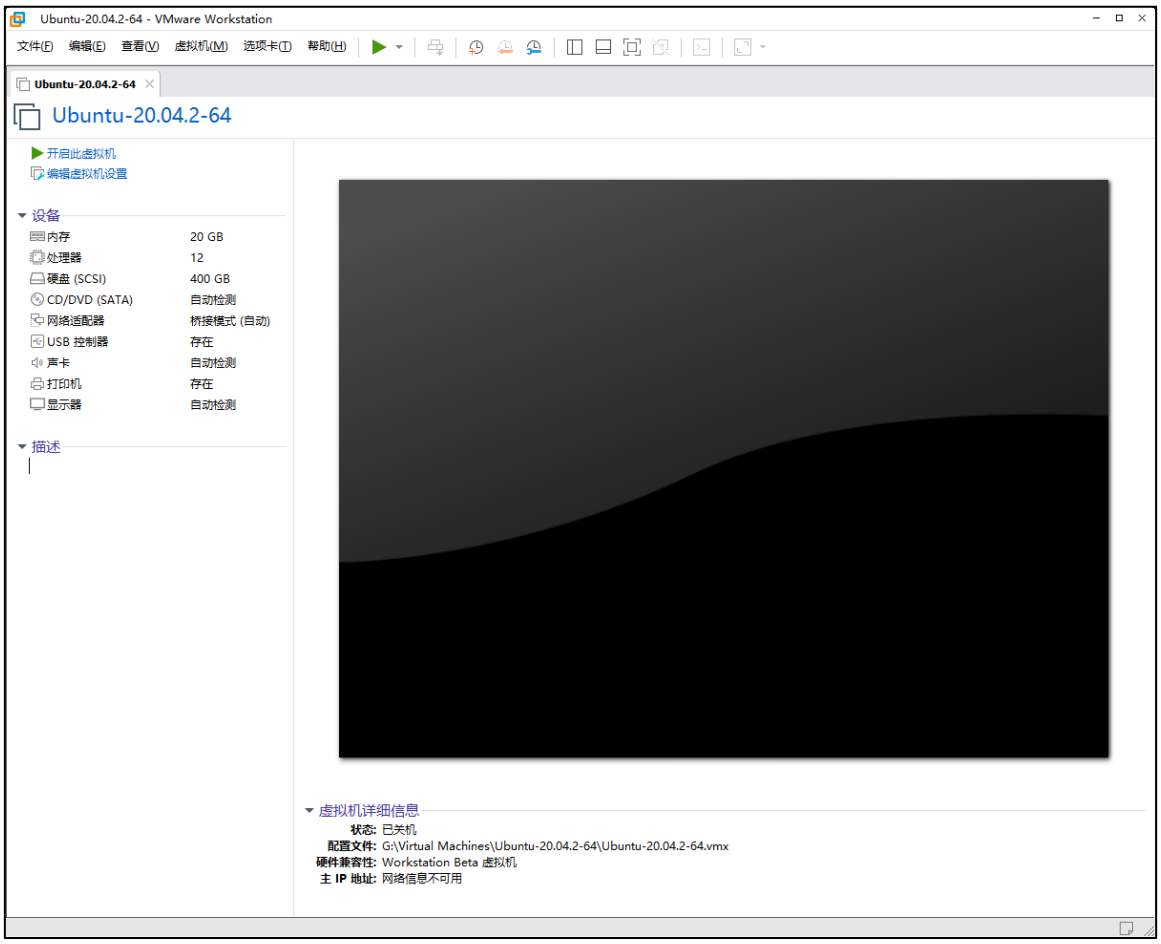


图 1.2.2.18 虚拟机创建完成

1.2.3 安装 Ubuntu 系统

接下来我们在虚拟机上安装 Ubuntu 20.04 操作系统，在创建好的虚拟机上点击“**虚拟机(M)→设置**”打开虚拟机设置界面：

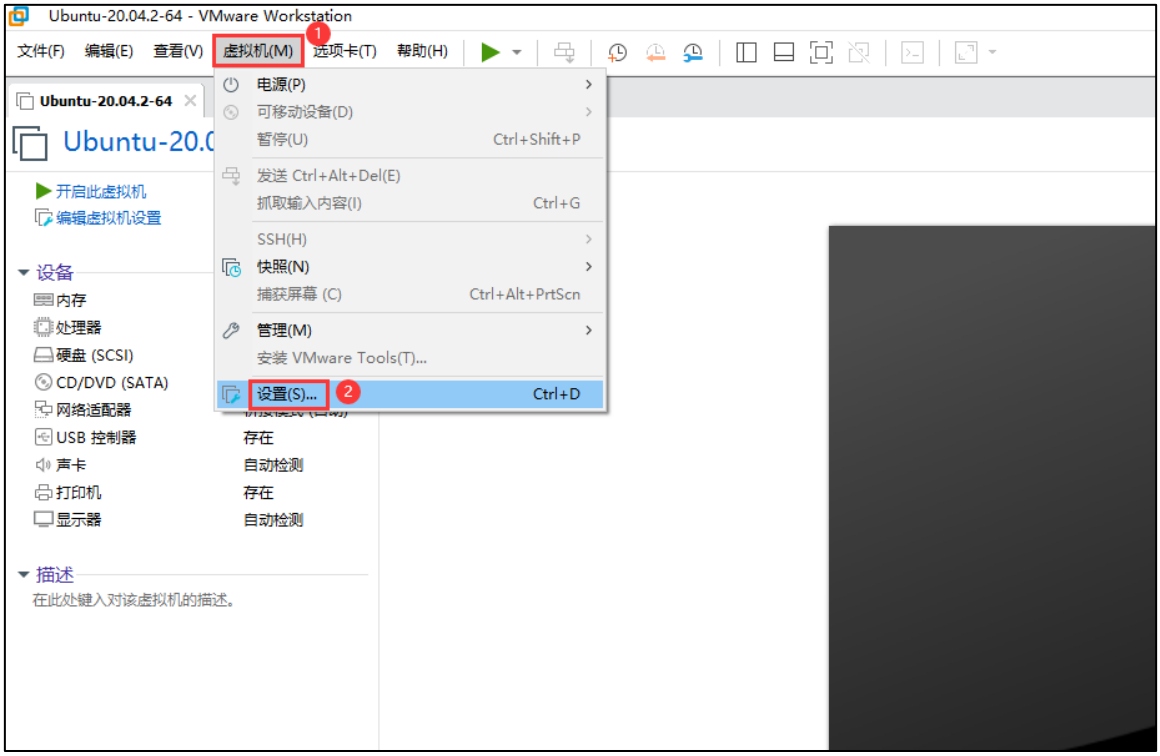


图 1.2.3.1 打开虚拟机设置界面

首先指定 Ubuntu 20.04 系统镜像文件 ubuntu-20.04.2-desktop-amd64.iso 所在路径：

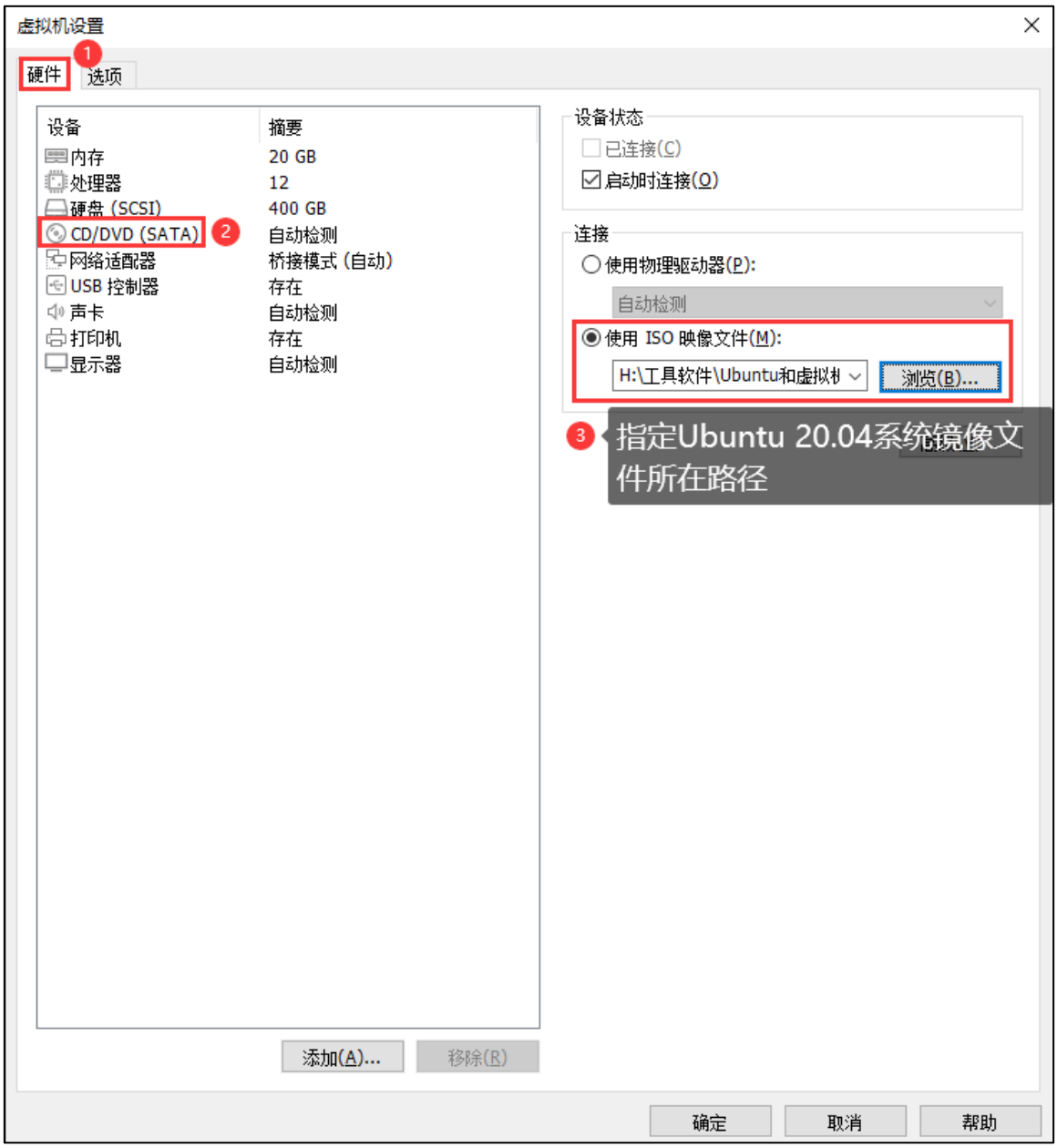


图 1.2.3.2 指定 Ubuntu 系统镜像

点击“浏览(B)”按钮，选择 Ubuntu 20.04 系统镜像文件 ubuntu-20.04.2-desktop-amd64.iso。接下来对虚拟机 USB 控制器进行设置：

图 1.2.3.3 设置 USB 控制器兼容性

USB 控制器的 USB 兼容性默认为 USB 2.0，当我们使用 USB 3.0 设备时，Ubuntu 系统可能识别不出来，因此我们需要调整 USB 兼容性为 USB 3.1，最后点击“确定”按钮退出窗口。

接下来开始安装 Ubuntu 操作系统，按照图 1.2.3.4~1.2.3.13 所示步骤安装 Ubuntu 系统：

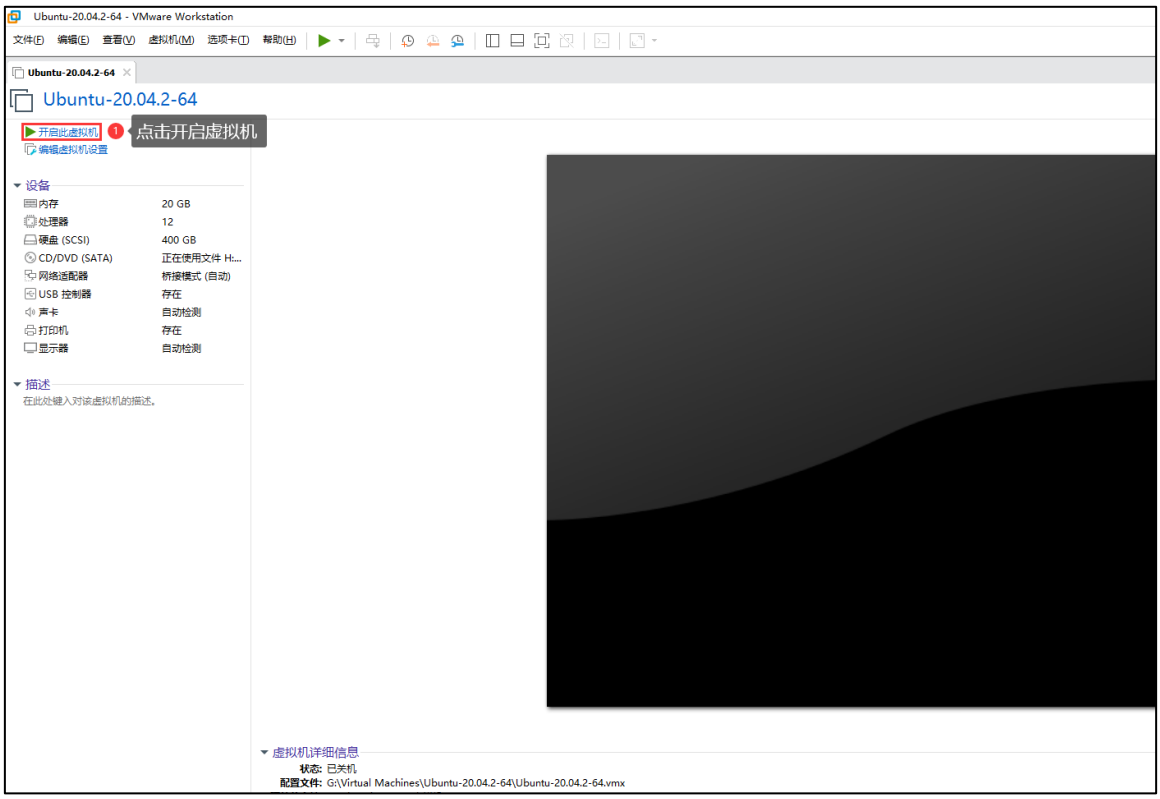


图 1.2.3.4 安装 Ubuntu 操作系统(1)

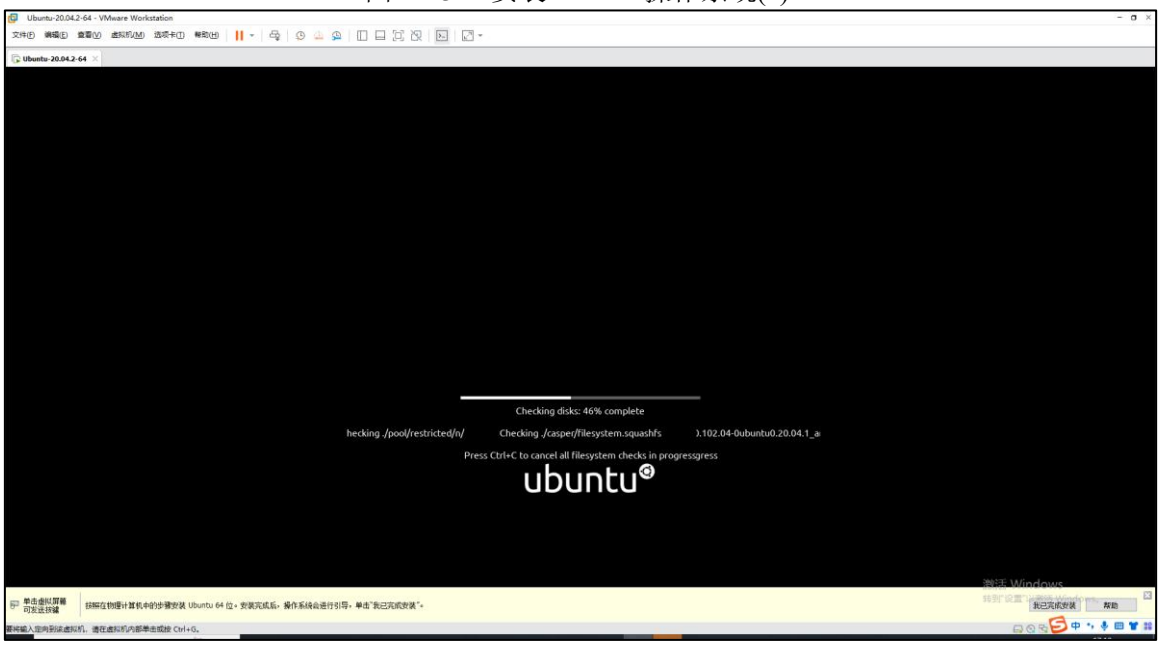


图 1.2.3.5 安装 Ubuntu 操作系统(2)

图 1.2.3.6 安装 Ubuntu 操作系统(3)

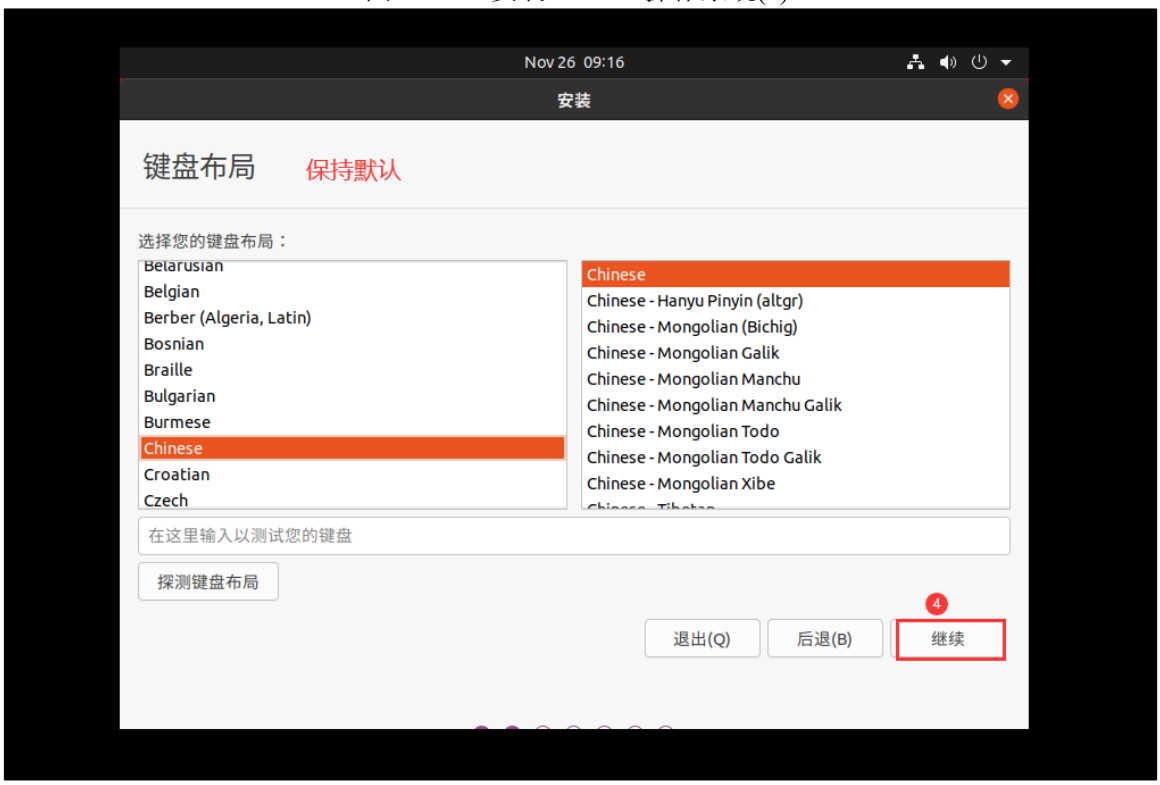


图 1.2.3.7 安装 Ubuntu 操作系统(4)

如果用户在安装过程中，由于窗口显示的问题，无法看到下面的三个按钮（退出、后退以及继续这三个按钮），导致无法点击按钮；这个时候可以按住 Win 键（Ctrl 键和 Alt 键中间的那个按键），然后使用鼠标左键拖动窗口。

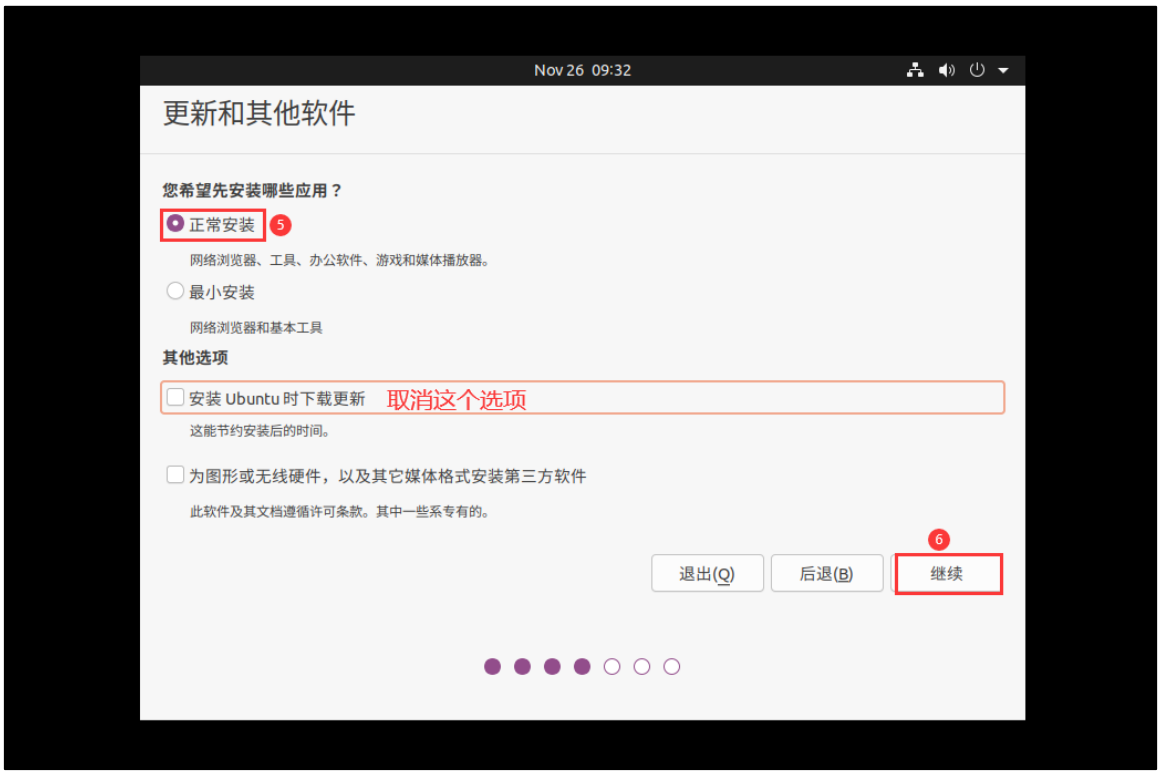


图 1.2.3.8 安装 Ubuntu 操作系统(5)

图 1.2.3.9 安装 Ubuntu 操作系统(6)

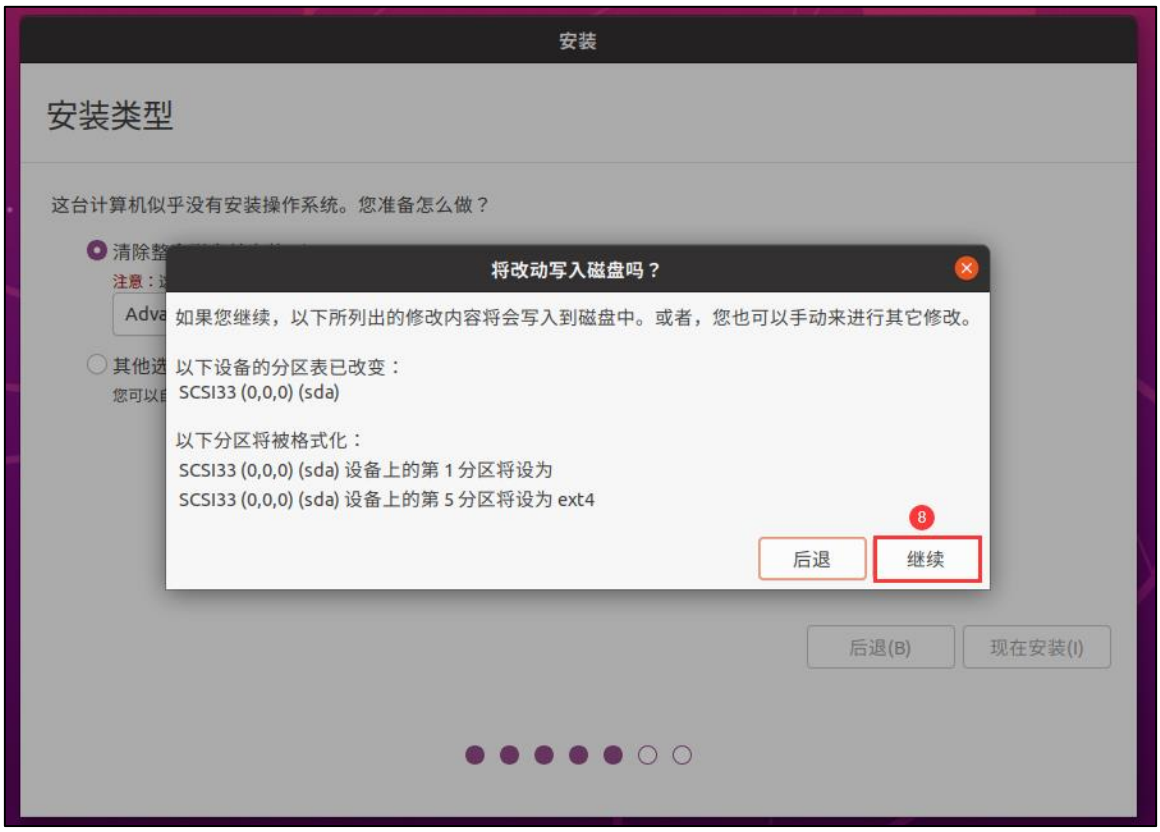


图 1.2.3.10 安装 Ubuntu 操作系统(7)

图 1.2.3.11 安装 Ubuntu 操作系统(8)

输入用户所在城市名称，这个信息主要用于确定用户所在地所处时区。

图 1.2.3.12 安装 Ubuntu 操作系统(9)

点击“继续”按钮后，开始正式安装：

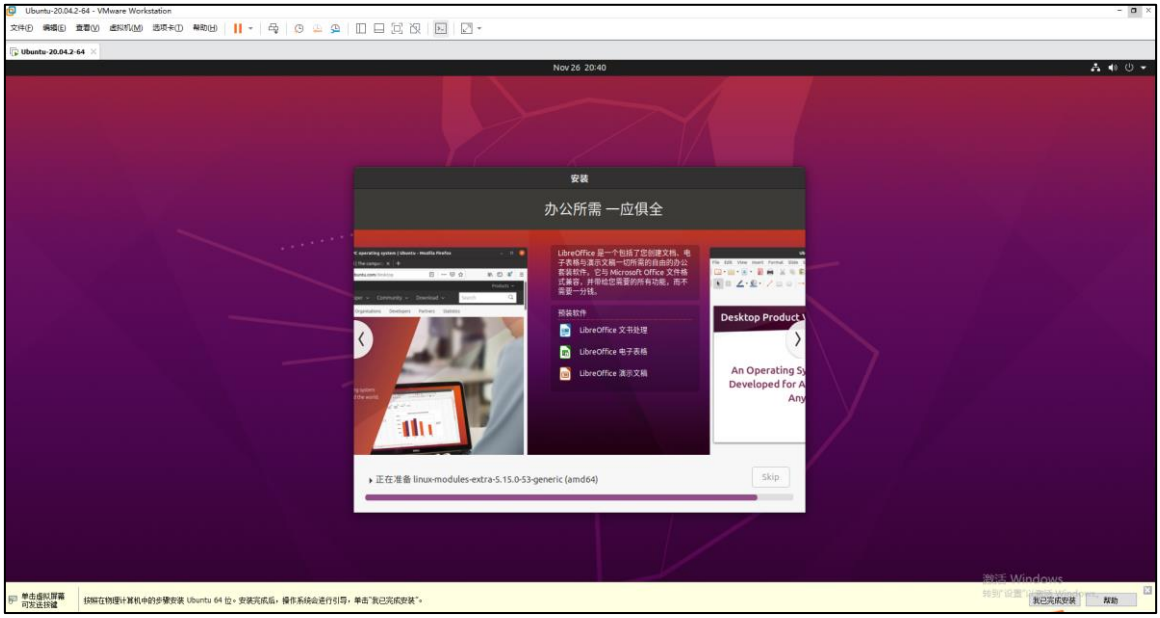


图 1.2.3.13 安装 Ubuntu 操作系统(10)

然后等待系统安装完成，安装完成后将会提示用户重启系统，如图 1.2.3.14 所示：

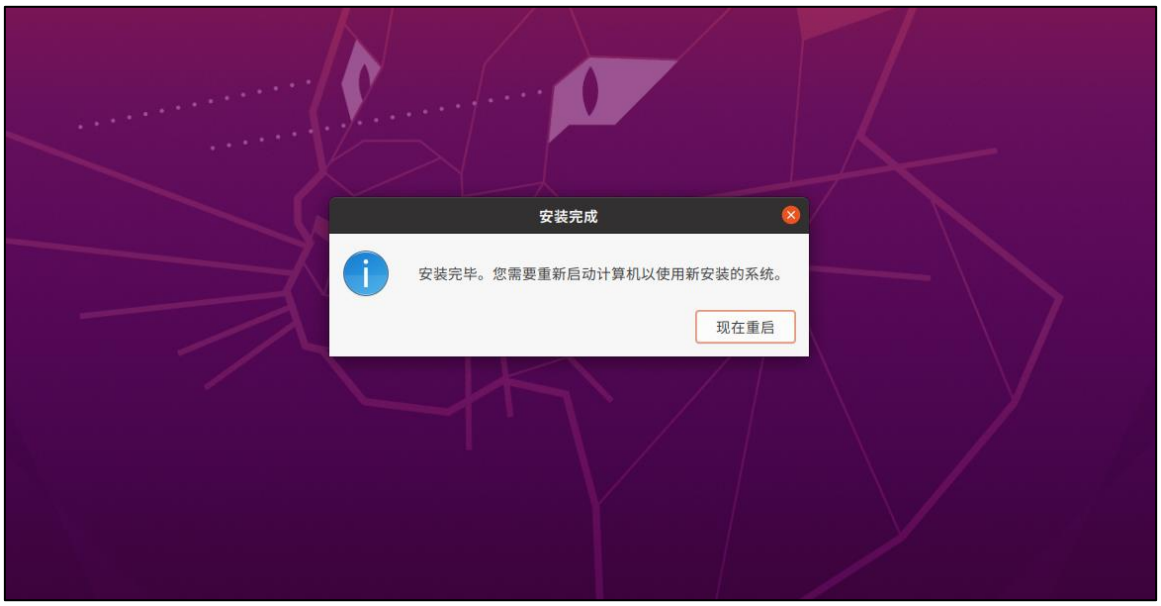


图 1.2.3.14 Ubuntu 系统安装完成

此时先别着急点击“现在重启”按钮，我们需要先关闭虚拟机，移除 Ubuntu 20.04 系统镜像文件。按照图 1.2.3.15~1.2.3.16 所示步骤关闭虚拟机：

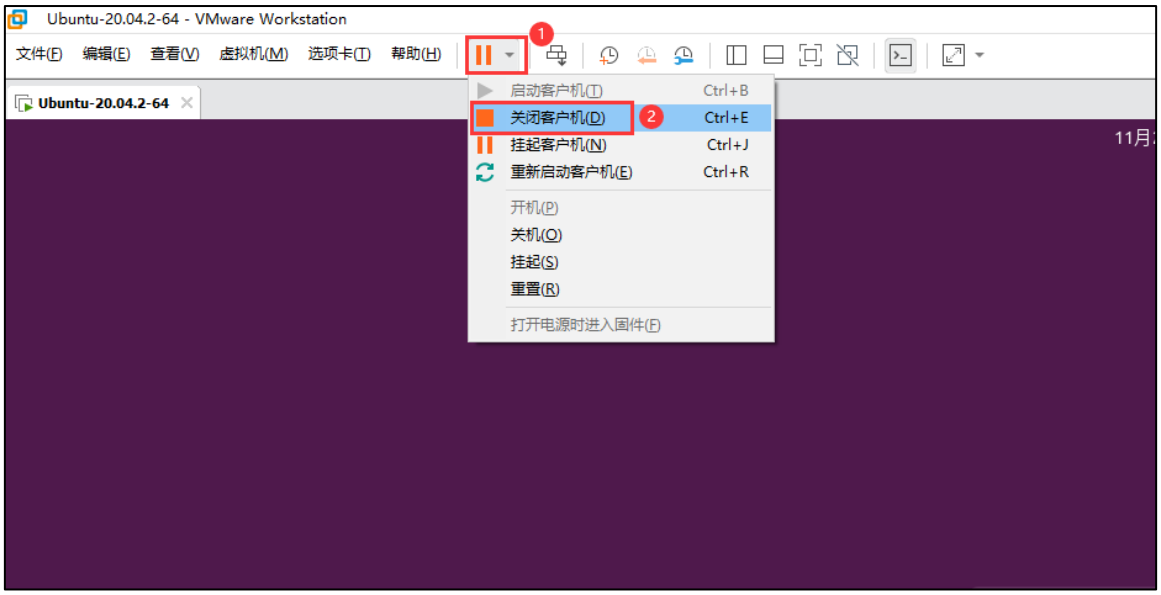


图 1.2.3.15 关闭虚拟机(1)

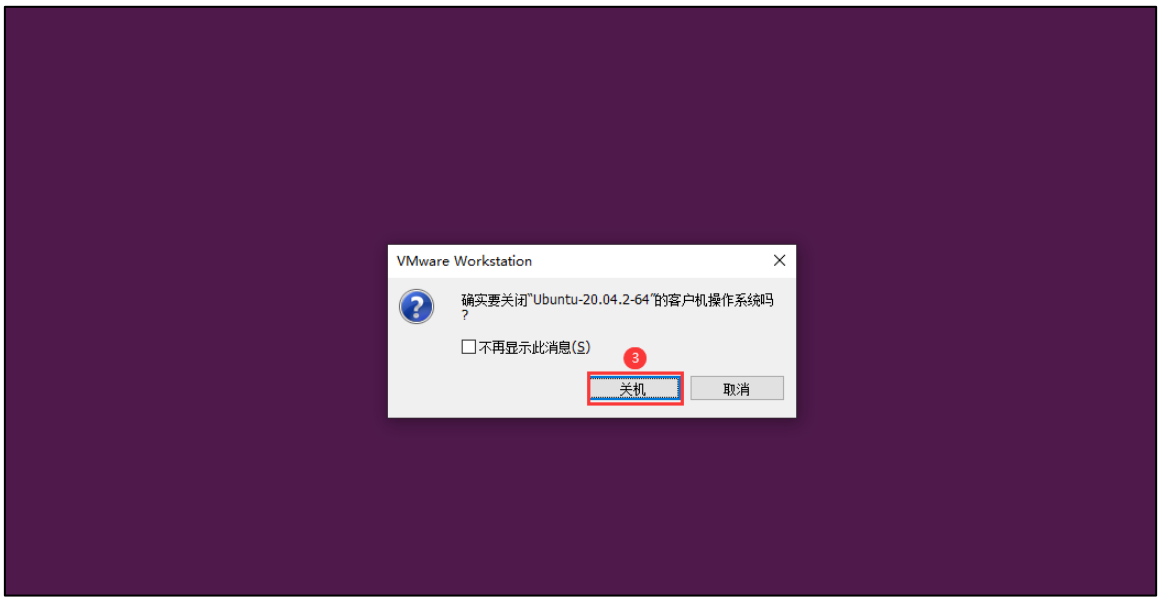


图 1.2.3.16 关闭虚拟机(2)

虚拟机关闭之后，点击菜单栏“**虚拟机(M)→设置(S)**”进入虚拟机设置界面，按照图 1.2.3.17 所示移除 Ubuntu 系统镜像文件：

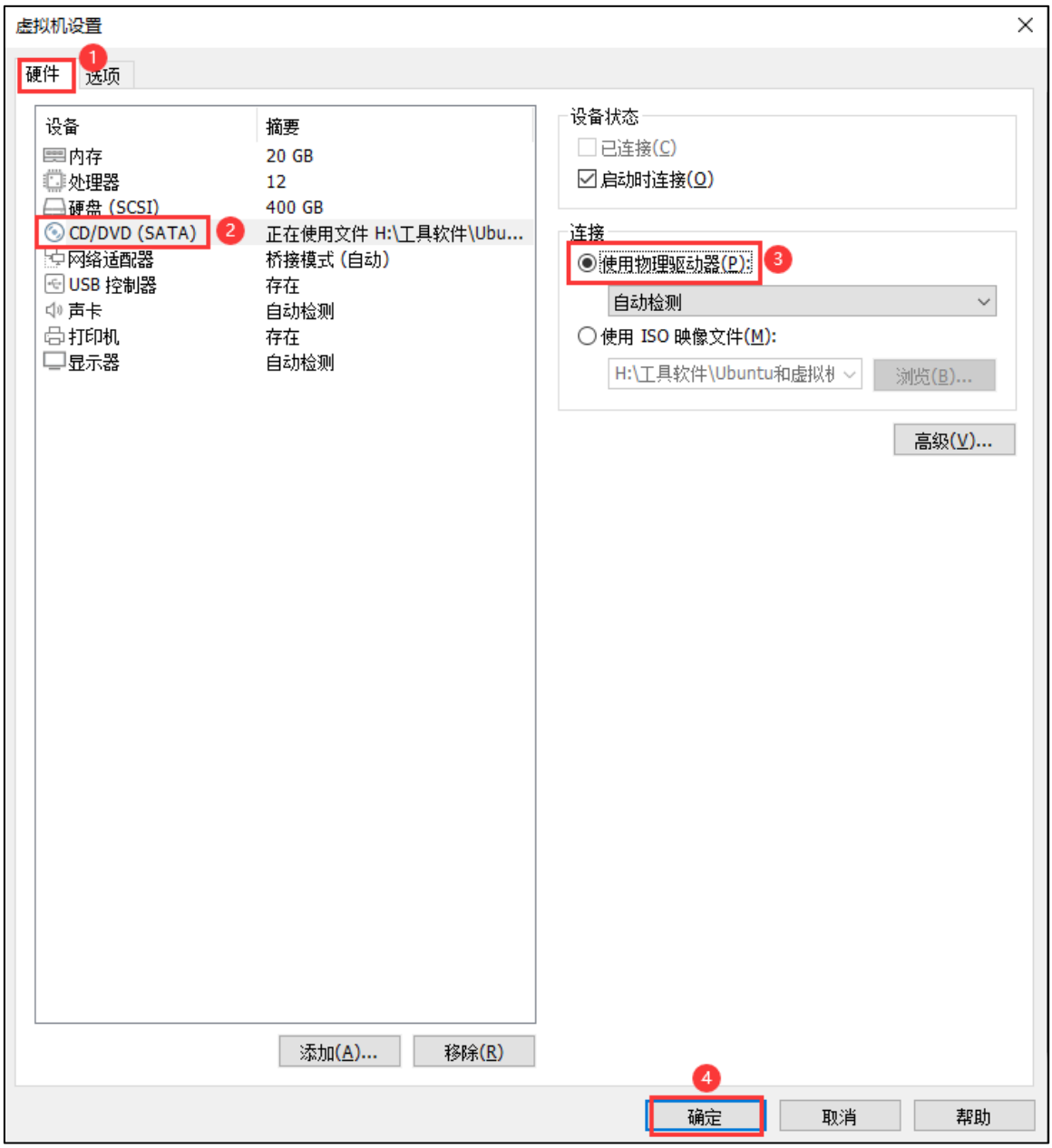


图 1.2.3.17 移除 Ubuntu 系统镜像文件

点击“确定”按钮后退出设置窗口，之后便可以开启虚拟机了，按照图 1.2.3.18 所示步骤开启虚拟机（也就是启动 Ubuntu 系统）：

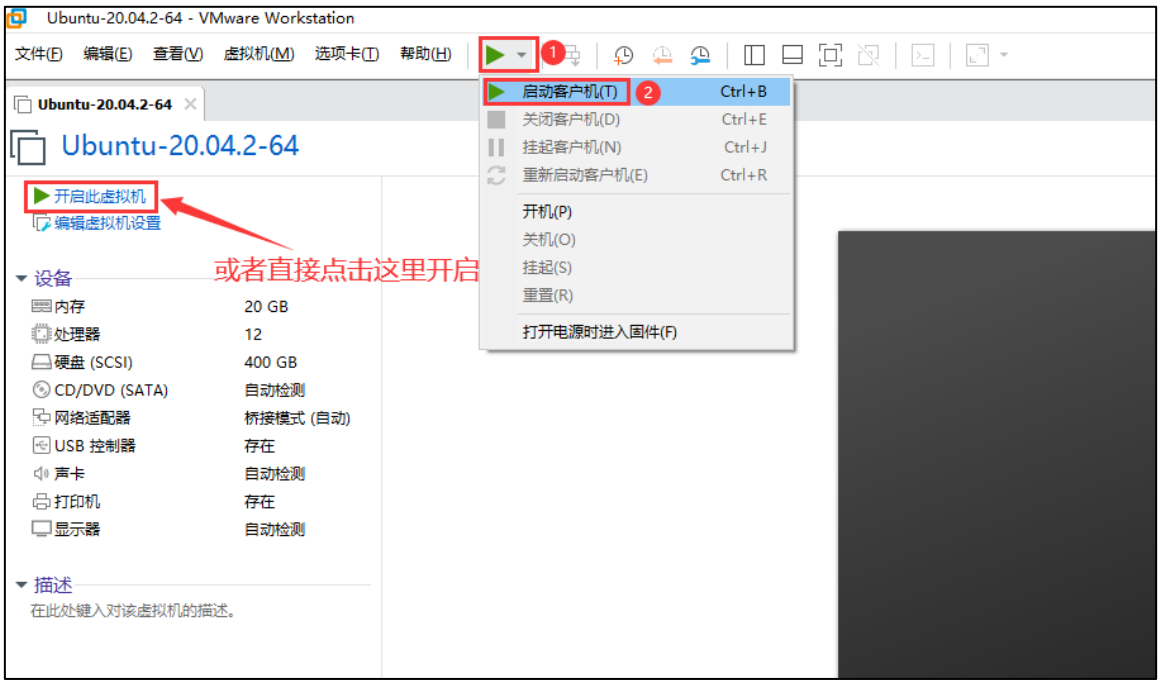


图 1.2.3.18 开启虚拟机

系统启动后如图 1.2.3.19 所示：

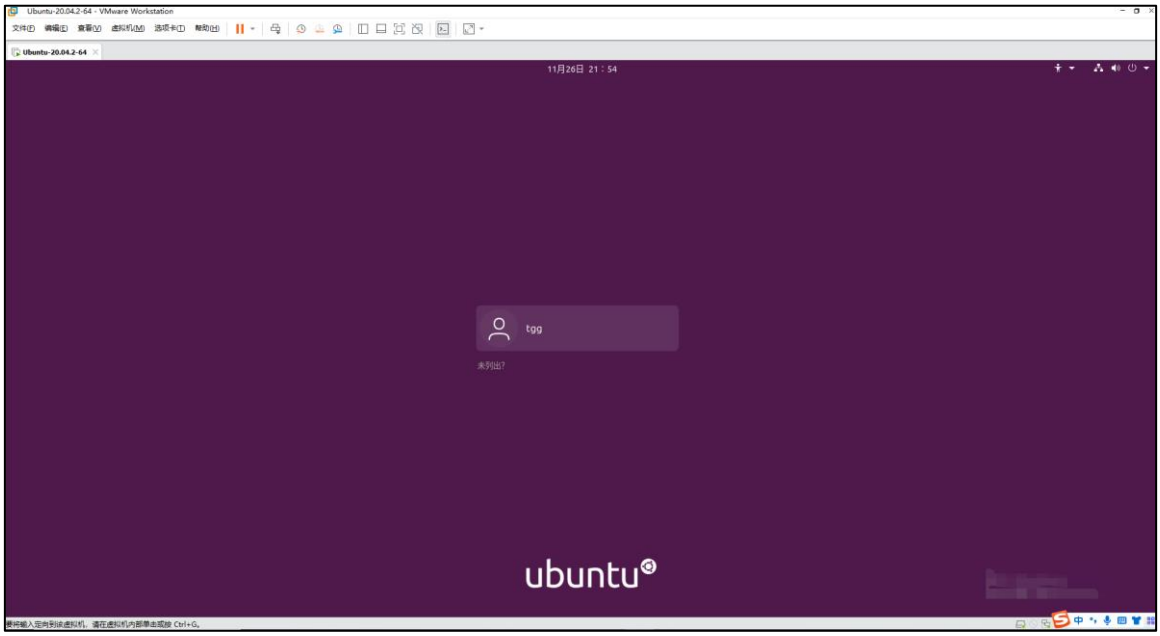


图 1.2.3.19 用户登录界面

点击接着用户名、输入密码后按回车键登录系统，进入 Ubuntu 系统主界面：

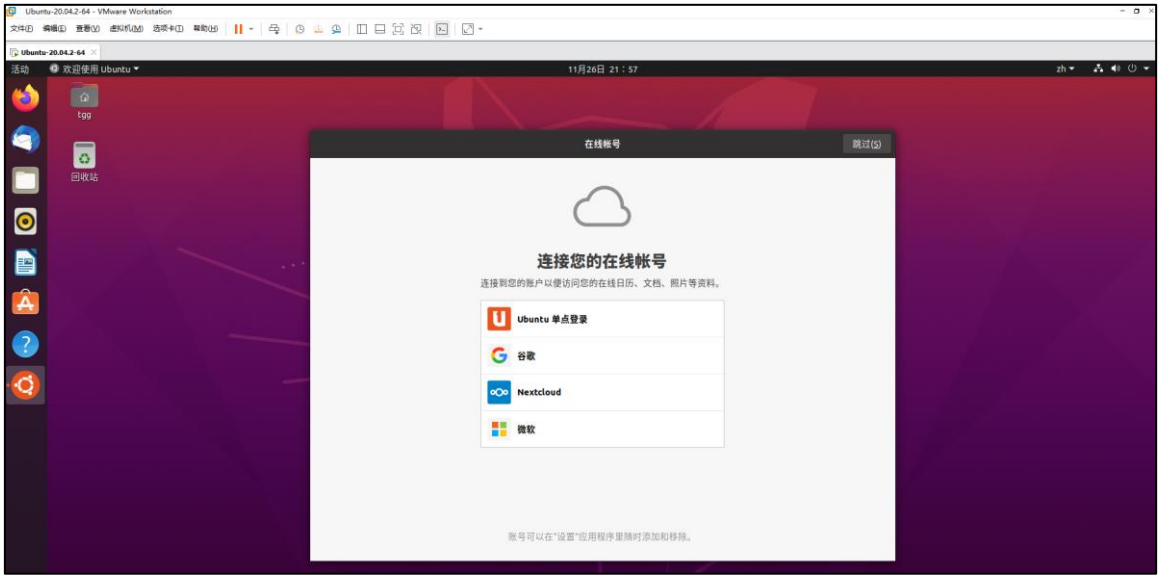


图 1.2.3.20 Ubuntu 系统主界面

首次进入 Ubuntu 20.04 系统桌面，会弹出“在线账号”窗口（也就是一个简易的使用引导），如图 1.2.3.20 所示；大家可以按照指引一步一步观看，也可以直接退出，如图 1.2.3.21 所示：

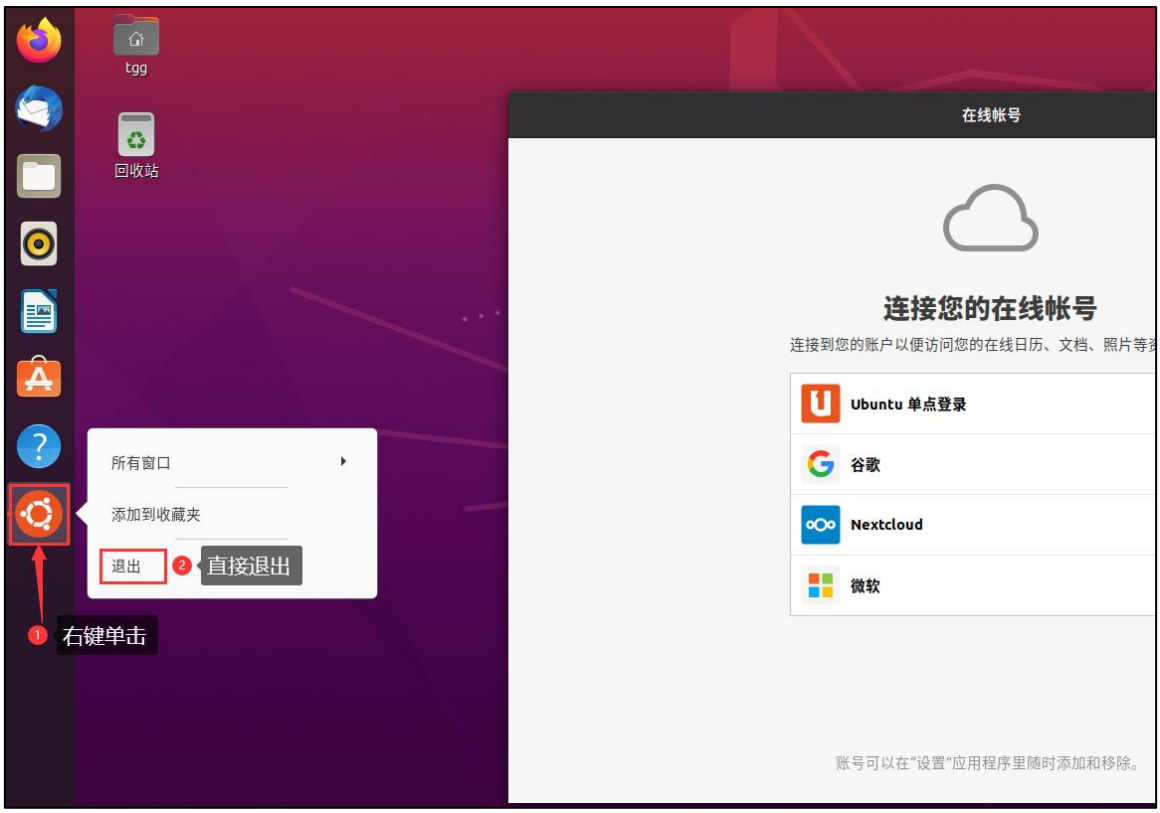


图 1.2.3.21 退出使用指引窗口

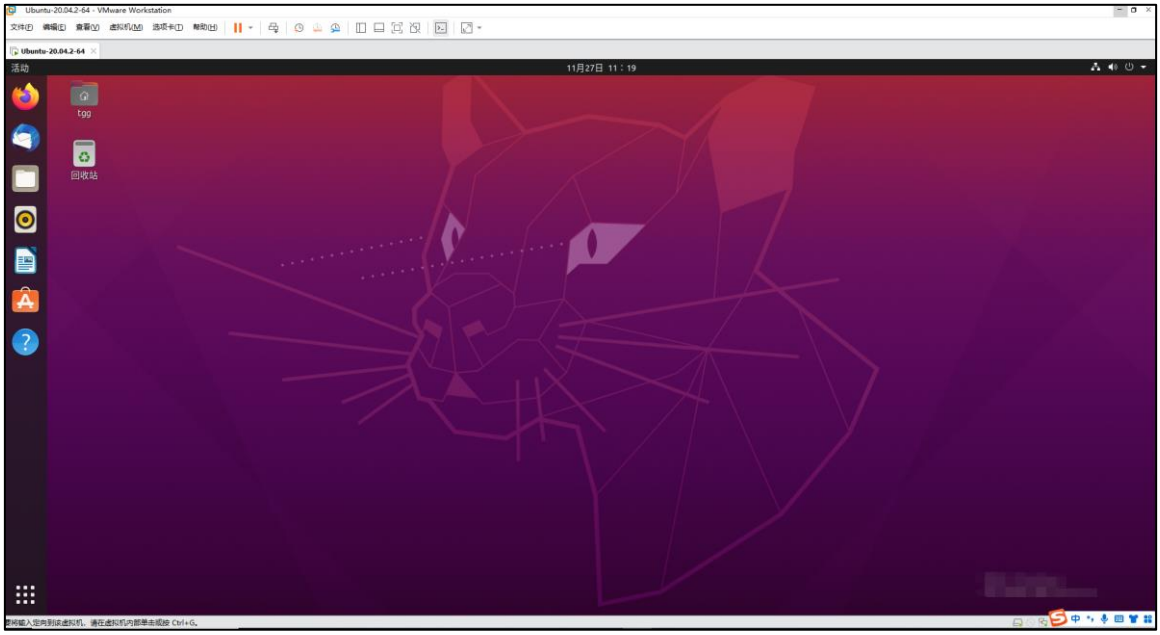


图 1.2.3.22 Ubuntu 系统桌面

至此，Ubuntu 20.04 操作系统安装完成！

第二章 开发环境搭建

在做 RK3568 嵌入式 Linux 开发之前，我们需要先搭建好开发环境；包括 Windows 和 Ubuntu 这两种操作系统下的环境搭建，因为嵌入式 linux 开发一般都是基于 Windows+Ubuntu 双系统开发环境。

本章将分为如下几个小节：

2.1 Ubuntu 系统设置
2.2 Ubuntu 与 Windows 之间文件互传
2.3 Ubuntu 系统下搭建 tftp 服务器
2.4 Ubuntu 系统下搭建 nfs 服务器
2.5 Ubuntu 系统下搭建 ssh 服务器
2.6 CH340 串口驱动安装
2.7 MobaXterm 软件安装
2.8 Rockchip USB 驱动安装
2.9 ADB 工具安装

2.1 Ubuntu 系统设置

本小节对 Ubuntu 系统进行一个简单的设置。

2.1.1 设置 root 用户密码

刚安装好的 Ubuntu 系统没有设置 root 用户密码，我们需要打开终端（Ctrl+T）执行如下命令设置 root 用户密码：

```
sudo passwd root
```

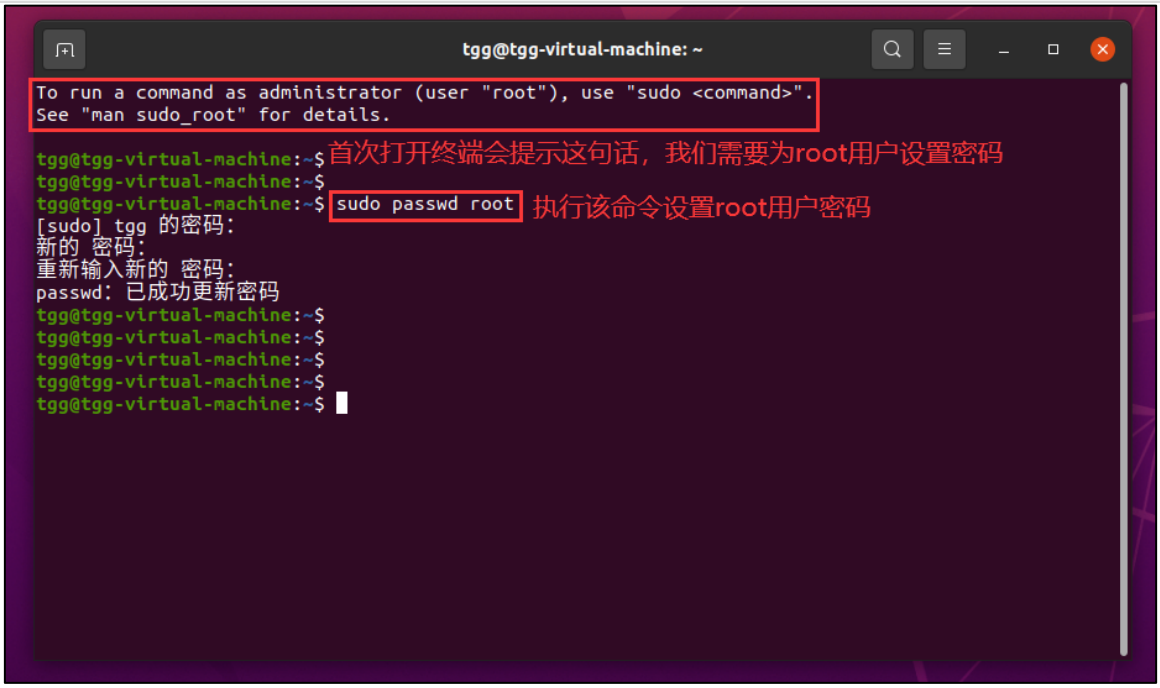


图 2.1.1.1 设置 root 用户密码

2.1.2 更换软件下载源

Ubuntu 系统默认的软件下载源由于服务器的原因，在国内的下载速度往往比较慢，这时我们可以将 Ubuntu 系统的软件下载源更改为国内软件源，譬如阿里源、中科大源、清华源等等，下载速度相比 Ubuntu 官方软件源会快很多！

