M460 高速 USB 虚拟多路串口

NuMicro® 32 位系列微控制器范例代码介绍

文件信息

应用简述	本范例代码介绍如何使用 M460 高速 USB 虚拟多路串口
BSP 版本	M460_Series_BSP_CMSIS_V3.00.002
开发平台	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

本范例代码介绍了利用 M460 高速 USB 来虚拟多路串口设备（简称 VCOM），用以多通道 UART 的高速数据采集，并介绍高速和全速 USB 的差异以及如何辨别设备名和硬件的对应关系。

- 高速 USB 和全速 USB 的异同（为何要用 HSUSB 虚拟串口）
- USB 串口设备的种类
- 如何辨别设备名和硬件的对应关系

1.1 高速 USB 和全速 USB 的异同

高速 USB 和全速 USB 都归属于 USB 2.0。USB 2.0 技术规范是由 Compaq、Hewlett Packard、Intel、Lucent、Microsoft、NEC、Philips 共同制定、发布的，规范把外设数据传输速度进行了提高。按照原定计划新的 USB 2.0 标准只是准备把这个标准定在 240 Mbps，后来经过努力将它提高到了 480 Mbps。

由于当时制订的标准有了变化，USB 规范就产生了三种速度选择：480 Mbps、12 Mbps、1.5 Mbps。 而 2003 年 6 月份，当 USB 2.0 标准开始逐渐深入人心之后，USB 协会重新命名了 USB 的规格和标准。

重新命名了 USB 标准将原先的 USB 1.1 改成了 USB 2.0 Full Speed（USB 2.0 全速版简称 FSUSB），同时将原有的 USB 2.0 改成了 USB 2.0 High-Speed（高速版简称 HSUSB），并同时公布了新的标识。不言而喻，高速版的 USB 2.0 速度当然超过全速版的 USB 2.0。图 1–1 USB 2.0 协议三个版本速率差异如下：

现用名	原名	速率
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

图 1–1 USB 2.0 各版本速率差异

在 USB 规范中，有个最基本的设备，名字叫 Endpoint(端点)，Endpoint 是 USB 设备的一个可以唯一寻址的部分，它是在主机和设备之间的通信流中信息的来源或接收点。

- **控制端点**

  控制端点(CEP)或端点 0 用于与设备间握手，它由端点 IN 和端点 OUT 组成。USB 规范要求每个 USB 设备都必须支持控制端点，因为它是用于设备枚举和基本通信的必要组成部分。

  HSUSB 有独立的 CEP 端点来进行基础通信，FSUSB 则需要占用一般通讯端点来实现数据收发。

- **端点数量**

  除了端点 0，其他端点的数量基于其设计要求，不同的控制器有不同的端点数量。

- **端点方向**

  数据收发本质上是双向的，在本例中，每个端点仅能配置为单接收或者单发送。

- **端点校验。**

  端点使用 CRC 来检测事务中的错误。这些计算的处理由 USB 硬件负责，实际使用过程中无需关注。

USB 体系定义了 Endpoint 四种传输类型：

- **控制传输**：主要用于在设备连接时对设备进行枚举以及其他设备的特定操作。
- **中断传输**：用于对延迟要求严格、小量数据的可靠传输。
- **批量传输**：用于对延迟要求宽松，大量数据的可靠传输。
- **同步传输**：用于对可靠性要求不高的实时数据传输。

本例中的串口设备使用了上述类型里的 3 种传输类型：

- **控制传输：**用以设置 USB 和串口配置，使用了独立的双向 USB 控制端点。
- **中断传输：**每路 VCOM 使用 1 路中断，占用 1 个 USB 端点，用以串口数据包的中断信号。

- **批量传输**：每路 VCOM 使用 2 路批量传输，占用 2 个 USB 端点，各用以传输串口数据包的接收和发送。

M460 端点	端点数量	SRAM 大小	DMA	端点最大数据包	控制端点
高速USB	12	4kbytes	✓	1024bytes	独立
全速USB	25	1.5kbyte	×	64bytes	共享

图 1–2 M460 高速和低速 USB 的差异

图 1–2 表示 M460 两种 USB 接口的硬件差异，由此我们看到 HSUSB 的每个端点最大包可以发送更多数据，并且参考图 1–1 USB 2.0 各版本速率差异，使用 HSUSB 能够更快速的响应 VCOM 发送的数据。由于 HSUSB 仅有 12 个端点且有独立的收发控制端点，最多只能虚拟出 4 路 VCOM，而 FSUSB 则最多可以虚拟出 7 路 VCOM。

