CYPRESS
PERFORM

CyUSB Suite for Linux

Version 1.0.4

Programmer's Guide

Document Version History

Version	Date	Author	Notes
1.0.0	08/07/12	V.Radhakrishnan	Detailed function description documentation added (format of libusb documentation)
1.0.2	10/01/12	Cypress Semiconductor	Update documentation formatting
1.0.3	01/23/13	Cypress Semiconductor	Updated version information. No API changes involved in this version.
1.0.4	07/11/13	Cypress Semiconductor	No API changes involved in this version. A new download_fx3 example is provided for FX3 firmware programming. download_fx2 example has been enhanced to support I2C programming. cyusb_linux GUI has been enhanced to support single step download of FX2/FX3 firmware to EEPROM.

Chapter – 1 : Introduction to CyUSB Suite for Linux

This guide helps you get started quickly with CyUSB Suite for Linux.

The software is a clone of the CyUSB Suite for Windows and helps you work with the EZ-USB FX2LP and FX3 USB Peripheral Controllers from Cypress, using a Linux host computer.

The FX2LP Peripheral Controller combines USB (High Speed) with an integrated, enhanced industry standard 8051 micro-controller, whereas the FX3 Peripheral Controller combines Super Speed USB 3.0 with an integrated ARM-9 based micro-controller. More details can be obtained by looking at the device data sheets available on the Cypress website.

The CyUSB Suite for Linux does not help you develop firmware either for the FX2 or FX3. However, firmware already developed using either the Keil Tools for FX2 or the Eclipse IDE and the GNU tool-chain for FX3, may be downloaded using the CyUSB Suite for Linux and drivers for the peripherals (host side) can be developed using the user space library provided by this software

The CyUSB Suite for Linux is a wrapper around an existing Open Source user-space USB Library called **libusb**.

The CyUSB Suite gets you started quickly with a simplified wrapper around libusb, as well as by providing an infrastructure for testing your peripheral after the firmware is downloaded.

In other words, the libusb software is a **pure 'C' library on linux**, whereas the CyUSB Suite for Linux is **a full-fledged Application** built on top of libusb in two flavors :

1. As a Command Line Interface [CLI]
2. With a GUI built as a Qt based application

The CyUSB Suite also provides a header file 'cyusb.h' and a shared object library 'libcyusb.so' which can be used to build your own user space USB applications. The applications in the CyUSB suite have been developed using the same library and header file.

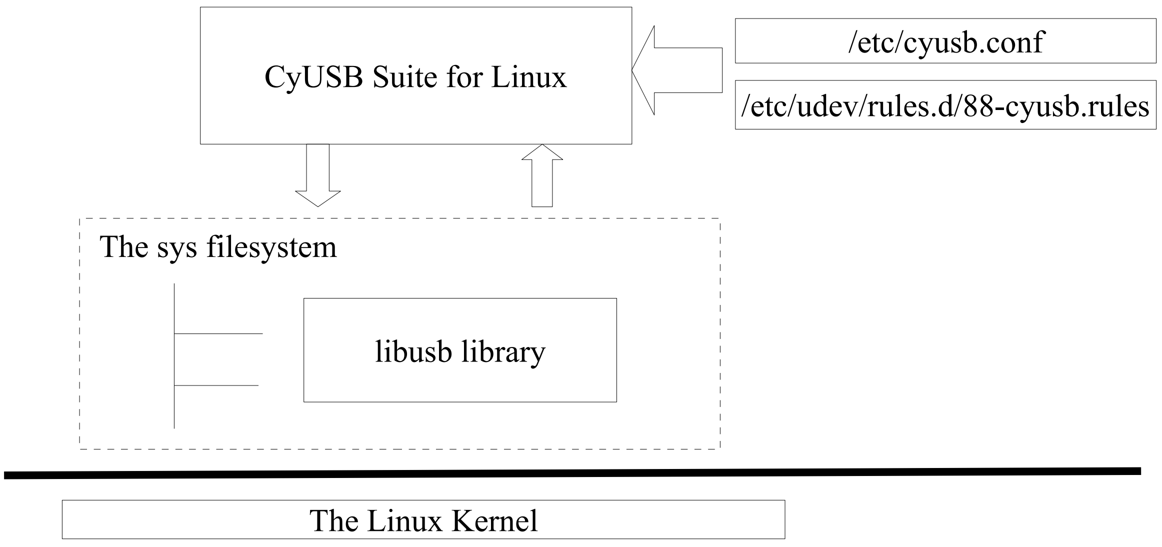


Figure-1 : Block Diagram of CyUSB Suite for Linux

Chapter - 2 : Folder structure

Once the building process is complete the following folder are created .

The ***doc*** subdirectory contains the user guide and programmers guide (this manual).

The ***configs*** subdirectory contains three files -

- cyusb.conf
- 88-cyusb.rules
- cy_renumerate.sh

The **cyusb.conf** can be modified to include additional 'devices of interest', which means those devices (typically Cypress devices, but not necessarily so) which you wish to communicate with CyUSB Suite for Linux.

NOTE:

If a Cypress device is attached and you are not getting a notification on GUI, then add the VID and PID of the new device to /etc/cyusb.conf manually. This step requires root (super-user) privileges.

The **88-cyusb.rules** is a simple UDEV rules file that can also be modified to indicate devices of interest (again, typically Cypress devices, but not necessarily so).

The **cy_renumerate.sh** script is responsible for sending a notification of plug/unplug of a device of interest to the cyusb driver library.

More information about these files would be provided later on in this guide.

The ***fx2_images and fx3_images*** sub-directories contains a set of firmware binaries that can be downloaded into the EZ-USB devices to demonstrate sample applications. These binaries are for the FX2LP and FX3 devices respectively.

The ***include*** sub-directory contains the main cyusb.h header file that has been used by the CLI as well as the GUI applications that constitute CyUSB Suite For Linux.

The ***lib*** subdirectory contains the source code as well as the shared object (equivalent to 'DLL' on Microsoft Windows) for libcyusb, the main library used to build CyUSBSuite for Linux. The source file is called libcyusb.c, whereas the shared object file is called libcyusb.so.1 and a soft link target called libcyusb.so which is the soname for the library.

The ***src*** subdirectory contains all the CLI Application sources used. A makefile is also provided to build the various sample applications.

