AN12458

LPC5500 和 LPC54018 上的 USB 转串口

版本 0 — 2019 年 6 月

应用笔记

内容

1 背景

串行异步通信接口（UART）往往作为标配外设出现在 MCU 和嵌入式应用中，随着 PC 机的发展，串口逐渐从 PC 机上消失了，对嵌入式工程师及开发人员来说，就会遇到无法直接通过 UART 接口调试嵌入式系统的问题，这时候可以借助一个 USB 接口来实现虚拟串口的功能，从而可以通过 PC 机上的虚拟串口和嵌入式系统进行通信。在一些嵌入式应用中，如果 MCU 上自带的 UART 数量不能满足系统的需求，此时也可以借助 USB 接口来模拟串口。使用 USB 协议规定的 CDC 类中的抽象控制模型子类中的通用 AT 命令协议可以实现 USB 转串口的功能。

本篇应用笔记将介绍如何在 LPC54018 和 LPC5500 上实现一个 USB 转多个虚拟串口的功能。一个 USB 设备可以支持一个或者多个虚拟串口，虚拟串口的数量主要依赖于设备所支持的物理端点数量，LPC54018 和 LPC5500 的全速（FS）USB 设备控制器支持 10 个物理端点，最大只能支持两个虚拟串口，高速（HS）USB 设备控制器支持 12 个物理端点, 最大也只能支持两个虚拟串口。为了支持更多的串口，在 SDK 代码的基础上，本篇应用笔记同时使用了全速和高速两个 USB 设备控制器来实现支持四个虚拟串口的功能，开发工具为 MCUXpresso IDE。

2 描述符的配置

一个 USB 转多个串口的功能需要借助 USB 的复合类来实现，复合类是一个可以在一个 USB 设备中实现多个不同的功能的特殊 USB 类，如一个复合 USB 设备可以实现鼠标+键盘的功能，或者串口+键盘的功能。实际上，USB 复合类可以实现几乎任意的 USB 功能的组合，不仅仅只是两个功能的组合，还可以是三个甚至多个，所以可以使用复合类实现两个 CDC 子类或者多个 CDC 子类的功能，但由于物理节点数量的限制，LPC54018 和 LPC5500 上的全速和高速 USB 设备控制器最多只能支持两个 CDC 类，即一个 USB 设备控制器最多只能支持两个虚拟串口，若同时使用全速和高速 USB 设备控制器，则可以实现两个 USB 设备转四个虚拟串口的功能。

USB 描述符相当于 USB 设备的名片，描述了该 USB 设备所有的属性和可配置信息，如设备所属的类（Class）、接口（Interface）、端点（Endpoint）等信息。可以说，获得了设备的描述符，就知道了该设备的类型和用途、通信的参数等，主机就可以对它进行配置，从而使得通信双方使用相同的参数工作。

一个 CDC 类设备由两个子类接口组成： CDC 类接口和数据类接口。

- CDC 类接口使用标准的接口描述符，它需要一个中断输入端点，此端点是可选的。
- 数据类接口是通信设备必须配置的接口，它需要两个端点：Bulk IN, Bulk OUT。

由此可知，实现一个 CDC 类设备，需要用到两个接口，三个端点。如果要用一个 USB 设备实现两个 CDC 类，则需要四个接口，六个端点。

本例程中使用的包含了两个 CDC 子类的复合类的 USB 描述符结构如 图 1 所示。

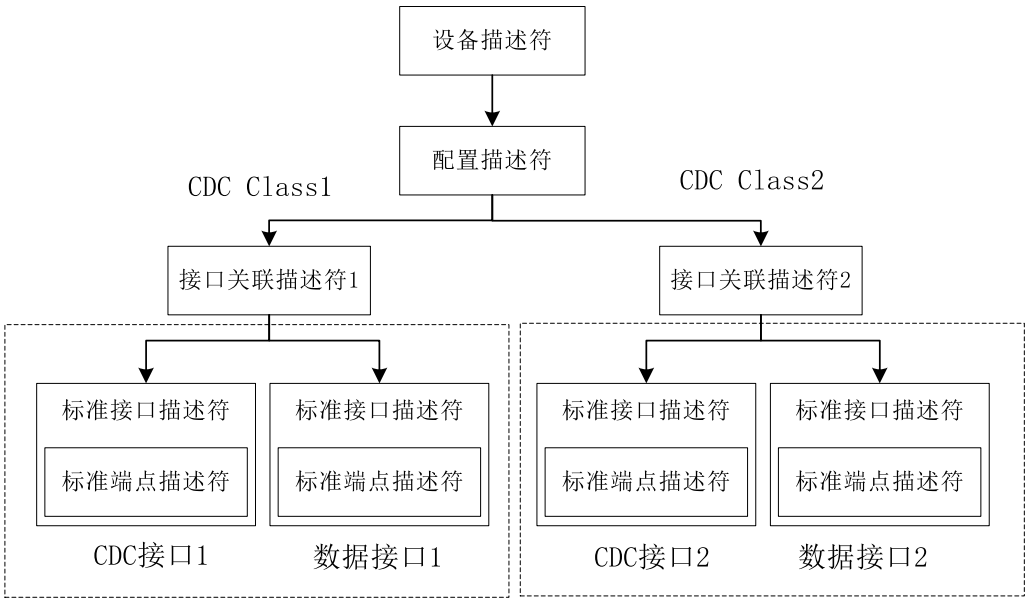


图 1. USB 描述符结构框图

图 1 中 CDC 接口 1 和数据接口 1 通过接口关联描述符 1 关联在一起共同描述一个虚拟串口的功能。 CDC 接口 2 和数据接口 2 通过接口关联描述符 2 关联在一起共同描述另一个虚拟串口的功能。

关于 USB 描述符中的细节，请参考 USB 规范 2.0 中的 9.6 章节和 SDK 代码。

3 接口的使用情况

由描述符的配置的描述可知每个 CDC 类需要三个物理端点，两个 CDC 类则需要六个物理端点，在同时使用了全速和高速 USB时，全速和高速 USB 设备的物理端点的使用情况如 表 1 和 表 2 所示。

表 1. 全速 USB 的物理端点使用情况

逻辑端点	物理端点	方向	端点类型	包大小 [1]	是否使用
EP0	0	OUT	Control	64	YES
EP0	1	IN	Control	64	YES
EP1	2	OUT	—	—	NO
EP1	3	IN	CDC1 interrupt	512	YES
EP2	4	OUT	—	—	NO
EP2	5	IN	CDC2 interrupt	512	YES
EP3	6	OUT	CDC1 bulk	512	YES
EP3	7	IN	CDC1 bulk	512	YES
EP4	8	OUT	CDC2 bulk	512	YES

Table continues on the next page...

