RTL8712 Driver Programming Guide

Preliminary

Revision: Draft 0.2

Released Date:2009/06/01

Revision History

Version	Data	Author	Change
0.1	04/06/2009	George	Initial draft
0.2	06/01/2009		Adding and modified the whole document.
0.3			
0.4			
0.5			
0.6			
0.7			

Table of Contents

1 Introduction

This document is intended for RTL8712 device driver programmers who need to port to different OS. RTL8712 is a processor-based IEEE802.11 b/g/n compliant RF-SoC.
To configure RTL8712 working properly, driver must control RTL8712 by conventional register read-write and the command/event mechanism. The embedded CPU will process the command file passed from the Host Driver, and it will also notify the proper events to the Host Driver as the appropriate conditions satisfy. Driver programmer should refer RTL8712 data sheet for detailed register set information.

2 Driver Architecture

RTL8712 Driver Architecture

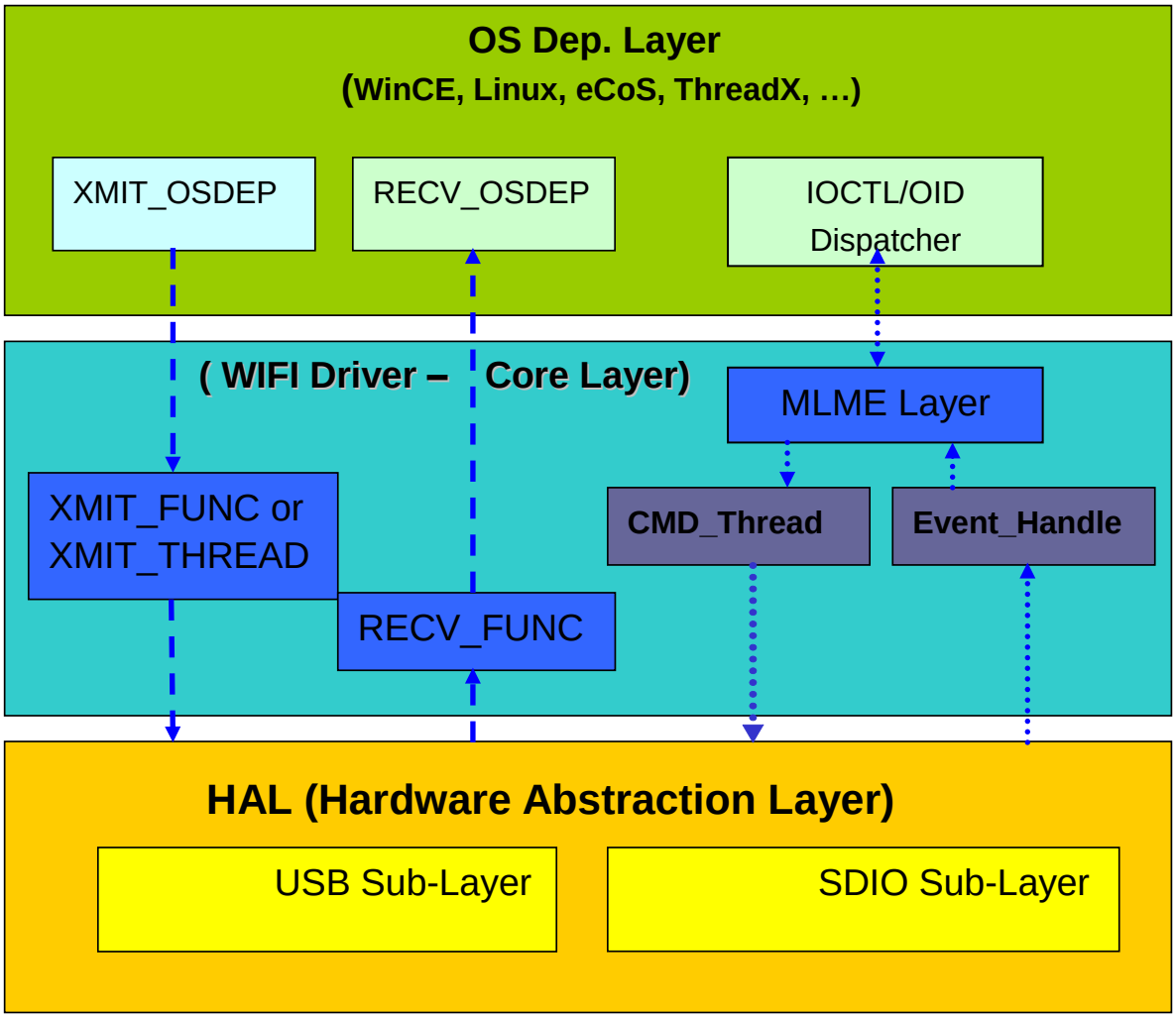

Figure 2-1

Overview of function blocks

(A) XMIT_OSDEP, XMIT_FUNC and XMIT_THREAD

XMIT_DESP include the functions that related with the OS interface.

XMIT_FUNC and XMIT_THREAD include the functions that are OS independent. The core packet transmission flow is in XMIT_FUNC.

(B) RECV_OSDEP and RECV_FUNC

RECV_OSDEP include the functions that related with the OS interface.

RECV_FUNC include the functions that are os independent. The core packet receiving flow is in RECV_FUNC.

(C) HCI Wrapper

The all io APIs are in this layer. The APIs is offered for uplayer, which didn't need to know the HCI type is USB or SDIO.

(D) OID/OICTL Dispatcher

This Dispatcher is for OS to access or control the WLAN card.

(E) CMD Thread

The CMD Thread is the method which the WLAN driver is used to communicates with WLAN firmware.

(F) Event Handle

The Event Handle is the method which the WLAN firmware is used to communicates with WLAN driver.

(G) MLME Layer

The MLME layer handles the mlme part of wlan protocol.