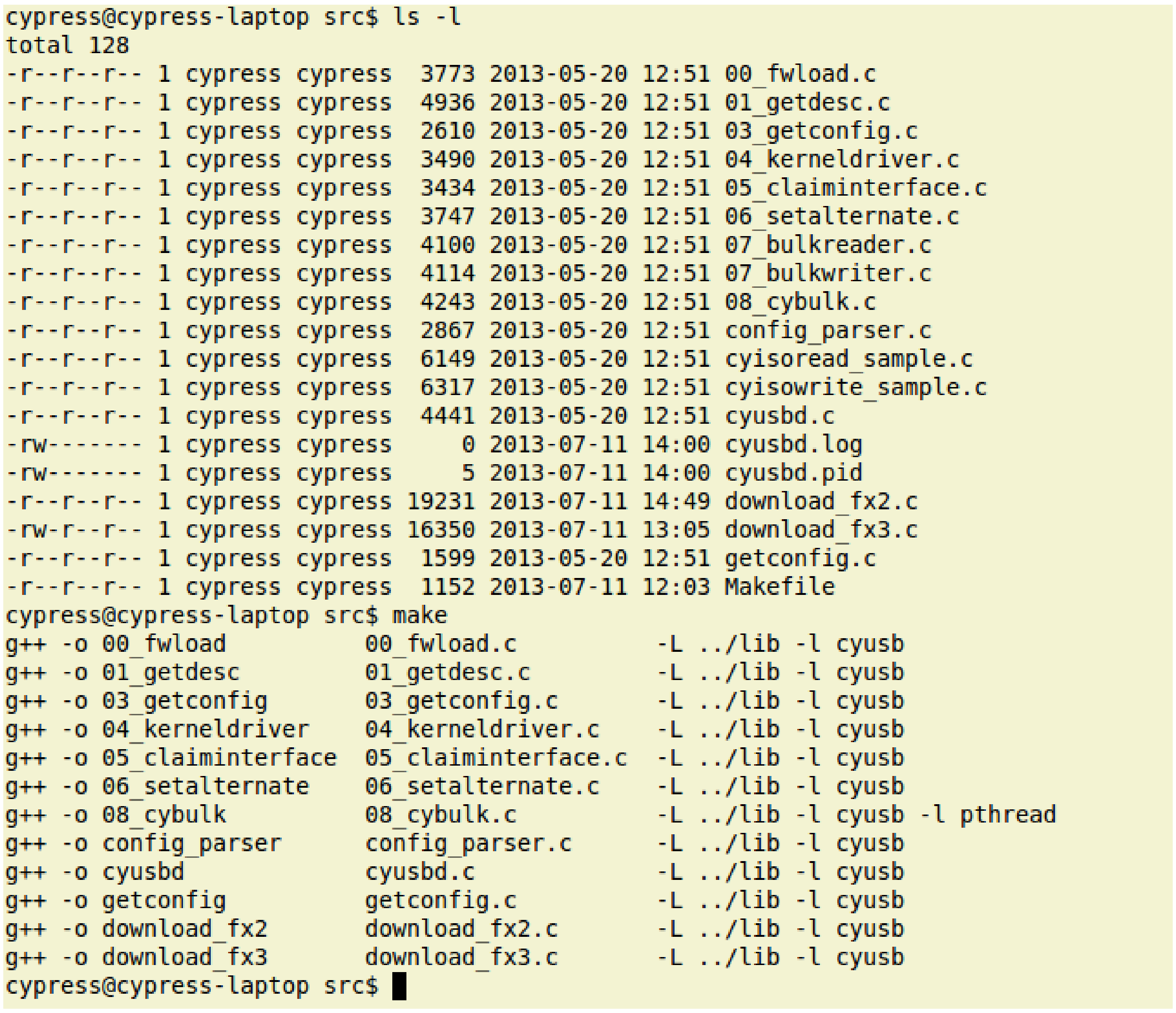
The ***gui_src*** subdirectory contains the source for the GUI, which is developed using QT.

Chapter - 3 : Getting started

It is assumed that the installation procedure documented in the user guide document has been followed before starting with the following steps.

Building the binaries :

Go to the **src** directory and do make to build all the example source files.