按照图 2.1.2.1~2.1.2.7 所示步骤进行更改：

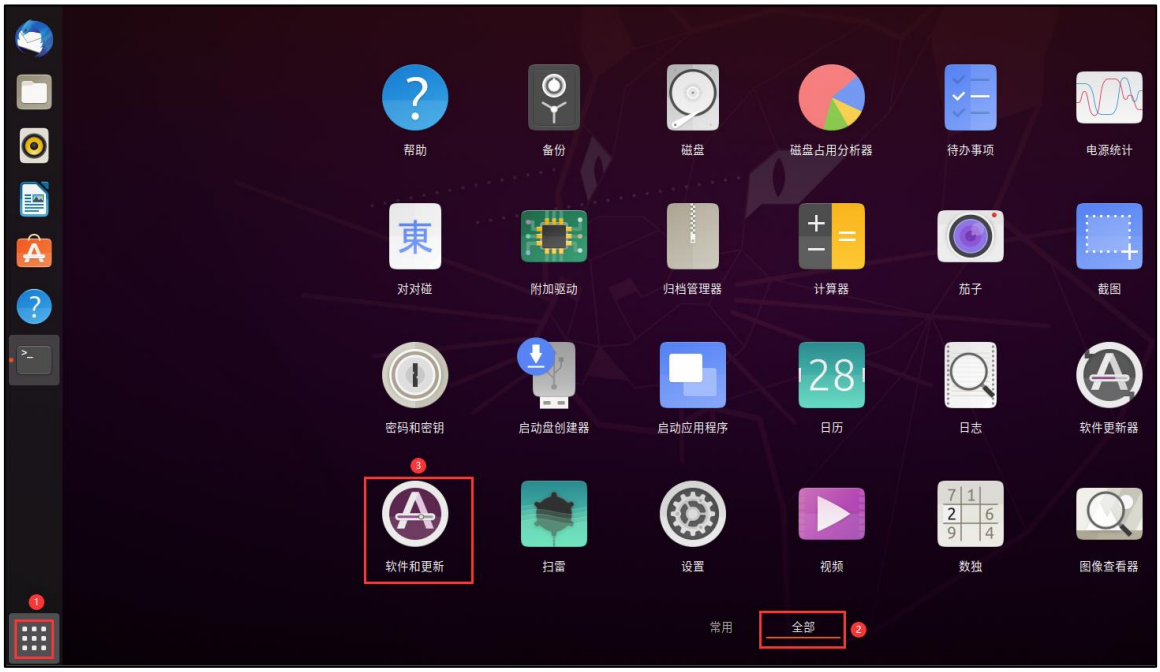


图 2.1.2.1 更改软件下载源(1)

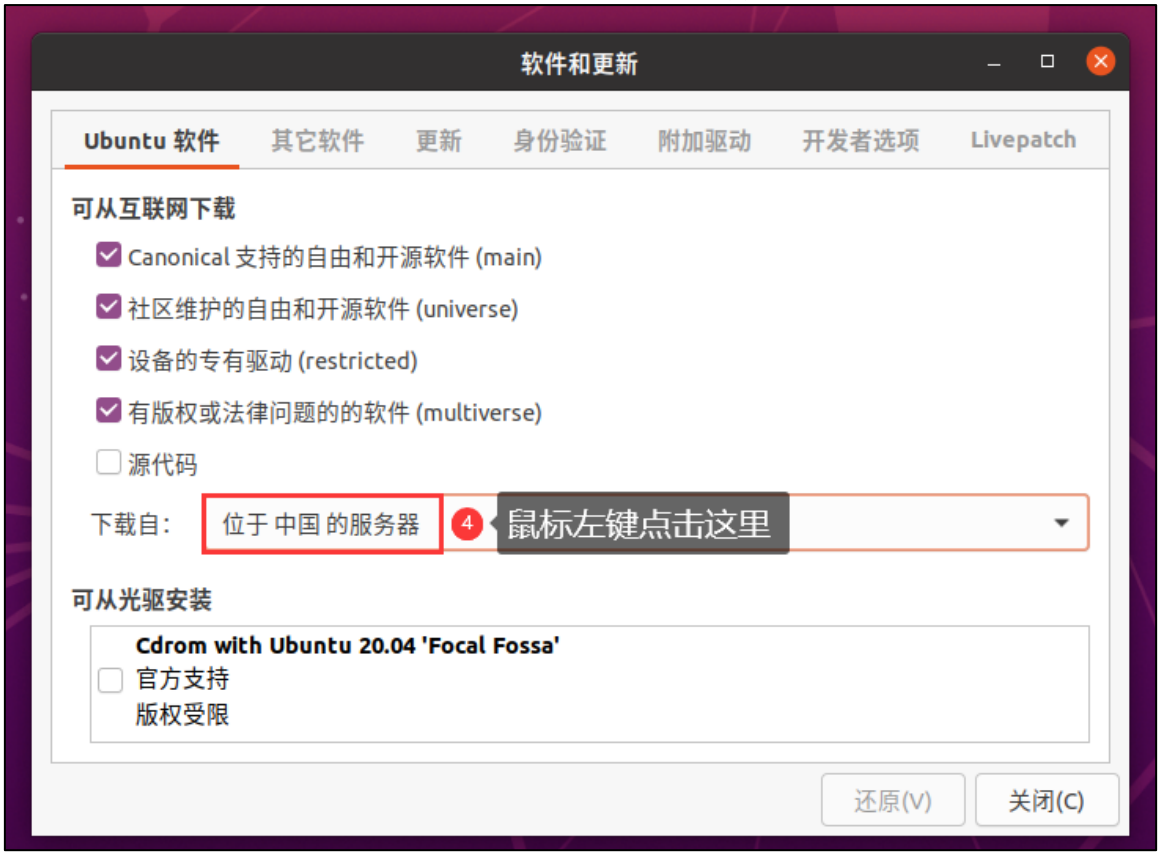


图 2.1.2.2 更改软件下载源(2)

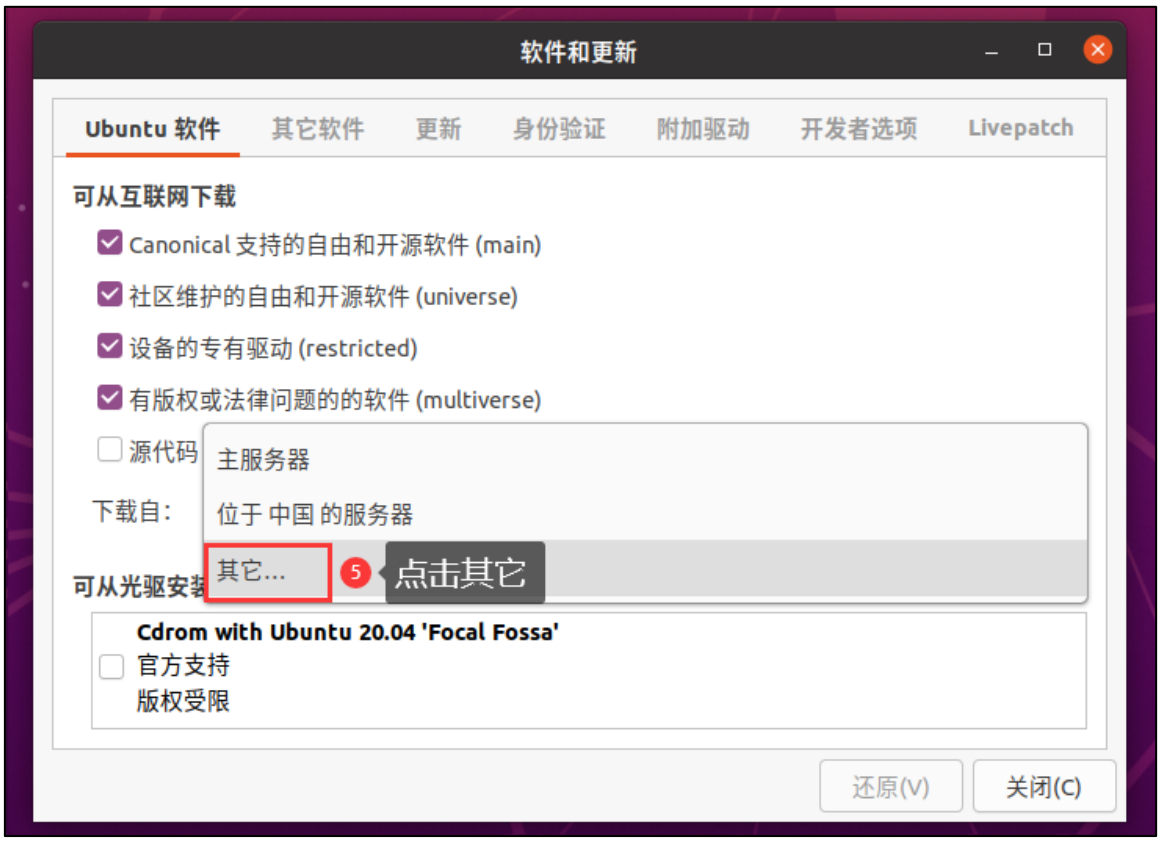


图 2.1.2.3 更改软件下载源(3)

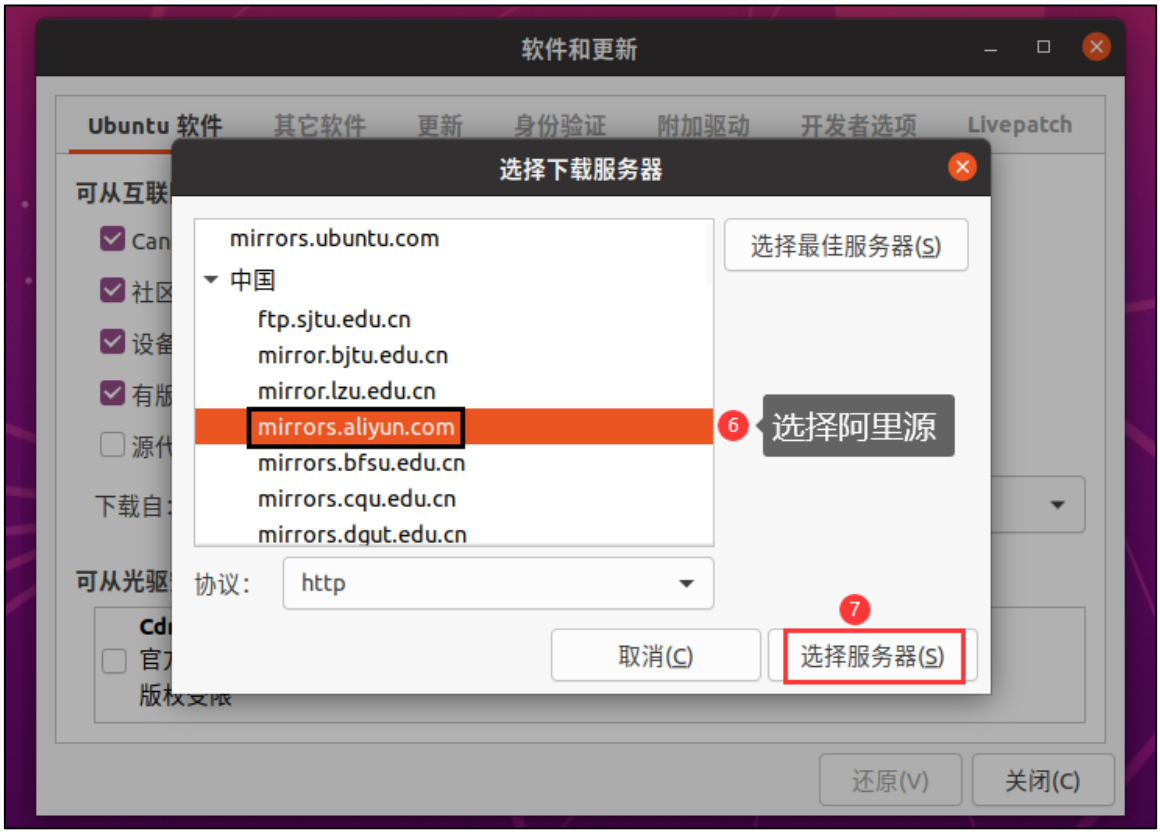


图 2.1.2.4 更改软件下载源(4)

从列表可知，有很多国内的下载源供我们选择，这里我们以阿里源 mirrors.aliyun.com 为例，当然也可以选择其它下载源。

图 2.1.2.5 更改软件下载源(5)

图 2.1.2.6 更改软件下载源(6)

图 2.1.2.7 更改软件下载源(7)

图 2.1.2.8 更改软件下载源(8)

2.1.3 关闭自动更新

同样，打开 Ubuntu 系统“**软件与更新**”窗口，按照图 2.1.3.1 所示步骤关闭自动更新功能：

图 2.1.3.1 关闭自动更新功能

2.2 Ubuntu 与 Windows 之间文件互传

我们在开发过程中，经常需要在 Windows 系统与 Ubuntu 系统之间进行文件传输，文件互传的方式比较多，譬如共享文件夹、Samba、FTP、scp 等等，本小节向用户介绍通过 FTP 方式进行文件传输。

通过 FTP（File Transfer Protocol，文件传输协议）在 Windows 与 Ubuntu 之间进行文件传输，需要完成以下两件事情。

2.2.1 Ubuntu 系统下搭建 FTP 服务器

在 Ubuntu 系统下打开终端，执行如下命令安装 FTP 服务：

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

vsftpd 安装完成后，使用 vi 命令打开/etc/vsftpd.conf 配置文件，如果没有安装 vim 软件，则需要先通过如下命令安装 vim 编辑器：

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

然后再执行如下命令打开/etc/vsftpd.conf 配置文件：

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

打开配置文件后，找到如下两行，确保其前面没有“#”（“#”号表示注释，我们要取消注释）：

```
26 #
27 # Uncomment this to allow local users to log in.
28 local_enable=YES
29 #
30 # Uncomment this to enable any form of FTP write command.
31 write_enable=YES
32 #
```

图 2.2.1.1 修改/etc/vsftpd.conf 配置文件

默认情况下，“write_enable=YES”前面有一个“#”号，我们需要将其去掉，使能该配置。修改完成后保存退出，然后执行如下命令重启 FTP 服务：

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

可通过如下命令确认 FTP 服务是否开启：

```
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:~$
```

图 2.2.1.2 确认 FTP 服务是否开启

2.2.2 Windows 下安装 FTP 客户端

我们需要在 Windows 系统下安装一个 FTP 客户端软件，这里选择 FileZilla 作为 FTP 客户端软件，这是一个免费的 FTP 客户端软件。

ATK-RK3568 开发板资料包中已经给用户提供了 FileZilla 软件安装包，路径为：**开发板光盘 A 盘-基础资料→04、软件→FileZilla_3.61.0_win64-setup.exe**，用户也可以通过链接地址：https://www.filezilla.cn/download/client，自己下载安装包文件。

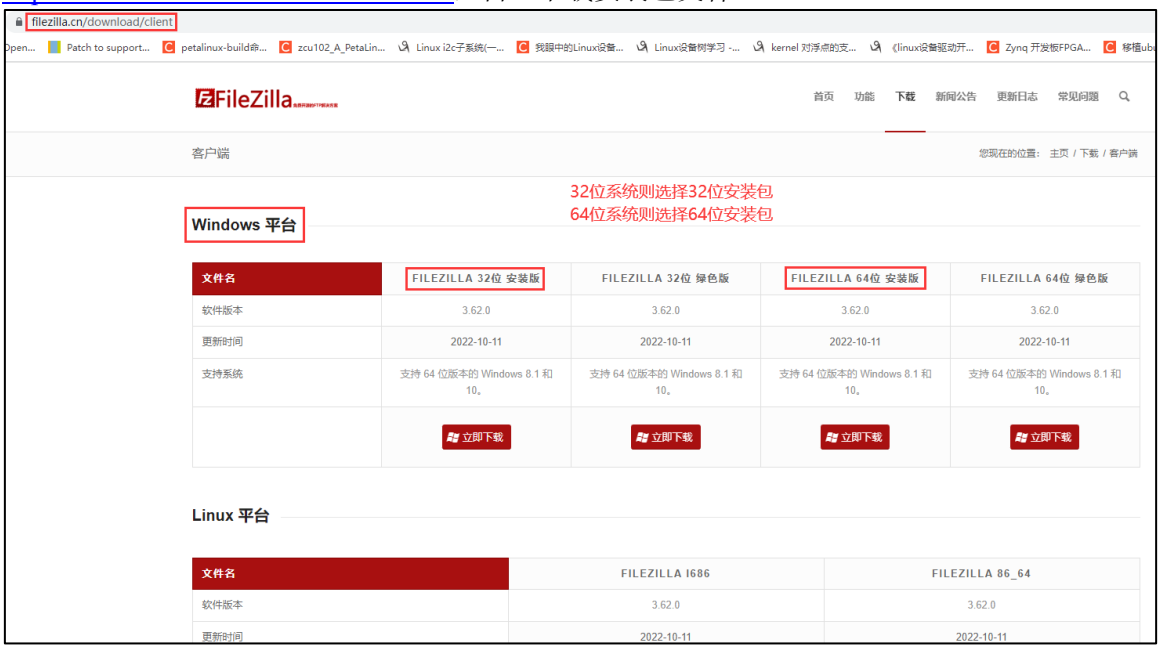


图 2.2.2.1 下载 FileZilla 软件安装包

下载完成如图 2.2.2.2 所示：

图 2.2.2.2 FileZilla 软件安装包

接着双击安装包文件 FileZilla_3.61.0_win64-setup.exe，按照图 2.2.2.3~2.2.2.8 所示步骤进行安装：

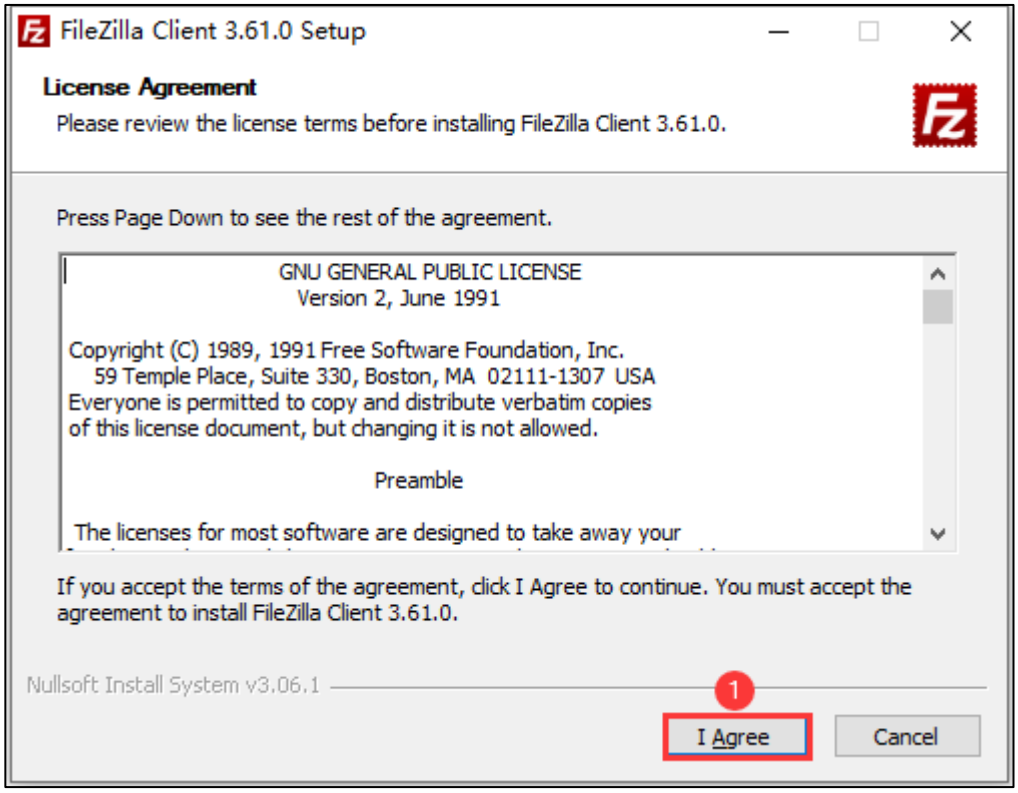


图 2.2.2.3 安装 FileZilla 软件(1)

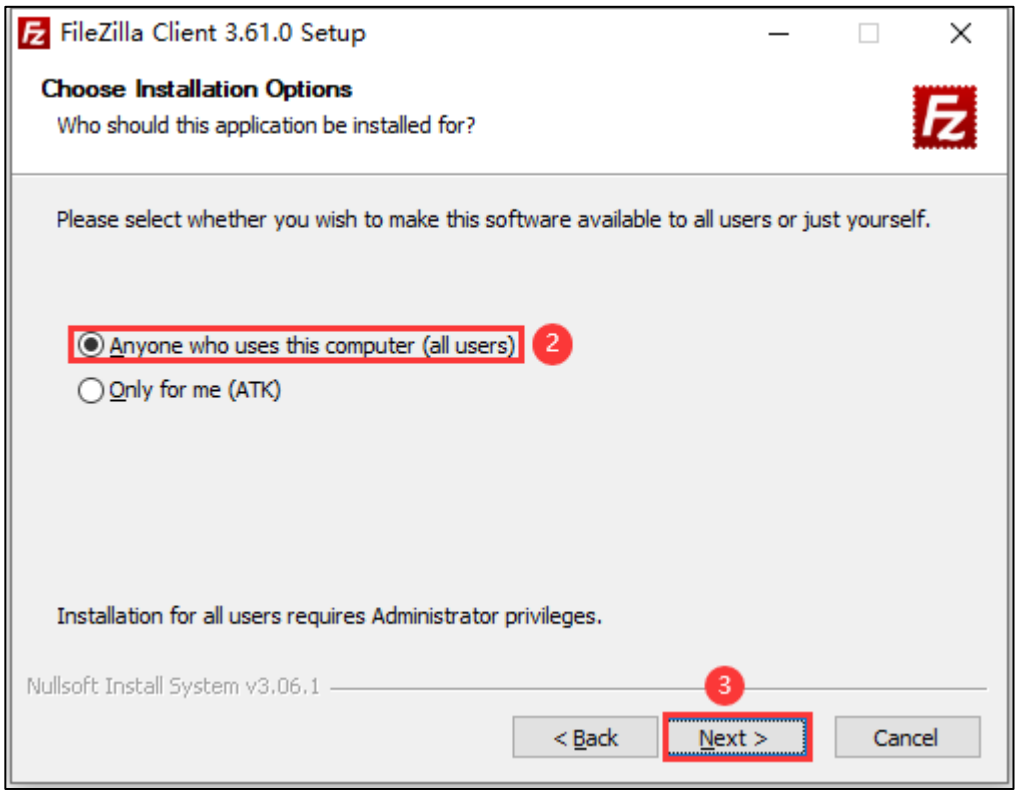


图 2.2.2.4 安装 FileZilla 软件(2)

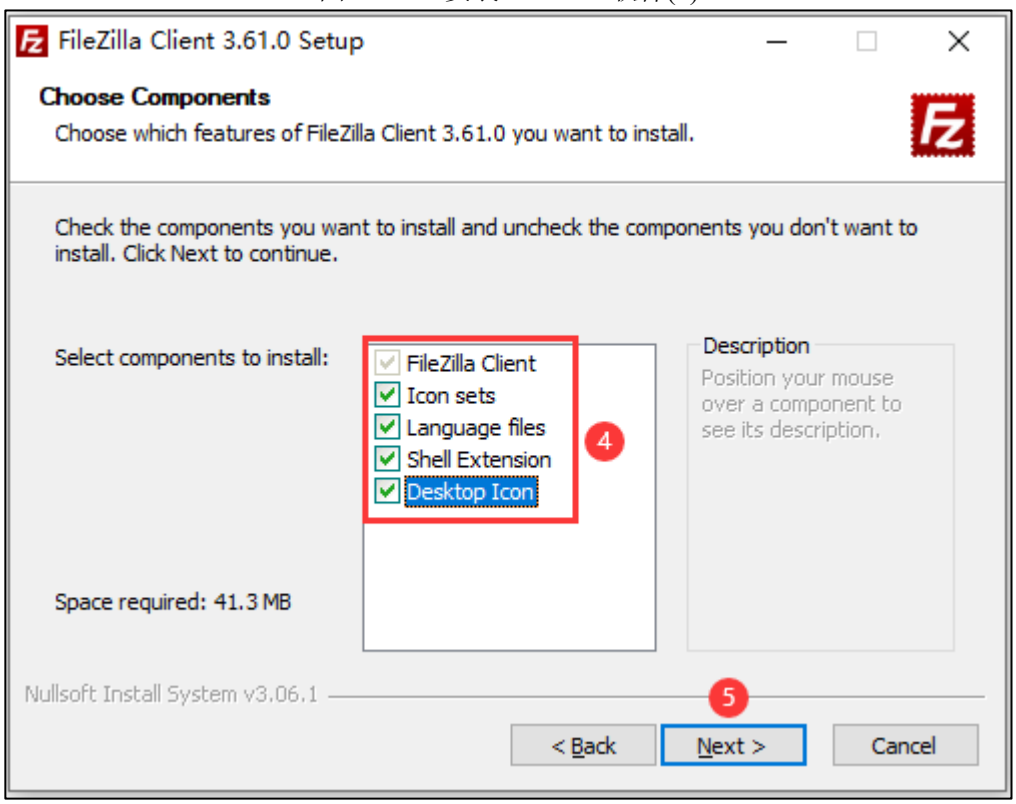


图 2.2.2.5 安装 FileZilla 软件(3)

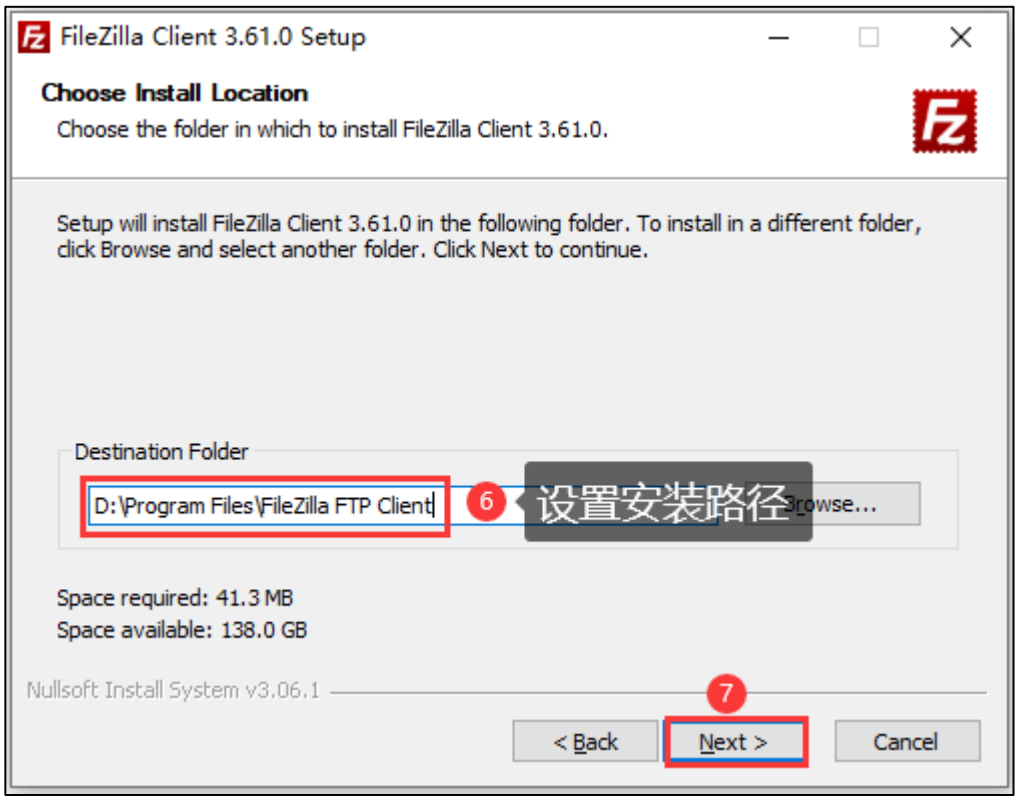


图 2.2.2.6 安装 FileZilla 软件(4)

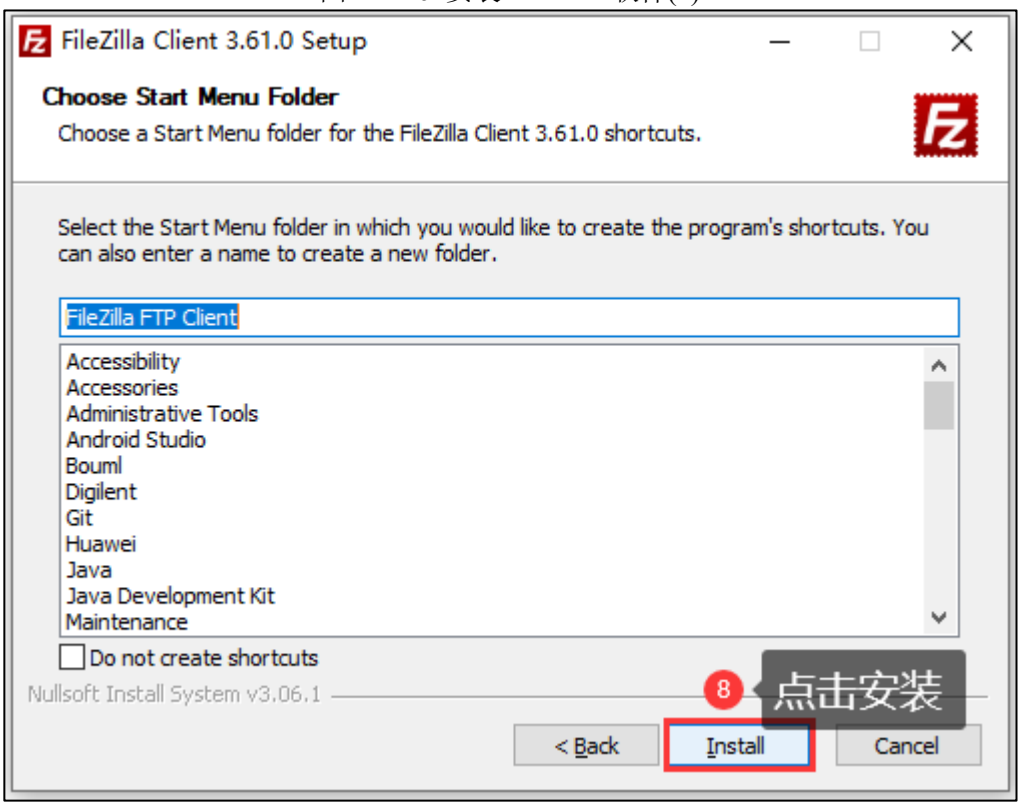


图 2.2.2.7 安装 FileZilla 软件(5)

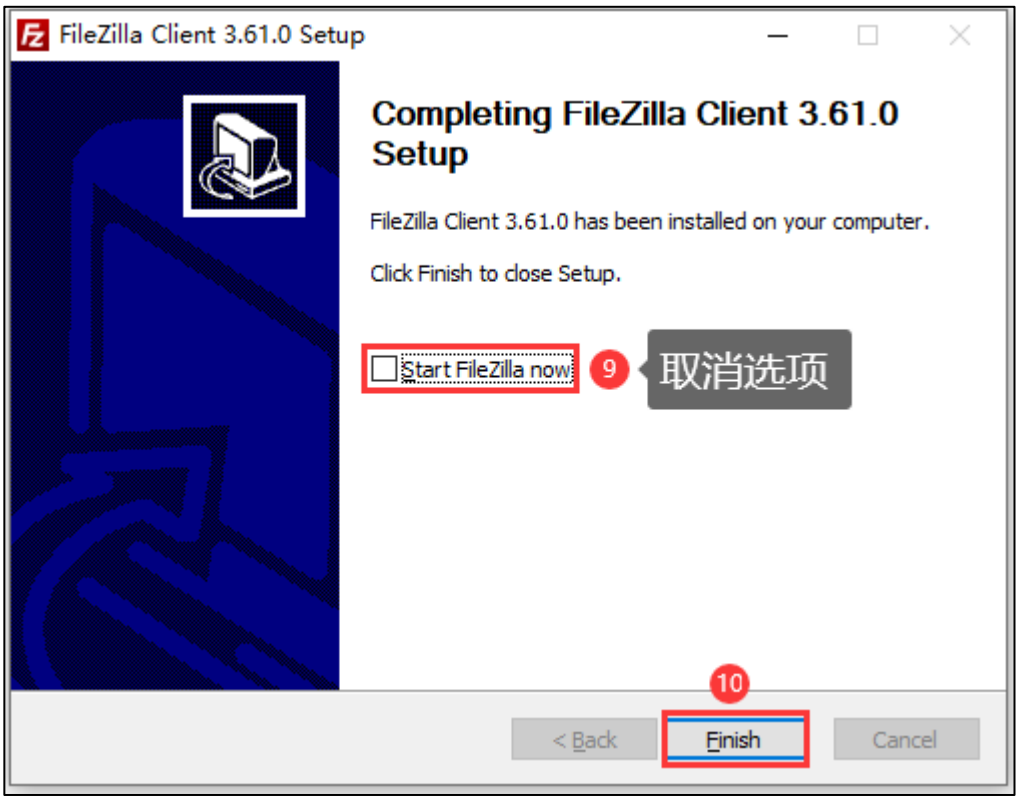


图 2.2.2.8 安装 FileZilla 软件(6)

至此，软件安装完成，桌面会生成对应的快捷方式：

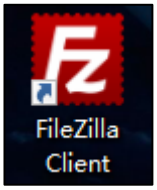


图 2.2.2.9 FileZilla 桌面图标

2.2.3 FileZilla 使用方法

接下来介绍一下 FileZilla 的使用方法，首先双击 FileZilla 桌面图标打开该软件：

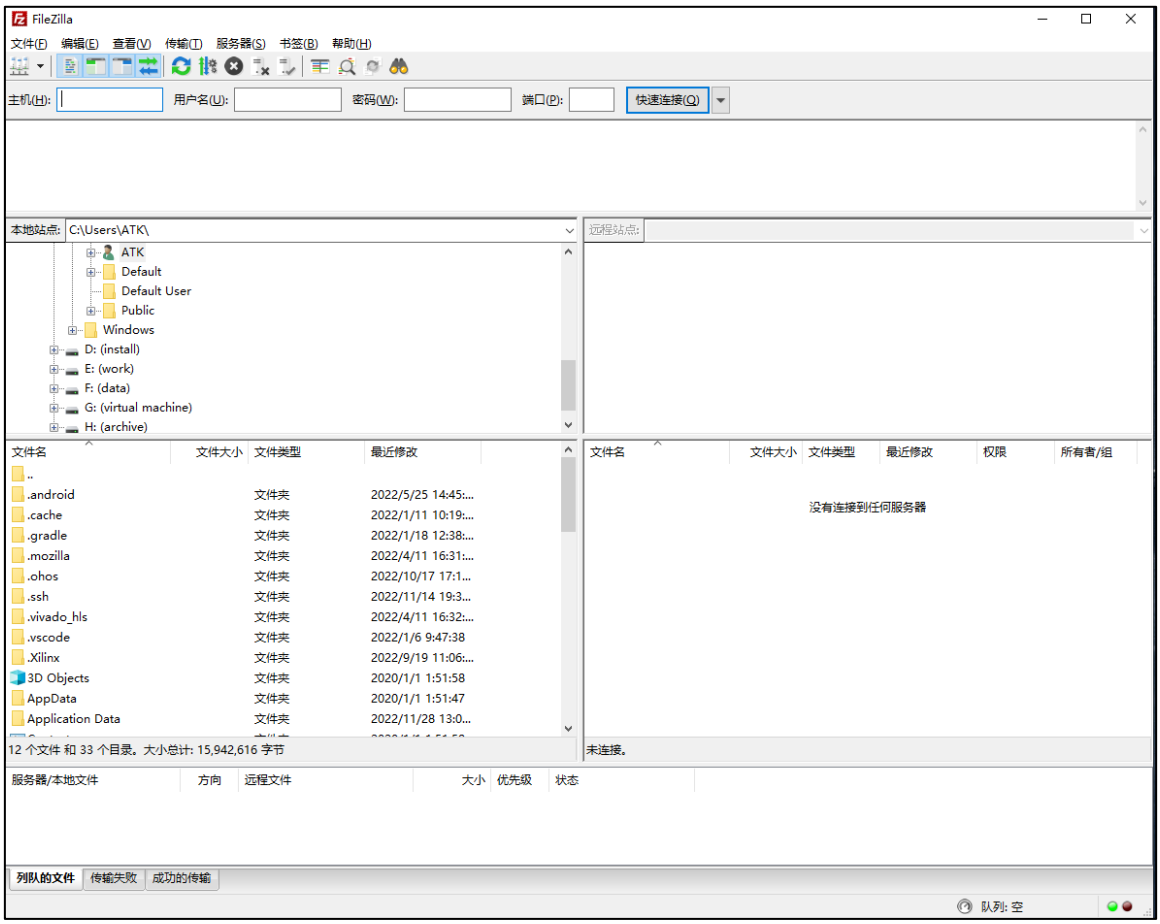


图 2.2.3.1 FileZilla 软件界面

Ubuntu 作为 FTP 服务器，Windows 作为 FTP 客户端，在进行文件传输之前，客户端需要先连接到服务器。按照图 2.2.3.2~2.2.3.6 所示步骤连接 FTP 服务器：

图 2.2.3.2 FTP 客户端连接 FTP 服务器(1)

图 2.2.3.3 FTP 客户端连接 FTP 服务器(2)

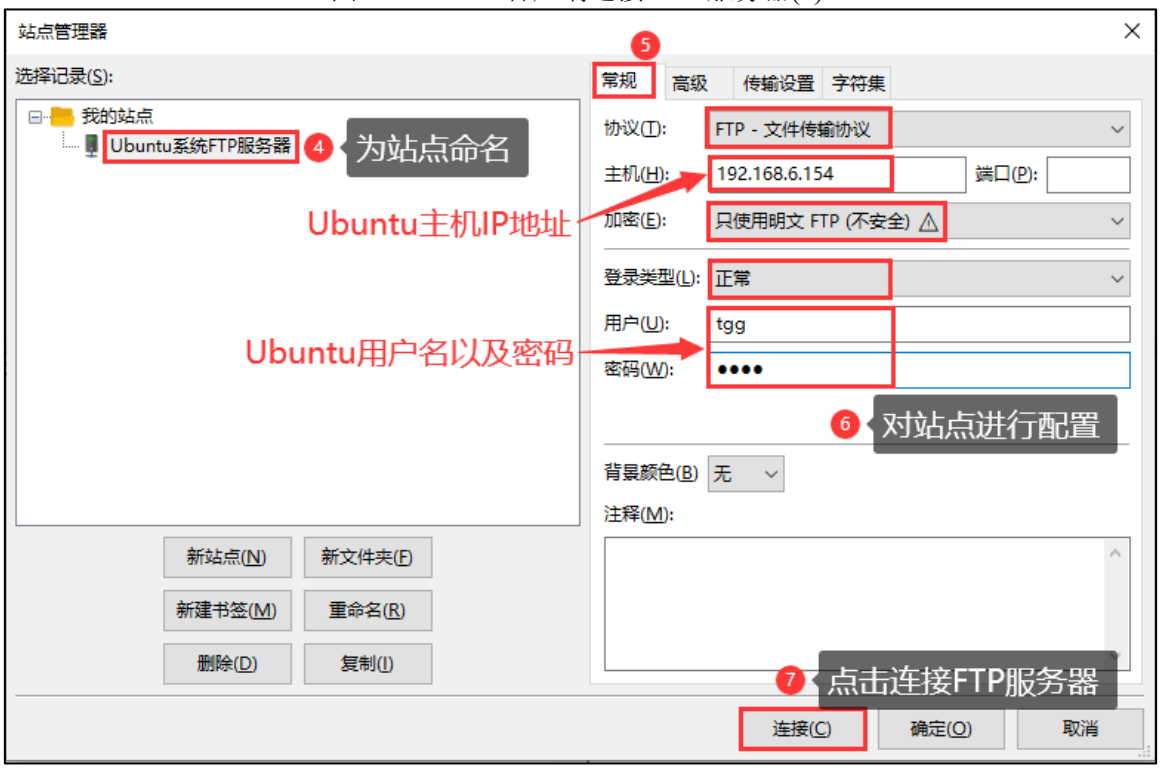


图 2.2.3.4 FTP 客户端连接 FTP 服务器(3)

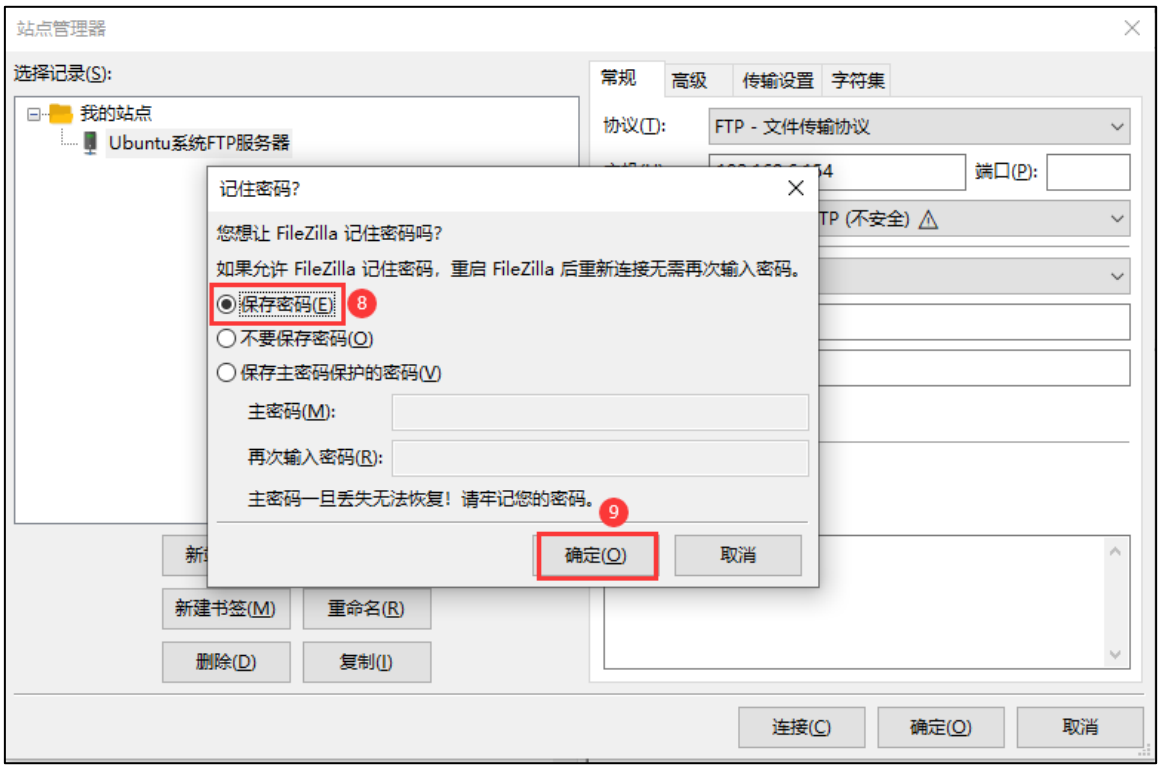


图 2.2.3.5 FTP 客户端连接 FTP 服务器(4)

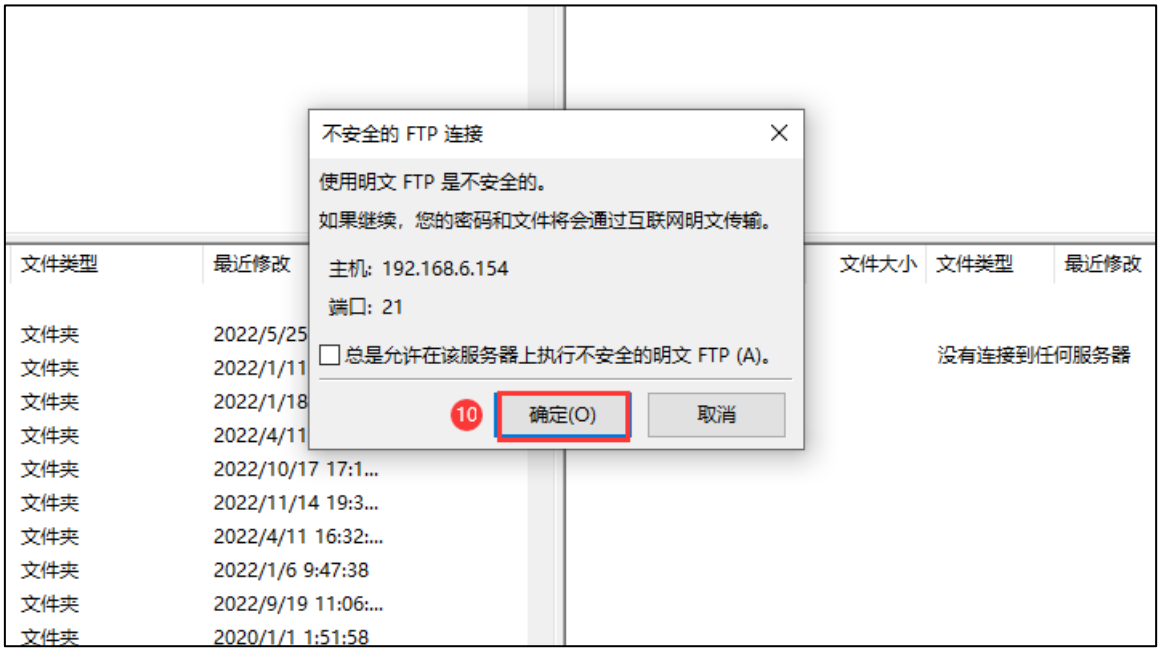


图 2.2.3.6 FTP 客户端连接 FTP 服务器(5)

连接成功如图 2.2.3.7 所示：

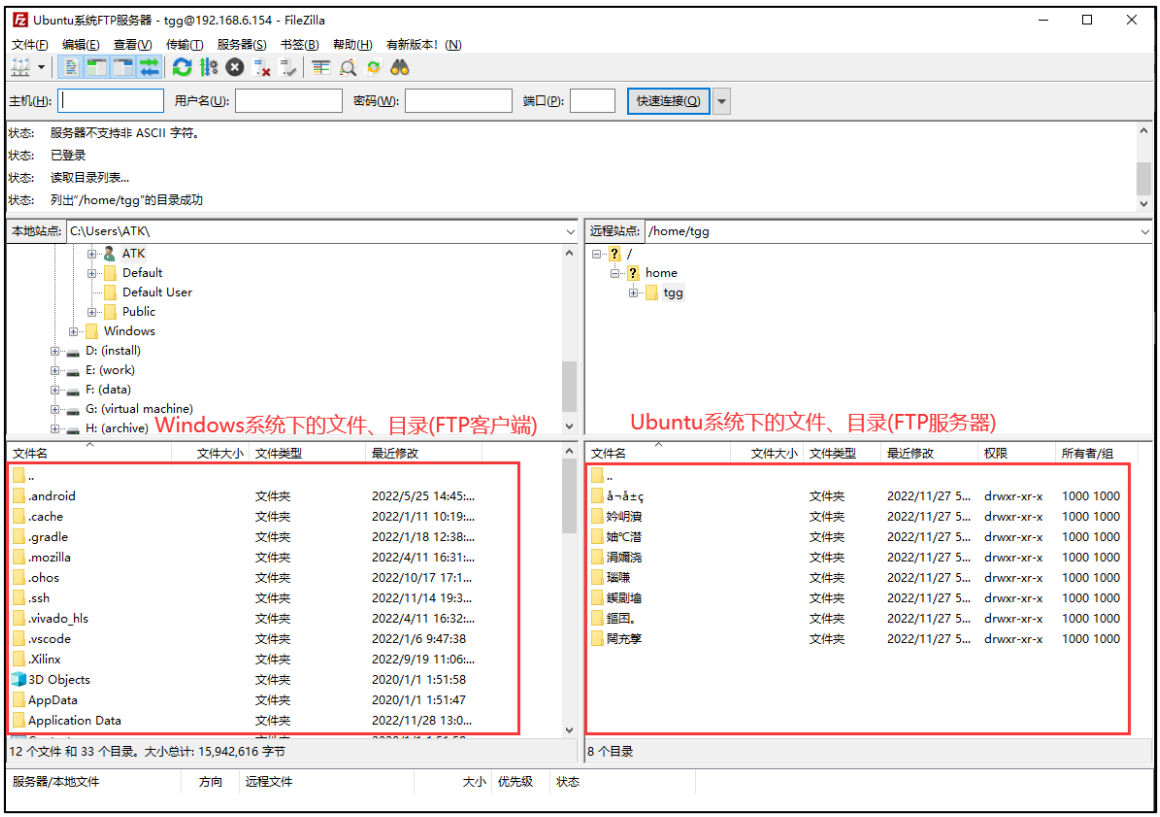


图 2.2.3.7 连接成功

其中左边是 Windows 系统下的文件、目录（FTP 客户端），右边是 Ubuntu 系统下的文件、目录（FTP 服务器，默认会进入到用户家目录，譬如“/home/tgg”）。从上图可知，Ubuntu 系统下的文件列表名称全是乱码，这是因为编码的问题导致的，我们需要修改编码方式，再次打开“站点管理器”，按照图 2.2.3.8 所示步骤设置编码方式：

图 2.2.3.8 设置字符编码方式(1)

图 2.2.3.9 设置字符编码方式(2)

再次连接之后，会发现文件名已经显示正常了，如图 2.2.3.10 所示：

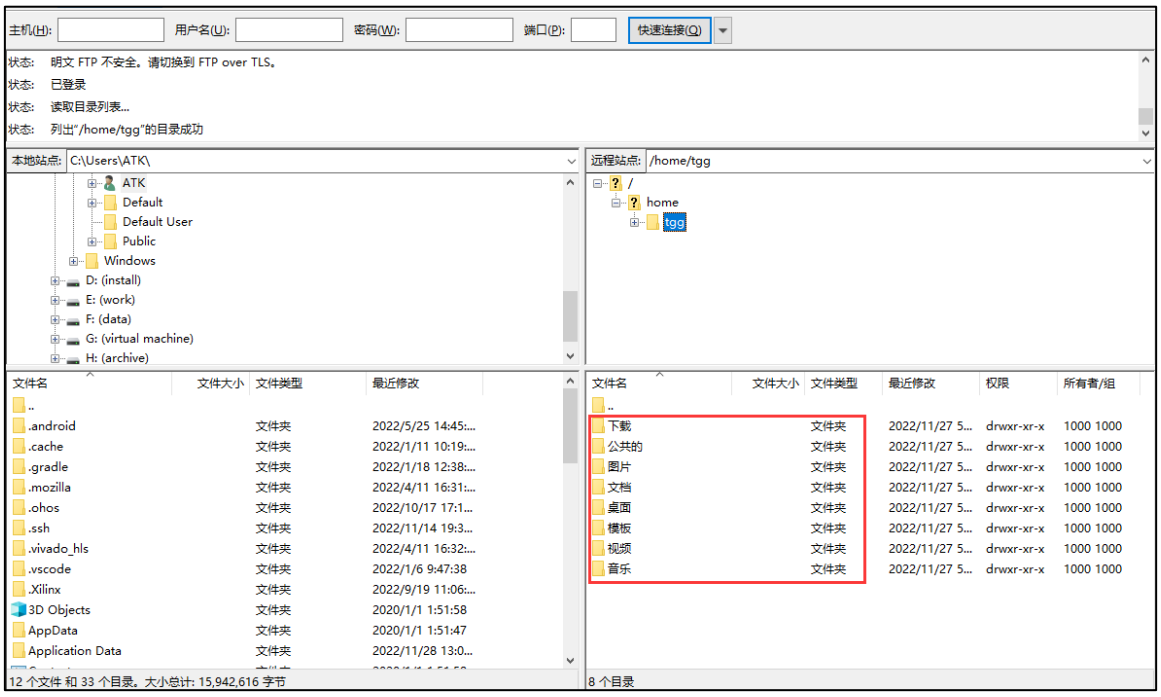


图 2.2.3.10 文件名显示正常

客户端成功连接上服务器成功后，便可以进行文件传输了。传输的方式也非常简单，选择要传输的文件，直接使用鼠标左键将其拖动到 Windows 目录区域或者 Ubuntu 系统目录区域即可！比如，将 Windows 系统下的 test.txt 文件拷贝到 Ubuntu 系统，首先在左侧 Windows 目录区域找到该文件 test.txt，然后使用鼠标左键将其拖动至右边 Ubuntu 目录区域释放即可；同理，Ubuntu 系统下的文件拷贝到 Windows 系统也是如此！

2.3 Ubuntu 系统搭建 tftp 服务器

后续有需求再考虑要不要加上！

2.4 Ubuntu 系统搭建 nfs 服务器

后续有需求再考虑要不要加上！

2.5 Ubuntu 系统搭建 ssh 服务器

在开发过程中，有时需要通过 ssh 远程连接、登录 Ubuntu 系统，为了实现这个功能，我们需要在 Ubuntu 系统下开启 ssh 服务。执行如下命令安装 openssh-server：

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

安装 openssh-server 后，ssh 服务将自动开启，可通过如下命令确认 ssh 服务是否开启：

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

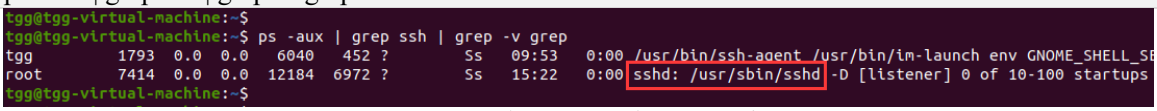


图 2.5.1 确认 ssh 服务是否开启

2.6 CH340 串口驱动安装

正点原子 ATK-DLRK3568 开发板使用了国产芯片 CH340 来实现 USB 转串口功能，可以直接通过 USB 线将开发板的调试串口连接到电脑、而无需使用 USB 转串口线，方便用户使用；但是需要在 Windows 下安装 CH340 驱动才能识别到开发板的调试串口。

开发板资料包中已经给用户提供了 CH340 驱动安装文件，路径为：**开发板光盘 A 盘-基础资料→04、软件→CH340 驱动(USB 串口驱动)_XP_WIN7 共用→SETUP.EXE**，双击 SETUP.EXE 可执行文件，按照图 2.6.1~2.6.2 所示步骤安装 CH340 驱动（安装之前先不要连接开发板调试串口）：