表 1. 全速 USB 的物理端点使用情况（续）

逻辑端点	物理端点	方向	端点类型	包大小 [1]	是否使用
EP4	9	IN	CDC2 bulk	512	YES

1. 包大小是实际使用的包大小并不是最大的包大小

两个 CDC 子类使用了六个非控制端点，再加上两个控制端点，共需要八个物理端点，而全速 USB 设备只支持十个物理端点，因此最大只能支持两个虚拟串口。

与全速 USB 设备不同的是，高速 USB 设备支持 12 个物理（六个逻辑）端点，高速 USB 设备的物理端点使用情况如 表 2 所示。

表 2. 高速 USB 的物理端点使用情况

逻辑端点	物理端点	方向	端点类型	包大小 [1]	是否使用
EP0	0	OUT	Control	64	YES
EP0	1	IN	Control	64	YES
EP1	2	OUT	—	—	NO
EP1	3	IN	CDC1 interrupt	512	YES
EP2	4	OUT	—	—	NO
EP2	5	IN	CDC2 interrupt	512	YES
EP3	6	OUT	CDC1 bulk	512	YES
EP3	7	IN	CDC1 bulk	512	YES
EP4	8	OUT	CDC2 bulk	512	YES
EP4	9	IN	CDC2 bulk	512	YES
EP5	10	OUT	—	—	NO
EP5	11	IN	—	—	NO

1. 包大小是实际使用的包大小并不是最大的包大小

每个 CDC 类需要两个 IN 方向的物理端点和一个 OUT 方向的端点，由于高速 USB 设备中只剩一个未被使用的 IN 方向的端点，所以最大也只能支持两个虚拟串口。

4 USB RAM 的使用

每个物理端点都需要一个缓冲区来存储接收到的数据或者要发送的数据，本节将介绍本篇应用笔记中的 USB 端点缓冲区的配置情况。

其中高速 USB 设备只能使用 USB SRAM(0x4010 0000 0x4010 2000)区域作为端点缓冲区，高速 USB EPLIST 也必须放在 USB RAM 中。SDK 代码中为了兼容高速 USB 和全速 USB， 全速 USB 设备也使用 USB SRAM 区域作为端点缓冲区。

Table 3. Memory usage and details (mass market)

Address range	General Use	Address range details and description	
0x0000 0000 to 0x1FFF FFFF	SRAMX	0x0000 0000 - 0x0002 FFFF	I&D SRAM bank (192 kB).
	Boot ROM	0x0300 0000 - 0x0300 FFFF	Boot ROM with API services in a 64 kB space.
	SPI Flash Interface (SPIFI)	0x1000 0000 - 0x17FF FFFF	SPIFI memory mapped access space (128 MB).
0x2000 0000 to 0x3FFF FFFF	Main SRAM Banks	0x2000 0000 - 0x2002 7FFF	SRAM0, SRAM1, SRAM2, SRAM3 (total 160 kB).
	SRAM bit band alias addressing	0x2200 0000 - 0x23FF FFFF	SRAM bit band alias addressing (32 MB).
	SRAM Bank	0x4010 0000 0x4010 2000	USB SRAM (8 kB).

图 2. USB RAM 的分布

USB 端点缓冲区的配置需要用到两个寄存器: EPLISTSTART 和 DATABUFSTART。EPLISTSTART 用来指示 USB EP Command/Status List 的起始地址，在 SDK 代码中，EPLISTSTART 寄存器是在 *USB_DeviceLpc3511IpSetDefaultState（ ）*函数中配置的，指向了 s_EpCommandStatusList1 全局数组。

```
lpc3511IpState->registerBase->EPLISTSTART = (uint32_t)lpc3511IpState->epCommandStatusList;
```

其中 s_EpCommandStatusList1 数组的定义如下所示：

```
USB_CONTROLLER_DATA USB_RAM_ADDRESS_ALIGNMENT(256) static uint32_t
s_EpCommandStatusList1[((USB_DEVICE_IP3511_ENDPOINTS_NUM)) * 4];
```

它是一个 256 字节对齐的全局数据，存储在 USB RAM 中的一个 256 字节对齐的空间中。在使用端点进行数据传输时，各个端点使用的缓冲区如 图 3 所示。

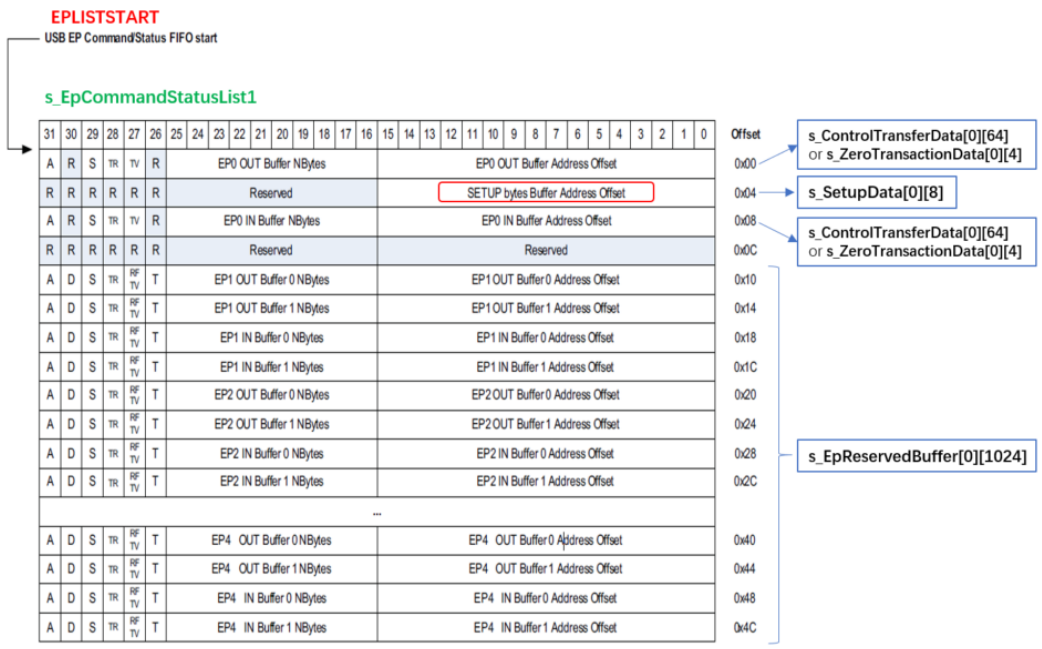


图 3. USB EPLIST 中的内容

注意

图中的内容是以全速 USB 设备为例的。本篇应用笔记中提到的所有的 OUT/IN 方向都是相对于 USB 主机来说的。

EPLIST 中的 EP OUT/IN Buffer Address offset 域中存放的是相应的缓冲区数组的地址的第 6-21 位。

s_SetupData、s_ControlTransferData、s_ZeroTransactionData、s_EpReservedBuffer 都为全局数组。

- s_SetupData 用来接收 setup 包数据。
- s_ControlTransferData 数组是作为控制端点 EP0 OUT 和 EP0 IN 端点的缓冲区，例如在处理标准请求时，在使用 EP0 IN 缓冲区发送数据或者 EP0 OUT 缓冲区接收数据前，需要将 s_ControlTransferData[0]的地址的第 6-21 位写入 EP OUT/IN Buffer Address offset 域中。
- s_ZeroTransactionData 是作为 0 长度数据包的，被用来向主机发送 0 长度数据包。
- s_EpReservedBuffer 数组是非控制端点的输入和输出缓冲区，每个物理端点使用数组中的 512 字节。

在使用 OUT 端点缓冲区接收数据或使用 IN 端点缓冲区向主机发送数据之前，需要先配置 EPLIST 中的相应的 EP OUT/IN Buffer Address offset 域，把即将用到的数组的地址的第 6-21 位写入到此域中，然后再设置相应的端点的 EPLIST 中的 ACTIVE 位，触发硬件进行数据的接收和发送。

在 SDK 中更新 EPLIST 的方法如下：

```
USB_LPC3511IP_ENDPOINT_SET_ENDPOINT(
    lpc3511IpState, endpointIndex, odd,
    (epState->stateUnion.stateBitField.epControlDefault <<
USB_LPC3511IP_ENDPOINT_CONFIGURE_BITS_SHIFT) |
        USB_LPC3511IP_ENDPOINT_ACTIVE_MASK,  length, (uint32_t)buffer);
```

其中 **USB_LPC3511IP_ENDPOINT_SET_ENDPOINT** 宏的定义如 图 4 所示。

```
122 #define USB_LPC3511IP_ENDPOINT_SET_ENDPOINT(lpcState, index, odd, value, NBytes, address)           \
123                                                                                                     \
124     *((volatile uint32_t *)(((uint32_t)(lpcState->epCommandStatusList)) | ((uint32_t)(index) << 3) | \
125                             (((uint32_t)(odd & 1U)) << 2U))) =                                      \
126         ((uint32_t)(value) | ((uint32_t)(NBytes) << USB_LPC3511IPFS_ENDPOINT_BUFFER_NBYTES_SHIFT) |  \
127          (((uint32_t)(address) >> 6) & USB_LPC3511IPFS_ENDPOINT_BUFFER_ADDRESS_OFFSET_MASK))
```

