CY4632 Bridge Firmware
User's Guide

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

October 1, 2004

CY4632 Keyboard Firmware User's Guide
Table of Contents

Table Listings

Figure Listings

1. INTRODUCTION

WirelessUSB™ LS is the ideal solution for a low-cost, low-power wireless device. The purpose of the WirelessUSB LS RDK is to provide a reference design wireless keyboard and mouse implementation. This document gives an overview of the firmware for the enCoRe™-based USB bridge and describes the steps needed to compile the firmware.

The WirelessUSB LS RDK supports the following features:

- Supports a Wireless USB LS keyboard and mouse
- Supports USB certification with the USBCV test application
- 64 kbps data rate
- 2-way communication
- Automatic bind procedure

1.1 Background Information

This document assumes the reader to be familiar with the general operation of WirelessUSB LS and USB HID devices. For more information on WirelessUSB LS please refer to ***WirelessUSB Theory of Operation*** and ***WirelessUSB LS 2-Way HID Systems*** application notes.

2. DEFINITIONS

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

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

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

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

Encore – The CY7C63743-PC chip that serves as the CPU for the bridge.

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

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

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

WirelessUSB™ – a trademark name for Cypress 2.4 GHz radio products.

3. LS HID BRIDGE

The WirelessUSB LS HID Bridge is provided with the RDK. This bridge may be plugged into the USB port on a PC to provide the Wireless USB bridge functionality. The bridge firmware runs on an enCoRe™ CY7C63743-PC chip, is written in assembly, and runs on the PDC-9168 USB HID Bridge. The rest of this section gives a functional overview of the bridge firmware.

The bridge connects the remote WirelessUSB LS peripheral to a low-speed USB host. This firmware supports 2-way communication with bridge and HID devices configured as transceivers.

The RDK also includes a WirelessUSB LS keyboard and WirelessUSB LS mouse hardware. Standard USB HID packets are encapsulated inside WirelessUSB LS packets, which also contain a packet header and checksum to help the bridge correctly process the USB HID data packets. Valid packets are then sent via USB to the USB host.

3.1 Bridge Photographs

Figure 1 shows the topside of the RDK Bridge board. The button on the lower left of the board is the “bind” button.

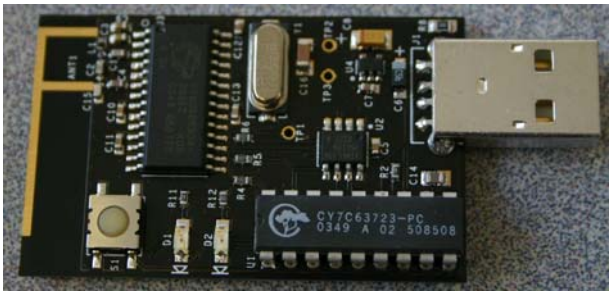

Figure 1: RDK Bridge Top

Figure 2 shows the bottom side of RDK Bridge board.

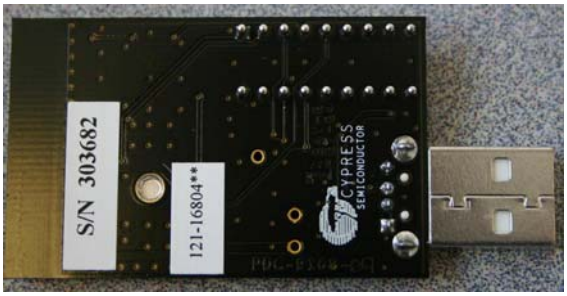

Figure 2: RDK Bridge Bottom

3.2 File Descriptions

The USB HID Bridge firmware is contained in the "**Firmware\Source Code\RDK Bridge"** directory on the CD. The purpose(s) of each file are shown in the following table:

Table 1: Source File Descriptions

Bridge Firmware	
Filename	**Purpose**
config.inc	Include file containing all user-configurable options
wusb-ls-headers.inc	Include file containing application specific items
637xx.inc	Include file containing chip specific items
usb.inc	Include file containing USB specific items
USBcode.asm	Assembly file containing source code for USB functionality
rdk_kbm desc.inc	Include file containing descriptor tables for USB HID keyboard/mouse combo
rdk_keyboard.asm	Assembly file containing source code for keyboard
rdk_mouse.asm	Assembly file containing source code for mouse
radio.inc	Include file containing radio specific items
radio.asm	Assembly file containing source code for controlling the WirelessUSB LS radio
wusb-ls-main.asm	Assembly file containing protocol source code
bind-auto.asm	Assembly file containing the auto bind procedure
dvk_hardware.asm	Assembly file containing bind button checking and LED routines.
E2.asm	Assembly file containing functions for accessing non-volatile memory.
utilities.asm	Assembly file containing helper functions for receiving data, verifying packets, transmitting packets, etc. (2-way only)
mfgtest.asm	Assembly file containing the manufacturing test source code
encrypt.asm	Assembly file containing encryption functions not included on the CD. Contact Cypress Application support for source code.
encrypt_support.asm	Assembly file containing encryption helper functions not included on the CD. Contact Cypress Application support for source code.

