CY5672

PRoC™ BLE Remote Control Reference Design Kit Guide

Doc. No. 001-97071 Rev. *A

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709
http://www.cypress.com

Copyrights

© Cypress Semiconductor Corporation, 2015-2017. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion LLC (“Cypress”). This document, including any software or firmware included or referenced in this document (“Software”), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress’s patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage (“Unintended Uses”). A critical component is any component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

Cypress, the Cypress logo, Spansion, the Spansion logo, and combinations thereof, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit cypress.com. Other names and brands may be claimed as property of their respective owners.

Contents

Safety Information **6**

1. Introduction **8**

1.1 Kit Contents 8
1.2 PRoC BLE Remote Control Details 9
1.3 PSoC Creator 11
1.4 Getting Started 12
1.5 Additional Learning Resources 12
1.5.1 Beginner Resources 12
1.5.2 Component Datasheets 12
1.5.3 Learning from Peers: Cypress Developer Community Forums 12
1.5.4 Other Kits 13
1.6 Technical Support 13
1.7 Document Conventions 13
1.8 Acronyms 14

2. Installation **15**

2.1 Before You Begin 15
2.2 Install Software 15
2.3 Uninstall Software 18

3. Kit Operation **19**

3.1 Connecting the PRoC BLE Remote Control with the CySmart USB Dongle to Communicate with PC 19
3.2 Connecting the PRoC BLE Remote Control with the CySmart USB Dongle to Communicate with the CySmart Software 20
3.2.1 Connecting the CY5672 Remote Control RDK to CySmart Software 21
3.3 Programming and Debugging 25
3.3.1 Programming and Debugging PRoC BLE on the Remote Control and Dongle 25
3.3.2 Programming PRoC BLE on the Remote Control and Dongle Using PSoC Programmer 25
3.3.3 Programming and Debugging PRoC BLE on Remote Control and Dongle Using PSoC Creator 28
3.3.4 Programming PSoC 5LP on the CySmart USB Dongle 32

4. Hardware **35**

4.1 Board Details 35
4.2 Theory of Operation 38
4.2.1 Overview of the PRoC BLE Remote Control 38
4.2.2 Overview of CySmart Dongle 39
4.3 Functional Description – PRoC BLE Remote Control 40
4.3.1 Power Supply 40

4.3.2 Clock and Reset 41
4.3.3 Program and Debug Circuit 42
4.3.4 PRoC BLE Device 42
4.3.5 LED 43
4.3.6 Button Matrix 44
4.3.7 IR LED 44
4.3.8 Motion Sensor 45
4.3.9 Battery Monitoring 46
4.3.10 Audio Input Device 46
4.3.11 Trackpad Interface 47
4.3.12 Test Points 48
4.4 Functional Description – CySmart USB Dongle 49
4.4.1 Power Supply System and Protection Circuitry 49
4.4.2 Clock and Reset 49
4.4.3 Program and Debug Circuit 50
4.4.4 PRoC BLE 51
4.4.5 PSoC 5 LP 52
4.4.6 LEDs and Buttons 53
4.4.7 USB Plug 54

5. Firmware 55

5.1 Firmware for PRoC BLE on the Remote Control 55
5.1.1 Firmware Architecture 56
5.1.2 Watchdog Timer Subsystem 64
5.1.3 BLE Subsystem 65
5.1.4 Trackpad Subsystem 73
5.1.5 Motion Sensor Subsystem 82
5.1.6 Button Subsystem 92
5.1.7 Keyboard Subsystem 95
5.1.8 Timer Subsystem 98
5.1.9 Battery Monitoring Subsystem 98
5.1.10 Audio Subsystem 100
5.1.11 IR LED Subsystem 108
5.1.12 LED Subsystem 113
5.1.13 Flash Subsystem 115
5.1.14 Debug Subsystem 115
5.2 Firmware for the PRoC BLE and PSoC 5LP on the CySmart USB Dongle 116
5.2.1 Firmware Architecture 116
5.2.2 BLE Subsystem 124
5.2.3 LED Subsystem 126
5.2.4 Button Subsystem 127
5.2.5 UART Subsystem 127
5.2.6 USB Subsystem 135

6. Advanced Topics 138

6.1 Connecting PRoC the BLE Remote Control with a Bluetooth Smart Ready Device 138
6.1.1 Windows 8.1 PC 138
6.1.2 MacBook Pro 139
6.1.3 Android 139

6.2 Current Measurement 140
6.2.1 Measure System Current Consumption 140
6.2.2 Measure PRoC BLE Current Consumption 140
6.2.3 Measure Voltage 141
6.3 Assembling and Disassembling the Remote Control 142
6.3.1 Disassembly Procedure 142
6.3.2 Assembly Procedure 142

Appendix A. Schematics and FAQ 143

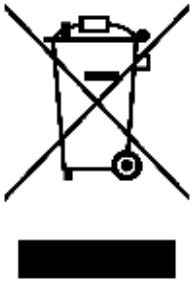
A.1 Schematics 143
A.1.1 Remote Control 143
A.1.2 Dongle 148
A.2 FAQ 150
A.3 Troubleshooting and FAQ for Interoperability Issues with Bluetooth Smart Ready Devices 152

Revision History 153

Document Revision History 153

Safety Information

The CY5672 PRoC™ BLE Remote Control Reference Design Kit (RDK) is intended for use as a development, demonstration, and evaluation platform for hardware or software in a laboratory environment. The kit is not intended for general consumer use. It generates, uses, and can radiate radio frequency energy. It has not been tested for compliance with the limits applicable under any standard. Operation of the equipment may cause interference with radio communications, in which case users, at their expense, will be required to take whatever measures may be required to correct this interference. Cypress recommends that the kit be used only in a shielded room.

	The CY5672 PRoC BLE Remote Control RDK boards contain electrostatic discharge (ESD)-sensitive devices. Electrostatic charges readily accumulate on the human body and any equipment, which can cause a discharge without detection. Permanent damage may occur to devices subjected to high-energy discharges. Proper ESD precautions are recommended to avoid performance degradation or loss of functionality. Store unused CY5672 PRoC BLE Remote Control RDK boards in the protective shipping package.
	End-of-Life/Product Recycling The end-of-life cycle for this kit is five years from the date of manufacture mentioned on the back of the box. Contact the nearest recycler to discard the kit.

General Safety Instructions

ESD Protection

ESD can damage boards and associated components. Cypress recommends that the user perform procedures only at an ESD workstation. If an ESD workstation is not available, use appropriate ESD protection by wearing an antistatic wrist strap attached to the chassis ground (any unpainted metal surface) on the board when handling parts.

Handling Boards

The boards provided with CY5672 are sensitive to ESD. This also applies to the boards that are provided with a plastic casing when they are removed from the casing. Hold the boards only by the edges. After removing a board from the box/casing, place it on a grounded, static-free surface. Use a conductive foam pad, if available. Do not slide the board over any surface.

Battery Care and Use

- Use the correct size and type of battery specified in this guide.
- Keep battery contact surfaces and battery compartment contacts clean by rubbing them with a clean pencil eraser or a rough cloth each time you replace the batteries.
- Remove the battery from a device when it is not expected to be used for several months.
- Make sure that you insert the battery into your device properly, with the + (plus) and – (minus) terminals aligned correctly.
- Do not place the battery next to metallic objects such as keys and coins.
- Never throw the battery into a fire.
- Do not open up the battery.
- Do not short the battery.
- Do not subject the battery to high temperatures or high humidity.
- Store the battery in a dry place.
- Do not recharge a battery unless it is marked "rechargeable."

Battery Disposal

Batteries can be safely disposed of with normal household waste. Never dispose of batteries in a fire because they can explode.

It is important not to dispose of large amounts of batteries in a group. Used batteries are often not completely “dead.” Grouping used batteries together can bring the “live” batteries into contact with one another, creating safety risks.

Regulatory Compliance Information

This kit contains devices that transmit and receive radio signals in accordance with the spectrum regulations for the 2.4-GHz unlicensed frequency range. This kit has passed precompliance tests for the following regulatory standards:

- CE
- FCC Part 15
- Canadian RSS-210
- EU EN 300 328

Contact Cypress technical support at bleapps@cypress.com for further details.

1. Introduction

Thank you for your interest in the CY5672 PRoC™ BLE Remote Control Reference Design Kit (RDK). This kit provides an implementation of a Bluetooth LE (BLE), or Bluetooth Smart, remote control using the PRoC BLE single-chip solution. PRoC BLE is a true single-chip solution that integrates Cypress's industry-leading capacitive touch sensing, an energy-efficient ARM® Cortex™-M0 CPU core, Bluetooth 4.1–compliant Bluetooth Smart connectivity, and a balun to minimize external components for a wide variety of applications.

The kit includes a CySmart™ USB dongle, which can be used to connect the remote control to devices like PCs, laptops, tablets, and smartphones. The dongle acts as a serial-USB bridge and can either pass data directly to the PC's HID drivers or it can interface with the CySmart tool for more detailed application analysis and debugging. The remote control and the dongle can be programmed, enabling users to customize and evaluate firmware per their requirements. Schematics, layout files, ready-to-use firmware, and source code are provided with the kit to enable PC peripheral designers to build their own remote control designs.

1.1 Kit Contents

The CY5672 PRoC BLE Remote Control RDK includes the following items, as shown in Figure 1-1.

1. PRoC BLE remote control
2. CySmart USB dongle
3. MiniProg3 programmer/debugger
4. 10-pin ribbon cable
5. Two M2 X 5mm screws
6. Two AAA batteries
7. Phillips head screwdriver
8. USB 2.0 standard A to mini-B cable

The kit also includes a quick start guide to enable users to connect the PRoC BLE remote control to a PC using the CySmart USB dongle.

Figure 1-1. Kit Contents



Note: The color of the remote control received as part of the kit may differ from the color shown in Figure 1-1.

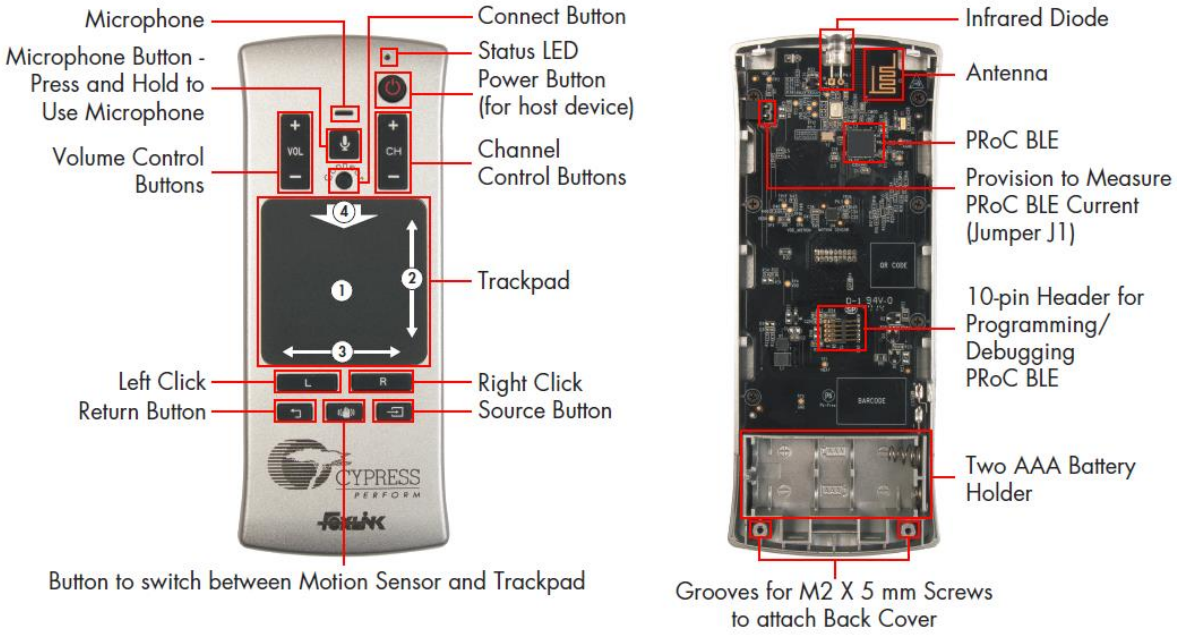
If any part of the kit is missing, contact your nearest Cypress sales office for help: www.cypress.com/go/support.

1.2 PRoC BLE Remote Control Details

This section details the blocks and explains the features of the PRoC BLE remote control.

Note: Figure 1-2 shows the PRoC BLE remote control without the back cover, so you can see all its features.

Figure 1-2. PRoC BLE Remote Control Features



The PRoC BLE remote control can be connected to a host device like a PC, a smart TV, a tablet, or a smartphone using Bluetooth Smart or USB (using the CySmart USB dongle). The quick start guide provided with the kit details the procedure to connect the PRoC BLE remote control to a PC using the CySmart USB dongle. You can connect the PRoC BLE remote control to any Bluetooth Smart or Bluetooth Smart Ready host device. Details for this procedure are available in the Connecting PRoC the BLE Remote Control with a Bluetooth Smart Ready Device section.

Once connected (via Bluetooth Smart or via USB), the PRoC BLE remote control provides features that can be directly used with a PC, smart TV, tablet, or smartphone.

Features supported on a PC or smart TV include the following:

- Gestures on trackpad
  - Single-finger tracking for mouse cursor movement
  - Single-finger tap for left-click and double-tap for double-click
  - Two-finger tap for right-click
  - Two-finger pinch in/out for zoom out/in
  - Single-finger swipe at right and bottom edges for vertical and horizontal scrolling respectively
  - Single-finger swipe down on top edge to navigate to Start screen in Windows 8 or later and to open Start menu in previous Windows versions

    Note: The behavior of this gesture on a smart TV, tablet, or smartphone varies based on the model and operating system)
- Buttons for left-click and right-click
- Volume control buttons to adjust the volume

Advanced features supported on a smart TV or supported via the CySmart tool on a PC include the following:

- Power button to switch on or switch off the smart TV
- Channel control buttons to change channels
- Source button to switch across audio-video inputs
- Return button for going back to previous view on a smart TV

When used with the CySmart USB dongle, the onboard microphone can be used to stream voice data to the PC, smart TV, and so on. The CySmart USB dongle enumerates as a USB microphone. A tool like a sound recorder can be used on the PC to record the voice data coming from the remote. This feature cannot be used when directly connected to a Bluetooth Smart Ready device since Bluetooth 4.0/4.1 does not specify standard audio profiles. The audio data is transmitted from the remote to the dongle using the HID Over GATT Profile (HOGP).

Note: When using the CySmart USB dongle, these features are dependent upon the operating system supporting the generic desktop and consumer usage pages of the human interface device (HID) class as well as the audio device class, over USB. When the PRoC BLE remote control is directly connected to a Bluetooth Smart Ready device, these features depend on the support available for the HOGP over Bluetooth Smart.

The PRoC BLE remote control also provides IR-based wireless connectivity. IR enables the PRoC BLE remote control to be backward compatible with legacy host devices such as TVs that do not support Bluetooth Smart or USB. The following features are supported over IR:

- Power button to switch on or switch off the TV
- Volume control buttons to adjust the volume
- Channel control buttons to change channels
- Source button to switch across audio-video inputs
- Return button for going back to previous view on a smart TV

The remote control also provides LED-based notifications:

- Red LED is on when remote control is advertising over Bluetooth Smart. It blinks three times and then switches off after successful connection.
- Red LED blinks on user activity when the remote control has lost the BLE link and it is trying to connect back to the previously connected host such as a PC or smart TV. This typically occurs when the remote control goes out of range.
- Red LED shows a slow breathing effect on user activity when the battery level is low (that is, the battery voltage is below 2.0 V).
- Red LED shows a fast breathing effect on user activity if the remote control is not connected to the CySmart USB dongle or a Bluetooth Smart Ready host device.

Moreover, the remote control offers the ability to:

- Debug and program the onboard PRoC BLE device using the 10-pin header
- Measure the current consumed by PRoC BLE

1.3 PSoC Creator

PSoC Creator™ is a state-of-the-art, easy-to-use integrated design environment (IDE). It introduces a revolutionary hardware and software co-design platform, powered by a library of verified and characterized PSoC Components™.

With PSoC Creator, you can:

- Drag and drop PSoC Components to build a schematic of your custom design
- Automatically place and route Components and configure GPIOs
- Develop and debug firmware using the included Component APIs

PSoC Creator also enables you to tap into an entire tool ecosystem, including integrated compiler chains and production programmers for PSoC® devices. For more information, visit www.cypress.com/creator.

1.4 Getting Started

This guide will help you get acquainted with the CY5672 PRoC BLE Remote Control RDK.

- The Kit Contents section details the contents of the kit.
- The quick start guide shipped with the kit provides instructions on how to start using the PRoC BLE remote control.
- The PRoC BLE Remote Control Details section details the features of the remote control.
- The Installation chapter describes the installation of the kit software. This includes the PSoC Creator IDE for development, programming, and debugging; PSoC Programmer for programming hex files; and the CySmart tool for Bluetooth LE host emulation
- The Kit Operation chapter describes common procedures like programming and debugging the PRoC BLE remote control and the CySmart USB dongle. It also introduces the CySmart tool for Bluetooth LE host emulation.
- The Hardware chapter describes the hardware contents of the kit.
- The Firmware chapter describes the firmware on the PRoC BLE remote control and the CySmart USB dongle.
- The Advanced Topics chapter explains topics like connecting the remote control to Bluetooth Smart Ready devices, assembling and disassembling the remote control, and measuring current/voltage on the remote control.
- Appendix A. Schematics and FAQ provides schematics and a list of frequently asked questions (FAQs) for troubleshooting.

1.5 Additional Learning Resources

Visit www.cypress.com/PRoCBLE for additional learning resources in the form of datasheets, technical reference manuals, and application notes.

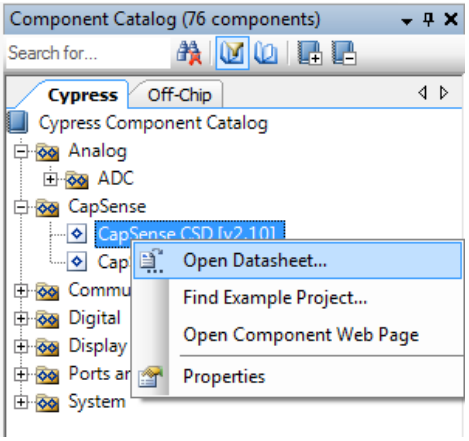
1.5.1 Beginner Resources

PSoC Creator Training: www.cypress.com/go/creatorstart/creatortraining

1.5.2 Component Datasheets

To view the datasheet of a Component from PSoC Creator, right-click on the Component and select **Open Datasheet** (see Figure 1-3).

Figure 1-3. Opening Component Datasheet



1.5.3 Learning from Peers: Cypress Developer Community Forums

Visit www.cypress.com/forums to interact with Cypress applications support teams.

1.5.4 Other Kits

Other kits available for PRoC BLE include the following.

- CY8CKIT-042-BLE Bluetooth Low Energy Pioneer Kit (www.cypress.com/CY8CKIT-042-BLE)
- CY5682 PRoC BLE Touch Mouse RDK (www.cypress.com/CY5682)
- CY5671 PRoC BLE Module (www.cypress.com/CY5671)
- CY5670 CySmart USB Dongle (www.cypress.com/CY5670)
- CY5674 PRoC BLE SMA Module (www.cypress.com/CY5674)

1.6 Technical Support

For assistance, visit Cypress Support or contact customer support at +1(800) 541-4736 Ext. 2 (in the USA) or +1 (408) 943-2600 Ext. 2 (International).

1.7 Document 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.

1.8 Acronyms

Table 1-2. Acronyms Used in This Document

Acronym	Definition
ADC	Analog-to-digital converter
AML	Air Motion Library
API	Application programming interface
BD address	Bluetooth device address
BLE	Bluetooth Low Energy
CDC	Communications Device Class
ESD	Electrostatic discharge
GAP	Generic Access Profile
GATT	Generic Attribute Profile
GPIO	General-purpose I/O
GUI	Graphical user interface
HID	Human-interface device
HOGP	HID over GATT Profile
I^2C	Inter-integrated circuit
IAS	Immediate Alert Service
IDAC	Interconnecting digital-analog converter
IDE	Integrated development environment
LED	Light-emitting diode
MTU	Maximum transmission unit
PHY	Physical layer
PrISM	Precise Illumination Signal Modulation
PRoC	Programmable Radio-on-Chip
PSoC	Programmable System-on-Chip
PWM	Pulse-width modulator
QFN	Quad flat no-lead (package)
SAR	Successive approximation register (ADC)
SWD	Serial wire debug
UART	Universal asynchronous receiver transmitter
UUID	Universally Unique Identifier

2. Installation

This chapter describes the steps to install the software tools and packages on a PC for the CY5672 PRoC BLE Remote Control RDK. The kit installer will install the following software tools and associated documentation on your PC.

- PSoC Creator: IDE used to view, develop, and build firmware for PRoC BLE based devices.
- PSoC Programmer: Software used to download a hex file to the PRoC BLE device.
- CySmart: A Bluetooth LE host emulation tool for windows PCs. The tool works with the CySmart USB dongle and provides an easy-to-use GUI to enable customers to evaluate and debug Bluetooth LE peripheral applications. Refer to the section Connecting the CY5672 Remote Control RDK to CySmart Software for more details on how to use the CySmart tool.
- Documentation: Includes kit documentation files, hardware schematics, layout, and BOM.

2.1 Before You Begin

All Cypress software installations require administrator privileges. Administrator privileges are not required to execute the software after installation. Before you install the kit software, close any other Cypress software that is currently running.

2.2 Install Software

Follow these steps to install the CY5672 PRoC BLE Remote Control RDK software:

1. Download the CY5672 PRoC BLE Remote Control RDK software from www.cypress.com/CY5672. The kit software is available in three formats for download:
   a. **CY5672 Kit Setup:** This installation package contains all files related to the kit as well as PSoC Creator, PSoC Programmer, and CySmart software. However, it does not include the Windows Installer or Microsoft .NET framework packages. If these packages are not on your computer, the installer directs you to download and install them from the Internet.
   b. **CY5672 Kit Only:** This executable file installs only the kit contents, which include kit source code and firmware, hardware files, and user documents. This package can be used if all the software prerequisites (PSoC Creator, PSoC Programmer, and CySmart) are installed on your PC.
   c. **CY5672 DVD ISO:** This file is a complete package, stored in a DVD-ROM image format, that can be used to create a DVD or extract using an ISO extraction program such as WinRAR. The file can also be mounted like a virtual DVD using virtual drive programs such as Virtual CloneDrive and MagicISO. This file includes all the required software, utilities, drivers, hardware files, and user documents.
2. If you have downloaded the ISO file, mount it in a virtual drive. Extract the ISO contents if you do not have a virtual drive to mount. Double-click *cyautorun.exe* in the root directory of the extracted content or mounted ISO if “Autorun from CD/DVD” is not enabled on the PC. The installation window will appear automatically.

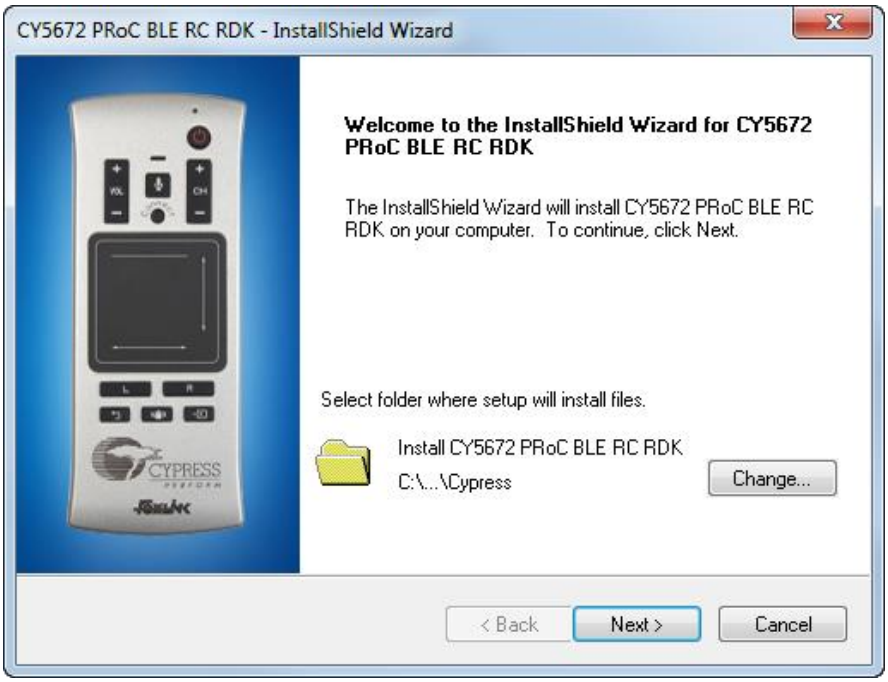
   Note: If you are using the CY5672 Kit Setup or CY5672 Kit Only file, then double-click on the file and go to step 4.
3. Click **Install CY5672 PRoC BLE RDK**, shown in Figure 2-1, to start the kit installation.

Figure 2-1. Kit Installer Screen



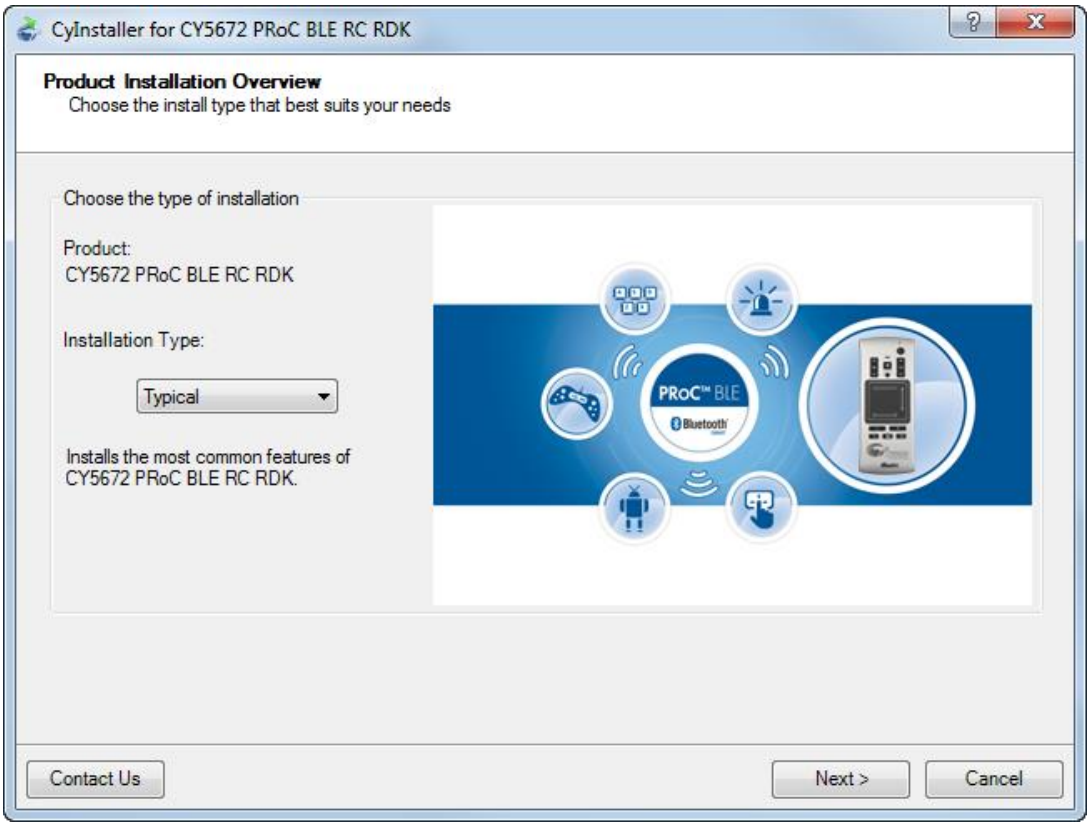
4. Select the folder in which you want to install the kit-related files by clicking **Change,** as shown in Figure 2-2. Choose the directory and click **Next.**

Figure 2-2. CY5672 PRoC BLE RDK – InstallShield Wizard



5. Choose the “Typical,” “Custom,” or “Complete” installation type in the **Product Installation Overview** window, as shown in Figure 2-3. Click **Next** after you select the installation type.

Figure 2-3. Product Installation Overview



6. Read the license agreement and select **I accept the terms in the license agreement** to continue with the installation. Click **Next**. When you click **Next**, the installer automatically installs the required software, if it is not present on your computer. Following are the required software:

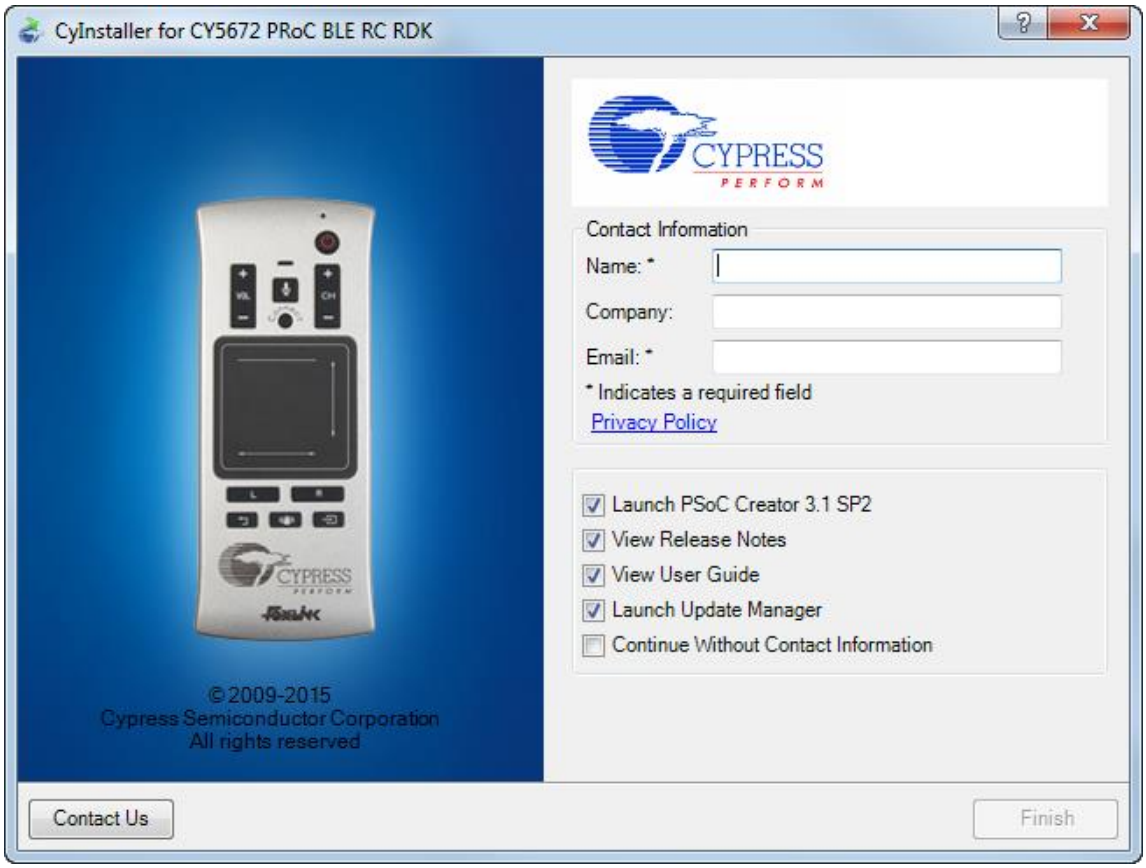
   - PSoC Creator 3.1 or later: Download from www.cypress.com/psoccreator.
   - PSoC Programmer 3.22 or later: Download from www.cypress.com/programmer.
   - CySmart 1.0 or later: Download from www.cypress.com/CySmart.

   Note: If you are using the CY5672 Kit Only package, you must download and install the software manually.

7. The installation begins by presenting the list of packages on the installation page. A green check mark appears next to each package after successful installation.

8. Enter your contact information or select the option **Continue Without Contact Information,** as shown in Figure 2-4. Click **Finish** to complete the kit installation.

Figure 2-4. Installer Asking for Contact Information



Note: The PSoC Creator version number displayed in the figure may vary depending upon the latest PSoC Creator packaged in the installer.

9. After the installation is complete, the kit contents are available at the following location:

   <Install_Directory>\CY5672 PRoC BLE RC RDK

   Default location:

   Windows 7 (64-bit):

   C:\Program Files (x86)\Cypress

   Windows 7 (32-bit):

   C:\Program Files\Cypress

   Note: For Windows 7/8/8.1 users, the installed files and the folder are read only.

2.3 Uninstall Software

The software can be uninstalled using one of the following methods:

- Go to **Start** > **All Programs** > **Cypress** > **Cypress Update Manager** > **Cypress Update Manager**; click the **Uninstall** button for the appropriate software package.
- Go to **Start** > **Control Panel** > **Programs and Features**; click the **Uninstall/Change** button for the appropriate software package.

3. Kit Operation

This chapter introduces the features of the CY5672 PRoC BLE Remote Control RDK that will be used as part of the kit operation. Topics include:

- Connecting the PRoC BLE Remote Control with the CySmart USB Dongle
- Connecting the PRoC BLE Remote Control with the CySmart USB Dongle to Communicate with the CySmart Software
- Programming and Debugging

3.1 Connecting the PRoC BLE Remote Control with the CySmart USB Dongle to Communicate with PC

The first use for the dongle is to allow the remote control to interact with a PC, which may not necessarily have the built-in Bluetooth Smart capability. In such a case, the dongle receives the BLE packets from the remote control, converts them to standard USB HID packets, and sends them to the PC.

In this document, “connecting” refers to establishing a Bluetooth Smart wireless link between the PRoC BLE remote control and the CySmart USB dongle. The process of establishing the connection for the first time involves “bonding,” which refers to storing encryption information from the remote control in the nonvolatile memory of the CySmart USB dongle. This stored information forms the “whitelist,” which is used by Bluetooth Smart and Bluetooth Smart Ready devices to scan and search for subsequent connections.

The PRoC BLE remote control is factory-bonded with the CySmart USB dongle. Therefore, when the dongle is plugged in, it begins to search for the bonded remote control (that is, remote control unit that is present in the dongle’s whitelist). After both AAA batteries are inserted, any user activity on the PRoC BLE remote control connects it to the dongle. Refer to step 1 in the Assembling and Disassembling the Remote Control section for instructions on removing the back cover to insert the two AAA batteries.

The remote control needs to be reconnected to the CySmart USB dongle whenever the remote control or the dongle is reprogrammed. The steps to reconnect the PRoC BLE remote control with the CySmart USB dongle are as follows.

Note: Making changes to the remote control firmware or CySmart USB dongle firmware that impact the connection establishment behavior may invalidate the following steps.

Ensure that the two AAA batteries provided with the kit are inserted into the battery holder at the back of the remote control before following these steps.

1. Insert the dongle into a USB port on the PC. The red power LED on the dongle glows to show that the dongle is powered on, and the green status LED on the dongle glows to show that enumeration is complete on the dongle side. The blue user LED on the dongle shows a slow breathing effect to indicate that the dongle is scanning for bonded devices.

   Notes:

   - The time taken to complete enumeration on the PC side for the first time can vary based on driver installation time.
   - The blue user LED on the CySmart USB dongle does not show a slow breathing effect if the dongle is reprogrammed. In this case, press the user button (SW2) on the CySmart USB dongle to start scanning. The blue LED shows the fast breathing effect.

2. Press the connect button on the remote control. The red LED on the remote control glows to show that the remote control is now in advertising mode. The remote control will try connecting to the CySmart USB dongle for a maximum interval of 30 seconds.

3. When a successful connection is established, the red LED on the remote control blinks three times and then switches off, while the blue user LED on the CySmart USB dongle glows continuously. When the connection is unsuccessful, the LED switches off after a 30-second timeout. In this event, repeat the sequence starting from step 2 to establish a connection.

   Note: Do not turn off the remote control while the red LED is blinking after a successful connection is established.

4. Place and move your finger on the trackpad surface. If the connection is successful, the mouse cursor on the PC will follow your finger movement.

3.2 Connecting the PRoC BLE Remote Control with the CySmart USB Dongle to Communicate with the CySmart Software

The second use for the dongle is as an interface to the CySmart software. CySmart is a Bluetooth LE host emulation tool for Windows PCs. This tool provides an easy-to-use GUI that enables you to test and debug your Bluetooth LE peripheral applications. It supports device discovery, connection establishment, and pairing and bonding. It also provides the ability to access the Generic Attribute Profile (GATT) server database for the PRoC BLE remote control.

When plugged into a USB port, the CySmart USB dongle enumerates, exposing the USB interfaces. Figure 3-1 and Figure 3-2 show the drivers for the CySmart USB dongle installation on a Windows 7 PC.

Figure 3-1. Driver Installation in Progress for CySmart USB Dongle on Windows 7

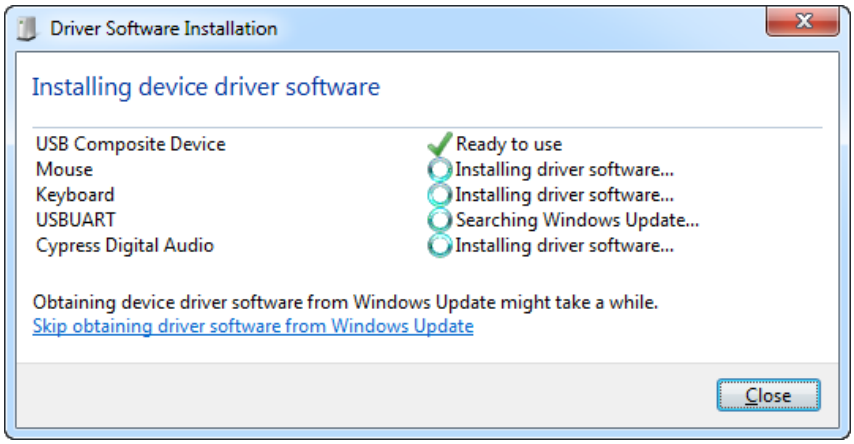
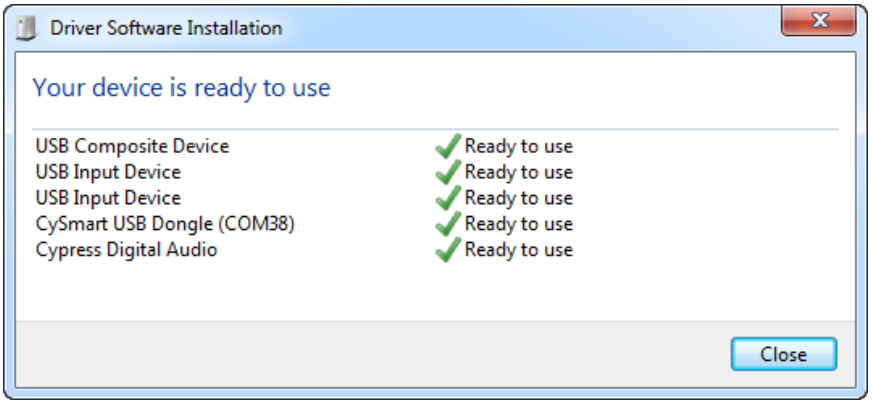


Figure 3-2. Driver Installation Complete for CySmart USB Dongle on Windows 7



Note: The appearance of the windows shown in and Figure 3-1 and Figure 3-2 may differ across operating systems and based on settings.

3.2.1 Connecting the CY5672 Remote Control RDK to CySmart Software

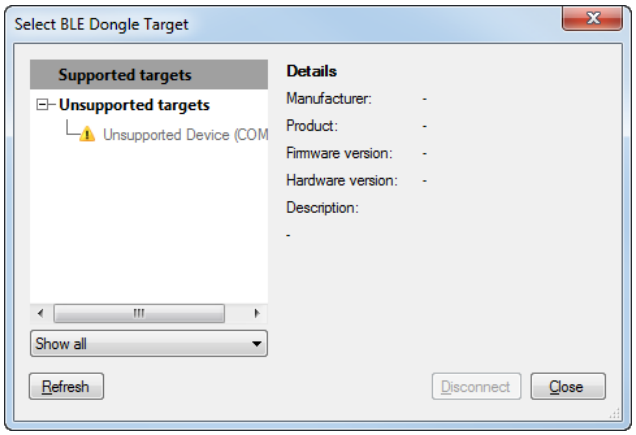
1. Plug the CySmart USB dongle into the PC, as shown in Figure 3-3.

Figure 3-3. CySmart USB Dongle Connected to Laptop



2. Open CySmart and click the **Select Dongle** button. A window appears, as shown in Figure 3-4.

Figure 3-4. BLE Dongle Target Window



3. Click the **Refresh** button and verify that the Cypress BLE HID dongle is listed, as shown in Figure 3-5. Click on **Cypress BLE HID Dongle (…)**, and click the **Connect** button. The CySmart tool appears as shown in Figure 3-6.

Figure 3-5. BLE Dongle Target Window

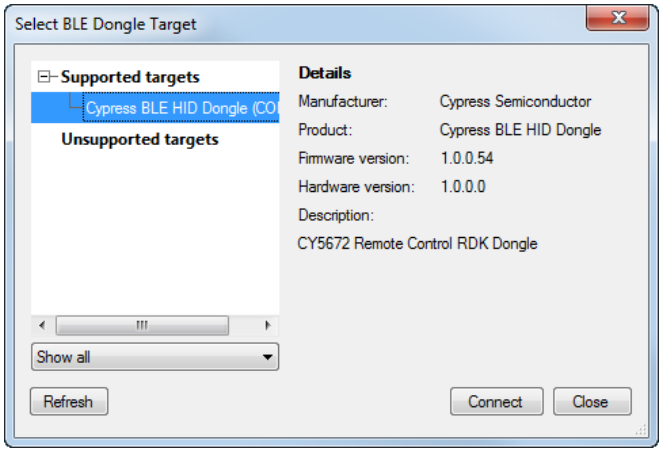
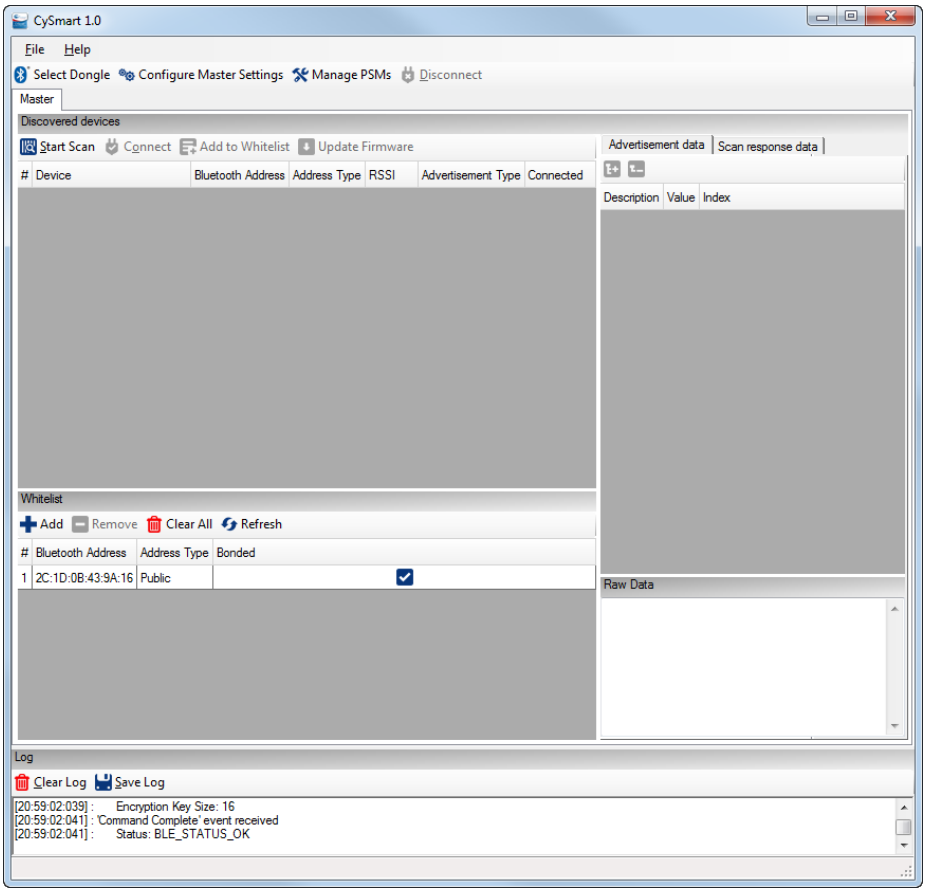


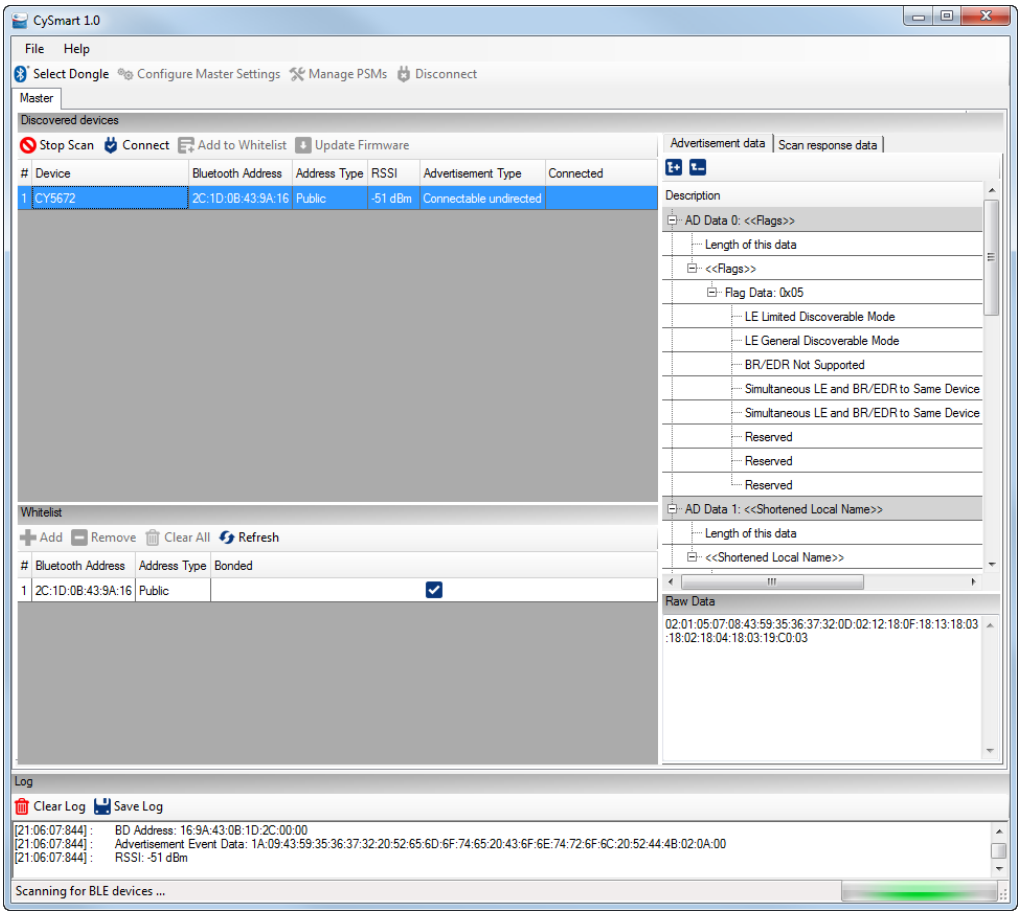
Figure 3-6. CySmart Window



Note: The Blue LED on the dongle turns OFF once the connection with CySmart tool is established. It will remain in the OFF state as long as the communication with the CySmart tool is active.

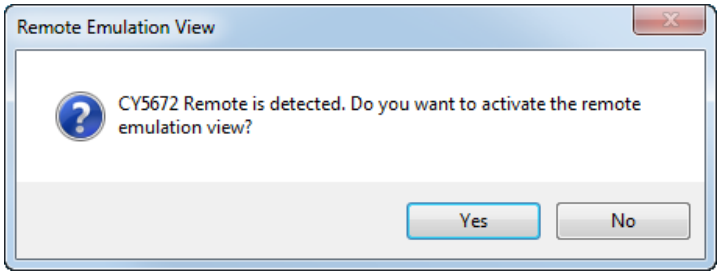
4. Insert two AAA batteries into the battery holder of the remote control.
5. Press the **Connect** button on the remote control. The red LED glows to indicate that the remote control is now in advertising mode.
6. Click the **Start Scan** button on the CySmart tool to start scanning for BLE devices that are advertising. Verify that "CY5672" is listed under **Discovered Devices,** as shown in Figure 3-7.

Figure 3-7. CySmart Discovered Devices Window



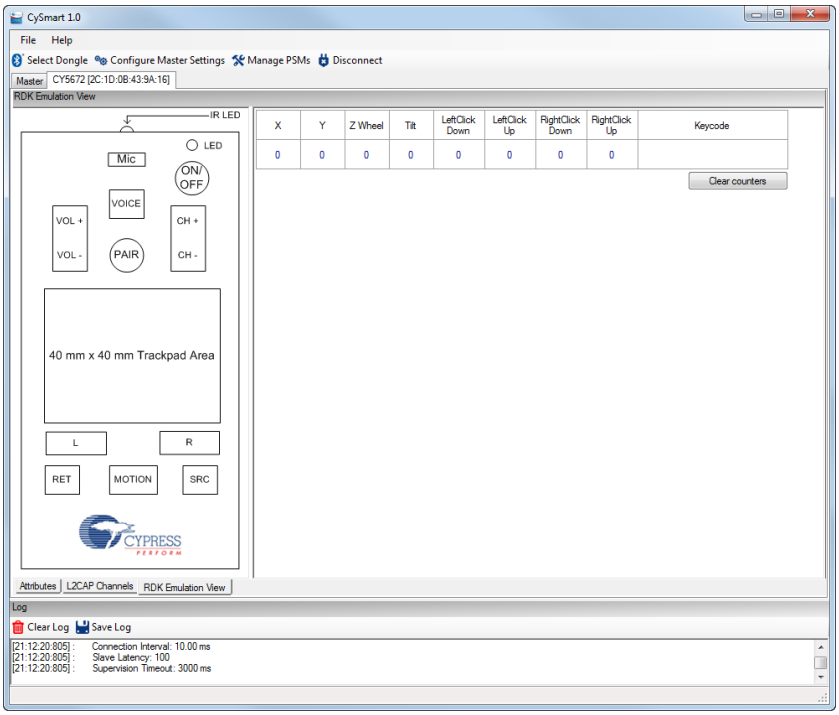
7. Double-click on “CY5672.” Click **Yes** in the dialog box to activate the remote emulator tab.

Figure 3-8. CySmart Remote Emulation View Dialog Box



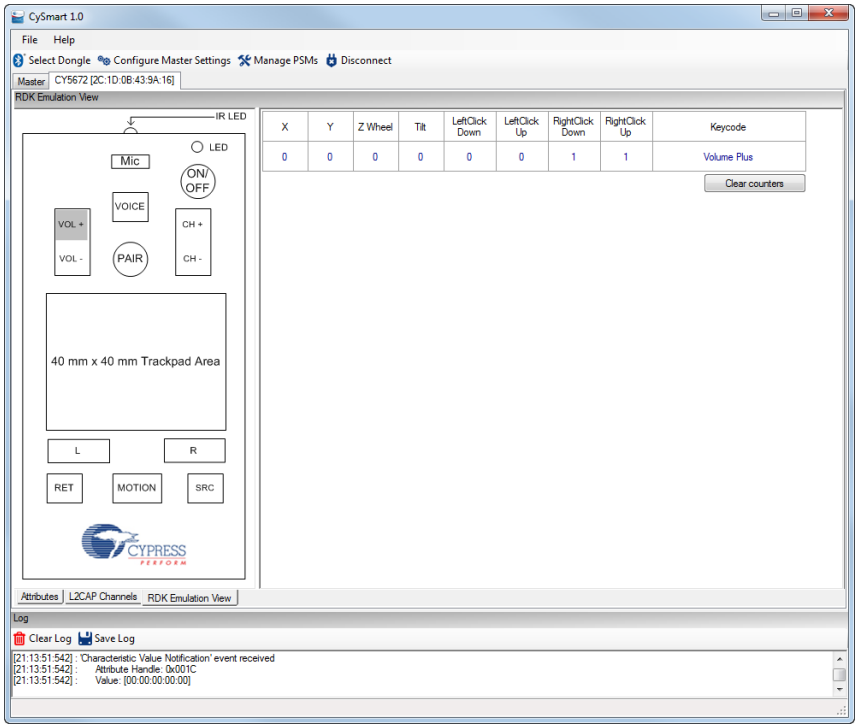
8. The remote emulator tab is displayed on the CySmart tool, as shown in Figure 3-9. This will take several seconds as the attributes are read from the remote. This tab allows you to test the features of the PRoC BLE remote control described in the PRoC BLE Remote Control Details section.

Figure 3-9. RDK Emulator in CySmart



9. Invoke the remote control features. The invoked features are highlighted on the emulator tab. For example, when you press VOL + on the remote control, the VOL + button on the remote control emulator is highlighted, as shown in Figure 3-10 as long as the VOL + button remains pressed on the remote control.

Figure 3-10. Testing Remote Control Features Using RDK Emulator



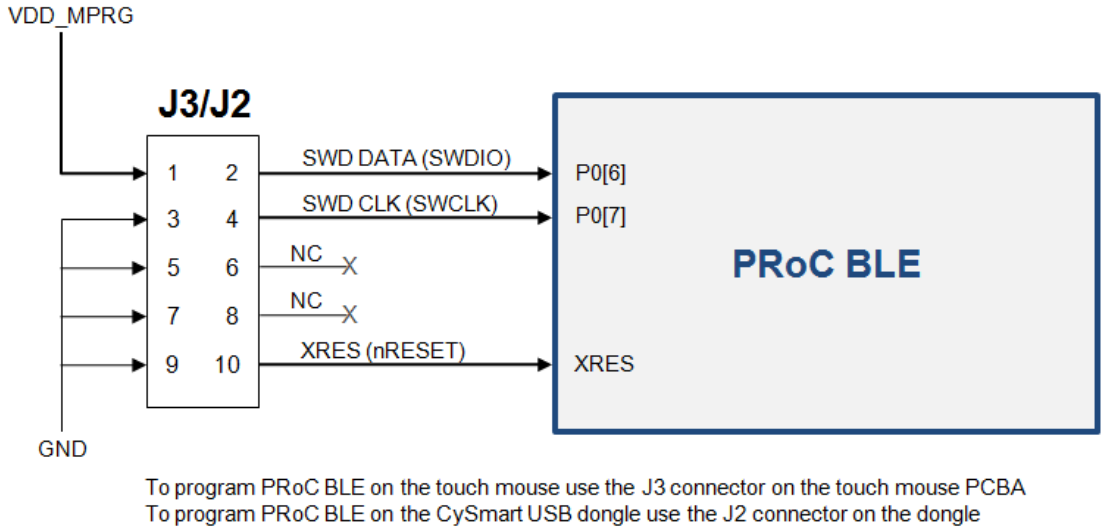
For more information on using the CySmart tool and the RDK Emulation View, download the guide for the CySmart tool from www.cypress.com/go/CySmartUserGuide.

3.3 Programming and Debugging

3.3.1 Programming and Debugging PRoC BLE on the Remote Control and Dongle

The PRoC BLE remote control and the CySmart USB dongle support programming and debugging using the MiniProg3 programmer/debugger provided with the kit. Both expose a 10-pin connector. Figure 3-11 is a block diagram showing the header and associated connections.

Figure 3-11. Programming/Debugging PRoC BLE



Notes:

- Remove the warning sticker before using MiniProg3.
- Before trying to program or debug the device, make sure that PSoC Creator and PSoC Programmer are installed on the PC.

You can program the remote control either with PSoC Programmer or directly from PSoC Creator.

3.3.2 Programming PRoC BLE on the Remote Control and Dongle Using PSoC Programmer

To program the remote control using PSoC Programmer, follow these steps.

1. Connect MiniProg3 to the PC using the USB A to mini-B cable provided with the kit. When it is properly connected, the four LEDs on the MiniProg3 turn on for a few seconds.

2. Connect one end of the 10-pin ribbon cable, provided with the kit, to the 10-pin header on the MiniProg3.

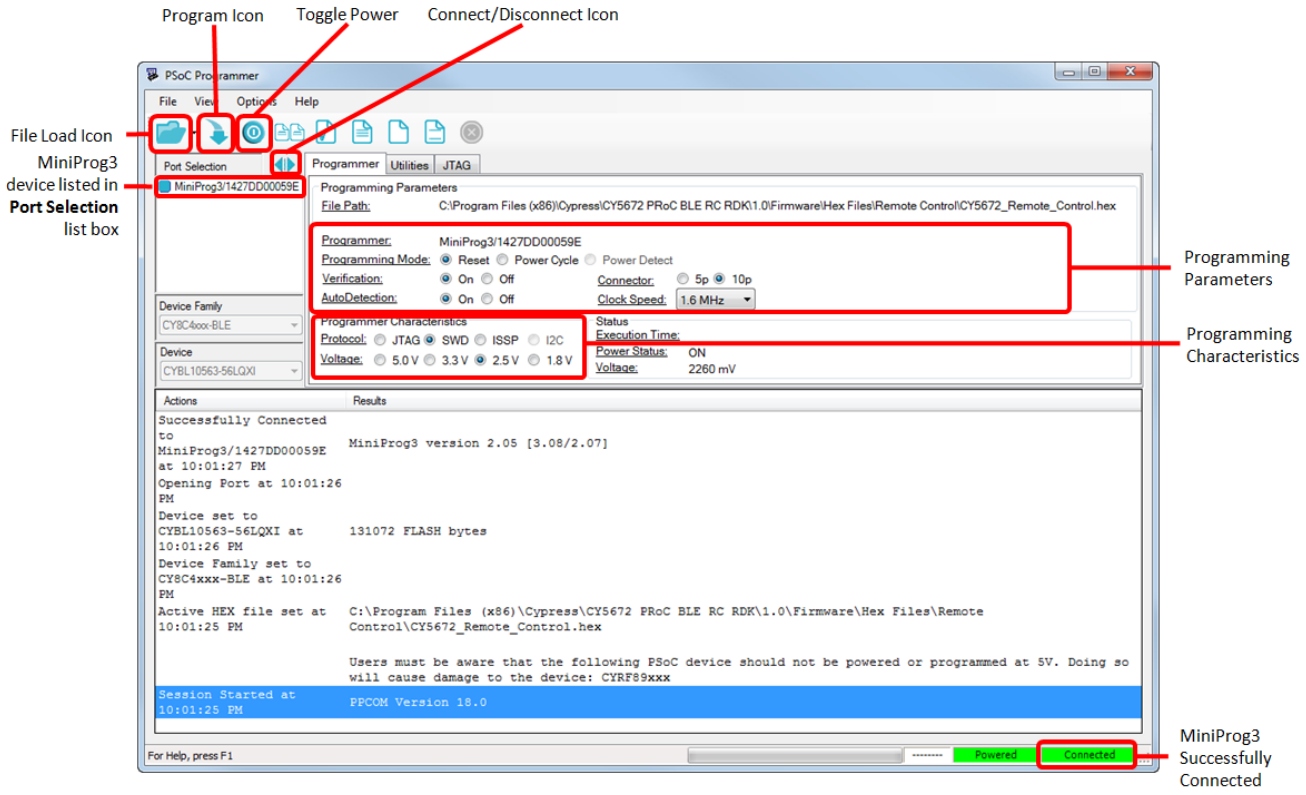
3. Connect the other end of the 10-pin ribbon cable to the 10-pin connector (J3 on the remote control or J2 on the CySmart USB dongle). Programming headers on both the remote control and the CySmart USB dongle (J3 and J2 respectively) are polarized. Do not apply excessive force; instead, ensure that the connectors are plugged in the correct orientation.

4. Run PSoC Programmer by choosing **Start** > **All Programs** > **Cypress** > **PSoC Programmer**.

   Note: Ensure that only one instance of PSoC Programmer is running on the PC.

5. To establish the connection with a programmer, select either the “MiniProg3” device from the **Port Selection** list box or the **Connect/Disconnect** icon, as shown in Figure 3-12.

Figure 3-12. PSoC Programmer



6. If the connection is successful, the green status LED on MiniProg3 lights up. A blue square with rounded corners appears next to “MiniProg3” in the **Port Selection** list box. Also, the status in the lower right corner of the PSoC Programmer window turns green and displays **Connected**.
7. Ensure that the settings under **Programming Parameters** in the **Programmer** tab are set to the values specified in Table 3-1.

Table 3-1. Programming Parameters

Programming Parameter	Setting
Programming Mode	Remote Control: Reset (only if batteries are inserted) or Power Cycle Dongle: Reset (if plugged into and powered via USB port) or Power Cycle (when unplugged from USB port)
Verification	On
AutoDetection	On
Connector	10p
Clock Speed	1.6 MHz

Note: Reset mode can be used even when batteries are not inserted if the **Toggle Power** button shown in Figure 3-12 is pressed before step 9.

8. Ensure that the parameters under **Programmer Characteristics** are set to the values specified in Table 3-2.

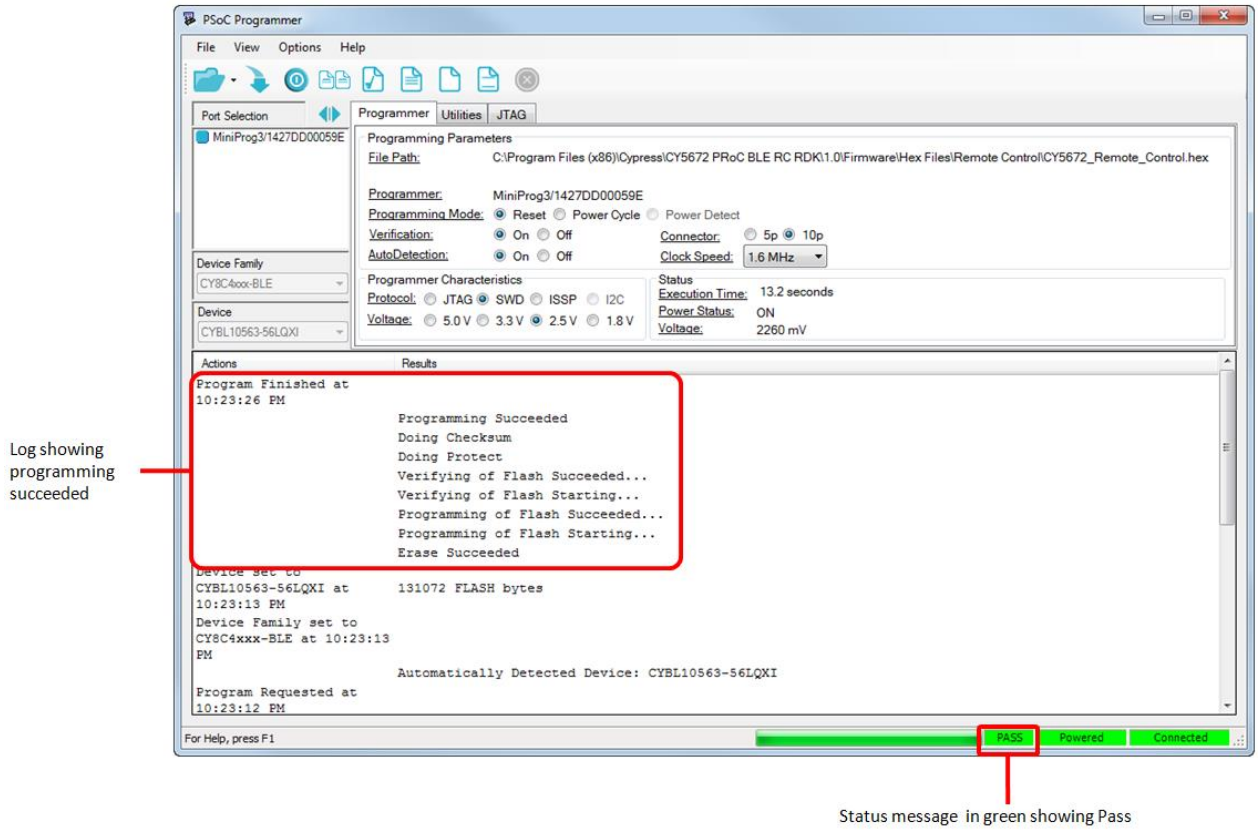
Table 3-2. Programmer Characteristics

Programmer Characteristics	Setting
Protocol	SWD (Serial Wire Debug)
Voltage	2.5 V

Note: The value of 5 V for **Voltage** in the **Programmer Characteristics** section is not supported, as it is beyond the operating voltage of the remote control. The remote control contains an overvoltage protection circuit to prevent damage to the kit if 5 V is selected.

