Dokumentation AT86RF230.c



Inhaltsverzeichnis

[1 Wichtige Definitionen 3](#_Toc450141922)

[2 Aufgerufene Funktionen 4](#_Toc450141923)

[2.1 Die probe-Funktion 4](#_Toc450141924)

[2.2 Statuswechsel 8](#_Toc450141925)

[3 Wichtige Strukturen 11](#_Toc450141926)

[4 IEEE802154\_OPS Funktionen 16](#_Toc450141927)

[4.1 AT86RF230\_START 16](#_Toc450141928)

[4.2 AT86RF230\_SET\_PROMISCUOUS\_MODE 16](#_Toc450141929)

[4.3 AT86RF230\_SET\_HW\_ADDR\_FILT 16](#_Toc450141930)

[4.4 AT86RF230\_SET\_CSMA\_PARAMS 16](#_Toc450141931)

[4.5 AT86RF230\_XMIT 17](#_Toc450141932)

[4.6 AT86RF230\_ISR – Senden 19](#_Toc450141933)

[4.7 AT86RF230\_ISR – Empfangen 21](#_Toc450141934)

[4.8 AT86RF230\_ED 22](#_Toc450141935)

[4.9 AT86RF230\_REMOVE 22](#_Toc450141936)

[5 Anhang 23](#_Toc450141937)

[5.1 Beschreibung des Socket Buffers 24](#_Toc450141938)

[5.2 Ausgaben bei einem Sendevorgang 25](#_Toc450141939)

# 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**](http://lxr.free-electrons.com/source/include/net/mac802154.h#L69) ***/\*\****

**[70](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L70)  *\* struct ieee802154\_hw - ieee802154 hardware***

**[71](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L71)  *\****

**[72](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L72)  *\* @extra\_tx\_headroom: headroom to reserve in each transmit skb for use by the***

**[73](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L73)  *\* driver (e.g. for transmit headers.)***

**[74](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L74)  *\****

**[75](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L75)  *\* @flags: hardware flags, see &enum ieee802154\_hw\_flags***

**[76](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L76)  *\****

**[77](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L77)  *\* @parent: parent device of the hardware.***

**[78](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L78)  *\****

**[79](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L79)  *\* @priv: pointer to private area that was allocated for driver use along with***

**[80](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L80)  *\* this structure.***

**[81](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L81)  *\****

**[82](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L82)  *\* @phy: This points to the &struct wpan\_phy allocated for this 802.15.4 PHY.***

**[83](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L83)  *\*/***

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.

# Aufgerufene Funktionen

## 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]](#footnote-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.

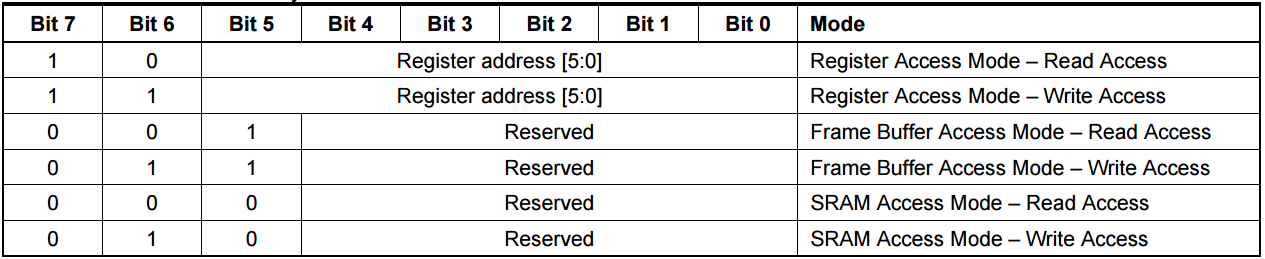


Abbildung 1: Zugriff auf das Lesen und Schreiben

Zur Initialisierung der SPI Messages wird die Funktion

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. Hier wird *lp*, also die Driver spezifischen Daten, in *spi->driver\_data* abgespeichert.

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](http://lxr.free-electrons.com/source/kernel/sched/completion.c" \l "L131)**  *\* This waits for either a completion of a specific task to be signaled or for a*

**[132](http://lxr.free-electrons.com/source/kernel/sched/completion.c" \l "L132)**  *\* specified timeout to expire. The timeout is in jiffies. It is not*

**[133](http://lxr.free-electrons.com/source/kernel/sched/completion.c" \l "L133)**  *\* 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 Abbildung 2 zeigt, in welche Register geschrieben wird und welche Daten dort abgelegt werden. Außerdem folgt danach eine detailliertere Beschreibung der unterschiedlichen Funktionen.

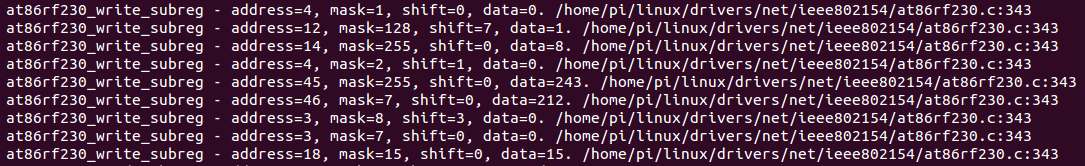


Abbildung 2: Ausgabe der beschriebenen Register

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**](http://lxr.free-electrons.com/source/kernel/irq/devres.c#L28) */\*\**

**[29](http://lxr.free-electrons.com/source/kernel/irq/devres.c" \l "L29)**  *\* devm\_request\_threaded\_irq - allocate an interrupt line for a managed device*

**[30](http://lxr.free-electrons.com/source/kernel/irq/devres.c" \l "L30)**  *\* @dev: device to request interrupt for*

**[31](http://lxr.free-electrons.com/source/kernel/irq/devres.c" \l "L31)**  *\* @irq: Interrupt line to allocate*

**[32](http://lxr.free-electrons.com/source/kernel/irq/devres.c" \l "L32)**  *\* @handler: Function to be called when the IRQ occurs*

**[33](http://lxr.free-electrons.com/source/kernel/irq/devres.c" \l "L33)**  *\* @thread\_fn: function to be called in a threaded interrupt context. NULL*

**[34](http://lxr.free-electrons.com/source/kernel/irq/devres.c" \l "L34)**  *\* for devices which handle everything in @handler*

**[35](http://lxr.free-electrons.com/source/kernel/irq/devres.c" \l "L35)**  *\* @irqflags: Interrupt type flags*

**[36](http://lxr.free-electrons.com/source/kernel/irq/devres.c" \l "L36)**  *\* @devname: An ascii name for the claiming device*

**[37](http://lxr.free-electrons.com/source/kernel/irq/devres.c" \l "L37)**  *\* @dev\_id: A cookie passed back to the handler function*

**[38](http://lxr.free-electrons.com/source/kernel/irq/devres.c" \l "L38)**  *\**

**[39](http://lxr.free-electrons.com/source/kernel/irq/devres.c" \l "L39)**  *\* Except for the extra @dev argument, this function takes the*