图 4. USB_LPC3511IP_ENDPOINT_SET_ENDPOINT 宏的定义

USB 主机和 USB 设备通过端点进行数据传输时，只知道缓冲区的地址偏移值是不够的，还需要知道这些缓冲区数组的基地址，基地址是在 DATABUFSTART 寄存器中设置的，DATABUFSTART 寄存器的低 22 位为 0，即指向了一个 4M 对齐的内存空间。由于这些端点缓冲区数组都要存储在 USB RAM[0x40100000-0x401020000]中，且 USB RAM 都被包含在一个以 0x40000000 为起始地址的 4 M 空间中，因此 DATABUFSTART 寄存器的值应为 0x40000000。

其中配置方法如下：

```
((USB_Type *)(lpc3511IpState->registerBase))->DATABUFSTART = (uint32_t)lpc3511IpState->setupData;
```

因此 s_SetupData、s_ControlTransferData、s_ZeroTransactionData、s_EpReservedBuffer 数组都存储在了 DATABUFSTART 指向的一个以 0x40000000 为基地址的 4M 对齐的内存中。

在定义 s_SetupData、s_ControlTransferData、s_ZeroTransactionData、s_EpReservedBuffer 数组时用 USB_GLOBAL 属性来修饰，将这四个数组放在 m_usb_global section 中，并在链接文件中，将 m_usb_global section 放 USB RAM 空间中。单独使用全速或高速 USB 设备时，USB RAM 的使用情况如 图 5 所示。

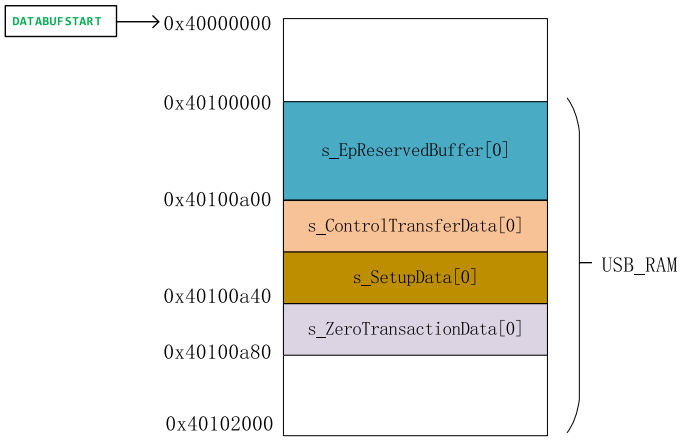


图 5. 使用一个 USB 设备时的 USB RAM 的使用情况

同时使用全速和高速 USB 时，USB RAM 的使用情况如 图 6 所示。

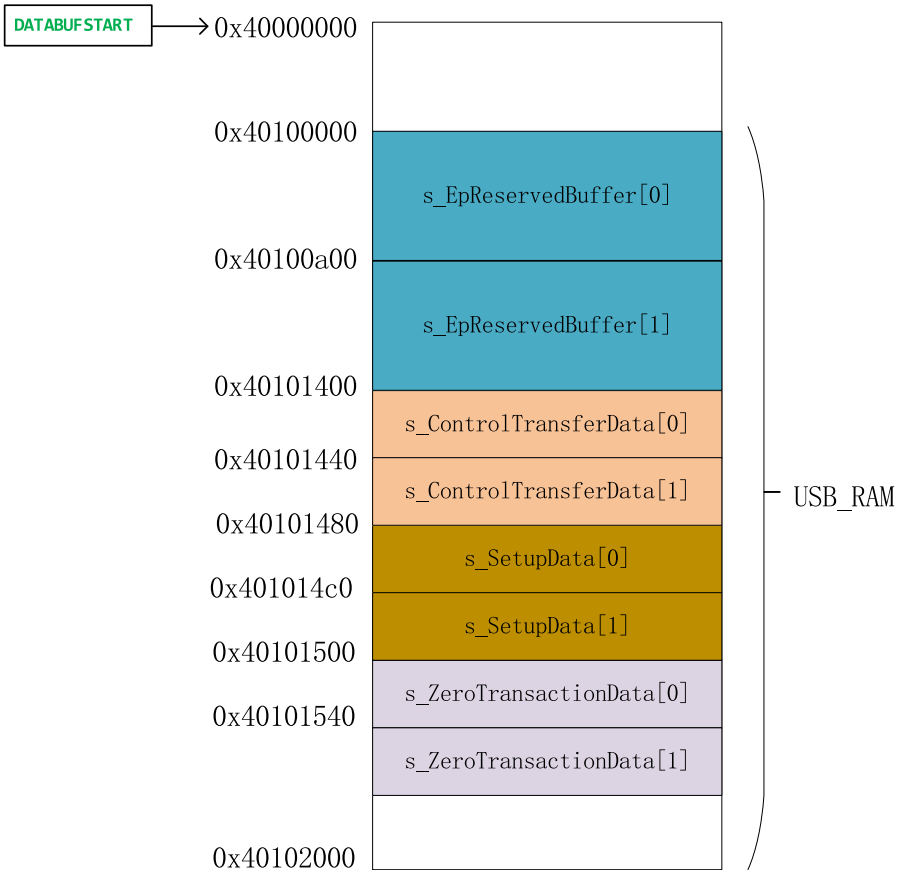


图 6. 同时使用全速和高速 USB 时 USB RAM 的使用情况

注意

为了使全速和高速 USB 同时使用 USB RAM，且不超过 USB RAM 的 8K 空间，本篇应用笔记中将宏 USB_DEVICE_IP3511_ENDPOINT_RESERVED_BUFFER_SIZE 的大小由（5*1024）修改为了 (5 * 512)。

5 软件工作流程

本篇应用笔记中使用的代码是在 SDK 中的 usb_device_composite_cdc_vcom_cdc_vcom_lite 例程的基础上修改而来的。USB 设备协议栈的基本工作流程依赖于回调函数和函数调用。回调函数通知应用程序协议中的所有状态变化和数据请求。在 USB 设备协议栈中有两类回调函数：Device 回调函数和端点回调函数。

- Device 回调函数：此回调函数通知应用程序整个协议栈的状态。
- 端点回调函数：此回调函数通知应用程序对应端点的数据传输结果。控制端点的回调函数处理所有的 USB 标准请求和类请求。

