Fhostif Development Guide

1. Virtual Network Card Mode

1. Hardware device connection instructions

```
                          USB/SDIO                          WIFI                    WIFI/Network cable
Linux-host1 <---------> AIC-MCU <-------> route <----------> Linux-host2
     ^ ^                                         ^                                          ^
        | <-----------------> |
     v                  COM-DEBUG                                                           v
     <------------------------------------------------------------>
                                             ping/tcp/udp
```

In this mode, AIC-MCU is equivalent to the Ethernet card of Linux-host1. The device actually accessed in Linux-host1 is

A virtual Ethernet card device, which can be viewed through ifconfig

2. AIC-MCU low power consumption target description

Taking AIC8800MC as an example, there are multiple fhostif related subdirectories in the config/aic8800mc directory.

```
aic8800-sdk/config/aic8800mc/
 ÿÿÿ target_wifi_fhostif ÿÿÿ                              a---> Wi-Fi function virtual network card
 target_ble_wifi_fhostif b---> Compared to a, add ble function
 ÿÿÿ target_wifi_fhostif_ramopt ÿÿÿ                       c---> Compared to a, the wifi protocol stack runs in flash, providing more memory
 target_ble_wifi_fhostif_ramopt d---> Compared to b, run the wifi protocol stack in flash to provide more memory
```

---

3. AIC-MCU low power consumption-compile and start

AIC-MCU

AIC-MCU refers to ***AIC8800x. The current AIC8800x*** chip series includes ***AIC8800M, AIC8800MC, AIC8800M40***

Take target_wifi_fhostif as an example,

1. Open the file /**config/aic8800x/target_wifi_fhostif/tgt_cfg/tgt_cfg_wifi.h**
2. Define the virtual network card mode

```
/**
  * Hostif mode selection, match with host driver
  * Current support:
  *        1) HOST_VNET_MODE
  * 2) HOST_RAWDATA_MODE */

#define CONFIG_HOSTIF_MODE HOST_VNET_MODE
```

3. Enter the directory **/config/aic8800x/target_wifi_fhostif. If you need to use the Bluetooth function, enter /config/aic8800x/target_ble_wifi_fhostif**

4. Compile using the script **build_fhostif_wifi_case.sh**

```
#Compile the USB interface version, the
 default USB interface./build_fhostif_wifi_case.sh HOSTIF =usb -j8
#Compile the sdio interface
 version./build_fhostif_wifi_case.sh HOSTIF =sdio -j8
```

5. Burn **/build/host-wifi-aic8800/host_wb.bin** to **x8000000**

   (Or /build/host-wifi-aic8800mc/host_wb_aic8800mc.bin is burned to **x 8000000)**

6. Use **g 8000000** to execute the MCU program

Linux Hosting

The Linux host runs two parts of the program, the virtual network card driver **+ application**

The Linux host will register the network device during the driver loading process. The MAC address of the network device must be consistent with the MCU. When loading the driver, the MAC address is obtained and the network device is directly registered, which is called non-fast startup; when loading the driver, a random MAC address is used and **synchronization is performed after the driver is loaded, which is called fast startup.**

a. Load the network card driver

1. Enter the **/wifi/LinuxDriver/aic8800_netdrv** directory

2. Open the **Makefile** file and modify the configuration according to the actual environment of the host

```
#The default host platform is UBUNTU
CONFIG_PLATFORM_NANOPI_M4 ?= n
CONFIG_PLATFORM_ALLWINNER ?= n
CONFIG_PLATFORM_INGENIC_T31 ?= n
CONFIG_PLATFORM_INGENIC_T40 ?= n
CONFIG_PLATFORM_UBUNTU ?= y

# Driver mode support list #Default virtual
network card mode
CONFIG_VNET_MODE ?= y
CONFIG_RAWDATA_MODE ?= n

#Fast startup is disabled
by default CONFIG_FAST_INSMOD ?= n

#The interface uses USB by default
CONFIG_SDIO_SUPPORT = n
CONFIG_USB_SUPPORT = y
```

3. Compile and load the driver

```
make -j8
sudo insmod aic8800_netdrv.ko ifconfig vnet0
up (needed for T31/T40/T41)

#After installing the driver, enter ifconfig in the Linux-host1 terminal and you will see that the virtual network card vnet0 has been registered.
```