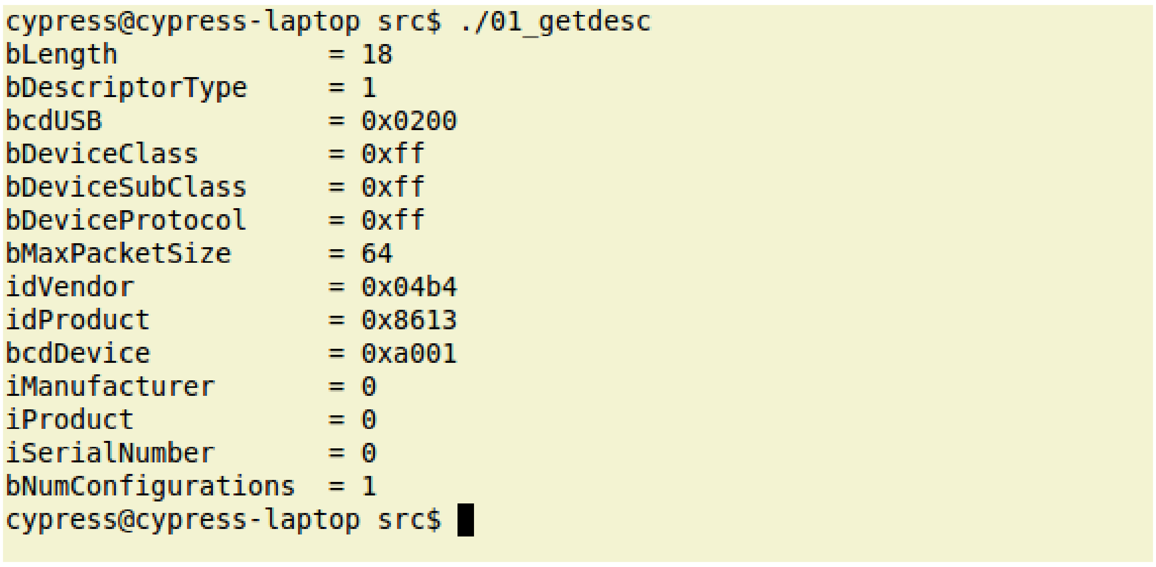


```
cypress@cypress-laptop src$ ls -l
total 128
-r--r--r-- 1 cypress cypress  3773 2013-05-20 12:51 00_fwload.c
-r--r--r-- 1 cypress cypress  4936 2013-05-20 12:51 01_getdesc.c
-r--r--r-- 1 cypress cypress  2610 2013-05-20 12:51 03_getconfig.c
-r--r--r-- 1 cypress cypress  3490 2013-05-20 12:51 04_kerneldriver.c
-r--r--r-- 1 cypress cypress  3434 2013-05-20 12:51 05_claiminterface.c
-r--r--r-- 1 cypress cypress  3747 2013-05-20 12:51 06_setalternate.c
-r--r--r-- 1 cypress cypress  4100 2013-05-20 12:51 07_bulkreader.c
-r--r--r-- 1 cypress cypress  4114 2013-05-20 12:51 07_bulkwriter.c
-r--r--r-- 1 cypress cypress  4243 2013-05-20 12:51 08_cybulk.c
-r--r--r-- 1 cypress cypress  2867 2013-05-20 12:51 config_parser.c
-r--r--r-- 1 cypress cypress  6149 2013-05-20 12:51 cyisoread_sample.c
-r--r--r-- 1 cypress cypress  6317 2013-05-20 12:51 cyisowrite_sample.c
-r--r--r-- 1 cypress cypress  4441 2013-05-20 12:51 cyusbd.c
-rw------- 1 cypress cypress     0 2013-07-11 14:00 cyusbd.log
-rw------- 1 cypress cypress     5 2013-07-11 14:00 cyusbd.pid
-r--r--r-- 1 cypress cypress 19231 2013-07-11 14:49 download_fx2.c
-rw-r--r-- 1 cypress cypress 16350 2013-07-11 13:05 download_fx3.c
-r--r--r-- 1 cypress cypress  1599 2013-05-20 12:51 getconfig.c
-r--r--r-- 1 cypress cypress  1152 2013-07-11 12:03 Makefile
cypress@cypress-laptop src$ make
g++ -o 00_fwload          00_fwload.c          -L ../lib -l cyusb
g++ -o 01_getdesc         01_getdesc.c         -L ../lib -l cyusb
g++ -o 03_getconfig       03_getconfig.c       -L ../lib -l cyusb
g++ -o 04_kerneldriver    04_kerneldriver.c    -L ../lib -l cyusb
g++ -o 05_claiminterface  05_claiminterface.c  -L ../lib -l cyusb
g++ -o 06_setalternate    06_setalternate.c    -L ../lib -l cyusb
g++ -o 08_cybulk          08_cybulk.c          -L ../lib -l cyusb -l pthread
g++ -o config_parser      config_parser.c      -L ../lib -l cyusb
g++ -o cyusbd             cyusbd.c             -L ../lib -l cyusb
g++ -o getconfig          getconfig.c          -L ../lib -l cyusb
g++ -o download_fx2       download_fx2.c       -L ../lib -l cyusb
g++ -o download_fx3       download_fx3.c       -L ../lib -l cyusb
cypress@cypress-laptop src$ █
```

Test to see if everything is installed fine, by first plugging in the FX2LP DVK and then running the 01_getdesc program.

Example - Get Device Descriptor : 01_getdesc.c

$ ./01_getdesc

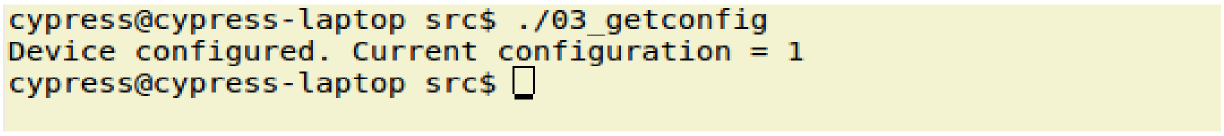


```
cypress@cypress-laptop src$ ./01_getdesc
bLength                 = 18
bDescriptorType         = 1
bcdUSB                  = 0x0200
bDeviceClass            = 0xff
bDeviceSubClass         = 0xff
bDeviceProtocol         = 0xff
bMaxPacketSize          = 64
idVendor                = 0x04b4
idProduct               = 0x8613
bcdDevice               = 0xa001
iManufacturer           = 0
iProduct                = 0
iSerialNumber           = 0
bNumConfigurations      = 1
cypress@cypress-laptop src$ 
```

The various programs demonstrated in the src directory can be run on the FX2 and/or FX3 devices. Some programs require a firmware binary to be downloaded and this is also illustrated in this guide, where appropriate.

Example - Get Device Configuration : 03_getconfig.c

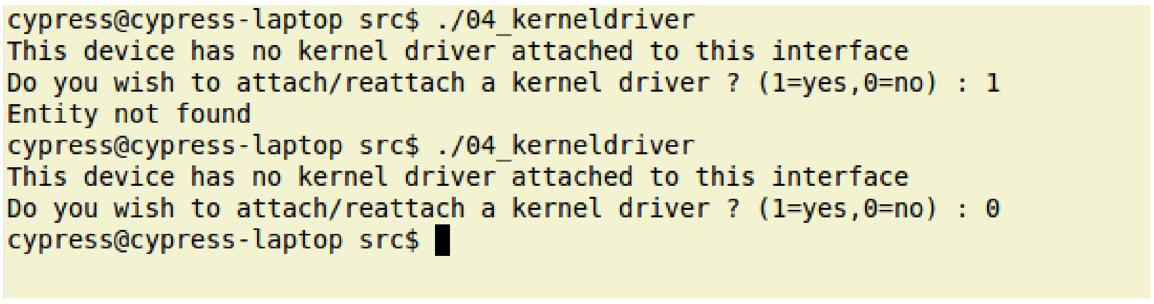
$./03_getconfig



```
cypress@cypress-laptop src$ ./03_getconfig
Device configured. Current configuration = 1
cypress@cypress-laptop src$ 
```

