CV180X & CV181X SDK 编译使用手册

Version: 1.0

Release date: 2022-10-28

目录

修订记录

Revision	Date	Description
0.0.0.1	2020/03/08	Initial.
0.0.0.2	2022/01/28	Update Document Chapter
0.0.0.3	2022/02/17	Update root file system related information
1.0	2022/10/28	Revise for CV180X/CV181X processor.

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)

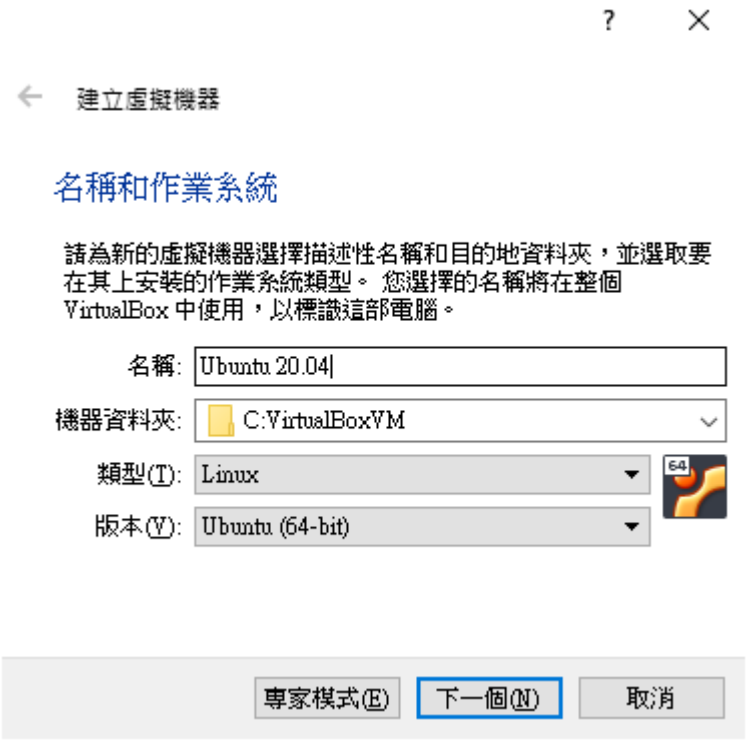
两种方式，都请安装成 Ubuntu 20.04 LTS 版本。

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

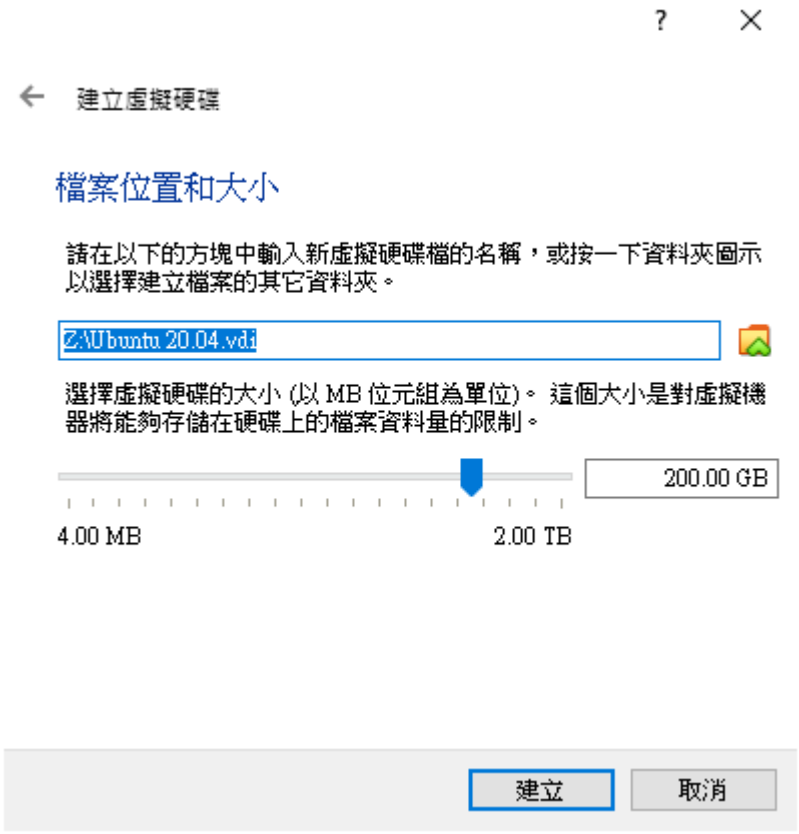
2.1.1 于 VirtualBoxVM 安装 Ubuntu

- 建立新的 VM，并加以命名



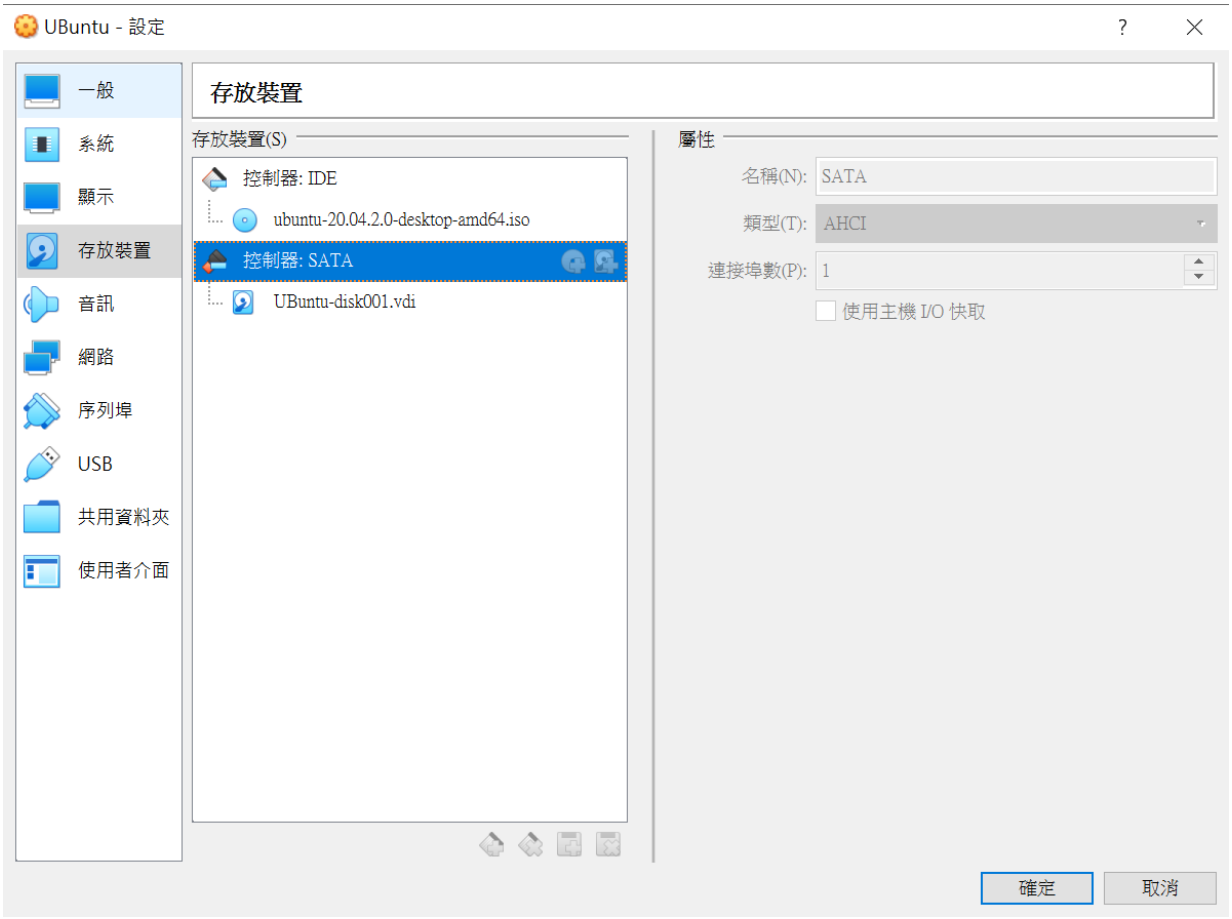
· 规划 8GB 记忆体供 VM 使用。

· 预留 200GB 硬盘空间，供后续存放 SDK 用。



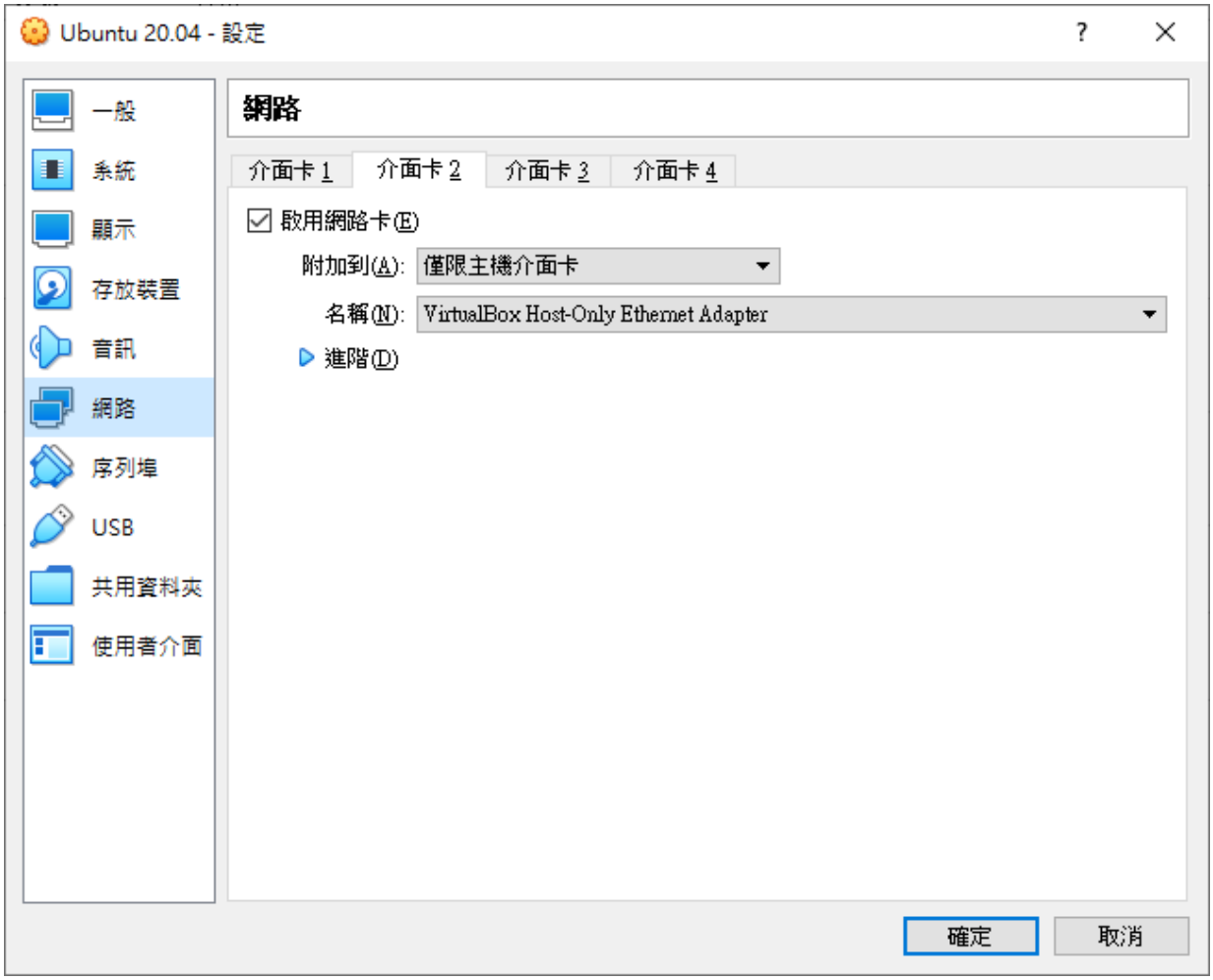
2.1.2 Ubuntu 开机设定

· 第一次开机需要挂载安装光盘 ISO 挡案


UBuntu - 設定
一般
系統
顯示
存放裝置
音訊
網路
序列埠
USB
共用資料夾
使用者介面
存放裝置
存放裝置(S)
控制器: IDE
ubuntu-20.04.2.0-desktop-amd64.iso
控制器: SATA
UBuntu-disk001.vdi
屬性
名稱(N): SATA
類型(T): AHCI
連接埠數(P): 1
使用主機 I/O 快取
確定
取消

· 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 安装 SSH Server

SSH Server 安装

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

安装后可以修改一些 ssh 的设定, 如 port, 密码认证, root 登入等

```
vim /etc/ssh/sshd_config

Port 22

PasswordAuthentication yes

PermitRootLogin yes -> 是否开放 root 登入
```

修改后要重启 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
```

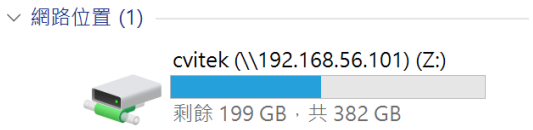
修改/etc/samba/smb.conf，增加以下的内容

```
