EZMAC® AND EZHOP USER'S GUIDE

1. Introduction

The EZMac® and EZHop solutions are wireless communication software modules for embedded systems using the EZRadioPRO® radios and wireless MCUs. The software stack supports both solutions in a single software package using the same API. Option selections can be made simply by selecting the appropriate definitions.

The EZHop is used for applications that need to meet FCC Part 15 Section 247 in order to transmit at high power (above –1.2 dBm). EZHop creates a Frequency Hopping protocol that uses up to 50 channels. According to the FCC requirements for frequency hopping systems operating in the 902–928 MHz band, if the +20 dB bandwidth of the hopping channel is less than 250 kHz, the system shall use at least 50 hopping frequencies. The maximum allowed +20 dB bandwidth of the hopping channel is 500 kHz.

In systems where high output power can be used without the need for compliance to the FCC rules, the basic implementation of the EZMac stack can be used. The basic implementation of the EZMac module supports up to four channels and is used to increase the robustness of communication. This implementation means communication is simpler and consumes less current and that additional features are available.

The EZMac and EZHop solutions transmit and receive data in short packets via an RF link in the ISM band and do not require an additional MCU. The EZMac and EZHop solutions run as background tasks on the main application microcontroller as two interrupt service routines. There is no part of the MAC engine in the foreground loop, so that can be used exclusively for the embedded application.

The EZMac module supports a wide range of addressing modes, packet filtering modes, collision, and error detection schemes. Additionally, the EZMac module provides a built-in acknowledgement and packet forwarding feature. The EZMac module is compatible with earlier versions of EZMac, which was designed to support the EZRadio® family of radio devices.

EZHop also supports a wide range of addressing modes, packet filtering modes, and collision detection with the built-in acknowledgement and packet forwarding feature and implements a better frequency search mechanism to find valid packets during the receiving process. Both EZMac and EZHop solutions support either peer-to-peer or star networks with up to 255 addressable nodes on a single subnet.

The EZMac and EZHop solutions support the Si403x transmitter, the Si433x receiver, and the Si443x transceiver ICs from the EZRadioPRO family and support the Si100x and Si101x Wireless MCU ICs; depending on the feature set required, certain capabilities can be disabled before compilation of the source code allowing designers to optimize code for any given application. "2.2. Memory Requirements" on page 4 provides greater detail on code and RAM size.

The EZMac and EZHop solutions are developed in a compiler-independent manner using the C programming language for the Silicon Labs C8051F93x microprocessor, but users may easily port the code to other MCU platforms if required, subject to licence requirements. The code is designed to be flexible and, usually, only requires small modifications to the compiler_defs.h and processor definition header files. Example compilers supported and defined in the compiler_defs.h are as follows:

- Keil:
  http://www.keil.com
- SDCC (Small Device C Compiler):
  http://sdcc.sourceforge.net
- Raisonance:
  www.raisonance.com, etc.

The EZMac and EZHop solutions are designed on the Silicon Labs' wireless Software Development Board using an EZRadioPRO test card (TX/RX Split, Single antenna, and Antenna diversity testcards are available). However, the software can also be run on the EZLink™ fast prototyping platform and on the Si10xx Motherboard used with an Si10xx testcards platform. There are dedicated project files for all hardware platforms. The source folders contain example projects for the three compiler versions and three prototyping platforms. For further details on hardware platforms, visit http://www.silabs.com.

TABLE OF CONTENTS

2. Hardware Requirements

2.1. Peripherals Requirements

The EZMac and EZHop solutions are designed to operate with the EZRadioPRO family of devices (Si403x, Si4330, Si443x) and Wireless MCUs devices (Si100x, Si101x) using a microcontroller’s SPI port (the software uses the HW SPI1 peripheral of the microcontroller) and interrupt pin. The microcontroller should be able to wake from SLEEP mode if the transceiver pulls the interrupt pin to logic low.

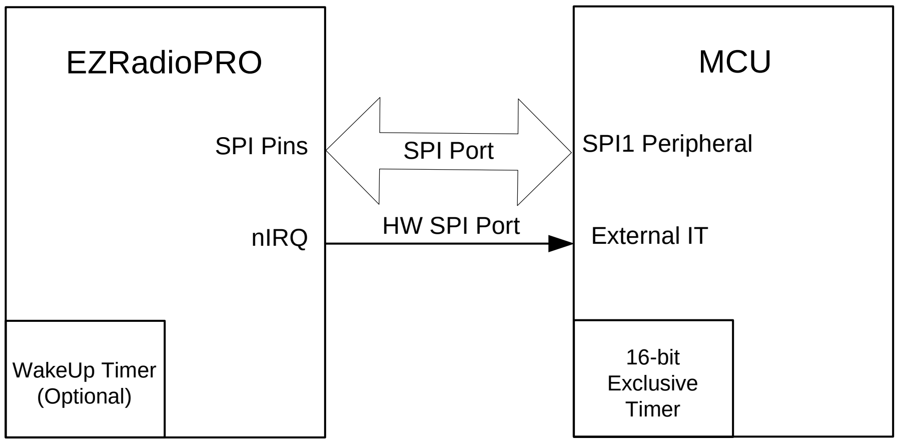


Figure 1. HW Configuration

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

The stack should provide a low-frequency time base for the application layer to realize time synchronized operations. The Wake-Up Timer of the radio is used for this purpose.

The source code is designed for the C8051F930 microprocessor and uses the INT0 interrupt source, Timer3 timer module, and SPI1 peripheral by default. This can easily be changed thanks to the layered software architecture of the stack.

The EZMac and EZHop solutions do not have any additional peripheral requirements.

2.2. Memory Requirements

2.2.1. Definitions

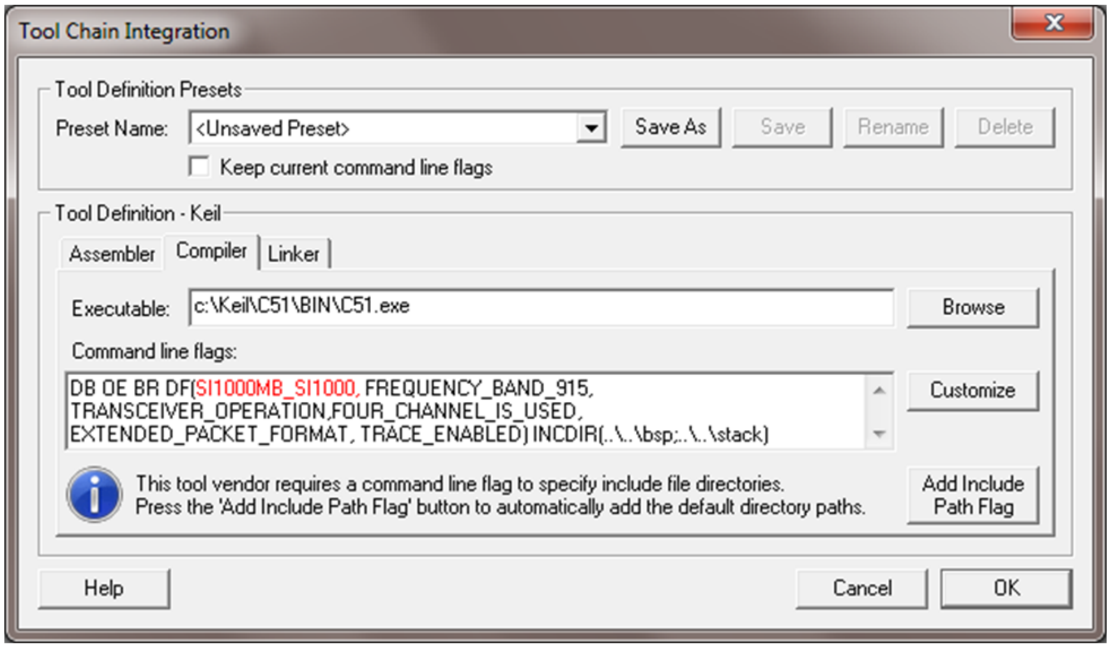
The memory requirement depends on the operational mode and the features selected. The following definitions determine the EZMac and EZHop stack behavior, properties, and selected features. These are pre-compile options, and, as such, the appropriate preprocessor symbols have to be defined in the Project/Tool Chain Integration/Compiler menu of the Silicon Labs IDE prior to compilation of the source code.

2.2.2. Hardware Platform Selection

The following HW platforms are supported by EZMac/EZHop:

HW Platform	Compiler symbol
Software Development Board/MSC-DBSB8	SDBC
EZLink Module	EZLINK
Si1000 daughtercard with Si1000 motherboard	SI1000MB_SI1000
Si1010 daughtercard with Si1000 motherboard	SI1000MB_SI1010

The corresponding compiler symbol should be defined in the Silicon Labs IDE under “Project” → “Tool Chain Integration” → “Compiler” → “Command line flags”. Only one of these should be defined at a time.



```
DB OE BR DF(SI1000MB_SI1000, FREQUENCY_BAND_915,
TRANSCEIVER_OPERATION,FOUR_CHANNEL_IS_USED,
EXTENDED_PACKET_FORMAT, TRACE_ENABLED) INCDIR(..\..\bsp;..\..\stack)
```

This tool vendor requires a command line flag to specify include file directories.
Press the 'Add Include Path Flag' button to automatically add the default directory paths.

2.2.2.1. Supported Chip Revision

If the B1_ONLY definition is not enabled, then the EZRadioPRO RevA0, RevV2, and revB1 radios are supported. This selection is made during runtime and checked during the initialization of the stack. Omitting this selection allows the code to be used on all EZRadioPRO IC revisions but increases code size. B1_ONLY is defined in hardware_defs.h.

2.2.2.2. Frequency Band Selection

All major sub-GHz ISM bands (434, 868, and 915 MHz) are supported by EZMac and selected via the appropriate definition in the Project/Tool Chain Integration/Compiler menu of the Silicon Laboratories IDE:

FREQUENCY_BAND_434

FREQUENCY_BAND_868

FREQUENCY_BAND_915

Note: Only one definition can be enabled at any time, and the enabled definition determines the frequency band.

2.2.2.3. Radio Operation Selection

The source code can be compiled for transmitter, receiver, or transceiver operation by enabling the appropriate definition:

TRANSCEIVER_OPERATION

TRANSMITTER_ONLY_OPERATION

RECEIVER_ONLY_OPERATION

Using transmitter only or receiver only operation will result in a smaller code space because only the required API functions are compiled into the code.

Note: Only one definition can be enabled at any one time, and the enabled definition determines the actual operation mode.

2.2.2.4. Channel Number Selection

These definitions define the operation mode of the stack: EZMac or EZHop mode.

If the FOUR_CHANNEL_IS_USED definition is enabled, the stack operates as the EZMac module and supports up to four channels. If the MORE_CHANNEL_IS_USED definition is enabled, the stack operates as EZHop and supports up to 50 channels.

FOUR_CHANNEL_IS_USED

MORE_CHANNEL_IS_USED

Note: Only one definition can be enabled at any one time.

2.2.2.5. Antenna Diversity

If the ANTENNA_DIVERSITY_ENABLED definition is enabled, the antenna diversity feature is compiled into the source code.

Note: Antenna diversity is supported only if EZMac mode is selected.

