M460 高速 USB 虛擬多路串口

NuMicro® 32 位系列微控制器範例程式介紹

文件資訊

應用簡述	本範例程式介紹如何使用 M460 高速 USB 虛擬多路串口
BSP 版本	M460_Series_BSP_CMSIS_V3.00.002
開發平台	NuMaker-M467HJ V1.0

The information described in this document is the exclusive intellectual property of Nuvoton Technology Corporation and shall not be reproduced without permission from Nuvoton.

Nuvoton is providing this document only for reference purposes of NuMicro microcontroller and microprocessor based system design. Nuvoton assumes no responsibility for errors or omissions.

All data and specifications are subject to change without notice.

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

1. 概述

本範例程式介紹了利用 M460 高速 USB 來虛擬多路串口（簡稱 VCOM），用以多通道 UART 的高速數據採集，並介紹高速和全速 USB 的差異以及如何辨別設備名和硬體的對應關係。

- 高速 USB 和全速 USB 的異同（為何要用 HSUSB 虛擬串口）
- USB 串口設備的種類
- 如何辨別設備名和硬體的對應關係

1.1 高速 USB 和全速 USB 的異同

高速 USB 和全速 USB 都歸屬於 USB 2.0。 USB 2.0 技術規範是由 Compaq、Hewlett Packard、Intel、Lucent、Microsoft、NEC、Philips 共同制定、發佈的，規範把周邊數據傳送速率進行了提高。 按照原定計劃新的 USB 2.0 標準只是準備把這個標準定在 240 Mbps，後來經過努力將它提高到了 480 Mbps。

由於當時制訂的標準有了變化，USB 規範就產生了三種速度選擇：480 Mbps、12 Mbps、1.5 Mbps。 而 2003 年 6 月份，當 USB 2.0 標準開始逐漸深入人心之後，USB 協會重新命名了 USB 的規格和標準。

重新命名 USB 標準，將原先的 USB 1.1 改成了 USB 2.0 Full-Speed（USB 2.0 全速版簡稱 FSUSB），同時將原有的 USB 2.0 改成了 USB 2.0 High-Speed（高速版簡稱 HSUSB），並同時公佈了新的標識。不言而喻，高速版的 USB 2.0 速度當然超過全速版的 USB 2.0。圖 1–1 USB 2.0 協定三個版本速率差異如下：

Current name	Old name	Speed
USB 2.0 High Speed	USB 2.0	480Mbps
USB 2.0 Full Speed	USB 1.1	12Mbps
USB 2.0 Low Speed	USB 1.0	1.5Mbps

圖 1–1 USB 2.0 各版本速率差異

在 USB 規範中，有個最基本的設備，名字叫 Endpoint（端點），Endpoint 是 USB 設備的一個可以唯一尋址的部分，它是在主機和設備之間的通訊流中資訊的來源或接收點。

- **控制端點**

  控制端點(CEP)或端點 0 用於與設備間握手，它由端點 IN 和端點 OUT 組成。USB 規範要求每個 USB 設備都必須支援控制端點，因為它是用於設備枚舉和基本通訊的必要組成部分。

  HSUSB 有獨立的 CEP 端點來進行基礎通信，FSUSB 則需要佔用端點 0 和端點 1 來實現收發。

- **端點數量**

  除了端點 0，其他端點的數量基於其設計要求，不同的控制器有不同的端點數量。

- **端點方向**

  數據收發本質上是雙向的，在本例中，每個端點僅能配置為單接收或者單發送。

- **端點校驗**

  端點使用 CRC 來檢測事務中的錯誤。 這些計算處理由 USB 硬體負責，實際使用過程中無需關注。

USB 體系定義了 Endpoint 四種傳輸類型：

- **控制傳輸**：主要用於在設備連接時，對設備進行枚舉以及其他設備的特定操作。
- **中斷傳輸**：用於對延遲要求嚴格、小量數據的可靠傳輸。
- **批量傳輸**：用於對延遲要求寬鬆，大量數據的可靠傳輸。
- **同步傳輸**：用於對可靠性要求不高的實時數據傳輸。

本例中的串口設備使用了上述類型裡的 3 種傳輸類型：

- **控制傳輸：**用以設置 USB 和串口配置，使用了獨立的雙向 USB 控制端點。
- **中斷傳輸：**每路 VCOM 使用 1 路中斷，佔用 1 個 USB 端點，用以串口數據包的中斷信號。

- **批量傳輸**：每路 VCOM 使用 2 路批量傳輸，佔用 2 個 USB 端點，各用以傳輸串口數據包的接收和發送。

M460 EndPoint	EndPoint number	SRAM Size	DMA	EndPoint Maximum Packet	Control EndPoint
HSUSB	12	4kbytes	✓	1024bytes	independent
FSUSB	25	1.5kbyte	×	64bytes	share

圖 1–2 M460 高速和低速 USB 的差異

圖 1–2 表示 M460 兩種 USB 介面的硬體差異，由此我們看到 HSUSB 的每個端點最大包可以發送更多數據，並且參考圖 1–1 USB 2.0 各版本速率差異，使用 HSUSB 能夠更快速的回應 VCOM 發送的數據。由於 HSUSB 僅有 12 個端點且有獨立的收發控制端點，最多只能虛擬出 4 路 VCOM，而 FSUSB 則最多可以虛擬出 7 路 VCOM。

HSUSB 的 VCOM 端點分配（獨立控制端點為收發雙向端點）：

VCOM 數量 = 12÷（1 個中斷傳輸+1 個接收批量傳輸+1 個發送批量傳輸）

= 12÷3

= 4

FSUSB 的 VCOM 端點分配（控制端點為單向端點）：

VCOM 數量 =（25-2 個控制端點（1 個接收+1 個發送））÷（1 個中斷傳輸+1 個接收批量傳輸+1 個發送批量傳輸）

=（25-2）÷3

= 7（最大整除數為 7）

USB 控制傳輸協定裡，另外還有一個重要的概念：描述符。

USB 描述符包含了設備的各種資訊，通過各自端點向主機發送設備的功能、特性和能力。USB 描述符包括設備描述符、配置描述符、介面描述符、端點描述符和字串描述符等。

- 設備描述符包含了設備的基本資訊，如設備類型、廠商 ID、產品 ID、設備版本號等。
- 字串描述符則提供了一些廠商和產品字元描述資訊。
- 配置描述符描述了設備的配置資訊，包括配置的數量、每個配置的屬性等。

