THIS SPEC IS OBSOLETE

Spec No: 001-74505

Spec Title: EZ-USB(R) FX2LP(TM) - DEVELOPING USB APPLICATION ON MAC OS X USING LIBUSB - AN74505

Sunset Owner: Gayathri Vasudevan (GAYA)

Replaced By: None

EZ-USB® FX2LP™ - Developing USB Application on MAC OS X using LIBUSB

Author: Gayathri Vasudevan, Praveen Kumar CP

Associated Project: Yes

Associated Part Family: CY7C68XXX

Software Version: libusb 1.0.0

Related Application Notes: None

Abstract

AN74505 describes how libusb-1.0 can be used to develop USB host application (Cocoa Application) on MAC OS X 10.6/10.7 for Cypress EZ-USB® FX2LP™ products. This includes step-by-step procedure for developing a host application to communicate with FX2LP products. Here, a bulkloop host application is demonstrated using a bulkloop firmware.

Contents

Introduction

AN74505 describes a host application built on the MAC OS platform that uses *libusb.* The host application (Cocoa Application) communicates with the BULK IN and BULK OUT endpoints of FX2LP, using the interfaces provided by the APIs of libusb. This host application implements the transfer only with devices that pass the particular VID/PID(=0x04B4/0x1004) identification. The example device used in this application note is the bulkloop device. The firmware that is attached along with this application note causes a loop back of data inside the device. It also supports vendor commands, demonstrated here by using a simple vendor command. Thus this host application, with the attached bulkloop device, demonstrates the loopback of bulk data, and control transfers using a vendor command.

This application note and accompanying software demonstrate how simple MAC applications, using libusb can communicate directly to EZ-USB based USB devices, without any need to write a kernel device driver. This document assumes that you have a working knowledge of MAC and gone through FX2LP documentation and windows based FX2LP tools/utilities.

System Requirements

Hardware Requirements

1. A FX2LP DVK (CY3684) board is used as the development and testing platforms for this example. A detailed schematic of the DVK is provided in the 'hardware folder' located in the attachment. More information about the board is available in the 'EZ-USB Advanced Development Board' section of the 'EZ-USB_GettingStarted' document, available at C:\Cypress\USB\doc\ General (after DVK install).
2. Apple laptop or Desktop with MAC OS X 10.6.x (Snow Leopard) or MAC OS X 10.7.x (LION) installed.

Software Requirements

1. Libusb application level USB driver - Libusb is an open source library that allows you to communicate with USB devices from user space. More information about the driver is available at www.libusb.org.
2. Xcode: Xcode 4 is the latest iteration of Apple's integrated development environment (IDE), a complete toolset for building Mac OS X and iOS applications. The Xcode IDE includes a powerful source editor, a sophisticated graphical UI editor, and many other features.

   Note that the host application developed in XCode 4 is not compatible with the earlier versions of Xcode.

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. Libusb is an open source library that allows you to communicate with USB devices from user space. Libusb-1.0 API Reference provides a detailed documentation of the libusb driver.

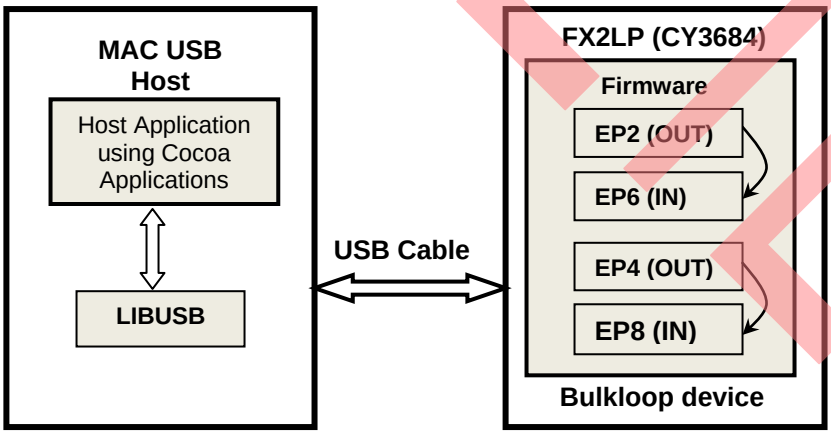
This application note use libusb-1.0. See libusb documentation available at http://www.libusb.org for more details.

LIBUSB versus CyUSB.sys

Feature	LIBUSB 1.0	CyUSB.sys
Plug and Play supported	No	Yes
Process Space	User space	Kernel space
All transfer types supported (control/bulk/interrupt/ isochronous)	Yes	Yes
Transfer interfaces supported: Synchronous Asynchronous	Yes	Yes
OS Support	Linux, FreeBSD, NetBSD, OpenBSD, Darwin, Mac OS X Windows	Windows only

Block Diagram

Figure 1. FX2LP Connected to MAC System



The "Host" (PC running MAC OS) and the "target" (FX2LP based CY3684 board) are connected through USB cable. The host application sends data to BULK OUT (EP2 or EP4) endpoints of FX2LP through libusb. When the bulk data reaches the endpoint EP2 (or EP4), the bulkloop firmware running on Fx2LP, takes this data and copies it back to EP6 IN (or EP8 IN) endpoint. The bulk data travels back in reverse through libusb and finally reaches the host

application. The host application displays the bulk data transferred.

Apart from the bulk data transfers, host application in this application note also demonstrates how to do control transfers using a vendor command. This will be explained in detail in the following sections.

For familiarizing with different components of block diagram, see the following documentation after installing FX2LP DVK.

- Libusb: libusb-1.0 API Reference, http://www.libusb.org/
- CY3684: EZ-USB_GettingStarted.pdf and DVK Users Guide.pdf available at C:\Cypress\USB\doc\General.
- FX2LP: EZ-USB_TRM.pdf and fx2lp_datasheet.pdf available at C:\Cypress\USB\doc.

Introduction to COCOA Applications

Cocoa is Apple's native object-oriented application programming interface (API) for the Mac OS X operating system. Cocoa applications are typically developed using the development tools provided by Apple, specifically Xcode (formerly Project Builder) and Interface Builder. Cocoa is programmed using an extended version of the C programming language called Objective C. For getting familiar with Objective-C, see "Learn Objective-C" or other Objective-C tutorials.

