
TN8000.18

Technical Note

***XE8000 driving XE1200 transceivers
Standard API definitions***

Table of Contents

1	Introduction	3
1.1	Goal of the document.....	3
1.2	SEMTECH API code.....	3
1.3	Hardware Interfaces.....	3
1.4	Document conventions.....	4
2	Data types	4
2.1	Data type definitions.....	4
3	Global variables	4
3.1	Variables common to all XE120xDivers.....	4
3.2	BitJockey™ specific variables.....	5
3.3	BitBang specific variables.....	5
3.4	SX1223 specificities.....	5
3.5	SX1211 specificities.....	5
3.6	Copy of transceiver read/write registers.....	5
4	Header files – contents	6
4.1	Global definitions.....	6
4.2	I/O ports definitions.....	6
4.3	Transceiver 3 wire serial interface macros definitions.....	6
4.4	Transceiver operating modes and bit definitions.....	6
4.5	Timing values definitions.....	6
4.6	BitJockey™ operating modes and bit definitions.....	6
5	API usage tutorial	7
5.1	Getting familiar with XE8000 software environment.....	8
5.2	Setting up the project.....	8
5.3	Initializing the microcontroller.....	8
5.4	Initializing transceiver parameters.....	9
5.5	Declaring the application global variables.....	9
5.6	Implementing the main function.....	10
5.7	Implementing the master function.....	11
5.8	Implementing the slave function.....	12
5.9	Testing the application.....	12
6	API functions	13
6.1	Configuration functions.....	13
6.2	Communication functions.....	15
6.3	Transceiver specific functions.....	19
6.4	Utility functions.....	21
7	Appendix	24
7.1	API functions availability.....	24
7.2	Hardware connections implemented for described drivers.....	25
7.3	Glossary.....	32

1 INTRODUCTION

1.1 Goal of the document

The purpose of the Application Program Interface (API) is to provide basic software drivers for Semtech transceivers to XE8000 developers. This API allows a quick evaluation of the XE1200 transceivers and aims to ease the start of a development, using standard calls to low level functions.

All functions are open source but not fully optimized, however developers can replace them, if necessary, with optimized object code from third party vendors through wrapper functions.

The API allows developers to evaluate performance and electrical characteristics of the transceiver and to subsequently modify or optimize the code to meet their design requirements.

A complete list of the API functions applicable to the Semtech XE1200 series transceivers family can be found in Appendix 7.1.

Note: please refer to the Semtech website at <http://www.semtech.com> to obtain the latest documentation available. Semtech welcomes feedback from developers to help ensure that customer's requirements are met.

1.2 SEMTECH API code

1.2.1 API current version

This document is related to the version 2.4 of the API functions.

1.2.2 API programming language

The API functions are written in C code. These functions can be optimized by rewriting in assembler.

1.2.3 Integrated Development Environment (IDE) used to implement and test the API

The code described in this API has been written under RIDE-Pro.

1.2.4 Files names

There are always two files per transceiver, a "C" file and an "H" file i.e. *XE120xDriver.c* and *XE120xDriver.h*, the C file contains the API functions implementation and the H file contains the API functions prototypes, #define directives and macros.

1.3 Hardware Interfaces

The API has been implemented for the following hardware interfaces:

1.3.1 BitBang

The BitBang technique consists of the transmission of data in a serial format, accomplished by rapidly toggling, in **software**, a single output bit. It is used with the "Continuous" data mode of the RF chip.

1.3.2 The BitJockey™ (all RF chips except SX1211)

The BitJockey™ is a PCM Bit Stream Encoder/Decoder or RF Receiver/Transmitter Interface. In a normal microcontroller, the bits and low level coding of the RF protocol are handled by software. This is both time-consuming and code inefficient, since the processor has to handle each bit as it is received or as it is about to be transmitted. The BitJockey™ simplifies the handling of this low level data for wireless data systems, using techniques similar to that of a UART interface for wired transmission systems. It is also used with the "Continuous" data mode of the RF chip.

For more information concerning the implementation of the BitJockey™, please refer to the Semtech XE8806A and XE8807A datasheet.

1.3.3 Buffered (XE1205 and SX1211) and Packet (SX1211 only)

A FIFO on the RF chip is used to store each data byte transmitted or received (only the payload bytes in the case of Packet mode). This data is written to/read from the FIFO via the SPI bus. It reduces processor overhead and reduces connections (the DATA input/output pin is not used in this operation mode). It is used with the "Buffered" and "Packet" data modes of the RF chip.

For more information concerning the implementation of the Buffered and Packet modes, please refer to the Semtech XE1205 or SX1211 datasheet.

1.4 Document conventions

- A file name is represented with italic characters. i.e. *XE1202Driver.h*
- Part of code or code example is written in courier font. i.e. `#define SLEEP 0`
- Defined values are always represented in upper case i.e. `#define SLEEP 0`
- Global variables name start with capital characters.
i.e. `static _U8 RFState = RF_STOP; // RF state machine`
- Local variables name start with lower case characters.
i.e. `_U8 mode;`
- Pointers variable name start with a lower case “p” character
i.e. `static _U8 *pRFFrame; // Pointer to the RF frame`

2 DATA TYPES

2.1 Data type definitions

All the data types described in this chapter are defined in *types.h*. This file is automatically included by the standard header file of the microcontroller being used (ex: *XE88LC06A.h*).

The table below describes the basic types used within the API.

Standard type name	Defined type name	Comment
bool (unsigned char)	<code>_U8</code>	false/true - FALSE/TRUE – 0/1
unsigned char	<code>_U8</code>	8 bit quantity
char	<code>_S8</code>	8 bit signed quantity
unsigned short	<code>_U16</code>	16 bit quantity
short	<code>_S16</code>	16 bit signed quantity
unsigned long	<code>_U32</code>	32 bit quantity
long	<code>_S32</code>	32 bit signed quantity
float	<code>_F24</code>	24 bit float quantity
double	<code>_F32</code>	32 bit float quantity

Table 1 Types definition

3 GLOBAL VARIABLES

API global variables are declared in *XE120xDriver.c*, these variables are necessary to store RF frame states, XE120x chip operating mode, etc... They may vary depending on the transceiver.

The variables declared as volatile can be used outside the *XE120xDriver.c*.

Some variables are initialized at declaration. Unless specified, the variables initialization values shouldn't be modified.

3.1 Variables common to all XE120xDivers

- ◆ `static _U8 RFState = RF_STOP; // RF state machine`
- ◆ `static _U8 *pRFFrame; // Pointer to the RF frame`
- ◆ `static _U8 RFFramePos; // RF frame current position`
- ◆ `static _U8 RFFrameSize; // RF frame size`
- ◆ `static _U16 ByteCounter = 0; // RF frame byte counter`
- ◆ `static _U8 PreMode = RF_SLEEP // Previous chip operating mode`
- ◆ `volatile _U8 EnableSyncByte = true; // Enables/disables the synchronization byte reception/transmission`
- ◆ `static _U8 SyncByte; // RF synchronization byte counter`
- ◆ `static _U8 PatternSize = 4; // Size of pattern detection`
- ◆ `static _U8 StartByte[4]; // Pattern detection values`
- ◆ `static _U16 RFFrameTimeOut = RF_FRAME_TIMEOUT(1200); // Reception counter value (full frame timeout generation)`

3.2 BitJockey™ specific variables

- `static _U8 RfifBaudrate = RFIF_BAUDRATE_1200; // BitJockey™ baudrate setting`
- `static _U8 RfifMode = RFIF_DISABLE; // BitJockey™ mode`

3.2.1 XE1201A and XE1209

- `static _U8 StartDetect = false; // Indicates when a start detection has been made`
- `static _U8 StartByteStatus = 0; // Indicates which start byte has been detected`
- `static _U8 StartByteCnt = 0; // Start byte counter`

3.3 BitBang specific variables (except SX1211, cf. below)

- `static _U16 RFBaudrate = TX_BAUDRATE_GEN_1200; // Transmission counter value (baudrate generation)`

3.4 SX1223 specificities

Contrary to the other XE1200, the SX1223 is a transmitter only and following points must be listed for proper use:

- Variable `RFFrameTimeOut` does not exist as well as all other reception related variables or functions.
- Variables `RfifBaudrate` and `RFBaudrate` must be manually set to the right value in coherence with what is programmed in `RegistersCfg`.
- Transmitted pattern should be defined accordingly by variables `PatternSize` and `StartByte`

3.5 SX1211 specificities

- ♦ In the SX1211, the equivalent notion of “Pattern”/“Start” is replaced by “Sync” or “Sync Word” while the previously called “Sync” notion (Synchronization bytes) does not exist.
- ♦ In continuous mode, the SX1211 provides a DCLK for the uC to send data synchronously, hence the `RFBaudrate` is not needed in BitBang implementation.
- ♦ In packet mode we do not refer to the “Frame” (ie whole packet including preamble and sync) but to the “Payload” since these two fields are automatically generated by the RF chip.

3.6 Copy of transceiver read/write registers

The `RegistersCfg` variable is an array which contains a copy of the internal registers of the XE1200 transceiver and is used at initialisation. This variable should be modified to set the transceiver parameters to those required for a particular application. Each array element corresponds to a transceiver register. Note that read only registers, are not reported and that some modifications (RF parameters aside) may affect the behavior of the Send and Receive functions. (Ex : disabling bit synchronizer, disabling pattern recognition, modifying IRQ mapping, etc...)