- 介面描述符描述了設備的介面資訊，包括介面的數量、每個介面的類型、協定等。
- 端點描述符描述了設備的端點資訊，包括端點的位址、方向、類型、最大數據包大小等。

USB 主機通過讀取這些描述符來了解設備的功能和特性，並根據這些資訊來載入相應的驅動程式，從而實現與設備的通信和互動。

描述符在 USB 互動中有些限定關係，不同的描述符有不同的功能，圖 1–3 是本例中使用到的一些描述符層次關係。

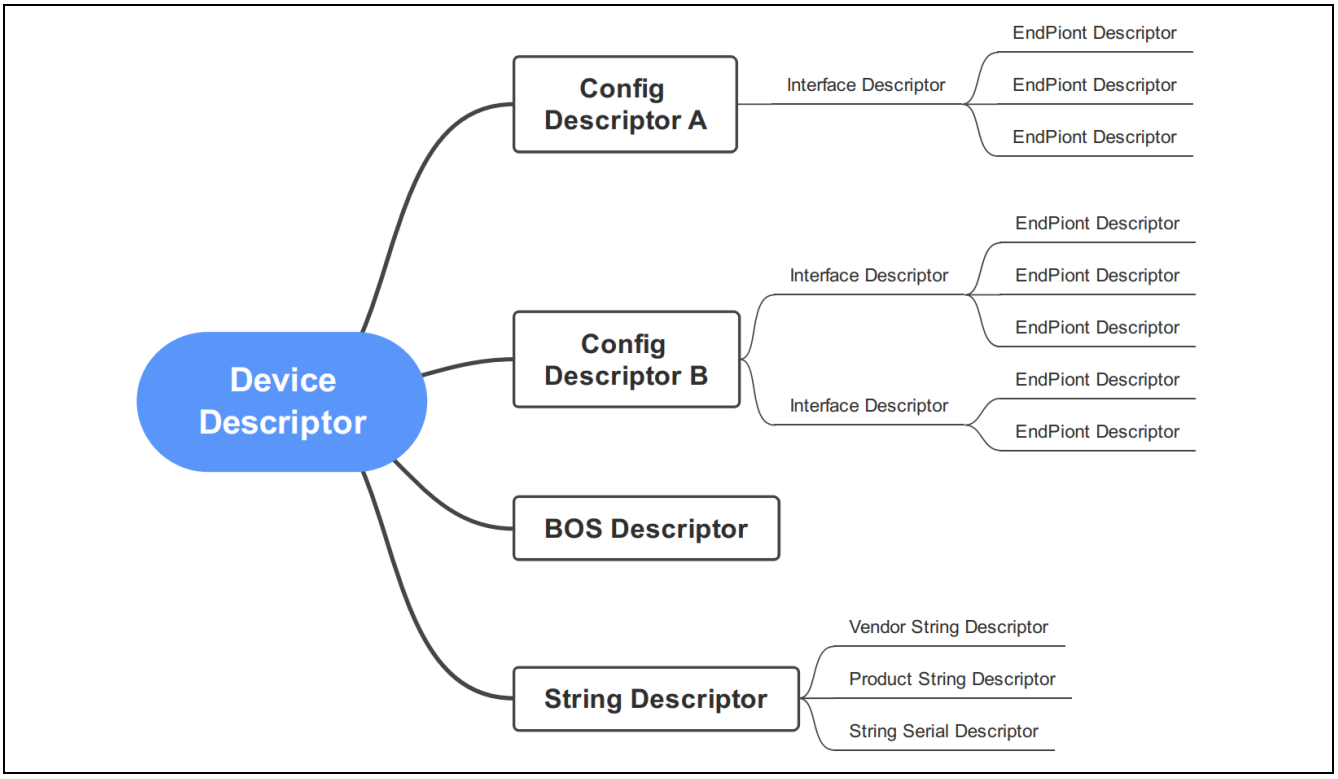

圖 1–3 USB 描述符層次關係

關於描述符 HSUSB 和 FSUSB 也是有些差異的，FSUSB 的描述符種類要比 HSUSB 要少一些，主要是由於介面速率引起的差異。除端點最大數據包大小有差異需要注意外，其他速率的配置描述符內容與主配置描述符內容相同。圖 1–4 是 M460 的描述符種類差異對比。

Device Descriptor	Config Descriptor	String Descriptor	BOS Descriptor	HID Report Descriptor	HID Report Descriptor Size	HID Descriptor Start Index	Qualifier Descriptor	Full Speed Config Descriptor	HSUSB Other Speed Config Descriptor	FSUSB Other Speed Config Descriptor
FSUSB Descriptor										
HSUSB Descriptor										

圖 1–4 HSUSB 和 FSUSB 描述符差異

1.2 USB 串口設備的種類

USB 轉 VCP 串口

- 支援高速率通訊及硬體流控。
- 功能齊全，支援 GPIO 等擴展功能。
- 虛擬標準串口，用於升級原串口外圍設備，或者通過 USB 增加額外串口。
- 計算機連接埠應用程式完全相容，無需修改。
- 驅動只需安裝一次也可以聯網自動安裝。

USB 轉 CDC 串口

- Windows 10 及以上的操作系統版本內置 CDC 串口驅動，老版本則需要安裝。
- 不支援軟體和硬體流控。
- 不建議高波特率時持續傳輸數據，建議在發送間隔內增加延時。
- 在系統睡眠休眠前建議關閉串口，否則有概率在喚醒後串口會停止接收。

USB 轉 HID 串口

- 使用操作系統自帶的 USB HID 設備驅動，USB HID 串口在系統上識別為標準 HID 設備。
- 支援免安裝的作業系統內置 HID 類驅動程式。
- 系統下不生成串口節點，無需手動選擇，同時也實現了一定程度的信息隱藏。
- 數據收發使用 USB 中斷傳輸。
- 需使用系統 HID 原生 API 或者第三方提供的動態庫操作串口。

本例中使用了 CDC 類型的串口，可以在主流 windows 上免驅使用，極大的方便了用戶的實用性，雖然 CDC 串口不支援流控，但是通過獨立的中斷端點也可以大大提高數據的實時回應。

1.3 如何辨別設備名和硬體的對應關係

我們先看一下韌體運行後，默認驅動在電腦上虛擬出的圖 1–5 默認驅動 VCOM 設備名稱：

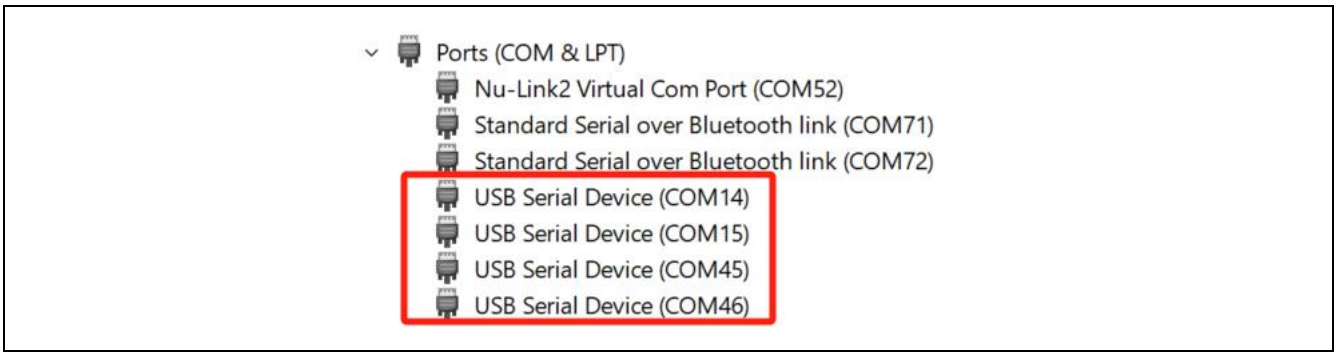

圖 1–5 默認驅動 VCOM 設備名稱

接上設備後，系統會根據已經註冊過設備的串口埠（COM port），分配設備原來的串口號或者新增串口號，但 M467 韌體是按照 USB 設備的介面號（Interface number）與硬體串口號（UART number）進行對應設置。 這種對應方式就導致了串口號（系統分配有隨機性）在實際使用過程中不能很好的找到硬體串口。