Interface Builder is used by Cocoa programmers to layout the user interface of an application. One Controller Class is created, that controls the objects on the view (GUI). Controller contains the code that will control the view. There are two important things to be known when we create variables/methods for the interface. They are as follows:

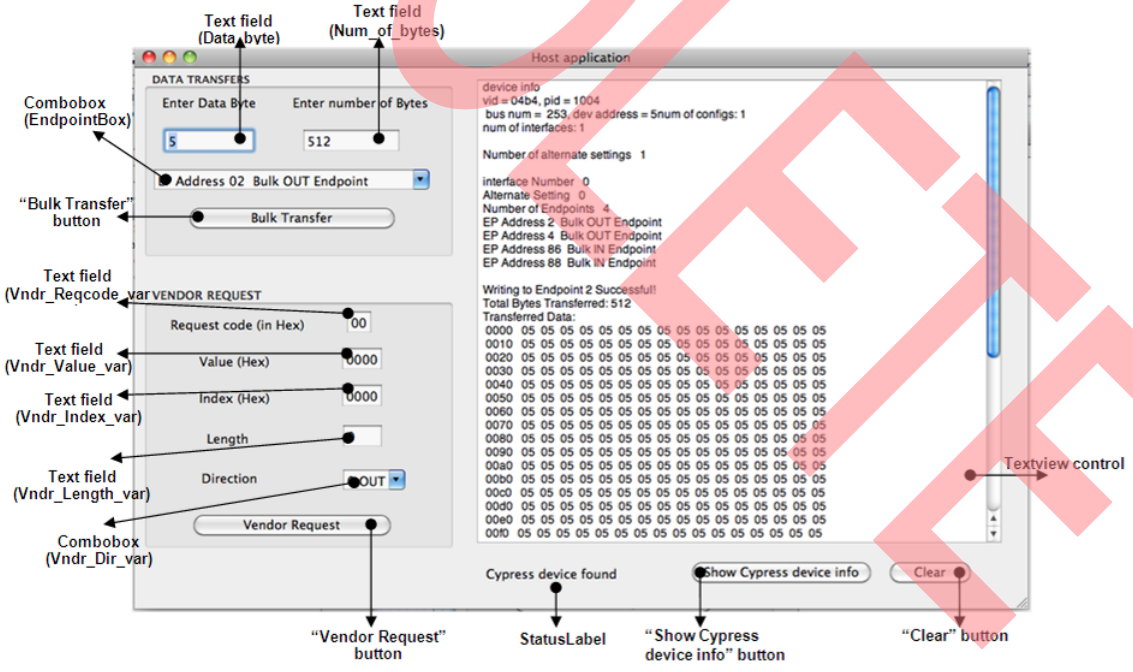
- **IBOutlet:** is a variable that will connect from our code to our view (GUI).
- **IBAction:** is about any methods that will connect from our code to our view.

All these are explained in detail in the following sections.

Host Application

The user interface of the host application is as follows:

Figure 2. GUI of Bulkloop_Cocoa_Apps



This application user interface is designed to show the following objects or controls:

- **StatusLabel** - label that displays whether a Cypress device (with a VID == 0x04B4) is found or not when the application starts.
- **Textview control** - to display the result of transfers, actual bytes transferred device information, and so on, as is the case.

- **"Show Cypress device info" button** – to display the connected Cypress device details, such as VID/PID and endpoints, in the Textview control.
- **"Clear" button** - to erase the TextView control.
- **"Bulk Transfer" button** - to initiate the Bulk transfers to/from the selected endpoint.
- **Combobox (EndpointBox)** - to display the various endpoints available in the device (any device with a VID = 0x04B4, PID = 0x1004).
- **Text field (Data_byte and Num_of_bytes)** - to enter the user inputs: data byte to be transferred, and the number of data bytes to be transferred.
- **"Vendor Request" button** - to initiate Vendor request to the device.
- **Text field (Vndr_Reqcode_var, Vndr_Value_var, Vndr_Index_var, Vndr_Length_var)** - to enter the corresponding fields in a SETUP packet required for a control transfer (Vendor request).
- **Combobox (Vndr_Dir_var)** - to input the direction of the vendor request.

OPERATION - BROAD VIEW

The firmware, which is provided in the attachment along with this application note, when downloaded into the FX2LP device causes it to enumerate with VID/PID of 0x04B4/0x1004 as a bulkloop device. (**Note** Downloading of the firmware needs to be done using a Windows PC). This particular device (let us call it the bulkloop device) has four Endpoints, configured as follows:

- EP2 – BULK OUT, double buffered, size 512
- EP6 – BULK IN, double buffered, size 512
- EP4 – BULK OUT, double buffered, size 512
- EP8 – BULK IN, double buffered, size 512

The firmware basically performs a loop back of data. Whenever there is any data in the EP2 OUT, it is immediately transferred to EP6 IN; and whenever there is any data in EP4 OUT, it is transferred to EP8 IN.

The host application that is explained in this application note is used to send the data needed for the loopback to the OUT endpoint of the device, and retrieve the data from the IN endpoint of the bulkloop device. This host application can be used to send data to OUT endpoint, and receive data from the IN endpoint of any device having the particular VID/PID. But in this application note, this is demonstrated using bulkloop firmware. Since the bulkloop firmware immediately transfers the data in the EP2 OUT (EP4 OUT) to EP6 IN (EP8 IN), the data received from the IN endpoint is the same as that was sent to the corresponding OUT endpoint. Also, one vendor command is implemented in the firmware, for the purpose of demonstration.

0xBF: IN vendor command with data transfer phase from FX2LP to USB host. It is made to return one byte (0xBF) in the data transfer phase.

- In the GUI developed in this host application, Cypress device (with a VID = 0x04B4) if attached, while the application starts, is recognized and the status is updated in the label as "Cypress device found". If the Cypress device is not found when the host application starts/or is attached at a later stage will not be recognized and hence the status will reflect as "Cypress device not found".
- Only if Cypress device is found at the startup, "Show Cypress device info" button will be enabled.
- Similarly, only if the bulkloop device (VID/PID = 0x04B4/0x1004) is found at the startup, buttons for initiating the "Bulk Transfer" and "Vendor Request" will be enabled. They are by default in disabled state.
- In the data transfers section, you need to select an endpoint from the combobox that lists the various endpoints available for the bulkloop device. Select one endpoint from the list, input the number of bytes to be transferred and the actual data byte (hex) into the corresponding textbox. Then click on the “Bulk Transfers” button to initiate the transfer. The actual bytes transferred and the result of the transfer is displayed in the Textview control.
- Similarly in Vendor Requests section, input all the parameters in the various textbox and the combobox, and then initiate the Vendor command by the "Vendor Request" button. Result of the vendor command transfer is displayed in the Textview control.
- “Show Cypress device info” button is to display the specs of the Cypress device found, in the Textview control.

