WirelessUSB™ LP/LPstar and PRoC™ LP/LPstar

Technical Reference Manual

Document # 001-12603 Rev. *G

August 1, 2011

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709

Phone (USA): 800.858.1810
Phone (Intnl): 408.943.2600
http://www.cypress.com

Copyrights

Disclaimer

Contents

Section A: Overview

This section lists the abbreviations and acronyms that Cypress uses in its product documentation.

Getting Started

The quickest path to understanding the WirelessUSB™ LP/LPstar and PRoC™ LP/LPstar is by reading the device's data sheet and application notes. This manual is useful for understanding the details of both the WirelessUSB LP/LPstar and the Low Power Programmable Radio System on a Chip–PRoC LP/LPstar (ICs).

Important Note For the most up-to-date ordering, packaging, or electrical specification information, refer to the individual device's data sheet or go to http://www.cypress.com.

Cypress Semiconductor Support

Technical Support can be reached at http://www.cypress.com/support or can be contacted by phone at: 1-800-541-4736.

The following related documentation can be found on the Cypress web site.

Document Number	Document Type	Document Name
38-16015	Data Sheet	CYRF6936 WirelessUSB LP 2.4GHz Radio SoC
001-07552	Data Sheet	CYRF69213 Programmable Radio on Chip-LP
001-07611	Data Sheet	CYRF69103 Programmable Radio on Chip–Low Power
001-66073	Data Sheet	CYRF6986 WirelessUSB LPstar 2.4GHz Radio SoC
001-66502	Data Sheet	CYRF69303 Programmable Radio on Chip Low Power
001-66503	Data Sheet	CYRF69313 Programmable Radio on Chip Low Power
AN4002	Application Note	Calculating Battery Life in WirelessUSB™ Systems
AN4003	Application Note	WirelessUSB™ 2-Way HID Systems
AN4004	Application Note	Interference Mitigation Challenges and Solutions in the 2.4 to 2.5 GHz ISM Band
AN6066	Application Note	Wireless Binding Methodologies
AN17581	Application Note	WirelessUSB™ LP/LPstar RDK Japanese Radio Law Testing and Verification
AN48610	Application Note	Design and Layout Guidelines for Matching Network and Antenna for WirelessUSB™ LP/LPstar
AN68105	Application Note	Migration of WirelessUSB™ LP Designs to WirelessUSB™ LPstar

Third-Party Support

There are two established providers of Cypress RF modules, Artaflex Inc. and Unigen Corp. These vendors can provide assembled and tested modules that are already EMC tested and approved for the various worldwide markets.

Cypress Semiconductor products are available from various worldwide distributors who can may also offer individual design assistance.

Document History

This section serves as a chronicle of the *WirelessUSB™ LP/LPstar and PRoC™ LP/LPstar Technical Reference Manual*.

WirelessUSB™ LP/LPstar and PRoC™ LP/LPstar Technical Reference Manual History

Release Date	Version	Originator	Description of Change
01/10/2007	**	ARI	This manual is a new document to the Cypress Document Control (Revision **) system; this manual has gone through several versions as an internal document. The manual itself is labeled Version 0.71 but is in fact the same as Version **. This document has been issued the document number 001-12603.
03/31/2007	*A	ARI	Added a Document History section. Added the Resets chapter.
06/28/2007	*B	ARI	Renamed the Power Management chapter to Managing Power. Added the Power Management Unit chapter.
09/19/2007	*C	ARI	Converted manual to two-column format.
04/25/2008	*D	HMT	Add Interrupts chapter. Update Introducing WirelessUSB™ LP chapter. Add Backward Compatibility. Update Crystal chapter. Update Registers chapter. Update pinout, reset, and crystal figures.
11/08/2010	*E	HEMP	Overview: Expand the App. Notes list. Add 3rd party section. Add abbreviation. Chap. 1: update 1.2.1. Fix p/n in Table 1-3. Change molded part number to the saw-cut part number. Chap. 3: 3.2.1: correct Kemet p/n and add note about capacitor reliability. Chap. 6: 6.1.3.1: Add additional detail about inductor saturation.
05/05/2011	*F	KKCN	Add LPstar to this TRM to make it compatible with LP/LPstar
08/01/2011	*G	KPMD	Changed posting to external web.

Document Conventions

This manual uses the `Courier New` font to distinguish code examples from regular text. File names are presented in *italics* text.

Register Conventions

The register conventions that are specific to this manual are described in Registers chapter on page 105.

Numeric Naming

Hexidecimal numbers are represented with all letters in uppercase with an appended lowercase ‘h’ (for example, ‘14h’ or ‘3Ah’) and hexidecimal numbers may also be represented by a ‘0x’ prefix, the C coding convention. Binary numbers have an appended lowercase ‘b’ (for example, 01010100b’ or ‘01000011b’). Numbers not indicated by an ‘h’ or ‘b’ are decimal.

Units of Measure

The following table lists the units of measurements that are used in this manual.

Units of Measurement

Symbol	Unit of Measure
°C	degree Celsius
dB	decibels
GHz	giga hertz
Hz	hertz
k	kilo, 1000
K	2^{10}, 1024
KB	1024 bytes
Kbit	1024 bits
Kbps	kilobits per second
kHz	kilohertz (32.000)
kΩ	kilohm
Mbps	megabits per second
MHz	megahertz
MΩ	megaohm
µA	microampere
µF	microfarad
µH	microhenry
µs	microsecond
mA	milli-ampere
ms	milli-second
mV	milli-volts
nA	nanoampere
ns	nanosecond
Ω	ohm
pF	picofarad
ppm	parts per million
sps	samples per second
V	volts

Acronyms

The following table lists the acronyms that are used in this manual.

Acronyms

Acronym	Description
8DR	eight x data rate
AC	alternating current
ACK	acknowledge
ADC	analog-to-digital converter
API	application programming interface
ATS	auto transaction sequencer
BER	bit error rate
BOM	bill-of-materials
BR	bit rate
CMRR	common mode rejection ratio
CPU	central processing unit
CRC	cyclic redundancy check
DC	direct current
DDR	double data rate
DMA	direct memory access
DSSS	direct sequence spread spectrum
EEPROM	electrically erasable programmable read only memory
EFTB	electrostatic fast transient burst
EMC	electromagnetic compatability
EMI	electromagnetic interference
EOP	end of packet
ESD	electro-static discharge
FEC	forward error correction
FCC	federal communications commission
FHSS	frequency hopping spread spectrum
FIFO	first in first out
FSK	frequency-shift keying
GFSK	Gaussian frequency shift key
GPIO	general purpose IO

Acronyms *(continued)*

Acronym	Description
HID	human interface device
ICE	in-circuit emulator
IDE	Integrated Development Environment
ILO	internal low speed oscillator
IMO	internal main oscillator
IO	input/output
IRQ	interrupt request
ISM	industrial, scientific and medical
ISR	interrupt service routine
ISSP	in system serial programming
LDO	low drop-out
LPF	low pass filter
LSb	least significant bit
LSB	least significant byte
LVD	low voltage detect
Mbps	megabits per second
MISO	master-in-slave-out
MOSI	master-out-slave-in
MSb	most significant bit
MSB	most significant byte
PER	packet error rate
PCB	printed circuit board
PMU	power management unit
PN	pseudo noise
POR	power on reset
RAM	random access memory
RF	radio frequency
RSSI	receive signal strength indication
RST pin	reset pin
RW	read/write
SDR	single data rate
SIE	serial interface engine
SNR	signal-to-noise ratio
SOF	start of frame
SOP	start of packet
SPI	serial peripheral interface
SPIM	serial peripheral interconnect master
SPIS	serial peripheral interconnect slave
SRAM	static random access memory
USB	universal serial bus
VCO	voltage controlled oscillator

1. Introducing WirelessUSB™ LP/LPstar

1.1 Introduction

Welcome to Cypress WirelessUSB™ and PRoC™ – Programmable Radio System-on-Chip. Cypress is now on its second generation of wireless technology, which is the focus of this manual. The first generation products included WirelessUSB LS™, WirelessUSB LR™, and the original PRoC device. Although there were different features and capabilities in these devices, they all fundamentally used the same radio and baseband technology.

In the same way, Cypress developed a new generation wireless technology to leverage among several devices. Because of the common underlying intellectual property these devices possess, they are discussed together with only minor diversions to highlight differences in capability. Therefore, the scope of this manual includes the following Cypress products:

- WirelessUSB LP
  - CYRF6936-40LTXC
- WirelessUSB LPstar
  - CYRF6986-40LTXC
- Low Power Programmable Radio System-on-Chip (PRoC LP)
  - CYRF69213-40LFXC
  - CYRF69103-40LFXC
- Low Power Programmable Radio System-on-Chip (PRoC LPstar)
  - CYRF69313-40LFXC
  - CYRF69303-40LFXC

All of these devices are 2.4 GHz Direct Sequence Spread Spectrum (DSSS) transceivers operating in the unlicensed worldwide Industrial, Scientific and Medical (ISM) band (2.400–2.483 GHz). They add a range of enhanced features over first generation devices, including increased operating voltage range(only for LP), reduced supply current in all operating modes, higher data rate options(only for LP), and reduced crystal start up, synthesizer settling, and link turn around times. In addition, WirelessUSB and PRoC LP/LPstar combat interference, achieve a better wireless communication link, and offer robust performance.

This manual supplements the data sheets for each of these devices. Although there is some duplication of information for convenience, this document goes beyond the data sheets. Where the data sheets focus on hardware and simple register descriptions, this manual focuses on ‘how to use’ the features discussed in the data sheet, as well as on system level issues.

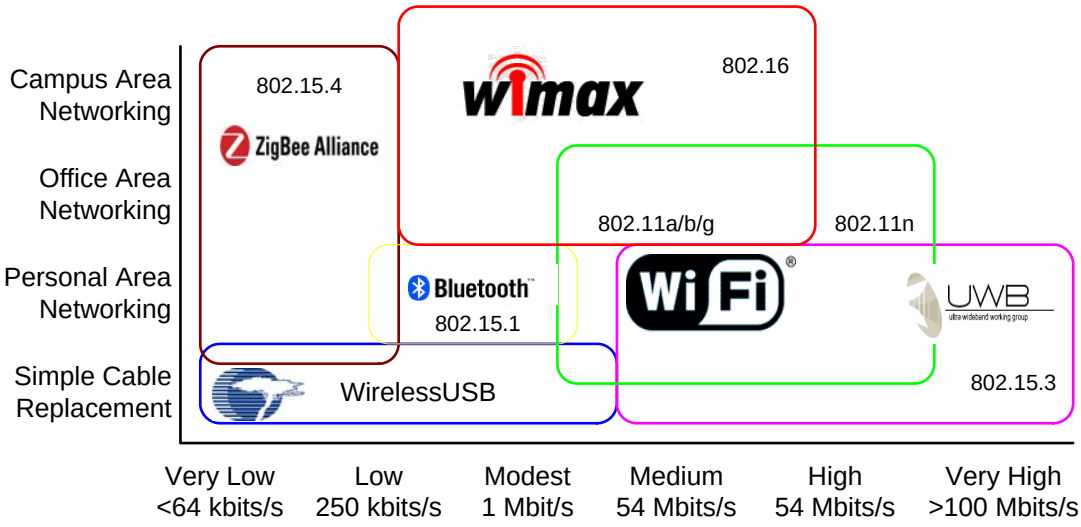
One additional note about the PRoC LP/LPstar devices: PRoC LP/LPstar is the marriage of IP from the WirelessUSB LP/LPstar radio with the microcontroller IP from the Cypress enCoRe™ II and Wireless enCoRe II devices. The data sheets for the PRoC LP/LPstar devices include a substantial amount of information about the microcontrollers, see Cypress Semiconductor Support on page 9. Furthermore, the PSoC Designer™ tool set, which is used for enCoRe II development, includes examples and firmware IP to jump start development with the features of the microcontrollers. Therefore, when discussing PRoC LP/LPstar devices, this manual is primarily concerned with the radio portion of the products.

1.2 Introduction to Wireless

Wireless technologies have gained rapid acceptance in the marketplace and the growth continues to accelerate. The ability to move data without having to connect a cable, run wires, or worry about having the right adapters is very appealing. Wireless offers the promise of convenience, speed, and ease of use.

The first thing to understand is that there are many different wireless technologies in use or emerging today. Examples include WiFi®, Bluetooth®, Zigbee™, Ultra-Wide Band (UWB), and many more.

Figure 1-1. Wireless Technologies



These technologies differ in many ways, such as frequency spectrum, data rates, data encoding, protocols, network topologies, and others. So where does Cypress WirelessUSB technology fit in? The Cypress WirelessUSB and PRoC products are aimed at mid to low data rates (up to 1 Mbps) and simple cable replacement or simple network topologies.

Although there are many Cypress WirelessUSB and PRoC products users who are experienced RF designers, the lure of 'going wireless' coupled with the accessible price point has introduced many new vendors with little or no experience with wireless products.

1.2.1 The Medium

Wireless technology involves transmission of information using electromagnetic waves at radio frequencies (RF).

There are a large number of factors that can influence what happens in this channel. Some of these factors are discussed in more detail in later sections, but here are some simple examples:

- Attenuation due to range
- Attenuation due to passing through objects such as product enclosures or walls
- Multi-path effects from reflections off walls, floors, or other objects
- Interference from other signals

All of these things impact the successful transmission of the signal from its source to its destination, and have it properly received.

1.2.2 Moving Data

Wireless is still all about moving information from point A to point B. Depending upon the needs of the application, data may move in one direction only, or it might need to move in both directions. An important thing to note is that WirelessUSB only supports moving data in one direction at a time, thus it is half duplex.

WirelessUSB systems also typically acknowledge receipt of information. Although this feature is optional, it adds tremendously to the robustness of the system and is fully automated in WirelessUSB LP/LPstar and PRoC LP/LPstar. The implication of acknowledgements is that even for systems that only need to send data in one direction, there is still communication in both directions. One side of the system transmits its information. When the other side receives it properly, it transmits an acknowledgement, or ACK, back to the originating station. Thus both sides know that the data arrived at its destination. If for some reason the originating station did not receive the ACK within a specified time period, it can make the decision to retransmit the data.

Systems requiring bidirectional transmission of data make use of the above method, but interleave the communications. This can vary greatly based on the needs of the application, but in the simplest form Station A transmits data to Station B. When Station B receives the data, it replies with an ACK. At this time, Station B can send data to Station A and Station A transmits an ACK when it receives the data. The timing and order of these interactions are based completely on the needs of the system.

1.2.3 Structure of Data

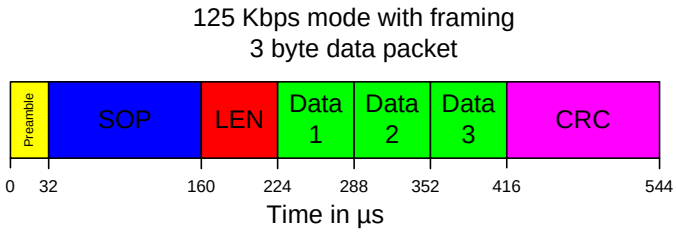
In many systems where data is moved from point to point, there is some level of framing of the data. This is as simple as using start and stop bits on a UART to more complex methods that allow error detection or other functions. WirelessUSB systems have to use some level of framing, but there are various options.

In WirelessUSB systems the data is packetized. The transmission starts with a preamble, a preset sequence transmitted over the air that is not part of the meaningful data, that helps the correlator on the receiving side to synchronize with the signal so that it can receive it properly.

There is also a limit to the amount of information contained within a packet. In wireless systems the receiver has to contend with distinguishing the real signal from the ever present noise in the channel. It is not as simple as the wired equivalent where a binary '1' is 5V and a binary '0' is 0V. Therefore, there is a data limit based on the potential timing mismatch between both sides of the system beyond which the receive correlator can get out of synchronization with the data stream.

Other options in WirelessUSB include using start of packet (SOP) symbols, a length field, and a cyclic redundancy check (CRC) field for error detection. These are also discussed in more detail in later sections.

Figure 1-2. Sample Packet Structure



1.2.4 Protocol

Some simple systems plausibly just send and receive raw data. However, to compete in the marketplace, most products require a high level of robustness and some level of protocol. Protocols involve an understanding by both sides of the system as to what the specific pieces of data transmitted mean.

As an example, the data portion of a WirelessUSB packet might consist of a single header byte followed by other data bytes. Some bits in the header byte could inform the receiving side of the meaning of the subsequent data bytes. For instance, they could distinguish between a packet from a temperature sensor versus a packet from a wall switch in a building control network. Other bits might provide status, such as "My receive buffer is full" or "empty". Other elements of the protocol are sometimes used to help all nodes in the system change channels when in the presence of an interfering signal that is causing loss of communication.

Implementing a robust protocol is quite challenging. Cypress has a couple of key examples that are used in some of its reference design and development kits. Although these are not industry standard protocols, like you might find with Bluetooth for example, they are widely re-used by its customers. Cypress has done thorough QA testing on the examples, and these examples are in use in high volume production with many customers.

1.3 Simple Network Parameters

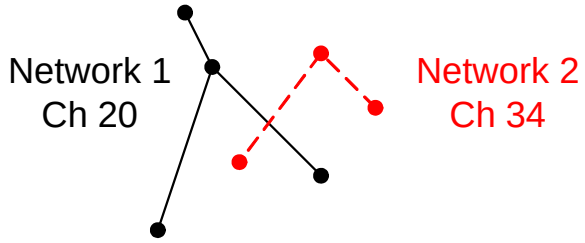
For experimenting with WirelessUSB, and even for some very simple real-world applications, it is possible to fix certain system parameters that govern how data is transmitted between elements in the system. But for most real products, it is necessary to establish some level of network.

For now, keep the definition of 'network' simple and say that it means establishing a couple of key parameters that let specific devices communicate with one another. It may be possible to have other devices communicating in the same general vicinity but not interacting with the network. This is possible by giving them a different set of network parameters.

1.3.1 Channel

The channel establishes the frequency band over which the network transmits its data. WirelessUSB allows discrete selections that are spaced at 1 MHz intervals. For example, our network might transmit on frequency 2.410 GHz. Another network in our vicinity might choose to use 2.412 GHz. The two networks can operate in proximity but cannot exchange information with one another. From a general standpoint, they also do not interfere with one another because their RF energy is in different frequency bands. The term for this is 'co-location'. In many applications it is desirable or even required to allow multiple networks to be co-located in the same area without interfering with each other's operation.

Figure 1-3. Two Co-located Networks



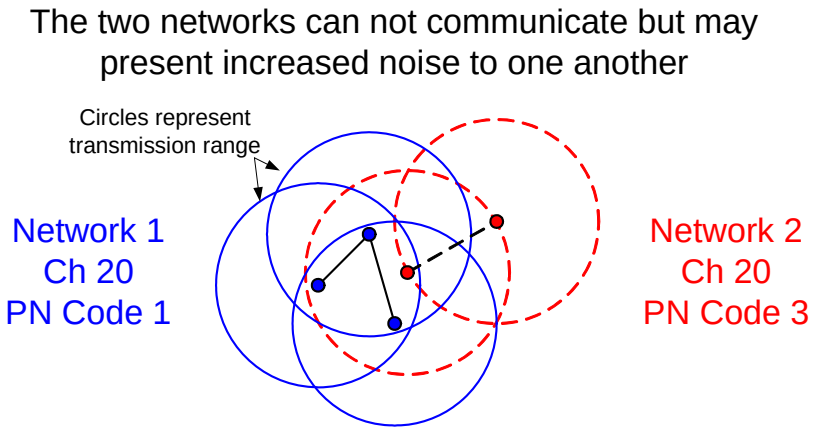
1.3.2 PN Code

Because they use direct sequence spread spectrum (DSSS) technology, WirelessUSB systems encode their data within Pseudo Noise (PN) codes. The main advantage is to increase the robustness and recoverability of the signal in the presence of interference. A simple explanation is that a single data bit from the application is represented by multiple bits when sent across the air, and decoded back into the original data bit on the receiving side.

One result of using PN codes is that devices in a given network must agree to use a common PN code in order to understand one another. Another advantage of this is that it increases the co-location capabilities since devices can share the same channel if they use different PN codes.

Devices with different PN codes but on the same frequency cannot communicate with one another, but might present some interference to one another. Because their RF energy is in the same frequency band, information transmitted within one network may be ‘heard’ by devices in another network, even though those devices cannot ‘understand’ the information because of the different PN code. The data may be perceived as interference, which could impact the reception of data if the two networks happen to have transmissions within the exact same time interval. A robust protocol must handle this situation with no observable impact by the end user.

Figure 1-4. Networks on Separate PN Codes



WirelessUSB LP/LPstar devices also offer 1 Mbps Gaussian Frequency Shift Key (GFSK) modes that do not make use of PN codes. In this case, all devices on the same frequency can communicate with one another and co-location capabilities are potentially reduced.

1.3.3 Other Parameters

To enable communication within a network there are other configuration options that must be agreed upon by all devices within the system, such as whether or not they will be using the various packet framing options like the start of packet symbols, length field, or CRC.

With CRC, an optional CRC seed can also be specified if wanted. If used, this is one more means to allow co-location of multiple networks. A system can reject data that is received if it does not match the appropriate CRC seed.

Most of these other options are discussed in greater detail in other sections of this manual.

1.4 WirelessUSB LP/LPstar Features Summary

WirelessUSB LP includes the following key features:

- 2.4 GHz Direct Sequence Spread Spectrum radio transceiver
- Operates in the unlicensed worldwide Industrial, Scientific and Medical (ISM) band (2.400 GHz–2.483 GHz)
- 21 mA operating current (transmit @ –5 dBm)
- Transmit power up to +4 dBm
- Receive sensitivity up to –97 dBm
- Sleep current <1 µA
- DSSS data rates up to 250 kbps, GFSK data rate of 1 Mbps
- Low external component count
- Auto Transaction Sequencer (ATS)–no MCU intervention
- Framing, length, CRC16, and Auto ACK
- Power Management Unit (PMU) for MCU/Sensor
- Fast start up and fast channel changes
- Separate 16 byte transmit and receive FIFOs
- AutoRate™–dynamic data rate reception
- Receive Signal Strength Indication (RSSI)
- Serial Peripheral Interface (SPI) control while in sleep mode
- 4 MHz SPI microcontroller interface
- Battery voltage monitoring circuitry
- Supports coin-cell operated applications
- Operating voltage from 1.8 V to 3.6 V
- Operating temperature from 0 to 70°C
- Space saving 40-pin QFN 6x6 mm package

For LPstar features please refer to the Features in CYRF6986 datasheet and the application note-AN68105-Migration of WirelessUSB™ LP Designs to WirelessUSB™ LPstar.

1.4.1 Backward Compatibility

The CYRF6936 IC is fully interoperable with the main modes of the first generation devices, namely the CYWUSB6934-LS and CYWUSB6935-LR devices. The 62.5 kbps mode is supported by selecting 32 chip DDR mode. Similarly, the 15.675 kbps mode is supported by selecting 64 chip SDR mode. In this way, a suitably configured CYRF6936 IC device may transmit data to or receive data from a first generation device, or both. Backward compatibility requires disabling the SOP, length, and CRC16 fields.

The CYRF6986 IC is fully interoperable with the main modes of CYRF6936 by selecting 32 chip 8DR mode or GFSK mode.

The following tables show the different configurations of the registers and firmware that enable communication between a second generation radio and a first generation radio. There are two possible modes: SDR mode and DDR mode (8-DR and GFSK modes are not present in the first generation radio). The second generation radio must be initialized using the RadioInitAPI of the LP radio driver and then the following register bits must be configured to the given byte values. Essentially, the following deactivates the added features of the second generation radio and takes it down to the level of the first generation radio. The data format, data rates, and the PN codes used are recognizable by the first generation radio.

1.4.1.1 DDR Mode

Table 1-1. DDR Mode

Register	Value	Description
TX_CFG_ADR	0X16	32 chip PN Code, DDR, PA = 6.
RX_CFG_ADR	0X4B	AGC is enabled. LNA and attenuator are disabled. Fast turnaround is disabled, the device uses high side receive injection, and Hi-Lo is disabled. Overwrite to receive buffer is enabled and the RX buffer is configured to receive 8 bytes maximum.
XACT_CFG_ADR	0X05	AutoACK is disabled. Forcing END STATE is disabled. The device is configured to transition to Idle mode after a receive or transmit. ACK timeout is set to 128 µs.
FRAMING_CFG_ADR	0X00	All SoP and framing features are disabled. Disable LEN_EN=0 if EoP is needed.
TX_OVERRIDE_ADR	0X04	Disable Transmit CRC-16.
RX_OVERRIDE_ADR	0X14	The receiver rejects packets with a zero seed. The RX CRC-16 Checker is disabled and the receiver accepts bad packets that do not match the seed in CRC_Seed registers. This helps communication with the first generation radio, which does not have CRC capabilities.
ANALOG_CTRL_ADR	0X01	Set ALL SLOW. When set, the synthesizer settle time for all channels is the same as the slow channels in the first generation radio.
DATA32_THOLD_ADR	0X03	Sets the number of allowed corrupted bits to 3.
EOP_CTRL_ADR	0x01	Sets the number of consecutive symbols for noncorrelation to detect EoP.
PREAMBLE_ADR	0xAAAA05	AAAA are the 2 preamble bytes. Other bytes can also be written into the Preamble register file. The recommended preamble bytes to be sent must be >4.

1.4.1.2 SDR Mode

Table 1-2. SDR Mode

Register	Value	Description
TX_CFG_ADR	0X3E	64 chip PN code, SDR mode, PA = 6.
RX_CFG_ADR	0X4B	AGC is enabled. LNA and attenuator are disabled. Fast turnaround is disabled, the device uses high side receive injection, and Hi-Lo is disabled. Overwrite to receive buffer is enabled and RX buffer is configured to receive 8 bytes maximum. Enables RXOW to allow loading new packets into the receive buffer. This also enables the VALID bit, which is used by the first generation radio's error correction firmware.
XACT_CFG_ADR	0X05	AutoACK is disabled. Forcing END STATE is disabled. The device is configured to transition to Idle mode after a receive or transmit. ACK timeout is set to 128 µs.
FRAMING_CFG_ADR	0X00	All SoP and framing features are disabled. Disable LEN_EN=0 if EoP is needed.
TX_OVERRIDE_ADR	0X04	Disable Transmit CRC-16.
RX_OVERRIDE_ADR	0X14	The receiver rejects packets with a zero seed. The RX CRC-16 Checker is disabled and the receiver accepts bad packets that do not match the seed in CRC_Seed registers. This helps communication with the first generation radio, which does not have CRC capabilities.
ANALOG_CTRL_ADR	0X01	Set ALL SLOW. When set, the synthesizer settle time for all channels is the same as the slow channels in the first generation radio for manual ACK consistency.
DATA64_THOLD_ADR	0X07	Sets the number of allowed corrupted bits to 7, which is close to the recommended 12% value.
EOP_CTRL_ADR	0x01	Sets the number of consecutive symbols for noncorrelation to detect EoP.
PREAMBLE_ADR	0xAAAA09	AAAA are the 2 preamble bytes. Other bytes can also be written into the Preamble register file. The recommended preamble bytes to be sent must be >8.

For further details, see Backward Compatibility on page 63.

1.5 PRoC LP/LPstar Features Summary

In addition to the complete set of radio capabilities described in the preceding section, PRoC LP integrates the Cypress industry leading enCoRe II and Wireless enCoRe II microcontroller technologies for a single system-on-chip solution. enCoRe II is Cypress's sixth generation low speed USB controller. Wireless enCoRe II shares the same core microcontroller technology, but in a non-USB, low voltage configuration that is perfect for use in power sensitive wireless applications.

The microcontroller function adds the following key features to both PRoC devices:

- M8C based 8-bit CPU, optimized for HID applications
- No crystal required for the microcontroller–operates independent of radio crystal's state
- 8K bytes of Flash memory with EEPROM emulation
- 256 bytes of SRAM
- In-system re-programmable
- 16 bit free running timer
- Low power wake up timer
- 12 bit programmable interval timer with interrupts
- Watchdog timer
- High current drive on GPIO pins. Configurable 8 mA or 50 mA/pin current sink on designated pins
- Each GPIO pin supports high-impedance inputs, configurable pull up, open-drain output, CMOS/TTL inputs and CMOS output
- Maskable interrupts on all IO pins

Additional features specific to the CYRF69213 USB device include:

- Integrated 3.3 V regulator (supplies radio function and external devices)
- Integrated pull up on D–
- Programmable though the USB connector (programming pins shared with D+/D–)
- Conforms to USB Specification Version 2.0
- Conforms to USB HID Specification Version 1.1
- Supports one low-speed USB device address
- Supports one control endpoint and two data end points
- Integrated USB transceiver with a second dedicated 3.3 V regulator
- Operating voltage from 4.0V to 5.5 V DC

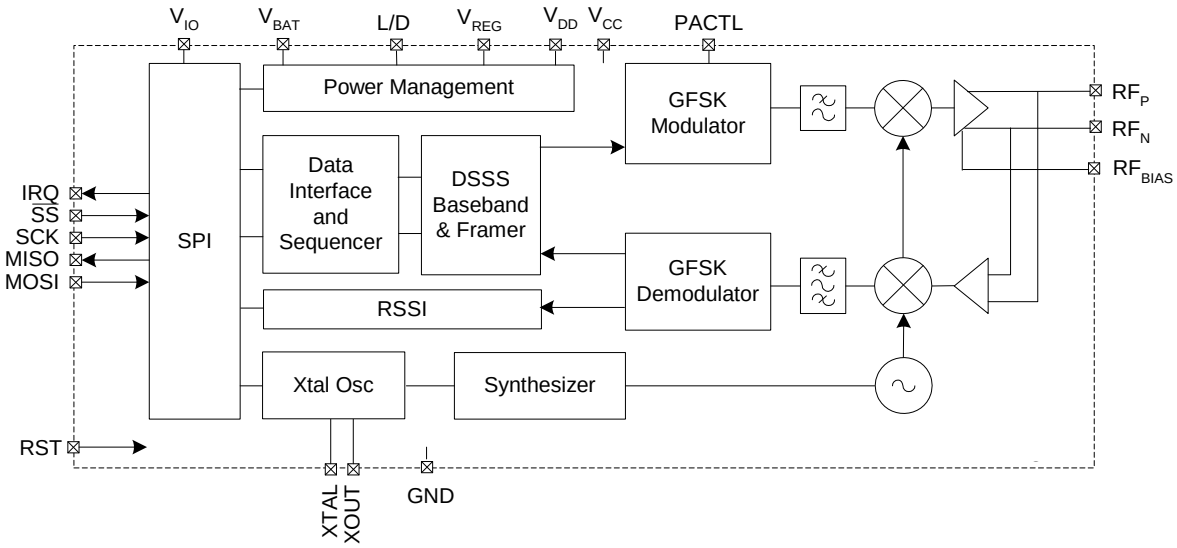
For PRoC LPstar features please refer to the Features in CYRF69303/313 datasheets.

1.6 Block Diagrams

1.6.1 WirelessUSB LP/LPstar

The WirelessUSB LP chip is comprised of two main functional blocks; one is the radio core and the other is the radio control/ baseband logic. For LPstar, please refer to the Logic Block Diagram in CYRF6986 datasheet.

Figure 1-5. WirelessUSB LP Block Diagram



1.6.2 PRoC LP

As discussed in section 1.5 PRoC LP/LPstar Features Summary on page 19, PRoC LP integrates the radio and microcontroller functions in a single package. The inner blocks of the radio function are equivalent to those shown in Figure 1-5.For PRoC LPstar, please refer to the Logic Block Diagram in CYRF69303/313 datasheets.

Figure 1-6. PRoC LP CYRF69213 Block Diagram

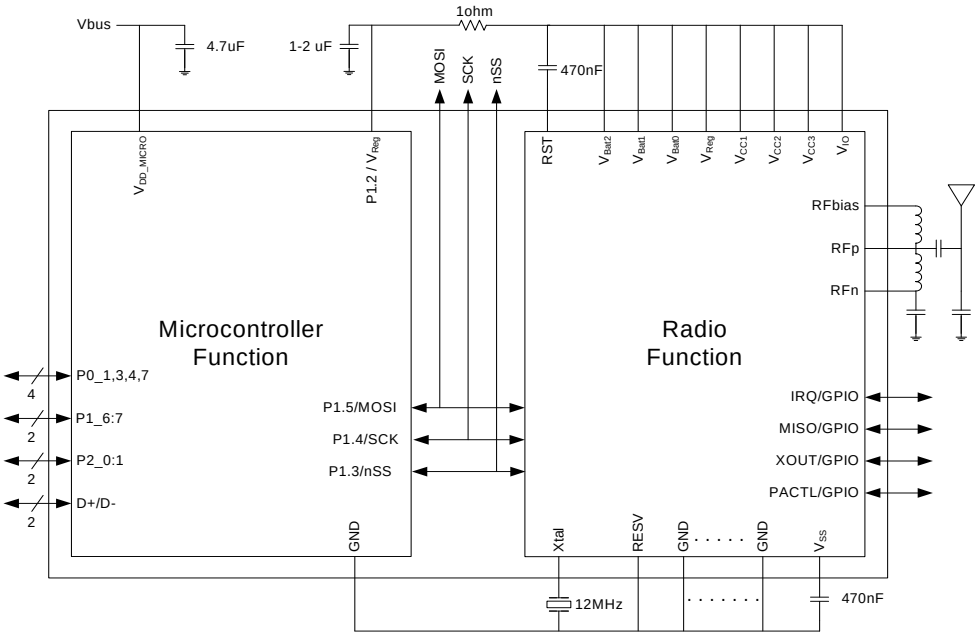
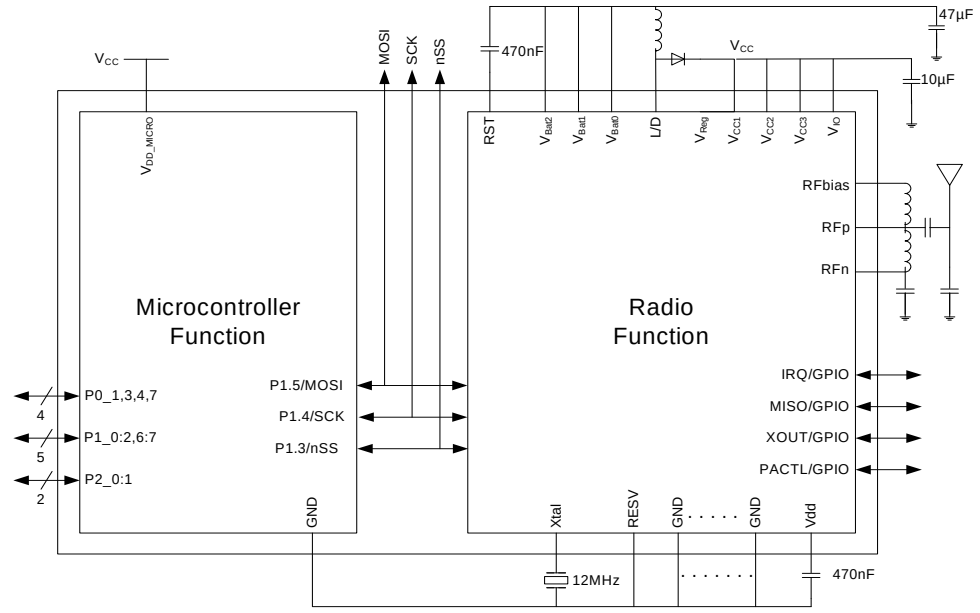


Figure 1-7. PRoC LP CYRF69103 Block Diagram



1.7 Device Options

Table 1-3. Device Options

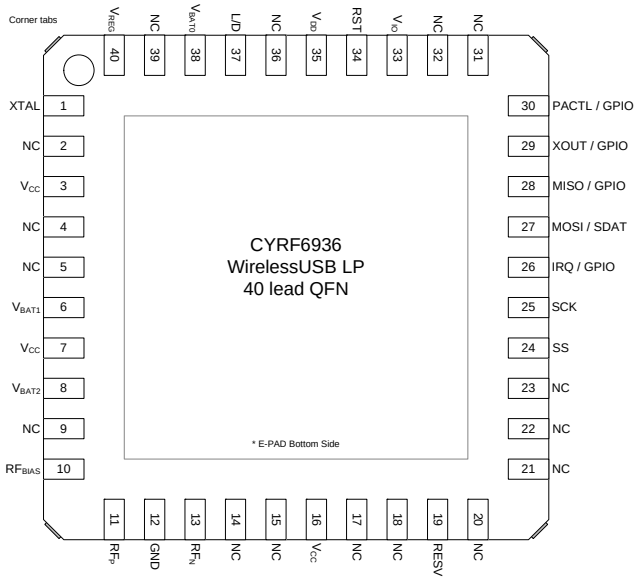
Device	Part Number	MCU	USB	Memory	Operating Voltage	V_{reg} Output	Boost Output
WirelessUSB LP	CYRF6936-40LTXC	N/A	N/A	N/A	1.8–3.6	N/A	2.4–2.7V
WirelessUSB LPstar	CYRF6986-40LTXC	N/A	N/A	N/A	2.7–3.6	N/A	
PRoC LP	CYRF69213-40LFXC	M8C	N/A	8K Flash	4.0–5.5	3.3V	N/A
PRoC LPstar	CYRF69313-40LFXC	M8C	N/A	8K Flash	4.0–5.25	3.3V	N/A
PRoC LP	CYRF69103-40LFXC	M8C	Low Speed	8K Flash	1.8–3.6	N/A	2.4–2.7V
PRoC LPstar	CYRF69303-40LFXC	M8C	Low Speed	8K Flash	2.7–3.6	N/A	

1.8 Packages

Following are the packages of WirelessUSB LP and PRoC LP. For WirelessUSB LPstar and PRoC LPstar, please refer to the their datasheets and the application note AN68105 that points out the package differences in LP and LPstar.