3 Modules and Function Blocks

3.1 XMIT Path

3.1.1 TX Data Structure

```
struct xmit_priv     {
        _lock       lock;
        _sema       xmit_sema;
        _sema       terminate_xmitthread_sema;

        _queue      be_pending;
        _queue      bk_pending;
        _queue      vi_pending;
        _queue      vo_pending;
        _queue      bm_pending;
        _queue      legacy_dz_queue;
        _queue      apsd_queue;

        u8 *pallocated_frame_buf;
        u8 *pxmit_frame_buf;

        uint free_xmitframe_cnt;
        uint mapping_addr;
        uint pkt_sz;

        _queue      free_xmit_queue;
        struct      hw_txqueue    be_txqueue;
        struct      hw_txqueue    bk_txqueue;
        struct      hw_txqueue    vi_txqueue;
        struct      hw_txqueue    vo_txqueue;
        struct      hw_txqueue    bmc_txqueue;
        uint  frag_len;

        _adapter  *adapter;
```

```
    u8      vcs_setting;
    u8    vcs;
    u8    vcs_type;
    u16    rts_thresh;
    uint  tx_bytes;
    uint  tx_pkts;
    uint  tx_drop;
    struct hw_xmit *hwxmits;
    u8    hwxmit_entry;

#ifdef CONFIG_USB_HCI
    _sema     tx_retevt;//all tx return event;
    u8          txirp_cnt;//

    //per AC pending irp
    int beq_cnt;
    int bkq_cnt;
    int viq_cnt;
    int voq_cnt;

#endif

#ifdef CONFIG_RTL8712
    _queue    free_amsdu_xmit_queue;
    u8 *pallocated_amsdu_frame_buf;
    u8 *pxmit_amsdu_frame_buf;
    uint free_amsdu_xmitframe_cnt;

    _queue free_txagg_xmit_queue;
    u8 *pallocated_txagg_frame_buf;
    u8 *pxmit_txagg_frame_buf;
    uint free_txagg_xmitframe_cnt;

    int cmdseq;
#endif
#ifdef CONFIG_SDIO_HCI
    u8 free_pg[8];
```

```
	u8	public_pgsz;
	u8	required_pgsz;
	u8	used_pgsz;
	u8	init_pgsz;
#ifdef PLATFORM_WINDOWS
	PMDL prd_freesz_mdl[2];
	u8 brd_freesz_pending[2];
	PIRP	prd_freesz_irp[2];
	PSDBUS_REQUEST_PACKET prd_freesz_sdrp[2];
	u8 rd_freesz_irp_idx;
#endif

#endif

	_queue free_xmitbuf_queue;
	_queue pending_xmitbuf_queue;
	u8 *pallocated_xmitbuf;
	u8 *pxmitbuf;
	uint free_xmitbuf_cnt;
};

struct	sta_xmit_priv {
	_lock	lock;
	sint	option;
	sint	apsd_setting;	//When bit mask is on, the associated edca queue supports APSD.

	struct tx_servq	be_q;			//priority == 0,3
	struct tx_servq	bk_q;			//priority == 1,2
	struct tx_servq	vi_q;			//priority == 4,5
	struct tx_servq	vo_q;			//priority == 6,7
	_list	legacy_dz;
	_list	apsd;

	uint	sta_tx_bytes;
	uint	sta_tx_pkts;
	uint	sta_tx_fail;
};
```

```
struct	xmit_frame	{
	_list list;
	struct	pkt_attrib attrib;
	_pkt *pkt;
	int frame_tag;
	_adapter *padapter;
	u8 *buf_addr;
	struct xmit_buf *pxmitbuf;

#ifdef CONFIG_SDIO_HCI
	u8 pg_num;
#endif

#ifdef CONFIG_USB_HCI
	//insert urb, irp, and irpcnt info below...
	//max frag_cnt = 8

	u8 *mem_addr;
	u32 sz[8];
	PURB	pxmit_urb[8];
#ifdef PLATFORM_WINDOWS
	PIRP	pxmit_irp[8];
#endif
	u8 bpending[8];
	sint ac_tag[8];
	sint last[8];
	uint irpcnt;
	uint fragcnt;
#endif

	uint  mem[1];
};
```

```
struct pkt_attrib	{
	u8	type;
	u8	subtype;
```

```
    u16  ether_type;
    int    pktlen;
    int    pkt_hdrlen;
    int    hdrlen;
    int    nr_frags;
    int    last_txcmdsz;
    int    encrypt;   //when 0 indicate no encrypt. when non-zero, indicate the encrypt algorith
    unsigned char iv[8];
    int    iv_len;
    unsigned char icv[8];
    int    icv_len;
    int    priority;
    int    ack_policy;
    int    mac_id;
    int    vcs_mode;

    u8    dst[ETH_ALEN];
    u8    src[ETH_ALEN];
    u8    ta[ETH_ALEN];
    u8    ra[ETH_ALEN];
};
```

3.1.2 TX Flow

When Windows/Linux wants to send packets, Windows/Linux will transfer the packet by calling the function handler (SendHandler/hard_start_xmit), which is registered by driver in DriverEntry/init_netdev. The SendHandler/ hard_start_xmit in RTL8712 is xmit_entry.

It has the main job is translating the ethernet network packet to wireless network packet and send the packet to air.

The function xmit_entry has two methods to send the packet:

1. XMIT_THREAD_MODE (SDIO):

The function xmit_entry update the information of packets then enqueues the packets to software queue. The other function xmit_thread dequeues the packets from the software queue then writes to hardware fifo and inform the FW to handle the packet.

Below are the main steps which are done by xmit_entry function:

I. Check the link status: if the device is not linked, drop the packets

II. Get a free xmitframe: check if enough xmitframe for linking Ethernet packet.

III. Function update_attrib: Driver will gather from Ethernet header and security setting and keep the information in struct pkt_attrib.

IV. Function xmit_enqueue in function pre_xmit: Driver enqueues the xmitframe to according software queue.

V. Inform the xmit_thread to handle the packet: up the specified semaphore to inform xmit_thread to handle the packets.

Below are the main steps which are done by xmit_thread function:

I. Waiting the semaphore: waiting the information from xmit_entry.

II. Function update_free_ffsz: update current hw fifo size for later use.

III. Function xmit_xmitframes:

a. First get a free xmitbuf, check if there is any free buffer.

b. checking if there is enough hw resource to transfer the packets.

c. Function xmitframe_coalesce decide the packet size and packet header and restrict the packet header ,payload and update txdesc , then copy to xmitbuf. Driver copy several xmitframe to xmitbuf until xmitbuf does not have enough space.(Burst TX for SDIO).

d. Writing the xmitbuf to corresponding mac tx fifo.

2. XMIT_DIRECT_MODE(USB):

The function xmit_entry first update the information of packets ,then the function pre_xmit write packets directly to hardware fifo or equeues the packets to the software queue if there is no xmitbuf or there is already packets in software queue. The callback function xmitframe_complete will write packets from software queue to hardware fifo if there is enough xmitbuf.

Below are the main steps which are done by xmit_entry function:

3 Check the link status: if the device is not linked, drop the packets

4 Get a free xmitframe: check if enough xmitframe for linking Ethernet packet.

5 Function update_attrib: Driver will gather from Ethernet header and security setting and keep the information in struct pkt_attrib.

Below are the main steps which are done by pre_xmit function:

I. If any pending packets in software queue or there is not enough xmitbuf. Driver enqueue packet to software queue.

II. Call the function xmit_direct: xmitframe_coalesce decide the packet size and packet header and restrict the packet header and payload. Driver copy xmitframe to xmitbuf. The function dump_xframe updates txdesc and writes the xmitbuf to corresponding mac tx fifo.