HSUSB 的 VCOM 端点分配（独立控制端点为收发双向端点）：

VCOM 数量 = 12÷(1 个中断传输+1 个接收批量传输+1 个发送批量传输)

= 12÷3

= 4

FSUSB 的 VCOM 端点分配（控制端点为单向端点）：

VCOM 数量 =（25-2 个控制端点(1 个接收+1 个发送)）÷(1 个中断传输+1 个接收批量传输+1 个发送批量传输)

=（25-2）÷3

= 7（最大整除数为 7）

USB 控制传输协议里另外还有一个重要的概念：描述符。

USB 描述符包含了设备的各种信息，通过各自端点向主机发送设备的功能、特性和能力。USB 描述符包括设备描述符、配置描述符、接口描述符、端点描述符和字符串描述符等。

- 设备描述符包含了设备的基本信息，如设备类型、厂商 ID、产品 ID、设备版本号等。
- 字符串描述符则提供了一些厂商和产品字符描述信息。
- 配置描述符描述了设备的配置信息，包括配置的数量、每个配置的属性等。
- 接口描述符描述了设备的接口信息，包括接口的数量、每个接口的类型、协议等。

- 端点描述符描述了设备的端点信息，包括端点的地址、方向、类型、最大数据包大小等。

USB 主机通过读取这些描述符来了解设备的功能和特性，并根据这些信息来加载相应的驱动程序，从而实现与设备的通信和交互。

描述符在 USB 交互中有些限定关系，不同的描述有不同的功能，图 1–3 是本例中使用到的一些描述符层次关系。

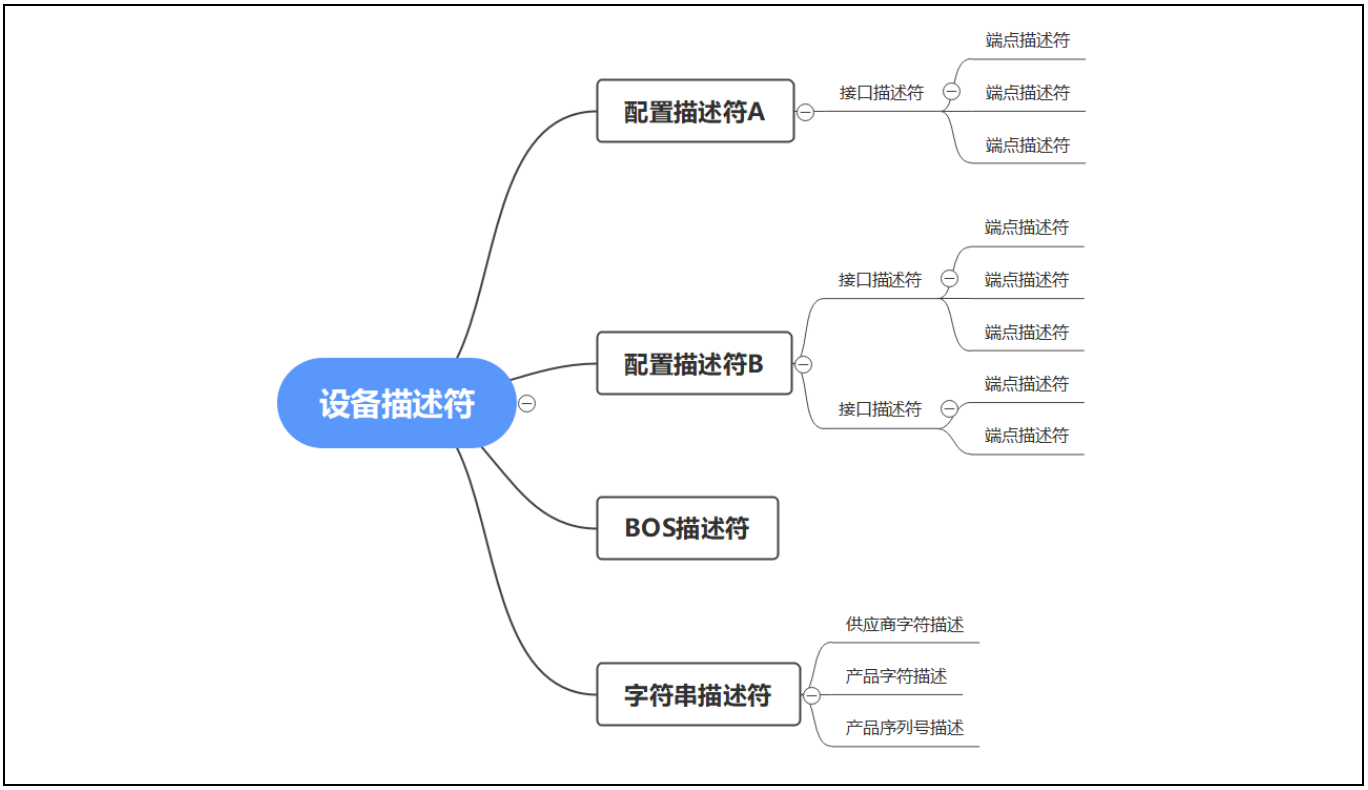

图 1–3 USB 描述符层次关系

关于描述符 HSUSB 和 FSUSB 也是有些差异的，FSUSB 的描述符种类要比 HSUSB 要少一些，主要是由于接口速率引起的差异。除了端点最大数据包大小有差异需要注意外，其他速率的配置描述符内同和主配置描述符内容相同。图 1–4 是 M460 的描述符种类差异对比。

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										

