AN73609

EZ-USB® FX2LP™/ FX3™ Developing Bulk-Loop Example on Linux

Author: Dhanraj Rajput

Associated Project: Yes
Associated Part Family: CY7C68XXX
Software Version: libusb 1.0.0
Related Application Notes: None

Abstract

AN73609 describes how libusb can be used to develop an USB host application on a Linux-based OS for Cypress EZ-USB® FX2LP™/ FX3™ products. It includes a step-by-step procedure for developing the bulk-loop example.

Introduction

This document and the accompanying software demonstrate how simple user-mode, using the libusb Linux applications, can communicate directly to EZ-USB based USB devices without any need to write a kernel device driver. The author assumes the reader has a working knowledge of Linux and gone through FX2LP/ FX3 documentation as well as Windows-based FX2LP/FX3 tools and utilities.

Some USB devices require the installation of kernel-mode drivers to function properly under the Linux operating system; writing kernel-mode drivers can be a complicated process, but for other specialized USB devices such as scanners, digital cameras, and mass-storage devices, communication between the host and the device can be accomplished directly, without the need for device driver development, by utilizing standard features of GNU/Linux starting with the 2.4 kernel. These new features include 'Hotplug' software used to perform dynamic reconfiguration of USB devices, pre-written generic or class drivers, and the USB Device File System APIs.

The included application, "BulkLoopApp", is developed to communicate with the vendor specific bulk-loop USB Device.

The Bulk-Loop USB Device

The bulk-loop USB Device is an EZ-USB based vendor specific device programmed to loop back data on bulk endpoints. This is a standard implementation provided as a part of CY3684 DVK Example code and FX3 SDK example code. Please see 'Section Testing the Application' described later in this application note for more details on setting up the DVK.

When connected to the PC, the FX2LP device enumerates as a vendor specific device with VID=0x04b4 and PID=0x1004. In the case of FX3, the PID is 0x00F0. The device has a single interface with Bulk Endpoints (EPs). As shown in Figure 1, the USB host sends data to a bulk-out endpoint and reads data from a bulk-in endpoint. The EZ-USB device internally loops bulk-out endpoint data to bulk-in endpoint data.

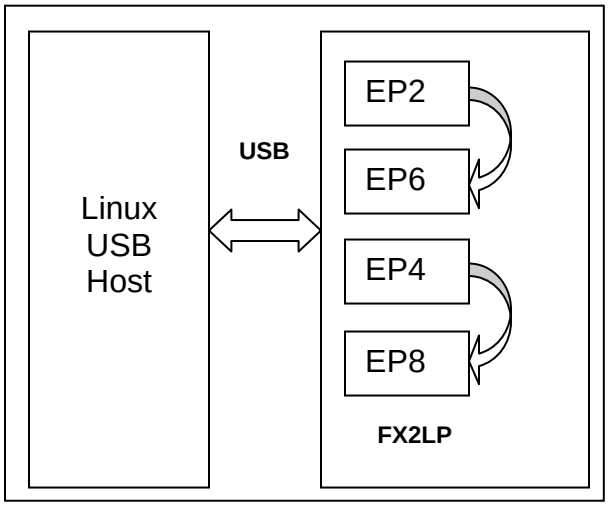
The firmware development for this device is not in the scope of this application note. For more details on the firmware, please see documentation and source code available with CY3684 DVK and FX3 SDK.

Libusb

Libusb is a suite of user-mode routines for controlling data transfer to and from USB devices on Unix-like systems without the need for kernel-mode drivers.

This application note uses libusb-1.0. Please see libusb documentation available at http://www.libusb.org for more details.

Figure 1. The Test Setup



The BulkLoopApp

The BulkLoopApp is a Linux user-mode application developed to communicate with the bulk-loop device using libusb. The application demonstrates how libusb can be used to get a list of connected USB devices, open the device of interest and transfer data over bulk endpoints. This application transfers data to out-endpoint of the bulk-loop device and reads data from in-endpoint of the bulk-loop device.