9. Click on the **File Load** icon. A dialog box appears. Browse to the location of the hex file to be programmed and select the hex file. The selected file appears as the "File Path" in the **Programmer** tab. The hex file for the firmware provided with the kit is as follows:
   - PRoC BLE remote control: *<Install_Directory>\CY5672 PRoC BLE RC RDK\1.0\Firmware\Hex Files\Remote Control\CY5672_Remote_Control.hex*
   - CySmart USB dongle: *<Install_Directory>\CY5672 PRoC BLE RC RDK\1.0\Firmware\Hex Files\Dongle\BLE_HID_CySmart_Dongle.hex*
10. Click the **Program** icon to program the PRoC BLE device on the remote control or the CySmart USB dongle with the selected hex file. After successful programming, the "Programming Succeeded" message will be displayed in the log area, as shown in Figure 3-13. Also the status in the lower right corner of the PSoC Programmer window turns green and shows **PASS**.

Figure 3-13. Programmer Window after Successful Programming

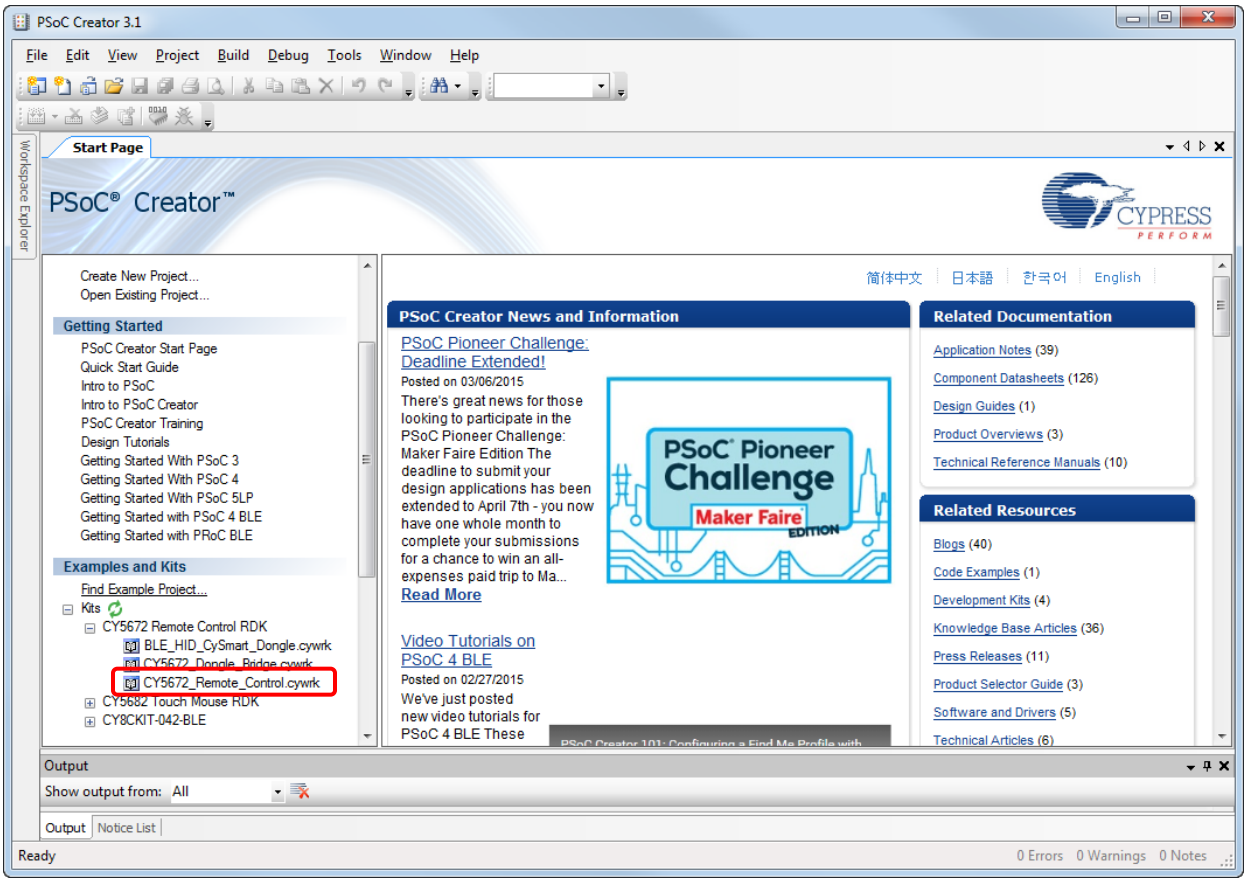


3.3.3 Programming and Debugging PRoC BLE on Remote Control and Dongle Using PSoC Creator

To program and debug PRoC BLE on the remote control or CySmart USB dongle using PSoC Creator, follow these steps.

1. Run PSoC Creator by choosing **Start** > **All Programs** > **Cypress** > **PSoC Creator**.
2. Go to the Start page, expand **Kits** > **CY5672 Remote Control RDK**, and then click on the project that you want to open.

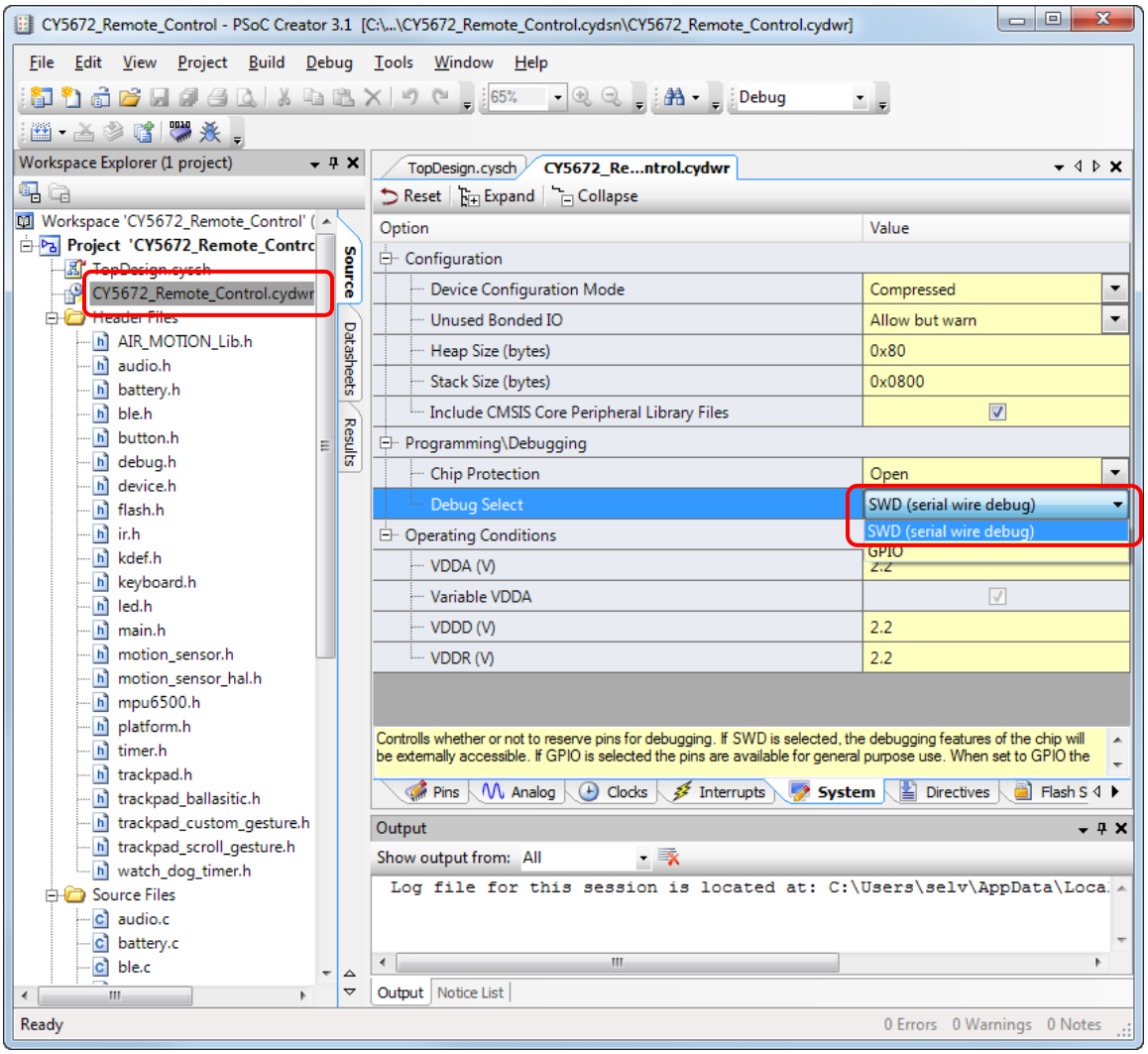
Figure 3-14. Opening a project in PSoC Creator



This allows you to save an editable copy of the project to the location of your choice. After the project is saved it is opened by the PSoC Creator for editing.

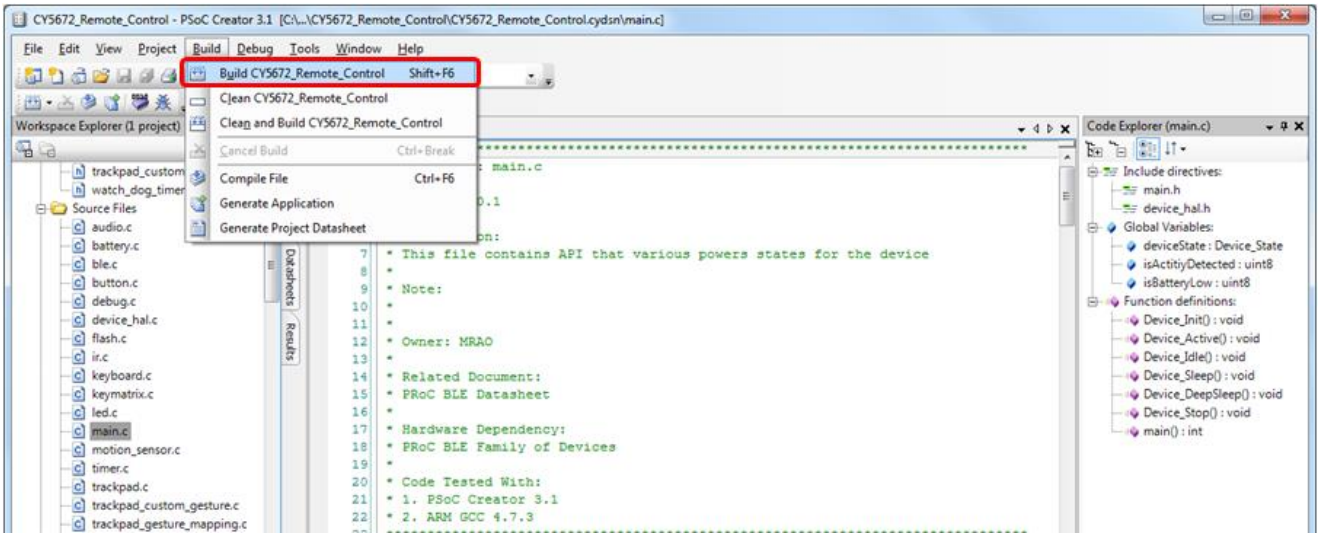
3. Note that the debug option is disabled by default in the remote control firmware to minimize power consumption. You can enable the debug option using the following procedure:
   a. Open *CY5672_Remote_Control.cydwr* from the Workspace Explorer, which is located by default along the left-hand side of the PSoC Creator window.
   b. Click the **System** tab.
   c. Choose the “SWD” option under **Debug Select,** as shown in Figure 3-15.

Figure 3-15. Enabling Debug Option in PSoC Creator



4. Build the project by choosing **Build** > **Build <*project name*>**, as shown in Figure 3-16.

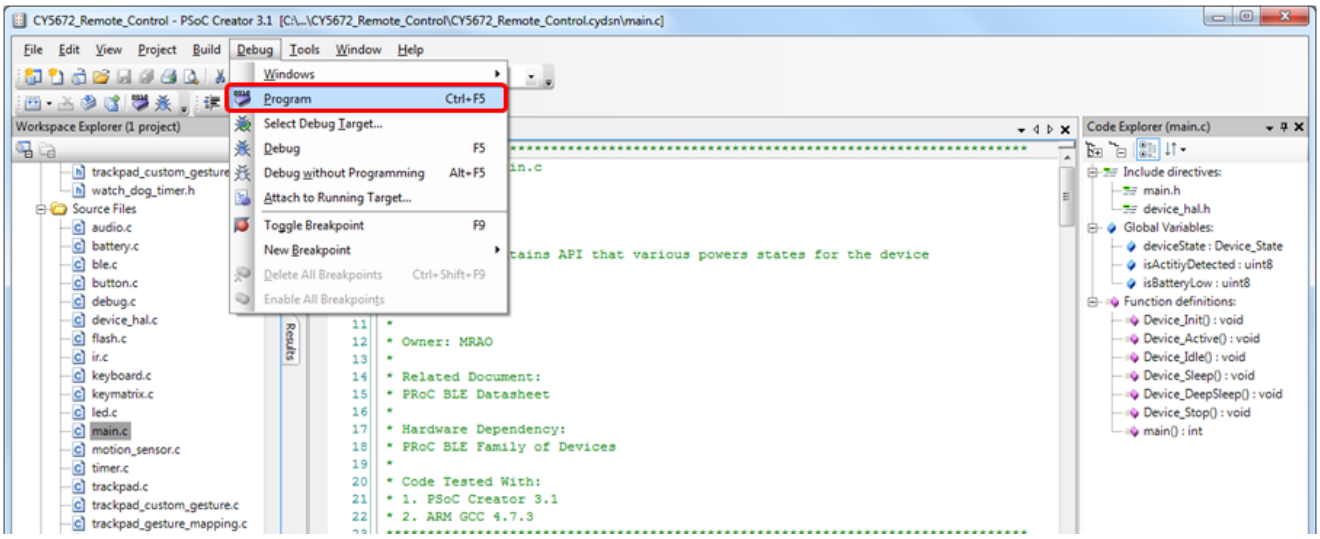
Figure 3-16. Building the Project in PSoC Creator



The **Output** window in PSoC Creator and the status bar display any errors encountered during the build process. Ensure that the project builds without errors.

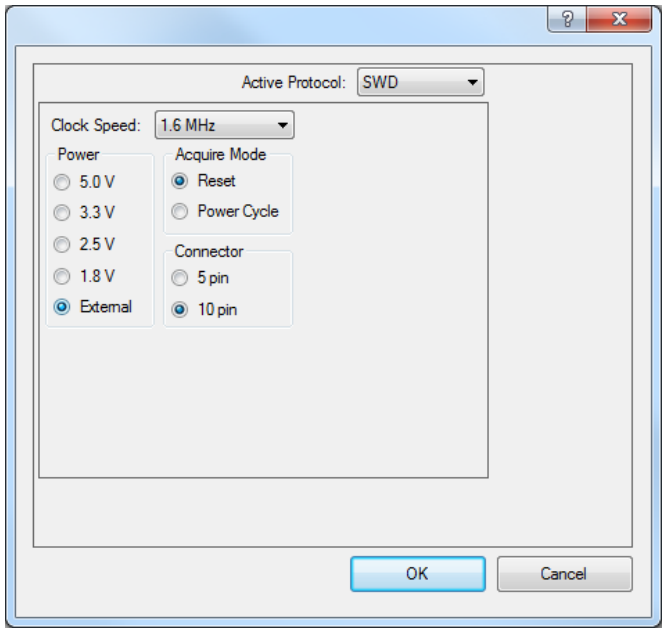
5. Connect the MiniProg3 to the PC using the USB A to mini-B cable provided with the kit. When it is properly connected, the four LEDs on the MiniProg3 turn on for a few seconds.
6. Connect one end of the 10-pin ribbon cable, provided with the kit, to the 10-pin header on the MiniProg3.
7. Connect the other end of the 10-pin ribbon cable to the 10-pin connector on the remote control (J3) or on the CySmart USB dongle (J2).
8. Start programming by choosing **Debug** > **Program,** as shown in Figure 3-17. This will bring up the **Select Debug Target** window.

Figure 3-17. Programming via PSoC Creator



9. In the **Select Debug Target** window, click the **Port Setting** button and set the configuration options, as shown in Figure 3-18.

Figure 3-18. Port Settings

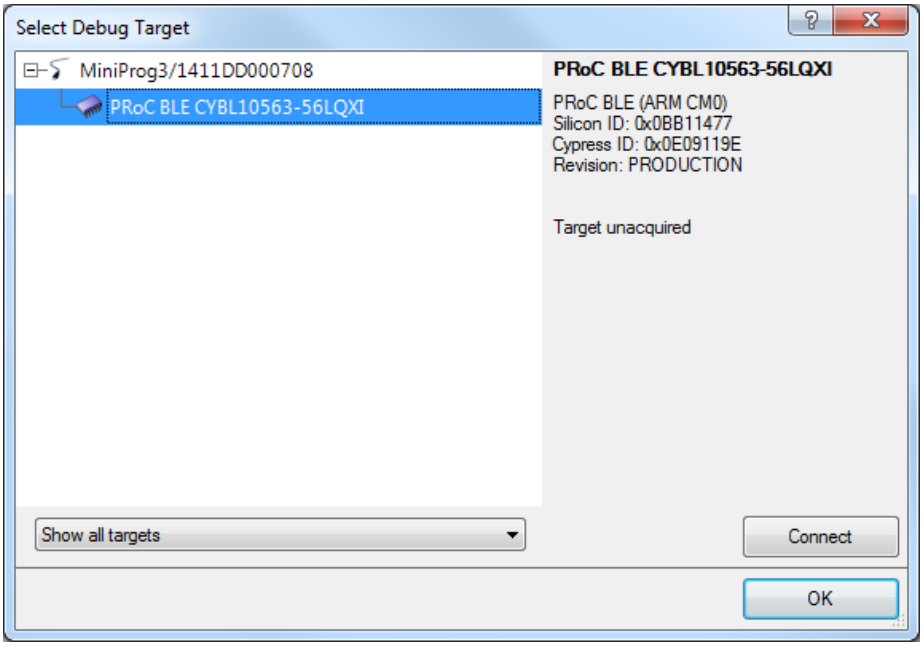


Notes:

- **Acquire Mode** can be set to “Reset” only if batteries are inserted. Otherwise, set it to “Power Cycle” and choose the Power setting as 1.8 V, 2.5 V, or 3.3 V.
- The **External** selection for the **Power** setting works only if batteries are inserted, regardless of the **Acquire Mode** selection.

10. In the **Select Debug Target** window, select the PRoC BLE device and click **Connect,** as shown in Figure 3-19. Then click **OK**.

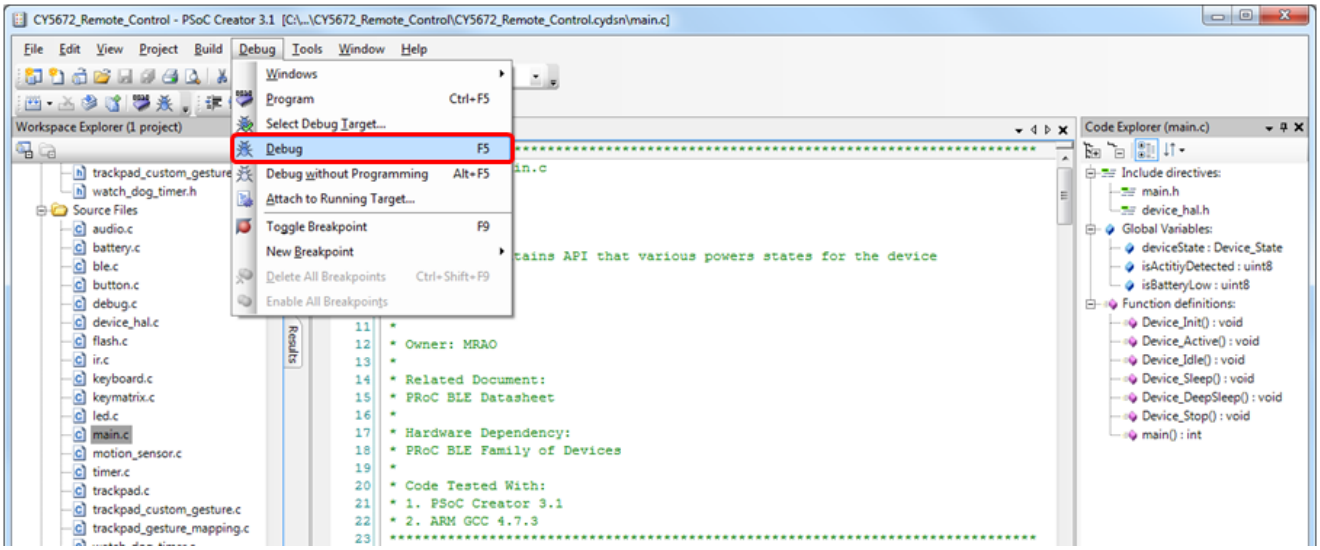
Figure 3-19. Select Debug Target



11. Check the **Output** window and the status bar to verify if programming is completed successfully.

12. If you need to debug, choose **Debug** > **Debug,** as shown in Figure 3-20, or press [**F5**] in PSoC Creator. In debug mode, you can set the breakpoints and step through the code as required. Note that you must enable debugging in the project for this to work, as shown in Figure 3-15.

Figure 3-20. Debugging via PSoC Creator



Detailed PSoC Creator tutorials are available at: www.cypress.com/training.

3.3.4 Programming PSoC 5LP on the CySmart USB Dongle

The PSoC 5LP controller on the CySmart USB dongle acts as a UART-to-USB bridge, which transfers the data received from PRoC BLE over UART to the USB interfaces listed in Table 3-3. The PSoC 5LP controller provides USB bootloader support to enable users to change the firmware.

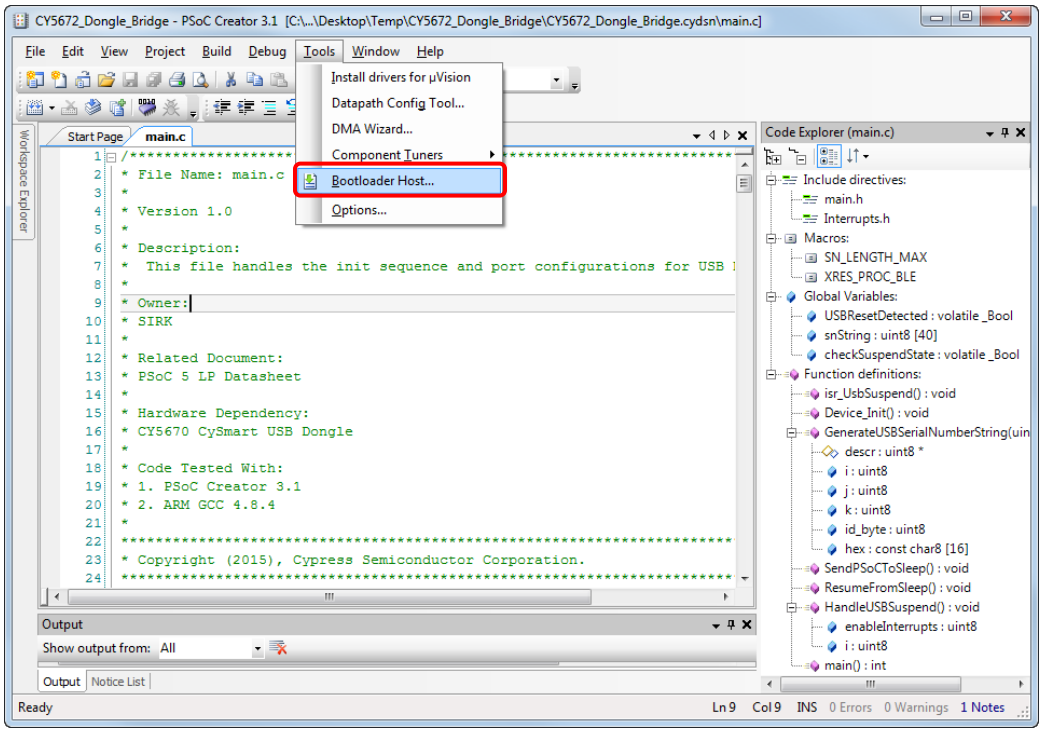
Table 3-3. Exposed CySmart USB Dongle Interfaces

Interface	Description
USB Composite Device	Generic USB device that supports multiple interfaces
USBUART (COM Port)	USB-UART bridge to work with CySmart tool
Mouse	Standard mouse device for generic mouse functionality
Keyboard	Keyboard device for Windows based shortcuts
Cypress Digital Audio	Microphone to stream voice data

Follow these steps to download the bootloadable project onto the PSoC 5LP device.

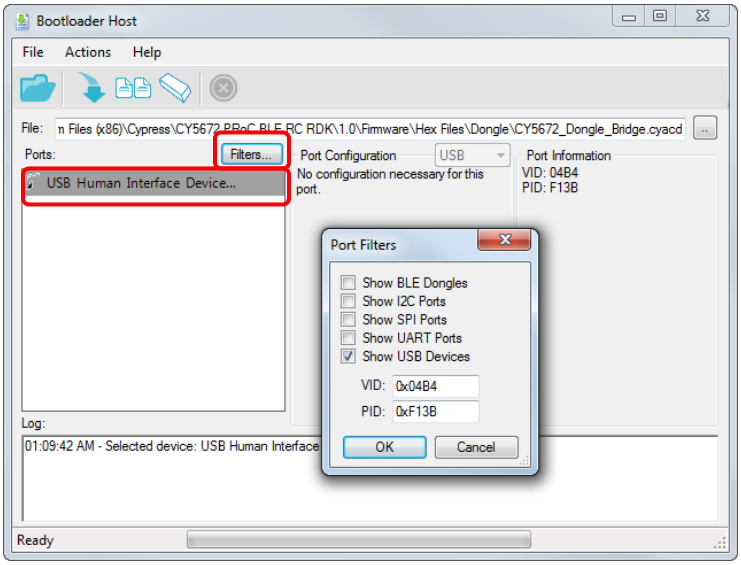
1. Keep the reset switch (SW1) pressed and insert the CySmart USB dongle into a USB port on the PC. If the switch is pressed for more than 100 ms, the PSoC 5LP on the dongle enters bootloader mode. This is indicated by a blinking green LED on the dongle.
2. Open the Bootloader Host tool from PSoC Creator by choosing **Tools** > **Bootloader Host,** as shown in Figure 3-21.

Figure 3-21. Open Bootloader Host Tool from PSoC Creator



3. In the Bootloader Host tool, click **Filters** and add a filter to identify the USB device. Set **VID** as “0x04B4” and **PID** as “0xF13B”, and then click **OK,** as shown in Figure 3-22. Click on “USB Human Interface Device” under **Ports**.

Figure 3-22. Port Filters Tab in Bootloader Host Tool



4. In the Bootloader Host tool, click the **Open File** button (shown in Figure 3-23) to browse to the location of the bootloadable file *(*.cyacd*), as shown in Figure 3-24. The bootloadable file for the firmware provided with the kit is *<Install_Directory>\CY5672 PRoC BLE RC RDK\1.0\Firmware\Hex Files\Dongle\CY5672_Dongle_Bridge.cyacd*

   Note: The **.cyacd* file is generated on building the project for PSoC 5LP, provided as part of the kit installable. This project is *<Install_Directory>\CY5672 PRoC BLE RC RDK\1.0\Firmware\Dongle\CY5672_Dongle_Bridge\CY5672_Dongle_Bridge.cywrk.*

Figure 3-23. Open/Program Bootloadable File from Bootloader Host Tool

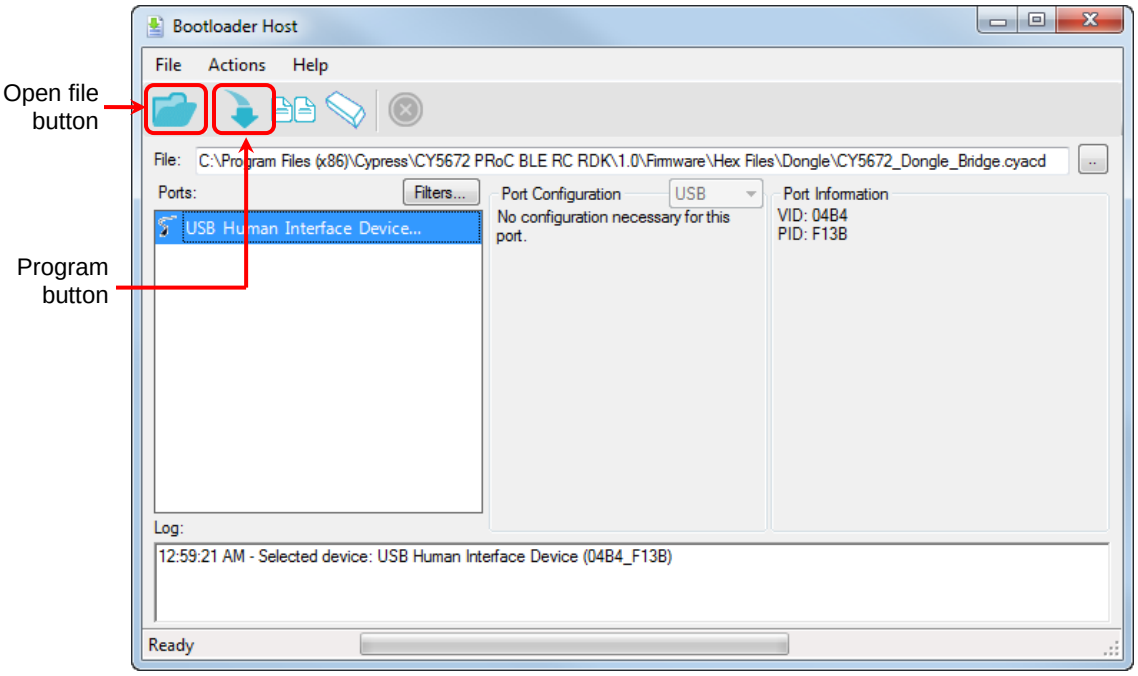
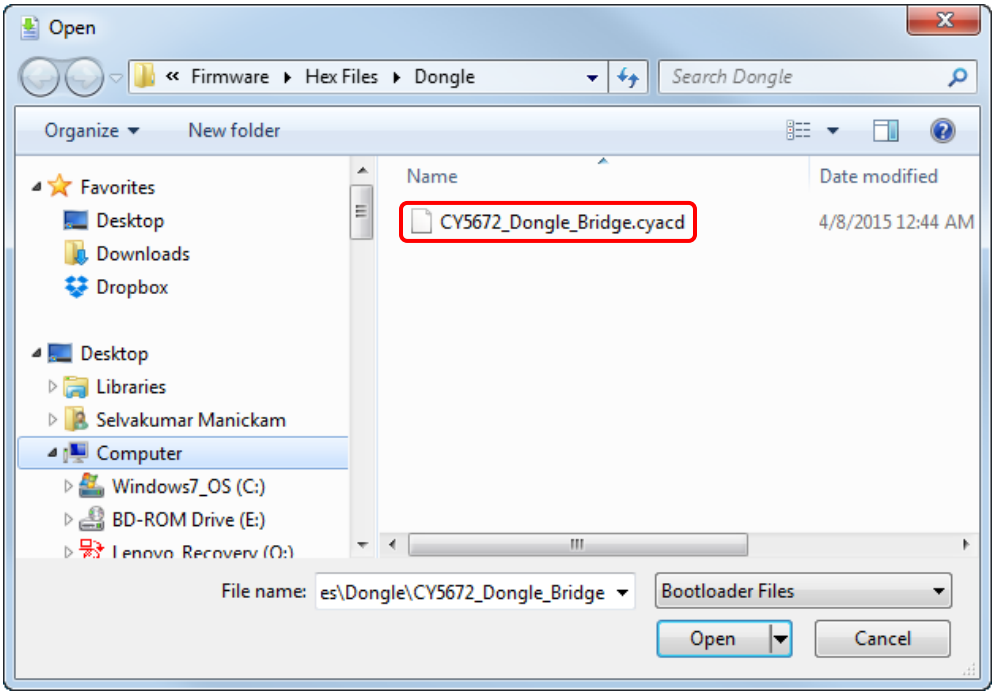


Figure 3-24. Select Bootloadable File from Bootloader Host



5. Click the **Program** button in the Bootloader Host tool to program the device, as shown in Figure 3-23.
6. If the bootload is successful, the log of the tool displays “Successful”; otherwise, it displays “Failed” with a statement describing the failure.

For additional information on bootloaders, refer to Cypress application note AN73503 – *USB HID Bootloader for PSoC 3 and PSoC 5LP.*

4. Hardware

This chapter describes the CY5672 RDK hardware such as the PRoC BLE remote control, the CySmart USB dongle, and its constituent blocks.

Note: “Module” in this chapter refers to a self-contained assembly of electronics components and circuitry, for example, the trackpad module.

4.1 Board Details

The PRoC BLE remote control has two boards connected via an 18-pin connector:

- Main board
- Trackpad module

The main board consists of the following components:

- CYBL10563-56LQXI PRoC BLE controller
- Buck-boost converter with two AAA batteries in series
- IR LED for sending commands
- Overvoltage protection circuit
- 10-pin SWD header for programming and debugging the PRoC BLE controller
- Onboard antenna for the BLE
- One red LED to indicate system status
- MEMS microphone for voice input
- Inertial motion sensor
- 3 × 3 button matrix + 3 additional buttons
- Battery monitoring circuit
- 18-pin male connector to connect to the trackpad module
- 24-MHz and 32.768-kHz crystals
- Wiggle antenna and matching network

The trackpad module board consists of the following key components:

- Trackpad sensors
- 18-pin female connector to connect to the main board

Top and bottom views of the main board and the trackpad module, with several key components marked, are shown in Figure 4-1 and Figure 4-2 respectively.

Figure 4-1. CY5672 PRoC BLE Remote Control Main Board and Trackpad Module Top View

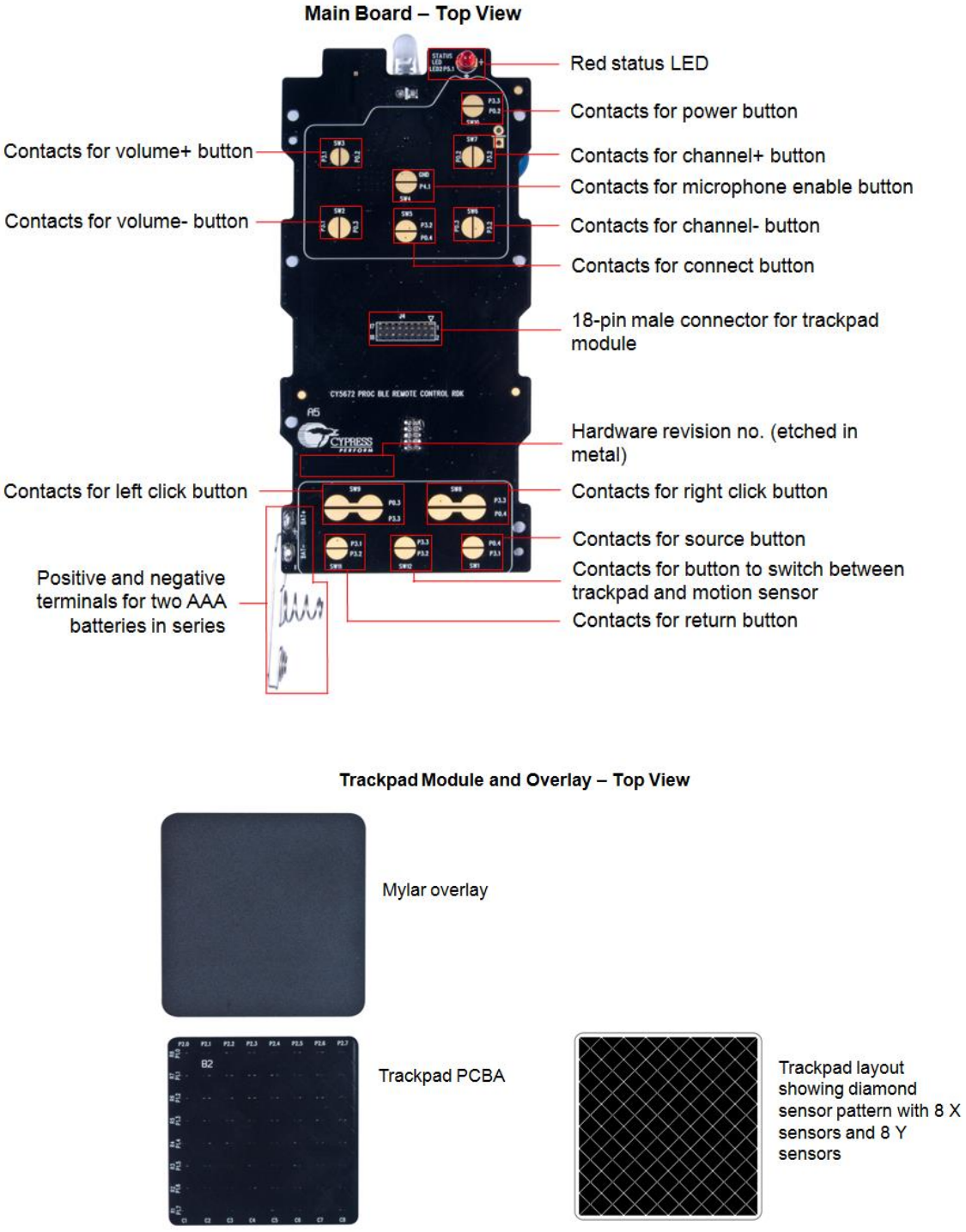
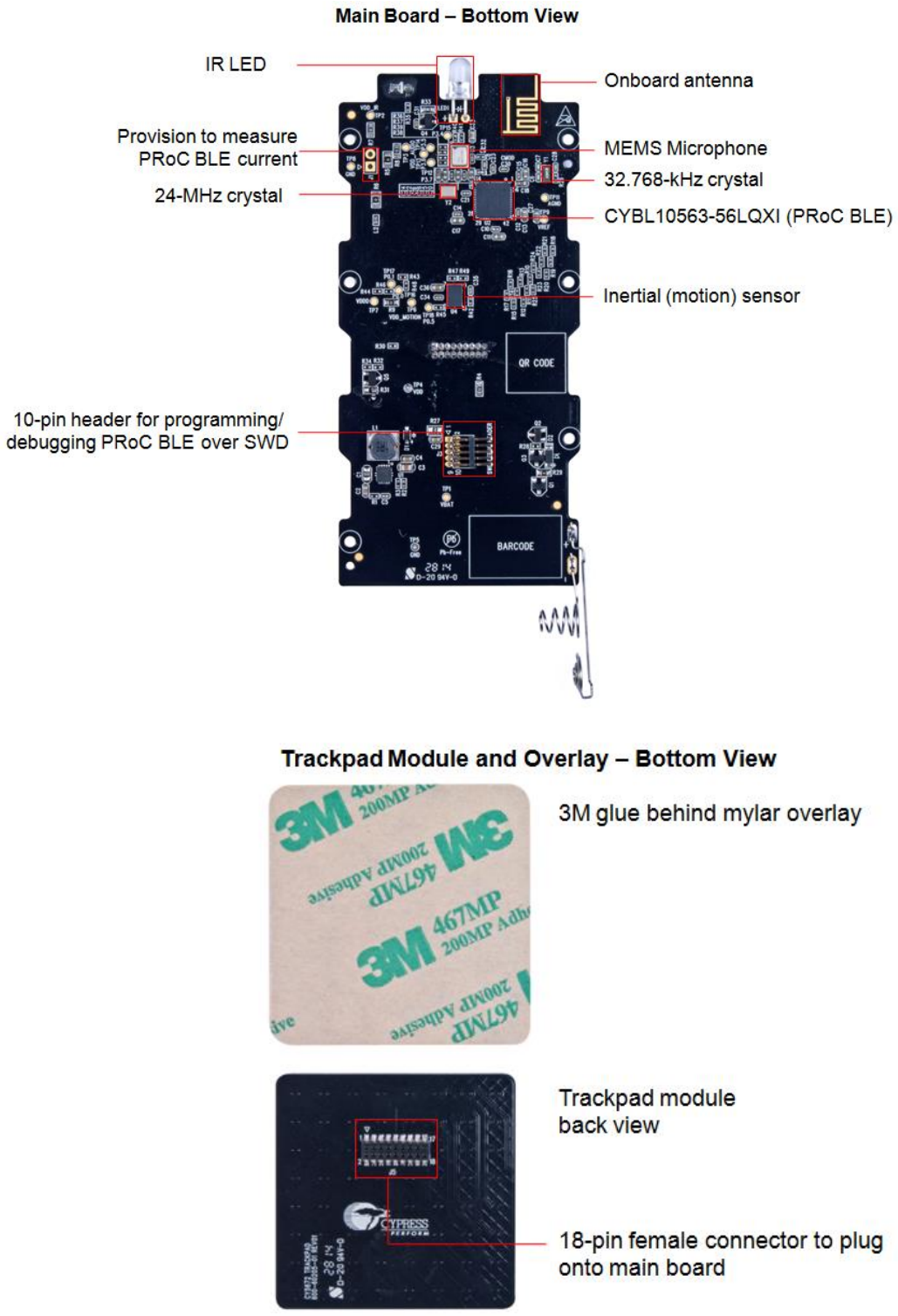


Figure 4-2. CY5672 PRoC BLE Remote Control Main Board and Trackpad Module Bottom View

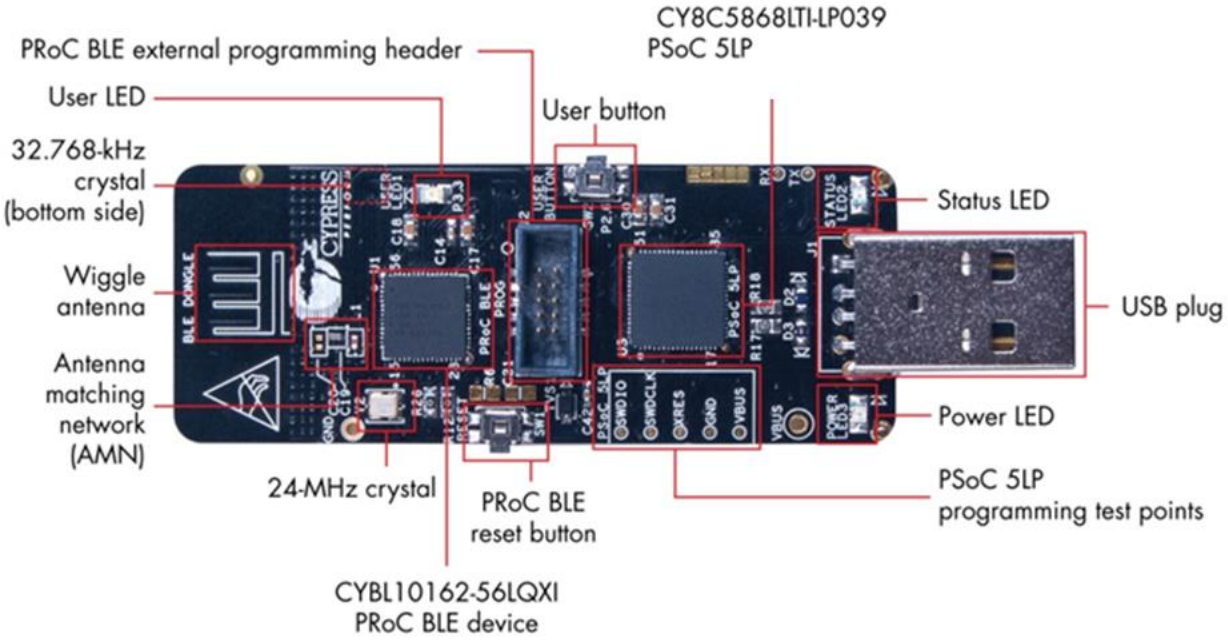


The CySmart USB dongle board consists of the following key components, as shown in Figure 4-3:

- CYBL10162-56LQXI PRoC BLE controller
- Hardware reset and button switch

- CY8C5868LTI-LP039 PSoC 5LP controller
- 10-pin header for programming and debugging PRoC BLE using SWD
- Onboard antenna for BLE
- USB 2.0 connector
- User and Reset buttons
- LEDs (User, Status & Power)
- 24-MHz and 32.768-kHz crystals
- Power and decoupling capacitor circuit
- Wiggle antenna and matching network

Figure 4-3. CySmart USB Dongle Description



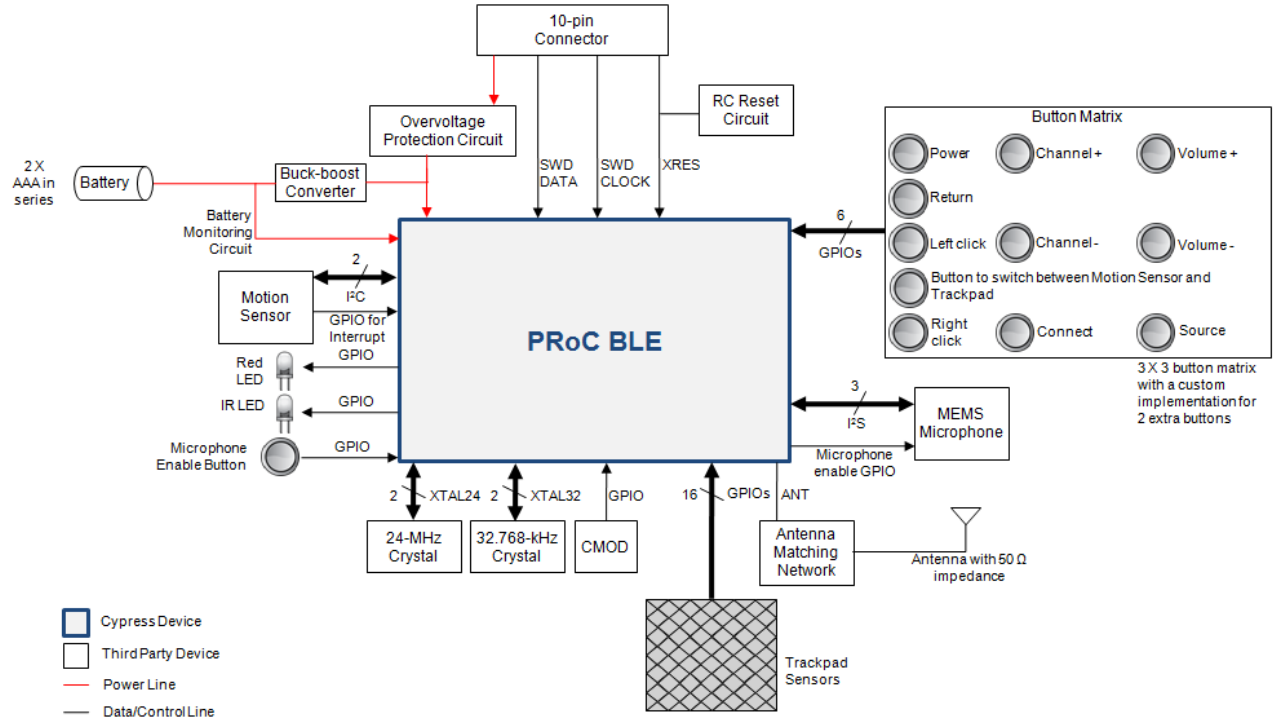
4.2 Theory of Operation

The CY5672 RDK contains a PRoC BLE-based remote control device and a PRoC BLE-based CySmart USB dongle.

4.2.1 Overview of the PRoC BLE Remote Control

Figure 4-4 shows the block diagram for the PRoC BLE remote control (including the PRoC BLE remote control main board and trackpad module).

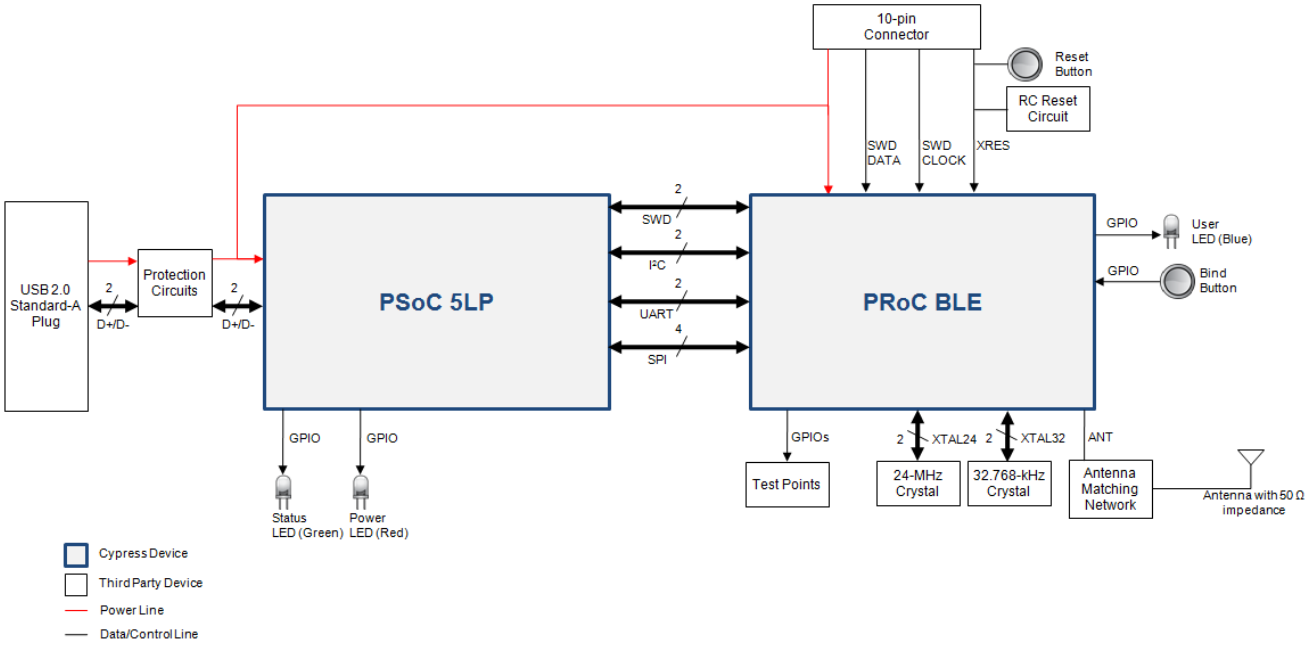
Figure 4-4. Block Diagram of PRoC BLE Remote Control



4.2.2 Overview of CySmart Dongle

The CySmart USB dongle is a Bluetooth Smart-to-USB bridge for USB-enabled devices (PCs, smart TVs, tablets, and so on). The dongle includes a PRoC BLE device, which is programmed to receive data from the remote control over a Bluetooth Smart link. It also contains a PSoC 5LP device, which functions as a serial-to-USB bridge by transferring the data received from PRoC BLE over UART to the PC over USB, as shown in Figure 4-5.

Figure 4-5. CySmart Dongle Block Diagram



The dongle is based on the CYBL10162-56LQXI PRoC BLE controller. It is intended to make it as easy as possible for Cypress customers/distributors to get started with Cypress's PRoC BLE-based reference designs (CY5672 PRoC BLE

Remote Control RDK and CY5682 PRoC BLE Touch Mouse RDK). The dongle can be connected to any device like a PC or a smart TV that supports HID over USB. Note that the PRoC BLE remote control can also be directly connected to any Bluetooth Smart Ready device; see the Connecting PRoC the BLE Remote Control with a Bluetooth Smart Ready Device section for details.

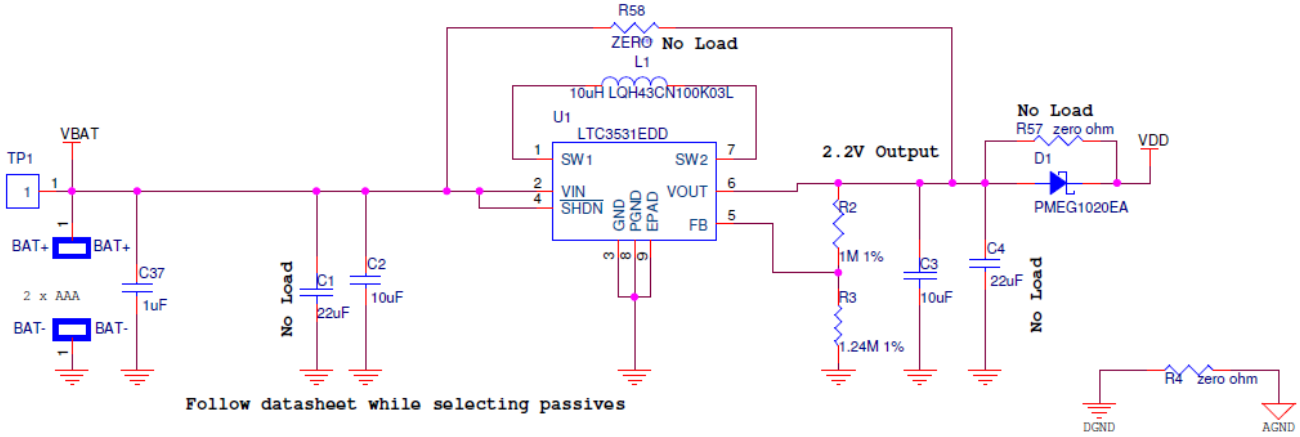
The CySmart dongle has a USB A-type plug to connect the PSoC 5LP device to the USB port of the host PC. The PSoC 5LP device then communicates with the PRoC BLE device over UART. The firmware provided with the kit installer uses the UART interface. The dongle has an onboard printed wiggle antenna. It also has a user LED, a user button, and a reset button for the PRoC BLE device. The CySmart USB dongle is powered directly through the USB port (VBUS) at 5.0 V. PSoC 5LP enables bootloading over USB to change the firmware for PSoC 5LP. See the Programming PSoC 5LP on the CySmart USB Dongle section for details on bootloading the PSoC 5LP.

4.3 Functional Description – PRoC BLE Remote Control

4.3.1 Power Supply

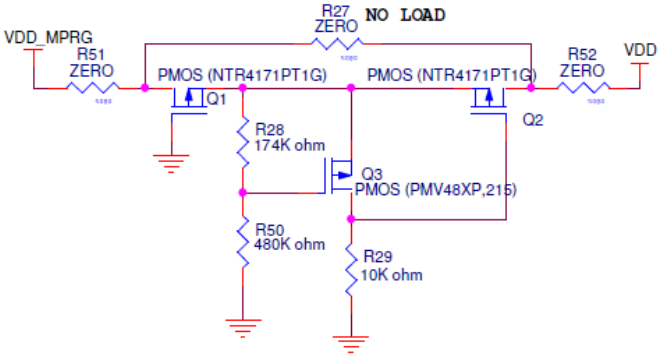
The remote control is powered using two AAA batteries connected in series to provide a maximum output voltage of 3 V. Figure 4-6 shows the DC-DC (buck-boost) converter circuit that regulates the output voltage at 2.2 V. This voltage is further dropped to 2.1 V because of the reverse protection diode, D1. The entire remote control circuit operates at 2.1 V.

Figure 4-6. Power Supply Circuit



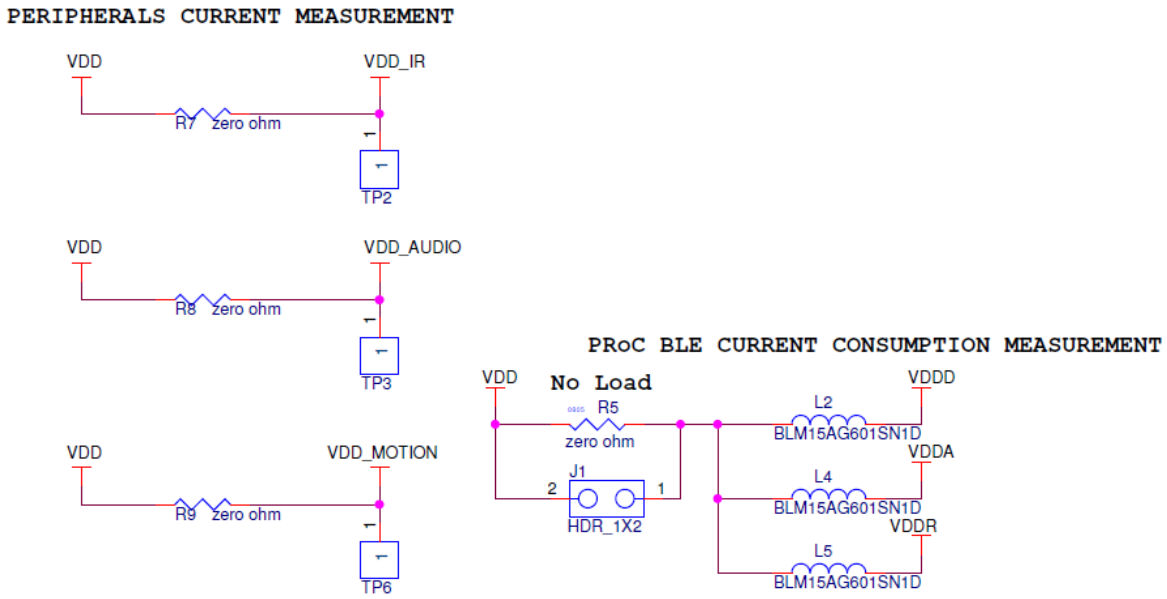
The overvoltage protection circuit shown in Figure 4-7 ensures that using MiniProg3 at 5 V for programming does not damage the system components.

Figure 4-7. Overvoltage Protection Circuit



The current consumption for the PRoC BLE device, IR LED, audio device, and motion sensor can be measured by using the circuits shown in Figure 4-8. For more information on how to measure the current consumption, refer to the Current Measurement section.

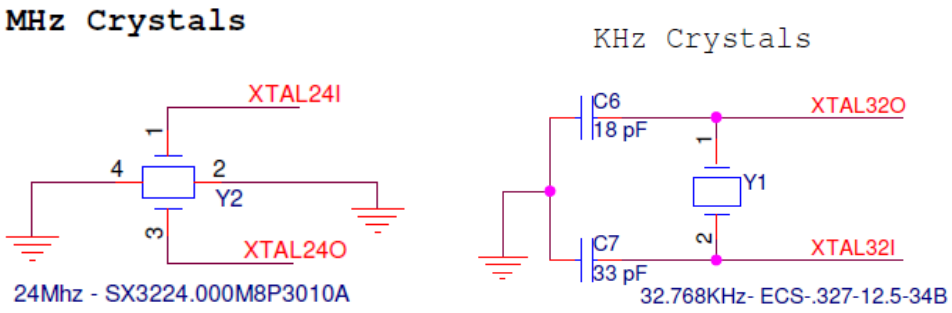
Figure 4-8. Current Measurement Circuits



4.3.2 Clock and Reset

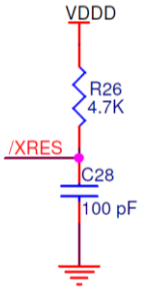
The PRoC BLE device requires two crystal oscillators, one running at 24 MHz and the other at 32.768 kHz, for its operation, as shown in Figure 4-9. Both oscillators are in the Pierce configuration. The 24-MHz oscillator is used by the Bluetooth Smart radio in the Active state, whereas the 32.768-kHz crystal is used to maintain Bluetooth Smart link timing in various Sleep modes.

Figure 4-9. Clock Design



The RC reset circuit shown in Figure 4-10 provides the power-on reset pulse for the PRoC BLE device.

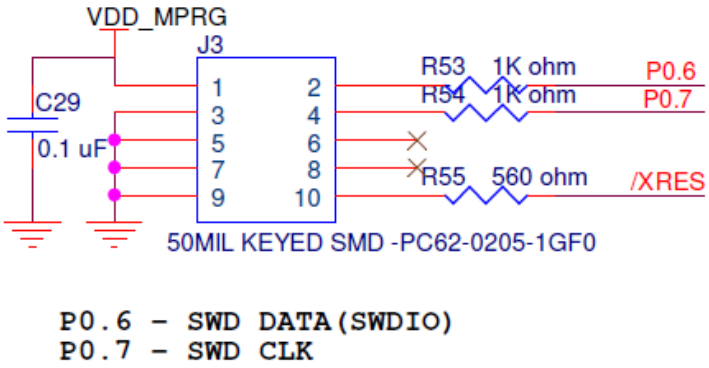
Figure 4-10. Reset Circuit



4.3.3 Program and Debug Circuit

PRoC BLE exposes the SWD interface on the 10-pin header J3, as shown in Figure 4-11. The Programming and Debugging PRoC BLE on the Remote Control and Dongle section describes how to use the J3 header to debug or program PRoC BLE on the remote control.

