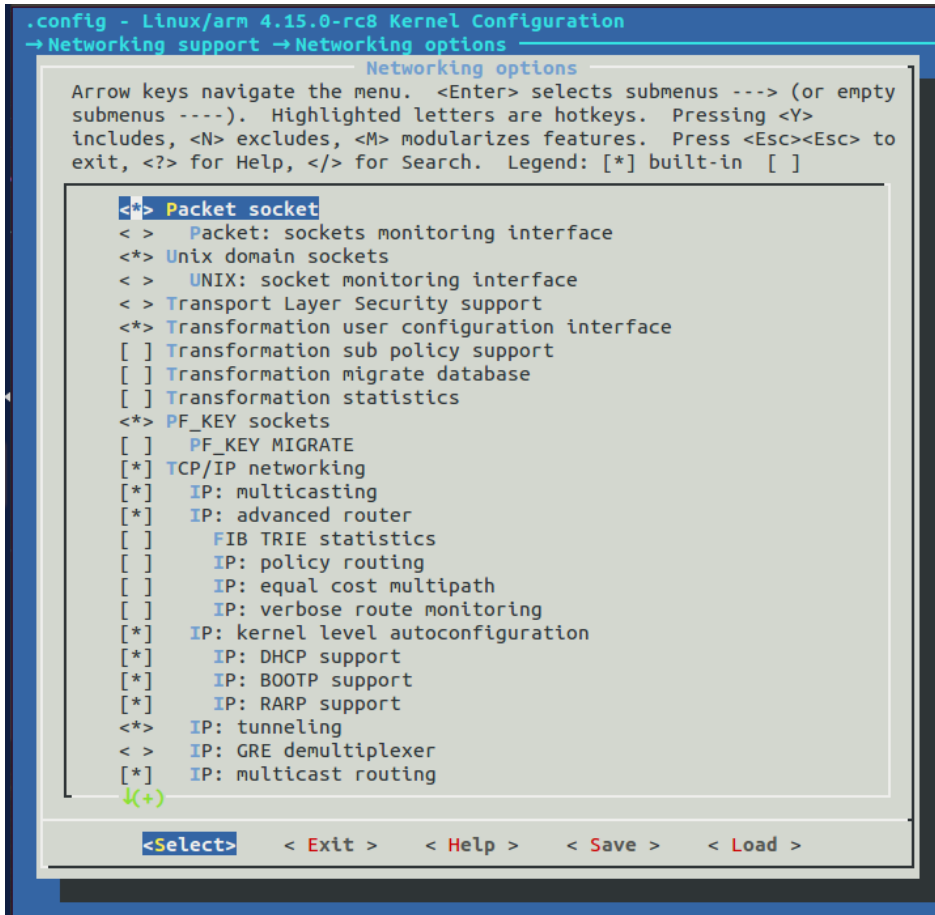
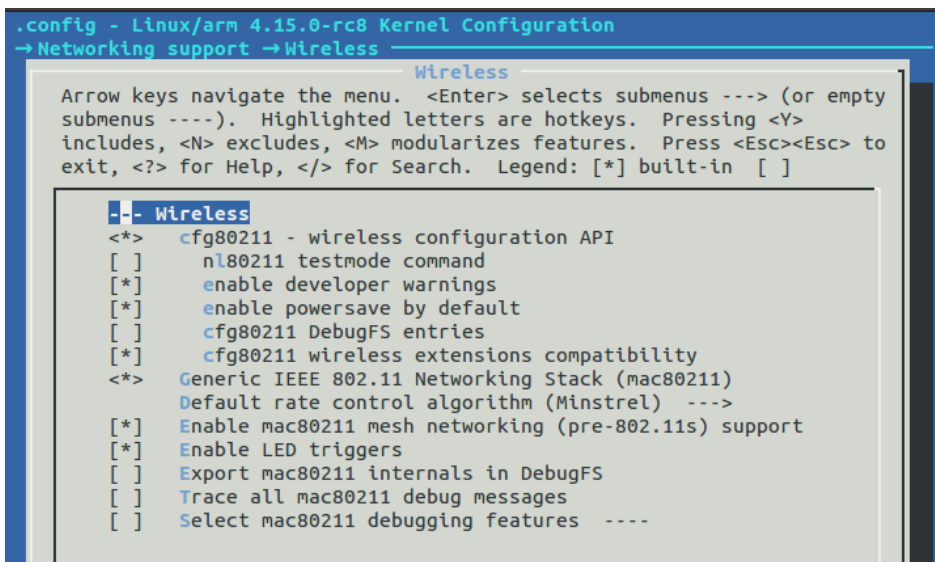


