CY3668

WirelessUSB™ NL Development Kit Guide

Doc. # 001-76173 Rev. **

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

Introduction

This document provides an understanding of enCoRe™ V, enCoRe II, and WirelessUSB™ NL technology using the CY3668 WirelessUSB NL Development Kit (DVK).

1.1 Kit Contents

The CY3668 Development Kit includes:

- CY3668 bridge/keyboard development boards (2)
- NL modules (2)
- CY3668- enCoRe II module (1)
- CY3668-enCoRe V modules (2)
- 3.3-V LCD (2)
- Power adaptor (2)
- Quick Start Guide
- Resource CD

1.2 PSoC Designer 5.1

Cypress's PSoC Designer is a 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, technical reference manual, and application notes.

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

2. Installation

2.1 Install Software

This section describes how to install the different software tools required to program the CY3668 WirelessUSB NL DVK.

2.1.1 PSoC Designer Installation

Install PSoC Designer 5.1 setup software from the CY3668 DVK CD-ROM.

2.1.2 PSoC Programmer Installation

Install PSoC Programmer setup software from the CY3668 DVK CD-ROM.

Note To download the latest versions of PSoC Designer and PSoC Programmer software, go to http://www.cypress.com.

2.1.3 CY3668 DVK Software Installation

Install the CY3668 WirelessUSB NL DVK software (*Install_CY3668_WirelessUSB NL DVK.exe*) from the kit CD. You can also download this software from http://www.cypress.com/go/CY3668.

2.2 Install Hardware

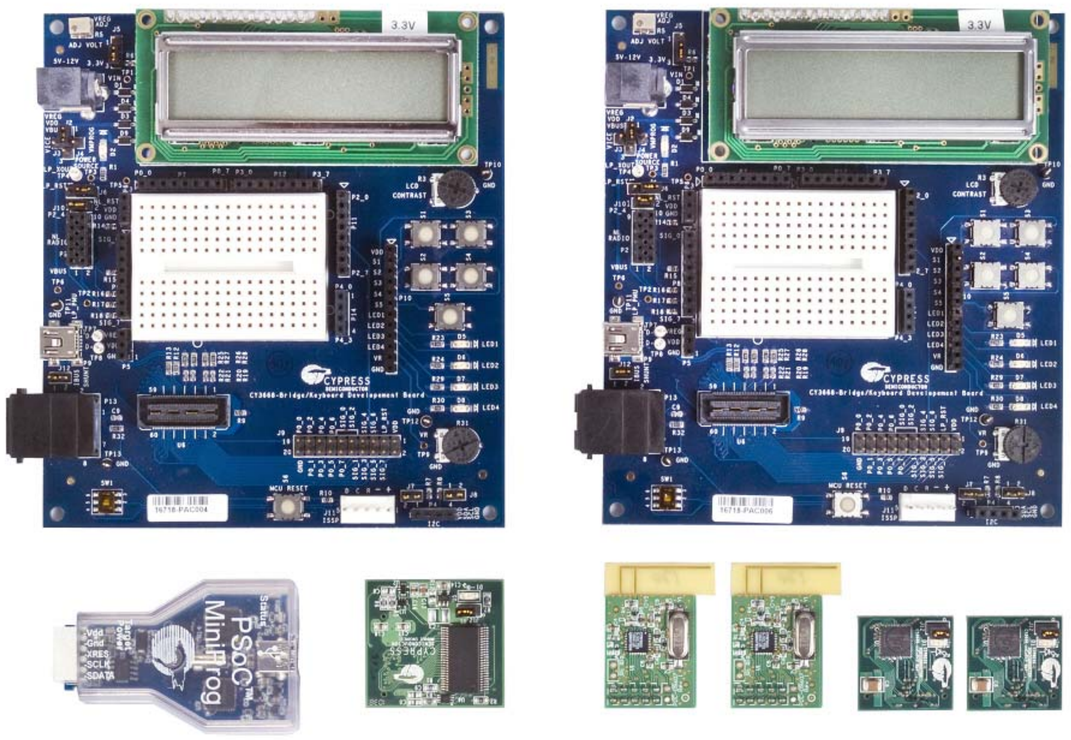
WARNING: Static discharges from the human body can easily reach 20,000 volts. This can damage the components on the DVK. Take precautions to ensure that any static is discharged before touching the hardware. Ensure the DVK is powered off before making any changes to the connections.

