CV180X & CV181X Peripheral Driver User Guide

Version: 1.0.1

Release date: 2023-02-06

Copyright © 2020 CVITEK Co., Ltd. All rights reserved.
No part of this document may be reproduced or transmiited in any form or by any means without prior written consent of CVITEK Co., Ltd.

Contents

Revision History

Revision	Date	Description
1.0.0	2022/10/31	Initial version
1.0.1	2023/02/06	Modified to be compatible with cv180x and cv181x
1.0.2	2023/02/23	Modified gpio number

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 Ethernet Operation Guide

2.1 Operation Example

The Ethernet module is built-in in the kernel by default, and there is no need to perform additional insmod operation.

The operation steps of using ethernet port under kernel are as follows :

- Configure ip address and netmask

```
ifconfig eth0 xxx.xxx.xxx.xxx netmask xxx.xxx.xxx.xxx up
```

- Set default gateway

```
route add default gw xxx.xxx.xxx.xxx
```

- Mount nfs

```
mount -t nfs -o nolock xxx.xxx.xxx.xxx:/your/path /mount-dir
```

- Using tftp to upload and download files in shell

  Be sure that there is tftp service software running on the server side

  — download document: `tftp -g -r [remote file name] [server ip]`

  Note: remote file name is the name of the file to be downloaded, and server IP is the IP address of the server where the file download from (ex: tftp -g -r test.txt 192.168.0.11)

  — upload document: `tftp -p -l [local file name] [server ip]`

  Note: local file name is the name of the file to be uploaded locally, and server IP is the IP address of the target server where to upload (ex: tftp -l -p test.txt 192.168.0.11)

  Note: cv180x/cv181x Ethernet module don’ t support TSO function.

  Note: The nfs tool will not be built into the file system by default. The user needs to add the tool by themselves when necessary.

2.2 IPv6 Description

The IPv6 functionality is disabled by default in the SDK package. To enable IPv6, kernel options need to be modified. The specific steps are as follows:

1. Cv180x series

   Modify

   build/boards/cv180x/{board_name}/linux/cvitek_{board_name}_defconfig

   Ex: build/boards/cv180x/cv1801c_wevb_0009a_spinor/linux/

   cvitek_cv1801c_wevb_0009a_spinor_defconfig add or modify to CONFIG_IPV6=y. Then recompile the kernel software.

2. Cv181x series

   Modify

   build/boards/cv181x/{board_name}/linux/cvitek_{board_name}_defconfig

   Ex: build/boards/cv181x/ cv1811c_wevb_0006a_spinor/linux/

   cvitek_cv1811c_wevb_0006a_spinor_defconfig add or modify to CONFIG_IPV6=y. Then recompile the kernel software.

The method for configuring an IPv6 environment is as follows:

- To configure an IPv6 address and gateway, use the following command:

  #ip -6 addr add <IPv6 address>/IPv6 prefixlen dev <port name>

  Ex: ip -6 addr add 2020:abc:102::8888/24 dev eth0

- IPv6 address specified by Ping

  #ping -6 <ipv6 address>

  Ex: ping -6 2020:abc:102::6666

2.3 IEEE 802.3x Flow Control Function

2.3.1 Flow Control Function Description

CV180x/CV181x Ethernet supports the flow control function defined by IEEE 802.3x. It achieves the purpose of flow control by sending flow control frames and receiving the flow control frames sent by the opposite end.

- Send flow control frame:

  In the process of receiving the packets sent by the opposite site, if it is found that the current receiving queue of the receiving site may not be able to receive the subsequent packets, the local site will send the flow control frame to the opposite site, requiring the opposite site to suspend sending packets for a period of time, so as to control the flow.

- Receive flow control frame:

  When the local site receives the flow control frame sent by the opposite site, the local site will delay sending packets to the opposite site according to the flow control time description within the frame, and then start sending after the flow control delay time. If the flow control frame sent by the opposite site is received in the waiting process and the flow control time described is 0, the transmission will be started directly.

2.3.2 Flow Control Function Configuration

The function of receiving flow control frame is off by default, and no software interface configuration is provided.

Send flow control frame function related configuration file in linux/drivers/net/ethernet/stmicro/stmmac/stmmac_main.c

```
static int flow_ctrl = FLOW_OFF;
module_param(flow_ctrl, int, 0644);
MODULE_PARM_DESC(flow_ctrl, "Flow control ability [on/off]");

static int pause = PAUSE_TIME;
module_param(pause, int, 0644);
MODULE_PARM_DESC(pause, "Flow Control Pause Time");
```

To enable the flow control function by default, you can modify the flow_ctrl = FLOW_AUTO.

If you want to modify the default pause time, you can configure "pause" to the target time.

2.3.3 Ethtool Configuration Interface Flow Control Function

Users can enable the flow control function through the standard ethtool tool interface.

ethtool – a eth0 command to view the flow control function status of eth0 port; the print is as follows

```
# ethtool -a eth0
Pause parameters for eth0:
Autonegotiate: on
RX: off
TX: off
```

Among them, RX flow control is off, TX flow control is off; the user can open or close TX flow control through the following command:

```
# ethtool -A eth0 tx off (turn off TX flow control)
# ethtool -A eth0 tx on (turn on TX flow control)
```

Note: The ethtool will not be built into the file system by default. The user needs to add the tool by themselves when necessary.

3 USB Operation Guide

3.1 Operational Readiness

USB 2.0 Host/Device is prepared as follows:

- Use the U-boot and Kernel released by SDK.
- The file system can use the local file system ext4 or squashfs, or NFS.
- Shell script "run_usb.sh" .run_usb.sh uses the USB ConfigFS function of the kernel to customize the USB device. Users can refer to and modify run_usb.sh to change the parameters related to PID / VID and function. For detailed operation, refer to the kernel file " linux/Documentation/usb/gadget_configfs.txt" .

3.2 Uboot Operation Process

3.2.1 The USB Host Operation Process Base on the Uboot

Only pen drive and hard disk storage devices are supported in Uboot. USB host is off by default in uboot. You must enable relevant config.

Step1. enable USB related driver under uboot:

```
CONFIG_USB=y
CONFIG_DM_USB=y
CONFIG_USB_STORAGE=y
CONFIG_CMD_USB=y
```

Step2.

2.1) Cv180x series modify include/configs/cv180x-asic.h.

2.2) Cv181x series modify include/configs/cv181x-asic.h, newly add the definition:

```
#define CONFIG_USB_DWC2
#define CONFIG_USB_DWC2_REG_ADDR        0x04340000
```

Step3. Compile the drivers. Compile uboot to generate fip.bin

```
build_uboot
```

3.2.2 The USB Drive Operation Process Based on the Uboot

The preparation before starting the Uboot USB Host:

USB Host base on the Uboot does not support hot plug. You must plug in the device before starting the USB host. If a USB Hub is installed on the platform, ensure that the Hub' s power source is enabled and the Switch in the USB path is switched to the Host connector.

Take cv180x as an example (the corresponding method is also applicable to cv181x) :

Power the platform, access the uboot Command Line Interface, and run the usb start command to check whether the identification is successful.

```
phobos_c906# usb start
starting USB...
USB0:   Core Release: 4.00a
scanning bus 0 for devices... Device NOT ready
   Request Sense returned 02 3A 00
2 USB Device(s) found
      scanning usb for storage devices... 2 Storage Device(s) found
```

If an enumeration error occurs after "usb start" or the device cannot be detected, run the "setenv usb_pgood_delay XXX" command on the uboot command- line interface (CLI), you can adjust the timeout value for a device that is preheated slowly or connected to the Hub. The recommended value ranges from 1000 to 3000.