Lite 版代码中的回调函数是在应用层中实现的，回调函数的处理过程如 图 7 所示。

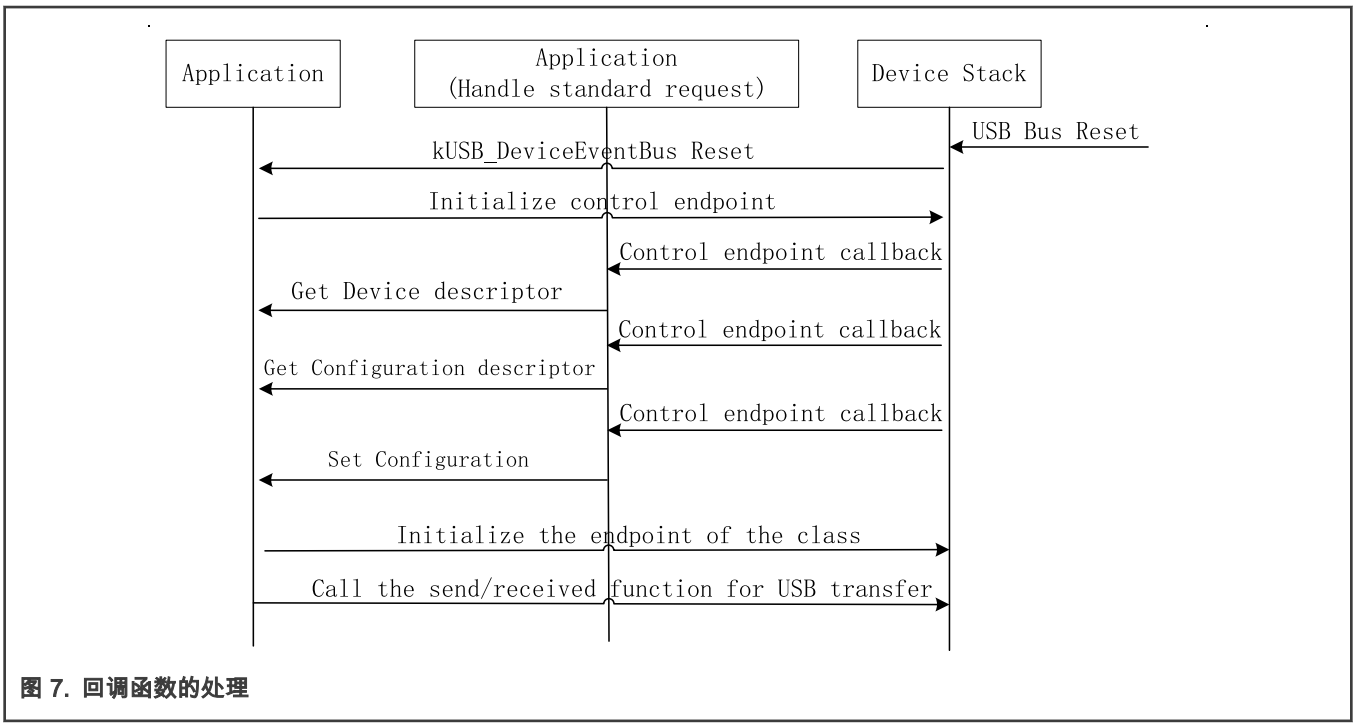


图 7. 回调函数的处理

当 USB 主机识别到有 USB 设备插入 USB 接口后，便会发起枚举行为。USB 枚举的本质就是 USB 主机获取 USB 设备的参数信息并且对于可配置参数进行配置的过程。USB 的枚举过程主要是在 USB 中断服务函数中完成的，本例程中全速和高速 USB 的中断服务函数的工作流程是相同的，这里以全速 USB 为例，USB0 的中断服务函数的处理流程如 图 8 所示。

5.1 USB 中断服务函数流程图

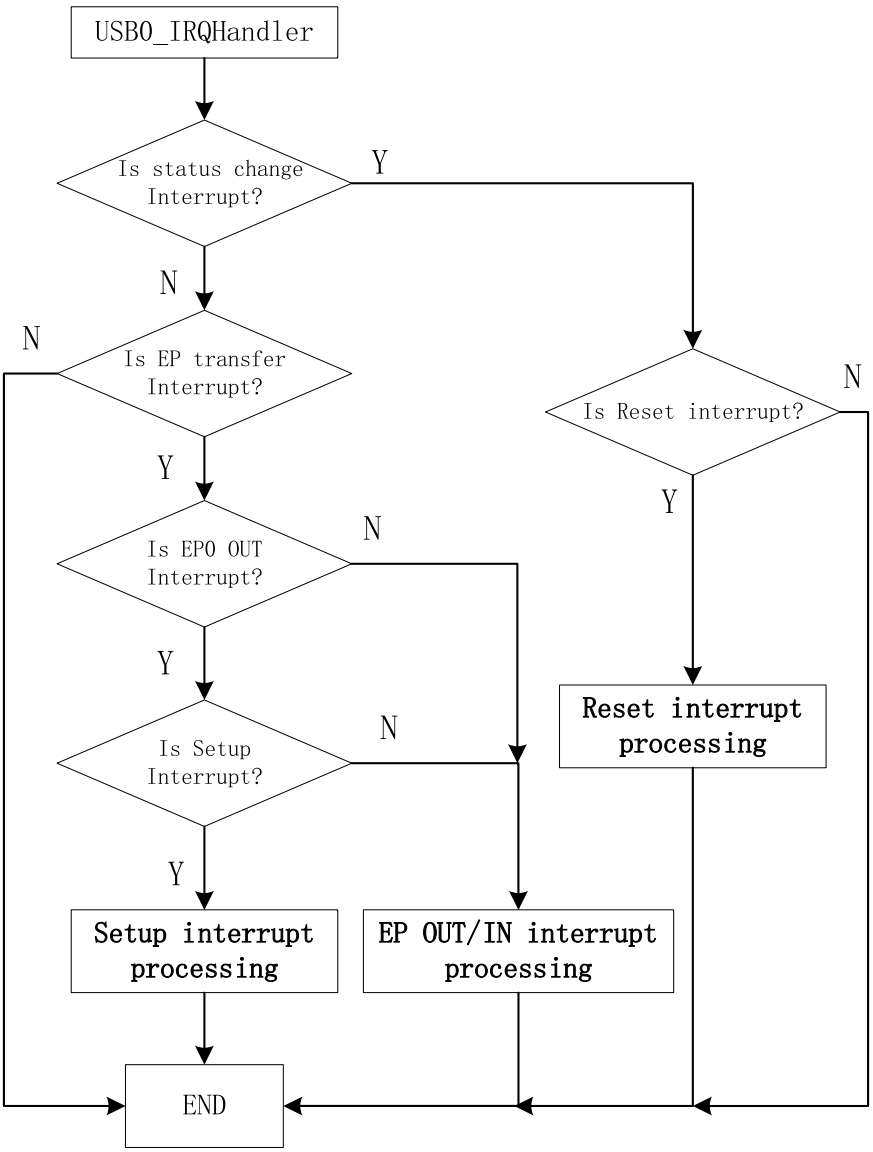


图 8. 全速 USB 中断服务函数的主流程图

在发生 USB0 中断并调用 USB0_IRQHandler 中断服务函数后，程序首先会判断是何种中断类型，是 USB 状态改变（Reset/Suspend/Resume）引起的中断还是端点传输中断（EP IN/OUT Interrupt）。如果为 EP0 OUT 中断，此时也要分两种情况：Setup 中断和普通 EP0 OUT 中断。

