



CV180X & CV181X SDK Compilation User Guide

Version: 1.0

Release date: 2022-10-28

Copyright © 2020 CVITEK Co., Ltd. All rights reserved.

No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of CVITEK Co., Ltd.

Contents

1	Disclaimer	2
2	Build CVITEK Software Compilation Environment	3
2.1	Linux Server	3
2.1.1	Install Ubuntu on VirtualBoxVM	3
2.1.2	Ubuntu Boot Settings	4
2.1.3	Install SSH Server	6
2.1.4	Install Samba Server	6
2.2	Build a compilation environment	7
2.2.1	Compile Using Docker	8
2.3	Config github account	10
2.4	Get SDK Source Code	11
2.5	Compilation	11
2.5.1	Environment Variables Description	11
2.5.2	Compile the Entire Package	11
2.5.2.1	Set by defconfig	12
2.5.2.2	Set by Menuconfig	13
2.5.3	Compile the Entire SDK Files	14
2.5.4	Compile part of the SDK files	14
2.5.4.1	Compile Uboot Separately	14
2.5.4.2	Compile kernel separately	16
2.5.4.3	Compile middleware separately	17
2.6	Partitions	17
2.6.1	Partition File Modification	18
3	Burning Instructions	20
3.1	Preparation Before Use	20
3.2	Operation Process	20
3.3	Operation Example	20
3.4	Precautions	21
4	EVB Interface Description	22
5	Root File System (rootfs)	24
5.1	Overview of Root File System	24
5.2	Rootfs	25
5.2.1	Pre-build Rootfs Structure	25
5.2.2	Compile the rootfs from buildroot	26
5.2.3	Package rootfs as a Burnable Image File	30
5.2.4	Linux kernel Auto-Load rootfs	31
6	Accelerate Development with NFS	32

6.1	Ubuntu Server Side Setting Instructions	32
6.2	EVB Side mount Description	32
6.3	Precautions	33

Revision History

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 Disclaimer



Terms and Conditions

The document and all information contained herein remain the CVITEK Co., Ltd' s ("CVITEK") confidential information, and should not disclose to any third party or use it in any way without CVITEK' s prior written consent. User shall be liable for any damage and loss caused by unauthority use and disclosure.

CVITEK reserves the right to make changes to information contained in this document at any time and without notice.

All information contained herein is provided in "AS IS" basis, without warranties of any kind, expressed or implied, including without limitation mercantability, non-infringement and fitness for a particular purpose. In no event shall CVITEK be liable for any third party' s software provided herein, User shall only seek remedy against such third party. CVITEK especially claims that CVITEK shall have no liable for CVITEK' s work result based on Customer' s specification or published shandard.

Contact Us

Address Building 1, Yard 9, FengHao East Road, Haidian District, Beijing, 100094, China

Building T10, UpperCoast Park, Huizhanwan, Zhancheng Community, Fuhai Street, Baoan District, Shenzhen, 518100, China

Phone +86-10-57590723 +86-10-57590724

Website <https://www.sophgo.com/>

Forum <https://developer.sophgo.com/forum/index.html>

2 Build CVITEK Software Compilation Environment

2.1 Linux Server

Developers can choose to use:

- Ubuntu OS computer
- Windows OS computer + Virtualbox VM (run Ubuntu on it)

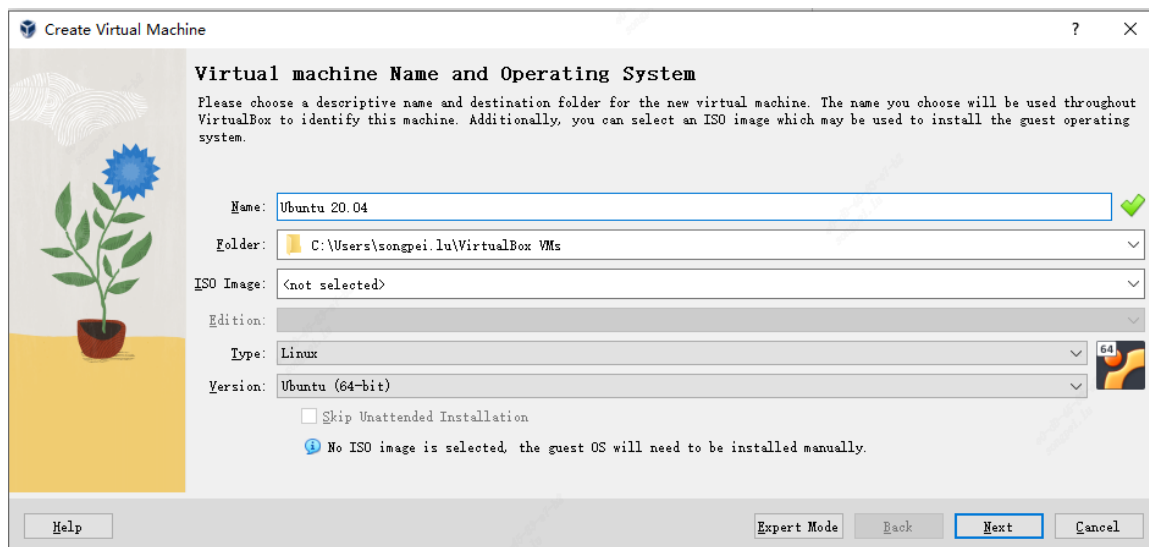
In both cases, install Ubuntu 20.04 LTS.

