CY4672

PRoC™ LP Reference Design Kit Guide

Doc. # 001-69289 Rev.*A

Flash Code Protection

Cypress products meet the specifications contained in their particular Cypress PSoC Data Sheets. Cypress believes that its family of PSoC products is one of the most secure families of its kind on the market today, regardless of how they are used. There may be methods, unknown to Cypress, that can breach the code protection features. Any of these methods, to our knowledge, would be dishonest and possibly illegal. Neither Cypress nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Cypress is willing to work with the customer who is concerned about the integrity of their code. Code protection is constantly evolving. We at Cypress are committed to continuously improving the code protection features of our products.

Contents

1.   Introduction

Thank you for your interest in the CY4672 PRoC™ LP Reference Design Kit (RDK). This kit provides an implementation of a low-power, 2.4 GHz bidirectional wireless keyboard, mouse, and USB bridge based on the PRoC LP single-chip MCU 2.4 GHz transceiver. Schematics, source code, firmware, and RF gerber files are provided to guide PC human interface device (HID) designers with their applications.

Chapter 2 describes the installation and configuration of the CY4672 PRoC LP RDK. Chapter 3 describes the kit operation. Chapter 4 describes the hardware operation. Chapter 5 describes the code examples provided along with the kit. The Appendix section provides the schematics, bill of materials (BOM), and protocols for the kit.

1.1   Kit Contents

The CY4672 PRoC LP Reference Design Kit includes:
- PRoC LP-based mouse with ADNS-3040 optical sensor
- WirelessUSB LS 101 multimedia keyboard
- Small form factor bridge
- Four batteries
- Kit CD/DVD with firmware source and documentation

Visit http://www.cypress.com/shop for more information. Inspect the contents of the kit. If any parts are missing, contact your nearest Cypress sales office for further assistance.

1.2   PSoC Designer

PSoC Designer is the integrated design environment (IDE) that you can use to customize your PSoC application. The latest version of PSoC Designer has many new features, bug fixes, and support for new PSoC devices.

For more information about PSoC Designer, see the PSoC Designer IDE Guide located at: `<Installed_directory>\Cypress\PSoC Designer\<version>\Documentation`

1.3   PSoC Programmer

PSoC Programmer 3.14 offers a simple GUI that connects to hardware and enables to program and configure PSoC devices.

1.4   Additional Learning Resources

Visit http://www.cypress.com for additional learning resources in the form of data sheets, technical reference manual, and application notes.

1.4.1   Reference Documents

- CYRF6936 - WirelessUSB™ LP 2.4 GHz Radio SoC
  http://www.cypress.com/?rID=14284
- CY7C601xx, CY7C602xx - enCoRe™ II Low-Voltage Microcontroller
  http://www.cypress.com/?rID=13556
- CY7C63310, CY7C638xx - enCoRe™ II Low Speed USB Peripheral Controller
  http://www.cypress.com/?rID=14212
- CY7C603xx - enCoRe™ III Low Voltage
  http://www.cypress.com/?rID=13558
- ADNS-3040 - Ultra Low-Power Optical Mouse Sensor
  http://www.avagotech.com/docs/AV02-0150EN
- MiniProg
  http://www.cypress.com/?rID=37459
- PSoC Designer Training
  http://www.cypress.com/?rID=37459

1.5   Document History

Revision	PDF Creation Date	Origin of Change	Description of Change
**	05/06/2011	CSAI	Initial version of kit guide
*A	03/13/2012	ELIN	Kit guide updated with OOB review comments.

1.6   Documentation Conventions

Table 1-1. Document Conventions for Guides

Convention	Usage
Courier New	Displays file locations, user entered text, and source code: C:\ ...cd\icc\
Italics	Displays file names and reference documentation: Read about the *sourcefile.hex* file in the *PSoC Designer User Guide*.
[**Bracketed, Bold**]	Displays keyboard commands in procedures: [**Enter**] or [**Ctrl**] [**C**]
File > Open	Represents menu paths: File > Open > New Project
Bold	Displays commands, menu paths, and icon names in procedures: Click the **File** icon and then click **Open**.
Times New Roman	Displays an equation: $2 + 2 = 4$
Text in gray boxes	Describes cautions or unique functionality of the product.

2.   Getting Started

This chapter describes the installation of the CY4672 PRoC LP Reference Design Kit.

2.1   Kit Installation

To install the kit software, follow these steps:

1. Insert the kit CD into the CD drive of your PC. The CD is designed to auto-run and the kit installer startup screen appears.

   You can also download the latest kit installer ISO file from http://www.cypress.com/go/CY4672. Create an installer CD or extract the ISO using WinRar and install the executables.

2. Click **Install CY4672 PRoC LP RDK** to start the installation, as shown in Figure 2-1.

   Figure 2-1. Kit Installer Startup Screen

   Note If auto-run does not execute, double-click on the *cyautorun.exe* file in the root directory of the CD, as shown in Figure 2-2.

Figure 2-2. Root Directory of CD

3. The InstallShield Wizard screen appears. The default location for setup is shown on the InstallShield Wizard screen. You can change the location for setup using **Change,** as shown in Figure 2-3.

4. Click **Next** to launch the kit installer.

Figure 2-3. InstallShield Wizard
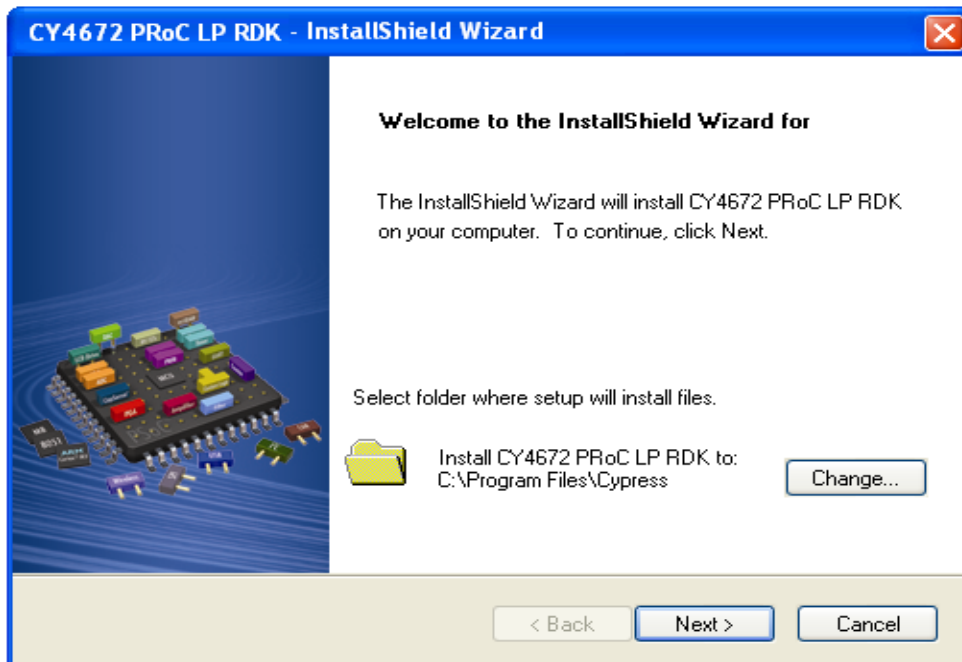
5. On the Product Installation Overview screen, select the installation type that best suits your requirement. The drop-down menu has three options - **Typical**, **Complete**, and **Custom**, as shown in Figure 2-4.

6. Click **Next** to start the installation.

Figure 2-4. Installation Type Options
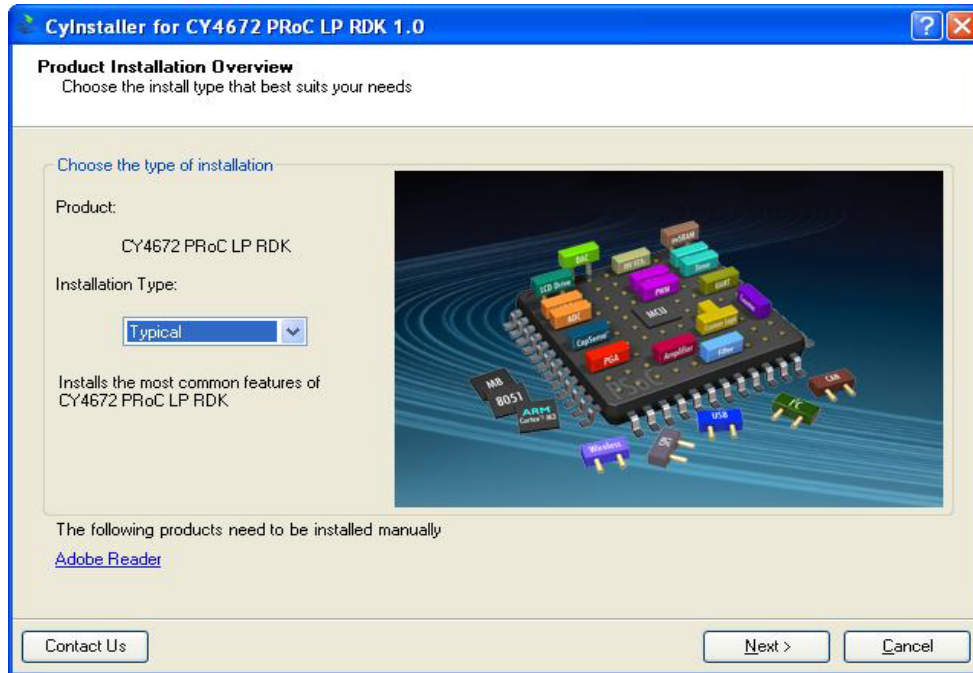
7. When the installation begins, all packages are listed on the Installation page. A green check mark appears adjacent to every package that is downloaded and installed, as shown in Figure 2-5.

8. Wait until all the packages are downloaded and installed successfully.

Figure 2-5. Installation Page
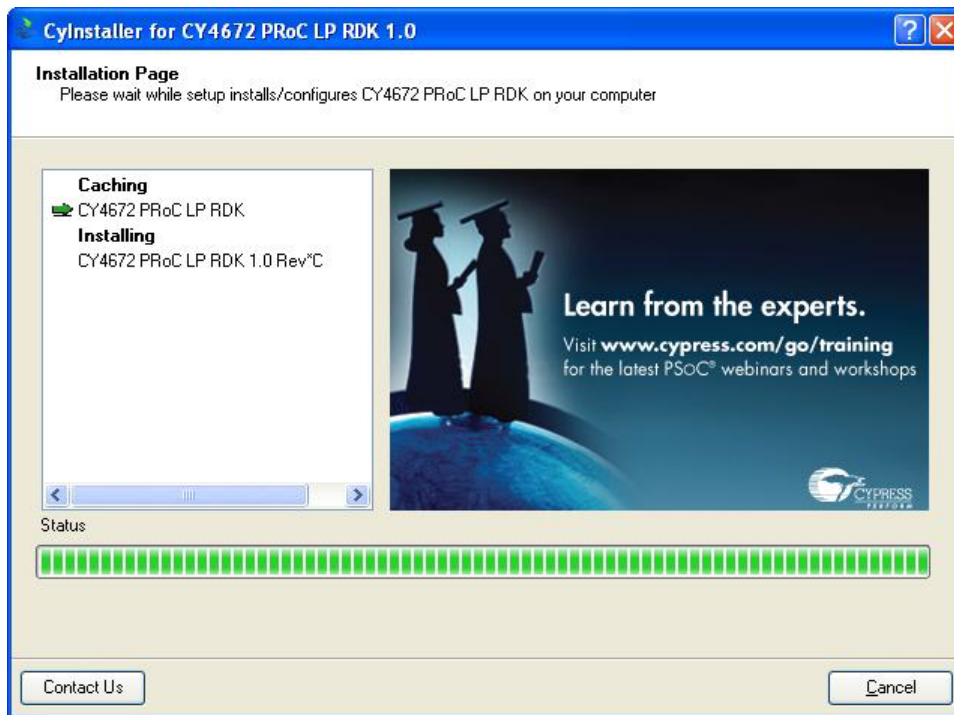
9. Click **Finish** to complete the installation.

Figure 2-6. Installation Completion Page
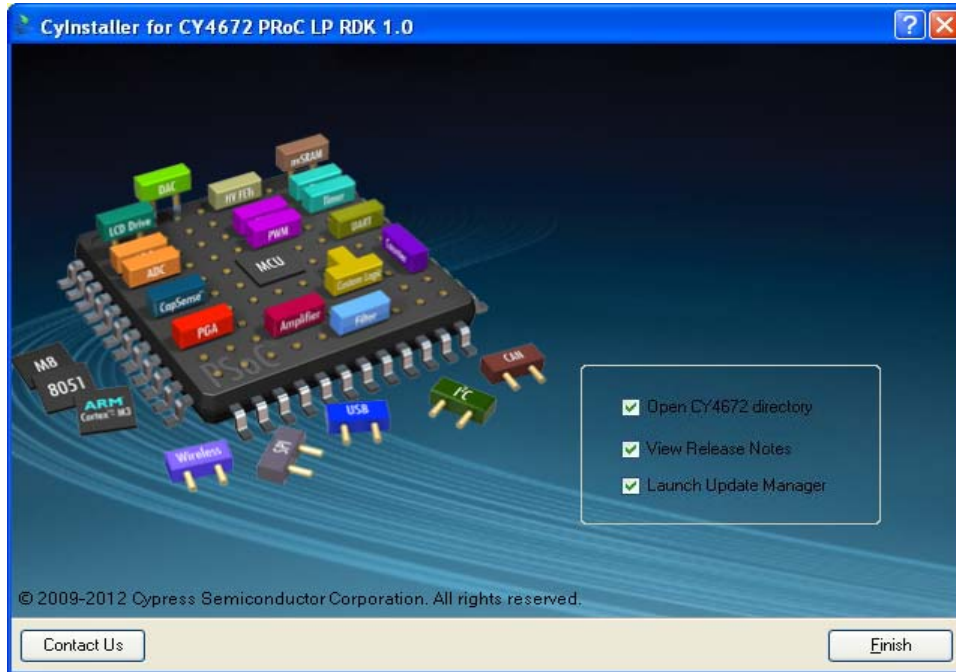
Note: After software installation, verify your installation and setup.

2.2   PSoC Designer

10. Click **Start > All Programs > Cypress > PSoC Designer <version> > PSoC Designer <version>**.

11. Click **File > New Project** to create a project. Click **File > Open Project/Workspace** to work with an existing project.

Figure 2-7. PSoC Designer Interconnect View
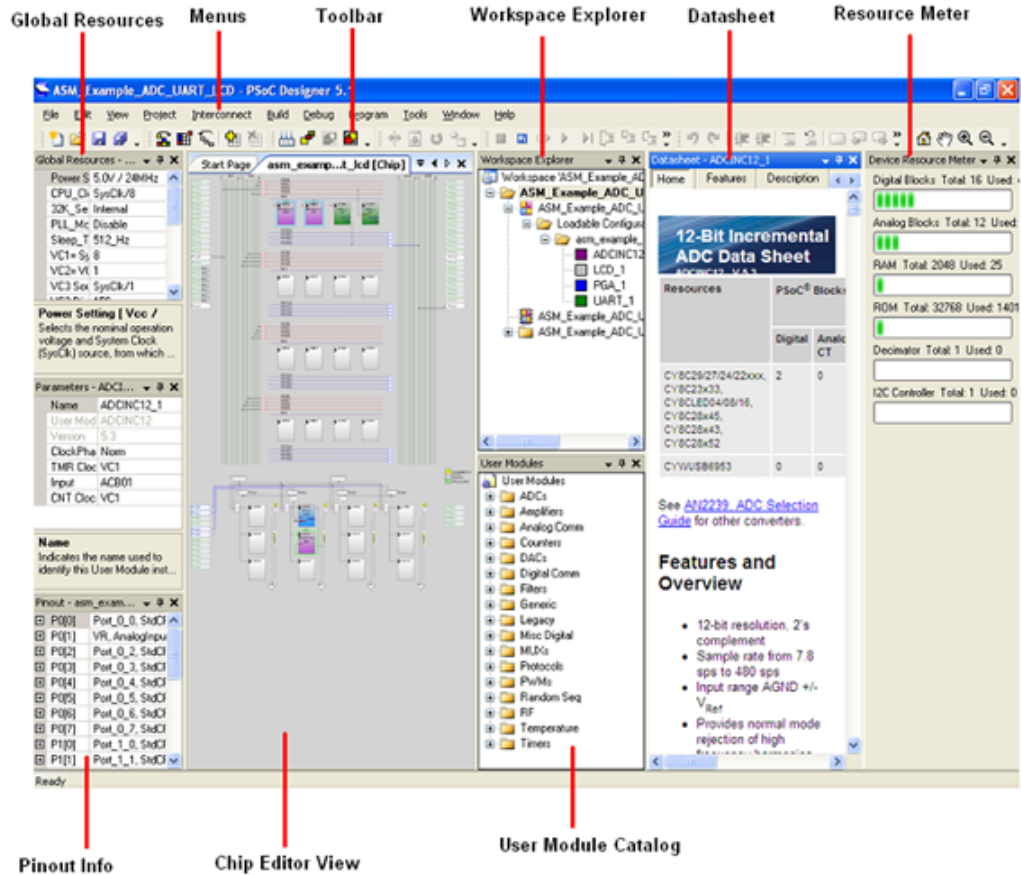
12. To experiment with the code examples, go to .

Note For more details on PSoC Designer, see the PSoC Designer IDE Guide at:
`<Install_directory>\Cypress\PSoC Designer\<version>\Documentation`.

2.3 PSoC Programmer

1. Click **Start > All Programs > Cypress > PSoC Programmer <version> > PSoC Programmer <version>**.

2. Select the MiniProg from **Port Selection**, as shown in Figure 2-8.

Figure 2-8. PSoC Programmer Window
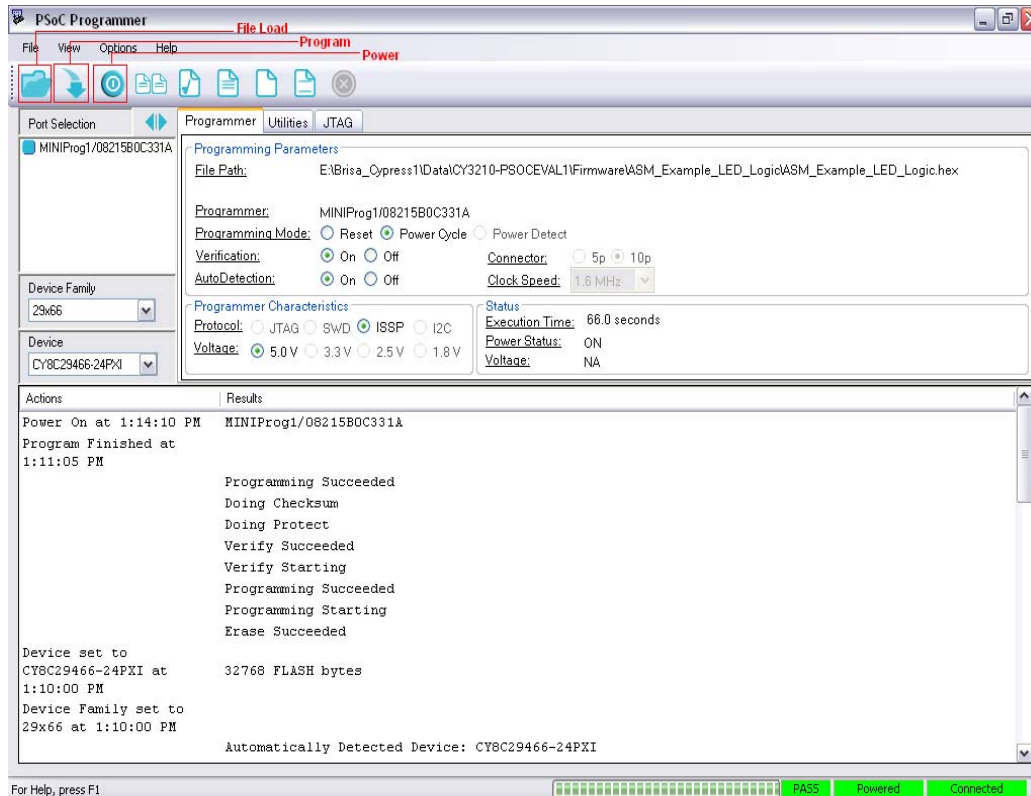
3. Click the **File Load** button in the PSoC Programmer menu bar; navigate and select the appropriate hex file.

4. Use the **Program** button to load the hex file on to the chip.

5. When programming is successful, the **Programming Succeeded** message is displayed on the Actions pane.

6. Close PSoC Programmer.

Note For more details on PSoC Programmer, see the user guide at:
`C:\Program Files\Cypress\Programmer\<version>\Documents.`

2.4 Installation of Battery Level and Link Quality Application

The instructions to install the battery level and link quality applications are as follows.

1. Copy *WirelessUSB.exe,* located from <Installed_Directory>\Cypress\CY4672 PRoC LP RDK\<Ver>\Software\Binaries to the desired directory (`C:\Cypress\WirelesUSB`).

2. Click **Start > Run** and manually execute the *WirelessUSB.exe* application with the setup option. (C:\Cypress\WirelesUSB\WirelessUSB.exe -setup) or double-click *WirelessUSB.exe.*

3. The application should run and register to auto-run at startup.

To uninstall battery level and link quality application:

1. Click **Start > Run** and manually execute the *WirelessUSB.exe* application with the remove option.
   (C:\Cypress\WirelesUSB\WirelessUSB.exe -remove)

2. The application will unload and no longer auto-run at startup.

3. Delete *WirelessUSB.exe.*

Figure 2-9. WirelessUSB
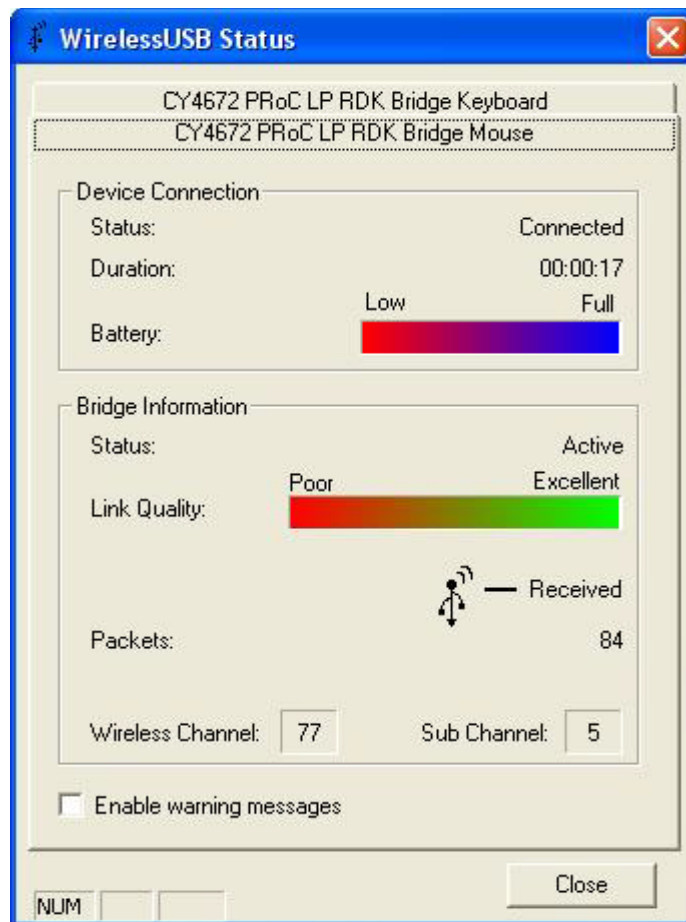
2.5 Install Hardware

No hardware installation is required for this kit.

CY4672 PRoC LP Reference Design Kit Guide, Doc. # 001-69289 Rev.*A

3.   Kit Operation

3.1   Introduction

The CY4672 PRoC LP RDK includes the following components:

■ **PRoC LP Bridge/Dongle** - A single-chip USB bridge based on the CYRF69213 PRoC LP for bridges. Supports Microsoft Windows Vista HotStart (Direct Application Launch) via two dedicated I/Os.

■ **PRoC LP Mouse** - The RDK mouse is a three-button optical mouse with a scroll wheel. It is based on the CYRF69103 PRoC LP for peripherals and the Avago 3040 optical sensor.

■ **WirelessUSB LP Keyboard** - The RDK keyboard is a 101-key keyboard with multimedia keys (such as internet and mail) and power keys. It features AES 128-bit encryption and bidirectional communication for toggling CAPS LOCK and NUM LOCK LEDs. It is based on the CYRF6936 WirelessUSB LP radio SoC and CY7C60123 Wireless enCoRe II MCU.

3.2   Bridge

The CY4672 RDK uses PRoC LP CYRF69213 for the bridge. The PRoC LP RDK Bridge has the capability of being programmed through the USB connector using a Cypress USB adapter board, CY3655-PLG.

Figure 3-1. Cypress USB Programming Adapter
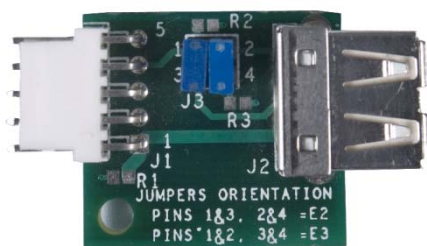
1. Connect LP RDK Bridge to the USB adapter board.
2. Connect the MiniProg to the ISSP header on the USB adapter, as shown in Figure 3-2.
3. Connect the MiniProg to the PC through USB A to Mini B cable.
4. On the USB adapter, place two jumpers between pins 1 and 3 and pins 2 and 4.
5. To program the hex file onto the Bridge using the MiniProg, open PSoC Programmer and select the MiniProg from the **Port Selection** window.
6. Click **Program**. While programming is in progress, the target power LED on the MiniProg turns on.

Figure 3-2. Programming the Bridge

7. When Programming Succeeded appears in the Actions pane, detach the MiniProg.

8. Connect the Bridge dongle through the USB port to the PC, wait for the bridge to enumerate. When the dongle is first plugged in, the red LED turns ON. It turns OFF when the USB enumeration process completes. **Note** If the bind button is pushed when the dongle is first plugged in, the firmware enters MTK test mode and blinks the LED. The LED blinks continuously until the dongle is removed from the computer.

9. Start using the Bridge, after it is enumerated.

10. To bind the bridge with the mouse or keyboard, press the bind button on the bridge; the bridge goes into Bind Mode. In Bind Mode, the bridge uses the Bind ID to communicate with the mouse or keyboard to bind to the system.

The red LED blinks ON/OFF when the dongle is in Bind Mode. The ON and OFF time is approximately 320 ms, which is the rate at which the dongle changes channels during the bind process.

The red LED also blinks ON/OFF when the PC is suspended. The blinking rate is approximately 1 second, which is the frequency of the wake up interrupts.

The green LED turns ON when the dongle receives data from mouse or keyboard. It remains on for 250 ms after the last received data packet.

The green LED turns ON and remains ON if a key is pressed and held (due to the keyboard's sending Keep Alive packets).

The green LED turns ON and remains ON during Ping Mode (in normal operation, Ping Mode is a very short period. You may not notice this period).

The red and green LEDs blink alternately when in Manufacturing Test mode.

3.3 Mouse

The CY4672 RDK uses a low cost PRoC LP for the RDK mouse. The RDK mouse is enclosed in a skin that is designed for the Avago ADNS-3040 Ultra Low-Power mouse sensor. The mouse features three buttons with one button combined with the scroll wheel function. There is a connect button on the bottom of the mouse enabling you to perform an explicit bind with the bridge.

Figure 3-3. Bind Button, ON-OFF Switch, and Battery Compartment

J10 is a programming header. Either the ICE-Cube or the MiniProg may be used to program the mouse microcontroller using this ISSP header.

Connect your computer to the ISSP header J10 on the mouse using the MiniProg and a USB cable (A to Mini B). Programming can be done using PSoC Programmer.

1. To program the hex file onto the mouse using the MiniProg, open PSoC Programmer and select the MiniProg from the **Port Selection** window.

2. Click **Program**. While Programming is in progress, the Target Power LED on the MiniProg is ON.

Figure 3-4. Programming Mouse

3. When the Programming Succeeded message appears in the Actions pane, detach the MiniProg.

4. Place a 2-pin jumper installed from a 2-pin jumper installed from J11 to J10.1 to enable the radio to power the processor. Jumper removal is required when programming U2 to disconnect the radio from the MiniProg 5 V source. The mouse is powered using two AA batteries.

5. To start using the mouse, place the batteries in the battery terminal, switch ON the mouse, and bind it with the bridge.

6. To bind the mouse with the bridge, press the bind button on the bridge; the bridge goes into Bind Mode. In Bind Mode, the red LED blinks ON/OFF. The bridge uses the Bind ID to communicate with the mouse to bind to the system and then press the bind button on the mouse.

When the mouse is bound with the Bridge dongle, the mouse works smoothly with the system. The green LED turns ON when the dongle receives data from mouse. It remains ON for 250 ms after the last received data packet.

3.4 Keyboard

The CY4672 Reference Design Kit uses an enCoRe™ II LV controller and CYRF6936 LP Radio for the RDK keyboard.

Figure 3-5. Keyboard Plastic

Figure 3-6. Exploded Keyboard

Keyboard PCB

Figure 3-6 shows the keyboard with the top removed. The radio/enCoRe II LV board (PDC-9265) is shown in the upper right hand corner.

Figure 3-7. Keyboard Battery Compartment and Bind Button

Bind Button

Battery Compartment
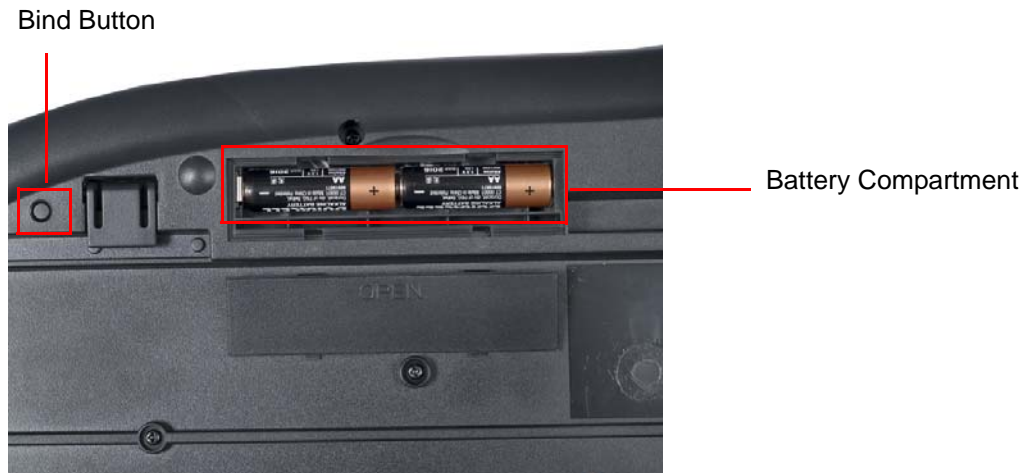
Figure 3-7 shows the integrated battery compartment and bind button located on the bottom side of the keyboard. The battery compartment cover is also shown.

The CY4672 keyboard can be programmed using a MiniProg or ICE-Cube. Connect your computer to the keyboard's ISSP connector (J2) using the MiniProg and a USB cable (A to Mini B). Programming can be done using PSoC Programmer.

1. To program the hex file onto the keyboard using the MiniProg, open PSoC Programmer and select the MiniProg from the **Port Selection** window.

2. Click **Program**. While Programming is in progress, the Target Power LED on the MiniProg is ON.

Figure 3-8. Programming Keyboard

3. When the Programming Succeeded message appears in the Actions pane, detach the MiniProg.

4. A 2-pin jumper installed from J3.1 to J2.1 enables the radio to power the processor. Jumper removal is required when programming U2 to disconnect the radio from the MiniProg 5 V source.

5. The keyboard is powered using two AA batteries. To start using the keyboard, place the batteries in battery compartment and bind it with the bridge.

6. To bind the keyboard with the bridge, press the bind button on the bridge, the bridge goes into Bind Mode. In Bind Mode, the red LED blinks ON/OFF. The bridge uses the Bind ID to communicate with the keyboard to bind to the system and then press the bind button on the keyboard.

When the keyboard is bound with the Bridge dongle, the keyboard works smoothly with the system. The green LED turns ON when the dongle receives data from the keyboard. It remains ON for 250 ms after the last received data packet.

The green LED turns ON and remains ON if a key is pressed and held (due to the keyboard's sending Keep Alive packets).

4.   Hardware

4.1   Bridge

The CY4672 Reference Design Kit uses PRoC LP CYRF69213 for the bridge. This bridge may be plugged into the USB port on a PC to provide the wireless USB bridge functionality. The architecture is designed to be modular for extendibility and maintainability. It can also be easily ported from one hardware platform to another assuming the use of an equivalent microprocessor. Porting to another microprocessor requires more work to account for the USB hardware support and other hardware specific changes.

Design efforts are made to reduce the 'on time' of the microprocessor and radio to conserve battery life of attached devices. This includes protocol optimizations along with using sleep features of the PRoC LP.

The bridge connects the remote RoC LP HIDs to a low-speed USB host. This firmware supports two-way communication with bridge and HID devices configured as transceivers. Packets similar to standard USB HID packets are encapsulated inside wireless PRoC LP packets, which also contain a packet header and CRC to help the bridge correctly process the USB HID data packets. Valid packets are then sent via USB to the USB host.

Figure 4-1. Bridge Block Diagram
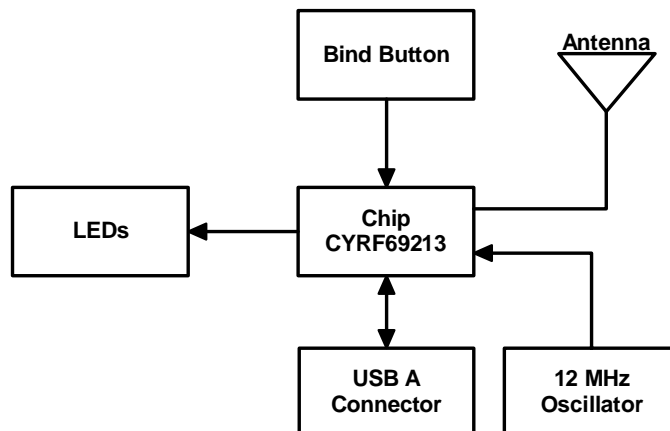
4.1.1   Functional Description

The following figure shows the different functional blocks on the CY4672 Bridge dongle.

Figure 4-2. Bridge Board Functional Blocks
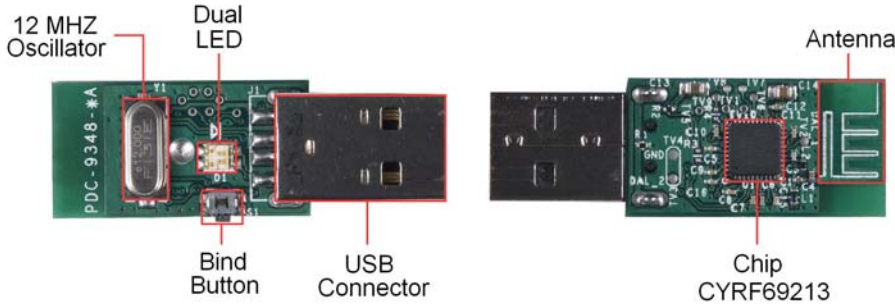
4.1.1.1   *CYRF69213 Chip*

PRoC LP devices have integrated radio and microcontroller functions in the same package to pro-vide a dual role single-chip solution. Communication between the microcontroller and the radio is via the SPI interface between both functions.

The CYRF69213 is a radio system-on-chip device, providing a complete RF system solution with a single device and a few discrete components. The CYRF69213 is designed to implement low-cost wireless systems operating in the worldwide 2.4 GHz Industrial, Scientific, and Medical (ISM) fre-quency band (2.400 GHz to 2.4835 GHz).

The microcontroller function is based on the powerful CYRF69213 microcontroller. It is an 8-bit flash programmable microcontroller with integrated low-speed USB interface. CYRF69213 is a single device, two functions, an 8-bit flash based USB peripheral MCU function and 2.4 GHz radio trans-ceiver function in a single device optimized for HID applications.

