# STM32 WinUSB(WCID)免驅高速通信 20M/s（附詳細測試）

Bingwueyi 於 2021-11-29 23:32:08 發布

原文链接：<https://blog.csdn.net/bingwueyi/article/details/121622001>

//---------------------我是廣告-----------------------

詳細配置及參考工程請移步淘寶店：https://item.taobao.com/item.htm?spm=a230r.1.14.11.18cc4864SdV6xL&id=660992604144&ns=1&abbucket=20#detail

//---------------------我是廣告-----------------------

## 1 概述

### 1.1 WCID 簡介

WCID全稱為“WINDOWS Compatible ID”，是一種可提供額外的信息，在Windows系統中，為了便於驅動程序自動安裝，並在某些情況下，允許即時訪問的USB設備。

WCID允許Windows應用程序幾乎在插入後立即使用設備，這既不是HID也不是大容量存儲的USB設備需要最終用戶執行手動驅動程序安裝。因此，WCID可以做到像HID和大容量存儲USB設備那樣“即插即用”。WCID是WinUSB設備功能的擴展，由Microsoft在Windows8開發人員預覽版期間提出，WinUSB自Windows XP SP2以來一直是Windows一部分的功能。

WinUSB設備是一種通用串行總線(USB)設備，其固件定義了某些Microsoft操作系統(OS)特徵描述符，這些描述符將兼容ID報告為"WINUSB"。WinUSB設備的用途是讓Windows將Winusb.sys作為設備的功能驅動程序載入，而無需自定義INF文件。對於WinUSB設備，你無須為設備分發INF文件，這大大簡化了驅動程序安裝過程。截至2012年5月，從Windows Vista開始的所有平台上都提供了自動 WinUSB WCID 驅動程序。在 Windows 8 或更高版本地系統上已經集成了該驅動程序，這意味著Windows 8 或更高版本地系統上不需要安裝驅動程序，系統自動會匹配WinUSB驅動，而對於Vista、Windows7、XP系統，可以通過一些第三方軟件（如Zadig）來安裝或生成WCID版本的驅動程序，同樣只需要安裝一次WCID版本的驅動軟件，以後所有WCID設備都可以使用該驅動，間接實現了免驅。

在WCID之前，只有標準類別的USB設備，如HID（USB鍵盤、鼠標、遊戲桿等）或大容量存儲（U盤、基於閃存的存儲、媒體播放器等）不需要安裝驅動程序，因為Windows會在設備第一次插入時自動處理。而其他設備通常需要自己手動提供和安裝驅動程序，在WIN8以後廠家自己提供的驅動程序還可能受到微軟的驅動程序簽名限制，造成不必要的麻煩。

原文參考：https://github.com/pbatard/libwdi/wiki/WCID-Devices

### 1.2 WinUSB限制條件

WinUSB有許多限制條件，一般來說注意以下幾點即可：

1) WinUSB不支持Windows 2003 ( 32bit/64bit )；

2）WinUSB不支持Windows XP 64位（可以通過第三方驅動支持）；

3) USB設備版本必需是2.0或2.0以上，WinUSB不支持USB1.0和1.1。

原文參考：https://www.cnblogs.com/shangdawei/archive/2013/04/17/3026394.html

## 2 WinUSB設備枚舉關鍵信息

設備枚舉為WinUSB的三個關鍵信息：

1）OS字符串描述符；

2）擴展兼容ID OS功能描述符；

3）擴展屬性OS功能描述符；

### 2.1 支持OS字符串描述

當有USB設備插入，系統會向設備索取設備描述符，根據設備描述符中的bcdUSB版本號判斷是否檢索0xEE位置的OS設備字串描述符，如果bcdUSB版本號大於等於2.0，則系統會檢索0xEE位置的OS設備字串描述符，支持WinUSB的OS描述如下：

一張含有 桌 的圖片

自動產生的描述

如果0xEE處的OS描述符不存在或者描述格式不正確，系統將設置osvc值為0x0000，表示該USB設備不是WCID設備。如果OS描述正確，系統將osvc設置為0x01##。

### 2.2 設置兼容ID特徵描述符

在獲取到osvc為有效值的前提下，系統將進一步獲取兼容ID特徵描述符，索引號為0x04，兼容ID特徵描述符包含緊跟一個或多個功能部分的標題部分，具體取決於是否是復合設備。標題部分指定整個描述符的長度、功能部分的數量以及版本號。對於非復合設備，標題後緊跟一個僅與設備的接口關聯的功能部分。該部分的compatibleID字段必須指定"WINUSB"作為字段值。復合設備有多個功能部分。每個功能部分的compatibleID字段必須指定"WINUSB"。兼容ID特徵描述符格式如下：