4. **Notes on driver loading**

```
### sodium
Before loading the sdio driver, you must first complete the sdio
card recognition process 1. sdio card recognition (clk, cmd, GND)
        AIC8800x connects to the main control sdio, and the main control kernel will start the card recognition process after loading. The card recognition process has nothing to
do with the driver. If the sdio interface is connected using flying wires or card slots, the card recognition may fail due to unequal wiring lengths. During the debugging process, a
small capacitor can be connected to the clk line and connected to GND. The size of the capacitor needs to be tested in practice. If the card recognition is successful, AIC8800x will get an
sdio device address assigned by the main control 2. Driver loading
        Similarly, if you use a flying wire or a slot connection, the sdio clock cannot be set too high. If it is set too high, it is easy to cause sdio data packet crc-
err. It is recommended to set the slot to about 20M and the flying wire to within 10M. In addition, after lowering the clock and loading the sdio driver, the main
control may successfully send a packet but fail to receive a packet. In this case, adjust the sdio phase.



### Modification Notes
// V2: AIC8800M/MC, V3: AIC8800M40

// V2 directly modify the following macros to
modify the clock and phase
# aicwf_sdio.h file #define SDIOWIFI_CLOCK_V2          20000000 // default 20MHz 20000000 //
#define SDIOWIFI_CLOCK_V3                             default 20MHz // 0: default, 2: 180°
#define SDIOWIFI_PHASE_V2                             0

// V3 needs to open the
following code #in aicwf_sdiov3_func_init interface
    u8 val = 0 ; // remove comments //
    compilation #if 1      open conditional
    if (host->ios.timing == MMC_TIMING_UHS_DDR50) {
        val = 0x21;//0x1D;//0x5; } else { val =

    0x01;//0x19;//0x1;
    }
    val |= SDIOCLK_FREE_RUNNING_BIT;
    sdio_f0_writeb(sdiodev->func, val, 0xF0, &ret); if (ret) {

        sdio_err("set iopad ctrl fail %d\n", ret); sdio_release_host(sdiodev-
        >func);
        return right;
    }
    sdio_f0_writeb(sdiodev->func, 0x0, 0xF8, &ret); if (ret) {

        sdio_err("set iopad delay2 fail %d\n", ret); sdio_release_host(sdiodev-
        >func);
        return right;
    }
    #Modify the value of register 0xF1, and set the five phases 0x10/0x20/0x40/0x80/0xF0
    sdio_f0_writeb(sdiodev->func, 0x20, 0xF1, &ret); if (ret) { sdio_err("set iopad
    delay1 fail
        %d\n", ret);
```

```
                sdio_release_host(sdiodev->func);
                return right;

        } msleep(1); #if
        1//SDIO CLOCK SETTING if (host-
        >ios.timing != MMC_TIMING_UHS_DDR50) { host->ios.clock =
                SDIOWIFI_CLOCK_V3; # aicwf_sdio.hÿÿÿ host->ops->set_ios(host, &host->ios);



        } #endif
        #endif
```

b. Start the virtual network card

Non-fast startup

1. Enter the **/wifi/LinuxDriver/app/custom_msg** directory and compile directly

   (If it is an embedded platform, you need to use the corresponding cross-compilation tool for compilation)

2. Connect to router AP

```
 ./custom_msg vnet0 1 ssid password ÿÿÿ ./

custom_msg vnet0 1 XIAOMI_HHH 1234567
```

3. Observe **the MCU-DEBUG** information through the serial port. After receiving the connection command, connect to the router and allocate **mcu-ip** + **mcu-gw.** You can observe the

   following settings through the MCU's serial port. They are

   all related to **mcu-ip** + **mcu-gw.**

4. Set the virtual network card ip and gw

```
sudo ifconfig vnet0 <mcu-ip> (ie: 192.168.3.36) sudo route add default gw
<mcu-gw> (ie: 192.168.3.1)
```

5. Set up network DNS

```
#Open the resolv.conf file sudo
chmod 777 /etc/resolv.conf vim /etc/resolv.conf


#Add the following content. Note that after the Linux-host1 system is restarted, the
file will restore the namespace mcu-gw (ie: 192.168.3.1)
```

6. Test network connectivity

```
#In practice, depending on the platform, if the network can be pinged through step 4, then step 5 does not need to be performed

#Route
ping 192.168.12.1

#LAN ping
192.168.12.10

#External
network ping 202.108.22.5
```

Quick Start

To reiterate: When the system is loading the driver, it does not obtain the mac address returned by the MCU, but uses the software random mac address to register vnet_dev. After the driver is loaded, the application is used to obtain the mac address of the MCU and synchronize it to the vnet_dev network device.

1. Modify **Makefile** settings

```
#Fast startup is disabled
by default CONFIG_FAST_INSMOD ? = y
```

2. After loading the driver, synchronize the MCU's mac address

```
#Get the mac address of the MCU and view the information
through dmesg custom_msg vnet0 5

#Synchronize the mac address
of vnet_dev sudo custom_msg vnet0 ndev mac_address
ie: sudo custom_msg vnet0 ndev 00:11:22:33:44:55

#Or , use the ifconfig command to modify the
mac address sudo ifconfig vnet0
down sudo ifconfig vnet0 hw ether 00:11:22:33:44:55 sudo ifconfig
vnet0 up
```

3. The rest of the operations are the same as **non-quick start**

c. Application

In the **/wifi/LinuxDriver/app/** directory, there are two applications: **custom_msg and fasync_demo**

custom_msg

Used to send instructions to the main control driver and send them to AIC-MCU through the SDIO/USB interface in the driver. custom_msg is used to send instructions to AIC-MCU through the SDIO/USB interface in the driver.