韌體中 USB 介面號和串口號的固定對應關係如下：

- Interface0 <==> UART0
- Interface2 <==> UART2
- Interface4 <==> UART3
- Interface6 <==> UART4

電腦上的串口埠有隨機性，並且設備名“USB Serial Device”都是一樣的，這樣無法直觀的選擇到需要操作的硬體 UART 上，只能從設備屬性裡去查看對應 USB 設備的介面號。

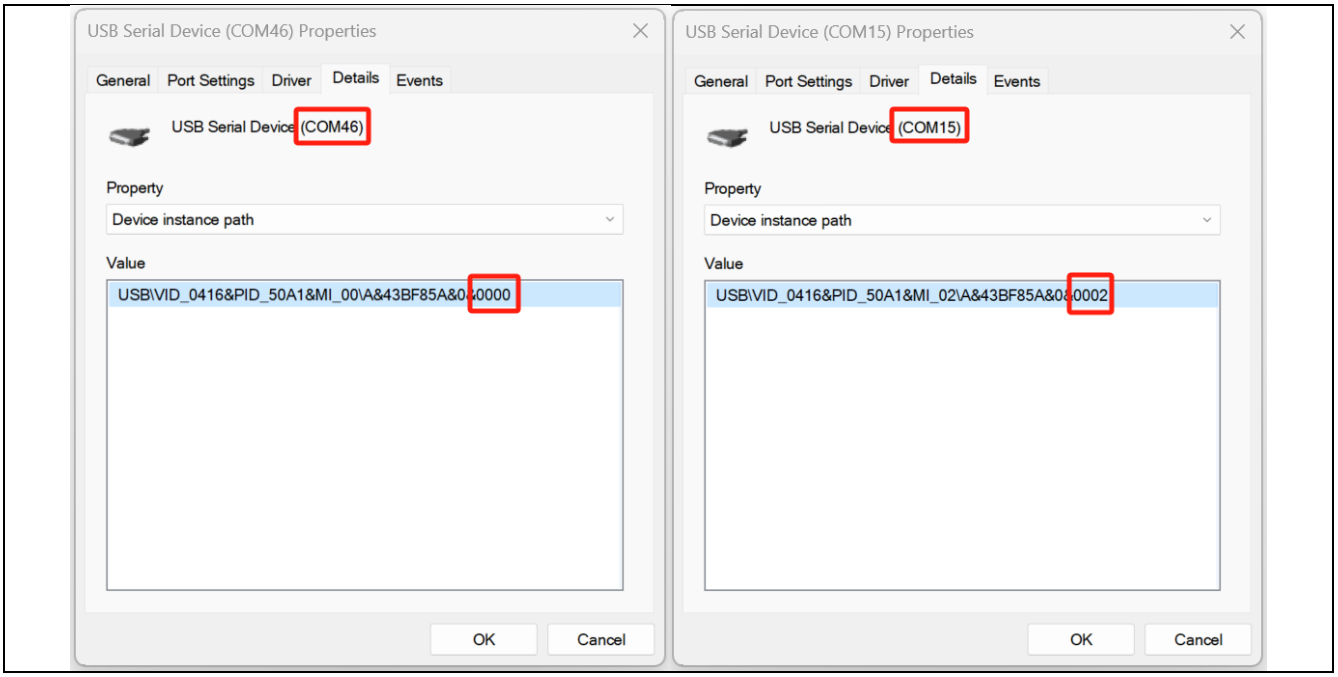

圖 1–6 不同 COM 對應的介面

圖 1–6 中紅框內的數位就是對應的介面號，那麼 COM 號和介面號對應關係如下：

- COM46 <==> Interface0
- COM15 <==> Interface2

針對上述問題我們把 USB 的驅動做了些調整，用不同的設備名稱來（VCOM 號）區分不同的介面號，這樣來方便用戶能夠直觀的選擇到自己需要操作的硬體 UART 上。

以下步驟可以安裝上我們調整好的驅動：

1. 安裝 Nu-Link USB 驅動，下載位址：

   https://www.nuvoton.com.cn/export/resource-files/en-us--Nu-Link_USB_Driver_V1.11.zip

2. 替換原有驅動檔 nuvcom.cat 和 nuvcom.inf，默認安裝路徑：
   C：\Program Files\Nuvoton Tools\Nu-Link_USB_Driver

3. 更新驅動程式：

   選取裝置 -> 右擊更新驅動程式 -> 瀏覽我的電腦以查找驅動 -> 讓我從電腦上的可用驅動程式可用清單中選取 -> 如圖 1–7 選擇對應驅動選擇相應的「VCOMx」驅動 -> 下一步 -> 完成。

