EZMAC®PRO USER'S GUIDE

1. Introduction

EZMAC®Pro is a wireless communication software module for embedded systems using EZRadioPRO® radios.

It transmits and receives data in short packets via an RF link in the ISM RF bands. EZMAC PRO runs in the background on the main application microcontroller in two interrupt service routines, using the resources of the EZRadioPRO RF IC to minimize CPU overhead.

EZMAC PRO supports a wide range of addressing modes, collision detections, and error detection capabilities. A built-in acknowledgement and packet forwarding feature is also included. Additionally, it uses frequency redundancy to ensure that information reaches its intended destination.

EZMAC PRO is designed so designers can easily build either peer-to-peer or star networks with up to 255 addressable nodes on a given sub-net.

EZMAC PRO supports all of the transmitters, receivers, and transceivers from the EZRadioPRO family. Designed in a modular approach where features can also be enabled or disabled prior to compilation so that the code size can be optimized to a given application. "2.2. Memory Requirements" on page 6 gives more detailed information about the required code and RAM sizes.

EZMAC PRO is developed in the C programming software language for the Silicon Laboratories C8051F9xx microprocessor family. The code is compiler-independent and requiring only the compiler_defs.h and processor definition header files so that designers can compile the code using a variety of compilers implementing the compiler_defs.h support. A number of compilers implementing this feature include vendors such as Keil (http://www.keil.com), IAR 8051 (http://www.iar.com), HiTech 8051 (http://htsoft.com), Small Device C Compiler (SDCC) (http://sdcc.sourceforge.net), Raisonance (http://www.raisonance.com), Tasking/Altium (http://www.altium.com/tasking), etc. The code can also be easily adapted for use with other microcontrollers with the appropriate license from Silicon Labs.

EZMAC PRO is designed on the SDBC-DK3 software development board using one of the available EZRadioPRO Testcards (TX/RX Split, Single Antenna, Antenna Diversity Testcards). However, the software can also be run on the EZLink™ fast prototyping platform. For all software examples, a Silicon Labs IDE project file is provided for both platforms.

TABLE OF CONTENTS

Section **Page**

2. Hardware Requirements

2.1. Peripherals Requirements

EZMAC PRO is designed to operate with the EZRadioPRO family using the SPI port (the software uses the HW SPI1 peripheral of the microcontroller on the C8051F930 reference design) and interrupt pin of the MCU. The microcontroller should be able to wake up from sleep mode if the transceiver pulls the interrupt pin to logic low.

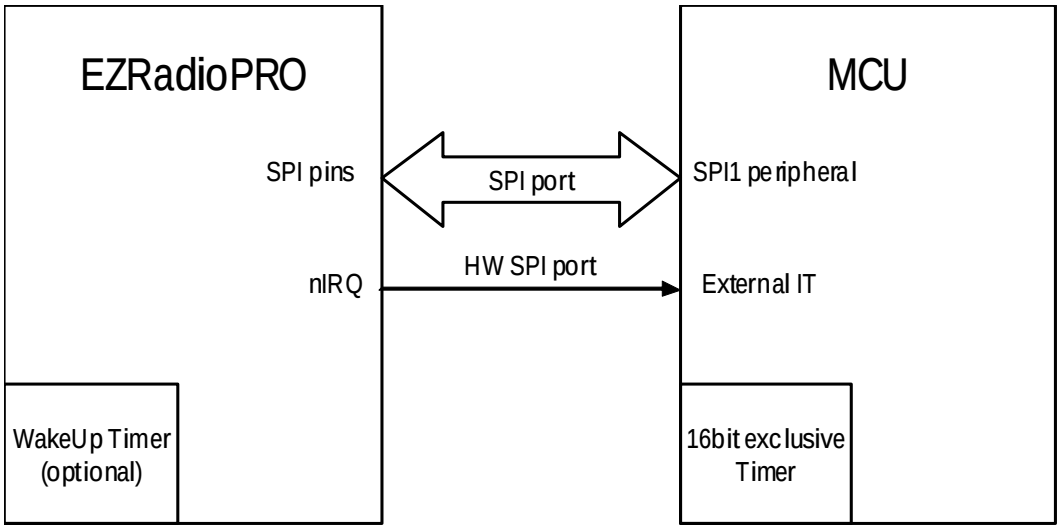

Figure 1. HW Configuration

EZMAC PRO also requires a 16-bit exclusive timer. The MCU should be able to wake up from IDLE mode if the timer expires.

EZMAC PRO can provide a low-frequency time base for the application layer to realize time-synchronized operation. For this purpose, the Wake Up Timer of the radio is used.

The reference source code is designed for the C8051F930 microprocessor, and uses the INT0 interrupt source, Timer0 timer module, and SPI1 peripheral.

EZMAC PRO does not have any additional peripheral requirements.

2.2. Memory Requirements

Most of the features are selected via compile options. These can be enabled and disabled according to the application needs. Tables 1, 2, and 3 below give an overview of the required code and memory sizes using different build options.

Notes: All the source code was compiled with Keil C51 Version 8.16a Release compiler for code size estimation. The parameters of the compiler were set to:

1. Optimization level: Level7
2. Emphasis: Favor Fast Code
3. Variable location: Small (dcata)
4. Code size: Large (64K functions)

Table 1. Transmit Operation

Basic Functions	AUTOMATIC_FREQ_CHANGE_SUPPORTED	LOW_BATTERY_DETECT_IS_USED	LOW_FREQUENCY_TIMER_IS_USED	ENABLE_BYTE_COUNTERS	ANTENNA_DIVERSITY_ENABLED	Code Size in Bytes	RAM Size in Bytes
X						2754	77 ... 96
X	X					2852	
X	X	X				2976	
X	X	X	X			3155	
X	X	X	X	X		3305	
X	X	X	X	X	X	3338	

Table 2. Receive Operation

Basic functions	LOW_BATTERY_DETECT_IS_USED	LOW_FREQUENCY_TIMER_IS_USED	ANTENNA_DIVERSITY_ENABLED	Code Size in Bytes	RAM Size in Bytes
X				3902	156 ... 163
X	X			4054	
X	X	X		4262	
X	X	X	X	4433	

Table 3. Transceiver Operation

Basic Functions	AUTOMATIC_FREQ_CHANGE_SUPPORTED	AUTOMATIC_ACK_SUPPORTED	LISTEN_BEFORE_TALK_SUPPORTED	PACKET_FORWARDING_SUPPORTED	LOW_BATTERY_DETECT_IS_USED	LOW_FREQUENCY_TIMER_IS_USED	ENABLE_BYTE_COUNTERS	ANTENNA_DIVERSITY_ENABLED	Code Size in Bytes	RAM Size in Bytes
X									4338	167 ... 232
X	X								4426	
X	X	X							4948	
X	X	X	X						5536	
X	X	X	X	X					6496	
X	X	X	X	X	X				6649	
X	X	X	X	X	X	X			6858	
X	X	X	X	X	X	X	X		7006	
X	X	X	X	X	X	X	X	X	7205	

2.3. MCU Clock

The EZMAC PRO reference design is designed to run on the C8051F930 microcontroller using a minimum 4 MHz system clock. Any system clock can be selected above 4 MHz. The system clock is defined by the SYSCLK_MHZ definition in the hardware_defs.h header file. All of the internal timing parameters are calculated based on this definition; so, it is very important to define accurately.

EZMAC PRO is designed in a way that does not require a crystal-based clock source for the microcontroller; the critical timings are done by the radio itself.

3. EZMAC PRO Supported RF Devices

EZMAC PRO is designed for the EZRadioPRO family. The family consists of transmitter (Si4030, Si4031, and Si4032), receiver (Si4330), and transceiver (Si4430, Si4431, and Si4432) RF integration circuits. The source code can be compiled for each device type: transmitter only, receiver only, or transceiver operation using the following definitions:

- TRANSMITTER_ONLY_OPERATION
- RECEIVER_ONLY_OPERATION
- TRANSCEIVER_OPERATION

If EZMAC PRO is compiled for transmitter or receiver operation, the non-required code segments are not compiled with the source code. The FLASH and RAM requirements of the microprocessor can be optimized for those applications.

For proper operation the revision of the radio chip has to be selected in the EZMacPro_defs.h header file:

- #define SI4432_V2 definition should be enabled if Si4032 revV2 or Si4432 revV2 chip is used
- #define SI4431_A0 definition should be enabled if Si4030, Si4031, Si4330, Si4430, Si4431 revA0 chip is used
- #define SI443X_B1 definition should be enabled if revB1 chip is used.

Notes: Only one of the options can be selected in the same time. In the other case, the compiler gives one of the following error messages:

1. "Both Transmitter Only and Receiver Only cannot be defined!"
2. "Both Transceiver and Transmitter Only cannot be defined!"
3. "Both Transceiver and Receiver Only cannot be defined!"

3.1. Receiving Process

The key feature of EZMAC PRO is the frequency-redundant receiving procedure. The receiver can scan up to four channels for increasing the robustness of the communication. The frequency-redundant receiver procedure allows a packet to be sent in a randomly-selected channel, and the receiver finds the packet independently of which channel it is transmitted through.

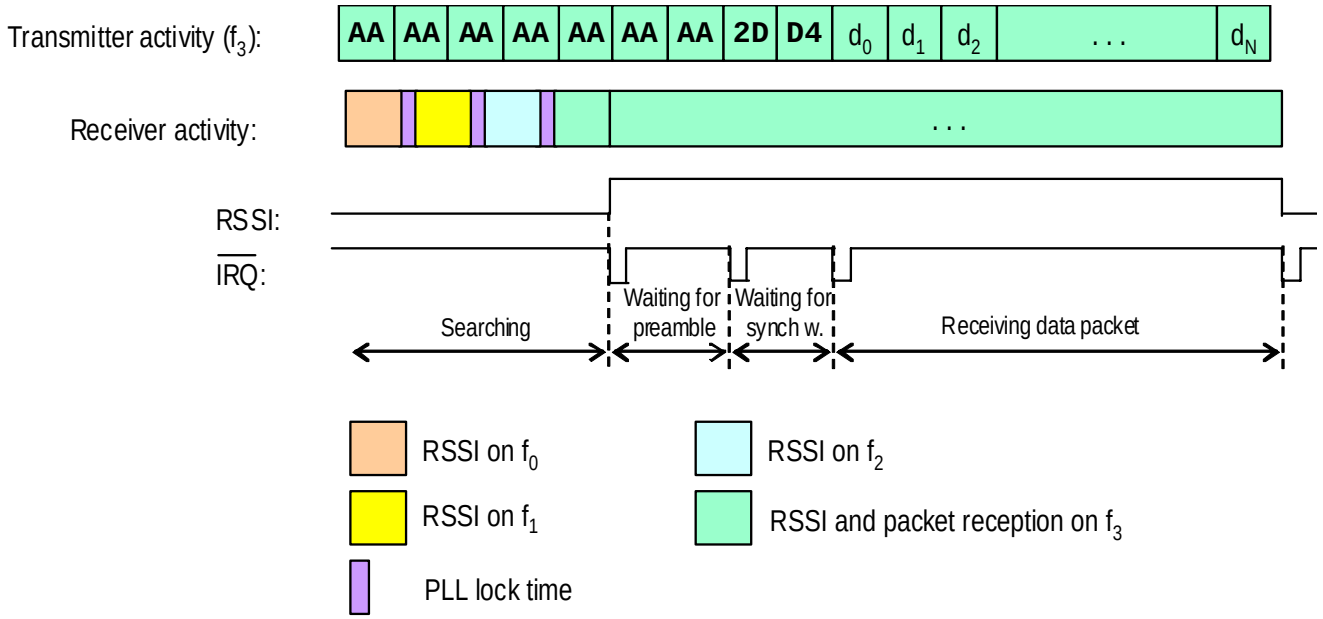

Figure 2. Receiving Process

The receiver scans the channels next to each by measuring the receiver signal strength indicator (RSSI) level. Based on the signal strength level, it quickly evaluates whether there is any RF activity on that channel. If there is no RF activity (the measured RSSI is below the programmable threshold), it jumps to the next channel. If the channel is occupied, further evaluation is needed. It then stays on the channel and waits to receive the preamble. If the preamble is detected within timeout, it waits for the synchron word and receives the entire packet. If the preamble is not detected within timeout or false preamble detection occurs, the node jumps to the next channel.

The receiver has to find the transmitted packet during the time the transmitter sends the preamble or else the receiver cannot receive it correctly. The receiver requires about 1 byte time to evaluate a channel (data-rate-dependent). Jumping from frequency to frequency requires about 200 µs (the receiver has to wait for the PLL settling time). Therefore, the transmitter has to send as long a preamble as required for the receiver to evaluate all the used channels. EZMAC PRO sets the preamble length automatically; the higher level software layers do not need to deal with this concern.