2.2.1 Default Jumper Settings

The following are the default jumper settings for the CY3668 board:

- J1, place a jumper
- J3, place a jumper between VREG and VDD
- J5, place a jumper between Pin 2 and 3.3 V
- J12, place a jumper
- J13, place a jumper
- Leave other jumpers open

Figure 2-1. CY3668 Boards



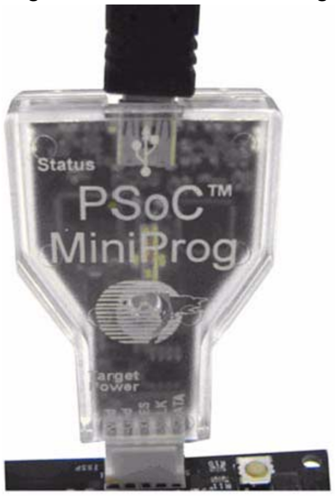
2.3 Programming and Emulation Procedures

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).

2.3.1 Programming Steps

1. If the NL Radio Module is connected to the NL radio interface connector (P2), remove it before the next step to avoid applying 5 V to this module.
2. Connect one end of the USB cable to the PSoC MiniProg and the other end to the PC.
3. Connect the PSoC MiniProg to the 5-pin ISSP header on the CY3668 DVK board.

Figure 2-2. PSoC MiniProg

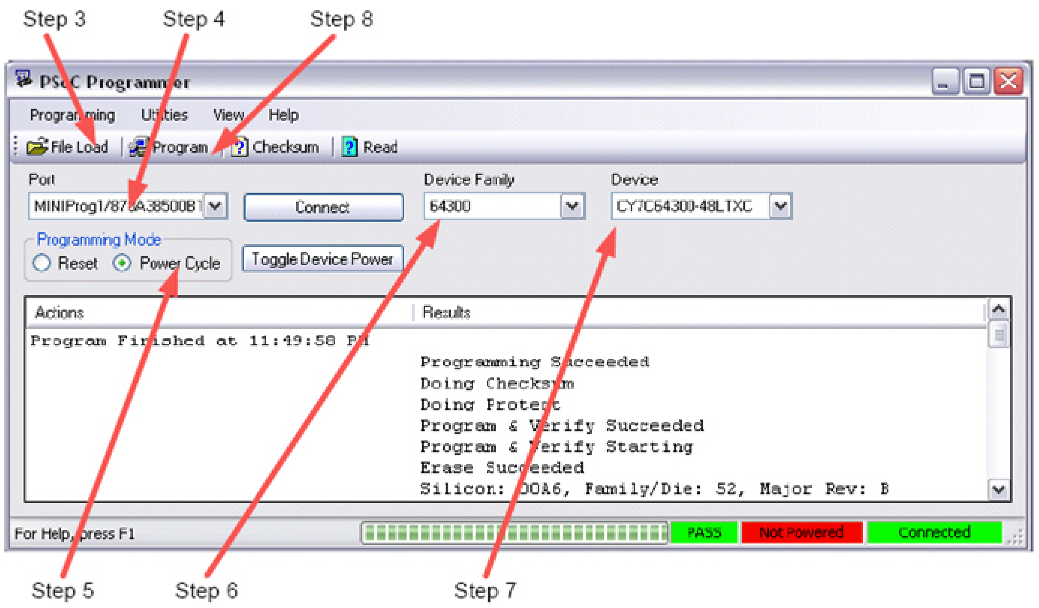


4. Launch PSoC Programmer version 3.00.0.92 or later.
5. Click **Programming > File Load** to load the desired binary (hex) file.
6. Select the MiniProg port. If the Actions column in the PSoC Programmer status box indicates "Firmware update required at ...", the MiniProg firmware is outdated.

   Before attempting to program the target PSoC chip, go to **Utilities > Upgrade Firmware** to update the MiniProg firmware.
7. For Programming Mode, select **Power Cycle**.

