8-bit AVR® Microcontrollers

Application Note

AVR2001: AT86RF230 Software Programmer's Guide

Features

- **Complete set of programming sequences for the AT86RF230 radio transceiver.**
  - **Extends the information in the datasheet.**
  - **Detailed Timing Information.**
  - **Hints and Notifications for optimized usage.**
- **Introduction to the Transceiver Access Toolbox and its API.**
- **Optimized usage of the radio transceiver in IEEE 802.15.4™ and ZigBee™ applications.**

1 Introduction

This application note contains the software programmer's guide for the AT86RF230 radio transceiver. This document goes into greater depth than the datasheet when it comes to correct configuration and usage of the features that the radio transceiver provides. Each feature is presented and its usage documented with sequence diagrams and descriptions. Timing information is also included where applicable.

For a person that knows the C programming language and the AVR® microcontroller, it will be easy to convert the programming sequences described in this document, and use them in a wireless application.

The Transceiver Access Toolbox (TAT) is such an implementation. This is a low level library that provides the end user with an easy to use interface to the rich set of features of the radio transceiver. The TAT is presented in the latter half of this document.

Rev. 8087A-AVR-07/07

2 Intended Audience and Overview

This application note is intended for any personal that will design and build applications using the AT86RF230 radio transceiver. The reason for providing a software programmer's guide is to extend and enhance many aspects of the datasheet. This document provides detailed information on the functionality of the radio transceiver optimized for an embedded programmer's point of view.

This application note is divided into two main sections. In chapter 3 to chapter 8 the main features of the radio transceiver is presented. Each feature is documented with sequence diagrams and textual descriptions. Some of the features are also documented with timing information and hints on optimized use and implementation. In chapter 9 the Transceiver Access Toolbox is presented. This is an example of suggested use of the radio transceiver implemented in the C programming language. A complete API description for the TAT can be found in Appendix A.

3 Programming Sequence: Transceiver Setup

This chapter will present the necessary steps to get the radio transceiver up and running after power-on. Some other basic functionality is also introduced, such as channel and output power selection. Table 3-1 gives a quick reference to the described sequences.

Table 3-1. Described Programming Sequences

Sequence	Chapter	Comment
Initialize Radio Transceiver	Section 3.1	Initialize radio after power-on.
Hardware Reset	Section 3.2	Reset all registers to their default value and the state machine.
Reset State Machine	Section 3.3	Rest the state machine. Registers are not reset to their default values.
Get Current Channel	Section 3.4	Read what channel the radio transceiver is operating on.
Set Current Channel	Section 3.5	Set new channel to operate on.
Get Transmit Power	Section 3.6	Read the output power of the radio transceiver.
Set Transmit Power	Section 3.7	Set new output power level.

3.1 Radio Transceiver Initialization

This section describes how to correctly bring the radio transceiver into the TRX_OFF state after a power-on event. Nevertheless, the described programming sequence can be executed from any state without any side effects.

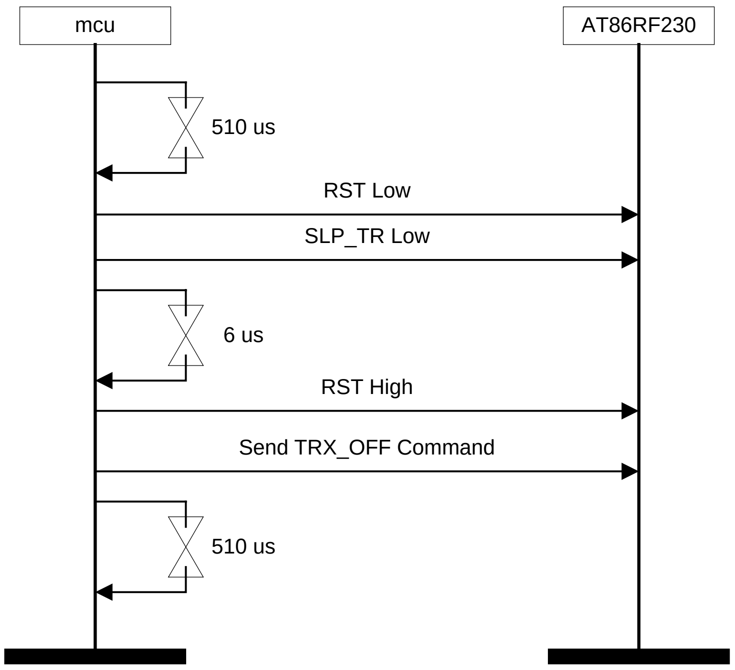
Figure 3-1 depicts the correct sequence of actions to initialize the radio transceiver. As mentioned above the radio transceiver can be in any state when this programming sequence is started.

A hardware reset is initiated by pulling the RST line low. 6 µs later the same line is pulled high again and the hardware reset finished. All registers will have their default values and the state machine will be in P_ON or TRX_OFF state. The state will be

P_ON if the programming sequence was initiated after a power-on event and TRX_OFF in any other case. The TRX_OFF command is written to the TRX_STATE.TRX_CMD sub register to ensure a transition from P_ON to TRX_OFF. This transition takes 510 µs from P_ON. The user must set the interrupt mask, since all interrupts were disabled during the hardware reset.

A description of the associated TAT implementation can be found in section 11.1.

Figure 3-1 MSC for Hardware Initialization



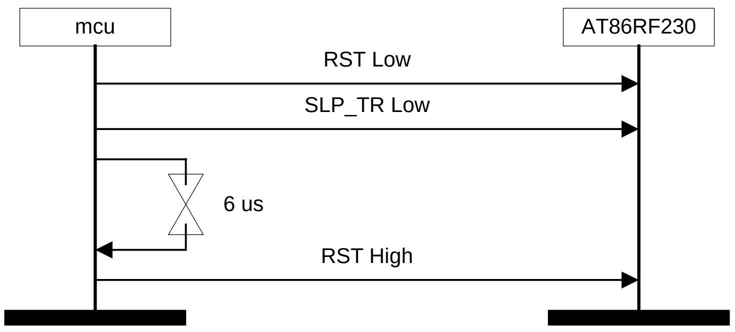
3.2 Hardware Reset

This section describes how to reset the radio transceiver from any state except P_ON. However, this should not be an issue, since the P_ON state can only be reached after a power-on event. The programming sequence will typically be used to recover from fatal errors and bring the radio transceiver to a known state.

Figure 3-2 depicts the correct sequence of actions to reset the radio transceiver. A hardware reset is initiated by pulling the RST line low. 6 µs later the same line is pulled high again and the hardware reset finished. All registers will have their default values and the state machine will be in TRX_OFF state.

A description of the associated TAT implementation can be found in section 11.25.

Figure 3-2 MSC for Hardware Reset



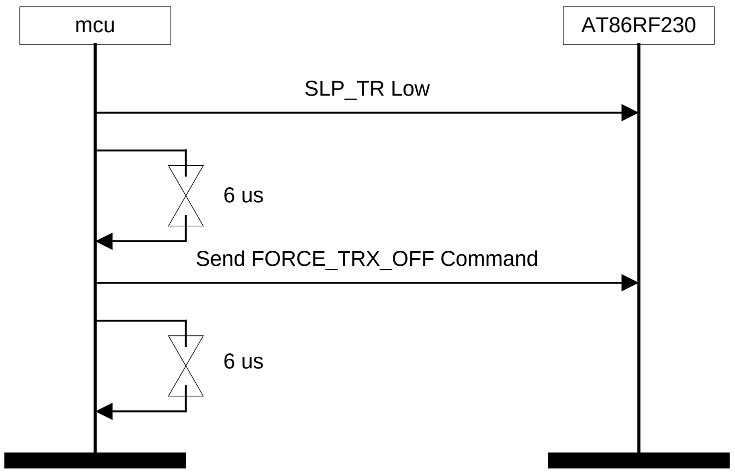
3.3 Reset State Machine

This section describes how to reset the state machine of the radio transceiver to the TRX_OFF state. All other registers are not affected, only currents state of the state machine. The programming sequence will typically be used to recover a locked state machine or to recover from other fatal errors.

Figure 3-3 depicts the correct sequence of actions to reset the state. The sequence assumes that the radio transceiver is not in the P_ON state or the SLEEP state. First the SLP_TR pin is pulled low to ensure that the radio transceiver is not in any of the *_NOCLK states. Then the FORCE_TRX_OFF command is written to the TRX_STATE.TRX_CMD sub register. The TRX_OFF state will be entered after 6 µs.

A description of the associated TAT implementation can be found in section 11.24.

Figure 3-3 MSC for State Machine Reset



3.4 Get Current Channel

The programming sequence that returns the operating channel used by the radio transceiver is very simple. This is simply done by reading the content of the PHY_CC_CCA.CHANNEL sub register. This register can be read at any time without affecting any frame transaction. It is assumed that the radio transceiver is not in the P_ON or the SLEEP state.

A description of the associated TAT implementation can be found in section 11.2.

3.5 Set Current Channel

This section describes the actions required to change the operating channel of the radio transceiver. The operating channel can be changed from any state, except SLEEP, by writing to the PHY_CC_CCA.CHANNEL sub register.

If the radio transceiver is in the RX_ON or PLL_ON state and the new operating channel does not equal the one currently used, the PLL_LOCK interrupt event will be signaled. The PLL should lock to the new channel within 150 µs.

A description of the associated TAT implementation can be found in section 11.3.

3.6 Get Transmit Power

The output power of the radio transceiver can be varied from 3 dBm to –17.2 dBm via the PHY_TX_PWR.TX_PWR sub register. The content of the mentioned sub register will be a number between 0 and 15. And can be read in any state except P_ON and SLEEP. See the datasheet for more information on the mapping between power levels and actual output power in dBm.

A description of the associated TAT implementation can be found in section 11.4.

3.7 Set Transmit Power

As mentioned in section 3.6, the output power can be varied in 16 levels ranging from 3 dBm to – 17.2 dBm. Writing to the PHY_TX_PWR.TX_PWR sub register sets a new output power. This can be done from any of the radio transceiver's states except P_ON and SLEEP.

A description of the associated TAT implementation can be found in section 11.5.

4 Programming Sequence: State Transitions

The state machine of the AT86RF230 is one of the key components to control the operation of the radio transceiver. Each state has a set of functionality associated with it, and by sequencing between the different states these are unveiled to the end user. In Table 4-1 and Table 4-2 the 14 defined states are listed and categorized as either basic or transient.

The transient states can only be reached from one of the basic states when a defined event occurs, such as the reception of a correct preamble and SFD in RX_ON. In this particular scenario the state will change to RX_BUSY until the frame has been completely received. Writing commands into the TRX_STATE.TRX_CMD sub register alters the basic state of the radio transceiver and hence how it operates. This chapter will focus on details in the transition between the basic states. RX_ON_NOCLK and RX_AACK_ON_NOCLK are not covered, since they are simply entered from RX_ON or RX_AACK_ON respectively by pulling the SLP_TR line high.

Table 4-1. Basic Radio Transceiver States

State	Section	Comment
TRX_OFF	Section 4.1	Transitions to TRX_OFF.
RX_ON	Section 4.2	Transitions to RX_ON.

State	Section	Comment
PLL_ON	Section 4.3	Transitions to PLL_ON.
RX_ON_NOCLK	Na	State is reached from RX_ON by pulling the SLP_TR line high.
RX_AACK_ON	Section 4.4	Transitions to RX_AACK_ON.
RX_AACK_ON_NOCLK	na	State is reached from RX_AACK_ON by pulling the SLP_TR line high.
TX_ARET_ON	Section4.5	Transitions to TX_ARET_ON.
SLEEP	Section 4.6	Transitions to SLEEP.

Table 4-2. Transient Radio Transceiver States

State	Comment
P_ON	This state is only entered during the power-up sequence of the radio transceiver.
BUSY_RX	This state is only entered from RX_ON or RX_ON_NOCLK state if a valid Preamble and SFD has been detected.
BUSY_RX_AACK	This state is only entered during frame reception in RX_AACK_ON or BUSY_RX_AACK_NOCLK.
BUSY_RX_AACK_NOCLK	Only entered from the RX_AACK_ON_NOCLK state during frame reception.
BUSY_TX	Entered from the PLL_ON state when a frame transmission is initiated.
BUSY_TX_ARET	Entered from the TX_ARET_ON state when the SLP_TR pin is pulled high.
STATE_TRANSITION	The radio transceiver's state machine is in transition between two states.

4.1 Transitions to TRX_OFF

Transitions to the TRX_OFF state can generally be divided in two.

Figure 4-1shows the correct sequence to do a state transition from any state where the PLL is active, that is: RX_ON, RX_AACK_ON, PLL_ON or TX_ARET_ON, and to TRX_OFF. This is simply done by writing the TRX_OFF command to the TRX_STATE.TRX_CMD sub register. As the figure indicates, this transition should be completed in 1 µs.

Figure 4-2 show another way to reach the TRX_OFF state. The FORCE_TRX_OFF command works as a state machine reset. Writing this command to the TRX_STATE.TRX_CMD sub register will set the state to TRX_OFF from any of the basic states, except SLEEP, within 1 µs.

Figure 4-1 PLL Active to TRX_OFF

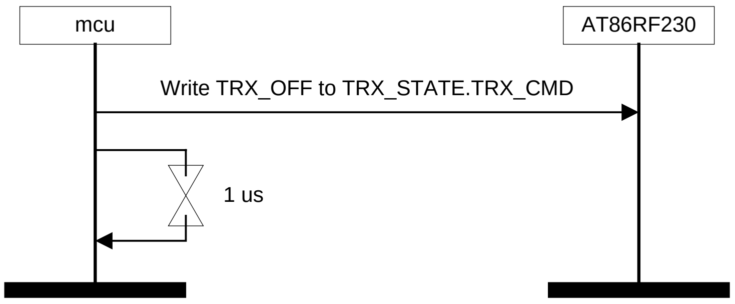
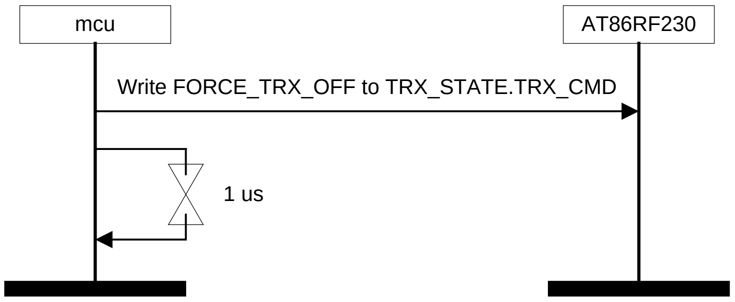


Figure 4-2 TRX_OFF Forced



4.2 Transitions to RX_ON

In the RX_ON state the digital receiver blocks and PLL frequency synthesizer is active. The preamble and SFD detectors are running in the digital processing path, searching for valid IEEE 802.15.4 SHRs. The state is automatically changed to RX_BUSY when a valid preamble and SFD is detected.

Figure 4-3 shows the correct sequence for changing the state to RX_ON from TRX_OFF. The state transition is initiated by writing the RX_ON command to the TRX_STATE.TRX_CMD sub register. The transition is completed within 180 µs. During this time period the PLL_LOCK event is signaled from the radio transceiver on the IRQ pin.

If the radio transceiver is already in a state where the PLL is active, RX_AACK_ON, PLL_ON or TX_ARET_ON, the RX_ON state can be reached within 1 µs after the RX_ON command was issued. Details of this sequence can be seen in Figure 4-4.

Figure 4-3 RX_ON from TRX_OFF

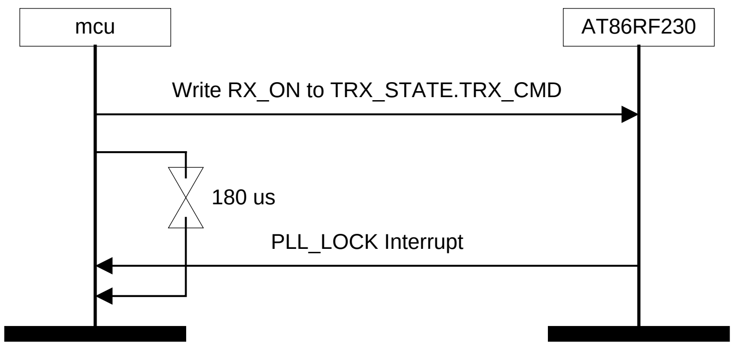
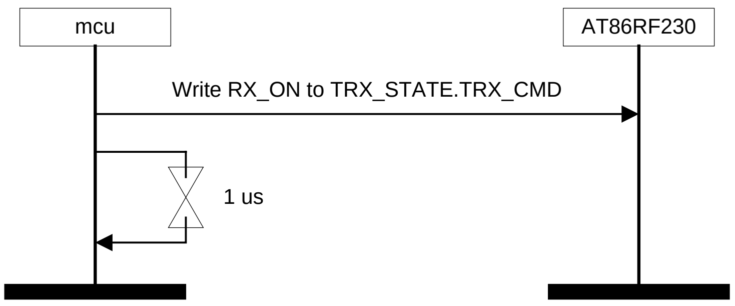


Figure 4-4 RX_ON from PLL Active



4.3 Transitions to PLL_ON

The PLL_ON state corresponds to the TX_ON state in the IEEE 802.15.4 standard. In this state the PLL frequency synthesizer is enabled and locked to the transmit frequency. In other words the radio transceiver is ready for frame transmission (Frame transmission is described in section 6.1 and section 6.2). The sequences described in this section is similar to those of section 4.2.