The EZRadioPRO devices require different length of preamble if the Auto Frequency Control feature is enabled or disabled. The Auto Frequency Control (AFC) works in receive mode during receiving the preamble. It tries to find the exact center frequency of the transmitter, so cancelling the offset between the transmitter and receiver node. If the AFC of the radio is enabled, then the radio can tolerate offset up to 0.35 times the IF bandwidth. If the AFC is not used, then the radio can tolerate offset up to 0.25 times the IF bandwidth.

Notes:

1. EZMAC PRO source code has the AFC enabled for Si4x3x revA0, Si4x3x revB1 devices and disabled for the Si4x32 revV2 devices.
2. The AFC selection is part the table which stores the RF parameters. For more details, please see "3.2. Data Rate and Frequency Channel Selection" on page 12.
3. The preamble detection threshold for every EZRadioPRO device is set to 5 nibbles (2.5 bytes). If less preamble detection threshold is sufficient for the given application, then the number of transmit preamble (for every data rate and frequency selection) can be decreased by the difference.

Table 4. Number of Required Transmit Preamble Bytes if AFC is not Used

Data Rate (kbps)	Number of Required Preamble (Byte)			
	4 Channels	3 Channels	2 Channels	1 Channel
2.4	7	6	5	4
4.8	7	6	5	4
9.6	8	6	5	4
10	8	6	5	4
20	8	7	5	4
50	10	8	6	4
100	13	10	7	4
128	14	11	7	4

Table 5. Number of Required Transmit Preamble Bytes if AFC is Enabled

Data Rate (kbps)	Number of Required Preamble (Byte)			
	4 Channels	3 Channels	2 Channels	1 Channel
2.4	8	7	6	5
4.8	8	7	6	5
9.6	9	7	6	5
10	9	7	6	5
20	9	8	6	5
50	11	9	7	5
100	14	11	8	5
128	15	12	8	5

The EZRadioPRO devices support an automatic antenna diversity feature. EZMAC PRO can be compiled for this operation by enabling the ANTENNA_DIVERSITY_ENABLED definition in the EZMacPRO_defs.h file. If antenna diversity is used, the receiver needs more time to evaluate the channel; so, the transmitter has to send a longer preamble. The receive process for antenna diversity operation is shown in Figure 3.

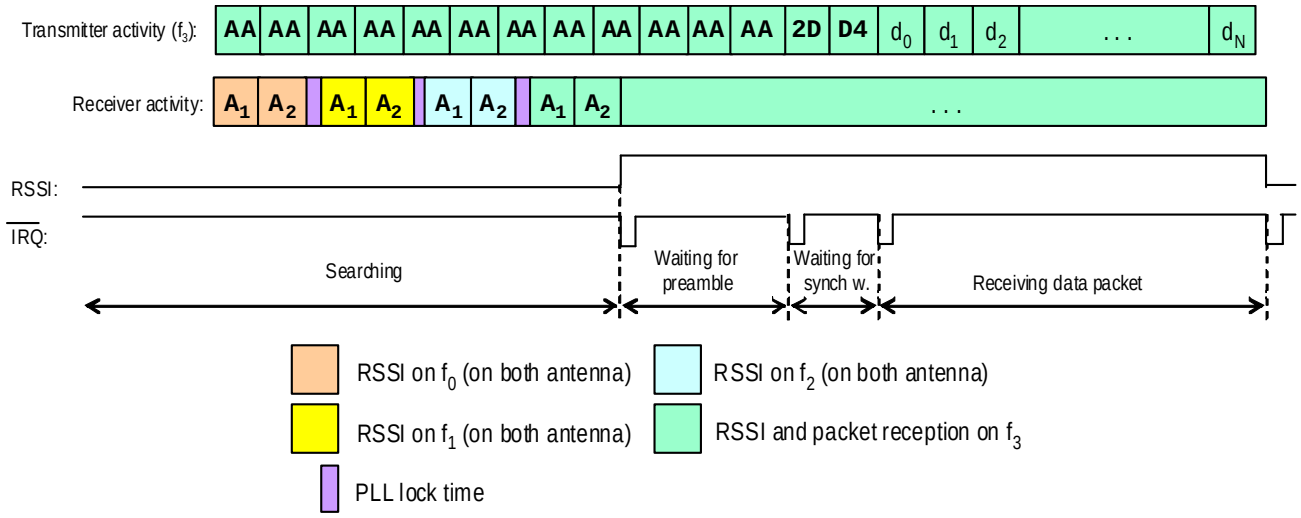

Figure 3. Receive Process (Antenna Diversity Enabled)

During the scan activity EZMAC PRO disables the built in antenna diversity mechanism and evaluates the channels manually: it scans the channels next to each by measuring the RSSI (receiver signal strength indicator) level on antenna 1 first. Based on the signal strength level it quickly evaluates whether there is any RF activity on that channel using antenna 1 or not. If there is no RF activity (the measured RSSI is below the programmable threshold), then it switch to antenna 2 and quickly evaluates the RSSI too. If there is no RF activity on the selected channel using antenna 2, then it jumps to the next channel.

If the RSSI level is above the programmable threshold on antenna 1 or antenna 2, then EZMAC PRO immediately enables the built in antenna diversity mechanism of the EZRadioPRO devices and tries to receive the packet (if antenna 1 shows RF activity, then the RSSI measurement on antenna 2 is discarded).

The packet reception consist of the following activities: after enabling the antenna diversity mechanism EZMAC PRO waits for evaluating which antenna gives better performances and also it waits for detecting the preamble. If the preamble is detected within timeout, then it waits for the synchron word and receives the entire packet. If the preamble is not detected within timeout or false preamble detection occurred, the node jumps to the next channel.

Table 6. Number of Required Preamble Bytes

Data Rate (kbps)	Number of Required Preamble			
	4 Channels	3 Channels	2 Channels	1 Channel
2.4	11	9	7	5
4.8	11	9	7	5
9.6	12	9	7	5
10	12	9	7	5
20	12	10	7	5
50	14	11	8	5
100	17	13	9	5
128	18	14	9	5

If the antenna diversity is enabled, then there is no difference in the number of required preamble whether the AFC is used or not.

Note: Only one option can selected: either non-antenna diversity or antenna diversity. The decision has to be made before compiling the source code and enabling or disabling the ANTENNA_DIVERSITY_ENABLED definition in the EZMacPRO_defs.h file.

3.2. Data Rate and Frequency Channel Selection

EZMAC PRO hides the RF parameter settings from the higher software layers. The application layer only has to select the data rate, and all other RF parameters are set automatically. Up to eight data rates can be selected runtime, during the MAC operation (DR[2:0] control bits). The data rate should not be fixed before compiling the source code; the higher layer can optimize the data throughput runtime depending on application needs.

The RF settings, timing parameters, and frequency information for the eight data rates are stored in a table that lists:

- Deviation and data rate register settings
- Modem register settings
- Required preamble length for antenna diversity and non antenna diversity operation
- Start frequency, the frequency step, and the maximum number of frequency channels

The source code contains predefined tables for all the ISM bands (434, 868, and 915 MHz) with the most common data rates (2.4 kbps ... 128 kbps). Separate tables are provided for antenna diversity and non antenna diversity mode. The RF parameters and the frequency assignment are provided to comply with the appropriate standard of the selected ISM band (ETSI or FCC depending on which frequency the network is operating). The tables are well separated in the si4432_const_v2.c, si443x_const_b1.c, and si4431_const_a0.c files and can be edited easily (by replacing one or more lines of the table, changing parameters like enabling or disabling the AFC, etc.) if the application requires a different data rate or RF setting. Figure 4 shows one line of the table. The different fields are explained in Table 7.

```
Const SEGMENT_VARIABLE (RfSettings[NUMBER_OF_SAMPLE_SETTING][NUMBER_OF_PARAMETER], U8, code) =
   {
    //IFBW, COSR, CRO2, CRO1, CRO0, CTG1, CTG0, TDR1, TDR0, MMC1, TXFDEV, RXFDEV, AFC, AFC Limiter, FR_S1,
FR_S2,  FR_S3,
    {0x1D, 0x41, 0x60, 0x27, 0x52, 0x00, 0x04, 0x13, 0xa9,  0x20, 0x3d, 0x3d, 0x40, 0x21, 0x75, 0x0A, 0x80, \\

//FR_ST, MAX_CH,  PR1,  PR2,  PR3,  PR4
48,     60,       5,    6,    7,    8},
...
};
```

Figure 4. Example of the RF Parameter Table

Table 7. Explanation of RF Parameter Table Fields

Mnemonic	Register Name	Note
IFBW	IF Filter Bandwidth	These are the modem registers. These configure the receiver for the right operation.
COSR	Clock Recovery Oversampling Ratio	
CRO2 … 0	Clock Recovery Offset 1 … 2	
CTG1 … 0	Clock Recovery Timing Loop Gain 0 … 1	
TDR1 … 0	TX Data Rate 1 …. 0	
MMC1	Modulation Mode Control 1	
TXFDEV	Frequency Deviation	This register defines the TX deviation.
RXFDEV	Frequency Deviation	This setting is used in case of Si4x32 revV2 only if AFC is enabled. It sets the AFC Limit.
AFC	AFC Loop Gearshift Override	This register defines whether the AFC is enabled or disabled.
AFC Limit	AFC Timing Control	This register is used in case of Si4x3x revA0 only if the AFC is enabled. It sets the AFC Limit.
FR_S1	Frequency Band Select	These registers define the frequency of the first channel.
FR_S2, FR_S3	Nominal Carrier Frequency 1 … 0	
FR_ST	Frequency Hopping Step Size	This register defines the channel spacing.
MAX_CH	—	Number of the channels.
PR1, …, PR4	—	Number of transmitted preamble bytes for 1, …, 4 channel cases.

The frequency band where the network is operating has to be selected before compiling the source code by using the correct definition:

- FREQUENCY_BAND_434
- FREQUENCY_BAND_868
- FREQUENCY_BAND_915

3.3. Packet Format

EZMAC PRO supports two packet configurations: standard EZMAC PRO compatible and extended packet configuration. The packet format has to be selected before compiling the source code by enabling only one of the following definitions in the EZMacPRO_defs.h file:

- STANDARD_EZMAC_PACKET_FORMAT
- EXTENDED_PACKET_FORMAT

Extended packet format contains an extra header byte (Control byte), which is used for packet forwarding and automatic acknowledgement. If these features are enabled, the extended packet configuration has to be selected.

EZMAC PRO transmits data in short packets; the maximum payload is 64 bytes.

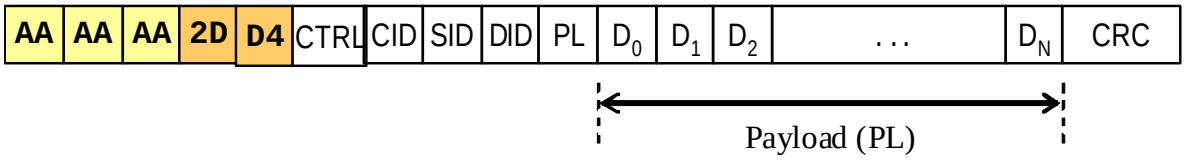

Figure 5. Packet Configuration

Table 8. Packet Configuration

Name	Description	Size [bytes]
AA AA AA AA AA	Preamble	min. 5
2D D4	Synchron pattern	2
CTRL [1]	Control byte	1
CID [2]	Customer ID	1
SID	Sender ID	1
DID	Destination ID	1
PL [2]	Payload Length	1
D_0 … D_N	Data bytes	0 … 64
CRC	Cyclic Redundancy Code	2

Notes:
1. This byte is used only for the extended packet configuration mode.
2. Optional field; may not be used on simple applications.

The number of preamble bytes is dependent on the number of selected frequencies used in the system. EZMAC PRO automatically sets the transmit preamble according to the number of enabled channels.

Control byte is used only in the extended packet configuration mode for supporting the packet forwarding and automatic acknowledgement (see “3.5. Packet Forwarding” for more details).

Customer ID (CID) is used to avoid unexpected interactions between the different systems using EZMAC PRO that may be installed in the same area in close proximity to each other. Although this byte is optional, it is recommended. The Customer ID feature has to be selected before compiling the source code. If the CID_IS_USED definition is enabled, the CID header field is added into the packet format, and the CID filter can be used. The value of the CID field can be defined during runtime by the SCID register.

Sender ID (SID) and Destination ID (DID) are used to exactly identify the devices taking part in the actual data exchange. Based on the addresses, EZMAC PRO has several packet filtering options (see "4.2. Packet Filtering Method" on page 23 for more details).

Payload Length (PL) is used in applications where the number of the transmitted data bytes changes dynamically. EZMAC PRO also supports the fixed payload length operations. In this case, the PL field is not transmitted in the packet.

For D0 … DN the maximum data to transmit is 64 byes.

Cyclic Redundancy Code (CRC) is used to recognize if there is any bad data bit in the packet. The length of the CRC is always 16 bits.

Note: If both packet configurations are enabled in the source code, the compiler gives the following error message: "Only one packet format can be selected!".