Example:

- ```
_U16 RegistersCfg[] = { // XE1201A configuration registers values
 DEF_REG_A | RF_A_CTRL_MODE_PIN | RF_A_CLOCK_ENABLE_ON | RF_A_CHIP_ENABLE_OFF
 | RF_A_TRANSMITTER_MODE | RF_A_BIT_SYNC_ON | RF_A_BAUDRATE_4800,
 DEF_REG_B,
 DEF_REG_C | RF_C_POWER_P_5 | RF_C_INVERT_OFF | RF_C_TRANS_AMP_ON
 | RF_C_TRANS_DATA_BIT_0 | RF_C_FDEV_125
};
```

Note that for SX1223 and SX1211, the fields ending with “\_VALUE” can be directly replaced by the decimal value desired.

## 4 HEADER FILES – CONTENTS

The API header files are divided in to 5 sections in the BitBang hardware interface and 6 sections in the BitJockey™ hardware interface.

### 4.1 Global definitions

This section defines the RF state machine states; functions return codes and RF frame definitions.

### 4.2 I/O ports definitions

The I/O ports used to interface between the transceiver and the microcontroller are defined to simplify code readability and portability.

The BitBang and BitJockey™ hardware interfaces introduce a difference between the corresponding headers files because the two interfaces don't use the same pins to communicate with the transceiver.

### 4.3 Transceiver 3 wire serial interface macros definitions

Macros are defined to simplify the API code implementation.

### 4.4 Transceiver operating modes and bit definitions

This section defines the transceiver register addresses and register contents. In addition, the different transceiver operating modes are defined.

### 4.5 Timing values definitions

This section defines pre-calculated values in order to generate the different timings such as transceiver specific timings, baud rates and timeouts.

### 4.6 BitJockey™ operating modes and bit definitions

This section defines each BitJockey™ register single bit, it also defines the different BitJockey™ operating modes.

#### Notes:

1. All the timing values in the *XE120xDriver.h* are calculated using a RC oscillator running at 2.4576 MHz.
2. The speed of the microcontrollers is not highly dependent upon supply voltage. However, by limiting the temperature range, the speed can be increased. For more information please refer to the microcontroller datasheet.

## 5 API USAGE TUTORIAL

This section details a simple application using the transceiver API functions based upon a simple “Ping-Pong” half-duplex communication protocol. MASTER/SLAVE architecture has been chosen to implement this application.

The application consists of a MASTER controller that transmits a RF frame containing a “PING” string and a SLAVE unit that receives the “PING” string and responds answers by transmitting a RF frame containing a “PONG” string.

To indicate that there is activity, the SLAVE unit will toggle a bit on its associated microcontroller Port B pin when the “PING” RF frame is received. In addition, the MASTER controller will also toggle a bit on the associated microcontroller Port B pin when the “PONG” RF frame is received

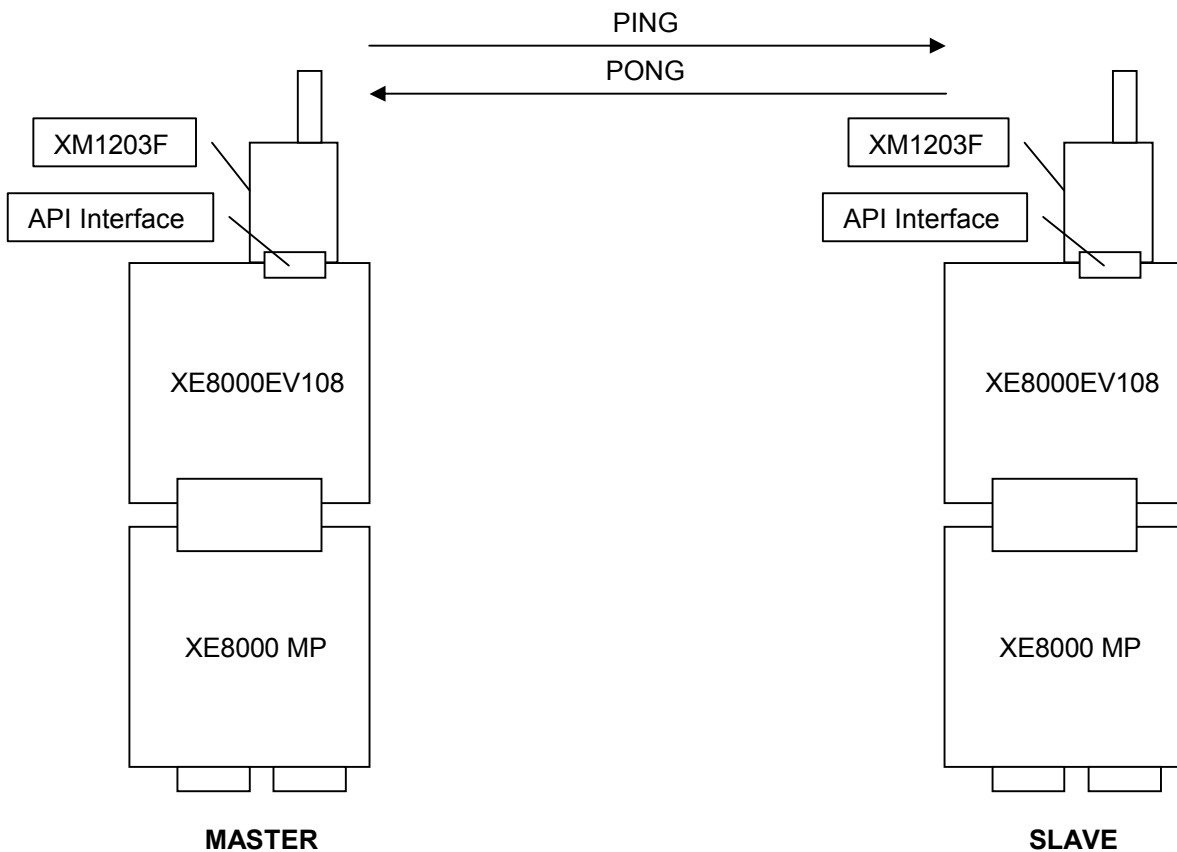
If the MASTER doesn’t receive a response from the SLAVE unit after a pre-defined time period, then the MASTER unit will re-transmit the “PING” RF frame.

In this tutorial we assume the following hardware configuration:

Transceiver: XE1203F

Microcontroller: XE8806A -> The BitJockey™ hardware interface is enabled

The microcontroller activity pin is the Port B pin 0 on both systems.



**Figure 1 Ping-Pong application principle**

Details of the implementation for the basic API hardware interface between the XE1200-series transceivers and XE8000-series microcontrollers can be found in appendix 7.2.

The basic hardware configuration consists of a Semtech ProStart II and an XM1203F transceiver module.

The ProStart II is composed with 1 XE8000MP (multi-purpose board) and 1 XE8000EV1XX (please refer to <http://www.semtech.com> for more information on microcontrollers evaluation tools)

Information about the different XM1200-series module family can be found on the Semtech website at <http://www.semtech.com>.

This combination of the ProStart II and the XM1200 module enables developers to quickly and easily configure their system.

### 5.1 Getting familiar with XE8000 software environment

As starting point for this tutorial please refer to the technical note TN8000.16 “coding with RIDE quick start” which can be found under. <http://www.semtech.com>

### 5.2 Setting up the project

- Download the source code corresponding to the API technical note from <http://www.semtech.com>.
- Uncompress the zip file to ex: “c:\”.
- Copy the *XE1203Driver.c* and *XE1203Driver.h* files that you will find in “c:\Api120x\1203\BitJockey\” directory to the RIDE project “src” directory.
- Add the *XE1203Driver.c* file to the RIDE project.

### Notes

1. If the BitBang hardware interface is to be implemented, use the XE120xDriver in the BitBang directory. If the BitJockey™ interface is to be implemented, use the XE120xDriver in the BitJockey™ directory.

2. The BitJockey™ Driver can only be used with the XE8806A or XE8807A microcontroller. The BitBang Driver can be used with all the Semtech microcontrollers.

In *Globals.h*, insert the following line in the section “Application specific Include files”:

```
#include "XE1203Driver.h"
```

The “Application specific Include files” section should then look like:

```
/*Application specific Include files*/
** Application specific Include files **
***/
#include "Initialisation.h"
#include "DFLLDriver.h"
#include "XE1203Driver.h"
```

### 5.3 Initializing the microcontroller

The API driver timings are based on CPU frequency of 2.4576MHz. This implies that the microcontroller RC oscillator frequency should be initialized with this value.

In *Initialisation.c* modify the *InitMicro* function by uncommenting the following line:

- `//InitRCoscillator(1228800); // Initialises the RC oscillator`

and change the parameter value to 2457600 the line should look like:

```
InitRCoscillator(2457600); // Initialises the RC oscillator
```

Comment out the following line:

```
InitUART(19200, 1, 0, 0, 2); // Initialises the UART communications
```

Afterwards the line should look like:

- `//InitUART(19200, 1, 0, 0, 2); // Initialises the UART communications`

#### 5.4 Initializing transceiver parameters

In *XE1203Driver.c* change `RegistersCfg` values according to the application requirements and to the XE1203F Data Sheet. The `RegistersCfg` variable is used by `InitRFChip` function to initialize the transceiver. When syntax is not obvious, please refer to the header file.