一張含有 桌 的圖片

自動產生的描述

### 2.3 擴展屬性描述符

最後是添加擴展屬性描述符，註冊設備接口GUID，包含緊跟一個或多個自定義屬性部分的標題部分，索引號為0x05。標題部分描述整個擴展屬性描述符，包含其總長度、版本號以及自定義屬性部分的數量。WinUSB應用程序正是通過GUID來檢索對應的USB設備，並進行訪問，詳細定義如下表所示：

一張含有 桌 的圖片

自動產生的描述

## 3 USB枚舉過程

通過Keil調試可以獲取到USB枚舉的全部信息，如下圖所示。枚舉過程中共有40條交互信息：

一張含有 桌 的圖片

自動產生的描述

在Excel中統計如表格所示，第一列表示方向：1為Windows往設備發，0為設備回覆信息。

一張含有 桌 的圖片

自動產生的描述

1. 當Windows檢測到有USB設備插入，首先會向設備索取設備描述符長度，設備正確回復後Windows分配USB設備地址，並以之前收到的實際長度去獲取完整的設備描述符；
2. 獲取USB設備信息，包含基本配置、設備接口、端點等信息；
3. Windows判斷第一步獲取的bcdUSB版本號如果大於等2.0，則還會索引0xEE處的OS字符串描述符，確認設備是否支持WCID，若回復內容為MSFT100則說明該設備支持WCID，後續才會繼續獲取兼容ID特徵描述等；
4. 獲取兼容ID特徵描述符，回復兼容信息為WinUSB設備；
5. 獲取兼容ID後，系統會根據兼容ID的情況請求屬性描述符，對於WinUSB設備需要注冊設備接口GUID；
6. 後續其他信息獲取…