Figure 4-5 depicts how to do a state transition from TRX_OFF to PLL_ON. The sequence is initiated by issuing the PLL_ON command (writing to the TRX_STATE.TRX_CMD sub register). Within 180 µs the PLL_LOCK event is signaled and the new state of radio transceiver will be PLL_ON.

The PLL_ON state can be reached faster from a state where the PLL is already active (RX_ON, RX_AACK_ON or TX_ARET_ON). For this group of states the transition time is only 1 µs after writing the PLL_ON command to the TRX_STATE.TRX_CMD sub register. See Figure 4-6 for further details.

Figure 4-5 PLL_ON from TRX_OFF

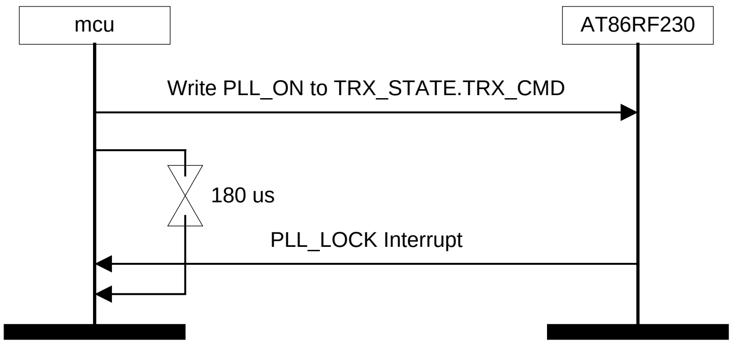
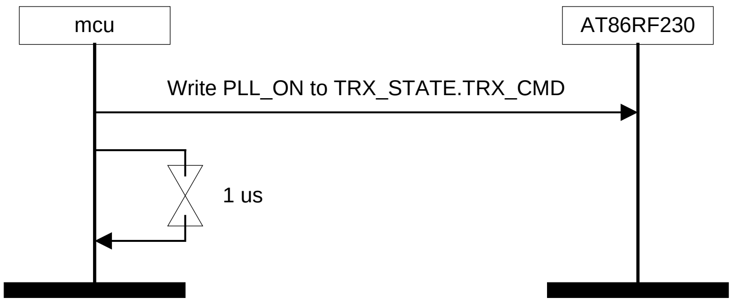


Figure 4-6 PLL_ON from PLL Active states



4.4 Transitions to RX_AACK_ON

The RX_AACK_ON state is one of the radio transceiver's automated states. This state is similar to the previously mentioned RX_ON state, but a frame filter as defined in the IEEE 802.15.4 standard, is applied to any incoming frame and an acknowledge sent if requested by the originator.

It is possible to go directly to the RX_AACK_ON state from TRX_OFF. However, this is not recommended, since it will be impossible to know if the PLL locked or not. This has to do with the PLL_LOCK interrupt being masked out in the RX_AACK_ON state (See the datasheet for further information about interrupt handling in the RX_AACK_ON state). Figure 4-7 depicts how to do the state transition from TRX_OFF, via RX_ON, to RX_AACK_ON and hence ensure that the PLL locks. The sequence is simply built from that described in Figure 4-3 with an additional transition from RX_ON to RX_AACK_ON.

In Figure 4-8 the sequence assumes that the radio transceiver is already in either RX_ON or PLL_ON. In both cases the state transition to RX_AACK_ON is executed by writing the RX_AACK_ON command to the TRX_STATE.TRX_CMD sub register. A 1 µs delay is associated with this particular state transition.

It is not possible to do a direct state transition from TX_ARET_ON to RX_AACK_ON, as it is between RX_ON and PLL_ON. Figure 4-9 illustrates the correct sequence for doing the above-mentioned state transition. It is necessary to an intermediate transition to PLL_ON. However, the complete sequence will take approximately 2 µs.

Figure 4-7 RX_AACK_ON from TRX_OFF

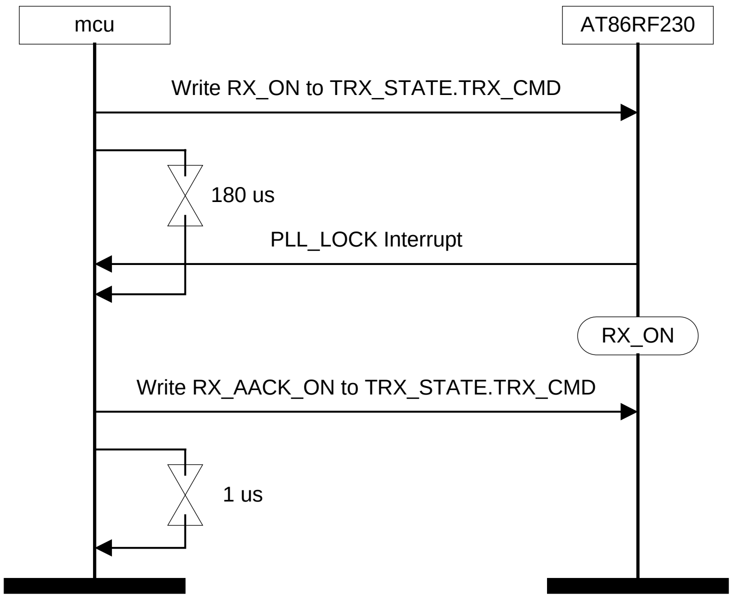

mcu
AT86RF230
Write RX_ON to TRX_STATE.TRX_CMD
180 us
PLL_LOCK Interrupt
RX_ON
Write RX_AACK_ON to TRX_STATE.TRX_CMD
1 us

Figure 4-8 RX_AACK_ON from RX_ON or PLL_ON

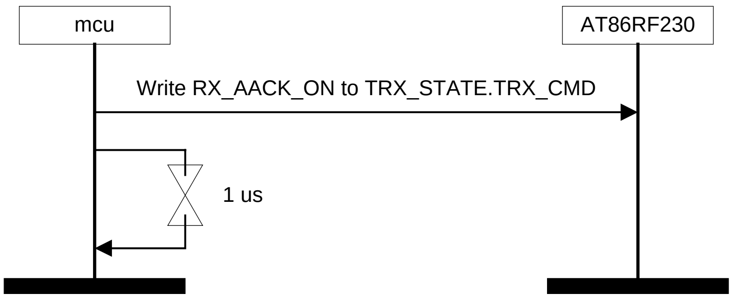
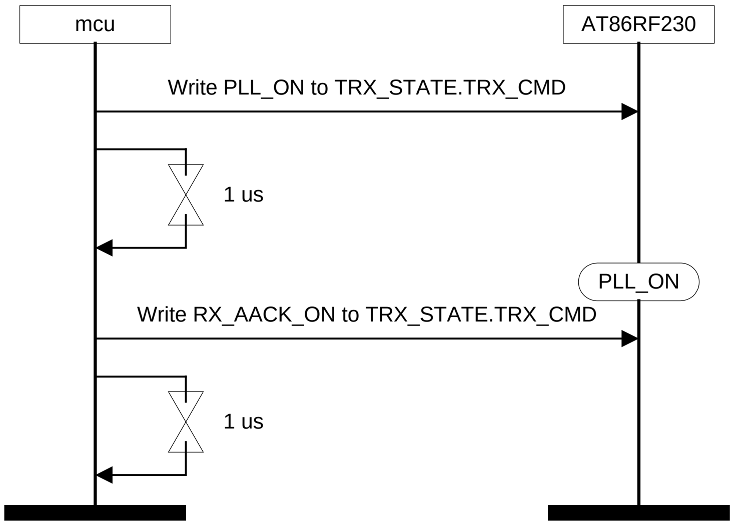


Figure 4-9 RX_AACK_ON from TX_ARET_ON


mcu
AT86RF230
Write PLL_ON to TRX_STATE.TRX_CMD
1 us
PLL_ON
Write RX_AACK_ON to TRX_STATE.TRX_CMD
1 us

4.5 Transitions to TX_ARET_ON

The TX_ARET_ON state is another automated state. This state is similar to the previously mentioned PLL_ON state, but it applies the CSMA algorithm as defined in the IEEE 802.15.4 standard, given that the associated parameters are configured (See section 7.2 and section 8.1 for detailed information about configuration and frame transmission).

It is possible to go directly to the TX_ARET_ON state from TRX_OFF. However, this is not recommended, since it will be impossible to know if the PLL locked or not. This has to do with the PLL_LOCK interrupt being masked out in the TX_ARET_ON state (See the datasheet for further information about interrupt handling in the TX_ARET_ON state). Figure 4-10 depicts how to do the state transition from TRX_OFF, via PLL_ON, to TX_ARET_ON and still ensure that the PLL has locked. The sequence is simply built from that described in Figure 4-5 with an additional transition from PLL_ON to TX_ARET_ON.

In Figure 4-11 the sequence assumes that the radio transceiver is already in either RX_ON or PLL_ON. In both cases the state transition to TX_ARET_ON is executed by writing the TX_ARET_ON command to the TRX_STATE.TRX_CMD sub register. A 1 µs delay is associated with this particular state transition.

It is not possible to do a direct state transition from RX_AACK_ON to TX_ARET_ON as it is between RX_ON and PLL_ON. Figure 4-12 illustrates the correct sequence for doing the above-mentioned state transition. It is necessary to an intermediate transition to RX_ON. However, the complete sequence will take approximately 2 µs.

Figure 4-10 TX_ARET_ON from TRX_OFF

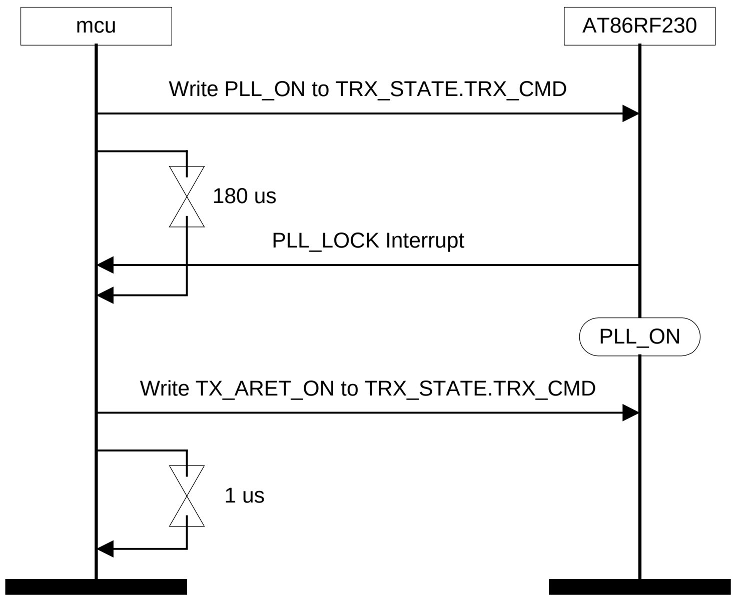

mcu
AT86RF230
Write PLL_ON to TRX_STATE.TRX_CMD
180 us
PLL_LOCK Interrupt
PLL_ON
Write TX_ARET_ON to TRX_STATE.TRX_CMD
1 us

Figure 4-11 TX_ARET_ON from RX_ON or PLL_ON

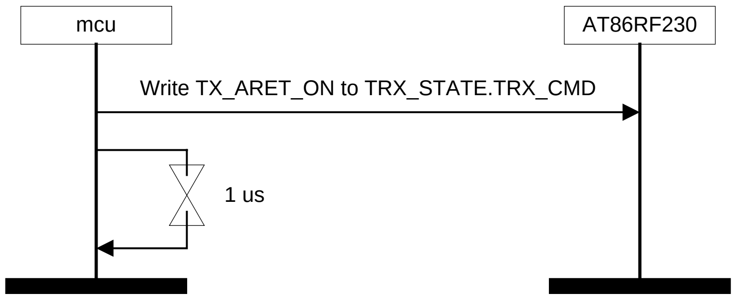
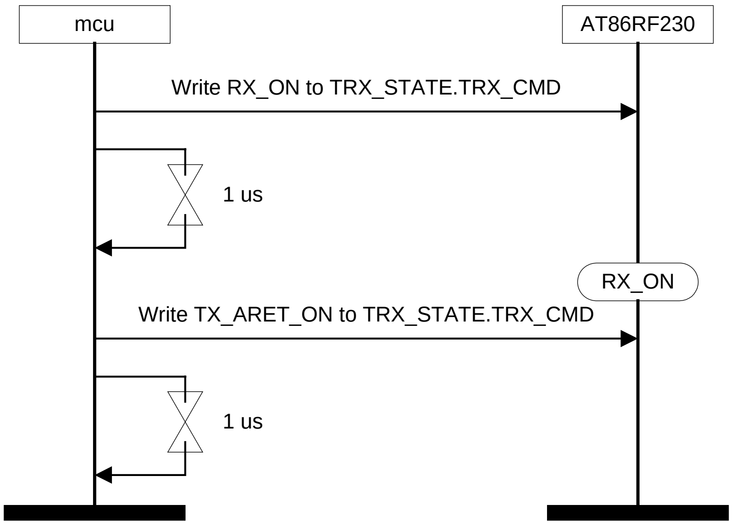


Figure 4-12 TX_ARET_ON from RX_AACK_ON


mcu
AT86RF230
Write RX_ON to TRX_STATE.TRX_CMD
1 us
RX_ON
Write TX_ARET_ON to TRX_STATE.TRX_CMD
1 us

4.6 Transitions to SLEEP

SLEEP is a low power state that is used to minimize the power consumption of the radio transceiver to a bare minimum, without erasing its memory. The state is only accessible from TRX_OFF. The reminder of this section describes how it is possible to enter and leave the SLEEP state from TRX_OFF and associated timing information.

Figure 4-13 depicts the correct sequence to follow when taking the radio transceiver to sleep. The operation is initiated by pulling the SLP_TR pin high in the TRX_OFF state. The radio transceiver will enter SLEEP 35 clock cycles on the CLKM pin later (At the default CLKM frequency of 1MHz this will take 35 µs). The reason for this 35-clock cycles delay is to ensure that any microcontroller clocked from this signal can complete its power-down sequence.

The opposite transition is illustrated in Figure 4-14. It will take up to 880 µs from the SLP_TR pin is pulled low before the radio transceiver's state machine is back in the TRX_OFF state.

Figure 4-13 SLEEP from TRX_OFF

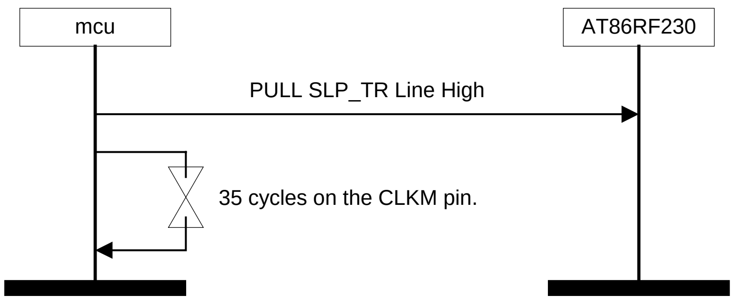
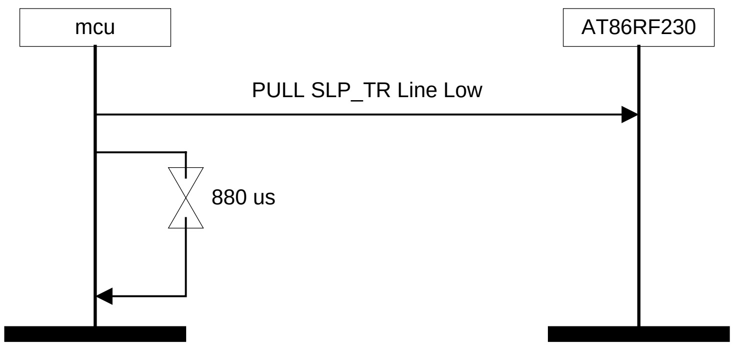


Figure 4-14 TRX_OFF from SLEEP



4.7 Short Summary

- Any transition between RX_ON and PLL_ON or RX_ON and PLL_ON to RX_AACK_ON or TX_ARET_ON can be done within 1 µs.
- The PLL will lock within 180 µs from TRX_OFF.
- The RX_AACK_ON and TX_ARET_ON states should only be accessed from RX_ON or PLL_ON. Not directly from TRX_OFF even if this is possible.

5 Programming Sequence: CCA, ED, LQI and RSSI Measurements

Table 5-1 gives a quick reference to the described sequences within this chapter.

Table 5-1. Described Programming Sequences

Sequence	Chapter	Comment
Clear Channel Assessment	Section 5.1	How to configure and use the CCA.
Energy Detection	Section 5.2	Describes how to use the ED measurement.
Link Quality Indication	Section 5.3	LQI measurement.
Received Signal Strength Indication	Section 5.4	How to read and interpret the RSSI value in different scenario.

5.1 Clear Channel Assessment

The Clear Channel Assessment (CCA) is an important part of the channel access scheme used by IEEE 802.15.4 and ZigBee. This procedure is used to determine if the channel is occupied with traffic or if it is ready for transmission.

