Please note that Cypress is an Infineon Technologies Company.
The document following this cover page is marked as “Cypress” document as this is the company that originally developed the product. Please note that Infineon will continue to offer the product to new and existing customers as part of the Infineon product portfolio.

Continuity of document content
The fact that Infineon offers the following product as part of the Infineon product portfolio does not lead to any changes to this document. Future revisions will occur when appropriate, and any changes will be set out on the document history page.

Continuity of ordering part numbers
Infineon continues to support existing part numbers. Please continue to use the ordering part numbers listed in the datasheet for ordering.



THIS SPEC IS OBSOLETE

Spec No: 001-14557

Spec Title: INTRODUCTION TO CYUSB.DLL BASED APPLICATION DEVELOPMENT USING VC# - AN14557

Sunset Owner: Nikhil Naik – (nikl)

Replaced by: NONE

AN14557

Introduction to CyUSB.dll Based Application Development Using VC#

Author: Amit Nanda/ Greg Nalder

Associated Project: Yes

Associated Part Family: CY7C68013/ CY7C68013A

Software Version: SuiteUSB 3.4.x, Microsoft Visual Studio 2008/ 2010

Related Application Notes: None

OBSOLETE

Abstract

SuiteUSB 3.4 is a .NET application development library from Cypress to create Windows Applications using Microsoft Visual Studio. AN14557 includes a history and guides you to write your first USB application on the Microsoft Visual C# platform using Cypress Suite USB C# library, CyUSB.dll. This document is primarily meant for beginners in Windows Application and CyUSB.dll.

Contents

Introduction 1
Cypress SuiteUSB 3
System Requirements 3
  Hardware 3
  Software 3
Writing Your First Application 3
  Application Code Analysis 5
Additional Features in the Application 6
  Add Buttons and Toggle a 7-Segment Display 7
  Detecting Devices 7
Summary 8
Advanced Examples 8
Additional Resources 8
Appendix 9
Document History 11
Worldwide Sales and Design Support 12