Example:

To change the XE1203F baud rate from 4800 to 38400 baud, whilst other register values remain at their default settings, the `RegistersCfg` variable should be modified as follows:

Original variable:

```
_U16 RegistersCfg[] = { // 1203 configuration registers values
 DEF_CONFIG | RF_CONFIG_MODE1,

 DEF_RTPARAM1 | RF_RT1_BIT_SYNC_ON | RF_RT1_BARKER_OFF | RF_RT1_RSSI_OFF
 | RF_RT1_RSSIR_LOW | RF_RT1_FEI_ON | RF_RT1_BW_600
 | RF_RT1_OSC_INT | RF_RT1_CLKOUT_OFF,
 .
 .
 .
 DEF_FSPARAM2 | RF_FS2_CHANGE_OSR_OFF | RF_FS2_BAUDRATE_4800,
 .
 .
 .
};
```

Modified variable:

```
_U16 RegistersCfg[] = { // 1203 configuration registers values
 DEF_CONFIG | RF_CONFIG_MODE1,

 DEF_RTPARAM1 | RF_RT1_BIT_SYNC_ON | RF_RT1_BARKER_OFF | RF_RT1_RSSI_OFF
 | RF_RT1_RSSIR_LOW | RF_RT1_FEI_ON | RF_RT1_BW_600
 | RF_RT1_OSC_INT | RF_RT1_CLKOUT_OFF,
 .
 .
 .
 DEF_FSPARAM2 | RF_FS2_CHANGE_OSR_OFF | RF_FS2_BAUDRATE_38400,
 .
 .
 .
};
```

#### 5.5 Declaring the application global variables

The application can now be implemented. Firstly, the application global variables need to be declared:

In *main.c* file start by declaring a variable that will contain the RF buffer. Then declare a variable that will contain the buffer size to be sent or the received buffer size, a variable that will indicate whether the application is a MASTER or SLAVE and a variable that indicates if a new packet needs to be sent by the MASTER controller.

In the *main.c* file “Global variables declaration” section add the following lines:

```
/*

** Global variables declaration

*/
_U8 RFbuffer[RF_BUFFER_SIZE]; // RF buffer
volatile _U8 RFbufferSize; // RF buffer size
volatile _U8 EnableMaster = true; // Master/Slave selection
volatile _U8 SendNextPacket = true; // Indicates when to send the next frame
```

The `RF_BUFFER_SIZE` is defined in the *XE1203Driver.h*. By default this value is set to 64 bytes. However, this buffer value can be changed according to the application and microcontroller RAM memory availability.

## 5.6 Implementing the main function

In the project main function (located in *main.c*) we need to call the API function `InitRFChip`

Once the transceiver is initialized the function enters an infinite loop where the application is tested to determine whether it is a MASTER or SLAVE, and then call the corresponding function that will implement the MASTER or SLAVE function.

The main function should be similar to that described below:

```
int main (void){
 InitMicro();

 _Monitor_Init();
 _Monitor_SoftBreak;

 InitRFChip();

 // Main Loop
 while(1){

 if(EnableMaster){
 OnMaster();
 }
 else{
 OnSlave();
 }
 }
 return 0;
}
```

## 5.7 Implementing the master function

This procedure implements the MASTER controller function. The main goal of this function is to transmit a "PING" and to test whether there is a "PONG" response. If a "PONG" response is received, then the procedure is restarted. In the case that there is no correct response after a pre-defined time-out period, the Master re-transmits a "PING".

The Master function should be similar to that described below:

```
void OnMaster(void){
 _U8 ReturnCode = -1;

 // Test if a PING needs to be sent
 if(SendNextPacket){

 // Copy the PING string to the buffer used to send the frame
 strcpy(RFbuffer, "PING");

 // Sends the frame to the RF chip
 SendRfFrame(RFbuffer, strlen(RFbuffer), &ReturnCode);

 // Indicates that we want to wait for an answer
 SendNextPacket = false;
 }
 else{
 // Receives the frame from the RF chip
 ReceiveRfFrame(RFbuffer, (_U8*)&RFbufferSize, &ReturnCode);

 // Tests if there is a TimeOut or if we have received a frame
 if(ReturnCode == OK){

 // Sets the last byte of received buffer to 0. Needed by strcmp
 // function
 RFbuffer[RFbufferSize] = '\0';

 // Tests if the received buffer size is greater than 0
 if(RFbufferSize > 0){

 // Tests if the received buffer value is PONG
 if(strcmp(RFbuffer, "PONG") == 0){

 // Indicates on a LED that the received frame is a PONG
 toggle_bit(RegPBOut, 0x01);
 }
 }

 // Indicates that we can send the next PING frame
 SendNextPacket = true;
 }
 else if(ReturnCode == RX_TIMEOUT){

 // Indicates that we can send the next PING frame
 SendNextPacket = true;
 }
 }
}
```

### 5.8 Implementing the slave function

This procedure implements the SLAVE unit function. The main goal of this function is to wait until a “PING” is received and then to transmit a “PONG” as a response.

An example of the Slave function is documented below:

```
void OnSlave(void){
 _U8 ReturnCode = -1;

 // Receives the frame from the RF chip
 ReceiveRfFrame(RFbuffer, (_U8*)&RFbufferSize, &ReturnCode);

 // Tests if we have received a frame
 if (ReturnCode == OK){

 // Tests if the received buffer size is greater than 0
 if (RFbufferSize > 0){

 // Sets the last byte of received buffer to 0. Needed by strcmp
 // function
 RFbuffer[RFbufferSize] = '\0';

 // Tests if the received buffer value is PING
 if(strcmp(RFbuffer, "PING") == 0){

 // Indicates on a LED that the received frame is a PING
 toggle_bit(RegPBOut, 0x01);

 // Copy the PONG string to the buffer used to send the frame
 strcpy(RFbuffer, "PONG");

 // Sends the frame to the RF chip
 SendRfFrame(RFbuffer, strlen(RFbuffer), &ReturnCode);
 }
 }
 }
}
```

### 5.9 Testing the application

The application is now finished.

The project can now be compiled and no errors should be generated.

To test the application:

1. Download the code to the MASTER controller system (refer to the “Getting started XE” document in the RIDE documentation).
2. Change the `EnableMaster` variable from `true` to `false` in the `main.c` file.
3. Compile the project.
4. Download the code to the SLAVE unit system.

If the application has been successfully implemented and the two systems are within RF communication range, the LED corresponding to the microcontroller Port B pin 0 should toggle on the two systems, verifying RF communications and subsequent “Ping-Pong” activity.

If the MASTER or SLAVE system is powered-down, then the LED on the powered-up system stops toggling.

## 6 API FUNCTIONS

The API functions are grouped in to four categories.

1. Configuration functions: Functions required for transceiver configuration.
2. Communication functions: Functions which permit the transmission and reception of RF data.
3. Transceiver specific functions: Functions specific to the different transceivers in the XE1200-series family and to their specific features.
4. Utility functions: Functions which perform miscellaneous operations, such as counters, events initialization, byte inversion, and delay generation, etc...

### 6.1 Configuration functions

#### 6.1.1 InitRFChip

##### Description

This function sends the values contained in `RegistersCfg` variable to the transceiver. It also initializes the `StartByte` and baud rates variables.

**Note:** The read-only registers are not included in the `RegistersCfg` variable.

##### Function table

|                  |     | BitBang                      | BitJockey™ | Buffered&Packet |
|------------------|-----|------------------------------|------------|-----------------|
| Function         |     | void InitRFChip (void)       |            |                 |
| Inputs           |     | -                            |            |                 |
| Outputs          |     | -                            |            |                 |
| Functions called |     | WriteRegister; InvertByte    |            |                 |
| Used Resources   |     | -                            |            |                 |
| Registers Used   | I/O | Dedicated configuration pins |            |                 |

\* Transceiver dependant defined in *XE120xDriver.h*

#### 6.1.2 SetRFMode

##### Description

This function defines the transceiver operating mode.

Please refer to the corresponding *XE120xDriver.h* file for details of the possible operating modes.

##### Function table

|                  |     | BitBang                               | BitJockey™                            | Buffered&Packet          |
|------------------|-----|---------------------------------------|---------------------------------------|--------------------------|
| Function         |     | Void SetRFMode( _U8 mode)             |                                       |                          |
| Inputs           |     | mode Defined in <i>XE120xDriver.h</i> |                                       |                          |
| Outputs          |     | -                                     |                                       |                          |
| Functions called |     | set_bit, clear_bit, Wait              | SetModeRFIF, set_bit, clear_bit, Wait | set_bit, clear_bit, Wait |
| Used Resources   |     | Counters A&B                          |                                       |                          |
| Registers Used   | I/O | Dedicated configuration pins          |                                       |                          |

\* Transceiver dependant defined in *XE120xDriver.h*

### 6.1.3 SetModeRFIF

#### Description

This function initializes all the BitJockey™ registers in order to set a predefined mode (see the XE8806A and XE8807A datasheet for the available modes).

#### Function table

|                  |     | BitBang | BitJockey™                                         | Buffered&Packet |
|------------------|-----|---------|----------------------------------------------------|-----------------|
| Function         |     | N.A.    | void SetModeRFIF ( _U8 mode)                       | N.A.            |
| Inputs           |     | N.A.    | mode Defined in <i>XE120xDriver.h</i>              | N.A.            |
| Outputs          |     | N.A.    | -                                                  | N.A.            |
| Functions called |     | N.A.    | -                                                  | N.A.            |
| Used Resources   |     | N.A.    | -                                                  | N.A.            |
| Registers Used   | Sys | N.A.    | RegRfifCmd1, RegRfifCmd2, RegRfifCmd3, RegIrqEnHig | N.A.            |

### 6.1.4 WriteRegister

#### Description

This function writes to the XE120x registers via dedicated programming pins, following the protocol given in the transceiver datasheet.