5.1.1 Setup and Configuration

The AT86RF230 radio transceiver support three different CCA modes as required by the IEEE802.15.4 standard:

1. Energy Above Threshold: A busy channel shall be reported upon detecting any energy above the threshold.
2. Carrier Sense Only: A busy channel is reported if signals with the same modulation and spreading characteristics of IEEE 802.15.4 are detected. The energy of these signals is not checked.
3. Carrier Sense with Energy Above Threshold: A busy channel is reported if signals with the same modulation and spreading characteristics of IEEE 802.15.4 and energy above a threshold are detected.

The mode and energy threshold parameters are controlled via the PHY_CC_CCA.CCA_MODE sub register and the CCA_THRES.CCA_ED_THRES sub register. It is also possible to tune the carrier sense algorithm, but it is strongly advised not to do so. The reset value of the CCA_THRES.CCA_CS_THRES sub register should be used to avoid a too strict channel access.

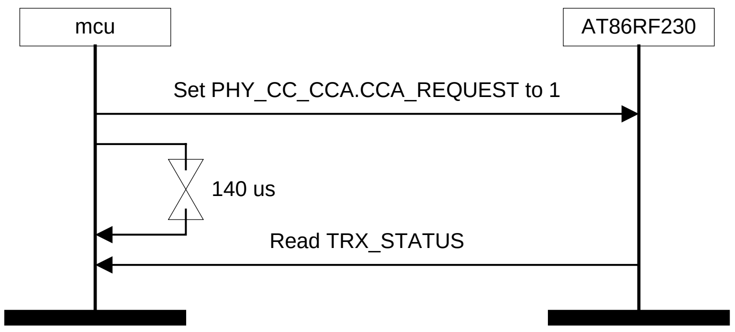
A description of the associated TAT implementation can be found in section 11.9.

5.1.2 Manual Clear Channel Assessment

A CCA can only be initiated if the radio transceiver is in the RX_ON or BUSY_RX state. Writing one to the PHY_CC_CCA.CCA_REQUEST sub register starts the assessment. After 140 µs the algorithm is finished and the result can be read from the TRX_STATUS.CCA_STATUS sub register. Reading the TRX_STATUS.CCA_DONE sub register can be done to check if the algorithm finished within the specified time period. The two mentioned sub registers are located in the same register, and will be cleared when the first read operation occurs after the conversion. See Figure 5-1 for further details.

A description of the associated TAT implementation can be found in section 11.10.

Figure 5-1 Manual CCA



5.2 Energy Detection Measurement

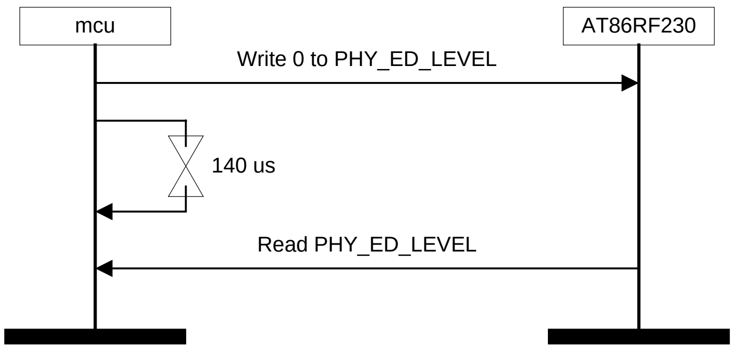
This section describes two ways to do an Energy Detection (ED) measurement. The measurement it self is done by averaging the RSSI (See section 5.4) value over 8 IEEE 802.15.4 symbols.

5.2.1 Manual Energy Detection Measurement

The programming sequence is initiated by writing any value to the PHY_ED_LEVEL.ED_LEVEL register. It is assumed that the radio transceiver is in one of the following states: RX_ON, BUSY_RX. The measurement result can be read back from the PHY_ED_LEVEL.ED_LEVEL register 140 µs after the register was first written.

A description of the associated TAT implementation can be found in section 11.9.

Figure 5-2 Energy Detection Measurement



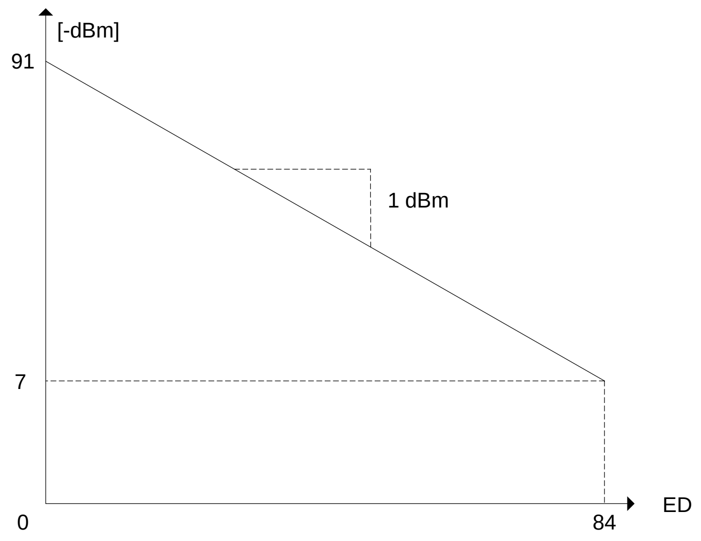
5.2.2 Energy Detection Measurement During Frame Reception

The AT86RF230 radio transceiver will automatically start an ED measurement when a Start Of Frame Delimiter (SFD) is found. 140 µs after the RX_START interrupt is signaled the measured ED level is written to the PHY_ED_LEVEL.ED_LEVEL register. The new value is available in 96 µs after the TRX_END interrupt for the same frame is signaled.

5.2.3 How to Interpret the Energy Detection Value

The PHY_ED_LEVEL.ED_LEVEL register is 8 bits wide, however the valid range is from 0 to 84. All other values will not occur (85 to 255). If zero is read from the register, this indicates that the measured energy is less than –91 dBm. Figure 5-3 illustrates the mapping between register value and dBm.

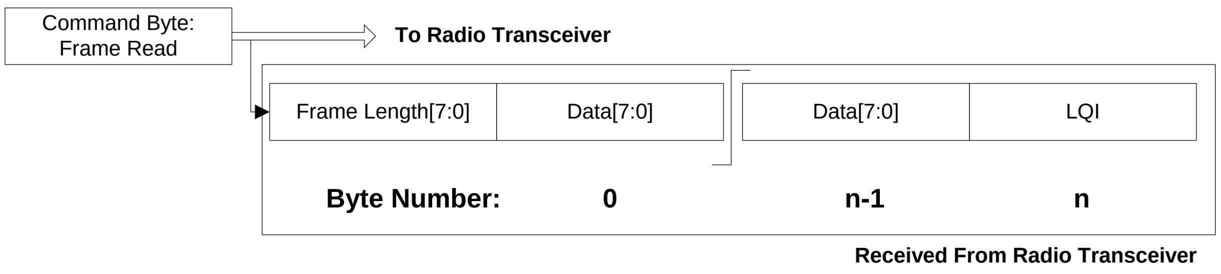
Figure 5-3 Mapping between ED value and dBm



5.3 Link Quality Indication Measurement

The Link Quality Indication (LQI) measurement is done by the radio transceiver each time a new frame is received. The LQI value will be written directly to the frame buffer and is not available through one of the registers. Figure 5-4 shows how the contents of the frame buffer are organized during reception. The last read byte holds the LQI value.

Figure 5-4 LQI in Frame Buffer



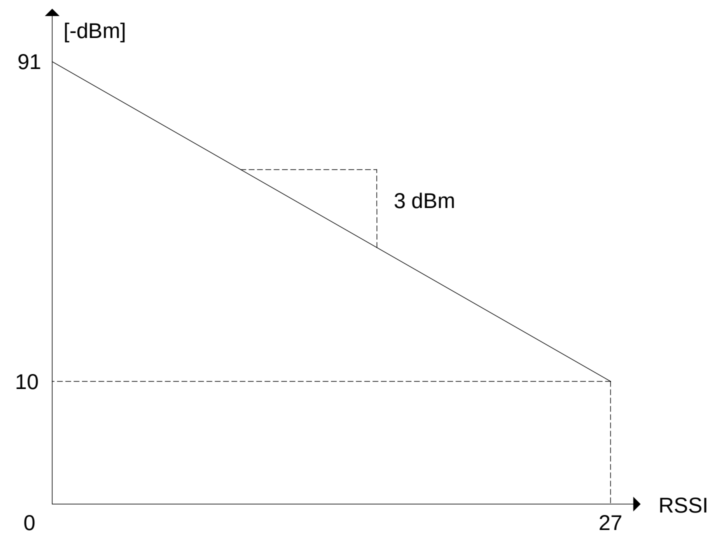
5.4 Received Signal Strength Indication Measurement

The Received Signal Strength Indication (RSSI) is a generic metric often used in wireless communication technology. The AT86RF230 radio transceiver updates it's internal RSSI register every 2 µs, given that it is in one of the following states:

- RX_ON
- BUSY_RX
- RX_AACK_ON
- BUSY_RX_AACK
- RX_AACK_NOCLK (Even if it is assumed that the controller is sleeping).

The RSSI value will be stored in the five bit wide PHY_RSSI.RSSI sub register. Figure 5-5 gives an illustration of how the values read from the RSSI register map to dBm. Subsection 5.4.1 and subsection 5.4.2 describes two different ways to read the RSSI value.

Figure 5-5 Mapping between RSSI value and dBm



5.4.1 Get RSSI

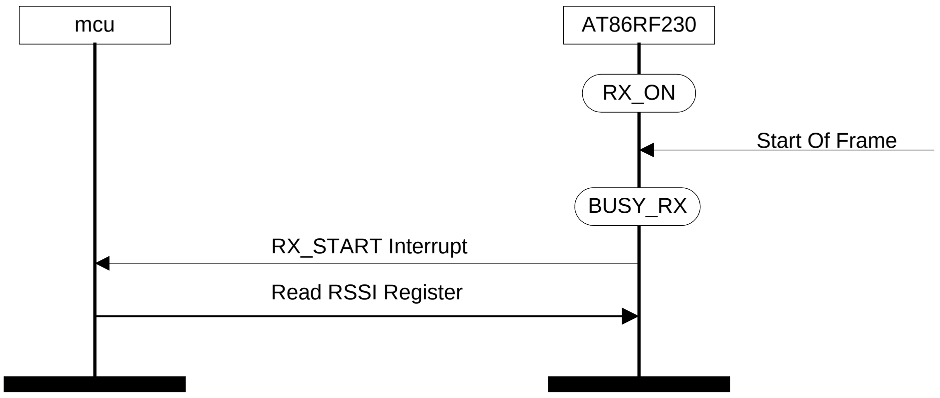
It is valid to read the RSSI value from the RX_ON and BUSY_RX states. The value is obtained by reading the PHY_RSSI.RSSI sub register. The read value will be a number between 0 and 28 indicating the current amount of radio energy measured on the antenna pins.

A description of the associated TAT implementation can be found in section 11.11.

5.4.2 Get Frame RSSI

For some applications it also makes sense to find the RSSI value associated to a newly received frame. This is possible since the RSSI register is updated once per symbol. Reading the register immediately after the RX_START interrupt is signaled represents the RSSI value for the frame under reception. See Figure 5-6 for further details.

Figure 5-6 Read RSSI value during frame reception



6 Programming Sequence: Frame Transaction

This chapter describes how to transmit and receive frames without using any of the automated features of the radio transceiver. In this basic mode of transmission and reception it is important to ensure that the timing is correct, especially for systems aiming for IEEE 802.15.4 compliance.

Table 6-1 gives a quick reference to the described sequences.

Table 6-1. Described Programming Sequences

Sequence	Chapter	Comment
Frame Transmission with Pin Start.	Section 6.1	Writing to the frame buffer and initiate the transmission by toggling the SLP_TR pin.
Frame Transmission with Register Start.	Section 6.2	Writing to the frame buffer and initiate the transmission via the TRX_STATE.TRX_CMD sub register.
Basic Frame Reception.	Section 6.3	Reading from the frame buffer after TRX_END interrupt.
Optimized Frame Reception.	Section 6.4	Reading from the frame buffer after the RX_START interrupt is signaled.

6.1 Frame Transmission with Pin Start

This section describes the programming sequence required to send a frame with pin start. The SLP_TR pin is used for this. This is a dual role pin that is either used for state transitions (sleep) or frame transmission.

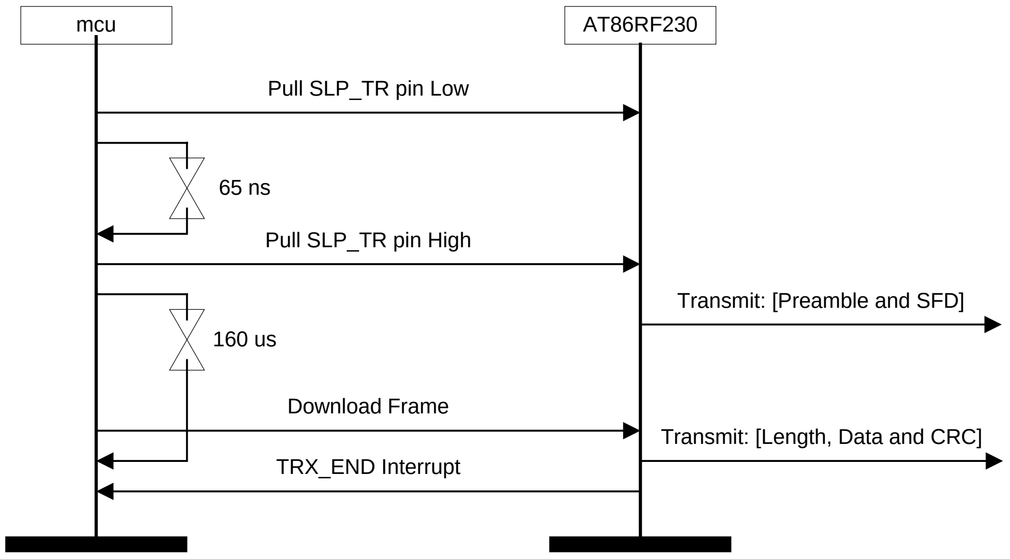
Figure 6-1 shows the details and timing information on how to transmit a frame with pin start. The sequence is initiated by pulling the SLP_TR high, and then back to low. This spike (Must be at least 65ns long) will make the radio transceiver enter the BUSY_TX mode and start transmitting the preamble and Start of Frame Delimiter fields as specified by the IEEE 802.15.4 standard. This synchronization header is 5 octets long and will take 160 µs to transmit (One octet takes 32 µs). It is very important that the frame length is written to the frame buffer within this time, and that the last byte arrives within:

160µs + 32µs * (frame length)

It is an absolute criterion that this timing is met. The transmission will be corrupted otherwise. A TRX_END event will signaled when the last octet is sent from frame buffer. The frame can now be considered sent, and the radio transceiver will return to PLL_ON state.

Downloading the frame completely to the radio transceiver and then toggling the SLP_TR line is also possible. This way it is not necessary to think about the strict timing of the method outlined above, however each transmission cycle takes more time.

Figure 6-1 Frame Transmission with Pin Start

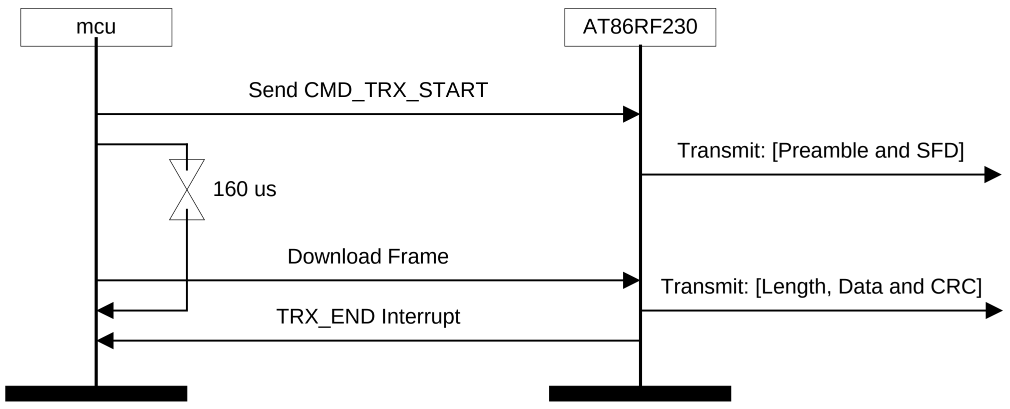


6.2 Frame Transmission with Register Start

This section describes the programming sequence required to send a frame with register start.

Figure 6-2 shows the details and timing information on how to transmit a frame with register start. The sequence is initiated by writing the CMD_TX_START command to the TRX_STATE.TRX_CMD sub register. Rest of the sequence is identical to that described in section 6.1.

Figure 6-2 Frame Transmission with Register Start



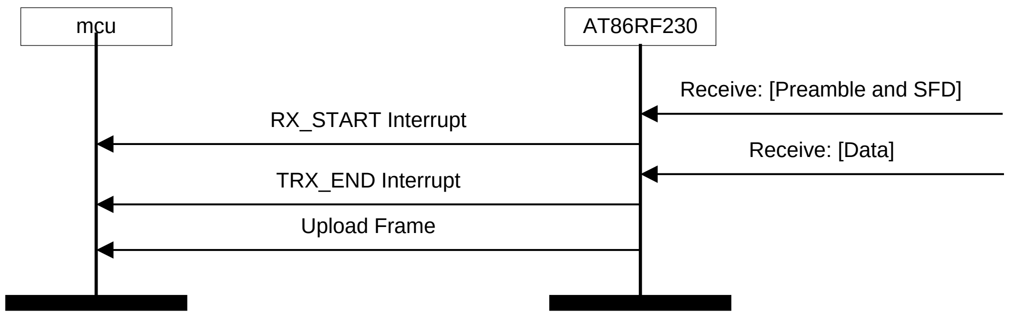
6.3 Basic Frame Reception

This programming sequence is used to load a frame from the radio transceiver after it has been completely stored in the frame buffer.