   Ensure the AC adapter is removed from P1 and the USB cable is unplugged from P9.
8. From the Device Family menu, select **64300**.
9. From the Device menu, select **CY7C64300-48LTXCT**.
10. Click **Program**. PSoC Programmer goes through programming and verification modes.
11. When programming is complete, remove the MiniProg from the CY3668 DVK board.

Figure 2-3. PSoC Programmer



Port: MINIProg1/873A38500B1

Device Family: 64300

Device: CY7C64300-48LTXC

Programming Mode: ( ) Reset [✓ Power Cycle]

Actions	Results
Program Finished at 11:49:58 PM	
	Programming Succeeded
	Doing Checksum
	Doing Protect
	Program & Verify Succeeded
	Program & Verify Starting
	Erase Succeeded
	Silicon: 00A6, Family/Die: 52, Major Rev: B

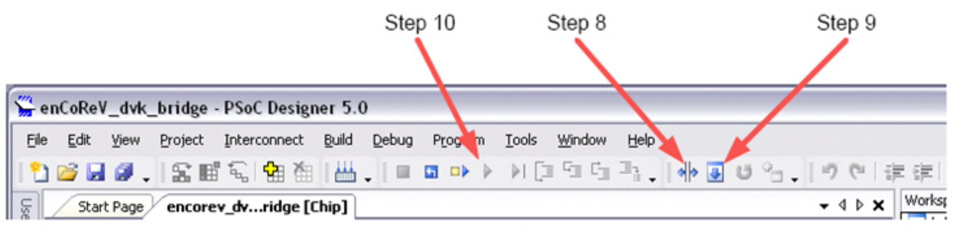
For Help, press F1 | PASS | Not Powered | Connected

2.3.2 Emulation Steps

1. Launch the desired .APP project file project from `C:\Cypress\CY3668 DVK\Example\<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 blue 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.

   NL radio voltage can thus meet the requirement.
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.

Figure 2-4. PSoC Designer Toolbar



3. WirelessUSB Protocol 2.2

3.1 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 NL Radio Driver API document that comes along with this kit for protocol requirement.

The Enhanced AgileHID protocol provides a set of APIs to develop wireless mouse, keyboard, generic devices, and bridge 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).

3.2 Protocol Functions

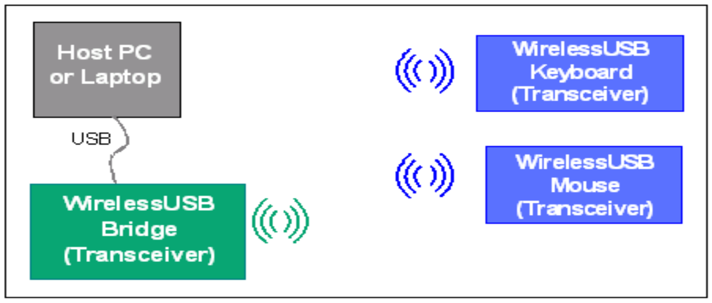
The following functionalities are visible to the application:

- 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 3-1. System Level Block Diagram


Host PC or Laptop
USB
WirelessUSB Bridge (Transceiver)
WirelessUSB Keyboard (Transceiver)
WirelessUSB Mouse (Transceiver)

3.2.1 Protocol API Use

The protocol is divided into two modules: master protocol on the bridge side and slave protocol on the device (keyboard/mouse) side. Only one of the modules can be present in a subsystem (either slave or master protocol). The protocol module is selected using 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.

3.2.1.1 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, then the bridge goes to the radio packet receive mode periodically. After a packet is received from a device, the application wakes up USB.

3.2.1.2 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. 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.

3.2.2 Requirements

3.2.2.1 Header Files

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

3.2.2.2 SW Interface

The Enhanced AgileHID Protocol runs on Cypress's 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, so the user only needs to focus on the application.