Virtualbox VM Download site: <https://www.virtualbox.org/wiki/Downloads>

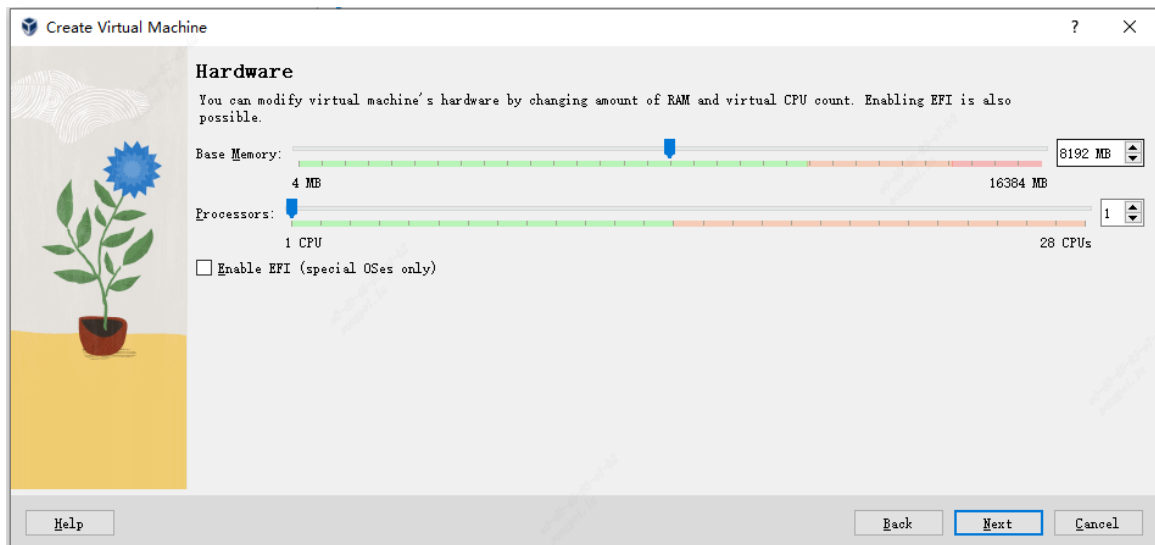
Ubuntu 20.04 LTS Download site: <https://releases.ubuntu.com/focal/ubuntu-20.04.6-desktop-amd64.iso>

2.1.1 Install Ubuntu on VirtualBoxVM

- Create and name the new VM



- Plan 8GB memory for VM use.

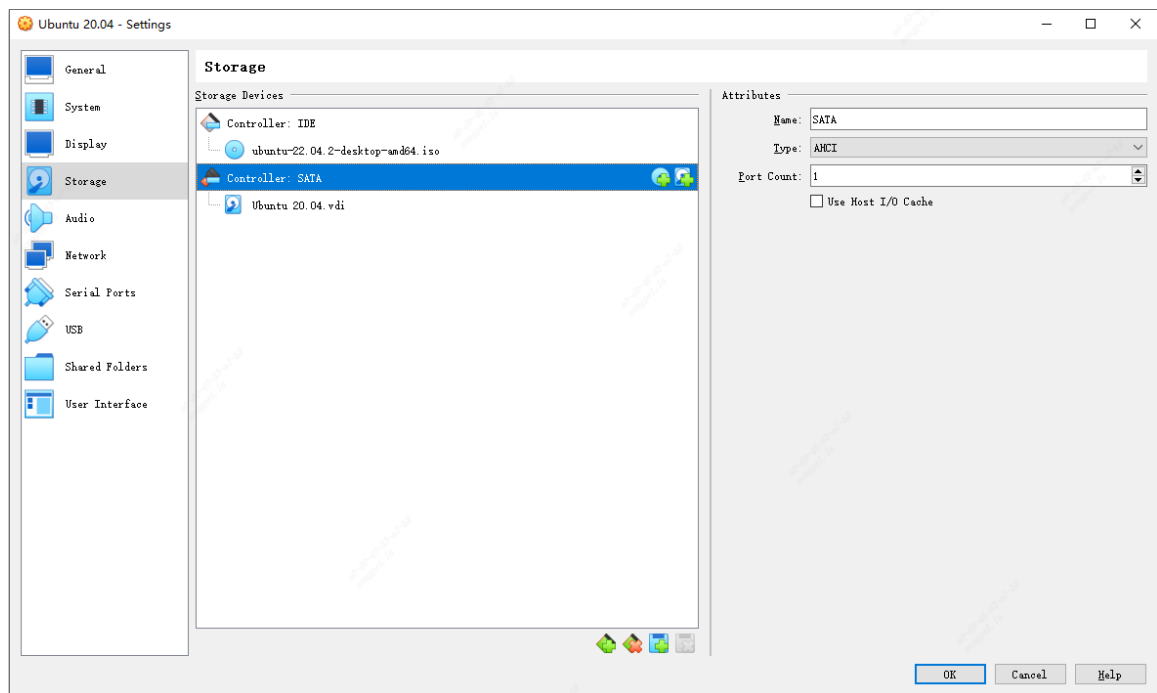


- Reserve 200GB hard disk space for subsequent storage of SDK.

