23.1 RF Core

The RF core contains an ARM Cortex-M0 processor that interfaces the analog RF and baseband circuitries, handles data to and from the system side, and assembles the information bits in a given packet structure. The RF core offers a high-level, command-based application program interface (API) to the system CPU (ARM® Cortex®-M3). The RF core can autonomously handle the time-critical aspects of the radio protocols (802.15.4 RF4CE and ZigBee®, Bluetooth® low energy, and so on), thus offloading the system CPU and leaving more resources for the user's application.

The RF core has a dedicated 4-KB SRAM block and runs almost entirely from separate ROM.

23.1.1 High-Level Description and Overview

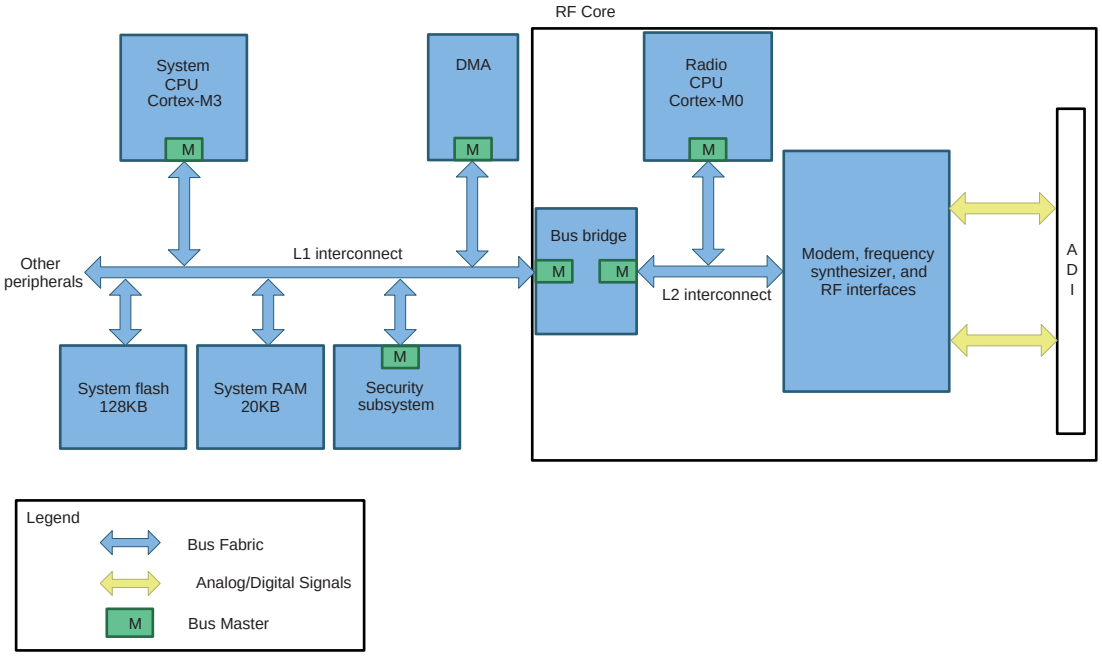
The RF core receives high-level requests from the system CPU and performs all the necessary transactions to fulfill them. These requests are primarily oriented to the transmission and reception of information through the radio channel, but can also include additional maintenance tasks such as calibration, test, or debug features.

As a general framework, the transactions between the system CPU and the RF core operate as follows:

- The RF core can access data and configuration parameters from the system RAM. This access reduces the memory requirements of the RF core, avoids needless traffic between the different parts of the system, and reduces the total energy consumption.
- In a similar fashion, the RF core can decode and write back the contents of the received radio packet, together with status information, to the system RAM.
- For protocol confidentiality and authentication support purposes, the RF core can also access the security subsystem.
- In general, the RF core recognizes complex commands from the system CPU (CCA transmissions, RX with automatic acknowledge, and so forth) and divides them into subcommands without further intervention of the system CPU.

Figure 23-1 shows the external interfaces and dependencies of the RF core.

Figure 23-1. Limited RF Core Overview With External Dependencies



Each block in Figure 23-1 performs the following functions:

System Side

- System CPU: Main system processor that runs the user's application, together with the high-level protocol stack (for a number of supported configurations) and eventually some higher-level MAC features for some protocols. The system CPU runs code from the boot ROM and the system flash.
- System RAM: Contains packet information (TX and RX payloads) and the different parameters or configuration options for a given transaction.
- Security Subsystem: Encompasses the different elements to provide protocol confidentiality and authentication.
- DMA: Optionally charged with the task of moving information from the radio RAM to the system RAM and vice versa, if direct CPU access is not used.

Radio Side

- Radio CPU: Main RF core processor. Receives high-level commands from the system CPU and schedules them into the different parts of the RF core.
- Modem, Frequency Synthesizer, RF Interfaces: This is the core of the radio, converting the bits into modulated signals and vice versa.

23.2 Radio Doorbell

The radio doorbell module (RFC_DBELL) is the primary means of communication between the system CPU and the radio CPU, also known as command and packet engine (CPE). The radio doorbell contains a set of dedicated registers, parameters in any of the RAMs of the device, and a set of interrupts to both the radio CPU and the system CPU.

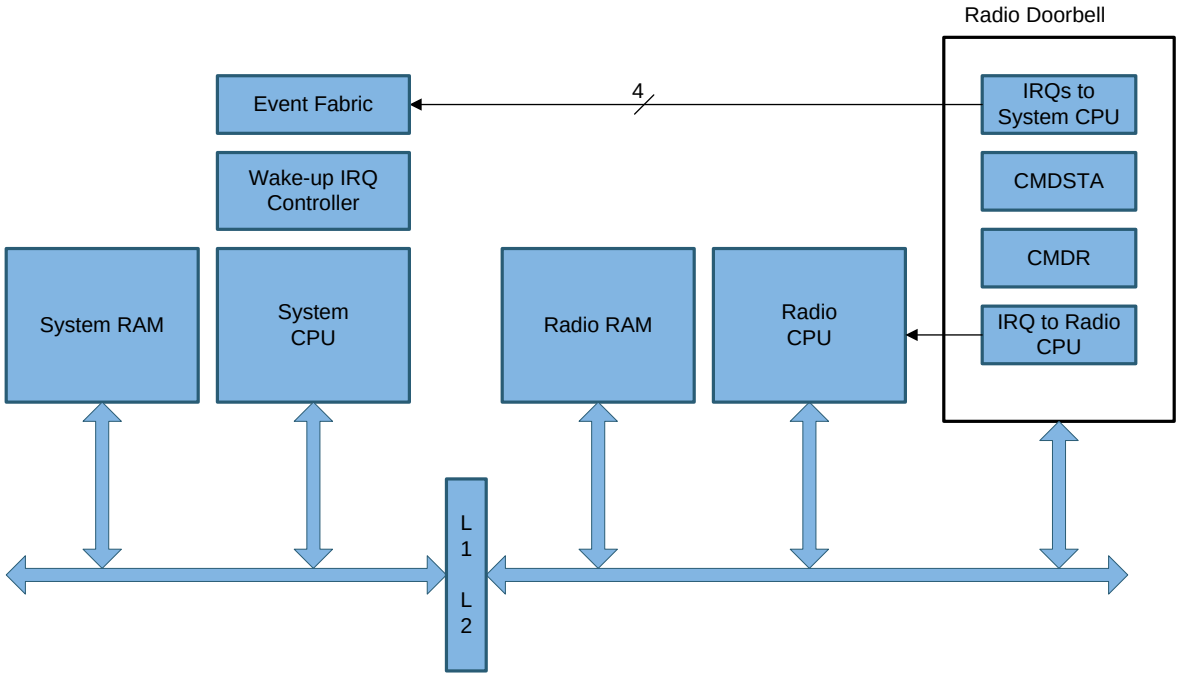
In addition, parameters and payload are transferred through the system RAM or the radio RAM. If any parameters or payload are in the system RAM, the system CPU must remain powered, while if everything is in the radio RAM, the system CPU may go into power-down mode to save current.

During operation, the radio CPU updates parameters and payload in RAM and raises interrupts. The system CPU may mask out interrupts, so that it remains in idle or power-down mode until the entire radio operation finishes.

Because the system CPU and the radio CPU share a common RAM area, ensure that no contention or race conditions can occur. This is achieved in software by rules set up in the radio hardware abstraction layer (HAL).

Figure 23-2 shows the relevant modules for information exchange between the CPUs.

Figure 23-2. Hardware Support for the HAL



23.2.1 Command and Status Register and Events

Commands are sent to the radio through the CMDR register, while the CMDSTA read-only register provides status back from the radio. The CMDR register can only be written while it reads 0; otherwise, writes are ignored. When the CMDR register is 0 and a nonzero value is written to it, the radio CPU is notified and the CMDSTA register becomes 0. After this, the value written is readable from the CMDR register until the radio CPU has processed the command, at which point it goes back to 0.

When the command has been processed by the radio CPU, the CMDSTA register contains a nonzero status, which is provided when the CMDR register goes back to 0. At the same time, an RFCMDACK interrupt occurs. This interrupt is also mapped to the RFACKIFG register, which should be cleared when the interrupt has been processed.

See Section 23.3.2 for the format of the command and status registers.

23.2.2 RF Core Interrupts

The RF core has four interrupt lines to the ARM Cortex-M3 (see Figure 23-2). The following interrupts are controlled by the radio doorbell module:

- RF_CPE0 (interrupt number 9)
- RF_CPE1 (interrupt number 2)
- RF_HW (interrupt number 10)
- RF_CMD_ACK (interrupt number 11)

23.2.2.1 RF Command and Packet Engine Interrupts

The two system-level interrupts RF_CPE0 and RF_CPE1 can be produced from a number of low-level interrupts produced by the CPE. Each of these low-level interrupts can be mapped to RF_CPE0 or RF_CPE1 using the RFCPEISL register. In addition, interrupt generation at system level may be switched on and off using the RFCPEIEN register.

In case of an event that triggers a low-level interrupt, the corresponding bit in the RFCPEIFG register is set to 1. Whenever a bit in RFCPEIFG and the corresponding bit in RFCPEIEN are both 1, the system-level interrupt selected in RFCPEISL is raised. This means that the interrupt service routine (ISR) must clear the bits in RFCPEIFG that correspond to low-level interrupts that have been processed.

A list of the available interrupts is found in the register description for RFCPEIFG in Section 23.8.2.5.

Clearing bits in RFCPEIFG is done by writing 0 to those bits, while any bits written to 1 remain unchanged.

NOTE: When clearing bits in the RFCPEIFG register, interrupts may be lost if a read-modify-write operation is done because interrupt flags that became active between the read and write operation might be lost. Thus, clearing an interrupt flag should be done as follows:

```
HWREG(RFC_DBELL_BASE + RFC_DBELL_O_RFCPEIFG) = ~(1 << irq_no);
```

and not as:

```
HWREG(RFC_DBELL_BASE + RFC_DBELL_O_RFCPEIFG) &= ~(1 <<
irq_no); // wrong
```

23.2.2.2 RF Core Hardware Interrupts

The system-level interrupt RF_HW can be produced from a number of low-level interrupts produced by RF core hardware. Interrupt generation at the system level may be switched on and off for each source by using the RFHWEN register.

In the case of an event that triggers a low-level interrupt, the corresponding bit in the RFHWIFG register is set to 1. Whenever a bit in RFHWIFG and the corresponding bit in RFHWIEN are both 1, the RF_HW interrupt is raised. This means that the ISR should clear the bits in RFHWIFG that correspond to low-level interrupts that have been processed.

A list of the available interrupts is found in the register description for RFHWIFG in Section 23.8.2.3. In general, TI does not recommend servicing these interrupts in the main CPU, but the available radio timer channel interrupts may be served this way.

Clearing bits in RFHWIFG is done by writing 0 to those bits, while any bits written to 1 remain unchanged.

NOTE: When clearing bits in RFHWIFG, interrupts may be lost if a read-modify-write operation is done. Therefore, the same rule applies for the RFHWIFG register as for RFCPEIFG (see Section 23.2.2.1).

23.2.2.3 RF Core Command Acknowledge Interrupt

The system-level interrupt RF_CMD_ACK is produced when an RF core command is acknowledged (that is, when the status becomes available in CMDSTA [see Section 23.8.2.2]). When the status becomes available, the RFACKIFG.ACKFLAG register bit is set to 1. Whenever this bit is 1, the RF_CMD_ACK interrupt is raised, which means that the ISR must clear RFACKIFG.ACKFLAG when processing the RF_CMD_ACK interrupt.

23.2.3 Radio Timer

The radio has its own dedicated timer, the radio timer (RAT) module. The RAT is a 32-bit free-running timer running on 4 MHz. The RAT has 8 channels with compare and capture functionality. Five of these channels are reserved for the radio CPU, while the remaining three channels are available for use by the ARM Cortex-M3. The available channels are numbered 5, 6, and 7.

The RAT can only run while the RF core is powered up. The RAT must be started by the command CMD_START_RAT or CMD_SYNC_START_RAT. The RAT must be running to execute a radio operation command with delayed start or any radio operation command that runs the receiver or transmitter.

When the RAT is running, the current value of the timer can be read from the RATCNT register (see Section 23.8.1.1).

23.2.3.1 Compare and Capture Events

The available channels may be set up in compare mode or capture mode.

Compare mode can be set up using the CMD_SET_RAT_CMP command (see Section 23.3.3.2.10). In this case, the timer generates an interrupt when the counter reaches the value given by compareTime. The interrupt is mapped to RFHWIFG (see Section 23.2.2.2 and Section 23.8.2.3). For the available RAT channels, the interrupt flags in use are RATCH5, RATCH6, and RATCH7. Optionally, it is also possible to control an I/O pin when the counter reaches the value given by compareTime (see Section 23.2.3.2). When the CMD_SET_RAT_CMP command has been sent, the value of compareTime is stored in the radio channel value register (RATCHnVAL) corresponding to the selected channel (see Table 23-154).

Capture mode can be used to capture a transition on an input pin and record the value of the RAT counter at the time when the transition occurred. Compare mode can be set up using the CMD_SET_RAT_CPT command (see Section 23.3.3.2.11). When the transition occurs, the current value of the RAT is stored in the RATCHnVAL register corresponding to the selected channel (see Table 23-154), and the timer generates an interrupt. As for compare mode, the interrupt is mapped to RFHWIFG. For the available RAT channels, the interrupt flags in use are RATCH5, RATCH6, and RATCH7. If single-capture mode is configured in CMD_SET_CPT, only the first transition is captured, unless the channel is armed again, as explained in the following paragraph. If repeated mode is configured, every transition is captured.

NOTE: In this case, the captured value in the RATCHnVAL register may be overwritten at any time if a new transition occurs.

A channel set up in compare mode or single capture mode may be armed or disarmed. When CMD_SET_RAT_CMP or CMD_SET_RAT_CPT is sent, the channel is armed automatically, and when the capture or compare event occurs, the channel is disarmed automatically. A disarmed channel does not produce any interrupt or cause any timer value to be captured. In addition, a channel may be armed or disarmed using CMD_ARM_RAT_CH or CMD_DISARM_RAT_CH (see Section 23.3.3.2.14 to Section 23.3.3.2.15). While disarmed, the channel keeps its configuration. To disable a channel that is not going to be re-armed with the same configuration, the CMD_DISABLE_RAT_CH command may be used (see Section 23.3.3.2.12).

23.2.3.2 Radio Timer Outputs

The RAT module has four controllable outputs, RAT_GPO0 to RAT_GPO3. These signals may be controlled by one of the RAT channels and mapped to signals available for the IOC using the SYSGPOCTL register (see Section 23.8.2.9). RAT_GPO0 is reserved for starting the transmitter and is controlled internally by the radio CPU (see Section 23.3.2.8). The other three signals may be configured using the CMD_SET_RAT_OUTPUT command (see Section 23.3.3.2.11). The different output modes decide the transition of the output when an interrupt occurs on the chosen RAT channel except for the always-0 and always-1 configurations, which take effect immediately and may be used for initialization.

23.2.3.3 Synchronization With Real-Time Clock

When the radio is powered down, the RAT module is not counting. To keep a consistent time base over time for synchronized protocols, it is possible to synchronize the RAT with the real-time clock (RTC) (see Chapter 14).

To allow synchronization after power up, the CMD_SYNC_STOP_RAT command (see Section 23.3.3.1.10) must be sent before the RF core is powered down. This command (until the next RTC tick) stops the RAT and returns a parameter rat0, and should be stored and provided when the RAT is restarted.

The next time the RF core is powered up and the RAT is started, this synchronization must be done using CMD_SYNC_START_RAT (see Section 23.3.3.1.11), where the rat0 parameter obtained from CMD_SYNC_STOP_RAT must be provided. This command starts the RAT, waits for an RTC tick, and adjusts the RAT. Depending on the application, it may not be necessary to run the CMD_SYNC_STOP_RAT command every time the radio is powered down; a previous value of rat0 may be reused. In some cases, however, this may cause issues if the radio has been powered for a long time and the low-frequency and high-frequency crystal oscillators have a significant error relative to each other.

To get accurate synchronization, it is important that the system is running on the high-frequency crystal oscillator starting before the CMD_SYNC_START_RAT command is run and extending beyond completion of the CMD_SYNC_STOP_RAT command.

NOTE: For the CMD_SYNC_START_RAT and CMD_SYNC_STOP_RAT commands, the AON_RTC:CTL RTC_UPD_EN register bit must be set to 1 (see Section 14.4.1.1). It is never necessary to reset this bit to 0; it may be set permanently to 1 when the RTC is started.

23.3 RF Core HAL

The RF core hides the complexity of the radio operations by providing a unified HAL to the system CPU.

NOTE: To ensure optimum radio performance always use the latest radio patches provided by TI. See the product pages on www.ti.com for the latest patches.

23.3.1 Hardware Support

The radio HAL is supported by hardware, by means of the radio doorbell module in the RF core area and command descriptors in the system RAM.

23.3.2 Firmware Support

The RF core accepts a set of high-level primitives. The following sections describe the desired functionality at a high level.

23.3.2.1 Commands

The radio CPU lets the user run a set of high-level primitives or commands from the system CPU. After a command has been issued through the CMDR register, the radio CPU examines it and decides a course of action.

Three classes of commands are issued:

- Radio operation command
- Immediate command
- Direct command

For the first two classes of commands, CMDR contains a pointer to a command structure. This pointer must be a valid pointer with 32-bit word alignment, so the 2 least significant bits (LSBs) must be 0 0, as shown in Figure 23-3. A direct command is signaled by setting the 2 LSBs to 01 and placing the command ID number in bits 16 to 31 of CMDR. Bits 8 through 15, or alternatively 2 through 15, may be used as an optional byte parameter. Figure 23-4 shows the format for a direct command.

Figure 23-3. CMDR Register for Radio Operation Commands and Immediate Commands

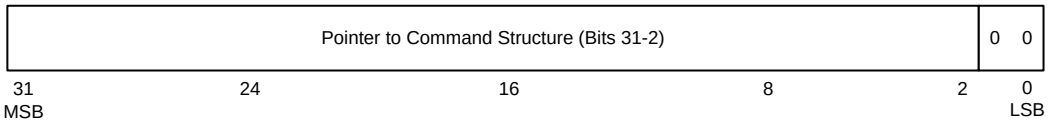
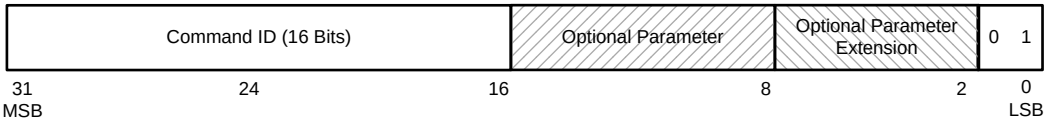


Figure 23-4. CMDR Register for Direct Commands



The data structure pointed to by the CMDR register for radio operation and immediate commands may be in the system RAM or the radio RAM. The system CPU must ensure that the memory area in use is free for access, in particular when using the radio RAM, where a part of the memory is reserved for use by the radio CPU. This information may be obtained with the CMD_GET_FW_INFO command (see Section 23.3.3.2.6). The format of the command follows the structure given in Section 23.3.2.6 and its subsection, and are defined in more detail specifically for each command.

When deciding in which memory area to place data, consider which modules may be powered down:

- The radio RAM is accessible for the radio CPU at any time, but does not have retention when the radio is powered down. Data that must be retained must therefore be copied into or out of the radio RAM whenever the radio is powered up or down, respectively.
- The system RAM has retention in most low-power modes. If the system side is powered down, the radio CPU requests that it is powered up again to access the RAM. The active current consumption from the radio CPU accessing the system RAM is higher than the current consumption from accessing the radio RAM, especially if the system side could otherwise have been powered down.
- The flash always has retention, and can only store parameters that are not written by the radio CPU. As with accessing the system RAM, the radio CPU must ensure that the system side is powered up to access the flash. The power consumption from the radio CPU accessing the flash is higher than the current consumption from accessing the system RAM, but in most cases the difference is negligible due to few accesses.
- The lowest peak-power consumption is obtained by putting all data structures in the radio RAM and powering down the system side while the radio CPU is running. In some cases, the average power consumption may be lower by putting data structures in the system RAM, because less copying is then needed, and the system side can still be powered down for long periods (for instance, while the receiver is in sync search).

A radio operation command causes the radio hardware to be accessed. Radio operation commands can do operations such as transmitting or receiving a packet, setting up radio hardware registers, or doing more complex, protocol-dependent operations. A radio operation command can normally be issued only while the radio is idle.

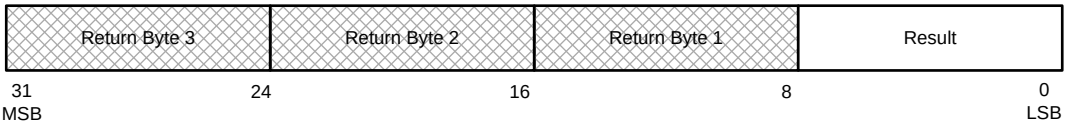
An immediate command is a command to change or request status of the radio, or to manipulate TX or RX data queues. An intermediate command can monitor status such as received signal strength. An immediate command can be issued at any time, but the response is, in many cases, only of interest while a radio operation is ongoing.

A direct command is an immediate command with no parameters, or in some cases, a direct command has 1- or 2-byte parameters. A direct command is issued by sending a value to the CMDR register with the format of Figure 23-4. The 16 most significant bits (MSBs) contain the command ID of the immediate command to run. Bits 8 through 15 or 2 through 15 may contain an optional parameter if specified for the command.

23.3.2.2 Command Status

After a command is issued, the CMDSTA register is updated by the radio CPU, causing an RFCMDACK interrupt to be sent back to the system CPU. This update occurs after the command finishes for immediate and direct commands and after the command is scheduled for radio operation commands. No new command may be issued until this interrupt is received. The CMDSTA register consists of 32 bits; the 8 LSBs give the result, while the upper 24 bits may be used for specific signaling in each command. Figure 23-5 shows this format.

Figure 23-5. Format of CMDSTA Register



In the result byte, bit 7 indicates whether an error occurred or not. The result byte of 0x00, meaning *pending*, is produced automatically by the radio doorbell hardware when a command is issued, and the other bits in the CMDSTA register also become 0, which is the value of CMDSTA before the RF_CMD_ACK interrupt is raised.

Table 23-1 lists the values of the result byte in the CMDSTA register.

Table 23-1. Values of the Result Byte in the CMDSTA Register

Value	Name	Description
No Error		
0x00	Pending	The command has not been parsed.
0x01	Done	Immediate command: The command finished successfully. Radio operation command: The command was successfully submitted for execution.
Error		
0x81	IllegalPointer	The pointer signaled in CMDR is not valid.
0x82	UnknownCommand	The command ID number in the command structure is unknown.
0x83	UnknownDirCommand	The command number for a direct command is unknown, or the command is not a direct command.
0x85	ContextError	An immediate or direct command was issued in a context where it is not supported.
0x86	SchedulingError	A radio operation command was attempted to be scheduled while another operation was already running in the RF core. The new command is rejected, while the command already running is not impacted.
0x87	ParError	There were errors in the command parameters that are parsed on submission. For radio operation commands, errors in parameters parsed after start of the command are signaled by the command ending, and an error is indicated in the status field of that command structure.
0x88	QueueError	An operation on a data entry queue was attempted, but the operation was not supported by the queue in its current state.
0x89	QueueBusy	An operation on a data entry was attempted while that entry was busy.

In addition to the command status register, each radio operation command contains a status field (see Table 23-8). This field may have values in the following categories.

- Idle: The command has not started.
- Pending: The command has been parsed, but the start trigger has not occurred.
- Active: The command is running.
- Suspended: The command has been active, and may become active again. The command is supported only by certain IEEE 802.15.4 commands.
- Finished: The command is finished, and the system CPU is free to modify the command structure or free memory.
- Skipped: The command was skipped and never executed.

For common commands and when parsing any command before starting, refer to the status codes listed in Table 23-2.

Table 23-2. Common Radio Operation Status Codes

Number	Name	Description
Operation Not Finished		
0x0000	IDLE	Operation has not started.
0x0001	PENDING	Waiting for a start trigger.
0x0002	ACTIVE	Running an operation.
0x0003	SKIPPED	Operation skipped due to condition in another command.
Operation Finished Normally		
0x0400	DONE_OK	Operation ended normally.
0x0401	DONE_COUNTDOWN	Counter reached zero.
0x0402	DONE_RXERR	Operation ended with CRC error.
0x0403	DONE_TIMEOUT	Operation ended with time-out.
0x0404	DONE_STOPPED	Operation stopped after CMD_STOP command.
0x0405	DONE_ABORT	Operation aborted by CMD_ABORT command.
Operation Finished With Error		
0x0800	ERROR_PAST_START	The start trigger occurred in the past.
0x0801	ERROR_START_TRIG	Illegal start trigger parameter
0x0802	ERROR_CONDITION	Illegal condition for next operation
0x0803	ERROR_PAR	Error in a command specific parameter
0x0804	ERROR_POINTER	Invalid pointer to next operation
0x0805	ERROR_CMDID	The next operation has a command ID that is undefined or not a radio operation command.
0x0807	ERROR_NO_SETUP	Operation using RX, TX, or synthesizer attempted without CMD_RADIO_SETUP.
0x0808	ERROR_NO_FS	Operation using RX or TX attempted without the synthesizer being programmed or powered on.
0x0809	ERROR_SYNTH_PROG	Synthesizer programming failed.
0x080A	ERROR_TXUNF	Modem TX underflow observed.
0x080B	ERROR_RXOVF	Modem RX overflow observed.
0x080C	ERROR_NO_RX	Data requested from last RX when no such data exists.

When the system CPU prepares a command structure, the CPU should initialize the status field to Idle. Commands may be set up in a loop. If so, the system CPU must not modify command structures until the radio CPU becomes idle (the system CPU receives a LAST_COMMAND_DONE interrupt, even if the status is finished or skipped [see Section 23.8.2.5]).

23.3.2.3 Interrupts

The radio CPU has 32 software interrupt sources that generate the RFCPE0 and RFCPE1 interrupts in the system CPU. An interrupt flag register can indicate which software interrupt has been raised, and the interrupts are enabled individually. In addition, the RFCMDACK interrupt is raised automatically when CMDSTA is updated.

Some software-defined interrupts have a common meaning across all commands; the details of each of the other interrupts are defined for each protocol that uses a particular interrupt. Some interrupts are used in only one protocol, while others are used in several protocols. The interrupts are listed in the description of the RFCPEIFG register (see Table 23-171).

23.3.2.4 Passing Data

There are two basic ways to pass data transmitted or received over the air: directly or through a queue.

The most straight-forward way to pass data is to append it as part of the command parameters (directly or through a pointer). The exact format depends on the command being run; normally there is a length field and a data buffer for TX and a maximum length, received length (if variable length), and receive buffer for RX.

For some operations, the number of packets received or transmitted (or which packet out of a few that will actually be transmitted) cannot be known in advance. For such operations, use a concept of queues. An operation can use one or more queues, for instance one RX and one TX queue for a combined RX/TX operation, or several queues depending on information in the received packets. Any operation using queues uses a common system for maintaining them, as explained in Section 23.3.2.7.

A radio operation command declares which data method is used.

23.3.2.5 Command Scheduling

The system CPU is responsible for scheduling the commands as required. When using low-power modes, the system CPU must wake up a short time before the start of the next operation, using the RTC.

A radio operation command can be scheduled with a delayed start (see Section 23.3.2.5.1). If a command is started with a delay, the radio CPU goes to idle mode until the command starts. The radio operation command is considered to be running during this delay, and no other radio operation command can be scheduled unless the pending command is first aborted or stopped.

The system CPU can schedule back-to-back radio operation commands by using the next operation pointer in any radio operation command. This pointer can point to the next command to perform in the chain, and by this method, complex operations can be made. Under some conditions (such as an error or the expiration of a timer), the next command is not started. Instead, the operation ends or a number of commands may be skipped (see Section 23.3.2.5.2). If a new command is scheduled while another command is running, the system CPU must wait for the previous command or chain of commands to finish. The IEEE 802.15.4 commands have exceptions for this rule.

When a radio operation command is finished, the radio CPU raises a COMMAND_DONE interrupt to the system CPU. If a number of commands are chained as explained previously, the COMMAND_DONE interrupt is raised after each command, while the LAST_COMMAND_DONE interrupt is raised after the last command in the chain. For one, nonchained command, the LAST_COMMAND_DONE interrupt is also raised after the command. When LAST_COMMAND_DONE is raised, COMMAND_DONE is always raised at the same time. Before raising the COMMAND_DONE interrupt, the radio CPU updates the status field of the command structure to a status that indicates that the command is finished. The radio CPU does not access the command structure after raising the COMMAND_DONE interrupt.

23.3.2.5.1 Triggers

Triggers can be used to set up a start time, or for other specific purposes in specific radio operation commands. A common trigger byte definition exists, as defined in Table 23-3.

Table 23-3. Format of Trigger Definition Byte

Bit Index	Field	Description
0–3	triggerType	The type of trigger
4	bEnaCmd	0: No alternative trigger command. 1: CMD_TRIGGER can be used as an alternative trigger.
5–6	triggerNo	The trigger number of the CMD_TRIGGER command that triggers this action.
7	pastTrig	0: A trigger in the past is never triggered, or for start of commands, gives an error. 1: A trigger in the past is triggered as soon as possible.

Table 23-4 lists possible values for the triggerType field. Other values are reserved.

Table 23-4. Supported Trigger Types

Number	Name	Description
0	TRIG_NOW	Now (not applicable to end triggers)
1	TRIG_NEVER	Never (except possibly by CMD_TRIGGER if bEnaCmd = 1)
2	TRIG_ABSTIME	At absolute time, given by timer parameter
3	TRIG_REL_SUBMIT	At a time relative to the time the command was submitted
4	TRIG_REL_START	At a time relative to start of this command (not allowed for start triggers)
5	TRIG_REL_PREVSTART	At a time relative to the start of the previous command
6	TRIG_REL_FIRSTSTART	At a time relative to the start of the first command of the chain
7	TRIG_REL_PREVEND	At a time relative to the end of the previous command
8	TRIG_REL_EVT1	At a time relative to event 1 of the previous command
9	TRIG_REL_EVT2	At a time relative to event 2 of the previous command
10	TRIG_EXTERNAL	On an external trigger input to the RAT

A 32-bit time parameter is used together with all triggers except for TRIG_NOW and TRIG_NEVER. Absolute timing uses the value of the 32-bit RAT. Relative timing uses the number of RAT ticks. The external trigger uses an identifier of source and edge, as defined in Table 23-5.

Table 23-5. Fields of Time Parameter for External Event Trigger

Bit Index	Field	Description
0–1		Reserved
2–3	inputMode	Input mode: 00: Rising edge 01: Falling edge 10: Both edges 11: Reserved
4–7		Reserved
8–12	source	22: RFC_GPI0 23: RFC_GPI1 Others: Reserved
13–31		Reserved

Relative timing can be relative to the time of submitting the command chain, the start of the command, the start of the previous or first command, or to certain observed events inside the command, to be defined for each command. The following rules apply:

- For the first command in a chain, if the start trigger is any of the types 5 to 9, the start is immediate. If another trigger referenced in the first command in a chain is any of the types 5 to 9, the trigger time is relative to the time the command was submitted.
- If the start trigger of a command is TRIG_REL_START, an error is produced.
- If the start trigger of a command is TRIG_NEVER and bEnaCmd is 0, an error is produced.
- Some radio operation commands define events 1 and 2. These are context-dependent events that can be observed by the radio CPU. See the description of each command for a definition in that context. If undefined, these events are the time of the start of the command.

If bEnaCmd is 1, the action may also be triggered with a command (CMD_TRIGGER command, see Section 23.3.3.2.5). The triggerNo parameter identifies the trigger number of this command.

If a trigger occurs in the past when evaluated, the behavior depends on the pastTrig bit. If this bit is 0, the trigger does not occur, or for start triggers, an error is produced. If the pastTrig bit is 1, the trigger occurs as soon as possible. If the pastTrig bit is 1 for start triggers, timing relative to the start of the command is relative to the programmed start time, not the actual start time.

For an external trigger, the radio CPU sets the RAT to use the selected input event as a one-capture trigger; the CPU then uses this capture interrupt to trigger the action. If the event occurs before the setup occurs, the event is not captured, and the pastTrig bit is ignored.

23.3.2.5.2 Conditional Execution

The execution of a command may be conditional on the result of the previous command. For each command, three results are possible:

- TRUE
- FALSE
- ABORT

The criteria are defined for each command. If not defined, the result is TRUE unless the command ended with an error, in which case the result is ABORT.

Each command structure contains a condition for running the next command. The format of the condition byte is given in Table 23-6. If the rule is COND_SKIP_ON_FALSE or COND_SKIP_ON_TRUE, the number of commands to skip is signaled in the nSkip field. If the number of skips is zero, rerun the same command. If the number of skips is one, run the next command in the chain. If the number of skips is two, run the command after the next, and so forth. If the rule is COND_NEVER and no previous commands use skipping, the next command pointer is ignored and may be NULL.

Table 23-6. Format of Condition Byte

Bit Index	Field Name	Description
0–3	rule	Rule for how to proceed, as defined in Table 23-7
4–7	nSkip	Number of skips + 1 if the rule involves skipping 0: Same, 1: next, 2: skip next, ...

Table 23-7. Condition Rules

Number	Name	Description
0	COND_ALWAYS	Always run next command (except in case of ABORT).
1	COND_NEVER	Never run next command (next command pointer can still be used for skip).
2	COND_STOP_ON_FALSE	Run next command if this command returned TRUE, stop if it returned FALSE.
3	COND_STOP_ON_TRUE	Stop if this command returned TRUE, run next command if it returned FALSE.
4	COND_SKIP_ON_FALSE	Run next command if this command returned TRUE, skip a number of commands if it returned FALSE.
5	COND_SKIP_ON_TRUE	Skip a number of commands if this command returned TRUE, run next command if it returned FALSE.

If execution is stopped, the radio CPU goes back to idle and no further commands are run until a new command is entered through the CMDR register. The LAST_COMMAND_DONE interrupt is raised.

If a command ends with the ABORT result, the execution ends regardless of the condition. The LAST_COMMAND_DONE interrupt is raised. An example of criterion for the ABORT result is that a CMD_ABORT command is issued.

23.3.2.5.3 Handling Before Start of Command

For all radio operation commands, the start trigger and condition code are checked before parsing the rest of the command. If the start trigger has an illegal trigger type (including TRIG_REL_START, which is not allowed for start triggers, and TRIG_NEVER in combination with no command trigger), the radio CPU sets the status field to ERROR_START_TRIG. If the condition field has an illegal value, the radio CPU sets the status field to ERROR_CONDITION. If the start trigger occurs in the past and startTrigger.pastTrig is 0, the radio CPU sets the status field to ERROR_PAST_START.

23.3.2.6 Command Data Structures

The data structures are listed in tables throughout this chapter. The Byte Index is the offset from the pointer to that structure. Multibyte fields are little-endian, and 16-bit halfword or 32-bit word alignment as given by the field size is required. For bit numbering, 0 is the LSB. The R/W column is used as follows:

R: The system CPU can read a result back; the radio CPU does not read the field.

W: The system CPU writes a value, the radio CPU reads it and does not modify it.

R/W: The system CPU writes an initial value, the radio CPU may modify it.

For data structures that are a specialization of another data structure, the fields from the parent structure are not repeated, but the Byte Index column reflects their presence.

The only mandatory field for all commands is the command ID number, which is a 16-bit number sent as the first 2 bytes of the command structure.

Some immediate commands have additional fields, which are defined for each command. The radio operation commands have additional mandatory fields as defined in Table 23-8.

All command fields marked as “Reserved” should be written to 0.

23.3.2.6.1 Radio Operation Command Structure

Table 23-8 shows the command structure for radio operation commands. Some commands have additional fields appended after this.

Table 23-8. Radio Operation Command Format

Byte Index	Field Name	Bits	Bit Field Name	Type	Description
0–1	commandNo			W	The command ID number
2–3	status			R/W	An integer telling the status of the command. This value is updated by the radio CPU during operation and may be read by the system CPU at any time.
4–7	pNextOp			W	Pointer to the next operation to run after this operation is done
8–11	startTime			W	Absolute or relative start time (depending on the value of the startTrigger field)
12	startTrigger				Identification of the trigger that starts the operation
13	condition			W	Condition for running the next operation

23.3.2.7 Data Entry Structures

A data entry must belong to a queue. The queues are set up as part of the command structure of a radio operation command.

Operations on queues available as commands are described in Section 23.3.4.

23.3.2.7.1 Data Entry Queue

Any command that uses a queue contains a pointer to a data entry queue structure, as given in Table 23-9. The system CPU allocates and initializes this queue structure.

Table 23-9. Data Entry Queue Structure

Byte Index	Field Name	Bits	Bit Field Name	Type	Description
0–3	pCurrEntry			R/W	Pointer to the data entry currently in use by the radio CPU (or next in line to be used if the radio is not using the queue). NULL means that no buffer is currently in the queue.
4–7	pLastEntry			R/W	Pointer to the last entry entered in this queue. If pCurrEntry is nonNULL and pLastEntry is NULL, additional entries may not be appended.

23.3.2.7.2 Data Entry

A data entry queue contains data entries of the type shown in Table 23-10. These entries are organized in a linked list. The first entry of the queue is pointed to by the pCurrEntry field of the queue structure (see Table 23-9). Each pNextEntry field points to the next entry. The pLastEntry field of the queue structure points to the last entry in the queue.

Table 23-10. General Data Entry Structure

Byte Index	Field Name	Bits	Bit Field Name	Type	Description
0–3	pNextEntry			R/W	Pointer to the next entry in the queue, NULL if this is the last entry
4	status			R/W	Indicates status of entry, including whether it is free to receive a write from the system CPU
5	config	0–1	type	W	Type of data entry structure: 0: General data entry 1: Multielement RX entry 2: Pointer entry
		2–3	lenSz	W	Size of length word in start of each RX entry element: 0: No length indicator 1: 1-byte length indicator 2: 2-byte length indicator 3: Reserved
		4–7	irqIntv	W	For partial read RX entry only: The number of bytes between interrupt generated by the radio CPU (0000: 16 bytes)
6–7	length			W	Length of data field, or for pointer entries, of the data buffer. For TX entries, this corresponds to one entry element (packet). For RX entries, this gives the total available storage space.
8–(7+n)	data			R/W	Array of data to be received or transmitted (n = length)

The status field may take the following values:

- 0: Pending: The entry is not yet in use by the radio CPU. This is the status to write by the system CPU before submitting the entry.
- 1: Active: The entry is the entry in the queue currently open for writing (RX) or reading (TX) by the radio CPU.
- 2: Busy: An ongoing radio operation is writing or reading an unfinished packet. Certain operations are not allowed while an entry is in this state (see Section 23.3.4).
- 3: Finished: The radio CPU is finished writing data into this entry, and is free for the system CPU to reuse or free memory (if dynamically allocated).

For data entries, the system CPU sets up the required data structure, either in system RAM or in the available part of the radio RAM. If the data structure is dynamically allocated, the system CPU frees the memory after use.

In an entry is being used for received data, the radio CPU may start the entry element with a length indicator. If config.lenSz is 00, no such indicator is written. This option must be used only if the length of the received packet can be determined by other means. If config.lenSz is 01, 1 byte indicates the number of bytes following the length byte. This option may be used only if no element of more than 255 bytes is written to the entry. If config.lenSz is 10, a 16-bit word indicates the number of bytes following the length word.

23.3.2.7.3 Pointer Entry

A pointer entry is an entry where the data are not contained in the entry itself, but the entry holds a pointer to the buffer. Such an entry is indicated by setting config.type to 2. The pointer replaces the data field, as shown in Table 23-11.

Table 23-11. Pointer Field in Pointer Entry Structure

Byte Index	Field Name	Bits	Bit Field Name	Type	Description
8–11	pData			W	Pointer to data buffer of size length bytes

The data is read from or stored in the buffer given by pData. The size of this buffer is given by length, just as is the size of the data field in a general data entry.

23.3.2.7.4 Partial Read RX Entry

Proprietary mode supports an RX entry where the data can be read before the entire packet is received over the air, which can be used for the following purposes:

- When data must be read before the entire packet is received
- When the length of the packet is not known in the beginning of the packet
- When the length of the packet is too long for the entire payload to be kept in memory simultaneously

To support this, a special variant of the structure in Table 23-10 is used. As for the multielement entry, several entry elements may be contained in the same entry. Each entry element corresponds to one packet received over the air, or part of it. The element may also contain additional fields. This type is selected by setting config.type to 3. In the multielement entry, the data field is composed as shown in Table 23-12 (the indexes are relative to the entire entry structure).

Table 23-12. Fields in a Partial Read RX Entry

Byte Index	Byte Field Name	Bits	Bit Field Name	Type	Description
8–9	pktStatus	0–12	numElements	R	Number of entry elements committed in the entry
		13	bEntryOpen	R	The entry contains an element that is still open for appending data.
		14	bFirstCont	R	The first element is a continuation of the last packet from the previous entry.
		15	bLastCont	R	The packet in the last element continues in the next entry.
10–11	nextIndex			R	Index to the byte after the last byte of the last entry element committed by the radio CPU
12–(7+n)	rxData			R	Data received. Exact format depends on operation being run. Each entry element may start with a length byte or word.

The entry is updated as follows:

- The nextIndex field is updated as new bytes are written to the buffer.
- While a packet is being received, the radio CPU sets pktStatus.bEntryOpen to 1.

When an entry element is finished, either because the packet ended or because the element reached the end of the entry, pktStatus.bEntryOpen is set to 0 by the radio CPU, and pktStatus.numElements is incremented. If the packet continues in the next entry, the radio CPU sets pktStatus.bLastCont to 1. In this case, the pktStatus.bFirstCont bit of the next entry is also set to 1 by the radio CPU. If no next entry is available, the status is set to Unfinished, otherwise it is set to Finished.

The length field specified in the beginning of an entry element (depending on config.lenSz) gives the length of that entry element within the entry, not the entire packet. If the length is not known when the entry is opened, the length field is written to the remaining length of the entry and updated by the radio CPU before the entry is finished.

For a partial read RX entry, the radio CPU generates an Rx_Data_Written interrupt to the system CPU whenever 1 or more bytes are written to the entry. In addition, the radio CPU generates an Rx_N_Data_Written interrupt when k bytes have been written since the last interrupt or the start of the entry element, where k is given by config.irqIntv.

23.3.2.8 External Signaling

The radio CPU controls four CPEGPOx signals that can be used for external signalling, for example for controlling external PAs and LNAs or debugging. CPEGPO0 is high when the internal LNA is enabled, CPEGPO1 is high when the internal PA is enabled and CPEGPO2 is high when synthesizer calibration is ongoing.

Two of the output signals from the RAT have automatic configuration that may be used for observation. The signal RATGPO0 goes high when transmission of a packet is initiated and low when transmission is done. RATGPO0 may be observed for accurate timing of packet transmission, because the same signal is used internally. RATGPO0 is very similar to CPEGPO1, but it goes high some microseconds earlier, and the timing is more accurate compared to the first transmitted symbol out of the modem.

By default, the radio CPU maps CPEGPO0 to the signal RFC_GPO0, CPEGPO1 to the signal RFC_GPO1, CPEGPO2 to the signal RFC_GPO2, and RATGPO0 to the signal RFC_GPO3 at boot time. This mapping can be modified by writing to the RFC_DBELL:SYSGPOCTL register.

The RFC_GPOx signals can be mapped to output pins using the system I/O controller. Refer to Chapter 11 for details.

NOTE: On the CC2640R2F device, the CPEGPO1 signal does not deassert when the internal PA is disabled. To control external PAs RATGPO0 must be used instead.

23.3.3 Command Definitions

There is a set of commands independent of the current RF protocol. These commands are related to the low-level operations of the radio.

23.3.3.1 Protocol-Independent Radio Operation Commands

For radio operation commands listed here, the operation ends due to one of the causes listed in Table 23-13, or by additional statuses listed for each command. After the operation has ended, the status field of the command structure (2 status bytes listed in Table 23-8) indicates why the operation ended. In each case, it is indicated if the result is TRUE, FALSE, or ABORT (see Section 23.3.2.5.2). This result indicates whether to start the next command (if any) indicated in pNextOp, or to return to an IDLE state.

Table 23-13. End of Radio Operation Commands

Condition	Status Code	Result
Finished operation	DONE_OK	TRUE
Received CMD_STOP while waiting for start trigger	DONE_STOPPED	FALSE
Received CMD_ABORT	DONE_ABORT	ABORT
The start trigger occurred in the past with startTrigger.pastTrig = 0	ERROR_PAST_START	ABORT
Illegal start trigger parameter	ERROR_START_TRIG	ABORT
Illegal condition for next operation	ERROR_CONDITION	ABORT
Observed illegal parameter	ERROR_PAR	ABORT
Invalid pointer to next operation	ERROR_POINTER	ABORT
Next operation has a command ID that is undefined or not a radio operation command	ERROR_CMDID	ABORT
Operation using RX, TX, or synthesizer attempted without CMD_RADIO_SETUP	ERROR_NO_SETUP	ABORT
Operation using RX or TX attempted without the synthesizer being programmed	ERROR_NO_FS	ABORT

23.3.3.1.1 CMD_NOP: No Operation Command

Command ID number: 0x0801

CMD_NOP is a radio operation command that only takes the mandatory arguments listed in Table 23-8. The command only waits for the start trigger, and then ends. The command can be used to test the communication between the system CPU and the radio CPU or to insert a wait.

23.3.3.1.2 CMD_RADIO_SETUP: Set Up Radio Settings Command

Command ID number: 0x0802

CMD_RADIO_SETUP is a radio operation command. In addition to the parameters listed in Table 23-8, the command structure contains the fields listed in Table 23-14.