2.2.2.6. Packet Format Selection

Two definitions determine the packet format:

STANDARD_PACKET_FORMAT

EXTENDED_PACKET_FORMAT

For a detailed description, see "3.3. Packet Format" on page 17. The automatic acknowledgement feature can be used only in EXTENDED_PACKET_FORMAT. Further information about the automatic acknowledgment can be found in "3.6. Automatic Acknowledgement" on page 22.

2.2.2.7. Packet Forwarding

If the PACKET_FORWARDING_SUPPORTED definition is enabled, the packet forwarding feature can be used. Packet forwarding can be used only with extended packet format.

2.2.3. Compiling Options

Most of the features with large FLASH or memory requirements can be disabled by definition under the Project/Tool Chain Integration/Compiler tab in the Silicon Labs IDE. Eleven main compile combinations are tested during the verification process.

The 11 compilation combinations can be divided into four groups:

- The EZMac module with standard packet format
- The EZMac module with extended packet format
- The EZMac module with extended packet format and antenna diversity
- EZHop

Tables 1 through 4 give an overview of the required code and memory requirements for the 11 tested compile options. The code and memory size were measured while the B1_ONLY definition was enabled. The enabled definitions are marked with Xs in the tables below.

2.2.3.1. EZMac Module with Standard Packet Format

There are three compile combinations defined and fully tested in this group.

Table 1. EZMac Module with Standard Packet Format

	TRANSCEIVER_OPERATION	TRANSMITTER_ONLY_OPERATION	RECEIVER_ONLY_OPERATION	FOUR_CHANNEL_IS_USED	MORE_CHANNEL_IS_USED	ANTENNA_DIVERSITY_ENABLED	STANDARD_PACKET_FORMAT	EXTENDED_PACKET_FORMAT	PACKET_FORWARDING_SUPPORTED	Code Size (Bytes)	RAM Size (Bytes)
Transceiver operation mode	X			X			X			5320	189
Transmitter Only operation mode		X		X			X			3184	82
Receiver Only operation mode			X	X			X			4220	165

2.2.3.2. EZMac Module with Extended Packet Format

There are three compile combinations defined and fully tested in this group.

Table 2. EZMac Module with Extended Packet Format

	TRANSCIEVER_OPERATION	TRANSMITTER_ONLY_OPERATION	RECEIVER_ONLY_OPERATION	FOUR_CHANNEL_IS_USED	MORE_CHANNEL_IS_USED	ANTENNA_DIVERSITY_ENABLED	STANDARD_PACKET_FORMAT	EXTENDED_PACKET_FORMAT	PACKET_FORWARDING_SUPPORTED	Code Size (Bytes)	RAM Size (Bytes)
Transceiver operation mode	X			X				X	X	6905	221
Transmitter Only operation mode		X		X				X		3374	92
Receiver Only operation mode			X	X				X		4309	172

2.2.3.3. EZMac Module with Extended Packet Format And Antenna Diversity

There are two compile combinations defined and fully tested in this group.

Table 3. Extended EZMac Module with Antenna Diversity

	TRANSCIEVER_OPERATION	TRANSMITTER_ONLY_OPERATION	RECEIVER_ONLY_OPERATION	FOUR_CHANNEL_IS_USED	MORE_CHANNEL_IS_USED	ANTENNA_DIVERSITY_ENABLED	STANDARD_PACKET_FORMAT	EXTENDED_PACKET_FORMAT	PACKET_FORWARDING_SUPPORTED	Code Size (Bytes)	RAM Size (Bytes)
Transceiver operation mode	X			X		X		X	X	6960	224
Receiver Only operation mode			X	X		X		X		4360	175

2.2.3.4. EZHop

There are three compile combinations defined and fully tested in this group.

Table 4. EZHop

	TRANSCIEVER_OPERATION	TRANSMITTER_ONLY_OPERATION	RECEIVER_ONLY_OPERATION	FOUR_CHANNEL_IS_USED	MORE_CHANNEL_IS_USED	ANTENNA_DIVERSITY_ENABLED	STANDARD_PACKET_FORMAT	EXTENDED_PACKET_FORMAT	PACKET_FORWARDING_SUPPORTED	Code Size (Bytes)	RAM Size (Bytes)
Transceiver operation mode	X				X			X	X	6693	266
Transmitter Only operation mode		X			X			X		3316	137
Receiver Only operation mode			X		X			X		4204	218

Note: All the source codes were compiled with a Keil C51 Version 8.16a compiler for code size estimations. The parameters of the compiler are defined as follows:

- Optimization level: Level7
- Emphasis: Favor Fast Code
- Variable location: Small (data)
- Code size: Large (64K functions)

2.3. MCU Clock

The EZMac and EZHop solutions are designed to run on the C8051F930 microcontroller using a minimum 4 MHz system clock. There are 4 system clock rates predefined in hardware_defs.h: 4 MHz, 8 MHz, and 16 MHz with the external crystal and 24.5 MHz using the internal precision oscillator. The timing parameters of the stack have been verified with all four SYSCLK options.

The EZMac and EZHop solutions are designed in a way that does not require a crystal-based clock source for the microcontroller; critical timings are performed by the radio itself.

3. EZMac Module Supported RF Devices

The EZMac and EZHop solutions are designed for the EZRadioPRO and Wireless MCU family of devices. The EZRadioPRO family consists of transmitters (Si403x), receivers (Si4330), and transceivers (Si443x, Si100x, Si101x). The source code can be compiled for each device using the following definitions:

- TRANSMITTER_ONLY_OPERATION
- RECEIVER_ONLY_OPERATION
- TRANSCEIVER_OPERATION

If the EZMac and EZHop solutions are compiled for transmitter-only or receiver-only operations, then unused code segments are not compiled, reducing FLASH and RAM memory.

For appropriate operation, the revision of the radio chip must be selected. This selection can be done in two ways:

- If the B1_ONLY definition is enabled in the EZMacPro_defs.h file, then only revision B1 can be used.
- If the B1_ONLY definition is disabled, then the stack automatically recognizes the device and configures the radio accordingly.

Notes:

1. Only one of the options can be selected at any one time; otherwise, the compiler will give one of the following error messages:
   "Both Transmitter Only and Receiver Only cannot be defined!"
   "Both Transceiver and Transmitter Only cannot be defined!"
   "Both Transceiver and Receiver Only cannot be defined!"
2. If the B1_ONLY definition is disabled, the code size will be larger. In this case, all constant tables have to be stored for each supported chip revision. Also, all revision-specific code segments have to be compiled into the source code.

3.1. Receiving Process

The key feature of the EZMac and EZHop solutions is the frequency-redundant receiving procedure. In the case of the EZMac module, the radio can scan up to four channels; however, in the case of EZHop, it can scan up to 50 channels. The frequency redundant/frequency hopping receiver procedure allows for sending a packet on a randomly-selected channel and the receiver to find the packet independently.

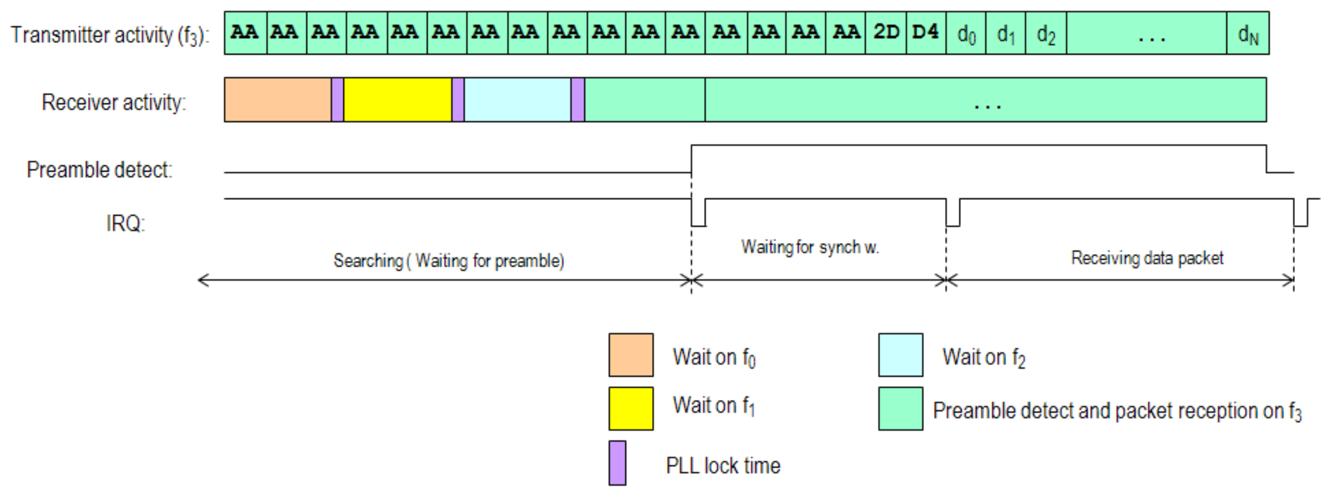

Transmitter activity (f_3):
AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA AA 2D D4 d_0 d_1 d_2 . . . d_N
Receiver activity:
. . .
Preamble detect:
IRQ:
Searching (Waiting for preamble)
Waiting for synch w.
Receiving data packet
Wait on f_0
Wait on f_1
PLL lock time
Wait on f_2
Preamble detect and packet reception on f_3

Figure 2. Receiving Process

For the EZMac module, The receive procedure is realized in the following manner: the receiver scans channels next to each by looking for the preamble. Based on the preamble detection interrupt, it evaluates whether there is any RF activity on that channel. If there is no RF activity (since there is no preamble detection interrupt within the timeout period), it jumps to the next channel.

If the channel is occupied and there is a preamble detection, further evaluation is performed, and it stays on the channel and waits for the synchron word. If the synchron word is detected within the timeout period, it waits to receive the entire packet. If the synchron word is not detected within the timeout period, the node jumps to the next channel.

The receive procedure is a little different for EZHop. If there is a valid RF activity on the channel, the channel is checked again as the procedure checks for a second preamble detect interrupt. If there is a valid RF activity on the channel a second time, the receive procedure continues as it did for the EZMac module. For EZHop, preamble lengths are much longer due to the number of channels, and waiting for the synchron word timeout will take too long. Checking a second time allows the procedure to confirm that a modulated signal is in the channel.

The receiver requires a minimum 3 byte time to evaluate a channel (this search time is data rate and preamble detection threshold dependent). Hopping from a frequency to a second frequency requires about 200 µs since the receiver has to wait for the PLL to settle. It is for this reason that the transmitter sends preambles sufficiently long to allow the receiver to evaluate the used channels. The EZMac and EZHop solutions set the appropriate preamble lengths automatically so that higher software layers do not need to configure them.

The EZRadioPRO device requires different lengths of preamble depending on whether the Auto Frequency Control feature is enabled or disabled. The Auto Frequency Control (AFC) operates in receive mode during the preamble receive. It tries to find the exact center frequency of the transmitter in order to adjust for any offset between that and receiver nodes. If the AFC of the radio is enabled, the radio can tolerate offset up to 0.35 times the IF bandwidth. If the AFC is not used, the radio can tolerate offset up to 0.25 times the IF bandwidth.