5.1.1 Reset 中断处理过程

若在中断服务主程序中判断出是状态改变引起的中断，且是复位中断，则程序会执行如 图 9 所示。

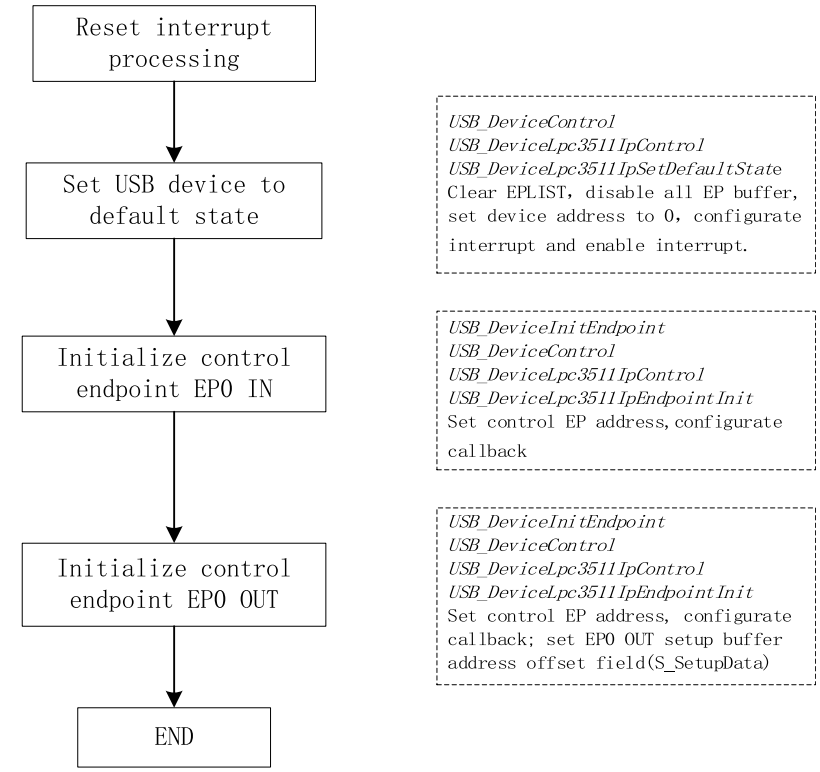


图 9. USB 设备的复位流程

在复位中断中主要完成的是设置 USB 设备为默认状态，并初始化控制端点 EP0 OUT 和 EP0 IN。

5.1.2 Setup 中断处理

若在中断服务主程序中判断出是 EP0 OUT 中断中的 Setup 中断，则执行如 图 10 所示的流程。

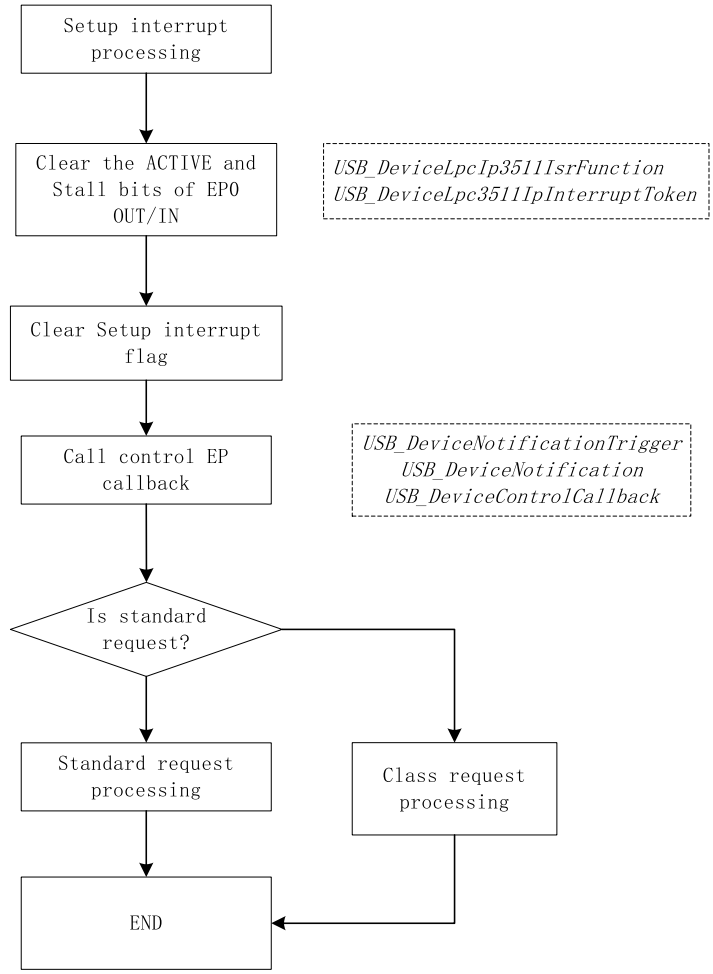


图 10. Setup 中断处理流程

USB 主机在向 USB 设备发送标准请求或者类请求时，会首先发送一个 Setup 事务，USB 设备在收到 Setup 事务并成功响应之后会产生 Setup 中断, USB 设备在 Setup 中断中判断是何种请求，并根据不同的请求类型做出不同的操作。

5.1.2.1 标准请求处理

若 USB 设备根据 Setup 包中的内容判断出是主机发出的标准请求，则会执行如 图 11 所示的流程。

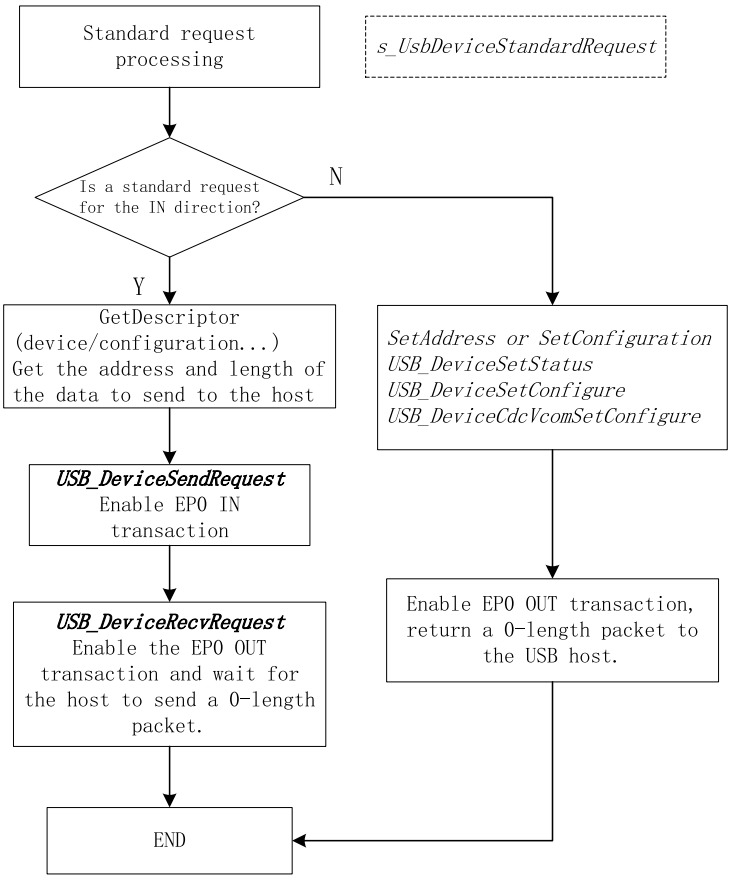


图 11. 标准请求处理过程

根据 Setup 包中的数据判断这个标准请求是 OUT 方向的还是 IN 方向的。若为 OUT 方向的请求，则准备接收下一个 OUT 事务；若为 IN 方向的请求，则准备向 USB 主机返回 IN 事务。常见的标准请求如 表 3 所示。

表 3. 常用标准请求

标准请求	方向
SetAddress	OUT
SetConfiguration	OUT
GetDescriptor(Device/Configuration/String)	IN

5.1.2.2 类请求处理

CDC 类设备除了支持标准请求处理外，还支持一些类请求。USB 主机在获取 CDC 类设备的各类描述符信息之后，还会向 USB设备发送类请求，CDC 类设备常用的类请求如 表 4 所示。

表 4. 常用 CDC 类请求

类请求	方向
SetLineCoding	OUT

Table continues on the next page...

