Dokumentation AT86RF230.c

Inhaltsverzeichnis

1 Wichtige Definitionen 3

2 Aufgerufene Funktionen 4

2.1 Die probe-Funktion 4

2.2 Statuswechsel 8

3 Wichtige Strukturen 11

4 IEEE802154_OPS Funktionen 16

4.1 AT86RF230_START 16

4.2 AT86RF230_SET_PROMISCUOUS_MODE 16

4.3 AT86RF230_SET_HW_ADDR_FILT 16

4.4 AT86RF230_SET_CSMA_PARAMS 16

4.5 AT86RF230_XMIT 17

4.6 AT86RF230_ISR 19

4.7 AT86RF230_ED 21

5 Anhang 22

5.1 Beschreibung des Socket Buffers 23

5.2 Ausgaben bei einem Sendevorgang 24

1 Wichtige Definitionen

Durch das Einbinden des Modules *(sudo insmod ./at86rf230.ko)* wird die Funktion

*at86rf230_probe(struct spi_device *spi)*

aufgerufen. Hier wird als erstes ein Struct vom Typen *ieee802154* definiert. Dort werden alle Daten über eine ieee802154 Hardware gespeichert.

```
69 /**
70  * struct ieee802154_hw - ieee802154 hardware
71  *
72  * @extra_tx_headroom: headroom to reserve in each transmit skb for use by the
73  *      driver (e.g. for transmit headers.)
74  *
75  * @flags: hardware flags, see &enum ieee802154_hw_flags
76  *
77  * @parent: parent device of the hardware.
78  *
79  * @priv: pointer to private area that was allocated for driver use along with
80  *      this structure.
81  *
82  * @phy: This points to the &struct wpan_phy allocated for this 802.15.4 PHY.
83  */
```

Der nächste Schritt ist die Erzeugung eines Structs *lp* vom Typen *at86rf230_local*. Dieser soll am Ende der Initialisierung alle Informationen und Daten über den Treiber und die Hardware enthalten.

2 Aufgerufene Funktionen

2.1 Die probe-Funktion

Im weiteren Verlauf der probe-Funktion werden eine Menge an Initialisierungen und Einstellungen bezogen auf die Hardware und die Kommunikation über SPI vorgenommen. Den Start macht die Funktion

at86rf230_get_pdata(spi, &rstn, &slp_tr, &xtal_trim).

Hier werden die Informationen über die PINs RST und SLP_TR beschafft. Diese beiden PINs werden benötigt, um den Status des Transceivers zu kontrollieren[1]. Darüber hinaus wird für die Kontrolle auf das Register TRX_STATE (0x02) zurückgegriffen. Ein erfolgreicher Statuswechsel wird über das Register TRX_STATUS (0x01) bestätigt.

Die Funktion

*ieee802154_alloc_hw(sizeof(*lp), &at86rf230_ops)*

stellt den benötigten Speicher für das Hardware Device zur Verfügung, indem sie einmal für jedes neue Device aufgerufen wird. Als Rückgabewert erhält man einen Pointer, mit welchem man im weiteren Verlauf auf das Hardware Device zugreifen muss. Der Struct *at86rf230_ops* ist vom Typen *ieee802154_ops* und beinhaltet alle wichtigen Methoden zum Einstellen der Hardware bezogenen Funktionen. Eine Erklärung dieser Funktionen findet weiter unten in Kapitel 4 statt. Nachdem der Aufruf von *ieee802154_alloc_hw()* erfolgt ist, können die Informationen über den Treiber und das Hardware Device in dem Pointer *lp* abgespeichert werden. Dies betrifft ebenfalls die Einstellungen über die SPI Kommunikation.

```
lp = hw->priv;
lp->hw = hw;
lp->spi = spi;
lp->slp_tr = slp_tr;
hw->parent = &spi->dev;
hw->vif data size = sizeof(*lp);
ieee802154_random_extended_addr(&hw->phy->perm_extended_addr);
```

Außerdem muss nun noch die Initialisierung der Registration Map folgen. Dies geschieht mit dem Aufruf von

devm_regmap_init_spi(spi, &at86rf230_regmap_spi_config).

Hier beschreibt *spi* das Device mit welchem die Kommunikation stattfinden soll. In *at86rf230_regmap_spi_config* sind die nötigen Register und Funktion fürs Schreiben und Lesen enthalten. Speziell für den AT86RF23x sind die Befehle in Abbildung 1 dargestellt.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Mode
1	0	Register address [5:0]						Register Access Mode – Read Access
1	1	Register address [5:0]						Register Access Mode – Write Access
0	0	1	Reserved					Frame Buffer Access Mode – Read Access
0	1	1	Reserved					Frame Buffer Access Mode – Write Access
0	0	0	Reserved					SRAM Access Mode – Read Access
0	1	0	Reserved					SRAM Access Mode – Write Access

Abbildung 1: Zugriff auf das Lesen und Schreiben

Zur Initialisierung der SPI Messages wird die Funktion

[1] Quelle: AVR, Low Power 2.4 GHz Transceiver for ZigBee, AT86RF230

*at86rf230_setup_spi_messages(struct at86rf230_local *lp)*

aufgerufen. Hier werden drei SPI Messages initialisiert:

- lp->state
- lp->irq
- lp->tx

Alle drei Structs sind vom Typ *at86rf230_state_change* und geben einen Wechsel der Operation Moden an. Diese Moden sind für IEEE802.15.4 Anwendungen ausgelegt und sind nach Abbildung 3 aufgebaut (dort erfolgt auch eine genauere Beschreibung). Für die Initialisierung der SPI Message und dem Hinzufügen der Nachricht an die Übertragungsliste werden die Funktionen

*spi_message_init(struct spi_message *m)*

*spi_message_add_tail(struct spi_transfer *t, struct spi_message *m)*

aus dem SPI Stack aufgerufen. Die erste Funktion initialisiert eine leere Transferliste mit Hilfe der endsprechenden SPI Message. Im Anschluss wird die SPI Message mit der zweiten Funktion der Transferliste angehängt.

Damit der angeschlossene Chip detektiert werden kann und damit die auf den Chip bezogenen Einstellungen vorgenommen werden können, findet die Funktion

*at86rf230_detect_device(struct at86rf230_local *lp)*

ihre Anwendung. Um die Bezeichnung des Chips auszulesen wird die Funktion

*__at86rf230_read(struct at86rf230_local *lp, unsigned int addr, unsigned int *data)*

eingesetzt. Hier wird die Manufacture ID aus den Registern MAN_ID_0 und MAN_ID_1 (0x1E und 0x1F) ausgelesen. Darüber hinaus wird noch das Register PART_NUM und VERSION_NUM (0x1C und 0x1D) ausgelesen, um im Anschluss über eine switch-case Anweisung den verwendeten Chip auszuwählen. Hier werden auch die Chip spezifischen Daten in *lp* abgelegt.

Damit ein Abschließen einer Übertragung gekennzeichnet werden kann, wird eine dynamisch allokierte Completion Struktur angelegt. Dies geschieht mit

init_completion(&lp->state_complete).

State_complete beschreibt dabei den Pointer auf die Completion Struktur.

Der nächste Schritt beinhaltet das setzten der Driver Daten in dem SPI Device. Dafür wird ebenfalls auf eine Funktion

spi_set_drvdata(spi, lp)

aus dem SPI Stack zurückgegriffen.

Nun folgt der Aufruf der Funktion

*at86rf230_hw_init(struct at86rf230_local *lp, u8 xtal_trim).*

Dort wird zuerst der Operation State mittels

at86rf230_sync_state_change(lp, STATE_FORCE_TRX_OFF)

so geändert, dass alle Übertragungen unterbrochen bzw. ausgeschaltet werden. Dies geschieht über einer Statuswechsel, welcher mit einem Aufruf der Funktion

at86rf230_async_state_change(lp, &lp->state, state, at86rf230_sync_state_change_complete, false)

hervorgerufen wird. Hier werden als Parameter die Daten des Treibers *lp* sowie die Informationen über den aktuellen Status in *&lp->state* übergeben. Darüber hinaus werden der Funktion mittels *state* der neue Status in den gewechselt werden soll, sowie die Completion-Funktion und ein boolescher Wert für das Aktivieren des Interrupt Request (IRQ), übermittelt. Die zuletzt genannten Daten werden in *&lp->state* (in dieser Funktion als *struct at86rf230_state_change *ctx* bezeichnet) abgelegt.

```
ctx->to_state = state;                //set the new state (ctx is lp->state)
ctx->complete = complete;
ctx->irq_enable = irq_enable;
at86rf230_async_read_reg(lp,RG_TRX_STATUS,ctx,at86rf230_async_state_change_start,irq_enable);
```

Der Ablauf eines Statuswechsels ist weiter unten in Kapitel 2.2 erklärt.

Nach dem Aufruf von *at86rf230_async_state_change()* folgt noch das Ausführen der Funktion

wait_for_completion_timeout().

