SCM1612
Wi-Fi 6 and BLE 5 Low-Power SoC

ScmChannel User Guide

Version 1.1
Date 2023-08-15

Contact Information
Senscomm Semiconductor (www.senscomm.com)
Room 303, International Building, West 2 Suzhou Avenue,
SIP, Suzhou, China
For sales or technical support, please send email to
info@senscomm.com

Disclaimer and Notice

Version History

Version	Date	Description
1.2	2024-04-22	Update APIs
1.1	2023-08-15	Format Modification
1.0	2023-08-04	Version 1.0 Release
0.1	2023-07-11	Initial Draft

Table of Contents

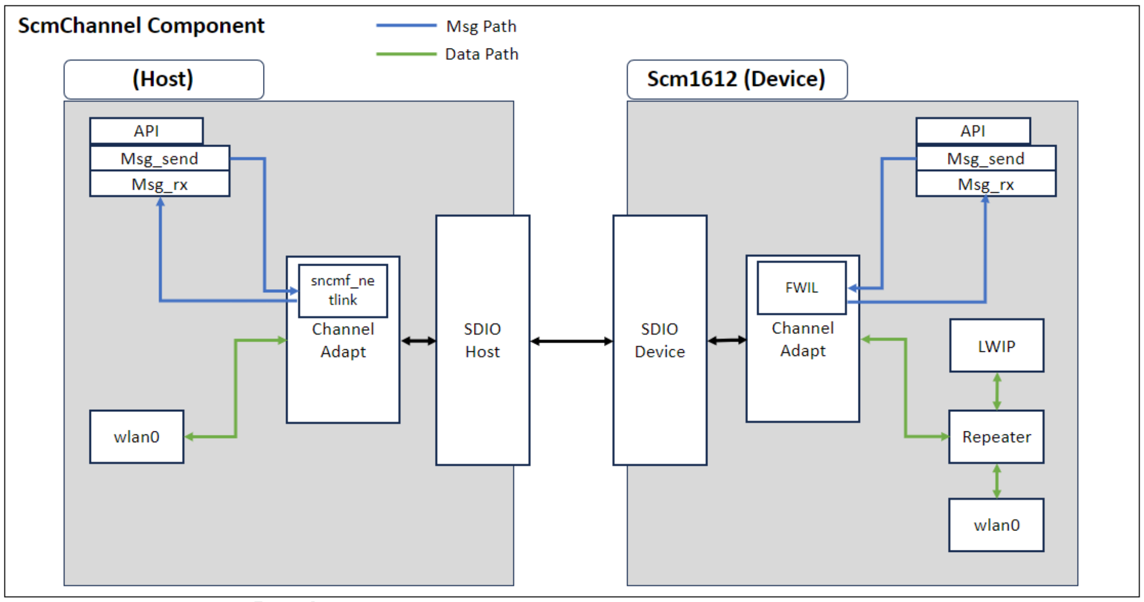
1 Introduction

The ScmChannel component primarily offers a low-power host-device solution. Its main function is to establish a communication channel between a host chip (e.g., Camera) and a Wi-Fi chip (device), which consists of two sub-channels:

- Msg Channel: Used for transmitting and receiving user-defined messages.
- Data Channel: Used for transmitting and receiving network data packets.

The software architecture of ScmChannel is illustrated in Figure 1-1.

Figure **1-1 ScmChannel** basic framework



Component Explanation:

- Wlan0: Both the host and device sides have a wlan0 network interface, indistinguishable from a regular network interface, primarily used for receiving/sending network data packets.
- Channel Adapt: Utilizes host-side sncmf_netlink and device-side FWIL module to encapsulate/strip the channel communication protocol.
- Repeater: The repeater can distinguish received network data packets and forward them to the host or device side, or forward them to both sides.

2 ScmChannel Usage

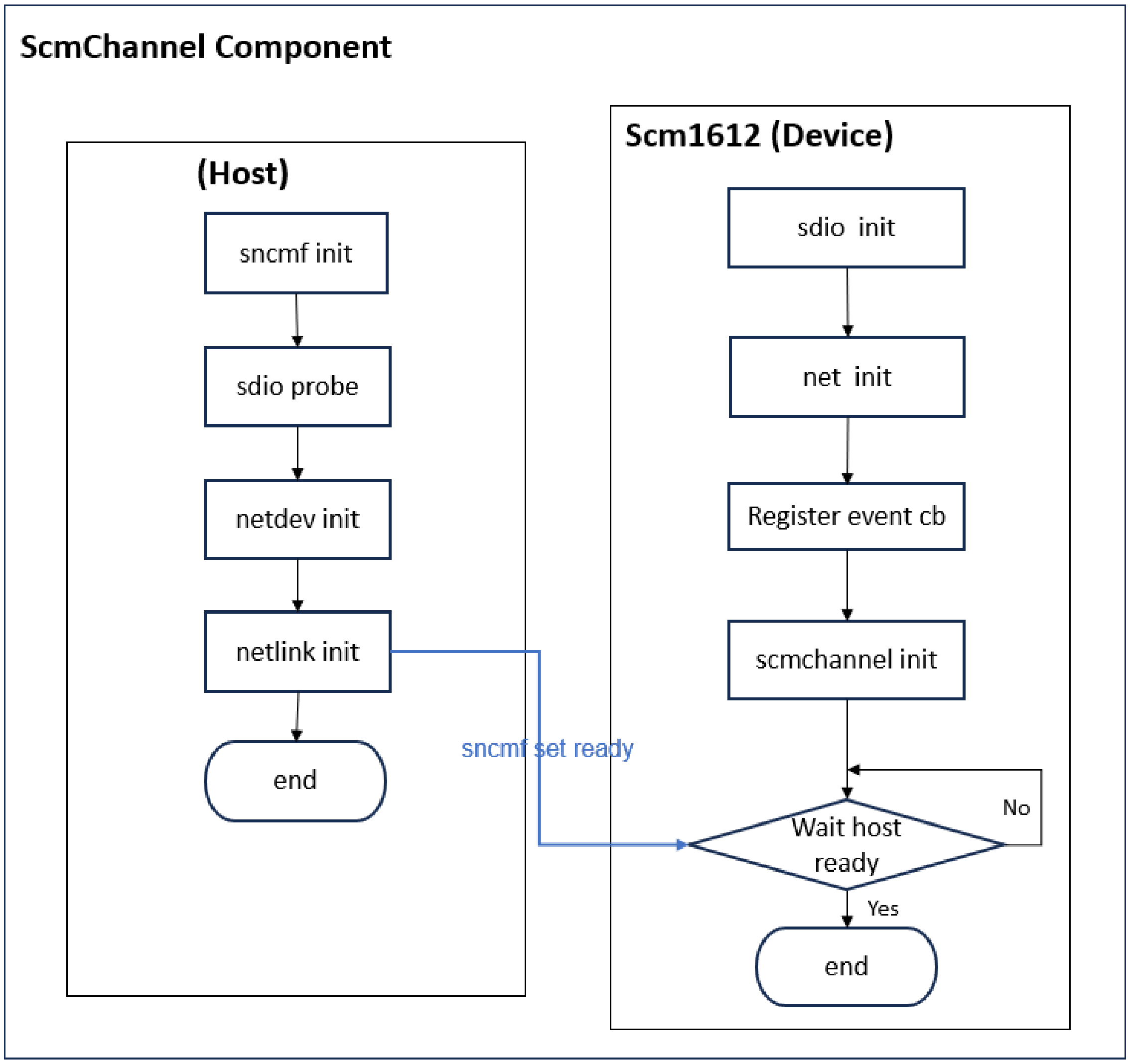
To facilitate users in quickly integrating the ScmChannel component with their own business, this chapter provides an overview of the basic usage rules for ScmChannel. It primarily covers the following aspects:

2.1 ScmChannel Initialization

The purpose of initializing the ScmChannel component is to establish a communication channel between the host chip and the Wi-Fi (device) chip, ensuring that both can engage in network data communication and user-defined message communication.

The initialization process of the ScmChannel component is illustrated in Figure 2-1.

Figure 2-1 ScmChannel component initialization



- On the Host:
  - ➢ sncmf init: Initializes the entire sncmf module.
  - ➢ sdio init: Initializes sdio-related resources.
  - ➢ netdev init: Initializes network nodes for receiving and transmitting network data packets.
  - ➢ netlink init: Initializes netlink, primarily used for establishing communication between the application layer and the kernel. Upon completion of initialization, a message "sncmf set ready" will be sent to SCM1612.
- On the device:

- sdio init: Configures the sdio device into detectable mode, allowing the sdio host device to establish a connection. This process is automatically executed during device startup.

- register event cb: Establishes callback functions for network node attribute changes and activates the ScmChannel message transmission functionality.

- ScmChannel init: Mainly sets up the Repeater module and registers callback functions on the SCM1612 side (Msg rx cb).
  - Set up Repeater: This module controls network data packet forwarding rules. For detailed filtering rules, refer to section 2.3 on Repeater forwarding rule development.
  - Register Callbacks: Registers a callback function for receiving messages on the SCM1612 side, primarily used for user-specific business development.

- wait host ready: The SCM1612 side will receive a "host ready" notification, indicating that the ScmChannel channel has been established.

ScmChannel Initialization Process Demo:

Step 1: Register a callback function for network node attribute changes and activate ScmChannel message transmission using the API `scm_wifi_register_event_callback`.

Step 2: Initialize scmchannel using the API `scm_channel_init`.

Step 3: Reset Repeater rules using the API `scm_vlwip_netif_reset`.

Step 4: Configure Repeater module forwarding rules using the Demo code function `scm_channel_set_default_filter`.

Step 5: Register a callback function for receiving messages on the SCM1612 side using the API `scm_channel_register_rx_cb`.

--- End

```
int scm_channel_init(void)
{

        if (scm_vlwip_netif_reset(WIFI_FILTER_TYPE_IPV4) != WISE_OK) {
                printf("%s: netif reset failed\n", __func__);
                return WISE_FAIL;
        }

        if (scm_channel_set_default_filter() != WISE_OK) {
                printf("%s: set_default_filter failed\n", __func__);
                return WISE_FAIL;
        }

        scm_channel_dump_filter();
        scm_channel_register_rx_cb(scm_channel_rx_callback);

        printf("ScmChannel    init OK\n");
        return WISE_OK;
}

int main(void)
{
        int ret = WISE_OK;
        char ifname[WIFI_IFNAME_MAX_SIZE + 1] = {0};
        int len = sizeof(ifname);

        scm_wifi_register_event_callback(event_handler, NULL);

        scm_wifi_sta_start(ifname, &len);

#ifdef CONFIG_API_SCMCHANNEL
        scm_channel_init();
#endif
        ret = scm_wifi_start_connect();

        return ret;
}
```

2.2 Sleep and Wake-up Configuration (TBD)

2.3 Repeater Forwarding Rule Development

The Repeater module within the ScmChannel component provides network forwarding capabilities. Specific network packet forwarding directions can be achieved by making calls to Repeater module APIs. On the host side, the Repeater module on the device side can be employed to filter the required network packets, allowing undesired packets to be managed on the device side.