Figure 4-11. SWD Debug Header

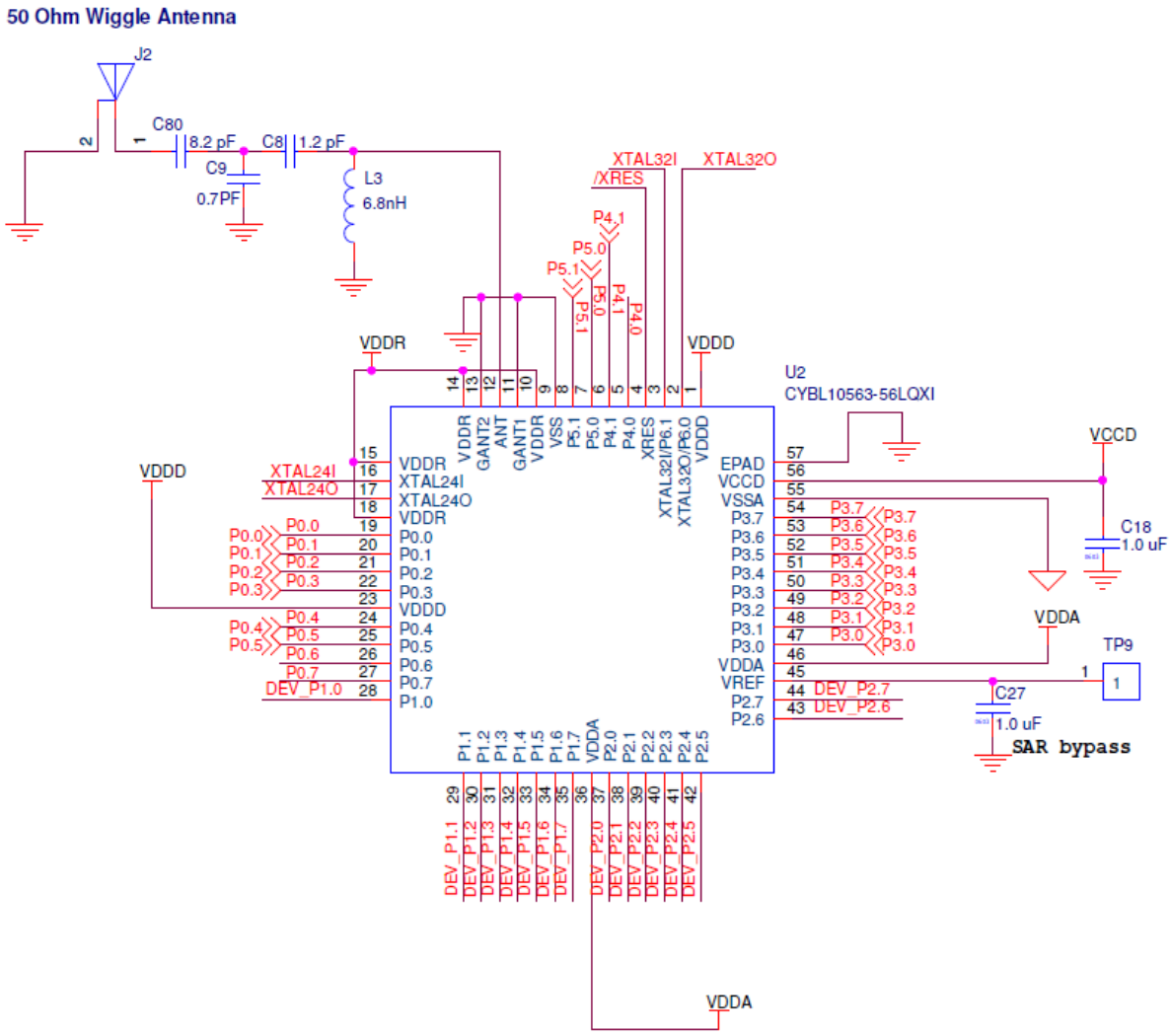


4.3.4 PRoC BLE Device

The CYBL10563-56LQXI PRoC BLE is a 32-bit, 48-MHz ARM Cortex-M0 solution with CapSense®, a 12-bit SAR ADC, 4 Timer Counter Pulse Width Modulators (TCPWMs), 36 GPIOs, 2 Serial Communication Blocks (SCBs), an LCD direct drive, inter-IC Sound Bus (I2S), and an integrated Bluetooth Smart radio with a balun. PRoC BLE includes a royalty-free BLE stack compatible with Bluetooth 4.1 and provides a complete, programmable, and flexible solution for HID. In addition, PRoC BLE provides a simple, low-cost way to add BLE connectivity to any existing system.

GPIOs and serial interfaces are used to interface the essential components for a working remote control—such as the microphone, motion sensor, trackpad, buttons, LEDs, and so on—with PRoC BLE. The data collected from these components is transmitted over the air using the Bluetooth Smart link. The PRoC BLE device is connected to a wiggle antenna through an LC filter circuit for optimum RF performance, as shown in Figure 4-12.

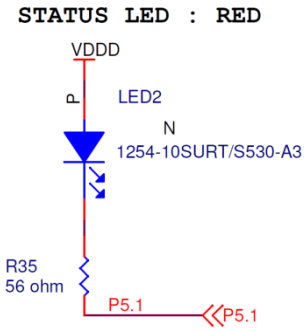
Figure 4-12. PRoC BLE Connections



4.3.5 LED

This section describes the hardware circuit for the red LED, shown in Figure 4-13, is used to show a notification for different firmware states.

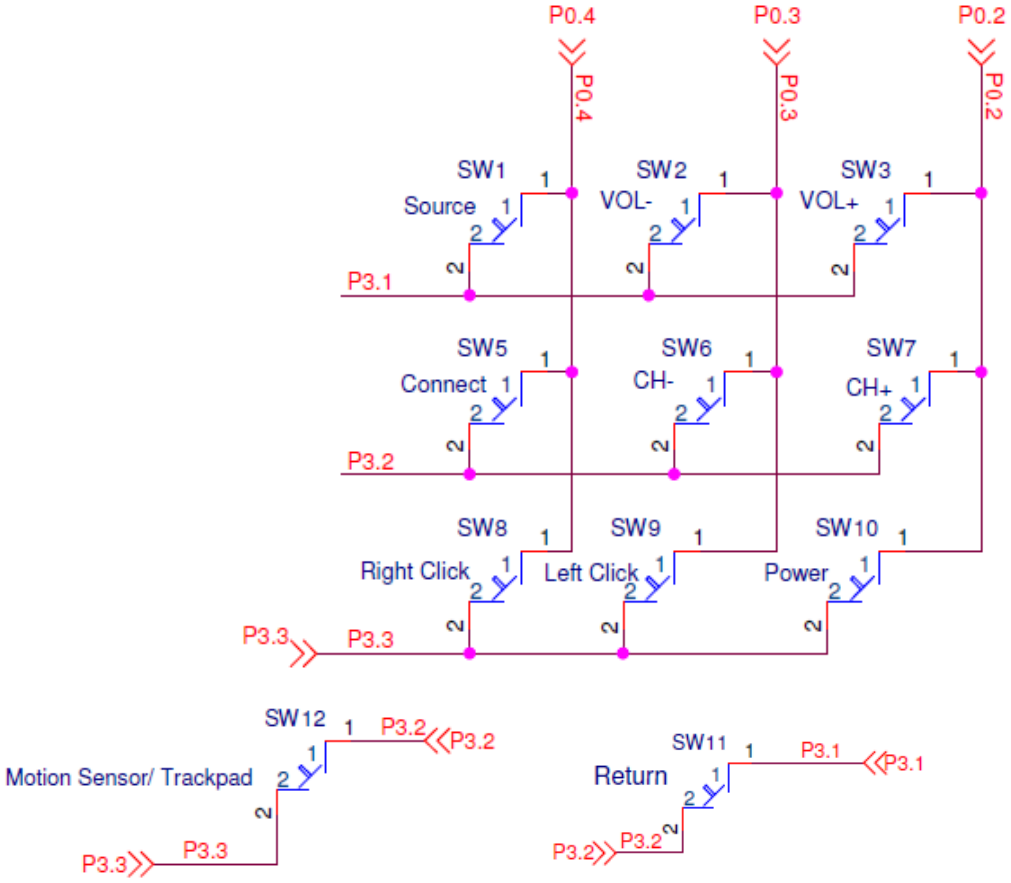
Figure 4-13. LED



4.3.6 Button Matrix

This section describes the hardware circuit button matrix in CY5672. A button matrix provides most of the required remote control key functionalities. A custom implementation is provided for two more buttons (Figure 4-14): a button to enable the motion sensor and the return button. These two buttons use GPIOs that are also assigned to rows in the keyboard matrix. A description of how these buttons are scanned is located in the Keyboard Subsystem section. An additional button is provided on the remote control, as shown in Figure 4-18. In the firmware provided with this kit, this button is used to switch between voice and normal remote control operation.

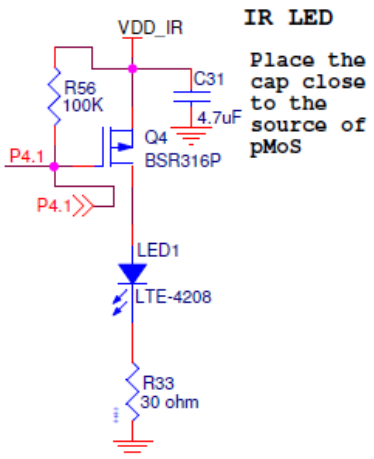
Figure 4-14. Button Matrix



4.3.7 IR LED

The IR LED onboard the remote control is used to implement the NEC and Samsung IR transmission protocols. These protocols are used to operate a TV in a traditional manner. The circuit uses a P-MOS (Q4) to switch the IR LED (LED1) shown in Figure 4-15. The switch is required because the IR LED operates at 30 mA (approximately), which is higher than the PRoC BLE ports can drive/sink.

Figure 4-15. IR LED Circuitry



4.3.8 Motion Sensor

An inertial (motion) sensor (MPU-6500) is used to detect the movement and rotation information in 6 degrees of freedom (X, Y and Z coordinates relative to the previous location). The motion sensor then provides an interrupt using INT pin (MOTION_INT), which is connected to pin 0.5 of the PRoC BLE device as shown in Figure 4-16 and Figure 4-17. On receiving the interrupt, PRoC BLE reads the gyroscope and accelerometer data over the I²C interface, as shown in Figure 4-17. Finally, PRoC BLE processes this information to derive the X, and Y coordinates and sends it over the Bluetooth Smart link.

Figure 4-16. Motion Sensor

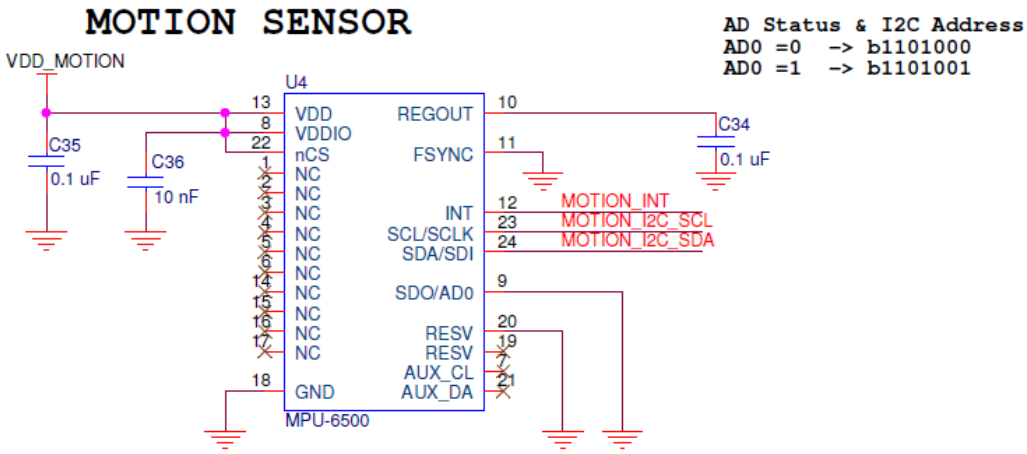
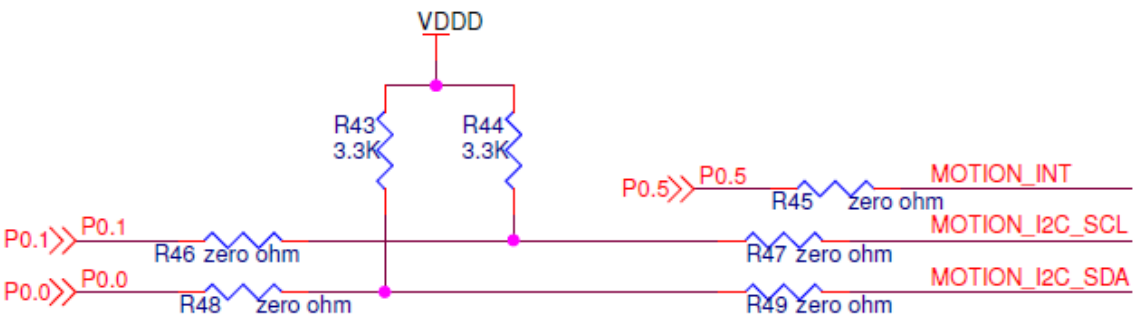


Figure 4-17. I²C Connection between PRoC BLE and Motion Sensor

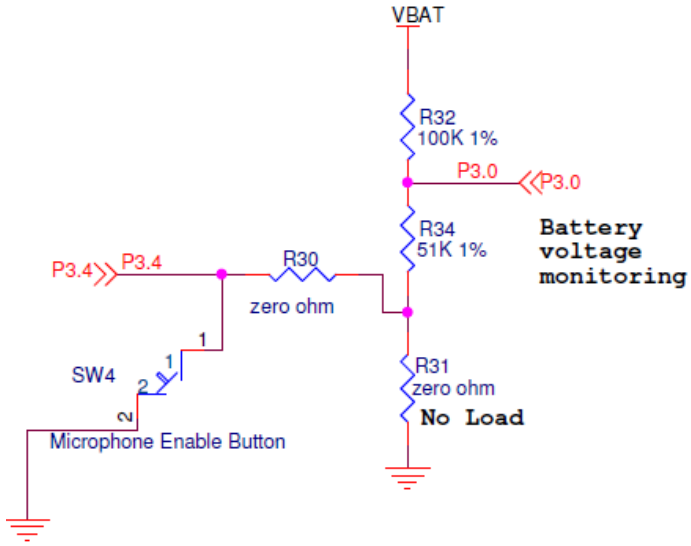


4.3.9 Battery Monitoring

Figure 4-18 shows the battery monitoring circuit along with the microphone enable or voice enable button. Note that the PRoC BLE pin P3[4] is used for following two purposes:

- To read the state of the microphone enable button
- To enable the battery voltage monitoring circuit

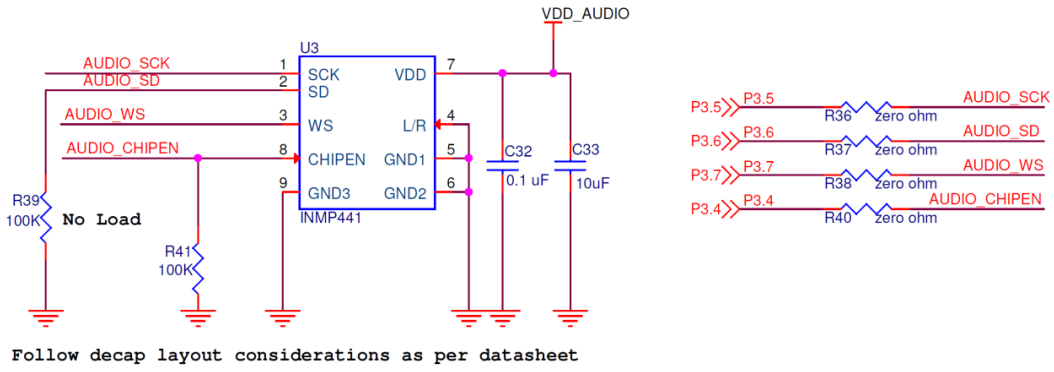
Figure 4-18. Battery Monitoring and Voice Enable Circuit



4.3.10 Audio Input Device

This section describes the hardware circuit implementation for sampling voice on the remote control in voice mode, as shown in Figure 4-19. This circuit utilizes a MEMS sensor–based single-chip solution (INMP441). It acts as the microphone and also implements signal conditioning, an analog-to-digital converter, anti-aliasing filters, and power management. The audio input device is interfaced with the PRoC BLE via I2S.

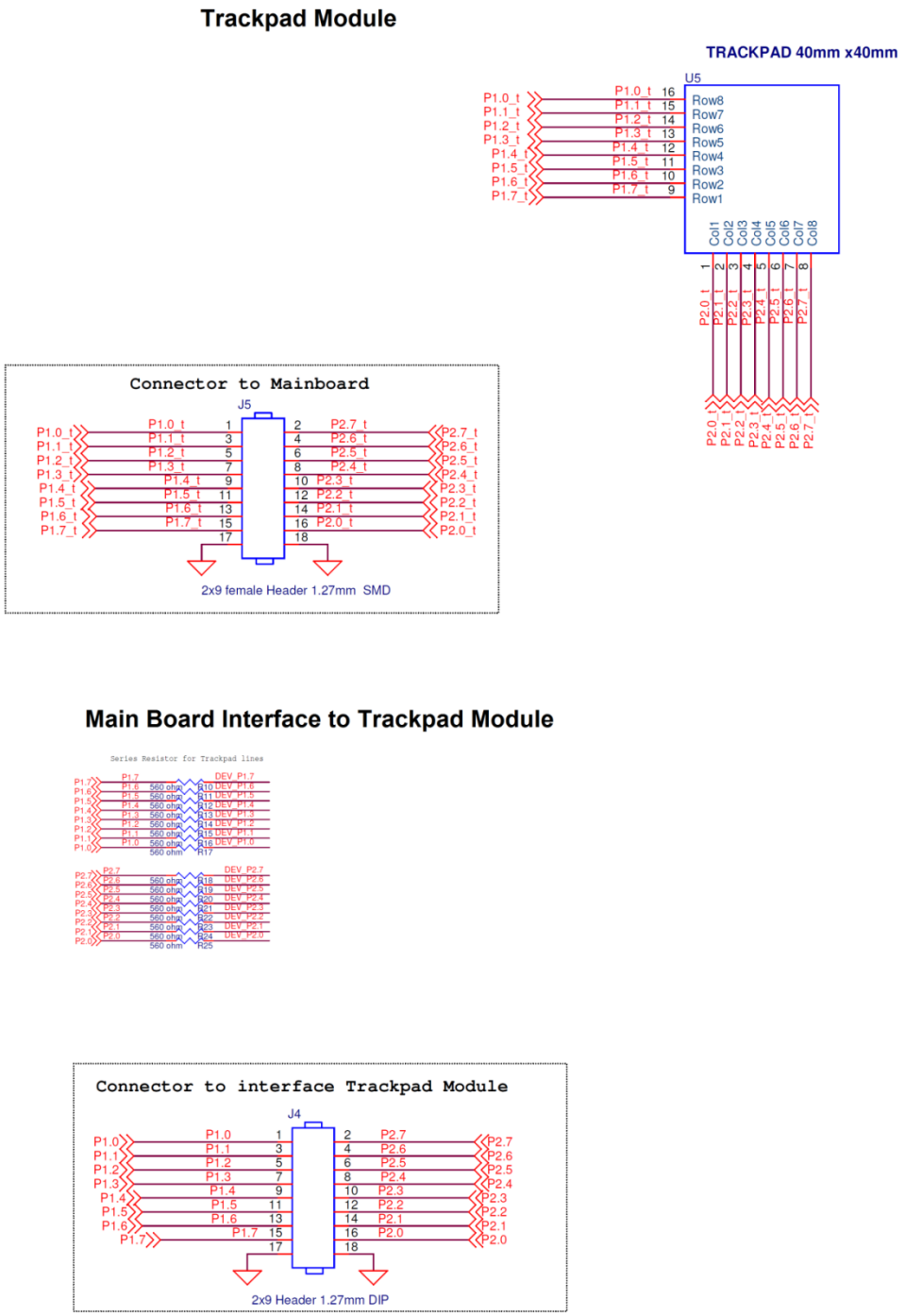
Figure 4-19. Audio Circuit



4.3.11 Trackpad Interface

This section describes the implementation of the trackpad module and its interface with the PRoC BLE device, as shown in Figure 4-20. The capacitive sensor array from the trackpad module is connected to the PRoC BLE GPIOs through series resistors, and they are internally configured as CapSense input lines. The CapSense Gesture Component in PRoC BLE scans and processes these lines to determine the coordinates for finger movement on the trackpad.

Figure 4-20. Trackpad Circuit



4.3.12 Test Points

Table 4-1 lists the test points available on the CY5672 remote control hardware and the associated signal names.

Table 4-1. Test Points on Remote Control

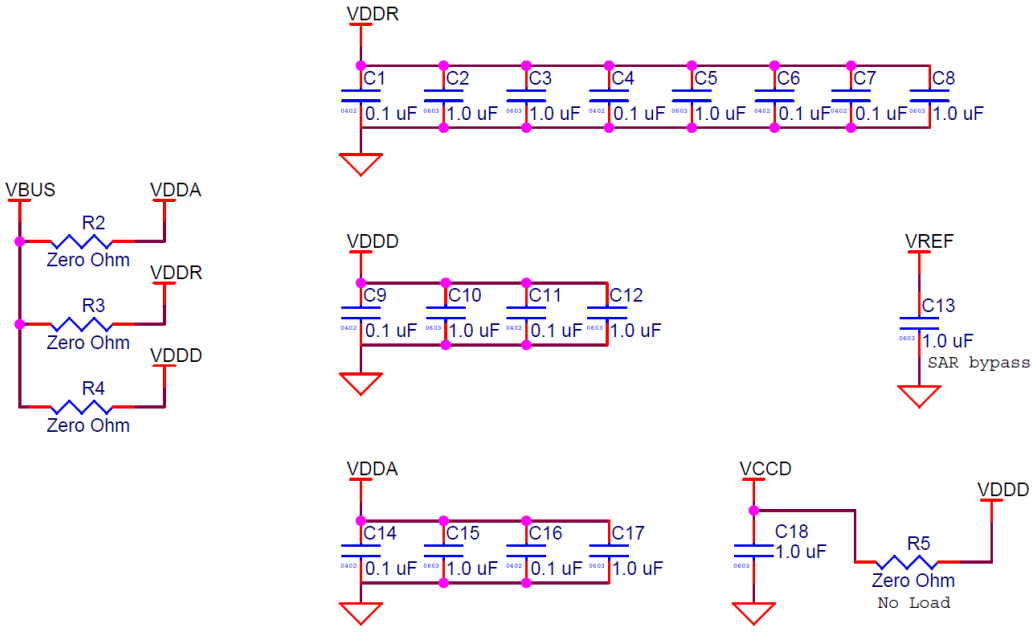
Test Point	Signal Name
TP1	VBAT
TP2	VDD_IR
TP3	VDD_AUDIO
TP4	System power VDD
TP5	System power GND
TP6	VDD_MOTION
TP7	PRoC BLE device power VDDD
TP8	PRoC BLE device power GND
TP9	VREF
TP11	AGND
TP12	AUDIO_WS
TP13	AUDIO_SD
TP14	AUDIO_SCK
TP15	AUDIO_CHIPEN
TP16	MOTION_I2C_SDA
TP17	MOTION_I2C_SCL
TP18	MOTION_INT

4.4 Functional Description – CySmart USB Dongle

4.4.1 Power Supply System and Protection Circuitry

All devices on the CySmart USB dongle are powered using the 5-V power supply from USB (VBUS). The decoupling capacitors are shown in Figure 4-21. The decoupling capacitors decouple the noise in the power supply.

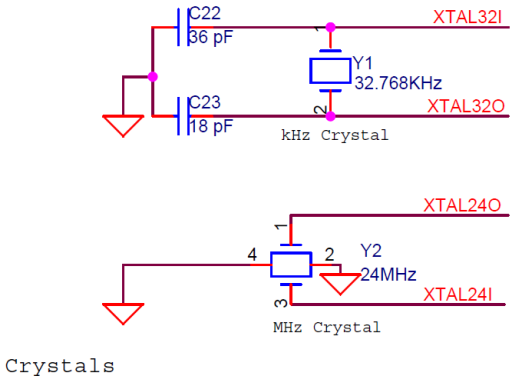
Figure 4-21. Decoupling Capacitors



4.4.2 Clock and Reset

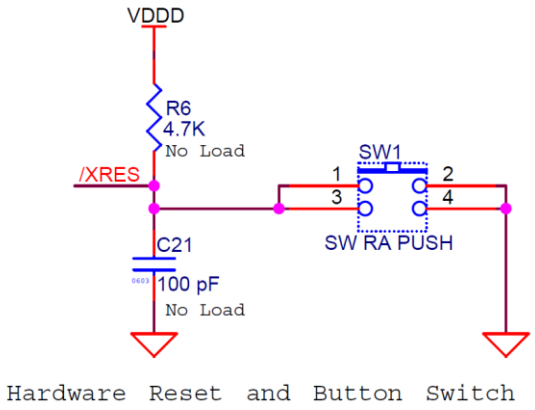
The PRoC BLE device requires two crystal oscillators, one running at 24 MHz and the other at 32.768 kHz, for its operation, as shown in Figure 4-22. Both oscillators are in the Pierce configuration. The 24-MHz oscillator is used by the Bluetooth Smart radio in the Active state, whereas the 32.768-kHz crystal is used to maintain Bluetooth Smart link timing in various Sleep modes.

Figure 4-22. Crystal Circuit



The RC reset circuit for PRoC BLE, including the reset button SW1, is shown in Figure 4-23. The PRoC BLE device is reset when SW1 is pressed.

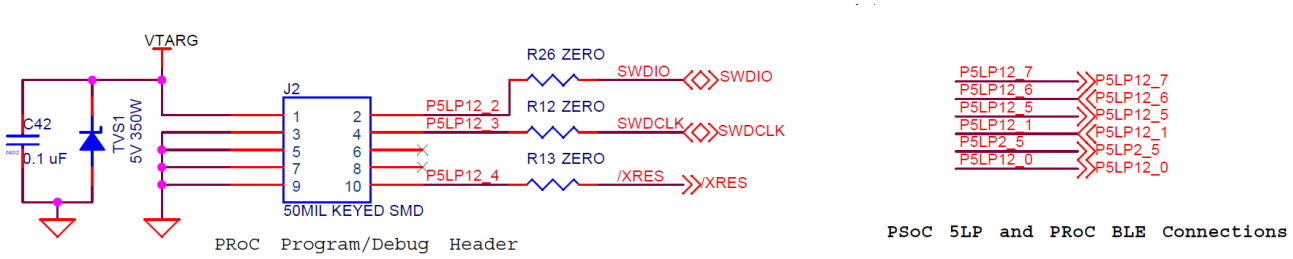
Figure 4-23. Reset Circuit



4.4.3 Program and Debug Circuit

PRoC BLE exposes the SWD interface on the 10-pin header J2, as shown in Figure 4-24. The Programming and Debugging PRoC BLE on the Remote Control and Dongle section describes how to use the J2 header to debug or program the PRoC BLE on the CySmart dongle.

Figure 4-24. SWD Debug Header

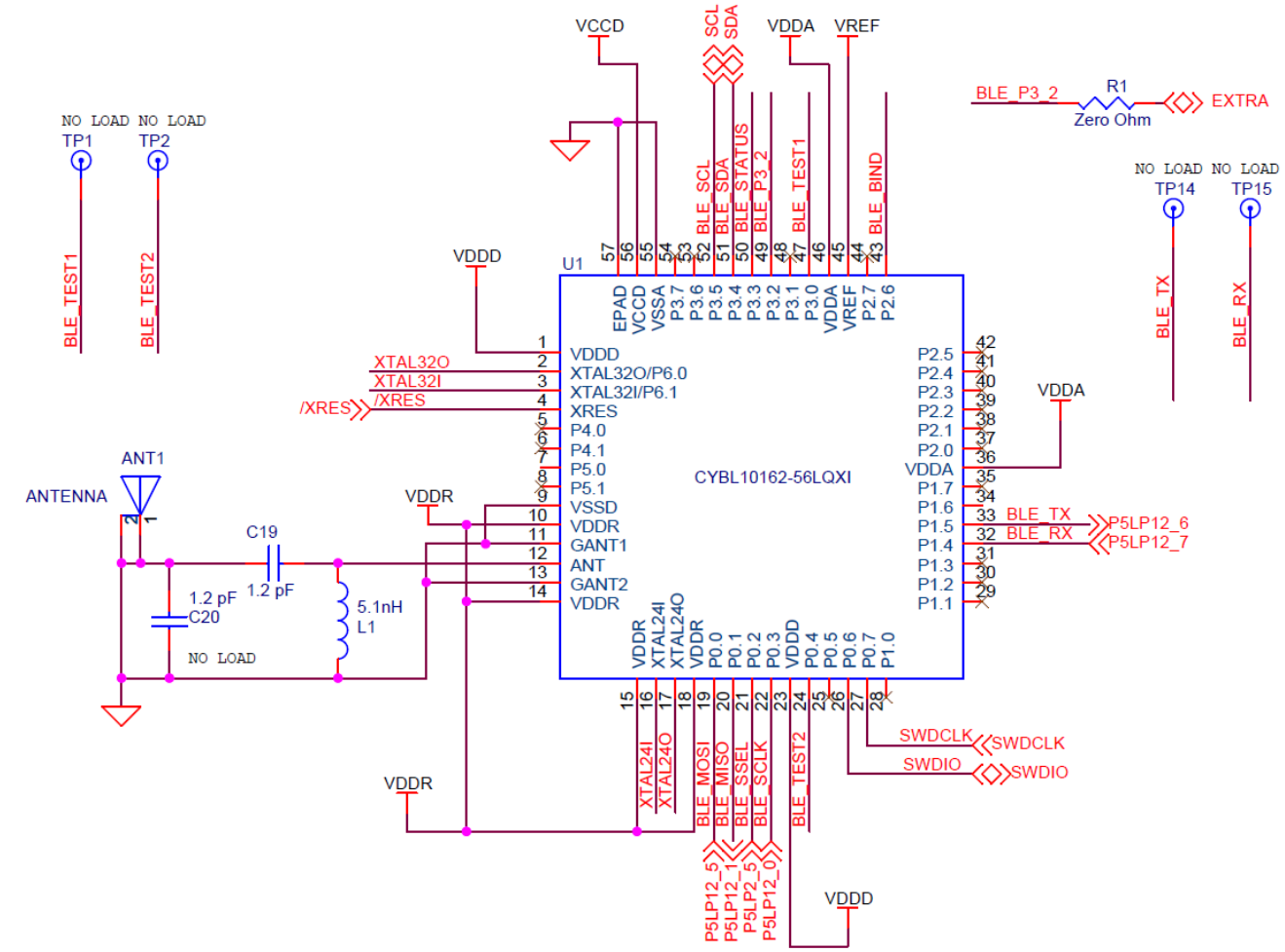


4.4.4 PRoC BLE

The CYBL10162-56LQXI PRoC BLE is a 32-bit, 48-MHz ARM Cortex-M0 solution with CapSense, a 12-bit SAR ADC, 4 TCPWMs, 36 GPIOs, 2 SCBs, LCD direct drive, I2S, and an integrated Bluetooth Smart radio with a balun. PRoC BLE includes a royalty-free BLE stack compatible with Bluetooth 4.1 and provides a complete, programmable, and flexible solution for HID. In addition, PRoC BLE provides a simple, low-cost way to add BLE connectivity to any existing system.

The PRoC BLE device on the CySmart dongle receives the data transmitted from the remote control over the Bluetooth Smart link and sends it to the PC over USB through the PSoC 5 LP device. The PRoC BLE device is connected to a wiggle antenna through an LC filter circuit for optimum RF performance. Figure 4-25 shows the hardware connections for the PRoC BLE device.

Figure 4-25. PRoC BLE Connections



PRoC BLE and Antenna

4.4.5 PSoC 5 LP

The PSoC 5LP acts as a USB to UART bridge between PRoC BLE and the host device (PC, smart TV, and so on). In addition to the UART connection, there are SPI and I²C connections between the PSoC 5LP and the PRoC BLE devices. Figure 4-26 shows the hardware connections for the PSoC 5LP device. The I²C bus contains firmware-controlled pull-ups using FET, which can be enabled or disabled using PSoC 5LP pins, as shown in Figure 4-27. Note that in the firmware provided as part of the kit, only the UART connection between PSoC 5LP and PRoC BLE is used.

Figure 4-26. PSoC 5LP Connections

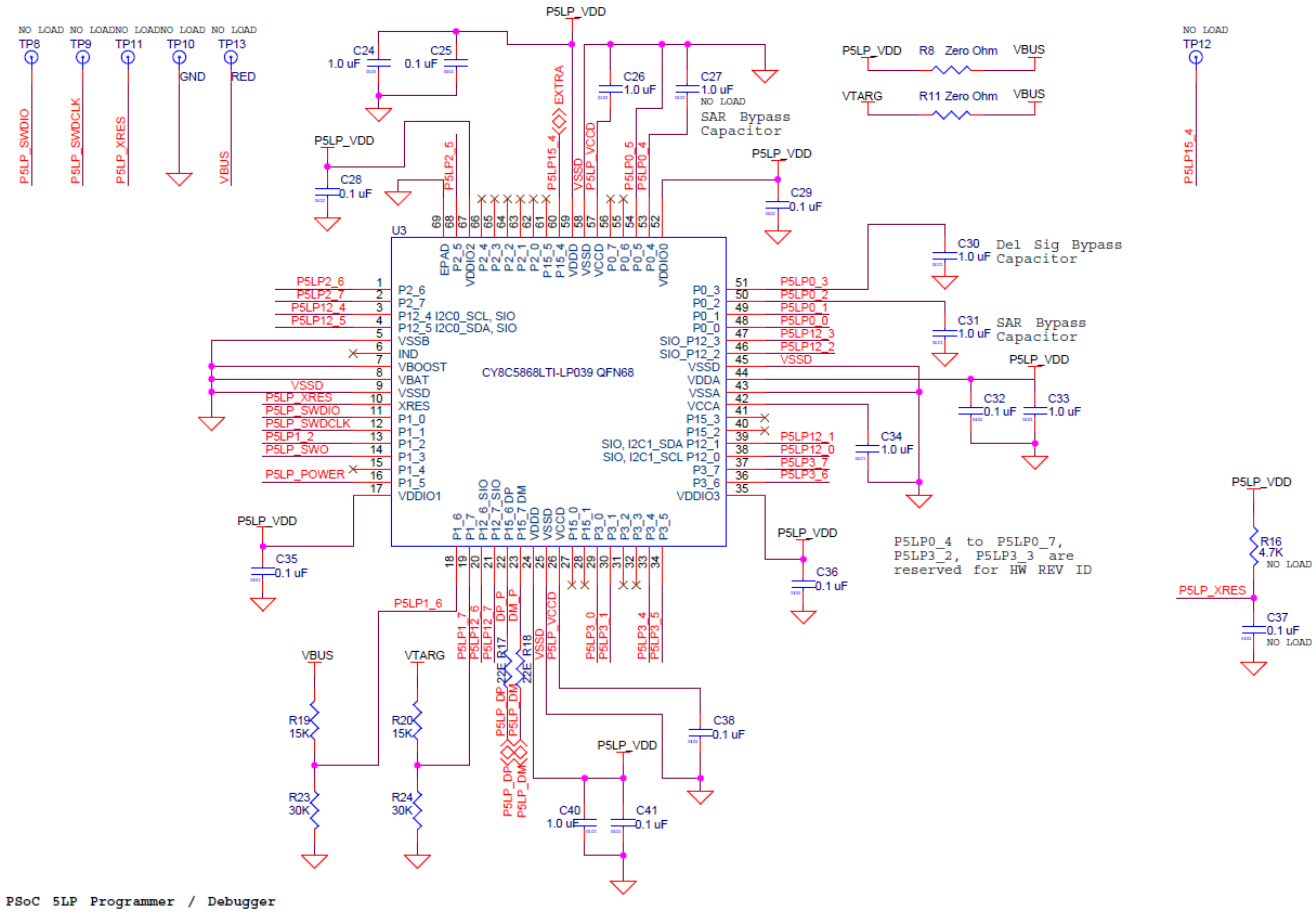
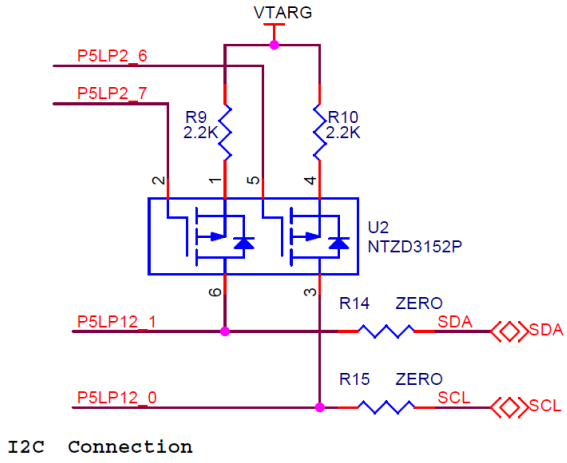


Figure 4-27. I²C Connection between PRoC BLE and PSoC 5LP

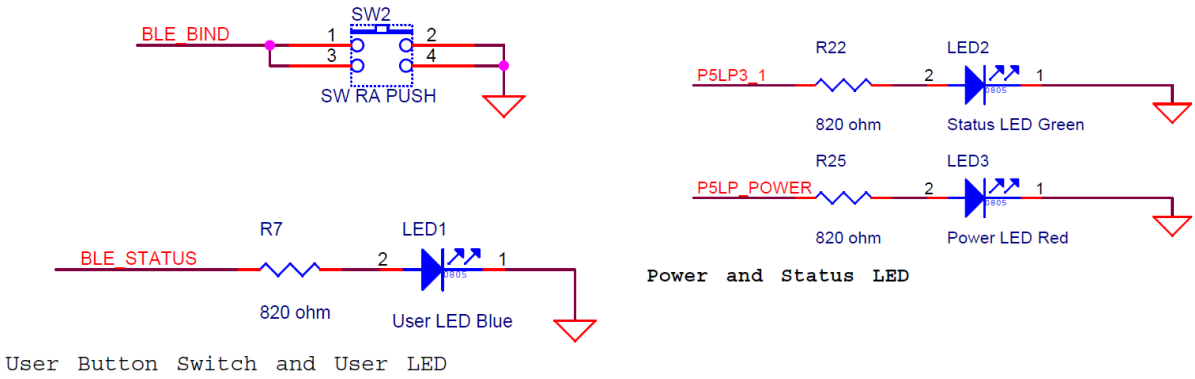


4.4.6 LEDs and Buttons

There are three LEDs for different status indications and a switch for providing input to the PRoC BLE device, as shown in Figure 4-28. In the example firmware provided with the kit, the switch (SW2 or user button) is used as a bind initiation input button. The red LED is for power indication. This LED is always on when the dongle is plugged into the USB port. The green LED turns on after enumeration is complete on the device side (that is, on the CySmart dongle). The blue LED indicates various Bluetooth Smart states. The behavior of the green and blue LEDs is controlled by the firmware on the PSoC 5LP and PRoC BLE devices respectively.

Note: Enumeration completion on the device side is not same as on the host device, like a PC. The green LED stays on when enumeration is complete from the device side. The host may take additional time to load drivers and make the device available for use with other applications.

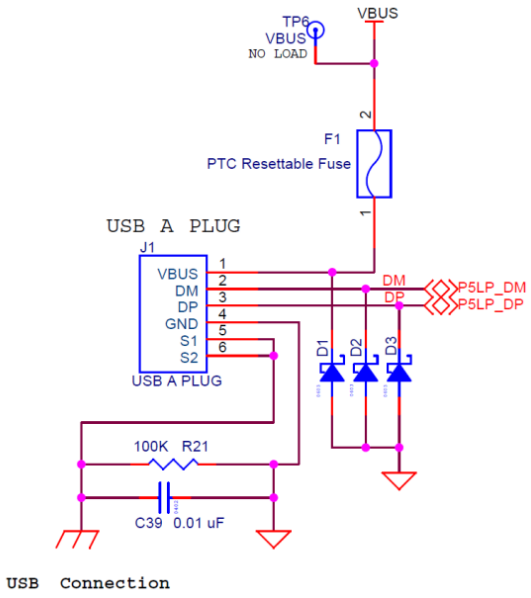
Figure 4-28. LEDs and Buttons



4.4.7 USB Plug

The CySmart dongle uses the USB port to communicate with the PC and for its power supply, as shown in Figure 4-29. The USB differential lines are connected with PSoC 5LP. PSoC 5LP acts as a bridge between the PC and the PRoC BLE device.

Figure 4-29. USB Connector Circuit



5. Firmware

This chapter explains the firmware provided with the CY5672 RDK. Topics covered include the following:

- Firmware for PRoC BLE on the Remote Control
- Firmware for the PRoC BLE and PSoC 5LP on the CySmart USB Dongle

Note: "Subsystem" in this chapter refers to self-contained firmware that enables complete functionality for a hardware feature such as the UART subsystem. Multiple subsystems are integrated to form the complete firmware for an application like a mouse or remote control.

Table 5-1 lists the firmware project names for the Remote Control and the CySmart USB Dongle. Note that there are two firmware projects for CySmart USB Dongle given that the hardware includes both PRoC BLE and PSoC 5LP silicon.

Table 5-1. CY5672 Remote Control RDK Firmware Projects

Hardware	Silicon Part	Associated Project Name and Project File Location
Remote Control	PRoC BLE	Project Name: CY5672_Remote_Control.cywrk Project Path: <*Install Dir*>*CY5672 PRoC BLE RC RDK\1.0\Firmware\RemoteControl*
CySmart USB Dongle	PRoC BLE	Project Name: BLE_HID_CySmart_Dongle.cywrk Project Path: <*Install Dir*>*CY5672 PRoC BLE RC RDK\1.0\Firmware\Dongle\BLE_HID_CySmart_Dongle*
	PSoC 5LP	Project Name: CY5672_Dongle_Bridge.cywrk Project Path: <*Install Dir*>*CY5672 PRoC BLE RC RDK\1.0\Firmware\Dongle\CY5672_Dongle_Bridge*

5.1 Firmware for PRoC BLE on the Remote Control

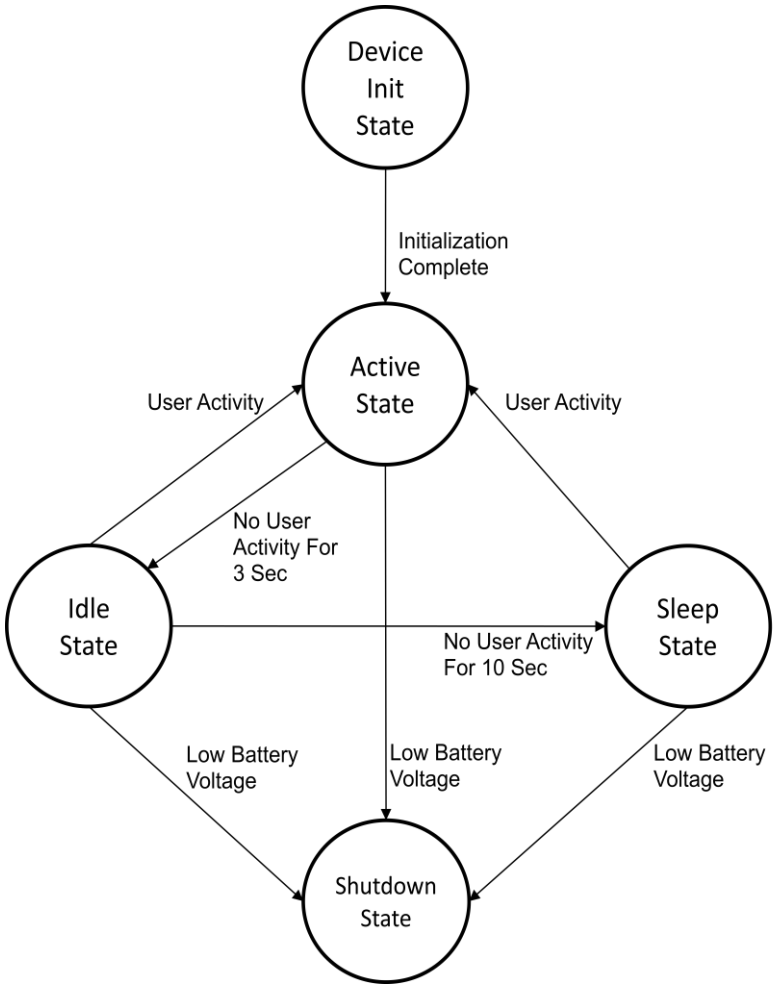
The PRoC BLE device is the heart of the remote control. The remote control firmware project (**CY5672_Remote_Control.cywrk**) runs on PRoC BLE and controls the following:

- System power states
- Bluetooth Smart and IR-based wireless communication
- Trackpad scanning and gesture detection
- Motion sensor
- Button scanning
- Audio acquisition
- LED indications
- Battery monitoring

5.1.1 Firmware Architecture

The remote control device firmware implements a state machine that has five states, as shown in Figure 5-1.

Figure 5-1. State Machine for Remote Control



- **Device Init state:** The remote control firmware enters this state after the device is powered on. All subsystems (motion sensor, ISR for buttons, and trackpad) in the remote control firmware are initialized in this state. The firmware transitions to the Active state after the initialization process is complete.
- **Active state:** The remote control firmware enters this state:
  - After device initialization from the Device Init state
  - On user activity from the Idle state
  - On user activity from the Sleep state

  The remote control application demands a report rate of 100 Hz in the Active state; to satisfy this requirement, every input subsystem (such as the motion sensor, trackpad, and buttons) in the remote control is polled at 10 ms interval. The firmware detects user activity and collects the data from the corresponding input subsystems. The device, in the connected state, sends this data over the Bluetooth Smart link per the HOGP specification.

  When the device is not connected to a peer device (such as a CySmart dongle), it starts advertising to re-establish a connection. While trying to establish the reconnection, the remote control will also send a valid TV control command via the IR LED using the Samsung IR protocol. If the remote control is connected to the dongle and link loss occurs, the user will receive a red LED fast blinking notification on a user activity (such as pressing a button).

  The firmware transitions to the Idle state if there is no user activity for three seconds.

The battery voltage is monitored every 3 seconds to save power. When the firmware detects a low-voltage condition (at approximately 2.0 V), it initiates a red LED slow blinking indication to notify the user about the battery condition. The firmware transitions to the Shutdown state when it detects a battery voltage below the critical voltage of 1.8 V.

- **Idle state**: The remote control firmware enters this state from the Active state if there is no user activity for 3 seconds. In this state, the firmware polls for user activity from the trackpad at a poll interval of 125 ms to save power. All other modules will interrupt if any user activity is present. The firmware also maintains the Bluetooth Smart link with a 10-ms connection interval and a slave latency of 100. The firmware returns to the Active state if any user activity is detected.

  The firmware transitions to the Sleep state if there is no activity for 10 seconds or to the Shutdown state if the battery voltage is below the critical voltage of 1.8 V.

- **Sleep state**: The remote control firmware enters this state from the Idle state if there is no user activity for 10 seconds. In this state, the firmware polls the trackpad every 250 ms. The firmware maintains the Bluetooth Smart link with a 10-ms connection interval and a slave latency of 100.

  While in this state, all other modules will generate interrupts on user activity, which wakes up the MCU and transitions the firmware to the Active state. The firmware transitions to the Shutdown state if the battery voltage is below the critical voltage of 1.8 V.

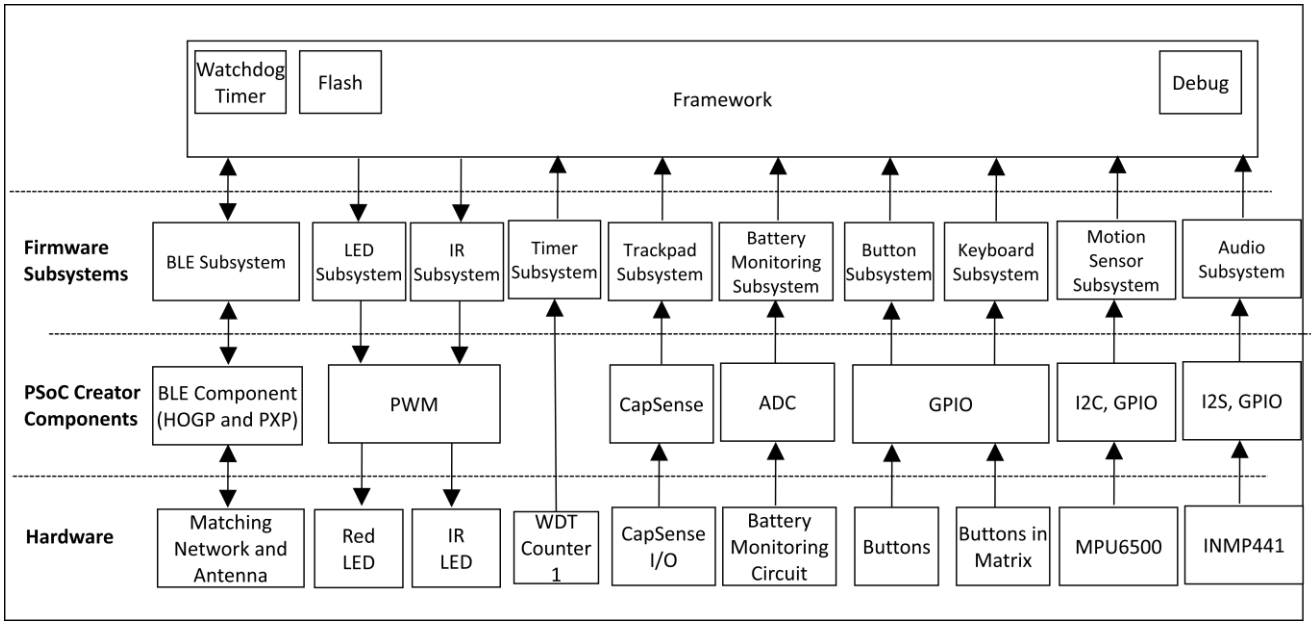
  The system power consumption is considerably lower in this state compared to that in the Active and Idle states.

- **Shutdown state:** The remote control firmware enters the Shutdown state when the battery voltage is below the critical voltage of 1.8 V. In this state, the firmware shuts down all input and output subsystems, including the BLE subsystem, and enters the MCU stop mode (PRoC BLE stop mode). This effectively renders the remote control nonfunctional. The user is expected to replace the batteries in this state to make the remote control operational again. That is, power must be removed from the system in this state before the remote control operates again.

In all the states except Shutdown, the battery is monitored every 3 seconds, and the MCU (PRoC BLE) is put into a Deep Sleep state at every possible opportunity.

The remote control firmware implements these states in an application framework with the help of the PSoC Creator Components noted in Figure 5-2. The input subsystems can detect and record any user activity. The output subsystems either transmit the data or provide an indication to the user. The framework implements the modes described previously.

Figure 5-2. Firmware Architecture



The following subsystems are present in the remote control device:

- Watchdog timer subsystem: Restores the mechanism to reset the remote control if any subsystem enters an unknown state
- BLE subsystem: Sends data over the air using the BLE HOGP

- LED subsystem: Indicates to the user certain activity by blinking or by glowing the red LED
- IR LED subsystem: Sends the multimedia key over IR
- Trackpad subsystem: Scans the trackpad for user activity and translates the user activity into the appropriate gesture
- Battery monitoring subsystem: Detects the battery voltage and calculates the battery level
- Button subsystem: Detects the user input to determine whether to enable audio input; also allows the user to switch between the trackpad and motion sensor.
- Keyboard matrix subsystem: Detects any key press to provide multimedia control functionality
- Motion sensor subsystem: Scans the motion sensor/gyro/accelerometer to detect user activity when the remote is moved and converts it into cursor movement
- Audio subsystem: Takes input from the microphone connected to the INMP441 codec
- Debug subsystem: Sends debug print using the UART

The IR LED subsystem, BLE subsystem, and LED subsystem are output subsystems, and the rest of the subsystems are input subsystems. These subsystems use PSoC Creator Components. Figure 5-3 gives a schematic view of the PRoC BLE remote control device.

Figure 5-3. PSoC Creator Schematic View of PRoC BLE Remote Control

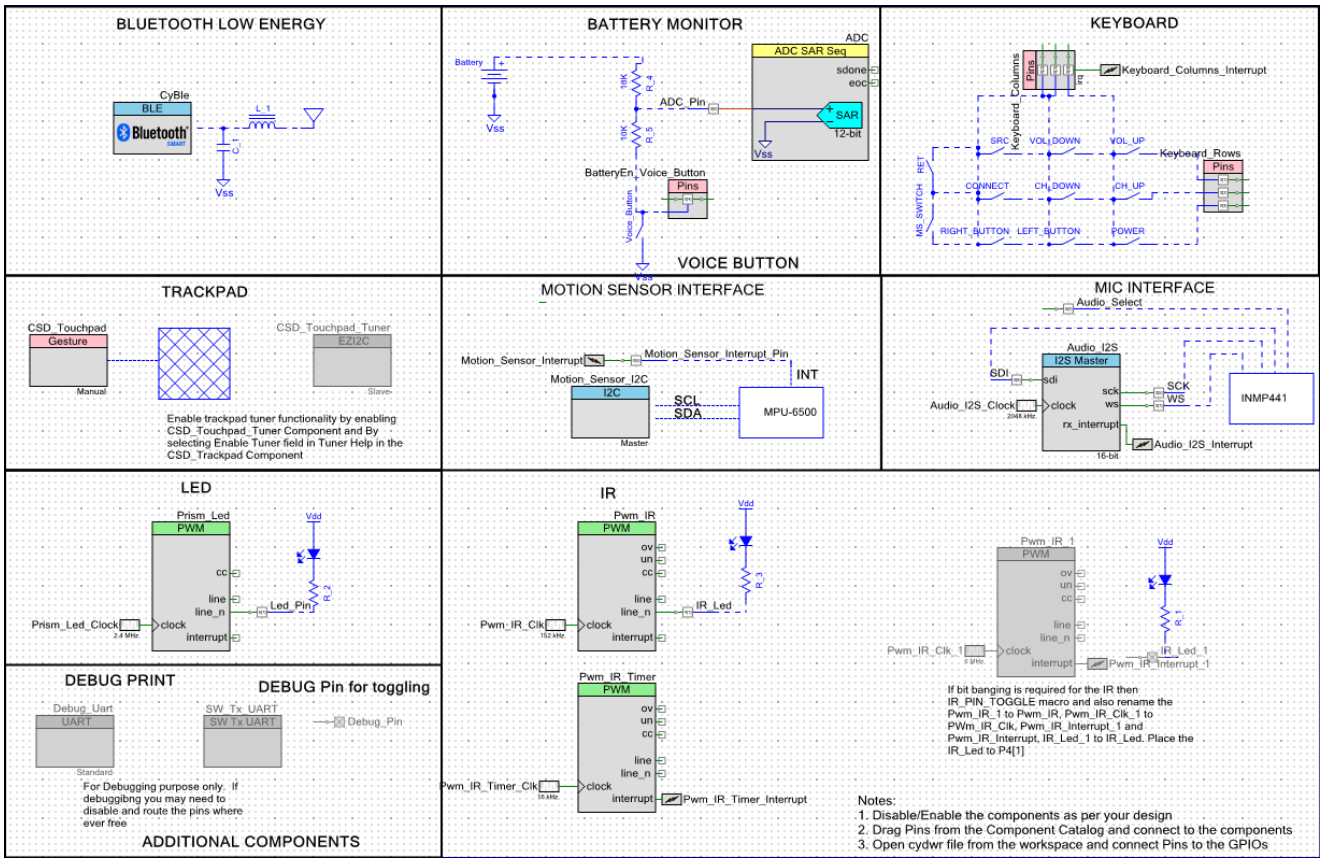


Table 5-2 lists the file structure used by the remote control project and the key functions implemented in each file.

Table 5-2. File Structure for the Remote Control Project

File Name	Details
main.c	This file is the top-level application, which initializes the system and handles the switching between power states based on user activity. This file has the following functions: ▪ ***main()*** – The main function for the application. ▪ ***Device_FW_Init()*** – Initializes all system blocks. ▪ ***Device_FW_Active()*** – Watchdog timer counter 1 is configured to trigger an interrupt every 1 ms. All subsystem blocks are polled every 10 ms, and notifications are sent to connected host when data is available. ▪ ***Device_FW_Idle()*** – Watchdog timer counter 1 is configured to trigger an interrupt every 125 ms. All system blocks are polled every 125 ms. ▪ ***Device_FW_Sleep()*** – Watchdog timer counter 1 is disabled and system blocks are put into a low-power mode. ▪ ***Device_FW_Stop()*** – All blocks are disabled and system is put into the hibernate mode.
device.c	This file handles the initialization of individual system blocks, time keeping, polling for events, low-power mode implementation, and sending notifications to the connected host when data is available. ▪ ***Device_Init()*** – Initializes all system blocks. ▪ ***Device_Active ()*** – Polls the data (based on activity) from all system blocks every 10 ms and sends a notification to the connected host. ▪ ***Device_Low_Power_State()*** – Polls the data from button, battery monitoring, and trackpad (if enabled) every 125 and 250 ms in idle and sleep mode respectively. Other modules should raise the interrupt if any activity is detected. ▪ ***Device_Timer_Callback()*** – This function is a callback from watchdog timer counter 1. The compare value for the Prism_Led (pseudo random PWM) is incremented. This is used to accomplish the breathing LED effect. • ***Device_ShutDown ()*** – Disables all system blocks.
timer.c	This file handles initialization and generation of a timer tick every 1 ms. ▪ ***Timer_Init()*** – Initializes the timer block. ▪ ***Timer_Get_Time_Stamp()*** – Returns the current timestamp. ▪ *Timer_Time_Elapsed()* – Returns true if the time exceeded an interval specified as one of parameters. ▪ ***Timer_Set_Period()*** – Sets the time period between timer interrupts. ▪ ***Timer_CallBack()*** – Increments the tick timer by a value equal to the tick increment. The tick increment depends on the state of the system.
watchdog_timer.c	This file handles watchdog timer reset functionality. Watchdog timer system reset is configured to occur after 6 seconds. ▪ ***Watch_Dog_Timer_Init()*** – Initializes the watchdog timer counter 0 and configures it to trigger a reset interrupt. ▪ ***Watch_Dog_Timer_Clear()*** – Clears the watchdog timer counter 0. ▪ ***Watch_Dog_Timer_Disable()*** – Disables the watchdog timer counter 0. ▪ ***Watch_Dog_Timer_Enable()*** – Enables the watchdog timer counter 0.

File Name	Details
ble.c	This file handles BLE initialization, configuration, advertisement, notifications, and responses to BLE events. ▪ ***Ble_Init()*** – Initializes the BLE Component. ▪ ***Ble_Is_Init_Completed()*** – Returns the status of BLE initialization. ▪ ***Ble_Configure()*** – Configures the BLE Component after the BLE stack on event. ▪ ***Ble_Set_Address()*** – Sets the public BLE address for the device. ▪ ***Ble_Create_Audio_Channel()*** – Creates an audio channel by sending the audio command to the dongle. ▪ ***Ble_Disconnect_Audio_Channel()*** – Disconnects the audio channel by sending the audio command to the dongle. ▪ ***Ble_Send_Battery_Data()*** – Sends the battery data to a connected client. ▪ ***Ble_Send_Data()*** – Sends the mouse/keyboard report to a connected client. ▪ ***Ble_StartAdvertisement()*** – Starts a directed/undirected advertisement. ▪ ***Ble_StopAdvertisement()*** – Stops a directed/undirected advertisement. ▪ ***Ble_Get_State()*** – Returns the current state of BLE. ▪ ***Ble_Set_State()*** – Sets the state of BLE based on the system state. ▪ ***Ble_Stop()*** – Stops the BLE block. ▪ ***Ble_Enter_LowPowerMode()*** **–** Puts BLE into deep sleep mode. ▪ ***Ble_WriteBondedList()*** – Writes the bond information into flash for future retrieval. ▪ ***Ble_Update_Serial_Number_String()*** – Updates the serial number string in the device information service with the silicon ID. ▪ ***Ble_AppCallBack()*** – Callback to get GAP, GATT, and L2CAP events from the BLE Component. ▪ ***Ble_BasCallBack()*** – Callback registered with the Battery Service. ▪ ***Ble_ScpsCallBack()*** – Callback registered with the Scan Parameters Service. ▪ ***Ble_HidsCallBack()*** – Callback registered with the HID service. ▪ ***Ble_IasCallBack()*** – Callback registered with the Immediate Alert Service. ▪ ***Ble_LinkLossCallBack()*** – Callback registered with the Link Loss Service. ***Ble_TxPowerCallBack()*** – Callback registered with the Tx Power Service.
trackpad.c	This file handles trackpad initialization, configuration, polling, centroid calculation, and gesture detection. ▪ ***Trackpad_Init()*** – Initializes the trackpad module. ▪ ***Trackpad_Start_Poll()*** – Starts the trackpad sensor scanning. ▪ ***Trackpad_IsComplete()*** – Returns the status of trackpad sensor scanning. ▪ ***Trackpad_Poll()*** – Detects the number of fingers and decodes the gestures on the trackpad. ▪ ***Trackpad_Get_Report()*** – Generates the report for gestures on the trackpad. ▪ ***Trackpad_Set_State()*** – Sets the state of the trackpad module based on the system state. ▪ ***Trackpad_IsActive()*** – Returns the user activity on the trackpad; returns true if the finger is present on the trackpad. ▪ ***Trackpad_Stop()*** – Stops trackpad scanning.
trackpad_gesture_mapping.c	This file maps the detected trackpad gestures with the appropriate HID commands that are recognized by the PC. ▪ ***Trackpad_Gesture_Map()*** – Maps trackpad gestures to the appropriate HID report format.

File Name	Details
trackpad_custom_gesture.c	This file handles the detection of custom gestures that are not available in the CSD Component. • ***Trackpad_Custom_Gesture_Init()*** – Initializes various parameters required for custom gestures ▪ ***Trackpad_Custom_Gesture_Detection()*** – Decodes custom gestures such as top-to-bottom swipe, vertical scroll, and horizontal scroll. ▪ ***Trackpad_Custom_PointerMovement_Calulation()*** – Calculates pointer movement and fills the appropriate HID report format ▪ ***Trackpad_Custom_PointerMovement_Reset()*** – Resets the pointer movement parameters
keyboard.c	This file handles GPIO initialization for reading the buttons that are connected in the matrix. It also detects changes in button status and decodes to the appropriate HID codes. All buttons except for the microphone enable/disable, motion sensor switch and return buttons are handled by this module. ▪ ***Keyboard_Init()*** – Initializes the keyboard module. ▪ ***Keyboard_Poll()*** – Polls the keyboard for activity. ▪ ***Keyboard_Get_Report()*** **–** Fills the HID data depending upon the button pressed. ▪ ***Keyboard_Set_State()*** – Modifies the configuration of the keyboard module based on the system state (for example, enabling and disabling interrupts). ▪ ***Keyboard_IsActive***() – Checks if any key is pressed in the keyboard matrix. ▪ ***Keyboard_Stop***() – Stops the keyboard module by disabling the interrupts and changing the state of the GPIOs to Highz. ▪ ***Keyboard_Scan()*** – Scans the keyboard matrix if any key is pressed. ▪ ***Keyboard_Scan_Column()*** – Scans the columns by exciting one row at a time. ▪ ***Keyboard_Detect_Ghost***() – Detects a condition where three buttons are pressed between two rows and two columns and the algorithm detects that four keys are pressed. ▪ ***Keyboard_Detect_Keys()*** – Identifies the keys that have been pressed. ▪ ***Keyboard_Count_Active_Keys***() – Counts the number of active columns. ▪ ***Keyboard_Add_Queue()*** – Adds the corresponding button to the queue. ▪ ***Keyboard_Update_Queue()*** – Updates the debounce value of the button that is present in the queue. ▪ ***Keyboard_Update_Release_Key_State***() – Updates the key status for the key that is released. ▪ ***Keyboard_Remove_Queue()*** – Removes the button from the queue. ▪ ***Keyboard_Interrupt_Callbck()*** – Interrupt callback function when there is a button status change in idle or sleep mode. ▪ ***Keyboard_Update_BackChannel_Data()*** – Updates the status of the LED that can be used for keys such as Caps Lock, Num Lock, and Scroll Lock. This is disabled for the remote control project.
button.c	This file handles GPIO initialization for ***b***uttons and decodes the state transition of buttons. This module only handles the microphone enable/disable, motion sensor switch and return buttons. ▪ ***Button_Init()*** – Initializes the GPIO and configures the interrupts. ▪ ***Button_Poll()*** – Polls for the button status. ▪ ***Button_Set_State()*** – Modifies the configuration of buttons based on the system state (enabling and disabling interrupts). ▪ ***Button_Get_Report()*** – Returns the button report only when there is a transition in button state. ▪ ***Button_Stop()*** – Stops the button module. All button interrupts are disabled. ▪ ***Button_Debounce()*** – Handles button debouncing and provides the correct button status.