**Note:** The clocking of the operation is based on the CPU frequency of the user program.

#### Function table

|                  |     | BitBang                                                                                            | BitJockey™           | Buffered&Packet |
|------------------|-----|----------------------------------------------------------------------------------------------------|----------------------|-----------------|
| Function         |     | void WriteRegister( _U8 address, _U16 value)                                                       |                      |                 |
| Inputs           |     | addressAddress of the register defined in the transceiver datasheet<br>value Value of the register |                      |                 |
| Outputs          |     | -                                                                                                  |                      |                 |
| Functions called |     | Serial macros, set_bit, clear_bit                                                                  | SPI macros, SpiInOut |                 |
| Used Resources   |     | -                                                                                                  |                      |                 |
| Registers Used   | I/O | Dedicated configuration pins*                                                                      |                      |                 |

\* Transceiver dependant defined in *XE120xDriver.h*

### 6.1.5 ReadRegister

#### Description

This function reads from registers via dedicated programming pins, following the protocol given in the transceiver datasheet.

**Note:** The clocking of the operation is based on the CPU frequency of the user program.

#### Function table

|                  |     | BitBang                                                             | BitJockey™           | Buffered&Packet |
|------------------|-----|---------------------------------------------------------------------|----------------------|-----------------|
| Function         |     | _U16 value = ReadRegister ( _U8 address)                            |                      |                 |
| Inputs           |     | addressAddress of the register defined in the transceiver datasheet |                      |                 |
| Outputs          |     | value Value of the register                                         |                      |                 |
| Functions called |     | Serial macros, set_bit, clear_bit                                   | SPI macros, SpiInOut |                 |
| Used Resources   |     | -                                                                   |                      |                 |
| Registers Used   | I/O | Dedicated configuration pins*                                       |                      |                 |

\* Transceiver dependant defined in *XE120xDriver.h*

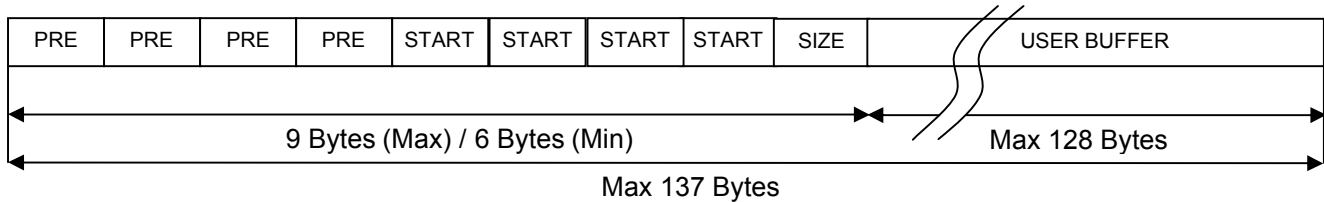
## 6.2 Communication functions

### 6.2.1 SendRfFrame

#### Description

This function encapsulates given User Buffer in an RF frame and sends it to the transceiver.

The RF frame format is described below:



Depending upon the specific user application the number of START bytes can be varied between 1 up to 4.

Example 1: XE1201A (No hardware pattern detection)

In XE1201ADriver.c Global variable section modify the `PatternSize` variable.

Original variable:

```
static _U8 PatternSize = 4; // Size of pattern detection
```

Modified variable:

```
static _U8 PatternSize = 2; // Size of pattern detection
```

Example 2: XE1202 (Hardware pattern detection)

In the `RegistersCfg` variable modify the register where the Pattern size is defined as indicated in the transceiver datasheet.

Original variable:

```
_U16 RegistersCfg[] = { // 1203 configuration registers values
 DEF_CONFIG | RF_CONFIG_MODE1,
 .
 .
 .
 DEF_ADPARAM1 | RF_AD1_PATTERN_ON | RF_AD1_P_SIZE_32 | RF_AD1_P_TOL_0
 | RF_AD1_CLK_FREQ_1_22_MHZ | RF_AD1_IQA_OFF,
 .
 .
 .
};
```

Modified variable:

```
_U16 RegistersCfg[] = { // 1203 configuration registers values
 DEF_CONFIG | RF_CONFIG_MODE1,
 .
 .
 .
 DEF_ADPARAM1 | RF_AD1_PATTERN_ON | RF_AD1_P_SIZE_16 | RF_AD1_P_TOL_0
 | RF_AD1_CLK_FREQ_1_22_MHZ | RF_AD1_IQA_OFF,
 .
 .
 .
};
```

**Example 3: XE1203F (Hardware pattern detection)**

In the `RegistersCfg` variable modify the register where the Pattern size is defined as indicated in the transceiver datasheet.

Original variable:

```
_U16 RegistersCfg[] = { // 1203 configuration registers values
 DEF_CONFIG | RF_CONFIG_MODE1,
 .
 .
 .
 DEF_ADPARAM1 | RF_AD1_P_SIZE_32 | RF_AD1_P_TOL_0 | RF_AD1_CLK_FREQ_1_22_MHZ
 | RF_AD1_INVERT_OFF | RF_AD1_REG_BW_OFF,
 .
 .
 .
};
```

Modified variable:

```
_U16 RegistersCfg[] = { // 1203 configuration registers values
 DEF_CONFIG | RF_CONFIG_MODE1,
 .
 .
 .
 DEF_ADPARAM1 | RF_AD1_P_SIZE_16 | RF_AD1_P_TOL_0 | RF_AD1_CLK_FREQ_1_22_MHZ
 | RF_AD1_INVERT_OFF | RF_AD1_REG_BW_OFF,
 .
 .
 .
};
```

Return values:

1. Status: OK                      Frame has been sent
2. Status: ERROR                 The RF frame buffer size is not correct
3. Status: TX\_TIMEOUT          Indicates that the function exits after waiting with no success to send a full RF frame
4. Status: TX\_RUNNING          Indicates that the transmission is not yet finished (BitJockey™ only)

**Function table**

|                  |     | BitBang                                                                                                                | BitJockey™                                  | Buffered&Packet                                         |
|------------------|-----|------------------------------------------------------------------------------------------------------------------------|---------------------------------------------|---------------------------------------------------------|
| Function         |     | void SendRfFrame ( _U8 *buffer, _U8 size, _U8 *pReturnCode)                                                            |                                             |                                                         |
| Inputs           |     | *buffer                      Address of buffer to send<br>size                              Size of the buffer to send |                                             |                                                         |
| Outputs          |     | *pReturnCode    Function status                                                                                        |                                             |                                                         |
| Functions called |     | TxEvntsOn, TxEventsOff, SetRFMode, SendByte                                                                            | SetRFMode                                   | WriteRegister, SendByte, SetRFMode                      |
| Used Resources   |     | Counters A & B                                                                                                         | -                                           | -                                                       |
| Registers Used   | Sys | RegEvn, RegEvnEn                                                                                                       | RegRfifTx, RegRfifTxSta                     | -                                                       |
|                  | I/O | Clock from transceiver -> PA1<br>Dedicated data pins *<br>Dedicated configuration pins *                               | PD0 – PD3<br>Dedicated configuration pins * | Dedicated data pins *<br>Dedicated configuration pins * |
|                  | Cnt | RegCntA, RegCntB, RegCntCtrlCk, RegCntConfig1, RegCntOn                                                                | -                                           |                                                         |

\* Transceiver dependant defined in `XE120xDriver.h`

### 6.2.2 ReceiveRfFrame

#### Description

This function receives an RF frame from the transceiver and returns the User Buffer and the User Buffer size.

The RF frame format is the same as the SendRfFrame.

Return values:

1. Status: OK                      RF frame reception done
2. Status: ERROR                The received RF frame buffer size is not correct
3. Status: RX\_TIMEOUT        Indicates that the function exits after waiting with no success to receive a full RF frame
4. Status: RX\_RUNNING        Indicates that the reception is not yet finished (BitJockey™ only)

**Note:** When using the BitJockey™ handling of a RF frame is done using the interrupts of the peripheral.

#### Function table

|                  |     | BitBang                                                                                                       | BitJockey™                                  | Buffered&Packet                                         |
|------------------|-----|---------------------------------------------------------------------------------------------------------------|---------------------------------------------|---------------------------------------------------------|
| Function         |     | void ReceiveRfFrame ( _U8 *buffer, _U8 *size, _U8 *pReturnCode)                                               |                                             |                                                         |
| Inputs           |     | *buffer                      Address of received buffer<br>*size                         Received buffer size |                                             |                                                         |
| Outputs          |     | *pReturnCode    Function status                                                                               |                                             |                                                         |
| Functions called |     | RxEventsOn, RxEventsOff, SetRFMode, ReceiveByte, set_bit, clear_bit                                           | SetRFMode, EnableTimeOut                    | WriteRegister, SetRFMode, EnableTimeOut                 |
| Used Resources   |     | Counters A & B                                                                                                | -                                           |                                                         |
| Registers Used   | Sys | RegEvn, RegEvnEn                                                                                              | -                                           |                                                         |
|                  | I/O | Clock from transceiver -> PA1<br>Dedicated data pins *<br>Dedicated configuration pins *                      | PD0 – PD3<br>Dedicated configuration pins * | Dedicated data pins *<br>Dedicated configuration pins * |
|                  | Cnt | RegCntA, RegCntB, RegCntCtrlCk, RegCntConfig1, RegCntOn                                                       | -                                           |                                                         |

\* Transceiver dependant defined in *XE120xDriver.h*

### 6.2.3 SendByte

#### Description

This function sends a byte through the dedicated transceiver communication pins. The function sends the byte LSB first.

Except for SX1211 which provides a compulsory Tx clock, the baud rate is generated by the A and B counters and their associated event. See TxEventsOn & TxEventsOff functions for a full description of these tasks.

All the timings are referenced to the RC clock-frequency of the microcontroller. The baud rate is automatically set when the InitRFChip function is called.