Below are the main steps which are done by xmitframe_complete function:

I. Dequeue from software queue.

II. Call the function xmitframe_coalesce and dump_xframe.

3.2 RECV Path

3.2.1 RX Data Structure

```
struct recv_priv {
	_lock	lock;
	_sema	recv_sema;
	_sema	terminate_recvthread_sema;
	_queue	free_recv_queue;
	_queue	recv_pending_queue;
	u8 *pallocated_frame_buf;
	u8 *precv_frame_buf;
	uint free_recvframe_cnt;
	_adapter  *adapter;

	u8 *pallocated_recv_buf;
	u8 *precv_buf;	// 4 alignment
	_queue	free_recv_buf_queue;
	u8	free_recv_buf_queue_cnt;

	//for A-MPDU Rx reordering buffer control
	struct recv_reorder_ctrl recvreorder_ctrl[16];

	uint  rx_bytes;
	uint  rx_pkts;
	uint  rx_drop;

	uint	rx_icv_err;
	uint	rx_largepacket_crcerr;
	uint	rx_smallpacket_crcerr;
	uint	rx_middlepacket_crcerr;

	// OS dependent and interface dependent fields
	// …
};
```

```
union recv_frame{
	union{
		_list list;
		struct recv_frame_hdr hdr;
		uint mem[RECVFRAME_HDR_ALIGN>>2];
	}u;
};
```

```
struct rx_pkt_attrib {
	u8	qos;
	u8	to_fr_ds;
	u8	frag_num;
	u16	seq_num;
	u8	pw_save;
	u8	mfrag;
	u8	mdata;
	u8	privicy;	// in frame_ctrl field
	int	hdrlen;	// the WLAN Header Len
	int	encrypt;
	int	iv_len;
	int	icv_len;
	int	priority;
	int	ack_policy;
	u8	dst[ETH_ALEN];
	u8	src[ETH_ALEN];
	u8	ta[ETH_ALEN];
	u8	ra[ETH_ALEN];
	u8	bssid[ETH_ALEN];
};
```

3.2.2 RX Flow

After the frame is received by hardware, driver will be notified. The notification mechanism depends on what kinds of interface that we use.

For SDIO, RXDONE interrupt will tell driver for incoming frame. Then RXDONE handler will process this event. For USB, during driver initialization, it will issue IRPs to tell USB bus driver for

sending IN TOKEN. Then bus driver will make polling to check whether there is an incoming frame. If there is an incoming frame, the corresponding callback function will be called. For all interfaces, *recv_entry* will call *recv_func* to process this frame. Below are the main steps which are done by this function:

1. *validate_recv_frame*
   - Update frame attribute and check data type (control/management/data)
   - Check to/from DS
     sta->ap, ap->sta, sta->sta
   - Decache
     sequence number, fragment number, retry bit
2. *decryptor*
   - Decrypt and set the *ivlen,icvlen* of the *recv_frame*
3. *recvframe_chk_defrag*
   - Check whether de-fragmentation is necessary
   - If it is necessary, enqueue the frame to *defrag_q*
4. *portctrl*
   - Set the security information in the *recv_frame*
   - When using WPA & WPA2, after association and before 4-way handshake, we only accept EAPOL packet
5. *process_recv_indicatepkts*
   - Reorder the 11n packets.
6. *wlanhdr_to_ethhdr*
   - Remove the 802.11 header and add the 802.3 header.
   - Add priority to IP header (TOS field) and OOB
7. *recv_indicatepkt*
   - Indicate 802.3 packet to OS.
   - After processing the packet, OS will return the packet to driver.

3.3 HCI Wrapper Layer

This chapter illustrates APIs that 8712 use to access hardware, including registers, sram and fifo (ports). Some of the supported APIs now are synchronous io, that is the bus driver will be waiting until the io is done. That also means all these APIs now are with no callback function. The others are asynchronous io, this kind of APIs have callback function.

This table shows all the APIs in this chapter, including done and to do in the feature.

Synchronous IO	**Asynchronous IO**
direct	***direct***
read8	
read16	
read32	
write8	
write16	
write32	
read_mem	
write_mem	
read_port(sdio)	read_port(usb)
write_port(sdio)	write_port(usb)

This function is used to read a character value from hardware registers.

```
u8 read8(
  _adapter* padapter,
  u32 address
);
```

- Parameters
  - **padapter**
    [IN] pointer to the_adapter* structure
  - **address**

[IN] Specify the address of register

- Return Values

  Return the character (u8) value of the register.

- Remarks

= =

This function is used to read a word value from hardware registers.

```
u16 read16(
  _adapter* padapter,
  u32 address
);
```

- Parameters
  - **padapter**

    [IN] pointer to the_adapter* structure
  - **address**

    [IN] Specify the address of register

- Return Values

  Return the word (u16) value of the register.

= =

This function is used to read a double-word value from hardware registers.

```
u32 read32(
  _adapter* padapter,
  u32 address
);
```

- Parameters

- **padapter**
  [IN] pointer to the_adapter* structure
- **address**
  [IN] Specify the address of register

- Return Values
  Return the double-word (u32) value of the register.

= =

This function is used to write a character value to hardware registers.

```
void write8(
  _adapter* padapter,
  u32 address,
   u8  val
);
```

- Parameters
  - **padapter**
    [IN] pointer to the_adapter* structure
  - **address**
    [IN] Specify the address of register
  - **val**
    [IN] Specify the character value you want to write to the register

- Return Values
  void

= =

This function is used to write a word value to hardware registers.

```
void write16(
  _adapter* padapter,
  u32 address,
```

```
  u16  val
);
```

- Parameters
  - **padapter**
    [IN] pointer to the_adapter* structure
  - **address**
    [IN] Specify the address of register
  - **val**
    [IN] Specify the word value (u16) you want to write to the register

- Return Values
  Void

= =

This function is used to write a double-word value to hardware registers.

```
void write32(
 _adapter* padapter,
 u32 address,
  u32  val
);
```

- Parameters
  - **padapter**
    [IN] pointer to the_adapter* structure
  - **address**
    [IN] Specify the address of register
  - **val**
    [IN] Specify the double-word (u32) value you want to write to the register

- Return Values
  void

= =

This function is used to read data from 8712 hardware sram.

```
void read_mem(
 _adapter* padapter,
 u32 address,
  u32  cnt,
  u8* pmem
);
```

- Parameters
  - **padapter**
    [IN] pointer to the_adapter* structure
  - **address**
    [IN] Specify the address of register
  - **cnt**
    [IN] Specify the total length you want to access the SRAM
  - **pmem**
    [IN/OUT] Pointer to store the read back data

- Return Values
  void

= =

This function is used to write data to 8712 hardware sram.

```
void write_mem(
 _adapter* padapter,
 u32 address,
  u32  cnt,
  u8* pmem
);
```

- Parameters
  - **padapter**
    [IN] pointer to the_adapter* structure
  - **address**
    [IN] Specify the address of register
  - **cnt**
    [IN] Specify the total length you want to access the SRAM
  - **pmem**
    [IN/OUT] Pointer to the write data

- Return Values
  void

==

This function is used to read data from 8711 hardware rx fifo.

```
void read_port(
 _adapter* padapter,
 u32 address,
  u32  cnt,
  u8* pmem
);
```