After completing the recognition, run the "usb tree" command to view the recognition rate. The following is an example of connecting a USB host to a Hub and a storage device:

```
phobos_c906# usb tree
USB device tree:
1  Hub (480 Mb/s, 0mA)
|   U-Boot Root Hub
|
+-2  Mass Storage (480 Mb/s, 500mA)
      Generic USB3.0 Card Reader 000000001532
```

Initialization and application:

After completing the recognition you can enter the follow operations:

Step1: check the device information

- CLI execute: usb info [dev], You can view information about all devices on the controller. The following is an example.

```
phobos_c906# usb info 1
config for device 1
2: Mass Storage,  USB Revision 2.10
```

(continues on next page)

(continued from previous page)

```
- Generic USB3.0 Card Reader 000000001532
- Class: (from Interface) Mass Storage
- PacketSize: 64  Configurations: 1
- Vendor: 0x05e3  Product 0x0749 Version 21.50
  Configuration: 1
  - Interfaces: 1 Bus Powered 500mA
  Interface: 0
  - Alternate Setting 0, Endpoints: 2
  - Class Mass Storage, Transp. SCSI, Bulk only
  - Endpoint 1 In Bulk MaxPacket 512
  - Endpoint 2 Out Bulk MaxPacket 512
```

Step2: Read the pen drive

- Run: usb read addr blk# cnt, in command line to read the data with the starting address of blk and the size of cnt to the DDR address of addr, as shown in the following example:

```
phobos_c906# usb read 0x90000000 0 10

USB read: device 0 block # 0, count 16 ... 16 blocks read: OK
```

Step3: Write the pen drive

- Run: usb write addr blk# cnt, in command line to write the data with DDR address addr and size cnt to the location with the starting address blk of the storage device. The example is as follows:

```
phobos_c906# usb write 0x90000000 2000 2000
USB read: device 0 block # 8192, count 8192 ... 8192 blocks write: OK
```

3.3 Linux Host

3.3.1 USB 2.0 Host Operation Process

Step2: start the platform and load ext3 or squashfs. (or use the NFS)

Step3: load the relevant drivers

insmod usb-common.ko

insmod usbcore.ko

insmod udc-core.ko

insmod roles.ko

insmod dwc2.ko

step4: set USB role

echo host > /proc/cviusb/otg_role

3.3.2 USB Pen Drive Operation Process

Insert detection:

Insert the USB drive directly and observe whether the enumeration is successful. Normally, the UART is printed as:

```
[   72.061964] usb 1-1: new high-speed USB device number 2 using dwc2
[   72.315816] usb-storage 1-1:1.0: USB Mass Storage device detected
[   72.335934] scsi host0: usb-storage 1-1:1.0
[   73.363027] scsi 0:0:0:0: Direct-Access     Generic  STORAGE DEVICE   1532␣
↪PQ: 0 ANSI: 6
[   73.374407] sd 0:0:0:0: Attached scsi generic sg0 type 0
[   73.558597] sd 0:0:0:0: [sda] 30253056 512-byte logical blocks: (15.5 GB/14.
↪4 GiB)
[   73.566961] sd 0:0:0:0: [sda] Write Protect is off
[   73.571922] sd 0:0:0:0: [sda] Mode Sense: 21 00 00 00
[   73.577899] sd 0:0:0:0: [sda] Write cache: disabled, read cache: enabled,␣
↪doesn't support DPO or FUA
[   73.593961]  sda: sda1
[   73.602607] sd 0:0:0:0: [sda] Attached SCSI removable disk
```

sda1 represents the first partition on the USB drive or portable hard drive. When there are multiple partitions, the words sda1, sd2, sda3, etc. will appear

Initialization and application：

After inserting the storage device, perform the following operations:

In sdXY, X is the disk number and Y is the partition number. Please modify it according to the specific system environment。

- The device node for partition command operation is sdX, example: $fdisk /dev/sda.
- The specific partition formatted with mkdosfs tool is sdXY: ~$ mkdosfs -F 32 /dev/sda1。
- The specific partition of mount is sdXY: ~$ mount /dev/sda1 /mnt

1. View partition information
   - Run the command “ls /dev” to view the system device files, and if there is no partition information sdXY, there is no partition. Please partition the storage device with fdisk and go to step 2.
   - If there is partition information sdXY, the pen drive partition has been detected and entered step 2.。
2. View formatted information
   - If it is not formatted, use mkdosfs to format and then go to step 3.
   - If formatted, go to step 3
3. Mount the directory
   - Run the “mount /dev/sdaXY /mnt” mount directory.。

4. Read and write the storage device.

3.4 Linux Device

3.4.1 USB 2.0 Device Operation Process

Step1. Compile the kernel driver module of USB2.0 Device

- Enter the following path of menuconfig and configure it as follows.

```
Device Driver --->
 [*]   USB support --->
     <*>   USB Gadget Support  --->
         <M>   USB functions configurable through configfs
         [*]     Abstract Control Model (CDC ACM)
         [*]     Mass storage
```

- Compile kernel module and generate .ko file

Step 2: Start the platform, load ext4 or squashfs file system, or use NFS

Step 3: When the platform is used as a device, the USB2.0 device module must be loaded to be recognized as a USB device on the Host side. Please refer to "operation example" for specific operation.

All USB 2.0 device drivers are listed below.

3.4.2 Examples of storage device operations in USB device

Step4: As a Device, the platform supports both eMMC and SD storage media as follows:

Step5: Load below kernel modules.

insmod configfs.ko

insmod usb-common.ko

insmod udc-core.ko

insmod libcomposite.ko

insmod usbcore.ko

insmod roles.ko

insmod dwc2.ko

step6: The paths of USB Device related modules under kernels are:

drivers/usb/gadget/libcomposite.ko

drivers/usb/gadget/function/usb_f_mass_storage.ko

fs/configfs/configfs.ko

step7: switch otg controller to device mode

echo device > /proc/cviusb/otg_role

step8: run shell script "usb_usb.sh"

run_usb.sh probe msc /dev/mmcblkXY

run_usb.sh start

mmcblkXY is the Yth partition in eMMC or SD of the Xth disk. Please select it according to your specific situation.

Step9: The path of the USB Device related module under rootfs is: /etc/run_usb.sh

Step10: By connecting the platform to the Host side through USB, the platform can be recognized as a USB storage device on the Host side, and the corresponding device nodes can be generated in the / dev directory.

Step11: On the Host side, the platform can be treated as a common USB storage device, partitioning, formatting, reading and writing.

3.4.3 Example of Terminal Device Operation in USB device

The platform acts as a device as a terminal device by doing the following:

Setp1: insmod below kernel modules.

insmod configfs.ko

insmod libcomposite.ko

insmod u_serial.ko

insmod usb_f_acm.ko

insmod usb_f_serial.ko

The paths of USB Device related modules under kernels are:

- drivers/usb/gadget/libcomposite.ko
- drivers/usb/gadget/function/usb_f_serial.ko
- drivers/usb/gadget/function/usb_f_acm.ko
- drivers/usb/gadget/function/u_serial.ko
- fs/configfs/configfs.ko

  Switch otg controller to device mode

  echo device > /proc/cviusb/otg_role

  run script "run_usb.sh" run_usb.sh probe acm run_usb.sh start

  The path of the USB Device related module under rootfs is:

  /etc/run_usb.sh

Step2: By connecting the platform to the Host through USB, the platform can be recognized as a USB terminal device in the Host side, and the corresponding device node ttyACMX, X, is the same type of terminal device number, is generated in the /dev directory. ttyGSY is generated in the device side /dev directory, Y is the same type of terminal device number.