Table 23-14. CMD_RADIO_SETUP Command Format

Byte Index	Field	Bit Index	Bit Field name	Type	Description
14	mode			W	This is the main mode to use. 0x00: Bluetooth low energy 0x01: IEEE 802.15.4 0x02: 2-Mbps GFSK 0x05: 5-Mbps coded 8-FSK 0xFF: Keep existing mode; update overrides only.
15	loDivider			W	CMD_PROP_RADIO_DIV_SETUP only: Divider setting to use. For the recommended settings per device and band, refer to Smart RF Studio.
16–17	config	0–2	frontEndMode		0x00: Differential mode 0x01: Single-ended mode RFP 0x02: Single-ended mode RFN 0x05: Single-ended mode RFP with external front-end control on RF pins (RFN and RXTX) 0x06: Single-ended mode RFN with external front-end control on RF pins (RFP and RXTX) Others: Reserved
		3	biasMode	W	0: Internal bias 1: External bias
		4-9	analogCfgMode	W	0x00: Write analog configuration. Required first time after boot and when changing frequency band or front-end configuration 0x2D: Keep analog configuration. May be used after standby or when changing mode with the same frequency band and front-end configuration. Others: Reserved
		10	bNoFsPowerup	W	0: Power up the frequency synthesizer. 1: Do not power up the frequency synthesizer.
		11–15			Reserved
18–19	txPower			W	Output power setting; use value from SmartRF Studio. For more details, see Section 23.3.3.2.16.
20–23	pRegOverride			W	Pointer to a list of hardware and configuration registers to override. If NULL, no override is used.

On start, the radio CPU sets up parameters for the operational mode given by mode.radioMode, with the modifications given in pRegOverride, a pointer to a structure containing override values for certain hardware registers, radio configuration controlled by the radio CPU, and protocol-related variables. If pRegOverride is NULL, no registers are overridden. The override value structure is a string of 32-bit entries provided by TI or produced by SmartRF Studio.

Running CMD_RADIO_SETUP or another radio setup command is mandatory before using any command that uses the receiver, transmitter, or frequency synthesizer. If the RF core is reset, CMD_RADIO_SETUP must be rerun.

When CMD_RADIO_SETUP is executing, trim values are read from FCFG1 unless they have been provided elsewhere. If these values are read from FCFG1, the following limitations apply while CMD_RADIO_SETUP is executing:

- The VIMS module must be powered, allowing flash reads.
- A SCLK_HF source switch must not occur, as Flash read access is momentarily disabled while the switch is ongoing.

If either of the preceding limitations is violated, the internal system bus might end up in a nonresponsive state. A system reset is required to exit this state. To avoid reading FCFG1 while running CMD_RADIO_SETUP, the values may be read in advance using CMD_READ_TRIM and provided to the radio using CMD_SET_TRIM. This is handled automatically by the TI provided RF driver.

The txPower parameter is stored and applied every time transmission of a packet starts to set an output power with temperature compensation. This setting can be changed later with the command CMD_SET_TX_POWER (see Section 23.3.3.2.16).

Table 23-15. Format of a Hardware Register Override Entry

Bit Index	Bit Field Name	Description
0–1	entryType	00: Hardware register 01: Array initiator; see Table 23-16 10: ADI register; see Table 23-17, or MCE/RFE override 11: Firmware-defined parameter; see Table 23-18
2–15	hwAddr	Bits 2–15 of the address to the hardware register. Bits 0–1 of the address are 0.
16–31	value	The value to write to the register

Table 23-16. Format of Array Initiator

Bit Index	Bit Field Name	Description
0–1	entryType	01: Array initiator
2–15	startAddr	First address or index to write to: Hardware registers: Bits 2–15 of the address (bits 0 and 1 are 0) ADI registers: ADI bus address, half-byte indicator in bit 6, ADI selector in bit 7 Firmware-defined parameters: Byte Index
16–29	length	Number of entries
30–31	arrayType	Type of array: 00: Hardware registers with 16-bit values 01: Hardware registers with 32-bit values 10: ADI registers 11: Firmware-defined parameters

Table 23-17. Format of an ADI Register Override Entry

Bit Index	Bit Field Name	Description
0–1	entryType	10: ADI register
2–9	adiValue2	Optional second value to write
10–15	adiAddr2	Optional second ADI bus address
16–23	adiValue	Value to write to register
24–29	adiAddr	ADI bus address
30	bHalfSize	0: Use full-size writes 1: Use half-size writes, causing read-modify-write functionality
31	adiNo	0: Write to ADI 0 (RF) 1: Write to ADI 1 (synthesizer)

Table 23-18. Format of a Firmware-Defined Parameter Override Entry

Bit Index	Bit Field Name	Description
0–1	entryType	11: Firmware-defined parameter
2–3	entrySubType	00: Firmware-defined parameter 01: MCE/RFE override mode (must be in first entry); see Table 23-19 10: Reserved 11: End of override list
4–14	fwAddr	Byte index into parameter structure
15	bByte	0: 16-bit value 1: 8-bit value
16–31	value	The value to write to the parameter

Table 23-19. Format of an MCE/RFE Override Mode Entry

Bit Index	Bit Field Name	Description
0–1	entryType	11: Firmware-defined parameter
2–3	entrySubType	01: MCE/RFE override mode
4–7		Reserved
8	bMceUseRam	0: Run MCE from ROM 1: Run MCE from RAM
9–11	mceRomBank	MCE ROM bank to run from
12	bRfeUseRam	0: Run RFE from ROM 1: Run RFE from RAM
13–15	rfeRomBank	RFE ROM bank to run from
16–23	mceMode	Mode to send to MCE
24–31	rfeMode	Mode to send to RFE

Table 23-20. Format of a Center Frequency Entry

Bit Index	Bit Field Name	Description
0–1	entryType	11: Firmware-defined parameter
2–3	entrySubType	10: Special configuration
4–7	specialType	0001: Center frequency entry
8		Reserved
9	bAutoTxIf	If 1, set TX IF to RX IF.
10	bApplyRx	If 1, use invRfFreq to recalculate RX IF.
11	bApplyTx	If 1, use invRfFreq to recalculate TX shape.
12–31	invRfFreq	Value where fRFMHz is center frequency in MHz: (12 × 24 × 220) / (fRFMHz × loDivider)

Table 23-21. Format of an End of List Entry

Bit Index	Bit Field Name	Description
0–1	entryType	11: Firmware-defined parameter
2–3	entrySubType	11: End of list segment
4–7	nextEntryRegion	0x0: End of list 0x1: SRAM. Base = 0x2000 0000 0x2: RF core RAM. Base = 0x2100 0000 0x3: Flash. Base = 0xA000 0000 0xF: End of list Others: Reserved
8–31	addrOffset	Address offset for next list part. Next address is: Base + (addrOffset × 4)

For hardware registers, bits 2–15 give the address of the hardware register to access, see Table 23-15. The register is written with a 32-bit write operation, but the 16 MSBs are always written as 0, while the 16 LSBs are as given by value. To write a full 32-bit hardware register, use an array operation of length 1.

An array initiator signals that the next words must be written to consecutive addresses (see Table 23-16). The type of accesses is decided by array type:

- 00 gives 32-bit writes to 16-bit hardware registers. The first register address is 0x4004 0000 + (startAddr << 2). Then length addresses are written. Each value is taken from the next 16-bit halfword of the override entry, and the register address is incremented by 4 each time a write occurs. If length is odd, padding is assumed so that the first entry after the array is 32-bit word-aligned.
- 01 gives 32-bit writes to hardware registers. The first register address is 0x4004 0000 + (startAddr << 2). Then length addresses are written. Each 32-bit value is taken from the next 32-bit word of the override entry, and the register address is incremented by 4 each time a write occurs.
- 10 gives byte writes to ADI registers. The first ADI bus address is given by bits 0–5 of startAddr. If bit 6 is set to 1, half-byte writes are used, otherwise full-byte writes (the LSB is ignored by the ADI in this case). Bit 7 selects to which ADI to write. Each value written on the ADI bus is taken from the next byte of the override entry, and the ADI register address is incremented by 1 in case of half-byte writes or by 2 in case of full-byte writes each time a write occurs. If length is not divisible by 4, padding is assumed so that the first entry after the array is 32-bit word-aligned.
- 11 gives writes to firmware-defined parameters. The first index into the configuration values is given by startAddr/4, and length bytes are copied from the override entry. If length is not divisible by 4, padding is assumed so that the first entry after the array is 32-bit word-aligned.

  For ADI registers, adiValue gives the value to write and adiAddr gives the address on the ADI bus (see Table 23-17). The ADI to write is selected through adiNo. If bHalfSize is 1, the write size bit on the ADI interface is set, causing the value to be masked half size; otherwise, it is a full-size write, and the LSB of the address is ignored. If adiAddr2 is nonzero, the value given by adiValue2 is written to the ADI bus address given by adiAddr2; otherwise, these two fields are ignored (if ADI address 0 is to be written, it must be done through adiAddr/adiValue). In this case, bHalfSize and adiNo apply to both writes.

  For radio firmware-defined parameters (see Table 23-18), fwAddr gives a Byte Index into an array of configuration values held in the radio. If bByte = 1, only the least significant byte (LSByte) of value is written to the addressed byte. If bByte = 0, all 16 bits are written to the 16-bit halfword at the given byte address, which must be even in this case. The selected value is set to the value specified in the value part of the override entry.

The first entry in the override list may contain an override of the MCE and RFE modes, as given by Table 23-19. If so, the MCE is set to run from RAM if bMceUseRam is 1 and bMceCopyRam is 0; otherwise the MCE runs from the ROM bank given by mceRomBank. The value of MDMCMDPAR0 that is set when the CPE runs the MCE configuration command is given by mceMode. Similarly, the RFE is set to run from RAM if bRfeUseRam is 1 and bRfeCopyRam is 0; otherwise the RFE runs from the ROM bank given by rfeRomBank. The value of RFECMDPAR0 set when the CPE runs the RFE configuration command is given by rfeMode.

If the pointer in pRegOverride is invalid, any override entry is invalid. If the length of an array is too large or zero, the operation ends with the status ERROR_PAR. If config.bNoFsPowerup = 0 and powering up the synthesizer fails, the command ends with ERROR_SYNTH_PROG as the status.

If CMD_ABORT or CMD_STOP is received while waiting for the start trigger, the operation ends without any setup. If CMD_STOP is received after the start trigger, setup proceeds until finished. If CMD_ABORT is received after the start trigger, the setup process is aborted. This leaves the registers in an incomplete state, so another CMD_RADIO_SETUP command must be issued before using the radio.

23.3.3.1.3 CMD_FS_POWERUP: Power Up Frequency Synthesizer

Command ID number: 0x080C

CMD_FS_POWERUP is a radio operation command. In addition to the parameters listed in Table 23-8, the command structure contains the fields listed in Table 23-22.

On start, the radio CPU powers up the frequency synthesizer and applies the register modifications given in pRegOverride. If pRegOverride is NULL, no registers are overridden. The format of the override structure is the same as the format for CMD_RADIO_SETUP (see Section 23.3.3.1.2). Only overrides applicable to the synthesizer hardware are applied.

Running CMD_FS_POWERUP is mandatory before using any command that uses the frequency synthesizer (and thus, the transmitter or receiver), unless the synthesizer has been powered up as part of the radio setup. The radio must be set up using CMD_RADIO_SETUP or another setup command before CMD_FS_POWERUP.

If the pointer in pRegOverride is invalid, the address or index is invalid, the length of an array is zero or is too large. If another parameter in an entry is not permitted, the operation ends with the status ERROR_PAR. If powering up the synthesizer fails, the command ends with ERROR_SYNTH_PROG as the status. When otherwise finished, the command ends with DONE_OK as the status.

Table 23-22. CMD_FS_POWERUP Command Format

Byte Index	Field Name	Bit Index	Bit Field Name	Type	Description
14–15					Reserved
16–19	pRegOverride			W	Pointer to a list of hardware and configuration registers to override. If NULL, no override is used.

23.3.3.1.4 CMD_FS_POWERDOWN: Power Down Frequency Synthesizer

Command ID number: 0x080D

CMD_FS_POWERDOWN is a radio operation command that only takes the mandatory arguments listed in Table 23-8. The command waits for the start trigger and then powers down the synthesizer. The act of powering down not only stops the synthesizer, as is done with CMD_FS_OFF (see Section 23.3.3.1.6) or at the end of certain other radio operation commands, but it also switches off analog modules.

After running CMD_FS_POWERDOWN, the synthesizer must be powered up again using CMD_FS_POWERUP, or another command that powers up the synthesizer before it is used.

CMD_FS_POWERDOWN must always be run before the radio is powered down (for instance, when the device is going into low-power modes).

When finished, the CMD_FS_POWERDOWN command ends with a DONE_OK status.

23.3.3.1.5 CMD_FS: Frequency Synthesizer Controls Command

Command ID number: 0x0803

CMD_FS is a radio operation command. In addition to the parameters listed in Table 23-8, the command structure contains the fields listed in Table 23-23, and can program the synthesizer to a specific frequency.

The frequency to use is given by frequency and fractFreq, and must be as close as possible to (frequency + fractFreq / 65536) MHz.

The synthesizer is set up in RX mode or TX mode, depending on synthConf.bTxMode. This mode may be changed by radio operation commands when setting up RX or TX. If synthConf.refFreq is nonzero, a reference frequency of 24 MHz / synthConf.refFreq is used instead of the default frequency.

If the synthesizer is programmed and reports loss of lock after having been in lock, the radio CPU raises the Synth_No_Lock interrupt. The synthesizer keeps running, but the system CPU may use this information to stop and restart the radio. The Synth_No_Lock interrupt is not raised more than once for each time the synthesizer is programmed. The interrupt may also occur for commands with implicit-frequency programming.

If the CMD_FS command is called with an illegal frequency or divider setting, the command ends with ERROR_PAR as the status. If the command is called without the radio being configured, it ends with ERROR_NO_SETUP as the status. If the command is called without the synthesizer being powered up, it ends with ERROR_NO_FS as status.

Table 23-23. CMD_FS Command Format

Byte Index	Field Name	Bit Index	Bit Field Name	Type	Description
14–15	frequency			W	The frequency in MHz to which the synthesizer should be tuned
16–17	fractFreq			W	Fractional part of the frequency to which the synthesizer should be tuned
18	synthConf	0	bTxMode	W	0: Start synthesizer in RX mode. 1: Start synthesizer in TX mode.
		1–6	refFreq	W	CC13x0: 0: Use default reference frequency. Others: Use reference frequency 24 MHz/refFreq. CC26x0: Reserved
		7	Reserved	W	Reserved
19–23			Reserved	W	Reserved

23.3.3.1.6 CMD_FS_OFF: Turn Off Frequency Synthesizer

Command ID number: 0x0804

CMD_FS_OFF is a radio operation command that only takes the mandatory arguments listed in Table 23-8, and turns off the frequency synthesizer if it has been started by CMD_FS or left on by a radio operation command that does not turn off the synthesizer.

When the command is started, the synthesizer outputs are disabled and the state machine is reset. The analog parts are still powered; CMD_FS_POWERDOWN (see Section 23.3.3.1.4) can power down the synthesizer to further reduce the current consumption.

23.3.3.1.7 CMD_RX_TEST: Receiver Test Command

Command ID number: 0x0807

CMD_RX_TEST is a radio operation command used to set the receiver in infinite RX mode for test purposes.

The sync word programmed in the receiver is given in the LSBs of syncWord. If config.bNoSync is 1, the correlation thresholds for sync search are set to the maximum value to avoid getting sync. The thresholds are restored after the command ends.

If pktConfig.bFsOff is 1, the synthesizer is turned off (corresponding to CMD_FS_OFF; see Section 23.3.3.1.6) after the operation is done; otherwise the synthesizer is left on.

A trigger to end the operation is set up by endTrigger and endTime (see Section 23.3.2.5.1). If the trigger that is defined by this parameter occurs, the radio operation ends.

The operation ends by one of the causes listed in Table 23-13.

The command structure for CMD_RX_TEST contains the fields listed in Table 23-24.

Table 23-24. CMD_RX_TEST Command Format

Byte Index	Byte Field Name	Bits	Bit Field Name	Type	Description
14	config	0	Reserved	W	Set to 0
		1	bFsOff	W	0: Keep frequency synthesizer on after command. 1: Turn frequency synthesizer off after command.
		2	bNoSync	W	0: Run sync search as normal for the configured mode. 1: Write correlation thresholds to the maximum value to avoid getting sync.
15	endTrigger			W	Trigger classifier for ending the operation
16–19	syncWord			W	Sync word to use for receiver
20–23	endTime			W	Time to end the operation

23.3.3.1.8 CMD_TX_TEST: Transmitter Test Command

Command ID number: 0x0808

CMD_TX_TEST is a radio operation command used to set the receiver in infinite TX mode and transmit either a CW or modulated data for test purposes.

When the command starts, the radio CPU starts the transmitter. The radio must be configured with the CMD_RADIO_SETUP command and the radio synthesizer must be started and in TX mode with the CMD_FS radio command before CMD_TX_TEST is issued. The radio transmits a preamble and a sync word of the size given by the current radio configuration, specified using CMD_RADIO_SETUP. The sync word is given in the LSBs of syncWord. The payload after the sync word consists of the 16-bit word txWord, repeated indefinitely. This word may be run through whitening, with the options given in config.whitenMode, which can take the following values:

- 0: No whitening is used. This is useful for testing with a repeated pattern, but gives spurs if used for spectral measurements.
- 1: Default whitening. This means that the whitening is as configured for the mode in use (potentially with overrides). If the mode does not use whitening, no whitening is applied.
- 2: PRBS-15: The polynomial $x^{15} + x^{14} + 1$ is used. This gives a pseudo-noise sequence with length 32767.
- 3: PRBS-31: The polynomial $x^{32} + x^{22} + x^2 + x + 1$ is used. This gives a pseudo-noise sequence with length 4294967295.

When config.whitenMode = 2 or 3, initialization is done by the radio CPU writing 0xAAAA 0000 to the PRBS value register before transmission starts. When config.whitenMode is 1, the default initialization is used.

The transmitter runs until the trigger set up by endTrigger and endTime (see Section 23.3.2.5.1) occurs, or until an abort command is issued.

If pktConfig.bFsOn is 1, the synthesizer is turned off (corresponding to CMD_FS_OFF; see Section 23.3.3.1.6) after the operation is done; otherwise it is left on.

The operation ends by one of the causes listed in Table 23-13.

The command structure for CMD_TX_TEST contains the fields listed in Table 23-25.

Table 23-25. CMD_TX_TEST Command Format

Byte Index	Field Name	Bits	Bit Field Name	Type	Description
14	config	0	bUseCw	W	0: Send modulated signal. 1: Send continuous wave.
		1	bFsOff	W	0: Keep frequency synthesizer on after command. 1: Turn frequency synthesizer off after command.
		2–3	whitenMode	W	0: No whitening 1: Default whitening 2: PRBS-15 3: PRBS-32
15					Reserved
16–17	txWord			W	Value to send to the modem before whitening
18					Reserved
19	endTrigger			W	Trigger classifier for ending the operation
20–23	syncWord			W	Sync word to use for transmitter
24–27	endTime			W	Time to end the operation

23.3.3.1.9 CMD_SYNC_STOP_RAT: Synchronize and Stop Radio Timer Command

Command ID number: 0x0809

CMD_SYNC_STOP_RAT is a radio operation command. In addition to the parameters listed in Table 23-8, the command structure contains the fields listed in Table 23-26.

For more details, see Chapter 14.

AON_RTC:CTL.RTC_UPD_EN must be 1.

Table 23-26. CMD_SYNC_STOP_RAT Command Format

Byte Index	Field Name	Bits	Bit Field Name	Type	Description
14–15					Unused
16–19	rat0			R	The returned RAT value corresponding to the value the RAT would have had when the RTC was zero.

When the command starts, the radio CPU sets up capture of an RTC tick and waits for this tick, then stops the RAT and calculates the value rat0. This value must be stored for use when the RAT restarts.

23.3.3.1.10 CMD_SYNC_START_RAT: Synchronously Start Radio Timer Command

Command ID number: 0x080A

CMD_SYNC_START_RAT is a radio operation command. In addition to the parameters listed in Table 23-8, the command structure contains the fields listed in Table 23-27.

For more details, see Chapter 14. TOP:AON_RTC:CTL.RTC_UPD_EN must be 1.

Table 23-27. CMD_SYNC_START_RAT Command Format

Byte Index	Field Name	Bits	Bit Field Name	Type	Description
14–15					Unused
16–19	rat0			W	The desired RAT value corresponding to the value the RAT would have had when the RTC was zero. This parameter is returned by CMD_SYNC_STOP_RAT.

When the command starts, the radio CPU starts the RAT and sets up capture of an RTC tick and waits for this tick, then calculates the necessary timer adjustment based on the input parameter rat0 and performs this adjustment. The input parameter rat0 is the value previously returned by CMD_SYNC_STOP_RAT.

Because the RAT is normally not running when this command is issued, the start trigger must be TRIG_NOW (see Section 23.3.2.5.1).

The first time the RAT is started after system boot, the command CMD_START_RAT must be used (see Section 23.3.3.2.7). As an alternative, CMD_SYNC_START_RAT may be issued with a fixed parameter such as 0; however, this gives an arbitrary start value for the RAT. Before powering down the radio, the system CPU must run the CMD_SYNC_STOP_RAT command. After powering up the radio again, the system CPU must run the CMD_SYNC_START_RAT command with the same parameter as the one received when CMD_SYNC_STOP_RAT was issued.

23.3.3.1.11 CMD_COUNT: Counter Command

Command ID number: 0x080B

CMD_COUNT is a radio operation command. In addition to the parameters listed in Table 23-8, the command structure contains the fields listed in Table 23-28.

Table 23-28. CMD_COUNT Command Format

Byte Index	Field Name	Bits	Bit Field Name	Type	Description
14–15	counter			R/W	Counter. When starting, the radio CPU decrements the value, and the end status of the operation differs if the result is zero.

When the command starts, the radio CPU decrements the counter field by 1 and writes the result back to this field. If the result of the decrement is 0, the operation ends with the status DONE_COUNTDOWN and the result FALSE. Otherwise, the operation ends with the status DONE_OK and the result TRUE, which can be used in conditional execution to create a loop.

If the operation is started with counter equal to 0, this is an illegal parameter, so the operation ends with the status ERROR_PAR.

The operation ends by one of the causes listed in Table 23-2 or Table 23-29.

Table 23-29. Additional End Causes for CMD_COUNT

Condition	Status Code	Result
Finished operation with counter > 0	DONE_OK	TRUE
Finished operation with counter = 0	DONE_COUNTDOWN	FALSE

23.3.3.1.12 CMD_SCH_IMM: Run Immediate Command as Radio Operation

Command ID number: 0x0810

CMD_SCH_IMM is a radio operation command. In addition to the parameters listed in Table 23-8, the command structure contains the fields listed in Table 23-30.

Table 23-30. CMD_SCH_IMM Command Format

Byte Index	Field Name	Bits	Bit Field Name	Type	Description
14–15					Reserved
16–19	cmdrVal			W	Value as would be written to CMDR
20–23	cmdstaVal			R	Value as would be returned in CMDSTA

When the command starts, the radio CPU takes the value in cmdrVal and processes it as if it had been written to the CMDR register. This command may be a pointer to an immediate command, or it may form a direct command as shown in Figure 23-4. A pointer to a radio operation command causes a scheduling error. The value that would normally have been returned in CMDSTA is written to cmdstaVal. This means that no command RF_CMD_ACK interrupt is raised. Instead, a COMMAND_DONE interrupt is raised, as for any other radio operation command.

Depending on the result of the immediate or direct command, the status and result of the radio operation command is as shown in Table 23-31.

CMD_SCH_IMM may run immediate commands as part of a chain of radio operation commands, or to schedule them in the future. If an immediate or direct command received in the CMDR register is being processed at the same time that a scheduled CMD_SCH_IMM command starts, the processing of the scheduled command starts after the other command has finished.

Table 23-31. End Statuses for CMD_SCH_IMM

Condition	Status Code	Result
Immediate command ended with the result DONE	DONE_OK	TRUE
There was an error in the execution of the command, giving a CMDSTA result not listed in the following table row.	DONE_FAILED	FALSE
There was an error in the command, giving one of the following results: • SchedulingError • UnknownCommand • UnknownDirCommand • IllegalPointer • ParError	ERROR_PAR	ABORT

23.3.3.1.13 CMD_COUNT_BRANCH: Counter Command With Branch of Command Chain

Command ID number: 0x0812

CMD_COUNT_BRANCH is a radio operation command. In addition to the parameters listed in Table 23-8, the command structure contains the fields listed in Table 23-32.

Table 23-32. CMD_COUNT_BRANCH Command Format

Byte Index	Field Name	Bits	Bit Field Name	Type	Description
14–15	counter			R/W	Counter When starting, the radio CPU decrements the value, and the end status of the operation differs if the result is zero.
16–19	pNextOpIfOk			W	Pointer to next operation if counter did not expire

When the command starts, the radio CPU decrements the counter field by 1, unless it was already 0, and writes the result back to this field. If the result of the decrement is 0, the operation ends with the status DONE_COUNTDOWN and the result FALSE. Otherwise, the operation ends with the status DONE_OK and the result TRUE. In this case, the next radio operation command to run is given by pNextOpIfOk instead of pNextOp (see Table 23-8), which can be used in conditional execution to create a loop.

If the operation is started with the counter equal to 0, the operation ends with the status DONE_OK and the next operation is taken from pNextOpIfOk. This operation can be used if the previous command is set up to skip optionally, as skipping from a previous command in the chain follows pNextOp.

The operation ends by one of the causes listed in Table 23-13 or Table 23-33.

Table 23-33. Additional End Causes for CMD_COUNT_BRANCH

Condition	Status Code	Result
Finished operation with counter = 0 when being started	DONE_OK	TRUE
Finished operation with counter > 0 after decrementing	DONE_OK	TRUE
Finished operation with counter = 0 after decrementing	DONE_COUNTDOWN	FALSE

23.3.3.1.14 CMD_PATTERN_CHECK: Check a Value in Memory Against a Pattern

Command ID number: 0x0813

CMD_PATTERN_CHECK is a radio operation command. In addition to the parameters listed in Table 23-8, the command structure contains the fields listed in Table 23-34.

Table 23-34. CMD_PATTERN_CHECK Command Format

Byte Index	Field Name	Bits	Bit Field Name	Type	Description
14–15	patternOpt	0–1	operation	W	Operation to perform: 0: TRUE if value == compareVal 1: TRUE if value < compareVal 2: TRUE if value > compareVal 3: Reserved
		2	bByteRev	W	If 1, interchange the 4 bytes of the value, so that they are read the MSB first.
		3	bBitRev	W	If 1, perform bit reversal of the value.
		4–8	signExtend	W	0: Treat value and compareVal as unsigned. 1–31: Treat value and compareVal as signed, where the value gives the number of the MSB in the signed number.
		9	bRxVal	W	0: Use pValue as a pointer. 1: Use pValue as a signed offset to the start of the last committed RX entry element.
16–19	pNextOpIfOk			W	Pointer to next operation if comparison result was true
20–23	pValue			W	Pointer from which to read, or offset from last RX entry if patternOpt.bRxVal == 1
24–27	mask			W	Bit mask to apply before comparison
28–31	compareVal			W	Value to which to compare

When the command starts, the radio CPU reads a 4-byte value from the location pointed to by pValue if patternOpt.bRxVal == 0. If patternOpt.bRxVal == 1, the location to read from is found by taking the pointer to the start of the last committed RX entry element and adding the signed number found in pValue as a byte offset. In either case, this pointer does not need to be 4-byte-aligned. If the pointer is not byte-aligned, the value is read byte by byte.

The value is then subject to the following operations in this order:

1. If patternOpt.bByteRev == 1, interchange byte 3 with byte 0, and byte 1 with byte 2, as if the bytes had been read MSB first.
2. If patternOpt.bBitRev == 1, reverse the bit order of the entire 32-bit word.
3. Perform a bitwise ‘AND’ operation between the value and mask.
4. If patternOpt.signExtend > 0, copy the value of bit number patternOpt.signExtend (where bit 0 is the LSB) into all the more significant bits.
5. Perform a compare operation between the resulting value and compareVal, depending on patternOpt.operation (see Table 23-34). The compare operation is unsigned if patternOpt.signExtend == 0; otherwise it is signed.

If patternOpt.operation or pValue have illegal values, the operation ends with a status of ERROR_PAR. Otherwise, the operation ends by one of the causes listed in Table 23-13 or Table 23-35, depending on the result of the comparison in Step 5 in the previous list. If the comparison result was TRUE, the next radio operation command to run is given by pNextOpIfOk instead of pNextOp.

Table 23-35. Additional End Causes for CMD_PATTERN_CHECK

Condition	Status Code	Result
Comparison result was TRUE.	DONE_OK	TRUE
Comparison result was FALSE.	DONE_FAILED	FALSE
Command run with patternOpt.bRxVal when no RX data is fully received	ERROR_NO_RX	ABORT

23.3.3.2 Protocol-Independent Direct and Immediate Commands

This section contains immediate commands that can be used across protocols. Commands for manipulating data queues are described in Section 23.3.4.

23.3.3.2.1 CMD_ABORT: Abort Command

Command ID number: 0x0401

CMD_ABORT is a direct command.

On reception, the radio CPU ends ongoing radio operation commands as soon as possible. Analog circuitry for RX and TX is safely turned off, and data structures are updated so they are not left in an unfinished state.

If a radio operation command is running when the CMD_ABORT command is issued, the radio CPU produces a COMMAND_DONE and LAST_COMMAND_DONE interrupt when the radio operation command finishes. The status of the command structure of that radio operation command reflects that the command was aborted.

If no radio operation command is running, no action is taken. The result signaled in the CMDSTA register is DONE in all cases. If a radio operation command is running, CMDSTA may be updated before the radio operation ends.

23.3.3.2.2 CMD_STOP: Stop Command

Command ID number: 0x0402

CMD_STOP is a direct command.

On reception, the radio CPU informs the radio operation command currently running that it has been requested to stop. The CMD_STOP command is more *graceful* than the CMD_ABORT command, but might take more time to finish. Normally, a packet being received or transmitted is handled to completion. The exact behavior on reception of CMD_STOP is described for each radio operation command. Some commands always end in a known time and do not respond to CMD_STOP.

If no radio operation command is running, no action is taken. The result signaled in the CMDSTA register is DONE in all cases. If a radio operation command is running, CMDSTA may be updated before the radio operation ends.

23.3.3.2.3 CMD_GET_RSSI: Read RSSI Command

Command ID number: 0x0403

CMD_GET_RSSI is an immediate command that takes no parameters, and therefore, can be used as a direct command.

On reception, the radio CPU reads the RSSI from an underlying receiver. The RSSI is returned in result byte 2 (bit 23–16) of CMDSTA (see Figure 23-5). The RSSI is given on signed form in dBm. If no RSSI is available, this is signaled with a special value of the RSSI (−128, or 0x80).

If no radio operation command is running, the radio CPU returns the result ContextError in CMDSTA. Otherwise, the radio CPU returns a result of DONE along with the RSSI value.

23.3.3.2.4 CMD_UPDATE_RADIO_SETUP: Update Radio Settings Command

Command ID number: 0x0001

CMD_UPDATE_RADIO_SETUP is an immediate command that takes the parameters listed in Table 23-36.

Table 23-36. CMD_UPDATE_RADIO_SETUP Command Format

Byte Index	Field Name	Bits	Bit Field Name	Type	Description
0–1	commandNo			W	The command ID number
2–3					Reserved
4–7	pRegOverride			W	Pointer to a list of hardware and configuration registers to override

On reception, the radio CPU updates the registers given in pRegOverride. This is a pointer to a structure containing an override value for certain hardware registers, a radio configuration controlled by the radio CPU, and protocol-related variables. The format is as for CMD_RADIO_SETUP (see Section 23.3.3.1). If done while the radio is running, the update must primarily be done on the radio and protocol configuration, as modifications to hardware registers may cause undesired behavior.

23.3.3.2.5 CMD_TRIGGER: Generate Command Trigger

Command ID number: 0x0404

CMD_TRIGGER is an immediate command that takes the parameters listed in Table 23-37.

Table 23-37. CMD_TRIGGER Command Format

Byte Index	Field Name	Bits	Bit Field Name	Type	Description
0–1	commandNo			W	The command ID number
2	triggerNo			W	Command trigger number

On reception, the radio CPU generates the command trigger specified with triggerNo, so that running radio operation commands respond accordingly (see Section 23.3.2.5.1).

If the trigger number is outside the valid range 0–3, the radio CPU returns the result ParError in CMDSTA. If no radio operation command running is pending on the trigger number sent, the radio CPU returns the result ContextError in CMDSTA. Otherwise, the radio CPU returns a result of DONE, which may be returned before the running radio operation command responds to the trigger.

CMD_TRIGGER may be sent as a direct command. If so, the trigger number is given by the parameter in bits 8–15 of CMDR.

23.3.3.2.6 CMD_GET_FW_INFO: Request Information on the Firmware Being Run

Command ID number: 0x0002

CMD_GET_FW_INFO is an immediate command that takes the parameters listed in Table 23-38.

Table 23-38. CMD_GET_FW_INFO Command Format

Byte Index	Field Name	Bits	Bit Field Name	Type	Description
0–1	commandNo			W	The command ID number
2–3	versionNo			R	Firmware version number
4–5	startOffset			R	The start of free RAM
6–7	freeRamSz			R	The size of free RAM
8–9	availRatCh			R	Bitmap of available RAT channels

On reception, the radio CPU reports information on the running radio firmware. A version number is returned in versionNo. The startOffset and freeRamSz fields contain information on the area in the radio RAM that is not used by the radio CPU for data (including stack and heap). This area is free to use by the system CPU for data exchange, radio CPU patching, or other purposes. The start and end address of the free RAM is given as offset from the start of the radio RAM.

NOTE: Some of this free RAM is used for patches provided by TI.

The availRatCh field is a bitmap where bit position n indicates whether RAT channel n may be used by the system CPU. A bit value of 1 indicates that the corresponding channel may be used by the system CPU, while a bit value of 0 means that the channel is reserved for the radio CPU or is nonexistent.

23.3.3.2.7 CMD_START_RAT: Asynchronously Start Radio Timer Command

Command ID number: 0x0405

CMD_ START_RAT is a direct command.

On reception, the radio CPU starts the RAT if it has not already started.

If the RAT is already running, the radio CPU returns the result ContextError in CMDSTA. Otherwise, the radio CPU returns a result of DONE.

23.3.3.2.8 CMD_PING: Respond With Interrupt

Command ID number: 0x0406

CMD_PING is a direct command.

On reception, the radio CPU returns a result of DONE in CMDSTA. This command can test the communication between the two CPUs, or check when the radio CPU is ready after boot.

23.3.3.2.9 CMD_READ_RFREG: Read RF Core Register

Command ID number: 0x0601

CMD_READ_RFREG is an immediate command that takes the parameters listed in Table 23-39.

Table 23-39. CMD_READ_RFREG Command Format

Byte Index	Field Name	Bits	Bit Field Name	Type	Description
0–1	commandNo			W	The command ID number
2–3	address			W	The offset from the start of the RF core hardware register bank (0x4004 0000)
4–7	value			R	Returned value of the register

On reception, the radio CPU reads the RF core register with address 0x4004 0000 + address. The result is written to value. If the address is not divisible by 4, the radio CPU returns ParError in CMDSTA.

CMD_READ_RFREG may be sent as a direct command. If so, the address is given by bits 2–15 of CMDR, with the 2 LSBs of the address set to 00.

When reading has been performed, the result is returned in value. The 24 LSBs of the result are returned in CMDSTA bits 8–31. The result returned in CMDSTA is DONE.

23.3.3.2.10 CMD_SET_RAT_CMP: Set RAT Channel to Compare Mode

Command ID number: 0x000A

CMD_SET_RAT_CMP is an immediate command that takes the parameters listed in Table 23-40.

Table 23-40. CMD_SET_RAT_CMP Command Format

Byte Index	Field Name	Bits	Bit Field Name	Type	Description
0–1	commandNo			W	The command ID number
2	ratCh			W	The radio timer channel number
3					Reserved
4–7	compareTime			W	The time at which the compare occurs

On reception, the radio CPU sets the RAT channel given by ratCh in compare mode, and sets the channel compare time to compareTime, which also arms the channel. A channel event occurs at the given time, and this can be enabled as an RF hardware interrupt to the system CPU through the RFC_DBELL module.

The channel number must indicate a channel that is not reserved for use by the radio CPU. Otherwise, the radio CPU returns ParError in CMDSTA. If the compare time is in the past when the command is evaluated, the radio CPU returns ContextError in CMDSTA and disables the RAT channel. If the compare event is successfully set up, the radio CPU returns a result of DONE in CMDSTA.

23.3.3.2.11 CMD_SET_RAT_CPT: Set RAT Channel to Capture Mode

Command ID number: 0x0603

CMD_SET_RAT_CPT is an immediate command that takes the parameters listed in Table 23-41.

Table 23-41. CMD_SET_RAT_CPT Command Format

Byte Index	Field Name	Bits	Bit Field Name	Type	Description
0–1	commandNo			W	The command ID number
2–3	config	0–2			Reserved
		3–7	inputSrc	W	Input source indicator: 22: RFC_GPI0 23: RFC_GPI1 Others: Reserved
		8–11	ratCh	W	The radio timer channel number
		12	bRepeated	W	0: Single capture mode 1: Repeated capture mode
		13–14	inputMode	W	Input mode: 00: Rising edge 01: Falling edge 10: Both edges 11: Reserved

On reception, the radio CPU sets the RAT channel given by config.ratCh in capture mode. If config.bRepeated is 0, the channel is set to single capture mode; otherwise, the channel is set to repeated capture mode. The radio CPU sets the input source to config.inputSrc and the input mode to config.inputMode. If the channel is set in single capture mode, it is also armed. This causes a channel event to occur when capture occurs, and can be enabled as an RF hardware interrupt to the system CPU through the RFC_DBELL module.

CMD_SET_RAT_CMP may be sent as a direct command. If so, bits 2–15 of the config word are given by bits 2–15 of CMDR (bits 0–1 of config are not used).

The channel number must indicate a channel that is not reserved for use by the radio CPU. Otherwise, the radio CPU returns ParError in CMDSTA. If the channel is successfully set up, the radio CPU returns a result of DONE in CMDSTA.

23.3.3.2.12 CMD_DISABLE_RAT_CH: Disable RAT Channel

Command ID number: 0x0408

CMD_DISABLE_RAT_CH is an immediate command that takes the parameters listed in Table 23-42.

Table 23-42. CMD_DISABLE_RAT_CH Command Format

Byte Index	Field Name	Bits	Bit Field Name	Type	Description
0–1	commandNo			W	The command ID number
2	ratCh			W	The RAT channel number

On reception, the radio CPU disables the RAT channel given by ratCh. This disables previous configurations of that channel done by the CMD_SET_RAT_CMP or CMD_SET_RAT_CPT command.

CMD_DISABLE_RAT_CH may be sent as a direct command. If so, ratCh is given by the parameter in bits 8–15 of CMDR.

The channel number must indicate a channel that is not reserved for use by the radio CPU. Otherwise, the radio CPU returns ParError in CMDSTA. If the channel number is valid, the CPU returns a result of DONE in CMDSTA after the channel has been disabled.

23.3.3.2.13 CMD_SET_RAT_OUTPUT: Set RAT Output to a Specified Mode

Command ID number: 0x0604

CMD_SET_RAT_OUTPUT is an immediate command that takes the parameters listed in Table 23-43.

Table 23-43. CMD_SET_RAT_OUTPUT Command Format

Byte Index	Field Name	Bits	Bit Field Name	Type	Description
0–1	commandNo			W	The command ID number
2–3	config	0–1			Reserved
		2–4	outputSel	W	Output event indicator: 1: RAT_GPO1 2: RAT_GPO2 3: RAT_GPO3 Others: Reserved
		5–7	outputMode	W	Output mode: 000: Pulse 001: Set 010: Clear 011: Toggle 100: Always 0 101: Always 1 Others: Reserved
		8–11	ratCh	W	The RAT channel number

On reception, the radio CPU sets the RAT output event given by config.outputSel in the mode given by config.outputMode, and to be controlled by the RAT channel given by config.ratCh. This command must be combined with setting this channel in compare mode, using the CMD_SET_RAT_CMP command.

CMD_SET_RAT_OUTPUT may be sent as a direct command. If so, bits 2–15 of the config word are given by bits 2–15 of CMDR (bits 0–1 of config are not used).

The channel number, config.ratCh, must indicate a channel that is not reserved for use by the radio CPU, and the output number, config.outputSel, must not be an output used by the radio CPU. Otherwise, the radio CPU returns ParError in CMDSTA. If the output event is successfully set up, the radio CPU returns a result of DONE in CMDSTA.

23.3.3.2.14 CMD_ARM_RAT_CH: Arm RAT Channel

Command ID number: 0x0409

CMD_ARM_RAT_CH is an immediate command that takes the parameters listed in Table 23-44.

Table 23-44. CMD_ARM_RAT_CH Command Format

Byte Index	Field Name	Bits	Bit Field Name	Type	Description
0–1	commandNo			W	The command ID number
2	ratCh			W	The RAT channel number

On reception, the radio CPU arms the RAT channel given by ratCh.

The CMD_DISABLE_RAT_CH command may be sent as a direct command. If so, ratCh is given by the parameter in bits 8–15 of CMDR.

The channel number must indicate a channel not reserved for use by the radio CPU. Otherwise, the radio CPU returns ParError in CMDSTA. If the channel number is valid, the CPU returns a result of DONE in CMDSTA after the channel is armed.

23.3.3.2.15 CMD_DISARM_RAT_CH: Disarm RAT Channel

Command ID number: 0x040A

CMD_DISARM_RAT_CH is an immediate command that takes the parameters listed in Table 23-45.

Table 23-45. CMD_DISARM_RAT_CH Command Format

Byte Index	Field Name	Bits	Bit Field Name	Type	Description
0–1	commandNo			W	The command ID number
2	ratCh			W	The RAT channel number

On reception, the radio CPU disarms the RAT channel given by ratCh.

CMD_DISABLE_RAT_CH may be sent as a direct command. If so, ratCh is given by the parameter in bits 8–15 of CMDR.

The channel number must indicate a channel not reserved for use by the radio CPU. Otherwise, the radio CPU returns ParError in CMDSTA. If the channel number is valid, the CPU returns a result of DONE in CMDSTA after the channel is armed.

23.3.3.2.16 CMD_SET_TX_POWER: Set Transmit Power

Command ID number: 0x0010

CMD_SET_TX_POWER is an immediate command that takes the parameters listed in Table 23-46 (CC26x0) and Table 23-47 (CC13x0).

Table 23-46. CMD_SET_TX_POWER Command Format (CC26x0)

Byte Index	Field Name	Bits	Bit Field Name	Type	Description
0–1	commandNo			W	The command ID number
2–3	txPower			W	New TX power setting. It is recommended to use values from SmartRF Studio. Bits 0-5: IB Value to write to the PA power control field at 25°C. See Equation 14 for details. Bits 6-7: GC Value to write to the gain control of the first stage of the PA. Bits 8-15: tempCoeff Temperature coefficient for IB. 0: No temperature compensation.

$$IB = IB_{25°C} + \frac{(Temperature[°C] - 25) \times tempCoeff}{512} \quad (14)$$

Table 23-47. CMD_SET_TX_POWER Command Format (CC13x0)

Byte Index	Field Name	Bits	Bit Field Name	Type	Description
0–1	commandNo			W	The command ID number
2–3	txPower			W	New TX power setting. It is recommended to use values from SmartRF Studio. Bits 0-5: IB Value to write to the PA power control field at 25°C. See Equation 15 for details. Bits 6-7: GC Value to write to the gain control of the first stage of the PA. Bit 8: boost Driver strength into the PA. 0: Low driver strength 1: High driver strength Bits 9-15: tempCoeff Temperature coefficient for IB. 0: No temperature compensation.

$$IB = IB_{25°C} + \frac{(Temperature[°C] - 25) \times tempCoeff}{256} \quad (15)$$

On reception, the radio CPU sets the transmit power for use the next time transmission begins. If a packet is being transmitted, the transmit power is not updated until transmission begins for the next packet.

Each time transmission of a packet begins, temperature compensation of the transmit power is done.

On completion, the radio CPU returns a result of DONE in CMDSTA.

23.3.3.2.17 CMD_UPDATE_FS: Set New Synthesizer Frequency Without Recalibration

Command ID number: 0x0011

CMD_UPDATE_FS is an immediate command that takes the parameters listed in Table 23-48.

Table 23-48. CMD_UPDATE_FS Command Format

Byte Index	Field Name	Bits	Bit Field Name	Type	Description
0–1	commandNo			W	The command ID number
2–13					Reserved
14–15	frequency			W	The frequency in MHz to tune to
16–17	fractFreq			W	Fractional part of the frequency to tune to

On reception, the radio CPU programs a new frequency in the synthesizer without restarting calibration. This must be a small change compared to the frequency used under calibration, otherwise the synthesizer is most likely unable to relock. Extra distortion may occur if the command is done during RX or TX.

This command is supported only in the 2.4-GHz frequency band.

NOTE: This command is not characterized. Limits for frequency changes are unknown.

The frequency to use is given by frequency and fractFreq, and the frequency must be as close as possible to (frequency + fractFreq / 65536) MHz.

If the synthesizer is not running and the calibration is done, the radio CPU returns ContextError in CMDSTA. If frequency is invalid, the radio CPU returns ParError in CMDSTA. Otherwise, the radio CPU returns a result of DONE in CMDSTA when the update is finished.

23.3.3.2.18 CMD_MODIFY_FS: Set New Synth Frequency Without Recalibration

Command ID number: 0x0013

CMD_MODIFY_FS is an immediate command that takes the parameters listed in Table 23-49.

Table 23-49. CMD_MODIFY_FS Command Format

Byte Index	Field Name	Bits	Bit Field Name	Type	Description
0–1	commandNo			W	The command ID number
2–3	frequency			W	The frequency in MHz to which to tune
4–5	fractFreq			W	Fractional part of the frequency to which to tune

On reception, the radio CPU will program a new frequency in the synthesizer without restarting calibration. This has to be a small change compared to the frequency used under calibration, otherwise the synthesizer will most likely be unable to relock. Extra distortion may occur if the command is done during RX or TX.

The frequency to use is given by frequency and fractFreq, and the frequency will be as close as possible to (frequency + fractFreq / 65536) MHz.

If the synthesizer is not running and the calibration done, the radio CPU will return ContextError in CMDSTA. If frequency is invalid, the radio CPU will return ParError in CMDSTA. Otherwise, the radio CPU will return DONE in CMDSTA when the update is complete.

23.3.3.2.19 CMD_BUS_REQUEST: Request System BUS Available for RF Core

Command ID number: 0x040E

CMD_BUS_REQUEST is an immediate command that takes the parameters listed in Table 23-50.

Table 23-50. CMD_BUS_REQUEST Command Format

Byte Index	Field Name	Bits	Bit Field Name	Type	Description
0–1	commandNo			W	The command ID number
2	bSysBusNeeded			W	0: System bus may sleep 1: System bus access needed

On reception, the radio CPU sets the bus request bit toward the PRCM to 1 if bSysBusNeeded is nonzero, or to 0 if bSysBusNeeded is zero. If bSysBusNeeded is nonzero, this indicates that the system bus stays awake even if the system goes to deep sleep, which must be done for the RF core to run and access the system side for one of the following reasons:

- Any command structure, data structure, and so on, pointed to by a pointer sent to the RF core is placed in system RAM or flash.
- The RF core must read the temperature because the TX power has a nonzero temperature coefficient.
- The RF core must read the RTC to synchronize with the RAT during CMD_SYNC_STOP_RAT or CMD_SYNC_START_RAT.

CMD_BUS_REQUEST may be sent as a direct command. If so, bSysBusNeeded is given by the parameter in bits 8–15 of CMDR.

The radio CPU returns a result of DONE in CMDSTA when the update finishes.

23.3.4 Immediate Commands for Data Queue Manipulation

The following commands are immediate commands used for data queue manipulation for all radio operations that use data queues.