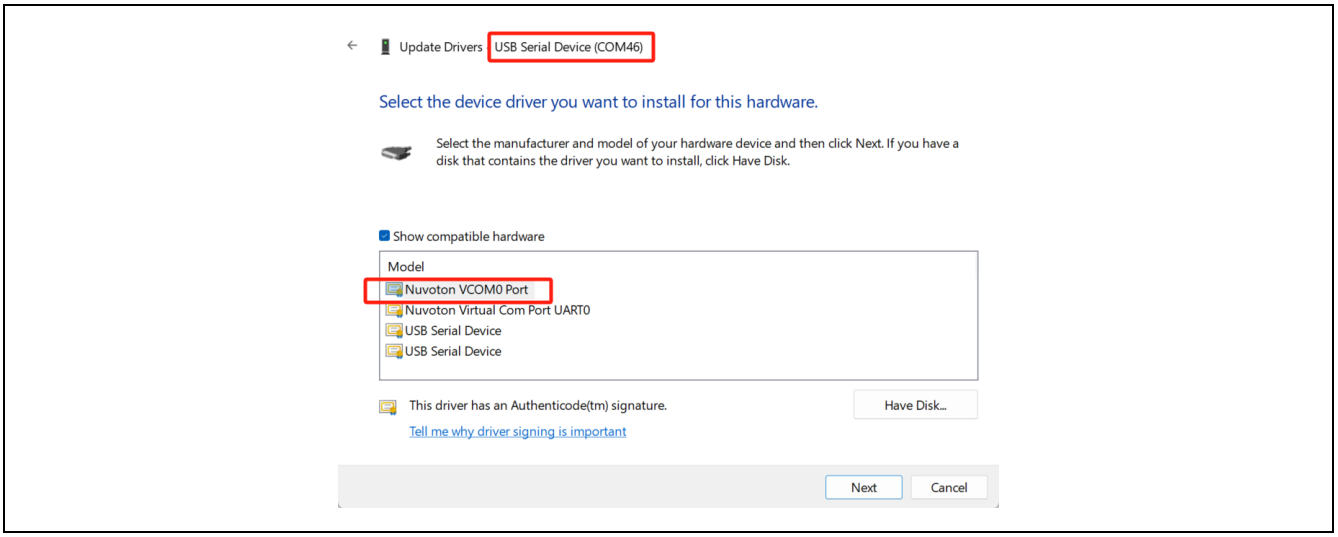

圖 1–7 選擇對應驅動

圖 1–8 修改後的設備名稱是驅動更新後顯示的 4 路 VCOM 的名稱：

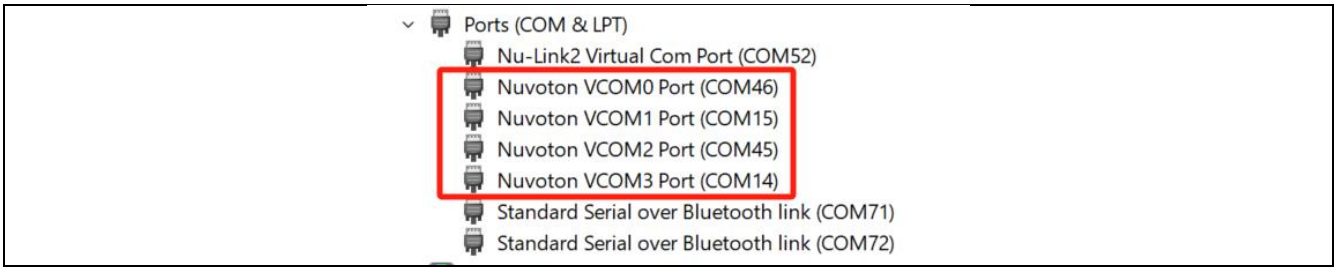

圖 1–8 修改後的設備名稱

從圖 1–8 修改後的設備名稱結合韌體預設的 Interface 和 UART 對應關係可以很容易的從 COM port 找到對應硬體 UART：

- Interface0 <==> VCOM0 <==> UART0 <==> COM46
- Interface2 <==> VCOM1 <==> UART2 <==> COM15
- Interface4 <==> VCOM2 <==> UART3 <==> COM45
- Interface6 <==> VCOM3 <==> UART4 <==> COM14

Interface 和 UART 的對應由韌體設定，請參考圖 1–5 的說明。

2. 程式介紹

USB 相關的程式較多，選取一部分使用者常用可自定義的內容來介紹，包含以下幾點內容：

- USB 參數宏定義
- 設備描述符
- 硬體串口配置
- 數據收發
- 實例測試

2.1 USB 參數宏定義

USB 參數的定義位於 *vcom_serial.h* 檔中。

定義 VCOM 的數量 1~4 ：VCOM_CNT，如果超出範圍則會編譯報錯。

```
#define VCOM_CNT 4

#if ((VCOM_CNT>=5) || (VCOM_CNT<=0))
#error VCOM number error, VCOM_CNT must be 1 ~ 4
#endif......
```

USB PID 和 VID 這個需要能夠讓驅動識別是 VCOM 設備，所以不能更改，否則會無法實現上述的設備名稱對應。

```
//------------------device descriptor define---------------------//
/* Define the vendor id and product id */
#define USBD_VID        0x0416
#define USBD_PID        0x50A1
```

為每個端點分配 USB 的 RAM 的大小，RAM 一共 4 KByte，每個 VCOM 的收發各分配 256 Byte，如果個別應用需要提高最大數據包可在這裡修改，注意同時還要修改對應分配的 RAM 位址。

控制端點 CEP 一般定義為 64 Bytes。

```
//------------------ram buffer define---------------------//
/* Define DMA Maximum Transfer length */
#define USBD_MAX_DMA_LEN    0x1000
#define EP_MAX_DATA_PKT_SIZE   256

/* Define EP maximum packet size */
```

```
/*4096 Bytes Configurable RAM used as endpoint buffer*/
#define CEP_MAX_PKT_SIZE        64
#define CEP_OTHER_MAX_PKT_SIZE  64
```

2.2 設備描述符

每一個 USB 設備都有自己的設備說明，本例中每個設備的描述類型都一樣。不同的是定義了各自的設備的介面號，最大數據包，端點號和端點類型。設備的描述內容在 descriptors.c 檔中。這裡僅顯示 VCOM0 的設備描述。

```
//-----------------VCOM0------------------//
#if (VCOM_CNT >= 1)
    // IAD
    LEN_IAD,        // bLength        : Interface Descriptor size
    DESC_IAD,       // bDescriptorType: IAD
    INTF_NUM_VCOM0, // bFirstInterface
    0x02,           // bInterfaceCount
    0x02,           // bFunctionClass     : CDC
    0x02,           // bFunctionSubClass
    0x01,           // bFunctionProtocol
    0x02,           // iFunction

    /* INTERFACE descriptor */
    LEN_INTERFACE,                          /* bLength              */
    DESC_INTERFACE,                         /* bDescriptorType      */
    INTF_NUM_VCOM0,                         /* bInterfaceNumber     */
    0x00,                                   /* bAlternateSetting    */
    0x01,                                   /* bNumEndpoints        */
    DESC_CDC_COMMUNICATION_INTERFACE_CLASS, /* bInterfaceClass      */
    DESC_CDC_ABSTRACT_CONTROL_MODEL,        /* bInterfaceSubClass   */
    0x01,                                   /* bInterfaceProtocol   */
    0x00,                                   /* iInterface           */

    /* Communication Class Specified INTERFACE descriptor */
    0x05,                 /* Size of the descriptor, in bytes */
    DESC_CDC_CS_INTERFACE,/* CS_INTERFACE descriptor type */
    DESC_CDC_HEADER,      /* Header functional descriptor subtype */
    0x10, 0x01, /* Communication device compliant to the communication spec. ver. 1.10 */

    /* Communication Class Specified INTERFACE descriptor */
    0x05,                       /* Size of the descriptor, in bytes */
    DESC_CDC_CS_INTERFACE,      /* CS_INTERFACE descriptor type */
    DESC_CDC_CALL_MANAGEMENT,   /* Call management functional descriptor */
    0x00,                       /* BIT0: Whether device handle call management itself. */
    /* BIT1: Whether device can send/receive call management information over a Data Class
Interface 0 */
    INTF_NUM_VCOM0 + 1,         /* Interface number of data class interface optionally used
for call management */

    /* Communication Class Specified INTERFACE descriptor */
    0x04,                           /* Size of the descriptor, in bytes */
    DESC_CDC_CS_INTERFACE,          /* CS_INTERFACE descriptor type */
```

```
    DESC_CDC_ABSTRACT_CONTROL_MODEL, /* Abstract control management functional descriptor
subtype */
    0x00,                            /* bmCapabilities       */

    /* Union Functional descriptor */
    0x05,                            /* bLength              */
    DESC_CDC_CS_INTERFACE,             /* bDescriptorType: CS_INTERFACE descriptor type */
    DESC_CDC_UNION,                    /* bDescriptorSubType   */
    INTF_NUM_VCOM0,                    /* bMasterInterface     */
    INTF_NUM_VCOM0 + 1,                /* bSlaveInterface0     */

    /* ENDPOINT descriptor */
    LEN_ENDPOINT,                      /* bLength          */
    DESC_ENDPOINT,                     /* bDescriptorType  */
    (EP_INPUT | INT_IN_EP_NUM_0),      /* bEndpointAddress */
    EP_INT,                            /* bmAttributes     */
    GET_BYTE0(EPC_OTHER_MAX_PKT_SIZE),
    GET_BYTE1(EPC_OTHER_MAX_PKT_SIZE), /* wMaxPacketSize   */
    0x01,                              /* bInterval        */

    /* INTERFACE descriptor */
    LEN_INTERFACE,      /* bLength              */
    DESC_INTERFACE,     /* bDescriptorType      */
    INTF_NUM_VCOM0 + 1, /* bInterfaceNumber     */
    0x00,               /* bAlternateSetting    */
    0x02,               /* bNumEndpoints        */
    0x0A,               /* bInterfaceClass      */
    0x00,               /* bInterfaceSubClass   */
    0x00,               /* bInterfaceProtocol   */
    0x00,               /* iInterface           */

    /* ENDPOINT descriptor */
    LEN_ENDPOINT,                      /* bLength          */
    DESC_ENDPOINT,                     /* bDescriptorType  */
    (EP_INPUT | BULK_IN_EP_NUM_0),     /* bEndpointAddress */
    EP_BULK,                           /* bmAttributes     */
    GET_BYTE0(EPA_OTHER_MAX_PKT_SIZE),
    GET_BYTE1(EPA_OTHER_MAX_PKT_SIZE), /* wMaxPacketSize   */
    0x00,                              /* bInterval        */

    /* ENDPOINT descriptor */
    LEN_ENDPOINT,                      /* bLength          */
    DESC_ENDPOINT,                     /* bDescriptorType  */
    (EP_OUTPUT | BULK_OUT_EP_NUM_0),   /* bEndpointAddress */
    EP_BULK,                           /* bmAttributes     */
    GET_BYTE0(EPB_OTHER_MAX_PKT_SIZE),
    GET_BYTE1(EPB_OTHER_MAX_PKT_SIZE), /* wMaxPacketSize   */
    0x00,                              /* bInterval        */
#endif
```

