# CY4632 Keyboard Firmware
# User's Guide

Cypress Semiconductor
3901 North First Street
San Jose, CA  95134
408-943-2600

October 1, 2004

# CY4632 Keyboard Firmware User's Guide
## Table of Contents


**1. Introduction ..... 5**
- 1.1 Scope ..... 5
- 1.2 Overview ..... 5
- 1.3 Design Goals ..... 5

**2. Definitions ..... 6**

**3. Hardware overview ..... 7**
- 3.1 Schematic of PDC-9174 Board ..... 7
- 3.2 RDK Keyboard Photographs ..... 7
- 3.3 Keyboard Matrix ..... 10

**4. Development Environment ..... 11**
- 4.1 Tools ..... 11
- 4.2 PSoC Configuration ..... 11

**5. Firmware Architecture ..... 12**
- 5.1 Model ..... 12
- 5.2 Normal Keyboard Operation ..... 13
  - 5.2.1 Ghost Key Detection ..... 13
- 5.3 Platform & Architecture Portability ..... 14
- 5.4 Initialization ..... 14
- 5.5 Configuration Options ..... 14
  - 5.5.1 KEYBOARD_KEEP_ALIVE_TIMEOUT ..... 14
  - 5.5.2 KEY_DOWN_DELAY_SAMPLE_PERIOD ..... 14
  - 5.5.3 KEYBOARD_DEBOUNCE_COUNT ..... 15
  - 5.5.4 KEYBOARD_MULTIMEDIA_SUPPORT ..... 15
  - 5.5.5 MFG_TEST_CODE ..... 15
  - 5.5.6 MFG_ENTER_BY_KEY_NOT_PIN ..... 15
  - 5.5.7 KEYBOARD_TEST_MODES ..... 16
  - 5.5.8 KEYBOARD_TEST_MODE_PERIOD ..... 16
  - 5.5.9 PANGRAM_TEST_MODE ..... 16
  - 5.5.10 KEYBOARD_BATTERY_VOLTAGE_SUPPORT ..... 16
  - 5.5.11 CBK_500_KEYBOARD_MATRIX ..... 16
  - 5.5.12 KEYBOARD_FAST_SCAN ..... 16
  - 5.5.13 KEYBOARD_TX_TIMEOUT ..... 16
  - 5.5.14 TIMER_CAL ..... 17
  - 5.5.15 ENCRYPT_DATA ..... 17
  - 5.5.16 MOUSE_EMULATION_MODE ..... 17
  - 5.5.17 KEYBOARD_POWER_ON_BIND ..... 17
  - 5.5.18 PLATFORM_H ..... 17
- 5.6 Radio Subsystem ..... 17
- 5.7 Keyboard Wireless Protocol ..... 18
  - 5.7.1 Keyboard Application Report Formats ..... 18
- 5.8 Test Modes ..... 24
- 5.9 Manufacturing Test Mode ..... 24
- 5.10 Flash Security ..... 25
- 5.11 Battery Monitor ..... 25

**6. Keyboard Code Modules ..... 26**
- 6.1 KEYBOARD ..... 26
  - 6.1.1 Defines & Types ..... 26
  - 6.1.2 Variable Definitions ..... 26
  - 6.1.3 Functions ..... 27

6.2 BATTERY ..... 28
6.2.1 Functions ..... 29
6.3 MFGTEST ..... 29
6.3.1 Defines & Types ..... 29
6.3.2 Functions ..... 31
6.4 PDC9174 ..... 32
6.5 PSOCGPIOINT ..... 32
6.6 ISR ..... 32
6.6.1 Functions ..... 33
6.7 TIMER ..... 33
6.7.1 Defines & Types ..... 34
6.7.2 Variable Definitions ..... 34
6.7.3 Functions ..... 34
6.8 TICKINT ..... 35
**7. References ..... 36**

## Table Listings


Table 1: RDK Keyboard Matrix ..... 10
Table 2: LS Generic Report ..... 19
Table 3: Standard 101 Keys Report Format ..... 19
Table 4: Example "a" down key Standard 101 Keys Report ..... 20
Table 5: Example USB report for the "a" down key ..... 20
Table 6: Standard 101 Key Null Packet Report (Null Packet Support enabled) ..... 20
Table 7: Example up key Standard 101 Keys Report (Null Packet Support disabled) ..... 20
Table 8: Example USB report for a Standard 101 Key Null Packet Report ..... 20
Table 9: Multimedia Keys Report Format ..... 21
Table 10: Example "Volume Increase" down key Multimedia Keys Report ..... 21
Table 11: Example up key Multimedia Keys Report ..... 21
Table 12: Power Keys Report Format ..... 22
Table 13: Example "Suspend/Sleep" down key Power Keys Report ..... 22
Table 14: Example up key Power Keys Report ..... 22
Table 15: Keep Alive Report (Null Packet Support enabled) ..... 22
Table 16: Example Keep Alive Report (Null Packet Support disabled) ..... 23
Table 17: Battery Voltage Level Report Format ..... 23
Table 18: Example "full" Battery Voltage Level Report ..... 23
Table 19: Example "low" Battery Voltage Level Report ..... 24
Table 20: Keyboard Module Defines & Types ..... 26
Table 21: HID_APP Structure Definitions ..... 26
Table 22: Keyboard Module Variable Definitions ..... 26
Table 23: Keyboard Module Functions ..... 27
Table 24: Battery Module Functions ..... 29
Table 25: MFGTest Module Defines & Types ..... 29
Table 26: MFGTest Module Functions ..... 31
Table 27: ISR Module Functions ..... 33
Table 30: Timer Module Defines & Types ..... 34
Table 31: Timer Module Variable Definitions ..... 34
Table 32: Timer Module Functions ..... 34


## Figure Listings


Figure 1: Keyboard Plastic ..... 7
Figure 2: Exploded Keyboard ..... 8
Figure 3: Radio & PSoC Board (PDC-9174) ..... 8
Figure 4: Keyboard Battery Compartment ..... 9
Figure 5: Bind Button ..... 9
Figure 6: RDK Keyboard Architecture ..... 12
Figure 7: Ghost Key Example ..... 13

# 1. INTRODUCTION

## 1.1 Scope

The audience for this document is intended to be firmware developers that desire to understand and modify the firmware on the WirelessUSB LS Keyboard.