23.3.4.1 CMD_ADD_DATA_ENTRY: Add Data Entry to Queue

Command ID number: 0x0005

CMD_ADD_DATA_ENTRY is an immediate command that takes the parameters listed in Table 23-51.

Table 23-51. CMD_ADD_DATA_ENTRY Command Format

Byte Index	Field Name	Bits	Bit Field Name	Type	Description
0–1	commandNo			W	The command ID number
2–3					Reserved
4–7	pQueue			W	Pointer to the queue structure to which the entry is added
8–11	pEntry			W	Pointer to the entry

On reception, the radio CPU appends the provided data entry to the queue indicated. The radio CPU performs the following operations:

```
Set pQueue-> pLastEntry-> pNextEntry = pEntry
Set pQueue-> pLastEntry = pEntry
```

If either of the pointers pQueue or pEntry are invalid (that is, in an address range that is not memory or without 32-bit word alignment), the command fails, and the radio CPU sets the result byte of CMDSTA to ParError. If the queue specified in pQueue is set up not to allow entries to be appended (see Section 23.3.2.7.1), the command fails, and the radio CPU sets the result byte of CMDSTA to QueueError.

23.3.4.2 CMD_REMOVE_DATA_ENTRY: Remove First Data Entry From Queue

Command ID number: 0x0006

CMD_REMOVE_DATA_ENTRY is an immediate command that takes the parameters listed in Table 23-52.

Table 23-52. CMD_REMOVE_DATA_ENTRY Command Format

Byte Index	Field Name	Bits	Bit Field Name	Type	Description
0–1	commandNo			W	The command ID number
2–3					Reserved
4–7	pQueue			W	Pointer to the queue structure from which the entry is removed
8–11	pEntry			R	Pointer to the entry that was removed

On reception, the radio CPU removes the first data entry from the queue indicated. The command returns a pointer to the entry that was removed. The radio CPU performs the following operations:

```
Set pEntry = pQueue->pCurrEntry
Set pQueue->pCurrEntry = pEntry->pNextEntry
Set pEntry->status = Finished
```

If the pointer pQueue is invalid, the command fails, and the radio CPU sets the result byte of CMDSTA to ParError. If the queue specified in pQueue is empty, the command fails, and the radio CPU sets the result byte of CMDSTA to QueueError. If the entry to be removed is in the BUSY state, the command fails, and the radio CPU sets the result byte of CMDSTA to QueueBusy.

23.3.4.3 CMD_FLUSH_QUEUE: Flush Queue

Command ID number: 0x0007

CMD_FLUSH_QUEUE is an immediate command that takes the parameters listed in Table 23-53.

Table 23-53. CMD_FLUSH_QUEUE Command Format

Byte Index	Field Name	Bits	Bit Field Name	Type	Description
0–1	commandNo			W	The command ID number
2–3					Reserved
4–7	pQueue			W	Pointer to the queue structure to be flushed
8–11	pFirstEntry			R	Pointer to the first entry that was removed

On reception, the radio CPU flushes the queue indicated, and returns a pointer to the first entry that was removed. The radio CPU performs the following operations:

```
Set pFirstEntry = pQueue->pCurrEntry
Set pQueue->pCurrEntry = NULL
Set pQueue->pLastEntry = NULL
```

If the pointer pQueue is invalid, the command fails, and the radio CPU sets the result byte of CMDSTA to ParError. If the first entry to be removed is in the BUSY state, the command fails, and the radio CPU sets the result byte of CMDSTA to QueueBusy. If the queue specified in pQueue is empty, the radio CPU does not need to do any operation, but this is still viewed as a success. The returned pFirstEntry is NULL in this case.

23.3.4.4 CMD_CLEAR_RX: Clear All RX Queue Entries

Command ID number: 0x0008

CMD_CLEAR_RX is an immediate command that takes the parameters listed in Table 23-54.

Table 23-54. CMD_CLEAR_RX Command Format

Byte Index	Field Name	Bits	Bit Field Name	Type	Description
0–1	commandNo			W	The command ID number
2–:3					Reserved
4–7	pQueue			W	Pointer to the queue structure to be cleared

On reception, the radio CPU makes all RX entries indicate that they are empty. The radio CPU performs the following operations:

```
Set pTemp = pQueue->pCurrEntry
Loop: Set pTemp->status = Pending
If pTemp->type == 1 then:
        Set pTemp->nextIndex = 0
        Set pTemp->numElements = 0
Set pTemp = pTemp->nextIndex
If pTemp != NULL and pTemp != pQueue->pCurrEntry, repeat from Loop
```

If the pointer pQueue is invalid, the command fails, and the radio CPU sets the result byte of CMDSTA to ParError. If the queue specified in pQueue is empty, the command fails, and the radio CPU sets the result byte of CMDSTA to QueueError. If the first entry to be removed is in the BUSY state, the command fails, and the radio CPU sets the result byte of CMDSTA to QueueBusy.

23.3.4.5 CMD_REMOVE_PENDING_ENTRIES: Remove Pending Entries From Queue

Command ID number: 0x0009

CMD_REMOVE_PENDING_ENTRIES is an immediate command that takes the parameters listed in Table 23-55.

Table 23-55. CMD_REMOVE_PENDING_ENTRIES Command Format

Byte Index	Field Name	Bits	Bit Field Name	Type	Description
0–1	commandNo			W	The command ID number
2–3					Reserved
4–7	pQueue			W	Pointer to the queue structure to be flushed
8–11	pFirstEntry			R	Pointer to the first entry that was removed

On reception, the radio CPU removes all entries that are in the Pending state from the queue indicated, and returns a pointer to the first entry that was removed. The radio CPU performs the following operations:

```
If pQueue->pCurrEntry->status = Pending, then
        Set pFirstEntry = pQueue->pCurrEntry
        Set pQueue->pCurrEntry = NULL
        Set pQueue->pLastEntry = NULL
else
        Set pFirstEntry = pQueue->pCurrEntry->pNextEntry
        Set pQueue->pCurrEntry->pNextEntry = NULL
        Set pQueue->pLastEntry = pQueue->pCurrEntry
```

If the pointer pQueue is invalid, the command fails, and the radio CPU sets the result byte of CMDSTA to ParError. If the queue specified in pQueue is empty, the radio CPU does not need to do any operation, but this is still viewed as a success. The returned pFirstEntry is NULL in this case.

23.4 Data Queue Usage

This section describes how the radio CPU uses data queues.

23.4.1 Operations on Data Queues Available Only for Internal Radio CPU Operations

Section 23.3.4 lists commands used for data queue manipulation. For internal radio CPU operations described, additional operations are available. These operations are described in the following sections.

23.4.1.1 PROC_ALLOCATE_TX: Allocate TX Entry for Reading

The procedure takes the following input parameters:

- Pointer to queue, pQueue

The procedure returns the following:

- Pointer to allocated data entry, pEntry

The procedure returns with error if the specified queue is empty, or if the first entry of the queue is already busy. Otherwise, the following is done:

```
Set pQueue->pCurrEntry->status = Busy
Set pEntry = pQueue->pCurrEntry
```

23.4.1.2 PROC_FREE_DATA_ENTRY: Free Allocated Data Entry

The procedure takes the following input parameters:

- Pointer to queue, pQueue

The procedure returns the following:

- Pointer to allocated data entry, pEntry

The procedure returns with error if the specified queue is empty. Otherwise, the following is done:

```
Set pQueue->pCurrEntry->status = Active
```

23.4.1.3 PROC_FINISH_DATA_ENTRY: Finish Use of First Data Entry From Queue

The procedure takes the following input parameters:

- Pointer to queue, pQueue

The procedure returns the following:

- Pointer to new entry, pEntry

The procedure returns with error if the specified queue is empty. Otherwise, the following is done:

```
Set pTemp = pQueue->pCurrEntry
Set pQueue->pCurrEntry = pTemp->pNextEntry
Set pTemp->status = Finished
Set pEntry = pQueue->pCurrEntry
```

23.4.1.4 PROC_ALLOCATE_RX: Allocate RX Buffer for Storing Data

The procedure takes the following input parameters:

- Pointer to queue, pQueue
- Size of entry element to store, size

The procedure returns the following:

- Pointer to data entry where data is stored, pEntry
- Pointer to a finished data entry, or NULL if not finished, pFinishedEntry

The procedure returns with error if the first entry of the queue is already busy. If there is not room for an entry element of the specified size, including if the queue is empty, a “no space” error is returned. The following procedure describes the operations:

```
Set pFinishedEntry == NULL
If pQueue->pCurrEntry == NULL then
        Return with no space error
end if
If pQueue->pCurrEntry->type != 1 then
        if pQueue->pCurrEntry->length < size then
                Return with no space error
        else
                Set pQueue->pCurrEntry->status = Busy
                Set pEntry = pQueue->pCurrEntry
        end if
else
        Set pTemp = pQueue->pCurrEntry
        If pTemp->nextIndex + 2 + size > pTemp->length then
                Set pQueue->pCurrEntry = pTemp->pNextEntry
                Set pTemp->status = Finished
                Set pFinishedEntry = pTemp
                Set pTemp = pTemp->pNextEntry
                If pTemp == NULL or pTemp->length < size + 2 then
                        Return with no space error
                end if
        end if
        Set pTemp->status = Busy
        Set pEntry = pTemp
end if
```

23.4.1.5 PROC_FINISH_RX: Commit Received Data to RX Data Entry

The procedure takes the following input parameters:

- Pointer to queue, pQueue
- Size of entry element that has been stored, size

The procedure returns the following:

- Pointer to data entry where data is stored, pEntry
- Pointer to a finished data entry, or NULL if not finished, pFinishedEntry

The procedure returns with error if the queue is empty or if there is not room for an entry element of the specified size. Otherwise, the following is done:

```
If pQueue->pCurrEntry->type != 1 then
        Set pTemp = pQueue->pCurrEntry
        Set pQueue->pCurrEntry = pTemp->pNextEntry
        Set pTemp->status = Finished
else
        Increase pQueue->pCurrEntry->nextIndex by size
        Increment pQueue->pCurrEntry->numElements by 1
        If pQueue->pCurrEntry->nextIndex + 2 == pQueue->pCurrEntry-
>length then
                Set pTemp = pQueue->pCurrEntry
                Set pQueue->pCurrEntry = pTemp->pNextEntry
                Set pTemp->status = Finished
                Set pFinishedEntry == pTemp
        else
                Set pQueue->pCurrEntry->status = Active
                Set pFinishedEntry == NULL
        end if
end if
```

This operation is done after doing PROC_ALLOCATE_RX and writing to the correct locations in the buffer; the size must be the same as with PROC_ALLOCATE_RX.

23.4.2 Radio CPU Usage Model

23.4.2.1 Receive Queues

When the radio CPU receives a packet, it prepares a buffer for reading by calling PROC_ALLOCATE_RX. If this is successful, the allocated buffer is used to store the incoming packet as defined for each protocol. If a no space error occurs, the received data cannot be stored, and the handling is defined for each protocol.

After a packet has been received, it may be kept or discarded depending on rules defined for each protocol. To keep the packet, the radio CPU calls PROC_FINISH_RX. This makes the received data available for the system CPU. To discard the packet, the radio CPU calls PROC_FREE_DATA_ENTRY, meaning that the next packet may overwrite the data received in the last packet.

23.4.2.2 Transmit Queues

When the radio CPU is about to transmit a packet from a TX queue, it calls PROC_ALLOCATE_TX to get a pointer to the data to transmit. When the packet transmits, the radio CPU calls PROC_FINISH_DATA_ENTRY or PROC_FREE_DATA_ENTRY. If PROC_FINISH_DATA_ENTRY is called, the system CPU is informed that the entry is finished and may be reused. This calling process must be used if retransmission of the packet is not an option. If PROC_FREE_DATA_ENTRY is called, the transmitted entry remains first in the queue so that it may be transmitted, which is used when an acknowledgment is expected.

If an acknowledgment is received on a packet that was transmitted, followed by the radio CPU calling PROC_FREE_DATA_ENTRY, the radio CPU calls PROC_ALLOCATE_TX followed by PROC_FINISH_DATA_ENTRY (this is equivalent to CMD_REMOVE_DATA_ENTRY, see Section 23.3.3.2). This calling process causes the next entry in the queue to be transmitted. If an acknowledgment is not received, the last transmitted packet is retransmitted.

23.5 IEEE 802.15.4

This section describes IEEE 802.15.4-specific command structure, interrupts, data handling, radio operation commands, and immediate commands.

23.5.1 IEEE 802.15.4 Commands

Table 23-56 and Table 23-57 define the IEEE 802.15.4-specific radio operation commands.

Table 23-56. IEEE 802.15.4 Radio Operation Commands on Background Level

ID	Command Name	Description
0x2801	CMD_IEEE_RX	Run receiver
0x2802	CMD_IEEE_ED_SCAN	Run energy detect scan

Table 23-57. IEEE 802.15.4 Radio Operation Commands on Foreground Level

ID	Command Name	Description
0x2C01	CMD_IEEE_TX	Transmit packet
0x2C02	CMD_IEEE_CSMA	Perform CSMA-CA
0x2C03	CMD_IEEE_RX_ACK	Receive acknowledgment
0x2C04	CMD_IEEE_ABORT_BG	ABORT background level operation

In addition, Table 23-58 defines immediate commands.

Table 23-58. IEEE 802.15.4 Immediate Commands

ID	Command Name	Description
0x2001	CMD_IEEE_MOD_CCA	Modify CCA parameters for running receiver
0x2002	CMD_IEEE_MOD_FILT	Modify frame filtering parameters for running receiver
0x2003	CMD_IEEE_MOD_SRC_MATCH	Modify source matching parameters for running receiver
0x2401	CMD_IEEE_ABORT_FG	ABORT foreground level operation
0x2402	CMD_IEEE_STOP_FG	Stop foreground level operation
0x2403	CMD_IEEE_CCA_REQ	Request CCA and RSSI information

23.5.1.1 IEEE 802.15.4 Radio Operation Command Structures

Table 23-8 defines the first 14 bytes for all radio operation commands. The CMD_IEEE_ABORT_BG command does not have any additional fields to those 14 bytes. Table 23-59 lists the IEEE 802.15.4 RX command structure for bytes 14–59.

Table 23-59. IEEE 802.15.4 RX Command Structure

Byte Index	Field Name	Type	Description
14	channel	W	Channel to tune to in the start of the operation: 0: Use existing channel 11–26: Use as IEEE 802.15.4 channel; that is, frequency is [2405 + 5 × (channel − 11)] MHz 60–207: Frequency is (2300 + channel) MHz Others: reserved
15	rxConfig	W	Configuration bits for the receive queue entries (see Table 23-69 for details)
16–19	pRxQ	W	Receive queue
20–23	pOutput	W	Pointer to result structure (see Table 23-68) (NULL: Do not store results)
24–25	frameFiltOpt	R/W	Frame filtering options (see Table 23-71 for details)
26	frameTypes	R/W	Frame types to receive in frame filtering (see Table 23-72 for details)
27	ccaOpt	R/W	CCA options (see Table 23-70 for details)
28	ccaRssiThr	R/W	RSSI threshold for CCA
29			Reserved
30	numExtEntries	W	Number of extended address entries
31	numShortEntries	W	Number of short address entries
32–35	pExtEntryList	W	Pointer to list of extended address entries
36–39	pShortEntryList	W	Pointer to list of short address entries
40–47	localExtAddr	W	The extended address of the local device
48–49	localShortAddr	W	The short address of the local device
50–51	localPanID	W	The PAN ID of the local device
52–54			Reserved
55	endTrigger	W	Trigger that causes the device to end the RX operation
56–59	endTime	W	Time parameter for endTrigger

Table 23-60 lists the IEEE 802.15.4 energy detect scan command structure.

Table 23-60. IEEE 802.15.4 Energy Detect Scan Command Structure

Byte Index	Field Name	Type	Description
14	channel	W	Channel to tune to at the start of the operation: 0: Use existing channel 11–26: Use as IEEE 802.15.4 channel; that is, frequency is [2405 + 5 × (channel – 11)] MHz 60–207: Frequency is (2300 + channel) MHz Others: reserved
15	ccaOpt	R/W	CCA options (see Table 23-70 for details)
16	ccaRssiThr	R/W	RSSI threshold for CCA
17			Reserved
18	maxRssi	R	The maximum RSSI recorded during the ED scan
19	endTrigger	W	Trigger that causes the device to end the RX operation
20–23	endTime	W	Time parameter for endTrigger

Table 23-61 lists the IEEE 802.15.4 CSMA-CA command structure.

Table 23-61. IEEE 802.15.4 CSMA-CA Command Structure

Byte Index	Field Name	Bits	Bit Field Name	Type	Description
14–15	randomState			R/W	The state of the pseudo-random generator
16	macMaxBE			W	The IEEE 802.15.4 MAC parameter macMaxBE
17	macMaxCSMABackoffs			W	The IEEE 802.15.4 MAC parameter macMaxCSMABackoffs
18	csmaConfig	0–4	initCW	W	The initialization value for the CW parameter
		5	bSlotted	W	0 for nonslotted CSMA, 1 for slotted CSMA
		6–7	rxOffMode	W	0: RX stays on during CSMA backoffs. 1: The CSMA-CA algorithm suspends the receiver if no frame is being received. 2: The CSMA-CA algorithm suspends the receiver if no frame is being received, or after finishing it (including auto ACK) otherwise. 3: The CSMA-CA algorithm suspends the receiver immediately during backoffs.
19	NB			R/W	The NB parameter from the IEEE 802.15.4 CSMA-CA algorithm
20	BE			R/W	The BE parameter from the IEEE 802.15.4 CSMA-CA algorithm
21	remainingPeriods			R/W	The number of remaining periods from a paused backoff countdown
22	lastRssi			R	RSSI measured at the last CCA operation
23	endTrigger			W	Trigger that causes the device to end the CSMA-CA operation
24–27	lastTimeStamp			R	Time of the last CCA operation
28–31	endTime			W	Time parameter for endTrigger

Table 23-62 lists the IEEE 802.15.4 TX command structure.

Table 23-62. IEEE 802.15.4 TX Command Structure

Byte Index	Field Name	Bits	Bit Field Name	Type	Description
14	txOpt	0	bIncludePhyHdr	W	0: Find PHY header automatically. 1: Insert PHY header from the buffer.
		1	bIncludeCrc	W	0: Append automatically calculated CRC. 1: Insert FCS (CRC) from the buffer.
		2			Reserved
		3–7	payloadLenMsb	W	Most significant bits of payload length. Must only be nonzero to create long nonstandard packets for test purposes.
15	payloadLen			W	Number of bytes in the payload
16–19	pPayload			W	Pointer to payload buffer of size payloadLen
20–23	timeStamp			R	Timestamp of transmitted frame

Table 23-63 lists the IEEE 802.15.4 Receive ACK command structure.

Table 23-63. IEEE 802.15.4 Receive ACK Command Structure

Byte Index	Field Name	Type	Description
14	seqNo	W	Sequence number to expect
15	endTrigger	W	Trigger that causes the device to give up acknowledgment reception
16–19	endTime	W	Time parameter for endTrigger

23.5.1.2 IEEE 802.15.4 Immediate Command Structures

Table 23-64 lists the IEEE 802.15.4 Modify CCA immediate command structure.

Table 23-64. IEEE 802.15.4 Modify CCA Immediate Command Structure

Byte Index	Field Name	Type	Description
0–1	commandNo	W	The command number
2	newCcaOpt	W	New value of ccaOpt for the running background level operation (see Table 23-70 for details)
3	newCcaRssiThr	W	New value of ccaRssiThr for the running background level operation

Table 23-65 lists the IEEE 802.15.4 modify frame filtering immediate command structure.

Table 23-65. IEEE 802.15.4 Modify Frame Filtering Immediate Command Structure

Byte Index	Field Name	Type	Description
0–1	commandNo	W	The command number
2–3	newFrameFiltOpt	W	New value of frameFiltOpt for the running background level operation
4	newFrameTypes	W	New value of frameTypes for the running background level operation

Table 23-66 lists the IEEE 802.15.4 enable or disable source matching entry immediate command structure.

Table 23-66. IEEE 802.15.4 Enable or Disable Source Matching Entry Immediate Command Structure

Byte Index	Field Name	Bits	Bit Field Name	Type	Description
0–1	commandNo			W	The command number
2	options	0	bEnable	W	0: Disable entry 1: Enable entry
		1	srcPend	W	New value of the pending bit for the entry
		2	entryType	W	0: Extended address 1: Short address
		3–7			Reserved
3	entryNo			W	Index of entry to enable or disable

Table 23-67 lists the IEEE 802.15.4 Request CCA state immediate command structure.

Table 23-67. IEEE 802.15.4 Request CCA State Immediate Command Structure

Byte Index	Field Name	Bits	Bit Field Name	Type	Description
0–1	commandNo			W	The command number
2	currentRssi			R	The RSSI currently observed on the channel
3	maxRssi			R	The maximum RSSI observed on the channel because RX was started
4	ccaInfo	0–1	ccaState	R	Value of the current CCA state: 00: Idle 01: Busy 10: Invalid
		2–3	ccaEnergy	R	Value of the current energy detect CCA :state. 00: Idle 01: Busy 10: Invalid
		4–5	ccaCorr	R	Value of the current correlator based carrier sense CCA state: 00: Idle 01: Busy 10: Invalid
		6	ccaSync	R	Value of the current sync found based carrier sense CCA state: 0: Idle 1: Busy
		7			Reserved

23.5.1.3 Output Structures

Table 23-68 lists the RX commands.

Table 23-68. RX Command

Byte Index	Field Name	Type	Description
0	nTxAck	R/W	Number of transmitted ACK frames
1	nRxBeacon	R/W	Number of received beacon frames
2	nRxData	R/W	Number of received data frames
3	nRxAck	R/W	Number of received acknowledgment frames
4	nRxMacCmd	R/W	Number of received MAC command frames
5	nRxReserved	R/W	Number of received frames with reserved frame type
6	nRxOk	R/W	Number of received frames with CRC error
7	nRxIgnored	R/W	Number of frames received that are to be ignored
8	nRxBufFull	R/W	Number of received frames discarded because the RX buffer was full
9	lastRssi	R	RSSI of last received frame
10	maxRssi	R	Highest RSSI observed in the operation
11			Reserved
12–15	beaconTimeStamp	R	Timestamp of last received beacon frame

23.5.1.4 Other Structures and Bit Fields

Table 23-69 lists the receive queue entry configuration bit fields.

Table 23-69. Receive Queue Entry Configuration Bit Field

Bits	Bit Field Name	Description
0	bAutoFlushCrc	If 1, automatically remove packets with CRC error from RX queue.
1	bAutoFlushIgn	If 1, automatically remove packets that can be ignored according to frame filtering from RX queue.
2	bIncludePhyHdr	If 1, include the received PHY header field in the stored packet; otherwise discard it.
3	bIncludeCrc	If 1, include the received CRC field in the stored packet; otherwise discard it. This requires pktConf.bUseCrc to be 1.
4	bAppendRssi	If 1, append an RSSI byte to the packet in the RX queue.
5	bAppendCorrCrc	If 1, append a correlation value and CRC result byte to the packet in the RX queue.
6	bAppendSrcInd	If 1, append an index from the source matching algorithm.
7	bAppendTimestamp	If 1, append a timestamp to the packet in the RX queue.

Table 23-70 lists the CCA configuration bit fields.

Table 23-70. CCA Configuration Bit Field

Bits	Bit Field Name	Description
0	ccaEnEnergy	Enable energy scan as CCA source.
1	ccaEnCorr	Enable correlator-based carrier sense as CCA source.
2	ccaEnSync	Enable sync found-based carrier sense as CCA source.
3	ccaCorrOp	Operator to use between energy-based and correlator-based CCA: 0: Report busy channel if either ccaEnergy or ccaCorr are busy. 1: Report busy channel if both ccaEnergy and ccaCorr are busy.
4	ccaSyncOp	Operator to use between sync found based CCA and the others: 0: Always report busy channel if ccaSync is busy. 1: Always report idle channel if ccaSync is idle.
5–6	ccaCorrThr	Threshold for number of correlation peaks in correlator-based carrier sense.
7		Reserved

Table 23-71 lists the frame filtering configuration bit fields

Table 23-71. Frame Filtering Configuration Bit Field

Bits	Bit Field Name	Description
0	frameFiltEn	0: Disable frame filtering 1: Enable frame filtering
1	frameFiltStop	0: Receive all packets to the end 1: Stop receiving frame once frame filtering has caused the frame to be rejected
2	autoAckEn	0: Disable auto ACK 1: Enable auto ACK
3	slottedAckEn	0: Nonslotted ACK 1: Slotted ACK
4	autoPendEn	0: Auto-pend disabled 1: Auto-pend enabled
5	defaultPend	The value of the pending data bit in auto ACK packets that are not subject to auto-pend.
6	bPendDataReqOnly	0: Use auto-pend for any packet 1: Use auto-pend for data request packets only
7	bPanCoord	0: Device is not PAN coordinator 1: Device is PAN coordinator
8–9	maxFrameVersion	Reject frames where the frame version field in the FCF is greater than this value.
10–12	fcfReservedMask	Value to be ANDed with the reserved part of the FCF; frame rejected if result is nonzero.
13–14	modifyFtFilter	Treatment of MSB of frame type field before frame-type filtering: 0: No modification 1: Invert MSB 2: Set MSB to 0 3: Set MSB to 1
15	bStrictLenFilter	0: Accept acknowledgment frames of any length ≥ 5 1: Accept only acknowledgment frames of length 5

Table 23-72 lists the frame type filtering bit fields.

Table 23-72. Frame Type Filtering Bit Field

Bits	Bit Field Name	Description
0	bAcceptFt0Beacon	Treatment of frames with frame type 000 (beacon): 0: Reject 1: Accept
1	bAcceptFt1Data	Treatment of frames with frame type 001 (data): 0: Reject 1: Accept
2	bAcceptFt2Ack	Treatment of frames with frame type 010 (ACK): 0: Reject, unless running ACK receive command 1: Always accept
3	bAcceptFt3MacCmd	Treatment of frames with frame type 011 (MAC command): 0: Reject 1: Accept
4	bAcceptFt4Reserved	Treatment of frames with frame type 100 (reserved): 0: Reject 1: Accept
5	bAcceptFt5Reserved	Treatment of frames with frame type 101 (reserved): 0: Reject 1: Accept
6	bAcceptFt6Reserved	Treatment of frames with frame type 110 (reserved): 0: Reject 1: Accept
7	bAcceptFt7Reserved	Treatment of frames with frame type 111 (reserved): 0: Reject 1: Accept

Table 23-73 lists the short address entry structures.

Table 23-73. Short Address Entry Structure

Byte Index	Field Name	Description
0–1	shortAddr	Short address of the entry
2–3	panID	PAN ID of the entry

Table 23-74 lists the extended address list structure.

Table 23-74. Extended Address List Structure

Byte Index	Field Name	Type	Description
0–(4K − 1)	srcMatchEn	R/W	Words with enable bits for each extAddrEntry; LSB of first word corresponds to entry 0. The array size K = ceil (N / 32), where N is the number of entries (given by numExtEntries, see Table 23-59) and ceil denotes rounding upward.
(4K)–(8K − 1)	srcPendEn	R/W	Words with pending data bits for each extAddrEntry; LSB of first word corresponds to entry 0.
(8K)–(8K + 7)	extAddrEntry[0]	W	Extended address number 0
...			
(8K + 8n)–(8K + 8n + 7)	extAddrEntry[n]	W	Extended address number n
...			
[8K + 8(N − 1)]–(8K + 8N + 7)	extAddrEntry[N-1]	W	Extended address number N−1 (last entry)

Table 23-75 lists the short address list structure.

Table 23-75. Short Address List Structure

Byte Index	Field Name	Type	Description
0–(4K − 1)	srcMatchEn	R/W	Words with enable bits for each shortAddrEntry; LSB of first word corresponds to entry 0. The array size K = ceil (N / 32), where N is the number of entries (given by numShortEntries, see Table 23-59) and ceil denotes rounding upward.
(4K)–(8K − 1)	srcPendEn	R/W	Words with pending data bits for each shortAddrEntry; LSB of first word corresponds to entry 0.
(8K)–(8K + 3)	shortAddrEntry[0]	W	Short address number 0; the entry is an address/PAN ID pair as defined in Table 23-73.
...			
(8K + 4n)–(8K + 4n + 3)	shortAddrEntry[n]	W	Short address number n; the entry is an address/PAN ID pair as defined in Table 23-73.
...			
[8K + 4(N − 1)]–(8K + 4N + 3)	shortAddrEntry[N-1]	W	Short address number N−1 (last entry); the entry is an address/PAN ID pair as defined in Table 23-73.

Table 23-76 lists the receive correlation/CRC result bit fields.

Table 23-76. Receive Correlation/CRC Result Bit Field

Bits	Bit Field Name	Description
0–5	corr	The correlation value
6	bIgnore	1 if the packet must be rejected by frame filtering; 0 otherwise
7	bCrcErr	1 if the packet was received with CRC error; 0 otherwise

23.5.2 Interrupts

The interrupts to be used by the IEEE 802.15.4 commands are listed in Table 23-77. Each interrupt may be enabled individually in the system CPU. Details for when the interrupts are generated are given in Section 23.5.4.

Table 23-77. Interrupt Definitions Applicable to IEEE 802.15.4

Interrupt Number	Interrupt Name	Description
0	COMMAND_DONE	A background level radio operation command has finished.
1	LAST_COMMAND_DONE	The last background level radio operation command in a chain of commands has finished.
2	FG_COMMAND_DONE	A foreground radio operation command has finished
3	LAST_FG_COMMAND_DONE	The last foreground radio operation command in a chain of commands has finished.
4	TX_DONE	Transmitted frame
5	TX_ACK	Transmitted automatic ACK frame
16	RX_OK	Frame received with CRC OK
17	RX_NOK	Frame received with CRC error
18	RX_IGNORED	Frame received with ignore flag set
22	RX_BUF_FULL	Frame received that did not fit in the TX queue
23	RX_ENTRY_DONE	TX queue data entry changing state to Finished
29	MODULES_UNLOCKED	As part of the boot process, the Cortex-M0 has opened access to RF core modules and memories.
30	BOOT_DONE	The RF core CPU boot is finished.
31	INTERNAL_ERROR	The radio CPU has observed an unexpected error.

23.5.3 Data Handling

For all the IEEE 802.15.4 commands, data received over the air is stored in a receive queue.

Data to be transmitted is fetched from a buffer given in the transmit command.

23.5.3.1 Receive Buffers

A frame being received is stored in the receive buffer. First, a length byte or word is stored, if configured in the RX entry, by config.lenSz, and calculated from the length received over the air and the configuration of appended status information.

The format of the entry elements in the receive queue pointed to by pRxQ is given by the configuration rxConfig defined in Section 23.6.1.4.

Following the length field, the received PHY header byte is stored if rxConfig.bIncludePhyHdr is 1. If a length field is present, this byte is redundant except for the reserved bit. The received MAC header and MAC payload is stored as received over the air. The MAC footer containing the 16-bit frame check sequence is stored if rxConfig.bIncludeCrc is 1.

If rxConfig.bAppendRssi is 1, a byte indicating the received RSSI value is appended. If rxConfig.bAppendCorrCrc is 1, a status byte of the type defined in Table 23-76 is appended. If rxConfig.bAppendSrcInd is 1, a byte giving the index of the first source matching entry that matches the header of the received packet is appended, or 0xFF if no match. If rxConfig.bAppendTimeStamp is 1, a timestamp indicating the start of the frame is appended. This timestamp is a 4-byte number from the radio timer. Though the timestamp is multibyte, no word-address alignment is made, so the timestamp must be written and read byte-wise. The timestamp is captured when SFD is found, but is adjusted to reflect the start of the frame (assuming 8 preamble bytes as per the standard), defined so that it corresponds to the time of the start trigger used on the transmit side. The adjustment is defined in the syncTimeAdjust firmware-defined parameter, and may be overridden.

Figure 23-6 shows the format of an entry element in the RX queues.

Figure 23-6. RX Queue Entry Element (Stapled Fields are Optional)

0–2 bytes	0 or 1 byte	0–125 bytes	0 or 2 bytes	0 or 1 byte	0 or 1 byte	0 or 1 byte	0 or 4 bytes
Element length	PHY header	MAC header and payload	MAC footer (FCS)	RSSI	Status	Source index	Timestamp

23.5.3.2 Transmit Buffers

In the transmit operation, a pointer to a buffer containing the payload is given by pPayload. The length of this buffer is given separately by payloadLen. The contents of the transmit buffer is given by the txOpt parameter. The transmit buffer always contains the MAC header and MAC payload. If txOpt.bIncludePhyHdr is 1, the buffer also includes the byte to be transmitted as a PHY header as the first byte in the buffer. If txOpt.bIncludeCrc is 1, the last 2 bytes of the buffer are transmitted as a CRC instead of the CRC being calculated automatically.

23.5.4 Radio Operation Commands

Before running any radio operation command described in this document, the radio must be set up in IEEE 802.15.4 mode using the command CMD_RADIO_SETUP. Otherwise, the operation ends with an error.

In IEEE 802.15.4 mode, the radio CPU accepts two levels of radio operation commands. Operations can run in the background level or in the foreground level. Each operation can run in only one of these levels. Operations in the foreground level normally require a background-level operation running at the same time.

The background-level operations are the receive and energy detect scan operations. Only one of these operations can run at a time. The foreground-level operations are the CSMA-CA operation, the receive ACK operation, the transmit operation, the abort background level operation, and the modify radio setup operation. These can be entered as one command or a command chain, even if a background-level operation is running. The CSMA-CA and receive ACK operations run simultaneously with the background-level operation. The transmit operation causes suspension of the background level operation until the transmission is done. Table 23-78 shows the allowed combinations of background and foreground-level operations. Violation of these combinations causes an error when the foreground-level command is about to start, signaled by the ERROR_WRONG_BG status in the status field of the foreground-level command structure.

Table 23-78. Allowed Combinations of Foreground and Background Level Operations

Foreground Level Operation	Background Level Operation		
	None	CMD_IEEE_RX	CMD_IEEE_ED_SCAN
None	Allowed	Allowed	Allowed
CMD_IEEE_TX	Allowed1	Allowed	Allowed
CMD_IEEE_CSMA	Forbidden	Allowed	Allowed
CMD_IEEE_RX_ACK	Forbidden	Allowed	Forbidden
CMD_IEEE_ABORT_BG	Allowed2	Allowed	Allowed

A non-15.4 radio operation may not be run simultaneously with a 15.4 radio operation; if a non-15.4 radio operation is entered while a 15.4 operation is running on either level, a scheduling error occurs. Chains of 15.4 and non-15.4 operations can be created, however.

When a foreground-level operation finishes, an FG_COMMAND_DONE interrupt is raised. If the command was the last one in a chain, a LAST_FG_COMMAND_DONE interrupt is raised as well (see Table 23-77). Background-level operations use the common interrupts, COMMAND_DONE and LAST_COMMAND_DONE (see Table 23-77).

The status field of the command structure is updated during the operation. When submitting the command, the system CPU writes this field with a state of IDLE. During the operation, the radio CPU updates the field to indicate the operation mode. When the operation is done, the radio CPU writes a status indicating that the command has finished. Table 23-79 lists the status codes for IEEE 802.15.4 radio operation.

Table 23-79. IEEE 802.15.4 Radio Operation Status Codes

Number	Name	Description
Operation Not Finished		
0x0000	IDLE	Operation not started
0x0001	PENDING	Waiting for start trigger
0x0002	ACTIVE	Running operation
0x2001	IEEE_SUSPENDED	Operation suspended
Normal Operation Ending		
0x2400	IEEE_DONE_OK	Operation ended normally
0x2401	IEEE_DONE_BUSY	CSMA-CA operation ended with failure
0x2402	IEEE_DONE_STOPPED	Operation stopped after stop command
0x2403	IEEE_DONE_ACK	ACK packet received with pending data bit cleared
0x2404	IEEE_DONE_ACKPEND	ACK packet received with pending data bit set
0x2405	IEEE_DONE_TIMEOUT	Operation ended due to time-out
0x2406	IEEE_DONE_BGEND	FG operation ended because necessary background level operation ended
0x2407	IEEE_DONE_ABORT	Operation aborted by command
Operation Ending With Error		
0x0806	ERROR_WRONG_BG	Foreground level operation is not compatible with running background level operation
0x2800	IEEE_ERROR_PAR	Illegal parameter
0x2801	IEEE_ERROR_NO_SETUP	Radio was not set up in IEEE 802.15.4 mode
0x2802	IEEE_ERROR_NO_FS	Synthesizer was not programmed when running RX or TX
0x2803	IEEE_ERROR_SYNTH_PROG	Synthesizer programming failed
0x2804	IEEE_ERROR_RXOVF	RX overflow observed during operation
0x2805	IEEE_ERROR_TXUNF	TX underflow observed during operation

The conditions for giving each status are listed for each operation. Some of the error causes listed in Table 23-79 are not repeated in these lists. In some cases, general error causes described in Section 23.3 may occur. In all of these cases, the result of the operation as defined in Section 23.3 is ABORT.

23.5.4.1 RX Operation

The receive radio operation is a background-level operation, started with the CMD_IEEE_RX command and using the command structure given in Table 23-59.

At the start of an RX operation, the radio CPU waits for the start trigger, then programs the frequency based on the channel parameter. If the channel is 0xFF, the operation keeps running on an already-configured channel. This requires that the operation follows another receive operation or a synthesizer programming operation. If the frequency synthesizer is not running, the operation ends with an error. After programming the frequency, the radio CPU configures the receiver to receive IEEE 802.15.4 packets.

When the demodulator obtains sync on a frame, the PHY header is read first. The 7 LSBs of this byte give the frame length. The further treatment depends on the setting of frameFiltOpt. If frameFiltOpt.frameFiltEn is 1, further frame filtering is done as explained in the following subsections. If frameFiltOpt.frameFiltEn is 0, no frame filtering is done.

The number of bytes given by the received PHY header are received and stored in the receive queue given by pRxQ. As explained in Section 23.6.3.1, the format depends on rxConfig. The last 2 bytes of the PHY payload are the FCS, or CRC, for the packet. These bytes are checked according to the FCS specification, and the further treatment depends on the CRC result.

If there is a CRC error and rxConfig.bAutoFlushCrc is 1, the packet is discarded from the RX buffer. If there is no available RX buffer with enough available space to hold the received packet, the received data is discarded. If frameFiltOpt.frameFiltStop is 1, the reception stops, otherwise the packet is received so that the CRC can be checked.

23.5.4.1.1 Frame Filtering and Source Matching

If frameFiltOpt.frameFiltEn is 1, frame filtering and source matching are performed as described in this section. The frame filtering may have several purposes:

- Distinction between different packet types
- Rejection of packets with a nonmatching destination address
- Rejection of packets with unknown version or illegal fields
- Automatic identification of source address
- Automatic acknowledgment transmission
- Automatic insertion of pending data bit based on source address

23.5.4.1.1.1 Frame Filtering

When frame filtering is enabled, the MAC header of the packet is investigated by the radio CPU. The frame control field (FCF) is checked first. The frame type subfield is the first subfield of the FCF to be checked, and determines the further processing. The MSB of the frame type is processed according to frameFiltOpt.modifyFtFilter before the check is made. The result of this modification is used only when checking, not when storing the FCF in the RX queue entry. For each of the eight possible values of the frame type field (including four reserved fields), the frame can be set up to be accepted or rejected. This is controlled by the bits of frameTypes. If the frame type is Acknowledgment (010b) and a CMD_RX_ACK operation is running in the foreground, the packet is processed further even if frameTypes.bAcceptFt2Ack is 0. In that case, Section 23.5.4.5 gives more details on the processing.

Filtering is performed on the Frame Version and Reserved subfields. If the frame version is greater than frameFiltOpt.maxFrameVersion, the frame is rejected.

If the Reserved subfield ANDed with frameFiltOpt.fcfReservedMask is nonzero, the frame is rejected. The addressing fields are checked to see if the frame must be accepted or not. This filtering follows the rules for third-level filtering (refer to the IEEE 802.15.4 standard). When checking against the local address, the localExtAddr or localShortAddr field is used, and when checking against the local PAN ID, the localPanID field is used.

If frameFiltOpt.bStrictLenFilter is 1 and the frame type indicates that the frame is an acknowledgment frame, the frame is rejected if the length of the PHY payload is not 5, which is the length of a correctly formulated ACK frame.

If frameFiltOpt.frameFiltStop is 1 and the frame filtering gives the conclusion that the frame is to be rejected, reception stops and the radio returns to sync search. Otherwise, the frame is received to the end.

The radio CPU checks the header to see if an acknowledgment is to be transmitted. This gives a preliminary result; the actual transmission of the ACK depends on the status at the end of the frame. The condition for transmitting an acknowledgment frame is given in Section 23.5.4.1.3.

23.5.4.1.1.2 Source Matching

Source matching is performed on frames accepted by the frame filtering with a source address present. If the source address was an extended address, the received address is compared against the entries in the list pExtEntryList. If the source address was a short address, the received address and source pan ID are compared against the entries in the list pShortEntryList.

The number of entries that the lists can hold is given by numExtEntries and numShortEntries. If either of these values is 0, no source matching is performed on addresses of the corresponding type, and the corresponding pointer is NULL. The lists start with source mapping enable bits, srcMatchEn, and continue with pending enable bits, srcPendEn, followed by the list entries, see Table 23-73 and Table 23-74. The enable bits consist of the number of 32-bit words needed to hold an enable bit for each entry in the list. For each entry where the corresponding srcMatchEn bit is 1, the entry is compared against the received source address for extended addresses, or against the received source address and PAN ID for short addresses. If a match is found, the index is stored, and reported back in the message footer if configured (see Section 23.6.3.1). If no match is found, the index reported back is 0xFF.

The source matching procedure may also be used to find the pending data bit to be transmitted in an auto-acknowledgment frame (see Section 23.5.4.1.3). If frameFiltOpt.autoPendEn is 1 and a source match was found, the pending data bit is set to the value of the bit in srcPendEn corresponding to the index of the match. If no match was found or if frameFiltOpt.autoPendEn is 0, the pending data bit is set equal to frameFiltOpt.defaultPend. If frameFiltOpt.bPendDataReqOnly is 1, the radio CPU investigates the frame to determine if it is a MAC command frame with the command frame identifier set to a Data Request. If not, the pending data bit of an auto ACK is set to 0, regardless of the source matching result and the value of frameFiltOpt.defaultPend.

23.5.4.1.2 Frame Reception

After frame filtering is done, the rest of the packet is received and stored in the receive queue. The last 2 bytes of the PHY packet are the MAC footer, or FCS, which is a checked CRC. The CRC is stored in the queue only if rxConfig.bIncludeCrc is 1.

The status of the received frame depends on the frame filtering result and the CRC result. Two status bits, bCrcErr and bIgnore, must be maintained. If configured, these 2 bits are present in the Status byte of the RX queue entry. The bCrcErr bit is 1 if the frame had a CRC error, and 0 otherwise. The bIgnore bit is 1 if frame filtering is enabled and the frame was rejected by frame filtering, and 0 otherwise.

NOTE: If frameFiltOpt.frameFiltStop is 1, frames with bIgnore equal to 1 are never observed, because the reception is stopped and the received bytes are not stored in the queue. If rxConfig.bAutoFlushCrc is 1, packets with bCrcErr equal to 1 are removed from the queue after reception; if rxConfig.bAutoFlushIgn is 1, packets with bIgnore equal to 1 are removed from the queue after reception.

After a packet has been received, an interrupt is raised and one of the counters in pOutput is incremented. Table 23-80 lists these conditions.

Table 23-80. Conditions for Incrementing Counters and Raising Interrupts for RX Operation

Condition	Counter Incremented	Interrupt Generated
Frame received with CRC OK and frame filtering disabled	nRxData	RX_OK
Frame received with CRC error	nRxNok	RX_NOK
Frame received that did not fit in the RX queue	nRxBufFull	RX_BUF_FULL
Beacon frame received with CRC OK and bIgnore = 0	nRxBeacon	RX_OK

Table 23-80. Conditions for Incrementing Counters and Raising Interrupts for RX Operation (continued)

Condition	Counter Incremented	Interrupt Generated
ACK frame received with CRC OK and bIgnore = 0	nRxAck	RX_OK
Data frame received with CRC OK and bIgnore = 0	nRxData	RX_OK
MAC command frame received with CRC OK and bIgnore = 0	nRxMacCmd	RX_OK
Frame with reserved frame type received with CRC OK and bIgnore = 0	nRxReserved	RX_OK
Frame received with CRC OK and bIgnore = 1	nRxIgnored	RX_IGNORED
The first RX data entry in the RX queue changed state to finished	—	RX_ENTRY_DONE

When a frame has been received, the RSSI observed while receiving the frame is written to pOutput->lastRssi. If the frame was a beacon frame accepted by the frame filtering and with CRC OK, the timestamp at the beginning of the frame is written to pOutput->beaconTimeStamp. If the timestamp is appended to the RX entry element (see Section 23.6.3.1), these two timestamps are the same for a beacon frame.

After a packet has been received, the radio CPU either restarts sync search or sends an acknowledgment frame. The conditions for the latter are as given in Section 23.5.4.1.3.

23.5.4.1.3 ACK Transmission

After a packet has been received, the radio CPU initiates transmission of an acknowledgment frame, given that all of the following conditions are met:

- Auto ACK is enabled by frameFiltOpt.autoAckEn = 1.
- The frame is accepted by frame filtering (bIgnore = 0).
- The frame is a data frame or a MAC command frame.
- The destination address is not the broadcast address.
- The ACK request bit of the FCF is set.
- The CRC check is passed (bCrcErr = 0).
- The frame fits in the receive queue.

The transmit time of the ACK packet is timed by the radio CPU, depending on frameFiltOpt.slottedAckEn. If this bit is 0, the ACK packet is transmitted 192 µs after the end of the received packet. Otherwise, slotted ACK is used. Assume that the received packet started on a backoff-slot boundary. The ACK frame then starts a whole number of backoff periods later than the start of the received frame, at the first backoff boundary following at least one TurnaroundTime-symbol period after the end of the received frame.

The contents of the automatically transmitted ACK frame are as follows:

- The PHY header is 0x05.
- The PHY payload consists of a 3-byte MAC header and a 2-byte MAC footer.
- The MAC header starts with the 2-byte FCF with the following fields:
  - The Frame Type subfield is 010b.
  - The Frame Pending subfield is set as described in Section 23.5.4.1.1.2.
  - The remaining subfields are set to all 0s.
- The next byte in the MAC header is the sequence number, which is set equal to the sequence number of the received frame.
- The MAC footer is the FCS, which is calculated automatically.

After the ACK frame has been transmitted, a TX_ACK interrupt is raised. The radio CPU then enables the receiver again.

23.5.4.1.4 End of Receive Operation

The receive operation can end as a result of the end trigger given by endTrigger and endTime, or by a command. The commands that can end the receive operation are the immediate commands CMD_ABORT and CMD_STOP, and the foreground-level radio operation command CMD_IEEE_ABORT_BG. The end-trigger and the CMD_STOP command cause the receiver to keep running until the end of the frame, or until the reception would otherwise be stopped if observed while a packet was being received. The CMD_ABORT and CMD_IEEE_ABORT_BG commands cause the receiver to stop as quickly as the implementation allows.