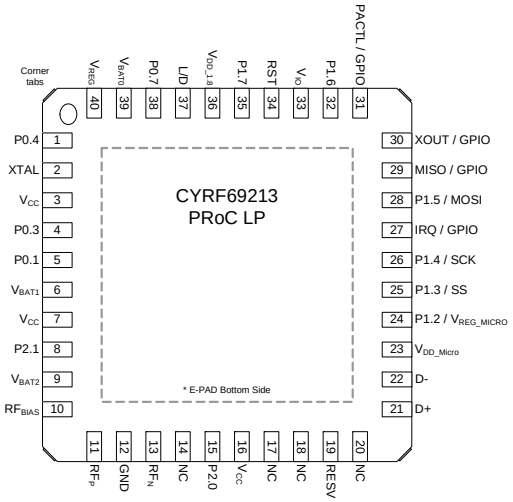
1.8.1 CYRF6936-40LTXC 40-QFN Package

Figure 1-8. CYRF6936-40LTXC Package



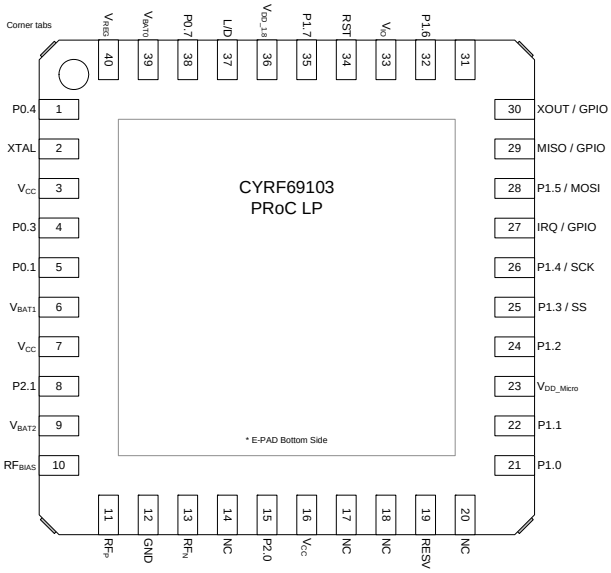
1.8.2 CYRF69213-40LFXC 40-QFN Package

Figure 1-9. CYRF69213-40LFXC Package



1.8.3 CYRF69103-40LFXC 40-QFN Package

Figure 1-10. CYRF69103-40LFXC Package



2. Design Considerations

2.1 Introduction

Wireless systems are complicated. Engineers have to manage all of the standard challenges associated with any electronic design, but there are now added dimensions in the mix. By definition a wireless system involves multiple nodes that must exchange data asynchronously. Without a physical connection there is not even assurance that data reaches its destination. Getting proper RF performance out of a product can be a multifaceted design effort in and of itself.

Although there are many RF knowledgeable designers using WirelessUSB, there are many more who are approaching wireless for the first time. Introducing WirelessUSB™ LP/LPstar, on page 13 already pointed out the lure of wireless: the capabilities wireless technology brings to the table, combined with the approachable price point have driven many companies to make the move to wireless for their products. But they often make this move without a full understanding of all of the system requirements that must be taken into account. Even experienced designers may want to go back to the basics every now and then to make sure they are not forgetting key points.

The purpose of this chapter is to review many of the considerations that must be taken into account before starting work on a wireless project. The intent is not necessarily to provide solutions, but to make sure that designers are aware of the right questions to ask themselves before diving into the design work. It is placed in front of much of the technical information concerning the WirelessUSB products for that very reason.

As with any complex project, forethought and careful advanced planning can keep a project on track, minimize risk, prevent unnecessary rework, and result in a product that has the right level of appeal to customers in its target market space. This chapter is intended primarily for system architects who will define the specifications for their project, but it may also be valuable to marketing personnel who will lay out the initial product feature requirements.

2.2 Power

For many wireless designs, power is one of the most critical considerations. One thing is almost certain: for a wireless system, the overall system power consumption will almost certainly be greater than that for a wired equivalent system. So where power budget may have been of trivial consequence in a legacy wired product, as companies move to wireless technology they may find that they have to put additional constraints on their design to manage power.

2.2.1 Power Source

There are two important distinctions to draw with respect to the power supply. Devices are generally supplied from a source that is either essentially limitless, or from batteries where the supply has a significant limit. One other key thing to note is that a complete system may involve different nodes that fit under different categories.

2.2.1.1 'Unlimited' Power Supplies

Although the term 'unlimited' is used, this is generally only to set it apart in order of magnitude from battery powered products. Even devices with effectively limitless power supplies probably still have a power budget. Draining batteries may not be a concern, but there are other factors, such as heat generation, that may place limits on the system.

Devices that derive their power from a wall power supply certainly fall into this category. Devices that draw their power from other systems can also fit this category. As a specific example, consider a product that plugs into a USB port and uses USB as its power source.

The key characteristic of devices that fit into this category is that they have the flexibility to choose any operating mode of the radio without regard to power consumption. In simpler terms, a radio could stay in receive mode constantly so that it is ready to receive information at any instant.

Revisiting the USB example, note that both desktop PCs that plug into the wall, as well as laptops with their own battery are placed in this category. The laptop is considered to have a power source of a significantly large order of magnitude that it does not place significant constraints on the operation of the device. So although devices in this category may still have power budget concerns, the flexibility of oper-

ating the radio in any mode takes precedence, and power considerations become a secondary issue.

2.2.1.2 Battery Powered Devices

In contrast with the above category, battery powered devices have a limited source of power, and must operate in a constrained fashion to effectively use the power supply. ‘Battery life’, the amount of time that the product can operate without replacing or recharging batteries, is likely to be a key marketing metric.

The constraint on these devices is that the radio will not be able to operate in any mode desired at any time. Radios have different operating modes with varying levels of current consumption. For example, WirelessUSB LP/LPstar devices have sleep, idle, synthesizer settle, transmit and receive modes, each with their own power consumption numbers. There are trade offs in responsiveness with respect to the different modes as well. Staying in transmit mode means that the radio can transmit a packet at any instant with negligible latency, but it consumes a high amount of current. Conversely, keeping the radio in sleep mode consumes negligible current, but means that it will take a relatively long time to wake up the radio and ready it for transmitting data. Within transmit mode there are also varying Power Amplifier (PA) levels that have different transmit power levels with a corresponding current consumption. Higher PA levels mean longer range, but greater power consumption.

Battery powered devices will typically have to define various operating modes that balance power consumption with other parameters such as responsiveness or range. These modes may apply to the radio, but they may also apply to other IC’s in the device such as external microcontrollers, optical sensors, and others. A typical device might be in sleep mode for a substantial period of time. It could then wake up based on a predefined interval or event, at which time different portions of the system can be powered up to perform appropriate tasks, such as the transmission of data.

2.2.2 Peak and Average Current

Given that the IC’s in the circuit will operate off of given voltage levels, the real parameter to analyze is current.

Peak current for a given operating mode can be a significant parameter, but generally only for devices that will keep the radio in a high power mode for a long period of time. This is typically only going to occur for those with unlimited power supplies. The primary example would be a device that keeps the radio in receive mode constantly in order to be ready to receive a packet at any time. The WirelessUSB LP/LPstar’s maximum Rx power parameter would probably be the one used in power budget calculations.

For devices, especially battery powered devices, that vary the operating mode of the radio frequently, average current is typically a much more significant metric. There are different profiles here, such as devices that alternate between transmit and receive constantly, but the more likely profile is one in which the device stays asleep for most of the time, and then wakes up, performs some task such as transmitting a packet, and then goes back to sleep. In this case the time in each mode versus the corresponding current consumption should be evaluated to determine the true power consumption of the device.

2.2.3 Voltage Range

Voltage range is generally a much less complicated issue, but it is dealt with here only because there are different capabilities of the WirelessUSB LP/LPstar and PRoC LP/LPstar products that may have an effect on the system power consumption.

A given device typically contains multiple IC’s that may have different operating voltage requirements. WirelessUSB LP contains the PMU that can power the radio as well as external devices at 2.4V–2.7V levels when boosting from 1.8V(LPstar doesn’t support this). PRoC LP contains the same PMU, and the CYRF69213 version also contains a regulator that can supply 3.3V to itself and external devices from a 5V supply(CYRF69313 also supports this). Both of these power management systems have current limits that must be taken into account, as well as different capabilities in sleep and awake operating modes.

Designers must identify the voltage and current needs of all devices in the system and assess if the PMU or voltage regulator are required, or appropriate, for the system. All operating modes of different IC’s in the system must be taken into account. In some cases an external component may be required, such as a battery powered device with another IC requiring a minimum of 3.0V. In this case the PMU is unable to boost to that level and an external DC-DC converter may be required. The designer may still have to determine if the DC-DC converter will supply all devices on the board, or if it will power only that IC while the PMU manages power for the radio. There will be trade offs in cost and efficiency that have to be considered.

2.3 Range

For wireless systems the distance over which information can be transmitted is obviously a key concern. But range is also a complex parameter with many factors affecting the achievable range in a real world environment. In general, range varies directly with the output power of the transmitter. Receive sensitivity of the receive side also plays a factor. Some of the high level interactions in the channel were discussed in the section "The Medium" on page 14. Between the transmitter and receiver the signal are attenuated by spreading, passing through objects, and interactions with other signals or its own reflections. The power and noise levels between the transmitter and receiver as well as all of the gain and loss factors between them define the link budget.

Marketing or systems architects must determine if there is a range requirement for the system. They should also attempt to define the environment in which the system is expected to be used. Examples include out door line-of-sight, in a home where there are lots of walls to pass through, or in an office environment where there may be a large open space but filled with many cubicles. Is a significant amount of interference expected? These items are external factors over which the designers have little control, however they do affect the link budget, so they must be taken into account first. At a first level this analysis determines if the product is even viable. From there the system architect can move to an evaluation of internal factors.

As with most parameters, range can be trade off against other items. If the link budget is not sufficient then what can be done to increase it? Or if it is above the requirements then it may be possible to drop it down to a minimum acceptable level, and thus make gains in other parts of the system.

Different transmit powers obviously produce different maximum ranges. Data rates also have an impact. WirelessUSB products have many data rates to choose from with their own sets of trade offs. Component selection also has an impact. RF performance can be impacted by power supply noise, antenna type, antenna matching, crystal oscillator performance, and other factors. There are many component choices that can directly impact these factors. As a general rule, more costly components produce better performance.

2.4 Throughput

Knowing how much data must be moved through the system is another significant point to define. There may be different operating modes of the system that might have their own throughput requirements so each must be considered separately. From this starting point, other key parameters can be defined.

2.4.1 Data Rate

For high data rate systems (‘high’ with respect to WirelessUSB’s capabilities) throughput is probably most closely related to data rate. The necessary throughput may determine if there is a minimum date rate that is acceptable for the application. When conducting this analysis do not forget to take into account the packet framing. There is overhead associated with every WirelessUSB packet (discussed in more detail in section "Payload Size" on page 27). High data rate wireless systems will also probably require the radio to be ON a large portion of the time and may not be able to take advantage of the sleep-wake cycle discussed in section "Battery Powered Devices" on page 26.

Even if the required throughput is low enough to support any data rate, there may still be benefits to higher data rates. The ‘chip rate’ for WirelessUSB is constant at one million chips per second regardless of the actual data rate selected. Two different PN code lengths, three different data encoding schemes, and a GFSK mode all contribute to the range of data rates available. If data is moved at a faster rate, the transmission is completed quicker; thus the radio can be moved from a high power mode to a low power mode more quickly resulting in moving the same amount of data at a lower power consumption.

There are trade offs, of course. Higher data rates are theoretically more subject to interference. This is certainly true of GFSK mode, without the benefits of DSSS to help recover data bits when chip errors occur. For the DSSS modes the performance may not be entirely as expected. The radio is designed around 8DR mode, and users will find that it is most effective at transmitting data in this mode. Within any given encoding scheme (8DR, DDR, and SDR) the 32-chip options are generally less effective than the 64-chip options at delivering data across the channel without the corruption of bits.

2.4.2 Payload Frequency (Report Rate)

Payload rate may only be an option for systems that have a low enough throughput rate that any or most of the data rates are available to them. For example, audio systems that require large amounts of data may find that they must keep the radio in transmit or receive modes constantly. Compare this to building automation systems with sensors that only need to transmit reports at an infrequent interval. And of course, there are systems with requirements that fill the entire spectrum between these two examples.

Systems that sleep between packets for power savings also need to analyze the maximum interval that they can spend asleep in order to meet the throughput requirements. This can also be closely related to latency that is discussed in the section Latency on page 28.

2.4.3 Payload Size

If it is necessary to transmit 20 bytes across the link, is it better to transfer one 20 byte packet, twenty 1 byte packets, or something in between? There are a couple of things to keep in mind when considering the payload size.

The first thing to consider is that there is overhead associated with each packet. Exactly how much varies with a number of the configuration options available with WirelessUSB. Once these options are selected, you can expect the overhead to be consistent regardless of the size of the packet. Therefore, it is generally more efficient to transfer larger packets.

The next thing to consider is the impact of bit error rate (BER). In a given environment with a fixed set of configuration options there is a probability that any given bit may be corrupted when it is transmitted. At the lowest level, is the probability of a DSSS chip being corrupted. The tolerance thresholds manage this to some extent, but with a set number of chips making up each bit you can still calculate a the-

oretical probability of an actual data bit being corrupted. This probability is the same for every bit; therefore, the more bits that are sent the greater the probability that any error occurs in the packet.

WirelessUSB LP/LPstar does employ a CRC mechanism to help detect errors, but there is no a built-in error correction capability. Although error correction could be added in software, there may also be a packet size penalty to pay for that. For the moment, only consider that if an error occurs, it is necessary to retry sending the packet again if accurate delivery of data is essential.

Thus, there is a conflict. Larger packets are more efficient, but they may also be subject to a higher probability of failure and require retransmission, therefore increasing overhead in another way. Finding the critical point where the balance shifts one way or the other may require experimentation under the conditions required for the application.

There is one final consideration when dealing with larger packet sizes. Keep in mind that WirelessUSB LP/LPstar has a 16 byte data buffer. This is significant particularly for battery powered devices that need to minimize power. For packet lengths up to 16 bytes it is possible to sleep the microcontroller after the packet is initiated because the radio's Auto Transaction Sequencer (ATS) can manage the transmission without microcontroller intervention. For packets lager than 16 bytes it is necessary for the microcontroller to be awake at least periodically to manage buffer over- or under-flow conditions. Therefore, it might be beneficial to break large packets into chunks less than or equal to 16 bytes.

2.5 Latency

Latency defines the amount of time between the occurrence of an event and the response to that event. Identifying the event is usually fairly simple; for example, the press of a button. Identifying the end point is a bit more tricky.

Consider a wireless keyboard. From an end user perspective the latency that is most important is from the time a key is pressed until the character is displayed on the screen. The problem is that this is not easy to measure. The engineer cannot hook up an oscilloscope and trigger on when a character appears on the screen. And the handling of the keypress by the operating system cannot be determined. Even if we know when the data was handed to the PC hardware, there is no set delay until it appears on the screen. Therefore, this definition is not useful. A more viable option is to measure when the receiving radio passes the key data to a microcontroller over the SPI interface, or when a USB packet with the data is transmitted to the USB host controller.

The point to be made here is that a useful definition of latency must be set, and then you can analyze the factors that impact it. Some of these may be related to the wireless portion of the system but others may not. The latency related to transmission of a packet over USB is an excellent example. It adds to the overall latency of the system and designers will have to account for it.

From a wireless perspective there are a few things that can effect latency. Payload frequency is probably the most critical. If the system goes to sleep for 1 ms and then wakes up for a brief interval to transmit data, then there is a potential latency of 1 ms due to the wireless link. Data rate also has an impact. Slower rates take longer to transmit the data.

The most obvious trade off here is power consumption. But there are also some other subtle things to consider. For example, the WirelessUSB LP/LPstar radio does not respond the same on all channels. The Synthesizer takes a different amount of time to settle for slow, medium, and fast channels. From both a latency and power consumption standpoint, it might be best to manage channels as oppose to preferentially always using the fastest channels. But this may add to code size and complexity.

2.6 Link Architecture

Link architecture defines the type of network being created. To a large extent this may be defined by the type of product that is being built, and thus might be a non-issue. But in some cases there may be options for how to construct the link. And in either case, understanding the architecture contributes to an understanding of the system requirements and trade offs.

2.6.1 Topology

Topology is the most relevant topic with respect to link architecture. This refers to the physical structure of the link and how devices relate to one another. Some sample topologies include:

One-to-One. An example of this might be an RS232 cable cutter. Two nodes that used to be connected through an RS232 cable are now moved to a wireless link.

Star Topology. This type of network has one or more nodes that communicate with a central device. Cypress's 2-way HID designs and N:1 both fit into this general category.

Mesh. In a mesh network, individual nodes communicate with one another without having to go through any central point.

2.6.2 Number of Devices

It is important to define the number of devices that are allowed on the network. In the start topology examples above, both the 2-way HID and N:1 designs were mentioned. In 2-way HID there is generally one mouse and one keyboard that communicate with one PC-based bridge, although there is some allowance for a 'few' other devices, such as a presenter tool or remote control. In N:1 there are a

large number (up to about 65,000) of sensor or actuator nodes that are linked to one central hub.

These could both be considered star topology, but there are very different protocols put in place to manage the different numbers of devices. Therefore, defining the number of devices can have a drastic impact on the details of the communication protocol and link layer of the system.

With small numbers of devices, collisions may be very infrequent and easily managed, relatively high throughputs may be supported, and latency can be kept small. Large numbers of devices may require substantially different ways of thinking about the link. Significant collision handling and retry mechanisms may be required, time division access or a broadcast mechanism to provide link framing might be instituted. Long latency might have to be required or a priority mechanism for certain nodes or types of data might be needed.

2.6.3 Direction

Direction sounds fairly simple, but it can also touch on some other more complicated definitions.

Most simply, direction defines which way information is transmitted. Most WirelessUSB systems will probably use ACKs to provide notification that data was received by the other end of the system, but this might not always be the case. Even if ACKs are used, meaning that bidirectional communication is taking place, the actual data flow may be only in one direction such as from a sensor to the central hub.

There are also many cases where data predominately flows in one direction but it may be necessary to occasionally transmit in the other direction. Because the data that is transmitted in the ‘backwards’ direction is infrequent, it might be possible to have different, less efficient means of handling it.

Direction of data flow can also be tied to the power consumption and operating modes of the devices. Look at a situation where data is transmitted in only one direction, such as from a sensor to a central hub. If the hub has an unlimited power supply and the sensor is battery powered then the hub can listen constantly, and the sensor can wake up to transmit only when required. Direction and power management are compatible with one another. But what if the hub was also battery powered and could not effectively listen at all times. This can greatly complicate the system architecture. It might be necessary to provide some mechanism for synching the two devices. For example, the hub might send out a broadcast packet at some interval signifying that it is ready to listen. A sensor with data to transmit might have to listen for the broadcast packet before sending its data.

Direction also leads into the concept of ‘who’ communicates ‘when’. One way to look at this is the concept of Master and Slave but this can get into gray areas. Consider the example of a wireless mouse. If we looked at the wired equivalent connected through the USB port, the PC is the master and the mouse is the slave. The mouse only sends data when the USB host first sends an IN token to the device.

But in the case of our wireless mouse, power constrains the system. The bridge on the PC can be listening at all times, but the mouse must conserve power and so it sleeps most of the time and only wakes up periodically to transmit a packet. The mouse is the master of the communication link. It decides when to transmit data, which is the opposite of the USB case.

To confuse matter more, look at the interference avoidance aspects of the system. Since the PC bridge is always in receive mode, it is ideally suited to monitoring the environment, detecting interference, and selecting a new channel when required. Thus the bridge is the master from the perspective of channel selection.

Many vendors use the term ‘transmitter’ and ‘receiver’. This works fairly well for devices that predominantly transmit in only one direction. Again, the wireless mouse is a good example. The data flow is almost always from the mouse to the bridge. The mouse would be defined as the transmitter and the bridge is the receiver. Just bear in mind that when ACKs are transmitted these two devices swap radio modes: the receiver transmits the ACK and the transmitter receives it.

This discussion is primarily semantics, but it highlights a common communication problem that arises around wireless systems. The take away is to define terminology early and try to find a good fit based on the system functionality. As an example, Cypress has used the terms ‘node’ and ‘hub’ for its N:1 designs, which are roughly equivalent to the terms ‘device’ and ‘bridge’ that it uses for its HID designs.

2.7 Interference Avoidance

The ISM band is a busy spectrum with a lot of different technologies sharing the band. Therefore, managing interference is an important consideration. One of the strengths of the Cypress WirelessUSB solutions is their immunity to interference. This is a combination of the radio technology and the algorithms employed in the solutions. System architects must consider what types of environments are expected, and how much effort must be placed on managing interference.

The DSSS technology itself is the first stage of managing interference. Individual chips may be corrupted due to interference, but the fact that a single data bit is encoded with 64 or 32 chips means that it is still possibly to recover the data on the receiving end. The tolerance for how many bits can be lost before rejecting a packet is a parameter that the developer controls. Cypress has defaults that we recommend in our solutions, but this is something that vendors may want to evaluate in the expected environment for their products. This discussion also highlights the fact that devel-

opers must think carefully before using the GFSK mode of the radio since all of these benefits are lost.

In the presence of stronger interference, the DSSS technology by itself may not be enough to overcome interference. Recall that we have already touched on the effects of PA level and data rate in other sections. Increased PA level, and certain data modes can improve the maximum range of a system, which also means they can help to overcome interference at shorter ranges. Using these techniques must be balanced against power consumption, and potentially even throughput and latency.

Up to this point we have only discussed mechanisms for overcoming interference. But interference can also be avoided entirely. There are two stages here. The first is detecting the interference, which can be accomplished with different methods. WirelessUSB has the capability to measure signal strength, which can be used to monitor the background noise level. Other techniques, such as monitoring packet error rate, can also be employed. Once interference has been detected, the system must then identify a channel with an acceptable level of interference, and then move all nodes on the system to this new channel. The method of communicating the move to all nodes in the system can be somewhat complicated. System architects will have to consider their options, such as attempts to actively communicate the channel change to other nodes, versus a passive mechanism where a master station moves and it is up to other nodes to seek out the master on its new channel.

2.8 Co-Location

Co-location refers to the number of systems that can operate in close proximity to one another. ‘Close’ means within communication range. They key is that from a co-location perspective you usually do not want the systems to be able to exchange data with one another.

A classic example is wireless keyboard and mouse systems in an office environment. There may be dozens of systems within about 10 meters of one another–well within the range at which they can communicate. The requirement is for each keyboard and mouse to be able to exchange data with its own PC, but they should not send data to any other PCs in the office. It would be very problematic if one person typed on his keyboard but the characters showed up on the screen of another employee a few desks away.

In addition to preventing the separate systems from exchanging data with one another, it is also important that the systems not interfere with one another, causing loss of data or excessive retries. Managing co-location can be achieved in a variety of ways. For WirelessUSB LP/LPstar systems the primary mechanisms are to distinguish systems based on channel, PN code, and CRC seed. There are trade offs associated with each of these parameters. These topics were also discussed earlier in Simple Network Parameters on page 15.

Designers have to determine how many systems may need to be co-located together. This has to be weighed against the interference environment, and possibly even other considerations, such as throughput. If only a few systems ever need to operate close together, then separation by channel may be sufficient. But channel selection is also part of the interference avoidance technique. Therefore, if there are more systems, PN code and/or CRC may need to be used as well. If the throughput requirements are high, then designers also have to consider the effects of two systems sharing a channel, but on different PN codes or CRC seeds. Systems on different PN codes may still see each others transmissions as increased noise. And systems that are only differentiated by CRC seed actually have to share bandwidth; they will receive each other’s data and reject it only after evaluating the CRC seed.

2.9 Transfer Type

Transfer Type refers to the type of data and carries an assumption of how it will be handled within the system. A close analogy to this would be USB data transfer types, such as bulk and isochronous data. In USB, bulk data has a guaranteed delivery but not necessarily a guaranteed timing. If a transfer fails it will be retried until it succeeds. It also receives a lower priority compared to other types of data and so may not be transmitted across the link as soon as it is available.

In contrast isochronous data is time sensitive. It is guaranteed a certain amount of bandwidth in the link and is transmitted with a high priority. But it also does not have guaranteed delivery. If an isochronous packet fails, USB does not retry it and the data is permanently lost.

Similar concepts may also have to be applied to wireless systems. System designers must understand how different types of data will be used in the system and determine the relative importance of things like guaranteed delivery, priority, and timing. Answers to these questions will have significant impact to the construction of the protocol that will be used to manage the link.

2.10 Binding

The industry uses various terms for the concept of binding such as pairing, connecting, and others. These all describe the mechanism by which a network is created or by which new devices are added into an existing network. This can be a very complex topic, especially for consumer devices where the end user must clearly understand what he is required to do to set up the system. For commercial or industrial systems where the network may be configured by a technician, it might be less of an issue.

Cypress has a separate application note, *Wireless Binding Methodologies*, that discusses this topic in more detail so it will only be reviewed at a high level here. Binding can be

achieved in many different ways so the system architect must think very carefully about the user experience that is required during the process, under what conditions it can occur, and how resistant it is to failure.

Examples of binding options include:

- Binding in the manufacturing process, such as through use of special test software/hardware
- Binding that occurs as a result of a user action, such as pressing buttons
- Binding that occurs upon the power up sequence of a device
- Binding that can occur in a dynamic or ad-hoc basis, such as whenever a device comes in proximity of a network
- Completely manual binding, such as selection of the channel through a switch, or entering parameters via a software interface

There are many trade offs that have to be considered so this topic should be dealt with early so that impacts on the rest of the system can be fully understood.

2.11 Footprint

Footprint is relatively simple to discuss. It simply refers to how large the physical solution is. A good example of where this is important is a wireless presenter tool which includes a USB dongle for the connection to the PC. Because the tool is made to be transported around, the dongle is made to slide into a small compartment in the presenter tool itself. Therefore, form factor is a critical requirement.

One of the main advantages of the PRoC LP device over the WirelessUSB device is form factor. PRoC LP/LPstar includes the microcontroller function in the same physical package that is used for just the radio in WirelessUSB LP/LPstar. Therefore designers can save the space required by the microcontroller.

There are other options for reducing size, but they do have their cost. Smaller components can be chosen for things like the crystal and discrete components for the PMU, matching network, and filters. These smaller components may have a higher cost, or may have performance limitations compared to larger devices. The design of the antenna is also a factor, particularly for PCB trace antennas that are typically used in small form factor designs. These small antennas may have performance limitations compared to other options.

2.12 Cost

Cost is almost a trivial topic by the time that all of the preceding items have been discussed. Everything has a cost associated with it, and obviously marketing and system designers must consider these costs to ensure they have a viable product with the right set of capabilities for the price point. Costs can occur up front in the design of the product or set up of manufacturing, or they can be born over the life of the product due to the bill-of-materials (BOM) and production costs. In some cases these might even be traded against one another. They are certainly traded against almost every other topic in this chapter.

2.13 Time-to-Market

Time-to-market is another fairly straight forward concept, even though it may be difficult to minimize in practice. It can have vast implications for product success and may require hard decisions for the developers. One of the most important factors in achieving the desired time to market is going through the topics in this chapter, thinking about them thoroughly, and then making and documenting appropriate decisions based on the requirements of the product. At the very least this will help set reasonable expectations for all members involved on the product development team.

2.14 Manufacturability

This is not a topic that will be addressed in detail, but it is still worthy of mention. Wireless designs, especially those pushing the performance limits, may face manufacturing challenges. This can especially be the case when manufacturing is intended for low cost facilities where the equipment available may not exactly be cutting edge. This will probably be one of the later topics looked at during the product analysis and specification, but it should still be examined before the design work starts. Many of the preceding questions will need at least preliminary answers to determine if there are any manufacturability concerns. Any manufacturing partners should also be consulted to understand their capabilities and limitations.

2.15 Testing

Testing is a topic that is not given its fair mind share far to often. It is often critical to overall success of the product. There are a few different aspects to consider, but all of them must be considered before the design work starts. The system must be designed for test from the start, because it is usually difficult to go back and add test capabilities after the fact.

2.15.1 System Quality Assurance

This is basically acceptance testing of the product once the design work is complete. The bulk of it is focused around building test cases that fully exercise the product's capabilities. Often special test modes might be required in the device to make sure that corner cases are being addressed, or to manage all of the test requirements efficiently.

2.15.2 Regulatory

RF products will have to pass regulatory test requirements such as FCC, ETSI, or TELEC, depending upon the regions in which they will be sold. These test will all require that the product be put into certain operating modes. As a result these must be considered at the start so that the product includes firmware, or controls, to enter the test modes.

There may be non-RF tests that fall into this general category as well, so designers must consider all testing that might be required. Examples include USB compliance, Windows compatibility testing, electromagnetic interference (EMI), electrostatic discharge (ESD), or electrostatic fast transient burst (EFTB) testing.

2.15.3 Manufacturing

Wired products can often rely on functional tests to thoroughly evaluate that the device is ready for shipment. Although functional tests are necessary for wireless products, it is usually a good idea to also include RF testing. RF testing is typically used to evaluate that there are no manufacturing defects that would impact the range or RF performance of the device. This type of testing is typically accomplished using a shielded chamber. The signal path is intentionally attenuated to simulate longer range conditions. Then packet or bit error rates are measured during data transfer to evaluate if the system meets the desired requirements. The Cypress CY3631 Manufacturing Test Kit provides an excellent reference for adding RF testing to the production line

3. Resets

3.1 Introduction

When designing for the WirelessUSB LP/LPstar or PRoC LP/LPstar, there are three types of radio resets to consider; each is subsequently discussed.

- Power On Reset (POR)
  - POR via capacitor
  - POR via microprocessor
- Hard Reset
- Soft Reset

When power is first applied to the radio, it typically takes a finite period to achieve the designed operating voltage range. During this time, the microcontroller may be required to maintain a specific state on its output pins and hold off any execution of register initialization code until the radio is stable.

The reset (RST) pin is an input that resets the entire chip when power is applied to the radio. The RST pin must be brought high momentarily upon power up. This pin can also be used during a system-wide reset event. When the RST pin is high, the clock will not run. The MODE_OVERRIDE_ADR.RST bit only resets the registers to their default values; it does not reset the entire chip.

A simple function such as reset can cause many problems since different applications impose very different conditions on the start up and power down of the radio. This chapter covers the main types of resets and aims to lead the user to a proven reset strategy by providing straightforward recommendations. Reset in its most basic form ensures that the radio functions in a controlled manner during normal device operation.

The parameters affecting reset are explained in detail in this chapter. In most applications several factors apply and so it may be a combination of these elements of reset functionality that must be considered to establish consistent and reliable radio communications.

3.2 Power On Reset

The radio must sense a POR event when power is applied to the radio otherwise the state of the radio control registers is unknown. The POR event is a RST pulse (V_{RST}) that must rise above V_{IH} ($0.7*V_{IO}$) and must hold that level until the V_{CC}, V_{REG}(for LP and PRoC LP only), and V_{DD} pins are greater than the minimum V_{BAT} which makes sure that all of the logic across the supply boundaries have sufficient supply voltage to function properly as logic. RST may be held high longer.

If this event does not occur (or is not successful), you may end up with some devices that turn on in a mode that draws supply current or otherwise behaves in some undesirable way. RST can be high before power is applied, or it can ramp up together with V_{BAT}. This is what an external capacitor does automatically if sized correctly relative to the power up ramp rate.

RST must remain low for the remainder of the operating session. Performing an initial POR event is the only way to guarantee consistent, repeatable start up behavior part-to-part and run-to-run. Those radios that do not function correctly after an incorrect POR event will not be recovered by strobing the RST pin at a later time. Therefore, a correct POR event is the only way to ensure proper functionality of the radio.

Upon completion of a POR event, the internal configuration registers revert to their default states. These values may be found in the device data sheet, see Cypress Semiconductor Support on page 9. Several registers must be changed or controlled during normal radio operation at other than their default settings. After a POR event, initialize radio registers via SPI communications to the radio interface.

SPI access is available during and after the POR event. The registers are accessible even during reset. However, writes have no effect until the RST pin is below the V_{IL} threshold. It is recommend that initialization firmware poll a radio register, XACT_CFG_ADDR at address 0x0F, by writing a non-default value of 82h to it while reading it back; when it successfully reads the contents of the register, it is an indication that it is safe to begin radio register initialization.

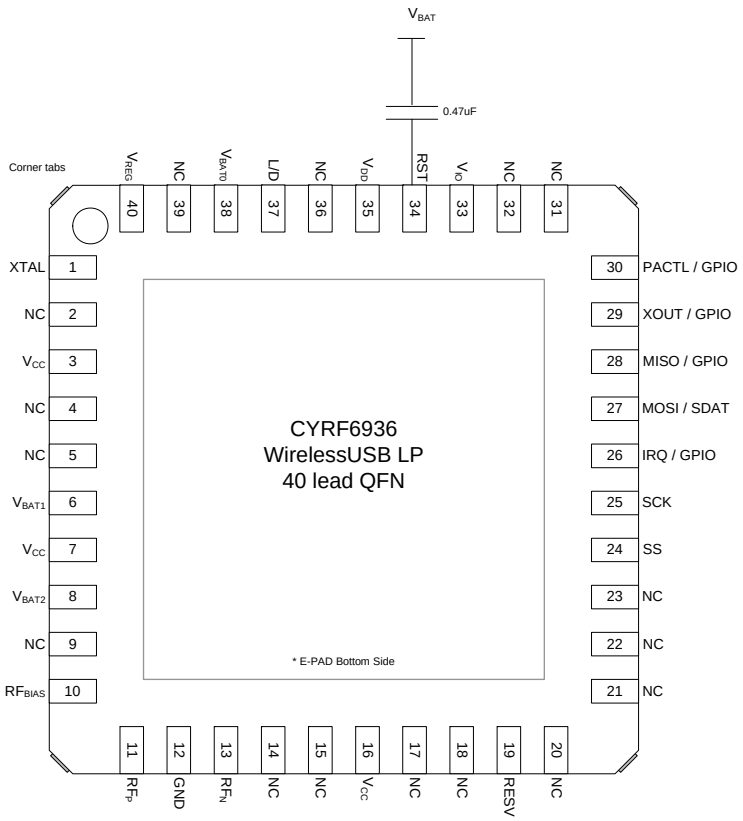
3.2.1 POR via Capacitor

The recommended method to ensure the POR event is successful is by placing an external capacitor (for example, 0.47 µF) tied to V_{BAT} and internal pull down resistor (10K ohm) with V_{BAT} settling within 2 ms. It is recommended that the external capacitor is X5R (for example, Kemet C0402C474K9PACTU) which has a ±15% maximum capac-

itance shift over -55C to +85C, see Figure 3-1 on page 34. For LPstar, place the same capacitor to V_{BAT} as the recommended POR circuit of LP.

If an external capacitor is used to provide the POR event, the value of the capacitor must be selected and V_{BAT} ramp controlled such that V_{RST} is met under all conditions. For example, using the recommended external capacitor value when V_{BAT} settles within 2 ms generates V_{RST} of at least 1.26V (that is, 0.7*V_{IO}) when V_{IO} is 1.8V. In this example, the POR event to RST low (V_{IL}) event takes approximately 10 ms. This is a typical time and will vary with capacitor tolerance, process, temperature, and voltage.

Figure 3-1. Recommended Power On Reset Circuit of LP

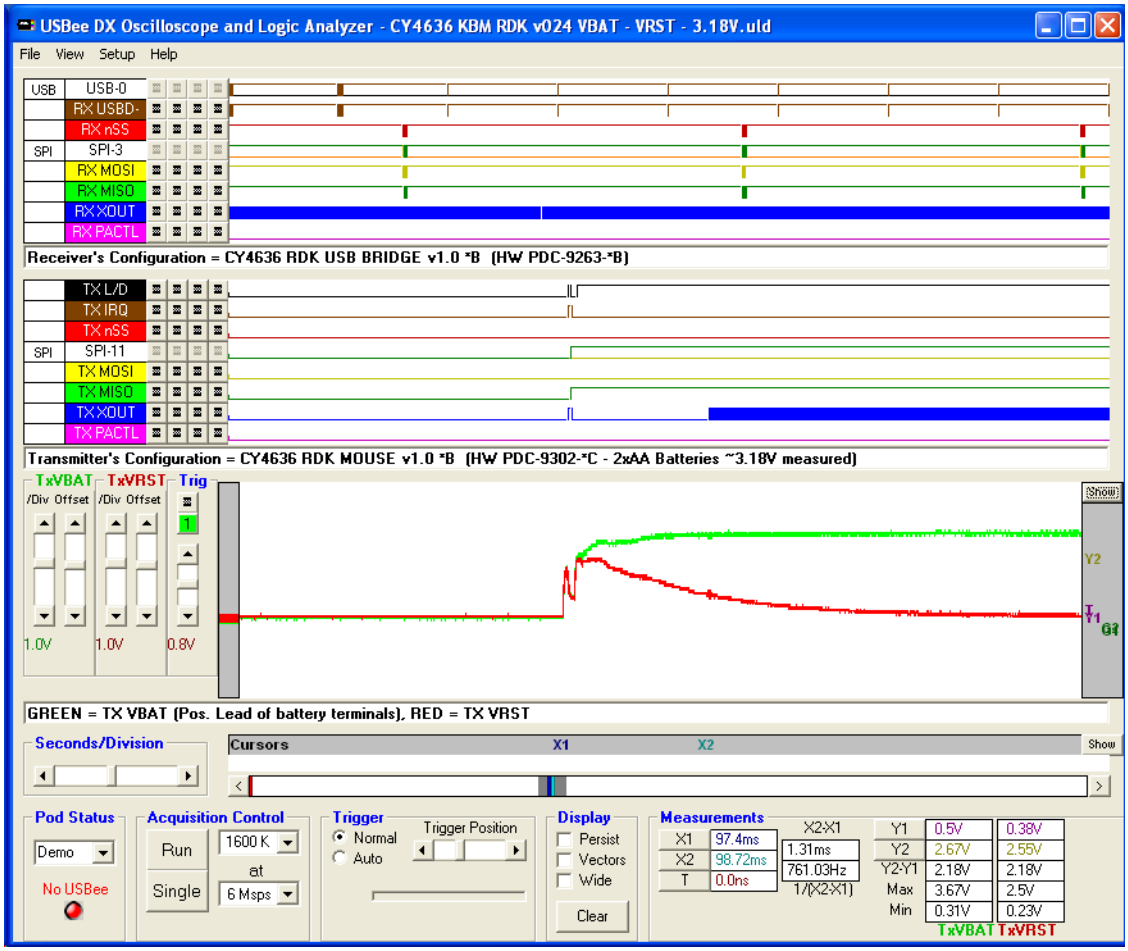


A note about reliability:

In high-volume manufacturing, attention must be paid to all aspects of reliability. Manufacturing a 0.47 µF, 0402 capacitor involves very thin dielectric construction. To achieve maximum reliability, the capacitor must be handled and soldered in accordance with the manufacturers instructions. In particular, be sure to observe the max. temperature during IR reflow, and proper temperature ramp-up and ramp-down. Failure to comply could result in intermittent capacitor cracks, resulting in overall product failure.

