

# SSV6X5X 驱动移植用户指南

## 1 驱动编译

### 1.1. 驱动可单独编译

请参考《linux ssv6x5x 驱动快速编译方法.txt》，建议优先采用快速编译这样可以提高移植效率。

### 1.2. Linux 内核中编译

#### 增加 kernel 里 ssv6x5x 驱动

将 ssv6x5x 驱动存放在 drivers/net/wireless/路径下。

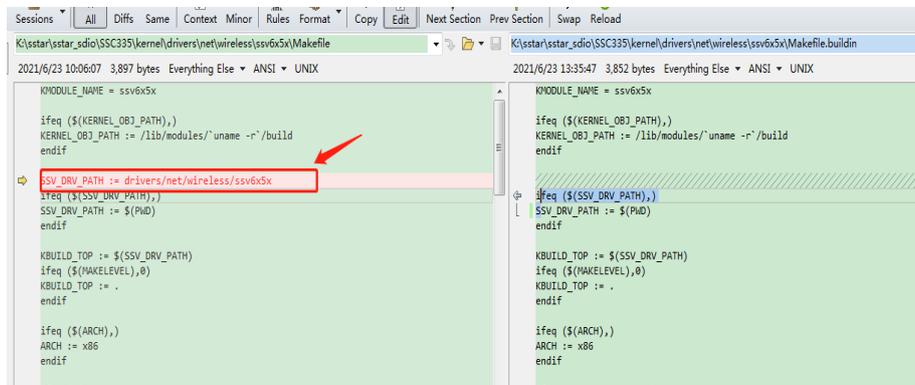
- 修改当前目录的 Makefile

```
obj-$(CONFIG_BCMDHD_AP6181) += bcmdhd_ap6181/
obj-$(CONFIG_MT7601_STA) += mt7601u/
obj-$(CONFIG_SSV6X5X) += ssv6x5x/
```

- 修改当前目录的 Kconfig

```
source "drivers/net/wireless/bcmdhd_1_141_66/Kconfig"
source "drivers/net/wireless/bcmdhd_ap6181/Kconfig"
source "drivers/net/wireless/mt7601u/Kconfig"
source "drivers/net/wireless/ssv6x5x/Kconfig"
```

- 针对 SSV6x5x 的支持，驱动里的 Makefile 增加一行



增加 p2p interface (此选项非必须，用于安卓 miracast，或者 AP+STATION 共存模式)

- 修改 linux/net/mac80211/main.c

```
/* add one default STA interface if supported */
if (local->hw.wiphy->interface_modes & BIT(NL80211_IFTYPE_STATION)) {
    result = ieee80211_if_add(local, "wlan%d", NULL, NL80211_IFTYPE_STATION, NULL);
    if (result)
        wiphy_warn(local->hw.wiphy, "Failed to add default virtual iface\n");
}
```

增加

```
if (local->hw.wiphy->interface_modes & (BIT(NL80211_IFTYPE_P2P_GO)|BIT(NL80211_IFTYPE_P2P_CLIENT))) {
    result = ieee80211_if_add(local, "p2p%d", NULL, NL80211_IFTYPE_STATION, NULL);
    if (result)
        wiphy_warn(local->hw.wiphy, "Failed to add default virtual iface\n");
}
```

配置内核

1. ssv6x5x WiFi 驱动配置。 .config 需要进行配置如下。如要 ssv6x5x buildin 到内核（前提是 **cfg80211/mac80211** 都需要 **buildin** 到内核）， buildin 选 Y，编成 module 选 M

```
^(-)
< > Ralink driver support --->
< > Realtek wireless card support
[ ] TI Wireless LAN support --->
< > ZyDAS ZD1211/ZD1211B USB-wireless support
< > Marvell WiFi-Ex Driver
< > Broadcom 43341 wireless cards support
< > Broadcom bcm43438 wireless cards support
< > Broadcom FullMAC wireless cards support
< > MT7601_util STA support
<M> SSV6X5X Wireless driver

<Select> < Exit > < Help > < Save >
```

2，内核需支持 MAC80211 & cfg80211， buildin 选 Y，编成 module 选 M。

```

--- Wireless
<*>  cfg80211 - wireless configuration API
[ ]   nl80211 testmode command
[ ]   enable developer warnings
[ ]   cfg80211 regulatory debugging
[ ]   enable powersave by default
[ ]   use statically compiled regulatory rules database
[*]   cfg80211 wireless extensions compatibility
[*]   Wireless extensions sysfs files
-*--  Common routines for IEEE802.11 drivers
[ ]   lib80211 debugging messages
[ ]   Allow reconnect while already connected
[*]   Generic IEEE 802.11 Networking Stack (mac80211)
[ ]   PID controller based rate control algorithm
[*]   Minstrel
[*]   Minstrel 802.11n support