A receive operation ends through one of the causes listed in Table 23-81. The status field of the command structure after the command has ended indicates the reason why the operation ended. In all cases, a COMMAND_DONE interrupt is raised. In each case, the result is indicated as TRUE, FALSE, or ABORT. This decides whether to start the next command (if any) indicated in pNextOp, or to return to an IDLE state. Before the receive operation ends, the radio CPU writes the maximum observed RSSI during the receive operation to pOutput->maxRssi.

If a transmit operation is started in the foreground, the receive operation is suspended. The receiver stops as when aborted, but the synthesizer is left on to the extent possible when switching to transmit mode. When the receiver has stopped, the status field of the command structure is set to IEEE_SUSPENDED. When the transmit command is done, the receiver restarts and the status field of the command structure is reset to RUNNING.

Table 23-81. End of Receive Operation

Condition	Status Code	Result
Observed end trigger and finished any ongoing reception	IEEE_DONE_OK	TRUE
Received CMD_STOP	IEEE_DONE_STOPPED	FALSE
Received CMD_ABORT or CMD_IEEE_ABORT_BG	IEEE_DONE_ABORT	ABORT
Observed illegal parameter	IEEE_ERROR_PAR	ABORT

23.5.4.1.5 CCA Monitoring

While the receiver is running, the radio CPU monitors some signals for use in clear-channel assessment. This monitoring is controlled by ccaOpt. There are three sources for CCA: RSSI above level (ccaEnergy), carrier sense based on the correlation value (ccaCorr), and carrier sense based on sync found (ccaSync). Each of these may have the state BUSY, IDLE, or INVALID.

The RSSI above-level is maintained by monitoring the RSSI. If the RSSI is greater than or equal to ccaRssiThr, ccaEnergy is busy. If the RSSI is smaller than ccaRssiThr, ccaEnergy is IDLE. When an RSSI calculation has not yet been completed because the receiver started, ccaEnergy is INVALID.

The carrier-sense monitoring based on correlation value uses correlation peaks as defined for use in the SFD search algorithm in the receiver. If the number of correlation peaks observed in the last 8-symbol periods (32 µs) is greater than ccaOpt.corrThr, ccaCorr is BUSY; otherwise, ccaCorr is IDLE. The value of ccaOpt.corrThr can be from 0 to 3. While the receiver is receiving a frame, ccaCorr is BUSY regardless of the observed correlation peaks. If the time since the receiver started is less than 8 symbol periods and the number of correlation peaks observed since the receiver started is less than or equal to ccaOpt.corrThr, ccaCorr is INVALID.

The carrier-sense monitoring based on sync found is maintained by the radio CPU as follows. If sync is obtained on the receiver, the radio CPU checks the PHY header to find the frame length. The radio CPU considers the channel to be busy for the duration of this frame. This check is done even if reception of the frame is stopped due to the frame filtering and sync search is restarted. If sync is found again while the channel is viewed as BUSY, the channel is viewed as BUSY until both these frames have ended according to the observed frame lengths. The INVALID state is not used for ccaSync.

If the radio is transmitting an ACK or is suspended for running a TX operation, ccaEnergy, ccaCorr, and ccaSync are all BUSY.

The overall CCA state ccaState depends on the ccaEnEnergy, ccaEnCorr, and ccaEnSync bits of ccaOpt together with the ccaCorrOp and ccaSyncOp bits. The following rules apply for finding the ccaState (ccaTmp is a helper state in the description):

- If ccaEnEnergy = 0 and ccaEnCorr = 0 and ccaEnSync = 0, then ccaState = IDLE
- If ccaEnEnergy = 1 and ccaEnCorr = 0, then ccaTmp = ccaEnergy
- If ccaEnEnergy = 0 and ccaEnCorr = 1, then ccaTmp = ccaCorr
- If ccaEnEnergy = 1 and ccaEnCorr = 1 and ccaCorrOp = 0, then:
  - If either ccaEnergy or ccaCorr is BUSY, then ccaTmp = BUSY
  - Otherwise, if either ccaEnergy or ccaCorr is INVALID, then ccaTmp = INVALID
  - Otherwise, ccaTmp = IDLE
- If ccaEnEnergy = 1 and ccaEnCorr = 1 and ccaCorrOp = 1, then:
  - If either ccaEnergy or ccaCorr is IDLE, then ccaTmp = IDLE
  - Otherwise, if either ccaEnergy or ccaCorr is Invalid, then ccaTmp = INVALID
  - Otherwise, ccaTmp = BUSY
- If ccaEnEnergy = 0 and ccaEnCorr = 0 and ccaEnSync = 1, then ccaState = ccaSync
- Otherwise, if ccaEnSync = 1 and ccaSyncOp = 0, then:
  - If either ccaTmp or ccaSync is BUSY, then ccaState = BUSY
  - Otherwise, if ccaTmp is Invalid, then ccaState = INVALID
  - Otherwise, ccaState = IDLE
- Otherwise, if ccaEnSync = 1 and ccaSyncOp = 1, then:
  - If either ccaTmp or ccaSync is IDLE, then ccaState = IDLE
  - Otherwise, if ccaTmp is INVALID, then ccaState = INVALID
  - Otherwise, ccaState = BUSY

The ccaSync CCA state is required to be IDLE for the overall CCA state to be IDLE, according to the IEEE 802.15.4 standard. Thus, to comply, ccaEnSync is 1 and cceSyncOp is 0.

CCA mode 1, as defined in the IEEE 802.15.4 standard, is implemented by setting ccaEnEnergy = 1 and ccaEnCorr = 0. CCA mode 2 is implemented by setting ccaEnEnergy = 0 and ccaEnCorr = 1. CCA mode 3 is implemented by setting ccaEnEnergy = 1 and ccaEnCorr = 1. With CCA mode 3, ccaCorrOp is allowed to be either 0 or 1; this distinguishes between the logical operator AND (1) and OR (0) as described in the IEEE 802.15.4 standard.

The CCA states and the current RSSI can be read by the system CPU by issuing the immediate command CMD_IEEE_CCA_REQ. If a CMD_IEEE_CSMA operation is running in the foreground, the radio CPU also monitors the CCA autonomously.

23.5.4.2 Energy Detect Scan Operation

The energy detect scan radio operation is a background-level operation that starts with the CMD_IEEE_ED_SCAN command and uses a command structure as given in Table 23-60.

At the start of an RX operation, the radio CPU waits for the start trigger, then programs the frequency based on the channel parameter. If the channel is 0xFF, the operation keeps running on an already-configured channel. This requires that the operation follows another receive operation or a synthesizer programming operation. If the frequency synthesizer is not running, the operation ends with an error. After programming the frequency, the radio CPU configures the receiver to receive IEEE 802.15.4 packets, but it does not store any received data.

While the receiver is running, CCA is updated as described in Section 23.5.4.1.5. When the demodulator obtains sync on a frame, the PHY header is read. This is used only to determine the carrier sense based on sync found, and sync search restarts immediately afterwards.

The energy detect scan operation ends under the same conditions as the RX operation, as described in Section 23.5.4.1.4. Before the operation ends, the radio CPU writes the maximum-observed RSSI during the energy detect scan operation to maxRssi.

23.5.4.3 CSMA-CA Operation

The CSMA-CA operation is a foreground-level operation that runs on top of a receive or energy-detect scan operation. If run on top of an energy-detect scan operation, this does not perform the energy-detect scan procedure, but starts a receiver without having to receive packets. This operation starts with the CMD_IEEE_CSMA command, and uses the command structure given in Table 23-61.

At the start of a CSMA-CA operation, the radio CPU waits for the start trigger.

The radio CPU maintains a variable CW, which initializes to csmaConfig.initCW.

If remainingPeriods is nonzero at the start of the command, the radio CPU delays for that number of backoff periods (default 320 µs) measured from the start trigger before proceeding. Otherwise, the radio CPU draws a pseudo-random number in the range 0 to $2^{(BE)-1}$, where BE is given by (Table 23-61). The radio CPU then waits that number of backoff periods from the start trigger before proceeding.

After this wait time, the radio CPU checks the CCA state from the background-level operation, as described in Section 23.5.4.1.5. If the CCA state was INVALID, the radio CPU waits before trying again. If csmaConfig.bSlotted = 1, the wait is for one backoff period, otherwise it waits until an RSSI result is available. If the CCA state was IDLE, the radio CPU decrements CW by 1, and if this results in a value of 0, the CSMA-CA operation is successful. If this results in a nonzero value, the radio CPU waits one backoff period timed from the end of the wait time, and then checks the CCA state again as described previously.

If the channel was BUSY when the CCA state was checked, the radio CPU updates the variables as follows:

CW = csmaConfig.initCW; NB + = 1; BE + = min (BE + 1, macMaxBE);

If NB after this update is greater than macMaxCSMABackoffs, the CSMA-CA operation ends with failure. Otherwise, the radio CPU draws a random number of backoff periods to wait as described previously, and proceeds as before. If csmaConfig.bSlotted = 1, the wait is from the next backoff period after the end of the previous wait time; otherwise, the wait is from a configurable time after the end of the previous wait time.

Figure 23-7 shows the flow chart for the CSMA-CA operation.

In addition to the CSMA-CA operation ending with success or failure as previously described, the operation can end as a result of the end trigger given by endTrigger and endTime, or by a command. The commands that can end the CSMA-CA operation are the immediate commands CMD_ABORT, CMD_STOP, CMD_IEEE_ABORT_FG, and CMD_IEEE_STOP_FG. When the CSMA-CA operation ends, the radio CPU writes lastTimeStamp with the timer value at the end of the most recent wait period before a CCA check was done, and lastRssi with the RSSI value at that time. If the operation ended because of a time-out or stop command, the radio CPU writes remainingPeriods with the number of backoff periods remaining of the wait time. Otherwise, the radio CPU writes remainingPeriods to 0.

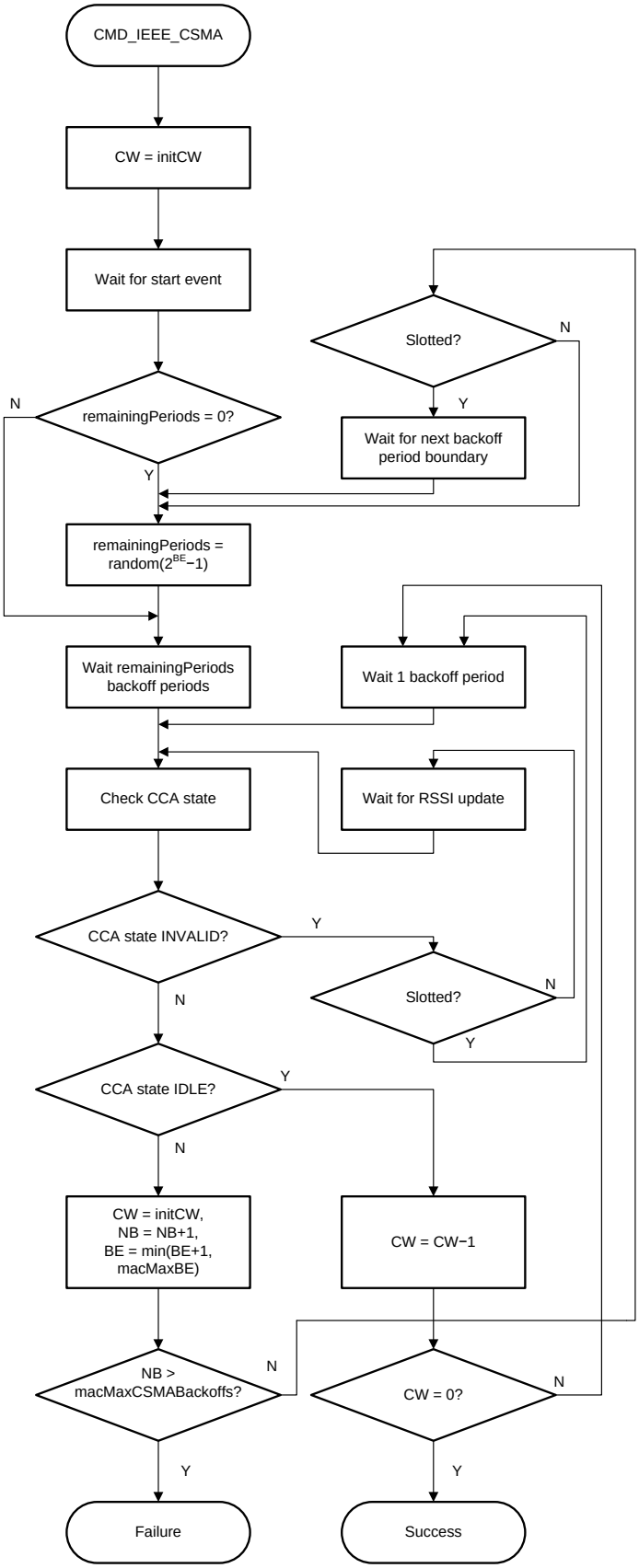
The pseudo-random algorithm is based on a maximum-length 16-bit linear-feedback shift register (LFSR). The seed is as provided in randomState. When the operation ends, the radio CPU writes the current state back to this field. If randomState is 0, the radio CPU self-seeds by initializing the LFSR to the 16 LSBs of the RAT. There is some randomness to this value, but this is limited, especially for slotted CSMA-CA, and seeding with a true-random number (or a pseudo-random number based on a true-random seed) by the system CPU is therefore recommended. If the 16 LSBs of the RAT are all 0, another fixed value is substituted.

Depending on csmaConfig.rxOffMode, the underlying RX operation may be suspended during the backoff before another CCA check, if there is enough time for it. The different values have the following meaning:

- rxOffMode = 0: The radio stays on during CSMA backoffs.
- rxOffMode = 1: If a frame is being received, an ACK being transmitted, or in the transition between those, the radio stays on. Otherwise, the radio switches off until the end of the backoff period.
- rxOffMode = 2: If a frame is being received, an ACK is being transmitted, or is in the transition between those, the radio stays on until the packet has been fully received and the ACK has been transmitted if applicable. After that, the radio switches off until the end of the backoff period.
- rxOffMode = 3: The radio switches off immediately at the beginning of a backoff period. This aborts a frame being received or an ACK being transmitted. The radio remains switched off until the end of the backoff period.

If the radio switches off this way, the receiver restarts sufficiently early for the next CCA operation to be done, and the radio only switches off it there is sufficient time. This feature can be used for power saving in systems that do not always need to be in RX. All modes except mode 0 may cause frames to be lost, at increasing probability.

Figure 23-7. Flow Chart for CSMA-CA Operation



For operation according to IEEE 802.15.4, the parameters must be initialized as follows before starting a new CSMA-CA operation:

- randomState must be set to a random value.
- csmaConfig.initCW must be set to 2 for slotted CSMA-CA and 1 for unslotted CSMA-CA.
- csmaConfig.bSlotted must be set to 1 for slotted CSMA-CA and 0 for unslotted CSMA-CA.
- NB must be set to 0.
- BE must be set to macMinBE, except for slotted CSMA-CA with battery-life extension, where BE must be set to min (2, macMinBE).
- remainingPeriods must be set to 0.
- macMaxBE and macMaxCSMABackoffs must be set to their corresponding MAC PIB attribute.

For slotted CSMA-CS, startTrigger must be set up to occur on a backoff-slot boundary. For slotted CSMA-CA, the endTrigger must be set up to occur at the latest time that the transaction can be completed within the superframe, as specified in the IEEE 802.15.4 standard. If the CSMA-CA ends due to time-out, the CSMA can be restarted without modifying the parameters (except possibly the end time) at the next superframe.

Table 23-82 lists the causes of a CSMA-CA operation end. After the command has ended, the status field of the command structure (2 status bytes listed in Table 23-8) indicates why the operation ended. In all cases, an FG_COMMAND_DONE interrupt is raised. In each case, it is indicated if the result is TRUE, FALSE, or ABORT. This result indicates whether to start the next command (if any) in pNextOp, or to return to an IDLE state.

Table 23-82. End of CSMA-CA Operation

Condition	Status Code	Result
CSMA-CA operation finished with success	IEEE_DONE_OK	TRUE
CSMA-CA operation finished with failure	IEEE_DONE_BUSY	FALSE
End trigger occurred	IEEE_DONE_TIMEOUT	FALSE
Received CMD_STOP or CMD_IEEE_STOP_FG	IEEE_DONE_STOPPED	FALSE
Received CMD_ABORT or CMD_IEEE_ABORT_FG	IEEE_DONE_ABORT	ABORT
Background operation ended	IEEE_DONE_BGEND	ABORT
Observed illegal parameter	IEEE_ERROR_PAR	ABORT

When the operation ends, the time of the last CCA check (that is, the time written into lastTimeStamp) is defined as event 1, and may be used for timing subsequent chained operations.

23.5.4.4 Transmit Operation

The transmit operation is a foreground-level operation that transmits one packet. The operation is started with the CMD_IEEE_TX command, and uses the command structure given in Table 23-62.

When the radio CPU receives the command, it waits for the start trigger. Any background-level operation keeps running during this wait time. At the start trigger, the radio CPU suspends the receiver and configures the transmitter. The synthesizer should be powered and calibrated. Therefore, if no background-level operation is running, a calibrate synthesizer command must precede the TX operation. If the frequency synthesizer is not running, the operation ends with an error.

The transmitter transmits the payload found in the buffer pointer to pPayload, which consists of payloadLen bytes. If txOpt.payloadLenMsb is nonzero, this field is multiplied by 256 and added to payloadLen to create (for test purposes) a long frame that is not compliant with IEEE 802.15.4. If txOpt.bIncludePhyHdr is 0, the radio CPU inserts a PHY header automatically, calculated from the payload length. Otherwise, no PHY header is inserted by the radio CPU, so for IEEE 802.15.4 compliance, the first byte in the payload buffer must be the PHY header. The payload is then transmitted as found in the payload buffer. If txOpt.bIncludeCrc is 0, the radio CPU appends two CRC bytes, calculated according to the IEEE 802.15.4 standard. Otherwise, no CRC is appended, so for IEEE 802.15.4 MAC compliance, the last 2 bytes in the payload buffer must be the MAC footer. The transmit operation can be ended by one of the immediate commands CMD_ABORT, CMD_STOP,

CMD_IEEE_ABORT_FG, or CMD_IEEE_STOP_FG. If CMD_ABORT or CMD_IEEE_ABORT_FG is received, the transmission ends as soon as possible in the middle of the packet. If CMD_STOP or CMD_IEEE_STOP_BG is received while the radio CPU is waiting for the start trigger, the operation ends without any transmission; otherwise, the transmission is finished, but the end status and result differ as explained in the following.

When transmission of the packet starts, the trigger RAT time used for starting the modem is written to the timeStamp field by the radio CPU. This timestamp is delayed by the firmware-defined parameter startToTXRatOffset, compared to the configured start time of the CMD_IEEE_TX command. If the transmitter and receiver have synchronized RAT timers, this timestamp is the same as the timestamp appended to the RX entry element, as in Section 23.6.3.1, although with estimation uncertainty on the receiver side.

When the operation ends, the end time of the transmitted frame is defined as event 1, and may be used for timing subsequent chained operations.

Table 23-83 lists the causes of a transmit operation end. After the command has ended, the status field of the command structure (2 status bytes listed in Table 23-8) indicates why the operation ended. In all cases, an FG_COMMAND_DONE interrupt is raised. In each case, it is indicated if the result is TRUE, FALSE, or ABORT. This indicates whether to start the next command (if any) in pNextOp, or to return to an IDLE state.

Table 23-83. End of Transmit Operation

Condition	Status Code	Result
Packet transmitted	IEEE_DONE_OK	TRUE
Received CMD_STOP or CMD_IEEE_STOP_FG, then finished transmitting if started	IEEE_DONE_STOPPED	FALSE
Received CMD_ABORT or CMD_IEEE_ABORT_FG	IEEE_DONE_ABORT	ABORT
Observed illegal parameter	IEEE_ERROR_PAR	ABORT

23.5.4.5 Receive Acknowledgment Operation

The receive-ACK operation is a foreground-level operation that runs on top of a receive operation. The operation starts with the CMD_IEEE_RX_ACK command, and uses the command structure listed in Table 23-63.

At the start of a receive-ACK operation, the radio CPU waits for the start trigger. If the receiver was suspended due to a TX operation before the receive-ACK operation, the background-level RX operation is not resumed until the start trigger occurs.

While the receive-ACK operation is running, the background-level RX operation runs normally. However, in addition to looking for the packets, the operation looks for ACK packets with the sequence number given in seqNo. The packet is stored in the receive queue only if configured in the background-level receive operation (frameTypes.bAcceptFt2Ack = 1). If ACK packets are filtered out in the background RX operation, for an ACK packet the sequence number is received, and if it matches, also the FCS.

If the ACK packet with the requested sequence number is received, the FCS is checked. If the CRC is OK, the receive-ACK operation ends, otherwise it continues. If the ACK is received OK, the pending-data bit of the header is checked.

In addition to the receive-ACK operation ending after receiving the ACK as described previously, the operation can end as a result of the end trigger given by endTrigger and endTime, or by a command. The commands that can end the receive-ACK operation are the immediate commands CMD_ABORT, CMD_STOP, CMD_IEEE_ABORT_FG, and CMD_IEEE_STOP_FG.

A receive-ACK operation ends due to one of the causes listed in Table 23-84. After the command has ended, the status field of the command structure (2 status bytes listed in Table 23-8) indicates why the operation ended. In all cases, an FG_COMMAND_DONE interrupt is raised. In each case, it is indicated if the result is TRUE, FALSE, or ABORT. This indicates whether to start the next command (if any) in pNextOp, or to return to an IDLE state.

Table 23-84. End of Receive ACK Operation

Condition	Status Code	Result
Requested ACK successfully received with pending data bit cleared	IEEE_DONE_ACK	FALSE
Requested ACK successfully received with pending data bit set	IEEE_DONE_ACKPEND	TRUE
End trigger occurred	IEEE_DONE_TIMEOUT	FALSE
Received CMD_STOP or CMD_IEEE_STOP_FG	IEEE_DONE_STOPPED	FALSE
Received CMD_ABORT or CMD_IEEE_ABORT_FG	IEEE_DONE_ABORT	ABORT
Background operation ended	IEEE_DONE_BGEND	ABORT
Observed illegal parameter	IEEE_ERROR_PAR	ABORT

23.5.4.6 Abort Background-Level Operation Command

The abort background-level operation command is a foreground-level command that stops the command running in the background. The abort background-level operation command is defined as a foreground-operation command so that it has a start time, and so that it can be chained with other foreground-operation commands. The command is executed with the CMD_IEEE_ABORT_BG command and uses a command structure with only the minimum set of parameters.

At the start of an abort background-level operation, the radio CPU waits for the start trigger, then aborts the ongoing background-level receive or energy-detect scan operation.

The operation may be stopped by a command while waiting for the start trigger. The commands that can stop the operation are CMD_ABORT, CMD_STOP, CMD_IEEE_ABORT_FG, and CMD_IEEE_STOP_FG. The first two commands cause the background-level operation to stop regardless.

An abort background-level operation ends due to one of the causes listed in Table 23-85. After the command has ended, the status field of the command structure (2 status bytes listed in Table 23-8) indicates why the operation ended. In all cases, an FG_COMMAND_DONE interrupt is raised. In each case, it is indicated if the result is TRUE, FALSE, or ABORT. This indicates whether to start the next command (if any) in pNextOp, or to return to an IDLE state.

Table 23-85. End of ABORT Background-Level Operation

Condition	Status Code	Result
Background level aborted	IEEE_DONE_OK	TRUE
Received CMD_STOP or CMD_IEEE_STOP_FG	IEEE_DONE_STOPPED	FALSE
Received CMD_ABORT or CMD_IEEEE_ABORT_FG	IEEE_DONE_ABORT	ABORT

23.5.5 Immediate Commands

23.5.5.1 Modify CCA Parameter Command

The CMD_IEEE_MOD_CCA command takes a command structure as defined in Table 23-64.

CMD_IEEE_MOD_CCA must only be sent while an RX or energy-detect scan operation is running. On reception, the radio CPU modifies the values of ccaRssiThr and ccaOpt for the running process into the values given by newCcaRssiThr and newCcaOpt, respectively. The radio CPU updates the command structure. The new settings are used for future CCA requests.

If the command is issued without an active or suspended background-level operation, the radio CPU returns the result ContextError in CMDSTA. If any of the parameters entered are illegal, the radio CPU returns the result ParError in CMDSTA. Otherwise, the radio CPU returns DONE.

23.5.5.2 Modify Frame-Filtering Parameter Command

The CMD_IEEE_MOD_FILT command takes a command structure as defined in Table 23-65.

CMD_IEEE_MOD_FILT must be sent only while an RX operation is running. On reception, the radio CPU modifies the values of frameFiltOpt and frameTypes for the running process into the values given by newFrameFiltOpt and newFrameTypes, respectively. The radio CPU updates the command structure.

The new values of the frame-filtering options are used from the next time frame filtering is started. If autoAckEn or slottedAckEn are changed, the change applies from the next time reception of a packet ends.

If the command is issued without an active or suspended background-level RX operation, the radio CPU returns the result ContextError in CMDSTA. If any of the parameters entered are illegal, the radio CPU returns the result ParError in CMDSTA. Otherwise, the radio CPU returns DONE.

23.5.5.3 Enable or Disable Source Matching Entry Command

The CMD_IEEE_MOD_SRC_MATCH command takes a command structure as defined in Table 23-65.

CMD_IEEE_MOD_SRC_MATCH must be sent only while an RX operation is running. On reception, the radio CPU enables or disables the source-matching entry signaled in the command structure. If options.entryType is 0, the entry is extended-address entry in the structure pointed to by pExtEntryList, and if options.entryType is 1, the entry is short-address entry in the structure pointed to by pShortEntryList. The index of the entry is signaled in entryNo. If options.bEnable is 0, the entry is disabled, and if it is 1, the entry is enabled. The corresponding source pending bit is set to the value of options.srcMatch.

The new values of the enable values are used from the next time source-matching is performed. The system CPU may modify the address of a disabled entry, but not an enabled one.

If the command is issued without an active or suspended background-level RX operation, the radio CPU returns the result ContextError in CMDSTA. If any of the parameters entered are illegal, for example, pointing to a nonexistent entry, the radio CPU returns the result ParError in CMDSTA. Otherwise, the radio CPU returns DONE.

23.5.5.4 Abort Foreground-Level Operation Command

CMD_IEEE_ABORT_FG is an immediate command that takes no parameters, and can thus be used as a direct command.

The CMD_IEEE_ABORT_FG command aborts the foreground-level operation while the background-level operation continues to run. For more detail, see the description of the foreground-level operations in Table 23-57.

If no foreground-level radio operation command is running, no action is taken. The result signaled in CMDSTA is DONE in all cases. If a foreground-level radio operation command was running, CMDSTA may be updated before the radio operation has ended.

23.5.5.5 Stop Foreground-Level Operation Command

CMD_IEEE_STOP_FG is an immediate command that takes no parameters, and can thus be used as a direct command.

The CMD_IEEE_STOP_FG command causes the foreground-level operation to stop gracefully, while the background-level operation continues to run. For more detail, see the description of the foreground-level operations in Table 23-57.

If no foreground-level radio operation command is running, no action is taken. The result signaled in CMDSTA is DONE in all cases. If a foreground-level radio operation command was running, CMDSTA may be updated before the radio operation has ended.

23.5.5.6 Request CCA and RSSI Information Command

The CMD_IEEE_CCA_REQ command takes a command structure as defined in Table 23-67.

CMD_IEEE_CCA_REQ must be sent only while an RX or energy-detect scan operation is running. On reception, the radio CPU writes the following figures back into the command structure:

- currentRssi is set to the RSSI number currently available from the demodulator.
- maxRssi is set to the maximum RSSI observed because the background-level operation was started.
- ccaState is set to the CCA state according to the current CCA options (see Section 23.5.4.1.5).
- ccaEnergy is set to the energy-detect CCA state, according to Section 23.5.4.1.5.
- ccaCorr is set to the correlator-based carrier-sense CCA state, according to Section 23.5.4.1.5.
- ccaSync is set to the sync found-based carrier-sense CCA state, according to Section 23.5.4.1.5.

If no valid RSSI is found when the request is sent, the currentRssi and maxRssi returned indicate this by using a special value (0x80).

If the command is issued without an active or suspended background-level RX operation, the radio CPU returns the result ContextError in CMDSTA. Otherwise, the radio CPU returns DONE.

23.6 Bluetooth low energy

This section describes Bluetooth low-energy-specific command structure, data handling, radio operation commands, and immediate commands.

23.6.1 Bluetooth low energy Commands

Table 23-86 defines the Bluetooth low-energy-specific radio operation commands.

Table 23-86. Bluetooth low energy Radio Operation Commands

ID	Command Name	Description
0x1801	CMD_BLE_SLAVE	Start slave operation
0x1802	CMD_BLE_MASTER	Start master operation
0x1803	CMD_BLE_ADV	Start connectable undirected advertiser operation
0x1804	CMD_BLE_ADV_DIR	Start connectable directed advertiser operation
0x1805	CMD_BLE_ADV_NC	Start the not-connectable advertiser operation
0x1806	CMD_BLE_ADV_SCAN	Start scannable undirected advertiser operation
0x1807	CMD_BLE_SCANNER	Start scanner operation
0x1808	CMD_BLE_INITIATOR	Start initiator operation
0x1809	CMD_BLE_GENERIC_RX	Receive generic packets (used for PHY test or packet sniffing)
0x180A	CMD_BLE_TX_TEST	Transmit PHY test packets

Table 23-87 defines the Bluetooth low-energy-specific immediate command.

Table 23-87. Bluetooth low energy Immediate Command

ID	Command Name	Description
0x1001	CMD_BLE_ADV_PAYLOAD	Modify payload used in advertiser operations

23.6.1.1 Command Data Definitions

This section defines data types that describe the data structures used to communicate between the system CPU and the radio CPU. The data structures are listed with tables. The Byte Index is the offset from the pointer to that structure. Multibyte fields are little-endian, and halfword or word alignment is required. For bit numbering, 0 is the LSB. The R/W column is used as follows:

- R: The system CPU can read a result back; the radio CPU does not read the field.
- W: The system CPU writes a value; the radio CPU reads it and does not modify the value.
- R/W: The system CPU writes an initial value; the radio CPU may modify the initial value.

23.6.1.1.1 Bluetooth low energy Command Structures

Table 23-88. Bluetooth low energy Radio Operation Command Structure (1)

Byte Index	Field Name	Bits	Bit Field Name	Type	Description
14	channel			W	Channel to use: 0–39: Bluetooth low energy advertising/data channel number 60–207: Custom frequency; (2300 + channel) MHz 255: Use existing frequency Others: reserved
15	whitening	0–6	init	W	If bOverride = 1 or custom frequency is used: 0: Do not use whitening Other value: Initialization for 7-bit LFSR whitener
		7	bOverride	W	0: Use default whitening for Bluetooth low energy advertising/data channels 1: Override whitening initialization with value of init
16–19	pParams			W	Pointer to command-specific parameter list
20–23	pOutput			W	Pointer to command-specific result (NULL: Do not store results)

(1) This command structure is used for all the radio operation commands for Bluetooth low energy support. Table 23-8 defines the first 14 bytes.

Table 23-89. Update Advertising Payload Command

Byte Index	Field Name	Type	Description
0–1	commandNo	W	The command number
2	payloadType	W	0: Advertising data 1: Scan response data
3	newLen	W	Length of the new payload
4–7	pNewData	W	Pointer to the buffer containing the new data
8–11	pParams	W	Pointer to the parameter structure to update

23.6.1.2 Parameter Structures

Table 23-90. Slave Commands

Byte Index	Field Name	Type	Description
0–3	pRxQ	W	Pointer to receive queue
4–7	pTxQ	W	Pointer to transmit queue
8	rxConfig	W	Configuration bits for the receive queue entries (see Table 23-103 for details)
9	seqStat	R/W	Sequence number status (see Table 23-70 for details)
10	maxNack	W	Maximum number of NACKs received before operation ends. 0: No limit
11	maxPkt	W	Maximum number of packets transmitted in the operation before it ends. 0: No limit
12–15	accessAddress	W	Access address used on the connection
16–18	crcInit	W	CRC initialization value used on the connection
19	timeoutTrigger	W	Trigger that defines time-out of the first receive operation
20–23	timeoutTime	W	Time parameter for timeoutTrigger
24–26			Reserved
27	endTrigger	W	Trigger that causes the device to end the connection event as soon as allowed
28–31	endTime	W	Time parameter for endTrigger

Table 23-91. Master Commands

Byte Index	Field Name	Type	Description
0–3	pRxQ	W	Pointer to receive queue
4–7	pTxQ	W	Pointer to transmit queue
8	rxConfig	W	Configuration bits for the receive queue entries (see Table 23-103 for details)
9	seqStat	R/W	Sequence number status (see Table 23-70 for details)
10	maxNack	W	Maximum number of NACKs received before operation ends. 0: No limit
11	maxPkt	W	Maximum number of packets transmitted in the operation before it ends. 0: No limit
12–15	accessAddress	W	Access address used on the connection
16–18	crcInit	W	CRC initialization value used on the connection
19	endTrigger	W	Trigger that causes the device to end the connection event as soon as allowed
20–23	endTime	W	Time parameter for endTrigger

Table 23-92. Advertiser Commands

Byte Index	Field Name	Bits	Bit Field Name	Type	Description
0–3	pRxQ			W	Pointer to receive queue
4	rxConfig			W	Configuration bits for the receive queue entries (see Table 23-103 for details)
5	advConfig	0–1	advFilterPolicy	W	The advertiser filter policy
		2	deviceAddrType	W	The type of the device address: public (0) or random (1)
		3	peerAddrType	W	Directed advertiser: The type of the peer address: public (0) or random (1)
		4	bStrictLenFilter	W	1: Discard messages with illegal length
6	advLen			W	Size of advertiser data
7	scanRspLen			W	Size of scan response data

Table 23-92. Advertiser Commands (continued)

Byte Index	Field Name	Bits	Bit Field Name	Type	Description
8–11	pAdvData			W	Pointer to buffer containing ADV*_IND data
12–15	pScanRspData			W	Pointer to buffer containing SCAN_RSP data
16–19	pDeviceAddress			W	Pointer to device address used for this device
20–23	pWhiteList			W	Pointer to white list or peer address (directed advertiser)
24–26					Reserved
27	endTrigger			W	Trigger that causes the device to end the advertiser event as soon as allowed
28–31	endTime			W	Time parameter for endTrigger

Table 23-93. Scanner Command

Byte Index	Field Name	Bits	Bit Field Name	Type	Description
0–3	pRxQ			W	Pointer to receive queue
4	rxConfig			W	Configuration bits for the receive queue entries (see Table 23-103 for details)
5	scanConfig	0	scanFilterPolicy	W	The scanner filter policy
		1	bActiveScan	W	0: Passive scan 1: Active scan
		2	deviceAddrType	W	The type of the device address – public (0) or random (1)
		3			Reserved
		4	bStrictLenFilter	W	1: Discard messages with illegal length
		5	bAutoWlIgnore	W	1: Automatically set ignore bit in white list
		6	bEndOnRpt	W	1: End scanner operation after each reported ADV*_IND and potentially SCAN_RSP
6–7	randomState			R/W	State for pseudo-random number generation used in backoff procedure
8–9	backoffCount			R/W	Parameter backoffCount used in backoff procedure
10	backoffPar	0–3	logUpperLimit	R/W	Binary logarithm of parameter upperLimit used in scanner backoff procedure
		4	bLastSucceeded	R/W	1 if the last SCAN_RSP was successfully received and upperLimit not changed
		5	bLastFailed	R/W	1 if reception of the last SCAN_RSP failed and upperLimit was not changed
11	scanReqLen			W	Size of scan request data
12–15	pScanReqData			W	Pointer to buffer containing SCAN_REQ data
16–19	pDeviceAddress			W	Pointer to device address used for this device
20–23	pWhiteList			W	Pointer to white list
24–25					Reserved
26	timeoutTrigger			W	Trigger that causes the device to stop receiving as soon as allowed
27	endTrigger			W	Trigger that causes the device to stop receiving as soon as allowed
28–31	timeoutTime			W	Time parameter for timeoutTrigger
32–35	endTime			W	Time parameter for endTrigger

Table 23-94. Initiator Command

Byte Index	Field Name	Bits	Bit Field Name	Type	Description
0–3	pRxQ			W	Pointer to receive queue
4	rxConfig			W	Configuration bits for the receive queue entries (see Table 23-103 for details)
5	initConfig	0	bUseWhiteList	W	Initiator filter policy: 0: Use specific peer address. 1: Use white list
		1	bDynamicWinOffset	W	1: Use dynamic WinOffset insertion
		2	deviceAddrType	W	The type of the device address – public (0) or random (1)
		3	peerAddrType	W	The type of the peer device address – public (0) or random (1)
		4	bStrictLenFilter	W	1: Discard messages with illegal length
6					Reserved
7	connectReqLen			W	Size of connect request data
8–11	pConnectReqData			W	Pointer to buffer containing LLData to go in the CONNECT_REQ
12–15	pDeviceAddress			W	Pointer to device address used for this device
16–19	pWhiteList			W	Pointer to white list or peer address
20–23	connectTime			R/W	Indication of timer value of the first possible start time of the first connection event. Set to the calculated value if a connection is made and to the next possible connection time (see Table 23-100) if not.
24–25					Reserved
26	timeoutTrigger			W	Trigger that causes the device to stop receiving as soon as allowed
27	endTrigger			W	Trigger that causes the device to stop receiving as soon as allowed
28–31	timeoutTime			W	Time parameter for timeoutTrigger
32–35	endTime			W	Time parameter for endTrigger

Table 23-95. Generic RX Command

Byte Index	Field Name	Type	Description
0–3	pRxQ	W	Pointer to receive queue. May be NULL; if so, received packets are not stored
4	rxConfig	W	Configuration bits for the receive queue entries (see Table 23-103 for details).
5	bRepeat	W	0: End operation after receiving a packet. 1: Restart receiver after receiving a packet.
6–7			Reserved
8–11	accessAddress	W	Access address used on the connection
12–14	crcInit	W	CRC initialization value used on the connection
15	endTrigger	W	Trigger that causes the device to end the RX operation
16–19	endTime	W	Time parameter for endTrigger

Table 23-96. TX Test Command

Byte Index	Field Name	Bits	Bit Field Name	Type	Description
0–1	numPackets			W	Number of packets to transmit 0: Transmit unlimited number of packets
2	payloadLength			W	The number of payload bytes in each packet
3	packetType			W	The packet type to be used
4–7	period			W	Number of radio timer cycles between the start of each packet
8	config	0	bOverride	W	0: Use default packet encoding 1: Override packet contents
		1	bUsePrbs9	W	If bOverride is 1: 1: Use PRBS9 encoding of packet
		2	bUsePrbs15	W	If bOverride is 1: 1: Use PRBS15 encoding of packet
9	byteVal			W	If config.bOverride is 1, value of each byte to be sent
10					Reserved
11	endTrigger			W	Trigger that causes the device to end the Test TX operation
12–15	endTime			W	Time parameter for endTrigger

23.6.1.3 Output Structures

Table 23-97. Master or Slave Commands

Byte Index	Field Name	Type	Description
0	nTx	R/W	Number of packets (including automatic empty and retransmissions) transmitted
1	nTxAck	R/W	Number of transmitted packets (including automatic empty) ACKed
2	nTxCtrl	R/W	Number of unique LL control packets from the TX queue transmitted
3	nTxCtrlAck	R/W	Number of LL control packets from the TX queue finished (ACKed)
4	nTxCtrlAckAck	R/W	Number of LL control packets ACKed and where an ACK has been sent in response
5	nTxRetrans	R/W	Number of retransmissions done
6	nTxEntryDone	R/W	Number of packets from the TX queue finished (ACKed)
7	nRxOk	R/W	Number of packets received with payload, CRC OK and not ignored
8	nRxCtrl	R/W	Number of LL control packets received with CRC OK and not ignored
9	nRxCtrlAck	R/W	Number of LL control packets received with CRC OK and not ignored, and then ACKed
10	nRxNok	R/W	Number of packets received with CRC error
11	nRxIgnored	R/W	Number of packets received with CRC OK and ignored due to repeated sequence number
12	nRxEmpty	R/W	Number of packets received with CRC OK and no payload
13	nRxBufFull	R/W	Number of packets received and discarded due to lack of buffer space
14	lastRssi	R	RSSI of last received packet
15	pktStatus	R/W	Status of received packets; see Table 23-107
16–19	timeStamp	R	Slave operation: Timestamp of first received packet

Table 23-98. Advertiser Commands

Byte Index	Field Name	Type	Description
0–1	nTxAdvInd	R/W	Number of ADV*_IND packets completely transmitted
2	nTxScanRsp	R/W	Number of SCAN_RSP packets transmitted
3	nRxScanReq	R/W	Number of SCAN_REQ packets received OK and not ignored
4	nRxConnectReq	R/W	Number of CONNECT_REQ packets received OK and not ignored
5			Reserved
6–7	nRxNok	R/W	Number of packets received with CRC error
8–9	nRxIgnored	R/W	Number of packets received with CRC OK, but ignored
10	nRxBufFull	R/W	Number of packets received that did not fit in RX queue
11	lastRssi	R	The RSSI of the last received packet
12–15	timeStamp	R	Timestamp of the last received packet

Table 23-99. Scanner Command

Byte Index	Field Name	Type	Description
0–1	nTxScanReq	R/W	Number of transmitted SCAN_REQ packets
2–3	nBackedOffScanReq	R/W	Number of SCAN_REQ packets not sent due to backoff procedure
4–5	nRxAdvOk	R/W	Number of ADV*_IND packets received with CRC OK and not ignored
6–7	nRxAdvIgnored	R/W	Number of ADV*_IND packets received with CRC OK, but ignored
8–9	nRxAdvNok	R/W	Number of ADV*_IND packets received with CRC error
10–11	nRxScanRspOk	R/W	Number of SCAN_RSP packets received with CRC OK and not ignored
12–13	nRxScanRspIgnored	R/W	Number of SCAN_RSP packets received with CRC OK, but ignored
14–15	nRxScanRspNok	R/W	Number of SCAN_RSP packets received with CRC error
16	nRxAdvBufFull	R/W	Number of ADV*_IND packets received that did not fit in RX queue
17	nRxScanRspBufFull	R/W	Number of SCAN_RSP packets received that did not fit in RX queue
18	lastRssi	R	The RSSI of the last received packet
19			Reserved
20–23	timeStamp	R	Timestamp of the last successfully received ADV*_IND packet that was not ignored

Table 23-100. Initiator Command

Byte Index	Field Name	Type	Description
0	nTxConnectReq	R/W	Number of transmitted CONNECT_REQ packets
1	nRxAdvOk	R/W	Number of ADV*_IND packets received with CRC OK and not ignored
2–3	nRxAdvIgnored	R/W	Number of ADV*_IND packets received with CRC OK, but ignored
4–5	nRxAdvNok	R/W	Number of ADV*_IND packets received with CRC error
6	nRxAdvBufFull	R/W	Number of ADV*_IND packets received that did not fit in RX queue
7	lastRssi	R/W	The RSSI of the last received packet
8–11	timeStamp	R	Timestamp of the received ADV*_IND packet that caused transmission of CONNECT_REQ

Table 23-101. Generic RX Command

Byte Index	Field Name	Type	Description
0–1	nRxOk	R/W	Number of packets received with CRC OK
2–3	nRxNok	R/W	Number of packets received with CRC error
4–5	nRxBufFull	R/W	Number of packets that have been received and discarded due to lack of buffer space
6	lastRssi	R	The RSSI of the last received packet
7			Reserved
8–11	timeStamp	R	Timestamp of the last received packet

Table 23-102. Test TX Command

Byte Index	Field Name	Type	Description
0–1	nTx	R/W	Number of packets transmitted

23.6.1.4 Other Structures and Bit Fields

Table 23-103. Receive Queue Entry Configuration Bit Field[1]

Bits	Bit Field Name	Description
0	bAutoFlushIgnored	If 1, automatically remove ignored packets from RX queue.
1	bAutoFlushCrcErr	If 1, automatically remove packets with CRC error from RX queue.
2	bAutoFlushEmpty	If 1, automatically remove empty packets from RX queue.
3	bIncludeLenByte	If 1, include the received length byte in the stored packet; otherwise discard it.
4	bIncludeCrc	If 1, include the received CRC field in the stored packet; otherwise discard it. This requires pktConf.bUseCrc to be 1.
5	bAppendRssi	If 1, append an RSSI byte to the packet in the RX queue.
6	bAppendStatus	If 1, append a status byte to the packet in the RX queue.
7	bAppendTimestamp	If 1, append a timestamp to the packet in the RX queue.

(1) This bit field is used for the rxConfig byte of the parameter structures.

Table 23-104. Sequence Number Status Bit Field

Bits	Bit Field Name	Description
0	lastRxSn	The SN bit of the header of the last packet received with CRC OK
1	lastTxSn	The SN bit of the header of the last transmitted packet
2	nextTxSn	The SN bit of the header of the next packet to transmit
3	bFirstPkt	For slave: 0 if a packet has been transmitted on the connection, 1 otherwise
4	bAutoEmpty	1 if the last transmitted packet was an auto-empty packet
5	bLlCtrlTx	1 if the last transmitted packet was an LL control packet (LLID = 11)
6	bLlCtrlAckRx	1 if the last received packet was the ACK of an LL control packet
7	bLlCtrlAckPending	1 if the last successfully received packet was an LL control packet that has not yet been ACKed

Table 23-105. White List Structure[1]

Byte Index	Field Name	Bits	Bit Field Name	Type	Description
0–7	entry[0]	0–7	Size	W	Number of white list entries
		8	bEnable	W	1 if the entry is in use, 0 if the entry is not in use
		9	addrType	W	The type address in the entry: public (0) or random (1)
		10	bWlIgn	R/W	1 if the entry is to be ignored by a scanner, 0 otherwise. Used to mask out entries that have already been scanned and reported.
		11–15			Reserved
		16–63	Address	W	The address contained in the entry
...					
8×n–8×n+7	entry[n]	0–7			Reserved
		8	bEnable	W	1 if the entry is in use, 0 if the entry is not in use
		9	addrType	W	The type address in the entry: public (0) or random (1)
		10	bWlIgn	R/W	1 if the entry is to be ignored by a scanner, 0 otherwise. Used to mask out entries that have already been scanned and reported.
		11–15			Reserved
		16–63	address	W	The address contained in the entry

(1) The white list structure has the form of an array. Each element consists of 8 bytes. The first byte of the first element tells the number of entries, and is reserved in the remaining entries. The second byte contains some configuration bits, and the remaining 6 bytes contain the address.

Table 23-106. Receive Status Byte Bit Field(1)

Bits	Bit Field Name	Description
0–5	channel	The channel on which the packet was received, provided channel is in the range 0–39; otherwise 0x3F
6	bIgnore	1 if the packet is marked as ignored, 0 otherwise
7	bCrcErr	1 if the packet was received with CRC error, 0 otherwise

(1) A byte of this bit field is appended to the received entries if configured.

The master and slave output structure field pktStatus follows the format described in Table 23-107. The bTimeStampValid bit is set to 0 by the radio CPU at the start of the operation, and to 1 if a timestamp is written to the output structure (this occurs for slave operation only). The bLastCrcErr bit is set according to the CRC result when a packet is fully received; if no packet is received, this bit remains unaffected. The remaining bits are set when a packet is received with CRC OK; if no packet is correctly received, these bits remain unaffected.