The entire source code for the application is available in attached "*bulkloopapp.c*" file. The important steps in the source code are described below:

1. At the start, initialize the libusb library.

```
/* initialize the libusb library */
err = libusb_init(NULL);
```

2. Get the list of devices connected to the host so that the device of interest can be handled.

```
/* get the list of all devices on the USB bus */
cnt = libusb_get_device_list(NULL, &devs);
```

3. Select the device of interest, based on VID, PID, and class of the device. An API "libusb_get_device_descriptor(device, &desc)" can be used to get the VID and PID of the devices. Look at the implementation of function "print_info" the attached source code for more details.

4. Get the handle to the device of interest.

```
//get device handle for all future operations
err = libusb_open (device, &dev_handle);
```

5. Now the device list can be freed. Use the following API call.

```
/* since the use of device list is over, free it */
libusb_free_device_list (devs, 1);
```

6. More information on the device configurations and interfaces can be queried using the following:

```
libusb_get_device_descriptor (device, &desc);
libusb_get_config_descriptor (device, 0, &config);
```

7. Claim the interface of interest for further operation. The example implementation here does bulk transfer on interface 0. Look at the implementation of "bulk_transfer()" function for more details.

```
err = libusb_claim_interface (dev_handle, 0);
```

8. The bulk transfer on OUT endpoint and on IN endpoint is carried out by the same function. This function knows about the direction of transfer from the MSB of the second argument, which can be either 0x00(0x00|EP_NUM) for OUT transfer and 0x80(0x80|EP_NUM) for IN transfers as shown:

```
//OUT transfer: data from PC to Device
err = libusb_bulk_transfer ( dev_handle, glOutEpNum, out_data_buf, glOutMaxPacketSize, &transffered_bytes, 100);
//IN transfer: data from Device to PC
err = libusb_bulk_transfer ( dev_handle, glInEpNum, in_data_buf, glInMaxPacketSize, &transffered_bytes, 100);
```

   In the 'libusb_bulk_transfer' function, the first argument contains the device handle. The second argument contains the transfer direction (0x00 or 0x80) OR'ed with the endpoint number (0x02, 0x06) on which the transfer has to happen. The third argument contains the address of the data buffer. The fourth argument contains the numbers of bytes to be transferred in a bulk transfer. The fifth argument contains the actual bytes transferred during the bulk transfer operation. The last argument contains the timeout value in milliseconds, which can be used for coming out of a blocked I/O situation after the timeout.

9. In the case application needs to exit due to errors or the completion of specified operations, ensure the following:
   a. The interface is released
   b. The device handle is released
   c. The device list is freed

d. The libusb exit is done

This can be done using the following API calls:

```
//release the interface claimed earlier
err = libusb_release_interface (dev_handle, 0);
/*all tasks done close the device handle */
libusb_close (dev_handle);
libusb_free_device_list (devs, 1);
/* exit from libusb library */
libusb_exit (NULL);
```

Test Setup

The setup used to test this example includes the following hardware and software:

1. PC running Ubuntu 11.10(Intel x86(32bit) architecture)
2. NEC USB 3.0 Host Card(optional)
3. libusb 1.0
4. CY3684 DVK
5. FX3 DVK
6. Bulk-Loop Firmware (*Bulkloop.iic* / *USBBulkLoopAuto.img*)
7. BulkloopApp
8. Windows PC with CyConsole or Control Center.