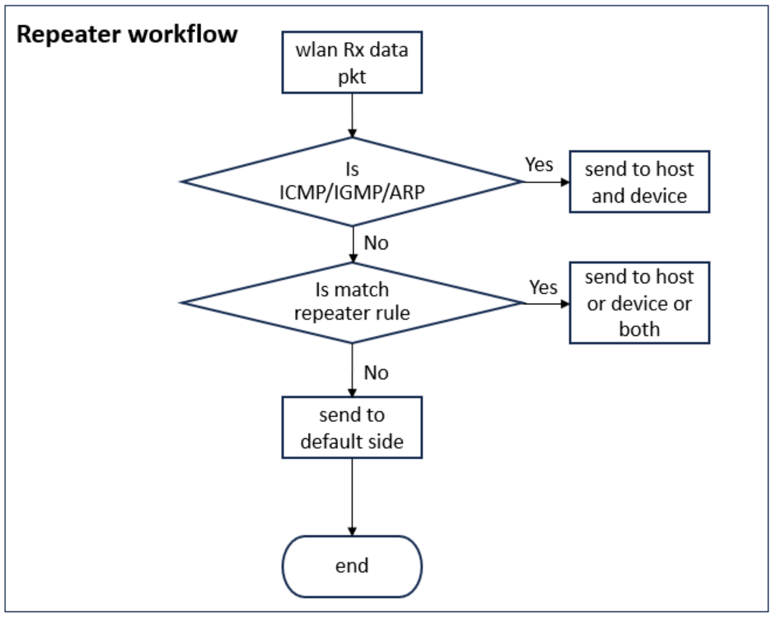
Due to the shared network configuration (MAC address, IP address, netmask, gateway) between the host and device, once the ScmChannel component is successfully established, users can directly utilize the device-side network interface on the host side to transmit network packets to external networks.

When sending outbound network packets, there is no mutual impact between the host and device. Both entities use the device to send packets externally, and this sending process does not involve any forwarding rules.

In terms of network packet reception, when the device side receives a network packet, it will apply the Repeater rules to determine whether it should be forwarded to the host or remain within the device.

The forwarding process on the device side upon receiving a packet is depicted in Figure 2-2:

Figure **2-2 Repeater Forwarding Process Flow:**



- ICMP, IGMP, and ARP packets from external sources are automatically forwarded to both the host and device without requiring specific forwarding rules. This is preset in the Repeater functionality.

- Upon receiving external packets that do not match any forwarding rules, they are forwarded according to default rules (By default, ScmChannel is set to forward to the host side, and users can modify this using the API `scm_wifi_set_default_filter`).

- When an external packet is received and matches a forwarding rule (based on IP, protocol type, port, or port range), it is forwarded to the corresponding destination (host, device, or both sides) as configured.

During Repeater initialization, specific forwarding rules are established based on packet attributes. Taking IPV4 as an example, the method to configure forwarding rules is as follows:

Users can modify the variable `ipv4_filter_def_setting[]` based on their business requirements and packet attributes. They can fill in one or multiple sets of filter array elements as needed. Using the `scm_wifi_add_filter` function will add each set of filters to the Repeater module. It's important to note that the default maximum limit for setting filters is 15 sets. If more sets are required, the `config` parameter can be adjusted accordingly.

```
static struct wifi_ipv4_filter ipv4_filter_def_setting[] = {
        /* UDP */
        {
                0,          /*  未设定  */
                7002,       /*  设置对端端口号，即报文携带的目的端口号需要为 7002 */
                0,          /*  未设定  */
                0,          /*  未设定  */
                0,          /*  未设定  */
                0,          /*  未设定  */
                0,          /*  未设定  */
                17,         /*  设置报文类型需要为 UDP 报文， UDP 为 17， TCP 为 6 */
                WIFI_FILTER_TO_LWIP,         /*  配置匹配本转发规则的报文转发到 device 侧  */
                WIFI_FILTER_MASK_REMOTE_PORT | WIFI_FILTER_MASK_PROTOCOL,
                /*  掩码字段配置需要同时匹配对端端口号、协议类型  */
        },
};
```

The struct wifi_ipv4_filter structure is defined as follows:

```
/* Data Filter Structure */
struct wifi_ipv4_filter {
        unsigned int remote_ip; /*  选设字段, 指定接收报文携带的对端 IP 地址。即报文中的源 IP 地址*/

        unsigned short local_port; /*  选设字段，指定接收报文对应的本端端口号。由于设备是接收方，即为对应报文携带
的目的端口号*/

        unsigned short localp_min; /*  选设字段，指定接收报文对应的本端端口范围最小值。由于设备是接收方，即为对应
        报文的目的端口
                                        范围最小值*/

        unsigned short localp_max; /*  选设字段，指定接收报文对应的本端端口范围最大值。由于设备是接收方，即为对应
报文的目的端口号
                                        范围最大值，最大值需大于最小值*/

        unsigned short remote_port; /*  选设字段，指定接收报文对应的对端端口号。由于设备是接收方，即为对应报文携
带的源端口号*/

        unsigned short remotep_min; /*  选设字段，指定接收报文对应的对端端口范围最小值。由于设备是接收方，即为对
应报文的源端口号
                                        范围最小值*/

        unsigned short remotep_max; /*  选设字段，指定接收报文对应的对端端口范围最大值。由于设备是接收方，即为对
应报文的源端口号
                                        范围最大值，最大值需大于最小值*/

        unsigned char packet_type; /*  选设字段，指定接收报文采用的传输层协议，一般指定为 TCP（6）或者 UDP（17）
*/

        unsigned char config_type; /*  必设字段，设置匹配本转发规则的报文的转发方向，  设置为 WIFI_FILTER_LWIP 转
发到 device 侧，设
                                        置为 WIFI_FILTER_VLWIP 转发到主控侧，  WIFI_FILTER_BOTH 则两边都转发*/

        unsigned char match_mask; /*  必设字段，指定上述哪些选设字段是有效值，每个选设字段通过对应 Bit 掩码置位。
置位之后代表需要
                                        同时满足匹配该字段，例如  WIFI_FILTER_MASK_IP |
                                        WIFI_FILTER_MASK_PROTOCOL  值为 0x1 | 0x2 = 0x3,表示报文需要同时满足匹配
                                        源 IP 地址和协议类型（TCP 或 UDP）才匹配本规则，匹配之后按照 config_type 指
                                        定的方向转发本报文  */
        unsigned char    resv;    /*  保留字段  */
};
```

In the above `wifi_ipv4_filter` structure, the `match_mask` field is defined by combining values from the mask enumeration. For instance, `"WIFI_FILTER_MASK_REMOTE_PORT | WIFI_FILTER_MASK_PROTOCOL"` indicates that a packet needs to simultaneously match the specified IP address and the remote port number to satisfy this rule.