1 WIFI内核配置

网络相关配置



Wireless配置



SDIO接口驱动

```
.config - Linux/arm 4.15.0-rc8 Kernel Configuration
→ Device Drivers → MMC/SD/SDIO card support
    MMC/SD/SDIO card support
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 [ ]

--- MMC/SD/SDIO card support
<*> HW reset support for eMMC
<*> Simple HW reset support for MMC
<*> MMC block device driver
(8)  Number of minors per block device
<*> SDIO UART/GPS class support
< > MMC host test driver
    *** MMC/SD/SDIO Host Controller Drivers ***
[ ]  MMC host drivers debugging
< > Secure Digital Host Controller Interface support
< > MMC/SD/SDIO over SPI
< > Synopsys DesignWare Memory Card Interface
< > VUB300 USB to SDIO/SD/MMC Host Controller support
< > USB SD Host Controller (USHC) support
< > Renesas USDHI6ROL0 SD/SDIO Host Controller support
<*> Allwinner sunxi SD/MMC Host Controller support
< > MediaTek SD/MMC Card Interface support
```

WIFI驱动

```
.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 WIFI驱动编译

2.1 将ESP8089驱动源码放入linux源码中的drivers/net/wireless, 进行解压。



esp8089-cleanup.zip
217.65KB

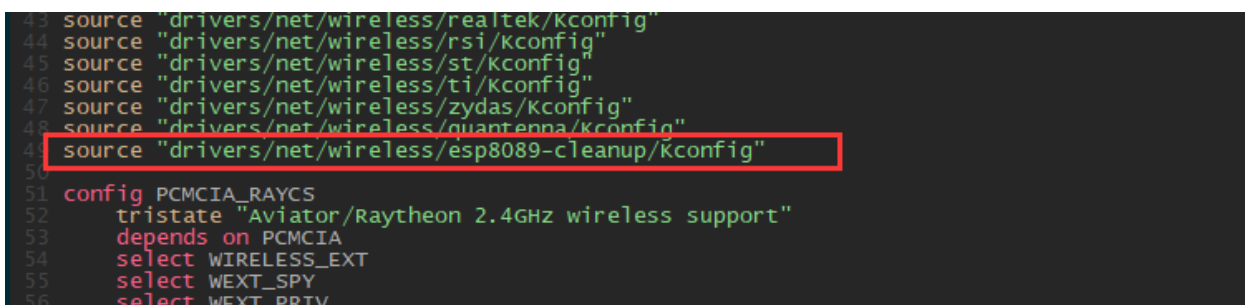
```
cd ./drivers/net/wireless
unzip esp8089-cleanup.zip
```

2.2 将驱动选项添加至配置界面中

添加配置选项至wireless中的Kconfig文件中，这样在make menuconfig的时候就可以看到ESP8089的选项

```
vi ./drivers/net/wireless/Kconfig
```

```
source "drivers/net/wireless/esp8089-cleanup/Kconfig"
```



```
43 source "drivers/net/wireless/realtek/Kconfig"
44 source "drivers/net/wireless/rsi/Kconfig"
45 source "drivers/net/wireless/st/Kconfig"
46 source "drivers/net/wireless/ti/Kconfig"
47 source "drivers/net/wireless/zydas/Kconfig"
48 source "drivers/net/wireless/quantenna/Kconfig"
49 source "drivers/net/wireless/esp8089-cleanup/Kconfig"
50
51 config PCMCIA_RAYCS
52     tristate "Aviator/Raytheon 2.4GHz wireless support"
53     depends on PCMCIA
54     select WIRELESS_EXT
55     select WEXT_SPY
56     select WEXT_PRTV
```

2.3 添加编译选项

将添加的驱动关联到源码中，打开wireless目录中的Makefile文件

```
vi Makefile
```

添加驱动路径

```
obj-$(CONFIG_ESP8089) += esp8089-cleanup/
```

也就是当CONFIG_ESP8089为y的时候将驱动编译进内核。当配置选项选中的时候CONFIG_ESO8089为y

