APPLICATION NOTE

ATBM IOCTL 控制接口使用说明

ATBM603X

1x1 802.11b/g/n Wi-Fi 芯片

Table of contents

AN9310
Doc Rev: 1.4
Released:2022-01-26

Version	Date	Author	Contents
001	20190918	Altobeam	Draft version.
	20200610	Yuzhihuang	修改了获取最优信道的参数
	20200703	Yuzhihuang	增加过滤特定帧的功能
	20210423	Yuzhihuang	增加 ETF 相关功能，增加获取版本号、efsue 功能，国家码信道扫描限制功能
	20220126	Yuzhihuang	修正设置功率命令

数据类型说明：

struct altm_wext_msg 结构体用于应用层与 wifi 驱动的数据交互。

```
struct altm_wext_msg{
    int type;
    int value;
    char externData[256];
};
```

type----具体定义如下：

```
enum atbm_msg_type{
        ATBM_DEV_IO_GET_STA_STATUS      = 0,
        ATBM_DEV_IO_GET_STA_RSSI        = 1, //STA connected AP's RSSI
        ATBM_DEV_IO_GET_AP_INFO         = 2,    //STA or AP
        ATBM_DEV_IO_GET_STA_INFO        = 3,
        ATBM_DEV_IO_SET_STA_SCAN        = 4,
        ATBM_DEV_IO_SET_FREQ            = 5,
        ATBM_DEV_IO_SET_SPECIAL_OUI     = 6, //use for Beacon and Probe package
        ATBM_DEV_IO_SET_STA_DIS         = 7,
        ATBM_DEV_IO_SET_IFTYPE          = 8,
        ATBM_DEV_IO_SET_ADAPTIVE        = 9,
        ATBM_DEV_IO_SET_TXPWR_DCXO      = 10,
        ATBM_DEV_IO_SET_TXPWR           = 11,
        ATBM_DEV_IO_GET_WORK_CHANNEL    = 12,
        ATBM_DEV_IO_SET_BEST_CHANNEL_SCAN    = 13,
        ATBM_DEV_IO_GET_AP_LIST     = 14,
        ATBM_DEV_IO_GET_TP_RATE     = 15,
        ATBM_DEV_IO_FIX_TX_RATE         = 22,
        ATBM_DEV_IO_MAX_TX_RATE     = 23,
    ATBM_DEV_IO_TX_RATE_FREE    = 24,
    ATBM_DEV_IO_SET_EFUSE_MAC       = 25,
    ATBM_DEV_IO_SET_EFUSE_DCXO      = 26,
    ATBM_DEV_IO_SET_EFUSE_DELTAGAIN = 27,
    ATBM_DEV_IO_MIN_TX_RATE         = 28,
    ATBM_DEV_IO_SET_RATE_POWER      = 29,
    ATBM_DEV_IO_SET_SPECIAL_FILTER = 30,
    ATBM_DEV_IO_SET_COUNTRY_CODE = 31,

        ATBM_DEV_IO_GET_DRIVER_VERSION = 32,
        ATBM_DEV_IO_GET_EFUSE           = 33,
        ATBM_DEV_IO_GET_ETF_START_RX_RESULTS = 34,
};
```