3.2.3 Type Declarations and Definitions

3.2.3.1 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
```

3.2.4 Protocol High Level Functions

3.2.4.1 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.

3.2.4.2 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.

3.2.4.3 MasterProtocolButtonBindMode

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

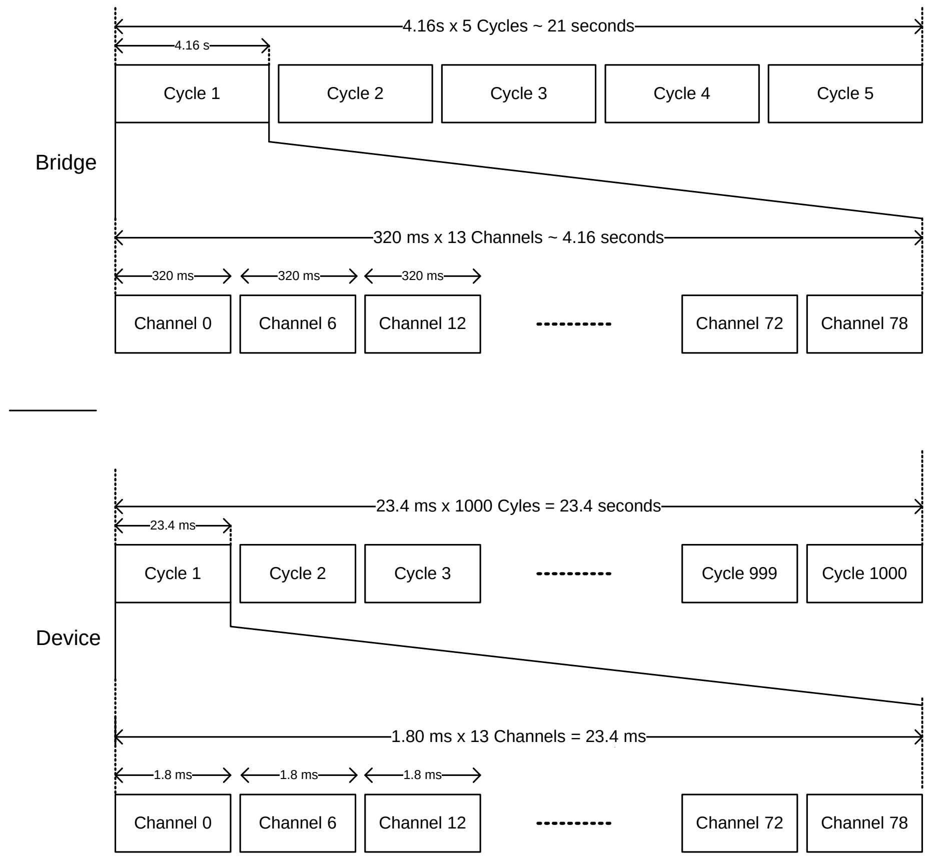
Parameters: unsigned short retry_count - Number of times the bridge changes the channel before it times out.

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 the following diagram. 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 of 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 3-2. Bridge Protocol Polls for Bind Request



3.2.4.4 *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 device.

3.2.4.5 *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.

3.2.4.6 *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.

3.2.4.7 *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.

3.2.4.8 *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.

3.2.4.9 *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 14.

3.2.4.10 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.

3.2.4.11 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[]).

3.3 Application Packet Format

3.3.1 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 do not show 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 2 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.

3.3.2 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 do not show the protocol packet format.

3.3.2.1 *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

3.3.2.2 *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)

3.3.2.3 *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

3.3.2.4 *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	

3.3.2.5 *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.

3.3.3 Sample Firmware Source Code

3.3.3.1 Bridge Source Code

Main Loop for Bridge application

```
void main(void)
{

   // Initialize the bridge.
   BridgeInit();

   while (1)
   {
      //Clear Watchdog and Sleep
      M8C_ClearWDTAndSleep;
      // Check USB IDLE timer
      CheckUsbIdle();
      // Check for Received Packets
      MasterProtocolDataMode();
      // Check for Bind Button
      // If bind button pressed
{
         MasterProtocolButtonBindMode(MASTER_BUTTON_BIND_RETRY_COUNT);
}
      // Check USB suspend
CheckUsbSuspend();
   }
}
```

Bridge Initialization function