图 1–4 HSUSB 和 FSUSB 描述符差异

1.2 USB 串口设备的种类

USB 转 VCP 串口

- 支持高速率通讯及硬件流控。
- 功能齐全，支持 GPIO 等扩展功能。
- 仿真标准串口，用于升级原串口外围设备，或者通过 USB 增加额外串口。
- 计算机端串口应用程序完全兼容，无需修改。
- 驱动只需安装一次也可以联网自动安装。

USB 转 CDC 串口

- Windows 10 及以上的操作系统版本内置 CDC 串口驱动，以下的系统版本则需要安装。
- 不支持软件和硬件流控。
- 使用串口高波特率时不建议持续传输数据，建议在发送间隔内增加延时。
- 在系统睡眠休眠前建议关闭串口，否则有概率在唤醒后串口会停止接收。

USB 转 HID 串口

- 使用操作系统自带的 USB HID 设备驱动，USB HID 串口在系统上识别为标准 HID 设备。
- 支持免安装的操作系统内置 HID 类驱动程序。
- 系统下不生成串口节点，无需手动选择，同时也实现了一定程度的信息隐藏。
- 数据收发使用 USB 中断传输。
- 需使用系统 HID 原生 API 或者第三方提供的动态库操作串口。

本例中使用了 CDC 类型的串口，可以在主流 windows 系统免驱使用，极大的方便了用户的实用性，虽然 CDC 串口不支持流控，但是通过独立的中断端点也可以大大提高数据的实时响应。

1.3 如何辨别设备名和硬件的对应关系

我们先看一下固件运行后默认驱动在电脑上虚拟出的图 1–5 默认驱动 VCOM 设备名称：

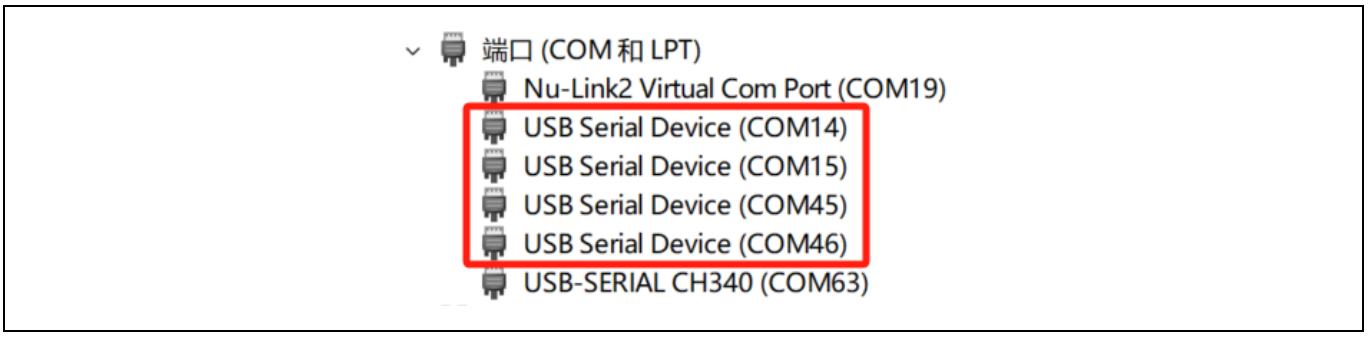

图 1–5 默认驱动 VCOM 设备名称

接上设备后系统会根据已经注册过设备的串口号（COM 号），分配设备原来的串口号或者新增串口号，但 M467 固件是按照 USB 设备的接口号（Interface 号）与物理串口（UART 号）进行对应设置。这样种对应方式就导致了串口号（系统分配有随机性）在实际使用过程中不能很方便的找到硬件串口。