Figure 6-3 shows the detail in this sequence. The RX_START interrupt event will be signaled when the radio transceiver receives a valid preamble and SFD. The state will change to BUSY_RX and the received data will be copied to the frame buffer. The TRX_END interrupt event will be signaled when the last octet is copied to the frame buffer. The next step is to upload the newly arrived frame from the radio transceiver to the connected microcontroller.

The suggested sequence above is very secure and should be considered as the correct way to handle a frame reception. Section 6.4 shows how to optimize the frame reception so that it takes shorter time. It should be evident that some time could be saved if the frame transaction could start directly after the RX_START interrupt event is signaled, and then read the octets as soon as they are copied to the frame buffer.

Figure 6-3 Basic Frame Reception



6.4 Advanced Frame Reception

The programming sequence described in this section is a time optimized version of that described in section 6.3. This sequence describes how it is possible to start reading from the frame buffer while the radio transceiver is still writing to it. Doing this will save some time compared to the algorithm described in section 6.3, but this is not straight forward due to:

- The speed of the air interface is 250 kbps. One octet (8-bits) will be written to the frame buffer every 32 µs.
- The SPI module on the AVR microcontroller can generally be clocked at half the system clock frequency. At 8 MHz the SPI can be run at 4 Mbps in master mode. Neglecting the data handling on the microcontroller side, a new octet can in theory be read from the frame buffer every 2 µs.

From the above presentation of the communication interfaces, it is clear that the SPI is in average 16 times faster than the air interface. If the frame upload is started directly after the RX_START interrupt event, it is very likely that the frame buffer is under run (Read too fast).

Figure 6-4 illustrates how the problem outlined above can be handled so that it is safe to start reading from the frame buffer immediately after the RX_START interrupt event is signaled from the radio transceiver. Immediately after the interrupt is signaled, a 32 µs delay is necessary to ensure that the frame length is received (First octet after the Preamble and SFD). The frame length is then read and used to calculate a new delay period. This delay period will ensure that the frame buffer is not under run even if the frame upload is started before the TRX_END interrupt event is signaled. Figure 6-5 shows different delay periods as a function of frame length.

Figure 6-4 Advanced Frame Reception

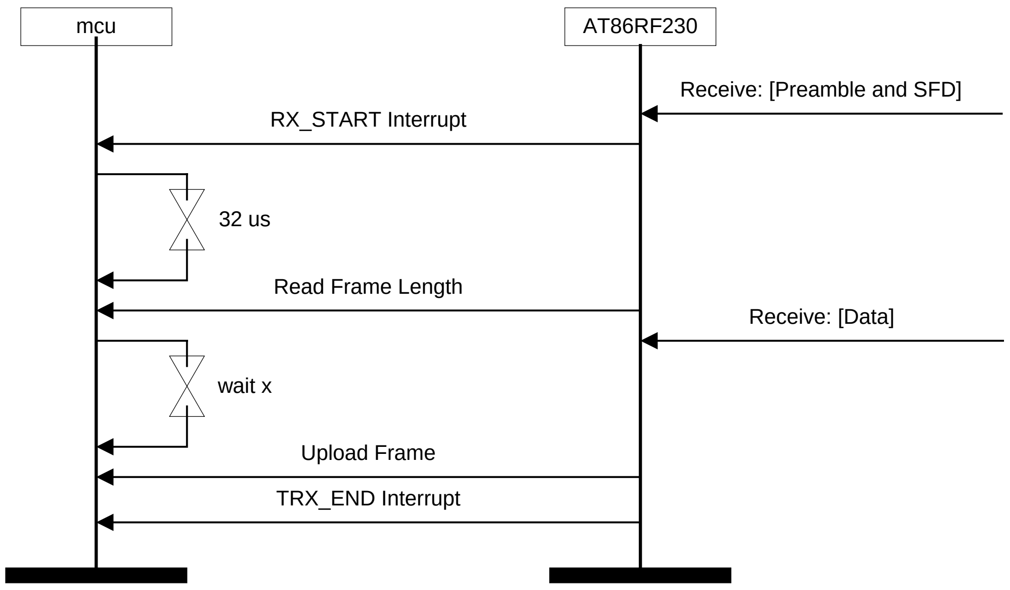
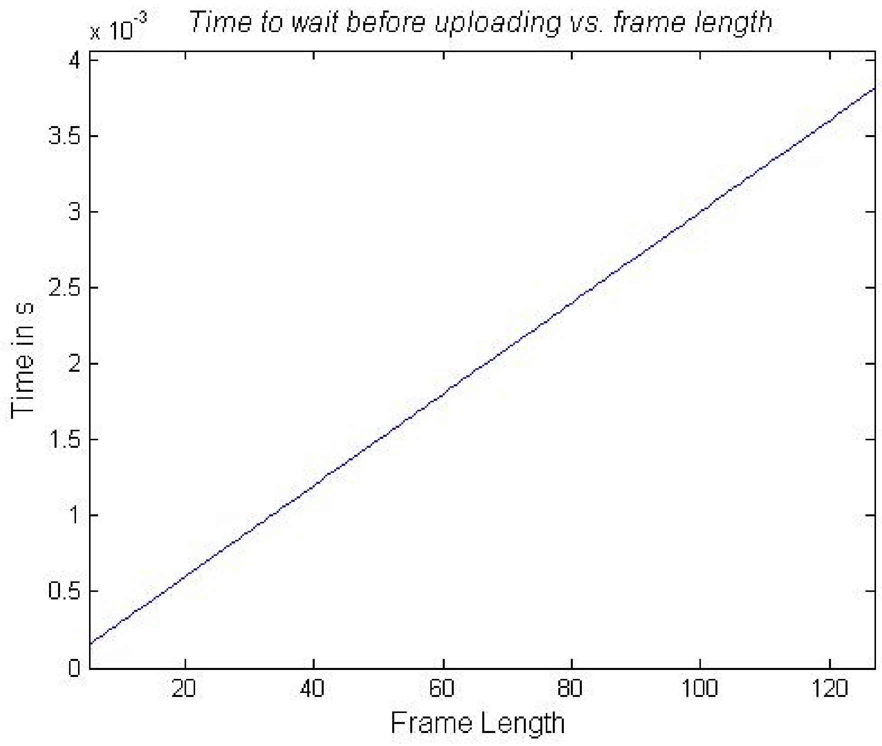


Figure 6-5 Delay before frame upload can start. SPI at 4Mbps



7 Programming Sequence: Setup of Extended Features

This chapter contains information on how configure some of the extended features of the AT86RF230 radio transceiver. These features are thoroughly described in the datasheet and chapter 8.

Table 7-1 gives a quick reference to the described sequences.

Table 7-1. Described Programming Sequences

Sequence	**Chapter**	**Comment**
Setup of the CRC block.	Section 7.1	The radio transceiver has an automated CRC for all frames to be transmitted.
Setup of the CSMA algorithm.	Section 7.2	How to configure the CSMA algorithm.
Setup of the Address Filter.	Section 7.3	How to use the radio transceivers address filter.

7.1 Auto Generated CRC for Transmit Frames

To detect bit errors, the IEEE 802.15.4 standard define a Frame Check Sequence (FCS) mechanism employing a 16-bit CRC algorithm. The 16-bit CRC value is the two last octets of any IEEE 802.15.4 frame. The AT86RF230 radio transceiver has hardware support to do this calculation when frames are written to the internal frame buffer.

The automatic CRC will be applied to any frame written to the frame buffer, when the PHY_TX_PWR.TX_AUTO_CRC_ON sub register is set to 1. Setting the same sub register to zero will disable the CRC.

A description of the associated TAT implementation can be found in section 11.26.

7.2 Automated CSMA Algorithm (TX_ARET_ON)

Section 8.1 describes a special mode that automates the channel access algorithm defined by the IEEE 802.15.4 standard. The necessary sequences to configure this feature are described in the reminder of this section.

The following parameters are used to configure the CSMA-CA algorithm:

- **CSMA_RETRIES:** This parameter defines how many times to retry the CSMA-CA channel access algorithm before giving up. The parameter is controlled via the XAH_CTRL.MAX_CSMA_RETRIES sub register.
- **CSMA_SEED:** This parameter is used to seed the random number generator used by the CSMA-CA algorithm. The parameter is controlled via the CSMA_SEED_0 register and the CSMA_SEED_1.CSMA_SEED_1 sub register.
- **MIN_BE:** This parameter defines the minimum back off exponent in the CSMA-CA algorithm. The parameter is controlled via the CSMA_SEED_1.MIN_BE sub register.
- **MAX_FRAME_RETRIES:** This parameter defines how many times to repeat a frame transmission before it fails. The parameter is controlled via the XAH_CTRL.MAX_FRAME_RETRIES sub register.

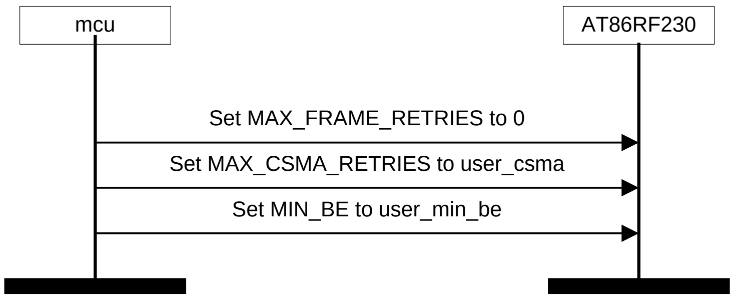
Note: The following is important to ensure correct operation of the radio transceiver:

1. MAX_FRAME_RETRIES must always equal zero. A frame will not be retried after the CSMA-CA algorithm has failed. This must be handled in software.
2. The CSMA_SEED parameter should not be altered from its reset value, since this will corrupt the operation of the random number generator.

Figure 7-1 shows the steps necessary to configure the automated CSMA-CA feature of the radio transceiver. First the XAH_CTRL.MAX_FRAME_RETRIES sub register will be set to zero as described above. Then the XAH_CTRL.MAX_CSMA_RETRIES and CSMA_SEED_1.MIN_BE sub registers can be updated with user-defined values.

A description of the associated TAT implementation can be found in section 11.34.

Figure 7-1 Configuration of Automated CSMA Parameters


mcu
AT86RF230
Set MAX_FRAME_RETRIES to 0
Set MAX_CSMA_RETRIES to user_csma
Set MIN_BE to user_min_be

7.3 Address Filter Setup (RX_AACK)

Section 8.2 describes a special mode that automates the acknowledging of frames received by the radio transceiver. The necessary sequences to configure this feature are described in the reminder of this section.

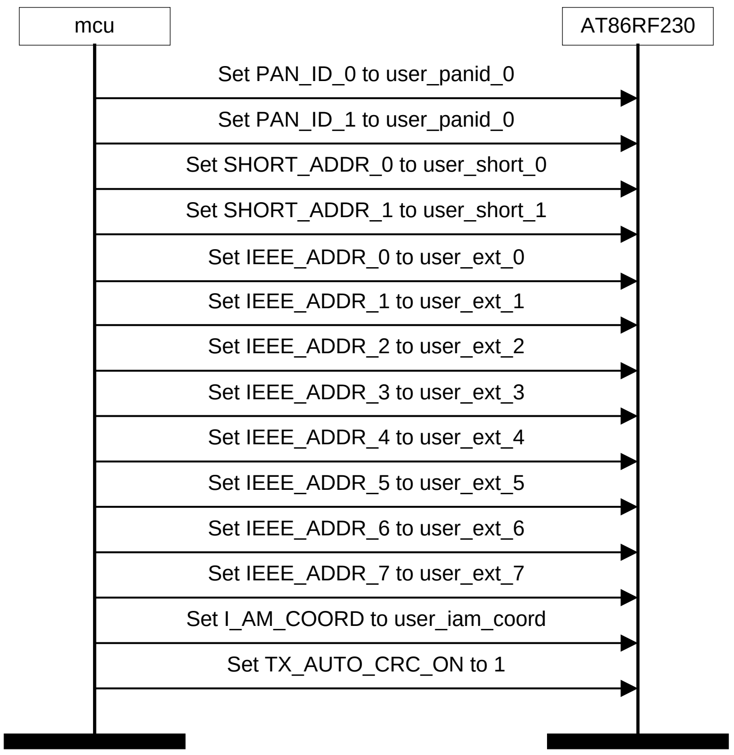
The following parameters are used to configure the automated acknowledge algorithm:

- 16-bit PAN ID: Personal Area Network Identifier (PAN ID) of the device.
- 16-bit Short Address: Short address of the device.
- 64-bit IEEE Address: Extended Address of the device.
- Coordinator Flag: Either one or zero if the device has role as PAN coordinator or not.
- TX_AUTO_CRC_ON Flag: This flag must be set to one.

Figure 7-2 shows the necessary steps to configure the radio transceiver so that the automated acknowledge feature can be used. This programming sequence can be applied in any state except: P_ON, SLEEP, RX_AACK_ON, BUSY_RX_AACK, RX_AACK_NOCLK and BUSY_RX_AACK_NOCLK. The following registers will be written:

1. PAN_ID_0: Lower 8-bits of PAN ID.
2. PAN_ID_1: Higher 8-bits of PAN ID.
3. SHORT_ADDR_0: Lower 8-bits of short address.
4. SHORT_ADDR_1: Higher 8-bits of short address.
5. IEEE_ADDR_0: Lower 8-bits of the IEEE address, bits [7:0].
6. IEEE_ADDR_1: 8-bits of the IEEE address, bits [15:8].
7. IEEE_ADDR_2: 8-bits of the IEEE address, bits [23:16].
8. IEEE_ADDR_3: 8-bits of the IEEE address, bits [31:24].
9. IEEE_ADDR_4: 8-bits of the IEEE address, bits [39:32].
10. IEEE_ADDR_5: 8-bits of the IEEE address, bits [47:40].
11. IEEE_ADDR_6: 8-bits of the IEEE address, bits [55:48].
12. IEEE_ADDR_7: 8-bits of the IEEE address, bits [63:56].
13. CSMA_SEED_1.I_AM_COORD: One if the device’s role is PAN coordinator, zero otherwise.
14. PHY_TX_PWR.TX_AUTO_CRC_ON: Must be set to one.

Figure 7-2 Address Filter Setup


mcu
AT86RF230
Set PAN_ID_0 to user_panid_0
Set PAN_ID_1 to user_panid_0
Set SHORT_ADDR_0 to user_short_0
Set SHORT_ADDR_1 to user_short_1
Set IEEE_ADDR_0 to user_ext_0
Set IEEE_ADDR_1 to user_ext_1
Set IEEE_ADDR_2 to user_ext_2
Set IEEE_ADDR_3 to user_ext_3
Set IEEE_ADDR_4 to user_ext_4
Set IEEE_ADDR_5 to user_ext_5
Set IEEE_ADDR_6 to user_ext_6
Set IEEE_ADDR_7 to user_ext_7
Set I_AM_COORD to user_iam_coord
Set TX_AUTO_CRC_ON to 1

8 Programming Sequence: Extended Frame Transmission and Reception

The AT86RF230 radio transceiver has some features that are hardware accelerated. Section 8.1 describes the accelerated CCA and frame transmission mode, and section 8.2 describes an accelerated frame reception and acknowledge mode. These two modes are tailored for IEEE 802.15.4 operation, and it is assumed that all frames are compliant to this standard. Also note that some of the interrupt events signaled from the radio transceiver has alternative meanings in the extended frame transmission and reception modes.

Table 8-1 gives a quick reference to the described sequences.

Table 8-1. Described Programming Sequences

Sequence	Chapter
Automated CSMA Algorithm and Frame Transmission.	Section 8.1
Automated Frame Reception and Acknowledge.	Section 8.2

8.1 Automated CSMA Algorithm and Frame Transmission (TX_ARET_ON)

As mentioned in the introduction of this chapter, the radio transceiver can provide an accelerated CCA followed by a frame transmission. The hardware will also switch to receive mode and wait for an acknowledge frame if this is requested.

Note that before using this feature it is required that the CSMA algorithm is configured as described in section 7.1 and 7.2.

The programming sequence is illustrated in Figure 8-1. The sequence is initiated by toggling the SLP_TR line similar to the frame transmission with pin start described in section 6.1. The radio transceiver will do a state transition from TX_ARET_ ON to BUSY_TX_ARET and start the CCA check. If the channel access fails a TRX_END interrupt event will be signaled and the TRX_STATE.TRAC_STATUS sub register will indicate a channel access failure and the sequence terminates.

