Data structure :

Here is a brief description of some associated structures which plays an important role in the analysis process of b43 source code.Some data structures, which I have set in **bold** style, are the most important data structures to analyze b43 source code.

```
/* Data structure for the WLAN parts (802.11 cores) of the b43 chip. */
struct b43_wl {

        /* Pointer to the active wireless device on this chip */
        struct b43_wldev *current_dev;

struct ieee80211_hw is the most important data structure, it represents hardware information and state, driver calls function ieee80211_alloc_hw() to allocate it.
        /* Pointer to the ieee80211 hardware data structure */
        struct ieee80211_hw *hw;

        /* Global driver mutex. Every operation must run with this mutex locked. */
        struct mutex mutex;
        /* Hard-IRQ spinlock. This lock protects things used in the hard-IRQ
         * handler, only. This basically is just the IRQ mask register. */
        spinlock_t hardirq_lock;

        /* Set this if we call ieee80211_register_hw() and check if we call
         * ieee80211_unregister_hw(). */
        bool hw_registred;

        /* We can only have one operating interface (802.11 core)
         * at a time. General information about this interface follows.
         */

        struct ieee80211_vif *vif;
        /* The MAC address of the operating interface. */
        u8 mac_addr[ETH_ALEN];
        /* Current BSSID */
        u8 bssid[ETH_ALEN];

        /* Interface type. (NL80211_IFTYPE_XXX) */
        int if_type;
        /* Is the card operating in AP, STA or IBSS mode? */
        bool operating;
        /* filter flags */
        unsigned int filter_flags;
        /* Stats about the wireless interface */
        struct ieee80211_low_level_stats ieee_stats;
```

```
#ifdef CONFIG_B43_HWRNG
	struct hwrng rng;
	bool rng_initialized;
	char rng_name[30 + 1];
#endif /* CONFIG_B43_HWRNG */

	/* List of all wireless devices on this chip */
	struct list_head devlist;
	u8 nr_devs;

	bool radiotap_enabled;
	bool radio_enabled;

	/* The beacon we are currently using (AP or IBSS mode). */
	struct sk_buff *current_beacon;
	bool beacon0_uploaded;
	bool beacon1_uploaded;
	bool beacon_templates_virgin; /* Never wrote the templates? */
	struct work_struct beacon_update_trigger;

	/* The current QOS parameters for the 4 queues. */
	struct b43_qos_params qos_params[B43_QOS_QUEUE_NUM];

	/* Work for adjustment of the transmission power.
	 * This is scheduled when we determine that the actual TX output
	 * power doesn't match what we want. */
	struct work_struct txpower_adjust_work;

	/* Packet transmit work */
	struct work_struct tx_work;

	/* Queue of packets to be transmitted. */
	struct sk_buff_head tx_queue[B43_QOS_QUEUE_NUM];

	/* Flag that implement the queues stopping. */
	bool tx_queue_stopped[B43_QOS_QUEUE_NUM];

	/* firmware loading work */
	struct work_struct firmware_load;

	/* The device LEDs. */
	struct b43_leds leds;

	/* Kmalloc'ed scratch space for PIO TX/RX. Protected by wl->mutex. */
	u8 pio_scratchspace[118] __attribute__((__aligned__(8)));
```

```
        u8 pio_tailspace[4] __attribute__((__aligned__(8)));
};
```

Hardware Descriptor: TX/RX

RX header for v4 firmware

```
struct b43_rxhdr_fw4 {
        __le16 frame_len;       /* Frame length */
         PAD_BYTES(2);
        __le16 phy_status0;    /* PHY RX Status 0 */
        union {
                /* RSSI for A/B/G-PHYs */
                struct {
                        __u8 jssi;        /* PHY RX Status 1: JSSI */
                        __u8 sig_qual;/* PHY RX Status 1: Signal Quality */
                } __packed;

                /* RSSI for N-PHYs */
                struct {
                        __s8 power0;  /* PHY RX Status 1: Power 0 */
                        __s8 power1;  /* PHY RX Status 1: Power 1 */
                } __packed;
        } __packed;
        union {
                /* HT-PHY */
                struct {
                        PAD_BYTES(1);
                        __s8 phy_ht_power0;
                } __packed;

                /* RSSI for N-PHYs */
                struct {
                        __s8 power2;
                        PAD_BYTES(1);
                } __packed;

                __le16 phy_status2;    /* PHY RX Status 2 */
        } __packed;
        union {
                /* HT-PHY */
                struct {
                        __s8 phy_ht_power1;
                        __s8 phy_ht_power2;
                } __packed;
```

```
                __le16 phy_status3;    /* PHY RX Status 3 */
        } __packed;
        union {
                /* Tested with 598.314, 644.1001 and 666.2 */
                struct {
                        __le16 phy_status4;    /* PHY RX Status 4 */
                        __le16 phy_status5;    /* PHY RX Status 5 */
                        __le32 mac_status;     /* MAC RX status */
                        __le16 mac_time;
                        __le16 channel;
                } format_598 __packed;

                /* Tested with 351.126, 410.2160, 478.104 and 508.* */
                struct {
                        __le32 mac_status;     /* MAC RX status */
                        __le16 mac_time;
                        __le16 channel;
                } format_351 __packed;
        } __packed;
} __packed;
```