Where space allows, use of larger 0603 or 0805 capacitor packages helps alleviate the capacitor reliability problem. Also, make sure the chosen capacitor has ample voltage rating. A voltage derating factor of at least 2x is recommended for best reliability.

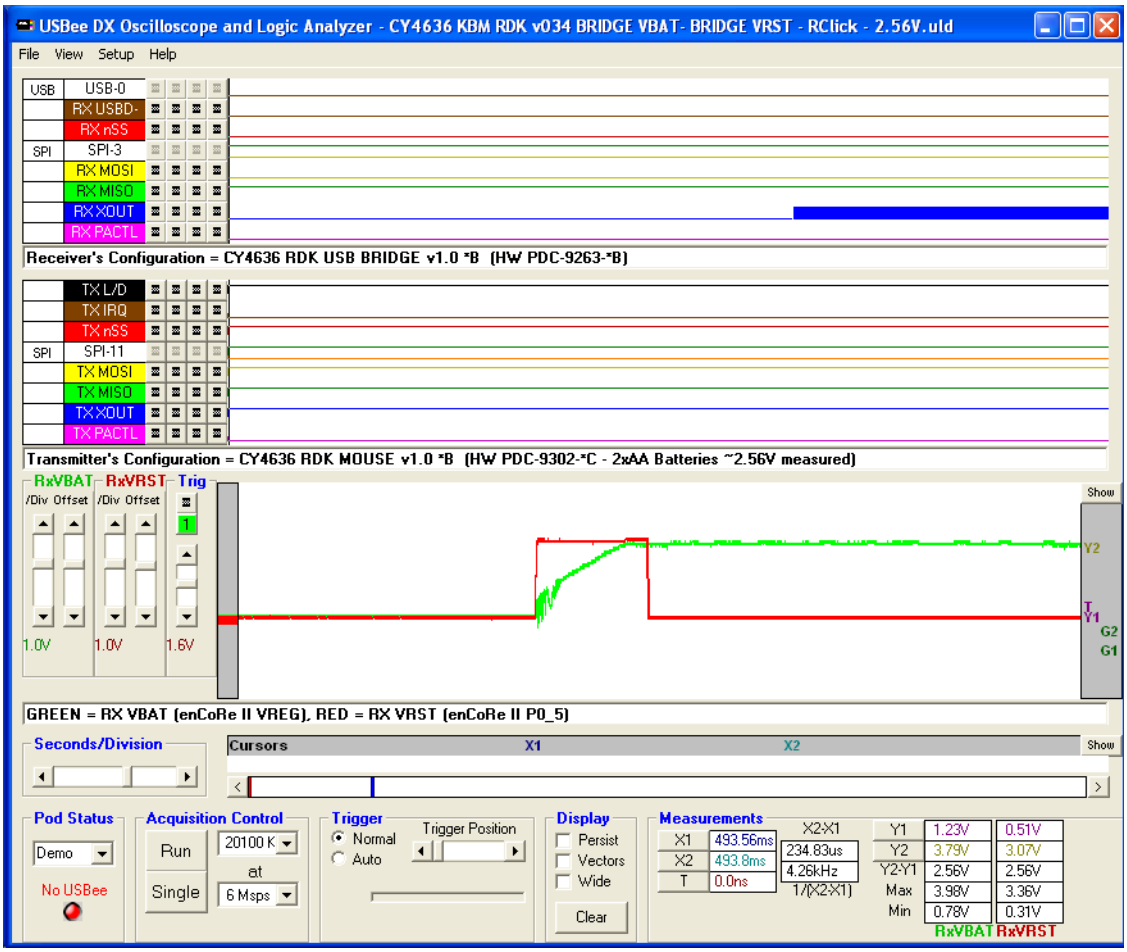
Figure 3-2. CY4636 RDK Mouse v1.0 *B (PDC-9302-*C - 2xAA Batteries ~3.18V measured)



3.2.2 POR via Microcontroller

The user may choose to use a digital source instead of placing the capacitor for the POR event. On initial connection to a power source, the external reset source must behave correctly to reset the radio in a similar fashion as having the RST capacitor in place, otherwise correct startup behavior and proper device functionality cannot be guaranteed. There are some limitation to using a digital source for the POR event. First, XOUT does not run during reset, so the microcontroller cannot be solely clocked from XOUT. Secondly, the default state of the RST pin must be high.

Figure 3-3. CY4636 RDK USB BRIDGE v1.0 *B (PDC-9263-*B)



3.3 Hard Reset

Although not necessary, the microcontroller can reset the radio by driving the RST pin high and waiting long enough to ensure that the RST pin voltage has exceeded 0.7 times the VIO pin voltage. Then the microcontroller places the pin in a high impedance state to allow the RST pin's internal 10K ohm pull down resistor to bring the RST pin voltage back down. Since the 10K ohm resistor and 0.47 µF capacitor, form an R-C network, the voltage drop is not instantaneous. After setting the GPIO pin in the high impedance state, the microcontroller should use the register write until read successful polling method mentioned above to determine when it is safe to begin radio initialization.

The data sheet (see Cypress Semiconductor Support on page 9) states that the voltage on any input pin may not exceed the voltage applied to the V_{IO} pin (max V_{IO} is 3.6 V). However, higher voltages may drive the pins through series resistors. The series resistors must limit current to less than 1 mA. Existing designs have done this using either 1K ohm or 2.2K ohm resistors.

3.4 Soft Reset

As mentioned before, a hard microcontroller reset is not necessary. After POR, the microcontroller can execute a radio reset by setting the MODE_OVERRIDE_ADR.RST bit. This is the method recommended to reset the microcontroller register and it should be part of the radio initialization firmware. This does not satisfy the POR event requirements.

3.5 POR Event Power Cycling

Users can briefly power cycle the radio by uninstalling and then quickly reinstalling batteries in `keyboard/mouse/etc applications`.

In the above example, it is feasible that V_{BAT} may not fall below V_{IL} which could in turn cause V_{RST} to not rise above V_{IH} when the wrong size of RST capacitor is used.

Properly resizing the external RST capacitor (for example, 0.47 µF) value increases the probability that V_{RST} rises above V_{IH} during V_{BAT} ramp by increasing V_{RST} time constant for all conditions. This assumes that V_{BAT} ramps well within the V_{RST} time constant (less than 2 ms is recommended).

4. Interrupts

4.1 Introduction

The radio provides an interrupt (IRQ) output that is configurable to indicate the occurrence of different events. Using an IRQ in a design eliminates latency caused by using a polling loop technique. The radio features three sets of interrupts: transmit, receive, and system interrupts. These interrupts all share a single pin (IRQ), but can be independently enabled or disabled. The contents of the enable registers are preserved when switching between transmit and receive modes. This IRQ pin may be used by other device interrupts in a system. These different interrupts must be appropriately handled by an interrupt service routine appropriately.

4.2 Physical Layer

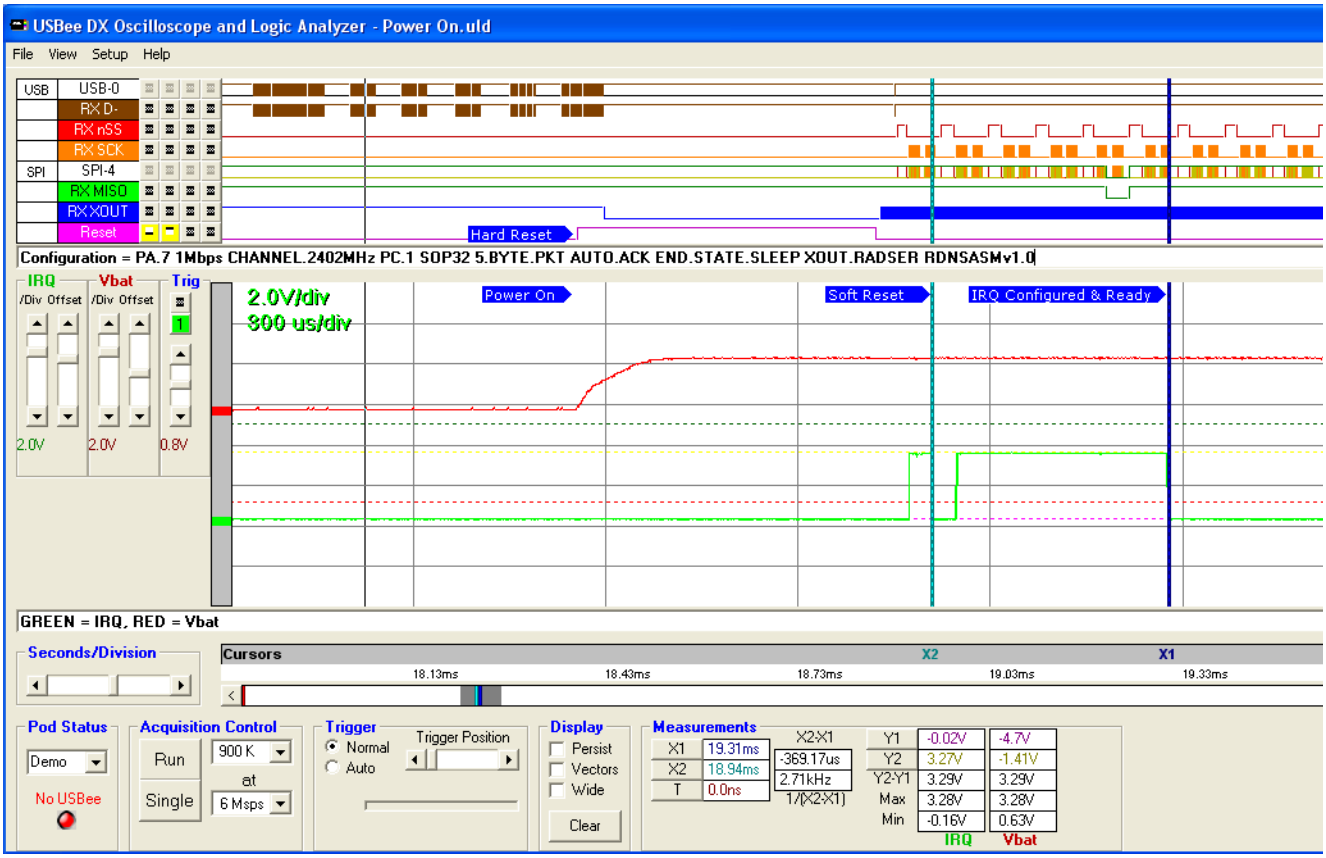
The IRQ pin (26) can be configured as an open drain output or a standard CMOS output in register IO_CFG_ADR(0xD) bit 7 (IRQ_OD). Clearing the IRQ_OD bit configures the IRQ pin as a CMOS output, with the output '1' drive voltage being equal to V_{IO} pin voltage. If higher voltages are desired, than the IRQ_OD bit must be set and an external pull up is needed. The polarity of the pin is also configurable by setting or clearing IRQ_POL (bit 6) in the same register set.

The IRQ pin can also be used as a GPIO pin by setting IRQ_GPIO. When this bit is set, the IRQ function is multiplexed onto the MOSI pin. In this case the IRQ signal state is presented on the MOSI pin whenever the $\overline{SS}$ signal is inactive.

Note When using a CYRF69103/303 (PRoC), the user must route the IRQ pin (27) to one of its GPIO pins.

4.3 IRQ During POR

Figure 4-1. IRQ During POR (LP for Example)



When a “Power On” reset event occurs, the device is held in “hard reset” via the reset pin (34) until V_{BAT} reaches its V_{MIN} range. At this point, the reset pin deasserts and the system is stable. The IRQ remains asserted after reset until the first SPI clock cycle clears it. The IRQ pin is asserted again due to a “soft reset” event that is issued to the device in order to synchronize the internal SPI clock and is deasserted by the next SPI clock cycle. IRQ remains deasserted but the polarity is changed to active high at the “IRQ Configured and Ready” event by writing a value of 0x40 to the IO_CFG_ADR (0x0D).

4.4 Transmit IRQs

- TX_IRQ_STATUS_ADR (0x04)

4.5 Receiver IRQs

- RX_IRQ_STATUS_ADR (0x07)

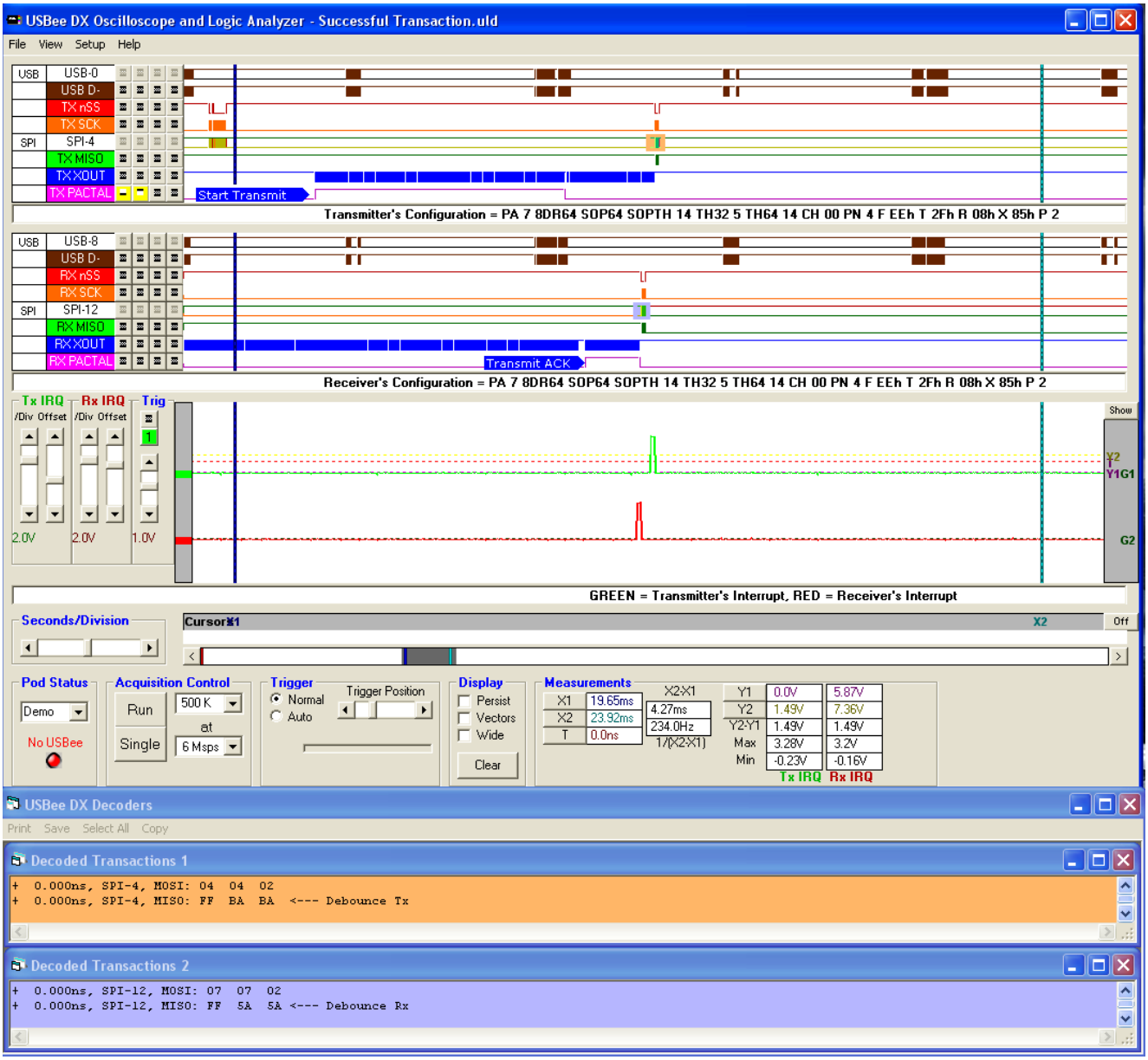
4.6 Debouncing Non-Atomic Radio Status Bits

Debouncing is needed when reading register 0x4 (TX_IRQ_STATUS_ADR) and register 0x7 (RX_IRQ_STATUS_ADR). If the first read of this returns RXC IRQ = 1 and RXC ERR = 0, then firmware must execute a second read to this register to determine if an error occurred by examining the status of RXE ERR. This is possible because this register can be written by the baseband asynchronously, so a complete event might occur first and then at a later time an error is detected and the bit is set. If the first read of this register returns RXC IRQ = 1 and RXE IRQ = 1, then the firmware must not execute a second read of this register for a given transaction. Debouncing is very important to ensure that the register is read correctly and the baseband did not set the complete bit before setting the error bit.

4.7 IRQ Events

4.7.1 Successful Transactions

Figure 4-2. Successful Transactions (LP for Example)



Successful transaction is when the transmitter (TX) and receiver (RX) complete the transaction without any errors. The status of the transaction can be determined by register 0x4 and 0x7, TX_IRQ_STATUS_ADR and RX_IRQ_STATUS_ADR, respectively. The state of all IRQ status bits in these two registers is valid regardless of whether or not the IRQ is enabled in register 0x5 (RX_CTRL_ADR) and 0x2 (TX_CTRL_ADR). The IRQ output of the device is in its active state whenever one or more bits in these registers is set and the corresponding IRQ enable bit is also set. Status bits are no-atomic (different flags may change value at different times in response to a single event).

The values in these registers are as follows: TX_IRQ_STATUS_ADR = 0xBA and RX_IRQ_STATUS_ADR = 0x5A.

TX_CTRL_ADR = 0xBA: First three bits [2:0], transmit is complete with no errors. Bit 1 is set signifying the IRQ has triggered when transmission is complete. Reading this register clears this bit. TXC IRQ and TXE IRQ flags may change value at different times in response to a single event. Due to the fact that these bits can be set at different times, it is necessary to debounce the register, as explained in ?$paratext>? on page 41. This case is shown Figure 4-2 and is marked as “Debounce.” Second three bits [5:3] indicate buffer status. Since all three bits are set in this capture, the buffer is empty. If an error occurred and all bytes were not transferred, then these three bits would contain the status. Bits [7:6] indicate system is stable and powered.

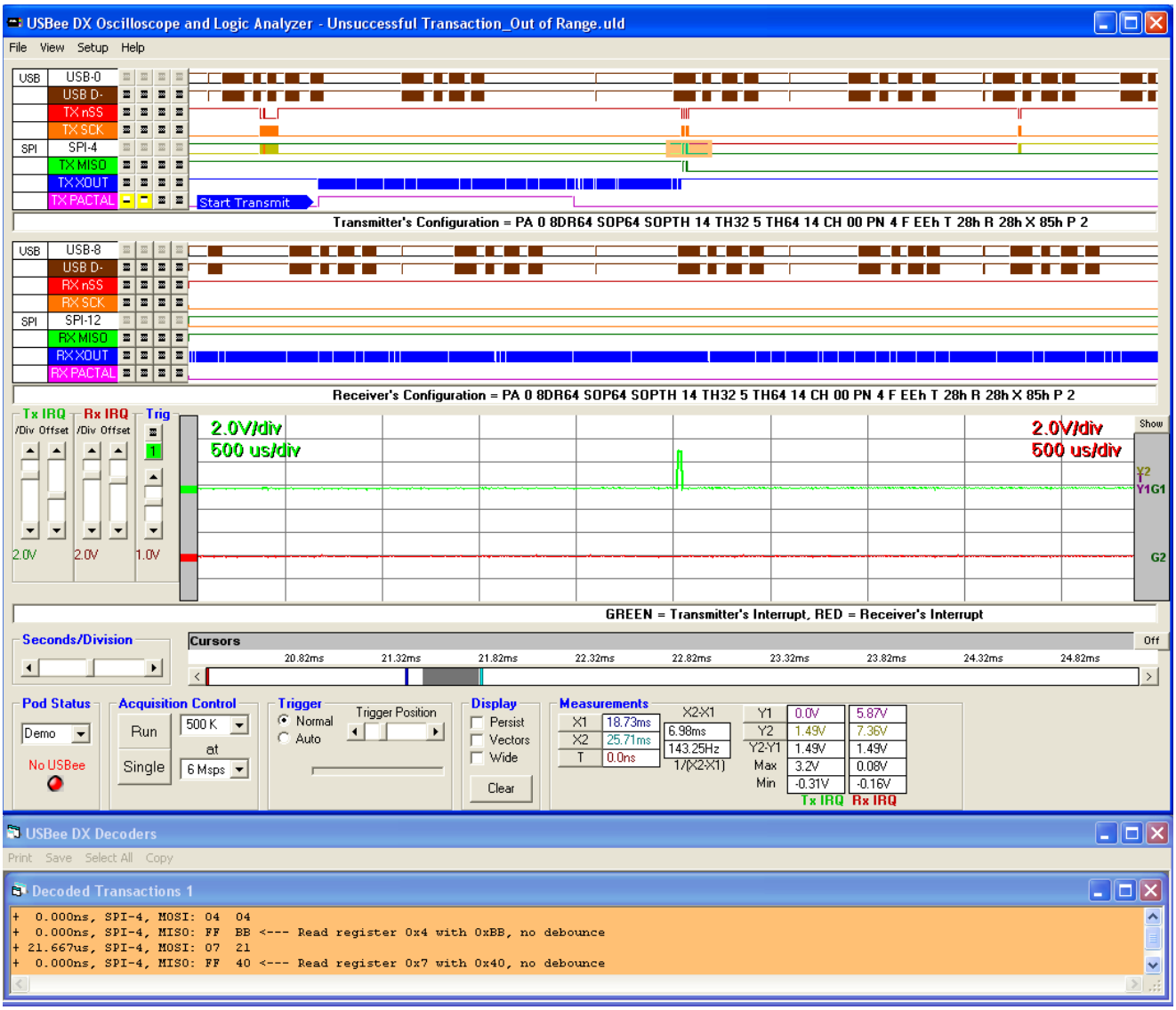
The TX IRQ fires once the packet is completely transmitted and the ACK is received from the receiver. The IRQ is cleared 22.83 µs when the status is read.

RX_CTRL_ADR=0x5A: First three bits [2:0], receive is complete with no errors. Bit 1 is set signifying the IRQ has triggered when receive is complete. Reading this register clears this bit. This register is also debounced as shown in Figure 4-2 due to the fact that these bits can be set at different times. Second three bits [5:3] indicate buffer status, since it is 011 that means 8 - 15 bytes were received. Bits [7:6] signify that start of packet was detected and received buffer was not overwritten.

RX IRQ fires after the ACK is sent back to the transmitter and is cleared once the status is read from register 0x7 (RX_IRQ_STATUS_ADR) after 22.33 µs.

4.7.2 Unsuccessful Transaction: Out of Range

Figure 4-3. Unsuccessful Transaction: Out of Range



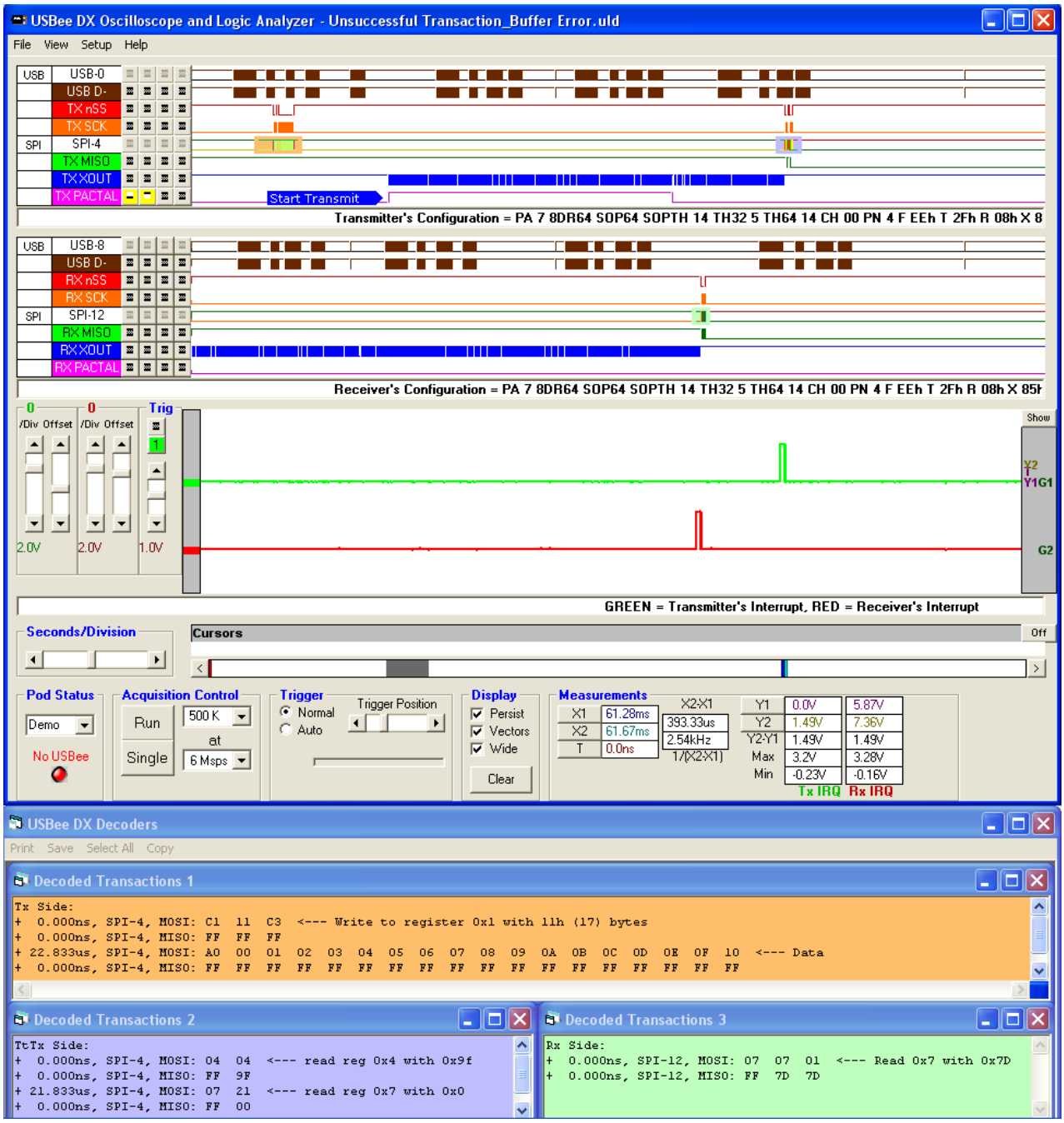
An unsuccessful transaction, as shown in Figure 4-3, shows a failing transaction because the receiver was out of range.

TX_IRQ_STATUS_ADR = 0xBB: First three bits [2:0] show transmit was complete but with error and no debounce occurred. Bits [5:3] show 111 meaning the buffer is empty and transmit has completed. The IRQ this time fires because there is an error in transmitting and the ACK was never received from the receiver. The IRQ is held for 22.67 µs and then cleared by reading the TX_IRQ_STATUS_ADR register.

RX side has no activity because the receiver was out of range. The receiver settings can be tuned to receive subsequent packets, by tuning LNA and ATT.

4.7.3 Unsuccessful Transaction: Buffer Error Due to Transmitting > 16 Bytes

Figure 4-4. Unsuccessful Transaction: Buffer Error Due to Transmitting > 16 Bytes(LP for Example)



The buffer size for the CYRF6936/6986 radio is 16 bytes. An error occurs when more than 16 bytes of data is transferred. Since the buffer size is 16 bytes, all data should be kept at or below 16 bytes.

TX_IRQ_STATUS_ADR = 0x9F: First three bits [2:0] indicate that there was a buffer error and the transmit is complete. Bit 2 being set signifies that TX_BUFFER_ADR is empty and the number of bytes remaining to be transmitted is greater than zero. This is also evident in bits [5:3] indicating 011, meaning more than a 16 byte transfer was attempted.

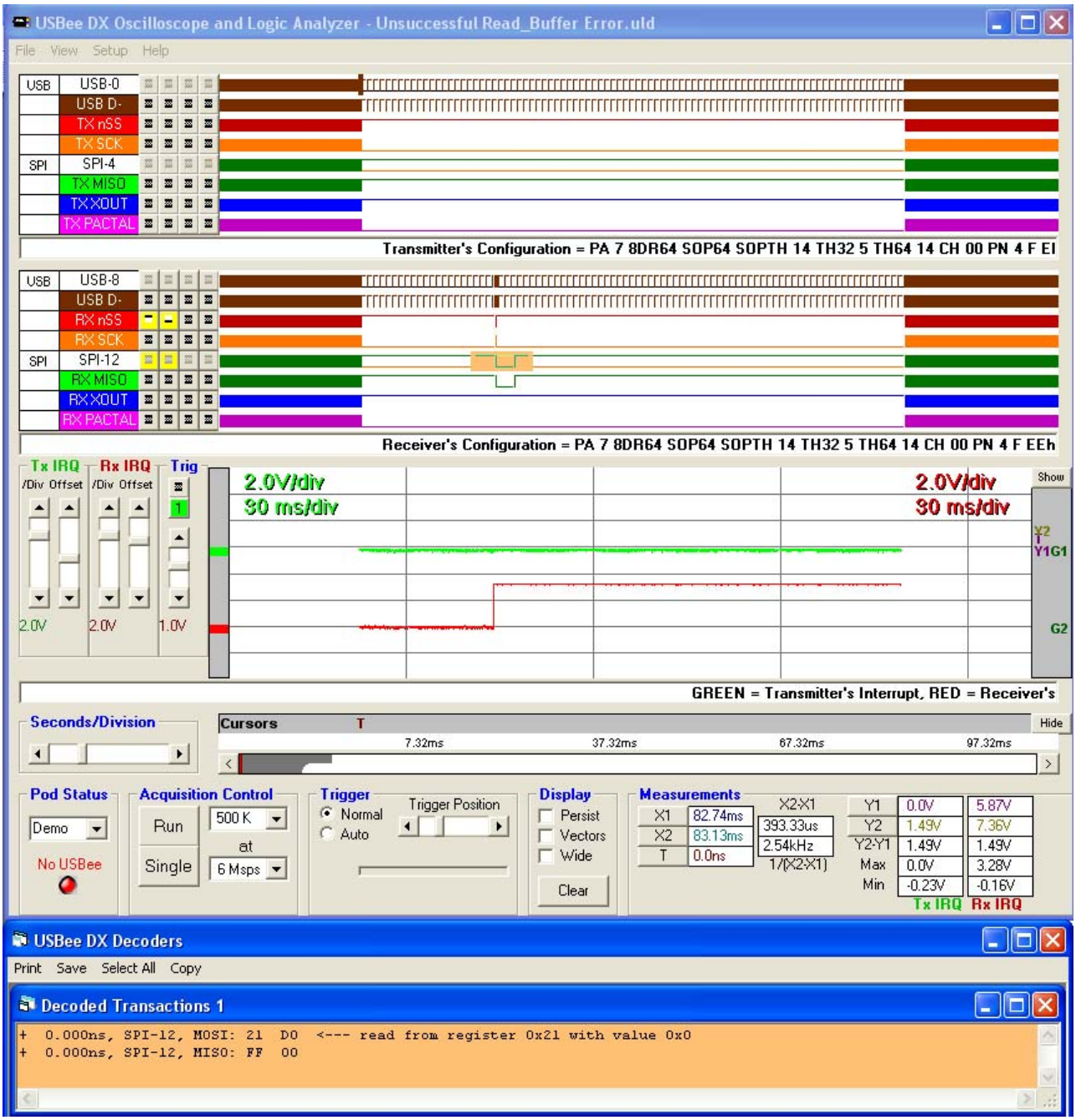
TX IRQ is shown in Figure 4-4. The IRQ fires because there is an error in transmit and no ACK is received from the receiver. This IRQ is cleared by reading the TX_IRQ_STATUS_ADR register.

RX_IRQ_STATUS_ADR = 0x7D: The first three bits [2:0] indicate that there was a buffer error and the receive did not complete. Examining this register further, bits [5:3] indicate the buffer is full. The buffer error interrupt bit is set because the receiver buffer is full and more data was received. This bit is cleared when RX GO is set and an SOP is received.

RX IRQ is shown in Figure 4-4 as well. This IRQ fires because 16 bytes of data were received but there was a buffer error that caused the receive to not fully complete.

4.7.4 Unsuccessful Read: Buffer Error - Reading From an Empty RX Buffer

Figure 4-5. Unsuccessful Read: Buffer Error - Reading From an Empty RX Buffer(LP for Example)



An unsuccessful read event can occur by attempting to read data when the receive buffer is empty. Looking at the registers reveals the following:

RX_IRQ_STATUS_REG = 0x14: First three bits [2:0] reveal a buffer error and transmit is not complete. This is apparent in Figure 4-5. The FW does not clear this IRQ status bit, therefore, the IRQ is held high. Bits [5:3] show that the buffer is empty and bits [7:6] indicate no start of packet was detected.

IRQ, in this case, was fired because the RXBERR IRQEN bit was set in register RX_CTRL_ADR. Anytime a buffer error occurs and this bit is set, the IRQ line asserts.

4.8 IRQ Sources

4.8.1 Transmit

4.8.1.1 TXE

4.8.1.2 TXC

4.8.1.3 TXBERR

4.8.1.4 TXB0

4.8.1.5 TXB8

4.8.1.6 TXB15

4.8.1.7 TX CLR

4.8.1.8 TX GO

4.8.2 Receive

4.8.2.1 RXE

4.8.2.2 RXC

4.8.2.3 RXBERR

4.8.2.4 RXB1

4.8.2.5 RXB8

4.8.2.6 RXB16

4.8.2.7 RX GO

4.8.2.8 RXOW

4.8.3 Power

4.8.3.1 LVI(For WirelessUSB LP and PRoC LP only)

4.8.4 Oscillator

4.8.4.1 Stable

4.9 Status

4.9.1 Transmit

4.9.2 Receive

4.9.2.1 SOPDET

The SOPDET bug could be worked around independently of RXB1. The SOPDET bug could be avoided by using the RXB1, RXB8, etc., to imply that an SOP was detected (this would not work on a zero-byte packet).

4.9.2.2 RXB1

The RXB1 bug could be worked around by reading the RX_COUNT register after RXC/RXE and reading out the remaining bytes left to be read. There are other workarounds as well.

4.10 Muxing IRQ Signal onto MOSI

5. Crystal

5.1 Introduction

A properly designed WirelessUSB LP/LPstar system can easily operate within a ten meter range. Carefully designed WirelessUSB LP/LPstar systems can operate beyond this ten meter range. There are many system design parameters that can affect the range of your system. One of the most important of these is a properly designed clock source.

5.2 Guidelines

This section discusses the following topics:

- Clock and its frequency
- PPM method of calculation
- Load capacitance
- Pullability and trim sensitivity
- Crystal choice considerations
- Clock frequency measurements
- Crystal layout

5.2.1 Clock

A good stable clock and its frequency are two of the most important parts of a wireless system. If the radios in a wireless system are not operating on the same frequency they cannot communicate with one another.

5.2.1.1 Clock Requirements

WirelessUSB LP/LPstar requires a clock frequency of 12 MHz. The output RF frequency ranges from 2.400 GHz to 2.480 GHz in 1 MHz increments (each channel is 1 MHz apart). This output frequency is produced by using the input clock as frequency reference for a VCO and PLL. The accuracy and stability of the input clock is dependent upon the external crystal circuitry.

Table 5-1. WirelessUSB LP/LPstar Crystal Requirements

Parameter	Value
Nominal Frequency	12 MHz
Operating Mode	Fundamental Mode
Resonance Mode	Parallel
Frequency Initial Stability	±30 ppm
Series Resistance	≤60 ohms
Load Capacitance	10 pF
Drive Level	100 µW

5.2.1.2 Clock Frequency

WirelessUSB LP/LPstar is designed to run with an input clock frequency of 12 MHz. An input clock running at 5416 Hz off of 12 MHz produces an output RF that is off by 1 MHz, or one channel. WirelessUSB operates in systems with both radio input clocks at 12005416 Hz, but its RF frequency is 1 MHz higher than expected.

5.2.1.3 Clock Accuracy

As noted in Clock Frequency, WirelessUSB LP/LPstar can operate with its input clock 5416 Hz off of 12 MHz. If it tries to talk to another WirelessUSB LP/LPstar that has an input clock of 12 MHz, and the two radios cannot communicate, it is possible that the two radios are on two adjacent channels. Even if the input clocks are 2700 Hz (225 PPM) apart they are effectively using two different channels. WirelessUSB LP/LPstar needs a more accurate input clock to effectively communicate. WirelessUSB LP/LPstar needs to be paired with another WirelessUSB LP/LPstar that has an input clock within ±30 PPM of its own frequency for effective communication. A system using WirelessUSB LP/LPstar operates with higher PPM difference, but performance decreases in both range and interference immunity.