Regarding the comparison rules for the source port and destination port defined in the `wifi_filter_field_enum`:

- If the source port number does not match, then it will further match against the specified source port number range.

- If the source port number matches, it is considered that the source port number range has also been matched.

Similarly, for scenarios where both the destination port number and the destination port number range need to be matched in the packet:

```
/* Data Filter Item */
typedef enum {
        WIFI_FILTER_MASK_IP = 0x01, /*  掩码枚举：表示源 IP 地址，对应 remote_ip 字段  */

        WIFI_FILTER_MASK_PROTOCOL = 0x02, /*掩码枚举：协议类型（  TCP 或 UDP)，对应 packet_type 字段，不指
                                                 则表示该字段不作为匹配条件  */

        WIFI_FILTER_MASK_LOCAL_PORT = 0x04, /*  掩码枚举：接收到报文的目的端口号，对应 local_port 字段，不指
                                                 定则表示该字段不作为匹配条件  */

        WIFI_FILTER_MASK_LOCAL_PORT_RANGE = 0x08, /*  掩码枚举：接收到报文的目的端口号范围，对应
                                                 localp_min 和 localp_max 字段，不指定则表示该字段不作为
                                                 配条件  */

        WIFI_FILTER_MASK_REMOTE_PORT = 0x10, /*  掩码枚举：接收到报文的源端口号，对应 remote_port 字段，不指
                                                 定则表示该字段不作为匹配条件*/

        WIFI_FILTER_MASK_REMOTE_PORT_RANGE = 0x20, /*掩码枚举：接收到报文的源端口号范围，对应
                                  remotep_min 和 remotep_max 字段，不指定则表示该字段不作为匹配条  */

        WIFI_FILTER_MASK_BUTT
} wifi_filter_field_enum;
```

In the example application, we have predefined that DHCP packets are sent to the lwip side, and DHCP information is updated to the host side through the network event change interface. If the server provides a dedicated port for interaction with the device side, the remote port number can be utilized as a forwarding rule identifier to forward packets from that server to the device side. Users can utilize Repeater rules to achieve appropriate task allocation to the host or device side, thus achieving power-saving effects.

The DHCP forwarding rule configuration is as follows:

```
static struct wifi_ipv4_filter ipv4_filter_def_setting[] = {
        /* DHCP */
        {
                0,                                                          /* remote ip */
                68,                                                         /* local port */
                0,                                                          /* localp_min */
                0,                                                          /* localp_max */
                0,                                                          /* remote_port */
                0,                                                          /* remotep_min */
                0,                                                          /* remotep_max */
                17,                                                         /* packet type */
                WIFI_FILTER_TO_LWIP,                                        /* config_type */
                WIFI_FILTER_MASK_LOCAL_PORT | WIFI_FILTER_MASK_PROTOCOL, /* match_mask */
        },
        /* DHCP */
        {
                0,                                                          /* remote ip */
                67,                                                         /* local port */
                0,                                                          /* localp_min */
                0,                                                          /* localp_max */
                0,                                                          /* remote_port */
                0,                                                          /* remotep_min */
                0,                                                          /* remotep_max */
                17,                                                         /* packet type */
                WIFI_FILTER_TO_LWIP,                                        /* config_type */
                WIFI_FILTER_MASK_LOCAL_PORT | WIFI_FILTER_MASK_PROTOCOL, /* match_mask */
        }
}
```

Here are the ScmChannel component's Repeater-related APIs:

- scm_wifi_set_default_filter
- scm_wifi_add_filter
- scm_wifi_del_filter
- scm_wifi_query_filter

2.4 API Usage Guide

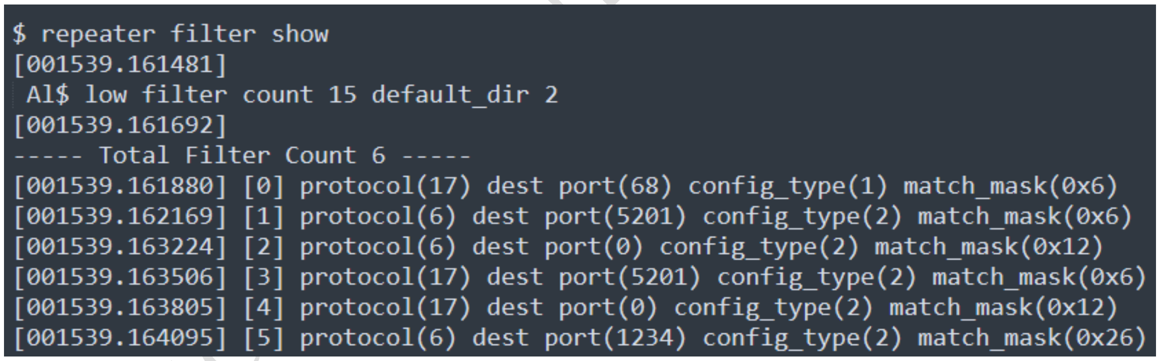
The APIs provided by ScmChannel are shown in Table 2-1.

Name	Description
scm_channel_set_default_filter	Set the default forwarding direction for Repeater (preset to forward to the host side).
scm_channel_add_filter	Add a filtering rule to Repeater's forwarding table.
scm_channel_del_filter	Remove a filtering rule from Repeater's forwarding table.

scm_channel_query_filter	Query the contents of Repeater's forwarding table.
scm_channel_reset_filter	Reset the filter configuration. During the bootup process, Repeater will have a set of preset filter configurations. This API must be executed before updating custom filters.
scm_channel_send_to_host	Send a message from the device to the host, with a maximum payload size of 1500 bytes.
scm_channel_register_rx_cb	Register a callback function to process received message data from the host.
scm_channel_host_ready	This API is used to determine if the host is in a state to receive messages.