## 1.2 Overview

This document will cover the design goals, architecture, firmware source code modules and configuration options for the WirelessUSB LS Keyboard. It will not cover the details of the radio subsystem or the configuration options that go with it. Please refer to the WirelessUSB Theory of Operation and CY4632 Protocol Library documents for theory of operation and details related to the radio subsystem.

## 1.3 Design Goals

There are several design goals that drove the requirements for the firmware development for the keyboard. Some of these are architecture related, while others are feature related.

The CY4632 Reference Design Kit utilizes a PSoC controller for the RDK Keyboard (Cypress part #: CY8C27643-24PVXI). All references to PSoC in this document refer to the CY8C27643-24PVXI PSoC. Please contact your local sales representative for more information on this PSoC controller.

The architecture was designed to be modular for extendibility and maintainability. It was also designed so that it could easily be ported from one hardware platform to another assuming the use of a PSoC microprocessor. While porting to another microprocessor will require more work, the hardware design was done to minimize usage of advanced PSoC features to expedite this effort.

Design efforts have been 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 PSoC microprocessor.

## 2. DEFINITIONS

Following are some definitions of acronyms and words found in this document. There may be other meanings to these definitions outside of this document.

**Bridge** – The Bridge is the receiving radio and USB hardware that connects to the PC and enumerates as a Human Interface Device.

**Device** – The reference to device in this document means the keyboard device that is sending packets via radio to the bridge.

**DVK** – A development kit produced by Cypress Semiconductor for showcasing Cypress products with a working development environment.

**HID** – Stands for Human Interface Device and is a product that allows an individual to interface with a computer. A keyboard and mouse are HID devices.

**PSoC** – A programmable mixed-signal array with an on-board processor.

**RDK** – A reference design kit produced by Cypress Semiconductor and used by 3rd parties to produce off-the-shelf products. Everything required to take a product to production is included in the kit. This document is part of the CY4632 Mouse/Keyboard RDK.

**USB** – The acronym for Universal Serial Bus, a well-known serial standard used in the computing world.

**WirelessUSB** – Trademark name for Cypress 2.4 GHz radio products.

# 3. HARDWARE OVERVIEW

## 3.1 Schematic of PDC-9174 Board

The schematic of the RDK Keyboard (PDC-9174) can be found on the CY4632_RDK release CD at:

\Hardware\RDK Keyboard\LS RDK Keyboard Sch.pdf.

## 3.2 RDK Keyboard Photographs

**Figure 1: Keyboard Plastic**

Figure 1 shows the RDK keyboard plastic.

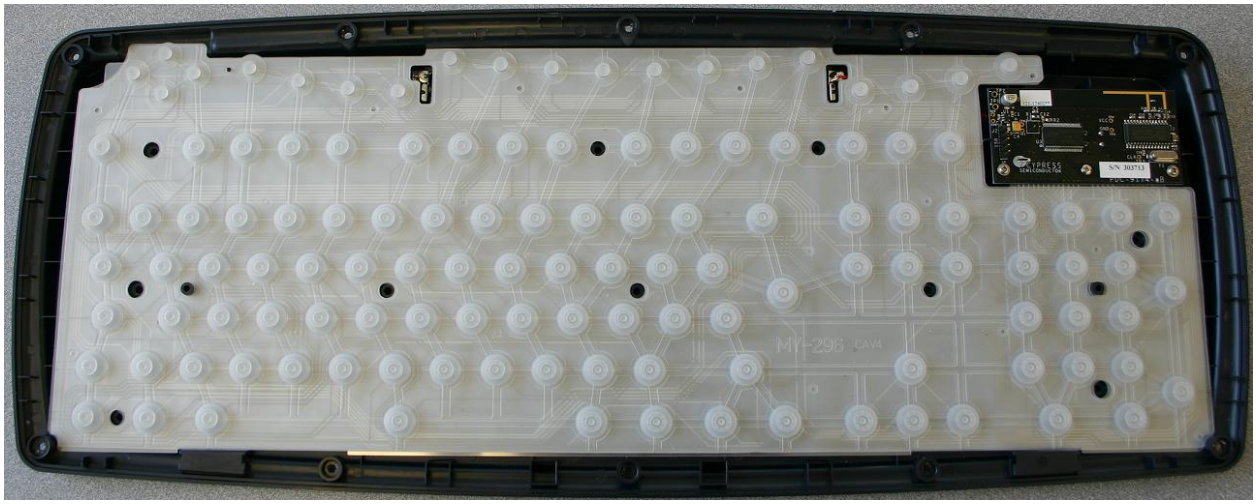

**Figure 2: Exploded Keyboard**

Figure 2 shows the keyboard with the top removed. The radio/PSoC board (PDC-9174) is shown in the upper right hand corner.

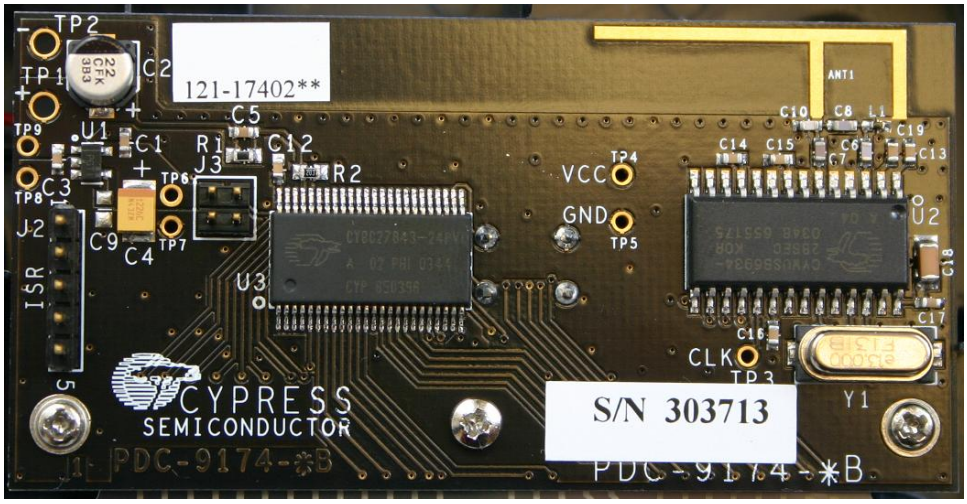


**Figure 3: Radio & PSoC Board (PDC-9174)**

Figure 3 shows the main controller board with the PSoC and WirelessUSB LS Radio. The “F” trace is the antenna. All of the components are on the top side of the board with the exception of the bind button.


