AN14322

在MCX C444系列MCU上实现USB转多个虚拟串口

第1版—2024年7月15日 应用笔记

文档信息

信息	内容
关键词	AN14322、MCX C、USB、虚拟串口（VCOM）
摘要	本文介绍了如何在MCX C444系列FRDM开发板（FRDM-MCXC444）上实现一个USB转多个虚拟串口（VCOM）的功能。

1 简介

本应用笔记介绍了如何在MCX C444系列FRDM开发板（FRDM-MCXC444）上实现一个USB转多个虚拟串口（VCOM）的功能。要实现USB转VCOM的功能，可使用USB协议规定的CDC类抽象控制模型子类中的常用AT命令。一个USB设备可以支持一个或多个VCOM，VCOM的数量主要取决于USB设备所支持的端点（EP）的数量。MCX C444系列MCU的全速（FS）USB设备控制器支持16个双向端点，并最多支持15个VCOM。本应用笔记基于SDK代码（dev_composite_cdc_vcom_cdc_vcom_lite_bm），实现了一个USB转15个VCOM的功能，所使用的开发工具为MCUXpresso IDE。

本应用笔记适用于MCX C444和MCX C242，本文仅以MCX C444为例。

2 USB描述符的配置

USB描述符相当于USB设备的名片。它描述了USB设备的所有属性和可配置信息，如类、接口信息和端点信息等。如果获得了设备的描述符，就可以知道该设备通信的类型、用途和参数等。USB主机可以对其进行配置，使通信双方以相同的参数进行工作。

一个USB转多个VCOM的功能可以通过使用USB复合（Composite）类来实现。复合类是一个特殊的USB类，可以在USB设备中实现多种不同的功能。例如一个设备可以实现**鼠标+键盘**功能，或者**虚拟串口+键盘**功能。实际上，USB复合类可以实现几乎任何USB功能的组合，而不只是两个功能的组合，还可以是三个或者更多，因此可以用复合类实现两个CDC或者多个CDC的功能。

CDC类设备由两个子类接口组成：CDC类接口和数据类接口。一个CDC设备可以包含零个或者多个数据接口。

- CDC类接口使用标准接口描述符，需要一个控制端点和一个可选的中断IN端点。
- 数据类接口需要一个Bulk IN和Bulk OUT类型的端点。但由于MCXC444上的16个端点都是双向端点，所以一个端点就可以同时实现输入和输出的功能，也就是说Bulk IN和Bulk OUT的功能可以由一个端点实现。

为了支持更多的VCOM，可以去掉CDC类接口中的中断IN端点，这样除了一个公共控制端点外，只需要一个双向端点即可实现VCOM。也就是说，除了使用EP0作为控制端点外，其他15个端点可以实现15个VCOM，本应用笔记就实现了这种功能。本应用笔记中使用的包含15个CDC子类的复合类的描述符结构如图1所示。

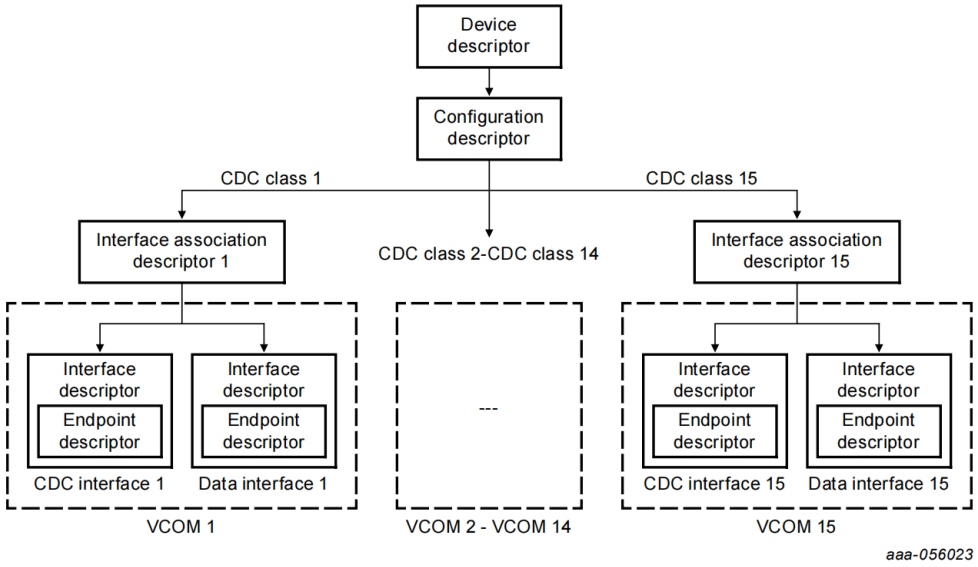

图1. USB描述符的结构框图

在图1中，接口关联描述符将CDC接口与数据接口关联起来，以描述VCOM的功能。有关USB描述符的详细信息，请参阅第9.6章“标准USB描述符定义和USB规范2.0”以及SDK代码。

注：*如果去掉了CDC接口的中断端点，则无法发送和接收串口状态。例如，RS232中的RTS和DTR信号将无法传输。如果个人计算机上的串口调试终端已经打开，则仍然可以发送和接收串口数据。*

3 端点的使用

如第2节所述，每个CDC类都需要1个双向端点，则15个CDC类就需要15个双向端点。EP0用作控制端点。本文档中USB设备端点的使用情况如表1所示。

表1. 端点的使用

端点	方向	端点类型	包大小（字节）	是否使用
EP0	OUT（输出）	控制	64	是
EP0	IN（输入）	控制	64	是
EP1	OUT	CDC1 Bulk	64	是
EP1	IN	CDC1 Bulk	64	是
EP2	OUT	CDC2 Bulk	64	是
EP2	IN	CDC2 Bulk	64	是
EP3	OUT	CDC3 Bulk	64	是
EP3	IN	CDC3 Bulk	64	是
EP4	OUT	CDC4 Bulk	64	是
EP4	IN	CDC4 Bulk	64	是
EP5	OUT	CDC5 Bulk	64	是
EP5	IN	CDC5 Bulk	64	是

表1. 端点的使用（续）

端点	方向	端点类型	包大小（字节）	是否使用
EP6	OUT	CDC6 Bulk	64	是
EP6	IN	CDC6 Bulk	64	是
EP7	OUT	CDC7 Bulk	64	是
EP7	IN	CDC7 Bulk	64	是
EP8	OUT	CDC8 Bulk	64	是
EP8	IN	CDC8 Bulk	64	是
EP9	OUT	CDC9 Bulk	64	是
EP9	IN	CDC9 Bulk	64	是
EP10	OUT	CDC10 Bulk	64	是
EP10	IN	CDC10 Bulk	64	是
EP11	OUT	CDC11 Bulk	64	是
EP11	IN	CDC11 Bulk	64	是
EP12	OUT	CDC12 Bulk	64	是
EP12	IN	CDC12 Bulk	64	是
EP13	OUT	CDC13 Bulk	64	是
EP13	IN	CDC13 Bulk	64	是
EP14	OUT	CDC14 Bulk	64	是
EP14	IN	CDC14 Bulk	64	是
EP15	OUT	CDC15 Bulk	64	是
EP15	IN	CDC15 Bulk	64	是