5.2.2 PPM

PPM is the abbreviation for ‘parts per million,’ a method of calculation used to specify the permissible frequency deviation of a crystal or oscillator. One PPM on a 12 MHz clock is 12 Hz and 10 PPM is 120 Hz.

Total PPM clock accuracy is the sum of base PPM, temperature PPM, aging PPM, and trim sensitivity PPM. Total PPM for a WirelessUSB LP/LPstar system should be less than ±30 PPM.

5.2.2.1 Base PPM

The base PPM is also known as the frequency tolerance of the crystal being used. Frequency tolerance is the allowable deviation from nominal frequency. Tolerance is usually specified in ± PPM, at +25° C and a specific load capacitance. Typical tolerances are from ± 10 to 50 PPM.

5.2.2.2 Temperature PPM

Temperature PPM is also known as frequency stability. Frequency stability is the allowable deviation, in parts per million (PPM), over a specified temperature range. Deviation is referenced to the measured frequency at +25° C. Typical frequency stability numbers range from ±10 to 30 PPM. Temperature PPM can be de-rated by de-rating the temperature range of the product.

5.2.2.3 Aging PPM

Aging is the change in the frequency of a quartz crystal unit with the passage of time. A typical aging PPM is 2 PPM per year. How aging PPM effects product reliability over time needs to be taken into account during crystal selection.

5.2.3 Load Capacitance

Load Capacitance is the value of capacitance used in conjunction with the crystal unit in a parallel resonant oscillator circuit. In a typical system the load capacitance of WirelessUSB LP/LPstar and printed circuit board (PCB) layout is 10 pF. Load capacitance of WirelessUSB LP/LPstar is typically 7 pF, but can vary 10% from radio to radio. Load capacitance also varies from one layout to the next and is dependent on signal routing, pad size, and layer stack up.

5.2.4 Pullability and Trim Sensitivity

Pullability is the change in crystal oscillator frequency due to a change in the load capacitance. This is due to the change in parallel resonant frequency when the load capacitance is changed. Changing the frequency by changing the load capacitance is referred to as ‘pulling’. The frequency can be pulled in a parallel resonant circuit by changing the value of load capacitance. A decrease in load capacitance causes an increase in frequency, and an increase in load capacitance causes a decrease in frequency.

Trim sensitivity is very closely related to pullability. In practical terms, the two are often interchangeable. Trim sensitivity is a measure of the incremental fractional frequency change for an incremental change in the value of load capacitance. Trim sensitivity is expressed in terms of PPM/pF.

Typical trim sensitivities range from 5 to 30 PPM/pF.

5.2.5 Crystal Choice Considerations

When selecting a crystal, the easy choice is to pick a crystal with low total PPM. Crystals with low total PPM can be expensive, so some trade-offs can be made for lower cost crystals. In a single system, all crystals used with WirelessUSB LP/LPstar should be the same type. Crystals of the same type have similar frequency stability and aging characteristics. Since most systems will be in the same environment (especially HID systems) the temperature of the system will be similar. Also, all parts of the system will be built at approximately the same time. Using crystals of the same type, therefore, reduces the effect of temperature and aging PPM.

5.2.6 Clock Frequency Measurements

The WirelessUSB LP/LPstar radio clock frequency can be measured with a frequency counter on the XOUT pin. Since this signal is not used in most systems and the QFN package is difficult to probe, a PCB test point is recommended for this signal. The clock output can be enabled with firmware by writing the XTAL_CTRL_ADR register in the WirelessUSB LP/LPstar radio. During normal operations, disable this pin to remove it as a possible noise source on the PCB and to reduce current consumption.

5.2.7 Crystal Layout

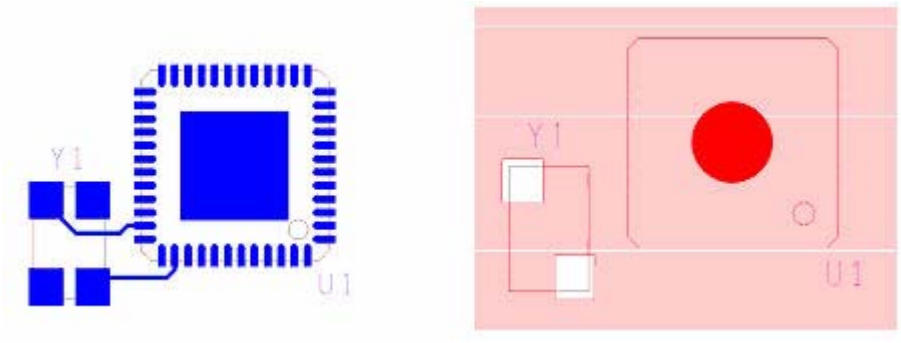
The ideal layout has the crystal on the same side of the PCB as the radio and placed close to the crystal signal pins of the radio with identical crystal trace lengths. This placement keeps the crystal trace paths short and reduces parasitic capacitances, which could produce noise in the system. The two crystal traces should have matched length, avoid vias, and have good isolation from noise sources. WirelessUSB LP/LPstar’s crystal circuit performs best when the crystal traces are closely matched in both lengths and parasitic capacitances. In summary, the crystal layout is best if the following conditions are met:

- Crystal and radio are on same side of PCB
- Crystal is placed near the crystal pin on the radio
- Crystal trace paths are as short as possible
- Match crystal trace paths for length and parasitic capacitance
- Avoid vias on crystal traces
- Isolate the crystal from noise sources

Crystal layouts should be identical for all radios in the system. By keeping layouts identical, the parasitic capacitance on the crystal traces will be similar for each PCB in the system. Similar parasitic capacitance produce similar load capacitance, thus reducing the effect of trim sensitivity PPM on each radio of the system.

On multilayer PCB's, one way of reducing parasitic capacitance on the crystal is to void the internal layers directly beneath the crystal pads (see Figure 5-1 on page 51). This is highly recommended when systems are comprised of PCB's with differing layer counts.

Figure 5-1. Crystal Layout



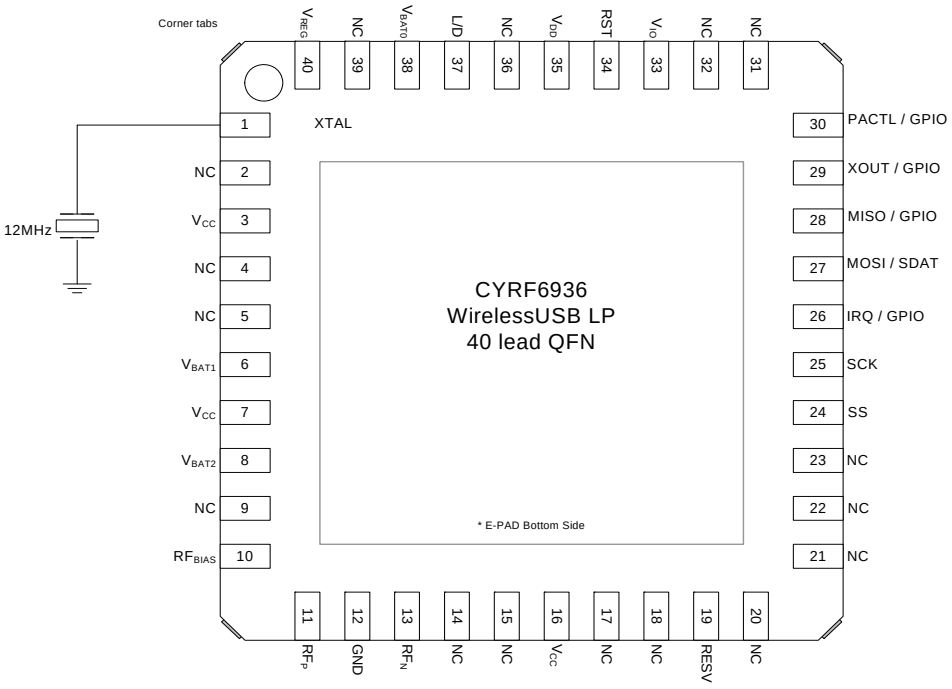
Crystal and Radio

Internal GND Plane

5.3 Typical Usage

Figure 5-2 shows the crystal circuit of LP. For LPstar, the circuit is similar as LP.

Figure 5-2. Crystal Circuit (LP for Example)

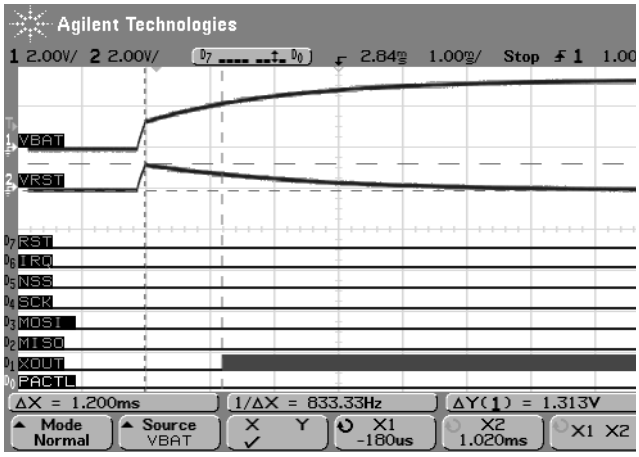


5.3.1 Startup

Two methods of starting the crystal are the cold start (power on reset) or the warm start (sleep/wake). Screen captures of signals for both are presented in the following sections.

5.3.1.1 Cold Start (Power On Reset)

Figure 5-3. Typical Power On Reset Timing

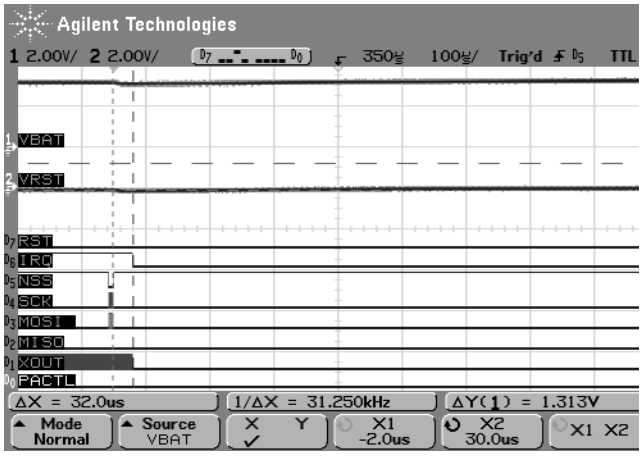


5.3.1.2 Warm Start (Sleep/Wake)

To test sleep timing:

1. Write 20h to register 0Ch
2. Write A1h to register 0Fh

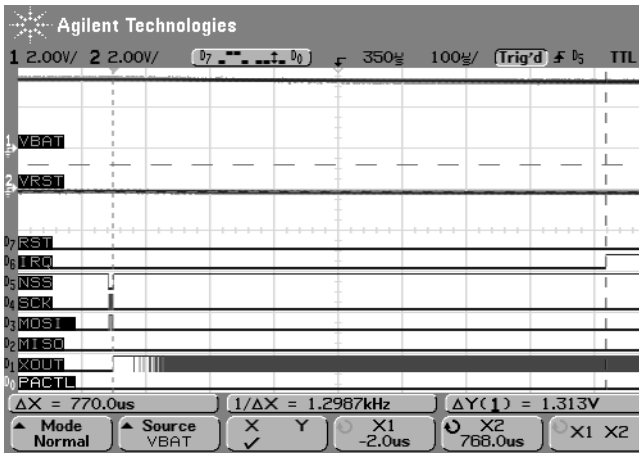
Figure 5-4. Typical Sleep Timing



To test wake timing:

1. Write 20h to register 0Ch
2. Write A1h to register 0Fh
3. Write A5h to register 0Fh

Figure 5-5. Typical Wake Timing



5.4 Notes

V_{BAT} (pin 38) is the power to the oscillator circuit. The default state of XTAL pin is V_{BAT}. The oscillator circuit can handle V_{BAT} input swings. The XTAL pin is driven to V_{BAT} in suspend AND during oscillator startup (POR, Wake, and others). In normal operation mode, during the oscillator start up sequence, the oscillator circuit drives the XTAL pin hard to GND for 40 ns. A RST pin event gates XOUT.

6. Power Management Unit

6.1 Introduction

The WirelessUSB LP's Power Management Unit (PMU) requires a single supply: 1.8V to 3.6V derived directly from an external battery voltage source or other external power supply connected to the V_{BAT} pins.

The user can configure the LP's PMU output voltage (V_{REG}) to several minimum values between 2.4V and 2.7V. V_{REG} provides up to 15 mA average load to external devices. It is possible to disable the PMU, and to provide an externally regulated DC supply voltage to the device's main supply in the range 2.4V to 3.6V. The PMU also provides a regulated 1.8V supply to internal logic. The PMU provides high boost efficiency (74–85% depending on input voltage, output voltage, and load) when using a Schottky diode and power inductor. This eliminates the need for an external boost converter in many systems where other components require a boosted voltage. However, the user achieves reasonable efficiencies (69–82% depending on input voltage, output voltage, and load) when using low cost components such as SOT23 diodes and 0805 inductors. The PMU also provides a configurable low battery detection function, which is read over the SPI interface. The user selects one of seven thresholds between 1.8V and 2.7V. The interrupt pin is configured to assert when the voltage on the V_{BAT} pins falls below the configured threshold. The low voltage IRQ signal is not a latched event. When the device is in sleep mode, the device disables battery monitoring.

Unlike a traditional DC to DC boost converter, when the input voltage is at or above the desire output voltage, there is a FET inside the part that connects from V_{BAT} to V_{REG}. This has an important advantage, in that with a traditional DC to DC boost, if $V_{BAT} < (V_{REG}$ + diode drop) the boost is switching; with the LP's PMU this is not the case, and switching does not start until $V_{BAT} < V_{REG}$. For more details, see Figure 6-1 on page 55.

If $V_{BAT} > V_{REG}$ (for example, V_{BAT} = 3.3V) then $V_{REG} = V_{BAT}$. V_{REG} is only at its configured voltage (2.4, 2.5, 2.6, or 2.7V) when $V_{BAT} < V_{REG}$. An important point to note is that the configured V_{REG} value is a guaranteed minimum, not a nominal. Thus, typically the voltage on V_{REG} with V_{REG} set to 2.7V and V_{BAT} = (for example) 1.8V varies between 2.73V and 2.78V. This is important, because it means that you can set the V_{REG} configured voltage to equal the V_{CC} (min) voltage of any chip that the PMU is powering.

Unlike LP, LPstar's PMU doesn't support boost or low battery detect function. For more details in their differences, please refer to the application note AN68105 Migration of WirelessUSB™ LP Designs to WirelessUSB™ LPstar.

6.1.1 Functional Description

The LP's PMU contains a linear low drop out regulator tied directly to the V_{BAT} (battery) pins. This regulator supplies the radio's digital logic with at least 1.8V when V_{BAT} is in the range of 1.9V to 3.6V and gracefully transitions to tracking V_{BAT} when below 1.9V (that is, the digital logic voltage droops down to as low as 1.7V when V_{BAT} is at 1.8V). The regulator transitions to sleep mode with a controlled rise time up to the external V_{BAT} voltage. It also transitions back to the regulated voltage with a similarly controlled fall time. The regulator is stable with an external 0.47 µF bypass capacitor for no load up to the maximum load current.

The LP's PMU contains a switching regulator-block based on classic, industry standard, boost mode, inductor-diode based, switching regulators. The switching regulator requires an external power inductor (tolerance ±20%) and diode (high current Schottky diode or switching diode), which adds cost. But, this cost is offset by the fact that the on-chip regulator alleviates the need for the designer to provide one externally; it supports up to 15 mA of additional load current to other devices. Efficiency is over 80% at high load current. It uses only two external, inexpensive electrolytic capacitors (tolerance -20% to +80% over the 0–70C temperature range). Performance depends on external component selection and PCB layout.

When PMU EN = 1

- When PMU OUTV = 00: The PMU output equals the battery voltage from 2.7V to 3.6V. It holds the output at 2.7V minimum for battery voltages from 1.8V to 2.7V through an active inductor-diode charge pump circuit connected to the L/D pin.
- When PMU OUTV = 01: The PMU output equals the battery voltage from 2.6V to 3.6V. It holds the output at 2.6V minimum for battery voltages from 1.8V to 2.6V through an active inductor-diode charge pump circuit connected to the L/D pin.

- When PMU OUTV = 10: The PMU output equals the battery voltage from 2.5V to 3.6V. It holds the output at 2.5V minimum for battery voltages from 1.8V to 2.5V through an active inductor-diode charge pump circuit connected to the L/D pin.
- When PMU OUTV = 11: The PMU output equals the battery voltage from 2.4V to 3.6V. It holds the output at 2.4V minimum for battery voltages from 1.8V to 2.4V through an active inductor-diode charge pump circuit connected to the L/D pin.

When PMU EN = 0

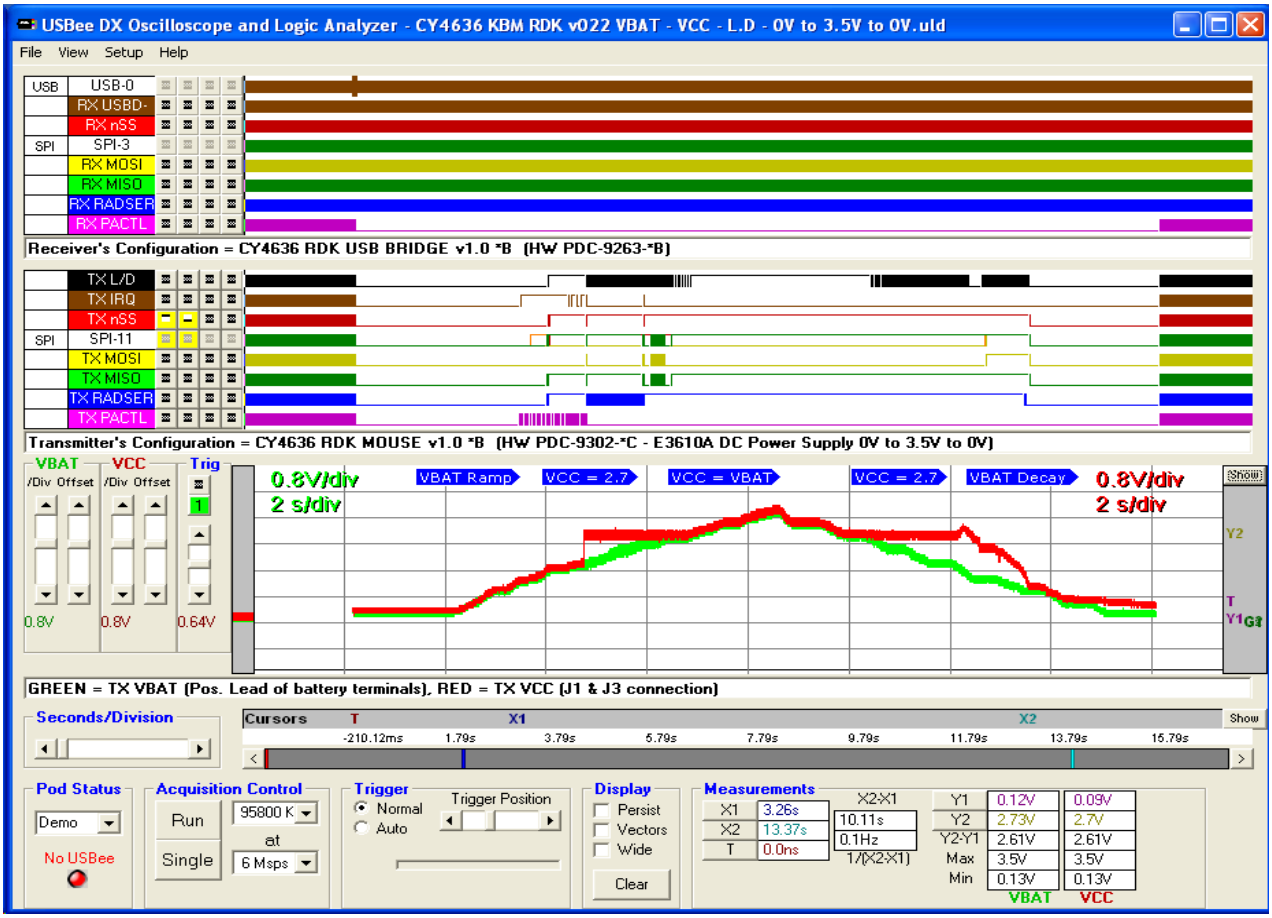
- The user disables the PMU (through SPI write) to allow direct drive from an external 2.4V to 3.6V voltage source (such as, USB connection through a linear 3.3V regulator). See the PWR_CTRL_ADR bit 7 (PMU Enable) descriptions in the 38-16015 data sheet for more details. See Cypress Semiconductor Support on page 9.
- The user disables the PMU (through SPI write) to allow an external boost voltage regulator to take over control of the V_{REG} voltage. See PWR_CTRL_ADR bit 7 (PMU Enable) descriptions in the 38-16015 data sheet for more details.

For LP , V_{REG} tracks the battery until the battery voltage falls below the V_{REG} programmed threshold (2.4, 2.5, 2.6, or 2.7V V_{REG} = 2.7V is the default). At voltages below the V_{REG} threshold, the PMU's switching regulator turns ON and holds the V_{REG} voltage to be no less than the threshold value for V_{REG} load current from 0 to 30 mA. The V_{REG} voltage then, is a saw tooth waveform that is approximately 10 Hz to 110 KHz with ripple of 20 mVpp to 110 mVpp. This depends on the V_{REG} load current and the V_{BAT} voltage; it potentially is from 1.7V to the V_{REG} programmed threshold. The rise slope of the sawtooth is nearly constant but the amplitude is a function of V_{BAT}. The fall slope is V_{REG} load current dependant. The minimum voltage of the sawtooth waveform is greater than or equal to the V_{REG} programmed threshold over the V_{REG} load current range of 0 to 30 mA. This includes the 0–15 mA of external load current allowed by the user for powering a microcontroller, optical diode, or both.

For LP , the V_{REG} output tracks the battery voltage when the V_{BAT} voltage is above the programmed threshold through a PFET switch that connects V_{BAT} to V_{REG} when $V_{BAT} > V_{REG}$ threshold. A voltage regulator charge pump automatically turns on when the battery voltage drops below the programmed threshold. The threshold is programmable for 2.4V, 2.5V, 2.6V, and 2.7V to accommodate currently available microcontrollers and optical diodes. The V_{REG} output supplies power for much of the radio chip circuitry (up to 15 mA) and can supply an additional 15 mA to power external devices such as a low power microcontroller and optical diode used in a wireless mouse application. For more details, see Figure 6-1 on page 55.

LPstar does not have a pin of V_{REG}. For more details please refer to LPstar datasheet.

Figure 6-1. Typical VBAT vs. VCC Behavior



The LP's PMU contains a low voltage monitoring and indication circuit (LVI) that has a programmable trip voltage of (1.8V, 2.0V, 2.2V, and VREG) where V_{REG} is the programmed regulated voltage setting (2.4V, 2.5V, 2.6V, or 2.7V). Use the LVI V_{REG} settings to inform the microcontroller of the switching regulator onset. You can also use the LVI as a seven-level battery capacity indicator.

The PMU supply current in sleep mode is approximately 50 µA (nominal) when supplying V_{REG} leakage load current. See the PWR_CTRL_ADR bit 5 (PMU Sleep Mode Enable) descriptions in the 38-16015 data sheet for more details.

The PMU disabled mode current is approximately 0.1 µA. See the PWR_CTRL_ADR bit 7 (PMU Enable) descriptions in the 38-16015 data sheet for more details.

Note If you use an external supply for V_{BAT}, make certain it is of the 'linear' regulator type that can deliver at least a 0.5A current (remember, the switching regulator's inductor has a peak charge current of up to 400 mA).

6.1.2 Power Supply Choices

There are two common ways of powering V_{BAT}.

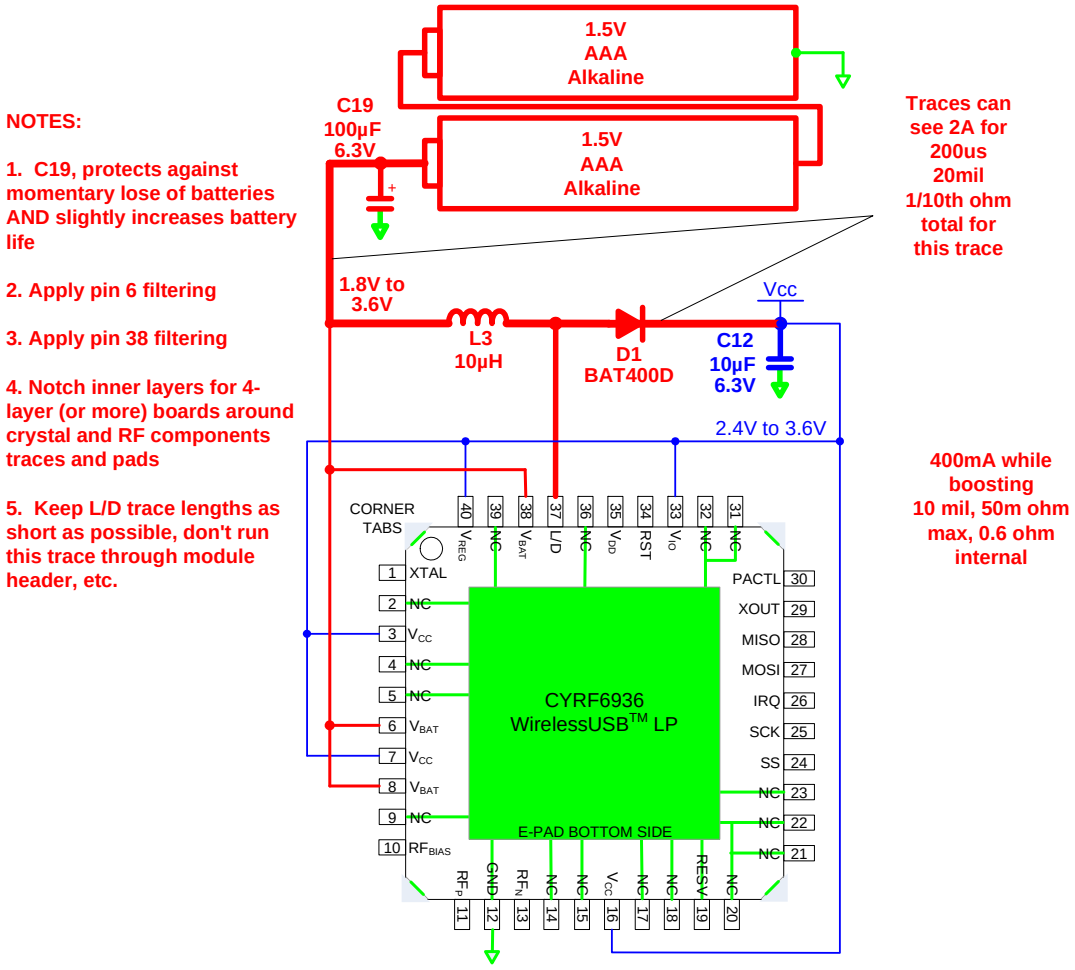
1. Use two AAA alkaline batteries or similar (coin-cell, and others). See Figure 6-2 on page 56.
2. Use a low noise LDO linear regulator. See Figure 6-3 on page 57.

6.1.2.1 Battery Configuration

If you use batteries, make certain to connect the battery to V_{BAT}. This method requires the use of an inductor, diode, and capacitors. The LP's PMU generates V_{REG}. It also generates the 1.8V for the core.

Unlike LP, LPstar does not support the boost functionality and hence in the below figure, the circuit consisting of the L/D pin, L3 and D1 is applicable for only LP. And the V_{BAT} and V_{CC} should be both from 2.7 V to 3.6 V.

Figure 6-2. Typical Battery Application Circuit for LP



The 100 µF capacitor (C19) primarily supports the supply during temporary battery disconnection during physical shock (for example, banging or dropping). Place the 100 µF capacitor close to the battery terminals between the power connector to the PCB and the supply line or plane that feed V_{BAT} to the radio. This effectively bypasses the inductance of the battery and power supply leads which sometimes are very long (a few inches or so). Keep the resistance of the V_{BAT} supply line between 100 µF capacitor and the radio very low (milliohms) to ensure no extra droop voltage occurs when the switching regulator is running. For a four layer board where V_{BAT} is an internal plane, you can place the 100 µF capacitor anywhere since the resistance of the plane is extremely low. But, for two layer boards where V_{BAT} is a trace, make certain that the trace is wide between the power connector and capacitor and between the capacitor and the radio so that the overall resistance is in the milliohm range.

The output impedance of near dead batteries is around 1–2 ohms which causes a lot of ripple when the switching regulator is running due to the 400 mA peak inductor charging current. You do not want the layout parasitic resistance to add to the battery resistance; that way you can take out all the energy from the battery that is possible before the system dies. Keep the battery wire resistance very low as well, so that only the battery internal resistance is the limiting factor.

6.1.3 Fixed Value Inductor and Capacitor

By the LP design, the values of the inductor and V_{REG} capacitor are fixed at 10 µH and 10 µF, respectively.

6.1.3.1 Selecting a Suitable Inductor

The Taiyo Yuden CB2012T100MR reference example part specification limit for 1 mS current pulse 1% duty cycle is 2.6 amps. The Taiyo Yuden inductor is only recommended for minimal loads (radio + MCU only, no additional loads). For external loads use a Sumida Part # CDH53100LC inductor or similar.

In boost regulators such as this, inductor saturation during current ramp-up must be avoided. Most good quality inductors will not show this problem. Inductors that are too small, or poor quality, will go into saturation, thus limiting their ability to store enough energy during the switch-ON conduction cycle, and may not produce enough DC output power. This problem is noticeable at low battery voltage.

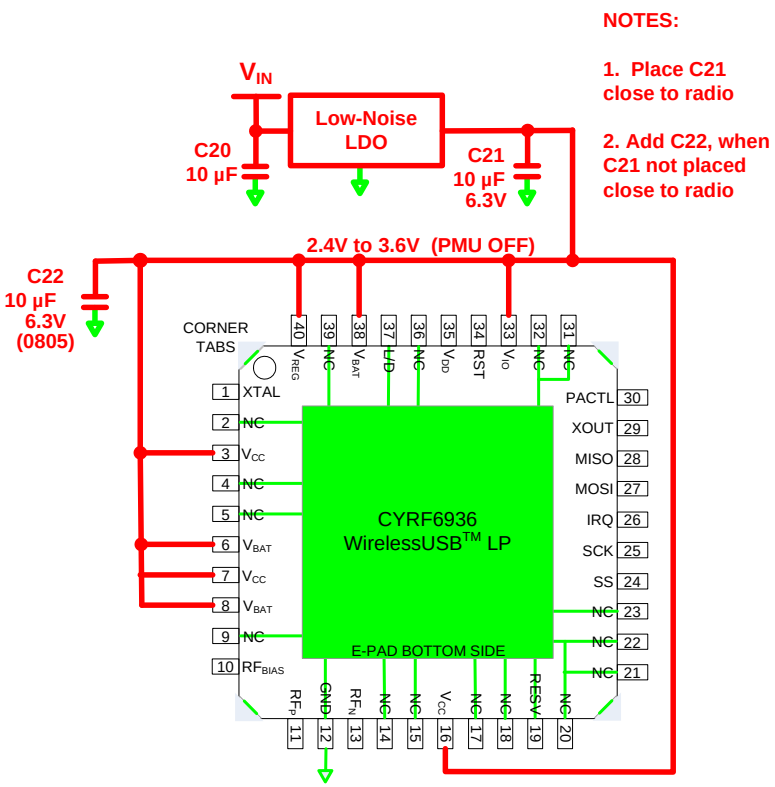
Inductor saturation can be observed with an oscilloscope, monitoring voltage at the L/D pin. During the switch-ON conduction cycle, voltage should be small, equal to the IR drop of R_{ds} of the internal FET switch, multiplied by the linearly increasing inductor current. If saturation occurs, however, the current increase is no longer linear, but a fast exponential rise. This would indicate a poor inductor selection.

6.1.3.2 Low-Noise LDO Linear Regulator Configuration

In this configuration for LP, V_{BAT}, V_{CC}, and V_{REG} are all tied to the output of a low noise LDO. The PMU boost function is not used. The inductor and diode are not used. L/D pin is connected to ground. The only PMU function is the 1.8V regulator for the core.

For LPstar, the configuration is similar to LP except the LDO output must be from 2.7 V to 3.6 V.

Figure 6-3. Typical Linear Regulator Application Circuit for LP



6.1.4 Filtering Power Supply Noise

As previously mentioned, the analog RF performance of the radio is significantly degraded by power supply noise. Apply the following filtering rules for the following V_{BAT} conditions.

When V_{BAT} range 1.8V < > 3.6V (example 2 x AAA):

- Pin 6 (V_{BAT1})–shunt capacitor C = 1 µF, series resistor R = 47 ohms
- Pin 38 (V_{BAT0})–shunt capacitor C = 10 µF, series resistor R = 1 ohm

When V_{BAT} range 2.0V < > 3.6V (example, CR2032):

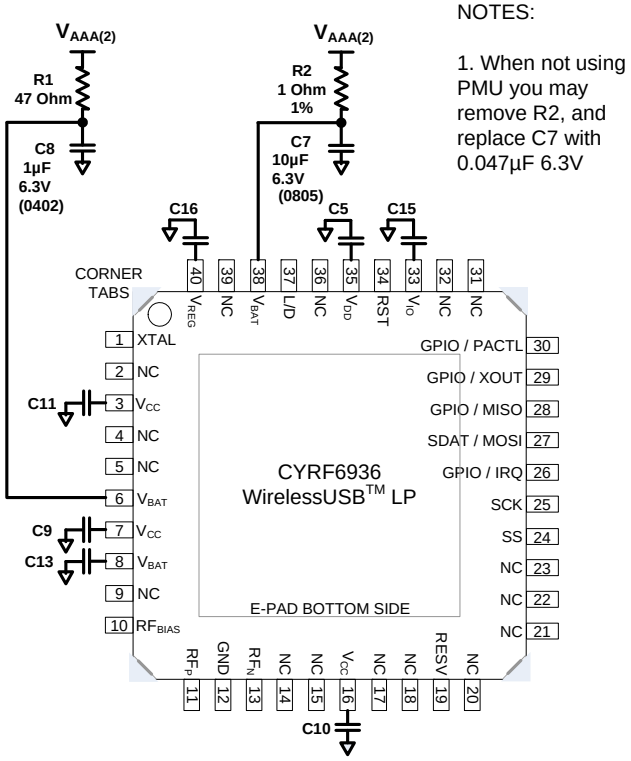
- Pin 6 (V_{BAT1})–shunt capacitor C = 0.047 µF
- Pin 38 (V_{BAT0})–shunt capacitor C = 10 µF, series resistor R = 1 ohm

When V_{BAT} range 2.4V < > 3.6V (example, Linear LDO):

- Pin 6 (V_{BAT1})–shunt capacitor C = 0.047 µF
- Pin 38 (V_{BAT0})–shunt capacitor C = 0.047 µF

For more details, see Figure 6-4 on page 58.

Figure 6-4. Typical Power Supply Filtering Options for LP



6.1.5 Switching Regulator Frequency

The steady state frequency of the switching regulator varies with load current and can range from 10 Hz (no load in sleep mode) and 110 KHz when delivering maximum current at V_{BAT} = 1.8V. The start up oscillator, which only runs when the part is first connected to a supply, runs at about 250 KHz.

6.2 Sleep Mode

To shut down the device to a fully static sleep mode, write to the FRC_END = 1 and END_STATE = 000 bits in the XACT_CFG_ADR register over the SPI interface. The device enters sleep mode within approximately 35 µs after the SPI/SS signal goes high at the end of this SPI transaction.

Note The time to enter sleep mode is largely set by the V_{DD} supply decoupling capacitor (0.47 µF) charging up to V_{BAT} in sleep mode. Once V_{DD} = V_{BAT}, the sleep mode current settles down to its minimum value. The maximum time occurs when V_{BAT} = 3.6V

6.2.1 Wake from Sleep Response Time

The wake from sleep response time is crystal dependent, but typically less than 1 ms.

In addition, the radio has an oscillator stable interrupt that when enabled, is driven onto the IRQ pin.

Note The MCU talks to the radio's digital block with clocks turned OFF.

6.3 LVI IRQ

LVI IRQ is an active level interrupt for LP and PRoC LP only. If the low voltage detect is enabled as an IRQ, it is passed from the internal analog signal straight to the IRQ pin. Reading the status does not clear an event, it reflects the status all the time. LVI is automatically disabled when the device is in sleep mode, otherwise it is active.