The frame preamble and SFD will be transmitted if the channel is found to be in idle. To save some time, the frame to be transmitted is written to the radio transceiver' frame buffer while it is sent onto the air. Now one of two things can occur:

1. The frame transmitted requests an acknowledgement: SUCCESS (0) is written to the TRX_STATE.TRAC_STATUS sub register if a SFD is detected within 34 symbols. And the timing details shown in Figure 8-2 is not violated.
2. No acknowledge frame is requested: A SFD is not detected within the requested 34 symbols; see Figure 8-2, and NO_ACK (5) is written to the TRX_STATE.TRAC_STATUS sub register.

For both cases the TRX_END interrupt event will be signaled. The radio transceiver will do a state transition from BUSY_TX_ARET back TX_ARET_ ON. The frame initially downloaded to the frame buffer is protected, and a retransmission can be done with the pin start method without writing to the frame buffer again.

Figure 8-1 Automated CSMA and Frame Transmission

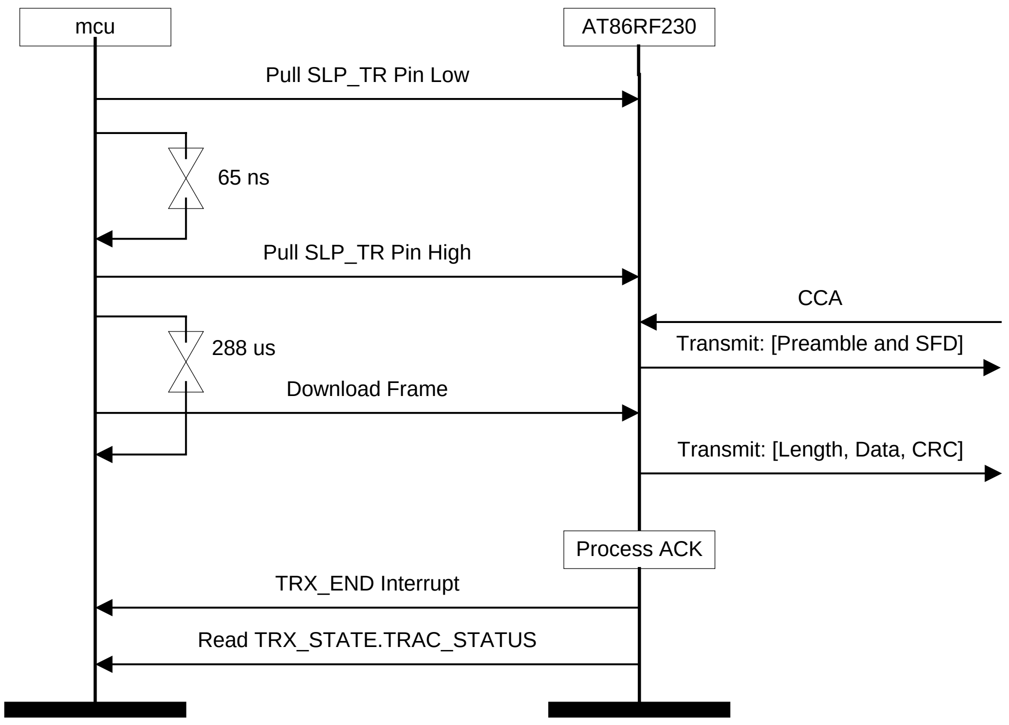
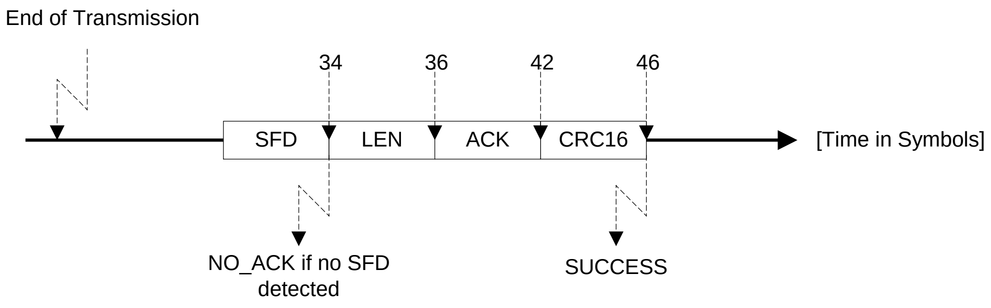


Figure 8-2 Timing Details in AUTO_CSMA



8.2 Automated Frame Reception and Acknowledge (RX_AACK_ON)

This section describes another automated feature that the AT86RF230 radio transceiver provides. With the system properly configured and in the RX_AACK_ON state, it is possible to automatically acknowledge frames that are received.

Note that before using this feature it is required that the address filter is configured as described in section 7.3.

Figure 8-3 shows the details of the automated acknowledge feature that can be applied when a new frame is received. This feature is only available with the radio in the RX_AACK_ON state and with the address filter properly configured. If the radio transceiver is in the RX_AACK_ON state and a valid frame preamble and SFD is detected, a state transition to the BUSY_RX_AACK will be done and the received data written to the internal frame buffer. The TRX_END interrupt event will be signaled to the connected microcontroller and the newly arrived frame can be uploaded.

If an acknowledgement is requested, the radio transceiver will handle this. The radio transceiver will stay in the BUSY_RX_AACK state and start a wait timer that equals the receive-to-transmit turnaround time as specified in the IEEE 802.15.4 standard. This timer will expire within 12 symbols or 192 µs, and then start transmitting the generated acknowledge frame. The radio transceiver will return to the RX_AACK_ON state when the acknowledge frame is sent. Further timing information regarding the acknowledge sequence can be found in Figure 8-4.

Figure 8-3 RX_AACK Details

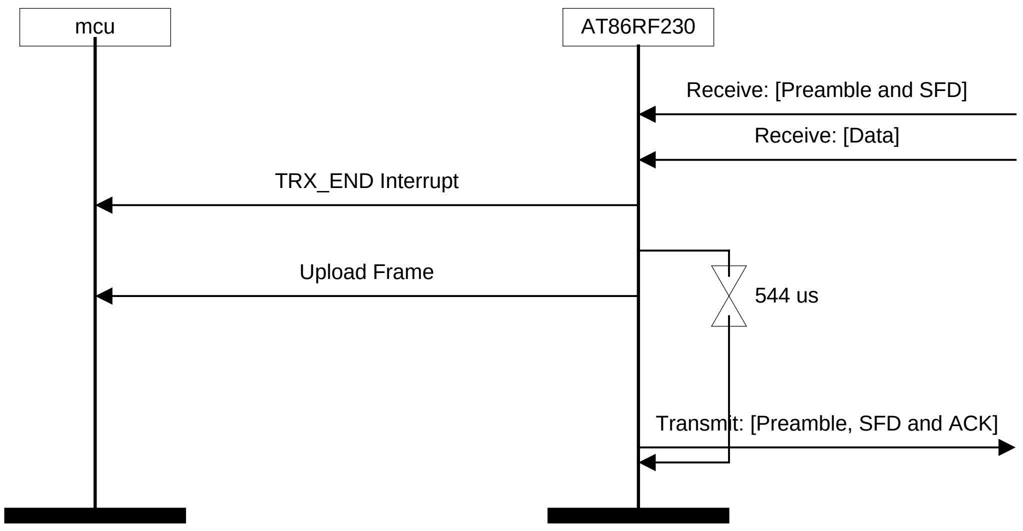
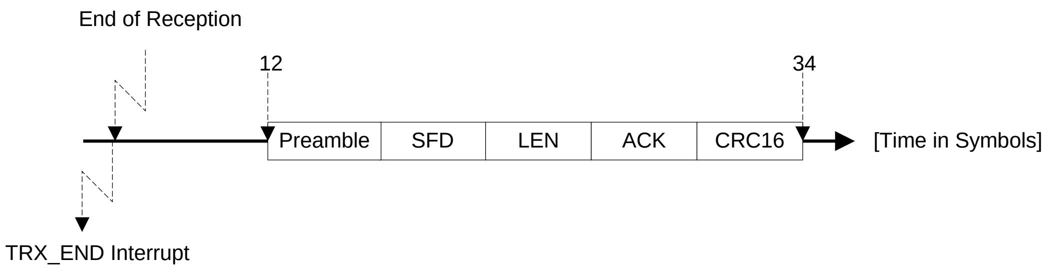


Figure 8-4 Timing Details during RX_AACK



9 The Transceiver Access Toolbox

The Transceiver Access Toolbox was mentioned briefly already in the introduction of this application note. The TAT is implemented as an easy-to-use library that covers most of the functionality provided by the AT86RF230 radio transceiver. The library is completely written in the C programming language.

This chapter will start by presenting the TAT architecture and the associated source code hierarchy. The current release of the TAT is made for the AVR microcontroller. However, special care has been taken so that also other Atmel microcontrollers can be supported.

Section 9.3 contains some hints on how to alter the current source code so that also other targets, such that SAM7 or AVR32 can be supported. Finally there is a overview of the resources used by the TAT for different compilers.

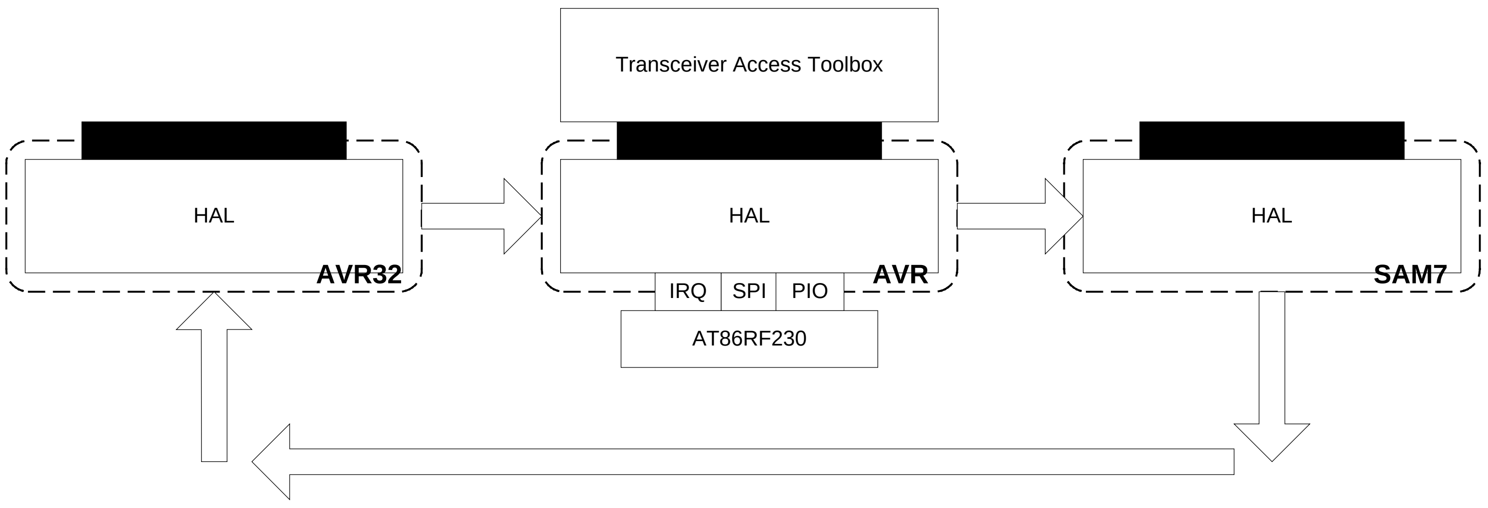
9.1 Architecture

The TAT is architected as two thin layers of code, as can be seen in Figure 9-1. This two-stage design was chosen, because one of the design goals for the TAT is that it should be compatible with multiple Atmel microcontroller architectures (AVR, AVR32 and SAM7) and the AT86RF230 radio transceiver.

Therefore a thin layer of hardware centric code, the Hardware Abstraction Layer (HAL), is found as very the bottom layer. The interface between the HAL and the next higher layer, the TAT, is through a well-defined API. This design ensures that the TAT is fully hardware independent. It will run on top of any HAL that provides the specified API and functionality. Though, the TAT is highly transceiver dependant, and will not work with other IEEE 802.15.4 radio transceivers on the market. It has been optimized and tuned for the AT86RF230 radio transceiver's features and dynamics.

The reminder of this section contains more information about the hardware independent and hardware dependant parts of the TAT.

Figure 9-1 TAT Architecture



9.1.1 Hardware Dependant Source Code – HAL

The HAL and the concept behind it were briefly introduced above. The idea is to have a hardware dependant layer of code, that ensure by software design that the TAT can run on multiple Atmel microcontroller architectures. The HAL is to provide an optimized interface with the necessary hooks to fully control and configure the AT86RF230 radio transceiver.

First it is natural to briefly introduce the necessary hardware interfaces to control and configure the radio transceiver and then the associated API.

Figure 9-2 gives an overview of the microcontroller interface for the AT86RF230 radio transceiver. The interface comprises a slave SPI and additional control signals. The SPI is used for frame buffer and register access. The additional control signals are connected to the PIO/IRQ interface of the microcontroller:

- **CLKM:** Prescalable clock output from the radio transceiver. Can be used as clock source for a timer or the microcontroller itself.
- **IRQ:** This line signals interrupt events from the radio transceiver. It will typically be connected to a pin change interrupt or timer input capture unit on the microcontroller.
- **RST:** This line is used to do a hardware reset of the radio transceiver.
- **SLP_TR:** This line is used both for state transition, SLEEP, and to start frame transmissions.
- **TST:** Special Continuous Transmission Test Mode pin. See the AT86RF230 radio transceiver's datasheet for more information about this pin (Appendix A). Not a part of the HAL API.

From the brief presentation above it is evident that the HAL must contain methods to do register and frame buffer accesses, and to control the necessary IO lines. Table 9-1 gives an overview of the HAL API. This API is the common interface that the TAT is using to interact with the radio transceiver.

Figure 9-2 AT86RF230 Microcontroller Interface

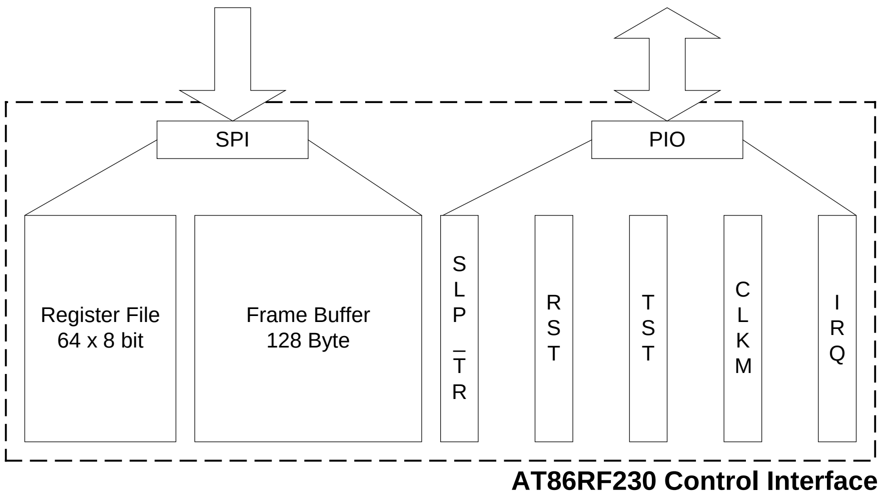


Table 9-1. HAL API

Function Name	Resource	Description
`hal_set_slptr_high( )`	PIO	Pull SLP_TR line high.
`hal_set_slptr_low( )`	PIO	Pull SLP_TR line low.
`hal_get_slptr( )`	PIO	Get state of the SLP_TR line (High/Low).
`hal_set_rst_high( )`	PIO	Pull RST line high.

Function Name	Resource	Description
`hal_set_rst_low( )`	PIO	Pull RST line low.
`hal_get_rst( )`	PIO	Get state of the RST line (High/Low).
`hal_enable_trx_interrupt( )`	PIO/IRQ	Enable interrupt from the radio transceiver.
`hal_disable_trx_interrupt( )`	PIO/IRQ	Disable interrupt from the radio transceiver.
`hal_init(  )`	na	Initialize the HAL. Must be called in advance of any other function.
`hal_register_read( )`	SPI	Read one of the registers in the register file.
`hal_register_write( )`	SPI	Write a value to a register in the register file.
`hal_subregister_read( )`	SPI	Read value from a sub register.
`hal_subregister_write( )`	SPI	Write value to a sub register.
`hal_read_frame_length( )`	SPI	Read length of frame currently stored in the frame buffer.
`hal_frame_read( )`	SPI	Read frame stored in frame buffer.
`hal_frame_write( )`	SPI	Write new frame to the frame buffer.
`hal_sram_read( )`	SPI	Read certain bytes from the frame buffer.
`hal_sram_write( )`	SPI	Write values to certain bytes in the frame buffer.
`hal_get_system_time( )`	TIMER	Returns the current system time in IEEE 802.15.4 symbols (16 µs).

9.1.2 Hardware Independent Code – TAT

Based on the chosen software architecture, the TAT itself contains only hardware independent code, but this code is highly transceiver dependant. The functionality provided by the TAT is optimized and tuned for the AT86RF230 radio transceiver. The implemented functionality is a direct projection of the AT86RF230 radio transceiver's feature set and dynamics.

The features and modules are controlled through simple function calls. Lets use configuration of automated CRC in transmit mode as an example.