The CYRF69213 PRoC LP low-speed is targeted for the following applications:

USB bridge for HID

- Wireless mice
- Wireless keyboards
- Remote controls
- Gaming applications

USB bridge for general purpose applications

- Consumer electronics
- Industrial applications
- White goods
- Home automation
- Personal health

Figure 4-3. Schematic View of CYRF69213 Chip
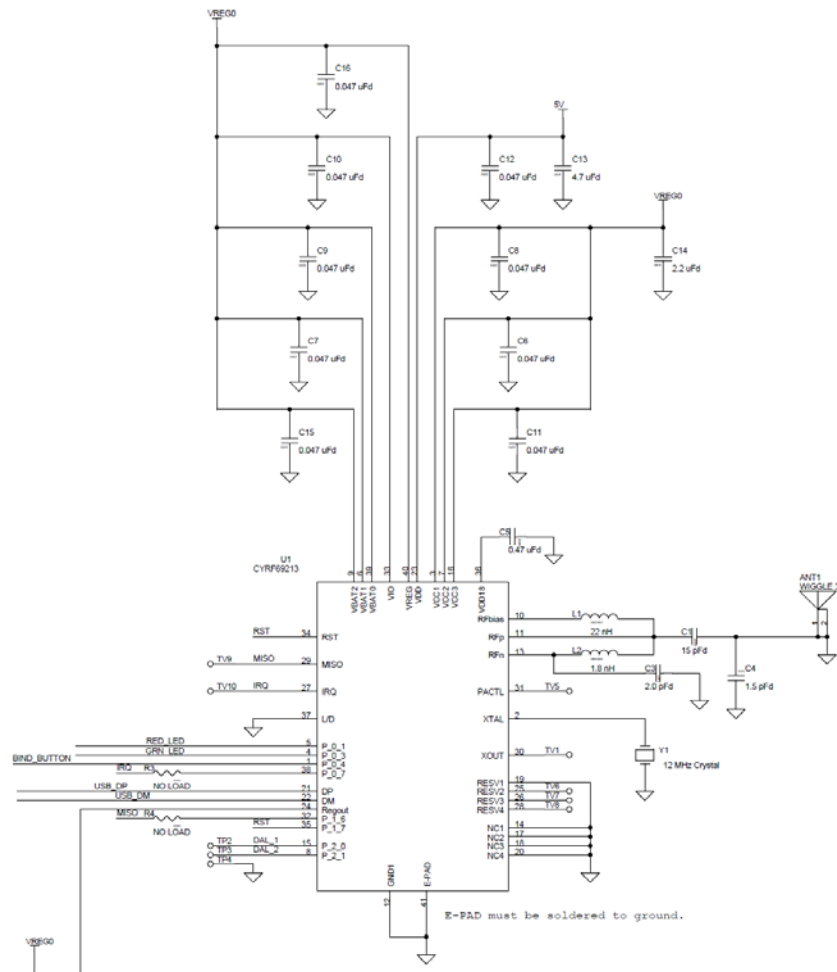
Table 4-1. Pin Details of CYRF69213 Chip

Pin	Name	Description
1	P0.4	Individually configured GPIO
2	Xtal_in	12 MHz crystal. External clock in
3	VCC	Connected to pin 24 via 0.047 µF capacitor
4	P0.3	Individually configured GPIO
5	P0.1	Individually configured GPIO
6	Vbat	Connected to pin 24 via 0.047 µFshunt capacitor
7	VCC	Connected to pin 24 via 0.047 µF capacitor
8	P2.1	GPIO. Port 2 Bit 1
9	Vbat	Connected to pin 24 via 0.047 µFshunt capacitor
10	RFBIAS	RF pin reference voltage
11	RFP	Differential RF signal to and from antenna
12	GND	Ground
13	RFN	Differential RF signal to and from antenna
14	NC	Connect to GND

Table 4-1. Pin Details of CYRF69213 Chip

Pin	Name	Description
15	P2.0	GPIO. Port 2 Bit 0
16	VCC	Connected to pin 24 via 0.047 µF capacitor
17	NC	Connect to GND
18	NC	Connect to GND
19	RESV	Reserved. Must be connected to GND
20	NC	Connect to GND
21	D+	Low Speed USB I/O
22	D-	Low Speed USB I/O
23	VDD_micro	4.0–5.5 for 12 MHz CPU and 4.75–5.5 for 24 MHz CPU
24	P1.2 / VREG	Must be configured as 3.3 V output. It must have a 1 µF to 2 µF output capacitor
25	P1.3 / nSS	Slave select SPI pin
26	P1.4 / SCK	Serial Clock pin from MCU function to radio function
27	IRQ	Interrupt output, configure high/low or GPIO
28	P1.5 / MOSI	Master Out Slave In
29	MISO	Master In Slave Out, from radio function. Can be configured as GPIO
30	XOUT	Buffered CLK, PACTL_n or GPIO
31	PACTL	Control for external PA or GPIO
32	P1.6	GPIO. Port 1 Bit 6
33	VIO	I/O interface voltage. Connected to pin 24 via 0.047 µF
34	Reset	Radio Reset. Connected to VDD via 0.47 µF capacitor or to microcontroller GPIO pin. Must have a RESET = HIGH event the very first time power is applied to the radio otherwise the state of the radio function control registers is unknown
35	P1.7	GPIO. Port 1 Bit 7
36	VDD_1.8	Regulated logic bypass. Connected via 0.47 µF to GND
37	L/D	Connect to GND
38	P0.7	GPIO. Port 0 Bit 7
39	Vbat	Connected to pin 24 via 0.047 µFshunt capacitor
40	VREG	Connected to pin 24
41	E-pad	Must be connected to GND
42	Corner Tabs	Do not connect corner tabs

4.1.1.2   USB A Connector

The USB connector communicates between the PC and the bridge. It also supplies an input voltage of 5 V to power up the bridge.

Figure 4-4. Schematic View of USB A Connector
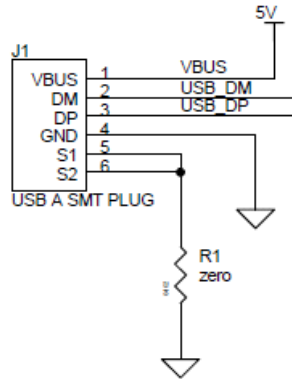
4.1.1.3   Bind Button

This button is used to bind the mouse and keyboard with the bridge. When the bind button on the bridge is pressed, the bridge goes into Bind Mode. In Bind Mode, the bridge uses the Bind ID to communicate with the mouse/keyboard to bind to the system.

Figure 4-5. Schematic View of Bind Button
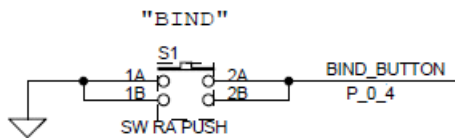
4.1.1.4   LEDs

The CY4672 Bridge board has dual LEDs to indicate different status.

LED2 (red):

■ When the dongle is first plugged in, the red LED turns ON. It turns OFF when the USB enumeration process completes. **Note** If the bind button is pushed when the dongle is first plugged in, the firmware enters MTK test mode and the LED blinks. The LED blinks continuously until the dongle is removed from the computer.

■ The red LED blinks ON/OFF when the dongle is in Bind Mode. The ON and OFF time is approximately 320 ms, which is the rate at which the dongle changes channels during the bind process.

■ The red LED also blinks ON/OFF when the PC is suspended. The blinking rate is approximately 1 second, which is the frequency of the wake up interrupts.

LED1 (green):

■ The green LED turns ON when the dongle receives data from mouse or keyboard. It remains ON for 250 ms after the last received data packet.

■ The green LED turns ON and remains ON if a key is pressed and held (due to the keyboard's sending Keep Alive packets).

■ The green LED turns ON and remains ON during Ping Mode (in normal operation, Ping Mode is a very short period. You may not notice this period).

The red and green LEDs blink alternately when in Manufacturing Test mode.

Figure 4-6. Schematic View of LEDs

Figure 4-7. Power Supply System Structure for the Bridge
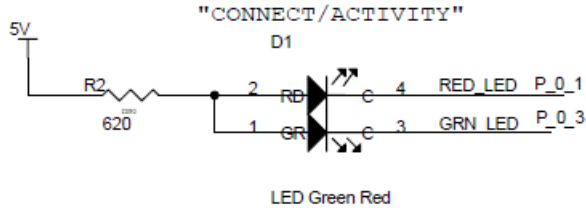
4.1.2   Power Supply System

The CY4672 Bridge board is supplied power from the USB A connector on the board.

Figure 4-7. Power Supply System Structure for the Bridge
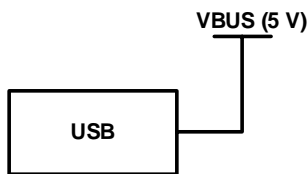
Figure 4-8. Schematic View of Power Supply System Structure for Bridge
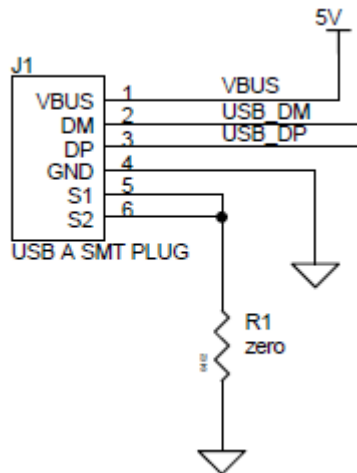
4.2   Mouse

The CY4672 Reference Design Kit uses a low-cost PRoC LP for the RDK mouse. The architecture is designed to be modular for extendibility and maintainability. It can also be easily ported from one hardware platform to another assuming the use of an equivalent microprocessor. Porting to another microprocessor family requires more work to account for hardware specific changes.

Design efforts are made to reduce the 'on' time of the microprocessor and radio to conserve battery life. This includes protocol optimizations along with using sleep features of the PRoC LP and optical sensor.

The mouse design uses the SS12 schottky diode (D1) and CDH53100LC inductor (L3) for its boost circuitry. With these high efficiency components, preliminary characterization data shows a range of approximately 74% to 87% efficiency for the 1.8 V to 2.7 V VBAT voltage range at different temperatures (–10 °C to 80 °C). The mouse is a higher power consumption device compared to the key-

board. Extending the battery life is one of the crucial design considerations in the mouse design. The trade off for a higher efficiency boost circuitry is the component costs and the board size (these components are slightly bigger in size compared to the ones used in the keyboard design). These components do not provide enough current capacity at the low end of the VBAT voltage range to handle the worst case optical sensor load and the PMU output voltage may droop under these conditions. The recommendation is to use an external DC/DC boost circuit for the optical system only.

Figure 4-9. Mouse Block Diagram
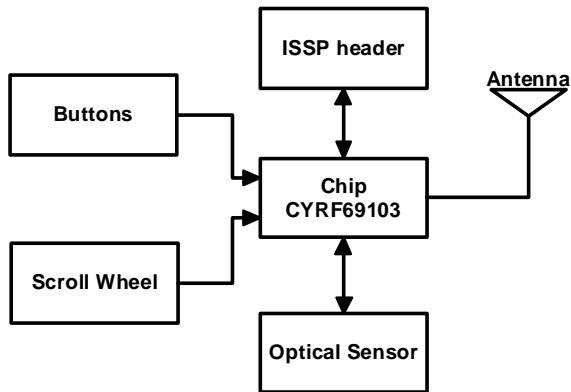
4.2.1   Functional Description

The mouse board includes the CYRF69103 chip, ISSP header, optical sensor, encoder, three buttons, scroll wheel, 12-MHz oscillator, and an antenna. The following figure shows the different functional blocks on the CY4672 mouse board.

Figure 4-10. Functional Blocks of Mouse
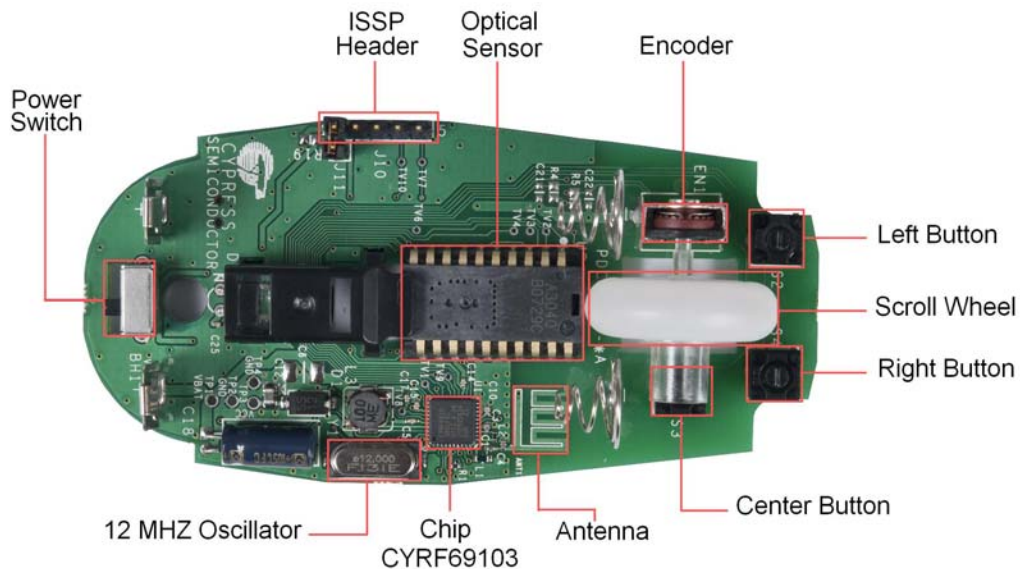
4.2.1.1   CYRF69103 Chip

The CYRF69103 is a radio system-on-chip device, providing a complete RF system solution with a single device and a few discrete components. The CYRF69103 is designed to implement low-cost wireless systems operating in the worldwide 2.4 GHz Industrial, Scientific, and Medical (ISM) frequency band (2.400 GHz to 2.4835 GHz).

The MCU function is an 8-bit flash-programmable microcontroller. The instruction set is optimized specifically for HID and a variety of other embedded applications. PRoC LP devices are integrated radio and microcontroller functions in the same package to provide a dual-role single-chip solution. Communication between the microcontroller and the radio is through the radio's SPI interface.

The SoC contains a 2.4 GHz 1 Mbps GFSK radio transceiver, packet data buffering, packet framer, DSSS baseband controller, Received Signal Strength Indication (RSSI), and SPI interface for data transfer and device configuration.

The CYRF69103 PRoC LP is targeted for the following applications:

Wireless HID devices:

- Mice
- Remote controls
- Presenter tools
- Barcode scanners
- POS terminal

General purpose wireless applications:

- Industrial applications
- Home automation
- White goods
- Consumer electronics
- Toys

Figure 4-11. Schematic View of CYRF69103 Chip
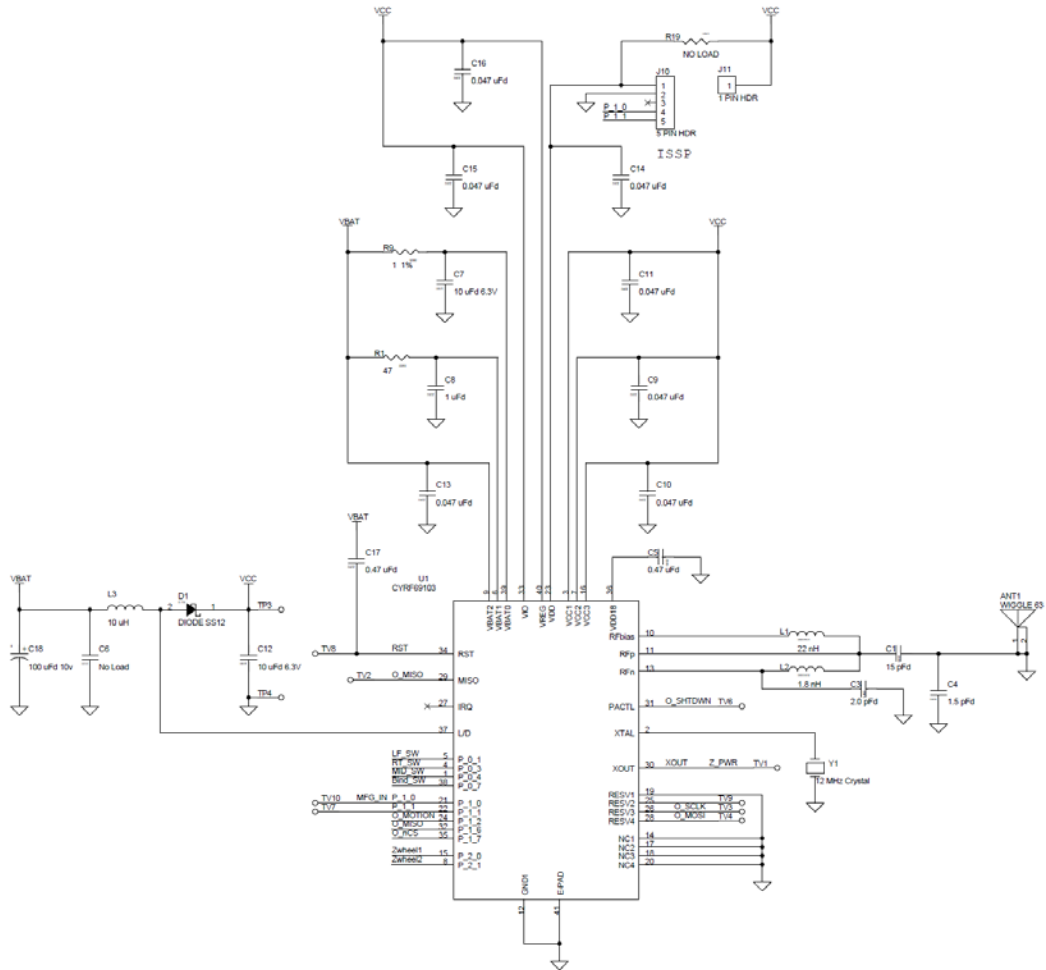
Table 4-2. Pin Details of CYRF69103 Chip

Pin	Name	Description
1	P0.4	Individually configured GPIO
2	XTAL	12 MHz crystal
3	VCC	2.4 V to 3.6 V supply. Connected to pin 40 (0.047 µF bypass)
4	P0.3	Individually configured GPIO
5	P0.1	Individually configured GPIO
6	Vbat1	Connect to 1.8 V to 3.6 V power supply, through 47 ohm series/1 µF shunt C
7	VCC	2.4 V to 3.6 V supply. Connected to pin 40 (0.047 µF bypass)
8	P2.1	GPIO. Port 2 Bit 1
9	Vbat2	Connected to1.8 V to 3.6 V main power supply, through 0.047 µF bypass C
10	RFbias	RF pin voltage reference
11	RFp	Differential RF to or from antenna
12	GND	GND
13	RFn	Differential RF to or from antenna
14	NC	No Connection

Table 4-2. Pin Details of CYRF69103 Chip

Pin	Name	Description
15	P2.0	GPIO
16	VCC	2.4 V to 3.6 V supply. Connected to pin 40 (0.047 µF bypass)
17	NC	No Connection
18	NC	No Connection
19	RESV	Reserved. Must connect to GND
20	NC	No Connection
21	P1.0	GPIO
22	P1.1	GPIO
23	VDD_micro	MCU supply connected to pin 40, max CPU 12 MHz
24	P1.2	GPIO
25	P1.3/nSS	Slave select
26	P1.4/SCK	SPI clock
27	IRQ	Radio Function Interrupt output, configure High, Low, or as Radio GPIO
28	P1.5 / MOSI	MOSI pin from microcontroller function to radio function
29	MISO	3-wire SPI mode configured as Radio GPIO. In 4-wire SPI mode sends data to MCU function
30	XOUT	Buffered CLK, PACTL_n or Radio GPIO
31	PACTL	Control for external PA or Radio GPIO
32	P1.6	GPIO
33	VIO	1.8 V to 3.6 V to main power supply rail for Radio I/O
34	RST	Radio Reset. Connected to pin 40 with 0.47 µF. Must have a RST=HIGH event the very first time power is applied to the radio otherwise the state of the radio control registers is unknown
35	P1.7	GPIO
36	VDD1.8	Regulated logic bypass. Connected to 0.47 µF to GND
37	L/D	Inductor/Diode connection for Boost. When Internal PMU is not being used connect L/D to GND
38	P0.7	GPIO
39	Vbat0	Connected to1.8 V to 3.6 V main power supply, through 0.047 µF bypass C
40	VREG	Boost regulator output voltage feedback
41	E-pad	Must be connected to ground
42	Corner Tabs	Do not connect corner tabs

4.2.1.2 *Optical Sensor*

The ADNS-3040 is an ultra low-power optical navigation sensor. It has a new, low-power architecture and automatic power management modes, making it ideal for battery- and power-sensitive applications such as cordless input devices.

The ADNS-3040 is capable of high-speed motion detection - up to 20 ips and 8G. In addition, it has an on-chip oscillator and LED driver to minimize external components.

Figure 4-12. Schematic View of Optical Sensor
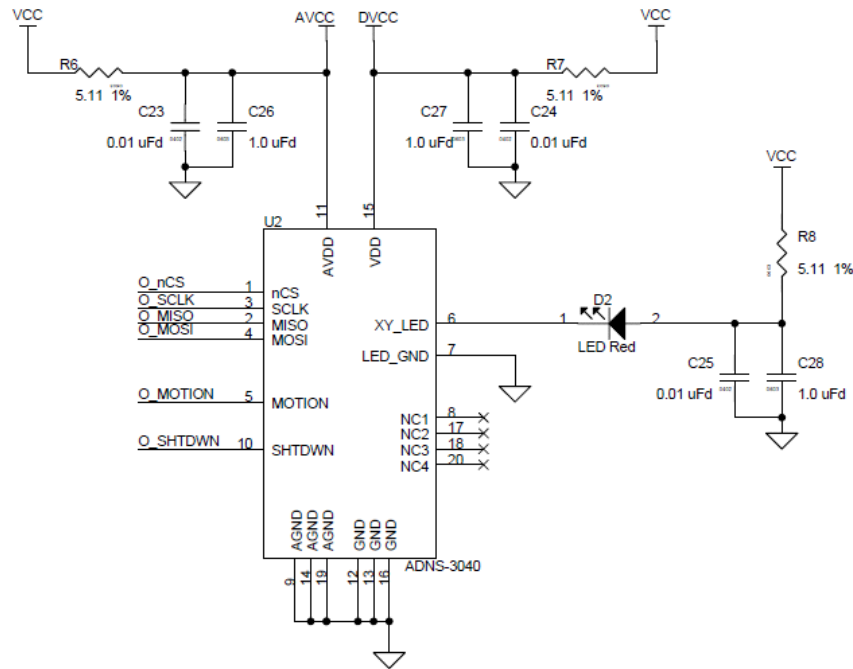
Table 4-3. Pin Details of ADNS-3040 Chip

Pin	Name	Description
1	NCS	Chip select (active low input)
2	MISO	Serial data output (Master In/Slave Out)
3	SCLK	Serial clock input
4	MOSI	Serial data input (Master Out/Slave In)
5	MOTION	Motion Detect (active low output)
6	XY_LED	LED control
7	LED_GND	Ground for LED current
8	NC	No Connection
9	AGND	Analog Ground
10	SHTDWN	Shutdown (active high input)
11	AVDD	Analog Supply Voltage
12	GND	Ground
13	GND	Ground
14	AGND	Analog Ground
15	VDD	Supply Voltage
16	GND	Ground
17	NC	No Connection
18	NC	No Connection
19	AGND	Analog Ground
20	NC	No Connection

4.2.1.3 ISSP Header

In-system serial programmer (ISSP) is used to program the device. Programming can be done using the MiniProg device.

Figure 4-13. Schematic View of ISSP Header
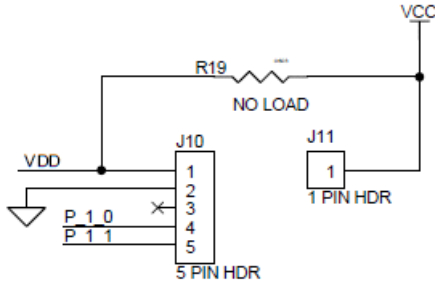
4.2.1.4 Bind Button

This button is used to bind the mouse with the bridge. When the bind button on the bridge is pressed, the bridge goes into Bind Mode. In Bind Mode, the bridge uses the Bind ID to communicate with the mouse to bind to the system.

Figure 4-14. Schematic View of Bind Button
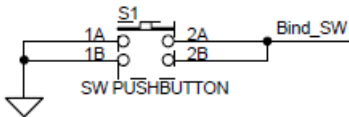
4.2.1.5 Wheel and Buttons

There are three buttons S2, S3, and S4 for left, middle, and right button operations of the mouse. The wheel with the encoder is used for scroll operation.

Figure 4-15. Schematic View of Buttons and Wheel
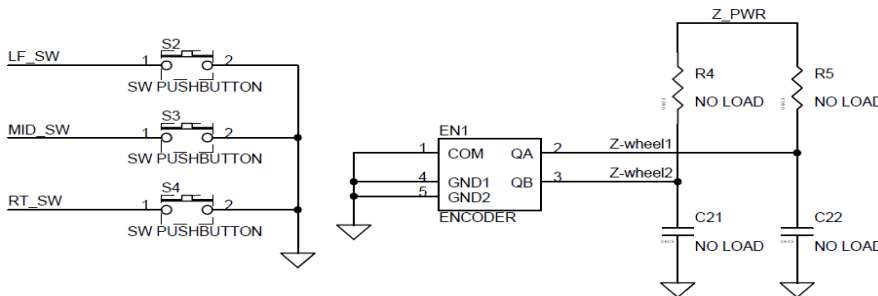
4.2.2 Power Supply System

The CY4672 mouse board has power sources from the battery supply terminal and ISSP on the board.

Figure 4-16. Mouse Board Power Supply System Structure
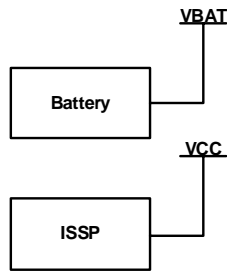
Figure 4-17. Schematic View of Power Supply System Structure for Mouse Board
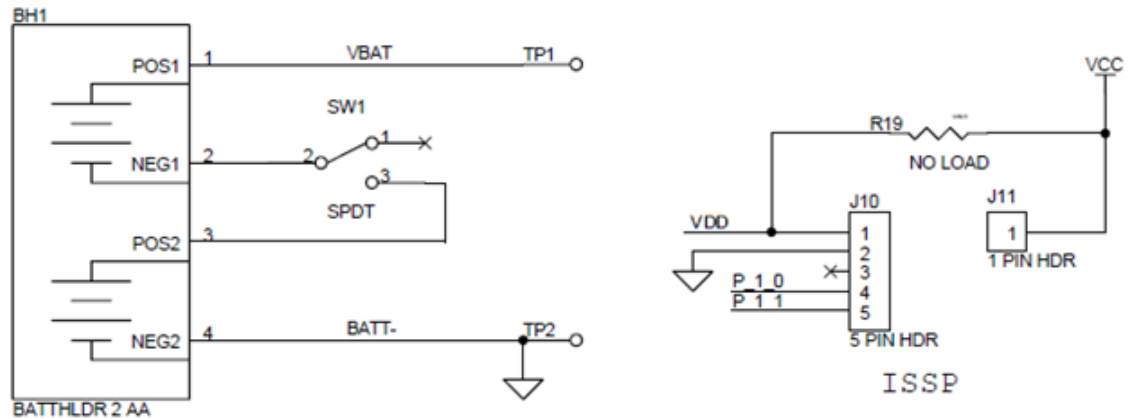
4.3   Keyboard

The CY4672 Reference Design Kit uses an enCoRe II LV controller for the RDK keyboard. The architecture is designed to be modular for extendibility and maintainability. It can also be easily ported from one hardware platform to another assuming the use of an enCoRe II LV microprocessor. While porting to another microprocessor requires more work, the hardware design is done to minimize use of advanced enCoRe II LV features to expedite this effort.

Design efforts are made to reduce the ON time of the microprocessor and radio to conserve battery life. This includes protocol optimizations along with using sleep features of the radio and enCoRe II LV microprocessor.

The keyboard design uses the BAT400D-7-F Schottky diode (D1) and CDH53100LC inductor (L3) for its boost circuitry. These low-cost components are used to reduce the overall system cost at the expense of lower boost efficiency. Preliminary characterization data shows a range of 68% to 81% efficiency for the 1.8 V to 2.7 V VBAT voltage range at different temperatures (–10 °C to 80 °C). Higher efficiency components such as the ones in the mouse design may be used at the expense of component costs and board size (these low cost components are smaller in size compared to the ones used in the mouse design.)

Figure 4-18. Keyboard Block Diagram
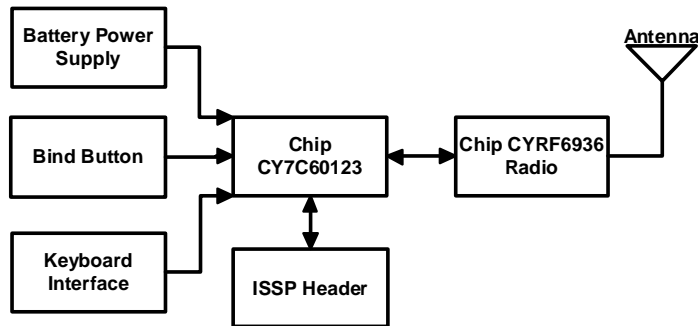
4.3.1   Functional Description

The following figure shows the different functional blocks on the CY4672 keyboard.

Figure 4-19. Functional Blocks of Keyboard PCB
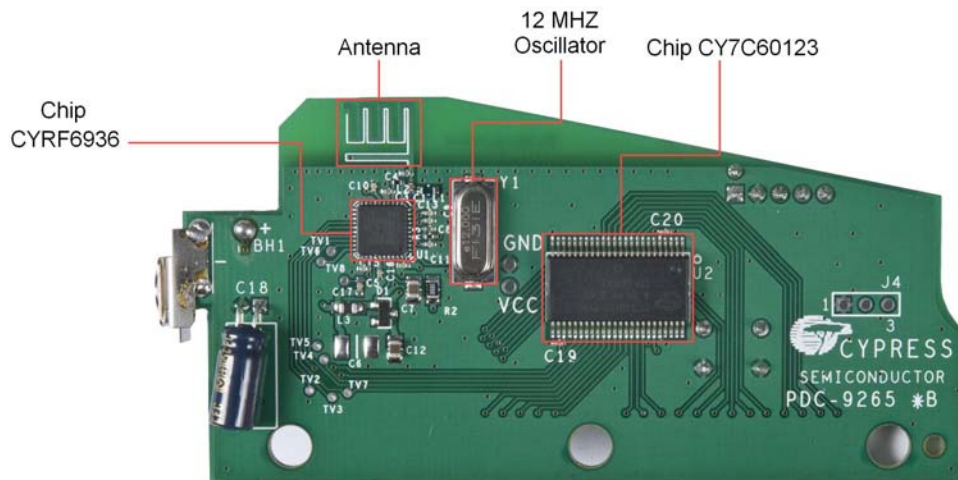
4.3.1.1   *CYRF6936 Chip*

The CYRF6936 WirelessUSB LP radio is a second generation member of the Cypress WirelessUSB Radio System-On-Chip (SoC) family. The CYRF6936 is interoperable with the first generation CYWUSB69xx devices. The CYRF6936 IC adds a range of enhanced features, including increased operating voltage range, reduced supply current in all operating modes, higher data rate options, reduced crystal start up, synthesizer settling, and link turnaround times.

The CYRF6936 chip operates with a 2.4 GHz Direct Sequence Spread Spectrum (DSSS) radio transceiver and operates in the unlicensed worldwide Industrial, Scientific, and Medical (ISM) band (2.400 GHz to 2.483 GHz). The operating current is 21 mA (transmit at –5 dBm). The transmit power is up to +4 dBm and a receive sensitivity of up to –97 dBm.

The CYRF6936 chip is targeted towards the following applications:

■ Wireless keyboards and mice

■ Wireless gamepads

■ Remote controls

■ Toys

■ VOIP and wireless headsets

- White goods
- Consumer electronics
- Home automation
- Automatic meter readers
- Personal health and entertainment

Figure 4-20. Schematic View of CYRF6936 Chip
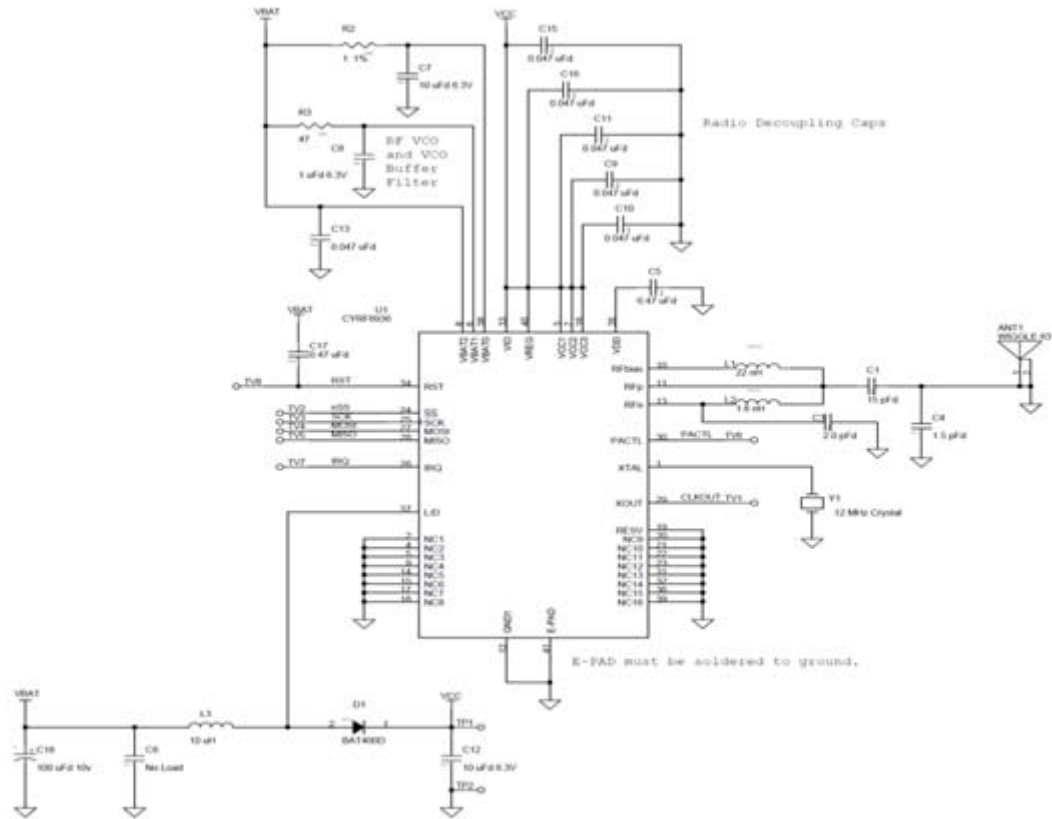
Table 4-4. Pin Details of CYRF6936 Chip

Pin	Name	Description
1	XTAL	12 MHZ crystal
2	NC	Connect to GND
3	VCC	VCC = 2.4 V to 3.6 V. Typically connected to VREG
4	NC	Connect to GND
5	NC	Connect to GND
6	VBAT(0-2)	VBAT = 1.8 V to 3.6 V. Main supply
7	VCC	VCC = 2.4 V to 3.6 V. Typically connected to VREG
8	VBAT(0-2)	VBAT = 1.8 V to 3.6 V. Main supply
9	NC	Connect to GND
10	RFBIAS	RF I/O 1.8 V reference voltage
11	RFP	Differential RF signal to and from antenna
12	GND	Ground

Table 4-4. Pin Details of CYRF6936 Chip

Pin	Name	Description
13	RFN	Differential RF signal to and from antenna
14	NC	Connect to GND
15	NC	Connect to GND
16	VCC	VCC = 2.4 V to 3.6 V. Typically connected to VREG
17	NC	Connect to GND
18	NC	Connect to GND
19	RESV	Must be connected to GND
20	NC	Connect to GND
21	NC	Connect to GND
22	NC	Connect to GND
23	NC	Connect to GND
24	SS	SPI enable, active LOW assertion. Enables and frames transfers
25	SCK	SPI clock
26	IRQ	Interrupt output (configurable active HIGH or LOW), or GPIO
27	MOSI	SPI data input pin (Master Out Slave In), or SDAT
28	MISO	SPI data output pin (Master In Slave Out), or GPIO (in SPI 3-pin mode). Tri-states when SPI 3PIN = 0 and SS is deasserted
29	XOUT	Buffered 0.75, 1.5, 3, 6, or 12 MHz clock, PACTL, or GPIO.Tri-states in sleep mode (configure as GPIO drive LOW)
30	PACTL	Control signal for external PA, T/R switch, or GPIO
31	NC	Connect to GND
32	NC	Connect to GND
33	VIO	I/O interface voltage, 1.8 V to 3.6 V
34	RST	Device reset. Internal 10 kohm pull down resistor. Active HIGH, connect through a 0.47 µF capacitor to VBAT. Must have RST = 1 event the first time power is applied to the radio. Otherwise the state of the radio control registers is unknown
35	VDD	Decoupling pin for 1.8 V logic regulator, connect through a 0.47 µF capacitor to GND
36	NC	Connect to GND
37	L/D	PMU inductor/diode connection, when used. If not used, connect to GND
38	VBAT(0-2)	VBAT = 1.8 V to 3.6 V. Main supply.
39	NC	Connect to GND
40	VREG	PMU boosted output voltage feedback

4.3.1.2   CY7C60123 Chip

enCoRe II low-voltage (enCoRe II LV) consists of an internal crystalless oscillator with support for optional external clock or external crystal or resonator with configurable I/O for real world interface without external components. It has an enhanced 8-bit microcontroller with Harvard architecture and a M8C CPU speed up to 12 MHz or sourced by an external crystal, resonator, or clock signal.