TX header for v4 firmware

```
struct b43_txhdr {
        __le32 mac_ctl;                 /* MAC TX control */
        __le16 mac_frame_ctl;           /* Copy of the FrameControl field */
        __le16 tx_fes_time_norm;        /* TX FES Time Normal */
        __le16 phy_ctl;                 /* PHY TX control */
        __le16 phy_ctl1;                /* PHY TX control word 1 */
        __le16 phy_ctl1_fb;             /* PHY TX control word 1 for fallback rates */
        __le16 phy_ctl1_rts;            /* PHY TX control word 1 RTS */
        __le16 phy_ctl1_rts_fb;         /* PHY TX control word 1 RTS for fallback rates */
        __u8 phy_rate;                  /* PHY rate */
        __u8 phy_rate_rts;              /* PHY rate for RTS/CTS */
        __u8 extra_ft;                  /* Extra Frame Types */
        __u8 chan_radio_code;           /* Channel Radio Code */
        __u8 iv[16];                    /* Encryption IV */
        __u8 tx_receiver[6];            /* TX Frame Receiver address */
        __le16 tx_fes_time_fb;          /* TX FES Time Fallback */
        struct b43_plcp_hdr6 rts_plcp_fb; /* RTS fallback PLCP header */
        __le16 rts_dur_fb;              /* RTS fallback duration */
        struct b43_plcp_hdr6 plcp_fb;   /* Fallback PLCP header */
        __le16 dur_fb;                  /* Fallback duration */
        __le16 mimo_modelen;            /* MIMO mode length */
        __le16 mimo_ratelen_fb;         /* MIMO fallback rate length */
        __le32 timeout;                 /* Timeout */
```

```
        union {
                /* Tested with 598.314, 644.1001 and 666.2 */
                struct {
                        __le16 mimo_antenna;            /* MIMO antenna select */
                        __le16 preload_size;            /* Preload size */
                        PAD_BYTES(2);
                        __le16 cookie;                  /* TX frame cookie */
                        __le16 tx_status;               /* TX status */
                        __le16 max_n_mpdus;
                        __le16 max_a_bytes_mrt;
                        __le16 max_a_bytes_fbr;
                        __le16 min_m_bytes;
                        struct b43_plcp_hdr6 rts_plcp;  /* RTS PLCP header */
                        __u8 rts_frame[16];             /* The RTS frame (if used) */
                        PAD_BYTES(2);
                        struct b43_plcp_hdr6 plcp;      /* Main PLCP header */
                } format_598 __packed;

                /* Tested with 410.2160, 478.104 and 508.* */
                struct {
                        __le16 mimo_antenna;                    /* MIMO antenna select */
                        __le16 preload_size;            /* Preload size */
                        PAD_BYTES(2);
                        __le16 cookie;                  /* TX frame cookie */
                        __le16 tx_status;               /* TX status */
                        struct b43_plcp_hdr6 rts_plcp;          /* RTS PLCP header */
                        __u8 rts_frame[16];             /* The RTS frame (if used) */
                        PAD_BYTES(2);
                        struct b43_plcp_hdr6 plcp;      /* Main PLCP header */
                } format_410 __packed;

                /* Tested with 351.126 */
                struct {
                        PAD_BYTES(2);
                        __le16 cookie;                  /* TX frame cookie */
                        __le16 tx_status;               /* TX status */
                        struct b43_plcp_hdr6 rts_plcp;          /* RTS PLCP header */
                        __u8 rts_frame[16];             /* The RTS frame (if used) */
                        PAD_BYTES(2);
                        struct b43_plcp_hdr6 plcp;      /* Main PLCP header */
                } format_351 __packed;

        } __packed;
} __packed;
```

Data structures for DMA transmission, per 80211 core.

```
struct b43_dma {
        struct b43_dmaring *tx_ring_AC_BK; /* Background */
        struct b43_dmaring *tx_ring_AC_BE; /* Best Effort */
        struct b43_dmaring *tx_ring_AC_VI; /* Video */
        struct b43_dmaring *tx_ring_AC_VO; /* Voice */
        struct b43_dmaring *tx_ring_mcast; /* Multicast */

        struct b43_dmaring *rx_ring;

        u32 translation; /* Routing bits */
        bool translation_in_low; /* Should translation bit go into low addr? */
        bool parity; /* Check for parity */
};
```