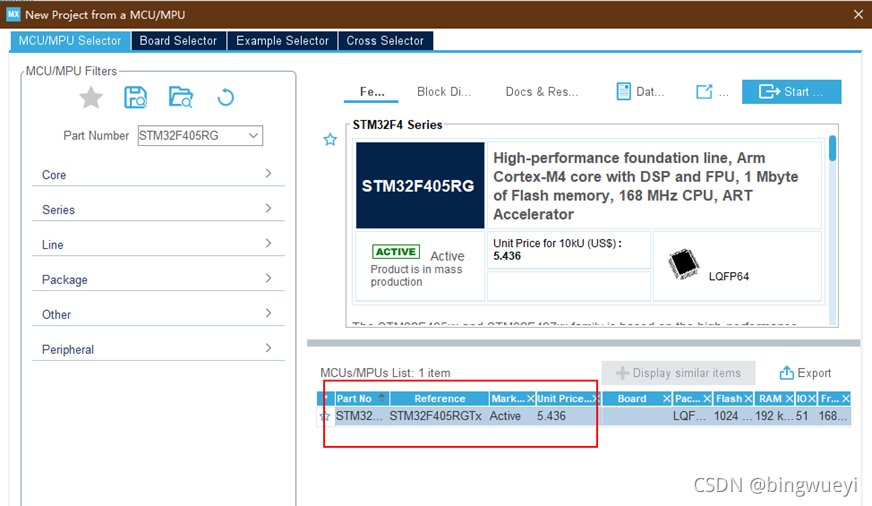
## 4 實例應用

### 4.1 生成USB 工程

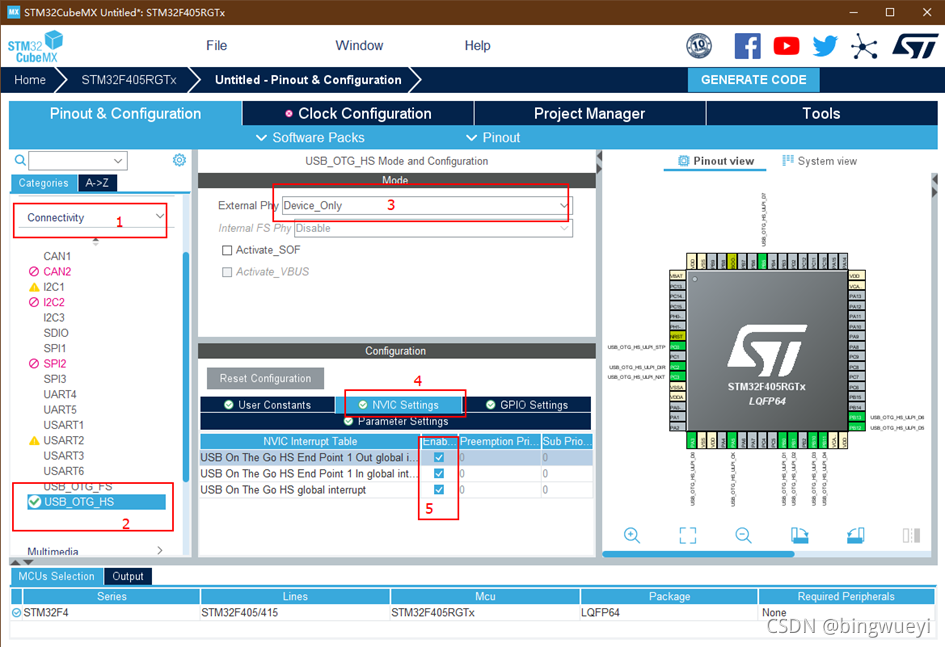
本實例採用STM32F405RGT6 + USBS3300 實現USB HS 高速WinUSB設備，使用STM32CubeMX生成基礎工程，CubeMX版本號為6.3，STM32F4 package版本為1.26.2，電腦系統為WIN10。

*備註：USB FS同樣按如下使用方法，需要注意的是USB FS只能用外部時鐘，且配置時需要注意外部時鐘頻率！*！

1）打開CubeMX選擇MCU型號STM32F405RGT6：



2）按如下順序在Connectivity中選中USB\_TG\_HS，然後設置External Phy為Device\_Only，並且將NVIC的三個中斷都勾選上：



3）按如下步驟在Middleware選項中選擇USB\_DEVICE，右側HS IP選擇Communication Device Class(Virtual Port Com)，然後在Device Descriptor中設置相關屬性：此處VID PID分別設置為0x1234、0xABCD（可自行更改，點擊輸框右側小齒輪將格式改為16進制顯示）；設置PRODUCT\_STRING，些處以GUOXUAN為例（可自行更改），這個字符串就是最終顯示在設備管理器中的名字；設置CONFIGURATION\_STRING 和 INTERFACE\_STRING，固定為WINUSB\_Config 和 WINUSB Interface。

一張含有 文字 的圖片

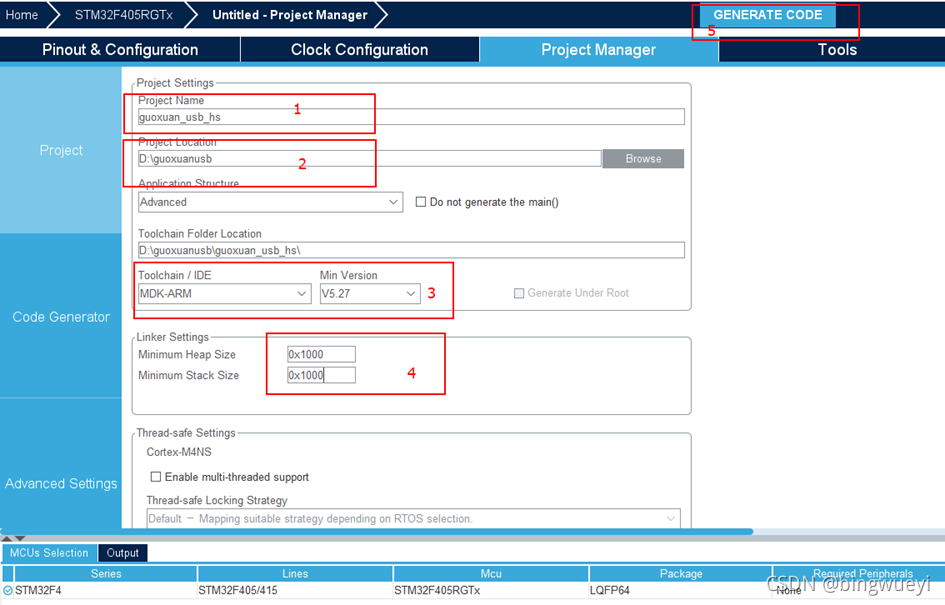
自動產生的描述

4）點擊下一頁Clock Configuration 直接點確定會自動配置好時鐘：

一張含有 桌 的圖片

自動產生的描述

5）點擊下一頁配置工程信息，適當將堆棧容量改大，避免一些意外問題，設置好後點擊右上角生成工程：



6）生成好後直接編譯工程，可以編譯成功：

一張含有 文字 的圖片

自動產生的描述

將程序燒入單片機後連接電腦，該USB設備被識別成了串口，並且沒有顯示出產品字符串，因為設備描述符沒有修改，設備被枚舉成了CDC串行通信類：

一張含有 文字 的圖片

自動產生的描述

### 4.2 修改工程文件

1. 增加兼容ID描述符和擴展屬性描述符定義，其中擴展描述符中的GUID定義可根據實際需求更改，本文以{12345678-ABCD-1234-ABCD-FEDCBA987654}為例：