**[40](http://lxr.free-electrons.com/source/kernel/irq/devres.c" \l "L40)**  *\* same arguments and performs the same function as*

**[41](http://lxr.free-electrons.com/source/kernel/irq/devres.c" \l "L41)**  *\* request\_threaded\_irq(). IRQs requested with this function will be*

**[42](http://lxr.free-electrons.com/source/kernel/irq/devres.c" \l "L42)**  *\* automatically freed on driver detach.*

**[43](http://lxr.free-electrons.com/source/kernel/irq/devres.c" \l "L43)**  *\**

**[44](http://lxr.free-electrons.com/source/kernel/irq/devres.c" \l "L44)**  *\* If an IRQ allocated with this function needs to be freed*

**[45](http://lxr.free-electrons.com/source/kernel/irq/devres.c" \l "L45)**  *\* separately, devm\_free\_irq() must be used.*

**[46](http://lxr.free-electrons.com/source/kernel/irq/devres.c" \l "L46)**  *\*/*

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](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L317)** */\*\**

**[318](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L318)**  *\* ieee802154\_register\_hw - Register hardware device*

**[319](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L319)**  *\**

**[320](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L320)**  *\* You must call this function before any other functions in*

**[321](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L321)**  *\* mac802154. Note that before a hardware can be registered, you*

**[322](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L322)**  *\* need to fill the contained wpan\_phy's information.*

**[323](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L323)**  *\**

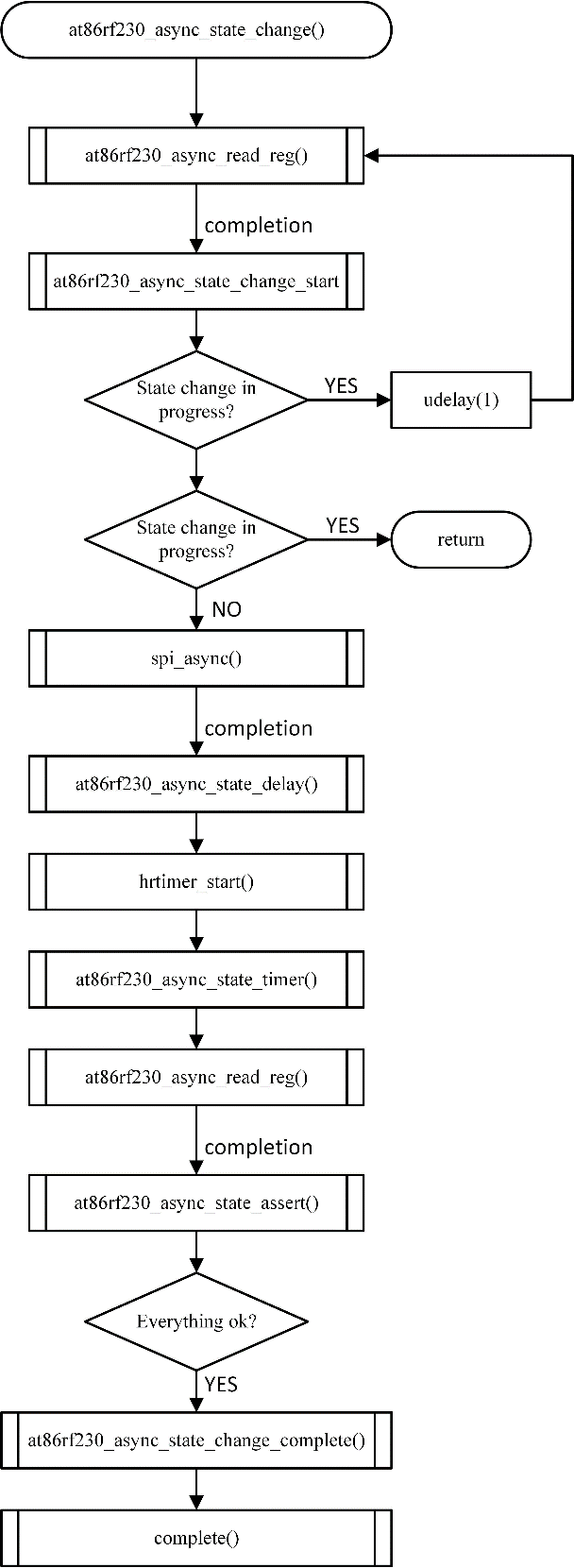
**[324](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L324)**  *\* @hw: the device to register as returned by ieee802154\_alloc\_hw()*

**[325](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L325)**  *\**

**[326](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L326)**  *\* Return: 0 on success. An error code otherwise.*

**[327](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L327)**  *\*/*

## 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 Abbildung 4 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 *spi\_async()* durchgeführt werden. In diesem Fall wird das Register nicht gelesen sondern via SPI beschrieben. Die Kommunikationsrichtung wird über führende Bits markiert.

Abbildung 3: Ablauf eines Statuswechsels

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

[**2253**](http://lxr.free-electrons.com/source/drivers/spi/spi.c#L2254) \* This call may be used in\_irq and other contexts which can't sleep,

**[2254](http://lxr.free-electrons.com/source/drivers/spi/spi.c" \l "L2254)** \* as well as from task contexts which can sleep.

**[2255](http://lxr.free-electrons.com/source/drivers/spi/spi.c" \l "L2255)** \*

**[2256](http://lxr.free-electrons.com/source/drivers/spi/spi.c" \l "L2256)** \* The completion callback is invoked in a context which can't sleep.

**[2257](http://lxr.free-electrons.com/source/drivers/spi/spi.c" \l "L2257)** \* Before that invocation, the value of message->status is undefined.

**[2258](http://lxr.free-electrons.com/source/drivers/spi/spi.c" \l "L2258)** \* When the callback is issued, message->status holds either zero (to

**[2259](http://lxr.free-electrons.com/source/drivers/spi/spi.c" \l "L2259)** \* indicate complete success) or a negative error code. After that

**[2260](http://lxr.free-electrons.com/source/drivers/spi/spi.c" \l "L2260)** \* callback returns, the driver which issued the transfer request may

**[2261](http://lxr.free-electrons.com/source/drivers/spi/spi.c" \l "L2261)** \* deallocate the associated memory; it's no longer in use by any SPI

**[2262](http://lxr.free-electrons.com/source/drivers/spi/spi.c" \l "L2262)** \* core or controller driver code.

**[2263](http://lxr.free-electrons.com/source/drivers/spi/spi.c" \l "L2263)** \*

**[2264](http://lxr.free-electrons.com/source/drivers/spi/spi.c" \l "L2264)** \* Note that although all messages to a spi\_device are handled in

**[2265](http://lxr.free-electrons.com/source/drivers/spi/spi.c" \l "L2265)** \* FIFO order, messages may go to different devices in other orders.