图2所示为*usb_device_descriptor.h*文件中端点的配置。

```
/* Endpoint usage */
#define USB_CDC_VCOM_DIC_BULK_IN_ENDPOINT (1)
#define USB_CDC_VCOM_DIC_BULK_OUT_ENDPOINT (1)
#define USB_CDC_VCOM_DIC_BULK_IN_ENDPOINT_2 (2)
#define USB_CDC_VCOM_DIC_BULK_OUT_ENDPOINT_2 (2)
#define USB_CDC_VCOM_DIC_BULK_IN_ENDPOINT_3 (3)
#define USB_CDC_VCOM_DIC_BULK_OUT_ENDPOINT_3 (3)
#define USB_CDC_VCOM_DIC_BULK_IN_ENDPOINT_4 (4)
#define USB_CDC_VCOM_DIC_BULK_OUT_ENDPOINT_4 (4)
#define USB_CDC_VCOM_DIC_BULK_IN_ENDPOINT_5 (5)
#define USB_CDC_VCOM_DIC_BULK_OUT_ENDPOINT_5 (5)
#define USB_CDC_VCOM_DIC_BULK_IN_ENDPOINT_6 (6)
#define USB_CDC_VCOM_DIC_BULK_OUT_ENDPOINT_6 (6)
#define USB_CDC_VCOM_DIC_BULK_IN_ENDPOINT_7 (7)
#define USB_CDC_VCOM_DIC_BULK_OUT_ENDPOINT_7 (7)
#define USB_CDC_VCOM_DIC_BULK_IN_ENDPOINT_8 (8)
#define USB_CDC_VCOM_DIC_BULK_OUT_ENDPOINT_8 (8)
#define USB_CDC_VCOM_DIC_BULK_IN_ENDPOINT_9 (9)
#define USB_CDC_VCOM_DIC_BULK_OUT_ENDPOINT_9 (9)
#define USB_CDC_VCOM_DIC_BULK_IN_ENDPOINT_10 (10)
#define USB_CDC_VCOM_DIC_BULK_OUT_ENDPOINT_10 (10)
#define USB_CDC_VCOM_DIC_BULK_IN_ENDPOINT_11 (11)
#define USB_CDC_VCOM_DIC_BULK_OUT_ENDPOINT_11 (11)
#define USB_CDC_VCOM_DIC_BULK_IN_ENDPOINT_12 (12)
#define USB_CDC_VCOM_DIC_BULK_OUT_ENDPOINT_12 (12)
#define USB_CDC_VCOM_DIC_BULK_IN_ENDPOINT_13 (13)
#define USB_CDC_VCOM_DIC_BULK_OUT_ENDPOINT_13 (13)
#define USB_CDC_VCOM_DIC_BULK_IN_ENDPOINT_14 (14)
#define USB_CDC_VCOM_DIC_BULK_OUT_ENDPOINT_14 (14)
#define USB_CDC_VCOM_DIC_BULK_IN_ENDPOINT_15 (15)
#define USB_CDC_VCOM_DIC_BULK_OUT_ENDPOINT_15 (15)
```

图2. 端点配置

4 端点缓冲区的配置

每个端点都需要一个缓冲区来存储接收到的数据或要发送的数据。本章介绍了USB端点缓冲区的配置。

4.1 缓冲区描述符表

为了管理USB端点通信，USBFS在系统内存中实现了一个缓冲区描述符表（BDT），如图3所示。BDT位于系统内存中的512字节边界上，并由BDT页寄存器指向。每个端点方向都需要两个8字节的缓冲区描述符（BD）条目。因此，具有16个全双向端点的系统将需要512字节的系统内存来实现BDT。这两个BD条目允许每个端点方向有一个偶数BD条目和一个奇数BD条目。BDT中存储的内容如表2所示。

表2. 缓冲区描述符格式

31:26	25:16	15.8	7	6	5	4	3	2	1	0
RSVD	BC（10位）	RSVD	OWN	DATA0/1	KEEP/ TOK_ PID[3]	NINC/ TOK_ PID[2]	DTS/ TOK_ PID[1]	BDT_ STALL/ TOK_ PID[0]	0	0

表2. 缓冲区描述符格式（续）

31:26	25:16	15.8	7	6	5	4	3	2	1	0
缓冲区地址（32位）										

EP缓冲区的地址存储在缓冲区地址字段中。BDT存储在512字节对齐的空间中，BDT的位置可以由BDT页寄存器确定：BDT页寄存器1、BDT页寄存器2和BDT页寄存器3。这三个寄存器提供了BDT地址的第9位至第31位，其中第0位到第8位为0，如表3所示。

表3. BDT地址

	BDT页寄存器3	BDT页寄存器2	BDT页寄存器1									
BDT地址	31位 – 24位	23位 - 16位	15位 - 9位	0	0	0	0	0	0	0	0	0

为了计算进入BDT的入口点，将BDT_PAGE寄存器与当前端点以及TX和ODD字段连接起来，形成了一个32位的地址。此地址机制如表4所示。

表4. BDT地址计算

31:24	23:16	15:9	8:5	4	3	2
BDT_PAGE_03	BDT_PAGE_02	BDT_PAGE_01[7:1]	端点	TX	ODD	000

表5. BDT的地址计算字段

字段	说明
BDT_PAGE	控制寄存器块中的BDT_PAGE寄存器
ENDPOINT	USB TOKEN中的ENDPOINT字段
TX	1用于发送传输，0用于接收传输
ODD	保留在USBFS SIE中。对应于当前使用的缓冲区。缓冲区以乒乓方式使用。