3.3 Compiling the Bridge Firmware

CYASM.EXE version 1.96 or higher is required to compile the bridge firmware. To compile type the following at a command prompt:

cyasm wusb-ls-main.asm

The CY7C63000 device programmer can be used to program enCoRe chips. The CY3083-07 adapter board is required. The CY3654 Developer Kit and CY3654-PO5 Personality Board are required to emulate the enCoRe (CY7C63743-PC). Technical and ordering information can be found at: http://www.cypress.com.

3.4 Bridge Radio Firmware Details

3.4.1 reset

On reset, the firmware initializes the radio and then waits until the USB host enumerates the device.

3.4.2 radio_init

This routine brings the radio out of reset and waits for the radio to be ready.

3.4.3 load_pn_code

This routine loads the PN Code from ROM into the radio registers.

3.4.4 setup_rx

Puts the radio into receive mode.

3.4.5 setup_tx

Puts the radio into transmit mode.

3.4.6 endpoint0

USB control endpoint handler. This interrupt handler formulates responses to USB SETUP and CONTROL transactions.

3.4.7 endpoint1

This interrupt routine handles the reserved data endpoint 1 (for a mouse). This interrupt happens every time a host sends an IN on endpoint 1. The data to send (NAK or data packet) is already loaded, so this routine just prepares the DMA buffers for the next packet.

3.4.8 endpoint2

3.4.9 power_on_mode (auto bind only)

power_on_mode initializes the channel and PN Code(s) and jumps to **ping_mode**.

3.4.10 ping_mode (auto bind only)

ping_mode implements ping mode as described in ***WirelessUSB LS 2-Way HID Systems*** application note. Upon finding an available channel the bridge will jump to **idle_mode**. If the bind button is pressed while in ping mode the bridge will jump to **bind_mode**.

3.4.11 idle_mode (auto bind only)

idle_mode implements idle mode as described in ***WirelessUSB LS 2-Way HID Systems*** application note. After establishing a connection the bridge will jump to **connected_mode**. If the bind button is pressed while in idle mode the bridge will jump to **bind_mode**.

3.4.12 bind_mode (auto bind only)

bind_mode implements bind mode as described in ***WirelessUSB LS 2-Way HID Systems*** application note. After binding with a HID (or timing out) the bridge will jump to **ping_mode**.

3.4.13 connected_mode

connected_mode implements connected mode as described in ***WirelessUSB LS 2-Way HID Systems*** application note.

When a data packet is received from a WirelessUSB device, the packet is processed and the appropriate HID report is submitted to the USB by process _data routine.

3.4.14 process_data

This routine checks for retransmitted packets (and discards the packet if it is a retransmission). If the packet contains new data it is loaded into the USB DMA buffer (**app_data_received_a/b routines** send the data to the USB host when it is polled for data). Receipt of the data from the device is then acknowledged via an ACK or an ACK/DATA packet.

3.4.15 process_rx_int

This routine is called when an interrupt occurs while the radio is in receive mode. The values of the valid and data registers are read and stored. If the end of frame (EOF) is reached a flag is set.

3.4.16 verify_packet

The parity is computed and compared to the received parity. If the parity is correct the valid field is then used to fix up to eight bit errors (one bit per bit position) using the valid bytes, received data and the checksum field. If more than one bit per bit position is invalid the packet is marked corrupt. After the packet has been fixed (if necessary) the checksum is calculated and compared to the received checksum. If the checksum is correct the packet is marked as valid.

3.4.17 transmit_sys

transmit_sys handles the transmission of all non-application packets such as BIND RESPONSE, CONNECT RESPONSE, PING and ACK packets.

3.4.18 transmit_app

transmit_app handles the transmission of all application packets such as DATA and ACK/DATA packets.

3.5 Bridge Application Firmware Details

The bridge includes application related functionality to operate with the wireless keyboard and mouse. This includes code to service keyboard and mouse reports, code to implement UpKey functionality, and EEPROM related functionality.