**[2266](http://lxr.free-electrons.com/source/drivers/spi/spi.c" \l "L2266)** \* Some device might be higher priority, or have various "hard" access

**[2267](http://lxr.free-electrons.com/source/drivers/spi/spi.c" \l "L2267)** \* time requirements, for example.

**[2268](http://lxr.free-electrons.com/source/drivers/spi/spi.c" \l "L2268)** \*

**[2269](http://lxr.free-electrons.com/source/drivers/spi/spi.c" \l "L2269)** \* On detection of any fault during the transfer, processing of

**[2270](http://lxr.free-electrons.com/source/drivers/spi/spi.c" \l "L2270)** \* the entire message is aborted, and the device is deselected.

**[2271](http://lxr.free-electrons.com/source/drivers/spi/spi.c" \l "L2271)** \* Until returning from the associated message completion callback,

**[2272](http://lxr.free-electrons.com/source/drivers/spi/spi.c" \l "L2272)** \* no other spi\_message queued to that device will be processed.

**[2273](http://lxr.free-electrons.com/source/drivers/spi/spi.c" \l "L2273)** \* (This rule applies equally to all the synchronous transfer calls,

**[2274](http://lxr.free-electrons.com/source/drivers/spi/spi.c" \l "L2274)** \* which are wrappers around this core asynchronous primitive.)

**[2275](http://lxr.free-electrons.com/source/drivers/spi/spi.c" \l "L2275)** \*

**[2276](http://lxr.free-electrons.com/source/drivers/spi/spi.c" \l "L2276)** \* Return: zero on success, else a negative error code.

**[2277](http://lxr.free-electrons.com/source/drivers/spi/spi.c" \l "L2277)** \*/

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.

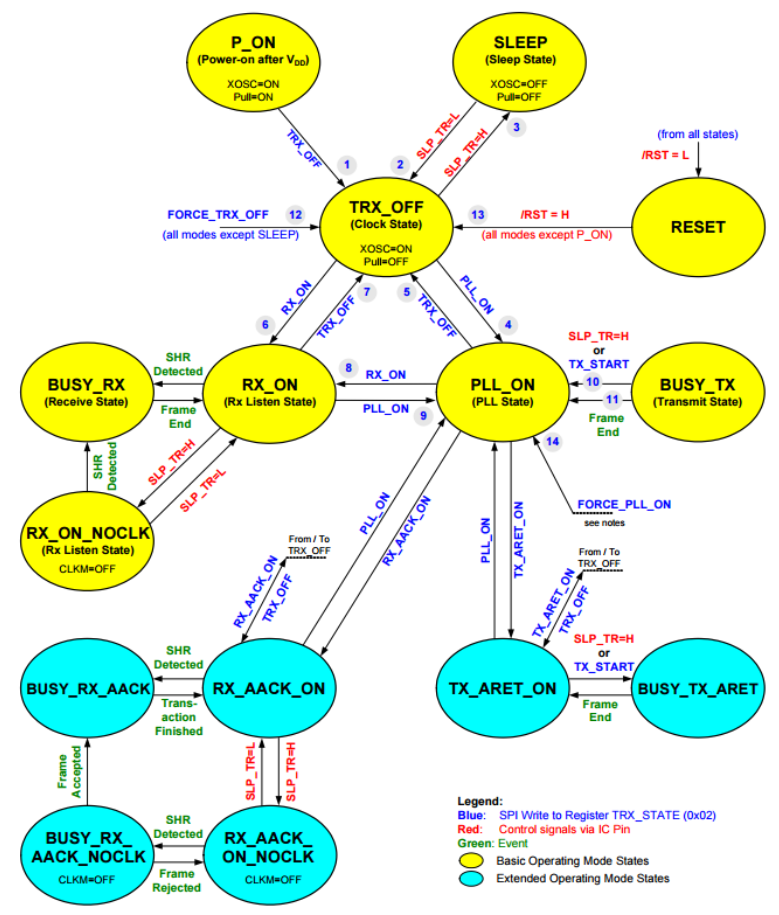


Abbildung 4: Extended Operating Mode

# 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](http://lxr.free-electrons.com/source/include/linux/regmap.h" \l "L113)** */\*\**

**[114](http://lxr.free-electrons.com/source/include/linux/regmap.h" \l "L114)**  *\* Configuration for the register map of a device.*

**[115](http://lxr.free-electrons.com/source/include/linux/regmap.h" \l "L115)**  *\**

**[116](http://lxr.free-electrons.com/source/include/linux/regmap.h" \l "L116)**  *\* @name: Optional name of the regmap. Useful when a device has multiple*

**[117](http://lxr.free-electrons.com/source/include/linux/regmap.h" \l "L117)**  *\* register regions.*

**[118](http://lxr.free-electrons.com/source/include/linux/regmap.h" \l "L118)**  *\**

**[119](http://lxr.free-electrons.com/source/include/linux/regmap.h" \l "L119)**  *\* @reg\_bits: Number of bits in a register address, mandatory.*

**[120](http://lxr.free-electrons.com/source/include/linux/regmap.h" \l "L120)**  *\* @reg\_stride: The register address stride. Valid register addresses are a*

**[121](http://lxr.free-electrons.com/source/include/linux/regmap.h" \l "L121)**  *\* multiple of this value. If set to 0, a value of 1 will be*

**[122](http://lxr.free-electrons.com/source/include/linux/regmap.h" \l "L122)**  *\* used.*

**[123](http://lxr.free-electrons.com/source/include/linux/regmap.h" \l "L123)**  *\* @pad\_bits: Number of bits of padding between register and value.*

**[124](http://lxr.free-electrons.com/source/include/linux/regmap.h" \l "L124)**  *\* @val\_bits: Number of bits in a register value, mandatory.*

**[125](http://lxr.free-electrons.com/source/include/linux/regmap.h" \l "L125)**  *\**

**[126](http://lxr.free-electrons.com/source/include/linux/regmap.h" \l "L126)**  *\* @writeable\_reg: Optional callback returning true if the register*

**[127](http://lxr.free-electrons.com/source/include/linux/regmap.h" \l "L127)**  *\* can be written to. If this field is NULL but wr\_table*

**[128](http://lxr.free-electrons.com/source/include/linux/regmap.h" \l "L128)**  *\* (see below) is not, the check is performed on such table*

**[129](http://lxr.free-electrons.com/source/include/linux/regmap.h" \l "L129)**  *\* (a register is writeable if it belongs to one of the ranges*

**[130](http://lxr.free-electrons.com/source/include/linux/regmap.h" \l "L130)**  *\* specified by wr\_table).*

[**131**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L131)  *\* @readable\_reg: Optional callback returning true if the register*

[**132**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L132)  *\* can be read from. If this field is NULL but rd\_table*

[**133**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L133)  *\* (see below) is not, the check is performed on such table*

**[134](http://lxr.free-electrons.com/source/include/linux/regmap.h" \l "L134)**  *\* (a register is readable if it belongs to one of the ranges*

**[135](http://lxr.free-electrons.com/source/include/linux/regmap.h" \l "L135)**  *\* specified by rd\_table).*

**[136](http://lxr.free-electrons.com/source/include/linux/regmap.h" \l "L136)**  *\* @volatile\_reg: Optional callback returning true if the register*