Steps for Developing Application

1. Download and install Xcode 4 on the system having MAC OS X. You can download Xcode 4 from Apple Developer Connection, under "Mac Dev Center".
2. Download the latest libusb 1.0 release. The one used in this application note is libusb-1.0.8. Unzip the libusb-1.0.8.tar.bz2 by double clicking on it. The libusb-1.0.8 folder obtained on unzipping contains *libusb.h* (header file) required for accessing the libusb API s, structures and other properties.
3. Mac OS X requires libusb-1.0 to be built from the source. In order to build and install the libusb, first

copy the libusb-1.0.8 folder, obtained from step 2, to a particular known location say Home directory. Then in the Bash Terminal, type the following (after navigating to the location containing libusb-1.0.8 folder):

```
$ cd libusb-1.0.8/
$ ./configure
$ make
$ sudo make install
```

4. After installing this you shall find a libusb-1.0.0.dylib file in the path Developer/SDKs/MacOSX10.6.sdk/usr /local/lib/ on your MAC (with Snow leopard OS). If your MAC has LION OS, then you will find libusb-1.0.0.dylib file in the path Developer/SDKs/ MacOSX10.7.sdk/usr /local/lib/ on your MAC. This is the folder in which all installed library files will reside.

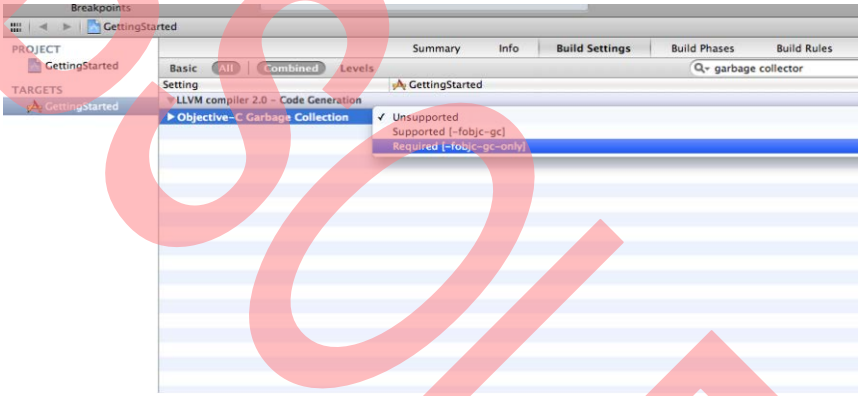
5. Launch Xcode. You can find it in /Developer/Applications. Start a new Xcode project by selecting “Create a New XCodeProject” option.

6. Under Mac OS X -> Application, choose Cocoa Application. Specify the Project Name. Ensure that all other checkboxes in the window is left unchecked, and other options default. Click Next.

7. Choose the location for your project and click 'Create'.

   Once you create the project, the main Xcode window appears. Select the Target and access the “Build Settings”. Within the Build settings, select 'All' tab. In Search tab, search for garbage collector. Then make 'Objective-C Garbage Collection' as 'Required' as is shown in the Figure 3. It is by default 'Unsupported'.

Figure 3. Memory Management



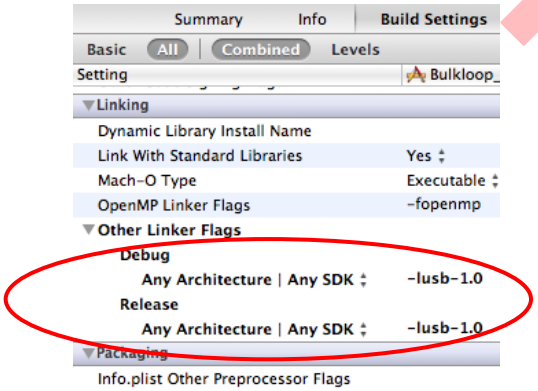
In order to know more about Garbage Collection, see Memory Management in Objective-C.

8. Right click on Project folder (Bulkloop_Cocoa_Apps) in the IDE and select Add files to “Bulkloop_Cocoa_Apps”. Navigate to the folder containing the *libusb.h* header file that was obtained in step 2 and add it to the project.

9. Select the Target and access the “Build Settings”. Within Build settings navigate to the "Linking" tab.

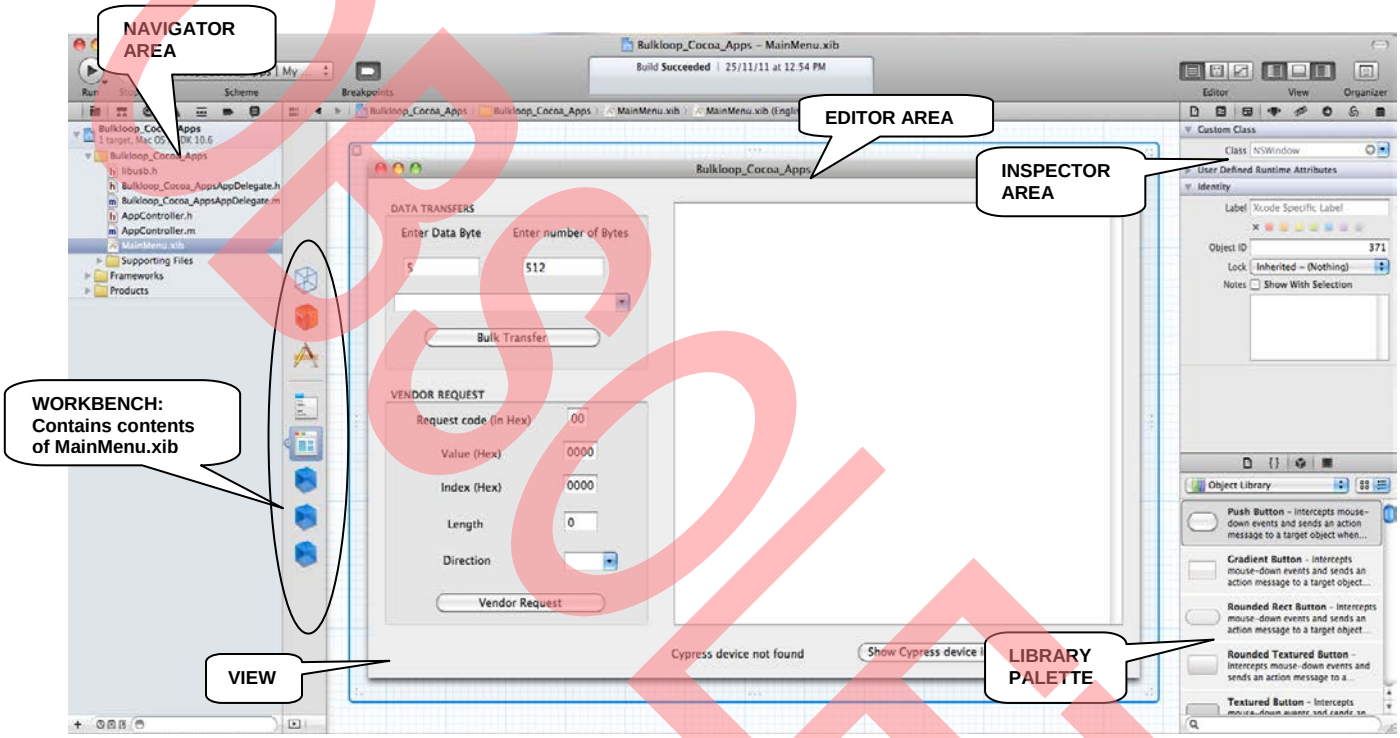
10. Add links to –lusb-1.0 in the "Other Linker Flags -> Debug" and "Other Linker Flags -> Release" as shown in Figure 4. This references the library libusb-1.0.0.dylib library file that was installed in step 3 to your project.