Table 23-107. Master and Slave Packet Status Byte

Bits	Bit Field Name	Description
0	bTimeStampValid	1 if a valid timestamp has been written to timeStamp; 0 otherwise
1	bLastCrcErr	1 if the last received packet had CRC error; 0 otherwise
2	bLastIgnored	1 if the last received packet with CRC OK was ignored; 0 otherwise
3	bLastEmpty	1 if the last received packet with CRC OK was empty; 0 otherwise
4	bLastCtrl	1 if the last received packet with CRC OK was empty; 0 otherwise
5	bLastMd	1 if the last received packet with CRC OK had MD = 1; 0 otherwise
6	bLastAck	1 if the last received packet with CRC OK was an ACK of a transmitted packet; 0 otherwise
7		Reserved

23.6.2 Interrupts

The radio CPU signals events back to the system CPU, using firmware-defined interrupts. Table 23-108 lists the interrupts used by the Bluetooth low energy commands. Each interrupt may be enabled individually in the system CPU. Section 23.6.4 gives the details about when the interrupts are generated.

Table 23-108. Interrupt Definitions Applicable to Bluetooth low energy

Interrupt Number	Interrupt Name	Description
0	COMMAND_DONE	A radio operation command has finished.
1	LAST_COMMAND_DONE	The last radio operation command in a chain of commands has finished.
4	TX_DONE	A packet has been transmitted.
5	TX_ACK	Acknowledgment received on a transmitted packet.
6	TX_CTRL	Transmitted LL control packet
7	TX_CTRL_ACK	Acknowledgment received on a transmitted LL control packet.
8	TX_CTRL_ACK_ACK	Acknowledgment received on a transmitted LL control packet, and acknowledgment transmitted for that packet.
9	TX_RETRANS	Packet has been retransmitted.
10	TX_ENTRY_DONE	TX queue data entry state changed to Finished.
11	TX_BUFFER_CHANGED	A buffer change is complete after CMD_BLE_ADV_PAYLOAD.
16	RX_OK	The packet is received with CRC OK, payload, and not to be ignored.
17	RX_NOK	The packet is received with CRC error.
18	RX_IGNORED	The packet is received with CRC OK, but to be ignored.

Table 23-108. Interrupt Definitions Applicable to Bluetooth low energy (continued)

Interrupt Number	Interrupt Name	Description
19	RX_EMPTY	The packet is received with CRC OK, not to be ignored, no payload.
20	RX_CTRL	LL control packet received with CRC OK, not to be ignored.
21	RX_CTRL_ACK	LL control packet received with CRC OK, not to be ignored, then acknowledgment sent.
22	RX_BUF_FULL	The packet is received that did not fit in the RX queue.
23	RX_ENTRY_DONE	RX queue data entry changing state to Finished.
29	MODULES_UNLOCKED	As part of the boot process, the Cortex-M0 has opened access to RF core modules and memories.
30	BOOT_DONE	The RF core CPU boot is finished.
31	INTERNAL_ERROR	The radio CPU has observed an unexpected error.

23.6.3 Data Handling

For all the Bluetooth low energy commands, data received over the air is stored in a receive queue. Data to be transmitted is fetched from a transmit queue for master and slave operation, while for the nonconnected operations, the data is fetched from a specific buffer, or created entirely by the radio CPU based on other available information.

23.6.3.1 Receive Buffers

A packet being received is stored in a receive buffer. First, a length byte or word is stored, if configured in the RX entry, by config.lenSz. This word is calculated from the length received over the air and the configuration of appended information.

Following the optional length field, the received header and payload is stored as received over the air. If rxConfig.bIncludeLenByte is 1, the full 16-bit header, including the received length field, is stored, despite the length field being redundant information if a length byte or word is present. If rxConfig.bIncludeLenByte is 0, only the first byte of the header is stored, so that the second byte, which only contains the redundant length field and some RFU bits, is discarded.

If rxConfig.bIncludeCrc is 1, the received CRC value is stored in the RX buffer. If rxConfig.bAppendRssi is 1, a byte indicating the received RSSI value is appended. If rxConfig.bAppendStatus is 1, a status byte of the type RXStatus_t, as defined in Table 23-106, is appended. If rxConfig.bAppendTimeStamp is 1, a timestamp indicating the start of the packet is appended. This timestamp corresponds to the ratmr_t data type. Even though the timestamp is multibyte, no word-address alignment is made, so the timestamp must be written and read byte-wise.

Figure 23-8 shows the format of an entry element in the RX queue.

Figure 23-8. Receive Buffer Entry Element

Optional	Mandatory fields		Optional fields			
0–2 bytes	1–2 bytes	0–37 bytes	0 or 3 bytes	0 or 1 byte	0 or 1 byte	0 or x bytes
Element length	BLE header	BLE payload	Received CRC	RSSI	Status	Timestamp

23.6.3.2 Transmit Buffers

For master and slave operations, transmit buffers are set up in a buffer queue. The length of the packet is defined by the length field in the data entry. The first byte of the data entry gives the LLID that goes into the data channel packet header. The NESN, SN, and MD bits are inserted automatically by the radio CPU, the RFU bits are set to 0, and the length field is calculated from the length of the data entry.

For advertising channel packets, the radio CPU automatically generates the header and the address fields of the payload. The data that comes after the address fields for each message type is given by a pointer to a data buffer. The number of bytes in this buffer is given in a separate parameter. If no data bytes are to be transmitted, this can be indicated by setting the length to 0. In this case, the pointer is ignored, and may be set to NULL. For Bluetooth low energy compliance, the ADV_DIRECT_IND and SCAN_REQ messages have no payload, but for the possibility of overriding this, data buffers are still present. For CONNECT_REQ messages, the data are required to have length 22 for Bluetooth low energy compliance, but the implementation allows any length.

23.6.4 Radio Operation Command Descriptions

Before running any radio operation command described in this document, the radio must be set up in Bluetooth low energy mode using the command CMD_RADIO_SETUP. Otherwise, the operation ends with an error.

The operations start with a radio operation command from the system CPU. The actual start of the operation is set up by the radio CPU according to startTrigger and startTime in the command structure. At this time, the radio CPU starts configuring the transmitter or receiver, depending on the type of operation. The system CPU must consider the setup time of the transmitter or receiver when calculating the start time of the operation.

The radio CPU sets up the channel based on the channel parameter. If the channel is in the range from 0 to 39, it indicates a data channel index or advertising channel index. In this case, only the values 0 to 36 are allowed in master and slave commands, and only the values 37 to 39 are allowed in advertiser, scanner, and initiator commands. If the channel is in the range from 60 to 207, it indicates an RF with an offset of 2300 MHz. If the channel is 255, the radio CPU does not program any frequency word, but keeps the frequency already programmed with CMD_FS. If the channel is 255 and the frequency synthesizer is not running, the operation ends with an error.

The whitening parameter indicates the initialization of the 7-bit LFSR used for data whitening in Bluetooth low energy. If whitening.bOverride is 0 and the channel is in the range from 0 to 39, the LFSR initializes with (0x40 | channel). Otherwise, the LFSR initializes with whitening.init. If whitening.init is 0 in this case, no whitening is used.

All packets transmitted using Bluetooth low energy radio operation commands have a Bluetooth low-energy-compliant CRC appended. On all packets received using Bluetooth low-energy radio-operation commands, a Bluetooth low-energy-compliant CRC-check is performed. The initialization of the CRC register is defined for each command.

The radio CPU times transmissions immediately following receptions, to fulfill the requirements for T_IFS. For reception immediately following transmissions, the radio CPU times the start of RX and time-out so that it always receives a packet transmitted at a time within the limits set by the Bluetooth low energy standard, but without excessive margins, to avoid false syncing on advertising channels. For the first receive operation in a slave command, the radio CPU sets up a time-out as defined in pParams->timeoutTrigger and pParams->timeoutTime. The time of this trigger depends on the sleep-clock uncertainty, both in the slave and the peer master.

When the receiver is running, the message is received into an RX entry as described in Section 23.6.3.1 and Section 23.3.2.7. The radio CPU has flags bCrcErr and bIgnore, which are written to the corresponding fields of the status byte of the RX entry if present. If there is a CRC error on the received packet, the bCrcErr flag is set. If the CRC is OK, the bIgnore flag may be set based on principles defined for each role. This flag indicates that the system CPU may ignore the packet. After receiving a packet, the radio CPU raises an interrupt to the system CPU.

If a packet is received with a length that is too great, the reception is stopped and treated as if sync had not been obtained on the packet. By default, the maximum allowed payload length of advertising channel packets is 37, and the maximum allowed length of data channel packets is 31 (which can never be violated because the length field in this case is 5 bits). If either the bCrcErr or bIgnore flag is set or if the packet was empty (as defined under each operation), the packet may be removed from the RX entry before raising the interrupt, depending on the bAutoFlushIgnored, bAutoFlushCrc, and bAutoFlushEmpty bits of pParams->rxConfig.

The status field of the command issued is updated during the operation. When submitting the command, the system CPU writes this field with a state of IDLE (see Table 23-109). During the operation, the radio CPU updates the field to indicate the operation mode. When the operation is done, the radio CPU writes a status indicating that the operation is finished. Table 23-109 lists the status codes to be used by a Bluetooth low energy radio operation.

Table 23-109. Bluetooth low energy Radio Operation Status Codes

Number	Name	Description
Operation Not Finished		
0x0000	IDLE	Operation not started
0x0001	PENDING	Waiting for start trigger
0x0002	ACTIVE	Running operation
Operation Finished Normally		
0x1400	BLE_DONE_OK	Operation ended normally
0x1401	BLE_DONE_RXTIMEOUT	Time-out of first RX of slave operation or end of scan window
0x1402	BLE_DONE_NOSYNC	Time-out of subsequent RX
0x1403	BLE_DONE_RXERR	Operation ended because of receive error (CRC or other)
0x1404	BLE_DONE_CONNECT	CONNECT_REQ received or transmitted
0x1405	BLE_DONE_MAXNACK	Maximum number of retransmissions exceeded
0x1406	BLE_DONE_ENDED	Operation stopped after end trigger
0x1407	BLE_DONE_ABORT	Operation aborted by abort command
0x1408	BLE_DONE_STOPPED	Operation stopped after stop command
Operation Finished With Error		
0x1800	BLE_ERROR_PAR	Illegal parameter
0x1801	BLE_ERROR_RXBUF	No available RX buffer (advertiser, scanner, initiator)
0x1802	BLE_ERROR_NO_SETUP	Radio was not set up in Bluetooth low energy mode
0x1803	BLE_ERROR_NO_FS	Synthesizer was not programmed when running RX or TX
0x1804	BLE_ERROR_SYNTH_PROG	Synthesizer programming failed
0x1805	BLE_ERROR_RXOVF	RX overflow observed during operation
0x1806	BLE_ERROR_TXUNF	TX underflow observed during operation

The conditions for giving each status are listed for each operation. Some of the error causes listed in Table 23-109 are not repeated in these lists. In some cases, general error causes may occur. In all of these cases, the result of the operation is ABORT.

23.6.4.1 Link Layer Connection

At the start of a slave or master operation, the radio CPU waits for the start trigger, then programs the frequency based on the channel parameter of the command structure. The channel parameter is not allowed to be 37, 38, or 39, because these are advertising channels. The radio CPU sets up the access address defined in pParams->accessAddress, and uses the CRC initialization value defined in pParams->crcInit. The whitener is set up as defined in the whitening parameter. The radio CPU then configures the receiver or transmitter. The operation continues with reception and transmission, until it is ended by one of the end-of-command criteria.

When the demodulator obtains sync on a message, the message is received into the first available RX buffer that can fit the packet. The flags bCrcErr and bIgnore are set according to Table 23-110 depending on the CRC result, and whether the SN field of the header was the same as the SN field of the last successfully received packet. A received packet that has a payload length of 0 is viewed as an empty packet. This means that if pParams->rxConfig.bAutoflushEmpty is 1 and bCrcErr and bIgnore are both 0, the packet is removed from the RX buffer.

Table 23-110. Actions on Received Packets

CRC Result	SN Different than Previous	bCrcErr	bIgnore
OK	Yes	0	0
OK	No	0	1
NOK	X	1	0

If there is no available RX buffer with enough available space to hold the received packet, the received data are discarded. The packet is received, however, so that the CRC can be checked. When the packet has been received, the radio CPU sets the sequence bits so that a retransmission of the lost packet is requested (that is, NACK), unless the packet would have been discarded from the RX queue anyway due to the setting of pParams->rxConfig.

If two subsequent packets are received with CRC error, the command ends, as required by the Bluetooth low energy specification.

When a packet must be transmitted or retransmitted, it is read from the current data entry in the TX queue unless the TX queue is empty or an automatic empty packet must be retransmitted. The radio CPU creates the header as follows: the LLID bits are inserted from the first byte of the TX data entry. The SN and NESN bits are set to values according to the Bluetooth low energy protocol. The MD bit is calculated automatically. If the TX queue is empty, an empty packet (LLID = 0x1, Length = 0) is transmitted. This empty packet is referred to as an automatic empty packet.

Interrupts can be raised on different conditions. The pOutput structure contains counters corresponding to the interrupts. Table 23-111 lists the conditions for incrementing each counter or raising an interrupt. More than one condition may be fulfilled after a packet is transmitted or received. In the list of conditions, the term *acknowledgment* is used, which is defined as a successfully received packet with an NESN value in the header different from the SN value of the last transmitted packet.

Table 23-111. Conditions for Incrementing Counters and Raising Interrupts for Master and Slave Commands

Condition	Counter Incremented	Interrupt Generated
Packet transmitted	nTx	TX_DONE
Packet transmitted and acknowledgment received	nTxAck	TX_ACK
Packet with LLID = 11b transmitted	nTxCtrl	TX_CTRL
Packet with LLID = 11b transmitted and acknowledgment received	nTxCtrlAck	TX_CTRL_ACK
Packet with LLID = 11b transmitted, acknowledgment received, and acknowledgment sent	nTxCtrlAckAck	TX_CTRL_ACK_ACK
Packet transmitted with same SN as previous transmitted packet	nTxRetrans	TX_RETRANS
Packet with payload transmitted and acknowledgment received	nTxEntryDone	TX_ENTRY_DONE
Packet received with bCrcErr = 0, bIgnore = 0, and payload length > 0	nRxOk	RX_OK
Packet received with CRC error (bCrcErr = 1)	nRxNok	RX_NOK
Packet received with bCrcErr = 0 and bIgnore = 1	nRxIgnored	RX_IGNORED
Packet received with bCrcErr = 0, bIgnore = 0, and payload length = 0	nRxEmpty	RX_EMPTY
Packet received with LLID = 11b, bCrcErr = 0 and bIgnore = 0	nRxCtrl	RX_CTRL
Packet received with LLID = 11b, bCrcErr = 0 and bIgnore = 0, and acknowledgment sent	nRxCtrlAck	RX_CTRL_ACK
Packet received which did not fit in RX buffer and was not to be flushed	nRxBufFull	RX_BUF_FULL
The first RX data entry in the RX queue changed state to finished	—	RX_ENTRY_DONE

The radio CPU maintains two counters: one packet counter nPkt, and one NACK counter nNack. These two counters are both initialized to pParams->maxPkt and pParams->maxNack, respectively, at the start of the master or slave radio operation. The packet counter nPkt is decremented each time a packet is transmitted. The NACK counter nNack is decremented if a packet is received that does not contain an acknowledgment of the last transmitted packet, and is reset to pParams->maxNack if an acknowledgment is received. If either counter counts to 0, the operation ends. This occurs after a packet has been received for master and a packet has been transmitted for slave. Setting pParams->maxPkt or pParams->maxNack to 0 disables the corresponding counter functionality.

A trigger to end the operation is set up by pParams->endTrigger and pParams->endTime. If the trigger defined by this parameter occurs, the radio operation ends as soon as possible. Any packet transmitted after this has MD = 0, and the connection event ends after the next packet has been transmitted for a slave or received for a master. If the immediate command CMD_STOP is received by the radio CPU, it has the same meaning as the end trigger occurring, except that the status code after ending is CMD_DONE_STOPPED.

The register pParams->seqStat contains bits that are updated by the radio CPU during operation, and are used to get correct operation on SN and NESN and retransmissions. The rules for the radio CPU follow:

- Before the first operation on a connection, the bits in pParams->seqStat are set as follows by the system CPU:
  - lastRXSn = 1
  - lastTXSn = 1
  - nextTXSn = 0
  - bFirstPkt = 1
  - bAutoEmpty = 0
  - bLlCtrlRX = 0
  - bLlCtrlAckRX = 0
  - bLlCtrlAckPending = 0
- When determining if the SN field of the header was the same as the SN field of the last successfully received packet, the received SN bit is compared to pParams->seqStat.lastRXSn.
- If a packet is received with correct CRC and the packet fits in an RX buffer, the received SN is stored in pParams->seqStat.lastRXSn. If the packet was an LL control packet (LLID = 11b) and the packet was not to be ignored, pParams->seqStat.bLlCtrlAckPending is set to 1 and an RX_CTRL interrupt is raised.
- If a packet is received with correct CRC and the received NESN bit is different from pParams->seqStat.lastTXSn, pParams->seqStat.nextTXSn is set to the value of the received NESN bit (regardless of whether the packet fits in an RX buffer).
- If pParams->seqStat.bFirstPkt = 0:
  - If pParams->seqStat.nextTXSn was updated and became different from pParams->seqStat.lastTXSn after reception of a packet, nNack is set to pParams->maxNack and a TX_ACK interrupt is raised.
  - Otherwise, nNack is decremented.
  - If pParams->seqStat.nextTXSn was updated and became different from pParams->seqStat.lastTXSn after reception of a packet, and pParams->seqStat.bAutoEmpty = 0, the current TX queue entry is finished and the next one is set as active, and a TX_ENTRY_DONE interrupt is raised. If pParams->seqStat.bLlCtrlTX = 1, an TX_CTRL_ACK interrupt is raised and pParams->seqStat.bLlCtrlAckRX is set to 1.
  - If pParams->seqStat.nextTXSn was updated and became different from pParams->seqStat.lastTXSn after reception of a packet, pParams->seqStat.bAutoEmpty is set to 0.
- If no buffer is available in the TX queue, or if pParams->seqStat.nextTXSn is equal to pParams->seqStat.lastTXSn and pParams->seqStat.bAutoEmpty = 1 when transmission of a packet is to occur, an automatically empty packet is transmitted. Nothing is read from the TX queue. Otherwise, the transmitted packet is read from the first entry of the TX queue.
- In the header of a transmitted packet, the SN bit is set to the value of pParams->seqStat.nextTXSn, and the NESN bit is set to the inverse of pParams->seqStat.lastRXSn.

- After a packet has been transmitted:
  - If pParams->seqStat.nextTXSn is equal to pParams->seqStat.lastTXSn, a TX_RETRANS interrupt is raised.
  - If pParams->seqStat.nextTXSn is different from pParams->seqStat.lastTXSn after a transmission and the transmitted packet had LLID = 11b, a TX_Ctrl interrupt is raised.
  - If pParams->seqStat.nextTXSn is different from pParams->seqStat.lastTXSn after a transmission and pParams->seqStat.bLlCtrlAckPending = 1, an RX_CTRL_ACK interrupt is raised.
  - If pParams->seqStat.nextTXSn is different from pParams->seqStat.lastTXSn after a transmission and pParams->seqStat.bLlCtrlAckRX = 1, a TX_CTRL_ACK_ACK interrupt is raised.
  - pParams->seqStat.lastTXSn is set to the value of pParams->seqStat.nextTXSn.
  - pParams->seqStat.bAutoEmpty is set to 1 if the packet was not read from the TX queue, otherwise to 0.
  - pParams->seqStat.bLlCtrlTX is set to 1 if the transmitted packet had LLID = 11, otherwise to 0.
  - pParams->seqStat.firstPkt, pParams ->seqStat.bLlCtrlAckPending, and pParams->seqStat.bLlCtrlAckRX is set to 0.
  - A TX_DONE interrupt is raised.
  - nPkt is decremented.

When an interrupt is raised as described previously, the corresponding counter given in Table 23-90 is incremented.

In the header of a transmitted packet, the MD bit is set according to the following rules:

- If the transmit queue is empty or the packet being transmitted is the last packet of the transmit queue, MD is 0.
- If the trigger described in pParams->endTrigger has occurred, MD is 0.
- If the counter nPkt is 1, MD is 0.
- Otherwise, MD is 1.

The pOutput structure contains counters that are updated by the radio CPU as explained previously and in Table 23-90. The radio CPU does not initialize the fields, so this must be done by the system CPU when a reset of the counters is desired. In addition to the counters, the radio CPU sets the following fields:

- If a packet is received, lastRssi is set to the RSSI of that packet.
- For slave commands, timeStamp is set to the timestamp of the start of the first received packet, if any packet is received. bValidTimeStamp is set to 0 at the beginning of the operation and to 1 if a packet is received so that timeStamp is written.

For correct operation, the value of pParams->seqStat is the same at the beginning of a command as at the end of the previous operation of the same connection. The TX queue must also be unmodified between commands operating on the same connection, except that packets may be appended to the queue.

23.6.4.2 Slave Command

A slave radio operation is started by a CMD_BLE_SLAVE command. In the command structure, it has a pParams parameter of the type defined in Table 23-90, and a pOutput parameter of the type defined in Table 23-97. The operation starts with reception. The parameters pParams->timeoutTrigger and pParams->timeoutTime define the time to end the operation if no sync is found by the demodulator. The startTrigger and pParams->timeoutTrigger together define the receive window for the slave.

The first received packet of a new LL connection on a slave is given special treatment, and is signaled by the system CPU by setting pParams->seqStat.bFirstPkt to 1 when starting the first slave operation of a new connection. When this flag is set, the received packet is not viewed as an ACK or NACK of a previous transmitted packet. When a packet has been transmitted, the radio CPU clears pParams->seqStat.bFirstPkt.

The radio CPU writes a timestamp of the first received packet of the radio operation into pOutput->timeStamp. The captured time can be used by the system CPU as an anchor point to calculate the start of future slave commands. This time is also defined as event 1, and may be used for timing subsequent chained operations. If no anchor point is found, event 1 is the time of the start of the slave operation.

If a packet is received with CRC error, the radio CPU ends the radio operation if the previous packet in the same radio operation was also received with CRC error (see Table 23-112). Otherwise, if a packet is received, the radio CPU starts the transmitter and transmits from the TX queue, or transmits an automatically empty packet if the TX queue is empty. The transmission may be a retransmission. Unless the operation ends due to the criteria listed in Table 23-112, the receiver starts after the transmission is done.

A slave operation ends due to one of the causes listed in Table 23-112. After the operation has ended, the status field of the command structure (2 status bytes listed in Table 23-8) indicates why the operation ended. In all cases, a COMMAND_DONE interrupt is raised. In each case, it is indicated if the result is TRUE, FALSE, or ABORT, which decides the next action.

Table 23-112. End of Slave Operation

Condition	Status Code	Result
Transmitted packet with MD=0 after having successfully received packet where MD bit of header is 0.	BLE_DONE_OK	TRUE
Transmitted packet with MD=0 after having received packet which did not fit in RX queue.	BLE_DONE_OK	TRUE
Finished transmitting packet and nPkt counted to 0.	BLE_DONE_OK	TRUE
Trigger indicated by pParams->timeoutTrigger occurred before demodulator sync is ever obtained after starting the command.	BLE_DONE_RXTIMEOUT	FALSE
No sync obtained on receive operation after transmit.	BLE_DONE_NOSYNC	TRUE
Two subsequent packets in the same operation were received with CRC error.	BLE_DONE_RXERR	TRUE
Finished transmitting packet after the internal counter nNack had counted down to 0.	BLE_DONE_MAXNACK	TRUE
Finished transmitting packet after having observed trigger indicated by pParams->endTrigger.	BLE_DONE_ENDED	FALSE
Finished transmitting packet after having observed CMD_STOP.	BLE_DONE_STOPPED	FALSE
Received CMD_ABORT.	BLE_DONE_ABORT	ABORT
Illegal value of channel	BLE_ERROR_PAR	ABORT
TX data entry length field has illegal value.	BLE_ERROR_PAR	ABORT

23.6.4.3 Master Command

A master radio operation is started by a CMD_BLE_MASTER command. In the command structure, it has a pParams parameter of the type defined in Table 23-91 and a pOutput parameter of the type defined in Table 23-97. The operation starts with transmission. After each transmission, the receiver is started.

A master operation ends due to one of the causes listed in Table 23-113. After the operation has ended, the status field of the command structure (2 status bytes listed in Table 23-8) indicates why the operation ended. In all cases, a COMMAND_DONE interrupt is raised. In each case, it is indicated if the result is TRUE, FALSE, or ABORT, which decides the next action.

Table 23-113. End of Master Operation

Condition	Status Code	Result
Successfully received packet with MD = 0 after having transmitted a packet with MD = 0.	BLE_DONE_OK	TRUE
Received packet which did not fit in RX queue after having transmitted a packet with MD = 0.	BLE_DONE_OK	TRUE
Received a packet after nPkt had counted to 0.	BLE_DONE_OK	TRUE
No sync obtained on receive operation after transmit.	BLE_DONE_NOSYNC	TRUE
Two subsequent packets in the same operation were received with CRC error.	BLE_DONE_RXERR	TRUE
The internal counter nNack counted down to 0 after a packet was received.	BLE_DONE_MAXNACK	TRUE
Received a packet after having observed trigger indicated by pParams >endTrigger.	BLE_DONE_ENDED	FALSE
Received a packet after having observed CMD_STOP.	BLE_DONE_STOPPED	FALSE
Received CMD_ABORT.	BLE_DONE_ABORT	ABORT
Illegal value of channel	BLE_ERROR_PAR	ABORT
TX data entry length field has illegal value.	BLE_ERROR_PAR	ABORT

23.6.4.4 Advertiser

At the start of any advertiser operation, the radio CPU waits for the start trigger, then programs the frequency based on the channel parameter of the command structure. The channel parameter is not allowed to be in the range from 0 to 36, as these are data channels. The radio CPU sets up the advertising channel access address and uses the CRC initialization value 0x55 5555. The whitener is set up as defined in the whitening parameter. The radio CPU then configures the transmitter. Except for an advertiser that is not connectable, the operation goes on with reception after transmission, and if a SCAN_REQ is received, another transmission of a SCAN_RSP may occur.

In Bluetooth low energy mode, advertising is usually done over all three advertising channels. To set this up, three command structures can be chained using the pNextOp parameter. Typically, the parameter and output structures can be the same for all channels.

The first packet transmitted is always an ADV*_IND packet. This packet consists of a header, an advertiser address, and advertising data, except for the ADV_DIRECT_IND packet used in directed advertising. The radio CPU constructs these packets as follows (the ADV_DIRECT_IND packet is described in Section 23.6.4.4.2). In the header, the PDU Type bits are as shown in Table 23-114. The TXAdd bit is as shown in pParams->advConfig.deviceAddrType. The length is calculated from the size of the advertising data, meaning that it is pParams->advLen + 6. The RXAdd bit is not used and is 0, along with the RFU bits. The payload starts with the 6-byte device address, which is read from pParams->pDeviceAddress. The rest of the payload is read from the pParams->pAdvData buffer (if pParams->advLen is nonzero).

Table 23-114. PDU Types for Different Advertiser Commands

Command	Type of Advertising Packet	Value of PDU Type Bits in Header
CMD_BLE_ADV	ADV_IND	0000b
CMD_BLE_ADV_DIR	ADV_DIRECT_IND	0001b
CMD_BLE_ADV_NC	ADV_NONCONN_IND	0010b
CMD_BLE_ADV_SCAN	ADV_SCAN_IND	0110b

Except when an advertiser is not connectable, the receiver starts after the ADV*_IND packet has been transmitted. Depending on the type of advertiser operation, the receiver listens for a SCAN_REQ or a CONNECT_REQ message. If the demodulator obtains sync, the header is checked once it is received, and if it is not a SCAN_REQ or CONNECT_REQ message, the demodulator is stopped immediately.

A SCAN_REQ or CONNECT_REQ message is received into the RX queue given by pParams->pRxQ, as described in Section 23.6.3.1. The bCrcErr and bIgnore bits are set according to the CRC result and the received message. For connectable undirected or scannable advertising, the AdvA field in the message, along with the TXAdd bit of the received header, is compared to the pParams->pDeviceAddress array and the pParams->advConfig.deviceAddrType bit, respectively, to see if the message was addressed to this advertiser. Then, depending on the advertising filter policy given by pParams->advConfig.advFilterPolicy, the received ScanA or InitA field, along with the RXAdd bit of the received header, is checked against the white list as described in Section 23.6.4.9, except for a directed advertiser, where the received header is compared against the peer address as described in Section 23.6.4.4.2. Depending on this comparison, the actions taken are as given in Table 23-115, where the definition of each action, including the value used on the bCrcErr and bIgnore bits, is given in Table 23-116. If pParams->advConfig.bStrictLenFilter is 1, only length fields that are compliant with the Bluetooth low energy specification are considered valid. For a SCAN_REQ, that means a length field of 12, and for a CONNECT_REQ it means a length field of 34. If pParams->advConfig.bStrictLenFilter is 0, all received packets with a length field less than or equal to the maximum length of an advertiser packet (37, but can be overridden) are considered valid. If the length is not valid, the receiver is stopped.

Table 23-115. Actions to Take Based on Received Packets for Advertisers(1)

PDU Type	CRC Result	Advertiser Type	Valid Length	AdvA Matches Own Address	Filter Policy	ScanA or InitA Present in White List	Action Number
SCAN_REQ	OK	C, S	Yes	No	X	X	1
SCAN_REQ	OK	C, S	Yes	Yes	1 or 3	No	1
SCAN_REQ	OK	C, S	Yes	Yes	1 or 3	Yes	2
SCAN_REQ	OK	C, S	Yes	Yes	0 or 2	X	2
SCAN_REQ	NOK	C, S	Yes	X	X	X	3
SCAN_REQ	X	C, S	No	X	X	X	5
SCAN_REQ	X	D	X	X	X	X	5
CONNECT_REQ	OK	C, D	Yes	No	X	X	1
CONNECT_REQ	OK	C, D	Yes	Yes	2 or 3	No	1
CONNECT_REQ	OK	C, D	Yes	Yes	2 or 3	Yes	4
CONNECT_REQ	OK	C, D	Yes	Yes	0 or 1	X	4
CONNECT_REQ	NOK	C, D	Yes	X	X	X	3
CONNECT_REQ	X	C, D	No	X	X	X	5
CONNECT_REQ	X	S	X	X	X	X	5
Other	X	X	X	N/A	X	N/A	5
No packet received	N/A	X	N/A	N/A	X	N/A	5

(1) C – connectable undirected; D – connectable directed; S – scannable; X – don't care; N/A – not applicable

Table 23-116. Descriptions of the Actions to Take on Received Packets

Action Number	bCrcErr	bIgnore	Description
1	0	1	End operation with BLE_DONE_OK status.
2	0	0	Transmit SCAN_RSP message.
3	1	0	End operation with BLE_DONE_RXERR status.
4	0	0	End operation with BLE_DONE_CONNECT status.
5	—	—	Stop receiver immediately and end operation with BLE_DONE_NOSYNC status.

If a SCAN_REQ packet is received with a length of 12 (or less), it is viewed as an empty packet. This means that if pParams->rxConfig.bAutoflushEmpty is 1 and the bCrcErr and bIgnore bits are both 0, the packet is removed from the RX buffer. If a packet is flagged by bIgnore or bCrcErr, it may also be removed, based on the bits in pParams->rxConfig.

If the packet received did not fit in the RX queue, the packet is received to the end, but the received bytes are not stored. If the packet would normally not have been discarded from the RX queue based on the bits in pParams->rxConfig, the command ends.

If, according to Table 23-115 and Table 23-116, the next action is to transmit a SCAN_RSP, the radio CPU starts the transmitter to transmit this packet. It consists of a header, an advertiser address, and advertising data. The radio CPU constructs these packets as follows. In the header, the PDU Type bits are 0100b. The TXAdd bit is as shown in pParams->advConfig.devicAddrType. The length is calculated from the size of the scan response data, pParams->scanRspLen + 6. The RXAdd bit is not used and is 0, along with the RFU bits. The payload starts with the 6-byte device address, which is read from pParams->pDeviceAddress. The rest of the payload is read from the pParams->pScanRspData buffer. After the SCAN_RSP has been transmitted, the command ends.

A trigger to end the operation is set up by pParams->endTrigger. If the trigger defined by this parameter occurs, the radio operation continues to completion, but the status code after ending is BLE_DONE_ENDED and the result is FALSE. This can, for instance, be used to stop execution instead of proceeding with the next chained operation by use of the condition in the command structure. If the immediate command CMD_STOP is received by the radio CPU, CMD_STOP has the same meaning as the end trigger occurring, except that the status code after ending is CMD_DONE_STOPPED.

The output structure pOutput contains fields which give information on the command being run. The radio CPU does not initialize the fields, so this must be done by the system CPU when a reset of the counters is desired. The fields are updated by the radio CPU as described in the following list. The list also indicates when interrupts are raised in the system CPU.

- When the ADV*_IND packet has been transmitted, nTXAdvInd is incremented and a TX_DONE interrupt is raised.
- If a SCAN_RSP packet has been transmitted, nTxScanRsp is incremented afterward, and a TX_DONE interrupt is raised.
- If a SCAN_REQ is received with CRC OK and the bIgnore is flag cleared, nRxScanReq is incremented. If the payload length is 12 or less, an RX_EMPTY interrupt is raised. If the payload length is greater than 12, an RX_OK interrupt is raised.
- If a CONNECT_REQ is received with CRC OK and the bIgnore flag is cleared, nRxConnectReq is incremented and an RX_OK interrupt is raised.
- If a packet is received with CRC error, nRxNok is incremented and an RX_NOK interrupt is raised.
- If a packet is received and the bIgnore flag is set, nRxIgnored is incremented and an RX_IGNORED interrupt is raised.
- If a packet is received that did not fit in the RX queue, nRxBufFull is incremented and an RX_BUF_FULL interrupt is raised.
- If a packet is received, lastRssi is set to the RSSI of that packet.
- If a packet is received, timeStamp is set to a timestamp of the start of that packet. For a CONNECT_REQ, this can be used to calculate the anchor point of the first packet.
- If the first RX data entry in the RX queue changed state to Finished after a packet was received, an RX_ENTRY_DONE interrupt is raised.

23.6.4.4.1 Connectable Undirected-Advertiser Command

A connectable undirected-advertiser operation is started by a CMD_BLE_ADV command. In the command structure, it has a pParams parameter of the type defined in Table 23-92, and a pOutput parameter of the type defined in Table 23-98. The operation starts with transmission and operates as described in Section 23.6.4.4.

A connectable undirected-advertiser operation ends with one of the statuses listed in Table 23-117. After the operation has ended, the status field of the command structure (2 status bytes listed in Table 23-8) indicates why the operation ended. In all cases, a COMMAND_DONE interrupt is raised. In each case, it is indicated if the result is TRUE, FALSE, or ABORT, which decides the next action.

Table 23-117. End of Connectable Undirected-Advertiser Operation

Condition	Status Code	Result
Performed Action Number 1 after running receiver.	BLE_DONE_OK	TRUE
Performed Action Number 2 and transmitted SCAN_RSP.	BLE_DONE_OK	TRUE
Performed Action Number 3 after running receiver.	BLE_DONE_RXERR	TRUE
Performed Action Number 4 after running receiver.	BLE_DONE_CONNECT	FALSE
Performed Action Number 5 after running receiver.	BLE_DONE_NOSYNC	TRUE
Observed trigger indicated by pParams->endTrigger, then performed Action Number 1, 2, 3, or 5.	BLE_DONE_ENDED	FALSE
Observed CMD_STOP, then performed Action Number 1, 2, 3, or 5.	BLE_DONE_STOPPED	FALSE
Received CMD_ABORT.	BLE_DONE_ABORT	ABORT
No space in RX buffer to store received packet.	BLE_ERROR_RXBUF	FALSE
Illegal value of channel	BLE_ERROR_PAR	ABORT
Advertising data or scan response data length field has illegal value.	BLE_ERROR_PAR	ABORT

23.6.4.4.2 Connectable Directed-Advertiser Command

A connectable directed-advertiser operation is started by a CMD_BLE_ADV_DIR command. In the command structure, it has a pParams parameter of the type defined in Table 23-92, and a pOutput parameter of the type defined in Table 23-98. The operation starts with transmission and operates as described in Section 23.6.4.4, with some modifications as described in the following paragraphs.

For the directed advertiser, pParams->pWhiteList points to a buffer containing only the device address of the device to which to connect. The address type of the peer is given in pParams->advConfig.peerAddrType. The first transmit operation sends an ADV_DIRECT_IND packet. The radio CPU constructs this packet as follows. In the header, the PDU Type bits are 0001b as shown in Table 23-114. The TXAdd bit is as shown in pParams->advConfig.deviceAddrType. The RXAdd bit is as shown in pParams->advConfig.peerAddrType.

The length is calculated from the size of the advertising data, pParams->advLen + 12. The RFU bits are 0. The payload starts with the 6-byte device address, which are read from pParams->pDeviceAddress, followed by the 6-byte peer address read from pParams->pWhiteList. By the Bluetooth low energy specification, there is no more payload, but a noncompliant message may be constructed by setting pParams->advLen to a nonzero value. If so, the rest of the payload is read from the pParams->pAdvData buffer.

The receiver is started after the ADV_DIRECT_IND packet has been transmitted as described in Section 23.6.4.4, and received packets are processed as described there. When checking the address against the white list, check the received RXAdd bit against pParams->advConfig.peerAddrType, and check the received InitA field against pParams->pWhiteList.

A directed-advertiser operation ends with one of the statuses listed in Table 23-118. After the operation has ended, the status field of the command structure (2 status bytes listed in Table 23-8) indicates why the operation ended. In all cases, a COMMAND_DONE interrupt is raised. In each case, it is indicated if the result is TRUE, FALSE, or ABORT, which decides the next action.

Table 23-118. End of Directed-Advertiser Operation

Condition	Status Code	Result
Performed Action Number 1 after running receiver.	BLE_DONE_OK	TRUE
Performed Action Number 3 after running receiver.	BLE_DONE_RXERR	TRUE
Performed Action Number 4 after running receiver.	BLE_DONE_CONNECT	FALSE
Performed Action Number 5 after running receiver.	BLE_DONE_NOSYNC	TRUE
Observed trigger indicated by pParams->endTrigger, then performed Action Number 1, 3, or 5.	BLE_DONE_ENDED	FALSE
Observed CMD_STOP, then performed Action Number 1, 3, or 5.	BLE_DONE_STOPPED	FALSE
Received CMD_ABORT.	BLE_DONE_ABORT	ABORT
No space in RX buffer to store received packet.	BLE_ERROR_RXBUF	FALSE
Illegal value of channel	BLE_ERROR_PAR	ABORT
Advertising data length field has illegal value.	BLE_ERROR_PAR	ABORT

23.6.4.4.3 Nonconnectable Advertiser Command

An advertiser operation that is not connectable is started by a CMD_BLE_ADV_NC command. In the command structure, it has a pParams parameter of the type defined in Table 23-92, and a pOutput parameter of the type defined in Table 23-98. The operation starts with transmission and operates as described in Section 23.6.4.4. After transmission of an ADV_NONCONN_IND, the operation ends without any receive operation.

An advertiser operation that is not connectable ends with one of the statuses listed in Table 23-119. After the operation has ended, the status field of the command structure (2 status bytes listed in Table 23-8) indicates why the operation ended. In all cases, a COMMAND_DONE interrupt is raised. In each case, it is indicated if the result is TRUE, FALSE, or ABORT, which decides the next action.

Table 23-119. End of Nonconnectable Advertiser Operation

Condition	Status Code	Result
Transmitted ADV_NONCONN_IND.	BLE_DONE_OK	TRUE
Observed trigger indicated by pParams->endTrigger, then finished transmitting ADV_NONCONN_IND.	BLE_DONE_ENDED	FALSE
Observed CMD_STOP, then finished transmitting ADV_NONCONN_IND.	BLE_DONE_STOPPED	FALSE
Received CMD_ABORT.	BLE_DONE_ABORT	ABORT
Illegal value of channel	BLE_ERROR_PAR	ABORT
Advertising data length field has illegal value.	BLE_ERROR_PAR	ABORT

23.6.4.4.4 Scannable Undirected-Advertiser Command

A scannable undirected-advertiser operation is started by a CMD_BLE_ADV_SCAN command. In the command structure, it has a pParams parameter of the type defined in Table 23-92, and a pOutput parameter of the type defined in Table 23-98. The operation starts with transmission and operates as described in Section 23.6.4.4.

A scannable undirected-advertiser operation ends with one of the statuses listed in Table 23-120. After the operation has ended, the status field of the command structure (2 status bytes listed in Table 23-8) indicates why the operation ended. In all cases, a COMMAND_DONE interrupt is raised. In each case, it is indicated if the result is TRUE, FALSE, or ABORT, which decides the next action.

Table 23-120. End of Scannable Undirected-Advertiser Operation

Condition	Status Code	Result
Performed Action Number 1 after running receiver.	BLE_DONE_OK	TRUE
Performed Action Number 2 and transmitted SCAN_RSP.	BLE_DONE_OK	TRUE
Performed Action Number 3 after running receiver.	BLE_DONE_RXERR	TRUE
Performed Action Number 5 after running receiver.	BLE_DONE_NOSYNC	TRUE
Observed trigger indicated by pParams->endTrigger, then performed Action Number 1, 2, 3, or 5.	BLE_DONE_ENDED	FALSE
Observed CMD_STOP, then performed Action Number 1, 2, 3, or 5.	BLE_DONE_STOPPED	FALSE
Received CMD_ABORT.	BLE_DONE_ABORT	ABORT
No space in RX buffer to store received packet.	BLE_ERROR_RXBUF	FALSE
Illegal value of channel	BLE_ERROR_PAR	ABORT
Advertising data or scan response data length field has illegal value.	BLE_ERROR_PAR	ABORT

23.6.4.5 Scanner Command

A scanner operation is started by a CMD_BLE_SCANNER command. In the command structure, it has a pParams parameter of the type defined in Table 23-93, and a pOutput parameter of the type defined in Table 23-99. At the start of a scanner operation, the radio CPU waits for the start trigger, then programs the frequency based on the channel parameter of the command structure. The channel parameter is not allowed to be in the range from 0 to 36, because these are data channels. The radio CPU sets up the advertising channel access address and uses the CRC initialization value 0x55 5555. The whitener is set up as defined in the whitening parameter. The radio CPU then configures the receiver.

Tuned to the correct channel, the radio CPU starts listening for an advertising-channel packet. If sync is obtained on the demodulator, the message is received into the RX queue. The header is checked, and if it is not an advertising packet, reception stops and sync search restarted. The bCrcErr and bIgnore bits are set according to the CRC result and the received message. Depending on the scanning filter policy, given by pParams->scanConfig.scanFilterPolicy, the received AdvA field in the message, along with the TXAdd bit of the received header is checked against white list as described in Section 23.6.4.9. For ADV_DIRECT_IND messages, the received InitA field and RXAdd bit are checked against pParams->deviceAddr and pParams->scanConfig.deviceAddrType, respectively. Depending on this, and whether the scan is active or passive as signaled in pParams->scanConfig.bActiveScan, the actions taken are as shown in Table 23-121, where the definition of each action, including the value used on bCrcErr and bIgnore, is given in Table 23-122. If pParams->scanConfig.bStrictLenFilter is 1, only length fields compliant with the Bluetooth low energy specification are considered valid. For an ADV_DIRECT_IND, valid means a length field of 12, and for other ADV*_IND messages valid means a length field in the range from 6 to 37. If pParams->advConfig.bStrictLenFilter is 0, all received packets with a length field less than or equal to the maximum length of an advertiser packet (37, but can be overridden) are considered valid. If the length is not valid, the receiver stops.

Table 23-121. Actions on Received Packets by Scanner(1)

PDU Type	CRC Result	Filter Policy	AdvA to be Ignored	AdvA Present in White List	InitA Match	Active Scan	Action Number
ADV_IND	OK	1	No	No	N/A	X	1
ADV_IND	OK	1	No	Yes	N/A	No	2
ADV_IND	OK	1	No	Yes	N/A	Yes	3
ADV_IND	OK	0	No	X	N/A	No	2
ADV_IND	OK	0	No	X	N/A	Yes	3
ADV_IND	OK	X	Yes	X	N/A	X	1
ADV_IND	NOK	X	X	X	N/A	X	4
ADV_SCAN_IND	OK	1	No	No	N/A	X	1
ADV_SCAN_IND	OK	1	No	Yes	N/A	No	2
ADV_SCAN_IND	OK	1	No	Yes	N/A	Yes	3
ADV_SCAN_IND	OK	0	No	X	N/A	No	2
ADV_SCAN_IND	OK	0	No	X	N/A	Yes	3
ADV_SCAN_IND	OK	X	Yes	X	N/A	X	1
ADV_SCAN_IND	NOK	X	X	X	N/A	X	4
ADV_NONCONN_IND	OK	1	No	No	N/A	X	1
ADV_NONCONN_IND	OK	1	No	Yes	N/A	X	2
ADV_NONCONN_IND	OK	0	No	X	N/A	X	2
ADV_NONCONN_IND	OK	X	Yes	X	N/A	X	1
ADV_NONCONN_IND	NOK	X	X	X	N/A	X	4
ADV_DIRECT_IND	OK	1	No	No	X	X	1
ADV_DIRECT_IND	OK	1	No	Yes	No	X	1
ADV_DIRECT_IND	OK	1	No	Yes	Yes	X	2
ADV_DIRECT_IND	OK	0	No	X	No	X	1
ADV_DIRECT_IND	OK	0	No	X	Yes	X	2
ADV_DIRECT_IND	OK	X	Yes	X	X	X	1
ADV_DIRECT_IND	NOK	X	X	X	X	X	4
ADV*_IND with invalid length	X	X	X	X	X	X	5
Other	X	X	N/A	N/A	N/A	X	5

(1) X = don't care.

Table 23-122. Descriptions of the Actions to Take on Packets Received by Scanner

Action Number	bCrcErr	bIgnore	Description
1	0	1	Continue scanning.
2	0	0	Continue scanning or end operation with BLE_DONE_OK status.
3	0	0	Perform backoff procedure and send SCAN_REQ and receive SCAN_RSP if applicable. Then continue scanning or end operation.
4	1	0	Continue scanning.
5	—	—	Stop receiving packet, then continue scanning.

If the packet being received did not fit in the RX queue, the packet is received to the end, but the received bytes are not stored. If the packet would normally not have been discarded from the RX buffer, the operation ends.

If the action from the received packet is number 3, a SCAN_REQ is transmitted if allowed after a backoff procedure. This procedure starts with decrementing pParams->backoffCount. If this variable is 0 after the decrement, a SCAN_REQ is transmitted. If not, the operation ends. If the action from the received packet is number 2 or number 3, the next action may be to continue scanning or end the operation. This is configured with pParams->scanConfig.bEndOnRpt; if 1, the operation ends, otherwise scanning continues.

When transmitting a SCAN_REQ message, the radio CPU constructs this packet. In the header, the PDU Type bits are 0011b. The TXAdd bit is as shown in pParams->scanConfig.deviceAddrType. The RXAdd bit is as shown in the TXAdd field of the header of the received ADV_IND or ADV_SCAN_IND message. The length is calculated from the size of the scan request data, pParams->scanReqLen + 12. The RFU bits are 0. The payload starts with the 6-byte device address, which is read from pParams->pDeviceAddress, followed by the 6-byte peer address read from the AdvA field of the received message. By the Bluetooth low energy specification, there is no more payload, but a noncompliant message may be constructed by setting pParams->scanReqLen to a nonzero value. If so, the rest of the payload is read from the
pParams->pScanData buffer.