**[137](http://lxr.free-electrons.com/source/include/linux/regmap.h" \l "L137)**  *\* value can't be cached. If this field is NULL but*

**[138](http://lxr.free-electrons.com/source/include/linux/regmap.h" \l "L138)**  *\* volatile\_table (see below) is not, the check is performed on*

**[139](http://lxr.free-electrons.com/source/include/linux/regmap.h" \l "L139)**  *\* such table (a register is volatile if it belongs to one of*

**[140](http://lxr.free-electrons.com/source/include/linux/regmap.h" \l "L140)**  *\* the ranges specified by volatile\_table).*

**[141](http://lxr.free-electrons.com/source/include/linux/regmap.h" \l "L141)**  *\* @precious\_reg: Optional callback returning true if the register*

[**142**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L142)  *\* should not be read outside of a call from the driver*

[**143**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L143)  *\* (e.g., a clear on read interrupt status register). If this*

[**144**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L144)  *\* field is NULL but precious\_table (see below) is not, the*

[**145**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L145)  *\* check is performed on such table (a register is precious if*

[**146**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L146)  *\* it belongs to one of the ranges specified by precious\_table).*

[**147**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L147)  *\* @lock: Optional lock callback (overrides regmap's default lock*

[**148**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L148)  *\* function, based on spinlock or mutex).*

[**149**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L149)  *\* @unlock: As above for unlocking.*

[**150**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L150)  *\* @lock\_arg: this field is passed as the only argument of lock/unlock*

[**151**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L151)  *\* functions (ignored in case regular lock/unlock functions*

[**152**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L152)  *\* are not overridden).*

[**153**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L153)  *\* @reg\_read: Optional callback that if filled will be used to perform*

[**154**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L154)  *\* all the reads from the registers. Should only be provided for*

[**155**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L155)  *\* devices whose read operation cannot be represented as a simple*

[**156**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L156)  *\* read operation on a bus such as SPI, I2C, etc. Most of the*

[**157**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L157)  *\* devices do not need this.*

[**158**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L158)  *\* @reg\_write: Same as above for writing.*

[**159**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L159)  *\* @fast\_io: Register IO is fast. Use a spinlock instead of a mutex*

[**160**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L160)  *\* to perform locking. This field is ignored if custom lock/unlock*

[**161**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L161)  *\* functions are used (see fields lock/unlock of struct regmap\_config).*

[**162**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L162)  *\* This field is a duplicate of a similar file in*

[**163**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L163)  *\* 'struct regmap\_bus' and serves exact same purpose.*

[**164**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L164)  *\* Use it only for "no-bus" cases.*

[**165**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L165)  *\* @max\_register: Optional, specifies the maximum valid register index.*

[**166**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L166)  *\* @wr\_table: Optional, points to a struct regmap\_access\_table specifying*

[**167**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L167)  *\* valid ranges for write access.*

[**168**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L168)  *\* @rd\_table: As above, for read access.*

[**169**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L169)  *\* @volatile\_table: As above, for volatile registers.*

[**170**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L170)  *\* @precious\_table: As above, for precious registers.*

[**171**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L171)  *\* @reg\_defaults: Power on reset values for registers (for use with*

[**172**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L172)  *\* register cache support).*

[**173**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L173)  *\* @num\_reg\_defaults: Number of elements in reg\_defaults.*

[**174**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L174)  *\**

[**175**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L175)  *\* @read\_flag\_mask: Mask to be set in the top byte of the register when doing*

[**176**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L176)  *\* a read.*

[**177**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L177)  *\* @write\_flag\_mask: Mask to be set in the top byte of the register when doing*

[**178**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L178)  *\* a write. If both read\_flag\_mask and write\_flag\_mask are*

[**179**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L179)  *\* empty the regmap\_bus default masks are used.*

[**180**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L180)  *\* @use\_single\_rw: If set, converts the bulk read and write operations into*

[**181**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L181)  *\* a series of single read and write operations. This is useful*

[**182**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L182)  *\* for device that does not support bulk read and write.*

[**183**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L183)  *\* @can\_multi\_write: If set, the device supports the multi write mode of bulk*

[**184**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L184)  *\* write operations, if clear multi write requests will be*

[**185**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L185)  *\* split into individual write operations*

[**186**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L186)  *\**

[**187**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L187)  *\* @cache\_type: The actual cache type.*

[**188**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L188)  *\* @reg\_defaults\_raw: Power on reset values for registers (for use with*

[**189**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L189)  *\* register cache support).*

[**190**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L190)  *\* @num\_reg\_defaults\_raw: Number of elements in reg\_defaults\_raw.*

[**191**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L191)  *\* @reg\_format\_endian: Endianness for formatted register addresses. If this is*

[**192**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L192)  *\* DEFAULT, the @reg\_format\_endian\_default value from the*

[**193**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L193)  *\* regmap bus is used.*

[**194**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L194)  *\* @val\_format\_endian: Endianness for formatted register values. If this is*

[**195**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L195)  *\* DEFAULT, the @reg\_format\_endian\_default value from the*

[**196**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L196)  *\* regmap bus is used.*

[**197**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L197)  *\**

[**198**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L198)  *\* @ranges: Array of configuration entries for virtual address ranges.*

[**199**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L199)  *\* @num\_ranges: Number of range configuration entries.*

[**200**](http://lxr.free-electrons.com/source/include/linux/regmap.h#L200)  *\*/*

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](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L142)** */\* struct ieee802154\_ops - callbacks from mac802154 to the driver*

**[143](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L143)**  *\**

**[144](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L144)**  *\* This structure contains various callbacks that the driver may*

**[145](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L145)**  *\* handle or, in some cases, must handle, for example to transmit*

**[146](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L146)**  *\* a frame.*

**[147](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L147)**  *\**

**[148](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L148)**  *\* start: Handler that 802.15.4 module calls for device initialization.*

**[149](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L149)**  *\* This function is called before the first interface is attached.*

**[150](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L150)**  *\**

**[151](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L151)**  *\* stop: Handler that 802.15.4 module calls for device cleanup.*

**[152](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L152)**  *\* This function is called after the last interface is removed.*

**[153](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L153)**  *\**

**[154](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L154)**  *\* xmit\_sync:*

**[155](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L155)**  *\* Handler that 802.15.4 module calls for each transmitted frame.*

**[156](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L156)**  *\* skb cntains the buffer starting from the IEEE 802.15.4 header.*

**[157](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L157)**  *\* The low-level driver should send the frame based on available*

**[158](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L158)**  *\* configuration. This is called by a workqueue and useful for*

**[159](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L159)**  *\* synchronous 802.15.4 drivers.*

**[160](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L160)**  *\* This function should return zero or negative errno.*

**[161](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L161)**  *\**

**[162](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L162)**  *\* WARNING:*

**[163](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L163)**  *\* This will be deprecated soon. We don't accept synced xmit callbacks*

