



# CV186AH SDK 编译使用手册

Version: 0.0.0.2

Release date: 2023/12

©2022 北京晶视智能科技有限公司  
本文件所含信息归北京晶视智能科技有限公司所有。  
未经授权，严禁全部或部分复制或披露该等信息。

# 目录

<b>1</b>	<b>声明</b>	<b>2</b>
<b>2</b>	<b>建构 CVITEK 软件编译环境</b>	<b>3</b>
2.1	Linux 服务器	3
2.1.1	于 VirtualBoxVM 安装 Ubuntu	3
2.1.2	Ubuntu 开机设定	5
2.1.3	安装 SSH Server	8
2.1.4	安装 Samba Server	9
2.2	建构编译环境	9
2.3	配置 github 账号	10
2.4	提供两种源码的方式	11
2.5	编译	12
2.5.1	环境变量说明	12
2.5.2	编译整个软件包	12
2.6	修改 uboot	13
2.7	修改 kernel	14
<b>3</b>	<b>烧录说明</b>	<b>16</b>
3.1	使用前准备	16
3.2	操作过程	16
3.3	操作实例	16
3.4	注意事项	16
<b>4</b>	<b>根文件系统 (rootfs)</b>	<b>17</b>
4.1	根文件系统简介	17
4.2	Rootfs	18
4.2.1	Pre-build rootfs 架构	18
4.2.2	编译来自 buildroot 的 rootfs	20
4.2.3	将 rootfs 包装成可烧录映像档	22
4.2.4	Linux kernel 自动加载 rootfs	23
<b>5</b>	<b>使用 NFS 加速开发</b>	<b>25</b>
5.1	Ubuntu Server 端设置说明:	25
5.2	EVB 板端 mount 说明:	25
5.3	注意事项:	26

## 修订记录

Revision	Date	Description
0.0.0.1	2023/11/06	Draft.
0.0.0.2	2023/11/23	Update for CV186AH

# 1 声明

---



## 法律声明

本数据手册包含北京晶视智能科技有限公司（下称“晶视智能”）的保密信息。未经授权，禁止使用或披露本数据手册中包含的信息。如您未经授权披露全部或部分保密信息，导致晶视智能遭受任何损失或损害，您应对因之产生的损失/损害承担责任。

本文件内信息如有更改，恕不另行通知。晶视智能不对使用或依赖本文件所含信息承担任何责任。本数据手册和本文件所含的所有信息均按“原样”提供，无任何明示、暗示、法定或其他形式的保证。晶视智能特别声明未做任何适销性、非侵权性和特定用途适用性的默示保证，亦对本数据手册所使用、包含或提供的任何第三方的软件不提供任何保证；用户同意仅向该第三方寻求与此相关的任何保证索赔。此外，晶视智能亦不对任何其根据用户规格或符合特定标准或公开讨论而制作的可交付成果承担责任。

## 联系我们

**地址** 北京市海淀区丰豪东路 9 号院中关村集成电路设计园（ICPARK）1 号楼

深圳市宝安区福海街道展城社区会展湾云岸广场 T10 栋

**电话** +86-10-57590723 +86-10-57590724

**邮编** 100094（北京）518100（深圳）

**官方网站** <https://www.sophgo.com/>

**技术论坛** <https://developer.sophgo.com/forum/index.html>

# 2 建构 CVITEK 软件编译环境

---

## 2.1 Linux 服务器

开发者可选择使用：

- Ubuntu OS 计算机
- Windows OS 计算机 + Virtualbox VM (上面运行 Ubuntu)

两种方式，都请安装成 Ubuntu 20.04 LTS 版本。

Virtualbox VM 下载网址: <https://www.virtualbox.org/wiki/Downloads>

Ubuntu 20.04 LTS 下载网址: <https://releases.ubuntu.com/20.04/ubuntu-20.04.2.0-desktop-amd64.iso>

### 2.1.1 于 VirtualBoxVM 安装 Ubuntu

- 建立新的 VM，并加以命名




### 名稱和作業系統

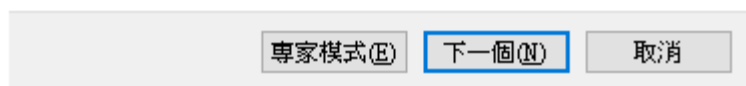
請為新的虛擬機器選擇描述性名稱和目的地資料夾，並選取要在其上安裝的作業系統類型。您選擇的名稱將在整個 VirtualBox 中使用，以標識這部電腦。

名稱:

機器資料夾:

類型(T):  

版本(V):