Figure 4. Adding Reference to Libusb



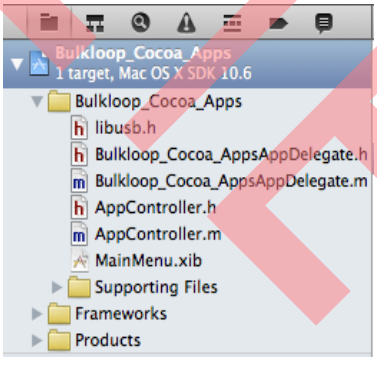
11. Now, getting started with the interface builder. In the Navigation area, under project folder, you can see *MainMenu.xib* file (earlier versions of XCode has the naming as *MainMenu.nib*. ".nib" and ".xib" files are basically equivalent). The editor area shows the user interface of the program. Interface Builder is used by Cocoa programmers to layout the user interface of an application. In XCode 4, interface builder is built into the XCode, which is very helpful. Now add the required objects from the Object library palette onto the view (GUI). You can adjust the various properties of the objects, such as the alignment, font, text, and so on, using the many inspectors found in the inspector area.

Figure 5. Interface Builder



12. Once you have added all the objects on to the view, you need to connect the code to the window. For that, we need to create an objective-C class. Select the 'File template Library' tab from the library palette and select 'MAC OS-X' option from the popup.
13. Drag the Objective - C class onto the group of files. As soon as you do that you will be prompted to enter the name for the class. Let us call it 'AppController'. Click 'Save'. *AppController.h* (interface file) and *AppController.m* (implementation file) will be added under project folder as is shown in Figure 6.

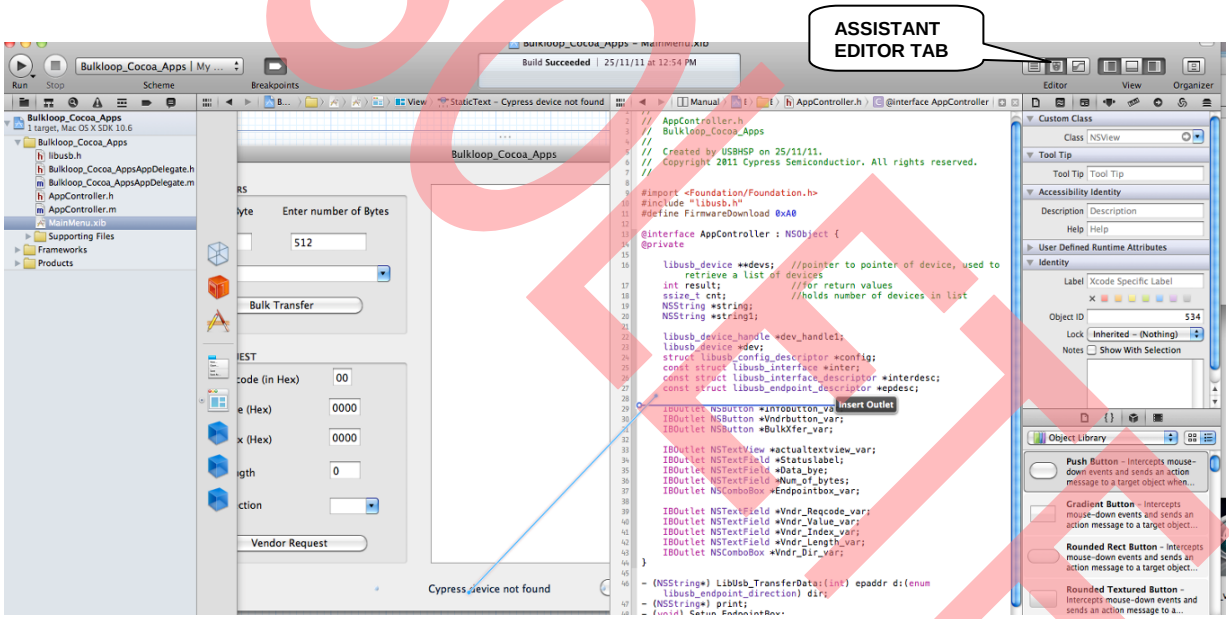
Figure 6. Navigation Area



You can do the same by right clicking on Project folder (here 'GettingStarted') from Navigator area. Select New File -> MAC OS -> Objective - C class. Save it as AppController.