#### Function table

|                  |     | <b>BitBang</b>                                          | <b>BitJockey™</b> | <b>Buffered&amp;Packet</b> |
|------------------|-----|---------------------------------------------------------|-------------------|----------------------------|
| Function         |     | void SendByte ( _U8 b)                                  | N.A               | void SendByte ( _U8 b)     |
| Inputs           |     | b Byte to be send                                       | N.A               | b Byte to be send          |
| Outputs          |     | -                                                       | N.A               | -                          |
| Functions called |     | set_bit, clear_bit                                      | N.A               | SPI macros                 |
| Used Resources   |     | Counters A & B                                          | N.A               | -                          |
| Registers Used   | Sys | RegEvn, RegEvnEn                                        | N.A               | -                          |
|                  | I/O | Dedicated data pins *                                   | N.A               | Dedicated data pins *      |
|                  | Cnt | RegCntA, RegCntB, RegCntCtrlCk, RegCntConfig1, RegCntOn | N.A               | -                          |

\* Transceiver dependant defined in *XE120xDriver.h*

### 6.2.4 ReceiveByte

#### Description

This function reads a byte on the dedicated transceiver communication pins. The function receives the byte LSB first.

RF frame timeout is generated when the A and B counters generate their event. Clock synchronization is achieved using the event occurring on the falling edge of PA1 pin. See RxEventsOn & RxEventsOff functions.

All the timings are referenced to the RC clock-frequency of the microcontroller. The baud rate is automatically set when the InitRFChip function is called.

#### Function table

|                  |     | <b>BitBang</b>                                          | <b>BitJockey™</b> | <b>Buffered&amp;Packet</b> |
|------------------|-----|---------------------------------------------------------|-------------------|----------------------------|
| Function         |     | _U8 byte = ReceiveByte ( )                              | N.A               | _U8 byte = ReceiveByte ( ) |
| Inputs           |     | -                                                       | N.A               | -                          |
| Outputs          |     | byte Data received                                      | N.A               | byte Data received         |
| Functions called |     | set_bit, clear_bit                                      | N.A               | SPI macros                 |
| Used Resources   |     | Counters A & B                                          | N.A               | -                          |
| Registers Used   | Sys | RegEvn, RegEvnEn                                        | N.A               | -                          |
|                  | I/O | Clock from transceiver -> PA1<br>Dedicated data pins *  | N.A               | Dedicated data pins *      |
|                  | Cnt | RegCntA, RegCntB, RegCntCtrlCk, RegCntConfig1, RegCntOn | N.A               | -                          |

\* Transceiver dependant defined in *XE120xDriver.h*

### 6.3 Transceiver specific functions

#### 6.3.1 AutoFreqControl

##### Description

This function enables an automatic frequency correction to align the transceiver LO frequency with the received transmitted carrier.

Depending on the transceiver, this can be achieved by either hardware or software.

##### Function table

|                  |     | BitBang                                                                            | BitJockey™ | Buffered&Packet |
|------------------|-----|------------------------------------------------------------------------------------|------------|-----------------|
| Function         |     | void AutoFreqControl ( _U8 *pReturnCode)                                           |            |                 |
| Inputs           |     | -                                                                                  |            |                 |
| Outputs          |     | *pReturnCode Function status (see section 4 for definitions)                       |            |                 |
| Functions called |     | WriteRegister, ReadRegister, ReadLO, WriteLO, InitFei, ReadFei, set_bit, clear_bit |            |                 |
| Used Resources   |     | Counters A & B                                                                     |            |                 |
| Registers Used   | Sys | RegEvnEn, RegEvn                                                                   |            |                 |
|                  | I/O | Dedicated configuration pins*                                                      |            |                 |
|                  | Cnt | RegCntOn, RegCntA, RegCntB                                                         |            |                 |

\* Transceiver dependant defined in *XE120xDriver.h*

#### 6.3.2 ReadLO

##### Description

Reads the LO frequency value from the transceiver.

##### Function table

|                  |     | BitBang                       | BitJockey™ | Buffered&Packet |
|------------------|-----|-------------------------------|------------|-----------------|
| Function         |     | _U16 data = ReadLO ( )        |            |                 |
| Inputs           |     | -                             |            |                 |
| Outputs          |     | data LO register value        |            |                 |
| Functions called |     | ReadRegister                  |            |                 |
| Used Resources   |     | -                             |            |                 |
| Registers Used   | I/O | Dedicated configuration pins* |            |                 |

\* Transceiver dependant defined in *XE120xDriver.h*

#### 6.3.3 WriteLO

##### Description

Writes the LO frequency value to the transceiver.

##### Function table

|                  |     | BitBang                                 | BitJockey™ | Buffered&Packet |
|------------------|-----|-----------------------------------------|------------|-----------------|
| Function         |     | void WriteLO ( _U8 value)               |            |                 |
| Inputs           |     | value Value to write in the LO register |            |                 |
| Outputs          |     | -                                       |            |                 |
| Functions called |     | WriteRegister                           |            |                 |
| Used Resources   |     | -                                       |            |                 |
| Registers Used   | I/O | Dedicated configuration pins *          |            |                 |

\* Transceiver dependant defined in *XE120xDriver.h*

### 6.3.4 InitFei

#### Description

This function initializes the XE120X to enable the FEI reading.

#### Function table

|                  |     | BitBang                                | BitJockey™ | Buffered&Packet |
|------------------|-----|----------------------------------------|------------|-----------------|
| Function         |     | void InitFei (void)                    |            |                 |
| Inputs           |     | -                                      |            |                 |
| Outputs          |     | -                                      |            |                 |
| Functions called |     | ReadRegister, set_bit, clear_bit, Wait |            |                 |
| Used Resources   |     | Counters A & B                         |            |                 |
| Registers Used   | Sys | RegEvnEn, RegEvn                       |            |                 |
|                  | I/O | Dedicated configuration pins *         |            |                 |
|                  | Cnt | RegCntOn, RegCntA, RegCntB             |            |                 |

\* Transceiver dependant defined in *XE120xDriver.h*

### 6.3.5 ReadFei

#### Description

This function reads the FeiOut register(s) and returns the value as a 16-bit value.

#### Function table

|                  |     | BitBang                                | BitJockey™ | Buffered&Packet |
|------------------|-----|----------------------------------------|------------|-----------------|
| Function         |     | _S16 value = ReadFei (void)            |            |                 |
| Inputs           |     | -                                      |            |                 |
| Outputs          |     | value Value of the FEI                 |            |                 |
| Functions called |     | ReadRegister, set_bit, clear_bit, Wait |            |                 |
| Used Resources   |     | Counters A & B                         |            |                 |
| Registers Used   | Sys | RegEvnEn, RegEvn                       |            |                 |
|                  | I/O | Dedicated configuration pins*          |            |                 |
|                  | Cnt | RegCntOn, RegCntA, RegCntB             |            |                 |

\* Transceiver dependant defined in *XE120xDriver.h*

### 6.3.6 InitRssi

#### Description

This function initializes the XE120X to enable the RSSI reading.

#### Function table

|                  |     | BitBang                       | BitJockey™ | Buffered&Packet |
|------------------|-----|-------------------------------|------------|-----------------|
| Function         |     | void InitRssi (void)          |            |                 |
| Inputs           |     | -                             |            |                 |
| Outputs          |     | -                             |            |                 |
| Functions called |     | set_bit, clear_bit, Wait      |            |                 |
| Used Resources   |     | Counters A & B                |            |                 |
| Registers Used   | Sys | RegEvnEn, RegEvn              |            |                 |
|                  | I/O | Dedicated configuration pins* |            |                 |
|                  | Cnt | RegCntOn, RegCntA, RegCntB    |            |                 |

\* Transceiver dependant defined in *XE120xDriver.h*

### 6.3.7 ReadRssi

#### Description

This function reads the Rssi value from XE120x and formats it to a 16-bit value.

#### Function table

|                  |     | BitBang                          | BitJockey™ | Buffered&Packet |
|------------------|-----|----------------------------------|------------|-----------------|
| Function         |     | _U16 value = ReadRssi (void)     |            |                 |
| Inputs           |     | -                                |            |                 |
| Outputs          |     | value Value of the RSSI register |            |                 |
| Functions called |     | ReadRegister , toggle_bit, Wait  |            |                 |
| Used Resources   |     | Counters A & B                   |            |                 |
| Registers Used   | Sys | RegEvnEn, RegEvn                 |            |                 |
|                  | I/O | Dedicated configuration pins     |            |                 |
|                  | Cnt | RegCntOn, RegCntA, RegCntB       |            |                 |

\* Transceiver dependant defined in *XE120xDriver.h*

## 6.4 Utility functions

### 6.4.1 Wait

#### Description

Loads the counter A & B and waits a pre-defined time period.

All the timings are referenced to the RC-clock frequency. The method used to set the correct frequency is defined in the *XE120xDriver.h* file.

#### Function table

|                  |     | BitBang                                                 | BitJockey™ | Buffered&Packet |
|------------------|-----|---------------------------------------------------------|------------|-----------------|
| Function         |     | void Wait ( _U8 cntVal)                                 |            |                 |
| Inputs           |     | cntVal Value of the counter                             |            |                 |
| Outputs          |     | -                                                       |            |                 |
| Functions called |     | -                                                       |            |                 |
| Used Resources   |     | Counters A & B                                          |            |                 |
| Registers Used   | Sys | RegEvn, RegEvnEn                                        |            |                 |
|                  | I/O | -                                                       |            |                 |
|                  | Cnt | RegCntA, RegCntB, RegCntCtrlCk, RegCntConfig1, RegCntOn |            |                 |

### 6.4.2 EnableTimeOut

#### Description

This function enables or disables the RF frame time-out generation.