根据表4中几个字段的值，可以在BDT中找到对应的BD和EP缓冲区，如图3所示。

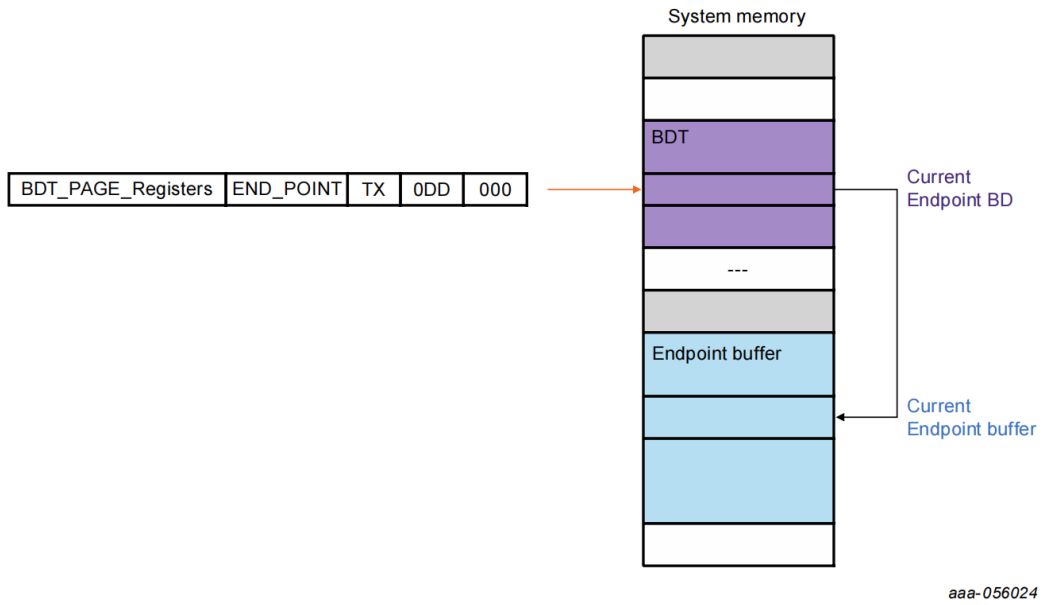

图3. 缓冲区描述符表

软件通过在需要时更新BDT来管理USBFS的缓冲区。这使得USBFS可以管理数据的发送和接收，而微处理器则执行通信开销处理和其他相关功能的应用程序。

4.2 端点缓冲区

控制端点的缓冲区是不固定的。当接收到setup包数据时，EP0缓冲区为`s_UsbDevice KhciState.setupPacketBuffer[8*2]`。当处理IN事务时，EP0缓冲区通常是一个描述符数组：`g_UsbDeviceDescriptor[]`、`g_UsbDeviceConfigurationDescriptor[]`等。非控制端点的缓冲区配置如表6所示。

表6. EP缓冲区的配置

VCOM序号	端点	方向	EP缓冲区
VCOM1	1	OUT	`s_currRecvBuf[0][64]`
VCOM1	1	IN	`s_currSendBuf[0][64]`
VCOM2	2	OUT	`s_currRecvBuf[1][64]`
VCOM2	2	IN	`s_currSendBuf[1][64]`
VCOM3	3	OUT	`s_currRecvBuf[2][64]`
VCOM3	3	IN	`s_currSendBuf[2][64]`
VCOM4	4	OUT	`s_currRecvBuf[3][64]`
VCOM4	4	IN	`s_currSendBuf[3][64]`
VCOM5	5	OUT	`s_currRecvBuf[4][64]`
VCOM5	5	IN	`s_currSendBuf[4][64]`
VCOM6	6	OUT	`s_currRecvBuf[5][64]`
VCOM6	6	IN	`s_currSendBuf[5][64]`
VCOM7	7	OUT	`s_currRecvBuf[6][64]`

表6. EP缓冲区的配置（续）

VCOM序号	端点	方向	EP缓冲区
VCOM7	7	IN	s_currSendBuf[6][64]
VCOM8	8	OUT	s_currRecvBuf[7][64]
VCOM8	8	IN	s_currSendBuf[7][64]
VCOM9	9	OUT	s_currRecvBuf[8][64]
VCOM9	9	IN	s_currSendBuf[8][64]
VCOM10	10	OUT	s_currRecvBuf[9][64]
VCOM10	10	IN	s_currSendBuf[9][64]
VCOM11	11	OUT	s_currRecvBuf[10][64]
VCOM11	11	IN	s_currSendBuf[10][64]
VCOM12	12	OUT	s_currRecvBuf[11][64]
VCOM12	12	IN	s_currSendBuf[11][64]
VCOM13	13	OUT	s_currRecvBuf[12][64]
VCOM13	13	IN	s_currSendBuf[12][64]
VCOM14	14	OUT	s_currRecvBuf[13][64]
VCOM14	14	IN	s_currSendBuf[13][64]
VCOM15	15	OUT	s_currRecvBuf[14][64]
VCOM15	15	IN	s_currSendBuf[14][64]

s_currRecvBuf和s_currSendBuf是两个全局数组，定义如下：

```
USB_DMA_NONINIT_DATA_ALIGN(USB_DATA_ALIGN_SIZE) static uint8_t
s_currRecvBuf[USB_DEVICE_CONFIG_CDC_ACM][DATA_BUFF_SIZE];
USB_DMA_NONINIT_DATA_ALIGN(USB_DATA_ALIGN_SIZE) static uint8_t
 s_currSendBuf[USB_DEVICE_CONFIG_CDC_ACM][DATA_BUFF_SIZE];
```

宏DATA_BUFF_SIZE的值为64。

OUT方向的非控制端点的缓冲区配置是在USB设备接收到主机的SetConfiguration标准请求之后，在USB_DeviceCdcVcomSetConfigure()函数中执行的，而在使用此IN端点发送数据之前，会配置IN方向的EP缓冲区。

5 软件工作流程图

本应用笔记中所用的代码是基于SDK中的`usb_device_composite_cdc_vcom_cdc_vcom_lite`示例的。USB设备协议栈的架构如图4所示。

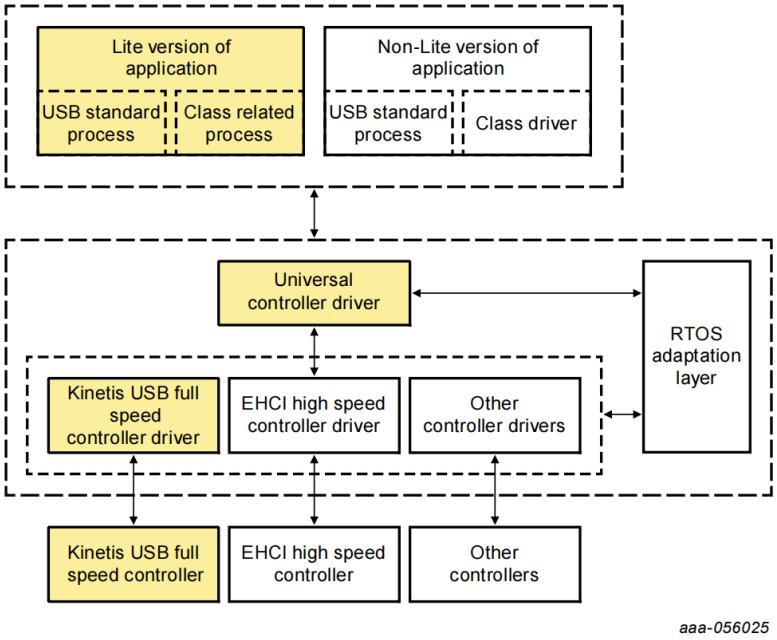

图4. SDK USB设备协议栈架构

USB设备协议栈是该架构的中间部分，由通用控制器驱动程序、专用控制器驱动程序和实时操作系统（RTOS）适配层组成。在本应用笔记中，采用了该应用程序的精简版本，即架构图中标有黄色的部分。