Keyboard and mouse reports are generated by the HID devices, and sent over the air to the radio. In order to minimize the amount of data to be sent over the air, the HID devices use variable length

packets so that only the necessary bytes are sent. The enCoRe then process these reports by expanding them to the packet size defined in the HID report descriptors and sends them to the host PC.

The UpKey functionality provides an UpKey (break, or null key) to the host PC in the event that the HID device looses its connection or power and fails to provide the required upkey. When a device has transmitted a downkey (make key), the host PC will normally repeat outputting that key until it receives the UpKey (break key). With wireless devices, this can cause a problem if the wireless device fails to provide the UpKey. The bridge firmware provides this functionality in the event that the wireless device looses the current connection.

The EEPROM related code provides non-volatile storage to the bridge.

3.5.1 Keyboard Application Specific Code

3.5.1.1 app_data_received_a

Input:
X Reg = ByteLenRept = Byte length of input report (-2 for hdr, chksum)
A Reg = NullPktDataBits = additional info
0x00 => UpKey (UpKeyStandardKeys only)
0x02 => KeepAlive
Keyboard Report bytes in byte_buffer[], app_buffer[]

Background:
byte_buffer[] contains the input bytes from the radio including the header.
app_buffer[] points to the first data byte.
app_buffer is currently 1 byte offset from byte_buffer to skip the 1 byte header, but app_buffer[] should be used to allow changes in the length of the header (i.e. to allow it to use a two byte header).

Note: hdr=HeaderByte sc=ScanCode mod=modifier chk=checksum res=reserved
ReptType = report type, ConsumerKeys = Multimedia Keys

```
Input Examples: (character "a" = "0x04", see Hid Usage Tables)
byte_buffer[]= 40 04 ; "a"
    app_buffer[]= 04 ; "a"
```

```
More app_buffer examples:
 ReptType/sc1 ; Scan Code 1 or RptType ConsumerKeys=0xFF PowerKeys=0xFE
  |  mod       ; Modifier keys (in StandardKeys reports only)
  |  |  sc2    ; Scan Code 2 ...
  |  |  |
04             ; "a"
04 20          ; "<RightShift>a"
04 00 20       ; "a" and "3"
04 20 20       ; "<RightShift>" and "A" and "3"
FF 00 E2       ; ConsumerKeys "<mute>"
FE 82          ; PowerKeys "<sleep>"
04 20 05 06 07 08    ; "A" "B" "C" "D" "E"
```

Note that the second byte, if present is the modifier keys, and it always gets sent when there is more than a single key in the packet (even if there are no modifier keys pressed and the value is zero). This is based on the assumption that the most common report will contain a single key press and no modifiers. Also note that not more than 5 data bytes + modifier should be sent over the air since the 6th will not make it to the PC (1 byte is displaced by Report ID)

Processing:
Forms the USB report (swap modifier, add res byte, ReportID) and sends data to USB.

UpKey detection - certain values identify Upkeys (don't use header byte)
if (ByteLenRept == 0) and (NullPktDataBits == 0) then UpKeyStandardKeys
if (ByteLenRept == 1) and (RptType == 0xFF) then UpKeyConsumerKeys
if (ByteLenRept == 1) and (RptType == 0xFE) then UpKeyPowerKeys

ReportTypeBatteryVoltage -
if (ByteLenRept == 2) and (RptType == 0xFD) then ReportTypeBatteryVoltage

```
So a sequence would be:
ByteLenRept NullPktDataBits    app_buffer:
         1               0     04            ; "a"
         0               0                   ; UpKeyStandardKeys
         3               0     FF 00 E2      ; ConsumerKeys "<mute>"
         1               0     FF            ; UpKeyConsumerKeys
         3               0     FE 82         ; PowerKeys "<sleep>"
         1               0     FE            ; UpKeyPowerKeys

A KeepAlive sequence would be:
         1               0     04            ; "a"
         0               2                   ; KeepAlive
         0               2                   ; KeepAlive ...
         0               0                   ; UpKeyStandardKeys
```