#### Function table

|                  |     | BitBang | BitJockey™                                              | Buffered&Packet |
|------------------|-----|---------|---------------------------------------------------------|-----------------|
| Function         |     | N.A.    | void EnableTimeOut( _U8 enable)                         |                 |
| Inputs           |     | N.A.    | enable 1 -> enable / 0 -> disables                      |                 |
| Outputs          |     | N.A.    | -                                                       |                 |
| Functions called |     | N.A.    | -                                                       |                 |
| Used Resources   |     | N.A.    | Counters A & B                                          |                 |
| Registers Used   | Sys | N.A.    | RegIrqEnHig                                             |                 |
|                  | I/O | N.A.    | -                                                       |                 |
|                  | Cnt | N.A.    | RegCntA, RegCntB, RegCntCtrlCk, RegCntConfig1, RegCntOn |                 |

### 6.4.3 TxEventsOn

#### Description

This function initializes the microcontroller counters and events that enable the data transmission.

**Note:** This function disables all interrupts.

#### Function table

|                  |     | BitBang                                                 | BitJockey™ | Buffered&Packet |
|------------------|-----|---------------------------------------------------------|------------|-----------------|
| Function         |     | void TxEventsOn (void)                                  | N.A.       |                 |
| Inputs           |     | -                                                       | N.A.       |                 |
| Outputs          |     | -                                                       | N.A.       |                 |
| Functions called |     | -                                                       | N.A.       |                 |
| Used Resources   |     | Counters A & B                                          | N.A.       |                 |
| Registers Used   | Sys | RegEvn, RegEvnEn                                        | N.A.       |                 |
|                  | I/O | -                                                       | N.A.       |                 |
|                  | Cnt | RegCntA, RegCntB, RegCntCtrlCk, RegCntConfig1, RegCntOn | N.A.       |                 |

### 6.4.4 TxEventsOff

#### Description

Disables all the counters and events used during the data transmission.

#### Function table

|                  |     | BitBang                                                 | BitJockey™ | Buffered&Packet |
|------------------|-----|---------------------------------------------------------|------------|-----------------|
| Function         |     | void TxEventsOff (void)                                 | N.A.       |                 |
| Inputs           |     | -                                                       | N.A.       |                 |
| Outputs          |     | -                                                       | N.A.       |                 |
| Functions called |     | -                                                       | N.A.       |                 |
| Used Resources   |     | Counters A & B                                          | N.A.       |                 |
| Registers Used   | Sys | RegEvn, RegEvnEn                                        | N.A.       |                 |
|                  | I/O | -                                                       | N.A.       |                 |
|                  | Cnt | RegCntA, RegCntB, RegCntCtrlCk, RegCntConfig1, RegCntOn | N.A.       |                 |

### 6.4.5 RxEventsOn

#### Description

This function initializes the microcontroller counters and events that enable the data reception.

**Note:** This function disables all interrupts.

#### Function table

|                  |     | BitBang                                                 | BitJockey™ | Buffered&Packet |
|------------------|-----|---------------------------------------------------------|------------|-----------------|
| Function         |     | void RxEventsOn (void)                                  | N.A.       |                 |
| Inputs           |     | -                                                       | N.A.       |                 |
| Outputs          |     | -                                                       | N.A.       |                 |
| Functions called |     | -                                                       | N.A.       |                 |
| Used Resources   |     | Counters A & B                                          | N.A.       |                 |
| Registers Used   | Sys | RegEvn, RegEvnEn                                        | N.A.       |                 |
|                  | I/O | -                                                       | N.A.       |                 |
|                  | Cnt | RegCntA, RegCntB, RegCntCtrlCk, RegCntConfig1, RegCntOn | N.A.       |                 |



## 7 APPENDIX

### 7.1 API functions availability

This section gives the availability of the functions for each transceiver.

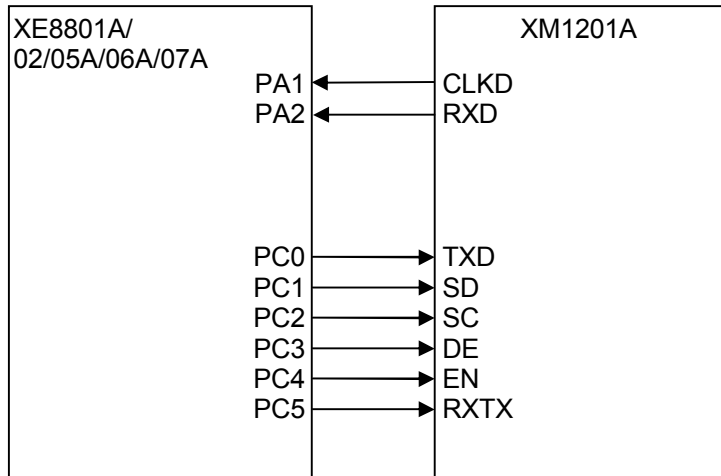
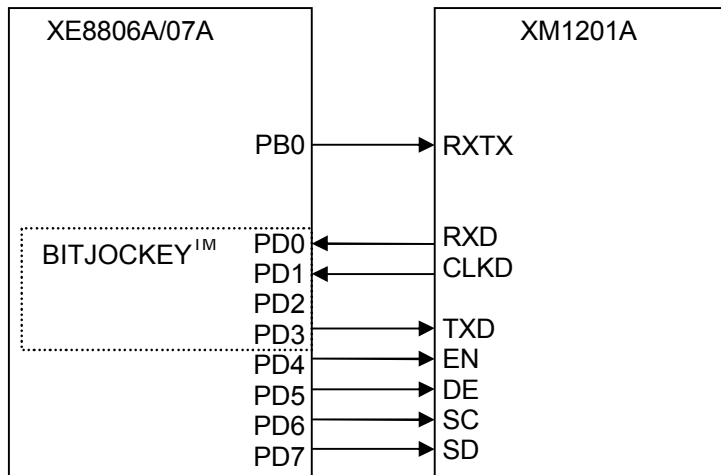
#### 7.1.1 XE8801A-02-05A-06A-07A functions

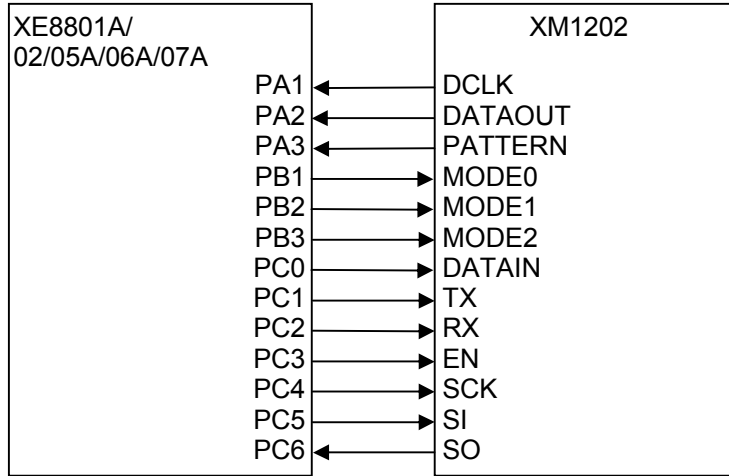
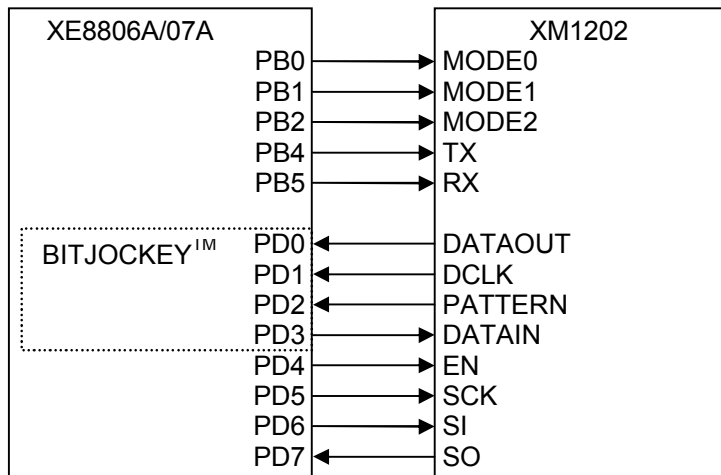
| Transceiver   |                 | Function name |             |        |        |        |        |                                                            |
|---------------|-----------------|---------------|-------------|--------|--------|--------|--------|------------------------------------------------------------|
|               |                 | XE1201A       | XE1203F/02A | XE1205 | XE1209 | SX1223 | SX1211 |                                                            |
| Configuration | InitRFChip      | ●/■           | ●/■         | ●/■/◇  | ●/■    | ●/■    | ●/◇    | Initializes the transceiver with registers default values. |
|               | SetRFMode       | ●/■           | ●/■         | ●/■/◇  | ●/■    | ●/■    | ●/◇    | Sets a predefined mode for the transceiver                 |
|               | SetModeRFIF     | -/■           | -/■         | -/■/-  | -/■    | -/■    | -/     | Sets the BitJockey™ to one of the predefined modes.        |
|               | WriteRegister   | ●/■           | ●/■         | ●/■/◇  | ●/■    | ●/■    | ●/◇    | Writes a value in the transceiver internal register        |
|               | ReadRegister    | ●/■           | ●/■         | ●/■/◇  | ●/■    | ●/■    | ●/◇    | Reads a value in the transceiver internal register         |
| Comm          | SendRfFrame     | ●/■           | ●/■         | ●/■/◇  | ●/■    | ●/■    | ●/◇    | Sends a frame through RF link                              |
|               | ReceiveRfFrame  | ●/■           | ●/■         | ●/■/◇  | ●/■    | -/-    | ●/◇    | Receives a frame through RF link                           |
|               | SendByte        | ●/-           | ●/-         | ●/-/◇  | ●/-    | ●/-    | ●/◇    | Sends a byte through the dedicated comm. pins              |
|               | ReceiveByte     | ●/-           | ●/-         | ●/-/◇  | ●/-    | -/-    | ●/◇    | Receives a byte through the dedicated comm. pins           |
| Transceiver   | AutoFreqControl | ○/□           | ●/■         | ●/■/◇  | -/-    | -/-    | -/-    | Does an automatic frequency correction                     |
|               | ReadLO          | -/-           | ●/■         | ●/■/◇  | -/-    | -/-    | -/-    | Reads the local oscillator frequency registers             |
|               | WriteLO         | -/-           | ●/■         | ●/■/◇  | -/-    | -/-    | -/-    | Write the local oscillator frequency registers             |
|               | InitFei         | -/-           | ●/■         | ●/■/◇  | -/-    | -/-    | -/-    | Initialize the transceiver for FEI operations              |
|               | ReadFei         | -/-           | ●/■         | ●/■/◇  | -/-    | -/-    | -/-    | Reads the FEI out register                                 |
|               | InitRssi        | -/-           | ●/■         | ●/■/◇  | -/-    | -/-    | -/-    | Initializes the transceiver for RSSI operations            |
|               | ReadRssi        | -/-           | ●/■         | ●/■/◇  | -/-    | -/-    | ●/◇    | Reads the RSSI out register                                |
| Utility       | Wait            | ●/■           | ●/■         | ●/■/◇  | ●/■    | ●/■    | ●/◇    | Waits a given time                                         |
|               | EnableTimeOut   | -/■           | -/■         | -/■/◇  | -/■    | -/-    | -/◇    | Enables/Disables RF frame time out                         |
|               | TxEventsOn      | ●/-           | ●/-         | ●/-/-  | ●/-    | ●/-    | ●/-    | Initializes counter and events for transmission.           |
|               | TxEventsOff     | ●/-           | ●/-         | ●/-/-  | ●/-    | ●/-    | ●/-    | Disables all the resources used during transmission.       |
|               | RxEventsOn      | ●/-           | ●/-         | ●/-/-  | ●/-    | -/-    | ●/-    | Initializes counter and events for reception.              |
|               | RxEventsOff     | ●/-           | ●/-         | ●/-/-  | ●/-    | -/-    | ●/-    | Disables all the resources used during reception.          |
|               | InvertByte      | -/-           | ●/■         | ●/■/◇  | -/-    | ●/■    | ●/◇    | Inverts a byte. MSB -> LSB, LSB -> MSB                     |