- 通用控制器驱动程序：为应用程序提供统一的接口，用户无需关心不同硬件控制器的详细信息。
- 特定的控制器驱动程序：USB控制器的类型很多，每种控制器都能实现一个特定的驱动程序。MCXC444系列MCU的USB设备控制器是一种全速USB设备控制器。
- RTOS适配层：此USB协议栈不仅可以在不同的RTOS环境下运行，还还可以在裸板环境中运行。本应用笔记中未使用RTOS。

此工程中的源代码和USB协议栈架构间的对应关系如图5所示。

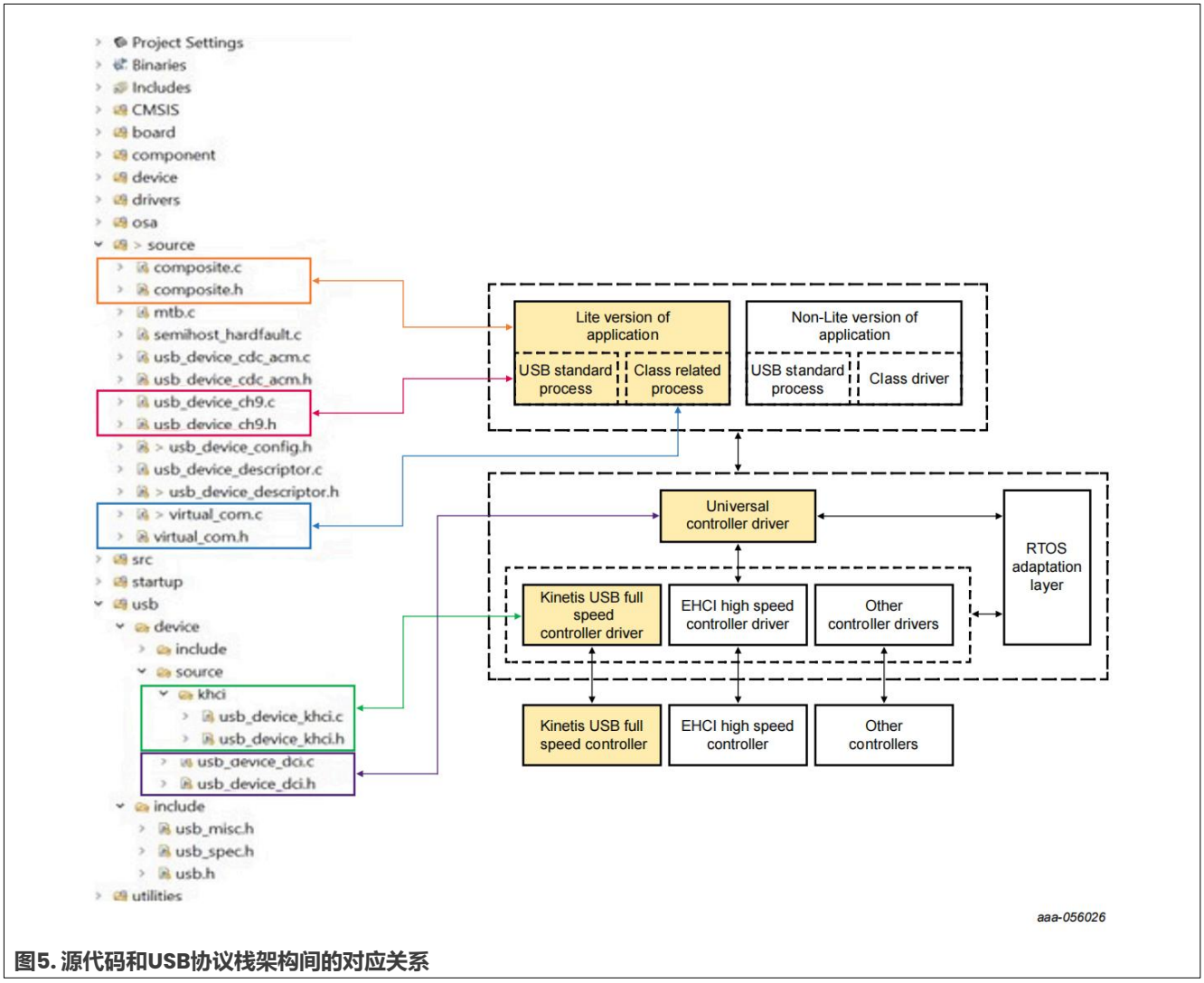

图5. 源代码和USB协议栈架构间的对应关系

设备协议栈的基本工作流程取决于回调函数和函数调用。回调函数将所有状态更改和设备协议栈的数据请求通知给应用程序。

设备协议栈中有两种类型的回调函数：

- 设备回调函数：将设备协议栈的状态通知给应用程序。
- 端点回调函数：将相应端点的数据传输结果通知给应用程序。控制端点回调函数处理所有USB标准请求和类请求。图6简要描述了回调函数的处理过程。

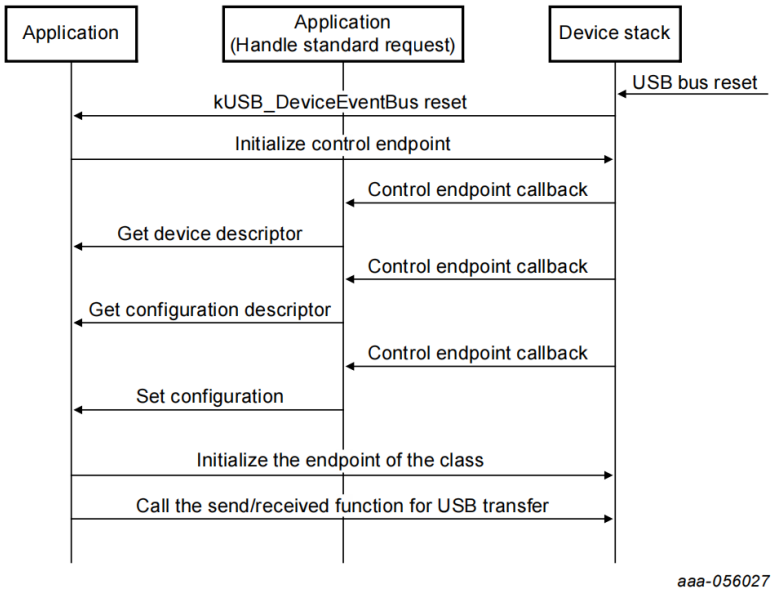

图6. 回调函数的处理过程

当USB主机识别到某个USB设备插入USB接口时，它将启动枚举进程。USB枚举的本质是USB主机获取USB设备的参数信息并配置可配置参数的过程。USB枚举进程是在USB中断服务函数中完成的。USB中断服务函数的处理流程如图7所示。