Output:
Sends USB packet out EP1.
ep1_dmabuff = data to USB

```
Output Examples:
RptId mod res key
   01  00  00  04  00 00 00 00  ; "a"
   01  00  00  00  00 00 00 00  ; UpKeyStandardKeys
   01  00  00  59  00 00 00 00  ; "1" (keypad 1)
   01  00  00  00  00 00 00 00  ; UpKeyStandardKeys
   01  00  00  16  5B 00 00 00  ; "s" and "3"
   01  00  00  00  00 00 00 00  ; UpKeyStandardKeys
   01  20  00  04  05 06 07 08  ; "<RightShift>" and "A" "B" "C" "D" "E"
   01  00  00  00  00 00 00 00  ; UpKeyStandardKeys
   02  E2 00  ; ConsumerKeys "<mute>"
   02  00 00  ; UpKeyConsumerKeys
   03  82      ; PowerKeys "<sleep>"
   03  00      ; UpKeyPowerKeys
```

3.5.2 Mouse Application Specific Code

3.5.2.1 app_data_received_b

Input:
X Reg = ByteLenRept = Byte length of input report (-2 for hdr, chksum)
A Reg = NullPktDataBits = additional info
0x02 => KeepAlive
Mouse Report bytes in byte_buffer[], app_buffer[]

Background:
byte_buffer[] contains the input bytes from the radio including the header.
app_buffer[] points to the first data byte.
app_buffer is currently 1 byte offset from byte_buffer to skip the 1 byte header, but app_buffer[] should be used to allow changes in the length of the header (i.e. to allow it to use a two byte header).

Note: hdr=HeaderByte sc=ScanCode mod=modifier chk=checksum res=reserved
ReptType = report type, ConsumerKeys = Multimedia Keys

```
Input Examples:
byte_buffer[]= 45 01 01 ; Move X=1, Y=1
    app_buffer[]= 01 01 ; Move X=1, Y=1
    app_buffer[]= 01 01 20 ; Move X=1, Y=1, LButton
```

Processing:
Forms the USB report (translate button bits) and sends data to USB.

UpKey detection -
The mouse does not use a special condition (as the keyboard does) to identify mouse UpKeys. Instead, a mouse UpKey is a normal mouse report with the button values set to 0. This report is normally sent when a mouse button is released. This routine will still handle a null report as an UpKey because the UpKey timeout results in a null report being generated.

KeepAlive detection -
The mouse uses a special condition to represent a KeepAlive message when a mouse button is held down. For the mouse, the KeepAlive message is a Null Packet header byte with the KeepAlive bit set. The calling routine puts this information into the A register - NullPktDataBits.

ReportTypeBatteryVoltage -
if (ByteLenRept == 4) then ReportTypeBatteryVoltage

```
So a sequence would be:
ByteLenRept NullPktDataBits    app_buffer:
         2              0     01 01         ; Move X=1, Y=1
         3              0     01 01 20      ; Move X=1, Y=1, LButton
         3              0     00 00 00      ; UpKeyMouse

A KeepAlive sequence would be:
         3              0     00 00 20      ; LButton
         0              2                   ; KeepAlive
         0              2                   ; KeepAlive ...
         3              0     00 00 00      ; UpKeyMouse
```

Output:
Sends USB packet out EP2.
ep2_dmabuff = data to USB

```
Output Examples:
Button mod res key
    00  01  01  00  ; Move X=1, Y=1
    01  00  00  00  ; LButton
    00  00  00  00  ; UpKeyMouse
```

3.5.3 UpKey Application Specific Code

3.5.3.1 app_idle

Input:
upkey_timer_a = Count down timer for sending UpKeyKeyboard
upkey_timer_b = Count down timer for sending UpKeyMouse

Processing:
upkey_timer_a counts down from UP_KEY_TIMEOUT to 1
if(upkey_timer_a == 0) timer is idle
if(upkey_timer_a == 1) timer triggers Upkey to be sent
if(upkey_timer_a > 1) timer is active (decrement it)

Background:
app_idle gets called ~= 50ms
UP_KEY_TIMEOUT == 0x07 per design
upkey_timer_a set to UP_KEY_TIMEOUT ~= 300ms Upkey timeout period
upkey_timer_a set to UP_KEY_TIMEOUT when a DownKey is received
upkey_timer_a set to UP_KEY_TIMEOUT when a KeepAlive received
upkey_timer_a cleared when an UpKey is received

Output:
Sends USB packet out EP1.
ep1_dmabuff = Keyboard data to USB
ep2_dmabuff = Mouse data to USB

```
Output Example:
RptId mod res key
   01  00  00  00  00 00 00 00  ; UpKeyStandardKeys
Button mod res key
    00  00  00  00  ; UpKeyMouse
```