- Parameters
  - **padapter**
    [IN] pointer to the_adapter* structure
  - **address**
    [IN] Specify the address of rx FIFO. For rx path, this value is “Host_R_FIFO”
  - **cnt**
    [IN] Specify the total length you want to access the rx FIFO
  - **pmem**
    [IN/OUT] Pointer to store the read back data
- Return Values
  void

= =

This function is used to write data to 8711 hardware tx fifo.

```
void write_port(
 _adapter* padapter,
 u32 address,
  u32  cnt,
  u8* pmem
);
```

- Parameters
  - **padapter**
    [IN] pointer to the_adapter* structure
  - **address**
    [IN] Specify the address of rx FIFO. For tx path, this value is "Host_W_FIFO"
  - **cnt**
    [IN] Specify the total length you want to access the tx FIFO
  - **pmem**
    [IN] Pointer to the data to write to tx FIFO

- Return Values
  Void

= =

3.3.1 SDIO sub-Layer

The sdio interface is using command52/command53 to access hardware. The register can use command52 or command53. The tx fifo and rx fifo are using command53. The address used in the SDIO interface is 17 bits. So this layer will translate the 32 bits address to 17 bits address. The translation rule is defined in datasheet.

3.3.2 USB sub-Layer

The RTL8712 uses the control pipe to access registers and uses bulk pipes to access Tx/Rx fifo. There are five endpoints in the RTL8712 hardware for USB interface. One control endpoint, two Tx Bulk out endpoints, one Rx Bulk in endpoint, one H2C Bulk out endpoint. The following table is the description for each endpoint of RTL8712 hardware.

	Endpoint number	Purpose
Control Endpoint	0x00	Used to access the MAC/RF/BaseBand registers
Bulk Out	0x04	Used to transmit the VO/VI data packets
Bulk Out	0x06	Used to transmit the BE/BK data packets
Bulk In	0x03	Used to receive the C2H event and the data packets from air
Bulk Out	0x0d	Used to transmit the H2C commands to RTL8712 firmware

The Control Endpoint is used by USB core to get the RTL8712 hardware information and starts the USB initialization procedure. The RTL8712 wireless driver is able to use the Control Endpoint to read/write the MAC/RF/BaseBand register.

The Bulk out endpoint with endpoint number 0x04 is used to transmit the data packets with VO/VI QoS tag. This endpoint is a high priority endpoint.

The Bulk out endpoint with endpoint number 0x06 is used to transmit the data packets with BE/BK QoS tag. This endpoint is a low priority endpoint.

The Bulk IN endpoint with endpoint number 0x03 is used to receive the data packets which the RTL8712 device got from the air. This endpoint is also used to receive the C2H event which is generated by RTL8712 firmware. For more detail about the C2H event, please check the section 3.5.2.

The Bulk out endpoint with endpoint number 0x0d is used to send the H2C command to RTL8712 firmware. Some tasks will be offloaded by RTL8712 firmware. For more details about H2C command, please check the section 3.5.1 and 3.5.3.

3.4 IOCTL Dispatcher

3.4.1 IOCTL overview

In a conventional operating system, the IOCTL is an interface between user-space and kernel space. The application on the user-space typically makes a request to the device driver on the kernel-space via the IOCTL.

In the rtl8712 driver, we define a number of dispatch functions for making responses to the IOCTL.

These APIs(or dispatching functions) which are defined in the “rtl871x_ioctl_linux.c” file support the standard wireless tools, e.g. iwconfig, iwlist, and wpa_supplicant.

The iwlist and iwconfig are for scanning (site survey) and connecting to AP respectively.

The wpa_supplicant is for key management & key negotiation with a WPA Authenticator.

3.4.2 Basic dispatch functions

When the iwlist tool makes a scanning request, in the rtl8712 driver the corresponding dispatch function translates the request to the core function which handles with the scanning request and collects the APs’(BSS) information scanned in the vicinity .
Similarly, when the iwconfig tool makes a connecting request, in the rtl8712 driver the corresponding dispatch function translates the request to the core function which processes the connecting procedure.

For more detail, please refer to the website
http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Tools.html/.

Fro Linux OS, the rtl8712 driver needs to register the dispatch (callback) functions with kernel as follows:

```
static iw_handler r8711_handlers[] =
{
        NULL,                           /* SIOCSIWCOMMIT */
```

```
r8711_wx_get_name,              /* SIOCGIWNAME */
dummy,                          /* SIOCSIWNWID */
dummy,                          /* SIOCGIWNWID */
r8711_wx_set_freq,          /* SIOCSIWFREQ */
r8711_wx_get_freq,              /* SIOCGIWFREQ */
r8711_wx_set_mode,               /* SIOCSIWMODE */
r8711_wx_get_mode,               /* SIOCGIWMODE */
dummy,//r8711_wx_set_sens,          /* SIOCSIWSENS */
r8711_wx_get_sens,                /* SIOCGIWSENS */
NULL,                                   /* SIOCSIWRANGE */
r8711_wx_get_range,         /* SIOCGIWRANGE */
NULL,                                   /* SIOCSIWPRIV */
NULL,                                   /* SIOCGIWPRIV */
NULL,                                   /* SIOCSIWSTATS */
NULL,                                    /* SIOCGIWSTATS */
dummy,                                   /* SIOCSIWSPY */
dummy,                                   /* SIOCGIWSPY */
NULL,                                    /* SIOCGIWTHRSPY */
NULL,                                   /* SIOCWIWTHRSPY */
r8711_wx_set_wap,               /* SIOCSIWAP */
r8711_wx_get_wap,                 /* SIOCGIWAP */
r871x_wx_set_mlme,                /* request MLME operation; uses struct iw_mlme */
dummy,                           /* SIOCGIWAPLIST -- depricated */
r8711_wx_set_scan,              /* SIOCSIWSCAN */
r8711_wx_get_scan,              /* SIOCGIWSCAN */
r8711_wx_set_essid,         /* SIOCSIWESSID */
r8711_wx_get_essid,         /* SIOCGIWESSID */
dummy,                                  /* SIOCSIWNICKN */
dummy,                                  /* SIOCGIWNICKN */
NULL,                                   /* -- hole -- */
NULL,                                    /* -- hole -- */
r8711_wx_set_rate,          /* SIOCSIWRATE */
r8711_wx_get_rate,               /* SIOCGIWRATE */
dummy,                                  /* SIOCSIWRTS */
r8711_wx_get_rts,                      /* SIOCGIWRTS */
r8711_wx_set_frag,              /* SIOCSIWFRAG */
r8711_wx_get_frag,               /* SIOCGIWFRAG */
```

```
        dummy,                          /* SIOCSIWTXPOW */
        dummy,                          /* SIOCGIWTXPOW */
        dummy,//r8711_wx_set_retry,     /* SIOCSIWRETRY */
        r8711_wx_get_retry,//           /* SIOCGIWRETRY */
        r8711_wx_set_enc,               /* SIOCSIWENCODE */
        r8711_wx_get_enc,               /* SIOCGIWENCODE */
        dummy,                          /* SIOCSIWPOWER */
        r8711_wx_get_power,             /* SIOCGIWPOWER */
        NULL,                           /*---hole---*/
        NULL,                           /*---hole---*/
        r871x_wx_set_gen_ie,            * SIOCSIWGENIE */
        NULL,                           /* SIOCGWGENIE */
        r871x_wx_set_auth,              /* SIOCSIWAUTH */
        NULL,                           /* SIOCGIWAUTH */
        r871x_wx_set_enc_ext,           /* SIOCSIWENCODEEXT */
        NULL,                           /* SIOCGIWENCODEEXT */
        r871x_wx_set_pmkid,             /* SIOCSIWPMKSA */
        NULL,                           /*---hole---*/
};
```