Notes:

1. In the source code, the AFC is enabled for the Si4x3x revision A0 and B1 devices and disabled for the Si4x32 revision V2 devices.
2. The AFC selection is part of the table that stores the RF parameters. For more details, see "3.2. Data Rate, Frequency Channel Selection" on page 13.
3. The preamble detection threshold for every EZRadioPRO device is set in the EZMacPro_const.h header file. If less preamble detection threshold is sufficient for the given application, then the number of transmit preamble (for every data rate) can be decreased by the difference.

3.1.1. Required Preamble Length for the EZMac Module

The required preamble length depends on the used data rate and preamble detection threshold. The following tables indicate the required preamble length for 1.5 and 2.5 bytes of preamble detection threshold. A shorter threshold may be selected if some packet loss is allowed by the network. In this case, transmitted preamble length will be shorter, but the reliability of the searching mechanism may decrease because of the possible false preamble detection in a nosy environment. Therefore, the preamble detection threshold is a design parameter; a shorter threshold has to be evaluated in an appropriate environment.

Table 5. Number of Required Transmit Preamble Bytes (2.5 Byte Preamble Detection Threshold)

Data Rate (kbps)	Number of Required Preamble (Byte)			
	4 Channels	3 Channels	2 Channels	1 Channel
2.4	16	13	10	4
9.6	16	13	10	4
50	24	19	14	4
128	32	25	18	4

Table 6. Number of Required Transmit Preamble Bytes (1.5 Byte Preamble Detection Threshold)

Data Rate (kbps)	Number of Required Preamble (Byte)			
	4 Channels	3 Channels	2 Channels	1 Channel
2.4	12	10	8	4
9.6	12	10	8	4
50	20	16	12	4
128	28	22	16	4

The EZRadioPRO devices support an embedded automatic antenna diversity mechanism, so the EZMac module can be compiled for this operation by enabling the ANTENNA_DIVERSITY_ENABLED definition in the EZMacPRO_defs.h file. The receive procedure is the same as when the diversity mode is disabled with one exception that the chip automatically selects the appropriate antenna for receiving the packet based on the antenna evaluation.

Table 7. Number of Required Preamble Bytes for Antenna Diversity (2.5 Byte Preamble Detection Threshold)

Data Rate (kbps)	Number of Required Preamble (Byte)			
	4 Channels	3 Channels	2 Channels	1 Channel
2.4	16	13	10	8
9.6	16	13	10	8
50	24	19	14	8
128	32	25	18	8

Table 8. Number of Required Preamble Bytes for Antenna Diversity (1.5 Byte Preamble Detection Threshold)

Data Rate (kbps)	Number of Required Preamble (Byte)			
	4 Channels	3 Channels	2 Channels	1 Channel
2.4	12	10	8	8
9.6	12	10	8	8
50	20	16	12	8
128	28	22	16	8

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

Note: Only one option can be chosen prior to the compilation of the source code: non-antenna diversity mode or antenna diversity mode. The decision is made by the ANTENNA_DIVERSITY_ENABLED definition in the EZMacPRO_defs.h file.

3.1.2. Required Preamble Length in EZHop

The required preamble length in EZHop depends on the number of frequency channels selected, the selected data rate, and the preamble detection threshold. This threshold is considered a design parameter. Table 9 shows the required transmit preamble length. In case of 2.4 kbps, 9.6 kbps data rate the preamble detection threshold is 2 bytes and in case of 50 kbps, 128 kbps data rate the preamble detection threshold is only 1.5 bytes. Shorter thresholds may be selected if some packet loss is allowed by the network. In this case, transmitted preamble length will be shorter, but the reliability of the searching mechanism might decrease because of possible false preamble detection in a nosy environment. Therefore, the preamble detection threshold is a design parameter; shorter threshold has to be evaluated in the final environment.

Note: *The shorter preamble detection threshold at a high data rate could cause higher packet error rate*. However, a longer preamble detection threshold would result in a longer transmitted preamble. In that case, 50 channels cannot be supported. An alternative solution could be to use only 25 channels. For further help, please contact technical support.

Table 9. Number of Required Preamble Bytes for EZHop Mode

Data Rate (kbps)	50 Channels
2.4	160
9.6	172
50	225
128	245

Note: Antenna diversity is not supported when using the EZHop mode, and the compiler will give the error message, "Antenna diversity is allowed only four channels!" if selected.

3.2. Data Rate, Frequency Channel Selection

The EZMac and EZHop solutions hide the RF parameter settings from the higher software layers. The application layer has to select the data rate only as all other RF parameters are set automatically. Up to four data rates can be selected at runtime during the MAC operation (DR[1:0] control bits). The data rate should not be fixed before compiling the source code; the higher layer can optimize the data throughput runtime according to application needs.

The source code contains predefined tables for all of the major sub-GHz ISM bands (434, 868, 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 standards of the selected ISM band. The RF parameters are stored in two different tables and two different files.

One of the tables stores the deviation, data rate register settings, and the modem parameters. This parameter table is different for Revision A0, V2, and B1 radios. The tables are well separated in the si4432_const_v2.c, si4431_const_a0.c, and si443x_const_b1.c files and can be easily edited by replacing one or more lines of the table, changing parameters, such as enabling or disabling the AFC, etc. if the application requires a different data rate or RF setting.

```
const SEGMENT_VARIABLE (RfSettingsB1[NUMBER_OF_SAMPLE_SETTING][NUMBER_OF_PARAMETER], U8, code) =
{
//IFBW,  COSR,  CRO2,  CRO1,  CRO0,  CTG1,   CTG0,  TDR1, TDR0, MMC1, TXFDEV, RXFDEV,  AFC,  AFC Limiter,  AFC Timing
{0x1C,     0x41,  0x60,  0x27,   0x52,  0x00,   0x04,  0x13,  0xA9,  0x24,   0x3D,    0x3D,     0x40, 0x1A,          0x0A},
```

Figure 3. Example of the RF Settings Table for B1 Revision

The above picture shows one line of the table; the different fields are explained in Table 10.

Table 10. Explanation of the RF Settings Table's Fields

Mnemonic	Register Name	Note
IFBW	IF Filter Bandwidth	These are the modem registers; they 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 Limiter	This register is used in case of Si4x3x revA0 and revB1 only if the AFC is enabled. It sets the AFC Limit.
AFC Timing	AFC Timing Control	This register is used in case of Si4x3x revA0 and revB1 only if the AFC is enabled. It sets the AFC Limit.

The second table stores the timing parameter, preamble length, start frequency, frequency step, and maximum number of frequency channels for the four data rates. This parameter table is in the EZMacPro_const.h file. These parameters are not chip-revision-dependent; therefore, they are stored separately from the RF parameters.

```
const SEGMENT_VARIABLE (Parameters[4][11], U8, code) =
{
        //FR_S1,  FR_S2,    FR_S3,   FR_ST,    MAX_CH, PR1,    PR2,    PR3,    PR4,    ST,     PRDT
        {0x53,    0x53,     0x40,    33,       4,      8,      10,     13,     16,     3,      5},
```

Figure 4. Example Parameters Table

Table 11. Explanation of the Parameters Table's Fields

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	—	The maximum index of the channel.
PR1, …, PR4	—	Number of transmitted preamble bytes for 1, …, 4 channel cases.
ST	—	Search time in bytes for the receiving process; the receiver stays on the channel ST byte time.
PRDT	—	Preamble detection threshold in nibbles.

The Parameters table is different for EZHop mode:

```
const SEGMENT_VARIABLE (Parameters[4][11], U8, code) =
{
        //FR_S1,  FR_S2,   FR_S3,   FR_ST,   ST,      CHNBR,   PREAL,   PREARV1, PREARV2, PRDT,     PLLT
{0x75,    0x0A,    0x80,    48,      24,      50,      160,     0x01,    0x40,    4,       0x52},
```

Figure 5. Example Parameters Table

Table 12. Explanation of the Parameter Table's Fields in EZHop

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.
ST	—	Search time in byte for the receiving process, the receiver stays on the channel ST bit time
CHNBR	—	Number of the used channels
PREAL	—	Number of transmitted preamble bytes.
PREARV1	Preamble Length	This register defines the preamble length.
PREARV2	Preamble Length	This register defines the preamble length.
PRDT	—	Preamble detection threshold in nibbles
PLLT	PLL Tune Time	PLL Tune Time register

3.2.1. Frequency Assignments

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

FREQUENCY_BAND_434

FREQUENCY_BAND_868

FREQUENCY_BAND_915

The number of available frequencies is dependant on the data rate; for higher data rates, the required bandwidth is higher, so less frequency channel can be allocated. The default frequency allocations for all data rates and frequency bands are defined to comply with the ETSI and FCC standard. In case of the 868MHz band the frequency allocation is defined for the general ETSI band, instead of the specific sub-bands. 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 and can be found in the EZMacPro_const.c file. The actual frequency is calculated as follows:

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

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

3.2.1.1. Frequency Assignment in Case of the EZMac Module

The EZMac module supports up to four channels. The number of used channels is programmable via the EZMac register during run time. The default frequency assignment of the EZMac module for different frequency bands is listed in Tables 13, 14, and 15 (it can be modified for the application needs).

Table 13. 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
9.6	433.33	330	4
50	433.61	360	2
128	433.91	0	1

Table 14. 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
9.6	863.55	450	14
50	863.75	780	8
128	864	1 MHz	6

Table 15. 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
9.6	900.84	480	60
50	900.84	780	37
128	900.84	1 MHz	29

Note: The default values of the Fx registers are F0 = 0 ... F3 = 3.

3.2.1.2. Frequency Assignment for the EZHop Mode

EZHop can support 50 channels in every data rate. The default frequency assignment of EZHop is listed in Table 16 (it can be modified for the application needs). EZHOP can be compiled only for the 915 MHz ISM Band; otherwise, the compiler gives an error message.

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 Used Channels**
2.4	900.84	480	50
9.6	900.84	480	50
50	900.84	480	50
128	900.84	480	50

According to FCC regulations, the frequency hop should occur in a pseudo-random order.

Note: The default value of the Fx registers are set according to the Frequency Table (for more details, see "3.2.1. Frequency Assignments" on page 15).

The pseudo random order can be different for each network implementation by customizing the Frequency Table. Modifying this table will help reduce unexpected interactions between different systems using EZHop that may be installed in close proximity to each other.

This frequency table is defined in the EZMacPro_const.h file.

```
const SEGMENT_VARIABLE (FrequencyTable[50], U8, code) =
{           0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
           10,11,12,13,14,15,16,17,18,19,
           20,21,22,23,24,25,26,27,28,29,
           30,31,32,33,34,35,36,37,38,39,
           40,41,42,43,44,45,46,47,48,49
};
```

Figure 6. Frequency Table in EZHop

3.3. Packet Format

The EZMac and EZHop solutions support two packet configurations: the standard and the 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_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.

The EZMac and EZHop solutions transmit data in short packets with a maximum payload of 64 bytes.

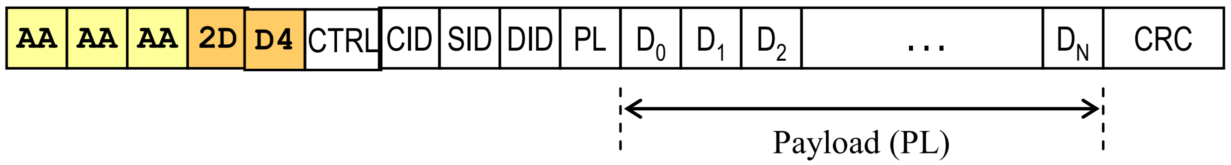


Figure 7. Packet Configuration

Table 17. Packet Configuration

Name	Description	Size [Bytes]
AA AA AAAA	Preamble	min. 4
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. This 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. The preamble length is longer in the case of EZHop because it uses more frequencies. The EZMac and EZHop solutions automatically set the transmit preamble according to the number of enabled channels.