The CY7C60123 is targeted for the following applications:

■ PC wireless human interface devices - Mice (optomechanical, optical, trackball), keyboards, presenter tools

■ Gaming - Joysticks, gamepad

■ General-purpose wireless applications - Remote controls, barcode scanners, POS terminal, consumer electronics, toys

Figure 4-21. Schematic View of CY7C60123 Chip
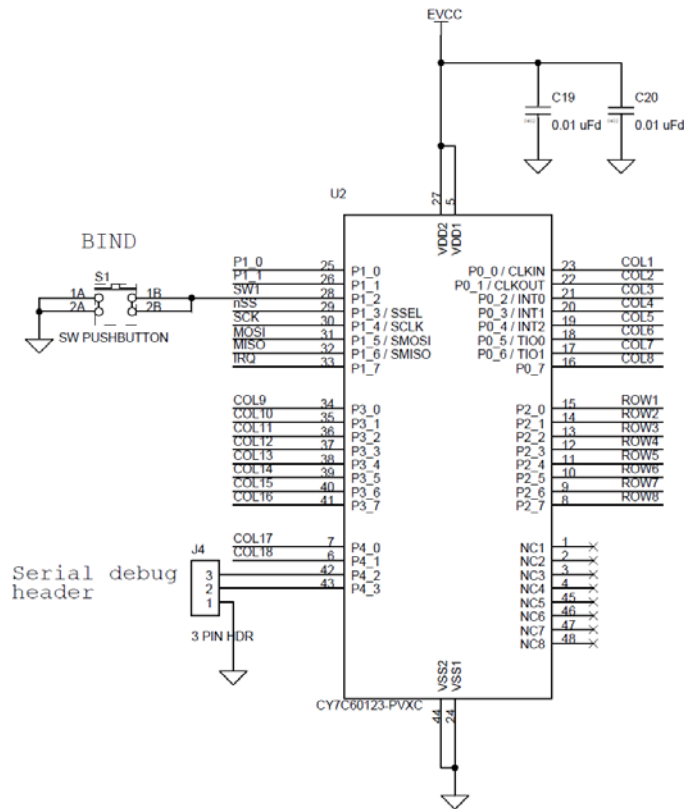
Table 4-5. Pin Details of CY7C60123 Chip

Pin	Name	Description
1	NC	No connect
2	NC	No connect
3	NC	No connect
4	NC	No connect
5	VDD	Power
6	P4.1	GPIO port 4-configured as a group (nibble)
7	P4.0	GPIO port 4-configured as a group (nibble)
8	P2.0	GPIO port 2-configured as a group (byte)
9	P2.1	GPIO port 2-configured as a group (byte)
10	P2.2	GPIO port 2-configured as a group (byte)
11	P2.3	GPIO port 2-configured as a group (byte)
12	P2.4	GPIO port 2-configured as a group (byte)

Table 4-5. Pin Details of CY7C60123 Chip

Pin	Name	Description
13	P2.5	GPIO port 2-configured as a group (byte)
14	P2.6	GPIO port 2-configured as a group (byte)
15	P2.7	GPIO port 2-configured as a group (byte)
16	P0.7	GPIO port 0 bit 7-Configured individually
17	P0.6/TIO1	GPIO port 0 bit 6-Configured individually Alternate function timer capture inputs or timer output TIO1
18	P0.5/TIO0	GPIO port 0 bit 5-Configured individually Alternate function timer capture inputs or timer output TIO0
19	P0.4/INT2	GPIO port 0 bit 4-Configured individually Optional rising edge interrupt INT2
20	P0.3/INT1	GPIO port 0 bit 3-Configured individually Optional rising edge interrupt INT1
21	P0.2/INT0	GPIO port 0 bit 2-Configured individually Optional rising edge interrupt INT0
22	P0.1/CLKOUT	GPIO port 0 bit 1-Configured individually On CY7C601xx, optional Clock Out when external oscillator is disabled or external oscillator output drive when external oscillator is enabled. On CY7C602xx, oscillator output when configured as Clock Out
23	P0.0/CLKIN	GPIO port 0 bit 0-Configured individually on CY7C601xx, optional Clock In when external oscillator is disabled or external oscillator input when external oscillator is enabled. On CY7C602xx, oscillator input when configured as Clock In
24	VSS	Ground
25	P1.0	GPIO port 1 bit 0 If this pin is used as a general-purpose output it draws current. It is, therefore, configured as an input to reduce current draw
26	P1.1	GPIO port 1 bit 1 If this pin is used as a general-purpose output it draws current. It is, therefore, configured as an input to reduce current draw
27	VDD	Power
28	P1.2	GPIO port 1 bit 2
29	P1.3/SSEL	GPIO port 1 bit 3-Configured individually Alternate function is SSEL signal of the SPI bus
30	P1.4/SCLK	GPIO port 1 bit 4-Configured individually Alternate function is SCLK signal of the SPI bus
31	P1.5/SMOSI	GPIO port 1 bit 5-Configured individually Alternate function is SMOSI signal of the SPI bus
32	P1.6/SMISO	GPIO port 1 bit 6-Configured individually Alternate function is SMISO signal of the SPI bus
33	P1.7	GPIO port 1 bit 7-Configured individually TTL voltage threshold
34	P3.0	GPIO port 3-configured as a group (byte)
35	P3.1	GPIO port 3-configured as a group (byte)
36	P3.2	GPIO port 3-configured as a group (byte)
37	P3.3	GPIO port 3-configured as a group (byte)
38	P3.4	GPIO port 3-configured as a group (byte)
39	P3.5	GPIO port 3-configured as a group (byte)
40	P3.6	GPIO port 3-configured as a group (byte)
41	P3.7	GPIO port 3-configured as a group (byte)
42	P4.2	GPIO port 4-configured as a group (nibble)
43	P4.3	GPIO port 4-configured as a group (nibble)

Table 4-5. Pin Details of CY7C60123 Chip

Pin	Name	Description
44	VSS	Ground
45	NC	No connect
46	NC	No connect
47	NC	No connect
48	NC	No connect

4.3.1.3 Keyboard Interface

A 26-pin keyboard interface J1 is present on the board, which takes input from 8 rows and 18 columns with an input from 101 keys.

Figure 4-22. Schematic View of Keyboard Interface
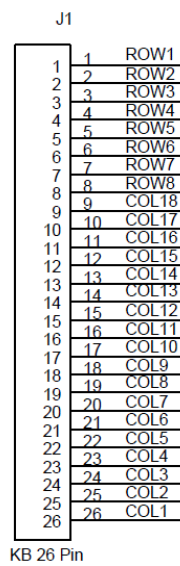
4.3.1.4 ISSP Header

ISSP header (J2) is used to program the device. Programming can be done using the MiniProg device.

Figure 4-23. Schematic View of ISSP Header
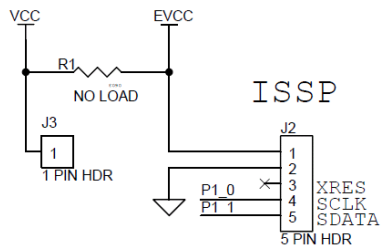
4.3.1.5   Bind Button

This button is used to bind the keyboard with the bridge. When the bind button on the bridge is pressed, the bridge goes into Bind Mode. The bind button on the keyboard is then pressed to bind the keyboard.

Figure 4-24. Schematic View of Bind Button
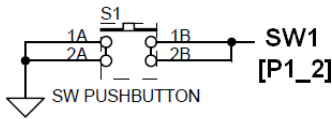
4.3.2   Power Supply System

The CY4672 keyboard has power sources from the battery supply terminal and ISSP header on the board.

Figure 4-25. Keyboard Power Supply System Structure
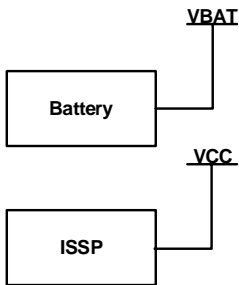
Figure 4-26. Schematic View of Power Supply System Structure for Keyboard
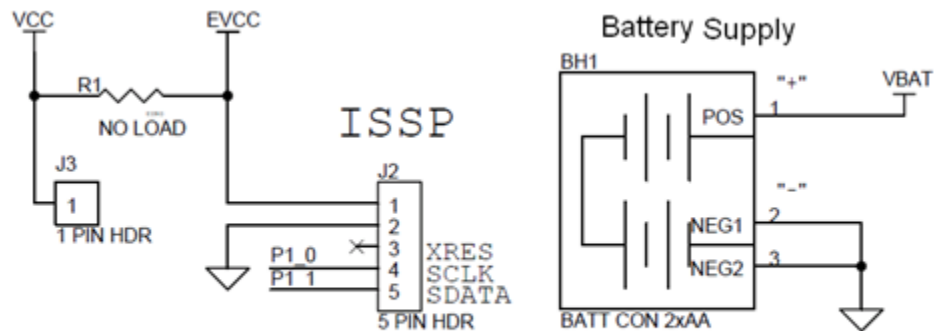
5. Code Examples

All code examples are available at: <install directory>\Cypress\CY4672 PRoC LP RDK\<ver>\Firmware\

5.1 Project 1- PRoC_LP_RDK_Bridge

5.1.1 Project Description

This code example demonstrates the features and functions offered by the bridge. The bridge continuously checks the USB idle timer, received packet, bind button, and USB suspend. The functions include:

■ Check the Received Packet - When the bridge receives a valid packet, it parses the packet. If it is a data packet, the bridge formats and sends a USB packet to the USB host. If it is a connect request with an approved device or a ping request, the bridge sends a response correspondingly.

■ Check the Bind Button: The bridge checks the bind button frequently. If this button is pressed, the bridge goes into the bind state.

■ Check the USB Suspend: When suspended, the bridge supports remote wakeup by intermittently turning the radio on when the sleep timer interrupt occurs, checking for valid data from the HID devices, and then turning the radio off again if no HID traffic is detected.

There are two architectural views of the bridge. The first is a microcontroller configuration view of user modules inside the controller. This architecture and configuration is best viewed in PSoC Designer when the project is loaded. The second view is a logical organization of the source code modules that make up the bridge application code and other support modules.

The following table shows the ROM/RAM usage. The top part exhibits the total ROM/RAM usage for basic functions, which disables all the build options. The bottom part exhibits the ROM/RAM usage for individual build options.

Table 5-1. ROM/RAM Usage

	Total ROM (Bytes)	Total RAM (Bytes)
Basic Functions	7251	170
Build Option	ROM Usage (Bytes)	RAM Usage (Bytes)
ENCRYPT_DATA TEA*	815	29
ENCRYPT_DATA AES	990	35
MFG_TEST_CODE	541	0
MFG_TX_MODES	651	2
BACK_CHANNEL_SUPPORT	147	1

Note* The ENCRYPT_DATA option requires 64 bytes of extra ROM space to store the nonvolatile session key.

5.1.2   Device Configurations

The PRoC LP Programmable Radio on Chip is configured using the Device Editor in PSoC Designer. The bridge uses the SPI Master, USB Device, and 1 Millisecond Interval Timer User Modules. The SPI Master User Module is used by firmware to communicate with the LP radio module. The USB Device User Module allows the bridge to operate as a low-speed USB device. The 1 Millisecond Interval Timer User Module is used for timing. Figure 5-1 shows the Device Editor with user module mapping.

Figure 5-1. Device Configuration for Project1
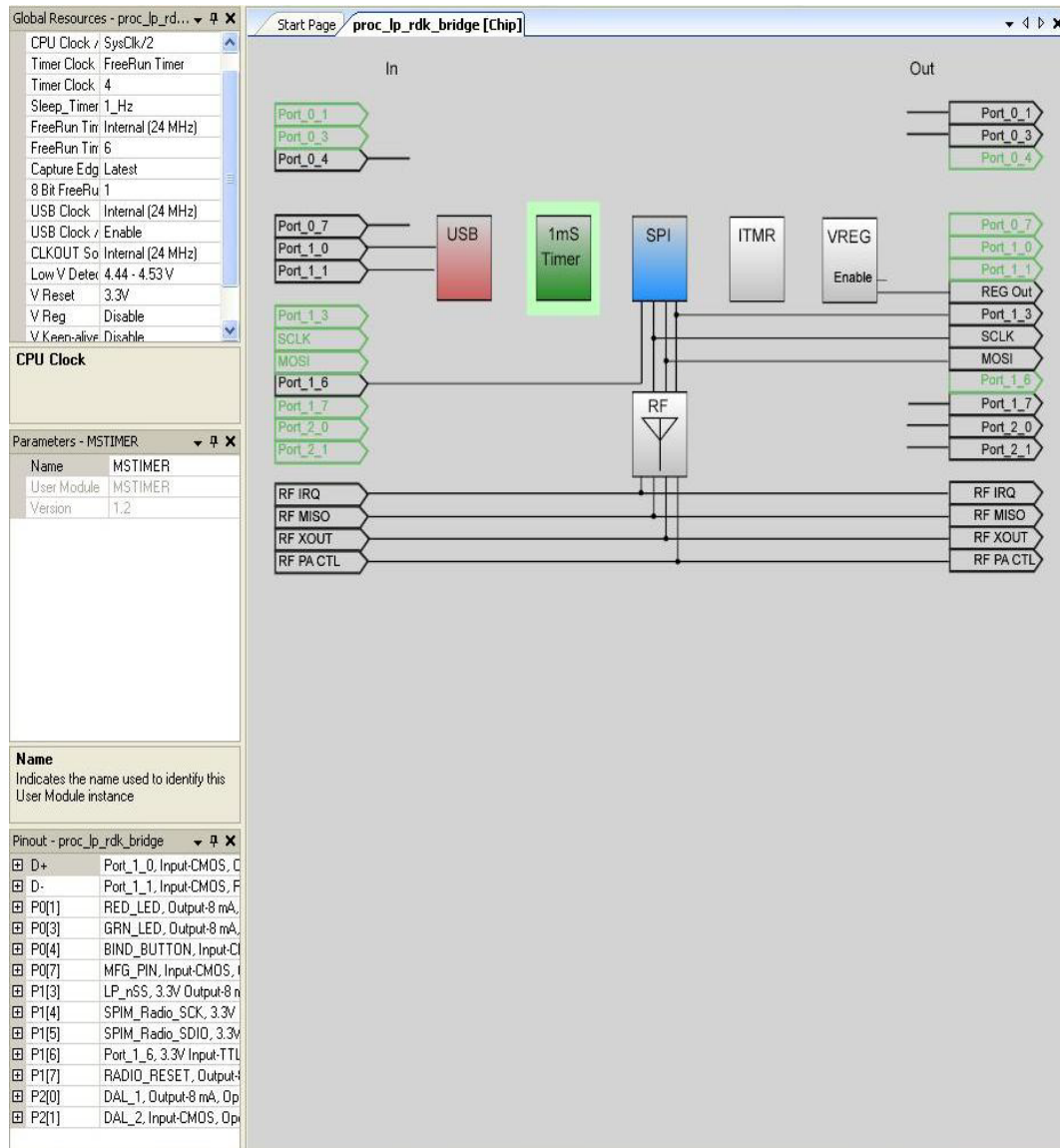
5.1.2.1   Global Configuration

Following is a description of the global resources that are configured for the PRoC LP CYRF69213. Take care when modifying these values, because they affect the following user modules.

- CPU Clock: This parameter is set to Internal (24 MHz). To run the CPU at 12 MHz, the CPU Clock/N must be set to '2'. This operating frequency provides for faster code execution.
- CPU Clock / N: This parameter is set to '2' to provide a 12 MHz clock.
- Timer Clock: This parameter is set to 'FreeRun Timer'.
- Timer Clock /N: This parameter is set to '4'.
- Sleep Timer: This parameter is set to '1_Hz'.
- FreeRun Timer: This parameter is set to 'Internal (24 MHz)'.
- FreeRun Timer /N: This parameter is set to '6'.
- Capture Edge: This parameter is set to 'Latest'.
- 8 Bit Capture Prescaler: This parameter is set to '1'.
- USB Clock: This parameter is set to 'Internal (24 MHz)'.
- USB Clock /2: This parameter is set to 'Enable'.
- CLKOUT Source: This parameter is set to 'Internal (24 MHz)'.
- Low V Detect: This parameter is set to 4.44 V – 4.53 V.
- V Reset: This parameter is set to '3.3 V'.
- V Reg: This parameter is set to 'Disable' and the VReg is enabled in the application code.
- V Keep-alive: This parameter is set to 'Disable'.
- Watchdog Enable: This parameter should be set to 'Enable', but may be set to 'Disable' for debug purposes.

5.1.2.2   SPI Master User Module

The SPI Master User Module is used to communicate with the radio transceiver. The radio transceiver supports leading edge data latching, non-inverted clock, and MSB first transmission as defaults. A clock divisor of 6 is chosen, which generates an SPI clock of 2 MHz. The interrupt API to this module is not used.

In the PRoC RDK bridge design, the bridge implements the "3 wire" SPI; therefore, the microcontroller's MISO and the radio MISO can be used as GPIOs. Also, the IRQ pin function is multiplexed on to the MOSI pin to save the GPIO pin.

5.1.2.3   USB Device User Module

The USB Device User Module handles the enumeration and data transfers over USB endpoints.

5.1.2.4   1 Millisecond Interval Timer User Module

The 1 Millisecond Interval Timer User Module is used to determine when a USB suspend has occurred, LED ON/OFF duration timing, RSSI checking, and others.

5.1.2.5   Flash Security

The bridge project within PSoC Designer has a file called *FlashSecurity.txt*. This file specifies access rules to blocks of the flash ROM. See the documentation listed at the top of the file for definitions. This file is shipped with a single change from its default configuration. The block starting at address 0x1FC0 is changed from W: Full (Write protected) to U: Unprotected. This location of flash is dedicated to saving the nonvolatile session key for the encrypt code module and the device flag for KISSBind.

5.1.3   Model

Figure 5-2. Firmware Architecture Model
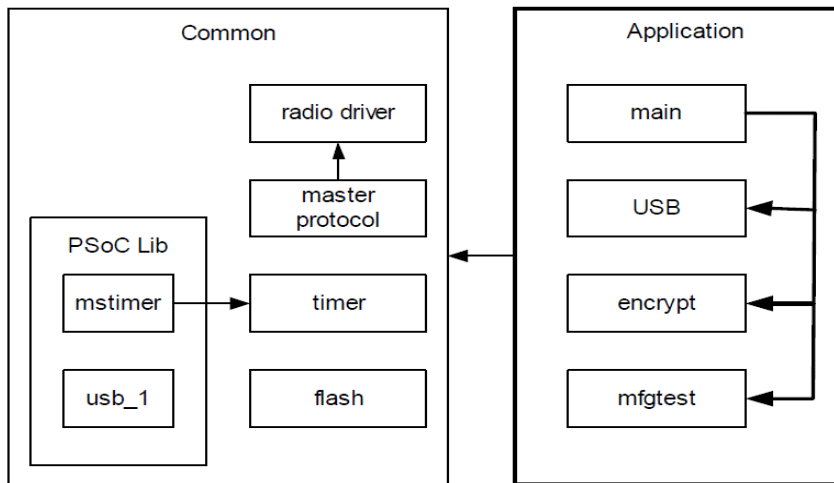
The bridge firmware is partitioned into two logical groups. The Common group is a collection of code modules that provide the underlying support for the application. This group provides services such as, radio protocol, radio driver, USB, timing, flash access, SPI, and interrupts.

The Application group implements the core functionality and features of the RDK wireless bridge. This includes USB HID packet formatting and reporting, encryption, and manufacturing test mode. The code modules for each of these groups are described here.

The following module descriptions have corresponding <module name>.c and <module name>.h source code files. The module API and definitions are exported in the header file while the module implementation and local definitions are contained in the C file.

5.1.3.1   Common Code

The modules consist of the common code logical grouping.

- PSoC Generated Library Code: There are currently only three files generated by PSoC Designer that are modified when using the application. A minimal amount of code is added to these modules in user protected areas that are preserved across code generation.
    - ❏ USB include (USB_1.inc): This file includes the additional code for the Battery Level and Link Quality software application in USB_1_cls_hid.asm.
    - ❏ USB HID Class Module (USB_1_cls_hid.asm): The additional user code provides support for the Battery Level and Link Quality software application.
    - ❏ 1 Millisecond Interval Timer Interrupt Module (MSTIMER.asm): The additional user code decrements application countdown timers and checks for USB activity to detect a USB suspend condition.
- Flash: This module includes routines to write to the PRoC LP flash.
- Timer: This module includes busy wait time routines.
- Radio Driver: The radio driver module is a low level module providing basic radio communication and configuration. Its application is such that it may not be changed by the firmware developer. It provides an interface to read/write radio registers, set PN codes and initialize the radio, and transmit or receive packets. See the Radio Driver documentation for details.

■ Master Protocol: The module includes LP RDK master protocol routines to handle ping, button bind, channel agility, and data packets. This module depends on the radio driver to send and receive formatted packets and the flash module.

5.1.3.2   Application Code

The group of modules that make up the application code is responsible for implementing the bridge functionality and behavior.

Bridge Module

The bridge module is the controlling code for the application. It is responsible for implementing various features and functions offered by the bridge. The function main() is the entry point for the bridge application. This function is called from the *boot.asm* file. The bridge first initializes all the application modules and then initializes the master_protocol module. There is an order dependency for some of these, so take care when modifying the bridge_init() function. For example, other modules depend upon the timer facility running to perform initialization. When each module is initialized, the application checks for entry to the manufacturing test mode. If the manufacturing test mode is not indicated, then normal bridge operation begins.

The bridge continuously checks the USB idle timer, received packet, bind button, and USB suspend.

■ Check the USB Idle Timer

The check_usb_idle() function is called within the main() function to properly handle the USB Set_Idle command. The USB Set_Idle command from the host PC is used to silence the keyboard or mouse report until a new event occurs or the specified amount of time passes. If the host PC's Set_Idle command sets the Idle Duration to '0', the keyboard or mouse endpoint inhibits reporting forever, only reporting when a change is detected in the report data. This causes the bridge to NAK any polls on the endpoint while its current report remains unchanged. If the Set_Idle command sets the Idle Duration to a non-zero number, a single report is generated by the endpoint if the given time duration elapses with no change in report data (see the HID Specification for more information on this topic).

The check_usb_idle() function also checks the timeout for down key and 'keep alive' packet. A 'keep alive' packet is transmitted every 65 ms during the time a key is pressed, so that the bridge can detect if the RF link is lost, and in that unlikely case, the bridge inserts a 'key up' event, to prevent a 'stuck key' state being transmitted to the PC. The number of milliseconds before upkey reports are generated is defined by DOWNKEY_TIME_OUT.

■ Check the Received Packet

When the bridge receives a valid packet, it parses the packet. If it is a data packet, the bridge formats and sends a USB packet to the USB host. If it is a connect request with an approved device or a ping request, the bridge sends a response correspondingly.

■ Check the Bind Button

The bridge checks the bind button frequently. If this button is pressed, the bridge goes into the bind state.

■ Check the USB Suspend

The check_usb_suspend() monitors the USB suspend condition on the USB bus and puts the system into a low-power state when no bus activity is observed for 3 ms. When suspended, the bridge supports remote wakeup by intermittently turning the radio on when the sleep timer interrupt occurs, checking for valid data from the HID devices, and then turning the radio off again if no HID traffic is detected.

USB Module

This module parses the radio packets, builds the appropriate keyboard and mouse USB packets, and loads these packets into the endpoints.

Mfgtest Module

The manufacturing test module may be conditionally compiled to provide manufacturing test support. The module configures the radio for reception and then enters a loop waiting for command packets to be sent from the tester. The test echoes all echo command packets appended with the number of invalid bits received and all other 'valid' command packets (no invalid bits). The manufacturing test code can only be exited by cycling power. The manufacturing test code in this bridge is compatible with Cypress's CY3631 Manufacturing Test Kit.

The manufacturing test mode on the PRoC LP RDK bridge can be entered by three different methods depending on the compile-time configuration.

- Method 1: Press the bind button during dongle insertion into the USB host to enter the manufacturing test mode.
- Method 2: Force an SE1 condition (D+ and D– are both high) on the USB bus and at the same time apply power to the bridge.
- Method 3: Ground the P0.7 pin during dongle insertion into the USB host to enter the manufacturing test mode.

Encrypt Module

This module may be conditionally compiled to provide encryption/decryption support. Encrypted data transfers are typically used between RDK keyboard devices and the RDK bridge. Contact Cypress Applications support for the encryption source code.

5.1.3.3   Configuration Options

All configuration options for the application are available in the config.h file; some of them are defined in the Project > Setting > Compiler > Macro defines. Each option is explained here and can be changed to values that meet the developer's needs.

- MFG_TEST_CODE

  This configuration definition is used to selectively compile the manufacturing test code. The manufacturing test code in this bridge is compatible with Cypress's CY3631 Manufacturing Test Kit. See the Mfgtest Module for a description of how this test mode is executed. The CY3631 Manufacturing Test Kit documentation has a description of the test operation.

- MFG_TX_MODES

  When the MFG_TEST_CODE is defined, the definition of this name adds in a carrier and random data TX test option. See the Mfgtest Module for more information on these TX modes.

- MFG_ENTER_BY_PIN

  This configuration definition is used to selectively compile a method to enter the manufacturing test code. When this value is defined, the manufacturing test code may be executed by grounding a specific pin, when inserting the LP RDK bridge into a powered USB port or applying external power.

- MFG_ENTER_BY_BUTTON

  This configuration definition is used to selectively compile a method to enter the manufacturing test code. When this value is defined, the manufacturing test code may be executed by holding the bind button, when inserting the LP RDK bridge into a powered USB port or applying external power.

- **MFG_ENTER_BY_USBSE1**

  This configuration definition is used to selectively compile a method to enter the manufacturing test code. When this value is defined, the manufacturing test code may be executed by causing a USB SE1 condition on the D+ and D– signals, when inserting the LP RDK bridge into a powered USB port or applying external power.

- **ENCRYPT_TEA**

  This configuration definition is used to selectively compile TEA encryption for the bridge. Contact Cypress Applications support for the encryption source code.

- **ENCRYPT_AES**

  This configuration definition is used to selectively compile AES encryption for the bridge. Contact Cypress Applications support for the encryption source code.

- **GREEN_LED_ON_TIME**

  This configuration definition defines the number of milliseconds the green LED stays on after a valid USB report is loaded in an endpoint.

- **DOWNKEY_TIME_OUT**

  This configuration definition defines the number of milliseconds before upkey reports are generated by the bridge in the absence of valid packets from an attached keyboard device.

- **BACK_CHANNEL_SUPPORT**

  This configuration definition is used to selectively compile the Back Channel Data Support feature. See Back Channel Data Support on page 100 for more information.

- **MASTER_PROTOCOL**

  This configuration definition is used to select the Master radio protocol or Slave radio protocol. For the bridge application, it should be defined.

- **PAYLOAD_LENGTH**

  This configuration definition is used to define the payload length. For the bridge application, it should be defined as 8.

- **POWER_BIND**

  This configuration definition is used to selectively compile Power Bind feature. When the Power Bind feature is selected, the bridge enters the bind mode twice at power up. Each time the bridge stays in bind mode for 1.5 seconds, and if a device is in the bind mode during this time, the device will be bound to this bridge.

- **KISS_BIND**

  This configuration definition is used to selectively compile the Enhanced KissBind feature. See Enhanced KISSBind on page 98 for more information. The bridge can be unbound by holding the bind button for 5 seconds. After being unbound, the bridge enters an infinite loop and the red LED is always on until it is unplugged from and plugged into a host PC.

  After being unbound, the device bound flags are cleared, and the HIDs can be bound to this bridge by KissBind.

- **RSSI_QUALIFY**

  This configuration definition is used to enable the RSSI qualification for the Enhanced KissBind. Only if the RSSI reading is above KISS_BIND_RSSI_THRESHOLD, can the KissBind request/ response be accepted by the Bridge/Devices.

- **PROMISCUOUS_MODE**

  This configuration definition is used to enable the Promiscuous mode qualification for the Enhanced KissBind. With this mode, multiple mice or keyboards can be bound to one bridge.

■ DAL_ENABLE

This configuration definition is used to enable Microsoft's Direct Application Launch (DAL) feature. When this feature is enabled, the DAL1 LED is turned on by holding the F11 key; the DAL2 LED is turned on by holding the F12 key. Direct Application Launch is a new feature that the Windows Vista operating system provides built-in support, for a fast system startup experience. More information on this can be found on Microsoft's Windows Hardware Developer Central website (http://msdn.microsoft.com/en-us/windows/hardware/gg463078.aspx).

5.1.3.4   Platform and Architecture Portability

The bridge firmware is designed to use the hardware features of the PRoC LP such as USB. Porting the code to another microprocessor architecture may require modification of the existing code to support the different processor specific features.

5.1.3.5   Initialization

The initialization of the PRoC LP chip is done by code that is generated in *boot.asm* by PSoC Designer. The module boot.asm calls main when the PRoC LP is configured and initialized.

Main initializes the components of the bridge along with the radio modules. The bridge firmware enters a loop to receive and handle radio packets and generate USB packets.

5.1.3.6   Wireless Protocol Data Payload

The RDK HID protocol is optimized to reduce the 'on time' of the radio, which equates to reduced power consumption on the LP devices. Refer to the RDK keyboard and RDK mouse sections for radio packet format details.

5.1.3.7   Suspend and Remote Wakeup

To meet the USBIF compliance requirements regarding power consumption during suspend state, the PRoC LP RDK bridge must reduce the over all power consumption to less than 500 µA if Remote Wakeup is not enabled (Remote Wakeup is the device ability to wake up a suspended PC with user input such as a key press, mouse movement, and others). Because the PRoC LP RDK is not configured to wake up the suspended host PC, the entire bridge must go into deep sleep state to conserve power. Only bus activity from the host PC can bring the bridge back to normal operation.

If Remote Wakeup is enabled, the bridge may draw up to 2.5 mA in suspend state. This requires that the radio circuitry be OFF most of the time. It is necessary to periodically turn the radio on to sense activity from the PRoC LP mouse or keyboard (and thereby know when to wake the host). The wake up period is configurable and is set to 1 second (see Register OSC_CR0 setting). Increasing the wakeup interrupt frequency results in a faster response to the user wakeup events at the expense of a slightly higher than average sleep current.

Table 5-2. Bridge Average Icc in Suspend State

Parameter	Icc	Units
Bridge Average Suspend Power Consumption–REMOTE WAKE UP ENABLED	1.44	mA
Bridge Average Suspend Power Consumption–REMOTE WAKE UP DISABLED	0.3	mA

5.1.3.8   Interrupt Usage/Timing

The polling method is used for the bind button.

5.1.3.9   Code Performance Analysis

A keyboard report processing is used to analyze the code performance. A typical keyboard report processing contains the following steps:

■ The bridge receives the keyboard report packet and process the packet. This step takes 108 μs.

■ MCU calls function handle_keyboard_report() to format USB packet and load this packet into the endpoint buffer. This function consumes 118 μs.

As a result, it takes 226 μs for the bridge to process a keyboard report.

5.1.3.10   USB Interface

USB Descriptors: The USB descriptors can be viewed/edited with the USB Setup Wizard.

■ Device/Config Descriptors

Figure 5-3. USB Device/Config Descriptors

■ Keyboard HID Report Descriptor: The keyboard HID report descriptor defines a Boot Protocol keyboard. This enables a LP RDK keyboard with the LP RDK bridge to work on different BIOS versions that do not correctly support the USB Report Protocol. Only standard 101(104) keys are sent using this format over endpoint 1.

Figure 5-4. Keyboard HID Report Descriptor (Endpoint 1)
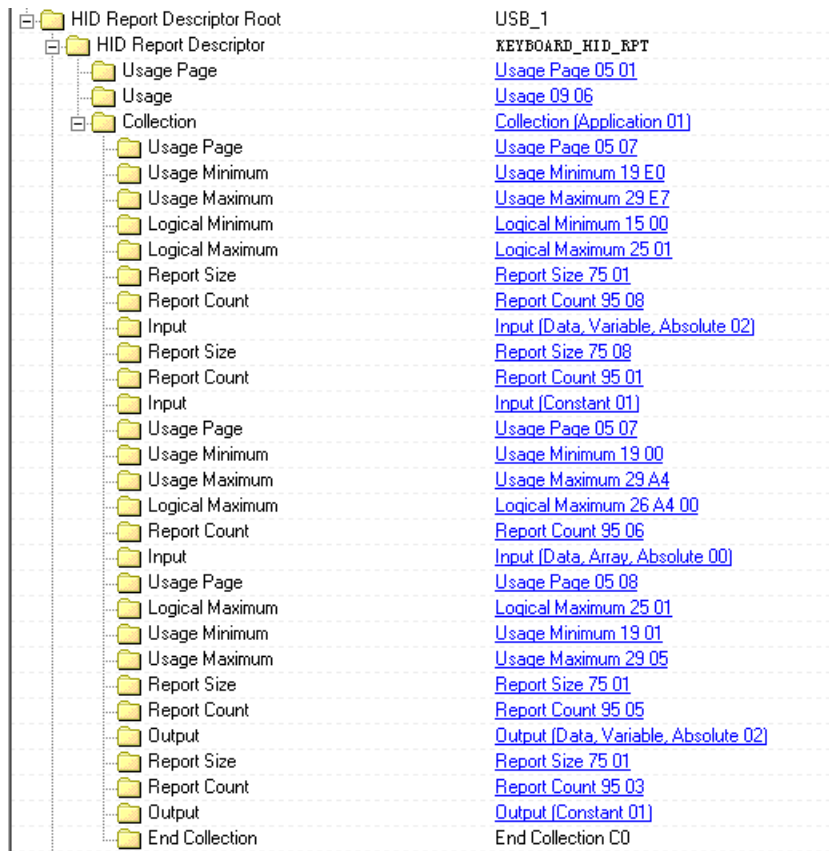
■ Mouse/Keyboard HID Report Descriptor: The mouse/keyboard HID Report Descriptor uses report protocol format with a unique report ID for each report. Mouse data uses Report ID 1. The mouse report include delta x, delta y, and scroll wheel data.

Figure 5-5. Mouse HID Report Descriptor (Report ID 1 - Endpoint 2)

Keyboard multimedia keys use Report ID 2.

Figure 5-6. Keyboard's MM Keys HID Report Descriptor (Report ID 2 - Endpoint 2)

Keyboard power keys use Report ID 3.

Figure 5-7. Keyboard's Power Keys HID Report Descriptor (Report ID 3 - Endpoint 2)


```
Usage Page                    Usage Page 05 01
Usage                         Usage 09 80
Collection                    Collection (Application 01)
  Report ID                   Report ID 85 03
  Usage Minimum               Usage Minimum 19 81
  Usage Maximum               Usage Maximum 29 83
  Logical Minimum             Logical Minimum 15 00
  Logical Maximum             Logical Maximum 25 01
  Report Size                 Report Size 75 01
  Report Count                Report Count 95 03
  Input                       Input (Data, Variable, Absolute 02)
  Report Count                Report Count 95 05
  Input                       Input (Constant 01)
  End Collection              End Collection C0
```

Report ID 4 is used to send the mouse battery level and link quality report.

Figure 5-8. Mouse's Battery/Link Quality Report Descriptor (Report ID 4 - Endpoint 2)


```
Usage Page                    Usage Page 06 01 FF
Usage                         Usage 09 02
Collection                    Collection (Application 01)
  Report ID                   Report ID 85 04
  Report Count                Report Count 95 01
  Report Size                 Report Size 75 08
  Logical Minimum             Logical Minimum 15 01
  Logical Maximum             Logical Maximum 25 0A
  Usage                       Usage 09 20
  Feature                     Feature (Constant 03)
  Logical Maximum             Logical Maximum 25 4F
  Usage                       Usage 09 21
  Feature                     Feature (Constant 03)
  Logical Maximum             Logical Maximum 25 30
  Usage                       Usage 09 22
  Feature                     Feature (Constant 03)
  Report Size                 Report Size 75 10
  Usage                       Usage 09 23
  Feature                     Feature (Constant 03)
  Usage                       Usage 09 24
  Feature                     Feature (Constant 03)
  End Collection              End Collection C0
```

Report ID 5 is used to send the keyboard battery level and link quality report.