```
void BridgeInit(void)
{

    // Initialize tick timer
    TimerInit();
    // Start the SPI. Below configuration is for NL radio
     SPIM_Radio_Start(SPIM_Radio_SPIM_MODE_1 | SPIM_Radio_SPIM_MSB_FIRST);
    // Start USB
    USB_1_Start(0x0);
    // Enable global interrupt
    M8C_EnableGInt;
    // Wait for USB enumeration
    while (USB_1_bGetConfiguration() == 0)
    {
        //Clear Watchdog and Sleep
        M8C_ClearWDTAndSleep;
     }
    // Initialize Protocol
    MasterProtocolInit();
}
```

3.3.3.2 Slave (Device) Source Code

Main loop function for device

```
void main(void)
{
// Do initialization for the entire device.
DeviceInit();

// Enter the main loop and be there forever.
while(1)
    {
        //Clear Watchdog and Sleep
        M8C_ClearWDTAndSleep;
        // If any packet to transmit
        {
      //Mouse packet is considered just for example
            radio_pkt->shared.optical.x = 0;
            radio_pkt->shared.optical.y = 0;
            SlaveProtocolSendPacket(MOUSE_DEVICE_TYPE, MOUSE_DEVICE_TYPE, 2);
        }
        // else if system is idle for long time
        {
            // Force radio to sleep state.
            RadioForceState(END_STATE_SLEEP);
                // Keep system is sleep mode and wakeup on interrupt
            // System coming out of sleep mode Force radio to idle state.
            RadioForceState(END_STATE_IDLE);

        }
        // Check for Bind Button
        // If bind button pressed
        {
            SlaveProtocolButtonBindMode(MASTER_BUTTON_BIND_RETRY_COUNT);
        }

    }
}
Device Init function
static void MouseInit(void)
{
    // Initialize the port

// Initialize the ISR
       // Initilize the Timer

       // Start the SPI
       SPIM_Radio_Start(SPIM_Radio_SPIM_MODE_1 | SPIM_Radio_SPIM_MSB_FIRST);