Host and Device can transmit data through the terminal device.

3.4.4 Example of RNDIS Device Operation in USB Device

The platform acts as an RNDIS device as follows

<<<<<<< HEAD Step3. Load below kernel modules. ======= Step1. insmod below kernel modules. >>>>>>> 8360b7f··· [fix](peripheral): Fix errors of Peripheral_Driver_Operation_Guide

insmod configfs.ko

insmod libcomposite.ko

insmod u_ether.ko

insmod usb_f_ecm.ko

insmod usb_f_eem.ko

insmod usb_f_rndis.ko

The paths of USB Device related modules under kernels are:

- drivers/usb/gadget/libcomposite.ko
- drivers/usb/gadget/function/usb_f_ecm.ko
- drivers/usb/gadget/function/usb_f_ecm.ko
- drivers/usb/gadget/function/usb_f_rndis.ko
- drivers/usb/gadget/function/u_ether.ko
- fs/configfs/configfs.ko

Switch otg controller to device mode

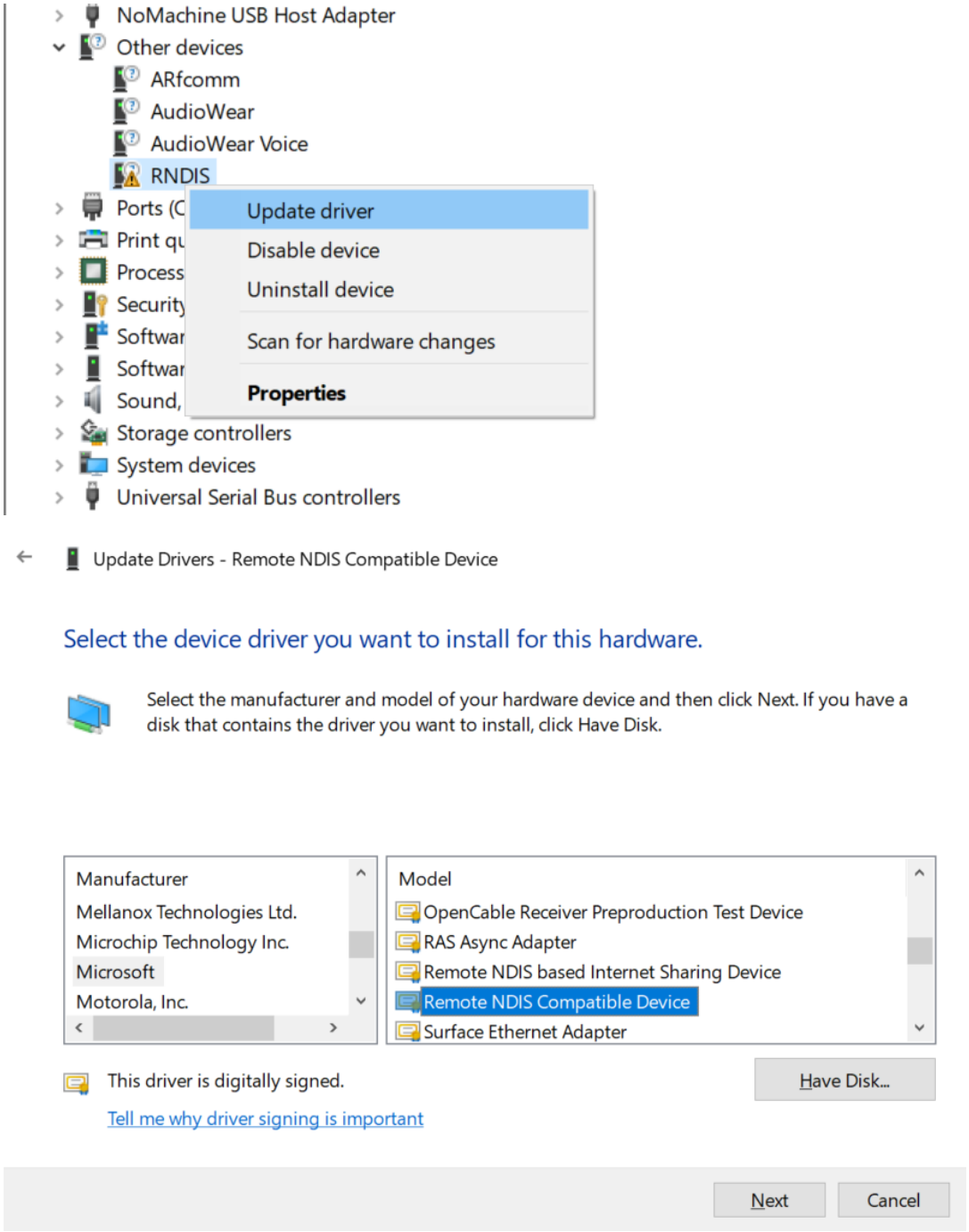
echo device > /proc/cviusb/otg_role

run script "run_usb.sh" run_usb.sh probe rndis run_usb.sh start

The path of the USB Device related module under rootfs is:

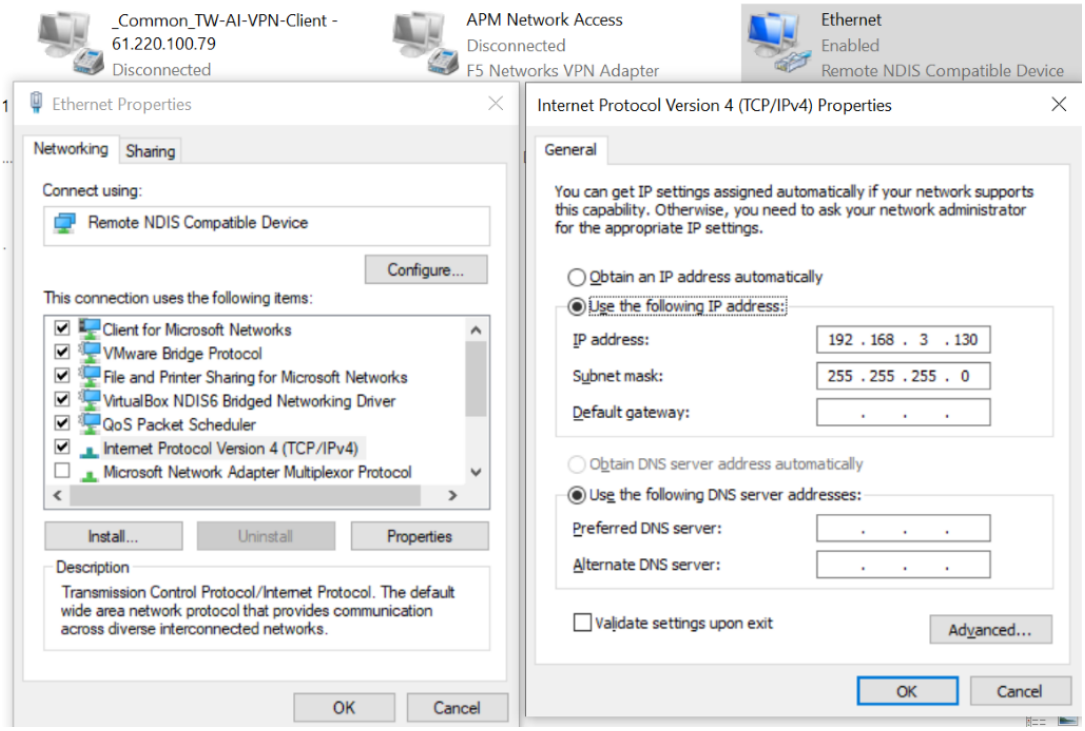
/etc/run_usb.sh

Step2: By connecting the platform to USB Host size via USB, you can recognize the platform as a USB Remote NDIS device on the Host side and install the Remote NDIS Compatible Device driver on Windows.