The interactive instructions between MCUs are as follows:

```
#After compiling the application, enter ./custom_msg and you will see the
command description "usage: custom_msg vnet0 [mode] <arg1> <arg2> <arg3>"
"---------------------"

">>>Interact with MCU:"
"custom_msg vnet0 1 ssid password "custom_msg          - connect ap" -
vnet0 2 "custom_msg vnet0                             disconnect ap" - close
3 "custom_msg vnet0 4                                 ble before sleep if used\r\n" - enter sleep" - exit sleep"
"custom_msg vnet0 5                                   - get mcu mac" - get
"custom_msg vnet0 6                                   wlan status"
"custom_msg vnet0 7
```

```
"custom_msg vnet0 8 ssid password band                    - start ap"
"                        -- band = <2.4G/5G>"
"custom_msg vnet0 9                                       - change ap mode\r\n" - stop ap"
"custom_msg vnet0 10                                      - scan wifi"
"custom_msg vnet0 11
"custom_msg vnet0 12 /your-path/update.bin - host ctrl OTA"
"------------------------"

">>>Interact with kernel:"
"sudo custom_msg vnet0 ndev mac_address                   - set vnet_dev mac"
```

Points to note

1. The **message blocking** driver can configure whether to block or not for the **custom_msg** message id

```
// handle_custom_msg function
case APP_CMD_CONNECT:
        /.../ // ÿÿ
        ÿÿ AICWF_CMD_WAITCFM
        cust_app_cmd.cmd_cfm.waitcfm = AICWF_CMD_WAITCFM; // ÿÿÿÿ
        CUST_CMD_CONNECT_IND
        cust_app_cmd.cmd_cfm.cfm_id = CUST_CMD_CONNECT_IND; ret =
        rwnx_tx_msg((u8 *)&cust_app_cmd.connect_req, sizeof(cust_app_cmd.connect_req),
&cust_app_cmd.cmd_cfm, command);
        break;

case APP_CMD_SCAN_WIFI:
        printk("APP_CMD_SCAN_WIFI\n");
        cust_app_cmd.common_req.cmd_id = CUST_CMD_SCAN_WIFI_REQ; // ÿÿÿ

        cust_app_cmd.cmd_cfm.waitcfm = AICWF_CMD_NOWAITCFM; // ÿÿÿÿÿ

        cust_app_cmd.cmd_cfm.cfm_id = 0; ret =
        rwnx_tx_msg((u8 *)&cust_app_cmd.common_req,
sizeof(cust_app_cmd.common_req), &cust_app_cmd.cmd_cfm, command);
        break;
```

2. The **timeout** driver sets the **custom_msg** message blocking time, which can be changed by the user.

```
// rwnx_main.h Note that if you need to return the networking result, the timeout should be set longer than the
time required for networking #define AICWF_CMDCFM_TIMEOUT2000
```

3. **Result echo** For blocked **custom_msg** messages, a feedback message will be returned. If not needed, just comment it out.

```
// custom_msg.c // If
don`t need return-result, it`s OK to comment out this printf. printf(APP_NAME "%s\n", (char *)&priv_cmd.buf[0]);
```

fasync_demo

By using the asynchronous message mechanism of the character device, the driver can actively send messages to the application layer. The fasync function is disabled by default. To enable this function, you need to modify the driver Makefile. After that, the character device for message sending will be registered during the driver loading process. After the driver is loaded, **fasync_user** needs to be run in the background to receive messages in real time.

```
# Msg Callback setting defaults to n
CONFIG_APP_FASYNC ?= y
```

Points to note

1. **The fasync_user program** should be started after the **aic8800_netdrv.ko driver is loaded and** before the custom_msg command is issued, and should be kept running in the background.

2. **Message writeback:** When the driver connection sends messages multiple times, the application layer may not have time to process the messages, which may cause message omissions. Therefore, the process of sending character device messages requires the application layer to set the status bit after receiving the message, and then the driver can send the next message. Users should pay attention to this when porting.

```
static void signal_handler(int signum) {

        int ret = 0; char
        data_buf[sizeof(struct rwnx_fasync_info)]; struct rwnx_fasync_info
        *fsy_info = (struct rwnx_fasync_info *)data_buf; ret = read(fd, data_buf, sizeof(struct rwnx_fasync_info));
        if(ret < 0) {

                printf(APP_NAME "Read kernel-data fail\n"); } else

        { printf(APP_NAME "%s\n", fsy_info->mem); // Set status to 0,
                indicating that the message has been
                received fsy_info->mem_status = 0;
                ret = write(fd, &fsy_info->mem_status, sizeof(fsy_info-
>mem_status));
                // Write back device
                buff if (ret < 0)
                        printf(APP_NAME "Write kernel-data fail\n");
                // Analyze kernel
                message analy_signal_msg(fsy_info->mem);
        }
}
```