Basic dispatch functions are described as below:

Command code	Registered function	Description
SIOCSIWSCAN	r8711_wx_set_scan()	for scanning the APs in the vicinity.
SIOCGIWSCAN	r8711_wx_get_scan()	get scanning results for further connecting.
SIOCSIWESSID	r8711_wx_set_essid()	connecting to the AP via the given AP's SSID.
SIOCSIWAP	r8711_wx_set_wap()	connecting to the AP via the given AP's BSSID
SIOCGIWRANGE	r8711_wx_get_range()	Get range of parameters.
SIOCSIWENCODE	r8711_wx_set_enc()	for wep key setting and setting open or shared system mode of the wep.

3.4.3 Advanced dispatch functions for WPA/WPA2

For key management & key negotiation with a WPA Authenticator, we adopt the Linux WPA/WPA2/IEEE 802.1X Supplicant. The driver APIs of the rtl8712 driver are compliant with “driver_wext.c”

For more detail, please refer to the website http://hostap.epitest.fi/wpa_supplicant/.

The advanced dispatch functions for wpa_supplicant are described as below:

Command code	Registered function	Description
SIOCSIWGENIE	r871x_wx_set_gen_ie()	set Generic IEEE 802.11 information element (e.g., for WPA/RSN/WMM).
SIOCGIWGENIE	n/a	get Generic IEEE 802.11 information, element (e.g., for WPA/RSN/WMM).
SIOCSIWMLME	r871x_wx_set_mlme()	request MLME operation; uses * struct iw_mlme.
SIOCSIWAUTH	r871x_wx_set_auth()	Authentication mode parameters. set authentication mode parameters.
SIOCGIWAUTH	n/a	Authentication mode parameters. get authentication mode parameters.
SIOCSIWENCODEEXT	r871x_wx_set_enc_ext()	Extended version of encoding configuration set encoding token & mode
SIOCGIWENCODEEXT	n/a	Extended version of encoding configuration get encoding token & mode
SIOCSIWPMKSA	r871x_wx_set_pmkid()	For PMKSA cache management.

3.5 CMD/EVENT

The CMD/EVENT mechanism is a communication method between RTL8712 wireless driver and RTL8712 firmware. Of course, the RTL8712 will perform the wireless service on the Host machine. We will use the term “Host driver” to present the RTL8712 wireless driver in the following session. Because of the CMD/EVENT mechanism, the host can offload some tasks by issuing a command (we call this command as H2C command) to the RTL8712 firmware. After the RTL8712 firmware receive the H2C command with the specific command code, the RTL8712 firmware will know what it should perform now. After the RTL8712 firmware finish the H2C command sent from the host driver, it will report some status or results to the host driver by issuing the event (we call this event as C2H Event) with specific event code. For example, the 802.11 MLME is implemented by this method.

The following figure shows the H2C command format:

<table>
<tr><td>BIT</td><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>Offset 0</td><td>OWN</td><td colspan="7">RSVD</td><td colspan="8">Offset=0</td><td colspan="16">TXPKTSIZE</td></tr>
<tr><td>Offset 4</td><td colspan="23">RSVD</td><td colspan="5">QSEL(5b)</td><td colspan="8">RSVD</td></tr>
<tr><td>Offset 8</td><td colspan="32">RSVD</td></tr>
<tr><td>Offset 12</td><td colspan="16">RSVD</td><td colspan="8">TAILPAGE(8b)</td><td colspan="8">NEXTHEADPAGE(8b)</td></tr>
<tr><td>Offset 16</td><td colspan="32">RSVD</td></tr>
<tr><td>Offset 20</td><td colspan="16">RSVD</td><td colspan="16">RSVD</td></tr>
<tr><td>Offset 24</td><td colspan="32">RSVD</td></tr>
<tr><td>Offset 28</td><td colspan="32">RSVD</td></tr>
<tr><td>Offset 32</td><td>C</td><td colspan="7">CMD_SEQ</td><td colspan="8">Element ID</td><td colspan="16">CMD_LEN</td></tr>
<tr><td>Offset 36</td><td colspan="32">RSVD</td></tr>
<tr><td>Offset 40</td><td colspan="32" rowspan="3">Content</td></tr>
<tr><td>~</td></tr>
<tr><td>Offset N</td></tr>
<tr><td>Offset N+4</td><td>C</td><td colspan="7">CMD_SEQ</td><td colspan="8">Element ID</td><td colspan="16">CMD_LEN</td></tr>
<tr><td>Offset N+8</td><td colspan="32">RSVD</td></tr>
</table>

Figure 4-1

The RTL8712 wireless driver (host driver) will send the H2C command to the RTL8712 firmware via the specific interface on the host platform. For the RTL8712 USB wireless device, the H2C command will be sent by using the Bulk OUT endpoint and this endpoint is just for the H2C command. For the RTL8712 SDIO wireless device, the H2C command will be sent by using the

CMD52/CMD53 and the SDIO host controller will transfer the H2C command to RTL8712 SDIO device. For more details about the H2C command format, please refer to the RTL8712 data sheet.

3.5.1 H2C Commands

As the previous description, if there are some tasks needs to be handled by RTL8712 firmware, the host driver will issue the H2C command to the RTL8712 firmware. There are lots of H2C commands the RTL8712 wireless driver supports. It means the RTL8712 firmware will offload lots of tasks for the host driver and it will reduce the host CPU utilization.

All supported H2C commands stand in the Rtl8712_cmd.h file and this file can be found in the include folder of RTL8712 driver source tree. In the following section, this document will have some description for each supported H2C command.

1.1.1.1. _setBCNITV

In the SoftAP mode or Adhoc mode, the RTL8712 wireless device has to send the beacon frame. The host driver can issue this command to modify the beacon interval value.

1.1.1.2. _JoinBss

When host driver issues this command to firmware, firmware will perform link up flow given the parameter of the command.

1.1.1.3. _DisConnect

Firmware will disconnect from the current associated BSS/IBSS when this command is received.

1.1.1.4. _CreateBss

Firmware will create an IBSS/BSS when this command is received.

1.1.1.5. _SetOpMode

Host driver can configure Firmware being as Infrastructure, Ad HoC or AP mode by issuing this command.

1.1.1.6. _SiteSurvey

Firmware will perform the site survey flow when this command is received.