**Figure 4: Keyboard Battery Compartment**

Figure 4 shows the integrated battery compartment located on the bottom side of the keyboard. The battery compartment cover is also shown.

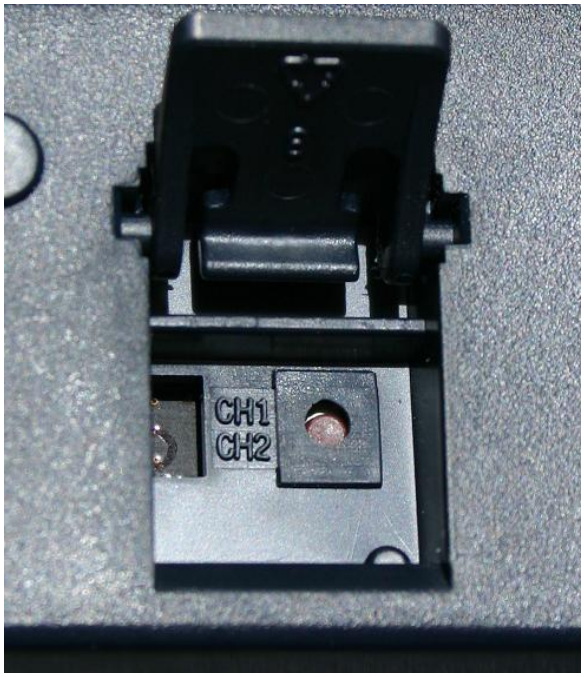


**Figure 5: Bind Button**

Figure 5 shows the bind button. The button access is located beneath the right adjustable foot on the bottom side of the keyboard.

## 3.3 Keyboard Matrix

The RDK keyboard matrix has 18 columns and 8 rows. Key presses generate a PSoC 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 1.

**Table 1: RDK Keyboard Matrix**

| | Row 1 | Row 2 | Row 3 | Row 4 | Row 5 | Row 6 | Row 7 | Row 8 |
|---|---|---|---|---|---|---|---|---|
| Column 1 | 0x14 | 0x17 | 0x1A | 0x1C | 0x08 | 0x18 | 0x15 | 0x0C |
| Column 2 | 0x5A | 0x58 | 0x5B | 0x44 | 0x62 | 0x45 | 0x63 | 0x46 |
| Column 3 | 0x24 | 0x2D | 0x25 | 0x2E | 0x26 | 0x2A | 0x27 | 0x2B |
| Column 4 | 0x60 | 0x5D | 0x61 | 0x5E | 0x56 | 0x57 | 0x5C | 0x59 |
| Column 5 | NA | 0x20 | 0x29 | 0x21 | 0x1E | 0x22 | 0x1F | 0x23 |
| Column 6 | 0x3F | 0x43 | 0x40 | 0x53 | 0x41 | 0x47 | 0x42 | 0x5F |
| Column 7 | NA | 0x00CD | NA | NA | 0x0225 | 0x0226 | NA | 0x0227 |
| Column 8 | 0x39 | 0x3B | 0x2C | 0x3C | 0x4E | 0x3D | 0x3A | 0x3E |
| Column 9 | 0x05 | 0x37 | 0x11 | 0x38 | 0x10 | 0x4C | 0x36 | 0x55 |
| Column 10 | 0x02 | 0x20 | NA | 0x00E2 | 0x0194 | NA | 0x00E9 | 0x65 |
| Column 11 | 0x34 | 0x1D | 0x35 | 0x1B | 0x4D | 0x06 | NA | 0x19 |
| Column 12 | NA | 0x00B6 | 0x08 | 0x82 | NA | NA | NA | NA |
| Column 13 | 0x07 | 0x0D | 0x09 | 0x0E | 0x0A | 0x0F | 0x0B | 0x33 |
| Column 14 | NA | NA | 0x83 | NA | 0x01 | NA | 0x10 | 0x54 |
| Column 15 | 0x12 | 0x28 | 0x13 | 0x51 | 0x2F | 0x04 | 0x30 | 0x16 |
| Column 16 | 0x31 | 0x48 | 0x4A | 0x50 | 0x52 | 0x4F | 0x4B | 0x49 |
| Column 17 | 0x0223 | 0x0183 | 0x0224 | 0x00B7 | NA | 0x80 | 0x00B5 | 0x022A |
| Column 18 | 0x0192 | 0x0221 | NA | 0x04 | 0x00EA | 0x40 | 0x18A | NA |

Notes:

- Yellow indicates Multimedia Key (16-bit value)
- Red indicates Power Key
- Blue indicates Modifier Key
- No color indicates a Standard 101 Key

# 4. DEVELOPMENT ENVIRONMENT

## 4.1 Tools

The following tools are used to develop the RDK keyboard firmware and can be purchased from Cypress Semiconductor.

- PSoC Development Tools Solution
  - PSoC Designer Software version 4.1 SP1
  - PSoC ICE-4000
- ICCM8C C Compiler version 1.28
- Relevant PSoC POD Kit

A Microsoft Windows based PC is used for tool execution.

## 4.2 PSoC Configuration

The PSoC is configured with one 8-bit timer module as a clock and an SPI module for radio communication. An optional UART module may be configured and used as a debug port. All other PSoC features were not utilized to maintain compatibility with other microprocessor implementations.