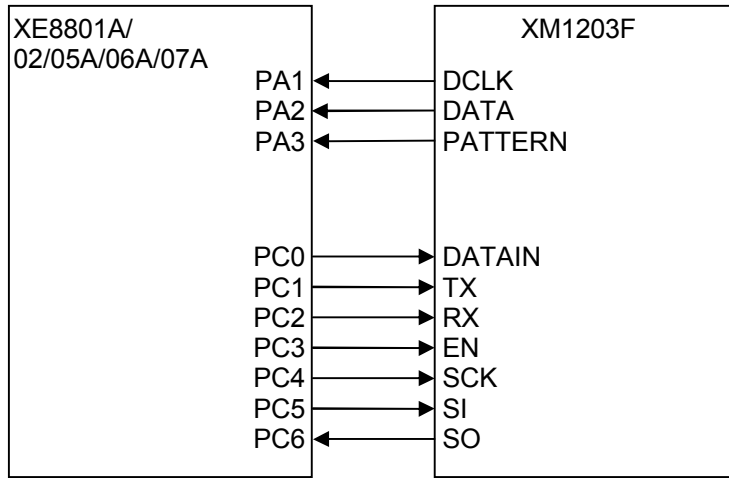
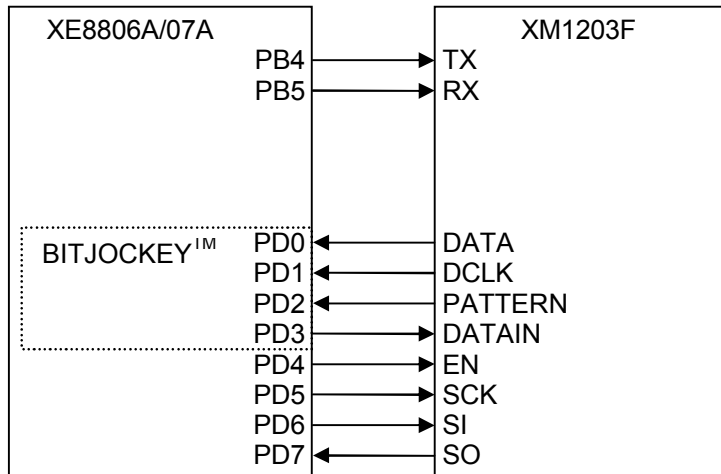
Table 2 Functions availability

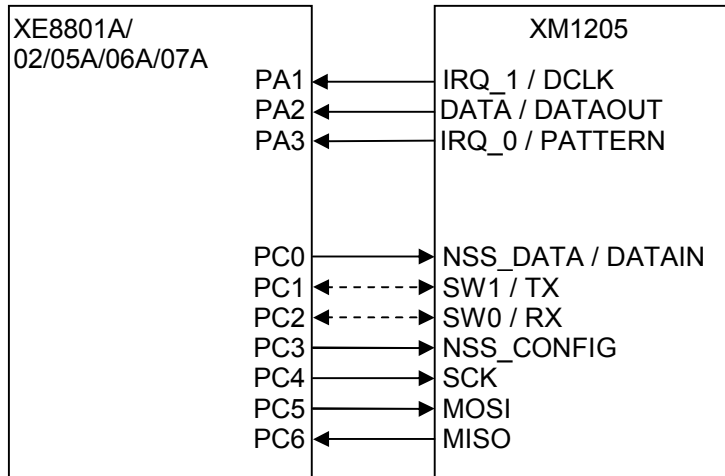
#### Legend used in the availability table

| Symbol | Meaning                                                                                         |
|--------|-------------------------------------------------------------------------------------------------|
| ○      | Not available yet but possible to implement for the given transceiver with BitBang interface    |
| ●      | Available with BitBang interface                                                                |
| □      | Not available yet but possible to implement for the given transceiver with BitJockey™ interface |
| ■      | Available with BitJockey™ interface                                                             |
| ◇      | Not available yet but possible to implement for the given transceiver with Buffered interface   |
| ◆      | Available with Buffered interface (and Packet for SX1211)                                       |
| -      | Unavailable/not-required                                                                        |

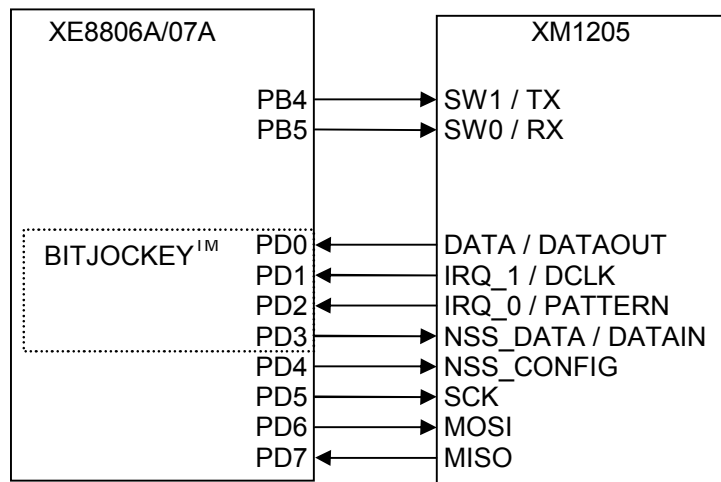
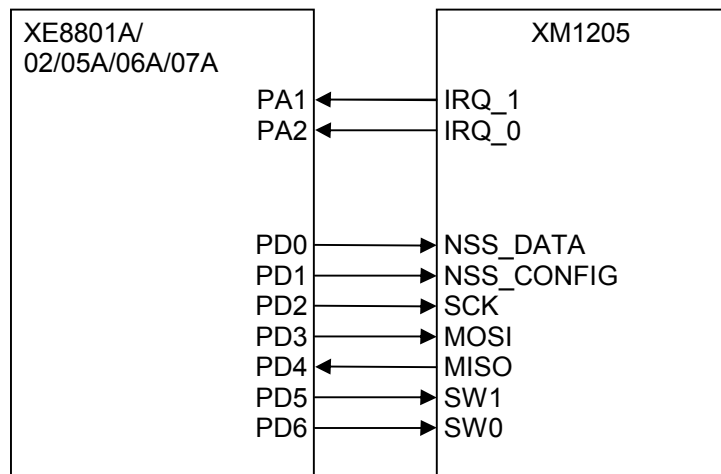
**7.2 Hardware connections implemented for described drivers.**
**7.2.1 Connections between XM1201A and XE8801A/02/05A/06A/07A BitBang**