5.1 USB中断服务函数的流程图

当USB中断发生时，中断服务函数会判断该中断的类型：令牌完成中断、复位中断或端点停顿中断，然后跳转到相应的操作。

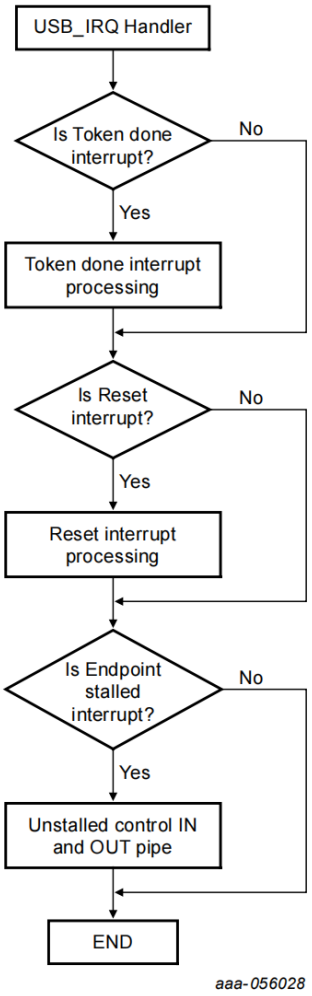

图7. USB中断服务函数的主要流程图

5.1.1 复位中断的过程

如果USB中断是一个复位中断，则程序会执行如图8所示的流程。

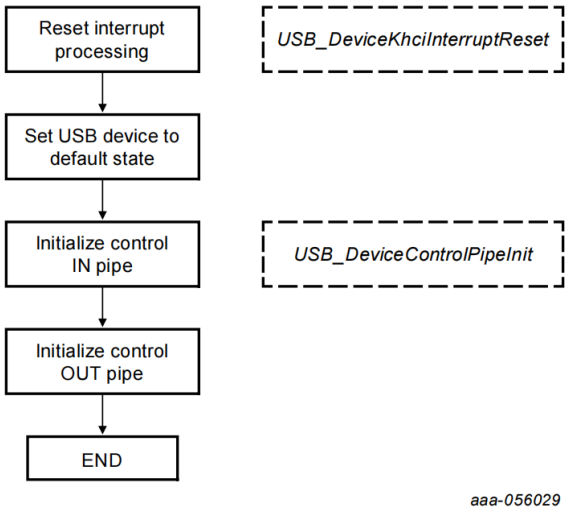

图8. USB设备复位的过程

复位中断的主要操作是将USB设备设置为默认状态，并初始化控制IN管道和OUT管道。

5.1.2 令牌完成中断的过程

如果发生了一个令牌完成中断，则程序会执行如图9所示的过程。

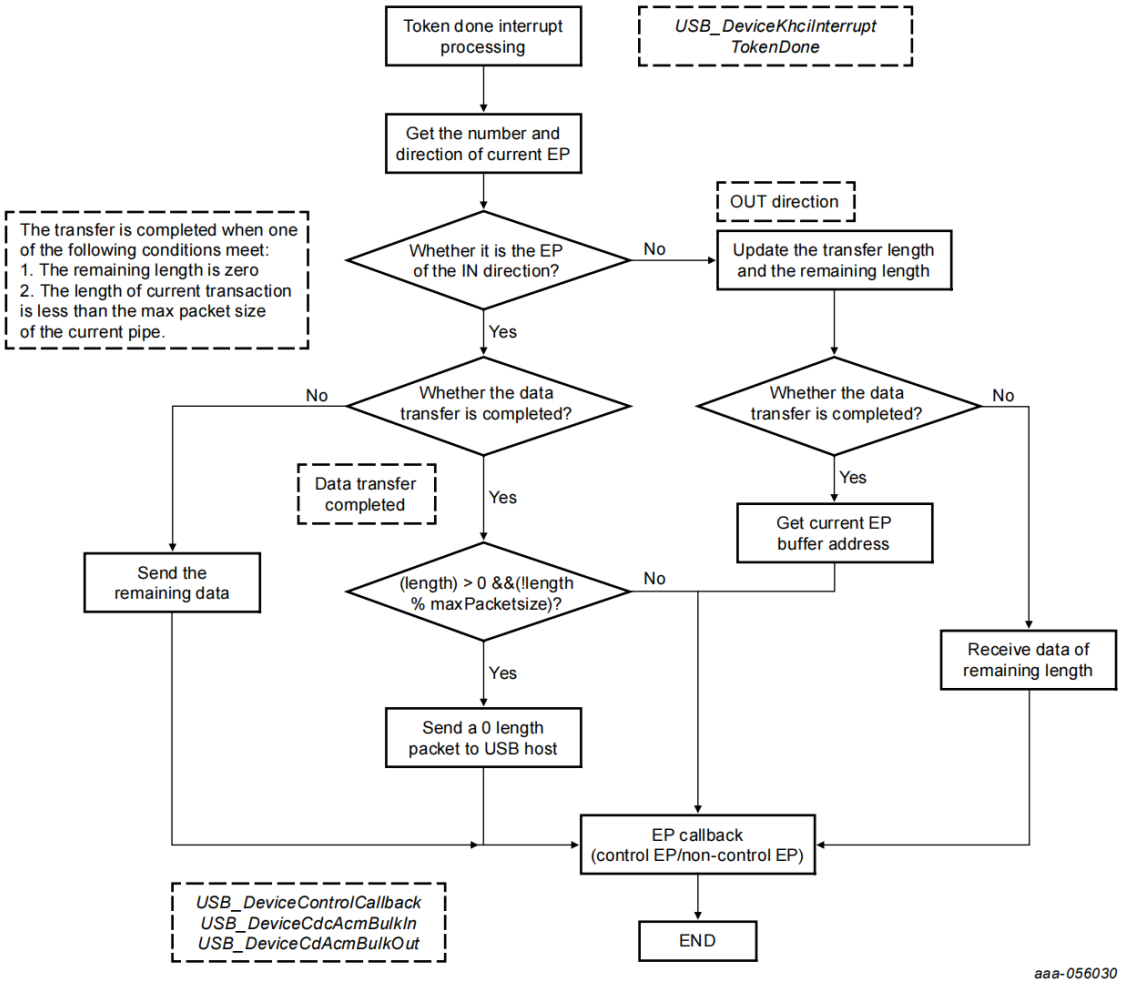

图9. 信令完成中断的过程

端点回调函数包括两种类型：控制端点回调函数和非控制端点回调函数。如果当前端点是一个控制端点，则会调用控制端点回调函数；如果当前端点是一个非控制端点，则会调用非控制端点回调函数。控制端点的回调函数实现了对标准和类请求的处理；非控制端点回调函数实现了数据传输功能，并将Bluk OUT端点接收到的数据通过Bluk IN端点传输到主机。本应用笔记中的15个VCOM的非控制端点回调函数是相同的。IN方向上的端点回调函数为`USB_DeviceCdcAcmBulkIn()`，OUT方向上的端点回调函数为`USB_DeviceCdcAcmBulkOut()`，即15个VCOM 实现的功能是相同的：将从USB主机接收到的数据传输到USB主机。