```

## 驱动 buildin 到 kernel 注意事项

1. 部分平台需要注释如下，否则驱动加载时无入口函数。

```

00031:     ssvdevice_exit();
00032:     return;
00033: }
00034:
00035: static int generic_wifi_init_module(void)
00036: {
00037:     return initWlan();
00038: }
00039:
00040: static void generic_wifi_exit_module(void)
00041: {
00042:     exitWlan();
00043: }
00044:
00045: EXPORT_SYMBOL(generic_wifi_init_module);
00046: EXPORT_SYMBOL(generic_wifi_exit_module);
00047:
00048: #if 0 // CONFIG_SSV6XXX // CONFIG_SSV6XXX=y
00049: late_initcall(generic_wifi_init_module);
00050: #else // CONFIG_SSV6XXX=m or =n
00051: module_init(generic_wifi_init_module);
00052: #endif
00053: module_exit(generic_wifi_exit_module);
00054:
00055: MODULE_LICENSE("Dual BSD/GPL");

```

2. 驱动 buildin 到内核里的话，驱动运行会报一个 warning 告警。请注释如下 code 可以去掉此告警 WARN\_ON。

```

00367:     size_t s;
00368:
00369:     for(s=0; cfg_cmds[s].cfg_cmd != NULL; s++) {
00370:         if ((cfg_cmds[s].def_val) != NULL) {
00371:             cfg_cmds[s].translate_func(cfg_cmds[s].def_val,
00372:                                     cfg_cmds[s].var, cfg_cmds[s].arg);
00373:         }
00374:     }
00375: }
00376:
00377: static void _import_default_cfg (char *stacfgpath)
00378: {
00379:     struct file *fp = (struct file *) NULL;
00380:     char buf[MAX_CHARS_PER_LINE], cfg_cmd[32], cfg_value[32];
00381:     mm_segment_t fs;
00382:     size_t s, read_len = 0, is_cmd_support = 0;
00383:     printk("\n*** %s, %s ***\n", __func__, stacfgpath);
00384:
00385:     // Init the buffer with 0
00386:     memset(&ssv_cfg, 0, sizeof(ssv_cfg));
00387:
00388:     // set default config value
00389:     _set_initial_cfg_default();
00390:
00391:     if (stacfgpath == NULL)
00392:     {
00393:         WARN_ON(1);
00394:         return;
00395:     }
00396:
00397:     memset(buf, 0, sizeof(buf));

```

也可以修改驱动 code，ssvdevice.c 里的全局变量指定默认的 cfg 路径（前提是驱动加载前时文件系统可用）。

```
static char *stacfgpath = /xxx/xxx/ssv6x5x-wifi.cfg;
```