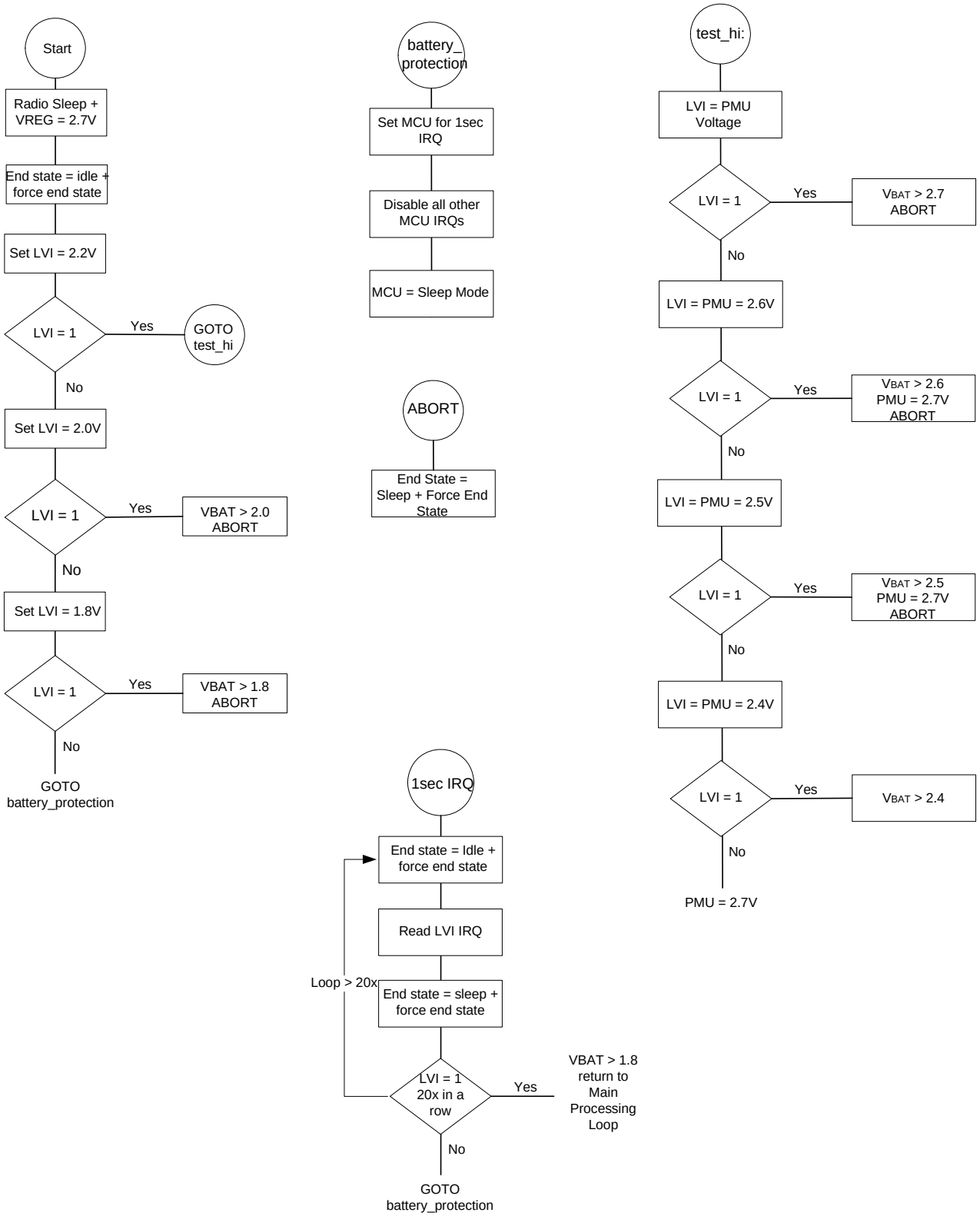
Applications that want to use the LVI feature to monitor the battery voltage, must enable the PMU during LVI sampling.

6.3.1 Implementing Low Voltage Monitoring

Figure 6-5 on page 60 shows one example of how you can make a crude battery monitoring ADC using the LVI IRQ.

Note Do not monitor the battery voltage while actively transmitting or receiving, as higher Icc affects the measurement.

Figure 6-5. Battery Monitoring Example



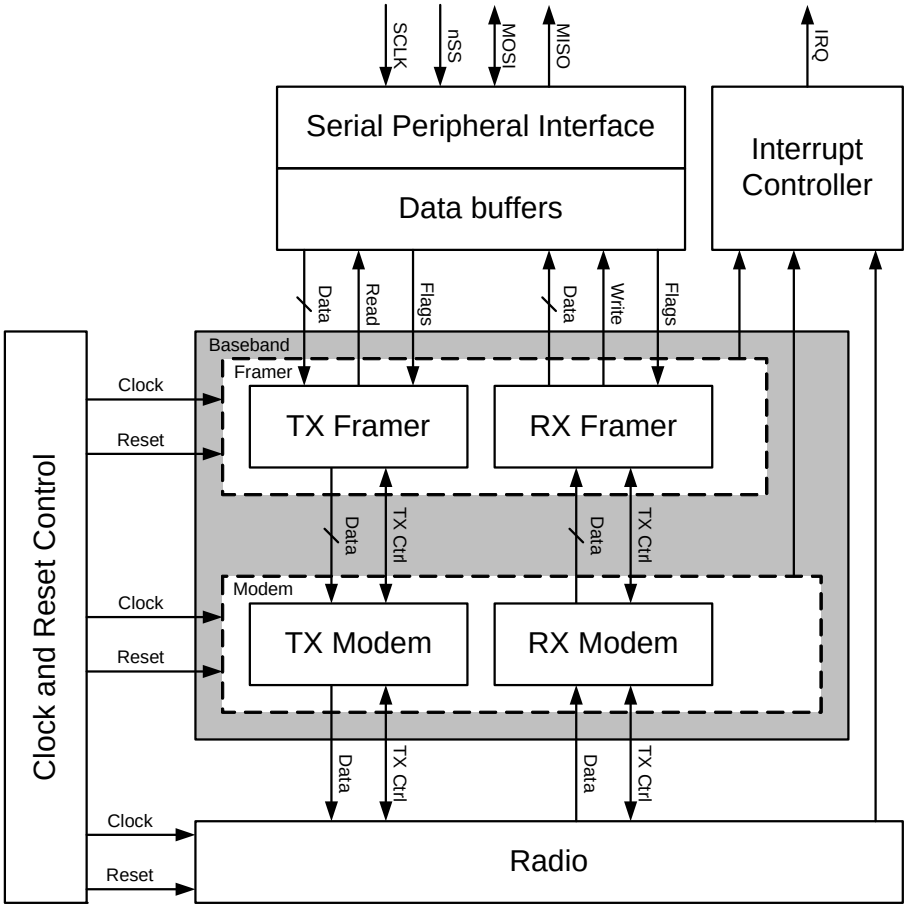
7. Digital Baseband

7.1 Introduction

The baseband consists of numerous sub-blocks. At a high level, it serves as the interface between the SPI block data buffers and the radio analog front end. The primary sub-blocks that are discussed in this section are the modem and framer. The modem manages the DSSS encoding and decoding of the data stream; the framer is responsible for start of packet generation and reception, CRC16 generation and checking, and also end of packet detection and length field management.

The baseband also provides control and interface to the clock and the power management unit. Those functions are discussed in separate chapters.

Figure 7-1. Baseband Simplified Block Diagram



There are multiple data rates available in the WirelessUSB LP family. For DSSS modes, the user's data is encoded in a series of chips, based on the configuration register settings. In raw GFSK mode the chips transmitted over the air are equivalent to the user's data. Regardless of the user selected data rate, the rate that information is actually transferred over the air (the chip rate) is always 1 Mbps. Different DSSS data rates are achieved by encoding a different number of bytes in each symbol.

When in receive mode with packet framing enabled, the device is always ready to receive data transmitted at any of the supported bit rates. This enables the implementation of mixed-mode systems in which different devices use different data rates. This also enables the implementation of dynamic data rate systems that use high data rates at shorter distances or in a low-moderate interference environments, and change to lower data rates at longer distances or in high interference environments.

7.2 Modem

The modem implements the coding layer that converts the stream of data in the data buffers to sequences of chips. It works with the framer to provide start of packet markers and other structures necessary to packetize the data. The modem also decodes received data, using a correlator function to center and decode the chip sequence.

Various data modes are supported by the modem, configured by register settings. These modes, in conjunction with a user specified 32 or 64 chip PN code, implement a variety of data rates. Data mode is set with bits 4:3, DATA MODE, of TX_CFG_ADR register 0x03. The length of the data PN code is set with bit 5, DATA CODE LENGTH. Data modes and data rates are listed in the following table.

Table 7-1. Data Modes and Data Rates

Data Mode	Abbreviation	Data Mode Bit Settings	Data Code Length Bit Setting	Data Rate
Gaussian Frequency Shift Key	GFSK	00	N/A	1 Mbps
Eight x Data Rate(64chip for LP only)	8DR	01	0 (32 chips/bit)	250 Kbps
			1 (64 chips/bit)	125 Kbps
Double Data Rate(for LP only)	DDR	10	0 (32 chips/bit)	62.5 Kbps
			1 (64 chips/bit)	31.25 Kbps
Single Data Rate(for LP only)	SDR	11	0 (32 chips/bit)	31.25 Kbps
			1 (64 chips/bit)	16.125 Kbps

The first mode, raw GFSK, does not use DSSS encoding of the data. Therefore each chip sent across the air during the data phase is directly equivalent to 1 bit of user data. The data rate is 1 Mbps. As shown, the remaining three modes can be used with either 64 or 32-chip data PN codes resulting in two possible data rates for each mode. In the DSSS modes a single transmission of a PN code equates to one symbol. Each symbol corresponds to a different number of data bits depending upon the data mode.

For the DSSS mode, a single 16 byte (128 bit) file register is used for storing the code: DATA_CODE_ADR file register 0x23. The value in this register is:

0x02F9939702FA5CE3012BF1DB0132BE6F

Do not change this value. All WirelessUSB LP family systems in all applications can use this code. Separate start of packet PN codes, discussed in the section Start of Packet on page 65, are used to co-locate different systems.

This register is broken up in different ways to accommodate the different data modes. The contents meet the requirements of our standard PN codes for good auto correlation and cross correlation. They also meet the requirements of multiplicative codes, meaning that its sub-sections can be used for both 32-chip and 64-chip codes that also have good auto correlation and cross correlation properties on their own.

7.2.1 Gaussian Frequency Shift Key

Gaussian Frequency Shift Key (GFSK) mode is the fastest data rate, however care must be taken in its use. By definition, this mode eliminates the advantages of DSSS technology making it much more susceptible to errors.

Some things to note when using GFSK mode:

A start of packet is required as discussed in the section Framer on page 65. The SOP is PN code encoded data, thus detection of a start of packet is as reliable for GFSK as it is for other modes.

The length field is also required, but is not encoded data. Therefore, the probability of this field being in error is much higher with GFSK packets. An error in this field results in erroneously truncated data or excess random data. An erroneously long length field also ties up the correlator for the amount of time required to receive a packet of the erroneous length.

The data is not encoded, therefore any chip error results directly in a data bit error. This makes the entire packet more susceptible to errors.

Only slow channels are supported. Bit 0, ALL SLOW, of ANALOG_CTRL_ADR register 0x39 should be set in order to make use of the full range of channels.

7.2.2 Eight x Data Rate

Eight x Data Rate (8DR) is the most robust mode of the WirelessUSB LP family of devices. The radio was designed around this operating mode and certain functions enable it to more accurately center on bits and correctly decode the chip stream. This has been determined in practice through in-chamber characterization and outdoor range testing.

There is generally little need for the DDR and SDR modes except in cases where backward compatibly is required; they generally provide slower performance and increased power consumption. There may be situations where the GFSK mode provides some advantages, buts its use has to be balanced against the less reliable transmission of data.

In 8DR mode, each symbol represents eight bits of user data. The 32-chip codes are used for 250 Kbps rate and 64-chip codes are used for 125 Kbps rate. In this proprietary mode, 128 derivative PN codes are generated based upon the organization of the contents of the DATA_CODE_ADR file register 0x23. These 128 codes plus the state of the code (normal or inverted) are used to represent the eight bits of data.

7.2.3 Double Data Rate

Double Data Rate (DDR) mode works in a fashion similar to the older WirelessUSB LS and LR devices. In DDR, each symbol represents two bits of data. Two PN codes are used, derived from the data stored in DATA_CODE_ADR file register 0x23, as shown in 7-2

Table 7-2. DDR Data PN Code Configuration

	DATA_CODE_ADR File Register 0x23															
	Byte 0	Byte 0	Byte 0	Byte 0	Byte 0	Byte 0	Byte 0	Byte 0	Byte 0	Byte 6	Byte 5	Byte 4	Byte 3	Byte 2	Byte 1	Byte 0
64-Chip DDR	Data PN Code 1								Data PN Code 0							
32-Chip DDR	Not Used				Data PN Code 1				Not Used				Data PN Code 0			

These two codes and their two states, normal and inverted, allow the encoding of two bits of data.

Table 7-3. DDR Symbol Encoding

Symbol	Transmitted Code
00	Data PN code 0
01	$\overline{\text{Data PN code 0}}$
10	Data PN code 1
11	$\overline{\text{Data PN code 1}}$

7.2.4 Single Data Rate

Single Data Rate (SDR) is the simplest encoding scheme. A single PN code is required. Each symbol represents a single bit of user data. The data PN code is transmitted in its normal state to represent a ’1’ and inverted to represent a ‘0’.

Table 7-4. SDR Data PN Code Configuration

	DATA_CODE_ADR File Register 0x23															
	Byte 0	Byte 0	Byte 0	Byte 0	Byte 0	Byte 0	Byte 0	Byte 0	Byte 0	Byte 6	Byte 5	Byte 4	Byte 3	Byte 2	Byte 1	Byte 0
64-Chip DDR	Not Used								Data PN Code							
32-Chip DDR	Not Used				Not Used				Not Used				Data PN Code			

7.3 Backward Compatibility

The CYRF6936 IC is fully interoperable with the main modes of the first generation devices, namely the CYWUSB6934-LS and CYWUSB6935-LR devices. The 62.5 kbps mode is supported by selecting 32 chip DDR mode. Similarly, the 15.675 kbps mode is supported by selecting 64 chip SDR mode.

In this way, a suitably configured CYRF6936 IC device may transmit data to or receive data from a first generation device, or both. Backward compatibility requires disabling the SOP, length, and CRC16 fields.

Shown in Table 7-5 and Table 7-6 are the different configurations of the registers and firmware that enable a second generation radio to communicate with a first generation radio.There are two possible modes: SDR mode and DDR mode (8-DR and GFSK modes are not present in the first generation radio). The second generation radio must be initialized using the RadioInitAPI of the LP radio driver and then the following register bits need to be configured to the given byte values. Essentially, the following deactivates the added features of the second generation radio and takes it down to the level of the first generation radio; the data format, data rates, and the PN codes used are recognizable by the first generation radio.

Table 7-5. DDR Mode

Register	Value	Description
TX_CFG_ADR	0X16	32 Chip PN Code, DDR, PA = 6.
RX_CFG_ADR	0X4B	AGC is enabled. LNA and attenuator are disabled. Fast turnaround is disabled, the device uses high side receive injection, and Hi-Lo is disabled. Overwrite to receive buffer is enabled and the RX buffer is configured to receive 8 bytes maximum.
XACT_CFG_ADR	0X05	AutoACK is disabled. Forcing END STATE is disabled. The device is configured to transition to Idle mode after a receive or transmit. ACK timeout is set to 128 µs.
FRAMING_CFG_ADR	0X00	All SOP and framing features are disabled. Disable LEN_EN=0 if EOP is needed.
TX_OVERRIDE_ADR	0X04	Disable Transmit CRC-16.
RX_OVERRIDE_ADR	0X14	The receiver rejects packets with a zero seed. The RX CRC-16 Checker is disabled and the receiver accepts bad packets that do not match the seed in CRC_Seed registers. Basically, this helps in communication with the first generation radio that does not have CRC capabilities.
ANALOG_CTRL_ADR	0X01	Set ALL SLOW. When set, the synthesizer settle time for all channels is the same as the slow channels in the first generation radio.
DATA32_THOLD_ADR	0X03	Sets the number of allowed corrupted bits to 3.
EOP_CTRL_ADR	0x01	Sets the number of consecutive symbols for noncorrelation to detect end of packet.
PREAMBLE_ADR	0xAAAA05	AAAA are the 2 preamble bytes. Any other byte can also be written into the Preamble register file. Recommended counts of the preamble bytes to be sent should be >4.

Table 7-6. SDR Mode

Register	Value	Description
TX_CFG_ADR	0X3E	64 Chip PN Code, SDR Mode, PA = 6.
RX_CFG_ADR	0X4B	AGC is enabled. LNA and attenuator are disabled. Fast turnaround is disabled, the device uses high side receive injection, and Hi-Lo is disabled. Overwrite to receive buffer is enabled and RX buffer is configured to receive 8 bytes maximum. Enables RXOW to allow new packets to be loaded into the receive buffer. This also enables the VALID bit, which is used by the first generation radio's error correction firmware.
XACT_CFG_ADR	0X05	AutoACK is disabled. Forcing END STATE is disabled. The device is configured to transition to Idle mode after receive or transmit. ACK timeout is set to 128 µs.
FRAMING_CFG_ADR	0X00	All SOP and framing features are disabled. Disable LEN_EN=0 if EOP is needed.
TX_OVERRIDE_ADR	0X04	Disable Transmit CRC-16.
RX_OVERRIDE_ADR	0X14	The receiver rejects packets with a zero seed. The RX CRC-16 checker is disabled and the receiver accepts bad packets that do not match the seed in the CRC_seed registers. Basically, this helps in communication with the first generation radio that does not have CRC capabilities.
ANALOG_CTRL_ADR	0X01	Set ALL SLOW. When set, the synthesizer settle time for all channels is the same as the slow channels in the first generation radio for manual ACK consistency.
DATA64_THOLD_ADR	0X07	Sets the number of allowed corrupted bits to 7, which is close to the recommended 12% value.
EOP_CTRL_ADR	0xA1	Sets the number of consecutive symbols for noncorrelation to detect end of packet.
PREAMBLE_ADR	0xAAAA09	AAAA are the 2 preamble bytes. Any other byte can also be written into the Preamble register file. Recommended counts of the preamble bytes to be sent should be >8.

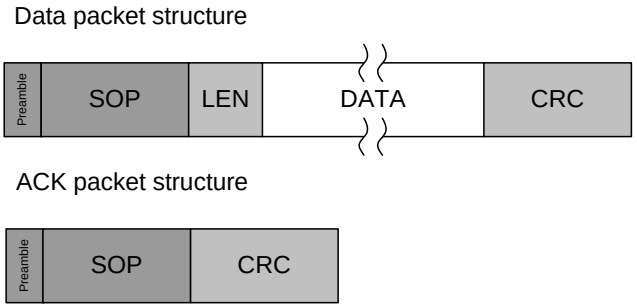
7.4 Framer

The framer block consists of two core state machines, one for the transmit path and the other for the receive path, each running at 12 MHz. Packetized WirelessUSB data has a number of sub-components. The framer is actually not responsible for all of them. The modem block manages the preamble and start of packet symbols; however, all of the packet structures are covered here for simplicity.

7.4.1 Packet Structure

Data packet and ACK packet structures are shown below.

Figure 7-2. Packet Structures



The darker gray areas are those handled by the modem block. The lighter gray areas are handled by the framer. The white regions are the user's data going to or coming from the data buffers. All of the packet sub-components can be disabled, but some are required in order to make use of certain data modes, and other fields, such as preamble, should be used to ensure the robustness of the wireless link.

The preamble, SOP, length, and CRC values are not stored or transmitted from the buffer memory. The TX framer and modem insert these values based on register settings and the RX framer and modem extract them from the incoming packet.

7.4.1.1 Preamble

The preamble is a 16-chip sequence transmitted at the start of a packet. Its primary purpose is to allow the receive correlator to establish the appropriate bit centering and synch up on the data stream before the other structures in the packet are transmitted.

The preamble pattern and length are set through the PREAMBLE_ADR file register 0x24. Byte 1 of the file register establishes the number of repetitions of the 16-chip sequence. The preamble may be disabled by writing 0x00 to this byte, but it is highly advised to make use of the preamble.

Disabling the preamble has a small savings in terms of overall packet length in most data modes, but can have a significant effect on the ability to receive a packet. Without the preamble sequence to allow the receive correlator an opportunity to synch on the data stream, it is much more likely for the receiver to fail to correlate on the SOP symbols in the case of framed data, or on the data itself in the case of unframed modes. Failing to correlate on a SOP means that no data packet will be received. For unframed modes, the result can be corrupted data.

The second and third bytes of the file register set the least significant eight chips and most significant eight chips of the preamble sequence respectively.

Note that when reading or writing this register, all three bytes must be read or written. Failing to do so means that the contents will be shifted by the number of reads or writes. A subsequent access will then not start with the first byte.

The default value of this register, 0x333302, is the recommended preamble sequence.

Figure 7-3. Default, Recommended Preamble Sequence

In general a 2-3 symbol repetition with a pattern such as 0x3333 or 0x5555 should provide suitable performance. Increasing the length of the preamble may help with correlation and packet reception in cases with higher interference, borderline firmware timing, or other issues. Before using other preamble patterns in a production application, their performance should be thoroughly evaluated in the intended environment.

7.4.1.2 Start of Packet

Start of packet (SOP) symbols are used to bound the beginning of the packet data, and also provide the added feature of encoding the data rate for the remainder of the packet. Use of SOP is required for GFSK and 8DR modes, but optional for other modes. Use of SOP is also required in order to make use of mixed mode systems and dynamic data rate systems, in which the receiver automatically detects and receives incoming packets at any data rate.

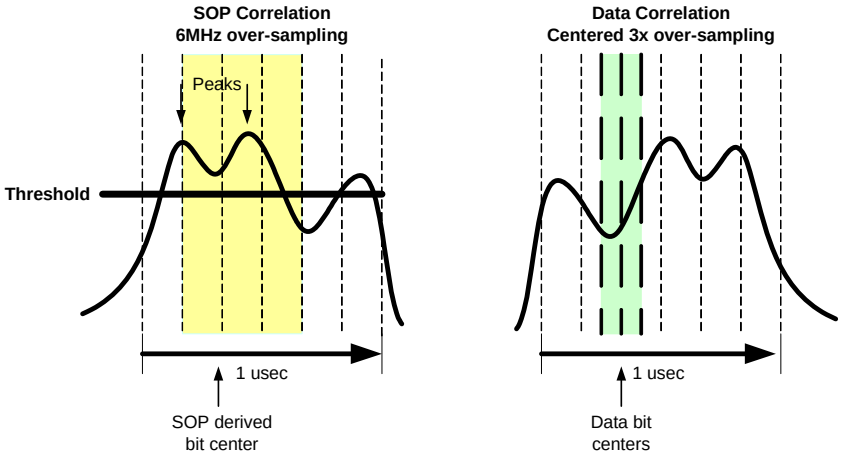
SOP is composed primarily of two symbols. Each symbol is either a 32-chip or 64-chip PN code(LPstar does not support 64-chip PN code). This is even true of the raw GFSK mode. PN codes used for SOP are different than PN codes used for the data portion of the packet.

WirelessUSB LP family devices correlate packets based upon SOP PN codes when enabled. What this means is that if a receiver successfully correlates on an SOP, then the receive state machine *will* run to completion and will receive a packet, regardless of the validity of the remainder of the

data. Failing to correlate on an SOP results in no packet being received even if valid data follows.

The modem uses SOP codes to provide additional bit centering, which is why they are required for 8DR and GFSK modes. The centering algorithm looks at peaks at an over-sampled rate of 6 MHz during the SOP phase. The receive correlator performs a centered 3x over-sampling on the data portion of the packet, but looking at a much narrower window around the pre-established bit center.

Figure 7-4. SOP Derived Bit Centering



SOP is enabled or disabled with bit 7, SOP EN, of the FRAMING_CFG_ADR register 0x10. The length, 32 chip or 64 chip, is set using bit 6, SOP LEN, of the FRAMING_CFG_ADR register 0x10.

The SOP code itself is stored in the SOP_CODE_ADR file register 0x22, which consists of eight bytes. When SOP LEN is set to '0', indicating 32 chips, only the first four bytes of SOP_CODE_ADR are used. When reading or writing this register the user must access all eight bytes to ensure the file register is reset to the correct start position for the next access.

Cypress has researched PN codes to determine codes with properties acceptable for use by WirelessUSB LP family systems. The first four bytes of these codes are suitable as standalone 32-chip codes and the full eight bytes are satisfactory 64-chip codes.

Table 7-7. Cypress Recommended SOP PN Codes

SOP PN Codes
0x91CCF8E291CC373C
0x0FA239AD0FA1C59B
0x2AB18FD22AB064EF
0x507C26DD507CCD66
0x44F616AD44F6E15C
0x46AE31B646AECC5A

Table 7-7. Cypress Recommended SOP PN Codes

SOP PN Codes
0x3CDC829E3CDC78A1
0x7418656F74198EB9
0x49C1DF6249C0B1DF
0x72141A7F7214E597

The complete SOP portion of the packet consists of a single symbol, followed by a possible 4-bit time delay, followed by a second symbol, followed by a 1-bit time delay. The symbols are either normal or inverted transmissions of the SOP PN code. The status of the PN code and the presence or absence of the delay are used to encode the data rate for the remainder of the packet.

Table 7-8. Data Rate Encoding in Start of Packet

Data Rate	1st Symbol	4 Bit Time Delay	2nd Symbol	1 Bit Time Delay
32-Chip DDR	SOP Code	No	SOP Code	Yes
64-Chip DDR	SOP Code	No	$\overline{\text{SOP Code}}$	Yes
GFSK	$\overline{\text{SOP Code}}$	No	SOP Code	Yes
32-Chip 8DR	SOP Code	Yes	SOP Code	Yes
64-Chip 8DR	SOP Code	Yes	$\overline{\text{SOP Code}}$	Yes
ACK	$\overline{\text{SOP Code}}$	No	$\overline{\text{SOP Code}}$	Yes

When discussing co-location using channel and PN code to distinguish between systems, it is the SOP PN codes and not the data PN code that are used to provide co-location.

7.4.1.3 Data

The data portion of the packet consists of the user's data bytes. Zero length packets are valid, and lengths can be up to 40 bytes. Framed 64-chip DDR packets are further limited to 16 bytes maximum length.

The 40 byte maximum packet size is based upon the crystal tolerance specification of ±30 PPM. The transmitting and receiving sides of the system can thus be off by as much as 60 PPM. At this extreme, up to 40 bytes can be received without losing bit centering.

In theory larger length packets can be transmitted with tighter tolerance; however, Cypress has not characterized this usage. The 60 PPM tolerance must also consider all factors such as temperature and aging variations of the crystal. Thus 40 bytes is the recommended maximum packet size.

Other factors may also have to be considered in selecting the packet size appropriate for the application. Any given bit has a certain probability of being affected by interference and thus being corrupted. With DSSS data the probability is much less than with raw GFSK data; however, in either case a corrupted data bit will result in the entire packet being flagged with an error. Thus large packets might have a higher susceptibility to error, requiring re-transmission of the data. Smaller packets have a higher overhead due to the framing. One other factor to consider is that for packet lengths greater than 16 bytes, the length of the transmit and receive buffers, the application has to add or remove data to prevent an under or over run condition. Each application will have to evaluate the right balance.

Transmit data to be sent is written to the TX_BUFFER_ADR file register 0x20. Any data in this buffer can be retransmitted without resetting and reloading the FIFO by setting TX_GO again. Received data is read from the RX_BUFFER_ADR file register 0x21.

There is also an optional capability to store data valid bits, primarily for compatibility with the legacy WirelessUSB LS/LR devices. This mode is set with bit 0, VLD EN, of the RX_CFG_ADR register 0x06. In this case the received data is presented with the byte of valid data bits. When the data is stored in the receive data buffer, the eight valid bits for each stored byte are stored in the next buffer location after the data. Thus only eight bytes of the receive data buffer are available for actual application data.

7.4.1.4 Length

Length is a 1 byte field. It is required in raw GFSK mode and in 8DR mode. It is optional in other modes. Length is enabled by setting bit 5, LEN EN, in the FRAMING_CFG_ADR register 0x10.

A length of zero is valid, and will transmit a packet with SOP, length, and CRC16 fields (if enabled), but no data field. Packet lengths of more than 16 bytes require that some data bytes be written after transmission of the packet has begun. Typically, length is updated prior to setting TX GO. It is written to the TX_LENGTH_ADR register 0x01.

7.4.1.5 Cyclic Redundancy Check

An optional CRC16 is performed on data to see if an error has occurred during the transmission of the packet. The CRC16 is performed on only the length and data fields of the packet. The CRC16 can be seeded with a user specified 16-bit value loaded into the CRC_SEED_LSB_ADR register 0x16 and CRC_SEED_MSB_ADR register 0x16. The RX Framer CRC16 check is not only against the seeded value, but may also be configured to simultaneously check the received data against the zero-seeded value. Received data that fails the CRC16 check is flagged with an error, but is still available to the application.

Multiple registers control how the CRC16 is used. Bit 2, DIS TXCRC, of the TX_OVERRIDE_ADR register 0x1F determines whether or not a CRC16 value is calculated and appended to the data packet. When set, no CRC16 is used. Bit 2, DIS RXCRC, of the RX_OVERRIDE_ADR register 0x1E determines whether or not the receive CRC checker is enabled. The user should ensure that DIS TXCRC and DIS RXCRC are matched on both sides of the RF link. Bit 3, DIS CRC0, of the RX_OVERRIDE_ADR causes the receiver to reject packets that match a zero CRC16 seed when set. Bit 1, ACE, of the RX_OVERRIDE_ADR allows the receiver to accept packets that do not match the CRC16 seed. Both of these last two bytes affect whether or not CRC16 errors are flagged on the received packet and Auto ACKs are sent (when enabled). If desired the CRC16 calculated results can be read at the TX_CRC_LSB_ADR register 0x16 and the TX_CRC_MSB_ADR register 0x18 on the transmit side and the RX_CRC_LSB_ADR register 0x19 and the RX_CRC_MSB_ADR register 0x20 on the receive side.

For CRC16 generation and checking, the shift registers in the receive and transmit framers are seeded with the specified pattern. For each data bit sent or received, the high order bit of the current remainder is XORed with the data bit and then the remainder is shifted left one bit and the low-order bit set to zero. If the result of that XOR is one, then the remainder is XORed with the generator polynomial. When the last bit of the checked field is sent, the CRC16 in the transmit framer is inverted and appended to the packet. When the last bit of the CRC16 is received by the receive framer and no errors have occurred, the remainder is equal to the polynomial residual. A CRC16 error exists if the computed checksum remainder at the end of a packet reception does not match the residual.

The CRC16 can be represented as a polynomial as shown below.

$$G(X) = X^{16} + X^{15} + X^{2} + 1$$

This method provides 100% coverage on single and double bit errors. It can also detect an odd number of bit errors and any error as wide as the CRC16 field itself.

Using a known CRC16 seed can be used to distinguish link data, and provides an additional method of co-locating systems. Allowing for the additional zero seed allows ‘new’ devices to connect to a host prior to being given a seed value.

A two stage pipeline before the receive FIFO is used to prevent the CRC16 from being written to the RX FIFO in the case where no length field is available and EOP detection is relied upon.

7.4.2 ACK Packets

ACK packets consist of preamble, SOP, and CRC16 (when enabled). In order for a transaction with an automated ACK response to complete successfully when CRC16 is enabled, the CRC in the received ACK must match the CRC transmitted in the last packet. The data mode of the CRC portion of a transmitted ACK must also match the data mode of the last received data packet.

Automatic transmission of ACK packets is enabled by setting bit 7, ACK EN, of the XACT_CFG_ADR register 0x0F.

7.4.3 End of Packet Detection

There are multiple methods of end of packet detection. These include length field detection and different variations of non-correlation.

The most reliable method is to use framed packets with the length field enabled. In this case, end of packet detection is explicitly based upon the number of data bytes specified in the length field plus the CRC16 (if enabled). Using the length field allows for a faster turn around. In rare cases, it is possible for the length field to be corrupted during transmission, thus resulting in an unplanned amount of data being received – either too short with the remainder of the legitimate data lost, or too long with random data filling the extra bytes. An erroneously long length field will also tie up the correlator for the amount of time required to receive a packet of the erroneous length.

If the length field is disabled, then the receiver must rely upon non-correlation for end of packet detection. There are two schemes for this: a fixed length of non-correlations (invalid bits) and a more sophisticated mode where after the ‘hint’ interval the received CRC is checked and if good, the EOP is detected at that point.

To use the fixed number of non-correlations the hint enable field must be cleared. This is done by setting bit 7, HEN, of the EOP_CTRL_ADR register 0x14 to ‘0’. Bits 3:0, EOP, of that register set the number of successive symbol non-correlations that are required to determine an EOP condition. The disadvantage of this mechanism is that the correlator must continue running past the end of the packet for the specified length of time before it can determine that an end of packet was previously reached. Therefore, the receive event is delayed for a longer time past the end of the packet.

When the hint enable field is set, the more sophisticated mechanism is used. An EOP is detected if no correlations have been detected for the configured number of symbol periods *and* the last two received bytes match the calculated CRC for all previously received bytes. The desired number of symbol periods is set using bits 6:4, HINT, of the EOP_CTRL_ADR register 0x14.

In the absence of chip errors, the hinting scheme causes an EOP to be detected on the very first invalid bit, thereby, shutting down the receiver quickly. For lower data rates this EOP delay would be shorter than that required to send a length field. However, there is a risk that a single invalid bit would cause a packet to be ‘truncated’ if it occurred at the right byte boundary where the data was such that the interim CRC value was also seen as good.

8. Radio Initialization and Operation

8.1 Introduction

The intent of the radio driver is to provide users with a consistent interface to the radio. The driver was designed to interface with both C and assembly written applications and consists of the following files:

Table 8-1. Radio Driver Files

File Name	Description
lpstreaming.asm	Core functions responsible for transmitting and receiving data in packets up to 254 bytes in length. This version of the driver manages the radio transmit and receive FIFO's by moving data in bursts of 8 bytes.
lpnonstreaming.asm	Core functions responsible for transmitting and receiving data in packets up to 16 bytes in length. This version of the driver is simpler and smaller than the streaming version.
Note: lpstreaming.asm and lpnonstreaming.asm implement the same API functions - only one of them should be used as part of a project.	
lpradio.asm	API functions primarily responsible for radio configuration and status and variable allocation.
lpradio.h	'C' function prototypes and type declarations for the entire radio driver - functions implemented in lpstreaming.asm or lpnon-streaming.asm, lpradio.asm and psocradio.asm or e2radio.asm.
lpradio.inc	Symbolic definitions used in lpstreaming.asm, lpnonstreaming.asm, lpradio.asm, psocradio.asm, and e2radio.asm.
lpregs.h	'C' defines for the radio registers and register fields.
lpregs.asm	Assembly symbolic definitions for the radio registers and register fields.
psocradio.asm	Low level radio interface routines provide SPI access for register read and write for single registers, file registers and bursts using the PSoC SPI interface.
e2radio.asm	Low level radio interface routines provide SPI access for register read and write for single registers, file registers and bursts using the enCoRe II SPI interface.
Note: psocradio.asm and e2radio.asm implement the same API functions - only one of them should be used as part of a project.	
irqmacros.inc	Support macros for M8C global IRQ management.
irqmacros.asm	Variable allocation for irqmacros.asm.

The radio driver supports various application and hardware requirements. Projects that use packets larger than 16 bytes must use lpstreaming.asm. Projects using packets less than or equal to 16 bytes should use lpnonstreaming.asm. The hardware specific SPI interface files *psocradio.asm* and *e2radio.asm* provide hardware SPI interface calls for the radio driver. *psocradio.asm* provides SPI communication for all PSoC based applications whereas *e2radio.asm* is to be used for enCoRe II based applications.

In addition to the SPI interface signals LP_nSS, SCK, MOSI, MISO, the radio driver also requires access to the IRQ output of the radio. This pin must be named LP_IRQ.

8.2 Usage

This section presents information on radio initialization and methods for transmitting and receiving. For the developing applications in LPstar, please use the radio driver from the application note-AN68105-Migration of WirelessUSB™ LP Designs to WirelessUSB™ LPstar.

8.2.1 Initialization

Before the radio can be used it must first be initialized using the API call RadioInit with parameters to be used for the XACT_CFG_ADR and TX_CFG_ADR registers. These two registers define the end state of the radio and the data mode to be used. Refer to the data sheet for specifics, see Cypress Semiconductor Support on page 9. Example usage:

```
RadioInit(ACK_EN|END_STATE_SLEEP|ACK_TO_15X,
DEFAULT_PA|DATCODE_LEN_32|DATMODE_8DR);
```

All other aspects of radio initialization and configuration, with the exception of XACT_CFG_ADR and TX_CFG_ADR are the responsibility of the application and must be done prior to starting a transmit or receive operation.

8.2.2 Transmitting

There are two types of transmit functions: blocking and non-blocking. Blocking does not return until the transmit process has completed. In the non-blocking case, the function starts the transmit and then returns. It is the responsibility of the calling application to monitor the stare of the transmit and terminate when necessary. Both forms of transmit require a call to RadioSetPtr with a buffer address as a parameter. This pointer points to the start of the buffer to be transmitted.

Table 8-2. Transmit Example

Blocking	Non-Blocking
RadioSetPtr	RadioSetPtr
RadioBlockingTransmit	
	RadioStartTransmit
	RadioGetTransmitState
	RadioEndTransmit

8.2.3 Receiving

By nature, receives are non-blocking and require a similar set of calls as was used in the non-blocking transmit case. In the event that a receive must be aborted then RadioAbort must be called. **Note** Once a receive is started, no calls can be made to the configuration access routines until the receive operation is terminated.

Table 8-3. Receive Example

Non-Blocking
RadioSetPtr
RadioStartReceive
RadioGetReceiveState
RadioEndReceive

8.3 Requirements

Radio header file requirements and hardware interface and pin requirement are discussed in this section.

8.3.1 Header Files

In order to use the radio driver you must include *lpradio.h* and *lpradio.inc* into any file that calls the radio driver functions.

8.3.2 Hardware Interface

The SPI block used to interface to the radio is named: SPIM_Radio. This block provides the firmware interface to the SPI pins, so their names have no requirements.

8.3.3 Pins

The pin connected to the radio's slave select pin must be called LP_nSS. The pin connected to the radio's IRQ pin must be named LP_IRQ.

8.4 Type Declarations

The type declarations are described below.