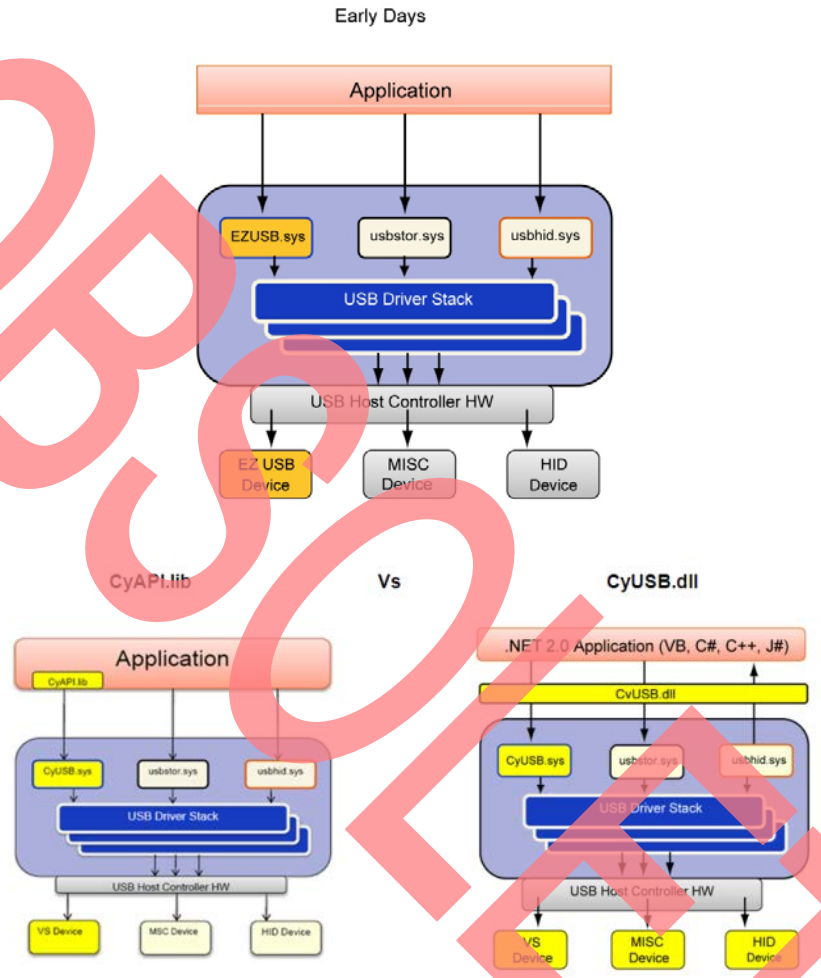
Introduction

Application communication with USB devices have evolved. Earlier, the application writing process was complex and involved making direct calls to drivers. The application had to first get a device handle and then call device I/O controls or read/write files. This made access, especially to mass storage class devices, very difficult.

In that initial development, Cypress released USB Developers' µStudio, which provided an easier, high level programming interface through which there was no need to get device handles. The API was implemented as a statically linked library. However, it was limited because it only accessed devices bound to the *cyusb.sys* driver.

The new generation of application development tools from Cypress has a simpler, more powerful API. SuiteUSB 3.4 supports the *cyusb.sys*, *usbstor.sys*, and *usbhid.sys* device drivers increasing the spectrum of devices that can be accessed with this tool. Figure 1 on page 2 illustrates a comparison of the earlier application development with the new Suite USB.net.

Figure 1. Early Days of Application Development Vs. the New Suite USB.net



OBSOLETE

Cypress's new SuiteUSB.net enables users to quickly develop custom USB applications. At the heart of the new SuiteUSB.net is c*yusb.dll*. This DLL is a managed Microsoft .NET class library. It provides a high level, powerful programming interface to USB devices. Instead of communicating with USB device drivers directly through Win32 API calls (such as SetupDiXxxx and DeviceIoControl), applications can access USB devices through library methods such as XferData and properties such as AltIntfc.

Because *cyusb.dll* is a managed .NET library, its classes and methods can be accessed from any of the Microsoft Visual Stuido.NET managed languages such as Visual Basic.NET, C#, Visual J#, and Managed C++. To use the library, you must add a reference to *cyusb.dll* to your project's References folder. Then, any source file that accesses the CyUSB name space must include a line to add the name space in the appropriate syntax.

The following section explains how to develop your first application with SuiteUSB. The following examples are written in C# but can be converted to C++ or J#.

Note Even though CyUSB.NET APIs can be accessed from any of the Visual Studio.NET Managed Languages, currently there are no examples on Visual Basic and Visual J#.

Cypress SuiteUSB

The host application here is developed using Cypress SuiteUSB C# library (CyUSB.dll) that comes with Cypress SuiteUSB. Cypress provides SuiteUSB, which is a set of development tools for Visual Studio to create .NET Windows applications. SuiteUSB.NET 3.4 includes the following:

- A Generic USB Device Driver
- A .NET Managed Class Library that has
  - CyUSB.dll, which is a C# library
  - CyAPI.lib, which is a C++ library
- USB Control Center: that serves as a USB experimenter's work-bench
- Sample Codes.

For more details, visit SuiteUSB 3.4.

Note Cypress Suite USB and Cypress USB driver CyUSB.sys are compatible only with Windows 2000, XP, Vista, and Windows 7. For users looking for USB applications on MAC/Linux, there are examples that show how to communicate with Cypress USB devices using the generic open source driver LIBUSB. Refer to the following if you are interested in MAC/ Linux Applications:

- MAC Users: EZ-USB® FX2LP™ - Developing USB Application on MAC OS X using LIBUSB
- Linux Users: EZ-USB® FX2LP™/ FX3™ Developing Bulk-Loop Example on Linux

System Requirements

Hardware

A FX2LP DVK (CY3684) board is used as the development and testing platforms for this example. A detailed schematic of the DVK can be found at the location "C:\Cypress\USB\Hardware\FX2LP" after installing FX2LP DVK. 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).