Step3: Set IP Address on Single Board, for example" ifconfig usb0 192.168.3.101 up"

Step4: Set the IP address on Windows.



Host and Device can transmit data through RNDIS devices.

3.4.5 Operation Example of CVITEK USB GADGET in USB device

The platform works as a Device using a custom CVTEK USB Gadget (CVG) as follows:

Step1: Insert below kernel modules

insmod configfs.ko

insmod libcomposite.ko

insmod usb_f_cvg.ko

The paths of USB Device related modules under kernels are:

- drivers/usb/gadget/libcomposite.ko
- drivers/usb/gadget/function/usb_f_cvg.ko
- fs/configfs/configfs.ko

Switch otg controller to device mode

echo device > /proc/cviusb/otg_role

run script "run_usb.sh"

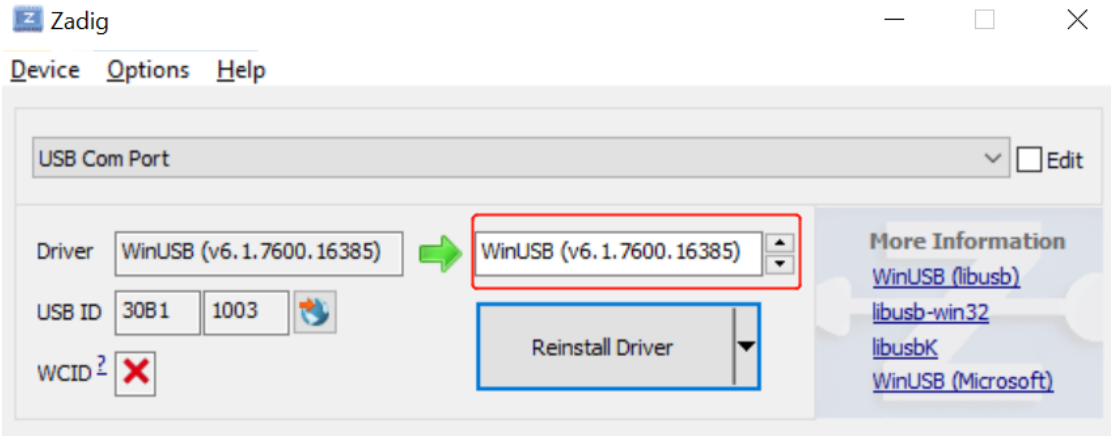
run_usb.sh probe cvg

run_usb.sh start

The path of the USB Device related module under rootfs is:

/etc/run_usb.sh

Step2: Connect the platform to the Host side via USB and use Zadig to install libusb (WinUSB) as the driver of the device.



Step3: Run test program sample_cvg [#TEST] on a single board.

Step4: Execute cvg/pctool/gen_patterns.sh on PC generates test Patterns. Execute " cvg/pctool/cvg_test.py" to start testing.

Step5: Refer to the CVITEK USB Gadget Usage Guide.docx for detailed usage files.

3.5 Points to Pay Attention to in Operation

- The following points should be noted in the operation: The system is preset to be Host mode after boot-up. To use Device mode, modules must be insmoded and USB ConfigFS scripts executed. Before switching to device mode, users must confirm the following:
  - The USB Cable is not connected to the Host.
  - The hardware on the platform has to switch to the corresponding USB mode. For example, before switching to Device mode, turn off the USB 5V power supply on the platform. If there is a Hub on the platform, turn off the Hub power and switch the path to Device mode connector
- After switching to Device mode, to use Host mode again, users must restart the platform
- When the platform is used as a terminal device, due to the TTY terminal characteristics, if a large amount of data is transmitted in a short time, it may cause data loss. Users should be aware of this limitation when using this feature.
- When reading a pen drive using USB Host under Uboot, be aware that if there is a Hub on the platform, you must turn on the Hub power and switch the path to the correct Connector.

4 SD/MMC card operation guide

4.1 Operation preparation

- Use U-boot and kernel in SDK
- File system:

  For SD/MMC cards, the SDK supports only the FAT file system, which can be read and written. After the kernel is started, mount it to the /mnt/sd directory or a directory as required.
- Partitioning can be done through the fdisk tool.
- Cv180x/cv181x SD supports 2.0 and 3.0:

At present, the cv180x/cv181x SD/MMC card supports only 3.3V VDDIO. Please Note that!

4.2 Operation Flow

1) By default, all SD/MMC driver modules have been compiled into the kernel. There is no need to run any additional loading commands.
2) Insert the card and power it on. You can view the card content by running fat commands in the U-boot.
3) When boot platform reaching the kernel, response nodes /dev/mmcblk0 and /dev/mmcblk0p1 are automatically scanned and identified.
4) In Uboot SD does not support hot plugging, but the kernel supports hot plugging. You can insert an SD card into the kernel to perform operations on the SD card. For details, see 3.3 Operation Examples.

4.3 Operation example

Examples of read and write operations for SD cards are as follows.

Initialization and application:

After the SD card is inserted, do the following (X below is the partition number whose value is determined by the fdisk tool when partitioning):

The specific directory for the specified fdisk operation is: ~ $ fdisk/dev/mmcblk0

Step 1. Check partition information

a. If p1 is not displayed, the SD card is not partitioned yet. Please partition with fdisk tool on Linux or format the SD card on Windows system before proceeding to Step 2.

b. If the partition information P1 is displayed, the SD card has been detected and partitioned and can be mounted in step 2.

Step 2. Mount

1. ~ $ mount /dev/mmcblk1pX /mnt/sd , This command mounts the Xth partition on the SD card to the /mnt/sd directory

4.4 Points to Pay Attention to in Operation

1. Make sure that the SD card has good connection with the slot hardware pin. If the connection is not good, there may be detection errors or error information related to read and write data errors, which may lead to read and write failures.
2. Each time an SD card is inserted, a mount operation is required to read and write the SD card. If the SD card is already mounted to the file system, you must do an umount operation before unplugging, otherwise you may not see the SD card partition after the next insertion of the SD card. In addition, unloading actions are also required for abnormal card unplugging.
3. You must ensure that the SD card has created a partition and formatted it as FAT or FAT32 file system (using the fdisk command under LINUX or the disk management tool under Windows).
4. Operation not allowed during normal operation:
   - Do not unplug the SD card when reading or writing it, otherwise some error message will be printed, and the file system in the card may be damaged.
   - If the current directory is under a mounted directory such as /mnt/sd, the unmount operation cannot be performed. You must leave the current directory such as /mnt/sd to perform the umount operation.
   - When there are reading or writing operation on mounted directories in the system, it cannot be umount until those operations have been completely ended. The task of operating mounted directories must be completely completed before umount can proceed properly.
   - When an exception occurs during the operation:

1. If the file system is damaged due to reading and writing data or other unknown reasons, there may be file system error messages when reading and writing SD cards. Umount, unplug, insert and mount the card again to read and write SD cards normally again.
2. Because the initialization of SD card takes some time for the detection/remove process, it is possible that no SD card can be detected if the card is inserted quickly after the card is unplugged.
3. If the card is unplugged abnormally during the test, the user needs to press ctrl+c to exit back to the kernel shell, otherwise the error message will be printed continuously.
4. When there is more than one partition on the SD card, you can switch between mounting partitions by mounting, but make sure that the number of mount operations is equal to the number of unmount operations to ensure that all mounted partitions are completely umount.