# 5. FIRMWARE ARCHITECTURE

## 5.1 Model

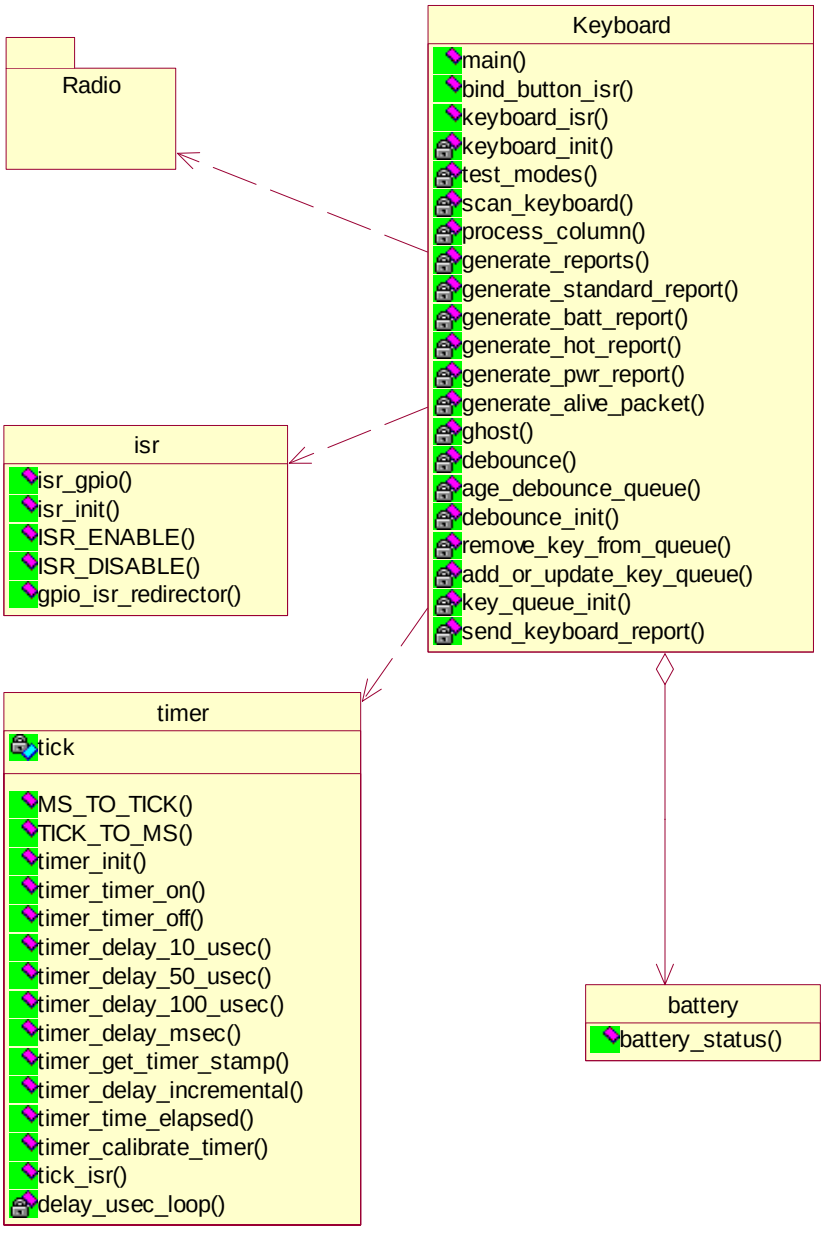


**Figure 6: RDK Keyboard Architecture**

## 5.2 Normal Keyboard Operation

Normal operation of the keyboard is initiated by plugging the WirelessUSB Bridge into a PC and waiting for it to enumerate. Once this is complete, insert three fresh AA batteries into the internal keyboard battery compartment (see Figure 4). In order for the keyboard to communicate with the bridge, they need to execute a “binding” process. Press the “bind” button on the WirelessUSB Bridge and the “bind” button on the keyboard, within approximately 5 seconds of each other, to initialize the “binding” process. The “bind” process can be repeated multiple times until successful. The keyboard will communicate with the Bridge and establish a connection. The keyboard is now ready for normal operation.

### 5.2.1 Ghost Key Detection

Ghost keys are possible on the RDK keyboard because it does not utilize 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,ColumnB) have been pressed as shown in Figure 7. It appears that the key (RowY,ColumnB) has been pressed as well when it has not since the other keys electrically connect RowY to ColumnB.

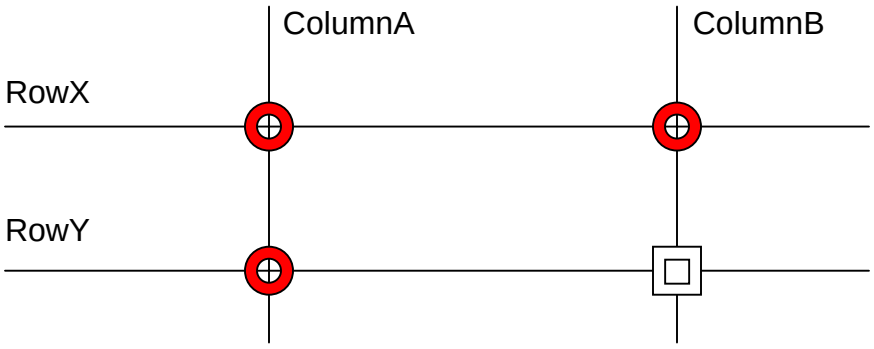


**Figure 7: Ghost Key Example**

## 5.3 Platform & Architecture Portability

The keyboard firmware was designed to be easily ported from one hardware platform to another platform with a simple re-mapping of pins on the PSoC. The file *pdc9174.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 *appconfig.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.

## 5.4 Initialization

Initialization of the PSoC chip is done by code that is generated in *boot.asm* by the PSoC designer software. The module *boot.asm* calls main once the PSoC has been 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.

## 5.5 Configuration Options

All configuration options for the application can be found in the *appconfig.h* file. There are other configuration options, pertaining to the radio subsystem, which can be found in the *ls_config.h* file. A description of these options can be found in the CY4632 Protocol Library document.

### 5.5.1 KEYBOARD_KEEP_ALIVE_TIMEOUT

This configuration value sets the period at which the firmware will generate a “KEEP_ALIVE” packet since the last keyboard report when a key is held down. The default is 100 milliseconds.

### 5.5.2 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 only active when the keyboard has not entered sleep because key(s) are currently being pressed. The default value is 10 milliseconds.