5.2 USB设备请求

标准请求和类请求的处理是在控制端点的回调函数中执行的。USB主机通过向设备发送设备请求（Device Request）来获取USB设备的信息并在USB设备上进行相关配置。设备请求由长度为8个字节的设置数据指定，其方向始终是从USB主机到USB设备。通用的标准请求如表7所示。

表7. 通用标准请求

标准请求	方向
SetAddress	OUT
SetConfiguration	OUT
GetDescriptor(设备/配置/字符串)	IN

除了标准请求外，CDC类设备还支持一些类请求。USB主机获得CDC类设备的各种描述符信息后，还会向USB设备发送一些类请求。通用的类请求如表8所列。

表8. 通用类请求

类请求	方向
SetLineCoding	OUT
SetControlLineState	OUT
GetLineCoding	IN

***注：** CDC接口中未使用中断IN端点，因此无法处理SetControlLineState类请求，请忽略此类请求。*

GetLineCoding请求是主机获取串行端口属性的请求，包括波特率、停止位、校验类型和数据位数。GetLineCoding请求的结构如表9所示。

表9. GetLineCoding请求的结构

bmRequestType	bRequestCode	wValue	wIndex	wLength	Data
10100001B	GET_LINE_CODIN G	0	接口	结构尺寸	线编码结构

线编码结构的内容如表10所示。

表10. 线编码结构

偏移	字段	大小/字节	说明
0	dwDTERate	4	数据终端速率，以每秒的位数为单位
4	bCharFormat	1	停止位 0：1停止位 1：1.5停止位 2：2停止
5	bParityType	1	奇偶校验位 0：无 3：标记
			1：奇数 4：空格 2：偶数
6	bDataBits	1	数据位（5、6、7、8或16）

SetLineCoding是一个输出类请求，对应于GetLineCoding。USB主机使用此命令来设置设备的串行端口属性，其数据结构与GetLineCoding相同。当USB主机完成这些标准请求和类请求后，枚举过程结束，USB主机将识别出多个VCOM。然后，PC上的串口调试终端就可以与MCX C444系列MCU的USB设备进行通信了。

6 关键步骤

6.1 如何扩展支持的VCOM数量

本节以SDK代码`dev_composite_cdc_vcom_cdc_vcom_lite_bm`为例，说明如何在两个VCOM的基础上扩展到三个VCOM。具体步骤如下：

1. 修改*usb_device_config.h*文件。
   - 将宏`USB_DEVICE_CONFIG_CDC_ACM`的值由**2**更改为**3**。

```
#define USB_DEVICE_CONFIG_CDC_ACM (2U) to
#define USB_DEVICE_CONFIG_CDC_ACM (3U)
```

   - 修改`USB_DEVICE_CONFIG_ENDPOINTS`宏的值。

```
#define USB_DEVICE_CONFIG_ENDPOINTS (5U) to
/* Values that meet functional requirements are ok */
#define USB_DEVICE_CONFIG_ENDPOINTS (8U)
```

2. 修改*usb_device_descriptor.h*文件。
   - 将宏`USB_INTERFACE_COUNT`的值由**4**更改为**6**。

```
#define USB_INTERFACE_COUNT (4) to
#define USB_INTERFACE_COUNT (6)
```

   - 为VCOM3增加一些宏定义。

```
#define USB_CDC_VCOM_INTERFACE_COUNT_3 (2)

#define USB_CDC_VCOM_CIC_INTERFACE_INDEX_3 (4)
#define USB_CDC_VCOM_DIC_INTERFACE_INDEX_3 (5)
/* If the CDC interface does not require an Interrupt IN endpoint, then this
 macro is 0 */ #define USB_CDC_VCOM_CIC_ENDPOINT_COUNT_3 (1)

/* If the CDC interface does not require an Interrupt IN endpoint, you do
 not need to define this macro*/
#define USB_CDC_VCOM_CIC_INTERRUPT_IN_ENDPOINT_3 (7)

#define USB_CDC_VCOM_DIC_ENDPOINT_COUNT_3 (2)
#define USB_CDC_VCOM_DIC_BULK_IN_ENDPOINT_3 (3)
#define USB_CDC_VCOM_DIC_BULK_OUT_ENDPOINT_3 (3)

#define HS_CDC_VCOM_INTERRUPT_IN_PACKET_SIZE_3 (16)
#define FS_CDC_VCOM_INTERRUPT_IN_PACKET_SIZE_3 (16)
#define HS_CDC_VCOM_INTERRUPT_IN_INTERVAL_3 (0x07)
#define FS_CDC_VCOM_INTERRUPT_IN_INTERVAL_3 (0x08)

#define HS_CDC_VCOM_BULK_IN_PACKET_SIZE_3 (512)
#define FS_CDC_VCOM_BULK_IN_PACKET_SIZE_3 (64)
#define HS_CDC_VCOM_BULK_OUT_PACKET_SIZE_3 (512)
#define FS_CDC_VCOM_BULK_OUT_PACKET_SIZE_3 (64)
```

3. 修改*usb_device_descriptor.c*文件。
   - 修改`g_interface`数组。

```
uint8_t g_interface[USB_CDC_VCOM_INTERFACE_COUNT +
 USB_CDC_VCOM_INTERFACE_COUNT_2
```

```
+ USB_CDC_VCOM_INTERFACE_COUNT_3];
```

- 修改配置描述符数组g_UsbDeviceConfigurationDescriptor：添加CDC3的接口描述符和端点描述符。相关的详细信息，请参阅附加的代码。

4. 修改*virtual_com.c*文件。
   - 扩展s_lineCoding、s_abstractState、s_countryCode和s_usbCdcAcmInfo数组。
   - 修改USB_DeviceCdcVcomSetConfigure()函数以增加CDC3中非控制端点的初始化。

   注：*在多个VCOM中，非控制端点的回调函数在本应用笔记中是相同的。如果要修改应用层的功能，则可以在此处使用不同功能的回调函数。*

6.2 如何删除CDC接口中的中断IN端点

为了使USB设备支持更多的VCOM，可以删除CDC接口中的中断IN端点。具体的实现步骤如下：

1. 修改*usb_device_config.h*文件。
   - 增加宏USB_CDC_CIC_INTERRUPT_IN_ENDPOINT_ENABLE的定义。

```
#define USB_CDC_CIC_INTERRUPT_IN_ENDPOINT_ENABLE (0U)
```

2. 修改usb_device_descriptor.h文件。
   - 修改宏USB_CDC_VCOM_CIC_ENDPOINT_COUNT的值为**0**。

```
#define USB_CDC_VCOM_CIC_ENDPOINT_COUNT ( 0 )
```