```
Makefile
1 # SPDX-License-Identifier: GPL-2.0
2 #
3 # Makefile for the Linux wireless network device drivers.
4 #
5
6 obj-$(CONFIG_WLAN_VENDOR_ADMTEK) += admtek/
7 obj-$(CONFIG_WLAN_VENDOR_ATH) += ath/
8 obj-$(CONFIG_WLAN_VENDOR_ATMEL) += atmel/
9 obj-$(CONFIG_WLAN_VENDOR_BROADCOM) += broadcom/
10 obj-$(CONFIG_WLAN_VENDOR_CISCO) += cisco/
11 obj-$(CONFIG_WLAN_VENDOR_INTEL) += intel/
12 obj-$(CONFIG_WLAN_VENDOR_INTERSIL) += intersil/
13 obj-$(CONFIG_WLAN_VENDOR_MARVELL) += marvell/
14 obj-$(CONFIG_WLAN_VENDOR_MEDIATEK) += mediatek/
15 obj-$(CONFIG_WLAN_VENDOR_RALINK) += ralink/
16 obj-$(CONFIG_WLAN_VENDOR_REALTEK) += realtek/
17 obj-$(CONFIG_WLAN_VENDOR_RSI) += rsi/
18 obj-$(CONFIG_WLAN_VENDOR_ST) += st/
19 obj-$(CONFIG_WLAN_VENDOR_TI) += ti/
20 obj-$(CONFIG_WLAN_VENDOR_ZYDAS) += zydas/
21 obj-$(CONFIG_WLAN_VENDOR_QUANTENNA) += quantenna/
22 obj-$(CONFIG_ESP8089) += esp8089-cleanup/
23
24
25 # 16-bit wireless PCMCIA client drivers
26 obj-$(CONFIG_PCMCIA_RAYCS) += ray_cs.o
27 obj-$(CONFIG_PCMCIA_WL3501) += wl3501_cs.o
28
29 obj-$(CONFIG_USB_NET_RNDIS_WLAN) += rndis_wlan.o
30
31 obj-$(CONFIG_MAC80211_HWSIM) += mac80211_hwsim.o
32
33
```

```
192.168.188.88 x
kconfig
1 config ESP8089
2     tristate "Espressif ESP8089 SDIO WiFi"
3     depends on MAC80211
4     ---help---
5     ESP8089 is a low-budget 2.4GHz WiFi chip by Espressif, used in many
6     cheap tablets with Allwinner or Rockchip soc
7
8 config ESP8089_DEBUG_FS
9     bool "Enable DebugFS support for ESP8089"
10    depends on ESP8089
11    default y
12    ---help---
13    DebugFS support for ESP8089
```

2.4配置WIFI

内核配置选择ESP8089如1中的图4所示。

3 编译驱动模块

make moudles

出现以下问题，这个是因为当前linux源码中所对应的linux/timer.h库版本有问题，由于当前的timer.h库中对应的没有data这个变量也没有对init_timers的声明，所以导致了这样的问题出现。我们需要去手动解决这些问题。

```
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
compatible-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
```

首先解决第一个问题，也就是timer中没有定义init_timer函数，我们通过查看6ULL的timer.h中发现 这个版本的timer中定义了，我们可以模仿着修改F1C100S中的timer.h文件，我们在通过下图可以看到6ULL中的定义如下：

```
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:
9: #define __init_timer_on_stack(timer, flags) \
0:     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) \
9: #endif
0:
1: #define init_timer(timer) \
2:     __init_timer((timer), 0)
```

通过查看我们将在timer.h中定义该函数 具体如下：

```
20
21 #define init_timer(timer) \
22     __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
```

接下来解决第二个，由于我们的timer.h中的timer_list结构体中没有定义data变量，所以我们需要查看这个值是如何使用的，这个data通过查看代码 发现在驱

动程序中只是通过该变量传递spi结构体的地址值，所以我们在timer.h中添加

```
12 struct tvec_base;
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     unsigned long data;
23 #ifdef CONFIG_LOCKDEP
```

另外还需要修改传递函数，由于timer_list中function是timer_list为结构体变量，

所以我们需要将sip_recalc_credit_timeout函数更改为下图的形式。

```
9: #include <linux/stringify.h>
10:
11: struct tvec_base;
12:
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: };
27:
28: #ifdef CONFIG_LOCKDEP
29: /*
```

```
219:
220: static void sip_recalc_credit_timeout(struct timer_list * 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:
```

这样这个问题也解决了

最后三个问题，提示没有这两个宏，我们通过查找源代码发现也是由于库函数不同导致的这个问题，我们还是和刚刚一样通过观察源代码，然后将其进行替换，解决以上问题。

替换前