Figure 5-9. Keyboard's Battery/Link Quality Report Descriptor (Report ID 5-Endpoint 2)


```
Usage Page                    Usage Page 06 01 FF
Usage                         Usage 09 01
Collection                    Collection (Application 01)
  Report ID                   Report ID 85 05
  Report Count                Report Count 95 01
  Report Size                 Report Size 75 08
  Logical Minimum             Logical Minimum 15 01
  Logical Maximum             Logical Maximum 25 0A
  Usage                       Usage 09 20
  Feature                     Feature (Constant 03)
  Logical Maximum             Logical Maximum 25 4F
  Usage                       Usage 09 21
  Feature                     Feature (Constant 03)
  Logical Maximum             Logical Maximum 25 30
  Usage                       Usage 09 22
  Feature                     Feature (Constant 03)
  Report Size                 Report Size 75 10
  Usage                       Usage 09 23
  Feature                     Feature (Constant 03)
  Usage                       Usage 09 24
  Feature                     Feature (Constant 03)
  End Collection              End Collection C0
```

Keyboard Report Format: The keyboard standard keys information is sent to the host PC via the data endpoint 1. The keyboard multimedia keys and power keys information is sent to the host PC via the data endpoint 2 using Report ID (the first byte in the report). The mouse uses Report ID 1. The keyboard multimedia keys use Report ID 2. The keyboard power keys use Report ID 3. The formats of the keyboard report are shown in Figure 5-10.

Figure 5-10. Keyboard Report Format

Right GUI	Right Alt	Right Shift	Right Ctrl	Left GUI	Left Alt	Left Shift	Left Ctrl
Reserved							
Standard Key 1							
Standard Key 2							
Standard Key 3							
Standard Key 4							
Standard Key 5							
Standard Key 6							
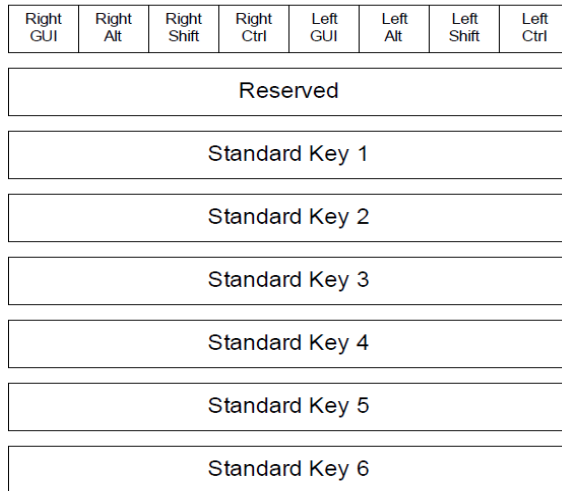
Keyboard Endpoint (EP1)

Figure 5-11. Multimedia and Power Keys Report Format

Mouse Endpoint (EP2)   Mouse Endpoint (EP2)

Report ID 2	Report ID 3
Multimedia Key	Power Key
Multimedia Key	Power Key
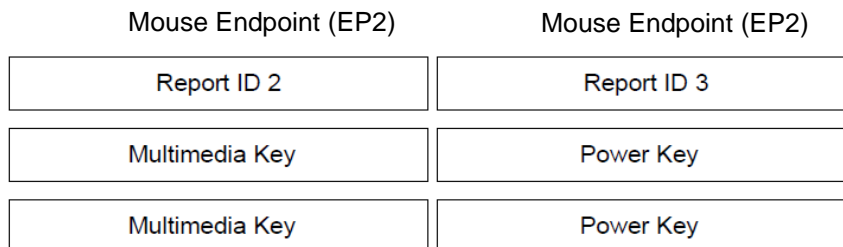
Mouse Report Format: The mouse data is sent over the data endpoint 2 using Report ID 1. The format of the mouse report is shown in Figure 5-12.

Figure 5-12. Mouse Report Format

Report ID 1			
Unused	Middle Button	Right Button	Left Button
X			
Y			
Z Wheel			
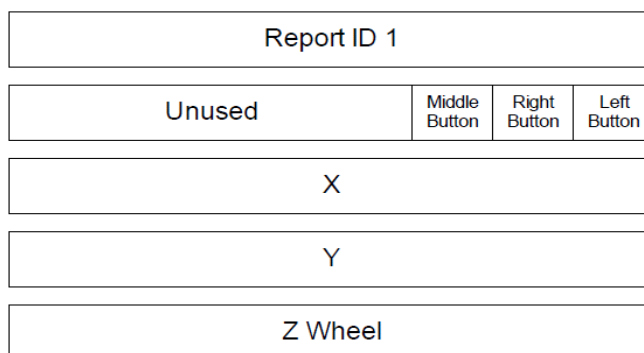
Battery Level and Link Quality Reports: The WirelessUSB LP bridge implements a mechanism to report the radio parameters of attached HID devices via the USB control endpoint. The code for this functionality is located in the user custom code section of the user module source file usb_1_cls_hid.asm.

The RadioParams HID report is a vendor-defined HID report for communicating several radio parameters of the WirelessUSB LP HID devices.

The HID Report Page is defined as:

Cypress WirelessUSB HID RadioParams Report Page (0xFF01 - Vendor Defined)

Table 5-3. USB Report Usage IDs

Usage ID	Usage Name
0x00	Undefined
0x01	WirelessUSB keyboard
0x 02	WirelessUSB mouse
0x03-0x1F	RESERVED
0x 20	Battery Level
0x 21	WirelessUSB Channel
0x 22	WirelessUSB PN Code
0x 23	Corrupt Packets
0x 24	Packets Transferred

The RadioParams Report is 8 bytes long and has the 6 data fields listed in Table 5-4.

Table 5-4. USB Report Format

Byte	Use	Range
0	Report ID #	0x04
1	Battery Level	0 – 0x0A
2	Channel #	0 – 0x4D
3	PN Code	0 – 0x30
4-5	Corrupt Packets	0 – 0xFFFF
6-7	Packets Transferred	0 – 0xFFFF

- Requesting a New Battery Reading

When the bridge receives a control endpoint request from the host with the following parameters, it returns an 8-byte RadioParams report over the control endpoint. An attached LP device should send an updated battery report whenever a reconnect or a change in the battery level occurs.

Control endpoint request for new battery reading.

Table 5-5. USB Set Report

	Value
bmRequestType	0x21 (To Device, Type = Class, Recipient = Interface)
Request Code	0x09 (**Set Report**)
wValue	0x0304 (Feature Report, ReportID = 4)
wIndex	0x0000 = Kbd, 0x0001 = mouse

■ Obtaining the RadioParams Report

When the bridge receives, from the host, a control endpoint request with the parameters listed on Table 5-6, it returns an 8-byte RadioParams report over the control endpoint.

Control endpoint request for RadioParams report are listed.

Table 5-6. USB Get Report

	Value
bmRequestType	0xA1 (From Device, Type = Class, Recipient = Interface)
Request Code	0x01 (**Get Report**)
wValue	0x0304 (Feature Report, ReportID = 4)
wIndex	0x0000 = Kbd, 0x0001 = mouse

When the bridge receives the Get Report control request code, it returns a RadioParams report and then resets the Packets Transferred parameter for the specified device to zero. The Link Quality value is updated whenever the bridge receives a radio packet from the wireless device. Battery Level is only updated when the device sends an updated battery level report. At startup, the Battery Level, Corrupt Packets, and Packets Transferred are initialized to zero.

Example USB Bus Analyzer (CATC) Traces: Figure 5-13 shows the USB data transmissions between the bridge and the host PC captured with the USB CATC Bus Analyzer. In this example, the Right Shift + 'g', 'h' keys were typed followed by the 'Volume Up', 'Volume Down' keys. Note the keyboard regular key reports are sent to the PC via the endpoint 1 while the multimedia key reports are sent via the endpoint 2 with Report ID 2.

Figure 5-13. Example Keyboard CATC Trace (Standard and MM Keys)
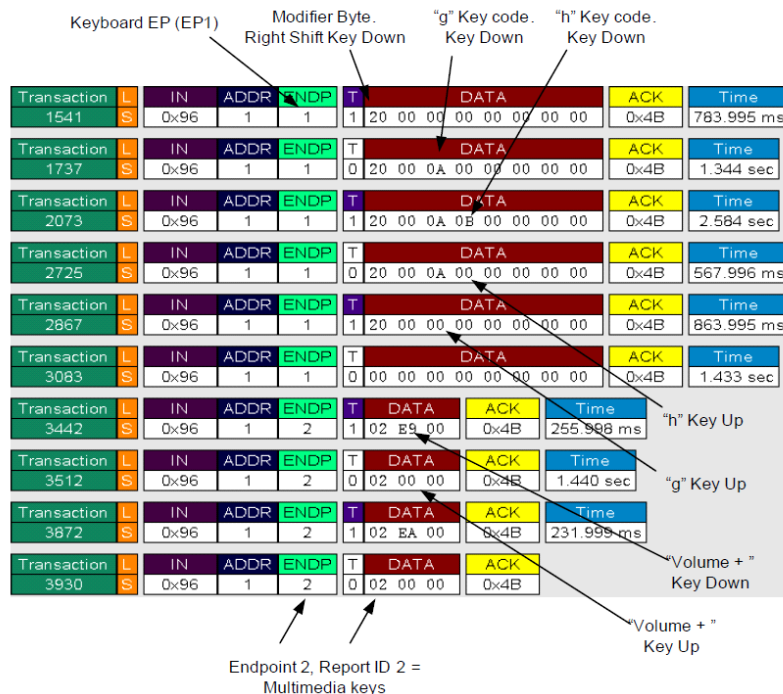
Figure 5-14 shows the mouse data being transferred between the dongle and the host PC. The first part of the trace shows the mouse data when the left button is pressed and held down as the mouse is moved, and then the left button is released. The second part of the trace shows the Z-wheel being moved down and up.

Figure 5-14. Example Mouse CATC Trace
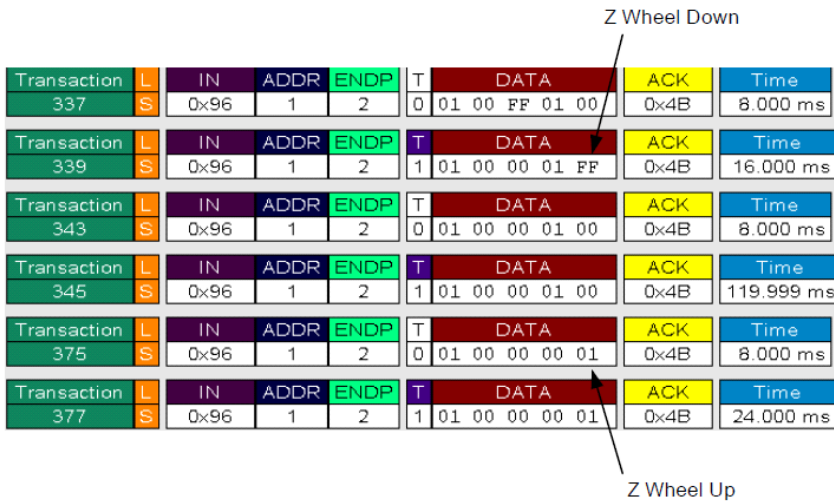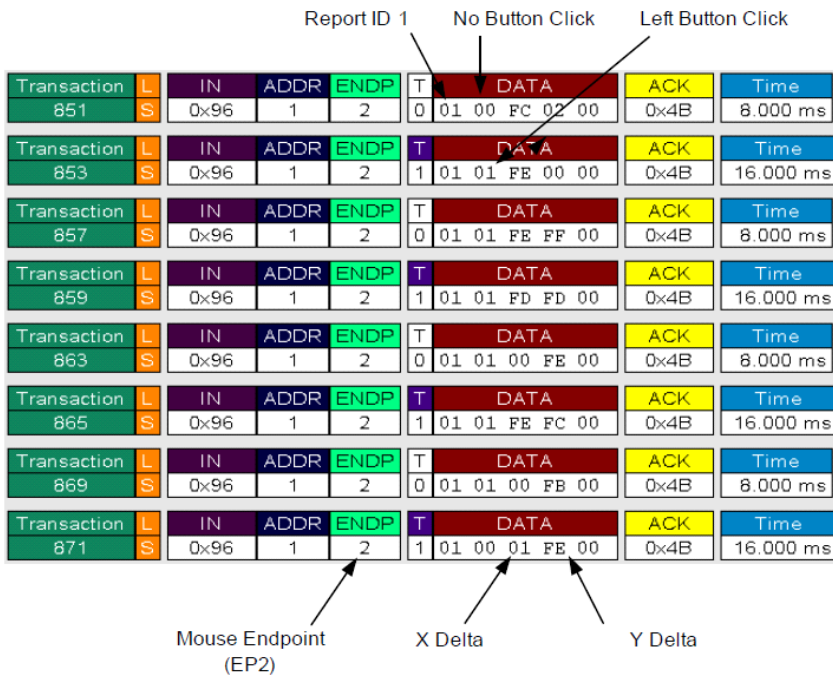
Figure 5-15 shows the Sleep key being pressed. Note the power key reports are sent via endpoint 2 and Report ID 3.

Figure 5-15. Example Keyboard CATC Trace (Power Key)
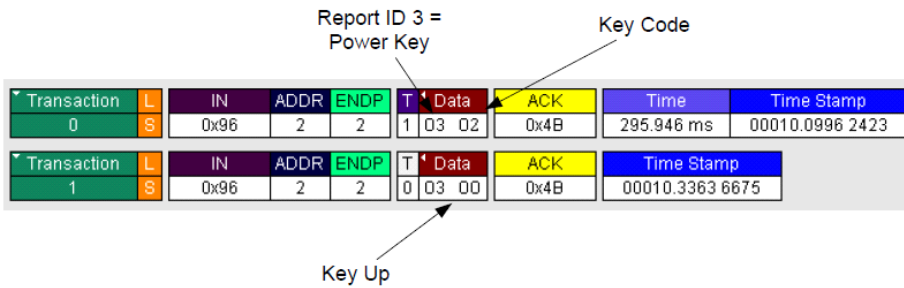
Figure 5-16 shows the Get_Report requests used to retrieve the keyboard and mouse battery level and link quality information. Note the data transfers occurred on the control endpoint, endpoint 0, and Report ID are used to differentiate keyboard and mouse requests.

Figure 5-16. CATC Trace of Battery and Link Quality Data Requests
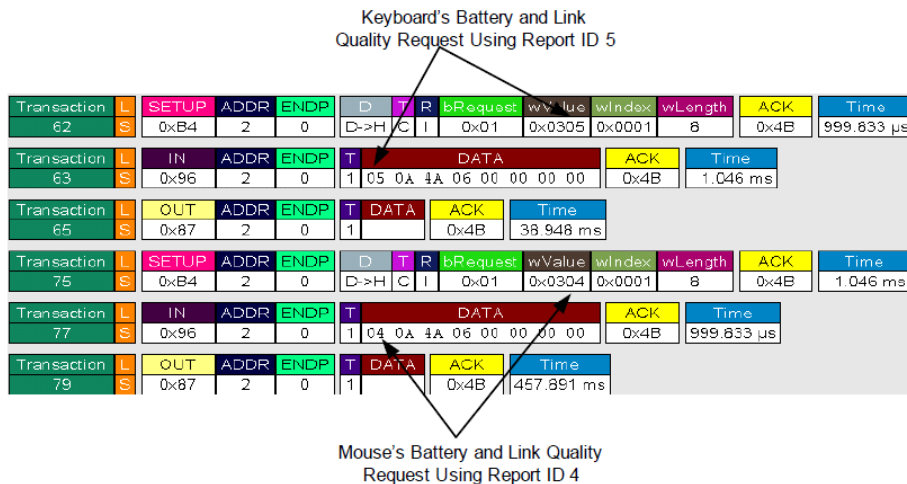
5.1.4   Verify Output

1. Connect the bridge to the PC and wait for the device to enumerate, the red LED turns ON, it turns OFF when the USB enumeration process completes. When enumeration is done, start using the bridge.

2. Press the bind button, the red LED blinks ON/OFF when the dongle is in Bind Mode.

3. Press the bind button on the mouse or keyboard, the devices are bound together.

4. Press any key on the keyboard or make mouse movements, the green LED turns on when the dongle receives data from mouse or keyboard.

5. The green LED turns on and remains on if a key is pressed and held.

5.2 Project 2 - PRoC_LP_RDK_Mouse

5.2.1 Project Description

This code example demonstrates various features and functions offered by the mouse. The mouse is a three button mouse that includes left button, right button, and a center button along with a scroll wheel.

5.2.1.1 ROM/RAM Usage

The following table shows the ROM/RAM usage. The top part exhibits the total ROM/RAM usage for basic functions, which disables all the build options. The bottom part exhibits the ROM/RAM usage for individual build options.

Table 5-7. ROM/RAM Usage

	Total ROM (Bytes)	Total RAM (Bytes)
Basic Function	5202	69

Build Option	ROM Usage (Bytes)	RAM Usage (Bytes)
MOUSE_BATTERY_STATUS	708	2
MOUSE_TEST_CODE	522	0
MFG_TEST_CODE	448	0
MFG_TX_MODES	735	3

5.2.2 Device Configurations

The PRoC LP Programmable Radio on Chip is configured using the Device Editor in PSoC Designer. The mouse uses two digital blocks to support two separate user modules. The first module is an SPI master to communicate with the optical sensor and the radio. The second module is a Programmable Interval Timer configured to operate as a 12-bit timer. Figure 5-17 shows the Device Editor with the user module mapping.

Figure 5-17. Device Configuration
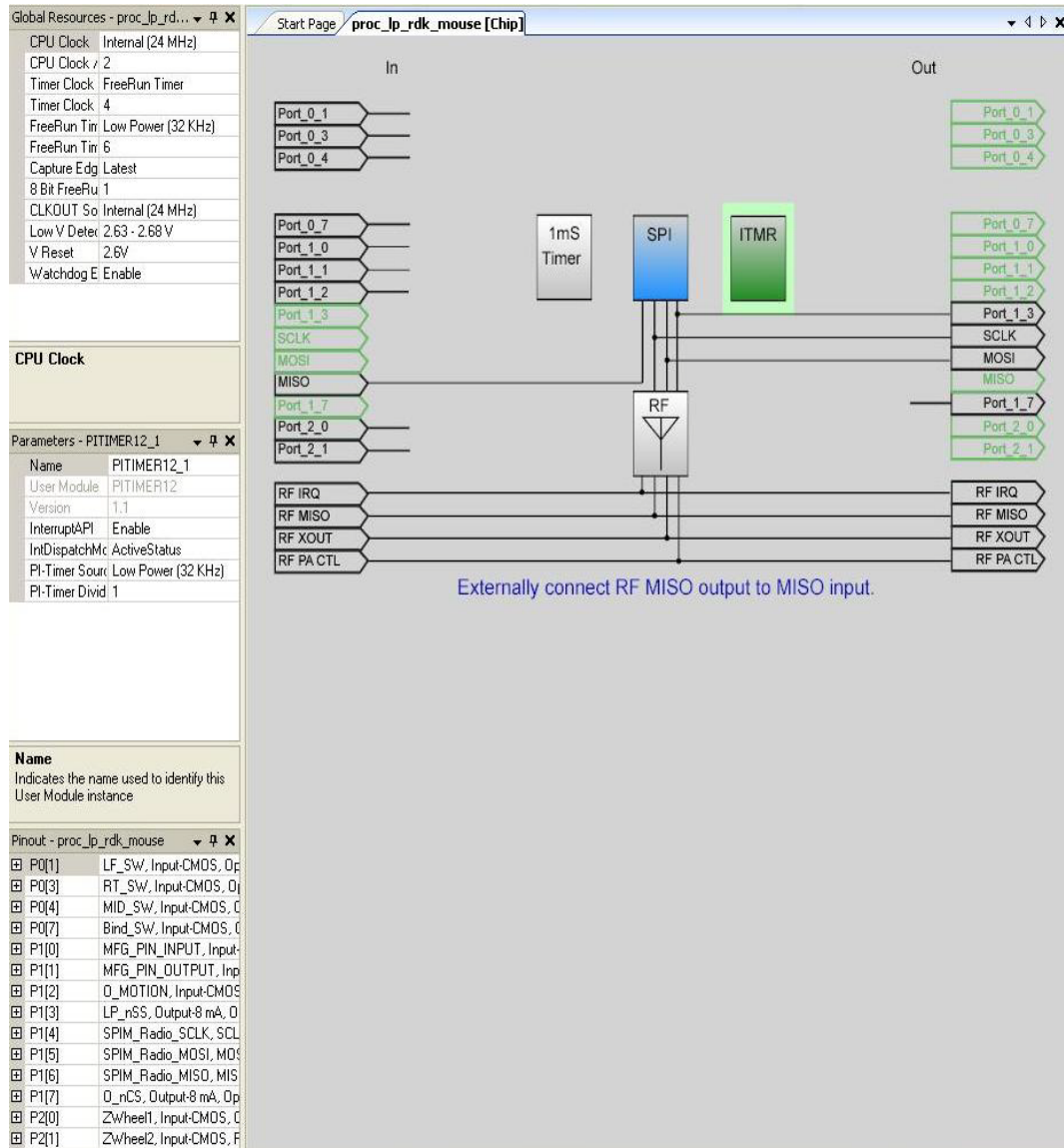
5.2.2.1 Global Configuration

Following is a description of the Global Resources that are configured for the CYRF69103 PRoC LP Programmable Radio on Chip. Take care when modifying these values because they affect the following user modules.

■ CPU Clock: This parameter is set to Internal (24 MHz). To run the CPU at 12 MHz, CPU Clock/N needs to be set to '2'. This operating frequency provides for faster code execution so that when events are detected the microcontroller can be put back into the sleep state quicker for improved power savings.

■ CPU Clock/N: This parameter is set to '2' to provide a 12 MHz clock.

■ Timer Clock: This parameter is set to 'FreeRun Timer'.

■ Timer Clock/N: This parameter is set to '4'.

■ FreeRun Timer: This parameter is set to 'Low Power (32 kHz)'.

- FreeRun Timer/N: This parameter is set to '6'.
- Capture Edge: This parameter is set to 'Latest'.
- 8-Bit Capture Prescaler: This parameter is set to '1'.
- CLKOUT Source: This parameter is set to 'Internal (24 MHz)'.
- Low V Detect: This parameter is set to 2.63 V to 2.68 V.
- V Reset: This parameter is set to '2.6 V'.
- Watchdog Enable: This parameter should be set to 'Enable', but may be set to 'Disable' for debug purposes.

5.2.2.2   *SPI Master User Module*

The SPI Master User Module is used to communicate with both the radio transceiver and the optical sensor. Both devices support leading edge data latching, noninverted clock and MSB first transmission as defaults. A clock divisor of 12 is chosen, which generates an SPI clock of 1 MHz. The interrupt API to this module is not used. See the SPI Module to learn how this module is used to implement communication with multiple devices on the SPI bus.

5.2.2.3   *PWM User Module*

The Programmable Interval Timer User Module is configured to use the Internal 32-kHz low-power oscillator. This module is used to provide a periodic interrupt to the timer code module to maintain a power saving millisecond sleep routine. The period of the timer is calibrated to the system clock at power on to provide a period of about 250 µs. This calibration is performed to account for variations in temperature and ILO variances from part to part. Configured the module to generate a terminal count interrupt. The period parameter is ignored because it is programmed at run time based upon the calibration results. See the Timer Module for more details on calibration.

5.2.2.4   *Flash Security*

The PSoC Designer mouse project has a file called FlashSecurity.txt. This file specifies access rules to blocks of the flash ROM. Refer to the documentation at the top of the file for definitions. This file is shipped with a single change from its default configuration. The block starting at hex address 1FC0 is changed from W: Full (Write protected) to U: Unprotected. This location of flash is dedicated to saving nonvolatile configuration values for the protocol code module (refer to the Protocol Module). **Note** When building the mouse firmware, check that the text image size does not occupy this block.

5.2.3   Model

Figure 5-18. Firmware Architecture Model
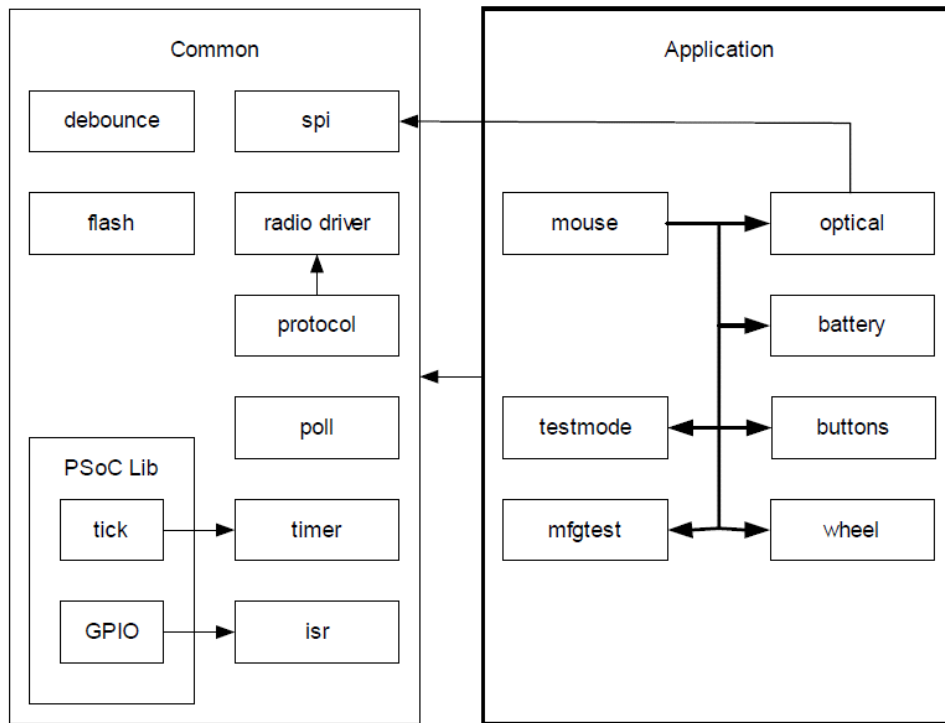
The mouse firmware is partitioned into two logical groups. The Common group is a collection of code modules that provide the underlying support for the application. This group provides services such as radio protocol, radio driver, timing, polling, flash access, contact debounce, SPI, and interrupts.

The Application group implements the core functionality and features of the PRoC LP RDK mouse. This includes power management, optical sensor, button, z-wheel, packet formatting and reporting, various test modes, and battery level sensing.

The following module descriptions have corresponding <module name>.c or <module name>.asm and <module name>.h source code files. The module API and definitions are exported in the header file while the module implementation and local definitions are contained in the C/ assembly file.

5.2.3.1   Common Code

The modules in the common code group are a combination of two sources. The first is PSoC Designer generated files in the lib directory that are modified to support the application. The second group is modules that are generally used by the application.

Generated Library Code

There is currently only one file, generated by PSoC Designer, that is modified for the use of the application. A minimal amount of code is added to this module in user protected areas that are preserved across code generation.

■   Timer Interrupt Module

The timer interrupt module is modified to provide a finer timing of 250 μs for the poll module and course timing by providing a 1 ms tick. When the timer module is turned OFF, it still provides a sense of time on the 1 ms tick by using the sleep timer. In this case, polling is disabled to conserve power. See the Poll Module and Timer Module for more details.

■ Debounce Module

The debounce module is an assembly coded routine to perform debounce on button presses and z-wheel motion. The algorithm is one that is published in EDN article as a way to perform hardware debounce in software.

The debounce is performed by polling the inputs at a fixed period and by adding a weighted value of the input to an accumulated value carried from the previous poll. The output is then passed to threshold logic, with built in hysteresis, and a logic value of one or zero is computed. The thresholds can be changed to adjust the hysteresis crossings by setting SCHMITT_HIGH_THRESH and SCHMITT_LOW_THRESH. When an input has changed state, the output can be observed to change approximately 10x the poll period later with the current threshold settings. With a poll period of 250 µs the input latency is about 2.5 ms. Refer to Contact-debouncing algorithm emulates Schmitt trigger at http://www.edn.com for more details on the operation of this algorithm.

■ SPI Module

This module provides an interface to the SPI bus for the optical sensor only. Physically the SPI bus is connected to the radio and the optical sensor. The radio driver is responsible for interfacing with the radio. The PRoC LP SPI Master module does not manage the selection of slave devices. This module is created to provide that functionality. This module has a dependency on the instantiation of a SPIM module in PSoC Designer that is properly connected to the devices.

In the PRoC RDK mouse design, the master SPI communicates with both the radio and optical sensor. Because the optical sensor does not supports 3-wire SPI mode, the 4-wire mode is employed. To save the GPIO pin, the IRQ pin function is multiplexed onto the MOSI pin.

■ Radio Driver

The radio driver module is a low level module providing basic radio communication and configuration. Its general application is such that it is likely not to be changed by the firmware developer. It provides an interface for reading/writing radio registers, setting PN codes and initialization of the radio and transmitting or receiving packets. See the PRoC LP Radio Driver documentation for details.

■ Protocol Module

The protocol module defines and implements the layer used to deliver packets from the device to the bridge. It manages the binding of devices to a bridge as well as the connection and interference immunity by channel hopping. This module has a dependency on the Radio Driver for sending formatted packets and the flash module for storing the manufacturing ID of the bridge the device is bound to.

■ Flash Module

The flash module is a smaller version of E2PROM module provided in PSoC Designer. It is limited in functionality and only implements the read/write routines required by the device. The flash-security.txt file must be modified so that the block being modified by this module is given read/write privilege, such as unprotected. Currently the very top most block in flash is used for this module.

■ Port Module

GPIO pins on the PRoC LP ports can be configured as outputs with a pull up resistor. This is the case for mouse buttons and the bind button. To activate the pull up, a data value of one must be written to the port data latch for the pin. This feature presents a problem when performing a read-modify- write on the port. For example, if a button is pressed (grounding the pin), a zero is read and written back out on the read-modify-write operation. This turns OFF the pull up for the button thereby, essentially disables the button. The port module provides an interface to treat ports, using the pull up feature, in a special way by caching the drive data for the port.

■ Poll Module

The poll module manages the timing, enabling/disabling and polling of the mouse buttons and zwheel inputs. When the mouse is active, polling is enabled and occurs at a rate of about 250 µs for the z-wheel (see the Timer Module) and a rate of about 3 ms for the mouse buttons. When the mouse is inactive, the buttons are changed to interrupt mode and the z-wheel is polled for change only when the sleep timer expires; see the Buttons Module and Wheel Module.

■ Timer Module

The PRoC LP has an internal low-power oscillator (ILO) that is used for generating a clock to a Programmable Interval Timer. This clock is affected by voltage and temperature and may drift over time. This module provides an interface to calibrate this clock to the system clock. The Programmable Interval Timer period is calibrated to be approximately 250 µs. Be especially careful when changing this period because the poll/debounce modules are coupled to this time value; see the Poll Module and Debounce Module. The timer module also provides a set of functions for performing busy waits in the microsecond resolution. For more coarse timing requirements, an API is provided for millisecond delays. The millisecond delay routines must be used as often as possible to provide for better power consumption because the microcontroller sleep feature is used. Also, when polling is enabled, it is performed as a background task during the millisecond delay. This module also adjusts the tick advancement based upon the sleep resolution. Turning OFF the timer provides for more power savings, yet a sense of time is still preserved for non-critical timing.

Note When using the ICE-Cube, define the macro PSOC_ICE so that busy waits are used instead of the sleep instruction. Using the sleep instruction with the ICE-Cube generates errors due to synchronization issues between the OCD part and the emulator.

■ ISR Module

This module provides an interface to initialize the interrupt.

5.2.3.2 Application Code

The group of modules that make up the application code are responsible for implementing the mouse functionality and behavior. Following is a high level description of each module responsibility and associated algorithms.

Mouse Module

The mouse module is the controlling code for the application. It has many responsibilities in implementing various features and functions offered by the mouse. The data formats and reporting algorithms along with power management are explained in this section. A few format types are defined to support the operation of the mouse. One of these is the packet format used when sending data to the bridge. This type is defined as TX_PACKET and is structured to support the different data packet formats as explained in Wireless Protocol Data Payload on page 48. The present definition combines z-wheel data with button data into one byte to conserve battery power by shortening the 'on time' of the radio. This format needs to change to support a mouse with more than three buttons and a z-wheel, perhaps sending four bytes instead of three.

The function main() is the entry point for the mouse application. This function is called from the boot.asm file. The mouse first initializes all of the application modules and then initializes the protocol module; see the Protocol Module. There is an order dependency for some of these, so care must be taken in modifying the mouse_init() function. For example, other modules depend upon the timer facility running to perform initialization. The SPI module must be initialized before the optical and protocol modules can be initialized; see the SPI Module, Optical Module and Protocol Module. When each module is initialized, then the application checks for entry to the 'LP' draw test mode or the manufacturing test mode. If neither of the test modes is indicated, then normal mouse operation begins.

The mouse module handles a variety of events at the main thread level. Most interrupt routines post notification that an event occurred by using the macros provided by the mouse interface. The mouse then processes these events at thread context rather than interrupt context. The mouse application is implemented using a state machine to manage the various power modes that it executes at any given time. The mouse initially enters a disconnected state. When there is any mouse activity, it enters the active state.

In the active state, the timer is turned on so that more accurate timing and mouse events can be collected, formatted and reported to the bridge. The mouse remains in this state as long as there is mouse activity to report to the bridge or a period of time without any mouse activity has expired, after which it returns to the idle state. If the mouse is unable to deliver a packet while in this state, it transitions to the disconnected state.

In idle state, the optical sensor is allowed to transition through its various rest modes to conserve power. In this state, the mouse application is waiting for input from the optical sensor, z-wheel or buttons. The timer is turned OFF to conserve power and the notion of time is maintained using the sleep timer. This state is maintained indefinitely until the batteries drop below 1.8 V at which point the mouse enters the OFF state. The OFF state is where the radio and optical sensor are prevented from turning on. This state is reached when the battery voltage drops below 1.8 V. It is designed to keep the battery drain to an absolute minimum to prevent battery leakage as a result of completely draining the batteries.

The battery level is reported by the mouse application when it detects a change from the disconnected state to the connected state. The battery level is measured when exiting the idle state. If there is a change in the battery level, it is reported in the active state. In the active state the mouse attempts to deliver a packet for the amount of time designated in MOUSE_TX_TIMEOUT_MS. If it is unable to send the packet in this time, then it transitions to the disconnected state.

The mouse application is responsible for detecting the bind button press and then calling the bind function in the protocol module; see the Protocol Module. The mouse application sends mouse reports as frequently as events arrive, but not any faster than the time defined in the macro MOUSE_REPORT_IN_MS. Carefully set this time so that the report rate does not exceed that which the USB bus is capable of handling. Keep in mind that the report rate varies slightly due to drift of the internal oscillator used to keep track of time.

Optical Module

The optical sensor module encapsulates the initialization, calibration and reading of the optical sensor. This module also handles any power management required by the sensor, along with motion detection if supported. The contents of this module potentially change with every design and are unique to the sensor used. This module has the responsibility to format the X and Y data into the mouse packet payload. See Wireless Protocol Data Payload on page 48 for a definition of the packet payload.

Testmode Module

The Testmode module provides code to continuously perform a vector drawing test within a drawing application. This test mode is used to check radio range, co-location and interoperability of the mouse with the keyboard. The test mode, when compiled in, is entered by holding down the left and right button while inserting the batteries. The buttons must be held down until the optical sensor begins to flash. As soon as the buttons are released the mouse repeatedly draws 'LP' in the drawing application. Each successive 'LP' must be drawn on top of the previous one. The test mode may only be exited by removing the batteries. All button presses and mouse movement are ignored when in the test mode. However, care must be taken not to bump other mice connected to the PC.

Note The mouse 'acceleration' or 'enhance pointer precision' option needs to be turned OFF in the Windows mouse Control Panel for this test to execute properly. If the letters are drawn erratically

with uneven sides or excessive amounts of space in between them, then check this setting or its equivalent (based upon your PC operating system).

When the macro DEBUG_INDEX is defined, code is generated to move the mouse pointer to the right and back again without the pen down. This is done in an incrementing fashion so that when observing packet data on a Listener, a correlation can be made with a USB protocol analyzer. This is useful for debugging data loss because the test mode guarantees packet delivery. Entry to this test mode can be changed by modifying the macro TESTMODE_BUTTONS in the testmode.c file. The button macros are defined in the buttons.h file.

Buttons Module

The buttons module provides an API for handling both the bind and mouse buttons. This module must be changed when adding or removing buttons for a new mouse design. The button portion of the packet payload is formatted by this module and needs to change if more buttons are added. See the Mouse Module for a definition of the packet payload format.

This module manages power configurations that may be implemented to conserve power related to button presses. For example, button polling is turned OFF and interrupts are used to detect button presses in the idle state. It also manages the acquisition of button information depending on the implementation: interrupts or polling.

When changes in a button state are detected, the mouse module is notified for collection and reporting of the data. **Note** It is important for the buttons module to always report the button state when a button is pressed. This condition frequently occurs when the mouse is moved with the button held down.

Mfgtest Module

The manufacturing test module may be optionally compiled in, at the expense of code space, by defining the macro MFG_TEST_CODE. In addition, a more complete version may be compiled in by defining MFG_TX_MODES. The TX modes include code to perform a carrier test and a random data test.