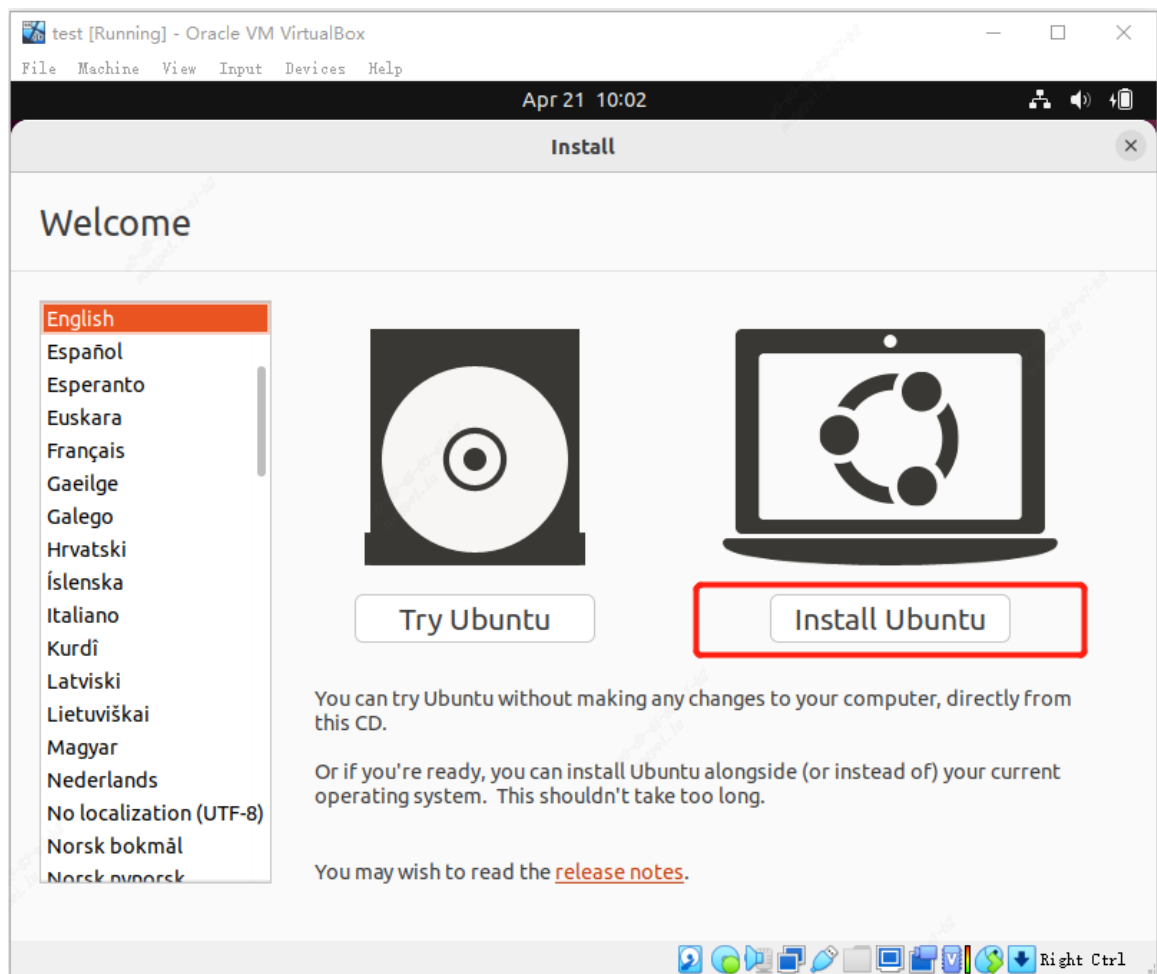


2.1.2 Ubuntu Boot Settings

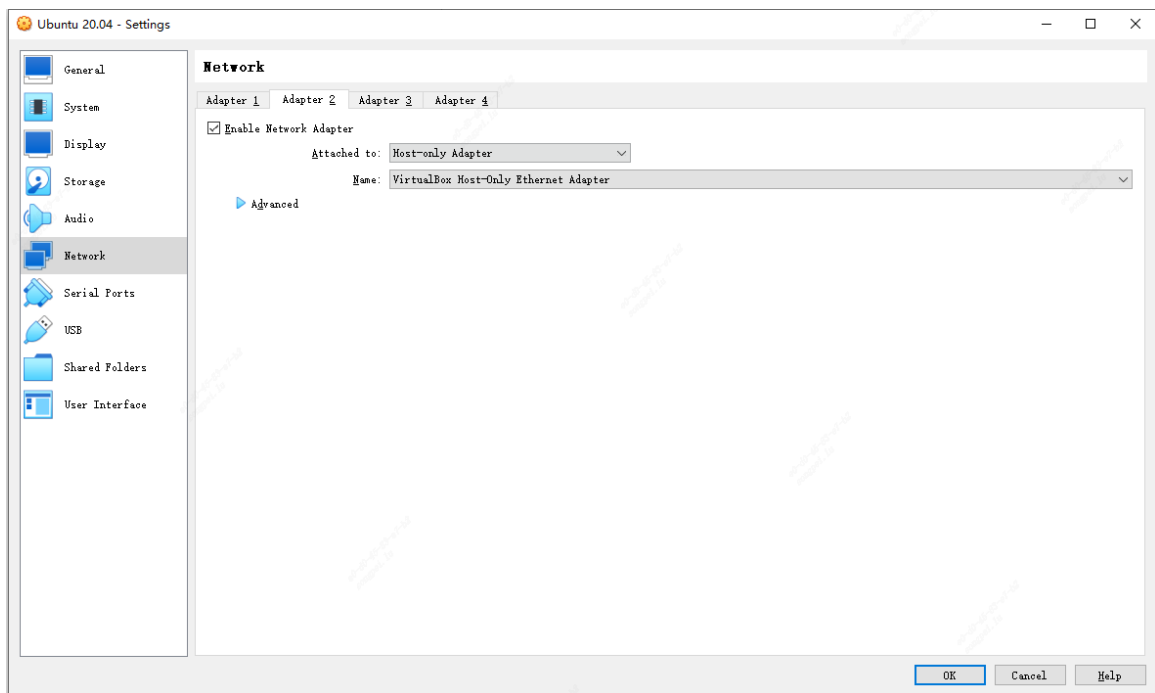
- First boot requires mounting the installation disk's ISO file



- Start Installation



- Set up VirtualBox Host-only Ethernet Adapter for Host to communicate with VirtualBox (terminal services and files sharing)



2.1.3 Install SSH Server

SSH Server Installation

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

After installation, some ssh settings can be changed, such as port, password authentication, root login, etc.

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

Restart SSH after modification

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

2.1.4 Install Samba Server

Ubuntu VB requires the Samba package to be installed for subsequent files sharing with Host PC.

Before installing Samba, use ifconfig to get IP information, the first installation will show that there is no net-tool support, it needs to install net-tool

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

Create the samba password for the account

```
sudo smbpasswd -a cvitek
```

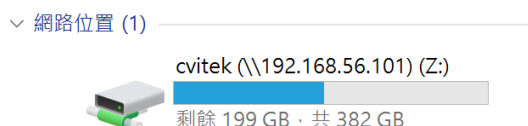
Modify /etc/samba/smb.conf and add the following

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

Start samba server

```
sudo service smbd restart
```

WINDOW PC connects to Samba server (<Server IP>)



Refer to 1.2 to install CVITEK Build Environment to compile.

2.2 Build a compilation environment

Before compiling the SDK, the following packages need to be installed in Ubuntu.

```
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
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
```

(continues on next page)

(continued from previous page)

```

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
sudo apt-get install -y pkg-config
sudo pip3 install -U yoctools

```

Attention: Make sure to check if the python command exists. If it doesn't, a symbolic link needs to be created to the python3 command.

```
sudo ln -s /usr/bin/python3 /usr/bin/python
```

2.2.1 Compile Using Docker

You can map the SDK to a docker container and run compilation commands inside the container. If you have issues with your compilation environment, you can use this method.