固件中 USB 接口号和 UART 号的固定对应关系如下：

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

电脑上的串口号有随机性，并且设备名“USB Serial Device”都是一样的，这样无法直观的选择到需要操作的物理 UART 上，只能从设备属性里去查看对应 USB 设备的接口号。

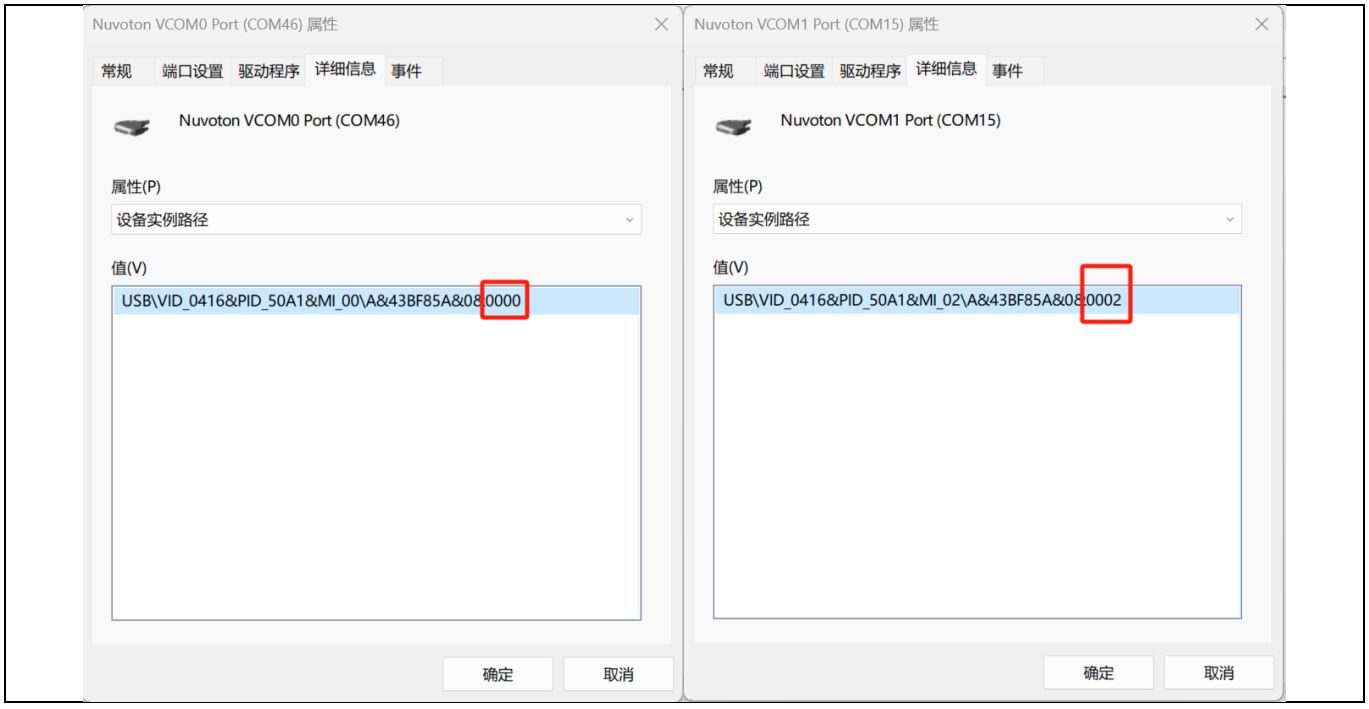

图 1–6 不同 COM 对应的接口

图 1–6 中红框内的数字就是对应的 Interface 号，那么 COM 号和 Interface 号对应关系如下：

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

针对上述问题我们把 USB 的驱动做了些调整，用不同的设备名称来（VCOM 号）区分不同的 Interface 号，这样来方便用户能够直观的选择到自己需要操作的物理 UART 口上。

以下步骤可以安装上我们调整好的驱动：

1. 安装 Nu-Link USB 驱动，下载地址：

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

2. 替换原有驱动文件 nuvcom.cat 和 nuvcom.inf，默认安装路径：

   C:\Program Files\Nuvoton Tools\Nu-Link_USB_Driver

3. 更新驱动程序：

   选中设备->右击更新驱动程序->浏览我的电脑以查找驱动->让我从计算机上的可用驱动程序可用列表中选取->如图 1–7 选择对应驱动选择相应的“VCOMx”驱动->下一步->完成。