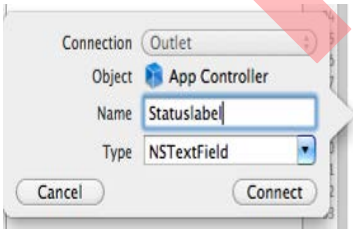
14. Since that you have added the reference to libusb-1.0.0.dylib, you have to expose the interface to it. This can be done by including a line **#include "libusb.h"**, which gives you access to the library's APIs, structures, and other properties that are needed to communicate with the USB devices. So add the line **#include "libusb.h"** in *AppController.h* and *AppController.m*.
15. Now, the View and the Controller (that is, AppController Class here) has to be linked together in a way that Controller will control all the objects contained in the view. In order to declare the IBOutlets / IBActions, we can either do that manually by typing it all and then connecting it to the objects later or in a more interactive and easy of doing that. Let us do it the second way here.
16. Connecting objects with variables / methods is kind of fun in Cocoa applications. For this, you need to create a new object of your class (AppController class in our example) on the workbench. For that, drag NSObject from the "Object Library Palette" and drop the object into our workbench, and it creates a generic object that can connect our code with our view.
17. Now we need to change the class of the object to AppController. For that, keeping the newly added NSObject selected, inside 'Identity inspector', change the 'Custom Class' to 'AppController'.
18. From the three Editor tabs (top right of Xcode screen), select 'Show the Assistant Editor' tab. Now the editor areas will be divided into two (primary editor area and the assistant editor area). Select *MainMenu.xib* from Navigator area that will show up in primary editor area and select the *AppController.h* in the Assistant editor area.
19. Now, as to define a IBOutlet for an object, say "StatusLabel", keeping the 'CTL' key on the keyboard pressed, drag from the 'StatusLabel' to the *AppController.h* as is shown in Figure 7.

Figure 7. Connecting Label Object to IBOutlet



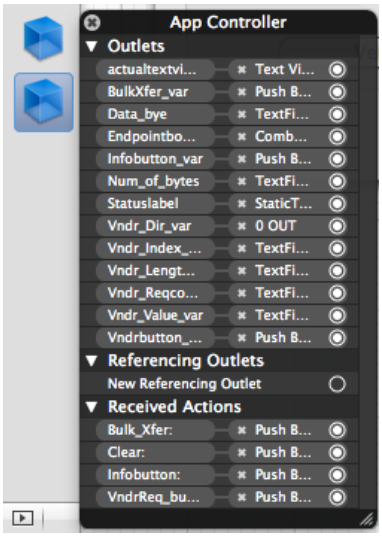
As you release the mouse, a small window pops up as is shown in the Figure 8. Note that the 'Connection' field in the window is 'Outlet'. Enter the name for the IBOutlet (Statuslabel in this example) in the 'Name' field. Leave other fields as such. (Note that the 'Object' field is same as 'AppController', which we created, and 'Type' field is 'NSTextField' since the object is label).

Figure 8. Connecting Object "Statuslabel" to IBOutlet



20. Similarly to define a method that will connect to the object, keeping the 'CTL' key on the keyboard pressed, drag from the particular object to the *AppController.h* (to the region below the '}' closing curly brace). As you leave the mouse, a small window will appear. But here the default value for 'Connection' field is 'Outlet'. Change that to 'Action'. Also, type the desired method name in the 'Name' field.
21. If you have defined the IBOutlets and IBActions manually by typing, then in order to make connections, hold down CTL key and drag from 'AppController' NSObject, from the workbench to the objects in our GUI (or vice versa). As you leave the mouse, it will list the various IBOutlets / IBActions. Select the appropriate one, and the connection will be successfully made.
22. In order to view the connections that you have made, right click on the AppController Class, and it will show all the connections made. Similarly, you can view the connections for a particular object by right clicking that particular object in the GUI. Figure 9 shows the various IBOutlets and IBActions that we have defined in this example, and the objects, to which they are connected to.

    Figure 9. List of IBOutlets and IBActions Connected to Various Objects



23. Now that you have defined all the required variables/IBOutlets/IBActions/Methods in *AppController.h* and connected it to the various objects, you can now further proceed to implement the various methods / IBActions in *AppController.m*.

Source Code Outlook

For viewing the source, open the "Bulkloop_Cocoa_Apps.xcodeproj" file, placed under "Application\Bulkloop_Cocoa_Apps" folder of the attachment. When the project opens, under Project navigator, select Bulkloop_Cocoa_Apps -> *AppController.m*. The editor area then shows the implementation file of the project.

Note If you want to build the project source, you may have to remove the existing reference for *libusb.h* and add reference to *libusb.h* again, pointing to the correct location of *libusb.h* in your system, as explained in step 9 under the section Steps for Developing Application.

The Important steps in source code are described as follows.

- At the start, initialization of the libusb library should be done. This is done by following function:

```
//initialize a library session
result = libusb_init(NULL);
```

- After the initialization, all the USB devices connected to the Linux host PC are detected and are listed. After the use, this device list should be freed. These functionalities are carried out by following two functions.

```
//gets the list of devices
cnt = libusb_get_device_list(NULL,
&devs);

//free the list, unref the devices in it
libusb_free_device_list(devs, 1);
```

- The device of interest is opened for carrying out various operations on it. Likewise after all operations are over the device should be closed. These functionalities are carried out by following two functions.

```
// opens the Bulkloop device (VID =
0x04B4/ PID = 0x1004) if found.
dev_handle =
libusb_open_device_with_vid_pid(NULL,
vid, pid);

//close the device we opened
libusb_close(dev_handle);
```

- For getting the device properties, such as vendor ID, product ID, the device descriptor is used. This is done as follows.

```
struct libusb_device_descriptor desc;
int r1 =
libusb_get_device_descriptor(dev,
&desc);
```

```
//gets the device descriptor of the USB device found
```

- For getting detailed information about the device, such as number of interfaces, number of alternate settings, endpoint number, direction, type information, the configuration descriptor is used, as follows.

```
libusb_get_config_descriptor(dev, 0, &config);
```

- For doing any operation on the device, the device handle is required. It is obtained as follows.

```
dev_handle = libusb_open_device_with_vid_pid(NULL, vid, pid);
// get device handle for all future operations
```

- For doing the bulk transfer, you need to claim the interface, on whose endpoint the bulk transfer has to happen. Also after the transfers are over, the interface has to be released. These functionalities are carried out by following two functions.

```
//Claim an interface on a given device handle.
r = libusb_claim_interface(dev_handle, 0);

//release the claimed interface
r = libusb_release_interface(dev_handle, 0);
```

- 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 part of second argument.

```
s = libusb_bulk_transfer(dev_handle, epaddr, data1,len, &actual, 100);
//Perform a USB bulk transfer
```

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

Control transfers can be performed according to the following API:

```
//Perform a USB control transfer
result1 = libusb_control_transfer(dev_handle1, bmRequestType, bRequest, wValue, wIndex, ret_data, wLength, 1000);
```

Also, after all the work is done the library should be closed. This is done by the following function:

```
//close the session
libusb_exit(NULL);
```

Code Snippets

"awakeFromNib" Method

This is the method where you can do initializations for the objects in the view. Init() method is called at the very start when the object is created and when nothing else is setup, that is, when no connections to the user interface is setup. Thus it cannot be used for the initialization of the objects in the user interface. Once the initialization for the class is done, then the connections are made. The method that gets called after the connections are made is 'awakeFromNib'. So, the initialization of the objects in the user interface can be done inside this method.