[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>）



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 update
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
```

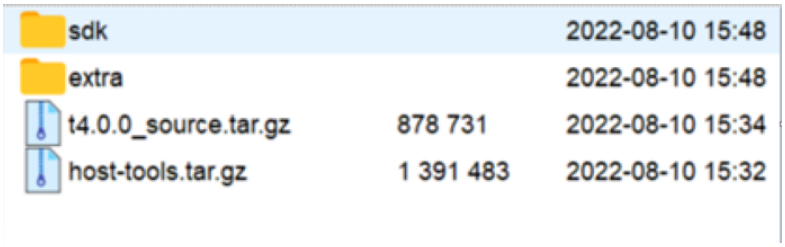
（下页继续）

(续上页)

```
sudo apt-get install -y squashfs-tools
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 tclsh
sudo apt-get install -y scons
sudo apt-get install -y parallel
sudo apt-get install -y ssh-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 device-tree-compiler
sudo apt-get install -y libssl-dev
sudo apt-get install -y ssh
sudo apt-get install -y cpio
sudo apt-get install -y squashfs-tools
sudo apt-get install -y fakeroot
sudo apt-get install -y libncurses5
sudo apt-get install -y flex
sudo apt-get install -y bison
```

2.3 下载 SDK

通常 SDK 会放置于客户专用 FTP，/home/SDK/cv18xx_t(SDK_Version) 底下，此处以 SDK V 4.0.0 为范例。



抓下 t4.0.0_source.tar.gz 后解开

```
$ tar zxvf t4.0.0 source.tar.gz
```

抓下 host-tool.tar.gz 后于 SDK 工作目录解开

```
$ cd cv180x_t4.0.0_source
$ tar zxvf host-tools.tar.gz
```

2.4 编译

2.4.1 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：$CHIP, $BOARD,

$CHIP 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。

例如：

$BOARD=wevb_0008a_spinor: 是 SPINOR + DDR2 1333 64 MB 硬件组合

$BOARD=wevb_0009a_spinand: 是 SPINAND + DDR3 1866 128MB 硬件组合

注：wevb_0008a / wevb_0009a 可以直接看 EVB 上雷射型号得知。

2.4.2 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 IC 以及 EVB 版号.