3.4. Listen before Talk

EZMAC PRO performs a Listen-Before-Talk prior to sending a packet if the feature is enabled by the LBTEN control bit. This mechanism prevents packet transmission if the channel is already being used by another node to avoid a packet collision. The Listen-Before-Talk mechanism is configurable and can be easily set to comply with proprietary protocols and standards such as the ETSI standard.

The Listen-Before-Talk mechanism operates as shown in Figure 6.

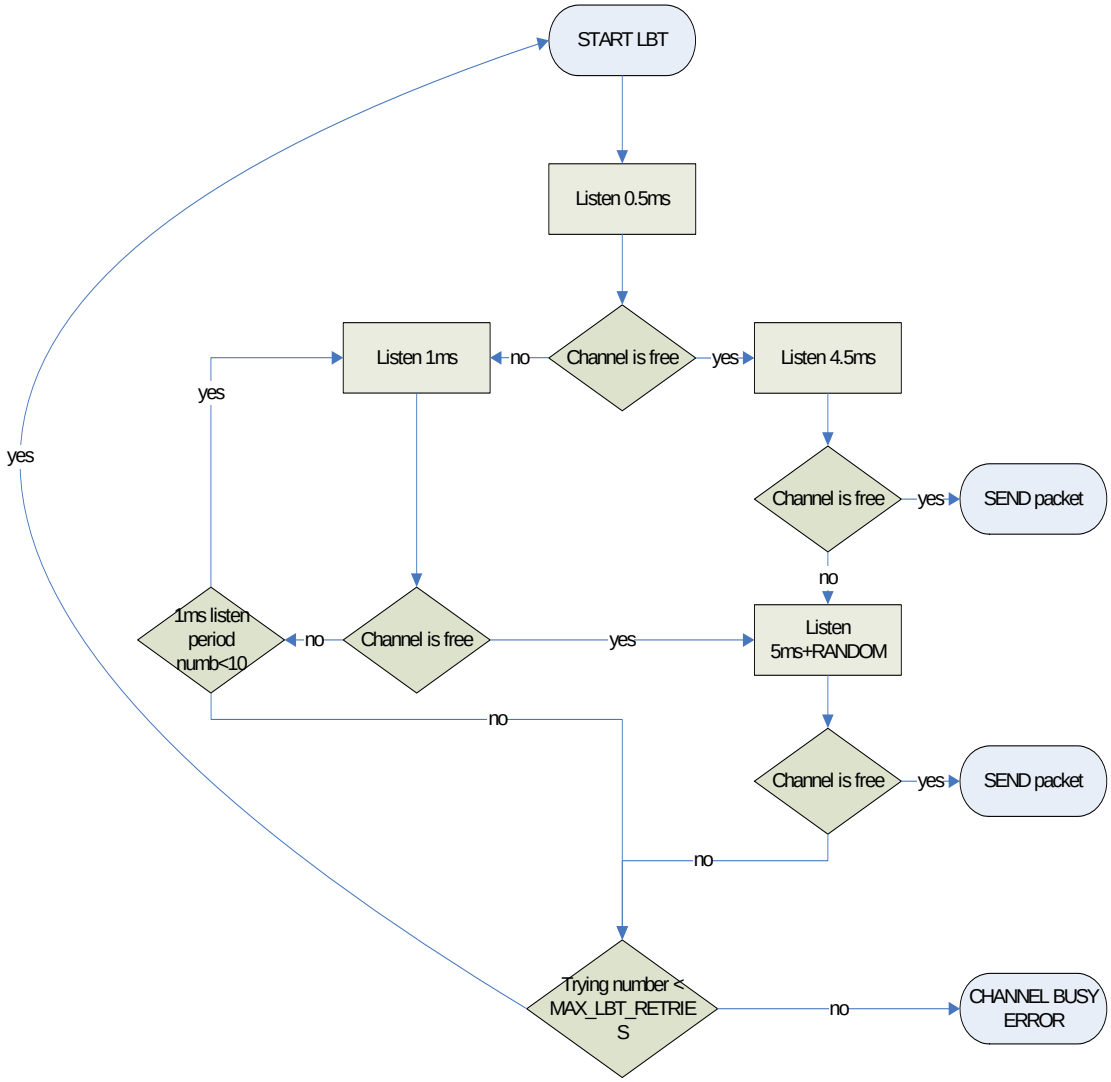

Figure 6. The Listen-Before-Talk Mechanism

1. EZMAC PRO enables the receiver chain and listens to the channel for 0.5 ms. If the channel remains free during this time as determined by the RSSI value not exceeding a predetermined Listen-Before-Talk limit, then EZMAC PRO continues to listen to the channel for an additional 4.5 ms. If the channel remains available for the full 5.0 ms time period, then EZMAC PRO will send the packet. By contrast, if the channel becomes busy in the last 4.5 ms, then the state machine operation jumps to step 3.
2. If the channel was considered busy during the first 0.5 ms, then EZMAC PRO samples the RSSI every 1 ms until the RSSI falls below the threshold or the total of 10 ms expires. If the total of 10 ms expires, then the state machine operation jumps to step 5.
3. If the RSSI falls below the threshold during this period, EZMAC PRO will wait an additional 5 ms plus an additional pseudo random time. The range and resolution of the pseudo random time are configurable:

   TPS =n x LBTI [6:0]

   where n is a random number between 0…15 and LBTI is the Listen-Before-Talk Interval register. LBTI can be set in byte intervals (1…127 byte(s) interval) or it can be set in fixed time intervals (100 µs…12.7 ms). The selection between the two options depends on an application's need (e.g., ETSI specifies the LBTI in 0.5 ms intervals.)

4. If the RSSI remains below the threshold during the “fixed listen time” and the “pseudo random listen time,” then EZMAC PRO will send the packet. If the RSSI is above the limit during this time period, then the state machine operation will jump to step 5.
5. If the attempt number is below MAX_LBT_RETRIES, then the node repeats the Listen-Before-Talk mechanism. The MAX_LBT_RETRIES is determined by a definition in EZMacPRO_defs.h.
   If the attempt number is above the limit, then the state machine operation jumps to step 6.
6. If EZMAC PRO is unable to send the packet because a frequency was found to be busy, then the stack will go into the TX_ERROR_CHANNEL_BUSY state and call the EZMacProTX_ErrorLBT_Timeout() callback function.

Notes:

1. If the application does not require the Listen-Before-Talk feature, it can be disabled by disabling the LISTEN_BEFORE_TALK_SUPPORTED definition. In this case, the feature will not be compiled into the source code, saving code and RAM space. When implementing Packet Forwarding, the Listen-Before-Talk becomes mandatory and cannot be disabled.
2. When sending an acknowledgement packet, EZMAC PRO does not perform the Listen-Before-Talk operation.
3. If the source code is compiled for a transmitter only device, the Listen-Before-Talk feature is not available.

3.5. Packet Forwarding

Packet forwarding enables retransmitting a packet if a node receives it and it is not sent to that particular node. Using this feature, the packet can be sent to a node that is not in the range of the originator. The packet can reach the destination via multiple nodes. This way, the range between nodes (and the coverage of the network) can be increased drastically.

Packet forwarding requires an additional header byte (Control byte) in the transmitted packet. To understand how Packet Forwarding works, the fields of the Control byte have to be understood first.

CTRL							
7	6	5	4	3	2	1	0
SEQ[3:0]				ACK	ACKRQ	RAD[1:0]	

Figure 7. CTRL Field of the Packet

3.5.1. Radius Field

The Radius field specifies how many times the packet can be forwarded. The node, which received a packet, forwards the packet, if all of the following apply:

- The radius field is not zero
- The destination address is different than the self address of the receiver node
- The packet forwarding feature is enabled

The node receives the entire packet and decreases the radius by one. EZMAC PRO calls the void EZMacPRO_PacketForwarding(void) callback function to inform the next higher layer about the packet forwarding. The next higher layer can then read and modify the content of the packet payload (e.g. adding routing information into it) before the packet is forwarded. Finally, the MAC retransmits the packet. If the Radius field is zero, the node does not forward the packet. This mechanism prevents a packet from circulating in the air forever.

The initial value of the Radius field is specified by the RAD[1:0] bits; the maximum number is 3.

3.5.2. Sequence Number

Every packet can be identified by the sender ID and the sequence number (SEQ). It helps to avoid retransmitting the same already-forwarded packet with lower RADIUS. All the nodes store the information (sender ID, sequence number of the packet, channel on which the packet has received) for the last few received packets in a table, called the Forwarded Packet Table. The number of the rows (number of managed packets) in this table is a parameter in the EZMacPRO_defs.h file, which can be changed before compiling the source code: FORWARDED_PACKET_TABLE_SIZE. This parameter has to be less than 16. EZMAC PRO manages the Forwarded Packet Table by itself; the next higher layer does not need to take care of that. EZMAC PRO handles the sequence numbering also. It is set to zero after Power On Reset and incremented every time a packet

transmission has initiated.

The node retransmits the packet if all of the following conditions are met:

- With non-zero Radius field.
- The destination address is different than the self address of the node.
- The packet forwarding feature is enabled.
- There is no entry in the Forward Packet Table with the sender ID, sequence number of the received packet.

After the node forwards the packet, it makes a new entry in the Forward Packet Table: it saves the sender ID, the sequence number of the packet, and the channel where the packet is received. The table management is very simple:

- If there is no space for the new entry, the oldest one will be replaced with the new entry.
- If there is another message in the table from the same sender ID, it will be replaced with the new entry. Only one entry is kept in the table from each node.

Packet forwarding is done automatically; however, the MAC calls the void EZMacPRO_Packet_Forwarding(void) callback function before the packet is forwarded; so, the upper software layer has the ability to modify the payload of the packet in order to add routing information, if needed. It is important to not change the CTRL and sender ID header bytes of the packet. During packet forwarding, EZMAC PRO is in one of the forward states:

- RX_STATE_FORWARDING_WAIT_FOR_TX
- RX_ERROR_FORWARDING_WAIT_FOR_TX

so the next higher layer can monitor the status of the process. It can stop the whole process by calling the EZMacPRO_Idle() function.

If packet forwarding is enabled, the node receives every packet, irrespective of the destination address. The Destination Address Filter is disabled during packet forwarding, but all other filters will be applied (only the final destination node applies the Destination Address Filter). If the packet is received, the state machine decides what the next step will be as follows:

- If the packet is sent to the receiver node, it enables all the active packet filters and checks the packet against them. If the packet passes the filters, the MAC notifies the higher layer about the packet reception. If required, the MAC sends back the acknowledgement.
- If the packet is sent to a different node, the receiver forwards the packet (if all the criteria are passed for packet forwarding).

Every message is forwarded on the same frequency channel where it is received.

Notes:

1. The packet forwarding feature can be enabled or disabled before compiling the source code: if the PACKET_FORWARDING_SUPPORTED definition is enabled in the EZMacPRO_defs.h file, it is compiled into the source code; otherwise, some code space can be saved.
2. The packet forwarding feature can be used only if the extended packet format is selected. Otherwise, the compiler gives the error message, "Packet forwarding requires the extended packet configuration!".

3.6. Automatic Acknowledgement

EZMAC PRO supports automatic acknowledgement. If the transmitter wants to get feedback on whether the packet arrived at the destination, it needs to set the ACKRQ bit (this will then set the ACKRQ bit in the CTRL byte of the transmitted packet). After EZMAC PRO transmits the packet, it goes into Waiting For ACK state (TX_STATE_WAIT_FOR_ACK) and waits for the acknowledgement packet. If the receiver node receives the packet which was addressed to it and the ACKRQ bit is set in the control header byte, then it automatically generates an acknowledgement message and sends it back to the transmitter node.

The transmitter waits for the acknowledgement for a predefined time. The timeout is always defined by the actual network parameters: whether the packet forwarding is enabled or not, how the radius field is set, the actual data rate, etc. The timeout is handled by EZMAC PRO automatically. It indicates the result of the acknowledgement reception for the next higher layer in the following ways:

- It goes into a SLEEP, IDLE, or RECEIVING state (depending on the SATX[1:0] bits) after successful acknowledgement reception.

- It goes into an ACK Receiving Error state (TX_ERROR_NO_ACK) if the predefined timeout expires without receiving the acknowledgement packet.

The packet structure of the acknowledgement packet is special: the ACK bit is set in the control byte, and the data field is 0x00 (only one 0x00 data byte if variable packet length is used, or all data byte 0x00 if fixed packet length is used). Also, the source and destination addresses are swapped.

Every message is acknowledged on the same frequency channel where it is received.

An acknowledgement packet is not sent if the destination address of the received packet is the broadcast address.

Notes:

1. The automatic acknowledgement feature can be enabled or disabled before compiling the source code. If the AUTOMATIC_ACK_SUPPORTED definition is enabled in the EZMacPRO_defs.h file, the feature is compiled into the source code; otherwise, some code space can be saved.
2. The automatic acknowledgement feature can be used only if the extended packet format is selected; otherwise, the compiler gives the error message, "Automatic acknowledgement requires the extended packet configuration!".