If the fasync_user program is not started , or mem_status is not set in the program, the driver will wait for a timeout before sending the next message to the application layer.

```
# rwnx_main.h can adjust the timeout
#define FASYNC_APP_TIMEOUT                    100
```

3. **Automatic network configuration** When fasync_user parses the network information, it will automatically extract **the ip** and gw, and automatically configure the main

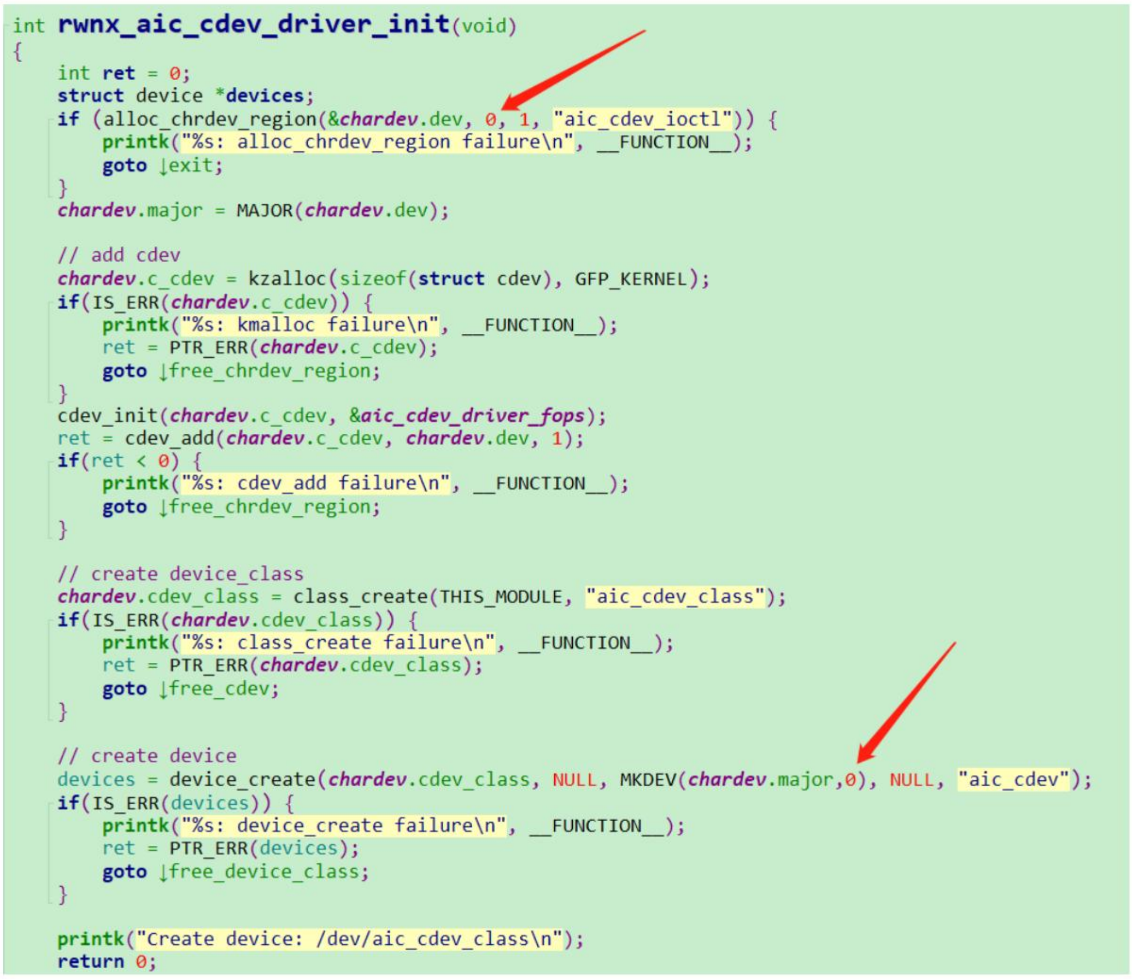
Monitor network status, users can parse messages and make corresponding configurations according to specific needs

```
void analy_signal_msg(char *mem) {

        char buff[64]; struct
        wlan_settings wlan; // auto wlan-
        settings char *ptr = strstr(mem,
        "3003"); if (ptr) { // set vnet0 ip ptr = strstr(mem,
        "ip");
               memcpy(buff, ptr+4,
                WLAN_SET_LEN); wlan.ip =
                str2uint(buff); sprintf(buff, "ifconfig vnet0 %d.%d.
                %d.%d",
                              (unsigned int)((wlan.ip >> 0 )&0xFF), (unsigned int)
((wlan.ip >> 8 )&0xFF),
                              (unsigned int)((wlan.ip >> 16)&0xFF), (unsigned int)
((wlan.ip >> 24)&0xFF));
                system(buff);

                // set vnet0 gw
                memcpy(buff, ptr+18, WLAN_SET_LEN);
                wlan.gw = str2uint(buff);
                sprintf(buff, "route add default gw %d.%d.%d.%d",
                              (unsigned int)((wlan.gw >> 0 )&0xFF), (unsigned int)
((wlan.gw >> 8 )&0xFF),
                              (unsigned int)((wlan.gw >> 16)&0xFF), (unsigned int)
((wlan.gw >> 24)&0xFF));
                system(buff);
                system("ifconfig");
        }
}
```

