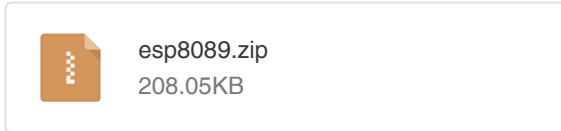


前提：我们经过测试发现，esp8089这颗芯片在V3S上是可以成功使用AP模式的，所以我们可以将V3S上的esp8089的驱动源码拷贝过来进行移植。

准备：V3S上esp8089的驱动源码



1、首先我们将这个驱动源码拷贝到我们的linux源码内，并解压，然后修改上层目录的Kconfig文件和 Makefile 文件，让驱动编译到内核中，

文件路径：[linux/drivers/net/wireless](#)

首先修改Kconfig文件，Kconfig文件是make menuconfig时候解析的文件，我们需要在里面添加我们驱动配置信息，esp8089的驱动源码中也对应着一个Kconfig文件，我们要将这个路径添加到[linux/drivers/net/wireless](#)的Kconfig文件中，修改如下

```
source "drivers/net/wireless/intersil/kconfig"
source "drivers/net/wireless/marvell/kconfig"
source "drivers/net/wireless/mediatek/kconfig"
source "drivers/net/wireless/ralink/kconfig"
source "drivers/net/wireless/realtek/kconfig"
source "drivers/net/wireless/rsi/kconfig"
source "drivers/net/wireless/st/kconfig"
source "drivers/net/wireless/ti/kconfig"
source "drivers/net/wireless/zydas/kconfig"
source "drivers/net/wireless/quantenna/kconfig"
source "drivers/net/wireless/esp8089/kconfig"

config PCMCIA_RAYCS
    tristate "Aviator/Raytheon 2.4GHz wireless support"
    depends on PCMCIA
```

我们查看esp8089的驱动源码中的Kconfig文件我们可以知道，当我们make menuconfig选中我们这个驱动的时候会定义 CONFIG_ESP8089 这个宏，所以我们要在上层的Makefile中将我们的驱动编译进去。

```

config ESP8089
    tristate "Espressif ESP8089 SDIO WiFi"
    depends on MAC80211
    ---help---
    ESP8089 is a low-budget 2.4GHz WiFi chip by Espressif Systems, used in cheap tablets with Allwinner or Rockchip SoC

config ESP8089_DEBUG_FS
    bool "Enable DebugFS support for ESP8089"
    depends on ESP8089
    default y
    ---help---
    DebugFS support for ESP8089

```

我们修改 `linux/drivers/net/wireless` 目录下的 Makefile 文件，当我们的 `CONFIG_ESP8089` 这个宏定义后就编译我们的驱动。

```

#
# Makefile for the Linux wireless network device drivers.
#

obj-$(CONFIG_WLAN_VENDOR_ADMTEK) += admtek/
obj-$(CONFIG_WLAN_VENDOR_ATH) += ath/
obj-$(CONFIG_WLAN_VENDOR_ATMEL) += atmel/
obj-$(CONFIG_WLAN_VENDOR_BROADCOM) += broadcom/
obj-$(CONFIG_WLAN_VENDOR_CISCO) += cisco/
obj-$(CONFIG_WLAN_VENDOR_INTEL) += intel/
obj-$(CONFIG_WLAN_VENDOR_INTERSIL) += intersil/
obj-$(CONFIG_WLAN_VENDOR_MARVELL) += marvell/
obj-$(CONFIG_WLAN_VENDOR_MEDIATEK) += mediatek/
obj-$(CONFIG_WLAN_VENDOR_RALINK) += ralink/
obj-$(CONFIG_WLAN_VENDOR_REALTEK) += realtek/
obj-$(CONFIG_WLAN_VENDOR_RSI) += rsi/
obj-$(CONFIG_WLAN_VENDOR_ST) += st/
obj-$(CONFIG_WLAN_VENDOR_TI) += ti/
obj-$(CONFIG_WLAN_VENDOR_ZYDAS) += zydas/
obj-$(CONFIG_WLAN_VENDOR_QUANTENNA) += quantenna/
obj-$(CONFIG_ESP8089) += esp8089/

# 16-bit wireless PCMCIA client drivers
obj-$(CONFIG_PCMCIA_RAYCS) += ray_cs.o
obj-$(CONFIG_PCMCIA_WL3501) += wl3501_cs.o

obj-$(CONFIG_USB_NET_RNDIS_WLAN) += rndis_wlan.o

obj-$(CONFIG_MAC80211_HWSIM) += mac80211_hwsim.o