1. Prepare Dockerfile, save the following content to file cvitek-linux-Dockerfile

```

# based on ubuntu:20.04
FROM ubuntu:20.04

ENV DEBIAN_FRONTEND=noninteractive
ENV TZ=Asia/Shanghai

# install deps
RUN apt-get update \
    && apt-get install -y \
    pkg-config

RUN DEBIAN_FRONTEND=noninteractive apt-get install -y \
    build-essential \
    ninja-build \

```

(continues on next page)

(continued from previous page)

```
automake \  
autoconf \  
libtool \  
wget \  
curl \  
git \  
gcc \  
libssl-dev \  
bc \  
slib \  
squashfs-tools \  
android-sdk-libsparse-utils \  
android-sdk-ext4-utils \  
jq \  
cmake \  
python3-distutils \  
tclsh \  
scons \  
parallel \  
ssh-client \  
tree \  
python3-dev \  
python3-pip \  
device-tree-compiler \  
libssl-dev \  
ssh \  
cpio \  
squashfs-tools \  
fakeroot \  
libncurses5 \  
flex \  
bison \  
rsync \  
&& apt-get clean \  
&& rm -rf /var/lib/apt/lists/*  
  
# install yoctools for building alios  
RUN ln -s /usr/bin/python3 /usr/bin/python  
RUN pip3 install yoctools -U  
  
# set work dir  
WORKDIR /cvitek-sdk  
  
# set mount point  
VOLUME ["/cvitek-sdk"]  
  
CMD ["/bin/bash"]
```

2. Build docker image


```
docker build -t cvitek-linux -f cvitek-linux-Dockerfile .
```

3. After getting the SDK to folder `/data/xxx/SDK/` (see description below), start the container:

```
docker run -it --name cvitek-linux \
-v /data/xxx/SDK:/cvitek-sdk \
cvitek-linux
```

2.3 Config github account

Create a personal account on GitHub and set up the SSH key. A personal GitHub account is required to download the code.

1. Set up account email address

```
git config --global user.name "your_name"

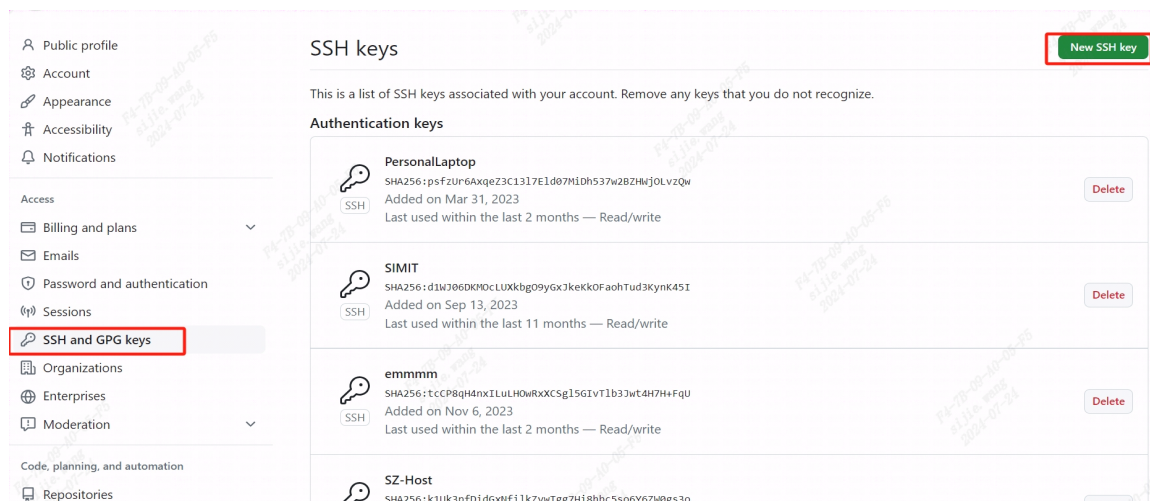
git config --global user.email "your_email@example.com"
```

2. Configure the ssh key

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

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

3. Add the public key to GitHub



4. Verify whether SSH is configured successfully

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

2.4 Get SDK Source Code

1. Use Git to pull the latest SDK source code from GitHub. See: <https://github.com/sophgo/sophpi>

```
git clone -b sg200x-evb git@github.com:sophgo/sophpi.git
# 获取 v4.1.0 SDK
./sophpi/scripts/repo_clone.sh --gitclone sophpi/scripts/subtree.xml
```

2.5 Compilation

2.5.1 Environment Variables Description

The pre-compilation operation is mostly for setting two environment variables: \$CHIP, \$BOARD, \$CHIP variable is required to be set according to the user's SOC.

\$BOARD variable has a different driver for each EVB and must be set correctly.

For example:

\$BOARD=wevb_0008a_spinor: is SPINOR + DDR2 1333 64 MB hardware combination

\$BOARD=wevb_0009a_spinand: is SPINAND + DDR3 1866 128MB hardware combination

Note: wevb_0008a / wevb_0009a can be obtained by the EVB laser model directly.

2.5.2 Compile the Entire Package

Before setting the environment variables, the following commands are required to initialize the environment. The ICs and EVB version numbers currently supported by the SDK will be listed by the system.

```
$ source build/cvsetup.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.
```

(continues on next page)

(continued from previous page)

```
ex: $ defconfig cv1835_wevb_0002a
ex: $ defconfig cv1826_wevb_0005a_spinand
ex: $ defconfig cv180x_fpga_c906
-----
```

After initialization, the compilation configuration can be set in the following two ways

2.5.2.1 Set by defconfig

Select IC: Take cv180x as an example, the system will print out the EVB (\$CHIP_\$BOARD) board supported by cv180x built-in.

```
$ defconfig cv180x
* cv180x * the available 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]
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]
```

Select the EVB version number cv1801c_wevb_0009a_spinor, then the system will list the automatically set environment variables. (Subsequently, cvi_print_env can also be used to print the environment variables currently in use)

```
$ 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
```

2.5.2.2 Set by Menuconfig

After initialization, type in menuconfig to enter the following page to select various SDK internal settings, including CHIP, EVB board number, etc.

```

■ (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)

```

The configuration process can be set/returned by [Enter] [Space] [ESC]

After configuration, press [S] to save the configuration file, then press [Q] to leave the graphical interface

(Or press ESC and a graphical interface will automatically pop up asking whether to save)

