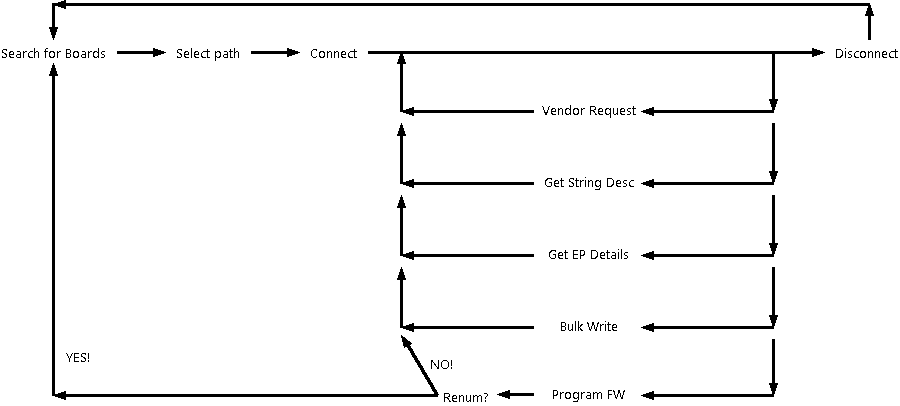
ADI\_CYUSB\_USB4.dll

May 10th, 2011

Program Flow Outline:



Function Descriptions:

* Search for Boards
  + IN : VID (unsigned int)
  + IN : PID (unsigned int)
  + IN : \*Number\_Boards (unsigned int)
  + IN : \*Path[], array to be filled (char [Number\_Boards])
  + OUT : Success/Error (int)

This function discovers how many devices on the system use the CyUSB.sys driver and then checks, one by one, if they match our ***PID*** *and* ***VID***. If so, ***Number\_Boards*** will be incremented. The path of the device will be stored in the ***Path[]*** array.

* Connect
  + IN : VID (unsigned int)
  + IN : PID (unsigned int)
  + IN : Path[] (char)
  + IN : \*Handle, to be filled (unsigned int)
  + OUT : Success/Error (int)

This function attempts to connect to the board at ***Path*** (checking that it still points to a device with relevant VID & PID) and returns ***Handle*** which refers to this connection. If the driver is already connected to a device, that connection is terminated (done automatically by the driver) before the new one is established.

ERR\_NO\_MATCHING\_DEVICE means a device was found but details don’t match (path or IDs)

ERR\_NO\_DEVICES\_FOUND means that no devices were found on the driver

* Disconnect
  + IN : Handle (unsigned int)
  + OUT : Success/Error (int)

This function disconnects from the board ***Handle ,*** and deletes the reference.

* DownloadFirmware
  + IN : Handle (unsigned int)
  + IN : Filepath[] (char[], null terminated)
  + OUT : Success/Error (int)

This function downloads the firmware file (\*.hex) located at ***Filepath*** to the board ***Handle.***

\*\*\*\*\*NOTE: If your download firmware performs a re-numeration, it will be necessary to search for boards and connect to the relevant board again afterwards! If you’re not sure whether this happens in your software or not, it probably doesn’t!

* Vendor Request
  + IN : Handle (unsigned int)
  + IN : Request (unsigned char)
  + IN : Value (unsigned short)
  + IN : Index (unsigned short)
  + IN : Direction (unsigned char)
  + IN : DataLength (unsigned short)
  + IN : Data array (unsigned char, filled or to be filled, direction dependent)
  + OUT : Success/Error (int)

This function performs a control transfer to the board ***Handle***. ***Direction*** is specified and ***Data Array*** is either be sent or filled accordingly. Return value of 0 indicates success.

* Get String Descriptor
  + IN : Handle (unsigned int)
  + IN : Data array (to be filled)
  + IN : String Index (unsigned char)
  + OUT : Success/Error (int)

This function performs a control transfer to the board. All fields are filled as necessary. Data Array should be big enough to contain your string descriptor and 2 additional bytes of data. DataArray[0] contains the length of the descriptor. DataArray[2] onwards contains the actual string.

* Get EP Info (and more!)
  + IN: Handle (unsigned int)
  + IN: \*Num\_EPs (unsigned char)
  + IN: \*Array[] (unsigned char)
  + OUT: Success/Error (int)

This function retrieves the number of endpoints (Num\_EPs) available at the *Handle* device. This number does not include the control endpoint (always present). The Array contains the configuration descriptor, the interface descriptor and each of the endpoint descriptors. See below for indepth description of the returned array.

* BulkTransfer
  + IN: Handle (unsigned int)
  + IN: USB\_Pipe\_Address (unsigned char)
  + IN: \*Length (unsigned int)
  + IN/OUT: \*Buffer[] (unsigned char)

This function performs a bulk transfer to/from the addresses pipe. For transfers from the device to the PC, the timeout is set to 1000ms although this is only used in transfers when the requested data isn’t available.

Descriptor Array

* Configuration Descriptor, 9 bytes
* Interface Descriptor, 9 bytes
* Endpoint Descriptor(s), 7 bytes per endpoint

|  |  |  |  |
| --- | --- | --- | --- |
| Index | Byte # | Configuration Descriptor | |
| 0 | 0 | 09 | Length of configuration descriptor |
| 1 | 1 | 02 | Descriptor Type |
| 2 | 2 | 2E | Length of data being returned - low bytes |
| 3 | 3 | 00 | Length of data being returned - high bytes |
| 4 | 4 | 01 | Num of interfaces supported in this configuration |
| 5 | 5 | 01 | Configuration Value |
| 6 | 6 | 00 | Configuration Index |
| 7 | 7 | 80 | Attributes |
| 8 | 8 | 32 | Max Power - in 2mA units |
| Index | Byte # | Interface Descriptor | |
| 9 | 0 | 09 | Length of interface descriptor |
| 10 | 1 | 04 | Descriptor Type |
| 11 | 2 | 00 | Interface number |
| 12 | 3 | 00 | Value used to select this alternate setting |
| 13 | 4 | 04 | Number of Endpoints, excluding EP0 |
| 14 | 5 | FF | Class code - FF Vendor Specific |
| 15 | 6 | 00 | Interface Subclass |
| 16 | 7 | 00 | Interface Protocol |
| 17 | 8 | 00 | Index of String Descriptor describing this interface |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Index |  | EP1 | EP2 | EP3 | EP4 |  |
| 18, 25, 32, 39 | 0 | 07 | 07 | 07 | 07 | Length |
| 19, 26, 33, 40 | 1 | 05 | 05 | 05 | 05 | Descriptor Type |
| 20, 27, 34, 41 | 2 | 02 | 04 | 86 | 88 | Endpoint address - MSB direction |
| 21, 28, 35, 42 | 3 | 02 | 02 | 02 | 02 | Attributes - 2 bulk |
| 22, 29, 36, 43 | 4 | 00 | 00 | 00 | 00 | Max Packet Size - low bytes |
| 23, 30, 37, 44 | 5 | 02 | 02 | 02 | 02 | Max Packet Size - high bytes |
| 24, 31, 38, 45 | 6 | 00 | 00 | 00 | 00 | Interval |

Important Array locations:

Array[2] = size of data returned (0x2E = d46)

Array[13] = Number of non-control endpoints

Array[20], Array[27], Array[34], Array[41] = EP addresses

[index = (i\*7) + 20)] for subsequent endpoints

Total array length is 18 + (#endpoints\*7)

The MSB of the address is a direction bit (1 = Device to PC, 0 = PC to device) and the lower byte is the endpoint number. The direction bit should be ignored for EP0, the control endpoint, as its direction is specified elsewhere.