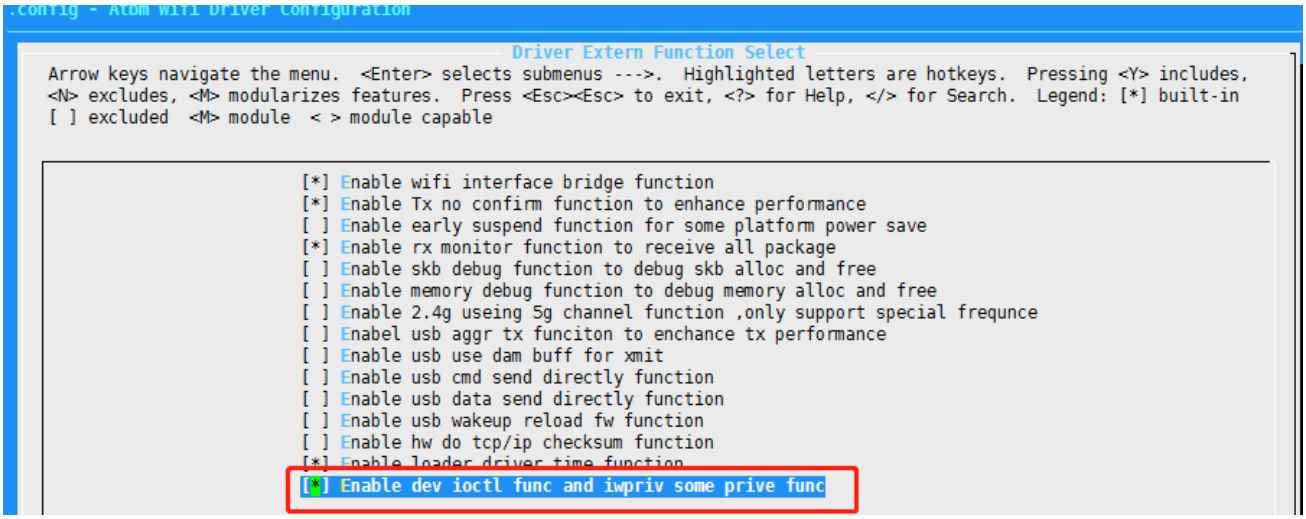
value----预留参数；

externData----传递数据。

一、Device IOCTRL 接口

概述：

通过 SIOCDEVPRIVATE，用于应用层与 Wi-Fi 驱动的交互。
改功能需要打开驱动的 PRIVATE_IE 宏 15323 以上的版本驱动需要选上：



驱动版本低于 1532，需要在打开 hal_apollo/Makefile 里面打开宏：

```
ifeq ($(USB_BUS),y)
ccflags-y += -DUSB_BUS
ccflags-y += -DUSB_USE_TASTLET_TXRX
ccflags-y += -DUSB_BUS_BUG
ccflags-y += -DCONFIG_USB_URB_RX_SUBMIT_DIRECTLY
#ccflags-y += -DCONFIG_WSM_CMD_XMIT_DIRECTLY
#ccflags-y += -DCONFIG_USB_DATA_XMIT_DIRECTLY
#ccflags-y += -DCONFIG_USE_DMA_ADDR_BUFFER
#ccflags-y += -DCONFIG_USB_AGGR_URB_TX
#ccflags-y += -DCONFIG_TX_NO_CONFIRM
#ccflags-y += -DATBM_IPC_ASSOC
#ccflags-y += -DATBM_PRIVATE_IE
#ccflags-y += -DSIGMASTAR_FILTER_MACADDR_ONLY
#ccflags-y += -DCONFIG_ATBM_5G_PRETEND_2G
endif
```

1 获取当前 station 的连接状态

Type ID	ATBM_DEV_IO_GET_STA_STATUS
输入参数	无
返回值	1 类型：bool 2 数据：externData[0:3] 3 说明：0（disconnect） 1（associated）

例子	{
	struct altm_wext_msg msg;
	memset(&msg, 0, sizeof(msg));

	msg.type = ATBM_DEV_IO_GET_STA_STATUS;
	ioctrl(..., &msg, ...);
	}

2 获取当前 station 连接 AP 的信号强度 RSSI

Type ID	ATBM_DEV_IO_GET_STA_RSSI
输入参数	无
返回值	1 类型：int 2 数据：externData[0:3] 3 说明：收到最后一包数据的 RSSI 值

例子	{
	struct altm_wext_msg msg;
	int rssi = 0;
	memset(&msg, 0, sizeof(msg));
	msg.type = ATBM_DEV_IO_GET_STA_RSSI;
	
	ioctrl(..., &msg, ...);
	
	memcpy(&rssi, &msg.externData[0], sizeof(int));
	}

3 获取 AP 模式下，所有已经连接的 Station 信息

Type ID	ATBM_DEV_IO_GET_AP_INFO
输入参数	1 类型：char * 2 数据：externData[0:3] 3 说明：四个字节的内存地址，传给 wifi 驱动，用于保存所有 STA 信息。

返回值	1 类型：
	typedef struct _atbm_wifi_ap_info_{
	int wext_rssi; //信号强度
	unsigned long rx_packets; //收包数量
	unsigned long tx_packets; //发包数量
	unsigned long tx_retry_count; //重传次数
	int last_rx_rate_idx; //收包速率
	unsigned char wext_mac[6]; //Station 的 MAC 地址
	unsigned char sta_cnt; //已经连接的 AP 个数
	}atbm_wifi_ap_info;
	2 数据：数据 copy 到输入参数的内存地址中
	3 说明：
	因为，AP 模式下，最多支持 14 个 STA 的连接。