File Name	Details
audio.c	This file handles I2S initialization for streaming audio. ▪ ***Audio_ADPCM_Encode()*** – Encodes audio data using ADPCM compression. ▪ ***Audio_CallBack()*** **–** Callback function when data is available in the audio module. ▪ ***Audio_Init()*** – Initializes the audio module by starting the I2S Component. ▪ ***Audio_Start()*** – Starts reading the audio data from the codec. ▪ ***Audio_Poll()*** – Polls if any audio data is available. ▪ ***Audio_Get_Report()*** **–** Packetizes the audio data if available. ***Audio_Stop()*** – Stops the audio module by stopping the I2S Component.
motion_sensor.c	▪ ***Motion_Sensor_Init()*** – Initializes/configures the motion sensor. ▪ ***Motion_Sensor_Start_Sampling()*** – Starts the sampling of data inside the motion sensor. ▪ ***Motion_Sensor_Poll()*** – Checks if the motion sensor has some movement detected. ▪ ***Motion_Sensor_Stop_Sampling()*** – Stops the sampling of data inside the motion sensor. ▪ ***Motion_Sensor_Set_Resolution()*** – Updates the resolution of mouse movement ▪ ***Motion_Sensor_Set_State()*** – Sets the enable/disable state of the motion sensor depending upon the state. ▪ ***Motion_Sensor_Get_Report()*** – Calculates the mouse position shift and updates the HID report passed as a parameter. ▪ ***MotionSensor_IsMotionDetectedLowPower()*** **–**- Polls for the motion sensor activity in low-power mode. ▪ ***Motion_Sensor_IsActive()*** – Checks if the motion sensor is active or not. ▪ ***Motion_Sensor_Stop()*** – Changes the motion sensor's state to low-power mode. ▪ ***Motion_Sensor_CallBack()*** – Callback function called when motion sensor interrupt is triggered
motion_sensor_hal.c	▪ ***Motion_Sensor_Hal_Init()*** – Initializes the motion sensor by calling the motion sensor driver. ▪ ***Motion_Sensor_Hal_GetGyroData()*** – Collects the gyro data. ▪ ***Motion_Sensor_Hal_GetAccData()*** – Collects the accelerometer data. ▪ ***Motion_Sensor_Hal_Enable_LowPowerMode()*** – Puts the motion sensor in low-power mode by disabling the gyro and enabling the accerolometer. ▪ ***Motion_Sensor_Hal_Disable_LowPowerMode()*** – Disables the motion sensor low-power mode and restores the default configuration ▪ ***Motion_Sensor_Hal_CallBack()*** – Provides call back to an interrupt from the motion sensor
battery.c	This file handles initialization of the ADC and the measurement of battery voltage using the ADC. ▪ ***Battery_Init()*** – Starts the battery module by starting the ADC Component. ▪ ***Battery_Get_Value()*** – Measures and returns battery voltage. ▪ ***Battery_Stop()*** – Stops the battery module by stopping the ADC Component. ▪ ***Battery_No_Battery_Connected***() – Changes drives modes of the battery enable pin. This is enabled if macro ENABLE_MINIPROG_NO_BATTERY_WORKAROUND is enabled.
led.c	This file handles LED initialization and the creation of various effects in glowing LEDs. ▪ ***Led_Init()*** – Initializes the LED block. ▪ ***Led_Blinking()*** – Creates a specific LED effect based on the configuration of the LED breathing effect. ▪ ***Led_On()*** – Keeps the LED in the ON state with a specific intensity.

File Name	Details
	▪ ***Led_Update()*** – This function is a callback function. The compare value for the Prism_Led (pseudo random PWM) is incremented in this function. ▪ ***Led_IsComplete() –*** Returns the status of the LED breathing effect (complete/not complete). ▪ ***Led_Stop()*** – Stops the LED module and resets all variables.
flash.c	This file handles storage/retrieval of bond information to/from flash. ▪ ***Flash_Save()*** – Stores the information into the flash. ▪ ***Flash_Load()*** – Restores the information from flash and copies it into the RAM buffer.
debug.c	This file handles print debug messages (logs) over the UART. ▪ ***Putc() –*** This function is used to print a character on the UART. ▪ ***Byte_To_ASCII() –*** This function is used to map a byte to its corresponding ASCII value. ▪ ***Debug_Print_Start() –*** This function is used to initialize debug functionality. ▪ ***Debug_Print() –*** This function is used to print a message (logs) on the UART.
platform.h	This file contains macros that control various options in the firmware related to debug, touchpad tuning, use of MCU power states, manufacturing test kit (MTK), and BLE power states. • ***DEVICE_BATTERY_PARALLEL*** – Enables the macro if the battery terminal is of a parallel type; otherwise, the battery terminal is expected to be in series. ▪ ***ENABLE_SYSTEM_DEEP_SLEEP*** – Enables deep sleep mode. BLE will be disconnected when it enters this mode. ▪ ***MCU_DEEP_SLEEP_DISABLED*** – Disables the MCU deep sleep. ▪ ***ENABLE_DEBUG_PIN*** – Enables the macro for toggling GPIO. Make sure the Debug_Pin component is enabled in the TopDesign schematic. ▪ ***DEBUG_PRINT*** – Enables debug prints. • ***SOFTWARE_UART*** – Make sure to enable SW_Tx_UART component in the TopDesign after defining the SOFTWARE_UART macro. Else enable Debug_UART component. Debug_UART provides faster data transfer compared to SW_Tx_UART. However Debug_UART can be routed only to specific pins ▪ ***DISABLE_TIMER*** – Disables the timer module. It not recommended to disable the timer block, as much of the functionality of this application depends upon it. ▪ ***DISABLE_FLASH*** – Disables the flash module. ▪ ***DISABLE_WATCH_DOG*** – Disables the watchdog reset function but not the use of watchdog as a 1-ms timer ▪ ***DISABLE_BATTERY*** – Disables the battery module. ▪ ***ENABLE_HARDWARE_WORKAROUND_MINIPROG*** - Enables the drive mode of the BatteryEn_Voice_Button GPIO from Open Drain to Pull Up when powered using MiniProg3 and batteries are not connected. This is required as battery enable and voice button are mapped to the same GPIO. If this macro is not enabled and MiniProg3 is connected without batteries, there is a possibility of firmware sensing a voice button press even though it is not actually pressed. • ***DISABLE_AUDIO*** – Disables the audio module. ▪ ***AUDIO_RUN_ON_ECO*** – Defines the macro to run the I2S on ECO, not on IMO. ▪ ***AUDIO_SEND_FREQUENT_SYNC_PACKET*** – Enables the logic to send the sync packet frequently. ▪ ***DISABLE_LED*** – Disables the LED module. ▪ ***DISABLE_MOTION_SENSOR*** – Disables the motion sensor module. ▪ ***DISABLE_MOTION_SENSOR_PINREAD*** – Disables reading the pin status and he motion

File Name	Details
	sensor register to detect user activity from the motion sensor in low-power mode. ▪ ***ENABLE_MOTION_SENSOR_FLASH_WRITE –*** Enables the logic to store the gyro offset on the flash. ▪ ***DISABLE_IR*** – Disables the IR module. ▪ ***NEC_PROTOCOL*** – Defines the macro for sending IR data in the NEC protocol format; otherwise, the custom IR format will be used for sending data. ▪ ***IR_PIN_TOGGLE*** – Defines the macro for bit banging the IR data by toggling the GPIO. Otherwise, TCPWM will be used for this purpose. **Note:** Please check the comment in the *TopDesign.cysch* file. ▪ ***DISABLE_BUTTON*** – Disables the button module. • ***DISABLE_KEYBOARD*** – Defines the macro for disabling the keyboard module. **Note:** When DISABLE_KEYBOARD is defined, then BLE will advertise when powered on. ▪ ***KEYBOARD_MULTIMEDIA_SUPPORT*** – Defines the macro for supporting multimedia keys. ▪ ***ENABLE_IR_CODES*** – Enables support for IR key codes in the Keyboard module. ▪ ***DISABLE_KEYBOARD_INTER_COLUMN_BUTTON*** – Defines the macro for disabling button scanning between the rows. ▪ ***DISABLE_BLE*** – Defines the macro for disabling the BLE module. • ***ENABLE_BLE_LOW_POWER_MODE*** – Defines the macro for enabling BLE low-power mode. **Note:** If ENABLE_BLE_LOW_POWER_MODE is disabled, then the power of the remote will go high. ▪ ***ENABLE_CONNECTION_UPDATE_DISCONNECT*** – Defines the macro for disconnecting from the peer device if the L2CAP parameter update is rejected. ▪ ***ENABLE_BLESS_CLOCK_CONFIGURATION*** – Enables the logic to set the BLE clock configuration. **Note:** This needs to be set depending upon the WCO clocks. ▪ ***ENABLE_BONDING*** – Defines the ENABLE_BONDING macro for storing the bond information in the flash. ▪ ***BLE_GET_STACK_VERSION*** – Enables the code to get the version of the BLE stack being used in the firmware. ▪ ***DISABLE_TRACKPAD*** –Disables the trackpad module. • ***TOUCHPAD_TUNER*** – Enables the logic for the trackpad to enter into tuner mode. The pins are available by default at P0[0] and P0[1]. ▪ ***DISABLE_TRACKPAD_SLEEP*** – Defines the DISABLE_TRACKPAD_SLEEP macro for disabling the trackpad sleep. ▪ ***TRACKPAD_TEMPERATURE_VARIATION*** – Enables a logic to maintain the sensor baseline when the device is exposed to sudden temperature variation ▪ ***DISABLE_CUSTOM_POINTER_MOVEMENT –*** Enables the logic to disable the use of the ballistic movement from the *BallisticMovement.a* library. ▪ ***DISABLE_CUSTOM_SCROLL_GESTURE*** – Enables the logic to disable the use of the scroll gesture from the scrollgesture.a library. ▪ ***DISABLE_EDGE_SWIPE_TOP –*** Enables the logic to disable the top-edge swipe gesture. ▪ ***DISABLE_CLICK_DETECTION_POINTER_MOVEMENT –*** Enables the logic to stop the pointer movement when click detection is in progress.

5.1.2 Watchdog Timer Subsystem

The Watchdog Timer subsystem provides a recovery mechanism from any undesired firmware behavior. It resets if the watchdog timer subsystem is not cleared. This subsystem uses the PRoC BLE counter 0 of the watchdog timer hardware block for this functionality. (**Note:** The PRoC BLE watchdog timer has two 16-bit counters (counter 0 and counter 1) and

one 32-bit counter (counter 2). These counters can be configured to work independently or in cascade). In this project, counter 1 is used as a tick timer, which is described separately in the Timer Subsystem section.

Counter 0 is configured to generate a system reset interrupt after three continuous unhandled interrupts. A counter 0 interrupt is generated on reaching the specified terminal count. The terminal count value of the remote control firmware is set to 0xFFFF, which converts to approximately a 2-second time interval. Based on this terminal count setting, a system interrupt is generated approximately 6 seconds after three continuous interrupts are unhandled.

A system reset interrupt will cause the MCU to reset, and firmware execution will start from the beginning. To prevent a system reset, counter 0 must be cleared at least once in a 6-second time period. This is done in firmware in **Device_Active()** and **Device_Low_Power_State()** functions of *device.c*.

5.1.3 BLE Subsystem

The BLE subsystem transmits data provided by the application framework over a Bluetooth Smart link. It implements the HOGP and proximity profile (PXP) for the remote control. Figure 5-4 shows the flow chart for the BLE subsystem. The HOGP provides the following services:

- One HID service for transmitting the mouse, keyboard, multimedia, audio and power reports
- Battery service for transmitting the battery status to the peer device
- Device information service that exposes manufacturer and/or vendor information about a device
- Scan parameter service that enables a GATT client to store the Bluetooth LE scan parameters it is using on a GATT server device so that the GATT server can use the information to adjust the behavior to optimize power consumption and/or reconnection latency

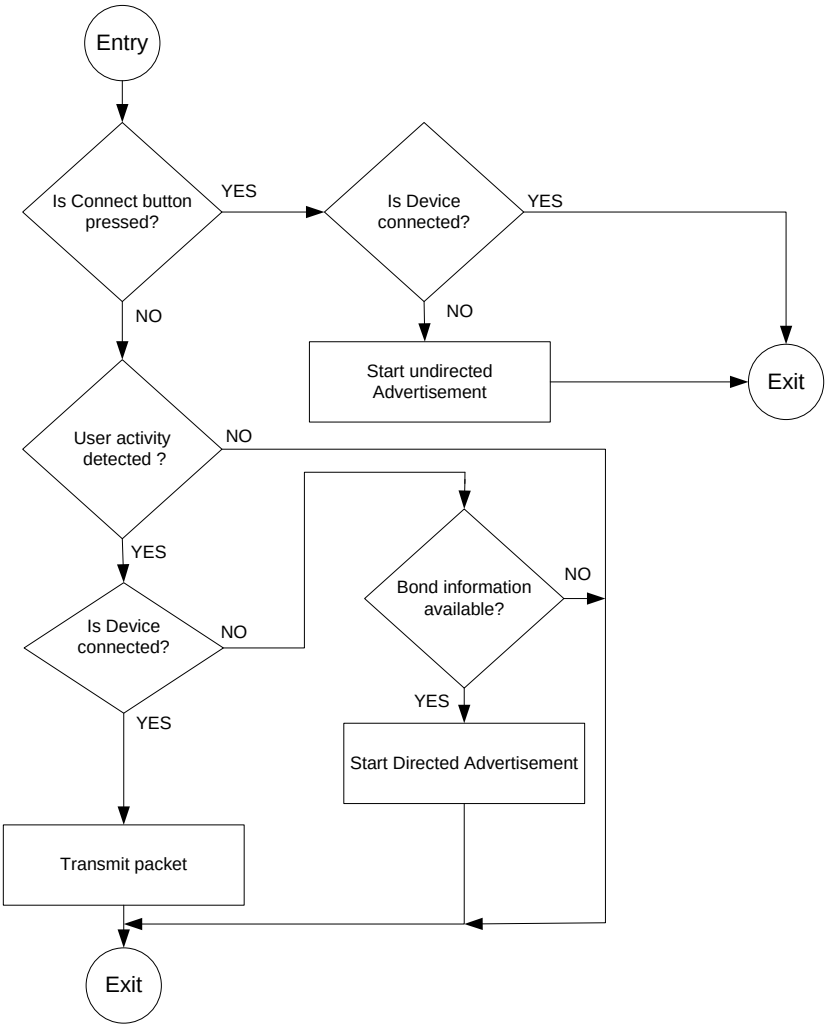
PXP is used to check the link loss status between the remote control and host device. The PXP provides the following services:

- Link Loss Service (LLS) that defines behavior when a link is lost between two devices
- Immediate Alert Service (IAS) that exposes a control point to allow a peer device to cause the device to alert immediately
- Tx Power Service (TPS) that exposes a device's current transmit power level in a connection

The IAS and TPS are used to determine path loss occurrence by monitoring the RSSI.

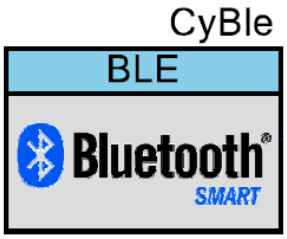
In all the firmware states, the BLE Component connection interval is set to 10 ms, and the slave latency is set to 100 ms. The subsystem enters the Deep Sleep state when it is disconnected or if no peer device is found.

Figure 5-4. BLE Subsystem Flow Chart



The BLE subsystem is built using the BLE Component (Figure 5-5) provided in PSoC Creator.

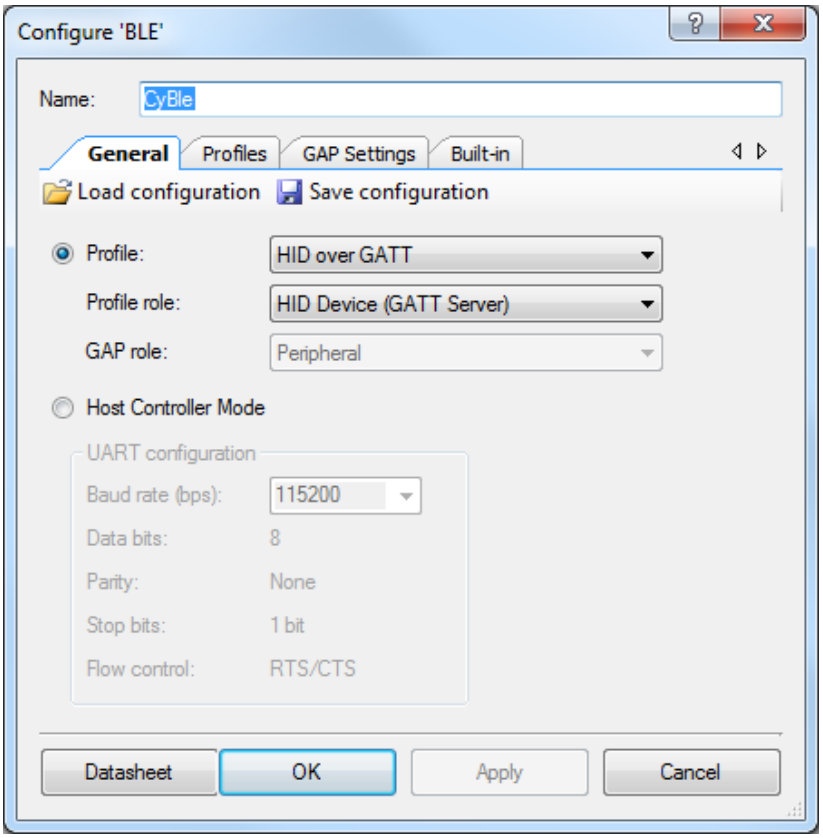
Figure 5-5. BLE Component in PSoC Creator



5.1.3.1 Device Role/Profile Settings

The BLE Component is configured for multiple profiles and roles, as shown in Figure 5-6.

Figure 5-6. General Tab of BLE Component



The settings used for a remote control for the BLE subsystem **General** tab are as follows:

- **Profile:** “HID over GATT” is selected for the remote control, as it is acts as an HID.
- **Profile role:** “HID Device (GATT Server)” is selected for the remote control. It acts as a GATT server, which sends data over notification to the GATT client whenever data is available.
- **GAP role:** “Peripheral” is selected based on the profile and profile role for remote control.

5.1.3.2 Profiles Settings

The **Profiles** tab of the BLE Component is shown in Figure 5-7. It consists of set of services, characteristics, and descriptors, which are described in Table 5-3.

Figure 5-7. Profiles Tab of BLE Component

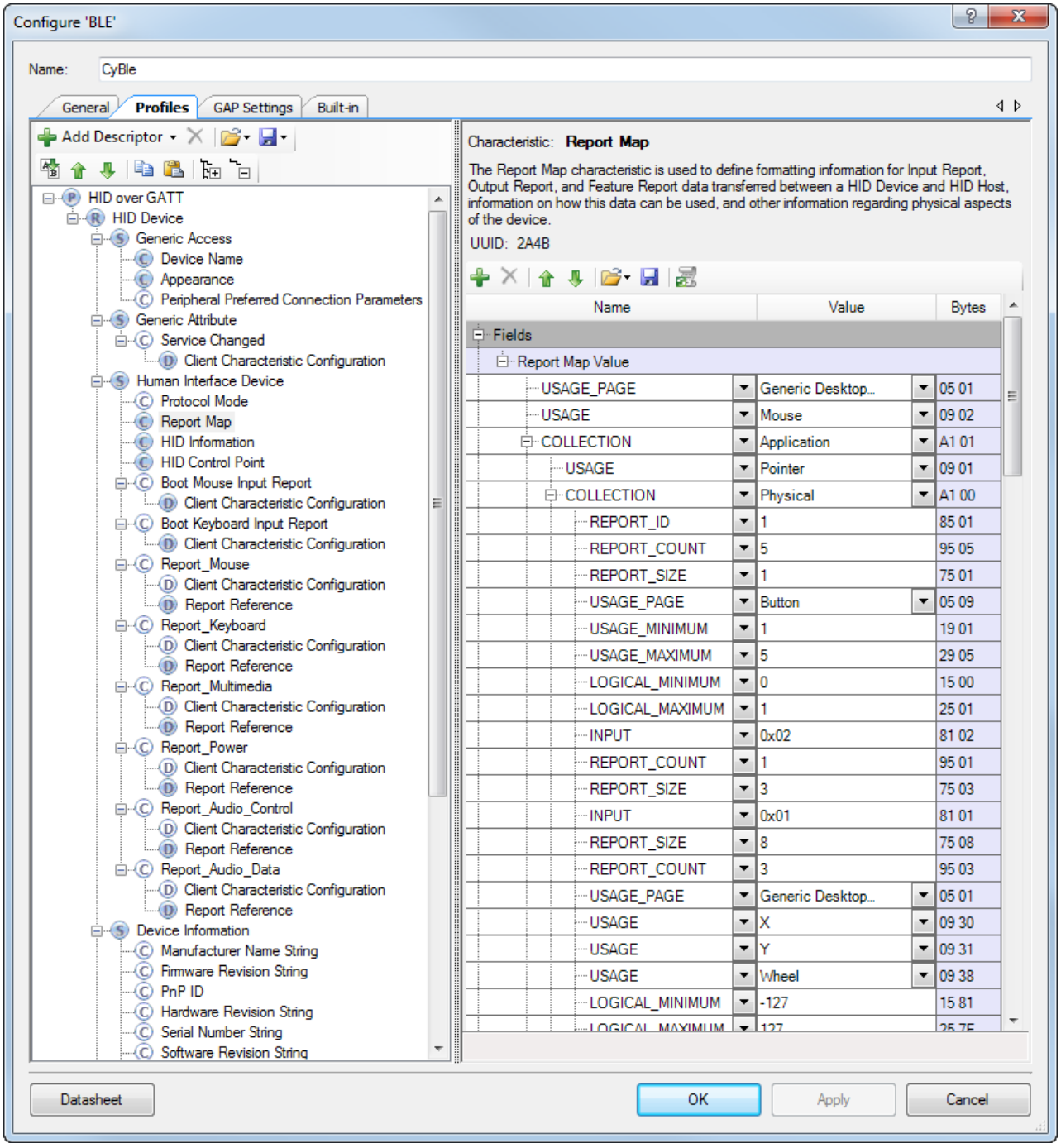


Table 5-3. BLE Component Profile Details

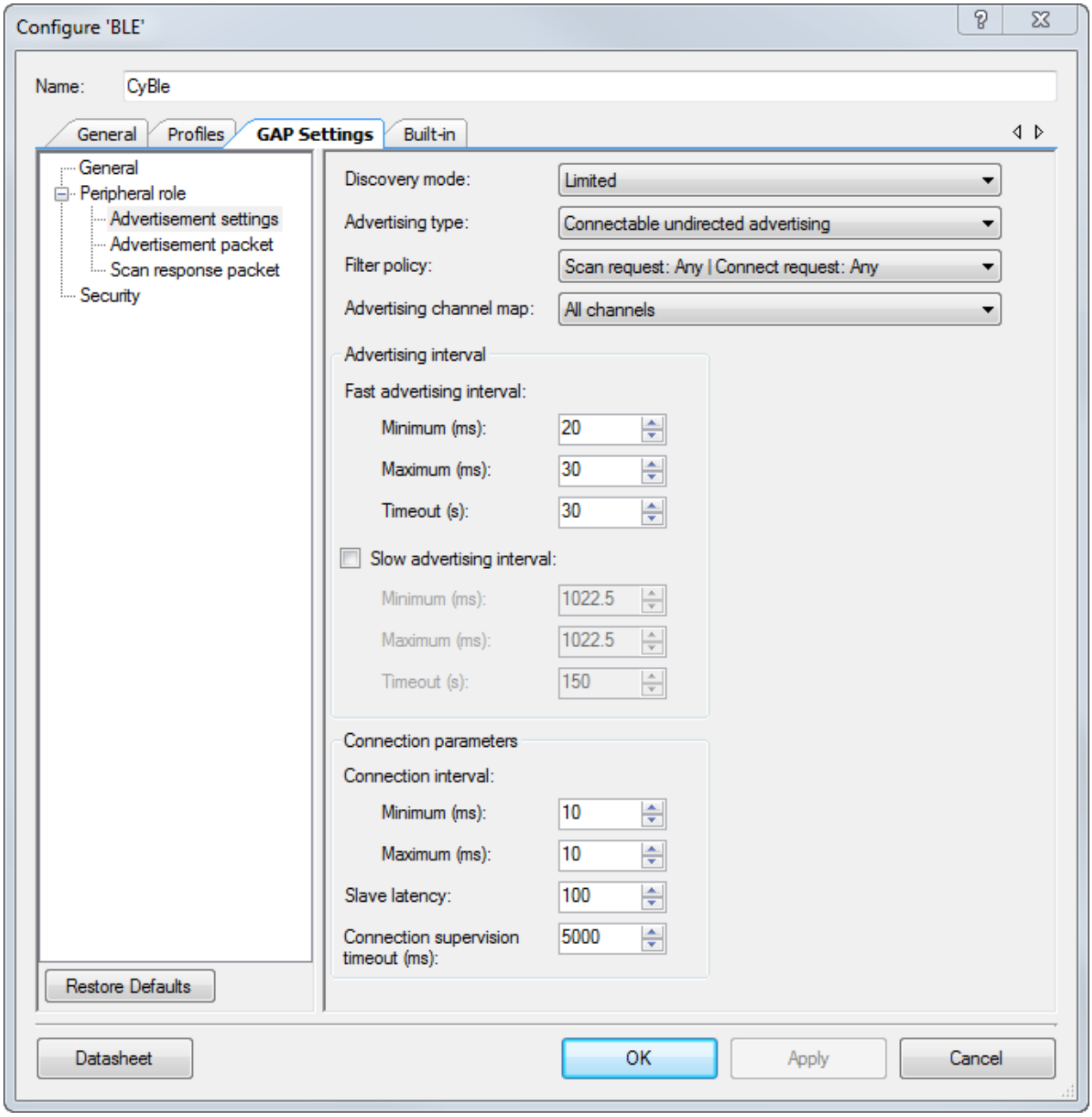
Services	Characteristic	Details
Generic Access Service	Device Name	Configure the value of this characteristic as “CY5672 Remote Control RDK.”
	Appearance	Configure the value of this characteristic as “Human Interface Device (HID).”
	Peripheral Preferred Connection Parameters	Characteristic values are automatically populated by the Component based on advertisement settings configured in the GAP Settings tab of the BLE Component.
Generic Attribute Service	The default BLE configuration of this service is used for remote control.	
Human Interface Device Service	Protocol Mode	The default BLE configuration of this characteristic is used for the remote control.
	Report Map	The Report Map characteristic defines the HID descriptor for the remote control. The report map for the remote control consists of mouse, keyboard, power, multimedia, and audio report structure.
	HID Information	The default BLE configuration of this characteristic is used for the remote control.
	HID Control Point	The default BLE configuration of this characteristic is used for the remote control.
	Boot Mouse Input Report	The default BLE configuration of this characteristic is used for the remote control.
	Boot Keyboard Input Report	The default BLE configuration of this characteristic is used for the remote control.
	Boot Keyboard Output Report	The default BLE configuration of this characteristic is used for the remote control.
	Report Mouse	Modify the read permissions to “Encryption Required” and configure the report ID of the report reference descriptor to “1”. The rest of the parameters are left as the default values set by the BLE Component.
	Report Keyboard	Modify the read permissions to “Encryption Required” and configure the report ID of the report reference descriptor to “2”. The rest of the parameters are left as the default values set by the BLE Component.
	Report Multimedia	Modify the read permissions to “Encryption Required” and configure the report ID of the report reference descriptor to “3”. The rest of the parameters are left as the default values set by the BLE Component.
	Report Power	Modify the read permissions to “Encryption Required” and configure the report ID of the report reference descriptor to “4”. The rest of the parameters are left as the default values set by the BLE Component.
	Report Audio Control	Modify the read permissions to “Encryption Required” and configure the report ID of the report reference descriptor to “31”. The rest of the parameters are left as the default values set by the BLE Component.
	Report Audio Data	Modify the read permissions to “Encryption Required” and configure the report ID of the report reference descriptor to “30”. The rest of the parameters are left as the default values set by the BLE component.
Device Information Service	Manufacturer String Name	Configure the value as “Cypress Semiconductor.”
	Firmware Revision String	Configure the value as “1.0.0.0”
	PnP ID	Configure Vendor ID as “0x4B4,” Product ID as “0x5673,” and Product Version as “0x0001.”
	Hardware Revision String	Configure the value as “1.0”
	Serial Number String	This value is over written by firmware with the BD Address generated using the silicon ID.
	Software Revision String	Configure the value as “PSoC Creator.”
	Model String Number	Configure the value as “CY5672.”
Battery Service	Configure the value as “100”; the rest of the values are the default values of the BLE Component.	

Scan Parameter Service	The default BLE configuration of this service is used for the remote control.
Link Loss Service	The default BLE configuration of this service is used for the remote control.
Immediate Alert Service	The default BLE configuration of this service is used for the remote control.
TX Power Service	Configure the Tx Power Level as “0”. The remaining default BLE configuration of this service is left as the default values.

5.1.3.3 GAP Settings

The BLE Component supports different types of discovery modes. This application uses the limited discovery mode so that battery power is not wasted when the user presses the advertisement button. The advertising type is chosen to connect to any host that is HOGP-compliant. In the case of the remote control, the report rate expected is approximately 100 Hz. This means that the data must be sent out approximately every 10 ms. Thus, the firmware maintains the connection interval at 10 ms. Figure 5-9 shows the advertisement settings, and Figure 5-9 shows the security options chosen for the remote control firmware.

Figure 5-8. GAP Settings Tab of BLE Component Showing Advertisement Settings

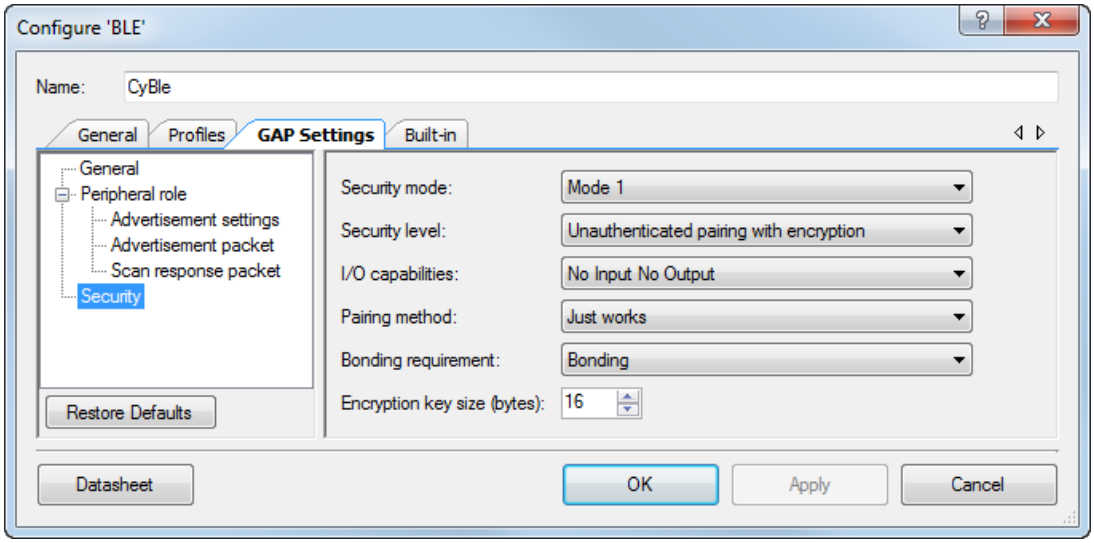


The settings used for a remote control for the BLE subsystem **GAP Settings** > **Advertisement settings** are as follows:

- **Discovery mode:** This mode is used by devices that need to be discoverable only for a limited period of time or for a specified event. The timeout duration is defined by the applicable advertising timeout parameter. The remote control configures this parameter as “Limited” so that it advertises for a limited period of time when the connect button is pressed.
- **Advertising type:** This parameter defines the advertising type to be used by the link layer of PRoC BLE. The remote control configures it as “Connectable undirected advertising,” which allows any other device to connect to it.
- **Filter policy:** This parameter defines how the scan and connection requests are filtered. The remote control configures it as ”Scan request: Any | Connection request: Any,” which allows it to process connection requests and scan requests from all devices.

- **Advertising channel map:** This parameter is used to enable a specific advertisement channel. The remote control configures it as “All channels” so that it allows advertisement on all three possible BLE advertisement channels.
- **Advertising interval:** This parameter defines the interval between two advertising events.
- **Fast advertising interval:** This advertisement interval results in a faster connection.
  - **Minimum (ms):** This parameter defines the minimum interval for advertising the data and establishing a connection. It is configured to increment in multiples of 0.625 ms. The valid range is from 20 ms to 10240 ms. A minimum value of “20” is configured for the remote control.
  - **Maximum (ms):** This parameter defines the maximum interval for advertising the data and establishing a connection. It is configured to increment in multiples of 0.625 ms. The valid range is from 20 ms to 10240 ms. A maximum value of “30” is configured for the remote control.
  - **Timeout (s):** This parameter defines the timeout value of advertising with fast advertising interval parameters. A timeout interval of “30” is configured for the remote control.
- **Connection parameters:** These parameters define the connection event timing for a central device communicating with the remote control. Consecutive connection events are separated by the defined **Connection interval.**
  - **Minimum (ms):** This parameter is the minimum permissible connection time value to be used during a connection event. It is configured in steps of 1.25 ms. The range is from 7.5 ms to 4000 ms. A minimum connection interval of “10” is configured for the remote control to achieve a report rate above 100 Hz.
  - **Maximum (ms):** This parameter is the maximum permissible connection time value to be used during a connection event. It is configured in steps of 1.25 ms. The range is from 7.5 ms to 4000 ms. A maximum connection interval of “10” is configured for the remote control to achieve a report rate above 100 Hz.
- **Slave latency**: This parameter defines the latency of the slave in responding to a connection event in consecutive connection events. It is expressed in terms of multiples of connection intervals, where only one connection event is allowed per interval. The range is from 0 to 499 events. A slave latency of “100” is configured for the remote control.
- **Connection supervision timeout (ms):** This parameter defines the link supervision timeout interval. It defines the timeout duration for which a link needs to be sustained in case of no response from a peer device over the LE link. The time interval is configured in multiples of 10 ms. The range is from 100 ms to 32000 ms. A connection supervision timeout of “3000” is configured for the remote control.

Figure 5-9. GAP Settings Tab of BLE Component Showing Security Options



The settings used for a remote control application for the BLE subsystem **GAP Settings** > **Security** are as follows:

- **Security mode:** This parameter defines the GAP security modes for the Component. This mode is configured as “Mode 1” for the remote control. Mode 1 is chosen when data encryption is required.
- **Security level:** This parameter is configured as “Unauthenticated pairing with encryption.” With this level of security, the remote control will send encrypted data after establishing a connection with the client device.

- **I/O capabilities:** This parameter refers to the device's input and output capability that can enable or restrict a particular pairing method or security level. It is configured as “No Input No Output,” as the remote control does not have the capability to enter and display authentication key data.
- **Pairing method:** This parameter is configured as “Just works” for the remote control. The device will use the simple pairing procedure without authentication. With this method, transferred data is vulnerable to “man in the middle” attacks.
- **Bonding requirement:** This parameter is used to configure the bonding requirements. The purpose of bonding is to create a relation between two Bluetooth devices based on a common link key (a bond). The link key is created and exchanged (pairing) during the bonding procedure and is expected to be stored by both Bluetooth devices, to be used for future authentication. This parameter is configured as “Bonding” for the remote control. The remote control will store the link key of a connection after pairing with the client device. It uses a previously stored key for the connection when the connection is lost and then re-established.
- **Encryption key size (bytes):** This parameter defines the encryption key size based on the profile requirement. The valid values of encryption key size are 7 to 16 bytes. This parameter is configured as “16” for the remote control.

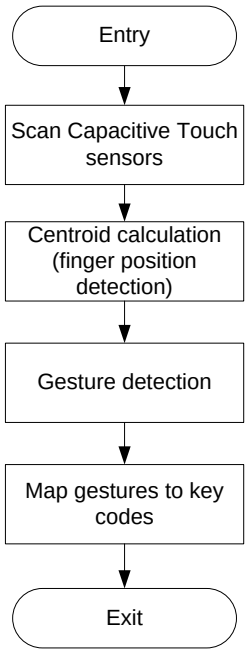
The datasheet details the APIs supported by the BLE Component. To go to the datasheet, click the **Datasheet** button at the bottom left corner of the BLE Component GUI, as shown in Figure 5-9.

5.1.4 Trackpad Subsystem

The trackpad for the remote control is composed of eight rows and eight columns of sensors. The firmware detects and generates key codes for the different gestures and reports them to the application framework.

The CapSense Gesture Component provided in PSoC Creator supports capacitive touch sensor (trackpad) scanning, centroid calculation, and gesture detection. It provides an API for each of these functionalities. The trackpad subsystem calls the relevant APIs to detect user activity. The trackpad subsystem is executed every 10 ms in the Active state, every 125 ms in the Idle state, and every 250 ms in the Sleep state. The flow chart in Figure 5-10 shows the how the trackpad subsystem operates.

Figure 5-10. Trackpad Subsystem Flow Chart



- **Scan:** Detecting the trackpad activity involves scanning the capacitive touch sensors. The presence of a finger on the trackpad will result in non-zero values for signals on those particular sensors.
- **Centroid calculation:** Signal values from the scanning stage are used as inputs for the centroid calculation algorithm. This algorithm estimates the total number of fingers and their position on the trackpad.

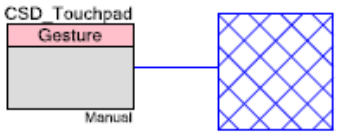
- **Gesture detection:** The number of fingers and finger positions calculated in the centroid calculation stage are used as inputs to the gesture detection algorithm. Table 5-4 shows the gestures detected by firmware and the corresponding key codes sent over BLE.

Table 5-4. List of Supported Gestures

Gesture	Action Performed	Key Code
Left-click	Single-finger single tap	Left-click (0th bit in 0th byte of the mouse report)
Right-click	Two-finger single tap	Right-click (1st bit in 0th byte of the mouse report)
Double-click	Single-finger double tap	Left-click will be sent twice (0th bit in 0th byte of the mouse report)
Pointer movement	Single-finger movement on trackpad area	X, Y (1st and 2nd byte of the mouse report)
Vertical scroll	Single-finger movement in a top/down motion on right edge	Z-wheel (3rd byte of the mouse report)
Horizontal scroll	Single-finger movement in a left/right motion on bottom edge	H-wheel (4th byte of the mouse report)
Home screen	Single-finger swipe in top/down direction starting from top edge	Window key (3rd bit of the 0th byte of the keyboard report)
Zoom in/out	Two-finger pinch	Ctrl key (0th bit of the 0th byte of the keyboard report) + Z-wheel (fourth byte of the mouse report)

Figure 5-11 shows the CapSense Gesture Component. The following sections describe its configuration for the remote control.

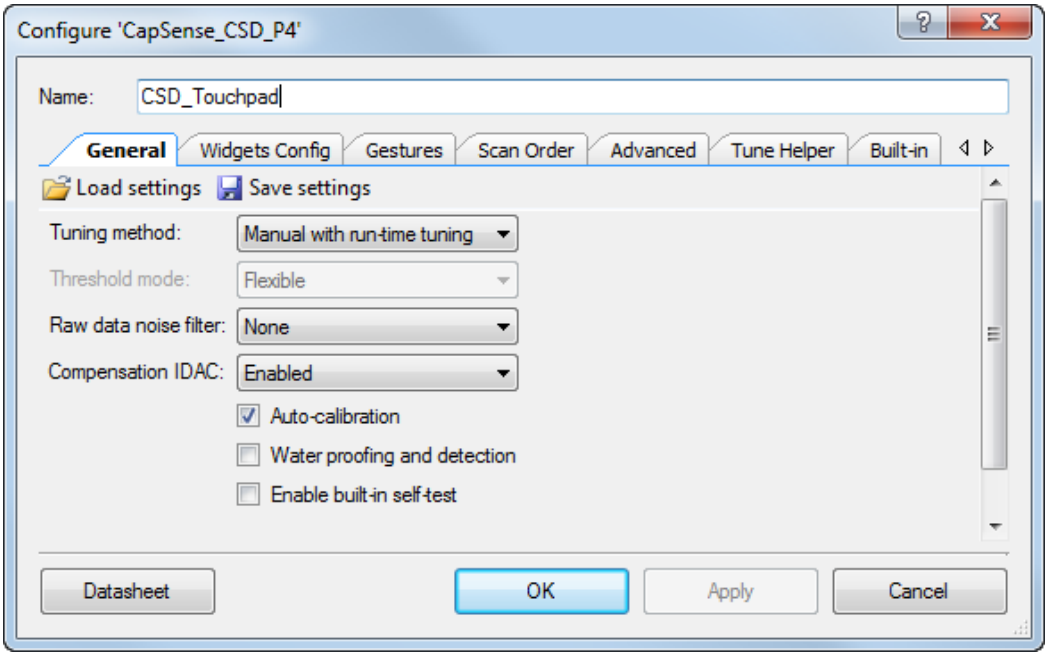
Figure 5-11. CapSense Gesture Component in PSoC Creator



5.1.4.1 General Tab

Figure 5-12 shows the **General** tab of the CapSense Gesture Component.

Figure 5-12. General Tab of CapSense Gesture Component

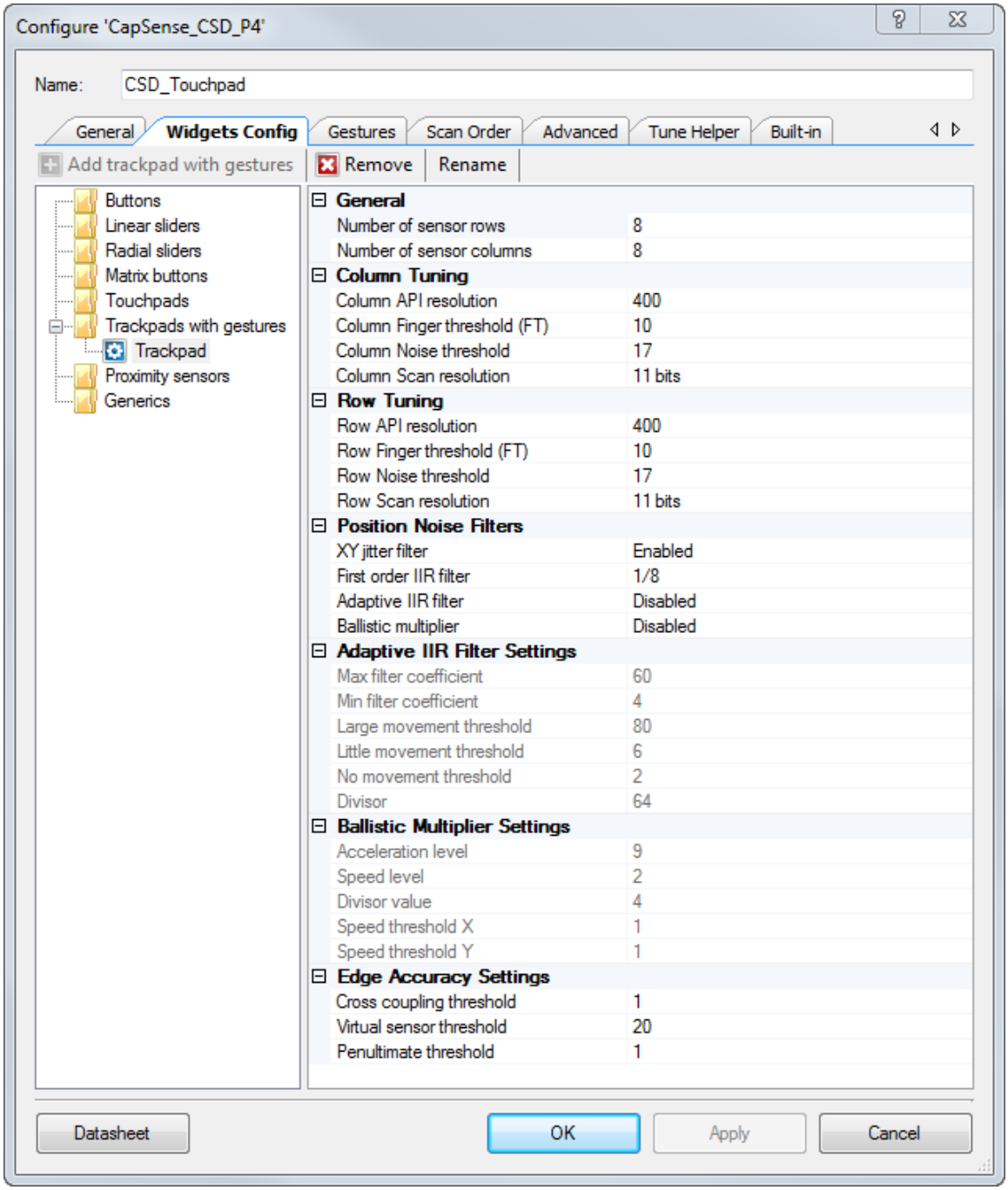


- **Tuning method:** This parameter is configured as “Manual with run-time tuning” for the remote control. This option allows run-time tuning of the CSD_Touchpad Component.
- **Raw data noise filter:** This parameter selects the raw data filter. Only one filter can be selected, which is applied to all sensors. The filter is used to reduce the effect of noise during sensor scans. This parameter is configured as “None” for the remote control.
- **Compensation IDAC:** This mode provides increased sensitivity and SNR. The Compensation IDAC is connected to the AMUX bus full-time during CapSense operation and is intended to compensate for the sensor’s parasitic capacitance. This parameter is configured as “Enabled” for the remote control.
- **Auto calibration:** This option allows IDAC auto-calibration, and it is enabled for the remote control.

5.1.4.2 Widgets Config Tab

The **Trackpad with gestures** widget is added as shown in Figure 5-13.

Figure 5-13. Widgets Config Tab of CapSense Gesture Component

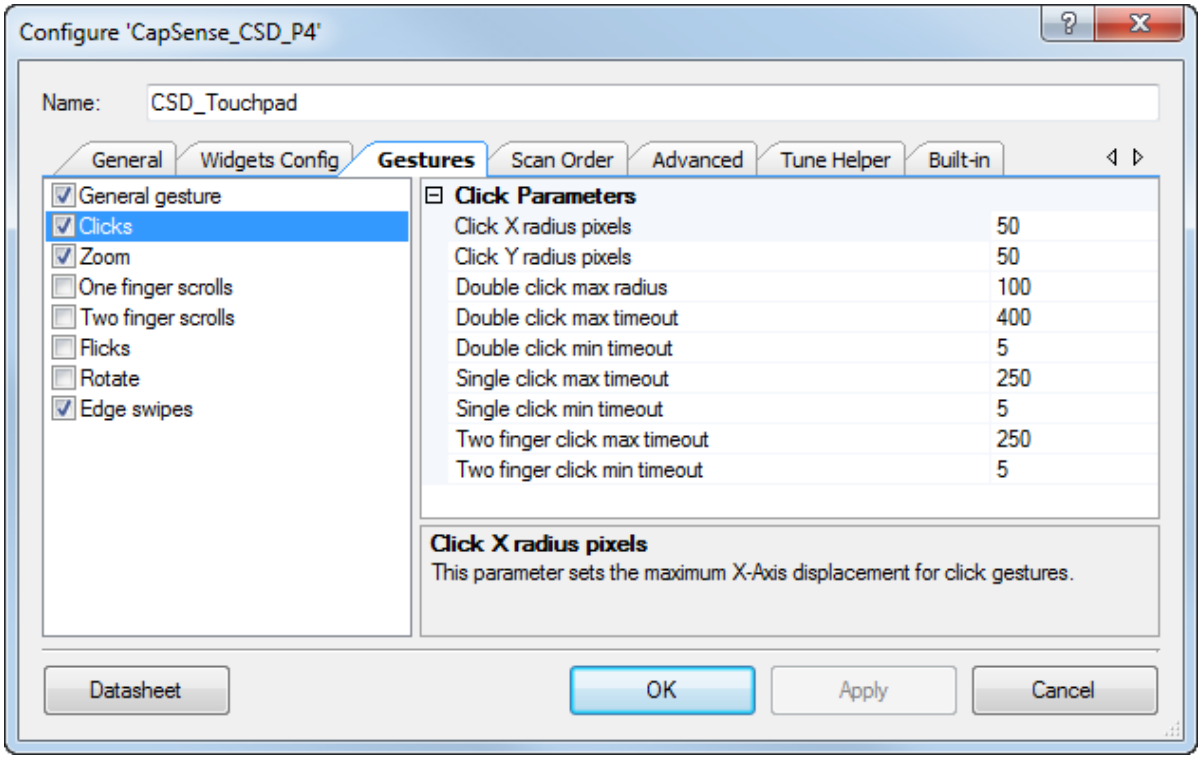


- **Number of sensor rows/Number of sensor columns:** These parameters depend on the number of sensors along the X and Y axis. Their values are set based on the trackpad design. These parameters are configured as “8” rows and “8” columns for the remote control.
- **Column Tuning/Row Tuning:** The tuning parameters for the remote control are selected to achieve optimum touch performance.

5.1.4.3 Gestures Tab

The **General gesture**, **Clicks, Zoom** and **Edge swipes** options are selected. The configuration of these gesture parameters shown in Figure 5-14 is the default.

Figure 5-14. Gestures Tab of CapSense Gesture Component

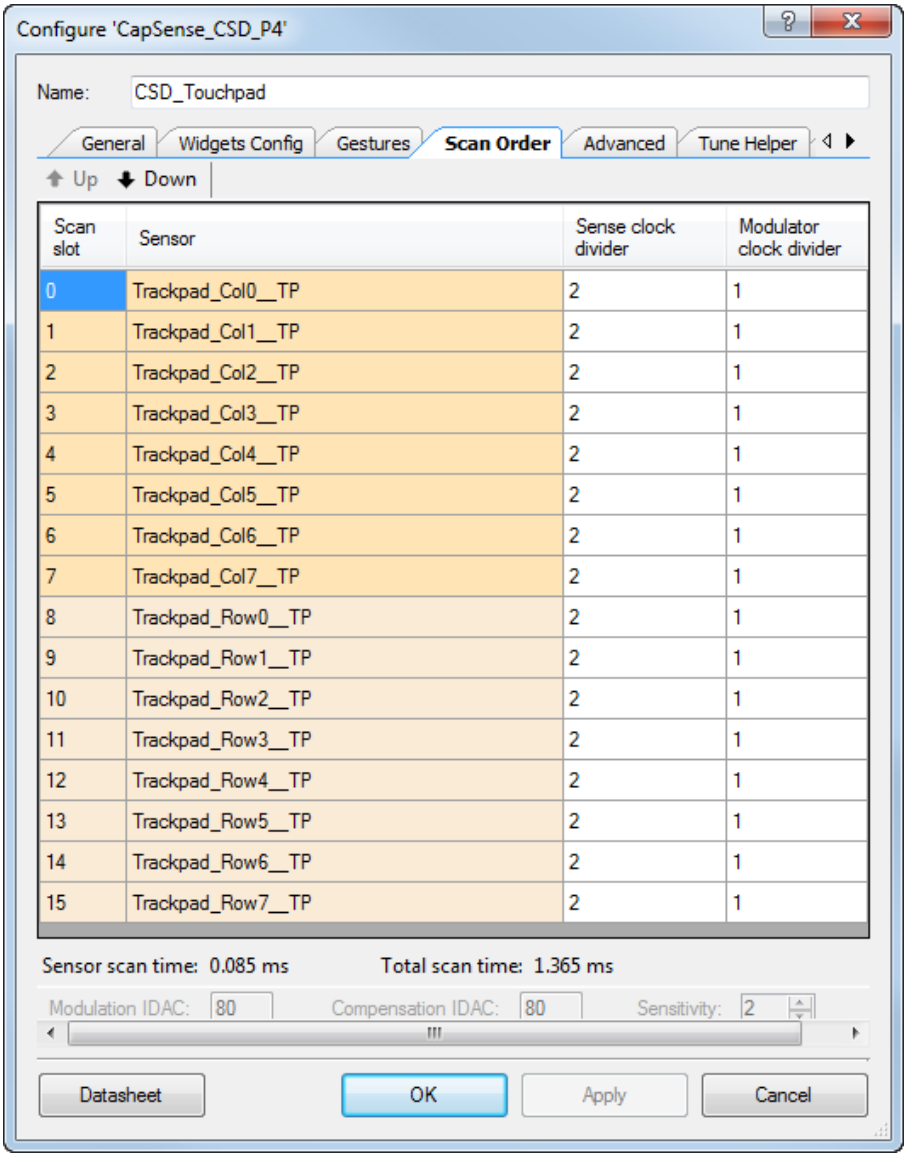


5.1.4.4 Scan Order Tab

The settings in the **Scan Order** tab are shown in Figure 5-15. The sense clock divider is modified to “2”. All other parameters are left at their default settings.

The datasheet provides a detailed table for the selection of the sense clock divider in the **Sense clock divider** section. To go to the datasheet, click the **Datasheet** button at the bottom left corner of the CapSense Gesture Component GUI.

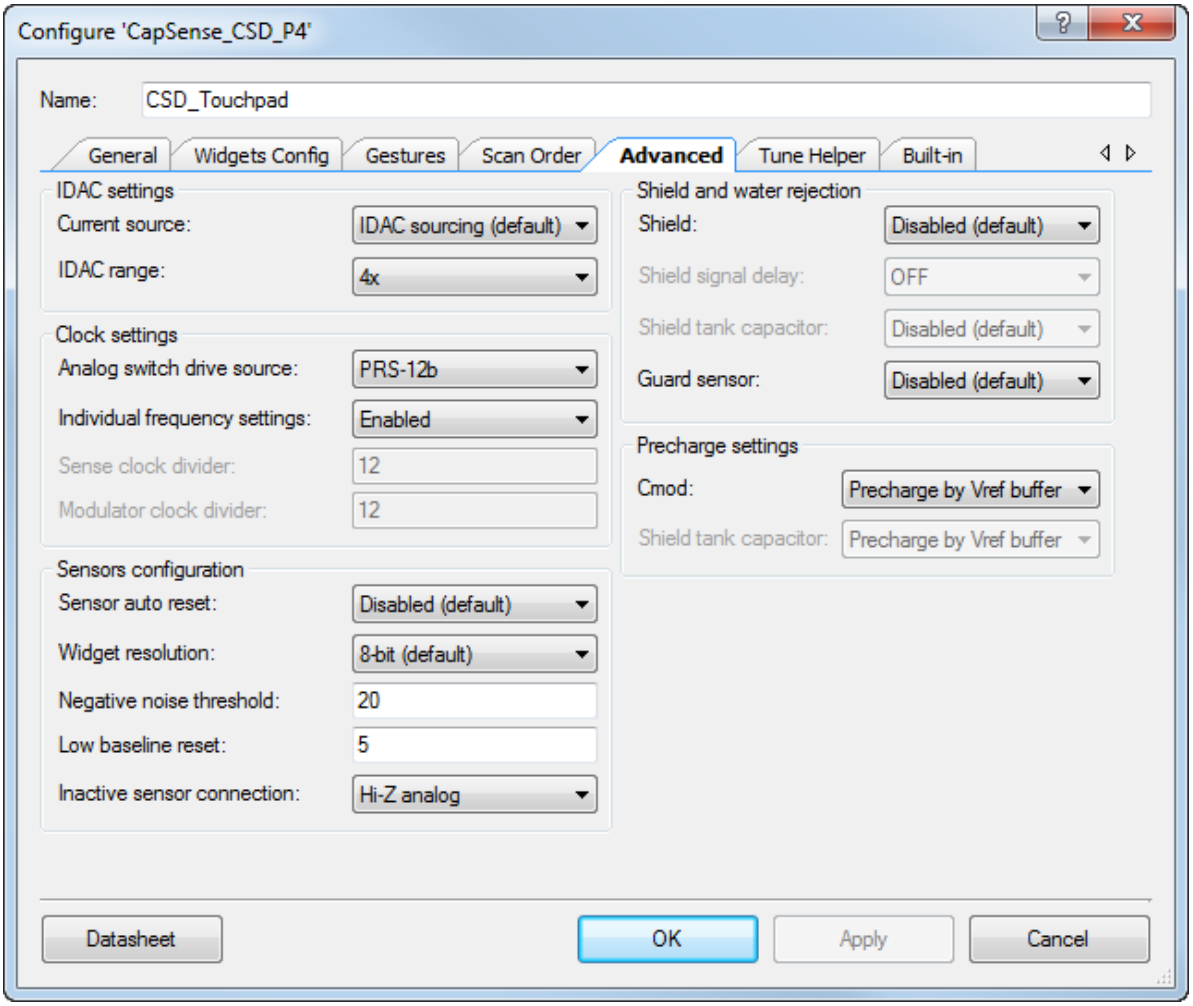
Figure 5-15. Scan Order Tab of CapSense Gesture Component



5.1.4.5 *Advanced Tab*

The settings in this tab are not modified; that is, the default settings are retained, as shown in Figure 5-16.

Figure 5-16. Advanced Tab of CapSense Gesture Component

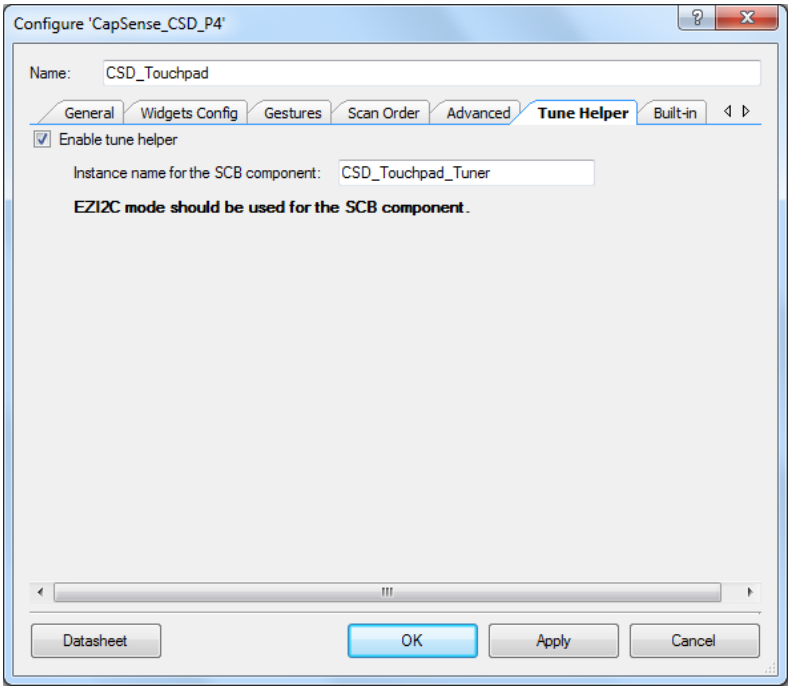


5.1.4.6 *Trackpad Subsystem Tuning*

The trackpad of the remote control RDK is already tuned; therefore, it is not necessary to repeat the tuning exercise. Tuning of the trackpad is required during the development phase. Every new design needs tuning to ensure optimal performance. The following steps provide guidance for connecting to the tuner.

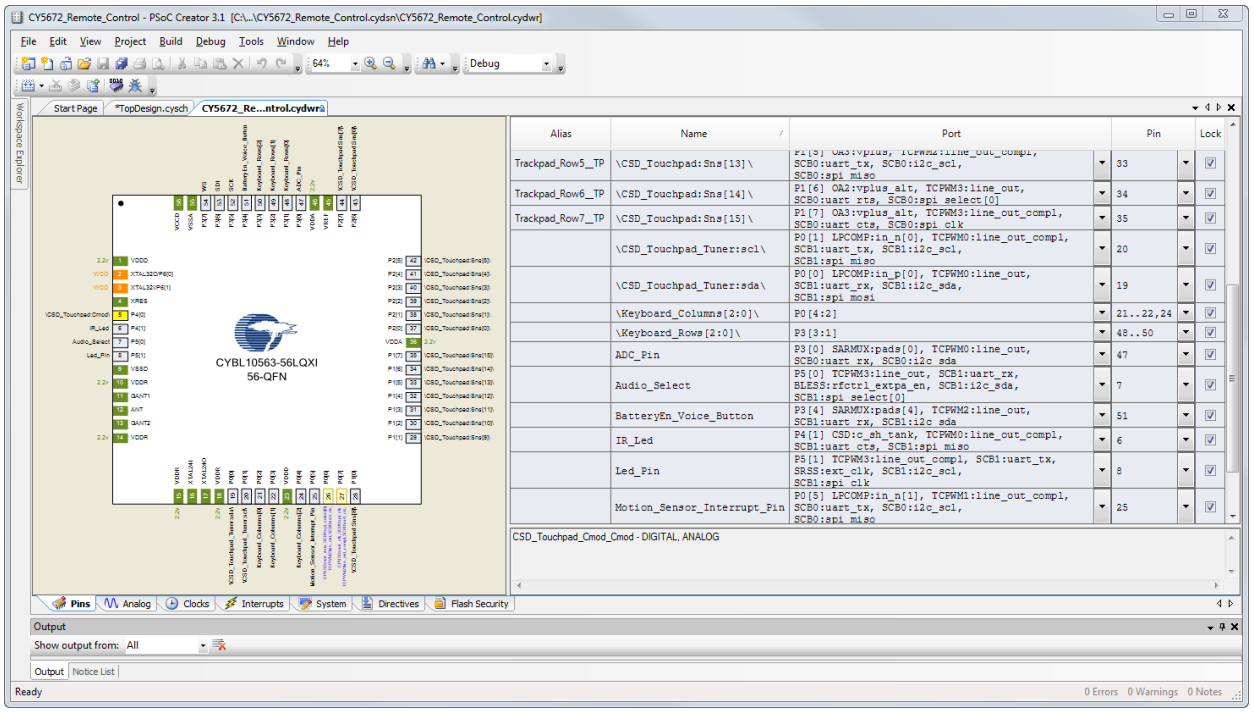
1. Enable the TOUCHPAD_TUNER macro in the *platform.h* file.
2. Enable the DISABLE_MOTION_SENSOR macro in the *platform.h* file.
3. Right-click on the “CSD_Touchpad_Tuner” Component and select “Enable” to enable the Component.
4. Right-click on the “Motion_Sensor_I2C” Component and select “Disable” to disable the Component.
5. Click the **Tune Helper** tab of the CSD_Touchpad Component and select the “Enable tune helper” option, as shown in Figure 5-17 .

Figure 5-17. Tune Helper Tab of CapSense Gesture Component



6. The CSD_Touchpad_Tuner:scl and CSD_Touchpad_Tuner:sda pins appear in the *CY5672_Remote_Control.cydwr*, as shown in Figure 5-18. Verify that the pins are assigned as P0[0] for SDA and P0[1] for SCL.

Figure 5-18. CY5672_Remote_Control.cydwr view

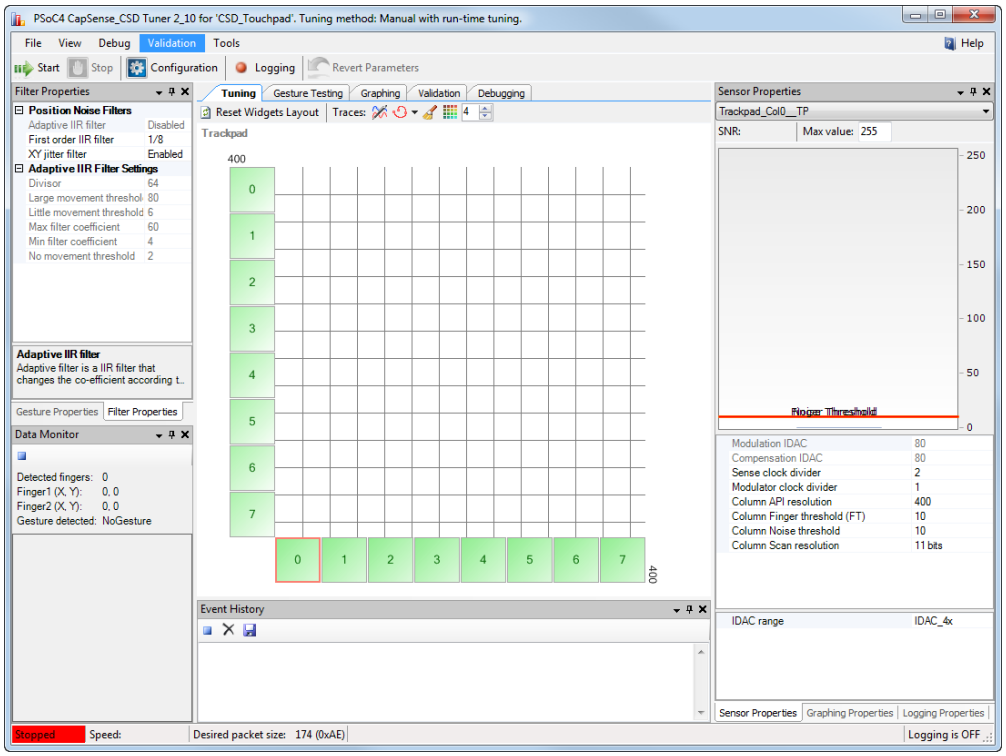


7. Rebuild the remote control by choosing **Build** > **Clean and Build CY5672_Remote_Control**. Program the hex into the remote control.