|  |
| --- |
| //-----------ADD GUOXUAN----------------  #define USB\_LEN\_OS\_FEATURE\_DESC 0x28  \_\_ALIGN\_BEGIN uint8\_t USBD\_WINUSB\_OSFeatureDesc[USB\_LEN\_OS\_FEATURE\_DESC] \_\_ALIGN\_END =  {  0x28, 0, 0, 0, // length  0, 1, // bcd version 1.0  4, 0, // windex: extended compat ID descritor  1, // no of function  0, 0, 0, 0, 0, 0, 0, // reserve 7 bytes  // function  0, // interface no  0, // reserved  'W', 'I', 'N', 'U', 'S', 'B', 0, 0, // first ID  0, 0, 0, 0, 0, 0, 0, 0, // second ID  0, 0, 0, 0, 0, 0 // reserved 6 bytes  };  #define USB\_LEN\_OS\_PROPERTY\_DESC 0x8E  \_\_ALIGN\_BEGIN uint8\_t USBD\_WINUSB\_OSPropertyDesc[USB\_LEN\_OS\_PROPERTY\_DESC] \_\_ALIGN\_END =  {  0x8E, 0, 0, 0, // length 246 byte  0x00, 0x01, // BCD version 1.0  0x05, 0x00, // Extended Property Descriptor Index(5)  0x01, 0x00, // number of section (1)  //; property section  0x84, 0x00, 0x00, 0x00, // size of property section  0x1, 0, 0, 0, //; property data type (1)  0x28, 0, //; property name length (42)  'D', 0,  'e', 0,  'v', 0,  'i', 0,  'c', 0,  'e', 0,  'I', 0,  'n', 0,  't', 0,  'e', 0,  'r', 0,  'f', 0,  'a', 0,  'c', 0,  'e', 0,  'G', 0,  'U', 0,  'I', 0,  'D', 0,  0, 0,    // change by yourself  0x4E, 0, 0, 0, // ; property data length  '{', 0,  '1', 0,  '2', 0,  '3', 0,  '4', 0,  '5', 0,  '6', 0,  '7', 0,  '8', 0,  '-', 0,  'A', 0,  'B', 0,  'C', 0,  'D', 0,  '-', 0,  '1', 0,  '2', 0,  '3', 0,  '4', 0,  '-', 0,  'A', 0,  'B', 0,  'C', 0,  'D', 0,  '-', 0,  'F', 0,  'E', 0,  'D', 0,  'C', 0,  'B', 0,  'A', 0,  '9', 0,  '8', 0,  '7', 0,  '6', 0,  '5', 0,  '4', 0,  '}', 0,  0, 0,  };  //------------------------------ |

1. 增加OS字符串描述，固定為“MSFT100”，VENDOR\_CODE可自行更改；增加這三個描述的接口指針函數：

|  |
| --- |
| //-----------ADD GUOXUAN----------------  const uint8\_t USBD\_OS\_STRING[8] =  {  'M',  'S',  'F',  'T',  '1',  '0',  '0',  0xA0, //USB\_REQ\_MS\_VENDOR\_CODE,  };  uint8\_t \*USBD\_WinUSBOSStrDescriptor(uint16\_t \*length)  {  USBD\_GetString((uint8\_t \*)USBD\_OS\_STRING, USBD\_StrDesc, length);  return USBD\_StrDesc;  }  uint8\_t \*USBD\_WinUSBOSFeatureDescriptor(uint16\_t \*length)  {  \*length = USB\_LEN\_OS\_FEATURE\_DESC;  return USBD\_WINUSB\_OSFeatureDesc;  }  uint8\_t \*USBD\_WinUSBOSPropertyDescriptor(uint16\_t \*length)  {  \*length = USB\_LEN\_OS\_PROPERTY\_DESC;  return USBD\_WINUSB\_OSPropertyDesc;  }  //------------------------------------ |

1. 響應設備描述請求中的的的的USB\_REQ\_TYPE\_VENDOR請求：

|  |
| --- |
| case USB\_REQ\_TYPE\_VENDOR:  USBD\_WinUSBGetDescriptor( pdev, req ); //-----CHANGE GUOXUAN-------  break; |