如果 stacfgpath 未指定驱动是用 code 默认的配置。见下

```

00264: } ? end __string2configuration ?
00265:
00266: // Note: if there is no default value, set default to NULL, and it will be initialized as zero
00267: struct ssv6xxx_cfg_cmd_table cfg_cmds[] = {
00268: { "hw_mac", (void *) ssv_cfg.maddr[0][0], 0, __string2mac, NULL},
00269: { "hw_mac_2", (void *) ssv_cfg.maddr[1][0], 0, __string2mac, NULL},
00270: { "def_chan", (void *) ssv_cfg.def_chan, 0, __string2u32, "6"},
00271: { "hw_cap_ht", (void *) ssv_cfg.hw_caps, 0, __string2flag32, "on"},
00272: { "hw_cap_gf", (void *) ssv_cfg.hw_caps, 1, __string2flag32, "off"},
00273: { "hw_cap_2ghz", (void *) ssv_cfg.hw_caps, 2, __string2flag32, "on"},
00274: { "hw_cap_5ghz", (void *) ssv_cfg.hw_caps, 3, __string2flag32, "off"},
00275: { "hw_cap_security", (void *) ssv_cfg.hw_caps, 4, __string2flag32, "on"},
00276: { "hw_cap_sgi", (void *) ssv_cfg.hw_caps, 5, __string2flag32, "on"},
00277: { "hw_cap_ht40", (void *) ssv_cfg.hw_caps, 6, __string2flag32, "on"},
00278: { "hw_cap_ap", (void *) ssv_cfg.hw_caps, 7, __string2flag32, "on"},
00279: { "hw_cap_p2p", (void *) ssv_cfg.hw_caps, 8, __string2flag32, "on"},
00280: { "hw_cap_ampdu_rx", (void *) ssv_cfg.hw_caps, 9, __string2flag32, "on"},
00281: { "hw_cap_ampdu_tx", (void *) ssv_cfg.hw_caps, 10, __string2flag32, "on"},
00282: { "hw_cap_tdis", (void *) ssv_cfg.hw_caps, 11, __string2flag32, "off"},
00283: { "hw_cap_stbc", (void *) ssv_cfg.hw_caps, 12, __string2flag32, "on"},
00284: { "hw_cap_hci_rx_aggr", (void *) ssv_cfg.hw_caps, 13, __string2flag32, "on"},
00285: { "hw_cap_beacon", (void *) ssv_cfg.hw_caps, 14, __string2flag32, "off"},
00286: { "hw_cap_krack", (void *) ssv_cfg.hw_caps, 15, __string2flag32, "off"},
00287: { "hw_cap_wow", (void *) ssv_cfg.hw_caps, 16, __string2flag32, "on"},
00288: { "hw_cap_bq4", (void *) ssv_cfg.hw_caps, 17, __string2flag32, "off"},
00289: { "hw_cap_hci_tx_aggr", (void *) ssv_cfg.hw_caps, 18, __string2flag32, "on"},
00290: { "hw_cap_report_tx_ack", (void *) ssv_cfg.hw_caps, 19, __string2flag32, "off"},
00291: { "xtal_clock", (void *) ssv_cfg.crystal_type, 0, __string2u32, "24"},
00292: { "volt_regulator", (void *) ssv_cfg.volt_regulator, 0, __string2u32, "1"},
00293: { "firmware_path", (void *) ssv_cfg.firmware_path[0], 0, __string2str, NULL},
00294: { "flash_bin_path", (void *) ssv_cfg.flash_bin_path[0], 0, __string2str, NULL},
00295: { "mac_address_path", (void *) ssv_cfg.mac_address_path[0], 0, __string2str, NULL},
00296: { "mac_output_path", (void *) ssv_cfg.mac_output_path[0], 0, __string2str, NULL},
00297: { "ignore_efuse_mac", (void *) ssv_cfg.ignore_efuse_mac, 0, __string2u32, NULL},
00298: { "efuse_rate_gain_mask", (void *) ssv_cfg.efuse_rate_gain_mask, 0, __string2u32, "0xf"},
00299: { "mac_address_mode", (void *) ssv_cfg.mac_address_mode, 0, __string2u32, NULL},
00300: { "register", NULL, (void *) ssv_cfg.beacon_rssi_minimal, 0, __string2configuration, NULL},
00301: { "beacon_rssi_minimal", (void *) ssv_cfg.beacon_rssi_minimal, 0, __string2u32, NULL},

```

## 2 驱动运行

### 2.1 硬件和配置检查

#### 检查晶振配置

首先确认模块或 COB 上的晶振是多少。目前 SSV6155/6255 支持 25M/ 40M 晶振



图上为 40 字样，表示 40M 晶振。

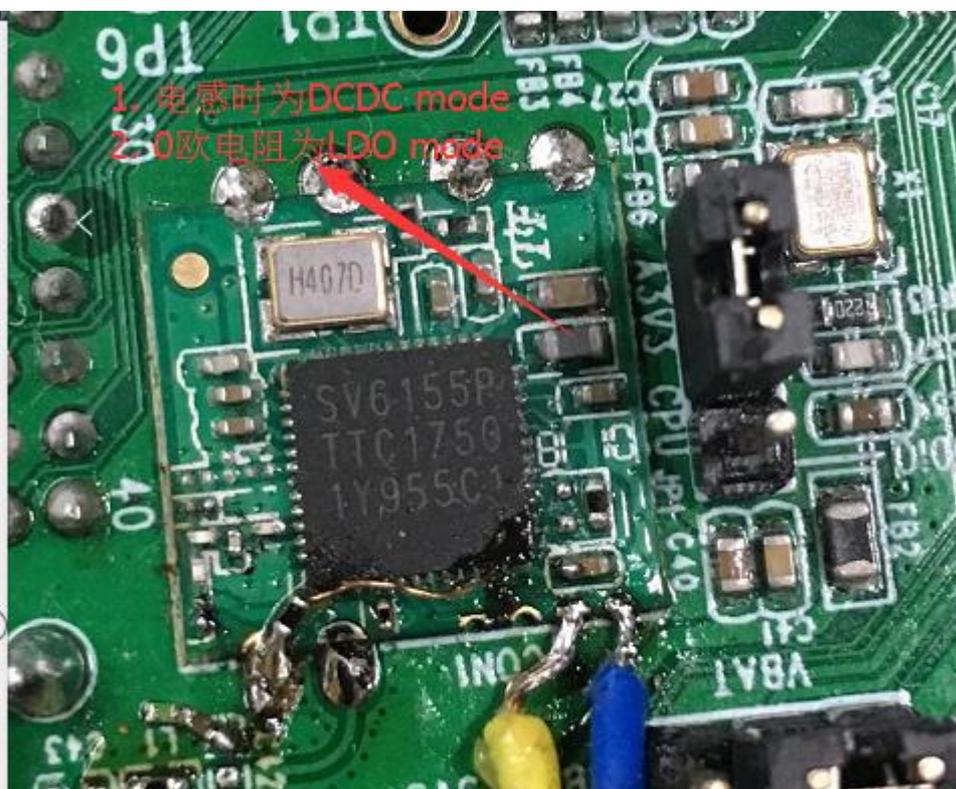
请修改板上配置文件 xxxx-wifi.cfg（默认是 ssv6x5x-wifi.cfg）中晶振设置。