5 I2C Operation Guide

5.1 Operation Preparation

I2C is prepared for operation as follows:

- Use the kernel released by SDK。

5.2 Operation Process

- Load the kernel. The default I2C-related modules are all built into the kernel, and no install commands need to be executed.
- The I2C devices mounted on the I2C controller can be read and written by running the I2C read and write command under the console or by writing the I2C read and write program in kernel or user mode.

5.3 Interface Rate Setting Instructions

If you want to change the interface rate, you need to modify the "clock_frequency" of i2c node in build/boards/default/dts/cv180x/cv180x_base.dtsi or

build/boards/default/dts/cv181x/cv181x_base.dtsi, and recompile the kernel.

```
i2c0: i2c@04000000 {
    compatible = "snps,designware-i2c";
    clocks = <&clk CV180X_CLK_I2C>;
    reg = <0x0 0x04000000 0x0 0x1000>;
    clock-frequency = <400000>;

    #size-cells = <0x0>;
    #address-cells = <0x1>;
    resets = <&rst RST_I2C0>;
    reset-names = "i2c0";
};
```

5.3.1 Examples of I2C Read and Write Commands:

You can send relevant iic commands in linux terminal to detect bus devices and read or write the i2c devices in bus.

1. i2cdetect -l

   Detect iic buses in system. (It can be i2c-0, i2c-1, i2c-2, i2c-3, i2c-4 in cv180x)

2. i2cdetect -y -r N

   Detect all the address of devices which are connected with i2c-N bus. The example below is used detect devices which are connected with i2c-2 bus.

```
70: -- -- -- -- -- -- -- --
[root@cvitek]~# i2cdetect -y -r 2
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- 29 -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
[root@cvitek]~#
```

3. i2cdump -f -y N M

   View the values of all registers in the device with address M on i2c-N

4. i2cget -f -y 0 0x3c 0x00//

   Reads the value of register 0x00 on a device at address 0x3c on i2c-0

5. i2cset -f -y 0 0x3c 0x40 0x12//

   Write to register 0x40 on device at address 0x3c on i2c-0

5.3.2 I2C Read-Write Program Example with Kernel Mode:

This example demonstrates how to read and write I2C device in the kernel space.

Step 1. Assuming that the I2C device is known to be mounted on I2C controller 0, call i2c_get_adapter() function to get the I2C controller structure adapter:

```
adapter = i2c_ger_adapter(0);
```

Step 2. Through i2c_new_device() function to associate the I2C controller with the I2c device to obtain the client structure of the I2C device:

```
client = i2c_new_device(adapter, &info)
```

Note: The info structure provides the device address for i2c

Step 3. Call the standard read and write functions provided by the I2C core layer to read and write to device:

```
ret = i2c_master_send(client, buf, count);

ret = i2c_master_recv(client, buf, count);
```

Note: Client is the client structure obtained in step 2, buf is the register address and data to be read and written, count is the length of buf.

The code example is as follows:

```
// Announce a I2C device named "dummy" with device address 0x3c
static struct i2c_board_info info = {
      I2C_BOARD_INFO("dummy", 0x3C),
};
static struct i2c_client *client;

static int cvi_i2c_dev_init(void) {
// Assign I2C Controller Pointer
struct i2c_adapter *adapter;

adapter = i2c_get_adapter(0);
client = i2c_new_device(adapter, &info);
i2c_put_adapter(adapter);
return 0;
}

static int i2c_dev_write(char *buf, unsigned int count) {
int ret;

    ret = i2c_master_write(client, buf, count);
    return ret;
}

static int i2c_dev_read(char *buf, unsigned int count) {
int ret;

    ret = i2c_master_recv(client, buf, count);
    return ret;
}
```

5.3.3 I2C Read-Write Program Example with User Mode:

This operation example reads and writes I2C device through the I2C reader in the user space.

Step 1. Open the device file corresponding to the I2C bus and get the file descriptor:

```
i2c_file = open("/dev/i2c-0", O_RDWR);
   if (i2c_file < 0) {
      printf("open I2C device failed %d\n", errno);
```

(continues on next page)

(continued from previous page)

```
        return -ENODEV;
    }
```

Step 2. Read and write data:

```
ret = ioctl(file, I2C_RDWR, &packets);
    if (ret < 0) {
        perror("Unable to send data");
        return ret;
    }
```

Note: Read and write operations need to be specified on flags

```
struct i2c_msg messages[2];
    int ret;

    /*
     * In order to read a register, we first do a "dummy write" by writing
     * 0 bytes to the register we want to read from.  This is similar to
     * the packet in set_i2c_register, except it's 1 byte rather than 2.
     */
    outbuf = reg;
    messages[0].addr = addr;
    messages[0].flags = 0;
    messages[0].len = sizeof(outbuf);
    messages[0].buf = &outbuf;

    /* The data will get returned in this structure */
    messages[1].addr = addr;
    /* | I2C_M_NOSTART */
    messages[1].flags = I2C_M_RD;
    messages[1].len = sizeof(inbuf);
    messages[1].buf = &inbuf;
```

6 SPI Operation Guide

6.1 Operation Preparation

The preparation for operation of SPI is as follows:

- Kernel and file system using published SDK. File systems can use squashFS or ext4 published by the SDK. You can also mount to NFS over the network via the local file system

6.2 Operation Process

- Load the kernel. The default SPI-related modules are all built into the kernel and no install commands need to be executed.
- Running SPI read and write commands under the console or writing SPI read and write programs in kernel or user space which allows you to read and write SPI devices mounted on the SPI controller.

6.3 Operation Example

6.3.1 SPI Read-Write Program Example with Kernel Mode:

This operation example demonstrates how to read and write to SPI device through SPI reader and writer in kernel space.

Step 1. Call the SPI core-level function spi_busnum_to_master(), to get a description of the SPI controller architecture:

```
master = spi_busnum_to_master(bus_num);
// bus_num is the controller number of the SPI device for reading and writing
// master is the spi_master struct type pointer to describe the SPI controller.
```

Step 2. Call the SPI core layer function by the name of the spi device on the core layer to get the structure that is mounted on the SPI controller to describe the SPI device:

snprintf(str, sizeof(str), "%s.%u" , dev_name(&master->dev), cs);

dev = bus_find_device_by_name(&spi_bus_type, NULL, str);

spi = to_spi_device(dev);

//spi_buf_type is a bus_type structure type variable that describes the SPI bus

// spi is to describe the SPI peripheral spi_device structure type pointer

Step 3. Calling the SPI core layer function will spi_transfer added to spi_message queue. spi_message_init(&m)

spi_message_add_tail(&t, &m)

//t is spi_transfer structure type variable

//m isspi_message structure type variable

Step 4. Call the SPI Core Layer Read-Write function to read and write to device

status = spi_sync(spi, &m);

status = spi_async(spi, &m)