1. 響應接口請求USBD\_StdItfReq中的**case USBD\_STATE\_CONFIGURED**：

|  |
| --- |
| case USBD\_STATE\_CONFIGURED  if (LOBYTE(req->wIndex) <= USBD\_MAX\_NUM\_INTERFACES)  {  ret = (USBD\_StatusTypeDef)pdev->pClass->Setup(pdev, req);  if ((req->wLength == 0U) && (ret == USBD\_OK))  {  (void)USBD\_CtlSendStatus(pdev);  }  }  //-----------------ADD GUOXUAN-------------  if ( req->bmRequest == 0xC1 )  {  USBD\_WinUSBGetDescriptor( pdev, req );  break;  }  //----------------------------------------  else  {  USBD\_CtlError(pdev, req);  }  break; |

1. 響應0xEE請求：

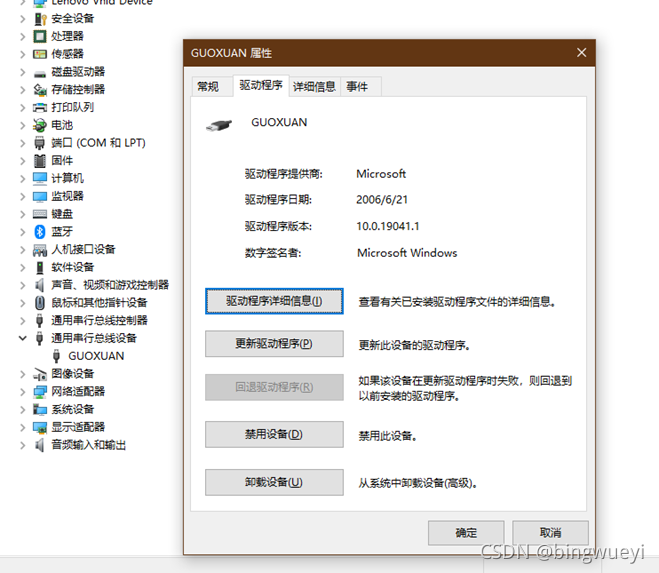
|  |
| --- |
| //---------------ADD GUOXUAN----------  case 0xEE: // OS String  pbuf = (uint8\_t \*) pdev->pClass->GetWinUSBOSDescriptor(&len);  break;  //------------------------------------ |