2.3 硬體串口配置

硬體串口使用一個陣列來存放串口配置參數，預設值 115200bps，1 個停止位，8 位數據，無奇偶校驗。這個陣列定義在 *vcom_serical.c* 檔中，在 PC 機打開串口的時候，USB 通過控制端點接收存放 PC 發送的配置參數，並重新配置串口參數。

```
/*-------------------------------------------------------------------------*/
/* Stop bit     */
/* parity       */
/* data bits    */
STR_VCOM_LINE_CODING gLineCoding[VCOM_CNT] =
#if (VCOM_CNT>=1)
{   {115200, 0, 0, 8},   /* Baud rate : 115200    */
#endif
#if (VCOM_CNT>=2)
    {115200, 0, 0, 8},   /* Baud rate : 115200    */
#endif
#if (VCOM_CNT>=3)
    {115200, 0, 0, 8},   /* Baud rate : 115200    */
#endif
#if (VCOM_CNT>=4)
    {115200, 0, 0, 8},   /* Baud rate : 115200    */
#endif
};
```

*vcom_serical.c*檔中的函式*VCOM_ClassRequest（ ）*用來讀取和設置USB發送過來的串口設置數據。

```
/**
 *  @brief      USB Control Endpoint to Config UART port
 *  @param[in]  None
 *  @return     None
 *  @details    The function is used to:
                  Read UART Config information
                  Write UART Config information
*/
void VCOM_ClassRequest(void)
{   static uint8_t intface;
    if（gUsbCmd.bmRequestType & 0x80）     /* request data transfer direction */
    {
        // Device to host
        switch(gUsbCmd.bRequest)
        {
        case GET_LINE_CODE:
        {
            intface = gUsbCmd.wIndex & 0xff;
            HSUSBD_PrepareCtrlIn((uint8_t *)&gLineCoding[intface>>1], 7);

            HSUSBD_CLR_CEP_INT_FLAG(HSUSBD_CEPINTSTS_INTKIF_Msk);
            HSUSBD_ENABLE_CEP_INT(HSUSBD_CEPINTEN_INTKIEN_Msk);
```

```
            break;
        }
        default:
        {
            /* Setup error, stall the device */
            HSUSBD_SET_CEP_STATE(HSUSBD_CEPCTL_STALLEN_Msk);
            break;
        }
        }
    }
    else
    {
        // Host to device
        switch(gUsbCmd.bRequest)
        {
        case SET_CONTROL_LINE_STATE:
        {
            intface = gUsbCmd.wIndex & 0xff;
            gCtrlSignal[intface>>1] = gUsbCmd.wValue;

            // DATA IN for end of setup
            /* Status stage */
            HSUSBD_CLR_CEP_INT_FLAG(HSUSBD_CEPINTSTS_STSDONEIF_Msk);
            HSUSBD_SET_CEP_STATE(HSUSBD_CEPCTL_NAKCLR);
            HSUSBD_ENABLE_CEP_INT(HSUSBD_CEPINTEN_STSDONEIEN_Msk);
            break;
        }
        case SET_LINE_CODE:
        {
            intface = gUsbCmd.wIndex & 0xff;
            HSUSBD_CtrlOut((uint8_t *)&gLineCoding[intface>>1], 7);

            /* Status stage */
            HSUSBD_CLR_CEP_INT_FLAG(HSUSBD_CEPINTSTS_STSDONEIF_Msk);
            HSUSBD_SET_CEP_STATE(HSUSBD_CEPCTL_NAKCLR);
            HSUSBD_ENABLE_CEP_INT(HSUSBD_CEPINTEN_STSDONEIEN_Msk);

            /* UART setting */
            VCOM_LineCoding(intface>>1);
            break;
        }
        default:
        {
            /* Setup error, stall the device */
            HSUSBD_SET_CEP_STATE(HSUSBD_CEPCTL_STALLEN_Msk);
            break;
        }
        }
    }
}
```

收到USB發送過來的串口配置後，通過*uart_cfg.c*檔中*VCOM_LineCoding（）*來進行硬體串口參數的設置。

```
/**
 *  @brief      USB VCOM set uart config
 *  @param[in]  UART COM Index:
 *              - \ref IDX_VCOM0
 *              - \ref IDX_VCOM1
 *              - \ref IDX_VCOM2
 *              - \ref IDX_VCOM3
 *  @return     None
 *  @details
*/
void VCOM_LineCoding(uint8_t port)
{
    uint32_t u32Reg;
    uint32_t u32Baud_Div;
    volatile struct vcom_var_t *vtp=&vcom_dat[port];

    switch(port)
    {
#if (VCOM_CNT>=1)
    case IDX_VCOM0 :
        NVIC_DisableIRQ(UART0_IRQn);
        break;
#endif
#if (VCOM_CNT>=2)
    case IDX_VCOM1 :
        NVIC_DisableIRQ(UART2_IRQn);
        break;
#endif
#if (VCOM_CNT>=3)
    case IDX_VCOM2 :
        NVIC_DisableIRQ(UART3_IRQn);
        break;
#endif
#if (VCOM_CNT>=4)
    case IDX_VCOM3 :
        NVIC_DisableIRQ(UART4_IRQn);
        break;
#endif
    }

    // Reset software fifo
    vtp->comRbytes = 0;
    vtp->comRhead = 0;
    vtp->comRtail = 0;

    vtp->comTbytes = 0;
    vtp->comThead = 0;
    vtp->comTtail = 0;

    // Reset hardware fifo
    uart_port[port]->FIFO = 0x3;

    // Set baudrate
    u32Baud_Div = UART_BAUD_MODE2_DIVIDER(__HIRC, gLineCoding[port].u32DTERate);

    if(u32Baud_Div > 0xFFFF)
```

```
        uart_port[port]->BAUD = (UART_BAUD_MODE0 | UART_BAUD_MODE0_DIVIDER(__HIRC,
gLineCoding[port].u32DTERate));
    else
        uart_port[port]->BAUD = (UART_BAUD_MODE2 | u32Baud_Div);

    // Set parity
    if(gLineCoding[port].u8ParityType == 0)
        u32Reg = 0;        // none parity
    else if(gLineCoding[port].u8ParityType == 1)
        u32Reg = 0x08;     // odd parity
    else if(gLineCoding[port].u8ParityType == 2)
        u32Reg = 0x18;     // even parity
    else
        u32Reg = 0;

    // bit width
    switch(gLineCoding[port].u8DataBits)
    {
    case 5:
        u32Reg |= 0;
        break;
    case 6:
        u32Reg |= 1;
        break;
    case 7:
        u32Reg |= 2;
        break;
    case 8:
        u32Reg |= 3;
        break;
    default:
        break;
    }

    // stop bit
    if(gLineCoding[port].u8CharFormat > 0)
        u32Reg |= 0x4; // 2 or 1.5 bits

    uart_port[port]->LINE = u32Reg;

    // Re-enable UART interrupt
    switch(port)
    {
#if (VCOM_CNT>=1)
    case IDX_VCOM0 :
        NVIC_EnableIRQ(UART0_IRQn);
        break;
#endif
#if (VCOM_CNT>=2)
    case IDX_VCOM1 :
        NVIC_EnableIRQ(UART2_IRQn);
        break;
#endif
#if (VCOM_CNT>=3)
    case IDX_VCOM2 :
        NVIC_EnableIRQ(UART3_IRQn);
```

```
        break;
#endif
#if (VCOM_CNT>=4)
    case IDX_VCOM3 :
        NVIC_EnableIRQ(UART4_IRQn);
        break;
#endif
    }
}
```