Notes:

- he default number of Repeater component filtering rules is set to 15. If you wish to modify this, please adjust the \`CONFIG_SUPPORT_WIFI_REPEATER_IPV4_CNT\` macro.
- During the Scm1612 bootup process, multiple sets of filters are preconfigured. Users can use the \`repeater filter show\` command to query these filters.



```
$ repeater filter show
[001539.161481]
 Al$ low filter count 15 default_dir 2
[001539.161692]
----- Total Filter Count 6 -----
[001539.161880] [0] protocol(17) dest port(68) config_type(1) match_mask(0x6)
[001539.162169] [1] protocol(6) dest port(5201) config_type(2) match_mask(0x6)
[001539.163224] [2] protocol(6) dest port(0) config_type(2) match_mask(0x12)
[001539.163506] [3] protocol(17) dest port(5201) config_type(2) match_mask(0x6)
[001539.163805] [4] protocol(17) dest port(0) config_type(2) match_mask(0x12)
[001539.164095] [5] protocol(6) dest port(1234) config_type(2) match_mask(0x26)
```

It is recommended to execute \`scm_vlwip_netif_reset\` before configuring custom filters to perform a clearing action.

2.5 Inter-chip Heartbeat Mechanism (TBD)

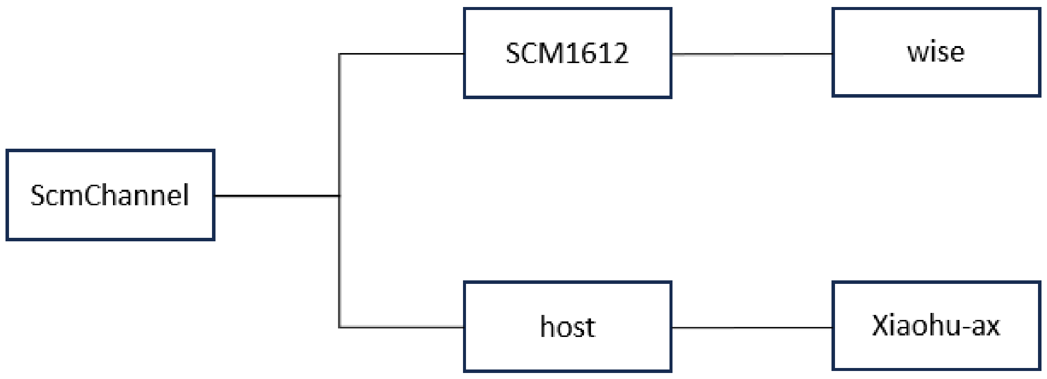
3 ScmChannel Usage

ScmChannel is a part of the SCM1612 system. This chapter describes how to integrate the ScmChannel software package into the SCM1612 software package so that you can quickly integrate the ScmChannel component into SCM1612.

3.1 Directory Structure of the ScmChannel Software Package

The directory structure of the ScmChannel software package is shown in Figure 3-1.

Figure 3-1 Directory Structure of the ScmChannel Software Package



3.2 ScmChannel Build

Step 1: On SCM1612

1. **Run the following command in the wise-sdk software package:**

```
make disclean
make scm1612s_sdio_defconfig
make
```

2. Generate wise.mcuboot.bin in the wise-sdk directory.

Step 2: On host

i.
```
cd xiaohu-ax/
cp configs/cfg_xxx.mk cfg.mk
```

Given the support for multiple platforms and configuration modes, such as SDIO OOB mode, SDIO INT mode, USB mode, etc., you need to select a configuration file based on your platform and requirements for compilation. The system provides default configuration files stored in the configs/ directory:

config	Platform/Mode
cfg_sdio_normal_fullhan.mk	fullhan, 4 bit mode interrupt
cfg_sdio_normal_goke.mk	goke, 4 bit mode interrupt
cfg_sdio_normal.mk	Linux PC, 4 bit mode interrupt
cfg_sdio_polling.mk	Linux PC, polling mode
cfg_sdio_oob_int_goke.mk	goke, OOB interrupt mode
cfg_usb.mk	Linux PC

ii. Compiling the sncmfmac.ko Driver

While ARCH and CROSS_COMPILE are already configured in the file, the 'KDIR' parameter needs to be explicitly specified based on your environment.

```
make KDIR=/home/apache/page/linux-4.9
```

If 'KDIR' is not specified, the system will use the default kernel KDIR, and the compiled driver will be suitable for running on a Linux PC.

```
make
```

iii. Compiling Applications

```
make KDIR=/home/apache/page/linux-4.9 apps
```

Currently, there are two types of applications:

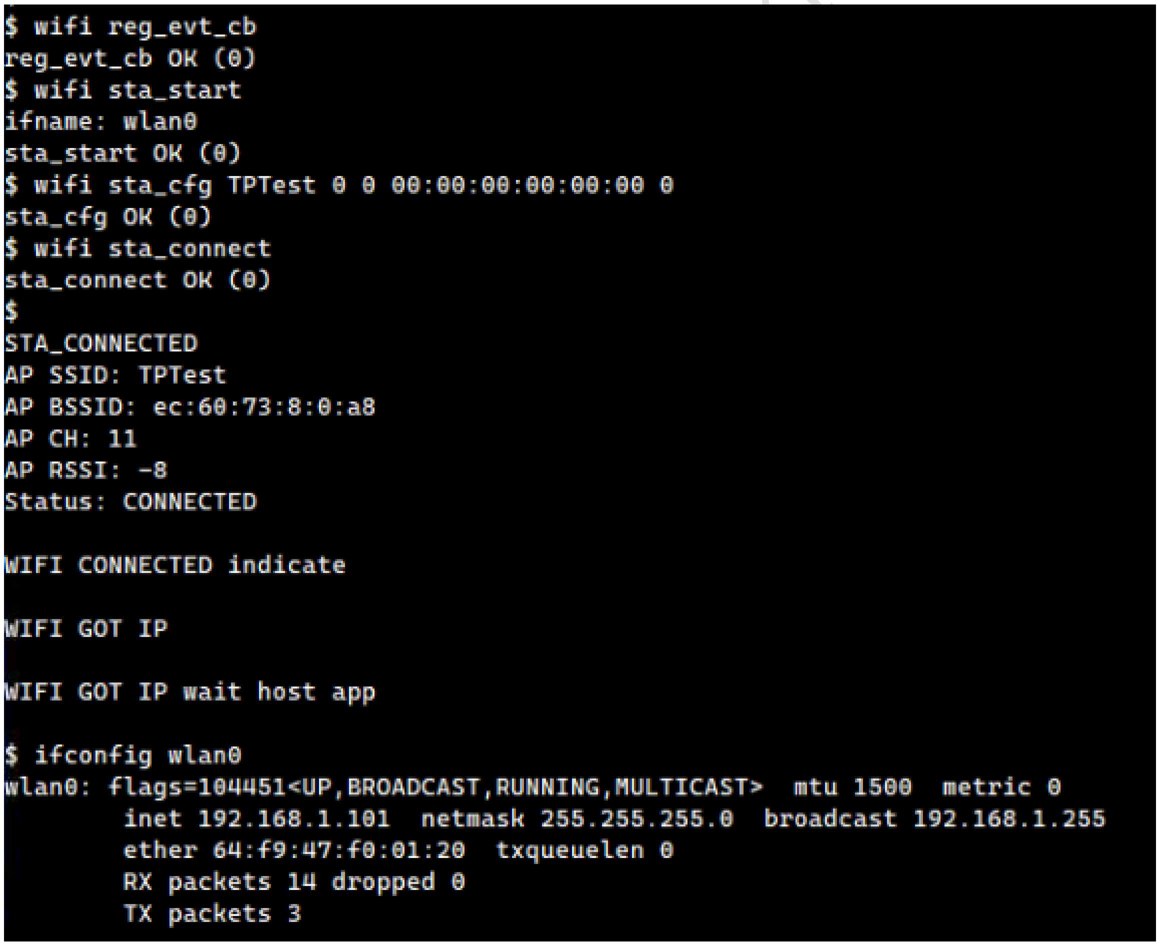
'sncm_cmd': Configures and controls Wi-Fi through host commands.
'sncm_chn' (also known as ScmChannel): Wi-Fi configuration and control are completed on the SCM1612 side, obtaining Wi-Fi connection

information through ScmChannel. 'sample_link' is mainly used to synchronize network node information such as Mac address, IP address, etc., on the SCM1612 side.
'sample_cli': Mainly used to send custom information from the client.

3.3 ScmChannel Example Usage

3.3.1 Configuring Network Information and Establishing Connection

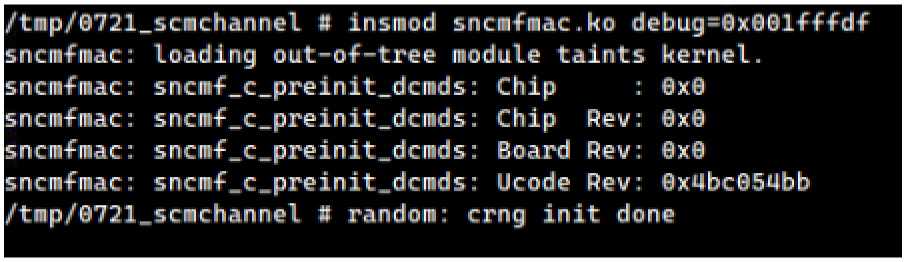
To start, on the SCM1612 side, refer to the third chapter of the document "SCM1612_Wi-Fi Software Development Guide.pdf," specifically within the section titled "Wi-Fi STA Functionality." Utilize the API functions or CLI commands outlined in that section to configure the network settings:



```
$ wifi reg_evt_cb
reg_evt_cb OK (0)
$ wifi sta_start
ifname: wlan0
sta_start OK (0)
$ wifi sta_cfg TPTest 0 0 00:00:00:00:00:00 0
sta_cfg OK (0)
$ wifi sta_connect
sta_connect OK (0)
$
STA_CONNECTED
AP SSID: TPTest
AP BSSID: ec:60:73:8:0:a8
AP CH: 11
AP RSSI: -8
Status: CONNECTED

WIFI CONNECTED indicate

WIFI GOT IP

WIFI GOT IP wait host app

$ ifconfig wlan0
wlan0: flags=104451<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500  metric 0
        inet 192.168.1.101  netmask 255.255.255.0  broadcast 192.168.1.255
        ether 64:f9:47:f0:01:20  txqueuelen 0
        RX packets 14 dropped 0
        TX packets 3
```

3.3.2 Verifying SDIO Connection

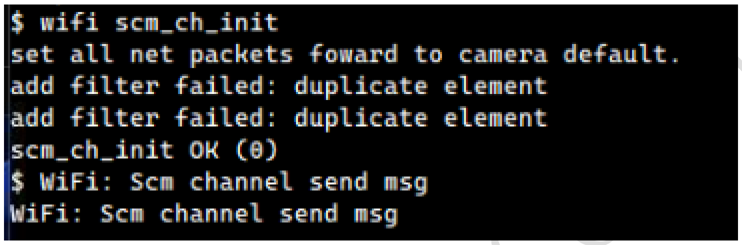
On the host side, first verify that the SDIO connection is established.



```
/tmp/0721_scmchannel # insmod sncmfmac.ko debug=0x001fffdf
sncmfmac: loading out-of-tree module taints kernel.
sncmfmac: sncmf_c_preinit_dcmds: Chip      : 0x0
sncmfmac: sncmf_c_preinit_dcmds: Chip  Rev: 0x0
sncmfmac: sncmf_c_preinit_dcmds: Board Rev: 0x0
sncmfmac: sncmf_c_preinit_dcmds: Ucode Rev: 0x4bc054bb
/tmp/0721_scmchannel # random: crng init done
```