Table 8-4. Type Declarations

Type Declaration		Description
BYTE		Used for 8-bit register values
WORD		Used for 16-bit register values.
RADIO_CONST_PTR		Used for ROM buffer pointers.
RADIO_BUFFER_PTR		Used for RAM buffer pointers.
RADIO_LENGTH #define RADIO_ABORT_SUCCESS	 255	Used for radio field lengths.
RADIO_REG_ADDR		Used for Radio register addresses.
RADIO_STATE typedef unsigned char RADIO_STATE; #define RADIO_IDLE #define RADIO_RX #define RADIO_TX #define RADIO_DATA #define RADIO_COMPLETE #define RADIO_ERROR	 0x00 0x80 0x20 RXB1_IRQ RXC_IRQ RXE_IRQ	See the definition of RadioState below for more detail on the meanings of these bits.
RADIO_RX_STATUS typedef unsigned char RADIO_RX_STATUS; #define RADIO_RX_ACK #define RADIO_RX_MISS #define RADIO_RX_OF #define RADIO_RX_CRC0 #define RADIO_BAD_CRC #define RADIO_DATCODE_LEN #define RADIO_SDR #define RADIO_DDR #define RADIO_8DR #define RADIO_GFSK	 0x80 0x40 0x20 0x10 0x08 0x04 0x03 0x02 0x01 0x00	

8.5 Radio High Level Functions

These functions provide configuration, transmit, and receive functionality.

8.5.1 RadioInit

Description:

This function calls RadioReset then initializes the state of the radio as used by the radio driver. This includes setting the transmit offset to 1 MHz – the same as the receiver IF frequency to allow zero turn-around time for the fastest possible transitions between transmit and receive modes and minimal auto-ACK time. All other aspects of radio initialization and configuration, with the exception of XACT_CFG_ADR and TX_CFG_ADR, are the responsibility of the application and must be done prior to starting a transmit or receive operation.

Example:

```
RadioInit(ACK_EN | END_STATE_SLEEP | ACK_TO_15X,
          DEFAULT_PA | DATCODE_LEN_32 | DATMODE_8DR);
```

C Prototype:

```
void RadioInit(XACT_CONFIG xactConfig, TX_CONFIG txConfig);
```

Assembly:

```
A: xactConfig
X: txConfig
```

Parameters:

xactConfig: Specifies the end state of the radio and the data mode to be used.

txConfig: Specifies the data mode to be used.

Symbolic names provided in C and assembly, and their associated values, are given in the following table.

Symbolic Name	Value	Description

Return Value:

```
A: Undefined
X: Undefine
```

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

8.5.2 RadioSetChannel

Description:

This function sets the radio channel to a specified frequency. The frequency has the value of frequency + 2402 MHz. This function takes an 8-bit argument that is passed to it by BYTE channel.

Example:

```
RadioSetChannel(10); // Put carrier at 2.412GHz
```

C Prototype:

```
void RadioSetChannel(BYTE channel);
```

Assembly:

```
A: channel
X: Unused
```

Parameters:

channel: Specifies the channel used.

Return Value:

```
A: Undefined
X: Undefined
```

Side Effects:

```
The A and X registers may be modified by this or future implementations of this function.
The same is true for all RAM page pointer registers in the Large Memory Model. When nec-
essary, it is the calling function's responsibility to preserve the values across calls
to fastcall16 functions.
```

8.5.3 RadioSetFrequency

Description:

This function sets the frequency in MHz above 2.400 GHz. Example: 0 means 2.400 GHz, 83 means 2.483 GHz. This function sets the radio frequency to one where communication occurs. This function takes an 8-bit argument that is passed to it and writes it into the channel register.

The receiver has an Intermediate frequency of 1 MHz; that means that the PLL frequency and actual RF carrier frequency are 1 MHz apart. RadioInit sets the transmit offset at 1 MHz, which means that the transmit carrier is also 1 MHz higher than the PLL frequency that is programmed. This function compensates for the 1 MHz by decrementing the passed frequency so that the passed frequency is the carrier frequency, not the PLL frequency, which is 1 MHz lower.

Example:

```
RadioSetFrequency(10); // Put carrier at 2.410GHz
```

C Prototype:

```
void RadioSetFrequency(BYTE frequency);
```

Assembly:

```
A: frequency
X: Unused
```

Parameters:

frequency: Specifies the actual RF carrier frequency.

Return Value:

```
A: Undefined
X: Undefined
```

Side Effects:

```
The A and X registers may be modified by this or future implementations of this function.
The same is true for all RAM page pointer registers in the Large Memory Model. When nec-
essary, it is the calling function's responsibility to preserve the values across calls
to fastcall16 functions.
```

8.5.4 RadioGetChannel

Description:

This function gets the 8-bit value from the channel; that value is the frequency –2.

Example:

```
RadioSetFrequency(10); // Put carrier at 2.410GHz
c = RadioGetChannel(); // Returns 8.
```

C Prototype:

```
BYTE RadioGetChannel(void);
```

Assembly:

```
A: Unused
X: Unused
```

Parameters:

None.

Return Value:

```
A: channel
X: Undefined
```

Side Effects:

```
The A and X registers may be modified by this or future implementations of this function.
The same is true for all RAM page pointer registers in the Large Memory Model. When nec-
essary, it is the calling function's responsibility to preserve the values across calls
to fastcall16 functions.
```

8.5.5 RadioGetFrequency

Description:

This function returns the frequency value from the channel address in MHz above 2.4 GHz. Example: 0 means 2.400 GHz, 83 means 2.483 GHz.

C Prototype:

```
BYTE RadioGetFrequency(void);
```

Assembly:

```
A: Unused
X: Unused
```

Parameters:

None.

Return Value:

```
A: frequency
X: Undefined
```

Side Effects:

```
The A and X registers may be modified by this or future implementations of this function.
The same is true for all RAM page pointer registers in the Large Memory Model. When nec-
essary, it is the calling function's responsibility to preserve the values across calls
to fastcall16 functions.
```

8.5.6 RadioSetTxConfig

Description:

This function sets the transmitter configuration. The transmitter configuration is also set when initializing the radio by calling RadioInit. Both of these functions write the same register in the radio.

This example sets a 250 kbps transmit configuration using 32 chips per byte, 8DR encoding, and a transmit power of -15 dBm:

```
RadioSetTxConfig(DATCODE_LEN_32 | DATMODE_8DR | PA_N15_DBM);
```

C Prototype:

```
void RadioSetTxConfig(TX_CONFIG config);
```

Assembly:

```
A: Unused
X: Unused
```

Parameters:

config: Specifies the transmitter configuration.

Return Value:

```
A: Undefined
X: Undefined
```

Side Effects:

```
The A and X registers may be modified by this or future implementations of this function.
The same is true for all RAM page pointer registers in the Large Memory Model. When nec-
essary, it is the calling function's responsibility to preserve the values across calls
to fastcall16 functions.
```

8.5.7 RadioGetTxConfig

Description:

This function returns the transmitter configuration.

This example sets a transmit power of 0dBm:

```
RadioSetTxConfig((RadioGetTxConfig() & ~PA_VAL_MSK) | PA_0_DBM);
```

C Prototype:

```
TX_CONFIG RadioGetTxConfig(void);
```

Assembly:

```
A: Unused
X: Unused
```

Parameters:

None.

Return Value:

```
A: config
X: Undefined
```

Side Effects:

```
The A and X registers may be modified by this or future implementations of this function.
The same is true for all RAM page pointer registers in the Large Memory Model. When nec-
essary, it is the calling function's responsibility to preserve the values across calls
to fastcall16 functions.
```

8.5.8 RadioSetXactConfig

Description:

This function sets the transaction configuration. This function can be used to change the end state, the state the radio moves to after completing an operation, but cannot be used to force the radio into that new end state.

If you need to change the current state of the radio, for example from idle to sleep, follow the call to RadioSetXactConfig with a call to RadioAbort:

```
RadioSetXactConfig(ACK_EN | END_STATE_SLEEP | ACK_TO_8X);
RadioAbort();
```

C Prototype:

```
void RadioSetXactConfig(XACT_CONFIG config);
```

Assembly:

```
A: config
X: Unused
```

Parameters:

config: Specifies the value to be written to the XACT_CFG_ADR register.

Return Value:

```
A: Undefined
X: Undefined
```

Side Effects:

```
The A and X registers may be modified by this or future implementations of this function.
The same is true for all RAM page pointer registers in the Large Memory Model. When nec-
essary, it is the calling function's responsibility to preserve the values across calls
to fastcall16 functions.
```

8.5.9 RadioGetXactConfig

Description:

This function returns the transaction configuration register: XACT_CFG_ADR.

For example, to turn off the auto-ACK function:

RadioSetXactConfig(RadioGetXactConfig() & ~ ACK_EN);

C Prototype:

```
XACT_CONFIG RadioGetXactConfig(void);
```

Assembly:

```
A: Unused
X: Unused
```

Parameters:

None.

Return Value:

```
A: config
X: Undefined
```

Side Effects:

```
The A and X registers may be modified by this or future implementations of this function.
The same is true for all RAM page pointer registers in the Large Memory Model. When nec-
essary, it is the calling function's responsibility to preserve the values across calls
to fastcall16 functions.
```

8.5.10 RadioSetFrameConfig

Description:

This function sets the framing configuration value in the FRAMING_CFG_ADR register. For example, with this call the radio is set to framed, 64-chip SOP, with length fields and a SOP threshold of 8:

```
RadioSetFrameConfig(SOP_EN | SOP_LEN | LEN_EN | 8);
```

C Prototype:

```
void RadioSetFrameConfig(RADIO_FRAME_CONFIG config);
```

Assembly:

```
A: config
X: Unused
```

Parameters:

config: Value to be written to the FRAMING_CFG_ADR register.

Return Value:

```
A: Undefined
X: Undefined
```

Side Effects:

```
The A and X registers may be modified by this or future implementations of this function.
The same is true for all RAM page pointer registers in the Large Memory Model. When nec-
essary, it is the calling function's responsibility to preserve the values across calls
to fastcall16 functions.
```

8.5.11 RadioGetFrameConfig

Description:

This function returns the current framing configuration from the FRAMING_CFG_ADR register.

For example, to force the length field enabled:

```
RadioSetFrameConfig(RadioGetFrameConfig() | LEN_EN);
```

C Prototype:

```
RADIO_FRAME_CONFIG RadioGetFrameConfig(void);
```

Assembly:

```
A: Unused
X: Unused
```

Parameters:

None.

Return Value:

```
A: config
X: Undefined
```

Side Effects:

```
The A and X registers may be modified by this or future implementations of this function.
The same is true for all RAM page pointer registers in the Large Memory Model. When nec-
essary, it is the calling function's responsibility to preserve the values across calls
to fastcall16 functions.
```

8.5.12 RadioSetThreshold32

Description:

This function sets the data threshold for the 32-chip data modes.

Example:

```
RadioSetThreshold32(5);
```

C Prototype:

```
void RadioSetThreshold32(BYTE threshold);
```

Assembly:

```
A: threshold
X: Unused
```

Parameters:

threshold: 8-bit value that is passed to the function.

Return Value:

```
A: Undefined
X: Undefined
```

Side Effects:

```
The A and X registers may be modified by this or future implementations of this function.
The same is true for all RAM page pointer registers in the Large Memory Model. When nec-
essary, it is the calling function's responsibility to preserve the values across calls
to fastcall16 functions.
```

8.5.13 RadioGetThreshold32

Description:

This function returns the threshold for the 32-chip data modes.

C Prototype:

```
BYTE RadioGetThreshold32(void);
```

Assembly:

```
A: Unused
X: Unused
```

Parameters:

None.

Return Value:

```
A: threshold
X: Undefined
```

Side Effects:

```
The A and X registers may be modified by this or future implementations of this function.
The same is true for all RAM page pointer registers in the Large Memory Model. When nec-
essary, it is the calling function's responsibility to preserve the values across calls
to fastcall16 functions.
```

8.5.14 RadioSetThreshold64

Description:

This function sets the threshold for the 64-chip data modes.

C Prototype:

```
void RadioSetThreshold64(BYTE threshold);
```

Assembly:

```
A: threshold
X: Unused
```

Parameters:

threshold: 8-bit value passed to the function.

Return Value:

```
A: Undefined
X: Undefined
```

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

8.5.15 RadioGetThreshold64

Description:

This function returns the threshold for the 64-chip data modes. The threshold value is in the DATA64_THOLD_ADR register.

C Prototype:

```
BYTE RadioGetThreshold64(void);
```

Assembly:

```
A: Unused
X: Unused
```

Parameters:

None.

Return Value:

```
A: threshold
X: Undefined
```

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

8.5.16 RadioSetPreambleCount

Description:

This function sets the preamble repetition count. The preamble repetition count is the number of times the 16-chip preamble pattern is repeated prior to the SOP code for packets transmitted.

C Prototype:

```
void RadioSetPreambleCount(BYTE count);
```

Assembly:

```
A: count
X: Unused
```

Parameters:

count: 8-bit value that is passed to the function.

Return Value:

```
A: Undefined
X: Undefined
```

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

8.5.17 RadioGetPreambleCount

Description:

This function returns the preamble repetition count.

Example:

```
printf("The preamble is %d chips long.\r\n", RadioGetPreambleCount() * 16);
```

C Prototype:

```
BYTE RadioGetPreambleCount(void);
```

Assembly:

```
A: Unused
X: Unused
```

Parameters:

None.

Return Value:

```
A: count
X: Undefined
```

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

8.5.18 RadioSetPreamblePattern

Description:

This function sets the preamble pattern. The preamble pattern is transmitted at the chip rate, 1 µs per bit, prior to the SOP for all packets. The pattern is repeated the number of times specified by the preamble repetition count.

C Prototype:

```
void RadioSetPreamblePattern(WORD pattern);
```

Assembly:

```
A: pattern low order
X: pattern high order
```

Parameters:

pattern: Specifies the 16-bit pattern.

Return Value:

```
A: Undefined
X: Undefined
```

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

8.5.19 RadioGetPreamblePattern

Description:

This function returns the current 16-bit preamble pattern.

C Prototype:

```
WORD RadioGetPreamblePattern(void);
```

Assembly:

```
A: Unused
X: Unused
```

Parameters:

pattern: Specifies the 16-bit pattern.

Return Value:

```
A: pattern low order
X: pattern high order
```

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

8.5.20 RadioSetCrcSeed

Description:

This function sets the value used as the CRC seed-value for both transmit and receive. The CRC seed value of 0x0000 is special in that the receiver can correctly CRC check packets transmitted with a CRC seed of 0x0000 even if the receiver CRC seed is set to a different value.

This allows cross-network broadcasting in systems that use CRC seed as part of the system network identifier.

C Prototype:

```
void RadioSetCrcSeed(WORD crcSeed);
```

Assembly:

```
A: crcSeed low order
X: crcSeed high order
```

Parameters:

crcSeed: The crcSeed value.

Return Value:

```
A: Undefined
X: Undefined
```

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

8.5.21 RadioGetCrcSeed

Description:

This function returns the value used as the CRC seed-value for both transmit and receive.

C Prototype:

```
WORD RadioGetCrcSeed(void);
```

Assembly:

```
A: Unused
X: Unused
```

Parameters:

crcSeed: The crcSeed value.

Return Value:

```
A: crcSeed low order
X: crcSeed high order
```

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

8.5.22 RadioGetFuses

Description:

This function gets the fuse values from the radio. The six bytes of fuse data are stored at the buffer pointed to by the most recent call to RadioSetPtr.

Example:

```
unsigned char fuses[6];
RadioSetPtr(fuses);
RadioSetLength(sizeof(fuses));
RadioGetFuses();
```

C Prototype:

```
void RadioGetFuses(void);
```

Assembly:

```
A: Unused
X: Unused
```

Parameters:

None.

Return Value:

```
A: Undefined
X: Undefined
```

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

8.5.23 RadioGetRssi

Description:

This function returns the Receive Signal Strength Indicator value. When read after a receive has completed, this is the signal strength for the received packet. While in the receive state, but before a packet has been received, this function can be called continuously to check the carrier strength at the current frequency. When used in this continuous mode, the first value returned is garbage and should be discarded.

C Prototype:

```
RADIO_RSSI RadioGetRssi(void);
```

Assembly:

```
A: Unused
X: Unused
```

Parameters:

None.

Return Value:

```
A: Undefined
X: Undefined
```

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

8.5.24 RadioSetConstSopPnCode

Description:

This function sets the Start Of Packet PN Code. This function can be used to set arbitrary SOP PN codes. Great care should be taken in selecting PN Codes – poorly formed PN codes can substantially reduce or negate radio performance.

C Prototype:

```
void RadioSetConstSopPnCode(const BYTE *patternAddr);
```

Assembly:

```
A: patternAddr address high
X: patternAddr address low
```

Parameters:

patternAddr: Const (ROM) pointer to the 8 byte SOP PN code.

Return Value:

```
A: Undefined
X: Undefined
```

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

8.5.25 RadioSetConstDataPnCode

Description:

This function sets the data PN Code. This function can be used to set arbitrary Data PN codes. Great care should be taken in selecting PN Codes – poorly formed PN codes can substantially reduce or negate radio performance.

C Prototype:

```
void RadioSetConstDataPnCode(const BYTE *patternAddr);
```

Assembly:

```
A: patternAddr address high
X: patternAddr address low
```

Parameters:

patternAddr Const: Const (ROM) pointer to the 8-byte data PN code.

Return Value:

```
A: Undefined
X: Undefined
```

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

8.5.26 RadioSetSopPnCode

Description:

This function sets the start-of-packet PN code from a table of codes embedded in the radio driver. The PN code table in the driver contains 10 PN codes, with another 10 that can be added if wanted.

C Prototype:

```
void RadioSetSopPnCode(BYTE patternNum);
```

Assembly:

```
A: patternNum
X: Unused
```

Parameters:

patternNum: Array index for the 8-byte SOP PN code.

Return Value:

```
A: Undefined
X: Undefined
```

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

8.5.27 RadioStartTransmit

Description:

This function starts the non-blocking transmission of a packet. The location of the packet buffer to transmit must have previously been set with a call to RadioSetPtr.

After starting the transmission of a packet with this call, the state of the transmit operation must be checked by calling RadioGetTransmitState. When RadioGetTransmitState indicates that the transmission has completed a call must be made to RadioEndTransmit.

Note After calling RadioStartTransmit, no calls can be made to the configuration access routines until the transmit operation is terminated with a call to RadioEndTransmit or RadioAbort. Until one of those calls is made to end the transmit operation the only other call supported is RadioGetTransmitState.

Example:

```
unsigned char bufferToTransmit[14] = "PacketPayload";
RADIO_STATE txState;
RadioSetPtr(bufferToTransmit);
RadioStartTransmit(0, sizeof(bufferToTransmit));
while (!((txState = RadioGetTransmitState()) & RADIO_COMPLETE)) {
    CallSomeRoutineToGetWorkDoneWhileWaitingForTransmitToComplete();
    }
RadioEndTransmit();
if (txState & RADIO_ERROR) printf("Transmit failed.\r\n");
```

C Prototype:

```
void RadioStartTransmit(BYTE retryCount, RADIO_LENGTH length);
```

Assembly:

```
A: retryCount
X: length
```

Parameters:

retryCount: Number of times the packet should be retried it the transmit fails.

length:Length, in bytes, of the packet.

Return Value:

```
A: Undefined
X: Undefined
```

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

8.5.28 RadioGetTransmitState

Description:

This function returns the state of the current transmit operation. This call must be made after starting a transmit operation with the RadioStartTransmit function. The value returned is of the type RADIO_STATE which includes some bits maintained by the radio driver itself and some bits that are read from the TX_IRQ_STATUS_ADR register.

Although the bits in the status register in the hardware clear automatically, RadioEndState returns a status value that holds the status bits until RadioEndReceive is called.

C Prototype:

```
RADIO_STATE RadioGetTransmitState(void);
```

Assembly:

```
A: Unused
X: Unused
```

Parameters:

None.

Return Value:

```
A: RadioState
X: Undefined
```

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

8.5.29 RadioEndTransmit

Description:

This function completes a transmit operation.

C Prototype:

```
void RadioEndTransmit(void);
```

Assembly:

```
A: Unused
X: Unused
```

Parameters:

None.

Return Value:

```
A: Undefined
X: Undefined
```

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

8.5.30 RadioAbort

Description:

This function aborts a receive operation or forces the radio state while idle. RadioAbort must not be called while the radio is performing a transmit operation.

In the case where a receive operation is currently in progress it is possible that the receive operation completes (successfully or with an error) prior to the RadioAbort call. If this is the case the value returned is the length of a packet received without errors. If auto-ACK is enabled, the packet has been ACK'ed as well. Regardless of whether the RadioAbort call returns a packet (if called during a receive operation) when the call returns, the radio will be idle – or whatever the current end state is.

If the radio is not performing a receive operation, it always returns RADIO_ABORT_SUCCESS.

Example:

```
unsigned char receiveBuffer[16];
RADIO_LENGTH rxAbortStatus;
RadioSetPtr(receiveBuffer);
RadioSetLength(sizeof(receiveBuffer));
RadioStartReceive();
rxAbortStatus = RadioAbort();
if (rxAbortStatus != RADIO_ABORT_SUCCESS)
    printf("Received %d byte packet.\r\n", rxAbortStatus);
```

RadioAbort can also be used to force the radio into the current end state as specified in the transaction control register.

```
RadioSetXactConfig(ACK_EN | END_STATE_SLEEP | ACK_TO_8X);
RadioAbort();
```

C Prototype:

```
RADIO_LENGTH RadioAbort(void);
```

Assembly:

```
A: Unused
X: Unused
```

Parameters:

None.

Return Value:

```
A: length
X: Undefined
```

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

8.5.31 RadioBlockingTransmit

Description:

This function transmits a packet; it blocks execution until it completes. This function attempts to transmit a packet. The address of the packet buffer should have previously been set with a call to RadioSetPtr.

This routine gives the user very little control – probably less than most applications require. This function is primarily intended for very simple applications that have no use for a timeout.

Example:

```
unsigned char bufferToTransmit[14] = "PacketPayload";
RADIO_STATE txState;
RadioSetPtr(bufferToTransmit);
txState  = RadioBlockingTransmit(4, sizeof(bufferToTransmit));
if (txState & RADIO_ERROR)
   printf("Transmit failed after 5 attempts.\r\n");
```

C Prototype:

```
RADIO_STATE RadioBlockingTransmit(BYTE retryCount, RADIO_LENGTH length);
```

Assembly:

```
A: retryCount
X: length
```

Parameters:

retryCount: Number of times the packet should be retried it the transmit fails.

length:Length, in bytes, of the packet.

Return Value:

```
A: RadioState
X: Undefined
```

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

8.5.32 RadioStartReceive

Description:

This function starts the reception of a packet. The location and length of the packet buffer to receive the data into must have previously been set with calls to RadioSetPtr and RadioSetLength.

After starting the reception of a packet with this call, the state of the receive operation must be checked by calling RadioGetReceiveState. When RadioGetReceiveState indicates that the transmission has completed, a call must be made to RadioEndReceive.

After calling RadioStartReceive NO CALLS can be made to the configuration access routines until the receive operation is terminated with a call to RadioEndReceive or RadioAbort.

Until one of those calls is made to end the receive operation, the only other calls supported are RadioGetReceiveState and RadioGetRssi.

C Prototype:

```
void RadioStartReceive(void);
```

Assembly:

```
A: Unused
X: Unused
```

Parameters:

None.

Return Value:

```
A: Undefined
X: Undefined
```

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

8.5.33 RadioGetReceiveState

Description:

This function returns the state of the current receive operation. This call should be made after starting a receive operation with the RadioStartReceive function.

The value returned is of the type RADIO_STATE, which includes some bits maintained by the radio driver itself and some bits that are read from the RX_IRQ_STATUS_ADR register. Although the hardware bits clear automatically, the driver makes make them sticky until the RadioEndReceive.

Example:

```
RadioStartReceive(); // Start the receiver
while(1) {
    unsigned char ReceivedPayloadSize;
    RADIO_STATE rxState;
    RADIO_RX_STATUS rxStatus;

    rxState = RadioGetReceiveState();
    if (rxState & RADIO_COMPLETE) {
        ReceivedPayloadSize = RadioEndReceive();
        if (!(rxState & RADIO_ERROR)) {
            ++ReceivedCount; // Successfully received a packet
        }
        else {
           RxStatus = RadioGetReceiveStatus();
           // Handle various error types here.
        }
    }
```

C Prototype:

```
RADIO_STATE RadioGetReceiveState(void);
```

Assembly:

```
A: Unused
X: Unused
```

Parameters:

None.

Return Value:

```
A: RadioState
X: Undefined
```

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

8.5.34 RadioEndReceive

Description:

This function completes a receive operation; it returns the length of the packet that was received. If the packet payload truncation occurs (due to inadequate buffer length) it does not change this return value.

C Prototype:

```
RADIO_LENGTH RadioEndReceive(void);
```

Assembly:

```
A: Unused
X: Unused
```

Parameters:

None.

Return Value:

```
A: length
X: Undefined
```

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

8.5.35 RadioGetReceiveStatus

Description:

The status register includes information about the encoding of the most recently received packet and if there was an error, the cause of the error.

C Prototype:

```
RADIO_RX_STATUS RadioGetReceiveStatus(void);
```

Assembly:

```
A: Unused
X: Unused
```

Parameters:

None.

Return Value:

```
A: status
X: Undefined
```

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

8.5.36 RadioInterrupt

Description:

This function manages the radio in an ISR.

For interrupt-based systems RadioInterrupt can be called from the GPIO interrupt. This function does nothing if the IRQ is not asserted thereby making it easy to share the IRQ with other GPIO interrupts.

Using this function in an IRQ can eliminate latency caused by calling RadioGetTransmitState and/or RadioGetRecev-eiveState in a polling loop. When using this routine to maintain the radio, the state of that maintenance is communicated to the outside code through the global variable RadioState.

RadioInterrupt terminates with a RETI. It is intended to be the target of a JMP instruction from the interrupt vector directly in systems where the radio does not share the GPIO interrupt, or it can be JMP'd to at the end of an ISR that manages the other GPIO interrupt sources in a system.

Because RadioInterrupt is intended to be JMP'd to from the interrupt vector it leaves the registers unaffected.

Example (In boot.tpl / boot.asm):

```
    org   1Ch                     ;GPIO Interrupt Vector
    ljmp RadioInterrupt           ; `@INTERRUPT_7`
...elsewhere:
while (1) {
    if (RadioState & RADIO_RX) {
        if (RadioState & RADIO_COMPLETE) {
            if (!RadioState & RADIO_ERROR) {
                length = RadioEndReceive();
            }
        }
    } else if (RadioState & RADIO_TX) {
        if (RadioState & RADIO_COMPLETE) {
            if (!RadioState & RADIO_ERROR) {
                RadioEndTransmit();
            }
        }
    }
}
```

C Prototype:

```
void RadioInterrupt(void);
```

Assembly:

```
A: Unused
X: Unused
```

Parameters:

None.

Return Value:

```
A: Undefined
X: Undefined
```

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

8.5.37 RadioPoll

Description:

RadioPoll can be used in a polling loop to replace the use of RadioInterrupt in an interrupt handler. This allows most of the radio state management code to be the same as it is in the case where RadioInterrupt is being used. Note in the example below that the radio state management is the same as the RadioInterrupt example above. The only difference is the addition of the RadioPoll() call to replace the use of RadioInterrupt in an interrupt handler.

Example:

```
while (1) {
    RadioPoll();
    if (RadioState & RADIO_RX) {
        if (RadioState & RADIO_COMPLETE) {
            if (!RadioState & RADIO_ERROR) {
                length = RadioEndReceive();
            }
        }
    } else if (RadioState & RADIO_TX) {
        if (RadioState & RADIO_COMPLETE) {
            if (!RadioState & RADIO_ERROR) {
                RadioEndTransmit();
            }
        }
    }
}
```

C Prototype:

```
void RadioPoll(void);
```

Assembly:

```
A: Unused
X: Unused
```

Parameters:

None.

Return Value:

```
A: Undefined
X: Undefined
```

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

8.5.38 RadioState

Description:

RADIO_STATE is a global variable that contains the most recent return value from RadioGetReceiveState or RadioGetTransmitState. This value can be used in lieu of those return values; this is especially useful when using RadioInterrupt or RadioPoll to perform radio state management.

RadioState can even be used in lieu of the return values from RadioGetReceiveState or RadioGetTransmitState calls; for example, to save storage to contain a return value or to simplify code architecture.

The following are values for RadioState:

RADIO_IDLE	When the radio is not performing a transmit or receive operation SradioState is RADIO_IDLE.
RADIO_RX	When the radio is performing a receive operation, the RADIO_RX bit is set. This bit is set between calls to RadioStartReceive and RadioEndReceive.
RADIO_TX	When the radio is performing a receive operation, the RADIO_RX bit is set. This bit is set between calls to RadioStartTransmit and RadioEndTransmit.
RADIO_DATA	During a receive operation, this bit indicates that some data has been received. This bit is not set during a transmit operation.
RADIO_COMPLETE	This bit, when set indicates that the current transmit or receive operation is complete.
RADIO_ERROR	This bit, when set indicates that the current transmit or receive operation ended in an error. This bit is only set when the RADIO_COMPLETE bit is set.

C Prototype:

```
extern RADIO_STATE RadioState;
```

Assembly:

```
A: n/a
X: n/a
```

Parameters:

None.

Return Value:

```
A: RadioState
X: Undefined
```

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

8.6 Radio SPI Access Routines

These routines allow access directly to the radio registers. Most applications do not need to use these routines, instead relying on the functions provided by the radio driver's higher-level functions.

8.6.1 RadioReset

Description:

This routine resets the radio via MODE_OVERRIDE_ADR.RST=1.

C Prototype:

```
void RadioReset(void);
```

Assembly:

```
A: Unused
X: Unused
```

Parameters:

None.

Return Value:

```
A: Undefined
X: Undefined
```

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

8.6.2 RadioRead

Description:

This routine reads a single byte from a radio register. For both the 'C' and assembly call, the top two bits of the register address must be cleared.

C Prototype:

```
BYTE RadioRead(RADIO_REG_ADDR regAddr);
```

Assembly:

```
A: regAddr
X: Unused
```

Parameters:

None.

Return Value:

```
A: value
X: Undefined
```

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

8.6.3 RadioWrite

Description:

This routine writes a single byte to a radio register. For both the 'C' and assembly call, the top two bits of the register address must be cleared.

C Prototype:

```
void RadioWrite(RADIO_REG_ADDR regAddr, BYTE value);
```

Assembly:

```
A: regAddr
X: value
```

Parameters:

None.

Return Value:

```
A: Undefined
X: Undefined
```

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

8.6.4 RadioSetPtr

Description:

This routine sets the buffer pointer address for the RadioBurstRead, RadioFileRead, RadioBurstWrite, and RadioFile-Write functions.

C Prototype:

```
void RadioSetPtr(unsigned char ramPtr);
```

Assembly:

```
A: ramPtr
X: Unused
```

Parameters:

RamPtr. Pointer to RAM buffer for future operations.

Return Value:

```
A: Undefined
X: Undefined
```

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

8.6.5 RadioSetLength

Description:

This routine sets the buffer length used by RadioBurstRead and RadioFileRead functions. When the length of the buffer is exhausted, reads continue to occur but the data is thrown away.

This example reads only the first four bytes of fuse data:

```
unsigned char buffer[4];
RadioSetPtr(buffer);
RadioSetLength(sizeof(buffer));
RadioReadFuses();
```

C Prototype:

```
void RadioSetLength(RADIO_LENGTH length);
```

Assembly:

```
A: Unused
X: Unused
```

Parameters:

length:Length of buffer pointed to by most recent call to RadioSetPtr.

Return Value:

```
A: Undefined
X: Undefined
```

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

8.6.6 RadioBurstWrite

Description:

This routine writes a sequence of bytes to a sequence of radio registers. Prior to the RadioBurstWrite call, the developer must make a call to RadioSet Ptr with the address of the data buffer containing the data to write.

The following example writes register addresses 0-7 of the radio. For both the 'C' and assembly call, the top two bits of the register address must be cleared.

```
unsigned char buffer[8];
…later…
RadioSetPtr(buffer);
RadioBurstWrite(0, 8);
```

C Prototype:

```
void RadioBurstWrite(RADIO_REG_ADDR regAddr, BYTE cnt);
```

Assembly:

```
A: regAddr
X: cnt
```

Parameters:

regAddr: Address of buffer to write. The register address is incremented after each byte.

cnt: Length of the buffer.

Return Value:

```
A: Undefined
X: Undefined
```

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

8.6.7 RadioFileWrite

Description:

This routine writes a sequence of bytes to a single radio register. For both the 'C' and assembly call, the top two bits of the register address must be cleared. It must call RadioSetPtr prior to RadioFileWrite.

C Prototype:

```
void RadioFileWrite(RADIO_REG_ADDR regAddr, BYTE cnt);
```

Assembly:

```
A: regAddr
X: cnt
```

Parameters:

regAddr: Address of buffer to write.

cnt: Length of the buffer.

Return Value:

```
A: Undefined
X: Undefined
```

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

8.6.8 RadioBurstRead

Description:

This routine reads ‘cnt’ bytes from consecutive register addresses starting with address ‘regAddr’. The address and length of the buffer used to store the data need to be set with calls to RadioSetPtr and RadioSetLength prior to the call to RadioBurstRead. For both the 'C' and assembly call, the top two bits of ‘regAddr’ must be cleared.

In the following example 10 bytes are read from radio registers 0-9, but only the first 8 are stored in the buffer. The extra two bytes are thrown away:

```
unsigned char buffer[8];
RadioSetPtr(buffer);
RadioSetLength(sizeof(buffer));
RadioBurstRead(0, sizeof(buffer)+2);
```

C Prototype:

```
void RadioBurstRead(RADIO_REG_ADDR regAddr, BYTE cnt);
```

Assembly:

```
A: regAddr
X: cnt
```

Parameters:

regAddr: Address of buffer to read. The first register number to read. Subsequent reads come from incrementing register addresses.

cnt: Length of the buffer.

Return Value:

```
A: Undefined
X: Undefined
```

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

8.6.9 RadioFileRead

Description:

This routine reads a sequence of bytes from a single radio register. This function is the same as RadioBurstRead except that all the data is read from a single register and the address does not increment. For both the 'C' and assembly call, the top two bits of the register address must be cleared.

C Prototype:

```
void RadioFileRead(RADIO_REG_ADDR regAddr, BYTE cnt);
```

Assembly:

```
A: regAddr
X: cnt
```

Parameters:

```
A:  regAddr
X:  cnt
```

Return Value:

```
A: Undefined
X: Undefined
```

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the Large Memory Model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions.

9. Managing Power

9.1 Introduction

The WirelessUSB™ product family has been designed primarily for systems in which one node in the system is externally powered, and the other node or nodes are powered by batteries. The externally powered node (the Master) is therefore not particularly power sensitive, whereas minimizing power usage in the other nodes (the Slaves) is an important objective.

Many wireless networking solutions, particularly Frequency Hopping Spread Spectrum (FHSS) systems, require tight synchronization between nodes, resulting in a requirement for all nodes to use highly accurate oscillators, and to keep those oscillators running even when a node is not transmitting or receiving data. The WirelessUSB example protocols provided by Cypress have no such requirement, and are therefore able to achieve significantly lower average operating currents than synchronized systems.

Minimizing power consumption in WirelessUSB systems, and in particular battery powered slave devices, requires attention to a combination of system design and detailed firmware design considerations.

At a system level, the design should minimize the number of bits sent across the air, not necessarily in a single short interval of time, but overall across the life of the batteries.

At a detail level, the design should be optimized so that the Slave radio spends the minimum possible time in the high power modes, and the maximum possible time in Sleep mode, within the constraints imposed by the latency and throughput requirements of the system.

This chapter describes the various power saving modes of WirelessUSB (hereafter referred to as ‘the Radio’), and methods of using those modes to minimize power consumption in slave devices.

9.2 Power Advantages of DSSS

By taking advantage of the coding gain inherent in Direct Sequence Spread Spectrum (DSSS) systems, WirelessUSB systems achieve a very low Packet Error Rate (PER). Because retransmission on packet error is rare in WirelessUSB systems, the impact of occasional retransmissions on overall power consumption is minimal.

On the other hand, typical FSK and FHSS radio systems are subject to significant Bit Error Rates (BER), even in low interference environments. This necessitates frequent retransmission of data packets unless complex bandwidth consuming error correction schemes such as FEC are employed. Unless FEC is employed, the retransmission rate in an FSK system must be factored into power consumption calculations, and if so, the redundant data overhead needs to be considered. Packet error rates exceeding 10% are not uncommon in ISM band FSK radio systems, even in a low interference environment. In the high interference environment commonly found in the 2.4 GHz ISM band, retransmission rates of 50% or more have been observed in simple FSK radio systems.

The DSSS modulation of WirelessUSB enables reception of data packets with almost zero BER, even in environments where the error rate present in the raw symbols received is as high as 10%[1]. This is equivalent to an improvement in receive sensitivity and interference rejection of approximately 7 dB.

1. The relationship between BER and raw symbol (‘chip’) error rate (CER) is a function of the length of the spreading code and the receiver correlator thresholds selected.

9.3 General Principles

System design considerations play a key role in minimizing power consumption and therefore maximizing battery life in WirelessUSB slave designs.

In most cases, designers building WirelessUSB products and systems should be guided by the following principles:

- Transmit larger packets, less often. Although small by comparison with most networking protocols, WirelessUSB systems have a protocol overhead associated with transmitting a data packet. Therefore, transmitting two packets, each containing *n* bits of data takes more time in a high power mode, and therefore consumes more power overall than transmitting one packet containing *2n* bits. Consequently, to the maximum extent consistent with the latency requirements of the system, designers should seek to transmit data in the largest possible packets, and as infrequently as possible[2].
- Send the minimum amount of data. Many 'wired' protocols have unused or rarely changing data fields. One example of this is a typical USB keyboard report. This contains eight bytes, only one or two of which are non-zero in the vast majority of transmissions. To optimize power, wherever possible, designers should only send data that has changed, and avoid sending unchanged status data. In the keyboard example, this may cause the designer to use variable length packets, and only make a transmission when a new key was pressed or released. This packet may contain only data about the new key, rather than the state of other keys which had not changed since the last event.
- Compress data when possible. Many data fields in transmitted data packets are commonly encoded in such a way that more bits are transmitted than necessary. For example, information about the state of 16 flags is often transmitted as 16 bits, even though only a small number of flag setting combinations are valid. Power consumption can be minimized by transmitting the data values encoded into the smallest possible number of bits, which can then be decoded (for example using a lookup table) by the receiver.
- Leverage DSSS advantages. Designers familiar with simple ISM FSK radio systems may be accustomed to employing the error correction and/or detection encoding schemes with the substantial overhead necessary in such designs. However, because data errors (rather than erasures) are highly improbable in WirelessUSB systems, such schemes are unnecessary when using WirelessUSB. The payload data in a WirelessUSB data packet should simply be the data that needs to be transferred, unmodified by any redundant data encoding.
- Sweat the small stuff. Optimize firmware to ensure that the Radio is not in a high power mode for a microsecond longer than absolutely required. The example protocol firmware is designed to be readily understood and readily ported for use on other processors and in a wide variety of applications. Designers seeking to minimize power consumption in their application should customize the firmware for their own specific processor and application, so that it is optimized for power, paying particular attention to speed of firmware execution at the end of transmit and receive processing.