注意 usb wifi 一般晶振规格是 25M/40M，sdio wifi 一般晶振规格是 24M/26M

```

40
41 #####
42 # Hardware setting
43 #
44 #volt regulator(DCDC-0 LDO-1)
45 #
46 #####
47 xtal_clock = 40
48 volt_regulator = 1
49
50 #####
    
```

1. 请检查模块上是 LDO 还是 DCDC mode。并确认 xxxx-wifi.cfg 配置中是否一致



```

39 #mac_output_path = /data/wifi/mac
40
41 #####
42 # Hardware setting
43 #
44 #volt regulator(DCDC-0 LDO-1)
45 #
46 #####
47 xtal_clock = 40
48 volt_regulator = 1
49
    
```

## 确认软件是否识别到 wifi ic

对于 usb wifi, 执行如下确认 (for 6155/6255)

```
lsusb
```

```
Bus 001 Device 002: ID 8065:6000
```

对于 sdio wifi, 执行如下确认 (for 6152/6256)

```
cat /sys/bus/mmc/devices/mmc1\:0001/mmc1\:0001\:1/vendor
```

```
0x3030
```

对于 sdio wifi, 执行如下确认 (for 6158)

```
cat /sys/bus/mmc/devices/mmc1\:0001/mmc1\:0001\:1/vendor
```

```
0x5653
```

### 注意:

1. 如果以上信息无法获取, 请确认 wifi 硬件的电源和 LDO\_EN 管脚是否正常。
2. 硬件检查 ok 还是认不到 sdio vendor id, 有些平台的 mmc 或 sdio 驱动, 需要手动调用 api 重新认 sdio。例如某平台需在 ko 加载入口里添加 wifi\_card\_detect 或 sdio rescan 后, 就可以认到 sdio。

## 2.2 检测 ssv6x5x.ko 是否安装成功

- 复制相关文件到板子的文件系统中
  - 驱动目录中 ssv6x5x-wifi.cfg 文件, 例如 copy 到板子如下位置  
[/etc/firmware/]

### ssv6x5x-wifi.cfg 文件配置介绍

MAC address 此选项针对 mac 地址

hw\_mac, hw\_mac\_2 一般用于调试, 强制 wifi 的 mac 地址。此选项请关闭。

```

1 #####
2 # Ingenic T10
3 # WIFI-CONFIGURATION
4 #####
5 #hw_mac =
6 #hw_mac_2 =
7

```

ignore\_efuse\_mac 一般都是 0, wifi 模组一般都会出厂时烧录好 mac 地址在 efuse 里。

如果是 COB, wifi 默认 efuse 没有 mac 地址, 可根据 mac\_address\_mode 选择 mac 地址产生的算法。

mac\_address\_mode 为 0 时, 固定 mac 地址为 00:33:33:33:33:33

mac\_address\_mode 为 1 时, mac 地址每次都是随机产生。

mac\_address\_mode 为 2 时, mac 地址一次是随机产生, 然后此地址会保存在 mac\_output\_path 的路径文件中, 之后每次都用 mac\_output\_path 文件里保存的 mac 地址。

```

#####
MAC address
Priority 1. From wifi.cfg [ hw_mac & hw_mac_2 ]
Priority 2. From e-fuse[ON/OFF switch by wifi.cfg]
Priority 3. From insert module parameter
Priority 4. From external file path
path only support some special charater "_" ":" "/" "." "-"
Priority 5. Default[Software mode]
0. => 00:33:33:33:33:33
1. => Always random
2. => First random and write to file[Default path mac_output_path]
#####
ignore_efuse_mac = 0
#mac_address_mode = 2
#mac_output_path = /data/wifimac