	所以，STA 信息的 total size 大于 256 bytes，不用 externData 传送数据。

例子	{
	struct altm_wext_msg msg;
	atbm_wifi_ap_info atbm_ap_info[14];
	int addr_val = (int)(&atbm_ap_info[0]);
	
	memset(&msg, 0, sizeof(msg));
	msg.type = ATBM_DEV_IO_GET_AP_INFO;
	memcpy(&msg.externData[0], &addr_val, sizeof(int));
	
	ioctrl(..., &msg, ...);
	}

4 获取 STA 模式下，已经连接 AP 的信息

Type ID	ATBM_DEV_IO_GET_STA_INFO
输入参数	无

返回值	1 类型：
	typedef struct _atbm_wifi_sta_info_{
	int rssi; //信号强度
	unsigned long rx_packets; //收包数量
	unsigned long tx_packets; //发包数量
	unsigned long tx_retry_count; //重传次数
	unsigned char bssid[6]; //AP 的 MAC 地址
	unsigned char ssid[32]; //AP 的 SSID
	unsigned long ssid_len; //AP 的 SSID 长度
	}atbm_wifi_sta_info;
	2 数据：externData[0:N]
	--N 是数据类型的长度
	3 说明：

例子	{
	struct altm_wext_msg msg;
	atbm_wifi_sta_info atbm_sta_info;
	
	memset(&msg, 0, sizeof(msg));
	msg.type = ATBM_DEV_IO_GET_STA_INFO;
	ioctrl(..., &msg, ...);
	memcpy((u8*)&atbm_sta_info, (u8*)msg.externData,
	sizeof(atbm_sta_info));
	}

5 获取 STA 模式下，启动私有扫描

Type ID	ATBM_DEV_IO_SET_STA_SCAN
输入参数	参数一 1 类型：unsigned short 2 数据：externData[0:1] 3 说明：信道号（1~13，0：全信道扫描） 参数二 1 类型：unsigned char filter_mac[8][6]; 2 数据：externData[2:49] 3 说明：现在支持 8 组 MAC 设定，其它 MAC 地址的 AP 将会被过滤掉； 如果不设置 MAC 地址，驱动将不会进行过滤。 注：如果参数二为空（即，不设置 MAC 地址），wifi 驱动将不会根据 MAC 地址对扫描到的 AP 进行过滤。 参数三 1 类型：char * 2 数据：externData[50:53] 3 说明：四个字节的内存地址，传给 wifi 驱动，用于保存扫描结果。
返回值	1 类型： typedef struct _Wifi_Recv_Info { unsigned char Ssid[32]; unsigned char channel; unsigned int Rssi; unsigned int Quality; unsigned char phy_noise; unsigned char Bssid[6]; unsigned char User_data[USER_DATE_LEN+1]; }Wifi_Recv_Info_t; 2 数据：数据 copy 到输入参数的内存地址中。 3 说明： 扫描到的 AP 会首先判断是否包含特殊数据字段，如果不包含特殊字段将被过滤掉，其次再根据扫描的 channel 进行过滤，最后根据设定的 MAC 进行过滤，扫描结果最大保存 8 个 AP 的信息。

例子	{
	struct altm_wext_msg msg;
	unsigned short channel = 11;
	unsigned char filter_mac[8][6];
	Wifi_Recv_Info_t private_ap_info[8];
	int addr_val = (int)(&private_ap_info[0]);
	
	memset(&msg, 0, sizeof(msg));

	msg.type = ATBM_DEV_IO_SET_STA_SCAN;
	memcpy(&msg.externData[0], &channel, (sizeof(channel)));
	memcpy(&msg.externData[2], &filter_mac[0][0], 8*6);
	memcpy(&msg.externData[50], &addr_val, sizeof(int));
	