The control byte is used only in the extended packet configuration mode to support 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 the EZMac and EZHop solutions that may be installed in close proximity to each other. Although this byte is optional, it is recommended. The Customer ID feature has to be selected via the CIDE (MCR register) control bit. If the CIDE control bit 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 nodes taking part in the actual data exchange. The EZMac and EZHop solutions have several packet filtering options that are based on the address (see "4.2. Packet Filtering Method" on page 26 for more details).

Payload Length (PL) is used in applications in which the number of the transmitted data bytes changes dynamically. The EZMac and EZHop solutions also support the fixed payload length operations. In this case, the PL field is not transmitted in the packet.

For D_0 … D_N, the maximum data to transmit is 64 bytes.

Cyclic Redundancy Code (CRC) is used to recognize if there are any corrupt data bits 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 will generate the following error message: "Only one packet format can be selected!".

3.4. Listen Before Talk

The EZMac and EZHop solutions both perform a Listen-Before-Talk procedure 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 easily be set to comply with proprietary protocols and standards, such as the ETSI standard. The Listen-Before-Talk mechanism operates as shown in Figure 6.

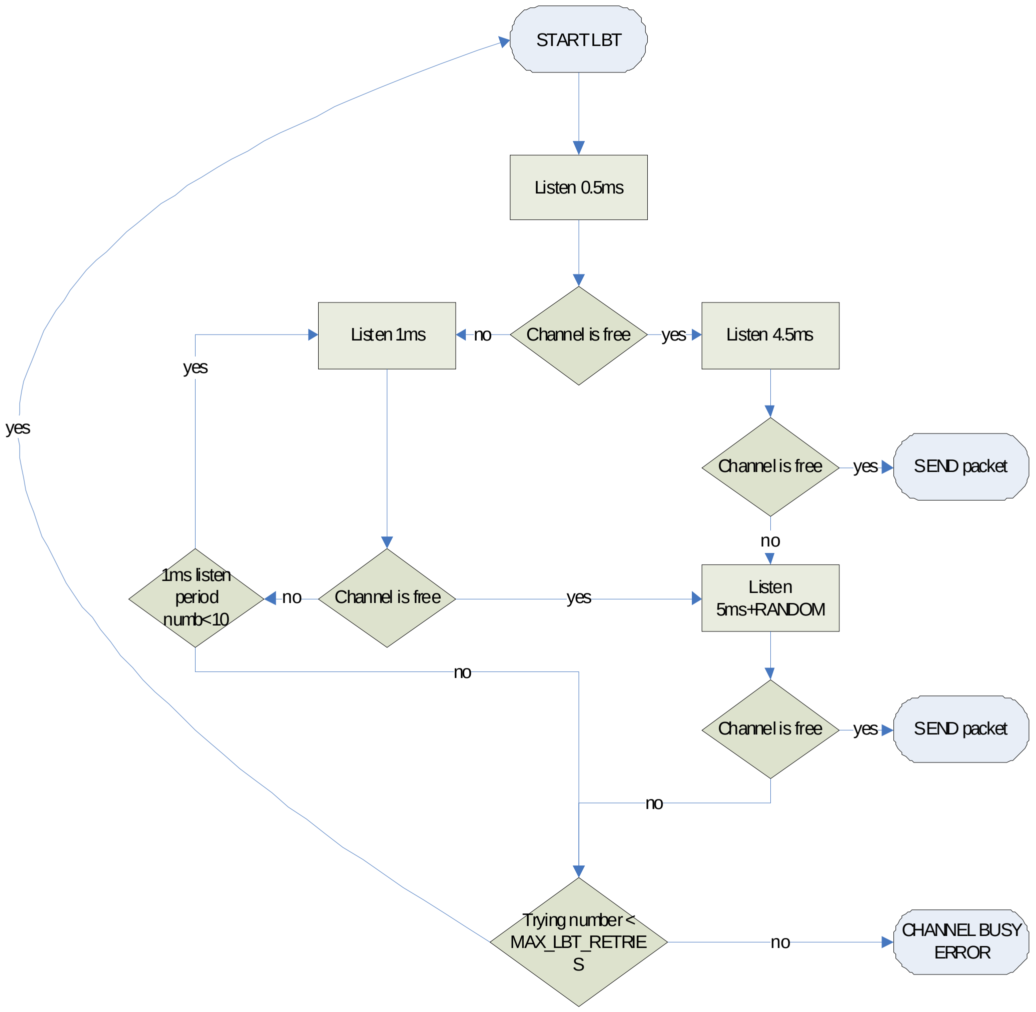


Figure 8. The Listen Before Talk Mechanism

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

$$TPS = n \times LBTI\,[6:0]$$

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 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 the EZMac module or EZHop will send the packet. If the RSSI is above the limit during this time period, the state machine operation will jump to Step 5.
5. If the attempt number is below MAX_LBT_RETRIES, 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 the EZMac module or EZHop are unable to send the packet because a frequency was found to be busy, the stack will go into the TX_ERROR_CHANNEL_BUSY state and call the EZMacPro_LBTTimeout() callback function.

Notes:

1. When sending an acknowledgement packet, neither the EZMac module nor EZHop perform the Listen-Before-Talk operation.
2. If the source code is compiled for a transmitter only device, the Listen-Before-Talk feature is not available.

3.5. Packet Forwarding

The packet forwarding feature is the same for both the EZMac module and EZHop. Packet forwarding enables retransmission of a packet if a node receives a packet that was not sent to that particular node. Using this feature, the packet can be sent to a node that is not in range of its originator; the packet can reach its destination via multiple nodes. Using packet forwarding, the range between nodes (and the coverage of the network) can be increased dramatically.

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:

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

Figure 9. CTRL Field of the Packet

3.5.1. Radius Field

The Radius field specifies how many times can be the packet forwarded. The node that received a packet forwards the packet if the following conditions are met:

- 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 1. The EZMac and EZHop solutions call the void EZMacPRO_PacketForwarding(void) callback function to inform the next higher layer about the packet forwarding. The higher layer can then read and modify the content of the packet payload (e.g. by 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 network indefinitely.

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). The purpose of this is to avoid the retransmission of a previously forwarded packet with a lower RADIUS. All nodes store information (sender ID, packet sequence number, channel number) for the last few received packets in a table, called the Forwarded Packet Table. The number of rows (number of managed packets) in this table is a parameter in the EZMacPRO_defs.h file. This value can be changed prior to the compilation of the source code: FORWARDED_PACKET_TABLE_SIZE and must be less than 16.

The EZMac and EZHop solutions manage the Forwarded Packet Table themselves; the higher software layers maintain the table. The EZMac and EZHop solutions also handle the sequence numbering; it is set to zero after Power-On Reset and incremented each 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 and sequence number of the received packet.

After the node forwards the packet, it makes a new entry in the Forward Packet Table; it then saves the sender ID, the sequence number of the packet, and the channel on which the packet is received.

The table management is very simple; if there is no space for a new entry, the oldest entry will be replaced with the new entry. Additionally, 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 any given node.

Packet forwarding is done automatically; however, the MAC calls the EZMacPRO_PacketForwarding () callback function before the packet is forwarded to allow upper software layers to modify the payload of the packet if the customer needs to add routing information. It is important that the CTRL and sender ID header bytes of the packet remain unchanged. The higher layer can stop the whole process by calling the EZMacPRO_Idle() function.

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

- 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 software layers of the packet reception. If required, the MAC layer sends 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 on which it was received.

Notes:

1. The packet forwarding feature can be enabled or disabled before compilation of the source code. The PACKET_FORWARDING_SUPPORTED symbol can be defined in the IDE or command line.
2. The packet forwarding feature can be used only if the extended packet format is selected. If disabled, the compiler will issue the error message, "Packet forwarding requires the extended packet configuration!".

3.6. Automatic Acknowledgement

The EZMac and EZHop solutions support automatic acknowledgement, which is the same in both modes of operation. If the transmitter wants to get feedback on whether a packet arrived at its destination, it needs to set the ACKRQ bit; this will then set the ACKRQ bit in the CTRL byte of the transmitted packet. After the EZMac module and EZHop transmit the packet, the node goes into a Waiting For ACK state (TX_STATE_WAIT_FOR_ACK) where it waits for the acknowledgement packet. If the receiver node receives the packet that was addressed to it and the ACKRQ bit is set in the control header byte, 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, how the radius field is set, what the actual data rate is configured to be, etc. The timeout is handled by the EZMac module and EZHop automatically and indicates the result of the acknowledgement reception for the next higher layer in the following way:

- Acknowledgement received: It calls the EZMacPRO_PacketSent () call back and goes into the next state defined in SATX[1:0].
- No acknowledgement received within timeout period: It calls the EZMacPRO_AckTimeout() callback and goes into the next stack defined in SATX[1:0].

In either case the higher layer is notified about entering a new state by the respective callback. See Section “6.3. Registers” for more information about API callbacks.

The packet structure of the acknowledgement packet is special in that the ACK bit of the Control byte is set; the source and destination addresses are swapped, and the payload of the packet can be filled with custom data. The API provides the EZMacPRO_Ack_Write() function (see "6.1.11. MacParams EZMacPRO_Ack_Write(U8 length, U8* payload)" on page 33) for this purpose, which can be called from the void EZMacPRO_AckSending(void) callback function. When variable packet length is used, the acknowledgement packet size can be dynamically changed up to ACK_BUFFER_SIZE bytes. The default size is set using the ACK_PAYLOAD_DEFAULT_SIZE definition. If fixed packet length is used, then acknowledgement packets have the same size as regular packets. In this case, if the length of the data to be written into the payload is smaller than the size of the payload, the remaining bytes are filled with zeros.

Every message is acknowledged on the frequency channel on which it is received. The acknowledge packet preamble length is set to four bytes in the EZMac module and also in EZHop if the packet forwarding feature is not compiled.

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

Note: The automatic acknowledgement feature can be enabled using the EXTENDED_PACKET_FORMAT definition.

3.7. Signal Strength Level Indication

The EZRadioPRO devices are capable of measuring the receiver signal strength (RSSI) level. The EZMac and EZHop solutions use this information for the listen before talk mechanism, but the input signal level may also be useful for the higher software layers, if current consumption monitoring is important. The application layers can monitor the RSSI level during packet reception, and, if the signal level is high enough, the receiver can inform the transmitter node to lower the output power. This mechanism allows for the transmit operation to use less power.

The EZMac and EZHop solutions always read the actual RSSI value directly from the radio during receive mode and update the RSSI register accordingly. If the EZMac and EZHop solutions are not in the receive mode, the RSSI register holds the last RSSI information.

3.8. Low Frequency Periodic Timer Support

The EZMac and EZHop solutions can provide a periodic, low-frequency time base for the higher software layers. This is based on the Wake Up Timer of the EZRadioPRO radio. The Wake Up Timer can have two clock sources: the internal RC oscillator or an external 32.768 kHz crystal.