```

晶振和 volt regulator 请参考章节 硬件和配置检查。  
其它参数建议不用修改，如需修改请和原厂技术确认。

### 加载驱动 ssv6x5x.ko

- 执行如下指令：（stacfgpath=路径请根据实际路径填写）

```
insmod ssv6x5x.ko stacfgpath=/etc/firmware/ssv6x5x-wifi.cfg
```

执行 dmesg 或 cat /dev/kmsg 确认 或 打开 kernel log。调整 log 级别为最高。echo 7 >

/proc/sys/kernel/printk

可以看到如下信息，表示驱动运行成功。

```
[ 80.734637] *** _import_default_cfg, /mnt/t20/ssv6x5x-wifi.cfg ***
[ 80.734637]
[ 80.779830] ssv6xxx_hci_init() start
[ 80.784016] ssv6xxx_sdio_init
[ 80.790171] ssv6xxx_usb_init
[ 80.796030]
[ 80.801179] ==          TURISMO - USB          ==
[ 80.806233]
[ 80.811728] SSV6XXX_USB 1-1:1.0: CHIP ID: SSV6006D0
[ 80.817515] Attach SSV6006 family HWIF HAL function
[ 80.822992] Chip type a0
[ 80.825614] Load SSV6006 HWIF HAL HWIF function
[ 80.830829] ssv6xxx_dev_probe(): SSV6X5X device "SSV6006D" found !
[ 80.837274] SSV6006D
[ 80.839534] Attach SSV6006 family HAL function
[ 80.847458] Load SSV6006 common code
[ 80.851236] Load SSV6006C/D HAL MAC function
[ 80.855810] Chip type a0
[ 80.858432] Load SSV6006 HAL common PHY function
[ 80.863339] Load SSV6006C/D HAL BB-RF function
[ 80.868230] SSV WLAN driver SSV6006D: Enable RX(ep4) acc
[ 80.873821] SSV6XXX HCI TX Task started.
[ 80.878145] CHIP TAG: 2017071400053011
[ 80.882622] MAC address from e-fuse
[ 80.886240] EFUSE configuration
[ 80.889493] Read efuse chip identity[70000000]
[ 80.894146] r_calibration_result- 0
[ 80.897670] sar_result- 0
[ 80.900382] crystal_frequency_offset- 99
[ 80.904504] tx_power_index_1- 5
[ 80.907759] tx_power_index_2- 4
[ 80.911051] MAC address - 58:04:54:60:01:6d
[ 80.915386] rate_table_1- 0
[ 80.918278] rate_table_2- 0
[ 80.921450] flash file /tmp/flash.bin not found
[ 80.926151] ssv6006_if_chk_mac2: is not need to check MAC address 2 for this model
[ 80.934140] not support 5G for this chip!!
[ 80.938474] not support 5G for this chip!!
[ 80.942852] ssv6006_adj_config: clear hci rx aggregation setting
[ 80.949166] ssv6006_adj_config: clear hci tx aggregation setting
[ 80.955526] ssv6006_adj_config: not support external PA for this chip
[ 80.962490] SSV6XXX RX Task started.
[ 80.966365] ssv6xxx_usb_rx_task: nr_recvbuff=5
[ 81.032201] wait 0 ms for usb xom code ready
[ 81.036641] SSV WLAN driver SSV6006D: Disable RX(ep4) acc
[ 81.043100] SSV WLAN driver SSV6006D: Enable RX(ep4) acc
[ 81.059091] Using firmware "ssv6x5x-sw.bin".
[ 81.063546] SSV WLAN driver SSV6006D: Jump to ROM
[ 81.760507] Firmware version 8461
[ 81.987460] patch[ccb0e134 => 00100010]
[ 81.993486] chan change ch 6, type 1, off_chan 0
[ 82.030948] INIT SSV CONTROL GENERIC NETLINK MODULE
```

如果 log 里没有出现 “TURISMO - USB” 或者 “RUN SDIO” 的字符，说明系统没有认到 usb 或 sdio wifi，请参考上面的说明检查主控系统软硬件设置，再次强调认不到 sdio 或 usb 的情况，和驱动代码和 cfg 本身无关。