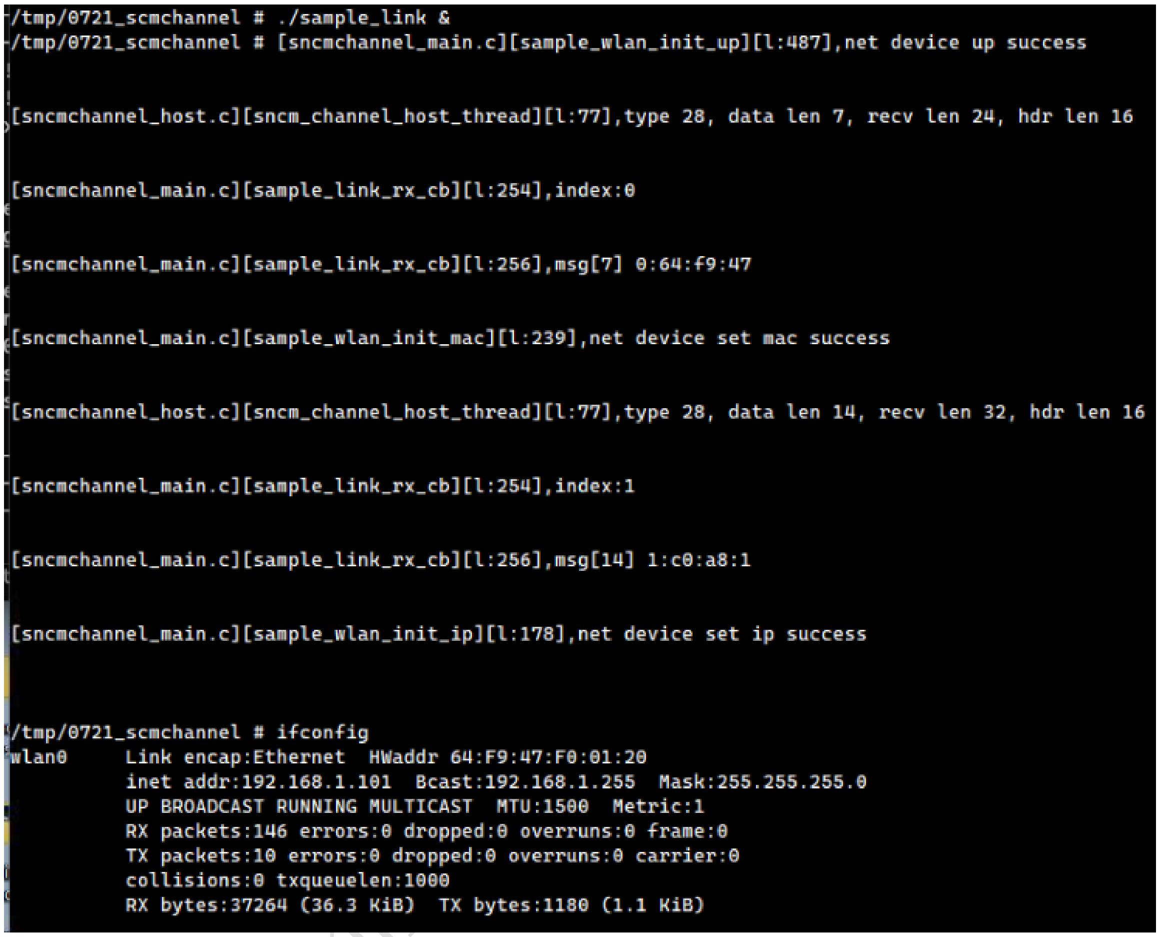
3.3.3 Establishing ScmChannel Connection

a. On the 1612 side, initialize the ScmChannel.



```
$ wifi scm_ch_init
set all net packets foward to camera default.
add filter failed: duplicate element
add filter failed: duplicate element
scm_ch_init OK (0)
$ WiFi: Scm channel send msg
WiFi: Scm channel send msg
```

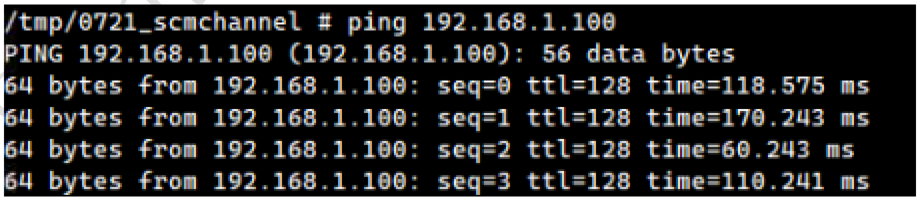
b. On the host side, use sample_link to automatically obtain the wlan0 IP and MAC addresses. Once successful, you can confirm the synchronized information by using the ifconfig wlan0 command.



```
/tmp/0721_scmchannel # ./sample_link &
/tmp/0721_scmchannel # [sncmchannel_main.c][sample_wlan_init_up][l:487],net device up success


[sncmchannel_host.c][sncm_channel_host_thread][l:77],type 28, data len 7, recv len 24, hdr len 16


[sncmchannel_main.c][sample_link_rx_cb][l:254],index:0


[sncmchannel_main.c][sample_link_rx_cb][l:256],msg[7] 0:64:f9:47


[sncmchannel_main.c][sample_wlan_init_mac][l:239],net device set mac success


[sncmchannel_host.c][sncm_channel_host_thread][l:77],type 28, data len 14, recv len 32, hdr len 16


[sncmchannel_main.c][sample_link_rx_cb][l:254],index:1


[sncmchannel_main.c][sample_link_rx_cb][l:256],msg[14] 1:c0:a8:1


[sncmchannel_main.c][sample_wlan_init_ip][l:178],net device set ip success



/tmp/0721_scmchannel # ifconfig
wlan0     Link encap:Ethernet  HWaddr 64:F9:47:F0:01:20
          inet addr:192.168.1.101  Bcast:192.168.1.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:146 errors:0 dropped:0 overruns:0 frame:0
          TX packets:10 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:37264 (36.3 KiB)  TX bytes:1180 (1.1 KiB)
```