Software

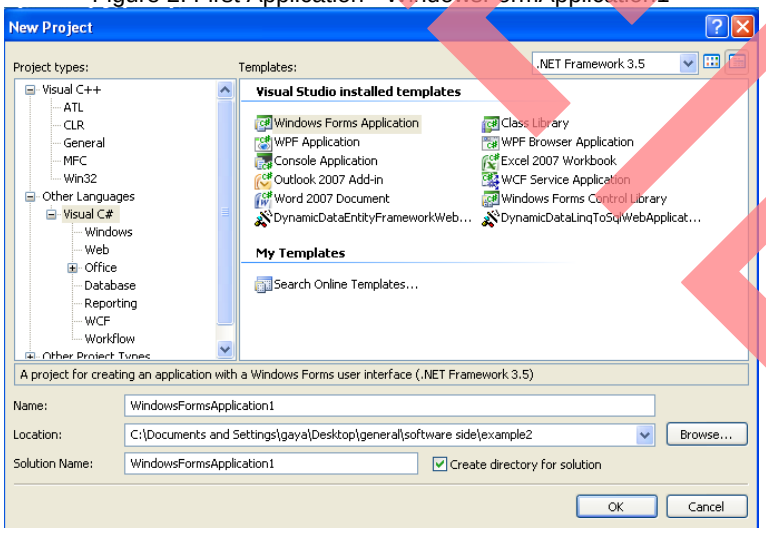
1. Cypress SuiteUSB 3.4
2. Microsoft Visual Studio 2008.

Writing Your First Application

Before you begin writing your first application, ensure you have installed SuiteUSB 3.4 and Visual Studio 2008. The following steps guide you to develop your first VC# application using CyUSB.dll.

1. Start Visual Studio 2008 and choose File > New Project > Windows Form Application.
2. In the window that pops up, make sure you give your application a unique name. In this example, the application name is 'WindowsFormApplication1' as shown in Figure 2.

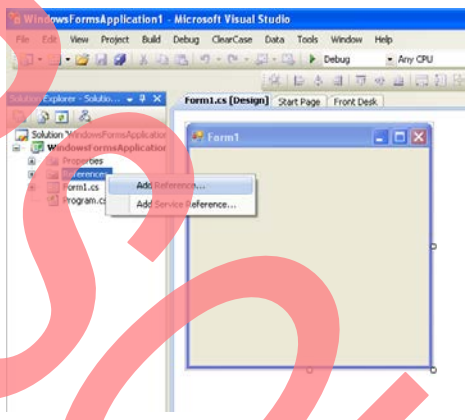
Figure 2. First Application - WindowsFormApplication1



OBSOLETE

3. Click **OK** and a blank form displays. This form is a functional application. Click on the green arrow (Run button) to start the application. The blank form appears as you start the application.
4. There are many interesting things you can do with *CyUSB.dll*. To use this library, you must add a reference to *CyUSB.dll* to your project's References folder. Any source file that accesses the CyUSB name space must include a line to include the name space in the appropriate syntax.
5. For that, right click on **Reference** and **Add Reference**, under the Solution Explorer window as in Figure 3. Select the Browse tab from the window that pops up and browse to the installation directory of SuiteUSB (`C:\Cypress\Cypress Suite USB 3.4.x\CyUSB.NET\lib`) and double-click *CyUSB.dll*. This references the library to your project. However, you cannot use it just yet.

Figure 3. Adding reference to CyUSB.dll



6. If you see the blank form, right click outside in the white space and click View Code. This is your code view window. Note that Microsoft enters initial code to start your project.
7. At the top, there are directives, starting with the word 'Using'. For C/C++ users, these are the same as '#include'. Since you added the reference to CyUSB.dll, you must inform the application that it is going to be used. Under the last 'using' line, add the words: using CyUSB; this gives you access to the library's APIs, classes, and other functionality.
8. Actually, the 'using' directive serves as a shortcut in the code so that you do not have to explicitly reference the CyUSB name space when you use its functions. You can leave out the 'using' directive if you affix 'CyUSB' to all the CyUSB member references.
9. Insert the following code in your application (refer to Application Code Analysis).

```
1.    USBDeviceList usbDevices;
2.    CyUSBDevice myDevice;
3.    public Form1()
4.    {
5.       InitializeComponent();
6.       usbDevices = new USBDeviceList(CyConst.DEVICES_CYUSB);
7.       usbDevices.DeviceAttached +=new EventHandler(usbDevices_DeviceAttached);
8.       usbDevices.DeviceRemoved += new EventHandler(usbDevices_DeviceRemoved);
9.
10.     // Get the first device having VendorID == 0x04B4 and ProductID == 0x1003
11.      myDevice = usbDevices[0x04B4, 0x1003] as CyUSBDevice;
12.   }
13.   void usbDevices_DeviceAttached(object sender, EventArgs e)
14.   {
15.      //Add code to handle Device arrival
16.   }
17.   void usbDevices_DeviceRemoved(object sender, EventArgs e)
18.   {
19.      //Add code to handle Device removal
20.   }
```