	ioctrl(..., &msg, ...);
	}

6 设置特殊频点

Type ID	ATBM_DEV_IO_SET_FREQ
输入参数	参数一 1 类型：unsigned short 2 数据：externData[0:1] 3 说明：信道号（1~13） 参数二 1 类型：int 2 数据：externData[2:5] 3 说明：特殊频率（2300 ~ 2400（5M 一个频点），2407 - 2489 ，2500 ~ 2600（5M 一个频点）） 目前我们只验证了 2380 和 2504 的频点，其它的特殊频点性能不确定。 注：信道和频点会保存到文件系统中， sys/module/atbm_wifi/parameters/wifi_freq； 如果没有设置特殊频点值，会显示 NULL： ~ # cat sys/module/atbm_wifi/parameters/wifi_freq NULL ~ # 如果有设置成功特殊频点，会显示如下信息： ~ # cat sys/module/atbm_wifi/parameters/wifi_freq ch:11, freq:2455 ~ #
返回值	无

例子	{
	struct altm_wext_msg msg;
	unsigned short channel = 11;
	int freq = 2407;
	
	memset(&msg, 0, sizeof(msg));
	msg.type = ATBM_DEV_IO_SET_FREQ;
	memcpy(&msg.externData[0], &channel, (sizeof(channel)));
	memcpy(&msg.externData[2], &freq, sizeof(int));

	ioctrl(..., &msg, ...); }

7 设置特殊字段的数据内容

Type ID	ATBM_DEV_IO_SET_SPECIAL_OUI

输入参数	1 类型：char
	2 数据：externData[0:254]
	3 说明：设置私有字段的数据内容，用于 Probe Req，Probe Resp 和 beacon 封包，数据长度 0~255 bytes；
	私有字段的 ElementId=233，OUI Type = {0x41,0x54,0x42,0x4D};//ATBM
	注：ElementId 和 OUI Type 是暂定的，可以根据用户的实际需求进行修改。
	设置的数据会被保存到文件系统
	sys/module/atbm_wifi/parameters/spec_oui

返回值	无

例子	{
	struct altm_wext_msg msg;
	
	memset(&msg, 0, sizeof(msg));
	msg.type = ATBM_DEV_IO_SET_SPECIAL_OUI;
	memcpy(&msg.externData[0], “11223344aabbcc”, 14);
	
	ioctrl(..., &msg, ...);
	}

8 Station 模式下主动断掉与 AP 之间的连接

Type ID	ATBM_DEV_IO_SET_STA_DIS
输入参数	无
返回值	无

例子	{
	struct altm_wext_msg msg;
	
	memset(&msg, 0, sizeof(msg));
	msg.type = ATBM_DEV_IO_SET_STA_DIS;
	
	ioctrl(..., &msg, ...);
	}

9 Monitor 模式的 Enable/Disable

Type ID	ATBM_DEV_IO_SET_IFTYPE
输入参数	参数一 1 类型：char 2 数据：externData[0] 3 说明：iftype 内核的枚举定义如下： enum ieee80211_internal_iftype{ IEEE80211_INTERNAL_IFTYPE_REQ__MANAGED, IEEE80211_INTERNAL_IFTYPE_REQ__MONITOR, };参数二 1 类型：char 2 数据：externData[1] 3 说明：信道号 注：使用的时候需要先把应用关掉。把 hostapd kill 掉。
返回值	无

例子	{
	struct altm_wext_msg msg;
	char ch_type = 0;
	char channel= 1;
	
	memset(&msg, 0, sizeof(msg));
	msg.type = ATBM_DEV_IO_SET_IFTYPE;
	memcpy(&msg.externData[0], &ch_type, sizeof(char));
	memcpy(&msg.externData[1], & channel, sizeof(char));
	
	ioctrl(..., &msg, ...);
	}

10 Wi-Fi 自适应（CE 认证）

Type ID	ATBM_DEV_IO_SET_ADAPTIVE
输入参数	1 类型：int 2 数据：externData[0:3] 3 说明：0:diable 1:enable
返回值	无

例子	{
	struct altm_wext_msg msg;
	int adaptive= 1;//enable

	memset(&msg, 0, sizeof(msg));
	msg.type = ATBM_DEV_IO_SET_ADAPTIVE;
	memcpy(&msg.externData[0], &adaptive, sizeof(int));
	
	ioctrl(..., &msg, ...);
	}

11 Map 导入与 Map 内容读写

Type ID	ATBM_DEV_IO_SET_TXPWR_DCXO
输入参数	参数一 1 类型：int 2 数据：externData[0:3] 3 说明：低频，tx power (-32~32) 参数二 1 类型：int 2 数据：externData[4:7] 3 说明：中频，tx power (-32~32) 参数三 1 类型：int 2 数据：externData[8:11] 3 说明：高频，tx power (-32~32) 参数四 1 类型：int 2 数据：externData[12:15] 3 说明：dcxo (0~127) 该功能设置到寄存器，不写到 efuse
返回值	无

