CY3668

WirelessUSB™-NL Development Kit Guide

Doc. # 001-76173 Rev. *D

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709
Phone (USA): 800.858.1810
Phone (Intnl): 408.943.2600
http://www.cypress.com

Copyrights

Contents

1. Introduction

The CY3668 Development Kit (DVK) User Guide allows you to develop wireless applications in a 2.4-GHz ISM band using Cypress's WirelessUSB™-NL product.

1.1 Kit Contents

The CY3668 DVK includes:

- Two CY3668 bridge/keyboard development boards with integrated LCD panels
- Two WirelessUSB-NL modules
- Two CY3668-enCoRe V modules
- One CY3668-enCoRe II module
- Two 3.3-V LCDs
- Two power adaptors
- One PSoC® MiniProg
- One USB A to Mini-B cable
- Ten connecting wires
- Quick Start Guide
- Installation CD

1.2 PSoC Designer™ 5.3

Cypress's PSoC Designer is an easy-to-use software development integrated design environment (IDE) that introduces a hardware and software co-design environment based on schematic entry and embedded design methodology.

With PSoC Designer, you can:

- Automatically place and route select components and integrate simple glue logic normally residing in discrete muxes.
- Trade off hardware and software design considerations allowing you to focus on what matters and getting to market faster.

1.3 Additional Learning Resources

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

1.4 Documentation Conventions

Table 1. Documentation Conventions for User 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
No text, gray table cell	Represents a reserved bit in register tables.

1.5 Document Revision History

Revision	PDF Creation Date	Origin of Change	Description of Change
**	02/23/2012	CSAI	Initial version of kit guide
*A	10/05/2012	SELV	Updated PSoC Designer version from 5.1 to 5.3 Modified sections Default Jumper Settings on page 9, Programming Steps on page 10, and WUSB-NL Bridge Example on page 51 Organized the Installation chapter into Getting Started on page 7 and Kit Operation on page 9 Added Hardware on page 13 and Troubleshooting on page 55
*B	12/03/2012	SELV	Updated directory path across the document.
*C	12/07/2012	SELV	Updated Programming Steps on page 10 Updated all procedures in chapter Example Projects on page 35 Added Sections Setting up WirelessUSB Bridge on page 50 and Exercising WirelessUSB Mouse Example Project on page 53.
*D	01/25/2013	SELV	Updated Example Projects on page 35. Updated Issue 4 in Troubleshooting on page 55. Removed sample firmware code.

2. Getting Started

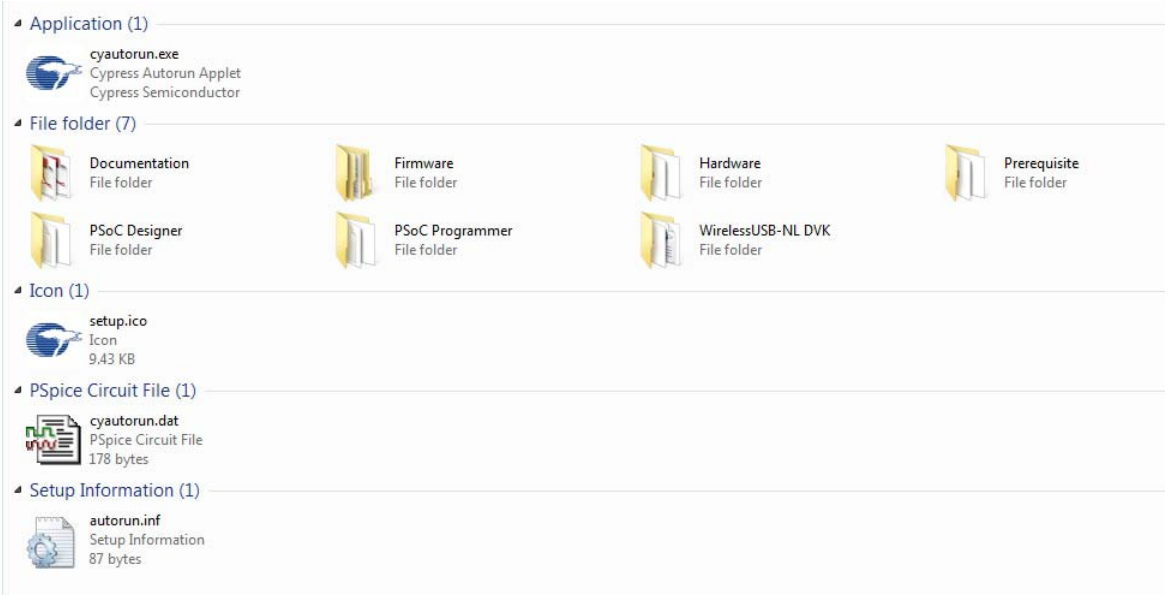
This chapter describes the procedure to install the required tools and the software required to program WirelessUSB-NL DVK.

2.1 Install Software

Follow these steps to install the required tools and software:

1. Insert the kit CD into the CD/DVD drive of your PC. The CD is designed to automatically invoke the Kit Installer and the following installer screen appears.

Note If the installer is not automatically invoked, double-click *cyautorun.exe* in the root directory of the CD as shown in the following screenshot.



2. Click 'Install WirelessUSB-NL DVK …' on the Installer screen and go through the installation procedure. This installs the following tools and software on your PC:
   a. PSoC Designer
   b. PSoC Programmer
   c. WirelessUSB-NL DVK software
3. This completes the software installation process.

3. Kit Operation

3.1 Install Hardware

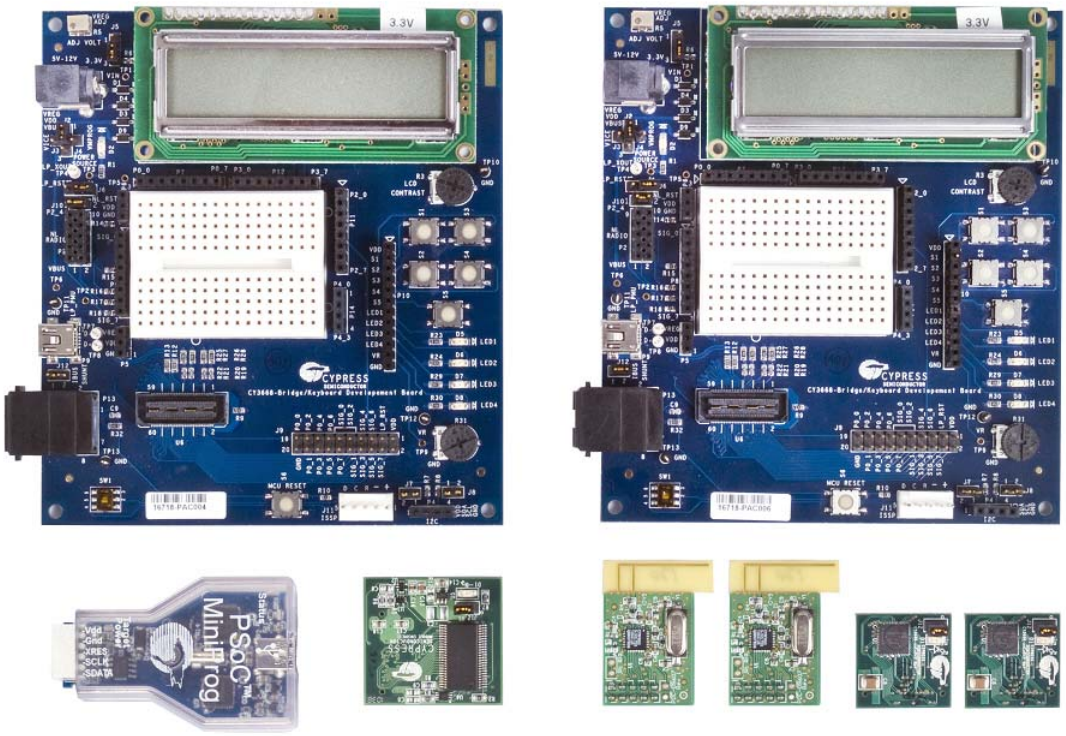
WARNING: Static discharges from the human body can easily reach 20,000 volts. This can damage the components on the DVK. Make sure that any static is discharged before touching the hardware. Ensure the DVK is powered off before making any changes to the connections.

3.1.1 Default Jumper Settings

The following are the default jumper settings for both the CY3668 boards:

- J10 - place a jumper
- J2 - place a jumper between VREG and VDD
- J5 - place a jumper between pin 2 and 3.3 V
- J12 - place a jumper
- J6 - place a jumper between pins 2 and 3
- Leave other jumpers open

Figure 3-1. CY3668 Boards



3.2 Programming and Emulation Procedures

Note You can program the kit using either the PSoC MiniProg or the ICE-Cube. However, the ICE-Cube is required for emulation. The ICE-Cube is available in the PSoC Development Kit and can be purchased from the Cypress's online store (PSoC Development Kit, CY3215-DK).

3.2.1 Programming Steps

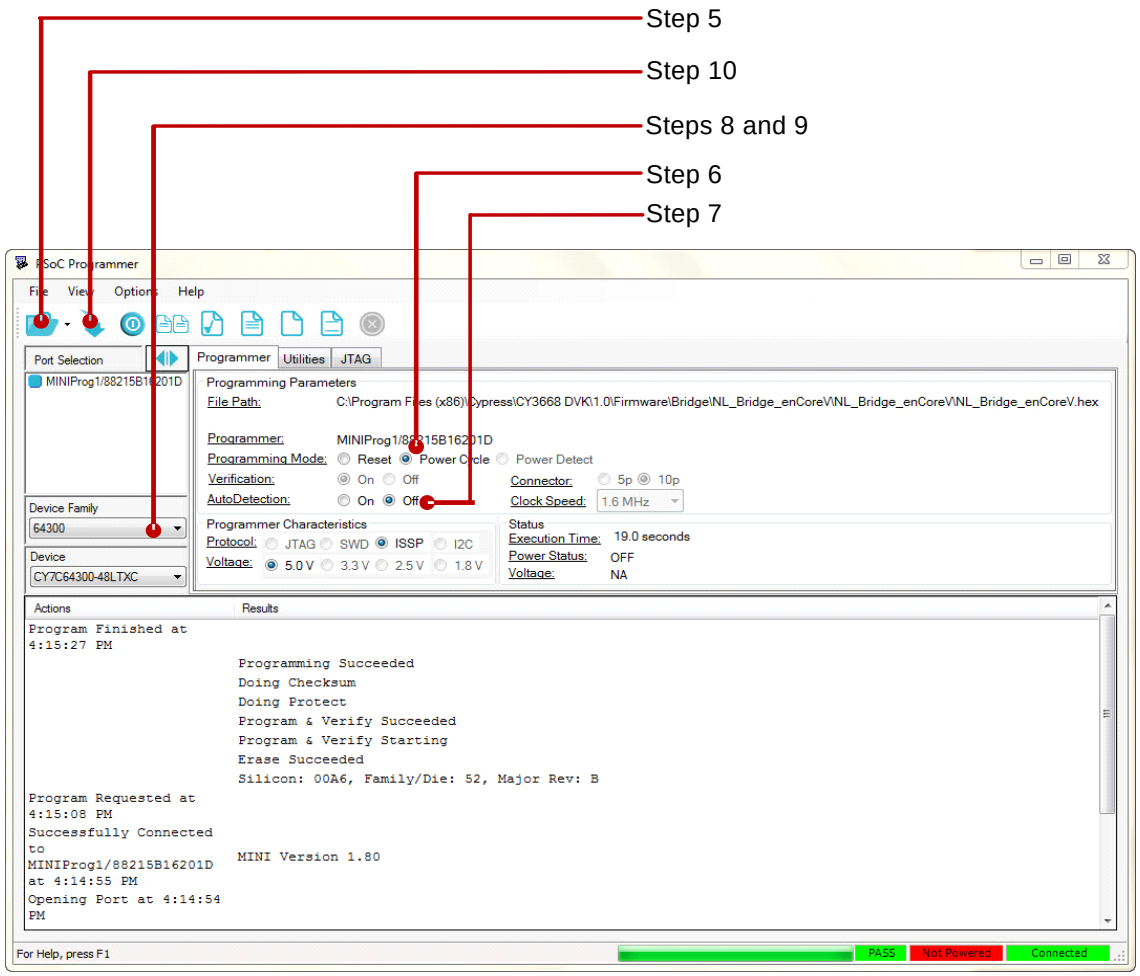
Important

- Disconnect power supply from the 12-V power adapter.
- Remove the WUSB-NL module from the development board before programming the firmware.
- For the enCoRe V module, position the SW1 switch slider away from the LCD panel (see enCoRe II/enCoRe V Programming Selector on page 19); for the enCoRe II module, position the SW1 switch slider towards the LCD panel.

Follow these steps to program the kit:

1. Connect one end of the USB cable to the PSoC MiniProg and the other end to the computer.
2. Connect the MiniProg to the 5-pin ISSP header on the CY3668 DVK board.
3. Launch PSoC Programmer version 3.16 or later.
4. If the **Firmware Update required at ...** dialog box is displayed, upgrade the MiniProg's firmware by clicking **Utilities > Upgrade Firmware** before proceeding with the remaining steps.

Figure 3-2. PSoC Programmer

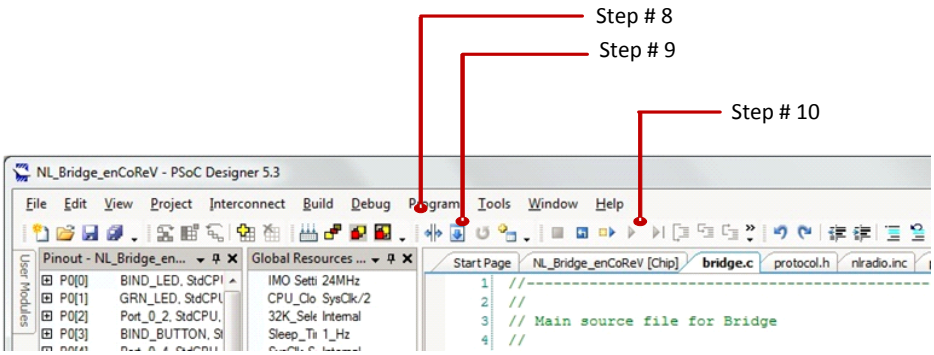


5. Click on **File Load** to load the desired binary (hex) file.
6. For **Programming Mode**, select **Power Cycle**.
7. Turn off **AutoDetection.**
8. Select the following part from the **Device Family** list based on the MCU module:
   - **64300** for enCoRe V Module
   - **63900** for enCoRe II Module
9. Select the following part from **Device** list based on the MCU module:
   - CY7C64300-48LTXC for enCoRe V Module
   - CY7C63923-PVXC for enCoRe II Module
10. Click **Program**. PSoC Programmer goes through programming and verification steps. Results are displayed on the bottom half of the screen.
11. When programming is complete, remove the MiniProg from the CY3668 DVK board.

3.2.2 Emulation Steps

1. Launch the desired .APP project file project from `<Install_Directory>\Cypress\CY3668 DVK\1.0\Firmware\<example name>\[PSoC Project name]\`.
2. Connect one end of the blue CAT 5e cable to the ICE-Cube.
3. Connect the other end of the CAT 5e cable to the CY3668 DVK board.
4. Select **Build > Generate Configuration Files for '<Project name>' Project**.
5. Select **Build > Build '<Project name>' Project** or press **[F7]**.
6. Configure the debug port setting. Select **Project > Settings > Debugger**, then select ICE-Cube.
7. Select the **Pod Power Source** and **Pod Supply Voltage**.
   a. If **External only** is selected, retain the default jumper setting.
   b. If **ICE may power pod** is selected, choose **3.3 V** as **Pod Supply Voltage** and place a jumper between VDD and VICE at J3.

Figure 3-3. PSoC Designer Toolbar



8. Select **Debug > Connect**, or select the **Connect** icon.
9. Select **Debug > Download to Emulator**, or select the **Download to Emulator** icon.
10. Select **Debug > Go**, press **[F5]**, or select the **Go** icon.