OBSOLETE

Application Code Analysis

Before we analyse the previous code, note that you can find more details about the CyUSB.dll APIs in the API guide (CyUSB.NET.chm or CyUSB.NET.pdf) at `C:\Cypress\Cypress Suite USB 3.4.x\CyUSB .NET` after installing SuiteUSB 3.4.

An application normally creates an instance of the USBDeviceList class, which represents a list of USB devices. Thus a good working knowledge of the USBDeviceList class is essential. The USBDeviceList represents a dynamic list of USB devices that are accessible through the class library. When an instance of USBDeviceList is created, it populates itself with USBDevice objects representing all the USB devices served by the indicated device selector mask, such as (line 6) in the previous code snippet:

```
usbDevices = new USBDeviceList
(CyConst.DEVICES_CYUSB);
```

Selector masks are defined as follows:

1. CyConst.DEVICES_CYUSB – mask to select devices that are bound to the CyUSB driver.
2. CyConst.DEVICES_HID – mask to select devices that are in the HID class.
3. CyConst.DEVICES_MSC – mask to select devices that are in the MSC class.

These USBDevice objects are all properly initialized and are ready for use. After an instance of the USBDeviceList class is constructed, the USBDeviceList index operators make it easy to locate a particular device and begin using it, such as (line 11) in the previous code snippet:

```
CyUSBDevice myDevice = usbDevices[0x04B4,
0x1003] as CyUSBDevice;
```

If you want to search for other methods of indexing through the device list, type: **CyUSBDevice myDev = usbDevices[**. After you type in the open bracket, Visual Studio displays different methods of using the USBDeviceList index. Because USBDeviceList implements the IEnumerable interface, you iterate through a USBDeviceList object's items using the 'foreach' keyword.

Windows PlugNPlay (PnP) event handling for the devices in a USBDeviceList are also supported by the library. To enable handling of PnP events,USBDeviceList provides two event handlers DeviceAttached and DeviceRemoved.

DeviceAttached - When a new USB device is plugged into the bus, the arrival of a new USB device is detected by this event. Handling of the event requires that an EventHandler object be assigned to the DeviceAttached event handler.

```
usbDevices.DeviceAttached+=new
EventHandler(usbDevices_DeviceAttached);
```

The previous line of code assigns `usbDevices_DeviceAttached` as the event handler for the DeviceAttached Event. Any action to be taken on arrival of the USB device should be done within the following event handler function

```
void usbDevices_DeviceAttached(object
sender, EventArgs e)
```

DeviceRemoved - When a USB device is disconnected from the bus, the removal of the USB device is detected by this event. Handling of the event requires that an EventHandler object be assigned to the DeviceRemoved event handler.

```
usbDevices.DeviceRemoved += new
EventHandler(usbDevices_DeviceRemoved);
```

The above line of code assigns `usbDevices_DeviceRemoved` as the event handler for the DeviceRemoved Event. Any action to be taken on removal of the USB device should be done within the following event handler function:

```
void usbDevices_DeviceRemoved(object
sender, EventArgs e)
```

You can write something in both the Event Handlers and then test the code.

1. Inside the `usbDevices_DeviceRemoved` event handler, type the following:**Text = "Device Removed";**
2. Inside the `usbDevices_DeviceAttached` event handler, type the following: **Text = "Device Attached";**

The 'Text' property controls the text seen in the top left hand corner when you run your application. For example, if you run the application without the above mentioned code, the word 'Form1' is displayed – not very informative information. By adding this code every time a device is plugged in or removed, the software displays the text provided.