- 规划 8GB 记忆体供 VM 使用。



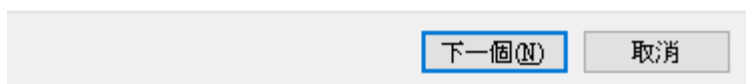
### 記憶體大小

選取配置到虛擬機器的記憶體量 (RAM)，單位 MB。

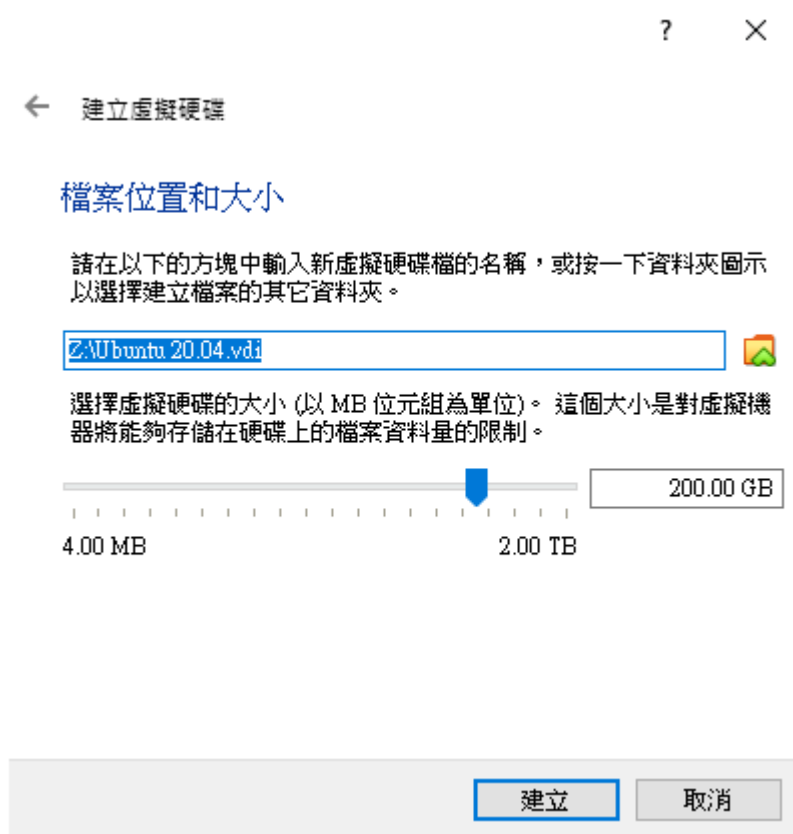
建議的記憶體大小為 **1024MB**。

MB

4 MB 16384 MB

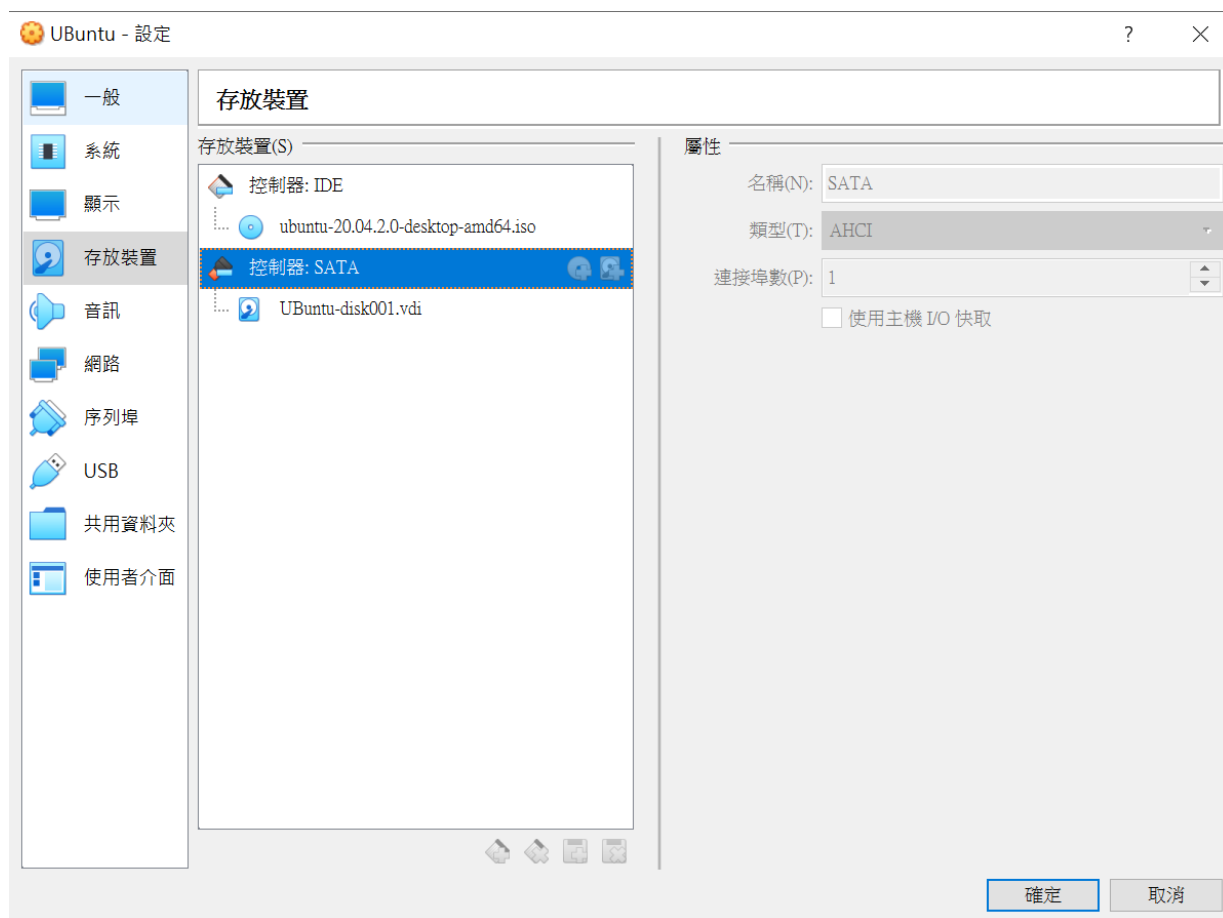


- 预留 200GB 硬盘空间，供后续存放 SDK 用。



## 2.1.2 Ubuntu 开机设定

- 第一次开机需要挂载安装光盘 ISO 档案

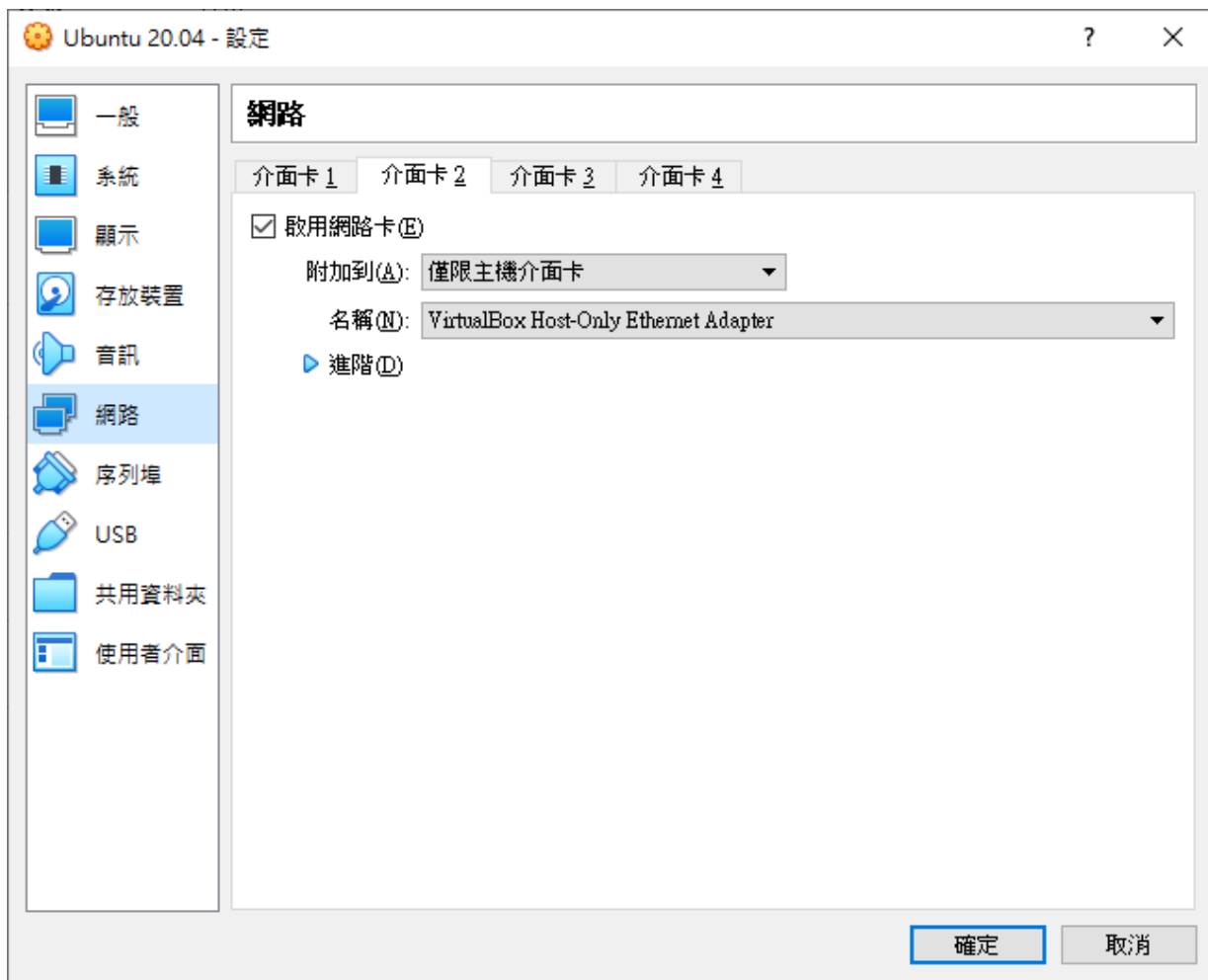


· 开始安装





- 设定 VirtualBox Host-only Ethernet Adapter 以便 Host 与 VirtualBox 沟通（终端服务以及档案分享）



### 2.1.3 安裝 SSH Server

SSH Server 安裝

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

安裝後可以修改一些 ssh 的設定, 如 port, 密碼認證, root 登入等

```
vim /etc/ssh/sshd_config  
  
Port 22  
  
PasswordAuthentication yes  
  
PermitRootLogin yes -> 是否開放 root 登入
```

修改後要重啟 SSH

```
/etc/init.d/ssh restart
```

## 2.1.4 安装 Samba Server

Ubuntu VB 需要安装 Samba 套件，方便后续 Host PC 与其做档案分享。

安装 Samba 前，先用 ifconfig 获取 IP 资讯，第一次安装会发现没有 net-tool 支持，需要安装 net-tool

```
sudo apt install net-tools
sudo apt-get install samba samba-common
```

建立账号的 samba 密码

```
sudo smbpasswd -a cvitek
```

修改/etc/samba/smb.conf，增加以下内容

```
[cvitek]
path = /home/cvitek
writable = yes
browseable = yes
valid users = cvitek
```

启动 samba server

```
sudo service smbd restart
```

WINDOW PC 端连接 Samba server (<Server IP>)

▼ 網路位置 (1)



参考2.2. 安装 CVITEK Build Environment 即可进行编译。

## 2.2 建构编译环境

在编译 SDK 之前，Ubuntu 需要安装以下套件：

```
sudo apt-get install -y build-essential
sudo apt-get install -y ninja-build
sudo apt-get install -y automake
sudo apt-get install -y autoconf
sudo apt-get install -y libtool
sudo apt-get install -y wget
sudo apt-get install -y curl
sudo apt-get install -y git
sudo apt-get install -y gcc
sudo apt-get install -y libssl-dev