表 4. 常用 CDC 类请求（续）

类请求	方向
SetControlLineState	OUT
GetLineCoding	IN

其中 GetLineCoding 请求是主机获取串口属性的请求，包括波特率设置、停止位设置、校验类型以及数据位位数。GetLineCoding请求的结构如 表 5 所示。

表 5. GetLineCoding 请求的结构

bmRequestType	bRequestCode	wValue	wIndex	wLength	Data
10100001B	GET_LINE_CODING	Zero	Interface	Size of Structure	Line Coding Structure

其中 Data 域的内容如所示。

表 6. Line Coding 数据结构

偏移量	域	大小/字节	描述
0	dwDTERate	4	Data terminal rate, in bits per second
4	bCharFormat	1	Stop bits 0: 1 Stop bit 1: 1.5 Stop bit 2: 2 Stop
5	bParityType	1	Parity 0: None 1: Odd 2: Even 3: Mark 4: Space
6	bDataBits	1	Data bits (5,6,7,8, or 16)

在本例程中 Line Coding 结构体的内容如 图 12 所示。

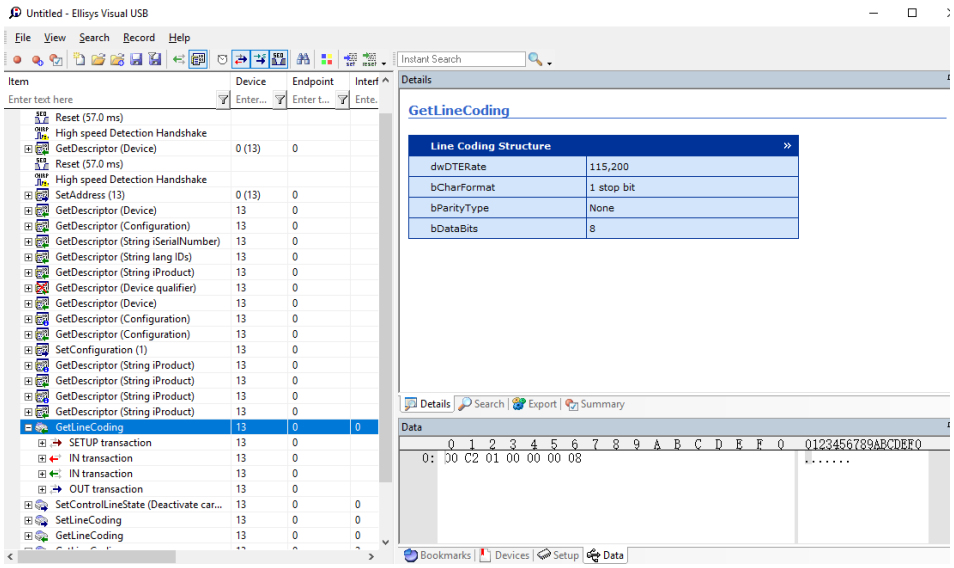


图 12. GetLineCoding 数据

从 图 12 中可以看出，虚拟串口的配置为：波特率为 115200, 1 个停止位, 无校验位，8 个数据位。若 USB 设备根据 Setup 包中的内容判断出是主机发出的类请求，则会执行如 图 13 所示的流程。

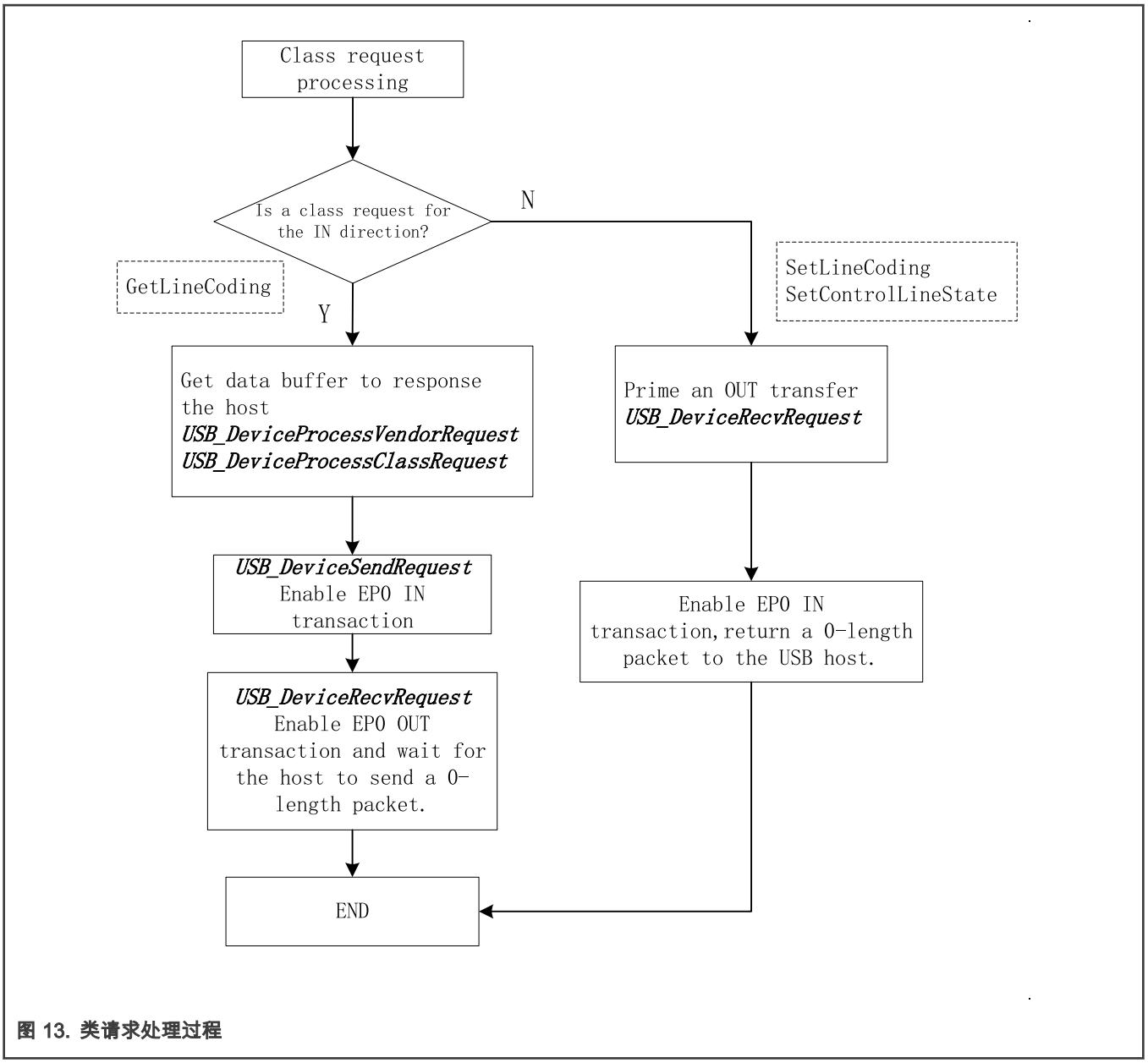


图 13. 类请求处理过程

根据 Setup 包中的数据判断是 OUT 方向的标准请求还是 IN 方向的请求。若为 OUT 方向的请求，则准备接收下一个 OUT 事务；若为 IN 方向的请求，则准备向 USB 主机返回 IN 事务。