Mit Hilfe dieser Methode wird sichergestellt, dass das Ausführen des Statuswechsels nicht unterbrochen wird. Es wird auf eine Vollendung des Transfers gewartet. Bei einer erfolgreichen Übertragung wird der Wert 0 zurückgegeben.

```
131  * This waits for either a completion of a specific task to be signaled or for a
132  * specified timeout to expire. The timeout is in jiffies. It is not
133  * interruptible.
```

Der Ablauf des Initialisierungsprozesses befindet sich momentan noch in der Abarbeitung der Funktion *at86rf230_hw_init()*. Hier wird als nächster Schritt mittels

irq_get_triger_type(lp->spi->irq)

ermittelt, auf welchen Typen der IRQ getriggert wird.

Im Anschluss werden mit der Funktion

at86rf230_write_subreg()

verschiedene Register beschrieben. Die Ausgabe in **Fehler! Verweisquelle konnte nicht gefunden werden.** zeigt, in welche Register geschrieben wird und welche Daten dort abgelegt werden. Außerdem folgt danach eine detailliertere Beschreibung der unterschiedlichen Funktionen.

```
at86rf230_write_subreg - address=4, mask=1, shift=0, data=0. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:343
at86rf230_write_subreg - address=12, mask=128, shift=7, data=1. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:343
at86rf230_write_subreg - address=14, mask=255, shift=0, data=8. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:343
at86rf230_write_subreg - address=4, mask=2, shift=1, data=0. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:343
at86rf230_write_subreg - address=45, mask=255, shift=0, data=243. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:343
at86rf230_write_subreg - address=46, mask=7, shift=0, data=212. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:343
at86rf230_write_subreg - address=3, mask=8, shift=3, data=0. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:343
at86rf230_write_subreg - address=3, mask=7, shift=0, data=0. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:343
at86rf230_write_subreg - address=18, mask=15, shift=0, data=15. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:343
```

Abbildung 2: Ausgabe der beschriebenen Register

1. Im Einzelnen bedeutet dies, dass mit dem ersten Aufruf die PIN Polarität des IRQ eingestellt wird. Standartmäßig löst eine steigende Flanke einen Interrupt aus. Hier wird dies geändert und der Interrupt ist auf einen fallende Flanke sensitiv.
2. Darauf folgt die Aktivierung der Dynamic Frame Buffer Protection. Diese Einstellung sorgt dafür, dass bei einem Empfangen von Daten, welche in dem Frame Buffer abgelegt werden, diese nicht von nachfolgenden Daten überschrieben werden. Die Einstellung verhindert dieses zeitliche Problem.
3. Mittels der dritten Einstellung wird der Antenna Diversity Application Algorithm aktiviert. Dadurch wird die Zuverlässigkeit erhöht und eine Multipath Propagation reduziert.
4. Die nächste Einstellung befasst sich mit dem *IRQ_MASK_MODE* und wird gesetzt, wenn ein Interrupt-Event auch aus dem Statusregister gelesen werden kann, trotz eines maskierten Registers. Hier wird diese Einstellung allerdings nicht aktiviert.

5. Hier wird dem Register *CSMA_SEED_0* ein Teil einer Zufallszahl für den CSMA-CA Algorithmus übergeben.
6. Hier erfolgt die Übergabe des Restes der Zufallszahl an das Register *CSMA_SEED_1*.
7. In dem Register TRX_CTRL_0 (0x03) wird das Clock Rate Update Scheme angepasst. Eine Änderung der Clock Rate wirkt sich direkt aus.
8. In dem gleichen Register wie unter Punkt 7, wird hier die Clock deaktiviert. (Außerdem bietet dieses Register auch die Einstellung der Taktfrequenz der Clock.)

Im Anschluss wird die Spannung DVDD, welche eine 1,8 V Ausgangsspannung der internen Spannungsregulatoren ist, geprüft. Dafür findet die Funktion

at86rf230_read_subreg(lp, SR_DVDD_OK, &dvdd)

Anwendung. Diese ruft wiederum die Funktion

__at86rf230_read()

auf, welche dann auf die regmap-Implementierung zurückgreift und dort mittels

regmap_read(lp->regmap, addr, data)

das entsprechende Register ausliest. Damit ist der Prozess von *at86rf230_hw_init()* abgeschlossen.

In der probe-Funktion wird nun das Register IRQ_STATUS (0x0F) mittels *at86rf230_read_subreg()* ausgelesen. Dies wird gemacht, damit alle Interrupt Bits zurückgesetzt werden. Diesem Aufruf folgt die Ermittlung des Triggers des IRQ und der Speicherung dieses Wertes in *irq_type*.

Mit dieser Information wird über

devm_request_irq(&spi->dev, spi->irq, at86rf230_isr, IRQF_SHARED | irq_type,
dev_name(&spi->dev), lp)

ein Interrupt allokiert.

```
28 /**
29  *      devm_request_threaded_irq - allocate an interrupt line for a managed device
30  *      @dev: device to request interrupt for
31  *      @irq: Interrupt line to allocate
32  *      @handler: Function to be called when the IRQ occurs
33  *      @thread_fn: function to be called in a threaded interrupt context. NULL
34  *                  for devices which handle everything in @handler
35  *      @irqflags: Interrupt type flags
36  *      @devname: An ascii name for the claiming device
37  *      @dev_id: A cookie passed back to the handler function
38  *
39  *      Except for the extra @dev argument, this function takes the
40  *      same arguments and performs the same function as
41  *      request_threaded_irq().  IRQs requested with this function will be
42  *      automatically freed on driver detach.
43  *
44  *      If an IRQ allocated with this function needs to be freed
45  *      separately, devm_free_irq() must be used.
46  */
```

Den Abschluss bildet der Aufruf der Funktion

ieee802154_register_hw(lp->hw).

Notwendig ist dies, um die Hardware zu registrieren. Dabei ist es wichtig, dass dies vor allen anderen Aufrufen in *mac802154* geschieht. Als Argument wir hier der Rückgabewert von *ieee802154_alloc_hw()* benötigt. Außerdem müssen dafür alle notwendigen Informationen des wpan_phy's eingestellt werden.

```
317 /**
318  * ieee802154_register_hw - Register hardware device
319  *
320  * You must call this function before any other functions in
321  * mac802154. Note that before a hardware can be registered, you
322  * need to fill the contained wpan_phy's information.
323  *
324  * @hw: the device to register as returned by ieee802154_alloc_hw()
325  *
326  * Return: 0 on success. An error code otherwise.
327  */
```

2.2 Statuswechsel

Der Basic Operation Mode befasst sich mit allen Status, welche der Chip besitzt und einnehmen kann. Dies umfasst die grundlegenden Funktionen wie

- Empfangen
- Senden
- Einschalten
- Sleep-Modus

und ist für die IEEE 802.15.4 Standard entworfen worden. Eine Übersicht mit den Wechseln zwischen den Status ist in **Fehler! Verweisquelle konnte nicht gefunden werden.** dargestellt.

Bei dem Ausführen eines Statuswechsels wird die Funktion

at86rf230_async_state_change()

aufgerufen.

Diese Daten, welche *at86rf230_async_state_change* erhalten hatte, werden dann an die Funktion

at86rf230_async_read_reg()

übergeben. Hier geschieht der Aufruf der Funktion

spi_async(lp->spi, &ctx->msg),

welche für den asynchronen Transfer über SPI zuständig ist und das gewünschte Register ausliest. Der Wert wird in ctx->buf abgelegt und steht somit auch im weiteren Verlauf zur Verfügung. Der Parameter *&ctx->msg* beschreibt den Datentransfer, welche ebenfalls die Callback-Funktion der Completion-Struktur enthält. In diesem Fall handelt es sich dabei um die Funktion

at86rf230_async_state_change_start.