sudo apt-get install -y bc
sudo apt-get install -y slib
sudo apt-get install -y squashfs-tools
```

(下页继续)

(续上页)

```
sudo apt-get install -y android-sdk-libsparse-utils
sudo apt-get install -y android-sdk-ext4-utils
sudo apt-get install -y jq
sudo apt-get install -y cmake
sudo apt-get install -y python3-distutils
sudo apt-get install -y tcl
sudo apt-get install -y scons
sudo apt-get install -y parallel
sudo apt-get install -y openssh-client
sudo apt-get install -y tree
sudo apt-get install -y python3-dev
sudo apt-get install -y python3-pip
sudo apt-get install -y ssh
sudo apt-get install -y libncurses5
sudo apt-get install -y pkg-config
sudo apt-get install -y lzop
sudo apt-get install -y bison
sudo apt-get install -y flex
sudo apt-get install -y rsync
sudo apt-get install -y kmod
sudo apt-get install -y cpio
sudo apt-get install -y sudo
sudo apt-get install -y fakerooot
sudo apt-get install -y dpkg-dev
sudo apt-get install -y device-tree-compiler
sudo apt-get install -y u-boot-tools
sudo apt-get install -y uuid-dev
sudo apt-get install -y libxml2-dev
sudo apt-get install -y debootstrap
sudo apt-get install -y qemu
sudo apt-get install -y qemu-user-static
sudo apt-get install -y kpartx
sudo apt-get install -y binfmt-support
sudo apt-get install -y git-lfs
sudo apt-get install -y libisl-dev
sudo apt-get install -y texlive-xetex
sudo apt-get install -y libgflags-dev
```

## 2.3 配置 github 账号

在 github 建立个人账号，并配置好 ssh key，下载代码需要用到个人 github 账号

### 1. 设置账号邮箱

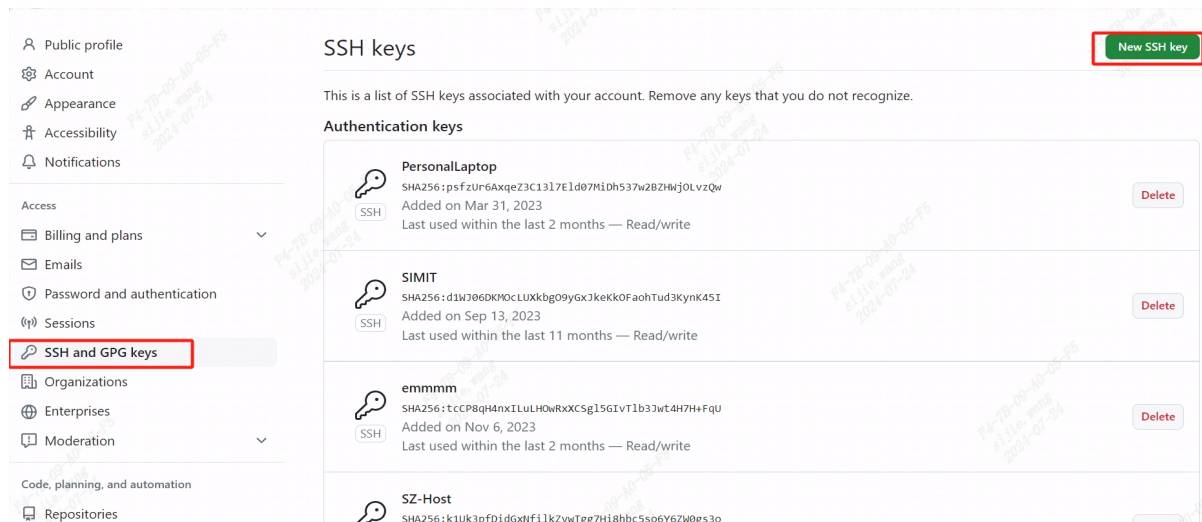
```
git config --global user.name "your_name"
git config --global user.email "your_email@example.com"
```