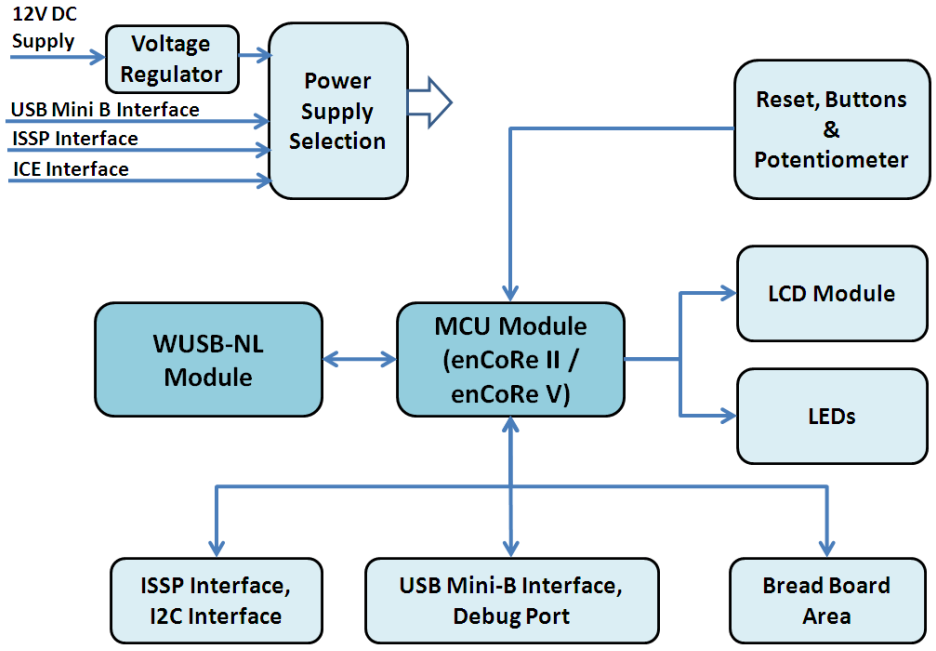
4. Hardware

4.1 System Block Diagram

The CY3668 DVK consists of the following blocks:

- Voltage Regulator and Power Supply Selection
- MCU Module (enCoRe II / enCoRe V)
- WUSB-NL Module
- LCD Module
- LEDs
- Reset, Push Buttons, and Potentiometer
- Bread Board Area
- USB Mini-B Interface
- Debug Port
- ISSP Interface
- I2C Interface

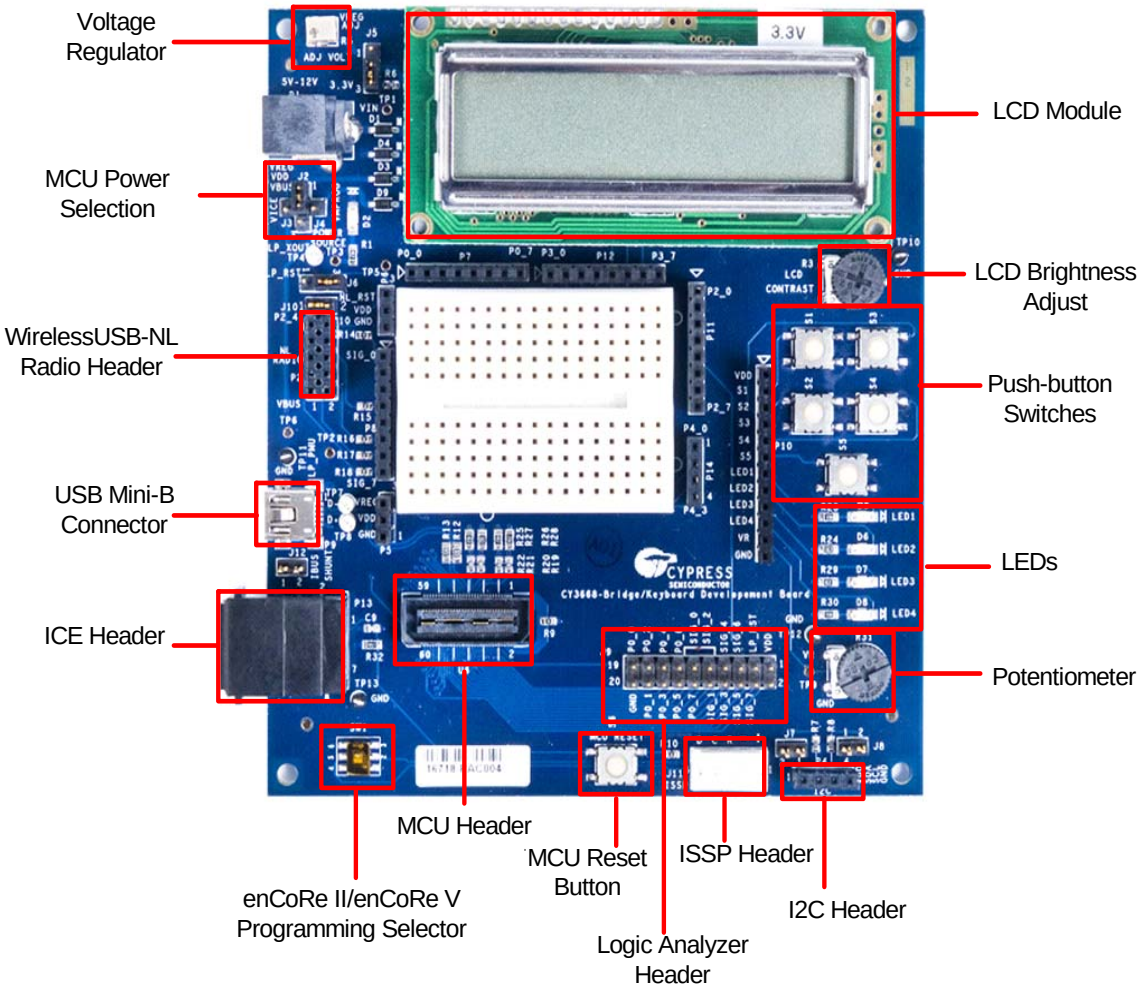
Figure 4-1. System Block Diagram



4.2 Functional Description

Figure 4-2 shows the different functional blocks on the CY3668 bridge/keyboard development board.

Figure 4-2. CY3668 Development Board Functional Blocks

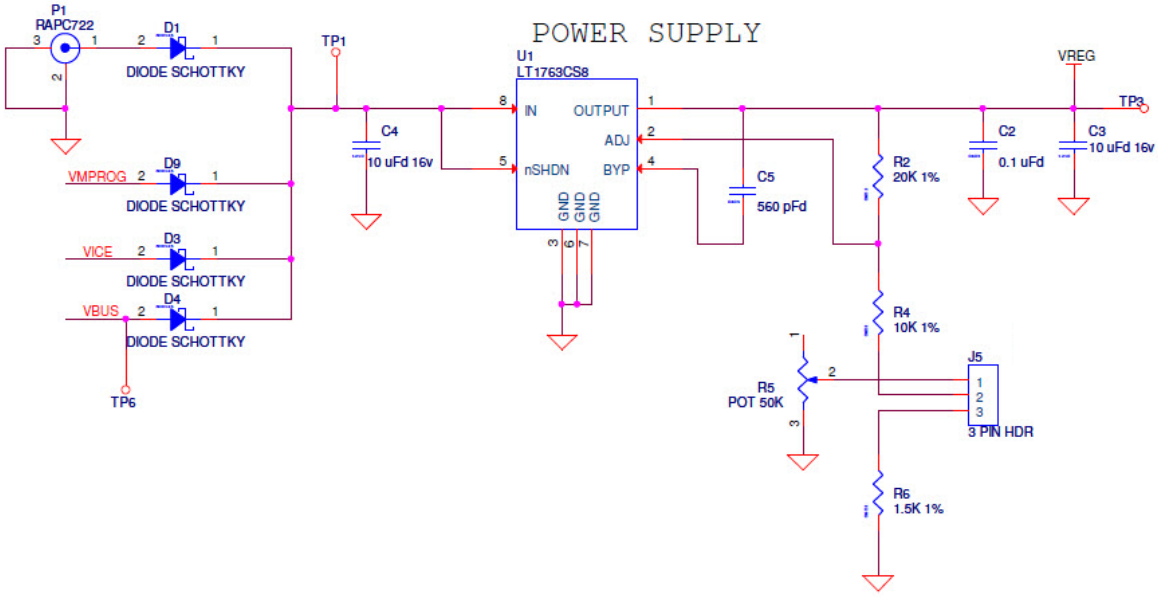


4.2.1 Voltage Regulator

The following circuit diagram shows the kit power supply design. On the kit, the regulator is present under the LCD module (you can unmount this module from the kit). There are four input sources to the voltage regulator:

- Power jack Mini 0.08-inch RA PCMT connector
- MiniProg
- PSoC ICE
- USB mini-B connector

Figure 4-3. Voltage Regulator Circuit Diagram



The voltage regulator regulates the inputs mentioned earlier and provides a fixed 3.3-V output or an adjustable voltage in the range 3.66 V to 1.63 V, depending on the J5 jumper and R5 (potentiometer) configuration. Table 4-1 gives details about how the different configurations can be achieved.

Table 4-1. Voltage Regulator Configuration

Configuration	Setting
3.3 V fixed	J5: Short pins 2 and 3
Adjustable voltage from 3.66 V to 1.63 V	J5: Short pins 1 and 2 Adjust the R5 potentiometer

Note When using the adjustable power supply configuration of the kit, use a multimeter at TP3 to monitor the output of the regulator.

Important Before changing any power supply configuration on the kit, first power off the kit and unmount any MCU modules or WirelessUSB-NL modules on the kit. When all the power configurations are fixed, remount all the necessary modules to resume operation.

4.2.2 MCU Power Selection

Figure 4-4. MCU Power Selection Circuit Diagram

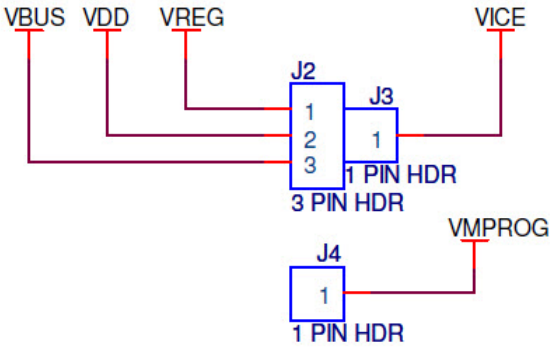


Figure 4-4 shows the power selection jumper configuration for the MCU module. The MCU module can be powered directly from any of the following voltage sources:

- Voltage regulator
- USB bus
- PSoC ICE
- MiniProg

Table 4-2 shows the configurations to select the different voltage sources.

Table 4-2. Voltage Source Configuration

Voltage Source	Jumper Setting
Voltage regulator	J2: Short pins 1 and 2
USB bus	Short J2: Pins 2 and 3 J12: Pins 1 and 2
PSoC ICE	Short J3: Pin 1 J2: Pin 2
MiniProg	Short J4: Pin 1 J2: Pin 2

4.2.3 WirelessUSB-NL Radio Header

Figure 4-5. WirelessUSB-NL Interface Circuit Diagram

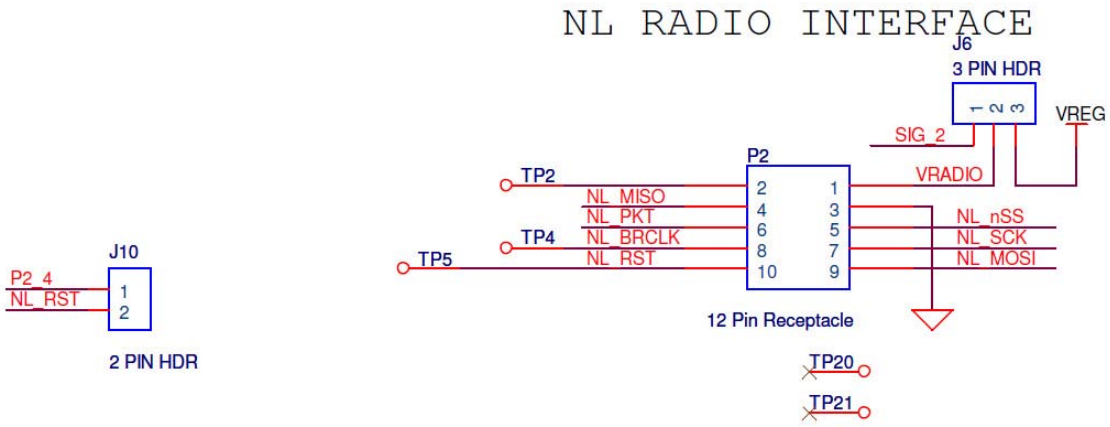


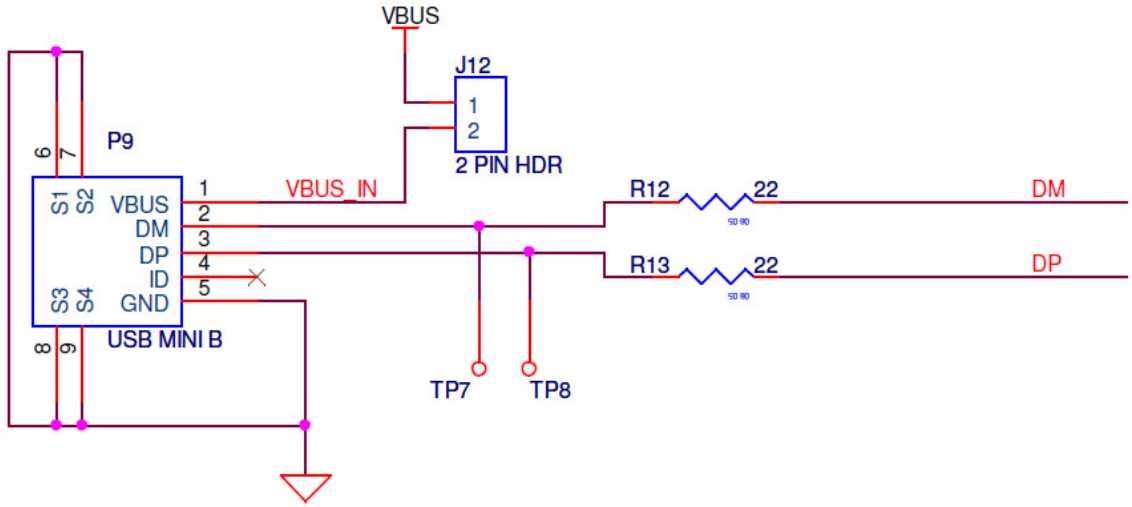
Figure 4-5 shows the WirelessUSB-NL radio header and other jumper settings in detail. By default, port 2, bit 4 (P2.4) is connected to the WirelessUSB-NL radio reset through jumper J10. This jumper will be populated (short) by default. Remove this jumper to disconnect P2.4 from the WirelessUSB-NL reset. When disconnected, the WirelessUSB-NL reset can be reconnected to other port pins by using jumper wires.

The J6 jumper provides power to the radio module. By default, pins 2 and 3 of J6 are shorted. In this configuration, the radio module is powered from output of voltage regulator. The other jumper setting possible is shorting pins 1 and 2 of jumper J6. In this configuration, the radio module can be powered from port 1, bit 2.

4.2.4 USB Mini-B Connecter

The kit can be powered using the USB bus power. This can be achieved by shorting pins 1 and 2 of the J12 jumper. In this configuration, the kit can operate as a bus-powered device.

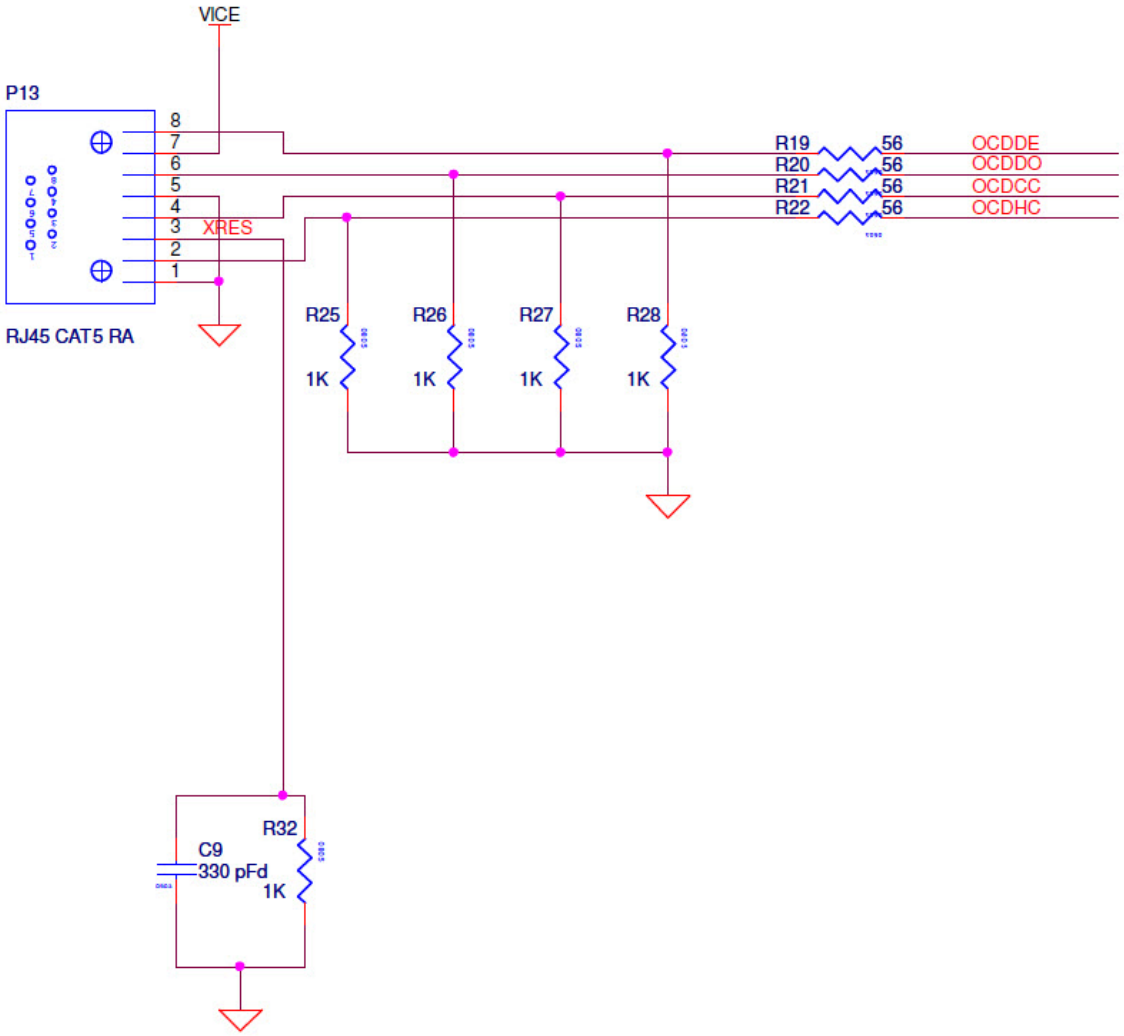
Figure 4-6. USB Connector and Power Jumper Circuit Diagram



4.2.5 ICE Header

The ICE-Cube debugger allows debugging and viewing the contents of specific memory locations. The ICE-Cube debugger can be connected to the RJ45 connector on the board to connect to the MCU with on-chip debugger (OCD) capability to enable debugging.