Hier wird nun das Ändern des Status durchgeführt. Falls sich das Device noch in einem Statuswechsel befindet, wird mit dem angefordertem Wechsel gewartet und das Auslesen der Register erneut aufgerufen. Außerdem wird der Status dahingehend geprüft, ob sich das Device schon in dem gewünschten Zustand befindet. Aus diesem Grund wird vor jeden Statuswechsel die oben erwähnte Funktion *at86rf230_async_read_reg()* aufgerufen. Sind die beiden genannten Fälle ausgeschlossen, kann der Wechsel mit

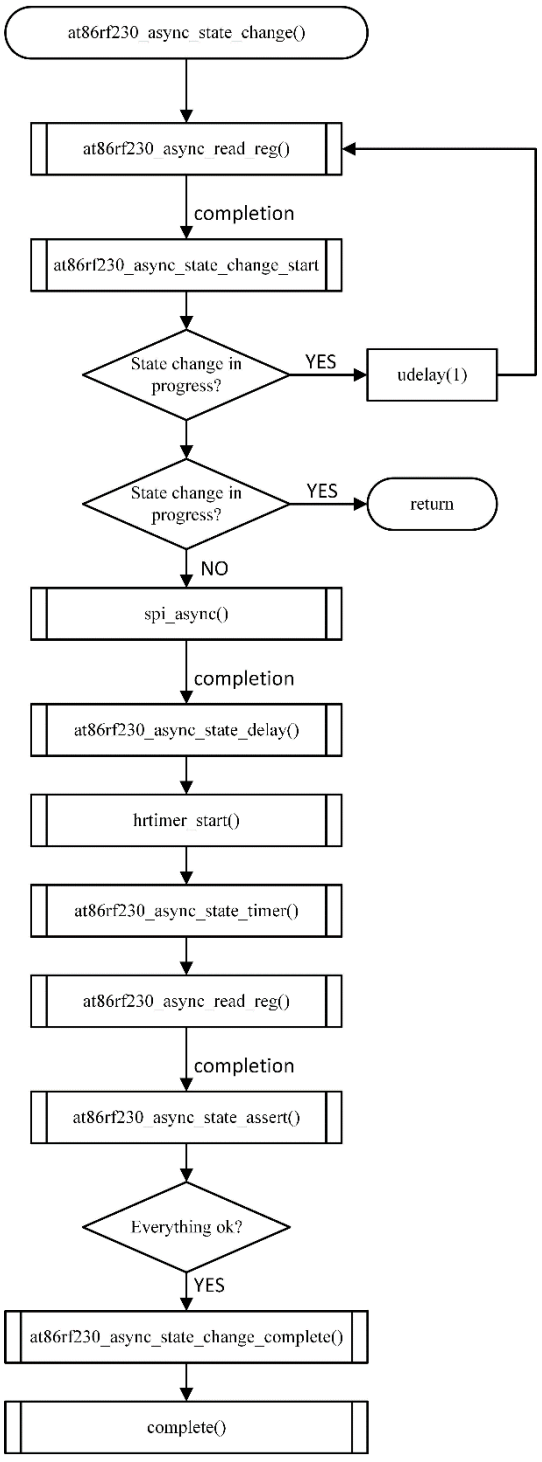


Abbildung 3: Ablauf eines Statuswechsels

spi_async() durchgeführt werden. In diesem Fall wird das Register nicht gelesen sondern via SPI beschrieben. Die Kommunikationsrichtung wird über führende Bits markiert.

Weitere wichtige Informationen zu dem asynchronen Transfer per SPI finden sich in dem Kommentar zur entsprechenden Funktion.

```
2253  * This call may be used in_irq and other contexts which can't sleep,
2254  * as well as from task contexts which can sleep.
2255  *
2256  * The completion callback is invoked in a context which can't sleep.
2257  * Before that invocation, the value of message->status is undefined.
2258  * When the callback is issued, message->status holds either zero (to
2259  * indicate complete success) or a negative error code.  After that
2260  * callback returns, the driver which issued the transfer request may
2261  * deallocate the associated memory; it's no longer in use by any SPI
2262  * core or controller driver code.
2263  *
2264  * Note that although all messages to a spi_device are handled in
2265  * FIFO order, messages may go to different devices in other orders.
2266  * Some device might be higher priority, or have various "hard" access
2267  * time requirements, for example.
2268  *
2269  * On detection of any fault during the transfer, processing of
2270  * the entire message is aborted, and the device is deselected.
2271  * Until returning from the associated message completion callback,
2272  * no other spi_message queued to that device will be processed.
2273  * (This rule applies equally to all the synchronous transfer calls,
2274  * which are wrappers around this core asynchronous primitive.)
2275  *
2276  * Return: zero on success, else a negative error code.
2277  */
```

Wenn die asynchrone Übertragung des SPI Modules abgeschlossen ist, wird als Completion-Funktion

at86rf230_async_state_delay()

aufgerufen. Hier werden die für den Status spezifischen Wartezeiten gesetzt. Gestartet wird der Timer über den Aufruf von

hrtimer_start().

Nach Ablauf des Timers wird in der Funktion

at86rf230_async_state_timer()

erneut ein Auslesen des veränderten Registers angestoßen. Diesmal wird allerdings als Completion-Funktion nicht die Funktion zum Starten eines Statuswechsels übergeben, sondern die Funktion

at86rf230_async_state_assert().

Diese Implementierung dient einer Abschließenden Überprüfung, ob der Statuswechsel erfolgreich war. Sollte dies der Fall sein und keinerlei Fehler aufgetreten sein, wird über die Funktion

at86rf230_state_change_complete()

die completion-Funktion *complete()* aus dem completion-Stack aufgerufen.

Abbildung 4: Extended Operating Mode

3 Wichtige Strukturen

Das Module AT86RF230 enthält fünf wichtige Datensätze. Dazu zählen die Structs:

- at86rf230_chip_data
- at86rf230_state_change
- at86rf230_local
- regmap_config
- ieee802154_ops

Die erste Datenstruktur beinhaltet die auf den Chip bezogenen Einstellungen. Dies umfasst hauptsächlich zeitbezogene Definitionen, welche in Mikrosekunden angegeben werden.

```
struct at86rf2xx chip data {
          u16 t_sleep_cycle;
          u16 t_channel_switch;
          u16 t_reset_to_off;
          u16 t off to aack;
          u16 t off to tx on;
          u16 t frame;
          u16 t_p_ack;
          int rssi_base_val;

          int (*set channel)(struct at86rf230 local *, u8, u8);
          int (*get desense steps)(struct at86rf230 local *, s32);
};
```

Der nächste Datentyp at86rf230_state_change wurde bereits oben angesprochen. Dieser dient, wie der Name schon andeutet, für den Wechsel eines Operation Modes.

```
struct at86rf230 state change {
          struct at86rf230 local *lp;
          int irq;

          struct hrtimer timer;
          struct spi_message msg;
          struct spi transfer trx;
          u8 buf[AT86RF2XX MAX BUF];

          void (*complete)(void *context);
          u8 from_state;
          u8 to_state;

          bool irq enable;
};
```

Der dritte wichtige Datentyp beschreibt und speichert alle relevanten Daten und Informationen über das Device. Dies beinhaltet die Daten für die SPI Kommunikation, die Hardware bezogenen Daten aus dem IEEE802154 Stack und die Einstellungen für die Registration Map *regmap*. Außerdem werden Strukturen vom Typ *completion* und *at86rf230_state_change* angelegt.

```
struct at86rf230_local {
          struct spi_device *spi;

          struct ieee802154_hw *hw;
          struct at86rf2xx_chip_data *data;
          struct regmap *regmap;
          int slp_tr;

          struct completion state_complete;
          struct at86rf230_state_change state;

          struct at86rf230_state_change irq;

          bool tx_aret;
          unsigned long cal_timeout;
          s8 max_frame_retries;
          bool is_tx;
          bool is_tx_from_off;
          u8 tx_retry;
          struct sk_buff *tx_skb;
          struct at86rf230_state_change tx;
};
```

Eine wichtige Rolle spielt die Datenstruktur *regmap_config*. Diese enthält die Einstellungen für die Register Map eines Devices.

```
static const struct regmap_config at86rf230_regmap_spi_config = {
          .reg_bits = 8,
          .val_bits = 8,
          .write_flag_mask = CMD_REG | CMD_WRITE, //0x80 | 0x40 - write access
          .read_flag_mask = CMD_REG,             //0x80 - read access
          .cache_type = REGCACHE_RBTREE,
          .max_register = AT86RF2XX_NUMREGS,
          .writeable_reg = at86rf230_reg_writeable,
          .readable_reg = at86rf230_reg_readable,
          .volatile_reg = at86rf230_reg_volatile,
          .precious_reg = at86rf230_reg_precious,
};
```

Eine besondere Rolle spielen hier die letzten vier Funktionen. Diese beschreiben die Eigenschaften der zur Verfügung stehenden Register (siehe Tabelle 1).