例子	{
	struct altm_wext_msg msg;
	int tx_pwr_L= 30;
	int tx_pwr_M= 30;
	int tx_pwr_H= 30;
	Int dcxo = 20
	
	memset(&msg, 0, sizeof(msg));
	msg.type = ATBM_DEV_IO_SET_TXPWR_DCXO;
	memcpy(&msg.externData[0], &tx_pwr_L, sizeof(int));
	memcpy(&msg.externData[4], &tx_pwr_M, sizeof(int));
	memcpy(&msg.externData[8], &tx_pwr_H, sizeof(int));
	memcpy(&msg.externData[12], &dcxo, sizeof(int));

	ioctrl(..., &msg, ...); }

12 功率设置

Type ID	ATBM_DEV_IO_SET_TXPWR
输入参数	1 类型：char 2 数据：externData[0] 3 说明：tx power (-16,16)
返回值	无

例子	{
	struct altm_wext_msg msg;
	int tx_pwr =3;
	
	memset(&msg, 0, sizeof(msg));
	msg.type = ATBM_DEV_IO_SET_TXPWR;
	msg.externData[0] = tx_pwr;
	ioctrl(..., &msg, ...);
	}

13 获取当前工作的信道

Type ID	ATBM_DEV_IO_GET_WORK_CHANNEL
输入参数	无
返回值	1 类型：unsigned short 2 数据：externData[0:1] 3 说明：当前的信道号（1~13） 注：不支持 5G 信道。

例子	{
	struct altm_wext_msg msg;
	unsigned short channel = 0；
	
	memset(&msg, 0, sizeof(msg));
	msg.type = ATBM_DEV_IO_GET_WORK_CHANNEL;
	
	ioctrl(..., &msg, ...);
	
	memcpy(&channel, &msg.externData[0], sizeof(unsigned short));

	}

14 最优信道

Type ID	ATBM_DEV_IO_SET_BEST_CHANNEL_SCAN
输入参数	无

返回值	1 类型：
	typedef struct _best_ch_scan_result_
	{
	unsigned int channel_ap_num[14];//每个信道的 AP 个数
	unsigned int busy_ratio[14];//每个信道的繁忙比例
	unsigned char suggest_ch;//建议的最优信道号
	}Best_Channel_Scan_Result;
	2 数据：externData[0:N]
	3 说明：增加信道限制功能
	参数：
	msg.externData[0]起始信道
	msg.externData[1]截止信道
	
	最佳信道选择会在该范围内选择。

例子	{
	struct altm_wext_msg msg;
	Best_Channel_Scan_Result rls_buf;
	
	memset(&msg, 0, sizeof(msg));
	msg.type = ATBM_DEV_IO_SET_BEST_CHANNEL_SCAN;
	if(argv[0]){
	msg.externData[0] = atoi(argv[0]);
	if(argv[1])
	msg.externData[1] = atoi(argv[1]);
	}
	ioctrl(..., &msg, ...);
	
	memcpy(&rls_buf, &msg.externData[0], sizeof(rls_buf));
	}

15 获取 AP 列表

Type ID	ATBM_DEV_IO_GET_AP_LIST
输入参数	1 类型：char * 2 数据：externData[0:3] 3 说明：四个字节的内存地址，传给 wifi 驱动，用于保存所有扫描到的 AP 信息。
返回值	1 类型： typedef struct _scan_ap_info_ { u8 ssid[32]; u8 mac_addr[6]; u8 rssi; u8 flag; u8 enc_type; }SCAN_AP_INFO; 2 数据：AP list 列表存放在输入参数的内存地址中。 3 说明： 需要的内存 buffer 比较大，18Kbytes； 现在每个信道保存 32 个 AP 信息，每个 AP 信息的结构如上所示，所以需要的内存是 14*32*41 = 18368 bytes. 4enc_type 参数说明 0 open 1 WPA 2 WPA2 3 WPA/WPA2 4WEP