If the timer is enabled, then EZRadioPRO calls the void EZMacPRO_LFTimerExpired(void) callback function periodically with the predefined time interval. The body of the callback function is empty and has to be defined by the higher software layers. The callback function is always called from the external interrupt routine; so, it should be considered when defining the operation of the callback function (the processing time of the callback function should be kept 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 has to be 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) is set before enabling the timer.

Clock source is considered a design consideration. The internal RC based or external based 32.768kHz crystal based timer have to following advantage or disadvantages:

- A crystal has better accuracy when compared to an RC based timer. Longer time periods can be created with smaller errors. Due to the smaller drift between the nodes, SLEEP mode can be set to a longer time period in order to save battery power.
- Better accuracies require the external crystal option which adds to the bill of material cost, additionally GPIO0 is used to connect to the crystal and as such it cannot be used for other purposes.

If the internal RC oscillator is selected as a clock source, then the EZMac and EZHop solutions configure 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, then the EZMac and EZHop solutions configure GPIO0 as input pin for the crystal.

3.9. Low Battery Detection

The EZMac and EZHop solutions 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 5bit digital number (VBAT[4:0] bits in the LBDR register). The voltage can be calculated by the following formula:

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

- The upper software layer can define a power supply voltage threshold (LBDT[4:0] bits in the LBDR register) according to the following formula:

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

If the power supply goes below the threshold, then the EZMac and EZHop solutions call the *void EZMacPRO_LowBattery(void)callback* function. The low battery detect function can be enabled by setting the LBDE bit.

4. Operation

4.1. Overview

The EZMac and EZHop solutions are 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 the EZMac and EZHop solutions are 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 10.

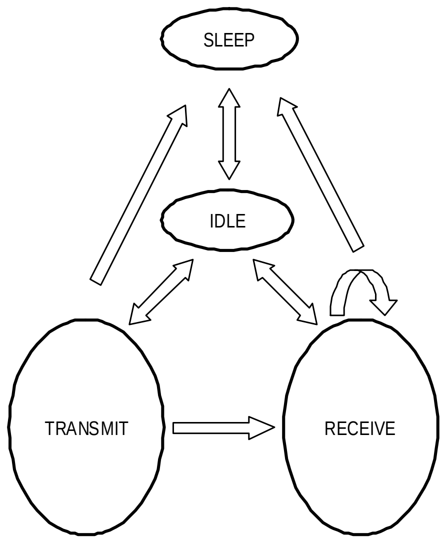


Figure 10. Simplified Block Diagram

4.1.1. SLEEP Mode

After initialization, the stack is in SLEEP mode. In this mode, the RF hardware is completely switched off and consumes less than 1 µA.The RF 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 the EZMac or EZHop solutions with the wakeup command, it switches to IDLE mode. In this mode, the MAC is waiting for further commands. The crystal oscillator runs in the radio, but all RF blocks are disabled, and the current consumption of the radio is about 0.6 mA. The EZMac and EZHop solutions stay in this mode until a RECEIVE, TRANSMIT, or SLEEP command is received.

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 the EZMac or EZHop solutions; this can be done in SLEEP or in IDLE state. After the transmit command has been sent, the stack starts transmission. Listen before Talk is performed before each packet transmission if the ENLBT bit is set.

If the automatic acknowledgement feature is enabled, the EZMac and EZHop solutions automatically wait for the acknowledgement after sending a packet. If the acknowledgement packet arrives within the timeout period, the stack calls the EZMacPRO_PacketSent() callback and automatically goes into the next selected state: IDLE, SLEEP, or RECEIVE. If timeout occurs without receiving the acknowledgement packet, the EZMacPRO_AckTimeout() callback is called before the stack automatically goes into the next state: IDLE, SLEEP, or RECEIVE.

The EZMac and EZHop solutions support the following communication scenarios:

- The transmitter sends the packet on one channel and it does not wait for any feedback from the receiver node as to whether the packet was received correctly or not. This method uses the less current, but the link is less robust: the transmitter just sends the packet once and it does not have information about the result. This method can be used between receive and transmit only devices too.
- The transmitter sends the packet on one channel and wait for acknowledgement from the receiver node. If the acknowledgement does not arrive it retransmits the packet (potentially on a different channel to increase robustness). This method is very robust, since the transmitter always has information about the receiver, but uses more current on both side of the link.

The EZMac module supports the additional communication scenarios:

- If the AFCH bit is set, the packet is transmitted automatically on all the enabled channels one after another. This very simple method can be useful for applications in which the receiver current consumption is important or the receiver node cannot send back acknowledgements (receiver-only device), but high levels of robustness are critical. The intent is for the transmitter to send the same packet on multiple frequencies in order to increase the robustness and probability that the receiver receives at least one of the data packets. The receiver searches on all channels and should receive at least one of the packets. It is not intended for it to receive all the packets. To save battery power, the receiver does not send back an acknowledgement.

Notes:

1. 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.
2. Automatic Frequency Change is not available in EZHop.

Upon finishing the transmission(s), the EZMac and EZHop solutions automatically go into one of the following states, depending on the *SATX[1:0]* bits: IDLE, SLEEP, or RECEIVE.

4.1.4. RECEIVE Mode

If a Receive command is issued in IDLE state, the stack will transition to RECEIVE state where it 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 stack goes into the selected mode: IDLE, SLEEP, or RECEIVE.

If the packet forwarding feature is enabled and the node has to forward the packet, 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 goes back into RECEIVE state automatically.

If the receiver has to acknowledge the received packet, the EZMac and EZHop solutions automatically send back the acknowledgement without performing a listen before talk before the packet. After finishing the acknowledgement transmission, the EZMac and EZHop solutions automatically go into the selected mode: IDLE, SLEEP, or RECEIVE.

4.2. Packet Filtering Method

There is an intelligent packet filtering feature implemented in the EZMac and EZHop solutions. The filter can be configured by several options to best fit the demands of the upper software layers. Once the EZMac and EZHop solutions are configured, the upper layers will send and receive only the payloads. The EZMac and EZHop solutions take care of all the packet building and unpacking tasks. CRC is calculated by the radio, so the upper layer will get only error-free packets.

The flowchart of packet filtering is shown in Figure 11.

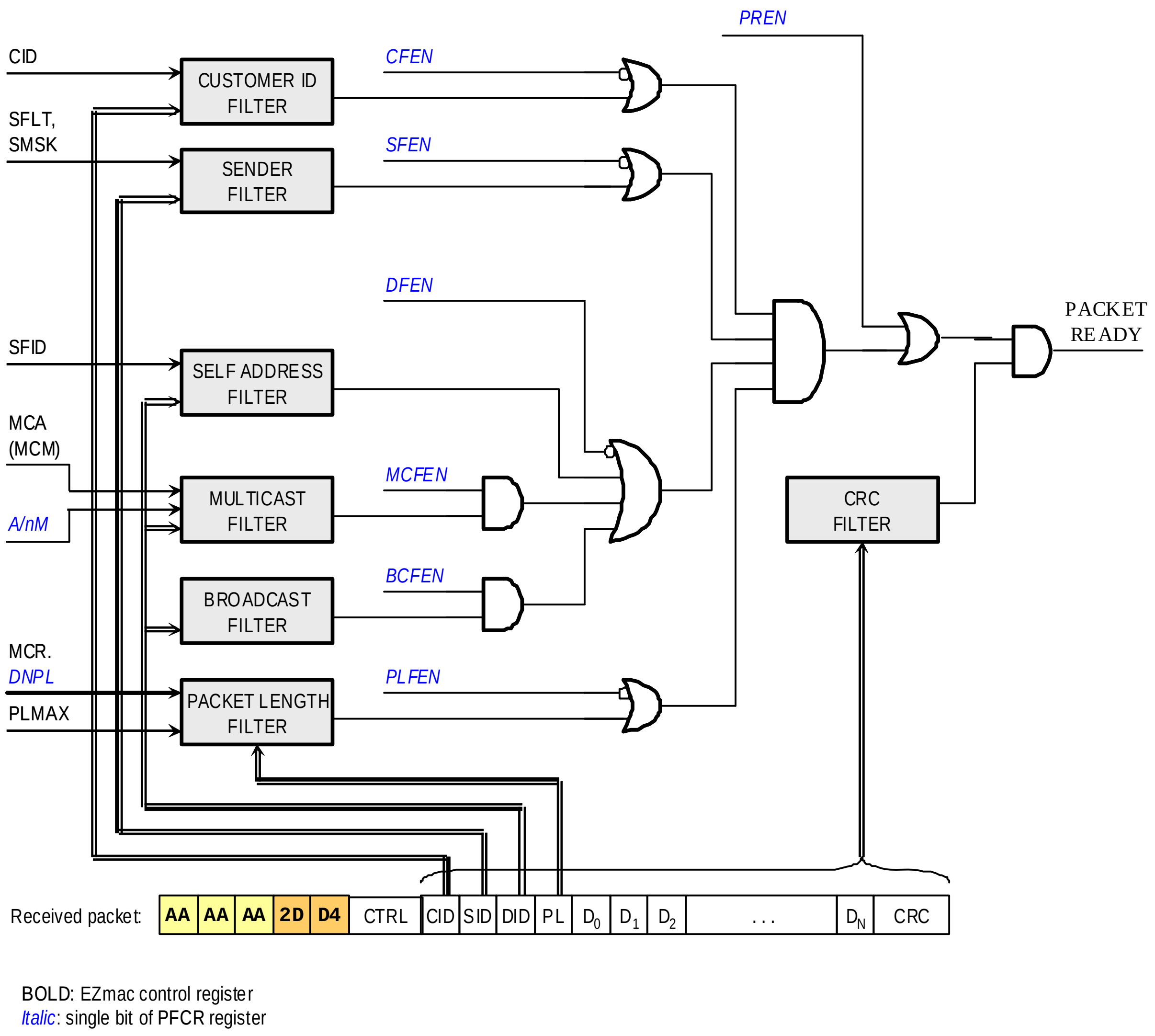


Figure 11. Packet Filtering

4.2.1. Packet Filter Operation

The packet filtering algorithm is run after packet reception. If any segment fails in the corresponding enabled filter, the packet is discarded, EZMacPRO_PacketDiscarded() callback is called, and the stack is placed into the next state configured by SARX[1:0] bits of SECR register.

4.2.2. Customer ID Filter

CID is a unique ID for each customer using the EZMac and EZHop solutions protocol. Using CID is optional but strongly recommended to avoid unexpected interactions between different systems using the EZMac and EZHop solutions that may be installed within the same area. If CID is used, this is transmitted shortly after the preamble and synchron pattern. The EZMac and EZHop solutions 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.

4.2.3. Sender Filter

The EZMac and EZHop solutions filter the incoming packet by testing the sender ID field in the header of the received packet. If the Sender Filter is enabled, it accepts 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 three address filters, all of them testing the DID byte of the received packet header:

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

The EZMac and EZHop solutions perform all the enabled destination filtering on the received packet. If the packet passes any of the enabled filters, it also passes the destination filter.

4.2.4.1. Self Address Filter

Self address (SFID) is used to uniquely identify a node 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 whether the node that received the packet is a member of this group. There are two methods 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.

Notes:

1. 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.
2. If the Packet Forwarding feature is used, then 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, the EZMac and EZHop solutions will ignore the packet.