3.7. Signal Strength Level Indication

The EZRadioPRO devices are capable of measuring the receiver signal strength (RSSI) level. EZMAC PRO uses this information during the frequency search mechanism and for the listen-before-talk mechanism. However, the input signal level can also be very useful for the higher layers if the current consumption is important. The application can monitor the RSSI level during packet reception. If the signal level is high enough, the receiver can inform the transmitter node to lower the output power. This way, a lot of current can be saved at the transmitter side.

Reading the RSSI register during receive mode, EZMAC PRO always reads the actual RSSI value from the radio and updates the RSSI register accordingly. If EZMAC PRO is not in receiving mode, the RSSI register holds the last RSSI information:

- If the higher layer does not read the RSSI register during the last packet reception, the RSSI register contains the RSSI value taken when the chip received the synchron word.
- If the higher layer read the RSSI register during packet reception, the RSSI register contains the last time-measured value.

3.8. Low Frequency Periodic Timer Support

EZMAC PRO can provide periodic, low-frequency time base for the next higher layer. It works based on the Wake Up Timer of the EZRadioPRO radio. The Wake Up Timer can have two clock sources: the internal RC oscillator or the external 32.768 kHz crystal.

If the timer is enabled, EZRadioPRO periodically calls the void EZMacPRO_LFTimerExpired(void) callback function with the predefined time interval. The body of the callback function is empty; it has to be defined by the next higher layer. The callback function is always called from the external interrupt routine; so, it has to be considered when defining the functionality of the callback function (keep the processing time of the callback function as low as possible).

The accuracy of the time base depends on the selected oscillator option (IETB bit), but the actual frequency of the timer is 32.768 kHz. The time period (how often the callback function is called) is defined by the LFTMR 0 …2 registers. The low frequency timer is automatically set up and started once the LFTMRE bit is changed from 0 to 1. The time period (LFTMR0…1 registers) shall be set before the enabling the timer.

One design consideration is which clock source is used. The internal RC-based or external 32.768 kHz crystal-based timer has the following advantages/disadvantages:

- The crystal has better accuracy compared to the RC-based timer. A longer time period can be delayed with a smaller error. Due to the smaller error (drift between the devices), the nodes can be in sleep mode for a longer time period; so, they can save battery power.
- However, the better accuracy requires an external crystal, which is an extra bill of materials cost. In this case, GPIO0 has to be used for connecting the crystal to the radio; so, it cannot be used for other purposes.

If the internal RC oscillator is selected as a clock source, EZMAC PRO configures GPIO0 according to the predefined feature selected by the GPIO0_FUNCTION definition in the EZMacPRO_defs.h. If the external 32.768 kHz crystal is selected, EZMAC PRO configures GPIO0 as input pin for the crystal.

3.9. Low Battery Detection

EZMAC PRO can monitor the power supply voltage of the radio (using the built-in Low Battery Detect circuit of the radio).

- It can measure and provide the actual power supply voltage of the radio as a 5-bit digital number (VBAT[4:0] bits in the LBDR register). The voltage can be calculated by the following formula:
  V_{Supply} = (1.675 + VBAT[4:0] x 50 mV) ± 25 mV
- The upper software layer can define a power supply voltage threshold (LBDT[4:0] bits in the LBDR register) according to the following formula:
  $V_{Threshold}$ = (1.675 + LBDT[4:0] x 50 mV) ± 25 mV
  If the power supply goes below the threshold, EZMAC PRO calls the void EZMacPRO_LowBattery(void)callback function. The low battery detect function can be enabled by setting the LBDE bit.

3.10. Transmitted Byte Counters

According to the European standard (ETSI), the nodes can occupy the channel for 0.1% or 10% of the time (depending on the sub-band and channel space), measured for the last 1 hour. EZMAC PRO has a counter for all the frequency channels that counts the already-transmitted bytes for each channel separately. These counters can be used by the higher software layer to calculate the duty cycle of the channel occupancy. The higher layer can read the number of transmitted bytes, and, knowing the data rate, it can easily calculate the duty cycle. The counters are not cleared by EZMAC PRO automatically (only during initialization); the next software layer has to do it periodically.

The Byte Counter feature is optional; if the ENABLE_BYTE_COUTERS definition is enabled in the EZMacPRO_defs.h, the function is available, and the counter registers are implemented; otherwise, 8 bytes of RAM and some code space can be saved.

Note: The byte counters can be read and cleared if the MAC is not in receive or transmit mode.

4. EZMAC PRO Operation

4.1. Overview

EZMAC PRO is implemented as a state machine, running in two interrupt routines. Timer IT is used for different timing purposes, and an external IT handles the interrupt requests of the radio chip.

The behavior of EZMAC PRO is determined by a set of parameters stored in different registers. The upper software layers can interact with the software module via commands implemented as C functions.

A simplified flow chart is shown in Figure 8.

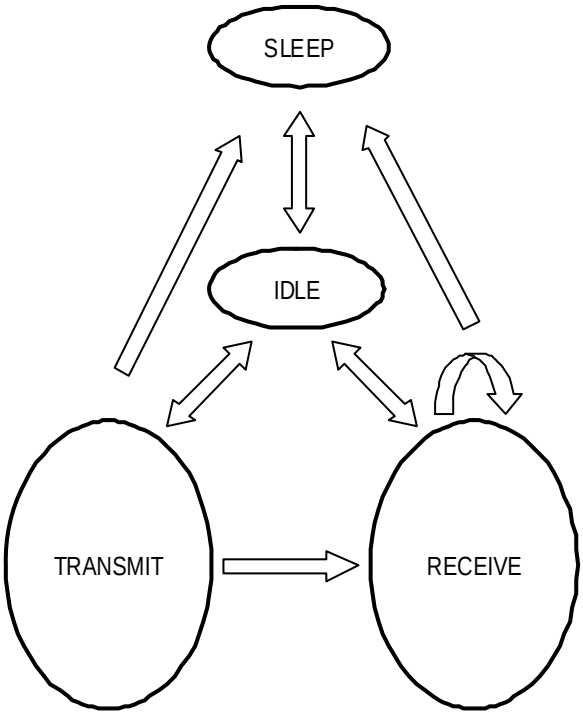

Figure 8. Simplified Block Diagram

4.1.1. SLEEP Mode

After initialization, EZMAC PRO is in SLEEP mode. In this mode, the RF hardware is completely switched off and consumes less than 1 µA. The MAC should be woken up at least 1 ms before the start of any transmission or reception. This time is the maximum time needed by the crystal oscillator to achieve stability.

4.1.2. IDLE Mode

After waking up EZMAC PRO with the wakeup command (MacParams EZMacPRO_Wake_Up(void) function), it switches to IDLE mode. In this mode, the MAC is waiting for further commands. The crystal oscillator is running in the radio, but all RF blocks are disabled, and the current consumption of the chip is about 0.6 mA. EZMAC PRO stays in this mode until it receives a RECEIVE, TRANSMIT, or SLEEP command.

4.1.3. TRANSMIT Mode

Before sending a TRANSMIT command, the data to be sent and its destination (address) must be loaded into the appropriate registers of EZMAC PRO (it can also be done in sleep mode). After the transmit command has been sent, the MAC starts transmission. Listen before talk is performed before each packet transmission if the ENLBT bit is set; otherwise, the packet is sent without the channel access mechanism.

If the automatic acknowledgement feature is enabled, after sending a packet, EZMAC PRO automatically waits for the acknowledgement. If the acknowledgement packet arrives within timeout, the software automatically goes into the selected next state (IDLE, SLEEP or RECEIVE).

If timeout occurs without receiving the acknowledgement packet, the software goes into TX_ERROR_NO_ACK state and waits for the higher layer for the next command (this state is similar to the IDLE state, only the crystal oscillator of the radio is running so it consumes 0.6 mA).

EZMAC PRO supports the following communication scenarios:

- The transmitter sends the packet on one channel and it does not wait for any feedback from the receiver node, whether the packet received correctly or not. This method uses less current, but the link is not robust; the transmitter just sends the packet once, and it does not have information about the result. This method can also be used between receive and transmit-only devices.
- The transmitter sends the packet on one channel and waits for acknowledgement from the receiver node. If the acknowledgement does not arrive, it retransmits the packet (possibly on a different channel to increase the robustness). This method is very robust since the transmitter always has information about the receiver, but uses more current at both sides of the link.
- If the AFCH bit is set, the very same packet is transmitted automatically on all the enabled channels next to each other. This very simple method can be useful for such applications when the receiver current consumption is important or the receiver node cannot send back acknowledgement (receiver-only device), but the robustness is critical. The transmitter sends the same packet on more frequencies to increase the robustness and the probability that the receiver receives at least one of the packets. The receiver searches on all channels and tries to receive at least one of the packets. It is not intended to receive all the packets. To save battery power, the receiver does not send back an acknowledgement.

Note: If the AFCH bit is set and the automatic acknowledgement feature is enabled, the transmitter will not wait for the acknowledgement (clears the ACKRQ bit in the packet); it immediately sends the same packet on the next channel.

Finishing the transmission(s), EZMAC PRO automatically goes into one of the following states, depending on the SATX[1:0] bits: IDLE, SLEEP, RECEIVE.

4.1.4. RECEIVE Mode

If EZMAC PRO gets a receive command during IDLE mode, it will change into the RECEIVE mode. The MAC scans the available frequencies for a valid data transmission. Once a valid data packet has been received and passes all the enabled error detection and address filters, the MAC turns the radio into power saving mode and waits for the upper software layer to read out the received data. After the data has been read, the MAC goes into one of the selected modes: IDLE, SLEEP, or RECEIVE.

If the packet forwarding feature is enabled and the node has to forward the packet, then it decreases the radius field of the packet and performs a listen-before-talk mechanism. If the listen-before-talk mechanism shows that the channel is free, it forwards the packet. If the channel is occupied, it performs another listen-before-talk mechanism. The node repeats the listen-before-talk mechanism up to MAX_LBT_RETRY times, which is a definition in the EZMacPRO_defs.h file. If the node was not able to forward the packet after the maximum retry number, it drops the packet and automatically goes back into the RECEIVE state.

If the receiver has to acknowledge the received packet, EZMAC PRO automatically sends back the acknowledgement without performing a listen before talk before the packet. After finishing the acknowledgement transmission, EZMAC PRO automatically goes into the selected next mode (IDLE, SLEEP, or RECEIVE).

4.2. Packet Filtering Method

There is an intelligent packet filtering feature implemented in EZMAC PRO. The filter can be configured by several options to best fit the demands of the upper software layers. Once EZMAC PRO is configured, the upper layers will send and receive only the payloads. EZMAC PRO takes care of all the packet building and unpacking tasks. The radio runs CRC; the upper layer gets only error-free packets.

The flow chart of packet filtering is shown in Figure 9.

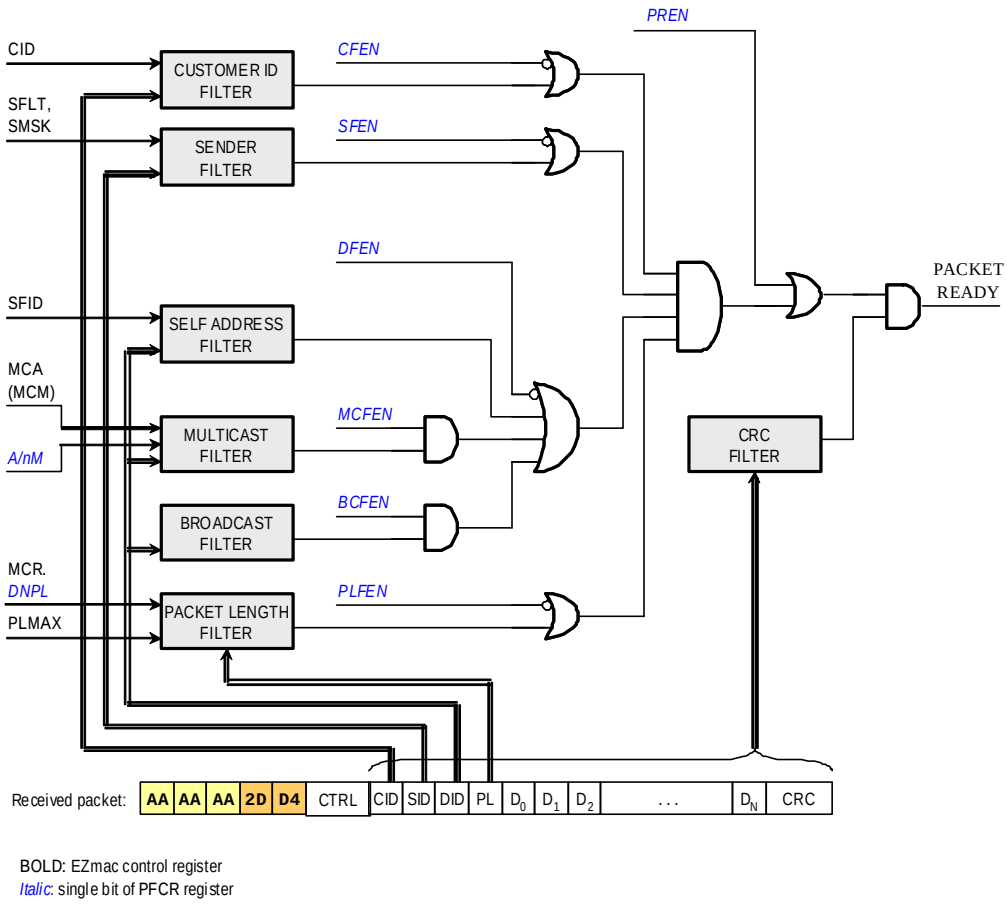

Figure 9. Packet Filtering

4.2.1. Real Time Operation

Packet filtering runs in real time during reception. This means that as the packet header is received, the filter will check the validity of the header bytes. If any segment of a packet header fails in the corresponding filter, packet reception will be aborted. This technique enables EZMAC PRO to ignore unexpected invalid packets at a very early stage in the reception process. After aborting a packet reception, EZMAC PRO will keep on searching for a valid data packet by hopping to the next available frequency. Real-time packet filtering saves resources (processor power and battery life) and may significantly increase the performance of the system by reducing resources wasted for invalid packet processing while missing important packets present on other frequencies.