The AT86RF230 radio transceiver has hardware support for CRC calculation on outgoing frames. When enabled, this feature will calculate the CCITT-16 CRC over the current frame buffer content, and write it to the last two bytes in the frame buffer. The associated function in the TAT configuring the above-described feature is:

```
void tat_use_auto_tx_crc( bool auto_crc_on )
```

The automated CRC will be enabled or disabled depending on the Boolean parameter supplied in the function call. The above referenced function will internally write the PHY_TX_PWR.TX_AUTO_CRC_ON sub register. This is done by calling the `hal_subregister_write( )` function with the correct parameter set.

The essence in the above outlined example is that any function in the TAT layer will only contain calls to HAL level function exclusively. And this way ensure portability between a wide range of microcontrollers.

9.2 File Description

Illustration of the file directory.

9.3 Configuration and Porting

The current release of the TAT only contains support for the AVR microcontroller. However, the source code contains compile switches for both the AVR32 and SAM7 microcontrollers. These two targets will be added at a later date. If other microcontrollers than those listed here are necessary, it should be a fairly easy task to add support for them:

- Extend the compile switches so your “hal_xxx.c”, “hal_xxx.h” and “compiler_xxx.h” are available for the TAT during compilation.
- Implement a “hal_xxx.c” that complies with the HAL API presented in Table 9-1.

9.4 Resource Consumption

The following two tables, Table 9-2 and Table 9-3, contain information about the resources consumed by the TAT, with respect to different compilers. The Atmega1281 running at 8MHz was used as test platform.

Table 9-2 Resource Consumption - Code Size

Compiler	Program Memory	Data Memory
`IAR EWAVR 4.30A (Medium Optimization Level)`	2194 Bytes	16 Bytes
`AVR-GCC (-Os Optimization Level)`	TBD	TBD

Table 9-3 Resource Consumption – Interrupt Response Time

Compiler	Best Case	Worst Case
`IAR EWAVR 4.30A`	TBD	TBD
`AVR-GCC`	TBD	TBD

10 Abbreviations and Definitions

API	Application Programming Interface
CCA	Clear Channel Assessment
CRC	Cyclic Redundancy Check
CSMA-CA	Carrier Sense Multiple Access with Collision Avoidance
dBm	decibel (dB) of the measured power referenced to one milliwatt (mW).
ED	Energy Detection
FCS	Frame Check Sequence

LQI	Link Quality Indication
MAC	Medium Access Control Layer
MSC	Message Sequence Chart
PAN	Personal Area Network
PHY	Physical Layer
RSSI	Received Signal Strength Indication
SFD	Start Of Frame Delimiter
SHR	Synchronization Header
SPI	Serial Peripheral Interface
TAT	Transceiver Access Toolbox

11 Appendix A: Transceiver Access Toolbox API

11.1 Initialize Transceiver Access Toolbox

11.1.1 Semantics of the function

```
tat_status_t tat_init( void )
```

11.1.2 Appropriate usage

The *tat_init* function is called by the end-user directly after power-on to configure the transceiver access toolbox and the radio transceiver. A call to this function must be done prior to calling any of the other functions in the TAT library. It is safe to call this function from any of the radio transceiver's states.

11.1.3 Effect on receipt

When calling the *tat_init* function the modules necessary to interface and control the AT86RF230 radio transceiver will be initialized. The radio transceiver will be taken form P_ON state to TRX_OFF.

Table 11-1. Possible Return Values

Value	Description
TAT_SUCCESS	The initialization procedure was successful. The radio transceiver is now in the TRX_OFF state.
TAT_UNSUPPORTED_DEVICE	The current device connected to the controller is not an Atmel AT86RF230 radio transceiver.
TAT_TIMED_OUT	The radio transceiver was not able to enter the TRX_OFF state within time.

11.2 Get Current Channel

11.2.1 Semantics of the function

```
uint8_t tat_get_operating_channel( void )
```

11.2.2 Appropriate usage

The *tat_get_operating_channel* is called by the end-user to find what channel (Frequency) the radio transceiver's PLL is locked to.

11.2.3 Effect on receipt

When calling the *tat_get_operating_channel* the PHY_CC_CCA.CHANNEL sub register is read and its content returned.

Table 11-2 Possible Return Values

Value	Description
11 to 26	Valid channels.

11.3 Set Current Channel

11.3.1.1 Semantics of the function

```
tat_status_t tat_set_operating_channel( uint8_t channel )
```

Table 11-3. tat_set_operating_channel Parameters

Name	Type	Valid Range	Description
channel	uint8_t	11 to 26	New channel to lock on.

11.3.1.2 Appropriate usage

The *tat_set_operating_channel* function is called by the end-user to change the operating channel.

11.3.1.3 Effect on receipt

When calling the *tat_set_operating_channel* the channel parameter will first be checked. TAT_INVALID_ARGUMENT is returned if it is not within the valid range defined in Table 11-3. It is possible to change the radio transceiver's operating channel from any state except SLEEP. TAT_WRONG_STATE is returned if the device is found to be at sleep.

Now the new channel is written to the PHY_CC_CCA.CHANNEL sub register. If the radio transceiver's state equals RX_ON or PLL_ON, a delay of 150 s is added to the sequence, so that the PLL has time to lock to the new channel.

The PHY_CC_CCA.CHANNEL sub register is read to verify if the new operating channel was set. TAT_SUCCESS is returned if the channel was set correctly, and TAT_TIMED_OUT in any other case.

Table 11-4. Possible Return Values

Value	Description
TAT_SUCCESS	Channel changed to the one specified.
TAT_WRONG_STATE	The radio transceiver is not in a valid state to change the channel. It is in SLEEP.
TAT_INVALID_ARGUMENT	The channel parameter is not within the specified bounds.

Value	Description
TAT_TIMED_OUT	The channel switch took too long time. PLL did not lock within the specified time.

11.4 Get Transmit Power Level

11.4.1 Semantics of the function

```
uint8_t tat_get_tx_power_level( void )
```

11.4.2 Appropriate usage

The *tat_get_tx_power_level* function is called by the end-user to read the current output power level of the radio transceiver.

11.4.3 Effect on receipt

When calling the *tat_get_tx_power_level* the current output power level of the radio transceiver will be read (PHY_TX_PWR.TX_PWR sub register).

Table 11-5. Possible Return Values

Value	Description
0 to 15	Current output power in TX Power Setting.

11.5 Set Transmit Power Level

11.5.1 Semantics of the function

```
tat_status_t tat_set_tx_power_level( uint8_t power_level )
```

Table 11-6. tat_set_tx_power_level Parameters

Name	Type	Valid Range	Description
power_level	uint8_t	0 to 15	Output Power in TX Power Setting

11.5.2 Appropriate usage

The *tat_set_tx_power_level* function is called by the end-user whenever he needs to change the output power of the radio transceiver.

11.5.3 Effect on receipt

When calling the *tat_set_tx_power_level* the new output power will be checked to be within the defined bounds. TAT_INVALID_ARGUMENT will be returned if the supplied function parameter is out of bounds.

TAT_WRONG_STATE is returned if the radio transceiver is found to be in the SLEEP state. If these checks are passed, the new power setting will be written to the PHY_TX_PWR.TX_PWR sub register and TAT_SUCCESS is returned.

Table 11-7. Possible Return Values

Value	Description

Value	Description
TAT_SUCCESS	New output power set.
TAT_INVALID_ARGUMENT	The supplied function argument is out of bounds.
TAT_WRONG_STATE	The transmit power cannot be changed in SLEEP.

11.6 Detect Energy on Antenna Pins

11.6.1 Semantics of the function

```
tat_status_t tat_do_ed_scan( uint8_t *ed_level )
```

Table 11-8. tat_do_ed_scan Parameters

Name	Type	Valid Range	Description
ed_level	uint8_t *	pointer	Pointer to where the detected Energy Level is to be written.

11.6.2 Appropriate usage

The *tat_do_ed_scan* function is used to measure the energy level sensed on the antenna pins. This measurement is an important building block for the PLME-ED.request primitive found in the IEEE 802.15.4 standard.

11.6.3 Effect on receipt

When calling the *tat_do_ed_scan* the current state of the radio transceiver will be read. TAT_WRONG_STATE is returned if the transceiver's state is not found to be: RX_ON or BUSY_RX.

An energy detection measurement will then be initiated. After 140 µs the result is available and will be written to the supplied pointer (Memory Address). TAT_SUCCESS is finally returned indicating that the measurement is valid.

Table 11-9. Possible Return Values

Value	Description
TAT_SUCCESS	ED scan successful.
TAT_WRONG_STATE	This function cannot be called for the current state of the radio transceiver.

11.7 Get CCA Mode

11.7.1 Semantics of the function

```
tat_status_t tat_get_cca_mode( void )
```

11.7.2 Appropriate usage

The *tat_get_cca_mode* function returns the current CCA mode used by the radio transceiver.

11.7.3 Effect on receipt

When calling the *tat_get_cca_mode* the PHY_CC_CCA.CCA_MODE sub register is read and its contents returned.

Table 11-10. Possible Return Values

Value	Description
0 to 3	The possible CCA modes.

11.8 Get CCA Mode Energy Threshold

11.8.1 Semantics of the function

```
uint8_t tat_get_ed_threshold( void )
```

11.8.2 Appropriate usage

The *tat_get_ed_threshold* function is used to read the energy threshold that is used by the CCA algorithm in mode 1 and mode 3.

11.8.3 Effect on receipt

When calling the *tat_get_ed_threshold* the CCA_THRES.CCA_ED_THRES sub register is read and its contents returned.

Table 11-11 Possible Return Values

Value	Description
0 to 15	Current CCA_ED_THRES level.

11.9 Set CCA Mode

11.9.1 Semantics of the function

```
tat_status_t tat_set_cca_mode( cca_mode_t mode, uint8_t
ed_threshold )
```

Table 11-12. tat_set_cca_mode Parameters

Name	Type	Valid Range	Description
mode	cca_mode_t	1 to 3	New CCA mode to use.
ed_threshold	uint8_t	0 to 15	ED threshold

11.9.2 Appropriate usage

The *tat_set_cca_mode* function is called by the end-user to configure one of the three different clear channel assessment algorithms supported by the radio transceiver.

11.9.3 Effect on receipt

When calling the *tat_set_cca_mode* the mode parameter will first be read. Three different flavors of the clear channel assessment algorithm is supported:

- **Mode 1: Energy Above Threshold Only**

  In this mode the radio transceiver will measure the energy detected on the antenna pins, and determine if this level is above or below the ed_threshold. Example: If the ed_threshold is set to be –61 dBm and the measured energy level is –60 dBm the channel is assumed to be busy

- **Mode 2: Carrier Sense Only**

  In the carrier sense mode a special correlation scheme is used on each received symbol to determine if this is an IEEE 802.15.4 signal or not. In this mode it is not possible to set a threshold for the algorithm. The radio transceiver supports this, but tests done by Atmel show that it is not advisable to alter the default value.

- **Mode 3: Carrier Sense with Energy above Threshold**

  This mode combines the two previous modes. A "AND" operation is applied on the two results. A busy channel is only signaled if both Mode1 and Mode2 indicate a busy channel.

TAT_INVALID_ARGUMENT will be returned if one of the function's parameters is out of bounds. A state check is also done to verify that the radio transceiver is not in the SLEEP state. TAT_WRONG_STATE is returned if the radio transceiver is found to be at SLEEP.

After the initial state and parameter check is passed, the CCA_THRES.CCA_ED_THRES and PHY_CC_CCA.CCA_MODE sub registers are updated. Finally TAT_SUCCESS is returned to confirm a successful configuration.

Table 11-13. Possible Return Values

Value	Description
TAT_SUCCESS	CCA mode configuration was successful.
TAT_WRONG_STATE	Device is at SLEEP.
TAT_INVALID_ARGUMENT	One of the three function parameters is out of bounds.

11.10 Do Clear Channel Assessment

11.10.1 Semantics of the function

```
tat_status_t tat_do_cca( void )
```

11.10.2 Appropriate usage

The *tat_do_cca* function will execute the clear channel assessment algorithm and detect if a channel is ready for transmission or not.

11.10.3 Effect on receipt

When calling the *tat_do_cca* the current state of the radio transceiver will be read. TAT_WRONG_STATE is returned if the transceiver is not found to be in the PLL_ON. The transceiver is then requested to do a state transition to RX_ON. TAT_TIMED_OUT is returned if this transition fails.

The clear channel assessment is initiated if the state transition was successful (Writing one to the PHY_CC_CCA.CCA_REQUEST sub register). After 140 µs the result is ready and read from the radio transceiver. TAT_TIMED_OUT will be returned if the CCA algorithm did not finish within the specified time.

Finally TAT_CCA_IDLE or TAT_CCA_BUSY will be returned if the channel was found to be ready for transmission or not.

Table 11-14. Possible Return Values

Value	Description
TAT_CCA_IDLE	The channel is in idle state. State of the radio transceiver is RX_ON.
TAT_CCA_BUSY	The channel is in busy state. State of the radio transceiver is RX_ON.
TAT_WRONG_STATE	Transceiver not in PLL_ON state.
TAT_TIMED_OUT	Internal state transition or CCA request timed out. State of the radio transceiver is unknown when this value is returned.

11.11 Get Received Signal Strength Indication

11.11.1 Semantics of the function

```
tat_status_t tat_get_rssi_value( uint8_t *rssi )
```

Table 11-15. tat_get_rssi_value Parameters

Name	Type	Valid Range	Description
rssi	uint8_t*	na	Pointer to the measured RSSI level.

11.11.2 Appropriate usage

The *tat_get_rssi_value* function is called by the end-user to read the RSSI value from the radio, typically immediately after a new frame has been received.

11.11.3 Effect on receipt

When calling the *tat_get_rssi_value* the current state of the radio transceiver will be read. TAT_WRONG_STATE is returned if the state is not found to be RX_ON or BUSY_RX.

If the state check described above is passed, the PHY_RSSI.RSSI sub register will be read and the value written to the memory location pointed to by the rssi parameter. Finally TAT_SUCCESS is returned.

Table 11-16. Possible Return Values

Value	Description
TAT_SUCCESS	RSSI read successfully.
TAT_WRONG_STATE	The radio transceiver is not in RX_ON or BUSY_RX.

11.12 Get Battery Threshold used by Battery Monitor

11.12.1 Semantics of the function

```
uint8_t tat_batmon_get_voltage_threshold( void )
```

11.12.2 Appropriate usage

The *tat_batmon_get_voltage_threshold* is used to read the current voltage threshold used by the battery monitor.

11.12.3 Effect on receipt

When calling the *tat_batmon_get_voltage_threshold* the BATMON.BATMON_VTH sub register is read and returned.

Table 11-17. Possible Return Values

Value	Description
0 to 15	Threshold Voltage.

11.13 Get Voltage Range used by Battery Monitor

11.13.1 Semantics of the function

```
uint8_t tat_batmon_get_voltage_range( void )
```

11.13.2 Appropriate usage

The *tat_batmon_get_voltage_range* function is used to find out if the low or high voltage threshold is currently used by the battery monitor. The battery monitor can select between a low or high voltage thresholds. Low voltage threshold runs from 1.70V to 2.45V with 16 steps defined, while high voltage range runs from 2.55V to 3.675V with the same number of steps defined.

11.13.3 Effect on receipt

When calling the *tat_batmon_get_voltage_range* the BATMON.BATMON_HR is read and returned.

Table 11-18. Possible Return Values

Value	Description
0 or 1	Low or High Voltage Used.

11.14 Configure Battery Monitor

11.14.1 Semantics of the function

```
tat_status_t tat_batmon_configure( bool range, uint8_t
voltage_threshold )
```

Table 11-19. tat_batmon_configure Parameters

Name	Type	Valid Range	Description
range	bool	False or true	Setting this parameter to false selects the low voltage threshold. True indicates that the high voltage threshold is to be used.

Name	Type	Valid Range	Description
voltage_threshold	uint8_t	0 to 15	Threshold Voltage.

11.14.2 Appropriate usage

The *tat_batmon_configure* function is used to set the voltage threshold that the battery shall use.

If BATMON_HR = = 0 //Low Voltage Range.

Threshold Voltage = 1.70 + 0.05 * voltage_threshold [V]

If BATMON_HR = = 1 //High Voltage Range.

Threshold Voltage = 2.55 + 0.075 * voltage_threshold [V]

11.14.3 Effect on receipt

When calling the *tat_batmon_configure* the current state of the radio transceiver and supplied function parameters will be checked. If they are found to be out of bounds, or the device is in SLEEP, then TAT_INVALID_ARGUMENT or TAT_WRONG_STATE is returned.

Then the BATMON.BATMON_HR and BATMON.BATMON_VTH are written with the supplied range and voltage threshold. Finally TAT_SUCCESS is returned.

Table 11-20. Possible Return Values

Value	Description
TAT_SUCCESS	Configuration successful.
TAT_WRONG_STATE	Radio transceiver is at SLEEP.
TAT_INVALID_ARGUMENT	One of the supplied function parameters is out of bounds.

11.15 Get Status from Battery Monitor

11.15.1 Semantics of the function

```
tat_status_t tat_batmon_get_status( void )
```

11.15.2 Appropriate usage

The *tat_batmon_get_status* function is used to poll the BATMON.BATMON_OK bit. If the voltage threshold is less than the measured supply voltage, this bit will be 1. In any other case the bit is zero, indicating that the supply voltage is lower than the configured threshold.

11.15.3 Effect on receipt

When calling the *tat_batmon_get_status* the BATMON.BATMON_OK sub register is read. TAT_BAT_LOW is returned if the bit equals 0, and TAT_BAT_OK returned if the bit equals one.