8. Solder four wires from the remote control to a five-pin male connector in the following order: VDD (TP7), GND (TP8), no connect, Pin0[1], and Pin0[0].

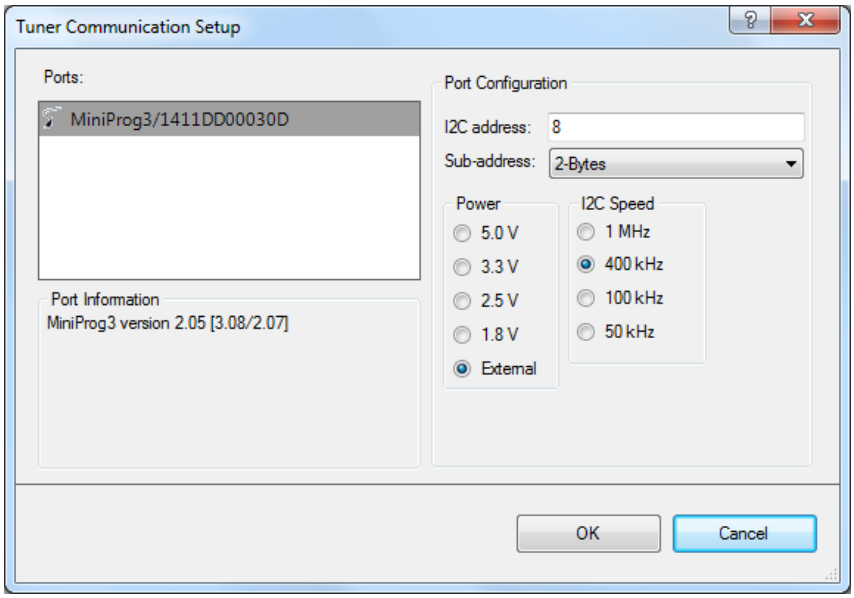
9. Connect the five-pin male header to the five-pin female header of MiniProg3.
10. Connect MiniProg3 to a PC or laptop using a USB cable.
11. Right-click on the CSD_Touchpad Component and select “Launch Tuner.”
12. The tuner tool pops up, as shown in Figure 5-19.

Figure 5-19. Tuner Tool



13. Click the **Configuration** button to invoke the **Tuner Communication Setup** window. Configure the settings as shown in Figure 5-20.

Figure 5-20. Tuner Communication Setup



14. Ensure that two AAA batteries are placed in the remote control.
15. Click the **Start** button and observe that tuner starts running. For more information on how to use the tuner GUI, refer to the PSoC 4 CapSense Tuning Guide.

The datasheet provides the APIs supported by the CapSense Gesture Component. It also provides documentation on all the configurable parameters mentioned previously in this section. To go to the datasheet, click the **Datasheet** button at the bottom left corner of the CapSense Gesture Component GUI, as shown in Figure 5-16.

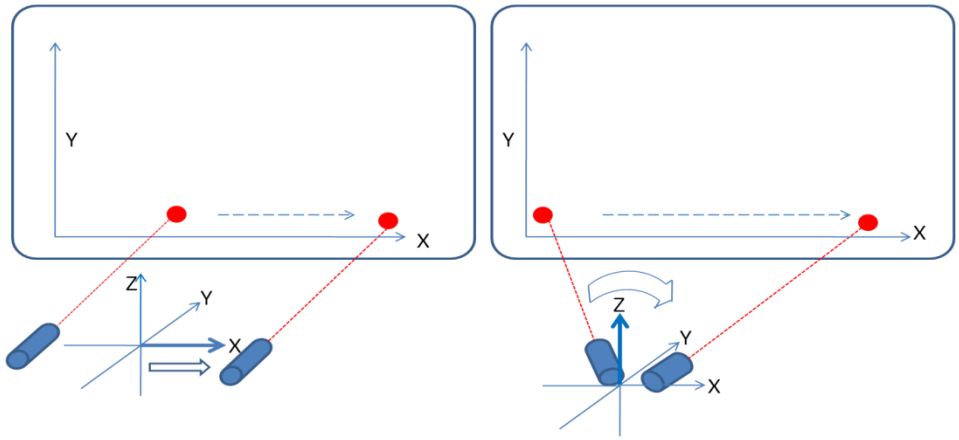
5.1.5 Motion Sensor Subsystem

5.1.5.1 Motion Sensor Usage in the Remote Control RDK

The motion sensor subsystem implements the motion mouse feature in the remote control. The firmware detects the direction and magnitude of movement of the remote control and converts the device movement to mouse pointer movement. The change in mouse pointer position is sent to the BLE host through the HID report. The requirement of the motion mouse is to enable the user to use the remote for moving the mouse pointer through his or her gesture. It is similar to using a laser pointer (instead of a trackpad mouse) to point at a location on a screen. Now imagine how a laser pointer produces movement onscreen.

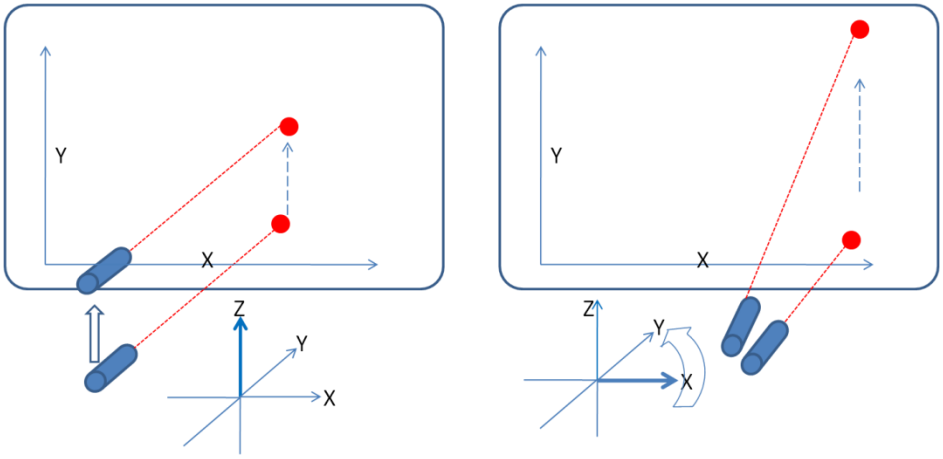
To move the laser pointer in the X direction of the screen, the presenter/user will either move himself laterally in the X direction of the device or rotate the pointing device around the Z axis of the device, as shown in Figure 5-21.

Figure 5-21. Moving Laser Pointer in X Direction



Similarly, to move the laser pointer in the Y direction of the screen, the presenter/user will either move himself laterally in the Z direction of the device or rotate the pointing device around the X axis of the device, as shown in Figure 5-22.

Figure 5-22. Moving Laser Pointer in Y Direction



Practically, the user will remain steady in one place and use rotation of the device to move the pointer rather than moving around to laterally shift the device. Hence, while implementing the similar operation in the remote control RDK, you want only the rotation to control the pointer movement, and not the shift in position of the device.

Note that with the PRoC BLE remote control, the later shift also produces a pointer movement, but this is the result of unintended, unnoticeable rotation produced due to imperfect motion caused by the hand movement.

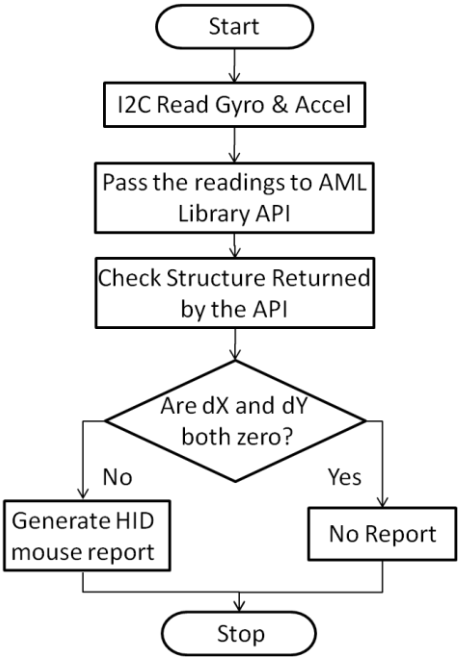
Thus the objectives can be summarized as follows:

- Measure the angle of rotation of the remote about the Z-axis of the remote.
- Map it to the pointer movement on the screen in the X direction of the screen.
- Measure the angle of rotation of the remote about the X-axis of the remote control.
- Map it to the pointer movement on the screen in the Y direction of the screen.

The motion sensor used in the remote control RDK for these measurements is MPU6500 from Invensense. It has three axis accelerometers and three axis gyroscopes. The output of the six sensors is read by PRoC BLE using the I²C bus. The output of the accelerometers and gyroscopes is 16-bit signed data for each sensor. The sampled output data from the motion sensor is converted to the mouse pointer movement in the form of a change in mouse pointer position in the X and Y direction of the screen.

A third-party library from Invensense, called the Air Motion Library (AML), is used to process the accelerometer and gyroscope data, as shown in Figure 5-23. The data from the motion sensor is polled every connection interval and passed to the AML processing API. The API returns the delta X and delta Y for the mouse pointer, which are used to produce the mouse HID report. The polling routine is performed every 10 ms, as the connection interval is set to 10 ms for the remote. Thus, the sampling frequency of the motion sensor is set to 100 samples per second.

Figure 5-23. Processing of Accelerometer and Gyroscope Data



5.1.5.2 *Interfacing the Motion Sensor*

The MPU6500 motion sensor provides two options for a communication interface: I²C and SPI. In the Remote RDK, the I²C interface is used for the communication. The reasons for choosing I²C are as follows:

- I²C needs only two I/O s.
- The Sampling speed required for the application is 100 samples per second, which is easily sufficed by I²C data rate.

The I2C Component in PSoC Creator is to be configured for the I²C communication with MUP6500. Currently, the design uses a 100-kHz I²C clock. The figure below shows how the I2C Component is configured.

Figure 5-24. Configuration Tab of I2C Component Configuration

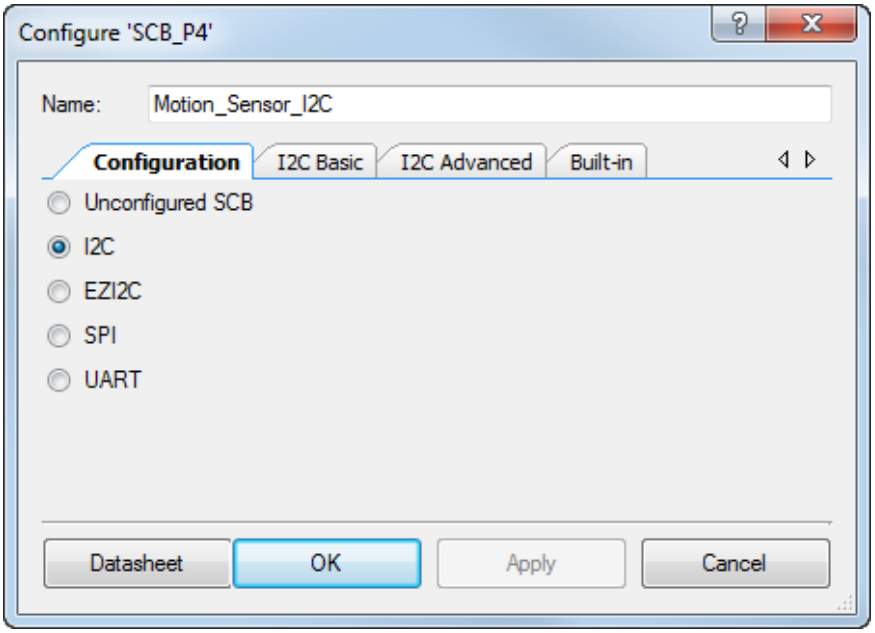


Figure 5-25. I2C Basic Tab of I2C Component Configuration

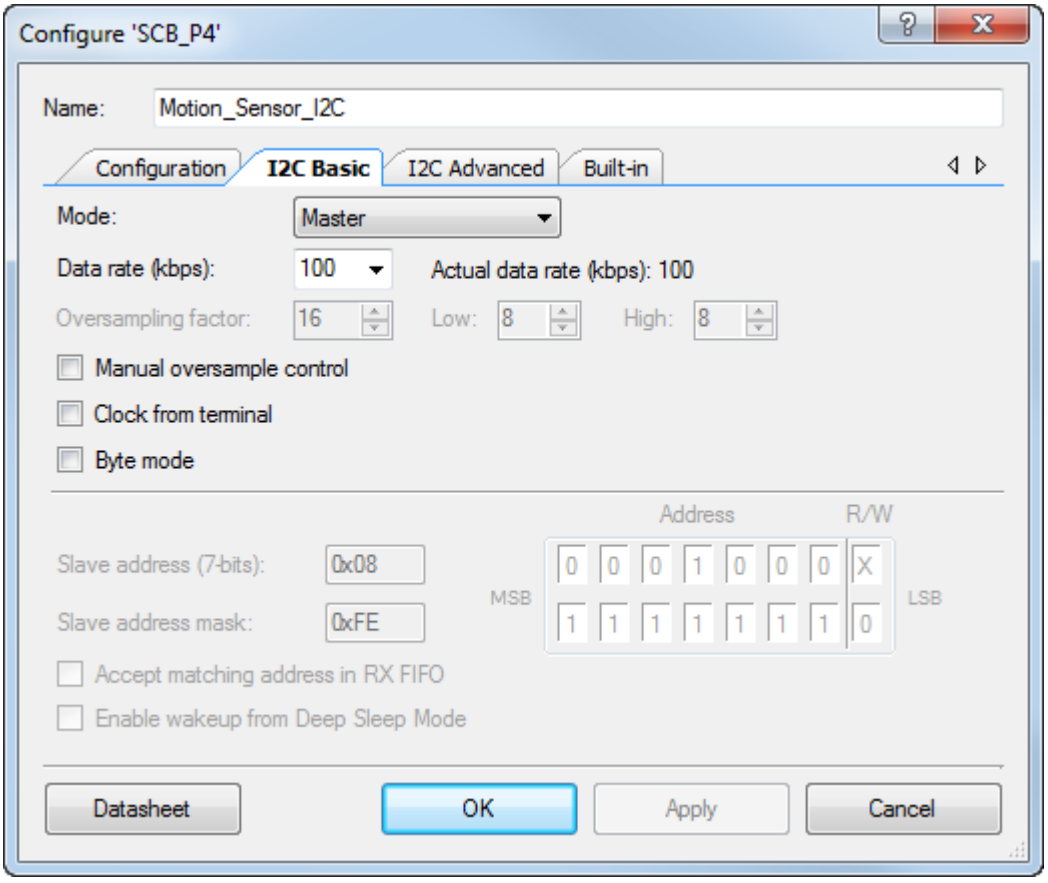
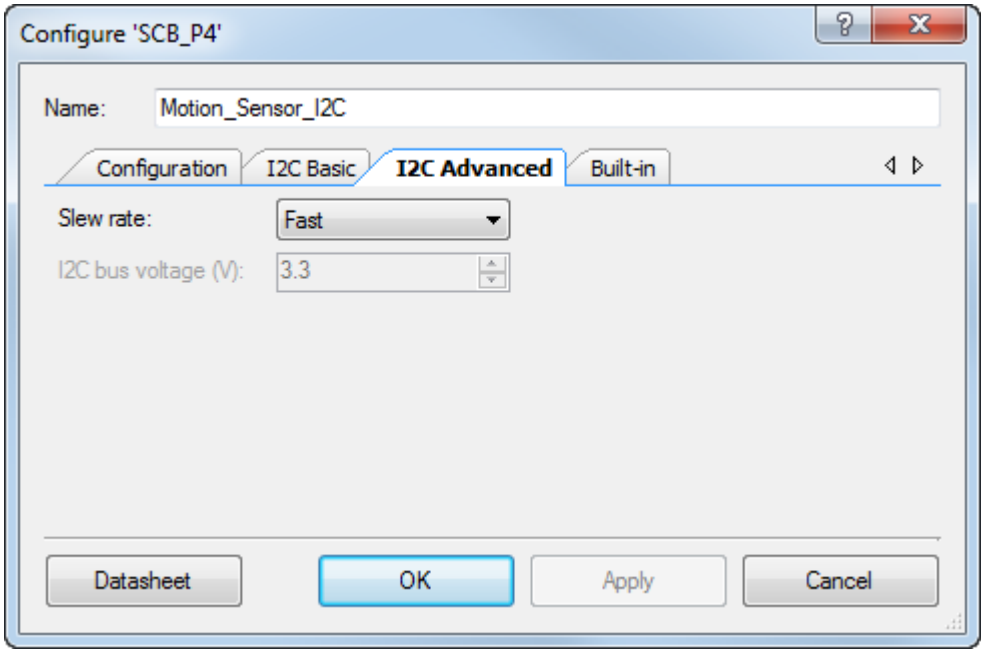
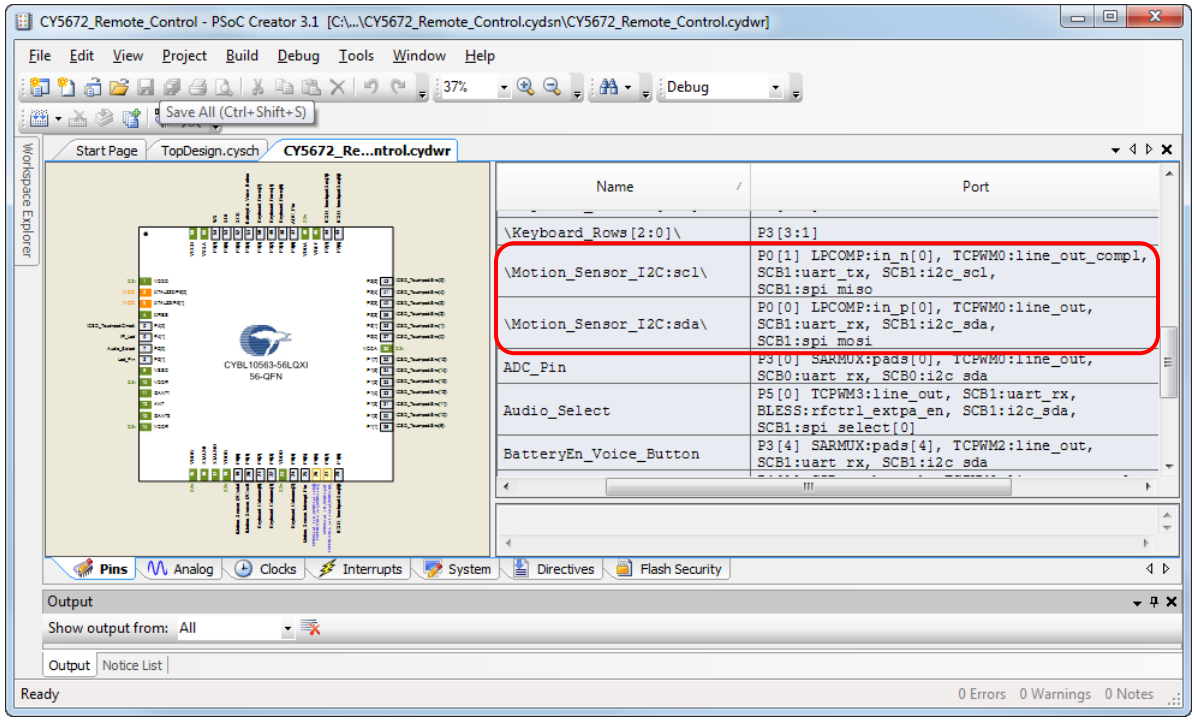


Figure 5-26. Advanced Tab of I2C Component Configuration



After configuring the I²C Component, allocate the SCL and SDA lines to specific pins of PRoC. The I/O Allocation is done in the Pins tab of .cydwr file as shown below.

Figure 5-27. Pins Tab of .cydwr File Used to Select I/Os for I²C



Note that only a few specific pairs of I/Os can be routed to I²C lines. If some other I/Os are allocated for the I²C lines, the build process would give an error stating that those I/O could not be routed in the design. The INT pin of MPU6500 is connected to P0.5 to trigger an interrupt. The INT pin can be used in two ways as follows:

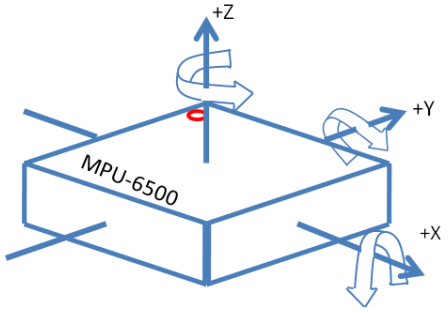
1) The MPU6500, when configured in Low Power Motion detect mode, any motion of the device will assert a level triggered interrupt on INT pin of MPU6500. On the PRoC side, this will cause an interrupt on P0.5 and will wake up the PRoC if it is in deep sleep mode and trigger an interrupt and its ISR will be serviced. The ISR would lead to reading of sampled sensor data which will clear the INT pin of MPU6500.

2) When the MPU6500 is in normal sampling mode, the availability of samples in FIFO will trigger a level-trigger interrupt on INT pin of MPU6500. On PRoC interrupt on P0.5 will be disabled and will be polled as a input signal. If the pin is asserted, the polling routine will read the samples in FIO and the interrupt on INT pin of MPU6500 will be cleared.

5.1.5.3 Using AML: Hardware Abstraction Level

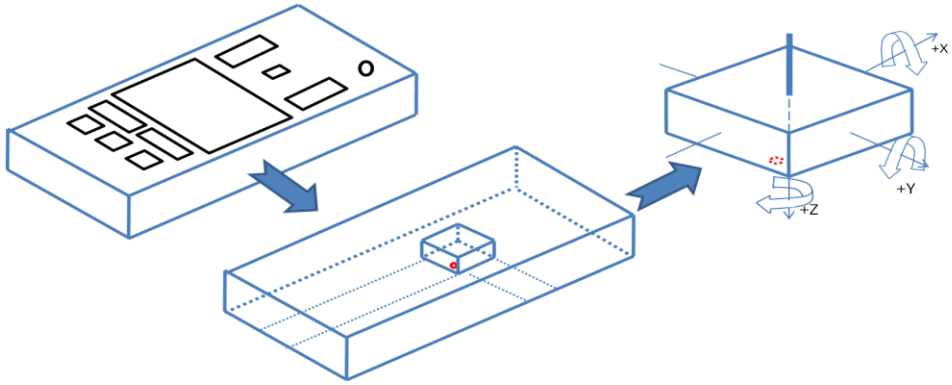
Figure 5-28 shows the motion sensor axis orientation with respect to the chip packaging. The rotation of the device that produces a positive output on the corresponding gyroscope is indicated by the curved arrows.

Figure 5-28. Motion Sensor Axis Orientation



However, according to the placement of the motion sensor device on the remote PCBA, the motion sensor axis orientation is different than that shown in Figure 5-28. The chip is flipped about the Y axis and rotated around the Z axis, while the remote is held in a pointing position. The axes of the sensor mounted on the PCB is changed as shown in Figure 5-29.

Figure 5-29. Placement of Motion Sensor on Remote PCBA

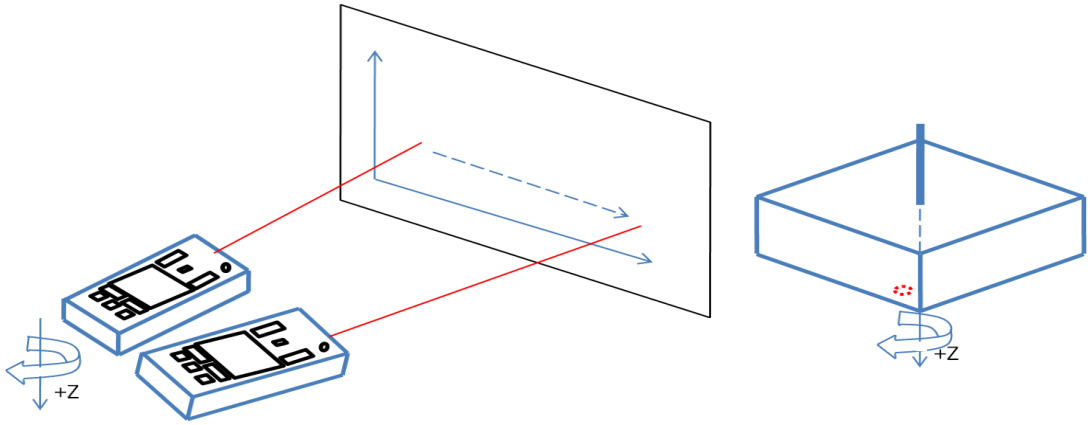


The AML is designed assuming the following referential conventions:

- A clockwise rotation around the gyroscope Y-axis generates positive values, that is, negative delta Y onscreen (cursor moves to the top).
- A clockwise rotation around the gyroscope Z-axis generates positive values, that is, positive delta X onscreen (cursor moves to the right).
- When the device lies flat on its back, the accelerometer must see +1G on the Z-axis.
- When the device lies on its right side, the accelerometer must see +1G on the Y-axis.
- When the device is held vertically pointing down, the accelerometer must see +1G on the X-axis.

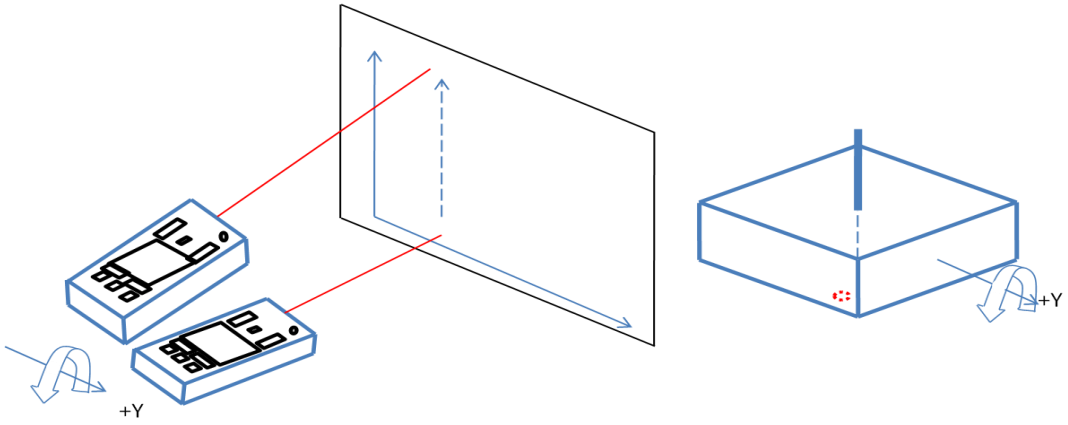
Considering these effective orientations of the motion sensor axes, it is concluded that the motion sensor mounted on the Remote Control RDK produces positive output on the Z-axis while the cursor is moved to the right, as shown in Figure 5-30, thus meeting AML convention 1.

Figure 5-30. Cursor Movement in X Direction



The motion sensor mounted on the Remote Control RDK produces a positive output on the Y axis while the cursor is moved to the top, as shown in Figure 5-31, thus meeting AML convention 2.

Figure 5-31. Cursor Movement in Y Direction



Conventions 3, 4, and 5 will not be met because the directions of the accelerometer axes are reversed due to flipping the chip for mounting. Hence, to meet conventions 3, 4, and 5, the reversed direction of the accelerometer axes will be compensated for by negating the output of the accelerometer.

When using other motion sensors and motion processing libraries, there may be a need to process the data read from the motion sensor to meet the conventions of the data processing library. This processing is part of the hardware abstraction layer of the firmware, so in the Remote Control RDK firmware, it is implemented in `Motion_Sensor_Hal_GetAccData()` and `Motion_Sensor_Hal_GetGyroData()` in the *motion_sensor_hal.c*.

5.1.5.4 Using Air Motion Library at Application Level

The AML exposes the following two APIs.

- `AIR_MOTION_Init()`: This function is used to initialize the processing logic with various configuration parameters. These parameters are passed to the function through a structure. The elements of this structure and their significance are as follows:
  - `DeltaGain`: Delta gain values for X and Y axes. More gain results in more pointer displacement for the same angular change.
  - `GyroOffsets`: Initial gyroscope offset values. Specifying the correct offsets eliminates the time consumed during calibration.
  - `GyroStaticMaxNoise`: Gyroscope maximum value per axis for the device to be considered currently static.

- StaticSamples: Number of consecutive “static” samples for the device to be considered fully static.
- SwipeMinDist: Minimum distance, as deltas sum, for a swipe to be detected (‘0’ means no processing). Used only if gestures are enabled.
- SwipeMaxNoise: Maximum noise level, as deltas sum, for a swipe to be rejected. Used only if gestures are enabled.
- StartupSamples: Number of samples to discard before starting computation. It should be equal to the time taken by the gyroscopes to stabilize after powering up.
- ClickStillSamples: Maximum number of null pointing samples after a button press. This should be equal to the time for which a button press can move the mouse pointer.
- ClickStillTolerance: Stillness tolerance level, as maximum movement quantity, for pointing samples to be forced null after a click press.
- IsRollCompEnabled: Activate roll compensation feature.
- Acc1gLsb: Norm value read from the accelerometer when the device is static (Earth's gravitational acceleration). Specify this value according to the full-scale range of the accelerometer set by the motion sensor driver.
- GyroSensitivity: Gyroscope sensitivity as “16*LSB / Â°/s.” Specify this value according to the full-scale range of the gyroscope set by the motion sensor driver.

These parameters should be tuned properly to achieve an optimal performance of the motion mouse. Default values of the parameters used in the RDK firmware may need to be changed on the actual end-product depending on its structure and ergonomic characteristics.

- AIR_MOTION_ProcessDelta(): This function processes the motion sensor data. Motion sensor data is passed as parameters through a structure. The elements of the structure are as follows:
  - GyroSamples: Gyroscope samples
  - AccSamples: Accelerometer samples
  - ClickSample: Click buttons state sample. It notifies the library that a button is pressed. This information is used by the click tolerance logic to avoid wiggling of the mouse on button press.
  - This function returns the computed deltas for the mouse pointer and updated gyroscope offsets through a structure. Elements of this structure are the following:
    - Status: Status bits of the last sample processing. Holds the status of the device notifying and if the device is steady, if new gyroscope offsets are computed, and if the mouse delta is computed.
    - GyroOffsets: Value of gyroscope offsets
    - Delta: Computed delta values
    - SwipesDetected: Bit field of detected swipes. This element is used only if gesture recognition is used.

The AIR_MOTION_Init() function is called while initializing the library or performance parameters such as gyroscope offsets, delta gain, and so on. The AIR_MOTION_ProcessDelta() function is called as part of the polling routine for the motion sensor. The structure returned by this API is used to generate the mouse report and update the gyroscope offsets. The APIs exposed by the AML are independent of the motion sensor used and therefore can be used in the application layer of the motion sensor module for a motion sensor. The code corresponding to the application layer is organized in *motion_sensor.c.*

5.1.5.5 AML Features

Apart from the computation of mouse pointer movement, the AML also implements features that improve the user experience and performance of the air mouse.

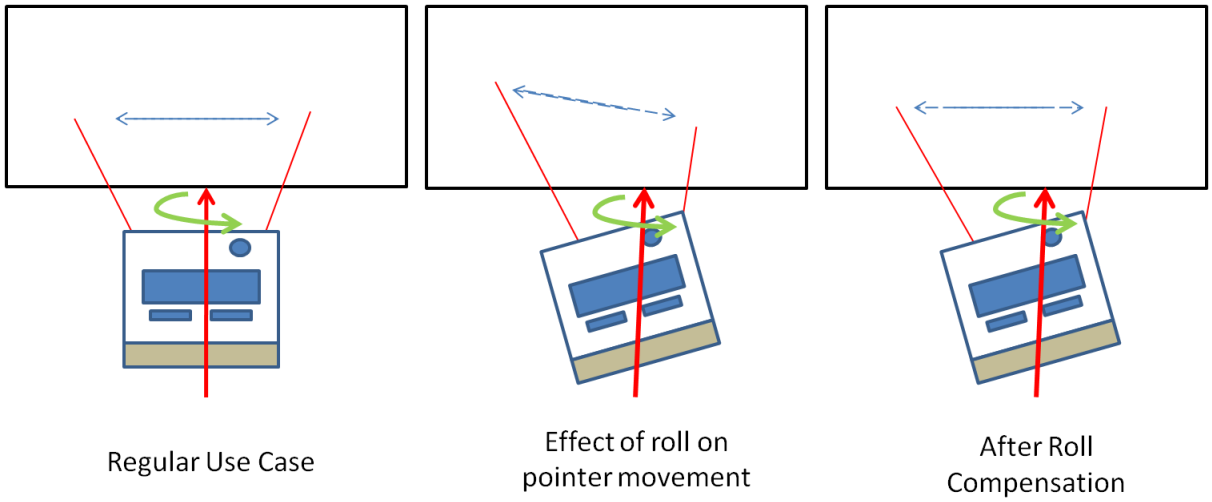
- Auto-calibration of gyroscopes: The function of the auto-calibration feature is to eliminate gyroscope offset and drifts. The calibration is automatically carried out when the motion sensor samples are passed to AIR_MOTION_ProcessDelta()as part of polling routine. At the application layer, the generation of mouse reports stops until calibration is in progress. When the calibration is complete, the function indicates this fact through the NewGyroOffset element of t_struct_AIR_MOTION_Status. It also keeps updating the offsets automatically without interfering with the mouse movement computation (once calibration is successful). First-time calibration takes

time corresponding to the initalization value of `StaticSamples`. Once the offsets of `gyro` are calculated, they can be saved in nonvolatile memory such as EEPROM or flash memory. These offsets can be used thereafter directly for initialization of the AML. Passing the gyroscope offsets saved in nonvolatile memory to the initialization structure of `AIR_MOTION_Init()` at the beginning of device initialization bypasses the calibration, which enables seamless mouse operation after powering up the device.

Note: For the first time after the RDK is programmed, gyroscope offsets are not computed, so the default offsets used to initialize the AML are passed as 0x0000. This triggers auto-calibration at the beginning of motion sensor polling. The calibration logic waits for the device to be steady for `StaticSamples` amount of time before computing offsets and then starts the calibration. Hence, the RDK should be kept still for `StaticSamples` amount of time for the completion of calibration; otherwise, calibration will not take place. If the calibration is not completed, the application will not generate mouse reports and the mouse pointer will be kept stalled.

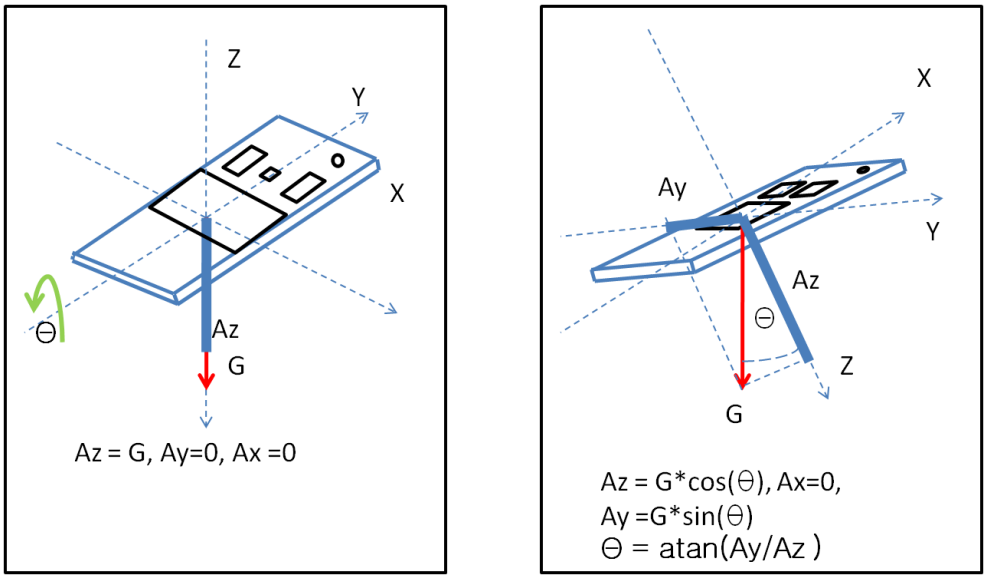
- Roll compensation: If the user holds the remote in a tilted manner, the gyroscope axes will also be tilted by the same angle. This angle is called the “roll” of the remote. The roll angle would tilt the x-y direction of pointer movement by a roll angle, resulting in horizontal movement of the remote to produce a tilted pointer movement, as shown in Figure 5-32.

Figure 5-32. Roll Compensation



The roll compensation features eliminates this issue. It rotates the x-y direction of the pointer movement by the same angle in which the remote is tilted, but in the opposite direction. The angle of the tilt is computed using accelerometer data. The accelerometer in a steady state is acted upon by acceleration due to gravity, which produces nonzero acceleration output on all the axes of the accelerometer, depending on the angles made by the motion sensor axes with the direction of gravity. Thus, the measurement from the accelerometer is used to calculate the orientation of the device and hence the roll angle, as depicted in Figure 5-33.

Figure 5-33. Estimation of Roll Using Accelerometer Output



The AML uses this roll angle to nullify the effect of roll in the mouse pointer movement, called “roll compensation.” However, the calculation of roll is performed only when the mouse pointer is steady. So, only the roll of the remote at the beginning of pointer movement is compensated for. The compensation logic does not take into account the tilt introduced in the discourse of movement.

The roll compensation feature is enabled by assigning the value of `IsRollCompEnabled` to 1 while initializing the AML using `AIR_MOTION_Init()`.

- Click tolerance: When a button is being pressed on the remote, the remote moves a bit and causes unintended pointer movement. This movement creates difficulties when the user is trying to click on a specific location. The click tolerance feature resolves this issue. In this feature, the button press is reported to `AIR_MOTION_ProcessDelta()` by assigning parameter `ClickSample` to 1 when a button press or release is detected. As soon as a button press is detected, the AML logic keep reporting zero delta for X and Y mouse pointer movement for a configurable time interval. The time interval is configured using `ClickStillSamples` parameter, which is passed to `AIR_MOTION_Init()`. Typically this time is 300 ms. However, the movement of mouse is frozen only if the movement is within the level of `ClickStillTolerance`. This tolerance level is also set in the firmware while initializing the AML library.

5.1.5.6 Power Management in Motion Sensing Mode

The most current-consuming process for the motion sensor module is sampling the motion sensor data when all the gyroscopes and accelerometers are active. Hence, polling is carried out in the Active state of the firmware state machine during the motion sensor polling routine. If the remote is steady, HID reports for the mouse are not generated, as delta X and delta Y are 0. If no mouse reports are generated for 3 seconds, it implies that the user is not making any activity, so the firmware moves into the Idle state.

In the Idle state, the firmware puts the motion sensor in the low-power motion detect mode. In this mode, the gyroscopes are turned off. Accelerometers are turned on and are sampled at a low frequency, typically eight samples per second. The internal motion detect logic present in MPU6500 is enabled. This logic detects the motion of the device and generates an interrupt on the INT pin of MPU6500, triggered when the accelerometer output data exceeds a programmable threshold level. This threshold is configured using the WOM_Threshold[7:0] register of MPU6500. In the RDK firmware, the threshold value is defined as MOTION_WAKEUP_THRESHOLD in *MPU6500.h*. This threshold value needs to be configured in the firmware by the developers per the performance requirement.

5.1.5.7 Tuning Parameters for Good Performance:

While deploying the motion mouse in the end product, the user experience for mouse handling needs to be optimized via the following parameters:

- **ClickStillTolerance**

  Description: This parameter determines the quantity of movement that should be tolerated during a button press.

  Value: This parameter can be assigned three values:
  - `AirMotionLow`
  - `AirMotionNorma`
  - `AirMotionHigh`.

  Considerations: if this parameter is set to high (`AirMotionHigh`), a greater movement in pointer is tolerated as a button press activity. However, a greater movement is required for pointer to start moving after the button press is complete. The effect of a high tolerance level would reflect in the click-and-drag action, which would show a jerk in pointer movement after the click action. Thus, an appropriate level needs to be decided that is suitable for the desired user experience.

  Firmware details: Macro `AIR_MOTION_DEFAULT_CLICKSTILLTOLERANCE` in *motion_sensor.h*

- **ClickStillSamples**

  Description: This parameter determines the time for which the pointer needs to be frozen after a button press is detected.

  Value: Can be any uint8 number. Typically, this value is 10 to 80 corresponding to a time duration of 100 to 800 ms.

  Considerations: The optimal value for this parameter depends mostly on the mechanical and ergonomic aspect of the end product design. If the buttons of the remote are very soft for presses, the movement produced due to a button press will be small and of a shorter duration. On the other hand, if the buttons on the remote are harder, the movement produced by a button press would be of a longer duration. Therefore, this parameter should be set to a value that is high enough to last until the movement of the button press stops and low enough to report an immediate user activity after a button press.

  Firmware details: Macro `AIR_MOTION_DEFAULT_CLICKSTILLSAMPLES` in *motion_sensor.h*

- **Motion Detect threshold**

  Description: This parameter specifies the value of the MPU6500 register used by the Motion Detect logic of the sensor.

  Value: Any uint8 number. Typically, the value can be from 1 to 50.

  Considerations: This parameter determines the amount of acceleration in milli-gravity (mg) that would be considered as movement of the device. The detection of motion would raise an interrupt to the firmware and move the firmware state machine from Idle or Low Power state to Active State. This threshold should be sufficiently low to detect a small amount of movement caused when the user picks up the remote from the desk. However, it should be high enough not to detect taps or vibrations on desk, if the threshold is low, and the remote wakes up due to vibration. This will not affect user experience, but it will cause the remote to spend more amount of time in the Active State and turn on the gyroscopes and process its data. This will increase average current consumption of the device, leading to reduced battery life.

  Firmware details: Macro `MOTION_WAKEUP_THRESHOLD` in *mpu6500.h*

- **DeltaGain**

  Description: This parameter specifies the gain values that determine the sensitivity of the pointer to the movement of the device.

  Value: Can be any int8 number. Typically, it ranges from 10 to 20.

  Considerations: More gain results in greater pointer displacement for the same angular change. The end user experience is the only factor that can be used to tune this parameter.

  Firmware details: Macro `AIR_MOTION_DEFAULT_GAIN_DELTA_X, AIR_MOTION_DEFAULT_GAIN_DELTA_Y` in motion_sensor.h

- **GyroStaticMaxNoise**

  Description: This parameter specifies the amount of peak variations in the gyroscope output when the device is steady.

  Value: Any uint8 number

  Consideration: The value determines how much change in gyroscope output should the AML consider as noise. The value should be determined by separately analyzing the gyroscope output samples collected when the device is steady and determining the peak noise in the gyroscope output for a period of at least 1 second. Setting this parameter to more than the actual noise in the gyroscope output will not produce any movement when device is moved slowly. On the other hand, if this value is set to lower than the noise in the gyroscope output, an erroneous pointer movement can be observed when the mouse is steady.

  Firmware details: Macro `AIR_MOTION_DEFAULT_GYROSTATIC_NOISE` in *motion_sensor.h*

- **StaticSamples**:

  Description: This parameter specifies the number of consecutive samples from the sensor for AML to conclude that the device is at rest.

  Value: It can be any uint16 number. It can vary from 5 to 4000 depending on the use case.

  Considerations: The value of static samples is used to update gyroscope offsets of the AML. Calling the `AIR_MOTION_ProcessDelta()` function for SaticSamples number of times when the device is static results in an update in gyroscope offsets. The updated offsets are returned by this function in the form of the `GyroOffsets` element of the returned structure. Also `Status.NewGyroOffset` is set to TRUE in the same returned structure. Thus, a small value of the static sample will result in fast updates in the gyroscope offsets. If the firmware is writing the offsets into a nonvolatile memory when new offsets are available, the memory writes will be frequent. This may result in reduced age for the memory. On the other hand, if the value of the StaticSamples id very large, the gyroscope offsets will not be calculated for long durations. If the offsets are not updated frequently, the drifts in the gyroscope due to temperature changes and other possible reasons will not be compensated, which may result in unnecessary pointer movements.

  Firmware details: Macro `AIR_MOTION_DEFAULT_STATIC_SAMPLES` in *motion_sensor.h*

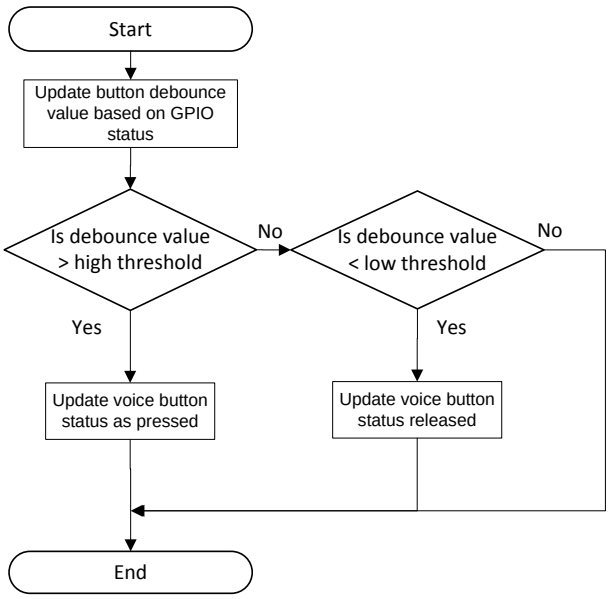
5.1.6 Button Subsystem

The button subsystem checks the status of the following buttons:

- Voice button: Activates the voice mode
- Motion button: Activates the motion sensor and subsystem
- Return button: Sends the return keyboard code of 0x9E over BLE

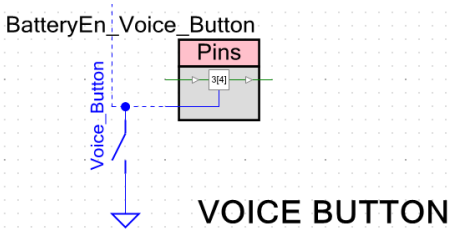
Figure 5-34 shows how the voice button is scanned. The voice button is connected to a dedicated GPIO, which is also used in battery monitoring. For more information on the hardware, see the Battery Monitoring section.

Figure 5-34. Voice Button Flow Chart



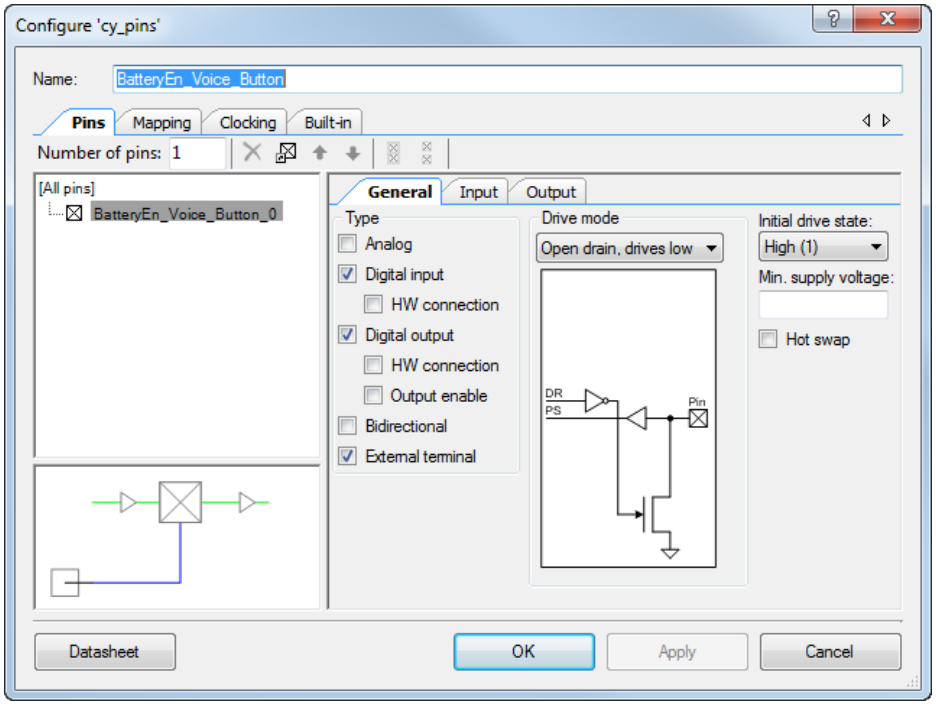
The button subsystem uses the cy_pins Component shown in Figure 5-35.

Figure 5-35. cy_pins Component for Button Module



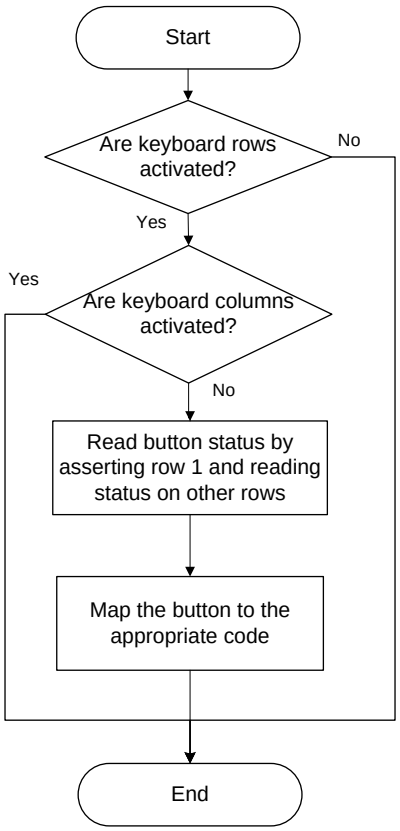
The cy_pins Component for the voice button is configured as a **Digital output with Open drain,** and **Digital input** as shown in Figure 5-36. The digital output function is used to drive the pin LOW during battery voltage measurement.

Figure 5-36. Pins Tab Configuration Settings



The motion and return button are multiplexed between the rows of the keyboard matrix. Figure 5-37 shows the flow chart for scanning these buttons.

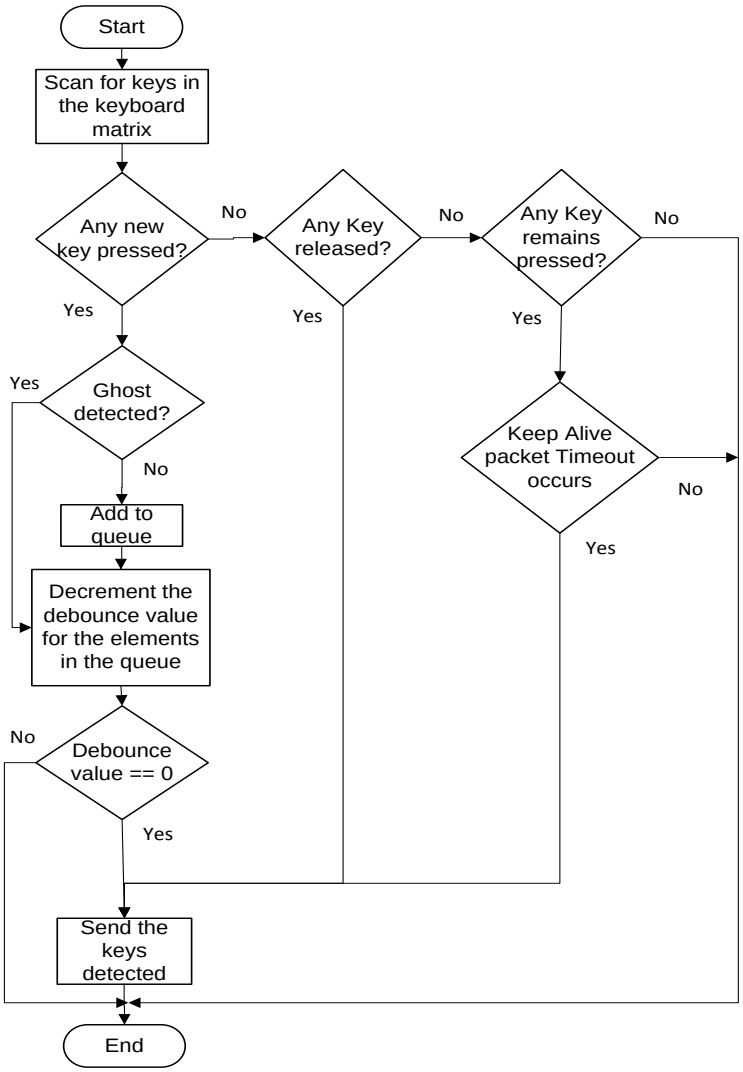
Figure 5-37. Keyboard Matrix Button Flow Chart



5.1.7 Keyboard Subsystem

The keyboard subsystem is configured for three-row and three-column button matrix operation. The keyboard subsystem scans and debounces any activated keys in the keyboard matrix. The activated keys can be one of the following types: standard, multimedia, or power keys. The information from this subsystem is passed to the application framework. Figure 5-38 shows the flow chart for the keyboard subsystem operation.

Figure 5-38. Keyboard Flow Chart



To scan the keys, all the rows are pulled down to logic 0 first. Then, each column is pulled to logic 1 through a resistive pull-up and its status is read after a 20-µs delay. If any button on a given column is pressed, the firmware will read a logic 0 on that column. In this case, the firmware identifies the pressed button by pulling a row to logic 0 while keeping all other rows at logic 1 and reading the column status. If the pressed button is located on a given row, the firmware will read logic 0.

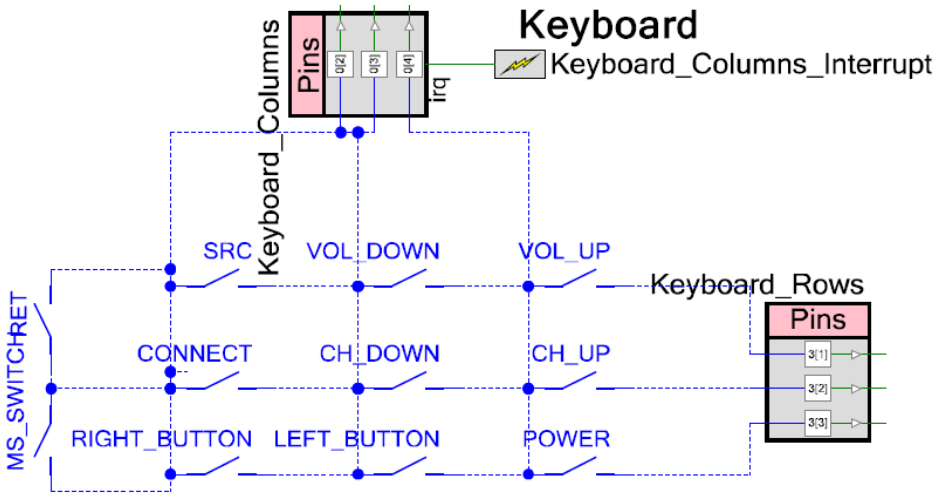
If a new key press is detected, the subsystem checks if it is a ghost key. A ghost key condition is detected when three physical keys are pressed but four keys are detected. If the identified key is not a ghost key, add the detected key to the queue for denouncing. After denouncing is complete, the detected keys are passed to the application framework. Refer to Table 5-5 for a complete list of key codes implemented in the kit firmware.

Table 5-5. Keyboard Matrix Key Mappings

Key	Key Codes
Volume Up	0x00E9 (0th word of the of the multimedia report)
Volume Down	0x00EA (0th word of the of the multimedia report)
Channel Up	0x009C (0th word of the of the multimedia report)
Channel Down	0x009D (0th word of the of the multimedia report)
Source	0x0087 (0th word of the of the multimedia report)
On/Off	0x0030 (0th word of the of the multimedia report)
Left button	Left-click (0th bit of the 0th byte of the mouse report)
Right button	Right-click (1st bit of the 0th byte of the mouse report)
Connect	Nothing is sent over BLE

The keyboard subsystem uses the cy_pins Component (Figure 5-39) to interface with the key matrix. Separate sets of pins are used for rows and columns.

Figure 5-39. cy_pins Component for Keyboard Module



The row and column cy_pins Components are configured as follows. Since the keyboard is an input device, all column and row pins are configured as input, as shown in Figure 5-40 and Figure 5-41.

Figure 5-40. cy_pins Component Settings for Columns

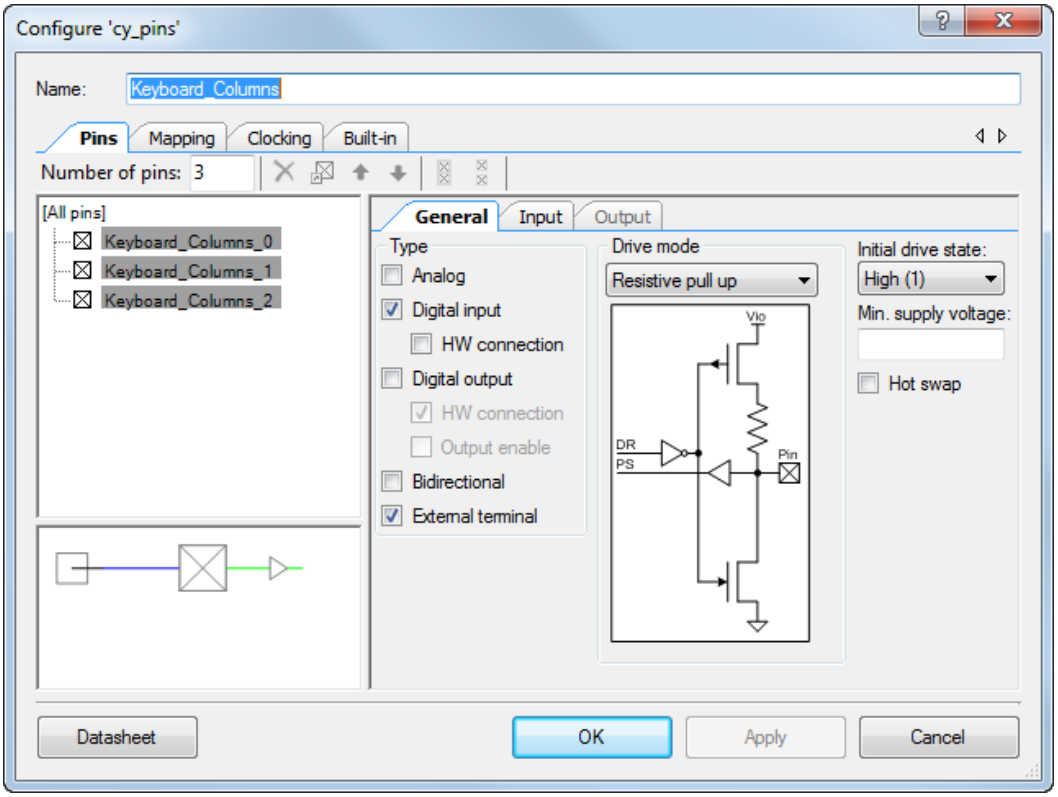
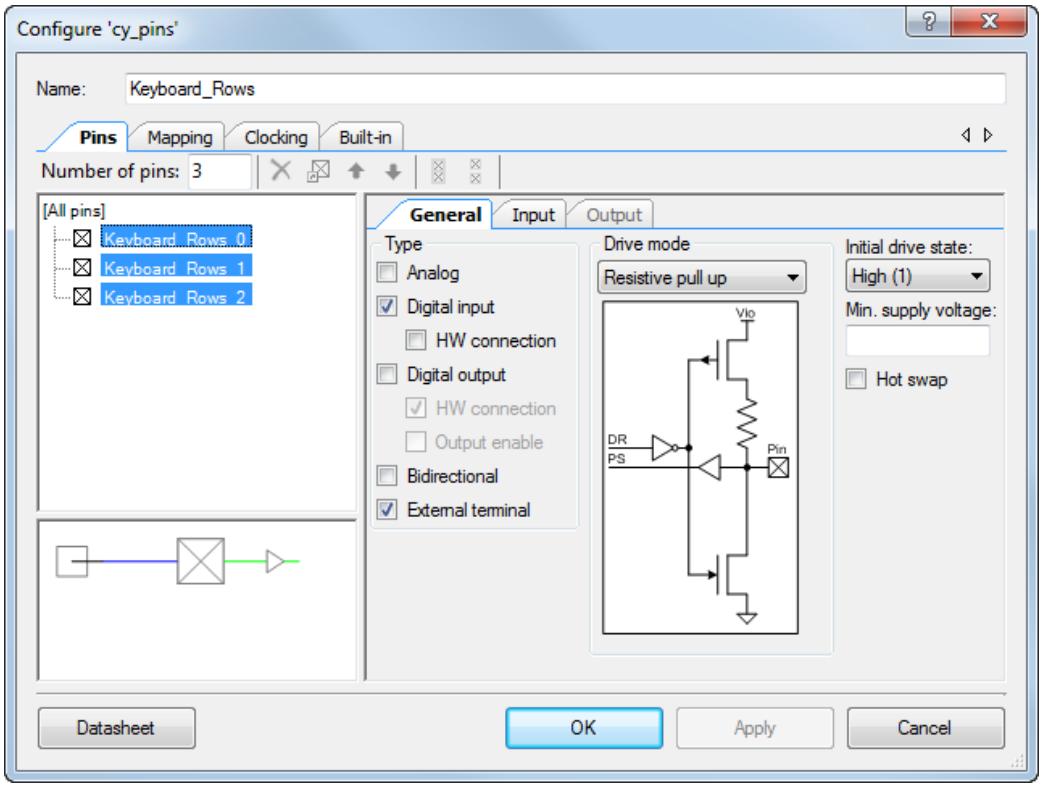


Figure 5-41. cy_pins Component Settings for Rows



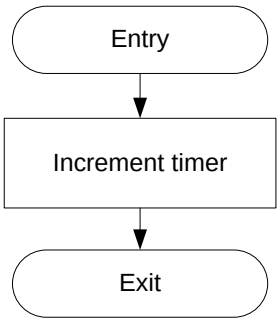
5.1.8 Timer Subsystem

The timer subsystem is implemented using the watchdog timer. The PRoC BLE watchdog timer has two 16-bit counters (counter 0 and counter 1) and one 32-bit counter (counter 2). These counters can be configured to work independently or in cascade.

The watchdog timer uses the 32767-Hz clock for its operations. It is the only timer that is available when the MCU is in Deep Sleep. The watchdog timer is configured to generate an interrupt every 1 ms in the Active state, every 125 ms in the Idle state, and every 250 ms in the Sleep state.

The remote firmware utilizes counter 1 for generating interrupts at a regular interval. The application framework uses this timer value to switch between states. Figure 5-42 shows the flow chart for the timer subsystem.

Figure 5-42. Timer Module Flow Chart



5.1.9 Battery Monitoring Subsystem

The battery monitoring subsystem measures the voltage at the battery terminal. This subsystem is polled every 3 seconds. The Sequencing SAR ADC Component provided in PSoC Creator is used to measure the battery voltage. Figure 5-43 shows the flow chart for this subsystem.