1.1.1.7. _SetAuth

Host driver can configure the authentication algorithm by issuing this command. The authentication

algorithm can be open, shared, WPA, and WPA2.

1.1.1.8. _SetKey

Host driver can ask Firmware to set up default Key by issuing this command.

1.1.1.9. _SetStaKey

Host driver can ask Firmware to set up Key-Mapping Key by issuing this command.

1.1.1.10._SetAssocSta

Host driver can ask Firmware to allow the association of a STA by issuing this command. This command will be used when RTL8712 device is in AdHoc or AP mode.

1.1.1.11._DelAssocsta

Host Driver can ask Firmware to disassociate a STA by this command. This command will be used when in AdHoC or AP mode.

1.1.1.12._SetBasicRate

The basic rate is the transmission rate to send the management and control frame. Host driver can use this command to set the basic rate.

1.1.1.13._SetDataRate

The data rate is the transmission rate to send the data frame. Host driver can use this command to set the list of data rates. The RTL8712 firmware will use an appropriate rate as the data rate from the list of data rates.

1.1.1.14._SetAtim

Host driver can issue this command to set the ATIM window in the Adhoc mode. In the power saving environment of Adhoc network, each station must wake up after the TBTT time for a while to listen the ATIM packet. This packet will indicate the station had buffered some packets for you or not.

1.1.1.15._SetPwrMode

This command is used to set the power saving mode of RTL8712 wireless device.

1.1.1.16._SetUsbSuspend

This command can be used to set the power action of RTL8712 device. If the action is 0, the host driver will try to wake the RTL8712 up. If the action is 1, the host driver will make the RTL8712 sleep.

1.1.1.17._AddBAReq

If the A-MPDU feature is enabled, the RTL8712 has to send the AddBA action frame to the wireless AP. After receiving the AddBA response sent from AP, the RTL8712 will just know the detail parameters of AP's capability of A-MPDU. Host driver can issue this command to ask the RTL8712 firmware to send the AddBA request frame to AP.

3.5.2 Firmware Events

The following figure illustrates the procedure for the host driver to receive notified event from firmware.

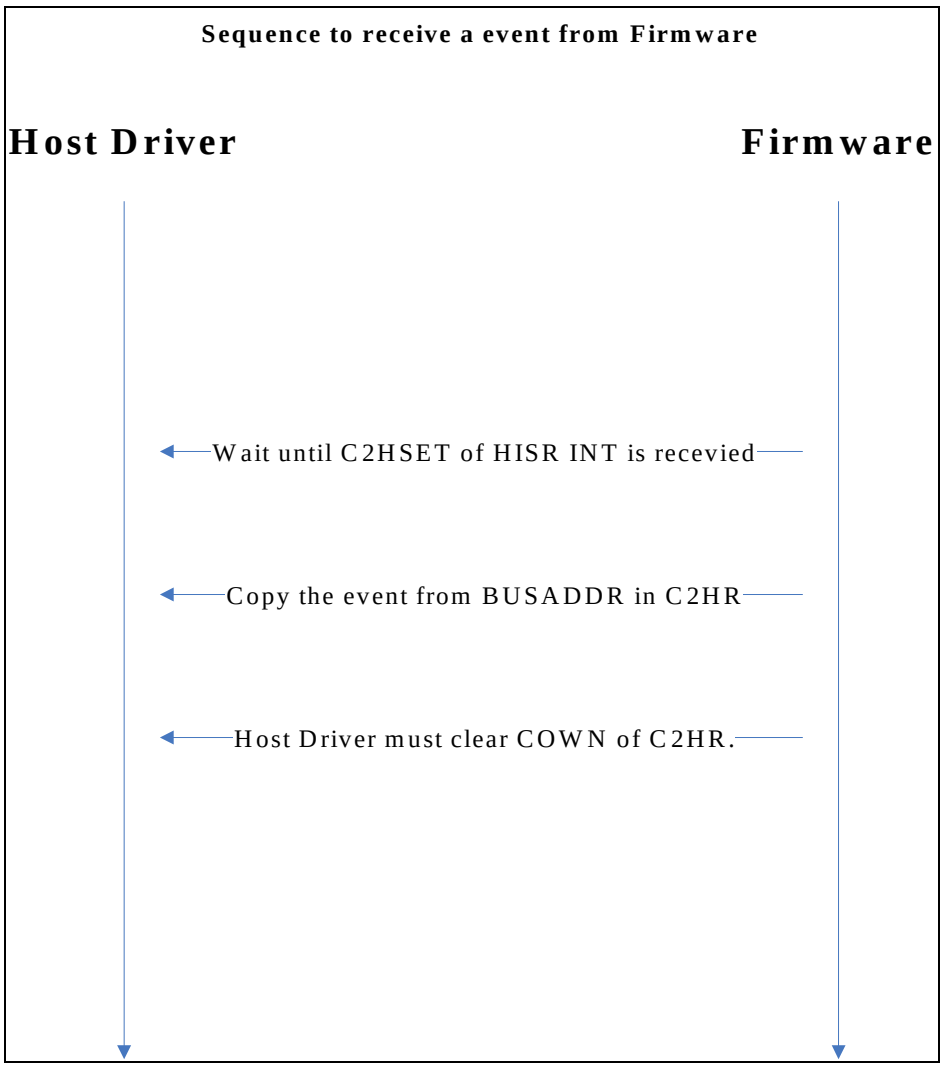

Figure 4-3

The supported firmware events are specified in the following sub-sections:

1.1.1.18.Survey_EVT

When a BSS is sensed, Firmware will report the information of the BSS to Host Driver by this event.

Event Structure		
Field name	Type	Description
bss	NDIS_WLAN_BSSID_EX	Report a bss has been scanned

1.1.1.19.SurverDone_EVT

Firmware will notify the end of site survey by this event.

Event Structure		
Field name	Type	Description
bss_cnt	unsigned int	Report that the requested site survey has been done. bss_cnt : ***indicates the number of bss that has been reported.***

1.1.1.20.JoinBss_EVT

Firmware will notify the result of the previous join BSS/IBSS request by this command.

Event Structure		
Field name	Type	Description
network	wlan_network	Report the link result of joinning the given bss. join_res: -1: ***authentication fail*** -2: ***association fail*** > 0: ***TID***

The structure of wlan_network :

```
struct    wlan_network {
            _list      list;
            int         network_type;
            int         fixed;
```

```
        unsigned long   last_scanned;
        int     aid;
        int     join_res;
        NDIS_WLAN_BSSID_EX      network;
};

typedef struct  list_head       _list;

struct list_head {
        struct list_head *next, *prev;
};
```

1.1.1.21. AddSTA_EVT

Firmware will notify a new STA has already joined the BSS/IBSS by this command. This event will be used when in AdHoC or AP mode.

Event Structure		
Field name	Type	Description
macaddr[6]	unsigned char	The MAC address of the station added by firmware.
rsvd[2]	unsigned char	Reserved

1.1.1.22. DelSTA_EVT

Firmware will notify an associated STA has already quit the BSS/IBSS by this command. This event will be used when in AdHoC or AP mode.