The manufacturing test code is designed to be compatible with the CY3631 Manufacturing Test Kit Tester. Entry into this mode on the mouse is performed by placing a shorting block over pins four and five of the ISSP programming header and then inserting the batteries. The test mode may only be exited by removing the batteries and shorting block. For more information on how to use this test mode, refer to the CY3631 Manufacturing Test Kit documentation.

It is recommended that you not make changes to this module unless similar changes are made to the CY3631 Tester.

Wheel Module

The wheel module implements the functionality of the z-wheel. It is responsible for power modes associated with the z-wheel, polling, z-wheel interrupts, wheel position tracking, and partial packet formatting for z-wheel reports.

When the z-wheel is being polled, the GPIO pins are turned on with internal pull up resistors just long enough to read the state. This is done to conserve power when the mouse is active. When the polling timer is turned OFF the wheel_poll_sleep() function is called which only looks for change from the last state; it does not keep track of wheel position.

Z-wheel position tracking is done by comparing debounced wheel input to the previous two states. Depending upon the wheel input phase transition the direction of the wheel can be determined. The poll rate must be frequent enough to debounce and catch these transitions for a smooth response. The RDK mouse is shipped with a mechanical encoder. It is typical for this decoder to rest on a detent such that the z-wheel inputs are either both high or both low, hence the reason for only turning

on the pull ups when polling the input. Transition from one of these states to the other is reported as a ±1 motion. **Note** Sometimes the mechanical detents do not align with the high-high or low-low state and movement may not be seen every time from detent to detent.

When z-wheel motion is detected, the mouse module is notified for collection and reporting of the data; see the Mouse Module.

Battery Module

The battery monitor circuit is implemented using the Low Voltage Interrupt (LVI) on the LP radio. Following is an explanation of the process to measure the battery voltage. The process first sets the LVI threshold to 1.8 V and then checks for an LVI interrupt. If the interrupt does not occur then it repeatedly sets the LVI TH and PMU OUTV with the following combination and checks the status.

Table 5-8. LVI TH and PMU OUTV Combinations

LVI TH	PMU OUTV	VOLTAGE IF INTERRUPT OCCURS
1.8V	2.7V	< 1.8V
2.0V	2.7V	< 2.0V
2.2V	2.7V	< 2.2V
PMU OUTV	2.7V	< 2.7V

It then returns a battery level between 1 and 10: 1 being below 1.8 V and 10 being above 2.7 V.

5.2.3.3   Configuration Options

All configuration options for the application can be found in the config.h file, and some of them are defined in the Project > Setting > Compiler > Macro defines. Each option is explained below and can be changed to values that meet the developer's needs.

- MOUSE_REPORT_IN_MS

  This configuration value sets the shortest period at which the firmware honors events from the mouse hardware to transmit using the radio. The default value is approximately 10 milliseconds. Setting this value to something smaller than the USB poll period of 8 milliseconds generates excessive radio retries from the mouse and is not recommended. Larger values improve battery life, but may affect usability of the mouse. See the Timer Module for a description of timing accuracy. This valued is defined in milliseconds.

- MOUSE_ACTIVE_MS

  This value sets how long the timer module runs generating poll interrupts for the z-wheel and buttons. This time affects power consumption of the mouse. When this time expires, the buttons and zwheel go into a power down state, improving battery life. In power down state, z-wheel movement exhibits latency. See the Buttons Module and Wheel Module for descriptions of power down states and operation. This value is defined in milliseconds.

- MOUSE_DISCONNECTED_POLL_MS

  Sets the rate at which the battery voltage is monitored while in the disconnected state. This ensures that if the batteries go below the minimum battery voltage of 1.8 V, the radio and optical sensor are prevented from turning on.

- MOUSE_TX_TIMEOUT_MS

  The transmit loop in the mouse attempts to guarantee delivery of mouse events. This loop eventually times out if it does not receive a response from the bridge. This value sets that time-out time. The default value is 2000. This value is defined in milliseconds.

- MOUSE_CONNECT_ATTEMPT_TIMES

This value sets the attempt times for the mouse trying to connect to the bridge before entering the Briefcase Mode. The default value is 20.

- PLATFORM_H

This configuration value identifies the header file that has the platform configuration information. The default value is pdc9347.h, which is the identifier for the mouse board that is shipped with the RDK. This macro changes when the code is ported to another platform.

- MOUSE_800_NOT_400_CPI

This configuration definition is used to select between 800 or 400 counts per inch (cpi) when configuring the optical chip. If it is defined then 800 cpi is selected. If it not defined then 400 cpi is selected. The default is 800 cpi.

- MOUSE_BATTERY_STATUS

Enabling this feature causes the battery level measurement code to be compiled into the mouse image. The mouse then measures the battery level and reports any changes to the bridge. Notification of the battery level is done at the following events: the battery level changes, the mouse transitions from the idle state to the active state, mouse transitions from the disconnected state to the connected state.

- MOUSE_TEST_MODE

This configuration definition is used to selectively compile code for mouse test mode. If this value is defined, then the test mode is compiled into the executable image. The test mode moves the mouse in a fashion to repeatedly draw the letters 'LP' in a drawing program. When performing this test, turn OFF Mouse acceleration or advanced motion. See the Testmode Module for more information on entering this test mode.

- MFG_TEST_CODE

This configuration definition is used to selectively compile the manufacturing test code. The manufacturing test code in this mouse is compatible with the CY3631 Manufacturing Test Kit offered by Cypress Semiconductor. See the Mfgtest Module for a description of how this test mode is executed. The CY3631 Manufacturing Test Kit documentation has a description of the test operation.

- MFG_TX_MODES

When the MFG_TEST_CODE is defined, then the definition of this name adds in a carrier and random data TX test option. See the Mfgtest Module for more information on these TX modes.

- MASTER_PROTOCOL

This configuration definition is used to select the Master radio protocol or Slave radio protocol. For the mouse application, it should be undefined.

- PAYLOAD_LENGTH

This configuration definition is used to define the payload length. For the mouse application, it should be defined as 3.

- KISS_BIND

This configuration definition is used to selectively compile the Enhanced KissBind feature. See Enhanced KISSBind on page 98 for more information. The mouse can be unbound by holding the right and middle buttons, and pressing the bind button. After being unbound, the mouse enters an infinite loop until a POR. After being unbound, the mouse can be bound to a bridge by KissBind.

- RSSI_QUALIFY

This configuration definition is used to enable the RSSI qualification for the Enhanced KissBind. Only if the RSSI reading is above KISS_BIND_RSSI_THRESHOLD, can the KissBind request/response be accepted by the Bridge/Devices.

- AUTO_CONNECT

When the bridge is absent, after MOUSE_CONNECT_ATTEMPT_TIMES times of attempts to connect to the bridge, the mouse enters the Briefcase Mode. In this mode, the mouse shuts down the sensor to save power. When the mouse enters the Briefcase Mode, if the AUTO_CONNECT is defined, the mouse tries to connect to the bridge automatically every MOUSE_DISCONNECTED_POLL_MS seconds. If the AUTO_CONNECT is not defined, the mouse tries to connect to the bridge only when the buttons are pressed.

5.2.3.4   Platform and Architecture Portability

The mouse firmware is designed to be easily ported from one hardware platform to another platform with a simple re-mapping of pins on the PRoC LP. The file pdc9347.h maintains the pin mapping definitions that are used throughout the code and is included in about every file by using the macro PLATFORM_H that is defined in config.h. Porting the code to another microprocessor architecture requires modification or leverage of the existing code for processor specific features, along with pin definitions.

5.2.3.5   Initialization

Initialization of the PRoC LP chip is done by code that is generated in boot.asm by PSoC Designer. The module boot.asm calls main() in the mouse module when the Wireless PRoC LP is configured and initialized; see the Mouse Module.

5.2.3.6   Wireless Protocol Data Payload

The mouse protocol is optimized to reduce the 'on-time' of the radio, which equates to reduced power consumption. This optimization relies upon the PRoC LP RDK requirement of a three-button mouse. With this requirement, it is possible to combine the z-wheel and the button report into a single byte, allowing five bits of information for the z-wheel and three bits for the buttons.

The protocol code module offers the ability to send variable length packets, thereby allowing a reduced number of bytes to be transmitted over the air, to extend battery life.

Because mouse usage data demonstrates that X, Y optical sensor data is more frequent than z-wheel or button presses, the following transmission packet formats are implemented in this RDK. The packet formats only show the application payload and do not show the protocol packet format.

- Packet Format 1: When there is only X, Y delta data, the transmitted packet is two bytes.

Table 5-9. Packet Format 1

Byte 1	Byte 2
X Delta (8 bits)	Y Delta (8 bits)

- Packet Format 2: When there is either z-wheel data or button data, the transmitted packet is three bytes. In the case where there is no X, Y delta data, but there is z-wheel or button data, the X, Y delta bytes is set to zero. The z-wheel data is a signed value with bit 4 as the sign bit.

Table 5-10. Packet Format 2

Byte 1	Byte 2	Byte 3
X Delta (8 bits)	Y Delta (8 bits)	Buttons (Bits[7:5]), Z Delta (Bits[4:0])

- Packet Format 3: When battery voltage level is communicated, the transmitted packet is 1 byte.

Table 5-11. Packet Format 3

Byte 1
Battery Level (1 − 10)

5.2.3.7   Interrupt Usage and Timing

In the RDK mouse, the following interrupts are enabled:

- Motion interrupt from the optical sensor
- Button (Left, Middle and Right buttons) interrupt
- Bind button interrupt

The interrupt latency includes two portions. The first portion is the time between the assertion of an enabled interrupt and the start of its ISR, which can be calculated using the following equation:

Latency1 = Time for current instruction to finish + Time for M8C to change program counter to interrupt address + Time for LJMP instruction in interrupt table to execute.

For example, if the 5-cycle JMP instruction is executing when an interrupt becomes active, the total number of CPU clock cycles before the ISR begins are as follows:

(1 to 5 cycles for JMP to finish) + (13 cycles for interrupt routine) + (7 cycles for LJMP) = 21 to 25 cycles.

In the example above, at 12 MHz, 25 clock cycles take 2.083 µs.

The second portion is the time between the start of the ISR and the post of the event flag. For example, the motion interrupt takes 23 CPU clock cycles for this portion. Therefore, the Latency2 equals to 1.917 µs for the 12 MHz CPU.

Consequently, the total latency for a motion interrupt is:

Latency1 + Latency2 = 4 µs

5.2.3.8   Code Performance Analysis

A mouse motion report is used to analyze the code performance. A typical mouse motion report contains the following steps:

- Optical sensor responds to a mouse motion. With the mouse the sensor in the rest 1 state, it takes 16.5 ms for the sensor to responds to this sensor motion.
- The sensor interrupts the MCU by lowering its motion pin. In the previous section, it is calculated that it takes 4 µs for MCU to respond to this Interrupt.
- In the function timer_wait_event (), the MCU exits the sleep state and spends 53 µs to finish the wheel poll.
- Firmware delays MOUSE_REPORT_IN_MS, which is 10 ms for the default. This delay is to prevent excessive radio retries from the mouse.
- Firmware calls function mouse_do_report() to read the Delta_X and Delta_Y value and send the packet to the bridge. This step takes 1.98 ms, which includes 1.66 ms radio transmission time.

As a result, if a mouse is in the rest 1 state, it takes 28.6 ms for the mouse to report a motion to a bridge.

5.2.4   Verify Output

1. Connect the bridge dongle to the PC, power the mouse with the two AA batteries.

2. Press the bind button on the bridge, the red LED blinks ON/OFF in the bind mode. Now press the bind button on the mouse the red LED stops blinking on binding the mouse with the bridge.

3. Press any switch on the mouse the green LED turns on when the dongle receives data from the mouse.

5.3   Project 3 - PRoC_LP_RDK_Keyboard

5.3.1   Project Description

This code example demonstrates various features and functions offered by the keyboard. This is a multimedia keyboard with 101 key inputs. It has 18 columns and 6 rows inputs.

There are two architectural views of the keyboard. The first is a microcontroller configuration view of user modules. This architecture and configuration is best viewed in PSoC Designer unwhen the project is loaded. The second view is a logical organization of the source code modules that make up the keyboard application code and other support modules.

5.3.1.1   ROM/RAM Usage

The following table shows the ROM/RAM usage. The top part exhibits the total ROM/RAM usage for basic functions, which disables all the build options. The bottom part exhibits the ROM/RAM usage for individual build options.

Table 5-12. ROM/RAM Usage

	Total ROM (Bytes)	Total RAM (Bytes)
Basic Functions	5861	127
Build Option	**ROM Usage (Bytes)**	**RAM Usage (Bytes)**
KEYBOARD_MULTIMEDIA_SUPPORT	756	1
KEYBOARD_TEST_MODES	339	1
KEYBOARD_BATTERY_VOLTAGE_SUPPORT	179	1
ENCRYPT_DATA TEA*	871	13
ENCRYPT_DATA AES	563	37
MOUSE_EMULATION_MODE	610	1
MFG_TEST_CODE	532	0
MFG_TX_MODES	707	1
BACK_CHANNEL_SUPPORT	185	5

Note *ENCRYPT_TEA option needs 64 bytes extra flash space to store the nonvolatile session key.

5.3.1.2   Keyboard Matrix

The RDK keyboard matrix has 18 columns and 8 rows. Key presses generate a GPIO interrupt when a column is connected (shorted) to a row. The keyboard then scans the matrix to determine which keys have been pressed.

The RDK keyboard matrix with the USB scan codes are shown in Table 5-13.

Table 5-13. RDK Keyboard Matrix

	Row 0	Row 1	Row 2	Row 3	Row 4	Row 5	Row 6	Row 7
Column 0	0x09	0x0A	0x19	0x05	0x17	0x15	0x21	0x22
Column 1	0x0D	0x0B	0x10	0x11	0x1C	0x18	0x24	0x23
Column 2	0x0E	0x3F	0x36	NA	0x30	0x0C	0x25	0x2E
Column 3	0x0F	NA	0x37	NA	0x40	0x12	0x26	0x41
Column 4	0x33	0x34	NA	0x38	0x2F	0x13	0x27	0x2D
Column 5	0x31	0x3E	0x28	0x2C	0x2A	NA	0x43	0x42
Column 6	0x5A	0x62	0x54	0x4F	0x5D	0x60	0x45	0x49
Column 7	0x59	NA	0x53	0x51	0x5C	0x5F	0x44	0x4C
Column 8	0x5B	0x63	0x55	0x56	0x5E	0x61	0x4E	0x4B
Column 9	0x07	0x3D	0x06	NA	0x3C	0x08	0x20	0x3B
Column 10	0x16	NA	0x1B	NA	0x39	0x1A	0x1F	0x3A
Column 11	0x04	0x29	0x1D	NA	0x2B	0x14	0x1E	0x35
Column 12	0x58	0x52	0x48	0x50	NA	0x57	0x4D	0x4A
Column 13	NA	0x04	NA	0x40	0x0192	0x47	0x46	0x0223
Column 14	0x02	0x00CD	0x20	NA	0x02	NA	0x0221	0x018A
Column 15	NA	NA	0x10	NA	0x00E9	NA	NA	0x01
Column 16	0x7D	0x00E2	0x80	0x7C	0x00B7	0x00EA	0x022A	NA
Column 17	0x08	0x0225	NA	0x7B	0x0224	0x65	0x00B6	0x00B5

Notes

■ Yellow indicates Multimedia Key (16-bit value)

■ Red indicates Power Key

■ Blue indicates Modifier Key

■ No color indicates a Standard 101 Key

5.3.2   Device Configurations

The enCoRe II LV is configured using the Device Editor in PSoC Designer. The Device Editor allows the Global Resources for the part and user module parameters to be configured. The keyboard uses two separate user modules. The first module is an SPI master for communicating with the radio. The second module is a Programmable Interval Timer configured to operate as a 12-bit timer. Figure 5-19 shows the Device Editor with the user module mapping.

Figure 5-19. Device Configuration
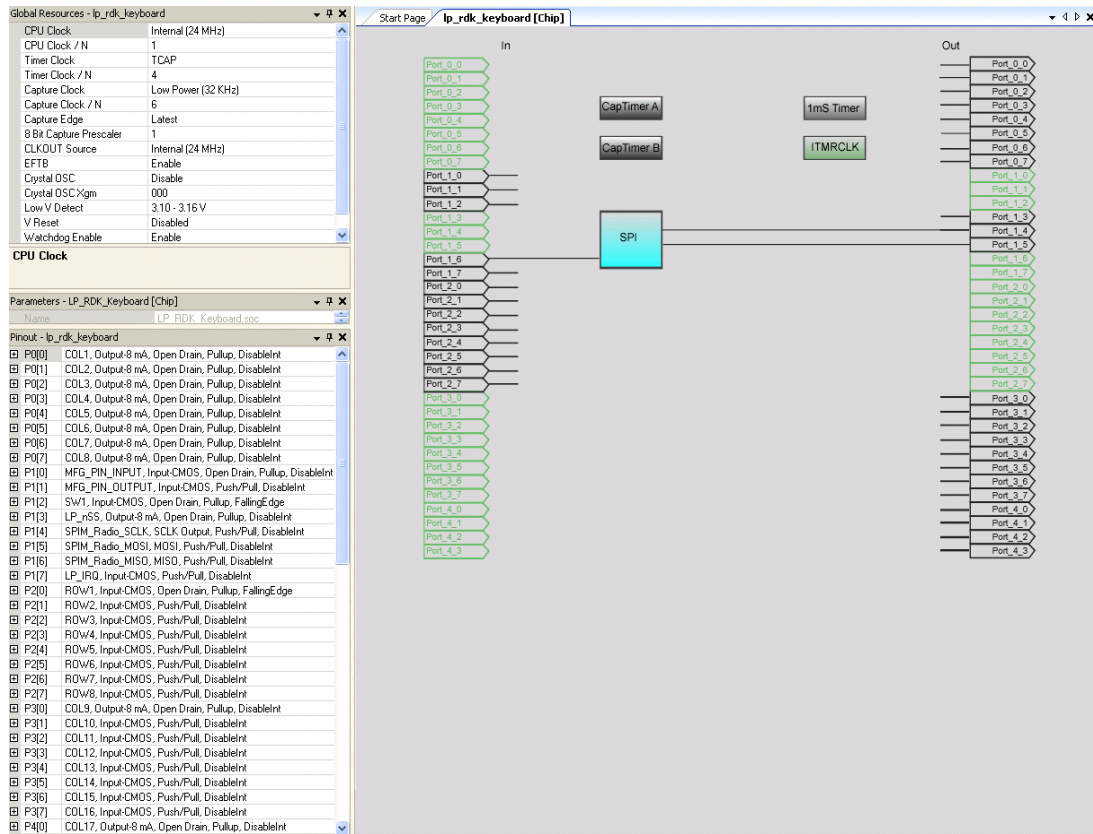
5.3.2.1   Global Configuration

The following is a description of the Global Resources that are configured for the CY7C60123-PVXC enCoRe II LV microcontroller. Take care when modifying these values because they affect the following user modules.

- CPU Clock: This parameter is set to Internal (24 MHz). To run the CPU at 12 MHz, CPU Clock/N must to be set to '2'. This operating frequency provides for faster code execution so that when events are detected the microcontroller can be put back into the sleep state quicker for improved power savings.

- CPU Clock / N: This parameter is set to '1' to provide a 12 MHz clock.

- Timer Clock: This parameter is set to 'TCAP'.

- Timer Clock /N: This parameter is set to '4'.

- Capture Clock: This parameter is set to 'Low Power (32 kHz)'.

- Capture Clock /N: This parameter is set to '6'.

- Capture Edge: This parameter is set to 'Latest'.

- 8 Bit Capture Prescaler: This parameter is set to '1'.

- CLKOUT Source: This parameter is set to 'Internal (24 MHz)'.

- EFTB: This parameter is set to 'Enable'.

- Crystal OSC: This parameter is set to 'Disable'.

- Crystal OSC Xgm: This parameter is set to '000'.

- Low V Detect: This parameter is set to 2.63 - 2.68 V.

- V Reset: This parameter is set to '2.6 V'.
- Watchdog Enable: This parameter should be set to 'Enable', but may be set to Disable for debug purposes.

5.3.2.2 SPI Master User Module

The SPI Master User Module is used to communicate with the radio transceiver. The radio transceiver supports leading edge data latching, non-inverted clock, and MSB first transmission as defaults. A clock divisor of 6 is chosen which generates an SPI clock of 2 MHz. The interrupt API to this module is not used. In the PRoC RDK keyboard design, the 4-wire mode is employed.

5.3.2.3 Programmable Interval Timer User Module

The Programmable Interval Timer User Module is configured to use the Internal 32-KHz Low-power Oscillator. This module is used to provide a periodic interrupt to the timer code module to maintain a power saving millisecond sleep routine. The period of the timer is calibrated to the system clock at power on to provide a period of about 1 ms. This calibration is performed to account for variations in temperature and ILO variances from part to part. Configure the module to generate a terminal count interrupt. The period parameter is ignored because it is programmed at run time based upon the calibration results. See the Timer Code Module for more details on calibration.

5.3.2.4 Flash Security

The keyboard project within PSoC Designer has a file called FlashSecurity.txt. This file specifies access rules to blocks of flash ROM. See the documentation at the top of the file for definitions. This file is shipped with a single change from its default configuration. The blocks starting at address 1FC0 hex are changed from W: Full (Write protected) to U: Unprotected. These locations of flash are dedicated to save nonvolatile configuration values for the protocol code module and nonvolatile session key for the encrypt code module. Note when building the mouse firmware, be sure to check that the text image size does not occupy this block.

5.3.3 Model

Figure 5-20. Firmware Architecture Model
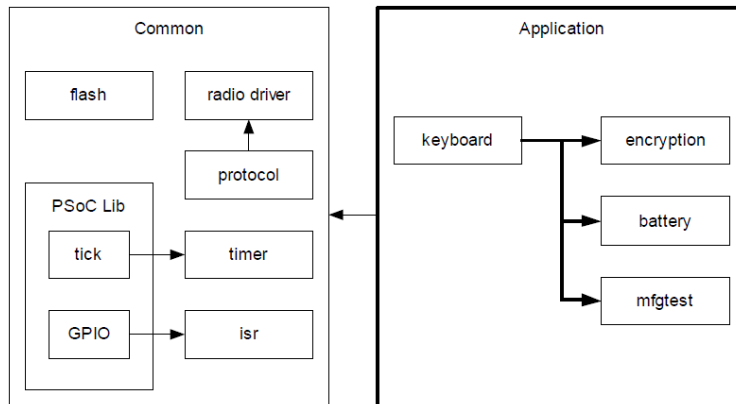
The keyboard firmware is partitioned into two logical groups. The Common group is a collection of code modules that provide the underlying support for the application. This group provides services such as, radio protocol, radio driver, timing, flash access, and interrupts.

The Application group implements the core functionality and features of RDK wireless keyboard. This includes power management, encryption, packet formatting and reporting, various test modes,

and battery level sensing. The code modules for each of these groups are described below in further detail.

All the following module descriptions have corresponding <module name>.c or <module name>.asm and <module name>.h source code files. The module API and definitions are exported in the header file while the module implementation and local definitions are contained in the C/assembly file.

5.3.3.1   Common Code

The modules in the common code group are a combination of two sources. The first is PSoC Designer generated files in the 'lib' directory that are modified to support the application. The second group is modules that are generally used by the application.

Generated Library Code

There are currently no files, generated by PSoC Designer, that are modified for the use of the application.

Radio Driver

The radio driver module is a low level module providing basic radio communication and configuration. Its general application is such that it is likely not to be changed by the firmware developer. It provides an interface for reading/writing radio registers, setting PN codes and initialization of the radio and transmitting or receiving packets. See the Radio Driver documentation for details.

Protocol Module

The protocol module defines and implements the layer used to deliver packets from the device to the bridge. It manages the binding of devices to a bridge as well as the connection and interference immunity by channel hopping. This module has a dependency on the radio driver for sending and receiving formatted packets and the flash module for storing the manufacturing ID of the bridge the device is bound to.

Flash Module

The flash module is a smaller version of the E2PROM module provided in PSoC Designer. It is limited in functionality and only implements the read/write routines required by the device. The flashsecurity.txt file must be modified so that the block being modified by this module is given read/write privilege, such as unprotected. Currently the one very top most block in flash is used by this module for storing the encryption key if encryption is enabled and the bind parameters.

ISR Module

This module provides an interface to initialize the interrupt.

Timer Module

The timer module provides a one-millisecond tick for the system. The tick resolution can be changed, but is set for one millisecond for the keyboard. This module requires the use of a 12-bit Programmable Interval Timer user module of the enCoRe II LV. The delay function used for millisecond timing provides at least the delay requested with no more than one additional millisecond of delay. The millisecond delay function puts the PSoC to sleep for the duration of the requested delay. The microprocessor wakes just long enough to update the tick every millisecond and check if the delay is met and then returns to sleep state if it has not. See the documentation in the module for requirements on configuring the enCoRe II LV block.

5.3.3.2   Application Code

The group of modules that make up the application code is responsible for implementing the keyboard functionality and behavior. Following is a high level description of each module responsibility and associated algorithms.

Keyboard Module

The keyboard module is the controlling code for the application. It has many responsibilities in implementing various features and functions offered by the keyboard.

The function main() is the entry point for the keyboard application. This function is called from the boot.asm file. The keyboard first initializes all of the application modules and then initializes the protocol module. There is an order dependency for some of these, so care must be taken in modifying the keyboard_init() function. For example, other modules depend upon the timer facility running to perform initialization. When each module is initialized, then the application checks for entry to the manufacturing test mode. If the manufacturing test mode is not indicated, then normal keyboard operation begins.

There are two states for the keyboard operation: the idle state and the active state. The keyboard initially enters idle state; when there is any keystroke, it enters the active state. In active state the keyboard is scanned for both the keys and the bind button. The keystrokes are collected, formatted and reported to the bridge. After that, the keyboard goes into the idle state.

In idle state, the MCU and radio go to sleep to save power, and the keyboard application remains waiting for input from the keys or bind button. The timer is turned OFF to conserve power. This state is maintained indefinitely until a keystroke or a button press occurs.

The battery level is reported by the keyboard application when it detects a keystroke after it has been in an idle state for 8 seconds.

In the active state, the keyboard attempts to deliver a packet for the amount of time designated in KEYBOARD_TX_TIMEOUT. The keyboard application is also responsible for detecting the bind button press and then calling the bind function in the protocol module.

The keyboard application sends keyboard reports as frequently as events arrive, but not any faster than the time defined in the macro KEY_DOWN_DELAY_SAMPLE_PERIOD. Carefully set this time so that the report rate does not exceed that which the USB bus is capable of handling. Keep in mind that the report rate varies slightly due to drift of the internal oscillator used to keep track of time.

Mfgtest Module

The RDK provides a compile-time option of adding a manufacturing test mode to the keyboard. The manufacturing test code in this keyboard is compatible with the CY3631 Manufacturing Test Kit offered by Cypress Semiconductor.

If MFG_TEST_CODE is defined and ENTER_BY_PIN is not defined, holding down the system sleep key and the bind button while inserting the batteries into the keyboard enters the manufacturing test mode.

If MFG_TEST_CODE and ENTER_BY_PIN are both defined, connecting pin 4 and 5 on the ISP header with a shunt and then inserting the batteries into the keyboard enters the manufacturing test mode.

The only way to exit this mode is to cycle power.

Battery Module

The battery monitor circuit is implemented using the Low Voltage Interrupt (LVI) on the LP radio. Following is an explanation of the process to measure the battery voltage.

The process first sets the LVI threshold to 1.8 V and then checks for an LVI interrupt. If the interrupt does not occur then it repeatedly sets the LVI TH and PMU OUTV with the following combination and checks the status.

Table 5-14. LVI TH and PMU OUTV Combinations

LVI TH	PMU OUTV	VOLTAGE IF INTERRUPT OCCURS
1.8V	2.7V	< 1.8V
2.0V	2.7V	< 2.0V
2.2V	2.7V	< 2.2V
PMU OUTV	2.7V	< 2.7V

It then returns a battery level between 1 and 10: 1 being below 1.8 V and 10 being above 2.7 volts.

Test Module

This RDK keyboard provides a compile-time option of adding test modes to the keyboard. The test mode module is implemented in a way that it can be easily extended to add other test modes. Currently there are only two test modes supported in the module. When this option is not enabled then all test mode code is removed from the compilation.

The first test mode is initiated by holding down the left Ctrl, left Alt, right Alt, right Ctrl, and F1 keys at the same time. If PANGRAM_TEST_MODE is defined, the test sends the key up/down scan codes for the test pangram:"a quick brown fox jumps over the lazy dog.<carriage return>". Otherwise the up/down scan codes are repeatedly sent for the test sequence 'wirelessusb' followed by the same number of backspaces. The test repeats the appropriate sequence until the escape key is pressed. When the test has finished execution, the keyboard returns to normal operation.

The repeating 'x' test selection is initiated by holding down the left Ctrl, left Alt, right Alt, right Ctrl, and F3 keys at the same time. The test continuously sends the 'x' key up/down scan codes. The test continues until the escape key is pressed. When the test has finished execution, the keyboard returns to normal operation.

Encrypt Module

This module may be conditionally compiled in to provide encryption/decryption support. Encrypted data transfers are typically used between RDK keyboard devices and the RDK bridge. Contact Cypress Applications support for the encryption source code.

5.3.3.3 Configuration Options

All configuration options for the application can be found in the config.h file, and some of them are defined in the Project > Setting > Compiler > Macro defines. Each option is explained below and can be changed to values that meet the developer's needs.

- KEYBOARD_KEEP_ALIVE_TIMEOUT

  When a key is held down, this configuration value sets the period at which the firmware generates a KEEP_ALIVE packet since the last keyboard report. The default is 65 milliseconds.

- KEY_DOWN_DELAY_SAMPLE_PERIOD

  This configuration value sets the period at which the firmware polls the hardware for keyboard events to transmit over the radio. This poll period is active only when the keyboard has not entered sleep because keys are currently being pressed. The default value is 10 milliseconds.

- KEYBOARD_DEBOUNCE_COUNT

  The button debounce logic detects changes in the button state and immediately indicates a change causing a report to be sent to the radio. The debounce logic then blocks out any further button state changes for the specified debounce time. This operation is somewhat different from the usual method of waiting for a button to stabilize during a debounce interval, and then reporting the change in button state. It is implemented this way to improve button-reporting latency.

This configuration value sets the debounce time for buttons that are pressed. It is measured in units of the poll rate. For example, if KEYBOARD_DEBOUNCE_COUNT is defined as '2' and KEY_DOWN_DELAY_SAMPLE_PERIOD is defined as 10, the button debounce time is 20 milliseconds. The default setting is '2'.

■ KEYBOARD_MULTIMEDIA_SUPPORT

This configuration definition is used to selectively compile support for multimedia (hot) keys. If this value is defined, then multimedia key support is compiled into the executable image. If it is not defined, then the multimedia support code is omitted.

■ KEYBOARD_TEST_MODES

This configuration definition is used to selectively compile code for keyboard test modes. If this value is defined, then test modes are compiled into the executable image. If it is not defined, then the test mode code is omitted.

■ KEYBOARD_TEST_MODE_PERIOD

This configuration value sets the period that the keyboard generates on test key presses. A key press consists of a scan code as the down key and a NULL as the up key. The default value is 10 ms.

■ PANGRAM_TEST_MODE

This configuration definition is used to selectively compile the pangram test mode. A pangram is a sentence that contains all of the letters of the alphabet at least once.

■ KEYBOARD_BATTERY_VOLTAGE_SUPPORT

This configuration definition is used to selectively compile support for battery voltage level reporting. If this value is defined, then battery voltage level reporting is compiled into the executable image. If it is not defined, then the battery voltage level reporting code is omitted.

■ LP_RDK_KEYBOARD_MATRIX

This configuration definition is used to selectively compile the keyboard matrix for the RDK keyboard hardware.

■ KEYBOARD_TX_TIMEOUT

This configuration value sets the maximum time that the keyboard tries to send a report to the bridge. The default value is 5000 ms.

■ TIMER_CAL

This configuration definition is used to selectively compile the one-millisecond timer calibration routine. The routine is called on power on and during protocol reconnect.

■ ENCRYPT_TEA

This configuration definition is used to selectively compile TEA encryption for the keyboard. Contact Cypress Applications support for the encryption source code.

■ ENCRYPT_AES

This configuration definition is used to selectively compile AES encryption for the keyboard. Contact Cypress Applications support for the encryption source code.

■ MFG_TEST_CODE

This configuration definition is used to selectively compile the manufacturing test code. The manufacturing test code in this keyboard is compatible with the CY3631 Manufacturing Test Kit offered by Cypress Semiconductor. See the Mfgtest Module for a description of how this test mode is executed. The CY3631 Manufacturing Test Kit documentation has a description of the test operation.

■ MFG_ENTER_BY_PIN

This configuration definition is used to select whether the manufacturing test code is executed by connecting pin 4 and 5 on the ISP (programming) header. When this value is not defined, then

the manufacturing test code may be executed by holding the system sleep key and the bind button when the batteries are inserted into the keyboard.

■ MFG_TX_MODES

When the MFG_TEST_CODE is defined, the definition of this name adds in a carrier and random data TX test option. See the Mfgtest Module for more information on these TX modes.

■ MOUSE_EMULATION_MODE

This configuration definition is used to selectively compile the mouse Emulation mode. The Scroll Lock key is used to toggle this mode ON/OFF. When in this mode, the arrow keys are used to move the mouse. The Delete key is the left mouse button, the End key is the right mouse button, and Page Up and Page Down emulate the scroll wheel.

■ BACK_CHANNEL_SUPPORT

This configuration definition is used to selectively compile the Back Channel Data Support feature. See Back Channel Data Support on page 100 for more information.

■ MASTER_PROTOCOL

This configuration definition is used to select the Master radio protocol or Slave radio protocol. For the keyboard application, it should be undefined.

■ PAYLOAD_LENGTH

This configuration definition is used to define the payload length. For the keyboard application, it should be defined as 8.

■ KISS_BIND

This configuration definition is used to selectively compile the Enhanced KissBind feature. See Enhanced KISSBind on page 98 for more information. The keyboard can be unbound by holding the 'Esc' key and 'Delete' key. After being unbound, the keyboard enters an infinite loop until a POR. After being unbound, the keyboard can be bound to a bridge by KissBind.

■ RSSI_QUALIFY

This configuration definition is used to enable the RSSI qualification for the Enhanced KissBind. Only if the RSSI reading is above KISS_BIND_RSSI_THRESHOLD, can the KissBind request/response be accepted by the Bridge/Devices.

■ PLATFORM_H

This configuration value identifies the header file that has the platform configuration information. The default value is pdc9265.h, which is identifier for the keyboard board that is shipped with the RDK. It is anticipated that this macro will change when the code is ported to another platform.

5.3.3.4   Platform and Architecture Portability

The keyboard firmware is designed to be easily ported from one hardware platform to another platform with a simple re-mapping of pins on the enCoRe II LV. The file pdc9265.h maintains the pin mapping definitions that are used throughout the code and is included in about every file by using the macro PLATFORM_H that is defined in config.h.

The keyboard scan matrix is defined in kdefs.h and may need to be changed for different keyboards. Porting the code to another microprocessor architecture requires modification or leverage of the existing code for processor specific features, along with pin definitions.

5.3.3.5   Initialization

Initialization of the enCoRe II LV chip is done by code that is generated in boot.asm by PSoC Designer. The module boot.asm calls main when the enCoRe II LV is configured and initialized.

Main initializes the components of the keyboard along with timer, isr and radio modules. The main routine then goes into an infinite loop monitoring keyboard activity and sleeping between keystrokes.

5.3.3.6 Wireless Protocol Data Payload

The keyboard protocol is optimized to reduce the ON time of the radio and power consumption. The radio driver offers the ability to send variable length packets, allowing the opportunity to minimize the number of bytes transmitted over the air, to extend battery life.

The following transmission packet formats are implemented in this RDK. The report formats show the application payload and the radio protocol overhead with example packet headers.

Keyboard Application Report Formats

The first byte of the data packet payload, byte 2 of the radio packet, is used as a keyboard application report header. There are five possible keyboard application reports. The reports are:

- Standard 101 Keys Report
- Multimedia Keys Report
- Power Keys Report
- Keep Alive Report
- Battery Voltage Level Report

The first application report byte is Scan Code 1 if the byte is less than 0xFC. Otherwise, the first application report byte is the Application Report Header (Multimedia, Power, Battery, or Keep Alive).

This also assumes that multimedia and power keys do not use modifier keys and that 0xFF, 0xFE, 0xFD and 0xFC are not valid Standard 101 key scan codes.

Trailing zeros in the reports are also removed to further minimize the number of bytes sent by the radio.

The LP radio sends the reports with the format shown in Table 5-15.

Table 5-15. LP Generic Report

Byte	1					2	..	..	N
Bits:	7:4	3	2	1	0	7:0	7:0	7:0	7:0
Field:	4	0	Toggle	ID	0	Application Report Header			Byte N

- Standard 101 Keys Report

If the Application Report Header byte is less than 0xFC then this indicates that this report is a Standard 101 Keys Report and the first byte is the actual scan code rather than the Report Header. This is done to optimize the packet size based on the fact that the most common report has only one non-zero scan code without a modifier. The full Standard 101 Keys report format is shown in Table 5-16.