4. **Device number conflict** If the driver fails to load because CONFIG_APP_FASYNC is turned on, you can first confirm whether it is because of the character

   If the device number conflicts, modify it as follows

```
#Related
number ls -l/instructions to view the device file
dev cat /proc/devices to view the device
number ls /sys/class to view the character device class
```



```
int rwnx_aic_cdev_driver_init(void)
{
    int ret = 0;
    struct device *devices;
    if (alloc_chrdev_region(&chardev.dev, 0, 1, "aic_cdev_ioctl")) {
        printk("%s: alloc_chrdev_region failure\n", __FUNCTION__);
        goto ↓exit;
    }
    chardev.major = MAJOR(chardev.dev);

    // add cdev
    chardev.c_cdev = kzalloc(sizeof(struct cdev), GFP_KERNEL);
    if(IS_ERR(chardev.c_cdev)) {
        printk("%s: kmalloc failure\n", __FUNCTION__);
        ret = PTR_ERR(chardev.c_cdev);
        goto ↓free_chrdev_region;
    }
    cdev_init(chardev.c_cdev, &aic_cdev_driver_fops);
    ret = cdev_add(chardev.c_cdev, chardev.dev, 1);
    if(ret < 0) {
        printk("%s: cdev_add failure\n", __FUNCTION__);
        goto ↓free_chrdev_region;
    }

    // create device_class
    chardev.cdev_class = class_create(THIS_MODULE, "aic_cdev_class");
    if(IS_ERR(chardev.cdev_class)) {
        printk("%s: class_create failure\n", __FUNCTION__);
        ret = PTR_ERR(chardev.cdev_class);
        goto ↓free_cdev;
    }

    // create device
    devices = device_create(chardev.cdev_class, NULL, MKDEV(chardev.major,0), NULL, "aic_cdev");
    if(IS_ERR(devices)) {
        printk("%s: device_create failure\n", __FUNCTION__);
        ret = PTR_ERR(devices);
        goto ↓free_device_class;
    }

    printk("Create device: /dev/aic_cdev_class\n");
    return 0;
```

4. AIC-MCU low power consumption-verification process

1. Linux-Host1 automatically shuts down when no tasks are executed, and wakes up and resumes working state when tasks need to be executed. 2. After the AIC-MCU enters sleep mode, the current of the EVB board can be monitored in real time to measure the AIC-MCU sleep power consumption in the application. 3. The EVB board used to measure power consumption must disconnect external devices, use vbat power supply, and measure the core power supply to avoid high power consumption data.

The verification process can be referred to as follows

Please ***analyze it in combination with the aic8800_netdrv driver and fhostif_cmd.c application code in the SDK.***

1. Power on and run AIC-MCU and Linux-Host1. AIC-MCU can be switched to normal mode and will automatically execute after power on.

2. Linux-Host1 loads the driver

```
#Master control operation
sudo insmod aic8800_netdrv.ko
```

3. AIC-MCU network configuration currently supports three modes:

1) Network configuration through **AIC-MCU** serial port interactive commands

2) Send instructions to the driver through the Linux **application layer interactive program wifi/LinuxDriver/app/custom_msg** to control MCU network configuration

```
#Serial port
command connect 0 ssid passward #Master
control
 command ./custom_msg vnet0 1 ssid passward
```

3) Network configuration via Bluetooth (compile target with Bluetooth)

```
# fhostif_example.c do_blewifi_config Bluetooth network configuration routine 1. Start
through the serial port, ble_wifi_cfg 2. Modify the
routine to start the network configuration at the appropriate time
```

4. Linux-Host1 obtains MCU network configuration information and configures the network, and performs related tasks

5. Linux-Host1 is idle, and the application program sends a command to tell the MCU to enter sleep mode. At the same time, the main control driver will stop receiving upper-layer application signals. Data Pack

```
#Turn off Bluetooth. Bluetooth function will affect
WiFi sleep. If Bluetooth function is compiled, Bluetooth must be
 turned off before entering sleep./
custom_msg
 vnet0 3 #Master control command./custom_msg vnet0 4
```

6. Linux-Host1 uninstalls the driver and powers off

```
#Master control
operation sudo rmmod aic8800_netdrv
```

7. AIC-MCU receives the sleep command and sets the low power wake-up source

```
# MCU
operation # reflected in custom_msg_enter_sleep_handler
sleep_level_set(...); // Set sleep level user_sleep_wakesrc_set(...); // Set
wakeup source user_sleep_allow(1); // Allow sleep
```