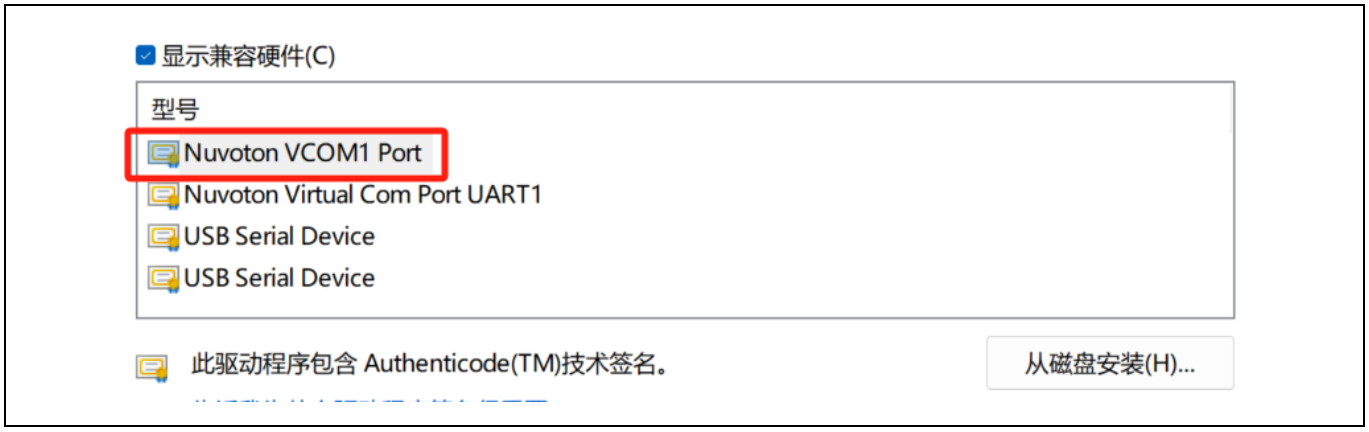

图 1–7 选择对应驱动

图 1–8 修改后的设备名称是驱动更新后显示的 4 路 VCOM 的名称：

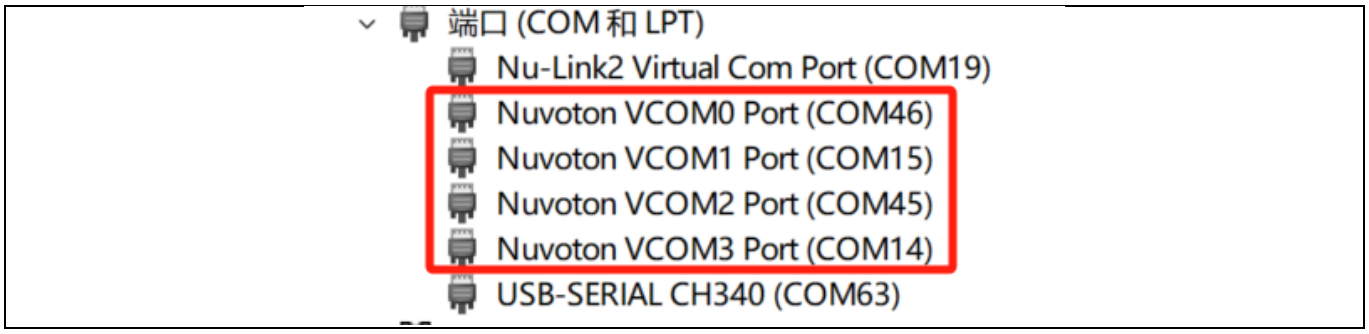

图 1–8 修改后的设备名称

从图 1–8 修改后的设备名称结合固件默认的 Interface 和 UART 对应关系可以很容易的从 COM 口找到物理 UART 口：

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

Interface 和 UART 的对应由固件设定，请参考图 1–5 的说明。

2. 代码介绍

USB 相关的代码较多，选取一部分用户常用且可自定义的内容来介绍，包含以下几点内容：

- USB 参数宏定义
- 设备描述符
- 物串口配置
- 数据收发
- 实例测试

2.1 USB 参数宏定义

USB 参数的定义位于 *vcom_serial.h* 文件中。

定义 VCOM 的数量 1~4 ：VCOM_CNT，如果超出范围则会编译报错。

```
#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 这个需要能够让驱动识别是 VCOM 设备，所以不能更改，否则会无法实现上述的设备名称对应。

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

为每个端点分配 USB 的 RAM 的大小，RAM 一共 4 Kbytes，每个 VCOM 的收发各分配 256 Bytes，如果个别应用需要提高最大数据包可在这里修改，注意同时还要修改对应分配的 RAM 地址。

控制端点 CEP 一般定义为 64 个字节。