1. 修改CDC配置描述信息：

|  |
| --- |
| //---------------CHANGE GUOXUAN---------------  /\* USB CDC device Configuration Descriptor \*/  #define WINUSB\_CONFIG\_DESC\_SIZE 32  \_\_ALIGN\_BEGIN uint8\_t USBD\_CDC\_CfgHSDesc[WINUSB\_CONFIG\_DESC\_SIZE] \_\_ALIGN\_END =  {  /\*Configuration Descriptor\*/  0x09, /\* bLength: Configuration Descriptor size \*/  USB\_DESC\_TYPE\_CONFIGURATION, /\* bDescriptorType: Configuration \*/  WINUSB\_CONFIG\_DESC\_SIZE, /\* wTotalLength:no of returned bytes \*/  0x00,  0x01, /\* bNumInterfaces: 1 interface for WINUSB \*/  0x01, /\* bConfigurationValue: Configuration value \*/  USBD\_IDX\_CONFIG\_STR, /\* iConfiguration: Index of string descriptor describing the configuration \*/  0xC0, /\* bmAttributes: self powered \*/  0x32, /\* MaxPower 0 mA \*/  /\*---------------------------------------------------------------------------\*/  /\*Data class interface descriptor\*/  0x09, /\* bLength: Endpoint Descriptor size \*/  USB\_DESC\_TYPE\_INTERFACE, /\* bDescriptorType: \*/  0x00, /\* bInterfaceNumber: Number of Interface, zero based index of this interface \*/  0x00, /\* bAlternateSetting: Alternate setting \*/  0x02, /\* bNumEndpoints: Two endpoints used \*/  0xff, /\* bInterfaceClass: vendor \*/  0x00, /\* bInterfaceSubClass: \*/  0x00, /\* bInterfaceProtocol: \*/  0x00, /\* iInterface: \*/  /\*Endpoint OUT Descriptor\*/  0x07, /\* bLength: Endpoint Descriptor size \*/  USB\_DESC\_TYPE\_ENDPOINT, /\* bDescriptorType: Endpoint \*/  CDC\_OUT\_EP, /\* bEndpointAddress \*/  0x02, /\* bmAttributes: Bulk \*/  LOBYTE(CDC\_DATA\_HS\_MAX\_PACKET\_SIZE), /\* wMaxPacketSize: \*/  HIBYTE(CDC\_DATA\_HS\_MAX\_PACKET\_SIZE),  0x00, /\* bInterval: ignore for Bulk transfer \*/  /\* Endpoint IN Descriptor \*/  0x07, /\* bLength: Endpoint Descriptor size \*/  USB\_DESC\_TYPE\_ENDPOINT, /\* bDescriptorType: Endpoint \*/  CDC\_IN\_EP, /\* bEndpointAddress \*/  0x02, /\* bmAttributes: Bulk \*/  LOBYTE(CDC\_DATA\_HS\_MAX\_PACKET\_SIZE), /\* wMaxPacketSize: \*/  HIBYTE(CDC\_DATA\_HS\_MAX\_PACKET\_SIZE),  0x00 /\* bInterval: ignore for Bulk transfer \*/  };  /\* USB CDC device Configuration Descriptor \*/  \_\_ALIGN\_BEGIN uint8\_t USBD\_CDC\_CfgFSDesc[WINUSB\_CONFIG\_DESC\_SIZE] \_\_ALIGN\_END =  {  /\*Configuration Descriptor\*/  0x09, /\* bLength: Configuration Descriptor size \*/  USB\_DESC\_TYPE\_CONFIGURATION, /\* bDescriptorType: Configuration \*/  WINUSB\_CONFIG\_DESC\_SIZE, /\* wTotalLength:no of returned bytes \*/  0x00,  0x01, /\* bNumInterfaces: 1 interface for Game IO \*/  0x01, /\* bConfigurationValue: Configuration value \*/  USBD\_IDX\_CONFIG\_STR, /\* iConfiguration: Index of string descriptor describing the configuration \*/  0xC0, /\* bmAttributes: self powered \*/  0x32, /\* MaxPower 0 mA \*/  /\*---------------------------------------------------------------------------\*/  /\*Data class interface descriptor\*/  0x09, /\* bLength: Endpoint Descriptor size \*/  USB\_DESC\_TYPE\_INTERFACE, /\* bDescriptorType: \*/  0x00, /\* bInterfaceNumber: Number of Interface, zero based index of this interface \*/  0x00, /\* bAlternateSetting: Alternate setting \*/  0x02, /\* bNumEndpoints: Two endpoints used \*/  0xff, /\* bInterfaceClass: vendor \*/  0x00, /\* bInterfaceSubClass: \*/  0x00, /\* bInterfaceProtocol: \*/  0x00, /\* iInterface: \*/  /\*Endpoint OUT Descriptor\*/  0x07, /\* bLength: Endpoint Descriptor size \*/  USB\_DESC\_TYPE\_ENDPOINT, /\* bDescriptorType: Endpoint \*/  CDC\_OUT\_EP, /\* bEndpointAddress \*/  0x02, /\* bmAttributes: Bulk \*/  LOBYTE(CDC\_DATA\_FS\_MAX\_PACKET\_SIZE), /\* wMaxPacketSize: \*/  HIBYTE(CDC\_DATA\_FS\_MAX\_PACKET\_SIZE),  0x00, /\* bInterval: ignore for Bulk transfer \*/  /\* Endpoint IN Descriptor \*/  0x07, /\* bLength: Endpoint Descriptor size \*/  USB\_DESC\_TYPE\_ENDPOINT, /\* bDescriptorType: Endpoint \*/  CDC\_IN\_EP, /\* bEndpointAddress \*/  0x02, /\* bmAttributes: Bulk \*/  LOBYTE(CDC\_DATA\_FS\_MAX\_PACKET\_SIZE), /\* wMaxPacketSize: \*/  HIBYTE(CDC\_DATA\_FS\_MAX\_PACKET\_SIZE),  0x00 /\* bInterval: ignore for Bulk transfer \*/  }; |

#### **編譯下載及問題處理**

修改完上述4個文件後，編譯工程並下載，下載完後設備最好斷一次電，重新插入USB後，在設備管理器中可以看到已經成功枚舉出了設備名為GUOXUAN的通用串行總線設備：



但凡事都有意外，當下載好後重新插入USB也可能出現以下情況（不限於以下這兩種！！），只要設備名上有嘆號，說明明枚舉不成功，此時除了檢查代碼有無錯誤以外還可以嘗試刪除註冊表中已有的VID PID信息，因為CubeMX生成好工程時測試連接過一次，枚舉為了串口，注測表中已記錄這個設備的信息，但現在設備描述符又指定為了WinUSB設備，所以造成了枚舉失敗。