8. AIC-MCU will execute **the host_if_poweroff** function to turn off the SDIO/USB interface

```
host_if_poweroff cannot be called in custom_msg_enter_sleep_handler, which may cause buff problems . In the routine, host_if_poweroff
is called in timer_handler

#Serial port command
# Note: During the test, you can close the interface on the MCU serial port, but this method cannot stop the main control driver from continuing to send data to the MCU interface.
(SDIO/USB) send data packet
hpof
```

9. AIC-MCU enters low power consumption and performs other related keep-alive operations

```
#Application development instructions in low power
state1 . After the MCU sleeps for a period of time, it wakes up regularly to perform
related tasks2 . If there is a user application that needs to be executed, the MCU will enter sleep mode after executing the related tasks
```

10. AIC-MCU is awakened by related events, exits low power consumption, and powers on the main control through the specified IO

```
# MCU
operation user_sleep_allow(0);                          // Do not allow sleep
sleep_level_set(PM_LEVEL_ACTIVE); // Set the activity level
# IO power-on, not reflected in the demo, customers can add it by themselves
```

11. AIC-MCU restores the interface with the host control, executes **the host_if_repower** function, and opens the SDIO/USB interface

```
#Serial port
command hpon
```

12. Linux-Host1 is powered on and the driver is reloaded

```
#Master control
operation sudo insmod aic8800_netdrv
```

13. Linux-Host1 obtains AIC-MCU network configuration information and configures the network, and re-executes the task

---

5. AIC-MCU low power consumption-status description

Define the four states of the AIC-MCU and main control interface based on whether the AIC-MCU is connected to the main control, whether the msg is connected, and whether the data is connected.

```
#Define the status of the AIC-MCU and Linux-Host1 interface
typedef enum hostif_status
{
        HOSTIF_ST_INIT          = 0, // host power off -> null msg, null data
        HOSTIF_ST_IDLE          = 1, // host power on -> tx/rx msg, null data
        HOSTIF_ST_AWAKE         = 2, // host power on -> tx/rx msg, tx/rx data
        HOSTIF_ST_DEEPSLEEP = 3, // host power off -> null msg, null data
} hostif_status_e;
```

state	describe
HOSTIF_ST_INIT	MCU-AIC is just powered on, not connected to the main control, data and msg are not available
HOSTIF_ST_IDLE	Connect to the main control, no network configuration, only msg communication
	set_hostif_wlan_status(HOSTIF_ST_IDLE);
HOSTIF_ST_AWAKE	Connect to the main control, network configured, data and msg are both available
	set_hostif_wlan_status(HOSTIF_ST_AWAKE);
HOSTIF_ST_DEEPSLEEP	Not connected to the main control, network configured, data and msg are not available
	set_hostif_wlan_status(HOSTIF_ST_DEEPSLEEP);

STA state transition

```
# INIT -> HOSTIF_ST_INIT
# IDLE -> HOSTIF_ST_IDLE
# AWAKE -> HOSTIF_ST_AWAKE
# SLEEP -> HOSTIF_ST_DEEPSLEEP

# MCU-AIC state transition 1: Do not start compiling the softAP function
                    ÿ   ----------3-----------                  ÿ --> Forbid this change!!
      ÿ   --------1------------                  ÿ
                                                 V                  V
        INIT --1--> IDLE --2--> AWAKE --3--> SLEEP
                           ^                 | ^ |
                           ÿ ---2---- ÿ          ÿ ---1---- ÿ
                           ^
                           ÿ ----------1----------- ÿ

#Transfer conditions description
1. custom_msg_get_wlan_status_handler
The main control must call this handler when loading the driver to obtain the AIC-MCU network configuration status

2. custom_msg_connect_status_ind_handler
    and custom_msg_disconnect_status_ind_handler
After connecting to the main control, AIC-MCU will call these two handlers when connecting to and disconnecting from the network.

3. custom_msg_enter_sleep_handler
After the master sends the sleep command, call this handler to enter sleep mode.
```