Table 5-16. Standard 101 Keys Report Format

Byte	Name
2	Scan Code 1 (< 0xFC)
3	Modifier Keys
4	Scan Code 2
5	Scan Code 3
6	Scan Code 4
7	Scan Code 5
8	Scan Code 6

Example: The following reports is sent if you press an 'a' on the keyboard. The down key packet sent from the keyboard to the bridge is shown in Table 5-17.

Table 5-17. Example 'a' down key Standard 101 Keys Report

Byte 2
Scan Code 1
0x04

The bridge adds the trailing zeros, inserts the reserved byte, rearranges the modifier, and scans code 1 bytes and remove the packet header to produce the USB report shown in Table 5-18.

Table 5-18. Example USB report for the 'a' down key

Modifier keys	Reserved	Scan Code 1	Scan Code 2	Scan Code 3	Scan Code 4	Scan Code 5	Scan Code 6
0x00	0x00	0x04	0x00	0x00	0x00	0x00	0x00

The up key packet sent from the keyboard to the bridge (all data bytes are zero) is shown in Table 5-19.

Table 5-19. Example up key Standard 101 Keys Report

Byte 2
Scan Code 1
0x00

The bridge adds the trailing zeros, inserts the reserved byte, and removes the packet header to produce the USB report shown in Table 5-20.

Table 5-20. Example USB report for a Standard 101 Key Null Packet Report

Modifier keys	Reserved	Scan Code 1	Scan Code 2	Scan Code 3	Scan Code 4	Scan Code 5	Scan Code 6
0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00

■ Multimedia Keys (Hot keys) Report

An Application Report Header of 0xFF indicates that this report is a Multimedia Keys Report. The Multimedia Keys report format is shown in Table 5-21.

Table 5-21. Multimedia Keys Report Format

Byte	Name
2	Application Report Header 0xFF
3	Hot Key Scan Code (upper 8 bits)
4	Hot Key Scan Code (lower 8 bits)

Example: The following reports is sent if you press 'Volume Increase' (Hot Key 8) key on the keyboard. The 'Volume Increase' down key packet sent from the keyboard to the bridge is shown in Table 5-22.

Table 5-22. Example 'Volume Increase' down key Multimedia Keys Report

Application Report		
Application Report Header	Hot Key Scan Code (upper 8 bits)	Hot Key Scan Code (lower 8 bits)
0xFF	0x00	0xE9

■   Power

The up key packet sent from the keyboard to the bridge is shown in Table 5-23.

Table 5-23. Example up key Multimedia Keys Report

Application Report
Application Report Header
0xFF

■   Power Keys (Suspend/Sleep) Report

An Application Report Header of 0xFE indicates that this report is a Power Keys Report. The Power Keys report format is shown in Table 5-24.

Table 5-24. Power Keys Report Format

Byte	Name
2	Application Report Header (0xFE)
3	Power Key Scan Code

Example: The following reports is sent if you press the Suspend/Sleep (Power Key 0) key on the keyboard.

The Suspend/Sleep down key packet sent from the keyboard to the bridge is shown in Table 5-25.

Table 5-25. Example Suspend/Sleep Down Key Power Keys Report

Application Report	
Application Report Header	Power Key Scan
0xFE	0x02

The up key packet sent from the keyboard to the bridge is shown in Table 5-26.

Table 5-26. Example Up Key Power Keys Report

Application Report
Application Report Header
0xFE

■ Keep Alive Report

An Application Report Header of 0xFC indicates that this report is a Keep Alive Report. An example of a Keep Alive reports sent from the keyboard to the bridge is shown in Table 5-27.

Table 5-27. Example Keep Alive Report (Null Packet Support disabled)

Application Report
Application Report Header
0xFC

If the bridge does not receive a Keep Alive packet or an up key within a specified interval (DOWNKEY_TIME_OUT) while a down key is present, the bridge generates an up key to the computer.

■ Battery Voltage Level Report

An Application Report Header of 0xFD indicates that this report is a Battery Voltage Level Report. The Battery Voltage Level report format is shown in Table 5-28.

Table 5-28. Battery Voltage Level Report Format

Byte	Name
2	Application Report Header 0xFD
3	Battery Voltage Level

The Battery Voltage Level ranges from 1 (low) to 10 (full). The Battery Voltage Level Report is sent after a keystroke that occurs whenever the keyboard has been in idle for more than 8 seconds. An example of a Battery Voltage Level Report with fully charged batteries is shown in Table 5-29.

Table 5-29. Example 'full' Battery Voltage Level Report

Application Report	
Application Report Header	Battery Voltage Level
0xFD	0x0A

An example of a Battery Voltage Level Report with low batteries is shown in Table 5-30.

Table 5-30. Example 'low' Battery Voltage Level Report

Application Report	
Application Report Header	Battery Voltage Level
0xFD	0x01

5.3.3.7   Ghost Key Detection

Ghost keys are possible on the RDK keyboard because it does not use diodes with the keyboard switches. Ghost keys are caused when three keys are pressed at the same time and two of the keys are on the same column and two of the keys are on the same row. When scanning the keyboard, it appears that four keys have been pressed and it is impossible to tell which three of the four keys are actually valid. The keyboard code detects this condition and does not send a report until one of the three keys is released.

For example, assume the keys (RowX, ColumnA), (RowX, ColumnB), and (RowY, ColumnA) have been pressed as shown in Figure 5-21. It appears that the key (RowY, ColumnB) has been pressed as well when it has not because the other keys electrically connect RowY to ColumnB.

Figure 5-21. Ghost Key Example
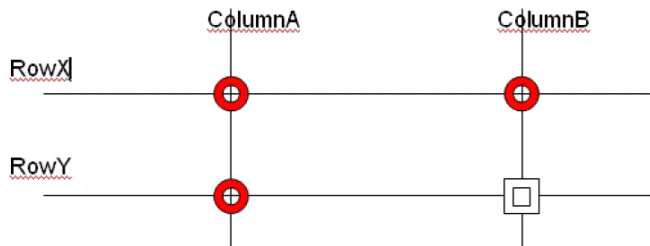
5.3.3.8   Interrupt Usage and Timing

In the RDK keyboard, the following interrupts are enabled:

■ Row Port interrupt

■ Bind button interrupt

When either of the above interrupts occurs, its ISR sets the flag. The interrupt latency includes two portions. The first portion is the time between the assertion of an enabled interrupt and the start of its ISR, which can be calculated using the following equation:

Latency1 = Time for current instruction to finish + Time for M8C to change program counter to interrupt address + Time for LJMP instruction in interrupt table to execute.

For example, if the 5-cycle JMP instruction is executing when an interrupt becomes active, the total number of CPU clock cycles before the ISR begins is as follows:

(1 to 5 cycles for JMP to finish) + (13 cycles for interrupt routine) + (7 cycles for LJMP) = 21 to 25 cycles.

In the example above, at 12 MHz, 25 clock cycles take 2.083 μs. The second portion is the time between the start of the ISR and the set of the flag. For example, the row port interrupt (caused by pressing any key) takes 19 CPU clock cycles for this portion. Therefore, the Latency2 equals to 1.583 μs for the 12 MHz CPU.

Consequently, the total latency for a button interrupt is Latency1 + Latency2 = 3.667 μs

5.3.3.9   Code Performance Analysis

A key press report is used to analyze the code performance. A typical key press report contains the following steps:

■ A key press interrupts the MCU. As calculated in the previous section, it takes 3.667 μs for MCU to responds to this Interrupt.

■ MCU exits the sleep state, scans the bind button and turns on the timer. It takes 40.8 μs.

- MCU calls function scan_keyboard() to detect which key is pressed. This function consumes 1.15 ms.
- MCU calls function generate_standard_report () to format the report and send the report to the bridge. This step takes 2.01 ms, which includes 1.66 ms radio transmission time.

As a result, it takes 3.20 ms for the keyboard to report a key press to the bridge.

5.3.3.10   Modifying Keyboard Matrix or Adding New Keys

The current keyboard matrix with the USB scan codes are shown in Table A-6. Customers may modify the keyboard matrix or they may add new keys to their keyboard. The following sections explain the procedure.

Modifying the Keyboard Matrix: In the file kdefs.h, a table called default_keyboard_scan_table matches the keyboard matrix shown in Table A-6. By modifying this table, the keyboard matrix is automatically modified.

Adding New Keys: Example: The customer wants to add a multimedia key called My Computer, which is located at Column 15 and Row 6 and has a scan code of 0x0194. The following steps must be performed:

1. Go to file kdefs.h, and search for default_keyboard_scan_table. In the Col 15 (0xF) section, modify line 7 from {NO_DEVICE, NOKEY} to {DEVICE_2, 0x000E}. The 0x000E is the index into the device 2 table.

2. Go to the table called device_2_keyboard_scan_table within the same file and add the scan code of 0x0194 to the end of the table, as shown:

```
const UINT16 device_2_keyboard_scan_table[] =
{
0x0192, // Calculator
0x0223, // WWW Home
0x00CD, // Play/Pause
0x0221, // WWW Search
0x018A, // Mail
0x00E9, // Volume Up
0x00E2, // Mute
0x00B7, // Stop
0x00EA, // Volume Down
0x022A, // WWW Favorites
0x0225, // WWW Forward
0x0224, // WWW Back
0x00B6, // Scan Previous Track
0x00B5, // Scan Next Track
0x0194, // My Computer\
};
```

3. Build the firmware, the new key My Computer will work.

5.3.4   Verify Output

1. Connect the bridge dongle to the PC, power the keyboard with the 2 AA batteries.

2. Press the bind button on the bridge, the red LED blinks ON/OFF in the bind mode. Now press the bind button on the keyboard the red LED stops blinking on binding the keyboard with the bridge.

3. Press any key on the keyboard the green LED turns on when the dongle receives data from the keyboard.

4. The green LED turns on and remains on if a key is pressed and held (due to the keyboard's sending Keep Alive packets).

CY4672 PRoC LP Reference Design Kit Guide, Doc. # 001-69289 Rev.*A

A.   Appendix

A.1   Schematic

Figure A-1. Bridge
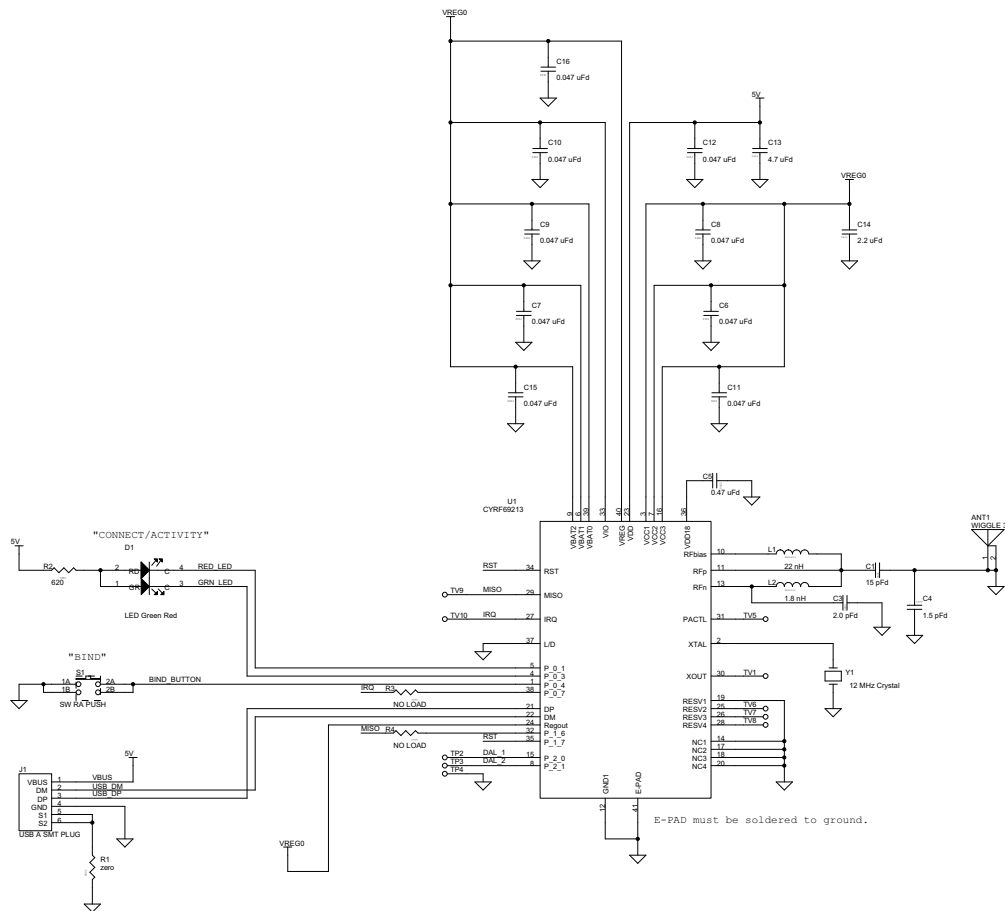
Figure A-2. Mouse
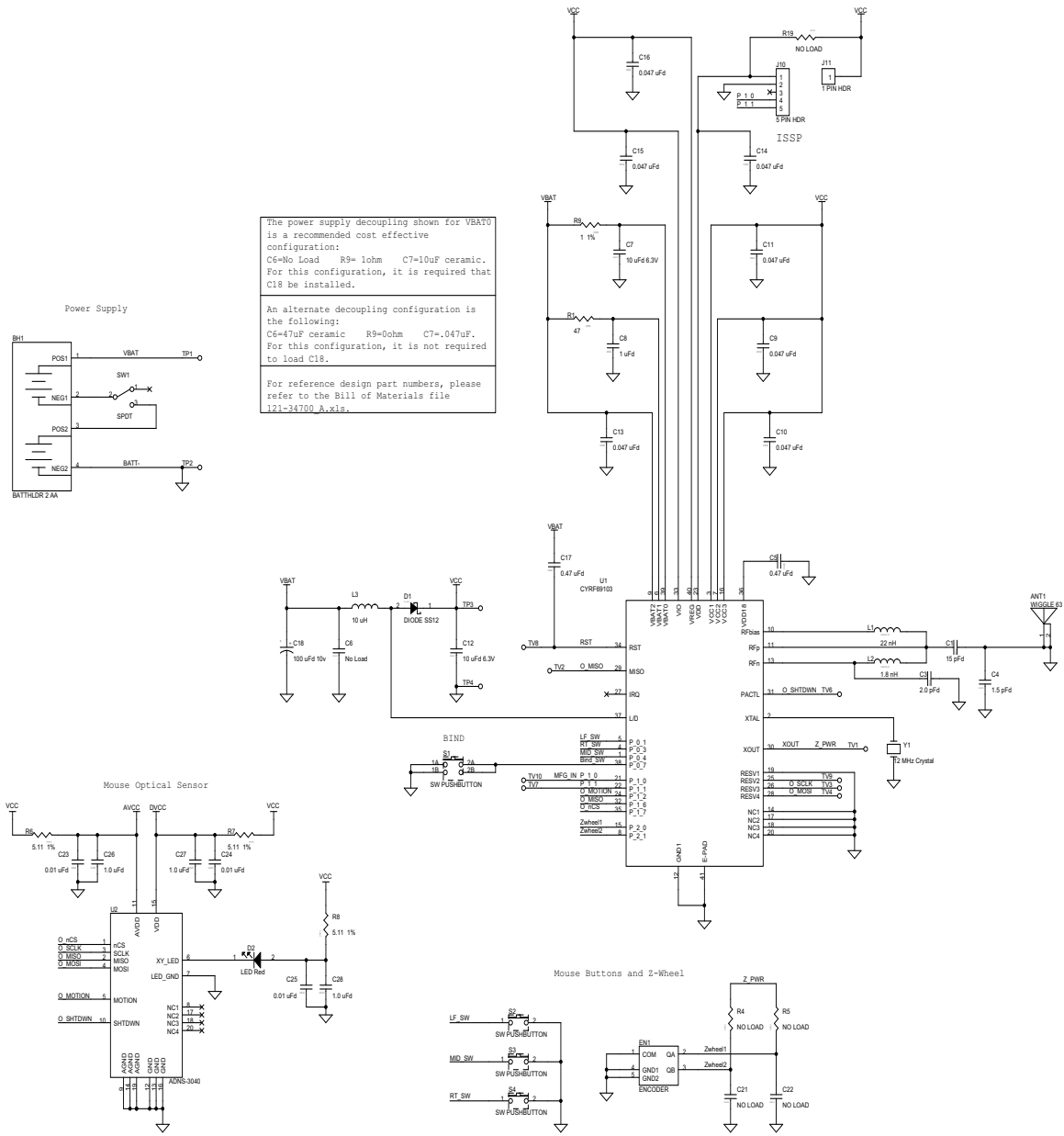
Figure A-3. Keyboard
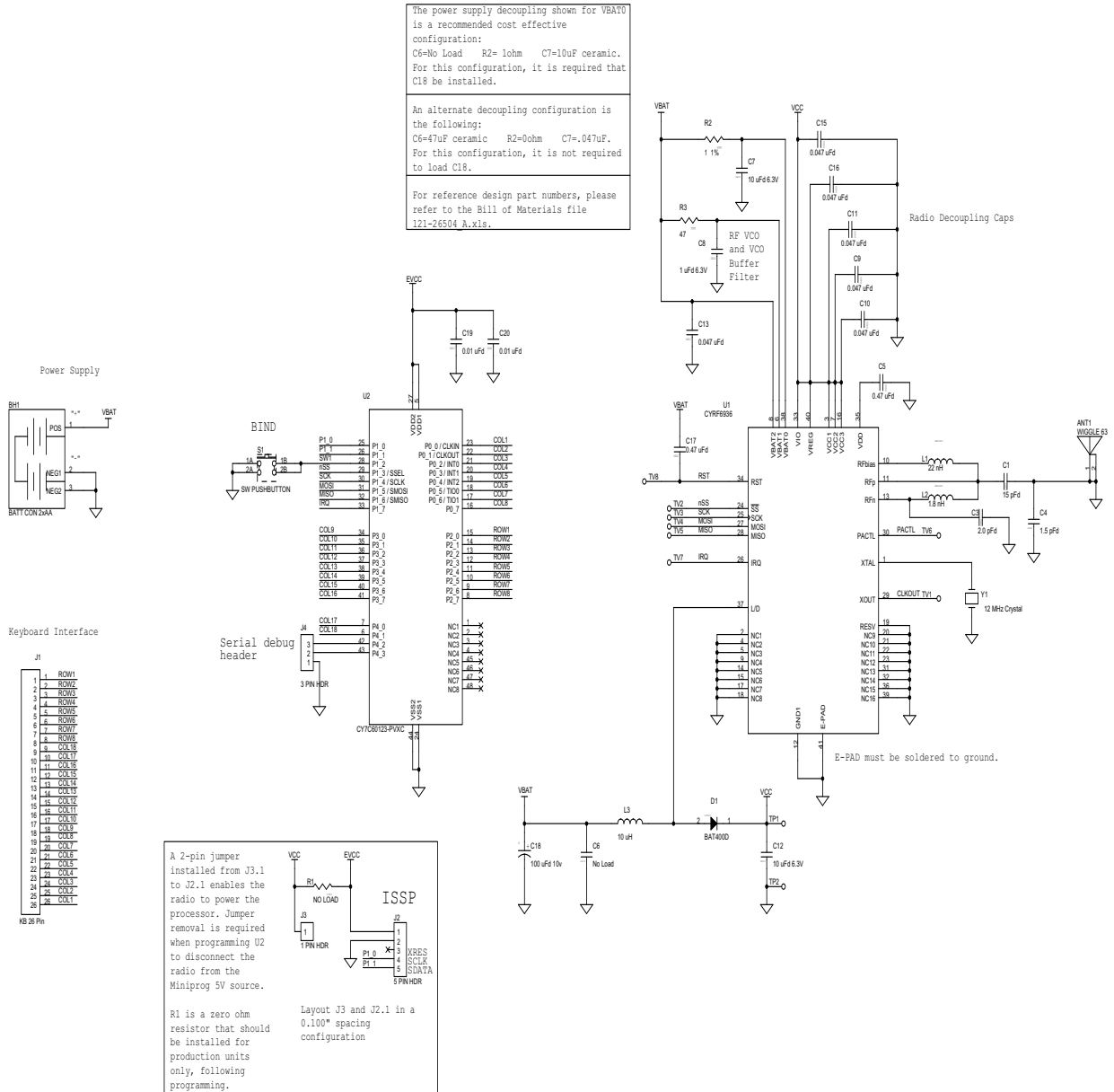
A.2   Board Layout

Figure A-4. Bridge Top
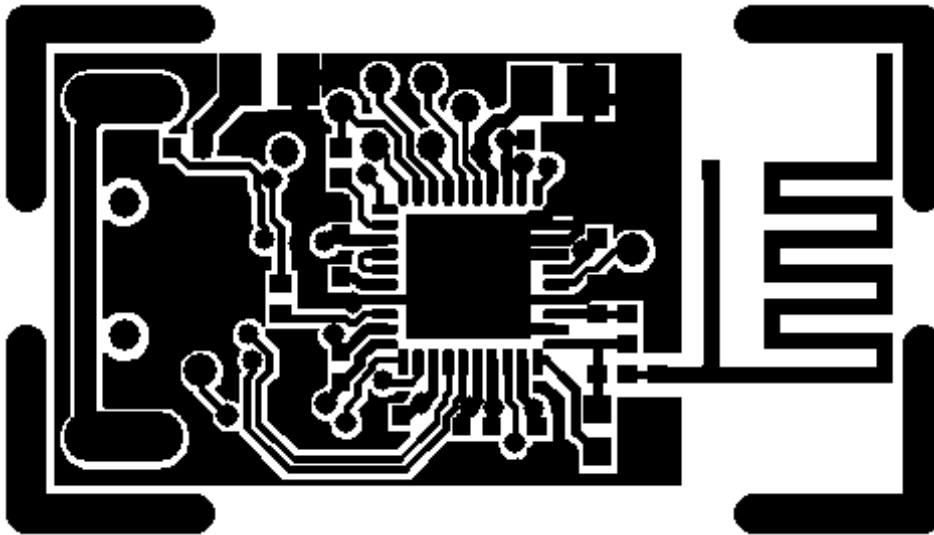
Figure A-5. Bridge Bottom
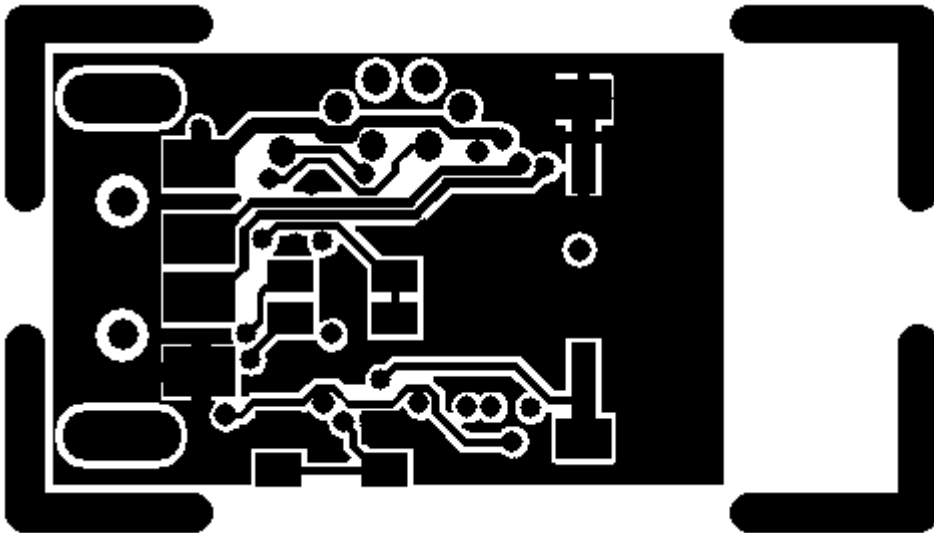
Figure A-6. Mouse Top

Figure A-7. Mouse Bottom

CY4672 PRoC LP Reference Design Kit Guide, Doc. # 001-69289 Rev.*A
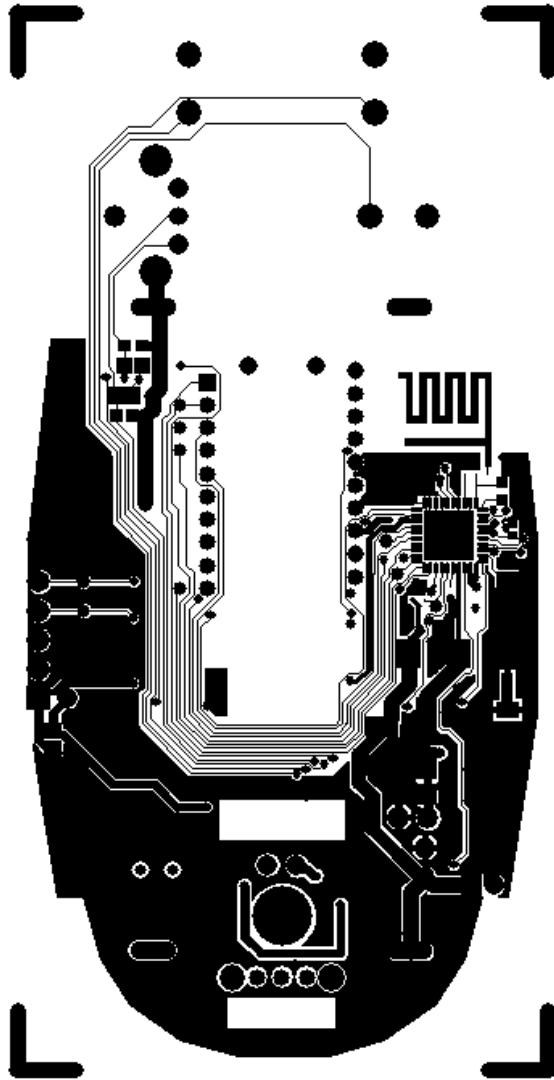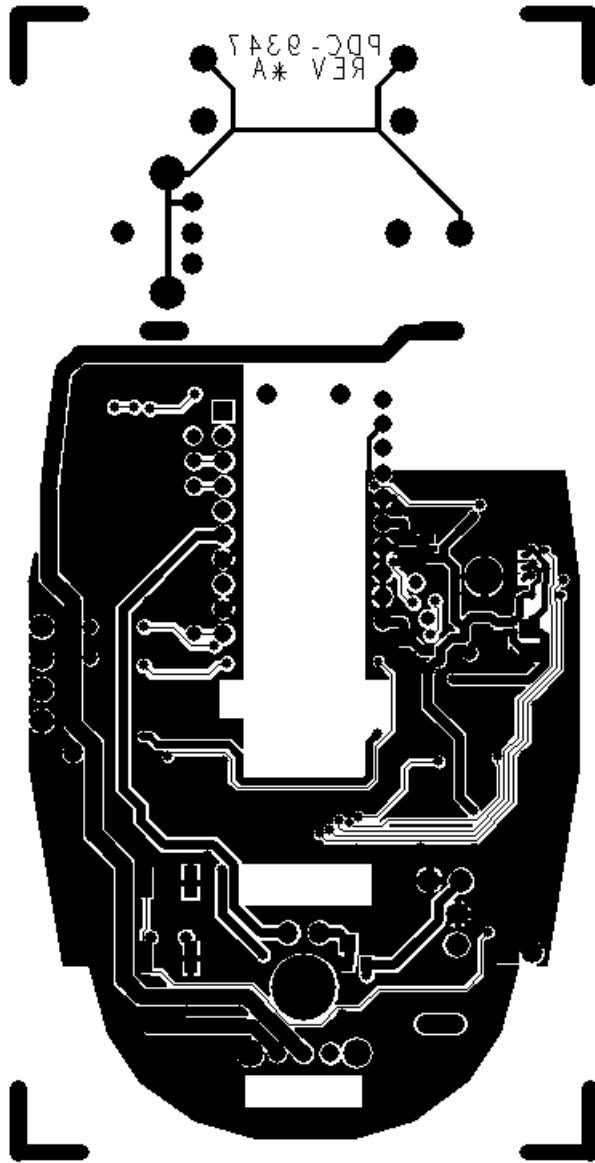
Figure A-8. Keyboard Top
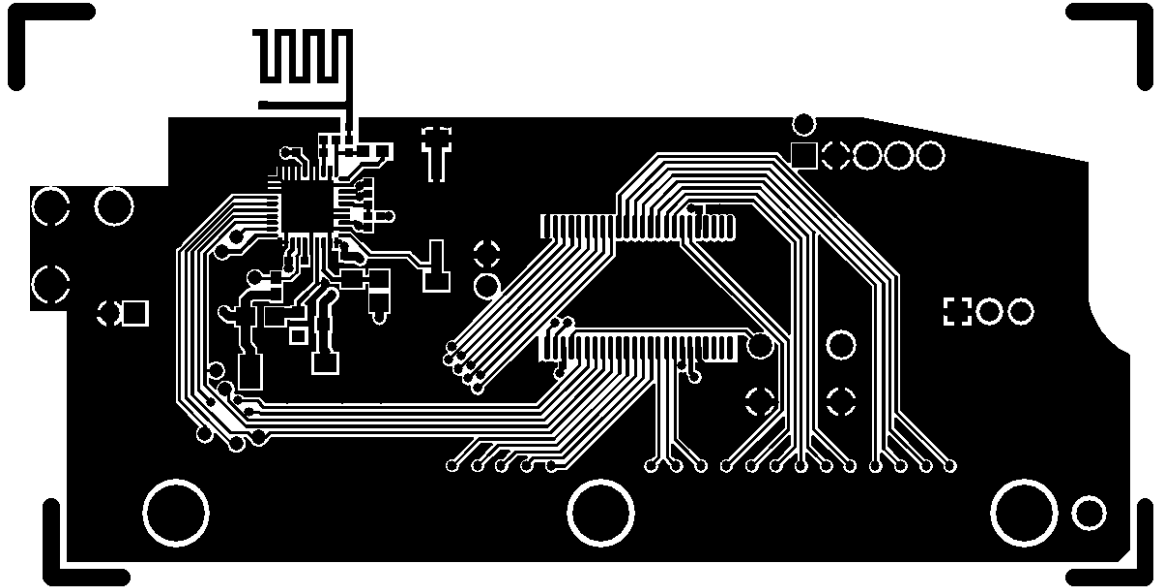
Figure A-9. Keyboard Bottom
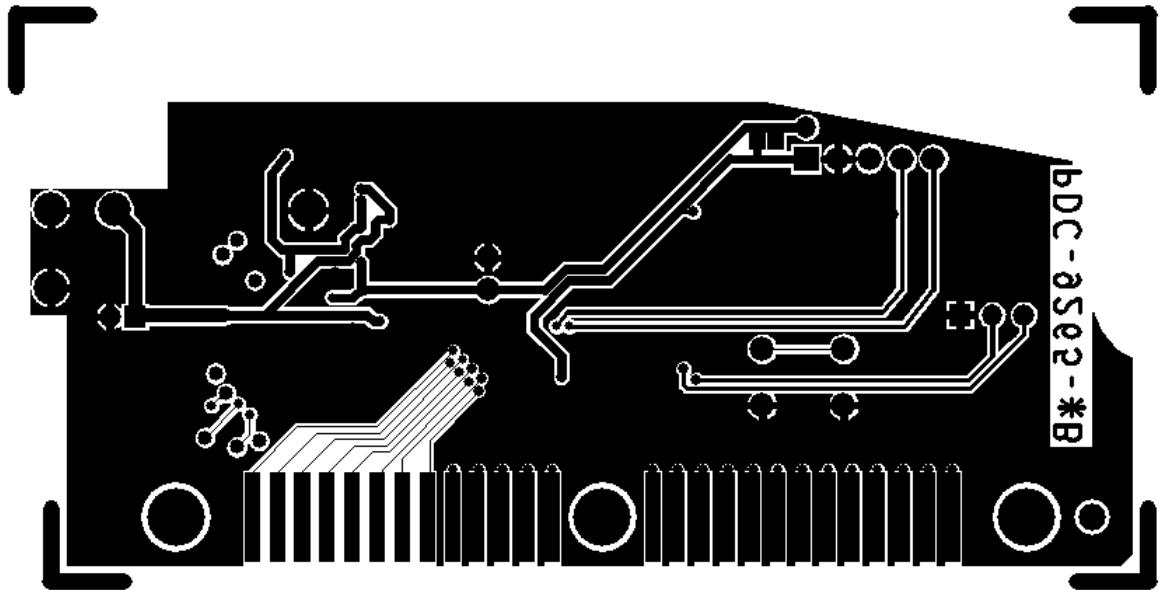
A.3 WirelessUSB Two-Way HID Protocol Overview

The WirelessUSB protocol 2.2 is designed to address two-way HID and general purpose devices; it provides reliable two-way communication between a wireless device configured as 1:1 (one HID and one bridge) or 2:1 (two HIDs and one bridge) systems. The WirelessUSB protocol 2.2 allows HID applications to establish a connection to the bridge and receive ACK and DATA packets.

Figure A-10. WirelessUSB Two-Way System
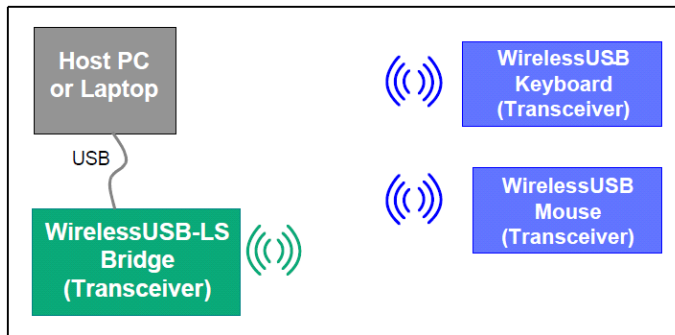
A.3.1 Radio Channel Management

WirelessUSB uses the unlicensed 2.4 GHz Industrial, Scientific, and Medical (ISM) band for connectivity. WirelessUSB uses 78 of the available channels and splits the 78 channels into 6 channel subsets consisting of 13 channels each. The channel subsets are used by each network to minimize the probability of interference from other WirelessUSB systems (see "Channel Selection Algorithm" on page 95 for more details). A designated channel subset is used during bind mode (along with an associated pseudo noise code) to enable all WirelessUSB devices to effectively communicate during this procedure.

A.3.2 Pseudo-Noise Codes

Pseudo-Noise codes (PN codes) are used to achieve the special matched filter characteristics of direct sequence spread spectrum (DSSS) communication. Certain codes referred to as 'multiplicative codes' are used for WirelessUSB two-way communication. These codes have minimal cross-correlation properties, meaning they are less susceptible to interference caused by overlapping transmissions on the same channel. The length of the PN code results in different communication characteristics. Higher data rates are achieved with 32-chips/bit PN codes, while 64-chips/bit PN codes allow a longer range. The number of frequency/code pairs is large enough to comfortably accommodate hundreds of WirelessUSB devices in the same space. Each bridge/HID pair must use the same PN code and channel to communicate with each other.

A.3.3 Chip Error Correction

In the presence of interference (or near the limits of range), the transmitted PN code is often received with some PN code chips corrupted. DSSS receivers use a data correlator to decode the incoming data stream. WirelessUSB LP supports a separate start of packet (SOP) and data threshold. The RDK uses an SOP threshold of '4'. The data threshold is set to the default value of '4'.

A.3.4 Automatic Acknowledgment

The WirelessUSB LP radio contains an automatic acknowledgment feature that allows it to automatically send an ACK to any valid packet that is received. The WirelessUSB LP radio also uses the concept of transactions to allow the radio in the HID to automatically power down after transmitting a

packet and receiving an AutoACK instead of waiting for the firmware to power the radio down. This conserves power and reduces the firmware complexity of WirelessUSB applications.

A.3.5   Network ID

The network ID contains the parameters for the channel selection algorithm and the PN code to be used. HIDs retrieve the network ID information from the bridge during the bind procedure. A special network ID is reserved for bind mode, known as the bind ID. The bind ID gives a common channel subset so that any two devices can communicate with each other during bind mode. The network ID is composed of the following fields:

PIN - This is a random number, between 2 and 5, that defines the channel subset and is used in the channel selection algorithm.

Base Channel - This is the first channel to be used in the channel selection algorithm that determines which channels are contained in the channel subset.

PN Code - This is used as an index to select one of 10 used SOP PN codes, as noted in the radio driver.

CRC Seed - This 8-bit value is used for the CRC calculation that further diversifies transmissions from different networks. All packets sent between non-bound devices use the default CRC seed of 0x0000. All packets sent between bound devices use a CRC seed that is common to all devices bound to a particular bridge or network, but unique from network to network.