```
//------------------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 设备描述符

每一个 USB 设备都有自己的设备说明，本例中每个设备的描述类型都一样。不同的是定义了各自的设备的接口号，最大数据包，端点号和端点类型。设备的描述内容在 *descriptors.c* 文件中。这里仅显示 VCOM0 的设备描述。

```
//-----------------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 物理串口配置

物理串口使用一个数组来存放串口配置参数，默认值 115200bsp，1 个停止位，8 位数据，无奇偶校验。这个数组定义在 *vcom_serical.c* 文件中，在 PC 机器打开串口的时候，USB 通过控制端点接收存放 PC 发送的配置参数，并重新配置串口。

```
/*-------------------------------------------------------------------------*/
/* 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 文件中的函数 *VCOM_ClassRequest()*用来读取和设置 USB 发送过来的串口设置数据。

```
/**
 *  @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 发送过来的串口配置后，通过 *uart_cfg.c* 文件中 *VCOM_LineCoding()*来进行硬件串口参数的设置。

```
/**
 *  @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 数据收发

数据的收发是双工的，每个接口都有对应的收发处理机制。

当 USB 发送缓存中填入数据，并在发送完成后对应端点会产生一个中断，将发送标志给清除；当 USB 接收缓存中有数据后，对应端点会产生一个接收中断，将接收标志置位，提示有数据接收完成。

```
//-----------------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 通过中断接收物理接口发送过来的数据，存入缓存区。主函数不断检测 UART 接收缓存区是否有数据，如果检测到有数据，则将数据搬移到 USB 发送区，并触发对应 USB 端点来发送数据。同样在主函数里也不断检测 USB 接收缓存区是否有数据，如果有数据同样触发 UART 的发送来送出 USB 的数据。此功能函数 VCOM_TransferData()在 uart_cfg.c 文件中，并且需要放在 main()函数里循环执行。

```
/**
 *  @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 实例测试

测试 2 组不同串口速率的数据互发，看看运行结果：

1. 先将 UART0 和 UART2 收发脚用导线互联。
2. 串口波特率 2 Mbps，8 位数据，无奇偶校验，1 位停止位。
3. VCOM0 和 VCOM1 互发数据。