### 2. 配置密钥

```
ssh-keygen -t ed25519 -C "your_email@example.com"

cat ~/.ssh/id_ed25519.pub
```

### 3. 将公钥添加到 github



### 4. 验证 ssh 是否配置成功

```
ssh -T git@github.com
```

## 2.4 提供两种源码的方式

#### 1. 使用 repo 从 github 拉取最新 SDK 源码

```
repo init -u https://github.com/sophgo/manifest.git -m release/all_repos.xml

repo sync -j4
```

#### 2. 从算能官网获取

从算能官网 (<https://developer.sophgo.com/site/index/material/all/all.html>) 下载 BM1688 & CV186AH 选项中最新端侧 SDK(sophonsdk\_device\_v1.x\_official\_release), 然后执行下面的命令可获取 SDK 最新版本

```
unzip sophonsdk_device_v1.x_offical_release.zip

cd sophonsdk_device_v1.x_offical_release/sdk_release

tar -xvf device_repo_sourcecode_1.8.tar.tgz (注:源码压缩包名字可能稍有调整)

repo sync
```

解压 tar.gz 包, 将看到如下文件:

```

|----build
| |--envsetup_soc.sh ——>编译脚本入口
|----fsbl
|----host-tools
|----isp-tuning
|----linux_5.10 ——> kernel源代码
|----middleware
|----ramdisk
|----u-boot-2021.10 ——> u_boot源代码
|----...

```

## 2.5 编译

### 2.5.1 环境变量说明

编译前置动作最主要是为了设置两个环境变量：\$SIDE\_TYPE, \$BOARD,

\$SIDE\_TYPE 变量是需要根据用户的 SOC 来做设置。

\$BOARD 变数是针对每张 EVB, 有不同的驱动, 必须要正确设置。

### 2.5.2 编译整个软件包

设定环境变量前需要先透过下列命令初始化环境, 系统会列出目前 SDK 支持的端侧以及 EVB 版号。初始化环境:

```
$source build/envsetup_soc.sh
```

Usage:

- (1) menuconfig - Use menu to configure your board.  
ex: \$ menuconfig
- (2) defconfig \$CHIP\_ARCH - List EVB boards(\$BOARD) by CHIP\_ARCH.  
\*\* sophon \*\* -> ['edge', 'device']  
ex: \$ defconfig device
- (3) defconfig \$BOARD - Choose EVB board settings.  
ex: \$ defconfig device\_wevb\_emmc

配置对应的版型, 系统会打印出端侧内建支持的 EVB (\$SIDE\_TYPE\_\$BOARD) 板:

```

$ defconfig
device_fpga          device_palladium      device_palladium_c906 device_wevb_emmc
device_wevb_spinand  device_wevb_spinor    edge_wevb_emmc

```

选取 EVB 版号为 device\_wevb\_emmc, 此时系统会列出自动设定好的环境变数。(后续亦可用 cvi\_print\_env 来打印目前使用的环境变数 > )

```
$ defconfig device_wevb_emmc
===== Environment Variables =====

PROJECT: device_wevb_emmc, DDR_CFG=ddr_auto
CHIP_ARCH: SOPHON, DEBUG=0
SDK_VERSION: 64bit, RPC=0
ATF options: ATF_KEY_SEL=default, BL32=1
Linux source folder:linux_5.10, Uboot source folder: u-boot-2021.10
CROSS_COMPILE_PREFIX: aarch64-none-linux-gnu-
ENABLE_BOOTLOGO: 0
Flash layout xml: /project/sophon/sync_code/a2_release_all_repos/build/boards/sophon/device_
→wevb_emmc/partition/partition_emmc.xml
Sensor tuning bin: gcore_gc4653
Output path: /project/sophon/sync_code/a2_release_all_repos/install/soc_device_wevb_emmc
```

编译 SDK:

```
$ clean_device_all && build_device_all
```

## 2.6 修改 uboot

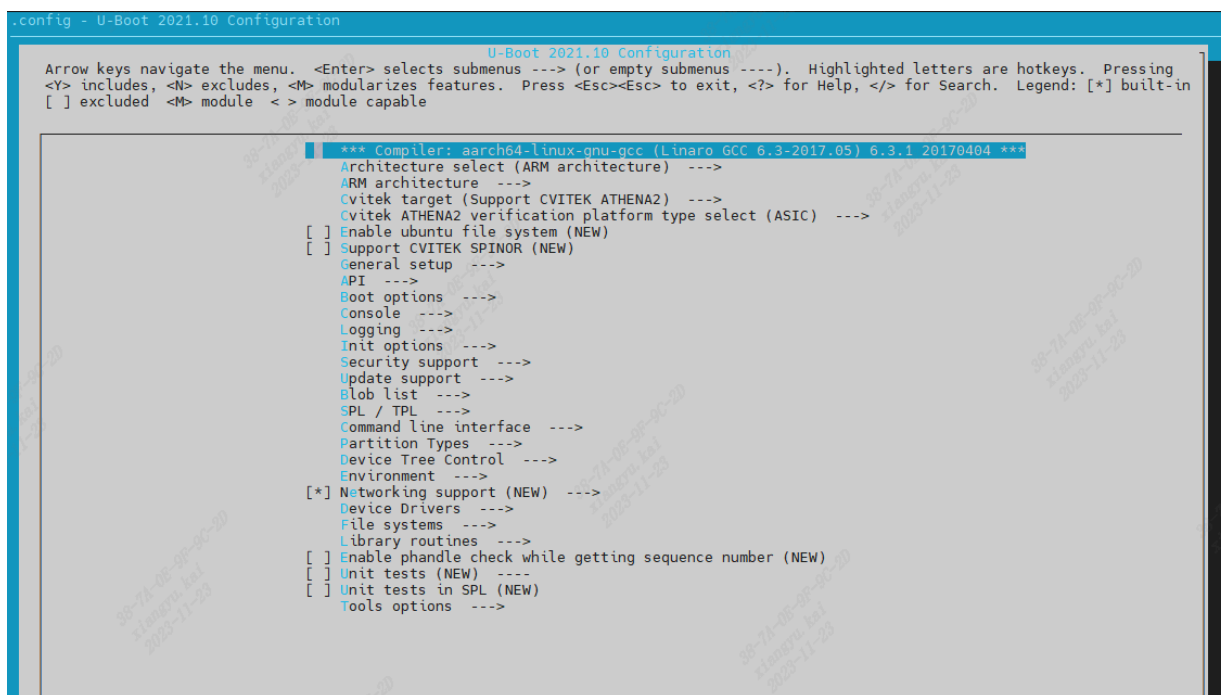
每个 EVB 板会在特定位置定义进入 U-Boot 之前, EVB 需要采取的初始化动作或是定义特定 PINMUX。以 device\_wevb\_emmc 这张板子为例, 会定义在: build/boards/\$CHIP\_\_\$BOARD/u-boot/cvi\_\_board\_\_init.c

```
int cvi_board_init(void)
{
    PINMUX_CONFIG(CAM_MCLK0, CAM_MCLK0, G9);
    PINMUX_CONFIG(CAM_MCLK1, CAM_MCLK1, G9);
    PINMUX_CONFIG(CAM_MCLK2, CAM_MCLK2, G9);
    ...
    PINMUX_CONFIG(GPIO1, GPIO112, G12);
    PINMUX_CONFIG(GPIO0, GPIO111, G12);
    PINMUX_CONFIG(GPIO3, GPIO114, G12);
    return 0;
}
```