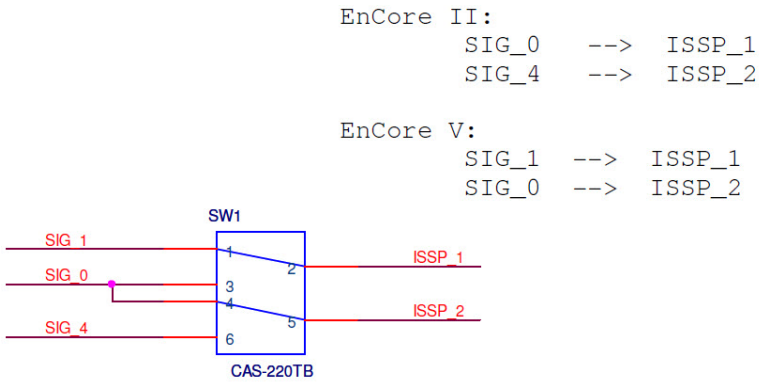
Figure 4-7. ICE Debug Header Circuit Diagram



4.2.6 enCoRe II/enCoRe V Programming Selector

Using the selector switch (SW1), you can choose between enCoRe V and enCoRe II modules for programming. See Figure 4-8 for the MCU programming selector switch circuit diagram.

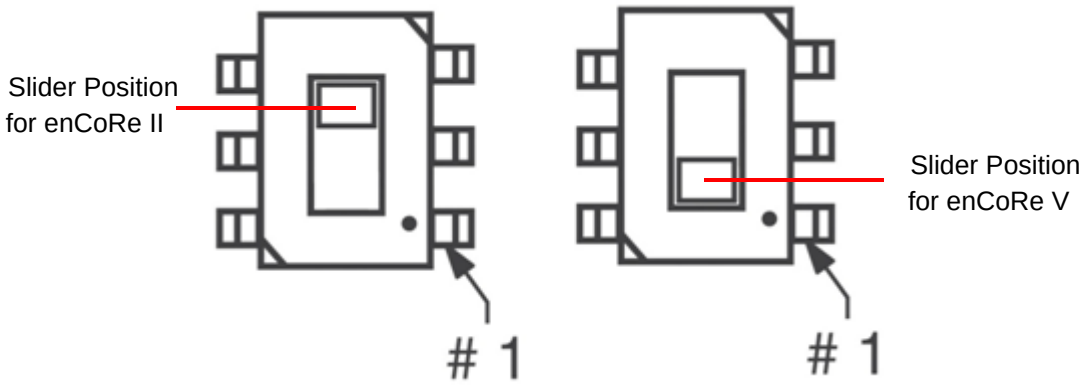
Figure 4-8. MCU Programming Selector Switch Circuit Diagram



The following programming configurations are possible through the selector switch:

- To program an enCoRe II module, position the slider of the switch away from the pin#1 of the switch, as indicated in figure 9.
- To program an enCoRe V module, position the slider of the switch towards the pin#1 of the switch, as indicated in figure 9.

Figure 4-9. Slider Orientation in Relation to Pin#1 for both MCUs



4.2.7 MCU Header

Figure 4-10 shows the MCU header and connections. This kit support Cypress's enCoRe V and enCoRe II MCUs via replaceable modules, which are supplied with this kit.

Cypress's enCoRe V Full-Speed USB peripheral microcontroller (MCUs) and enCoRe V LV (low-voltage) wireless MCU provide the following capabilities:

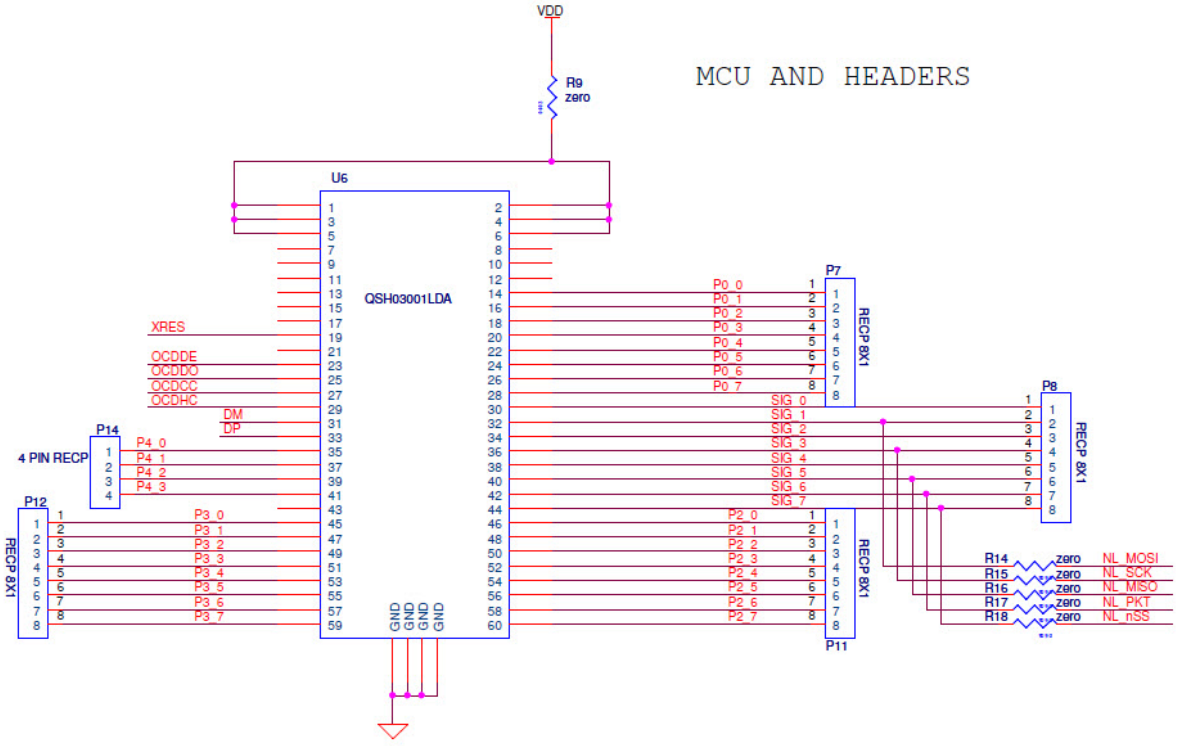
- Up to 32 KB of flash memory
- Three 16-bit timers
- Up to 36 general-purpose I/Os (GPIOs) to accommodate enhanced multimedia features in human interface devices (HIDs).

The families are supported by the CY3668 enCoRe V/LV development kit, which shortens the design time of laser mice, gaming controllers and keyboards, wireless dongles, remote controls, mobile handset accessories, and point-of-sales devices.

The in-system reprogrammable enCoRe V and enCoRe V LV MCUs provide design flexibility with a 10-bit ADC, flash memory with EEPROM emulation, and small form-factor packages. The enCoRe V devices also include eight USB endpoints. The enCoRe V LV family operates in the full range of 1.7 V to 3.6 V, with low power consumption for prolonged battery life in wireless applications, especially when paired with Cypress's low-power WirelessUSB-NL 2.4-GHz radio. All enCoRe MCUs include Cypress's patented crystal-less oscillator and integrated pull-up resistor to reduce component count and bill of material (BOM) cost.

The enCoRe II is a programmable low-speed USB controller. It eliminates the external crystal or resonator, pull-up resistors, wakeup circuitry, and 3.3-V regulator to reduce the overall system cost. The enCoRe II features a wide selection of I/O (up to 20 GPIOs) and memory (up to 8 KB of flash for user code and 256 bytes of RAM) options targeted for USB applications. The enCoRe II LV is a low-voltage, low-cost 8-bit flash controllable microcontroller, bringing enhanced component reduction and enCoRe II functionality to non-USB applications. By integrating an integrated oscillator, wakeup circuitry, and flash memory, enCoRe II LV helps to reduce the overall system costs.

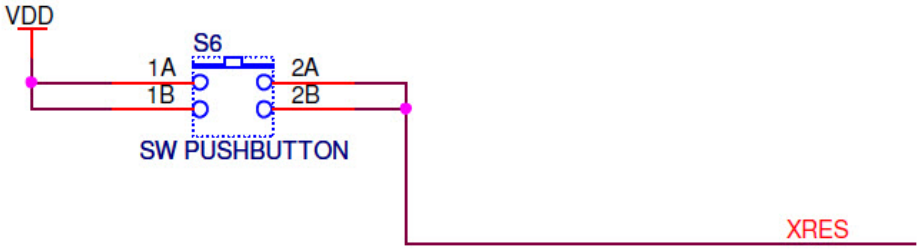
Figure 4-10. MCU Header Circuit Diagram



4.2.8 MCU Reset Button

The MCU reset button can be used to reset the MCU. Figure 4-11 shows the reset circuit.

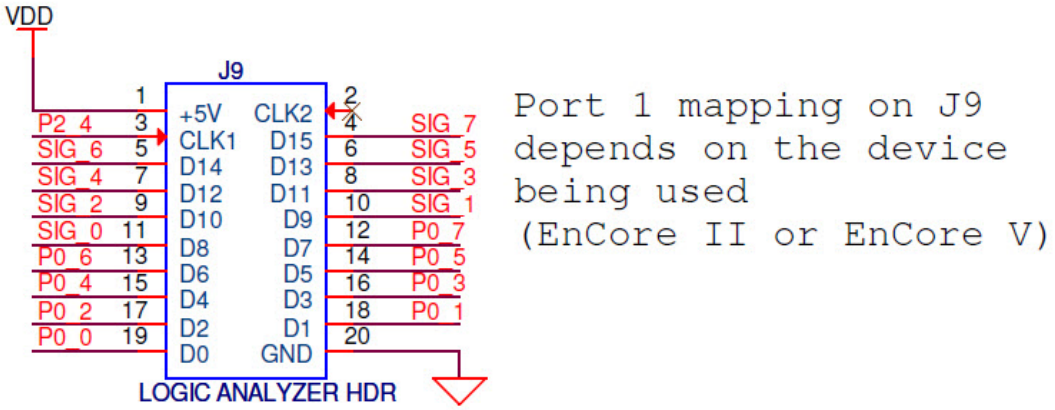
Figure 4-11. MCU Reset Button



4.2.9 Logic Analyzer Header

The logic analyzer header can be used for viewing different signals on an oscilloscope or the logic analyzer. This interface enables hardware level debugging. Figure 4-12 shows the circuit diagram for the header.

Figure 4-12. Logic Analyzer Header Details

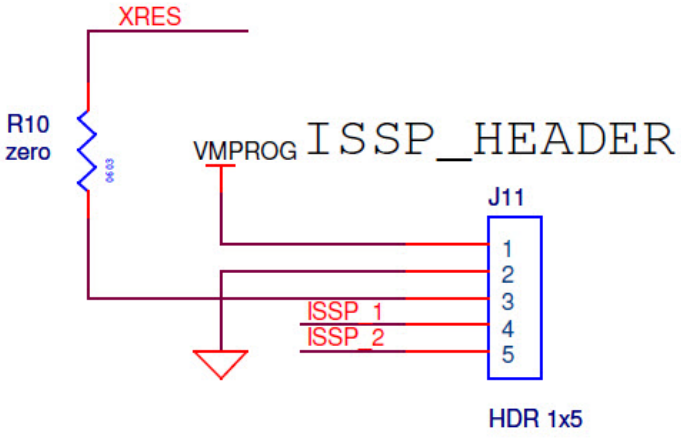


4.2.10 ISSP Header

In-system serial programmer (ISSP) is used to program the device using the MiniProg programmer device and the USB cable. ISSP/I2C programming is done through the 5-pin connector.

Figure 4-13 shows the pin mapping for the ISSP connector.

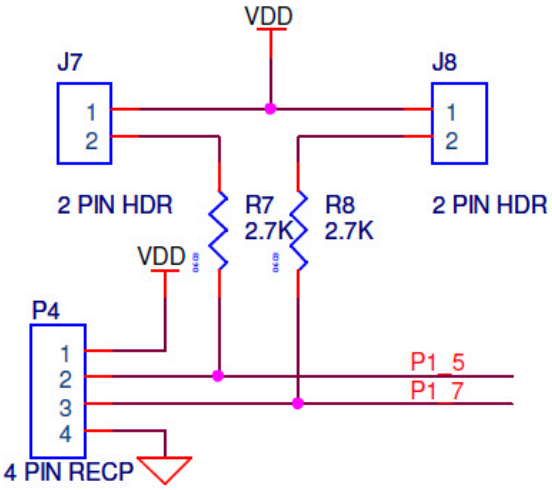
Figure 4-13. ISSP Header Circuit Diagram



4.2.11 I2C Header

The CY3668 development board is provided with an I2C connector.

Figure 4-14. I2C Header



4.2.12 Potentiometer

This potentiometer is used for varying the voltage to the ADC input. The output of the potentiometer can be accessed from P10, pin 11. See Figure 4-15 on page 24 for the circuit diagram.

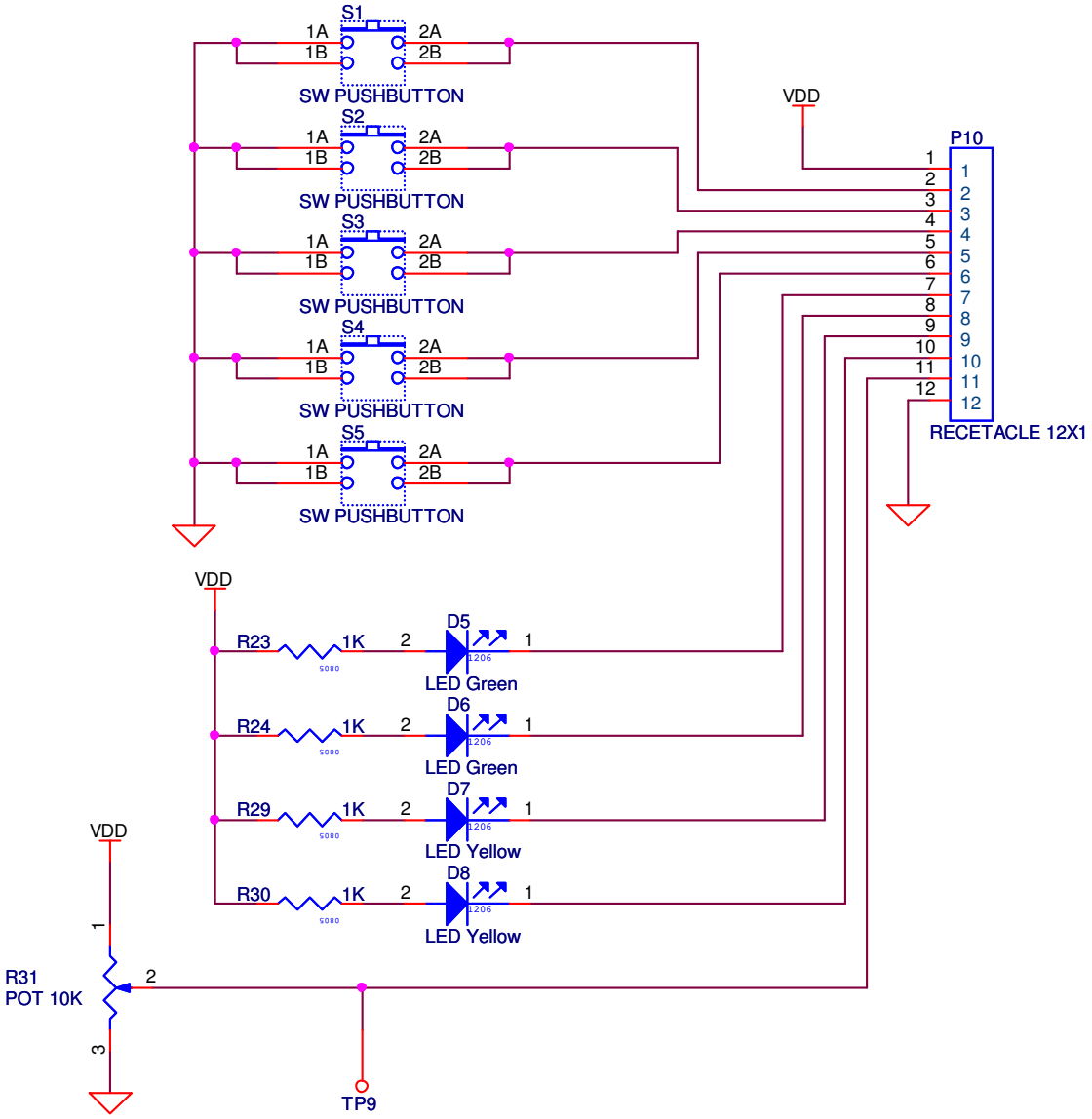
4.2.13 LEDs

The LEDs provides a mechanism to indicate the firmware status. They are exposed via P10, pins 7 through 10. See Figure 4-15 on page 24 for the circuit diagram.