Inside this function, initialization of the libusb library should be done. Initialize the library by calling the function libusb_init and creating a session. This function must be called before calling any other libusb function. Disable the various buttons in the view. This calls a method as to compute the specs of the Cypress device (VID = 0x04B4), if found.

"print" Method

Returns the specs of the Cypress device (VID = 0x04B4), if found. Enables the "Info" button, if Cypress device is found and updates the Status label. Enables "Bulk Transfers" and "Vendor Requests" button if Bulkloop device (VID = 0x04B4/PID = 0x1004) is found. Also if bulkloop device found, it calls another method Setup_EndpointBox as to setup the combobox "EndpointBox" with the endpoints available in the bulkloop device.

"Setup_EndpointBox" Method

Sets up the combobox "EndpointBox" with endpoints available in the bulkloop device.

"Infobutton" IBACtion

IBAction linked with "Show Cypress Device Info" button. Display's the specs of the Cypress device found in the Textview control, which was computed in "print" method.

"LibUsb_TransferData: d:" Method

Initiates bulk data transfers from/to the endpoint, which is passed as parameter.

"Bulk_Xfer" IBAction

IBAction linked with Bulk Transfers button. Calls the "LibUsb_TransferData: d:" method with appropriate parameters, depending on the endpoint selected by the user in the combobox, "EndpointBox". The data

transferred and the result of the transfer is displayed in the Textview control.

"VndrReq_button:" IBAction

IBAction linked with Vendor Requests button. It initiates the Vendor command to the device, with appropriate parameters as entered by you. Result of the Vendor command is displayed in Textview control.

"Clear" IBAction

IBAction linked with Clear button: "Clear:". Clears the text of the "Textview" control.

Hardware Connections

For this example, all the jumpers on CY3684 should be left to their default states. You can refer to the default jumper settings of the DVK in the Table 1 of EZ-USB_GettingStarted.pdf. This document is available in "USB\doc\General" folder of the DVK install. The Schematic of CY3684 development board can be found inside the folder Hardware in the application.

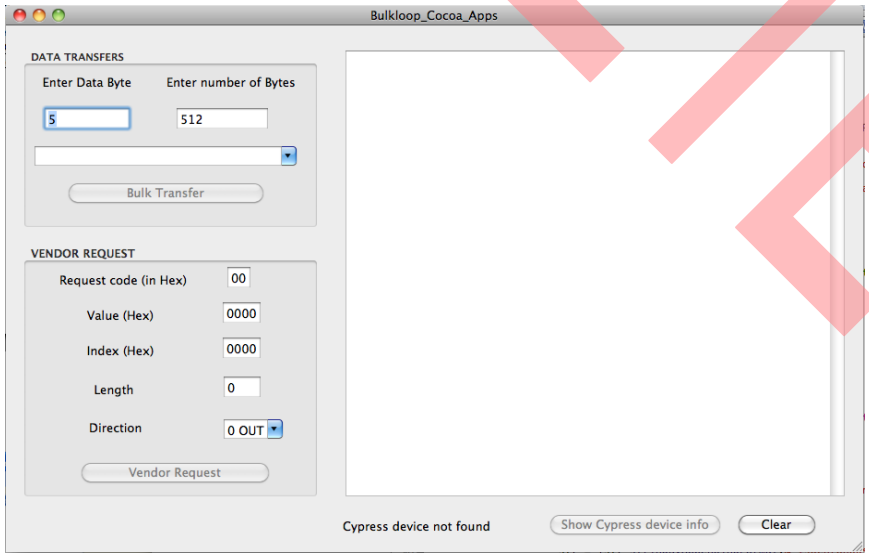
How to Test the Application

On the target side, program the target board (CY3684) with the 'bulkloop' firmware. 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 'Firmware\Bulkloop' directory. This can be done from any windows system, since the 'suiteUSB' installation and the 'Control Centre' utility from cypress is only available for windows.

1. In any Windows PC, install SuiteUSB 3.4. This installs **Control Centre** utility.
2. In CY3684 FX2LP DVK, select "EEPROM ENABLE" switch to "No EEPROM" position. Connect the (FX2LP DVK) board to the PC. The board enumerates with the default internal descriptor. Use the *CyUSB.inf* file in the driver's folder to bind with the device. For more information on binding the driver, see MatchingDriverToUSBDevice section in *CyUSB.chm* inside folder "Drivers" or the link Drivers for FX1/FX2LP.
3. Change the "EEPROM ENABLE" switch position to "EEPROM" position, and the "EEPROM SELECT" switch to the "LARGE EEPROM" position on the device.
4. Open the Control Center application present at "Start -> programs -> Cypress -> Cypress Suite USB 3.4.7 -> Control Center". Download the *Bulkloop.iic* from *Firmware\Bulkloop* folder to the large EEPROM present on the FX2LP DVK using the Control Centre utility.
5. Once the target is programmed properly, disconnect it from the windows host and connect it to the MAC host (test system).