其对应的 u-boot 组态, 定义在: ./build/boards/\$CHIP\_\_\$BOARD/u-boot/\$CHIP\_\_\$BOARD\_\_defconfig

```
CONFIG_ARM=y
CONFIG_SYS_MALLOC_F_LEN=0x2000
CONFIG_NR_DRAM_BANKS=1
CONFIG_DEFAULT_DEVICE_TREE="cv186ah_wevb_emmc"
CONFIG_ARMV8_SET_SMPEN=y
CONFIG_DISTRO_DEFAULTS=y
CONFIG_FIT=y
...
```

以图形化接口修改 Uboot Config



make 退出后会把设定储存在: `./u-boot/build/" $CHIP" __" $BOARD" /.config` 执行编译

```
$ build_uboot
```

完成后会生成 `fip.bin`

## 2.7 修改 kernel

修改 kernel , 重新编译 Linux kernel image 每张 EVB 都有对应的 dts 档案来定义其 device tree, 以 `device_wevb_emmc` 为例, 其 DTS 档案定义在: `./build/boards/" $CHIP" __" $BOARD" /dts__arm64/" $CHIP" __" $BOARD" .dts`

```
/dts-v1/;
#include "soph_base_arm64.dtsi"
// #include "soph_asic_bga.dtsi"
#include "soph_asic_emmc.dtsi"
#include "soph_default_memmap.dtsi"
...
```

其相对应的 linux 组态, 定义在: `./build/boards/" $SIDE_TYPE" __" $BOARD" /linux/" $SIDE_TYPE" __" $BOARD" _defconfig`

```
CONFIG_SYSVIPC=y
CONFIG_POSIX_QUEUE=y
CONFIG_NO_HZ_IDLE=y
CONFIG_HIGH_RES_TIMERS=y
CONFIG_TASKSTATS=y
CONFIG_IKCONFIG=y
CONFIG_IKCONFIG_PROC=y
CONFIG_SCHED_AUTOGROUP=y
```

(下页继续)

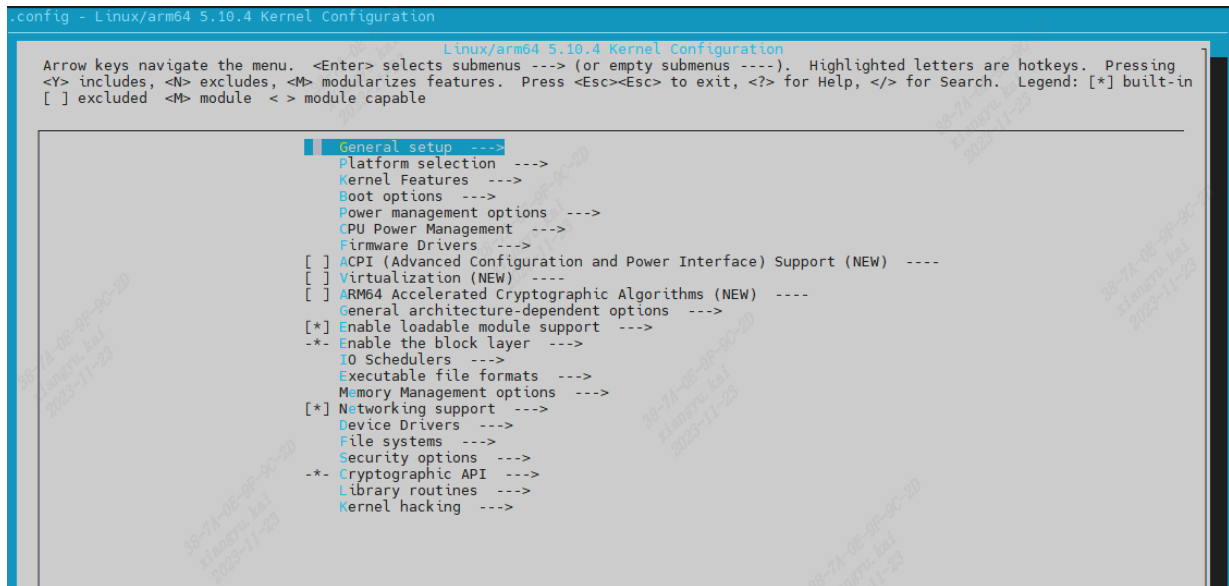


(续上页)

```
CONFIG_BLK_DEV_INITRD=y  
CONFIG_BLK_DEV_RAM=y  
...
```

以图形化接口修改 Kernel Config

```
$ menuconfig_kernel
```



退出后会把设定储存在: `./linux/build/"$CHIP_ARCH"_"$BOARD"/.config`

```
$ build_kernel
```