4.2.2. Customer ID Filter

CID is a unique ID for each customer using the EZMAC PRO protocol. Using CID is optional but strongly recommended to avoid unexpected interactions between different systems using EZMAC PRO that may be installed within the same area. If CID is used, this will be transmitted shortly after the preamble and synchron pattern. EZMAC PRO can recognize the beginning of the reception as to whether it is an in-system packet or not. If not, the MAC will cancel reception and hop to the next frequency to search for a valid packet. Customers can request a CID from Silicon Labs.

4.2.3. Sender Filter

EZMAC PRO filters the incoming packet by testing the sender ID field in the header of the received packet. If the Sender Filter is enabled, it will accept packets only from a specific node (if SMSK is set to 0xFF) or group of nodes (SMSK can be used to mask out bits that are not relevant by setting them to “0”). The group is defined by masking the important bits of the SID field of the packet header and the same bits of the SFLT register.

Packets will pass the filter if:

SFLT & SMSK == SID & SMSK

Where:

&: Bitwise AND operator

SID: sender ID field in the header of the received packet

Setting SMSK to 0xFF, the group filtering option is not used; so, the MAC will accept packages only from the node having the SFLT address.

4.2.4. Destination Filter

The destination filter consists of 3 address filters, with all of them testing the DID byte of the received packet header:

- Self address filter
- Multicast address filter
- Broadcast address filter

EZMAC PRO performs all the enabled Destination Filters on the received packet. If the packet passes any of the enabled filters, it will also pass the destination filter.

4.2.4.1. Self Address Filter

Self address (SFID) is used to uniquely identify EZMAC PRO within a communication network. Only those packets whose DID field in the header equals SFID will pass the self address filter.

4.2.4.2. Multicast Address Filter

Multicast address checks whether the received packet is dedicated to a group of nodes and if EZMAC PRO is a member of this group. There are two ways of multicast addressing:

- Defining a special address called multicast address (MCA). Only those packets whose DID field in the header equals MCA will pass this multicast filter.
- Defining a multicast mask (MCM) and using it at self address filtering:

DID & MCM == SFID & MCM

where & is the bitwise AND operator

Only one of the MCA and MCM filters can be active at the same time. The actual mode can be selected by the A/nM bit of the PFCR register.

4.2.4.3. Broadcast Address Filter

Broadcast address is a special address: 0xFF.

Only those packets whose DID field in the header is 0xFF will pass the broadcast filter.

Note: The complete destination filter can be enabled / disabled by the DFEN bit of the Packet Filter Control Register. If this bit is cleared, the destination filtering will be disabled, and packets with any destination address will be received.
If the Packet Forwarding feature is used, the Destination Address Filter is disabled during forwarding. It is applied only on the destination node.

4.2.5. Promiscuous Mode

Enabling the promiscuous mode by setting the PREN bit of the PCFR register, all the address filters and the packet length filter will be ignored.

4.2.6. CRC Filter

This is the only filter that cannot be ignored. This filter performs CRC on all the bytes of the packet (including the packet header). If the received packet fails the CRC, EZMAC PRO will ignore the packet.

4.3. Error Detection Method

EZMAC PRO can detect several types of errors, which helps the upper software layer to select the best data transmission strategy. EZMAC PRO error detection has a separate 8-bit error counter for each frequency channel and a common control register. The control register is used to enable/disable the different types of error detections, and the counters simply count the enabled errors separately for each frequency channel. The counters do not overflow and can be cleared by the upper software layer.

4.3.1. Channel is Busy (Collision)

If the Listen before Talk (LBT) feature and the Channel Busy error counter are enabled, EZMAC PRO will increment the error counter if the listen-before-talk mechanism shows an occupied channel.

4.3.2. Bad CID

Using the Customer ID in the packet header, EZMAC PRO is able to detect in the early state of a packet reception if the packet is an in-system packet or a packet transmitted by third-party product using the EZRadioPRO chipset. As the CID is one of the first bytes of the packet header, EZMAC PRO can abort the reception quickly and continue to scan the available frequencies for a valid packet.

4.3.3. Bad Address

If the address bytes fail on the address filter logic, EZMAC PRO detects a bad address error, aborts the reception, and continues to scan the available frequencies for a valid packet.

4.3.4. Bad CRC

If a received packet does not pass the Cyclic Redundancy Code (CRC) check, EZMAC PRO will create a bad CRC error and discard the packet. Scanning for a valid packet will continue. This type of error gives useful information about the quality of the RF link.

A high number of CRC errors will indicate a low-quality link. Detecting other types of errors but only a few CRC errors indicates a good link, but a bad communication strategy:

- Significant third-party transmissions
- Significant in-system collisions

4.4. State Machine

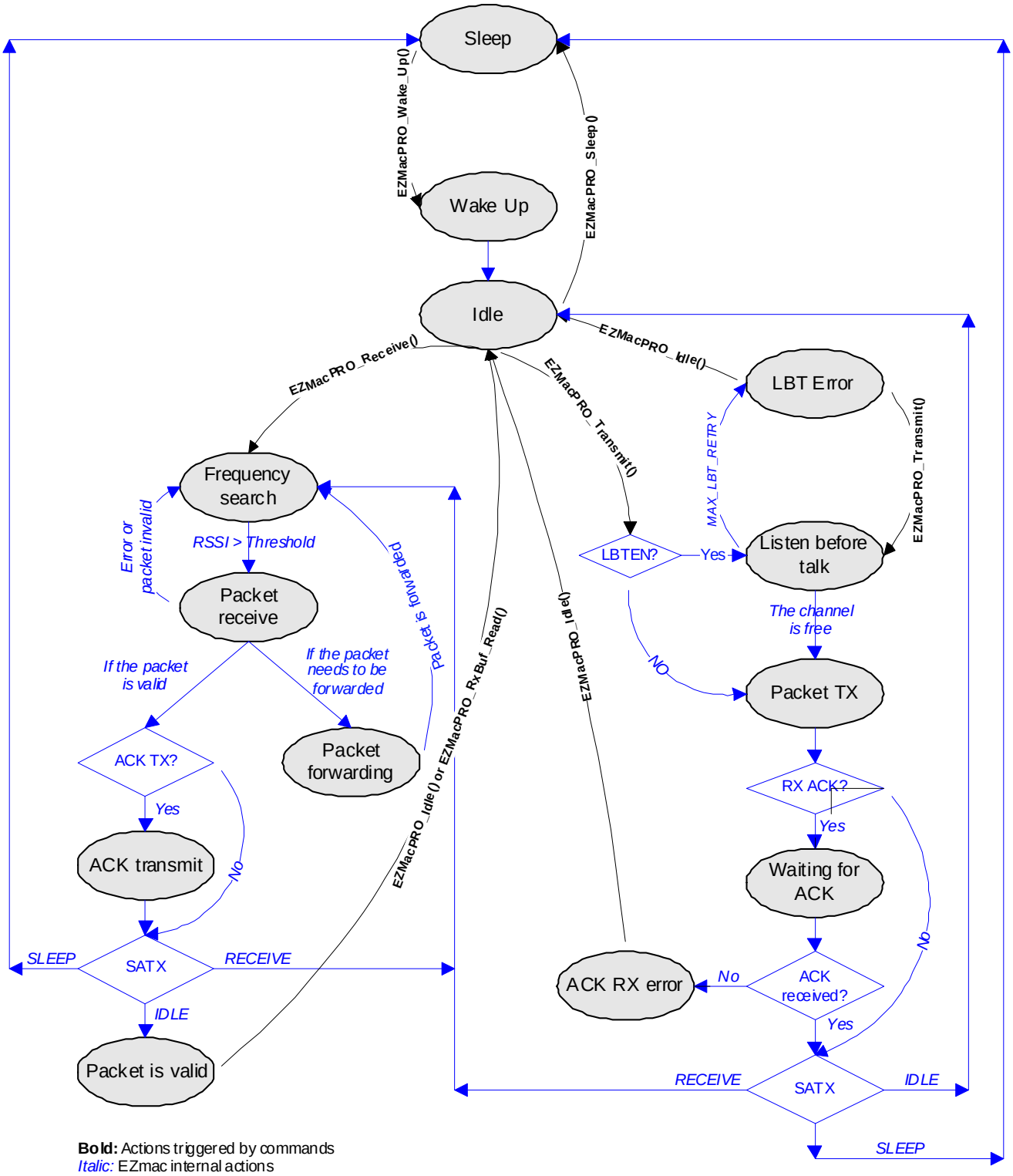

Figure 10. State Machine

5. EZMAC PRO Application Programming Interface

5.1. EZMAC PRO Commands

The upper software layer can interact with EZMAC PRO through function calls. This section gives detailed information about the Application Programming Interface.

There is a function for initializing the EZRadioPRO and a separate function for initializing the microcontroller. These functions must be called in the power-on initializing routines:

- *void EZMacPRO_Init(void)*
- *void MCU_Init (void)*

During normal operation, EZMAC PRO uses the following functions to interact with the upper software layers:

- *MacParams EZMacPRO_Wake_Up(void)*
- *MacParams EZMacPRO_Sleep(void)*
- *MacParams EZMacPRO_Idle(void)*
- *MacParams EZMacPRO_Transmit(void)*
- *MacParams EZMacPRO_Receive(void)*
- *MacParams EZMacPRO_Reg_Write(MacRegs name, U8 value)*
- *MacParams EZMacPRO_Reg_Read(MacRegs name, U8* value)*
- *MacParams EZMacPRO_TxBuf_Write(U8 length, U8* payload)*
- *MacParams EZMacPRO_RxBuf_Read(U8* length, U8* payload)*

EZMAC PRO can provide feedback for the next higher layer by calling callback functions. The following callback functions are available:

- *void EZMacPRO_LFTimerExpired(void)*
- *void EZMacPRO_LowBattery(void)*
- *void EZMacPRO_WokeUp(void)*
- *void EZMacPRO_WokeUp_Error(void)*
- *void EZMacPRO_SyncWordReceived(void)*
- *void EZMacPRO_PacketReceived(U8 rssi)*
- *void EZMacPRO_PacketForwarding(void)*
- *void EZMacPROTX_ErrorLBT_Timeout (void)*
- *void EZMacPRO_PacketSent(void)*
- *void EZMacPROTX_ErrorNoAck (void)*

The callback functions are empty functions called from interrupt routines by the state machine. The body of the function is empty; it has to be realized according the higher layer needs. All these functions are called from interrupt routine; so, the processing time has to be kept as short as possible.

The MAC uses only a few resources of the microcontroller:

- The SPI1 peripheral
- The Timer0
- An external interrupt source

These peripherals are handled by EZMAC PRO; the upper SW layer does not need to take care of them. The Timer0 is used for different timing purposes, while the external IT is used to serve the interrupt requests of the EZRadioPRO devices. The MAC engine is implemented as a state machine and runs in these functions:

- *INTERRUPT(timerIntT0_ISR, INTERRUPT_TIMER0)*
- *INTERRUPT(externalIntISR, INTERRUPT_INT0)*

There are no other restrictions on the structure of the program. The foreground loop (the main() function) can be completely utilized by the application.

5.1.1. void EZMacPRO_Init(void)

Function: Initializes the EZRadioPRO device.

Requires: The function has to be called in the power-on initializing routine.

5.1.2. void MCU_Init (void)

Function: Initializes the used peripherals of the microcontroller (SPI1, Interrupts, and Timer0).

Requires: The function has to be called in the power-on initializing routine.

5.1.3. MacParams EZMacPRO_Wake_Up(void)

Function: Switches the MAC from SLEEP mode into IDLE mode. Turns on the crystal oscillator of the radio, so the current consumption increases.

Returns: **MAC_OK**: the operation was successful.
STATE_ERROR: the operation is ignored because the MAC was not in SLEEP mode.