// spi is the spi_device structure type pointer to describe the SPI device

//spi_sync function is used to reads and writes SPI synchronously

//spi_async function is used to reads and writes SPI asynchronous

The code example is as follows: This code sample is for reference only, not for practical use.

```
// Incoming SPI controller bus number and processor number
static unsigned int busnum;
module_param(busnum, uint, 0);
MODULE_PARM_DESC(busnum, "SPI bus number (default=0)");

static unsigned int cs;
module_param(cs, uint, 0);
MODULE_PARM_DESC(cs, "SPI processor select (default=0)");

extern struct bus_type spi_bus_type;

// Declare the structure of the SPI controller static struct spi_master *master;

// Declare the structure of SPI peripherals
static struct spi_device *spi_device;

static int __init spidev_init(void) {
    char *spi_name;
    struct device *spi;

    master = spi_busnum_to_master(busnum);
    spi_name = kzalloc(strlen(dev_name(&master->dev)), GFP_KERNEL);

if (!spi_name)
```

(continues on next page)

(continued from previous page)

```
        return -ENOMEM;

snprintf(spi_name, sizeof(spi_name), "%s.%u", dev_name(&master->dev), cs);
    spi = bus_find_device_by_name(&spi_bus_type, NULL, spi_name);
    if (spi == NULL)
        return -EPERM;

    spi_device = to_spi_device(spi);
    if (spi_device ==NULL)
        return -EPERM;

    put_device(spi);
    kfree(spi_name);

    return 0;
}

int spi_dev_write(, void *buf, unsigned long len, int buswidth)
{
    struct spi_device *spi = spi_device;
struct spi_transfer t = {
      .speed_hz = 2000000,
      .tx_buf = buf,
          // buf needs to fill in device addr, register addr, write data and␣
↪other information according to peripheral device specifications
      .len = len,
   };
struct spi_message m;
spi->mode = SPI_MODE_0;

if (buswidth == 16)
      t.bits_per_word = 16;
   else
      t.bits_per_word = 8;

   if (!spi) {
      return - ENODEV;
   }

   spi_message_init(&m);
   spi_message_add_tail(&t, &m);
   return spi_sync(spi, &m);
}

int spi_dev_read(unsigned char devaddr, unsigned char reg_addr,  void *buf,␣
↪size_t len)
{
     struct spi_device *spi = spi_device;
```

(continues on next page)

(continued from previous page)

```
   int ret;
   u8 txbuf[4] = { 0, };
   struct spi_transfer t = {
      .speed_hz = 2000000,
      .rx_buf = buf,
      .len = len,
      };
   struct spi_message   m;
 spi->mode = SPI_MODE_0;

   if (!spi) {
      return -ENODEV;
   }
   txbuf[0] = devaddr;
     txbuf[1] = 0;
     txbuf[2] = reg_addr; //txbuf[1] &txbuf[2] Fill in 1 byte or 2 bytes␣
↪depending on the device bit width, this example is 2 bytes bit width
   t.tx_buf = txbuf;

   spi_message_init(&m);
   spi_message_add_tail(&t, &m);
   ret = spi_sync(spi, &m);

   return ret;
}
```

6.3.2 SPI Read-Write Program Example In User Space

This operation example reads and writes to SPI device mounted on SPI controller 0 in user space. (Specific implementations can refer to tools/spi/spidev_test.c)

Step 1: Open the device file corresponding to the SPI bus to get the file descriptor.

```
static const char *device = "/dev/spidev32766.0";
...
fd = open(device, O_RDWR);
if (fd < 0)
   pabort("can't open device");
```

Note: The default node for peripherals mounted on SPI Controller 1 is "dev/spidev32765.0"

The default node for peripherals mounted on SPI Controller 2 is "dev/spidev32764.0"

The default node for peripherals mounted on SPI Controller 3 is "dev/spidev32763.0"

Simply replace the node name and the rest will be the same as the device mounted on the SPI controller 0.

Step 2: Setting SPI transfer mode through ioctl:

```
/*
 * spi mode
 */
ret = ioctl(fd, SPI_IOC_WR_MODE32, &mode);
if (ret == -1)
   pabort("can't set spi mode");
ret = ioctl(fd, SPI_IOC_RD_MODE32, &mode);
if (ret == -1)
   pabort("can't get spi mode");
```

Note: Refer to the following figure or the kernel code for the model value configuration include/linux/spi/spi.h,

Ex. mode = SPI_MODE_3 | SPI_LSB_FIRST;

```
#define   SPI_CPHA    0x01          /* clock phase */
#define   SPI_CPOL    0x02          /* clock polarity */
#define   SPI_MODE_0    (0|0)           /* (original MicroWire) */
#define   SPI_MODE_1    (0|SPI_CPHA)
#define   SPI_MODE_2    (SPI_CPOL|0)
#define   SPI_MODE_3    (SPI_CPOL|SPI_CPHA)
```

Step 3: Setting SPI transmission bandwidth through ioctl:

```
/*
 * bits per word
 */
ret = ioctl(fd, SPI_IOC_WR_BITS_PER_WORD, &bits);
if (ret == -1)
   pabort("can't set bits per word");

ret = ioctl(fd, SPI_IOC_RD_BITS_PER_WORD, &bits);
if (ret == -1)
   pabort("can't get bits per word");
```

Step 4: Set SPI transfer speed through ioctl (generally recommended speed = 25M):

```
/*
 * max speed hz
 */
ret = ioctl(fd, SPI_IOC_WR_MAX_SPEED_HZ, &speed);
if (ret == -1)
   pabort("can't set max speed hz");

ret = ioctl(fd, SPI_IOC_RD_MAX_SPEED_HZ, &speed);
if (ret == -1)
   pabort("can't get max speed hz");
```

Step 5: Read and write data using ioctl:

```
ret = ioctl(fd, SPI_IOC_MESSAGE(1), &tr);
if (ret < 1)
   pabort("can't send spi message");
```

Note: tr transmits the first address of the spi_ioc structure array of a message.

7 GPIO Operation Guide

7.1 GPIO Preparation

- Use the kernel released by SDK.

7.2 Operation Process

- By default, the GPIO-related kernel modules are built into the kernel and no load commands need to be executed.
- GPIO can be input mode or output mode by executing GPIO read or write commands under the console or by calling GPIO APIs in kernel or user space APP.

7.3 Operation Example

7.3.1 GPIO Operation Command Example

Step 1: Use the echo command in the console to export GPIO number N for manipulation:

echo N > /sys/class/gpio/export

N Indicates the number of the GPIO to be operated. GPIO number = GPIO group number + offset value.

Taking GPIO1_2 pin in the schematic diagram as an example, GPIO1 corresponds to GPIO group number 448 and offset value 2.