完成后会生成 `boot.spinor`

# 3 烧录说明

---

## 3.1 使用前准备

- 前述章节产生的烧录档案。
- FAT32 格式的 Micro SD 卡。

## 3.2 操作过程

- 将烧录档案（如下表）放到 SD 卡中。
- 将 SD 卡插入 CVITEK EVB 的 SD 卡槽中。
- 将平台重开机。

## 3.3 操作实例

使用前确认文件，文件路径在 `/install/soc_device_wevb_emmc/` 下，将目录下 `upgrade.zip` 解压出来的内容和 `ramboot.itb` 拷贝到 SD 卡，插入 SD 卡，接上电源后开机，自动启动烧录程序，平台烧录完成时，将平台断电，拔出 SD 卡，再重开机，即完成烧录过程。

## 3.4 注意事项

请确认 SD Card 被正确格式化为 FAT32 格式。

# 4 根文件系统 (rootfs)

## 4.1 根文件系统简介

内核是 Linux 操作系统的核心，文件系统是用户和操作系统沟通的主要工具。所以要使用 Linux 时，要先了解文件系统原理。

根文件系统结构是以“/”为“根 (root)”起始的树状目录结构，当内核程序映像 (uImage) 启动会挂载一个设备 (ex:eMMC) 在根目录上，根文件系统通常存放在内部存储器 (DRAM) 或非挥发内存 (FLASH) 中，或是透过网络存取的文件系统 (NFS)。所有应用程序和函式库都会按照分类放入文件系统中，以下列出根文件系统目录结构图。

```
/ 根目录
├── bin 可执行文件
├── dev 设备文件
├── etc 系统配置文件(ex: 启动文件)
├── home 用户目录
├── init 开机执行script
├── kdump 内核除错目录
├── lib 函式库包含glibc, shared library和内核模块
├── mnt 临时文件系统的挂载点
├── proc 内核和行程信息的虚拟文件系统
├── sbin 系统管理的可执行文件
├── sys 系统设备和文件层次结构，提供内核数据数据
├── usr 此目录下包含用户自定义应用程序和文文件
└── var 存放系统日志和服务程序文件
```

## 4.2 Rootfs

本章节是描述文件系统之组成方式，详细路径于 `ramdisk/rootfs/`

### 4.2.1 Pre-build rootfs 架构

文件系统之结构目录主要拆了三个类型，且逐层迭加于 Rootfs，将于下方分别描述：

- **Basic rootfs:**

现阶段本公司提供了基于以下四种 Arch 产生之 pre-build rootfs 档案

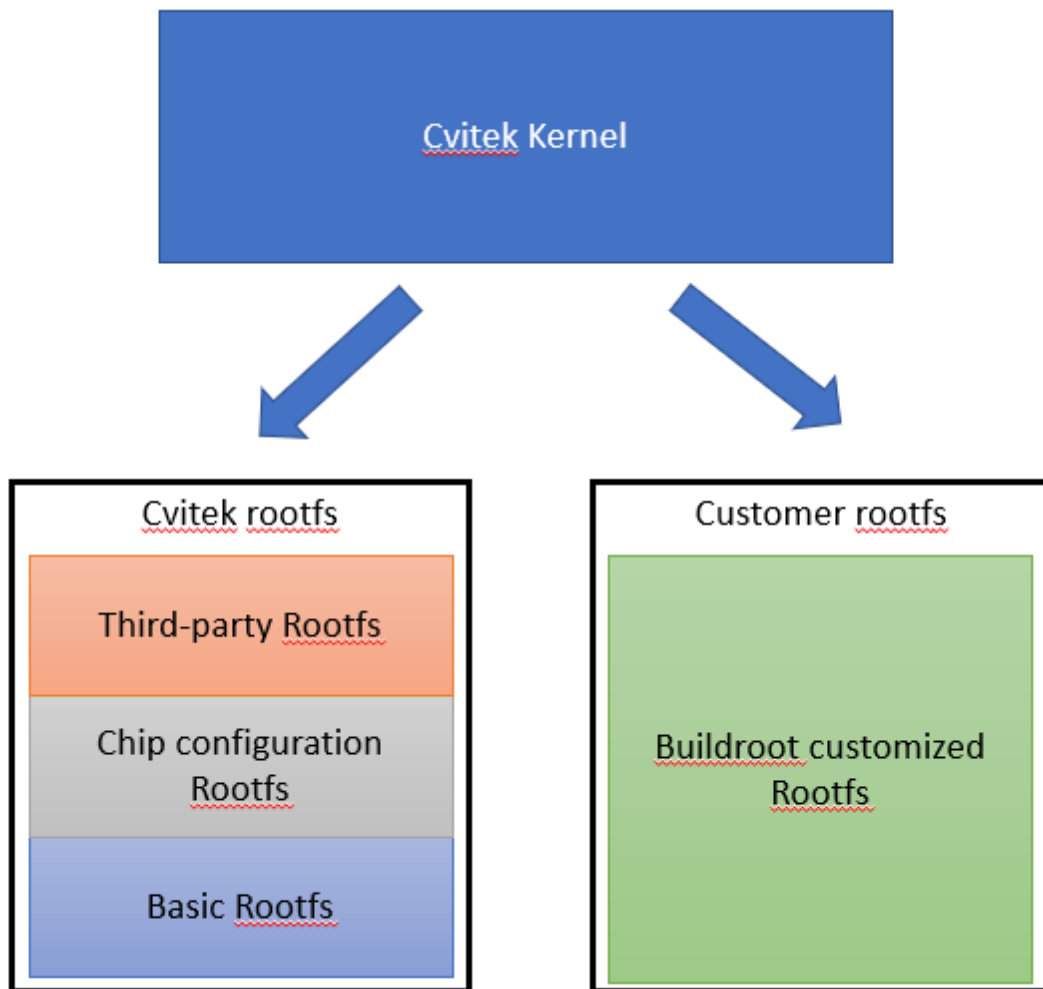
Arch	Libc	Pre-build ramdisk path
Arm	glibc	<code>ramdisk/rootfs/common_arm/</code>
Arm	uclibc	<code>ramdisk/rootfs/common_uclibc/</code>
Aarch64	glibc	<code>ramdisk/rootfs/common_arm64/</code>
Riscv64	glibc	<code>ramdisk/rootfs/common_riscv64/</code>
Riscv64	musl	<code>ramdisk/rootfs/common_musl64/</code>