```

rx_status->band = NL80211_BAND_2GHZ;
rx_status->flag = RX_FLAG_DECRYPTED | RX_FLAG_MMIC_STRIPPED;
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 {

```

替换后

最后一个问题，提示函数类型不匹配，通过观察函数功能，然后我们更改函数如下

更改前

```

392 static void drv_handle_beacon(unsigned long data)
393 {
394     struct ieee80211_vif *vif = (struct ieee80211_vif *) data;
395     struct esp_vif *evif = (struct esp_vif *) vif->drv_priv;
396     struct sk_buff *beacon;
397     struct sk_buff *skb;
398     static int dbgcnt = 0;
399     bool tim_reach = false;
400
401     if (evif->epub == NULL)
402         return;
403
404     mdelay(200 * (cycle_beacon_count % 25) % 1000 / 100);

```

编译通过

接下来就是测试是否可以成功加载驱动，我们将编译好的驱动放到文件系统中
cp esp8089.ko /root

将固件放到文件系统中的lib目录中，

另外WIFI连接过程中需要一些工具，我们需要将工具添加进去，其中我们用到的工具有

- Wireless_tools
- wpa_supplicant
- openssl
- iw

我们在Buildroot中打开相关选项，
make menuconfig

-->Target packages

-->Networking applications

-->iw

-->>wireless tools

-->>wpa_supplicant

-->Install wpa_passphrase binary

-->Install wpa_client shared library

-->Install wpa_cli binary

打开相关的设置后编译新的文件系统,

烧录文件后 测试发现可以加载esp8089.ko, 也可以出现wlan0 网卡, 但是无法连接网络。因此我们需要继续移植。

首先我们先打开驱动的调试信息, 查看源码发现, 只有当printk的等级为ERROR或者SHOW的时候会显示, 但是貌似打印的时候都是通过TRACE登记打印的, 所以需要再次添加改登记

```
00068: //unsigned int esp_msg_level = 0;
00069: unsigned int esp_msg_level = ESP_DBG_ERROR | ESP_SHOW;
00070:
00071: struct esp_dbg_ctl *if_ctl = NULL;

00073: #include "esp_file.h"
00074: #define esp_dbg(mask, fmt, args...) do {
00075:     if (esp_msg_level & mask)
00076:     {
00077:         if (log_off)
00078:             printk(fmt, ##args);
00079:         else
00080:             logger_write(4, "esp_wifi", fmt, ##args);
00081:     }
00082: } while (0)
00083: #else
00084: #define esp_dbg(mask, fmt, args...) do {
00085:     if (esp_msg_level & mask)
00086:         printk(fmt, ##args);
00087: } while (0)
00088: #endif /* ESP_ANDROID_LOGGER */
00089:
```


打印发现没有发现有价值的，只能从荔枝派上能用的着手，发现我们之前加载网卡

时用的insmod命令，如果用modprobe命令就会提示说不存在4.15.0-rc8-licheepi-nano+目录，然后我看demo板上在lib/modules/下有这个目录，这个目录是存放驱动的地方，我们通过查看esp8089驱动源码中的Makefile，发现它会在下面目录下找驱动，

```
/lib/modules/$(shell uname -r)/build
```

```
5 # version from the kernel sources.
6 KVERS_UNAME ?= $(shell uname -r)
7
8 # KBUILD is the path to the Linux kernel build tree. It is usually the
9 # same as the kernel source tree, except when the kernel was compiled in
10 # a separate directory.
11 KBUILD ?= $(shell readlink -f /lib/modules/$(KVERS_UNAME)/build)
12
13 # ... / $(KBUILD) \
```

uname -r 是当前系统的版本号，我们在板子上输入打印出来的就是

```
serial-com12 x
# uname -r
4.15.0-rc8-licheepi-nano+
#
```

4.15.0-rc8-licheepi-nano+

modprobe在加载驱动的时候会根据modules.dep查找依赖关系，但是通过insmod加载不会解决驱动的依赖关系，所以大概知道了问题所在。我们要解决这个问题，

首先在编译内核后，编译出驱动

```
make modules_install
```

```
root@ubuntu-virtual-machine:/home/ubuntu/zj/software/F1C100S/linux# make modules_install
INSTALL crypto/echainiv.ko
INSTALL drivers/input/touchscreen/goodix.ko
INSTALL drivers/net/mii.ko
INSTALL drivers/net/usb/rt18150.ko
INSTALL drivers/net/wireless/esp8089-cleanup/esp8089.ko
INSTALL drivers/staging/rt18723bs/r8723bs.ko
INSTALL drivers/usb/mon/usbmon.ko
INSTALL drivers/video/backlight/lcd.ko
DEPMOD 4.15.0-rc8-licheepi-nano+
root@ubuntu-virtual-machine:/home/ubuntu/zj/software/F1C100S/linux#
```

这个时候我们在根目录下的/lib/modules/下找到我们生产的文件