Event Structure		
Field name	Type	Description
macaddr[6]	unsigned char	The MAC address of the station deleted by firmware.
rsvd[2]	unsigned char	Reserved

3.5.3 CMD Threads

CMD thread is responsible for handling the command which is fired by host driver. When the driver needs the RTL8712 firmware to accomplish a task, the host driver will queue a required command with specific command code then signal the CMD thread to de-queue the command. CMD thread sends the command to RTL8712 firmware via the APIs which is provided by Host Driver Controller (ex: USH Host Controller, SDIO Host Controller). The related data structure is as follows:

```
struct      cmd_priv {
        _sema     cmd_queue_sema;
        _sema     cmd_done_sema;
        _sema     terminate_cmdthread_sema;
        _queue    cmd_queue;
        u8    cmd_seq;
        u8    *cmd_buf;      //shall be non-paged, and 4 bytes aligned
        u8    *cmd_allocated_buf;
        u8    *rsp_buf; //shall be non-paged, and 4 bytes aligned

        u8    *rsp_allocated_buf;
        u32  cmd_issued_cnt;
        u32  cmd_done_cnt;
        u32  rsp_cnt;
        _adapter *padapter;
};
```

```
struct cmd_obj {
        u16  cmdcode;
        u8    res;
        u8    *parmbuf;
        u32  cmdsz;
        u8    *rsp;
        u32  rspsz;
        //_sema    cmd_sem;
        _list list;
};
```

3.5.4 Event Handling

rxcmd_event_hdl

When the firmware indicates an event, the driver will receive an interrupt of C2HSET, the driver in the ISR calls **rxcmd_event_hdl** to handle this event. The event length and sequence can be retrieved from the header of event. After the information is obtained, **event_handle** is responsible for the event handling.

Event_handle:

First of all, several procedures should be done to ensure the event is valid. Events with invalid sequence number, size or event code should be dropped and driver will continue to obtain next one. If there is an corresponding event callback to the event, it will be executed. Event callback structure is illustrated as follows:

```
 struct fwevent wlanevents[] =
{     {0, dummy_event_callback},     /*0*/
      {0, NULL},
      {0, NULL},
      {0, NULL},
      {0, NULL},
      {0, NULL},
      {0, NULL},
      {0, NULL},
      {0, NULL},
      {0, &survey_event_callback},          /*8*/
      {sizeof (struct surveydone_event), &surveydone_event_callback},      /*9*/
      {0, &joinbss_event_callback},         /*10*/
      {sizeof(struct stassoc_event), &stassoc_event_callback},
      {sizeof(struct stadel_event), &stadel_event_callback},
      {0, &atimdone_event_callback},
      {0, dummy_event_callback},
      {0, NULL},      /*15*/
      {0, NULL},
      {0, NULL},
      {0, NULL},
      {0, fwdbg_event_callback},
      {0, NULL},        /*20*/
      {0, NULL},
      {0, NULL},
      {0, &cpwm_event_callback},
```

```
};
```

3.6 MLME Layer

The 802.11 MLME which performs the 802.11 MAC Layer Management tasks is carried out based on the cmd/event mechanism between RTL8712 host driver and firmware. For instance, the functions of site survey, joining a BSS, disassociating a BSS, and so on. Depending on different OS platforms, these utilities for wireless configuration can direct the driver to interact with the firmware based on cmd/event mechanism in order to achieve the 802.11 MLME tasks. These utilities use the ioctl method to control the driver, for example, the NDIS driver on the windows XP cooperate with window-zero-config utility, and Linux WIFI driver can use the wireless tools as the iwconfig/iwlist, etc.

The clause 3.6.1 ~ 3.6.3 illustrate the RTL8712's 802.11 MLME flow among utility, driver, and firmware based on the Windows XP OS. And the clause 3.6.4 illustrates the same flow based on LINUX OS using the iwconfig/iwlist wireless tools.