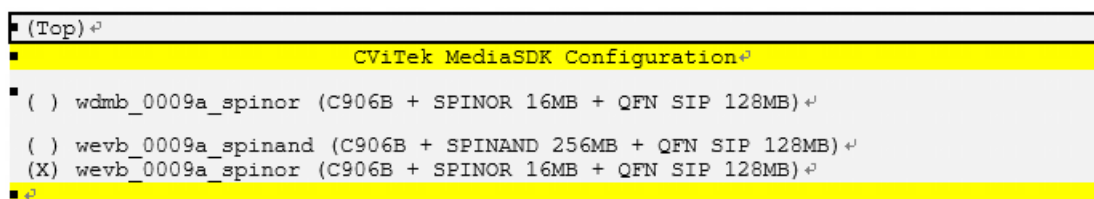
Select IC (cv1801c as an example)

```

■ (Top)
■ CViTek MediaSDK Configuration
■ ( ) none
■ ( ) cv1829
■ ( ) cv1832
■ ( ) cv1835
■ ( ) cv1838
■ ( ) cv7581
■ ( ) cv9520
■ ( ) cv1820
■ ( ) cv1821
■ ( ) cv1822
■ ( ) cv1823
■ ( ) cv1825
■ ( ) cv1826
■ ( ) cv7327
■ ( ) cv7357
■ ( ) cv1810c
■ ( ) cv1811c
■ ( ) cv1811h
■ ( ) cv1812h
■ ( ) cv181x
■ ( ) cv1820a
■ ( ) cv1821a
■ ( ) cv1823a
■ ( ) cv1800b
■ ( ) cv1800c
■ ( ) cv1801b
■ (X) cv1801c
■ ( ) cv180x

```

The EVB version number will list the corresponding choices, and the selection of EVB will also specify the compiled image's DDR and Flash size (for this example, the selected EVB comes with DDR3 and 16MB of SPINOR Flash).



Finally, exit the selection and save the settings (they will be stored in. /build/.config) to complete the selection of the SDK compilation configuration.

2.5.3 Compile the Entire SDK Files

Execute the compilation and get the images available for burning.

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

The compiled file will be placed in ./install/soc_<EVB Name>/

2.5.4 Compile part of the SDK files

2.5.4.1 Compile Uboot Separately

Each EVB board defines in a specific location the initialization operations that the EVB needs to take before entering U-Boot or defines a specific PINMUX. Taking the board cv1801c_wevb_0009a_spinor as an example, they will be defined in.

```
build/boards/CV181X/$CHIP_$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;
    }
}
```

The corresponding u-boot configuration, defined in CV181X:

```
./build/boards/CV181X/$CHIP_$BOARD/u-boot/$CHIP_$BOARD_defconfig
```

```
Partial cvitek_cv1801c_wevb_0009a_spinor_defconfig
CONFIG_RISCV=y
```

(continues on next page)

(continued from previous page)

```
CONFIG_SYS_MALLOC_F_LEN=0x2000
CONFIG_NR_DRAM_BANKS=1
CONFIG_DEFAULT_DEVICE_TREE="cv180x_asic"
CONFIG_IDENT_STRING=" cvitek_cv180x"
```

Modify Uboot Config with a graphical interface

```
$ menuconfig uboot.
.config - U-Boot 2021.10 Configuration.
Arrow keys navigate the menu. <Enter> selects submenus --- (or
empty submenus ---). Highlighted letters are hotkeys. Pressing
<Y> includes, <N> excludes, <M> modularizes features. Press
<Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*]
built-in [ ]
Architecture select (RISCv architecture) --->
ARM architecture --->
General setup --->
Boot images --->
API --->
Boot timing --->
Boot media --->
Environment ----
(1) delay in seconds before automatically booting.
Console --->
<Select> <Exit> <Help> <Save> <Load>
```

After exiting, the settings will be stored in:

```
./u-boot/build/"$CHIP"_"$BOARD"/.config
```

Execute compilation

```
$ build_uboot
```

When completed, fip.bin will be generated

Compile U-Boot fragments in Makefile

```
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)
```

2.5.4.2 Compile kernel separately

Modify the kernel (ex:*.dts, kernel) and recompile the Linux kernel image.

Each EVB has a corresponding dts file to define its device tree, take cv1801c_wevb_0009a_spinor for example, its DTS file is defined in:

```
./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"

/ {
```

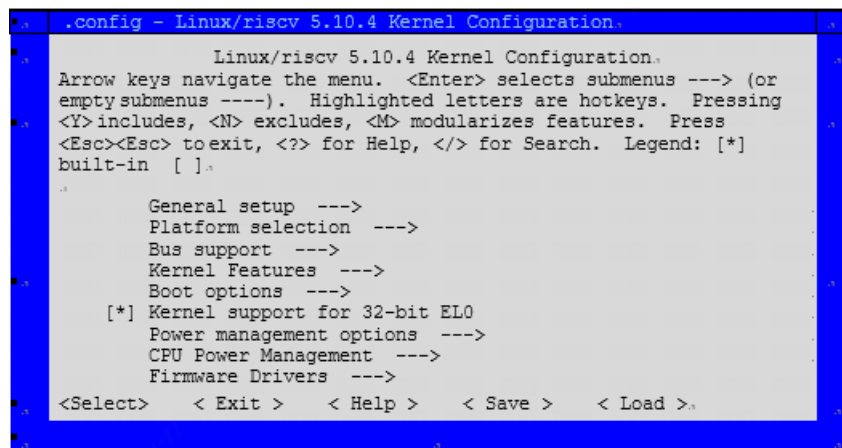
Its corresponding linux configuration is defined in:

```
./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
...
```

Modify Kernel Config with graphical interface

```
$ menuconfig_kernel
```



After exiting, the settings will be store in:

```
./linux/build/"$CHIP"_"$BOARD"/.config
```

```
$ build_kernel
```

When completed, boot.spinor will be generated

Compile Kernel fragments in Makefile.

```
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_install INSTALL_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
```

2.5.4.3 Compile middleware separately

Modify middleware (cvi_test / sample_dsi), recompile middleware and system

The generated Install/PROJECT_NAME/system.* contains the latest middleware

```
$ build_middleware; pack_rootfs
```

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

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

build_middleware will recompile the Sensor driver (in middleware/component/isp/) and sample application (in middleware/sample/), and finally pack_rootfs will package the changed driver and application into a burnable image.

2.6 Partitions

CV181X SDK will generate the following image files, each representing a different partition, listed below:

- FIP : Bootloader/U-Boot partition
 - CV180X/ CV181X C906 adopts FSBL+OPENSBI+UBOOT architecture, and also reuses (FIP) file name after the final package, which is convenient for subsequent use.

- BOOT : Partition for Linux Kernel
- 2nd(dual-system) : The partition where Yun on Processor (YOC) is located.
- MISC : Boot Logo partition
- ROOTFS : root file system partition
- SYSTEM: Partition where CVITEK libraries are located
- DATA : Using data partition

Note: The 2nd partition is a dual-system-specific partition and exists only in a dual-system environment.

2.6.1 Partition File Modification

The same EVB may have different Flash, and the SDK will separate them with different board numbers. For example cv1811c_wdmb_0006a_spinand and cv1811c_wdmb_0006a_spinor represent the Flash on the development board as SPINAND and SPINOR respectively, and the partition files are placed in

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

Note: physical_partition supports SPINAND/SPINOR.