**[164](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L164)**  *\* drivers anymore.*

**[165](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L165)**  *\**

**[166](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L166)**  *\* xmit\_async:*

**[167](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L167)**  *\* Handler that 802.15.4 module calls for each transmitted frame.*

**[168](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L168)**  *\* skb cntains the buffer starting from the IEEE 802.15.4 header.*

**[169](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L169)**  *\* The low-level driver should send the frame based on available*

**[170](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L170)**  *\* configuration.*

**[171](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L171)**  *\* This function should return zero or negative errno.*

**[172](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L172)**  *\**

**[173](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L173)**  *\* ed: Handler that 802.15.4 module calls for Energy Detection.*

**[174](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L174)**  *\* This function should place the value for detected energy*

**[175](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L175)**  *\* (usually device-dependant) in the level pointer and return*

**[176](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L176)**  *\* either zero or negative errno. Called with pib\_lock held.*

**[177](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L177)**  *\**

**[178](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L178)**  *\* set\_channel:*

**[179](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L179)**  *\* Set radio for listening on specific channel.*

**[180](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L180)**  *\* Set the device for listening on specified channel.*

**[181](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L181)**  *\* Returns either zero, or negative errno. Called with pib\_lock held.*

**[182](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L182)**  *\**

**[183](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L183)**  *\* set\_hw\_addr\_filt:*

**[184](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L184)**  *\* Set radio for listening on specific address.*

**[185](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L185)**  *\* Set the device for listening on specified address.*

**[186](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L186)**  *\* Returns either zero, or negative errno.*

**[187](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L187)**  *\**

**[188](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L188)**  *\* set\_txpower:*

**[189](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L189)**  *\* Set radio transmit power in mBm. Called with pib\_lock held.*

**[190](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L190)**  *\* Returns either zero, or negative errno.*

**[191](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L191)**  *\**

**[192](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L192)**  *\* set\_lbt*

**[193](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L193)**  *\* Enables or disables listen before talk on the device. Called with*

**[194](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L194)**  *\* pib\_lock held.*

**[195](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L195)**  *\* Returns either zero, or negative errno.*

**[196](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L196)**  *\**

**[197](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L197)**  *\* set\_cca\_mode*

**[198](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L198)**  *\* Sets the CCA mode used by the device. Called with pib\_lock held.*

**[199](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L199)**  *\* Returns either zero, or negative errno.*

**[200](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L200)**  *\**

**[201](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L201)**  *\* set\_cca\_ed\_level*

**[202](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L202)**  *\* Sets the CCA energy detection threshold in mBm. Called with pib\_lock*

**[203](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L203)**  *\* held.*

**[204](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L204)**  *\* Returns either zero, or negative errno.*

**[205](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L205)**  *\**

**[206](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L206)**  *\* set\_csma\_params*

**[207](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L207)**  *\* Sets the CSMA parameter set for the PHY. Called with pib\_lock held.*

**[208](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L208)**  *\* Returns either zero, or negative errno.*

**[209](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L209)**  *\**

**[210](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L210)**  *\* set\_frame\_retries*

**[211](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L211)**  *\* Sets the retransmission attempt limit. Called with pib\_lock held.*

**[212](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L212)**  *\* Returns either zero, or negative errno.*

**[213](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L213)**  *\**

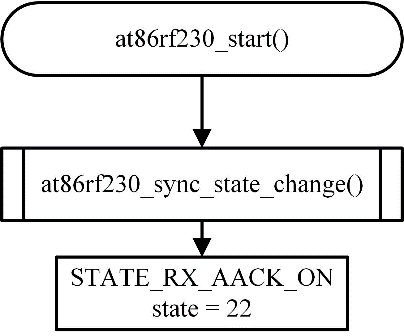
**[214](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L214)**  *\* set\_promiscuous\_mode*

**[215](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L215)**  *\* Enables or disable promiscuous mode.*

**[216](http://lxr.free-electrons.com/source/include/net/mac802154.h" \l "L216)**  *\*/*

# IEEE802154\_OPS Funktionen

## 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.

## 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.

## 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].

## 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.

## 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)

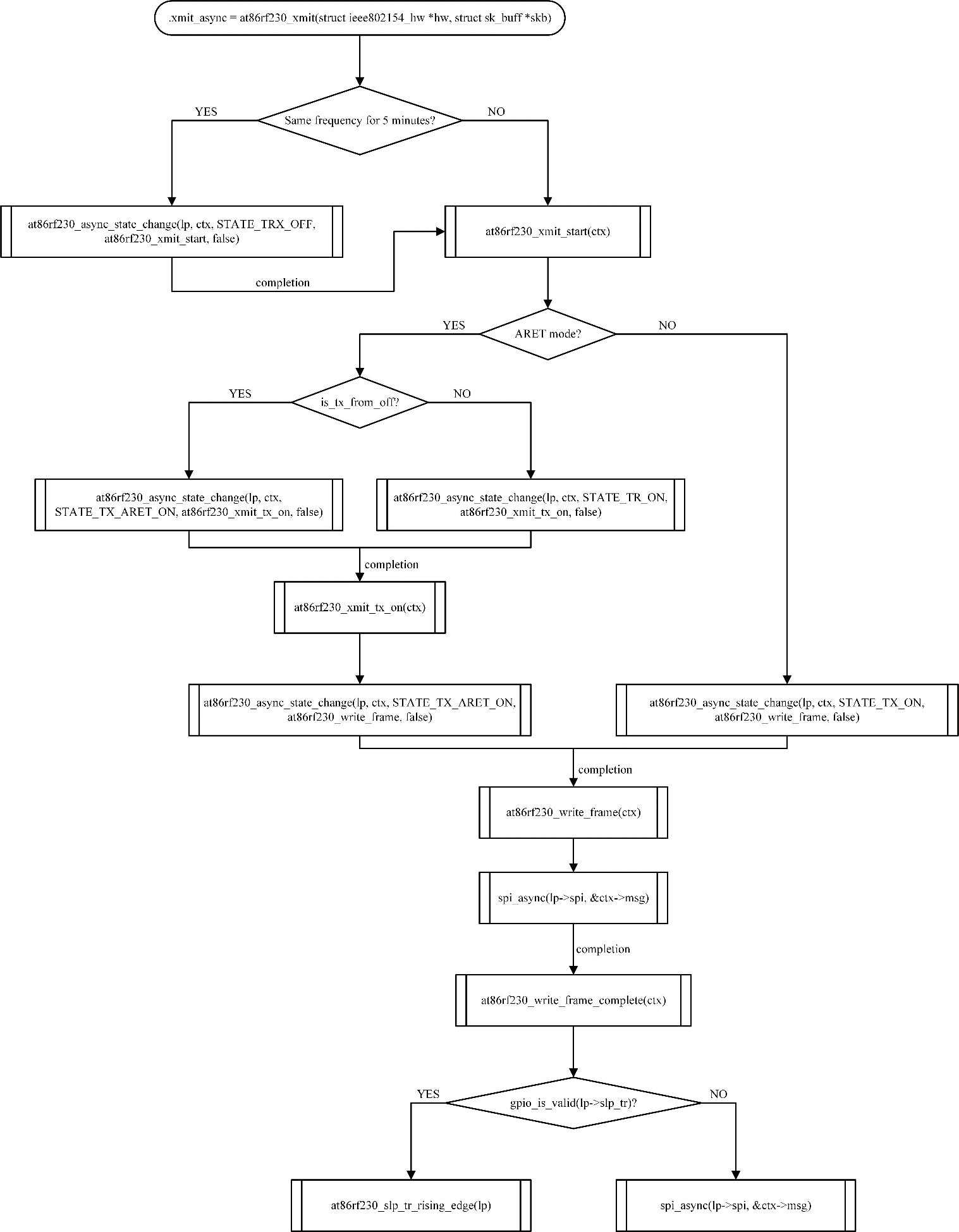
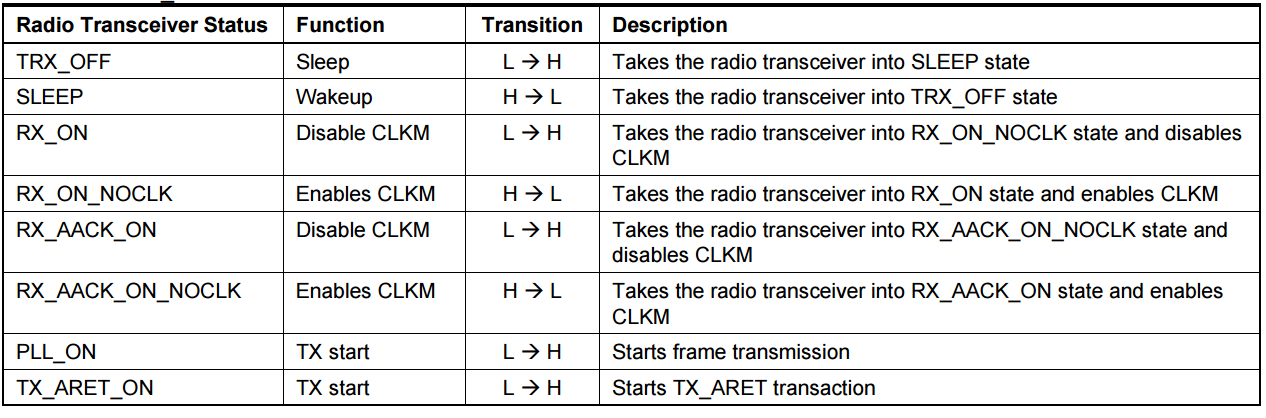