Example - Attach / Detach Kernel Mode Driver for an USB device

$ ./04_kerneldriver



```
cypress@cypress-laptop src$ ./04_kerneldriver
This device has no kernel driver attached to this interface
Do you wish to attach/reattach a kernel driver ? (1=yes,0=no) : 1
Entity not found
cypress@cypress-laptop src$ ./04_kerneldriver
This device has no kernel driver attached to this interface
Do you wish to attach/reattach a kernel driver ? (1=yes,0=no) : 0
cypress@cypress-laptop src$ 
```

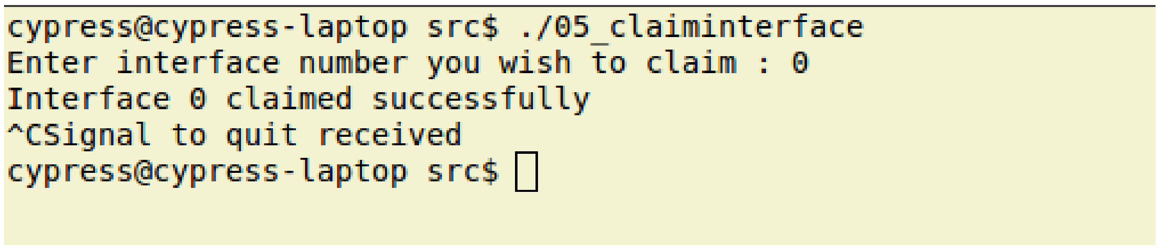
This example shows how to detach a kernel mode driver (if one exists) before you attach a user mode driver through CyUSB.

For a USB device, there is a driver per interface and each interface has to be free (not claimed) before it can be used for data traffic.

Therefore, if a device-interface already has a driver interface claimed by a kernel driver module, this has to be first released ('freed') before you can use it with CyUSB.

Example - Claim an interface for a USB device

$ ./05_claiminterface



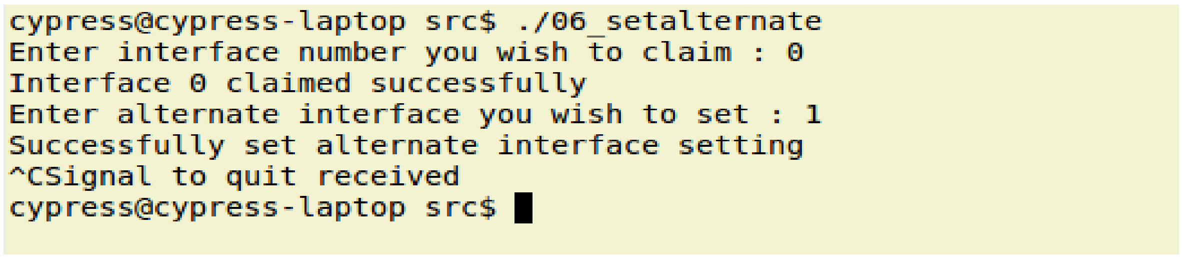
```
cypress@cypress-laptop src$ ./05_claiminterface
Enter interface number you wish to claim : 0
Interface 0 claimed successfully
^CSignal to quit received
cypress@cypress-laptop src$ 
```

Example - Set an alternate interface (after claiming an interface)

The FX2 device has 1 interface, and four alternate interfaces numbered as 0, 1, 2 and 3. The alternate interface selected by default is 0.

We will demonstrate how to select an alternate interface of 1 for the interface 1

$ ./06_setalternate



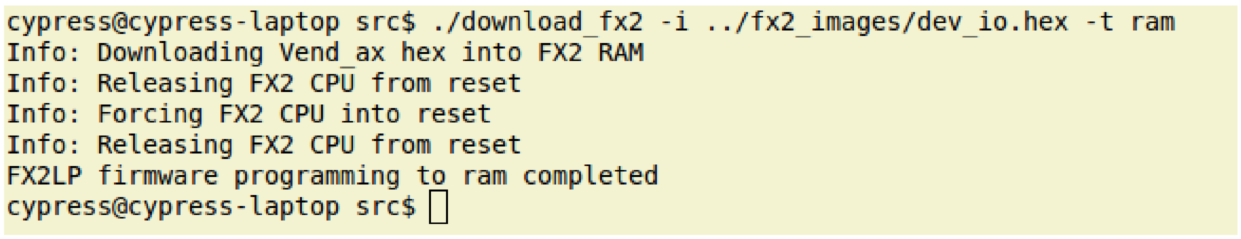
```
cypress@cypress-laptop src$ ./06_setalternate
Enter interface number you wish to claim : 0
Interface 0 claimed successfully
Enter alternate interface you wish to set : 1
Successfully set alternate interface setting
^CSignal to quit received
cypress@cypress-laptop src$ 
```

The download_fx2 example supports downloading FX2LP firmware into the RAM (internal or external), small I2C EEPROM or large I2C EEPROM.

Example - Downloading the dev_io.hex file to the FX2LP RAM

$ ./download_fx2 -i ../fx2_images/dev_io.hex -t ram

Download the dev_io.hex firmware example that demonstrates GPIO based switches and 7-segment LED display on the FX2LP DVK.



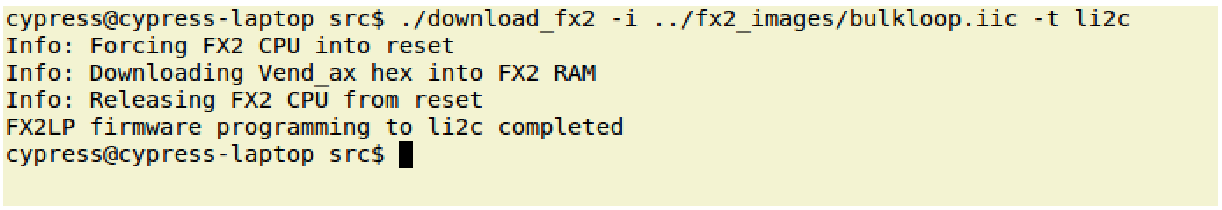
```
cypress@cypress-laptop src$ ./download_fx2 -i ../fx2_images/dev_io.hex -t ram
Info: Downloading Vend_ax hex into FX2 RAM
Info: Releasing FX2 CPU from reset
Info: Forcing FX2 CPU into reset
Info: Releasing FX2 CPU from reset
FX2LP firmware programming to ram completed
cypress@cypress-laptop src$ 
```

Example - Downloading the bulkloop.iic file to large I2C EEPROM

The bulkloop firmware loops back EP2OUT to EP6IN and EP4OUT to EP8IN. The iic version of the firmware binary (generated through hex2bix) is downloaded to the large I2C EEPROM in this example.