3. Press the green **Play** button and attach and detach a USB device.

Make sure it is a Cypress USB device because the event handlers you just wrote only handles devices tied to the CyUSB driver, for now.

4. Unplug and plug the device repeatedly and watch the text change. That is your first application.

The next few sections discuss features used to make the application more productive.

OBSOLETE

Additional Features in the Application

Before proceeding to the next topic, a more detailed discussion about *CyUSB.dll* is required. One of the most important classes in the library is CyUSBDevice. The CyUSBDevice class represents a USB device attached to the *CyUSB.sys* device driver. A list of CyUSBDevice objects are generated by passing DEVICES_CYUSB mask to the USBDeviceList constructor. After you obtain a CyUSBDevice object, you can communicate with the device through the objects' various endpoint (ControlEndPt, BulkInEndPt, BulkOutEndPt, and others) members. Because CyUSBDevice is a descendant of USBDevice, it inherits all the members of USBDevice.

CyUSBDevice provides three main components (an in depth list is located in the *SuiteUSB Programmers Reference Guide*).

1. Functions (in C# parlance, these are called methods)
2. Properties
3. Objects

These three components give you access to most of the USB controls you need in your application, including functions such as GetDeviceDescriptor() and Reset(); properties such as AltIntfc and ConfigCount; and objects such as BulkIn/Out Endpt and IsocEndpt.

The first application you wrote allowed you to detect plug and play events and change the text of the application. You can add some buttons to your form and experiment with some alternate interfaces. The next example uses the EZ-USB® FX2LP™.

1. In this example, we use a specific fimrware example to demonstrate advanced features in CyUSB.dll. We will use th e CyStream firmware, which is located at `C:\Cypress\Cypress Suite USB 3.4.x\Firmware\CyStreamer` after installing SuiteUSB 3.4.
2. Next, you must download the *CyStream.iic* file to the EEPROM of the EZ-USB FX2LP.
3. For that, first connect the FX2LP DVK with the "EEPROM ENABLE" switch in 'No EEPROM' position to the PC.
4. It enumerates with the default internal descriptor. Use the appropriate CyUSB.inf file in the "Drivers" folder to bind with the device. For more information on binding the driver, see MatchingDriverToUSBDevice section in CyUSB.chm at `C:\Cypress\Cypress Suite USB 3.4.x\Driver` after installing SuiteUSB 3.4 or the link Drivers for FX1/FX2LP to bind with the device.
5. Change the 'EEPROM ENABLE' switch position to "EEPROM" position, and the EEPROM SELECT switch to the LARGE EEPROM position on the device.
6. Open the Control Center application present at 'Start > Programs > Cypress > Cypress Suite USB 3.4.x > Control Center'. Download the CyStream.iic from `C:\Cypress\Cypress Suite USB 3.4.x\Firmware\CyStreamer` to the large EEPROM present on the FX2LP DVK using the Control Centre utility.
7. Reset the FX2LP and the device then enumerates running the CyStream firmware. Let us call it the CyStream device.
8. If Windows pops up asking you to bind the driver, repeat the steps explained in Step 4.
9. The CyStream example has many Alternate settings for interface. The selected Alternate setting is displayed on the 7-segment display on the FX2LP DVK. For more information on the CyStream example, refer to the source code CyStream.uV2, which you can find at `C:\Cypress\Cypress Suite USB 3.4.x\Firmware\CyStreamer`.
10. Now, add an event handler on your application to select the Alternate setting of the CyStream device.

OBSOLETE

Add Buttons and Toggle a 7-Segment Display

1. Click on the tab in Visual Studio that reads **Form1.cs[Design]**.
2. Click on **View** > **Toolbox**.
3. When the Toolbox opens, click and drag **Button** anywhere on your application.
4. A button labeled Button1 appears on your form. Double-click **Button**.
5. Windows creates an event handler. Any time you click the button, your program does what is inside this function call.

   ```
   private void button1_Click(object sender, EventArgs e)
   ```

   `object sender` – where the event came from. If you have multiple functions calling this function you can determine where it came from.

   `EventArgs e` – any arguments that are passed in when the event happens.