```
$ source build/cvisetup.sh
------------------------------------------------------------------
 Usage:
 (1) menuconfig - Use menu to configure your board.
     ex: $ menuconfig
 (2) defconfig $CHIP_ARCH - List EVB boards($BOARD) by CHIP_ARCH.
     ** cv183x ** -> ['cv1829', 'cv1832', 'cv1835', 'cv1838', 'cv9520', 'cv7581']
    ** cv182x ** -> ['cv1820', 'cv1821', 'cv1822', 'cv1823', 'cv1825', 'cv1826', 'cv7327', 'cv7357']
    ** cv181x ** -> ['cv181x', 'cv1823a', 'cv1821a', 'cv1820a', 'cv1811h', 'cv1811c', 'cv1810c', 'cv1812h']
    ** cv180x ** -> ['cv180x', 'cv1800b', 'cv1800c', 'cv1801b', 'cv1801c']
     ex: $ defconfig cv183x    (3) defconfig $BOARD - Choose EVB board settings.
     ex: $ defconfig cv1835_wevb_0002a
     ex: $ defconfig cv1826_wevb_0005a_spinand
     ex: $ defconfig cv180x_fpga_c906
------------------------------------------------------------------
```

初始化之后，可以下列两种方式进行编译组态设定

2.4.2.1 透过 defconfig 设定

选取 IC：以 cv180x 为例，系统会打印出 cv180x 内建支持的 EVB（$CHIP_$BOARD）板。

```
$ defconfig cv180x
* cv180x * the avaliable cvitek EVB boards
  cv180x - cv180x_fpga [FPGA]
           cv180x_palladium [PALLADIUM]
  cv1800b - cv1800b_wdmb_0008a_spinor [C906B + SPINOR 8MB + QFN SIP 64MB]
            cv1800b_wevb_0008a_spinor [C906B + SPINOR 16MB + QFN SIP 64MB]
```

（下页继续）

(续上页)

```
cv1800c - cv1800c_wevb_0009a_spinor [C906B + SPINOR 16MB + QFN SIP 64MB]
cv1801b - cv1801b_wevb_0008a_spinor [C906B + SPINOR 16MB + QFN SIP 128MB]
cv1801c - cv1801c_wdmb_0009a_spinor [C906B + SPINOR 16MB + QFN SIP 128MB]
        cv1801c_wevb_0009a_spinand [C906B + SPINAND 256MB + QFN SIP 128MB]
        cv1801c_wevb_0009a_spinor [C906B + SPINOR 16MB + QFN SIP 128MB]
        cv1812h_wevb_0007a_spinor [C906B + SPINOR 16MB + BGA SIP 256MB]
```

选取 EVB 版号为 cv1801c_wevb_0009a_spinor，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 cv1801c_wevb_0009a_spinor

======= Environment Variables =======

PROJECT: cv1801c_wevb_0009a_spinor, DDR_CFG=ddr3_1866_x16
CHIP_ARCH: CV180X, DEBUG=0
SDK VERSION: musl_riscv64, 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: riscv64-unknown-linux-musl-
ENABLE_BOOTLOGO: 0
Flash layout xml: build/boards/cv180x/cv1801c_wevb_0009a_spinor/part
ition/partition_spinor.xml
Sensor tuning bin: gcore_gc4653
Output path: install/soc_cv1801c_wevb_0009a_spinor
```

2.4.2.2 透过 Menuconfig 设定

初始化之后，键入 menuconfig 进入以下页面，即可选择各种 SDK 内部设定，包含 CHIP，EVB 板号等等。

```
(Top)
                    CViTek MediaSDK Configuration
(generic) Customer define
    Chip selection (cv1801c)  --->
    Board selection (wevb_0009a_spinor (C906B + SPINOR 16MB + QFN SIP 64MB))--->
    DDR configuration selection (ddr3_1866_x16)  --->
(riscv) Arch define
    Compile-time checks and compiler options  --->
    SDK options  --->
```

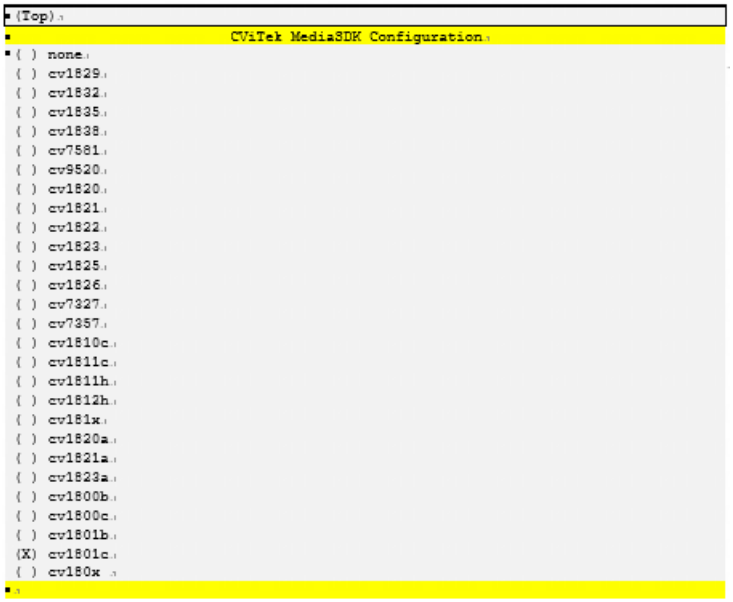
```
(Top)
    FIP setting  --->
    Storage settings  --->
    Sensor settings  --->
    Panel settings  --->
    uboot options  --->
    Kernel options  --->
    ROOTFS options  --->
    Turnkey options  --->
    RTOS options  --->
    Rootfs packages  --->

[Space/Enter] Toggle/enter  [ESC] Leave menu           [S] Save
[O] Load                    [?] Symbol info           [/] Jump to symbol
[F] Toggle show-help mode   [C] Toggle show-name mode  [A] Toggle show-all mode
[Q] Quit (prompts for save) [D] Save minimal config (advanced)
```

配置过程可以透过 [Enter] [Space] [ESC] 等进行设置/返回的动作

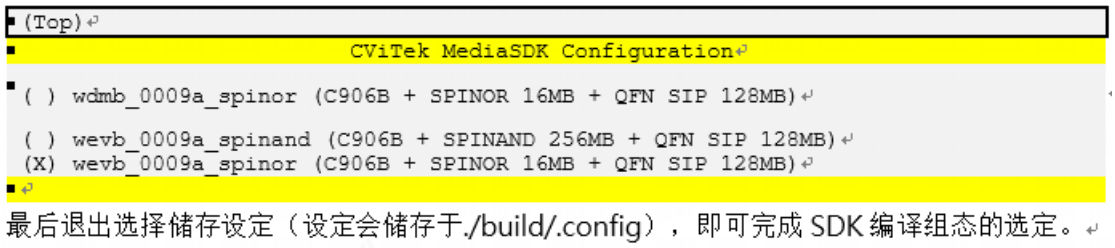
配置完毕后，按下 [S] 储存配置文件，接着按下 [Q] 离开图形化接口

(或者按下 ESC, 会自动弹出是否需要储存的图形)