Please note that only iic files should be downloaded to the I2C EEPROMs. The download program does not do any verification of the file format or content.

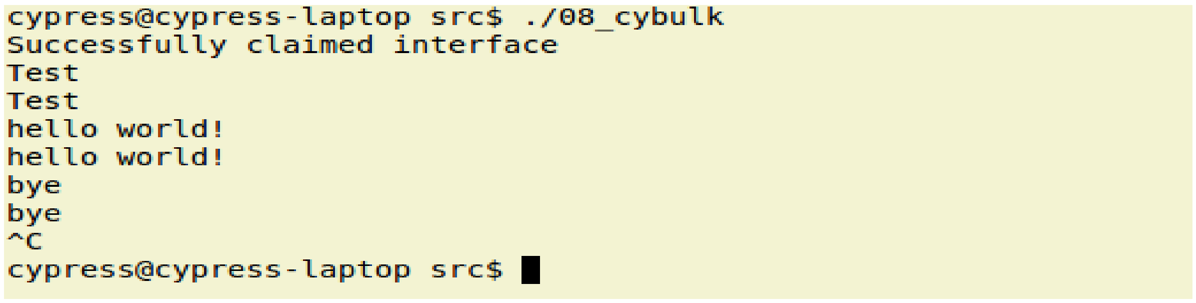
$ ./download_fx2 ../fx2_images/bulkloop.iic -t li2c



```
cypress@cypress-laptop src$ ./download_fx2 -i ../fx2_images/bulkloop.iic -t li2c
Info: Forcing FX2 CPU into reset
Info: Downloading Vend_ax hex into FX2 RAM
Info: Releasing FX2 CPU from reset
FX2LP firmware programming to li2c completed
cypress@cypress-laptop src$ █
```

Test with the program 08_cybulk, the program echoes what you type in...

$ ./08_cybulk



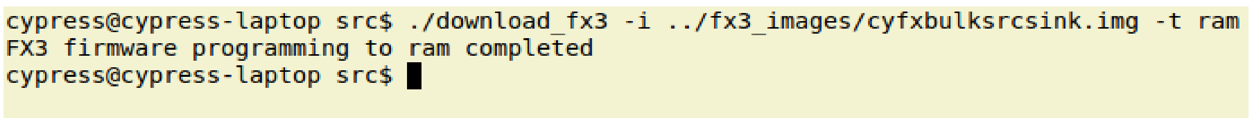
```
cypress@cypress-laptop src$ ./08_cybulk
Successfully claimed interface
Test
Test
hello world!
hello world!
bye
bye
^C
cypress@cypress-laptop src$ █
```

The download_fx3 program supports FX3 firmware download to device RAM, I2C EEPROM or SPI Flash devices.

Example - Downloading the cyfxbulksrcsink.img file to FX3 RAM

The cyfxbulksrcsink firmware implements a pair of IN and OUT endpoints that serve as perfect data sink and source. EP1OUT continuously accepts all data sent to it, and EP1IN continuously sends full data packets upto the USB host.

$ ./download_fx3 -i ../fx3_images/cyfxbulksrcsink.img -i ram

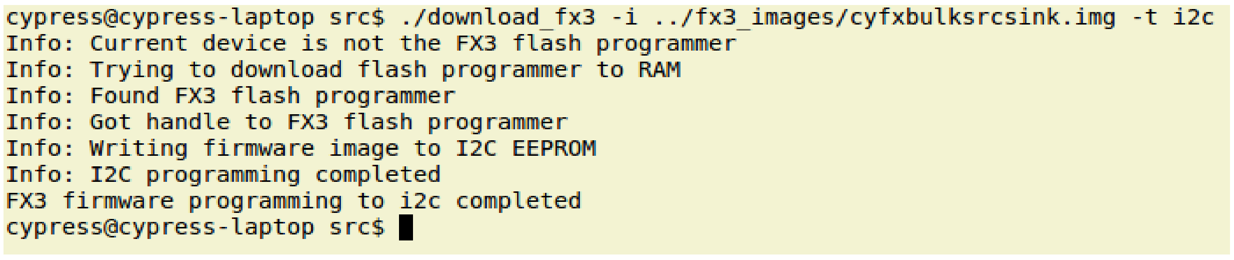


```
cypress@cypress-laptop src$ ./download_fx3 -i ../fx3_images/cyfxbulksrcsink.img -t ram
FX3 firmware programming to ram completed
cypress@cypress-laptop src$ █
```

Example – Downloading FX3 firmware to I2C EEPROM

$ ./download_fx3 -i ../fx3_images/cyfxbulksrcsink.img -i i2c

Download the cyfxbulksrcsink firmware onto I2C EEPROM devices. The download_fx3 program automatically downloads the cyfxflashprog firmware to FX3 RAM, and then uses it to do the EEPROM programming. As the I2C programming is really slow, this operation could take upto a couple of minutes.



```
cypress@cypress-laptop src$ ./download_fx3 -i ../fx3_images/cyfxbulksrcsink.img -t i2c
Info: Current device is not the FX3 flash programmer
Info: Trying to download flash programmer to RAM
Info: Found FX3 flash programmer
Info: Got handle to FX3 flash programmer
Info: Writing firmware image to I2C EEPROM
Info: I2C programming completed
FX3 firmware programming to i2c completed
cypress@cypress-laptop src$ █
```

Chapter - 4 : Programmers Guide to the CyUSB Library

This chapter gives details on how to develop code using the CyUSB Suite for Linux, as well as steps to compile and link the code, using the libcyusb.so library

Developing code :

The **cyusb.h** header file present in $CyHome/include subdirectory, needs to be included in your program.