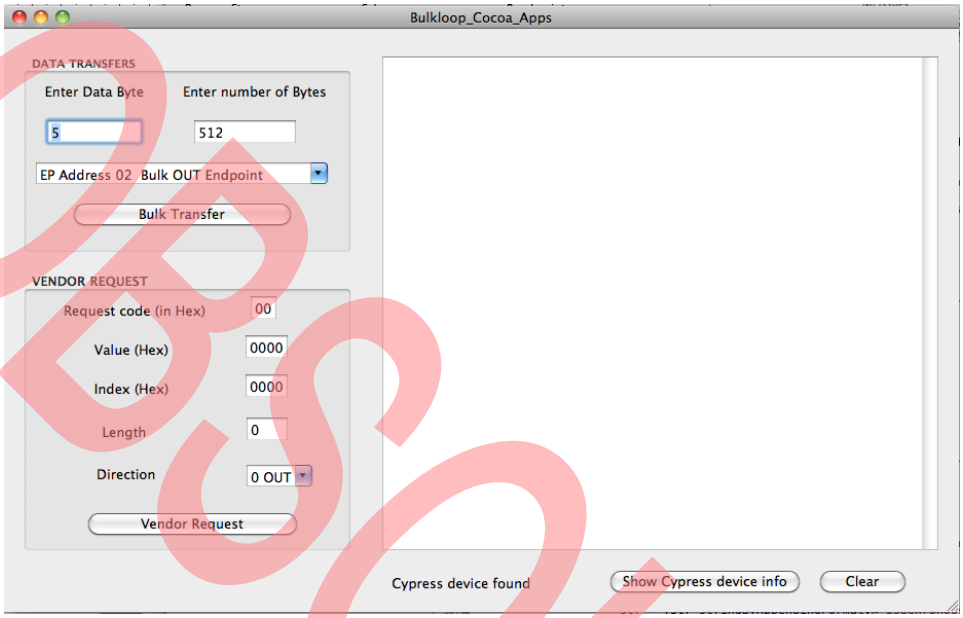
   Note For testing the *.app* file available in the attachment along with this application note also, XCode and Libusb needs to be installed. For that, follow the steps 1 to 3 under the section Steps for Developing Application.
6. Now open the Host Application GUI by opening Bulkloop_Cocoa_Apps.app at \Application\Bulkloop_Cocoa_Apps. If any Cypress device is not found the GUI appears as in Figure 10, with the Status as Cypress Device not found:

Figure 10. GUI at Startup, with no Cypress Device Connected



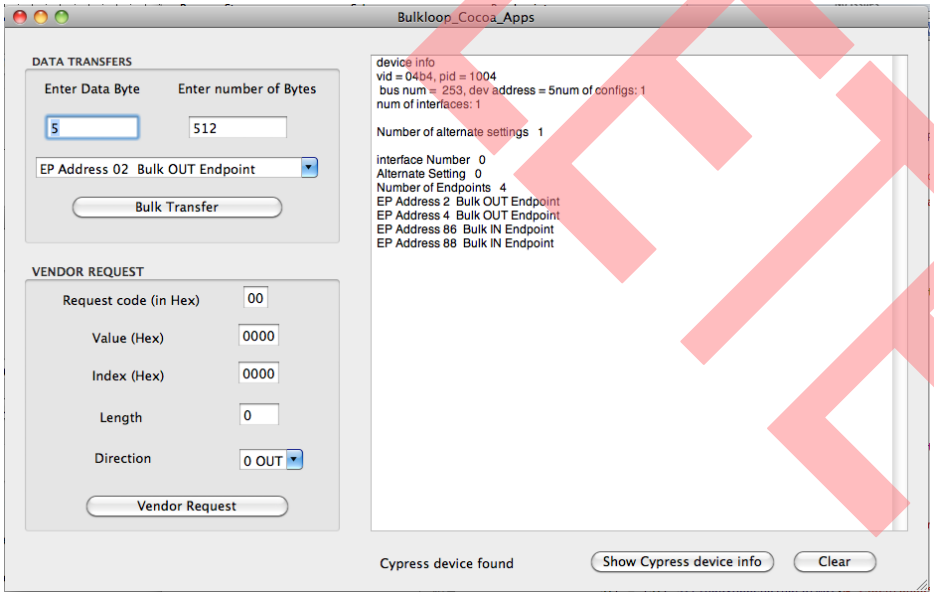
7. When the host App starts, if a Cypress device is found, Status appears as "Cypress device found" and Show Cypress Device info" button enabled. If the Cypress device found on the startup is the Bulkloop device, the GUI appears as in Figure 11, with "Bulk transfer" and "Vendor Request" button enabled.

Figure 11. GUI at Startup, with Bulkloop Device Connected



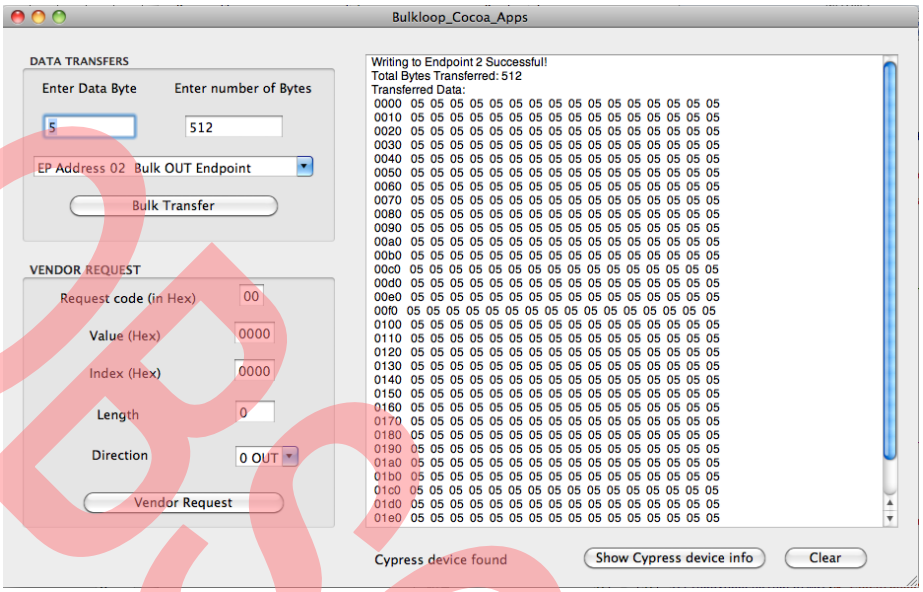
8. If any Cypress device is connected, clicking on the "Show Cypress device info" button displays the specs of the device.