一張含有 文字 的圖片

自動產生的描述

#### **刪除註冊表中USB設備信息的方法**：

1. 下載PStools工具包 <https://technet.microsoft.com/en-us/sysinternals/bb897553.aspx>，解壓到C:\pstools
2. 以管理員身份運行CMD，輸入命令：cd C:\pstools ，切換到pstools目錄，再輸入命令：$ psexec.exe -i -d -s regedit.exe
3. 進入**HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Enum\USB\**刪除相關設備（以本文為例，VID PID分別為 1234 ABCD）
4. 進入**HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Control\UsbFlags\**刪除相關設備（以本文為例，VID PID分別為 1234 ABCD）
5. OK，此時再插入該Device，即被當做全新設備插入，裝載驅動。

原文參考：

<https://www.codenong.com/41644081/>

<https://blog.csdn.net/lxc1014/article/details/45170107?utm_source=app&app_version=4.17.0&code=app_1562916241&uLinkId=usr1mkqgl919blen>

### 4.3 與上位機通信

在本測試中上位機發送一個數據包到USB設備，數據包的前兩個字節設置為數據長度，設備接收到該數據包後根據數據長度返回一包數據。

#### **4.3.1 修改數據接收和發送函數**

修改**usbd\_cdc\_if.c**文件中數據接收函數，首先添加定義：

|  |
| --- |
| //-----------------ADD GUOXUAN-----------------------  extern volatile int8\_t usb\_rxne;  extern uint8\_t usb\_rx[2048];  //--------------------------------------------------- |

然後修改數據接收處理：

|  |
| --- |
| //-------------ADD GUOXUAN--------------------  memcpy(usb\_rx, Buf, \*Len);  usb\_rxne = SET;  //-------------------------------------------- |

在**main.c**中添加數據發送函數：

首先引用頭文件：

|  |
| --- |
| //---------------ADD GUOXUAN--------------------------  #include "usbd\_cdc\_if.h"  //---------------------------------------------------- |

添加定義：

|  |
| --- |
| //--------------ADD GUOXUAN-----------------------  uint8\_t usb\_rx[2048];  uint8\_t usb\_tx[81920];  volatile int8\_t usb\_rxne = RESET;  extern USBD\_HandleTypeDef hUsbDeviceHS;  //----------------------------------------------- |

添加接收函數：

|  |
| --- |
| //------------ADD GUOXUAN-------------------  void WinUSB\_Receive\_HS()  {  uint16\_t i = 0;  if (usb\_rxne != SET) return;  i = usb\_rx[0] + (usb\_rx[1] << 8);  CDC\_Transmit\_HS(usb\_tx, i);  usb\_rxne = RESET;  }  //------------------------------------------ |

最後在main裡面的while(1)裡調用：

|  |
| --- |
| while (1)  {  /\* USER CODE END WHILE \*/  WinUSB\_Receive\_HS();  /\* USER CODE BEGIN 3 \*/  } |

#### **4.3.2 與上位機通信測試**

上位機採用C#編寫，使用開源庫WinUSBNet（XP使用LibUsbDotNet）進行訪問，上位機發送一個包，設備回覆一個包；上行和下行的包大小自由設置（本測試主要測上行速率，下發的包大小置固定為64，若下發包大小超過512，設備USB接收函數需要做相應處理，因為單次傳輸最大包長為512，大於512將會被分包接收）；上行包的大小可改變，以測試不同包長時的速率；點擊一次send發送1000個包，並統計速率（本次測試沒有對接收和發送的數據作任何處理，實際應用情況下，加上數據處理後通信速率稍低於測試值）。WIN8、WIN7、XP均是在虛擬機運行測試的，通信速率可能會受到一些影響。