For example, if you are in the $CyHome/src subdirectory, you would say

#include "../include/cyusb.h"

This header file describes one major data type called **struct cydev**, which is declared as follows :

```
struct cydev {
        cyusb_device          *dev;
        cyusb_handle          *handle;
        unsigned short        vid;
        unsigned short        pid;
        unsigned char         is_open;
        unsigned char         busnum;
        unsigned char         devaddr;
        unsigned char         filler;
};
```

The **cyusb_device** data-type maps to the opaque libusb data-type called **struct libusb_device**, and the **cyusb_handle** data-type maps to the opaque libusb data-type called
struct libusb_device_handle.

1. **Opening a device (** and populating the cydev array **)**

   int cyusb_open(void);

   This function populates the cydev [] array and returns the number of **interesting devices** found. An '**interesting device**' or alternatively, '**a device of interest**' is a device whose vendor ID/device ID is present in the **/etc/cyusb.conf** file described earlier.

   This function is overloaded and a simpler alternative is

   int cyusb_open(unsigned short vid, unsigned short pid);

   which populates the cydev array with just one entry and returns 1 if a single device is found that matches the vendor ID and device ID mentioned in the parameters. Usually, you would typically get a return value of 1, since you would be dealing with a single device of interest, but the library has been built (and tested) to support multiple instances of the same vendor id/device id combinations, in which case, you would need to traverse the cydev[] array and extract the handle for the appropriate device by matching with bus number and device address.

2. **Obtaining a cyusb_handle**

   cyusb_handle * cyusb_gethandle(int index);

   This function usually takes as input parameter the number 0 since you would usually be dealing with a single device of interest, as discussed earlier. The function then returns a non-null cyusb_handle which is then used subsequently for data transfers.

3. **Getting Device Information**

   Given a handle, it is possible to extract the vendor ID and device ID as follows :

   unsigned short cyusb_getvendor(cyusb_handle *); and
   unsigned short cyusb_getproduct(cyusb_handle *);

4. **Closing all cyusb devices of interest**

   void cyusb_close(void);

   This function closes ALL cyusb devices of interest discovered.

5. **Getting Bus Number and Device Address, given the handle**

   int cyusb_get_busnumber(cyusb_handle *);
   int cyusb_get_devaddr(cyusb_handle *);

   These two functions return th bus number and the device address, given the handle. Since the handle itself is returned, given the index of the cydev[] array, it is useful in those circumstances where you have multiple instances of the same device.

6. **Determining whether a usb device interface already is claimed by a kernel driver**

   The CyUSB Suite for Linux software is essentially a user mode driver library for a device. This means it is possible to communicate with a USB device provided it is not already claimed by another driver (user mode or kernel mode). The function described below returns true if a kernel mode driver is active for a given usb device handle :

   int cyusb_kernel_driver_active(cyusb_handle *, int interface);

7. **Detach / Attach a kernel mode driver for a usb device of interest** :

   In case a device already has a kernel mode driver active, as returned by true in the earlier API described just above, then this API allows one to detach a kernel mode driver, which is

then normally followed by claiming the interface by a user mode application like CyUSB Suite.
int cyusb_detach_kernel_driver(cyusb_handle *, int interface);
int cyusb_attach_kernel_driver(cyusb_handle *, int interface);

8. **Claiming and releasing an interface :**

User mode applications such as CyUSB Suite for Linux, can only work after claiming an interface. Use the following API.

int cyusb_claim_interface(cyusb_handle *h, int interface);

and the API for releasing the interface is :

int cyusb_release_interface(cyusb_handle *h, int interface);

9. **Getting USB Descriptors through a standard request :**

```
int cyusb_get_descriptor(cyusb_handle *h, unsigned char desc_type,
                              unsigned char desc_index,unsigned char *data, int len);
```

This function translates into the standard USB Request using a control type of transfer.

However, there are easier and more direct API to obtain Device, Configuration, String etc.

```
int cyusb_get_device_descriptor(cyusb_handle *h,
        struct libusb_device_descriptor *desc);

int cyusb_get_active_config_descriptor(cyusb_handle *h,
        struct libusb_config_descriptor **);

int cyusb_get_config_descriptor(cyusb_handle *h, unsigned char index,
        struct libusb_config_descriptor **);

int cyusb_get_config_descriptor_by_value(cyusb_handle *h,
        unsigned char bConfigurationValue, struct libusb_config_descriptor **config);
```

10. **Doing Data Transfers, after obtaining the handle :**

```
int cyusb_control_transfer (cyusb_handle *h, unsigned char bmRequestType,
        unsigned char bRequest, unsigned short wValue,
        unsigned short wIndex, unsigned char *data,
        unsigned short wLength, unsigned int timeout);

int cyusb_bulk_transfer(cyusb_handle *h, unsigned char endpoint,
        unsigned char *data, int length, int *transferred, int timeout);

int cyusb_interrupt_transfer(cyusb_handle *h, unsigned char endpoint,
```

unsigned char *data, int length, int *transferred, unsigned int timeout);

These translate directly into libusb commands and hence you are requested to look into the libusb library for more details.

Downloading Firmware into FX2

download_fx2.c

This program downloads a .hex file into the FX2 device.

The invocation of this program is as follows :

./download_fx2 <file.hex> OR

./download ../fx2_images/download.hex <file.hex>

The first form of the invocation is used when we wish to download a single .hex file directly using the Vendor Extension 0xA0 command.

The second form of the download invocation is used when we wish to download a .hex file using the Vendor Extension 0xA3 command (loading an image into external RAM).

Note: The Library does not provide an interface for Isochronous data transfer. Please refer to cyisoread_sample.c and cyisowrite_sample.c in /src folder for reference on how to use the libusb APIs to do Isochronous data transfers.

Using the library - Detailed Function Documentation

Structure Documentation

```
typedef          struct  libusb_device          cyusb_device;       /* Opaque object from libusb */
typedef  struct  libusb_device_handle           cyusb_handle;       /* Opaque object from libusb */

struct cydev {
        cyusb_device          *dev;                    /* as above ... */
        cyusb_handle          *handle;                 /* as above ... */
        unsigned short        vid;                     /* Vendor ID  */
        unsigned short        pid;                     /* Product ID */
        unsigned char         is_open;                 /* When device is opened, val = 1 */
        unsigned char         busnum;                  /* The bus number of this device */
        unsigned char         devaddr;                 /* The device address              */
        unsigned char         filler;                  /*  Padding to make struct = 16 bytes */
};
```

The above structure gets populated (as an array) when the library is opened using the cyusb_open() call or the overloaded function cyusb_open(unsigned short vid, unsigned short pid); The array would contain only 'devices of interest' i.e if the device is mentioned in the configuration file /etc/cyusb.conf.