AP state transition

fhostif provides two modes of AP

model	**describe**	**HOSTIF_ST**
AIC_AP_MODE_CONFIG	SoftAP data flow is only processed locally in the MCU and is used for AP Distribution Network	HOSTIF_ST_IDLE
AIC_AP_MODE_DIRECT	The softAP data stream is connected to the main control and can be used for data transmission and reception	HOSTIF_ST_AWAKE

```
# INIT -> HOSTIF_ST_INIT
# IDLE -> HOSTIF_ST_IDLE
# AWAKE -> HOSTIF_ST_AWAKE
# SLEEP -> HOSTIF_ST_DEEPSLEEP

# MCU-AIC state transfer 2: Start compiling softAP function-AIC_AP_MODE_CONFIG
                        ÿ   -----------3-----------                    ÿ --> Forbid this change!!
         ÿ   ---1---- | ---------- ÿ 4/5 ÿ |          ÿ

                          | V                     V                  V
        INIT --1--> IDLE --2--> AWAKE --3--> SLEEP
                         ^ ^                        | ^ |
                          | ÿ ---2--- ÿ | ÿ ---1---- ÿ
                          | ^ | |
                          ÿ -----4----- ÿ |
                              ÿ ----------1----------- ÿ
#Transfer conditions description
1~3 Same as above
4. custom_msg_start_ap_handler
The master controller sends the command to start AP. If AIC-MCU is connected to a router at this moment, it will disconnect the router first and return to the IDLE state.

5. custom_msg_stop_ap_handler
The master controller sends a command to stop the AP

# MCU-AIC state transition 3: Start compiling softAP function-AIC_AP_MODE_DIRECT
                        ÿ   ------------3-----------                      ÿ --> Forbid this change!!
         ÿ   ---1---- | ------------                ÿ
                      ÿ 4/5 ÿ |
                          | V                        V                  V
        INIT --1--> IDLE --2/6--> AWAKE --3--> SLEEP
                         ^ ^                           | ^
                          | ÿ ---2/6--- ÿ | ÿ ---1---- ÿ
                          | ^ |
                          ÿ ------4------ ÿ
                              ÿ ------------1----------- ÿ
#Transfer conditions description
1~3 Same as above
4. custom_msg_start_ap_handler
The master controller sends the command to start AP. If AIC-MCU is connected to a router at this moment, it will disconnect the router first and return to the IDLE state.

5. custom_msg_stop_ap_handler
The master controller sends a command to stop the AP

6. custom_msg_change_ap_mode_handler
The master sends the command to switch to AP mode
```

Basic process of data analysis

Combined with the above AIC-MCU low power consumption-verification process, ignoring the filtered data packets, the general data packet parsing process

```
                                    The master controller issues instructions
|-Master control, MCU is powered on for the first time-| <------------> |-Master control is powered off, MCU is in sleep mode-| -->> Master control parses WIFI data packets
                                    MCU is awakened by an event
|---MCU enters sleep---| <-------------> |--MCU wakes up and exits low power consumption--| -->> MCU parses WIFI data packets
                                     The master controller issues instructions
|---The main control is powered on again---| <------------> |-The main control is powered off, and the MCU goes into sleep mode-| -->> The main control parses the WIFI data packet
```

The above process continues to loop until the system is shut down. For details, see **the set_hostif_wlan_status** interface call setting in the demo.

6. AIC-MCU low power consumption-data filtering

fhostif provides two packet filtering modes, VNET_FILTER_DIRECT and ***VNET_FILTER_SHARED. As described above for AIC-MCU low power consumption - status, only when HOSTIF_ST_STATUS*** is set to ***HOSTIF_ST_AWAKE, AIC-MCU will transparently transmit the data packet to the master for analysis. Setting*** filtering rules can keep the data packets that meet the conditions in ***AIC-MCU*** for processing.

Set filter mode

```
// In each fhostif-target directory, there is a file target_xxx_fhostif/tgt_cfg/tgt_cfg_wifi.h /** * Hostif rx pkt filter mode * Current suppoer: 1)

VNET_FILTER_DIRECT 2) VNET_FILTER_SHARED

  *
  *
  */
#define CONFIG_HOSTIF_FILTER_MODE VNET_FILTER_DIRECT
```

VNET_FILTER_DIRECT

This mode only judges based on the dst_port and protocol in the data packet to distinguish whether the data packet is sent to the master or processed locally

```
// set ip pkt filter filter.used = 1;
filter.protocol = 17; // UDP
filter.dst_port = 68; // DHCP client ret =
set_hostif_user_filter(&filter); if (ret) {


        dbg("failed to set ip pkt filter\n");
}
```

VNET_FILTER_SHARED

This mode provides a variety of filtering rules. When setting the filter, set the rules through filter_mask

Filter rules	mask
src_ipaddr	PACKET_FILTER_MASK_SRC_IP
src_ipaddr && src_port	PACKET_FILTER_MASK_SRC_IP | PACKET_FILTER_MASK_SRC_PORT
protocol && src_port	PACKET_FILTER_MASK_PROTOCOL | PACKET_FILTER_MASK_SRC_PORT
protocol && dst_port	PACKET_FILTER_MASK_PROTOCOL | PACKET_FILTER_MASK_DST_PORT
src_ipaddr, protocol && src_port	PACKET_FILTER_MASK_SRC_IP | PACKET_FILTER_MASK_PROTOCOL | PACKET_FILTER_MASK_SRC_PORT

```
// set ip pkt filter filter.used
= 1; filter.protocol =
17; // UDP filter.dst_port = 68; // DHCP
client filter.filter_mask =
PACKET_FILTER_MASK_PROTOCOL | PACKET_FILTER_MASK_DST_PORT; ret =
set_hostif_user_filter(&filter); if (ret) {

        dbg("failed to set ip pkt filter\n");
}
```

In addition, this mode has shared processing for **ping packets .** When AIC-MCU is connected to the network, both the main control and MCU can ping the external network. When the external network initiates a ping request, the AIC-MCU will respond.