Figure 5-43. Battery Monitoring Module Flow Chart

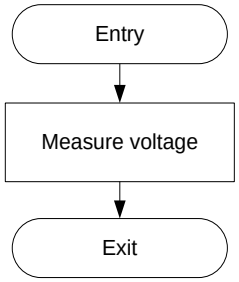
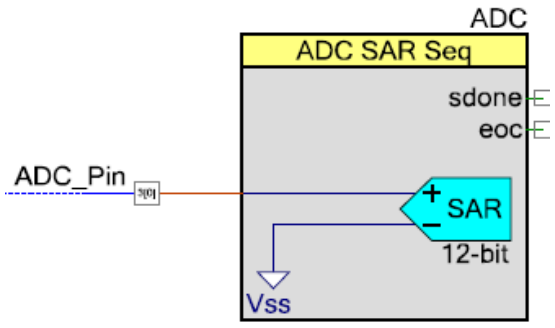


Figure 5-44 shows the schematic of the Sequencing SAR ADC Component. The following sections describe the configuration settings.

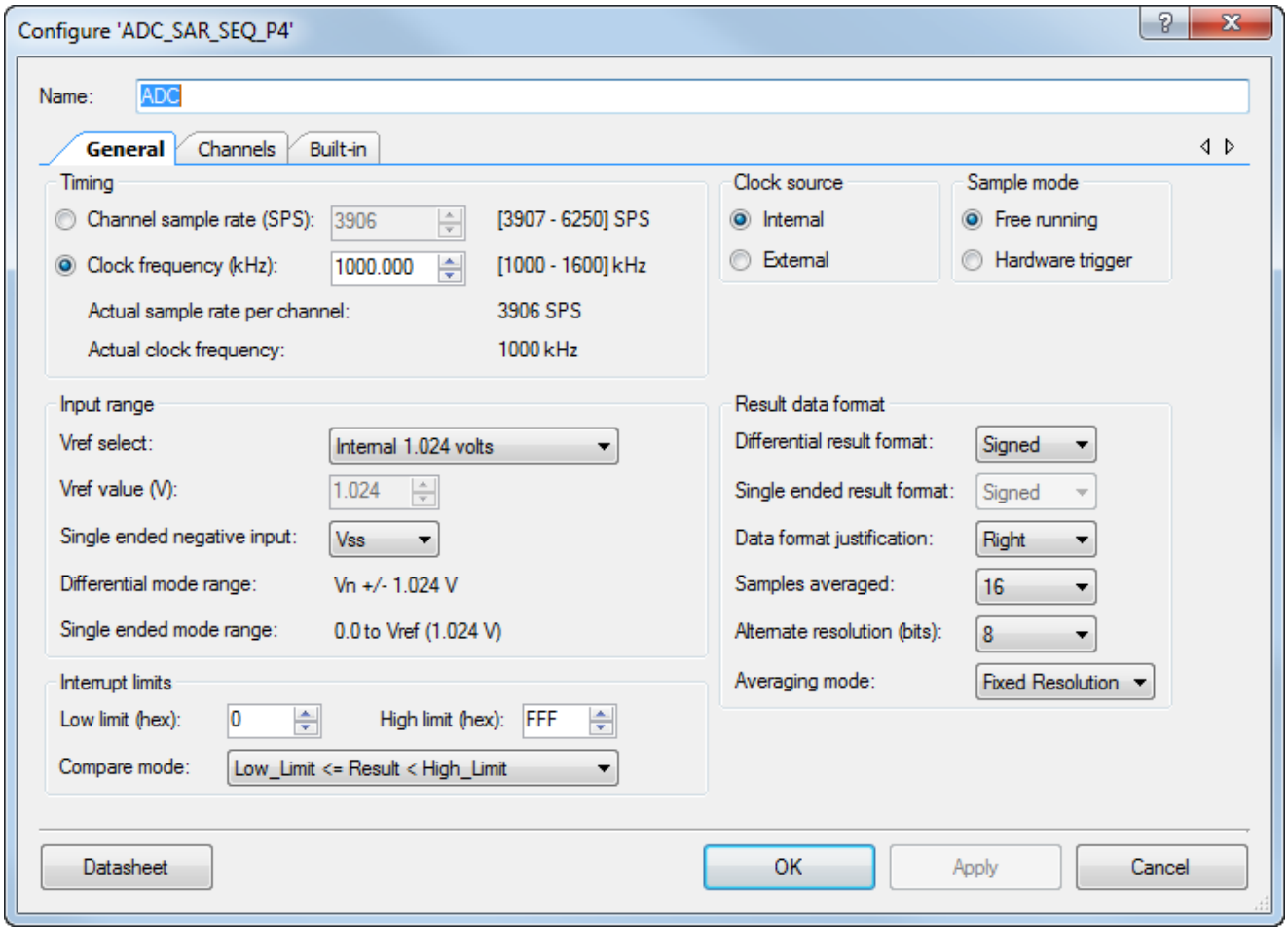
Figure 5-44. Sequencing SAR ADC Component in PSoC Creator



5.1.9.1 General Tab

The **Clock frequency** and other parameters shown in Figure 5-45 are selected to achieve a sampling rate of 3,906 samples per second.

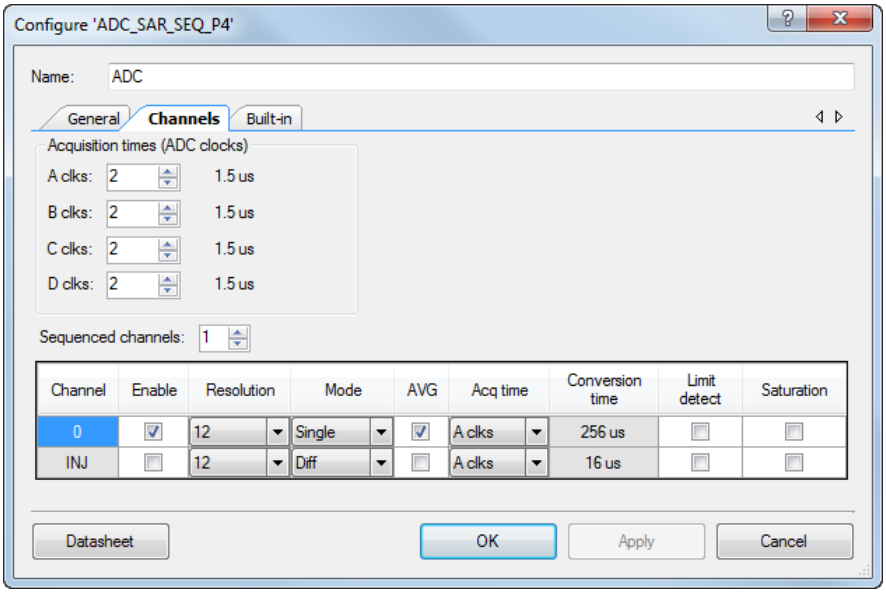
Figure 5-45. General Tab of Sequencing SAR ADC Component



5.1.9.2 Channels Tab

The ADC operates with a single 12-bit resolution channel input. The clock is set to “A clks” to get a conversion time of 256 µs, as shown in Figure 5-46.

Figure 5-46. Channels Tab of Sequencing SAR ADC Component

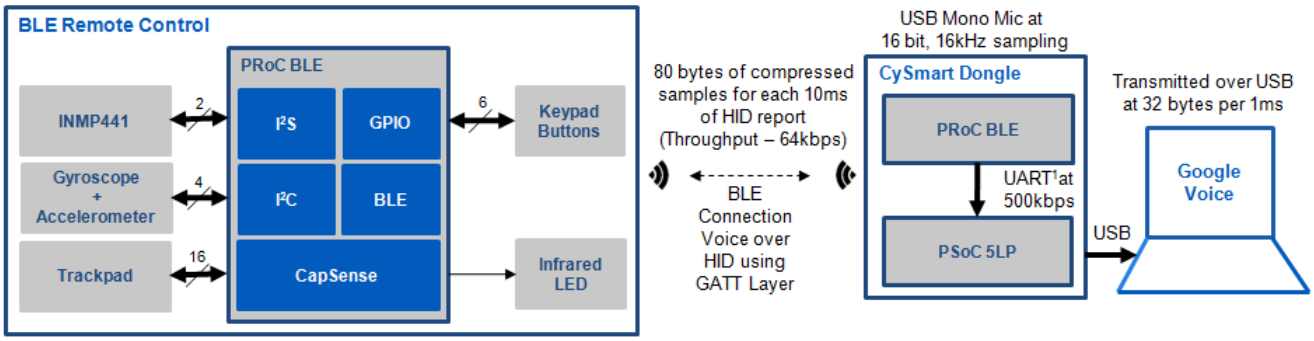


The datasheet provides a list of APIs supported by the Sequencing SAR ADC Component. To go to the datasheet, click the **Datasheet** button at the bottom left corner of the Sequencing SAR ADC Component GUI shown in Figure 5-46.

5.1.10 Audio Subsystem

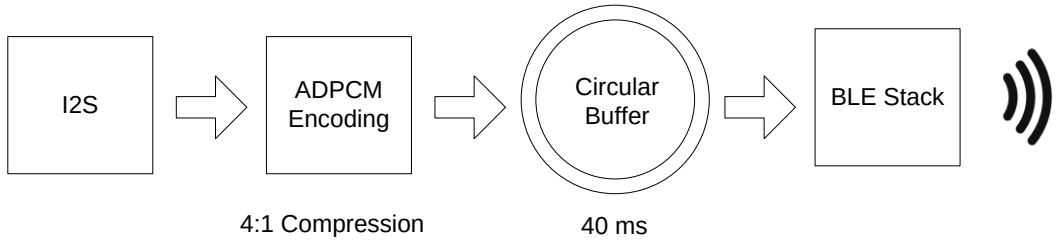
The CY5672 remote control supports audio capability to enable voice recognition function on the host device (PC or Smart TV). The audio data (voice commands) are transferred over air to the host with voice recognition capability, which decodes the voice commands and triggers the corresponding action. Figure 5-47 provides the system level view of voice command transfer from the remote control.

Figure 5-47. Voice Recognition Using CY5672 Remote Control



The firmware data flow for the audio data on the remote control is illustrated in Figure 5-48.

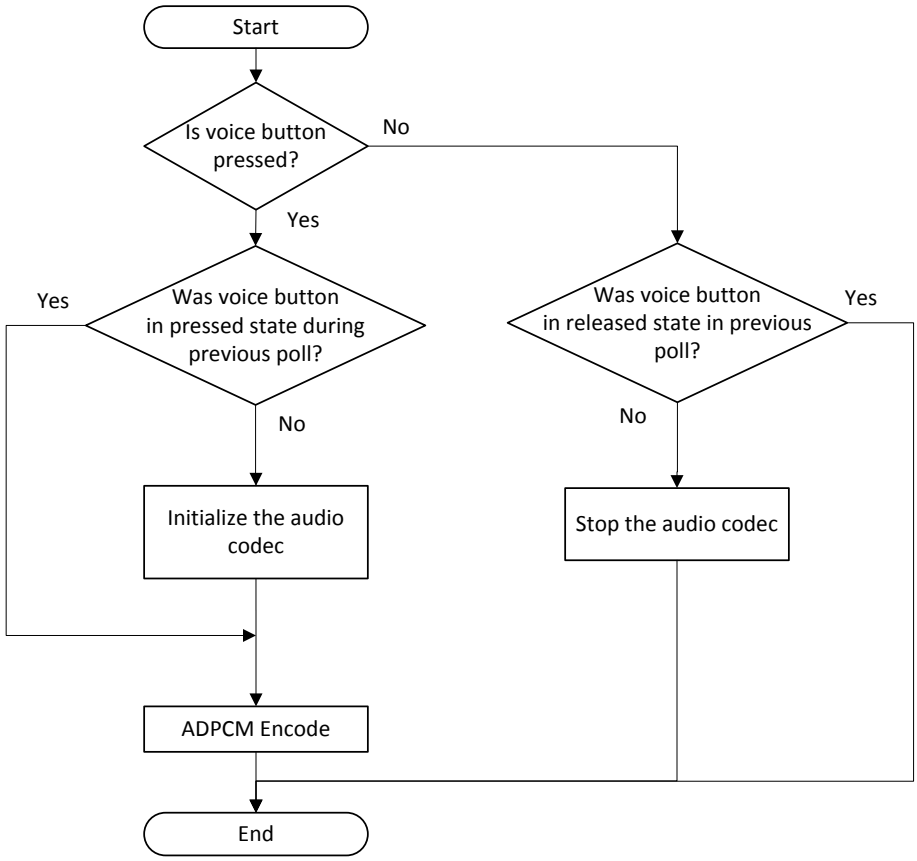
Figure 5-48. Firmware Data Flow for Audio Feature



In the host side, the audio data is received as a HID report, depending on the host side implementation it can use a standard driver to collect the HID data and send to voice recognition software like Google voice or Nuance for decoding the voice command. When a CySmart dongle is used in the host system, the audio data received over BLE is sent to the dongle, which enumerates as an USB audio device to the host device.

The audio subsystem collects audio data from the INMP441 codec (16 bit, 16 kHz) over the I2S interface. The subsystem uses the I2S Component. The I2S interrupts when it receives audio data in the PCM format. The received data is read from the I2S Component and stored in a local buffer. The subsystem compresses the audio data in the buffer using the ADPCM algorithm (compressing 16-bit audio data to 4-bit). Figure 5-49 shows the flow chart for the audio subsystem.

Figure 5-49. Audio Subsystem Flow Chart



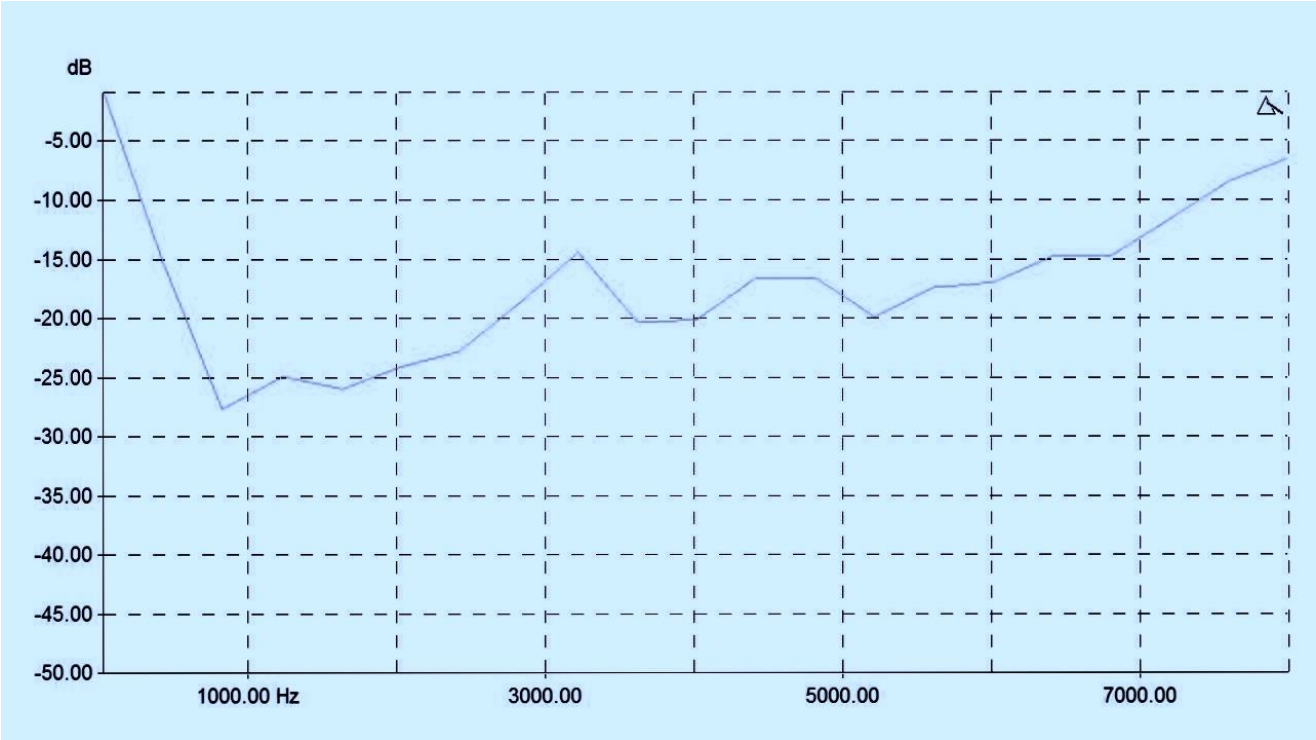
5.1.10.1 Quality of Voice Required for Voice Recognition

A typical voice recognition software requires a threshold signal-to-noise ratio (SNR) for the recognition of voice syllables correctly By using ADPCM as compression, the raw SNR is reduced as the 16-bit data is effectively reduced to 4 bits. The effective SNR after compression hovers around 20 dB.

Most of the prevalent voice recognition software like Google voice and Nuance operates well around 15 dB -20 dB SNR. Figure 5-50 shows the SNR value for a frequency range of 100 Hz to 8000 Hz for the voice data processed using ADPCM

coding. The value of SNR is observed to be more than 15 dB. Typically, voices stay in the range of 200 Hz to 6000 Hz. Thus, the ADPCM coding is still sufficient to enable voice recognition.

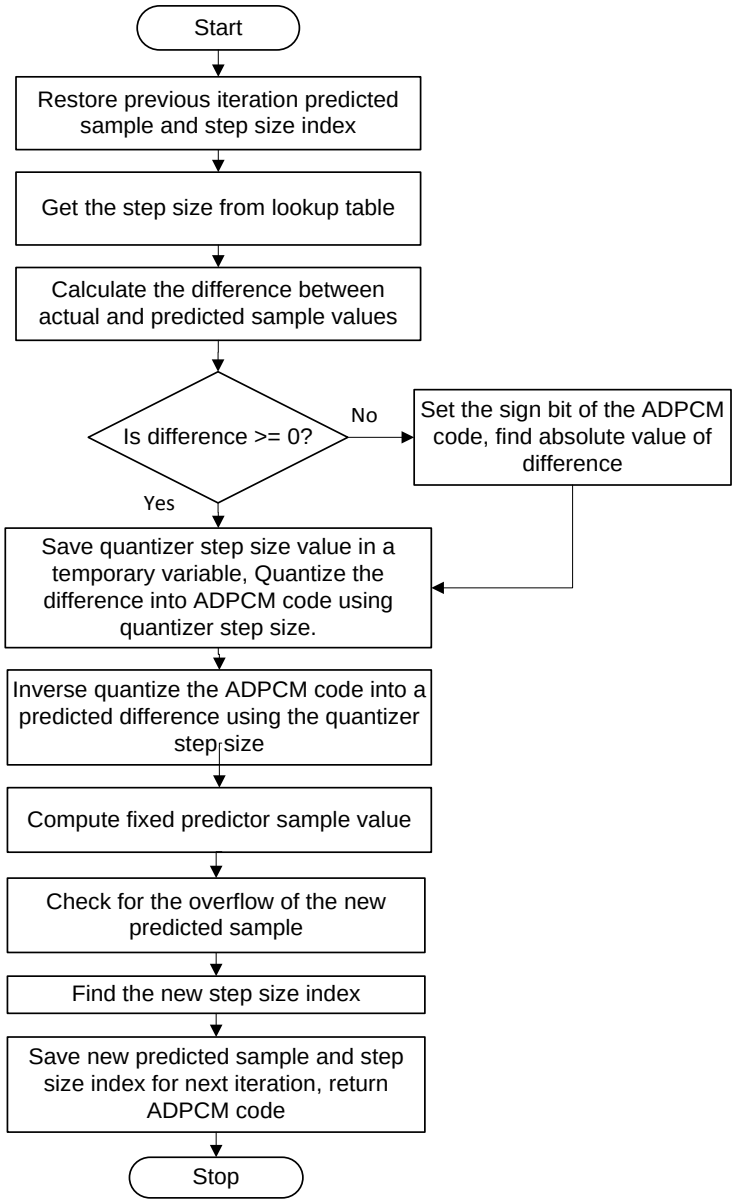
Figure 5-50. SNR for Voice Data Processed Using ADPCM Coding



5.1.10.2 ADPCM Encoding

The ADPCM algorithm takes advantage of the high correlation between consecutive speech samples, which enables future sample values to be predicted. Instead of encoding the speech sample, ADPCM encodes the difference between a predicted sample and the speech sample. This method provides more efficient compression with a reduction in the number of bits per sample, yet preserves the overall quality of the speech signal. Figure 5-51 is the flow chart for the ADPCM encoding algorithm.

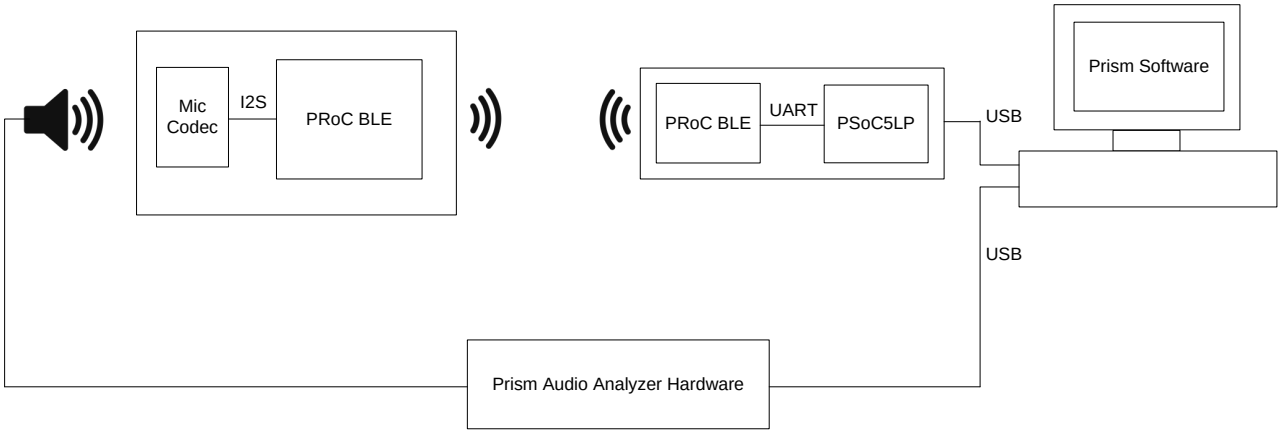
Figure 5-51. ADPCM Algorithm Flow Chart



5.1.10.3 Voice Performance of the Firmware

This section explains the key scientific audio quality parameters measured on the remote control. Note that the audio quality measurements were done by placing the speaker and remote in an anechoic box. This document does report the usual measurement in a normal environment instead of a professional acoustic measurement. Depending on the microphone and the speaker used with a particular enclosure, the results can vary. The measurement is more illustrative of the end-to-end system than as a reference benchmark.

Figure 5-52. Audio Quality Measurement Setup



The three important considerations for voice subsystem are as follows:

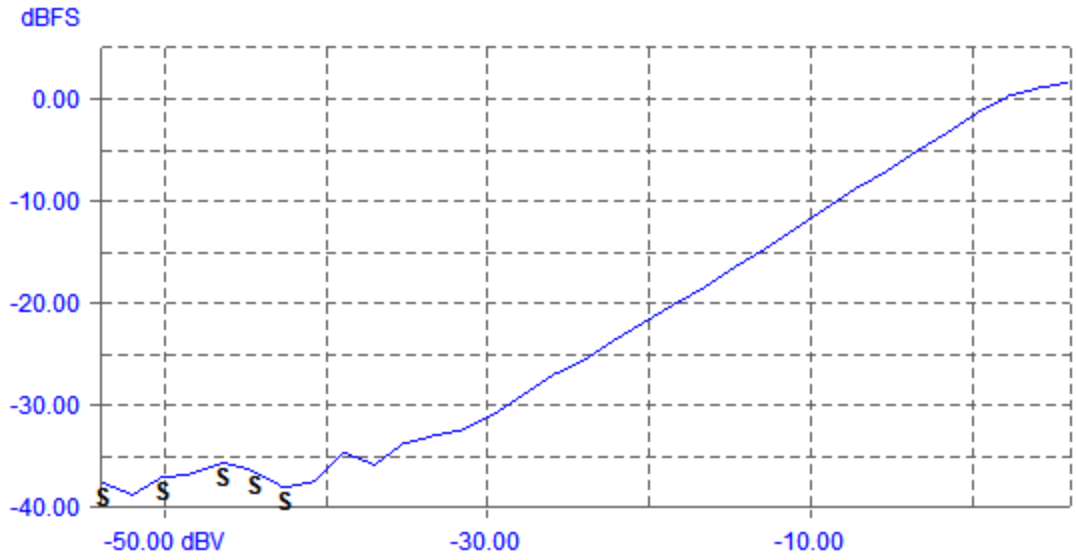
1. Input Output characteristics
2. Frequency response
3. SNR

The measurement is done through Prism Audio Analyzer. The Prism audio analyzer drives the speaker that is kept at a distance of 4 inches from the microphone of the remote. The USB audio output is observed through Prism Audio analyzer.

Input –Output Test

In this test, the Prism Audio Analyzer hardware feeds the data to the remote control's microphone using a speaker. The data received on the CySmart dongle is received over USB by the Prism software running on the PC. Figure 5-53 shows the audio data received on the PC in dB full-scale. The output is expected to track linearly.

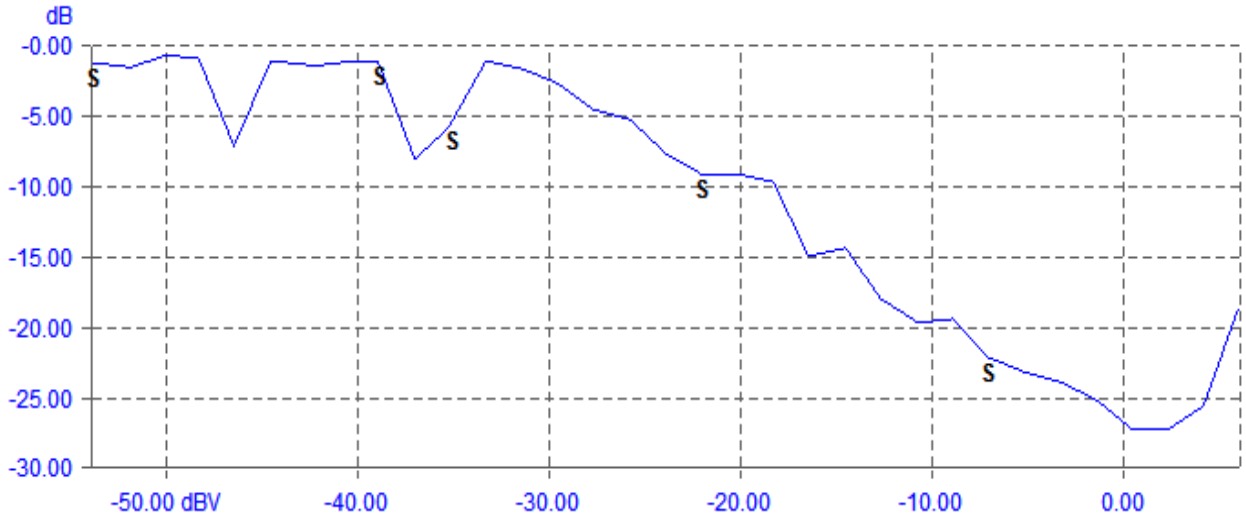
Figure 5-53. Audio Input-Output Test



Signal-to-Noise Ratio

The signal-to-noise ratio test specifies how much noise rides over the signal. This is a specification for many voice recognition systems. For this test the Speaker input was varied from 2 V to 20 µV from Prism Generator. The sound generated is in the normal voice level (96-20 dbSPL). The output total harmonic distortion and noise are read from the Prism software. This is the negative of SNR.

Figure 5-54. SNR Plot



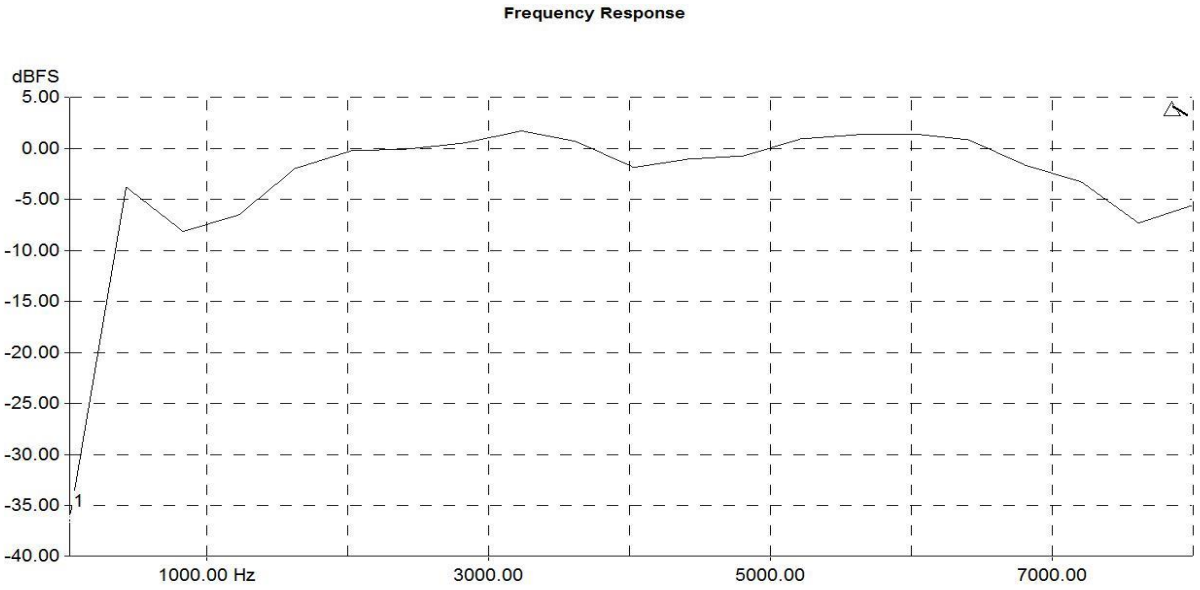
As seen from the graph, the SNR maximizes around 27 dB. This is expected from an ADPCM type encoding and decoding scheme. The linearity range observed t be approximately 33 dB.

The discontinuity in low-range, and occasionally in the mid-range points to the step adjustment of the ADPCM algorithm with RF disturbances. This is not a repeatable measurement and depending on the sweep of the input amplitude and the ADPCM seed, the result will vary. This only shows the maximum THD-N that we can get in a ADPCM system. To get a better idea, the experiment should be performed without a sweep of amplitude from the audio generator and the THD-N should be observed for a single amplitude over a period of time and the plot should be reconstructed.

Frequency Sensitivity of Voice Input

The aim of this experiment is to observe if there is any frequency selectivity with ADPCM coding and I2S codec. It is observed that in frequency range from 8000 Hz to 200 Hz, the maximum deviation is -8 dBFS to 3 dBFS.

Figure 5-55. Signal-to-Noise Ratio Plot



The frequency response is dominated by the frequency response of the speaker. The electrical frequency response with I2S input at the BLE chip and digital output at the USB audio device will be better than this as the speaker non linearity will be out of the system.

FFT Response

The FFT response shows any artifacts that arise from the coding or audio codec configuration. For the test, a 1-V, 1-K tone from Prism Audio Analyzer is applied to the speaker and the output spectrum is observed on the data received from USB. FFT is performed on this digital data. The spectrum does not show any spurious tone.

Summary

The following table describes the critical scientific audio quality parameters, their pass criteria values to ensure optimal voice recognition performance, and the measured values on CY5672 remote control.

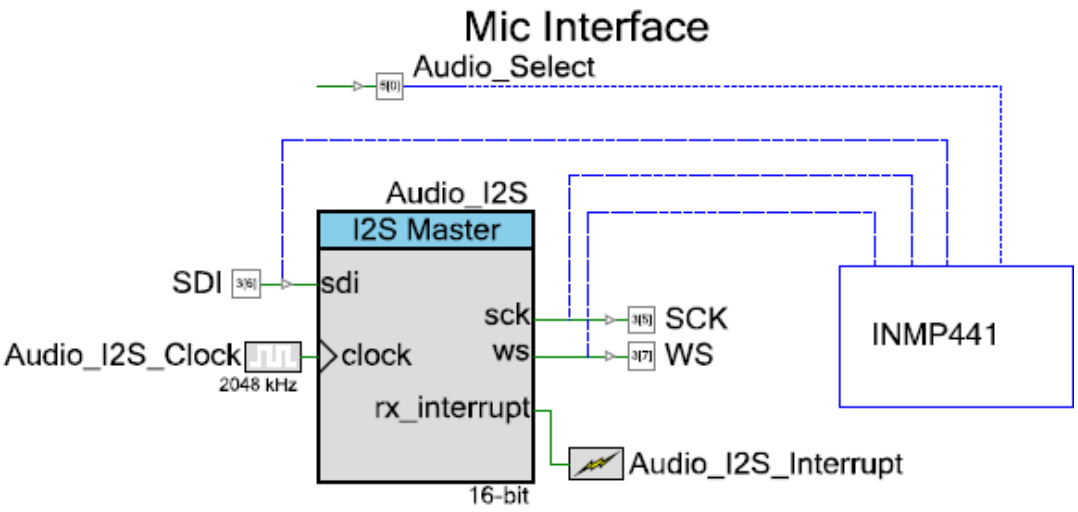
Table 5-6. Audio Quality Parameters: Pass Criteria and CY5672 Values

Parameter	Pass Criteria for Typical Voice Recognition Implementation	Measured Results on CY5672 Remote Control
Linearity	20 dB	30 dB
SNR	>15 dB	15-20 dB
Frequency response	100-4 K	100-7 K

As specified in the above table, the CY5672 remote control implementation meets the pass criteria for the scientific audio quality parameters required to ensure optimal voice recognition performance.

The following sections explain the I2S Component (Figure 5-56) configuration settings for remote control.

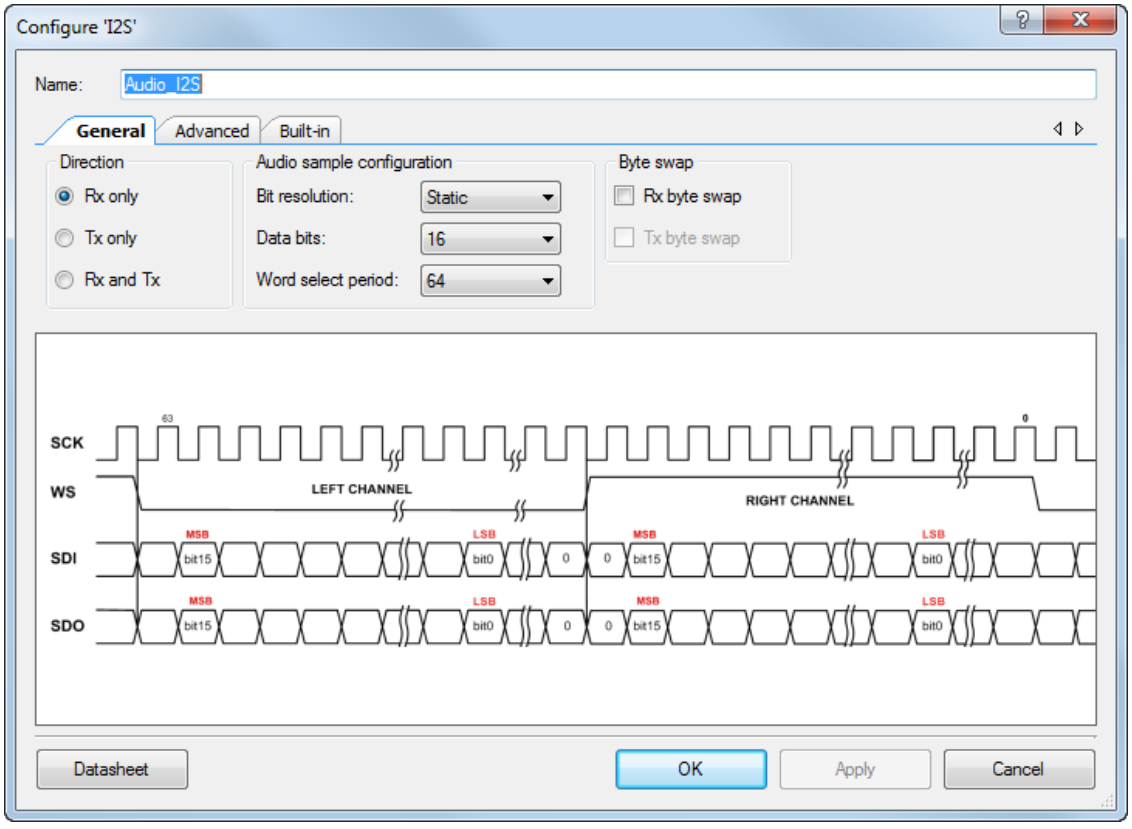
Figure 5-56. I2S Component for Audio Subsystem



5.1.10.4 General Tab

The **RX only** direction is selected (Figure 5-57), since the remote control contains only a microphone.

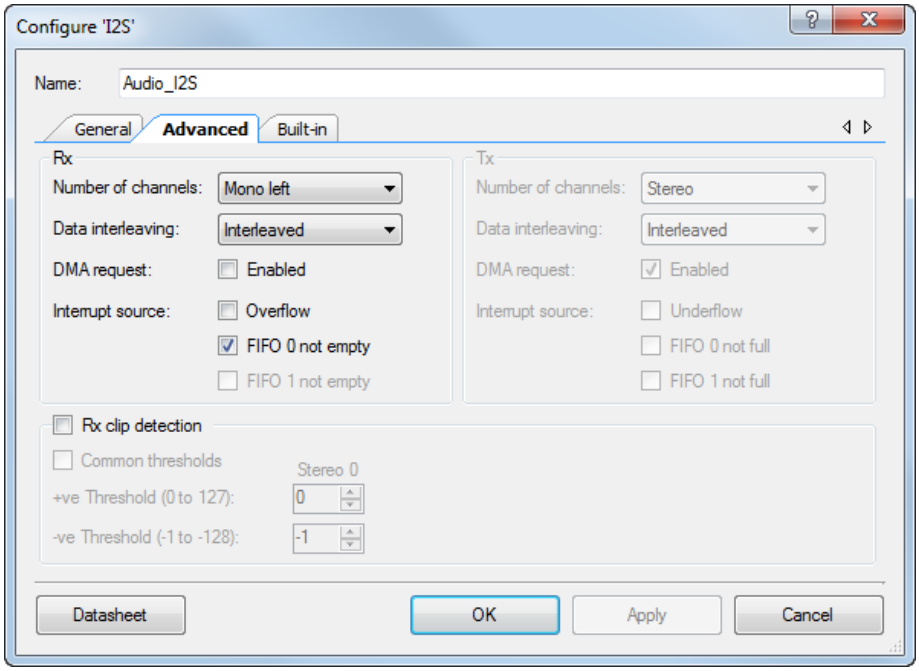
Figure 5-57. General Tab of I2S Component



5.1.10.5 Advanced Tab

The **Number of channels** is set as “Mono left” and the **Interrupt source** as “FIFO 0 not empty,” as shown in Figure 5-58.

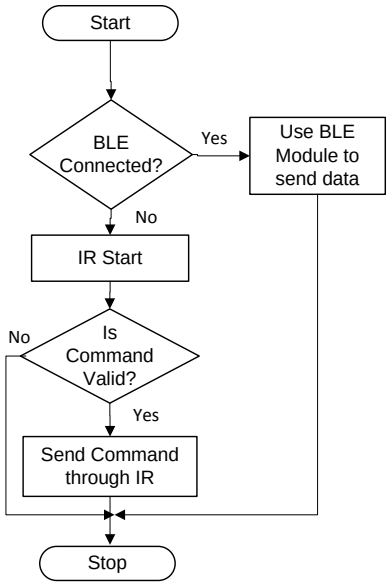
Figure 5-58. Advanced Tab of I2S Component



5.1.11 IR LED Subsystem

When the remote control cannot communicate with the host over a Bluetooth LE link (for example, in the case of a legacy TV that is not Bluetooth Smart Ready and does not have a USB port), the IR subsystem is used to send commands. Regular TV control commands like volume controls, channel controls, source, power, and return command are sent via this IR subsystem. The IR subsystem sends key codes over the IR LED interface using the Samsung or NEC IR protocols. A compile time switch (NEC_PROTOCOL) selects between the Samsung and NEC IR protocol implementation. By default, the Samsung TV IR protocol is enabled in the firmware supplied with the kit. Figure 5-59 shows the flow chart for the IR subsystem.

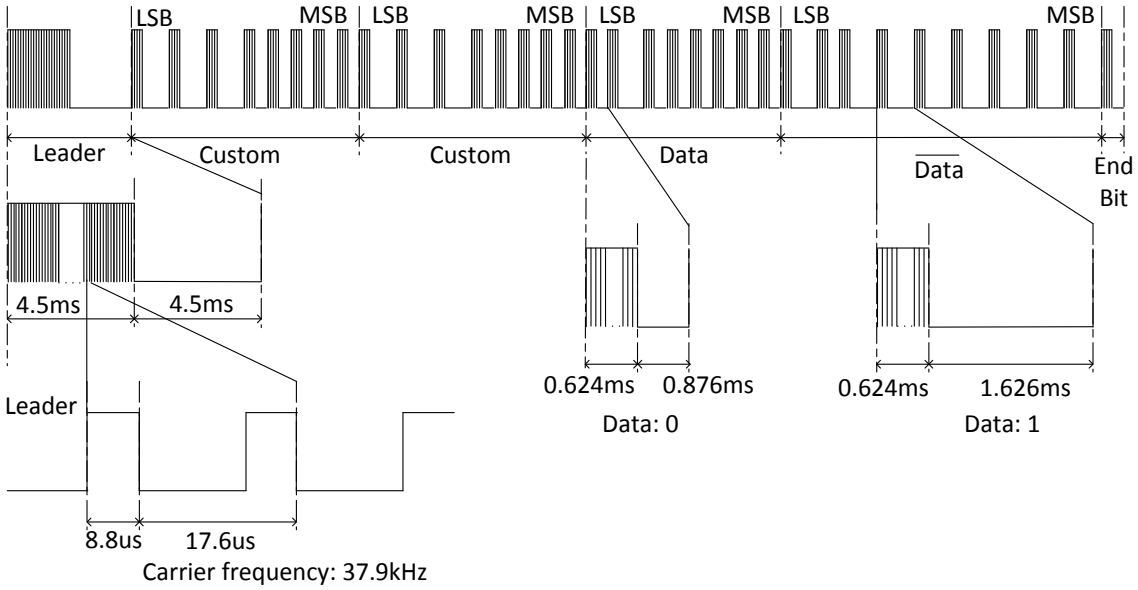
Figure 5-59. IR Subsystem Flow Chart



5.1.11.1 Samsung Protocol

The Samsung protocol uses pulse-distance encoding of the bits, containing a 16-bit address and 8-bit command, followed by an 8-bit command inverse. Each pulse is a 624-µs long, 38-kHz carrier burst (about 21 cycles). A logical '1' takes 2.25 ms to transmit, while a logical '0' takes only half that time: 1.125 ms. The recommended carrier duty cycle is 1/4 or 1/3. Figure 5-60 shows the Samsung protocol pulse train.

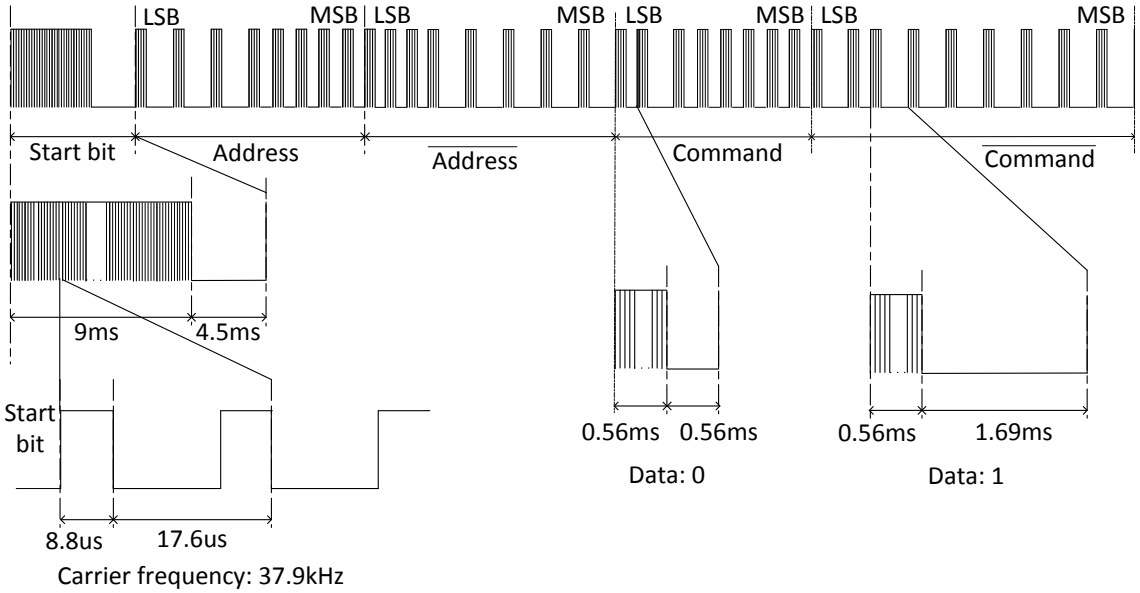
Figure 5-60. Samsung Protocol Pulse Train



5.1.11.2 NEC Protocol

The NEC protocol uses pulse distance encoding of the bits, which contains an 8-bit address followed by an 8-bit address inverse, and an 8-bit command followed by an 8-bit command inverse. Each pulse is a 562.5-µs long, 38-kHz carrier burst (about 21 cycles). A logical '1' takes 2.25 ms to transmit, while a logical '0' takes only half that time: 1.125 ms. The recommended carrier duty cycle is 1/4 or 1/3. Figure 5-61 shows the NEC protocol pulse train.

Figure 5-61. NEC Protocol Pulse Train



5.1.11.3 Implementation of the IR Protocol Using PWMs

Both the Samsung and NEC IR protocols require a pulse train at 38 kHz. To generate the required binary pattern (which includes the leader/start, custom/address, and data/command in Samsung and NEC protocols respectively), the 38-kHz pulse train must be turned ON and OFF for certain durations. Refer to Figure 5-60 and Figure 5-61 for the timing details.

Two PWMs are used in the firmware to generate the required binary pattern:

- Pwm_IR: This PWM is used to generate the 38-kHz pulse train.
- Pwm_IR_Timer: This PWM is used to maintain the timing for transmitting a bit of information over IR. For example, to send one command bit of value ‘0’ using the Samsung IR protocol, the 38-KHz pulse train should be turned ON for 0.624 ms in the span of 1.125 ms as shown in Figure 5-60. The Pwm_IR_Timer configuraton is modified dynamically depending on the value of the bit to be transmitted. Note that with Pwm_IR_Timer, there are the following two types of interrupts:
  - On terminal count: This interrupt is used to detect the completion of transmission of a bit. For a '0,' the value is set to 1.125 ms and for '1,' the value is set to 2.25 ms.
  - On compare/capture count: This interrupt is used to detect the completion of transmission of the 38-KHz pulse train for a given bit and IR protocol.

The following sections discuss the configuration of the PWM Components.

Figure 5-62 Pwm_IR Component

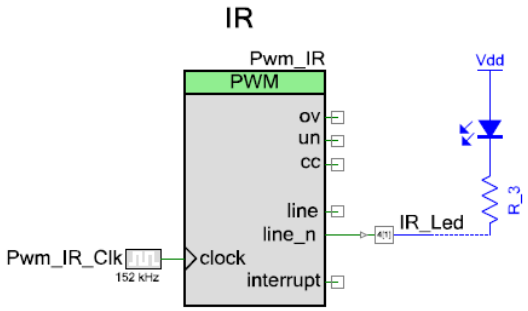
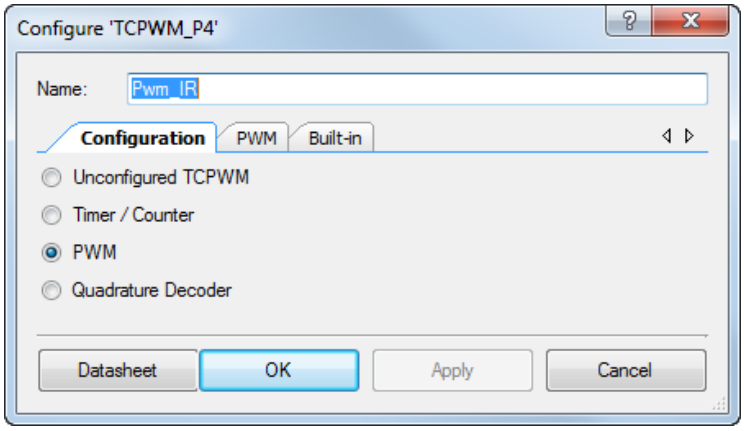
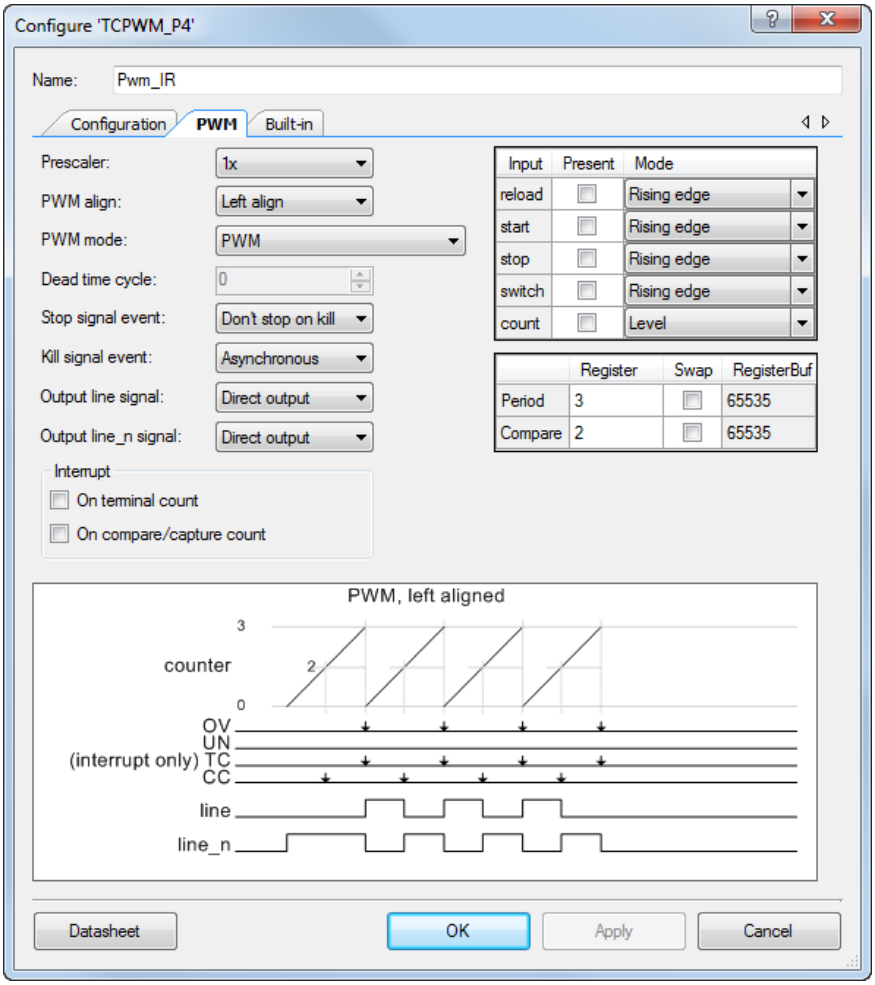


Figure 5-63. Configuration Tab of Pwm_IR Component



In the PWM tab, the initial period and compare values are set as shown in Figure 5-63. These values will generate a 38-kHz pulse train with a 50-percent duty cycle. The interrupts for **On terminal count** and **On compare/capture count** are disabled.

Figure 5-64. PWM Tab of Pwm_IR Component



See the Component datasheet for information on the APIs supported by the PWM Component. Click the **Datasheet** button at the bottom left corner of the PWM Component GUI shown in Figure 5-64 to access the datasheet.

Figure 5-65. Pwm_IR_Timer Component

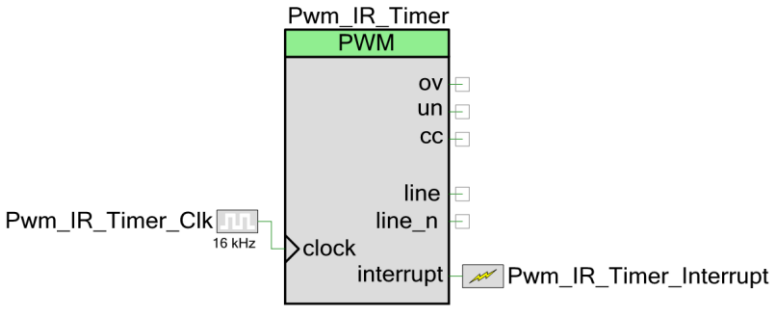
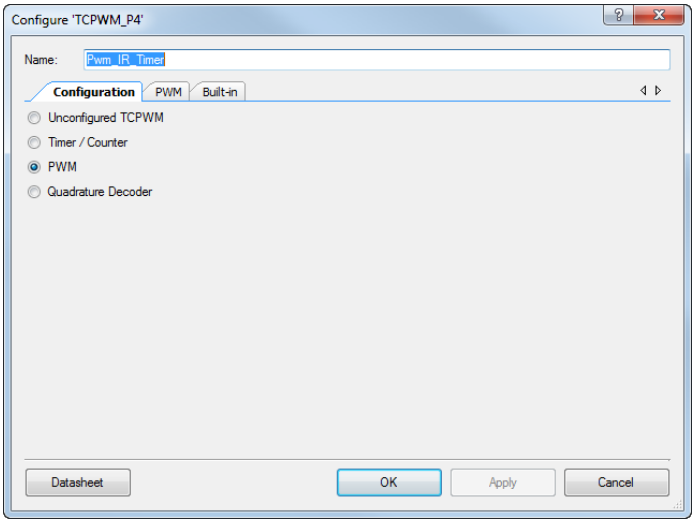
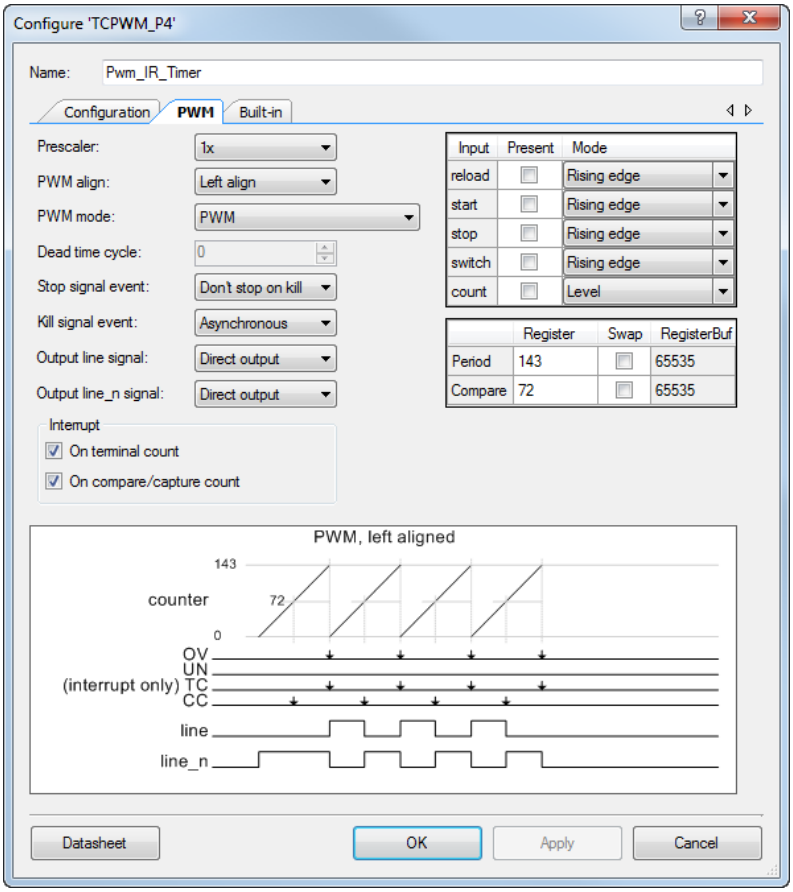


Figure 5-66. Configuration Tab of Pwm_IR_Timer Component



In the PWM tab, **t**he **Interrupts** for **On terminal count** and **On compare/capture count** are enabled, as shown in Figure 5-64.

Figure 5-67. PWM Tab of Pwm_IR_Timer Component



Note that there is an alternate implementation of the IR protocol that uses bit banging. Do the following to enable it:

1. Enable IR_PIN_TOGGLE macro in *platform.h.*
2. Remove the existing Pwm_IR Component and its associated GPIO and clock.
3. Rename Pwm_IR_1 to Pwm_IR. Also, rename the associated GPIO and clock in the Schematics Editor.

5.1.11.4 List of IR Key Codes

Table 5-7 lists the IR key codes used for the Samsung and NEC protocols.

Table 5-7. IR Key Codes for NEC Protocol

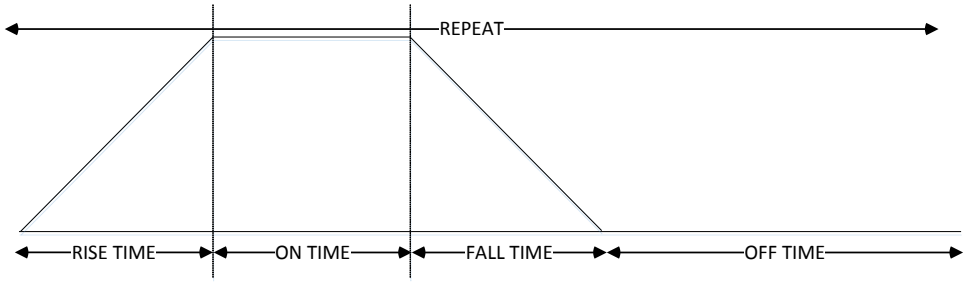
Key	IR Key Codes for Samsung Protocol	IR Key Codes for NEC protocol
Volume Up	0x07	0x1F
Volume Down	0x0B	0x40
Channel Up	0x12	0x49
Channel Down	0x10	0x4A
Source	0x01	0x02
Return	0x58	0x16
On/Off	0x02	0x00

5.1.12 LED Subsystem

The LED subsystem shows device notifications to the user through the red LED and supports the breathing effect (at different rates). The LED subsystem provides the following LED effects:

- LED ON: The LED is continuously in the ON state. The intensity can be varied by using PWM control. This indication is used to notify undirected advertisement (behavior is observed when the connect button is pressed on the remote control).
- LED breathing effect: The LED gradually increases in brightness and goes to the maximum brightness state. From the maximum brightness state, it goes back gradually to the OFF state. The entire LED brightness behavior is displayed in Figure 5-68. This effect supports different blink rates (slow and fast). A slow breathing effect is used to indicate a low battery, and a fast breathing effect is used to indicate directed advertisement to the host or undirected advertisement to the devices that is already present in the whitelist (behavior is observed if the remote control was bonded to at least one device earlier, and it is either out of range or received a disconnect).

Figure 5-68. Timing Diagram for Breathing Effect



RISE TIME: Time in milliseconds for rise time

ON TIME: Time in milliseconds for high standby time

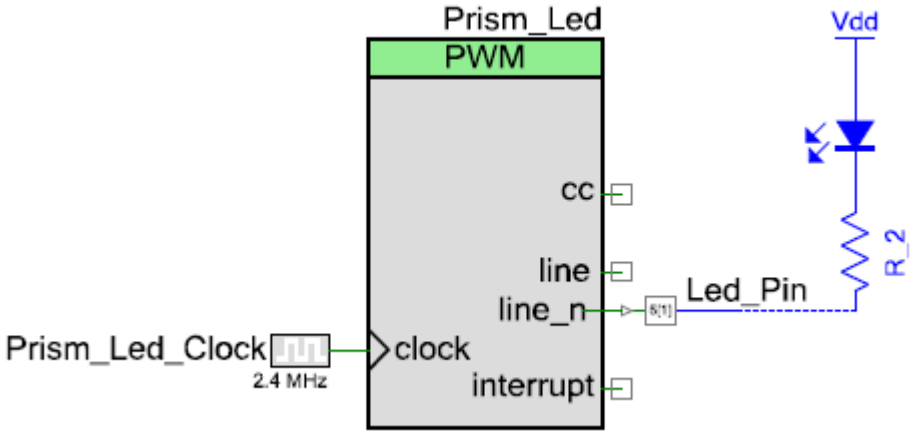
FALLTIME: Time in milliseconds for fall time

OFF TIME: Time in milliseconds for low standby time

REPEAT: Repeating the above pattern

The blinking or breathing effect on the LEDs is generated via the PWM Component (Figure 5-69) provided in PSoC Creator. The following sections explain the configuration of the PWM Component.

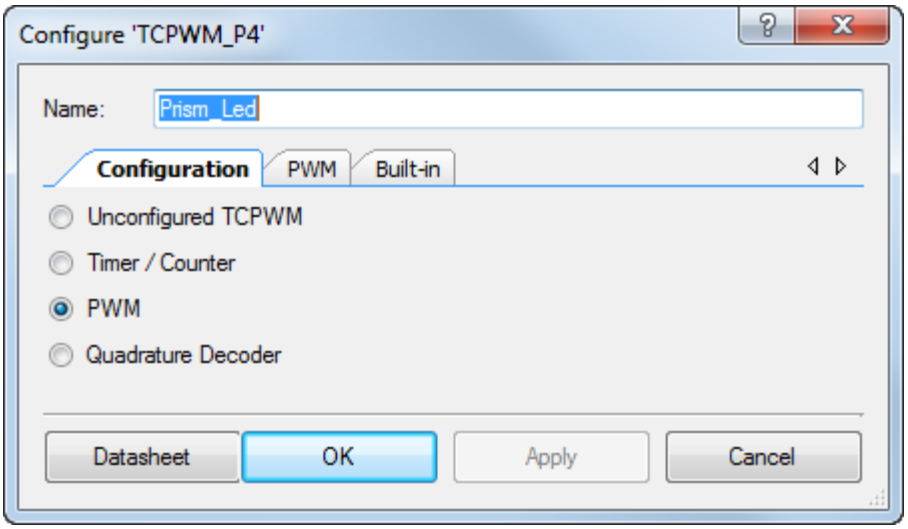
Figure 5-69. PWM Component



5.1.12.1 Configuration Tab

The **PWM** option is selected by default, as shown in Figure 5-70.