Tabelle 1: Funktionen für die Ermittlung der Registereigenschaften

Strukturvariable	**Funktionsname**	**Aufgabe**
.writeable_reg	*at86rf230_reg_writeable*	Rückgabewert ist *true*, wenn das Register beschrieben werden darf
.readable_reg	at86rf230_reg_readable	Rückgabewert ist *true*, wenn das Register ein Read-Only Register ist
.volatile_reg	at86rf230_reg_volatile	Register, welche während der Laufzeit nicht geändert werden können
.precious_reg	at86rf230_precious	Sollte nicht von außerhalb des Treibers aufgerufen werden

```
113 /**
114  * Configuration for the register map of a device.
115  *
116  * @name: Optional name of the regmap. Useful when a device has multiple
117  *        register regions.
118  *
119  * @reg_bits: Number of bits in a register address, mandatory.
120  * @reg_stride: The register address stride. Valid register addresses are a
121  *              multiple of this value. If set to 0, a value of 1 will be
122  *              used.
123  * @pad_bits: Number of bits of padding between register and value.
124  * @val_bits: Number of bits in a register value, mandatory.
125  *
126  * @writeable_reg: Optional callback returning true if the register
```

```
127  *                 can be written to. If this field is NULL but wr_table
128  *                 (see below) is not, the check is performed on such table
129  *                 (a register is writeable if it belongs to one of the ranges
130  *                  specified by wr_table).
131  * @readable_reg: Optional callback returning true if the register
132  *                can be read from. If this field is NULL but rd_table
133  *                 (see below) is not, the check is performed on such table
134  *                 (a register is readable if it belongs to one of the ranges
135  *                  specified by rd_table).
136  * @volatile_reg: Optional callback returning true if the register
137  *                value can't be cached. If this field is NULL but
138  *                volatile_table (see below) is not, the check is performed on
139  *                such table (a register is volatile if it belongs to one of
140  *                the ranges specified by volatile_table).
141  * @precious_reg: Optional callback returning true if the register
142  *                should not be read outside of a call from the driver
143  *                (e.g., a clear on read interrupt status register). If this
144  *                field is NULL but precious_table (see below) is not, the
145  *                check is performed on such table (a register is precious if
146  *                it belongs to one of the ranges specified by precious_table).
147  * @lock:         Optional lock callback (overrides regmap's default lock
148  *                function, based on spinlock or mutex).
149  * @unlock:       As above for unlocking.
150  * @lock_arg:     this field is passed as the only argument of lock/unlock
151  *                functions (ignored in case regular lock/unlock functions
152  *                are not overridden).
153  * @reg_read:     Optional callback that if filled will be used to perform
154  *                all the reads from the registers. Should only be provided for
155  *                devices whose read operation cannot be represented as a simple
156  *                read operation on a bus such as SPI, I2C, etc. Most of the
157  *                devices do not need this.
158  * @reg_write:    Same as above for writing.
159  * @fast_io:      Register IO is fast. Use a spinlock instead of a mutex
160  *                to perform locking. This field is ignored if custom lock/unlock
161  *                functions are used (see fields lock/unlock of struct regmap_config).
162  *                This field is a duplicate of a similar file in
163  *                'struct regmap_bus' and serves exact same purpose.
164  *                 Use it only for "no-bus" cases.
165  * @max_register: Optional, specifies the maximum valid register index.
166  * @wr_table:     Optional, points to a struct regmap_access_table specifying
167  *                valid ranges for write access.
168  * @rd_table:     As above, for read access.
169  * @volatile_table: As above, for volatile registers.
170  * @precious_table: As above, for precious registers.
171  * @reg_defaults: Power on reset values for registers (for use with
172  *                register cache support).
173  * @num_reg_defaults: Number of elements in reg_defaults.
174  *
175  * @read_flag_mask: Mask to be set in the top byte of the register when doing
176  *                  a read.
177  * @write_flag_mask: Mask to be set in the top byte of the register when doing
178  *                   a write. If both read_flag_mask and write_flag_mask are
179  *                   empty the regmap_bus default masks are used.
180  * @use_single_rw: If set, converts the bulk read and write operations into
181  *                  a series of single read and write operations. This is useful
182  *                  for device that does not support bulk read and write.
183  * @can_multi_write: If set, the device supports the multi write mode of bulk
184  *                   write operations, if clear multi write requests will be
185  *                   split into individual write operations
186  *
187  * @cache_type: The actual cache type.
188  * @reg_defaults_raw: Power on reset values for registers (for use with
189  *                     register cache support).
190  * @num_reg_defaults_raw: Number of elements in reg_defaults_raw.
191  * @reg_format_endian: Endianness for formatted register addresses. If this is
192  *                     DEFAULT, the @reg_format_endian_default value from the
193  *                     regmap bus is used.
194  * @val_format_endian: Endianness for formatted register values. If this is
195  *                     DEFAULT, the @reg_format_endian_default value from the
196  *                     regmap bus is used.
197  *
198  * @ranges: Array of configuration entries for virtual address ranges.
199  * @num_ranges: Number of range configuration entries.
200  */
```

In der letzten Struktur müssen alle relevanten Callbacks von mac802154 zum Treiber hin abgelegt werden. Dies umfasste eine ganze Reihe an notwenigen Funktionen, wie zum Beispiel eine start- und stop-Funktion für die Initialisierung und Bereinigung des Devices. Die genauere Beschreibung und der Aufbau der einzelnen Funktionen wird im nächsten Kapitel genauer erläutert.

```
static const struct ieee802154 ops at86rf230 ops = {
          .owner = THIS_MODULE,
          .xmit_async = at86rf230_xmit,
          .ed = at86rf230_ed,
          .set channel = at86rf230 channel,
          .start = at86rf230 start,
          .stop = at86rf230 stop,
          .set_hw_addr_filt = at86rf230_set_hw_addr_filt,
          .set_txpower = at86rf230_set_txpower,
          .set_lbt = at86rf230_set_lbt,
          .set cca mode = at86rf230 set cca mode,
          .set cca ed level = at86rf230 set cca ed level,
          .set_csma_params = at86rf230_set_csma_params,
          .set_frame_retries = at86rf230_set_frame_retries,
          .set_promiscuous_mode = at86rf230_set_promiscuous_mode,
};
```

```
142 /* struct ieee802154_ops - callbacks from mac802154 to the driver
143  *
144  * This structure contains various callbacks that the driver may
145  * handle or, in some cases, must handle, for example to transmit
146  * a frame.
147  *
148  * start: Handler that 802.15.4 module calls for device initialization.
149  *        This function is called before the first interface is attached.
150  *
151  * stop:  Handler that 802.15.4 module calls for device cleanup.
152  *        This function is called after the last interface is removed.
153  *
154  * xmit_sync:
155  *        Handler that 802.15.4 module calls for each transmitted frame.
156  *        skb cntains the buffer starting from the IEEE 802.15.4 header.
157  *        The low-level driver should send the frame based on available
158  *        configuration. This is called by a workqueue and useful for
159  *        synchronous 802.15.4 drivers.
160  *        This function should return zero or negative errno.
161  *
162  *        WARNING:
163  *        This will be deprecated soon. We don't accept synced xmit callbacks
164  *        drivers anymore.
165  *
166  * xmit_async:
167  *        Handler that 802.15.4 module calls for each transmitted frame.
168  *        skb cntains the buffer starting from the IEEE 802.15.4 header.
169  *        The low-level driver should send the frame based on available
170  *        configuration.
171  *        This function should return zero or negative errno.
172  *
173  * ed:    Handler that 802.15.4 module calls for Energy Detection.
174  *        This function should place the value for detected energy
175  *        (usually device-dependant) in the level pointer and return
176  *        either zero or negative errno. Called with pib_lock held.
177  *
178  * set_channel:
179  *        Set radio for listening on specific channel.
180  *        Set the device for listening on specified channel.
181  *        Returns either zero, or negative errno. Called with pib_lock held.
182  *
183  * set_hw_addr_filt:
184  *        Set radio for listening on specific address.
185  *        Set the device for listening on specified address.
186  *        Returns either zero, or negative errno.
187  *
188  * set_txpower:
189  *        Set radio transmit power in mBm. Called with pib_lock held.
190  *        Returns either zero, or negative errno.
191  *
192  * set_lbt
193  *        Enables or disables listen before talk on the device. Called with
194  *        pib_lock held.
195  *        Returns either zero, or negative errno.
196  *
197  * set_cca_mode
198  *        Sets the CCA mode used by the device. Called with pib_lock held.
199  *        Returns either zero, or negative errno.
200  *
201  * set_cca_ed_level
202  *        Sets the CCA energy detection threshold in mBm. Called with pib_lock
203  *        held.
204  *        Returns either zero, or negative errno.
205  *
206  * set_csma_params
207  *        Sets the CSMA parameter set for the PHY. Called with pib_lock held.
208  *        Returns either zero, or negative errno.
209  *
210  * set_frame_retries
211  *        Sets the retransmission attempt limit. Called with pib_lock held.
212  *        Returns either zero, or negative errno.
213  *
214  * set_promiscuous_mode
215  *        Enables or disable promiscuous mode.
216  */
```

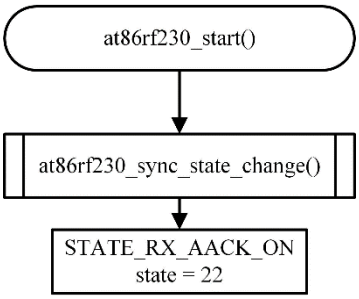
4 IEEE802154_OPS Funktionen

4.1 AT86RF230_START

Wenn das erste Mal eine Übertragung eingeleitet wird, wird vom IEEE 802.15.4 Layer die Funktion

at86rf230_start()

aufgerufen. Diese sorgt lediglich dafür, dass mittels *at86rf230_sync_state_change()* ein Statuswechsel in den Status STATE_RX_AACK_ON (state = 22) eingeleitet wird.



4.2 AT86RF230_SET_PROMISCUOUS_MODE

Der Aufruf der Funktion

at86rf230_set_promiscuous_mode()

wird ebenfalls durch den IEEE Layer geregelt. Sie dient dazu den Promiscuous Modus (in diesem Fall hier) zu deaktivieren. Dies geschieht über die Funktion *at86rf230_write_subreg()*. Betroffen sind dabei die Register 0x17 und 0x2E, in welchen das erste bzw. das vierte Bit auf „0“ gesetzt wird.

