C++ API: Functions

API Functions descriptions

All these API functions have some parameters to set: Timeout, BufferSize and others.

The first set of API is a set of DLL functions mainly used to open/close/enumerate the TE USB FX2 module connected through host computer's USB ports; this API uses the Cypress C++ CyAPI.lib as a basis.

The second set of API is a set of DLL functions mainly used to communicate (data transfer using EP6 and EP8) between the host computer and the EZ-USB FX2 microcontroller endpoints; this API uses the Cypress C++ CyAPI.lib as a basis.

The third set of API is a single API function: TE_USB_FX2_SendCommand(). This function is able to send the API commands to the EZ-USB FX2 microcontroller endpoint EP1 and receive the response binary code using endpoint EP1.

Set	SW C++ API function exported
First	TE_USB_FX2_ScanCards()
	TE_USB_FX2_Open()
	TE_USB_FX2_Close()
Second	TE_USB_FX2_GetData_InstanceDriverBuffer()
	TE_USB_FX2_GetData()
	TE_USB_FX2_SetData_InstanceDriverBuffer()
	TE_USB_FX2_SetData()
Third	TE_USB_FX2_SendCommand()

Exported function list

Synchronous Functions

All these functions use a synchronous version for the data transfer (Cypress XferData() function). They perform synchronous (i.e. blocking) I/O operations and do not return until the transaction completes or the endpoint TimeOut has elapsed. A synchronous operation (aka blocking operation) is an operation that owns (in an exclusive way) resources and CPU until its job is done. A synchronous function monopolizes resources until its end, even during idle time.

If the program uses the synchronous XferData() function (both in C# with TE_USB_FX2_CyUSB.dll and C++ with TE_USB_FX2_CyAPI.dll), the array of data to transfer is the only one that is subdivided into packets (with packet length MaxPktSize = 512 byte) and scheduled over the USB buffer for data transmission. Until the data array is completely transferred, no other data array can be scheduled into packets over the USB, even if there is free packet time to be used by other data. The array of data passed to the XferData() function is the only owner of the USB bus until all data of this array are transferred (successfully or unsuccessfully). While using XferData() method, the OS will schedule the next XferData() only after the previous XferData() completes, leading to delay.

XferData() just calls asynchronous functions BeginDataXfer(), WaitForXfer() and FinishDataXfer() in sequence and does error handling accordingly. WaitForXfer() is the one which implements the timeout period for larger transfers. Cypress recommends the following: "You will usually want to use the synchronous XferData method rather than the asynchronous BeginDataXfer / WaitForXfer / FinishDataXfer approach.".

The API uses the synchronous version because it is more suitable to be included in a DLL and it is already fast. With synchronous version, the API functions are simpler to use.

BufferSize (also called XferSize)

BufferSize is the size of the buffer used in data/command transmission/reception of a ***single endpoint;*** the total buffer size is the sum of BufferSize of every endpoint used.

> ⚠ BufferSize has a strong influence on DataThroughput. If BufferSize is too small, the DataThroughput can be 1/3 to 1/2 of the maximum value (36-38 Mbyte/s for read and 25-28 Mbyte/s for write transactions). If the BufferSize has a large value (a roomy buffer), the application should be able to cope with the non-deterministic behavior of C# without losing packets.

See Data Transfer Throughput Optimization for some insights into this kind of influence.

PacketSize

PacketSize is the size of packets used in data/command transmission/reception of a ***single endpoint***.

> ⚠ PacketSize has also a strong influence on DataThroughput. If PacketSize is too small (512 byte for example) you can achieve very low data throughput (2.2 Mbyte/s) even if you use a large BufferSize (driver buffer size = 131,072 byte)

See section Data Transfer Throughput Optimization for further insights on this influence.

Timeout Setting

Timeout is the time that is allowed to the function for sending/receiving the data packet passed to the function; this timeout shall be large enough to allow the data/command transmission/reception. Otherwise the transmission/reception will fail.

TimeOut shall be set according to the following formula:

TimeOut (ms) = [DataLength / DataThroughput] + 1 ms.

Note: TimeOut is integer so you shall round up the result.

For write transactions, assume DataThroughput 20 Mbyte/s (it is lower than actual value, give some margin).

For read transactions, assume DataThroughput 30 Mbyte/s (it is lower than actual value, give some margin).

For SendCommand() assume DataThroughput 1 Mbyte/s (close to actual value).

These values have been verified for a Core i7 processor at 2.20 GHz with Microsoft Windows 7. Other configuration may require others value. An AMD Athlon II at 1.30 GHz with Microsoft Windows 7 might require much (e.g. two or three times) larger values. If your host computer is not highly responsive, you should set TimeOut to even larger values : e.g 20, 50, 200, 1000 ms (the less responsive the host computer is, the higher the recommended values shall be).