Function Documentation

Prototype	:	**int cyusb_open(void);**
Description	:	This initializes the underlying libusb library, populates the cydev[] array, and returns the number of devices of interest detected. A 'device of interest' is a device which appears in the /etc/cyusb.conf file.
Parameters	:	None
Return Value	:	Returns an integer, equal to number of devices of interest detected.

Prototype	:	**int cyusb_open(unsigned short vid, unsigned short pid);**
Description device	:	This is an overloaded function that populates the cydev[] array with just one that matches the provided vendor ID and Product ID.
Parameters	:	unsigned short vid : Vendor ID unsigned short pid : Product ID
Return Value	:	Returns 1 if a device of interest exists, else returns 0. This function is only useful if you know in advance that there is only 1 device with the given VID and PID attached to the host system.

Prototype	:	**cyusb_handle * cyusb_gethandle(int index);**
Description	:	This function returns a libusb_device_handle given an index from the cydev[] array.
Parameters	:	int index : Equal to the index in the cydev[] array that gets populated during the cyusb_open() call described above.
Return Value	:	Returns the pointer to a struct of type cyusb_handle, also called as libusb_device_handle.

Prototype	:	**unsigned short cyusb_getvendor(cyusb_handle *);**
Description	:	This function returns a 16-bit value corresponding to the vendor ID given a device's handle.
Parameters	:	cyusb_handle *handle : Pointer to a struct of type cyusb_handle.
Return Value	:	Returns the 16-bit unique vendor ID of the given device.

Prototype	:	**unsigned short cyusb_getproduct(cyusb_handle *);**
Description	:	This function returns a 16-bit value corresponding to the device ID given a device's handle.
Parameters	:	cyusb_handle *handle : Pointer to a struct of type cyusb_handle.
Return Value	:	Returns the 16-bit product ID of the given device.

Prototype	:	**void cyusb_close(void);**
Description	:	This function closes the libusb library and releases memory allocated to cydev[].
Parameters	:	none.
Return Value	:	none.

Prototype	:	**int cyusb_get_busnumber(cyusb_handle * handle);**
Description	:	This function returns the Bus Number pertaining to a given device handle
Parameters	:	cyusb_handle *handle : The libusb device handle
Return Value	:	An integer value corresponding to the Bus Number on which the device resides. This is also the same value present in the cydev[] array.

Prototype	:	**int cyusb_get_devaddr(cyusb_handle * handle);**
Description	:	This function returns the device address pertaining to a given device handle
Parameters	:	cyusb_handle *handle : The libusb device handle
Return Value	:	An integer value corresponding to the device address (between 1 to 127). This is also the same value present in the cydev[] array.

Prototype	:	**int cyusb_get_max_packet_size(cyusb_handle * handle, unsigned char endpoint);**
Description	:	This function returns the max packet size that an endpoint can handle, without taking into account high-bandwidth capability. It is therefore only useful for Bulk, not Isochronous endpoints.
Parameters	:	cyusb_handle *handle : The libusb device handle unsigned char endpoint : The endpoint number

Return Value	:	An integer value corresponding to the max packet size capable of being handled by that endpoint.
Prototype	:	**int cyusb_get_max_iso_packet_size(cyusb_handle * handle, unsigned char endpoint);**
Description	:	This function returns the max packet size that an isochronous endpoint can handle, after considering multiple transactions per microframe if present.
Parameters	:	cyusb_handle *handle : The libusb device handle unsigned char endpoint : The endpoint number
Return Value	:	An integer value corresponding to the max packet size capable of being handled by that isochronous endpoint.
Prototype	:	**int cyusb_get_configuration(cyusb_handle * handle, int *config);**
Description	:	This function determines the bConfiguration value of the active configuration.
Parameters	:	cyusb_handle *handle: The libusb device handle int * config : Address of an integer variable that will store the currently active configuration number.
Return Value	:	0 on success, or an appropriate LIBUSB_ERROR
Prototype	:	**int cyusb_set_configuration(cyusb_handle * handle, int config);**
Description	:	This function sets the device's active configuration (standard request).
Parameters	:	cyusb_handle *handle : The libusb device handle int config : Configuration number required to be made active.
Return Value	:	0 on success, or an appropriate LIBUSB_ERROR
Prototype	:	**int cyusb_claim_interface(cyusb_handle * handle,** int interface);
Description	:	This function claims an interface for a given device handle. You must claim an interface before performing I/O operations on the device.
Parameters	:	cyusb_handle *handle : The libusb device handle int interface : The bInterfaceNumber of the interface you wish to claim.
Return Value	:	0 on success, or an appropriate LIBUSB_ERROR
Prototype	:	**int cyusb_claim_interface(cyusb_handle * handle,** int interface);
Description	:	This function claims an interface for a given device handle. You must claim an interface before performing I/O operations on the device.
Parameters	:	cyusb_handle *handle : The libusb device handle int interface : The bInterfaceNumber of the interface you wish to claim.
Return Value	:	0 on success, or an appropriate LIBUSB_ERROR

Prototype	:	**int cyusb_release_interface(cyusb_handle * handle, int interface);**
Description	:	This function releases an interface previously claimed for a given device handle. You must release all claimed interfaces before closing a device handle. This is a blocking function, where a standard SET_INTERFACE control request is sent to the device, resetting interface state to the first alternate setting.
Parameters	:	cyusb_handle *handle: The libusb device handle int interface : The bInterfaceNumber of the interface you wish to release
Return Value	:	0 on success, or an appropriate LIBUSB_ERROR

Prototype	:	**int cyusb_set_interface_alt_setting(cyusb_handle * handle, int interface, int altsetting);**
Description	:	This function activates an alternate setting for an interface. The interface itself must have been previously claimed using cyusb_claim_interface.This is a blocking funcion, where a standard control request is sent to the device.
Parameters	:	cyusb_handle *handle: The libusb device handle int interface : The bInterfaceNumber of the interface you wish to set. int altsetting : The bAlternateSetting number to activate
Return Value	:	0 on success, or an appropriate LIBUSB_ERROR

Prototype	:	**int cyusb_clear_halt(cyusb_handle * handle, unsigned char endpoint);**
Description	:	This function clears a halt condition on an endpoint. Endpoints with a halt condition are unable to send/receive data unless the condition is specifically cleared by the Host. This is a blocking funcion.
Parameters	:	cyusb_handle *handle : The libusb device handle unsigned char endpoint : The endpoint for which the clear request is sent.
Return Value	:	0 on success, or an appropriate LIBUSB_ERROR

Prototype : **int cyusb_reset_device(cyusb_handle * handle);**

Description : This function performs a USB port reset to the device.
This is a blocking funcion.