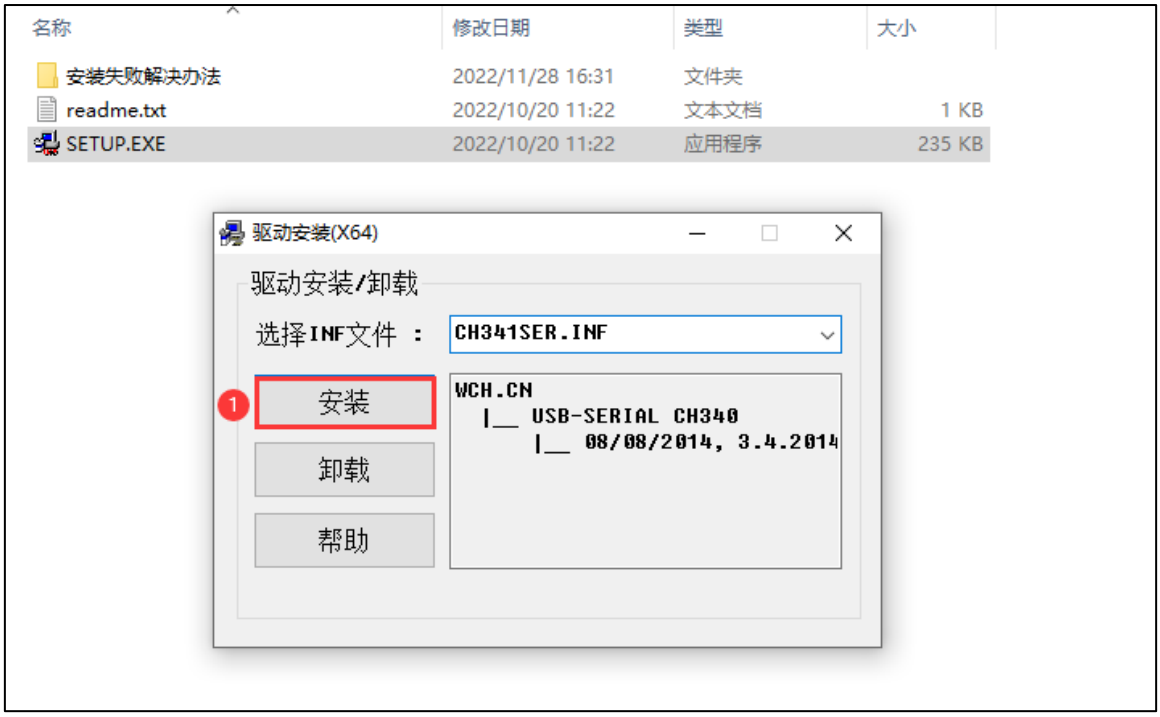


图 2.6.1 安装 CH340 驱动(1)

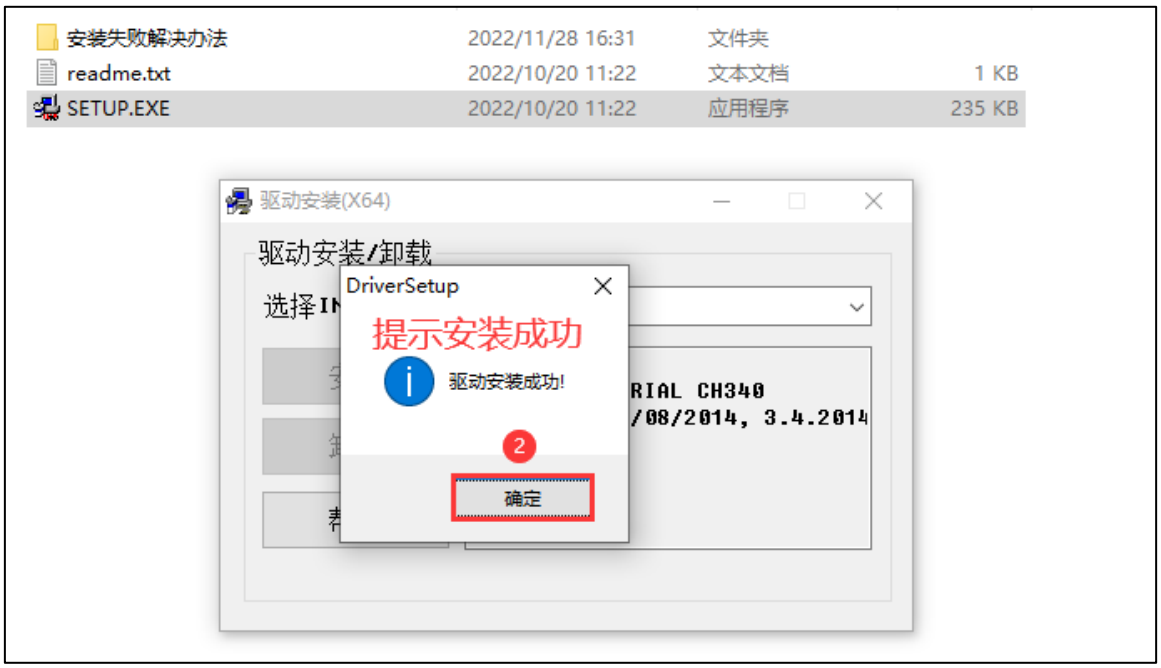


图 2.6.2 安装 CH340 驱动(2)

安装成功后，通过 USB 线将开发板调试串口与电脑相连，连接方式如图 2.6.3 所示：

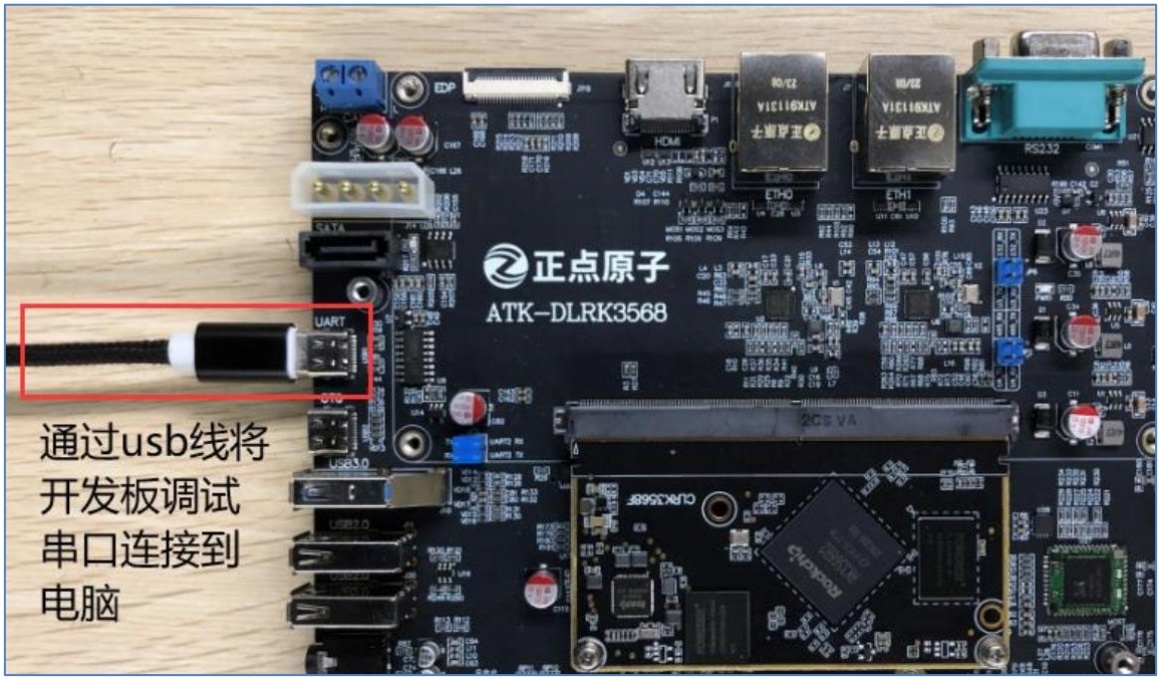


图 2.6.3 开发板调试串口与电脑相连

连接成功后，此时电脑会检测到一个 USB 设备，打开 Windows 设备管理器：

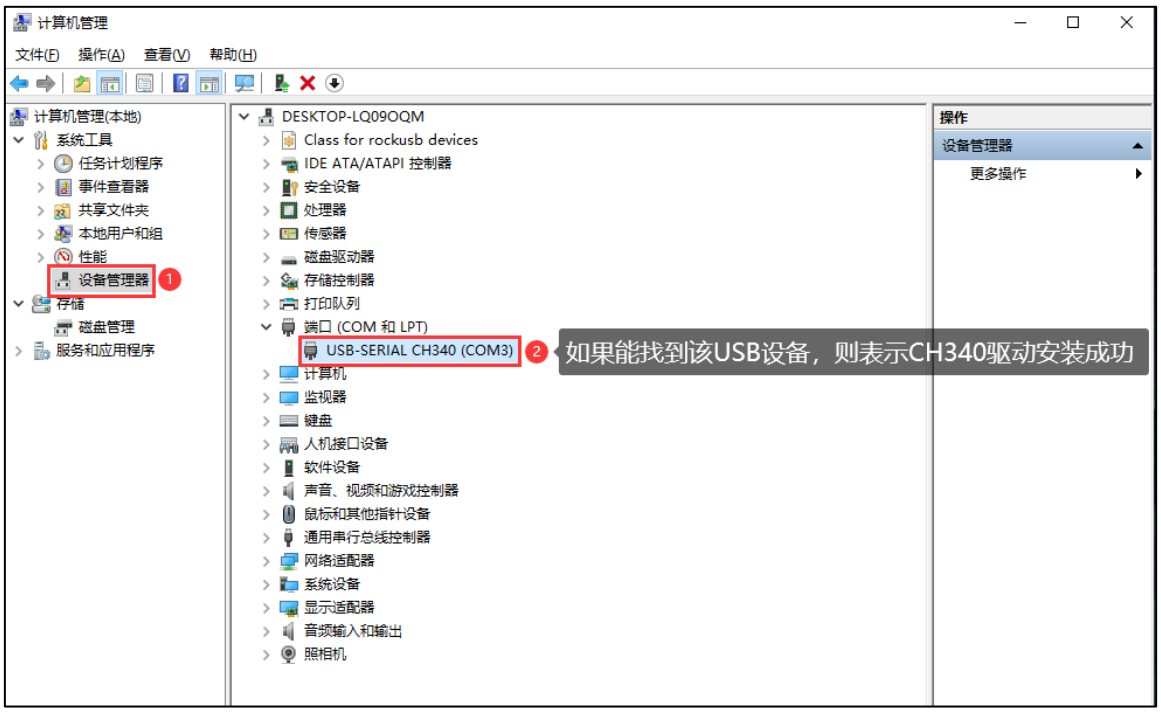


图 2.6.4 Windows 设备管理器

如果在“**设备管理器→端口（COM 和 LPT）**”下能找到一个名为“**USB-SERIAL CH340**”的设备，则表示 CH340 驱动安装成功！如果找不到名为“**USB-SERIAL CH340**”的设备，请用户自行检查硬件连接是否有误！可拔掉 USB 线、重新连接，或者连接电脑的其它 USB 口试试。

2.7 MobaXterm 软件安装

MobaXterm 是一款多功能远程终端软件，功能强大、而且免费（也有收费版本），支持创建 SSH、Telnet、Rsh、Xdmc、RDP、VNC、FTP、SFTP、串口(Serial COM)等超多远程连接功能。MobaXterm 提供了人性化的操作界面，功能十分强大，所以推荐用户使用 MobaXterm 这款终端软件。

2.7.1 MobaXterm 软件下载

开发板资料包中已经给用户提供了 MobaXterm 软件安装包，路径为：**开发板光盘 A 盘-基础资料→04、软件→MobaXterm_Installer_v12.3.zip**；用户也可以通过链接地址：https://mobaxterm.mobatek.net/download.html，自己下载：

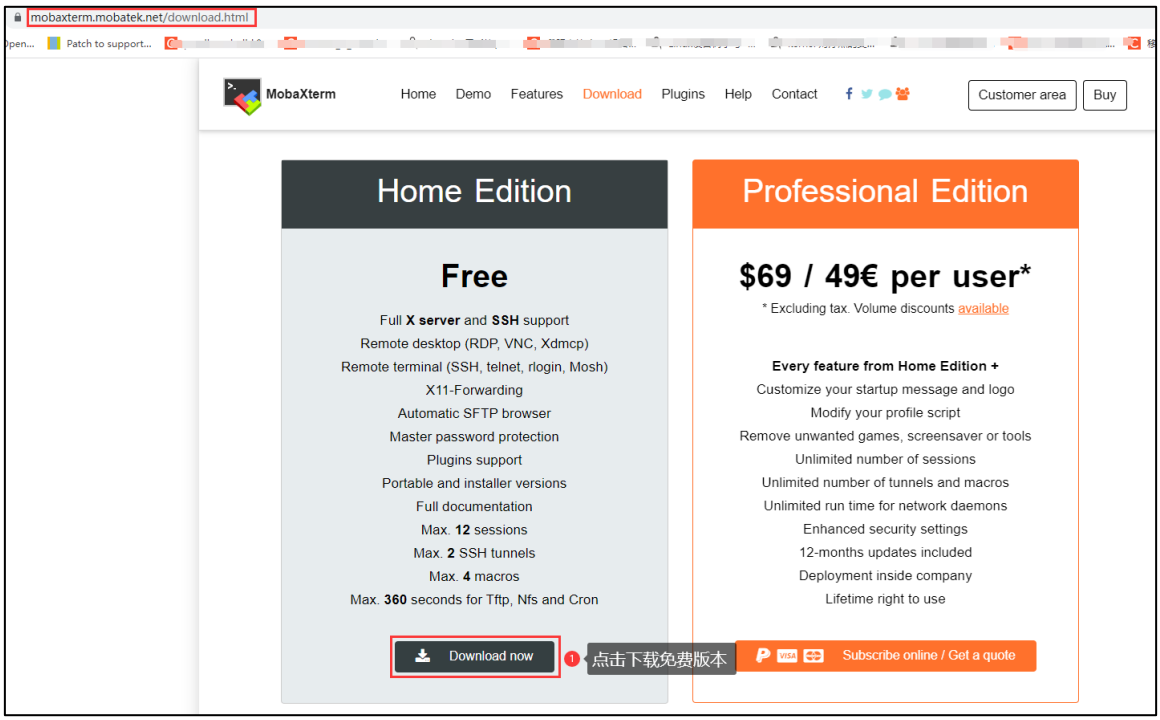


图 2.7.1.1 下载 MobaXterm 软件(1)

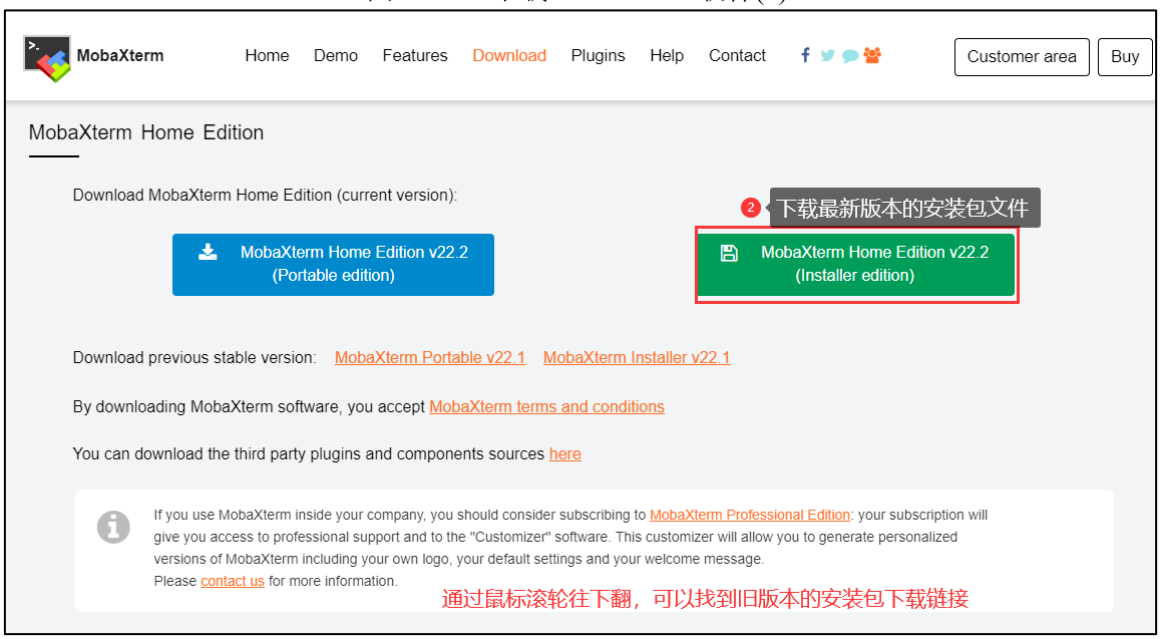


图 2.7.1.2 下载 MobaXterm 软件(2)

下载完成后会得到一个名为 MobaXterm_Installer_vxxx.zip 的压缩包文件（xxx 为版本号），目前最新版本为 22.2，资料包中给用户提供的安装包对应的版本为 12.3，用新的版本也行，旧的版本也可以，这个都没什么关系，这里我们以 12.3 版本为例。

2.7.2 MobaXterm 软件安装

将 MobaXterm_Installer_v12.3.zip 压缩包文件解压，解压之后如图所示：

图 2.7.2.1 MobaXterm_Installer_v12.3.zip 解压后的文件

接着双击运行 MobaXterm_Installer_v12.3.msi 文件，按照图 2.7.2.2~2.7.2.6 所示步骤安装 MobaXterm 软件：

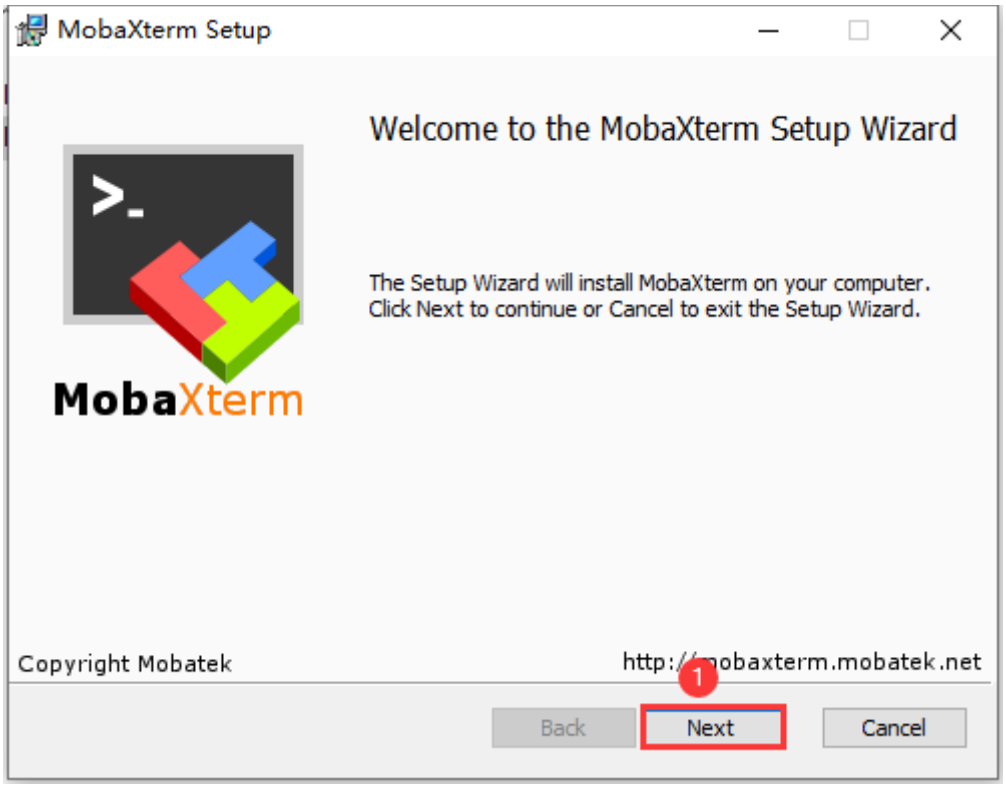


图 2.7.2.2 安装 MobaXterm 软件(1)

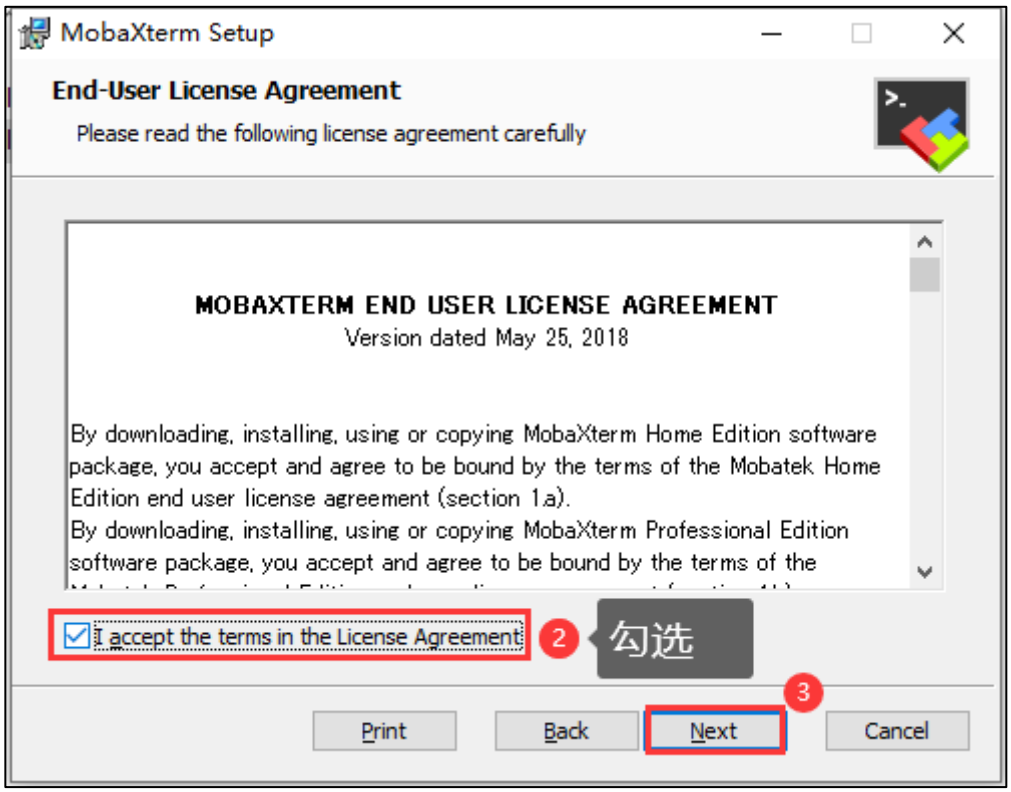


图 2.7.2.3 安装 MobaXterm 软件(2)

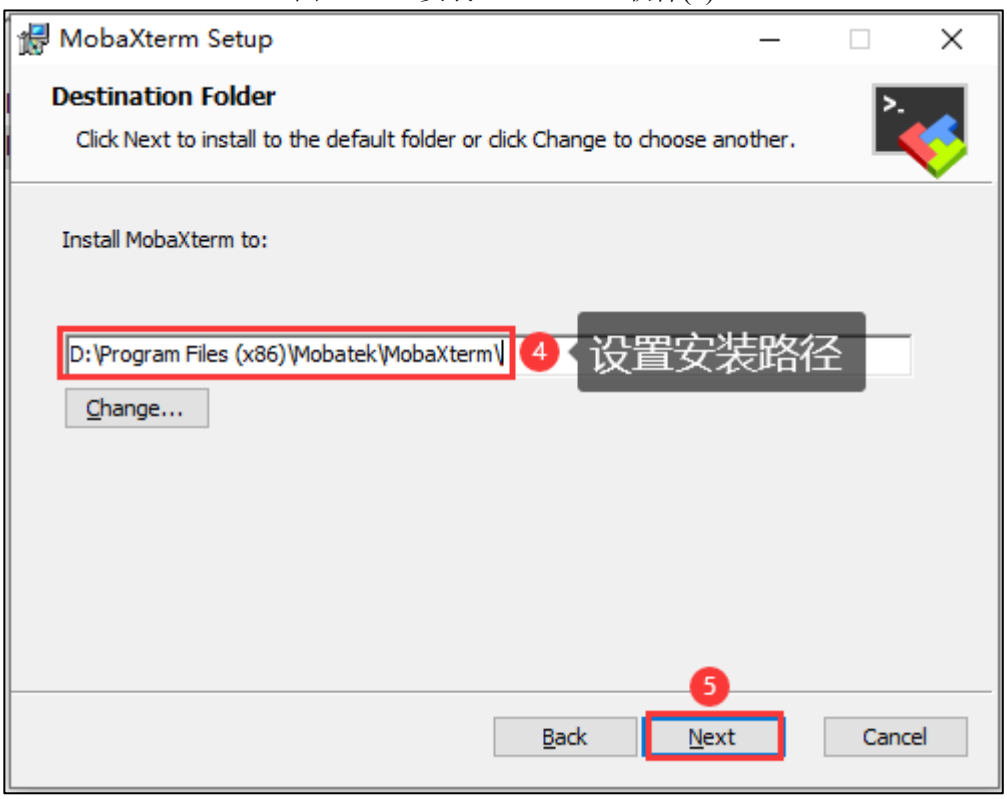


图 2.7.2.4 安装 MobaXterm 软件(3)

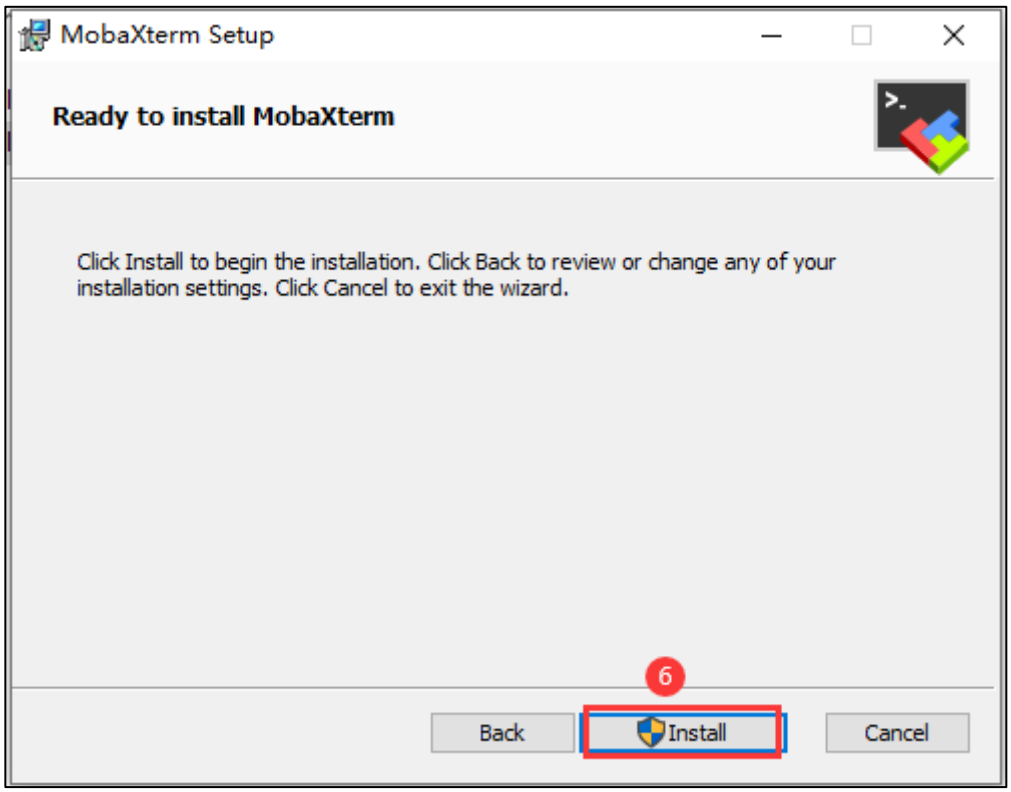


图 2.7.2.5 安装 MobaXterm 软件(4)

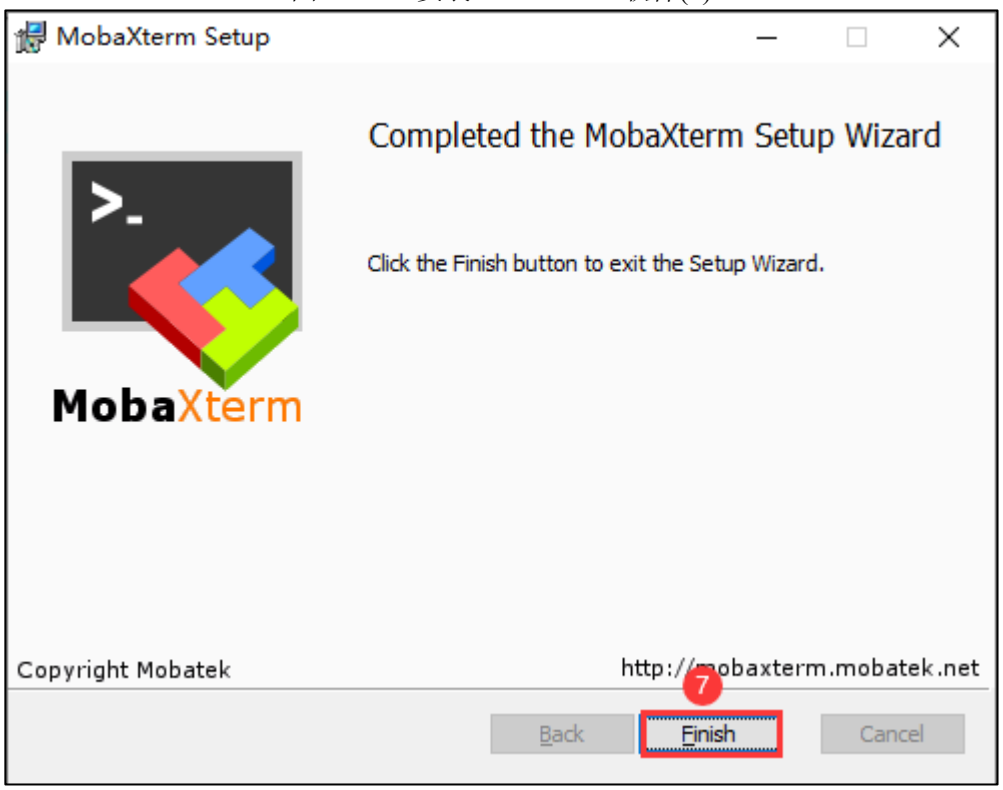


图 2.7.2.6 安装 MobaXterm 软件(5)

至此，软件安装完成，桌面会自动生成 MobaXterm 软件快捷方式图标：

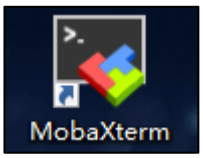


图 2.7.2.7 MobaXterm 软件桌面图标

2.7.3 MobaXterm 软件的使用

双击 MobaXterm 桌面图标打开该软件，如图 2.7.3.1 所示：

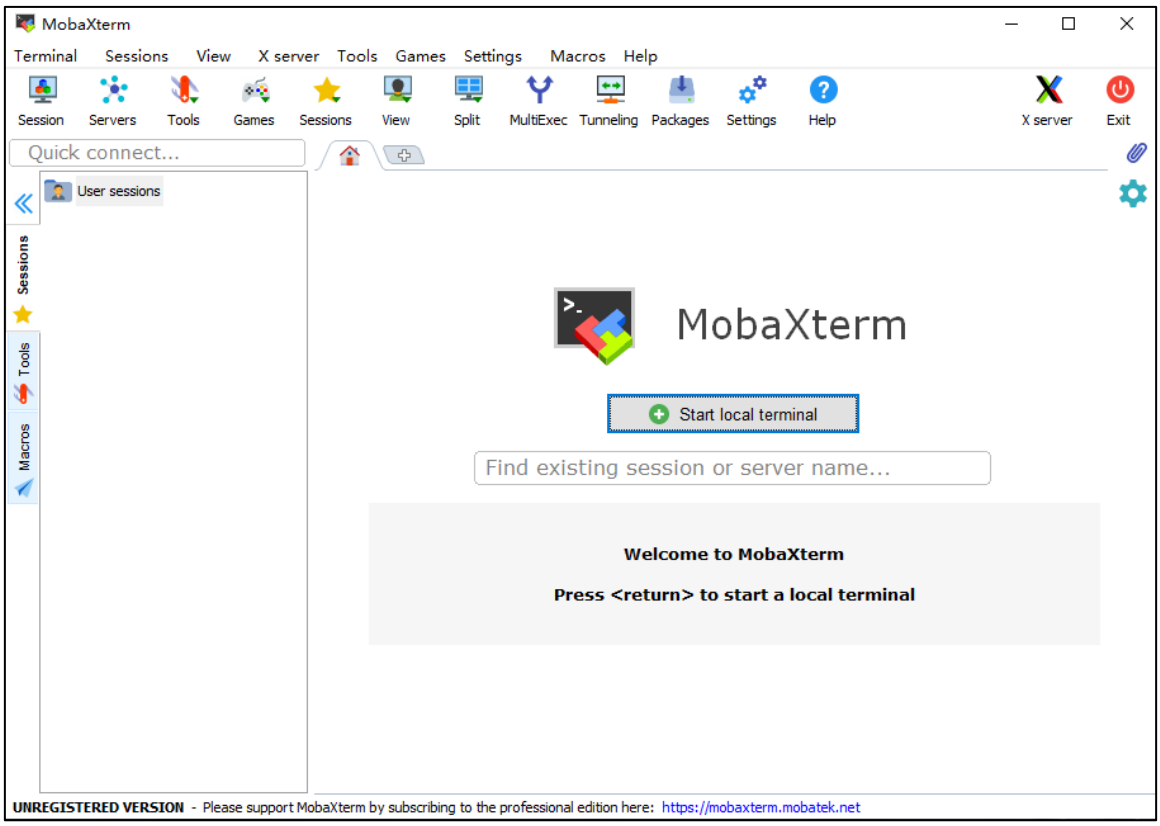


图 2.7.3.1 MobaXterm 软件主界面

接下来向大家介绍如何建立 Serial（串口）连接以及 ssh 远程连接。

1. 串口连接

ATK-RK3568 开发板上有一个调试串口，可直接通过 USB 线将其连接到电脑。串口作为嵌入式设备最为常见的通信接口之一，不但能实现计算机与嵌入式设备之间的数据传输，而且还能实现计算机对嵌入式设备的控制，嵌入式开发过程中，通常将其作为调试接口（串口调试），用于调试嵌入式设备。

按照图 2.7.3.2~2.7.3.3 所示操作步骤建立一个 Serial（串口）连接（在建立连接之前，需要通过 USB 线将开发板的调试串口与电脑相连、并且已经安装了 CH340 驱动）：

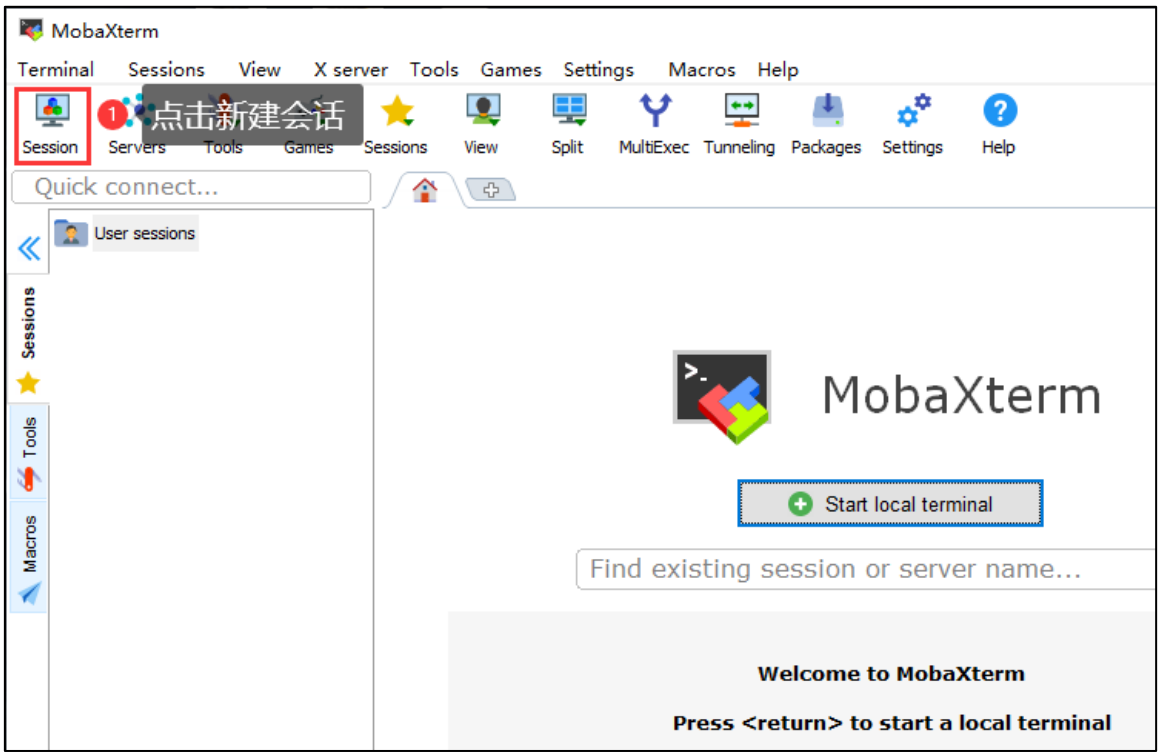


图 2.7.3.2 建立串口连接(1)

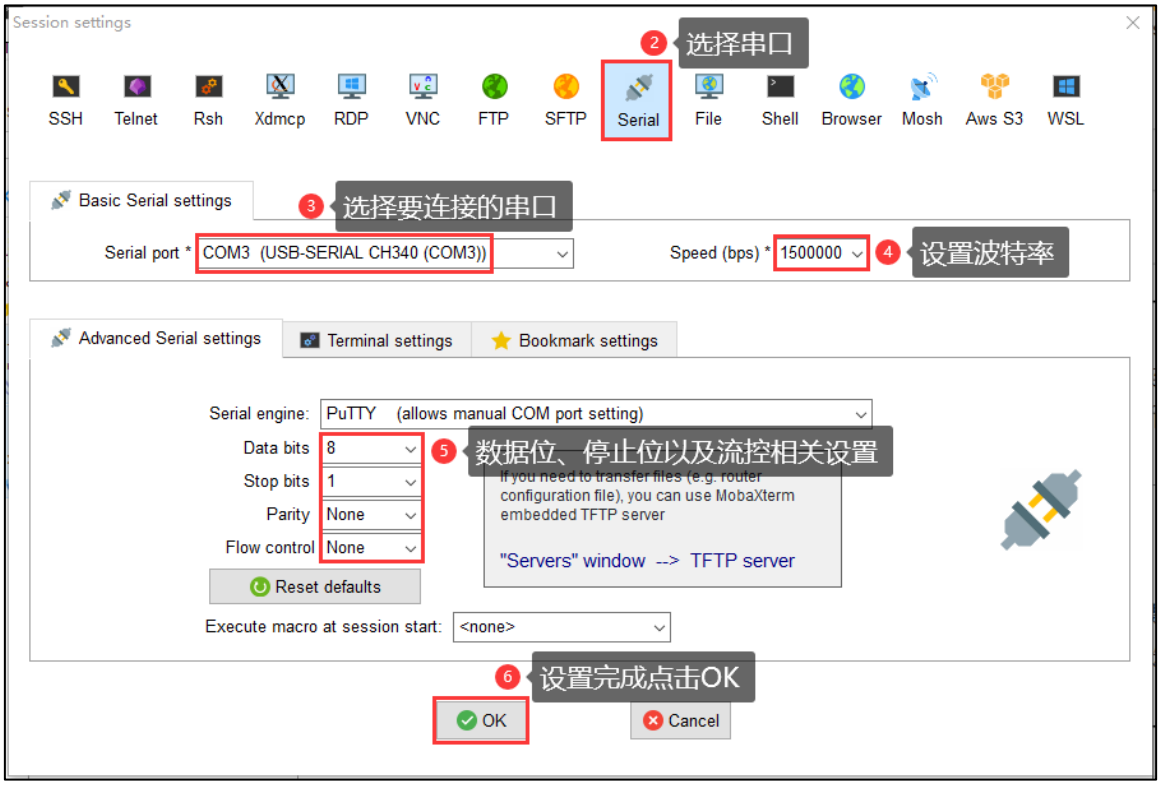


图 2.7.3.3 建立串口连接(2)

首先选择需要进行连接的串口，确保开发板的调试串口与电脑已经通过 USB 线相连、并且 CH340 驱动已经安装成功（“**设备管理器→端口（COM 和 LPT）**”下能找到一个名为“**USB-SERIAL CH340**”的设备），那么 MobaXterm 软件才能识别到开发板的调试串口，我们便可以在“**Serial port**”下拉列表中找到开发板对应的串口（**USB-SERIAL CH340**），然后选择它即可！

接着设置串口通信波特率，根据实际情况进行设置，RK3568 平台的调试串口，其默认波特率为 1500000（15M）；然后设置数据位、停止位以及流控等，设置完成后点击“**OK**”按钮。

Serial 连接就建立成功了，如图 2.7.3.4 所示：

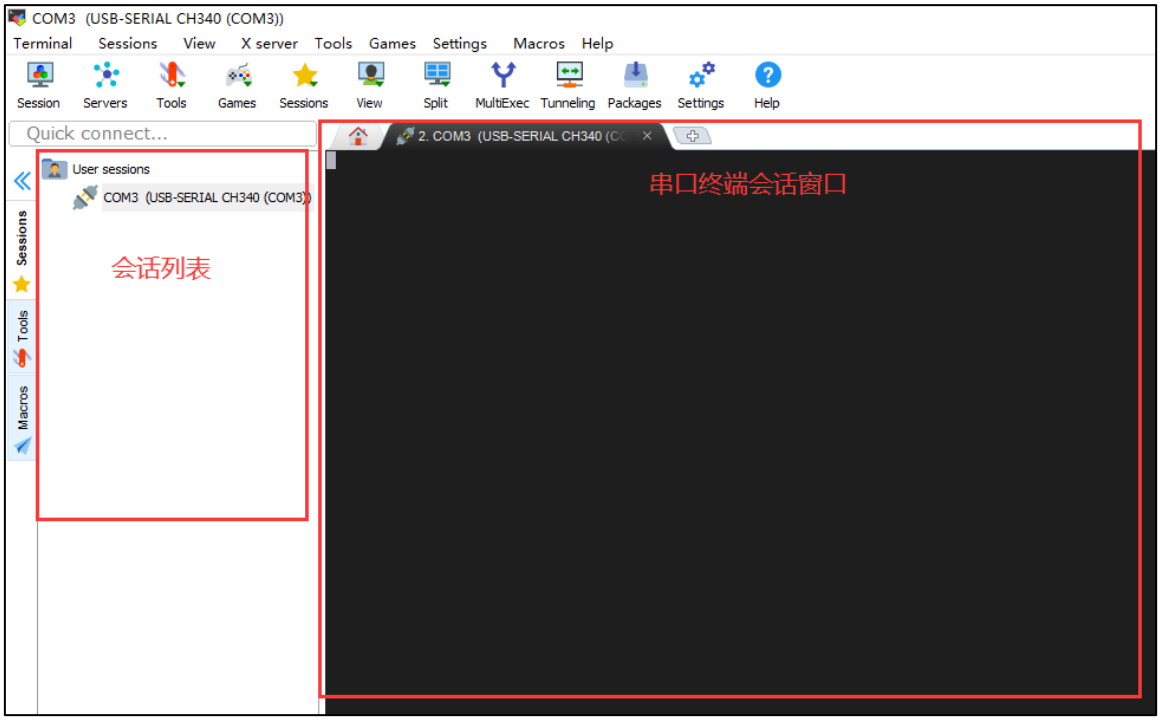


图 2.7.3.4 建立串口连接(3)

开发板上电启动，运行 Linux 系统或 Android 系统，系统启动过程中的 log 信息将会在串口终端会话窗口显示出来，如图 2.7.3.5 所示；

```
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
```

图 2.7.3.5 系统启动 log 信息

系统启动成功后，用户可以通过串口终端执行命令、命令执行结果也会通过串口终端显示出来。

2. ssh 连接

接下来介绍如何建立 ssh 远程连接，通过 ssh 可以实现远程登录，譬如远程登录 Ubuntu 系统、远程登录开发板 Linux 系统；按照图 2.7.3.5~2.7.3.6 所示操作步骤建立 ssh 远程连接、实现远程登录操作（以远程登录 Ubuntu 系统为例）：

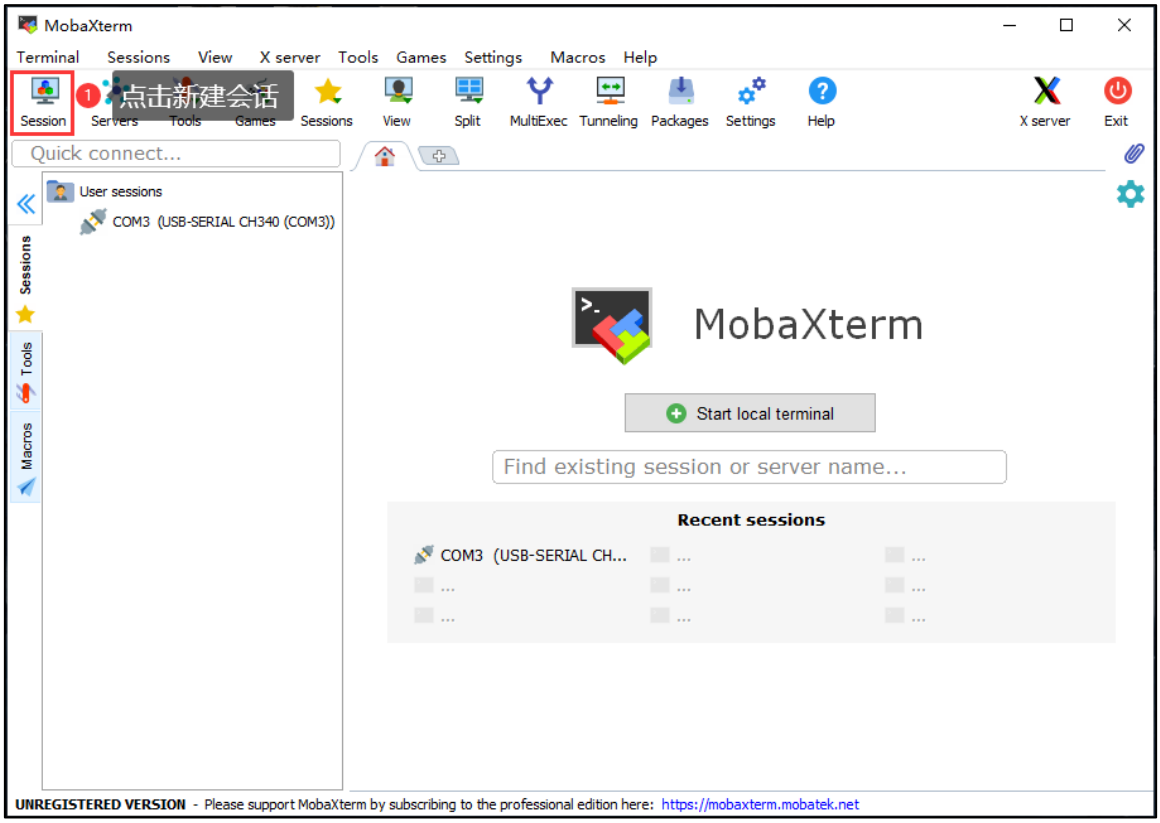


图 2.7.3.6 建立 ssh 远程连接(1)

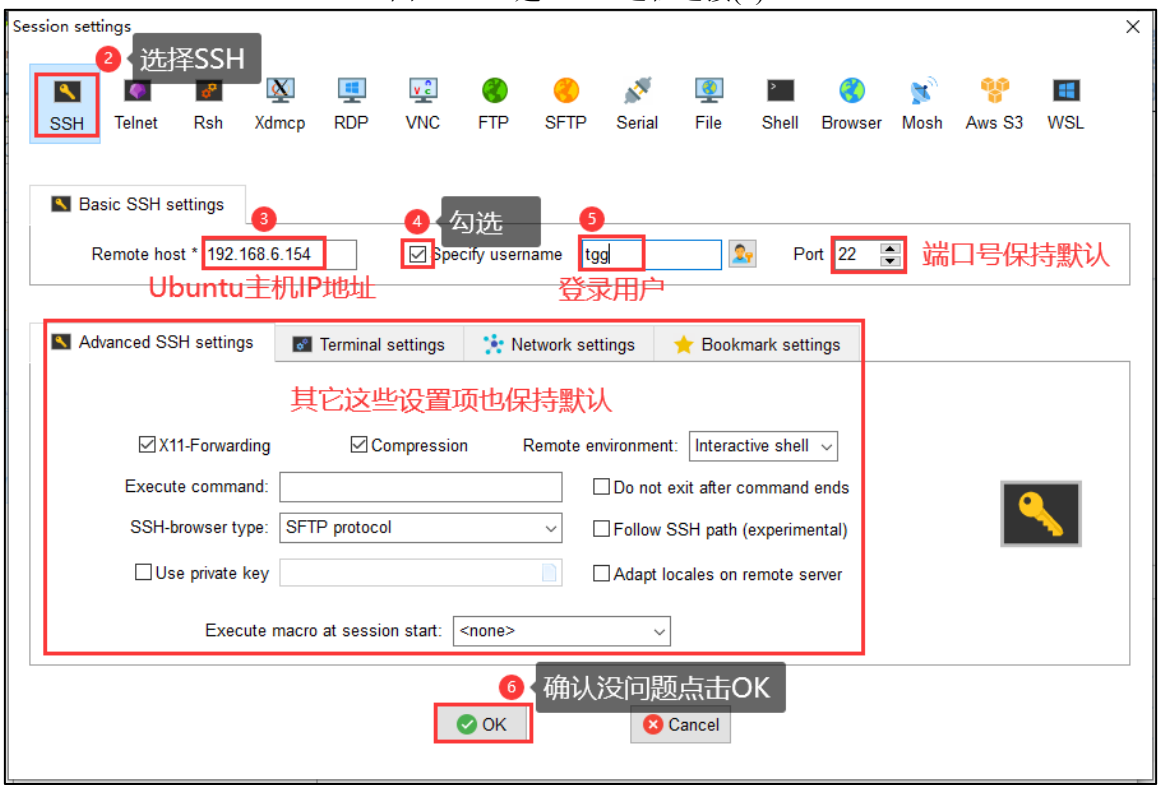


图 2.7.3.7 建立 ssh 远程连接(2)

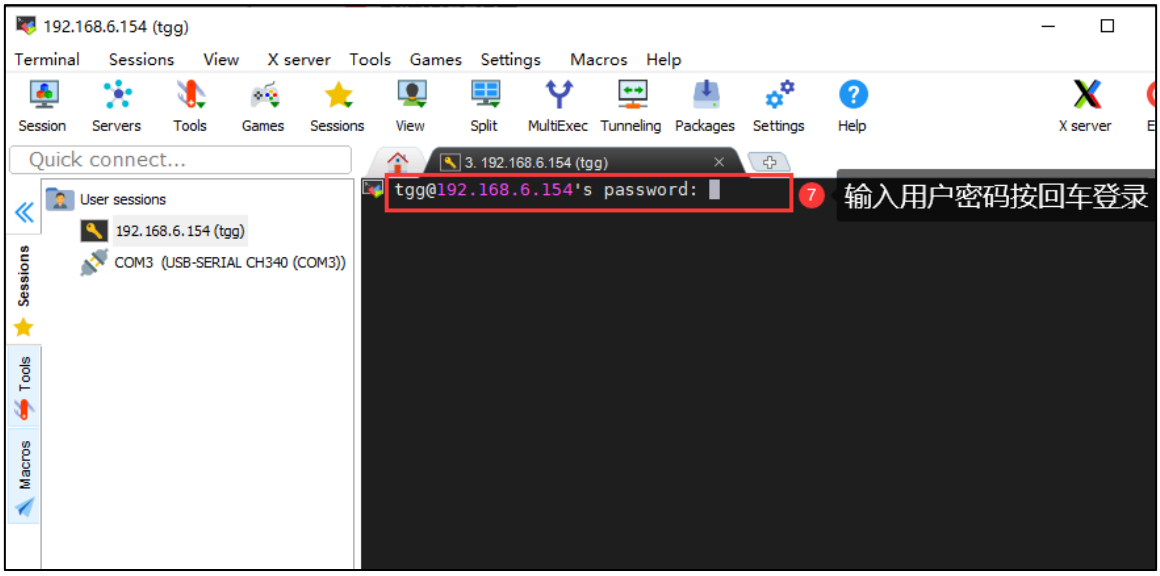


图 2.7.3.8 建立 ssh 远程连接(3)

图 2.7.3.9 建立 ssh 远程连接(4)

弹出窗口询问用户是否需要保存密码，我们可以选择“**Yes**”保存密码、也可以选择“**No**”不保存，取决于个人。至此，ssh 远程连接就创建成功了，如图 2.7.3.9 所示：

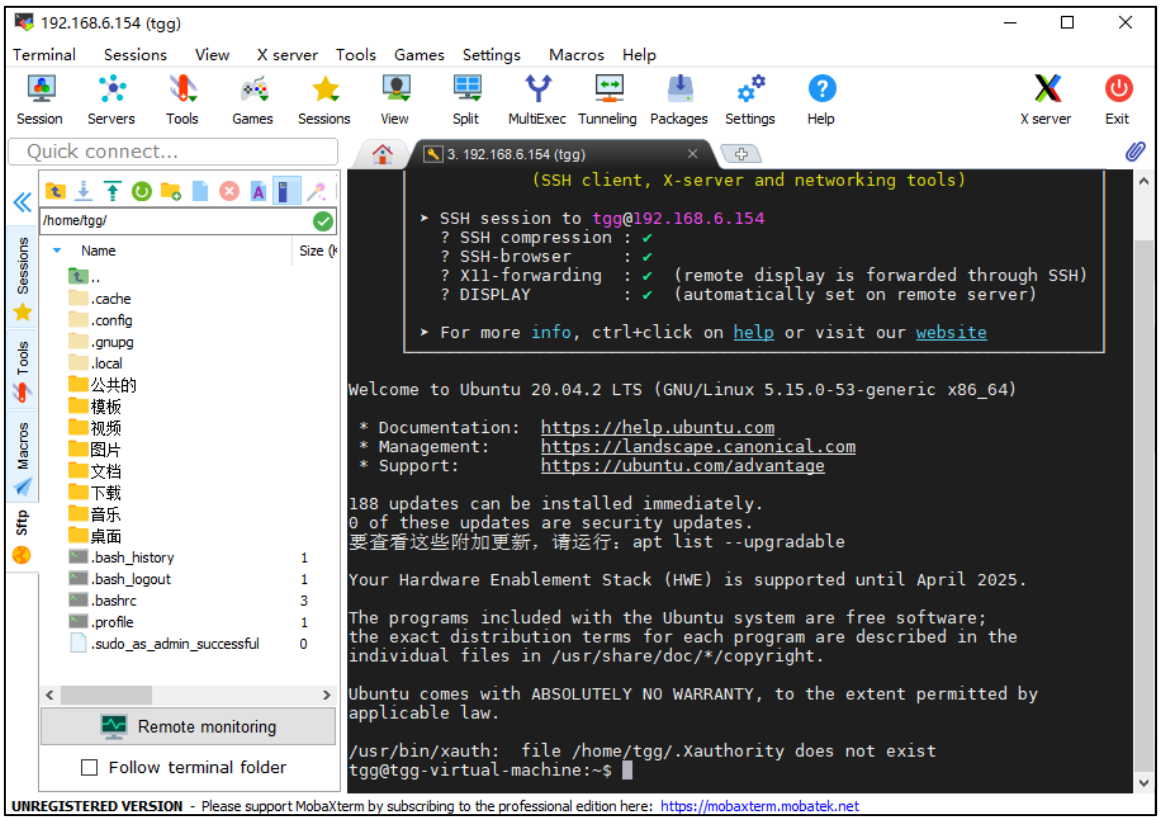


图 2.7.3.10 ssh 远程登录成功

在 ssh 远程登录的情况下，我们也可通过 MobaXterm 软件来实现 Windows 与 Ubuntu 之间的文件互传（通过 sftp 协议进行文件传输），具体方式如图 2.7.3.10 所示：