4.3. Error Detection Method

The EZMac solution can detect several types of errors, which helps the upper software layer select the best data transmission strategy. The EZMac solution 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.

Note: Error detection is available for the EZMac solution only and is not available in EZHop mode.

4.3.1. Channel is Busy (Collision)

If the Listen before Talk (LBT) feature and the Channel Busy error counter are enabled, the EZMac solution 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, the EZMac solution is able to detect (at an early stage of a packet reception) whether the packet is an in-system packet or a packet transmitted by a third-party product using the EZRadioPRO chipset. Since the CID is one of the first bytes of the packet header, the EZMac solution can quickly abort the reception and continue to scan the available frequencies for a valid packet.

4.3.3. Bad Address

In case the address bytes fail on the address filter logic, the EZMac solution 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, the EZMac solution will create a bad CRC error and disregard the packet. Scanning for a valid packet will continue. This type of error gives useful information about the quality of the RF link.

A large number of CRC errors indicate a low-quality link. Detecting other types of errors (but only a few CRC errors) indicates a good link, but with one of the following two conditions:

- Significant third party transmission
- Significant in-system collision, meaning a bad communication strategy.

5. State Machine

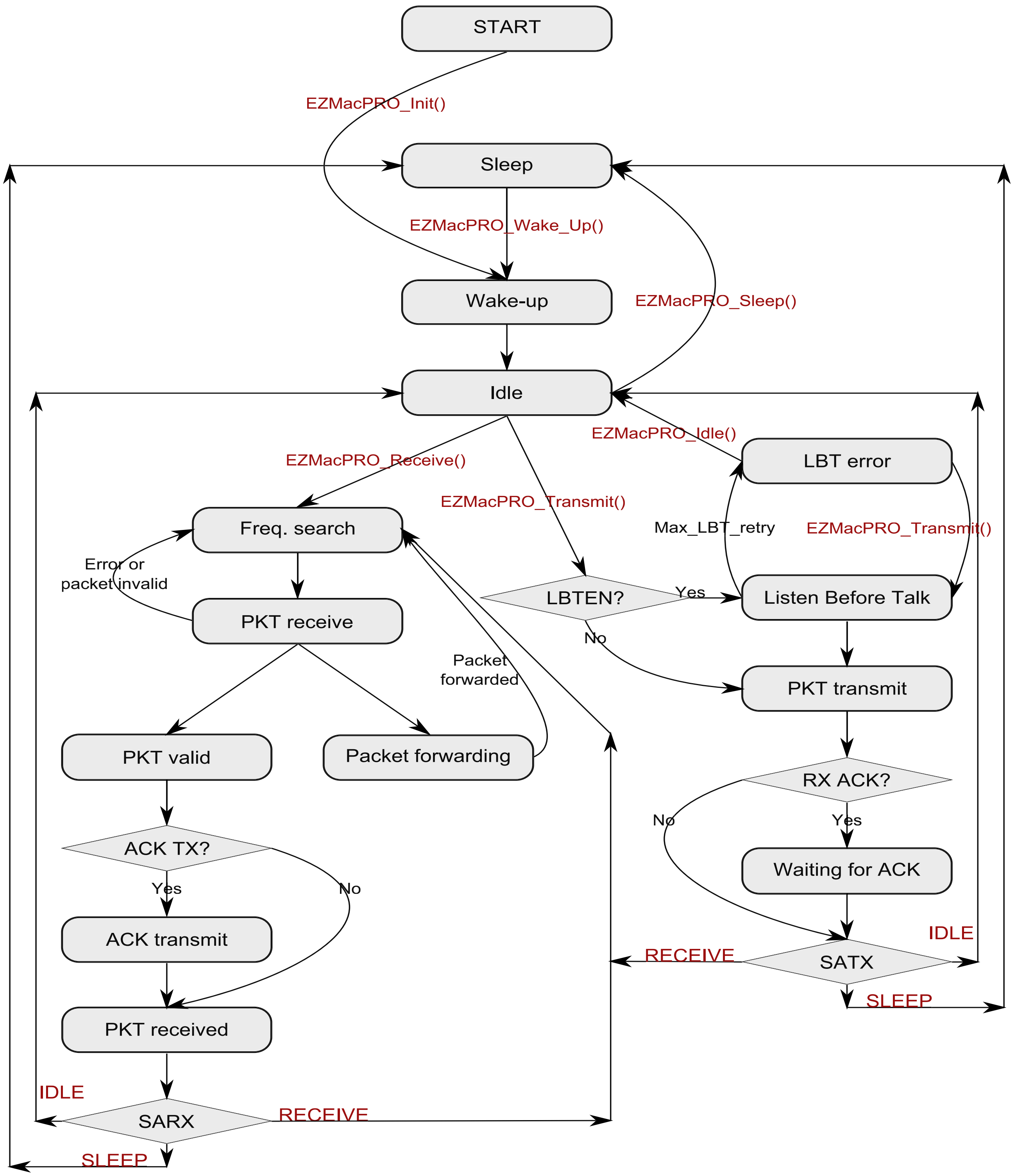


Figure 12. State Machine

Note: The high level state machine is the same for both the EZMac and EZHop solutions.

6. Application Programming Interface

The upper software layer can interact with the EZMac or EZHop solutions through API function calls and gets notified about certain more important events through API callback functions. This section gives detailed information about the Application Programming Interface of the stack.

6.1. API Commands

The following commands are provided with the higher layer to interact with EZMac/EZHop:

- *MacParams EZMacPRO_Init(void)*
- *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)*
- *MacParams EZMacPRO_Ack_Write(U8 length, U8* payload)*

The stack uses only a few resources of the microcontroller: the SPI1 peripheral (handled by the EZMac and EZHop solutions—the upper SW layer does not need to take care of that), the Timer3, and an external interrupt source. The Timer3 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(timerInt3_ISR, INTERRUPT_TIMER3)*
- *INTERRUPT(externalIntISR, INTERRUPT_INT0)*

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

Note: The application programming interface is the same for both the EZMac and EZHop solutions; however, there are few differences in the registers.

6.1.1. MacParams EZMacPRO_Init(void)

Function:	Initializes the EZRadioPRO device and places the stack into SLEEP state. If the initialization was successful, the EZMacRadio_StateWakeUpEntered() and the EZMacPRO_StatesleepEntered() callbacks are sequentially called.
Returns:	MAC_OK: The operation was successul.
Requires:	The function has to be called in the power-on initializing routine.

6.1.2. MacParams EZMacPRO_Wake_Up(void)

Function:	Switch the stack from SLEEP mode into IDLE mode. Turns on the crystal oscillator of the radio; so, the current consumption increases. Upon success the EZMacPRO_StateWakeUpEntered() and the EZMacPRo_StateIdleEntered() callbacks are sequentially called.
Returns:	**MAC_OK**: The operation was successful.
	STATE_ERROR: The operation was ignored because the MAC was not in SLEEP mode.
Requires:	The MAC has to be in SLEEP mode when calling the function.

6.1.3. MacParams EZMacPRO_Sleep(void)

Function:	Switch the MAC from IDLE to SLEEP mode. It turns off the crystal oscillator of the radio. Upon success the EZMacPRO_StateSleepEntered() callback is called.
Returns:	**MAC_OK**: The operation was successful.
	STATE_ERROR: The operation was ignored because the MAC was not in IDLE mode.

Requires: The MAC has to be in IDLE mode.

6.1.4. 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. Upon calling this function, the EZMac and EZHop solutions go into IDLE state and the EZMacPRO_StateIdleEntered() gets called.

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.

6.1.5. MacParams EZMacPRO_Transmit(void)

Function: It starts to transmit a packet. After successful execution the stack goes to Transmit state and calls the EZMacPRO_StateTXEntered() callback. Once the packet is succesfully transmitted, the EZMacPRO_PacketSent() callback is called, then the next state is SATX[1:0] is entered and its corresponding state transition callback is also called.

Returns: **MAC_OK**: The transmission started correctly.

STATE_ERROR: The operation was ignored (the transmission has not been started) because the EZMac and EZHop solutions were not in IDLE mode.

Requires: All the parameters have to be set before calling this function. The EZMac and EZHop solutions have to be in IDLE mode when calling this function.

6.1.6. 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. After successful execution the stack goes to Receive state and calls the EZMacPRO_StateRXEntered() callback. Once a valid packet is received, either EZMacPRO_PacketDiscarded() or EZMacPRO_PacketReceived() is called before the next state in SARX[1:0] is entered and its corresponding state transition callback is entered. The search can be stopped by the *EZMacPRO_Idle()* function.

Returns: **MAC_OK**: The search mechanism started correctly.

STATE_ERROR: The operation was ignored (the search mechanism has not been started), because the EZMac and EZHop solutions were not in IDLE mode.

Requires: All the parameters have to be set before calling this function. The EZMac and EZHop solutions have to be in IDLE mode when calling this function.

6.1.7. MacParams EZMacPRO_Reg_Write(MacRegs name, U8 value)

Function: It writes **value** into the register identified by **name**. MacRegs type is predefined; the names of the available registers are listed in "6.3.1. Register Summary" on page 37. The function also takes care of updating the settings of the radio chip if needed. This function may also be called in SLEEP mode.

Returns: **MAC_OK**: The register is set properly.

NAME_ERROR: The register name is unknown.

VALUE_ERROR: The **value** is out of the range of the register.

STATE_ERROR: The operation is ignored because transmission and reception are in progress.

INCONSISTENT_SETTING: If the EZMac solution or EZHop change to register values that are not supported by the data rate, then the value of the inconsistent frequency ID is automatically changed to 0.

Requires: None of the registers could be set during packet transmission or reception!
See “6.3.1. Register Summary” for more details.

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

Function:	Gives back the value (over the **value** pointer) of the register identified by **name**. MacRegs type is predefined; the names of available registers are listed in “6.3.1. Register Summary”. 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.

6.1.9. 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 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 the EZMac solution and EZHop copy **length** number of **payload** bytes into the TX FIFO of the radio, set PLEN register of the EZMac and EZHop solutions, and set the packet length register of the radio. If fix packet length is used, then PLEN register has to be set first because the function copies only Payload Length number of bytes into the FIFO even if the **length** is greater. It also 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.

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

Function:	After a successful packet reception, the EZMac and EZHop solutions copy 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 cannot be greater than 64 bytes. 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**. It also gives back the number of received bytes by **length**.
Returns:	**MAC_OK**: The operation performed correctly.
	STATE_ERROR: The operation is ignored because reception is in progress.
Requires:	Nothing.

6.1.11. MacParams EZMacPRO_Ack_Write(U8 length, U8* payload)

Function:	The function copies length number of bytes into the acknowledgement buffer. The acknowledgement buffer is declared in the EZMacPro.c file as: SEGMENT_VARIABLE(AckBuffer[ACK_BUFFER_SIZE], U8 , BUFFER_MSPACE); The length of the acknowledge buffer is defined by the ACK_BUFFER_SIZE definition in the EZMacPro_defs.h. It can be adjusted for the application needs, but it cannot be greater than 64 bytes. Its default value is 16 bytes. The buffer is declared to be placed into the XDATA memory, which can be adjusted by changing the BUFFER_MSPACE definition.
	If variable packet length is used and length is not greater than ACK_BUFFER_SIZE (which cannot be greater than 64), then the EZMac/EZHop solutions copy length number of bytes into the acknowledgement buffer, set PLEN register of the EZMac/EZHop solutions and set the packet length register of the radio. If fix packet length is used, then PLEN register has to be set first because the function can copy only PLEN or ACK_BUFFER_SIZE number of bytes (whichever is smaller) into the buffer even if the length is greater. It also fills the buffer with extra 0x00 bytes if the length is smaller than the payload size.
Returns:	**MAC_OK**: Operation successfully finished.
	STATE_ERROR: The operation is ignored as the statemachine is not in any of the receive states.
	VALUE_ERROR: The length parameter is greater than the size of the acknowledgement buffer.
Requires:	The function can only be called in receive states.