A.3.6   Manufacturing ID

Each WirelessUSB radio contains a 4-byte manufacturing ID (MID), that is laser fused into the device during manufacturing. The bridge uses its MID to help randomize channel subsets, PN codes, and network CRC seeds. The bridge sends its MID to the HIDs when binding. The HID then stores the bridge's MID in nonvolatile memory after binding. The HID sends the bridge's MID as part of the connect request packet, allowing the bridge to verify the identity of the HID when establishing a connection.

Both the bridge and the HID use the bridge's MID to generate the device network ID components. The following equations ensure that each network has a unique set of network ID components:

PN Code = [(mid_1 << 2) + mid_2 + mid_3] mod 10

Base Channel = [(mid_2 >> 2) - (mid_1 << 5) + mid_3] mod 78

PIN = [(((mid_1 -mid_2) & PIN_MASK) + MIN_PIN)] mod 78

CRC Seed = ((mid_2 >> 6)) + mid_1 + mid_3 if=0 then Seed = 1

A.3.7   Channel Selection Algorithm

The channel selection algorithm produces a subset containing 13 of the possible 78 channels. The channel selection algorithm is based on the Network ID, with each channel in the subset 6 MHz from the nearest neighboring channels in the subset. This algorithm reduces the possibility of multiple bridges selecting the same channels in the same order at the same time.

A.3.8   Protocol Modes

Figure A-11. Protocol Master
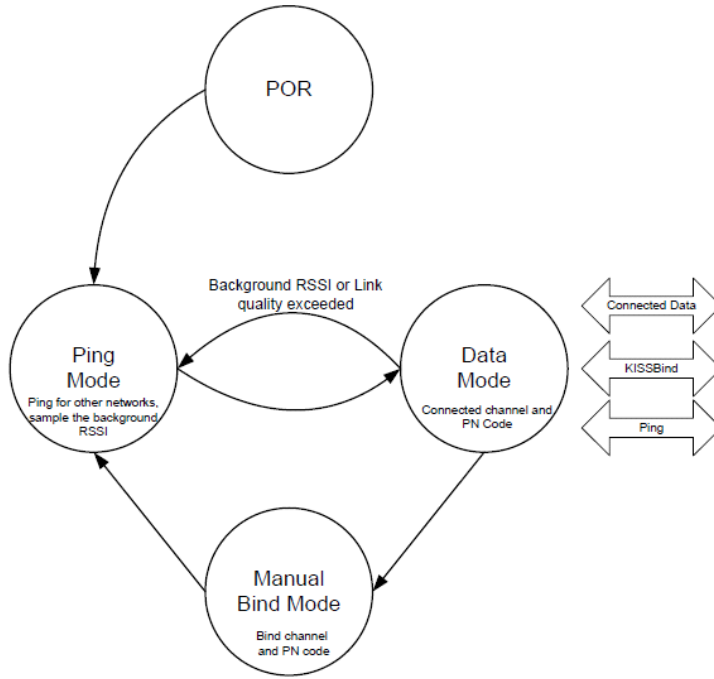
Figure A-12. Protocol Slave
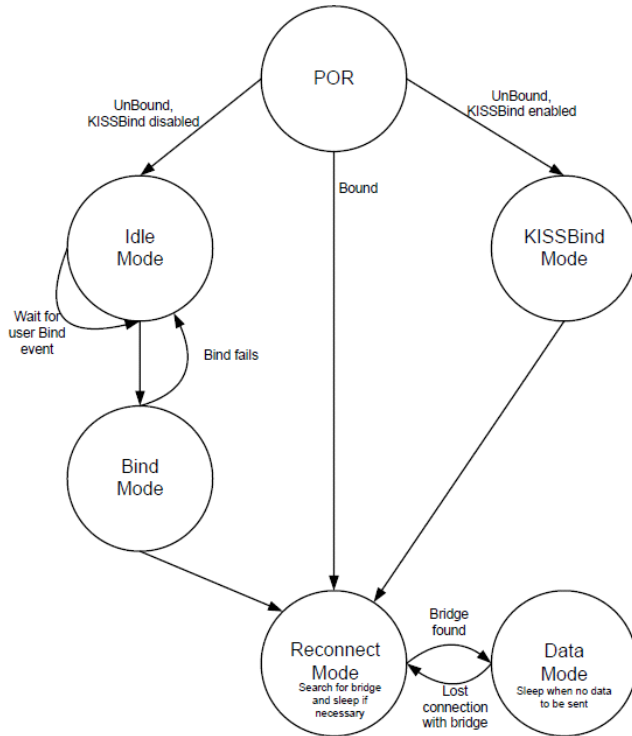
A.3.8.1 Ping Mode (Bridge Only)

Ping mode is used by the bridge to find an available channel; channels are unavailable if they are being used by another network with the same PN code, or if there is excessive noise on the channel. The bridge first listens for activity on the selected channel. If the channel is inactive, the bridge alternately transmits ping packets and listens for ping response packets for a defined period of time. During ping mode the bridge also checks the Receive Signal Strength Indicator (RSSI) of the radio to determine if a non-WirelessUSB device is using this channel (or a WirelessUSB device on the same channel using a different PN code). If a ping response is received, indicating that another bridge is using this channel the bridge selects the next channel using the channel selection algorithm and repeats this procedure. The bridge also selects another channel using the channel selection algorithm if RSSI is high; this indicates that there are other RF sources on the channel. If a ping response is not received and RSSI is low, the bridge assumes the channel is available and moves to data mode. Bridges send ping response packets in response to all received ping packets if the bridge is in data mode. HIDs never respond to ping packets.

Note The timeout value is configurable using the PING_NUM_RSSI define.

A.3.8.2 Idle Mode (HID only)

The HID enters this mode after a power on reset before it has had any communication with the RDK bridge. If the bridge's MID is stored in nonvolatile memory, the HID retrieves the bridge's MID, calculates the network ID and moves to reconnect mode. If the bridge's MID is not stored in nonvolatile-memory, the HID waits in idle mode until a user-initiated event causes the HID to enter bind mode. After a defined period of time in idle mode, the HID goes to sleep to conserve power. When the HID wakes up due to a user action, it re-enters Idle mode.

A.3.8.3 Reconnect Mode (HID only)

Reconnect mode is used by the HID to discover the current channel used by the bridge and to establish a connection with the bridge. Upon entering reconnect mode, the HID uses the network ID to select a channel using the channel selection algorithm. The HID transmits 'connect requests' containing the manufacturing ID of the desired bridge and listens for an AutoACK. If an AutoACK is received, the HID disables the AutoACK and continues to listen for a 'connect response'. If a bridge in data mode receives a 'connect request' containing its manufacturing ID, it sends a positive 'connect response' to the HID. If a HID receives a positive 'connect response', it moves to data mode. If a HID does not receive a positive 'connect response', it selects the next channel using the channel selection algorithm and repeats the procedure. If the HID does not receive a positive 'connect response' on any of the channels in the subset, it enters goes to sleep to conserve power. When the HID wakes up due to a user action it reenters reconnect mode.

A.3.8.4 Bind Mode

HID

Bind mode allows the HID to retrieve the bridge's manufacturing ID, which is used to calculate the network ID. Upon entering bind mode, the HID sets the current channel and PN code to the channel and PN code specified in the bind ID. The HID then transmits bind requests and listens for an Auto-ACK. If an AutoACK is received, the HID (keeping the AutoACK enabled) continues to listen for a bind response (containing the bridge's MID) from the bridge. If a bind response is not received, the HID moves to the next channel. If a bind response is received, the HID stores the bridge's MID, calculates the network ID, and moves to reconnect mode. The algorithms used to calculate these fields are implementation specific and should be the same on the bridge and the HID (both devices use the bridge's manufacturing ID to calculate these fields). If a defined period of time has elapsed while in bind mode without receiving a bind response, the HID exits bind mode and restores the channel and PN code settings that were in use before entering bind mode. Bind mode should last long enough to

locate and push the button on both the bridge and the HID. A user-initiated event can cause the HID to enter bind mode from any other mode.

Note The timeout value is configurable using the BIND_RETRY_COUNT define.

Bridge

Upon entering bind mode, the bridge sets the current channel and PN code to the channel and PN code specified in the bind ID. The bridge listens for a bind request on each channel for approximately 320 ms before selecting the next channel using the channel selection algorithm. This reduces the possibility of the bridge not receiving the bind request from the HID in the event of channel interference. If the bridge receives a bind request from the HID containing a supported device type, it sends a bind response containing the bridge's manufacturing ID and then switches to ping mode. The bridge also switches to ping mode if the defined time period has elapsed while in bind mode. The channel selection algorithm uses the bind ID to produce the channel subset for bind mode.

Note The timeout value is configurable using the NUM_CHANNELS_PER_SUBSET define.

A.3.8.5   Enhanced KISSBind

KISSBind provides the ability to automatically bind out of the box without any intervention other than installing the batteries. KISSBind essentially is a bind process while in the data mode. The bridge goes through the normal process of pinging and then going to the data mode. The device upon powering up and determining that it is not bound, transmits KISS_BIND_REQ packets on all channels and PN codes looking for a bridge that is not bound to that specific device. The bridge keeps track of which device is connected and only responds with a KISS_BIND_RESP packet if it is not already bound to that specific device. When bound, the bridge stores the device specific state in flash.

HID

When the HID first powers up it checks the validity of the flash bind parameters. If the bind parameters checksum is not valid, then the HID is considered to be unbound. The HID then transmits KISS-Bind request packets on all channels and PN codes using a CRC seed of zero to locate the bridge. If an AutoACK is received, the HID enters the receive state to listen for a KISSBind response packet from the bridge. The HID completes the KISSBind process if a KISSBind response packet is received from the bridge. If, after RX_PACKET_TIMEOUT (ms), the HID does not receive from the bridge, it then resumes the channel/PN code search for the bridge. If the search sequence is unsuccessful after BIND_RETRY_COUNT attempts, then the HID enters a low power state waiting for a button press or other activity and begin the search process all over.

Bridge

The bridge, upon powering up, enters the ping mode to locate a suitable channel/PN code based on its MID. When the ping mode is complete, the bridge enters the data mode. If a KISSBind Request packet is received, the bridge checks the bind status for the specific device that sent the KISSBind request (mouse or keyboard) based on the device type in the packet header. If the specific device is not bound, then the bridge proceeds by sending a KISSBind response packet and completing the KISSBind process. When an AutoACK is received from the HID in response to the KISSBind response, the bridge updates the flash bind status for the specific device.

Figure A-13. KISSBind Transaction Sequence
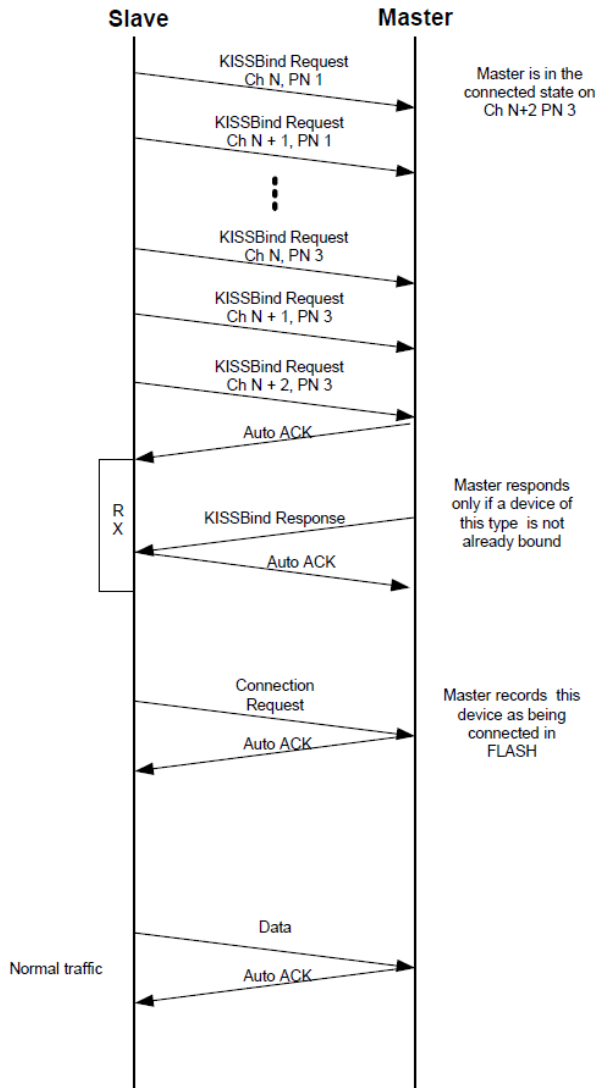
A.3.8.6   Unbind

An 'unbind' mechanism allows the bridge and HIDs to return to their default unbind mode as if they had never bound to any system before. The bridge dedicates a bind flag to each device type that it supports. A bind flag is a 1-bit field in =Flash. When the bridge is bound to an HID by either KISS-Bind or button bind mechanism, the bridge sets the corresponding bind flag for that device type and stores the flag in its flash. If the bind flag for a particular device type is set, the bridge treats future KISSBind packets from this device type as nonfunctional packets. The bridge unbind process clears all bind flags, and the bridge allows devices to KISSBind. The HID dedicates a byte in its flash, called SIGNATURE, to indicate whether or not the HID has bound to a bridge before. The SIGNATURE is set to 0x90 after a successful KISSBind or button bind. If the SIGNATURE is not set to 0x90, the HID tries to KISSBind to any bridge in the area that allows the HID to KISSBind. When the SIGNATURE is set, the HID does not attempt to KISSBind.

Note When the HID enters unbind mode, power-cycle the HID to exit this mode. When the bridge is unbound, the bridge continues to communicate with any HID that already has the bridge MID. To completely unbind the system, the HIDs and bridge must be unbound.

A.3.8.7 Data Mode

HID

When the HID application has data to send to the bridge the HID transmits a DATA packet and listens for an AutoACK. If an AutoACK is not received, the HID retransmits the packet. If the HID does not receive an AutoACK after N DATA_PACKETS_RETRIES of retransmissions of the data packet it assumes the channel has become unavailable due to excessive interference and moves to reconnect mode.

Bridge

Data mode allows application data to be transmitted from the HID to the bridge. The bridge continuously listens for data packets from the HID. When valid data is received from the HID the bridge sends an ACK to the HID and sends the data to the USB host. If invalid data is received the bridge ignores the packet and listens for the HID to retransmit the data. The bridge monitors the interference level and moves to ping mode if the RSSI interference threshold RSSI_NOISE_THRESHOLD is reached. This ensures that the bridge is operating on a clean channel.

A.3.8.8 Back Channel Data Support

Back channel data support provides a mechanism for the host to send data to the device at the request of the device. The device is responsible for interrogating the bridge for back channel data either as part of a forward data packet or a simple null packet. The device starts by setting the BCDR bit in the data header. If the packet is successfully acknowledged by the bridge then the device inverts the upper byte of the checksum seed and then waits for N ms before trying to receive from the bridge for M ms. The bridge also inverts the checksum seed and wait N ms before attempting to transmit to the device. If the bridge has more data to send then it can also set the BCDR flag and can then expect the device to receive another packet.

Figure A-14. Back Channel Transaction Sequence
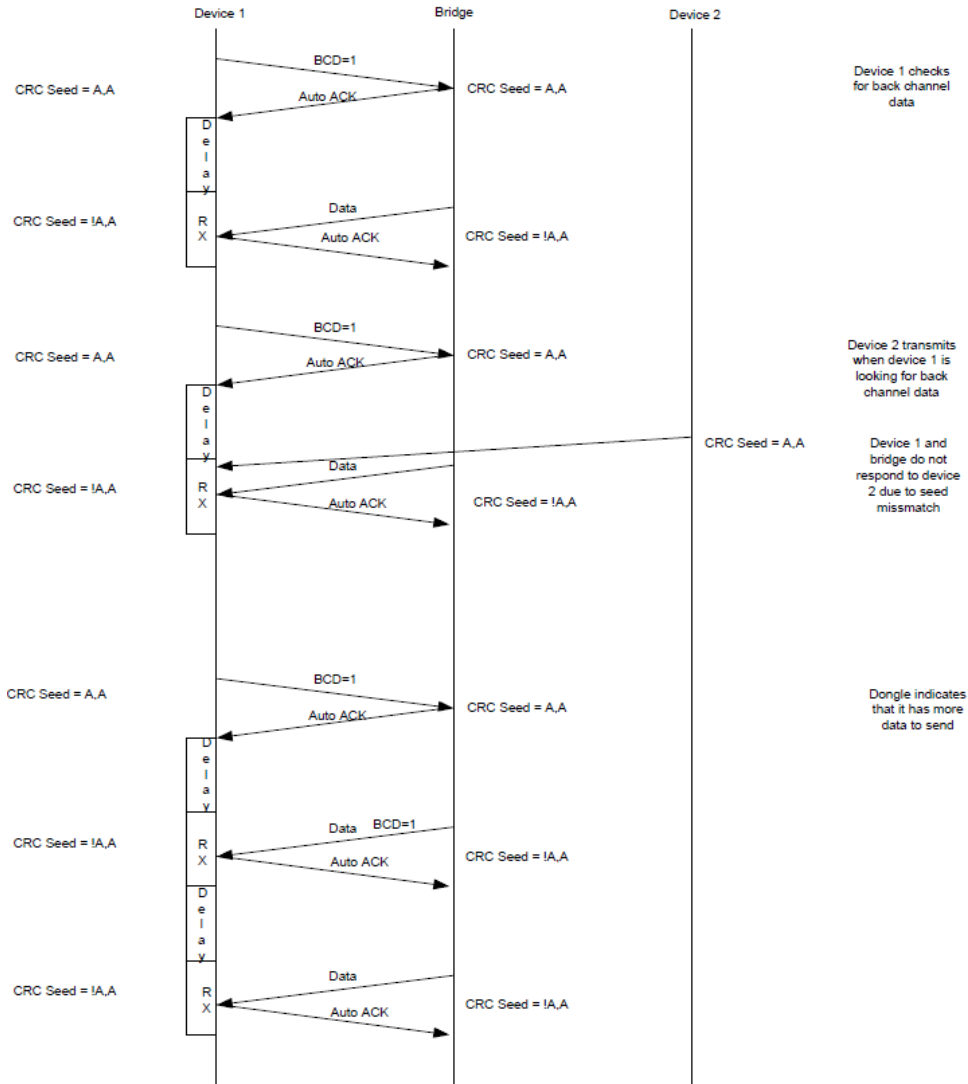
A.3.8.9 Dynamic Data Rate and Dynamic PA

Dynamic data rate and dynamic PA provide the ability to improve the immunity to interference and reduce power consumption. Dynamic data rate is device behavior based and two data modes, GFSK and 8DR, are used for the data transmission. Depending on the retry number of prior packets, the protocol determines whether to stay with the current data mode or change to another data mode. The dynamic PA relies on both the behavior and the bridge signal strength to the device.

HID

■ Dynamic Data Rate

If the HID fails to transmit either the application or protocol data to the bridge after DATA_PACKET_RETRIES of retransmissions, it toggles data modes and transmit again. If the HID still fails after DATA_PACKET_RETRIES of retransmissions, it assumes the channel has become unavailable due to excessive interference and moves to reconnect mode.

If the HID transmits application data successfully, the retries for PKT_NUM packets are summed up in 'total_retry'. If the total_retry exceeds the threshold 'total_retry_threshold', the HID changes to another data mode.

The total_retry_threshold is an adaptive number that has a minimum value of TOTAL_RETRY_THRESHOLD_LOW. The following rules are applied to it:

- ❏ When the data mode is toggled, the total_retry for the previous data mode is used as current total_retry_threshold.
- ❏ If the total_retry is zero, the total_retry_threshold is decreased by one until TOTAL_RETRY_THRESHOLD_LOW.
- ■ Dynamic PA

If the total_retry is zero and the RSSI reading for the bridge AutoACK packet is above PA_RSSI_RX_THRESHOLD, the PA is decreased by one until DATA_MODE_PA_MIN. When one of the following cases occurs, the PA is set back to its maximum value of

DATA_MODE_PA_MAX:

- ❏ The data mode is toggled.
- ❏ The RSSI reading for the bridge AutoACK packet is equal to or below PA_RSSI_RX_THRESHOLD.
- ❏ The retries for any packet exceeds PA_RETRY_THRESHOLD.

Bridge

The bridge does not need to implement the dynamic PA because it is bus powered. Its PA is always set to the maximum value DATA_MODE_PA_MAX to get the highest transmission power. The dynamic data rate is driven by the devices. When the bridge receives the packet, it sets the transmission data mode to the data mode of received packet. As a result, when it sends the back channel data, it uses the same data mode as the device.

A.3.9   Packet Structures

The first byte of each packet is the Header byte. Some packets may consist only of the header byte while other packets may contain up to 5 bytes.

Byte	1	
Bits:	7:4	3:0
Field:	Packet Type	Res.

Type[7:4]: The following packet types are supported:

BIND_REQ (HID) = 0x0, // Bind Request Packet Type

BIND_RESP (bridge) = 0x0, // Bind Response Packet Type

CONNECT_REQ = 0x1, // Connect Request Packet Type

CONNECT_RESP = 0x2, // Connect Response Packet Type

PING_PACKET = 0x3, // Ping Packet Type

DATA_PACKET = 0x4, // Data Packet Type

BACK_CHANNEL_PACKET = 0x5, // Back Channel Packet Type

NULL_PACKET = 0x7, // Null Packet Type

ENCRYPT_KEY_REQ = 0x8, // Key Packet Type for encryption

ENCRYPT_KEY_RESP = 0x8, // Key Packet Type for encryption

KISS_BIND_REQ = 0xD, // KISSBind request

KISS_BIND_RESP = 0xD, // KISSBind response

Res[3:0]: The lower nibble is used for packet specific information. The packet definitions below define how these four bits are used in each case.

A.3.9.1 Bind/KISSBind Request Packet (HID)

Byte	1			
Bits:	7:4	3	2.1	0
Field:	0/0D	Reserved	Device Type	Reserved

Byte 1

Packet Type: 0 for bind request and 0xD for KISSBind request.

Device Type: This is a 2-bit field that specifies a vendor-defined device type. This allows the bridge to determine HID type.

Currently the following device types are defined in the PRoC LP RDK:

0x0 Presenter

0x1 Reserved

0x2 Keyboard

0x3 Mouse

A.3.9.2 Bind Response Packet (Bridge)

Byte	1				2	3	4	5
Bits:	7:4	3	2:1	0	7:0	7:0	7:0	7:0
Field:	0	Reserved	Device Type	Reserved	Bridge MID1	Bridge MID 2	Bridge MID 3	Bridge MID 4

Byte 1

Packet Type: 0 for bind request, 0xD for KISSBind request

Device Type: This is a 2-bit field that specifies a vendor-defined device type. This allows the bridge to determine HID type.

Currently the following device types are defined in the KBM RDK:

0x0 Presenter

0x1 Reserved

0x2 Keyboard

0x3 Mouse

Byte 2-5

Manufacturing ID (MID 1-MID 4): This is the 4-byte manufacturing ID retrieved from the bridge's radio and is used by the HID.

A.3.9.3 Connect Request (HID)

Byte	1				2	3	4	5
Bits:	7:4	3	2:1	0	7:0	7:0	7:0	7:0
Field:	1	Reserved	Device Type	Reserved	Bridge MID 1	Bridge MID 2	Bridge MID 3	Bridge MID 4

Byte 1

Device Type: 0x1

Byte 2-5

Manufacturing ID (MID 1-MID 4): This is the 4-byte MID that is received from the bridge during the bind procedure. This enables the bridge to identify if the HID belongs to its network.

A.3.9.4 Connect Response Packet (Bridge)

Connect response packets are sent from the bridge to the HID in Idle and data mode in response to valid connect requests.

Byte	1		
Bits:	7:4	3	2:0
Field:	2	Flag	Reserved

Byte 1

Packet Type: 2

Flag (F): This is a 1-bit field that specifies a positive or negative connect response packet (1 = positive, 0 = negative).

A.3.9.5 Ping Packet (Bridge)

Byte	1		
Bits:	7:4	3:1	0
Field:	3	Reserved	Flag

Byte 1

Packet Type - 3

Flag (F) - This is a 1-bit field specifying a Ping or Ping Response (0 = Ping, 1 = Ping Response).

A.3.9.6 Data Packet (Bridge and HID)

Data packets are sent from the HID to the bridge in Connected Mode. They are also sent from the bridge to the HID in Connected Mode if there is an asynchronous back channel.

Byte	1					2	..	..	N
Bits:	7:4	3	2	1	0	7:0	7:0	7:0	7:0
Field:	4 / 5	BCDR	Toggle	DT0	DT1	Byte 1			Byte N

Byte 1

Data Packet Type:

4 = Data Packet type

5 = Back Channel Packet Type

BCDR: This is a 1-bit field used to request back channel data. Setting this bit indicates to the bridge that the HID is listening for data following the transaction.

Data Toggle Bit: This is a 1-bit field that is toggled for each new data packet. It is used to distinguish between new and retransmitted packets.

Data Device Type 0/1: This is a 2-bit field that specifies a vendor-defined device type. This allows the bridge to determine HID type. The two bits are swapped to be backward compatible. In most cases the data device type is the same as the device type in the bind request and connect request. However, they may differ when one device tries to simulate the other device, for instance, the keyboard simulates the mouse.

DT0=0, DT1= 0 Presenter (0x0)

DT0=1, DT1= 0 Undefined (0x1)

DT0=0, DT1= 1 Keyboard (0x2)

DT0=1, DT1= 1 Mouse (0x3)

Byte 2-N

Data Byte 0-N - This is byte-aligned application data.

A.3.10   Bind and Reconnect Timing

When the bind button on the bridge is pressed, the bridge goes into bind mode. In bind mode, the bridge uses the bind ID to communicate with any HID that want to bind to the system (see "Bind Mode" on page 97 for more information on the bind ID). The bridge enables its receiver and 'listens' for any bind request packets from the HID, starting from channel 0. The bridge listens for approximately 320 ms on the channel, and if there is no bind request packet, it moves to the next channel in the bind channel subset (the bind channel subset consists of channels 0, 6, 12, 18, 24 … 78). It takes the bridge approximately 4.16 seconds to sequentially 'listen' on all 13 channels of the bind channel subset. The bridge repeats the process for up to five times before it times out and exits bind mode (time out is approximately 21 seconds). If it receives a valid bind request packet, it immediately responds to the request with a bind response packet and exit the bind mode. When the bind button on the HID is pressed, the HID goes into bind mode. While in bind mode, the HID also uses the bind ID to communicate with the bridge. The HID sends a bind request packet and listens for an AutoACK packet. If the HID does not receive the AutoACK, it moves to the next channel in the bind channel subset and repeats the bind request packet. It takes the HID approximately 23.4 ms to sequentially hop through all 13 channels of the bind channel subset, and the HID repeats the process for up to 1000 times before it times out. Because the bind buttons of the bridge and HID may be pressed at different times, the HID and the bridge can be on very different channels when the two are in bind mode. However, because the HID 'hops' very quickly on all bind channels while the bridge stays relatively long on a channel, the bridge and HID have multiple opportunities of being on the same channel. As a result, binding normally completes very quickly as soon as the bridge and the HID are both in bind mode (at 1.8 ms/ channel 'hopping' frequency of the HID and the bridge's 320 ms/channel, the two 'meet' on the same channel at least 13 times in any 320 ms period).

Figure A-15. Bind Timing Diagram
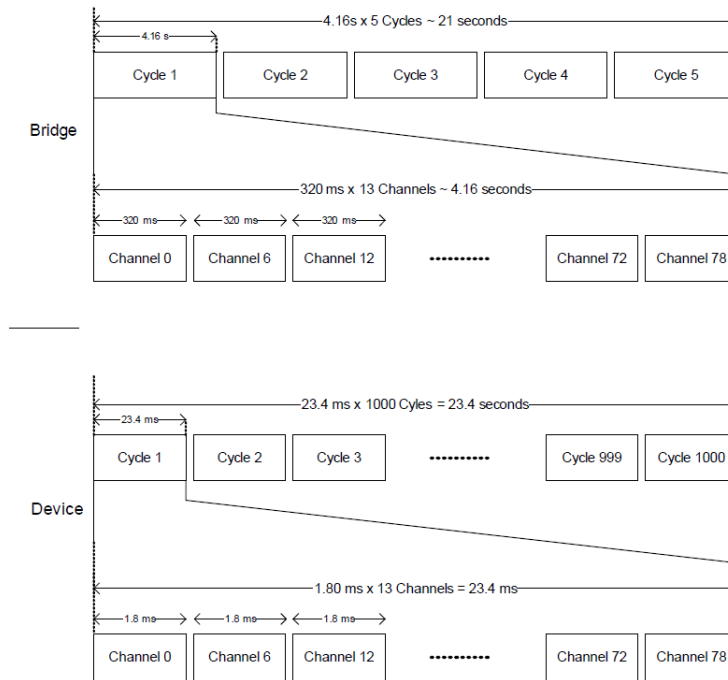
The bridge uses the RSSI to determine the noise level on the channel. If the channel has become noisy, the bridge moves to ping mode to find a quieter channel in its channel subset. When the HID loses connection with the bridge, it moves to reconnect mode to find the bridge. The HID sends a connect request packet and listens for an AutoACK packet. If the HID receives the AutoACK, it immediately enables its receiver and listens for the connect response packet from the bridge. If the HID does not receive the AutoACK it selects the next channel using the channel selection algorithm and repeats this procedure. As shown in the reconnect timing diagram below, the reconnect attempt takes approximately 1.76 ms/channel. The HID moves through its channel subset up to 19 times before it times out and exits reconnect mode. The keyboard tries to send the data for up to five seconds, and the mouse tries for two seconds, causing the HID to re-enter reconnect mode multiple times if necessary.

Figure A-16. Reconnect Timing Diagram
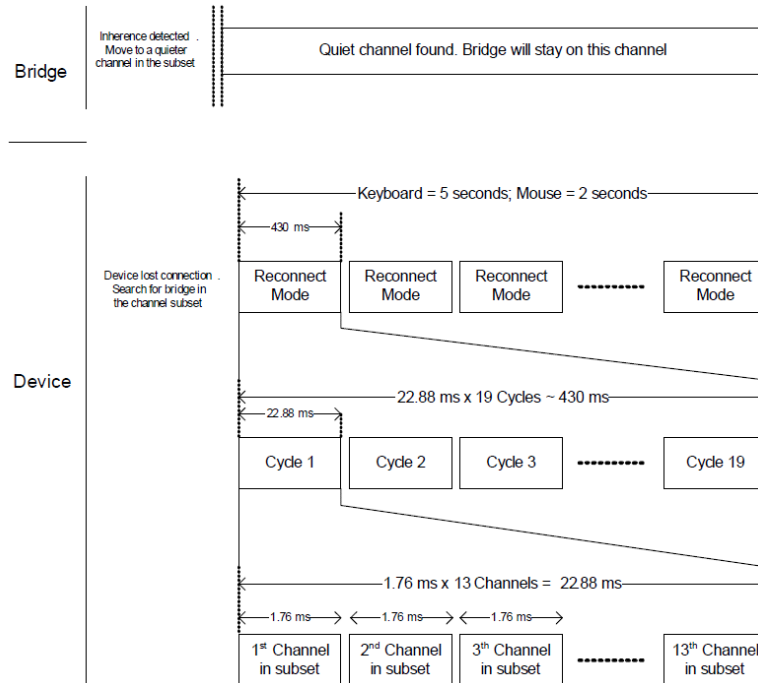
A.3.11   Signature Byte

The PRoC LP RDK uses the Signature byte to determine if the HID has been bound to any bridge before.

If the HID has never bound to a bridge, the nonvolatile memory used to store the signature and the bridge's MID data remains in its default value. When the HID has bound to a bridge, the Signature byte is set to 0x90 and the bridge's MID is also stored.

At power up, the HID reads the Signature and the MID bytes to determine its next action. If the Signature byte is 0x90, the HID uses the retrieved MID to calculate the networkID and moves to Reconnect mode. If the Signature byte is not 0x90, the HID goes to sleep, waiting for the user to initiate the bind process.

A.3.12   Encryption

WirelessUSB PRoC LP RDK supports Tiny Encryption Algorithm (TEA) and Advanced Encryption Standard (AES) 128 to encrypt application data. Data packets may be encrypted for privacy. All encrypted data packets must have a payload of 8 or 16 bytes depending on the method chosen; this is the minimum block size for the encryption algorithm.

A.3.12.1   TEA Encryption

Some of the features of TEA are:

■ 128-bit encryption key

■ 8-byte block size

■ Minimal RAM requirements

■ Small code size

■ Highly resistant to differential crypt analysis

To use the TEA algorithm both the bridge and HIDs must possess the data encryption key. The bridge is responsible for creating the key, which is then shared with the HIDs. There are a variety of possible methods to share the key between the two devices. The key may be exchanged over the WirelessUSB link using the encryption key request and encryption key response packets.

TEA Key Management over WirelessUSB

After binding and connecting to the bridge, the HID transmits an encryption key request packet and listens for an AutoACK followed by an encryption key response packet that contains the first half of the data encryption key. The HID then uses the key encryption key (calculated from the bridge and the HID MIDs) to decrypt the data encryption key. The HID repeats this process for the second half of the data encryption key and stores the key in flash. After receiving both halves of the data encryption key the HID may begin transmitting encrypted data to the bridge.

Figure A-17. TEA Encryption Key Management

CY4672 PRoC LP Reference Design Kit Guide, Doc. # 001-69289 Rev.*A
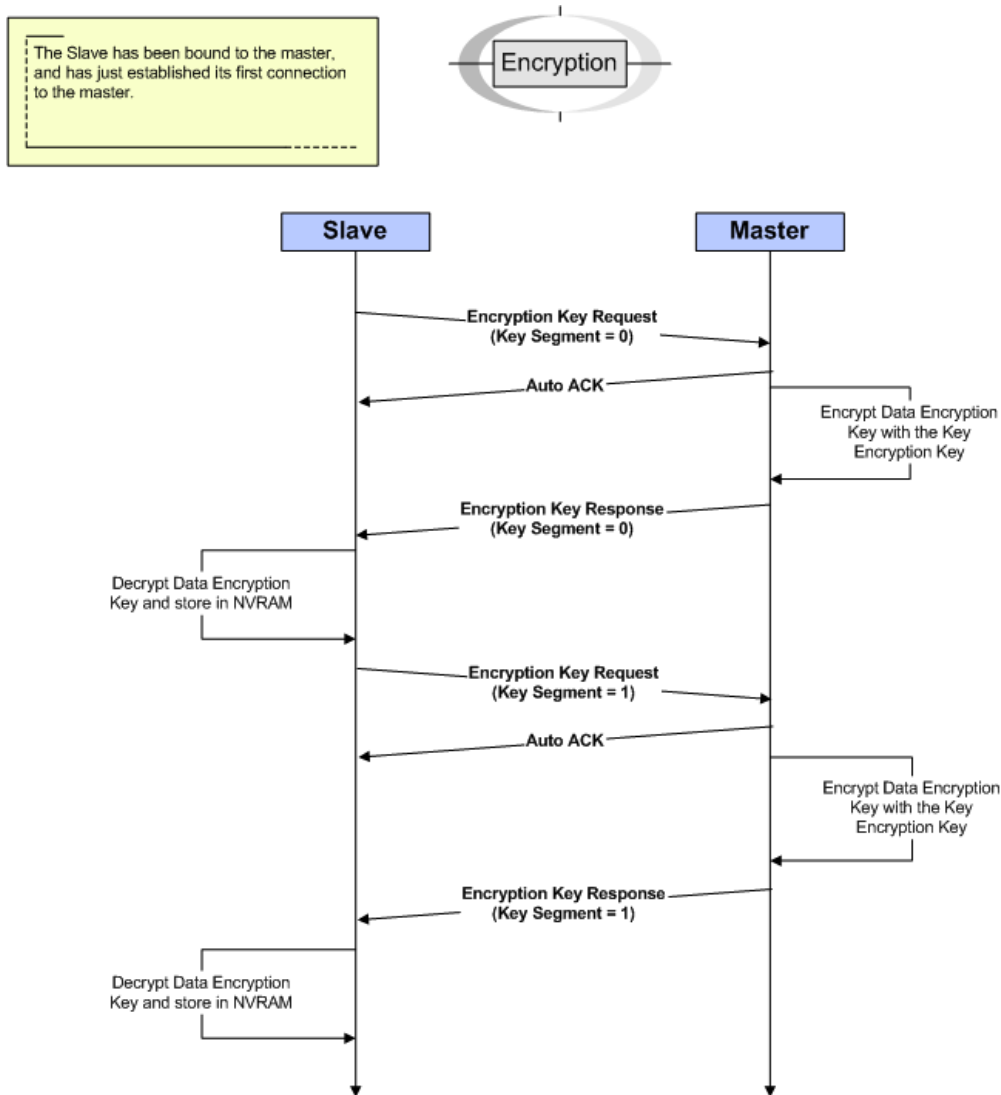
AES Encryption

AES_Encrypt requires the two variables tx_packet and AES_Key to be set before the call. Each contains a 16 byte (128 bit) value. At the completion of the function, tx_packet is encrypted in place and contains the cipher text. AES_Key is scheduled in-place during the encryption process, so multiple calls to AES_Encrypt each need to be proceeded with reloading of AES_Key.