9.4 WirelessUSB Power Saving Modes

WirelessUSB devices have five main operating modes:

- Transmitting
- Receiving
- Synthesizer Settling
- Idle
- Power Down (also known as Sleep)

The power consumption values for these five modes can be found in the data sheet, see Cypress Semiconductor Support on page 9.

The Idle and Sleep modes offer the lowest current consumption, and are therefore together referred to as the power saving modes. In order to minimize power consumption in a WirelessUSB product the designer should seek to operate the WirelessUSB IC in these modes except when actively transmitting or receiving.

As there are timing considerations involved in transitioning between the operating modes of WirelessUSB devices, minimizing power consumption involves making careful trade-offs between the mode in use at a particular time, and the latency of data transmitted.

9.5 Two-Way System Overview

In a typical WirelessUSB system, 'the Master' is normally configured in receive mode, and 'the Slave' is normally in one of the power saving modes.

When the Slave is ready to transmit, it transitions from the low power mode into transmit mode, and sends data. After sending data, it transitions to receive mode, receives a handshake packet from the Master, and then transitions back into a power saving mode until the next transmission. (This process is described in more detail in the next section).

When the Master receives data, it waits for the end of the packet, and then transitions into transmit mode, transmits a handshake packet, and then immediately transitions back into receive mode, and waits for the next data packet.

This process is deliberately designed to minimize the power consumed by the Slaves, by shifting the power burden onto

2. The maximum payload length is 40 bytes.

the Master, which is typically not particularly power sensitive.

In some applications, the flow of data is from the Master to the Slaves; this can be accommodated using a variant of the procedure described above. In this case, rather than sending a data packet, the Slave device periodically polls the Master by sending a short ‘Data Request’ packet. If no data is available to send to the Slave, the Master responds with the standard handshake packet and when it receives this, the Slave transitions back into sleep mode. If data is available for the Slave, the Master transmits that data and then waits for a handshake from the Slave. When the Slave receives a data packet, it transmits a handshake packet back to the Master if the data is correctly received. If the data packet is not correctly received, no handshake packet is transmitted (or a NAK packet is transmitted), causing the Master to retransmit the data packet, ensuring that there is no loss of data.

9.6 Data Transmission Process

In two-way WirelessUSB systems, the Slave node typically follows the following procedure in order to transmit data:

1. Between transmissions, the Radio is in sleep mode, with its oscillator stopped.
2. When the Microcontroller (MCU) in the Slave needs to send data, it first sets the TX_GO bit in the TX_CTRL_ADR register. This causes the Radio to enter transaction mode. Transaction mode first instructs the oscillator to start. While the oscillator starts the MCU loads the TX_BUFFER_ADR with the desired payload data.
3. When the oscillator has stabilized to within 10 ppm of its final value, the synthesizer automatically starts.
4. When the synthesizer has settled, the transmitter automatically starts.
5. When the transmit completes, the Radio automatically transitions from transmit mode to receive mode in order to receive the ACK packet.
6. When the ACK packet completes, the Radio must briefly transition to idle mode.
7. The MCU may then configure the Radio to transition to sleep mode.

Figure 9-1. Packet Transmission Process

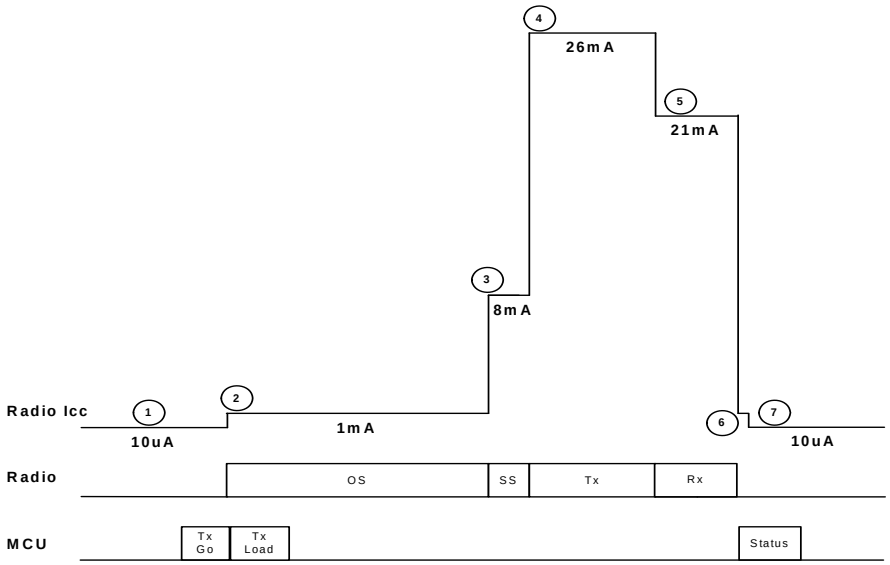


Figure 9-1 illustrates this process together with an indication of current consumption of the Radio at each stage[3].

This process is implemented in its entirety by the example protocols provided by Cypress. Designers choosing to use the protocols unmodified do not need to create firmware to perform this process. However, understanding of the process is important in managing power in WirelessUSB systems even when using the provided protocols unmodified.

9.7 Timing Considerations

WirelessUSB may be used in a wide variety of systems. Applications such as wireless sensors may require occasional low latency transmission of a small amount of data. Wireless HID applications may require the frequent trans-

3. Refer to the data sheet for precise values, see Cypress Semiconductor Support on page 9.

mission of data when the device is in use, but no transmissions for periods of hours or days when it is not.

In order to manage power most effectively in WirelessUSB systems, it is therefore necessary to understand the timing of transitions between the various operating modes, as well as the current consumption in each mode.

The timing values for these transitions may be found in the data sheet, see Cypress Semiconductor Support on page 9 .

9.8 Wakeup Timing

The wake time (oscillator stable) is influenced by two main factors – the Equivalent Series Resistance (ESR) of the crystal, and the capacitance with which the crystal has been designed to operate.

The wakeup times specified in the data sheet are for a 60 ohm ESR crystal, designed to work with a 10 pF load.

9.9 Synthesizer Start Timing

The time taken for the synthesizer to lock is variable, depending on the channel and whether or not the automatic calibration procedure needs to execute. An automatic calibration procedure is triggered after each write operation to the CHANNEL_ADR register. This procedure extends the next settling time, while subsequent settling times depends only on the channel used: 100 µs (fast), 180 µs (medium), and 270 µs (slow).

9.10 Preamble Timing

Before transmitting data, the Radio transmits a preamble, in order to assist the receiver in locking on to the transmitted radio signal. In the presence of other RF energy at the receiver (for example caused by other 2.4 GHz radio systems operating in the same area), 16 µs of preamble may occasionally be insufficient for the receiving radio to lock to the incoming WirelessUSB signal before the first data bit is transmitted.

The designer has a couple of options for dealing with this situation:

- Extend the pre-amble length by increasing the preamble length field in the PREAMBLE_ADR register. This is the least power-effective solution.
- Let retransmission take care of the error. The checksum enables any packet with a missing first bit to be detected as being in error, and the Master therefore does not send an ACK handshake. Consequently, the Slave retransmits the data packet. As the retransmission rate caused by this effect is typically around 0.3%, the additional time spent in higher power modes is less if 0.3% of packets are retransmitted than if all packets are extended by 16 µs of additional pre-amble.

For the most power-effective solution, aggressive applications may configure the preamble symbol length to ‘1’ (16 µs) or even ‘0’, if so desired.

9.11 Transmit and Receive Mode Timing

The Transmit and Receive modes are the highest power modes of the devices. Microcontroller firmware should therefore pay particular attention to spending the least possible amount of time in these modes, when in power-sensitive applications.

In order to minimize the amount of time spent in Transmit and Receive modes, it is recommended that full use is made of the Auto-ACK feature.

9.12 Debugging Tips

Even using an In-Circuit Emulator (ICE) and/or a logic analyzer connected to the SPI bus, it may not be possible to readily observe the power mode that the Radio is in during packet transmission, and therefore verifying that the timing is optimized may not be straightforward.

As the performance of any radio may be affected by ripple or noise on the Vcc supply to the IC, it is common design practice to minimize ripple and power supply noise at the radio by placing a small resistor (for example, 1 ohm) in series between the Radio IC Vcc and the power supply to the rest of the circuit. This is valid for WirelessUSB Radios as well as for any other radio system[4]. During debugging, this resistor can be used as a current sensing resistor, which can give a valuable real-time indication of the Radio operation. By connecting an isolated oscilloscope probe across the resistor, the current being consumed by the Radio can be observed. In the case of a 1 ohm resistor, each mV on the trace is equivalent to 1 mA of current consumption.

The designer can use this technique to analyze the amount of time the Radio spends in each mode during a data transmission, and verify that it is optimized to minimize overall power consumption.

4. It is generally not recommended that an inductor be used for this purpose in WirelessUSB applications, because the large changes in Radio current between the low and high power modes may cause ringing in the radio Vcc at the resonant frequency of the inductor and decoupling capacitors in response to step changes in the current the Radio is drawing.

10. Registers

10.1 Introduction

This section describes the WirelessUSB LP/LPstar registers. The registers are named according to the following conventions.

The general register format is **XXX_nnn_ADR**, where:

XXX_ is the operation.

nnn_ is the type.

- LENGTH, is the length of the data in either the transmit or receive buffers.
- CTRL, is a control register.
- CFG, is a configuration register.
- STATUS, is a status register.
- THOLD, is a PN Code threshold register.

_ADR is the suffix to uniquely identify all radio registers.

10.1.1 Example Register Formats

- TX_LENGTH_ADR is the length of the transmit data packet.

10.1.2 Other Conventions

EN Means enable.

OP Means output.

IP Means input.

VAL Means valid.

IRQEN Means interrupt request enable.

This chapter is a reference for all the WirelessUSB LP/LPstar device registers in address order.

10.2 Maneuvering Around the Registers

For ease-of-use, this chapter has been formatted so that there is one register per page, although some registers use two pages. On each page, from top to bottom, there are four sections:

1. Register name and address (from lowest to highest).
2. Register table showing the bit organization, with reserved bits grayed out.
3. Written description of register specifics or links to additional register information.
4. Detailed register bit descriptions.

10.2.1 Accessing Registers

All registers must be configured or accessed when the radio is in idle or sleep mode. Registers cannot be accessed during active transmit or receive modes.

However, the following registers can be configured or accessed during active transmit or receive mode.

- PWR_CTRL_ADR 0X0B
- IO_CFG_ADR 0x0D
- GPIO_CTRL_ADR 0x0E
- RSSI_ADR 0x13

10.3 Register Conventions

The following table lists the register conventions that are specific to this chapter.

Table 10-1. Register Conventions

Convention	Example	Description
'x' in a register name	ACBxxCR1	Multiple instances/address ranges of the same register
R	R : 00	Read register or bit(s)
W	W : 00	Write register or bit(s)
L	RL : 00	Logical register or bit(s)
C	RC : 00	Clearable register or bit(s)
00	RW : 00	Reset value is 0x00 or 00h
XX	RW : XX	Register is not reset
0,	0,04h	Register is in bank 0
1,	1,23h	Register is in bank 1
x,	x,F7h	Register exists in register bank 0 and register bank 1
Empty, grayed-out table cell		'Not Used' bit or group of bits, unless otherwise stated

10.4 Register Descriptions

All registers are read and writable, except where noted. Registers may be written to or read from either individually or in sequential groups.

Table 10-2. Register Map Summary

Address	Mnemonic	Access by Bit[1]	Access	Page
0x00	CHANNEL_ADR	-bbbbbbb	RW	106
0x01	TX_LENGTH_ADR	bbbbbbbb	RW	107
0x02	TX_CTRL_ADR	bbbbbbbb	RW	108
0x03	TX_CFG_ADR	--bbbbbb	RW	109
0x04	TX_IRQ_STATUS_ADR	rrrrrrrr	R	110
0x05	RX_CTRL_ADR	bbbbbbbb	RW	113
0x06	RX_CFG_ADR	bbbbb-bb	RW	114
0x07	RX_IRQ_STATUS_ADR	brrrrrrr	RW	115
0x08	RX_STATUS_ADR	rrrrrrrr	R	117
0x09	RX_COUNT_ADR	rrrrrrrr	R	118
0x0A	RX_LENGTH_ADR	rrrrrrrr	R	119
0x0B	PWR_CTRL_ADR	bbb-bbbb	RW	120
0x0C	XTAL_CTRL_ADR	bbb--bbb	RW	121
0x0D	IO_CFG_ADR	bbbbbbbb	RW	122
0x0E	GPIO_CTRL_ADR	bbbbrrrr	RW	123
0x0F	XACT_CFG_ADR	b-bbbbbb	RW	124
0x10	FRAMING_CFG_ADR	bbbbbbbb	RW	125
0x11	DATA32_THOLD_ADR	----bbbb	RW	126
0x12	DATA64_THOLD_ADR	---bbbbb	RW	127
0x13	RSSI_ADR	r-rrrrrr	R	128
0x14	EOP_CTRL_ADR	bbbbbbbb	RW	129
0x15	CRC_SEED_LSB_ADR	bbbbbbbb	RW	130
0x16	CRC_SEED_MSB_ADR	bbbbbbbb	RW	131
0x17	TX_CRC_LSB_ADR	rrrrrrrr	R	132
0x18	TX_CRC_MSB_ADR	rrrrrrrr	R	133
0x19	RX_CRC_LSB_ADR	rrrrrrrr	R	134
0x1A	RX_CRC_MSB_ADR	rrrrrrrr	R	135
0x1B	TX_OFFSET_LSB_ADR	bbbbbbbb	RW	136
0x1C	TX_OFFSET_MSB_ADR	----bbbb	RW	137
0x1D	MODE_OVERRIDE_ADR	wwwww--w	W	138
0x1E	RX_OVERRIDE_ADR	bbbbbbb-	RW	139
0x1F	TX_OVERRIDE_ADR	bbbbbbbb	RW	140
0x26	XTAL_CFG_ADR	wwwwwwww	W	141
0x27	CLK_OFFSET_ADR	wwwwwwww	W	142
0x28	CLK_EN_ADR	wwwwwwww	W	143
0x29	RX_ABORT_ADR	wwwwwwww	W	144
0x32	AUTO_CAL_TIME_ADR	wwwwwwww	W	145
0x35	AUTO_CAL_OFFSET_ADR	wwwwwwww	W	146
0x39	ANALOG_CTRL_ADR	wwwwwwww	W	147
Register Files				
0x20	TX_BUFFER_ADR	wwwwwwww	W	148
0x21	RX_BUFFER_ADR	rrrrrrrr	R	149
0x22	SOP_CODE_ADR[2]	bbbbbbbb	RW	150
0x23	DATA_CODE_ADR[3]	bbbbbbbb	RW	151
0x24	PREAMBLE_ADR[4]	bbbbbbbb	RW	152
0x25	MFG_ID_ADR	rrrrrrrr	R	153

Notes
1. b = read/write; r = read only; w = write only; '-' = not used, default value is undefined.
2. SOP_CODE_ADR default = 0x17FF9E213690C782.
3. DATA_CODE_ADR default = 0x02F9939702FA5CE3012BF1DB0132BE6F.
4. PREAMBLE_ADR default = 0x333302.

10.5 Register Details and Examples

The registers that follow are described in detail and listed one register per page.

10.5.1 CHANNEL_ADR

Channel Register

Individual Register Names and Addresses:

CHANNEL_ADR 0x00h

	7	6	5	4	3	2	1	0
Access : POR	R/W:x	R/W:1	R/W:0	R/W:0	R/W:1	R/W:0	R/W:0	R/W:0
Bit Name		Channel						

Bit	Name	Description
6:0	**Channel**	This field selects the channel. 0x00 sets 2400 MHz; 0x62 sets 2498 MHz. Values above 0x62 are not valid. The default channel is a fast channel above the frequency typically used in non-overlapping WiFi systems. Any write to this register impacts the time it takes the synthesizer to settle.

fast (100 µs) - 0 3 6 9 12 15 18 21 24 27 30 33 36 39 42 45 48 51 54 57 60 63 66 69 **72** 96
medium (180 µs) - 2 4 8 10 14 16 20 22 26 28 32 34 38 40 44 46 50 52 56 58 62 64 68 70 74 76 78 80 82 84 86 88 90 92 94
slow (270 µs) - 1 5 7 11 13 17 19 23 25 29 31 35 37 41 43 47 49 53 55 59 61 65 67 71 73 75 77 79 81 83 85 87 89 91 93 95 97

Note: Usable channels subject to regulation.

10.5.2 TX_LENGTH_ADR

Transmit Length Address Register

Individual Register Names and Addresses:

TX_LENGTH_ADR 0x01h

	7	6	5	4	3	2	1	0
Access : POR	R/W:0	R/W:0	R/W:0	R/W:0	R/W:0	R/W:0	R/W:0	R/W:0
Bit Name	TX Length							

Bit	Name	Description
7:0	**TX Length**	This register sets the length of the packet to be transmitted. A length of zero is valid, and a packet with SOP, length, and CRC16 fields (if enabled) is transmitted, but no data field. Packet lengths of more than 16 bytes require that some data bytes be written after transmission of the packet has begun. Typically, length is updated prior to setting TX GO. The maximum packet length for all packets is 40 bytes except for framed 64-chip DDR where the maximum packet length is 16 bytes.

Note: Maximum packet length is limited by the delta between the transmitter and receiver crystals of 60 ppm or better.

10.5.3 TX_CTRL_ADR

Transmit Control Address Register

Individual Register Names and Addresses:

TX_CTRL_ADR 0x02h

	7	6	5	4	3	2	1	0
Access : POR	R/W:0	R/W:0	R/W:	R/W:0	R/W:0	R/W:0	R/W:1	R/W:1
Bit Name	TX GO	TX CLR	TXB15 IRQEN	TXB8 IRQEN	TXB0 IRQEN	TXBERR IRQEN	TXC IRQEN	TXE IRQEN

Bit	Name	Description
7	**TX GO**	Start Transmission. Setting this bit triggers the transmission of a packet. Writing a 0 to this flag has no effect. This bit is cleared automatically at the end of packet transmission. The transmit buffer may be loaded either before or after setting this bit. If data is loaded after setting this bit, the length of time available to load the buffer depends on the starting state (sleep, idle or synth), the length of the SOP code, the length of preamble, and the packet data rate. For example, if starting from idle mode on a fast channel in 8DR mode with 32-chip SOP codes, the time available is 100 µs (synth start) + 32 µs (preamble) + 64 µs (SOP length) + 32 µs (length byte) = 228 µs. If there are no bytes in the TX buffer at the end of transmission of the length field, a TXBERR IRQ occurs and transmission aborts.
6	**TX CLR**	Clear TX Buffer. Writing a 1 to this register clears the transmit buffer. Writing a 0 to this bit has no effect. The previous packet (16 bytes or fewer) may be retransmitted by setting TX GO and not setting this bit. A new transmit packet may be loaded and transmitted without setting this bit if TX GO is set after the new packet is loaded to the buffer. If a new transmit packet is to be loaded before/after the TX GO bit has been set, then this bit should be set before loading a new transmit packet to the buffer.
5	**TXB15 IRQEN**	Buffer Not Full Interrupt Enable. See section 10.5.5 TX_IRQ_STATUS_ADR on page 110 for description.
4	**TXB8 IRQEN**	Buffer Half Empty Interrupt Enable. See section 10.5.5 TX_IRQ_STATUS_ADR on page 110 for description.
3	**TXB0 IRQEN**	Buffer Empty Interrupt Enable. See section 10.5.5 TX_IRQ_STATUS_ADR on page 110 for description.
2	**TXBERR IRQEN**	Buffer Error Interrupt Enable. See section 10.5.5 TX_IRQ_STATUS_ADR on page 110 for description.
1	**TXC IRQEN**	Transmission Complete Interrupt Enable. See section 10.5.5 TX_IRQ_STATUS_ADR on page 110 for description. TXC IRQEN and TXE IRQEN must be set together.
0	**TXE IRQEN**	Transmit Error Interrupt Enable. See section 10.5.5 TX_IRQ_STATUS_ADR on page 110 for description. TXC IRQEN and TXE IRQEN must be set together.

10.5.4 TX_CFG_ADR

Register

Individual Register Names and Addresses:

TX_CFG_ADR: 0x03h

LP	7	6	5	4	3	2	1	0
Access : POR	R/W:x	R/W:x	R/W:0	R/W:0	R/W:0	R/W:1	R/W:0	R/W:1
Bit Name			Data Code Length	Data Mode		PA Setting		

LPstar	7	6	5	4	3	2	1	0
Access : POR	R/W:x	R/W:x	R/W:0	R/W:0	R/W:0	R/W:1	R/W:0	R/W:1
Bit Name			Data Code Length	RSVD	Data Mode	PA Setting		

Bit	Name	Description
5	**Data Code Length**	Data Code Length. This bit selects the length of the DATA_CODE_ADR code for the data portion of the packet. This bit is ignored when the data mode is set to GFSK. 1 = 64-chip codes 0 = 32-chip codes
4:3	**Data Mode**	Data Mode. This field sets the data transmission mode. For LPstar, only bit 3 is used. 00 = 1-Mbps GFSK. 01 = 8DR Mode 10 = DDR Mode 11 = SDR Mode. It is recommended that firmware sets the ALL SLOW bit in register ANALOG_CTRL_ADR when using GFSK data rate mode.
2:0	**PA Setting**	PA Setting. This field sets the transmit signal strength. For LPstar, the max set is 6= 0 dBm. 0 = –35 dBm 1 = –30 dBm 2 = –24 dBm 3 = –18 dBm 4 = –13 dBm 5 = –5 dBm 6 = 0 dBm 7 = +4 dBm

10.5.5 TX_IRQ_STATUS_ADR

Register

Individual Register Names and Addresses:

TX_IRQ_STATUS_ADR 0x04h

LP	7	6	5	4	3	2	1	0
Access : POR	R/W:1	R/W:0	R/W:1	R/W:1	R/W:1	R/W:0	R/W:0	R/W:0
Bit Name	OS IRQ	LV IRQ	TXB15 IRQ	TXB8 IRQ	TXB0 IRQ	TXBERR IRQ	TXC IRQ	TXE IRQ

The state of all IRQ status bits is valid regardless of whether or not the IRQ is enabled. The IRQ output of the device is in its active state whenever one or more bits in this register is set and the corresponding IRQ enable bit is also set. Status bits are non-atomic (different flags may change value at different times in response to a single event).

.

LPstar	7	6	5	4	3	2	1	0
Access : POR	R/W:1	R/W:0	R/W:1	R/W:1	R/W:1	R/W:0	R/W:0	R/W:0
Bit Name	OS IRQ	RSVD	TXB15 IRQ	TXB8 IRQ	TXB0 IRQ	TXBERR IRQ	TXC IRQ	TXE IRQ

Bit	Name	Description
7	**OS IRQ**	Oscillator Stable IRQ Status. This bit is set when the internal crystal oscillator has settled (synthesizer sequence starts).
6	**LV IRQ**	Low Voltage Interrupt Status. This bit is set when the voltage on V_{BAT} is below the LVI threshold (see "PWR_CTRL_ADR" on page 119). This interrupt is automatically disabled whenever the PMU is disabled. When enabled, this bit reflects the voltage on V_{BAT}. For LPstar, this bit is RSVD.
5	**TXB15 IRQ**	Buffer Not Full Interrupt Status. This bit is set whenever there are 15 or fewer bytes remaining in the transmit buffer.
4	**TXB8 IRQ**	Buffer Half Empty Interrupt Status. This bit is set whenever there are eight or fewer bytes remaining in the transmit buffer.
3	**TXB0 IRQ**	Buffer Empty Interrupt Status. This bit is set at any time the transmit buffer is empty.
2	**TXBERR IRQ**	Buffer Error Interrupt Status. This IRQ is triggered by either of two events: (1) When the transmit buffer (TX_BUFFER_ADR) is empty and the number of bytes remaining to be transmitted is greater than zero. (2) When a byte is written to the transmit buffer and the buffer is already full. This IRQ is cleared by setting bit the TX CLR in TX_CTRL_ADR.
1	**TXC IRQ**	Transmission Complete Interrupt Status. This IRQ is triggered when transmission is complete. If transaction mode is not enabled, then this interrupt is triggered immediately after transmission of the last bit of the CRC16. If transaction mode is enabled, this interrupt is triggered at the end of a transaction. Reading this register clears this bit. TXC IRQ and TXE IRQ flags may change value at different times in response to a single event. If transaction mode is enabled and the first read of this register returns TXC IRQ=1 and TXE IRQ=0, then firmware must execute a second read to this register to determine if an error occurred by examining the status of TXE. There can be a case when this bit is not triggered when ACK EN = 1 and there is an error in transmission. If the first read of this register returns TXC IRQ = 1 and TXE IRQ = 1, then the firmware must not execute a second read to this reg-

		ister for a given transaction. If an ACK is received, RXC IRQ and RXE IRQ may be asserted instead of TXC IRQ and TXE IRQ.
0	**TXE IRQ**	Transmit Error Interrupt Status. This IRQ is triggered when there is an error in transmission. This interrupt is only applicable to transaction mode. It is triggered whenever no valid ACK packet is received within the ACK timeout period. Reading this register clears this bit.

10.5.6 RX_CTRL_ADR

Register

Individual Register Names and Addresses:

RX_CTRL_ADR : 0x05h

	7	6	5	4	3	2	1	0
Access : POR	R/W:0	R/W:0	R/W:0	R/W:0	R/W:0	R/W:1	R/W:1	R/W:1
Bit Name	RX GO	RSVD	RXB16 IRQEN	RXB8 IRQEN	RXB1 IRQEN	RXBERR IRQEN	RXC IRQEN	RXE IRQEN

Bit	Name	Description
7	**RX GO**	Start Receive. Setting this bit causes the device to transition to receive mode. If necessary, the crystal oscillator and synthesizer start automatically after this bit is set. Firmware must never clear this bit. This bit must not be set until after it self clears. The recommended method to exit receive mode when an error has occurred is to force END STATE and then dummy read all RX_COUNT_ADR bytes from RX_BUFFER_ADR or poll RSSI_ADR.SOP (bit 7) until set. See “XACT_CFG_ADR” on page 124 and “RX_ABORT_ADR” on page 144 for description.
6	**RSVD**	Reserved. Must be zero.
5	**RXB16 IRQEN**	Buffer Full Interrupt Enable. See “RX_IRQ_STATUS_ADR” on page 115 for description.
4	**RXB8 IRQEN**	Buffer Half Empty Interrupt Enable. See “RX_IRQ_STATUS_ADR” on page 115 for description.
3	**RXB1 IRQEN**	Buffer Not Empty Interrupt Enable. See“RX_IRQ_STATUS_ADR” on page 115 for description.
2	**RXBERR IRQEN**	Buffer Error Interrupt Enable. See “RX_IRQ_STATUS_ADR” on page 115 for description.
1	**RXC IRQEN**	Packet Reception Complete Interrupt Enable. See“RX_IRQ_STATUS_ADR” on page 115 for description.
0	**RXE IRQEN**	Receive Error Interrupt Enable. See“RX_IRQ_STATUS_ADR” on page 115 for description.

10.5.7 RX_CFG_ADR

Register

Individual Register Names and Addresses:

RX_CFG_ADR : 0x06h

	7	6	5	4	3	2	1	0
Access : POR	R/W:1	R/W:0	R/W:0	R/W:1	R/W:0	R/W:x	R/W:R/W:1	0
Bit Name	AGC EN	LNA	ATT	HILO	FAST TURN EN		RXOW EN	VLD EN

Status bits are non-atomic (different flags may change value at different times in response to a single event).

Bit	Name	Description
7	**AGC EN**	Automatic Gain Control (AGC) Enable. When this bit is set, AGC is enabled, and the LNA is controlled by the AGC circuit. When this bit is cleared, the LNA is controlled manually using the LNA bit. Typical applications clear this bit during initialization. It is recommended that this bit be disabled and bit 6 (LNA) be enabled unless the device is used in a system where it may receive data from a device using an external PA to transmit signals at >+4 dBm.
6	**LNA**	Low Noise Amplifier (LNA) Manual Control. When AGC EN (Bit 7) is cleared, this bit controls the state of the receiver LNA; when AGC EN is set, this bit has no effect. Setting this bit enables the LNA; clearing this bit disables the LNA. Device current in receive mode is slightly lower when the LNA is disabled. Typical applications set this bit during initialization.
5	**ATT**	Receive Attenuator Enable. Setting this bit enables the receiver attenuator. The receiver attenuator may be used to de-sensitize the receiver so that only very strong signals may be received. This bit should only be set when the AGC EN is disabled and the LNA is manually disabled.
4	**HILO**	HILO. When FAST TURN EN is set, this bit is used to select whether the device uses the high frequency for the channel selected, or the low frequency: 1 = high; 0 = low. When FAST TURN EN is not enabled this also controls the HILO bit to the receiver and must be left at the default value of ‘1’ for high side receive injection. Typical applications clear this bit during initialization.
3	**FAST TURN EN**	Fast Turn Mode Enable. When this bit is set, the HILO bit determines whether the device receives data transmitted 1 MHz above the RX Synthesizer frequency or 1 MHz below the receiver synthesizer frequency. Use of this mode allows for very fast turn-around, because the same synthesizer frequency may be used for both transmit and receive, thus eliminating the synthesizer re-settling period between transmit and receive. Note that when this bit is set, and the HILO bit is cleared, received data bits are automatically inverted to compensate for the inversion of data received on the ‘image’ frequency. Typical applications set this bit during initialization.
1	**RXOW EN**	Overwrite Enable. When this bit is set, if an SOP is detected while the receive buffer is not empty, then the existing contents of receive buffer are lost, and the new packet is loaded into the receive buffer. When this bit is set, the RXOW IRQ is enabled. If this bit is cleared, then the receive buffer may not be over-written by a new packet, and whenever the receive buffer is not empty SOP conditions are ignored, and it is not possible to receive data until the previously received packet has been completely read from the receive buffer.
0	**VLD EN**	Valid Flag Enable. When this bit is set, the receive buffer can store only eight bytes of data interleaved with valids (data0, valids0, data1, valids1,...). Typically, this bit is set only when interoperability with first generation devices is desired. See “RX_BUFFER_ADR” on page 149 for more detail.

10.5.8 RX_IRQ_STATUS_ADR

Register

Individual Register Names and Addresses:

RX_IRQ_STATUS_ADR : 0x07h

	7	6	5	4	3	2	1	0
Access : POR	R/W:x	R:x	R:x	R:x	R:x	R:x	R:x	R:x
Bit Name	RXOW IRQ	SOPDET IRQ	RXB16 IRQ	RXB8 IRQ	RXB1 IRQ	RXBERR IRQ	RXC IRQ	RXE IRQ

The state of all IRQ Status bits is valid regardless of whether or not the IRQ is enabled. The IRQ output of the device is in its active state whenever one or more bits in this register is set and the corresponding IRQ enable bit is also set. Status bits are non-atomic (different flags may change value at different times in response to a single event).

Bit	Name	Description
7	**RXOW IRQ**	Receive Overwrite Interrupt Status. This IRQ is triggered when the receive buffer is over-written by a packet being received before the previous packet has been read from the buffer. This bit is cleared by writing any value to this register. This condition is only possible when the RXOW EN bit in RX_CFG_ADR is set. This bit must be written ‘1’ by firmware before the new packet may be read from the receive buffer.
6	**SOPDET IRQ**	Start of packet detect. This bit is set whenever the start of packet symbol is detected.
5	**RXB16 IRQ**	Receive Buffer Full Interrupt Status. This bit is set whenever the receive buffer is full, and cleared otherwise.
4	**RXB8 IRQ**	Receive Buffer Half Full Interrupt Status. This bit is set whenever there are eight or more bytes remaining in the receive buffer. Firmware must read exactly eight bytes when reading RXB8 IRQ.
3	**RXB1 IRQ**	Receive Buffer Not Empty Interrupt Status. This bit is set at any time that there are one or more bytes in the receive buffer, and cleared when the receive buffer is empty. It is possible, in rare cases, that the last byte of a packet may remain in the buffer even though the RXB1_IRQ flag has cleared. This can ONLY happen on the last byte of a packet and only if the packet data is being read out of the buffer while the packet is still being received. The flag is trustworthy under all other conditions, and for all bytes prior to the last. When using RXB1_IRQ and unloading the packet data during reception, the user should be sure to check the RX_COUNT_ADR value after the RXC IRQ/RXE IRQ is set and unload the last remaining byte if the number of bytes unloaded is less than the reported count, even though the RXB1_IRQ is not set.
2	**RXBERR IRQ**	Receive Buffer Error Interrupt Status. This IRQ is triggered in one of two ways: (1) When the receive buffer is empty and there is an attempt to read data. (2) When the receive buffer is full and more data is received. This flag is cleared when RX GO is set and an SOP is received.

(continued on next page)

10.5.8 RX_IRQ_STATUS_ADR (continued)

1	**RXC IRQ**	Packet Receive Complete Interrupt Status. This IRQ is triggered when a packet has been received. If transaction mode is enabled, then this bit is not set until after transmission of the ACK. If transaction mode is not enabled, then this bit is set as soon as a valid packet is received. This bit is cleared when this register is read. RXC IRQ and RXE IRQ flags may change value at different times in response to a single event. There are cases when this bit is not triggered when ACK EN = 1 and there is an error in reception. Therefore, firmware should examine RXC IRQ, RXE IRQ, and CRC 0 to determine receive status. If the first read of this register returns RXC IRQ = 1 and RXE IRQ = 0, then firmware must execute a second read to this register to determine if an error occurred by examining the status of RXE IRQ. If the first read of this register returns RXC IRQ = 1 and RXE IRQ = 1, then the firmware must not execute a second read to this register for a given transaction.
0	**RXE IRQ**	Receive Error Interrupt Status. This IRQ is triggered when there is an error in reception. It is triggered whenever a packet is received with a bad CRC16, an unexpected EOP is detected, a packet type (data or ACK) mismatch, or a packet is dropped because the receive buffer is still not empty when the next packet starts. The exact cause of the error may be determined by reading RX_STATUS_ADR. This bit is cleared when this register is read.

10.5.9 RX_STATUS_ADR

Register

Individual Register Names and Addresses:

RX_STATUS_ADR : 0x08h .

	7	6	5	4	3	2	1	0
Access : POR	R/W:0	R/W:0	R/W:0	R/W:0	R/W:1	R/W:x	R/W:x	
Bit Name	RX ACK	PKT ERR	EOP ERR	CRC0	Bad CRC	RX Code	RX Data Mode	

It is expected that firmware does not read this register until after TX GO self clears. Status bits are non-atomic (different flags may change value at different times in response to a single event).

Bit	Name	Description
7	**RX ACK**	RX Packet Type. This bit is set when the received packet is an ACK packet, and cleared when the received packet is a standard packet.
6	**PKT ERR**	Receive Packet Type Error. This bit is set when the packet type received is what not was expected and cleared when the packet type received was as expected. For example, if a data packet is expected and an ACK is received, this bit is set.
5	**EOP ERR**	Unexpected EOP. This bit is set when an EOP is detected before the expected data length and CRC16 fields have been received. This bit is cleared when SOP pattern for the next packet has been received. This includes the case where there are invalid bits detected in the length field and the length field is forced to 0.
4	**CRC0**	Zero-seed CRC16. This bit is set whenever the CRC16 of the last received packet has a zero seed.
3	**Bad CRC**	Bad CRC16. This bit is set when the CRC16 of the last received packet is incorrect.
2	**RX Code**	Receive Code Length. This bit indicates the DATA_CODE_ADR code length used in the last correctly received packet. 1 = 64-chip code 0 = 32-chip code
1:0	**RX Data Mode**	Receive Data Mode. These bits indicate the data mode of the last correctly received packet. 00 = 1-Mbps GFSK 01 = 8DR 10 = DDR 11 = Not Valid These bits do not apply to unframed packets.

10.5.10 RX_COUNT_ADR

Register

Individual Register Names and Addresses:

RX_COUNT_ADR : 0x09h

	7	6	5	4	3	2	1	0
Access : POR	R/W:0							
Bit Name	RX Count							

Count bits are non-atomic (updated at different times).

Bit	Name	Description
7:0	**RX Count**	This register contains the total number of payload bytes received during reception of the current packet. After packet reception is complete, this register matches the value in RX_LENGTH_ADR unless there was a packet error. This register is cleared when RX_LENGTH_ADR is automatically loaded, if length is enabled, after the SOP. Count must not be read when RX_GO=1 during a transaction.

10.5.11 RX_LENGTH_ADR

Register

Individual Register Names and Addresses:

RX_LENGTH_ADR : 0x0Ah

	7	6	5	4	3	2	1	0
Access : POR	R/W:0							
Bit Name	RX Length							

Length bits are non-atomic (different flags may change value at different times in response to a single event).

Bit	Name	Description
7:0	**RX Length**	This register contains the length field which is updated with the reception of a new length field (shortly after start of packet detected). If there is an error in the received length field, 0x00 is loaded instead, except when using GFSK data rate, and an error is flagged.