Requires: The MAC has to be in SLEEP mode when calling the function.

5.1.4. MacParams EZMacPRO_Sleep(void)

Function: Switch the MAC from IDLE to SLEEP mode. It turns off the crystal oscillator on the radio.

Returns: **MAC_OK**: the operation was successful.
STATE_ERROR: the operation is ignored because the MAC was not in IDLE mode.

Requires: The MAC has to be in IDLE mode.

5.1.5. MacParams EZMacPRO_Idle(void)

Function: This is the only function that aborts the ongoing reception and transmission. It could be used to reset the state of the MAC. Calling this function, EZMAC PRO goes into IDLE state.

Returns: **MAC_OK**: the MAC is set into IDLE mode, and no transmission or reception was aborted.
STATE_ERROR: the MAC is set into IDLE mode, and transmission or reception was aborted.

Requires: The function cannot be called in SLEEP mode.

5.1.6. MacParams EZMacPRO_Transmit(void)

Function: It starts to transmit a packet.

Returns: **MAC_OK**: the transmission started correctly.
STATE_ERROR: the operation ignored (the transmission has not been started), because the EZMAC PRO was not in IDLE mode.

Requires: All the parameters have to be set before calling this function. EZMAC PRO has to be in IDLE mode when calling this function.

5.1.7. MacParams EZMacPRO_Receive(void)

Function: It starts searching for a new packet on the defined frequencies. If the receiver finds RF activity on a channel, it tries to receive and process the packet. The search can be stopped by the EZMacPRO_Idle() function.

Returns: **MAC_OK**: the search mechanism started correctly.
STATE_ERROR: the operation is ignored (the search mechanism has not been started) because the EZMAC PRO was not in IDLE mode.

Requires: All the parameters have to be set before calling this function. EZMAC PRO has to be in IDLE mode when calling this function.

5.1.8. MacParams EZMacPRO_Reg_Write(MacRegs name, U8 value)

Function: Writes the **value** into the register identified by **name**. MacRegs type is predefined; the names of the available registers are listed in the Register Assignment section. When required, this function also updates the radio register setting directly. This function is also available to be called during SLEEP mode.

Returns: **MAC_OK**: The register is set correctly.
NAME_ERROR: The register name is unknown.
VALUE_ERROR: The **value** is “out of the range: for this register.
STATE_ERROR: The requested operation is currently unavailable and ignored because either transmission or reception is currently in progress.
INCONSISTENT_SETTING: If the data rate is changed and the current Frequency ID is not supported by the new data rate, the value of the inconsistent Frequency ID is automatically changed to a 0. In this case, the function returns with 'INCONSISTENT_SETTING'.
See "5.2. EZMAC PRO Registers" on page 32 for more details.

Requires: None of the registers can be set during packet transmission or reception!
See “5.2. EZMAC PRO Registers” for more details.

5.1.9. MacParams EZMacPRO_Reg_Read(MacRegs name, U8* value)

Function: Gives back the value (over value pointer) of the register identified by **name**. *MacRegs* type is predefined; the name of the available registers are listed in “5.2. EZMAC PRO Registers”. This function may also be called in SLEEP mode.

Returns: **MAC_OK**: The operation was successful.
NAME_ERROR: The register name is unknown.

Requires: Nothing.

5.1.10. MacParams EZMacPRO_TxBuf_Write(U8 length, U8* payload)

Function: The function copies **length** number of **payload** bytes into the transmit FIFO of the radio chip. There is no dedicated transmit buffer in the source code.
Upon calling this function, it clears the TX FIFO of the radio first.
If variable packet length is used and the **length** is not greater than the RECEIVED_BUFFER_SIZE definition (it cannot be greater than 64), then EZMAC PRO copies **length** number of **payload** bytes into the TX FIFO of the radio, sets the PLEN register of EZMAC PRO, and also sets the packet length register of the radio. If fix packet length is used, then the PLEN register has to set first, because the function copies only Payload Length number of bytes into the FIFO even if the **length** is greater. Also it fills the FIFO with extra 0x00 bytes if the **length** is smaller than the value of the Payload Length register in fix packet length mode.

Returns: **MAC_OK**: The operation performed correctly.
STATE_ERROR: The operation is ignored because transmission and reception are in progress.

Requires: The function cannot be called during transmission and reception.

5.1.11. MacParams EZMacPRO_RxBuf_Read(U8* length, U8* payload)

Function: After a successful packet reception EZMAC PRO copies the received data bytes into the receive data buffer. The receive data buffer is declared in the EZMacPro.c file as SEGMENT_VARIABLE(RxBuffer[RECEIVED_BUFFER_SIZE], U8, BUFFER_MSPACE);
The length of the receive buffer is defined by the RECEIVED_BUFFER_SIZE definition in the EZMacPro_defs.h. It can be adjusted for the application needs, but it can not be greater than 64bytes. The receive buffer is declared to be placed into the XDATA memory, it also can be adjusted by changing the BUFFER_MSPACE definition. Upon calling the EZMacPRO_RxBuf_Read() function, it copies received data from the receive data buffer to **payload**. Also it gives back the number of received bytes by length.

Returns: **MAC_OK**: The operation performed correctly.
STATE_ERROR: The operation is ignored because reception and transmission is in progress.

Requires: None.

5.1.12. void EZMacPRO_LFTimerExpired(void)

Function: The callback function is called if the low-frequency timer expired. It is called from the external interrupt routine.

Returns: None.

Requires: The timer has to start by setting the time period in the *LFTMR0* ... 3 registers.

5.1.13. void EZMacPRO_LowBattery(void)

Function: The callback function is called when the Low Battery Detector circuit in the radio detects that the power supply is below the *LBDT* threshold. The function is called from the external interrupt routine.

Returns: None.

Requires: None.

5.1.14. void EZMacPRO_WokeUp(void)

Function: The callback function is called after EZMAC PRO performs the SLEEP to IDLE state transition correctly and the crystal of the radio is stabilized. The function is called from the external interrupt routine.

Returns: None.

Requires: None.

5.1.15. void EZMacPRO_WokeUp_Error(void)

Function: The callback function is called when the SLEEP to IDLE transition does not finish within timeout (the crystal of the radio did not stabilize within timeout). After EZMAC PRO calls this callback function, it goes into wake up error state, called WAKE_UP_ERROR. This state is equivalent with SLEEP state. The function is called from the timer interrupt routine.

Returns: None.

Requires: None.

5.1.16. void EZMacPRO_SyncWordReceived(void)

Function: The callback function is called when the synchron word of the packet is received. This is a fix time in each packet and can be used for time synchronization. The function is called form the external interrupt routine.

Returns: None.

Requires: None.

5.1.17. void EZMacPRO_PacketReceived(U8 rssi)

Function: The callback function is called when EZMAC PRO receives a valid packet. The input parameter (**rssi**) of
the callback function is the RSSI of the received packet measured after the synchron word is received. The function is called from the external interrupt routine.

Returns: None.

Requires: None.

5.1.18. void EZMacPRO_PacketForwarding(void)

Function: The callback function is called when EZMAC PRO receives a packet that has to be forwarded. Prior to the packet forwarding, the upper software layer has the ability to read and change the content of the payload. It provides the possibility that the upper software layer adds routing information into the packet if needed. All other necessary modification on the packet (e.g. decreasing the radius field) and packet forwarding management is done by the MAC itself. The function is called from the external interrupt routine.

Returns: None.

Requires: None.

5.1.19. void EZMacPROTX_ErrorLBT_Timeout (void)

Function: The callback function is called if the Listen Before Mechanism failed to access the channel after the maximum number of retry times (defined by the MAX_LBT_RETRIES definition). After EZMAC PRO calls this callback function, it goes into LBT error state, called TX_ERROR_CHANNEL_BUSY. This state is equivalent to the IDLE state. The function is called from the timer interrupt routine.

Returns: None.

Requires: None.

5.1.20. void EZMacPRO_PacketSent(void)

Function: The callback function is called after EZMAC PRO transmits a packet correctly. If acknowledgement is also requested, the callback function is called after the acknowledgement has arrived. The function is called from the external interrupt routine.

Returns: None.

Requires: None.

5.1.21. void EZMacPROTX_ErrorNoAck (void)

Function: The callback function is called if EZMAC PRO did not receive the acknowledgement packet within timeout. After this callback function is called, EZMAC PRO goes into ACK error state, called TX_ERROR_NO_ACK. This state is equivalent to the IDLE state. The function is called from the timer interrupt routine.

Returns: None.

Requires: None.

Note: The enumeration of the return parameters (U8 type) can be found in the EZMacPro.h file, so the upper software layer can refer to them by their name.

5.2. EZMAC PRO Registers

5.2.1. Register Summary

Table 9. Register Summary

R/W	Name	Description	Default
R/W	MCR	Master Control Register	0x1C
R/W	SECR	State & Error Counter Control Register	0x50
R/W	TCR	Transmit Control Register	0x38
R/W	RCR	Receiver Control Register	0x04
R/W	FR0	Frequency Register 0	0x00
R/W	FR1	Frequency Register 1	0x01
R/W	FR2	Frequency Register 2	0x02
R/W	FR3	Frequency Register 3	0x03
R/W	EC0	Error Counter of Frequency 0	0x00
R/W	EC1	Error Counter of Frequency 1	0x00
R/W	EC2	Error Counter of Frequency 2	0x00
R/W	EC3	Error Counter of Frequency 3	0x00
R/W	PFCR	Packet Filter Control Register	0x02
R/W	SFLT	Sender ID Filter	0x00
R/W	SMSK	Sender ID Filter Mask	0x00
R/W	MCA/MCM	Multicast Address / Multicast Mask	0x00
R/W	MPL	Maximum Packet Length	0x40
R	MSR	MAC Status Register	0x00
R	RSR	Receive Status Register	0x00
R	RSSI	Received Signal Strength Indicator	0x00
R/W	SCID[1]	Self Customer ID	0xCD
R/W	SFID	Self ID	0x01
R	RCTRL	Received Control Byte	0x00
R	RCID[1]	Received Customer ID	0x00
R	RSID	Received Sender ID	0x00
R/W	DID	Destination ID	0x00

Notes:

1. Optional registers; these are implemented only if the customer ID support is compiled into the source code (CID_IS_USED definition).
2. Optional registers; these are implemented only if the byte counter feature is compiled into the source code (ENABLE_BYTE_COUNTERS definition).

Table 9. Register Summary (Continued)

R/W	Name	Description	Default
R/W	PLEN	Payload Length	0x01
R/W	RSSILR	RSSI Limit Register	0x32
R/W	LBTIR	Listen Before Talk Interval Register	0x8A
R/W	LBTLR	Listen Before Talk Limit Register	0x78
R/W	BCH0[2]	Byte Counter Low of Frequency 0	0x00
R/W	BCL0[2]	Byte Counter High of Frequency 0	0x00
R/W	BCH1[2]	Byte Counter Low of Frequency 1	0x00
R/W	BCL1[2]	Byte Counter High of Frequency 1	0x00
R/W	BCH2[2]	Byte Counter Low of Frequency 2	0x00
R/W	BCL2[2]	Byte Counter High of Frequency 2	0x00
R/W	BCH3[2]	Byte Counter Low of Frequency 3	0x00
R/W	BCL3[2]	Byte Counter High of Frequency 3	0x00
W	LFTMR0	Low Frequency Timer Setting Register 0	0x00
W	LFTMR1	Low Frequency Timer Setting Register 1	0x00
W	LFTMR2	Low Frequency Timer Setting Register 2	0x40
R/W	LBDR	Low Battery Detect Register	0x14
R/W	ADCTSR	ADC and Temperature Sensor Register	0x00
R/W	ADCTSV	ADC / Temperature Value Register	0x00

Notes:

1. Optional registers; these are implemented only if the customer ID support is compiled into the source code (CID_IS_USED definition).
2. Optional registers; these are implemented only if the byte counter feature is compiled into the source code (ENABLE_BYTE_COUNTERS definition).

Note: The structure of the EZMAC PRO registers is declared in the EZMacPro.h files. The availability of the registers is depending on the configuration of the EZMAC PRO. Therefore it is highly suggested to reference to the registers with their name and not with their address.

5.2.1.1. Master Control Register

MCR							
7	6	5	4	3	2	1	0
DR[2:0]			RAD[1:0]		DNPL	NRF[1:0]	

DR[2:0] Data rate. This bit selects the actual data rate. If these bits are changed, EZMAC PRO automatically updates the radio chip with the correct settings for the given data rate.

Table 10. Data Rate and Deviation Settings

DR[0:1]	Data Rate	Deviation
0	2.4 kbps	±38.4
1	4.8 kbps	±38.4
2	9.6 kbps	±38.4
3	10 kbps	±40
4	20 kbps	±20
5	50 kbps	±25
6	100 kbps	±50
7	128 kbps	±64

Note: If the data rate is changed and the previously selected Frequency ID is not supported by the new data rate, the value of the inconsistent Frequency ID is automatically changed to 0. In this case, the EZMacPRO_Reg_Write() function returns with INCONSISTENT_SETTING.