```

然后我们进入 `make menuconfig` 将我们的驱动选中

```
.config - Linux/arm 4.15.0-rc8 Kernel Configuration
→ Device Drivers → Network device support → Wireless LAN
Wireless LAN
Arrow keys navigate the menu. <Enter> selects submenus ---> (or empty
submenus ----). Highlighted letters are hotkeys. Pressing <Y>
includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to
exit, <?> for Help, </> for Search. Legend: [*] built-in [ ]

-- Wireless LAN
[ ] ADMtek devices
[ ] Atheros/Qualcomm devices
[ ] Atmel devices
[ ] Broadcom devices
[ ] Cisco devices
[ ] Intel devices
[ ] Intersil devices
[ ] Marvell devices
[ ] MediaTek devices
[ ] Ralink devices
[ ] Realtek devices
[ ] Redpine Signals Inc devices
[ ] STMicroelectronics devices
[ ] Texas Instrument devices
[ ] ZyDAS devices
[ ] Quantenna wireless cards support
<*> Espressif ESP8089 SDIO WiFi
[*] Enable DebugFS support for ESP8089
< > Simulated radio testing tool for mac80211
< > Wireless RNDIS USB support
```

2、编译

make

编译过程中出现了几个问题，

```
drivers/net/wireless/esp8089-cleanup/esp_sip.c: In function 'sip_recalc_credit_init':
drivers/net/wireless/esp8089-cleanup/esp_sip.c:234:2: error: implicit declaration of function 'init_timer'; did you
mean 'init_timers'? [-werror=implicit-function-declaration]
    init_timer(&sip->credit_timer);
    ^
drivers/net/wireless/esp8089-cleanup/esp_sip.c:235:19: error: 'struct timer_list' has no member named 'data'
    sip->credit_timer.data = (unsigned long) sip;
    ^
drivers/net/wireless/esp8089-cleanup/esp_sip.c:236:29: error: assignment from incompatible pointer type [-werror=inc
ompatible-pointer-types]
    sip->credit_timer.function = sip_recalc_credit_timeout;
    ^
drivers/net/wireless/esp8089-cleanup/esp_sip.c: In function 'sip_parse_mac_rx_info':
drivers/net/wireless/esp8089-cleanup/esp_sip.c:1654:22: error: 'RX_FLAG_HT' undeclared (first use in this function);
did you mean 'RX_ENC_HT'?
    rx_status->flag |= RX_FLAG_HT;
    ^
drivers/net/wireless/esp8089-cleanup/esp_sip.c:1654:22: note: each undeclared identifier is reported only once for e
ach function it appears in
drivers/net/wireless/esp8089-cleanup/esp_sip.c:1657:23: error: 'RX_FLAG_SHORT_GI' undeclared (first use in this func
tion); did you mean 'RX_ENC_FLAG_SHORT_GI'?
    rx_status->flag |= RX_FLAG_SHORT_GI;
    ^
drivers/net/wireless/esp8089-cleanup/esp_sip.c:1657:23: note: each undeclared identifier is reported only once for e
ach function it appears in
cc1: some warnings being treated as errors
make[4]: *** [drivers/net/wireless/esp8089-cleanup/esp_sip.o] Error 1
```