选取 IC （以 cv1801c 为例）



EVB 版号会列出相对应的选择，选取 EVB 的同时也会决定编译出的 image 适配的 DDR 以及 Flash Size。（以这个例子选取的 EVB 上所带的 DDR 为 DDR3，Flash 为 16MB 的 SPINOR Flash）



最后退出选择储存设定（设定会储存于./build/.config），即可完成 SDK 编译组态的选定。

2.4.3 编译完整 SDK 文档

执行编译，会得到可用于烧录的 images。

```
cvitek@cvitek-VirtualBox:~/working_dir$ build_all
. Run build_uboot () function
...
/work/install/cv1801c_wevb_0009a_spinor/upgrade.zip done!
```

编译出的挡案会放置于./install/soc_<EVB Name>/ 之下 fip.bin

2.4.4 编译部份 SDK 文档

2.4.4.1 单独编译 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 cv1801c_wevb_0009a_spinor 这张板子为例，会定义在：

build/boards/mars/$CHIP_$BOARD/u-boot/cvi_board_init.c

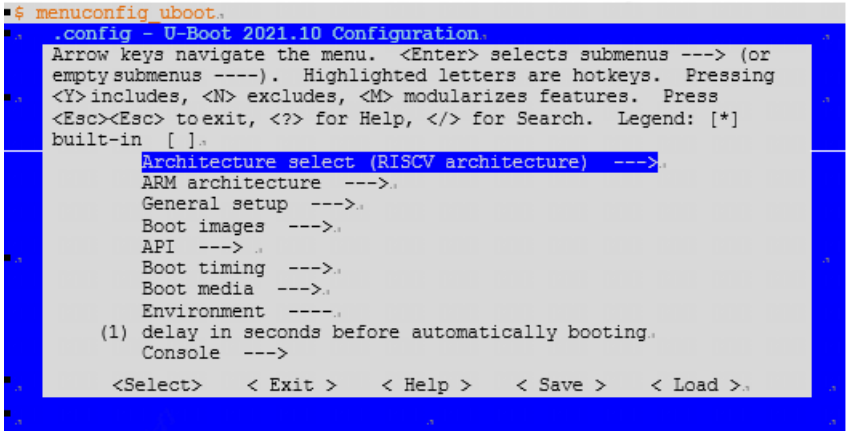
```
int cvi_board_init(void)
{
#if defined(CV180X_QFN_88_PIN)
    PINMUX_CONFIG(PAD_MIPI_TXP1, IIC2_SCL);
    PINMUX_CONFIG(PAD_MIPI_TXM1,IIC2_SDA);
    PINMUX_CONFIG(PAD_MIPI_TXP0, XGPIOC_13);
    PINMUX_CONFIG(PAD_MIPI_TXM0, CAM_MCLK1);
#elif defined(CV180X_QFN_88_PIN_38)
    return 0;
}
```

其对应的 u-boot 组态，定义在：

./build/boards/mars/$CHIP_$BOARD/u-boot/$CHIP_$BOARD_defconfig

```
Partial cvitek_cv1801c_wevb_0009a_spinor_defconfig
CONFIG_RISCV=y
CONFIG_SYS_MALLOC_F_LEN=0x2000
CONFIG_NR_DRAM_BANKS=1
CONFIG_DEFAULT_DEVICE_TREE="cv180x_asic"
CONFIG_IDENT_STRING=" cvitek_cv180x"
...
```

以图形化接口修改 Uboot Config



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

Makefile 中编译 U-Boot 的片段。

```
u-boot-build: ${UBOOT_PATH}/${UBOOT_OUTPUT_FOLDER} ${UBOOT_CVIPART_DEP}
${UBOOT_OUTPUT_CONFIG_PATH}
  $(call print_target)
  ${Q}rm -f ${UBOOT_CVI_BOARD_INIT_PATH}
  ${Q}ln -s ${BUILD_PATH}/boards/${PROJECT_FULLNAME}/u-boot/cvi_board_init.c
${UBOOT_CVI_BOARD_INIT_PATH}
  ${Q}$(MAKE) -j${NPROC} -C ${UBOOT_PATH} olddefconfig
  ${Q}$(MAKE) -j${NPROC} -C ${UBOOT_PATH} all
  $(call uboot_compress_action)
```

2.4.4.2 单独编译 kernel

修改 kernel (ex: *.dts, 内核), 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 cv1801c_wevb_0009a_spinor 为例，其 DTS 档案定义在:

./build/boards/cv180x/"$CHIP"_"$BOARD"/dts_riscv/"$CHIP"_"$BOARD".dts

```
#/dts-v1/;
#include "cv180x_base_riscv.dtsi"
#include "cv180x_asic_qfn.dtsi"
#include "cv180x_asic_spinor.dtsi"
#include "cv180x_default_memmap.dtsi"
/ {
};
```

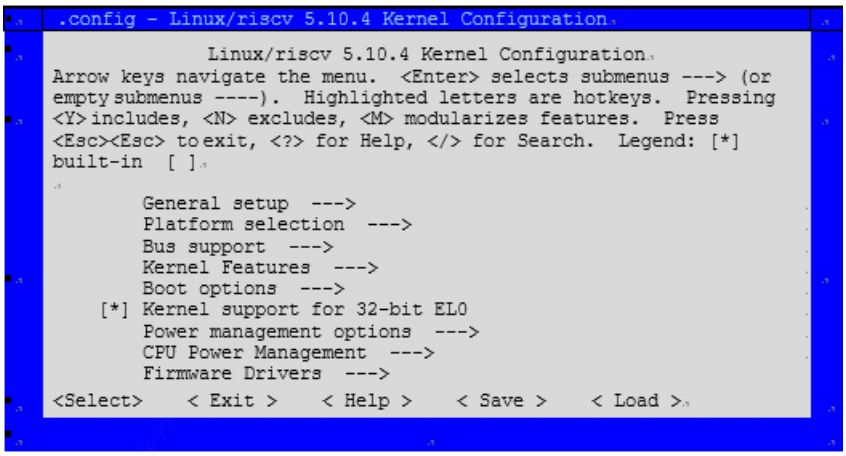
其相对应的 linux 组态，定义在:

./build/boards/"$CHIP"_"$BOARD"/linux/"$CHIP"_"$BOARD"_defconfig

```
Partial cv1801c_wevb_0009a_spinor
CONFIG_SYSVIPC=y
CONFIG_POSIX_MQUEUE=y
CONFIG_NO_HZ_IDLE=y
CONFIG_HIGH_RES_TIMERS=y
CONFIG_PREEMPT=y
CONFIG_IKCONFIG=y
CONFIG_IKCONFIG_PROC=y
CONFIG_LOG_BUF_SHIFT=15
CONFIG_BLK_DEV_INITRD=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/"$CHIP"_"$BOARD"/.config

```
$ build_kernel
```

完成后会生成 boot.spinor

Makefile 中编译 Kernel 的片段。

```
kernel-build: ${KERNEL_OUTPUT_CONFIG_PATH}
  $(call print_target)
  ${Q}echo LOCALVERSION=${LOCALVERSION}
  ${Q}$(MAKE) -j${NPROC} -C ${KERNEL_PATH}O=${KERNEL_PATH}/${KERNEL_
↪OUTPUT_FOLDER}
olddefconfig
  ${Q}$(MAKE) -j${NPROC} -C ${KERNEL_PATH}/${KERNEL_OUTPUT_FOLDER}Image␣
↪modules
  ${Q}$(MAKE) -j${NPROC} -C ${KERNEL_PATH}/${KERNEL_OUTPUT_FOLDER}modules_
↪install
headers_installINSTALL_HDR_PATH=${KERNEL_PATH}/${KERNEL_OUTPUT_FOLDER}/
↪$(ARCH)/usr
  ${Q} ln -sf ${KERNEL_PATH}/${KERNEL_OUTPUT_FOLDER}/$(ARCH)/usr/include
${KERNEL_PATH}/${KERNEL_OUTPUT_FOLDER}/usr/include
```

2.4.4.3 单独编译 middleware

修改 middleware (cvi_test / sample_dsi), 重新编译 middleware 及 system

生成的 Install/PROJECT_NAME/system.* 包含最新的 middleware