4.2.14 Push-button Switches

The switches provide firmware inputs to users. They are exposed via P10 pins 1 through 6. See Figure 4-15 on page 24 for the circuit diagram.

Figure 4-15. Potentiometer, LEDs, and Switches



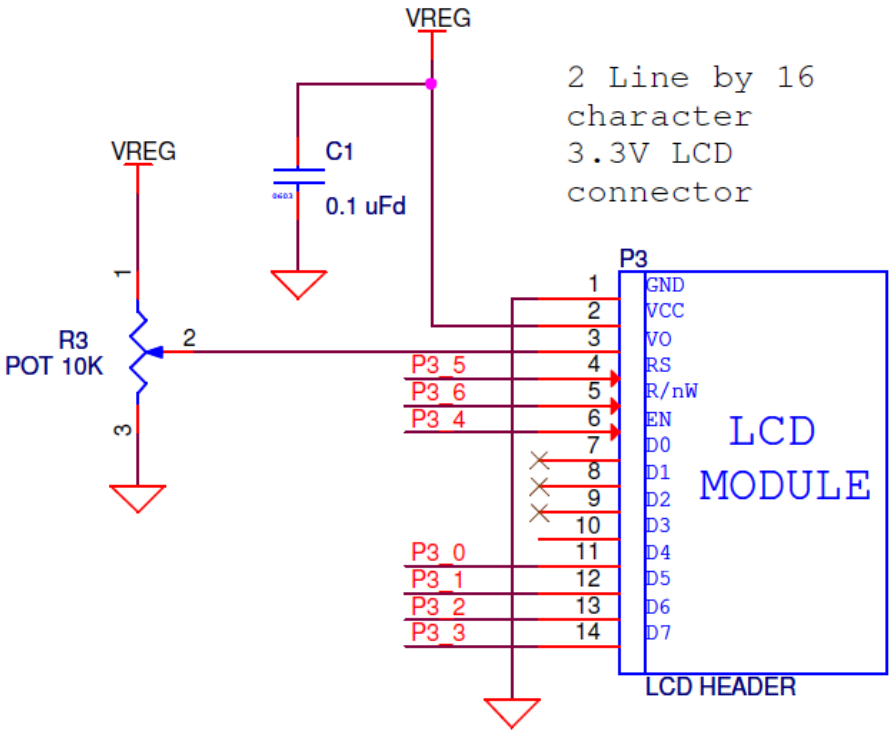
4.2.15 LCD Brightness Adjust

The contrast of the LCD can be varied using the LCD contrast potentiometer. See Figure 4-16 for the circuit diagram.

4.2.16 16×2 LCD Module

The LCD interface connector is used to connect an LCD. It is a two-line 16-character LCD.

Figure 4-16. LCD Interface and Brightness Control



5. Enhanced AgileHID™ Protocol 3.0

5.1 Overview

The Enhanced AgileHID™ protocol is a proprietary wireless protocol developed by Cypress, specifically designed for human interface device (HID) applications such as wireless mouse and keyboard. This protocol runs on Cypress's WirelessUSB-NL radio driver. Refer to the WirelessUSB-NL Radio Driver API document that comes along with this kit for protocol requirement.

The Enhanced AgileHID protocol provides a set of application programming interfaces (APIs) to develop wireless mouse, keyboard, generic devices, and bridge (dongle) applications. This protocol runs on Cypress's radio driver and enCoRe II or enCoRe V controllers. The protocol is designed to interface with C-based applications and consists of the following files:

- Protocol.c
- Protocol.h
- Usb.c (for bridge only)
- Usb.h (for bridge only)

Because the bridge is connected to the PC through USB, this protocol also uses USB driver APIs to transfer packets to and from the PC. See the HID Specifications for USB driver requirements for this protocol. This protocol also uses other controller peripherals (such as interrupt controllers, timers, and GPIOs).

5.2 Protocol Functions

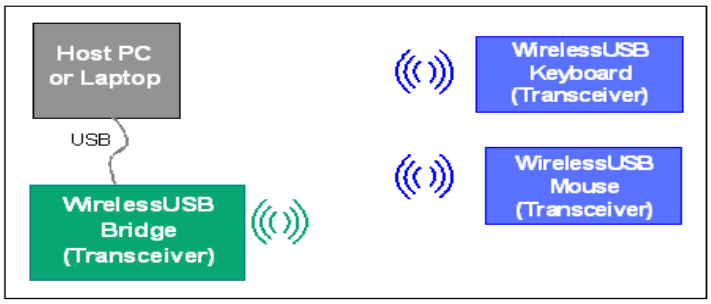
The following functionalities are visible to the application:

- Binding device (mouse/keyboard) with bridge connected to PC
- Packet transmit
- Packet receive

The following functionalities of the protocol are not visible to the application:

- Configuring transmit, receive, and transaction parameters
- Channel change algorithm to handle interference/noise
- Reconnect between device and bridge
- CRC generation
- Network ID generation

Figure 5-1. System Level Block Diagram



5.2.1 Protocol API Use

The protocol is divided into two modules: master protocol on the bridge and slave protocol on the device (keyboard/mouse). Only one of the modules can be present in a subsystem (either slave or master protocol). The protocol module is selected using a `define` during the pre-compilation stage. The protocol functions specific to a module are prefixed with the respective module name. For example, the “protocol init” function 'MasterProtocolInit' for master and 'SlaveProtocolInit' for slave. Some APIs are common to both the modules and are not prefixed with a module name.

5.2.1.1 Master (Bridge)

Initialization. Before the protocol is used, it must be first initialized using API MasterProtocolInit. Configuration parameters are handled within the protocol and do not need to be passed.

Binding. This application uses the MasterProtocolButtonBindMode API to bind between the bridge and the device. This function takes retry count as the parameter. Retry count refers to the number of times the bridge waits for a bind request from the device.

Packet Polling and Processing. The function MasterProtocolDataMode is used to poll the packet and to pass required data to the USB. This function abstracts all complex functionality of the protocol such as channel hop, quiet channel detection, reconnecting to device, and so on. The response to the device, if needed, is also handled within this function.

Bridge Suspend. The bridge decides to 'suspend' when it receives the suspend command from the USB host. It also calls the RadioForceState API to force the state to sleep. If the remote wakeup feature is enabled, the bridge goes to the radio packet receive mode periodically. After a packet is received from a device, the application wakes up the USB.

5.2.1.2 Slave (Mouse/Keyboard)

Initialization. Before the protocol is used, it must be first initialized using the API SlaveProtocolInit. Configuration parameters are handled within the protocol and do not need to be passed.

Binding. The device application uses SlaveProtocolButtonBind API to bind between the bridge and the device. This function takes the device type as the parameter. The device is bound to the bridge as this device type.

Transmission. The transmit buffer is maintained by protocol. The application uses the API SlaveProtocolGetTxPkt to get the buffer and fills it with the data to be transmitted. It uses the API SlaveProtocolSendPacket to transmit a packet. This API is a blocking API. This function internally changes the channel and reconnects to a bridge during transmission failure.

Reception. According to the AgileHID protocol, the device is not expected to receive any packet from the bridge. For keyboard applications, the back-channel-data packet is used by the protocol to get some of the key status (such as Caps lock, Num lock, and Scroll lock) from the bridge. The status of keys received from the bridge is stored in rx_packet.data.payload[0x0]. The first three bits in this byte corresponds to status of NUM LOCK, CAP LOCK, and SCRL LOCK respectively. The keyboard application is expected to implement a function "void NotifyDownloadBackChannelData (void)". Based on the key status present in rx_packet.data.payload[0x0], this function can make the corresponding LED glow.

5.2.2 Requirements

5.2.2.1 Header Files

To use the protocol, include protocol.h in any file that calls protocol functions.

5.2.2.2 Software Interface

The Enhanced AgileHID Protocol runs on Cypress's WirelessUSB-NL radio driver and PSoC Designer USB User Module. It also uses some of the utility functions such as timer, flash read/write, and so on. Typically, these utility functions along with sample mouse, keyboard, and bridge applications are provided along with the DVK; therefore, the user only needs to focus on the application.

5.2.3 Type Declarations and Definitions

5.2.3.1 BACK_CHANNEL_SUPPORT

This definition is needed by the protocol regardless of whether it requires backchannel support.

```
DEVICE_TYPE
typedef enum _DEVICE_TYPE
{
    PRESENTER_DEVICE_TYPE     = 0x00,
    RESERVED_DEVICE_TYPE      = 0x01,
    KEYBOARD_DEVICE_TYPE      = 0x02,
    MOUSE_DEVICE_TYPE         = 0x03,
}DEVICE_TYPE;
```

```
PROTOCOL_STATUS
typedef enum _PROTOCOL_STATUS
{
    PROTOCOL_SUCCESS,
    PROTOCOL_FAILED,
    PROTOCOL_TX_TIMEOUT

} PROTOCOL_STATUS
```

5.2.4 Protocol High Level Functions

5.2.4.1 MasterProtocolInit

```
void MasterProtocolInit (void);
```

Parameters: None

Return Value: None

Description: This function initializes the Enhanced AgileHID protocol and is called during bridge application initialization. Platform specific GPIO, SPIM, timer, and PSoC Designer USB User Module initialization should be done before calling this function.

5.2.4.2 *MasterProtocolDataMode*

```
void MasterProtocolDataMode (void);
```

Parameters: None

Return Value: None

Description: This function checks if any packet is received from the device. If received, it processes the packet. The transfer of the HID packet to USB is internally done by this function. It also ensures channel change in the event of a noisy environment. This function is continuously called by the bridge application code.

5.2.4.3 *MasterProtocolButtonBindMode*

```
void MasterProtocolButtonBindMode(unsigned short retry_count)
```

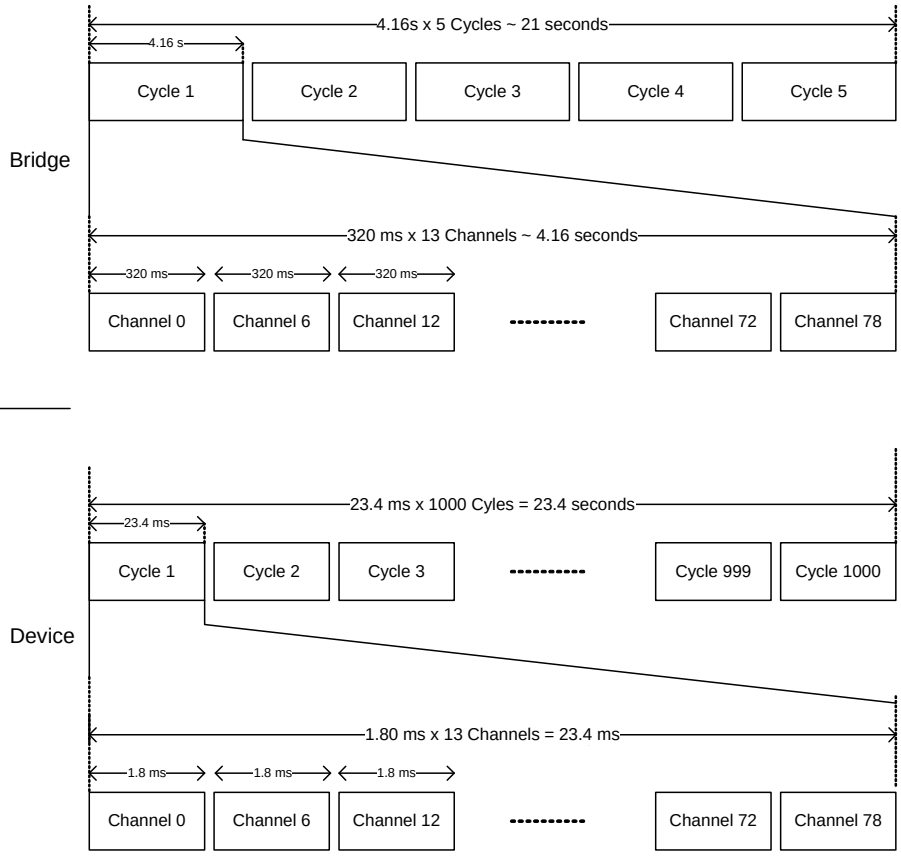
Parameters: unsigned short retry_count - Number of times the bridge changes the channel before it times out.

Return Value: None

Description: The bridge application invokes this function when a bind event, such as bind button press, occurs. In this API, the bridge protocol polls for the bind request in all the bind channels in a periodic manner, as shown in Figure 5-2. It continues to poll for a bind request until it receives a bind request or times out. From the device side, it transmits a bind request and waits for the bind response. The device continues to do this in all the bind channels in a round-robin manner. The polling time slot of the bridge for each channel is big enough to accommodate one cycle of device transmitting connect request in all the channels. The device is bound to at least one channel assuming at least one of the bind channels is quiet.

This function takes retry_count as a parameter to decide when to time out. The typical value is around 65.

Figure 5-2. Bridge Protocol Polls for Bind Request



5.2.4.4 *CheckUsbIdle*

```
void CheckUsbIdle(void)
```

Parameters: None

Return Value: None

Description: This function is called by the bridge application to find out whether the USB bus is idle for a specified time. If it is idle, then this function sends the last HID report to the USB host. This is to emulate a wired HID.

5.2.4.5 *CheckUsbSuspend*

```
void CheckUsbSuspend(void)
```

Parameters: None

Return Value: None

Description: This function is called by the bridge application to find out whether the USB bus is suspended. If it is suspended and remote wakeup is not enabled, then this function keeps the system in sleep mode to save power. If remote wakeup is enabled, then this function sends a wakeup signal to the USB host when a packet is received from the device.

5.2.4.6 SlaveProtocolInit

```
void SlaveProtocolInit(void)
```

Parameters: None

Return Value: None

Description: This function initializes the Enhanced AgileHID protocol and is called during device application initialization. Platform specific GPIO, SPIM, and timer initialization must be done before calling this function.

5.2.4.7 SlaveProtocolSendPacket

```
PROTOCOL_STATUS SlaveProtocolSendPacket (DEVICE_TYPE dev_type,
DEVICE_TYPE data_dev_type,
unsigned char data_length,
unsigned char back_channel
)
```

Parameters:

```
DEVICE_TYPE dev_type - Type of device
DEVICE_TYPE data_dev_type - Type of device in packet. Ex - keyboard trans-
mitting mouse packet.
unsigned char data_length - length of packet
unsigned char back_channel - flag. True if back channel data expected from
bridge.
```

Return Value: None

Description: This function is used by the device application to transmit a packet. The second parameter decides the mouse or keyboard packet to be transmitted. The device application populates the buffer that it receives from the SlaveProtocolGetTxPkt API.

5.2.4.8 SlaveProtocolGetTxPkt

```
void * SlaveProtocolGetTxPkt (void)
```

Parameters: None

Return Value: Void * - pointer to the buffer where device application fills with data

Description: This function is called by the device application to know where to fill the data to be transmitted.