Abbildung : Flow-Chart für den Ablauf der at86rf230\_xmit Funktion

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.

## AT86RF230\_ISR – Senden

Die Funktion

at86rf230\_isr()

wird nach jedem Aufruf von *at86rf230\_xmit()* durch ein IRQ 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.

Tabelle 2: Auswirkungen von Pegeländerungen am Pin SLP\_TR

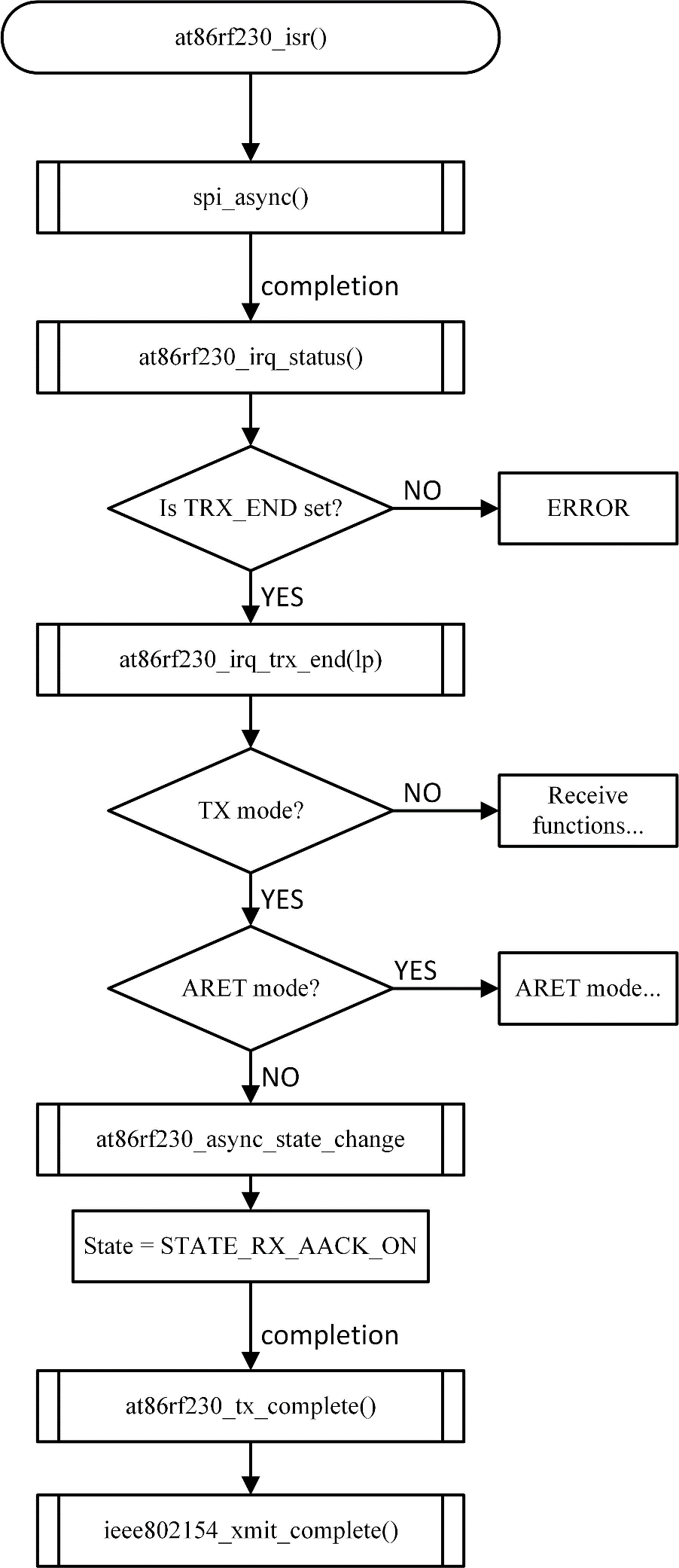
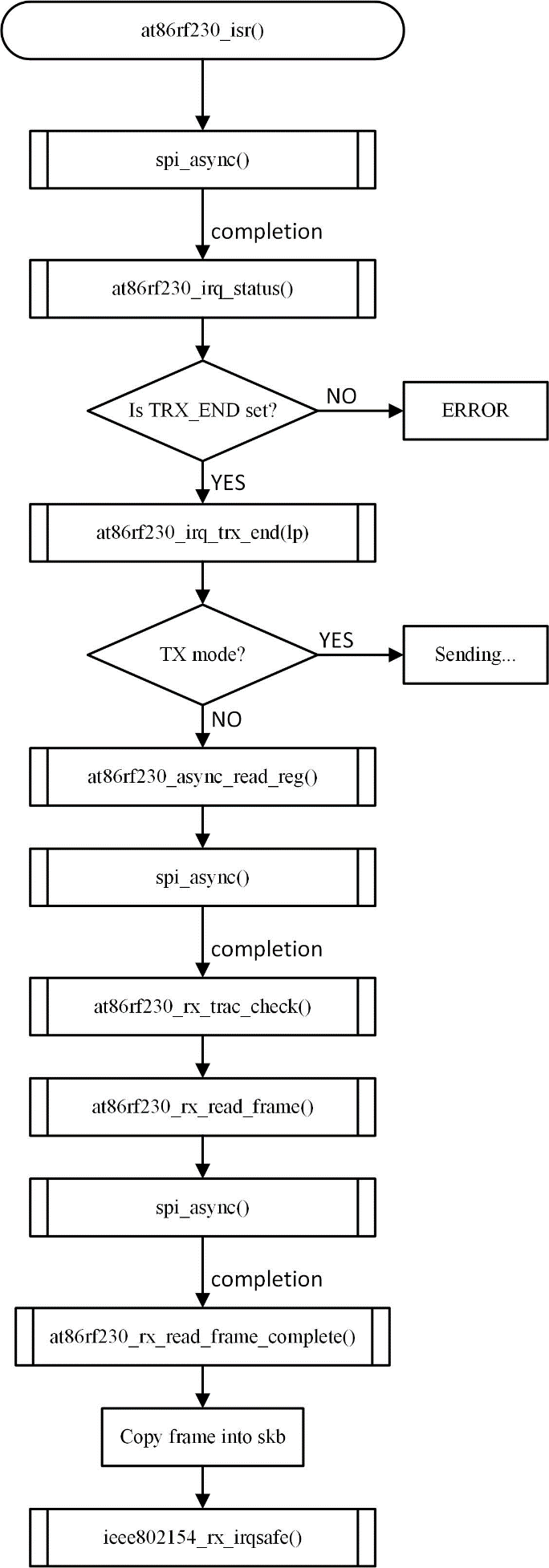