10.5.12 PWR_CTRL_ADR

Register

Individual Register Names and Addresses:

PWR_CTRL_ADR : 0x0Bh

LP	7	6	5	4	3	2	1	0
Access : POR	R/W:1	R/W:0	R/W:1	R/W:x	R/W:0		R/W:0	
Bit Name	PMU EN	LVIRQ EN	PMU SEN	PFET DIS	LVI TH		PMU OUTV	

LPstar	7	6	5	4	3	2	1	0
Access : POR	R/W:1	R/W:0	R/W:1	R/W:x	R/W:0		R/W:0	
Bit Name	*The firmware should set "00010000" to this register while initiating*							

Bit	Name	Description
7	**PMU EN**	Power Management Unit (PMU) Enable. Setting this bit enables the PMU. When the PMU is disabled, or if the PMU is enabled and the V_{BAT} voltage is above the value set in Bits 1:0 of this register, the V_{REG} pin is internally connected to the V_{BAT} pin. if the PMU is enabled and the V_{BAT} voltage is below the value set by PMU OUTV, then the PMU boosts the V_{REG} pin to a voltage not less than the value set by PMUOP.
6	**LVIRQ EN**	Low Voltage Interrupt Enable. Setting this bit enables the LV IRQ interrupt. When this interrupt is enabled, if the V_{BAT} voltage falls below the threshold set by LVI TH, then a low voltage interrupt is generated. The LVI is not available when the device is in sleep mode. The LVI event on IRQ pin is automatically disabled whenever the PMU is disabled.
5	**PMU SEN**	PMU Sleep Mode Enable. If this bit is set, the PMU continues to operate normally when the device is in sleep mode. If this bit is not set, then the PMU is disabled when the device is in sleep mode. In this case, if V_{BAT} is below the PMU OUTV voltage and PMU EN is set, when the device enters sleep mode the V_{REG} voltage falls to the V_{BAT} voltage as the V_{REG} capacitors discharge.
4	**PFET DIS**	Setting this bit to '1' disables the FET, thereby safely allowing V_{BAT} to be connected to a separate reference from Vcc when the PMU is disabled to the radio.
3:2	**LVI TH**	Low Voltage Interrupt Threshold. This field sets the voltage on V_{BAT} at which the LVI is triggered. 11 = 1.8V 10 = 2.0V 01 = 2.2V 00 = PMU OUTV voltage
1:0	**PMU OUTV**	PMU Output Voltage. This field sets the minimum output voltage of the PMU. 11 = 2.4V 10 = 2.5V 01 = 2.6V 00 = 2.7V. When the PMU is active, the voltage output by the PMU on V_{REG} is never less than this voltage provided that the total load on the V_{REG} pin is less than the specified maximum value, and the voltage in V_{BAT} is greater than the specified minimum value.

10.5.13 XTAL_CTRL_ADR

Register

Individual Register Names and Addresses:

XTAL_CTRL_ADR : 0x0Ch

	7	6	5	4	3	2	1	0
Access : POR	R/W:0	R/W:0	R/W:0	R/W:x		R/W:1	R/W:0	R/W:0
Bit Name	XOUT FN		XSIRQ EN			FREQ		

Bit	Name	Description
7:6	**XOUT FN**	XOUT Pin Function. This field selects between the different functions of the XOUT pin. 00 = Clock frequency set by XOUT FREQ 01 = Active LOW PA Control 10 = Radio data serial bit stream. If this option is selected and SPI is configured for 3-wire mode, then the MISO pin outputs a serial clock associated with this data stream. 11 = GPIO. To disable this output, set to GPIO mode, and set the GPIO state in IO_CFG_ADR.
5	**XSIRQ EN**	Crystal Stable Interrupt Enable. This bit enables the OS IRQ interrupt. When enabled, this interrupt generates an IRQ event when the crystal has stabilized after the device has woken from sleep mode. This event is cleared by writing zero to this bit.
2:0	**FREQ**	XOUT Frequency. This field sets the frequency output on the XOUT pin when XOUT FN is set to 00. 0 = 12 MHz 1 = 6 MHz 2 = 3 MHz 3 = 1.5 MHz 4 = 0.75 MHz Other values are not defined.

10.5.14 IO_CFG_ADR

Register

Individual Register Names and Addresses:

IO_CFG_ADR : 0x0Dh

LP	7	6	5	4	3	2	1	0
Access : POR	R/W:0	R/W:0	R/W:0	R/W:0	R/W:0	R/W:0	R/W:0	R/W:0
Bit Name	IRQ OD	IRQ POL	MISO OD	XOUT OD	PACTL OD	PACTL GPIO	SPI 3PIN	IRQ GPIO

To use a GPIO pin as an input, the output mode must be set to open drain, and a ‘1’ written to the corresponding output register bit.

LPstar	7	6	5	4	3	2	1	0
Access : POR	R/W:0	R/W:0	R/W:0	R/W:0	R/W:0	R/W:0	R/W:0	R/W:0
Bit Name	IRQ OD	IRQ POL	MISO OD	XOUT OD	RSVD	RSVD	SPI 3PIN	IRQ GPIO

Bit	Name	Description
7	**IRQ OD**	IRQ Pin Drive Strength. Setting this bit configures the IRQ pin as an open drain output. Clearing this bit configures the IRQ pin as a standard CMOS output, with the output ‘1’ drive voltage being equal to the V_{IO} pin voltage.
6	**IRQ POL**	IRQ Polarity. Setting this bit configures the IRQ signal polarity to be active HIGH. Clearing this bit configures the IRQ signal polarity to be active low.
5	**MISO OD**	MISO Pin Drive Strength. Setting this bit configures the MISO pin as an open drain output. Clearing this bit configures the MISO pin as a standard CMOS output, with the output ‘1’ drive voltage being equal to the V_{IO} pin voltage.
4	**XOUT OD**	XOUT Pin Drive Strength. Setting this bit configures the XOUT pin as an open drain output. Clearing this bit configures the XOUT pin as a standard CMOS output, with the output ‘1’ drive voltage being equal to the V_{IO} pin voltage.
3	**PACTL OD**	PACTL Pin Drive Strength. Setting this bit configures the PACTL pin as an open drain output. Clearing this bit configures the PACTL pin as a standard CMOS output, with the output ‘1’ drive voltage being equal to the V_{IO} pin voltage. For LPstar, this bit is RSVD.
2	**PACTL GPIO**	PACTL Pin Function. When this bit is set the PACTL pin is available for use as a GPIO. For LPstar, this bit is RSVD.
1	**SPI 3PIN**	SPI Mode. When this bit is cleared, the SPI interface acts as a standard 4-wire SPI Slave interface. When this bit is set, the SPI interface operates in ‘3-Wire Mode’ combining MISO and MOSI on the same pin (SDAT), and the MISO pin is available as a GPIO pin.
0	**IRQ GPIO**	IRQ Pin Function. When this bit is cleared, the IRQ pin is asserted when an IRQ is active; the polarity of this IRQ signal is configurable in IRQ POL. When this bit is set, the IRQ pin is available for use as a GPIO pin, and the IRQ function is multiplexed onto the MOSI pin. In this case the IRQ signal state is presented on the MOSI pin whenever the $\overline{SS}$ signal is inactive (HIGH).

10.5.15 GPIO_CTRL_ADR

Register

Individual Register Names and Addresses:

GPIO_CTRL_ADR : 0x0Eh

LP	7	6	5	4	3	2	1	0
Access : POR	RW:0	R/W:0	R/W:0	R/W:0	R:x	R:x	R:x	R:x
Bit Name	XOUT OP	MISO OP	PACTL OP	IRQ OP	XOUT IP	MISO IP	PACTL IP	IRQ IP

To use a GPIO pin as an input, the output mode must be set to open drain, and a '1' written to the corresponding output register bit.

LPstar	7	6	5	4	3	2	1	0
Access : POR	RW:0	R/W:0	R/W:0	R/W:0	R:x	R:x	R:x	R:x
Bit Name	XOUT OP	MISO OP	RSVD	IRQ OP	XOUT IP	MISO IP	RSVD	IRQ IP

Bit	Name	Description
7	**XOUT OP**	XOUT Output. When the XOUT pin is configured to be a GPIO, the state of this bit sets the output state of the XOUT pin.
6	**MISO OP**	MISO Output. When the MISO pin is configured to be a GPIO, the state of this bit sets the output state of the MISO pin.
5	**PACTL OP**	PACTL Output. When the PACTL pin is configured to be a GPIO, the state of this bit sets the output state of the PACTL pin. For LPstar, this bit is RSVD.
4	**IRQ OP**	IRQ Output. When the IRQ pin is configured to be a GPIO, the state of this bit sets the output state of the IRQ pin.
3	**XOUT IP**	XOUT Input. When the XOUT pin is configured to be a GPIO, the state of this bit reflects the voltage on the XOUT pin.
2	**MISO IP**	MISO Input. When the MISO pin is configured to be a GPIO, the state of this bit reflects the voltage on the MISO pin.
1	**PACTL IP**	PACTL Input. When the PACTL pin is configured to be a GPIO, the state of this bit reflects the voltage on the PACTL pin. For LPstar ,this bit is RSVD.
0	**IRQ IP**	IRQ Input. When the IRQ pin is configured to be a GPIO, the state of this bit reflects the voltage on the IRQ pin

10.5.16 XACT_CFG_ADR

Register

Individual Register Names and Addresses:

XACT_CFG_ADR : 0x0Fh

	7	6	5	4	3	2	1	0
Access : POR	R/W:1	R/W:x	R/W:0	R/W:0			R/W:0	
Bit Name	ACK EN		FRC END	END STATE			ACK TO	

Bit	Name	Description
7	**ACK EN**	Acknowledge Enable. When this bit is set, an ACK packet is automatically transmitted whenever a valid packet is received; in this case the device is considered to be in transaction mode. After transmission of the ACK packet, the device automatically transitions to the END STATE. When this bit is cleared, the device transitions directly to the END STATE immediately after the end of packet transmission. This bit affects both transmitting and receiving devices.
5	**FRC END**	Force End State. Setting this bit forces a transition to the state set in END STATE. By setting the desired END STATE at the same time as setting this bit the device may be forced to immediately transition from its current state to any other state. This bit is automatically cleared upon completion. Firmware MUST never try to force END STATE while TX GO is set, nor when RX GO is set and a SOP has already been received (packet reception already in progress).
4:2	**END STATE**	Transaction End State. This field defines the mode to which the device transitions after receiving or transmitting a packet. 000 = Sleep Mode 001 = Idle Mode 010 = Synth Mode (TX) 011 = Synth Mode (RX) 100 = RX Mode In normal use, this field is typically set to 000 or 001 when the device is transmitting packets, and 100 when the device is receiving packets. Note that when the device transitions to receive mode as an END STATE, the receiver must still be armed by setting RX GO before the device can begin receiving data. If the system only support packets <=16 bytes then firmware should examine RXC IRQ and RXE IRQ to determine the status of the packet. If the system supports packets > 16 bytes ensure that END STATE is not sleep, force RXF=1, perform receive operation, force RXF=0, and if necessary set END STATE back to sleep.
1:0	**ACK TO**	ACK Timeout. When the device is configured for transaction mode, this field sets the timeout period after transmission of a packet during which an ACK must be correctly received in order to prevent a transmit error condition from being detected. This timeout period is expressed in terms of a number of SOP_CODE_ADR code lengths; if SOP LEN is set, then the timeout period is this value multiplied by 64 µs and if SOP LEN is cleared then the timeout is this value multiplied by 32 µs. 00 = 4x 01 = 8x 10 = 12x 11 = 15x the SOP_CODE_ADR code length ACK_TO must be set to greater than 30 + Data Code Length (only for 8DR) + Preamble Length + SOP Code Length (x2).

10.5.17 FRAMING_CFG_ADR

Register

Individual Register Names and Addresses:

FRAMING_CFG_ADR : 0x10h

	7	6	5	4	3	2	1	0
Access : POR	R/W:1	R/W:0	R/W:1	R/W:0	R/W:0	R/W:1	R/W:0	R/W:1
Bit Name	SOP EN	SOP LEN	LEN EN	SOP TH				

Bit	Name	Description
7	**SOP EN**	SOP Enable. When this bit is set, each transmitted packet begins with a SOP field, and only packets beginning with a valid SOP field are received. If this bit is cleared, no SOP field is generated when a packet is transmitted, and packet reception begins whenever two successive correlations against the DATA_CODE_ADR code are detected.
6	**SOP LEN**	SOP PN Code Length. When this bit is set the SOP_CODE_ADR code length is 64 chips. When this bit is cleared the SOP_CODE_ADR code length is 32 chips.
5	**LEN EN**	Packet Length Enable. When this bit is set the 8-bit value contained in TX_LENGTH_ADR is transmitted immediately after the SOP field. In receive mode, the 8 bits immediately following the SOP field are interpreted as the length of the packet. When this bit is cleared no packet length field is transmitted. 8DR always sends the packet length field (forces LEN EN =1). GFSK requires user set LEN EN = 1.
4:0	**SOP TH**	SOP Correlator Threshold. This is the receive data correlator threshold used when attempting to detect a SOP symbol. There is a threshold for the SOP_CODE_ADR code. This (single) threshold is applied independently to each of SOP1 and SOP2 fields. There are then two thresholds for each of the 64-chip DATA_CODE_ADR codes and 32-chip DATA_CODE_ADR codes. When SOP LEN is set, all 5 bits of this field are used. When SOP LEN is cleared, the most significant bit is disregarded. Typical applications configure SOP TH = 04h for SOP32 and SOP TH = 0Eh for SOP64.

10.5.18 DATA32_THOLD_ADR

Register

Individual Register Names and Addresses:

DATA32_THOLD_ADR : 0x11h

	7	6	5	4	3	2	1	0
Access : POR	R/W:x				R/W:0	R/W:1	R/W:0	R/W:0
Bit Name					TH32			

Bit	Name	Description
3:0:	**TH32**	32-Chip Data PN Code Correlator Threshold. This register sets the correlator threshold used in DSSS modes when DATA CODE LENGTH (see “TX_CFG_ADR” on page 109) is set to 32. Typical applications configure TH32 = 03h.

10.5.19 DATA64_THOLD_ADR

Register

Individual Register Names and Addresses:

DATA64_THOLD_ADR 0x12h

	7	6	5	4	3	2	1	0
Access : POR	R/W:x			R/W:0	R/W:1	R/W:0	R/W:1	R/W:0
Bit Name				TH64				

Bit	Name	Description
4:0	**TH64**	64-Chip Data PN Code Correlator Threshold. This register sets the correlator threshold used in DSSS modes when the DATA CODE LENGTH (see "TX_CFG_ADR" on page 109) is set to 64. Typical applications configure TH64 = 07h.

10.5.20 RSSI_ADR

Register

Individual Register Names and Addresses:

RSSI_ADR : 0x13h

	7	6	5	4	3	2	1	0
Access : POR	R/W:0	R/W:x	R/W:1	R/W:0				
Bit Name	SOP		LNA	RSSI				

A Received Signal Strength Indicator (RSSI) reading is taken automatically when an SOP symbol is detected. In addition, an RSSI reading is taken whenever RSSI_ADR is read. The contents of this register are not valid after the device is configured for receive mode until either a SOP symbol is detected, or the register is read. The conversion can occur as often as once every 12 µs. The approximate slope of the curve is 1.9 dB/count, but is not guaranteed.

If it is desired to measure the background RF signal strength on a channel before a packet has been received then the MCU should perform a 'dummy' read of this register, the results of which should be discarded. This 'dummy' read causes an RSSI measurement to be taken, and therefore subsequent readings of the register yield valid data.

Bit	Name	Description
7	**SOP**	SOP RSSI Reading. When set, this bit indicates that the reading in the RSSI field was taken when a SOP symbol was detected. When cleared, this bit indicates that the reading stored in the RSSI field was triggered by a previous SPI read of this register.
5	**LNA**	LNA State. This bit indicates the LNA state when the RSSI reading was taken. When cleared, this bit indicates that the LNA was disabled when the RSSI reading was taken; if set this bit indicates that the LNA was enabled when the RSSI reading was taken.
4:0	**RSSI**	RSSI Reading. This field indicates the instantaneous strength of the RF signal being received at the time that the RSSI reading was taken. A larger value indicates a stronger signal. The signal strength measured is for the RF signal on the configured channel, and is measured after the LNA stage.

10.5.21 EOP_CTRL_ADR

Register

Individual Register Names and Addresses:

EOP_CTRL_ADR : 0x14h

	7	6	5	4	3	2	1	0
Access : POR	R/W:1	R/W:0	R/W:1	R/W:0	R/W:0	R/W:1	R/W:0	R/W:0
Bit Name	HEN	HINT			EOP			

If the LEN EN bit is set, then the contents of this register has no effect. If the LEN EN bit is cleared, then this register is used to configure how an EOP (end of packet) condition is detected.

Bit	Name	Description
7	**HEN**	EOP Hint Enable. When set, this bit causes an EOP to be detected if no correlations have been detected for the number of symbol periods set by the HINT field and the last two received bytes match the calculated CRC16 for all previously received bytes. Use of this mode reduces the chance of non-correlations in the middle of a packet from being detected as an EOP condition.
6:4	**HINT**	EOP Hint Symbol Count. Hint is the minimum number of consecutive non-correlations symbols at which the last two bytes are checked against the calculated CRC16 to detect an EOP condition. This value cannot be ‘0’, i.e., these bits cannot be ‘000’.
3:0	**EOP**	EOP Symbol Count. An EOP condition is deemed to exist when the number of consecutive non-correlations is detected

10.5.22 CRC_SEED_LSB_ADR

Register

Individual Register Names and Addresses:

CRC_SEED_LSB_ADR : 0x15h

	7	6	5	4	3	2	1	0
Access : POR	R/W:0							
Bit Name	CRC SEED LSB							

The CRC16 seed allows different devices to generate or recognize different CRC16s for the same payload data. If a transmitter and receiver use a randomly selected CRC16 seed, the probability of correctly receiving data intended for a different receiver is 1/65535, even if the other transmitter/receiver are using the same SOP_CODE_ADR codes and channel.

Bit	Name	Description
7:0	**CRC SEED LSB**	CRC16 Seed Least Significant Byte. The LSB of the starting value of the CRC16 calculation

10.5.23 CRC_SEED_MSB_ADR

Register

Individual Register Names and Addresses:

CRC_SEED_MSB_ADR : 0x16h

	7	6	5	4	3	2	1	0
Access : POR	R/W:0							
Bit Name	CRC SEED MSB							

Bit	Name	Description
7:0	**CRC SEED MSB**	CRC16 Seed Most Significant Byte. The MSB of the starting value of the CRC16 calculation.

10.5.24 TX_CRC_LSB_ADR

Register

Individual Register Names and Addresses:

TX_CRC_LSB_ADR : 0x17h

	7	6	5	4	3	2	1	0
Access : POR	R/W:x							
Bit Name	TX CRC LSB							

Bit	Name	Description
7:0	**TX CRC LSB**	Calculated CRC16 LSB. The LSB of the CRC16 that was calculated for the last transmitted packet. This value is only valid after packet transmission is complete.

10.5.25 TX_CRC_MSB_ADR

Register

Individual Register Names and Addresses:

TX_CRC_MSB_ADR : 0x18h

	7	6	5	4	3	2	1	0
Access : POR	R/W:x							
Bit Name	TX CRC MSB							

Bit	Name	Description
7:0	**TX CRC MSB**	Calculated CRC16 MSB. The MSB of the CRC16 that was calculated for the last transmitted packet. This value is only valid after packet transmission is complete.

10.5.26 RX_CRC_LSB_ADR

Register

Individual Register Names and Addresses:

RX_CRC_LSB_ADR : 0x19h

	7	6	5	4	3	2	1	0
Access : POR	R/W:1							
Bit Name	RX CRC LSB							

Bit	Name	Description
7:0	**RX CRC LSB**	Received CRC16 LSB. The LSB of the CRC16 field from the last received packet. This value is valid whether or not the CRC16 field matched the calculated CRC16 of the received packet.

10.5.27 RX_CRC_MSB_ADR

Register

Individual Register Names and Addresses:

RX_CRC_MSB_ADR: 0x1Ah

	7	6	5	4	3	2	1	0
Access : POR	R/W:1							
Bit Name	RX CRC MSB							

Bit	Name	Description
7:0	**RX CRC MSB**	Received CRC16 MSB. The MSB of the CRC16 field from the last received packet. This value is valid whether or not the CRC16 field matched the calculated CRC16 of the received packet.

10.5.28 TX_OFFSET_LSB_ADR

Register

Individual Register Names and Addresses:

TX_OFFSET_LSB_ADR: 0x1Bh

	7	6	5	4	3	2	1	0
Access : POR	R/W:0							
Bit Name	STRIM LSB							

Bit	Name	Description
7:0	**STRIM LSB**	The least significant 8 bits of the synthesizer offset value. This is a 12-bit 2's complement signed number which may be used to offset the transmit frequency of the device by up to ±1.5 MHz. A positive value increases the transmit frequency, and a negative value reduces the transmit frequency. A value of +1 increases the transmit frequency by 732.6 Hz; a value of –1 decreases the transmit frequency by 732.6 Hz. A value of 0x0555 increases the transmit frequency by 1 MHz; a value of 0xAAB decreases the transmit frequency by 1 MHz. Typically, this register is loaded with 0x55 during initialization. Typically this feature is used to avoid the need to change the synthesizer frequency when switching between TX and RX. As the IF = 1 MHz the RX frequency is offset 1 MHz from the synthesizer frequency; therefore, transmitting with a 1 MHz offset allows the same synthesizer frequency to be used for both transmit and receive. Synthesizer offset has no effect on receive frequency.

10.5.29 TX_OFFSET_MSB_ADR

Register

Individual Register Names and Addresses:

TX_OFFSET_MSB_ADR : 0x1Ch

	7	6	5	4	3	2	1	0
Access : POR	R/W:x				R/W:0			
Bit Name					STRIM MSB			

Bit	Name	Description
3:0	**STRIM MSB**	The most significant 4 bits of the synthesizer trim value. Typically, this register is loaded with 0x05 during initialization.

10.5.30 MODE_OVERRIDE_ADR

Register

Individual Register Names and Addresses:

MODE_OVERRIDE_ADR : 0x1Dh

	7	6	5	4	3	2	1	0
Access : POR	R/W:0	R/W:0	R/W:0	R/W:0	R/W:0	R/W:x		R/W:0
Bit Name	RSVD	RSVD	FRC SEN	FRC AWAKE				RST

Bit	Name	Description
7:6	**RSVD**	Reserved. Must be zero.
5	**FRC SEN**	Manually Initiate Synthesizer. Setting this bit forces the synthesizer to start. Clearing this bit has no effect. For this bit to operate correctly, the oscillator must be running before this bit is set.
4:3	**FRC AWAKE**	Force Awake. Force the device out of sleep mode. Setting both bits of this field forces the oscillator to keep running at all times regardless of the END STATE setting. Clearing both of these bits disables this function.
0	**RST**	Reset. Setting this bit forces a full reset of the device. Clearing this bit has no effect.

10.5.31 RX_OVERRIDE_ADR

Register

Individual Register Names and Addresses:

RX_OVERRIDE_ADR : 0x1Eh

	7	6	5	4	3	2	1	0
Access : POR	R/W:0	R/W:0	R/W:0	R/W:0	R/W:0	R/W:0	R/W:0	R/W:x
Bit Name	ACK RX	RXTX DLY	MAN RXACK	FRC RXDR	DIS CRC0	DIS RXCRC	ACE	

This register provides the ability to over-ride some automatic features of the device.

Bit	Name	Description
7	**ACK RX**	When this bit is set, the device uses the transmit synthesizer frequency rather than the receive synthesizer frequency for the given channel when automatically entering receive mode.
6	**RXTX DLY**	When this bit is set and ACK EN is enabled, the transmission of the ACK packet is delayed by 20 µs.
5	**MAN RXACK**	Force Expected Packet Type. When this bit is set, and the device is in receive mode, the device is configured to receive an ACK packet at the data rate defined in TX_CFG_ADR.
4	**FRC RXDR**	Force Receive Data Rate. When this bit is set, the receiver ignores the data rate encoded in the SOP symbol, and receives data at the data rate defined in TX_CFG_ADR.
3	**DIS CRC0**	Reject packets with a zero-seed CRC16. Setting this bit causes the receiver to reject packets with a zero-seed, and accept only packets with a CRC16 that matches the seed in CRC_SEED_LSB_ADR and CRC_SEED_MSB_ADR.
2	**DIS RXCRC**	The RX CRC16 checker is disabled. If packets with CRC16 enabled are received, the CRC16 is treated as payload data and stored in the receive buffer.
1	**ACE**	Accept Bad CRC16. Setting this bit causes the receiver to accept packets with a CRC16 that do not match the seed in CRC_SEED_LSB_ADR and CRC_SEED_MSB_ADR. An ACK is to be sent regardless of the condition of the received CRC16

10.5.32 TX_OVERRIDE_ADR

Register

Individual Register Names and Addresses:

TX_OVERRIDE_ADR : 0x1Fh

	7	6	5	4	3	2	1	0
Access : POR	R/W:0	R/W:0	R/W:0	R/W:0	R/W:0	R/W:0	R/W:0	R/W:0
Bit Name	ACK TX	FRC PRE	RSVD	MAN TXACK	OVRD ACK	DIS TXCRC	RSVD	TX INV

This register provides the ability to over-ride some automatic features of the device.

Bit	Name	Description
7	**ACK TX**	When this bit is set, the device uses the receive synthesizer frequency rather than the transmit synthesizer frequency for the given channel when automatically entering transmit mode.
6	**FRC PRE**	Force Preamble. When this bit is set, the device transmits a continuous repetition of the preamble pattern (see “PREAMBLE_ADR” on page 152) after TX GO is set. This mode is useful for some regulatory approval procedures. Firmware must set bit RST of MODE_OVERRIDE_ADR to exit this mode.
5	**RSVD**	Reserved. Must be zero.
4	**MAN TXACK**	Transmit ACK Packet. When this bit is set, the device sends an ACK packet when TX GO is set.
3	**OVRD ACK**	ACK Override. Use TX_CFG_ADR to determine the data rate and the CRC16 used when transmitting an ACK packet.
2	**DIS TXCRC**	Disable Transmit CRC16. When set, no CRC16 field is present at the end of transmitted packets.
1	**RSVD**	Reserved. Must be zero.
0	**TX INV**	TX Data Invert. When this bit is set the transmit bit stream is inverted.

10.5.33 XTAL_CFG_ADR

Register

Individual Register Names and Addresses:

XTAL_CFG_ADR : 0x26h

	7	6	5	4	3	2	1	0
Access : POR	W:0	W:0	W:0	W:0	W:0	W:0	W:0	W:0
Bit Name	RSVD	RSVD	RSVD	RSVD	START DLY	RSVD	RSVD	RSVD

This register provides the ability to override some automatic features of the device.

Bit	Name	Description
7:4	**RSVD**	Reserved. Must be zero.
3	**START DLY**	Crystal Startup Delay. Setting this bit sets the crystal startup delay to 150 ms to handle warm restarts of the crystal. Firmware MUST set this bit during initialization.
2:0	**RSVD**	Reserved. Must be zero.

10.5.34 CLK_OFFSET_ADR

Register

Individual Register Names and Addresses:

CLK_OFFSET_ADR : 0x27h

	7	6	5	4	3	2	1	0
Access : POR	R/W:0	R/W:0	R/W:0	R/W:0	R/W:0	R/W:0	R/W:0	R/W:0
Bit Name	RSVD	RSVD	RSVD	RSVD	RSVD	RSVD	RXF	RSVD

This register provides the ability to over-ride some automatic features of the device.

Bit	Name	Description
7:2	**RSVD**	Reserved. Must be zero.
1	**RXF**	Force Receive Clock. Streaming applications MUST set this bit during receive mode, otherwise this bit should be cleared.
0	**RSVD**	Reserved. Must be zero.

10.5.35 CLK_EN_ADR

Register

Individual Register Names and Addresses:

CLK_EN_ADR : 0x28h

	7	6	5	4	3	2	1	0
Access : POR	R/W:0	R/W:0	R/W:0	R/W:0	R/W:0	R/W:0	R/W:0	R/W:0
Bit Name	RSVD	RSVD	RSVD	RSVD	RSVD	RSVD	RXF	RSVD

This register provides the ability to over-ride some automatic features of the device.

Bit	Name	Description
7:2	**RSVD**	Reserved. Must be zero.
1	**RXF**	Force Receive Clock Enable. Streaming applications MUST set this bit during initialization.
0	**RSVD**	Reserved. Must be zero.

10.5.36 RX_ABORT_ADR

Register

Individual Register Names and Addresses:

RX_ABORT_ADR : 0x29h

	7	6	5	4	3	2	1	0
Access : POR	R/W:0	R/W:0	R/W:0	R/W:0	R/W:0	R/W:0	R/W:0	R/W:0
Bit Name	RSVD	RSVD	ABORT EN	RSVD	RSVD	RSVD	RSVD	RSVD

This register provides the ability to over-ride some automatic features of the device.

Bit	Name	Description
7:6	**RSVD**	Reserved. Must be zero.
5	**ABORT EN**	Receive Abort Enable. Typical applications disrupt any pending receive by first setting this bit, otherwise this bit is cleared.
4:0	**RSVD**	Reserved. Must be zero.

10.5.37 AUTO_CAL_TIME_ADR

Register

Individual Register Names and Addresses:

AUTO_CAL_TIME_ADR 0x32h

	7	6	5	4	3	2	1	0
Access : POR	R/W:0							
Bit Name	AUTO_CAL_TIME							

This register provides the ability to over-ride some automatic features of the device.

Bit	Name	Description
7:0	**AUTO_CAL_TIME**	Auto Cal Time. Firmware MUST write 3Ch to this register during initialization.

10.5.38 AUTO_CAL_OFFSET_ADR

Register

Individual Register Names and Addresses:

AUTO_CAL_OFFSET_ADR : 0x35h

	7	6	5	4	3	2	1	0
Access : POR	R/W:0							
Bit Name	AUTO_CAL_OFFSET							

This register provides the ability to over-ride some automatic features of the device.

Bit	Name	Description
7:0	**AUTO_CAL_OFFSET**	Auto Cal Offset. Firmware MUST write 14h to this register during initialization.

10.5.39 ANALOG_CTRL_ADR

Register

Individual Register Names and Addresses:

ANALOG_CTRL_ADR : 0x39h

	7	6	5	4	3	2	1	0
Access : POR	R/W:0	R/W:0	R/W:0	R/W:0	R/W:0	R/W:0	R/W:0	R/W:0
Bit Name	RSVD	RSVD	RSVD	RSVD	RSVD	RSVD	RX INV	ALL SLOW

This register provides the ability to over-ride some automatic features of the device.

Bit	Name	Description
7:2	**RSVD**	Reserved. Must be zero.
1	**RX INV**	When set, the incoming receive data is inverted. Firmware MUST set this bit when interoperability with first generation devices is desired.
0	**ALL SLOW**	All Slow. When set, the synth settling time for all channels is the same as for slow channels. It is recommended that firmware set this bit when using GFSK data rate mode. This bit must be set to communicate in DDR/SDR mode for legacy mode communication.

10.5.40 TX_BUFFER_ADR

Byte Register File

Individual Register Names and Addresses:

TX_BUFFER_ADR : 0x20h

	7	6	5	4	3	2	1	0
Access : POR	W:x	W:x	W:x	W:x	W:x	W:x	W:x	W:x
Byte Name								

	15	14	13	12	11	10	9	8
Access : POR	W:x	W:x	W:x	W:x	W:x	W:x	W:x	W:x
Byte Name								

The transmit buffer is a FIFO. Writing to this file adds a byte to the packet being sent. Writing more bytes to this file than the packet length in TX_LENGTH_ADR has no effect, and these bytes are lost. The FIFO accumulates data until it is reset via TX CLR in TX_CTRL_ADR. A previously sent packet of 16 bytes or less, can be transmitted if TX_GO is set without resetting the FIFO. The contents of TX_BUFFER_ADR are not affected by the transmission of an Auto ACK packet.

10.5.41 RX_BUFFER_ADR

Byte Register File

Individual Register Names and Addresses:

RX_BUFFER_ADR : 0x21h

	7	6	5	4	3	2	1	0
Access : POR	R:x	R:x	R:x	R:x	R:x	R:x	R:x	R:x
Byte Name								

	15	14	13	12	11	10	9	8
Access : POR	R:x	R:x	R:x	R:x	R:x	R:x	R:x	R:x
Byte Name								

The receive buffer is a FIFO. Received bytes may be read from this file register at any time that it is not empty, but when reading from this file register before a packet has been completely received care must be taken to ensure that error packets (for example with bad CRC16) are handled correctly.

When the receive buffer is configured to be overwritten by new packets (the alternative is for new packets to be discarded if the receive buffer is not empty), similar care must be taken to verify after the packet has been read from the buffer that no part of it was overwritten by a newly received packet while this file register is being read.

When the VLD EN bit in RX_CFG_ADR is set, the bytes in this file register alternate—the first byte read is data, the second byte is a valid flags for each bit in the first byte, the third byte is data, the fourth byte valid flags, and so on. In SDR and DDR modes the valid flag for a bit is set if the correlation coefficient for the bit exceeded the correlator threshold, and is cleared if it did not. In 8DR mode, the MSB of a valid flags byte indicates whether or not the correlation coefficient of the corresponding received symbol exceeded the threshold. The seven LSB's contain the number of erroneous chips received for the data.

10.5.42 SOP_CODE_ADR

Byte Register File

Individual Register Names and Addresses:

SOP_CODE_ADR : 0x22h

	7	6	5	4	3	2	1	0
Access : POR	R/W:	R/W:	R/W:	R/W:	R/W:	R/W:	R/W:	R/W:
Bit Name								

When using 32-chip SOP_CODE_ADR codes, only the first four bytes of this register are used; in order to complete the file write process, these four bytes must be followed by four bytes of 'dummy' data. However, a class of codes known as 'multiplicative codes' may be used; there are 64-chip codes with good auto-correlation and cross-correlation properties where the least significant 32 chips themselves have good auto-correlation and cross-correlation properties when used as 32-chip codes. In this case the same eight-byte value may be loaded into this file and used for both 32-chip and 64-chip SOP symbols.

When reading this file, all eight bytes must be read; if fewer than eight bytes are read from the file, the contents of the file are rotated by the number of bytes read. This applies to writes, as well.

Recommended SOP Codes:

```
0x91CCF8E291CC373C
0x0FA239AD0FA1C59B
0x2AB18FD22AB064EF
0x507C26DD507CCD66
0x44F616AD44F6E15C
0x46AE31B646AECC5A
0x3CDC829E3CDC78A1
0x7418656F74198EB9
0x49C1DF6249C0B1DF
0x72141A7F7214E597
```

10.5.43 DATA_CODE_ADR

Byte Register File

Individual Register Names and Addresses:

DATA_CODE_ADR : 0x23h

	7	6	5	4	3	2	1	0
Access : POR	R/W:x	R/W:x	R/W:x	R/W:x	R/W:x	R/W:x	R/W:x	R/W:x
Bit Name								

	15	14	13	12	11	10	9	8
Access : POR	R/W:x	R/W:x	R/W:x	R/W:x	R/W:x	R/W:x	R/Wx:	R/W:x
Bit Name								

In GFSK mode, this file register is ignored.

In 64-SDR mode, only the first eight bytes are used.

In 32-DDR mode, only eight bytes are used. The format for these eight bytes is the following:

0x0000 0000 BBBB BBBB 0000 0000 AAAA AAAA,

where ‘0’ represents unused locations.

Example: 0x00000000B2BB092B00000000B86BC0DC;

where ‘B86BC0DC’ represents AAAAAAAA, ‘0000000’ represents unused locations, ‘B2BB092B’ represents BBBBBBBB, and ‘00000000’ represents unused locations.

In 64-DDR and 8DR modes, all sixteen bytes are used.

When reading this file, all sixteen bytes must be read; if fewer than sixteen bytes are read from the file, the contents of the file are rotated by the number of bytes read. This applies to writes, as well.

Certain sixteen-byte sequences have been calculated that provide excellent auto-correlation and cross-correlation properties, and it is recommended that such sequences be used; the default value of this register is one such sequence. In typical applications, all devices use the same DATA_CODE_ADR codes, and devices and systems are addressed by using different SOP_CODE_ADR codes; in such cases it may never be necessary to change the contents of this register from the default value.

Typical applications should use the default code.

10.5.44 PREAMBLE_ADR

Byte Register File

Individual Register Names and Addresses:

PREAMBLE_ADR : 0x24h

	7	6	5	4	3	2	1	0
Access : POR						R/W:x	R/W:x	R/W:x
Bit Name								

This register is three bytes in length.

Byte	Name	Description
3		Most significant eight chips of the preamble sequence.
2		Least significant eight chips of the preamble sequence.
1		The number of repetitions of the preamble sequence that are to be transmitted. The preamble may be disabled by writing 0x00 to this byte.

If using 64-SDR to communicate with CYWUSB69xx devices, set number of repetitions to eight for optimum performance.

If using DDR/32 -DDR to communicate with CYWUSB69xx devices, set number of repetitions to four for optimum performance.

When reading this file, all three bytes must be read; if fewer than three bytes are read from the file, the contents of the file are rotated by the number of bytes read. This applies to writes, as well.

10.5.45 MFG_ID_ADR

Byte Register File

Individual Register Names and Addresses:

MFG_ID_ADR : 0x25h

	7	6	5	4	3	2	1	0
Access : POR			R:x	R:x	R:x	R:x	R:x	R:x
Bit Name								

This register is six bytes long.

To minimize ~190 µA of current consumption (default), execute a ‘dummy’ single-byte SPI write to this address with a zero data stage after the contents have been read. Non-zero to enable reading of fuses. Zero to disable reading fuses.

Index

D

E

F

R

S

T

U

V

X