6. Add code to control the 7-segment display. In your function, type the following:

   ```
   myDevice.AltIntfc++;
   Text = myDevice.AltIntfc.ToString();
   ```

   AltIntfc is a property that is used to get or set the alternate interface settings for the device. Therefore, when you click on the button1, you set the next alternate interface setting on CyStream device. Since the 7-segment display shows the current alternate interface setting, this code increments the numbers on display. At the same time, the text in your application outputs what is currently displayed on 7-segment. You can use this code in an application, such as a thermometer, to display the temperature of a device on screen if the device is hidden inside a box.

7. Run your application.
8. For a robust application, enable the button only when the CyStream device is detected. To do this, add the following code inside `usbDevices_DeviceAttached` event handler and inside Form1() constructor. For details, refer to the Appendix section.

   ```
   myDevice = usbDevices[0x04B4,
   0x1003] as CyUSBDevice;
    if (myDevice != null)
      button1.Enabled = true;
   ```

9. Add the following code to the event handler, `usbDevices_DeviceRemoved`:

   ```
   myDevice = usbDevices[0x04B4,
   0x1003] as CyUSBDevice;
     if (myDevice == null)
       button1.Enabled = false;
   ```

10. To disable the button1 by default, add the following line inside the Form1() constructor before writing the code mentioned in step 8 inside Form1() constructor:

    ```
    button1.Enabled = false;
    ```

For exact positioning of the code statements in various finctions, refer to the Appendix section.

11. Now, try to implement a button to decrement the count.

Detecting Devices

This section explains how to create a tree view that displays the currently connected USB devices. The following code generates an application that detects all devices connected to the bus.

1. Move the buttons to one side of your form.
2. Drag and drop **TreeView** onto the form and expand it to take up most of the room on the form.
3. Right click in the white area outside of your form and click **View Code**.
4. Add code to the function: Form1()

   ```
   foreach (USBDevice dev in usbDevices)
   treeView1.Nodes.Add(dev.Tree);
   ```

   After adding just two lines of code, the program gets all devices connected to the host and fills in the tree when the application starts. Every USB device has a Tree property associated with it. When you call *dev.Tree*, the configuration of all the devices is returned and displayed.

5. Add the following code to both the PnP event handlers. Before doing this, you have to add one more line to ensure repeat devices do not show up in the tree, that is, to clear the treeview first and then populate it with the fresh list of devices. Add the following code under both `usbDevices_DeviceAttached` and `usbDevices_DeviceRemoved` event handlers:

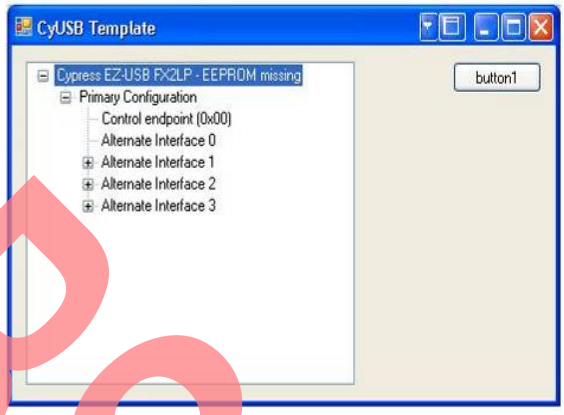
   ```
   treeView1.Nodes.Clear();

   foreach (USBDevice dev in usbDevices)
   treeView1.Nodes.Add(dev.Tree);
   ```

   This clears the tree every time a device is plugged or unplugged and then it re-initializes the tree. Your view should look similar to Figure 4.

OBSOLETE

Figure 4. CYUSB Template



After you perform the previous exercises, try writing and testing your own applications. Experiment with different properties and tools and see if you can write an application that meets your requirements.

Summary

SuiteUSB.net enables users to quickly develop custom USB applications. This application note explained how to write a simple application on VC# using CyUSB.dll. You can use this application as a stepping stone to develop real world applications involving data transfer to and from Cypress devices. Refer to the sections, Advanced Examples and Additional Resources, for more details.

Advanced Examples

Now that you know how to write simple application on VC# using CyUSB.dll, you may want to know how to develop more practical applications to transfer data to and from device. Here are the other examples:

EZ-USB FX2LP™ Bulk Transfer Application in C# Using SuiteUSB C# Library (CyUSB.dll)

More application examples are also provided along with Cypress SuiteUSB at this location:

`C:\Cypress\Cypress SuiteUSB3.4.x\CyUSB.NET\examples` after installing SuiteUSB.