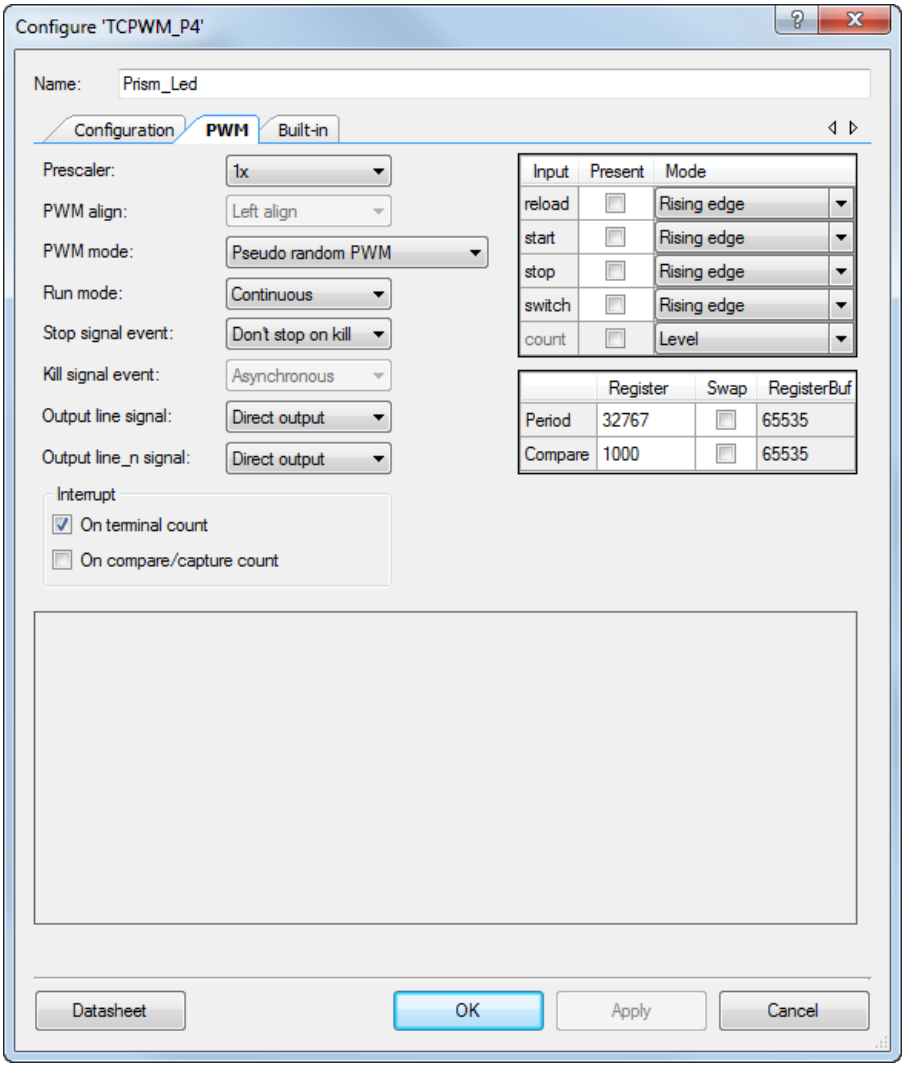
Figure 5-70. Configuration Tab of TCPWM Component



5.1.12.2 PWM Tab

The **PWM mode** is set as “Pseudo random PWM.” The **Period** and **Compare** values are set to “32767” and “1000” respectively, as shown in Figure 5-71.

Figure 5-71. PWM Tab of TCPWM Component



The datasheet provides a list of APIs supported by the TCPWM Component. To go to the datasheet, click the **Datasheet** button at the bottom left corner of the TCPWM Component GUI shown in Figure 5-71.

5.1.13 Flash Subsystem

The flash subsystem supports storing and retrieving of the peer device BD (Bluetooth Device) address and the motion sensor gyro offset data. The remote control receives the BD address of the peer device on a connection event. This address is stored in flash memory. The peer device BD address is needed for directed advertisement; therefore, it is retrieved from flash every time the remote control system power is reset.

The data stored in the flash by the firmware is protected using a 8-bit checksum. The firmware provides an option to disable writes to flash by disabling the macro DISABLE_FLASH in the *platform.h* file.

5.1.14 Debug Subsystem

The debug subsystem is used to print debug information on the UART terminal. This subsystem is disabled by default. The firmware provides an option to enable debug by enabling the DEBUG_PRINT or SOFTWARE_UART macro in the *platform.h* file. If the DEBUG_PRINT macro is enabled, you should enable the SCB UART Component in the schematic view of PSoC Creator (Debug_Uart). If the SOFTWARE_UART macro is enabled, enable the software Transmit UART Component (SW_Tx_UART). In addition, assign the corresponding pins from these Components to the appropriate GPIOs in *CY5672_Remote_Control.cydwr*.

The debug subsystem provides an option to print the selected debug logs by setting the appropriate debug level. For example, if the default debug level is set to '0', during Device_FW_Init(), the debug prints with a setting greater than '0' are printed.

5.2 Firmware for the PRoC BLE and PSoC 5LP on the CySmart USB Dongle

The CySmart USB dongle firmware comprises the firmware for both PRoC BLE and PSoC 5LP devices. Note that these two devices each have their own separate PSoC Creator projects as listed Table 5-8, although they are described together in the following section.

Table 5-8. CySmart USB Dongle Projects

Silicon Part	Associated Project Name
PRoC BLE	BLE_HID_CySmart_Dongle.cywrk
PSoC 5LP	CY5672_Dongle_Bridge.cywrk

The CySmart USB dongle has two modes of operation. The dongle can only operate in one mode at a time:

- HID host: In this mode, the dongle acts as the HID host and transfers the BLE data to USB. This is the normal operating mode.
- CySmart emulator: In this mode, the dongle performs all BLE operations triggered via the CySmart tool. This is used for examining the BLE data directly.

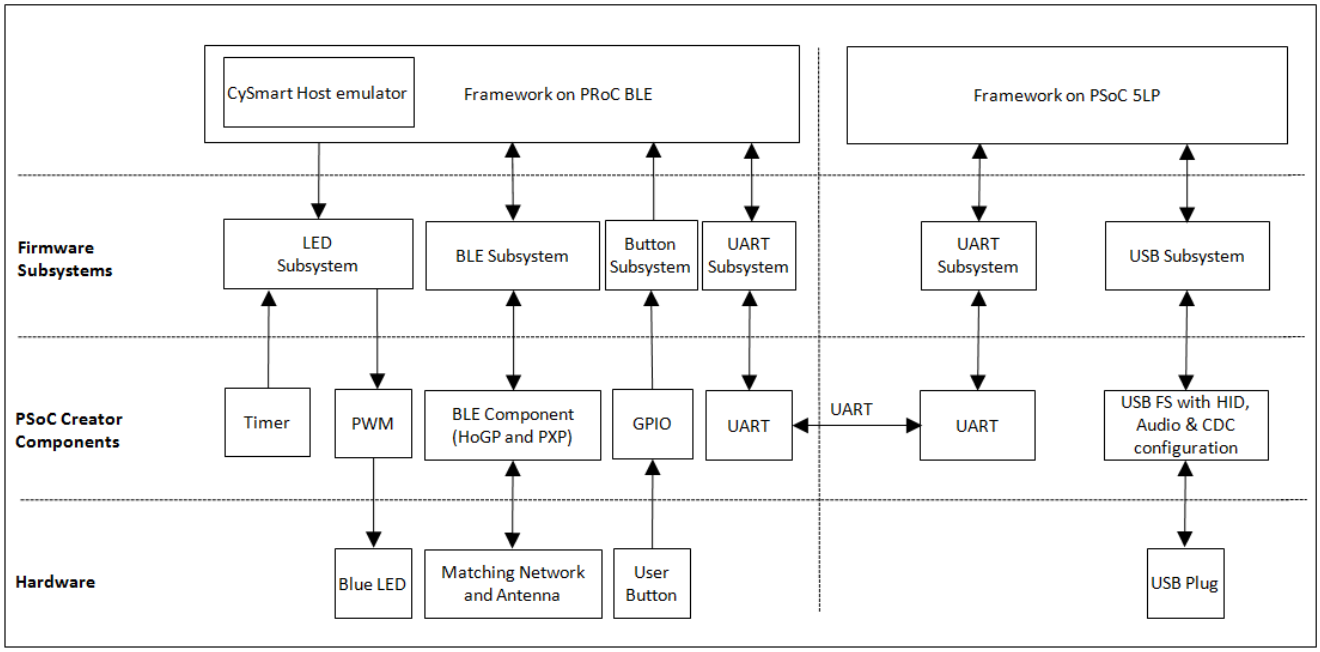
When powered on, the dongle is in the HID host mode and communicates with the HOGP-complaint BLE device. On connecting the dongle to the CySmart tool, the dongle switches to the CySmart emulator mode.

On disconnection from CySmart tool, the dongle switches back to the HID mode.

5.2.1 Firmware Architecture

The firmware for the CySmart USB dongle consists of the framework (on PRoC BLE and PSoC 5 LP) and various input/output subsystems, as shown in Figure 5-72.

Figure 5-72. PRoC BLE Firmware Architecture for CySmart USB Dongle



In HID mode, only the audio interface and HID interface will be active. The PRoC BLE device will format the HID/audio packet along with a report ID and send it to PSoC 5LP with a unique code for each HID endpoint. For audio, PSoC 5LP will perform ADCPM decompression on an audio packet and send it to the USB audio interface.

In CySmart mode, only the UART interface will be used; other interfaces will be inactive. PSoC 5LP sends the UART data to the PC using a virtual COM port via the USB interface.

The dongle firmware includes the following subsystems:

- BLE subsystem (PRoC BLE)
- LED subsystem (PRoC BLE and PSoC 5LP)
- Button subsystem (PRoC BLE)
- UART subsystem (PRoC BLE and PSoC 5LP)
- USB subsystem (PSoC 5LP)

These subsystems use PSoC Components. Figure 5-73 shows a schematic view of the PRoC BLE device, and Figure 5-74 shows a schematic view of the PSoC 5LP device.

Figure 5-73. Schematic View of BLE HID CySmart Dongle Firmware (PRoC BLE)

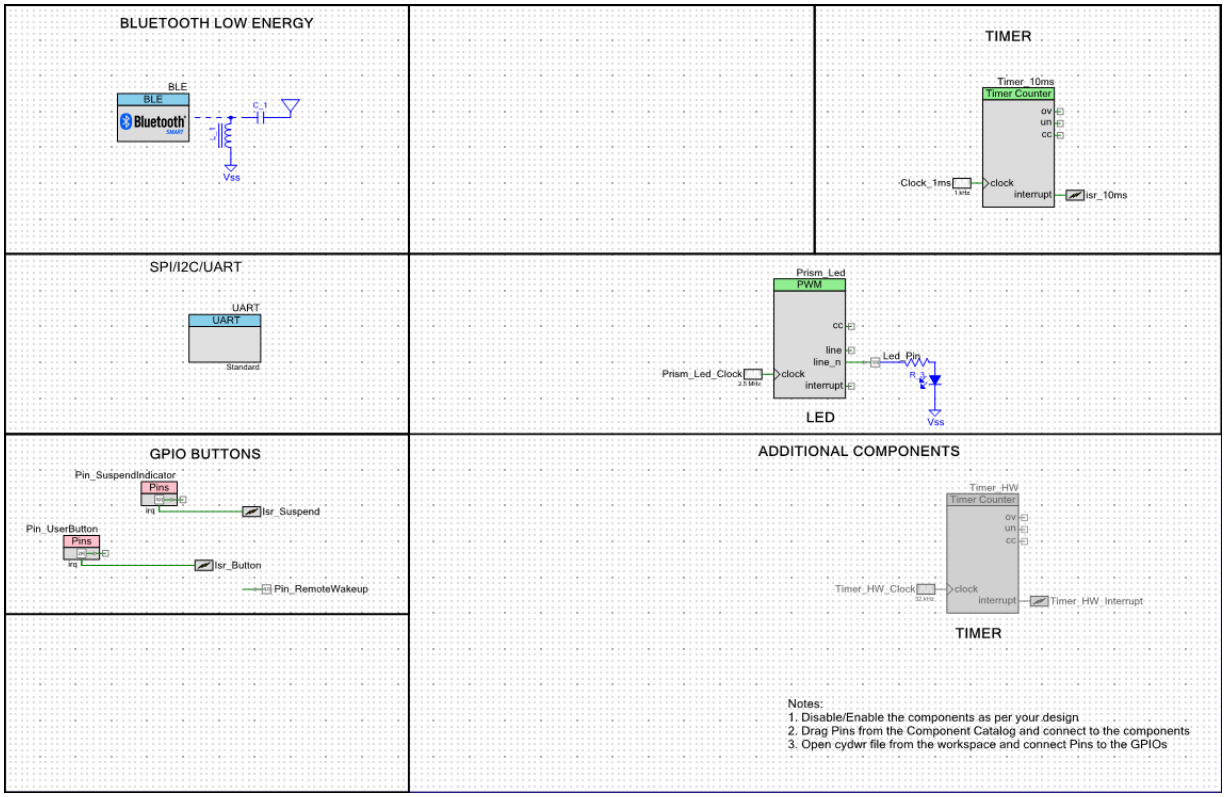
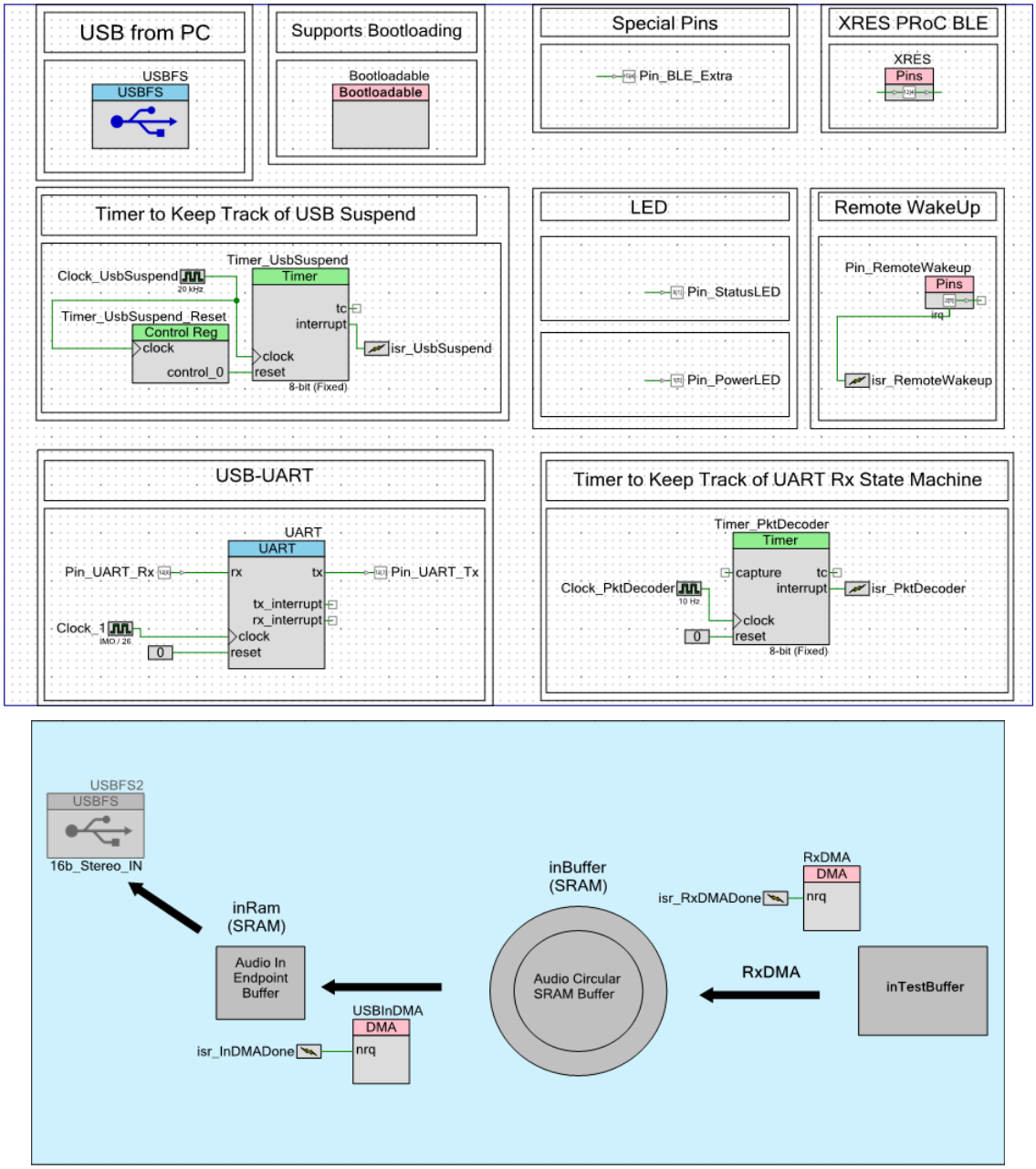
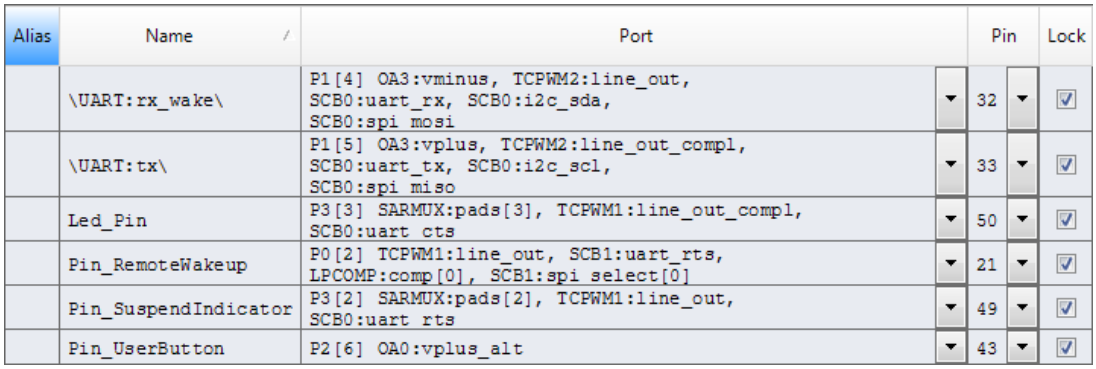


Figure 5-74. Schematic View of CY5672 Dongle Bridge Firmware (PSoC 5LP)



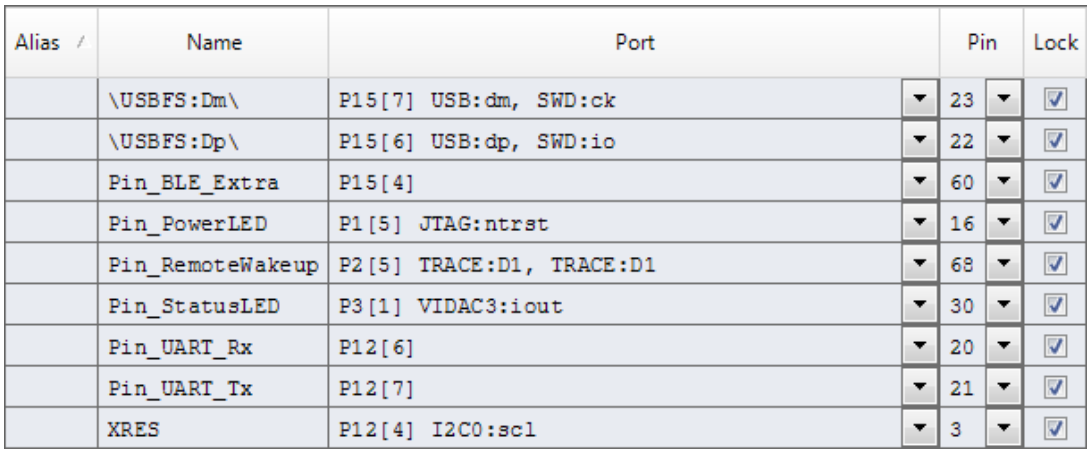
The GPIOs are configured per the dongle hardware connections, as shown in Figure 5-75 and Figure 5-76.

Figure 5-75. GPIO Configuration of BLE HID CySmart Dongle Firmware (PRoC BLE)



Alias	Name	Port	Pin	Lock
	\UART:rx_wake\	P1[4] OA3:vminus, TCPWM2:line_out, SCB0:uart_rx, SCB0:i2c_sda, SCB0:spi_mosi	32	☑
	\UART:tx\	P1[5] OA3:vplus, TCPWM2:line_out_compl, SCB0:uart_tx, SCB0:i2c_scl, SCB0:spi_miso	33	☑
	Led_Pin	P3[3] SARMUX:pads[3], TCPWM1:line_out_compl, SCB0:uart_cts	50	☑
	Pin_RemoteWakeup	P0[2] TCPWM1:line_out, SCB1:uart_rts, LPCOMP:comp[0], SCB1:spi_select[0]	21	☑
	Pin_SuspendIndicator	P3[2] SARMUX:pads[2], TCPWM1:line_out, SCB0:uart_rts	49	☑
	Pin_UserButton	P2[6] OA0:vplus_alt	43	☑

Figure 5-76. GPIO Configuration of CY5672 Dongle Bridge Firmware (PSoC 5LP)



Alias	Name	Port	Pin	Lock
	\USBFS:Dm\	P15[7] USB:dm, SWD:ck	23	☑
	\USBFS:Dp\	P15[6] USB:dp, SWD:io	22	☑
	Pin_BLE_Extra	P15[4]	60	☑
	Pin_PowerLED	P1[5] JTAG:ntrst	16	☑
	Pin_RemoteWakeup	P2[5] TRACE:D1, TRACE:D1	68	☑
	Pin_StatusLED	P3[1] VIDAC3:iout	30	☑
	Pin_UART_Rx	P12[6]	20	☑
	Pin_UART_Tx	P12[7]	21	☑
	XRES	P12[4] I2C0:scl	3	☑

Table 5-9 describes the file structure used by the projects and lists the key functions implemented in each file.

Table 5-9. File Structure and Key Functions

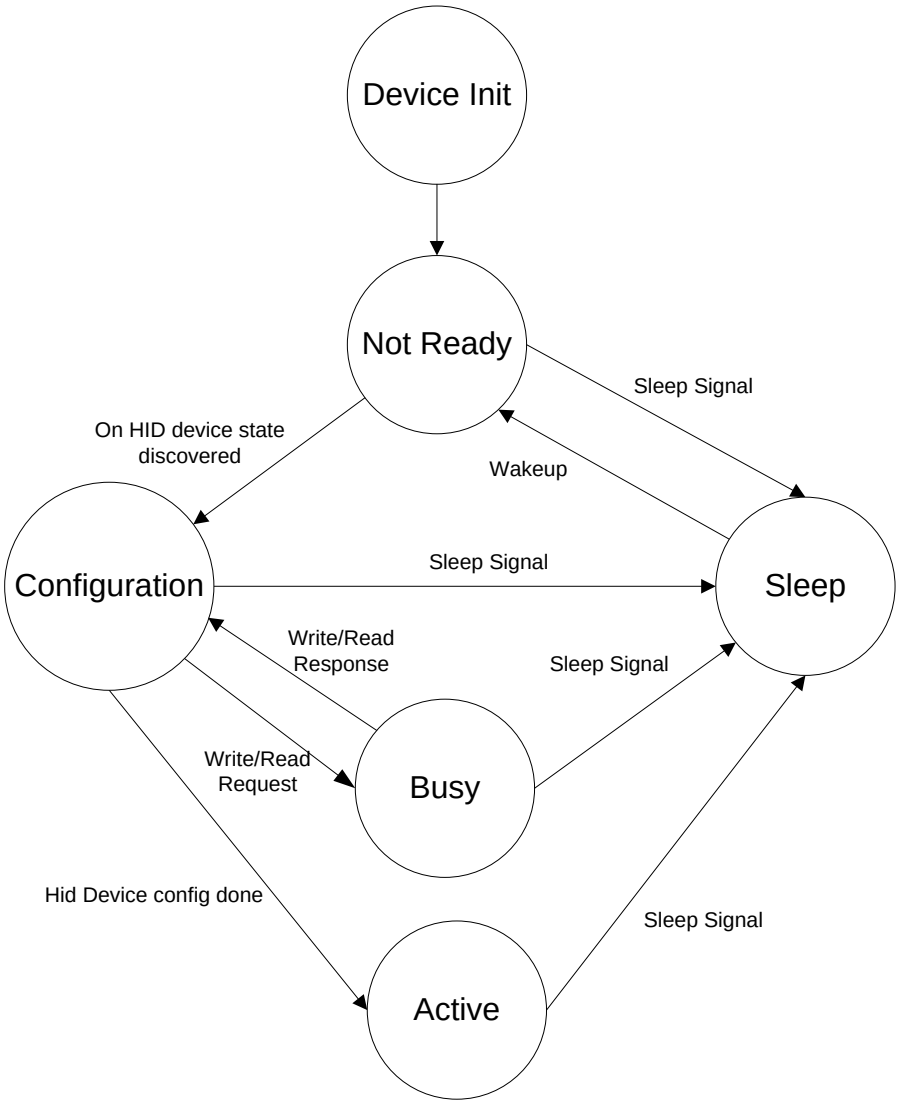
File Name	Details
Project: BLE_HID_CySmart_Dongle (PRoC) main.c	This is the top-level application file. It initializes the system and runs the main loop. ***main()*** – The main function for the application, which handles the switching between HID mode and CySmart mode ***Device_Init()*** – Initializes all the blocks (BLE, UART, LED, Timer, and GPIOs) of the system ***Device_Timer_Callback()*** – Handles the timer interrupt configured for the LED module

File Name	Details
Project: BLE_HID_CySmart_Dongle (PRoC BLE) Application.c	This file handles the HID mode functionality of the project. It handles the BLE events and the state machine of the HID host. ▪ ***BLE_Init()* –** Handles the BLE-specific initialization and registers the callback functions for each profile. The BLE Component is started with HID mode configuration. ▪ ***BLE_DeInit()* –** De-registers the callback functions for each profile and stops the HID mode BLE Component. ▪ ***Dongle_Ble_State_Handler()* –** Handles the state machines for BLE scanning, connection, GATT discovery, and profile-level configurations. ▪ ***FilterScanResponsePackets()* –** Parses the advertisement packet from the device and filters the packets for the HID service Universally Unique Identifier (UUID). The dongle will connect to devices containing the HID service or to devices sending a directed advertisement to the dongle. ▪ ***SendEmptyReports()* –** Sends a dummy report on BLE disconnection. If any key down packet is sent by the device before disconnection, this dummy report will ensure that the key press state is reverted. ▪ ***BleCallBack()* –** Handles general events from the BLE stack such as disconnect, timeout, scan response, authentication events, and so on. ▪ ***HidsCallBack()* –** Handles the HID-specific events. This function gathers the report IDs of each report reference descriptor to match with the notification data. On receiving HID notification, this function matches it to the expected report ID and sends the report to the USB Bridge controller. ▪ ***BasCallBack()* –** Handles the BAS-specific events for notification enable and notification events. The Battery notification will be sent as an HID feature report to the USB interface. ***Process_Suspend_Command()*** – Handles the USB suspend command from the USB Bridge controller. The system will go into sleep mode as long as the suspend command is active. This function also sends the HID Suspend command to the HID device if it is connected. This will inform the device to enter sleep mode.
Project: BLE_HID_CySmart_Dongle (PRoC BLE) led.c	This module is same as described in the remote control project. See the LED Subsystem section.
Project: BLE_HID_CySmart_Dongle (PRoC BLE) timer.c	This module is same as described in the remote control project. See the Timer Subsystem section. The watchdog timer–based tick timer is used by default. The firmware has a provision to use the Timer_HW Component instead of the watchdog timer by doing the following: ▪ Enable the **ENABLE_TIMER_COMPONENT** macro in *Timer.h*. ▪ Disable the **ENABLE_WDT_TIMER** macro in *Timer.h*. ▪ Right-click on the **Timer_HW** Component in the schematic view and select “enable”.
Project: CY5672_Dongle_Bridge (PsoC 5LP) main.c	This is the top-level application file. It initializes the system and runs the main loop. ▪ ***main()*** – Handles the USB reset and runs the main loop to monitor UART and USB data transfers. ▪ ***Device_Init()*** – Initializes all the blocks of the system, waits for the USB enumeration, and sets the enumeration LED. This function also resets PRoC BLE after system initialization to ensure that both the controllers are in sync.

File Name	Details
Project: CY5672_Dongle_Bridge (PsoC 5LP) USB_Interface.c	This file handles the bridge functionality between the USB and UART interfaces. ▪ ***UARTTransmit()*** – Checks for USB CDC OUT data and if it is present sends it to the UART by configuring the expected baud rate by PRoC BLE. ▪ ***RxPktDecoder ()*** – Parses the UART packets and loads them into the respective endpoint buffers. This runs a state machine on each RX byte and looks for the packet format. Predefined opcodes are used to identify the endpoint to which the decoded packet is to be sent. If a proper packet is detected that does not belong to the known opcodes, it will be routed to the CDC endpoint, which will be used by the CySmart tool. ▪ ***SendToUSB()*** – Monitors each endpoint status and loads the pending data when the endpoint is available. For the mouse and keyboard endpoints, both endpoints must be free before a new packet can be sent so that the HID combo key order is maintained.
Project: CY5672_Dongle_Bridge (PsoC 5LP) AudioControl.c	This file handles the USB audio IN control and data path handling. ▪ ***ConfigureAudioPath()*** – Initializes and configures the DMA channels required for the audio data path. This needs to be called after USB enumeration. ▪ ***ProcessAudioIn()*** – Handles the data transfer from the circular buffer to the USB endpoint. ▪ ***HandleAudioInDisable()*** – Stops the audio data path on a control request from USB. ▪ ***HandleUnderflow()*** – Maintains the dummy packet to be available for the USB Audio interface if no valid data is available.
Project: CY5672_Dongle_Bridge (PsoC 5LP) ADPCMDecoder.c	This file handles the decompression of a nibble into a 2-byte word using the ADPCM decoder. ▪ ***ADPCMDecoder()*** – Decompresses a nibble into a 2-byte voice sample using ADPCM.
Project: CY5672_Dongle_Bridge (PsoC 5LP) Interrupts.c	This file contains the ISRs for all the interrupts used in the system. ***InDMADone_Interrupt()*** – Handles the data transfer to USB from the circular buffer. ***RxDMADone_Interrupt()*** – Handles the data transfer to the circular buffer from the UART. ***isr_PktDecoder()*** – Handles the timeout for the UART packet decoder state machine.

The framework running on PRoC BLE adheres to the state machine for the HID host mode, as shown in Figure 5-77.

Figure 5-77. State Machine for PRoC BLE Firmware



The following sections describe the states in the state machine.

5.2.1.1 Device Init State

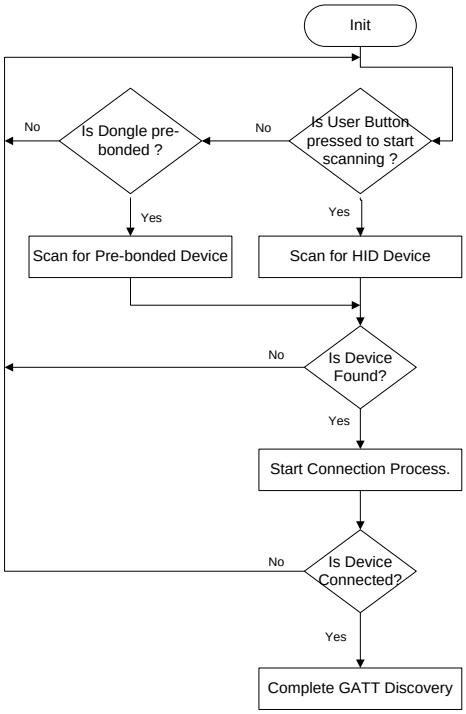
In the Device Init state, the following actions are performed:

- All global variables are initialized.
- The UART Component is initialized and started.
- ISRs for UART, button, and sleep are initialized and started.
- The LED module is initialized and started.
- The BLE Component is started, and the callback functions for events are registered.

5.2.1.2 Not Ready State

The dongle will enter the Not Ready state on each power cycle. This state represents that the BLE Component is not ready to accept any commands. In this state, the firmware continuously scans for a peripheral device until it gets a scan response from an HID device, and then it issues a connect request. On successfully connecting to the HID device, the dongle starts discovering the characteristics exposed by the server, as shown in Figure 5-78. When the discovery is complete, the device transitions to the Configuration state.

Figure 5-78. Code Flow in BLE Not Ready State



5.2.1.3 Configuration State

In the Configuration state, the dongle starts configuring the connected HID device per the required behavior. It performs the following operations on every new connection:

- Completes the bonding procedure with the security level “Unauthenticated pairing with encryption.” On successful bonding, the keys will be stored in flash.
- Reads the “Peripheral preferred connection parameters” attribute and updates the connection parameters of the BLE link.
- Configures the “HID Control Point” characteristic with the value “Exit Suspend,” signaling to the HID device that the HID host device is exiting the power-saving mode.
- Configures the HID protocol mode with the value “Report Protocol Mode.”
- Configures the Immediate Alert characteristic with “No alert.” This alert level will be changed when the path loss is below the threshold value.
- Reads the Tx Power Level to calculate the path loss as required for the proximity monitor: Path loss = Tx Power – RSSI.
- Reads the “Report Reference descriptor index” that contains the report ID mapping for each type of HID report configured by the HID server, The dongle is configured to support a maximum of six HID input records. The dongle does not support HID output reports.
- Configures the Link Loss Alert Level characteristic with a high alert level.
- Enables notifications for the Scan Refresh and battery-level characteristics.
- Enables notifications for all the HID report types: Mouse, Keyboard, multimedia, and power reports.

On each GATT request, which requires a response, the PRoC BLE application framework enters the Busy state until the corresponding pass/fail response is received from a peripheral such as a mouse or remote control. When the configuration is complete, the PRoC BLE application framework enters the Active state.

5.2.1.4 Busy State

This state indicates that the dongle is waiting for a response from a peer device and is not available to issue the next BLE command.

5.2.1.5 Active State

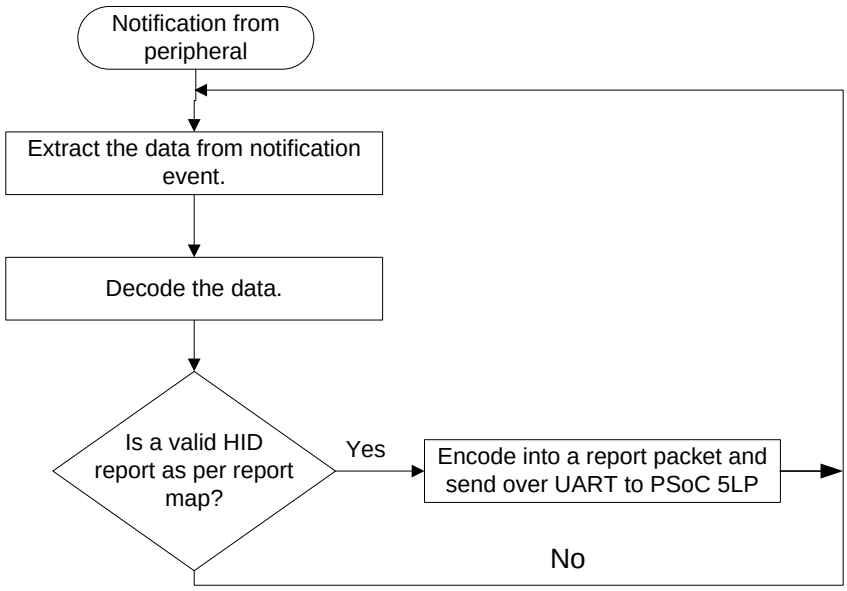
This state indicates that the dongle has completed the configuration of the HID device and is available for BLE commands.

The RSSI level is monitored continuously; when the signal strength is below the threshold, the dongle sets an immediate alert with “High alert,” which will be used by the peer device to signal the user about the signal quality. This alert will revert to “No alert” if the signal strength exceeds the threshold.

Note: By default, the proximity profile implementation is disabled. It can be enabled using the ENABLE_PATH_LOSS_ALERT macro in the *Application.c* source file.

When the dongle reaches the Active state, it starts processing HID report notifications, encodes them into the UART packet format, and sends them to PSoC 5LP. Figure 5-79 shows the flow chart of this operation when the HID report is received from the BLE HID device.

Figure 5-79. Code Flow for HID Notifications



5.2.1.6 Sleep State

A host PC issues the USB suspend command if the PC is in sleep mode. To enable the PRoC BLE device to detect the suspend command, one GPIO of PSoC 5LP is used as a suspend indicator. This GPIO is triggered by PSoC 5LP on receiving the USB suspend command.

PRoC BLE monitors this pin for sleep and wake commands by configuring a GPIO interrupt. The dongle enters the low-power mode on suspend; it exits the low-power mode when the pin state changes. This transition can happen in all other states as well.

5.2.2 BLE Subsystem

The PRoC BLE–based firmware configures the BLE Smart Component available in PSoC Creator with HOGP and the profile role as “Report and Boot Host (GATT client)” with support for the following services:

- Device Information
- Scan Parameters
- Battery
- Human Interface Device

- Immediate Alert
- Link Loss
- Tx Power

A GATT server (HID device) configured as a peripheral (GAP role) can start an advertisement to connect to the central device (CySmart USB dongle). When the peripheral advertises, the client detects the device and connects to it. After connecting to the device, the client discovers the services provided by the HID device and enables notifications for the characteristics it supports.

After the client configures the peripheral device, it receives data when the peripheral device sends a notification. The received data is then decoded by the application logic and transmitted to the PSoC 5LP firmware via UART.

The BLE subsystem is built on the BLE Component provided in PSoC Creator, as shown in Figure 5-80.

Figure 5-80. BLE Component of PSoC Creator

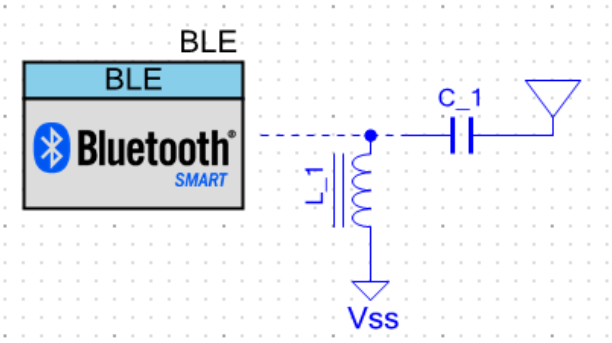


Figure 5-81 and Figure 5-82 show the configuration for the BLE Component.

Figure 5-81. BLE Component General Configuration

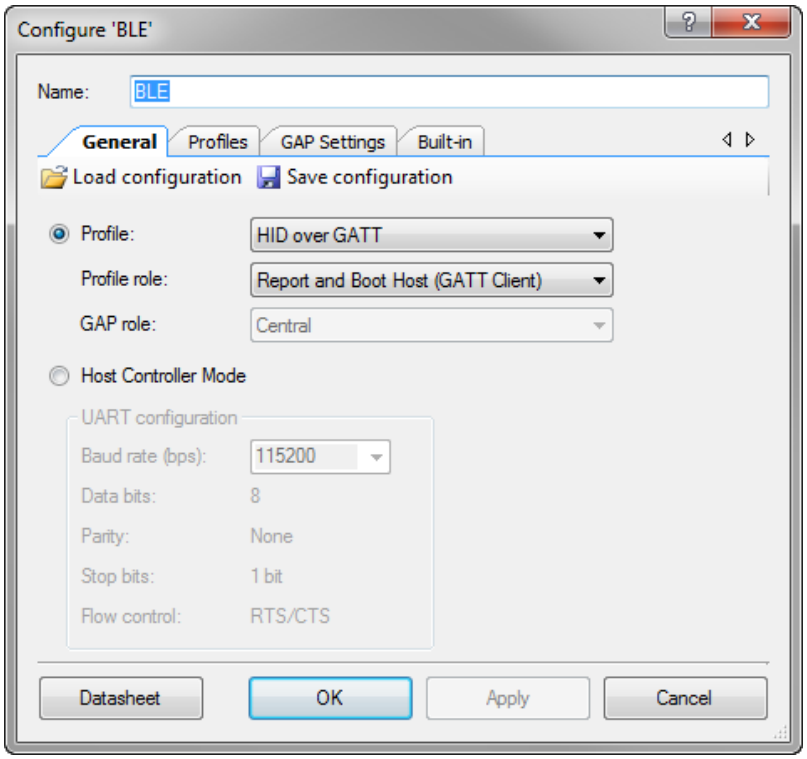
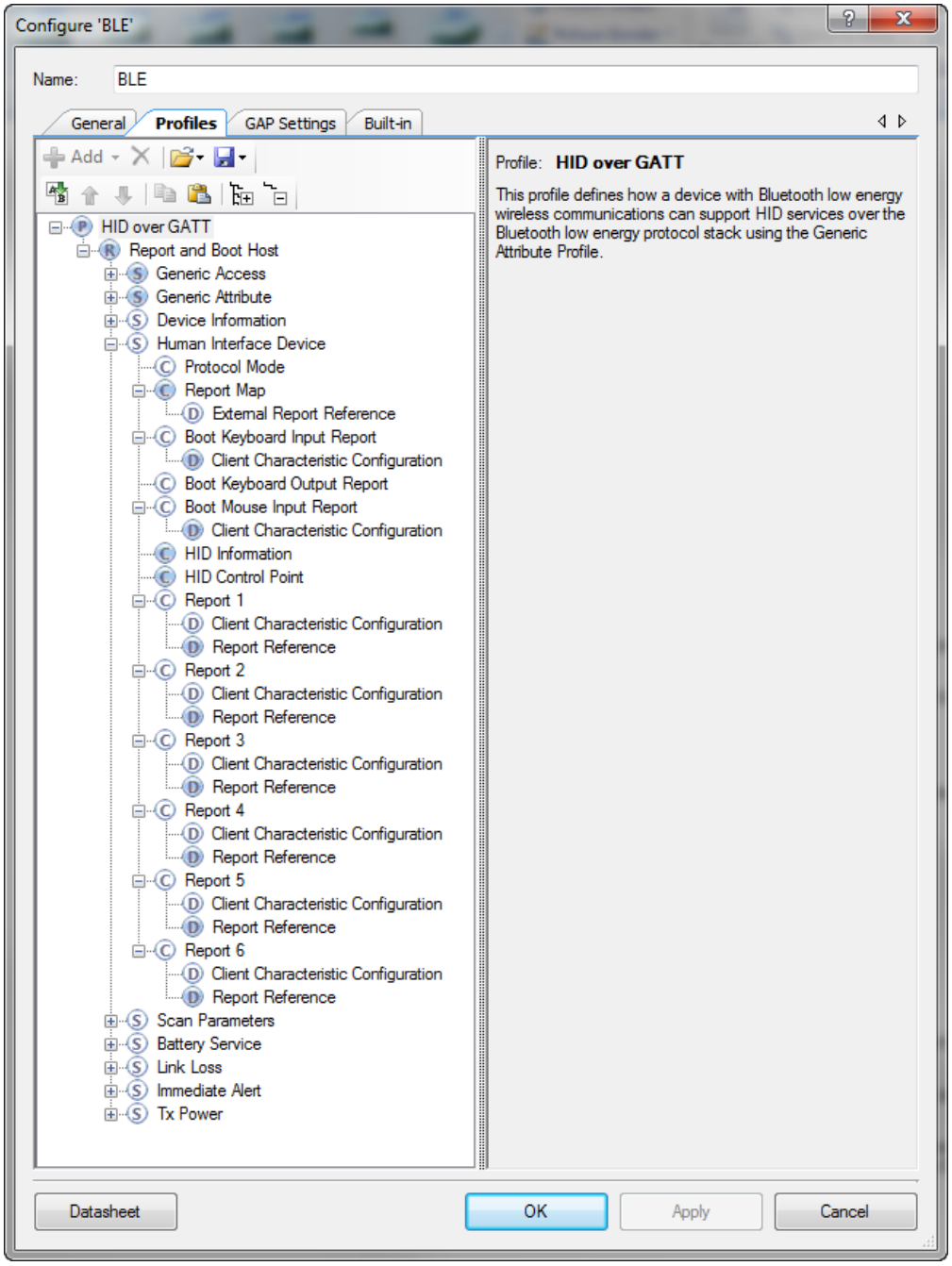


Figure 5-82. Profiles Supported by the Host Device



Open the BLE Component wizard from the project for more details about the Component configuration.

All the HID report notifications except keyboard are appended with the report ID before sending to the UART to match the report with the USB HID report map configured in the USB subsystem.

Audio data and control notifications will be received as HID feature reports, which will be forwarded to the UART system.

5.2.3 LED Subsystem

The dongle has three LEDs. The red and green LEDs are connected to PSoC 5LP, while the blue LED is connected to PRoC BLE. The blue LED has three modes of operation:

- Blinking effect: The dongle is scanning for new HID devices; this is initiated by pressing the user button (SW2).
- Slow breathing effect: The dongle is scanning for preconnected devices.
- Stay on: The BLE connection to the HID device is active.

The LED subsystem uses the PWM and Timer Components to create the effects required by the dongle. The design is the same as that of the remote control explained in the LED Subsystem section.

PSoC 5LP has two LEDs:

- The red LED indicates the power status. It is ON if the dongle is powered.
- The green LED indicates the USB status.
  - Stays on upon successful enumeration
  - Blinks if the dongle is in the bootloader mode

5.2.4 Button Subsystem

The dongle has one user button, which is configured for an edge-triggered input. This button is used as the pairing button, which triggers the dongle to scan for new BLE HID peripherals.

The dongle, by default, will only scan for prebonded devices if they are present in the bonded list. If no devices are present in the list, the dongle will wait for a button press to start scanning.

When this button is pressed, the dongle disconnects the current connection if one exists. It starts scanning for new devices by ignoring the prebonded device list. So, whenever you want to connect a new HID device to the dongle, press the pairing button.

5.2.5 UART Subsystem

The UART subsystem is the core subsystem in the dongle operation. The UART interface connects the PRoC BLE and PSoC 5LP controllers to each other.

The PRoC BLE UART subsystem frames the packet for each type of data using the format described in the following section.

The PSoC 5LP UART subsystem decodes the packet and routes it to the corresponding USB endpoints.

5.2.5.1 PRoC BLE UART

The UART subsystem is built on the UART Component provided in PSoC Creator, as shown in Figure 5-83. The UART Component is configured as shown in Figure 5-84 and Figure 5-85.

Figure 5-83. UART Component of PSoC Creator

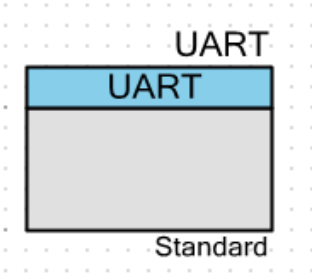


Figure 5-84. UART Component UART Basic Tab

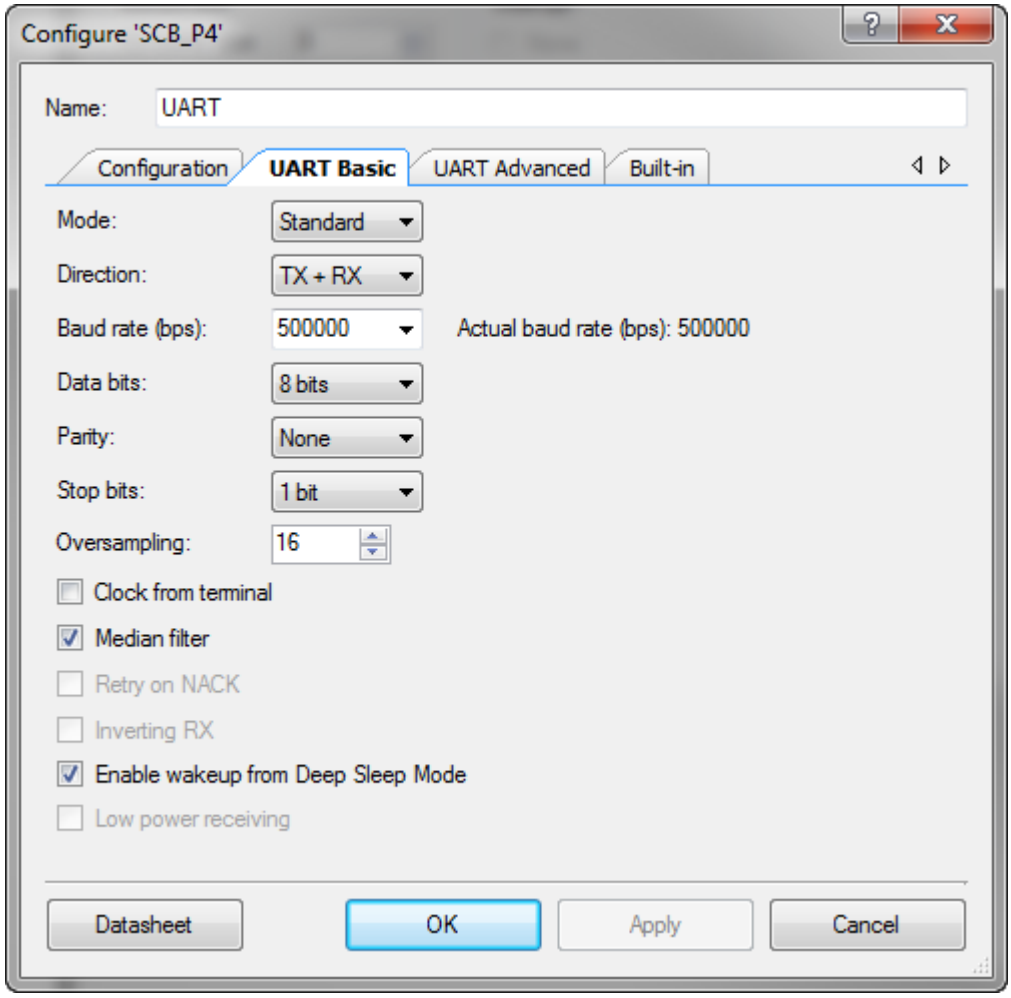
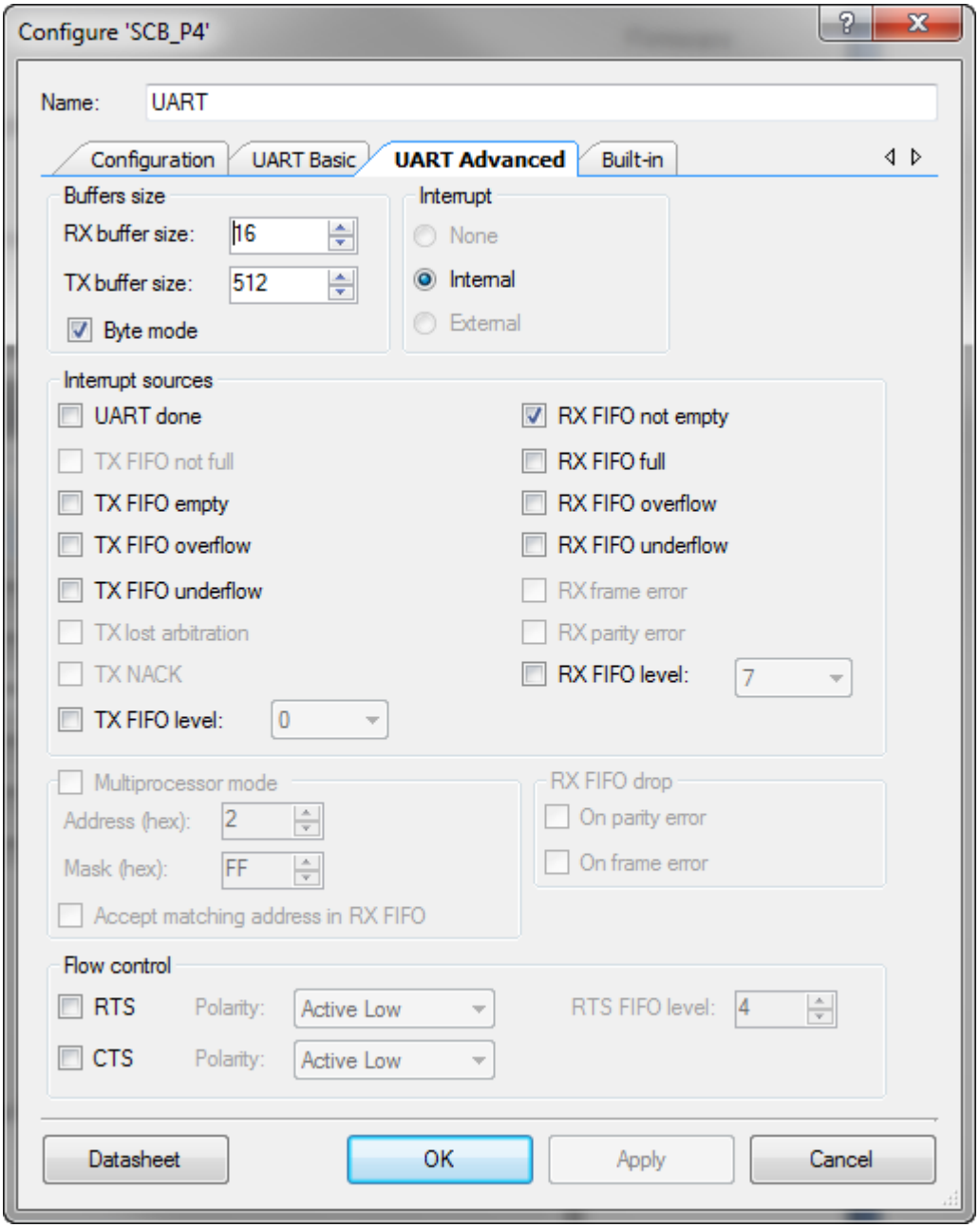


Figure 5-85. UART Component UART Advanced Tab



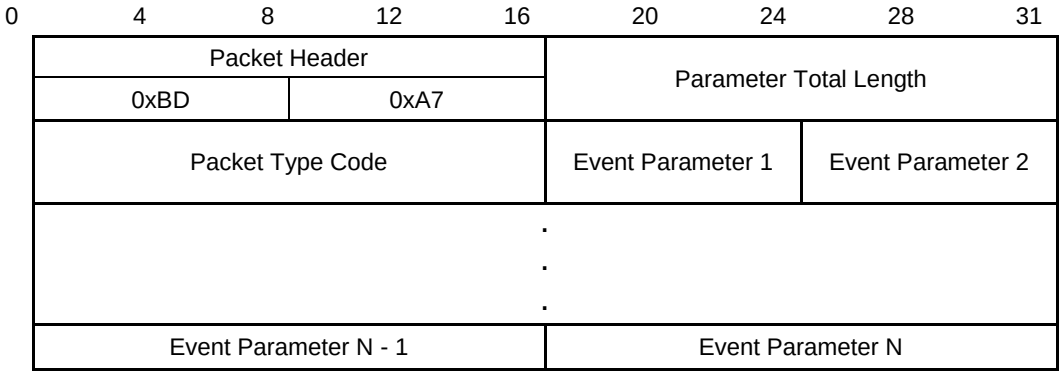
This subsystem has two modes of operation:

- HID host mode
- CySmart emulator mode

In the HID host mode, the subsystem transfers the formatted packets to the UART. The BLE subsystem on PRoC BLE packetizes the received data notifications (mouse, keyboard, audio, multimedia, battery, and power data) over the BLE link and sends them to the UART subsystem.

The UART packet structure shown in Figure 5-95 is decoded by the PSoC 5LP UART subsystem.

Figure 5-86. UART Packet Structure



Each row in Figure 5-86 corresponds to 4 bytes of data.

The packet type codes are predefined to differentiate among types of data to be sent to the PC:

```
MOUSE_REPORT          = 0x0461u
KEYBOARD_REPORT       = 0x0462u,
AUDIO_REPORT          = 0x0463u,
AUDIO_CONTROL_STATUS  = 0x0464u,
AUDIO_SYNC_PACKET     = 0x0465u
```

In PSoC 5LP, the mouse endpoint is configured to use the report ID, whereas the keyboard endpoint does not use the report ID. See the USB Subsystem section.

To match this configuration, the report ID number is appended to the mouse report notifications before sending to the UART. Keyboard notifications will be sent only by adding the packet header field.

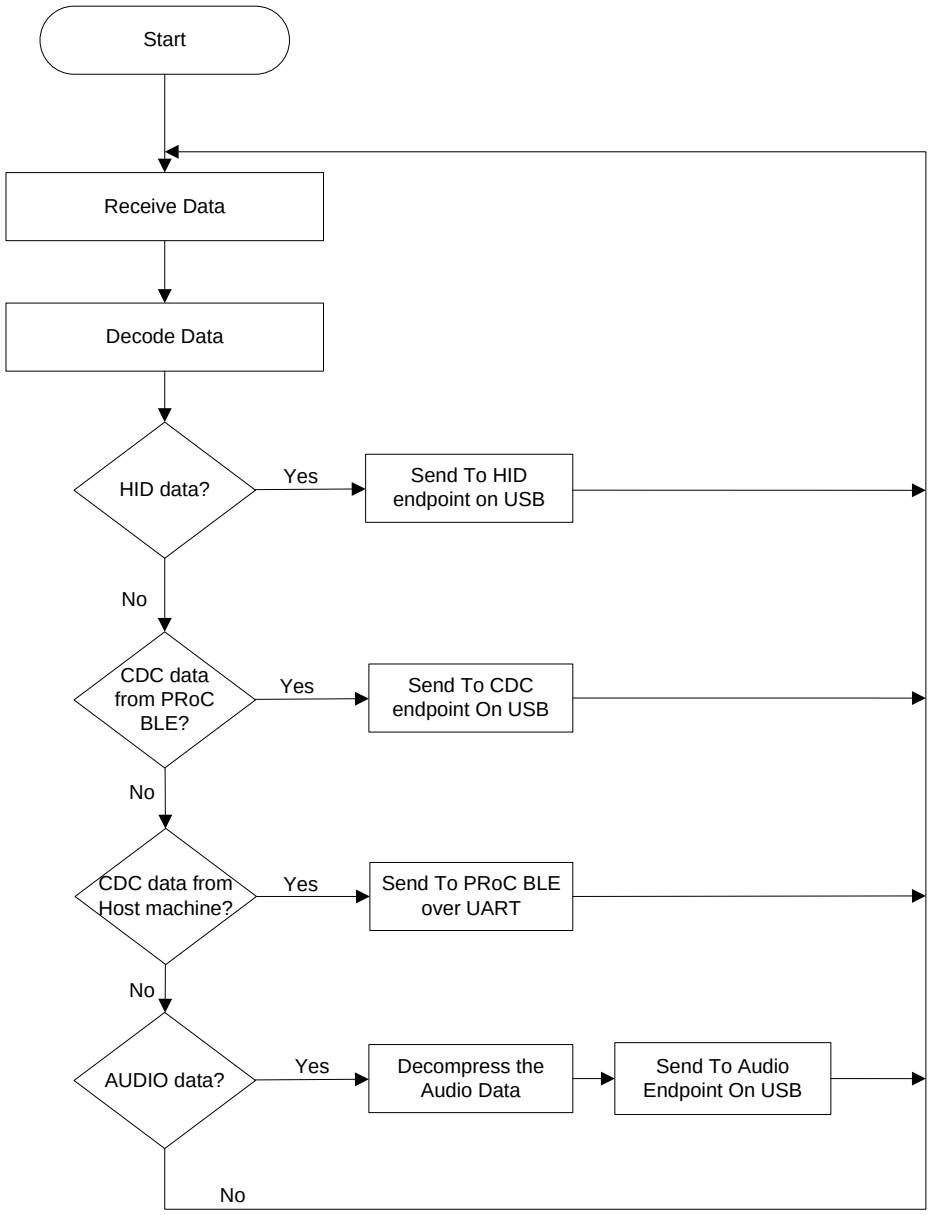
The HID audio feature reports are packetized into audio control and data packets and sent to the UART.

In the CySmart emulator mode, the UART subsystem uses a custom command-event protocol to communicate with the CySmart tool via a custom CDC interface.

5.2.5.2 PSoC 5LP UART

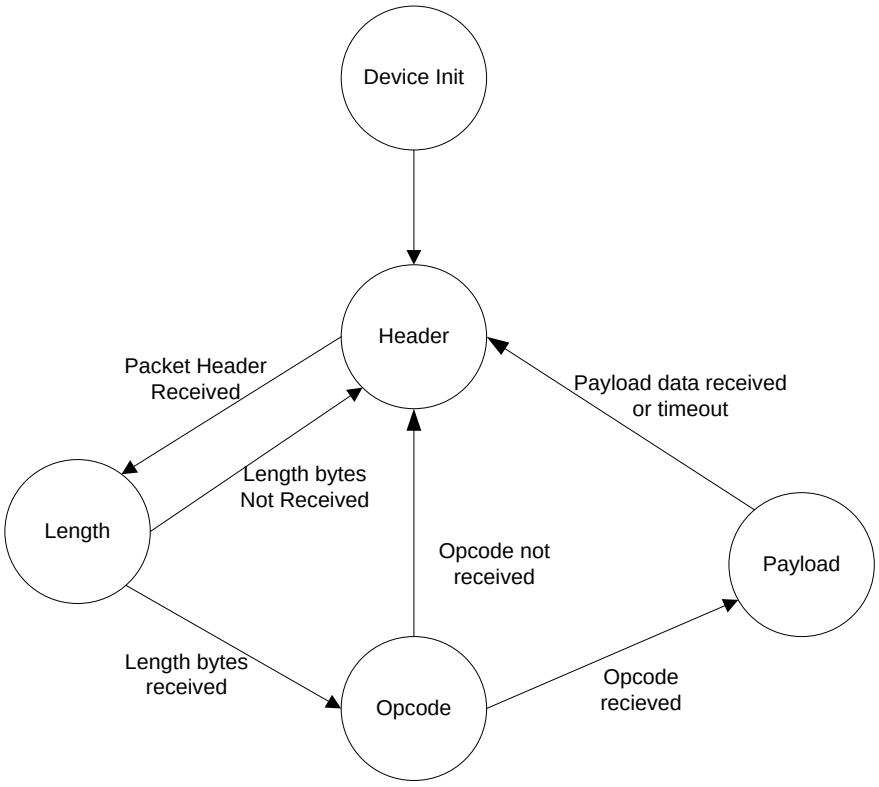
This subsystem decodes the data packets and sends them to the respective USB interface, as shown in Figure 5-87. The UART data is processed only if it follows the defined packet format.

Figure 5-87. UART-to-USB Subsystem Data Flow Chart



The state machine shown in Figure 5-88 decodes the packet format and extracts the data as described in the following sections.

Figure 5-88. UART Packet State Machine



Device Init. In the Device Init state, the following activities are performed:

- All global variables are initialized.
- The power LED is turned on.
- The ISR for the USB suspend command is started.
- The ISR for the UART Component is started.
- The USBFS Component is started and the status LED is switched on (after successful start and enumeration).
- The UART Component is started.
- PRoC BLE is reset.

Header State. In this state, the application logic processes the data received by the UART and transitions to the Length state on receipt of a valid header.

Length State. In this state, the application logic receives the length bytes and transitions to the Opcode state on receipt of a valid length; otherwise, it transitions to the Header state.

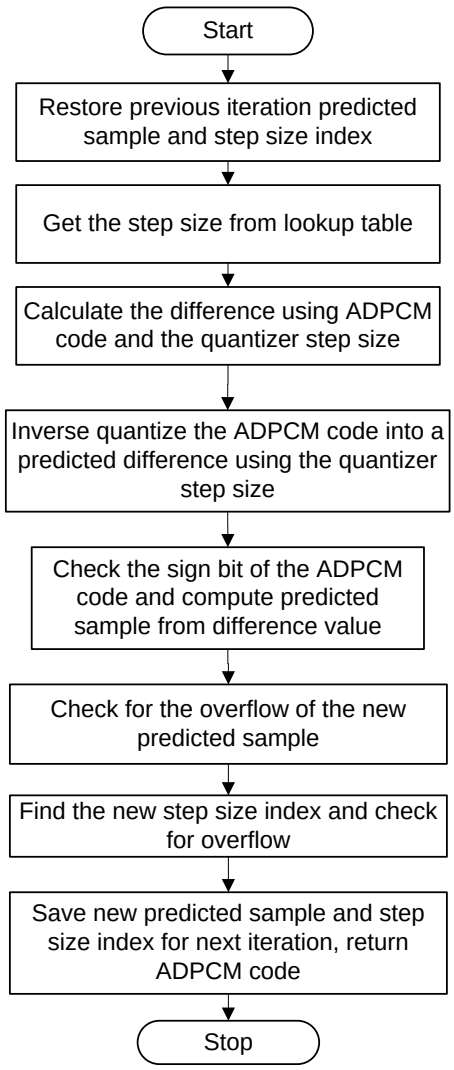
Opcode State. In this state, the application logic receives the opcode bytes and transitions to the Payload state on receipt of a valid opcode; otherwise it transitions to the Header state.

Payload State. In this state, the application logic receives the payload bytes, transitions to the Header state on receipt of a valid data payload, and starts processing the received packet. On receipt of an invalid payload, the application logic transitions to the Header state and repeats the entire cycle.

For audio packets, the payload data will go through the decompression module byte by byte, and decompressed data will be moved to the USB buffer.