DMA DESCRIPTOR: use by DMA Engine

Since there are multiple types of data structure for each chipset family above, there is a struct b43_dmadesc which is a combination of all type of descriptors above for easily coding.

```
/* 32-bit DMA descriptor. */
struct b43_dmadesc32 {
        __le32 control;
        __le32 address;
} __packed;

/* 64-bit DMA descriptor. */
struct b43_dmadesc64 {
        __le32 control0;
        __le32 control1;
        __le32 address_low;
        __le32 address_high;
} __packed;

struct b43_dmadesc_generic {
        union {
                struct b43_dmadesc32 dma32;
                struct b43_dmadesc64 dma64;
        } __packed;
} __packed;

struct b43_dmadesc_meta {
```

```
        /* The kernel DMA-able buffer. */
        struct sk_buff *skb;
        /* DMA base bus-address of the descriptor buffer. */
        dma_addr_t dmaaddr;
        /* ieee80211 TX status. Only used once per 802.11 frag. */
        bool is_last_fragment;
};
```

TX Report Descriptor: contains information to report upper layer

```
/* Transmit Status */
struct b43_txstatus {
        u16 cookie;             /* The cookie from the txhdr */
        u16 seq;                /* Sequence number */
        u8 phy_stat;            /* PHY TX status */
        u8 frame_count;                 /* Frame transmit count */
        u8 rts_count;           /* RTS transmit count */
        u8 supp_reason;                 /* Suppression reason */
        /* flags */
        u8 pm_indicated;        /* PM mode indicated to AP */
        u8 intermediate;        /* Intermediate status notification (not final) */
        u8 for_ampdu;           /* Status is for an AMPDU (afterburner) */
        u8 acked;               /* Wireless ACK received */
};
```

Ring with DMA engine / QUEUES with PIO engine /

struct b43_dmaring /- Transmit ring state. One of these exists for each hardware transmit ring/queue. Packets sent to us from above are assigned to ring based on their QOS priority.

```
struct b43_dmaring {
        /* Lowlevel DMA ops. */
        const struct b43_dma_ops *ops;
        /* Kernel virtual base address of the ring memory. */
        void *descbase;
        /* Meta data about all descriptors. */
        struct b43_dmadesc_meta *meta;
        /* Cache of TX headers for each TX frame.
         * This is to avoid an allocation on each TX.
         * This is NULL for an RX ring.
         */
        u8 *txhdr_cache;
```

```
        /* (Unadjusted) DMA base bus-address of the ring memory. */
        dma_addr_t dmabase;
        /* Number of descriptor slots in the ring. */
        int nr_slots;
        /* Number of used descriptor slots. */
        int used_slots;
        /* Currently used slot in the ring. */
        int current_slot;
        /* Frameoffset in octets. */
        u32 frameoffset;
        /* Descriptor buffer size. */
        u16 rx_buffersize;
        /* The MMIO base register of the DMA controller. */
        u16 mmio_base;
        /* DMA controller index number (0-5). */
        int index;
        /* Boolean. Is this a TX ring? */
        bool tx;
        /* The type of DMA engine used. */
        enum b43_dmatype type;
        /* Boolean. Is this ring stopped at ieee80211 level? */
        bool stopped;
        /* The QOS priority assigned to this ring. Only used for TX rings.
         * This is the mac80211 "queue" value. */
        u8 queue_prio;
        struct b43_wldev *dev;
#ifdef CONFIG_B43_DEBUG
        /* Maximum number of used slots. */
        int max_used_slots;
        /* Last time we injected a ring overflow. */
        unsigned long last_injected_overflow;
        /* Statistics: Number of successfully transmitted packets */
        u64 nr_succeed_tx_packets;
        /* Statistics: Number of failed TX packets */
        u64 nr_failed_tx_packets;
        /* Statistics: Total number of TX plus all retries. */
        u64 nr_total_packet_tries;
#endif /* CONFIG_B43_DEBUG */
};
```

INTERRUPT VALUE

These Interrupt Value is triggered by function b43_do_interrupt_thread() in (main.c) .

RX Interrupt value

B43_DMAIRQ_RDESC_UFLOW : RX descriptor underrun
B43_DMAIRQ_RX_DONE : Frame successfully received

TX interrupt value

B43_IRQ_TX_OK : Frame transmission success
B43_IRQ_MAC_TXERR : MAC transmission error
B43_IRQ_PHY_TXERR : PHY transmission error