Table 11-21. Possible Return Values

Value	Description
TAT_BAT_OK	Supply voltage above the predefined voltage threshold.
TAT_BAT_LOW	Supply voltage is below the predefined voltage threshold.

11.16 Get CLKM Frequency

11.16.1 Semantics of the function

```
uint8_t tat_get_clock_speed( void )
```

11.16.2 Appropriate usage

The *tat_get_clock_speed* function is used to read the current frequency supplied on the CLKM pin of the radio transceiver.

11.16.3 Effect on receipt

When calling the *tat_get_clock_speed* the TRX_CTRL_0.CLKM_CTRL sub register will be read and returned.

Table 11-22. Possible Return Values

Value	Description
CLKM_DISABLED	No Clock available.
CLKM_1MHZ	1 MHz clock frequency on the CLKM pad.
CLKM_2MHZ	2 MHz clock frequency on the CLKM pad.
CLKM_4MHZ	4 MHz clock frequency on the CLKM pad.
CLKM_8MHZ	8 MHz clock frequency on the CLKM pad.
CLKM_16MHZ	16 MHz clock frequency on the CLKM pad.

11.17 Set CLKM Frequency

11.17.1 Semantics of the function

```
tat_status_t tat_set_clock_speed( bool direct, uint8_t clock_speed
)
```

Table 11-23. tat_set_clock_speed Parameters

Name	Type	Valid Range	Description
direct	bool	false or true	True indicates that the frequency should be changed directly. If this parameter is set to false, the frequency on the CLKM pin will be changed next time the device returns from SLEEP.
clock_speed	uint8_t	0 to 5	Possible frequency settings.

11.17.2 Appropriate usage

The *tat_set_clock_speed* function is used to configure the frequency supplied on the CLKM pin. Writing to the TRX_CTRL_0.CLKM_CTRL sub register controls this:

- 0: Clock disabled. CLKM pin high.
- 1: Frequency on the CLKM pin is 1 MHz.
- 2: Frequency on the CLKM pin is 2 MHz.
- 3: Frequency on the CLKM pin is 4 MHz.
- 4: Frequency on the CLKM pin is 8 MHz.
- 5: Frequency on the CLKM pin is 16 MHz.

The direct parameter is used to control if the frequency is changed directly or not. Two options are available:

1. direct == true: Change the frequency directly.
2. direct == false: The frequency will be changed when the radio transceiver does a state transition from SLEEP to TRX_OFF.

11.17.3 Effect on receipt

When calling the *tat_set_clock_speed* the `clock_speed` parameter will be checked. TAT_INVALID_ARGUMENT will be returned if an illegal frequency is requested.

Then the TRX_CTRL_0.CLKM_SHA_SEL sub register will be written reflecting the `direct` function parameter (Direct or indirect frequency change). The TRX_CTRL_0.CLKM_CTRL sub register is also updated with the requested frequency. Finally TAT_SUCCESS is returned.

Table 11-24. Possible Return Values

Value	Description
TAT_SUCCESS	CLKM frequency updated
TAT_INVALID_ARGUMENT	The `clock_speed` parameter is out of bounds.

11.18 Calibrate Filter

11.18.1 Semantics of the function

```
tat_status_t tat_calibrate_filter( void )
```

11.18.2 Appropriate usage

The *tat_calibrate_filter* function is used to calibrate the Single Side Band Filter transfer function independent of temperature and part-to-part variations.

11.18.3 Effect on receipt

When calling the *tat_calibrate_filter* the current state of the radio transceiver will be read. TAT_WRONG_STATE is returned if the transceiver's state is not found to be: TRX_OFF or PLL_ON.

The filter tuning is initiated by writing one to the FTN_CTRL.FTN_START sub register. Then a small delay cycle is entered so that the filter tuning cycle can complete.

When the delay is completed, the FTN_CTRL.FTN_START sub register is read again to verify if the filter tuning completed or not. TAT_TIMED_OUT is returned if the filter-tuning algorithm did not complete within the specified time, and TAT_SUCCESS returned if it did complete within the specified time.

Table 11-25. Possible Return Values

Value	Description
TAT_SUCCESS	Filter Calibration Successful.
TAT_WRONG_STATE	Radio transceiver is not in TRX_OFF or PLL_ON state.
TAT_TIMED_OUT	The calibration algorithm did not finish within time.

11.19 Calibrate PLL

11.19.1 Semantics of the function

```
tat_status_t tat_calibrate_pll( void )
```

11.19.2 Appropriate usage

The *tat_calibrate_pll* function is used to calibrate the PLL's center frequency and delay unit.

11.19.3 Effect on receipt

When calling the *tat_calibrate_pll* the current state of the radio transceiver will be read. TAT_WRONG_STATE is returned if the transceiver's state is not found to be in PLL_ON.

The calibration it self is initiated by writing a one to the PLL_CF.PLL_CF_START and PLL_DCU.PLL_DCU_START sub registers. The calibration itself takes about 150 µs, so a delay loop is entered. When the delay is finished, the status of the two calibrations initiated will be checked.

TAT_SUCCESS is returned if the PLL calibration was successful, in any other case TAT_TIMED_OUT is returned.

Table 11-26. Possible Return Values

Value	Description
TAT_SUCCESS	PLL calibration successful.
TAT_WRONG_STATE	The radio transceiver is not in the PLL_ON state.
TAT_TIMED_OUT	PLL calibration did not complete within the specified time.

11.20 Get Radio Transceiver's State

11.20.1 Semantics of the function

```
uint8_t tat_get_trx_state( void )
```

11.20.2 Appropriate usage

The *tat_get_trx_state* function is used to read the current state of the radio transceiver’s state machine.

11.20.3 Effect on receipt

When calling the *tat_get_trx_state* the TRX_STATUS.TRX_STATUS sub register will be read, and the current state of the radio transceiver is returned.

Table 11-27. Possible Return Values

Value	Description
P_ON (0)	Current radio transceiver state is P_ON.
BUSY_RX (1)	Current radio transceiver state is BUSY_RX.
BUSY_TX (2)	Current radio transceiver state is BUSY_TX.
RX_ON (6)	Current radio transceiver state is RX_ON.
TRX_OFF (8)	Current radio transceiver state is TRX_OFF.
PLL_ON (9)	Current radio transceiver state is PLL_ON.
SLEEP (15)	Current radio transceiver state is SLEEP.
BUSY_RX_AACK (17)	Current radio transceiver state is BUSY_RX_AACK.
BUSY_TX_ARET (18)	Current radio transceiver state is BUSY_TX_ARET.
RX_AACK_ON (22)	Current radio transceiver state is RX_AACK_ON.
TX_ARET_ON (25)	Current radio transceiver state is TX_ARET_ON.
RX_ON_NOCLK (28)	Current radio transceiver state is RX_ON_NOCLK.
RX_AACK_ON_NOCLK (29)	Current radio transceiver state is RX_AACK_ON_NOCLK.
BUSY_RX_AACK_NOCLK (30)	Current radio transceiver state is BUSY_RX_AACK_NOCLK.
State Transition (31)	The radio transceiver is busy doing a state transition.

11.21 Set Radio Transceiver’s State

11.21.1 Semantics of the function

```
tat_status_t tat_set_trx_state( uint8_t new_state )
```

Table 11-28. tat_set_trx_state Parameters

Name	Type	Valid Range	Description
new_state	uint8_t	6, 8, 9, 22, 25	Requested new state.

11.21.2 Appropriate usage

The *tat_set_trx_state* function is used to change state in the radio transceiver’s state machine.

11.21.3 Effect on receipt

When calling the *tat_set_trx_state* the `new_state` parameter will be checked to be a valid state for transition. Then it is checked if the radio transceiver is sleeping or busy, and TAT_WRONG_STATE or TAT_BUSY_STATE is returned respectively.

If these checks pass, the requested state will be written to the TRX_STATE.TRX_CMD sub register. Depending upon the requested state, a delay loop is entered since some transitions can take up to 180 µs.

Then the TRX_STATUS.TRX_STATUS sub register is read and the value compared with the `new_state` parameter. TAT_SUCCESS is returned if the read state is the same as the requested state, otherwise TAT_TIMED_OUT is returned.

Table 11-29. Possible Return Values

Value	Description
TAT_SUCCESS	State transition successful.
TAT_WRONG_STATE	The requested `new_state` is not valid.
TAT_BUSY_STATE	The radio transceiver is currently busy.
TAT_TIMED_OUT	The state transition took too long time.

11.22 Go to SLEEP

11.22.1 Semantics of the function

```
tat_status_t tat_enter_sleep_mode( void )
```

11.22.2 Appropriate usage

The *tat_enter_sleep* function is used to do a state transition from any state to SLEEP.

11.22.3 Effect on receipt

When calling the *tat_enter_sleep* the current state of the radio transceiver will be read. TAT_SUCCESS is returned if the radio transceiver is already in the SLEEP state.

If the radio transceiver is not currently sleeping, it will be forced to the TRX_OFF state by using the FORCE_TRX_OFF command (Writing 3 to the TRX_STATE.TRX_CMD sub register). TAT_TIMED_OUT is returned if the transition to TRX_OFF takes too long.

When it is clear that the radio transceiver is in the TRX_OFF state, pulling the SLP_TR line high enters SLEEP. TAT_SUCCESS is finally returned.

Table 11-30. Possible Return Values

Value	Description
TAT_SUCCESS	SLEEP state entered.
TAT_TIMED_OUT	State transition from TRX_OFF to SLEEP took too long time.

11.23 Leave SLEEP

11.23.1 Semantics of the function

```
tat_status_t tat_leave_sleep_mode( void )
```

11.23.2 Appropriate usage

The *tat_leave_sleep* function is used to do a state transition from SLEEP to TRX_OFF.

11.23.3 Effect on receipt

When calling the *tat_leave_sleep* the current state of the radio transceiver will be read. TAT_SUCCESS if returned if the radio transceiver is not sleeping.

If the device is at sleep, the SLP_TR pin will be pulled low and a delay loop started. The state transition from SLEEP to TRX_OFF takes approximately 880 µs.

The current state of the radio transceiver will be read after the delay has finished. TAT_SUCCESS is returned if the state is found to be TRX_OFF. TAT_TIMED_OUT is returned in any other case.

Table 11-31. Possible Return Values

Value	Description
TAT_SUCCESS	State transition from SLEEP to TRX_OFF was successful.
TAT_TIMED_OUT	The state transition took too long time.

11.24 Reset State Machine

11.24.1 Semantics of the function

```
void tat_reset_state_machine( void )
```

11.24.2 Appropriate usage

The *tat_reset_state_machine* function is called by the end-user to reset the radio transceiver's state machine. The state after the reset will be TRX_OFF.

11.24.3 Effect on receipt

When calling the *tat_reset_state_machine* the SLP_TR line will be pulled low to ensure that any *_NOCLK state is exited. Then the FORCE_TRX_OFF command will be issued and the state transition to TRX_OFF will be completed within 6 µs.

11.25 Reset Radio Transceiver

11.25.1 Semantics of the function

```
void tat_reset_trx( void )
```

11.25.2 Appropriate usage

The *tat_reset_trx* function is called by the end-user to reset the radio transceiver's registers and state machine. The state after the reset will be TRX_OFF.

11.25.3 Effect on receipt

When calling the *tat_reset_trx* the RST line will be toggled. This will cause all registers to be reset and the state machine to enter the TRX_OFF state.

11.26 Enable or Disable Automatic CRC

11.26.1 Semantics of the function

```
void tat_use_auto_tx_crc( const bool auto_crc_on )
```

Table 11-32. tat_use_auto_tx_crc Parameters

Name	Type	Valid Range	Description
auto_crc_on	bool	false	Turn auto CRC off.
		true	Turn auto CRC on.

11.26.2 Appropriate usage

The *tat_use_auto_tx_crc* function is called by the end-user to enable or disable the automatic CRC feature for transmit frames.

11.26.3 Effect on receipt

When calling the *tat_use_auto_tx_crc* the PHY_TX_PWR.TX_AUTO_CRC_ON sub register will be read and checked against the supplied function parameter

When using the automatic CRC mechanism the data length must be increased by two (16 bit CRC) bytes and two bytes must be added to the frame payload. Typically two 0x00 bytes will be appended at the end of the frame during download to the radio transceiver's frame buffer.

11.27 Send Data

11.27.1 Semantics of the function

```
tat_status_t tat_send_data( uint8_t *data, uint8_t data_length )
```

Table 11-33. tat_send_data Parameters

Name	Type	Valid Range	Description
data	uint8_t*	pointer	Pointer to data to transmit.
data_length	Uint8_t	0 to 127	Length of data to transmit.

11.27.2 Appropriate usage

The *tat_send_data* function is used to load data into the radio transceiver's frame buffer and send it.

11.27.3 Effect on receipt

When calling the *tat_send_data* the `data_length` function parameter will be checked. TAT_INVALID_ARGUMENT is returned if the parameter is found to be out of bounds.

The current state of the radio transceiver is also read, and checked to equal PLL_ON. TAT_WRONG_STATE is returned if the read state does not equal PLL_ON.

If these checks are passed, the SLP_TR pin will be toggled and the frame pointed to by the `data` parameter downloaded. The initial toggle of the SLP_TR pin will actually start the transmission, but the fact that a preamble and SFD is transmitted before the frame buffer is utilized. This saves some time, but can cause an error if data is not available in the frame buffer at the correct point in time (This is met by keeping the SPI rate higher than 250kbps). TAT_SUCCESS is returned when the frame has been downloaded.

Table 11-34. Possible Return Values

Value	Description
TAT_SUCCESS	Data sent successfully.
TAT_WRONG_STATE	Data can only be sent from the PLL_ON state.
TAT_INVALID_ARGUMENT	The `data_length` parameter is out of bounds.

11.28 Get PAN ID

11.28.1 Semantics of the function

```
uint16_t tat_get_pan_id( void )
```

11.28.2 Appropriate usage

The *tat_get_pan_id* function is used to read the current PAN ID used by the radio transceiver’s frame filter.

11.28.3 Effect on receipt

When calling the *tat_get_pan_id* the PAN_ID_0 and PAN_ID_1 registers will be read, and their values returned as a 16-bit number.

Table 11-35. Possible Return Values

Value	Description
0 to 0xFFFF	PAN ID used by the address filter.

11.29 Set PAN ID

11.29.1 Semantics of the function

```
void tat_set_pan_id( uint16_t new_pan_id )
```

Table 11-36. tat_set_pan_id Parameters

Name	Type	Valid Range	Description
new_pan_id	uint16_t	0 to 0xFFFF	New PAN ID.

11.29.2 Appropriate usage

The *tat_set_pan_id* function is used to set a new PAN ID used by the radio transceiver's address filter.

11.29.3 Effect on receipt

When calling the *tat_set_pan_id* the new_pan_id parameter will be written to the PAN_ID_0 and PAN_ID_1 register.

11.30 Get Short Address

11.30.1 Semantics of the function

```
uint16_t tat_get_short_address( void )
```

11.30.2 Appropriate usage

The *tat_get_short_address* function is used to read the current short address used by the radio transceiver's address filter.

11.30.3 Effect on receipt

When calling the *tat_get_short_address* a 16-bit return value will be generated from the content in the SHORT_ADDR_0 and SHORT_ADDR_1 registers.

Table 11-37. Possible Return Values

Value	Description
0 to 0xFFFF	Possible Short Addresses.

11.31 Set Short Address

11.31.1 Semantics of the function

```
void tat_set_short_address( uint16_t new_short_address )
```

Table 11-38. tat_set_short_address Parameters

Name	Type	Valid Range	Description
new_short_address	uint16_t	0 to 0xFFFF	New Short Address.

11.31.2 Appropriate usage

The *tat_short_address* function is used to set the short address used by the radio transceiver's address filter.

11.31.3 Effect on receipt

When calling the *tat_short_address* the SHORT_ADDR_0 and SHORT_ADDR_1 registers will be updated according to the `new_short_address` parameter.

11.32 Get Extended Address

11.32.1 Semantics of the function

```
uint64_t tat_get_extended_address( void )
```

11.32.2 Appropriate usage

The *tat_get_extended_address* function is used to read the current extended address used by the radio transceiver's address filter.

11.32.3 Effect on receipt

When calling the *tat_get_extended_address* a 64-bit return value will be generated from the content in the IEEE_ADDR_0, IEEE_ADDR_1, IEEE_ADDR_2, IEEE_ADDR_3, IEEE_ADDR_4, IEEE_ADDR_5, IEEE_ADDR_6 and IEEE_ADDR_7 registers.

Table 11-39. Possible Return Values

Value	Description
0 to 2^{64}-1	Valid Extended Addresses.

11.33 Set Extended Address

11.33.1 Semantics of the function

```
void tat_set_extended_address( uint64_t new_extended_address )
```

Table 11-40. tat_set_extended_address Parameters

Name	Type	Valid Range	Description
new_extended_address	uint64_t	0 to 2^{64}-1	New Extended Address.

11.33.2 Appropriate usage

The *tat_set_extended_address* function is used to set the extended address (IEEE Address) used by the radio transceiver's address filter.

11.33.3 Effect on receipt

When calling the *tat_set_extended_address* the IEEE_ADDR_0, IEEE_ADDR_1, IEEE_ADDR_2, IEEE_ADDR_3, IEEE_ADDR_4, IEEE_ADDR_5, IEEE_ADDR_6 and IEEE_ADDR_7 registers will be updated according to the `new_extended_address` parameter.