首先第一个问题和第二个问题都是由于linux的timer.h这个库和驱动不匹配导致的，

第一个问题是由于没有定义init_timer这个宏导致的，我们通过查看6ULL和V3S的内核源码可以模仿他们进行定义。下图是6ULL的定义的

```

2: #ifdef CONFIG_LOCKDEP
3: #define __init_timer(_timer, _flags) \
4:     do { \
5:         static struct lock_class_key __key; \
5:         init_timer_key((_timer), (_flags), #_timer, &__key); \
7:     } while (0)
3:
3: #define __init_timer_on_stack(_timer, _flags) \
3:     do { \
1:         static struct lock_class_key __key; \
2:         init_timer_on_stack_key((_timer), (_flags), #_timer, &__key); \
3:     } while (0)
4: #else
5: #define __init_timer(_timer, _flags) \
5:     init_timer_key((_timer), (_flags), NULL, NULL)
7: #define __init_timer_on_stack(_timer, _flags) \
3:     init_timer_on_stack_key((_timer), (_flags), NULL, NULL)
3: #endif
3:
1: #define init_timer(timer) \
2:     __init_timer((timer), 0)

```

我们的定义如下：

```

20
21 #define init_timer(timer) \
122     __init_timer((timer), NULL, 0)
23
24 /**
25  * timer_setup - prepare a timer for first use
26  * @timer: the timer in question
27  * @callback: the function to call when timer expires

```

这样第一个问题就解决了，第二个问题是由于内核版本的timer库没有定义data这个成员。

V3S 内核定义的timer_list结构体

```

12: struct timer_list {
13:     /*
14:      * All fields that change during normal runtime grouped to the
15:      * same cacheline
16:      */
17:     struct hlist_node entry;
18:     unsigned long expires;
19:     void (*function)(unsigned long);
20:     unsigned long data;
21:     u32 flags;
22:
23: #ifdef CONFIG_TIMER_STATS
24:     int start_pid;
25:     void *start_site;
26:     char start_comm[16];
27: #endif
28: #ifdef CONFIG_LOCKDEP
29:     struct lockdep_map lockdep_map;
30: #endif
31: } « end timer_list » ;

```

F1C100s内核定义的timer_list结构体

```

13: struct timer_list {
14:     /*
15:      * All fields that change during normal runtime grouped to the
16:      * same cacheline
17:      */
18:     struct hlist_node entry;
19:     unsigned long expires;
20:     void (*function)(struct timer_list *);
21:     u32 flags;
22:
23: #ifdef CONFIG_LOCKDEP
24:     struct lockdep_map lockdep_map;
25: #endif
26: };

```

从上面两个图就可以看出差别，主要就是老版本的timer_list 中定义了data成员变量而新版本的是没有定义的。并且function 这个函数指针类型也变了。我们通过查阅资料知道了 timer_list结构体是用描述定时器的，

- expires : 指定定时器到期的时间
- function: 定时器时间到达后，所要运行的函数
- data: 定时器函数调用的参数

新版本的定时器删掉了data成员变量，并且 function的传入参数由原来的 unsigned long 内心变成了timer_list * 的指针类型。其实这点改动我有点没有明白的，就是如果如果我要执行的函数的参数不是timer_list 指针类型，那么我该如何传参，感觉设计的不是很合理。

知道了上面这些后我们就开始解决问题。首先我们在timer_list中加入data成员变量，这样我们就可以将函数的参数传入进去，这样第三个问题就解决了。

第四成问题是函数类型不匹配，我们上面说了 新版本的function的函数类型为
void (*function)(struct timer_list *);

但是我们定时器所调用的函数类型定义为

void (*function)(unsigned long);

所以需要修改sip_recalc_credit_timeout函数的类型

```

220 static void sip_recalc_credit_timeout(unsigned long data)
221 {
222     struct esp_sip *sip = (struct esp_sip *) data;
223
224     esp_dbg(ESP_DBG_ERROR, "ret");
225
226     sip_recalc_credit_claim(sip, 1);/* recalc again */
227 }
228
229 static void sip_recalc_credit_init(struct esp_sip *sip)
230 {
231     atomic_set(& sip->credit_status, RECALC_CREDIT_DISABLE); //set it disable
232
233     init_timer(& sip->credit_timer);
234     sip->credit_timer.data = (unsigned long) sip;
235     sip->credit_timer.function = sip_recalc_credit_timeout;
236 }
237