- 如上述已成功则执行：`ifconfig wlan0 up`

```
[root@Ingenic-g1_1:t20]# ifconfig wlan0 up
[ 95.801873] SVN version 7807
[ 95.804867] SVN ROOT URL http://10.10.20.22/svn/wifi/host/SMAC/branch/L.SMAC.19Q3.0000.00
[ 95.813534] COMPILER HOST ssv-Veriton-M490-1
[ 95.818050] COMPILER DATE 05-25-2021-09:20:52
[ 95.822686] COMPILER OS linux
[ 95.825849] COMPILER OS ARCH x86_64-linux-gnu-thread-multi
[ 95.843122] Set USB LFM support to 0
[ 95.847536] ssv6200_start(): current channel: 1,sc->ps_status=0
[ 95.853770] chan change ch 1, type 0, off_chan 0
[ 95.870067] [I] ssv6200_add_interface():
[ 95.874262] ssv6xxx_config_vif_res_id[0].
[ 95.878484] SSV WLAN driver SSV6006D: Set new macaddr
[ 95.884931] SSV WLAN driver SSV6006D: VIF 58:04:54:60:01:6d of type 2 is added.
[ 95.892662] BSS Changed use_short_preamble[0]
[ 95.897177] BSS Changed use_cts_prot[0]
[ 95.901413] BSS_CHANGED_ERP_SLOT: use_short_slot[0]
[ 95.906644] NL80211_IFTYPE_STATION!!
[ 95.910482] [I] ssv6200_bss_info_changed(): leave
[ 95.915435] [I] sv6200_conf_tx vif[0] qos[0] queue[0] aifsn[2] cwmin[15] cwmax[1023] txop[0]
[ 95.924756] [I] sv6200_conf_tx vif[0] qos[0] queue[1] aifsn[2] cwmin[15] cwmax[1023] txop[0]
[ 95.934008] [I] sv6200_conf_tx vif[0] qos[0] queue[2] aifsn[2] cwmin[15] cwmax[1023] txop[0]
[ 95.943188] [I] sv6200_conf_tx vif[0] qos[0] queue[3] aifsn[2] cwmin[15] cwmax[1023] txop[0]
[ 95.952366] NL80211_IFTYPE_STATION!!
[ 95.956072] [I] ssv6200_bss_info_changed(): leave
[ 95.961128] IEEE80211_CONF_CHANGE_POWER change power level to 20
[ 95.967470] Disable IEEE80211_CONF_PS ps_aid=0
[ 95.972178] Disable IEEE80211_CONF_MONITOR
[root@Ingenic-g1_1:t20]#
[root@Ingenic-g1_1:t20]# ifconfig
eth0      Link encap:Ethernet HWaddr CE:0A:CC:BD:3B:E7
          inet addr:10.10.10.189 Bcast:10.255.255.255 Mask:255.0.0.0
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
          RX packets:475 errors:0 dropped:0 overruns:0 frame:0
          TX packets:289 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:634119 (619.2 KiB) TX bytes:24572 (23.9 KiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1 Mask:255.0.0.0
          UP LOOPBACK RUNNING MTU:65536 Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

wlan0     Link encap:Ethernet HWaddr 14:B2:E5:76:2D:2E
          UP BROADCAST MULTICAST MTU:1500 Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)
```

检查驱动 log 是否有异常的打印。

- 如果运行正常 ifconfig -a 可看到 wlan0 接口已经生成。
- 附 SSV 驱动成功加载 log 参考

 [ssv6x5x\(usb\)驱动加载log.txt](#)

 [ssv6x5x\(sdio\)驱动加载log.txt](#)

 [ssv6158\(sdio\)驱动加载log.txt](#)

### 3 安卓系统的上层修改

请参考具体平台的移植说明。如需帮助，请与 wifi 原厂联系。

## 4 问题与解答

### 3.1. Station 模式

启动 wpa\_supplicant

根据配置文件启动 wpa\_supplicant 进程发起连线

```
wpa_supplicant -iwlan0 -Dnl80211 -c/mnt/wpa_supplicant.conf &
```

建议 wpa\_supplicant 启动采用 nl80211，不建议使用 wext

扫描 AP

Wlan0 启动

```
ifconfig wlan0 up #wlan0 up
```

如用 iwlist 工具

```
iwlist wlan0 scan #wlan0 扫描
```

如用 wpa\_cli 工具

```
wpa_cli ifname=wlan0 scan #wlan0 扫描
```

无密码连接

如用 iwconfig 工具

```
iwconfig wlan0 essid ChinaNet #连接无密码的 ESSID，为了测试专门设置一个无密码的。
```

```
iwconfig #查看是否连接上
```

如用 wpa\_cli 工具

```
wpa_cli add_network #增加连线
```

```
wpa_cli set_network 0 ssid #配置 ssid
```

```
wpa_cli set_network 0 key_mgmt NONE #配置无密码加密
```

```
wpa_cli enable_network 0 #启动连线
```

```
wpa_cli status #查看当前连线状态
```