- **Chip configuration rootfs:**

本公司将所有 Chipset 相依之开机设置均放置于 `ramdisk/rootfs/overlay/$CHIP`

- **Third-party rootfs:**

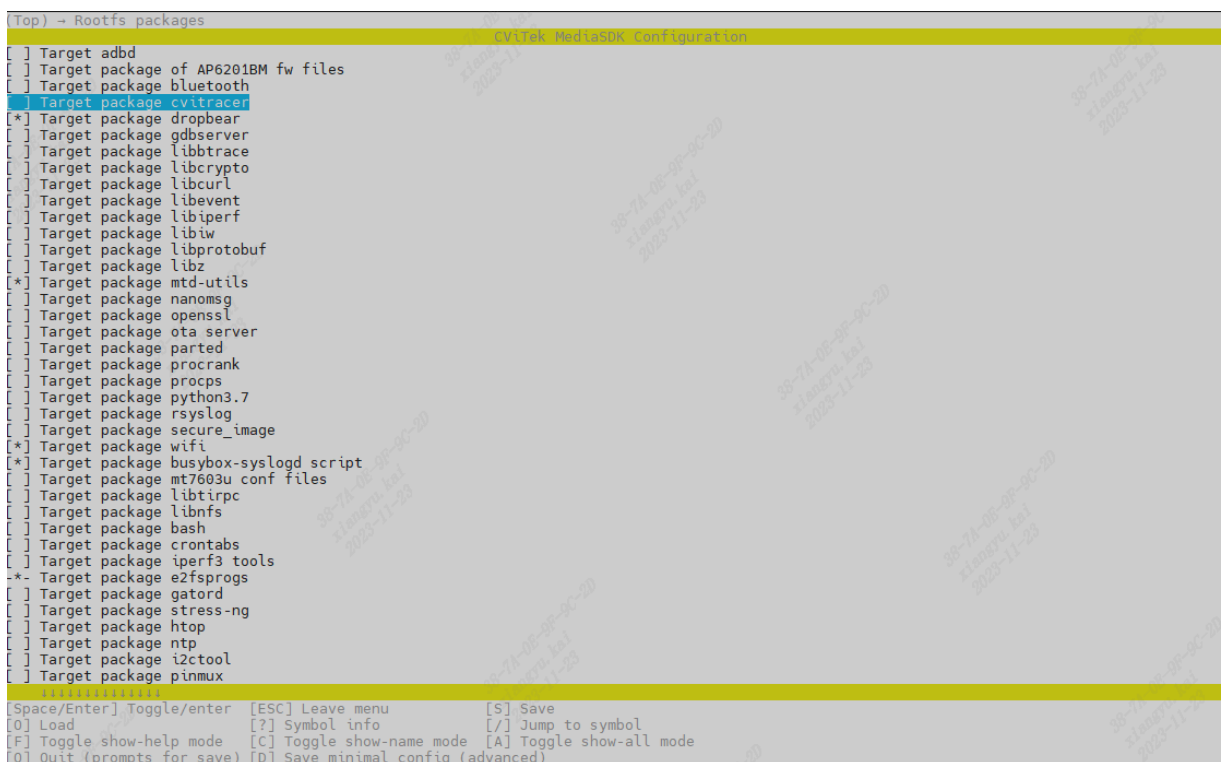
本公司将所有第三方软件编译出来之 library、utility、related file 均放置于 `ramdisk/rootfs/public/`



可以透过选单的方式决定需要那些 Third-party software 要放置进 Rootfs

\$ menuconfig

```
Top)
CVitek MediaSDK Configuration
generic) Customer define
  Chip selection (cv186ah) --->
  Board selection (wevb_emmc (CA53 + EMMC 512MB + BGA SIP 8GB)) --->
  DDR configuration selection (ddr4_3200_x16_2s) --->
arm64) Arch define
  Compile-time checks and compiler options --->
  SDK options --->
  FIP setting --->
  Storage settings --->
  Sensor settings --->
  Panel settings --->
  uboot options --->
  Kernel options --->
  ROOTFS options --->
  Turnkey options --->
  RTOS options --->
  Rootfs packages --->
```



## 4.2.2 编译来自 buildroot 的 rootfs

此章节是示范如何从 buildroot 产生 rootfs 并且于 EVB 上面运行的例子，若采用上一章节描述的 pre-build rootfs，可忽略此章节。

1. 获取 buildroot 原始码 (本章节使用 buildroot-2021.02.9.tar.gz 为示范)

<https://buildroot.org/download.html>

2. 解压缩原始码

```
$ tar xzf buildroot-2021.02.9.tar.gz
```

3. 设置 Arch Info & Toolchain

```
$ make menuconfig
```

	Buildroot 2021.11-1153-gd7cf67fc94 Configuration	
	Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty submenus ----). Highlighted letters are hotkeys. Pressing <Y> selects a feature, while <N> excludes a feature. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*] feature is selected	
	Target options --->	
	Build options --->	
	Toolchain --->	
	System configuration --->	
	Kernel --->	
	Target packages --->	
	Filesystem images --->	
	Bootloaders --->	
	Host utilities --->	
	Legacy config options --->	
	<Select> < Exit > < Help > < Save > < Load >	

### 设定架构

	Target options	
	Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty submenus ----). Highlighted letters are hotkeys. Pressing <Y> selects a feature, while <N> excludes a feature. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*] feature is selected	
	Target Architecture (AArch64 (little endian)) --->	
	Target Binary Format (ELF) --->	
	Target Architecture Variant (cortex-A53) --->	
	Floating point strategy (FP-ARMv8) --->	
	<Select> < Exit > < Help > < Save > < Load >	

### 设定 Toolchain

	Toolchain	
	Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty submenus ----). Highlighted letters are hotkeys. Pressing <Y> selects a feature, while <N> excludes a feature. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*] feature is selected	
	Toolchain type (External toolchain) --->	
	*** Toolchain External Options ***	
	Toolchain origin (Pre-installed toolchain) --->	
	(/home/cvitek/host-tools/gcc/gcc-linaro-6.3.1-2017.05-x86_64_arm- (arm-linux-gnueabihf) Toolchain prefix	
	External toolchain gcc version (6.x) --->	
	External toolchain kernel headers series (4.6.x) --->	
	External toolchain C library (uClibc/uClibc-ng) --->	
	[ ] Toolchain has WCHAR support?	
	<Select> < Exit > < Help > < Save > < Load >	