5.2.4.9 SlaveProtocolButtonBind

```
void SlaveProtocolButtonBind (DEVICE_TYPE dev_type)
```

Parameters:

DEVICE_TYPE dev_type - Type of device with which it is bound to the bridge.

Return Value: None

Description: The device application invokes this function whenever a bind event such as bind button press occurs. In this API, the device continuously tries to bind to the bridge in all bind channels. During binding, the device sends out a bind request to the bridge and waits for the bind response. If it does not receive the response, it continues to repeat it in all the bind channels. See the description of MasterProtocolButtonBindMode on page 30.

5.2.4.10 *RadioSendPacket*

```
PROTOCOL_STATUS RadioSendPacket (unsigned char retry, unsigned char
length)
```

Parameters:

unsigned char retry - Number of times to retry transmission

unsigned char length - Length of the packet to be transmitted.

Return Value:
`PROTOCOL_STATUS- PROTOCOL_SUCCESS, Transmission successful`
`- PROTOCOL_TX_TIMEOUT, Transmission failure`

Description: This function is a generic function used by both the bridge and device (mouse/keyboard) to transmit a packet. Typically, this function does not have to be called by an application. If the application wants to verify the basic transmit and receive functionalities, then this function can be used. The application fills the data to be transmitted in the buffer it received from the API SlaveProtocolGetTxPkt.

5.2.4.11 *RadioReceivePacket*

```
unsigned char RadioReceivePacket  ( void )
```

Parameters: None

Return Value:
`unsigned char- length of the packet received in bytes.`

Description: This function is a generic function used by both the bridge and device (mouse/keyboard) to receive a packet. Typically, this function need not be called by an application. If the application wants to verify the basic transmit and receive functionalities, then this function can be used. The received data is present in the protocol buffer (rx_packet.payload[]).

5.3 Application Packet Format

5.3.1 Mouse Application Packet Header Format

A typical three-button mouse application packet structure is shown in the following table. The packet formats only show the application report payload and not the protocol packet format.

First Byte	Second Byte	Third Byte
X Delta	Y Delta	Z Delta [0:4] Buttons [5:7]

If there is no change in Z Delta and Button status, then the mouse application can send only two bytes containing X and Y coordinates. The bridge can detect this based on the received packet length. However, the button or Z Delta value alone cannot be sent. Instead, the X and Y Delta value also should be sent keeping X and Y as zero.

5.3.2 Keyboard Application Packet Header Format

The first application report byte is Scan Code 1 if the byte is less than 0xF0. Otherwise, the first application report byte is the Application Report Header (Multimedia, Power, Battery, or Keep Alive). This also assumes that the multimedia and power keys do not use modifier keys and that 0xFF, 0xFE, 0xFD, and 0xFC are not valid Standard 101 key scan codes.

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

The following section shows the keyboard packet. The packet formats only show the application report payload and not the protocol packet format.

5.3.2.1 *Standard 101 Keys Report*

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

First Byte	Second Byte	Third Byte	Fourth Byte	Fifth Byte	Sixth Byte	Seventh Byte
Scan Code 1 (<0xFC)	Modifier Keys	Scan Code 2	Scan Code 3	Scan Code 4	Scan Code 5	Scan Code 6

5.3.2.2 *Multimedia Keys*

An Application Report Header of 0xFF indicates that this report is a Multimedia Keys report. The Multimedia Keys report format is as follows:

First Byte	Second Byte	Third Byte
Application Report Header (0XFF)	Hot Key Scan Code (upper 8 bits)	Hot Key Scan Code (lower 8 bits)

5.3.2.3 *Power Keys*

An Application Report Header of 0xFE indicates that this report is a Power Keys report. The Power Keys report format is as follows:

First Byte	Second Byte
Application Report Header (0XFE)	Power Key Scan Code

5.3.2.4 *Battery Voltage Level and Version Report*

An Application Report Header of 0xFD indicates that this report is a Battery Voltage Level and Version report. The Battery Voltage Level and Version report format are as follows:

First Byte	Second Byte	Third Byte	Fourth Byte	Fifth Byte	Sixth Byte
Application Report Header (0XFD)	Battery Voltage Level	FW Version		HW Version	

5.3.2.5 *Bridge Application Header Format*

The packets received from the devices are converted to standard HID packet, which can be detected by the HID class driver on the USB Host. For more details, see the HID specification.

6. Example Projects

The example projects that are provided with the kit demonstrate the basic functionality of a WirelessUSB HID keyboard and mouse. The following sections describe the theory of operation and the procedure to exercise the keyboard and mouse example projects with a bridge. Two development boards with WirelessUSB-NL radio modules are used. One board is used as a bridge, which enumerates on the PC as a bus-powered USB composite device with two interfaces – keyboard and mouse. The other board is used as a keyboard or as a mouse.

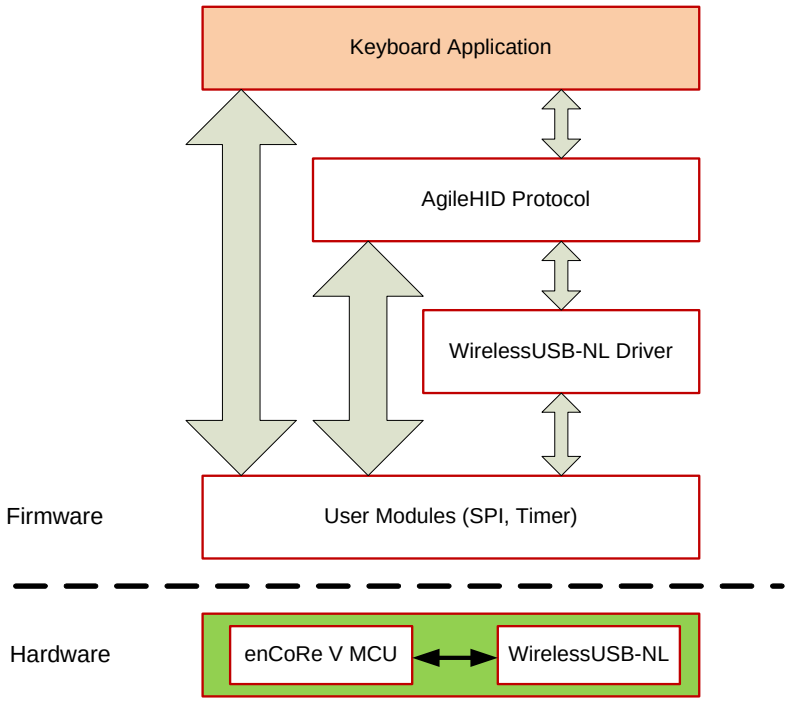
6.1 Theory of Operation

This section explains the design and implementation of wireless keyboard, mouse, and bridge example projects. This includes the software block diagram, flow chart, and details about the key functions that implement the application logic. These details help you to customize the example projects based on your requirements.

6.1.1 Keyboard

6.1.1.1 Firmware Block Diagram

Figure 6-1. Keyboard Firmware Block Diagram



The firmware runs on an enCoRe V MCU, which is interfaced to WirelessUSB-NL over an SPI interface. The firmware consists of the following three layers, apart from the standard PSoC Designer features such as timers:

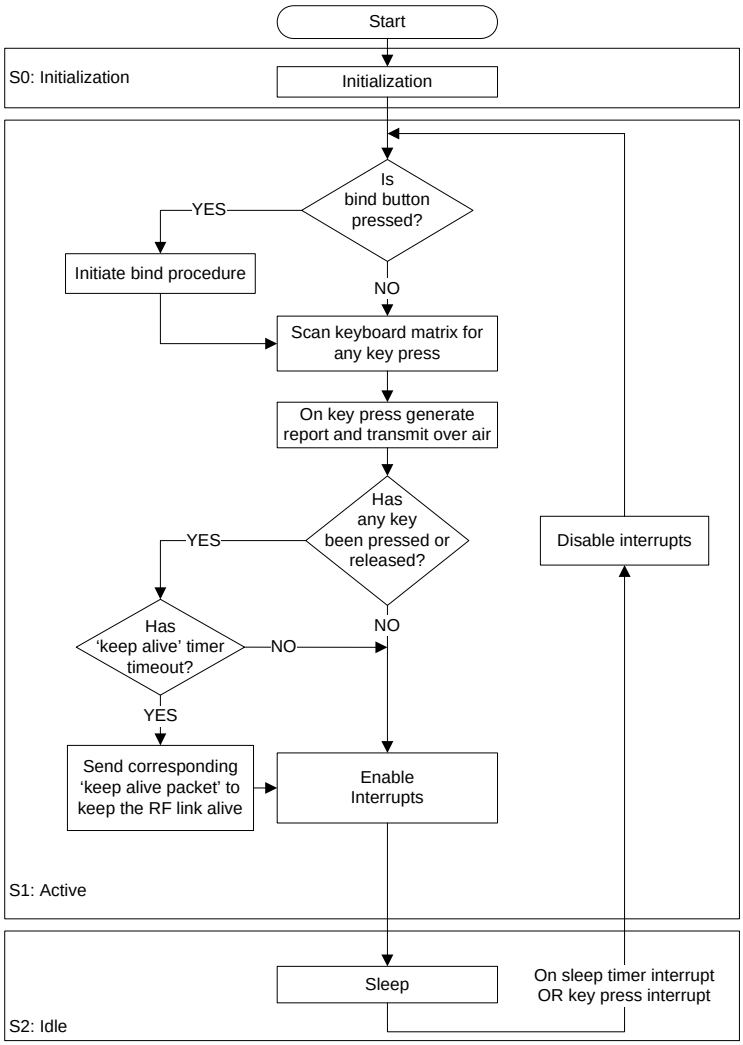
- WirelessUSB-NL Driver: This driver interfaces with the WirelessUSB-NL radio and provides initialization, transmit, and receive APIs.
- AgileHID Protocol: This layer implements the Human Interface Device (HID) protocol, which includes creation of HID packets using the keyboard input data, binding, and an RF channel hopping algorithm to handle RF interference. This layer accesses WirelessUSB-NL through the WirelessUSB-NL Driver.
- Keyboard Application: This layer implements the keyboard application logic.

The following sections describe the implementation of the keyboard application to enable you to customize the application logic based on your requirements. The remaining firmware layers (WirelessUSB-NL Driver and Enhanced AgileHID protocol) are fixed and need not be changed.

6.1.1.2 Top Level Program Flow

The following flow chart describes the top level program flow implemented in the keyboard example project.

Figure 6-2. Program Flow Chart



6.1.1.3 Code Details

The keyboard application logic is implemented in *keyboard.c* and the main() function is located in this file. This file is located at: `<Install_Directory>\Cypress\CY3668 DVK\1.0\Firmware\Keyboard\NL_Keyboard_enCoreV\NL_Keyboard_enCoreV\NL_Keyboard_enCoreV`. Note that the CY3668 DVK software is installed in the `<Install_Diretory>`. For example, its typical location on a Windows 7 64-bit PC will be `C:\Program Files (x86)`.

The following table lists the functions, which implement the application logic.

Table 6-1. Keyboard Code Details

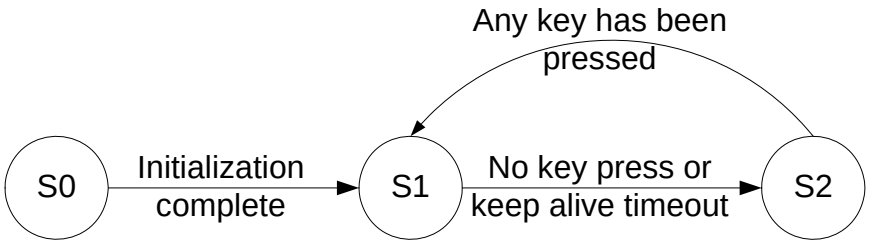
File	Functions

keyboard.c – implements a polling based keyboard.	static void ScanBindButton(void);
	static void DebounceInit(void);
	static void AgeDebounceQueue(void);
	static BOOL Debounce(UINT8 index);
	static void SendKeyboardReport(UINT8 report_size, UINT8 protocol_device_type);
	static void GenerateStandardReport(void);
	static void GenerateReport(void);
	static void ProcessColumn(UINT8 column_index, UINT8 * current_key_state_ptr);
	static void ScanKeyboard(void);
	static void KeyboardInit(void);

6.1.1.4 Keyboard Firmware Implementation

The following figure shows details of the different states the keyboard firmware passes through.

Figure 6-3. Keyboard Firmware Flow



S0: Initialization. When the keyboard is powered on, the firmware initializes different peripherals (such as GPIOs, timers, debounce queue, LCD, and WirelessUSB-NL). This is done by the KeyboardInit function. At the end of the keyboard initialization function, the message "WirelessUSB NL Keyboard" is printed on the LCD panel.

S1: Active. In the next step, the keyboard scans if the bind button was pressed and released. In this case the keyboard goes into the bind mode. This is implemented by the ScanBindButton.

After this, the ScanKeyboard function helps to scan the keyboard. This function polls each of the three lock buttons (switches) and updates their status. If any of the keys is pressed, that key is added to the debounce queue and a counter is initialized. This is implemented by the Debounce function. The value of this counter is determined by KEYBOARD_DEBOUNCE_COUNT. A keyboard packet is sent to the bridge only when the key remains pressed until the counter reaches 0. Note that

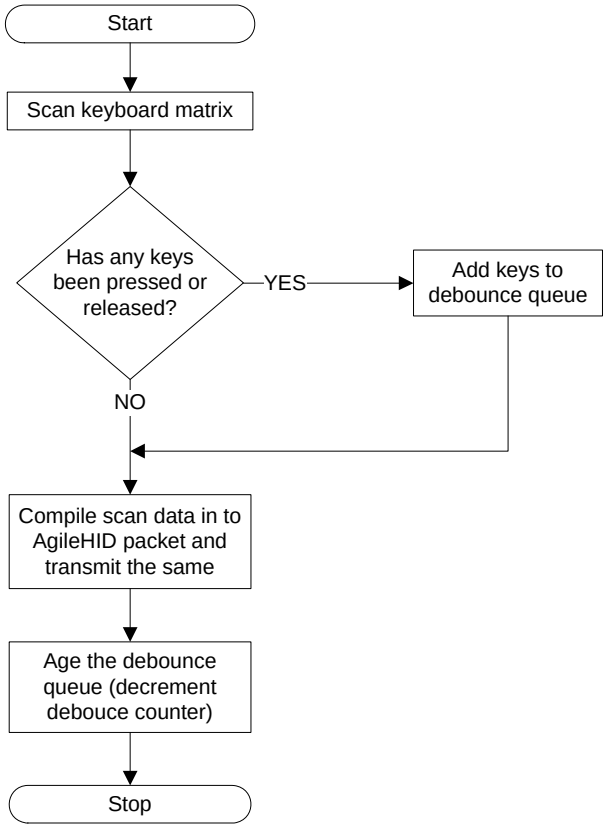
in the example project only the three lock buttons are implemented while in a real application the entire keyboard is scanned in the ScanKeyboard function.

The information stored in the debounce queue is processed by GenerateReport, with the help of GenerateStandardReport to generate an AgileHID keyboard packet. This report is sent to the bridge by the SendKeyboardReport function, which in turn calls the SlaveProtocolSendPacket API of the AgileHID protocol.

After a packet is transmitted, the keyboard tries to keep the radio link alive by transmitting 'Keep Alive' packets. These packets are transmitted on regular intervals governed by KEYBOARD_KEEP_ALIVE_TIMEOUT. If a key is kept pressed, the keyboard sends the corresponding key code along with the 'Keep Alive' packet. The same occurs when a key is released; however, there is a time limit, which is determined by KEY_UP_KEEP_LIVE_TIMEOUT.

Firmware keeps track of the keys present in the debounce queue by decrementing the counter value associated with each key. This is implemented in the AgeDebounceQueue function.