加密连接

```
killall wpa_supplicant #杀死上次运行的 wpa_supplicant 进程
```

```
wpa_passphrase ${SSID} ${PASSWD} >> /mnt/wpa_supplicant.conf #生成加密 ssid 的配置文件
```

```
wpa_supplicant -iwlan0 -Dnl80211 -c/mnt/wpa_supplicant.conf & #根据配置文件启动 wpa_supplicant 进程发起连线
```

获取 ip 地址

```
udhcpc -i wlan0& #获取 IP 地址
```

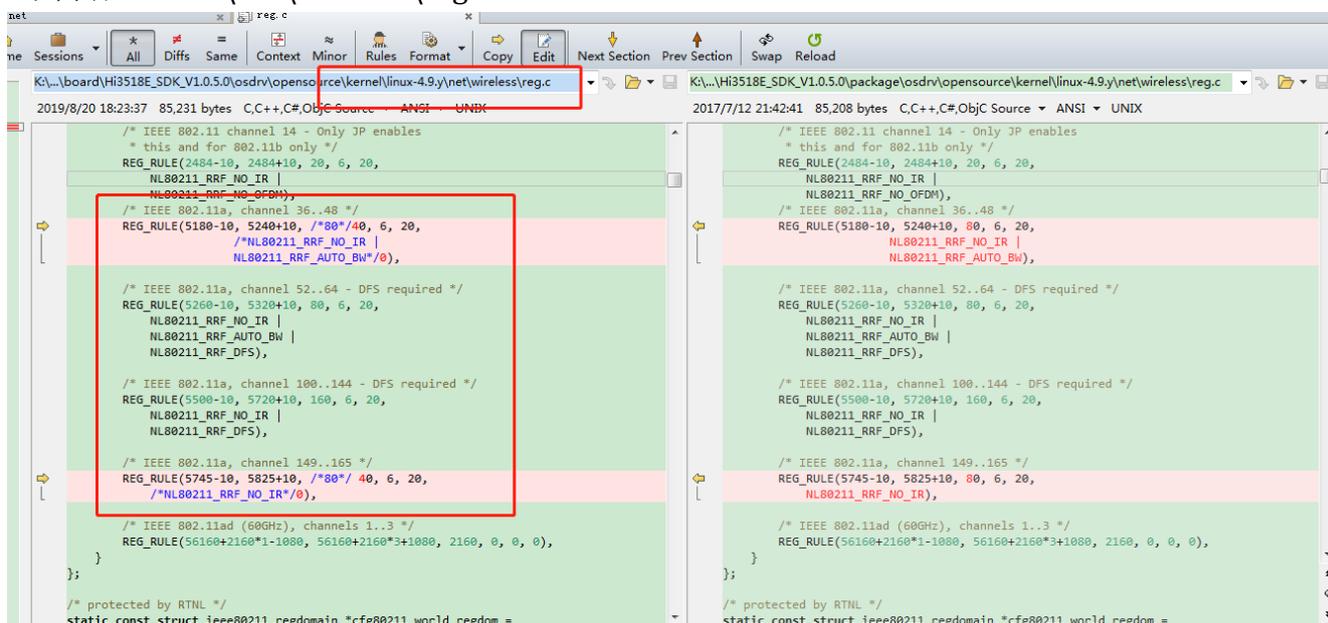
```
busybox ifconfig #查看 ip 地址
```

```
ping 网关 ip #ping 网关 ip, 测试网络
```