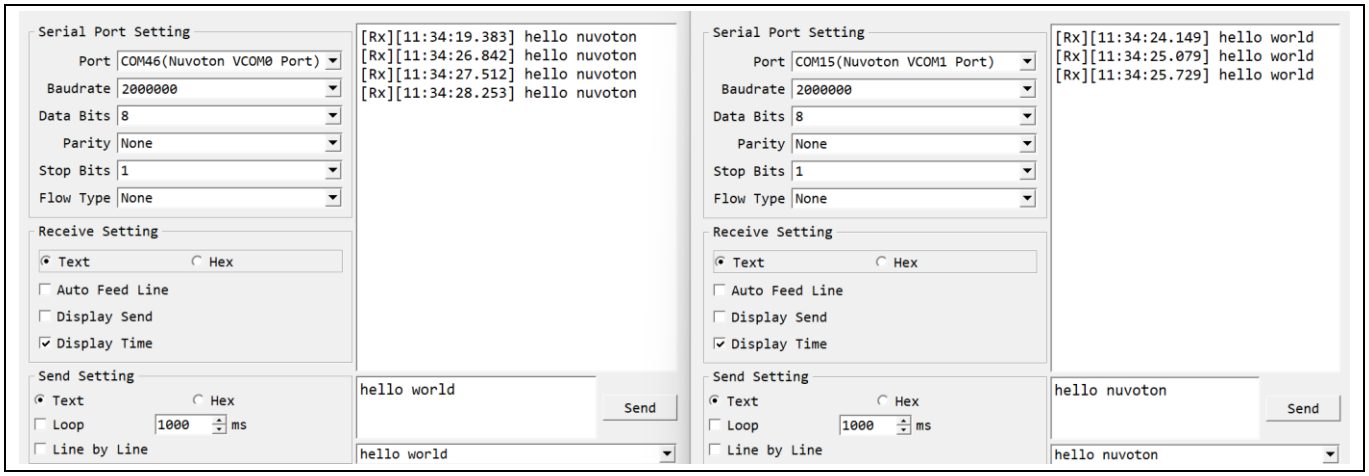

图 2–1 VCOM0 和 VCOM1 互发数据

1. UART3 和 UART4 收发脚用导线互联。
2. 串口波特率 9.6kbps，8 位数据，无奇偶校验，1 位停止位。
3. VCOM2 和 VCOM3 对接互发数据。

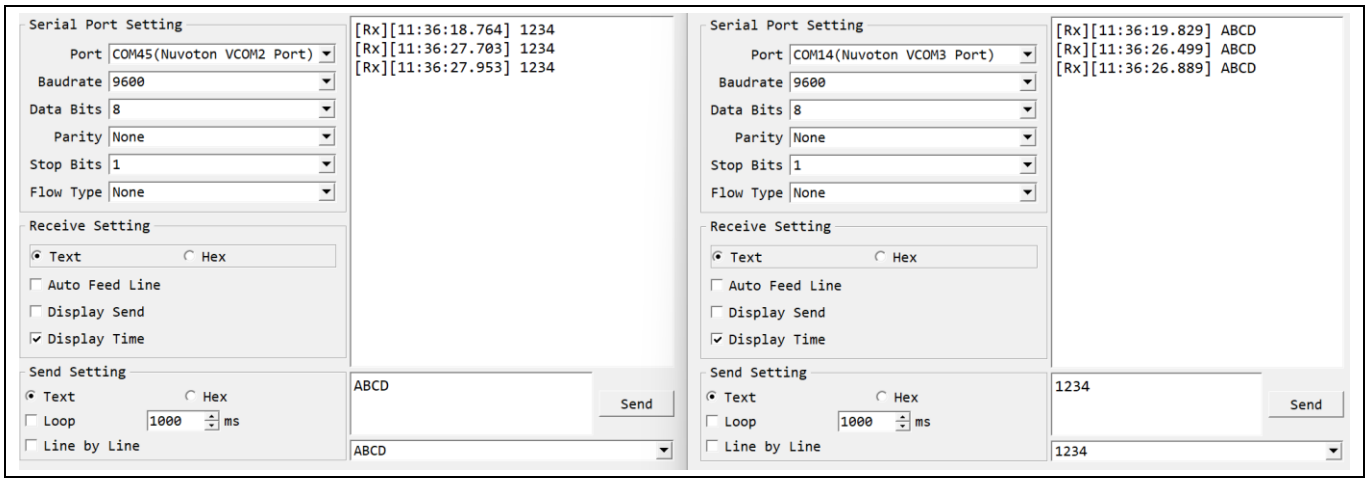

图 2–2 VCOM2 和 VCOM3 互发数据

3. 软件与硬件需求

3.1 软件需求

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

3.2 硬件需求

- 电路组件
  - NuMaker-M467HJ_V1.0
- 线路示意图
  - 将 M467HJ 开发板 HSUSB 连接到 PC，以虚拟 4 路 VCOM。
  - 将开发板上需要测试的 UART 口互联，用以测试例程的执行结果。