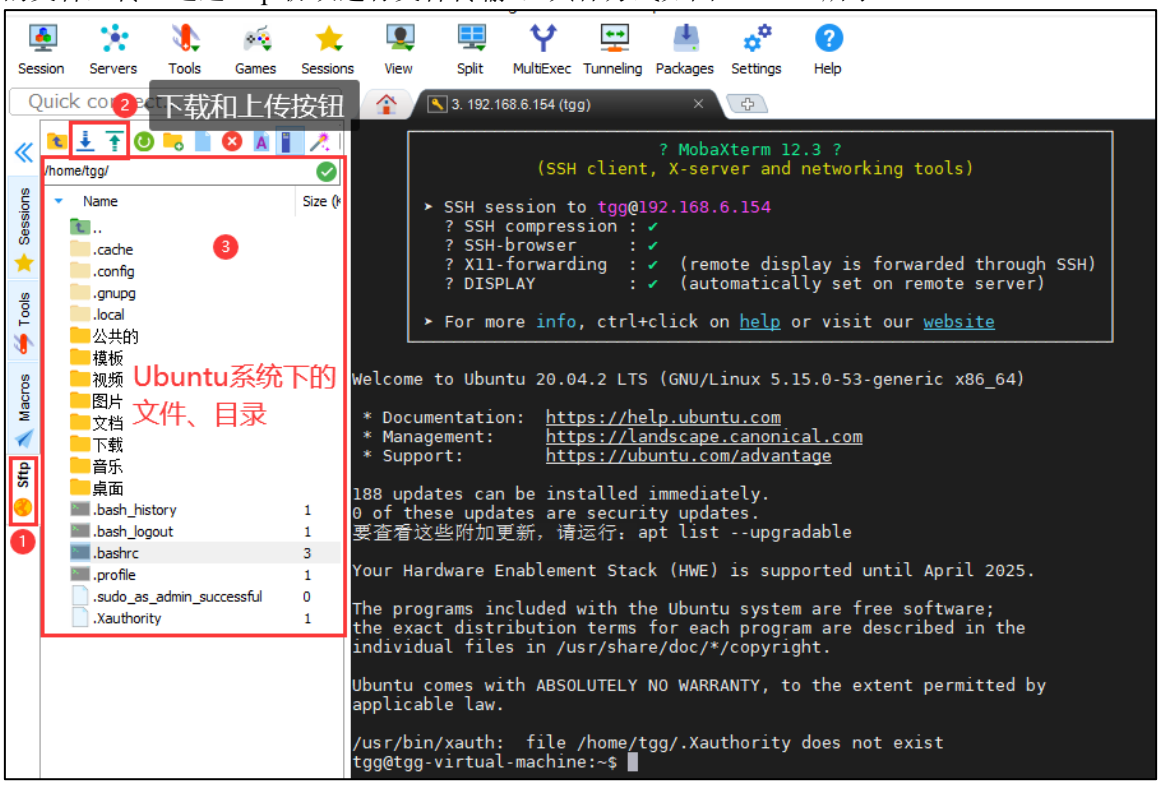


图 2.7.3.11 MobaXterm 实现 Windows 与 Ubuntu 之间文件互传

首先需要选中①“**Sftp**”选项卡，通过“**下载**”按钮可以将 Ubuntu 系统下的文件拷贝到 Windows 系统，通过“**上传**”按钮可以将 Windows 系统下的文件拷贝到 Ubuntu 系统。也可

以直接使用鼠标左键拖动，③所示区域便是 Ubuntu 系统下文件、目录，可以直接使用鼠标左键将 Ubuntu 系统下的文件拖动至 Windows 系统、也可以将 Windows 系统下的文件拖动至 Ubuntu 系统。

2.8 Rockchip USB 驱动安装

在开发调试阶段，经常需要将设备切换至 **Loader** 模式或者 **Maskrom** 模式，譬如使用瑞芯微开发工具（RKDevTool）烧录镜像时（需连接 USB 线），设备需要工作在 Loader 模式或 Maskrom 模式，此时才可进行烧写；需要在 Windows 下安装 Rockchip USB 驱动才能识别到设备。

开发板资料包中已经给用户提供了 Rockchip USB 驱动安装包，路径为：**开发板光盘 A 盘-基础资料→05、开发工具→RKTools→windows→DriverAssitant_v5.12.zip**，支持 xp、win7_32、win7_64、win10_32、win10_64 等操作系统；将 DriverAssitant_v5.12.zip 压缩文件解压，解压之后得到如图 2.8.1 所示文件、目录列表：

图 2.8.1 DriverAssitant_v5.12.zip 解压

直接双击 DriverInstall.exe 文件，按照图 2.8.2~2.8.4 所示步骤安装 Rockchip USB 驱动：

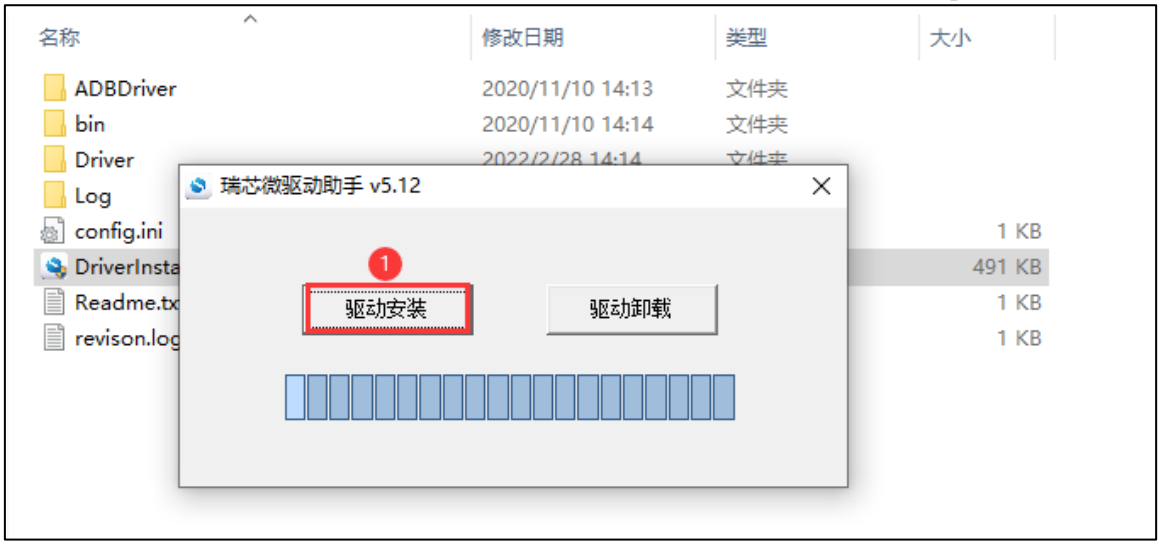


图 2.8.2 安装 Rockchip USB 驱动(1)

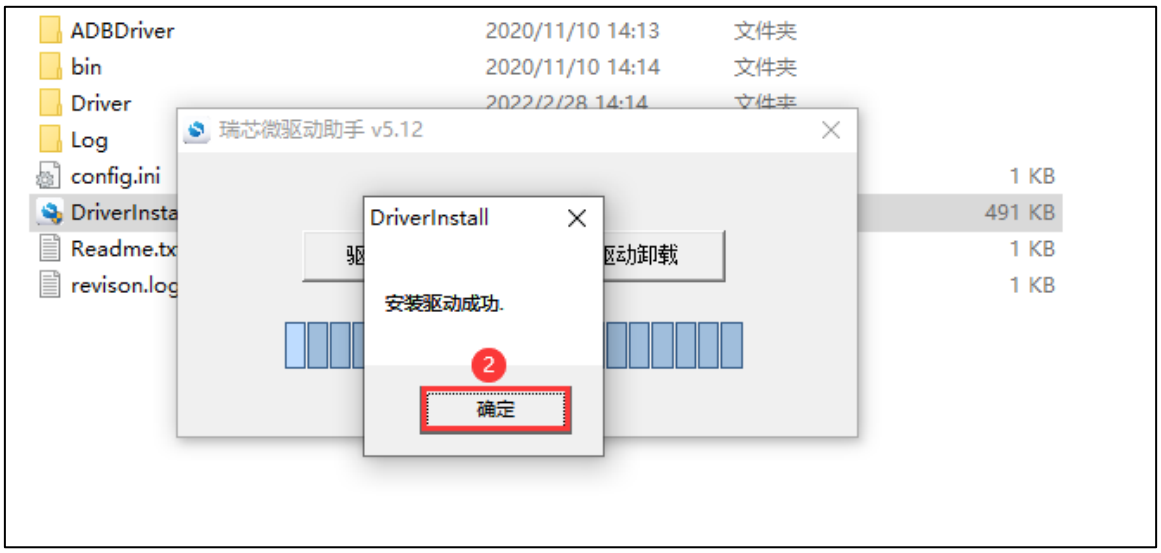


图 2.8.3 安装 Rockchip USB 驱动(2)

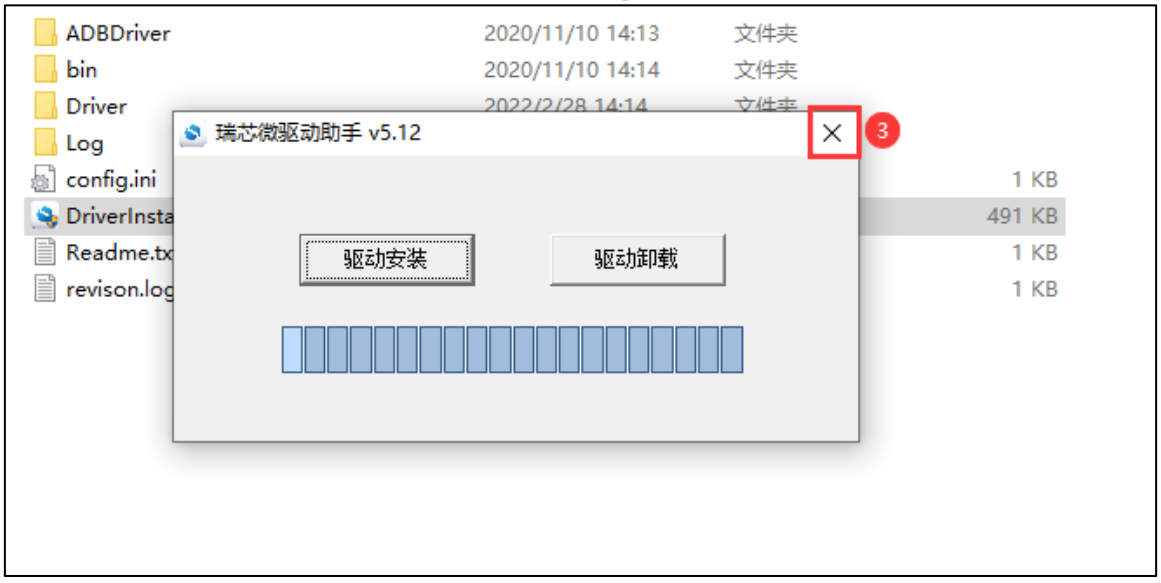


图 2.8.4 安装 Rockchip USB 驱动(3)

2.9 镜像烧录工具的使用

Rockchip 平台提供了多种镜像烧写方式，譬如在 **Windows 下通过 RKDevTool（瑞芯微开发工具）工具烧写、通过 SD 卡方式烧写、通过 FactoryTool 工具批量烧写（量产烧写工具、支持 USB 一拖多烧写）**以及在 **Ubuntu 系统下通过 Linux_Upgrade_Tool 工具烧写**等等，总之，烧写镜像的方式有很多种，用户可以选择合适的烧写方式进行烧写，将镜像文件烧写至开发板。

本小节简单介绍下 Windows 下的烧写工具 RKDevTool 以及 Linux 下的烧写工具 Linux_Upgrade_Tool，在我们开发调试过程中，这两个应该是使用最多的烧写工具。

2.9.1 烧写模式介绍

Rockchip 平台硬件设备运行的几种模式如下表所示，只有当设备处于 Maskrom 以及 Loader 模式时（**必须要通过 USB 线将开发板的 OTG 口连接到电脑**），才能够烧写镜像，或对板上镜像进行更新操作。

模式	是否支持烧录	描述
Maskrom	支持	Flash 在未烧录镜像时，芯片会引导进入 Maskrom 模式，可以进行初次镜像的烧写；开发调试过程中若遇到无法

		正常进入 Loader 模式的情况，也可进入 Maskrom 模式烧写镜像。
Loader	支持	Loader 模式下可以进行镜像烧写、更新升级，可以通过烧写工具单独烧写某一个分区镜像文件，方便调试。
Recovery	不支持	通过引导 recovery 镜像进入 recovery 模式，recovery 模式主要作用是升级、恢复出厂设置类操作。
Normal Boot	不支持	系统正常启动进入该模式，通过引导 rootfs 启动，加载 rootfs 根文件系统，大多数的开发都是在这个模式下进行的。

表 2.9.1.1 Rockchip 平台运行模式说明

进入 Maskrom 烧写模式的方法：（先连接电源适配器和 OTG）

- 开发板未烧录过镜像，上电之后就会进入到 Maskrom 模式；
- 开发板烧录过镜像，按住开发板上的 **UPDATE 按键**，然后开发板上电或复位，系统将会进入到 Maskrom 模式；
- 开发板烧录过镜像，按住开发板上的 **V-按键**（音量-）按键，然后开发板上电或复位，系统将会进入到 Maskrom 模式；
- 开发板烧录过镜像，在 U-Boot 命令行下，执行“**rbrom**”命令进入到 Maskrom 模式。

进入 Loader 烧写模式的方法：（先连接电源适配器和 OTG）

- 开发板烧录过镜像，按住 **V+按键**（音量+）按键，然后开发板上电或复位，系统将会进入到 Loader 模式；
- 开发板烧录过镜像，在 U-Boot 命令行下，执行“**download**”命令进入到 Loader 模式；
- 开发板烧录过镜像，在 Linux 系统下，通过串口或 ADB 执行命令“**reboot loader**”重启进入到 Loader 模式；
- 开发板烧录过镜像，上电或复位后开发板正常启动进入到系统后，瑞芯微开发工具（也就是 RKDevTool 工具）会显示“**发现一个 ADB 设备**”，然后点击“**切换**”按钮，进入到 Loader 模式，如下图。

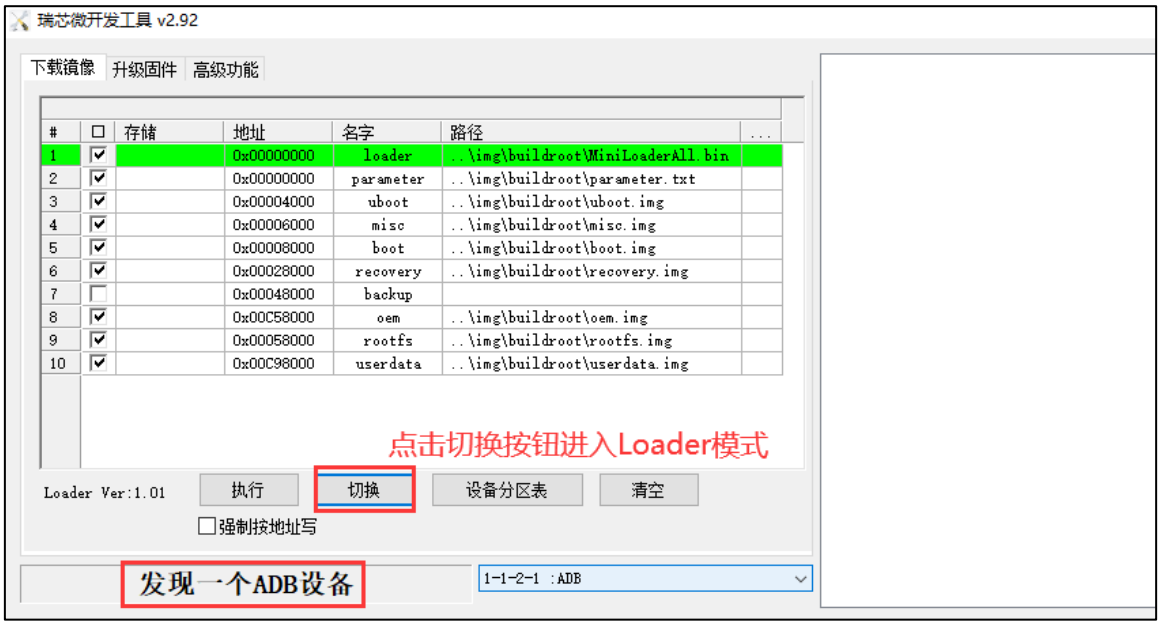


图 2.9.1.1 通过瑞芯微开发工具进入 Loader 模式

以上给大家介绍了开发板如何进入到 Maskrom 或 Loader 烧写模式，只有进入 Maskrom 模式或 Loader 模式后才可进行烧录。

2.9.2 Windows 下 RKDevTool 工具的使用

在 Windows 下烧写镜像使用 RKDevTool 工具，RK 的文档中一般也将其称为瑞芯微开发工具，该工具除了用于烧录镜像之外，还有其它的一些功能，是我们开发、调试过程中最常用的工具。

正点原子 ATK-DLRK3568 开发板资料包中已经给用户提供了该工具，路径为：**开发板光盘 A 盘-基础资料→05、开发工具→RKTools→windows→RKDevTool_Release_v2.92.zip**，当然该工具在 RK3568 Linux SDK 中也可以找到。

首先将 RKDevTool_Release_v2.92.zip 压缩文件解压开来，解压之后的内容如下所示：

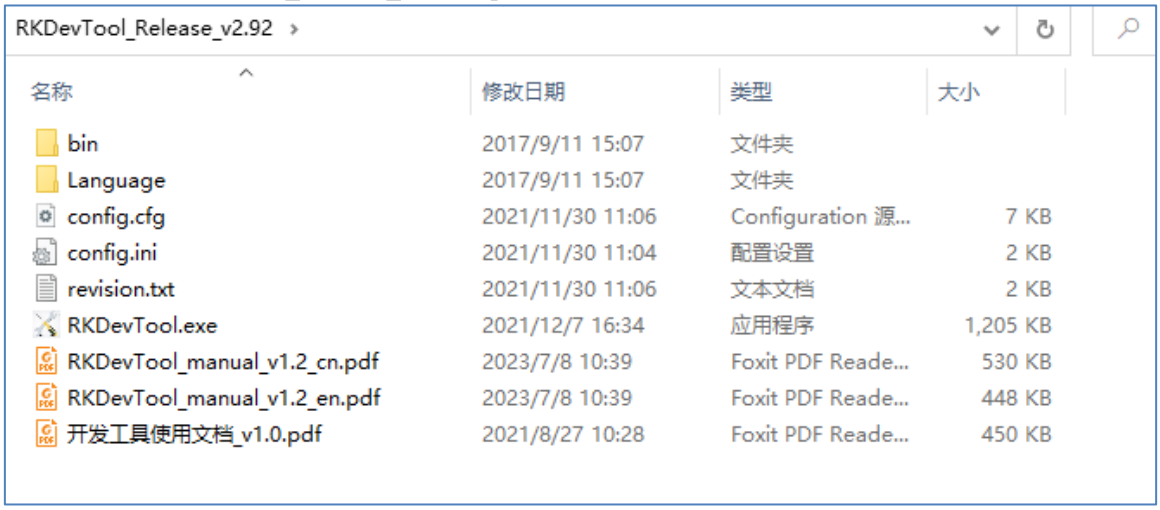


图 2.9.2.1 RKDevTool 目录

该目录下有两份文档：《**开发工具使用文档_v1.0.pdf**》和《**RKDevTool_manual_v1.2_cn.pdf**》，这两份文档向用户介绍了如何使用 RKDevTool 工具（瑞芯微开发工具），它们由 RK 官方提供；关于 RKDevTool 工具的详细使用方法请参考这两份文档，本文档只做一个简单的介绍。

双击 **RKDevTool.exe** 可执行文件打开瑞星微开发工具，如下图所示：

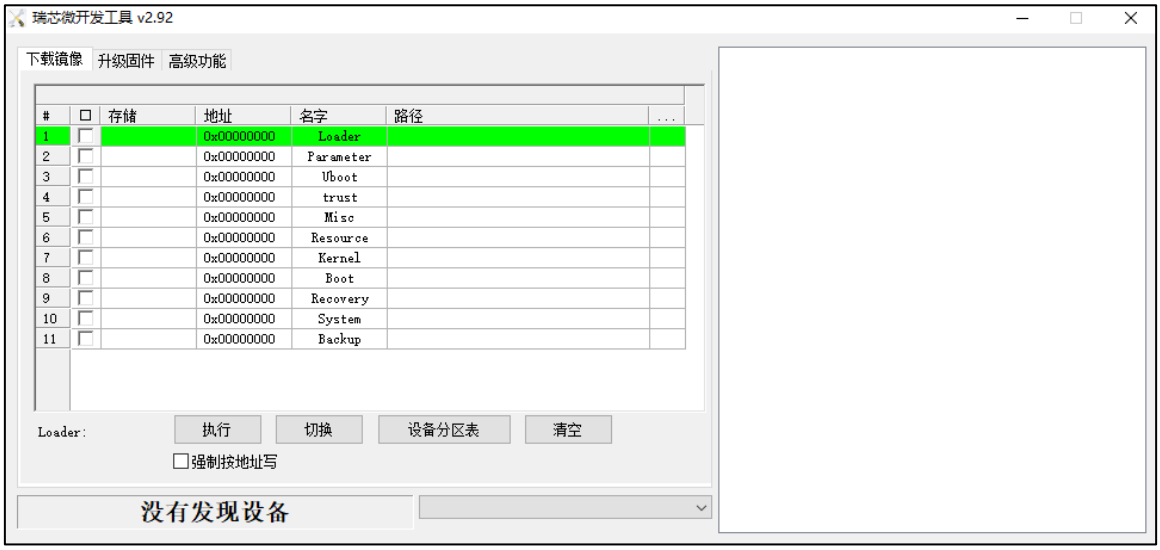


图 2.9.2.4 瑞星微开发工具界面

上图便是瑞芯微开发工具的一个主界面，RKDevTool 工具有两种镜像烧录方式：

- **单独烧录各镜像**。RK3568 Linux SDK 编译成功会生成多个镜像文件，包括：boot.img、MiniLoaderAll.bin、uboot.img、misc.img、rootfs.img、recovery.img、userdata.img、oem.img 共计 8 个镜像文件，其中还包括一个分区表文件 parameter.txt；使用 RKDevTool 工具可单独烧录这些镜像。

- **烧录完整 update.img 固件**。update.img 并非编译得到，它其实是由多个镜像打包而成，包括 boot.img、MiniLoaderAll.bin、uboot.img、misc.img、rootfs.img、recovery.img 等，是这些分立镜像的集合体，通过 RK 提供的工具可将各个分立镜像打包成一个 update.img 固件包。

接下来讲一下如何烧录各个分立镜像，2.9.3 小节将向用户介绍如何烧录 update.img 固件。

按照图 2.9.2.5~2.9.2.8 所示操作步骤，导入一个.cfg 配置文件（配置文件中包含了烧录时的配置信息，包括镜像烧录地址、镜像所在路径等信息，当然也可自己手动配置这些信息，后面会讲）：

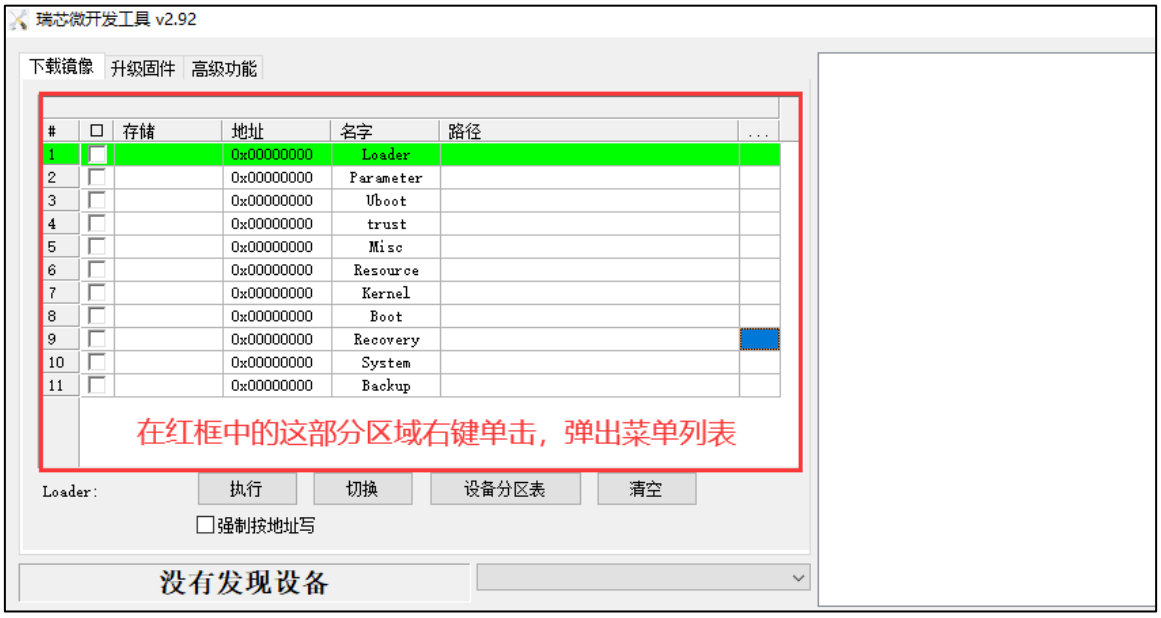


图 2.9.2.5 导入配置文件(1)

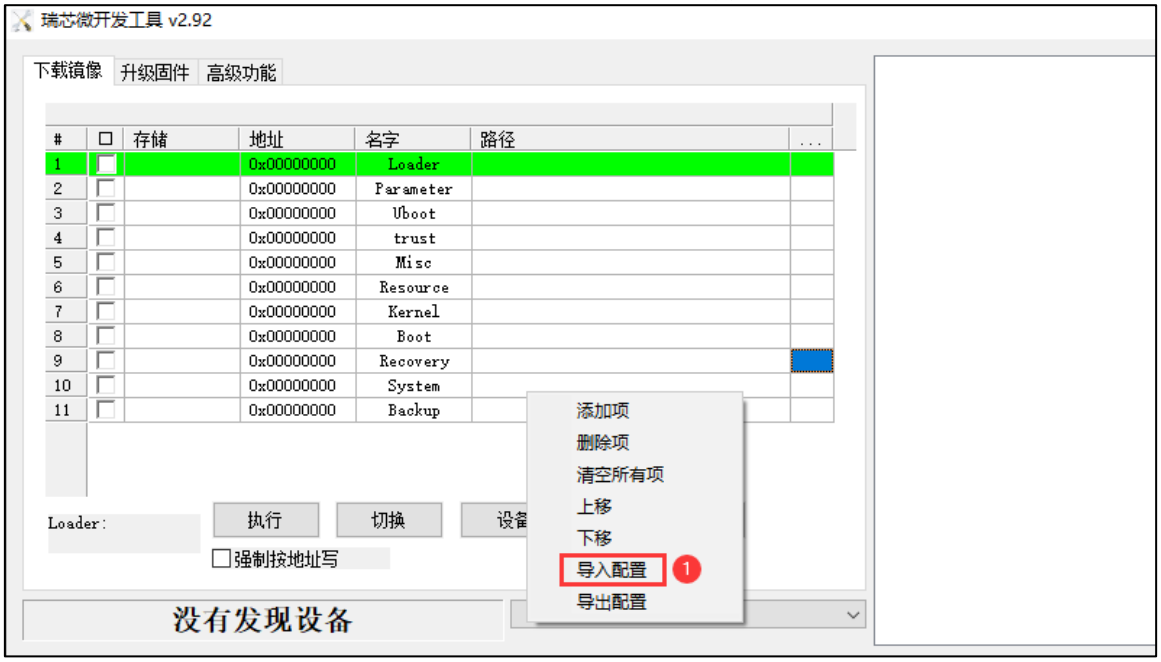


图 2.9.2.6 导入配置文件(2)

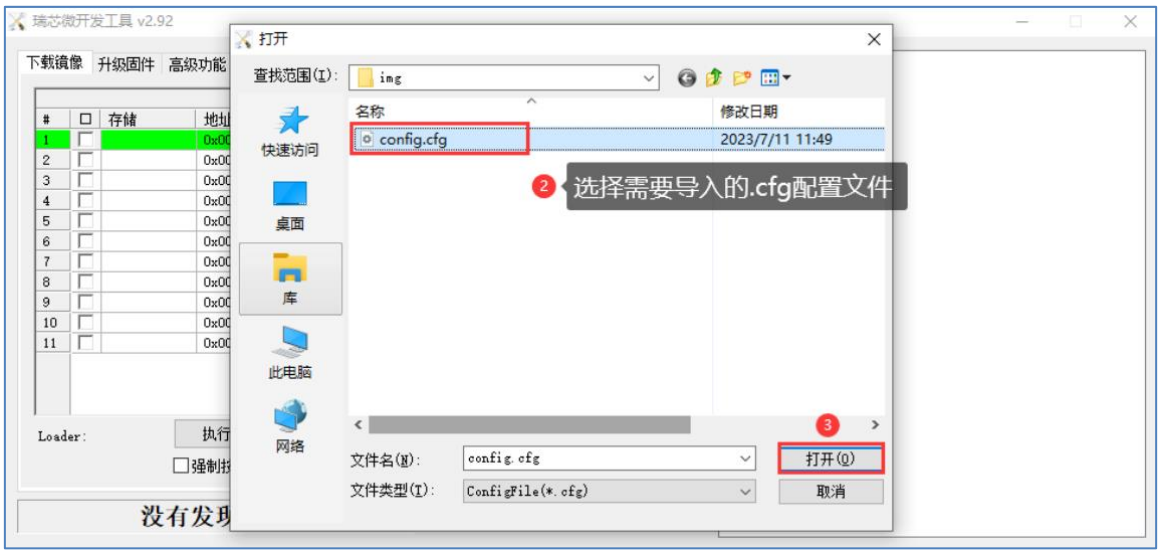


图 2.9.2.7 导入配置文件(3)

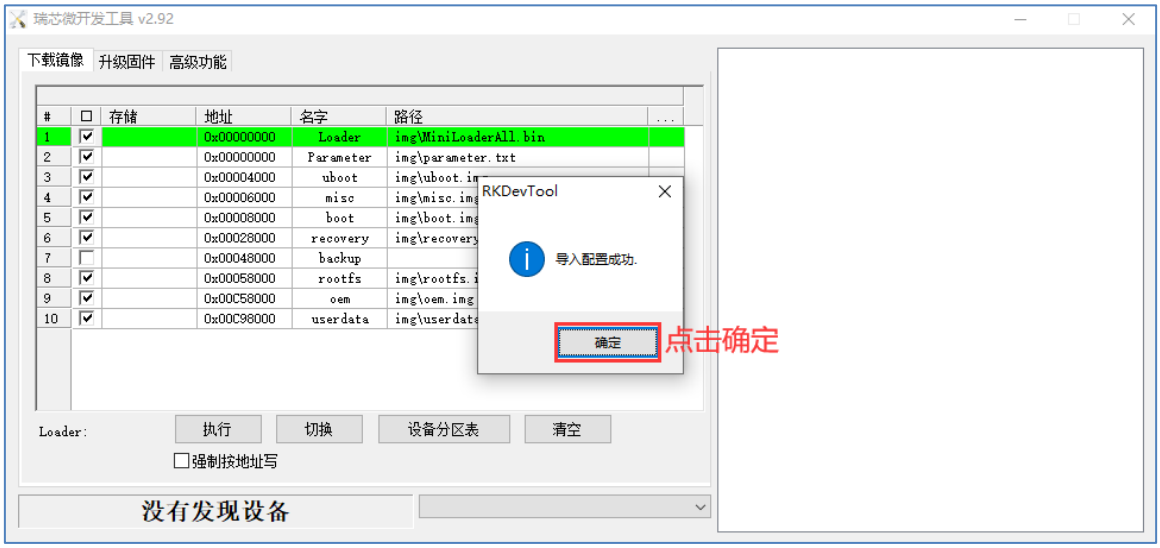


图 2.9.2.8 导入配置文件(4)

至此，我们就导入了一个烧录配置文件（导入配置文件）。

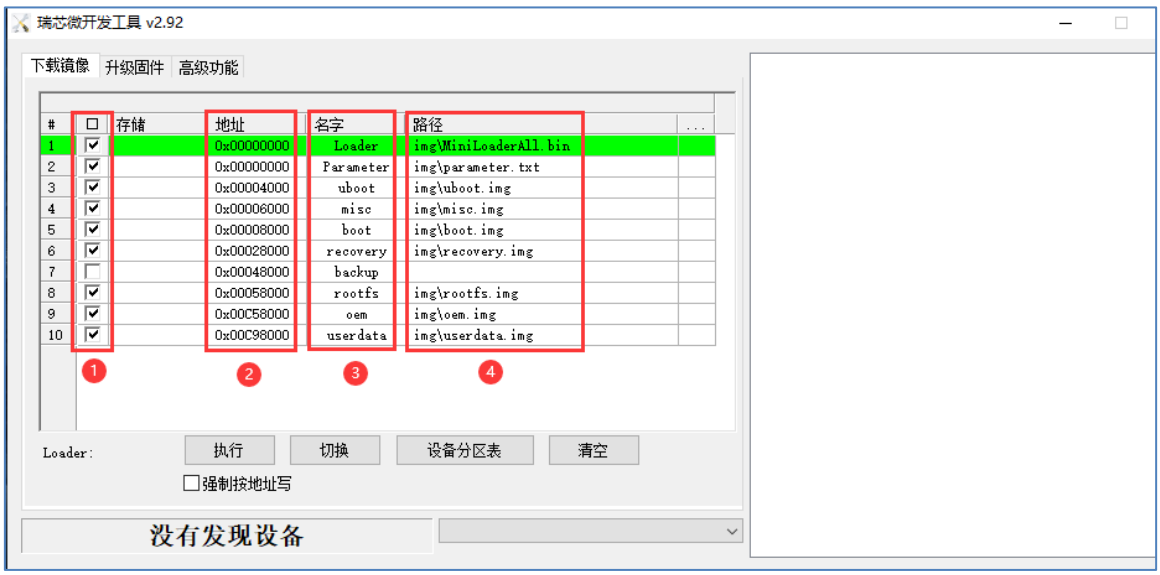


图 2.9.2.9 配置信息划分说明

①部分用于控制是否烧写对应的镜像，勾选表示需要烧写、不勾选表示不烧写；

②部分表示烧写的地址（也就是镜像的烧录地址）；

③部分表示该烧录项的名字；

④部分用于指定镜像的所在路径。

这些配置信息都是可以手动进行更改的，改完之后可以重新导出到配置文件中进行保存，方便下次导入该配置。

导入配置后，接下来便可以进行烧写了。执行烧写操作之前，**开发板必须处于 Maskrom 模式或 Loader 模式，客户拿到手的开发板默认都是烧录过镜像的，可以进入到 Loader 模式，当然也可以进入到 Maskrom 模式。首先开发板需要先连接电源适配器以及 OTG 口**，如下图所示：

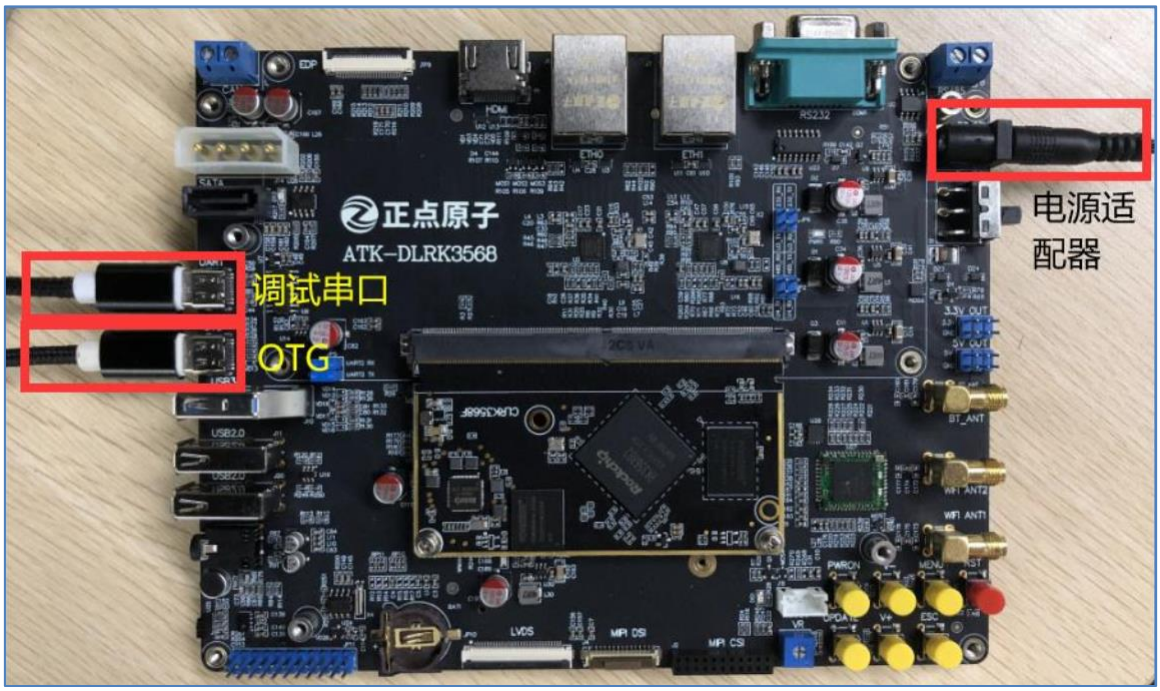


图 2.9.2.10 开发板硬件连接示意图

建议使用 ATK-DLRK3568 开发板配套的 12-2.5A 电源适配器；通过 USB 线（一端为 USB 接口、另一端为 Type-C 接口）将开发板的 UART 调试串口连接到电脑，同样使用 USB 线（一端为 USB 接口、另一端为 Type-C 接口）将开发板的 OTG 口连接到电脑。

硬件连接好之后，按住开发板的 **V+**（音量+）按键，然后开发板上电或复位，系统将会进入到 Loader 模式；此时瑞芯微开发工具便会显示“**发现一个 LOADER 设备**”字样：

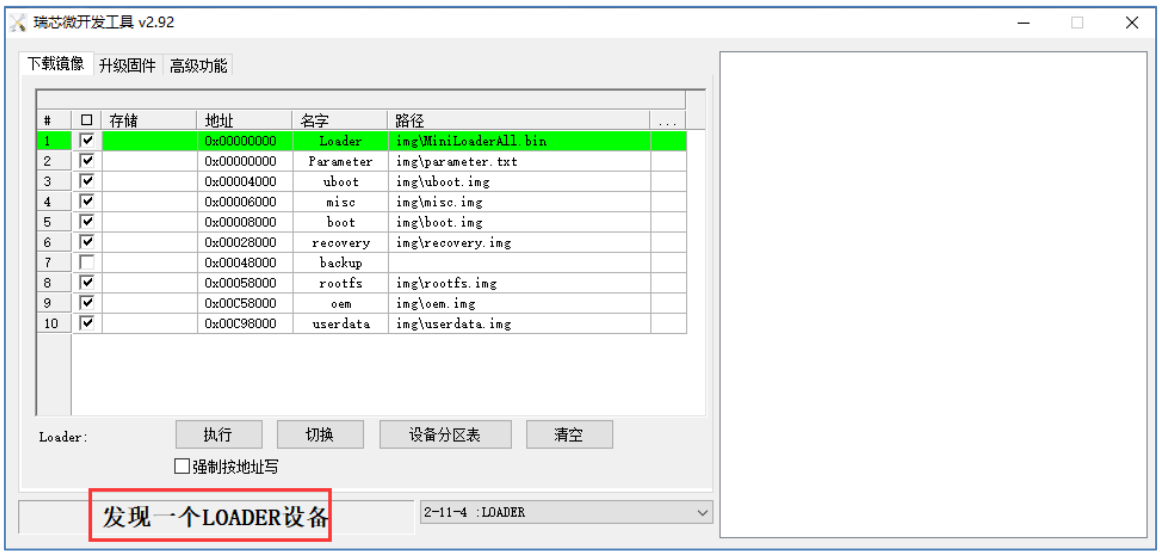


图 2.9.2.11 瑞芯微开发工具检测到 loader 设备

表示开发板当前处于 Loader 模式，直接点击“**执行**”按钮即可开始烧写镜像，如下所示：

图 2.9.2.12 执行烧写操作

当然也可以在 Maskrom 模式下烧写，2.9.1 小节中已经介绍了如何进入 Maskrom 模式，进入 Maskrom 模式后，同样也是点击“**执行**”按钮进行烧写。

如果没有出现意外，那么烧写将会成功，**也就意味着这些镜像已经成功烧写到了开发板 Flash 存储器中（ATK-DLRK3568 开发板使用了 eMMC 作为板载存储器，所以默认情况下镜像会烧录到开发板 eMMC 中，然后可以通过 eMMC 启动开发板；当然也可以将镜像烧写到 SD 卡，通过 SD 卡启动）**；如果不幸，烧写过程中出现了错误、导致烧写失败，那么大家可以参考 RK 官方提供的文档《**RKDevTool_manual_v1.2_cn.pdf**》，该文档中有针对几种常见的错误进行说明。

烧录完成后，会自动重启开发板，通过 MobaXterm 软件连接开发板的调试串口，可以查看系统启动过程中的 log 信息。

2.9.3 update.img 镜像的烧录方法

update.img 是多个镜像的集合体（由多个镜像打包合并而成），使用 RK 提供的工具可以将各个分立镜像（譬如 uboot.img、boot.img、MiniLoaderAll.bin、parameter.txt、misc.img、rootfs.img、oem.img、userdata.img、recovery.img）打包成一个 update.img 固件，方便用户烧录、升级。

本小节介绍如何使用瑞芯微开发工具烧写 update.img，首先打开瑞芯微开发工具，选择“升级固件”：

图 2.9.3.1 升级固件选项卡

然后点击“固件”按钮选择我们需要进行升级、烧录的 update.img 固件（开发板资料包中并未给用户提供 update.img 固件，这里只是给用户演示烧写 update.img 固件的方法，后续在开发过程中再去实践操作），导入 update.img 固件之后会显示该固件的一些信息：

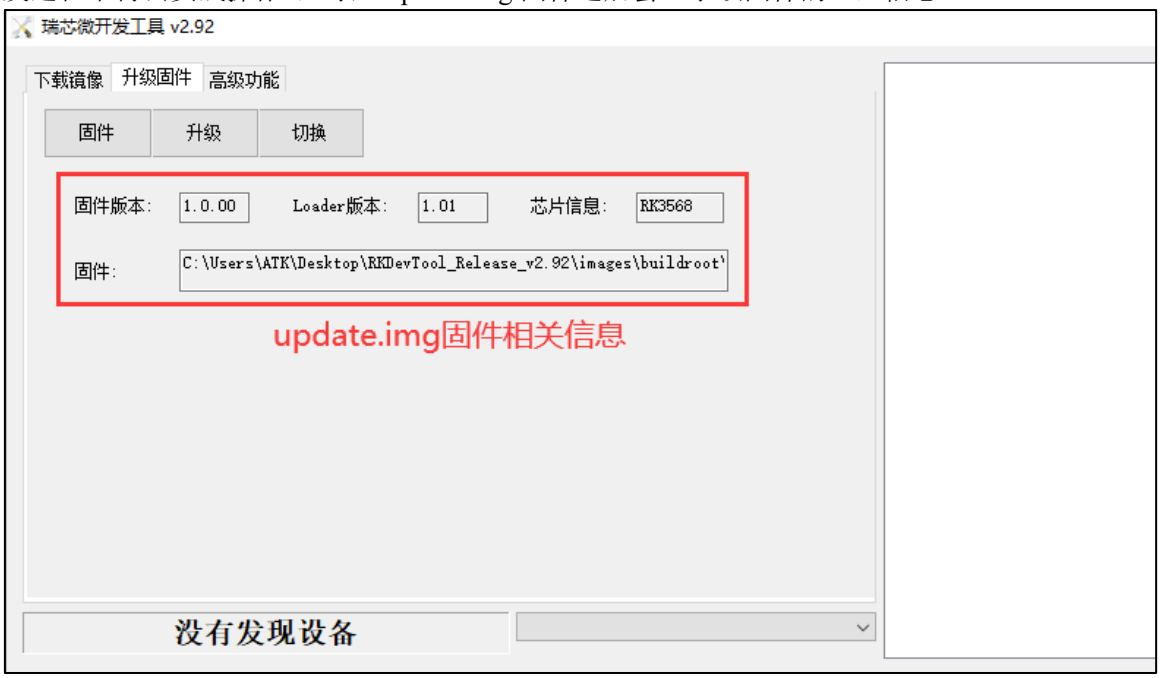


图 2.9.3.2 导入 update.img 镜像

首先让设备进入 Maskrom 或 Loader 模式，然后点击“升级”按钮进行固件升级、更新：

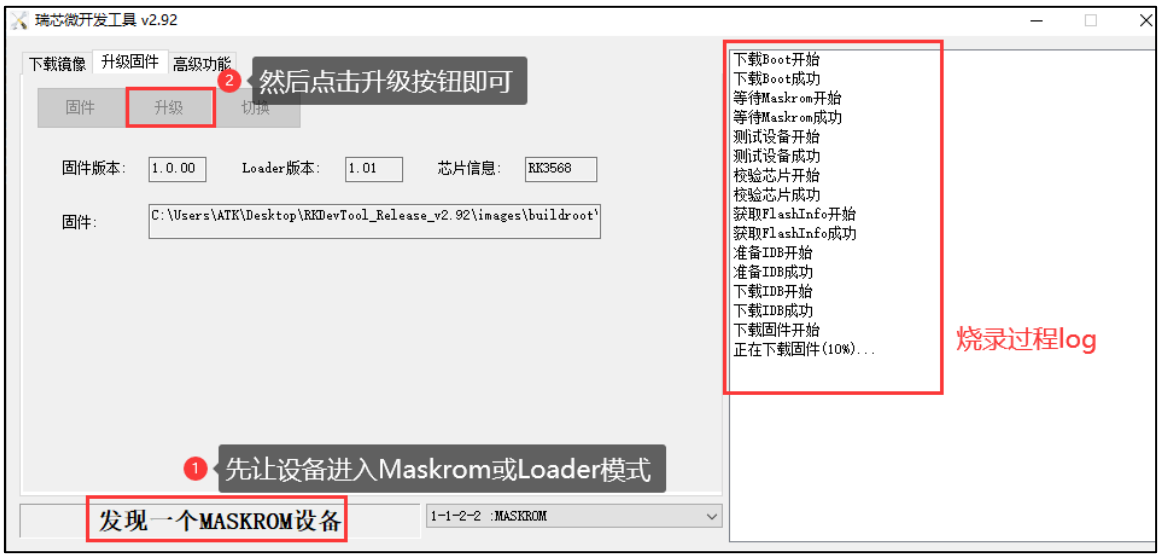


图 2.9.3.3 update.img 升级

固件烧录完成后，会自动重启开发板。

2.9.4 擦除操作

通过瑞芯微开发工具可以擦除 Flash，将烧录到 Flash 中的镜像擦除。打开瑞芯微开发工具，点击“**高级功能**”选项卡，如下所示：

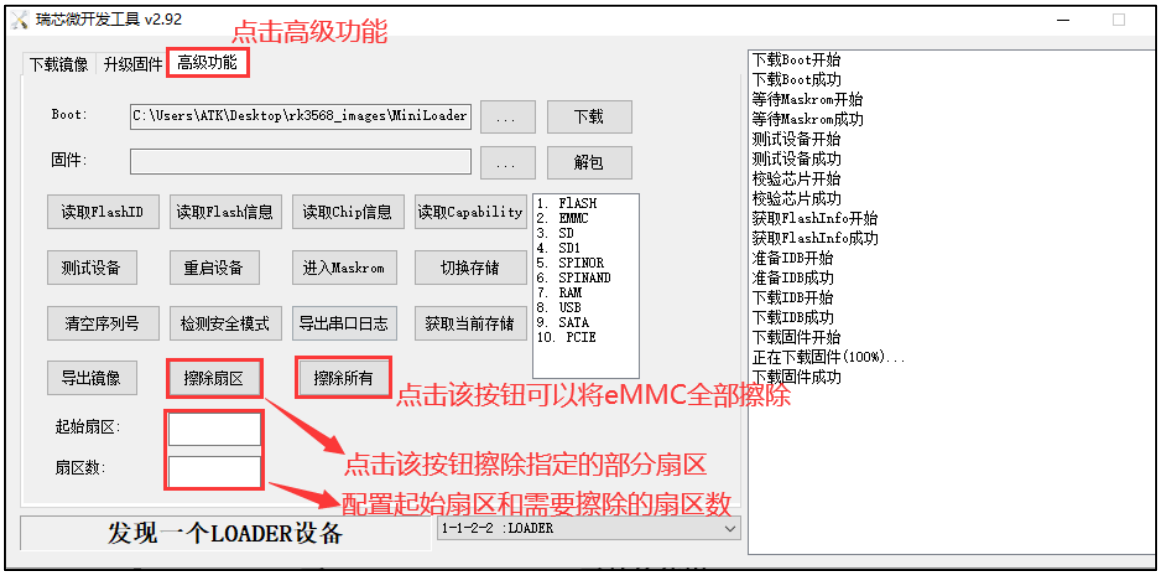


图 2.9.4.1 高级功能页面

需要注意，擦除操作需要在 Loader 模式下进行（Maskrom 模式也可以执行擦除操作，但需要先下载 MiniLoaderAll.bin，首先选择 MiniLoaderAll.bin 镜像，然后点击上图中的“**下载**”按钮执行下载操作，详情请参考<开发工具使用文档_v1.0.pdf>文档）。点击“**擦除所有**”按钮可以将开发板 eMMC 中的数据全部擦除；点击“**擦除扇区**”按钮可以擦除用户指定的部分扇区，将擦除的起始扇区填写至“**起始扇区**”输入框，将需要擦除的总扇区数填写至“**扇区数**”输入框即可。

譬如点击“**擦除所有**”按钮来擦除开发板 eMMC 中的全部数据：

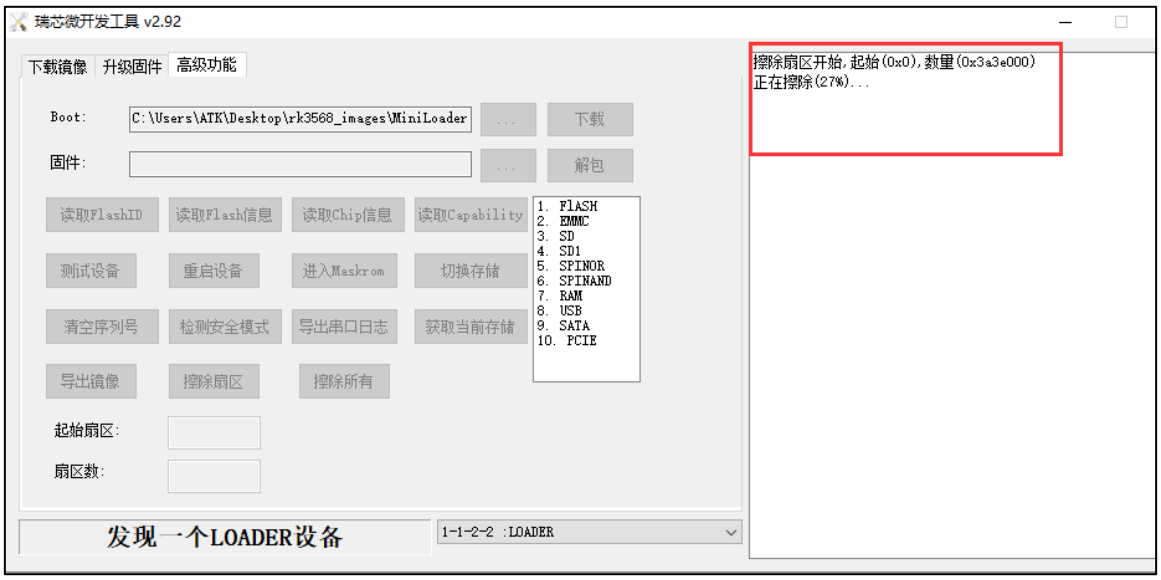


图 2.9.4.2 擦除所有操作

2.9.5 Ubuntu 下 Linux_Upgrade_Tool 工具的使用

开发、调试阶段通常是在 Ubuntu 下编译源码、编译生成的镜像也存在于 Ubuntu 系统中；使用瑞芯微开发工具（RKDevTool）烧录之前需要先将这些镜像文件拷贝到 Windows 系统，然后再进行烧写，所以确实会有些麻烦。

那么除了可以在 Windows 下烧录之外，我们还可以在 Ubuntu 下烧录镜像，这样就避免了镜像的拷贝过程；在 Ubuntu 系统下，可以使用 RK 提供的 Linux_Upgrade_Tool 工具进行烧录。

该工具在 SDK 中，SDK 自带了该烧录工具，**第五章**再给大家介绍。

2.10 ADB 工具安装

adb 全称 Android Debug Bridge，直译过来就是 Android 调试桥，它是一个通用的命令行工具。adb 做为 Android 设备与 PC 端连接的一个桥梁，它可以让开发者通过网络或 USB 与 Android 设备进行通信，从而作为 Android 设备的调试工具；在 Android 开发中，adb 是必不可少的一个工具。

用户可以通过 adb 在电脑上对 Android 模拟器或者真实的 Android 设备进行一系列操作，比如安装、卸载和调试应用（apk），拷贝推送文件（在电脑与 Android 设备之间拷贝文件），查看设备信息，抓取 log 等操作。它的主要功能包括：

- 运行设备的 shell（命令行）；
- 管理模拟器或设备的端口映射；
- 计算机与设备之间的文件上传/下载；
- 将本地 apk 软件安装至模拟器或 Android 设备。

需要说明的是，adb 虽然是 Android 设备的调试工具，但随着 adb 的普及，不仅仅是 Android 设备，在嵌入式开发中，很多 Linux 设备也同样支持 adb 调试，例如 Rockchip 平台。

关于 ADB 工具的安装以及使用方法，本文档不做介绍，用户可以参考正点原子提供的另一份文档《**adb 工具使用说明.pdf**》，路径为：**开发板光盘 A 盘-基础资料→10、用户手册→03、辅助文档→【正点原子】adb 工具使用说明.pdf**，这份文档会向用户介绍如何在 Windows 系统以及 Ubuntu 系统下安装 adb 工具、以及如何使用 adb 工具。