3.6.1 Site Survey - BSSID SCAN and LIST

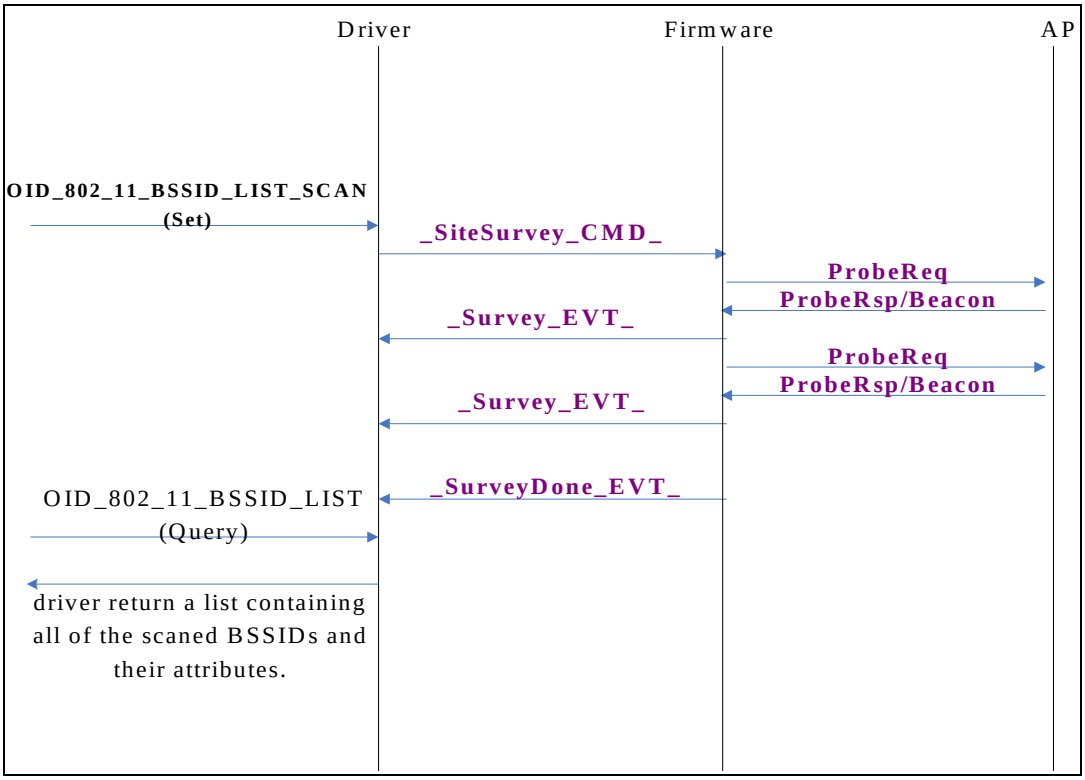

Figure 4-4

3.6.2 Join a BSS - Set SSID

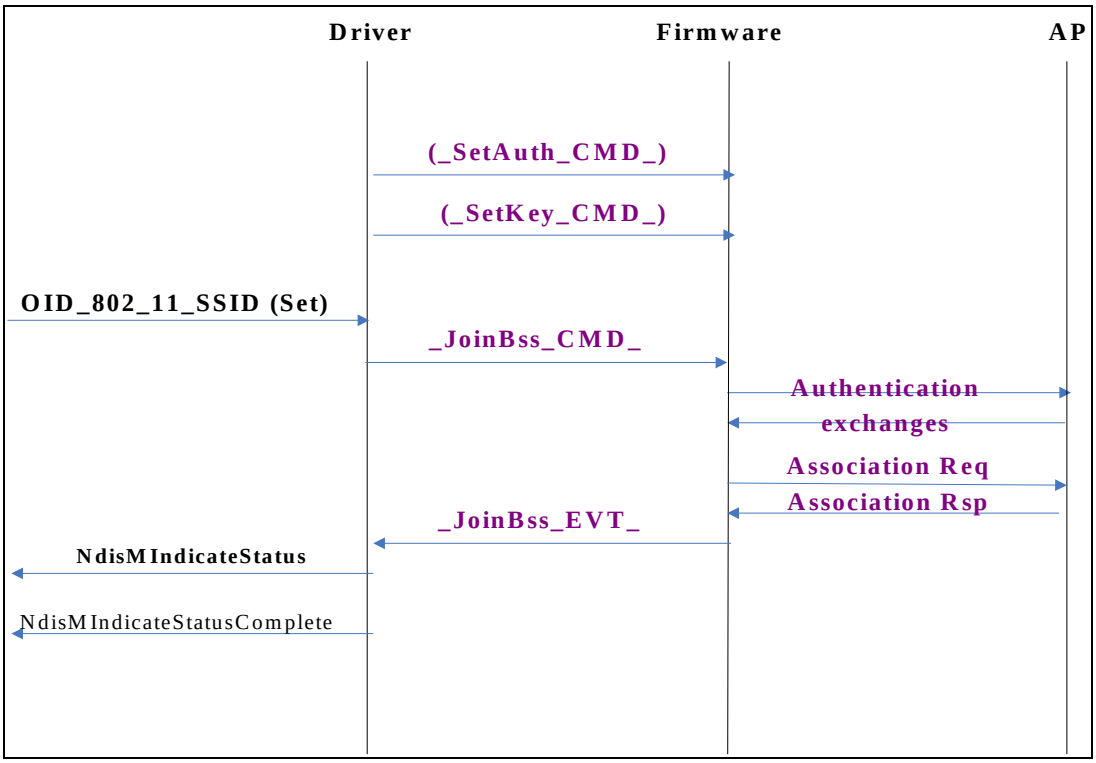

Figure 4-5

3.6.3 Disconnect - Disassociate CMD

3.6.4 Wireless LAN IOCTL on Linux OS

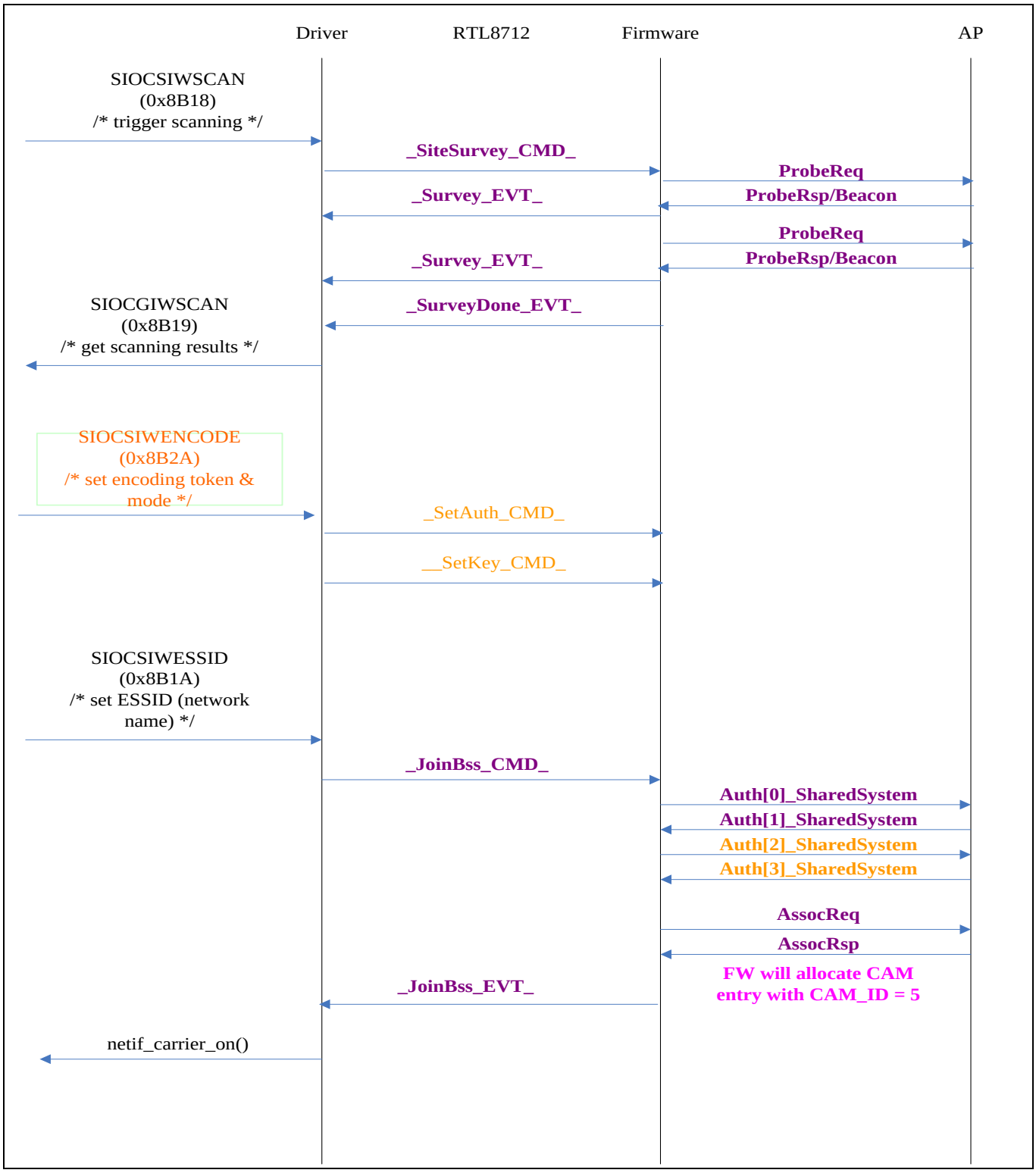

Figure 4-6

4 INT Handler

This INT Handler is used in SDIO interface. When the SDIO bus driver find there is a interrupt triggered by SDIO card, the bus driver will call the corresponding INT Handler. The INT Handler will read the SDIO_HIMR register to check there is a truly interrupt or fake interrupt. The INT Handler will do the respond handling according to the INT BIT.

5 Basic Operations

6 References