4.3 AT86RF230_SET_HW_ADDR_FILT

Im weiteren Verlauf der Initialisierungsphase für die erste Übertragung werden die Register für den Frame Filter beschrieben. Dafür wird insgesamt drei mal die Funktion

at86rf230_set_hw_addr_filter()

aufgerufen. Dort werden dann je nach dem mit welchen Parametern die Funktion aufgerufen wird, mittels *__at86rf230_write()* verschiedene Register angepasst.

Im ersten Aufruf betrifft das die Register PAN_ID_0 und PAN_ID_1 (0x22 und 0x23), welche die Information über die MAC PAN ID erhalten. Dies ist die Adresse, welche ebenfalls bei der Initialisierung der WPAN Devices eingestellt wird (hier: 0xdead). Diese findet sich ebenfalls in dem weiter unten beschriebenen Frame-Buffer wieder. Dort handelt es sich um die Array-Einträge buf[5] und buf[6].

Der zweite Aufruf beschreibt die Register IEEE_ADDR_0 bis IEEE_ADDR_1 (0x24 bis 0x2B), welche die MAC IEEE Frame Filter addresses for address recognation darstellen. Es werden die Daten [8D, 1F, 10, EB, 26, C5, 57, 2C] in die Register geschrieben.

Der dritte Aufruf speichert die MAC short address in den Registern SHORT_ADDR_0 und SHORT_ADDR_1 (0x20 und 0x21). Hier handelt es sich ebenfalls um die Adresse, welche bei der Initialisierung des WPAN Devices eingestellt wird (0xbeef). In dem Frame-Buffer handelt es sich um die Einträge buf[8] und buf[9].

4.4 AT86RF230_SET_CSMA_PARAMS

Diese Funktion dient dem Setzen der Einstellungen für den Carrier Sense Multiple Access (CSMA). Dafür wird die Funktion

at86rf230_set_csma_params()

verwendet, welche mit *at86rf230_write_subreg()* das Register CSMA_BE (0x2F) beschreibt. Hier werden die maximalen und minimalen back-off Exponenten für den CSMA-CA Algorithmus gesetzt. Durch die return-Anweisung ruft *at86rf230_set_csma_params()* die Funktion at86rf230_write_subreg() auf und stellt so im Register XAH_CTRL_0 (0x2C) die maximale Anzahl an

Wiederholungen ein, welche im ARET Modus durchgeführt werden sollen, bevor der Vorgang abgebrochen wird.

4.5 AT86RF230_XMIT

Um Daten von dem Raspberry an den AT86RF231 zu übertragen und diese anschließend zu senden, wird zunächst auf die Funktion

*at86rf230_xmit(struct ieee802154_hw *hw, struct sk_buf *skb),*

welche in dem *ieee802154_ops* Struct unter *.xmit_async* abgelegt ist, aufgerufen. Der Ablauf eines solchen Vorganges ist in Abbildung 5 dargestellt. Als Argument bekommt diese Funktion einen struct von Typen Socket Buffer übergeben. Der wichtigste Inhalt dieses Structes sind die Daten, welche das Device senden soll und sich unter @data befinden. Die Kommentare zu dem Struct befinden sich im Anhang unter im 5.1.

Nun wird überprüft, wie lange die PLL schon auf der gleichen Frequenz läuft. Sollte die Laufzeit eine Zeit von 5 Minuten überschreiten, wird vom Hersteller eine Neukalibrierung empfohlen. Diese Kalibrierung wird hervorgerufen durch einen Statuswechsel von TRX_OFF zu TX_ON. Daher wird bei einer notwendigen Kalibrierung mittels *at86rf230_async_state_change(lp, ctx, STATE_TRX_OFF, at86rf230_xmit_start, false)* die Übertagung ausgeschaltet. Bei Abschluss dieses Vorgangs wird, wie auch bei einer nicht notwendigen Kalibrierung, die Funktion

at86rf230_xmit_start(ctx)

aufgerufen. Hier erfolgt als erstes eine Abfrage, ob die Übertragung im Automatic Retransmission Modus (ARET) ausgeführt werden soll. Sollte dies der Fall sein, muss der Status in STATE_TX_ARET_ON gewechselt werden. Dafür wird als completion-Funktion die Funktion

at86rf230_xmit_tx_on(ctx)

übergeben. Dadurch wird der gewollte Wechsel des Status veranlasst und nach dessen Abschluss die Funktion

at86rf230_write_frame(ctx)

durch die completion-Struktur aufgerufen. Dies geschieht ebenfalls, wenn das System nicht im ARET Modus betrieben werden soll. Hier wird allerdings nur in den State 9 gewechselt, was dem Aktivieren der PLL entspricht.

Von *at86rf230_write_frame()* wird nun das Übertragen des Frames an das Device veranlasst. Dafür wird ein Array *u8 *buf* erstellt. Um ein Schreibvorgang zu kennzeichnen, müssen die MSBs des ersten Bytes 0 1 1 sein. Außerdem muss das zweite Byte die Länge des Frames plus 2 beinhalten. Nach diesem Setup wird nun das gewünschte Frame dem Buffer angehangen, welches der Funktion *at86rf230_xmit()* zu Beginn übergeben wurde. Da die Funktion durch den IEEE802.15.4 Layer aufgerufen wird, enthalten die ersten acht Bytes des Buffers Informationen über den IEEE802.15.4 Header. Mittels *spi_async(lp->spi, &ctx->msg)* wird die Übertragung des Buffers via SPI an das Device veranlasst. Als completion-Funktion wird hier

at86rf230_write_frame_complete()

übergeben. Falls ein gültige GPIO für *slp_tr* erkannt wurde, wird mit

at86rf230_slp_tr_rising_edge(lp)

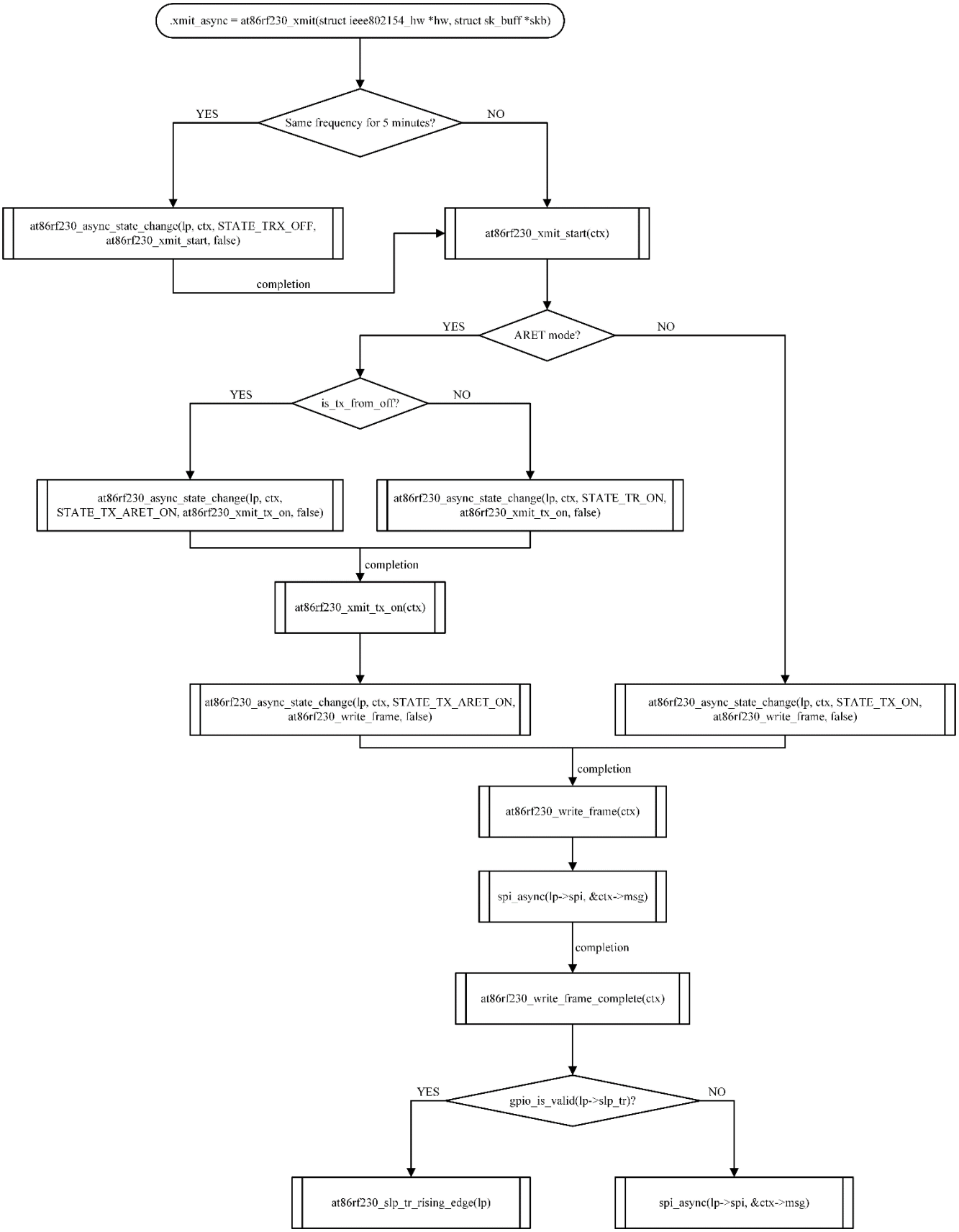


Abbildung 5: Flow-Chart für den Ablauf der at86rf230_xmit Funktion

eine steigende Flanke am Pin SLP_TR erzeugt. Die Auswirkungen dieser Flanke sind abhängig von dem jeweiligen Status, in dem sich das Device befindet. Die Abhängigkeiten und Auswirkungen sind in Tabelle 2 aufgelistet. In dem hier konkreten Fall befindet sich das Device in dem Status PLL_ON, was bedeutet, dass bei einer steigenden Flanke das Senden des Frames veranlasst wird.

