CyUSB FX3/FX2 Suite for MAC

Programmers Guide

1. INTRODUCTION

The CyUSB Suite for MAC OS provides a framework for customers to program and perform data transfers from/to Cypress USB devices, from a MAC based USB host. The CyUSB suite consists of a library providing a set of convenience API to talk to generic USB devices; and a set of command line applications implemented using this library.

This document gives details on how to develop code using the CyUsb library for MAC.

Dependencies

The CyUsb library is built on top of the open source libusb-1.0 library. It is expected that the libusb library has been installed on the host machine before installation and usage of the CyUsb tools.

2. Developing Application

This section gives an overview of the steps involved in developing USB applications using the CyUSB library.

The cyusb.h header file under the $CyHome/include folder, and the libcyusb.dylib library provide the public interface and the functionality of the CyUSB library. Please refer to the cyusb_mac_userguide documet for the procedure to build the libcyusb.dylib library.

The cyusb.h header file defines one major data type called **struct cydev**, which is declared as follows:

```
struct cydev {
   cyusb_device *dev;        /* Pointer to a CyUsb device structure. */
   cyusb_handle *handle;     /* Handle to a CyUsb device. */
   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 size = 16 bytes */
};
```

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.**

2.1. Opening the Device

int cyusb_open (void);

This function identifies all devices of interest, creates and populates an array of cydev elements with the device information, and returns the number of devices of interest found. A device of interest is a device whose vendor ID/device ID is present in the /etc/cyusb.conf file.

This function is overloaded and a simpler alternative is

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

This function 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 specified.

The library supports multiple devices with the same VID and PID, and an array of devices will be created if multiple such devices are found. The user is then expected to traverse through cydev [] array and extract the handle for the appropriate device by matching with bus number and device address.

2.2. Obtaining the CyUSB Handle

cyusb_handle * cyusb_gethandle (int index);

This function returns a handle that can be used to do data transfers from/to the CyUSB device with the specified index. Since there will be only one device of interest connected to the host in most cases, the input parameter will usually be 0.

2.3. Closing the Device

void cyusb_close (void);

This function closes all cyusb devices of interest discovered.

2.4. Determine if a KERNEL Driver is bound to the Device

int cyusb_kernel_driver_active (cyusb_handle *, int interface);

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

2.5. Detach the KERNEL driver bound to the Device

int cyusb_detach_kernel_driver (cyusb_handle *, int interface);

Detach a kernel mode driver for a usb device of interest. In case a device already has a kernel mode driver active, then this API allows one to detach the kernel mode driver. This call is normally followed by claiming the interface by a user mode application like CyUSB Suite.

int cyusb_attach_kernel_driver (cyusb_handle *, int interface);

Re-attach the Kernel mode driver that was previously bound to the device.

2.6. Claiming and releasing an interface:

int cyusb_claim_interface (cyusb_handle *h, int interface);

User mode applications such as CyUSB Suite for MAC, can only work after claiming an interface. Once the interface is successfully claimed by the driver, application now can interact with device by getting information of various descriptors and doing data transfer over endpoints of interest.

Refer to the sample applications in the package for examples.

3.CyUSB API GUIDE.

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 when the library is initialized using the cyusb_open call. The array would contain only 'devices of interest'; i.e devices whose Ids have been specified 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	:	This is an overloaded function that populates the cydev[] array with just one device 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 micro-frame 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 function, 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 function.
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 function.
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_descriptor(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. Note that this is a generic API that allows both read and write transfers on the control endpoint. The direction bit in the bmRequestType parameter should be set for READ (IN) transfers and cleared for WRITE (OUT) transfers.

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_control_read(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 READ transfer on the USB Control endpoint. This API is a wrapper to the cyusb_control_transfer function, and ensures that the direction bit in the bmRequestType parameter is set. It is not advisable to use this function when wLength is zero, because most hosts/devices do not handle an IN control transfer with no data properly.

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_control_write(cyusb_handle *h,**

```
void cyusb_control_write(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 WRITE transfer on the USB Control endpoint. This API is a wrapper to the cyusb_control_transfer function, and ensures that the direction bit in the bmRequestType parameter is cleared.

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