   根据宏USB_CDC_CIC_INTERRUPT_IN_ENDPOINT_ENABLE的值，屏蔽CDC接口中中断IN端点的定义。

```
#if USB_CDC_CIC_INTERRUPT_IN_ENDPOINT_ENABLE
/* No need interrupt IN endpoint for CIC interface */ #define
 USB_CDC_VCOM_CIC_INTERRUPT_IN_ENDPOINT (0)
#define USB_CDC_VCOM_CIC_INTERRUPT_IN_ENDPOINT_2 (0)
#define USB_CDC_VCOM_CIC_INTERRUPT_IN_ENDPOINT_3 (0)
#define USB_CDC_VCOM_CIC_INTERRUPT_IN_ENDPOINT_4 (0)
#define USB_CDC_VCOM_CIC_INTERRUPT_IN_ENDPOINT_5 (0)
#define USB_CDC_VCOM_CIC_INTERRUPT_IN_ENDPOINT_6 (0)
#define USB_CDC_VCOM_CIC_INTERRUPT_IN_ENDPOINT_7 (0)
#define USB_CDC_VCOM_CIC_INTERRUPT_IN_ENDPOINT_8 (0)
#define USB_CDC_VCOM_CIC_INTERRUPT_IN_ENDPOINT_9 (0)
#define USB_CDC_VCOM_CIC_INTERRUPT_IN_ENDPOINT_10 (0)
#define USB_CDC_VCOM_CIC_INTERRUPT_IN_ENDPOINT_11 (0)
#define USB_CDC_VCOM_CIC_INTERRUPT_IN_ENDPOINT_12 (0)
#define USB_CDC_VCOM_CIC_INTERRUPT_IN_ENDPOINT_13 (0)
#define USB_CDC_VCOM_CIC_INTERRUPT_IN_ENDPOINT_14 (0)
#define USB_CDC_VCOM_CIC_INTERRUPT_IN_ENDPOINT_15 (0)
#endif
```

3. 修改*usb_device_descriptor.c*文件。
   - 修改g_UsbDeviceConfigurationDescriptor配置描述符数组中的数组长度字段。

```
USB_SHORT_GET_LOW(USB_DESCRIPTOR_LENGTH_CONFIGURE +
                  (USB_IAD_DESC_SIZE + USB_DESCRIPTOR_LENGTH_INTERFACE +
 USB_DESCRIPTOR_LENGTH_CDC_HEADER_FUNC +
                  USB_DESCRIPTOR_LENGTH_CDC_CALL_MANAG +
 USB_DESCRIPTOR_LENGTH_CDC_ABSTRACT + USB_DESCRIPTOR_LENGTH_CDC_UNION_FUNC
```

```
 + 0 + USB_DESCRIPTOR_LENGTH_INTERFACE + USB_DESCRIPTOR_LENGTH_ENDPOINT +
 USB_DESCRIPTOR_LENGTH_ENDPOINT) * USB_DEVICE_CONFIG_CDC_ACM),
USB_SHORT_GET_HIGH(USB_DESCRIPTOR_LENGTH_CONFIGURE +
                   (USB_IAD_DESC_SIZE + USB_DESCRIPTOR_LENGTH_INTERFACE +
 USB_DESCRIPTOR_LENGTH_CDC_HEADER_FUNC +
                   USB_DESCRIPTOR_LENGTH_CDC_CALL_MANAG +
 USB_DESCRIPTOR_LENGTH_CDC_ABSTRACT + USB_DESCRIPTOR_LENGTH_CDC_UNION_FUNC
 + 0 + USB_DESCRIPTOR_LENGTH_INTERFACE + USB_DESCRIPTOR_LENGTH_ENDPOINT +
 USB_DESCRIPTOR_LENGTH_ENDPOINT) *
USB_DEVICE_CONFIG_CDC_ACM),
```

- 删除所有CDC接口的中断端点描述符。

```
#if USB_CDC_CIC_INTERRUPT_IN_ENDPOINT_ENABLE
/*Notification Endpoint descriptor */
        USB_DESCRIPTOR_LENGTH_ENDPOINT,
        USB_DESCRIPTOR_TYPE_ENDPOINT,
        USB_CDC_VCOM_CIC_INTERRUPT_IN_ENDPOINT |(USB_IN << 7U),
 USB_ENDPOINT_INTERRUPT,
        USB_SHORT_GET_LOW(FS_CDC_VCOM_INTERRUPT_IN_PACKET_SIZE),
        USB_SHORT_GET_HIGH(FS_CDC_VCOM_INTERRUPT_IN_PACKET_SIZE),
        FS_CDC_VCOM_INTERRUPT_IN_INTERVAL,
#endif
```

4. 修改*virtual_com.c*文件。
   - 在`USB_DeviceCdcVcomClassRequest()`函数中修改`SetControlLineState`类请求的处理。

```
     if (0 == vcomInstance->hasSentState)
                              {
                              #if USB_CDC_CIC_INTERRUPT_IN_ENDPOINT_ENABLE
                              error = USB_DeviceSendRequest(handle,
 vcomInstance->interruptEndpoint,
                              acmInfo->serialStateBuf, len);
                              if (kStatus_USB_Success != error)
                              {
                              usb_echo("kUSB_DeviceCdcEventSetControlLineState
 error!");
                              }
                              #endif
                              vcomInstance->hasSentState = 1;
                              }
```

   - 添加数组`g_CdcVcomInterruptInPackageSzie`, `g_CdcVcomInterruptInInterval`, `g_CdcVcomInterruptInEndpoint`, `g_CdcVcomDicBulkInEndpoint`, `g_CdcVcomDicBulkInEndpoint`, `g_CdcVcomDicBulkOutEndpoint`, `g_CdcVcomDicInterfaceIndex`, `g_CdcVcomCicInterfaceIndex`, `g_CdcVcomBulkInPacketSize`和`g_CdcVcomBulkOutPacketSize`。
   - ***注：*** *这些数组仅用于简化`USB_DeviceCdcVcomSetConfigure()`函数中每个端点的初始化。*
   - 修改`USB_DeviceCdcVcomSetConfigure()`函数中每个CDC类中的中断端点的初始化过程。

```
     if (USB_COMPOSITE_CONFIGURE_INDEX == configure)
     {
       for ( uint8_t i = 0; i < USB_DEVICE_CONFIG_CDC_ACM; i++)
     {
          /****** Initiailizecdc endpoint *****/
         g_deviceComposite-> cdcVcom [i]. attach = 1;
#if USB_CDC_CIC_INTERRUPT_IN_ENDPOINT_ENABLE
```

```
        /* Initiailizeendpoint for interrupt pipe */
        epCallback.callbackFn = USB_DeviceCdcAcmInterruptIn;
        epCallback.callbackParam = (void

        *)&g_deviceComposite->cdcVcom[i].communicationInterfaceNumber;
        epInitStruct.zlt = 0;

        epInitStruct.transferType = USB_ENDPOINT_INTERRUPT;
        epInitStruct.endpointAddress =
        g_CdcVcomCicInterruptInEndpoint[i] | (USB_IN <<

        USB_DESCRIPTOR_ENDPOINT_ADDRESS_DIRECTION_SHIFT);
        epInitStruct.maxPacketSize = FS_CDC_VCOM_INTERRUPT_IN_PACKET_SIZE;
        epInitStruct.interval = FS_CDC_VCOM_INTERRUPT_IN_INTERVAL;
       g_deviceComposite->cdcVcom[i].interruptEndpoint =

        g_CdcVcomCicInterruptInEndpoint[i];
        g_deviceComposite->cdcVcom[i].interruptEndpointMaxPacketSize =

        epInitStruct.maxPacketSize;
        g_deviceComposite->cdcVcom[i].communicationInterfaceNumber =
        g_CdcVcomCicInterfaceIndex[i];
        USB_DeviceInitEndpoint(handle, &epInitStruct, &epCallback);
#else
        g_deviceComposite-> cdcVcom [i]. communicationInterfaceNumber =
        g_CdcVcomCicInterfaceIndex[i];
#endif
```