7. AIC-MCU low power consumption-sleep description

AIC-MCU's low power sleep has different levels to choose from (common levels)

Chip Model	Sleep Level
AIC8800M40	PM_LEVEL_ACTIVE (not sleeping)
	PM_LEVEL_LIGHT_SLEEP
	PM_LEVEL_DEEP_SLEEP
AIC8800MC	PM_LEVEL_ACTIVE (not sleeping)
	PM_LEVEL_LIGHT_SLEEP
	PM_LEVEL_DEEP_SLEEP
AIC8800M	PM_LEVEL_ACTIVE (not sleeping)
	PM_LEVEL_LIGHT_SLEEP
	PM_LEVEL_DEEP_SLEEP
	PM_LEVEL_HIBERNATE

For support of wake-up IO, please refer to the document "AIC8800 Low Power Consumption"

8. AIC-MCU low power consumption-KeepAlive

After the host controller loads the driver, AIC-MCU sets the interface status with the host controller to **HOSTIF_ST_IDLE, and further according to** <5. AIC-MCU low power consumption - Status Description > Change interface status. However, when the host SDIO/USB interface is abnormally interrupted, usually in this unexpected situation, the AIC-MCU cannot automatically identify it and can only rely on the watchdog to reset it.

To this end, a **Keep-Alive** mechanism is added between the main control and AIC-MCU . The main control sends the specified msg data packet regularly. After receiving it, AIC-MCU replies and updates the heartbeat time. At the same time, AIC-MCU will start the timer to check whether the heartbeat time is updated in time. Otherwise, it is considered that the connection with the main control is disconnected, and then the interface status will be set to **HOSTIF_ST_INIT**

```
# fhostif_cmd.c

1. keep_alive is disabled by
default #define HOSTIF_KEEP_ALIVE 0

2. Heartbeat packet time
setting (ms) #define KEEP_ALIVE_PERIOD 250
```

9. AIC-MCU low power consumption-master DHCP

Normally, after the AIC-MCU connects to the AP, it will automatically obtain an IP address through DHCP. If the DHCP process needs to be performed by the master control, you need to first configure virtual IP information for the AIC-MCU. After the master control obtains the IP address, the master control will send the synchronized IP information to the AIC-MCU.

```
# fhostif_cmd.c

Master DHCP is disabled by default
#define HOSTIF_CNTRL_DHCP 0
```

10. AIC-MCU low power consumption-OTA automatic restart

Normally, the AIC-MCU will not automatically restart after an OTA upgrade, and the new software version will run only after restarting.

```
# fhostif_cmd.c

OTA automatic restart is disabled by default
#define HOST_OTA_REBOOT 0
```

Precautions

1. Normal system shutdown process,

```
# a. Linux-host1 uninstalls the driver
sudo rmmod aic8800_netdrv
# b. Power off Linux-host1
# c. MCU power off
```

2. After the system is running normally, ensure that **the AIC-MCU** is powered stably. If **the AIC-MCU** loses power unexpectedly during operation, **the SDIO/USB** connection will be disconnected.

   The connection is abnormal. **Linux-host1** may not be able to uninstall the driver normally and may need to be shut down forcefully.

2. Rawdata mode

Hardware device connection instructions

```
                USB/SDIO                WIFI            WIFI/Network cable
Linux-host1 <---------> AIC-MCU <-------> route <----------> Linux-host2
     ^                          ^ ^                                      ^
     |          COM-DEBUG       | |          ping/tcp/udp                |

     v <----------------> vv <---------------------------- ---> v
```

In this mode, AIC-MCU is equivalent to the USB/SDIO device of Linux-host1 and can be used for large amount of data transfer.

Compile and start

AIC-MCU

1. **Change tgt_cfg_wifi.h mode configuration**

```
/**
  * Hostif mode selection, match with host driver
  * Current support:
  *        1) HOST_VNET_MODE
  * 2) HOST_RAWDATA_MODE */

#define CONFIG_HOSTIF_MODE HOST_RAWDATA_MODE
```

2. The remaining steps are the same as the above virtual network card

Linux Hosting

a. Linux driver

1. Enter the **/wifi/LinuxDriver/aic8800_netdrv** directory

2. Configure **Makefile** related macros

```
# Driver mode support list
CONFIG_VNET_MODE ?= n
CONFIG_RAWDATA_MODE ?= y
```

3. Compile and load the driver

```
make -j8
sudo insmod aic8800_netdrv.ko
```

b. Application

1. Enter the /**wifi/LinuxDriver/app/nlaic_demo** directory

2. Compile and run

```
make -j8
#Permissions must be added to run the application
# demo-1
sudo ./nlaic_user 0 <char msg> <uint data> <bytes byte_ptr> # demo-2

sudo ./nlaic_user 1 # demo-3

sudo ./nlaic_user 2 <uint block_size> <uint block_count>
```