Radio Transceiver Status	Function	Transition	Description
TRX_OFF	Sleep	L → H	Takes the radio transceiver into SLEEP state
SLEEP	Wakeup	H → L	Takes the radio transceiver into TRX_OFF state
RX_ON	Disable CLKM	L → H	Takes the radio transceiver into RX_ON_NOCLK state and disables CLKM
RX_ON_NOCLK	Enables CLKM	H → L	Takes the radio transceiver into RX_ON state and enables CLKM
RX_AACK_ON	Disable CLKM	L → H	Takes the radio transceiver into RX_AACK_ON_NOCLK state and disables CLKM
RX_AACK_ON_NOCLK	Enables CLKM	H → L	Takes the radio transceiver into RX_AACK_ON state and enables CLKM
PLL_ON	TX start	L → H	Starts frame transmission
TX_ARET_ON	TX start	L → H	Starts TX_ARET transaction

Tabelle 2: Auswirkungen von Pegeländerungen am Pin SLP_TR

4.6 AT86RF230_ISR

Die Funktion

at86rf230_isr()

wird nach jedem Aufruf von *at86rf230_xmit()* durch den IEEE Layer aufgerufen. Der zeitliche Verlauf ist in Abbildung 6 dargestellt. Hier wird dann mittels *spi_async()* das Register IRQ_STATUS ausgelesen. Als Callback-Funktion wird die Funktion

at86rf230_irq_status()

übergeben. Hier wird nun mit dem vorher ausgelesenem Registerwert von *spi_async()* überprüft, ob der Sende- bzw. Empfangsvorgang abgeschlossen ist. Sollte dies der Fall sein, wird die Funktion

at86rf230_irq_trx_end(lp)

aufgerufen. Diese dient dazu als erstes die Richtung der Übertragung zu ermitteln. Dazu wird abgefragt, ob man sich im Sendemodus befindet. Falls dies der Fall ist, erfolgt eine weitere Abfrage bezüglich der Aktivierung des ARET Modus. Hier wird erstmal nur auf diesen Fall eingegangen, dass sich das Device im Sendemodus und nicht im ARET Modus befindet.

Im Anschluss an diese Abfragen wird in dem hier behandelten Fall mittels der Funktion *at86rf230_async_state_change()* der Operating Mode in den Status STATE_RX_AACK_ON versetzt. Als Callback-Funktion wird hier die Funktion

at86rf230_tx_complete()

definiert. Diese sorgt zum Abschluss noch für den Aufruf der Funktion

ieee802154_xmit_complete()

aus dem IEEE Layer. Durch diesen Schritt ist ein Sendevorgang komplett abgeschlossen.

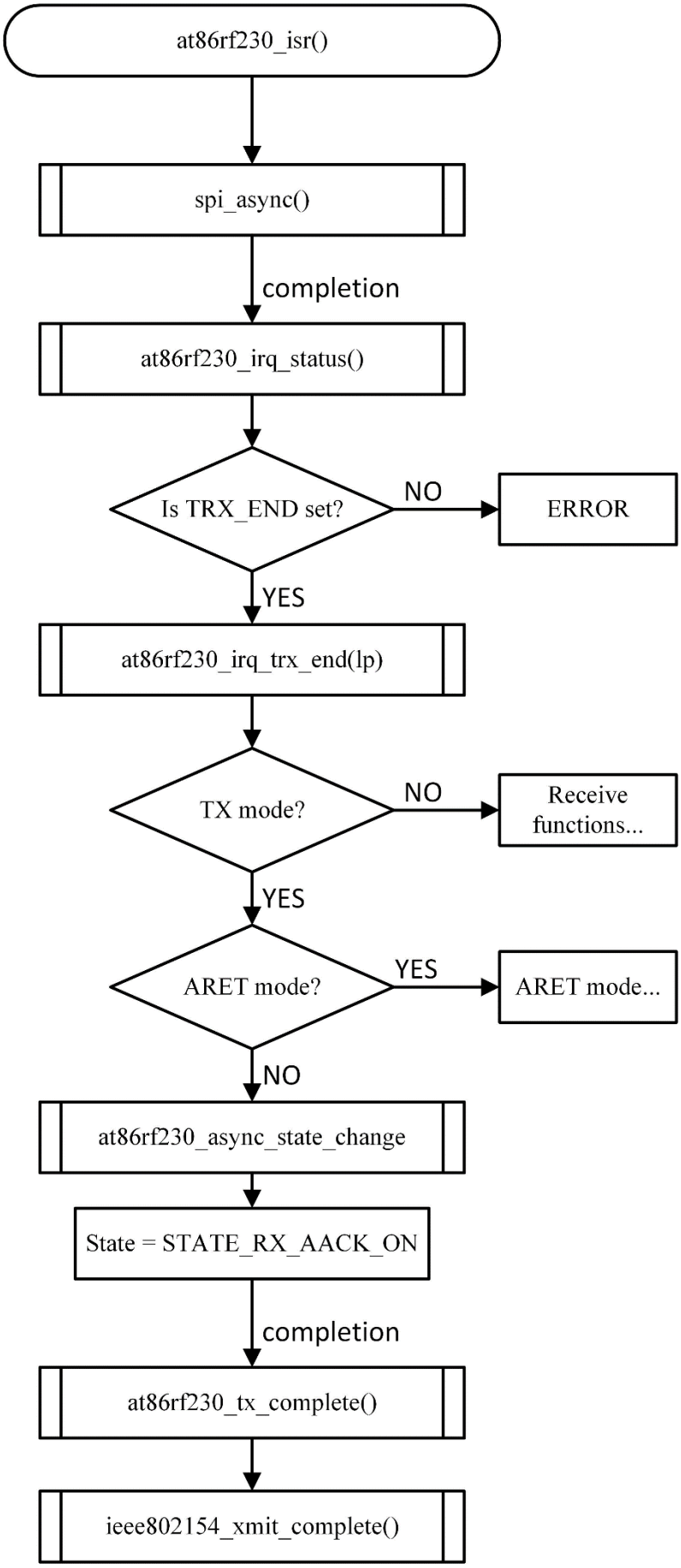


Abbildung 6: Ablauf nach dem Aufruf der Funktion at86rf230_isr

4.7 AT86RF230_ED

Die Funktion

at86rf230_ed

dient dem Speichern des Wertes der detektierbaren Energie in dem Pointer *level*.