### 4. 编译 buildroot 软体包

```
$ make
```

5. 得到 rootfs , 其位置在 output/images/rootfs.tar
6. 修改 build/Makefile 让 SDK 使用 buildroot 产生的 rootfs

```

buildroot-prepare:
$(call print_target)
#clean_all
rm -rf $(ROOTFS_DIR)
#extract buildroot roofs
tar xf $(BUILROOT_DIR)/output/images/rootfs.tar -C $(ROOTFS_DIR)
rootfs-pack:export CROSS_COMPILE_KERNEL=$(patsubst "%",%,$(CONFIG_CROSS_
→COMPILE_KERNEL))
rootfs-pack:export CROSS_COMPILE_SDK=$(patsubst "%",%,$(CONFIG_CROSS_COMPILE_
→SDK))
rootfs-pack:$(OUTPUT_DIR)/rawimages
#rootfs-pack:rootfs-prepare
rootfs-pack:buildroot-prepare
rootfs-pack:
$(call print_target)
${Q}printf '\033[1;36;40m Striping rootfs \033[0m\n'
ifeq (${FLASH_SIZE_SHRINK},y)
${Q}printf 'remove unneeded files'
${Q} $(COMMON_TOOLS_PATH)/spinand_tool/clean_rootfs.sh $(ROOTFS_DIR)
endif
${Q}find $(ROOTFS_DIR) -name "*.ko" -type f -printf 'striping %p\n' -exec $(CROSS_COMPILE_
→KERNEL)strip --strip-unneeded {} \;
${Q}find $(ROOTFS_DIR) -name "*.so*" -type f -printf 'striping %p\n' -exec $(CROSS_
→COMPILE_KERNEL)strip --strip-all {} \;
${Q}find $(ROOTFS_DIR) -executable -type f ! -name "*.sh" ! -path "*etc*" ! -path "*.ko" -printf
→'striping %p\n' -exec $(CROSS_COMPILE_SDK)strip --strip-all {} 2>/dev/null \;

```

7. 产生新的 ROOTFS 可烧录映像档

```
$ pack_rootfs
```

8. 透过第二章所提的步骤烧录到板端

### 4.2.3 将 rootfs 包装成可烧录映像档

将前述步骤产生之 rootfs folder 透过 mksquashfs 工具做最终打包, 压缩方式为 XZ, 最终产物即是可刻录于 Flash 上的 rootfs.spinor / rootfs.spinand / rootfs.emmc。

详细参考 build/Makefile 下之 rootfs-pack:

```

rootfs-pack:export CROSS_COMPILE_KERNEL=$(patsubst "%",%,$(CONFIG_CROSS_
→COMPILE_KERNEL))
rootfs-pack:export CROSS_COMPILE_SDK=$(patsubst "%",%,$(CONFIG_CROSS_COMPILE_
→SDK))
rootfs-pack:$(OUTPUT_DIR)/rawimages
rootfs-pack:rootfs-prepare
rootfs-pack:
$(call print_target)
${Q}printf '\033[1;36;40m Striping rootfs \033[0m\n'
ifeq (${FLASH_SIZE_SHRINK},y)

```

(下页继续)



(续上页)

```

${Q}printf 'remove unneeded files'
${Q} $(COMMON_TOOLS_PATH)/spinand_tool/clean_rootfs.sh $(ROOTFS_DIR)
endif
${Q}find $(ROOTFS_DIR) -name "*.ko" -type f -printf 'striping %p\n' -exec $(CROSS_COMPILE_
→KERNEL)strip --strip-unneeded {} \;
${Q}find $(ROOTFS_DIR) -name "*.so*" -type f -printf 'striping %p\n' -exec $(CROSS_
→COMPILE_KERNEL)strip --strip-all {} \;
${Q}find $(ROOTFS_DIR) -executable -type f ! -name "*.sh" ! -path "*etc*" ! -path "*.ko" -printf
→'striping %p\n' -exec $(CROSS_COMPILE_SDK)strip --strip-all {} 2>/dev/null \;
ifeq ($(STORAGE_TYPE),spinor)
${Q}mksquashfs $(ROOTFS_DIR) $(OUTPUT_DIR)/rawimages/rootfs.sqsh -root-owned -comp xz
else
${Q}mksquashfs $(ROOTFS_DIR) $(OUTPUT_DIR)/rawimages/rootfs.sqsh -root-owned -comp xz -
→e mnt/cfg/*
endif
ifeq ($(STORAGE_TYPE),spinand)
${Q}python3 $(COMMON_TOOLS_PATH)/spinand_tool/mkubiimg.py --ubionly $(FLASH
→PARTITION_XML) ROOTFS $(OUTPUT_DIR)/rawimages/rootfs.sqsh $(OUTPUT_DIR)/
→rawimages/rootfs.spinand -b $(CONFIG_NANDFLASH_BLOCKSIZE) -p $(CONFIG_
→NANDFLASH_PAGESIZE)
${Q}rm $(OUTPUT_DIR)/rawimages/rootfs.sqsh
else
${Q}mv $(OUTPUT_DIR)/rawimages/rootfs.sqsh $(OUTPUT_DIR)/rawimages/rootfs.
→$(STORAGE_TYPE)
endif

```

## 4.2.4 Linux kernel 自动加载 rootfs

Linux kernel 会根据 uboot 设定之 bootargs 内的 root= 变量决定 rootfs 位于哪个 device

'root=...'

This argument tells the kernel what device is to be used as the root filesystem while booting. The default of this setting is determined at compile time, and usually is the value of the root device of the system that the kernel was built on. To override this value, and select the second floppy drive as the root device, one would use 'root=/dev/fd1'.

The root device can be specified symbolically or numerically. A symbolic specification has the form /dev/XXYN, where XX designates the device type (e.g., 'hd' for ST-506 compatible hard disk, with Y in 'a'-'d'; 'sd' for SCSI compatible disk, with Y in 'a'-'e'), Y the driver letter or number, and N the number (in decimal) of the partition on this device.

Note that this has nothing to do with the designation of these devices on your filesystem. The '/dev/' part is purely conventional.

The more awkward and less portable numeric specification

(下页继续)

(续上页)

of the above possible root devices in major/minor format is also accepted. (For example, /dev/sda3 is major 8, minor 3, so you could use 'root=0x803' as an alternative.)

Ref: <https://man7.org/linux/man-pages/man7/bootparam.7.html>

# 5 使用 NFS 加速开发

## 5.1 Ubuntu Server 端设置说明:

安装 nfs-kernel-server

```
sudo apt-get install nfs-kernel-server
```

建立 mount 文件夹

```
例: mkdir /home/nfs_server
```

修改/etc/exports 文件, 添加如下内容

```
/home/nfs_server *(rw,sync,no_subtree_check,no_root_squash)
```

restart nfs 服务

```
/etc/init.d/rpcbind restart  
/etc/init.d/nfs-kernel-server restart
```

## 5.2 EVB 板端 mount 说明:

在/mnt/data 文件系统内建立 mount point

```
mkdir /mnt/data/nfs
```

mount nfs

```
mount -t nfs -o nolock 192.168.1.103:/home/nfs_server /mnt/data/nfs/
```

## 5.3 注意事项：

PC 和板子连接在同一局域网。