RAD[1:0] Radius. The radius defines how many times the packet can be forwarded. This control field only has an effect if the packet forwarding feature is compiled into the source code.

DNPL Dynamic packet length. If this bit is set, the payload length field is included in the transmitted packet, and the receiver expects to receive the payload length field in the header of the incoming packet; otherwise, the payload length field is not transmitted, and fix packet length will be used.

NRF[1:0] Number of used frequencies. The hopping system will use the first NRF frequencies defined in the Frequency registers (0 means only FR0 is used; 1 means FR0 and FR1 can be used, etc.).

5.2.1.2. State and Error Counter Control Register

SECR							
7	6	5	4	3	2	1	0
SATX[1:0]		SARX[1:0]		CB	BCID	BADDR	BCRC

SATX[1:0] State After Transmit. These bits define the next state after EZMAC PRO finishes a successful packet transmission:

- If the automatic acknowledgement feature is disabled, EZMAC PRO performs the automatic state changes if the packet was transmitted.
- If the automatic acknowledgement feature is enabled, EZMAC PRO performs the automatic state changes only after it receives the acknowledgement packet.

Table 11. SATX Settings

SATX[0:1]	Next State
0	SLEEP
1	IDLE
2	RECEIVE
3	Reserved

SARX[1:0] State After Receive. These bits define the next state after EZMAC PRO successfully receives a packet:

- If the automatic acknowledgement feature is disabled, EZMAC PRO performs the automatic state changes after packet reception.
- If the automatic acknowledgement feature is enabled, EZMAC PRO performs the automatic state changes only after the acknowledgement packet is transmitted correctly.
- If EZMAC PRO forwards a packet, it always goes back to receiving mode.

Table 12. SARC Settings

SARX[0:1]	Next State
0	SLEEP
1	IDLE
2	RECEIVE
3	Reserved

CB Channel Busy. If this bit is set, the error counters count the channel busy errors.

BCID Bad Customer ID. If this bit is set and the customer ID support is compiled into the source code, then the error counters count the bad customer ID errors. If the customer ID is not compiled into the source code, this bit does not have any effect.

BADDR Bad Addresses. If this bit is set, the error counters count the bad address errors.

BCRC Bad CRC. If this bit is enabled, the error counters count the bad CRC errors.

Note: See "4.3. Error Detection Method" on page 25 for more details.

5.2.1.3. Transmit Control Register

ACKRQ Acknowledgement Request. If the automatic acknowledgement feature is compiled into the source code and this bit is set, the transmitter sets the ACKRQ bit in the Control Header byte asking the receiver to send back an acknowledgement packet.

OP[2:0] Output Power. These bits set the output power of the radio chip.

Table 13. Output Power Settings

Output Power		
OP[2:0]	**Si4432 / Si4032**	**Si4431 / Si4031**
0	11 dBm	–8 dBm
1	14 dBm	–5 dBm
2	17 dBm	–2 dBm
3	20 dBm	1 dBm
4	Reserved	4 dBm
5	Reserved	7 dBm
6	Reserved	10 dBm
7	Reserved	13 dBm

LBTEN Listen Before Talk Enable. If this bit is set, EZMAC PRO performs Listen Before Talk before all packet transmission. It will not be performed in the following cases even if this bit is set:

- Before EZMAC PRO transmits the automatic acknowledgement packet
- Before packet transmission if the AFCH bit is set.

AFCH Automatic Frequency Change. If this bit is set, the very same packet will be automatically transmitted on all enabled frequencies (first packet is sent on F0) next to each other without performing listen before talk even if the LBTEN bit is set.
If this bit is cleared, the packet is transmitted only once on the frequency defined by TF[1:0] bits.

TF[1:0] Transmit Frequency. These bits have an effect only if the AFCH bit is cleared. These bits define the frequency where the packet is transmitted.

5.2.1.4. Receive Control Register

PFEN Packet Forwarding Enable. If the packet forwarding feature is compiled into the source code and this bit is set, then EZMAC PRO forwards all packets that are supposed to be forwarded
(see "3.5. Packet Forwarding" on page 17 for more details).

FMASK3…0 Frequency Mask. If this bit is set, then the corresponding frequency is disabled for the frequency search mechanism. It can be used to temporarily extract one or more frequencies if significant interference appears in that channel(s).

Note: If the register is written in a way that all bits are set, then the MAC does not change the state of these bits and the EZMacPRO_Reg_Write() function returns with VALUE_ERROR.

SCHEN Search enabled. If this bit is set, then EZMAC PRO uses the frequency search mechanism and it searches on all enabled, not-masked frequencies for packets. After enabling the receiver, EZMAC PRO
starts looking for transmission on the first enabled channel. If this bit is cleared, EZMAC PRO waits for transmission on a frequency determined by RF[1:0].

RF[1:0] Receiving frequency. If the SCHEN bit is cleared, then this bit defines the frequency at which the receiver waits for transmission. If the SCHEN bit is set, this bit does not have any effect.

5.2.1.5. Frequency Registers

FRx							
7	6	5	4	3	2	1	0
Frequency ID of Channel x[7:0]							

F0…3

These registers set the channel number used as frequency 0…3. The number of available frequencies depends on the data rate (for higher data rates, the required bandwidth is higher; so, less frequency channel can be allocated). The actual center frequency is defined by three parameters: the start frequency, the frequency step, and the frequency ID. These parameters are different for different data rates and frequency bands. These parameters are defined in the EZMacPRO_defs.h file. The actual frequency is calculated as follows:

$$\text{center\_frequency} = \text{start\_frequency} + (F_x \times \text{frequency\_step})$$

where F_x is the frequency ID. The first channel is identified with $F_x = 0$ (e.g. if the maximum number of the channels is 4, then the valid range for the frequency ID is 0…3).

The default frequency assignment of EZMAC PRO for different frequency bands is the following (it can be modified for the application needs):

Table 14. Frequency Assignment for the 434 MHz ISM Band

434 MHz ISM band			
Data Rate [kbps]	**Start Frequency [MHz]**	**Frequency Step Size [kHz]**	**Number of Available Channels**
2.4	433.33	330	4
4.8	433.33	330	4
9.6	433.33	330	4
10	433.33	330	4
20	433.53	390	3
50	433.61	360	2
100	433.91	0	1
128	433.91	0	1

Table 15. Frequency Assignment for the 868 MHz ISM Band

868 MHz ISM band			
Data Rate [kbps]	**Start Frequency [MHz]**	**Frequency Step Size [kHz]**	**Number of Available Channels**
2.4	863.55	450	14
4.8	863.55	450	14
9.6	863.55	450	14
10	863.55	450	14
20	863.70	550	11
50	863.75	780	8
100	864	1MHz	6
128	864	1MHz	6

Table 16. Frequency Assignment for the 915 MHz ISM Band

915 MHz ISM band			
Data Rate [kbps]	**Start Frequency [MHz]**	**Frequency Step Size [kHz]**	**Number of Available Channels**
2.4	900.84	480	60
4.8	900.84	480	60
9.6	900.84	480	60
10	900.84	480	60
20	900.84	550	52
50	900.84	780	37
100	900.84	1MHz	29
128	900.84	1MHz	29

Note: If a greater value is written into the register than the maximum ID of the available frequencies, then EZMAC PRO ignores the setting, and the EZMacPRO_Reg_Write() function returns with VALUE_ERROR.
The default frequency allocations for all data rates and frequency bands are defined to comply with the ETSI and FCC standards.

5.2.1.6. Error Counters

ECx							
7	6	5	4	3	2	1	0
Error Counter of Channel x[7:0]							

Error Counter of channel 0…3.

EZMAC PRO can detect several types of errors during packet transmission and reception. If an error is detected and the corresponding error detection is enabled, the error counter of the actual operating frequency is incremented by 1. The error counters do not overflow the counting stops at 0xFF. The counters can be read and cleared by the upper software layer.

See "4.3. Error Detection Method" on page 25 for more details.

5.2.1.7. Packet Filter Control Register

The register controls the incoming packet filter of EZMAC PRO. See "4.2. Packet Filtering Method" on page 23 for more details.

CFEN If this bit is set, the Customer ID filter is enabled.

SFEN If this bit is set, the Sender filter is enabled.

DFEN If this bit is set, the Destination filter is enabled.

MCFEN If this bit is set, then multicast packets can also be received.

BCEN If this bit is set, then broadcast packages can also be received.

PLFEN If this bit is set, the Packet Length filter is enabled.

PREN If this bit is set, EZMAC PRO is set into Promiscuous mode. In this mode, all the filters are ignored, it receives all kind of packets with valid CRC.

A/nM Selects the mode of the Multicast Filter. 1: Multicast Address mode; 0: Multicast Mask mode.

5.2.1.8. Sender ID Filter, Filter Mask Registers

SFLT							
7	6	5	4	3	2	1	0
Sender ID Filter[7:0]							

SMSK							
7	6	5	4	3	2	1	0
Sender ID Filter Mask[7:0]							

EZMAC PRO uses these bytes for sender filtering. Packets will pass the filter if:

SFLT & SMSK == SID & SMSK

where:

- & - bitwise AND operation
- SID - sender ID header field of the received packet.

5.2.1.9. Multicast Address / Multicast Mask Register

MCA/MCM							
7	6	5	4	3	2	1	0
Multicasr Address[7:0] / Multicast Mask [7:0]							

The register works different way in Multicast Address and Multicast Mask mode. The selection between the two modes is done by the A/nM bit.

Multicast Address Mode (A/nM = 1)

EZMAC PRO filters the incoming packet by testing the destination ID field in the header of the received packet. If the Multicast Address Filter is enabled, EZMAC PRO will accept packets coming with MCA at the destination ID header field of the received packet.

Multicast Mask Mode (A/nM = 0)

EZMAC PRO filters the incoming packet by testing the destination ID header field of the received packet. If the Multicast mask filter is enabled, EZMAC PRO will accept packets if:

DID & MCM == SFID & MCM

where:

- & is the bitwise AND operator
- DID is the destination ID field in the header of the received packet.

Note: Only one of the Multicast Address and Multicast Mask filters can be active at the same time!

5.2.1.10. Maximum Packet Length

MPL							
7	6	5	4	3	2	1	0
Maximum Packet Length[7:0]							

If the maximum packet length filter is enabled and the PL header field of the received packet is greater than MPL, then the packet is ignored. It works in dynamic packet length mode only.

5.2.1.11. MAC Status Register

MSR							
7	6	5	4	3	2	1	0
Internal States							

MAC status register gives information about the current state of EZMAC PRO. The register holds the actual state of the state machine.

Table 17. States of the State Machine

EZMAC PRO States	State for	Timer IT	Ext. IT	Processor Load
EZMAC_PRO_ IDLE	Idle	—	—	—
EZMAC_PRO_ SLEEP	Sleep	—	—	—
EZMAC_PRO_ WAKE_UP	Wake Up	Enabled	Enabled	Low
WAKE_UP_ERROR	Wake Up	—	—	Low
TX_STATE_LBT_START_LISTEN	Transmit	Enabled	Enabled	Medium
TX_STATE_LBT_LISTEN	Transmit	Enabled	Enabled	Medium
TX_STATE_LBT_RANDOM_LISTEN	Transmit	Enabled	Enabled	Medium
TX_STATE_WAIT_FOR_TX	Transmit	Enabled	Enabled	Low
TX_STATE_WAIT_FOR_ACK	Transmit	Enabled	Enabled	Low
TX_ERROR_NO_ACK	Transmit	—	—	Low
TX_ERROR_CHANNEL_BUSY	Transmit	—	—	Low
TX_ERROR_STATE	Transmit	—	—	Low
RX_STATE_FREQUENCY_SEARCH	Receive	Enabled	Enabled	High
RX_STATE_CHANGE_ANTENNA	Receive	Enabled	Enabled	High
RX_STATE_WAIT_FOR_PREAMBLE	Receive	Enabled	Enabled	Low
RX_STATE_WAIT_FOR_SYNC	Receive	Enabled	Enabled	Low
RX_STATE_WAIT_FOR_HEADERS	Receive	Enabled	Disabled	Medium
RX_STATE_WAIT_FOR_PACKET	Receive	Enabled	Enabled	Medium
RX_STATE_WAIT_FOR_SEND_ACK	Receive	Enabled	Enabled	Low
RX_STATE_FORWARDING_LBT_START_LISTEN	Receive	Enabled	Enabled	Medium
RX_STATE_FORWARDING_LBT_LISTEN	Receive	Enabled	Enabled	Medium
RX_STATE_FORWARDING_LBT_RANDOM_LISTEN	Receive	Enabled	Enabled	Medium
RX_STATE_FORWARDING_WAIT_FOR_TX	Receive	Enabled	Enabled	Low
RX_ERROR_FORWARDING_WAIT_FOR_TX	Receive	—	—	Low
RX_ERROR_STATE	Receive	—	—	Low

Upper nibble of the MAC Status Register reflects the actual main status of the MAC.