Abbildung : Ablauf nach dem Aufruf der Funktion at86rf230\_isr

## AT86RF230\_ISR – Empfangen



Soll ein Frame empfangen werden, sieht der Ablauf dieser Routine zu Beginn genauso aus, wie der Ablauf unter Kapitel 4.6 erläutert wurde. Erst wenn die Abfolge zu der Abfrage kommt, ob sich der Chip im Sendemodus befindet, unterscheidet sich der weitere Verlauf. Anstelle eines Statuswechsels wird die Funktion

at86rf230\_async\_read\_reg

aufgerufen. Hier wird mit Hilfe von *spi\_async()* das Register *RG\_TRX\_STATE* ausgelesen. Als completion-Funktion wird anschließend die Funktion

at86rf230\_rx\_trac\_check()

ausgeführt. Momentan sind in dieser Funktion keine weiteren Aufgaben implementiert, außer der Aufruf von

at86rf230\_rx\_read\_frame().

Es könnte noch eine Überprüfung mittels dem vorher ausgelesenem Register vorgenommen werden. Dieses ist allerdings momentan laut dem Datenblatt nicht erforderlich.

Die Funktion *at86rf230\_rx\_read\_frame()* ist nun dafür zuständig das empfangene Frame mittels *spi\_async()* auszulesen. Dafür wird buf[0] der entsprechende Befehl für das Lesen eines Frame-Buffers übergeben. Als complition-Funktion erhält das asynchrone Auslesen die Funktion

at86rf230\_rx\_read\_frame\_complete().

Dieser Aufruf ist notwendig, um das ausgelesene Frame in ein Struct vom Typ *sk\_buff* abzulegen. Dafür werden mittels *memcpy()* aus *buf[]* nur die gewollten Bytes ausgelesen und damit die ersten beiden und die Abschlussbytes vernachlässigt.

Als Abschluss des Empfangens wird die Funktion

ieee802154\_rx\_irqsafe()

aus den Top-Layern aufgerufen. Diese dient dazu das Frame auch im IEEE-Layer zu empfangen und dort weiter verarbeitet zu werden.

## AT86RF230\_ED

Die Funktion

at86rf230\_ed

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

## AT86RF230\_REMOVE

Wenn das Modul mit *sudo rmmod at86rf230* entfernt wird, wird die Funktion

at86rf230\_remove()

aufgerufen. Damit alle noch aktiven Interrupts deaktiviert werden, werden in dem Register IRQ\_MASK (0x0E) alle Bits auf logisch 0 gesetzt. Im Anschluss werden die beiden Funktionen

ieee802154\_unregister\_hw()

und

ieee802154\_free\_hw()

aufgerufen und die Hardware freizugeben. Außerdem wird in diesem Ablauf doch die Funktion *.stop* aus dem Struct *ieee802154\_ops* aufgerufen. Dies ist in diesem Fall die Funktion

at86rf230\_stop().

Diese hat nur noch eine Aufgabe, welche das Veranlassen eines Statuswechsels in den State *STATE\_FORCE\_TRX\_OFF* ist. Damit ist das Entfernen des Moduls abgeschlossen.