```
root@ubuntu-virtual-machine:/lib/modules# ls
4.15.0-rc8-licheepi-nano+ 4.4.0-31-generic
root@ubuntu-virtual-machine:/lib/modules#
```

把这个目录放入我们Buildroot生成的文件系统中，放到lib/moudles/下，然后通过脚本生成下载文件。这个地方要避免一个坑，就是脚本生成下载文件的时候会将你准备好的文件系统的modules目录删除掉，然后加入自己的，我们把几句话注释掉。

```
34 dd if=/dev/zero of=flashimg.bin bs=1k count=10 &&\
35 echo_log "Packing Uboot..."
36 dd if=${UBOOT_FILE} of=flashimg.bin bs=1K conv=notrunc &&\
37 echo_log "Packing dtb..."
38 dd if=${DTB_FILE} of=flashimg.bin bs=1K seek=1024 conv=notrunc &&\
39 echo_log "Packing zImage..."
40 cp ${KERNEL_FILE} ./zImage &&\
41 dd if=./zImage of=flashimg.bin bs=1K seek=1088 conv=notrunc &&\
42 mkdir rootfs
43 echo_log "Packing rootfs..."
44 tar -xzvf ${ROOTFS_FILE} -C ./rootfs >/dev/null &&\
45 cp -r ${MOD_FILE} rootfs/lib/modules/ &&\
46 mkfs.jffs2 -s 0x100 -e 0x10000 --pad=0xAF0000 -d rootfs/ -o jffs2.img &&\
47 dd if=jffs2.img of=flashimg.bin bs=1K seek=5184 conv=notrunc &&\
48 mv ./flashimg.bin ${IMG_FILE} &&\
49 echo_log "Bin update done!"
50 cd .. &&\
51 rm -rf ./_temp
52 echo_tip "You configure your LCD parameters as ${SCREEN_PRAM}"
53 echo_log "Pack ${IMG_FILE} finished"
54 exit
```

然后下载到板子上，这样我们就可以通过modprobe 命令加载esp8089.ko驱动了，但是还是无法使用!!! 这个时候我们读驱动上的README.md，发现如下

```
Must load mac80211.ko first if not baked in.

sudo modprobe esp8089.ko

If you get a wlan interface, but scanning shows no networks try using:

sudo modprobe esp8089.ko config=crystal_26M_en=1

or:

sudo modprobe esp8089.ko config=crystal_26M_en=2

To load the module.
```

通过阅读README.md 我们发现加载驱动的时候如果无法扫描WIFI，我们可以通过在加载的时候加上config=crystal_26M_en=1 配置一下 就可以了，

我们试一下

```
modprobe esp8089.ko config=crystal_26M_en=1
```

这个时候成功出现了wlan0 网口，我们开启wlan0网口

ifconfig wlan0 up

接下来配置网络配置文件

vi /etc/wpa_supplicant.conf

```
ctrl_interface=/var/run/wpa_supplicant
ap_scan=1

network={
  ssid="360HCFA"
  key_mgmt=WPA-PSK
  proto=WPA2
  pairwise=CCMP
  psk="HCFahcfa"
}
~
~
~
```

```
1 ctrl_interface=/var/run/wpa_supplicant
2 ap_scan=1
3
4 network={
5   ssid="360HCFA"
6   key_mgmt=WPA-PSK
7   proto=WPA2
8   pairwise=CCMP
9   psk="HCFahcfa"
10 }
```

连接网络

```
wpa_supplicant -B -i wlan0 -c /etc/wpa_supplicant.conf
```

分配IP和网关

```
udhcpc -i wlan0
```

测试

ping一下www.baidu.com

成功，到此ESP8089驱动移植完毕。

总结：

在移植WIFI驱动的过程中我们不必去更改驱动本身，因为驱动本身是别人验证过的，我们只需要去配置和编译好驱动，然后打开内核中的相关选项，然后移植相关的工具即可。另外我们编译驱动后一定要通过 `make modules_install` 命

令生成驱动，然后把整个生成的驱动放到文件系统中，然后通过modprobe命令去加载，因为该命令可以读取modules.dep文件找到驱动的依赖关系。