WIN10上測試結果如下：

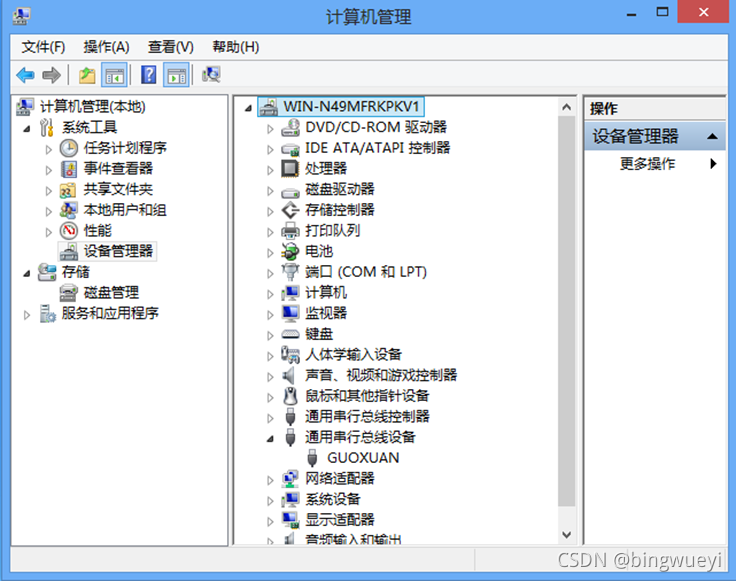
一張含有 文字 的圖片

自動產生的描述

一張含有 文字 的圖片

自動產生的描述

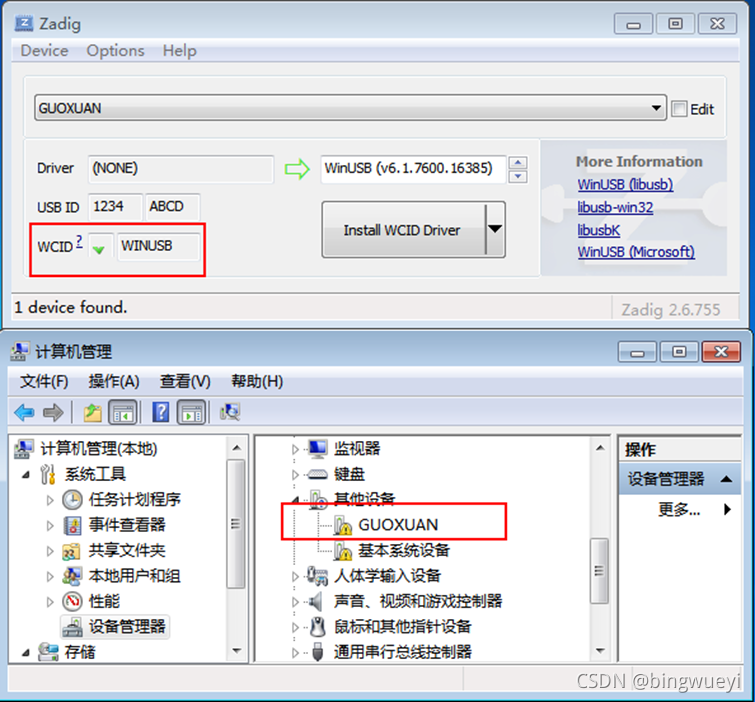
WIN8上測試結果如下：



一張含有 文字 的圖片

自動產生的描述

WIN7和XP上默認不帶WinUSB驅動，設備插到電腦後，設備管理器可以看到設備枚舉失敗，但用zadig軟件可以識別到該設備為WICD設備，可以安裝WinUSB驅動程序，省去了手動寫inf文件，也可以用zadig軟件將驅動程序導出，通過右鍵更新驅動程序的方式安裝。



WIN7上測試結果如下：

一張含有 文字 的圖片

自動產生的描述

一張含有 文字 的圖片

自動產生的描述

XP上測試結果如下(XP使用LibUsbDotNet，默認情況下接收buffer只能到4095)：

一張含有 文字 的圖片

自動產生的描述

一張含有 文字 的圖片

自動產生的描述

## 5 常見問題

1. 沒有USB測試儀無法抓包分析:

用Keil進行抓包，全新的USB枚舉過程數據通信大概有40條左右，最大長度不超過150字節，所以定義一個數組usbLog[50][200]，在USB設備請求通信入口 USBD\_StdDevReq() 函數處記錄上位機下發的信息，在USB設備回覆數據USBD\_CtlSendData() 函數處記錄回覆給上位機的信息，即可抓取到全部通信內容。第3章節USB枚舉過程中信息正是用此方法獲取的。

1. 抓包分析抓不到**0xEE**命令:

調試過程中，可能第一次並未完全修改正確，插入電腦後枚舉失敗，但是電腦已經記錄下了該設備PID VID，當下次枚舉的時候，該PID VID已經存在並且記錄了該設備不是WinUSB設備，所示不會發送0xEE命令，解決辦法是更換PID VID 或者4.2.5所述的方法刪除註冊表中相應的PID VID即可。

1. 完全按上述流程修改後還是無法枚舉成功:

同第2條。

1. 下載正確程序後不枚舉或枚舉失敗:

因為USB HS使用的是外部PHY，重新燒寫單片機程序可能不會讓USB重新枚舉，通常斷電一次即可枚舉成功。

1. 傳輸過程中有丟包現象:

檢查包長設置是否合理，緩衝區buffer是否足夠。