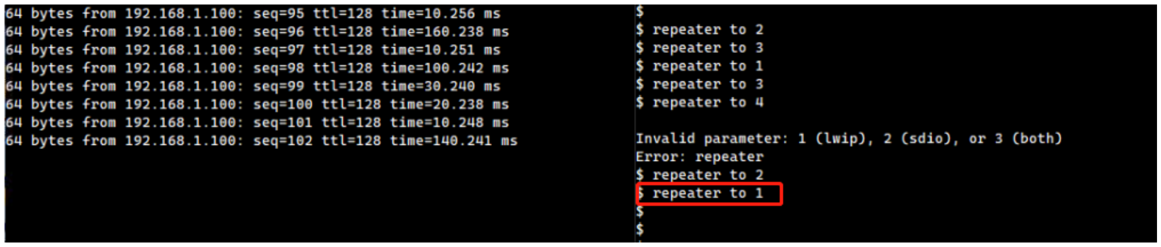
3.3.4 Modifying Forwarding Rules

a. On the host, ping the PC1 within the local network.



```
/tmp/0721_scmchannel # ping 192.168.1.100
PING 192.168.1.100 (192.168.1.100): 56 data bytes
64 bytes from 192.168.1.100: seq=0 ttl=128 time=118.575 ms
64 bytes from 192.168.1.100: seq=1 ttl=128 time=170.243 ms
64 bytes from 192.168.1.100: seq=2 ttl=128 time=60.243 ms
64 bytes from 192.168.1.100: seq=3 ttl=128 time=110.241 ms
```

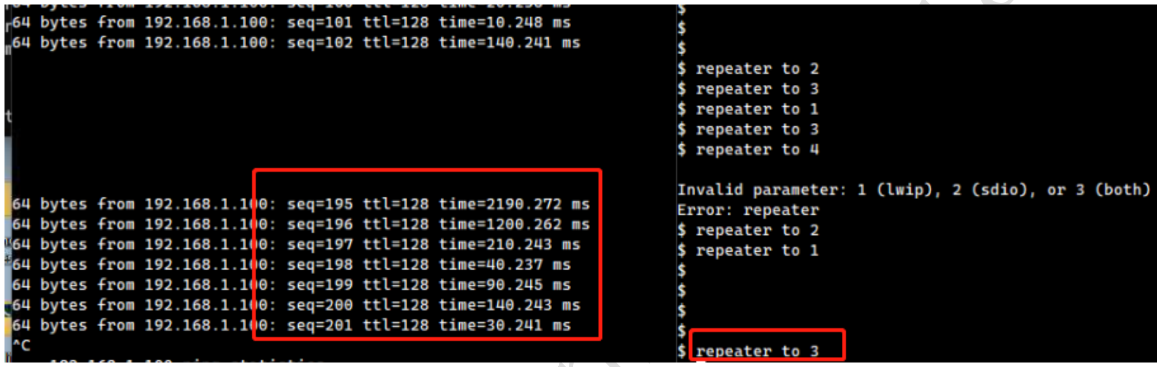
b. Modify the forwarding rules to forward only to the 1612 side and not to the host. The ping should be interrupted.



```
64 bytes from 192.168.1.100: seq=95 ttl=128 time=10.256 ms
64 bytes from 192.168.1.100: seq=96 ttl=128 time=160.238 ms
64 bytes from 192.168.1.100: seq=97 ttl=128 time=10.251 ms
64 bytes from 192.168.1.100: seq=98 ttl=128 time=100.242 ms
64 bytes from 192.168.1.100: seq=99 ttl=128 time=30.240 ms
64 bytes from 192.168.1.100: seq=100 ttl=128 time=20.238 ms
64 bytes from 192.168.1.100: seq=101 ttl=128 time=10.248 ms
64 bytes from 192.168.1.100: seq=102 ttl=128 time=140.241 ms
```

```
$
$ repeater to 2
$ repeater to 3
$ repeater to 1
$ repeater to 3
$ repeater to 4

Invalid parameter: 1 (lwip), 2 (sdio), or 3 (both)
Error: repeater
$ repeater to 2
$ repeater to 1
$
$
```

c. Modify the forwarding rules to forward to both the host and the 1612 side simultaneously. Ping should be restored.



```
64 bytes from 192.168.1.100: seq=101 ttl=128 time=10.248 ms
64 bytes from 192.168.1.100: seq=102 ttl=128 time=140.241 ms




64 bytes from 192.168.1.100: seq=195 ttl=128 time=2190.272 ms
64 bytes from 192.168.1.100: seq=196 ttl=128 time=1200.262 ms
64 bytes from 192.168.1.100: seq=197 ttl=128 time=210.243 ms
64 bytes from 192.168.1.100: seq=198 ttl=128 time=40.237 ms
64 bytes from 192.168.1.100: seq=199 ttl=128 time=90.245 ms
64 bytes from 192.168.1.100: seq=200 ttl=128 time=140.243 ms
64 bytes from 192.168.1.100: seq=201 ttl=128 time=30.241 ms
^C
```

```
$
$
$ repeater to 2
$ repeater to 3
$ repeater to 1
$ repeater to 3
$ repeater to 4

Invalid parameter: 1 (lwip), 2 (sdio), or 3 (both)
Error: repeater
$ repeater to 2
$ repeater to 1
$
$
$
$
$ repeater to 3
```