**7.2.2 Connections between XM1201A and XE8806A/07A BitJockey™**


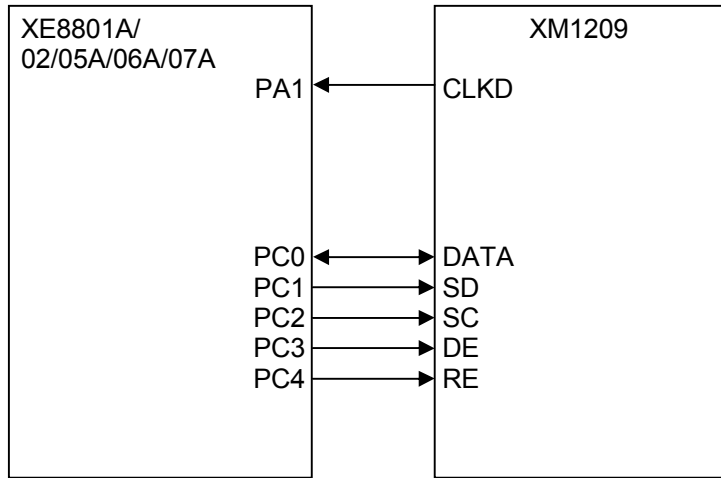
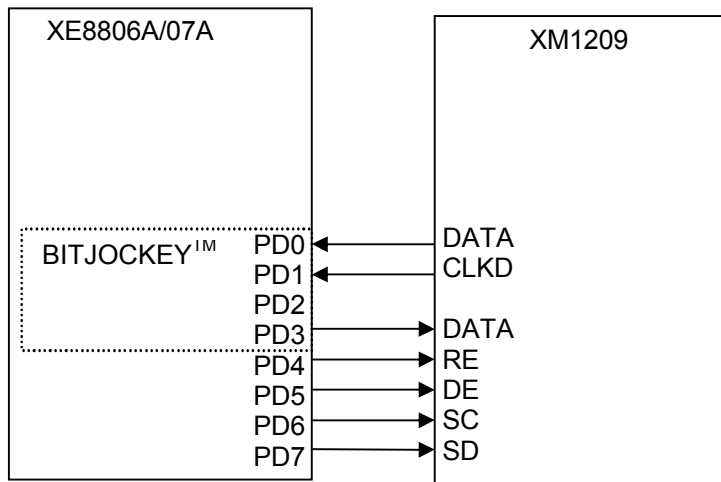
**7.2.3 Connections between XM1202 and XE8801A/02/05A/06A/07A BitBang**

**7.2.4 Connections between XM1202 and XE8806A/07A BitJockey™**


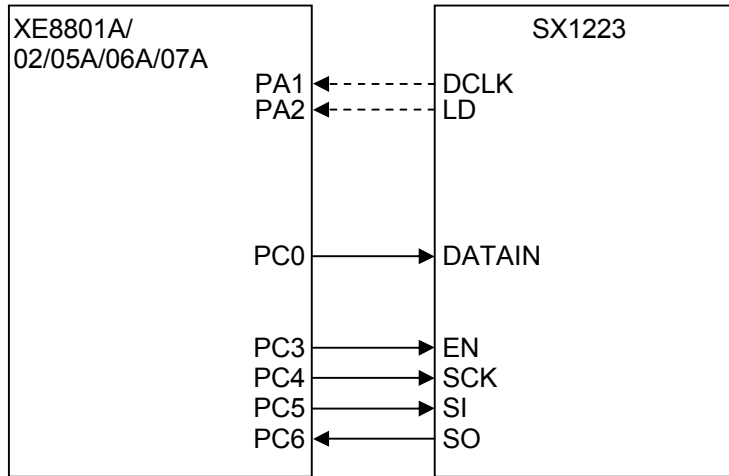
**7.2.5 Connections between XM1203F and XE8801A/02/05A/06A/07A BitBang**

**7.2.6 Connections between XM1203F and XE8806A/07A BitJockey™ (XE1283 compatible)**


**7.2.7 Connections between XM1205 and XE8801A/02/05A/06A/07A BitBang**


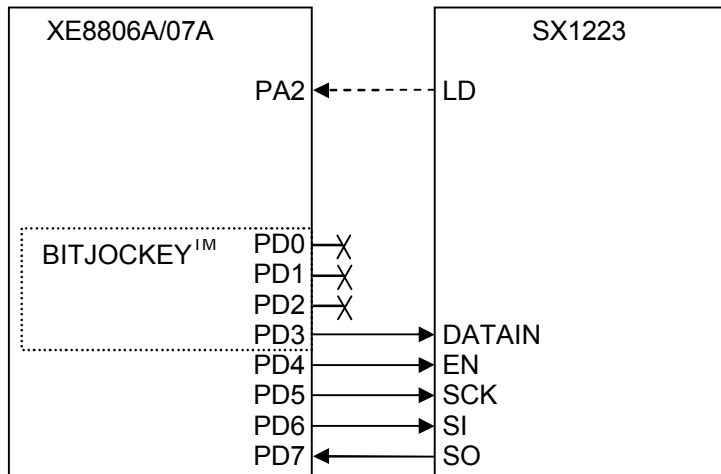
Note for all XM1205 connections: SW0 and SW1 pins are not used in the current implementations. If connected, the respective IO they are tied to must be configured accordingly.

**7.2.8 Connections between XM1205 and XE8806A/07A BitJockey™**

**7.2.9 Connections between XM1205 and XE8801A/02/05A/06A/07A Buffered**


**7.2.10 Connections between XM1209 and XE8801A/02/05A/06A/07A BitBang**

**7.2.11 Connections between XM1209 and XE8806A/07A BitJockey™**


**7.2.12 Connections between SX1223 and XE8801A/02/05A/06A/07A BitBang**


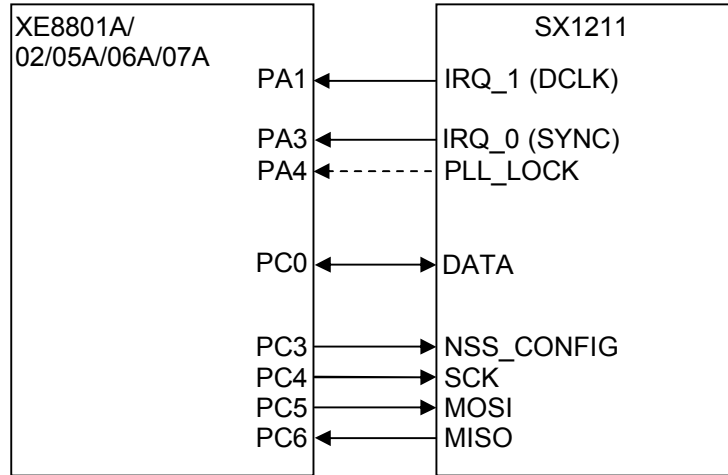
Note: DCLK and LD pins are not used in the current implementation. Connection is optional.

**7.2.13 Connections between SX1223 and XE8806A/07A BitJockey™**


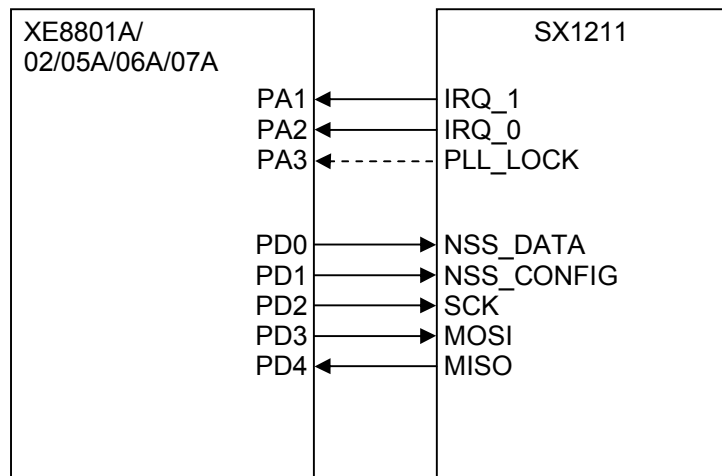
Note1: DCLK pin is not required when utilizing BitJockey™, thus it is not connected.

Note2: LD pin is not used in the current implementation. Connection is optional.

Note3: PD0, PD1, PD2 are not used (Tx only) but are reserved by the BitJockey™ peripheral

**7.2.14 Connections between SX1211 and XE8801A/02/05A/06A/07A BitBang (Continuous)**


Note: PLL\_LOCK signal is not used in the current implementation. Connection is optional.

**7.2.15 Connections between SX1211 and XE8801A/02/05A/06A/07A Buffered & Packet**


Note: PLL\_LOCK signal is not used in the current implementation. Connection is optional.

### 7.3 Glossary

|             |                                    |
|-------------|------------------------------------|
| <b>AFC</b>  | Automatic Frequency Control        |
| <b>API</b>  | Application Program Interface      |
| <b>FEI</b>  | Frequency Error Indicator          |
| <b>LO</b>   | Local Oscillator                   |
| <b>RSSI</b> | Received Signal Strength Indicator |

© Semtech 2007

All rights reserved. Reproduction in whole or in part is prohibited without the prior written consent of the copyright owner. The information presented in this document does not form part of any quotation or contract, is believed to be accurate and reliable and may be changed without notice. No liability will be accepted by the publisher for any consequence of its use. Publication thereof does not convey nor imply any license under patent or other industrial or intellectual property rights. Semtech. assumes no responsibility or liability whatsoever for any failure or unexpected operation resulting from misuse, neglect improper installation, repair or improper handling or unusual physical or electrical stress including, but not limited to, exposure to parameters beyond the specified maximum ratings or operation outside the specified range.

SEMTECH PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS. INCLUSION OF SEMTECH PRODUCTS IN SUCH APPLICATIONS IS UNDERSTOOD TO BE UNDERTAKEN SOLELY AT THE CUSTOMER'S OWN RISK. Should a customer purchase or use Semtech products for any such unauthorized application, the customer shall indemnify and hold Semtech and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs damages and attorney fees which could arise.

### Contact Information

Semtech Corporation  
Advanced Communication and Sensing Products Division  
200 Flynn Road, Camarillo, CA 93012  
Phone (805) 498-2111 Fax: (805) 498-3804