3.5.4 EEPROM Related Functionality

3.5.4.1 E2_ReadByte

Input: X Reg = Address of an E2 byte to be read

Processing: AT25010 data byte is read over SPI bus

Output: spi_data reg = data byte for calling app

Usage: Calling app should assign spi_data to destination after return
Example:

```
for(i=0; i<MAX_E2_BYTES; i++)
{
  X = i;                      ; Input var E2 address
  E2_ReadByte();
```

```
  E2ByteInBuff[i] = spi_data ; Output E2 Data Byte
}
```

3.5.4.2 E2_WriteByte

Input:
X Reg = Write address of an E2 byte
A Reg = Data Byte value to write

Processing:
AT25010 data byte write over SPI bus

Output: None

Example:

```
for(i=0; i<MAX_E2_BYTES; i++)
{
  X = i;                    ; Input var E2 address
  A = E2ByteOutBuff[i];     ; Input var write byte
  E2_WriteByte();
}
```

3.5.4.3 E2_GetBindParms

Input:
NVParms = Channel, PN Code, etc.

Processing:
NVParms are retrieved from the Serial EEPROM

Output:
A reg == 0 ==> Valid NVParms

3.5.4.4 E2_PutBindParms

Processing:
NVParms are stored on the Serial EEPROM
An E2 Signature byte is written at E2 address 0

3.6 Other Bridge Functionality

3.6.1 Remote Wakeup

When the USB is suspended by the host, the bridge must reduce it's current draw to less than 2.5 mA. This requires that the radio

circuitry be off most of the time. However, in order to sense activity from the WirelessUSB mouse or keyboard (and thereby know to wake the host) the radio must be on.

When suspended, the bridge supports remote wakeup by intermittently turning the radio on, checking for data from the HIDs, and, then, turning the radio off again if no HID traffic was detected.

By adjusting the duration that the radio is on, a balance can be achieved between suspend current and traffic detection reliability. The **rwu_timer_1** variable (in the **rwu_poll_radio** routine of **wusb-ls-main.asm**) controls this radio on-time.
The integral value of **rwu_timer_1** corresponds, approximately, to 1.06 ms of radio on-time Radio off-time, then, is approximately:

offTime = (255 – **rwu_timer_1**) * 1.06

3.6.2 The RadioParams Report

The WirelessUSB LS Bridge implements a mechanism to report the radio parameters of attached HID devices via the USB control endpoint.

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

The HID Report Page is defined as:

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

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

The RadioParams Report is 8 bytes long and has the following 6 data fields:

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

3.6.2.1 Requesting A New Battery Reading

When the Bridge receives, from the host, a control endpoint request with the following parameters, it requests a battery level reading from the specified HID device on the next radio transaction with that device.

Control endpoint request for new battery reading:

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

3.6.2.2 Obtaining the RadioParams Report

When the Bridge receives, from the host, a control endpoint request with the following parameters, it returns an 8-byte RadioParams report over the control endpoint.

Control endpoint request for RadioParams report:

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

When the Bridge receives the ***Get Report*** control request code, it returns a RadioParams report and then resets the *Packets Transferred* parameter for the specified device to zero.

The *Link Quality* value is updated whenever the bridge receives a radio packet from the wireless device.

Battery Level is only updated when requested by the ***Set Report*** control request described above.

At startup, the *Battery Level, Corrupt Packets* and *Packets Transferred* are initialized to zero. The Bridge is then configured as if it had received a ***Set Report*** control request for each device, requesting its battery level. If the device is present, it will report the new battery level shortly after the first radio packet is received by the Bridge.

3.6.3 Manufacturing Test Mode

This module may be conditionally compiled in to provide manufacturing test support. The module configures the radio for reception and then enters a loop waiting for packets to be sent from the tester. It computes the total number of bits received and the number of invalid bits received. It then sends packets to the tester followed by the previously computed results. After which it re-enters the reception loop waiting for another test cycle. The red and green LEDs will alternate on/off in this test mode for bridge devices with LEDs. The manufacturing test code can only be exited by cycling power.

The manufacturing test mode can be entered by two different methods depending on the compile-time configuration.

Method 1: Forcing an SE1 condition (D+ and D- are both high) on the USB bus and applying power to the bridge.

Method 2: Holding the bind button during insertion into a USB Host will enter the test mode after a successful USB enumeration.

4. REFERENCES

WirelessUSB Theory of Operation

WirelessUSB LS 2-Way HID Systems

WirelessUSB LS Getting Started Guide

CY4632 Protocol Library

Device Class Definition for Human Interface Devices (HID) (*http://www.usb.org/developers/hidpage*)

CYWUSB6934 WirelessUSB LS Datasheet (38-16007)