2.4 數據收發

數據的收發是雙工的，每個介面都有對應的收發處理機制。

當 USB 發送緩存中填入數據，並在發送完成後對應端點會產生一個中斷，將發送標誌給清除；當 USB 接收緩存中填入數據後，對應端點會產生一個接收中斷，將接收標誌置位，提示有數據接收完成。

```
//-----------------VCOM0------------------//
void EPA_Handle(void) //Send complete endpoint
{   //data out mcu
    volatile uint32_t IrqSt = HSUSBD->EP[EPA]. EPINTSTS & HSUSBD->EP[EPA]. EPINTEN;
    volatile struct vcom_var_t *vtp=&vcom_dat[IDX_VCOM0];

    vtp->gu32TxSize = 0;
    HSUSBD_ENABLE_EP_INT(EPA, 0);
    HSUSBD_CLR_EP_INT_FLAG(EPA, IrqSt);
}

void EPB_Handle(void) //Receive complete endpoint
{   //data in mcu
    volatile uint32_t IrqSt = HSUSBD->EP[EPB]. EPINTSTS & HSUSBD->EP[EPB]. EPINTEN;
    volatile struct vcom_var_t *vtp=&vcom_dat[IDX_VCOM0];
    int volatile i;

    vtp->gu32RxSize = HSUSBD->EP[EPB]. EPDATCNT & 0xffff;
    for(i = 0; i < vtp->gu32RxSize; i++)
        vtp->gUsbRxBuf[i] = HSUSBD->EP[EPB]. EPDAT_BYTE;

    /* Set a flag to indicate bulk out ready */
    vtp->gi8BulkOutReady = 1;
    HSUSBD_CLR_EP_INT_FLAG(EPB, IrqSt);
}
```

UART 通過中斷接收硬體介面發送過來的數據，存入緩存區。主函式不斷檢測 UART 接收緩存區是否有數據，如果檢測到有數據，則將數據搬移到 USB 發送區，並觸發對應 USB 端點來發送數據。同樣在主函式裡也不斷檢測 USB 接收緩存區是否有數據，如果有數據同樣觸發 UART 的發送來送出 USB 的數據。此功能函式 *VCOM_TransferData（）*在 *uart_cfg.c* 檔中，並且需要放在 *main（）*函式裡循環執行。

```
/**
 *  @brief      UART & USB Receive Buffer Transmit
 *  @param[in]  None
 *  @return     None
 *  @details    Put it in main Loop to receive and transmit uart data
*/
void VCOM_TransferData(void)
{
    int32_t i, i32Len,cnt;
    volatile struct vcom_var_t *vtp;

    for(cnt=0; cnt<VCOM_CNT; cnt++)
    {   vtp=&vcom_dat[cnt];

        /* Check if any data to send to USB & USB is ready to send them out */
        if(vtp->comRbytes && (vtp->gu32TxSize == 0))
        {
            i32Len = vtp->comRbytes;
            if(i32Len > EP_MAX_DATA_PKT_SIZE)
                i32Len = EP_MAX_DATA_PKT_SIZE;

            for(i = 0; i < i32Len; i++)
            {
                vtp->gRxBuf[i] = vtp->comRbuf[vtp->comRhead++];
                if(vtp->comRhead >= RXBUFSIZE)
                    vtp->comRhead = 0;
            }

            switch(cnt)
            {
#if (VCOM_CNT>=1)
            case IDX_VCOM0 :
                NVIC_DisableIRQ(UART0_IRQn);
                vtp->comRbytes -= i32Len;
                NVIC_EnableIRQ(UART0_IRQn);
                break;
#endif
#if (VCOM_CNT>=2)
            case IDX_VCOM1 :
                NVIC_DisableIRQ(UART2_IRQn);
                vtp->comRbytes -= i32Len;
                NVIC_EnableIRQ(UART2_IRQn);
                break;
#endif
#if (VCOM_CNT>=3)
            case IDX_VCOM2 :
                NVIC_DisableIRQ(UART3_IRQn);
                vtp->comRbytes -= i32Len;
                NVIC_EnableIRQ(UART3_IRQn);
                break;
#endif
#if (VCOM_CNT>=4)
            case IDX_VCOM3 :
                NVIC_DisableIRQ(UART4_IRQn);
                vtp->comRbytes -= i32Len;
                NVIC_EnableIRQ(UART4_IRQn);
```

```
                break;
#endif
            }

            vtp->gu32TxSize = i32Len;

            switch(cnt)
            {
#if (VCOM_CNT>=1)
            case IDX_VCOM0 :
                for(i = 0; i < i32Len; i++)
                    HSUSBD->EP[EPA]. EPDAT_BYTE = vtp->gRxBuf[i];
                HSUSBD->EP[EPA]. EPRSPCTL = HSUSBD_EP_RSPCTL_SHORTTXEN;     // packet end
                HSUSBD->EP[EPA]. EPTXCNT = i32Len;
                HSUSBD_ENABLE_EP_INT(EPA, HSUSBD_EPINTEN_INTKIEN_Msk);
                break;
#endif
#if (VCOM_CNT>=2)
            case IDX_VCOM1 :
                for(i = 0; i < i32Len; i++)
                    HSUSBD->EP[EPD]. EPDAT_BYTE = vtp->gRxBuf[i];
                HSUSBD->EP[EPD]. EPRSPCTL = HSUSBD_EP_RSPCTL_SHORTTXEN;    // packet end
                HSUSBD->EP[EPD]. EPTXCNT = i32Len;
                HSUSBD_ENABLE_EP_INT(EPD, HSUSBD_EPINTEN_INTKIEN_Msk);
                break;
#endif
#if (VCOM_CNT>=3)
            case IDX_VCOM2 :
                for(i = 0; i < i32Len; i++)
                    HSUSBD->EP[EPG]. EPDAT_BYTE = vtp->gRxBuf[i];
                HSUSBD->EP[EPG]. EPRSPCTL = HSUSBD_EP_RSPCTL_SHORTTXEN;    // packet end
                HSUSBD->EP[EPG]. EPTXCNT = i32Len;
                HSUSBD_ENABLE_EP_INT(EPG, HSUSBD_EPINTEN_INTKIEN_Msk);
                break;
#endif
#if (VCOM_CNT>=4)
            case IDX_VCOM3 :
                for(i = 0; i < i32Len; i++)
                    HSUSBD->EP[EPJ]. EPDAT_BYTE = vtp->gRxBuf[i];
                HSUSBD->EP[EPJ]. EPRSPCTL = HSUSBD_EP_RSPCTL_SHORTTXEN;    // packet end
                HSUSBD->EP[EPJ]. EPTXCNT = i32Len;
                HSUSBD_ENABLE_EP_INT(EPJ, HSUSBD_EPINTEN_INTKIEN_Msk);
                break;
#endif
            }
        }
        /* Process the Bulk out data when bulk out data is ready. */
        if(vtp->gi8BulkOutReady && (vtp->gu32RxSize <= TXBUFSIZE - vtp->comTbytes))
        {
            for(i = 0; i < vtp->gu32RxSize; i++)
            {
                vtp->comTbuf[vtp->comTtail++] = vtp->gUsbRxBuf[i];
                if(vtp->comTtail >= TXBUFSIZE)
                    vtp->comTtail = 0;
            }
```

```
            switch(cnt)
            {
#if (VCOM_CNT>=1)
            case IDX_VCOM0 :
                NVIC_DisableIRQ(UART0_IRQn);
                vtp->comTbytes += vtp->gu32RxSize;
                NVIC_EnableIRQ(UART0_IRQn);
                break;
#endif
#if (VCOM_CNT>=2)
            case IDX_VCOM1 :
                NVIC_DisableIRQ(UART2_IRQn);
                vtp->comTbytes += vtp->gu32RxSize;
                NVIC_EnableIRQ(UART2_IRQn);
                break;
#endif
#if (VCOM_CNT>=3)
            case IDX_VCOM2 :
                NVIC_DisableIRQ(UART3_IRQn);
                vtp->comTbytes += vtp->gu32RxSize;
                NVIC_EnableIRQ(UART3_IRQn);
                break;
#endif
#if (VCOM_CNT>=4)
            case IDX_VCOM3 :
                NVIC_DisableIRQ(UART4_IRQn);
                vtp->comTbytes += vtp->gu32RxSize;
                NVIC_EnableIRQ(UART4_IRQn);
                break;
#endif
            }

            vtp->gu32RxSize = 0;
            vtp->gi8BulkOutReady = 0; /* Clear bulk out ready flag */
        }

        /* Process the software Tx FIFO */
        if(vtp->comTbytes)
        {
            /* Check if Tx is working */
            if((uart_port[cnt]->INTEN & UART_INTEN_THREIEN_Msk) == 0)
            {
                /* Send one bytes out */
                UART_WRITE(uart_port[cnt], vtp->comTbuf[vtp->comThead++]);
                if(vtp->comThead >= TXBUFSIZE)
                    vtp->comThead = 0;

                vtp->comTbytes--;

                /* Enable Tx Empty Interrupt. (Trigger first one) */
                uart_port[cnt]->INTEN |= UART_INTEN_THREIEN_Msk;
            }
        }
    }
}
}
```