Note: It is advised to define ACK_BUFFER_SIZE as small as possible to keep the memory footprint of the EZMac/EZHop low.

6.2. API Callbacks

The EZMac and EZHop solutions can provide feedback for the higher layer by calling callback functions. The following callback functions are implemented:

- void EZMacPRO_StateWakeUpEntered(void)
- void EZMacPRO_StateSleepEntered(void)
- void EZMacPRO_StateIdleEntered(void)
- void EZMacPRO_StateRxEntered(void)
- void EZMacPRO_StateTXEntered(void)
- void EZMacPRO_StateErrorEntered(void)
- void EZMacPro_LFTimerExpired(void)
- void EZMacPro_LowBattery(void)
- void EZMacPro_SyncWordReceived(void)
- void EZMacPro_PacketDiscarded(void)
- void EZMacPro_PacketReceived(void)
- void EZMacPro_PacketForwarding(void)
- void EZMacPro_PacketSent(void)
- void EZMacPro_CRCError(void)
- void EZMacPro_LBTTimeout(void)
- void EZMacPro_AckTimeout(void)
- void EZMacPro_AckSending (void)

The callback functions are called from interrupt routines of the state machine and can be used to set a flag that signals a certain event towards higher layer. The bodies of the functions can be customized according to higher layer needs. All of these functions are called from interrupt routines, so the processing time has to be kept as short as possible.

6.2.1. void EZMacPRO_StateWakeUpEntered(void)

Function:	The callback function is called when the stack enters Wake-up state.
Returns:	Nothing.
Requires:	Nothing.

6.2.2. void EZMacPRO_StateSleepEntered(void)

Function:	The callback function is called when the stack enters Sleep state.
Returns:	Nothing.
Requires:	Nothing.

6.2.3. void EZMacPRO_StateIdleEntered(void)

Function:	The callback function is called when the stack enters Idle state.
Returns:	Nothing.
Requires:	Nothing.

6.2.4. void EZMacPRO_StateRxEntered(void)

Function:	The callback function is called when the stack enters Receive state.
Returns:	Nothing.
Requires:	Nothing.

6.2.5. void EZMacPRO_StateTxEntered(void)

Function:	The callback function is called when the stack enters Transmit state.
Returns:	Nothing.
Requires:	Nothing.

6.2.6. void EZMacPRO_StateErrorEntered(void)

Function:	The callback function is called when the stack enters Error state.
Returns:	Nothing.
Requires:	Nothing.

6.2.7. void EZMacPRO_LFTimerExpired(void)

Function:	The callback function is called when the Low-Frequency Timer expires. It is called from the external interrupt routine.
Returns:	Nothing.
Requires:	The time period has to be set before enabling the LFT. The corresponding EZMac registers are LFTMR0...2.

6.2.8. void EZMacPRO_SyncWordReceived(void)

Function:	The callback function is called when the synchron word of the packet is received. This is a fixed time in each packet and can be used for time synchronization. The function is called from the external interrupt routine.
Returns:	Nothing.
Requires:	Nothing.

6.2.9. void EZMacPRO_PacketDiscarded(void)

Function:	The callback function is called when EZMac or EZHop received a valid packet that did not pass enabled packet filters and thus was discarded. The function is called from the external interrupt routine.
Returns:	Nothing.
Requires:	Nothing.

6.2.10. void EZMacPRO_PacketReceived(U8 rssi)

Function:	The callback function is called when the EZMac and EZHop solutions receive 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:	Nothing.
Requires:	Nothing.

6.2.11. void EZMacPRO_PacketForwarding(void)

Function:	The callback function is called when the EZMac and EZHop solutions receive 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. decrease the radius field) and packet forwarding management is done by the MAC itself. The function is called from the external interrupt routine.
Returns:	Nothing.
Requires:	Nothing.

6.2.12. void EZMacPRO_LBTTimeout (void)

Function:	The callback function is called if the Listen Before Talk mechanism failed to access to the channel after the maximum number of retries (defined by the MAX_LBT_RETRIES definition). After the EZMac and EZHop solutions call this callback function, the stack goes into TX_ERROR_CHANNEL_BUSY state. EZMacPRO_Idle() command should be used to recover from this error state. The function is called from the timer interrupt routine.
Returns:	Nothing.
Requires:	Nothing.

6.2.13. void EZMacPRO_AckSending(void)

Function:	Acknowledgements are sent only if an ACK was requested and the Sender ID isn't a broadcast address. The callback function is called after the headers of the to be sent acknowledgement packet and the packet length register of the radio are set, and the acknowledgement buffer is cleared. The callback is intended to give the opportunity for the designer to load custom data (using the EZMacPRO_Ack_Write() API function) into the payload field of the acknowledgement packet, before it is followed by immediate transmission.
Returns:	Nothing.
Requires:	Nothing.

6.2.14. void EZMacPRO_PacketSent(void)

Function:	The callback function is called after the EZMac and EZHop solutions transmitted a packet correctly. If acknowledgement is also requested, then the callback function is called after the acknowledgement has arrived. The function is called from the external interrupt routine.
Returns:	Nothing.
Requires:	Nothing.

6.2.15. void EZMacPRO_AckTimeout (void)

Function:	The callback function is called if the EZMac and EZHop solutions did not receive the acknowledgement packet within timeout. After this callback function is called, the EZMac and EZHop solutions go to the next state defined in SATX[1:0]. The function is called from the timer interrupt routine.
Returns:	Nothing.
Requires:	Nothing.

6.2.16. void EZMacPRO_CRCError(void)

Function:	The callback function is called if the EZMac and EZHop solutions received a packet with incorrect CRC. After this callback function is called, the EZMac and EZHop solutions go into the receive state. The function is called from the external interrupt routine.
Returns:	Nothing.
Requires:	Nothing.

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.

6.3. Registers

6.3.1. Register Summary

Table 18. Register Summary

R/W	Name	Description	Default	Comments
R/W	MCR	Master Control Register	0x1C	The NRF[1:0] bits are implemented only in EZMac.
R/W	SECR	State & Error Counter Control Register	0x50	The CB, BCID, BADDR, BCRC bits are implemented in the EZMac solution only.
R/W	TCR	Transmit Control Register	0x38	The AFCH bit is implemented only in EZMac.
R/W	RCR	Receiver Control Register	0x04	The SCHEN bit is implemented in the EZMac solution only.
R/W	FRx*	Frequency Register x	0x00	
R/W	FSR	Frequency Select Register	0x00	T/RF[1:0] in EZMac, and T/RF[5:0] are implemented in EZHop only.
R/W	EC0	Error Counter of Frequency 0	0x00	This register is implemented in the EZMac solution only.
R/W	EC1	Error Counter of Frequency 1	0x00	This register is implemented in the EZMac solution only.
R/W	EC2	Error Counter of Frequency 2	0x00	This register is implemented in the EZMac solution only.
R/W	EC3	Error Counter of Frequency 3	0x00	This register is implemented in the EZMac solution only.
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	RFSR	Received Frequency Status Register	0x00	PRF[1:0] is implemented in EZMac, and PRF[5:0] is implemented in EZHop only.
R	RSSI	Received Signal Strength Indicator	0x00	
R/W	SCID	Self Customer ID	0xCD	
R/W	SFID	Self ID	0x01	

Table 18. Register Summary (Continued)

R/W	Name	Description	Default	Comments
R	RCTRL	Received Control Byte	0x00	
R	RCID	Received Customer ID	0x00	
R	RSID	Received Sender ID	0x00	
R/W	DID	Destination ID	0x00	
R/W	PLEN	Payload Length	0x01	
R/W	LBTIR	Listen Before Talk Interval Register	0x8A	
R/W	LBTLR	Listen Before Talk Limit Register	0x78	
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	
***Note:** There are four frequency registers in the EZMac solution and fifty frequency registers in EZHop.				

Master Control Register

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

CIDE	Customer ID enable. If this bit is set, then the CID header field is added into the packet format.
DR[1:0]	Data rate. This bit selects the actual data rate. If these bits are changed, then the EZMac solution and EZHop automatically update the radio chip with the correct settings for the given data rate.

Note: If the data rate is changed and the previously-selected Frequency ID is not supported by the new data rate, then 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, then the payload length field is included into the transmitted packet, and the receiver also expects to receive the payload length field in the header of the incoming packet. If this bit is cleared, then the payload length field is not transmitted in the packet, and fix packet length is 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.).

Note: The NRF[1:0] control bits are available in the EZMac solution and are not used in EZHop.

6.3.1.1. 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 the EZMac and EZHop solutions finish a successful packet transmission:

- If the automatic acknowledgement feature is disabled, then the EZMac and EZHop solutions perform the automatic state changes if the packet transmitted.
- If the automatic acknowledgement feature is enabled, then the EZMac and EZHop solutions perform the automatic state changes even if no acknowledgement is received before timeout.

Table 19. 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 the EZMac and EZHop solutions receive a packet successfully:

- If the automatic acknowledgement feature is disabled, then the EZMac and EZHop solutions perform the automatic state changes after the packet reception.
- If the automatic acknowledgement feature is enabled, then the EZMac and EZHop solutions perform the automatic state changes only after the acknowledgement packet is transmitted correctly.
- If the EZMac and EZHop solutions forward a packet, it always goes back to receiving mode.

Table 20. SARX 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.
BADDR	Bad Addresses. If this bit is set, then the error counters count the bad address errors.
BCRC	Bad CRC. If this bit is enabled, then the error counters count the bad CRC errors.

Notes:

1. See "4.3. Error Detection Method" on page 28 for more details.
2. CB, BCID, BADDR, BCRC control bits are not available in EZHop.

6.3.1.2. Transmit Control Register

TCR							
7	6	5	4	3	2	1	0
ACKRQ	OP[2:0]			LBTEN	AFCH	Reserved	

ACKRQ	Acknowledgement Request. If the automatic acknowledgement feature is compiled into the source code and this bit is set, then the transmitter set 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 21. Output Power Settings

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

LBTEN Listen Before Talk Enable. If this bit is set, then the EZMac and EZHop solutions perform Listen Before Talk before all packet transmission. It will not be performed in the following cases even if this bit is set:

- Before the EZMac and EZHop solutions transmit the automatic acknowledgement packet.
- Before packet transmission if the AFCH bit is set.

AFCH Automatic Frequency Change. If this bit is set, then 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, then the packet is transmitted only once on the frequency defined by T/RF[1:0] (T/RF[5:0] in Frequency Hopper extension) bits.

Notes:

1. Automatic Frequency Change is not available in EZHop
2. The Si4x32 revision V2 has fewer output power steps. In this case, the OP[1:0] bits can be used for the power setting. The maximum output power is +20 dBm and the minimum is +11 dBm.