Figure 6-4. Flow Chart for Active State

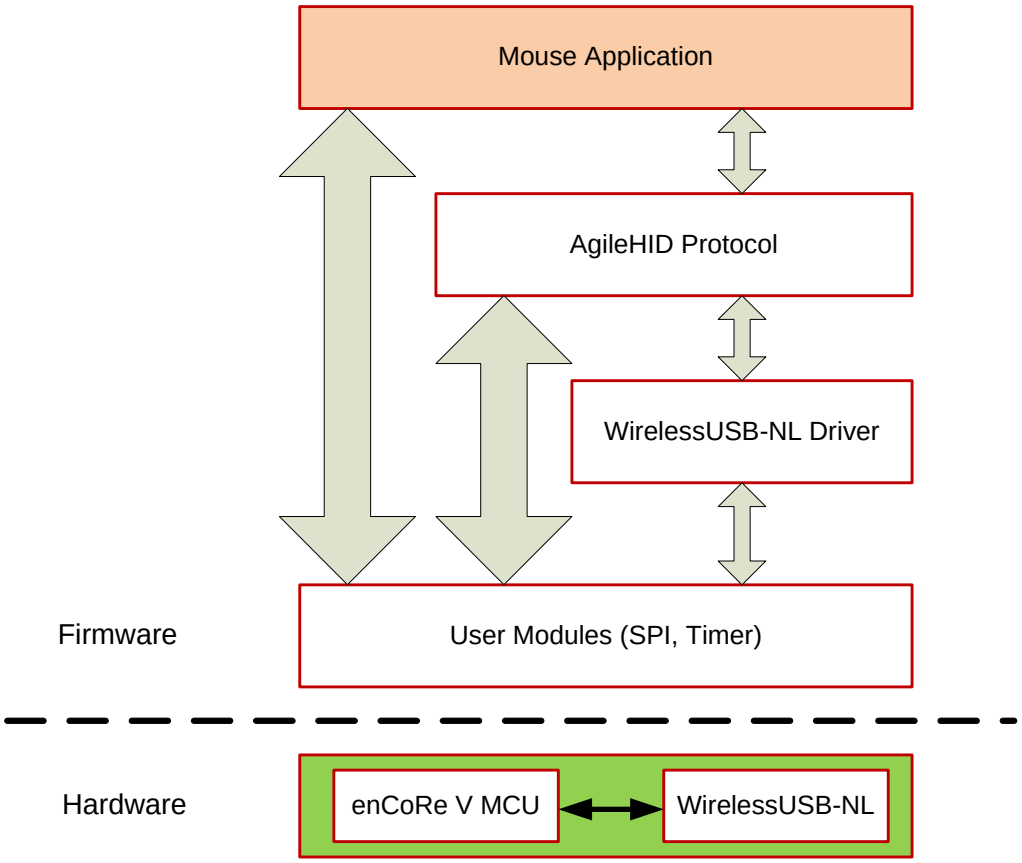


S2: Idle. This section discusses the battery-saving aspect. After the processing described in the previous sections is complete, the keyboard tries to sleep. This can happen on two occasions: first, when a key is held down and second, when none of the keys are held down. In the first case, the sleep period is governed by KEY_DOWN_DELAY_SAMPLE_PERIOD milliseconds. After this period expires, the keyboard wakes up and sends the key again. Second, if no keys are pressed the keyboard goes to sleep forever and wakes up only on a GPIO interrupt (from a key) or a timer. On wake-up, if no key press is detected the keyboard clears the watchdog timer and goes to sleep again. If keys are pressed, the firmware scans the keys and sends the report as mentioned above.

6.1.2 Mouse

6.1.2.1 Firmware Block Diagram

Figure 6-5. Mouse Firmware Block Diagram



The firmware runs on an enCoRe V MCU, which is interfaced to WirelessUSB-NL over SPI interface. The firmware consists of the following three layers, apart from the standard PSoC Designer features such as timers:

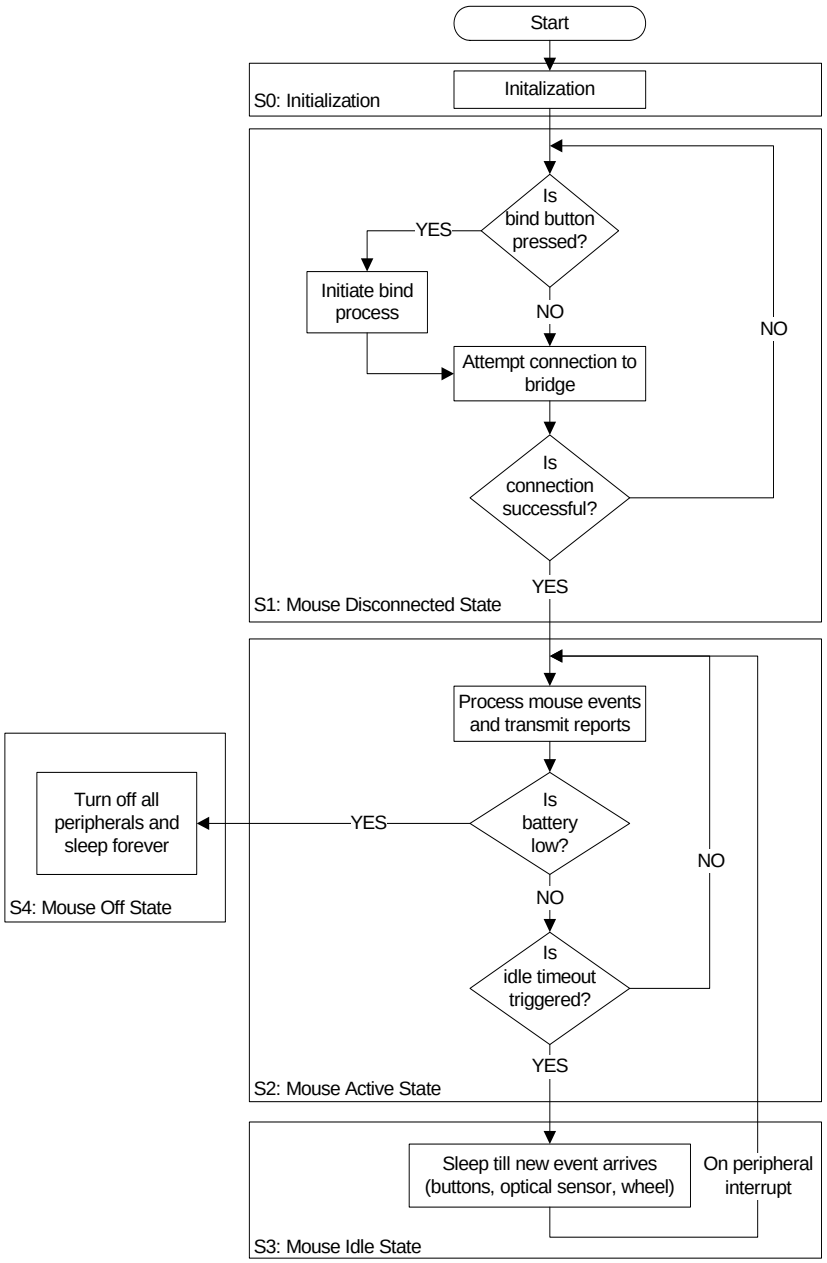
- WirelessUSB-NL Driver: This driver interfaces with the WirelessUSB-NL radio and provides initialization, transmit, and receive APIs.
- AgileHID Protocol: This layer implements the HID protocol, which includes creation of HID packets using mouse input data, binding, and an RF channel-hopping algorithm to handle RF interference. This layer accesses WirelessUSB-NL through the WirelessUSB-NL driver.
- Mouse Application: This layer implements the mouse application logic.

The following sections describe the implementation of the mouse application to enable you to customize the application logic based on your requirements. The remaining firmware layers (WirelessUSB-NL Driver and Enhanced AgileHID protocol) are fixed and need not be changed.

6.1.2.2 *Top Level Program Flow*

The following flow chart describes the top level program flow implemented in the mouse example project.

Figure 6-6. Program Flow Chart



6.1.2.3 Code Details

The mouse application logic is implemented in *mouse.c* and the main() function is located in this file. This file is located at `<Install_Directory>\Cypress\CY3668 DVK\1.0\Firmware\Mouse\NL_Mouse_enCoreV\NL_Mouse_enCoreV\NL_Mouse_enCoreV`. Note that the CY3668 DVK software is installed in the `<Install_Diretory>`. For example, its typical location on a Windows 7 64-bit PC will be `C:\Program Files (x86)`.

The following table lists the functions, which implement the application logic.

Table 6-2. Mouse Code Details

File	Functions
mouse.c – implements a state machine that will simulate a three button mouse.	void main(void) static void MouseInit(void) static void MouseGoDisconnected(void) static void MouseGoActive(void) static void MouseGoIdle(void) static void MouseGoOff(void) static void MouseDoBind(void) static void MouseDoReport(void) static void MouseDoBattLevel(void) static void MouseDoLowVoltage(void) BOOL MouseGetReport(TX_PACKET *packet) void RetrieveGloabSystemParameters(void) void StoreGloabSystemParameters(void)

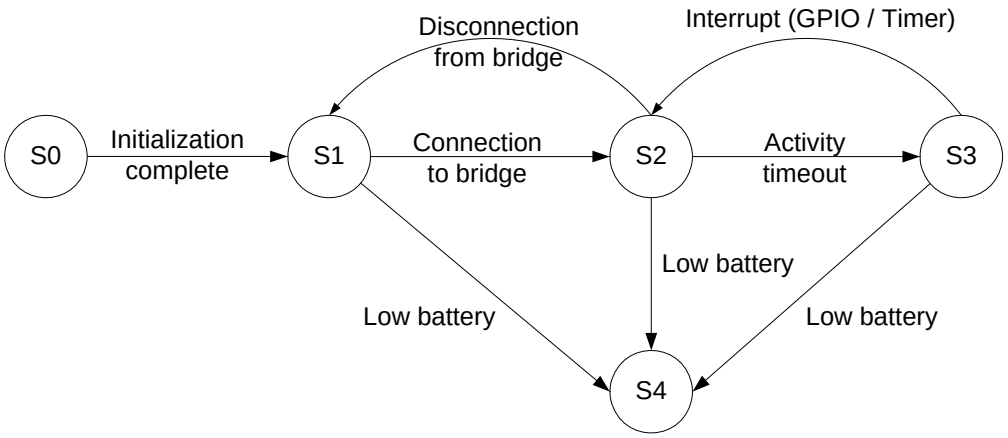
6.1.2.4 Mouse Firmware Implementation

The mouse firmware is a state machine implementation with the following states.

- S0: Initialization
- S1: Disconnected
- S2: Active
- S3: Idle
- S4: Off

The following figure shows details of the state machine implementation.

Figure 6-7. Mouse Firmware Flow



Most of the code in *main.c* and in other application files implements this state machine. To help you understand the firmware better, this section explains what each state does and the corresponding functions used to implement that state.

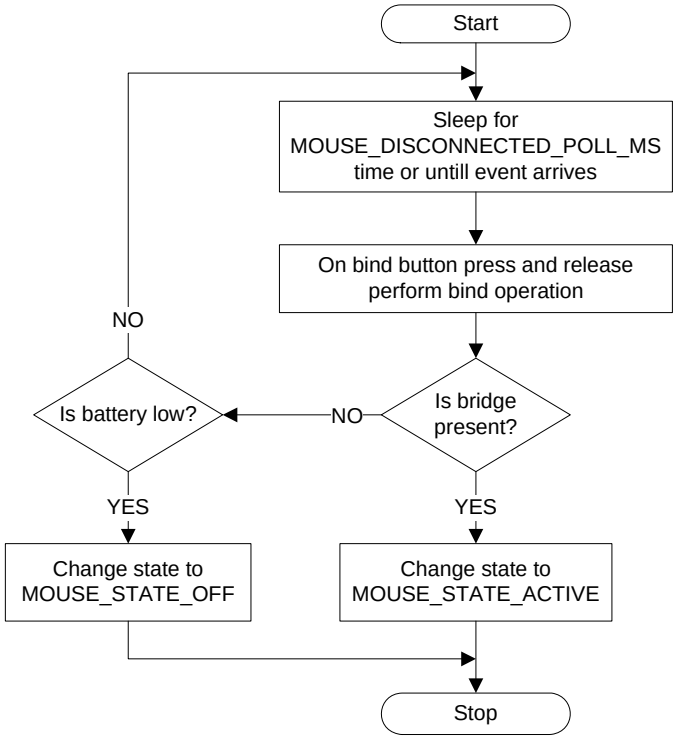
Initialization state (S0). On power-up, the mouse firmware initializes the various resources used. These resources include ports, interrupt service routines, timers, optical sensor, buttons, scroll wheel, SPI master, and radio. After all these resources are initialized, the LCD module displays the message "WirelessUSB-NL Mouse". During initialization the SPI Master User Module is set to mode 1 and MSB first configuration. The MouseInit function performs this initialization.

Disconnected state (S1). After all initializations are over, the mouse goes into the disconnected state (S1). The disconnected state is implemented by the MouseGoDisconnected function. The mouse sleeps for a specified time given by the MOUSE_DISCONNECTED_POLL_MS macro or until some kind of interrupt (such as a sensor, scroll wheel, or button interrupt) happens. In this case, each interrupt is considered as an event and information regarding this is stored in mouse_event. These mouse events are updated by corresponding ISRs present in the *isr.c*. This kind of sleep or wait is achieved by the TimerWaitEvent function.

When the firmware wakes up due to an expired timer or interrupt, it checks for a bind button press. If the bind button is pressed and released, the mouse performs a bind operation. These actions are performed by the MouseDoBind function.

The next step is to try and connect to the bridge to which the mouse is bound. This connection request is initiated by calling the SlaveProtocolSendPacket with no payload. If the protocol makes a successful connection with the bridge, the mouse changes its state to active (MOUSE_STATE_ACTIVE). If the connection is unsuccessful, the mouse checks for low battery. If low battery is detected, it goes to MOUSE_STATE_OFF; otherwise it goes to sleep. These steps are repeated until a connection is successfully established.

Figure 6-8. Flow Chart for Disconnected state



Active state (S2). The active state (S2) is entered only if there is a successful connection to the bridge or if there are any new events to report. The active state is implemented by the MouseGoActive function. In the active state, you must first check if there is enough power left in the battery. If the battery has low power, the mouse changes to the off state (MOUSE_STATE_OFF). This is achieved by the MouseDoLowVoltage function.

Next, the mouse generates and transmits reports for the new events that happened. This is achieved through the MouseDoReport function. MouseDoReport gathers the state of each peripheral (such as sensor, scroll wheel, and button interrupt) by calling the MouseGetReport function. MouseDoReport compiles this into an AgileHID packet and transmits it over the air through the protocol call SlaveProtocolSendPacket.

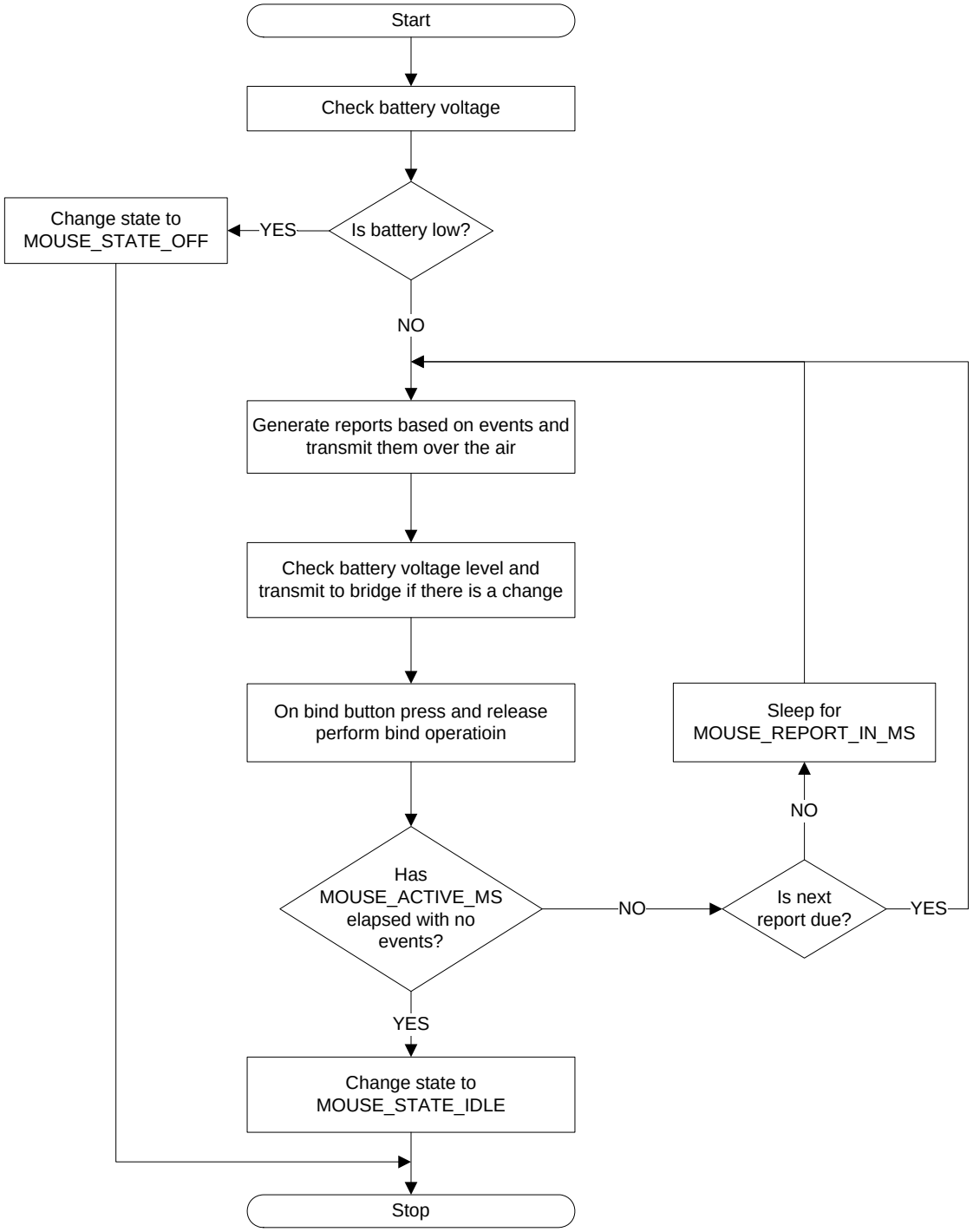
After all the events are processed, the mouse checks for the battery voltage level. The measured battery voltage levels are transmitted to the bridge. This is achieved through MouseDoBattLevel. This feature of the firmware can be turned off by undefining the MOUSE_BATTERY_STATUS macro. By default this feature is turned off.