例子	{
	struct altm_wext_msg msg;
	SCAN_AP_INFO scan_ap_info_buff[14][32];
	int addr_val = (int)(&scan_ap_info_buff[0][0]);
	
	memset(&msg, 0, sizeof(msg));
	msg.type = ATBM_DEV_IO_GET_AP_LIST;
	memcpy(&msg.externData[0], &addr_val, 4);
	
	ioctrl(..., &msg, ...);
	
	}

16 获取吞吐率

Type ID	ATBM_DEV_IO_GET_TP_RATE
输入参数	1 类型：char 2 数据：externData[0:5] 3 说明： AP 模式需要获取连接的 sta 的当前环境的吞吐量，该参数对应的是 sta 的 mac 地址; STA 模式该参数可以为空。
返回值	1 类型：int 2 数据：externData[0:3] 3 说明：rate value

例子	{
	struct altm_wext_msg msg;
	char mac[6] = {0x11, 0x12, 0x13, 0x14, 0x15, 0x16};
	int rate_val = 0;
	
	memset(&msg, 0, sizeof(msg));
	msg.type =ATBM_DEV_IO_GET_TP_RATE;
	memcpy(&msg.externData[0], mac, 6);
	
	ioctrl(..., &msg, ...);
	
	memcpy(&rate_val, &msg.externData[0], sizeof(int));
	
	}

17 速率上限设置

Type ID	ATBM_DEV_IO_MAX_TX_RATE
输入参数	1 类型：int 2 数据：value，设置为 0 恢复默认 3 说明：设置发送最大速率 11b: 10,20,55,110 11g:60,90,120,180,240,360,480,540 11n:65,130,195,260,390,520,585,650
返回值	无

例子	{
	struct altm_wext_msg msg;
	
	msg.type = ATBM_DEV_IO_MAX_TX_RATE;
	msg.value = 650；

	ioctrl(..., &msg, ...); }

18 固定速率设置

Type ID	ATBM_DEV_IO_FIX_TX_RATE
输入参数	1 类型：int 2 数据：value，设置为 0 恢复默认 3 说明：固定发送速率 11b: 10,20,55,110 11g:60,90,120,180,240,360,480,540 11n:65,130,195,260,390,520,585,650
返回值	无

例子	{
	struct altm_wext_msg msg;
	
	memset(&msg, 0, sizeof(msg));
	msg.type = ATBM_DEV_IO_FIX_TX_RATE;
	msg.value = 650；
	ioctrl(..., &msg, ...);
	}

19 速率控制取消

Type ID	ATBM_DEV_IO_TX_RATE_FREE
输入参数	无
返回值	1 类型：int 2 数据：externData[0:3] 3 说明：rate value

例子	{
	struct altm_wext_msg msg;
	memset(&msg, 0, sizeof(msg));
	msg.type = ATBM_DEV_IO_TX_RATE_FREE;
	ioctrl(..., &msg, ...);
	}

20 获取 efuse

Type ID	ATBM_DEV_IO_GET_EFUSE
输入参数	无
返回值	无
例子	{

	struct altm_wext_msg msg;
	memset(&msg, 0, sizeof(msg));
	msg.type = ATBM_DEV_IO_GET_EFUSE;
	ioctrl(..., &msg, ...);
	
	}

21 更改 efuse mac 地址

Type ID	ATBM_DEV_IO_SET_EFUSE_MAC
输入参数	U8 mac[6]
返回值	无

例子	{
	struct altm_wext_msg msg;
	u8 mac[6] = {0x00,0x01,0x22,0x33,0x44,0x55};
	memset(&msg, 0, sizeof(msg));
	msg.type =ATBM_DEV_IO_SET_EFUSE_MAC;
	memcpy(&msg.externData[0], mac, 6);
	ioctrl(..., &msg, ...);
	}

22 设置 efuse dcxo

Type ID	ATBM_DEV_IO_SET_EFUSE_DCXO
输入参数	U8 dcxo
返回值	无

例子	{
	struct altm_wext_msg msg;
	u8 dcxo = 18;
	memset(&msg, 0, sizeof(msg));
	msg.type =ATBM_DEV_IO_SET_EFUSE_DCXO;
	msg.externData[0] = dcxo;
	ioctrl(..., &msg, ...);
	}