## 3.2. 5G softap 相关

修改 kernel code

请修改如下内容 kernel\net\wireless\reg.c



```

2019/8/20 18:23:37 85,231 bytes C,C++,C#,ObjC Source ANSI UNIX
/* IEEE 802.11 channel 14 - Only JP enables
 * this and for 802.11b only */
REG_RULE(2484-10, 2484+10, 20, 6, 20,
NL80211_RRF_NO_IR |
NL80211_RRF_NO_OFDM),
/* IEEE 802.11a, channel 36..48 */
REG_RULE(5180-10, 5240+10, /*80*/40, 6, 20,
/*NL80211_RRF_NO_IR |
NL80211_RRF_AUTO_BW*/0),
/* IEEE 802.11a, channel 52..64 - DFS required */
REG_RULE(5260-10, 5320+10, 80, 6, 20,
NL80211_RRF_NO_IR |
NL80211_RRF_AUTO_BW |
NL80211_RRF_DFS),
/* IEEE 802.11a, channel 100..144 - DFS required */
REG_RULE(5500-10, 5720+10, 160, 6, 20,
NL80211_RRF_NO_IR |
NL80211_RRF_DFS),
/* IEEE 802.11a, channel 149..165 */
REG_RULE(5745-10, 5825+10, /*80*/ 40, 6, 20,
/*NL80211_RRF_NO_IR*/0),
/* IEEE 802.11ad (60GHz), channels 1..3 */
REG_RULE(56160+2160*1-1080, 56160+2160*3+1080, 2160, 0, 0, 0),
};
/* protected by RTNL */
static const struct ieee80211_regdomain *cfr80211_world_regdom =

2017/7/12 21:42:41 85,208 bytes C,C++,C#,ObjC Source ANSI UNIX
/* IEEE 802.11 channel 14 - Only JP enables
 * this and for 802.11b only */
REG_RULE(2484-10, 2484+10, 20, 6, 20,
NL80211_RRF_NO_IR |
NL80211_RRF_NO_OFDM),
/* IEEE 802.11a, channel 36..48 */
REG_RULE(5180-10, 5240+10, 80, 6, 20,
NL80211_RRF_NO_IR |
NL80211_RRF_AUTO_BW),
/* IEEE 802.11a, channel 52..64 - DFS required */
REG_RULE(5260-10, 5320+10, 80, 6, 20,
NL80211_RRF_NO_IR |
NL80211_RRF_AUTO_BW |
NL80211_RRF_DFS),
/* IEEE 802.11a, channel 100..144 - DFS required */
REG_RULE(5500-10, 5720+10, 160, 6, 20,
NL80211_RRF_NO_IR |
NL80211_RRF_DFS),
/* IEEE 802.11a, channel 149..165 */
REG_RULE(5745-10, 5825+10, 80, 6, 20,
NL80211_RRF_NO_IR),
/* IEEE 802.11ad (60GHz), channels 1..3 */
REG_RULE(56160+2160*1-1080, 56160+2160*3+1080, 2160, 0, 0, 0),
};
/* protected by RTNL */
static const struct ieee80211_regdomain *cfr80211_world_regdom =

```

Hostapd 参考源码

附 hostapd 源码和 hostapd.conf 文件参考



hostapd\_5g.tar.gz



hostapd\_wlan0\_5g.conf

## 3.3. 5G station 相关

修改 kernel 的 kernel/net/wireless/scan.c , 增大 scan ap list 的保留时间。

```
#define IEEE80211_SCAN_RESULT_EXPIRE (20 * HZ)
```

5g 信道多, 节省 scan 时间 patch.



scan\_patch.zip

## 5G 部分信道不扫描

检查下 `kernel/net/wireless/reg.c` 改为 7。

解法如截图。在 `common/net/wireless/reg.c` 这个文件里面 `reg_rules` 的值由默认的 6 修改为 7 或者 8

```

static const struct ieee80211_regdomain world_regdom = {
    .n_reg_rules = 7,
    .alpha2 = "00",
    .reg_rules = {
        /* IEEE 802.11b/g, channels 1..11 */
        REG_RULE(2412-10, 2462+10, 40, 6, 20, 0),
        /* IEEE 802.11b/g, channels 12..13 */
        REG_RULE(2467-10, 2472+10, 40, 6, 20,
            NL80211_RRF_NO_IR),
        /* IEEE 802.11 channel 14 - Only JP enables
         * this and for 802.11b only */
        REG_RULE(2484-10, 2484+10, 20, 6, 20,
            NL80211_RRF_NO_IR |
            NL80211_RRF_NO_OFDM),
        /* IEEE 802.11a, channel 36..48 */
        REG_RULE(5180-10, 5240+10, 160, 6, 20,
            NL80211_RRF_NO_IR),

        /* IEEE 802.11a, channel 52..64 - DFS required */
        REG_RULE(5260-10, 5320+10, 160, 6, 20,
            NL80211_RRF_NO_IR |
            NL80211_RRF_DFS),

        /* IEEE 802.11a, channel 100..144 - DFS required */
        REG_RULE(5500-10, 5720+10, 160, 6, 20,
            NL80211_RRF_NO_IR |
            NL80211_RRF_DFS),

        /* IEEE 802.11a, channel 149..165 */
        REG_RULE(5745-10, 5825+10, 80, 6, 20,
            NL80211_RRF_NO_IR),

        /* IEEE 802.11ad (60GHz), channels 1..3 */
        REG_RULE(56160+2160*1-1080, 56160+2160*3+1080, 2160, 0, 0, 0),
    }
}

```

amlogic 平台上默认的值是 6，导致 149-165 信道不会被 scan，将 `n_reg_rules` 的值改为 7 或者 8