The next step checks if the bind button was pressed and released. If this event is detected, the mouse performs a bind operation. These actions are performed by the MouseDoBind function.

After all the tasks are performed, the mouse waits (sleeps) for MOUSE_REPORT_IN_MS time. This wait time ensures the mouse maintains a specific report rate. Report rate is the number of reports that the PC receives from a mouse every second. By default this wait period is set to 6 ms to ensure a report rate of approximately 125. Increasing this interval lowers the report rate and decreasing this interval raises the report rate. The limitation of a low-speed bridge (enCoRe II) is that it can only maintain a report rate of 125. However, a full-speed bridge (enCoRe V) can do reports up to 1000 per second. If the mouse is bound to a low-speed bridge, it is beneficial (in terms of power consumption) to maintain a report rate of around 125 and sleep for rest of the time.

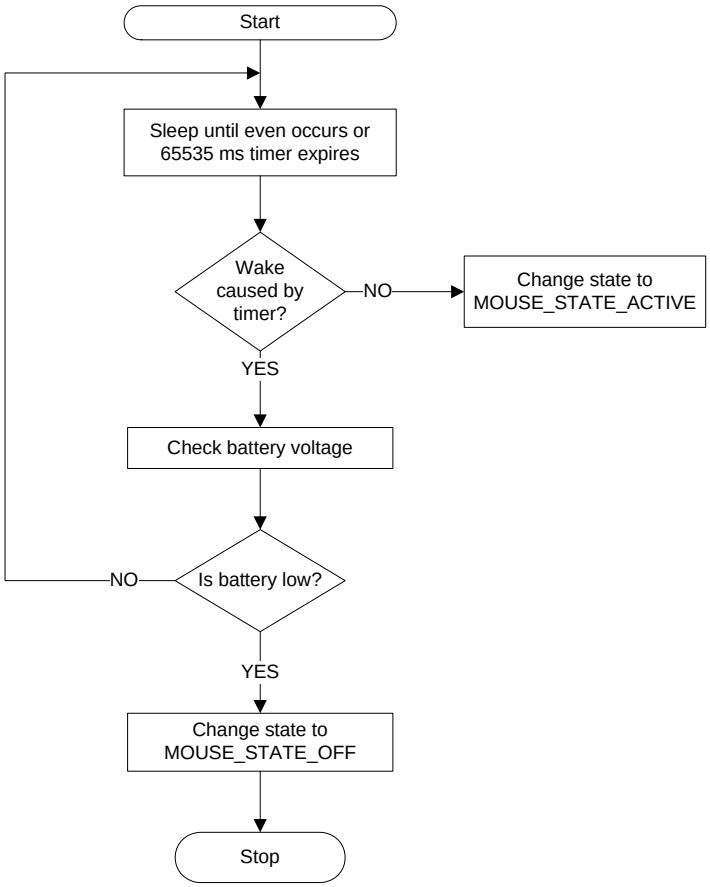
After the report interval timeout, the mouse goes back and does all the actions mentioned in this section. This process continues until there are no more events to be reported. Because there are no more events to report, the mouse counts down MOUSE_ACTIVE_MS. At the end of this countdown the mouse goes to the idle state. By default MOUSE_ACTIVE_MS is 8000 ms.

Figure 6-9. Flow Chart for Active State



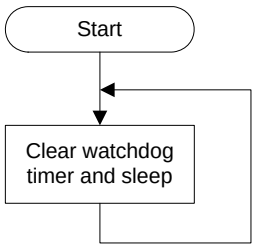
Idle State (S3). If the mouse is inactive and there are no events to report to the bridge for MOUSE_ACTIVE_MS time, then the mouse goes into the idle state. In the idle state the mouse sleeps for 65535 ms, waking up in between only for events such as a timer interrupt or GPIO interrupt. Ideally the mouse will sleep forever but the timer has a 16-bit limit. Due to this limitation the mouse wakes up every 65535 ms, checks the battery level, and then goes back to sleep. If there is any activity or event (such as a timer interrupt or a GPIO interrupt) during this state, the mouse state is changed to MOUSE_STATE_ACTIVE and goes to the active state. The idle state is implemented by the MouseGoIdle function.

Figure 6-10. Flow Chart for Idle State



Off State (S4). The mouse enters this state only if the battery voltage is too low for the device to operate. In this case the mouse turns off all peripherals and goes to sleep forever. The device remains in this state until the batteries are replaced. This state is implemented by the MouseGoOff function.

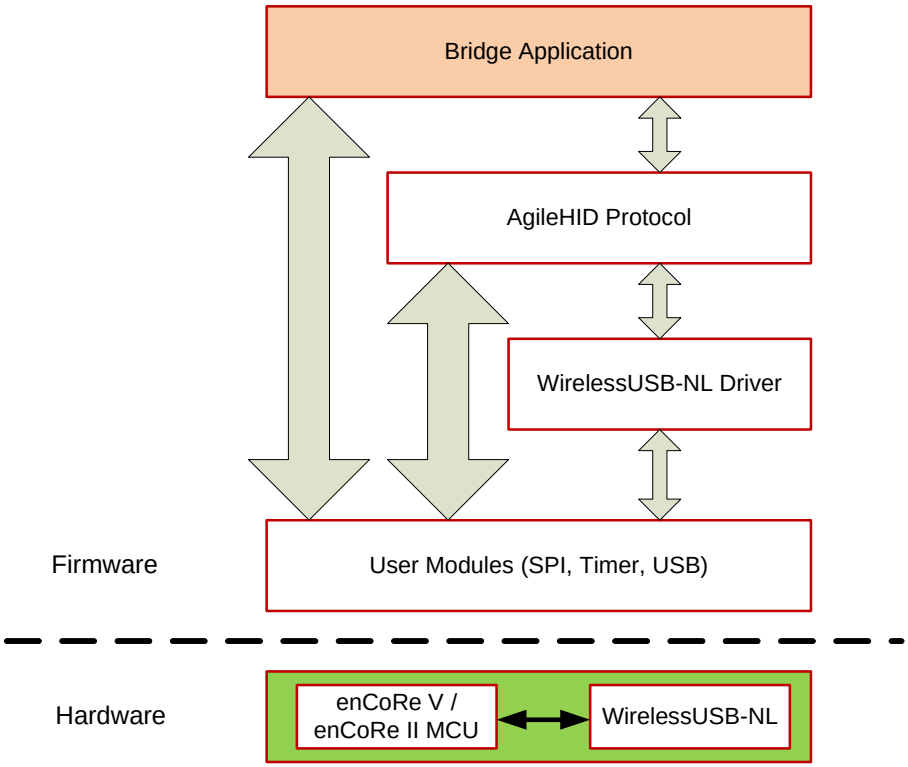
Figure 6-11. Flow Chart for Off State



6.1.3 Bridge

6.1.3.1 Firmware Block Diagram

Figure 6-12. Bridge Firmware Block Diagram



The firmware runs on an enCoRe V or enCoRe II MCU, which is interfaced to WirelessUSB-NL over SPI interface. The firmware consists of the following three layers, apart from the standard PSoC Designer features such as timers:

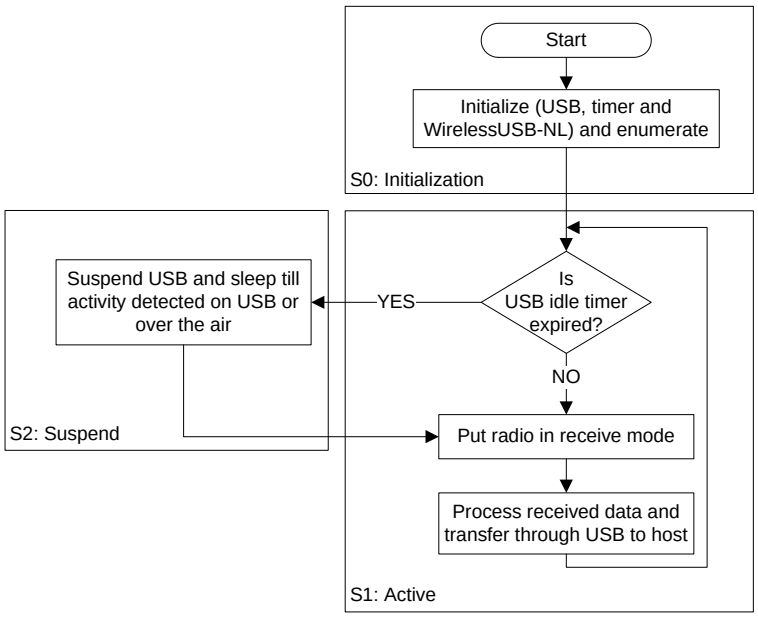
- WirelessUSB-NL Driver: This driver interfaces with the WirelessUSB-NL radio and provides initialization, transmit, and receive APIs.
- AgileHID Protocol: This layer implements the HID protocol, which includes reception of HID packets from a mouse and keyboard, binding, and an RF channel-hopping algorithm to handle RF interference. This layer accesses WirelessUSB-NL through the WirelessUSB-NL driver.
- Bridge Application: This layer implements the bridge application logic.

The following sections describe the implementation of the bridge application to enable you to customize the application logic based on your requirements. The remaining firmware layers (WirelessUSB-NL Driver and Enhanced AgileHID protocol) are fixed and need not be changed.

6.1.3.2 Top Level Program Flow

The following flow chart describes the top level program flow implemented in the bridge example project.

Figure 6-13. Top Level Program Flow



6.1.3.3 Code Details

The bridge application logic is implemented in *bridge.c* and *usb.c*. The main() function is located in the *bridge.c* file. For the enCoRe V bridge, these files are located at `<Install_Directory>\Cypress\CY3668 DVK\1.0\Firmware\Bridge\NL_Bridge_enCoreV\NL_Bridge_enCoreV\NL_Bridge_enCoreV`. For the enCoRe II bridge, these files are located at `<Install_Directory>\Cypress\CY3668 DVK\1.0\Firmware\Bridge\NL_Bridge_enCoreII\NL_Bridge_enCoreII\NL_Bridge_enCoreII`.

Note that the CY3668 DVK software is installed in the `<Install_Directory>`. For example, its typical location on a Windows 7 64-bit PC will be `C:\Program Files (x86)`.

The following table lists the functions, which implement the application logic. There is no difference in the implementation details of the enCoRe II and the enCoRe V bridge.

Table 6-3. Bridge Code Details

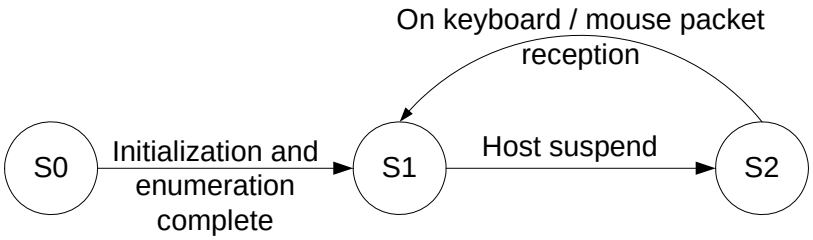
File	Functions
bridge.c – implements a unified bridge for both keyboard and mouse. This application acts as a bridge between the wireless device (i.e. mouse / keyboard) and the USB port of a PC / laptop.	void main(void) void CheckBindButton(void) void BridgeInit(void)
usb.c – implements USB reporting functions.	UINT8 handle_keyboard_report(void) UINT8 handle_mouse_report(void) void CheckUsbSuspend(void) void CheckUsbIdle(void)

Note that the only difference between the enCoRe II and enCore V firmware is that enCoRe II firmware does not display any content on the LCD panel.

6.1.3.4 *Bridge Firmware Implementation*

The following figure shows details of the different states the bridge firmware passes through.

Figure 6-14. Bridge Firmware Flow



S0: Initialization. On plugging into the USB port, the bridge initializes the different peripherals required. The different peripherals include USB, timers, and the WirelessUSB-NL radio. In the case of enCoRe V bridge, the LCD panel is also initialized and the message "WirelessUSB-NL Bridge" is displayed on it. The initialization is achieved by the BridgeInit function. The initialization function is a blocking function because of USB enumeration and LCD in the case of the enCoRe V bridge.

S1: Active. After the initialization is complete, the bridge checks if the USB bus is idle. This is implemented by CheckUsbIdle. The CheckUsbIdle checks for end point availability. If an end point is available for transaction, the bridge loads a packet received from any of the devices (mouse and keyboard) to the end point.

The next action is to listen for more radio packets from the mouse or keyboard. This is achieved through the MasterProtocolDataMode. This function is part of the AgileHID protocol and has call-backs that are implemented by the application layer. These call-back functions are called if a packet is received either from the mouse or the keyboard. The two call-back functions are handle_keyboard_report and handle_mouse_report. The handle_keyboard_report is called if a keyboard packet is received and the handle_mouse_report is called if a mouse packet is received. These functions extract data from the AgileHID radio packets and compile it to a USB HID packet. These packets are, in turn, reported through the USB via CheckUsbIdle.

After all the received radio packets are dispatched accordingly, the bridge checks for the bind button event (press and release of the bind button). On this event, the bridge goes into the bind mode by calling the AgileHID MasterProtocolButtonBindMode API. The bind process is handled by the CheckBindButton in the bridge code.

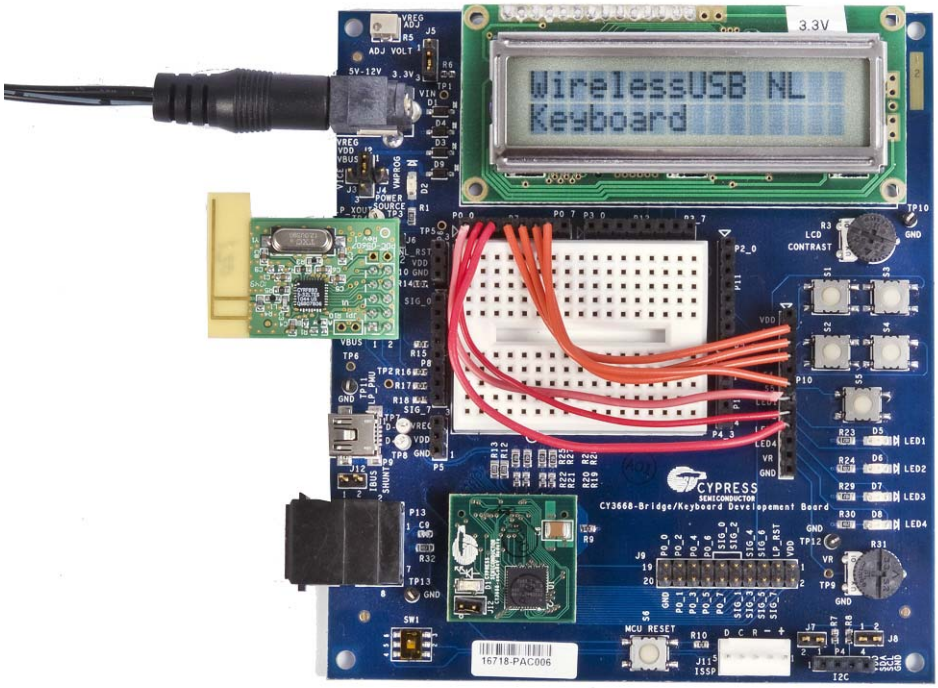
S2: Suspend. The firmware then checks USB_ACTIVITY_TIMEOUT_COUNTER. If the value of this counter is 0, then the firmware puts the device in USB suspend state by initiating sleep. This is implemented in CheckUsbSuspend. The value of the USB_ACTIVITY_TIMEOUT_COUNTER is set to 3 ms according to the USB specification, which requires devices to go to low-power state when no activity happens on the USB bus for 3 ms. When the bridge comes out of suspend due to the activity on the USB bus, the USB_ACTIVITY_TIMEOUT_COUNTER is re-initialized to 3 ms.