### 5.5.3 KEYBOARD_DEBOUNCE_COUNT

The button debounce logic detects changes in 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 time, 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, then the button debounce time will be 20 milliseconds. The default setting is 2.

### 5.5.4 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.

### 5.5.5 MFG_TEST_CODE

This configuration definition is used to selectively compile in the manufacturing test code.

### 5.5.6 MFG_ENTER_BY_KEY_NOT_PIN

This configuration definition is used to select whether the manufacturing test code is executed by pulling a PSoC pin to ground or by the test mode module. When this value is 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. If it is not defined, then the manufacturing test code may be executed by pulling a defined port pin to ground within 100 milliseconds of power on. See section 6.3.1 for the port pin definition.

### 5.5.7 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. The test modes are described in section 5.8.

### 5.5.8 KEYBOARD_TEST_MODE_PERIOD

This configuration value sets the period that the keyboard generates the 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 100 ms.

### 5.5.9 PANGRAM_TEST_MODE

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

### 5.5.10 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.

### 5.5.11 CBK_500_KEYBOARD_MATRIX

This configuration definition is used to selectively compile in the keyboard matrix for the beta and final RDK keyboard hardware.

### 5.5.12 KEYBOARD_FAST_SCAN

This configuration definition is used to selectively compile in the Cypress Semiconductor fast scan algorithm.

### 5.5.13 KEYBOARD_TX_TIMEOUT

This configuration value sets the maximum time that the keyboard will try to send a report to the bridge.

### 5.5.14 TIMER_CAL

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

### 5.5.15 ENCRYPT_DATA

This configuration definition is used to selectively compile in data encryption for the keyboard. Please contact Cypress Applications support for the encryption source code.

### 5.5.16 MOUSE_EMULATION_MODE

This configuration definition is used to selectively compile in the Mouse Emulation Mode. The Scroll Lock key is used to toggle this mode on/off. Once 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.

### 5.5.17 KEYBOARD_POWER_ON_BIND

This configuration definition is used to selectively compile in the option to enter bind mode on power-up when the device has not been previously bound to a bridge.

### 5.5.18 PLATFORM_H

This configuration value identifies the header file that has the platform configuration information. The default value is *pdc9174.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.

## 5.6 Radio Subsystem

The radio subsystem is composed of the following modules and provides an API to the application for sending data over the radio.

- bind
- protocol
- radio
- spi
- nvram

Please refer to the WirelessUSB Theory of Operation and CY4632 Protocol Library documentation for theory of operation along with a detailed description of the API and configuration options. The ***spi*** module is coupled to the platform in order to communicate with the radio.

## 5.7 Keyboard Wireless Protocol

The keyboard protocol has been optimized to reduce the “on-time” of the radio and power consumption.

The radio library offers the ability to send variable length packets, allowing the opportunity to minimize the number of bytes transmitted over the air, in order 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 and CRC bytes.

### 5.7.1 Keyboard Application Report Formats

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 reports are distinguished by the first application report byte to reduce the number of bytes required in the most common report (Standard 101 Keys 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 Report Type (Multimedia, Power, Battery, or Keep Alive). This also assumes that multimedia and power keys do not utilize modifier keys and that 0xFF, 0xFE, 0xFD and 0xFC are not valid Standard 101 key scan codes.

Packet Headers starting with 0x7X require special handling when Null Packet Support is enabled. A Packet Header of 0x70 is a “NULL” packet and indicates that all the data bytes are zero for the

Standard 101 Keys Report. A Packet Header of 0x79 is a Keep Alive packet.

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

The LS radio sends the reports with the format shown in Table 2.

**Table 2: LS Generic Report**

| Packet Header | Application Report | Checksum |
|---|---|---|

Note: The Packet Headers and Checksums utilized in this document are for example purposes only and may change as required by the protocol.

5.7.1.1 Standard 101 Keys Report

A "Scan Code 1" byte not equal to 0xFF, 0xFE, 0xFD and 0xFC indicates that this report is a Standard 101 Keys Report. The Standard 101 Keys report is formatted to optimize the assumption that the most common report will have only one non-zero scan code without a modifier. The full Standard 101 Keys report format is shown in Table 3.

**Table 3: Standard 101 Keys Report Format**

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

Example: The following reports would be sent if a user presses an "a" on the keyboard. The down key packet sent from the keyboard to the bridge is shown in Table 4.

**Table 4: Example "a" down key Standard 101 Keys Report**

| Packet Header | Application Report | Checksum |
|---|---|---|
| 0x40 | Scan Code 1 | 0x44 |
| | 0x04 | |

The bridge would add the trailing zeros, insert the reserved byte, rearrange the modifier and scan code 1 bytes and remove the packet header and checksum to produce the USB report shown in Table 5.

**Table 5: 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 6 with Null Packet Support enabled and in Table 7 with Null Packet Support disabled.

**Table 6: Standard 101 Key Null Packet Report (Null Packet Support enabled)**

| Packet Header |
|---|
| 0x70 |

**Table 7: Example up key Standard 101 Keys Report (Null Packet Support disabled)**

| Packet Header | Application Report | Checksum |
|---|---|---|
| 0x40 | Scan Code 1 | 0x40 |
| | 0x00 | |

The bridge would add the trailing zeros, insert the reserved byte, and remove the packet header to produce the USB report shown in Table 8.

**Table 8: 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 |

5.7.1.2 Multimedia Keys (Hot keys) Report

A “Report Type” byte of 0xFF indicates that this report is a Multimedia Keys Report. The Multimedia Keys report format is shown in Table 9.

**Table 9: Multimedia Keys Report Format**

| Byte | Name |
|---|---|
| 1 | Report Type (0xFF) |
| 2 | Hot Key Scan Code (upper 8 bits) |
| 3 | Hot Key Scan Code (lower 8 bits) |

Example: The following reports would be sent if a user presses “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 10.

**Table 10: Example "Volume Increase" down key Multimedia Keys Report**

| Packet Header | Application Report | | | Checksum |
|---|---|---|---|---|
| 0x41 | Report Type | Hot Key Scan Code (upper 8 bits) | Hot Key Scan Code (lower 8 bits) | 0x29 |
| | 0xFF | 0x00 | 0xE9 | |

The up key packet sent from the keyboard to the bridge is shown in Table 11.