5 Anhang

```
at86rf230: at86rf230_probe - start: /home/pi/linux/drivers/net/ieee802154/at86rf230.c:1693
at86rf230: at86rf230_probe - spi->irq: 417: /home/pi/linux/drivers/net/ieee802154/at86rf230.c:1694
at86rf230: at86rf230_get_pdata - 2 rstn: 24, slp_tr: 25, xtal_trim: 15: /home/pi/linux/drivers/net/ieee802154/at86rf230.c:1551
at86rf230: at86rf230_probe - after init spi /home/pi/linux/drivers/net/ieee802154/at86rf230.c:1751
at86rf230: at86rf230_setup_spi_messages - start /home/pi/linux/drivers/net/ieee802154/at86rf230.c:1649
at86rf230: at86rf230_detect_device - start: /home/pi/linux/drivers/net/ieee802154/at86rf230.c:1566
at86rf230: at86rf230_detect_device - befor read: val=3662662664; /home/pi/linux/drivers/net/ieee802154/at86rf230.c:1569
at86rf230: __at86rf230_read - ragmap_read is called with addr: 30, data: 3662662664. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:318
at86rf230: at86rf230_detect_device - after read: val=31; /home/pi/linux/drivers/net/ieee802154/at86rf230.c:1571
at86rf230: at86rf230_detect_device - befor read: val=31; /home/pi/linux/drivers/net/ieee802154/at86rf230.c:1576
at86rf230: __at86rf230_read - ragmap_read is called with addr: 31, data: 31. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:318
at86rf230: at86rf230_detect_device - after read: val=0; /home/pi/linux/drivers/net/ieee802154/at86rf230.c:1578
at86rf230: __at86rf230_read - ragmap_read is called with addr: 28, data: 3662662656. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:318
at86rf230: __at86rf230_read - ragmap_read is called with addr: 29, data: 0. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:318
at86rf230: at86rf230_detect_device - variables: val=0, part=3, version=2; /home/pi/linux/drivers/net/ieee802154/at86rf230.c:1591
at86rf230 spi0.0: Detected at86rf231 chip version 2
at86rf230: at86rf230_probe - after device detection: /home/pi/linux/drivers/net/ieee802154/at86rf230.c:1766
at86rf230: at86rf230_hw_init - start. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:1418
at86rf230: at86rf230_sync_state_change - start. state=3. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:767
at86rf230: at86rf230_async_state_change - state=3. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:739
at86rf230: at86rf230_async_read_reg - start. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:503
at86rf230: at86rf230_async_read_reg - start. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:503
at86rf230: at86rf230_sync_state_change - end./home/pi/linux/drivers/net/ieee802154/at86rf230.c:780
at86rf230: at86rf230_write_subreg - address=4, mask=1, shift=0, data=0. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:343
at86rf230: at86rf230_write_subreg - address=12, mask=128, shift=7, data=1. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:343
at86rf230: at86rf230_write_subreg - address=14, mask=255, shift=0, data=8. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:343
at86rf230: at86rf230_write_subreg - address=4, mask=2, shift=1, data=0. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:343
at86rf230: at86rf230_write_subreg - address=45, mask=255, shift=0, data=243. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:343
at86rf230: at86rf230_write_subreg - address=46, mask=7, shift=0, data=212. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:343
at86rf230: at86rf230_write_subreg - address=3, mask=8, shift=3, data=0. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:343
at86rf230: at86rf230_write_subreg - address=3, mask=7, shift=0, data=0. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:343
at86rf230: at86rf230_write_subreg - address=18, mask=15, shift=0, data=15. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:343
at86rf230: at86rf230_read_subreg - address=16, mask=4. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:329
at86rf230: __at86rf230_read - ragmap_read is called with addr: 16, data: 0. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:318
at86rf230: at86rf230_hw_init - end. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:1519
at86rf230: at86rf230_write_subreg - address=44, mask=1, shift=0, data=0. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:343
at86rf230: at86rf230_read_subreg - address=15, mask=255. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:329
at86rf230: __at86rf230_read - ragmap_read is called with addr: 15, data: 1. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:318
at86rf230: at86rf230_probe - end: /home/pi/linux/drivers/net/ieee802154/at86rf230.c:1806
```

Abbildung 7: Ausgabe bei der Einbindung von AT86RF230.ko

5.1 Beschreibung des Socket Buffers

```
477  *      struct sk_buff - socket buffer
478  *      @next: Next buffer in list
479  *      @prev: Previous buffer in list
480  *      @tstamp: Time we arrived/left
481  *      @rbnode: RB tree node, alternative to next/prev for netem/tcp
482  *      @sk: Socket we are owned by
483  *      @dev: Device we arrived on/are leaving by
484  *      @cb: Control buffer. Free for use by every layer. Put private vars here
485  *      @_skb_refdst: destination entry (with norefcount bit)
486  *      @sp: the security path, used for xfrm
487  *      @len: Length of actual data
488  *      @data_len: Data length
489  *      @mac_len: Length of link layer header
490  *      @hdr_len: writable header length of cloned skb
491  *      @csum: Checksum (must include start/offset pair)
492  *      @csum_start: Offset from skb->head where checksumming should start
493  *      @csum_offset: Offset from csum_start where checksum should be stored
494  *      @priority: Packet queueing priority
495  *      @ignore_df: allow local fragmentation
496  *      @cloned: Head may be cloned (check refcnt to be sure)
497  *      @ip_summed: Driver fed us an IP checksum
498  *      @nohdr: Payload reference only, must not modify header
499  *      @nfctinfo: Relationship of this skb to the connection
500  *      @pkt_type: Packet class
501  *      @fclone: skbuff clone status
502  *      @ipvs_property: skbuff is owned by ipvs
503  *      @peeked: this packet has been seen already, so stats have been
504  *              done for it, don't do them again
505  *      @nf_trace: netfilter packet trace flag
506  *      @protocol: Packet protocol from driver
507  *      @destructor: Destruct function
508  *      @nfct: Associated connection, if any
509  *      @nf_bridge: Saved data about a bridged frame - see br_netfilter.c
510  *      @skb_iif: ifindex of device we arrived on
511  *      @tc_index: Traffic control index
512  *      @tc_verd: traffic control verdict
513  *      @hash: the packet hash
514  *      @queue_mapping: Queue mapping for multiqueue devices
515  *      @xmit_more: More SKBs are pending for this queue
516  *      @ndisc_nodetype: router type (from link layer)
517  *      @ooo_okay: allow the mapping of a socket to a queue to be changed
518  *      @l4_hash: indicate hash is a canonical 4-tuple hash over transport
519  *              ports.
520  *      @sw_hash: indicates hash was computed in software stack
521  *      @wifi_acked_valid: wifi_acked was set
522  *      @wifi_acked: whether frame was acked on wifi or not
523  *      @no_fcs:  Request NIC to treat last 4 bytes as Ethernet FCS
524   *     @napi_id: id of the NAPI struct this skb came from
525  *      @secmark: security marking
526  *      @offload_fwd_mark: fwding offload mark
527  *      @mark: Generic packet mark
528  *      @vlan_proto: vlan encapsulation protocol
529  *      @vlan_tci: vlan tag control information
530  *      @inner_protocol: Protocol (encapsulation)
531  *      @inner_transport_header: Inner transport layer header (encapsulation)
532  *      @inner_network_header: Network layer header (encapsulation)
533  *      @inner_mac_header: Link layer header (encapsulation)
534  *      @transport_header: Transport layer header
535  *      @network_header: Network layer header
536  *      @mac_header: Link layer header
537  *      @tail: Tail pointer
538  *      @end: End pointer
539  *      @head: Head of buffer
540  *      @data: Data head pointer
541  *      @truesize: Buffer size
542  *      @users: User count - see {datagram,tcp}.c
```