6.2 Setting up and Exercising Example Projects

6.2.1 Setting up WirelessUSB Keyboard

1. Open the WirelessUSB Example keyboard project in PSoC Designer. The project file is located at `<Install_Directory>\Cypress\CY3668 DVK\1.0\Firmware\Keyboard \NL_Keyboard_enCoreV\NL_Keyboard_enCoreV.app`.
2. Go to **Project > Settings > Compiler** to choose the applicable compiler.
3. Select **Build > Generate Configuration Files for 'NL_Keyboard_enCoReV' Project**.
4. Build the project. Select **Build > Build 'NL_Keyboard_enCoReV' Project** or press **[F7]**.
5. Because the enCoRe V module is used, position the SW1 switch slider away from the LCD panel (see enCoRe II/enCoRe V Programming Selector on page 19).
6. After the build process is successful, the output hex file is located at `<Install_Directory>\Cypress\CY3668 DVK\1.0\Firmware\Keyboard\ NL_Keyboard_enCoReV\NL_Keyboard_enCoReV\NL_Keyboard_enCoReV\output`.
7. Download the built firmware (hex file) on the CY3668 Development Board by following the Programming Steps on page 10.
   Note Make sure to remove the power source (12-V power adapter or USB cable) from the CY3668 Development Board before proceeding with the remaining steps.
8. Attach the WirelessUSB-NL Radio module to P2.
9. Wire up button S5 as a bind button. On the CY3668 DVK board, attach a wire from P0_6 on P7 to S5 on P10.
10. Wire up button S1 as Num Lock key, button S2 as Caps Lock key, and button S3 as Scroll Lock Key. On the CY3668 DVK board, attach wires from P0_3 on P7 to S1 on P10, from P0_4 on P7 to S2 on P10, and from P0_5 on P7 to S3 on P10 separately.
11. Wire up LED1 as Num Lock LED, LED2 as Caps Lock LED, and LED3 as Scroll Lock LED. On the CY3668 DVK board, attach wires from P0_0 on P7 to LED1 on P10, from P0_1 on P7 to LED2 on P10, and from P0_2 on P7 to LED3 on P10 separately.
12. Place a two-pin jumper on J10 for selecting the NL_RST pin to P2_4 for enCoRe V.
13. Place a two-pin jumper on J6 connecting VRADIO and VREG.
14. Place a two-pin jumper on J2 and J3 connecting VDD and VREG.
15. J5 is used to select the fixed 3.3 V VREG or adjustable VREG from 3.66 V to 1.63 V. In this example, it is suggested to use fixed 3.3 V VREG.
16. Connect the 12 V power adapter to P1.

Figure 6-15. WUSB-NL Keyboard Example



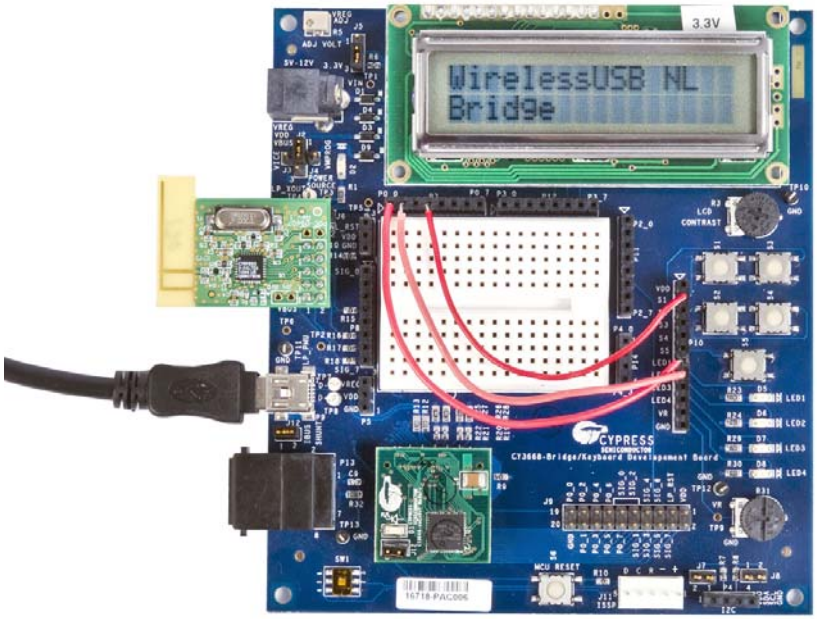
6.2.2 Setting up WirelessUSB Bridge

1. Open the NL_Bridge_enCoReV bridge project in PSoC Designer. The project file is located at `<Install_Directory>\Cypress\CY3668 DVK\1.0\Firmware\Bridge\NL_Bridge_enCoReV\NL_Bridge_enCoReV.app`. Note that `<Install_Diretory>` is the directory under which CY3668 DVK Software is installed during Installation process. For example, its typical value on a Windows 7 64 bit PC shall be `C:\Program Files (x86)`.

   Note Use NL_Bridge_enCoReII to test the bridge code for the enCoRe II device.
2. If enCoRe V module is used, position the SW1 switch slider away from the LCD panel (see enCoRe II/enCoRe V Programming Selector on page 19); for enCoRe II module, position SW1 switch slider towards the LCD panel.
3. Go to **Project > Settings > Compiler** and select the Imagecraft compiler.
4. Select **Build > Generate Configuration Files for 'NL_Bridge_enCoReV' Project**.
5. Build the project. Select **Build > Build 'NL_Bridge_enCoReV' Project** or press **[F7]**.
6. After the build process is successful, the output hex file is located at `<Install_Directory>\Cypress\CY3668 DVK\1.0\Firmware\Bridge\NL_Bridge_enCoReV\NL_Bridge_enCoReV\ NL_Bridge_enCoReV\output`.
7. Download the built firmware (hex file) on the CY3668 Development Board by following the Programming Steps on page 10.
   Note Make sure to remove the power source (12-V power adapter or USB cable) from the CY3668 Development Board before proceeding with the remaining steps.
8. Attach the WirelessUSB-NL Radio module to P2.
9. Wire up button S1 as Bind button. On the CY3668 DVK board, attach a wire from P0_3 on P7 to S1 on P10.
10. Wire up LED1 and LED2. On the CY3668 DVK board, attach wires from P0_0 on P7 to LED1 on P10 and from P0_1 on P7 to LED2 on P10 separately.

11. Place a two-pin jumper on J12.
12. Place a two-pin jumper on J10 for selecting the NL_RST pin to P2_4 for enCoRe V. If an enCoRe II module is used for testing, **do not** place this jumper; instead use an external wire between P0_7 and NL_RST (in P6).
13. Place a two-pin jumper on J6 connecting VRADIO and VREG.
14. Place a two-pin jumper on J2 and J3 connecting VDD and VREG.
15. J5 is used to select the fixed 3.3 V VREG or adjustable VREG from 3.66 V to 1.63 V. In this example, it is suggested to use a fixed 3.3 V VREG.
16. Connect the mini-B side of the USB A/Mini-B cable to the CY3668 DVK board.
17. Connect the A side of the USB A/Mini-B cable to the PC.
18. The NL_Bridge_enCoReV bridge should enumerate on the PC as a composite USB device supporting both keyboard and mouse.

Figure 6-16. WUSB-NL Bridge Example



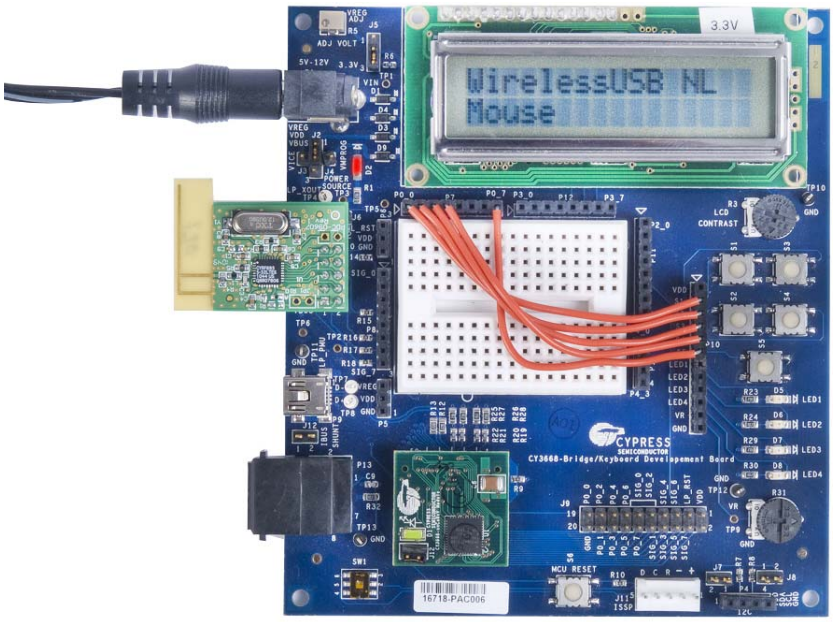
6.2.3 Exercising WirelessUSB Keyboard Example Project

1. Press S1 on the Development Board running the Bridge firmware to change to bind mode. LED1 on the Bridge blinks to denote bind mode. Press S5 on the Development board running Keyboard firmware to change to bind mode; wait for 2 seconds. On successful binding, LED1 on the Bridge switches off and the wireless link is now ready for data transfer.
2. On the Development Board running Keyboard firmware, press S1 for Num Lock, S2 for Caps Lock, and S3 for Scroll Lock. The actions corresponding to the pressed key is executed on the PC.

6.2.4 Setting up the WirelessUSB Mouse

1. Open the WirelessUSB example mouse project in PSoC Designer. The project file is located at `<Install_Directory>\Cypress\CY3668 DVK\1.0\Firmware\Mouse\NL_Mouse_enCoReV\NL_Mouse_enCoReV.app`.
2. Go to **Project > Settings > Compiler** to choose the applicable compiler.
3. Select **Build > Generate Configuration Files for 'NL_Mouse_enCoReV' Project**.
4. Build the project. Select **Build > Build 'NL_Mouse_enCoReV' Project** or press **[F7]**.
5. Because enCoRe V module is used, position the SW1 switch slider away from the LCD panel (see enCoRe II/enCoRe V Programming Selector on page 19).
6. After the build process is successful, the output hex file is located at `<Install_Directory>\Cypress\CY3668 DVK\1.0\Firmware\Mouse\NL_Mouse_enCoReV\NL_Mouse_enCoReV\ NL_Mouse_enCoReV\output`.
7. Download the built firmware (hex file) on the CY3668 Development Board by following the Programming Steps on page 10.
   Note Make sure to remove the power source (12-V power adapter or USB cable) from the CY3668 Development Board before proceeding with the remaining steps.
8. Attach the WirelessUSB-NL Radio module to P2.
9. Wire up S5 as the Bind Button. On the CY3668 board, connect a wire between P0[6] in P7 and S5 on P10.
10. Wire up S1 as the left button of the mouse, S2 as the right button, S3 as the middle button, and S4 as the movement button on the CY3668 board.
    a. Connect a wire from P0[0] in P7 to S1 in P10 for left button
    b. Connect a wire from P0[1] in P7 to S2 in P10 for right button
    c. Connect a wire from P0[2] in P7 to S3 in P10 for middle button
    d. Connect a wire from P0[3] in P7 to S4 in P10 for movement button
11. Place a two-pin jumper on J6 between pins 2 and 3.
12. Place a two-pin jumper on J2 and J3 connecting VDD and VREG.
13. J5 is used to select the fixed 3.3 V VREG or adjustable VREG from 3.66 V to 1.63 V. In this example, it is suggested to use fixed 3.3 V VREG.
14. Connect the 12-V power adapter to P1.

Figure 6-17. WUSB-NL Mouse Example



6.2.5 Exercising WirelessUSB Mouse Example Project

1. Press S1 on the Bridge to change to bind mode. LED1 on the Bridge blinks to denote bind mode. Press S5 on the Development Board running the Mouse firmware to change to bind mode; wait for 2 seconds. On successful binding, LED1 on the Bridge switches off and the wireless link is now ready for data transfer.
2. On the Development Board running Mouse firmware, press S1 for Left Button Click, S2 for Right Button Click, and S3 for Middle Button Click. The actions corresponding to the pressed key is executed on the computer. Press S4 to send simulated circular optical sensor movement to the PC. This will result in mouse pointer moving in circular fashion on the PC.

7. Troubleshooting

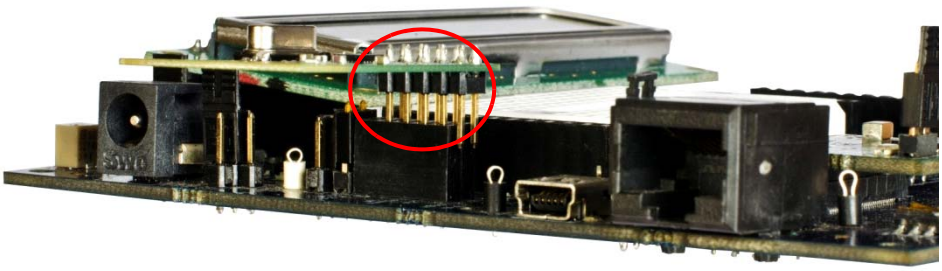
1. Issue: Unable to download firmware to enCoRe II module after plugging it on the development board.

Fix: Switch SW1 determines the MCU type (enCoRe V LV/enCoRe II) to be used on a given development board. To select enCoRe II, ensure that the switch is pushed towards the LCD panel. See enCoRe II/enCoRe V Programming Selector on page 19 for details.

2. Issue: Firmware does not come up after plugging in the WUSB-NL module.

Fix: Make sure all 10 pins of the WUSB-NL module are plugged on to the 10 pins of the socket located on the development board and are properly aligned.

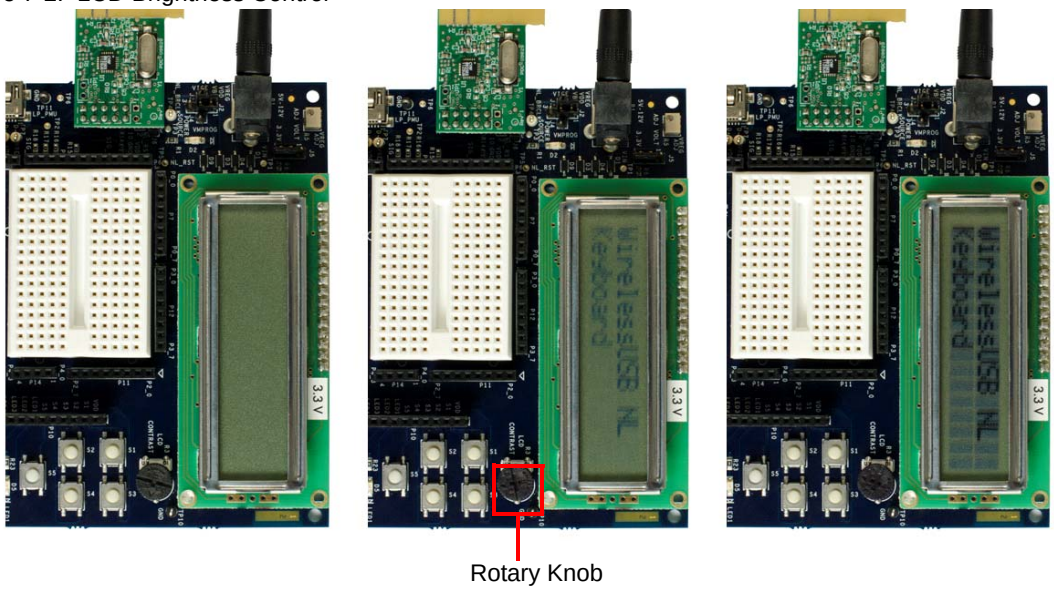
Figure 7-1. WirelessUSB-NL Module Plugged-in Incorrectly



3. Issue: Text displayed on the LCD display area is not clear or is not visible.

Fix: The rotary knob located below the LCD controls the contrast of the LCD display. Turn the knob in the clockwise direction to increase the contrast of the LCD screen. Set the contrast to a level of your choice.

Figure 7-2. LCD Brightness Control



4. Issue: Development board is not powered using the USB cable.

Fix: Position of jumpers at J2, J3, and J4 determine the power source mechanism for the development board. To power using the USB cable, short pins 2 and 3 of J2 and pins 1 and 2 of J12. See MCU Power Selection on page 16 for details.

5. Issue: Not able to send data from Keyboard Development Board to Bridge Development Board.

Fix: For data transfer to happen, the Keyboard should successfully bound to the Bridge. If the data transfer does not happen, the binding process may not be successful. Please follow this procedure to bind to the bridge.

- Place Bridge Development Board and Keyboard Development Board close to each other (within one meter distance)
- Power On the Bridge and the Keyboard
- Press **S1** button on Bridge. Confirm that LED1 on Bridge is blinking continuously.
- Press **S5** button on Keyboard.
- Wait for LED1 on Bridge to go off in around 2 seconds.
- Keyboard is now successfully bound to the Bridge.

If the LED1 on the Bridge does not go off in a few seconds, repeat this procedure.