```
$ build_middleware; pack_rootfs
```

编译 Middleare 的 shell script 片段

```
pushd "$MW_PATH"/component/isp
make all
popd

pushd "$MW_PATH"/sample
make all
```

build_middleware 会针对 Sensor driver（位于 middleware/component/isp/ 下）以及 sample application（位于 middleware/sample/下）重新编译，末了 pack_rootfs 会将变更后的 driver 以及 application 包装成可烧录映像档。

2.5 区间划分

Mars SDK 会生成以下的 image file，每个挡案代表不同的分区，分列如下：

- FIP：Bootloader/U-Boot 分区
- CV180X/ CV181X C906 采用 FSBL+OPENSBI+UBOOT 架构，最后打包后也复用 (FIP) 檔名，方便后续使用。
- 2nd(双系统)：Yun on Processor (YOC) 所在分区
- BOOT：Partition for Linux Kernel 分区
- MISC：Boot Logo 分区
- ROOTFS：root file system 分区
- SYSTEM: CVITEK libraries 所在分区
- DATA：使用这资料分区

注：2nd 分区为双系统特有分区，仅在双系统环境下存在

2.5.1 分区挡案修改

相同的开发板可能会上不同的 Flash，SDK 会以不同的板号作区隔。比如说 cv1811c_wdmb_0006a_spinand 与 cv1811c_wdmb_0006a_spinor 分别代表在开发板上的 Flash 各为 SPINAND 及 SPINOR，分区档案分别置于

./build/boards/<CHIP>/<EVB_Name>/partition/partition_<physical_partition>.xml

Note：physical_partition 支持 SPINAND/SPINOR。

例如 cv1811c_wdmb_0006a_spinor 的分区档案，陈列如下：

单系统环境下：

```
build/boards/cv181x/cv1811c_wdmb_0006a_spinor/partition/partition_spinor.xml
<physical_partition type="spinor">
  <partition label="fip" size_in_kb="800" readonly="false" file="fip.bin"/>
  <partition label="BOOT" size_in_kb="2600" readonly="false" file="boot.spinor"/>
  <partition label="ENV" size_in_kb="64" file="" />
  <partition label="ROOTFS" size_in_kb="4000" readonly="false" file="rootfs.spinor" />
  <partition label="DATA"  size_in_kb="512" readonly="false" file="data.spinor" mountpoint="/
↪mnt/data" type="jffs2" />
</physical_partition>
```

双系统环境下：

```
build/boards/cv181x/cv1811c_wdmb_0006a_spinor/partition/partition_spinor.xml
<physical_partition type="spinor">
  <partition label="fip" size_in_kb="512" readonly="false" file="fip.bin"/>
  <partition label="2nd" size_in_kb="3072" readonly="false" file="yoc.bin"/>
  <partition label="BOOT" size_in_kb="5120" readonly="false" file="boot.spinor"/>
  <partition label="MISC" size_in_kb="128" readonly="false" file="logo.jpg"/>
  <partition label="PARAM" size_in_kb="64" file="" />
  <partition label="PARAM_BAK" size_in_kb="64" file="" />
  <partition label="ENV" size_in_kb="64" file="" />
  <partition label="ENV_BAK" size_in_kb="64" file="" />
  <partition label="ROOTFS" size_in_kb="3392" readonly="false" file="rootfs.spinor" />
  <partition label="DATA"  size_in_kb="1024" readonly="false" file="data.spinor" mountpoint="/
↪mnt/data" type="jffs2" />
</physical_partition>
```

- physical_partiti type：flash 种类。
- partition label：分区名称。
- size_in_kb：分区大小 (以 KB 为单位) 。
- file：所指向的 image file 名称。
- type：（在 partition label 栏位中）文件系统格式。
- mountpoint：分区挂载路径。

注：2nd 分区为双系统特有分区，仅在双系统环境下存在

3 烧录说明

3.1 使用前准备

- 2.4.2.3 前述章节产生的烧录挡案。
- 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

以 SPINAND 为例	以 SPINOR 为例	以 EMMC 为例
1.8M yoc.bin(双系统)	1.8M yoc.bin(双系统)	1.8M yoc.bin(双系统)
535K fip.bin	508K fip.bin	486K fip.bin
2.6M fw_payload_uboot.bin	182K fip_spl.bin	2.5M fw_payload_uboot.bin
4.0M ramboot.itb	2.49M	5.3M ramboot.itb
2.6M boot.spinand	fw_payload_uboot.bin	2.8M boot.emmc
3M rootfs.spinand	4.3M ramboot.itb	6.7M rootfs.emmc
1.9M cfg.spinand	3.7M boot.spinor	9.5M cfg.emmc
1.9M system.spinand	3.18M rootfs.spinor	4.7M system.emmc
	292B data.spinor	

注：上表中 yoc.bin 文件为双系统独有文件，仅在双系统环境下存在

插入 SD 卡, 并将 CV180X/CV181X 平台接上电源后开机，自动启动烧录程序，DL flag 表示主 IC 侦测到目前 SD Card 中存在可以烧录的挡案。

```
In:    serial
Out:   serial
Err:   serial
Net:
Warning: ethernet@4070000 (eth0) using random MAC address - 3a:6d:3f:aa:9e:d6
eth0: ethernet@4070000
Hit any key to stop autoboot:  0
## Resetting to default environment
Start SD downloading.....
```

平台烧录完成时，可于 UART 端口看到以下信息，将平台断电，拔出 SD 卡，再重开机，即完成烧录过程。(log 会针对每个分区，显示读取挡案，写入 Flash 所在)