6.3.1.3. Receive Control Register

RCR							
7	6	5	4	3	2	1	0
PFEN	FMASK3	FMASK2	FMASK1	FMASK0	SCHEN	Reserved	

PFEN	Packet Forwarding Enable. If the packet forwarding feature is compiled into the source code and this bit is set, then the EZMac and EZHop solutions forwards all packets what is intend to be forwarded (see "3.5. Packet Forwarding" on page 20 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).

Notes:

1. 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.
2. The FMASKx control bits don't effect in EZHop, because Frequency Mask feature is not implemented in EZHop.

SCHEN	Search enabled. If this bit is set, then the EZMac solution uses the frequency search mechanism, and it searches on all enabled, not masked, frequencies for packets. After enabling the receiver the EZMac solution starts looking for transmission on the first enabled channel. If this bit is cleared, then the EZMac solution waits for transmission on a frequency determined by T/RF[1:0] (T/RF[5:0] in EZHop) bits.

Notes:

1. The SCHEN control bit is not available in EZHop.
2. EZHop always uses the frequency search mechanism.

6.3.1.4. Frequency Registers

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

These registers set the channel number used as frequency 0…3 in case of the EZMac solution and 0...49 in case of EZHop. 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 can be found in the EZMacPro_const.c 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 Fx = 0 (e.g. if the maximum number of channels is 4, then the valid range for the frequency ID is 0…3).

Using EZMac, up to four channels are available, and the number of used channels is programmable via the EZMac solution register during run time. The default frequency assignment of the EZMac solution for different frequency bands is listed in Table 22 (it can be modified for the application needs).

Table 22. 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
9.6	433.33	330	4
50	433.61	360	2
128	433.91	0	1

Table 23. 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
9.6	863.55	450	14
50	863.75	780	8
128	864	1 MHz	6

Table 24. 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	903.00	480	60
9.6	903.00	480	60
50	903.00	780	37
128	903.00	1 MHz	29

Using EZHop, the number of used channels is fixed at 50. The default frequency assignment of EZHop for the 915 MHz ISM band is the following (it can be modified for the application needs).

Table 25. Frequency Assignment for the 915 MHz ISM Band

915 MHz ISM Band			
Data Rate (kbps)	**Start Frequency (MHz)**	**Frequency Step Size (kHz)**	**Number of Used Channels**
2.4	903.00	480	50
9.6	903.00	480	50
50	903.00	480	50
128	903.00	480	50

According to the FCC regulation, it is desirable to change the transmit frequency pseudo random order. EZHop fulfills this requirement if the channel numbers are listed in the channel table in a pseudo-random way, and the Frequency Select register is incremented after every packet transmission.

Notes:

1. The default values of the Fx registers are F0 = 0, ..., F3 = 3 in the case of EZMac.
2. The default values of the Fx registers are set according to the frequency table (see "3.2.1. Frequency Assignments" on page 15 for more details).
3. If a greater value is written into the register than the maximum ID of the available frequencies, then the stack ignores the setting, and the EZMacPRO_Reg_Write() function returns with VALUE_ERROR.
4. The default frequency allocations for all data rates and frequency bands are defined to comply with the ETSI and FCC standards.

6.3.1.5. Frequency Select Registers

FSR							
7	6	5	4	3	2	1	0
Reserved						T/RF[1:0]	

T/RF[1:0]	Transmit/Receive Frequency. These bits have an effect only if the AFCH bit is cleared. These bits define the frequency where the packet is transmitted in the case of EZMac. If the SCHEN bit is cleared, then these bits define the frequency where the receiver waits for transmission. If the SCHEN bit is set, these bits do not have any effect.

Note: Using EZHop, T/RF[5:0] is six bits wide (supporting the 50 channels).

6.3.1.6. Error Counters

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

Error Counter of channel 0…3.

There are several types of errors that the EZMac solution can detect during packet transmission and reception. If an error is detected and the corresponding error detection is enabled, then 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 28 for more details.

Note: Error counters are only available in the case of EZMac.

6.3.1.7. Packet Filter Control Register

PFCR							
7	6	5	4	3	2	1	0
CFEN	SFEN	DFEN	MCFEN	BCEN	PLFEN	PREN	A/nM

This register controls the incoming packet filter of the EZMac and EZHop solutions. See "4.2. Packet Filtering Method" on page 26 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 be received too. This bit is ignored if the destination filter is disabled.
BCEN	If this bit is set, then broadcast packages can be received too. This bit is ignored if the destination filter is disabled.
PLFEN	If this bit is set, the Packet Length filter is enabled.
PREN	If this bit is set, the EZMac and EZHop solutions are set into Promiscuous mode. In this mode, all the filters are ignored, it receives all packets with valid CRC.
A/nM	Selects the mode of the Multicast Filter. 1: Multicast Address mode; 0: Multicast Mask mode.

6.3.1.8. 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]							

The EZMac solution 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

6.3.1.9. 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 differently 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)

The EZMac and EZHop solutions filter the incoming packet by testing the destination ID field in the header of the received packet. If the Multicast Address Filter is enabled, then the EZMac and EZHop solutions will accept packets coming with MCA at the destination ID header field of the received packet.

Multicast Mask Mode (A/nM = 0)

The EZMac and EZHop solutions filter the incoming packet by testing the destination ID header field of the received packet. If the Multicast mask filter is enabled, then the EZMac and EZHop solutions will accept packets if:

DID & MCm == SFID & MCM

Where:

- & bitwise AND operator
- DID: 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.

6.3.1.10. 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.

6.3.1.11. MAC Status Register

MSR							
7	6	5	4	3	2	1	0
Internal States							

The MAC status register gives information about the current state of the EZMac and EZHop solutions. The register holds the actual state of the state machine.

Table 26. States of the State Machine

EZMac 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_CHANNEL_BUSY	Transmit	—	—	Low
TX_ERROR_STATE	Transmit	—	—	Low
RX_STATE_FREQUENCY_SEARCH	Receive	Enabled	Enabled	High
RX_STATE_WAIT_FOR_PREAMBLE$^{\text{Note1}}$	Receive	Enabled	Enabled	Low
RX_STATE_WAIT_FOR_SYNC	Receive	Enabled	Enabled	Low
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
***Note:** This state is available only in EZHop.				

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

Table 27. Main States

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

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

6.3.1.12. Receive Status Register

RSR							
7	6	5	4	3	2	1	0
SELFA	MCA	BCA	-	-	-	-	-

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.

6.3.1.13. Received Frequency Status Register

RFSR							
7	6	5	4	3	2	1	0
-	-					PRF[1:0]	

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

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

Note: PRF[5:0] is six bits wide in case of EZHop.

6.3.1.14. 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, the EZMac and EZHop solutions measure 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 for information about how the RSSI value is related to the input signal strength.

6.3.1.15. 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.

6.3.1.16. 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.

6.3.1.17. 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.

6.3.1.18. 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.

6.3.1.19. 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.

6.3.1.20. 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.

6.3.1.21. 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 needing to be transmitted. In fix packet length mode, the register defines the length of the packet both for transmitting and receiving modes.

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

6.3.1.22. 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 requirements of the EZMac and EZHop solutions 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. The value of the definition cannot be greater than 64, or else 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 is 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.

6.3.1.23. 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 fixed 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 where:

- n is a random number between 0…15.
- LBTI[6:0] is the Listen Before Talk Interval.
  LBTI can be set in byte intervals (1…127 bytes interval) or it can be set in fixed 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 follows, the byte interval might be sufficient.

Note: See "3.4. Listen Before Talk" on page 19 for more details.

6.3.1.24. Listen Before Talk Limit Register

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

The RSSI[7:0] bits define the RSSI limit for the Listen Before Talk mechanism. If the input power is above the limit, then the channel is considered occupied. If the input power is below 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.

6.3.1.25. Low Frequency Timer Setting Registers

The Wake Up Timer settings are different for EZRadioPRO revision A0, revision V2, and revision B1. Therefore, the LFTMR0...2 registers have to be interpreted in a different way.

In case of EZRadioPRO revision A0 and revision V2:

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)

The time period of the Wake Up Timer is calculated as follows:

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

In case of EZRadioPRO revision B1:

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[4:0]				

WTM[15:0]	Wake Up Timer Mantissa
WTR[4:0]	Wake Up Timer Exponent (R)

The time period of the Wake Up Timer is calculated as follows:

$$T_{WUT} = \left(\frac{4 \times WTM \times 2^{WTR}}{32.768}\right)[ms]$$

Note: Valid value of WTR[4:0] is 0 ... 20 decimal.

Common control bits for all the revisions:

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 a clock source. If this bit is cleared, then the Wake Up Timer uses the external 32.768kHz watch crystal. The external crystal has to be connected to GPIO0 of the EZRadioPRO device.

Note: If the external crystal mode is selected, then the EZMac and EZHop solutions automatically change the functionality of the GPIO0 pin to serve the external crystal. If IETB bit is cleared, then the EZMac and EZHop solutions set the functionality of GPIO0 as defined by the GPIO0_FUNCTION definition.

LFTMRE Setting this bit enables the Wake Up Timer of the radio. The EZMac and EZHop solutions then call the void EZMacPRO_LFTimerExpired(void) callback function periodically every time the wake Up Timer expires.

Note: The LFTMR0 and LFTMR1 registers have to set before the LFTMRE bit is set because the Wake Up Timer is configured only if that bit is changes from 0 to 1. Simply changing these registers will not have any effect on the time period.

6.3.1.26. 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.
- The EZMac and EZHop solutions call the void EZMacPRO_LowBattery(void) callback function in case the power supply voltage of the radio is below that of 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:

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

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

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

Note: After enabling the Low Battery Detect circuit of the radio, 500 ms waiting time is needed before the first measurement is done.

6.3.1.27. 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. The AD conversion takes about 350 µs.

Table 28. 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

Table 29. ADC Reference Voltage Selection

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

Table 30. 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 can only be written in IDLE mode.
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.

6.3.1.28. 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 and read out.

The proper usage of this register is as follows:

1. Start the conversion with the ADCTSR register.
2. Wait around 400 µs.
3. Read out the ADCTSV register.

Document Change List

Revision 2.0 to Revision 2.1

- Updated Figures 3, 4, and 5.
- Updated Table 9.
  - EZHOP can use 50 channels in all data rates.
- Updated Tables 11 and 12.
- Removed Table 10.
- Removed Tables 17, 18, 28, and 29
  - EZHOP can work only the 915 MHz band
- Added Si10xx motherboard platform support.
- Added hardware platform definitions.
- Added Raisonance compiler support.

Revision 2.1 to Revision 2.2

- Added support for custom data in the Acknowledgement packet.

Revision 2.2 to Revision 3.0

- Updated document to better reflect state of the stack.
- Added new API callback functions.
- Updated Figure 12, “State Machine,” on page 29.
- Corrected “2. Hardware Requirements”.

Revision 3.0 to Revision 3.1

- Updated Figure 12, “State Machine,” on page 29.
- Added bit information to CFEN and BCEN in “6.3.1.7. Packet Filter Control Register”.

Revision 3.1 to Revision 3.2

- Updated Section 2.2.2.
- Updated Section 4.2.1.

NOTES:

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.