11.34 Configure the CSMA Algorithm

11.34.1 Semantics of the function

```
tat_status_t tat_configure_csma( const uint8_t seed0, const uint8_t
be_csma_seed1 )
```

Table 11-41. tat_configure_csma arguments

Name	Type	Valid Range	Description

Name	Type	Valid Range	Description
seed0	uint8_t	0 to 0xFF	Seed in the random back off algorithm.
be_csma_seed1	uint8_t	0 to 0xFF	Compound variable containing seed1, min_be and max_csma_retries.

11.34.2 Appropriate usage

The *tat_configure_csma* function is called by the end-user to configure the CSMA algorithm used by the auto retry mechanism.

11.34.3 Effect on receipt

When calling the *tat_configure_csma* the current state of the radio transceiver will be read. TAT_WRONG_STATE will be returned if the state is found to be in SLEEP.

All the function parameters will then be extracted and written to the associated registers. TAT_SUCCESS will finally be returned.

The XAH_CTRL.MAX_FRAME_RETRIES sub register will be set to 0 to ensure correct operation of the auto mode (See the AT86RF230 datasheet's errata section for further information).

Table 11-42. Possible Return Values

Value	Description
TAT_SUCCESS	CSMA algorithm configured.
TAT_WRONG_STATE	The radio transceiver is currently sleeping, and it is not possible to configure the CSMA algorithm.

11.35 Clear BAT_LOW flag

11.35.1 Semantics of the function

```
void tat_clear_bat_low_flag_ref( void )
```

11.35.2 Appropriate usage

The *tat_clear_bat_low_flag_ref* function is used to disable the user interface for the BAT_LOW interrupt.

11.35.3 Effect on receipt

When calling the *tat_clear_bat_low_flag_ref* the BAT_LOW bit in the `tat_isr_flag_mask` will be cleared. The result is that an associated interrupt flag will not be incremented when a BAT_LOW interrupt is signaled.

11.36 Set BAT_LOW flag

11.36.1 Semantics of the function

```
void tat_set_bat_low_flag_ref( uint8_t volatile *flag )
```

Table 11-43. tat_set_bat_low_flag_ref Parameters

Name	Type	Valid Range	Description

Name	Type	Valid Range	Description
flag	uint8_t *	pointer	BAT_LOW flag.

11.36.2 Appropriate usage

The *tat_set_bat_low_flag_ref* function is used to register an interrupt flag to the BAT_LOW interrupt and enable the associated interrupt user interface.

11.36.3 Effect on receipt

When calling the *tat_set_bat_low_flag_ref* the BAT_LOW bit in the `tat_isr_flag_mask` will be set. The result is that supplied interrupt flag will be incremented when a BAT_LOW interrupt is signaled.

11.37 Enable BAT_LOW Interrupt

11.37.1 Semantics of the function

```
void tat_enable_bat_low_isr( void )
```

11.37.2 Appropriate usage

The *tat_enable_bat_low_isr* function is used to re-enable the BAT_LOW interrupt from the radio transceiver. The BAT_LOW interrupt will be disabled the first time it is signaled. This is done to prevent that the interrupt is continuously signaled.

When the supply voltage is restored, it is safe to enable the BAT_LOW interrupt again by calling the *tat_enable_bat_low_isr* function.

11.37.3 Effect on receipt

When calling the *tat_enable_bat_low_isr* the BAT_LOW bit in IRQ_MASK register will be set, and thus enabling the interrupt.

11.38 Clear TRX_UR Flag

11.38.1 Semantics of the function

```
void tat_clear_trx_ur_flag_ref( void )
```

11.38.2 Appropriate usage

The *tat_clear_trx_ur_flag_ref* function is used to disable the user interface for the TRX_UR interrupt.

11.38.3 Effect on receipt

When calling the *tat_clear_trx_ur_flag_ref* the TRX_UR bit in the `tat_isr_flag_mask` will be cleared. The result is that an associated interrupt flag will not be incremented when a TRX_UR interrupt is signaled.

11.39 Set TRX_UR Flag

11.39.1 Semantics of the function

```
void tat_set_trx_ur_flag_ref( uint8_t volatile *flag )
```

Table 11-44. tat_set_trx_ur_flag_ref Parameters

Name	Type	Valid Range	Description
flag	uint8_t*	pointer	TRX_UR Flag.

11.39.2 Appropriate usage

The *tat_set_trx_ur_flag_ref* function is used to register an interrupt flag to the TRX_UR interrupt and enable the associated interrupt user interface.

11.39.3 Effect on receipt

When calling the *tat_set_bat_low_flag_ref* the BAT_LOW bit in the `tat_isr_flag_mask` will be set. The result is that supplied interrupt flag will be incremented when a BAT_LOW interrupt is signaled.

11.40 Clear TRX_END Flag

11.40.1 Semantics of the function

```
void tat_clear_trx_end_flag_ref( void )
```

11.40.2 Appropriate usage

The *tat_clear_trx_end_flag_ref* function is used to disable the user interface for the TRX_END interrupt.

11.40.3 Effect on receipt

When calling the *tat_clear_trx_end_flag_ref* the TRX_END bit in the `tat_isr_flag_mask` will be cleared. The result is that an associated interrupt flag will not be incremented when a TRX_END interrupt is signaled.

11.41 Set TRX_END Flag

11.41.1 Semantics of the function

```
void tat_set_trx_end_flag_ref( uint8_t volatile *flag )
```

Table 11-45. tat_set_trx_end_flag_ref Parameters

Name	Type	Valid Range	Description
flag	uint8_t*	pointer	TRX_END Flag.

11.41.2 Appropriate usage

The *tat_set_trx_end_flag_ref* function is used to register an interrupt flag to the TRX_END interrupt and enable the associated interrupt user interface.

11.41.3 Effect on receipt

When calling the *tat_set_trx_end_flag_ref* the TRX_END bit in the `tat_isr_flag_mask` will be set. The result is that supplied interrupt flag will be incremented when a TRX_END interrupt is signaled.

11.42 Clear RX_ON Flag

11.42.1 Semantics of the function

```
void tat_clear_rx_start_flag_ref( void )
```

11.42.2 Appropriate usage

The *tat_clear_rx_start_flag_ref* function is used to disable the user interface for the RX_START interrupt.

11.42.3 Effect on receipt

When calling the *tat_clear_rx_start_flag_ref* the RX_START bit in the `tat_isr_flag_mask` will be cleared. The result is that an associated interrupt flag will not be incremented when a RX_START interrupt is signaled.

11.43 Set RX_ON Data Buffer

11.43.1 Semantics of the function

```
void tat_set_rx_buffer( uint8_t *rx_buffer )
```

Table 11-46. tat_set_rx_buffer Parameters

Name	Type	Valid Range	Description
rx_buffer	uint8_t*	pointer	New receive data buffer.

11.43.2 Appropriate usage

The *tat_set_rx_buffer* function is used to update the receive buffer currently used by the TAT.

11.43.3 Effect on receipt

When calling the *tat_set_rx_buffer* the `rx_data` variable will be set to the memory location pointed to by `rx_buffer`.

11.44 Set RX_ON Flag

11.44.1 Semantics of the function

```
void tat_set_rx_start_flag_ref( uint8_t volatile *flag, uint8_t
*rx_buffer )
```

Table 11-47. tat_set_rx_start_flag_ref Parameters

Name	Type	Valid Range	Description
flag	uint8_t*	pointer	RX_ON Flag.
rx_buffer	uint8_t*	pointer	Receive Buffer.

11.44.2 Appropriate usage

The *tat_set_rx_start_flag_ref* function is used to register an interrupt flag to the RX_START interrupt and enable the associated interrupt user interface.

11.44.3 Effect on receipt

When calling the *tat_set_rx_start_flag_ref* the RX_START bit in the `tat_isr_flag_mask` will be set. The result is that supplied interrupt flag will be incremented when a RX_START interrupt is signaled.

11.45 Clear PLL_UNLOCK Flag

11.45.1 Semantics of the function

```
void tat_clear_pll_unlock_flag_ref( void )
```

11.45.2 Appropriate usage

The *tat_clear_pll_unlock_flag_ref* function is used to disable the user interface for the PLL_UNLOCK interrupt.

11.45.3 Effect on receipt

When calling the *tat_clear_pll_unlock_flag_ref* the PLL_UNLOCK bit in the `tat_isr_flag_mask` will be cleared. The result is that an associated interrupt flag will not be incremented when a PLL_UNLOCK interrupt is signaled.

11.46 Set PLL_UNLOCK Flag

11.46.1 Semantics of the function

```
void tat_set_pll_unlock_flag_ref( uint8_t volatile *flag )
```

Table 11-48. tat_set_pll_unlock_flag_ref Parameters

Name	Type	Valid Range	Description
flag	uint8_t*	pointer	PLL_UNLOCK Flag.

11.46.2 Appropriate usage

The *tat_set_pll_unlock_flag_ref* function is used to register an interrupt flag to the PLL_UNLOCK interrupt and enable the associated interrupt user interface.

11.46.3 Effect on receipt

When calling the *tat_set_pll_unlock_flag_ref* the PLL_UNLOCK bit in the `tat_isr_flag_mask` will be set. The result is that supplied interrupt flag will be incremented when a PLL_UNLOCK interrupt is signaled.

11.47 Clear PLL_LOCK Flag

11.47.1 Semantics of the function

```
void tat_clear_pll_lock_flag_ref( void )
```

11.47.2 Appropriate usage

The *tat_clear_pll_lock_flag_ref* function is used to disable the user interface for the PLL_LOCK interrupt.

11.47.3 Effect on receipt

When calling the *tat_clear_pll_lock_flag_ref* the PLL_LOCK bit in the `tat_isr_flag_mask` will be cleared. The result is that an associated interrupt flag will not be incremented when a PLL_LOCK interrupt is signaled.

11.48 Set PLL_LOCK Flag

11.48.1 Semantics of the function

```
void tat_set_pll_lock_flag_ref( uint8_t volatile *flag )
```

Table 11-49. tat_set_pll_lock_flag_ref Parameters

Name	Type	Valid Range	Description
flag	uint8_t*	pointer	PLL_LOCK Flag.

11.48.2 Appropriate usage

The *tat_set_pll_lock_flag_ref* function is used to register an interrupt flag to the PLL_LOCK interrupt and enable the associated interrupt user interface.

11.48.3 Effect on receipt

When calling the *tat_set_pll_lock_flag_ref* the PLL_LOCK bit in the `tat_isr_flag_mask` will be set. The result is that supplied interrupt flag will be incremented when a PLL_LOCK interrupt is signaled.

11.49 HAL Interrupt Handler

11.49.1 Semantics of the function

```
void hal_user_interrupt_handler( uint8_t source, uint32_t timestamp
)
```

Table 11-50. hal_user_interrupt_handler Parameters

Name	Type	Valid Range	Description
source	uint8_t	0 to 0xFF	Interrupt Source.
timestamp	uint32_t	0 to 2^{32}-1	Interrupt Timestamp.

11.49.2 Appropriate usage

The *hal_user_interrupt_handler* function is called from the HAL level interrupt handler when a valid interrupt is signaled from the radio transceiver. This function should not be called by the end-user under any circumstance.

11.49.3 Effect on receipt

When calling the *hal_user_interrupt_handler* the `source` parameter will be used to branch the code execution for each supported interrupt:

- **BAT_LOW:** Disable the BAT_LOW interrupt to prevent an interrupt storm. The associated user flag is incremented if enabled. The interrupt can be enabled again when by calling the tat_enable_bat_low_isr function.
- **TRX_UR:** The associated user flag is incremented if enabled.
- **TRX_END:** The associated user flag is incremented if enabled.

- **RX_START:** Length of the received frame is read and the delay necessary to prevent the frame buffer from being under-run is calculated. The received frame will be uploaded after the calculated delay. The associated user flag is incremented if enabled.
- **PLL_UNLOCK:** The associated user flag is incremented if enabled.
- **PLL_LOCK:** The associated user flag is incremented if enabled.

12 Table of Contents

Features ***1***

1 Introduction ***1***

2 Intended Audience and Overview ***2***

3 Programming Sequence: Transceiver Setup ***2***

3.1 Radio Transceiver Initialization 2

3.2 Hardware Reset 3

3.3 Reset State Machine 4

3.4 Get Current Channel 4

3.5 Set Current Channel 5

3.6 Get Transmit Power 5

3.7 Set Transmit Power 5

4 Programming Sequence: State Transitions ***5***

4.1 Transitions to TRX_OFF 6

4.2 Transitions to RX_ON 7

4.3 Transitions to PLL_ON 8

4.4 Transitions to RX_AACK_ON 9

4.5 Transitions to TX_ARET_ON 11

4.6 Transitions to SLEEP 13

4.7 Short Summary 14

5 Programming Sequence: CCA, ED, LQI and RSSI Measurements ***14***

5.1 Clear Channel Assessment 15

5.1.1 Setup and Configuration 15

5.1.2 Manual Clear Channel Assessment 15

5.2 Energy Detection Measurement 16

5.2.1 Manual Energy Detection Measurement 16

5.2.2 Energy Detection Measurement During Frame Reception 16

5.2.3 How to Interpret the Energy Detection Value 17

5.3 Link Quality Indication Measurement 17

5.4 Received Signal Strength Indication Measurement 17

5.4.1 Get RSSI 18

5.4.2 Get Frame RSSI 18

6 Programming Sequence: Frame Transaction 19

6.1 Frame Transmission with Pin Start 19

6.2 Frame Transmission with Register Start 20

6.3 Basic Frame Reception 21

6.4 Advanced Frame Reception 21

7 Programming Sequence: Setup of Extended Features 23

7.1 Auto Generated CRC for Transmit Frames 23

7.2 Automated CSMA Algorithm (TX_AUTO_CSMA) 24

7.3 Address Filter Setup (RX_AACK) 25

8 Programming Sequence: Extended Frame Transmission and Reception 26

8.1 Automated CSMA Algorithm and Frame Transmission (TX_ARET_ON) 26

8.2 Automated Frame Reception and Acknowledge (RX_AACK) 28

9 The Transceiver Access Toolbox 29

10 Abbreviations and Definitions 33

11 Appendix A: Transceiver Access Toolbox API 34

11.1 Initialize Transceiver Access Toolbox 34
11.1.1 Semantics of the function 34
11.1.2 Appropriate usage 34
11.1.3 Effect on receipt 34

11.2 Reset Radio Transceiver 36
11.2.1 Semantics of the function 48
11.2.2 Appropriate usage 49
11.2.3 Effect on receipt 49

11.3 Reset State Machine 48
11.3.1 Semantics of the function 48
11.3.2 Appropriate usage 48
11.3.3 Effect on receipt 48

11.4 Get Current Channel 34
11.4.1 Semantics of the function 34
11.4.2 Appropriate usage 35
11.4.3 Effect on receipt 35

11.5 Set Current Channel 35

11.6 Get Transmit Power Level 36
11.6.1 Semantics of the function 36
11.6.2 Appropriate usage 36
11.6.3 Effect on receipt 36

11.7 Set Transmit Power Level 36
11.7.1 Semantics of the function 36
11.7.2 Appropriate usage 36
11.7.3 Effect on receipt 36

11.8 Set CCA Mode 38
11.8.1 Semantics of the function 38
11.8.2 Appropriate usage 38
11.8.3 Effect on receipt 38

11.9 Do Clear Channel Assessment 39
11.9.1 Semantics of the function 39
11.9.2 Appropriate usage 39
11.9.3 Effect on receipt 39

11.10 Detect Energy on Antenna Pins 37
11.10.1 Semantics of the function 37
11.10.2 Appropriate usage 37
11.10.3 Effect on receipt 37

11.11 Get Received Signal Strength Indication 40
11.11.1 Semantics of the function 40
11.11.2 Appropriate usage 40
11.11.3 Effect on receipt 40

11.12 Enable or Disable Automatic CRC 49
11.12.1 Semantics of the function 49
11.12.2 Appropriate usage 49
11.12.3 Effect on receipt 49

11.13 Configure the CSMA Algorithm 52
11.13.1 Semantics of the function 52
11.13.2 Appropriate usage 53
11.13.3 Effect on receipt 53

12 Table of Contents 59

Disclaimer 62

Headquarters

Atmel Corporation
2325 Orchard Parkway
San Jose, CA 95131
USA
Tel: 1(408) 441-0311
Fax: 1(408) 487-2600

International

Atmel Asia
Room 1219
Chinachem Golden Plaza
77 Mody Road Tsimshatsui
East Kowloon
Hong Kong
Tel: (852) 2721-9778
Fax: (852) 2722-1369

Atmel Europe
Le Krebs
8, Rue Jean-Pierre Timbaud
BP 309
78054 Saint-Quentin-en-Yvelines Cedex
France
Tel: (33) 1-30-60-70-00
Fax: (33) 1-30-60-71-11

Atmel Japan
9F, Tonetsu Shinkawa Bldg.
1-24-8 Shinkawa
Chuo-ku, Tokyo 104-0033
Japan
Tel: (81) 3-3523-3551
Fax: (81) 3-3523-7581

Product Contact

Web Site
www.atmel.com

Technical Support
avr@atmel.com

Sales Contact
www.atmel.com/contacts

Literature Request
www.atmel.com/literature