Parameters : cyusb_handle *handle : The libusb device handle

Return Value : 0 on success, or an appropriate LIBUSB_ERROR

Prototype : **int cyusb_kernel_driver_active(cyusb_handle * handle,**
int interface);

Description : This function returns whether a kernel driver has already claimed an interface.

If a kernel driver is active and has claimed an interface, cyusb cannot perform I/O operations on that interface unless the interface is first released.

Parameters : cyusb_handle *handle: The libusb device handle
int interface : The interface which you are testing.

Return Value : 0 if no kernel driver is active, 1 if a kernel driver IS active or an appropriate error.

Prototype : **int cyusb_detach_kernel_driver(cyusb_handle * handle,**
int interface);

Description : This function detaches a kernel mode driver (in order for cyusb to claim the interface)

If a kernel driver is active and has claimed an interface, cyusb cannot perform I/O operations on that interface unless the interface is first released.

Parameters : cyusb_handle *handle: The libusb device handle
int interface : The interface which you want to be detached.

Return Value : 0 on success, or an appropriate LIBUSB_ERROR.

Prototype : **int cyusb_attach_kernel_driver(cyusb_handle * handle,**
int interface);

Description : This function reattaches a kernel mode driver which was previously detached

Parameters : cyusb_handle *handle : The libusb device handle
int interface : The interface which you want to be reattached.

Return Value : 0 on success, or an appropriate LIBUSB_ERROR.

Prototype : **int cyusb_get_device_descriptot(cyusb_handle * handle,**
struct libusb_device_descriptor *);

Description : This function returns the usb device descriptor for the given device.

Parameters : cyusb_handle *handle : The libusb device handle
struct libusb_device_descriptor *desc: Address of a device_desc structure

Return Value : 0 on success, or an appropriate LIBUSB_ERROR.The libusb_device_descriptor structure will contain detailed information if success.

Prototype : **int cyusb_get_active_config_descriptor(cyusb_handle * handle,**
struct libusb_config_descriptor **);

Description : This function returns the usb configuration descriptor for the given device.
Only valid if return value was 0.
Must be freed with cyusb_free_config_descriptor() explained below.

Parameters : cyusb_handle *handle : The libusb device handle
struct libusb_configuration_descriptor **desc: Address of a config_descriptor

Return Value : 0 on success, or an appropriate LIBUSB_ERROR.
The libusb_config_descriptor structure will contain detailed information if success.

Prototype : **int cyusb_get_config_descriptor(cyusb_handle * handle,**
unsigned char index,
struct libusb_config_descriptor **);

Description : This function returns the usb configuration descriptor for the given device.
Only valid if return value was 0.
Must be freed with cyusb_free_config_descriptor() explained below.

Parameters : cyusb_handle *handle : The libusb device handle
unsigned char index : Index of configuration you wish to retrieve.
struct libusb_configuration_descriptor **desc : Address of a config_descriptor

Return Value : 0 on success, or an appropriate LIBUSB_ERROR.
The libusb_config_descriptor structure will contain detailed information if success.

Prototype : **void cyusb_free_config_descriptor(**
struct libusb_config_descriptor *);

Description : Frees the configuration descriptor obtained earlier.

Parameters : struct libusb_config_descriptor * : The config descriptor you wish to free.

Return Value : NIL.

Prototype : **void cyusb_control_transfer(cyusb_handle *h,**

unsigned char bmRequestType,
unsigned char bRequest,
unsigned short wValue,
unsigned short wIndex,
unsigned char *data,
unsigned short wLength,
unsigned int timeout);

Description :	Performs a USB Control Transfer.	
Parameters :	cyusb_handle *h	: Device handle
	unsigned char bmRequestType	: The request type field for the setup packet
	unsigned char bRequest	: The request field of the setup packet
	unsigned short wValue	: The value field of the setup packet
	unsigned short wIndex	: The index field of the setup packet
	unsigned char *data	: Data Buffer (for input or output)
	unsigned short wLength	: The length field of the setup packet The data buffer must be at least this size.
	unsigned int timeout	: Timeout in milliseconds. For unlimited timeout, use 0.
Return Value :	0 on success, or an appropriate LIBUSB_ERROR.	

Prototype : **void cyusb_bulk_transfer(cyusb_handle *h,**
unsigned char endpoint,
unsigned char *data,
int length,
int *transferred,
int timeout);

Description :	Performs a USB Bulk Transfer.	
Parameters :	cyusb_handle *h	: Device handle
	unsigned char endpoint	: Address of endpoint to communicate with
	unsigned char *data	: Data Buffer (for input or output)
	unsigned short wLength	: The length field of the data buffer for read or write
	int * transferred	: Output location of bytes actually transferred

		unsigned int timeout	: Timeout in milliseconds. For unlimited timeout, use 0.
Return Value	:	0 on success, or an appropriate LIBUSB_ERROR.	

Prototype	:	**void cyusb_download_fx2 (cyusb_handle *handle, char *filepath , char vendor_command);**
Description	:	Downloads firmware on to Fx2 device.
Parameters	:	cyusb_handle *handle: Device handle
		filepath : Path for the FX2 firmware file.
		vendor_command : Vendor command specifying where to load the firmware.This normally needs to be 0xA0 as firmware is loaded to RAM.
Return Value	:	0 on success or an appropriate LIBUSB_ERROR

Prototype	:	**void cyusb_download_fx3(cyusb_handle *handle, char *filepath);**
Description	:	Downloads the firmware on to fx3 device
Parameters	:	cyusb_handle *handle: Device handle
		filepath : Path for the FX3 firmware file.
Return Value	:	0 on success or an appropriate LIBUSB_ERROR