```

修改后的如下

```

220 static void sip_recalc_credit_timeout(struct timer_list *time)
221 {
222     struct esp_sip *sip = (struct esp_sip *) time->data;
223
224     esp_dbg(ESP_DBG_ERROR, "ret");
225
226     sip_recalc_credit_claim(sip, 1);/* recalc again */
227 }
228

```

新版本的timer的调用传参的形式是传递的timer_list结构体，所以我们要从结构体里面把我们的函数参数取出来，具体如上。这样第4个问题就解决了。

第5个问题，是由于我们的内核版本高了后，里面的库有所改动，这个原因是mac80211.h这个库 去掉了RX_FLAG_HT和RX_FLAG_SHORT_GI 这两个标志。

```

1652     if (mac_ctrl->sig_mode) {
1653         rx_status->flag |= RX_FLAG_HT;
1654         rx_status->rate_idx = mac_ctrl->MCS;
1655         if (mac_ctrl->SGI)
1656             rx_status->flag |= RX_FLAG_SHORT_GI;
1657     } else {
1658         rx_status->rate_idx = esp_wmac_rate2idx(mac_ctrl->rate);
1659     }
1660     if (mac_ctrl->rxend_state == RX_FCS_ERR)
1661         rx_status->flag |= RX_FLAG_FAILED_FCS_CRC;
1662

```

首先说一下 mac80211是linux kernel中的一个子系统，它为无线设备soft-MAC/half-MAC提供了分享实施方案，所以这个地方是一个比较棘手的地方，但是幸好老版本内核中带了很多WIFI 驱动，所以我们可以对比修改这个，下图是我们修改的后的。

```

if (mac_ctrl->sig_mode) {
/*
    rx_status->flag |= RX_FLAG_HT;
    rx_status->rate_idx = mac_ctrl->MCS;
    if (mac_ctrl->SGI)
        rx_status->flag |= RX_FLAG_SHORT_GI;
*/
} else {
    rx_status->rate_idx = esp_wmac_rate2idx(0xc);
    rx_status->rate_idx = esp_wmac_rate2idx(mac_ctrl->rate);
}
if (mac_ctrl->rxend_state == RX_FCS_ERR)
    rx_status->flag |= RX_FLAG_FAILED_FCS_CRC;

```

到此这个问题也解决了。

接下来我们解决最后一个问题也是最重要的一个点，我们的整个AP不能启动就是因为这个函数没有正常调用，这个问题和第四个问题一样，所以我们修改后如下。

```

static void init_beacon_timer(struct ieee80211_vif *vif)
{
    struct esp_vif *evif = (struct esp_vif *) vif->drv_priv;

    ESP_IEEE80211_DBG(ESP_DBG_OP, " %s enter: beacon interval %x\n",
        __func__, evif->beacon_interval);

    beacon_tim_init();
    init_timer(&evif->beacon_timer); //TBD, not init here...
    cycle_beacon_count = 1;
    init_jiffies = jiffies;
    evif->beacon_timer.expires =
        init_jiffies +
        msecs_to_jiffies(cycle_beacon_count *
            vif->bss_conf.beacon_int * 1024 / 1000);
    evif->beacon_timer.data = (unsigned long) vif;
    evif->beacon_timer.function = drv_handle_beacon;
    add_timer(&evif->beacon_timer);
}

```



```

static void drv_handle_beacon(struct timer_list *timer)
{
    struct ieee80211_vif *vif = (struct ieee80211_vif *) timer->data;
    struct esp_vif *evif = (struct esp_vif *) vif->drv_priv;
    struct sk_buff *beacon;
    struct sk_buff *skb;
    static int dbgcnt = 0;
    bool tim_reach = false;

    if (evif->epub == NULL)
        return;

    mdelay(2400 * (cycle_beacon_count % 25) % 10000 / 1000);
    beacon = ieee80211_beacon_get(evif->epub->hw, vif);
    tim_reach = beacon_tim_alter(beacon);

    if (beacon && !(dbgcnt++ % 600)) {
        ESP_IEEE80211_DBG(ESP_DBG_TRACE, " beacon length:%d,fc:0x%x\n",
            beacon->len,
            ((struct ieee80211_mgmt *) (beacon->
                data))->
                frame_control);
    }

    if (beacon)
        sip_tx_data_pkt_enqueue(evif->epub, beacon);

    if (cycle_beacon_count++ == 100) {
        init_jiffies = jiffies;
        cycle_beacon_count -= 100;
    }
    mod_timer(&evif->beacon_timer,
        init_jiffies +
        msecs_to_jiffies(cycle_beacon_count *
            vif->bss_conf.beacon_int * 1024 /
            1000));
    //FIXME:the packets must be sent at home channel
    //send buffer mcast frames
    if (tim_reach) {
        skb = ieee80211_get_buffered_bc(evif->epub->hw, vif);
        while (skb) {
            sip_tx_data_pkt_enqueue(evif->epub, skb);
            skb =
                ieee80211_get_buffered_bc(evif->epub->hw, vif);
        }
    }
}