// Initialize the protocol layer.
       SlaveProtocolInit();
   }
```

4. Code Examples

The code examples that come with this kit demonstrate the basic functionality of a WirelessUSB HID keyboard and mouse. The following sections describe how to test the wireless keyboard application. Two development boards with NL radio modules are used in this example. One board is used as a bridge, which will enumerate on the PC as a bus-powered USB composite device having two interfaces: keyboard and mouse. The other board is used as a keyboard, which will simulate a simple keyboard with three keys (Number lock, Capitals lock, and Scroll lock) and three LEDs (Num lock LED, Caps lock LED, and Scroll lock LED).

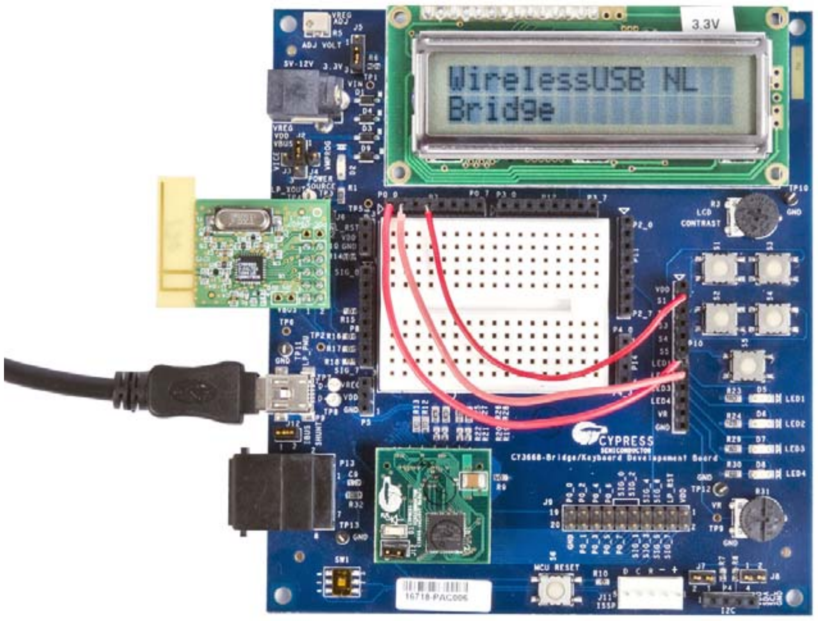
4.1 WirelessUSB Bridge

1. Open the NL_Bridge_enCoReV bridge project in PSoC Designer. The project file is located at `C:\Program Files\Cypress\CY3668 DVK\Firmware\Bridge\NL_Bridge_enCoReV\NL_Bridge_enCoReV.app`.

   Note Use NL_Bridge_enCoReII to test the bridge code for the enCoRe II device.
2. Go to **Project > Settings > Compiler** and select the Imagecraft compiler.
3. Select **Build > Generate Configuration Files for 'NL_Bridge_enCoReV' Project**.
4. Build the project. Select **Build > Build 'NL_Bridge_enCoReV' Project** or press **[F7]**.
5. If enCoRe V module is used, turn the SW1 switch downwards; for enCoRe II module, turn SW1 position upwards.
6. Program the CY3668 DVK board with the hex file located at `C:\Program Files\Cypress\CY3668 DVK\Firmware\Bridge\NL_Bridge_enCoReV\NL_Bridge_enCoReV\output\`.
7. Select **Program > PSoC Programmer** in PSoC Designer and follow the Programming Steps on page 8.
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 enCoRe II module is used for testing **do not** place this jumper; instead use 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 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 4-1. WUSBNL Bridge Example



4.1.1 Results for Bind

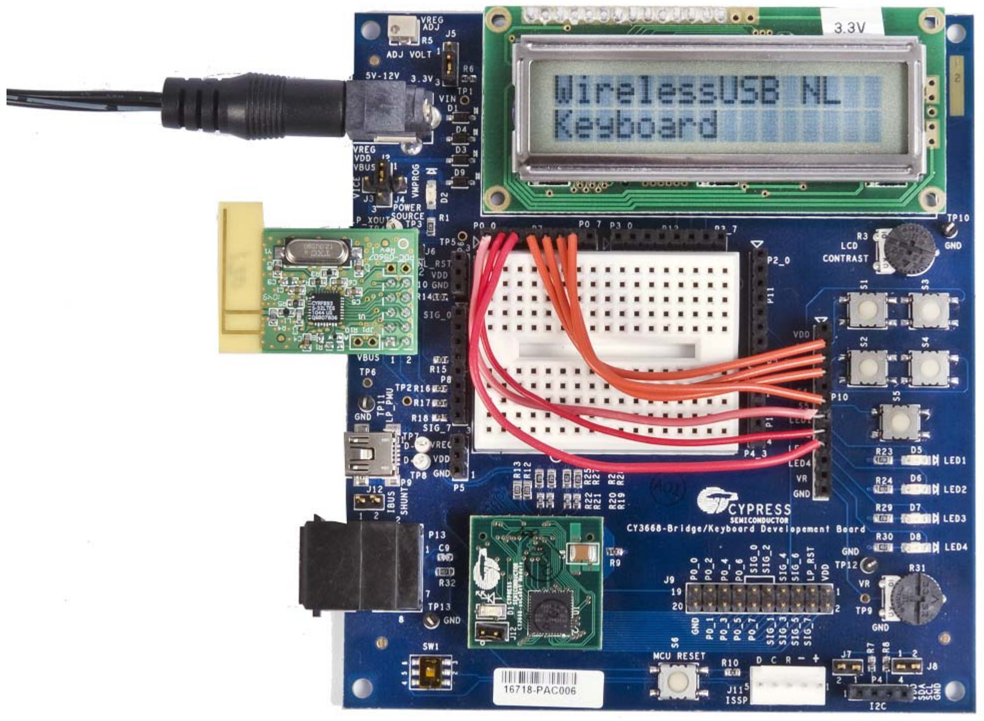
After the bridge is enumerated and the LEDs are off, press the Bind button S1 on the bridge and the Bind button S5 on the keyboard or S5 on the mouse to complete the 'binding' process. While in the bind mode, the LED2 on the bridge blinks until the Bind button is pushed on the keyboard or mouse. The bridge LED2 stops blinking to indicate the keyboard or mouse is now ready for normal operation. Note that the bridge LED2 will also stop blinking after 20 seconds timeout without receiving any bind request from the keyboard or mouse.