**Table 11: Example up key Multimedia Keys Report**

| Packet Header | Application Report | Checksum |
|---|---|---|
| 0x45 | Report Type | 0x44 |
| | 0xFF | |

5.7.1.3 Power Keys (Suspend/Sleep) Report

A “Report Type” byte of 0xFE indicates that this report is a Power Keys Report. The Power Keys report format is shown in Table 12.

**Table 12: Power Keys Report Format**

| Byte | Name |
|---|---|
| 1 | Report Type (0xFE) |
| 2 | Power Key Scan Code |

Example: The following reports would be sent if a user presses 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 13.

**Table 13: Example "Suspend/Sleep" down key Power Keys Report**

| Packet Header | Application Report | | Checksum |
|---|---|---|---|
| 0x41 | Report Type | Power Key Scan | 0xC1 |
| | 0xFE | 0x82 | |

The up key packet sent from the keyboard to the bridge is shown in Table 14.

**Table 14: Example up key Power Keys Report**

| Packet Header | Application Report | Checksum |
|---|---|---|
| 0x45 | Report Type | 0x43 |
| | 0xFE | |

5.7.1.4 Keep Alive Report

A "Report Type" byte of 0xFC indicates that this report is a Keep Alive Report. When Null Packet Support is enabled, a "Packet Header" byte of 0x79 also indicates that this report is a Keep Alive Report.

Example of a Keep Alive reports sent from the keyboard to the bridge are shown in Table 15 with Null Packet Support enabled and Table 16 with Null Packet Support disabled.

**Table 15: Keep Alive Report (Null Packet Support enabled)**

| Packet Header |
|---|
| 0x79 |

**Table 16: Example Keep Alive Report (Null Packet Support disabled)**

| Packet Header | Application Report | Checksum |
|---|---|---|
| 0x45 | Report Type | 0x41 |
| | 0xFC | |

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

#### 5.7.1.5 Battery Voltage Level Report

A “Report Type” byte of 0xFD indicates that this report is a Battery Voltage Level Report. The Battery Voltage Level report format is shown in Table 17.

**Table 17: Battery Voltage Level Report Format**

| Byte | Name |
|---|---|
| 1 | Report Type (0xFD) |
| 2 | Battery Voltage Level |

The Battery Voltage Level ranges from 1 (low) to 10 (full).

The Battery Voltage Level Report can be sent at any time. However, it is anticipated that the report will be sent approximately every 10K - 16K reports.

Example of a Battery Voltage Level Report with fully charged batteries is shown in Table 18.

**Table 18: Example "full" Battery Voltage Level Report**

| Packet Header | Application Report | | Checksum |
|---|---|---|---|
| 0x41 | Report Type | Battery Voltage Level | 0x48 |
| | 0xFD | 0x0A | |

Example of a Battery Voltage Level Report with low batteries is shown in Table 19.

**Table 19: Example "low" Battery Voltage Level Report**

| Packet Header | Application Report | | Checksum |
|---|---|---|---|
| 0x41 | Report Type | Battery Voltage Level | 0x3F |
| | 0xFD | 0x01 | |

## 5.8 Test Modes

This RDK provides a compile-time option of adding test modes to the keyboard; see Section 5.5.7 for enabling this option. 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. Once the test has finished execution, the keyboard will return 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. Once the test has finished execution, the keyboard will return to normal operation.

## 5.9 Manufacturing Test Mode

The RDK provides a compile-time option of adding a manufacturing test mode to the keyboard; see Section 5.5.5 for enabling this option.

If MFG_TEST_CODE and ENTER_BY_KEY_NOT_PIN are defined, holding down the system sleep key and the bind button while inserting the batteries into the keyboard will enter the manufacturing test mode. The only way to exit this test mode is to cycle power.

## 5.10 Flash Security

The keyboard project within PSoC Designer has a file called *FlashSecurity.txt*. This file specifies access rules to blocks of the Flash ROM. Please see the documentation at the top of the file for definitions. This file is shipped with its default configuration in the RDK with the exception of one change. The block starting at address 3E80 hex has been changed from **W**: Full (Write protected) to **U**: Unprotected. This location of Flash has been dedicated to saving non-volatile configuration values for the radio subsystem.

## 5.11 Battery Monitor

The battery monitor circuit is implemented using an RC circuit connected to two PSoC pins. The firmware has been tuned to provide a ten level measurement of the battery voltage. This is a process that is time intensive. Following is an explanation of the process to measure the battery voltage.

The process starts by first setting the PSoC pins BATT_LEV1 and BATT_LEV2 to high impedance inputs and allowing the RC circuit to reach a steady state. This allows BATT_LEV1 to reach the same potential as the battery with the capacitor. BATT_LEV2 is then driven to ground and the RC time constant is measured to the point at which the BATT_LEV1 detects a logic 0 state. This time measurement is the battery voltage measurement. Next, both BATT_LEV1 and BATT_LEV2 are driven to a logic 1 state initializing the RC circuit for a VCC measurement. BATT_LEV1 is set to a high impedance input and BATT_LEV2 is driven to ground. Once again the RC time constant is measured to the point at which the BATT_LEV1 detects a logic 0 state. This time becomes a VCC reference measurement.

The firmware then uses the battery measurement and VCC measurement to compute a battery level between 1 and 10 inclusive. Battery status reports should have a minimum of one second between requests to allow the RC circuit to reach a steady state.

# 6. KEYBOARD CODE MODULES

Following are descriptions of the module contents. For concept of operation, see section 5.

## 6.1 KEYBOARD

The keyboard module contains the main entry point, test modes, a routine for scanning the keyboard as well as management of report frequency to the Bridge.