5.1.3 端点中断处理

若在中断服务主程序中判断出是端点中断，执行如 图 14 所示的流程进行端点数据的接收和发送。

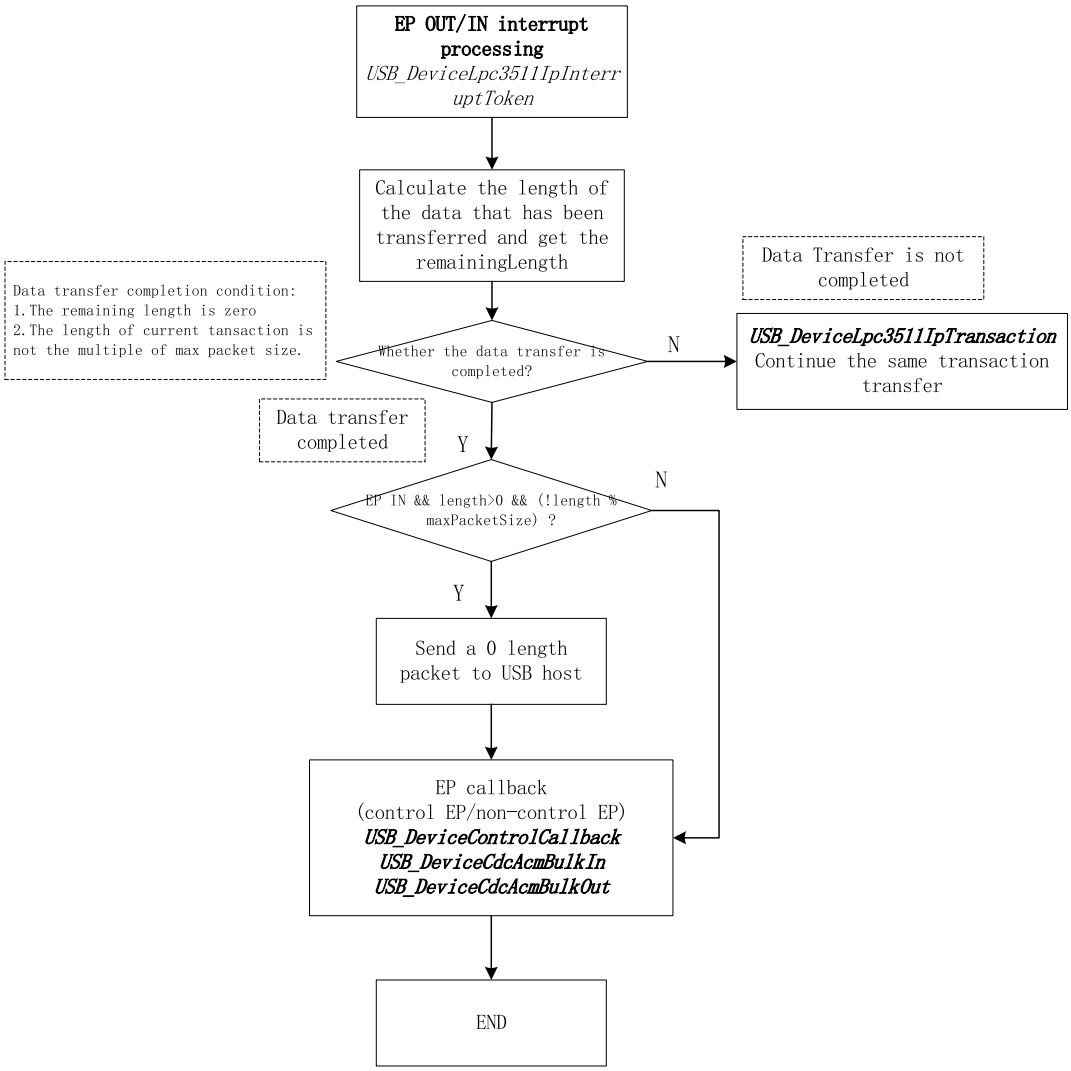


图 14. 端点中断处理流程

在处理端点传输中断时，首先会计算已经传输的数据长度和剩余的需要传输的数据长度，判断数据是否传输完成。

传输完成的标志：

- Length>0, 且不为最大包长度的整数倍
- RemainLength 等于 0

若数据还没发送完成，则继续传输相同的事务。如果数据发送完成，若为 IN 方向中断，且最后的数据包长度等于最大包长度，还需要发送一个 0 长度数据包告诉主机发送完成。若数据传输完成，则调用相应的端点回调函数。在本例程中，SDK 提供的端点回调函数如 表 7 所示。

表 7. 端点回调函数

端点	回调函数
EP0 OUT	USB_DeviceControlCallback()
EP0 IN	USB_DeviceControlCallback()

Table continues on the next page...

表 7. 端点回调函数（续）

端点	回调函数
EP1 IN	USB_DeviceCdcAcmBulkIn()
EP2 IN	USB_DeviceCdcAcmBulkIn2()
EP3 OUT	USB_DeviceCdcAcmBulkOut()
EP3 IN	USB_DeviceCdcAcmBulkIn()
EP4 OUT	USB_DeviceCdcAcmBulkOut2()
EP4 IN	USB_DeviceCdcAcmBulkIn2()

在本例程中，四个非控制端点的回调函数实现是相同的功能：更新接收到的数据长度并将端点缓冲区中的数据（s_EpReserveBuffer）拷贝至全局接收数组(s_currRecvBuf)中。然后在 while 循环中，将数据 s_currRecvBuf 返回给主机。

6 FS 和 HS 的配置及验证

在本篇应用笔记中，全速 USB 和高速 USB 描述符的配置是完全相同，即它们共用相同的描述符数组，全速和高速 USB 的端点回调函数的功能也是相同的，都是将接收到的数据再返回给 USB 主机。若只想使用一个 USB 设备，则只需要修改相应的宏定义即可。

1. 只使用全速 USB

```
/*! @brief LPC USB IP3511 FS instance count */
#define USB_DEVICE_CONFIG_LPCIP3511FS (1U)
/*! @brief LPC USB IP3511 HS instance count */
#define USB_DEVICE_CONFIG_LPCIP3511HS (0U)
```

2. 只使用高速 USB

```
/*! @brief LPC USB IP3511 FS instance count */
#define USB_DEVICE_CONFIG_LPCIP3511FS (0U)
/*! @brief LPC USB IP3511 HS instance count */
#define USB_DEVICE_CONFIG_LPCIP3511HS (1U)
```

3. 同时使用全速 USB 和高速 USB

```
/*! @brief LPC USB IP3511 FS instance count */
#define USB_DEVICE_CONFIG_LPCIP3511FS (1U)
/*! @brief LPC USB IP3511 HS instance count */
#define USB_DEVICE_CONFIG_LPCIP3511HS (1U)
```

若同时使用全速和高速 USB，程序运行后，USB 主机会识别出四个虚拟串口，如 图 15 所示。

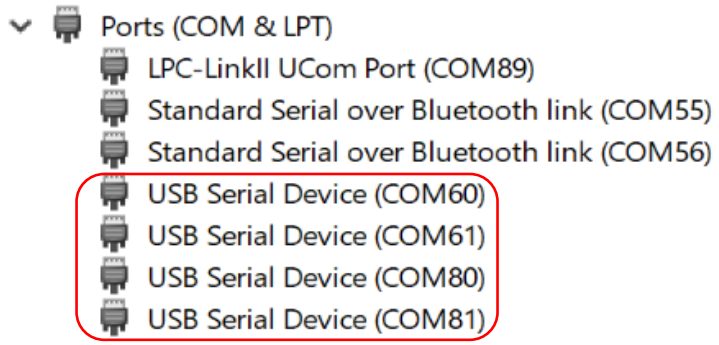
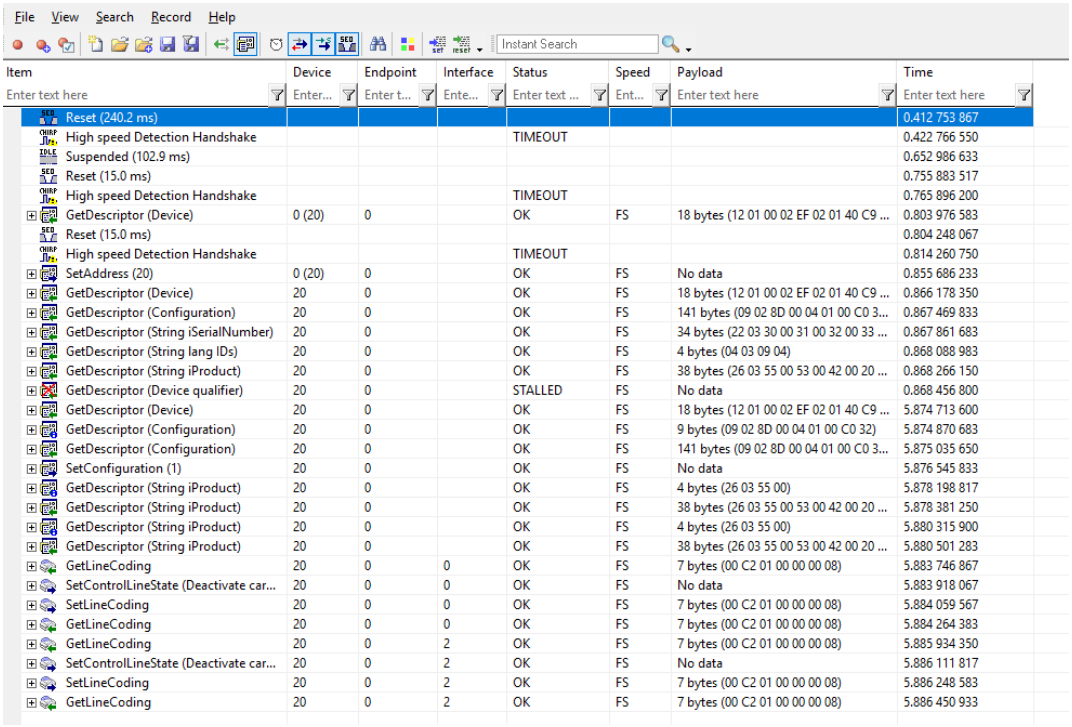