Table 18. Main States

EZMAC PRO States	Status Register
Sleep	0x00
Idle	0x40
Wake Up	0x80
Wake Up Error	0x8F
Receive	0x1Z
Transmit	0x2Z

Note: “Z” means that the value of the lower nibble depends on the actual state of the main state.

5.2.1.12. Receive Status Register

RSR							
7	6	5	4	3	2	1	0
SELFA	MCA	BCA	-	-	-	PRF[1:0]	

The register is updated after each packet reception and gives information about the received packet.

SELFA This bit is set if the received packet passed the Self Address Filter.

MCA This bit is set if the received packet passed the Multicast Filter.

BCA This bit is set if the received packet passed the Broadcast Address Filter.

PRF[1:0] These bits show the frequency channel ID on which the packet was received.

5.2.1.13. Received Signal Strength Indicator

RSSI							
7	6	5	4	3	2	1	0
Received Signal Strength Indicator[7:0]							

After receiving the synchron word of the incoming packet EZMAC PRO measures the input signal strength level.

This register can be read during receive: it gives back the actual signal strength level.

Note: Refer to the EZRadioPRO data sheet about how the RSSI value is related to the input signal strength.

5.2.1.14. Self Customer ID

SCID							
7	6	5	4	3	2	1	0
Self Customer ID[7:0]							

The value of this register defines the CID header field of the transmitted packet and is also used for Customer ID filtering.

5.2.1.15. Self ID

SFID							
7	6	5	4	3	2	1	0
Self ID[7:0]							

The value of the register defines the sender ID header field of the transmitted packet and is also used for the Self Address filtering.

5.2.1.16. Received Control Byte

RCTRL							
7	6	5	4	3	2	1	0
SEQ[3:0]				ACK	ACKRQ	RAD[1:0]	

The register holds the value of the received CTRL byte.

SEQ[3:0] The sequence number of the received packet (packet ID)

ACK If this bit is set, the packet is an acknowledgement packet.

ACKRQ If this bit is set, then the originator asked for acknowledgement.

RAD[1:0] The radius field of the received packet.

5.2.1.17. Received Customer ID

RCID							
7	6	5	4	3	2	1	0
Received Customer ID[7:0]							

The register holds the value of the customer ID header field of the received packet.

Note: This register is implemented only if the Customer ID support is compiled into the source code.

5.2.1.18. Received Sender ID

RSID							
7	6	5	4	3	2	1	0
Received Sender ID[7:0]							

The register holds the value of the sender ID header field of the received packet.

5.2.1.19. Destination ID

DID							
7	6	5	4	3	2	1	0
Destination ID[7:0]							

The register has two functions:

- Upon packet reception, it holds the value of the destination ID header field of the received packet.
- Upon packet transmission, the register defines the destination address to which the packet needs to be transmitted.

Note: It is recommended to write the destination address each time into the register before a packet is transmitted because a received packet overwrites the previous value.

5.2.1.20. Payload Length

PLEN							
7	6	5	4	3	2	1	0
Payload Length[7:0]							

The register has two functions:

- Upon packet reception, it holds the payload length of the received packet.
- Upon packet transmission, it only has an effect in fix packet length mode: it defines the payload length of the packet that needs to be transmitted. In fix packet length mode, the register defines the length of the packet both for transmitting and receiving mode.

Note: The maximum value that can be written into this register is defined by RECEIVED_BUFFER_SIZE definition (however, it cannot be greater than 64). If a greater value is written into the register, then RECEIVED_BUFFER_SIZE value is set, and the EZMacPRO_Reg_Write() function returns with VALUE_ERROR.

5.2.1.21. Received Data Buffer

After successful packet reception, the content of the received payload is copied into the Received Data buffer, and the Payload Length register is updated accordingly. The size of the Received Data Buffer is configurable before compiling the source code; so, the RAM requirement of EZMAC PRO can be reduced if the system does not send long packets. The size of the buffer is defined by the RECEIVED_BUFFER_SIZE definition in the EZMacPRO_defs.h file. If the value of the definition is greater than 64, the compiler provides the following error message: "The maximum size of the Received Data Buffer is 64!"

Note: The Received Data Buffer can be read by the EZMacPRO_RxBuf_Read() function.

There is no buffer assigned for packet transmission; the data immediately filled into the FIFO of the EZRadioPRO device when the EZMacPRO_TxBuf_Write() function is called and the Payload Length register is updated accordingly.

5.2.1.22. RSSI Limit Register

RSSILR							
7	6	5	4	3	2	1	0
RSSIL[7:0]							

RSSIL[7:0]

These bits define the RSSI limit for the Frequency Search mechanism. If the input signal strength of the given channel is above this limit, then EZMAC PRO stays on the channel and tries to receive the preamble and then the entire packet. If the input signal strength is below this limit, then EZMAC PRO immediately jumps to the next channel.

Note: The limit has to be defined carefully because it defines the sensitivity limit of the node. The node does not receive packets with lower input power strength than the RSSI Limit setting. However, if the limit is set below the sensitivity level of the radio, then EZMAC PRO will spend too much not-expected time by waiting for the preamble. This could cause packet loss.
The recommended RSSI limit is a few dB above the sensitivity level of the radio for the given circumstances (it is defined by the RF parameters, such as data rate, deviation, and receiver filter bandwidth).
Refer to the EZRadioPRO data sheet for information about how the RSSI value is related to the input signal strength.

5.2.1.23. Listen Before Talk Interval Register

LBTIR							
7	6	5	4	3	2	1	0
TBI	LBTI[6:0]						

TBI — Time or byte time interval. If this bit is set, then LBTI is specified in fix time intervals. If this bit is cleared, then LBTI is specified in byte time intervals.

LBTI[6:0] — It specifies the pseudo random time period for the Listen Before Talk mechanism:
$T_{PS} = n \times LBTI[6:0]$ where:

- n is a random number between 0…15 (the range is doubled for each retry, but the maximum value is 0…63)
- LBTI[6:0] is Listen Before Talk Interval.

LBTI can be set in byte intervals (1…127byte interval) or it can be set in fix time intervals as well (100 µs…12.7 ms). The selection between the two options depends on the application needs: ETSI specifies the LBTI in approximately 0.5 ms intervals; if another standard is used, the byte interval may be sufficient.

Note: See "3.4. Listen before Talk" on page 15 for more details.

5.2.1.24. Listen Before Talk Limit Register

LBTL							
7	6	5	4	3	2	1	0
RSSI[7:0]							

RSSI[7:0] These bits define the RSSI limit for the Listen Before Talk mechanism. If the input power is above the limit, then the channel is considered to be occupied. If the input power is under the limit, the channel is considered to be free.

Note: Refer to the EZRadioPRO data sheet for information about how the RSSI value is related to the input signal strength.

5.2.1.25. Byte Counters

BCHx							
7	6	5	4	3	2	1	0
Byte Counter of Channel x[15:8]							

BCLx							
7	6	5	4	3	2	1	0
Byte Counter of Channel x[7:0]							

Byte Counters of channel 0…3

If the byte counter feature is compiled into the source code, then EZMAC PRO counts the number of transmitted bytes for each channel separately. The counters do not overflow, and the counting stops at 0xFFFF. These can be read and cleared by the upper software layer. Based on the number of transmitted bytes, the upper software layer can calculate the duty cycle of the channel usage: how often the given node occupied the given frequency channel. This feature helps to monitor network behavior and also helps to implement an ETSI-compliant system.

Note: These registers are not implemented if the Byte Counter feature is not compiled into the source code.

5.2.1.26. Low Frequency Timer Setting Registers

LFTMR0							
7	6	5	4	3	2	1	0
WTM[7:0]							

LFTMR1							
7	6	5	4	3	2	1	0
WTM[15:8]							

LFTMR2							
7	6	5	4	3	2	1	0
LFTMRE	IETB	WTR[3:0]				WTD[1:0]	

WTM[15:0] Wake Up Timer Mantissa

WTD[1:0] Wake Up Timer Exponent (D)

WTR[3:0] Wake Up Timer Exponent (R)

IETB Internal / External Time Base. If this bit is set, then the Wake Up Timer of the EZRadioPRO device uses the internal RC oscillator as clock source. If this bit is cleared, then the Wake Up Timer uses the external 32.768 kHz watch crystal. The external crystal has to be connected to GPIO0 of the EZRadioPRO device.

Note: If the external crystal mode is selected, then EZMAC PRO automatically changes the functionality of the GPIO0 pin to serve the external crystal. If IETB bit is cleared, then EZMAC PRO sets the functionality of GPIO0 as defined by the GPIO0_FUNCTION definition.

LFTMRE Setting this bit enables the Wake Up Timer of the radio. EZMAC PRO then periodically calls the void EZMacPRO_LFTimerExpired(void) callback function every time the wake Up Timer expires. The time period of the Wake Up Timer is calculated as follows:

$$T_{WUT} = \left(\frac{32 \times WTM \times 2^{WTR-WTD}}{32.768}\right)[ms]$$

Note: The Wake Up Timer of the radio is configured and started only if the LFTMRE bit is set from 0 to 1. That is why the LFTMR0 and LFTMR1 registers have to set before the LFTMR2 register is written; otherwise, the time period will not be changed, and the Wake Up Timer will not be started.

5.2.1.27. Low Battery Detect Register

LBDR							
7	6	5	4	3	2	1	0
LBDE	-	-	LBDT[4:0] / VBAT[4:0]				

LBDE If this bit is set, the Low Battery Detector is enabled in the radio:

- The power supply voltage of the radio can be read by the VBAT bits
- EZMAC PRO calls the void EZMacPRO_LowBattery(void) callback function in case the power supply voltage of the radio is below the LBDT threshold.

If the LBDE bit is cleared, the Low Battery Detect is disabled in the radio: the voltage cannot be read, and it does not check the voltage threshold.

LBDT[4:0] Writing into these bits defines the threshold for the low battery detector circuit. The threshold voltage is programmable:

$$V_{Threshold} = (1.675 + LBDT[4:0] \times 50\ mV) \pm 25\ mV$$

VBAT[4:0] Reading from these bits gives back the actual power supply voltage of the radio:

$$V_{Supply} = (1.675 + VBAT[4:0] \times 50\ mV) \pm 25\ mV$$

5.2.1.28. ADC and Temperature Sensor Register

ADCTSR							
7	6	5	4	3	2	1	0
ADCS	ADCSEL[2:0]			ADCREF[1:0]		TSRANGE[1:0]	

ADCS If this bit is set, then the AD converter is set according to the ADCSEL / ADCREF / TSRANGE bits. The conversion starts and the ADCTSV register is cleared. If the conversion is done, the ADCS bit is cleared and the result is copied into the ADC / Temperature Value Register. The AD conversion takes about 350 µs.

ADCSEL[2:0] These select the input source of the ADC:

Table 19. ADC Input Source Selection

ADCSEL[2:0]	Input Source Selection
0	Internal temperature Sensor
1	GPIO0, single-ended
2	GPIO1, single-ended
3	GPIO2, single-ended
4	GPIO0(+) – GPIO1(–), differential
5	GPIO1(+) – GPIO2(–), differential
6	GPIO0(+) – GPIO2(–), differential
7	GND, ADC is disabled

ADCREF[1:0]ADC Reference Voltage Selection:

Table 20. ADC Reference Voltage Selection

ADCREF[0:1]	ADC Reference Voltage Selection
0	Bandgap Voltage (1.2 V)
1	
2	VDD/3
3	VDD/2

TSRANGE[1:0] Temperature Sensor Range selection:

Table 21. Temperature Sensor Range Selection

TSRANGE[0:1]	Temperature Sensor Range Selection
0	–40 °C … 64 °C, resolution 0.5 °C
1	–40 °C … 85 °C, resolution 1 °C
2	0 °C … 85 °C, resolution 0.5 °C
3	–10 °F … 216 °F, resolution 1 °F

Notes:
1. The register could be written in IDLE mode only.
2. The gain of the ADC is set by the EZMACPRO_ADC_GAIN definition in the EZMacPRO_defs.h. See the EZRadioPRO data sheet for more details regarding ADC gain.
3. The ADC sensor amplifier offset is set by the EZMACPRO_ADC_AMP_OFFSET definition in the EZMacPRO_defs.h. See the EZRadioPRO data sheet for more details regarding the offset.

5.2.1.29. ADC/Temperature Value Register

ADCTSV							
7	6	5	4	3	2	1	0
ADC and Temperature Sensor Value[7:0]							

The register gives back the result of the ADC conversion. If the temperature sensor is selected as a source of the ADC, then the register provides the measured temperature value.

The last measured result is kept in the register until a new conversion is started.

CONTACT INFORMATION

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
Tel: 1+(512) 416-8500
Fax: 1+(512) 416-9669
Toll Free: 1+(877) 444-3032

Please visit the Silicon Labs Technical Support web page:
https://www.silabs.com/support/pages/contacttechnicalsupport.aspx
and register to submit a technical support request.