Figure 12. Specs of the Device Displayed



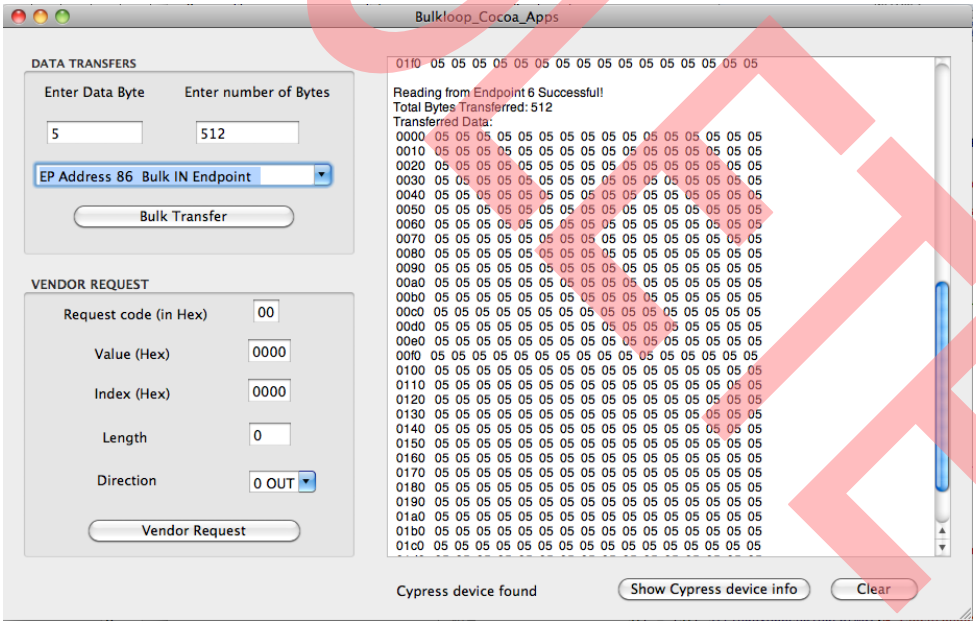
9. If Bulkloop device is connected, select an OUT endpoint, say EP2 OUT, from the combo box.
10. Enter the data byte (in Hex) to be transferred and the number of bytes to be transferred, in corresponding Textbox controls.
11. Click the “Bulk Transfer” command button for transferring data OUT. The data bytes transferred and the result is displayed in the Textview control.

Figure 13. BULK OUT Transfer Successful



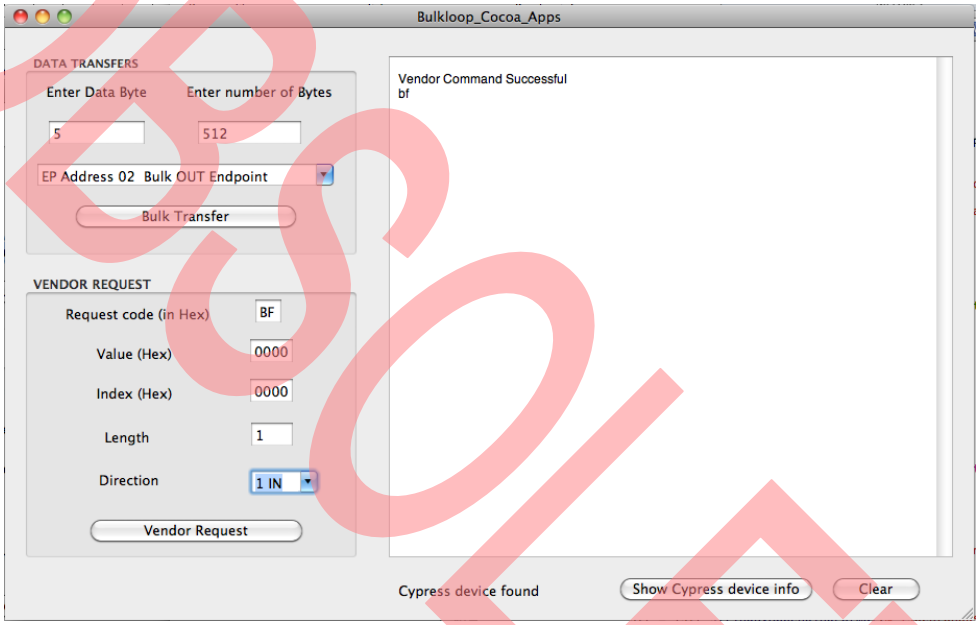
12. Now select the corresponding IN endpoint, EP6 IN from the combobox and click “Bulk Transfer” button for transferring the data in.
13. It can be seen that the data transferred IN is same as that is transferred OUT, due to the bulkloop firmware running in the device.

Figure 14. BULK IN Transfer Successful



14. Similarly, it can be tested using the EP4 OUT and EP8 IN as well.
15. Now to test the Vendor command, let us send the 0xBF vendor command, which is implemented in the bulkloop firmware provided with this Application note. For that, enter the appropriate text fields as follows:
    - Request code (in hex) -> BF
    - Value (Hex) -> 0000
    - Index (Hex) -> 0000
    - Length -> 1
    - Direction -> 1 IN
16. Click on the "Vendor Request" button, which initiates the vendor command to the bulkloop device. The result of the Vendor command transfer is displayed in the Textview control.

Figure 15. Vendor Command Successful



Additional Resources

- Getting Started with FX2LP™

Glossary

Term	Meaning
Cy3684 DVK	Cypress proprietary development kit for the EZ-USB FX2LP™ family. For more information, see the Cypress webpage.
EZ-USB	Generic name for all Cypress proprietary high-speed USB products such as FX2LP.
FX2LP	Cypress proprietary high-speed USB controllers. For more information, see the Cypress webpage.
SuiteUSB	Cypress proprietary USB development tools for Visual Studio. For more information, see the Cypress webpage.
USB Control Center	Cypress proprietary application for general-purpose USB functionality such as device identification and image downloads. The application is available through Suite USB 3.4/ CyUSB.NET\bin\cycontrol.exe.
Keil uVision	The 4K-byte evaluation version is available with the CY3684 DVK. For the full version, contact Keil.
Cocoa Applications	Apple's native object-oriented application programming interface (API) for the Mac OS X operating system. Typically developed using the development tools provided by Apple, specifically Xcode (formerly Project Builder) and Interface Builder.
XCode 4	Latest iteration of Apple's integrated development environment (IDE), a complete toolset for building Mac OS X and iOS applications. The Xcode IDE includes a powerful source editor, a sophisticated graphical UI editor, and many other features.
LIBUSB	Libusb is an open source library that allows you to communicate with USB devices from user space. See libusb documentation available at http://www.libusb.org for more details.
IBOutlets	Variable that will connect from our code to our view (GUI)
IBActions	Any method that will connect from our code to our view

About the Author

Name: Gayathri Vasudevan
Title: Applications Engineer
Contact: gaya@cypress.com

Document History

Document Title: EZ-USB® FX2LP™ - Developing USB Application on MAC OS X using LIBUSB – AN74505

Document Number: 001-74505

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	3508326	GAYA	02/14/2012	New application note
*A	4663936	GAYA	02/17/2015	Updated to new template. Completing Sunset Review.
*B	4827741	GAYA	07/08/2015	Obsolete document.

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
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 4 | PSoC 5LP

Cypress Developer Community

Community | Forums | Blogs | Video | Training

Technical Support

cypress.com/go/support

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

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