For example, the partition files of cv1801c_wdmb_0009a_spinor are displayed as follows:

In a single-system environment:

```
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>
```

In a dual-system environment:

```
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="" />
```

(continues on next page)

(continued from previous page)

```
<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 type.
- partition label: partition name.
- size_in_kb: partition size (in KB).
- file: name of the image file pointed to.
- type: (in the partition tab field) file system format.
- mountpoint: partition mount path.

Note: The 2nd partition is a dual-system-specific partition and exists only in a dual-system environment.

3 Burning Instructions

3.1 Preparation Before Use

- The burning files generated by the previous section 2.4.2.3.
- Micro SD card in FAT32 format.

3.2 Operation Process

- Put the burning files (as shown in the table below) into the SD card.
- Insert the SD card into the SD card slot of the CVITEK EVB.
- Reboot the platform.

3.3 Operation Example

Confirm the files before use

Take SPINAND as an example	Take SPINOR as an example	Take EMMC as an example
1.8M yoc.bin(dual-system)	1.8M yoc.bin(dual-system)	1.8M yoc.bin(dual-system)
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	

Note: The “yoc.bin” file in the above table is exclusive to the dual-system and exists only in a dual-system environment.

Insert the SD card and connect the CV180X/CV181X platform to the power supply and then boot it up, the burning process will start automatically. DL flag indicates that the main IC detects that there are files in the SD Card that can be burned at present.

```

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.....

```

When the platform is finished burning, the following information can be seen in the UART port, power off the platform, pull out the SD card and reboot it, then the burning process is finished. (The log will show for each partition the files read and write to the Flash location)

```

## 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#

```

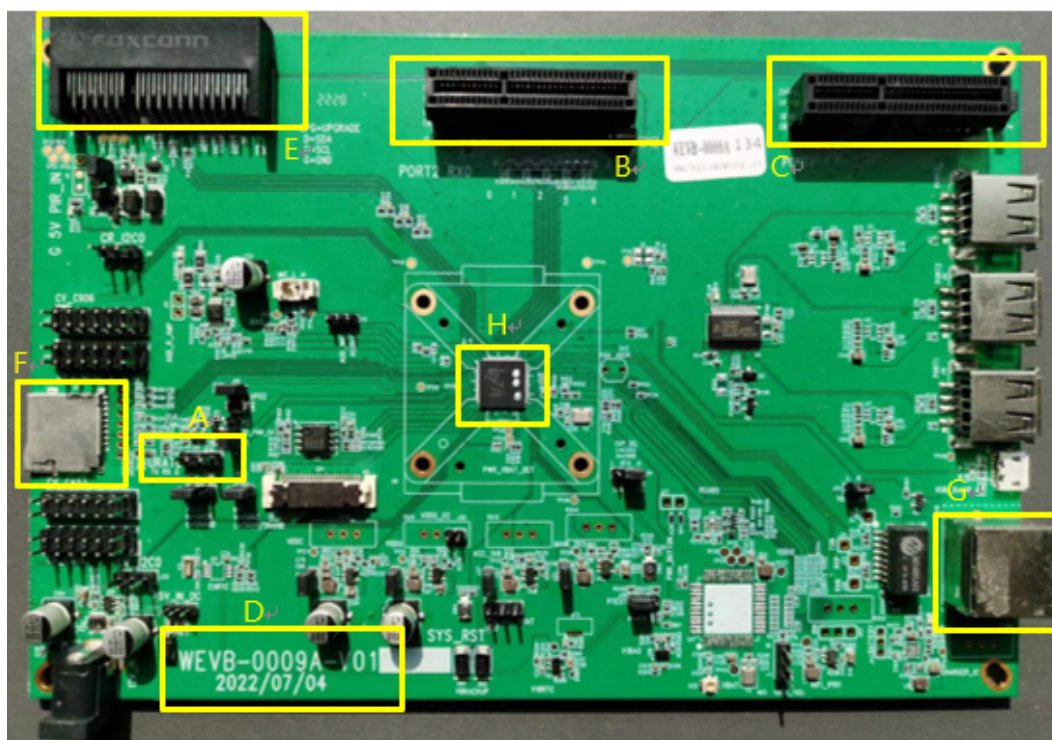
3.4 Precautions

Please make sure the SD Card is properly formatted to FAT32 format.

4 EVB Interface Description

The picture below shows CV180x EVB

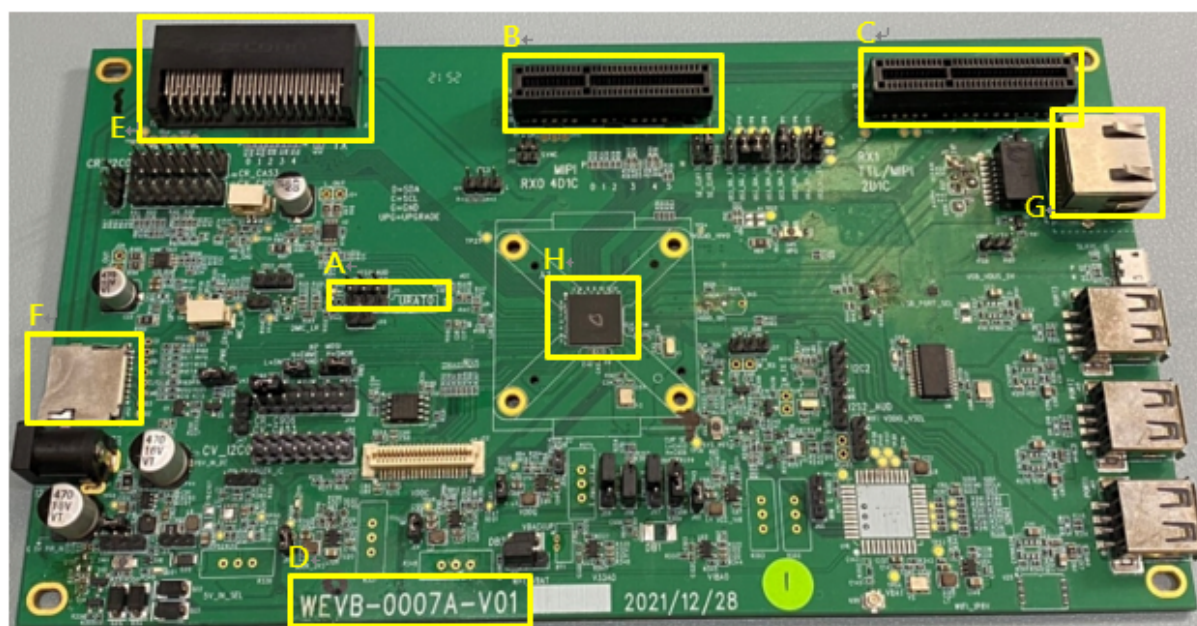
- A. UART Debug Port
- B. Sensor 0 EVB slot
- C. Sensor 1 EVB slot
- D. EVB version number
- E. Panel EVB slot
- F. SD card slot
- G. Ethernet 0 connection port
- H. Host IC CV180x



The picture below shows CV181x EVB

- I. UART Debug Port
- J. Sensor 0 EVB slot

- K. Sensor 1 EVB slot
- L. EVB version number
- M. Panel EVB slot
- N. SD card slot
- O. Ethernet 0 connection port
- P. Host IC CV181x



5 Root File System (rootfs)

5.1 Overview of Root File System

The kernel is the core of the Linux operating system and the file system is the main tool for communication between the user and the operating system. So to use Linux, it is important to understand the file system principles first.

The root file system structure is a tree directory structure starting with “/” as the “root” , and a device (ex: eMMC) is mounted on the root directory when the kernel program image (uImage) is booted. The root file system is usually stored in internal memory (DRAM) or non-volatile memory (FLASH), or in a file system accessible via the network (NFS). All applications and libraries are placed in the file system according to categories, and the following diagram shows the root file system directory structure.