第三章 正点原子 ATK-DLRK3568 平台简介

参考文档：

《【正点原子】ATK-DLRK3568 开发板硬件参考手册.pdf》

3.1 RK3568 简介

参考文档：

《【正点原子】ATK-DLRK3568 开发板硬件参考手册.pdf》第一章

3.2 正点原子 ATK-RK3568 开发板硬件资源简介

参考文档：

《【正点原子】ATK-DLRK3568 开发板硬件参考手册.pdf》第二章

第四章 RK3568 Linux SDK 软件包

本章向用户介绍正点原子 ATK-DLRK3568 硬件平台 Linux SDK 软件包的安装以及使用方法，该 SDK 适用于正点原子 ATK-DLRK3568 开发板以及基于此开发板进行二次开发的所有 Linux 产品；基于本 SDK，可以有效实现系统定制和应用移植开发，帮助用户快速开发、提高开发效率！

Linux SDK 支持 buildroot、Yocto 以及 Debian 三种根文件系统，Linux 内核版本为 4.19、U-Boot 版本为 2017.09。

本章将分为如下几个小节：

4.1 安装 RK3568 Linux SDK
4.2 SDK 软件架构介绍
4.3 SDK 全自动编译
4.4 单独编译
4.5 SDK 清理
4.6 镜像介绍

RK 官方参考文档：
<SDK>/docs/Rockchip_Developer_Guide_Linux_Software_CN.pdf

开发板光盘 A 盘-基础资料→08、RK 官方文档→Linux→ Rockchip_Developer_Guide_Linux_Software_CN.pdf
强烈建议大家去看看！

4.1 安装 RK3568 Linux SDK

本小节向用户介绍如何在 Ubuntu 系统下安装正点原子 ATK-DLRK3568 开发板 SDK 软件包。

4.1.1 安装依赖软件包

首先需要在 Ubuntu 系统下安装 SDK 编译环境所依赖的软件包，执行如下命令安装软件包：

```
sudo apt-get update
sudo apt-get install curl python2.7 python-pyelftools git ssh make gcc libssl-dev liblz4-tool expect g++ patchelf chrpath gawk texinfo chrpath diffstat binfmt-support qemu-user-static live-build bison flex fakeroot cmake gcc-multilib g++-multilib unzip device-tree-compiler python3-pip libncurses-dev python3-pyelftools vim mtd-utils
```

安装过程中确保 Ubuntu 系统网络连接正常，安装过程需要一定的时间，请用户耐心等待！

将 python2 设置为系统默认 python 版本：

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

4.1.2 安装 repo（跳过）

repo 是建立在 Git 上的一个多仓库管理工具，可以组织多个仓库的上传和下载，用于管理多个 Git 存储仓库。RK3568 Linux SDK 中代码和相关文档被划分成了若干个 git 仓库分别进行版本管理（SDK 包含了若干个 git 仓库），开发者可以使用 repo 工具对这些 git 仓库进行统一下载、提交、切换分支等操作。

安装 SDK 需要使用到 repo 工具，所以要先安装 repo。

首先在用户家目录下创建一个 bin 文件夹，并将其导出到 PATH 环境变量：

```
mkdir ~/bin
```

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

如果可以访问 Google，可通过如下命令下载 repo，并赋予其可执行权限：

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

国内网络环境下如果执行上述命令后发现~/bin/repo 文件为空，此时可以访问国内的站点来下载 repo 工具，并赋予其可执行权限：

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

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

图 4.1.2.1 下载 repo 工具

repo 工具其实就是一个 python 脚本。

4.1.3 Git 配置

使用 repo 之前需要用户配置自己的 git 信息，否则后面的操作可能会遇到 hook 检查的障碍：

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

请用户根据实际情况配置。可通过如下命令查看 git 配置信息：

```
git config --list
```

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

图 4.1.3.1 查看 git 配置信息

4.1.4 安装 SDK

接下来开始安装 RK3568 Linux SDK，开发板资料包中已经给用户提供了 SDK 的压缩包文件，路径为：**开发板光盘 B 盘-开发环境及 SDK→02、ATK-DLRK3568 开发板 SDK→atk-rk3568_linux_release_v1.0_20230620.tgz**，随着版本的更新，SDK 压缩包文件的名字也将会发生改变，但均以 **atk-rk3568_linux_release_版本_发布日期.tgz** 方式进行命名。每次发布新版本 SDK 时，将会替换网盘中的旧版本 SDK，如果用户需要获取旧版本 SDK，请联系正点原子 Linux 技术支持获取！

将 tgz 压缩文件拷贝到 Ubuntu 系统的用户家目录下，并执行如下命令将其解压到家目录下的 rk3568_linux_sdk 目录：

```
mkdir ~/rk3568_linux_sdk
tar xvf atk-rk3568_linux_release_v1.0_20230620.tgz -C ~/rk3568_linux_sdk
```

```
tgg@tgg-virtual-machine:~$
tgg@tgg-virtual-machine:~$ ls
公共的  模板  视频  图片  文档  下载  音乐  桌面  atk-rk3568_linux_release_v1.0_20230620.tgz  bin
tgg@tgg-virtual-machine:~$
tgg@tgg-virtual-machine:~$ mkdir ~/rk3568_linux_sdk
tgg@tgg-virtual-machine:~$ tar xvf atk-rk3568_linux_release_v1.0_20230620.tgz -C ~/rk3568_linux_sdk
```

图 4.1.4.1 SDK 包解压

解压完成后，~/rk3568_linux_sdk/目录下会存在一个.repo 文件夹，如图 4.1.4.2 所示：

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

图 4.1.4.2 .repo 文件夹

执行如下命令可检出源码：

```
cd ~/rk3568_linux_sdk/
.repo/repo/repo sync -l -j10
```

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

图 4.1.4.3 检出源码

同步完成后，~/rk3568_linux_sdk/目录下的内容如图 4.1.4.4 所示：

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

图 4.1.4.4 SDK 工程目录

4.1.5 SDK 更新

正点原子技术团队会对 SDK 软件进行更新、升级，并以压缩包文件的形式提供给用户，压缩包文件的命名方式为 **atk-rk3568_linux_release_版本_发布日期.tgz**，每次发布新版本 SDK 时会替换资料网盘中的旧版 SDK，所以，网盘中的 SDK 即为当前最新版本 SDK；如需获取最新版本 SDK，只能通过正点原子资料网盘进行下载；如需获取旧版本 SDK，请联系正点原子 Linux 技术支持！

每一次的 SDK 版本更新将会通过 **SDK/.repo/manifests/rk356x_linux/ATK-RK3568_Linux_SDK_Note.md** 文档进行记录，如果用户想要了解 SDK 版本更新记录可以查看该文档（建议使用 Markdown 文档阅读器打开）。

4.1.6 SDK 问题反馈

用户在使用、开发过程中，如发现了 SDK 的一些问题、软件 BUG、具体技术问题、技术咨询等，可以通过 QQ 群或者淘宝联系正点原子 Linux 技术支持，并向其说明问题，后续将会安排开发人员进行相应的处理、跟踪。

4.1.7 SDK 瘦身

如果用户的 Ubuntu 系统磁盘空间比较紧张，可以对 SDK 进行瘦身，将 SDK 源码根目录下的.repo（隐藏文件夹，使用 ls -a 可看到）文件夹删除；注意，.repo 文件夹中保存了 SDK 所有 git 仓库的 git 提交信息，一旦删除，则后续将无法查看到任何仓库的 git 提交信息，所以不建议删除。

4.2 SDK 软件架构介绍

本小节向用户介绍 SDK 软件架构。

4.2.1 SDK 工程目录介绍

SDK 源码根目录下包含有 app、buildroot、debian、device、external、tools、u-boot、yocto 等多个目录，每个目录或其子目录会对应一个 git 工程；因为 SDK 的代码和相关文档被划分成了若干 git 仓库分别进行版本管理（所以 SDK 实际上包含有若干 git 仓库），它们按照功能、所属模块划分，分别组织到不同的目录下。

- app：存放上层应用 app，包括 Qt 应用程序，以及其它的 C/C++应用程序。
- buildroot：基于 buildroot 开发的根文件系统。
- debian：基于 Debian 开发的根文件系统。
- device/rockchip：存放各芯片板级配置文件和 Parameter 分区表文件，以及一些编译与打包固件的脚本和预备文件。
- docs：存放芯片模块开发指导文档、平台支持列表、芯片平台相关文档、Linux 开发指南等。
- external：存放所需的第三方库，包括音频、视频、网络、recovery 等。
- kernel：Linux 4.19 版本内核源码。
- prebuilts：存放交叉编译工具链。
- rkbin：存放 Rockchip 相关的 Binary 和工具。
- rockdev：存放编译输出固件，编译 SDK 后才会生成该文件夹。
- tools：存放 Linux 和 Windows 操作系统环境下常用的工具，包括镜像烧录工具、SD 卡升级启动制作工具、批量烧录工具等，譬如前面给大家介绍的 RKDevTool 工具以及 Linux_Upgrade_Tool 工具都存放在该目录。
- u-boot：基于 v2017.09 版本进行开发的 uboot 源码。
- yocto：基于 Yocto 开发的根文件系统。

4.2.2 SDK 软件框图

SDK 软件框图如图 4.2.2.1 所示，从下至上分为 Bootloader、Linux Kernel、Libraries、Applications 四个层次，各层次内容如下：

- BootLoader 层主要提供底层系统支持包，如 BootLoader、U-Boot、ATF 相关支持。
- Kernel 层主要提供 Linux Kernel 的标准实现，Linux 也是一个开放的操作系统。Rockchip 平台的 Linux 核心为标准的 Linux4.4/4.19/5.10 内核，提供安全性、内存管理、进程管理、网络协议栈等基础支持；主要是通过 Linux 内核管理设备硬件资源，如 CPU 调度、缓存、内存、I/O 等。

- Libraries 层对应一般嵌入式系统，相当于中间件层次，包含了各种系统基础库，以及第三方开源程序库支持，对应用层提供 API 接口，系统定制者和应用开发者可以基于 Libraries 层的 API 开发新的应用。
- Applications 层主要是实现具体的产品功能及交互逻辑，需要一些系统基础库及第三方程序库支持，开发者可以开发实现自己的应用程序，提供系统各种能力给到最终用户。

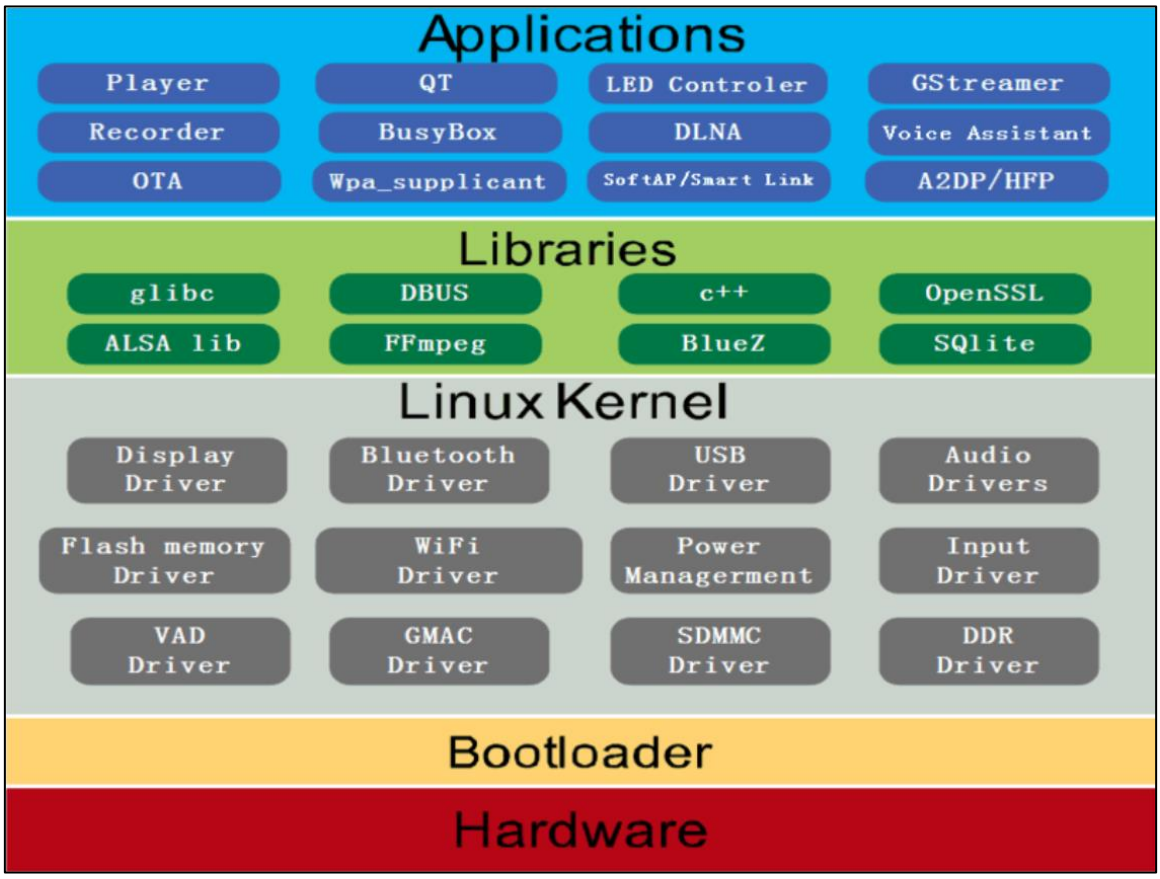


图 4.2.2.1 SDK 软件框图

4.2.3 SDK 版本查询

正点原子 RK3568 Linux SDK 的版本可分为 RK 版本和 ATK 版本；所谓 RK 版本，则表示本 SDK 是基于 RK 官方（Rockchip，瑞芯微）的某版本 Linux SDK 进行的二次开发；在 RK 原生 SDK 的基础上进行二次开发，以适配我们正点原子的 ATK-DLRK3568 开发平台以及进行一些相应的扩展。

而 ATK 版本则表示正点原子 Linux 技术团队对 RK3568 Linux SDK 所定义的内部版本号，每一次发布的 SDK 都会有一个版本号与之对应。

在 SDK 源码根目录下，执行如下命令可查询当前 SDK 的 ATK 版本：

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

```
dengtao@android:/[redacted]/rk3568_linuxSDK$ realpath .repo/manifests/rk3568_linux_release.xml
/[redacted]/.repo/manifests/rk356x_linux/atk-rk3568_linux_release_v1.0_xxx.xml
dengtao@android:/[redacted]/rk3568_linuxSDK$
dengtao@android:/[redacted]/rk3568_linuxSDK$ _
```

图 4.2.3.1 查询 SDK 的 ATK 版本

从图中可知，当前 SDK 的 ATK 版本为 V1.0，发布日期为 xxx。

在 SDK 源码根目录下，执行如下命令可查询当前 SDK 的 RK 版本：

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

```
dengtao@android:/[redacted]/rk3568_linuxSDK$ ls .repo/manifests/rk356x_linux/rk356x_linux_release*
.repo/manifests/rk356x_linux/rk356x_linux_release_v1.3.0_20220620.xml
dengtao@android:/[redacted]/rk3568_linuxSDK$
dengtao@android:/[redacted]/rk3568_linuxSDK$ _
```

图 4.2.3.2 查询 SDK 的 RK 版本

从图中可知，当前 SDK 的 RK 版本为 V1.3.0，发布日期为 2022 年 06 月 20 日。

4.3 SDK 全自动编译

本小节向用户介绍如何编译 RK3568 Linux SDK，首先进入到 SDK 源码根目录下，在编译之前先执行如下命令指定 SDK 的板级配置文件：

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

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

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

0. default BoardConfig.mk
1. BoardConfig-rk3566-evb2-lp4x-v10-32bit.mk
2. BoardConfig-rk3566-evb2-lp4x-v10.mk
3. BoardConfig-rk3568-atk-evb1-ddr4-v10.mk ATK-DLRK3568开发板对应的板级配置文件
4. BoardConfig-rk3568-evb1-ddr4-v10-32bit.mk
5. BoardConfig-rk3568-evb1-ddr4-v10-spi-nor-64M.mk
6. BoardConfig-rk3568-evb1-ddr4-v10.mk
7. BoardConfig-rk3568-nvr-spi-nand.mk
8. BoardConfig-rk3568-nvr.mk
9. BoardConfig-rk3568-uvc-evb1-ddr4-v10.mk
10. BoardConfig.mk
Which would you like? [0]: 3 <- 输入板级配置文件所对应的序号
```

图 4.3.1 执行 build.sh lunch 选择板级配置文件

输入板级配置文件对应的序号、然后按回车确认：

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

图 4.3.2 确认选择对应的板级配置文件

build.sh 是 RK 提供的一个编译脚本，使用该脚本可以方便用户快速构建出各种镜像文件以及对镜像进行打包操作，既可以一键全自动编译整个 SDK，也可以单独编译 U-Boot、Linux Kernel、buildroot 等，非常方便！

build.sh 脚本其实是一个软链接文件，实际指向了 device/rockchip/common/build.sh 文件，如下图所示：

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

图 4.3.3 build.sh 软链接文件

可通过执行如下命令查看 build.sh 脚本的使用方法：

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

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

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

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

图 4.3.4 查看 build.sh 脚本帮助信息

从图中可知，build.sh 脚本支持的参数比较多，在开发过程中常用的也就几个而已，如下表所示：

build.sh 脚本参数	说明	示例
lunch	选择板级配置文件	./build.sh lunch
uboot	编译 u-boot	./build.sh uboot
kernel	编译 kernel	./build.sh kernel
modules	编译内核模块	./build.sh modules
rootfs	编译根文件系统	./build.sh rootfs
buildroot	编译 buildroot 根文件系统	./build.sh buildroot
debian	编译 Debian 根文件系统	./build.sh debian
recovery	编译 recovery	./build.sh recovery
all	编译整个 SDK，包括 uboot、kernel、rootfs、recovery	./build.sh all
cleanall	清理整个 SDK	./build.sh cleanall
firmware	将镜像打包到 rockdev 目录	./build.sh firmware
updateimg	将所有镜像打包成一个 update.img 固件	./build.sh updateimg

表 4.3.1 build.sh 常用命令

后续还会给大家介绍这些参数的使用方法，这里先对其有一个基本的了解即可！

选择板级配置文件后，接下来便可以进行编译了；整个 SDK 编译过程中最耗时的部分便是根文件系统的编译了，在编译根文件系统的过程中会通过网络下载很多的第三方库文件；首先，下载过程会占用很多时间导致编译时间拉长；其次，如果用户的网络环境不稳定或者第三方库

文件的下载源发生变更，很容易导致下载失败，进而导致根文件系统编译出错；所以，为了加快根文件系统的编译过程、也为了降低编译根文件系统时出现问题的概率，我们可以预先把编译根文件系统所需的第三方库文件拷贝到 SDK 中。

正点原子 ATK-DLRK3568 开发板资料包中已经给用户提供了这些所需的第三方库文件，具体路径为：**开发板光盘 B 盘-开发环境及 SDK→02、ATK-DLRK3568 开发板 SDK→dl.tgz**，首先将该压缩文件拷贝到 Ubuntu 系统家目录下，如下所示：

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

图 4.3.5 dl.tgz 压缩文件

执行如下命令将其解压到<SDK>/buildroot 目录下：

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

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

图 4.3.6 dl.tgz 解压

解压完成后，可以进入到<SDK>/buildroot 目录下，该目录下会存在一个 dl 目录，该目录下存放的便是第三方库文件，如下所示：

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

图 4.3.7 dl 目录下的库文件

准备工作做完之后接下来便可以编译 SDK 了，进入到 SDK 源码根目录下，执行如下命令编译整个 SDK：

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

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk$ ./build.sh all
processing option: all
=========================================
TARGET_ARCH=arm64
TARGET_PLATFORM=rk356x
TARGET_UBOOT_CONFIG=rk3568
TARGET_SPL_CONFIG=
TARGET_KERNEL_CONFIG=rockchip_linux_defconfig
TARGET_KERNEL_DTS=rk3568-atk-evb1-ddr4-v10-linux
TARGET_TOOLCHAIN_CONFIG=
TARGET_BUILDROOT_CONFIG=rockchip_rk3568
TARGET_RECOVERY_CONFIG=rockchip_rk356x_recovery
TARGET_PCBA_CONFIG=
TARGET_RAMBOOT_CONFIG=
=========================================
============Start building uboot============
TARGET_UBOOT_CONFIG=rk3568
=========================================
grep: .config: No such file or directory
## make rk3568_defconfig -j24
  HOSTCC  scripts/basic/fixdep
  HOSTCC  scripts/kconfig/conf.o
  SHIPPED scripts/kconfig/zconf.tab.c
  SHIPPED scripts/kconfig/zconf.lex.c
  SHIPPED scripts/kconfig/zconf.hash.c
  HOSTCC  scripts/kconfig/zconf.tab.o
In file included from scripts/kconfig/zconf.tab.c:2468:
scripts/kconfig/confdata.c: In function ‘conf_write’:
scripts/kconfig/confdata.c:771:19: warning: ‘%s’ directive writing likely 7 or more bytes into a region of size between 1 an
  771 |  sprintf(newname, "%s%s", dirname, basename);
      |                   ^~~~~~
scripts/kconfig/confdata.c:771:19: note: assuming directive output of 7 bytes
In file included from /usr/include/stdio.h:867,
                 from scripts/kconfig/zconf.tab.c:82:
/usr/include/x86_64-linux-gnu/bits/stdio2.h:36:10: note: ‘__builtin___sprintf_chk’ output 1 or more bytes (assuming 4104) in
   36 |   return __builtin___sprintf_chk (__s, __USE_FORTIFY_LEVEL - 1,
      |          ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   37 |        __bos (__s), __fmt, __va_arg_pack ());
```

图 4.3.8 一键全自动编译整个 SDK

整个编译过程将会持续很长一段时间，整个编译过程所花费的时间长短与个人电脑配置有关，快则一个小时左右、慢则 3、4 个小时，甚至更长的时间。

如果编译失败、并且出现类似如下错误信息，则表示你的 Ubuntu 系统内存不足：

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

出现这种错误 表示内存不足

图 4.3.9 内存不足导致编译失败

那么在这种情况下，可以有两种方法尝试去解决它；首先，如果硬件条件允许的情况下，直接扩大 Ubuntu 系统的内存容量即可，这也是最直接、最有效的方法；如果硬件条件不允许，已经没有多余的物理内存再分给 Ubuntu 系统了，那么这种情况下，我们可以增大 Ubuntu 系统的 swap 交互空间，通过增大 swap 交换空间来尝试解决这个由于内存不足所导致的编译失败问题。

推荐大家使用第一种方法来解决该问题，也就是增大 Ubuntu 内存容量；实在不行，再尝试通过增大 swap 交换空间来解决；Ubuntu 系统 swap 交换空间默认大小为 2G，譬如可以将其配置为 8G、16G、32G 等等。对于如何配置、增大 swap 交换空间，本文档不作说明，大家可以自行百度解决，也可以参考正点原子提供的文档**《Ubuntu 系统扩充 swap 交换空间.pdf》**，文档路径为：**开发板光盘 A 盘-基础资料→10、用户手册→03、辅助文档→【正点原子】Ubuntu 系统扩充 swap 交换空间.pdf**。

在编译过程中，除了可能会遇到该问题之外，可能还会遇到其它的一些问题，毕竟每个人的开发环境可能或多或少的存在一些差异，所以大家应尽量按照本文档说明进行操作，尤其是 4.1.1 小节中所要求安装的依赖软件包。

如果没有出现意外，整个 SDK 编译将会成功，如图 4.3.10 所示：

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

整个SDK编译成功

图 4.3.10 SDK 编译成功

出现了“**Running build_all succeeded**”字符串则表示 SDK 编译成功了，那么恭喜各位！

编译完成后，会生成各种镜像，包括 boot.img、uboot.img、MiniLoaderAll.bin、rootfs.img、recovery.img 等等，但是这些镜像文件散布在各自的源码目录下、不方便用户查找，此时我们可以执行如下命令将它们打包到 SDK/rockdev 目录：

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

或者直接执行 SDK 源码根目录下的./mkfirmware.sh 脚本（./build.sh firmware 命令其实就是执行了 mkfirmware.sh 脚本）：

```
./mkfirmware.sh
```

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

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

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

图 4.3.11 打包镜像到 rockdev 目录

执行完命令后进入到 rockdev 目录下，如下所示：

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

图 4.3.12 rockdev 目录下的文件

该目录下的文件基本都是软链接，链接到真正的镜像文件。

除了使用“**./build.sh all**”命令外，我们还可以直接执行“**./build.sh**”脚本（不带任何参数）来编译整个 SDK；运行“**./build.sh**”命令会在“**./build.sh all**”命令的基础上增加如下操作：

- 1、执行./mkfirmware.sh 将所有镜像打包到 rockdev 目录
- 2、将 rockdev 目录下的所有镜像打包成一个 update.img 固件
- 3、复制 rockdev 目录下的镜像到 IMAGE/***_RELEASE_TEST/IMAGES 目录(***表示编译日期)
- 4、保存各个模块的补丁到 IMAGE/***_RELEASE_TEST/PATCHES 目录
- 注：./build.sh 和./build.sh allsave 命令一样

4.4 单独编译

上一小节向用户介绍了如何一键全自动编译整个 SDK 得到所有镜像，本小节向用户介绍如何单独编译各部分源码得到相应的镜像。

4.4.1 单独编译 U-Boot

通过 build.sh 脚本单独编译 U-Boot，在 SDK 源码根目录下执行如下命令：

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

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

图 4.4.1.1 单独编译 U-Boot(1)

```
 Configuration 0 (conf)
  Description:  rk3568-evb
  Kernel:       unavailable
  Firmware:     atf-1
  FDT:          fdt
  Loadables:    uboot
                atf-2
                atf-3
                atf-4
                atf-5
                atf-6
                optee
********boot_merger ver 1.2********
Info:Pack loader ok.
pack loader okay! Input: /home/tgg/rk3568_linux_sdk/rkbin/RKBOOT/RK3568MINIALL.ini
/home/tgg/rk3568_linux_sdk/u-boot

Image(no-signed, version=0): uboot.img (FIT with uboot, trust...) is ready
Image(no-signed): rk356x_spl_loader_v1.13.112.bin (with spl, ddr...) is ready
pack uboot.img okay! Input: /home/tgg/rk3568_linux_sdk/rkbin/RKTRUST/RK3568TRUST.ini

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

图 4.4.1.2 单独编译 U-Boot(2)

编译成功后，会生成如下两个镜像：

<SDK>/uboot/uboot.img
<SDK>/uboot/rk356x_spl_loader_v1.13.112.bin

rk356x_spl_loader_v1.13.112.bin 其实就是 MiniLoaderAll.bin，只是进行了重命名而已。

4.4.2 单独编译 Kernel

通过 build.sh 脚本单独编译 Linux Kernel，在 SDK 源码根目录下执行如下命令：

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

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk$ ./build.sh kernel
processing option: kernel
============Start building kernel============
TARGET_ARCH          =arm64
TARGET_KERNEL_CONFIG =rockchip_linux_defconfig
TARGET_KERNEL_DTS    =rk3568-atk-evb1-ddr4-v10-linux
TARGET_KERNEL_CONFIG_FRAGMENT =
=============================================
  HOSTCC  scripts/basic/fixdep
  HOSTCC  scripts/kconfig/conf.o
  YACC    scripts/kconfig/zconf.tab.c
  LEX     scripts/kconfig/zconf.lex.c
  HOSTCC  scripts/kconfig/zconf.tab.o
  HOSTLD  scripts/kconfig/conf
#
# configuration written to .config
#
  WRAP    arch/arm64/include/generated/uapi/asm/errno.h
  WRAP    arch/arm64/include/generated/uapi/asm/ioctl.h
  WRAP    arch/arm64/include/generated/uapi/asm/ioctls.h
  WRAP    arch/arm64/include/generated/uapi/asm/ipcbuf.h
  WRAP    arch/arm64/include/generated/uapi/asm/kvm_para.h
  WRAP    arch/arm64/include/generated/uapi/asm/mman.h
  WRAP    arch/arm64/include/generated/uapi/asm/msgbuf.h
  WRAP    arch/arm64/include/generated/uapi/asm/poll.h
  WRAP    arch/arm64/include/generated/uapi/asm/resource.h
  WRAP    arch/arm64/include/generated/uapi/asm/sembuf.h
  WRAP    arch/arm64/include/generated/uapi/asm/shmbuf.h
  WRAP    arch/arm64/include/generated/uapi/asm/socket.h
  WRAP    arch/arm64/include/generated/uapi/asm/sockios.h
  WRAP    arch/arm64/include/generated/uapi/asm/swab.h
  WRAP    arch/arm64/include/generated/uapi/asm/termbits.h
```

图 4.4.2.1 单独编译 Kernel(1)

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

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

4.4.2.2 单独编译 Kernel(2)

编译成功后会生成 **boot.img** 镜像，路径为：**<SDK>/kernel/boot.img**。

执行“**./build.sh kernel**”命令会编译 Linux 内核源码，包括内核设备树、内核模块，如果我们需要单独编译内核模块，可以执行如下命令进行编译：

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

当然，编译内核模块之前需要先编译好内核源码。

4.4.3 单独编译 rootfs

rootfs 也就是根文件系统，RK3568 Linux SDK 支持多种根文件系统，包括 buildroot、yocto 以及 Debian，本小节主要介绍如何编译 buildroot 根文件系统，**同样也推荐用户使用 buildroot，不推荐使用 Yocto，RK 官方本身也不推荐用户使用 Yocto**。

通过 build.sh 脚本单独编译 buildroot 根文件系统，在 SDK 源码根目录下执行如下命令：

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

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

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

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

图 4.4.3.1 单独编译 buildroot(1)

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

图 4.4.3.2 单独编译 buildroot(2)

编译成功后会生成 buildroot 根文件系统镜像，镜像输出在 buildroot/output/rockchip_rk3568/images/目录下，如下所示：

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

图 4.4.3.3 buildroot 根文件系统镜像

编译生成了多个不同格式的 rootfs 镜像文件，对于 RK3568 平台来说，使用 ext4 格式镜像 **rootfs.ext2**，并通常会将其重命名为 rootfs.img。

除此之外，还可以通过“**./build.sh rootfs**”命令编译 buildroot 根文件系统，该命令用于编译根文件系统，但不局限于 buildroot，也可以编译 Yocto 以及 Debian；默认情况下编译的是 buildroot，可以通过环境变量 **RK_ROOTFS_SYSTEM** 指定需要编译的 rootfs（yocto/debian/buildroot）：

```
# 指定编译 buildroot 根文件系统
export RK_ROOTFS_SYSTEM=buildroot
./build.sh rootfs
```

需要注意的是：编译根文件系统之前，需提前编译好 Linux 内核；因为编译根文件系统的过程中、也会编译部分未集成在内核源码中的驱动模块（单独提供驱动源码，譬如蓝牙驱动模块 hci_uart.ko）、而且也会将内核源码目录下编译生成的.ko 驱动模块拷贝至根文件系统（譬如 WiFi 驱动模块 8852bs.ko），所以必须先编译好内核。

4.4.4 单独编译 recovery

通过 build.sh 脚本单独编译 recovery，在 SDK 源码根目录下执行如下命令：

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

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

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

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

图 4.4.4.1 单独编译 recovery(1)

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

图 4.4.4.2 单独编译 recovery(2)

recovery.img 用于进入 recovery 模式，该镜像会烧录到开发板 recovery 分区。

recovery.img 是由多个镜像合并而成，其中包含 ramdisk（recovery 模式下挂载的根文件系统）、内核镜像、内核 DTB 以及资源镜像 resource.img。所以，在编译 recovery 之前，也必须提前编译好 Linux 内核。

编译成功后，会生成 recovery.img，该镜像输出在 buildroot/output/rockchip_rk356x_recovery/images/目录下，如下所示：

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

图 4.4.4.3 recovery 镜像文件

4.4.5 打包成 update.img 镜像

update.img 是多个镜像的集合体（由多个镜像打包合并而成），使用 RK 提供的工具可以将各个分立镜像（譬如 uboot.img、boot.img、MiniLoaderAll.bin、parameter.txt、misc.img、rootfs.img、oem.img、userdata.img、recovery.img 等）打包成一个 update.img 固件，方便用户烧录、升级。

我们可以通过如下命令将 rockdev 目录下的各个分立镜像打包成一个 update.img 固件，使用 update.img 固件更加方便烧录、更新！

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

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk$ ./build.sh updateimg
processing option: updateimg
Make update.img
start to make update.img...
Android Firmware Package Tool v2.0
------ PACKAGE ------
Add file: ./package-file
package-file,Add file: ./package-file done,offset=0x800,size=0x28b,userspace=0x1
Add file: ./Image/MiniLoaderAll.bin
bootloader,Add file: ./Image/MiniLoaderAll.bin done,offset=0x1000,size=0x719c0,userspace=0xe4
Add file: ./Image/parameter.txt
parameter,Add file: ./Image/parameter.txt done,offset=0x73000,size=0x1f4,userspace=0x1
Add file: ./Image/uboot.img
uboot,Add file: ./Image/uboot.img done,offset=0x73800,size=0x400000,userspace=0x800
Add file: ./Image/misc.img
misc,Add file: ./Image/misc.img done,offset=0x473800,size=0xc000,userspace=0x18
Add file: ./Image/boot.img
boot,Add file: ./Image/boot.img done,offset=0x47f800,size=0x1621600,userspace=0x2c43
Add file: ./Image/recovery.img
recovery,Add file: ./Image/recovery.img done,offset=0x1aa1000,size=0x1d93400,userspace=0x3b27
Add file: ./Image/rootfs.img
rootfs,Add file: ./Image/rootfs.img done,offset=0x3834800,size=0x27d11000,userspace=0x4fa22
Add file: ./Image/oem.img
oem,Add file: ./Image/oem.img done,offset=0x2b545800,size=0x10a6000,userspace=0x214c
Add file: ./Image/userdata.img
userdata,Add file: ./Image/userdata.img done,offset=0x2c5eb800,size=0x444000,userspace=0x888
Add CRC...
Make firmware OK!
------ OK ------
********rkImageMaker ver 2.0********
Generating new image, please wait...
Writing head info...
Writing boot file...
Writing firmware...
Generating MD5 data...
MD5 data generated successfully!
New image generated successfully!
Making ./Image/update.img OK.
Running build_updateimg succeeded.
tgg@tgg-virtual-machine:~/rk3568_linux_sdk$
```

图 4.4.5.1 打包成 update.img

打包成功后，会在 rockdev 目录下生成 update.img 固件，如下所示：

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

图 4.4.5.2 update.img 固件

4.5 SDK 清理

在 SDK 源码根目录下通过 build.sh 脚本可以执行清理操作，执行如下命令：

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

执行该命令将会清理 uboot、kernel、buildroot（rootfs、recovery）。

4.6 镜像介绍

前两个小节向用户介绍了如何编译 SDK，编译后会生成多个镜像文件，如下表所示：

镜像名称	作用
uboot.img	uboot.img 是一种 FIT 格式镜像，它由多个镜像合并而成，其中包括 trust 镜像（ARM Trusted Firmware + OP-TEE OS）、u-boot 镜像、u-boot dtb；编译 U-Boot 时会将这些镜像打包成一个 uboot.img。 uboot.img 会烧录到开发板 uboot 分区
boot.img	boot.img 也是一种 FIT 格式镜像，它也是由多个镜像合并而成，其中包括内核镜像、内核 DTB、资源镜像 resource.img。 boot.img 会烧录到开发板 boot 分区
MiniLoaderAll.bin	该镜像是运行在 RK3568 平台 U-Boot 之前的一段 Loader 代码（也就是比 U-Boot 更早阶段的 Loader），MiniLoaderAll.bin 由 TPL 和 SPL

	两部分组成，TPL 用于初始化 DDR，运行在 SRAM；而 SPL 运行在 DDR，主要负责加载、引导 uboot.img。
misc.img	包含 BCB（Bootloader Control Block）信息，该镜像会烧写到开发板 misc 分区。 misc 分区是一个很重要的分区，其中存放了 BCB 数据块，主要用于 Android/Linux 系统、U-Boot 以及 recovery 之间的通信
oem.img	给厂家使用，用于存放厂家的 APP 或数据，该镜像会烧写至开发板 oem 分区，系统启动之后会将其挂载到/oem 目录。
parameter.txt	一个 txt 文本文件，是 RK3568 平台的分区表文件（记录分区名以及每个分区它的起始地址、结束地址）；烧写镜像时，并不需要将 parameter.txt 文件烧写到 Flash，而是会读取它的信息去定义分区。
recovery.img	recovery 模式镜像，recovery.img 用于进入 recovery 模式，recovery.img 会烧录到开发板 recovery 分区。 recovery 模式是一种用于对设备进行修复、升级更新的模式。recovery.img 也是 FIT 格式镜像，也是由多个镜像合并而成，其中包括 ramdisk（进入 recovery 模式时挂载该根文件系统）、内核镜像（进入 recovery 模式时启动该内核镜像）、内核 DTB 以及 resource.img。
rootfs.img	正常启动模式下对应的根文件系统镜像，包含有大量的库文件、可执行文件等。 rootfs.img 会烧录到开发板 rootfs 分区
userdata.img	给用户使用，可用于存放用户的 App 或数据；该镜像会烧写至开发板 userdata 分区，系统启动之后，会将其挂载到/userdata 目录

表 4.6.1 RK3568 各镜像介绍

以上便是对这些镜像的一个简单介绍。

第五章 SDK 镜像烧录

本章向用户介绍如何将编译得到的镜像文件（uboot.img、boot.img、rootfs.img、recovery.img 等）烧写并运行在 ATK-DLRK3568 开发板上。

Rockchip 平台提供了多种镜像烧写方式，譬如在 Windows 下通过瑞星微开发工具烧写、通过 SD 卡方式烧写、通过 FactoryTool 工具批量烧写（量产烧写工具、支持 USB 一拖多烧写）以及在 Ubuntu 下通过 Linux_Upgrade_Tool 工具烧写等等；总之，烧写镜像的方式有很多种，用户可以选择合适的烧写方式进行烧写，将镜像文件烧写至开发板。

5.1 烧写模式介绍

2.9.1 小节已经详细介绍了开发板的烧写模式相关问题，如果对此还有疑问，请移步前往该小节阅读！

5.2 Windows 系统下烧写

Windows 下通过瑞芯微开发工具（RKDevTool）来烧写镜像。烧写之前，先将<第四章>编译 SDK 得到的镜像文件（<SDK>/rockdev/目录下的镜像）从 Ubuntu 系统拷贝到 Windows 下，譬如将这些镜像拷贝到 Windows 桌面 rk3568_images 目录，包括 boot.img、MiniLoaderAll.bin、misc.img、oem.img、parameter.txt（分区表文件，不是镜像）、recovery.img、rootfs.img、uboot.img、userdata.img，如下所示：

图 5.2.1 桌面 rk3568_images 目录下的文件

打开瑞芯微开发工具：

图 5.2.2 瑞芯微开发工具

5.2.1 分区表 parameter.txt 介绍

本小节讲一下 RK 平台分区表文件 parameter.txt，该文件是一个 txt 文本文件。parameter.txt 文件描述了开发板的分区表信息，每个分区的名字、分区的起始地址以及分区的大小等信息，我们来看下它的内容：

```
FIRMWARE_VER: 1.0
MACHINE_MODEL: RK3568
MACHINE_ID: 007
MANUFACTURER: RK3568
MAGIC: 0x5041524B
ATAG: 0x00200800
MACHINE: 0xffffffff
CHECK_MASK: 0x80
PWR_HLD: 0,0,A,0,1
TYPE: GPT
CMDLINE:
mtdparts=rk29xxnand:0x00002000@0x00004000(uboot),0x00002000@0x00006000(misc),0x
00020000@0x00008000(boot),0x00020000@0x00028000(recovery),0x00010000@0x00048000
(backup),0x00c00000@0x00058000(rootfs),0x00040000@0x00c58000(oem),-
@0x00c98000(userdata:grow)
uuid:rootfs=614e0000-0000-4b53-8000-1d28000054a9
```

parameter.txt 文件中除了分区表信息之外，还包含其它标识，譬如 FIRMWARE_VER、MACHINE_MODEL、MACHINE_ID、MAGIC 等，本文对此不做介绍，详情请参考 RK 官方文档：**<SDK>/docs/Common/TOOL/Rockchip_Introduction_Partition_CN.pdf**。

这里只给大家介绍 **mtdparts** 标识所定义的分区表信息。

mtdparts 定义的信息如下：

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

rk29xxnand 是一个标识，为了兼容性，rockchip 平台都是用 rk29xxnand 做标识。

诸如 0x00002000@0x00004000(uboot)、0x00002000@0x00006000(misc)等信息用于定义分区，@符号之前的数值是分区大小，@符号之后的数值是分区的起始位置，括号里面的字符是分区的名字；所有数值的单位都是 sector（扇区），1 个 sector 为 512 字节。所以由此可知，uboot 分区的起始位置为 0x4000 sectors 位置，大小为 0x2000 sectors（4MB）；misc 分区的起始位置为 0x6000 sectors 位置，大小也是 0x2000 sectors。

为了性能，每个分区起始地址需要 32KB（64 sectors）对齐，大小也需要 32KB 的整数倍。

最后一个分区需要指定 grow 参数，表示将剩余存储空间全部分配给该分区：

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

userdata 分区虽然指定了起始位置，但并未指定分区大小，而是使用了“-”来代替，然后在分区名后面加入“**:grow**”，表示将剩余空间全部分配给 userdata 分区。

每个分区的作用如下表所示：

分区名	说明
uboot	用于存放 uboot.img，uboot.img 镜像会烧录到该分区
misc	misc 分区是一个很重要的分区，其中包括 BCB 数据块，主要用于 Android/Linux 系统、U-Boot 以及 recovery 之间的通信 misc.img 镜像会烧录到该分区
boot	用于存放 boot.img，boot.img 镜像会烧录到该分区
recovery	用于 recovery 模式，recovery.img 会烧录到该分区，系统引导 recovery.img 进入到 recovery 模式。 recovery 模式是一种用于对设备进行修复、升级更新的模式。
backup	预留分区，暂时没有使用到
rootfs	根文件系统分区，用于存放 rootfs.img，正常启动模式下的根文件系统镜像 rootfs.img 会烧录到该分区

oem	给厂家使用的一个分区，存放厂家的 APP 或数据，oem.img 镜像会烧录到该分区；系统启动之后，该分区会被挂载到根文件系统/oem 目录
userdata	供最终用户使用的分区，存放用户的 APP 或数据；系统启动之后，该分区会被挂载到根文件系统/userdata 目录

表 5.2.1.1 各分区的作用介绍

5.2.2 配置

通过上小节分析可知，parameter.txt 文件中一共定义了 8 个分区，提供了每个分区的名字、起始地址以及分区大小，接下来我们需要手动配置瑞芯微开发工具，配置完之后如下图所示：

图 5.2.2.1 手动配置瑞芯微开发工具

这里需要注意几个点：

1.瑞芯微开发工具中地址数值的单位也是 sectors（扇区，一个扇区等于 512 字节）；

2.第一项对应的是 MiniLoaderAll.bin 镜像，**它的烧录地址不用配置，直接使用 0x0 即可，因为 MiniLoaderAll.bin 镜像有专门的烧录地址，无需用户配置**，而且它的名字一般都是 Loader（或小写 loader），不要去改动它；

3.上图中第二项对应的是分区表 parameter.txt，**同样它的地址也不用配置，直接使用 0x0 即可，因为 parameter.txt 文件不会烧录到 Flash 中，但会读取该文件定义的分区、去初始化 Flash 物理分区**；同样，它的名字为 parameter（或者大写 Parameter），不要去改动它，因为底层需要通过这个“**parameter**”名字来识别分区表文件。

4.除了 **MiniLoaderAll.bin** 和 **parameter.txt** 稍微特殊一点之外，其它镜像直接根据 parameter.txt 分区表定义的起始地址进行配置即可，名字尽量使用 parameter.txt 文件中所定义的分区名。

配置完成之后，我们还可以将这些配置信息导出、保存到一个.cfg 文件中，方便下次直接导入；导出的方法很简单，步骤如图 5.2.2.2~5.2.2.3 所示：

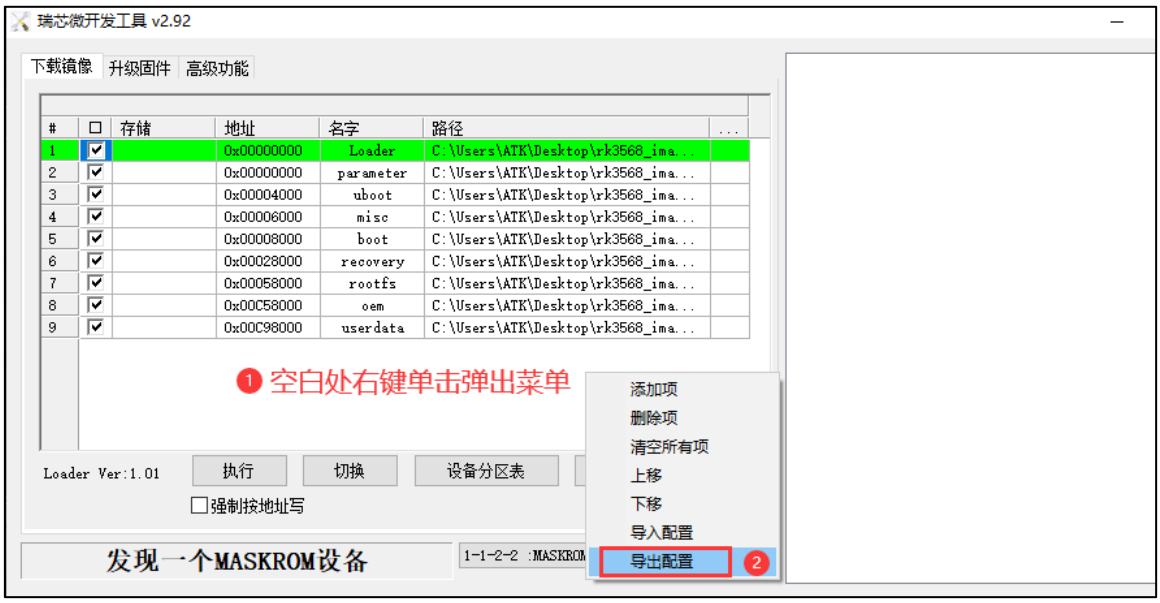


图 5.2.2.2 导出配置(1)

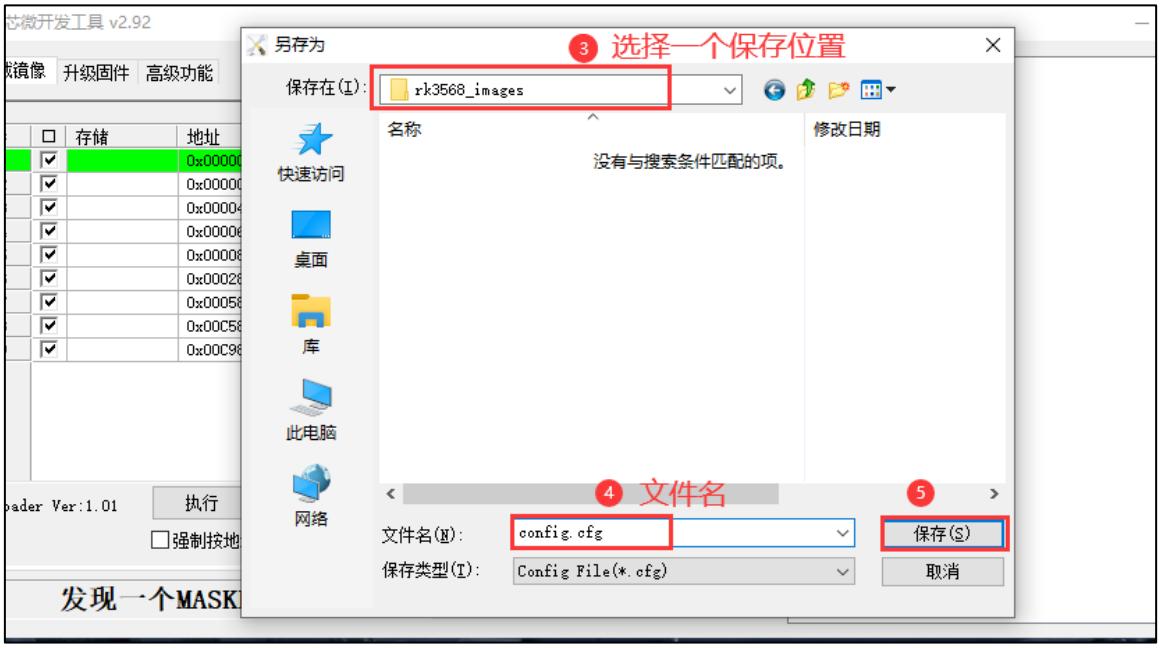


图 5.2.2.3 导出配置(2)

将配置信息导出、保存到 config.cfg 文件中，方便下次导入该配置信息。

5.2.3 烧录

开发板先连接好电源适配器以及 OTG，烧写之前，让开发板进入 Maskrom 或 Loader 模式。

譬如通过 Maskrom 模式烧写镜像，按住开发板上的 **UPDATE** 按键，然后给开发板上电或复位，此时设备便会进入 Maskrom 模式（瑞芯微开发工具会提示用户“**发现一个 MASKROM 设备**”），然后点击“**执行**”按钮烧录镜像：

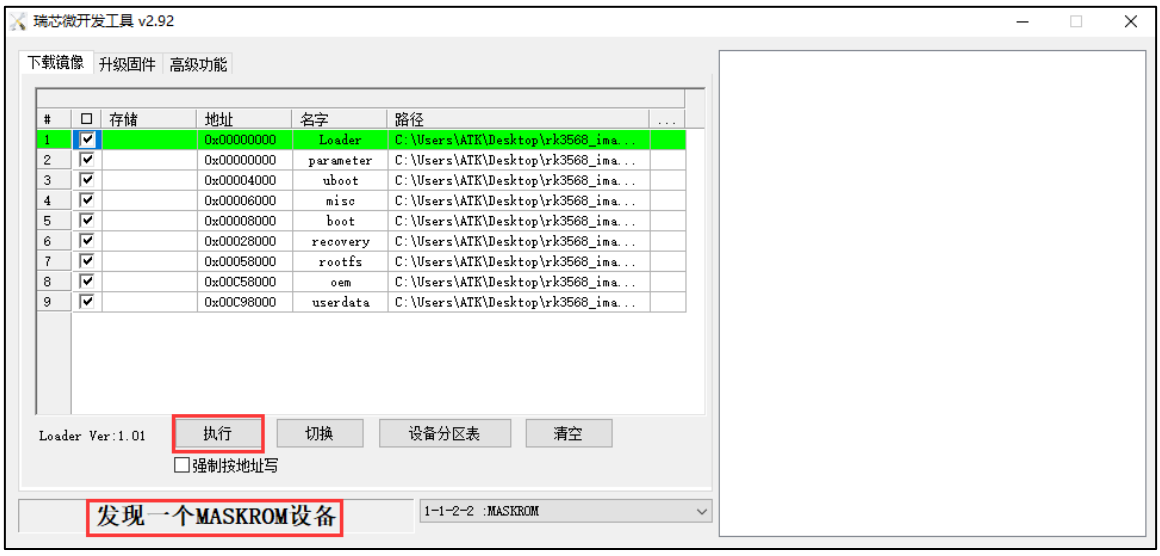


图 5.2.3.1 执行烧录

烧录过程中，右边空白处会输出 log 信息，如下所示：

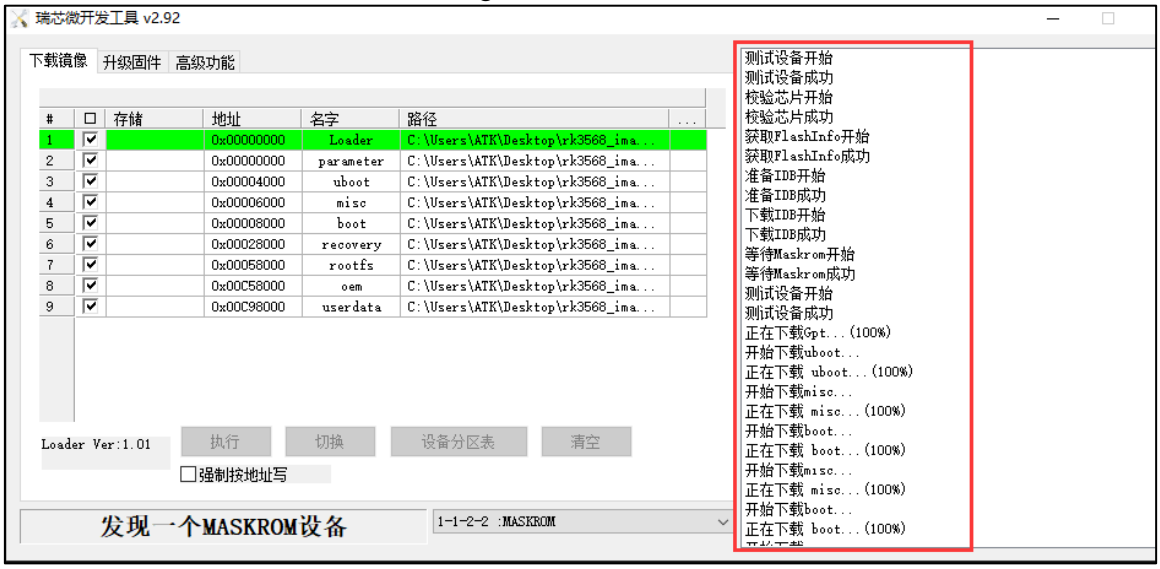


图 5.2.3.2 烧录过程 log 信息

也可单独烧录某个指定镜像，譬如单独烧录 boot.img 到 boot 分区，只需勾选对应的这一项即可，如下所示：

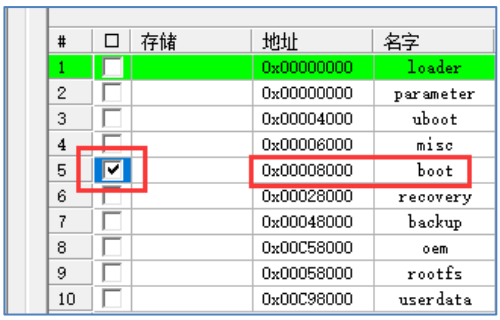


图 5.2.3.3 单独烧录 boot.img

然后点击“执行”按钮进行烧录即可。这在开发、调试过程中很有用，因为有时我们仅仅只是为了更新某个分区、只需将镜像烧录到该分区替换旧的镜像而已，无需重烧整个系统。

5.2.4 启动系统

烧录完成后会自动重启开发板，进入 Linux buildroot 系统，串口终端会输出信息：

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

图 5.2.4.1 进入系统

5.2.4 烧录 update.img

可以使用瑞芯微开发工具烧录 update.img 固件，首先将 4.4.5 小节生成的 update.img 固件拷贝到 Windows 下，然后通过瑞芯微开发工具将其烧录到开发板，烧录方法参考 2.9.3 小节。

5.3 Ubuntu 系统下烧写

本小节向用户介绍如何在 Ubuntu 下烧录镜像。在 Windows 下烧录，需要先将镜像从 Ubuntu 系统拷贝到 Windows 系统下，如此才可进行烧录。如果在 Ubuntu 下烧录，那么就不用拷贝镜像。

前面给大家讲过，在 Ubuntu 下可以使用 Linux_Upgrade_Tool 工具进行烧录，该工具集成在 SDK 中，路径为：**<SDK>/tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool**，如下所示：

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

图 5.3.1 Linux_Upgrade_Tool 目录

该目录下有两份 RK 提供的使用说明文档：《**命令行开发工具使用文档.pdf**》、《**Linux 开发工具使用手册_v1.32.pdf**》，关于 upgrade_tool 工具的详细使用方法请参考这两份文档。

upgrade_tool 有两种运行模式：**命令行模式**和**工具模式**，直接运行 upgrade_tool 命令，不加任何参数则会进入到工具模式，工具模式其实就是一个交互模式，在交互模式下用户可以输入指令、然后按回车执行该指令，那么就会触发该指令所对应的操作，譬如下载镜像、擦除镜像、读取设备信息等等，有点像命令行终端。

运行 upgrade_tool 命令时加入参数（譬如 UF UL DI DB 等等）则会进入命令行模式，所谓命令行模式，运行 upgrade_tool 命令后无法与用户进行交互，通过传入的参数来告诉 upgrade_tool 工具本次需要执行什么操作，譬如说烧写镜像、擦除等，执行完任务后就会退出。

5.3.1 将开发板连接到 Ubuntu

在 Ubuntu 下使用 upgrade_tool 工具烧写镜像之前，需要将开发板连接到 Ubuntu 系统。首先，连接好硬件（连接电源适配器以及 OTG 口），让开发板处于 Maskrom 或 Loader 模式下；在 Ubuntu 系统右下角可以看到有一个名字为“**Fuzhou Rockchip Rockusb Device**”的设备（需要将鼠标移动到相应图标上时才会显示出来），如下图所示：

图 5.3.1.1 Rockchip Rockusb Device

同样，在“**虚拟机→可移动设备**”下面也可以找到该设备，如下图所示：

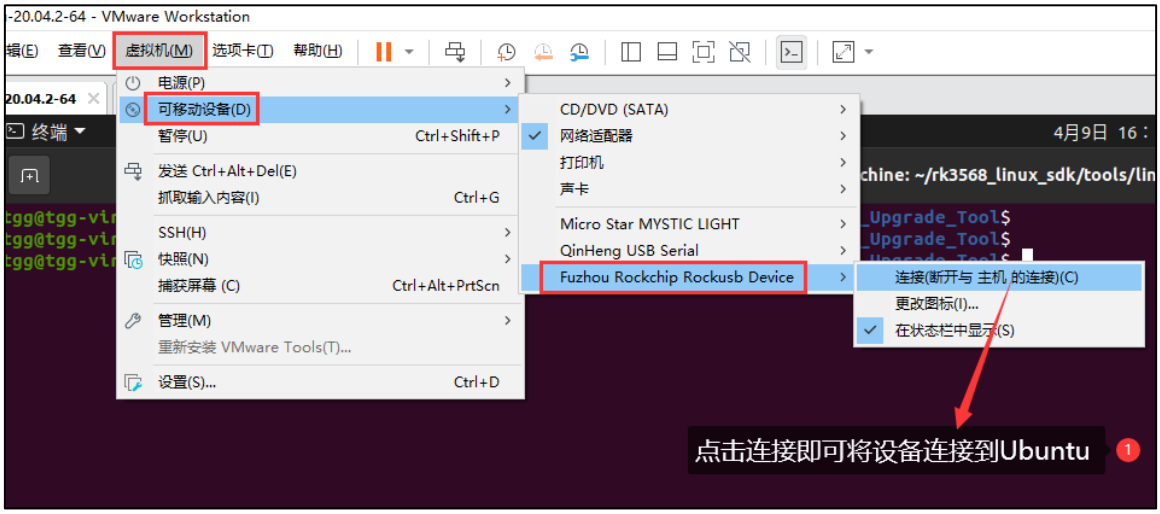


图 5.3.1.2 将设备连接到 Ubuntu(1)

也可以通过右键单击右下角的 Rockusb 设备图标，然后选择“**连接**”：

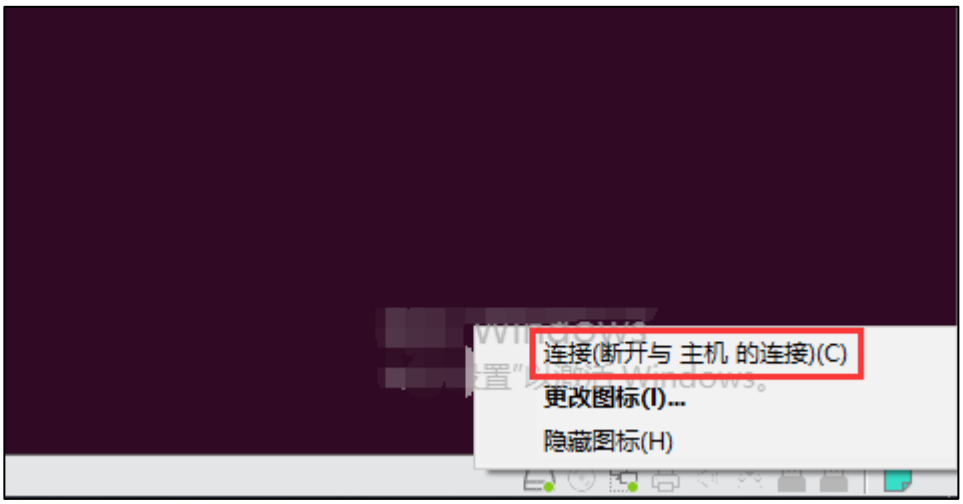


图 5.3.1.3 将设备连接到 Ubuntu(2)

通过这样操作之后，便可将开发板连接到 Ubuntu 系统。

5.3.2 使用 upgrade_tool 工具烧写

本小节向用户介绍如何使用 upgrade_tool 工具烧录镜像（只介绍命令行模式烧写）。为了方便操作，我们直接进入到<SDK>/rockdev 目录。在烧写之前，开发板需要处于 Maskrom 或 Loader 模式下。

upgrade_tool 工具支持很多指令，不同指令可以执行不同的操作，譬如 CD、LD、SD、UF、DI、DB、TD、RD 等等（**大小写都行**），有些指令需要带参数、而有些指令则不需要带参数，详情请参考<**命令行开发工具使用文档.pdf**>和<**Linux 开发工具使用手册_v1.32.pdf**>。

使用 UL 指令烧写 MiniLoaderAll.bin 镜像，使用 DI 指令烧写其它镜像（uboot.img、boot.img、oem.img、userdata.img、rootfs.img、misc.img 等）以及分区表文件 parameter.txt。

先烧写 MiniLoaderAll.bin 镜像，使用 UL 指令烧写 MiniLoaderAll.bin（**执行 upgrade_tool 命令时需要加入 sudo 获取到 root 用户权限，否则操作会失败！**）：

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

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/rockdev$ pwd
/home/tgg/rk3568_linux_sdk/rockdev
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/rockdev$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/rockdev$ sudo ../tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/upgrade_tool UL MiniLoaderAll.bin -noreset
[sudo] tgg 的密码:
Using /home/tgg/rk3568_linux_sdk/tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/config.ini
Program Log will save in the /root/upgrade_tool/log/
Loading loader...
Support Type:RK3568      Loader ver:1.01 Loader Time:2023-04-10 11:33:46
Upgrade loader ok.
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/rockdev$
```