```
## Resetting to default environment
Start SD downloading...
497664 bytes read in 25 ms (19 MiB/s)
spinor id = 1C 71 18
SF: Detected EN25QX128A with page size 256 Bytes, erase size 64 KiB, total 16 MiB
......
Header Version:1
2996612 bytes read in 137 ms (20.9 MiB/s)
device 0 offset 0x100000, size 0x2db944
0 bytes written, 2996548 bytes skipped in 0.125s, speed 23972384 B/s
sf update speed 22.196 MB/s
64 bytes read in 3 ms (20.5 KiB/s)
Header Version:1
......
Saving Environment to SPIFlash... Erasing SPI flash...Writing to SPI flash...done
Valid environment: 1
OK
cv180x_c906#
```

3.4 注意事项

请确认 SD Card 被正确格式化为 FAT32 格式。

4 EVB 接口说明

下方图片为 CV180x EVB

A.UART Debug Port

B.Sensor 0 EVB 插槽

C.Sensor 1 EVB 插槽

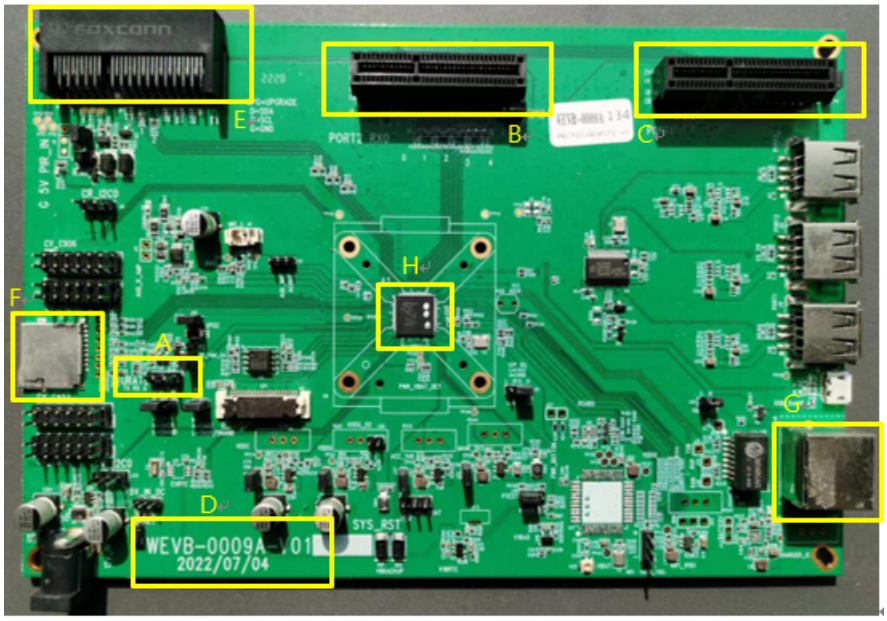
D.EVB 版号

E.Panel EVB 插槽

F.SD 卡插槽

G.Ethernet 0 连接口

H. 主 IC CV180x



下方图片为 CV181x EVB

I.UART Debug Port

J.Sensor 0 EVB 插槽

K.Sensor 1 EVB 插槽

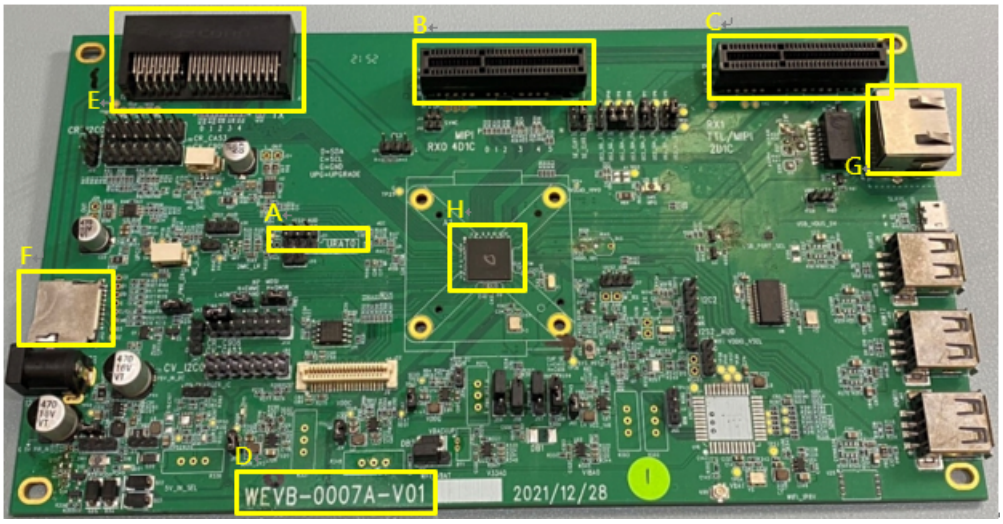
L.EVB 版号

M.Panel EVB 插槽

N.SD 卡插槽

O.Ethernet 0 连接口

P. 主 IC CV181x



5 根文件系统 (rootfs)

5.1 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
```

5.2 Rootfs

本章节是描述文件系统之组成方式，详细路径于 ramdisk/rootfs/

5.2.1 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

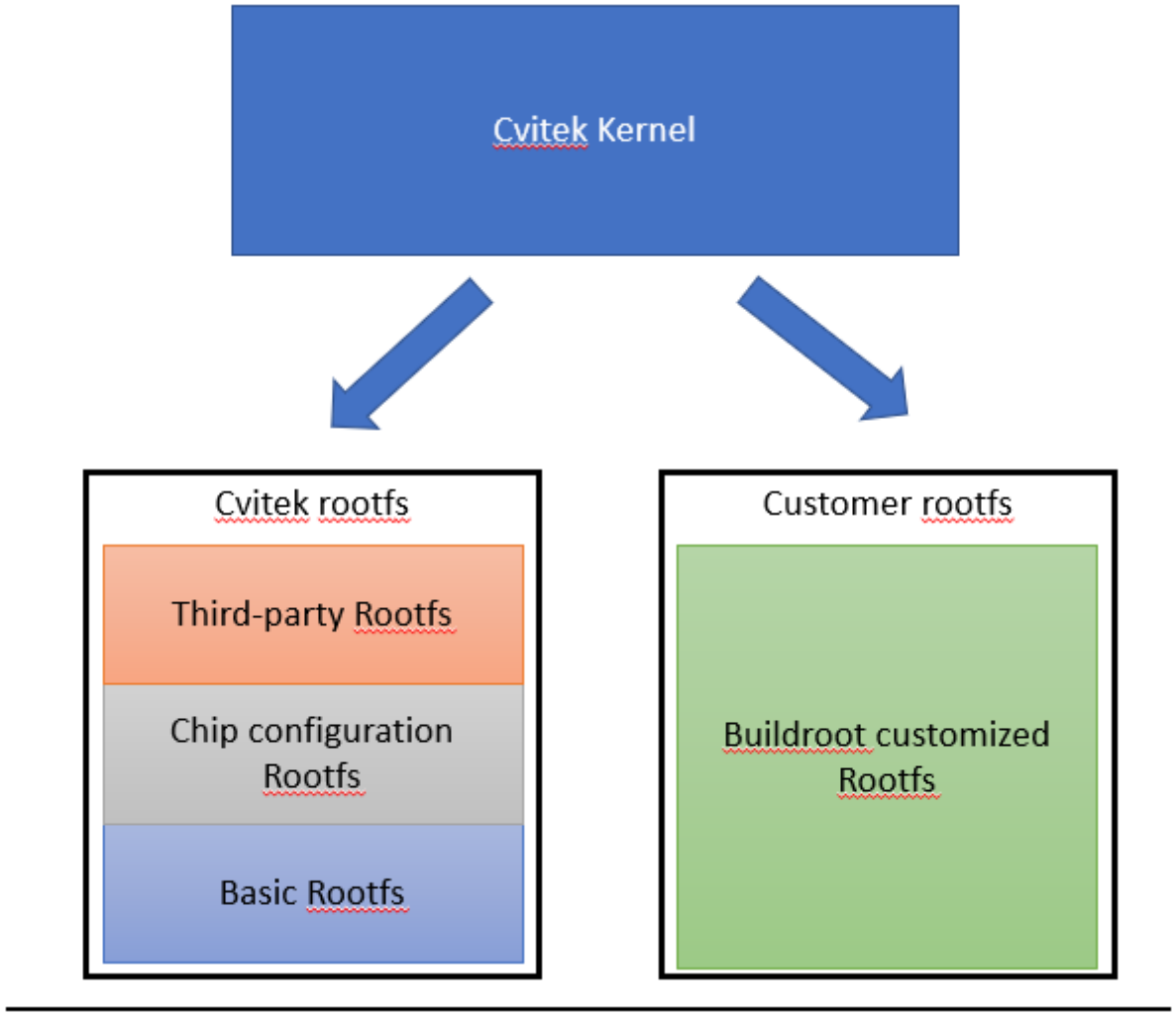
Arch	Libc	Pre-build ramdisk path
Arm	glibc	ramdisk/rootfs/common_arm/
Arm	uclibc	ramdisk/rootfs/common_uclibc/
Aarch64	glibc	ramdisk/rootfs/common_arm64/
Riscv64	glibc	ramdisk/rootfs/common_riscv64/
Riscv64	musl	ramdisk/rootfs/common_musl64/