```
/      Root directory
bin   Executable file
dev   Device file
etc   System configuration file (ex: startup file)
home  User Directory
init  Script executed at boot
kdump Kernel debugging directory
lib   Libraries include glibc, shared library and kernel modules
mnt   Mount points for temporary file systems
proc  Virtual file system for kernel and process information
sbin  System managed executable file
sys   System device and file hierarchy, providing kernel data
usr   This directory contains user-defined applications and files
var   Store system logs and service program files
```


5.2 Rootfs

This section describes how the file system is composed, and the detailed path is in `ramdisk/rootfs/`

5.2.1 Pre-build Rootfs Structure

The structure of the file system is divided into three main types of directories, which are iterated over Rootfs layer by layer, and are described below:

- **Basic rootfs:**

At this stage, we provide pre-build rootfs files based on the following four types of Arch

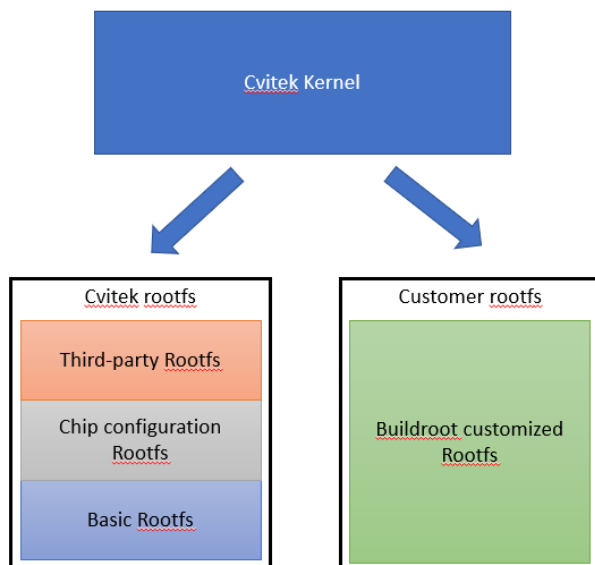
Arch	Libc	Pre-build ramdisk path
Arm	glibc	<code>ramdisk/rootfs/common_arm/</code>
Arm	uclibc	<code>ramdisk/rootfs/common_uclibc/</code>
Aarch64	glibc	<code>ramdisk/rootfs/common_arm64/</code>
Riscv64	glibc	<code>ramdisk/rootfs/common_riscv64/</code>
Riscv64	musl	<code>ramdisk/rootfs/common_musl64/</code>

- **Processor configuration rootfs:**

We place all Processorset dependent boot settings in `ramdisk/rootfs/overlay/$CHIP`

- **Third-party rootfs:**

We place all third-party software compiled library, utility, and related file in `ramdisk/rootfs/public/`



It can be decided by means of a menu which Third-party software should be placed in 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)↵

■ (Top) → Rootfs packages ↵
■ CViTek MediaSDK Configuration ↵
■ [ ] Target adbd↵
  [ ] Target package of AP6201BM fw files↵
  [ ] Target package bluetooth↵
  [ ] Target package cvitracer↵
  [ ] Target package dropbear↵
  [ ] Target package e2fsprogs↵
  [*] Target gdbserver↵
  [ ] Target package htop↵
  [ ] Target package libbtrace↵
  [ ] Target package libcrypto↵
  [ ] Target package libcurl↵
  [ ] Target package libevent↵
  [ ] Target package libiperf↵
  [ ] Target package libiw↵
  [ ] Target package libopenssl↵
  [ ] Target package libprotobuf↵
  [ ] Target package libz↵
■ ↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↵
■ [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 Compile the rootfs from buildroot

This section is an example of how to generate the rootfs from buildroot and run it on EVB. If the pre-build rootfs described in the previous section is used, this section can be ignored.

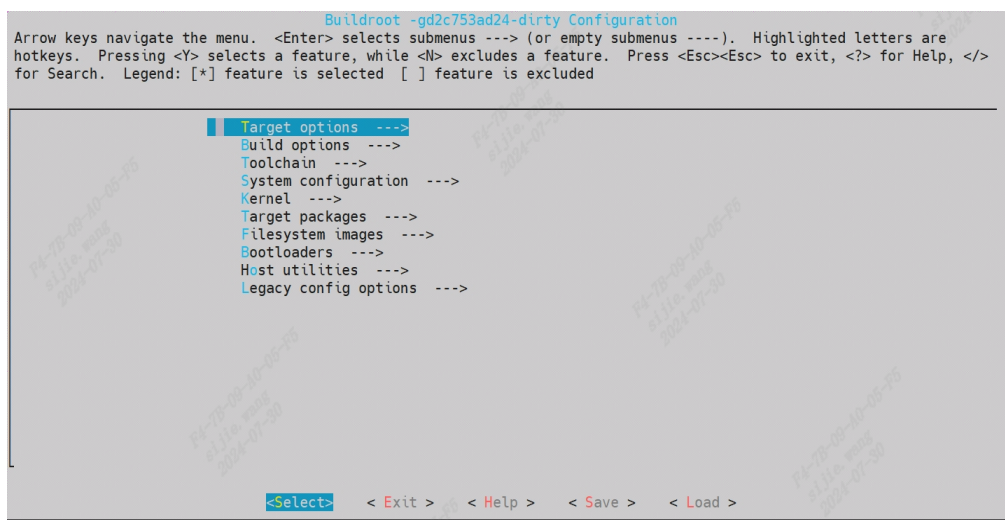
1. Get buildroot repository

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

2. Configure buildroot