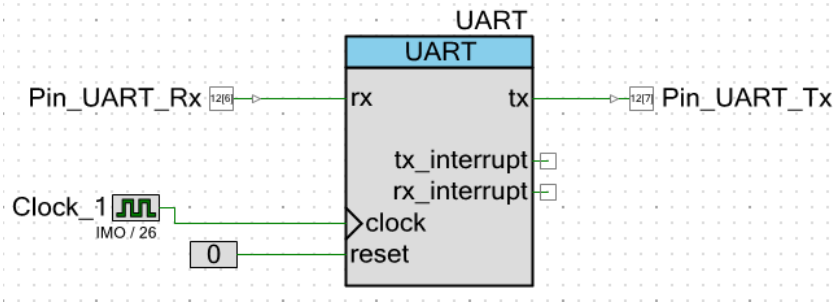
The ADPCM decoding follows the same method as described in the ADPCM Encoding section. The flow chart in Figure 5-89 explains the ADPCM decoding algorithm.

Figure 5-89. ADPCM Decoding Algorithm Flow Chart



Once all packet data is decompressed, it will be scheduled for USB audio endpoint transfer using the DMA channels. This subsystem is built using the UART Component (Figure 5-90) provided in PSoC Creator.

Figure 5-90. UART Schematic View



The Component is configured as shown in Figure 5-91 and Figure 5-92.

Figure 5-91. UART Component Configure Tab

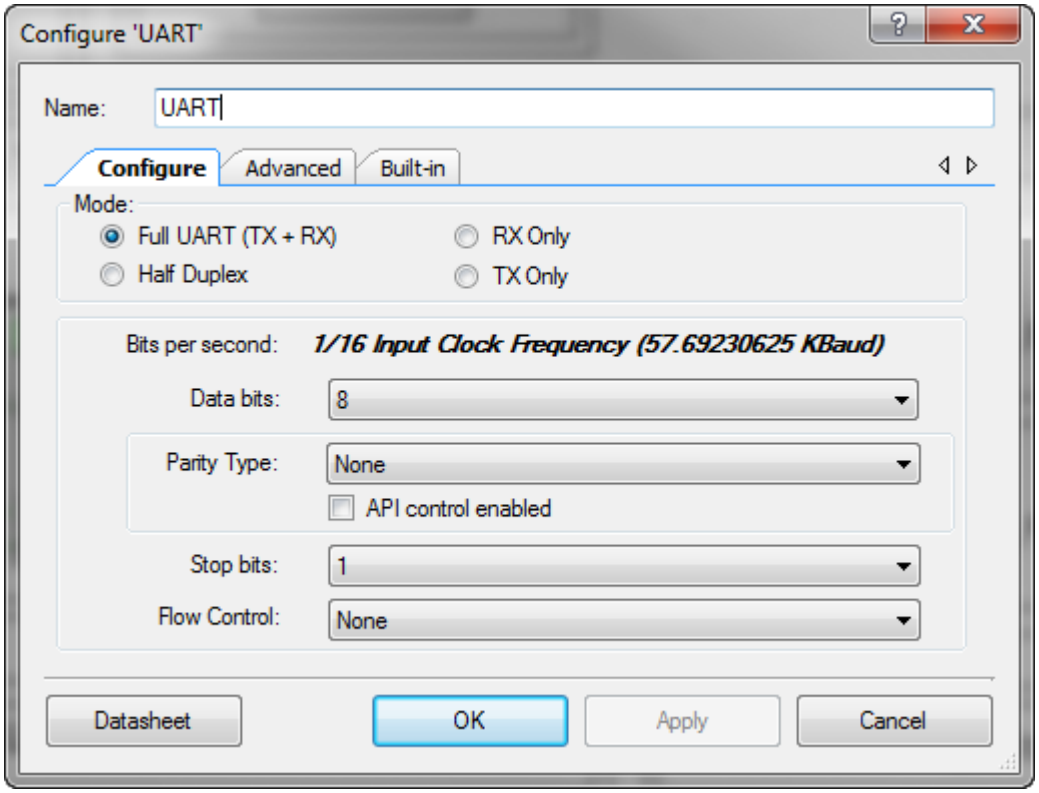
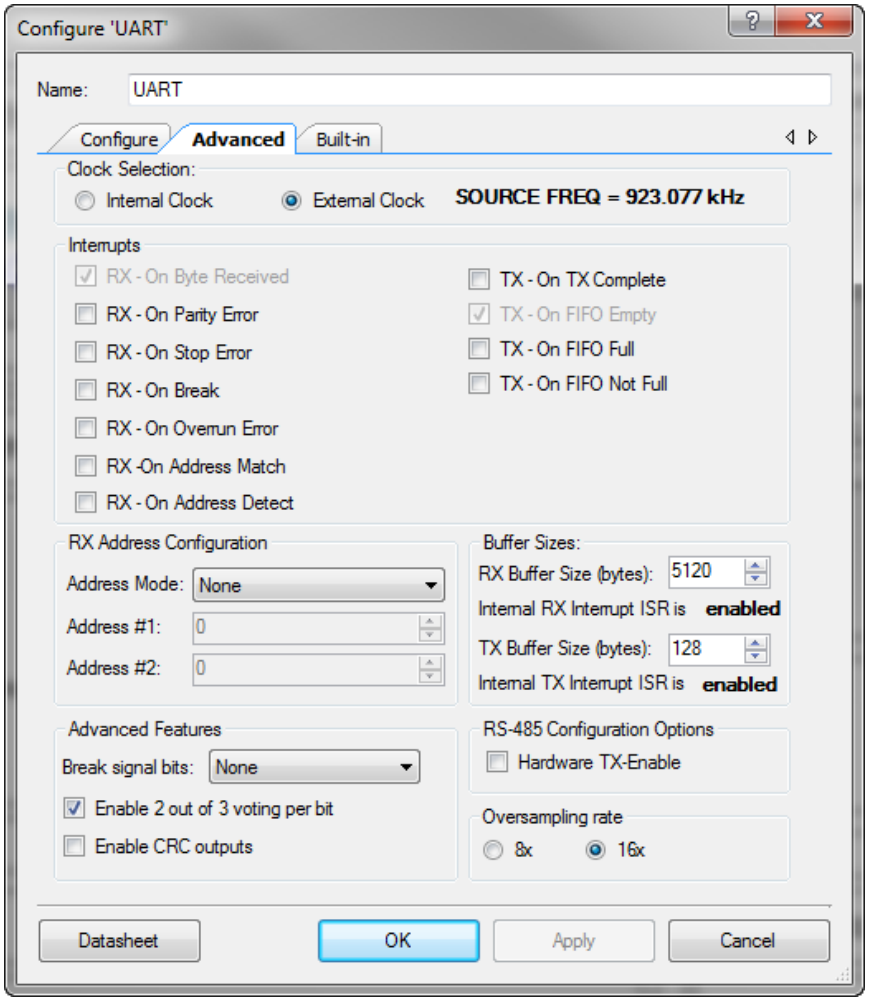


Figure 5-92. UART Component Advanced Tab



5.2.6 USB Subsystem

When connected to the host machine, the dongle enumerates as the following devices:

- HID keyboard device
- HID mouse device
- Audio input device
- CDC COM port (custom CySmart interface)

All the decoded packets will be loaded to the corresponding endpoint buffers, which are used by USB interfaces.

The USB subsystem is built on the USBFS Component provided in PSoC Creator (Figure 5-93). The USBFS Component is configured with standard descriptors to comply with the USB device class, as shown in Figure 5-94, Figure 5-95, and Figure 5-96.

Figure 5-93. USB Component of PSoC Creator

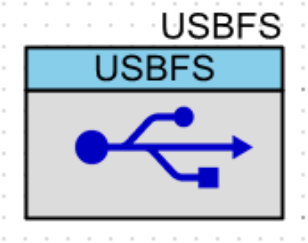


Figure 5-94. USBFS Component Device Configuration

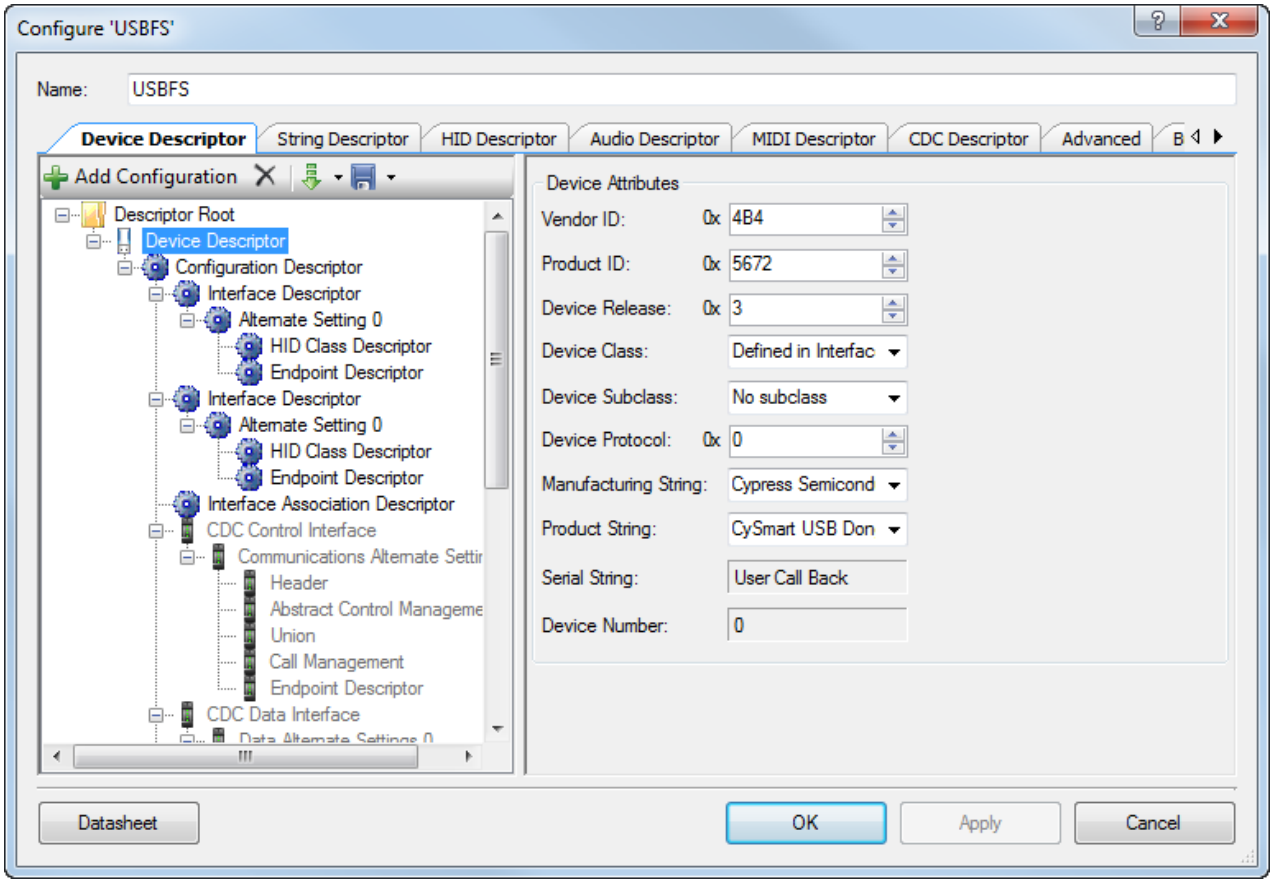


Figure 5-95. USBFS Component HID Descriptor Configuration

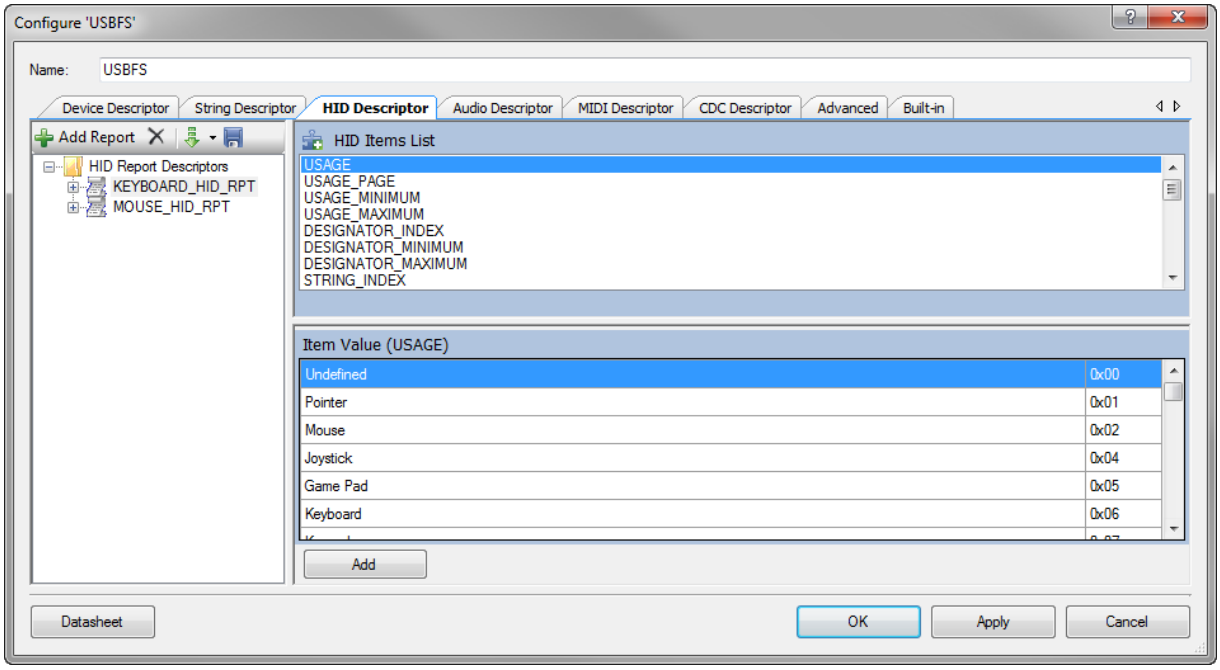
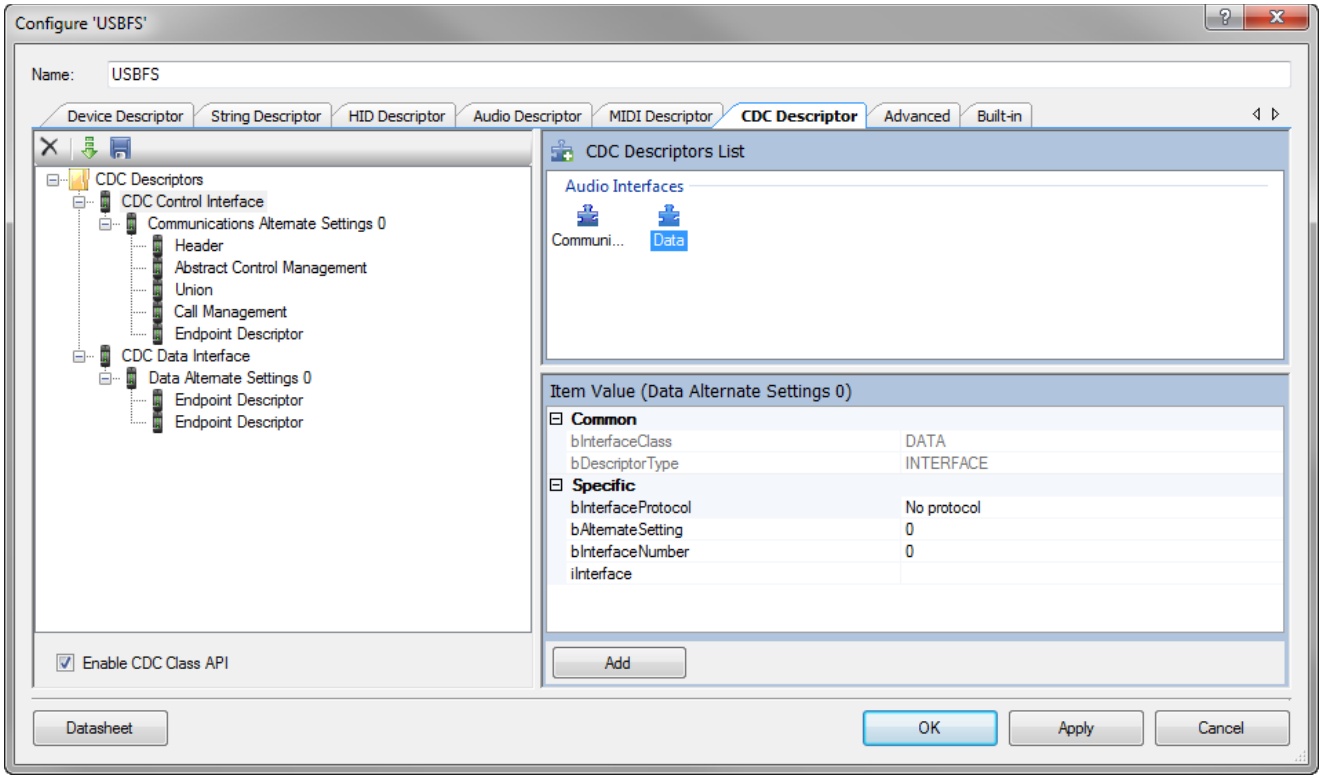


Figure 5-96. USBFS Component CDC Descriptor Configuration



6. Advanced Topics

6.1 Connecting PRoC the BLE Remote Control with a Bluetooth Smart Ready Device

The PRoC BLE remote control can be connected to any Bluetooth Smart Ready device provided that the device implements (or is backward compatible with) the Bluetooth 4.0 or Bluetooth 4.1 specification (that is, the device is Bluetooth Smart Ready) and the operating system on the device supports the HOGP. If your PC/operating system does not meet these criteria, follow the steps outlined in the Connecting the PRoC BLE Remote Control with the CySmart USB Dongle section.

Table 6-1 lists the Bluetooth Smart Ready devices and operating systems with which the CY5672 Remote Control RDK has been tested, along with the mouse features supported on each.

Table 6-1. Remote Control Features Supported on Devices and OS

Bluetooth Smart Ready Device	Operating System	Operating System Version	Remote Control Features Supported
Lenovo T440	Windows	8.0 and 8.1	All features* except the audio feature
OnePlus One Mobile Moto G	Android	4.4.2, 4.4.4 and 5.0.2	
MacBook Pro	OS X Yosemite	10.10.1	

* Horizontal scroll will work only with applications that support the USB HID AC Pan application control feature. In addition, the action triggered for Channel +/-, Power, Source and Return button presses depends on the operating system and the application running on the device.

The steps to connect the remote control with a Bluetooth Smart Ready device depend on the operating system running on the device.

6.1.1 Windows 8.1 PC

For a PC running Windows 8.1, follow these steps.

6.1.1.1 Connecting to the PC

1. Insert two AAA batteries (provided with the kit) into the battery holder on the remote control.
2. On the PC, navigate to **PC Settings** > **PC and devices > Bluetooth**. Turn ON Bluetooth
3. Press the pairing button on the remote control. The red LED is turned on to show that the remote control is now in advertising mode.
4. On the PC select the CY5672 Remote Control RDK device, and click **Pair**.
5. Place and move your finger along the trackpad surface. If the connection is successful, the mouse cursor on the PC will follow the finger movement. If a connection is not established within 30 seconds, the red LED is switched off.

6.1.1.2 Removing from the PC

To clear the CY5672 Remote Control RDK from Windows 8.1, follow these steps.

1. Navigate to **PC Settings** > **PC and devices > Bluetooth**. Turn on Bluetooth.

2. Locate “CY5672 Remote Control RDK” in the list of Bluetooth devices and click on it
3. Click the **Remove device** button.
4. A pop-up window appears with the message “Are you sure that you want to remove this device ?” Click **Yes** to remove the CY5672 Remote Control RDK from list of Bluetooth devices.
5. Observe that “CY5672 Remote Control RDK” no longer appears in the list of Bluetooth devices.

6.1.2 MacBook Pro

6.1.2.1 Connecting to a MacBook Pro

To connect the remote control with a Bluetooth Smart Ready MacBook Pro follow these steps.

1. Navigate to **System Preferences…** > **Bluetooth**. Click on the **Bluetooth** icon.
2. Plug the two AAA batteries provided with the kit into the battery holder on the remote control.
3. Press the connect button on remote control. The red LED is turned ON to show that the remote control is now in the advertising mode.
4. The “CY5672 Remote Control RDK” device should appear in the list of available Bluetooth devices. Click on “CY5672 Remote Control RDK” and then click the **Pair** button. A “Connected” message appears below “CY5672 Remote Control RDK.”
5. Wait while the CY5672 device is installed. Once the installation is complete, the “Connected” message below “CY5672 Remote Control RDK” disappears.

   Note: Installation time may vary based on the operating system, system configuration, and Internet connection.
6. Place and move your finger along the trackpad surface. If the connection is successful, the mouse cursor on the MacBook Pro will follow the movement of the finger. If a connection is not established within 30 seconds, the red LED turns OFF. In this case, repeat the sequence from step 4 to connect the remote control with the MacBook Pro.

6.1.2.2 Removing from a MacBook Pro

To clear the CY5672 Remote Control RDK from a MacBook Pro, follow these steps.

1. Navigate to **Terminal** and get into the root directory.
2. Change the directory to Library/Preferences using the command “cd Library/Preferences” and press the Enter key on the keyboard
3. Enter the command “ls com.apple.Bluetooth*” and press the Enter key on the keyboard.
4. Observe “com.apple.Bluetooth.plist” in the list.
5. Enter the command “sudo rm com.apple.Bluetooth.plist” and press the Enter key on the keyboard.
6. A request for password appears. Enter the system password and press the Enter key on the keyboard
7. Enter the command “ls com.apple.Bluetooth*” and press the Enter key on the keyboard.
8. Observe the message “No such file or directory.”
9. Now the Bluetooth list is cleared. Restart the MacBook Pro and observe that there are no Bluetooth devices in the list.

6.1.3 Android

6.1.3.1 Connecting to an Android Device

To connect the remote control with a Bluetooth Smart Ready Android device, follow these steps.

1. Navigate to **Settings** > **Bluetooth**. Turn on Bluetooth and click **SERACH FOR DEVICES** to start scanning for devices..
2. Plug the two AAA batteries provided with the kit into the battery holder on the remote control.
3. Press the connect button on the remote control. The red LED is turned ON to show that the remote control is now in the advertising mode.
4. “CY5672 Remote Control RDK” device should appear in the list of available Bluetooth devices. Click on “CY5672 Remote Control RDK” and observe the message “Pairing…” under “CY5672 Remote Control RDK.”

5. Wait while the CY5672 device is installed. The message “Connecting...” appears under “CY5672 Remote Control RDK.” Once the installation is complete, “CY5672 Remote Control RDK” appears in the list with a message below it stating “Connected.”

   Note: Installation time may vary based on the operating system, system configuration, and Internet connection.

6. Place and move your finger along the trackpad surface. If the connection is successful, the mouse cursor on the Android device will follow the movement of the finger. If a connection is not established within 30 seconds, the red LED turns OFF. In this case, repeat the sequence from step 4 to connect the remote control with the Android device.

6.1.3.2 Removing from an Android Device

To clear the CY5672 Remote Control RDK from Android, follows these steps.

1. Navigate to **Settings** > **Bluetooth**.
2. Locate “CY5672 Remote Control RDK” in the list of Bluetooth devices and click on the settings icon beside it.
3. Locate the **Unpair** button and click on it. Observe that “CY5672 Remote Control RDK” is removed from the list of Bluetooth devices.

6.2 Current Measurement

6.2.1 Measure System Current Consumption

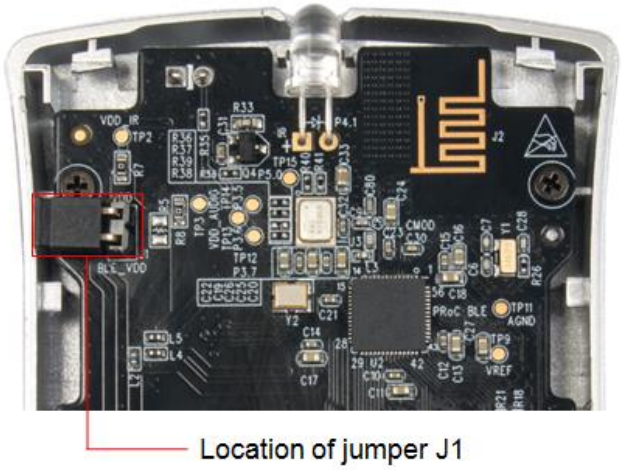
The system current can be measured by removing the batteries and powering up the remote control using an external power supply with an ammeter in series.

6.2.2 Measure PRoC BLE Current Consumption

Disassemble the remote control as described in the section Assembling and Disassembling the Remote Control to remove the bottom cover.

1. The current consumed by PRoC BLE can be measured at header J1 of the remote control PCBA, as shown in Figure 6-1.

Figure 6-1. Location of J1 Header and Jumper



2. Remove the jumper installed on header J1 and connect an ammeter on the header terminals
3. Put two AAA batteries in the battery compartment.
4. Measure the current consumed by PRoC BLE part across J1 as the remote control is operated.

   Note: The remote control can be operated without the bottom cover, as the remote control PCBA is mechanically integrated with the top cover via the screws. However, care must be taken to make sure that the AAA batteries do not fall from their positions.

5. When the current measurement is done, install the jumper at J1.

6. Assemble the remote control using the steps described in the Assembling and Disassembling the Remote Control section.

Battery life:

Table 6-2, Table 6-3, and Table 6-4 present the battery life for the remote control for different modes of operation. It also provides the current consumption measured in each state.

Table 6-2. Battery Life Calculation for Trackpad Mode of the Remote Control

State	Current Measured at Battery Terminals (mA) (System Current)	Usage Model (Time Used per Week in Hours)	Battery Capacity Used (mAh)
Active with data transfer	4.5	3.5	15.75
Active without data transfer	1.65	7	11.55
Idle	0.12	17.5	2.1
Low Power	0.085	140	11.9
Total battery capacity per week	**: 41.3 mAh**		
Battery life with 1250-mAh battery capacity	**: 30.27 weeks**		

Table 6-3. Battery Life Calculation for Motion Sensor Mode of the Remote Control

State	Current Measured at Battery Terminals (mA) (System Current)	Usage Model (Time Used per Week in Hours)	Battery Capacity Used (mAh)
Active with data transfer	7.3	3.5	25.55
Active without data transfer	4.6	7	32.2
Idle	0.09	17.5	1.575
Low Power	0.072	140	10.08
Total battery capacity per week	**: 69.405 mAh**		
Battery life with 1250-mAh battery capacity	**: 18.01 weeks**		

Table 6-4. Battery Life calculation for Audio Mode of the Remote Control

State	Current Measured at Battery Terminals (mA) (System Current)	Usage Model (Time Used per Week in Hours)	Battery Capacity Used (mAh)
Active with data transfer	13.9	3.5	48.65
Active without data transfer	1.6	7	11.2
Idle	0.12	17.5	2.1
Low Power	0.085	140	11.9
Total battery capacity per week	**: 73.85 mAh**		
Battery life with 1250-mAh battery capacity	**: 16.93 weeks**		

The following are the assumptions made while calculating the battery life:

- Usage model as described in the tables
- Current measurement made at constant 3 V
- Battery capacity assumed to be 1250 mAh

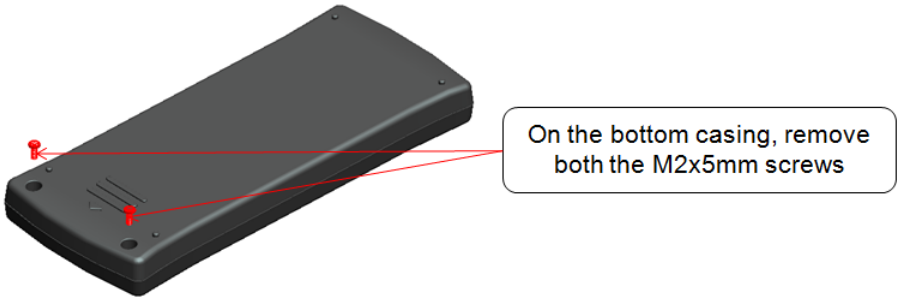
6.2.3 Measure Voltage

Test points for each power supply and ground are provided to measure the voltage. For details, refer to the Test Points section.

6.3 Assembling and Disassembling the Remote Control

6.3.1 Disassembly Procedure

1. Remove the screws from the bottom cover.

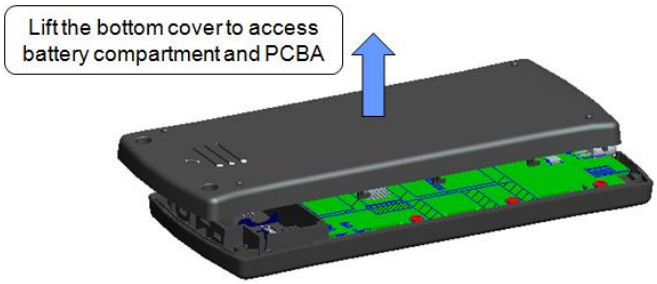


Note: The screws are not fastened to the remote control at the time of shipment from the factory. This step is required only if you have fastened the screws to the remote.

2. Slide the bottom cover out.



3. Lift the bottom cover



6.3.2 Assembly Procedure

Follow the disassembly procedure in the reverse order to assemble the remote control.

Appendix A. Schematics and FAQ

A.1 Schematics

A.1.1 Remote Control

Figure A-1. Remote Control Main Board Schematic – 1 of 4

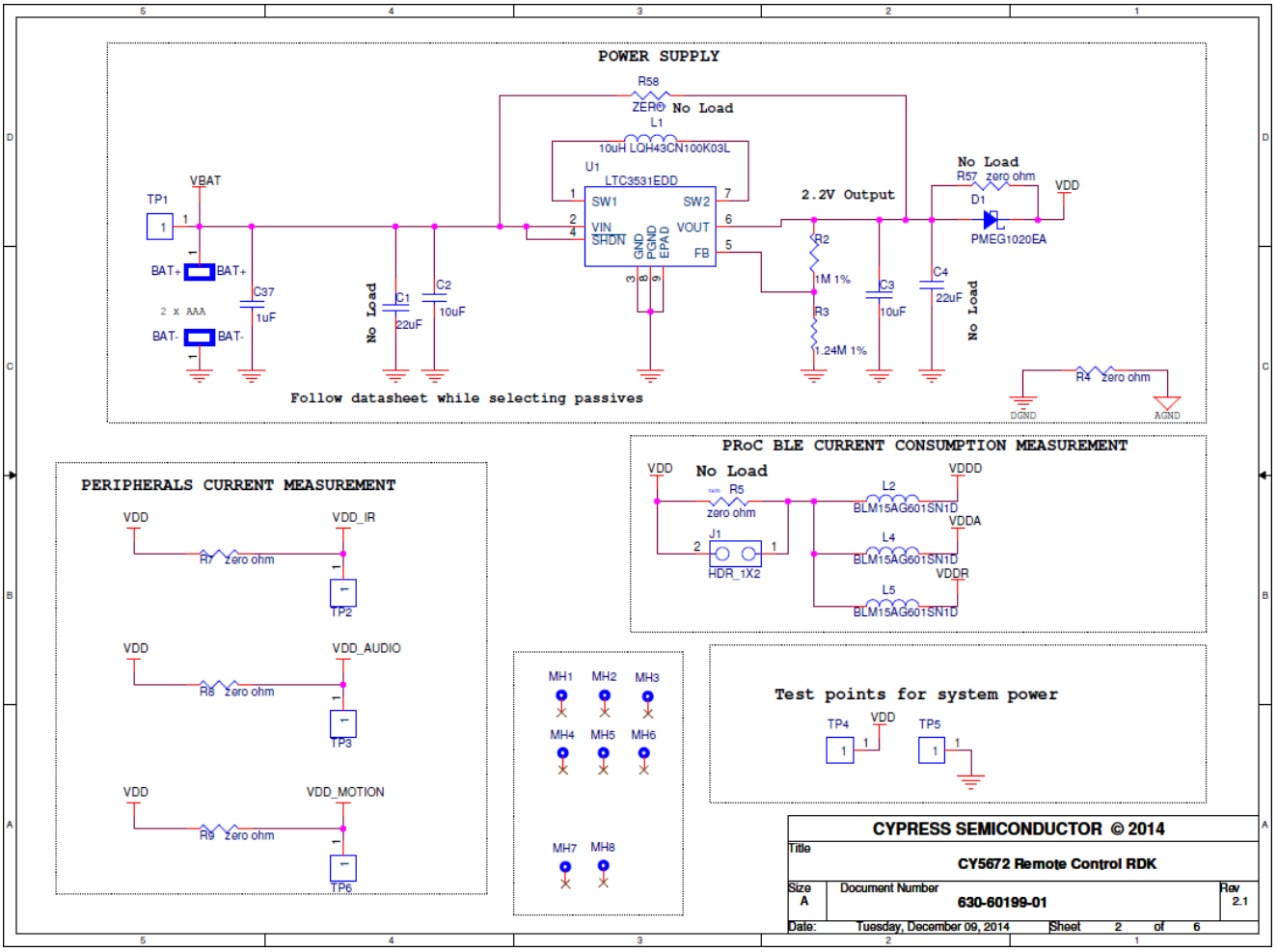


Figure A-2. Remote Control Main Board Schematic – 2 of 4

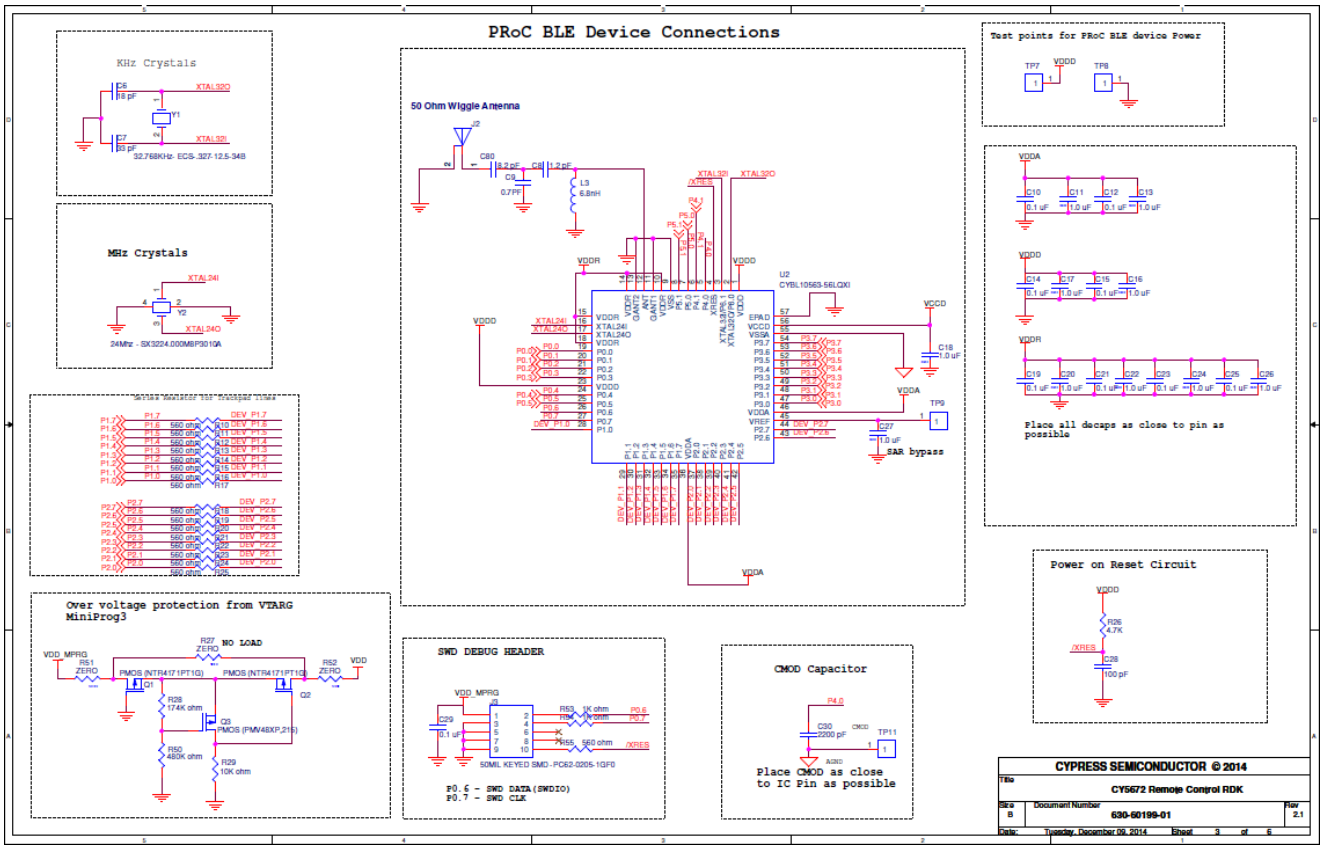


Figure A-3. Remote Control Main Board Schematic – 3 of 4

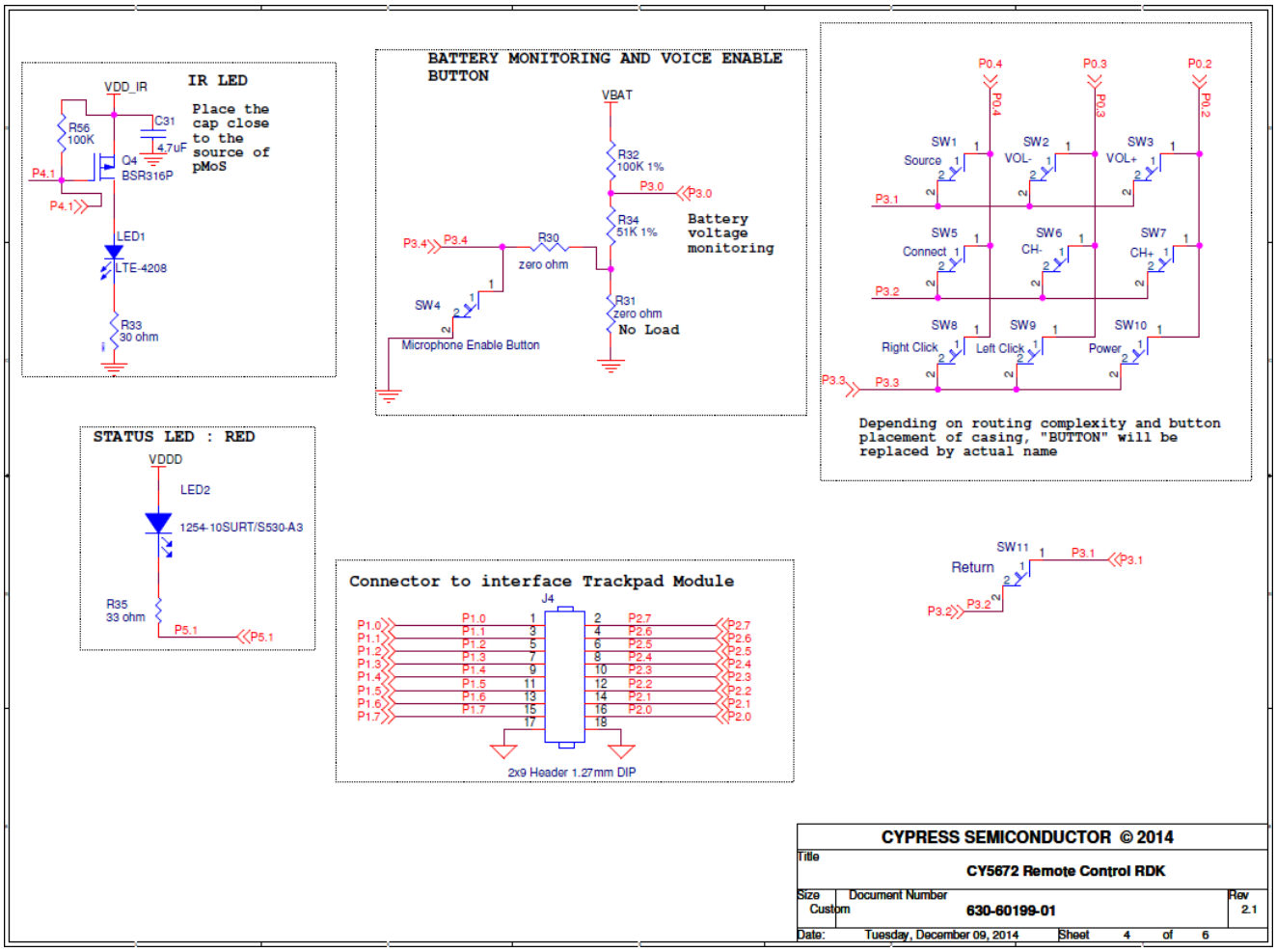


Figure A-4. Remote Control Main Board Schematic – 4 of 4

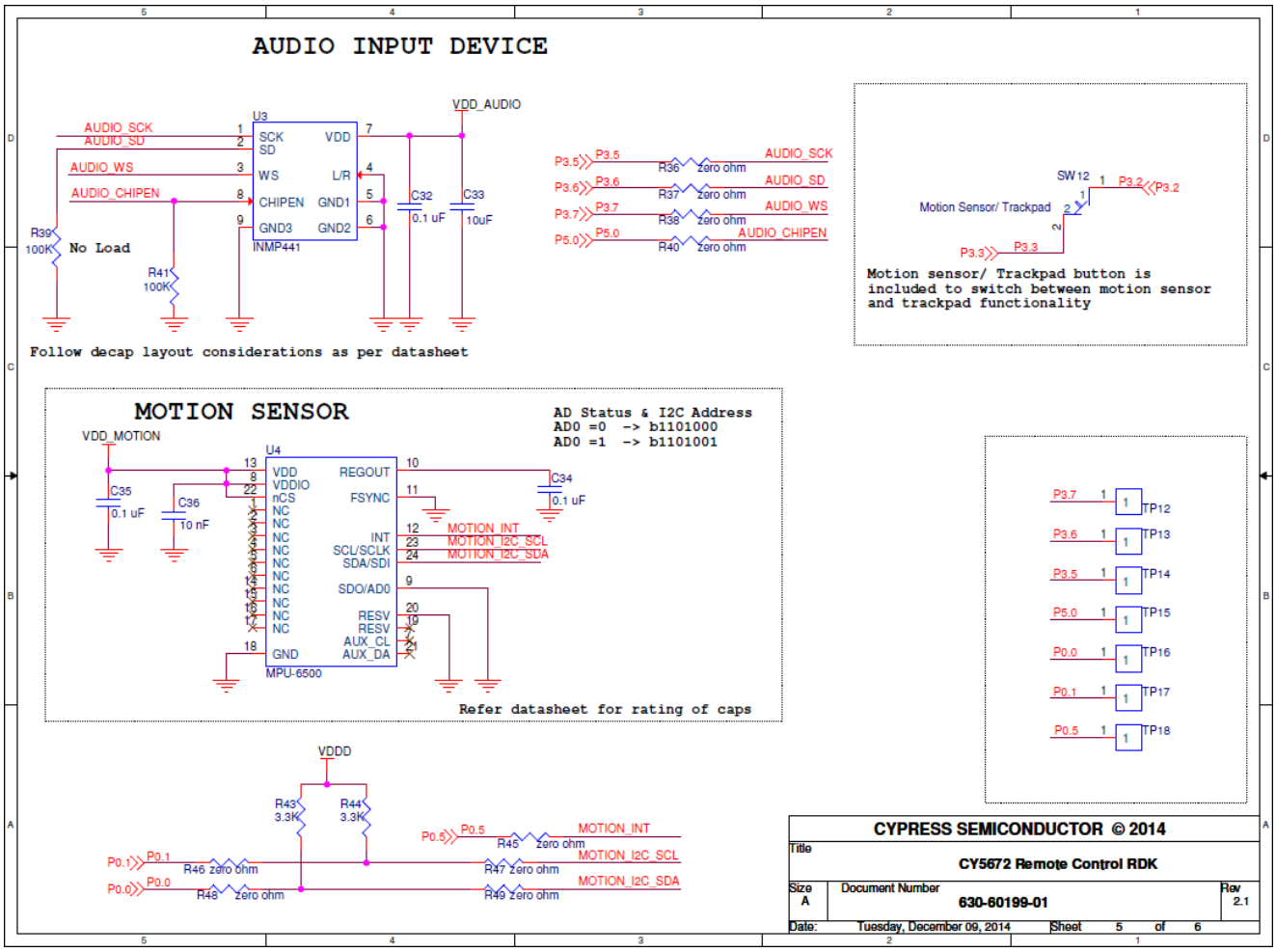
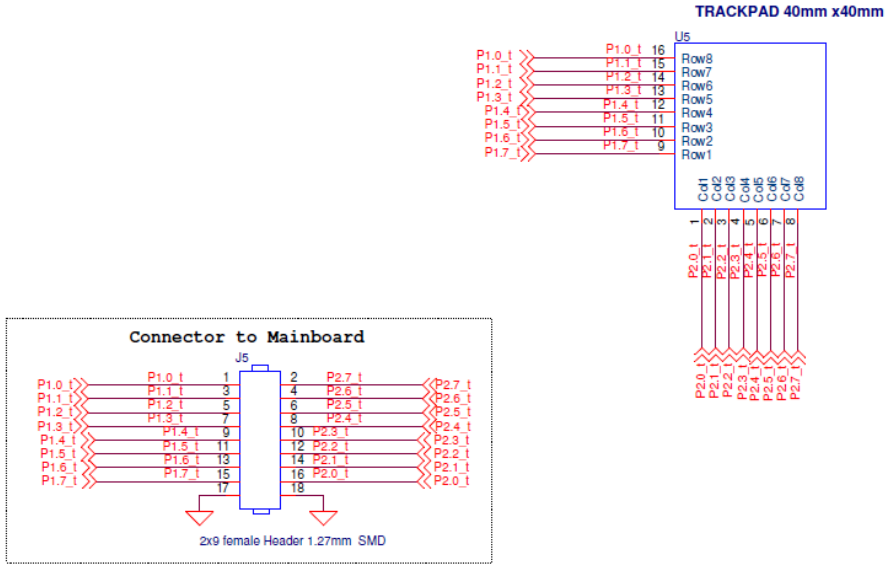


Figure A-5. Remote Control Trackpad Module Schematic



CYPRESS SEMICONDUCTOR © 2014			
Title: TRACKPAD			
Size: Custom	Document Number: –		Rev: 1.2
Date: Wednesday, September 24, 2014	Sheet 6 of 6		

A.1.2 Dongle

Figure A-6. Dongle Board Schematic – 1 of 2

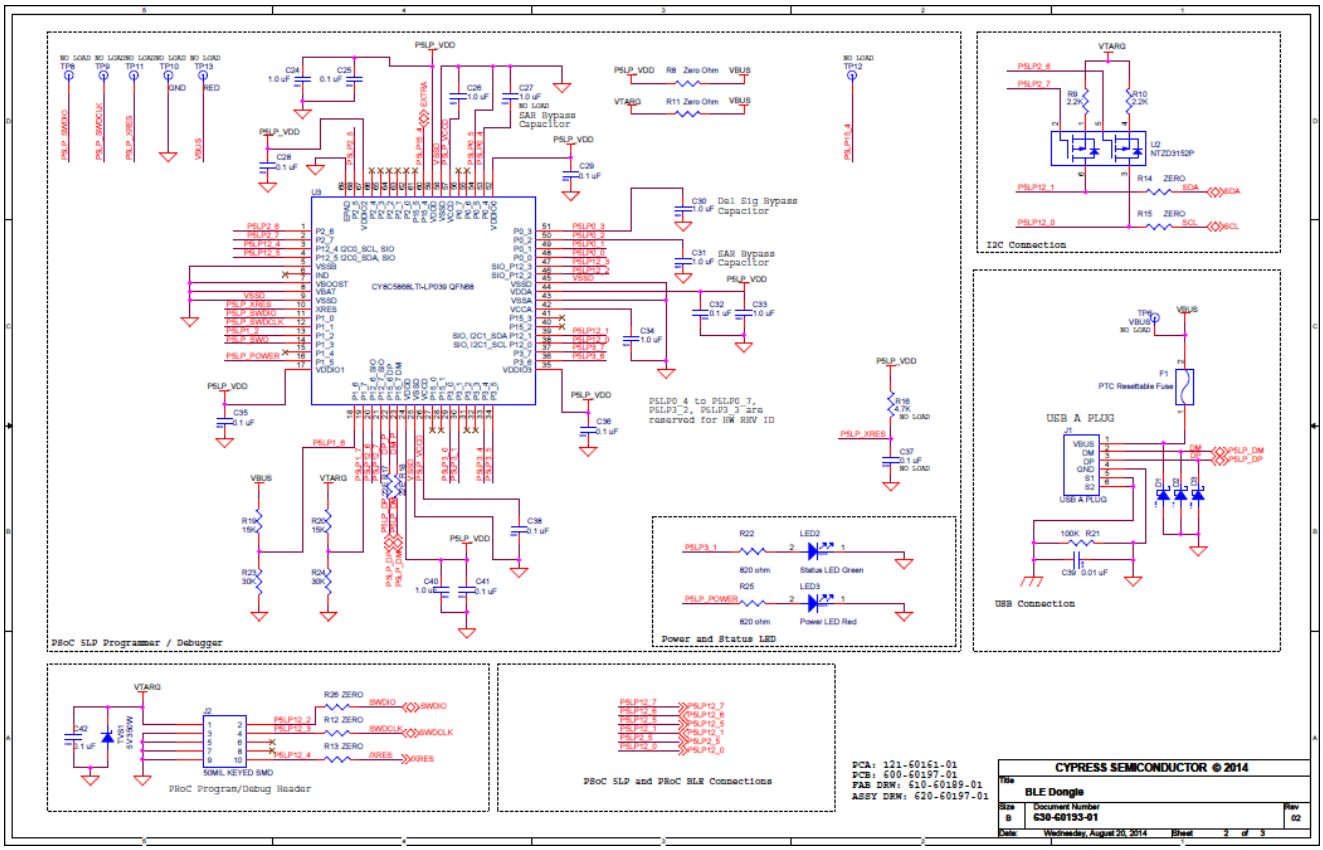
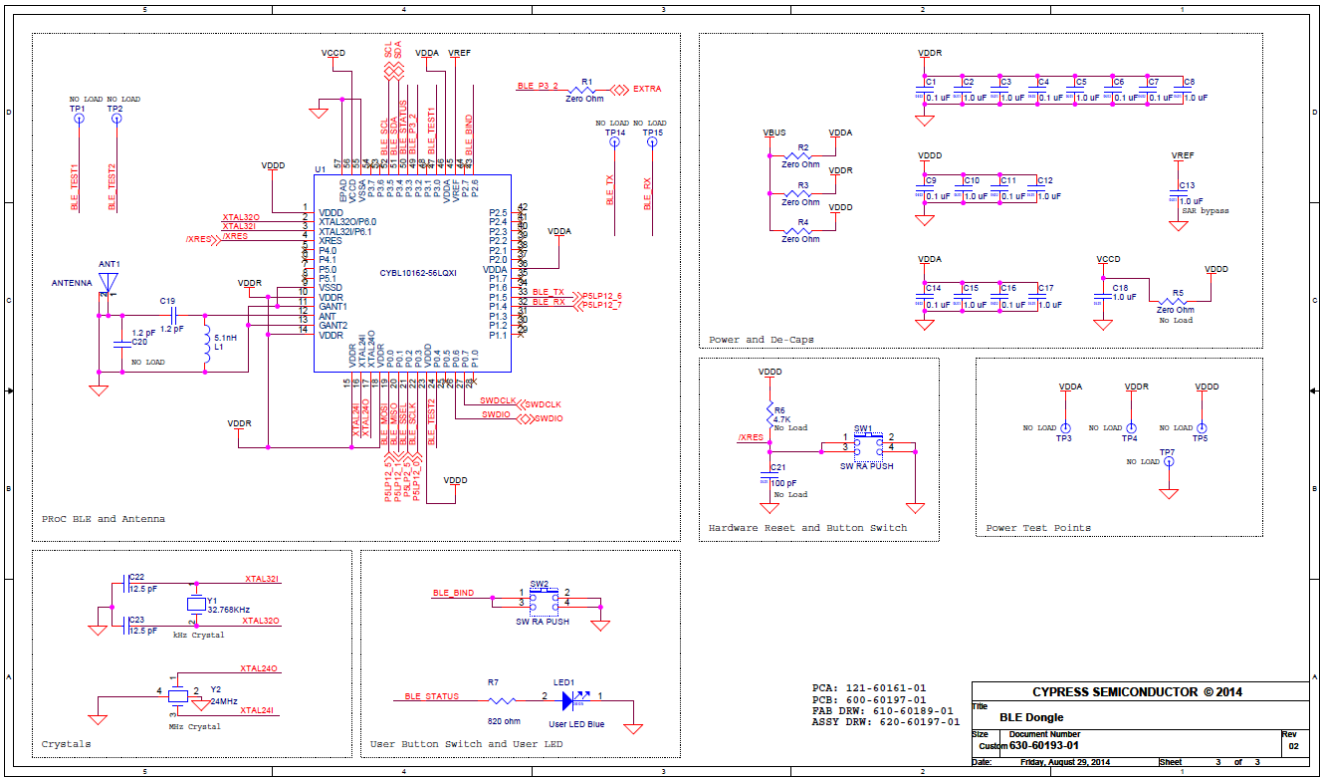


Figure A-7. Dongle Board Schematic – 2 of 2



A.2 FAQ

- **My remote control does not work with the CySmart USB dongle out of the box. How can I resolve this issue?**

  This may happen very rarely due to human error while packaging the kit. You can easily resolve this issue by following the steps in the Connecting the PRoC BLE Remote Control with the CySmart USB Dongle section.

- **Why does the Programming and Debugging PRoC BLE on the Remote Control and Dongle section recommend programming at 2.5 V?**

  PRoC BLE has an operating voltage range of 1.9 V to 5.5 V. Cypress recommends programming the remote control at 2.5 V when powered using batteries as the hardware is designed to operate at 2.2V. Note that the remote control incorporates an overvoltage protection circuit as described in the Power Supply section to prevent damage to the circuit due to programming at high voltages.

- **How do I open the remote control to insert batteries?**

  The section Assembling and Disassembling the Remote Control provides instructions to open the remote control. Once the remote control is opened, you will be able to see the two AAA battery holders.

- **What is the expected battery life of the remote control?**

  Battery life is dependent on the use model. Refer to Table 6-2 to get details about the battery life for a typical use-case..

- **When the remote is moved over or shaken very fast in a perfectly horizontal plane, the mouse cursor shows some movement along the Y-axis as well as well as movement along the X-axis. Ideally this action should only result in mouse cursor movement along the X-axis. Why is this happening?**

  This issue is due to the roll compensation feature that is enabled while initializing the Air Motion Library (in `AIR_MOTION_Init()`). There are many contributing factors to this accumulated error, however it is mainly caused by the choice not to calibrate accelerometers offsets, for more cost-efficient manufacturing. The implementation of Air Motion Library on ARM Cortex-M0 processor core currently limits the improvement of performance and resolution of this behavior. For more details and queries on Air Motion Library, the alternative host-side solution or calibration-based solution, please contact Invensense Inc. (sales@invensense.com). Note that this issue is not observed during normal usage and is only noticeable when moving the remote in a perfectly horizontal plane.

- **Do's and Don'ts**
  - Avoid applying a voltage of more than 2.5 V to the test points provided on the remote control. This may cause damage to the circuit due to overvoltage or reverse voltage.
  - Avoid spilling any kind of liquid on the remote control or the CySmart USB dongle. These devices are not waterproof and may get damaged.
  - Do not press the Motion Sensor or Return button together with any other key on the remote control. This may trigger an unpredictable action on the host device. This is caused due to the keypad design where buttons are arranged in matrix and direct configuration to optimize the GPIO usage.
  - The trackpad has designated areas to trigger vertical scroll (right-side edge of the trackpad) and horizontal scroll (bottom edge of the trackpad). Any finger movement initiated on these areas shall be processed with respect to scroll only as long as the finger stays within the designated area. Therefore, do not initiate other actions such as mouse pointer movement or gestures in the vertical/horizontal scroll area.

- **What does the mouse, keyboard, multimedia, power, audio control and audio report structure look like for the CY5672 Remote Control?**

  The mouse report is used to send codes related to trackpad use. The report ID is set to one. The zeroth byte is used to send left-click and right-click. The first and second bytes are used to send the delta X and delta Y respectively. The third byte is used to send vertical scroll data. Finally, the fourth byte is used to send horizontal scroll data. The left button, right button, pointer movement from the trackpad and motion sensor, and region-based scroll use this report.

```
typedef struct _Mouse_Report_
{
    uint8 click;                        /* Click button status
                                         * 0th bit is used for left-click,
                                         * 1st bit is used for right-click, and
                                         * 2nd bit is used for middle-click.
                                         * 3rd and 4th bits are used for user-
                                         * defined click. */
```

```
    int8 x;                             /* Delta x movement */
    int8 y;                             /* Delta y movement */
    int8 zwheel;                        /* Delta vertical movement */
    int8 hwheel;                        /* Delta horizontal movement */
}Mouse_Report;
```

The keyboard report is used to send codes related to keyboard/keypad use. The report ID is set to two. The zeroth byte is a modifier key that is used to send the Ctrl, Windows, Alt, and Shift keys. The first byte is used to learn the length of the key code. And the last six bytes are used to send keyboard keys. The top-to-bottom gesture and zoom gesture on the trackpad use this report.

```
typedef struct _Keyboard_Report_
{
    uint8 mkey;                         /* Modifier key */
    uint8 keylength_used;               /* Number of keys used in the key code*/
    uint8 keycode[NUMBER_OF_KEYCODES];  /* Key codes */
}Keyboard_Report;
```

The multimedia report is used to send the consumer control usage page. The report ID is set to three. The first byte is used to align the multimedia key code to 16 bits. The multimedia_key_code is used to send consumer control codes. The Volume Up, Volume Down, Channel Up, Channel Down, and source and power buttons use this report.

```
typedef struct _Multimedia_Report_
{
    uint16 multimedia_key_code;         /* Multimedia key code */
}Multimedia_Report;
```

The power report is used to send system control codes. The report ID is set to four. The second byte is used to send the system control codes to the peer device. This report is not currently used in the remote control.

```
typedef struct _Power_Report_
{
    uint8 power_key_code;               /* power key code */
}Power_Report;
```

The audio report is used to send the audio data using the HID vendor page. The report ID is set to 30. Multiple audio packets will be sent over the connection interval. The audio report collects all such packets. The audio_data variable in the Audio_Report structure is a pointer for collecting the audio data packet. noOfAudioPacket has the count of the audio packet.

```
typedef struct _Audio_Report_
{
    Audio_Data *audio_data[AUDIO_BUFFER]; /* Pointer for collecting the audio
packet */
    uint8 noOfAudioPacket;  /* Number of the audio packets to be sent */
}Audio_Report;
```

The Audio_Data is the structure that is used for collecting the audio data. The audio_buffer variable is used to send audio data. The audio_length variable indicates the number of bytes in the audio_data variable.

```
/* Structure for sending audio report */
typedef struct _Audio_Data_
{
    uint8 audio_length;                     /* Length of the Audio packet */
    uint8 audio_buffer[AUDIO_BUFFER_LENGTH]; /* Audio data buffer */
}Audio_Data;
```

The Audio control structure is used to send the audio control information using the HID vendor page. The report ID is set to 31. The first byte is the control code. There can be two types of control codes that is used in the remote firmware: 0xFF and 0xFE. 0xFF is used for sending the status of the voice button. 0xFE is used to send the sync packet. The controlData is used to send the control information. The controlDataLength parameter is used to determine the length of the control information. The controlBufferPoint parametger is used to determine the audio packet number after which this information needs to be sent. This is to maintain the sync between the two devices.

```
typedef struct _Audio_Control_Packet_
{
    uint8 controlCode;  /* Control data ID */
    uint8 controlData[CONTROL_DATA_LENGTH]; /* Control information */
    uint8 controlBufferPoint; /* Position of the audio packet after which the
control information need to be sent */
    uint8 controlDataLength; /* Length of the control information */
}Audio_Control_Packet;
```

A.3 Troubleshooting and FAQ for Interoperability Issues with Bluetooth Smart Ready Devices

- **After a disconnection because the device was out-of-range, if the remote control is brought back in-range, it does not connect back to MacBook Pro.**
  In order to connect back to a MacBook Pro, the remote control does undirected advertisement with whitelist filtering that needs to be recognized and responded to by the MacBook Pro computer. It is observed that MacOS (10.10.1) on MacBook Pro sometimes does not respond to the advertisement in this scenario. If the remote control is unable to connect back automatically, on your MacBook Pro, turn Bluetooth OFF and then turn it back ON. Now, move your finger on the remote control's trackpad if the remote was in the trackpad mode, or move the remote if it was in the motion sensor mode. It should connect back automatically.

- **Buttons mapped to the HID consumer controls usage page do not work with Samsung Galaxy NotePRO and Note3 running Android 4.4.2**
  It is observed that the Android 4.4.2 operating system running on Samsung Galaxy NotePRO and Note 3 do not enable the CCD values for Multimedia_Report characteristics. It is recommended to upgrade the devices to Android 4.4.4 or 5.0 and retest.

- **After a disconnection because the device was out of range, if the remote control is brought back in-range it connects back but does not work with Samsung Galaxy Pro Tab XM-P9600 running Android 4.4.2**
  It is observed that the Samsung Galaxy Pro Tab XM-P9600 running Android 4.4.2 does not recognize the notifications sent by the remote control in this scenario. If this behavior is seen, turn Bluetooth OFF on the tab and then turn it back ON. Now, move your finger on the remote control's trackpad if the remote was in the trackpad mode or move the remote if it was in the motion sensor mode. The remote control should now work. To avoid this issue, upgrade the tab to Android 4.4.4 or Android 5.0.

- **Zoom In/Out does not work with my MacOS- or Android-based devices**
  The remote control firmware implementation sends a 'Ctrl + zwheel' keycode when a Zoom In/Out gesture is triggered. The action triggered on the host device is dependent on the operating system and the application running in the foreground. It is possible to customize the remote control firmware implementation to change the key codes for this gesture. Refer to the FAQ section to get more details about the keyboard report structure that is used to send the keycode for Zoom In/Out gesture.

- **When the audio data received from the remote control is recorded on a MacBook Pro running MacOS 10.10.1, the volume of the recorded audio is found to be low. Why?**
  This is due to a limitation of the microphone (INMP441) used on the remote control which provides 24-bit audio data over I2S, whereas the voice-over-BLE firmware implementation (see Audio Subsystem) expects 16-bit audio samples. This issue can be avoided by using a digital microphone which provides 16-bit audio samples over I2S. Note that voice recognition, which is the primary use case for the voice-over-BLE feature, is not affected by this issue and works fine on the MacBook Pro.

- **When the remote is continuously used with a Samsung Galaxy Pro Tab XM-P9600 running Android 4.4.2 operating system, the Bluetooth service on the tab stops working. Why is this happening?**
  This issue is typically observed after an hour of continuous usage on the Samsung Galaxy Pro Tab XM-P9600 running Android 4.4.2 operating system. If the issue is encountered while using the remote control with the Samsung Galaxy Pro Tab XM-P9600, restart the tab and move your finger on the remote control's trackpad if the remote was in the trackpad mode or move the remote control if it was in the motion sensor mode; The Bluetooth service on the tab should work and the remote control should connect back automatically.

Revision History

Document Revision History

Document Title: CY5672 PRoC™ BLE Remote Control Reference Design Kit Guide Document Number: 001-97071			
Revision	**Issue Date**	**Origin of Change**	**Description of Change**
**	04/24/2015	SELV	First version of this kit guide
*A	05/12/2017	GNKK	Updated the Cypress logo and copyright information.