### 6.1.1 Defines & Types

**Table 20: Keyboard Module Defines & Types**

| Define/Type | Description |
|---|---|
| APP_TX_PACKET | This union structure defines the data payload portion of the radio transmit packet. |
| HID_APP | This structure defines all of the HID related variables. |

**Table 21: HID_APP Structure Definitions**

| Element | Description |
|---|---|
| key_queue | A KEY structure array of KEY_QUEUE_LEN elements. Used to store pressed key index values. |
| status | A STATUS_STATE structure used to maintain the current status of the keyboard. |
| debounce | A DEBOUNCE_ENTRY structure array of DEBOUNCE_QUEUE_LENGTH elements. Used to debounce pressed key index values. |
| prior_key_state | An UINT8 array of COLUMNS elements. Used to store the previous state of keyboard matrix. |

### 6.1.2 Variable Definitions

**Table 22: Keyboard Module Variable Definitions**

| Variable | Description |
|---|---|
| hid | A HID_APP structure containing the HID related variables. |
| report_packet | This is a pointer to an APP_TX_PACKET structure. |
| ts | A TIME_STAMP structure used for timing. |

| | |
|---|---|
| last_transfer_ts | A TIME_STAMP structure used for timing to generate KEEP_ALIVE packets. |
| test_mode | A variable used to store the current test mode state if KEYBOARD_TEST_MODES is defined. |
| sentence_table_index | A variable used to store the current position in the test sequence if KEYBOARD_TEST_MODES is defined. |
| mouse_emulation | A variable used to store the current state of the MOUSE_EMULATION_MODE. |

### 6.1.3 Functions

**Table 23: Keyboard Module Functions**

| Function | Linkage | Description |
|---|---|---|
| main | external | This function is the main entry point for the application. It initializes all keyboard application components and radio components and executes the poll loop. Radio transmission takes place when there are keyboard events to send. |
| bind_button_isr | external | This function is called from interrupt context when the bind button is pressed. |
| keyboard_isr | external | This function is called from interrupt context when a key is pressed. |
| keyboard_init | static | This function initializes all components of the keyboard application. |
| test_modes | static | This function adds the appropriate test key index to the key queue if a test mode is active. |
| scan_keyboard | static | This function scans the keyboard matrix and stores the results in the array hid.key_state[]. |
| process_column | static | This function compares the arrays hid.key_state[] to hid.prior_key_state[] for the specified column. |

| | | |
|---|---|---|
| generate_reports | static | This function calls the other generate functions if the GENERATE_REPORT status bit is set. |
| generate_standard_report | static | This function generates the Standard 101 Keys report. |
| generate_batt_report | static | This function generates the Battery Voltage Level report. |
| generate_hot_report | static | This function generates the Multimedia (Hot) Keys report. |
| generate_pwr_report | static | This function generates the Power Keys report. |
| generate_alive_packet | static | This function generates the Alive Packet report. |
| ghost | static | This function determines if a ghost key condition exists. |
| debounce | static | This function starts the debounce clock when a button is pressed and blocks out further button changes until the debounce time has expired. |
| age_debounce_queue | static | This function decrements debounce values for all keys every time the keyboard buttons are polled. This provides a debounce period for when a button is pressed |
| debounce_init | static | This function initializes the debounce logic for the buttons. |
| remove_key_from_queue | static | This function removes the specified key index from the hid.key_queue[] array. |
| add_or_update_key_queue | static | This function adds the specified key index to the hid.key_queue[] array. |
| key_queue_init | static | This function initializes the hid.key_queue[] array. |
| send_keyboard_report | static | This function attempt to send the keyboard report to the bridge. |

## 6.2 BATTERY

This module measures the battery voltage level and computes a level from 1 to 10. Time for the RC circuit to stabilize with the battery voltage needs to be given before calling this module. An application should guarantee that a minimum time of one second between successive battery status requests.

### 6.2.1 Functions

**Table 24: Battery Module Functions**

| Function | Linkage | Description |
|---|---|---|
| battery_status | external | This function reads a time constant related to the battery voltage and VCC (for reference) and computes a battery level from 1 to 10. See section 5.11 for details of operation. |

## 6.3 MFGTEST

This module may be conditionally compiled in to provide manufacturing test support. The module configures the radio for reception and then enters a loop waiting for packets to be sent from the tester. It computes the total number of bits received and the number of invalid bits received. It then sends packets to the tester followed by the previously computed results. After which it re-enters the reception loop waiting for another test cycle. The manufacturing test code may be entered by grounding a microprocessor port pin or by the test mode mechanism defined in section 5.8. The method of entry is a compile time option described in section 5.5.6. The manufacturing test code can only be exited by cycling power.

### 6.3.1 Defines & Types

**Table 25: MFGTest Module Defines & Types**

| Define/Type | Description |
|---|---|
| MFG_PN_CODE | This macro defines the PN code used for the manufacturing test. See radio documentation. |
| MFG_CHANNEL | This macro defines the channel used for the manufacturing test. See radio documentation. |

| | |
|---|---|
| MFG_THRESHOLD_L | This macro defines the threshold when correlating a '0' data bit. See the radio documentation. |
| MFG_THRESHOLD_H | This macro defines the threshold when correlating a '1' data bit. See the radio documentation. |
| MFG_PA_BIAS | This macro defines the power amplifier bias used for the test. See radio documentation. |
| MFG_EOF_TIMEOUT | This macro defines the number of bits that need to be seen before end-of-frame is detected. See radio documentation. |
| MFG_RX_TIMEOUT | The number of milliseconds between two packets, once packet sending has started, before the receiving loop will timeout. |
| MFG_NUM_PACKETS | The number of packet expected to receive and send. |
| MFG_PACKET_PAYLOAD_SIZE | The data size of the packets being received and sent. Does not include header and checksum overhead. |
| MFG_PRE_TX_DELAY | The time in milliseconds to wait after receiving and before sending out data packets. |
| MFG_INTER_TX_DELAY | The time in milliseconds to wait in between data packets being sent out. |
| MFG_NUM_RESULTS_PACKETS | The number of times to send each result packet. |
| MFG_PRE_RESULTS_DELAY | The time in milliseconds to wait before sending out the results packets. |
| MFG_INTER_RESULTS_DELAY | The time in milliseconds in between each repetition of result packet sent. |
| MFG_TEST_PORT | The port owning the pin used to enter manufacturing test mode. |
| MFG_TEST_PIN | The pin used to enter manufacturing test mode. This is used in conjunction with MFG_TEST_PORT. |

| | |
|---|---|
| MFG_TEST_DM0 | Drive mode register 0 for the port used in MFG_TEST_PORT. This is used to change drive characteristics during power-up. |
| MFG_TEST_DM1 | Drive mode register 1 for the port used in MFG_TEST_PORT. This is used to change drive characteristics during power-up. |
| MFG_HDR_TEST | Header byte for test data packets sent to the tester from the device. |
| MFG_HDR_RES1 | Header byte for results packet number one. |
| MFG_HDR_RES2 | Header byte for results packet number two. |
| MFG_HDR_RES3 | Header byte for results packet number three. |
| MFG_TEST_PKT_SIZE | Total packet size defined as MFG_PACKET_PAYLOAD_SIZE + header byte + checksum byte. |
| MFG_RES_PKT_SIZE | Packet size for results packets. This is hard coded to 10 bytes and includes header byte and checksum byte. |