- **Processor configuration rootfs:**

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

- **Third-party rootfs:**

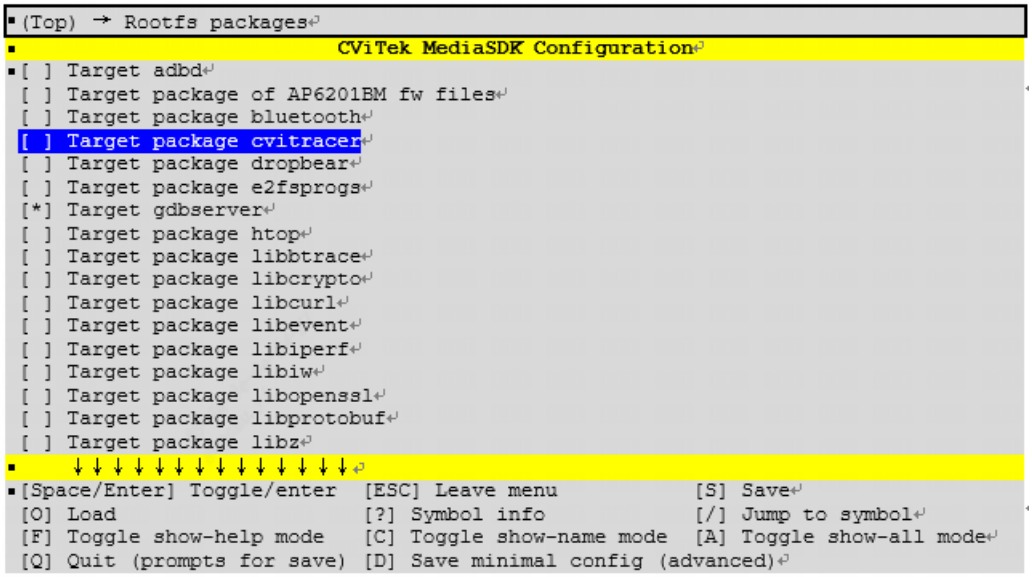
本公司将所有第三方软件编译出来之 library、utility、related file 均放置于 ramdisk/rootfs/public/



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
    Chip selection (cv1801c)  --->
    Board selection (wevb_0009a_spinor (C906B + SPINOR 16MB + QFN SIP 64MB)--->
    DDR configuration selection (ddr3_1866_x16)  --->
(arm64) Arch define
    Compile-time checks and compiler options  --->
    SDK options  --->
    FIP setting  --->
    Storage settings  --->
    Sensor settings  --->
    panel settings  --->
    Kernel options  --->
    ROOTFS options  --->
    Turnkey options  --->
    RTOS options  --->
    Rootfs packages  --->

[Space/Enter] Toggle/enter  [ESC] Leave menu           [S] Save
[O] Load                    [?] Symbol info           [/] Jump to symbol
[F] Toggle show-help mode   [C] Toggle show-name mode [A] Toggle show-all mode
[Q] Quit (prompts for save) [D] Save minimal config (advanced)
```



5.2.2 编译来自 buildroot 的 rootfs

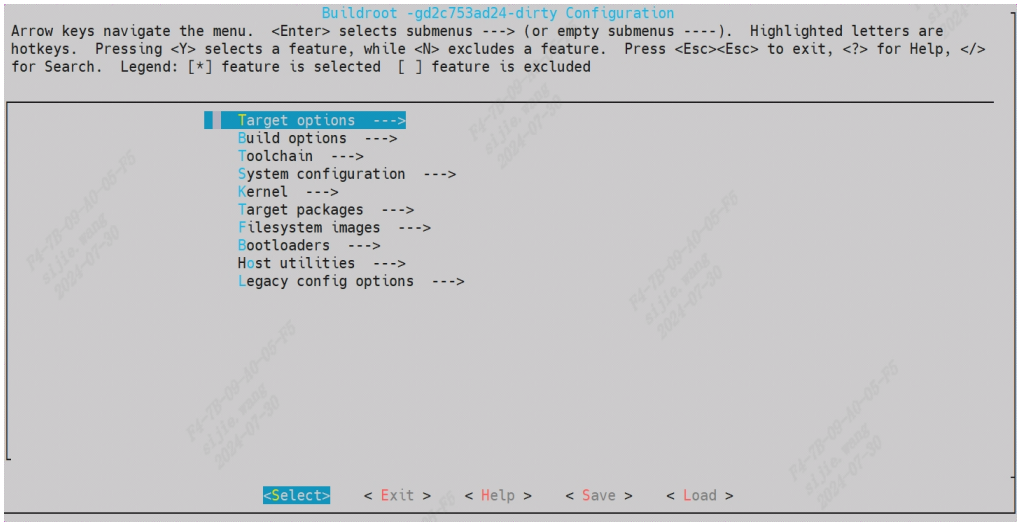
此章节是示范如何从 buildroot 产生 rootfs 并且于 EVB 上面运行的例子，若采用上一章节描述的 pre-build rootfs，可忽略此章节。

1. 拉取 buildroot 仓库

https://github.com/sophgo/buildroot-2021.05

2. 配置 buildroot