After a SCAN_REQ message is transmitted, the radio CPU configures the receiver and looks for a SCAN_RSP message from the advertiser to which the SCAN_REQ was sent. If sync is obtained on the demodulator, the header is checked when it is received, and if it is not a SCAN_RSP message, the demodulator is stopped immediately. If the header is a SCAN_RSP message, then it is received into the RX queue. Depending on the received SCAN_RSP, the values of bCrcErr and bIgnore are as given in Table 23-123. If pParams->scanConfig.bStrictLenFilter is 1, only length fields that are compliant with the Bluetooth low energy specification are considered valid. For a SCAN_RSP, valid means a length field in the range from 6 to 37. If pParams->scanConfig.bStrictLenFilter is 0, all received packets with a length field less than or equal to the maximum length of an advertiser packet (37, but can be overridden) are considered valid. If the length is not valid, the receiver is stopped.

Table 23-123. Actions on Packets Received by Scanner After Transmission of SCAN_REQ(1)

PDU Type	CRC Result	AdvA Same as in Request	bCrcErr	bIgnore	SCAN_RSP Result
SCAN_RSP	OK	No	0	1	Failure
SCAN_RSP	OK	Yes	0	0	Success
SCAN_RSP	NOK	X	1	0	Failure
SCAN_RSP with invalid length	X	X	–	–	Failure
Other	X	N/A	–	–	Failure
No packet received	N/A	N/A	–	–	Failure

(1) X = don't care.

After receiving or trying to receive a SCAN_RSP message, the backoff parameters are updated by the radio CPU. The update depends on the result as given in the SCAN_RSP Result column of Table 23-123 and the old values of the backoff parameters. The backoff parameters given in pParams->backoffPar are updated as shown in Table 23-124. After this update, the radio CPU sets pParams->backoffCount to a pseudo-random number between 1 and $2^{pParams->backoffPar.logUpperLimit}$.

Table 23-124. Update of Backoff Parameters(1)

SCAN_RSP Result	Old pParams->backoffPar		New pParams->backoffPar		
	bLastSucceeded	bLastFailed	bLastSucceeded	bLastFailed	logUpperLimit
Failure	X	0	0	1	logUpperLimit
Failure	0	1	0	0	min(logUpperLimit+1, 8)
Success	0	X	1	0	logUpperLimit
Success	1	0	0	0	max(logUpperLimit-1, 0)

(1) X = don't care.

If pParams->scanConfig.scanFilterPolicy and pParams->scanConfig.bAutoWlIgnore are both 1, the radio CPU automatically sets the bWlIgn bit of the white-list entry corresponding to the address from which an ADV*_IND message was received. This setting is done either after Action Number2 is performed, or after Action Number3 is performed and a SCAN_RSP is received with the result Success. This prevents reporting multiple advertising messages from the same device, and scanning the same device repeatedly.

The pseudo-random algorithm is based on a maximum-length 16-bit linear-feedback shift register (LFSR). The seed is as provided in pParams->randomState. When the operation ends, the radio CPU writes the current state back to this field. If pParams->randomState is 0, the radio CPU self-seeds by initializing the LFSR to the 16 LSBs of the RAT. This is done only when the LFSR is first needed (that is, after receiving an ADV*_IND), so there is some randomness to this value. If the 16 LSBs of the RAT are all 0, another fixed value is substituted.

When the device enters the scanning state, the system CPU must initialize as follows:

- pParams->backoffCount to 1
- pParams->backoffPar.logUpperLimit to 0
- pParams->backoffPar.bLastSucceeded to 0
- pParams->backoffPar.bLastFailed to 0
- pParams->randomState to a true-random value (or a pseudo-random number based on a true-random seed)

When starting new scanner operations while remaining in the scanning state, the system CPU must keep pParams->randomState, pParams->backoffCount, and pParams->backoffPar at the values they had at the end of the last scanner operation.

Two triggers to end the operation are set up by pParams->endTrigger/pParams->endTime and pParams->timeoutTrigger/pParams->timeoutTime, respectively. If either of these triggers occurs, the radio operation ends as soon as possible. If these triggers occur while waiting for sync on an ADV*_IND packet, the operation ends immediately. If they occur at another time, the operation continues until the scan would otherwise be resumed, and then ends. If the immediate command CMD_STOP is received by the radio CPU, it has the same meaning as the end trigger occurring, except that the status code after ending is CMD_DONE_STOPPED. The differences between the two triggers are the status and result at the end of the operation. Typically, timeoutTrigger can be used at the end of a scan window, while endTrigger can be used when scanning is to end entirely.

The output structure pOutput contains fields that give information on the command being run. The radio CPU does not initialize the fields, so this must be done by the system CPU when resetting the counters is desired. The fields are updated by the radio CPU as described in the following list. The list also indicates when interrupts are raised in the system CPU.

- If a SCAN_REQ packet has been transmitted, nTXScanReq is incremented and a TX_DONE interrupt is raised.
- If a SCAN_REQ is not transmitted due to the backoff procedure, nBackedOffScanReq is incremented.
- If an ADV*_IND packet is received with CRC OK and the bIgnore flag cleared, nRxAdvOk is incremented, an RX_OK interrupt is raised, and timeStamp is set to a timestamp of the start of the packet.
- If an ADV*_IND packet is received with CRC OK and the bIgnore flag set, nRxAdvIgnored is incremented and an RX_IGNORED interrupt is raised.
- If an ADV*_IND packet is received with CRC error, nRxAdvNok is incremented and an RX_NOK interrupt is raised.
- If an ADV*_IND packet is received and did not fit in the RX queue, nRxAdvBufFull is incremented and an RX_BUF_FULL interrupt is raised.
- If a SCAN_RSP packet is received with CRC OK and the bIgnore flag cleared, nRxScanRspOk is incremented and an RX_OK interrupt is raised.
- If a SCAN_RSP packet is received with CRC OK and the bIgnore flag set, nRxScanRspIgnored is incremented and an RX_IGNORED interrupt is raised.
- If a SCAN_RSP packet is received with CRC error, nRxScanRspNok is incremented and an RX_NOK interrupt is raised.
- If a SCAN_RSP packet is received and did not fit in the RX queue, nRxScanRspBufFull is incremented and an RX_BUF_FULL interrupt is raised.
- If a packet is received, lastRssi is set to the RSSI of that packet.
- If the first RX data entry in the RX queue changed state to Finished after a packet was received, an RX_ENTRY_DONE interrupt is raised.

A scanner operation ends with one of the statuses listed in Table 23-125. After the operation has ended, the status field of the command structure (2 status bytes listed in Table 23-8) indicates why the operation ended. In all cases, a COMMAND_DONE interrupt is raised. In each case, it is indicated if the result is TRUE, FALSE, or ABORT, which decides the next action.

Table 23-125. End of Scanner Operation

Condition	Status Code	Result
Performed Action Number2 with pParams->scanConfig.bEndOnRpt = 1.	BLE_DONE_OK	TRUE
Performed Action Number3 with pParams->scanConfig.bEndOnRpt = 1 and did not send SCAN_REQ due to backoff.	BLE_DONE_OK	TRUE
Performed Action Number3 with pParams->scanConfig.bEndOnRpt = 1, sent SCAN_REQ and received SCAN_RSP with bCrcErr = 0 and bIgnore = 0.	BLE_DONE_OK	TRUE
Performed Action Number3 with pParams->scanConfig.bEndOnRpt = 1, sent SCAN_REQ and received SCAN_RSP with bCrcErr = 1 or bIgnore = 1.	BLE_DONE_RXERR	TRUE
Performed Action Number3 with pParams->scanConfig.bEndOnRpt = 1, sent SCAN_REQ, but did not get sync or found wrong packet type or invalid length.	BLE_DONE_NOSYNC	TRUE
Observed trigger indicated by pParams->timeoutTrigger while waiting for sync on ADV*_IND.	BLE_DONE_RXTIMEOUT	TRUE
Observed trigger indicated by pParams->timeoutTrigger, then performed Action Number1, 2, 3, 4, or 5.	BLE_DONE_RXTIMEOUT	TRUE
Observed trigger indicated by pParams->endTrigger while waiting for sync on ADV*_IND.	BLE_DONE_ENDED	FALSE
Observed trigger indicated by pParams->endTrigger, then performed Action Number1, 2, 3, 4, or 5.	BLE_DONE_ENDED	FALSE
Observed CMD_STOP while waiting for sync on ADV*_IND.	BLE_DONE_STOPPED	FALSE
Observed CMD_STOP, then performed Action Number1, 2, 3, 4, or 5.	BLE_DONE_STOPPED	FALSE
Received CMD_ABORT.	BLE_DONE_ABORT	ABORT

Table 23-125. End of Scanner Operation (continued)

Condition	Status Code	Result
No space in RX buffer to store received packet	BLE_ERROR_RXBUF	FALSE
Illegal value of channel	BLE_ERROR_PAR	ABORT
Scan request data length field has illegal value.	BLE_ERROR_PAR	ABORT

23.6.4.6 Initiator Command

An initiator operation is started by a CMD_BLE_INITIATOR command. In the command structure, it has a pParams parameter of the type defined in Table 23-93 and a pOutput parameter of the type defined in Table 23-99. At the start of an initiator operation, the radio CPU waits for the start trigger, then programs the frequency based on the channel parameter of the command structure. The channel parameter is not allowed to be in the range from 0 to 36, because these are data channels. The radio CPU sets up the advertising channel access address and uses the CRC initialization value 0x55 5555. The whitener is set up as defined in the whitening parameter. The radio CPU then configures the receiver.

After tuning to the correct channel, the radio CPU starts listening for an advertising channel packet. If sync is obtained on the demodulator, the message is received into the RX queue. The header is checked, and if it is not a connectable advertising packet, reception is stopped and sync search is restarted. The bCrcErr and bIgnore bits are set according to the CRC result and the received message. The parameter pParams->initConfig.bUseWhiteList determines if the initiator must try to connect to a specific device or against the white list. If this parameter is 0, the white list is not used, and pParams->pWhiteList points to a buffer containing only the device address of the device to which to connect. The address type of the peer is given in pParams->advConfig.peerAddrType. Otherwise, pParams->pWhiteList points to a white list. If the white list is not used, the received AdvA field in the message is checked against the address found in pParams->pWhiteList, and the TXAdd bit of the received header is checked against pParams->initConfig.peerAddrType. If the white list is used, the received AdvA field in the message (along with the TXAdd bit of the received header) is checked against white list as described in Section 23.6.4.9. For ADV_DIRECT_IND messages, the received InitA field and RXAdd bit are checked against pParams->deviceAddr and pParams->initConfig.deviceAddrType, respectively. Depending on this, the actions taken are as listed in Table 23-126, where the definition of each action, including the value used on bCrcErr and bIgnore, is listed in Table 23-127. If pParams->initConfig.bStrictLenFilter is 1, only length fields compliant with the Bluetooth low energy specification are considered valid. For an ADV_DIRECT_IND, valid means a length field of 12, and for ADV_IND messages it means a length field in the range from 6 to 37. If pParams->initConfig.bStrictLenFilter is 0, all received packets with a length field less than or equal to the maximum length of an advertiser packet (37, if not overridden) are considered valid. If the length is not valid, the receiver is stopped.

Table 23-126. Actions on Packets Received by Initiator(1)

PDU Type	CRC Result	AdvA Match	InitA Match	Action Number
ADV_IND	OK	No	N/A	1
ADV_IND	OK	Yes	N/A	2
ADV_IND	NOK	X	N/A	3
ADV_DIRECT_IND	OK	No	X	1
ADV_DIRECT_IND	OK	Yes	No	1
ADV_DIRECT_IND	OK	Yes	Yes	2
ADV_DIRECT_IND	NOK	X	X	3
ADV*_IND with invalid length	X	X	X	4
Other	X	N/A	N/A	4

(1) X = don't care.

Table 23-127. Descriptions of the Actions to Take on Packets Received by Initiator

Action Number	bCrcErr	bIgnore	Description
1	0	1	Continue scanning
2	0	0	Send CONNECT_REQ and end operation
3	1	0	Continue scanning
4	—	—	Stop receiving packet, then continue scanning

If the packet received did not fit in the RX queue, the packet is received to the end, but the received bytes are not stored. If the packet would normally not have been discarded from the RX buffer, the operation ends.

If the action from the received packet is 2, a CONNECT_REQ packet is transmitted. When transmitting a CONNECT_REQ, the radio CPU constructs this packet. In the header, the PDU Type bits are 0101b. The TXAdd bit is as shown in pParams->initConfig.deviceAddrType. The RXAdd bit is as shown in the TXAdd field of the header of the received ADV_IND or ADV_DIRECT_IND message. The length is calculated from the length of the LLData, pParams->connectReqLen + 12. The RFU bits are 0. The payload starts with the 6-byte device address, read from pParams->pDeviceAddress, followed by the 6-byte peer address read from the AdvA field of the received message. The rest of the payload is read from the pParams->pConnectData buffer. If pParams->initConfig.bDynamicWinOffset is 1, the radio CPU replaces the bytes in the WinSize and WinOffset position with a calculated value as explained in the following paragraphs. After a CONNECT_REQ message has been transmitted, the operation ends.

Two triggers to end the operation are set up by pParams->endTrigger/pParams->endTime and pParams->timeoutTrigger/pParams->timeoutTime, respectively. If either of these triggers occurs, the radio operation ends as soon as possible. If these triggers occur while waiting for sync on an ADV*_IND packet, the operation ends immediately. If the triggers occur at another time, the operation continues until the scan would otherwise be resumed, and then ends. If the immediate command CMD_STOP is received by the radio CPU, it has the same meaning as the end trigger occurring, except that the status code after ending is CMD_DONE_STOPPED. The differences between the two triggers are the status and result at the end of the operation. Typically, timeoutTrigger is used at the end of a scan window, while endTrigger is used when scanning is to end entirely.

If pParams->initConfig.bDynamicWinOffset is 1, the radio CPU performs automatic calculation of the WinSize and WinOffset parameters in the transmitted message. WinSize is byte 7 of the payload, and WinOffset is byte 8 and 9. The radio CPU finds the possible start times of the first connection event from the pParams->connectTime parameter and the connection interval, which are given in 1.25-ms units by the interval field (byte 10 and 11) from the payload to be transmitted. The possible times of the first connection event are any whole multiple of connection intervals from pParams->connectTime, which may be in the past or the future from the start of the initiator command. The radio CPU calculates a WinOffset parameter to be inserted in the transmitted CONNECT_REQ. The calculated WinOffset ensures that the transmit window covers the first applicable connection event with enough margin after the end of the CONNECT_REQ packet. The radio CPU sets up the transmit window (WinOffset and WinSize) so that there is margin both between the start of the transmit window and the start of the first master packet, and between the start of the first master packet and the end of the transmit window. The inserted WinSize is either 1 or 2; ensuring such a margin. The radio CPU writes the calculated values for WinSize and WinOffset into the corresponding locations in the pParams->pConnectData buffer. The start time of the first connection event used to transmit the first packet within the signaled transmit window is written back by the radio CPU in pParams->connectTime. If no connection is made, the radio CPU adds a multiple of connection intervals to pParams->connectTime, so that it is the first possible time of a connection event after the operation ended.

The output structure pOutput contains fields that give information on running the command. The radio CPU does not initialize the fields, so this must be done by the system CPU when a reset of the counters is desired. The fields are updated by the radio CPU as described in the following list. The list also indicates when interrupts are raised in the system CPU.

- If a CONNECT_REQ packet has been transmitted, nTXConnectReq is incremented and a TX_DONE interrupt is raised.
- If an ADV*_IND packet is received with CRC OK and the bIgnore flag is cleared, nRxAdvOk is incremented, an RX_OK interrupt is raised, and timeStamp is set to a timestamp of the start of the packet.

- If an ADV*_IND packet is received with CRC OK and the bIgnore flag set, nRxAdvIgnored is incremented and an RX_IGNORED interrupt is raised.
- If an ADV*_IND packet is received with CRC error, nRxAdvNok is incremented and an RX_NOK interrupt is raised.
- If an ADV*_IND packet is received and did not fit in the RX queue, nRxAdvBufFull is incremented and an RX_BUF_FULL interrupt is raised.
- If a packet is received, lastRssi is set to the RSSI of that packet
- If the first RX data entry in the RX queue changed state to Finished after a packet was received, an RX_ENTRY_DONE interrupt is raised.

An initiator operation ends with one of the statuses listed in Table 23-128. After the operation has ended, the status field of the command structure (2 status bytes listed in Table 23-8) indicates why the operation ended. In all cases, a COMMAND_DONE interrupt is raised. In each case, it is indicated if the result is TRUE, FALSE, or ABORT, which decides the next action.

Table 23-128. End of Initiator Operation

Condition	Status Code	Result
Performed Action Number2 (transmitted CONNECT_REQ)	BLE_DONE_CONNECT	FALSE
Observed trigger indicated by pParams->timeoutTrigger while waiting for sync on ADV*_IND	BLE_DONE_RXTIMEOUT	TRUE
Observed trigger indicated by pParams->timeoutTrigger, then performed Action Number1, 2, 3, 4, or 5	BLE_DONE_RXTIMEOUT	TRUE
Observed trigger indicated by pParams->endTrigger while waiting for sync on ADV*_IND	BLE_DONE_ENDED	FALSE
Observed trigger indicated by pParams->endTrigger, then performed Action Number1, 2, 3, or 4	BLE_DONE_ENDED	FALSE
Observed CMD_STOP while waiting for sync on ADV*_IND	BLE_DONE_STOPPED	FALSE
Observed CMD_STOP, then performed Action Number1, 2, 3, or 4	BLE_DONE_STOPPED	FALSE
Received CMD_ABORT	BLE_DONE_ABORT	ABORT
No space in RX buffer to store received packet	BLE_ERROR_RXBUF	FALSE
Illegal value of channel	BLE_ERROR_PAR	ABORT
LLData length field has illegal value	BLE_ERROR_PAR	ABORT

23.6.4.7 Generic Receiver Command

The generic receiver command is used to receive physical layer test packets or to receive any packet, such as in a packet sniffer application.

A generic receiver operation is started by a CMD_BLE_GENERIC_RX command. In the command structure, CMD_BLE_GENERIC_RX has a pParams parameter of the type defined in Table 23-95, and a pOutput parameter of the type defined in Table 23-101. At the start of a generic receiver operation, the radio CPU waits for the start trigger, then programs the frequency based on the channel parameter of the command structure. The radio CPU sets up the access address defined in pParams->accessAddress and uses the CRC initialization value defined in pParams->crcInit. The whitener is set up as defined in the whitening parameter. The radio CPU then configures the receiver.

In a generic receiver operation, the only assumptions made on the packet format are that the 6 LSBs of the second received byte is a length field which indicates the length of the payload following that byte, and that a standard Bluetooth low-energy-type CRC is appended to the packet.

When tuned to the correct channel, the radio CPU starts listening for a packet. If sync is obtained on the demodulator, the message is received into the RX queue (if any). If the length is greater than the maximum allowed length for Bluetooth low energy advertising packets (37, but can be overridden), reception is stopped and restarted.

If pParams->pRxQ is NULL, the received packets are be stored. The counters are still updated and interrupts are generated.

If a packet is received with CRC error, the bCrcErr bit is set. The bIgnored flag is never set for the generic RX command.

If the packet being received did not fit in the RX queue, the packet is received to the end, but the received bytes are not stored. If the packet would normally not have been discarded from the RX buffer, the operation ends.

A trigger to end the operation is set up by pParams->endTrigger and pParams->endTime. If the trigger defined by this parameter occurs, the radio operation ends as soon as possible. If the trigger occurs while waiting for sync, the operation ends immediately. If the trigger occurs at another time, the operation continues until the current packet is fully received, and then ends. If the immediate command CMD_STOP is received by the radio CPU, it has the same meaning as the end trigger occurring, except that the status code after ending is CMD_DONE_STOPPED. The output structure pOutput contains fields that give information on the command being run. The radio CPU does not initialize the fields, so this must be done by the system CPU when a reset of the counters is desired. The fields are updated by the radio CPU, as described in the following list. The list also indicates when interrupts are raised in the system CPU.

- If a packet is received with CRC OK, nRxOk is incremented and an RX_OK interrupt is raised.
- If a packet is received with CRC error, nRxNok is incremented and an RX_NOK interrupt is raised.
- If a packet is received and did not fit in the RX queue, nRxBufFull is incremented and an RX_BUF_FULL interrupt is raised.
- If a packet is received, lastRssi is set to the RSSI of that packet
- If a packet is received, timeStamp is set to a timestamp of the start of that packet
- If the first RX data entry in the RX queue changed state to Finished after a packet was received, an RX_ENTRY_DONE interrupt is raised.

When a packet is received, reception is restarted on the same channel if pParams->bRepeat = 1, the end event has not been observed, and the packet fits in the receive queue. If pParams->bRepeat = 0, the operation always ends when a packet is received.

A generic RX operation ends with one of the statuses listed in Table 23-129. After the operation has ended, the status field of the command structure (2 status bytes listed in Table 23-8) indicates why the operation ended. In all cases, a COMMAND_DONE interrupt is raised. In each case, it is indicated if the result is TRUE, FALSE, or ABORT, which decides the next action. The pNextOp field of a generic RX command structure may point to the same command structure. That way, RX may be performed until the end trigger, or until the RX buffer becomes full.

Table 23-129. End of Generic RX Operation

Condition	Status Code	Result
Received a packet with CRC OK and pParams->bRepeat = 0	BLE_DONE_OK	TRUE
Received a packet with CRC error and pParams->bRepeat = 0	BLE_DONE_RXERR	TRUE
Observed trigger indicated by pParams->endTrigger while waiting for sync	BLE_DONE_ENDED	FALSE
Observed trigger indicated by pParams->endTrigger, then finished receiving packet	BLE_DONE_ENDED	FALSE
Observed CMD_STOP while waiting for sync	BLE_DONE_STOPPED	FALSE
Observed CMD_STOP, then finished receiving packet	BLE_DONE_STOPPED	FALSE
Received CMD_ABORT	BLE_DONE_ABORT	ABORT
No space in RX buffer to store received packet	BLE_ERROR_RXBUF	FALSE
Illegal value of channel	BLE_ERROR_PAR	ABORT

23.6.4.8 PHY Test Transmit Command

The test packet transmitter command may be used to transmit physical layer test packets.

A test packet transmitter operation is started by a CMD_BLE_TX_TEST command. In the command structure, CMD_BLE_TX_TEST has a pParams parameter of the type defined in Table 23-96, and a pOutput parameter of the type defined in Table 23-102. At the start of a test TX operation, the radio CPU waits for the start trigger, then programs the frequency based on the channel parameter of the command structure. The radio CPU sets up the test mode packet access address and uses the CRC initialization value 0x55 5555. The whitener is set up as defined in the whitening parameter. To produce PHY test packets conforming to the Bluetooth low energy Test Specification, the whitener must be disabled.

The radio CPU transmits pParams->numPackets packets, then ends the operation. If pParams->numPackets is 0, transmission continues until the operation ends for another reason (time-out, stop, or abort command). The time (number of RAT ticks) between the start of each packet is given by pParams->period. If this time is smaller than the duration of a packet, each packet is transmitted as soon as possible. Each packet is assembled as follows by the radio CPU. The first byte is a header byte, containing the value of pParams->packetType, provided this is one of the values listed in Table 23-130. The next byte is the length byte, which is the value of pParams->payloadLength, and is followed by a number of payload bytes, which are as listed in Table 23-130. The number of payload bytes is equal to pParams->payloadLength. If pParams->packetType is 0, the bytes are from the PRBS9 sequence. Otherwise, all the bytes are the same, as listed in Table 23-130. A 3-byte CRC, according to the Bluetooth low energy specification, is appended.

Table 23-130. Supported PHY Test Packet Types

Value of Packet Type	Transmitted Bytes
0	PRBS9 sequence
1	Repeated 0x0F
2	Repeated 0x55
3	PRBS15 sequence
4	Repeated 0xFF
5	Repeated 0x00
6	Repeated 0xF0
7	Repeated 0xAA

The PRBS15 payload type defined in the Bluetooth low energy standard, which corresponds to payload type 3, is implemented using the polynomial $x^{15} + x^{14} + 1$. The initialization is taken from the RAT for the first packet transmitted, and is not reinitialized for subsequent packets.

If pParams->config.overrideDefault is 1, the packet is nonstandard. The header contains the value given in pParams->packetType, and each byte transmitted is as given in pParams->byteVal. If pParams->config.bUsePrbs9 is 1, the sequence is generated by XORing each byte of the PRBS9 sequence used for packet type 0 with pParams->byteVal. If pParams->config.bUsePrbs15 is 1, the sequence is generated by XORing each byte of the PRBS15 sequence used for packet type 3 with pParams->byteVal.

If either of the PRBS sequences is used, whitening is disabled regardless of the setting in the whitening parameter.

A trigger to end the operation is set up by pParams->endTrigger and pParams->endTime. If the trigger defined by this parameter occurs, the radio operation ends as soon as possible. If the trigger occurs while waiting between packets, the operation ends immediately. If the trigger occurs at another time, the operation continues until the current packet is fully transmitted, and then ends. If the immediate command CMD_STOP is received by the radio CPU, it has the same meaning as the end trigger occurring, except that the status code after ending is CMD_DONE_STOPPED.

The output structure pOutput contains only the field nTX, and is incremented each time a packet is transmitted. The radio CPU does not initialize the field, so this must be done by the system CPU when a reset of the counters is desired. A TX_DONE interrupt is raised each time a packet is transmitted.

A PHY test TX operation ends with one of the statuses listed in Table 23-131. After the operation has ended, the status field of the command structure (2 status bytes listed in Table 23-8) indicates why the operation ended. In all cases, a COMMAND_DONE interrupt is raised. In each case, it is indicated if the result is TRUE, FALSE, or ABORT, which decides the next action.

Table 23-131. End of PHY Test TX Operation

Condition	Status Code	Result
Transmitted pParams->numPackets packets	BLE_DONE_OK	TRUE
Observed trigger indicated by pParams->endTrigger while waiting between packets	BLE_DONE_ENDED	FALSE
Observed trigger indicated by pParams->endTrigger, then finished transmitting packet	BLE_DONE_ENDED	FALSE
Observed CMD_STOP while waiting between packets.	BLE_DONE_STOPPED	FALSE
Observed CMD_STOP, then finished transmitting packet	BLE_DONE_STOPPED	FALSE
Received CMD_ABORT	BLE_DONE_ABORT	ABORT
Illegal value of channel	BLE_ERROR_PAR	ABORT
Illegal value of pParams->packetType	BLE_ERROR_PAR	ABORT

23.6.4.9 White List Processing

A white list is used in advertiser, scanner, and initiator operation. The white list consists of a configurable number of entries. The white list is an array of entries of the type defined in Table 23-71. The first entry of the array contains the array size in the size field.

The minimum number of entries in a white list array is 1, but if no white list is to be used, pParams->pWhiteList may be NULL. The maximum number is at least 8.

Each entry contains one address and three configuration bits. The bEnable bit is 1 if the entry is enabled, otherwise the address is ignored when doing white-list filtering. The addrType bit indicates if the entry is a public or random address. The bIgnore bit can be used by a scanner to avoid reporting and scanning the same device multiple times.

When an address is checked against the white list, the address is compared against the address field of each entry in the white list. The address is considered present in the white list only if there is an entry where one or all of the following conditions are met:

- The bEnable bit is 1.
- addrType is equal to the address type of the address to check.
- All bytes of the address array are equal to the bytes of the address to check.
- For scanner only: the bWlIgn bit is 0.

For scanners, the bWlIgn bit may be set in the white list to indicate that a device is ignored even if the white list entry would otherwise be a match. This feature can be used to check for advertisers that have already been scanned, or where the advertising data has already been reported. Even if no white list filtering is performed, this feature may be used. The white list is scanned for devices that match the address and address type, and where bWlIgn is 1. Such devices are ignored. The bEnable bit is not checked in this case. It is possible to configure the radio CPU to automatically set the bWlIgn bit, see Section 23.6.4.5.

23.6.5 Immediate Commands

In addition to the immediate commands from Section 23.3.4, the following immediate command is also supported.

23.6.5.1 Update Advertising Payload Command

The CMD_BLE_ADV_PAYLOAD command can change the payload buffer for an advertising command. The command may be issued regardless of whether an advertising command is running or not.

The command structure has the format given in Table 23-89. When received, the radio CPU checks if an advertiser radio operation command is running, using the parameter structure given in pParams of the immediate command structure. If the advertiser radio operation command is not running, the radio CPU updates the parameter structure immediately. If a radio operation command is running using the parameter structure to be updated, the radio CPU only modifies the parameter structure if the payload to be changed is not currently being transmitted. If the payload to be changed is being transmitted, the radio CPU stores the request and updates as soon as transmission of the packet has finished.

When updating the parameter structure, the payload to change depends on the payloadType parameter of the command structure. If payloadType is 0, the radio CPU sets pParams->advLen equal to newLen and pParams->pAdvData equal to newData. If payloadType is 1, the radio CPU sets pParams->scanRspLen equal to newLen and pParams->pScanRspData equal to newData. After the update occurs, the radio CPU raises a TX_BUFFER_CHANGED interrupt (see Section 23.8.2.5). This interrupt is raised regardless of whether the update was delayed or not.

If any of the parameters are illegal, the radio CPU responds with ParError in CMDSTAT and does not perform any update. Otherwise, the radio CPU responds with Done in CMDSTAT, which may be done before the update occurs.

23.7 Proprietary Radio

This section describes proprietary radio command structure, data handling, radio operations commands, and immediate commands. The commands define a flexible packet handling compatible with the CC110x, CC111x, CC112x, CC120x, CC2500, and CC251x devices, as well as supporting other legacy modes.

23.7.1 Packet Formats

For compatibility with existing TI parts, the packet format given in Figure 23-9 can be used in most cases. This packet format is supported through the use of the commands CMD_PROP_TX and CMD_PROP_RX.

Figure 23-9. Standard Packet Format

1 bit to 32 bytes	8 to 32 bits	0 or 1 byte	0 or 1 byte	0 to 255 bytes	0 or 16 bits (0 to 32 bits)
Preamble	Sync word	Length field	Address	Payload	CRC

A more flexible packet format is also possible, as defined in Figure 23-10. This format is supported by the commands CMD_PROP_RX_ADV and CMD_PROP_TX_ADV. The format in Figure 23-9 is an example of this.

Figure 23-10. Advanced Packet Format

1 bit to 32 bytes or repetition	8 to 32 bits	0 to 32 bits	0 to 8 bytes	Arbitrary	0 or 16 bits (0 to 32 bits)
Preamble	Sync word	Header	Address	Payload	CRC

23.7.2 Commands

Table 23-132 defines the proprietary radio operation commands.

Table 23-132. Proprietary Radio Operation Commands

ID	Command Name	Supported Devices	Description
0x3801	CMD_PROP_TX	CC26x0, CC2640R2F, CC13x0	Transmit packet
0x3802	CMD_PROP_RX	CC26x0, CC2640R2F, CC13x0	Receive packet or packets

Table 23-132. Proprietary Radio Operation Commands (continued)

ID	Command Name	Supported Devices	Description
0x3803	CMD_PROP_TX_ADV	CC26x0, CC2640R2F, CC13x0	Transmit packet with advanced modes
0x3804	CMD_PROP_RX_ADV	CC26x0, CC2640R2F, CC13x0	Receive packet or packets with advanced modes
0x3805	CMD_PROP_CS	CC2640R2F, CC13x0	Run carrier sense command
0x3806	CMD_PROP_RADIO_SETUP	CC26x0, CC2640R2F, CC1350	Set up radio in proprietary mode (used only on CC1350 when operating at 2.4 GHz)
0x3807	CMD_PROP_RADIO_DIV_SETUP	CC13x0	Set up radio in proprietary mode
0x3808	CMD_PROP_RX_SNIFF	CC2640R2F, CC13x0	Receive packet or packets with sniff mode support
0x3809	CMD_PROP_RX_ADV_SNIFF	CC2640R2F, CC13x0	Receive packet or packets with advanced modes and sniff mode support

Table 23-133 defines the proprietary immediate commands.

Table 23-133. Proprietary Immediate Commands

ID	Command Name	Description
0x3401	CMD_PROP_SET_LEN	Set length of packet being received
0x3402	CMD_PROP_RESTART_RX	Stop receiving a packet and go back to sync search

23.7.2.1 Command Data Definitions

This section defines data types used in describing the data structures used for communication between the system CPU and the radio CPU. The data structures are listed with tables. The Byte Index is the offset from the pointer to that structure. Multibyte fields are little-endian, and halfword or word alignment is required. For bit numbering, 0 is the LSB. The R/W column is used as follows:

R: The system CPU can read a result back; the radio CPU does not read the field.

W: The system CPU writes a value, the radio CPU reads it and does not modify the value.

R/W: The system CPU writes an initial value, the radio CPU may modify the initial value.

23.7.2.1.1 Command Structures

For all the radio operation commands, the first 14 bytes are as defined in Table 23-8. Table 23-134 through Table 23-140 define the additional command structures.

Table 23-134. CMD_PROP_TX Command Structure

Byte Index	Field Name	Bits	Bit Field name	Type	Description
14	pktConf	0	bFsOff	W	0: Keep frequency synthesizer on after command. 1: Turn frequency synthesizer off after command.
		1–2			Reserved
		3	bUseCrc	W	0: Do not append CRC. 1: Append CRC.
		4	bVarLen	W	0: Fixed length 1: Transmit length as first byte
		5–7			Reserved
15	pktLen			W	Packet length
16–19	syncWord			W	Sync word to transmit

Table 23-134. CMD_PROP_TX Command Structure (continued)

Byte Index	Field Name	Bits	Bit Field name	Type	Description
20–23	pPkt			W	Pointer to packet

Table 23-135. CMD_PROP_TX_ADV Command Structure

Byte Index	Field Name	Bits	Bit Field name	Type	Description
14	pktConf	0	bFsOff	W	0: Keep frequency synthesizer on after command. 1: Turn frequency synthesizer off after command.
		1–2			Reserved
		3	bUseCrc	W	0: Do not append CRC. 1: Append CRC.
		4	bCrcIncSw	W	0: Do not include sync word in CRC calculation. 1: Include sync word in CRC calculation.
		5	bCrcIncHdr	W	0: Do not include header in CRC calculation. 1: Include header in CRC calculation.
		6–7			Reserved
15	numHdrBits			W	Number of bits in header (0 to 32)
16–17	pktLen			W	Packet length. 0: Unlimited
18	startConf	0	bExtTxTrig	W	0: Start packet on a fixed time from the command start trigger. 1: Start packet on an external trigger (Contact TI to enable this feature).
		1–2	inputMode	W	Input mode if external trigger is used for TX start. 00: Rising edge 01: Falling edge 10: Both edges 11: Reserved
		3–7	source	W	RAT input event number used for capture if external trigger is used for TX start.
19	preTrigger			W	Trigger for transition from preamble to sync word. If this is set to “now," one preamble as configured in the setup is sent. Otherwise, the preamble is repeated until this trigger is observed.
20–23	preTime			W	Time parameter for preTrigger
24–27	syncWord			W	Sync word to transmit
28–31	pPkt			W	Pointer to packet, or TX queue for unlimited length

Table 23-136. CMD_PROP_RX and CMD_PROP_RX_SNIFF Command Structure

Byte Index	Field Name	Bits	Bit Field Name	Type	Description
14	pktConf	0	bFsOff	W	0: Keep frequency synthesizer on after command. 1: Turn frequency synthesizer off after command.
		1	bRepeatOk	W	0: End operation after receiving a packet correctly. 1: Go back to sync search after receiving a packet correctly.
		2	bRepeatNok	W	0: End operation after receiving a packet with CRC error. 1: Go back to sync search after receiving a packet with CRC error.
		3	bUseCrc	W	0: Do not check CRC. 1: Check CRC.
		4	bVarLen	W	0: Fixed length 1: Receive length as first byte.
		5	bChkAddress	W	0: No address check 1: Check address.
		6	endType	W	0: Packet is received to the end if end trigger occurs after sync is obtained. 1: Packet reception is stopped if end trigger occurs.
		7	filterOp	W	0: Stop receiver and restart sync search on address mismatch. 1: Receive packet and mark it as ignored on address mismatch.
15	rxConf			W	RX configuration, see Table 23-143 for details.
16–19	syncWord			W	Sync word to listen for
20	maxPktLen			W	Packet length for fixed length, maximum packet length for variable length 0: Unlimited or unknown length
21	address0			W	Address
22	address1			W	Address (Set equal to address0 to accept only one address. If 0xFF, accept 0x00 as well.)
23	endTrigger			W	Trigger classifier for ending the operation
24–27	endTime			W	Time to end the operation
28–31	pQueue			W	Pointer to receive queue
32–35	pOutput			W	Pointer to output structure
36–47	CMD_PROP_RX_SNIFF only: carrier sense options as given in Table 23-142 (CC13x0 only)				

Table 23-137. CMD_PROP_RX_ADV and CMD_PROP_RX_ADV_SNIFF Command Structure

Byte Index	Field Name	Bits	Bit Field Name	Type	Description
14	pktConf	0	bFsOff	W	0: Keep frequency synthesizer on after command. 1: Turn frequency synthesizer off after command.
		1	bRepeatOk	W	0: End operation after receiving a packet correctly. 1: Go back to sync search after receiving a packet correctly.
		2	bRepeatNok	W	0: End operation after receiving a packet with CRC error. 1: Go back to sync search after receiving a packet with CRC error.
		3	bUseCrc	W	0: Do not check CRC. 1: Check CRC.
		4	bCrcIncSw	W	0: Do not include sync word in CRC calculation. 1: Include sync word in CRC calculation.
		5	bCrcIncHdr	W	0: Do not include header in CRC calculation. 1: Include header in CRC calculation.
		6	endType	W	0: Packet is received to the end if end trigger occurs after sync is obtained. 1: Packet reception is stopped if end trigger occurs.
		7	filterOp	W	0: Stop receiver and restart sync search on address mismatch. 1: Receive packet and mark it as ignored on address mismatch.
15	rxConf			W	RX configuration, see Table 23-143 for details.
16–19	syncWord0			W	Sync word to listen for
20–23	syncWord1			W	Alternative sync word if nonzero
24–25	maxPktLen			W	Maximum length of received packets: 0: Unlimited or unknown length
26–27	hdrConf	0–5	numHdrBits	W	Number of bits in header (0–32)
		6–10	lenPos	W	Position of length field in header (0–31)
		11–15	numLenBits	W	Number of bits in length field (0–16)
28–29	addrConf	0	addrType	W	0: Address after header 1: Address in header
		1–5	addrSize	W	If addrType = 0: Address size in bytes. If addrType = 1: Address size in bits.
		6–10	addrPos	W	If addrType = 1: Bit position of address in header. If addrType = 0: Nonzero to extend address with sync word identifier.
		11–15	numAddr	W	Number of addresses in address list
30	lenOffset			W	Signed value to add to length field
31	endTrigger			W	Trigger classifier for ending the operation
32–35	endTime			W	Time to end the operation
36–39	pAddr			W	Pointer to address list
40–43	pQueue			W	Pointer to receive queue
44–47	pOutput			W	Pointer to output structure
48–59	CMD_PROP_RX_ADV_SNIFF only: carrier sense options as given in Table 23-142 (CC13x0 only)				

Table 23-138. CMD_PROP_CS Command Structure (CC13x0 Only)

Byte Index	Field Name	Bits	Bit Field Name	Type	Description
14	csFsConf	0	bFsOffIdle	W	0: Keep synthesizer running if command ends with channel IDLE. 1: Turn off synthesizer if command ends with channel IDLE.
		1	bFsOffBusy	W	0: Keep synthesizer running if command ends with channel BUSY. 1: Turn off synthesizer if command ends with channel BUSY.
15					Reserved
16–27	Carrier sense options as given in Table 23-142.				

Table 23-139. CMD_PROP_RADIO_SETUP and CMD_PROP_RADIO_DIV_SETUP Command Structure

Byte Index	Field Name	Bits	Bit Field Name	Type	Description
14–15	modulation	0–2	modType	W	0: FSK 1: GFSK Others: Reserved
		3–15	deviation	W	Deviation (250-Hz steps) for FSK modulations
16–19	symbolRate	0:3	preScale	W	Prescaler value (see Section 23.7.5.2)
		4–7			Reserved, set to 0
		8–28	rateWord	W	Rate word (see Section 23.7.5.2)
		29–31			Reserved, set to 0
20	rxBw			W	Receiver bandwidth, see Table 23-147 1–18: Legacy mode (bandwidth 88–4240 kHz) (CC26x0 and CC13x0) 32–52: Normal mode (bandwidth 45–4240 kHz) (CC13x0)
21	preamConf	0–5	nPreamBytes	W	0: 1 preamble bit 1–16: Number of preamble bytes 18, 20, ..., 30: Number of preamble bytes 31: 4 preamble bits 32: 32 preamble bytes Others: Reserved
		6–7	preamMode	W	00: Send 0 as the first preamble bit. 01: Send 1 as the first preamble bit. 10: Send same first bit in preamble and sync word. 11: Send different first bit in preamble and sync word.

Table 23-139. CMD_PROP_RADIO_SETUP and CMD_PROP_RADIO_DIV_SETUP Command Structure (continued)

Byte Index	Field Name	Bits	Bit Field Name	Type	Description
22–23	formatConf	0–5	nSwBits	W	Number of sync word bits. Valid values are from 8 to 32.
		6	bBitReversal	W	0: Use positive deviation for 1. 1: Use positive deviation for 0.
		7	bMsbFirst	W	0: LSB transmitted first 1: MSB transmitted first
		8–11	fecMode	W	Select Coding: 0000: Uncoded binary modulation 1000: Long Range Mode 1010: Manchester coded binary modulation (only CC13x0 FSK/GFSK) Others: Reserved
		12			Reserved
		13–15	whitenMode	W	000: No whitening 001: CC1101 and CC2500 compatible whitening 010: PN9 whitening without byte reversal 011: Reserved 100: No whitener, 32-bit IEEE 802.15.4g compatible CRC (only CC13x0) 101: IEEE 802.15.4g compatible whitener and 32-bit CRC (only CC13x0) 110: No whitener, dynamically IEEE 802.15.4g compatible 16-bit or 32-bit CRC (only CC13x0) 111: Dynamically IEEE 802.15.4g compatible whitener and 16-bit or 32-bit CRC (only CC13x0)
24–25	config	0–2	frontEndMode	W	0x00: Differential mode 0x01: Single-ended mode RFP 0x02: Single-ended mode RFN 0x05 Single-ended mode RFP with external front-end control on RF pins (RFN and RXTX) 0x06 Single-ended mode RFN with external front-end control on RF pins (RFP and RXTX) Others: Reserved
		3	biasMode	W	0: Internal bias 1: External bias
		4-9	analogCfgMode	W	0x00: Write analog configuration. Required first time after boot and when changing frequency band or front-end configuration. 0x2D: Keep analog configuration. May be used after standby or when changing mode with the same frequency band and front-end configuration. Others: Reserved
		10	bNoFsPowerUp	W	0: Power up frequency synthesizer. 1: Do not power up frequency synthesizer.
		11–15			Reserved
26–27	txPower			W	Output power setting, use value from SmartRF Studio. See Section 23.3.3.2.16 for more details.
28–31	pRegOverride			W	Pointer to a list of hardware and configuration registers to override. If NULL, no override is used.
32–33	centerFreq			W	CMD_PROP_RADIO_DIV_SETUP only: Center frequency of the band. To be used in the initial parameter computations.
34–35	intFreq			W	CMD_PROP_RADIO_DIV_SETUP only: Intermediate frequency to use for RX, in MHz on 4.12 signed format. TX will use same intermediate frequency if supported, otherwise 0. 0x8000: Use default.
36	loDivider			W	CMD_PROP_RADIO_DIV_SETUP only: Divider setting to use. See the Smart RF Studio for the recommended settings per device and band.

Table 23-140. CMD_PROP_SET_LEN Command Structure

Byte Index	Field Name	Bits	Bit Field Name	Type	Description
0–1	RXLen			W	Payload length to use

23.7.2.2 Output Structures

Table 23-141. Receive Commands

Byte Index	Field Name	Type	Description
0–1	nRxOk	R/W	Number of packets that have been received with payload, CRC OK and not ignored
2–3	nRxNok	R/W	Number of packets that have been received with CRC error
4	nRxIgnored	R/W	Number of packets that have been received with CRC OK and ignored due to address mismatch
5	nRxStopped	R/W	Number of packets not received due to illegal length or address mismatch with pktConf.filterOp = 1
6	nRxBufFull	R/W	Number of packets that have been received and discarded due to lack of buffer space
7	lastRssi	R	RSSI of last received packet. RSSI is captured when sync word is found.
8–11	timeStamp	R	Timestamp of last received packet

23.7.2.3 Other Structures and Bit Fields

Table 23-142. Carrier Sense Fields for CMD_PROP_RX_SNIFF, CMD_PROP_RX_ADV_SNIFF, and CMD_PROP_CS (Only Applicable for CC13x0)

Byte Index	Field Name	Bits	Bit Field Name	Type	Description
0	csConf	0	bEnaRssi	W	If 1, enable RSSI as a criterion.
		1	bEnaCorr	W	If 1, enable correlation as a criterion.
		2	operation	W	0: Busy if either RSSI or correlation indicates BUSY. 1: Busy if both RSSI and correlation indicates BUSY.
		3	busyOp	W	0: Continue carrier sense on channel BUSY. 1: End carrier sense on channel BUSY. For an RX command, the receiver continues when carrier sense ends, then it does not end if the channel goes IDLE.
		4	idleOp	W	0: Continue on channel Idle. 1: End on channel Idle.
		5	timeoutRes	W	0: Time-out with channel state Invalid treated as BUSY. 1: Time-out with channel state Invalid treated as IDLE.
1	rssiThr			W	RSSI threshold
2	numRssiIdle			W	Number of consecutive RSSI measurements below the threshold needed before the channel is declared IDLE.
3	numRssiBusy			W	Number of consecutive RSSI measurements above the threshold needed before the channel is declared BUSY.
4–5	corrPeriod			W	Number of RAT ticks for a correlation observation periods.
6	corrConfig	0–3	numCorrInv	W	Number of subsequent correlation tops with maximum corrPeriod RAT ticks between them needed to go from IDLE to INVALID.
		4–7	numCorrBusy	W	Number of subsequent correlation tops with maximum corrPeriod RAT ticks between them needed to go from INVALID to BUSY.
7	csEndTrigger			W	Trigger classifier for ending the carrier sense
8–11	csEndTime			W	Time to end carrier sense.

Table 23-143. Receive Queue Entry Configuration Bit Field(1)

Bits	Bit Field Name	Description
0	bAutoFlushIgnored	If 1, automatically discard ignored packets from RX queue.
1	bAutoFlushCrcErr	If 1, automatically discard packets with CRC error from RX queue.
2		Reserved
3	bIncludeHdr	If 1, include the received header or length byte in the stored packet; otherwise discard it.
4	bIncludeCrc	If 1, include the received CRC field in the stored packet; otherwise discard it. This requires pktConf.bUseCrc to be 1.
5	bAppendRssi	If 1, append an RSSI byte to the packet in the RX queue.
6	bAppendTimestamp	If 1, append a timestamp to the packet in the RX queue.
7	bAppendStatus	If 1, append a status byte to the packet in the RX queue.