```
$ cd buildroot-2021.05
```

```
$ make menuconfig
```

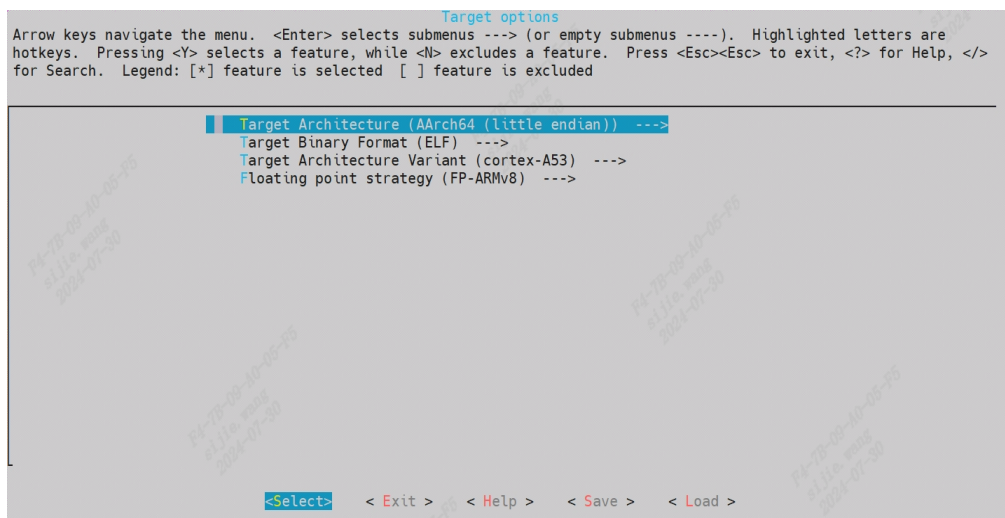


3. Set Arch Info & Toolchain & Packages

There are some default configuration files in the configs directory, which can be quickly configured by running `make cvitek_XXX_defconfig`

Set the architecture

ARM



RISCV

```

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

Target Architecture (RISCv) --->
  Target Binary Format (ELF) --->
  Target Architecture Variant (Custom architecture) --->
  *** Instruction Set Extensions ***
  [*] Integer Multiplication and Division (M)
  -- Atomic Instructions (A)
  [*] Single-precision Floating-point (F)
  [*] Double-precision Floating-point (D)
  [*] Compressed Instructions (C)
  Vector Instructions (Vector 0.7 Instructions (V0P7)) --->
  [*] T-HEAD Extensions
  Target Architecture Size (64-bit) --->
  Target ABI (lp64d) --->

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

```

Set the Toolchain

ARM

```

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

Toolchain type (External toolchain) --->
  *** Toolchain External Options ***
  Toolchain (Custom toolchain) --->
  Toolchain origin (Pre-installed toolchain) --->
  ($(CROSS_COMPILE_PATH_64)) Toolchain path
  (aarch64-linux-gnu) Toolchain prefix
  External toolchain gcc version (6.x) --->
  External toolchain kernel headers series (4.6.x) --->
  External toolchain C library (glibc/eglibc) --->
  [*] Toolchain has SSP support?
  [*] Toolchain has SSP strong support?
  [*] Toolchain has RPC support?
  [*] Toolchain has C++ support?
  [ ] Toolchain has D support?
  [*] Toolchain has Fortran support?
  [*] Toolchain has OpenMP support?
  [ ] Copy gdb server to the Target
  *** Host GDB Options ***
  [ ] Build cross gdb for the host
  i(+)

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

```

RISCV

```

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

Toolchain type (External toolchain) --->
  *** Toolchain External Options ***
  Toolchain (Custom toolchain) --->
  Toolchain origin (Pre-installed toolchain) --->
  ($(CROSS_COMPILE_PATH_MUSL_RISCV64)) Toolchain path
  (riscv64-unknown-linux-musl) Toolchain prefix
  External toolchain gcc version (10.x) --->
  External toolchain kernel headers series (5.10.x) --->
  External toolchain C library (musl (experimental)) --->
  [*] Toolchain has SSP support?
  [*] Toolchain has SSP strong support?
  [*] Toolchain has C++ support?
  [ ] Toolchain has D support?
  [ ] Toolchain has Fortran support?
  [*] Toolchain has OpenMP support?
  [ ] Copy gdb server to the Target
  *** Toolchain Generic Options ***
  ( ) Extra toolchain libraries to be copied to target
  ( ) Target Optimizations
  i(+)

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

```

Set the software package


```

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

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

4. Compile the buildroot package

```
$ make
```

5. Obtain the rootfs, which is located in output/images/rootfs.tar

6. Modify build/Makefile so that SDK can use rootfs generated by buildroot

```

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
#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. Generate the new ROOTFS burnable image files

```
$ pack_rootfs
```

8. Burn to the board by the steps mentioned in Chapter 3

5.2.3 Package rootfs as a Burnable Image File

Package the rootfs folder from the previous steps with the mksquashfs tool, compress it with XZ, and the end product is rootfs.spinor / rootfs.spinand / rootfs.emmc, which can be burned to Flash.

Refer to the rootfs-pack in build/Makefile for details:

```
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) $(OUTPUT_DIR)/rawimages/rootfs.sqsh -root-
↪owned -comp xz -e mnt/cfg/*
endif
ifeq ($(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 Auto-Load rootfs

The Linux kernel will determine which device the rootfs is located on based on the `root=` variable in the bootargs set by uboot

```
'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 Accelerate Development with NFS

6.1 Ubuntu Server Side Setting Instructions

Install nfs-kernel-server

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

例: `mkdir /home/nfs_server`

Modify the `/etc/exports` file and add the following

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

restart nfs service

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

6.2 EVB Side mount Description

Create mount point in the file system `/mnt/data`

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

mount nfs

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

6.3 Precautions

PC and board are connected to the same LAN.