### 6.3.2 Functions

**Table 26: MFGTest Module Functions**

| Function | Linkage | Description |
|---|---|---|
| mfg_pin_check | external | This function checks the state of the defined manufacturing port pin. If the port pin is low, then the manufacturing test is executed. |
| mfg_test | external | This is the top-level function for the manufacturing test. |
| mfg_setup_radio | static | This function initializes the radio to a manufacturing test state. |
| mfg_receive_test_packets | static | This function receives test packets sent from the tester to the device. |
| mfg_send_test_packets | static | This function sends test packets from the device to the tester. |

| | | |
|---|---|---|
| mfg_send_results | static | This function sends statistics of data that was received by the device. |
| mfg_send_result_packet | static | This function performs the actual sending of the results packet with appropriate delays. |

## 6.4 PDC9174

This module is used to implement platform specific code. Currently the *pdc9174.c* file is empty. The *pdc9174.h* file contains the entire platform specific defines for pin and port assignments for a specific feature. Porting from one platform to another should only require modifications to these two files assuming no other features are added or removed.

## 6.5 PSOCGPIOINT

This module is an assembly file and is generated automatically by the PSoC Designer in the lib directory. It is mentioned here because it needs to be modified after PSoC Designer has generated the code. PSoC Designer generates code such that when a GPIO interrupt occurs the vector jumps to this module. This module in turn needs to jump to the *isr_gpio* interrupt service routine provided in the ISR module, see section 6.6. If GPIO interrupts stop working, then this is a good place to check since this is a generated file.

## 6.6 ISR

The purpose of this module is to handle GPIO interrupt handling. It provides a single entry point for all GPIO interrupts and calls functions based upon which interrupt is enabled. The PSoC does not provide the ability to determine which GPIO generated the interrupt.

### 6.6.1 Functions

**Table 27: ISR Module Functions**

| Function | Linkage | Description |
|---|---|---|
| isr_gpio | external | This is the main GPIO interrupt service routine. It calls interrupt handlers if the interrupt is enabled using the *gpio_isr_redirector*() function. The interrupt handlers need to be callable even if they are not the source of the interrupt. |
| isr_init | external | This function initializes the interrupt enable registers and turns on interrupts for the microprocessor. |
| ISR_ENABLE | external | This macro function enables an interrupt handler at its associated port pin. |
| ISR_DISABLE | external | This macro function disables an interrupt handler at its associated port pin. |
| gpio_isr_redirector | external | This function is used as a mechanism to bypass the compiler's inefficient ISR mechanism of pushing all registers when an ISR makes another function call |

## 6.7 TIMER

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 timer block on the PSoC. The delay function used for millisecond timing will provide at least the delay requested with no more than one additional millisecond of delay. The microsecond delay functions have been tuned as best as possible for a 12 MHz clock setting for the microprocessor. The millisecond delay function will sleep the PSoC for the duration of the requested delay. The microprocessor wakes just long enough to update the tick every millisecond and check if the delay has been met and then returns to sleep mode if it

has not. See documentation in the module for requirements on configuring the PSoC block.

### 6.7.1 Defines & Types

**Table 28: Timer Module Defines & Types**

| Define/Type | Description |
|---|---|
| TIME_STAMP | This defines the type of the time stamp. |
| MS_TO_TICK | This macro is used to convert milliseconds to ticks. |
| TICK_TO_MS | This macro is used to convert ticks to milliseconds. |
| timer_delay_1_usec | This macro is used to delay for 1 microsecond with a microprocessor clock of 12MHz. |

### 6.7.2 Variable Definitions

**Table 29: Timer Module Variable Definitions**

| Variable | Description |
|---|---|
| tick | This variable keeps track of the number of ticks since the timer was turned on. |

### 6.7.3 Functions

**Table 30: Timer Module Functions**

| Function | Linkage | Description |
|---|---|---|
| tick_isr | external | This function is the interrupt service routine for the timer expired interrupt. It increments the tick counter. |
| timer_init | external | This function initializes the tick counter and starts the timer. |
| timer_timer_on | external | This function turns the timer and system tick on. |
| timer_timer_off | external | This function turns the timer and system tick off. |
| timer_delay_10_usec | external | This function delays for 10 microseconds. |
| timer_delay_50_usec | external | This function delays for 50 microseconds. |
| timer_delay_100_usec | external | This function delays for 100 microseconds. |

| | | |
|---|---|---|
| timer_delay_msec | external | This function delays for the number of requested milliseconds plus up to one additional millisecond. |
| timer_get_time_stamp | external | This function returns the current tick counter value. |
| timer_delay_incremental | external | This function takes a time stamp and delays for the incremental time between the time stamp and the requested period. This function is used for meeting poll rate requirements. |
| timer_time_elapsed | external | This function returns true if the specified amount of time has elapsed since a given time stamp. |
| timer_calibrate_timer | external | This function calibrates the one-millisecond timer by using the System Clock as a reference. |

## 6.8 TICKINT

This module is an assembly file and is generated automatically by the PSoC Designer in the lib directory. It is mentioned here because it needs to be modified after PSoC Designer has generated the code. PSoC Designer generates code such that when a Timer interrupt occurs the vector jumps to this module. This module in turn needs to jump to the *tick_isr* interrupt service routine discussed in section 6.7.

# 7. REFERENCES

CY3632 Wireless USB Development Kit documents

WirelessUSB Theory of Operation

CY4632 Protocol Library

PSoC Designer version 4.1 documentation

CY4632 RDK Kit schematics