2.5 實例測試

測試 2 組不同串口速率的數據互發，看看運行結果：

1. 先將 UART0 和 UART2 收發腳用導線互聯。
2. 串口波特率 2 Mbps，8 位數據，無奇偶校驗，1 位停止位。
3. VCOM0 和 VCOM1 互發數據。

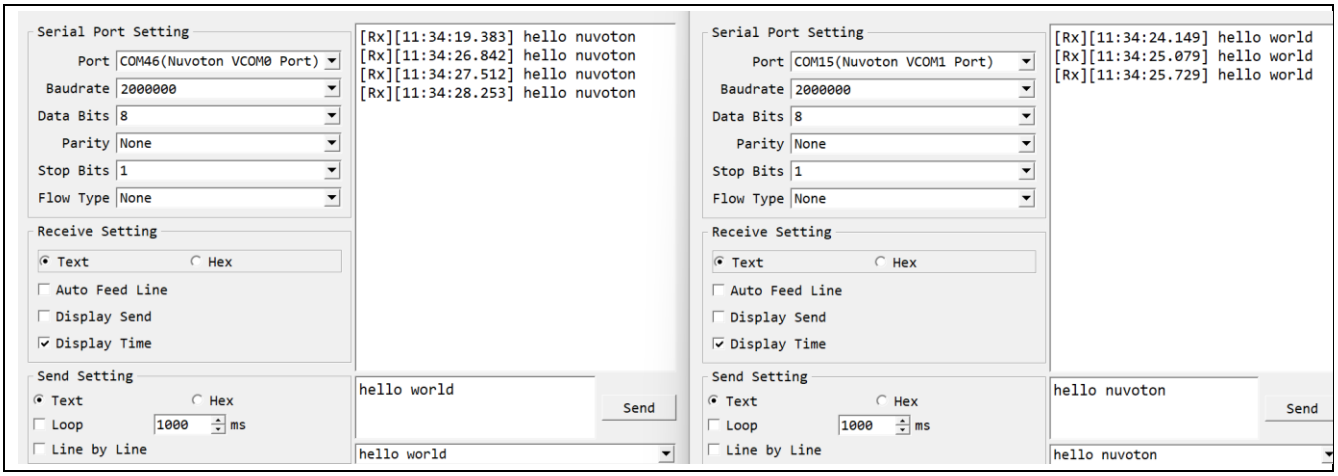

圖 2–1 VCOM0 和 VCOM1 互發數據

1. UART3 和 UART4 收發腳用導線互聯。
2. 串口波特率 9.6kbps，8 位數據，無奇偶校驗，1 位停止位。
3. VCOM2 和 VCOM3 對接互發數據。

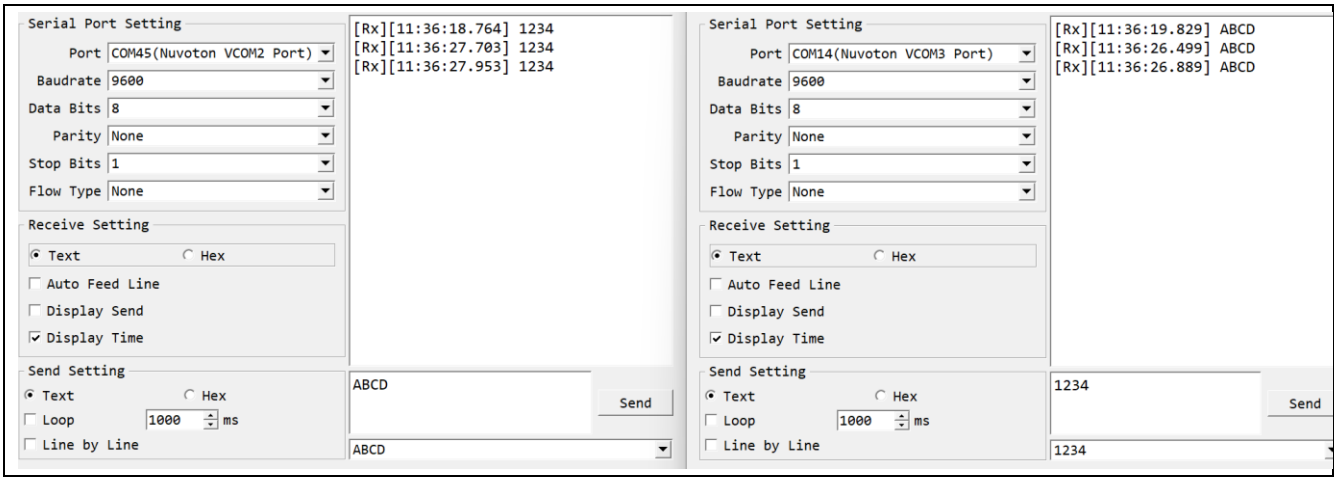

圖 2–2 VCOM2 和 VCOM3 互發數據

3. 軟體與硬體需求

3.1 軟體需求

- BSP 版本
  - M460_Series_BSP_CMSIS_V3.00.002
- IDE 版本
  - Keil uVersion 5.30

3.2 硬體需求

- 電路元件
  - NuMaker-M467HJ V1.0
- 線路示意圖
  - 將 M467HJ 開發板 HSUSB 連接到 PC ，以虛擬 4 路 VCOM。
  - 將開發板上需要測試的 UART 腳位連接，以顯示範例程式的執行結果。

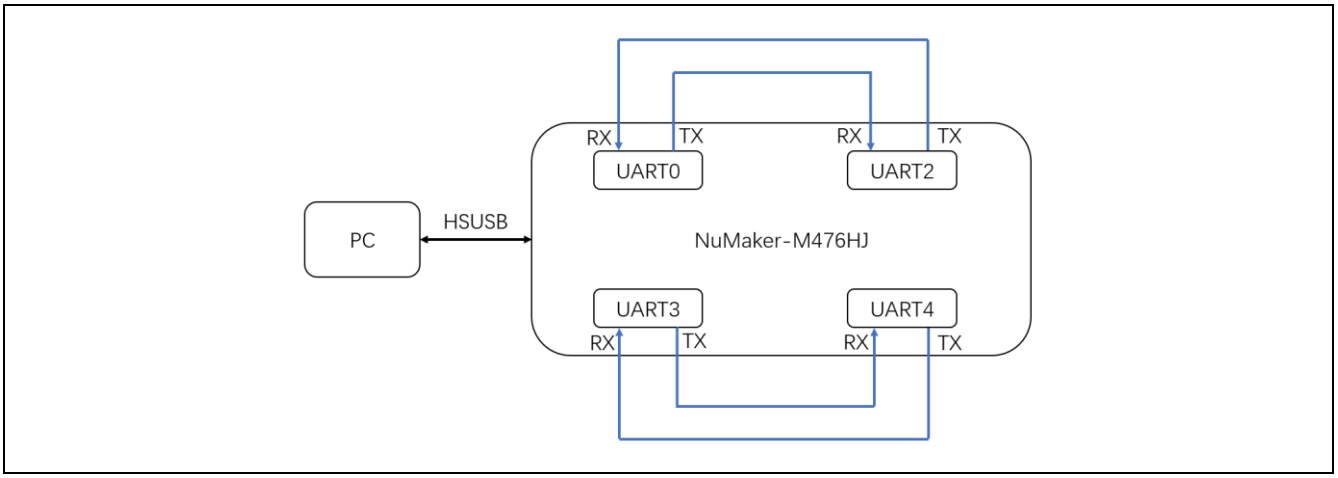

圖 3–1 線路示意圖

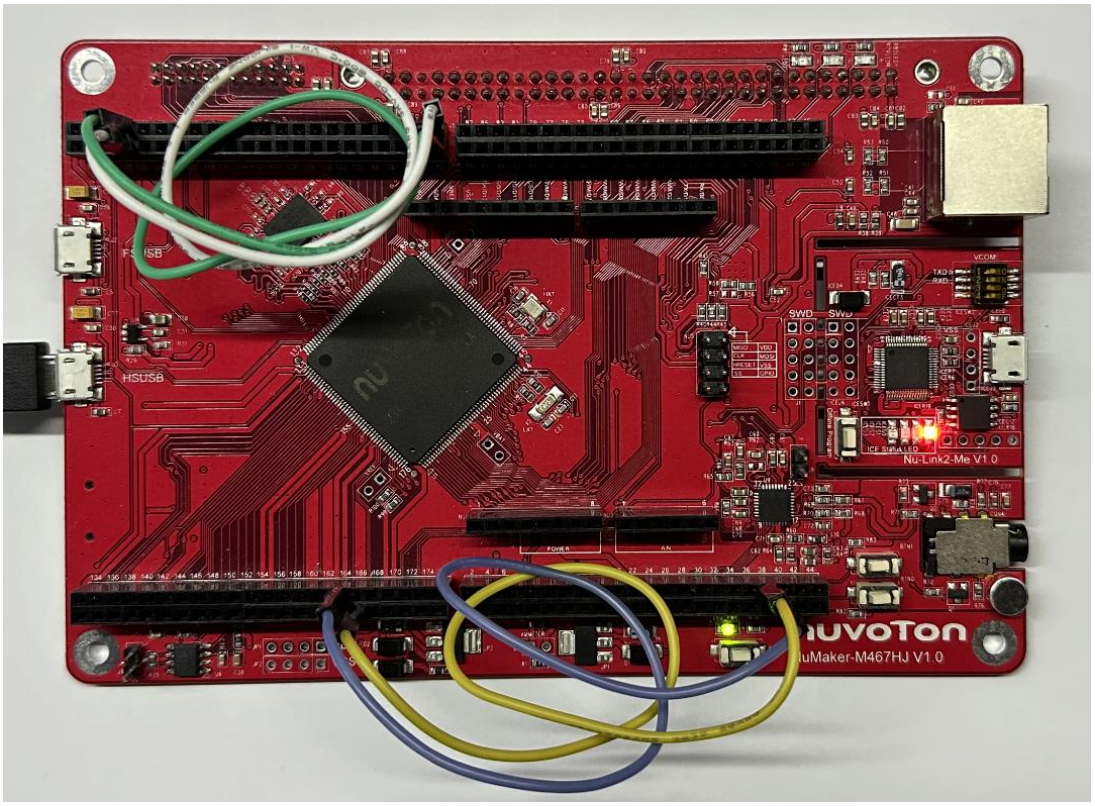

圖 3-2 實物接線

4. 目錄資訊

EC_M460_HSUSBD_VCOM_MultiPort_V1.00	
Library	Sample code header and source files
CMSIS	Cortex® Microcontroller Software Interface Standard (CMSIS) by Arm® Corp.
Device	CMSIS compliant device header file
StdDriver	All peripheral driver header and source files
SampleCode	
ExampleCode	
Project	Source file of example code

圖 3–2 目錄資訊

5. 範例程式執行

1. 根據目錄資訊章節進入 ExampleCode 路徑中的 KEIL 資料夾，雙擊

   HSUSBD_VCOM_MultiPort.uvprojx

2. 進入編譯模式介面

   - 編譯
   - 下載程式至記憶體
   - 進入 / 離開除錯模式

3. 進入除錯模式介面

   - 執行程式

6. 修訂紀錄

Date	Revision	Description
2024.04.28	1.00	初始發布

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.

Please note that all data and specifications are subject to change without notice.
All the trademarks of products and companies mentioned in this datasheet belong to their respective owners.