So the GPIO number N is 448 + 2 = 450

The corresponding group number is as below:

GPIO0 corresponds to the linux group number 480

GPIO1 corresponds to the linux group number 448

GPIO2 corresponds to the linux group number 416

GPIO3 corresponds to the linux group number 384

PWR_GPIO corresponds to the linux group number 352

After echo N > /sys/class/gpio/export, generate the directory: /sys/class/gpio/gpioN

Step 2: Use the echo command in the console to set the GPIO direction:

for input: `echo in > /sys/class/gpio/gpioN/direction`

for output: `echo out > /sys/class/gpio/gpioN/direction`

For example:

set GPIO1_2 (the number is 450) to input mode:

```
echo in > /sys/class/gpio/gpio450/direction
```

set GPIO1_2 (the number is 450)to output mode:

```
echo out > /sys/class/gpio/gpio450/direction
```

Step 3: Use the cat commands in the console to read GPIO input values or use the echo commands to set GPIO output values :

check the input value:

```
cat /sys/class/gpio/gpioN/value
```

output low:

```
echo 0 > /sys/class/gpio/gpioN/value
```

output high:

```
echo 1 > /sys/class/gpio/gpioN/value
```

Step 4: After the resource is used, run the echo command on the console to release resources:

```
echo N > /sys/class/gpio/unexport
```

Note: You can enable the sysfs debug function of GPIO by turning on the CONFIG_DEBUG_FS option, and check the base number corresponding to GPIO PIN with the following command before operation:

cat /sys/kernel/debug/gpio

7.3.2 GPIO Operation Program Example with Kernel Space

GPIO Read-Write Operation program example in kernel space:

Step 1: Register GPIO:

```
gpio_request(gpio_num, NULL);
```

gpio_num is the GPIO number to be operated on, which is equal to "GPIO Group Number + Intra-Group Offset Number"

Step 2: Set GPIO direction:

```
for input: gpio_direction_input(gpio_num)
for output: gpio_direction_output(gpio_num, gpio_out_val)
```

Step 3: View GPIO input values or set GPIO output value:

```
check the input value: gpio_get_value(gpio_num);
output low: gpio_set_value(gpio_num, 0);
output high: gpio_set_value(gpio_num, 1);
```

Step 4: Release registered GPIO number:

```
gpio_free(gpio_num);
```

GPIO interrupt operation program example with kernel mode:

Step 1: Register GPIO:

```
gpio_request(gpio_num, NULL);

gpio_num is the GPIO number to be operated on, which is equal to "GPIO Group␣
↪Number + Intra-Group Offset Number"
```

Step 2: Set GPIO Direction:

```
gpio_direction_input(gpio_num);
```

For GPIO pins to be interrupt sources, the direction must be configured as input mode.

Step 3: The interrupt number corresponding to the GPIO number of the mapping operation:

```
irq_num = gpio_to_irq(gpio_num);
```

The interrupt number is the return value.

Step 4: Register interrupt:

```
request_irq(irq_num, gpio_dev_test_isr, irqflags, "gpio_dev_test", &
↪gpio_irq_type))
```

Irqflags is a type of interrupt that needs to be registered, and the common types are:

IRQF_SHARED : Sharing Interrupt;

IRQF_TRIGGER_RISING : Rising edge triggering;

IRQF_TRIGGER_FALLING : Drop edge trigger;

IRQF_TRIGGER_HIGH : High level trigger;

IRQF_TRIGGER_LOW : Low level trigger

Step 5: Release registered interrupts and GPIO numbers at end:

```
free_irq(gpio_to_irq(gpio_num), &gpio_irq_type);
gpio_free(gpio_num);
```

7.3.3 GPIO Operation Example with User Mode

GPIO Read-Write Operation program example with user space:

Step 1: Number the GPIO as expor for manipulation:

```
fp = fopen("/sys/class/gpio/export", "w");
fprintf(fp, "%d", gpio_num);
fclose(fp);
```

gpio_num is the GPIO number to be operated on, which is equal to "GPIO Group Number + Intra-Group Offset Number"

Step 2: Set GPIO direction:

```
fp = fopen("/sys/class/gpio/gpio%d/direction", "rb+");
for input: fprintf(fp, "in");
for output: fprintf(fp, "out");

fclose(fp);
```

Step 3: View GPIO input value or set GPIO output value:

```
fp = fopen("/sys/class/gpio/gpio%d/direction", "rb+");
check input: fread(buf, sizeof(char), sizeof(buf) - 1, fp);
output low:
strcpy(buf, "0" );
fwrite(buf, sizeof(char), sizeof(buf) - 1, fp);
output high:
strcpy(buf, "1" );
fwrite(buf, sizeof(char), sizeof(buf) - 1, fp);
```

Step 4: Number the GPIO manipulated as unexport:

```
fp = fopen("/sys/class/gpio/unexport", "w");
fprintf(fp, "%d", gpio_num);
fclose(fp);
```

8 UART operation guide

8.1 The Operation Preparation of UART Is as Follows

- Use the kernel released by SDK。

8.2 Module Compilation

- The source path is drivers/uart. When users need to access UART devices, they first need to specify the UART source path and header file path in the compilation script. After successful compilation, a library file named libuart.a is generated in the out directory. The library file needs to be specified with the -luart parameter when linking

8.3 Operation Example

Step 1:

Call the following interface in the initialization function to implement UART driver registration：

```
uart_dev_init();
```

If dma is enabled to receive data, dmac initialization is performed and called in the initialization function：

```
cvi_dmac_init();
```

Step 2:

Open the specified UART by calling open from the /dev/ttySN node

Step 3:

After opening UART, ioctl configuration read can be called to read data, write can send data, read can be blocked by select

Step 4:

When UART is not used, close is called to close, after which UART controller no longer receives data from the serial port.

8.4 Action Sample UART API Reference

- *uart_dev_init*: UART Device Initialization
- *uart_suspend*: UART Device Suspend
- *uart_resume*: UART Device Wake Up

8.4.1 uart_dev_init

【Description】

UART Device Initialization

【Syntax】

```
int uart_dev_init(void);
```

【参数】

Parameters	Description	Input / Output
None	None	None

8.4.2 uart_suspend

【Description】

UART Device Suspend

【Syntax】

```
int uart_suspend(void);
```

【参数】

Parameters	Description	Input / Output
Data	Reserved, passed in NULL	None

【Return Value】

Return Value	Description
0	Success
Others	Failure

8.4.3 uart_resume

【Description】

UART Initial wake-up of device。

【Syntax】

```
int uart_resume(void);
```

【参数】

Parameters	Description	Input / Output
Data	Not used, passing in NULL	None

【Return Value】

Return Value	Description
0	Success
Others	Failure

8.5 ioctl Configuration Instructions

【Description】

After turning on UART, configure UART baud rate, dma reception, blocking reads, wired control, etc. through ioctl.

For example, configure baud rate:

ret = ioctl(fd, CFG_BAUDRATE, 9600);

【Configuration Instructions】

Command Number	Command Code	Parameters	Description
UART_CFG_BAUDRATE	0x101	baud rate	Configure baud rate, UART0 default baud rate 115200; UART1, UART2, UART3 are 9600 Supports a maximum baud rate of 921600
UART_CFG_DMA_RX	0x102	0 or 1	0: Configured as interrupt receiving mode; 1: Configured DMA reception defaults to interrupt type
UART_CFG_DMA_TX	0x103	0 or 1	0: Configured as interrupt receiving mode; 1: Configured DMA reception defaults to interrupt type
UART_CFG_RD_BLOCK	0x104	0 or 1	0: Configure read as non-blocking mode; 1: Configure event blocking read The default is blocking mode;
UART_CFG_ATTR	0x105	0 or 1	Configure check bits, data bits, stop bits, FIFO, CTS/RTS, etc The default is: no check bit, 8 bits of data bit, 1 bit Stop bit, no CTS/RTS. Refer to the header file struct uart_attr
UART_CFG_PRIV	0x110	Custo mization	Drive custom commands

9 Watchdog Operation Guide

9.1 Preparations of Watchdog Are as Follow:

- Use the kernel released by SDK

9.2 Module Compile

- Insert module :cv180x: insmod cv180x_wdt.ko, cv181x: insmod cv181x_wdt.ko.
- To operate the Watchdog, run the Watchdog read and write command in the console or write the Watchdog read and write program in kernel space or user space.