```
$ cd buildroot-2021.05

$ make menuconfig
```

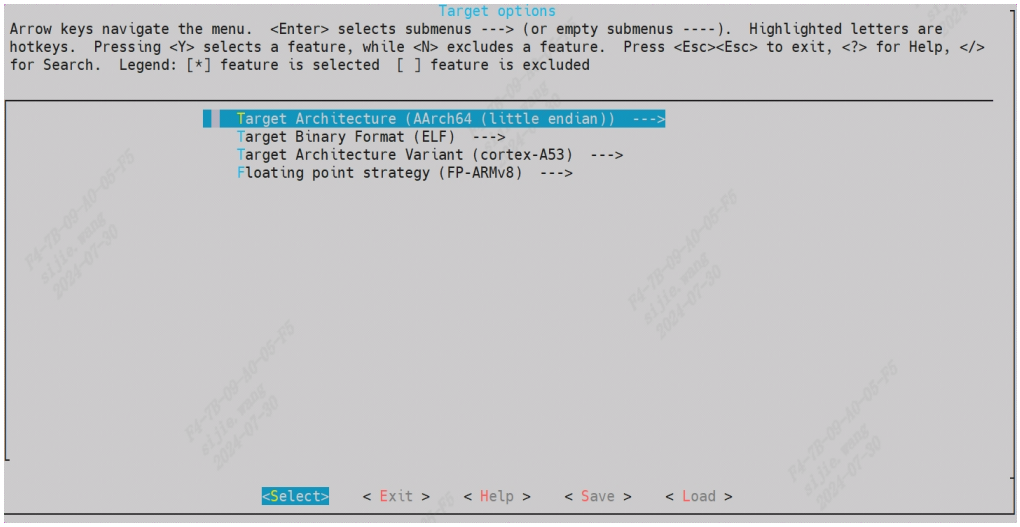


3. 设置 Arch Info & Toolchain & Packages

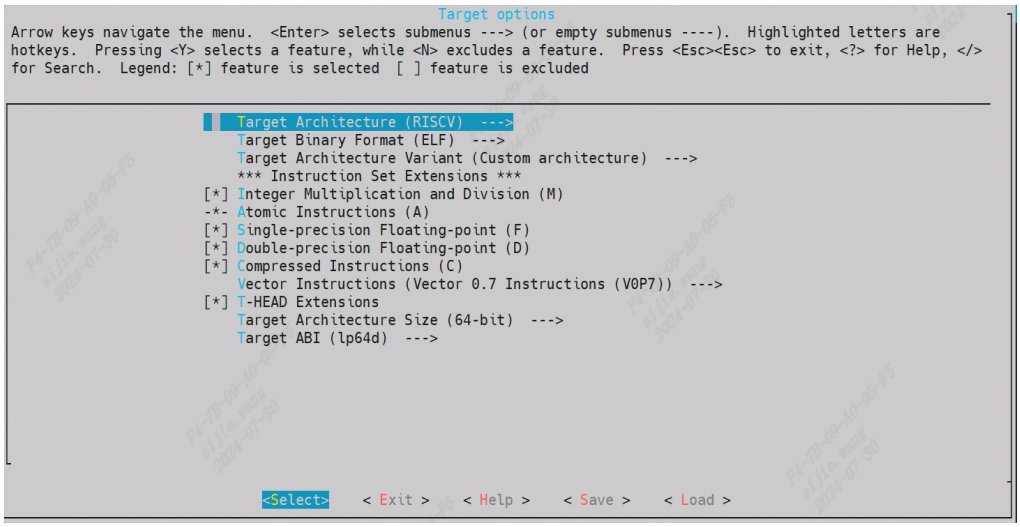
configs 目录下有一些默认配置文件，可执行 make cvitek_XXX_defconfig 快速配置

设置架构

ARM

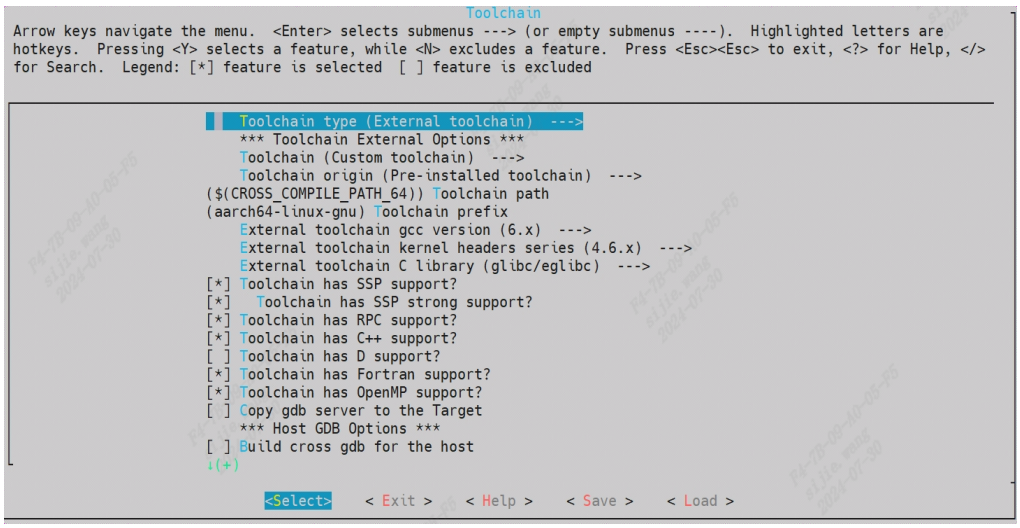


RISCV

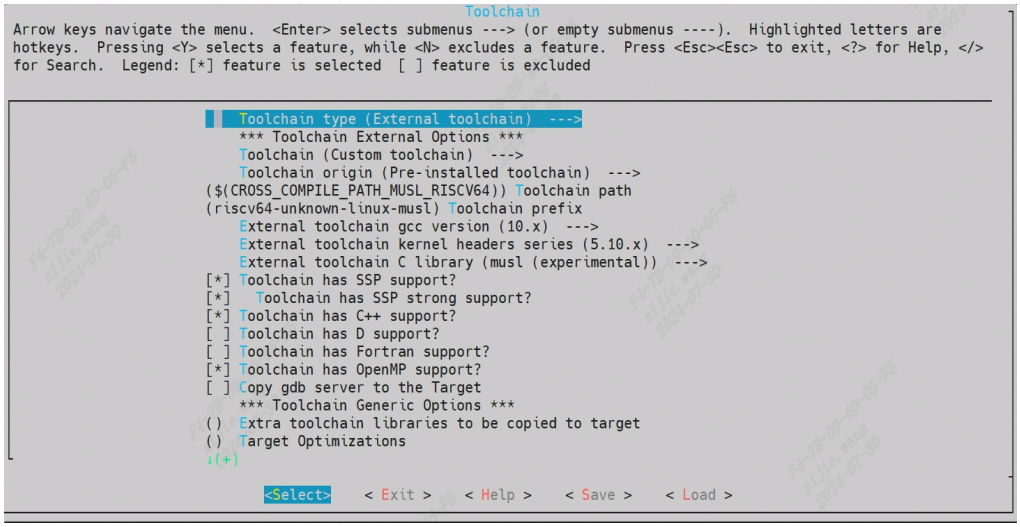


设置工具链

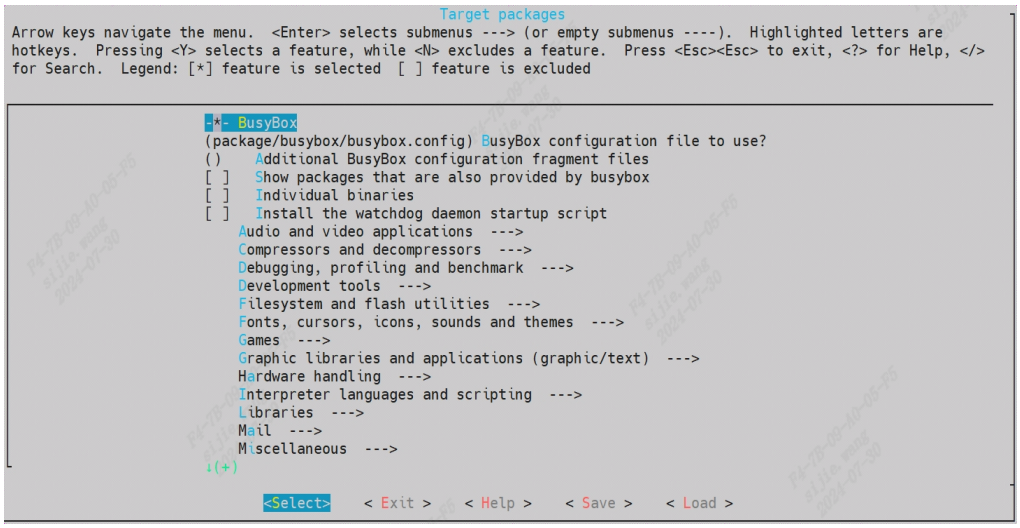
ARM



RISCV



设置需要安装的软件包



4. 编译 buildroot

```
$ 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:$(OUTPUT_DIR)/rootfs
  $(call print_target)
  #clean_all
  rm -rf $(ROOTFS_DIR)/*
  #extract buildroot roofs
  tar xf $(BR_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
```

(下页继续)

(续上页)

```
#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 可烧录映象档

```
$ pack_rootfs
```

8. 透过第二章所提的步骤烧录到版端

5.2.3 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)
  ${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/*
```

(下页继续)

(续上页)

```
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
```

5.2.4 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
  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

6 使用 NFS 加速开发

6.1 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
```

6.2 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/
```

6.3 注意事项：

PC 和板子连接在同一局域网。