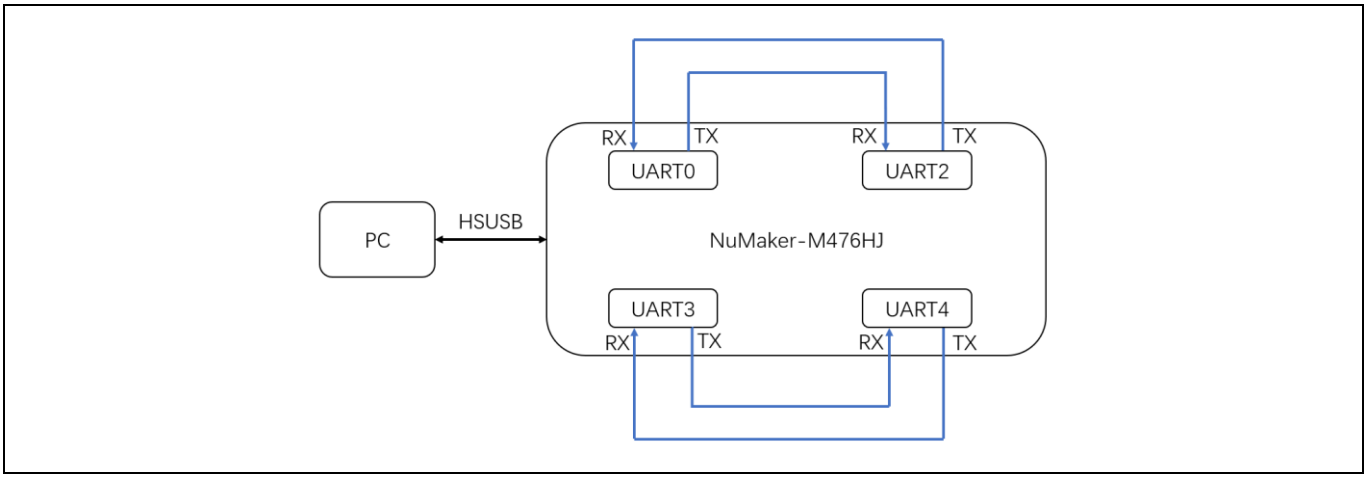

图 3–1 线路示意图

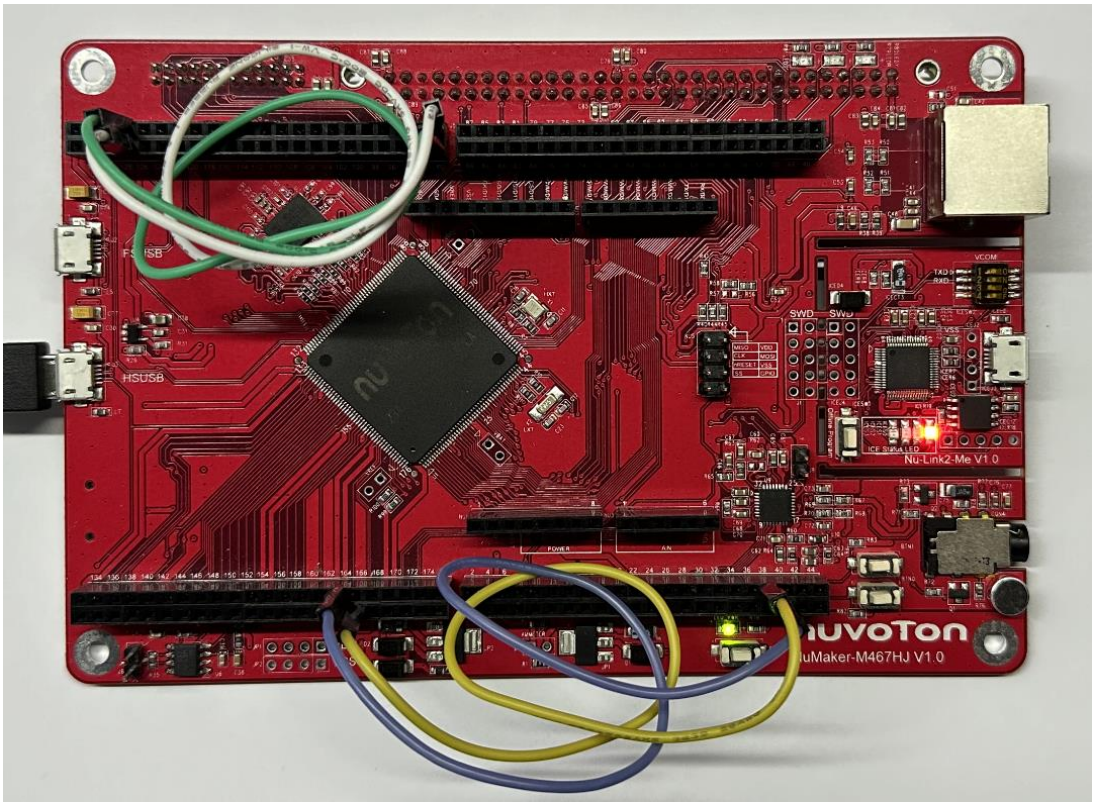

图 3-2 实物接线

4. 目录信息

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

图 4–1 目录信息

5. 范例程序执行

4. 根据目录信息章节进入 ExampleCode 路径中的 KEIL 文件夹，双击

   HSUSBD_VCOM_MultiPort.uvprojx

5. 进入编译模式界面

   - 编译
   - 下载代码至内存
   - 进入 / 离开仿真模式

6. 进入仿真模式界面

   - 执行代码

6. 修订纪录

Date	Revision	Description
2024.04.28	1.00	初始发布

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.