23 设置 efuse delta gain

Type ID	ATBM_DEV_IO_SET_EFUSE_DELTAGAIN
输入参数	U8 delta_gain1 U8 delta_gain2 U8 delta_gain3

返回值	无

例子	{
	struct altm_wext_msg msg;
	u8 delta_gain1= 18;
	u8 delta_gain2= 19;
	u8 delta_gain3= 20;
	
	memset(&msg, 0, sizeof(msg));
	msg.type =ATBM_DEV_IO_SET_EFUSE_DCXO;
	msg.externData[0] = delta_gain1;
	msg.externData[1] = delta_gain2;
	msg.externData[2] = delta_gain3;
	ioctrl(..., &msg, ...);
	}

24 设置 set min rate

Type ID	ATBM_DEV_IO_MIN_TX_RATE
输入参数	1 类型：int 2 数据：value ， 设置为 0 恢复默认 3 说明：固定发送速率 11b: 10,20,55,110 11g:60,90,120,180,240,360,480,540 11n:65,130,195,260,390,520,585,650
返回值	无

例子	{
	struct altm_wext_msg msg;
	
	memset(&msg, 0, sizeof(msg));
	msg.type =ATBM_DEV_IO_MIN_TX_RATE;
	msg.value = 650；
	ioctrl(..., &msg, ...);
	}

25 设置 set rate power

Type ID	ATBM_DEV_IO_SET_RATE_POWER
输入参数	Char rate_index [0:10] Char power [-16:16] rate_idx：0~10，对应的速率如下， 0→ 1/2M 1→ 5.5/11M

	2→ 6/6.5M 3→ 9M 4→ 12/13M 5→ 18/19.5M 6→ 24/26M 7→ 36/39M 8→ 48/52M 9→ 54/58.5M 10→ 65M rate_txpower: -16~16
返回值	无

例子	{
	struct altm_wext_msg msg;
	char rate_index = 1;
	char power=1;
	memset(&msg, 0, sizeof(msg));
	msg.type =ATBM_DEV_IO_SET_RATE_POWER;
	msg.externData[0] = rate_index;
	msg.externData[1] = power;
	ioctrl(..., &msg, ...);
	}

26 设置过滤功能

Type ID	ATBM_DEV_IO_SET_SPECIAL_FILTER

输入参数	msg.value:
	1 filter_frame 过滤的帧类型，只能过滤 beacon/probe req
	2 filter_ie 过滤携带 IE 的帧
	3 filter clean
	4 filter show
	msg.externData[0]:
	filter_frame : 80 or 40 ， 80 是 beacon,40 是 probe req
	filter_ie : ie ， 携带该 IE 的帧，十进制值
	msg.externData[1~3]:
	filter_ie : oui1 oui2 pui3 ，IE 里面的 OUI，十进制值
	
	enum SPECIAL_FILTER_TYPE{
	FILTER_FRAME = 1,
	FILTER_IE = 2,
	FILTER_CLEAR = 3,
	FILTER_SHOW = 4,
	};

返回值	无

例子	{
	int ret = -1;
	int status;
	struct altm_wext_msg msg;
	memset(&msg,0,sizeof(msg));
	msg.type = ATBM_DEV_IO_SET_SPECIAL_FILTER;
	if(argc > 0)
	msg.value = atoi(argv[0]);
	if(argc > 1)
	msg.externData[0] = atoi(argv[1]);
	if(argc > 2)
	msg.externData[1] = atoi(argv[2]);
	if(argc > 3)
	msg.externData[2] = atoi(argv[3]);
	if(argc > 4)
	msg.externData[3] = atoi(argv[4]);
	ret = ioctrl_wext_send_ack(nl_connect, &msg, &status);
	if(ret != 0){
	printf("error, ioctrl send\n");
	}
	if(msg.value == FILTER_SHOW)
	printf("[show] %s \n ",msg.externData);
	return ret;
	}

具体使用	过滤 beacon 帧 Atbm_Wext wlan0 filter_special 1 80 过滤 IE 为 243 的帧 Atbm_wext wlan0 filter_special 2 243 过滤 IE 为 243 ， OUI 为 04-16-21 的帧 Atbm_Wext wlan0 filter_special 2 243 4 22 33 查看设置了哪些过滤条件 Atbm_Wext wlan0 filter_special 4 清除所有过滤条件 Atbm_wext wlan0 filter_special 3