```

到此所以的移植问题都解决了，AP模式可以正常使用。

总结：驱动的移植过程中一定要对比老版本和新版本之前的差异，并且我们要通过对比的方式去排错，这样我们可以缩小问题点，通过不断的修改V3S上的源代码，来测试看我们这几个驱动移植的点是哪个影响了AP模式的使用，最终定位到最后一个问题。其次就是查看V3S内部运行过程中一些关键的点，比如我们WIFI运行的时候会进入AP模式，其中定义就在nl80211.h中定义的，当我们这个标志位NL80211_IFTYPE_AP的时候WIFI开启AP模式，这样我们就可以找到整个驱动执行的流程。


```

enum nl80211_iftype {
    NL80211_IFTYPE_UNSPECIFIED,
    NL80211_IFTYPE_ADHOC,
    NL80211_IFTYPE_STATION,
    NL80211_IFTYPE_AP,
    NL80211_IFTYPE_AP_VLAN,
    NL80211_IFTYPE_WDS,
    NL80211_IFTYPE_MONITOR,
    NL80211_IFTYPE_MESH_POINT,
    NL80211_IFTYPE_P2P_CLIENT,
    NL80211_IFTYPE_P2P_GO,
    NL80211_IFTYPE_P2P_DEVICE,
    NL80211_IFTYPE_OCB,
    NL80211_IFTYPE_NAN,

    /* keep last */
    NUM_NL80211_IFTYPES,
    NL80211_IFTYPE_MAX = NUM_NL80211_IFTYPES - 1
};

```

```

523:     }
524: } « end if vif->type==NL80211_IF... » else if (vif->type == NL80211_IFTYPE_AP) {
525:     if ((changed & BSS_CHANGED_BEACON_ENABLED) ||
526:         (changed & BSS_CHANGED_BEACON_INT)) {
527:         ESP_IEEE80211_DBG(ESP_DBG_TRACE,
528:             "%s AP change enable %d, interval is %d, bssid %pM\n",
529:             __func__, info->enable_beacon,
530:             info->beacon_int, info->bssid);
531:         if (info->enable_beacon && evif->ap_up != true) {
532:             evif->beacon_interval = info->beacon_int;
533:             init_beacon_timer(vif);
534:             sip_send_bss_info_update(epub, evif,
535:                 (u8 *) info->
536:                 bssid, 2);
537:             evif->ap_up = true;
538:         } else if (!info->enable_beacon && evif->ap_up &&
539:             !(hw->conf.flags & IEEE80211_CONF_OFFCHANNEL)
540:             ) {
541:             ESP_IEEE80211_DBG(ESP_DBG_TRACE,
542:                 "%s AP disable beacon, interval is %d\n",
543:                 __func__,
544:                 info->beacon_int);
545:             evif->beacon_interval = 0;
546:             del_timer_sync(&evif->beacon_timer);
547:             sip_send_bss_info_update(epub, evif,
548:                 (u8 *) info->
549:                 bssid, 2);
550:             evif->ap_up = false;
551:         }
552:     } « end if (changed&BSS_CHANGED... »
553: } « end if vif->type==NL80211_IF... » else {
554:     ESP_IEEE80211_DBG(ESP_DBG_ERROR,
555:         "%s op mode unspecified\n", __func__);
556: }
557: } « end esp_op_bss_info_changed »
558:

```