9.2.1 Operation Example

Watchdog uses the standard linux framework to provide a hardware watchdog. Users can use watchdog simply by turning it on, off, or setting timeout. The system restarts when the watchdog timeout occurs.

The Watchdog is disabled by default. Customers can decide whether to enable it. The timeout times that can be specified are 1s, 2s, 5s, 10s, 21s, 42s, and 85s.

When the user input timtout time is 8s, the driver will select a timeout greater than or equal to the value of 10s; If timeout is not set, the driver uses 42s by default.

- Turn on WATCHDOG

Open the /dev/watchdog device node to start watchdog. You should ping (feed the dog) immediately after opening, otherwise wdt will restart immediately.

```
int wdt_fd = -1;
wdt_fd = open("/dev/watchdog", O_WRONLY);
if (wdt_fd == -1)
{
     // fail to open watchdog device
}
ioctl(fd, WDIOC_KEEPALIVE, 0);
```

- Turn off WATCHDOG

The driver supports "Magic Close". The magic character 'V' must be written to the watchdog device before closing the watchdog.

If the userspace daemon shuts down the device without sending 'V', the watchdog will keep counting.If the dog is not fed for a given period of time, it will result in a timeout and the system will restart.

The reference code is as follows:

```
int option = WDIOS_DISABLECARD;
ioctl(wdt_fd, WDIOC_SETOPTIONS, &option);
if (wdt_fd != -1)
{
   write(wdt_fd, "V", 1);
   close(wdt_fd);
   wdt_fd = -1;
}
```

- Set the TIMEOUT value

Set timeout with unit seconds by using the standard IOCTL command WDIOC_SETTIMEOUT. The timeout times that can be specified are 1s, 2s, 5s, 10s, 21s, 42s, and 85s.

```
#define  WATCHDOG_IOCTL_BASE 'W'
#define  WDIOC_SETTIMEOUT        _IOWR(WATCHDOG_IOCTL_BASE, 6, int)

int timeout = 10;
ioctl(wdt_fd, WDIOC_SETTIMEOUT, &timeout);
```

- PING watchdog

Ping watchdog through the standard IOCTL command, WDIOC_KEEPALIVE.

```
while (1) {
   ioctl(fd, WDIOC_KEEPALIVE, 0);
   sleep(1);
}
```

10 PWM Operation Guide

10.1 The Preparations for PWM Operation Are as Follow

- Use the kernel released by SDK

10.2 Operation Process

- Insert module :cv180x: Insmod cv180x_pwm.ko, cv181x: insmod cv181x_pwm.ko.
- Run PMW read/write command under console or write PWM read/write program in kernel space or user space to carry out input/output operation on PWM.
- PWM operation has 16 channels at 100MHz constant frequency clock, each channel can be controlled independently.
- Cv180X/CV181X has 4 PWM IP (pwmprocessor0/ pwmprocessor4/ pwmprocessor8/ pwmprocessor12), each IP controls 4 channels, can control 16 signals in total. The circuit diagram is represented by pwm0 to pwm15

  In Linux sysfs, the pwm0 to pwm3 device nodes are as follows:

  /sys/class/pwm/pwmprocessor0/pwm0~3

  In Linux sysfs, the pwm4 to pwm7 device nodes are listed as follows:

  /sys/class/pwm/pwmprocessor4/pwm0~3

  And so on

10.3 Operation Example

10.3.1 PWM Operation Commands Example

Step 1:

Use the echo command in the console to configure the PWM number to be operated, for example, PWM1:

```
echo 1 > /sys/class/pwm/pwmprocessor0/export
```

Step 2:

Set the duration of a PWM cycle (unit: ns):

```
echo 1000000 >/sys/class/pwm/pwmprocessor0/pwm1/period
```

Step 3:

Set the "ON" time of a cycle(unit: ns), namely the duty cycle= duty_cycle/period=50% :

```
echo 500000 >/sys/class/pwm/pwmprocessor0/pwm1/duty_cycle
```

Step 4:

Enable the PWM :

```
echo 1 >/sys/class/pwm/pwmprocessor0/pwm1/enable
```

10.3.2 An Example of a Program to Operate Through File IO

IO read-write operation program example with user space:

Step 1:

Configure the number of the PWM to be operated, for example, PWM1:

```
fd = open("/sys/class/pwm/pwmprocessor0/export", O_WRONLY);
if(fd < 0)
{
   dbmsg("open export error\n");
   return -1;
}
ret = write(fd, "1", strlen("0"));
if(ret < 0)
{
   dbmsg("Export pwm1 error\n");
   return -1;
}
```

Step 2:

Set the duration of a PWM cycle (unit: ns):

```
fd_period = open("/sys/class/pwm/pwmprocessor0/pwm1/period", O_RDWR);
ret = write(fd_period, "1000000" ,strlen("1000000" ));
if(ret < 0)
{
   dbmsg("Set period error\n");
   return -1;
}
```

Step 3:

Set the "ON" time of a cycle. (unit: ns) Duty cycle=50% for this example.

```
fd_duty = open("/sys/class/pwm/pwmprocessor0/pwm1/duty_cycle", O_
↪RDWR);
ret = write(fd_duty, "500000", strlen("500000"));
if(ret < 0)
{
   dbmsg("Set period error\n");
   return -1;
}
```

Step 4:

Enable PWM:

```
fd_enable = open("/sys/class/pwm/pwmprocessor0/pwm1/enable", O_RDWR);
ret = write(fd_enable, "1", strlen("1"));
if(ret < 0)
{
   dbmsg("enable pwm0 error\n");
   return -1;
}
```

11 ADC Operation Guide

11.1 The Preparations of ADC Operation Are as Follow

- Use the kernel released by SDK.

11.2 Operation Process

- Insert module: cv180x, insmod cv180x_saradc.ko, cv181x, insmod cv181x_saradc.ko.
- Run ADC read/write command under the console or write ADC read/write program in kernel space or user space to carry out input/output operation on ADC.
- The user layer accesses the IIO interface to implement trigger and sampling operations of three-channel and 12-bit ADC.
- 1.5v ref reference voltage.

11.3 Operation Example

11.3.1 ADC Operation Commands Example:

Step 1:

Specify ADC channels 1 to 6, in this example, ADC1:

(ADC channel 4 is dedicated to measuring VDDC_RTC; ADC channel 5 is PWR_GPIO1; ADC channel 6 is PWR_VBAT_

```
echo 1 > /sys/class/cvi-saradc/cvi-saradc0/device/cv_saradc
```

Step 2:

Read selected ADC channe

```
cat /sys/class/cvi-saradc/cvi-saradc0/device/cv_saradc
```

11.3.2 ADC Read-Write Operation Program Example with User Space

Step 1:

Configure the ADC channel number to be operated:

```
fd = open(/sys/class/cvi-saradc/cvi-saradc0/device/cv_saradc” , O_
↪RDWR|O_NOCTTY|O_NDELAY));
If (fd < 0)
   printf("open adc err!\n");
write(fd, “1” , 1);
```

Step 2:

Read the ADC values:

```
char buffer[512];
int len = 0;
int adc_value = 0;

len = read(fd, buffer, 10);
if (len != 0) {
    adc_value= atoi(buffer);
    printf("adc value is %d\n", adc_alue);
}
close(fd);
```