图 5.3.2.1 烧写 MiniLoaderAll.bin

upgrade_tool 命令后面携带了 3 个参数，第一个参数表示需要执行的指令，UL 指令用于烧写 MiniLoaderAll.bin；第二个参数用于指定 MiniLoaderAll.bin 所在路径；第三个参数-noreset 表示烧写完 MiniLoaderAll.bin 之后不要复位开发板（不要复位设备）。

烧写完 MiniLoaderAll.bin 之后，接下来需要通过 DI 指令下载 parameter.txt 分区表：

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

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/rockdev$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/rockdev$ sudo ../tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/upgrade_tool DI -p parameter.txt
Using /home/tgg/rk3568_linux_sdk/tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/config.ini
Program Log will save in the /root/upgrade_tool/log/
directlba=1,first4access=1,gpt=1
Write gpt...
Write gpt ok.
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/rockdev$
```

图 5.3.2.2 下载分区表

接下来烧录其它镜像：

```
sudo ../tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/upgrade_tool DI -uboot uboot.img
sudo ../tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/upgrade_tool DI -misc misc.img
sudo ../tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/upgrade_tool DI -boot boot.img
```

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

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/rockdev$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/rockdev$ sudo ../tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/upgrade_tool DI -uboot uboot.img
Using /home/tgg/rk3568_linux_sdk/tools/linux/Linux_Upgrade_Tool/Linux_Upgrade_Tool/config.ini
Program Log will save in the /root/upgrade_tool/log/
directlba=1,first4access=1,gpt=1
Download uboot start...(0x00004000)
Download image ok.
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/rockdev$
```

图 5.3.2.3 烧录 uboot.img

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

图 5.3.2.4 烧录 misc.img

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

图 5.3.2.5 烧录 boot.img

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

图 5.3.2.6 烧录 recovery.img

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

图 5.3.2.7 烧录 oem.img

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

图 5.3.2.8 烧录 rootfs.img

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

图 5.3.2.9 烧录 userdata.img

“DI -<partition_name>”中的 partition_name 便是分区名，譬如-boot 就是 boot 分区、-oem 就是 oem 分区；使用 upgrade_tool 工具烧写镜像无需用户指定烧写地址，parameter.txt 文件已经定义了各分区的起始位置，也就是镜像的烧录地址。

如果执行命令出错，可以尝试复位、重启开发板，再次操作。

最后，当所有镜像全部烧录完成后，我们可以执行下面这条命令复位开发板、重新启动系统：

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

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

图 5.3.2.10 复位开发板

5.3.3 烧写 update.img

本小节介绍如何使用 upgrade_tool 工具烧录 update.img 固件。

开发板处于 Maskrom 或 Loader 模式下，执行如下命令烧录 update.img 固件（使用 UF 指令烧录 update.img）：

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

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

图 5.3.3.1 烧写 update.img

烧录完后会自动复位开发板。

5.3.4 擦除操作

本小节介绍如何使用 upgrade_tool 工具擦除开发板 Flash。

开发板处于 Maskrom 或 Loader 模式下，执行如下命令可以擦除 Flash 中的所有数据（使用 EF 指令）：

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

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

图 5.3.4.1 擦除所有数据

当然，也可以按地址进行扇区擦除，由用户指定擦除的起始位置和大小（使用 EL 指令）。

5.3.5 使用 rkflash.sh 脚本烧写

<SDK>/rkflash.sh 是 RK 提供的烧录脚本，我们可以直接使用这个 rkflash.sh 脚本进行烧录；当然，这个脚本也是调用了 upgrade_tool 工具执行烧录操作。

用法也非常简单，首先让开发板处于 Maskrom 或 Loader 模式下，直接运行 rkflash.sh 脚本即可将<SDK>/rockdev 目录下的镜像烧录到开发板（**同样也需要加入 sudo，否则操作会失败！**）：

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

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

图 5.3.5.1 执行 rkflash.sh 烧录所有镜像

执行上述命令会将 rockdev 目录下的 boot.img、MiniLoaderAll.bin、misc.img、oem.img、recovery.img、rootfs.img、uboot.img、userdata.img 烧写到开发板对应分区。

烧录完之后会自动复位开发板。

除了之外，还可单独烧录某个指定镜像，如下表所示：

命令	作用
sudo ./rkflash.sh all	烧录所有镜像
sudo ./rkflash.sh	烧录所有镜像，与上等价
sudo ./rkflash.sh loader	烧写 Loader（也就是 MiniLoaderAll.bin）
sudo ./rkflash.sh parameter	下载分区表 parameter.txt
sudo ./rkflash.sh uboot	烧写 uboot.img
sudo ./rkflash.sh boot	烧写 boot.img
sudo ./rkflash.sh recovery	烧写 recovery.img
sudo ./rkflash.sh misc	烧写 misc.img
sudo ./rkflash.sh oem	烧写 oem.img
sudo ./rkflash.sh userdata	烧写 userdata.img
sudo ./rkflash.sh rootfs	烧写根文件系统镜像 rootfs.img
sudo ./rkflash.sh update	烧写 update.img 镜像包
sudo ./rkflash.sh erase	擦除 Flash 存储的所有数据

表 5.3.5.1 rkflash.sh 脚本的用法

第六章 SDK 开发

本章向用户介绍如何对 RK3568 Linux SDK 进行开发，包括 u-boot 开发、Linux 内核开发、buildroot 根文件系统开发、Qt 应用开发、C/C++应用开发等；用户可以基于本 SDK 进行二次开发、软件定制，以适配自己的 Linux 产品；基于本 SDK，可以有效实现系统定制和应用移植开发，帮助用户快速开发、提高开发效率！

6.1 SDK 板级配置文件

本小节向用户介绍 SDK 的板级配置文件，SDK 板级配置文件中提供了一些必要的配置信息。对于 RK3568 平台，其板级配置文件位于**<SDK>/device/rockchip/rk356x/**目录，如下所示：

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

图 6.1.1 板级配置文件

该目录下有多个 BoardConfig-xxxx.mk 文件，这些.mk 文件便是板级配置文件。其中**BoardConfig-rk3568-atk-evb1-ddr4-v10.mk** 就是我们的 ATK-DLRK3568 开发板所使用的板级配置文件；我们在 SDK 根目录下执行“**./build.sh lunch**”时所列举出来的文件就是从<SDK>/device/rockchip/rk356x/目录来的，如下所示：

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

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

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

图 6.1.2 lunch 列出配置文件

这些.mk 文件其实是一个 sh 脚本文件，打开 BoardConfig-rk3568-atk-evb1-ddr4-v10.mk 配置文件来看一下里面的内容：

```
#!/bin/bash

# Target arch
export RK_ARCH=arm64
# Uboot defconfig
export RK_UBOOT_DEFCONFIG=rk3568
# Uboot image format type: fit(flattened image tree)
export RK_UBOOT_FORMAT_TYPE=fit
# Kernel defconfig
export RK_KERNEL_DEFCONFIG=rockchip_linux_defconfig
```

```
# Kernel defconfig fragment
export RK_KERNEL_DEFCONFIG_FRAGMENT=
# Kernel dts
export RK_KERNEL_DTS=rk3568-atk-evb1-ddr4-v10-linux
# boot image type
export RK_BOOT_IMG=boot.img
# kernel image path
export RK_KERNEL_IMG=kernel/arch/arm64/boot/Image
# kernel image format type: fit(flattened image tree)
export RK_KERNEL_FIT_ITS=boot.its
# parameter for GPT table
export RK_PARAMETER=parameter-buildroot-fit.txt
# Buildroot config
export RK_CFG_BUILDROOT=rockchip_rk3568
# Recovery config
export RK_CFG_RECOVERY=rockchip_rk356x_recovery
# Recovery image format type: fit(flattened image tree)
export RK_RECOVERY_FIT_ITS=boot4recovery.its
# ramboot config
export RK_CFG_RAMBOOT=
# Pcba config
export RK_CFG_PCBA=
# Build jobs
export RK_JOBS=24
# target chip
export RK_TARGET_PRODUCT=rk356x
# Set rootfs type, including ext2 ext4 squashfs
export RK_ROOTFS_TYPE=ext4
# Set debian version (debian10: buster, debian11: bullseye)
export RK_DEBIAN_VERSION=buster
# yocto machine
export RK_YOCTO_MACHINE=rockchip-rk3568-evb
# rootfs image path
export RK_ROOTFS_IMG=rockdev/rootfs.${RK_ROOTFS_TYPE}
# Set ramboot image type
export RK_RAMBOOT_TYPE=
# Set oem partition type, including ext2 squashfs
export RK_OEM_FS_TYPE=ext2
# Set userdata partition type, including ext2, fat
export RK_USERDATA_FS_TYPE=ext2
#OEM config
export RK_OEM_DIR=oem_normal
# OEM build on buildroot
#export RK_OEM_BUILDIN_BUILDROOT=YES
#userdata config
export RK_USERDATA_DIR=userdata_normal
#misc image
export RK_MISC=wipe_all-misc.img
#choose enable distro module
export RK_DISTRO_MODULE=
# Define pre-build script for this board
export RK_BOARD_PRE_BUILD_SCRIPT=app-build.sh
```

这个脚本中通过 export 导出一些环境变量，如下所示：

RK_ARCH：用于指定目标架构，rk3568 对应 arm64；

RK_UBOOT_DEFCONFIG：用于指定 U-Boot 的 defconfig 配置文件；rk3568_defconfig。

RK_UBOOT_FORMAT_TYPE：用于指定 uboot.img 镜像的格式，rk3568 平台默认使用的是 fit（flattened image tree）格式镜像；

RK_KERNEL_DEFCONFIG：用于指定 Linux Kernel（内核）的 defconfig 配置文件；rockchip_linux_defconfig。

RK_KERNEL_DEFCONFIG_FRAGMENT：用于指定 Linux 内核的 defconfig fragment，对于 rk3568 来说是空置；

RK_KERNEL_DTS：用于指定内核设备树文件；rk3568-atk-evb1-ddr4-v10-linux.dts。

RK_BOOT_IMG：设置为 boot.img 即可；

RK_KERNEL_IMG：用于指定内核镜像的路径；kernel/arch/arm64/boot/Image。

RK_KERNEL_FIT_ITS：rk3568 平台 Linux 系统使用的启动镜像 boot.img 也是 FIT 格式镜像，FIT 使用 its（image source file）文件来描述 image 的信息，RK_KERNEL_FIT_ITS 用于指定这个 its 文件，这个 its 文件必须要存放在<SDK>/device/rockchip/rk356x/目录下；boot.its。

RK_PARAMETER：用于指定分区表文件，也就是前面给大家介绍的 parameter.txt 文件；

RK_CFG_BUILDROOT：用于指定 buildroot 根文件系统（普通模式）的 defconfig 配置文件；rockchip_rk3568_defconfig。

RK_CFG_RECOVERY：用于指定 recovery 模式下根文件系统（recovery 模式下使用 ramdisk 根文件系统）的 defconfig 配置文件；rockchip_rk356x_recovery_defconfig。

RK_RECOVERY_FIT_ITS：用于指定 recovery.img 镜像对应的 its 文件。recovery.img 也是 FIT 格式镜像，需要使用 its 文件来描述 image 的信息；boot4recovery.its。

RK_CFG_RAMBOOT：默认是空置，然不知其意；

RK_CFG_PCBA：用于指定 PCBA 的 defconfig 配置文件；

RK_JOBS：用于指定 make 编译时的线程数，譬如：make -j24；

RK_TARGET_PRODUCT：用于指定目标产品名，对于 rk3568 平台来说，默认将其设置为 rk356x；

RK_ROOTFS_TYPE：用于指定根文件系统的类型，譬如 ext2、ext4；

RK_DEBIAN_VERSION：用于指定 Debian 的版本，debian10: buster, debian11: bullseye，默认使用的是 Debian 10，不要去改动它；

RK_YOCTO_MACHINE：编译 yocto 时，用于指定 machine；

RK_ROOTFS_IMG：用于指定根文件系统镜像的路径；

RK_RAMBOOT_TYPE：默认是空置，然不知其意；

RK_OEM_FS_TYPE：用于指定 oem 分区的类型，保持默认就行；

RK_USERDATA_FS_TYPE：用于指定 userdata 分区的类型，保持默认就行；

RK_OEM_DIR：用于指定 oem 对应的文件夹，<SDK>/device/rockchip/oem 目录下有多个 oem_xxx 文件夹，这些文件夹中存放了厂家的 APP 或数据等，最终会编译成 oem.img 镜像；

RK_USERDATA_DIR：用于指定 userdata 对应的文件夹，<SDK>/device/rockchip/userdata 目录下有多个 userdata_xxx 文件夹，这些文件夹中存放了最终用户的 APP 或数据等，最终会编译成 userdata.img 镜像；

RK_MISC：用于指定 misc 镜像。编译完 SDK 后，生成的<SDK>/rockdev/misc.img 镜像其实就是 RK_MISC 所指定的这个文件，只不过是进行了重命名而已；RK_MISC 所指定的 misc 镜像必须要存放在<SDK>/device/rockchip/rockimg 目录下；

RK_DISTRO_MODULE：未使用到；

RK_BOARD_PRE_BUILD_SCRIPT：

关于这个板级配置文件就讲这么多，用户可以在**<SDK>/device/rockchip/rk356x** 目录下添加自己的板级配置文件，根据实际情况对配置文件中的变量进行修改、或添加新的变量。

6.1 U-Boot 开发

U-Boot 源码在**<SDK>/u-boot** 目录，如下所示：

```
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$
```

图 6.1.1 U-Boot 源码目录

RK 提供了一份文档详细向用户介绍了 Rockchip 平台 U-Boot 所涉及到的知识点、技术点，包括 Rockchip 平台 U-Boot 基础简介、U-Boot 架构、U-Boot 启动流程、U-Boot 系统模块、驱动模块、Kernel-DTB、AB 系统、AVB 安全启动、TPL、SPL、U-Boot 快捷键等等，内容非常多，对 U-Boot（Rockchip 平台 U-Boot）不太熟悉的用户建议一定要去看看；该文档的路径为：**<SDK>/docs/Common/UBOOT/Rockchip_Developer_Guide_UBoot_Nextdev_CN.pdf**。同样，ATK-DLRK3568 开发板资料包中也有提供，路径为：**开发板光盘 A 盘-基础资料→08、RK 官方文档→Linux→Common→UBOOT→Rockchip_Developer_Guide_UBoot_Nextdev_CN.pdf**。

6.1.1 U-Boot 的设备树

U-Boot 中，RK3568 的设备树文件是<U-Boot>/arch/arm/dts/rk3568-evb.dts，该设备树文件包含了 rk3568.dtsi 和 rk3568-u-boot.dtsi，包含关系如下：

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

原生的 U-Boot 只支持 U-Boot 自己的 DTB，RK 平台在原生 U-Boot 基础上增加了 kernel DTB 机制的支持，即 U-Boot 会使用 kernel DTB 去初始化外设。这样设计的目的主要是为了兼容外设板级差异，譬如 power、clock、display 等。

U-Boot 设备树负责初始化存储、调试串口等基础外设；而 kernel 设备树初始化存储、调试串口之外的外设，譬如 LCD 显示、千兆网等。执行 U-Boot 代码时先用 U-Boot 的设备树完成存储、调试串口的初始化操作，然后从存储上加载 kernel 的设备树并转而使用这份设备树继续初始化其余外设。

所以用户一般不需要去修改 U-Boot 的设备树文件（除非更换调试串口）。

6.1.2 U-Boot 编译

U-Boot 源码目录下提供了一个编译脚本 make.sh，可以直接使用该脚本编译 U-Boot 源码，譬如在 U-Boot 源码目录下执行如下命令编译 U-Boot：

```
./make.sh rk3568
```

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/u-boot$ ./make.sh rk3568
## make rk3568_defconfig -j24
#
# configuration written to .config
#
scripts/kconfig/conf  --silentoldconfig Kconfig
  CHK     include/config.h
  CFG     u-boot.cfg
  GEN     include/autoconf.mk.dep
  CFG     spl/u-boot.cfg
  CFG     tpl/u-boot.cfg
  GEN     include/autoconf.mk
  GEN     tpl/include/autoconf.mk
  GEN     spl/include/autoconf.mk
  CHK     include/config/uboot.release
  CHK     include/generated/timestamp_autogenerated.h
  UPD     include/generated/timestamp_autogenerated.h
  CHK     include/config.h
  CFG     u-boot.cfg
  CHK     include/generated/version_autogenerated.h
  CHK     include/generated/generic-asm-offsets.h
  CHK     include/generated/asm-offsets.h
  HOSTCC  tools/mkenvimage.o
  HOSTCC  tools/fit_image.o
  HOSTCC  tools/image-host.o
  HOSTCC  tools/dumpimage.o
  HOSTCC  tools/mkimage.o
  HOSTCC  tools/rockchip/boot_merger.o
  HOSTCC  tools/rockchip/loaderimage.o
  HOSTLD  tools/mkenvimage
  HOSTLD  tools/loaderimage
  HOSTLD  tools/dumpimage
  HOSTLD  tools/mkimage
tools/rockchip/boot_merger.c: In function ‘main’:
tools/rockchip/boot_merger.c:895:11: warning: array subscript 20 is outside array bounds of ‘char[20]’ [-Warray-bounds]
  895 |  str[len] = 0;
      |  ~~~~~~~~~^~~
tools/rockchip/boot_merger.c:933:7: note: while referencing ‘name’
  933 |  char name[MAX_NAME_LEN];
      |       ^~~~
```

图 6.1.2.1 编译 U-Boot 源码(1)

```
 Configuration 0 (conf)
  Description:  rk3568-evb
  Kernel:       unavailable
  Firmware:     atf-1
  FDT:          fdt
  Loadables:    uboot
                atf-2
                atf-3
                atf-4
                atf-5
                atf-6
                optee
********boot_merger ver 1.2********
Info:Pack loader ok.
pack loader okay! Input: /home/tgg/rk3568_linux_sdk/rkbin/RKBOOT/RK3568MINIALL.ini
/home/tgg/rk3568_linux_sdk/u-boot

Image(no-signed, version=0): uboot.img (FIT with uboot, trust...) is ready
Image(no-signed): rk356x_spl_loader_v1.13.112.bin (with spl, ddr...) is ready
pack uboot.img okay! Input: /home/tgg/rk3568_linux_sdk/rkbin/RKTRUST/RK3568TRUST.ini

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

图 6.1.2.2 编译 U-Boot 源码(2)

编译完成后将会生成 **uboot.img** 和 **rk356x_spl_loader_v1.13.112.bin** 两个镜像文件。如下所示：

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

图 6.1.2.3 U-Boot 编译后生成的镜像

当然，除了这两个镜像之外，还生成了很多的.bin 镜像，譬如 bl31_xxx.bin、tee.bin、u-boot.bin、u-boot-dtb.bin、u-boot-nodtb.bin 等等，这些镜像都是中间产物，最终烧录到开发板只有 uboot.img 和 rk356x_spl_loader_v1.13.112.bin，接下来讲一下这两个镜像文件。

1. uboot.img 镜像

前面提到过，uboot.img 是由多个镜像合并而成，包括 u-boot 镜像、u-boot dtb 以及 trust 镜像（ARM Trusted Firmware + OP-TEE）。uboot.img 是一种 FIT（**flattened image tree**）格式镜像，支持任意多个 image 打包和校验。使用 file 命令查看 uboot.img，如下所示：

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/u-boot$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/u-boot$ file uboot.img
uboot.img: Device Tree Blob version 17, size=3584, boot CPU=0, string block size=197, DT structure block size=2824
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/u-boot$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/u-boot$
```

FIT 使用 its（**image source file**）文件来描述 image 的信息，最后通过 mkimage 工具生成 itb（**flattened image tree blob**）镜像，那么这个 itb 镜像其实就是 uboot.img 镜像，uboot.img 镜像通常含有多份 itb 镜像，如下所示：

```
uboot.img = uboot.itb * N（N 一般是 2）
```

这种设计也是为了避免如果第一份镜像启动失败、还可以尝试使用第二份镜像启动。

U-Boot 编译成功后，U-Boot 源码目录下会生成很多.bin 镜像以及.dtb 镜像：

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

图 6.1.2.4 U-Boot 目录下的 bin 镜像

整理一下，如下表所示：

镜像	说明
u-boot.bin	u-boot 本身镜像，包含了 u-boot dtb
u-boot.dtb	u-boot dtb
u-boot-dtb.bin	u-boot 本身镜像，包含了 u-boot dtb，与 u-boot.bin 等价、相同
u-boot-nodtb.bin	u-boot 本身镜像，但不包含 u-boot dtb
bl31_0x00040000.bin	都是 ARM Trusted Firmware 固件（ATF）
bl31_0x00068000.bin	
bl31_0x0006a000.bin	
bl31_0xfdcc1000.bin	
bl31_0xfdcce000.bin	
bl31_0xfdcd0000.bin	
tee.bin	OP-TEE 固件

表 6.1.2.1 <U-Boot>目录镜像介绍

uboot.img 镜像最终由 **u-boot-nodtb.bin**、**u-boot.dtb**、**bl31_0x00040000.bin**、**bl31_0x00068000.bin**、**bl31_0x0006a000.bin**、**bl31_0xfdcc1000.bin**、**bl31_0xfdcce000.bin**、**bl31_0xfdcd0000.bin**、**tee.bin** 这些镜像合并而成。

uboot.img 是 FIT 格式镜像，使用 its 文件来描述 image 的信息，最终通过**<U-Boot>/tools/mkimage** 工具生成 itb 镜像；its 文件和生成的 itb 文件都在<U-Boot>/fit 目录下：

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

图 6.1.2.5 fit 目录下的文件

u-boot.its 文件语法规则与设备树 DTS 的语法规则相同，非常灵活！

u-boot.its 文件中描述了有哪些镜像会参与合并成 uboot.itb，以及这些镜像的路径等信息：

```
/dts-v1/;

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

        images {

                uboot {
                        description = "U-Boot";
                        data = /incbin/("u-boot-nodtb.bin");
                        type = "standalone";
                        arch = "arm64";
                        os = "U-Boot";
                        compression = "none";
                        load = <0x00a00000>;
                        hash {
                                algo = "sha256";
                        };
                };
                atf-1 {
                        description = "ARM Trusted Firmware";
                        data = /incbin/("./bl31_0x00040000.bin");
                        type = "firmware";
                        arch = "arm64";
                        os = "arm-trusted-firmware";
                        compression = "none";
                        load = <0x00040000>;
                        hash {
                                algo = "sha256";
                        };
                };
                atf-2 {
                        description = "ARM Trusted Firmware";
                        data = /incbin/("./bl31_0xfdcc1000.bin");
                        type = "firmware";
                        arch = "arm64";
                        os = "arm-trusted-firmware";
                        compression = "none";
                        load = <0xfdcc1000>;
                        hash {
                                algo = "sha256";
                        };
                };
                atf-3 {
                        description = "ARM Trusted Firmware";
                        data = /incbin/("./bl31_0x0006a000.bin");
                        type = "firmware";
                        arch = "arm64";
                        os = "arm-trusted-firmware";
                        compression = "none";
                        load = <0x0006a000>;
                        hash {
                                algo = "sha256";
                        };
                };
```

```
                atf-4 {
                        description = "ARM Trusted Firmware";
                        data = /incbin/("./bl31_0xfdcd0000.bin");
                        type = "firmware";
                        arch = "arm64";
                        os = "arm-trusted-firmware";
                        compression = "none";
                        load = <0xfdcd0000>;
                        hash {
                                algo = "sha256";
                        };
                };
                atf-5 {
                        description = "ARM Trusted Firmware";
                        data = /incbin/("./bl31_0xfdcce000.bin");
                        type = "firmware";
                        arch = "arm64";
                        os = "arm-trusted-firmware";
                        compression = "none";
                        load = <0xfdcce000>;
                        hash {
                                algo = "sha256";
                        };
                };
                atf-6 {
                        description = "ARM Trusted Firmware";
                        data = /incbin/("./bl31_0x00068000.bin");
                        type = "firmware";
                        arch = "arm64";
                        os = "arm-trusted-firmware";
                        compression = "none";
                        load = <0x00068000>;
                        hash {
                                algo = "sha256";
                        };
                };
                optee {
                        description = "OP-TEE";
                        data = /incbin/("tee.bin");
                        type = "firmware";
                        arch = "arm64";
                        os = "op-tee";
                        compression = "none";

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

        configurations {
                default = "conf";
                conf {
                        description = "rk3568-evb";
                        rollback-index = <0x0>;
                        firmware = "atf-1";
```

```
                        loadables = "uboot", "atf-2", "atf-3", "atf-4", "atf-5",
"atf-6", "optee";

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

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

由此可知，its 文件的语法规则与 DTS 是完全相同的，并无差别！详情可以参考**<U-Boot>/doc/uImage.FIT/**目录下的文档。

对于 ARM Trusted Firmware 以及 OP-TEE，它们是闭源的，RK 只提供了二进制镜像文件，并没提供源码。ARM Trusted Firmware 固件对应**<U-Boot>/bl31.elf**、OP-TEE 固件对应**<U-Boot>/tee.bin**，如下所示：

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

图 6.1.2.6 ATF 固件和 OP-TEE 固件

编译 U-Boot 源码之前，这两个镜像是不存在的，编译之后才会出现；实际上来自于**<SDK>/rkbin/bin/rk35/rk3568_bl31_v1.33.elf** 和**<SDK>/rkbin/bin/rk35/rk3568_bl32_v2.08.bin**这两个镜像文件。打包 uboot.itb 的过程中，会将<SDK>/rkbin/bin/rk35/rk3568_bl31_v1.33.elf 拷贝到<U-Boot>/bl31.elf，将<SDK>/rkbin/bin/rk35/rk3568_bl32_v2.08.bin 拷贝到<U-Boot>/tee.bin。

bl31.elf 固件并不是直接打包进 uboot.itb，而是将 bl31.elf 分解成多个 bl31_xxx.bin 文件，最终将这些 bl31_xxx.bin 镜像打包进 uboot.itb。

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/u-boot$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/u-boot$ ls -l bl31_0x*
-rw-rw-r-- 1 tgg tgg 163840 4月  11 11:07 bl31_0x00040000.bin
-rw-rw-r-- 1 tgg tgg   7732 4月  11 11:07 bl31_0x00068000.bin
-rw-rw-r-- 1 tgg tgg  20267 4月  11 11:07 bl31_0x0006a000.bin
-rw-rw-r-- 1 tgg tgg  40960 4月  11 11:07 bl31_0xfdcc1000.bin
-rw-rw-r-- 1 tgg tgg   8192 4月  11 11:07 bl31_0xfdcce000.bin
-rw-rw-r-- 1 tgg tgg   8192 4月  11 11:07 bl31_0xfdcd0000.bin
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/u-boot$
```

图 6.1.2.7 bl31_xxx.bin 固件

2. rk356x_spl_loader_v1.13.112.bin 镜像

再来讲一下 rk356x_spl_loader_v1.13.112.bin，这个镜像文件就是前面给大家介绍的 MiniLoaderAll.bin 镜像（由 rk356x_spl_loader_v1.13.112.bin 重命名而来）。前面提到过，MiniLoaderAll.bin 是运行在 RK3568 平台 U-Boot 之前的一段 Loader 代码（也就是比 U-Boot 更早阶段的 Loader）。

MiniLoaderAll.bin 由两部分构成：**TPL(Tiny Program Loader) + SPL(Secondary Program Loader)**构成。

TPL 运行在 SRAM 中（片内内存），由 rk3568 芯片内部所固化的 Maskrom（BootROM）代码引导启动；其作用是负责完成 DRAM 的初始化工作、并启动 SPL；SPL 运行在 DDR，SPL 的作用是负责完成系统的 lowlevel 初始化、完成 uboot.img 的加载和引导工作。

TPL、SPL 分别有两种实现方案：**开源版本**和**闭源版本**。

闭源版本

RK 向用户提供了 tpl 和 spl 的二进制镜像文件，不提供其对应的源码，其所在路径如下：

<SDK>/rkbin/bin/rk35/rk3568_ddr_1560MHz_v1.13.bin ----------> tpl
<SDK>/rkbin/bin/rk35/rk356x_spl_v1.12.bin ----------> spl

<SDK>/rkbin/bin 目录下存放了很多 RK 提供的二进制文件，这些二进制文件都是不开源的，所以只有二进制文件。

开源版本

tpl 和 spl 镜像也可以通过 u-boot 源码编译出来，既然是 U-Boot 源码的一部分，自然是开源的。使用“./make.sh rk3568”命令编译完 U-Boot 源码后，除了会生成 u-boot 镜像之外，还会编译生成 spl 以及 tpl 镜像，spl 镜像位于**<U-Boot>/spl** 目录下：

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/u-boot$ cd spl/
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/u-boot/spl$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/u-boot/spl$ ls
arch    cmd     disk     dts  fs       lib         u-boot-spl      u-boot-spl.dtb      u-boot-spl.lds  u-boot-spl-nodtb.bin
board   common  drivers  env  include  u-boot.cfg  u-boot-spl.bin  u-boot-spl-dtb.bin  u-boot-spl.map  u-boot-spl.sym
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/u-boot/spl$
```

图 6.1.2.8 spl 目录下的文件

该目录下也存在很多镜像文件：

镜像	说明
u-boot-spl	elf 格式的 spl 镜像
u-boot-spl.bin	spl 镜像，包含 spl dtb
u-boot-spl.dtb	spl dtb
u-boot-spl-dtb.bin	spl 镜像，包含 spl dtb，与 u-boot-spl.bin 是一样的
u-boot-spl-nodtb.bin	spl 镜像，不包含 spl dtb

表 6.1.2.2 spl 目录镜像介绍

tpl 镜像位于**<U-Boot>/tpl** 目录下：

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/u-boot/tpl$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/u-boot/tpl$ ls
arch    common  drivers  fs       u-boot.cfg      u-boot-tpl      u-boot-tpl.map      u-boot-tpl.sym
board   disk    dts      include  u-boot-spl.lds  u-boot-tpl.bin  u-boot-tpl-nodtb.bin
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/u-boot/tpl$
```

图 6.1.2.9 tpl 目录下的文件

在 RK 的文档中，将 RK 提供的闭源 TPL 镜像 <SDK>/rkbin/bin/rk35/rk3568_ddr_1560MHz_v1.13.bin 称为 **ddr bin**，将闭源 SPL 镜像 <SDK>/rkbin/bin/rk35/rk356x_spl_v1.12.bin 称为 **miniloader**。

rk356x_spl_loader_v1.13.112.bin 镜像文件是由**<U-Boot>/tools/boot_merger** 工具制作而成，该工具由 RK 提供。使用 boot_merger 工具制作 rk356x_spl_loader_v1.13.112.bin 镜像时需要提供一个.ini 文件，.ini 文件用于描述 image 的信息；在**<SDK>/rkbin/RKBOOT/**目录下有很多.ini 文件，如下图所示：

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

图 6.1.2.10 .ini 文件

对于 RK3568 平台来说，它默认使用的是 **RK3568MINIALL.ini**。文件内容如下：

```
[CHIP_NAME]
NAME=RK3568
[VERSION]
MAJOR=1
MINOR=1
[CODE471_OPTION]
NUM=1
Path1=bin/rk35/rk3568_ddr_1560MHz_v1.13.bin
Sleep=1
[CODE472_OPTION]
NUM=1
Path1=bin/rk35/rk356x_usbplug_v1.14.bin
[LOADER_OPTION]
NUM=2
LOADER1=FlashData
LOADER2=FlashBoot
FlashData=bin/rk35/rk3568_ddr_1560MHz_v1.13.bin
FlashBoot=bin/rk35/rk356x_spl_v1.12.bin
[OUTPUT]
PATH=rk356x_spl_loader_v1.13.112.bin
[SYSTEM]
NEWIDB=true
[FLAG]
471_RC4_OFF=true
RC4_OFF=true
```

FlashData 指定了 tpl 镜像的路径；FlashBoot 指定了 spl 镜像的路径。所以默认情况下，执行“./make.sh rk3568”命令生成的 rk356x_spl_loader_v1.13.112.bin 使用的是 RK 闭源的 ddr bin（<SDK>/rkbin/bin/rk35/rk3568_ddr_1560MHz_v1.13.bin）和 RK 闭源的 miniloader（<SDK>/rkbin/bin/rk35/rk356x_spl_v1.12.bin）。

除了使用 RK 闭源的 miniloader 之外，还可以使用 U-Boot 编译出来的开源 spl（**注意，不能用 U-Boot 编译出来的 tpl 镜像，我测试过直接启动不了，估计是 U-Boot TPL 部分代码支持**

不够完善），执行如下命令可以用 U-Boot 生成的 spl 镜像（<U-Boot>/spl/u-boot-spl.bin）替换 RK 闭源的 miniloader 去制作 rk356x_spl_loader_v1.13.112.bin：

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

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/u-boot$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/u-boot$ ./make.sh --spl
********boot_merger ver 1.2********
Info:Pack loader ok.
pack loader(SPL) okay! Input: /home/tgg/rk3568_linux_sdk/rkbin/RKBOOT/RK3568MINIALL.ini

/home/tgg/rk3568_linux_sdk/u-boot
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/u-boot$
```

图 6.1.2.11 使用开源 SPL 生成 MiniLoaderAll.bin

默认情况： MiniLoaderAll.bin = 闭源 ddr bin + 闭源 miniloader；

执行“./make.sh --spl”后：MiniLoaderAll.bin = 闭源 ddr bin + 开源 SPL。

make.sh 脚本还支持很多其它的参数，执行“**./make.sh -h**”可以查看它的使用帮助信息，大家自己去捣鼓。

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/u-boot$ ./make.sh -h

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

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

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

Example:

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

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

3. Debug:
        ./make.sh elf                       --- dump elf file with -D(default)
        ./make.sh elf-S                     --- dump elf file with -S
        ./make.sh elf-d                     --- dump elf file with -d
        ./make.sh elf-*                     --- dump elf file with -*
        ./make.sh <no reloc_addr>           --- unwind address(no relocated)
        ./make.sh <reloc_addr-reloc_off>    --- unwind address(relocated)
        ./make.sh map                       --- cat u-boot.map
        ./make.sh sym                       --- cat u-boot.sym
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/u-boot$
```

图 6.1.2.12 make.sh 脚本帮助信息

3. 镜像启动顺序

这里讲一下 RK3568 平台镜像的启动顺序。涉及到 Trust，目前 Rockchip 的 64 位 SoC 平台上使用的是 **ARM Trusted Firmware** + **OP-TEE** 的组合来实现 Trust，32 位 SoC 平台上使用的是 OP-TEE。

ARM Trusted Firmware 的体系架构里将整个系统分成四种安全等级，分别为：EL0、EL1、EL2、EL3。将整个安全启动的流程阶段定义为：BL1、BL2、BL31、BL32、BL33，其中 ARM Trusted Firmware 自身的源代码里提供了 BL1、BL2、BL31 的功能。Rockchip 平台仅使用了其

中的 BL31 的功能；对于 BL1 和 BL2，RK 有自己的一套实现方案。所以在 Rockchip 平台上，我们一般也可以“默认”ARM Trusted Firmware 指的就是 BL31，而 BL32 使用的则是 OP-TEE。

如果把上述这种阶段定义映射到 Rockchip 平台各级固件上，对应关系为：Maskrom（RK 芯片内部固化的引导代码，也叫 BootROM，BL1）、MiniLoaderAll.bin（BL2）、Trust（BL31：ARM Trusted Firmware + BL32：OP-TEE）、U-Boot（BL33）。

所以 Linux 系统的镜像启动顺序为：

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

还可以进行细分：

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

这个启动流程通过打印信息就可以分析出来，以下就是 ATK-DLRK3568 开发板上电启动时的打印信息：

```
# 执行 ddr bin，对开发板的 DDR 进行初始化
DDR Version V1.13 20220218
In
ddrconfig:15
DDR4, 324MHz
BW=32 Col=10 Bk=4 BG=2 CS0 Row=16 CS=1 Die BW=16 Size=2048MB
tdqss: cs0 dqs0: 24ps, dqs1: -48ps, dqs2: -96ps, dqs3: -96ps,

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

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

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

change to: 1560MHz(final freq)
clk skew:0x87
PHY drv:clk:37,ca:37,DQ:37,odt:139
vrefinner:50%, vrefout:61%
dram drv:34,odt:120
cs 0:
the read training result:
DQS0:0x32, DQS1:0x2c, DQS2:0x27, DQS3:0x30,
min  : 0xf  0xe  0xd  0x8  0x3  0x2  0x5  0x5 , 0x7  0x5  0x1  0x3  0x5  0x7  0x9
0x9 ,
      0x5  0x8  0x9  0x7  0x5  0x3  0x2  0x5 , 0xa  0x6  0x5  0x2  0x7  0xb  0x7
0x9 ,
mid  :0x28 0x27 0x27 0x21 0x1b 0x1b 0x1e 0x1c ,0x1f 0x1d 0x19 0x1c 0x1e 0x1f 0x21
0x1e ,
     0x1d 0x20 0x20 0x1e 0x1d 0x1c 0x1a 0x1d ,0x23 0x1e 0x1e 0x1a 0x20 0x23 0x20
0x21 ,
max  :0x41 0x41 0x41 0x3b 0x34 0x35 0x37 0x34 ,0x38 0x36 0x32 0x36 0x37 0x37 0x39
0x34 ,
     0x36 0x39 0x37 0x36 0x36 0x35 0x33 0x35 ,0x3c 0x37 0x38 0x32 0x39 0x3c 0x39
0x39 ,
range:0x32 0x33 0x34 0x33 0x31 0x33 0x32 0x2f ,0x31 0x31 0x31 0x33 0x32 0x30 0x30
0x2b ,
     0x31 0x31 0x2e 0x2f 0x31 0x32 0x31 0x30 ,0x32 0x31 0x33 0x30 0x32 0x31 0x32
0x30 ,
the write training result:
DQS0:0x8b, DQS1:0x7e, DQS2:0x74, DQS3:0x74,
```

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

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

```
INFO:    SPSR = 0x3c9

# 进入到 U-Boot，开始执行 U-Boot 代码
U-Boot 2017.09-gc2d9f80c05-220621 #tgg (Apr 10 2023 - 11:31:38 +0800)

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

dwmmc@fe2b0000: 1, dwmmc@fe2c0000: 2, sdhci@fe310000: 0
Bootdev(atags): mmc 0
MMC0: HS200, 200Mhz
PartType: EFI
DM: v1
boot mode: None
……
……
```

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

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

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

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

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

图 6.1.2.13 TPL(ddr bin)运行过程

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

图 6.1.2.14 SPL 运行过程

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

图 6.1.2.15 Trust 运行过程

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

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

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

图 6.1.2.16 U-Boot 运行过程

6.1.3 defconfig 配置文件

U-Boot 中，RK3568 平台使用的 defconfig 配置文件为：**<U-Boot>/configs/rk3568_defconfig**，开发者基本不用去改动它，其实对于整个 U-Boot 来说，如果你的产品没什么特殊的需求，基本不用去动 U-Boot 源码。

如果不使用 make.sh 脚本，直接通过 make 命令配置 U-Boot、然后编译 U-Boot，可以这样做：

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

<SDK>表示 SDK 根目录，根据你自己的实际路径修改；编译内核、编译 U-Boot 用的都是这个交叉编译器 **<SDK>/prebuilts/gcc/linux-x86/aarch64/gcc-linaro-6.3.1-2017.05-x86_64_aarch64-linux-gnu/bin/aarch64-linux-gnu-**。

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/u-boot$ make rk3568_defconfig -j24
  HOSTCC  scripts/basic/fixdep
  HOSTCC  scripts/kconfig/conf.o
  SHIPPED scripts/kconfig/zconf.tab.c
  SHIPPED scripts/kconfig/zconf.lex.c
  SHIPPED scripts/kconfig/zconf.hash.c
  HOSTCC  scripts/kconfig/zconf.tab.o
In file included from scripts/kconfig/zconf.tab.c:2468:
scripts/kconfig/confdata.c: In function 'conf_write':
scripts/kconfig/confdata.c:771:19: warning: '%s' directive writing likely 7 or more bytes into a region of size between 1 a
  771 |  sprintf(newname, "%s%s", dirname, basename);
      |                   ^~~~~~
scripts/kconfig/confdata.c:771:19: note: assuming directive output of 7 bytes
In file included from /usr/include/stdio.h:867,
                 from scripts/kconfig/zconf.tab.c:82:
/usr/include/x86_64-linux-gnu/bits/stdio2.h:36:10: note: '__builtin___sprintf_chk' output 1 or more bytes (assuming 4104) i
   36 |   return __builtin___sprintf_chk (__s, __USE_FORTIFY_LEVEL - 1,
      |          ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   37 |       __bos (__s), __fmt, __va_arg_pack ());
      |       ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
In file included from scripts/kconfig/zconf.tab.c:2468:
scripts/kconfig/confdata.c:774:20: warning: '.tmpconfig.' directive writing 11 bytes into a region of size between 1 and 40
  774 |  sprintf(tmpname, "%s.tmpconfig.%d", dirname, (int)getpid());
      |                   ^~~~~~~~~~~~~~~~
In file included from /usr/include/stdio.h:867,
                 from scripts/kconfig/zconf.tab.c:82:
/usr/include/x86_64-linux-gnu/bits/stdio2.h:36:10: note: '__builtin___sprintf_chk' output between 13 and 4119 bytes into a
   36 |   return __builtin___sprintf_chk (__s, __USE_FORTIFY_LEVEL - 1,
      |          ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   37 |       __bos (__s), __fmt, __va_arg_pack ());
      |       ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  HOSTLD  scripts/kconfig/conf
#
# configuration written to .config
#
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/u-boot$
```

图 6.1.3.1 配置 U-Boot

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/u-boot$ make PYTHON=python2 CROSS_COMPILE=/home/tgg/rk3568_linux_sdk/prebuilts/gcc/linux-x86/aarch64/gcc-linaro-6.3.1-2017.05-x86_64_aarch64-linux-gnu/bin/aarch64-linux-gnu- all --jobs=24
scripts/kconfig/conf  --silentoldconfig Kconfig
  CHK     include/config.h
  UPD     include/config.h
  CFG     u-boot.cfg
  GEN     include/autoconf.mk.dep
  CFG     spl/u-boot.cfg
  CFG     tpl/u-boot.cfg
  GEN     include/autoconf.mk
  GEN     spl/include/autoconf.mk
  GEN     tpl/include/autoconf.mk
  CHK     include/config/uboot.release
  CHK     include/generated/timestamp_autogenerated.h
  UPD     include/generated/timestamp_autogenerated.h
  HOSTCC  scripts/dtc/dtc.o
  HOSTCC  scripts/dtc/flattree.o
  HOSTCC  scripts/dtc/fstree.o
  HOSTCC  scripts/dtc/data.o
  HOSTCC  scripts/dtc/livetree.o
  HOSTCC  scripts/dtc/treesource.o
  HOSTCC  scripts/dtc/srcpos.o
  HOSTCC  scripts/dtc/checks.o
  HOSTCC  scripts/dtc/util.o
  SHIPPED scripts/dtc/dtc-lexer.lex.c
  SHIPPED scripts/dtc/dtc-parser.tab.h
  SHIPPED scripts/dtc/dtc-parser.tab.c
  HOSTCC  scripts/dtc/dtc-parser.tab.o
  HOSTCC  scripts/dtc/dtc-lexer.lex.o
  UPD     include/config/uboot.release
  CHK     include/generated/version_autogenerated.h
  UPD     include/generated/version_autogenerated.h
  CC      lib/asm-offsets.s
  CC      arch/arm/lib/asm-offsets.s
```

图 6.1.3.2 编译 U-Boot

编译完成之后，并没有生成 uboot.img 和 rk356x_spl_loader_v1.13.112.bin，因为这两个镜像属于 pack 打包镜像，需要通过专门的工具或脚本才能生成，make.sh 脚本中已经添加了打包操作，所以使用 make.sh 脚本编译后会生成 uboot.img 和 rk356x_spl_loader_v1.13.112.bin。

当然我们可以执行如下命令去打包生成这两个镜像：

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

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/u-boot$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/u-boot$ ./scripts/fit.sh --ini-loader ../rkbin/RKBOOT/RK3568MINIALL.ini --chip RK3568
SEC=1
pack u-boot.itb okay! Input: /home/tgg/rk3568_linux_sdk/rkbin/RKTRUST/RK3568TRUST.ini

FIT description: FIT Image with ATF/OP-TEE/U-Boot/MCU
Created:         Tue Apr 11 17:50:31 2023
 Image 0 (uboot)
  Description:  U-Boot
  Created:      Tue Apr 11 17:50:31 2023
  Type:         Standalone Program
  Compression:  uncompressed
  Data Size:    1267712 Bytes = 1238.00 KiB = 1.21 MiB
  Architecture: AArch64
  Load Address: 0x00a00000
  Entry Point:  unavailable
  Hash algo:    sha256
  Hash value:   b2a3d3cc74e7a4b9119e04e4f398b5ac62d8691de72ed9d211cb43c3144e454f
 Image 1 (atf-1)
  Description:  ARM Trusted Firmware
  Created:      Tue Apr 11 17:50:31 2023
  Type:         Firmware
  Compression:  uncompressed
  Data Size:    163840 Bytes = 160.00 KiB = 0.16 MiB
  Architecture: AArch64
  Load Address: 0x00040000
  Hash algo:    sha256
  Hash value:   6204b6f381d2ad6175ffa52cc82abe8c934eb23aaf8a00c7db2dda0367449fdc
 Image 2 (atf-2)
  Description:  ARM Trusted Firmware
  Created:      Tue Apr 11 17:50:31 2023
  Type:         Firmware
  Compression:  uncompressed
  Data Size:    40960 Bytes = 40.00 KiB = 0.04 MiB
  Architecture: AArch64
  Load Address: 0xfdcc1000
  Hash algo:    sha256
  Hash value:   5563d929dab73f22d2229dd7933b58de3dd6553334fc653502e97b1bd561365f
 Image 3 (atf-3)
  Description:  ARM Trusted Firmware
  Created:      Tue Apr 11 17:50:31 2023
  Type:         Firmware
  Compression:  uncompressed
  Data Size:    20267 Bytes = 19.79 KiB = 0.02 MiB
  Architecture: AArch64
```

图 6.1.3.3 执行打包操作(1)

```
  Hash value:   66bbd173528d12e9739c336926e33ee1ac1f4c7078fcac5712eeb8747d02163e
 Image 8 (fdt)
  Description:  U-Boot dtb
  Created:      Tue Apr 11 17:50:31 2023
  Type:         Flat Device Tree
  Compression:  uncompressed
  Data Size:    14377 Bytes = 14.04 KiB = 0.01 MiB
  Architecture: AArch64
  Hash algo:    sha256
  Hash value:   56cff76c011fcefff4f52549b1ed4a1ba3f5cccd48430161b0d24da4ae89918a
 Default Configuration: 'conf'
 Configuration 0 (conf)
  Description:  rk3568-evb
  Kernel:       unavailable
  Firmware:     atf-1
  FDT:          fdt
  Loadables:    uboot
                atf-2
                atf-3
                atf-4
                atf-5
                atf-6
                optee
********boot_merger ver 1.2********
Info:Pack loader ok.
pack loader okay! Input: ../rkbin/RKBOOT/RK3568MINIALL.ini
/home/tgg/rk3568_linux_sdk/u-boot

Image(no-signed, version=0): uboot.img (FIT with uboot, trust...) is ready
Image(no-signed): rk356x_spl_loader_v1.13.112.bin (with spl, ddr...) is ready
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/u-boot$
```

图 6.1.3.4 执行打包操作(2)

执行成功便会生成这两个镜像文件：

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

图 6.1.3.5 uboot.img 和 MiniLoaderAll.bin

scripts/fit.sh 脚本会调用 **tools/mkimage** 工具生成 uboot.img、调用 **tools/boot_merger** 工具生成 rk356x_spl_loader_v1.13.112.bin。还支持多种其它的用法，大家可以自己去摸索。