(1) This bit field is used for the rxConf byte of the parameter structures.

Table 23-144. Receive Status Byte Bit Field(1)

Bits	Bit Field Name	Description
0–4	addressInd	Index of address found (0 if not applicable)
5	syncWordId	0 for primary sync word, 1 for alternate sync word
6–7	result	00: Packet received correctly, not ignored 01: Packet received with CRC error 10: Packet received correctly, but can be ignored 11: Packet reception was aborted

(1) A byte of this bit field is appended to the received entries if configured.

23.7.3 Interrupts

The radio CPU signals events back to the system CPU using firmware-defined interrupts. Table 23-145 lists the interrupts to be used by the proprietary commands. Each interrupt may be enabled individually in the system CPU. Details for when the interrupts are generated are given in Section 23.7.4 and Section 23.7.5.

Table 23-145. Interrupt Definitions

Interrupt Number	Interrupt Name	Description
0	COMMAND_DONE	A radio operation command has finished.
1	LAST_COMMAND_DONE	The last radio operation command in a chain of commands has finished.
10	TX_ENTRY_DONE	For transmission of packets with unlimited length: Reading from a TX entry is finished.
16	RX_OK	Packet received with CRC OK, payload, and is not to be ignored.
17	RX_NOK	Packet received with CRC error.
18	RX_IGNORED	Packet received with CRC OK, but is to be ignored.
22	RX_BUF_FULL	Packet received did not fit in RX buffer.
23	RX_ENTRY_DONE	RX queue data entry changing state to FINISHED.
24	RX_DATA_WRITTEN	Data written to partial read RX buffer.
25	RX_N_DATA_WRITTEN	Specified number of bytes written to partial read RX buffer.
26	RX_ABORTED	Packet reception stopped before packet was done.
28	SYNTH_NO_LOCK	The synthesizer has reported loss of lock (only valid for CC13x0).
29	MODULES_UNLOCKED	As part of the boot process, the Cortex-M0 has opened access to RF core modules and memories.
30	BOOT_DONE	The RF core CPU boot is finished.
31	INTERNAL_ERROR	The radio CPU has observed an unexpected error.

23.7.4 Data Handling

For the proprietary mode TX commands, data received over the air is stored in a receive queue. Partial-read RX buffers are supported, and mandatory for unlimited length. Data transmitted is fetched from a specific buffer.

23.7.4.1 Receive Buffers

A packet being received is stored in an RX buffer. First, a length byte or word is stored if configured in the RX entry by config.lenSz, and calculated from the length received over the air and the configuration of appended information, or for a partial-read RX buffer initialized to maximal possible size of that segment, and set to the length of the segment in one buffer when finished.

Following the optional length field, the received header is stored as received over the air if rxConf.bIncludeHdr is 1. This header is the length byte for CMD_PROP_RX and a field with up to 32 bits for CMD_PROP_RX_ADV. In the case of the 32-bits header for the CMD_PROP_RX_ADV, the last byte of the header is padded with zeros in the MSBs if the number of bits does not divide by 8, and is followed by the received address (if configured) and the payload.

If rxConf.bIncludeCrc is 1, the received CRC value is stored in the RX buffer; otherwise, it is not stored, but only used to check the CRC result. If rxConf.bAppendRssi is 1, a byte indicating the received RSSI value is appended. If rxConf.bAppendStatus is 1, a status byte of the type defined in Table 23-144 is appended. If rxConf.bAppendTimeStamp is 1, a timestamp indicating the start of the packet is appended. This timestamp corresponds to the ratmr_t data type. Though the timestamp is multibyte, no word-address alignment is made, so the timestamp must be written and read byte-wise.

If the reception of a packet is aborted, the packet is immediately removed from the receive queue, except if a partial-read RX entry is used. In that case, the RSSI, Timestamp, and Status fields are appended if configured (except if no more buffer space is available), and the Status byte indicates that the reception was aborted.

Figure 23-11 shows the format of an entry element in the RX queue.

Figure 23-11. Receive Buffer Entry Element

0–2 bytes	0–4 bytes	*n* bytes	0–4 bytes	0 or 1 byte	0 or 4 bytes	0 or 1 byte
Element length	Header/length byte	Payload	Received CRC	RSSI	Timestamp	Status

An RX_ENTRY_DONE interrupt is raised whenthe state of an RX entry changes to FINISHED. Depending on the type of RX entry used, this means:

- For a general or pointer entry, an RX_ENTRY_DONE interrupt is raised after a packet is fully received, unless the packet is automatically flushed.
- For a multielement entry, an RX_ENTRY_DONE interrupt is raised when a new buffer is allocated and a new entry was taken into use, or when a buffer is finished and fills the entire entry.
- For a partial-read entry, an RX_ENTRY_DONE interrupt is raised when an RX entry is full, so writing must continue in the next entry.

For partial-read entries, an RX_Data_Written interrupt is raised whenever data is written to the receive buffer. An RX_N_Data_Written interrupt is raised whenever a multiple of config.irqIntv (as given in the data entry) bytes have been written since the start of the packet.

23.7.4.2 Transmit Buffers

The transmit operations contain a buffer with the data to be transmitted. The number of bytes in this buffer is given by pktLen. For the CMD_PROP_TX command, the length given in pktLen is transmitted as the first byte if pktConf.bVarLen is 1, and then followed by the contents of the transmit buffer.

For CMD_PROP_TX_ADV, the first bytes of the buffer contain the header if the header length is greater than 0. The number of bytes is the number of bits in the header divided by 8, rounded up. The MSBs of the last header byte are not sent if the number of bits does not divide by 8. If a length field is to be transmitted using CMD_PROP_TX_ADV, it must be given explicitly from the system side as part of the header.

If unlimited length is configured, a TX queue is used instead of one buffer. In this case, transmission of payload continues until the queue is emptied. Every time transmission from one entry is finished, meaning reading continues from the next entry or the entire payload is entered into the modem, a TX_ENTRY_DONE interrupt is raised.

23.7.5 Radio Operation Command Descriptions

Before running any of the proprietary RX or TX radio operation commands, the radio must be set up in proprietary mode using the command CMD_PROP_RADIO_SETUP or CMD_PROP_RADIO_DIV_SETUP, or in another compatible mode with CMD_RADIO_SETUP. Otherwise, the operation ends with an error. The RX and TX commands also require the CMD_FS command to program the synthesizer, which can typically be done by a command chain where an RX or TX command follows immediately after the CMD_FS.

23.7.5.1 End of Operation

The status field of the command issued is updated during the operation. When submitting the command, the system CPU must write this field with a state of IDLE. During the operation, the radio CPU updates the field to indicate the operation mode. When the operation is done, the radio CPU writes a status indicating that the operation is finished. Table 23-146 lists the status codes used by a proprietary radio operation.

Table 23-146. Proprietary Radio Operation Status Codes

Number	Name	Description
Operation not finished		
0x0000	IDLE	Operation not started
0x0001	PENDING	Waiting for start trigger
0x0002	ACTIVE	Running operation
Operation finished normally		
0x3400	PROP_DONE_OK	Operation ended normally
0x3401	PROP_DONE_RXTIMEOUT	Operation stopped after end trigger while waiting for sync
0x3402	PROP_DONE_BREAK	RX stopped due to time-out in the middle of a packet
0x3403	PROP_DONE_ENDED	Operation stopped after end trigger during reception
0x3404	PROP_DONE_STOPPED	Operation stopped after stop command
0x3405	PROP_DONE_ABORT	Operation aborted by abort command
0x3406	PROP_DONE_RXERR	Operation ended after receiving packet with CRC error
0x3407	PROP_DONE_IDLE	Carrier sense operation ended because of idle channel (valid only for CC13x0)
0x3408	PROP_DONE_BUSY	Carrier sense operation ended because of busy channel (valid only for CC13x0)
0x3409	PROP_DONE_IDLETIMEOUT	Carrier sense operation ended because of time-out with csConf.timeoutRes = 1 (valid only for CC13x0)
0x340A	PROP_DONE_BUSYTIMEOUT	Carrier sense operation ended because of time-out with csConf.timeoutRes = 0 (valid only for CC13x0)
Operation finished with error		
0x3800	PROP_ERROR_PAR	Illegal parameter
0x3801	PROP_ERROR_RXBUF	No RX buffer large enough for the received data available at the start of a packet
0x3802	PROP_ERROR_RXFULL	Out of RX buffer during reception in a partial read buffer
0x3803	PROP_ERROR_NO_SETUP	Radio was not set up in proprietary mode
0x3804	PROP_ERROR_NO_FS	Synthesizer was not programmed when running RX or TX

Table 23-146. Proprietary Radio Operation Status Codes (continued)

Number	Name	Description
0x3805	PROP_ERROR_RXOVF	TX overflow observed during operation
0x3806	PROP_ERROR_TXUNF	TX underflow observed during operation

The conditions for giving each status are listed for each operation. Some of the error causes listed in Table 23-146 are not repeated in these lists. If CMD_STOP or CMD_ABORT is received while waiting for the start trigger, the end cause is DONE_STOPPED or DONE_ABORT, with an end result of FALSE and ABORT, respectively. In some cases, general error causes may occur. For all these error cases, the result of the operation is ABORT.

23.7.5.2 Proprietary Mode Setup Command

For proprietary mode radio, the CMD_PROP_RADIO_SETUP and CMD_PROP_RADIO_DIV_SETUP commands are used instead of CMD_RADIO_SETUP. When CMD_PROP_RADIO_SETUP or CMD_PROP_RADIO_DIV_SETUP is executing, trim values are read from FCFG1 unless they have been provided elsewhere (for more details, see Section 23.3.3.1.2).

On start, the radio CPU sets up parameters for the proprietary mode with parameters given in Table 23-139. The modulation.modType parameter selects between GFSK and unshaped FSK. For FSK and GFSK, modulation.deviation gives the deviation in 250-Hz steps. The radio CPU uses this parameter to calculate a proper shape for use in TX.

The symbol rate is programmed with symbolRate. The parameters are passed directly to the modem and may be calculated using an external tool. The symbol rate is given by Equation 16.

$$f_{baud} = (R \times f_{clk}) / (p \times 2^{20}) \quad (16)$$

where

- f_{baud} is the obtained baud rate
- f_{clk} is the system clock frequency of 24 MHz
- R is the rate word given by symbolRate.rateWord
- p is the prescaler value, given by symbolRate.preScale, which can be from 4 to 15

The rxBw parameter gives the receiver bandwidth. Values from 32 to 52 give the supported bandwidths with the recommended settings. Values from 1 to 18 give the same bandwidths as settings from 35 to 52, for the CC26x0 and CC13x0 devices. Table 23-147 summarizes the values supported and corresponding settings are summarized in . These signals are also in calculation of other register settings.

Table 23-147. Receiver Bandwidth Settings

Setting CC26x0 and CC13x0	Setting Only CC13x0	Receiver Bandwidth (868 MHz)	Receiver Bandwidth (915 MHz)	Receiver Bandwidth (2432 MHz)	Default Intermediate Frequency
–	32	38.9 kHz	41.0 kHz	43.5 kHz (CC1350)	250 kHz
–	33	49.0 kHz	51.6 kHz	54.9 kHz (CC1350)	250 kHz
–	34	58.9 kHz	62.1 kHz	66.0 kHz (CC1350)	250 kHz
1	35	77.7 kHz	81.9 kHz	87.1 kHz	250 kHz
2	36	98.0 kHz	103.3 kHz	109.8 kHz	250 kHz
3	37	117.7 kHz	124.1 kHz	131.9 kHz	250 kHz
4	38	155.4 kHz	163.8 kHz	174.2 kHz	500 kHz
5	39	195.9 kHz	206.5 kHz	219.6 kHz	500 kHz
6	40	235.5 kHz	248.2 kHz	263.9 kHz	500 kHz
7	41	310.8 kHz	327.6 kHz	348.3 kHz	1 MHz
8	42	391.8 kHz	413.0 kHz	439.1 kHz	1 MHz
9	43	470.9 kHz	496.4 kHz	527.8 kHz	1 MHz
10	44	621.6 kHz	655.3 kHz	696.7 kHz	1 MHz
11	45	783.6 kHz	826.0 kHz	878.2 kHz	1 MHz

Table 23-147. Receiver Bandwidth Settings (continued)

Setting CC26x0 and CC13x0	Setting Only CC13x0	Receiver Bandwidth (868 MHz)	Receiver Bandwidth (915 MHz)	Receiver Bandwidth (2432 MHz)	Default Intermediate Frequency
12	46	941.8 kHz	992.8 kHz	1055.6 kHz	1 MHz
13	47	1243.2 kHz	1310.5 kHz	1393.3 kHz	1 MHz
14	48	1567.2 kHz	1652.1 kHz	1756.4 kHz	1 MHz
15	49	1883.7 kHz	1985.7 kHz	2111.1 kHz	1 MHz
16	50	2486.5 kHz	2621.1 kHz	2786.7 kHz	1 MHz
17	51	3134.4 kHz	3304.2 kHz	3512.9 kHz	1 MHz
18	52	3767.4 kHz	3971.4 kHz	4222.2 kHz	1 MHz
Others		Reserved			

The CMD_PROP_RADIO_DIV_SETUP command contains settings for frequency band and intermediate frequency. The center frequency of the band to use is given by centerFreq, and is used to calculate the transmitter shaping filter and the TX IF. The divider to use in the synthesizer is given by loDivider. The user must ensure that the setting is compatible with the given frequency. A value of 2 is allowed only for devices supporting operation in the 2.4-GHz band. In the CMD_PROP_RADIO_SETUP command, centerFreq defaults to 2432 MHz and loDivider defaults to 2.

For CMD_PROP_RADIO_DIV_SETUP, the intermediate frequency can be specified through the intFreq parameter, which calculates the setting in the modem for RX and is written to the configuration parameter area. If this parameter is 0x8000 and for CMD_PROP_RADIO_SETUP, a default intermediate frequency as given in Table 23-147 is used.

The preamConf setting gives the preamble. The preamble is a sequence of 1010... or 0101..., where preamConf.preamMode gives the first transmitted bit. For more than 16 bytes, only an even number of bytes is supported. Setting preamConf.nPreamBytes = 31 gives a 4-bit preamble.

The formatConf setting is used for various setup of the packet format. The sync word length is given by nSwBits, which can be up to 32 bits. The bit polarity for FSK type modulation is given by bBitReversal, which must be 1 for compatibility with CC1101. The bit ordering is given by bMsbFirst, where 1 gives compatibility with the CC1101 device, and so forth. The whitenMode setting can select a whitener scheme. Other whiteners are obtained using override settings. Details of the IEEE 802.15.4g settings are given in Section 23.7.5.2.1. The fecMode setting can be used to change the encoding of the transmitted or received signal. For long-range mode (fecMode = 8), the nSwBits setting and the sync word programmed in the RX and TX commands are ignored, and a hard-coded 64-bit sync word with good performance is used. Setting fecMode to 10 enables Manchester coding. Only encoding and decoding of the payload and CRC is supported. A 0 will be encoded as 01b and a 1 as 10b. More information about Manchester coding can be found in the Proprietary RF user's guide in the CC13x0SDK.

The command sets up a 16-bit CRC with the polynomial $x^{16} + x^{15} + x^2 + 1$ and initialization of all 1s. This is compatible with the CC1101 device. Other polynomials, lengths, and initializations can be obtained by parameter overrides.

The txPower parameter is used to set the output power. For CC13x0, in order to set maximum output power (+14 dBm), changes must also be made to the CCFG area. In the ccfg.c distributed through cc13xxware by TI, set CCFG_FORCE_VDDR_HH to 1. Essentially this will increase the VDDR level, making it possible to use +14 dBm output power. However, setting CCFG_FORCE_VDDR_HH to 1 also increases the overall power consumption. For all output power settings other than +14 dBm, TI recommendssetting CCFG_FORCE_VDDR_HH to 0 (default in ccfg.c distributed by TI), to achieve the lowest possible average power consumption.

The pRegOverride parameter gives a pointer to an override structure, just as the one given for CMD_RADIO_SETUP. This parameter can be used to override parameters calculated from the other settings in the commands, as well as from other parameters. If the value is NULL, no overrides are used.

23.7.5.2.1 IEEE 802.15.4g Packet Format (CC13x0 Only)

IEEE 802.15.4g PHY, including header, is supported by using the CMD_PROP_RX_ADV and CMD_PROP_TX_ADV commands.

The radio is configured to IEEE 802.15.4g mode by setting the formatConf.whitenMode field to the values 4, 5, 6, or 7, and formatConf.bMsbFirst must be set to 1 using the CMD_PROP_RADIO_DIV_SETUP command. For the CMD_PROP_TX_ADV and CMD_PROP_RX_ADV commands, pktConf.bCrcIncSw and pktConf.bCrcIncHdr must both be set to 0. For CMD_PROP_RX_ADV, hdrConf.numHdrBits must be set to 16, hdrConf.lenPos must be set to 0, hdrConf.numLenBits must be set to 11, and lenOffset must be −4.

When formatConf.whitenMode is 5 or 7, the radio is configured to produce the 32-bit CRC and whitening defined in IEEE 802.15.4g. When formatConf.whitenMode is 6 or 7, the radio also processes the headers in both receive and transmit as follows:

- If bit 15 of the header (counted from the LSB) is 1, the frame is assumed to consist of only a header, with no payload or CRC.
- If bit 12 of the header (counted from the LSB) is 1, the 16-bit CRC defined in IEEE 802.15.4g is assumed instead of the 32-bit CRC. For TX, 2 is added to the length offset to account for this, assuming the CRC is included in the received frame length.
- For mode 7: If bit 11 of the header (counted from the LSB) is 1, whitening is enabled; otherwise it is disabled.

NOTE: For modes 6 and 7, the transmitter adjusts CRC and whitening automatically based on transmitted PHY header. However, for this feature to work properly, extended preamble must be used (that is, CMD_PROP_TX_ADV.preTrigger.triggerType cannot be set to TRIG_NOW). As a workaround, set preTrigger.triggerType to TRIG_REL_START, preTrigger.pastTrig to 1 and preTime to 0. This will give normal preamble as configured.

NOTE: The IEEE 802.15.4g PHY header must be presented MSB first to the RF Core. In IEEE 802.15.4g specification, the payload part is LSB first, however the payload length info in physical layer header (PHR) is MSB first. This means that the payload must be flipped in the CM-3. This can be achieved with the Cortex-M3 assembly instruction RBIT.

The following example shows how to send a CRC-32 IEEE 802.15.4g frame with whitening enabled using the automatic headers processing feature (formatConf.whitenMode = 7).

```
/*
 * Prepare the .15.4g PHY header
 * MS=0, Length MSBits=0, DW and CRC settings read from 15.4g header (PHDR) by
RF core.
 * Total length = transmit_len (payload) + CRC length
 *
 * The Radio will flip the bits around, so tx_buf[0] must have the
 * length LSBs (PHR[15:8] and tx_buf[1] will have PHR[7:0]
 */

/* Length in .15.4g PHY HDR includes the CRC but not the HDR itself */
uint16_t total_length;
total_length = transmit_len + CRC_LEN; /* CRC_LEN is 2 for CRC-
16 and 4 for CRC-32 */
tx_buf[0] = total_length & 0xFF;
tx_buf[1] = (total_length >> 8) + 0x08 + 0x0; /* Whitening and CRC-32 bits */
tx_buf[2] = data;
```

NOTE: When IEEE 802.15.4g mode is configured (CMD_PROP_RADIO_SETUP with formatConf.whitenMode = 4, 5, 6, or 7), transmitting packets using unlimited length (pktLen = 0, pPkt pointing to a TX queue) and 32-bit CRC is not supported.

NOTE: To ensure correct crc-16 calculation when radio is configured for IEEE 802.15.4g, two overrides are needed: (uint32_t)0x943, (uint32_t)0x963, these overrides must be added to the override array.

An MCE patch is necessary to support FEC, mode switch, or other advanced features of IEEE 802.15.4g PHY.

23.7.5.3 Transmitter Commands

There are two commands for sending packets, CMD_PROP_TX and CMD_PROP_TX_ADV. The latter gives more flexibility in how the packet can be formed. Details of this are described in Section 23.7.5.3.1 and Section 23.7.5.3.2, respectively.

Both commands require the radio is set up in a compatible mode (such as proprietary mode), and that the synthesizer is programmed using CMD_FS.

For both commands, after the packet has been transmitted, the frequency synthesizer is turned off when the command ends if pktConf.bFsOff is 1. If pktConf.bFsOff is 0, the synthesizer keeps running, so that the command must either be followed by one of the following:

- An RX or TX command (which operate on the same frequency)
- A CMD_FS_OFF command to turn off the synthesizer

or

Table 23-148 lists the end statuses for use with CMD_PROP_TX and CMD_PROP_TX_ADV. This status decides the next operatio (see Section 23.7.5.1).

Table 23-148. End of Radio CMD_PROP_TX and CMD_PROP_TX_ADV Commands

Condition	Status Code	Result
Transmitted packet	PROP_DONE_OK	TRUE
Received CMD_STOP while transmitting packet and finished transmitting packet.	PROP_DONE_STOPPED	FALSE
Received CMD_ABORT while transmitting packet.	PROP_DONE_ABORT	ABORT
Observed illegal parameter.	PROP_ERROR_PAR	ABORT
Command sent without setting up the radio in a supported mode using CMD_PROP_RADIO_SETUP or CMD_RADIO_SETUP.	PROP_ERROR_NO_SETUP	ABORT
Command sent without the synthesizer being programmed.	PROP_ERROR_NO_FS	ABORT
TX underflow observed during operation.	PROP_ERROR_TXUNF	ABORT

23.7.5.3.1 Standard Transmit Command, CMD_PROP_TX

The CMD_PROP_TX command transmits a packet with the format from Table 23-134. The parameters are as given in Table 23-132.

The packet transmission starts at the given start trigger, with a fixed delay. The modem first transmits the preamble and sync word as configured. The sync word to transmit is given in the syncWord field, in the LSBs if less than 32 bits are used. The word is transmitted in the bit order programmed in the radio.

If pktConf.bVarLen is 1, a length byte equal to the value of pktLen is sent next. After this, the content of the buffer pointed to by pPkt is sent. This buffer consists of the number of bytes given in pktLen. If an address byte as shown in Figure 23-9 is needed, it must be sent as the first payload byte.

If pktConf.bUseCrc is 1, a CRC is calculated and transmitted at the end. The number of CRC bits, polynomial, and initialization are as configured in the radio. The CRC is calculated over the length byte (if present) and over the entire contents of the buffer pointed to by pPkt.

If whitening is enabled, the optional length byte, the entire contents of the buffer pointed to by pPkt, and the CRC are subject to whitening. The whitening is done after the data has been used for CRC calculation.

23.7.5.3.2 Advanced Transmit Command, CMD_PROP_TX_ADV

The CMD_PROP_TX_ADV command transmits a packet with the format from Figure 23-10. As a special case, the user can set up packets as outlined in Figure 23-9. The radio must be set up in a compatible mode (such as proprietary mode) and the synthesizer programmed using CMD_FS. The parameters are as given in Table 23-137.

The packet transmission starts at the given start trigger, with a fixed delay. Alternatively, if startConf.bExtTXTrig is 1, the packet transmission starts on an external trigger to the RF core. The trigger is identified as one of the inputs to the RAT, and can be configured as rising edge, falling edge, or both edges through the startConf parameter. The system must ensure that this trigger comes after the start trigger, otherwise it is lost. The minimum delay after the start trigger is implementation-dependent and subject to characterization.

The modem first transmits the preamble and sync word as configured. If preTrigger is not TRIG_NOW, the configured preamble is repeated until that trigger (seen in combination with preTime) has been observed. After the trigger is observed, the configured preamble under transmission finishes before the sync word transmission starts. If preTrigger is TRIG_NOW, the preamble is sent once, followed by the sync word. The sync word to transmit is given in the syncWord field, in the LSBs if less than 32 bits are used, and is transmitted in the bit order programmed in the radio.

If numHdrBits is greater than 0, a header of numHdrBits is sent next. The header may contain a length field or an address. If so, these fields must be inserted correctly in the packet buffer. The header to be transmitted is the first bytes of the buffer pointed to by pPkt. If numHdrBits does not divide by 8, the MSBs of the last byte of the header are ignored.

The header is transmitted as one field in the bit ordering programmed in the radio. If the header has more than 8 bits, it is always read from the transmit buffer in little-endian byte order. If the radio is configured to transmit the MSB first, the last header byte from the TX buffer is transmitted first.

After the header, the remaining bytes in the buffer pointed to by pPkt are transmitted. The payload is transmitted byte by byte, so after the header, no swapping of bytes occurs regardless of bit ordering over the air. The total number of bytes (including the header) in this buffer is given by pktLen. If this length is too small to fit the header, the operation ends with PROP_ERROR_PAR as status. If an address field after the header as shown in Figure 23-10 is needed, it must be sent as the first payload byte.

If pktLen is 0, unlimited length is used. In this case, pPkt points to a transmit queue instead of a buffer (see Section 23.5.3.2).

If pktConf.bUseCrc is 1, a CRC is calculated and transmitted at the end. The number of CRC bits, polynomial, and initialization are as configured in the radio. If pktConf.bCrcIncSw is 1, the transmitted sync word is included in the data set over which the CRC is calculated. If pktConf.bCrcIncHdr is 1, the transmitted header is included in the data set over which the CRC is calculated. The payload is always used to calculate the CRC.

If whitening is enabled, the optional header is subject to whitening if pktConf.bCrcIncHdr is 1. The entire payload and the CRC are always subject to whitening when enabled. The whitening is done after the data has been used for CRC calculation.

23.7.5.4 Receiver Commands

There are two commands for receiving packets, CMD_PROP_RX and CMD_PROP_RX_ADV. The latter gives more flexibility in how the packet can be formed. Details are described in Section 23.7.5.4.1 and Section 23.7.5.4.2, respectively.

For both commands, the radio must be set up in a compatible mode (such as proprietary mode), and the synthesizer must be programmed using CMD_FS before the command is sent to the radio core.

Both commands have an end trigger, given by endTrigger and endTime. If this trigger occurs while the receiver is searching for sync, the operation ends with the status PROP_DONE_RXTIMEOUT. If the trigger occurs while receiving a packet, the action depends on pktConf.endType.
If pktConf.endType = 0, the packet is received to the end and the operation then ends with PROP_DONE_ENDED as the status. If pktConf.endType = 1, the packet reception is aborted and the operation ends with PROP_DONE_BREAK as the status. The radio receives packets according to the details given in Section 23.7.5.4.1 and Section 23.7.5.4.2. After receiving a packet, an interrupt is raised. If pOutput is not NULL, an output structure as given in Table 23-137, pointed to by pOutput, is updated as well. The interrupt to raise and field to update is given in Table 23-149. This table also gives the result to write in the status field of the receive buffer, if enabled. The condition for packets being ignored is described in Section 23.7.5.4.1 and Section 23.7.5.4.2.

Table 23-149. Interrupt, Counter, and Result Field for Received Packets(1)

Condition	Interrupt Raised	Counter Incremented	Result Field of Status Byte
Packet fully received with CRC OK and not to be ignored.	RX_OK	nRxOk	0
Packet fully received with CRC error.	RX_NOK	nRxNok	1
Packet fully received with CRC OK and address mismatch (pktConf.filterOp = 1).	RX_IGNORED	nRxIgnored	2
Packet reception aborted due to timeout (pktConf.endType = 1), CMD_ABORT, too short length in CMD_PROP_SET_LEN, or CMD_PROP_RESTART_RX.	RX_ABORTED	nRxStopped	3(1)
Packet reception aborted due to illegal length or address mismatch (pktConf.filterOp = 0).	RX_ABORTED	nRxStopped	–
Packet could not be stored due to lack of buffer space.	RX_BUF_FULL	nRxBufFull	3(1)

(1) Provided partial-read entry is used and data has been written to the buffer.

For both types of commands, the packet length may be configured as unlimited or unknown at the start of packet reception, by setting maxPktLen to 0. This mode can only be used with partial-read RX buffers. If the length is later determined, it can be set by the immediate or direct command CMD_PROP_SET_LEN, where the number of bytes between the header (if any) and the CRC is given. In addition to setting the length this way, packet reception may be stopped in the following ways (CRC is not performed in the following cases):

- If CMD_PROP_SET_LEN is called with a smaller number of bytes than already received
- If CMD_PROP_RESTART_RX is given
- If no more RX buffer is available
- If the end trigger occurs and pktConf.endType is 1
- If the command is aborted with CMD_ABORT

For ignored packets and packets with CRC error, automatic flush of the RX buffer can be configured. In this case, packets are removed from the receive buffer after they have been received, so the next packet overwrites it and the counters are not updated to reflect the packet received.

NOTE: Automatic flush is not supported for partial-read RX entries. Packets with CRC error (that is, for which the RX_NOK interrupt is raised) are automatically flushed if rxConf.bAutoFlushCrcErr is 1.

Ignored packets (that is, for which the RX_IGNORED interrupt is raised) are automatically flushed if rxConf.bAutoFlushIgnored is 1. After a packet has been received, the next action depends on pktConf.bRepeat. If this is 0, the command ends. Otherwise, it goes back into RX, unless another criterion exists that leads to the command to end. When the command ends, the frequency synthesizer is turned off if pktConf.bFsOff is 1. If pktConf.bFsOff is 0, the synthesizer keeps running, so that the command must be followed by one of the following:

- An RX or TX command (which operate on the same frequency)
- A CMD_FS_OFF command to turn off the synthesizer

Table 23-150 lists the end statuses for CMD_PROP_RX and CMD_PROP_RX_ADV. This status decides the next operation (see Section 23.7.5.1).

Table 23-150. End of Radio CMD_PROP_RX and CMD_PROP_RX_ADV Commands

Condition	Status Code	Result
Received packet with CRC OK and pktConf.bRepeatOk = 0.	PROP_DONE_OK	TRUE
Received packet with CRC error and pktConf.bRepeatNok = 0.	PROP_DONE_RXERR	FALSE
Observed end trigger while in sync search.	PROP_DONE_RXTIMEOUT	FALSE
Observed end trigger while receiving packet with pktConf.endType = 1.	PROP_DONE_BREAK	FALSE
Received packet after having observed end trigger while receiving packet with pktConf.endType = 0.	PROP_DONE_ENDED	FALSE
Received CMD_STOP after command started and, if sync found, packet is received.	PROP_DONE_STOPPED	FALSE
Received CMD_ABORT after command started.	PROP_DONE_ABORT	ABORT
No RX buffer large enough for the received data available at the start of a packet.	PROP_ERROR_RXBUF	FALSE
Out of RX buffer during reception in a partial read buffer.	PROP_ERROR_RXFULL	FALSE
Observed illegal parameter.	PROP_ERROR_PAR	ABORT
Command sent without setting up the radio in a supported mode using CMD_PROP_RADIO_SETUP or CMD_RADIO_SETUP.	PROP_ERROR_NO_SETUP	ABORT
Command sent without the synthesizer being programmed.	PROP_ERROR_NO_FS	ABORT
TX overflow observed during operation.	PROP_ERROR_RXOVF	ABORT

23.7.5.4.1 Standard Receive Command, CMD_PROP_RX

The CMD_PROP_RX receives packets with the format from Figure 23-9. The parameters are as given in Table 23-138.

The modem configures the receiver and starts listening for sync. The sync word to listen for is given in the LSBs of the syncWord field if less than 32 bits are used. The word is in the bit order programmed in the radio.

If sync is found, the radio CPU starts receiving data. If pktConf.bVarLen is 1 and maxPktLen is nonzero, a length byte is assumed as the next byte. This length byte is compared to maxPktLen, and if it is greater, reception is stopped and synch search is restarted. Otherwise, this indicates the number of bytes after the length byte and before the CRC. If pktConf.bVarLen is 0, the length is fixed, and the receiver assumes maxPktLen bytes after the sync word and before the CRC. If maxPktLen is 0, the length is unlimited as described in the beginning of Section 23.7.5.4.

If pktConf.bChkAddress is 1, an address byte is checked next. The address byte is checked against the values of address0 and address1. If only one address is needed, these two fields must be set to the same value. If address1 is 0xFF, it is also checked against the value 0x00. To check for 0xFF without checking for 0x00, address0 must be set to 0xFF. If the address byte does not match the configured addresses, the further treatment depends on pktConf.filterOp. If pktConf.filterOp = 0, reception is stopped and sync search is restarted. If pktConf.filterOp = 1, the packet is received as if the address had matched, but it is marked as ignored.

If the packet is being received, the data is placed in the RX buffer, as shown in Section 23.5.3.1. This RX buffer is found from the receive queue pointed to by pQueue. If pQueue is NULL, the packet is never stored.

If pktConf.bUseCrc is 1, a CRC is received and checked at the end. The number of CRC bits, polynomial, and initialization are as configured in the radio. The CRC is calculated over the length byte (if present), the optional address, and the payload. If pktConf.bUseCrc is 0, the treatment is the same as for CRC OK.

If whitening is enabled, the optional length byte, the payload (including the optional address), and the received CRC are subject to dewhitening. The dewhitening is done before the CRC is evaluated.

If a status byte is appended (rxConf.bAppendStatus is 1) to the packet, it is formatted as follows (see Table 23-144). If pktConf.addressMode is nonzero, the addressInd field is 0 if the address matched address0, 1 if it matched address1, 2 if it matched 0x00 and this address was enabled, and 3 if it matched 0xFF and this address was enabled. Otherwise, addressInd is 0. The syncWordId field is always 0 for CMD_PROP_RX. The result field is written according to Table 23-150.

23.7.5.4.2 Advanced Receive Command, CMD_PROP_RX_ADV

The command CMD_PROP_RX_ADV is used to receive packets with the format from Figure 23-10. As a special case, the user can set up packets as in Figure 23-9. The parameters are as given in Table 23-139.

The modem configures the receiver and listens for sync. The sync word to listen for is given in the LSBs of the syncWord0 field if less than 32 bits are used. The word is in the bit order programmed in the radio. If syncWord1 is nonzero, the receiver also listens for the sync word given in the syncWord1 field (formatted in the same way) if supported in the MCE.

If sync is found, the radio CPU starts receiving data. The packet may contain a header, which can consist of any number of bits up to 32, given by hdrConf.numHdrBits. If the number of bits in the header does not divide by 8, it is considered to consist of a sufficient number of bytes to contain all the stored bits, as shown in Section 23.5.3.1. This header may contain a length field or an address.

The received packet may have fixed or variable length. If hdrConf.numLenBits is 0 and maxPktLen is nonzero, the packet has a fixed length, consisting of maxPktLen bytes after the header and before the CRC. If hdrConf.numLenBits is greater than 0, a field of hdrConf.numLenBits, read from bit number hdrConf.lenPos from the LSB of the header, is taken as a length field. The signed number lenOffset is added to the received length to give the number of bytes after the header and before the CRC. If this number is less than or equal to maxPktLen, the packet is received. If maxPktLen is 0, the length is unlimited as described in the beginning of Section 23.7.5.4. The definitions of packet length for CMD_PROP_RX_ADV and CMD_PROP_TX_ADV differ; see Section 23.7.5.4.2 where the header is included in the packet length.

Two kinds of addresses are supported. With the first option, the address is part of the header. In this case, the address size can be from 1 to 31 bits. The other option is to have an address after the header. If so, this address consists of from 1 to 8 bytes. To use an address as part of the header, addrConf.addrType must be set to 1. The number of bits in the address is given by addrConf.addrSize. These bits are read from bit number addrConf.addrPos from the first bit of the header. To use an address after the header, addrConf.addrType must be set to 0. In this case, the number of bytes in the address is given by addrConf.addrSize.

The received address is compared to an address list pointed to by pAddr. The address to compare against this list is as received. In addition, 1 bit identifying the sync word is concatenated with the address as the MSBs, if one of the following conditions is met:

- syncWord1 ≠ 0 and addrConf.addrType = 1
- syncWord1 ≠ 0, addrConf.addrType = 0, and addrConf.addrPos ≠ 0

This extra bit is 0 if the received sync word was syncWord0, and the extra bit is 1 if the received sync word was syncWord1. The entries in the address list have a size of 8, 16, 32, or 64 bits; the size in use is the smallest size that can fit the address size, including the sync word identification bit if applicable. The number of entries in the address list is given by addrConf.numAddr. The radio CPU scans through the addresses in the address list and compares it to the received address. If there is no match, the further treatment depends on pktConf.filterOp. If pktConf.filterOp is 0, reception is stopped and synch search is restarted. If pktConf.filterOp is 1, the packet is received as if the address had matched, but marked as ignored.

If addrConf.addrSize is 0, no address is used. In this case, pAddr is ignored and must be NULL.

If the packet is being received, the data is placed in the RX buffer, as in Section 23.5.3.1. This RX buffer is found from the receive queue pointed to by pQueue. If pQueue is NULL, the packet is never stored.

The header is received as one field in the bit ordering programmed in the radio. If the header has more than 8 bits and rxConf.bIncludeHdr is 1, the header is always written in little-endian byte order to the RX buffer. If the radio is configured to receive the MSB first, the last header byte stored in the RX buffer is received first. The payload is stored byte by byte, so after the header, no swapping of bytes occurs regardless of bit ordering over the air.

If pktConf.bUseCrc is 1, a CRC is received and checked at the end. The number of CRC bits, polynomial, and initialization are as configured in the radio. If pktConf.bCrcIncSw is 1, the received sync word (assuming it to be exactly equal to syncWord0 or syncWord1) is included in the data set over which the CRC is calculated. If pktConf.bCrcIncHdr is 1, the received header is included in the data set over which the CRC is calculated. The payload, including the optional address after the header, is always used for calculating the CRC. If pktConf.bUseCrc is 0, the treatment is the same as for CRC OK.

If whitening is enabled, the optional header is subject to dewhitening only if pktConf.bCrcIncHdr is 1. The payload (including the optional address after the header), and the received CRC are always subject to dewhitening when enabled. The dewhitening is done before the CRC is evaluated.

If a status byte is appended (rxConf.bAppendStatus is 1) to the packet, it is formatted as detailed in Table 23-144. If addrConf.addrSize is nonzero, the addressInd field is the first index into the address list that matched the received address if an address match existed. Otherwise, addressInd is 0. The syncWordId field is 0 if the received sync word was syncWord0, and 1 if syncWord1. The result field is written according to Table 23-149.

23.7.5.5 Carrier-Sense Operation (CC13x0 Only)

The carrier-sense operation detects if a signal is present, which has the following main purposes:

- Turns off the radio instead of receiving when no signal is present
- Turns the radio to transmit only if no signal is present

The carrier-sense operation can be used with the command CMD_PROP_CS to chain with another operation (for example, a transmit operation), or with the commands CMD_PROP_RX_SNIFF or CMD_PROP_RX_ADV_SNIFF to combine with a normal receive operation to implement sniff mode. The details of these commands are described in the following subsections.

23.7.5.5.1 Common Carrier-Sense Description

The parameters for the carrier-sense operation are common for all the commands, and are given in Section 23.7.2.3. Table 23-142 gives the offset from the first byte used for carrier-sense parameters.

The channel can be in one of three states: BUSY, IDLE, or INVALID. BUSY indicates a signal on the channel. IDLE indicates no signal is present on the channel. INVALID indicates that the state cannot be determined. There are two sources of channel information, RSSI and correlation, and a separate state is maintained for each source.

The operation starts when the radio is set up in receive mode. The RSSI or correlation is monitored, according to the enable bits csConf.bEnaRssi and csConf.bEnaCorr. If csConf.bEnaRssi is 1, the RSSI is monitored. If csConf.bEnaCorr is 1, the correlator is set up to correlate against the preamble. It is not possible to set both enable bits to 0.

If csConf.bEnaRssi is 1, the RSSI is monitored every time a new value is available from the radio. At each update, the RSSI is compared against the signed value rssiThr. If the RSSI is below rssiThr and if numRssiIdle consecutive RSSI measurements below the threshold have been observed, the RSSI state is IDLE. If the RSSI is above rssiThr and if numRssiBusy consecutive RSSI measurements above the threshold have been observed, the RSSI state is BUSY. Otherwise, the RSSI state is INVALID.

If csConf.bEnaCorr is 1, the radio CPU monitors correlation peaks from the modem. When the radio starts, the state is INVALID. If no correlation top is observed until corrPeriod RAT ticks after the carrier-sense command was started, the state becomes IDLE. If the state is IDLE and at least corrConfig.numCorrInv correlation tops with a maximum of corrPeriod RAT ticks between them are observed, the state becomes INVALID. If the state is INVALID and at least corrConfig.numCorrBusy correlation tops with at most corrPeriod RAT ticks between them are observed, the state becomes BUSY. If corrConfig.numCorrBusy is 0, the state goes directly to BUSY from IDLE. The value of corrConfig.numCorrIdle must be greater than 0. If the state is not IDLE and corrTime RAT ticks pass after the last correlation top, the state becomes IDLE again.

If only 1 of the enable bits is 1, the channel state is equal to the state of the corresponding source. If both enable bits are 1, the channel state depends on the state of the two sources and the csConf.operation bit, as shown in Table 23-151.

Table 23-151. Channel State When Both Sources are Enabled

csConf.operation = 0			
RSSI state	Correlation state		
	INVALID	IDLE	BUSY
INVALID	INVALID	INVALID	BUSY
IDLE	INVALID	IDLE	BUSY
BUSY	BUSY	BUSY	BUSY
csConf.operation = 1			
RSSI state	Correlation state		
	INVALID	IDLE	BUSY
INVALID	INVALID	IDLE	INVALID
IDLE	IDLE	IDLE	IDLE
BUSY	INVALID	IDLE	BUSY

If the state of the channel changes to BUSY, the action depends on csConf.busyOp and the command being run. If csConf.busyOp is 0, the operation continues. If csConf.busyOp is 1 and the command is CMD_PROP_CS, the operation ends with PROP_DONE_BUSY as the status. If csConf.busyOp is 1 and the command is CMD_PROP_RX_SNIFF or CMD_PROP_RX_ADV_SNIFF, the receive operation continues, but carrier sense is stopped, so the operation is not affected if the channel state later changes to IDLE.

If the state of the channel changes to IDLE, the action depends on csConf.idleOp. If the value of this field is 0, the receiver and carrier-sense operation continues. If the value of the bit field is 1, the operation ends with PROP_DONE_IDLE as status.

If the trigger given by csEndTrigger and csEndTime is observed, the action depends on the command being run and the channel state at that time. The details are described in Section 23.7.5.5.2 and Section 23.7.5.5.3.

23.7.5.5.2 Carrier-Sense Command, CMD_PROP_CS

When the carrier-sense command starts, the radio is set up in receive mode, and the operations described in Section 23.7.5.5.1 are performed. The radio must be set up in a compatible mode (such as proprietary mode) and the synthesizer programmed using CMD_FS.

If the trigger given by csEndTrigger and csEndTime is observed, the operation ends, and the current channel state is checked. If the state is BUSY or IDLE, the status is PROP_DONE_BUSY or PROP_DONE_IDLE, respectively. If the state is INVALID, the status depends on csConf.timeoutRes. If 0, the status is PROP_DONE_BUSYTIMEOUT; if 1, the status is PROP_DONE_IDLETIMEOUT.

When the command CMD_PROP_CS ends and the status is PROP_DONE_BUSY or PROP_DONE_BUSYTIMEOUT, the synthesizer is turned off if csFsConf.bFsOffBusy is 1. If the command ends and the status is PROP_DONE_IDLE or PROP_DONE_IDLETIMEOUT, the synthesizer is turned off if csFsConf.bFsOffIdle is 1. If the command ends with another status, the synthesizer is turned off if either of these bits is 1.

The end statuses for use with CMD_PROP_CS are summarized in Table 23-152. This status decides the next operation, as shown in Section 23.7.5.1.

Table 23-152. End of CMD_PROP_CS Command

Condition	Status Code	Result
Observed channel state BUSY with csConf.busyOp = 1.	PROP_DONE_BUSY	TRUE
Observed channel state IDLE with csConf.idleOp = 1.	PROP_DONE_IDLE	FALSE
Time-out trigger observed with channel state BUSY.	PROP_DONE_BUSY	TRUE
Time-out trigger observed with channel state IDLE.	PROP_DONE_IDLE	FALSE
Time-out trigger observed with channel state INVALID and csConf.timeoutRes = 0.	PROP_DONE_BUSYTIMEOUT	TRUE

Table 23-152. End of CMD_PROP_CS Command (continued)

Condition	Status Code	Result
Time-out trigger observed with channel state INVALID and csConf.timeoutRes = 1.	PROP_DONE_IDLETIMEOUT	FALSE
Received CMD_STOP after command started.	PROP_DONE_STOPPED	FALSE
Received CMD_ABORT after command started.	PROP_DONE_ABORT	ABORT
Observed illegal parameter.	PROP_ERROR_PAR	ABORT
Command sent without setting up the radio in a supported mode using CMD_PROP_RADIO_SETUP or CMD_RADIO_SETUP.	PROP_ERROR_NO_SETUP	ABORT
Command sent without the synthesizer being programmed.	PROP_ERROR_NO_FS	ABORT

23.7.5.5.3 Sniff Mode Receiver Commands, CMD_PROP_RX_SNIFF and CMD_PROP_RX_ADV_SNIFF

The commands CMD_PROP_RX_SNIFF and CMD_PROP_RX_ADV_SNIFF behave like the commands CMD_PROP_RX and CMD_PROP_RX_ADV, respectively, but they perform carrier-sense operations during sync search.

When started, the commands perform the carrier-sense operations described in Section 23.7.5.5.1. As described, the operation may end if the channel state becomes IDLE.

If the trigger given by csEndTrigger and csEndTime is observed, the current channel state is checked. If the channel state is BUSY, the receiver continues, but may end later if the channel state becomes IDLE and csConf.busyOp is 0. If the channel state is IDLE, the operation ends (even if csConf.idleOp is 0), and the status is PROP_DONE_IDLE. If the channel state is INVALID, the action depends on csConf.timeoutRes. If csConf.timeoutRes is 0, the receive operation continues, and if csConf.busyOp is 1, carrier sense is no longer checked. If csConf.timeoutRes is 1, the operation ends and the status is PROP_DONE_IDLETIMEOUT.

If sync is found, the receiver operates as described in Section 23.7.5.4. If sync search is restarted after a packet is received or after reception is stopped due to an invalid length field or address mismatch, the carrier-sense operation is resumed if it was running when sync was found.

The end statuses for use with CMD_PROP_RX_SNIFF and CMD_PROP_RX_ADV_SNIFF are listed in Table 23-150 and Table 23-153. This status decides the next operation, as in Section 23.7.5.1.

Table 23-153. Additional End Statuses for CMD_PROP_RX_SNIFF and CMD_PROP_RX_ADV_SNIFF

Condition	Status Code	Result
Observed channel state IDLE with csConf.idleOp = 1.	PROP_DONE_IDLE	FALSE
Time-out trigger observed with channel state IDLE.	PROP_DONE_IDLE	FALSE
Time-out trigger observed with channel state INVALID and csConf.timeoutRes = 1.	PROP_DONE_IDLETIMEOUT	FALSE

23.7.6 Immediate Commands

23.7.6.1 Set Packet Length Command, CMD_PROP_SET_LEN

The CMD_PROP_SET_LEN command takes a command structure as defined in Table 23-140.

CMD_PROP_SET_LEN must only be sent while a CMD_PROP_RX or CMD_PROP_RX_ADV command is running configured with unlimited packet length. When the command is received, the radio CPU sets the number of bytes to receive between the header and the CRC to RXLen. If at least this number of bytes has already been received, reception is aborted, as in Section 23.7.5.4 and Section 23.7.5.4.2.