Building the Application

1. Make sure that basic Linux development utilities such as GCC, GDBare installed on your Linux host PC.
2. The BulkLoopApp needs libusb. Please install it using the following commands:

   *# **apt**-get install libusb-1.0-0*

   *#**apt**-get install libusb-1.0-0-dev*
3. Extract the attached BulkLoopApp-1.0.tar.gz as shown in the following snapshot:

```
dbir@dbir:~$ tar -xvzf BulkLoopApp-1.0.tar.gz
BulkLoopApp-1.0/
```

4. On the command prompt (Terminal), navigate to the application directory as shown in the following snapshot and create the application binary by using the “make” command.

```
dbir@dbir:~$
dbir@dbir:~$ cd BulkLoopApp-1.0/
dbir@dbir:~/BulkLoopApp-1.0$
dbir@dbir:~/BulkLoopApp-1.0$ ls
bulkloopapp.c  bulkloopapp.h  Makefile
dbir@dbir:~/BulkLoopApp-1.0$
```

```
dbir@dbir:~/BulkLoopApp-1.0$ make
gcc -pipe -O2 -fno-exceptions -DUSBTESTAPP_VERSION=\"1
.0\" -W -Wall -Wformat   -c -o bulkloopapp.o bulkloopa
pp.c
gcc -pipe -lusb-1.0 bulkloopapp.o -o BulkLoopApp
dbir@dbir:~/BulkLoopApp-1.0$ ls
BulkLoopApp    bulkloopapp.h  Makefile
bulkloopapp.c  bulkloopapp.o
dbir@dbir:~/BulkLoopApp-1.0$ _
```

Testing the Application

1. On the target side, program the target board (CY3684/FX3 DVK) with the ‘bulkloop’ firmware.

1.1 For FX2LP(CY3684) DVK, this can be done by using the ‘Cyconsole’ utility available under suiteUSB installation directory in Windows OS. The bulkloop firmware (*bulkloop.iic*) is available in the ‘Firmware\Bulkloop’ directory, which is also placed under the ‘suiteUSB’ installation directory. This can be done from any Windows system, since the ‘suiteUSB’ installation and the ‘Cyconsole’ utility from Cypress is only available for Windows.

1.2 For FX3 DVK, this can be done by using the ‘Control Center’ available as part of FX3 SDK for Windows OS. The bulkloop firmware (*USBBulkLoopAuto.img*) can be built from FX3 SDK. Refer to the ‘Getting Started’ document available with FX3 SDK documentation for more details on the FX3 SDK.

Once the target is programmed properly, disconnect it from the Windows host and connect it to the Linux host (test system).

2. Run the application binary from the Linux command prompt:

   # sudo/BulkLoopApp

   (**Note:** The sudo is only required if you do not have root permissions.)
3. The application displays the list of devices connected to system, selects the device of interest by providing the bus number and device number as shown below:
4. The application displays the VID and PID of the selected device, opens it for further operations and displays a list of actions.

5. Select Option 2 for bulk transfer and follow the instructions to test the bulk transfer.

Summary

The libusb can be used to develop user-mode Linux applications to communicate with vendor specific USB devices. The BulkLoopApp includes various steps involved in developing the libusb based application. It can be used as a reference for developing the Linux applications for EZ-USB FX2LP/FX3 based vendor specific devices.

About the Author

Name:	Dhanraj Rajput
Title:	Applications Engg Sr.
Background:	M. Tech, IIT Bombay
Contact:	dbir@cypress.com

Document History

Document Title: EZ-USB® FX2LP™/ FX3™ Developing Bulk-Loop Example on Linux - AN73609

Document Number: 001-73609

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	3464533	DBIR	12/15/2011	New Application Note
*A	3564214	DBIR	03/28/2012	Added FX3 Support

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

Automotive	cypress.com/go/automotive
Clocks & Buffers	cypress.com/go/clocks
Interface	cypress.com/go/interface
Lighting & Power Control	cypress.com/go/powerpsoc cypress.com/go/plc
Memory	cypress.com/go/memory
Optical Navigation Sensors	cypress.com/go/ons
PSoC	cypress.com/go/psoc
Touch Sensing	cypress.com/go/touch
USB Controllers	cypress.com/go/usb
Wireless/RF	cypress.com/go/wireless

PSoC® Solutions

psoc.cypress.com/solutions

PSoC 1 | PSoC 3 | PSoC 5

Cypress Developer Community

Community | Forums | Blogs | Video | Training

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709

Phone : 408-943-2600
Fax : 408-943-4730
Website : www.cypress.com