4.2 WirelessUSB Keyboard

1. Open the WirelessUSB Example keyboard project in PSoC Designer. The project file is located at `C:\Program Files\Cypress\CY3668 DVK\Firmware\Keyboard\NL_Keyboard_enCoReV\ NL_Keyboard_enCoReV`.
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. Turn the SW1 switch downwards, as enCoRe V module is used.
6. Program the CY3668 DVK board with the hex file located at `C:\Program Files\Cypress\CY3668DVK\Firmware\Keyboard\NL_Keyboard_enCoReV\NL_Keyboard_enCoReV\output\`.
7. Select **Program > PSoC Programmer** in PSoC Designer and follow the Programming Steps on page 8.
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 4-2. WUSBNL Keyboard Example



4.2.1 Results for Keyboard

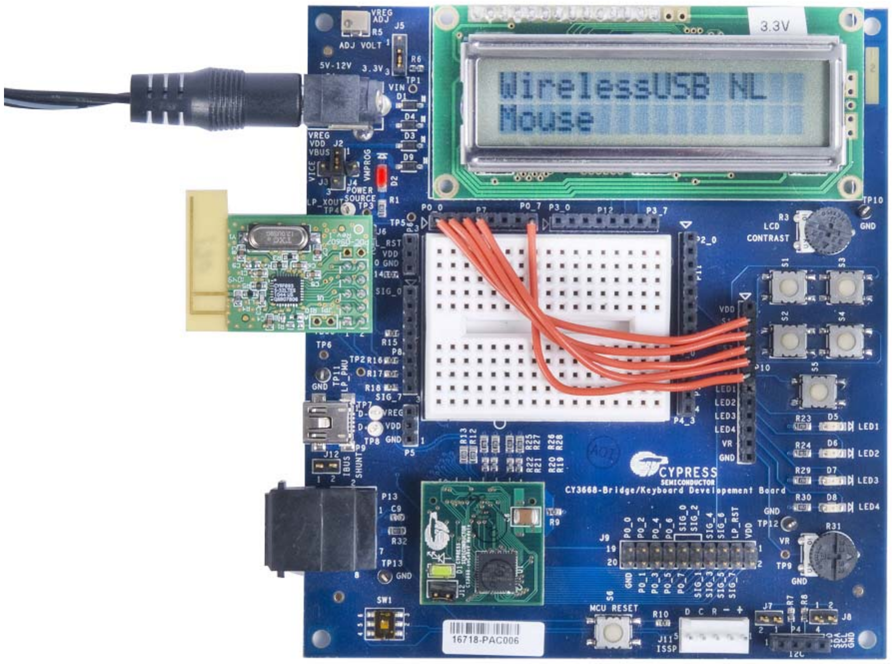
Press buttons on the keyboard that represents Num Lock, Caps Lock, and Scroll Lock. The keystroke data is sent to the bridge, which will transfer this data to the PC. After receiving the data from the bridge, the PC will process the Num Lock, Caps Lock, and Scroll Lock data and return the LED information to the bridge and then the keyboard. Finally, the keyboard device turns on/off the corresponding LEDs.

4.3 WirelessUSB Mouse

The WirelessUSB mouse example code simulates mouse movement in a circular fashion and implements left click, right click, and middle button click on the development board.

1. Open the WirelessUSB example mouse project in PSoC Designer. The project file is located at `C:\Program Files\Cypress\CY3668 DVK\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. Turn the SW1 switch downwards, as enCoRe V module is used.
6. Program the CY3668 DVK board with the hex file located at at `C:\Program Files\Cypress\CY3668 DVK\Firmware\Mouse\NL_Mouse_enCoReV\NL_Mouse_enCoReV\output\`.
7. Select **Program > PSoC Programmer** in PSoC Designer and follow the Programming Steps on page 8.
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 connecting VRADIO and VREG.
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 4-3. WUSBNL Mouse Example



4.3.1 Results for Mouse

Press buttons on the mouse that represents the left button, right button, and middle button. The keystroke data is sent to the bridge, which will transfer this data to the PC. If you press the movement button (S4), the optical navigation is simulated and the mouse pointer in the PC starts moving in a circular fashion.