Additional Resources

- Cypress CyUSB .NET DLL Programmer's Reference: For more information on CyUSB.dll
- Getting Started with FX2LP™
- Introduction to CyAPI.lib Based Application Development Using VC++ **-** Helps in getting started with developing host applications in VC++ using CyAPI.lib
- EZ-USB® FX2LP™ Host Application in VC++ 2008 Using Suite USB Library (CyUSB.dll) – advanced example on developing application on VC++ using CyUSB.dll

OBSOLETE

Appendix

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using CyUSB;

namespace windowsFormsApplication1
{
    public partial class Form1 : Form
    {
        USBDeviceList usbDevices;
        CyUSBDevice myDevice;
        public Form1()
        {
            InitializeComponent();
            usbDevices = new USBDeviceList(CyConst.DEVICES_CYUSB);
            usbDevices.DeviceAttached += new EventHandler(usbDevices_DeviceAttached);
            usbDevices.DeviceRemoved += new EventHandler(usbDevices_DeviceRemoved);

            // Get the first device having VendorID == 0x04B4 and ProductID == 0x1003
           button1.Enabled = false;
           myDevice = usbDevices[0x04B4, 0x1003] as CyUSBDevice;
           if (myDevice != null)
                button1.Enabled = true;

            foreach (USBDevice dev in usbDevices)
                treeView1.Nodes.Add(dev.Tree);

        }
        void usbDevices_DeviceAttached(object sender, EventArgs e)
        {
            //Add code to handle Device arrival
            Text = "Device Attached";
             myDevice = usbDevices[0x04B4, 0x1003] as CyUSBDevice;
            if (myDevice != null)
                button1.Enabled = true;

            treeView1.Nodes.Clear();
            foreach (USBDevice dev in usbDevices)
                treeView1.Nodes.Add(dev.Tree);

        }
        void usbDevices_DeviceRemoved(object sender, EventArgs e)
        {
            //Add code to handle Device removal
            Text = "Device Removed";
            myDevice = usbDevices[0x04B4, 0x1003] as CyUSBDevice;
            if (myDevice == null)
                button1.Enabled = false;

            treeView1.Nodes.Clear();
            foreach (USBDevice dev in usbDevices)
```

OBSOLETE

```
                treeView1.Nodes.Add(dev.Tree);
        }

        private void button1_Click(object sender, EventArgs e)
        {
            myDevice.AltIntfc++;
            Text = myDevice.AltIntfc.ToString();
        }

    }

}
```

OBSOLETE

Document History

Document Title: Introduction to CyUSB.dll Based Application Development Using VC# - AN14557

Document Number: 001-14557

Revision	ECN	Orig. of Change	Submission 14557Date	Description of Change
**	967160	GBN	04/13/07	New application note
*A	3026250	CPPK	09/09/10	Changed the title and added link to the DLL Programmer's Reference.
*B	3088800	CPPK	11/17/10	Changed title to "Getting Started With SuiteUSB Applications Using C#".
*C	3186856	CPPK	03/03/11	Updated the title and the abstract.
*D	3223474	CPPK	04/12/11	Minor Template Updates
*E	3240993	CPPK	04/26/11	Updated Code to comply with SuiteUSB 3.4.4. Updated screenshots of software from Visual Studio 2005 from Visual studio 2008.
*F	3600853	GAYA	04/30/ 2012	Updated template. Fixed links to figures and other documents, added references to advanced examples. Added Appendix with entire code for reference. Attached project files for VS 2008/VS 2010.
*G	3912984	NIKL	04/09/2013	Obsolete document.

OBSOLETE

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
Optical Navigation Sensors	cypress.com/go/ons
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 5

Cypress Developer Community

Community | Forums | Blogs | Video | Training

Technical Support

cypress.com/go/support

OBSOLETE

All trademarks or registered trademarks referenced herein are the property of their respective owners.

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

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

© Cypress Semiconductor Corporation, 2007-2013. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

This Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.