如果需要清理 U-Boot 源码工程，执行“**make distclean**”命令进行清理：

```
make distclean
```

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/u-boot$ make distclean
  CLEAN   dts/../arch/arm/dts
  CLEAN   dts
  CLEAN   examples/standalone
  CLEAN   tools
  CLEAN   tools/lib tools/common
  CLEAN   spl/arch spl/board spl/cmd spl/common spl/disk spl/drivers spl/dts spl/env spl/fs spl/lib spl/
pl.lds spl/u-boot-spl.map spl/u-boot-spl-nodtb.bin spl/u-boot-spl.sym tpl/arch tpl/board tpl/common tpl/
 tpl/u-boot-tpl.map tpl/u-boot-tpl-nodtb.bin tpl/u-boot-tpl.sym
  CLEAN   u-boot-dtb.bin u-boot.lds u-boot.dtb u-boot.map u-boot-nodtb.bin u-boot.srec u-boot.bin u-boot
 bl31_0xfdcd0000.bin bl31_0x00040000.bin bl31_0xfdcce000.bin bl31_0x0006a000.bin u-boot-nodtb.bin bl31_0
  CLEAN   scripts/basic
  CLEAN   scripts/dtc
  CLEAN   scripts/kconfig
  CLEAN   include/config include/generated spl tpl
  CLEAN   .config include/autoconf.mk include/autoconf.mk.dep include/config.h
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/u-boot$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/u-boot$
```

图 6.1.3.6 清理 U-Boot

6.1.4 快捷键

RK 平台 U-Boot 支持通过串口组合键触发一些事件用于烧写、调试等操作。RK U-Boot 运行后不会出现启动倒计时（启动内核的倒计时），它会直接启动内核，跳过倒计时。

当然，可以通过串口组合键“Ctrl + C”打断它启动内核、从而进入 U-Boot 命令行模式。RK U-Boot 支持如下串口组合键（**开机时长按**）：

- Ctrl + C：进入 U-Boot 命令行模式。
- Ctrl + D：进入 Loader 烧写模式；除了 2.9.1 小节中介绍的进入 Loader 模式的方法之外，这也是一种进入 Loader 烧写模式的方法。
- Ctrl + B：进入 Maskrom 烧写模式；除了 2.9.1 小节中介绍的进入 Maskrom 模式的方法之外，这也是一种进入 Maskrom 烧写模式的方法。
- Ctrl + F：进入 fastboot 模式。
- Ctrl + M：打印 bidram/system 的信息。
- Ctrl + I：使能内核 initcall_debug。
- Ctrl + P：打印 cmdline 信息。
- Ctrl + S："Starting kernel..."之后进入 U-Boot 命令行。

大家可以自己测试下，这里不多说。

6.1.5 HW-ID DTB

RK 平台 U-Boot 支持检测 GPIO 或者 ADC 的硬件状态实现动态加载不同的 kernel DTB，也就是说它可以根据 GPIO 或者 ADC 的硬件状态，从多份 Kernel DTB 文件中找到与当前硬件状态匹配的 DTB 进行加载；譬如说某个固定的 ADC 值、固定的某 GPIO 电平，RK 将这种功能称为 **HW-ID DTB**。

譬如正点原子 ATK-DLRK3568 开发板的 MIPI 屏接口支持 5.5 寸 720p MIPI 屏（正点原子 5.5 寸 MIPI 屏）、同样也支持 5.5 寸 1080p MIPI 屏（正点原子 5.5 寸 MIPI 屏），但是这两种屏的时序参数以及初始化参数都是不同的，两种 MIPI 屏就需要两个 DTS 文件分别进行配置，所以会导致出现这样一个情况：使用 720p MIPI 屏时，需要烧录 720p MIPI 屏对应的 DTB，使用 1080p MIPI 屏时，需要烧录 1080p MIPI 屏对应的 DTB。显得非常麻烦！

此时可以通过 HW-ID DTB 来解决这种麻烦，首先我们需要将 720p MIPI 屏的 DTB 文件和 1080p MIPI 屏的 DTB 文件全打包进同一个 resource.img 镜像中，U-Boot 引导内核时会检测硬件状态（ADC 或 GPIO），从 resource.img 镜像中找出与当期硬件状态（某个固定的 ADC 值或某个 GPIO 的电平）相匹配的 DTB、并加载。

1. 硬件设计参考

目前支持检测 ADC 和 GPIO 两种硬件状态。

正点原子 ATK-DLRK3568 开发板 MIPI 屏接口处有设计一个 SARADC_VIN2_LCD_ID 引脚：

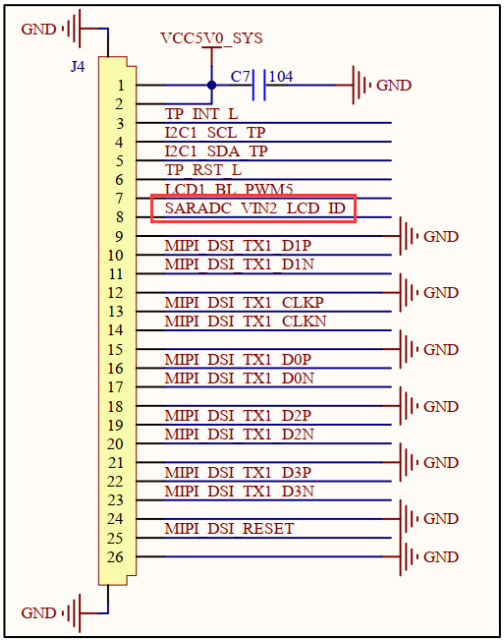
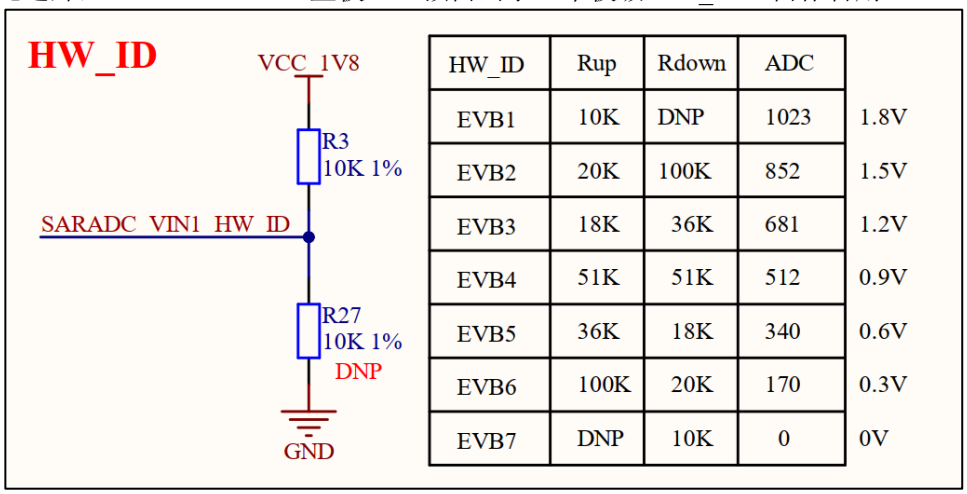


图 6.1.5.1 MIPI 屏接口

SARADC_VIN2_LCD_ID 是一个 ADC 引脚（使用 ADC 2 通道），用户接入 720p MIPI 屏和接入 1080p MIPI 屏，其对应的 ADC 值是不一样的（两种屏在硬件设计上使用了不同的分压电阻）。U-Boot 加载内核 DTB 之前，会读取这个 ADC 值，根据读取到的 ADC 值找到与之匹配的 DTB 进行加载。从而可以根据用户接入的 MIPI 屏实现动态加载 DTB（前提是需要将这两种屏的 DTB 文件全打包进 resource.img）。

除此之外，ATK-DLRK3568 主板上还预留出了一个板级 HW_ID（留作备用）：



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

图 6.1.5.2 主板上的 HW_ID

在底板上通过焊接不同的分压电阻（使用 ADC 1 通道），可以用于确定不同的硬件版本。用户可以针对不同的硬件版本，软件上提供对应的 DTB 文件（一个版本对应一个 DTB），当用户使用不同版本的硬件时、能够自动加载与当前版本相对应的 DTB。

目前没有 GPIO 的硬件参考设计，用户可自己定制。

2. DTB 命令规则

用户需要将 ADC/GPIO 的硬件唯一值信息体现在内核 DTB 文件名里。内核 DTB 命名规则如下：

ADC 作为 HW_ID DTB

- 文件名以.dtb 结尾；
- 文件名等于“自定义部分 + #_[controller]_ch[channel]=[adcval]”，“#_[controller]_ch[channel]=[adcval]”称为一个完整单元，描述 ADC 硬件唯一值信息：
  [controller]：内核设备树中 ADC 控制器的名字，通常是 saradc；
  [channel]：ADC 通道，譬如 0、1、2、3 等；
  [adcval]：ADC 的理论计算值，实际有效的范围是 adcval+-30。
- 每个完整单元（#_[controller]_ch[channel]=[adcval]）必须使用小写字母，内部不能有空格；
- 多个完整单元之间通过#号分隔，最多支持 10 个单元。如果某个 DTB 需要同时满足多个 ADC 硬件唯一值才可以匹配到它，那么这个 DTB 文件命名就会存在多个单元。

示例：

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

第一个 DTB 文件名表示，当从 ADC 2 通道读取到的 ADC 值为 111+-30 时，会匹配到该 DTB；第二个则表示，当从 ADC 2 通道读取到的 ADC 值为 111+-30、并且从 ADC 1 通道读取到的 ADC 值为 810+-30 时，才会匹配到该 DTB（条件 1 && 条件 2）。

GPIO 作为 HW_ID DTB

- 文件名以.dtb 结尾；
- 文件名等于“自定义部分 +#_gpio[pin]=[level]”，“#_gpio[pin]=[level]”称为一个完整单元，用于描述 GPIO 硬件唯一值信息：
  [pin]：GPIO 引脚，譬如 0a2 就表示 GPIO0_A2；
  [level]：GPIO 引脚的电平，0 或者 1；
- 每个完整单元（#_gpio[pin]=[level]）必须使用小写字母，内部不能有空格；
- 多个完整单元之间通过#号分隔，最多支持 10 个单元。如果某个 DTB 需要同时满足多个 GPIO 硬件唯一值才可以匹配到它，那么这个 DTB 文件命名就会存在多个单元。

示例：

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

第一个 DTB 文件名表示，当读取 GPIO0_C1 为高电平时，会匹配到该 DTB；第二个 DTB 文件名表示，当读取 GPIO0_C1 为高电平、并且读取 GPIO2_B3 为低电平时才会匹配到该 DTB（条件 1 && 条件 2）。

3. DTB 打包

通过<SDK>/kernel/scripts/mkmultidtb.py（内核源码目录下 scripts/mkmultidtb.py）脚本可将多个内核 DTB 打包进同一个 resource.img。具体的情况请看 6.2.1 小节。

4. 功能启动

RK U-Boot 默认没有使能 HW-ID DTB 功能，不过正点原子已经将其使能了，因为 ATK-DLRK3568 开发板硬件设计需要使用到这个功能，去适配两款 MIPI 屏。

通过将宏 CONFIG_ROCKCHIP_HWID_DTB=y 可以使能 HW-ID DTB。如果用户在自己的产品设计中不需要使用到该功能，可以将其禁用，打开<U-Boot>/configs/rk3568_defconfig 文件，找到下面这行，然后将其注释掉即可！

CONFIG_ROCKCHIP_HWID_DTB=y

如下所示：

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

图 6.1.5.3 禁用 HW-ID DTB

6.2 kernel 开发

Linux 内核源码在**<SDK>/kernel** 目录下：

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

图 6.2.1 内核源码目录

本节介绍内核开发相关的内容。

6.2.1 内核编译

编译之前，我们先执行“**make distclean**”清理一下。在内核源码目录下有一个 make.sh 脚本文件，可以直接使用这个脚本文件来编译内核源码，直接运行即可，如下所示：

```
./make.sh
```

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel$ ./make.sh
  HOSTCC  scripts/basic/fixdep
  HOSTCC  scripts/kconfig/conf.o
  YACC    scripts/kconfig/zconf.tab.c
  LEX     scripts/kconfig/zconf.lex.c
  HOSTCC  scripts/kconfig/zconf.tab.o
  HOSTLD  scripts/kconfig/conf
#
# configuration written to .config
#
  WRAP    arch/arm64/include/generated/uapi/asm/errno.h
  WRAP    arch/arm64/include/generated/uapi/asm/ioctl.h
  WRAP    arch/arm64/include/generated/uapi/asm/ioctls.h
  WRAP    arch/arm64/include/generated/uapi/asm/ipcbuf.h
  WRAP    arch/arm64/include/generated/uapi/asm/kvm_para.h
  WRAP    arch/arm64/include/generated/uapi/asm/mman.h
  WRAP    arch/arm64/include/generated/uapi/asm/msgbuf.h
  WRAP    arch/arm64/include/generated/uapi/asm/poll.h
  WRAP    arch/arm64/include/generated/uapi/asm/resource.h
  WRAP    arch/arm64/include/generated/uapi/asm/sembuf.h
  WRAP    arch/arm64/include/generated/uapi/asm/shmbuf.h
  WRAP    arch/arm64/include/generated/uapi/asm/socket.h
  WRAP    arch/arm64/include/generated/uapi/asm/sockios.h
  WRAP    arch/arm64/include/generated/uapi/asm/swab.h
  WRAP    arch/arm64/include/generated/uapi/asm/termbits.h
  WRAP    arch/arm64/include/generated/uapi/asm/termios.h
  WRAP    arch/arm64/include/generated/uapi/asm/types.h
  UPD     include/generated/uapi/linux/version.h
  UPD     include/config/kernel.release
  WRAP    arch/arm64/include/generated/asm/bugs.h
  WRAP    arch/arm64/include/generated/asm/delay.h
  WRAP    arch/arm64/include/generated/asm/div64.h
  WRAP    arch/arm64/include/generated/asm/dma.h
  WRAP    arch/arm64/include/generated/asm/dma-contiguous.h
```

图 6.2.1.1 编译内核(1)

```
  CC [M]  drivers/net/wireless/rockchip_wlan/rtl8852bs/phl/hal_g6/phy/rf/halrf_8852b/halrf_kfree_8852b.o
  CC [M]  drivers/net/wireless/rockchip_wlan/rtl8852bs/phl/hal_g6/phy/rf/halrf_8852b/halrf_psd_8852b.o
  LD [M]  drivers/net/wireless/rockchip_wlan/rtl8852bs/8852bs.o
  Building modules, stage 2.
  MODPOST 4 modules
  CC      drivers/net/wireless/marvell/mwifiex/mwifiex.mod.o
  CC      drivers/net/wireless/marvell/mwifiex/mwifiex_sdio.mod.o
  CC      drivers/net/wireless/rockchip_wlan/rkwifi/bcmdhd/bcmdhd.mod.o
  CC      drivers/net/wireless/rockchip_wlan/rtl8852bs/8852bs.mod.o
  LD [M]  drivers/net/wireless/rockchip_wlan/rkwifi/bcmdhd/bcmdhd.ko
  LD [M]  drivers/net/wireless/rockchip_wlan/rtl8852bs/8852bs.ko
  LD [M]  drivers/net/wireless/marvell/mwifiex/mwifiex_sdio.ko
  LD [M]  drivers/net/wireless/marvell/mwifiex/mwifiex.ko
found ./arch/arm64/boot/dts/rockchip/.rk3568-atk-evb1-mipi-dsi-720p.dtb.dts.tmp
found ./arch/arm64/boot/dts/rockchip/.rk3568-atk-evb1-mipi-dsi-720p.dtb.dts.tmp
found ./arch/arm64/boot/dts/rockchip/.rk3568-atk-evb1-mipi-dsi-720p.dtb.dts.tmp
found ./arch/arm64/boot/dts/rockchip/.rk3568-atk-evb1-mipi-dsi-720p.dtb.dts.tmp
found ./arch/arm64/boot/dts/rockchip/.rk3568-atk-evb1-mipi-dsi-720p.dtb.dts.tmp
found ./arch/arm64/boot/dts/rockchip/.rk3568-atk-evb1-mipi-dsi-720p.dtb.dts.tmp
found ./arch/arm64/boot/dts/rockchip/.rk3568-atk-evb1-mipi-dsi-720p.dtb.dts.tmp
0
  Image:  resource.img (with rk3568-atk-evb1-mipi-dsi-720p.dtb logo.bmp logo_kernel.bmp) is ready
  Image:  boot.img (with Image  resource.img) is ready
  Image:  zboot.img (with Image.lz4  resource.img) is ready
#################################################
  DTC     arch/arm64/boot/dts/rockchip/rk3568-atk-evb1-ddr4-v10-linux.dtb
  DTC     arch/arm64/boot/dts/rockchip/rk3568-atk-evb1-mipi-dsi-1080p.dtb
rk-kernel.dtb  rk3568-atk-evb1-mipi-dsi-720p#_saradc_ch2=0.dtb rk3568-atk-evb1-mipi-dsi-1080p#_saradc_ch2=341.dtb
Pack to resource.img successed!
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel$
```

图 6.2.1.2 编译内核(2)

编译完成后，会生成 boot.img 以及 resource.img，如下所示：

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

图 6.2.1.3 boot.img 镜像和 resource.img 镜像

关于这两个镜像，也要给大家讲一讲。

1. boot.img 镜像

RK3568 平台 Linux 系统使用的 boot.img 是一种 FIT 格式镜像，它由多个镜像合并而成，对于 RK3568 平台来说，烧录到开发板 boot 分区的 boot.img 包含了内核镜像 Image、内核 DTB、resource.img 这三部分。可以使用 file 命令查看 boot.img 文件，如下所示：

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

图 6.2.1.5 file 查看 boot.img

可以发现该文件是一个 Android 格式启动镜像（Android bootimg），并非 FIT 格式；这是怎么回事？**事实上，通过 make.sh 脚本编译后生成的 boot.img 确实是 Android 格式镜像；Android 格式 boot.img 用于 RK3568 平台 Android 系统，而 FIT 格式 boot.img 才用于 RK3568 平台 Linux 系统**。说明这个 boot.img 并不是最终烧录到开发板 boot 分区（Linux 系统）的 boot.img。

所以直接通过 make.sh 脚本编译生成的 boot.img 并不是最终使用的 boot.img，可通过如下命令生成 FIT 格式的 boot.img，首先得进入到<SDK>顶层目录下，执行命令：

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

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

图 6.2.1.6 生成 FIT 格式 boot.img

使用<SDK>/device/rockchip/common/mk-fitimage.sh 脚本生成 FIT 格式 boot.img 镜像。

第一个参数：表示输出文件 boot.img 的输出路径，如果使用相对地址、则必须相对于 SDK 源码根目录而言；

第二个参数：指定 its 源文件的路径，如果使用相对地址、则必须相对于 SDK 源码根目录而言；前面讲过，FIT 格式需要使用 its 文件来描述 image 的信息，所以这里需要指定一个 its 源文件。

执行上述命令，会将内核镜像、内核 DTB 以及资源镜像 resource.img 打包成一个 FIT 格式 boot.img；命令中，并未指定内核镜像、内核 DTB 以及 resource.img 的路径，因为它们都有默认值（阅读 mk-fitimage.sh 脚本可知），如下：

内核镜像： 板级配置文件中 RK_KERNEL_IMG 变量所指定的镜像

内核 DTB： kernel/arch/arm64/boot/dts/rockchip/$RK_KERNEL_DTS.dtb
资源镜像 resource.img： kernel/resource.img

RK_KERNEL_IMG 和 RK_KERNEL_DTS 都是板级配置文件中定义的变量：

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

此时再使用 file 命令查看**<SDK>/kernel/boot.img**：

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

图 6.2.1.7 file 查看 boot.img

“**Device Tree Blob**”表示该文件是一个 FIT 格式镜像。通过 mk-fitimage.sh 脚本打包生成的 boot.img 才是最终烧录到开发板 boot 分区（Linux 系统）的 boot.img。

使用 mk-fitimage.sh 脚本生成 boot.img 镜像时，它会去<Kernel>/arch/arm64/boot/目录下寻找内核镜像 Image、<Kernel>/arch/arm64/boot/dts/rockchip/目录下寻找内核 DTB 以及<Kernel>/目录下寻找 resource.img：

<Kernel>/arch/arm64/boot/Image：内核镜像（由板级配置文件中的 RK_KERNEL_IMG 变量指定）。

<Kernel>/arch/arm64/boot/dts/rk3568-atk-evb1-ddr4-v10-linux.dtb：内核 DTB（由板级配置文件中的 RK_KERNEL_DTS 变量所决定！）。

<Kernel>/resource.img：resource 资源镜像（RK 自己定义的镜像类型，将 logo、dtb 等资源打包进 resource.img）。

所以运行脚本时，需确保这些镜像文件已经存在，否则会执行失败！

综上可知，boot.img 由内核 DTB、内核镜像以及 resource.img 组成：

文件名	说明
rk3568-atk-evb1-ddr4-v10-linux.dtb	内核 DTB。对于 RK 平台来说，这份 DTB 是无效的，它只是一个摆设，RK 平台并未使用这份 DTB，而实际生效的是 resource.img 中的 DTB，因为 RK U-Boot 是直接从 resource.img 中去读取内核 DTB、并将其加载到内存
Image	内核镜像
resource.img	RK 平台资源镜像，resource.img 中包含了内核 DTB、logo 图片。RK U-Boot 会从 resource.img 中读取内核 DTB、而不是直接打包进 boot.img 的这份 DTB，这个一定要搞清楚！

如果改动了设备树文件，需要重新编译设备树得到新的 Kernel DTB，然后将它打包进 resource.img、之后再把 resource.img 打包进 boot.img，最后将新的 boot.img 烧录到开发板 boot 分区完成替换；虽然很麻烦，但必须得这么做。

2. resource.img 镜像

resource.img 是 RK 自己设计的一种镜像，用于存放一些资源，譬如 u-boot logo 图片、内核 logo 图片以及设备树镜像 DTB。resource.img 并不是单独烧录到开发板，而是将其打包进 boot.img 中，最终烧录 boot.img。

通过<Kernel>/scripts/resource_tool 工具可以生成 resource.img 镜像，可以将一个或多个资源（DTB、图片资源等）打包进 resource.img 镜像中，如下所示：

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

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel$ mkdir temp_dir
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel$ cd temp_dir/
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel/temp_dir$ cp ../arch/arm64/boot/dts/rockchip/rk3568-atk-evb1-ddr4-v10-linux.dtb ./
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel/temp_dir$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel/temp_dir$ cp ../logo.bmp ../logo_kernel.bmp ./
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel/temp_dir$ ls
logo.bmp  logo_kernel.bmp  rk3568-atk-evb1-ddr4-v10-linux.dtb
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel/temp_dir$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel/temp_dir$ ../scripts/resource_tool logo.bmp logo_kernel.bmp rk3568-atk-evb1-ddr4-v10-linux.dtb
Pack to resource.img successed!
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel/temp_dir$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel/temp_dir$ ls
logo.bmp  logo_kernel.bmp  resource.img  rk3568-atk-evb1-ddr4-v10-linux.dtb
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel/temp_dir$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel/temp_dir$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel/temp_dir$
```

图 6.2.1.8 制作 resource.img

示例中，我们将 u-boot logo 图片 logo.bmp、内核 logo 图片 logo_kernel.bmp 以及内核 DTB 打包成一个 resource.img 镜像。resource_tool 命令后面携带的资源文件不能有路径，所以需要将内核 DTB 文件、以及 logo 图片拷贝到当前目录。

使用 scripts/resource_tool 工具也可以将 resource.img 进行解包操作，执行如下命令进行解包：

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

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel/temp_dir$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel/temp_dir$ ls
logo.bmp  logo_kernel.bmp  resource.img  rk3568-atk-evb1-ddr4-v10-linux.dtb
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel/temp_dir$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel/temp_dir$ rm -rf logo.bmp logo_kernel.bmp rk3568-atk-evb1-ddr4-v10-linux.dtb
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel/temp_dir$ ls
resource.img
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel/temp_dir$ ../scripts/resource_tool --unpack
Dump header:
partition version:0.0
header size:1
index tbl:
        offset:1        entry size:1    entry num:3
Dump Index table:
entry(0):
        path:rk-kernel.dtb
        offset:4        size:141844
entry(1):
        path:logo_kernel.bmp
        offset:282      size:177764
entry(2):
        path:logo.bmp
        offset:630      size:177700
Unack resource.img to out successed!
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel/temp_dir$ ls
out  resource.img
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel/temp_dir$ ls out/
logo.bmp  logo_kernel.bmp  rk-kernel.dtb
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel/temp_dir$
```

图 6.2.1.9 resource.img 解包

解包后得到 u-boot logo 图片 logo.bmp、内核 logo 图片 logo_kernel.bmp 以及一个 DTB 文件；因为制作 resource.img 时，会将 DTB 文件重命名为 rk-kernel.dtb。

用户可以将多个内核 DTB 打包进同一个 resource.img 镜像中，譬如 ATK-DLRK3568 开发板出厂系统所使用的 resource.img 中就包含了 2 个内核 DTB，为了适配正点原子的两款 MIPI 屏

（5.5 寸 720p 和 5.5 寸 1080p）。这两个 DTB 分别是：arch/arm64/boot/dts/rockchip/rk3568-atk-evb1-mipi-dsi-720p.dtb 和 arch/arm64/boot/dts/rockchip/rk3568-atk-evb1-mipi-dsi-1080p.dtb。

因为 RK 平台 U-Boot 支持 HW-ID DTB 功能，所以在 U-Boot 中可以根据用户接入的 MIPI 屏动态加载对应的内核 DTB；用户接入的是 720p MIPI 屏，则加载 rk3568-atk-evb1-mipi-dsi-720p.dtb，接入的是 1080p MIPI 屏，则加载 rk3568-atk-evb1-mipi-dsi-1080p.dtb。

制作包含多个内核 DTB 的 resource.img 时，可以使用 scripts/mkmultidtb.py 脚本去做，比直接使用 scripts/resource_tool 工具更加方便；scripts/mkmultidtb.py 脚本依然是通过 scripts/resource_tool 工具去制作 resource.img，不过在此之前，它还会对 DTB 文件名进行规则化，这是 U-Boot HW-ID DTB 所要求的，前面 6.1.5 小节已经给大家讲过。

使用 scripts/mkmultidtb.py 脚本制作 resource.img 之前，需要开发者提前配置好该脚本，打开该文件：

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

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

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

图 6.2.1.10 mkmultidtb.py 脚本

该文件中已经做了 3 个配置，用户需要把打包的 DTB 文件写入 DTBS 字典里面，DTBS['RK3568-ATK-EVB1']这个是正点原子针对 ATK-DLRK3568 开发板所配置的，**如果用户需要自己定义，按照规则去配置就行了**。RK3568-ATK-EVB1 表示配置名，rk3568-atk-evb1-mipi-dsi-720p 和 rk3568-atk-evb1-mipi-dsi-1080p 表示需要打包的 DTB 文件名（**不带后缀.dtb**），它会去 arch/arm64/boot/dts/rockchip/目录下找这个 DTB 文件。

经过 mkmultidtb.py 脚本处理后：

rk3568-atk-evb1-mipi-dsi-720p.dtb ---> rk3568-atk-evb1-mipi-dsi-720p#_saradc_ch2=0.dtb
rk3568-atk-evb1-mipi-dsi-1080p.dtb ---> rk3568-atk-evb1-mipi-dsi-1080p#_saradc_ch2=341.dtb

处理后的 DTB 名字才符合 U-Boot HW-ID DTB 对 DTB 命名规则的要求。执行如下命令生成 resource.img：

scripts/mkmultidtb.py RK3568-ATK-EVB1

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel$ rm -rf resource.img
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel$ ./scripts/mkmultidtb.py RK3568-ATK-EVB1
rk-kernel.dtb  rk3568-atk-evb1-mipi-dsi-720p#_saradc_ch2=0.dtb rk3568-atk-evb1-mipi-dsi-1080p#_saradc_ch2=341.dtb
Pack to resource.img successed!
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel$ ls -l resource.img
-rw-rw-r-- 1 tgg tgg 458240 4月  12 16:27 resource.img
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel$
```

图 6.2.1.11 使用 mkmultidtb.py 制作 resource.img

我们可以使用 scripts/resource_tool 工具解包看看：

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel$ mkdir temp
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel$ cp resource.img ./temp/
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel$ cd temp/
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel/temp$ ls
resource.img
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel/temp$ ../scripts/resource_tool --unpack
Dump header:
partition version:0.0
header size:1
index tbl:
        offset:1        entry size:1    entry num:5
Dump Index table:
entry(0):
        path:rk-kernel.dtb
        offset:6        size:139535
entry(1):
        path:rk3568-atk-evb1-mipi-dsi-720p#_saradc_ch2=0.dtb
        offset:279      size:139535
entry(2):
        path:rk3568-atk-evb1-mipi-dsi-1080p#_saradc_ch2=341.dtb
        offset:552      size:139531
entry(3):
        path:logo_kernel.bmp
        offset:825      size:22364
entry(4):
        path:logo.bmp
        offset:869      size:12936
Unack resource.img to out successed!
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel/temp$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel/temp$ ls
out  resource.img
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel/temp$ ls out/*
 out/logo.bmp         'out/rk3568-atk-evb1-mipi-dsi-1080p#_saradc_ch2=341.dtb'   out/rk-kernel.dtb
 out/logo_kernel.bmp  'out/rk3568-atk-evb1-mipi-dsi-720p#_saradc_ch2=0.dtb'
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel/temp$
```

图 6.2.1.12 resource.img 解包

解包之后会发现有 3 个 DTB 文件：

- **rk-kernel.dtb**：一个默认的 DTB，所有 DTB 都没匹配成功时默认使用它，打包脚本 mkmultidtb.py 会将 DTBS 字典中的第一个 DTB 作为默认 DTB；
- **rk3568-atk-evb1-mipi-dsi-720p#_saradc_ch2=0.dtb**：包含 ADC 信息的 rk3568 DTB 文件，当从 ADC 2 通道读取到的 ADC 值为 0+-30 时才会加载该 DTB；
- **rk3568-atk-evb1-mipi-dsi-1080p#_saradc_ch2=341.dtb**：包含 ADC 信息的 rk3568 DTB 文件，当从 ADC 2 通道读取到的 ADC 值为 341+-30 时才会加载该 DTB。

U-Boot 必须要使能 HW-ID DTB 功能，否则 U-Boot 只会加载 rk-kernel.dtb。

3. 单独编译内核模块

运行 make.sh 脚本会编译整个内核源码，包括内核模块。当然我们也可以单独编译内核模块，这在调试内核模块时将很有用！单独编译内核模块之前需编译好内核源码，可执行如下命令编译内核模块：

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

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

图 6.2.1.13 编译内核模块

最后会生成.ko 文件：

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

图 6.2.1.14 生成.ko 文件

Linux 系统下，内核模块最终需要拷贝到根文件系统的相应目录，譬如/system/lib/modules、/usr/lib/modules 等；对于 8852bs.ko（ATK-DLRK3568 开发板板载 WiFi RTL8852 的驱动模块）模块，编译 buildroot 根文件系统过程中，会自动将其拷贝到根文件系统 system/lib/modules 目录下，最终打包生成根文件系统镜像（rootfs.img）。

编译 buildroot 根文件系统时，如果用户需要将自己的 ko 模块打包进 rootfs.img 镜像中，则需要在 buildroot 中添加相应的配置（Config.in、xxx.mk），（编译时）让 buildroot 将用户指定的 ko 模块拷贝到根文件系统，一起打包生成 rootfs.img。

6.2.2 内核设备树和 defconfig 配置文件

正点原子 ATK-DLRK3568 开发板、Linux Kernel 所使用的 defconfig 配置文件为**<Kernel>/arch/arm64/configs/rockchip_linux_defconfig**。用户可以根据自己的需求更改该文件、使能或禁用内核模块。

Rockchip 平台的所有设备树文件都存放在**<Kernel>arch/arm64/boot/dts/rockchip/**目录下。对于 ATK-DLRK3568 开发板来说，使用的设备树文件为：**rk3568-atk-evb1-ddr4-v10-linux.dts**。

rk3568-atk-evb1-ddr4-v10-linux.dts 包含有多个.dtsi 设备树，如下所示：

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

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

- rk3568.dtsi：该设备树文件是 RK3568 平台级设备树文件，与具体开发板硬件无关，纯 SoC 级别的设备树文件，由 RK 提供，开发者无需改动该文件！
- rk3568-evb.dtsi：RK3568 板级通用设备树文件，通常被板级设备树文件所包含；
- rk3568-linux.dtsi：该设备树包含 Linux 部分特有配置信息（与 Android 区分）；
- rk3568-atk-evb1-ddr4-v10.dtsi：ATK-DLRK3568 开发板板级设备树文件，该设备树文件包含板级通用设备树文件 rk3568-evb.dtsi 以及 RK3568 平台级设备树文件 rk3568.dtsi；
- rk3568-screen_choose.dtsi：该设备树用于选择需要使能的 LCD 屏，譬如 HDMI、MIPI、LVDS、VGA，支持单显、双显、三显。用户可在该文件中定义 ATK_LCD_TYPE_XXX 宏（详细内容请查看该文件）来使能对应的屏，譬如 ATK_LCD_TYPE_HDMI 宏表示使能 HDMI 显示，没定义该宏表示禁用；
- rk3568-lcds.dtsi：该设备树实现了 **ATK_LCD_TYPE_XXX 宏控制 LCD 屏使能**的逻辑，rk3568-lcds.dtsi 设备树会根据用户定义的 ATK_LCD_TYPE_XXX 宏，使能对应的显示接口模块、并使能与该屏相关联的外设，譬如触摸屏（只有使能 MIPI 或 LVDS 屏时才会使能触摸屏）、HDMI 音频（只有使能 HDMI 时才会使能 HDMI 音频）、背光（只有使能 MIPI 或 LVDS 时才会使能背光）。

rk3568-atk-evb1-ddr4-v10-linux.dts 设备树是提供给用户使用的，用户在编译内核源码时只需编译这个设备树即可！

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

rk3568-atk-evb1-ddr4-v10-linux.dts 是 ATK-DLRK3568 开发板对应的设备树文件。

rk3568-atk-evb1-ddr4-v10-linux.dts 设备树包含了 rk3568-screen_choose.dtsi，用户可自行修改 rk3568-screen_choose.dtsi 文件来选择需要使能的 LCD 屏，包括 HDMI、MIPI、LVDS 以及 eDP。该文件的内容如下所示：

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

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

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

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

图 6.2.2.1 rk3568-screen_choose.dtsi 设备树文件

默认情况下，只使能了 MIPI 屏和 HDMI。

如果不使用 make.sh 脚本，如何通过 make 命令编译 Linux 内核，如下所示：

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

第一步执行 defconfig 配置： make ARCH=arm64 rockchip_linux_defconfig
第二步编译内核 Image： make ARCH=arm64 Image -j24
第三步编译设备树： make ARCH=arm64 rockchip/rk3568-atk-evb1-ddr4-v10-linux.dtb -j24

执行上述命令编译成功后将会得到内核镜像 Image（arch/arm64/boot/Image）以及 Kernel DTB（arch/arm64/boot/dts/rockchip/rk3568-atk-evb1-ddr4-v10-linux.dtb）。

执行如下命令将 Kernel DTB、logo 图片打包到 resource.img 中：

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

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

图 6.2.2.2 打包 resource.img

最后一步，将 Kernel DTB、resource.img（包含了 Kernel DTB）、内核镜像 Image 打包成一个 boot.img，如下所示：

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

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

图 6.2.2.3 生成 boot.img

直接通过 mk-fitimage.sh 脚本生成<Kernel>/boot.img。

2. 修改 make.sh 脚本

用户拿到 SDK 后，需对内核源码根目录下的 make.sh 脚本进行修改；打开<kernel>/make.sh 脚本文件，然后对其进行修改，修改之后的内容如下所示：

```
#!/bin/bash

# 默认值
RK_ARCH=arm64
RK_DEFCONFIG=rockchip_linux_defconfig
RK_DEFCONFIG_FRAGMENT=
RK_DTS=rk3568-atk-evb1-ddr4-v10-linux
RK_JOBS=24

while [ $# -gt 0 ]; do
    case $1 in
        arch=*)
            arg=${1#*=}
            if [ -n "$arg" ]; then
                RK_ARCH=$arg
            fi
            shift 1
            ;;
        defconfig=*)
            arg=${1#*=}
            if [ -n "$arg" ]; then
                RK_DEFCONFIG=$arg
            fi
            shift 1
            ;;
        defconfig_fragment=*)
            RK_DEFCONFIG_FRAGMENT=${1#*=}
            shift 1
            ;;
        dts=*)
            arg=${1#*=}
            if [ -n "$arg" ]; then
                RK_DTS=$arg
            fi
            shift 1
            ;;
        jobs=*)
            arg=${1#*=}
            if [ -n "$arg" ]; then
                RK_JOBS=$arg
            fi
            shift 1
            ;;
        *)
            shift 1
            ;;
    esac
done

# 配置
make ARCH=$RK_ARCH $RK_DEFCONFIG $RK_DEFCONFIG_FRAGMENT
# 编译
make ARCH=$RK_ARCH $RK_DTS.img -j$RK_JOBS
```

默认情况下，使用 make.sh 脚本编译生成的 resource.img 镜像中并未包含 rk3568-atk-evb1-ddr4-v10-linux.dtb，导致 rk3568-atk-evb1-ddr4-v10-linux.dts 设备树配置不会生效，所以需要对其进行修改。

6.2.3 IO 电源域

本节内容参考文档：
<SDK>/docs/Common/IO-DOMAIN/Rockchip_Developer_Guide_Linux_IO_DOMAIN_CN.pdf

RK3568 一共有 10 个独立的 IO 电源域，分别为 PMUIO[0:2]和 VCCIO[1:7]，其中：

- PMUIO0、PMUIO1 为固定电平电源域，不可配置；
- PMUIO2 和 VCCIO1，VCCIO[3:7]电源域均要求硬件供电电压与软件的配置相匹配：
  ⑴、当硬件 IO 电平接 1.8V，软件电压配置也要相应配成 1.8V；
  ⑵、当硬件 IO 电平接 3.3V，软件电压配置也要相应配成 3.3V；
- 对于 VCCIO2 电源域，软件不需要配置，但是其硬件供电电压与 FLASH_VOL_SEL 状态需保持一致：
  ⑴、当 VCCIO2 供电是 1.8V，则 FLASH_VOL_SEL 管脚必须保持为高电平；
  ⑵、当 VCCIO2 供电是 3.3V，则 FLASH_VOL_SEL 管脚必须保持为低电平；

如果用户没有按照上述要求进行配置：

- 当软件配置为 1.8V，硬件供电 3.3V，会使得 IO 处于过压状态，长期工作会导致 IO 损坏；
- 当软件配置为 3.3V，硬件供电 1.8V，IO 功能会异常。

本小节讲一下如何配置 IO 电源域，总共分为 4 步：
第一步：查看硬件原理图并确认每个 IO 电源域的供电电压；
第二步：修改内核 DTS 的 IO 电源域配置节点 pmu_io_domains；
第三步：编译内核时，对 IO 电源域进行确认；
第四步：编译内核后检查 IO 电源域配置情况；

1. 查看硬件原理图并确认每个 IO 电源域的供电电压

以正点原子 ATK-DLRK3568 开发板为例，首先找到 ATK-DLRK3568 开发板硬件原理图，打开核心板原理图，核心板原理图所在路径为：**开发板光盘 A 盘-基础资料→02、开发板原理图→02、核心板原理图→ATK-CLRK3568F V1.2(核心板原理图).pdf**。

根据前面的介绍，10 个 IO 电源域中，PMUIO0 和 PMUIO1 是固定电平电源域，不用配置；对于 PMUIO2 和 VCCIO1 以及 VCCIO[3:7]这些电源域，均需要用户进行配置。

在核心板原理图上，搜索“**VCCIO1**”找到该电源域的供电电压，如下图所示：

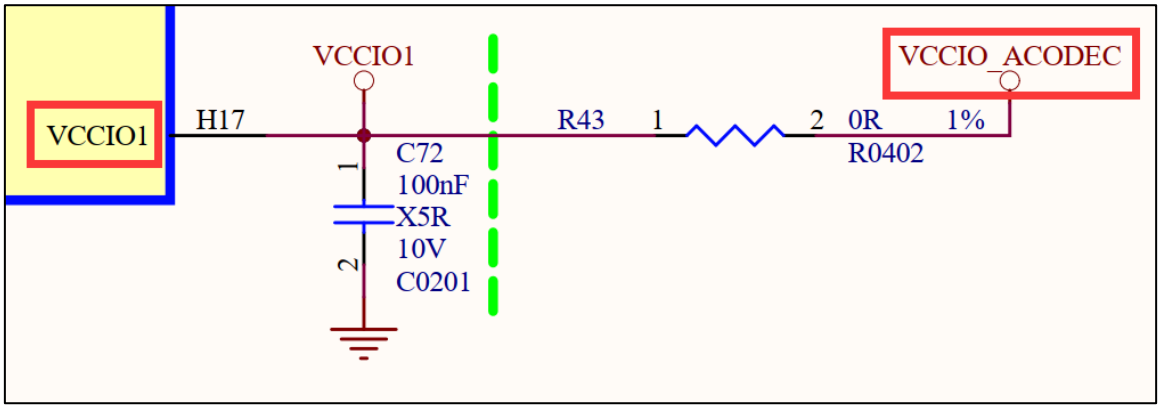


图 6.2.3.1 VCCIO1 电源域

从原理图可知，VCCIO1 的电源是 VCCIO_ACODEC，从核心板原理图上搜索“**VCCIO_ACODEC**”：

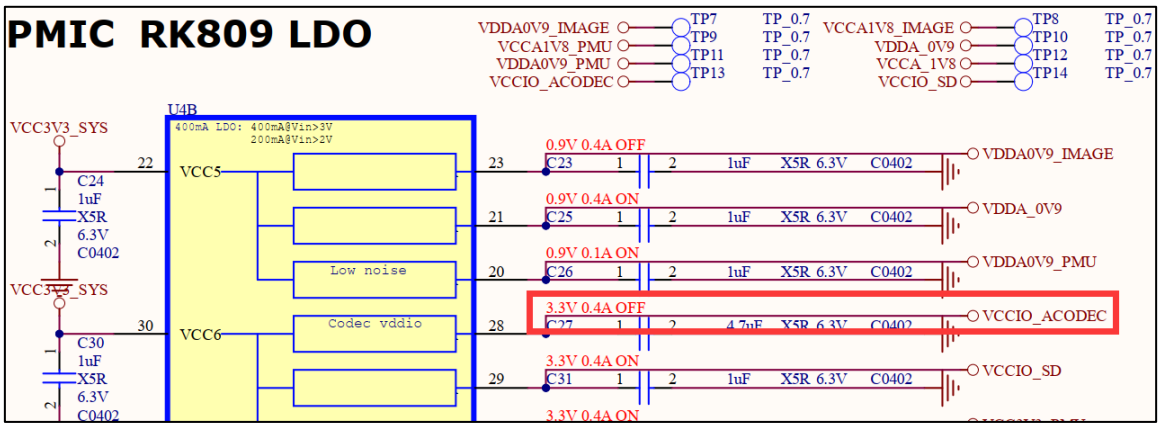


图 6.2.3.2 VCCIO_ACODEC 电源

所以由此可知，VCCIO_ACODEC 是由 RK809 的 LDO4 供电。原理图上标注该电源为 3.3V，实际电压是不是 3.3V，这个需要看设备树的配置。打开 arch/arm64/boot/dts/rockchip/rk3568-evb.dtsi 设备树文件，找到 rk809 配置信息中的 vccio_acodec 节点：

```
vccio_acodec: LDO_REG4 {
        regulator-always-on;
        regulator-boot-on;
        regulator-min-microvolt = <3300000>;
        regulator-max-microvolt = <3300000>;
        regulator-name = "vccio_acodec";
        regulator-state-mem {
                regulator-off-in-suspend;
        };
};
```

图 6.2.3.3 vccio_acodec 节点

设备树中将 VCCIO_ACODEC 电源配置为 3.3V，所以 VCCIO1 电源域的供电电压也就是 3.3V。

再来找一下 PMUIO2 电源域，在原理图上搜索“**PMUIO2**”：

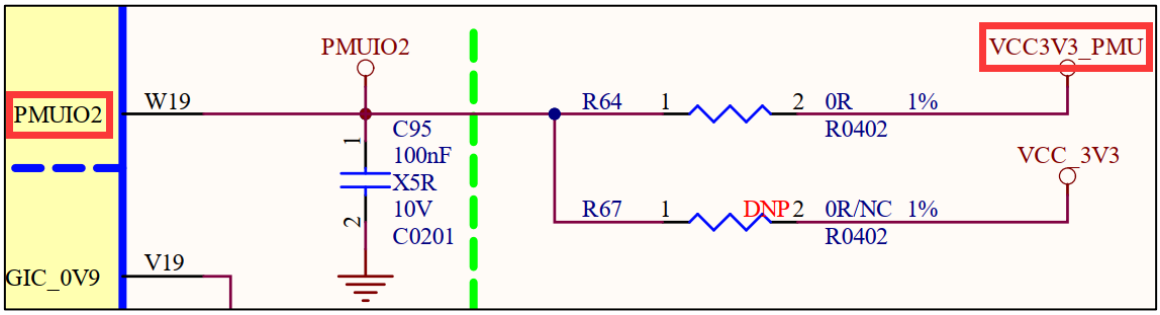


图 6.2.3.4 PMUIO2 电源域

从原理图可知，PMUIO2 的电源是 VCC3V3_PMU，从原理图上搜索“**VCC3V3_PMU**”：

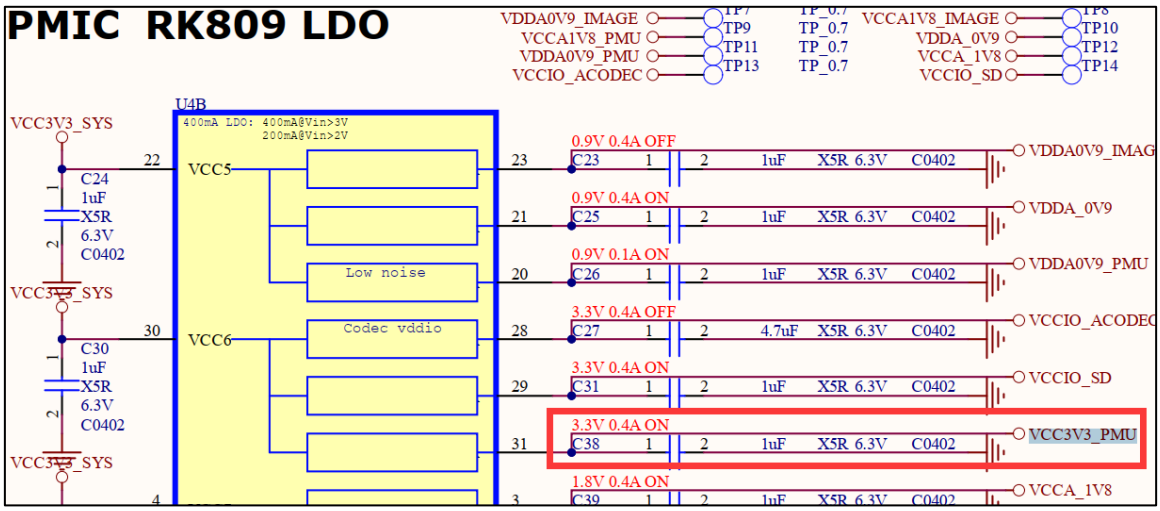


图 6.2.3.5 VCC3V3_PMU 电源

所以由此可知，VCC3V3_PMU 是由 RK809 的 LDO6 供电。打开 arch/arm64/boot/dts/rockchip/rk3568-evb.dtsi 设备树文件，找到 rk809 配置信息中的 vcc3v3_pmu 节点：

```
vcc3v3_pmu: LDO_REG6 {
        regulator-always-on;
        regulator-boot-on;
        regulator-min-microvolt = <3300000>;
        regulator-max-microvolt = <3300000>;
        regulator-name = "vcc3v3_pmu";
        regulator-state-mem {
                regulator-on-in-suspend;
                regulator-suspend-microvolt = <3300000>;
        };
};
```

图 6.2.3.6 vcc3v3_pmu 节点

设备树中将 VCC3V3_PMU 电源配置为 3.3V，所以 PMUIO2 电源域的供电电压也就是 3.3V。其它 IO 电源域的供电电压也是这样去查，这里就不再多说！

重点来讲一下 VCCIO2 电源域，VCCIO2 电源域软件上不需要配置，但是在硬件设计上有个要求：**当 VCCIO2 电源域的供电电压为 1.8V 时，FLASH_VOL_SEL 管脚必须保持为高电平、拉高，当 VCCIO2 电源域的供电电压为 3.3V 时，FLASH_VOL_SEL 管脚必须保持为低电平、拉低**。在原理图上搜索“**VCCIO2**”：

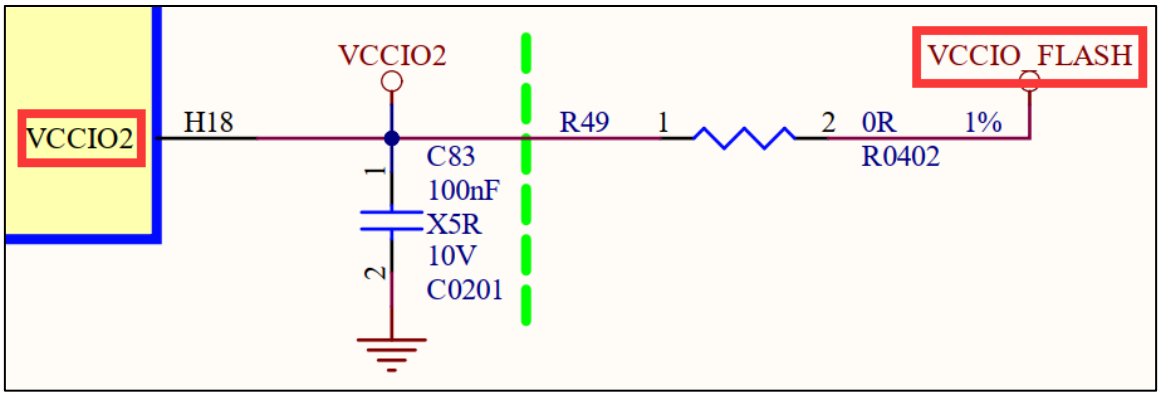


图 6.2.3.7 VCCIO2 电源域

从原理图可知，VCCIO2 的电源是 VCCIO_FLASH，在原理图上搜索“**VCCIO_FLASH**”：

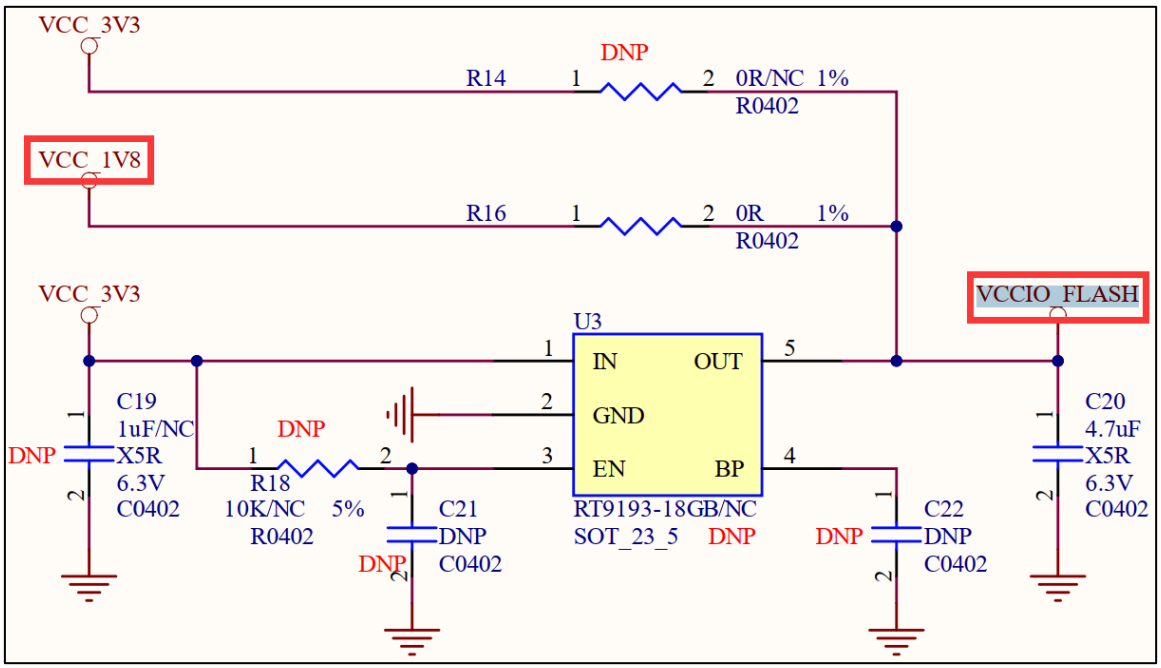


图 6.2.3.8 VCCIO_FLASH 电源

从原理图可知，VCCIO_FLASH 电源来自于 VCC_1V8，所以可知，VCCIO_FLASH 电源为 1.8V；最终可以确定 VCCIO2 电源域的供电电压为 1.8V。既然如此，按照设计要求，FLASH_VOL_SEL 管脚就必须保持为高电平、必须拉高，在原理图上找到 FLASH_VOL_SEL：

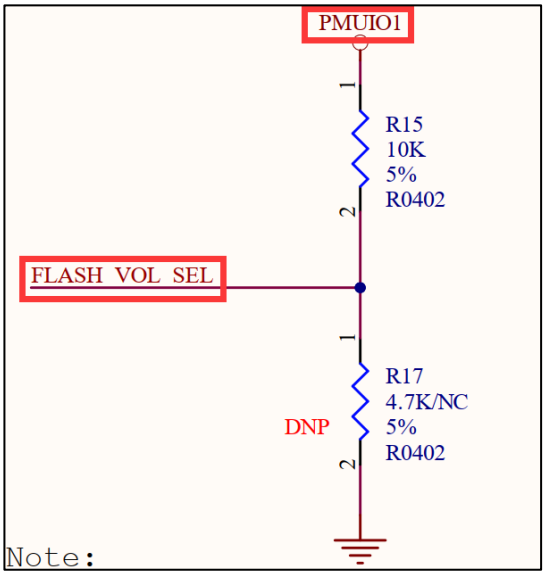


图 6.2.3.9 FLASH_VOL_SEL 管脚的电压

从原理图可知，FLASH_VOL_SEL 管脚确实被拉高了（PMUIO1 为 3.3V）、保持在一个高电平状态，满足上面提到的硬件设计要求。

2. 修改内核 DTS 的 IO 电源域配置节点 pmu_io_domains

从原理图上知道了每个 IO 电源域的供电电压后，接着就需要去配置设备树。设备树中通过 pmu_io_domains 节点对 IO 电源域进行配置，首先找到内核设备树中 IO 电源域配置节点 **pmu_io_domains**，然后对其进行修改。

pmu_io_domains 节点由 rk3568.dtsi 设备树所定义：

```
pmugrf: syscon@fdc20000 {
        compatible = "rockchip,rk3568-pmugrf", "syscon", "simple-mfd";
        reg = <0x0 0xfdc20000 0x0 0x10000>;

        pmu_io_domains: io-domains {
                compatible = "rockchip,rk3568-pmu-io-voltage-domain";
                status = "disabled";
        };

        reboot_mode: reboot-mode {
                compatible = "syscon-reboot-mode";
                offset = <0x200>;
                mode-bootloader = <BOOT_BL_DOWNLOAD>;
                mode-charge = <BOOT_CHARGING>;
                mode-fastboot = <BOOT_FASTBOOT>;
                mode-loader = <BOOT_BL_DOWNLOAD>;
                mode-normal = <BOOT_NORMAL>;
                mode-recovery = <BOOT_RECOVERY>;
                mode-ums = <BOOT_UMS>;
                mode-panic = <BOOT_PANIC>;
                mode-watchdog = <BOOT_WATCHDOG>;
        };
};
```

图 6.2.3.10 pmu_io_domains 节点的定义

我们不能直接去修改 rk3568.dtsi 文件，而是通过引用的方式进行修改（**&pmu_io_domains**），RK 默认是在 rk3568-evb.dtsi 设备树中配置，打开 rk3568-evb.dtsi 文件，找到如下位置：

```
1536 &pmu_io_domains {
1537         status = "okay";
1538         pmuio2-supply = <&vcc3v3_pmu>;
1539         vccio1-supply = <&vccio_acodec>;
1540         vccio3-supply = <&vccio_sd>;
1541         vccio4-supply = <&vcc_3v3>;
1542         vccio5-supply = <&vcc_3v3>;
1543         vccio6-supply = <&vcc_3v3>;
1544         vccio7-supply = <&vcc_3v3>;
1545 };
1546
```

图 6.2.3.11 配置 pmu_io_domains

这是 RK 默认的配置信息，它这个是根据 RK 自己的 EVB 板硬件设计情况进行配置的，与正点原子 ATK-DLRK3568 开发板的电源设计存在一些差异，主要就是 **VCCIO4** 和 **VCCIO6**：