# Anhang

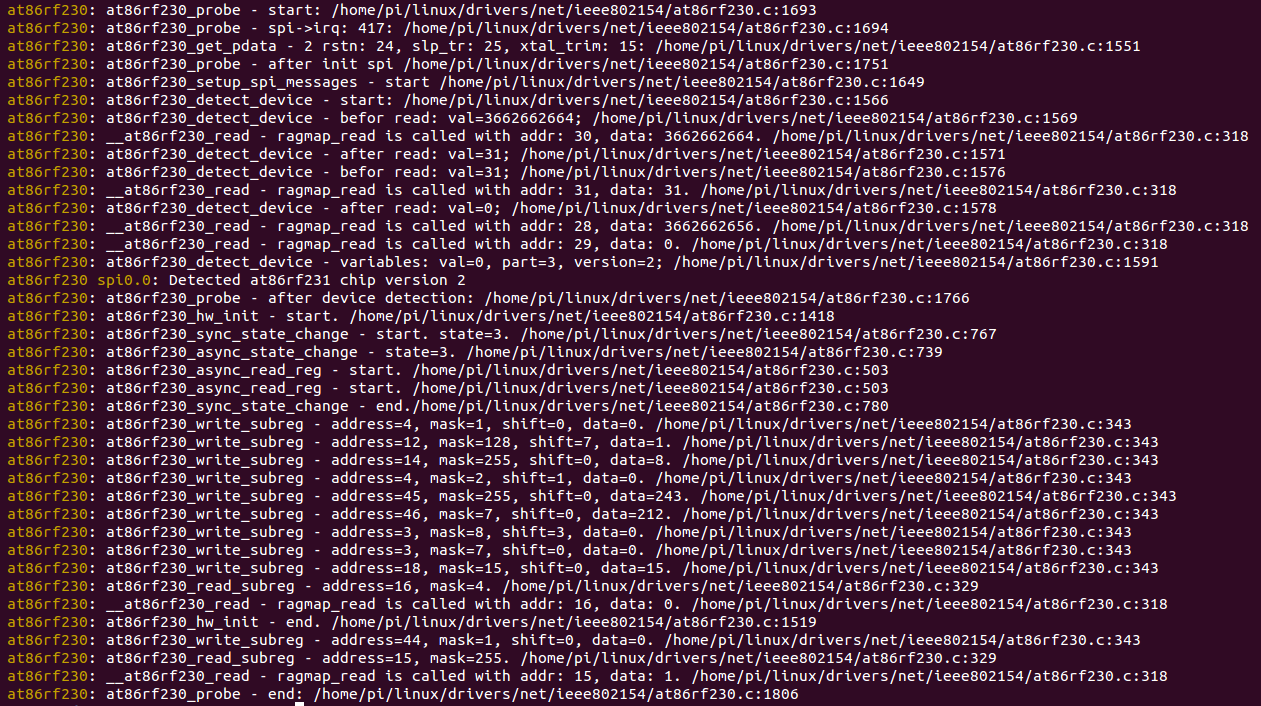


Abbildung 7: Ausgabe bei der Einbindung von AT86RF230.ko

## Beschreibung des Socket Buffers

**[477](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L477)**  *\* struct sk\_buff - socket buffer*

**[478](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L478)**  *\* @next: Next buffer in list*

**[479](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L479)**  *\* @prev: Previous buffer in list*

**[480](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L480)**  *\* @tstamp: Time we arrived/left*

**[481](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L481)**  *\* @rbnode: RB tree node, alternative to next/prev for netem/tcp*

**[482](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L482)**  *\* @sk: Socket we are owned by*

**[483](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L483)**  *\* @dev: Device we arrived on/are leaving by*

**[484](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L484)**  *\* @cb: Control buffer. Free for use by every layer. Put private vars here*

**[485](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L485)**  *\* @\_skb\_refdst: destination entry (with norefcount bit)*

**[486](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L486)**  *\* @sp: the security path, used for xfrm*

**[487](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L487)**  *\* @len: Length of actual data*

**[488](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L488)**  *\* @data\_len: Data length*

**[489](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L489)**  *\* @mac\_len: Length of link layer header*

**[490](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L490)**  *\* @hdr\_len: writable header length of cloned skb*

**[491](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L491)**  *\* @csum: Checksum (must include start/offset pair)*

**[492](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L492)**  *\* @csum\_start: Offset from skb->head where checksumming should start*

**[493](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L493)**  *\* @csum\_offset: Offset from csum\_start where checksum should be stored*

**[494](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L494)**  *\* @priority: Packet queueing priority*

**[495](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L495)**  *\* @ignore\_df: allow local fragmentation*

**[496](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L496)**  *\* @cloned: Head may be cloned (check refcnt to be sure)*

**[497](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L497)**  *\* @ip\_summed: Driver fed us an IP checksum*

**[498](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L498)**  *\* @nohdr: Payload reference only, must not modify header*

**[499](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L499)**  *\* @nfctinfo: Relationship of this skb to the connection*

**[500](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L500)**  *\* @pkt\_type: Packet class*

**[501](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L501)**  *\* @fclone: skbuff clone status*

**[502](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L502)**  *\* @ipvs\_property: skbuff is owned by ipvs*

**[503](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L503)**  *\* @peeked: this packet has been seen already, so stats have been*

**[504](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L504)**  *\* done for it, don't do them again*

**[505](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L505)**  *\* @nf\_trace: netfilter packet trace flag*

**[506](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L506)**  *\* @protocol: Packet protocol from driver*

**[507](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L507)**  *\* @destructor: Destruct function*

**[508](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L508)**  *\* @nfct: Associated connection, if any*

**[509](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L509)**  *\* @nf\_bridge: Saved data about a bridged frame - see br\_netfilter.c*

**[510](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L510)**  *\* @skb\_iif: ifindex of device we arrived on*

**[511](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L511)**  *\* @tc\_index: Traffic control index*

**[512](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L512)**  *\* @tc\_verd: traffic control verdict*

**[513](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L513)**  *\* @hash: the packet hash*

**[514](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L514)**  *\* @queue\_mapping: Queue mapping for multiqueue devices*

**[515](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L515)**  *\* @xmit\_more: More SKBs are pending for this queue*

**[516](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L516)**  *\* @ndisc\_nodetype: router type (from link layer)*

**[517](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L517)**  *\* @ooo\_okay: allow the mapping of a socket to a queue to be changed*

**[518](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L518)**  *\* @l4\_hash: indicate hash is a canonical 4-tuple hash over transport*

**[519](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L519)**  *\* ports.*

**[520](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L520)**  *\* @sw\_hash: indicates hash was computed in software stack*

**[521](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L521)**  *\* @wifi\_acked\_valid: wifi\_acked was set*

**[522](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L522)**  *\* @wifi\_acked: whether frame was acked on wifi or not*

**[523](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L523)**  *\* @no\_fcs: Request NIC to treat last 4 bytes as Ethernet FCS*

**[524](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L524)**  *\* @napi\_id: id of the NAPI struct this skb came from*

**[525](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L525)**  *\* @secmark: security marking*

**[526](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L526)**  *\* @offload\_fwd\_mark: fwding offload mark*

**[527](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L527)**  *\* @mark: Generic packet mark*

**[528](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L528)**  *\* @vlan\_proto: vlan encapsulation protocol*

**[529](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L529)**  *\* @vlan\_tci: vlan tag control information*

**[530](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L530)**  *\* @inner\_protocol: Protocol (encapsulation)*

**[531](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L531)**  *\* @inner\_transport\_header: Inner transport layer header (encapsulation)*

**[532](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L532)**  *\* @inner\_network\_header: Network layer header (encapsulation)*

**[533](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L533)**  *\* @inner\_mac\_header: Link layer header (encapsulation)*

**[534](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L534)**  *\* @transport\_header: Transport layer header*

**[535](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L535)**  *\* @network\_header: Network layer header*

**[536](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L536)**  *\* @mac\_header: Link layer header*

**[537](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L537)**  *\* @tail: Tail pointer*

**[538](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L538)**  *\* @end: End pointer*

**[539](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L539)**  *\* @head: Head of buffer*

**[540](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L540)**  *\* @data: Data head pointer*

**[541](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L541)**  *\* @truesize: Buffer size*

**[542](http://lxr.free-electrons.com/source/include/linux/skbuff.h?v=4.4" \l "L542)**  *\* @users: User count - see {datagram,tcp}.c*

## 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

1. Quelle: AVR, Low Power 2.4 GHz Transceiver for ZigBee, AT86RF230 [↑](#footnote-ref-1)