5.2 Ausgaben bei einem Sendevorgang

```
// device initialization - is called befor the first transmission
[ 5444.959157] at86rf230: at86rf230 start: /home/pi/linux/drivers/net/ieee802154/at86rf230.c:1182

// is called directly from at86rf230_start
[ 5444.959195] at86rf230: at86rf230_sync_state_change - start. state=22. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:809
               [ 5444.959212] at86rf230: at86rf230_async_state_change - start, state=22. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:775
                              [ 5444.959227] at86rf230: at86rf230_async_read_reg - start. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:516
                              [ 5444.959272] at86rf230: at86rf230_async_read_reg - end. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:528
               [ 5444.959292] at86rf230: at86rf230_async_state_change - end, state=22. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:784

               [ 5444.959381] at86rf230: at86rf230 async state change start - start. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:729
               [ 5444.959407] at86rf230: at86rf230 async state change start - end. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:764

               [ 5444.959443] at86rf230: at86rf230_async_state_delay - start. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:635

               [ 5444.959602] at86rf230: at86rf230_async_state_timer - start. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:614
                              [ 5444.959625] at86rf230: at86rf230_async_read_reg - start. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:516
                              [ 5444.959657] at86rf230: at86rf230_async_read_reg - end. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:528
               [ 5444.959673] at86rf230: at86rf230 async state timer - end. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:620

               [ 5444.959742] at86rf230: at86rf230 async state assert - start. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:552
               [ 5444.959762] at86rf230: at86rf230 async state assert - end. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:601

               [ 5444.959779] at86rf230: at86rf230_async_state_change_complete - start. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:793
               [ 5444.959808] at86rf230: at86rf230_async_state_change_complete - end. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:797
[ 5444.959845] at86rf230: at86rf230 sync state change - end./home/pi/linux/drivers/net/ieee802154/at86rf230.c:822

// disable promiscuos mode
[ 5444.959865] at86rf230: at86rf230 set promiscuous mode - start. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:1473
               [ 5444.959884] at86rf230: at86rf230 write subreg - regmap update bits is called with: address=23, mask=2, shift=1, data=0.
/home/pi/linux/drivers/net/ieee802154/at86rf230.c:345
               [ 5444.959996] at86rf230: at86rf230_write_subreg - regmap_update_bits is called with: address=46, mask=16, shift=4, data=0.
/home/pi/linux/drivers/net/ieee802154/at86rf230.c:345
[ 5444.960023] at86rf230: at86rf230_set_promiscuous_mode - end. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:1493

// Frame Filter Address Recognition
// The MAC sublayer shall filter incoming frames and present only the frames that are of interest to the upper layers
// MAC PAN ID
[ 5444.960042] at86rf230: at86rf230_set_hw_addr_filt - start. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:1285
               [ 5444.960058] at86rf230: __at86rf230_write - regmap_write is called with addr=34, data=57005.
/home/pi/linux/drivers/net/ieee802154/at86rf230.c:311
               [ 5444.960145] at86rf230: __at86rf230_write - regmap_write is called with addr=35, data=222.
/home/pi/linux/drivers/net/ieee802154/at86rf230.c:311
[ 5444.960230] at86rf230: at86rf230_set_hw_addr_filt - end. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:1324

[ 5444.960249] at86rf230: at86rf230 set hw addr filt - start. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:1285
               [ 5444.960266] at86rf230:   at86rf230 write - regmap write is called with addr=36, data=141.
/home/pi/linux/drivers/net/ieee802154/at86rf230.c:311
               [ 5444.960348] at86rf230: __at86rf230_write - regmap_write is called with addr=37, data=31.
/home/pi/linux/drivers/net/ieee802154/at86rf230.c:311
               [ 5444.960431] at86rf230: __at86rf230_write - regmap_write is called with addr=38, data=16.
/home/pi/linux/drivers/net/ieee802154/at86rf230.c:311
               [ 5444.960513] at86rf230: __at86rf230_write - regmap_write is called with addr=39, data=235.
/home/pi/linux/drivers/net/ieee802154/at86rf230.c:311
               [ 5444.960599] at86rf230:   at86rf230 write - regmap write is called with addr=40, data=38.
/home/pi/linux/drivers/net/ieee802154/at86rf230.c:311
               [ 5444.960682] at86rf230:   at86rf230 write - regmap write is called with addr=41, data=197.
/home/pi/linux/drivers/net/ieee802154/at86rf230.c:311
               [ 5444.960763] at86rf230: __at86rf230_write - regmap_write is called with addr=42, data=87.
/home/pi/linux/drivers/net/ieee802154/at86rf230.c:311
               [ 5444.960847] at86rf230: __at86rf230_write - regmap_write is called with addr=43, data=44.
/home/pi/linux/drivers/net/ieee802154/at86rf230.c:311
[ 5444.960928] at86rf230: at86rf230 set hw addr filt - end. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:1324

// MAC short address
[ 5444.960949] at86rf230: at86rf230 set hw addr filt - start. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:1285
               [ 5444.960965] at86rf230:   at86rf230 write - regmap write is called with addr=32, data=48879.
/home/pi/linux/drivers/net/ieee802154/at86rf230.c:311
               [ 5444.961046] at86rf230: __at86rf230_write - regmap_write is called with addr=33, data=190.
/home/pi/linux/drivers/net/ieee802154/at86rf230.c:311
[ 5444.961126] at86rf230: at86rf230 set hw addr filt - end. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:1324

// set ED max und min
[ 5444.961147] at86rf230: at86rf230 set csma params - start. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:1433
               [ 5444.961167] at86rf230: at86rf230 write subreg - regmap update bits is called with: address=47, mask=15, shift=0, data=3.
/home/pi/linux/drivers/net/ieee802154/at86rf230.c:345
               [ 5444.961261] at86rf230: at86rf230_write_subreg - regmap_update_bits is called with: address=47, mask=240, shift=4, data=5.
/home/pi/linux/drivers/net/ieee802154/at86rf230.c:345
[ 5444.961290] at86rf230: at86rf230 set csma params - end. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:1443

// return value of at86rf230 set csma params
// set the maximal CSMA retries
[ 5444.961307] at86rf230: at86rf230 write subreg - regmap update bits is called with: address=44, mask=14, shift=1, data=4.
/home/pi/linux/drivers/net/ieee802154/at86rf230.c:345

// deflaut value is set - no changes
[ 5444.961325] at86rf230: at86rf230_set_frame_retries - start. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:1454
[ 5444.961339] at86rf230: at86rf230_set_frame_retries - end. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:1462



// start xmit
[ 5451.440891] at86rf230: at86rf230_xmit - start. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:1144
               [ 5451.441007] at86rf230: at86rf230_xmit_start - start. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:1113
                              // state change - PLL_ON
                              [ 5451.441030] at86rf230: at86rf230_async_state_change - start, state=9.
/home/pi/linux/drivers/net/ieee802154/at86rf230.c:775
                                             //get the actual value from the register
                                             [ 5451.441048] at86rf230: at86rf230 async read reg - start.
/home/pi/linux/drivers/net/ieee802154/at86rf230.c:516
                                             [ 5451.441089] at86rf230: at86rf230_async_read_reg - end.
/home/pi/linux/drivers/net/ieee802154/at86rf230.c:528
                              [ 5451.441106] at86rf230: at86rf230_async_state_change - end, state=9.
/home/pi/linux/drivers/net/ieee802154/at86rf230.c:784
               [ 5451.441120] at86rf230: at86rf230 xmit start - end. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:1135
[ 5451.441133] at86rf230: at86rf230 xmit - end. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:1164

// callback function of &ctx->msg, is called by spi async() in at86rf230 async read reg
// starts the state change
[ 5451.441275] at86rf230: at86rf230 async state change start - start. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:729
[ 5451.441305] at86rf230: at86rf230_async_state_change_start - end. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:764

// comletion function of spi async()
[ 5451.441342] at86rf230: at86rf230 async state delay - start. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:635
[ 5451.441360] at86rf230: at86rf230 async state delay - end. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:714

[ 5451.441443] at86rf230: at86rf230 async state timer - start. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:614
               [ 5451.441464] at86rf230: at86rf230 async read reg - start. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:516
               [ 5451.441494] at86rf230: at86rf230_async_read_reg - end. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:528
[ 5451.441509] at86rf230: at86rf230_async_state_timer - end. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:620
```

```
// assert the state change
[ 5451.441679] at86rf230: at86rf230 async state assert - start. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:552
[ 5451.441702] at86rf230: at86rf230 async state assert - end. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:601

// write a frame - callback of at86rf230 async state change
[ 5451.441717] at86rf230: at86rf230 write frame - start. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:1074
[ 5451.441736] at86rf230: at86rf230 write frame - end. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:1089

// at86rf230_write_frame_complete is the completion function of spi_async which is called in at86rf230_write_frame
// device is in state: PLL_ON -> rising edge ->starts frame transmission
[ 5451.441860] at86rf230: at86rf230 write frame complete - start. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:1047
               [ 5451.441883] at86rf230: at86rf230 slp tr rising edge. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:354
[ 5451.441902] at86rf230: at86rf230 write frame complete - end. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:1062



// Is not called from the at86rf230 module
[ 5451.442581] at86rf230: at86rf230_isr - start. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:1021
[ 5451.442620] at86rf230: at86rf230_isr - end. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:1034

// completion function of spi async which is called in at86rf230 isr
[ 5451.442688] at86rf230: at86rf230 irq status - start. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:1001
               [ 5451.442708] at86rf230: at86rf230 irq trx end - start. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:970
                              // state change - RX AACK ON
                              [ 5451.442725] at86rf230: at86rf230_async_state_change - start, state=22.
/home/pi/linux/drivers/net/ieee802154/at86rf230.c:775
                                                     [ 5451.442739] at86rf230: at86rf230_async_read_reg - start.
/home/pi/linux/drivers/net/ieee802154/at86rf230.c:516
                                                     [ 5451.442755] at86rf230: at86rf230_async_read_reg - end.
/home/pi/linux/drivers/net/ieee802154/at86rf230.c:528
                              [ 5451.442769] at86rf230: at86rf230 async state change - end, state=22.
/home/pi/linux/drivers/net/ieee802154/at86rf230.c:784
               [ 5451.442784] at86rf230: at86rf230 irq trx end - end. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:990
[ 5451.442796] at86rf230: at86rf230_irq_status - end. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:1011

// completion function of at86rf230_async_read_reg
[ 5451.442828] at86rf230: at86rf230_async_state_change_start - start, state=22. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:729
[ 5451.442847] at86rf230: at86rf230 async state change start - end. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:764

[ 5451.442878] at86rf230: at86rf230 async state delay - start. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:635
[ 5451.442894] at86rf230: at86rf230 async state delay - end. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:714

[ 5451.442959] at86rf230: at86rf230 async state timer - start. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:614
               [ 5451.442979] at86rf230: at86rf230_async_read_reg - start. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:516
               [ 5451.443005] at86rf230: at86rf230_async_read_reg - end. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:528
[ 5451.443021] at86rf230: at86rf230 async state timer - end. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:620

[ 5451.443079] at86rf230: at86rf230 async state assert - start. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:552
[ 5451.443098] at86rf230: at86rf230 async state assert - end. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:601

// completion function of at86rf230 irq trx end
[ 5451.443113] at86rf230: at86rf230 tx complete - start. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:833
[ 5451.443150] at86rf230: at86rf230_tx_complete - end. /home/pi/linux/drivers/net/ieee802154/at86rf230.c:839
```