The command may be sent as a direct command if the payload length to set is 255 bytes or less. In this case, the RXLen parameter is written in bits 8–16 of CMDR, and the 8 MSBs of this parameter is 0.

If the command is issued without a CMD_PROP_RX or CMD_PROP_RX_ADV command running, or if such a command is not configured with unlimited length, the radio CPU returns the result ContextError in CMDSTA. Otherwise, the radio CPU returns DONE.

23.7.6.2 Restart Packet RX Command, CMD_PROP_RESTART_RX

The CMD_PROP_RESTART_RX command is a direct command that takes no parameters.

CMD_PROP_RESTART_RX must only be sent while a CMD_PROP_RX or CMD_PROP_RX_ADV command is running. If a packet is being received, reception is aborted, as described in Section 23.7.5.4 and the packet returns to sync search.

If the command is issued without an RX command running, the radio CPU returns the result ContextError in CMDSTA. Otherwise, the radio CPU returns DONE.

23.8 Radio Registers

23.8.1 RFC_RAT Registers

Table 23-154 lists the memory-mapped registers for the RFC_RAT. All register offset addresses not listed in Table 23-154 should be considered as reserved locations and the register contents should not be modified.

Table 23-154. RFC_RAT Registers

Offset	Acronym	Register Name	Section
4h	RATCNT	Radio Timer Counter Value	Section 23.8.1.1
80h	RATCH0VAL	Timer Channel 0 Capture/Compare Register	Section 23.8.1.2
84h	RATCH1VAL	Timer Channel 1 Capture/Compare Register	Section 23.8.1.3
88h	RATCH2VAL	Timer Channel 2 Capture/Compare Register	Section 23.8.1.4
8Ch	RATCH3VAL	Timer Channel 3 Capture/Compare Register	Section 23.8.1.5
90h	RATCH4VAL	Timer Channel 4 Capture/Compare Register	Section 23.8.1.6
94h	RATCH5VAL	Timer Channel 5 Capture/Compare Register	Section 23.8.1.7
98h	RATCH6VAL	Timer Channel 6 Capture/Compare Register	Section 23.8.1.8
9Ch	RATCH7VAL	Timer Channel 7 Capture/Compare Register	Section 23.8.1.9

23.8.1.1 RATCNT Register (Offset = 4h) [reset = 0h]

RATCNT is shown in Figure 23-12 and described in Table 23-155.

Return to Summary Table.

Radio Timer Counter Value

Figure 23-12. RATCNT Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNT																															
R/W-0h																															

Table 23-155. RATCNT Register Field Descriptions

Bit	Field	Type	Reset	Description
31-0	CNT	R/W	0h	Counter value. This is not writable while radio timer counter is enabled.

23.8.1.2 RATCH0VAL Register (Offset = 80h) [reset = 0h]

RATCH0VAL is shown in Figure 23-13 and described in Table 23-156.

Return to Summary Table.

Timer Channel 0 Capture/Compare Register

Figure 23-13. RATCH0VAL Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
VAL																															
R/W-0h																															

Table 23-156. RATCH0VAL Register Field Descriptions

Bit	Field	Type	Reset	Description
31-0	VAL	R/W	0h	Capture/compare value. The system CPU can safely read this register, but it is recommended to use the CPE API commands to configure it for compare mode.

23.8.1.3 RATCH1VAL Register (Offset = 84h) [reset = 0h]

RATCH1VAL is shown in Figure 23-14 and described in Table 23-157.

Return to Summary Table.

Timer Channel 1 Capture/Compare Register

Figure 23-14. RATCH1VAL Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
VAL																															
R/W-0h																															

Table 23-157. RATCH1VAL Register Field Descriptions

Bit	Field	Type	Reset	Description
31-0	VAL	R/W	0h	Capture/compare value. The system CPU can safely read this register, but it is recommended to use the CPE API commands to configure it for compare mode.

23.8.1.4 RATCH2VAL Register (Offset = 88h) [reset = 0h]

RATCH2VAL is shown in Figure 23-15 and described in Table 23-158.

Return to Summary Table.

Timer Channel 2 Capture/Compare Register

Figure 23-15. RATCH2VAL Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
VAL																															
R/W-0h																															

Table 23-158. RATCH2VAL Register Field Descriptions

Bit	Field	Type	Reset	Description
31-0	VAL	R/W	0h	Capture/compare value. The system CPU can safely read this register, but it is recommended to use the CPE API commands to configure it for compare mode.

23.8.1.5 RATCH3VAL Register (Offset = 8Ch) [reset = 0h]

RATCH3VAL is shown in Figure 23-16 and described in Table 23-159.

Return to Summary Table.

Timer Channel 3 Capture/Compare Register

Figure 23-16. RATCH3VAL Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
VAL																															
R/W-0h																															

Table 23-159. RATCH3VAL Register Field Descriptions

Bit	Field	Type	Reset	Description
31-0	VAL	R/W	0h	Capture/compare value. The system CPU can safely read this register, but it is recommended to use the CPE API commands to configure it for compare mode.

23.8.1.6 RATCH4VAL Register (Offset = 90h) [reset = 0h]

RATCH4VAL is shown in Figure 23-17 and described in Table 23-160.

Return to Summary Table.

Timer Channel 4 Capture/Compare Register

Figure 23-17. RATCH4VAL Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
VAL																															
R/W-0h																															

Table 23-160. RATCH4VAL Register Field Descriptions

Bit	Field	Type	Reset	Description
31-0	VAL	R/W	0h	Capture/compare value. The system CPU can safely read this register, but it is recommended to use the CPE API commands to configure it for compare mode.

23.8.1.7 RATCH5VAL Register (Offset = 94h) [reset = 0h]

RATCH5VAL is shown in Figure 23-18 and described in Table 23-161.

Return to Summary Table.

Timer Channel 5 Capture/Compare Register

Figure 23-18. RATCH5VAL Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
VAL																															
R/W-0h																															

Table 23-161. RATCH5VAL Register Field Descriptions

Bit	Field	Type	Reset	Description
31-0	VAL	R/W	0h	Capture/compare value. The system CPU can safely read this register, but it is recommended to use the CPE API commands to configure it for compare mode.

23.8.1.8 RATCH6VAL Register (Offset = 98h) [reset = 0h]

RATCH6VAL is shown in Figure 23-19 and described in Table 23-162.

Return to Summary Table.

Timer Channel 6 Capture/Compare Register

Figure 23-19. RATCH6VAL Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
VAL																															
R/W-0h																															

Table 23-162. RATCH6VAL Register Field Descriptions

Bit	Field	Type	Reset	Description
31-0	VAL	R/W	0h	Capture/compare value. The system CPU can safely read this register, but it is recommended to use the CPE API commands to configure it for compare mode.

23.8.1.9 RATCH7VAL Register (Offset = 9Ch) [reset = 0h]

RATCH7VAL is shown in Figure 23-20 and described in Table 23-163.

Return to Summary Table.

Timer Channel 7 Capture/Compare Register

Figure 23-20. RATCH7VAL Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
VAL																															
R/W-0h																															

Table 23-163. RATCH7VAL Register Field Descriptions

Bit	Field	Type	Reset	Description
31-0	VAL	R/W	0h	Capture/compare value. The system CPU can safely read this register, but it is recommended to use the CPE API commands to configure it for compare mode.

23.8.2 RFC_DBELL Registers

Table 23-164 lists the memory-mapped registers for the RFC_DBELL. All register offset addresses not listed in Table 23-164 should be considered as reserved locations and the register contents should not be modified.

Table 23-164. RFC_DBELL Registers

Offset	Acronym	Register Name	Section
0h	CMDR	Doorbell Command Register	Section 23.8.2.1
4h	CMDSTA	Doorbell Command Status Register	Section 23.8.2.2
8h	RFHWIFG	Interrupt Flags From RF Hardware Modules	Section 23.8.2.3
Ch	RFHWIEN	Interrupt Enable For RF Hardware Modules	Section 23.8.2.4
10h	RFCPEIFG	Interrupt Flags For Command and Packet Engine Generated Interrupts	Section 23.8.2.5
14h	RFCPEIEN	Interrupt Enable For Command and Packet Engine Generated Interrupts	Section 23.8.2.6
18h	RFCPEISL	Interrupt Vector Selection For Command and Packet Engine Generated Interrupts	Section 23.8.2.7
1Ch	RFACKIFG	Doorbell Command Acknowledgement Interrupt Flag	Section 23.8.2.8
20h	SYSGPOCTL	RF Core General Purpose Output Control	Section 23.8.2.9

23.8.2.1 CMDR Register (Offset = 0h) [reset = 0h]

CMDR is shown in Figure 23-21 and described in Table 23-165.

Return to Summary Table.

Doorbell Command Register

Figure 23-21. CMDR Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CMD																															
R/W-0h																															

Table 23-165. CMDR Register Field Descriptions

Bit	Field	Type	Reset	Description
31-0	CMD	R/W	0h	Command register. Raises an interrupt to the Command and packet engine (CPE) upon write.

23.8.2.2 CMDSTA Register (Offset = 4h) [reset = 0h]

CMDSTA is shown in Figure 23-22 and described in Table 23-166.

Return to Summary Table.

Doorbell Command Status Register

Figure 23-22. CMDSTA Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
STAT																															
R-0h																															

Table 23-166. CMDSTA Register Field Descriptions

Bit	Field	Type	Reset	Description
31-0	STAT	R	0h	Status of the last command used

23.8.2.3 RFHWIFG Register (Offset = 8h) [reset = 0h]

RFHWIFG is shown in Figure 23-23 and described in Table 23-167.

Return to Summary Table.

Interrupt Flags From RF Hardware Modules

Figure 23-23. RFHWIFG Register

31	30	29	28	27	26	25	24
RESERVED							
R-0h							
23	22	21	20	19	18	17	16
RESERVED				RATCH7	RATCH6	RATCH5	RATCH4
R-0h				R/W-0h	R/W-0h	R/W-0h	R/W-0h
15	14	13	12	11	10	9	8
RATCH3	RATCH2	RATCH1	RATCH0	RFESOFT2	RFESOFT1	RFESOFT0	RFEDONE
R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h
7	6	5	4	3	2	1	0
RESERVED	TRCTK	MDMSOFT	MDMOUT	MDMIN	MDMDONE	FSCA	RESERVED
R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h

Table 23-167. RFHWIFG Register Field Descriptions

Bit	Field	Type	Reset	Description
31-20	RESERVED	R	0h	Software should not rely on the value of a reserved. Writing any other value than the reset value may result in undefined behavior.
19	RATCH7	R/W	0h	Radio timer channel 7 interrupt flag. Write zero to clear flag. Write to one has no effect.
18	RATCH6	R/W	0h	Radio timer channel 6 interrupt flag. Write zero to clear flag. Write to one has no effect.
17	RATCH5	R/W	0h	Radio timer channel 5 interrupt flag. Write zero to clear flag. Write to one has no effect.
16	RATCH4	R/W	0h	Radio timer channel 4 interrupt flag. Write zero to clear flag. Write to one has no effect.
15	RATCH3	R/W	0h	Radio timer channel 3 interrupt flag. Write zero to clear flag. Write to one has no effect.
14	RATCH2	R/W	0h	Radio timer channel 2 interrupt flag. Write zero to clear flag. Write to one has no effect.
13	RATCH1	R/W	0h	Radio timer channel 1 interrupt flag. Write zero to clear flag. Write to one has no effect.
12	RATCH0	R/W	0h	Radio timer channel 0 interrupt flag. Write zero to clear flag. Write to one has no effect.
11	RFESOFT2	R/W	0h	RF engine software defined interrupt 2 flag. Write zero to clear flag. Write to one has no effect.
10	RFESOFT1	R/W	0h	RF engine software defined interrupt 1 flag. Write zero to clear flag. Write to one has no effect.
9	RFESOFT0	R/W	0h	RF engine software defined interrupt 0 flag. Write zero to clear flag. Write to one has no effect.
8	RFEDONE	R/W	0h	RF engine command done interrupt flag. Write zero to clear flag. Write to one has no effect.
7	RESERVED	R/W	0h	Software should not rely on the value of a reserved. Writing any other value than the reset value may result in undefined behavior.
6	TRCTK	R/W	0h	Debug tracer system tick interrupt flag. Write zero to clear flag. Write to one has no effect.

Table 23-167. RFHWIFG Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
5	MDMSOFT	R/W	0h	Modem synchronization word detection interrupt flag. This interrupt will be raised by modem when the synchronization word is received. The CPE may decide to reject the packet based on its header (protocol specific). Write zero to clear flag. Write to one has no effect.
4	MDMOUT	R/W	0h	Modem FIFO output interrupt flag. Write zero to clear flag. Write to one has no effect.
3	MDMIN	R/W	0h	Modem FIFO input interrupt flag. Write zero to clear flag. Write to one has no effect.
2	MDMDONE	R/W	0h	Modem command done interrupt flag. Write zero to clear flag. Write to one has no effect.
1	FSCA	R/W	0h	Frequency synthesizer calibration accelerator interrupt flag. Write zero to clear flag. Write to one has no effect.
0	RESERVED	R/W	0h	Software should not rely on the value of a reserved. Writing any other value than the reset value may result in undefined behavior.

23.8.2.4 RFHWIEN Register (Offset = Ch) [reset = 0h]

RFHWIEN is shown in Figure 23-24 and described in Table 23-168.

Return to Summary Table.

Interrupt Enable For RF Hardware Modules

Figure 23-24. RFHWIEN Register

31	30	29	28	27	26	25	24
RESERVED							
R-0h							
23	22	21	20	19	18	17	16
RESERVED				RATCH7	RATCH6	RATCH5	RATCH4
R-0h				R/W-0h	R/W-0h	R/W-0h	R/W-0h
15	14	13	12	11	10	9	8
RATCH3	RATCH2	RATCH1	RATCH0	RFESOFT2	RFESOFT1	RFESOFT0	RFEDONE
R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h
7	6	5	4	3	2	1	0
RESERVED	TRCTK	MDMSOFT	MDMOUT	MDMIN	MDMDONE	FSCA	RESERVED
R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h

Table 23-168. RFHWIEN Register Field Descriptions

Bit	Field	Type	Reset	Description
31-20	RESERVED	R	0h	Software should not rely on the value of a reserved. Writing any other value than the reset value may result in undefined behavior.
19	RATCH7	R/W	0h	Interrupt enable for RFHWIFG.RATCH7.
18	RATCH6	R/W	0h	Interrupt enable for RFHWIFG.RATCH6.
17	RATCH5	R/W	0h	Interrupt enable for RFHWIFG.RATCH5.
16	RATCH4	R/W	0h	Interrupt enable for RFHWIFG.RATCH4.
15	RATCH3	R/W	0h	Interrupt enable for RFHWIFG.RATCH3.
14	RATCH2	R/W	0h	Interrupt enable for RFHWIFG.RATCH2.
13	RATCH1	R/W	0h	Interrupt enable for RFHWIFG.RATCH1.
12	RATCH0	R/W	0h	Interrupt enable for RFHWIFG.RATCH0.
11	RFESOFT2	R/W	0h	Interrupt enable for RFHWIFG.RFESOFT2.
10	RFESOFT1	R/W	0h	Interrupt enable for RFHWIFG.RFESOFT1.
9	RFESOFT0	R/W	0h	Interrupt enable for RFHWIFG.RFESOFT0.
8	RFEDONE	R/W	0h	Interrupt enable for RFHWIFG.RFEDONE.
7	RESERVED	R/W	0h	Software should not rely on the value of a reserved. Writing any other value than the reset value may result in undefined behavior.
6	TRCTK	R/W	0h	Interrupt enable for RFHWIFG.TRCTK.
5	MDMSOFT	R/W	0h	Interrupt enable for RFHWIFG.MDMSOFT.
4	MDMOUT	R/W	0h	Interrupt enable for RFHWIFG.MDMOUT.
3	MDMIN	R/W	0h	Interrupt enable for RFHWIFG.MDMIN.
2	MDMDONE	R/W	0h	Interrupt enable for RFHWIFG.MDMDONE.
1	FSCA	R/W	0h	Interrupt enable for RFHWIFG.FSCA.
0	RESERVED	R/W	0h	Software should not rely on the value of a reserved. Writing any other value than the reset value may result in undefined behavior.

23.8.2.5 RFCPEIFG Register (Offset = 10h) [reset = 0h]

RFCPEIFG is shown in Figure 23-25 and described in Table 23-169.

Return to Summary Table.

Interrupt Flags For Command and Packet Engine Generated Interrupts

Figure 23-25. RFCPEIFG Register

31	30	29	28	27	26	25	24
INTERNAL_ERROR	BOOT_DONE	MODULES_UNLOCKED	SYNTH_NO_LOCK	IRQ27	RX_ABORTED	RX_N_DATA_WRITTEN	RX_DATA_WRITTEN
R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h
23	22	21	20	19	18	17	16
RX_ENTRY_DONE	RX_BUF_FULL	RX_CTRL_ACK	RX_CTRL	RX_EMPTY	RX_IGNORED	RX_NOK	RX_OK
R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h
15	14	13	12	11	10	9	8
IRQ15	IRQ14	IRQ13	IRQ12	TX_BUFFER_CHANGED	TX_ENTRY_DONE	TX_RETRANS	TX_CTRL_ACK_ACK
R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h
7	6	5	4	3	2	1	0
TX_CTRL_ACK	TX_CTRL	TX_ACK	TX_DONE	LAST_FG_COMMAND_DONE	FG_COMMAND_DONE	LAST_COMMAND_DONE	COMMAND_DONE
R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h

Table 23-169. RFCPEIFG Register Field Descriptions

Bit	Field	Type	Reset	Description
31	INTERNAL_ERROR	R/W	0h	Interrupt flag 31. The command and packet engine (CPE) has observed an unexpected error. A reset of the CPE is needed. This can be done by switching the RF Core power domain off and on in PRCM:PDCTL1RFC. Write zero to clear flag. Write to one has no effect.
30	BOOT_DONE	R/W	0h	Interrupt flag 30. The command and packet engine (CPE) boot is finished. Write zero to clear flag. Write to one has no effect.
29	MODULES_UNLOCKED	R/W	0h	Interrupt flag 29. As part of command and packet engine (CPE) boot process, it has opened access to RF Core modules and memories. Write zero to clear flag. Write to one has no effect.
28	SYNTH_NO_LOCK	R/W	0h	Interrupt flag 28. The phase-locked loop in frequency synthesizer has reported loss of lock. Write zero to clear flag. Write to one has no effect.
27	IRQ27	R/W	0h	Interrupt flag 27. Write zero to clear flag. Write to one has no effect.
26	RX_ABORTED	R/W	0h	Interrupt flag 26. Packet reception stopped before packet was done. Write zero to clear flag. Write to one has no effect.
25	RX_N_DATA_WRITTEN	R/W	0h	Interrupt flag 25. Specified number of bytes written to partial read Rx buffer. Write zero to clear flag. Write to one has no effect.
24	RX_DATA_WRITTEN	R/W	0h	Interrupt flag 24. Data written to partial read Rx buffer. Write zero to clear flag. Write to one has no effect.
23	RX_ENTRY_DONE	R/W	0h	Interrupt flag 23. Rx queue data entry changing state to finished. Write zero to clear flag. Write to one has no effect.
22	RX_BUF_FULL	R/W	0h	Interrupt flag 22. Packet received that did not fit in Rx queue. BLE mode: Packet received that did not fit in the Rx queue. IEEE 802.15.4 mode: Frame received that did not fit in the Rx queue. Write zero to clear flag. Write to one has no effect.
21	RX_CTRL_ACK	R/W	0h	Interrupt flag 21. BLE mode only: LL control packet received with CRC OK, not to be ignored, then acknowledgement sent. Write zero to clear flag. Write to one has no effect.

Table 23-169. RFCPEIFG Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
20	RX_CTRL	R/W	0h	Interrupt flag 20. BLE mode only: LL control packet received with CRC OK, not to be ignored. Write zero to clear flag. Write to one has no effect.
19	RX_EMPTY	R/W	0h	Interrupt flag 19. BLE mode only: Packet received with CRC OK, not to be ignored, no payload. Write zero to clear flag. Write to one has no effect.
18	RX_IGNORED	R/W	0h	Interrupt flag 18. Packet received, but can be ignored. BLE mode: Packet received with CRC OK, but to be ignored. IEEE 802.15.4 mode: Frame received with ignore flag set. Write zero to clear flag. Write to one has no effect.
17	RX_NOK	R/W	0h	Interrupt flag 17. Packet received with CRC error. BLE mode: Packet received with CRC error. IEEE 802.15.4 mode: Frame received with CRC error. Write zero to clear flag. Write to one has no effect.
16	RX_OK	R/W	0h	Interrupt flag 16. Packet received correctly. BLE mode: Packet received with CRC OK, payload, and not to be ignored. IEEE 802.15.4 mode: Frame received with CRC OK. Write zero to clear flag. Write to one has no effect.
15	IRQ15	R/W	0h	Interrupt flag 15. Write zero to clear flag. Write to one has no effect.
14	IRQ14	R/W	0h	Interrupt flag 14. Write zero to clear flag. Write to one has no effect.
13	IRQ13	R/W	0h	Interrupt flag 13. Write zero to clear flag. Write to one has no effect.
12	IRQ12	R/W	0h	Interrupt flag 12. Write zero to clear flag. Write to one has no effect.
11	TX_BUFFER_CHANGED	R/W	0h	Interrupt flag 11. BLE mode only: A buffer change is complete after CMD_BLE_ADV_PAYLOAD. Write zero to clear flag. Write to one has no effect.
10	TX_ENTRY_DONE	R/W	0h	Interrupt flag 10. Tx queue data entry state changed to finished. Write zero to clear flag. Write to one has no effect.
9	TX_RETRANS	R/W	0h	Interrupt flag 9. BLE mode only: Packet retransmitted. Write zero to clear flag. Write to one has no effect.
8	TX_CTRL_ACK_ACK	R/W	0h	Interrupt flag 8. BLE mode only: Acknowledgement received on a transmitted LL control packet, and acknowledgement transmitted for that packet. Write zero to clear flag. Write to one has no effect.
7	TX_CTRL_ACK	R/W	0h	Interrupt flag 7. BLE mode: Acknowledgement received on a transmitted LL control packet. Write zero to clear flag. Write to one has no effect.
6	TX_CTRL	R/W	0h	Interrupt flag 6. BLE mode: Transmitted LL control packet. Write zero to clear flag. Write to one has no effect.
5	TX_ACK	R/W	0h	Interrupt flag 5. BLE mode: Acknowledgement received on a transmitted packet. IEEE 802.15.4 mode: Transmitted automatic ACK frame. Write zero to clear flag. Write to one has no effect.
4	TX_DONE	R/W	0h	Interrupt flag 4. Packet transmitted. (BLE mode: A packet has been transmitted.) (IEEE 802.15.4 mode: A frame has been transmitted). Write zero to clear flag. Write to one has no effect.
3	LAST_FG_COMMAND_DONE	R/W	0h	Interrupt flag 3. IEEE 802.15.4 mode only: The last foreground radio operation command in a chain of commands has finished. Write zero to clear flag. Write to one has no effect.
2	FG_COMMAND_DONE	R/W	0h	Interrupt flag 2. IEEE 802.15.4 mode only: A foreground radio operation command has finished. Write zero to clear flag. Write to one has no effect.
1	LAST_COMMAND_DONE	R/W	0h	Interrupt flag 1. The last radio operation command in a chain of commands has finished. (IEEE 802.15.4 mode: The last background level radio operation command in a chain of commands has finished.) Write zero to clear flag. Write to one has no effect.
0	COMMAND_DONE	R/W	0h	Interrupt flag 0. A radio operation has finished. (IEEE 802.15.4 mode: A background level radio operation command has finished.) Write zero to clear flag. Write to one has no effect.

23.8.2.6 RFCPEIEN Register (Offset = 14h) [reset = FFFFFFFFh]

RFCPEIEN is shown in Figure 23-26 and described in Table 23-170.

Return to Summary Table.

Interrupt Enable For Command and Packet Engine Generated Interrupts

Figure 23-26. RFCPEIEN Register

31	30	29	28	27	26	25	24
INTERNAL_ERROR	BOOT_DONE	MODULES_UNLOCKED	SYNTH_NO_LOCK	IRQ27	RX_ABORTED	RX_N_DATA_WRITTEN	RX_DATA_WRITTEN
R/W-1h	R/W-1h	R/W-1h	R/W-1h	R/W-1h	R/W-1h	R/W-1h	R/W-1h
23	22	21	20	19	18	17	16
RX_ENTRY_DONE	RX_BUF_FULL	RX_CTRL_ACK	RX_CTRL	RX_EMPTY	RX_IGNORED	RX_NOK	RX_OK
R/W-1h	R/W-1h	R/W-1h	R/W-1h	R/W-1h	R/W-1h	R/W-1h	R/W-1h
15	14	13	12	11	10	9	8
IRQ15	IRQ14	IRQ13	IRQ12	TX_BUFFER_CHANGED	TX_ENTRY_DONE	TX_RETRANS	TX_CTRL_ACK_ACK
R/W-1h	R/W-1h	R/W-1h	R/W-1h	R/W-1h	R/W-1h	R/W-1h	R/W-1h
7	6	5	4	3	2	1	0
TX_CTRL_ACK	TX_CTRL	TX_ACK	TX_DONE	LAST_FG_COMMAND_DONE	FG_COMMAND_DONE	LAST_COMMAND_DONE	COMMAND_DONE
R/W-1h	R/W-1h	R/W-1h	R/W-1h	R/W-1h	R/W-1h	R/W-1h	R/W-1h

Table 23-170. RFCPEIEN Register Field Descriptions

Bit	Field	Type	Reset	Description
31	INTERNAL_ERROR	R/W	1h	Interrupt enable for RFCPEIFG.INTERNAL_ERROR.
30	BOOT_DONE	R/W	1h	Interrupt enable for RFCPEIFG.BOOT_DONE.
29	MODULES_UNLOCKED	R/W	1h	Interrupt enable for RFCPEIFG.MODULES_UNLOCKED.
28	SYNTH_NO_LOCK	R/W	1h	Interrupt enable for RFCPEIFG.SYNTH_NO_LOCK.
27	IRQ27	R/W	1h	Interrupt enable for RFCPEIFG.IRQ27.
26	RX_ABORTED	R/W	1h	Interrupt enable for RFCPEIFG.RX_ABORTED.
25	RX_N_DATA_WRITTEN	R/W	1h	Interrupt enable for RFCPEIFG.RX_N_DATA_WRITTEN.
24	RX_DATA_WRITTEN	R/W	1h	Interrupt enable for RFCPEIFG.RX_DATA_WRITTEN.
23	RX_ENTRY_DONE	R/W	1h	Interrupt enable for RFCPEIFG.RX_ENTRY_DONE.
22	RX_BUF_FULL	R/W	1h	Interrupt enable for RFCPEIFG.RX_BUF_FULL.
21	RX_CTRL_ACK	R/W	1h	Interrupt enable for RFCPEIFG.RX_CTRL_ACK.
20	RX_CTRL	R/W	1h	Interrupt enable for RFCPEIFG.RX_CTRL.
19	RX_EMPTY	R/W	1h	Interrupt enable for RFCPEIFG.RX_EMPTY.
18	RX_IGNORED	R/W	1h	Interrupt enable for RFCPEIFG.RX_IGNORED.
17	RX_NOK	R/W	1h	Interrupt enable for RFCPEIFG.RX_NOK.
16	RX_OK	R/W	1h	Interrupt enable for RFCPEIFG.RX_OK.
15	IRQ15	R/W	1h	Interrupt enable for RFCPEIFG.IRQ15.
14	IRQ14	R/W	1h	Interrupt enable for RFCPEIFG.IRQ14.
13	IRQ13	R/W	1h	Interrupt enable for RFCPEIFG.IRQ13.
12	IRQ12	R/W	1h	Interrupt enable for RFCPEIFG.IRQ12.
11	TX_BUFFER_CHANGED	R/W	1h	Interrupt enable for RFCPEIFG.TX_BUFFER_CHANGED.

Table 23-170. RFCPEIEN Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
10	TX_ENTRY_DONE	R/W	1h	Interrupt enable for RFCPEIFG.TX_ENTRY_DONE.
9	TX_RETRANS	R/W	1h	Interrupt enable for RFCPEIFG.TX_RETRANS.
8	TX_CTRL_ACK_ACK	R/W	1h	Interrupt enable for RFCPEIFG.TX_CTRL_ACK_ACK.
7	TX_CTRL_ACK	R/W	1h	Interrupt enable for RFCPEIFG.TX_CTRL_ACK.
6	TX_CTRL	R/W	1h	Interrupt enable for RFCPEIFG.TX_CTRL.
5	TX_ACK	R/W	1h	Interrupt enable for RFCPEIFG.TX_ACK.
4	TX_DONE	R/W	1h	Interrupt enable for RFCPEIFG.TX_DONE.
3	LAST_FG_COMMAND_DONE	R/W	1h	Interrupt enable for RFCPEIFG.LAST_FG_COMMAND_DONE.
2	FG_COMMAND_DONE	R/W	1h	Interrupt enable for RFCPEIFG.FG_COMMAND_DONE.
1	LAST_COMMAND_DONE	R/W	1h	Interrupt enable for RFCPEIFG.LAST_COMMAND_DONE.
0	COMMAND_DONE	R/W	1h	Interrupt enable for RFCPEIFG.COMMAND_DONE.

23.8.2.7 RFCPEISL Register (Offset = 18h) [reset = FFFF0000h]

RFCPEISL is shown in Figure 23-27 and described in Table 23-171.

Return to Summary Table.

Interrupt Vector Selection For Command and Packet Engine Generated Interrupts

Figure 23-27. RFCPEISL Register

31	30	29	28	27	26	25	24
INTERNAL_ERROR	BOOT_DONE	MODULES_UNLOCKED	SYNTH_NO_LOCK	IRQ27	RX_ABORTED	RX_N_DATA_WRITTEN	RX_DATA_WRITTEN
R/W-1h	R/W-1h	R/W-1h	R/W-1h	R/W-1h	R/W-1h	R/W-1h	R/W-1h
23	22	21	20	19	18	17	16
RX_ENTRY_DONE	RX_BUF_FULL	RX_CTRL_ACK	RX_CTRL	RX_EMPTY	RX_IGNORED	RX_NOK	RX_OK
R/W-1h	R/W-1h	R/W-1h	R/W-1h	R/W-1h	R/W-1h	R/W-1h	R/W-1h
15	14	13	12	11	10	9	8
IRQ15	IRQ14	IRQ13	IRQ12	TX_BUFFER_CHANGED	TX_ENTRY_DONE	TX_RETRANS	TX_CTRL_ACK_ACK
R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h
7	6	5	4	3	2	1	0
TX_CTRL_ACK	TX_CTRL	TX_ACK	TX_DONE	LAST_FG_COMMAND_DONE	FG_COMMAND_DONE	LAST_COMMAND_DONE	COMMAND_DONE
R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h

Table 23-171. RFCPEISL Register Field Descriptions

Bit	Field	Type	Reset	Description
31	INTERNAL_ERROR	R/W	1h	Select which CPU interrupt vector the RFCPEIFG.INTERNAL_ERROR interrupt should use. 0h = Associate this interrupt line with INT_RF_CPE0 interrupt vector 1h = Associate this interrupt line with INT_RF_CPE1 interrupt vector
30	BOOT_DONE	R/W	1h	Select which CPU interrupt vector the RFCPEIFG.BOOT_DONE interrupt should use. 0h = Associate this interrupt line with INT_RF_CPE0 interrupt vector 1h = Associate this interrupt line with INT_RF_CPE1 interrupt vector
29	MODULES_UNLOCKED	R/W	1h	Select which CPU interrupt vector the RFCPEIFG.MODULES_UNLOCKED interrupt should use. 0h = Associate this interrupt line with INT_RF_CPE0 interrupt vector 1h = Associate this interrupt line with INT_RF_CPE1 interrupt vector
28	SYNTH_NO_LOCK	R/W	1h	Select which CPU interrupt vector the RFCPEIFG.SYNTH_NO_LOCK interrupt should use. 0h = Associate this interrupt line with INT_RF_CPE0 interrupt vector 1h = Associate this interrupt line with INT_RF_CPE1 interrupt vector
27	IRQ27	R/W	1h	Select which CPU interrupt vector the RFCPEIFG.IRQ27 interrupt should use. 0h = Associate this interrupt line with INT_RF_CPE0 interrupt vector 1h = Associate this interrupt line with INT_RF_CPE1 interrupt vector
26	RX_ABORTED	R/W	1h	Select which CPU interrupt vector the RFCPEIFG.RX_ABORTED interrupt should use. 0h = Associate this interrupt line with INT_RF_CPE0 interrupt vector 1h = Associate this interrupt line with INT_RF_CPE1 interrupt vector
25	RX_N_DATA_WRITTEN	R/W	1h	Select which CPU interrupt vector the RFCPEIFG.RX_N_DATA_WRITTEN interrupt should use. 0h = Associate this interrupt line with INT_RF_CPE0 interrupt vector 1h = Associate this interrupt line with INT_RF_CPE1 interrupt vector

Table 23-171. RFCPEISL Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
24	RX_DATA_WRITTEN	R/W	1h	Select which CPU interrupt vector the RFCPEIFG.RX_DATA_WRITTEN interrupt should use. 0h = Associate this interrupt line with INT_RF_CPE0 interrupt vector 1h = Associate this interrupt line with INT_RF_CPE1 interrupt vector
23	RX_ENTRY_DONE	R/W	1h	Select which CPU interrupt vector the RFCPEIFG.RX_ENTRY_DONE interrupt should use. 0h = Associate this interrupt line with INT_RF_CPE0 interrupt vector 1h = Associate this interrupt line with INT_RF_CPE1 interrupt vector
22	RX_BUF_FULL	R/W	1h	Select which CPU interrupt vector the RFCPEIFG.RX_BUF_FULL interrupt should use. 0h = Associate this interrupt line with INT_RF_CPE0 interrupt vector 1h = Associate this interrupt line with INT_RF_CPE1 interrupt vector
21	RX_CTRL_ACK	R/W	1h	Select which CPU interrupt vector the RFCPEIFG.RX_CTRL_ACK interrupt should use. 0h = Associate this interrupt line with INT_RF_CPE0 interrupt vector 1h = Associate this interrupt line with INT_RF_CPE1 interrupt vector
20	RX_CTRL	R/W	1h	Select which CPU interrupt vector the RFCPEIFG.RX_CTRL interrupt should use. 0h = Associate this interrupt line with INT_RF_CPE0 interrupt vector 1h = Associate this interrupt line with INT_RF_CPE1 interrupt vector
19	RX_EMPTY	R/W	1h	Select which CPU interrupt vector the RFCPEIFG.RX_EMPTY interrupt should use. 0h = Associate this interrupt line with INT_RF_CPE0 interrupt vector 1h = Associate this interrupt line with INT_RF_CPE1 interrupt vector
18	RX_IGNORED	R/W	1h	Select which CPU interrupt vector the RFCPEIFG.RX_IGNORED interrupt should use. 0h = Associate this interrupt line with INT_RF_CPE0 interrupt vector 1h = Associate this interrupt line with INT_RF_CPE1 interrupt vector
17	RX_NOK	R/W	1h	Select which CPU interrupt vector the RFCPEIFG.RX_NOK interrupt should use. 0h = Associate this interrupt line with INT_RF_CPE0 interrupt vector 1h = Associate this interrupt line with INT_RF_CPE1 interrupt vector
16	RX_OK	R/W	1h	Select which CPU interrupt vector the RFCPEIFG.RX_OK interrupt should use. 0h = Associate this interrupt line with INT_RF_CPE0 interrupt vector 1h = Associate this interrupt line with INT_RF_CPE1 interrupt vector
15	IRQ15	R/W	0h	Select which CPU interrupt vector the RFCPEIFG.IRQ15 interrupt should use. 0h = Associate this interrupt line with INT_RF_CPE0 interrupt vector 1h = Associate this interrupt line with INT_RF_CPE1 interrupt vector
14	IRQ14	R/W	0h	Select which CPU interrupt vector the RFCPEIFG.IRQ14 interrupt should use. 0h = Associate this interrupt line with INT_RF_CPE0 interrupt vector 1h = Associate this interrupt line with INT_RF_CPE1 interrupt vector
13	IRQ13	R/W	0h	Select which CPU interrupt vector the RFCPEIFG.IRQ13 interrupt should use. 0h = Associate this interrupt line with INT_RF_CPE0 interrupt vector 1h = Associate this interrupt line with INT_RF_CPE1 interrupt vector
12	IRQ12	R/W	0h	Select which CPU interrupt vector the RFCPEIFG.IRQ12 interrupt should use. 0h = Associate this interrupt line with INT_RF_CPE0 interrupt vector 1h = Associate this interrupt line with INT_RF_CPE1 interrupt vector

Table 23-171. RFCPEISL Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
11	TX_BUFFER_CHANGED	R/W	0h	Select which CPU interrupt vector the RFCPEIFG.TX_BUFFER_CHANGED interrupt should use. 0h = Associate this interrupt line with INT_RF_CPE0 interrupt vector 1h = Associate this interrupt line with INT_RF_CPE1 interrupt vector
10	TX_ENTRY_DONE	R/W	0h	Select which CPU interrupt vector the RFCPEIFG.TX_ENTRY_DONE interrupt should use. 0h = Associate this interrupt line with INT_RF_CPE0 interrupt vector 1h = Associate this interrupt line with INT_RF_CPE1 interrupt vector
9	TX_RETRANS	R/W	0h	Select which CPU interrupt vector the RFCPEIFG.TX_RETRANS interrupt should use. 0h = Associate this interrupt line with INT_RF_CPE0 interrupt vector 1h = Associate this interrupt line with INT_RF_CPE1 interrupt vector
8	TX_CTRL_ACK_ACK	R/W	0h	Select which CPU interrupt vector the RFCPEIFG.TX_CTRL_ACK_ACK interrupt should use. 0h = Associate this interrupt line with INT_RF_CPE0 interrupt vector 1h = Associate this interrupt line with INT_RF_CPE1 interrupt vector
7	TX_CTRL_ACK	R/W	0h	Select which CPU interrupt vector the RFCPEIFG.TX_CTRL_ACK interrupt should use. 0h = Associate this interrupt line with INT_RF_CPE0 interrupt vector 1h = Associate this interrupt line with INT_RF_CPE1 interrupt vector
6	TX_CTRL	R/W	0h	Select which CPU interrupt vector the RFCPEIFG.TX_CTRL interrupt should use. 0h = Associate this interrupt line with INT_RF_CPE0 interrupt vector 1h = Associate this interrupt line with INT_RF_CPE1 interrupt vector
5	TX_ACK	R/W	0h	Select which CPU interrupt vector the RFCPEIFG.TX_ACK interrupt should use. 0h = Associate this interrupt line with INT_RF_CPE0 interrupt vector 1h = Associate this interrupt line with INT_RF_CPE1 interrupt vector
4	TX_DONE	R/W	0h	Select which CPU interrupt vector the RFCPEIFG.TX_DONE interrupt should use. 0h = Associate this interrupt line with INT_RF_CPE0 interrupt vector 1h = Associate this interrupt line with INT_RF_CPE1 interrupt vector
3	LAST_FG_COMMAND_DONE	R/W	0h	Select which CPU interrupt vector the RFCPEIFG.LAST_FG_COMMAND_DONE interrupt should use. 0h = Associate this interrupt line with INT_RF_CPE0 interrupt vector 1h = Associate this interrupt line with INT_RF_CPE1 interrupt vector
2	FG_COMMAND_DONE	R/W	0h	Select which CPU interrupt vector the RFCPEIFG.FG_COMMAND_DONE interrupt should use. 0h = Associate this interrupt line with INT_RF_CPE0 interrupt vector 1h = Associate this interrupt line with INT_RF_CPE1 interrupt vector
1	LAST_COMMAND_DONE	R/W	0h	Select which CPU interrupt vector the RFCPEIFG.LAST_COMMAND_DONE interrupt should use. 0h = Associate this interrupt line with INT_RF_CPE0 interrupt vector 1h = Associate this interrupt line with INT_RF_CPE1 interrupt vector
0	COMMAND_DONE	R/W	0h	Select which CPU interrupt vector the RFCPEIFG.COMMAND_DONE interrupt should use. 0h = Associate this interrupt line with INT_RF_CPE0 interrupt vector 1h = Associate this interrupt line with INT_RF_CPE1 interrupt vector

23.8.2.8 RFACKIFG Register (Offset = 1Ch) [reset = 0h]

RFACKIFG is shown in Figure 23-28 and described in Table 23-172.

Return to Summary Table.

Doorbell Command Acknowledgement Interrupt Flag

Figure 23-28. RFACKIFG Register

31	30	29	28	27	26	25	24
RESERVED							
R-0h							
23	22	21	20	19	18	17	16
RESERVED							
R-0h							
15	14	13	12	11	10	9	8
RESERVED							
R-0h							
7	6	5	4	3	2	1	0
RESERVED							ACKFLAG
R-0h							R/W-0h

Table 23-172. RFACKIFG Register Field Descriptions

Bit	Field	Type	Reset	Description
31-1	RESERVED	R	0h	Software should not rely on the value of a reserved. Writing any other value than the reset value may result in undefined behavior.
0	ACKFLAG	R/W	0h	Interrupt flag for Command ACK

23.8.2.9 SYSGPOCTL Register (Offset = 20h) [reset = 0h]

SYSGPOCTL is shown in Figure 23-29 and described in Table 23-173.

Return to Summary Table.

RF Core General Purpose Output Control

Figure 23-29. SYSGPOCTL Register

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RESERVED															
R-0h															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GPOCTL3				GPOCTL2				GPOCTL1				GPOCTL0			
R/W-0h				R/W-0h				R/W-0h				R/W-0h			

Table 23-173. SYSGPOCTL Register Field Descriptions

Bit	Field	Type	Reset	Description
31-16	RESERVED	R	0h	Software should not rely on the value of a reserved. Writing any other value than the reset value may result in undefined behavior.
15-12	GPOCTL3	R/W	0h	RF Core GPO control bit 3. Selects which signal to output on the RF Core GPO line 3. 0h = CPE GPO line 0 1h = CPE GPO line 1 2h = CPE GPO line 2 3h = CPE GPO line 3 4h = MCE GPO line 0 5h = MCE GPO line 1 6h = MCE GPO line 2 7h = MCE GPO line 3 8h = RFE GPO line 0 9h = RFE GPO line 1 Ah = RFE GPO line 2 Bh = RFE GPO line 3 Ch = RAT GPO line 0 Dh = RAT GPO line 1 Eh = RAT GPO line 2 Fh = RAT GPO line 3
11-8	GPOCTL2	R/W	0h	RF Core GPO control bit 2. Selects which signal to output on the RF Core GPO line 2. 0h = CPE GPO line 0 1h = CPE GPO line 1 2h = CPE GPO line 2 3h = CPE GPO line 3 4h = MCE GPO line 0 5h = MCE GPO line 1 6h = MCE GPO line 2 7h = MCE GPO line 3 8h = RFE GPO line 0 9h = RFE GPO line 1 Ah = RFE GPO line 2 Bh = RFE GPO line 3 Ch = RAT GPO line 0 Dh = RAT GPO line 1 Eh = RAT GPO line 2 Fh = RAT GPO line 3

Table 23-173. SYSGPOCTL Register Field Descriptions (continued)

Bit	Field	Type	Reset	Description
7-4	GPOCTL1	R/W	0h	RF Core GPO control bit 1. Selects which signal to output on the RF Core GPO line 1. 0h = CPE GPO line 0 1h = CPE GPO line 1 2h = CPE GPO line 2 3h = CPE GPO line 3 4h = MCE GPO line 0 5h = MCE GPO line 1 6h = MCE GPO line 2 7h = MCE GPO line 3 8h = RFE GPO line 0 9h = RFE GPO line 1 Ah = RFE GPO line 2 Bh = RFE GPO line 3 Ch = RAT GPO line 0 Dh = RAT GPO line 1 Eh = RAT GPO line 2 Fh = RAT GPO line 3
3-0	GPOCTL0	R/W	0h	RF Core GPO control bit 0. Selects which signal to output on the RF Core GPO line 0. 0h = CPE GPO line 0 1h = CPE GPO line 1 2h = CPE GPO line 2 3h = CPE GPO line 3 4h = MCE GPO line 0 5h = MCE GPO line 1 6h = MCE GPO line 2 7h = MCE GPO line 3 8h = RFE GPO line 0 9h = RFE GPO line 1 Ah = RFE GPO line 2 Bh = RFE GPO line 3 Ch = RAT GPO line 0 Dh = RAT GPO line 1 Eh = RAT GPO line 2 Fh = RAT GPO line 3

23.8.3 RFC_PWR Registers

Table 23-174 lists the memory-mapped registers for the RFC_PWR. All register offset addresses not listed in Table 23-174 should be considered as reserved locations and the register contents should not be modified.

Table 23-174. RFC_PWR Registers

Offset	Acronym	Register Name	Section
0h	PWMCLKEN	RF Core Power Management and Clock Enable	Section 23.8.3.1

23.8.3.1 PWMCLKEN Register (Offset = 0h) [reset = 1h]

PWMCLKEN is shown in Figure 23-30 and described in Table 23-175.

Return to Summary Table.

RF Core Power Management and Clock Enable

Figure 23-30. PWMCLKEN Register

31	30	29	28	27	26	25	24
RESERVED							
R-0h							
23	22	21	20	19	18	17	16
RESERVED							
R-0h							
15	14	13	12	11	10	9	8
RESERVED					RFCTRC	FSCA	PHA
R-0h					R/W-0h	R/W-0h	R/W-0h
7	6	5	4	3	2	1	0
RAT	RFERAM	RFE	MDMRAM	MDM	CPERAM	CPE	RFC
R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R/W-0h	R-1h

Table 23-175. PWMCLKEN Register Field Descriptions

Bit	Field	Type	Reset	Description
31-11	RESERVED	R	0h	Software should not rely on the value of a reserved. Writing any other value than the reset value may result in undefined behavior.
10	RFCTRC	R/W	0h	Enable clock to the RF Core Tracer (RFCTRC) module.
9	FSCA	R/W	0h	Enable clock to the Frequency Synthesizer Calibration Accelerator (FSCA) module.
8	PHA	R/W	0h	Enable clock to the Packet Handling Accelerator (PHA) module.
7	RAT	R/W	0h	Enable clock to the Radio Timer (RAT) module.
6	RFERAM	R/W	0h	Enable clock to the RF Engine RAM module.
5	RFE	R/W	0h	Enable clock to the RF Engine (RFE) module.
4	MDMRAM	R/W	0h	Enable clock to the Modem RAM module.
3	MDM	R/W	0h	Enable clock to the Modem (MDM) module.
2	CPERAM	R/W	0h	Enable clock to the Command and Packet Engine (CPE) RAM module. As part of RF Core initialization, set this bit together with CPE bit to enable CPE to boot.
1	CPE	R/W	0h	Enable processor clock (hclk) to the Command and Packet Engine (CPE). As part of RF Core initialization, set this bit together with CPERAM bit to enable CPE to boot.
0	RFC	R	1h	Enable essential clocks for the RF Core interface. This includes the interconnect, the radio doorbell DBELL command interface, the power management (PWR) clock control module, and bus clock (sclk) for the CPE. To remove possibility of locking yourself out from the RF Core, this bit can not be cleared. If you need to disable all clocks to the RF Core, see the PRCM:RFCCLKG.CLK_EN register.