ATK-DLRK3568 开发板 VCCIO4 电源域的供电电压是 1.8V，来自于电源 vcc_1v8；而 RK 官板的供电电压是 3.3V。ATK-DLRK3568 开发板 VCCIO6 电源域的供电电压是 1.8V，来自于电源 vcc_1v8；而 RK 官板的供电电压是 3.3V。

以上就是 ATK-DLRK3568 开发板与 RK 官板在 IO 电源域硬件设计上的一些小小区别。所以对于 ATK-DLRK3568 开发板来说，需要去修改 pmu_io_domains 节点。不建议直接在 rk3568-evb.dtsi 文件进行修改（前面给大家讲过，rk3568-evb.dtsi 属于板级通用设备树文件），我们可以在具体的板级配置文件中进行修改，譬如 rk3568-atk-evb1-ddr4-v10.dtsi（正点原子已经配置好了）：

```
210 &pmu_io_domains {
211         vccio4-supply = <&vcc_1v8>;
212         vccio6-supply = <&vcc_1v8>;
213 };
214
```

图 6.2.3.12 配置 pmu_io_domains

将 VCCIO4 的电源配置为 vcc_1v8，将 VCCIO6 的电源配置为 vcc_1v8。

这些需要改动的内容，正点原子都已经配置好了，这里只是给大家讲解如何去配置 IO 电源域。当用户需要做自己的产品、设计自己的硬件时，能够自己去配置这些东西。

3. 编译内核时对 IO 电源域进行确认

在编译 RK 内核源码的过程中，会弹出 IO-Domains 确认对话框，如下图中所示：

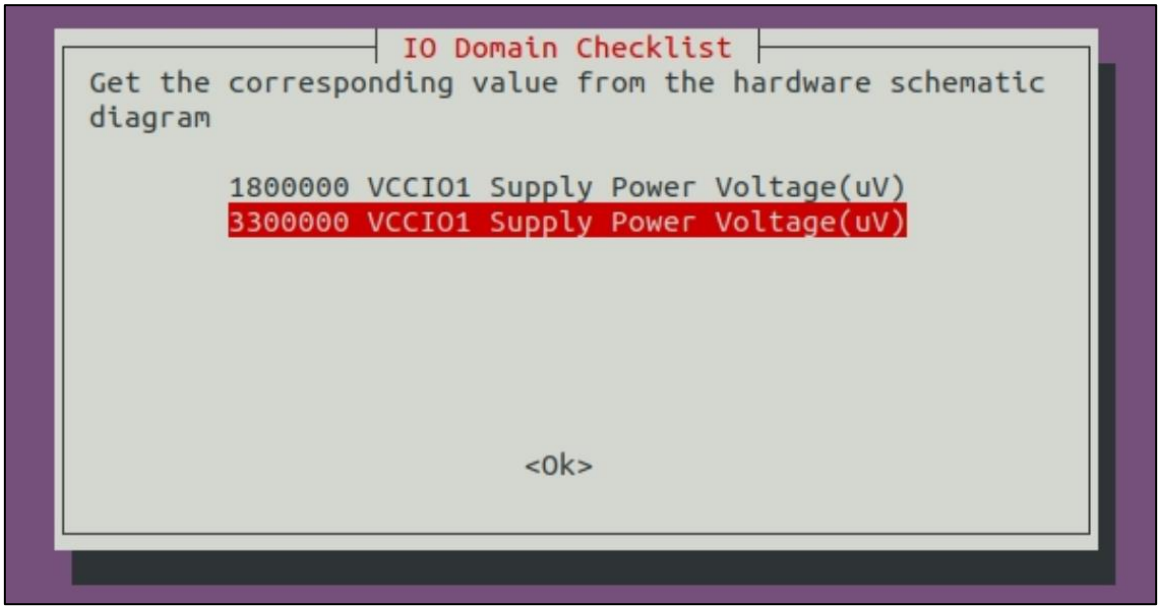


图 6.2.3.13 IO-Domains 确认对话框

弹出这个对话框的目的是**要求开发者再次确认硬件原理图和软件 DTS 的 IO 电压是否匹配**。虽然设备树中已经配置好了 pmu_io_domains 节点，但编译内核时会弹出这个对话框，要求你再次确认，说明这个很重要，**请务必仔细确认**！如果 IO 电压配置不正确，将会导致芯片 IO 烧坏。

当你通过这个对话框完成确认之后，下次编译便不会在出现了（选择的电压值与设备树 pmu_io_domains 节点的配置相同时才会编译通过，否则会编译失败！），如果设备树名字或者设备树中 pmu_io_domains 节点的配置发生了变化，则下次编译会继续弹出该对话框重新进行确认。

4. 编译内核后检查 IO 电源域配置情况

当内核编译成功后，会在**<Kernel>/arch/arm64/boot/dts/rockchip/**目录下生成一个以“.”符号开头（也就是 linux 系统中的隐藏文件）、以 dtb.dts.tmp.domain 结尾的文件，如图所示：

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel/arch/arm64/boot/dts/rockchip$ ls -l .*.tmp.domain
-rw-rw-r-- 1 tgg tgg 259 4月   6 17:49 .rk3568-atk-evb1-ddr4-v10-linux.dtb.dts.tmp.domain
-rw-rw-r-- 1 tgg tgg 259 4月  12 08:46 .rk3568-atk-evb1-mipi-dsi-720p.dtb.dts.tmp.domain
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel/arch/arm64/boot/dts/rockchip$
```

图 6.2.3.14 dtb.dts.tmp.domain 文件

文件的内容就是 IO 电压的配置情况（用户通过 IO-Domains 对话框所选择的结果，进行汇总），如下图所示：

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel/arch/arm64/boot/dts/rockchip$ cat .rk3568-atk-evb1-ddr4-v10-linux.dtb.dts.tmp.domain
PMUIO2 Supply Power Voltage1:3300000
VCCIO1 Supply Power Voltage1:3300000
VCCIO3 Supply Power Voltage1:3300000
VCCIO4 Supply Power Voltage1:1800000
VCCIO5 Supply Power Voltage1:3300000
VCCIO6 Supply Power Voltage1:1800000
VCCIO7 Supply Power Voltage1:3300000
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/kernel/arch/arm64/boot/dts/rockchip$
```

图 6.2.3.15 dtb.dts.tmp.domain 文件的内容

如果把这种 dtb.dts.tmp.domain 文件给删除，那么下次编译内核时又会弹出 IO-Domains 对话框。

6.2.4 替换 logo

包括 u-boot 阶段 logo 以及内核阶段 logo，这两个 logo 图片都存在内核源码目录下：

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

图 6.2.4.1 logo 图片文件

U-Boot logo 图片为 logo.bmp，默认 U-Boot logo 如下所示：

图 6.2.4.2 U-Boot logo

内核 logo 图片为 logo_kernel.bmp，默认内核 logo 如下所示：

图 6.2.4.3 Kernel logo

如果用户需要将其替换为自己的 logo，首先需要将你的 U-Boot logo 图片重命名为 logo.bmp、将你的内核 logo 图片重命名为 logo_kernel.bmp，然后将这两个 bmp 图片文件拷贝到内核源码根目录下，替换内核源码中默认的 logo 图片。

编译内核时，会将 logo.bmp 和 logo_kernel.bmp 打包进 resource.img 中，然后再把 resource.img 打包到 boot.img。

U-Boot 启动的时候会把这两个 bmp 文件加载到内存中，logo.bmp 会在 U-Boot 阶段开始显示，logo_kernel.bmp 在内存中的地址会被 U-Boot 传给 Linux Kernel，在 Linux Kernel 的 DRM 驱动初始化阶段显示。

6.2.5 内核模块开发文档汇总

<SDK>/docs/Common 目录下存放了很多 RK 提供的内核模块开发文档，将这些文档按照功能模块划分放置在不同的目录下、方便用户查找。如果大家在开发过程中遇到了一些问题，可以去看看 RK 提供的这些开发文档，相信会对你有所帮助！

下表对这些文档进行了一个汇总：

模块功能	子目录	对应文档
音频	AUDIO	Rockchip_Developer_Guide_Audio_CN.pdf
摄像头	CAMERA	文档存放在 ISP2X、ISP3X 子目录下
CAN	CAN	Rockchip_Develop_Guide_Can_CN.pdf Rockchip_Develop_Guide_CAN_FD_CN.pdf
时钟配置	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 ……
JTAG/GDB 等常用调试	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
显示	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 ……
CPU/GPU 等频率、电压调节	DVFS	Rockchip_Developer_Guide_CPUFreq_CN.pdf Rockchip_Developer_Guide_Devfreq_CN.pdf
以太网配置	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 通信	I2C	Rockchip_Developer_Guide_I2C_CN.pdf
GPIO 电源域	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
存储	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 配置	Pin-Ctrl	Rockchip_Developer_Guide_Linux_Pinctrl_CN.pdf
PMIC 电量计、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

电源、功耗	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
温控	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
看门狗	WATCHDOG	Rockchip_Developer_Guide_Linux_WDT_CN.pdf

表 6.2.5.1 内核模块开发文档汇总

表当中只是列举了部分文档，并非全部，一切以**<SDK>/docs/Common** 目录为准！

6.2.6 三屏显示

参考文档：**开发板光盘 A 盘-基础资料→10、用户手册→03、辅助文档→【正点原子】ATK-DLRK3568 开发板三屏显示参考手册.pdf**。

6.3 buildroot 开发

Buildroot 源码在<SDK>/buildroot 目录下：

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

图 6.3.1 buildroot 工程目录

推荐用户使用 buildroot 来构建根文件系统，不推荐使用 Yocto！

RK 已经配置好环境变量，BSP 配置以及各模块开发，方便用户基于 RK buildroot 进行开发、定制。

关于 buildroot 的介绍以及使用方法，本文档不作说明，可以参考以下文档：

开发板光盘 A 盘-基础资料→10、用户手册→03、辅助文档→【正点原子】Buildroot 用户手册中文版(正点原子翻译)_V1.0.pdf

开发板光盘 A 盘-基础资料→10、用户手册→03、辅助文档→【正点原子】Buildroot 快速使用手册 V1.0.pdf

编译 RK3568 Linux SDK 过程中，需要编译两次 buildroot：

- **编译 buildroot 得到根文件系统镜像 rootfs.img。rootfs.img 会烧录到开发板 rootfs 分区。**
- **编译 buildroot 得到 ramdisk 根文件系统镜像（进入 recovery 模式时挂载 ramdisk 根文件系统，而 rootfs.img 则是正常启动系统时挂载的根文件系统，这点要搞清楚！）。**

ramdisk 最终会打包进 recovery.img 镜像中，recovery.img 会烧录到开发板 recovery 分区。

6.3.1 buildroot 目录结构介绍

buildroot 源码目录结构如下图所示：

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

接下来简单介绍一下 buildroot 根目录下的文件夹：

顶层目录名	说明
arch	存放 buildroot 支持的所有 CPU 架构相关的配置文件及构建脚本
board	存放特定目标平台相关的文件，譬如内核配置或补丁文件、rootfs 覆盖文件等
boot	存放 buildroot 支持的 BootLoader 相关的补丁、校验文件、构建脚本、配置选项等
build	Buildroot 编译系统相关组件
configs	存放了所有目标平台的 defconfig 配置文件
dl	download 的缩写，该目录用于存放下载的各种开源软件包，譬如 alsa-lib 库、bluez 库、bzip2、curl 工具等；在编译过程中，buildroot 会从网络下载所需软件包、并将其放置在 dl/目录下；如果 buildroot 下载某软件包时失败、无法下载成功，此时我们也可以自己手动下载该软件包、并将其拷贝至 dl 目录下。所有软件包只需下载一次即可，不是每次编译都要下载一次（除非删除 dl 目录），只要 dl 目录下存在该软件包就不用下载了；所以往往第一次编译会比较慢，因为下载过程会占用很多时间
docs	存放相关的参考文档
fs	存放各种文件系统的源代码
linux	存放 linux 的构建脚本和配置选项
output	该文件夹会在编译 buildroot 后出现，output 目录用于存放编译过程中输出的各种文件，包括各种编译生成的中间目标文件、可执行文件、lib 库以及最终烧录到开发板的 rootfs 镜像等
package	存放所有 package（软件包）的构建脚本、配置选项；每个软件包目录下（package/<package_name>/）都有一个 Config.in 文件和 <package_name>.mk 文件（其实就是 Makefile 文件）；如果需要添加一个新的 package，则需对 package/目录进行改动。
support	存放一些为 buildroot 提供功能支持的脚本、配置文件等
toolchain	存放制作各种交叉编译工具链的构建脚本和相关文件，binutils、gcc、gdb、kernel-header 和 uClibc
utils	存放一些 buildroot 的实用脚本和工具

6.3.2 常见编译命令

在 buildroot 源码根目录下有一个 Makefile 文件，如下所示：

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

这就是 buildroot 源码工程的 Makefile 文件，所以我们可以通过 make 命令编译 buildroot，既可以编译整个根文件系统，也可以单独编译指定的 package。

①、编译根文件系统

编译之前，先进行配置；进入到 buildroot 目录下，执行如下命令进行配置（以 rk3568 平台为例）：

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

命令中最后一个参数（rockchip_rk3568）用于指定目标平台的 defconfig 配置文件（不带 **_defconfig** 后缀），所有目标平台的 defconfig 配置文件都存放在**<Buildroot>/configs** 目录下，在该目录下可以找到 rk3568 的配置文件 **rockchip_rk3568_defconfig**，如下所示：

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

配置完成后，直接执行 **make** 或“**make all**”命令编译根文件系统：

```
make
```

或

```
make all
```

使用多线程编译（-jN）：

```
make -j12
```

或

```
make all -j12
```

②、编译 package

编译之前，先进行配置。进入到 buildroot 目录下，执行如下命令进行配置（以 rk3568 平台为例）：

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

配置完成后，执行如下命令编译指定的 package：

```
make <package_name>
```

参数<package_name>用于指定 package 的名字，譬如 rkwifibt：

```
make rkwifibt
```

③、buildroot 清除命令

执行如下命令可删除所有构建时生成的文件（包括 build、host、staging and target trees、images 和 toolchain）：

```
make clean
```

执行如下命令可删除所有构建时生成的文件以及相关配置（彻底清除，相当于删除 output 目录）：

```
make distclean
```

执行如下命令可删除指定 package 的构建目录：

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

6.3.3 编译 RK3568 根文件系统镜像 rootfs.img

本小节介绍编译 buildroot 得到 RK3568 平台根文件系统镜像 rootfs.img，6.4 小节介绍如何通过 buildroot 编译出 ramdisk 根文件系统镜像。

编译之前先进行 defconfig 配置，在 buildroot 目录下执行如下命令：

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

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

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

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

图 6.3.3.1 执行 defconfig 配置(1)

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

图 6.3.3.2 执行 defconfig 配置(2)

命令中最后一个参数（rockchip_rk3568）用于指定目标平台的 defconfig 配置文件（不带 **_defconfig** 后缀），所有目标平台的 defconfig 配置文件都存放在**<Buildroot>/configs** 目录下，在该目录下可以找到 rk3568 的配置文件 **rockchip_rk3568_defconfig**，如下所示：

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

图 6.3.3.3 rockchip_rk3568_defconfig 配置文件

接着执行 make 命令或“make all”命令编译根文件系统：

```
make -j16
```

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

图 6.3.3.4 编译根文件系统

整个编译过程将会持续很长一段时间，需要耐心等待！

编译完成后，如下图所示：

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

图 6.3.3.5 根文件系统编译成功

编译生成的根文件系统镜像存放在 output/rockchip_rk3568/images 目录下：

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

图 6.3.3.6 rootfs 镜像

该目录下生成了多个不同格式的根文件系统镜像，RK3568 平台上使用的是 **rootfs.ext4**（ext4 格式，由 SDK 板级配置文件中的 **RK_ROOTFS_TYPE** 变量决定），rootfs.ext4 是一个软链接文件，实际指向的镜像为 rootfs.ext2，使用 file 命令查看 rootfs.ext2，如下所示：

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

图 6.3.3.7 file 命令查看 rootfs.ext2

可知，rootfs.ext2 就是 ext4 格式根文件系统镜像。将其重命名为 **rootfs.img**，rootfs.img 最终会烧录到开发板 rootfs 分区，作为正常启动系统时挂载的根文件系统。

直接使用 make 命令进行编译、编译过程会产生大量的输出信息，看起来非常乱，让人感觉不爽！想要在编译过程中过滤掉这些大量无用的输出信息，可以使用下面这条命令进行编译：

```
utils/brmake -j16
```

brmake 是一个 shell 脚本，使用该脚本进行编译，可以过滤掉绝大部分不重要的输出信息、保留关键信息，使终端的输出内容变的简洁。

6.3.4 output 目录介绍

output 目录用于存放编译过程中产生的各种文件，包括编译过程生成的中间目标文件、可执行文件、lib 库以及最终烧录到开发板的 rootfs 镜像等。

output 目录结构如下所示：

```
output/
├── rockchip_rk3568
│   ├── build          #包含所有构建的软件包，包括 buildroot 在宿主机上所需的工具以及为目标平台编译的软件包
│   ├── host           #包含为 Ubuntu 主机（宿主机）构建的工具，以及目标工具链
│   ├── images         #存放最终编译输出的镜像
│   ├── Makefile
│   ├── staging        #一个指向 host/目录中目标工具链 sysroot 的符号链接
│   └── target         #根文件系统系统目录，用来创建根文件系统镜像，rootfs.img 镜像的内容就是该目录下的内容
```

6.3.5 package 编译

本小节讲一下如何编译指定的软件包（package）。

首先，编译之前先进行配置；这里不再多说。接着执行如下命令编译指定的 package：

```
make <package_name>
```

或

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

参数 package_name 用于指定 package 的名字，譬如 rkwifibt：

```
make rkwifibt
```

或

```
utils/brmake rkwifibt
```

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

问题的关键在于，参数 package_name 如何确定/得知？

其实参数 package_name 对应的便是 buildroot/package/文件夹中的某个目录对应的名字（或某个子目录对应的名字），并且该目录中存在 Config.in 配置文件（用于定义 package 配置选项）以及<package_name>.mk 文件（Makefile 文件，用于定义 package 的构建逻辑）。譬如：

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

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

参数 package_name 指的便是 Config.in 和<package_name>.mk 文件所在目录对应的名字。

package（软件包）的构建过程可以分解为：configure（配置）、build（编译）、install（安装），其执行顺序为：**配置→编译→安装**，每一步可单独执行，执行如下命令：

```
make <package_name>-configure      #执行配置命令
make <package_name>-build          #执行编译命令
make <package_name>-install        #执行安装命令
```

或者

```
utils/brmake <package_name>-configure      #执行配置命令
utils/brmake <package_name>-build          #执行编译命令
utils/brmake <package_name>-install        #执行安装命令
```

以 rkwifibt 为例：

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

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

与之对应的还有：reconfigure（重新配置）、rebuild（重新编译）、reinstall（重新安装）：

```
make <package_name>-reconfigure    #从配置阶段开始，重新启动 package 的构建过程
make <package_name>-rebuild        #从编译阶段开始，重新启动 package 的构建过程
make <package_name>-reinstall      #从安装阶段开始，重新启动 package 的构建过程
```

或者

```
utils/brmake <package_name>-reconfigure  #从配置阶段开始，重新启动 package 的构建过程
utils/brmake <package_name>-rebuild      #从编译阶段开始，重新启动 package 的构建过程
utils/brmake <package_name>-reinstall    #从安装阶段开始，重新启动 package 的构建过程
```

以 rkwifibt 为例：

```
utils/brmake rkwifibt-reconfigure
```

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

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$ utils/brmake rkwifibt-reconfigure
2023-07-13T15:56:16 >>> rkwifibt 1.0.0 Syncing from source dir /home/tgg/rk3568_linux_sdk/buildroot/../external/rkwifibt
2023-07-13T15:56:16 >>> rkwifibt 1.0.0 Configuring
2023-07-13T15:56:16 >>> rkwifibt 1.0.0 Building                    重新执行配置-->编译-->安装
2023-07-13T15:56:23 >>> rkwifibt 1.0.0 Installing to target
Done in 11s
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$ utils/brmake rkwifibt-rebuild
2023-07-13T15:56:29 >>> rkwifibt 1.0.0 Syncing from source dir /home/tgg/rk3568_linux_sdk/buildroot/../external/rkwifibt
2023-07-13T15:56:29 >>> rkwifibt 1.0.0 Building
2023-07-13T15:56:36 >>> rkwifibt 1.0.0 Installing to target    重新执行编译-->安装
Done in 11s
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$ utils/brmake rkwifibt-reinstall
2023-07-13T15:56:42 >>> rkwifibt 1.0.0 Syncing from source dir /home/tgg/rk3568_linux_sdk/buildroot/../external/rkwifibt
2023-07-13T15:56:42 >>> rkwifibt 1.0.0 Installing to target
Done in 4s                                                        重新执行安装
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$
```

如果需要清理 package（软件包）构建目录，可以执行如下命令：

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

以 rkwifibt 为例：

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

6.3.6 添加 package

下面通过一个简单的例子向用户介绍如何在 buildroot/package 目录下添加一个自己的 package（软件包）。

1. 开发源码工程

首先进入<SDK>/app 目录下，在该目录下创建一个名为“mypackage”的文件夹，如下所示：

```
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
```

图 6.3.6.1 创建 mypackage 文件夹

在 mypackage 目录下创建一个.c 源文件 main.c，以及一个 Makefile 文件：

```
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$
```

图 6.3.6.2 创建源文件和 Makefile

大家可以自己在 main.c 源文件中编写一个简单的测试代码，譬如打印一个“Hello World”，Makefile 文件中的内容如下所示：

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

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

目的就是将 main.c 源文件编译成一个可执行文件 mypackage。

2. 添加 package

进入<Buildroot>/package 目录，在该目录下创建一个名为 mypackage 的目录：

```
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$
```

图 6.3.6.3 创建 mypackage 目录

在 mypackage 目录下创建两个文件：Config.in 和 mypackage.mk：

```
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$
```

图 6.3.6.4 创建 Config.in 和 mypackage.mk

Config.in 文件的内容如下所示：

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

Config.in 文件的语法规则与 Linux Kernel、U-Boot 中 Kconfig 文件的语法规则是一样的。

mypackage.mk 文件的内容如下所示：

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

# 给你的软件包定义一个版本号
MYPACKAGE_VERSION = 1.0

# 你的软件包所在目录
MYPACKAGE_SITE = $(TOPDIR)/../app/mypackage

# 获取软件包的方式，local 表示从本地获取，有些包可能需要通过网络下载，譬如 git 仓库中的项目
MYPACKAGE_SITE_METHOD = local

# 列出在编译软件包之前 需要执行的配置操作
define MYPACKAGE_CONFIGURE_CMDS
endef

# 列出编译软件包时 需要执行的操作
define MYPACKAGE_BUILD_CMDS
	$(MAKE) -C $(@D) CC=$(TARGET_CC)
endef

# 列出将软件包安装到 target 目录(<Buildroot>/output/rockchip_rk3568/target)时需要执行的操作
define MYPACKAGE_INSTALL_TARGET_CMDS
	$(INSTALL) -D -m 0755 $(@D)/mypackage $(TARGET_DIR)/usr/bin/mypackage
```

```
endef

# 表示当前软件包是一个通用型软件包基础结构
$(eval $(generic-package))
```

注意：该文件中定义了一些变量以及宏，所有的这些变量、宏都以前缀 MYPACKAGE_开头，不能乱来，它必须等于 Config.in、mypackage.mk 文件所在目录（mypackage）对应的名字（小写字母转换为大写）。

上面已经解释了这些变量、宏的作用，除了这些变量、宏之外，还可以在.mk 文件中定义很多其它的变量或者宏，每个变量或宏都有自己的意义，详细的使用方法请用户阅读文档：**开发板光盘 A 盘-基础资料→10、用户手册→03、辅助文档→【正点原子】Buildroot 用户手册中文版(正点原子翻译)_V1.0.pdf**。

$(MAKE)：表示 make 命令；

$(@D)：表示软件包所在目录，**注意这个目录并不是<SDK>/app/mypackage、而是该软件包在 output/rockchip_rk3568/build/目录下对应的文件夹**；编译软件包之前，buildroot 会将<SDK>/app/mypackage 拷贝至<Buildroot>/output/rockchip_rk3568/build/目录，并重命名为 mypackage-1.0（1.0 就是版本号）。所以这个“$(@D)”指的是 output/rockchip_rk3568/build/mypackage-1.0 这个目录。

$(TOPDIR)：表示 buildroot 顶层目录，也就是<SDK>/buildroot 目录。

$(TARGET_CC)：表示交叉编译器，RK 平台默认使用 buildroot 交叉编译器，交叉编译器所在路径为：**<Buildroot>/output/rockchip_rk3568/host/bin/aarch64-buildroot-linux-gnu-gcc**。

$(INSTALL)：表示 install 命令。

$(TARGET_DIR)：表示 target 目录<Buildroot>/output/rockchip_rk3568/target。

关于 mypackage.mk 文件的内容就给大家讲这么多，详情请参考《**【正点原子】Buildroot 用户手册中文版(正点原子翻译)_V1.0.pdf**》文档。

接下来打开 package/Config.in 文件，将下面这行内容添加到该文件中：

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

```
menu "Target packages"

        source "package/busybox/Config.in"
        source "package/rockchip/Config.in"
        source "package/mypackage/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-systemd/Config.in"
        source "package/skeleton-init-sysv/Config.in"

menu "Audio and video applications"
        source "package/alsa-utils/Config.in"
        source "package/alsa-plugins/Config.in"
        source "package/atest/Config.in"
        source "package/aumix/Config.in"
        source "package/bellagio/Config.in"
        source "package/bluez-alsa/Config.in"
        source "package/dvblast/Config.in"
        source "package/dvdauthor/Config.in"
        source "package/dvdrw-tools/Config.in"
        source "package/espeak/Config.in"
        source "package/faad2/Config.in"
```

添加这行

添加完成后保存退出。

3. 使能并编译 package

执行“make menuconfig”打开图形化配置界面，找到我们添加的 package，然后将其使能：

```
Target packages    --->
```

my package

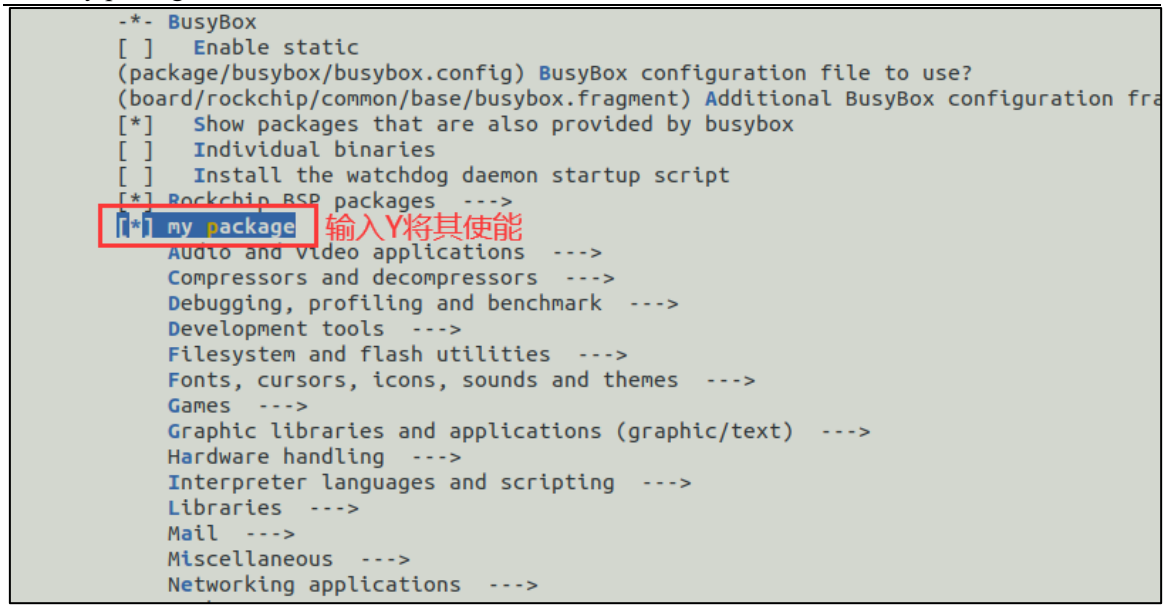


图 6.3.6.5 使能 mypackage

然后保存配置、退出图形化配置界面。

执行如下命令编译该软件包，如下所示：

```
make mypackage-rebuild
```

或

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

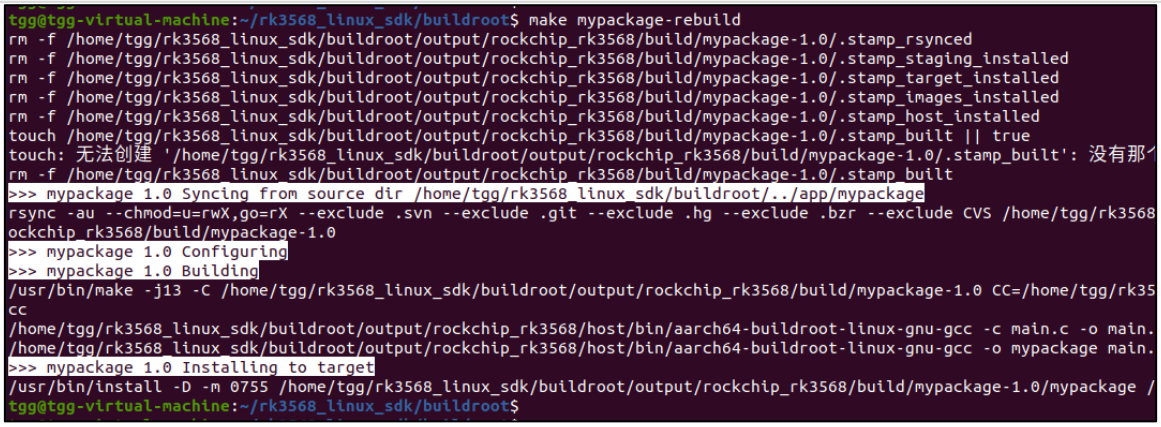


图 6.3.6.6 编译 mypackage

编译生成一个可执行文件 mypackage，其所在路径为：output/rockchip_rk3568/target/usr/bin/mypackage。

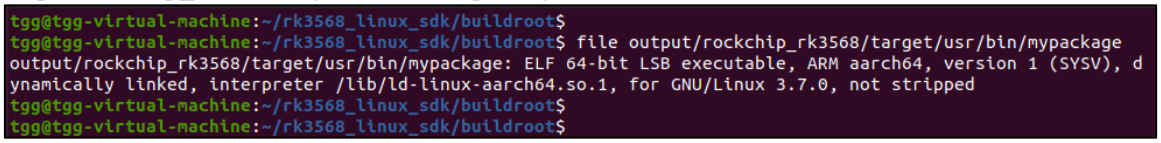


图 6.3.6.7 mypackage 可执行文件

6.3.7 rootfs 定制

用户可针对自己的 Linux 产品对 rootfs 进行定制，使得能够在满足产品功能的前提下、尽量减小 rootfs 的体积。

6.4 recovery 开发

recovery.img 主要用于让设备进入 recovery 模式，设备进入 recovery 模式的主要作用是擦除用户数据（譬如 userdata 分区数据）以及系统升级或者其它操作等。

recovery.img 会烧录到开发板 recovery 分区。recovery.img 也是一种 FIT 格式镜像，由多个镜像合并而成，包括 ramdisk（进入 recovery 模式时挂载该根文件系统）、内核 DTB、内核镜像以及 resource.img。U-Boot 会根据 misc 分区存放的 BCB 数据块来判断将要引导系统进入正常启动模式（Normal 模式）还是 recovery 模式。

如果判断需要进入 recovery 模式，那么 U-Boot 会从 recovery.img（存放在 recovery 分区）中读取资源镜像 resource.img，再从 resource.img 中找到内核 DTB 并将其加载到内存（内核 DTB 打包进 resource.img、resource.img 再打包进 recovery.img）。然后会加载和引导 recovery.img 中包含的内核镜像，最终进入系统后挂载 ramdisk 根文件系统。

所以 recovery.img 就是为系统进入 recovery 模式而准备的。

6.4.1 编译 recovery

recovery.img 由内核镜像 Image、内核 DTB、resource.img 以及 ramdisk 根文件系统打包而成，所以要制作 recovery.img，分为以下三步：

①、编译 Linux Kernel，得到内核镜像 Image、内核 DTB 以及 resource.img；
②、编译 buildroot 得到 ramdisk 根文件系统镜像；
③、最后打包成 recovery.img。

1. 编译 Linux Kernel

进入内核源码目录<SDK>/kernel，直接运行 make.sh 脚本编译内核，编译完成将会得到内核镜像 arch/arm64/boot/Image、以及资源镜像 resource.img。

2. 编译 buildroot 得到 ramdisk 根文件系统镜像

我们现在需要通过 buildroot 编译出一个 recovery 模式下的根文件系统。Recovery 模式下使用 ramdisk 根文件系统（使用内存模拟的根文件系统），所以这个根文件系统比较小巧、体积不大。

编译之前，先进行配置，进入到 buildroot 目录下，执行如下命令进行 defconfig 配置。

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

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

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

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

图 6.4.1.1 执行 defconfig 配置

命令中最后一个参数（rockchip_rk356x_recovery）用于指定目标平台的 defconfig 配置文件（不带**_defconfig** 后缀）。RK3568 平台 ramdisk 根文件系统对应的 defconfig 配置文件为 configs/rockchip_rk356x_recovery_defconfig；6.3.3 小节中基于 rockchip_rk3568_defconfig 编译出来的根文件系统是 RK3568 开发板正常启动系统时挂载的根文件系统，也就是 Normal 模式下的根文件系统，这点大家要搞清楚！

rockchip_rk356x_recovery_defconfig	#用于编译 ramdisk 根文件系统（recovery 模式）
rockchip_rk3568_defconfig	#用于编译正常启动系统时挂载的根文件系统

接着进行编译，执行如下命令：

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

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

图 6.4.1.2 编译 buildroot(1)

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

图 6.4.1.3 编译 buildroot(2)

编译生成的根文件系统镜像存放在 output/rockchip_rk356x_recovery/images/目录下：

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

图 6.4.1.4 根文件系统镜像

该目录下生成了多个不同格式的根文件系统镜像，**rootfs.cpio.gz** 便是我们所需的 ramdisk 根文件系统镜像。

3. 制作 recovery.img

接下来需要去制作 recovery.img。将内核镜像 Image、内核 DTB、resource.img 以及 ramdisk 根文件系统镜像 rootfs.cpio.gz 打包成一个 recovery.img。进入 buildroot 目录下，执行如下命令打包（命令比较长，其实是一条命令，<SDK>表示 SDK 顶层目录，用户需根据自己的实际情况进行填写）：

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

```
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$ ../device/rockchip/common/mk-fitimage.sh buildroot/output/rockchip_rk356
x_recovery/images/recovery.img device/rockchip/rk356x/boot4recovery.its /home/tgg/rk3568_linux_sdk/buildroot/output/rockchip_r
k356x_recovery/images/rootfs.cpio.gz
fdt {
kernel {
ramdisk {
resource {
FIT description: U-Boot FIT source file for arm
Created:         Sat Apr 15 23:15:27 2023
 Image 0 (fdt)
  Description:  unavailable
  Created:      Sat Apr 15 23:15:27 2023
  Type:         Flat Device Tree
  Compression:  uncompressed
  Data Size:    142051 Bytes = 138.72 KiB = 0.14 MiB
  Architecture: AArch64
  Load Address: 0xffffff00
  Hash algo:    sha256
  Hash value:   9e918b39233b06917557a36d2e8d5abfddb35f970eac89be44a57923924c0a5b
 Image 1 (kernel)
  Description:  unavailable
  Created:      Sat Apr 15 23:15:27 2023
  Type:         Kernel Image
  Compression:  uncompressed
  Data Size:    22603784 Bytes = 22074.01 KiB = 21.56 MiB
  Architecture: AArch64
  OS:           Linux
  Load Address: 0xffffff01
  Entry Point:  0xffffff01
  Hash algo:    sha256
  Hash value:   638f5b1ff43672df49ccdfb5b37393dfadd3a9a17cfcb318b62807df30db3d87
 Image 2 (ramdisk)
  Type:         RAMDisk Image
  Compression:  uncompressed
  Data Size:    7805956 Bytes = 7623.00 KiB = 7.44 MiB
  Architecture: AArch64
  OS:           Linux
  Load Address: 0xffffff02
  Entry Point:  unavailable
  Hash algo:    sha256
  Hash value:   4f0e62b81e22ce1257f26bb597c587ebc9fcbebc4887b43101d22065a7389365
 Image 3 (resource)
  Description:  unavailable
  Created:      Sat Apr 15 23:15:27 2023
  Type:         Multi-File Image
  Compression:  uncompressed
  Data Size:    786432 Bytes = 768.00 KiB = 0.75 MiB
  Hash algo:    sha256
  Hash value:   96a6fec15606489cd301fdebe95e782f881be7af3bd5154ae8d7d802737ac730
 Default Configuration: 'conf'
 Configuration 0 (conf)
  Description:  unavailable
  Kernel:       kernel
  Init Ramdisk: ramdisk
  FDT:          fdt
tgg@tgg-virtual-machine:~/rk3568_linux_sdk/buildroot$
```

制作 boot.img 时用的也是 device/rockchip/common/mk-fitimage.sh 脚本，不过它们使用的 its 文件不一样：

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

命令中的第一个参数表示输出文件 recovery.img 的输出路径，如果使用相对地址、则必须相对于 SDK 根目录而言。

命令中的第二个参数表示 its 文件的路径，如果使用相对地址、则必须相对于 SDK 根目录而言。

命令中的第三个参数表示 ramdisk 根文件系统镜像的路径，这里使用的是绝对路径，大家需要根据自己的实际情况进行修改。

上述命令中，仅仅指定了 ramdisk 的路径、并未指定内核镜像（第四个参数可用于指定内核镜像路径）、内核 DTB 以及 resource.img 的路径，因为它们都有默认值：

内核镜像 Image ---> kernel/arch/arm64/boot/Image（板级配置文件中 RK_KERNEL_IMG 变量所指定的镜像）
内核 DTB ---> kernel/arch/arm64/boot/dts/rockchip/rk3568-atk-evb1-ddr4-v10-linux.dtb（由板级配置文件中的 RK_KERNEL_DTS 决定）
Resource.img ---> kernel/resource.img

最终生成的 recovery.img 存放在<Buildroot>/output/rockchip_rk356x_recovery/images/目录下：

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

6.4.2 参考文档

Recovery 相关参考文档存放在<SDK>/docs/Linux/Recovery 目录下。

6.5 oem 和 userdata

oem.img 包含了厂家的 APP 或数据，该镜像会烧录到开发板的 oem 分区，系统启动之后会被挂载到根文件系统/oem 目录。userdata.img 包含了最终用户的 APP 或数据，该镜像会烧录到开发板的 userdata 分区，系统启动之后会被挂载到根文件系统/userdata 目录。

编译 Linux SDK 后（需执行“**./build.sh firmware**”或“**./mkfirmware.sh**”），会在 rockdev/目录下生成 oem.img 以及 userdata.img，使用 file 命令查看这两个镜像，如下所示：

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

由此可知，这两个镜像均是 ext2 格式文件系统镜像（分别由 SDK 板级配置文件中的 **RK_USERDATA_FS_TYPE** 和 **RK_OEM_FS_TYPE** 变量决定），可以直接使用 mount 命令将其挂载到 Ubuntu 系统进行查看。

这两个镜像均是由 mkfirmware.sh 脚本制作而成（该脚本最终会调用 device/rockchip/common/mk-image.sh 脚本，详细过程请阅读 mk-image.sh 脚本内容）。oem.img 和 userdata.img 的大小通常是自动的，无需配置；如有特殊需求，也可在 SDK 板级配置文件中定义 **RK_OEM_PARTITION_SIZE** 或 **RK_USERDATA_PARTITION_SIZE** 变量手动指定镜像大小，未定义这些变量则表示由脚本自动确定大小。

oem.img 由 device/rockchip/oem/oem_xxx（由板级配置文件中的 RK_OEM_DIR 决定）目录创建而成；userdata.img 由 device/rockchip/userdata/userdata_xxx（由板级配置文件中的 RK_USERDATA_DIR 决定）目录创建而成；如下所示：

```
51 #OEM config
52 export RK_OEM_DIR=oem_normal
53 # OEM build on buildroot
54 #export RK_OEM_BUILDIN_BUILDROOT=YES
55 #userdata config
56 export RK_USERDATA_DIR=userdata_normal
```

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

由此可见，oem.img 镜像的内容就是 device/rockchip/oem/oem_normal 目录下的内容；userdata.img 镜像的内容就是 device/rockchip/userdata/userdata_normal 目录下的内容。

如果 oem_normal 或 userdata_normal 目录下的内容发生了更改，则需重新打包生成 oem.img 或 userdata.img；先删除 rockdev 目录下的 oem.img 和 userdata.img，然后再次执行./mkfirmware.sh 脚本创建即可。

6.6 parameter.txt 分区表文件

parameter.txt 是 RK 平台的分区表文件，是一个 TXT 文本文件。烧写镜像时，并不需要将 parameter.txt 文件烧录到 Flash 器件（譬如 eMMC），但是会读取它的信息来定义存储器的物理分区，各分区的分区名、分区的起始地址以及分区大小均由 parameter.txt 文件定义。

SDK 板级配置文件中（device/rockchip/rk356x/BoardConfig-rk3568-atk-evb1-ddr4-v10.mk）定义了 ATK-DLRK3568 Linux 系统所使用的分区表文件，由 RK_PARAMETER 变量决定，如下所示：

```
20 export RK_KERNEL_FIT_ITS=boot.its
21 # parameter for GPT table
22 export RK_PARAMETER=parameter-buildroot-fit.txt
23 # Buildroot config
24 export RK_CFG_BUILDROOT=rockchip_rk3568
25 # Recovery config
```

该变量指向的文件便是 device/rockchip/rk356x/parameter-buildroot-fit.txt。所以编译 Linux SDK 后，rockdev/目录下生成的 parameter.txt 分区表文件其实就是由该文件（parameter-buildroot-fit.txt）copy 过去的。

6.7 misc.img 镜像

misc.img 镜像会烧录到开发板 misc 分区。

misc 分区的概念来源于 Android 系统，在 Android 系统中，misc 分区是一个很重要的分区，其主要用于 Android 系统和 BootLoader 之间的通信，misc 分区中包含了 BCB（BootLoader Control Block）数据块，BCB 是 Android 控制系统启动流程而设计的一种和 BootLoader 交互的机制。

BCB 数据结构如下图所示：

```
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：启动命令，目前支持以下三个：

参数	功能
bootonce-bootloader	启动进入 U-Boot fastboot
boot-recovery	启动进入 recovery
boot-fastboot	启动进入 recovery fastboot（简称 fastbootd）

- recovery：为进入 recovery 模式的附带命令，开头为”recovery\n”，后面可以带多个参数，以”--”开头，以”\n”结尾，例如” recovery\n--wipe_ab\n--wipe_package_size=345\n--reason=wipePackage\n”：

参数	功能
update_package	OTA 升级
retry_count	进 recovery 升级次数，比如升级时意外掉电，依据该值重新进入 recovery 升级
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	擦除整个 userdata 分区
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 卡固件升级
factory_mode	工厂模式，主要用于做一些设备测试，如 PCBA 测试
pcba_test	进入 PCBA 测试
resize_partition	重新规划分区大小，android Q 的动态分区支持
rk_fwupdate	指定rk SD/USB固件升级，作用域仅限于U-Boot

Linux 系统中常用来作为系统升级时或者恢复出厂设置时使用。U-Boot 阶段会读取 misc 分区中的 BCB 数据，根据 BCB 数据中的 command 命令内容决定是进入正常系统还是 recovery 模式（command 为 boot-recovery，则进入 recovery 模式；command 为空，则进入正常系统）。

设备进入 recovery 模式，会读取 misc 分区中 recovery 部分的内容，从而执行不同的动作，或升级分区固件，或擦除用户分区数据，或其它操作等等。

misc.img 镜像都是预编译好的，RK 提供了多个 misc.img 镜像供用户选择，这些镜像存放在 device/rockchip/rockimg/目录下，如下所示：

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

这些 misc.img 镜像中，常用的是 blank-misc.img 和 wipe_all-misc.img，另外几个不常用。它们之间的区别就在于 BCB 数据块中 command 以及 recovery 标识不同，我们可以使用 winhex 或 ultraEdit 等工具，以二进制形式打开 misc.img 镜像文件，找到 BCB 数据块所在位置（偏移 16KB 位置，0x4000），如下所示：

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

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

图 1 pcba_small_misc.img 镜像

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

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

图 2 wipe_all-misc.img 镜像

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

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

图 3 blank-misc.img 镜像

关于这些标识，上文已有介绍（对比 BCB 数据结构）。

当前使用的 misc.img 镜像由 SDK 板级配置文件中（device/rockchip/rk356x/BoardConfig-rk3568-atk-evb1-ddr4-v10.mk）的 RK_MISC 变量决定，如下所示：

```
56 export RK_USERDATA_DIR=userdata_normal
57 #misc image
58 export RK_MISC=blank-misc.img
59 #choose enable distro module
60 export RK_DISTRO_MODULE=
```

该变量指定的文件便是 rockimg 目录下的 blank-misc.img。所以编译 Linux SDK 后，rockdev/目录下生成的 misc.img 镜像其实就是由该文件（blank-misc.img）copy 过去的。

6.6 Qt 应用开发

参考文档：

开发板光盘 A 盘-基础资料→10、用户手册→03、辅助文档→【正点原子】ATK-DLRK3568 Qt 开发环境搭建.pdf。

开发板光盘 A 盘-基础资料→10、用户手册→02、开发文档→【正点原子】ATK-DLRK3568 嵌入式 Qt 开发实战。

6.7 其它应用开发

ApplicationNote 参考文档：

<SDK>/docs/Linux/ApplicationNote

```
├──── Rockchip_Developer_Guide_Debian_CN.pdf
├──── Rockchip_Developer_Guide_Debian_Docker_EN.pdf
├──── Rockchip_Developer_Guide_Debian_EN.pdf
├──── Rockchip_Developer_Guide_Linux_Docker_Deploy_CN.pdf
├──── Rockchip_Developer_Guide_Linux_Flash_Open_Source_Solution_CN.pdf
├──── Rockchip_Developer_Guide_Linux_Flash_Open_Source_Solution_EN.pdf
├──── Rockchip_Developer_Guide_Linux_PCBA_CN.pdf
├──── Rockchip_Developer_Guide_Linux_PCBA_EN.pdf
```

```
├──── Rockchip_Developer_Guide_Third_Party_System_Adaptation_CN.pdf
├──── Rockchip_Developer_Guide_Third_Party_System_Adaptation_EN.pdf
├──── Rockchip_Instruction_Linux_ROS2_CN.pdf
├──── Rockchip_Instruction_Linux_ROS_CN.pdf
├──── Rockchip_Instruction_Linux_ROS_EN.pdf
├──── Rockchip_Quick_Start_Linux_USB_Gadget_CN.pdf
├──── Rockchip_Quick_Start_Linux_USB_Gadget_EN.pdf
├──── Rockchip_Use_Guide_Linux_RetroArch_CN.pdf
├──── Rockchip_User_Guide_Linux_RetroArch_EN.pdf
└──── Rockchip_User_Guide_SDK_Docker_CN.pdf
```

Camera 摄像头开发相关文档：

<SDK>/docs/Linux/Camera

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

WIFI/蓝牙开发相关文档：

<SDK>/docs/Linux/Wifibt

```
├──── AP 模组 RF 测试文档
│     ├──── BT RF Test Commands for Linux-v05.pdf
│     └──── Wi-Fi RF Test Commands for Linux-v03.pdf
├──── REALTEK 模组 RF 测试文档
│     ├──── 00014010-WS-170731-RTL8723D_COB_MP_FLOW_R04.pdf
│     ├──── MP tool user guide for linux20180319.pdf
│     └──── Quick_Start_Guide_V6.txt
├──── RK 平台_RTL8723DS_AIRKISS 配网说明.pdf
├──── Rockchip_Developer_Guide_DeviceIo_Bluetooth_CN.pdf
├──── Rockchip_Developer_Guide_DeviceIo_Bluetooth_EN.pdf
├──── Rockchip_Developer_Guide_Linux_WIFI_BT_CN.pdf
├──── Rockchip_Developer_Guide_Linux_WIFI_BT_EN.pdf
├──── Rockchip_Developer_Guide_Network_Config_CN.pdf
├──── Rockchip_Developer_Guide_Network_Config_EN.pdf
├──── Rockchip_Introduction_RK3308_DeviceIo_WIFI_CN.pdf
└──── Rockchip_Introduction_RK3308_DeviceIo_WIFI_EN.pdf
```

<SDK>/external/deviceio_release/doc

```
├──── Rockchip_Developer_Guide_DeviceIo_Bluetooth_CN.pdf
├──── Rockchip_Developer_Guide_Linux_WIFI_BT_CN.pdf
├──── Rockchip_Developer_Guide_Network_Config_CN.pdf
└──── Rockchip_Introduction_RK3308_DeviceIo_WIFI_CN.pdf
```

<SDK>/external 目录下的文档汇总：

<SDK>/external

```
./rkfacial/doc/Rockchip_Instruction_Rkfacial_CN.pdf
./uvc_app/doc/zh-cn/Rockchip_Introduction_Linux_UVCApp_CN.pdf
./rockit/mpi/doc/Rockchip_FAQ_MPI_CN.pdf
./rockit/mpi/doc/Rockchip_Developer_Guide_MPI_CN.pdf
```

./rockit/tgi/doc/Rockchip_Developer_Guide_Linux_Rockit_CN.pdf
./uac_app/doc/zh-cn/Rockchip_Developer_Guide_Microphone_Array_TEST_CN.pdf
./uac_app/doc/zh-cn/Rockchip_Developer_Guide_Linux_UACApp_CN.pdf
./uac_app/doc/zh-cn/Rockchip_Quick_Start_Linux_UAC_CN.pdf
./deviceio_release/doc/Rockchip_Developer_Guide_DeviceIo_Bluetooth_CN.pdf
./deviceio_release/doc/Rockchip_Introduction_RK3308_DeviceIo_WIFI_CN.pdf
./deviceio_release/doc/Rockchip_Developer_Guide_Network_Config_CN.pdf
./deviceio_release/doc/Rockchip_Developer_Guide_Linux_WIFI_BT_CN.pdf
./camera_engine_rkaiq/rkisp2x_tuner/Doc/Rockchip_IQ_Tools_Guide_ISP2x_CN_v2.0.2.pdf
./rknn-toolkit2/doc/Rockchip_Quick_Start_RKNN_Toolkit2_CN-1.3.0.pdf
./rknn-toolkit2/doc/Rockchip_Quick_Start_RKNN_Toolkit2_EN-1.3.0.pdf
./rknn-toolkit2/doc/Rockchip_User_Guide_RKNN_Toolkit2_EN-1.3.0.pdf
./rknn-toolkit2/doc/Rockchip_User_Guide_RKNN_Toolkit2_CN-1.3.0.pdf
./rknn-toolkit2/rknn_toolkit_lite2/docs/Rockchip_User_Guide_RKNN_Toolkit_Lite2_V1.3.0_EN.pdf
./rknn-toolkit2/rknn_toolkit_lite2/docs/Rockchip_User_Guide_RKNN_Toolkit_Lite2_V1.3.0_CN.pdf
./rknpu2/doc/Rockchip_RKNPU_User_Guide_RKNN_API_V1.3.0_EN.pdf
./rknpu2/doc/Rockchip_RKNPU_User_Guide_RKNN_API_V1.3.0_CN.pdf
./rknpu2/doc/Rockchip_RV1106_Quick_Start_RKNN_SDK_V1.3.0b0_CN.pdf
./rknpu2/doc/Rockchip_Quick_Start_RKNN_SDK_V1.3.0_CN.pdf

Linux C 应用编程参考手册（正点原子提供）：
开发板光盘 A 盘-基础资料→10、用户手册→02、开发文档→【正点原子】Linux_C 应用编程参考手册 V1.0.pdf

6.8 Yocto 开发

不推荐用户使用 Yocto！正点原子 Linux 技术团队将不会对其进行维护。
更多资料参考：http://opensource.rock-chips.com/wiki_Yocto。

6.9 NPU 开发

RK 参考文档列表：
external/rknn-toolkit2/doc
├——— Rockchip_Quick_Start_RKNN_Toolkit2_CN-1.3.0.pdf
├——— Rockchip_Quick_Start_RKNN_Toolkit2_EN-1.3.0.pdf
├——— Rockchip_User_Guide_RKNN_Toolkit2_CN-1.3.0.pdf
├——— Rockchip_User_Guide_RKNN_Toolkit2_EN-1.3.0.pdf

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

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

6.10 Debian 开发

本 SDK 提供对 Debian 10 的支持，基于 X11 显示架构。
系统基于 Linaro 版本，添加了图形和视频加速的支持，包括 libmali、xserver、gstreamer rockchip 等 package。
Debian 开发文档参考（正点原子提供）：

Debian 开发文档参考（RK 提供）：
<SDK>/docs/Linux/ApplicationNote/Rockchip_Developer_Guide_Debian_Docker_EN.pdf
<SDK>/docs/Linux/ApplicationNote/Rockchip_Developer_Guide_Debian_CN.pdf

6.11 多媒体开发

RK 参考文档：
<SDK>/docs/Linux/Multimedia
├── Rockchip_Developer_Guide_Linux_RGA_CN.pdf
├── Rockchip_Developer_Guide_Linux_RGA_FAQ_CN.pdf
├── Rockchip_Developer_Guide_Linux_RKADK_CN.pdf
├── Rockchip_Developer_Guide_MPP_CN.pdf
├── Rockchip_Developer_Guide_MPP_EN.pdf
├── Rockchip_Introduction_Linux_Audio_3A_Algorithm_CN.pdf
├── Rockchip_Introduction_Linux_Audio_3A_Algorithm_EN.pdf
├── Rockchip_User_Guide_Linux_Gstreamer_CN.pdf
└── Rockchip_User_Guide_Linux_Gstreamer_EN.pdf

<SDK>/external/rockit/mpi/doc
├── Rockchip_Developer_Guide_MPI_CN.pdf
└── Rockchip_FAQ_MPI_CN.pdf

6.12 Graphics 开发

RK 参考文档：
<SDK>/docs/Linux/Graphics
├── Rockchip_Developer_Guide_Buildroot_Weston_CN.pdf
├── Rockchip_Developer_Guide_Buildroot_Weston_EN.pdf
├── Rockchip_Developer_Guide_Linux_Graphics_CN.pdf
└── Rockchip_Developer_Guide_Linux_Graphics_EN.pdf

6.13 安全机制开发

RK 参考文档：
U-Boot 文档
docs/Common/UBOOT/Rockchip_Developer_Guide_UBoot_Nextdev_CN.pdf

<SDK>/docs/Linux/Security
├── Rockchip_Developer_Guide_Crypto_HWRNG_CN.pdf
├── Rockchip_Developer_Guide_Linux_Secure_Boot_CN.pdf
├── Rockchip_Developer_Guide_Linux_Secure_Boot_EN.pdf
└── Rockchip_Developer_Guide_TEE_SDK_CN.pdf

6.14 A/B 系统开发

RK 参考文档：
U-Boot 文档
docs/Common/UBOOT/Rockchip_Developer_Guide_UBoot_Nextdev_CN.pdf
docs/Common/UBOOT/Rockchip_Developer_Guide_Linux_AB_System_CN.pdf

6.15 系统升级开发

系统升级方案，RK 参考文档：

<SDK>/docs/Linux/Recovery

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