图 15. USB 主机识别的结果

使用 USB 分析仪抓取到的 USB 主机的枚举过程如 图 16 所示。



Item	Device	Endpoint	Interface	Status	Speed	Payload	Time
Reset (240.2 ms)							0.412 753 867
High speed Detection Handshake				TIMEOUT			0.422 766 550
Suspended (102.9 ms)							0.652 986 633
Reset (15.0 ms)							0.755 883 517
High speed Detection Handshake				TIMEOUT			0.765 896 200
GetDescriptor (Device)	0 (20)	0		OK	FS	18 bytes (12 01 00 02 EF 02 01 40 C9 ...	0.803 976 583
Reset (15.0 ms)							0.804 248 067
High speed Detection Handshake				TIMEOUT			0.814 260 750
SetAddress (20)	0 (20)	0		OK	FS	No data	0.855 686 233
GetDescriptor (Device)	20	0		OK	FS	18 bytes (12 01 00 02 EF 02 01 40 C9 ...	0.866 178 350
GetDescriptor (Configuration)	20	0		OK	FS	141 bytes (09 02 8D 00 04 01 00 C0 3...	0.867 469 833
GetDescriptor (String iSerialNumber)	20	0		OK	FS	34 bytes (22 03 30 00 31 00 32 00 33 ...	0.867 861 683
GetDescriptor (String lang IDs)	20	0		OK	FS	4 bytes (04 03 09 04)	0.868 088 983
GetDescriptor (String iProduct)	20	0		OK	FS	38 bytes (26 03 55 00 53 00 42 00 20 ...	0.868 266 150
GetDescriptor (Device qualifier)	20	0		STALLED	FS	No data	0.868 456 800
GetDescriptor (Device)	20	0		OK	FS	18 bytes (12 01 00 02 EF 02 01 40 C9 ...	5.874 713 600
GetDescriptor (Configuration)	20	0		OK	FS	9 bytes (09 02 8D 00 04 01 00 C0 32)	5.874 870 683
GetDescriptor (Configuration)	20	0		OK	FS	141 bytes (09 02 8D 00 04 01 00 C0 3...	5.875 035 650
SetConfiguration (1)	20	0		OK	FS	No data	5.876 545 833
GetDescriptor (String iProduct)	20	0		OK	FS	4 bytes (26 03 55 00)	5.878 198 817
GetDescriptor (String iProduct)	20	0		OK	FS	38 bytes (26 03 55 00 53 00 42 00 20 ...	5.878 381 250
GetDescriptor (String iProduct)	20	0		OK	FS	4 bytes (26 03 55 00)	5.880 315 900
GetDescriptor (String iProduct)	20	0		OK	FS	38 bytes (26 03 55 00 53 00 42 00 20 ...	5.880 501 283
GetLineCoding	20	0	0	OK	FS	7 bytes (00 C2 01 00 00 00 08)	5.883 746 867
SetControlLineState (Deactivate car...	20	0	0	OK	FS	No data	5.883 918 067
SetLineCoding	20	0	0	OK	FS	7 bytes (00 C2 01 00 00 00 08)	5.884 059 567
GetLineCoding	20	0	0	OK	FS	7 bytes (00 C2 01 00 00 00 08)	5.884 264 383
GetLineCoding	20	0	2	OK	FS	7 bytes (00 C2 01 00 00 00 08)	5.885 934 350
SetControlLineState (Deactivate car...	20	0	2	OK	FS	No data	5.886 111 817
SetLineCoding	20	0	2	OK	FS	7 bytes (00 C2 01 00 00 00 08)	5.886 248 583
GetLineCoding	20	0	2	OK	FS	7 bytes (00 C2 01 00 00 00 08)	5.886 450 933

图 16. USB 枚举过程

四个虚拟串口都实现了将接收到的主机的数据返回给主机的功能，测试结果如 图 17 所示。

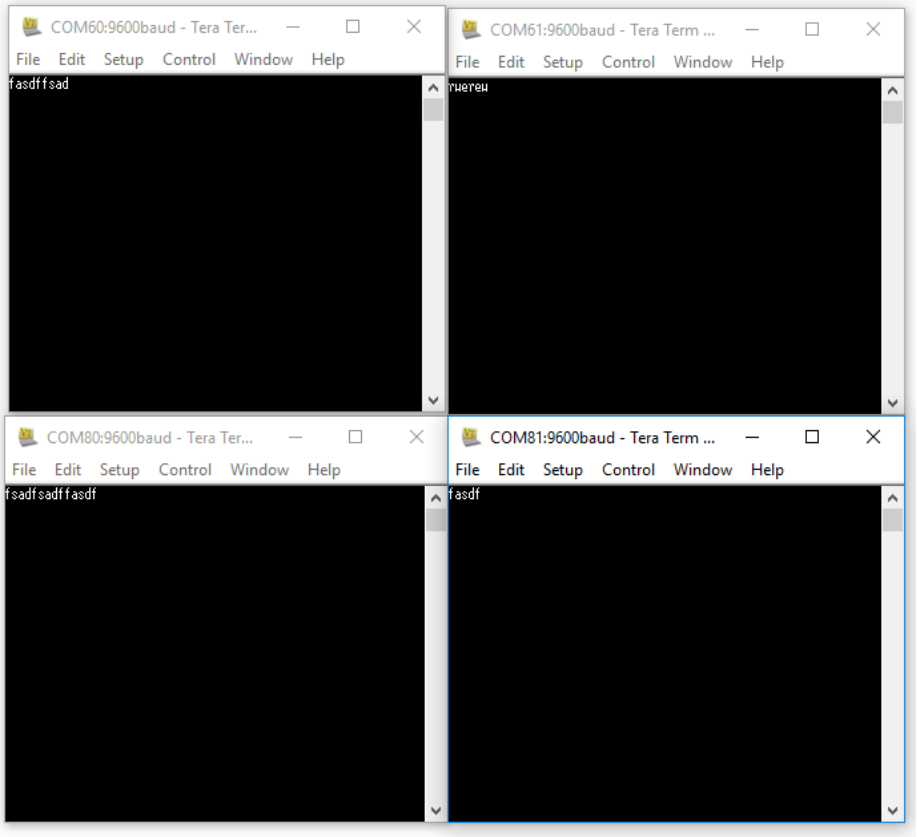


图 17. 虚拟串口的测试

7 总结

本篇应用笔记通过使用包含两个 CDC 子类的 USB 复合类，实现了一个 USB 设备支持两个 VCOM 的功能；并通过同时使能全速和高速 USB 设备，实现了两个 USB 设备转四个虚拟串口的功能，且在 LPC54018 和 LPC5500 上验证是可行的。

8 参考文献

1. *USB virtual COM port on LPC214x* (document AN10420)
2. USB 2.0 Specification
3. 微控制器 USB 的信号和协议实现
4. 微控制器 USB 的技术及应用入门

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

Limited warranty and liability — Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

Right to make changes - NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Security — Customer understands that all NXP products may be subject to unidentified or documented vulnerabilities. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer’s applications and products. Customer’s responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer’s applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately. Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP. NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

For more information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 2019 年 6 月

Document identifier: AN12458

arm