AES_Decrypt is the same as the AES_Encrypt; rx_packet and AES_Key both need to be loaded before each call. However there is one small difference (because the decrypt key schedule is in reverse order), the decryption key is not the same as the encryption key. The key used by AES_Decrypt is the same as the key left in the AES_Key field after an execution of AES_Encrypt. This is the key after it has gone through ten rounds of Key scheduling.

AES Key Management

The encrypt key is stored on the keyboard and the decrypt key is stored on the bridge during compiling time. There is no dynamic encrypt key exchange in the running time.

Encryption and Power Consumption Trade Off

If the keyboard encryption is enabled, each key code is encrypted into an 8 byte key code (TEA) or 16 byte key code (AES). When a single key is pressed, a non-encrypted key down packet consists of a 16-bit Preamble + 2 bytes SOP + 1 byte packet header + 1 byte key code + 2 bytes CRC; an encrypted key down packet consists of the same overhead packets plus 8 or 16 byte key code instead of one byte key code. This results in an increase in the average power consumption when encryption is enabled.

A.4 Manufacturing Test Support, MTK

A.4.1 Introduction

The Manufacturing Test Kit (MTK) provides production line test support in addition to providing FCC certification tests. This section provides a description of the Tester serial protocol, the RF protocol between the MTK Tester and the MTK Device-Under-Test (DUT) and a brief description of porting the MTK DUT code to different platforms.

Refer to the Manufacturing Test Kit User's Guide for instructions on operating the MTK Tester.

A.4.2 MTK Block Diagram

Figure A-18. Block Diagram
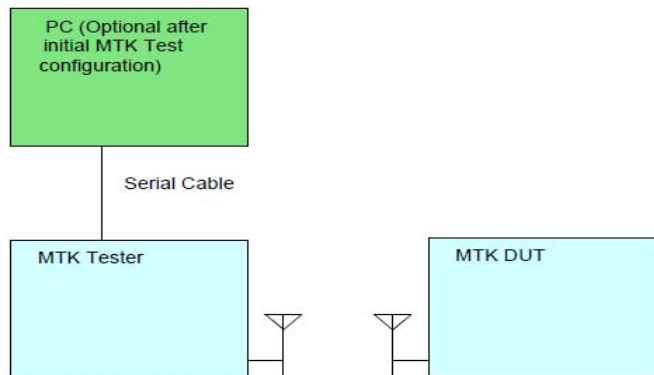
A.4.3   MTK Serial Protocol

The MTK Tester implements a text-based protocol over an RS232 serial port to provide both a configurable standard test and script-based testing.

All commands listed under the standard test set a configuration value that is stored in nonvolatile storage. All remaining serial commands only affect the current setting and are not stored (reset across power cycles). Commands are not case sensitive. All commands are of the form <command> space <command parameter>. For example 'TC 20'. Table A-1 describes the serial port protocol in the PC to tester direction.

Table A-1. Serial Command Protocol

Command		Command Description
ST		START STANDARD TEST
Standard Test Parameters	CL <power level>	CONFIGURE POWER LEVEL (0-7)
	PN <PN code>	CONFIGURE PN CODE INDEX (0-7)
	TT <tx error threshold>	CONFIGURE TX ERROR THRESHOLD (0-65535) units of bit errors
	RT <rx error threshold>	CONFIGURE RX ERROR THRESHOLD (0-65535) units of bit errors
	C1 <channel>	CONFIGURE CHANNEL (0-77)
	C2 <channel>	CONFIGURE SECOND CHANNEL (0-77)
	C3 <channel>	CONFIGURE THIRD CHANNEL (0-77)
	CB <# of bytes>	CONFIGURE NUMBER OF BYTES/PACKET PAYLOAD (0-15)
	CP <# of packets>	CONFIGURE NUMBER OF PACKETS (0-255)
TC <time>		TRANSMIT CARRIER (0-255)
TR <time>		TRANSMIT RANDOM (0-255)
SC <channel> <PN code> <power level> <correlator threshold>		SET COMMUNICATION (0-77) (0-7) (0-7) (0-16) Note: The device transmits on <channel> + 2. For example, 2=2.402 GHz
PD <packet data>		SET PACKET DATA (ASCII representation of hexadecimal numbers without any prefix, i.e. 5A 34 CB)
CA <crystal adjust>		SET CRYSTAL FREQUENCY ADJUST VALUE (0-63)
RE		RESTORE NVRAM DEFAULTS
CS		SHOW CURRENT CONFIGURATION
HE		SHOW HELP MENU

Every serial command issued by the PC is returned with a response when the command is complete. The valid responses are shown in Table A-2.

Table A-2. Serial Response Protocol

REPORT	REPORT DESCRIPTION
OK	COMMAND COMPLETE
CE	COMMAND ERROR
TE <transmit error count>	TX ERROR COUNT (units of bit errors)
RE <receive error count>	RX ERROR COUNT (units of bit errors)

All serial commands must end in either a carriage return or carriage return and line feed. All responses end with a carriage return and linefeed.

The serial port settings for the MTK Tester are shown in Table A-3. Neither software nor hardware handshake is supported.

Table A-3. Serial Port Parameter Settings

Serial Port Parameter	Setting
Baud Rate	9600
Parity	None
Number of Data Bits	8
Number of Stop Bits	1

A.4.4   MTK RF Protocol

Command packets received by the Device-Under-Test (DUT) are 'echoed' with the addition of an added byte that contains the count of invalid bits for the received packet. Extra bytes in packets that are larger than what the DUT can support are ignored. Commands other than 'Echo Packet' are only 'echoed' and executed if the number of invalid bits are zero.

The RF command packets exchanged between the MTK Tester and the MTK DUT contain two bytes. The first byte contains the command type and the subsequent byte(s) contain the parameter values as shown in Table A-4.

Table A-4. RF Commands

Description	Command	Parameter
Echo Packet	0x00	N/A
Set New Configuration	0x61	Channel (0-77) PN code index (0-7) PA (0-7) Correlator Threshold (0-16)
Transmit carrier	0x66	Time in seconds (0-255) Note: A zero runs the test continuously until a reset.
Transmit random pattern	0xA3	Time in seconds (0-255) Note: A zero runs the test continuously until a reset.

The 'Transmit carrier' and 'Transmit random pattern' test mode can be conditionally compiled with the define MFG_TX_MODES.

A.4.5   MTK DUT Source Code Porting

The RDK keyboard, bridge and mouse use the C source files mfgtest.c and mfgtest.h. Select the appropriate source files for the target platform as a starting point. Make code changes as necessary to work in your environment.

A.4.6   Accessing MTK in the DUT

Mouse: Apply a jumper across the ISSP header pins 4/5 and install the batteries.

Keyboard: Same as mouse.

Bridge: Press the button while plugging it into the USB port. The LEDs should blink.

A.5 Regulatory Testing Results

A.5.1 Introduction

The LP mouse is tested in a certified lab and meets FCC part 15, Subpart B, Title 47 CFR -Unintentional radiators, FCC Part 15 Subpart C -Intentional radiators, and Industry Canada RSS-Gen. The following table outlines the results of the testing.

Testing for the keyboard and bridge is expected in the near future.

Table A-5. EMC Test Results

Test Parameter	FCC Limit	Measured Value	Margin
Spurious Radiated Emissions	54 dBuV/m(Av)	50.5 dBuV/m	-3.5 dB
Spurious Conducted Emissions	-20dBc	-31.9 dBc	11.9 dB
Power Spectral Density	8 cBm/3 kHz	-9.0 dBm/3 kHz	17.0 dB
Output Power	30 dBm	2.3 mW	26.0 dB
Occupied Bandwidth	>500 kHz	960 kHz	460 kHz
Conducted Band Edge Compliance	20 dB below fundamental	-27.0 dBc	7.0 dB
Radiated Band Edge Compliance at 2482 Mhz	54 dBuV/m(Av)	51.7 dBuV/m	-2.3 dB
	Industry Canada limit		
Receiver Radiated Emissions	46 dBuV/m(QP)	35.4 dBuV/m	-10.6 dB

A.6 Power Considerations

A.6.1 RDK Keyboard

A.6.1.1 Usage Model

The following usage model are considered for the RDK keyboard.

- 4 hours a day of 6 keystrokes per second, 5 days a week
- 24 hours a day with no activity, 2 days a week
- A packet is transmitted on both key-up and key-down events
- A 'keep alive' is transmitted for each key-down event

A.6.1.2 Current Measurements

Per the keyboard usage model, there are 6 keystrokes per second in the active state, and every keystroke includes one 'down key' packet, one 'up key' packet and one 'keep alive' packet. The test mode firmware only sends out one 'down key' packet and one 'up key' packet for each keystroke.

Therefore, we need to set the typing rate to 8 keystrokes per second in test mode to consume the equivalent power of the usage model. It is accomplished by changing the KEYBOARD_TEST_MODE_PERIOD define in the config.h file to 50.

The following is the results of RDK keyboard current measurement:

Table A-6. Keyboard Current Measurement

Operation Mode	Icc (mA) with Supply Voltage = 2.5 V	Icc (mA) with Supply Voltage = 2.8 V	Average Icc (mA)
Active mode–Place the keyboard in test mode "the quick brown fox …" and set the keystroke rate to 8 character per second.	0.96	0.82	0.89
Idle mode–A keyboard is in its normal power on state and connected to the bridge with no keys pressed.	0.040	0.040	0.040
Not connected mode–Type the keyboard.	20.3	16.9	18.6
Not connected mode–No typing.	Transition to idle mode.	Transition to idle mode.	

A.6.1.3   Battery Life Calculations

The following table shows the times spent in each state by the RDK keyboard usage model. By substituting the current measurements in , the overall average Icc for RDK keyboard can be calculated.

Table A-7. RDK keyboard Power Consumption

	Mode	Hrs/day	Days	Average Icc (mA)	Charge (mAh)
Week day	Active	4	5	0.89	17.8
	Idle	20	5	0.04	4
Weekend	Idle	24	2	0.04	1.92
Charge Per Week (mAh)					23.72
Overall Average Icc (mA)					0.141

The RDK keyboard uses two AA cells and enables the PMU function. Therefore, it is able to access approximately 2850-mAh battery capacity, which yields a battery life estimate of 829 days or 27 months.

A.6.2   RDK Mouse

A.6.2.1   Usage Model

The following usage model is considered for the RDK mouse.

- 1 hour a day with the 3030/3040 sensor in 'active' state
- 2 hours a day with the 3030/3040 sensor in 'rest1' state
- 2 hours a day with the 3030/3040 sensor in 'rest2' state
- 19 hours a day with the 3030/3040 sensor in 'rest3' state
- 5 days a week as above, 2 days a week 24 hours in 'rest3' state

A.6.2.2 Current Measurements

The following is the results of RDK mouse current measurement:

Table A-8. Mouse Current Measurement

Operation Mode	Icc (mA) with Supply Voltage = 2.5V	Icc (mA) with Supply Voltage = 2.8V	Average Icc (mA)
Active mode–Move the mouse in a circle on white paper.	8	6.9	7.5
Rest1 mode–Allow the mouse to sit idle for 1 second after being in the active state.	2.03	1.81	1.92
Rest2 mode–Allow the mouse to sit idle for 10 seconds after being in the active state.	0.16	0.15	0.16
Rest3 mode–Allow the mouse to sit idle for 10 minutes after being in the active state.	0.071	0.070	0.071
Not connected mode–Briefcase Mode.	0.05	0.04	0.05

A.6.2.3 Battery Life Calculations

The following table shows the times spent in each state by the RDK mouse usage model. By substituting the current measurements in "Current Measurements" on page 114, the overall average Icc for RDK mouse can be calculated.

Table A-9. RDK Mouse Power Consumption

	Mode	Hrs/day	Days	Average Icc (mA)	Charge (mAh)
Week day	Active	1	5	7.5	37.5
	Rest1	2	5	1.92	19.2
	Rest2	2	5	0.16	1.6
	Rest3	19	5	0.07	6.65
Weekend	Rest3	24	2	0.07	3.36
Charge Per Week (mAh)					68.3
Overall Average Icc (mA)					0.407

The RDK mouse uses two AA cells and enables the PMU function. Therefore, it is able to access approximately 2850-mAh battery capacity, which yields a battery life estimate of 338 days or 11 months.

A.7 Software Guide

A.7.1 Introduction

This section describes the software source code modules used to communicate with the PRoC LP bridge HID device to obtain the current radio parameters for the attached WirelessUSB LP devices. It does not cover the details of the Microsoft Foundation Class (MFC) Library or the HID Library that contains standard system-supplied routines that user-mode applications use to communicate with USB devices that comply with the USB HID Standard. Refer to the Microsoft Visual C++ documentation for more on MFC and HID Class concepts, in addition to the Device Class Definition for Human Interface Devices (HID) defined by the USB Implementers Forum, Inc. (http://www.usb.org/developers/hidpage).

A.7.2 Software Code Modules

There are three main modules contained in the WirelessUSB software:

■ USB HID API module-generic class interface to HID class compliant devices

■ System Tray module-generic class to create and control an icon on the system tray

■ WirelessUSB System Tray Application module-main system tray application module

The following sections describe the software module contents.

A.7.2.1 USB HID API module

The USB HID API module defines two classes, CHidDevice and CHidManager. The CHidDevice class is the primary interface to a HID device, while the CHidManager class keeps track of the arrival and removal of HID devices, along with notification to the application of such events. The building blocks for the USB HID API module is derived from the HCLIENT sample code provided in the Windows DDK. This module is designed to provide a generic interface to any HID class compliant device and is not expected to require any modification, however all source code is provided for reference.

CHidDevice Class Methods

Table A-10. CHidDeviceClass Methods

Method	Type	Description
OpenHidDevice()	Public	This method sets appropriate access rights, attempts to open a handle to the HID device, obtains the top collection data, and makes a call to setup input, output, and feature data buffers.
CloseHidDevice()	Public	This method closes the HID device handle, un-registers the HID device notification, and frees pre-parsed data and data/report buffers.
RegisterHidDevice()	Public	This method registers the HID device handle for event notification.
IsOpen()	Public	This method is used to report if a valid handle is open to the HID device.
IsOpenForRead()	Public	This method is used to report if the handle open to the HID device allows for read access.
IsOpenForWrite()	Public	This method is used to report if the handle open to the HID device allows for write access.
IsOpenOverlapped()	Public	This method is used to report if the handle open to the HID device allows for overlapped I/O.
IsOpenExclusive()	Public	This method is used to report if the handle open to the HID device is setup for exclusive access.
GetHandle()	Public	This method returns the handle to the HID device.
Read()	Public	This method reads an input report from the HID device, performs a validity check, and unpacks the report data.
Write()	Public	This method is used for every report ID, packs a report buffer and writes the report data to the HID device.
GetFeature()	Public	This method obtains the feature report from each report ID exposed by the HID device.
SetFeature()	Public	This method sends a feature report for each report ID exposed by the HID device.
UnpackReport()	Public	This method scans through the HID report and if it can, fills in any data in the structures.
PackReport()	Public	This method packages the HID report based on the data in the structures.
GetManufacturerString()	Public	This method obtains the USB manufacturer string from the HID device.
GetProductString()	Public	This method obtains the USB product string from the HID device.
GetSerialNumberString()	Public	This method obtains the USB serial number string from the HID device.
RegGetValue()	Public	This method attempts to get a registry value from the registry key where the device-specific configuration information for the HID device is stored.
RegSetValue()	Public	This method attempts to set a registry value in the registry key where the device-specific configuration information for the HID device is stored.
SetupHidDevice()	Protected	This method sets up HID Input, Output and Feature data buffers used to simplify communication with HID devices.
ValidateHidDevice()	Protected	This method simply returns TRUE, it is expected that this routine will be overridden by the application where the actual validation will be handled.

CHidManager Class Methods

Table A-11. CHidManagerClass Methods

Method	Type	Description
Create()	Public	This method creates an invisible window and uses the returned window handle to register for HID device notification events, then it creates a list of existing HID devices that will be maintained by the HID manager.
IsHidDevicePresent()	Public	This method attempts to open a handle to the HID device to determine if it is present (or not) and returns the result.
RefreshHidDevices()	Public	This method validates that all HID devices in the list are still present, removes those from the list that are currently not present, scans the list of all existing HID devices present, and then attempts to add the existing HID devices to the list.
GetDeviceCount()	Public	This method returns the number of connected devices.
GetFirstHidDevice()	Public	This method returns a pointer to the first HID device in the list.
GetNextHidDevice()	Public	This method returns a pointer to the next HID device in the list.
GetCurrentHidDevice()	Public	This method returns a pointer to the current HID device in the list.
GetHidDeviceWithPath()	Public	This method scans the current list of HID devices and returns a pointer to the HID device that matches the device path provided.
GetHidDevice WithHandle()	Public	This method scans the current list of HID devices and returns a pointer to the HID device that matches the device handle provided.
HidDeviceAlreadyExists()	Public	This method determines if the HID device already exists in the list
AddHidDevice()	Public	This method checks if the provided HID device already exists, and if not, adds the new HID device to the end of the list, increments the HID device counter, and call the HID callback function to indicate a new HID device was added.
RemoveHidDevice()	Public	This method closes the outstanding handle to the HID device, calls the HID callback function to indicate that the HID device is being removed, removes the HID device from the list, and deletes the HID device.
RemoveAllHidDevices()	Public	This method scans though all HID devices in the list and removes them.
CreateUniqueDeviceID()	Public	This method attempts to create and maintain a unique ID for the associated HID device.
FreeUniqueDeviceID()	Public	This method frees the specified unique ID.
NewHidDevice()	Protected	This method allocates memory for a new HID device structure
DeleteHidDevice()	Protected	This method deletes previously allocated memory for an existing HID device structure.
RegisterHidNotification()	Protected	This method registers for notification of events for all HID devices and calls the HID callback function to indicate registration was completed.
HidDeviceArrival()	Protected	This method makes sure the HID device does not already exist in the list, and then creates a new HID device, opens a handle to the device, adds the new HID device to the list, and registers event notification for this new HID device.
HidDeviceQuery Removal()	Protected	This method readies the HID device for removal by making sure the handle is closed.
HidDeviceRemoval()	Protected	This method removes the HID device.

A.7.2.2 System Tray Module

The System Tray module defines the CCySysTray class, which provides the interface to the system tray for the application. This module is not expected to require any modification; however, all source code is provided for reference.

CCySysTray Class Methods

Table A-12. CCySysTray Methods

Method	Type	Description
Create()	Public	This method creates an invisible window and sets up the system tray icon (if needed).
SetIcon()	Public	This method sets (or replaces) the icon displayed on the system tray.
RemoveIcon()	Public	This method removes the icon from the system tray.
SetToolTip()	Public	This method sets the tool tip to be displayed on the system tray.
SetMenuItem()	Public	This method sets the default menu item executed when the icon is double-clicked on the system tray.
IsHidden()	Public	This method is used to determine if the system tray icon is hidden.
ShowBalloonTip()	Public	This method displays a balloon style tip message (only supported on W2K or higher).
OnTrayNotification()	Public	This method processes events that occur to the icon in the system tray.
OnTaskbarCreated()	Protected	This method is called when the system tray is being restarted (for example, if Explorer crashes).
WindowProc()	Protected	This method overrides the default WindowProc to call OnTrayNotification for messages targeting the system tray icon or OnTaskbarCreated if the system tray is being restarted.

A.7.2.3 WirelessUSB System Tray Application Module

The WirelessUSB System Tray module is the main system tray application. This module places the icon on the system tray bar, manages the HID devices, displays pop up messages, and controls the WirelessUSB Status Property Sheet. Additionally, via command-line parameters, this module can enable and disable the system tray application from running at startup.

CWirelessUSBTrayApp Class Methods

The CWirelessUSBTrayApp class performs application initialization and removal; in addition, it parses command-line parameters used to enable or disable the system tray application from being run at startup.

Table A-13. CWirelessUSBTrayApp Methods

Method	Type	Description
InitInstance()	Public	This method performs basic initialization and checks for any command-line parameters. If command-line parameters are found, it takes the appropriate action and ends the application; if no command-line parameters are found it checks to make sure the application is not currently running and, if not, proceeds to run the system tray application.
ExitInstance()	Public	This method performs some standard cleanup before the application ends.
RegisterAutoLoader()	Protected	This method registers the application (itself) to always be run at startup and optionally launches itself as well.
UnregisterAudoLoader()	Protected	This method un-registers the application (itself) to prevent running at startup and optionally ends itself from running.
AutoLoadExe()	Protected	This method launches the specified EXE application.

CMainFrame Class Methods

The CMainFrame class is the Visual C++ generated file that is a derived frame-window class for the system tray application's main frame window. This class is modified to also perform the timer based polling of the PRoC LP bridge HID device to obtain the radio parameters and display any appropriate pop up messages. Additionally, this class also processes the command message to create the WirelessUSB Status Property Sheet.

Table A-14. CMainFrame Methods

Method	Type	Description
OnCreate()	Public	This method is called when a new window is created for this frame. It sets up the HID Notification callback and device status property sheet, initializes the HID manager, creates the system tray icon, sets up the menu and tool tips. If any HID devices are present, it displays the icon on the system tray and makes a call to start the timer.
HIDNotification()	Public	This method processes notifications of when an HID device is added or removed from the list. It adds or removes property pages to the wireless status page and adds or removes the icon from the system tray when the first or last HID devices is added or removed.
OnStartTimer()	Public	This method starts the timer based on the hard-coded poll timer (currently set to once every 5 seconds).
OnStopTimer()	Public	This method stops the timer.
OnTimer()	Public	This method is the timer routine that is called when the timer expires. It loops through all the HID devices in the list and updates their status values then restarts the timer; also, it occasionally requests an update in the battery level, currently set to once every hour.
OnDestroy()	Public	This method is called when the frame window is destroyed. It stops the timer, removes the property sheet (if displayed), and removes the icon from the system tray.
OnAppWireless USBStatus()	Public	This method displays the wireless status page, if it is not already displayed.

CWirelessUSBStatusPropertyPage Class Methods

The CWirelessUSBStatusPropertyPage class is the Visual C++ generated file that implements the WirelessUSB Device Status Property Page, a unique property page is created for each WirelessUSB device enumerated.

Table A-15. CWirelessUSBStatusPropertyPage Methods

Method	Type	Description
OnInitDialog()	Public	This method initializes the wireless status page, reads the current value of the Disable Warning Message check box from the registry, and makes a call to start the timer.
OnDestroy()	Public	This method removes the wireless status page and stops the timer.
OnStartTimer()	Public	This method starts the timer for the wireless status page based on the hard-coded poll timer (currently set to once ever 500 ms).
OnStopTimer()	Public	This method stops the timer for the wireless status page.
CommaStr()	Public	This method takes a numeric value and returns a CString representation of the number with commas added.
OnTimer()	Public	This method updates the HID device values displayed on the status page then restarts the timer; also, it occasionally requests an update in the battery level, currently set to once every 5 seconds while the status page is displayed.
OnBnClickedWireless USBDisableWarning Message()	Public	This method is called when the Disable Warning Messages check box is changed. Base on the check box value, it either disables or enables battery and signal strength warning messages for the specific HID device. The updated value is then stored in the device-specific configuration information for the HID device.

CWirelessUSBStatusPropertySheet Class Methods

The CWirelessUSBStatusPropertySheet class is the Visual C++ generated file that implements the WirelessUSB Status Property Sheet, which generates a unique WirelessUSB Device Status Property Page for each WirelessUSB device enumerated.

Table A-16. CWirelessUSBStatusPropertySheet Methods

Method	Type	Description
OnInitDialog()	Public	This method initializes the wireless status property sheet and adds a property page for each HID device in the list.
OnBnClickedClose()	Public	This method ends the dialog box if the user selects the Close button.

CHidTrayDevice Class Methods

The CHidTrayDevice class is derived from the CHidDevice class and is the class used to interface with WirelessUSB devices.

Table A-17. CHidTrayDevice Methods

Method	Type	Description
RequestNewUsageValues()	Public	This method sets up and issues a Set Feature request to the HID device, which now simply requests the wireless device to provide an update of its battery level the next time it communicates with the USB bridge.
UpdateUsageValues()	Public	This method retrieves the latest usage values from the USB bridge, which includes wireless channel, wireless PN code, last reported battery level, and signal strength.
UpdateDeviceInfo()	Public	This method makes a call to update the HID device usage values and displays a warning message (if enabled).
GetUsageIDValue()	Public	This method extracts the value of the provided Usage ID from the feature data.
VerifyHidDevice()	Public	This method is called to verify that the HID device is one that should be added to the list; right now this is done by making sure the **usage page** reported is WIRELESSUSB_USAGEPAGE and the **usage** reported is either WIRELESSUSB_USAGE_KEYBOARD or WIRELESSUSB_USAGE_MOUSE.

CHidTrayManager Class Methods

The CHidTrayManager class is derived from the CHidManager class and is used to manage WirelessUSB devices.

Table A-18. CHidTrayManager Methods

Method	Type	Description
NewHidDevice()	Protected	This method creates a new HID device, initializes it, and adds it to the list of existing HID devices.
DeleteHidDevice()	Protected	This method removes the HID device from the list and deletes the HID device.

A.8 Bill of Materials (BOM)

A.8.1 Bridge

No.	Qty.	Reference	Description	Manufacturer	Mfr Part Number
1	1	ANT1	2.5GHZ H-STUB WIGGLE ANTENNA FOR 32MIL PCB	NA	NA
2	1	C1	CAP 15PF 50 V CERAMIC NPO 0402	Panasonic	ECJ-0EC1H150J
3	1	C3	CAP 2.0 PF 50 V CERAMIC NPO 0402	Kemet	C0402C209C5GACTU
4	1	C4	CAP 1.5PF 50 V CERAMIC NPO 0402 SMD	Panasonic	ECJ-0EC1H1R5C
5	1	C5	CAP CER .47UF 6.3 V X5R 0402	Murata	GRM155R60J474KE19D
6	9	C6,C7,C8,C9,C10,C11,C12,C15,C16	CAP CERM .047UF 10% 16 V X5R 0402	AVX	0402YD473KAT2A
7	1	C13	CAP CER 4.7UF 16 V X5R 0805	Panasonic - ECG	ECJ-2FB1C475K
8	1	C14	CAP CER 2.2UF 10 V 10% X7R 0805	Murata Electronics North America	GRM21BR71A225KA01L
9	1	D1	LED GREEN/RED BICOLOR 1210 SMD	LITEON	LTST-C155KGJRKT
10	1	J1	CONN USB PLUG TYPE A PCB SMT	ACON	UAR72-4N5J10
11	1	L1	INDUCTOR 22NH 2% FIXED 0603 SMD	Panasonic - ECG	ELJ-RE22NGFA
12	1	L2	INDUCTOR 1.8NH +-.3NH FIXED 0402 SMD	Panasonic - ECG	ELJ-RF1N8DFB
13	1	R1	RES ZERO OHM 1/16W 0402 SMD	Panasonic - ECG	ERJ-2GE0R00X
14	1	R2	RES CHIP 620 OHM 1/16W 5% 0402 SMD	Panasonic - ECG	ERJ-2GEJ621X
15	1	S1	SWITCH LT 3.5MMX2.9MM 160GF SMD	Panasonic - ECG	EVQ-P7J01K
16	3	TP2,TP3,TP4	TEST PAD 40 SMT	NA	NA
17	7	TV1,TV5,TV6,TV7,TV8,TV9,TV10	TEST VIA 36 HOLE 20 PLATED	NONE	
18	1	U1	IC, LP-PRoC 5CR 2.4 GHz SIP QFN-40	Cypress Semiconductor	Rev 6381B/6936A5
19	1	Y1	CRYSTAL 12.00MHZ HC49 SMD	eCERA	GF-1200008
20	1	PCB	PRINTED CIRCUIT BOARD	Cypress Semiconductor	
21	1	LABEL1	Serial Number		
22	1	LABEL2	PCA #		121-34801 **
No load components - Do not install					
23	2		NO LOAD		

A.8.2 Mouse

No.	Qty.	Reference	Description	Manufacturer	Mfr Part Number
1	1	C1	CAP 15PF 50 V CERAMIC NPO 0402	Panasonic	ECJ-0EC1H150J
2	1	C3	CAP 2.0 PF 50 V CERAMIC NPO 0402	Kemet	C0402C209C5GACTU
3	1	C4	CAP 1.5PF 50 V CERAMIC NPO 0402 SMD	Panasonic	ECJ-0EC1H1R5C
4	2	C17,C5	CAP CER .47UF 6.3 V X5R 0402	Murata	GRM155R60J474KE19D
5	2	C12,C7	CAP CERAMIC 10UF 6.3 V X5R 0805	Kemet	C0805C106K9PACTU
6	1	C8	CAP 1 uF 6.3 V CERAMIC X5R 0402	Panasonic	ECJ-0EB0J105M
7	7	C9,C10,C11,C13,C14,C15,C16	CAP CERM .047UF 10% 16 V X5R 0402	AVX	0402YD473KAT2A
8	1	C18	CAP 100UF 10 V ELECT FC RADIAL	Panasonic - ECG	EEU-FC1A101S
9	3	C23,C24,C25	CAP 10000PF 16 V CERAMIC 0402 SMD	Panasonic - ECG	ECJ-0EB1C103K
10	3	C26,C27,C28	CAP CERAMIC 1.0UF 10 V X5R 0603	Kemet	C0603C105K8PACTU
11	1	D1	DIODE SCHOTTKY 20 V 1A SMA	Taiwan Semiconductor	SS12
12	1	J10	CONN HDR BRKWAY 5POS STR AU PCB	AMP Division of TYCO	103185-5
13	1	J11	HEADER 1 POS 0.230 HT MODII .100CL	AMP/Tyco	103185-1
14	1	L1	INDUCTOR 22NH 2% FIXED 0603 SMD	Panasonic - ECG	ELJ-RE22NGFA
15	1	L2	INDUCTOR 1.8NH +-.3NH FIXED 0402 SMD	Panasonic - ECG	ELJ-RF1N8DFB
16	1	L3	COIL 10UH 1.23A UNSHIELDED SMD	Sumida	CDH53NP-100LC
17	1	R1	RES 47 OHM 1/16W 5% 0402 SMD	Panasonic - ECG	ERJ-2GEJ470X
18	3	R6,R7,R8	RES CHIP 5.11 OHM 1/16W 1% 0603 SMD	Yageo America	9C06031A5R11FGHFT
19	1	R9	RES 1.00 OHM 1/8W 1% 0805 SMD	Yageo	9C08052A1R00FKHFT
20	1	U1	IC, PRoC LP 5CR 2.4 GHz SIP QFN-40	Cypress Semiconductor	
21	1	Y1	CRYSTAL 12.00MHZ HC49 SMD	eCERA	GF-1200008
22	1	PCB	PRINTED CIRCUIT BOARD	Cypress Semiconductor	
23	1	LABEL1	Serial Number		
24	1	LABEL2	PCA #		121-34700 *A
No load components - Do not install					
25	1	C6	CAP NO LOAD 1210	NA	NA
26	2	C22,C21	CAP NO LOAD 0603	NA	NA
27	3	R4,R5,R19	RES NO LOAD 0603 SMD	NA	NA
28	4	TP1,TP2,TP3,TP4	TEST POINT 43 HOLE 65 PLATED	NONE	
29	9	TV1,TV2,TV3,TV4,TV6,TV7,TV8,TV9,TV10	TEST VIA 40 HOLE 20 PLATED	NONE	

No.	Qty.	Reference	Description	Manufacturer	Mfr Part Number
30	1	ANT1	2.5GHZ H-STUB WIGGLE ANTENNA FOR 63MIL PCB	NA	NA
Do not install the following components - these parts will be installed at Cypress, Boise					
31	1	BH1	BATTERY HOLDER BTC MOUSE SKIN	Recycled BTC Mouse	M957U
32	2		BATTERY CLIPS POSITIVE	Recycled BTC Mouse	Unknown
33	2		BATTERY CLIPS NEGATIVE	Recycled BTC Mouse	Unknown
34	1	EN1	ENCODER CHICONY MOUSE	Recycled BTC Mouse	Unknown
35	1	SW1	SWITCH SLIDE MINI SPDT BTC ON/OFF	Recycled BTC Mouse	1101M2S3CQE2
36	1	S1	LT SWITCH SMD BTC BIND	Recycled BTC Mouse	BTC_PUSH
37	3	S2,S3,S4	SWITCH 6MM VERT LEFT MIDDLE RIGHT	Recycled BTC Mouse	Unknown
38	1	U2	IC, ULTRA LOW POWER MOUSE SENSOR DIP20	Avago Technologies	ADNS-3040
39	1		MOUSE SENSOR PLASTIC SPACER	Recycled BTC Mouse	Unknown
40	1	D2	LED 3MM RED DIFFUSED TH	Lite-On	LTL-4261N
41	1		LED CLIP	Recycled BTC Mouse	Unknown
42	1		LENS	Stuart Lens	5042-8121
43	1		Z-WHEEL	Recycled BTC Mouse	Unknown

A.8.3   Keyboard

No.	Qty.	Reference	Description	Manufacturer	Mfr Part Number
1	1	ANT1	2.5GHZ H-STUB WIGGLE ANTENNA FOR 63MIL PCB	NA	NA
2	1	BH1	BATTERY CLIPS 2AA CELL CHICONY KB	CHICONY	KBR0108
3	1	C1	CAP 15PF 50 V CERAMIC NPO 0402	Panasonic	ECJ-0EC1H150J
4	1	C3	CAP 2.0 PF 50 V CERAMIC NPO 0402	Kemet	C0402C209C5GACTU
5	1	C4	CAP 1.5PF 50 V CERAMIC NPO 0402 SMD	PANASONIC	ECJ-0EC1H1R5C
6	2	C5,C17	CAP CER .47UF 6.3 V X5R 0402	Murata	GRM155R60J474KE19D
7	2	C12,C7	CAP CERAMIC 10UF 6.3 V X5R 0805	Kemet	C0805C106K9PACTU
8	1	C8	CAP 1 uF 6.3 V CERAMIC X5R 0402	Panasonic	ECJ-0EB0J105M
9	6	C9,C10,C11,C13,C15,C16	CAP 0.047 uF 16 V CERAMIC X5R 0402	AVX	0402YD473KAT2A
10	1	C18	CAP 100UF 10 V ELECT FC	Panasonic - ECG	EEU-FC1A101S
11	2	C20,C19	CAP 10000PF 16 V CERAMIC 0402 SMD	Panasonic - ECG	ECJ-0EB1C103K
12	1	D1	DIODE SCHOTTKY 0.5A 40 V SOT23	DIODES INC	BAT400D-7-F
13	1	J1	PCB COPPER PADS	NONE	
14	1	J2	CONN HDR BRKWAY 5POS STR AU PCB	AMP Division of TYCO	103185-5
15	1	J3	HEADER 1 POS 0.230 HT MODII .100CL	AMP/Tyco	103185-1
16	1	L1	INDUCTOR 22NH 2% FIXED 0603 SMD	Panasonic - ECG	ELJ-RE22NGF2
17	1	L2	INDUCTOR 1.8NH +-.3NH FIXED 0402 SMD	Panasonic - ECG	ELJ-RF1N8DF

No.	Qty.	Reference	Description	Manufacturer	Mfr Part Number
18	1	L3	COIL 10UH 1100MA CHOKE 0805	Newark	30K5421
19	1	R2	RES 1.00 OHM 1/8W 1% 0805 SMD	Yageo	9C08052A1R00FKHFT
20	1	R3	RES 47 OHM 1/16W 5% 0402 SMD	Panasonic - ECG	ERJ-2GEJ470X
21	1	S1	LT SWITCH 6MM 100GF H=7MM TH	Panasonic - ECG	EVQ-PAC07K
22	1	U1	IC, LP 2.4 GHz RADIO SoC QFN-40	Cypress Semiconductor	
23	1	U2	IC WIRELESS EnCore II CONTROLLER SSOP48	Cypress Semiconductor	
24	1	Y1	CRYSTAL 12.00MHZ HC49 SMD	eCERA	GF-1200008
25	1	PCB	PRINTED CIRCUIT BOARD	Cypress Semiconductor	
26	1	LABEL1	Serial Number		
27	1	LABEL2	PCA #		121-26505 **
No load components - Do not install					
28	1	C6	CAP 47UF 6.3 V CERAMIC X5R 1210	Panasonic	ECJ-4YB0J476M
29	1	R2	RES CHIP 0.0 OHM 1/10W 5% 0805 SMD	Phycomp USA Inc	9C08052A0R00JLHFT
30	1	C7	CAP 0.047 uF 50 V CERAMIC X5R 0402	AVX	0402YD473KAT2A
31	1	J4	HEADER 3POS FRIC STRGHT MTA 100	AMP/Tyco	644456-3
32	1	R1	RES ZERO OHM 1/16W 5% 0603 SMD	Panasonic - ECG	ERJ-3GEY0R00V