27 设置国家码

Type ID	ATBM_DEV_IO_SET_COUNTRY_CODE = 31
输入参数	目前只支持 2.4G 的部分设置： 中国：CN 信道限制：1~13 美国：US 信道限制：1~11 日本：JP 信道限制：1~14

返回值	无

例子	{
	struct altm_wext_msg msg;
	memset(&msg, 0, sizeof(msg));
	msg.type = ATBM_DEV_IO_SET_COUNTRY_CODE;
	memcpy(msg.externData,”CN”,2);
	ioctrl(..., &msg, ...);
	}

28 获取驱动版本

Type ID	ATBM_DEV_IO_GET_DRIVER_VERSION
输入参数	无
返回值	无

例子	{
	struct altm_wext_msg msg;
	memset(&msg, 0, sizeof(msg));
	msg.type = ATBM_DEV_IO_GET_DRIVER_VERSION;
	ioctrl(..., &msg, ...);
	
	}

29 定频 start_tx

Type ID	ATBM_DEV_IO_ETF_START_TX
输入参数	msg.externData[0]: 信道：1~14 msg.externData[1]~ msg.externData[2]： 发包速率： 11b:10 20 55 110 11g:60 90 120 180 240 360 480 540 11n:65 130 195 260 390 520 585 650 msg.externData[3]~ msg.externData[4]： 发包长度 msg.externData[5]： 发包带宽： 1 –>HT40 0 -->HT20 msg.externData[6]: 发包模式： 1 –> greedfiled mode 0 –> legacy mode
返回值	无

例子	{
	struct altm_wext_msg msg;
	short *p;
	memset(&msg, 0, sizeof(msg));
	msg.type = ATBM_DEV_IO_ETF_START_TX;
	msg.externData[0] =1;
	p = (short *)&msg.externData[1];
	*p = 650;
	p = (short *)&msg.externData[3];
	*p = 1000;
	msg.externData[5] = 0;
	msg.externData[6] = 0;
	ioctrl(..., &msg, ...);
	
	}

30 定频 stop_tx

Type ID	ATBM_DEV_IO_ETF_STOP_TX
输入参数	无
返回值	无

例子	{
	struct altm_wext_msg msg;
	memset(&msg, 0, sizeof(msg));
	msg.type = ATBM_DEV_IO_ETF_STOP_TX;
	ioctrl(..., &msg, ...);
	
	}

31 定频 start_rx

Type ID	ATBM_DEV_IO_ETF_START_RX
输入参数	msg.externData[0]: 信道：1~14 msg.externData[1]： 收包带宽： 1 –>HT40 0 -->HT20
返回值	无

例子	{
	struct altm_wext_msg msg;
	memset(&msg, 0, sizeof(msg));

	msg.type = ATBM_DEV_IO_ETF_START_RX;
	msg.externData[0] =1;
	msg.externData[1] = 0;
	ioctrl(..., &msg, ...);
	
	}

32 定频 stop_rx

Type ID	ATBM_DEV_IO_ETF_START_TX
输入参数	无
返回值	无

例子	{
	struct altm_wext_msg msg;
	short *p;
	memset(&msg, 0, sizeof(msg));
	msg.type = ATBM_DEV_IO_ETF_STOP_RX;
	ioctrl(..., &msg, ...);
	
	}

33 定频 start_rx_result

Type ID	ATBM_DEV_IO_GET_ETF_START_RX_RESULTS
输入参数	无 需要注意改命令需要在 start_rx 执行的前提下执行
返回值	无

例子	{
	struct altm_wext_msg msg;
	short *p;
	memset(&msg, 0, sizeof(msg));
	msg.type = ATBM_DEV_IO_GET_ETF_START_RX_RESULTS;
	ioctrl(..., &msg, ...);
	
	}

CONTACT INFORMATION

AltoBeam (China) Inc.

Address: B808, Tsinghua Tongfang Hi-Tech Plaza, Haidian, Beijing, China 100083

Tel: (8610) 6270 1811

Fax: (8610) 6270 1830

Website: www.altobeam.com

Email: support@altobeam.com

DISCLAIMER