***注：**请勿删除以下内容：`g_deviceComposite->cdcVcom[i].communication InterfaceNumber = USB_CDC_VCOM_CIC_INTERFACE_INDEX`。如需查看完整代码，请参阅AN12284SW。*

6.3 代码优化

第6.1节和第6.2节中的步骤有些复杂。用户需要对USB协议栈有一定了解。如果出现了步骤错误，则可能导致功能失败。对于每个附加的VCOM，用户需要重复第6.1节中的步骤。为了使VCOM功能的操作过程更加方便，在第6.1节和第6.2节的代码基础上进行了一些优化，方便用户配置VCOM的数量，并通过宏定义决定是否使用CIC接口的中断IN端点。

6.3.1 使用循环替代列表

例如，*virtual_com.c*文件中的`USB_DeviceCdcVcomConfigureEndpointStatus ()`函数：

```
usb_status_t USB_DeviceCdcVcomConfigureEndpointStatus(usb_device_handle handle,
 uint8_t ep, uint8_t status)
{
    usb_status_t error = kStatus_USB_Error;
    if (status)
    {
        if ((USB_CDC_VCOM_DIC_BULK_IN_ENDPOINT == (ep &
 USB_ENDPOINT_NUMBER_MASK)) &&
            (ep & USB_DESCRIPTOR_ENDPOINT_ADDRESS_DIRECTION_MASK))
    {
        error = USB_DeviceStallEndpoint(handle, ep);
    }
```

```
    else if ((USB_CDC_VCOM_DIC_BULK_OUT_ENDPOINT == (ep &
USB_ENDPOINT_NUMBER_MASK)) &&
             (!(ep & USB_DESCRIPTOR_ENDPOINT_ADDRESS_DIRECTION_MASK)))
    {
        error = USB_DeviceStallEndpoint(handle, ep);
    }
    else if ((USB_CDC_VCOM_DIC_BULK_IN_ENDPOINT_2 == (ep &
USB_ENDPOINT_NUMBER_MASK)) &&
             (ep & USB_DESCRIPTOR_ENDPOINT_ADDRESS_DIRECTION_MASK))
   {
         error = USB_DeviceStallEndpoint(handle, ep);
    }
    else if ((USB_CDC_VCOM_DIC_BULK_OUT_ENDPOINT_2 == (ep &
USB_ENDPOINT_NUMBER_MASK)) &&
    (           !(ep & USB_DESCRIPTOR_ENDPOINT_ADDRESS_DIRECTION_MASK)))
    {
        error = USB_DeviceStallEndpoint(handle, ep);
    }
    else
    {
    }
    }
    else
    {
        if ((USB_CDC_VCOM_DIC_BULK_IN_ENDPOINT == (ep &
USB_ENDPOINT_NUMBER_MASK)) &&
            (ep & USB_DESCRIPTOR_ENDPOINT_ADDRESS_DIRECTION_MASK))
    {
        error = USB_DeviceUnstallEndpoint(handle, ep);
     }
     else if ((USB_CDC_VCOM_DIC_BULK_OUT_ENDPOINT == (ep &
USB_ENDPOINT_NUMBER_MASK)) &&
              (!(ep & USB_DESCRIPTOR_ENDPOINT_ADDRESS_DIRECTION_MASK)))
     {
          error = USB_DeviceUnstallEndpoint(handle, ep);
      }
      else if ((USB_CDC_VCOM_DIC_BULK_IN_ENDPOINT_2 == (ep &
USB_ENDPOINT_NUMBER_MASK)) &&
               (ep & USB_DESCRIPTOR_ENDPOINT_ADDRESS_DIRECTION_MASK))
      {
          error = USB_DeviceUnstallEndpoint(handle, ep);
      }
      else if ((USB_CDC_VCOM_DIC_BULK_OUT_ENDPOINT_2 == (ep &
USB_ENDPOINT_NUMBER_MASK)) &&
               (!(ep & USB_DESCRIPTOR_ENDPOINT_ADDRESS_DIRECTION_MASK)))
      {
          error = USB_DeviceUnstallEndpoint(handle, ep);
      }
      else
      {
      }
      }
          return error;
      }
```

```
usb_status_t USB_DeviceCdcVcomConfigureEndpointStatus(usb_device_handle handle,
 uint8_t ep, uint8_t status)
{
    usb_status_t error = kStatus_USB_Error;
```

```
    uint8_t i;
    if (status)
    {
        for(i = 0; i < USB_DEVICE_CONFIG_CDC_ACM; i++)
        {
            if ((g_CdcVcomDicBulkInEndpoint[i] == (ep &
 USB_ENDPOINT_NUMBER_MASK)) &&
                (ep & USB_DESCRIPTOR_ENDPOINT_ADDRESS_DIRECTION_MASK))
            {
                error = USB_DeviceStallEndpoint(handle, ep);
            }
            else if ((g_CdcVcomDicBulkOutEndpoint[i] == (ep &
 USB_ENDPOINT_NUMBER_MASK)) &&
                     ep & USB_DESCRIPTOR_ENDPOINT_ADDRESS_DIRECTION_MASK))
            {
                error = USB_DeviceStallEndpoint(handle, ep);
            }
        }
    }
else
{
    for(i = 0; i < USB_DEVICE_CONFIG_CDC_ACM; i++)
    {
            if ((g_CdcVcomDicBulkInEndpoint[i] == (ep &
 USB_ENDPOINT_NUMBER_MASK)) &&
                 (ep & USB_DESCRIPTOR_ENDPOINT_ADDRESS_DIRECTION_MASK))
            {
                 error = USB_DeviceUnstallEndpoint(handle, ep);
             }
             else if ((g_CdcVcomDicBulkOutEndpoint[i] == (ep &
 USB_ENDPOINT_NUMBER_MASK)) &&
                         (ep & USB_DESCRIPTOR_ENDPOINT_ADDRESS_DIRECTION_MASK))
    {
error = USB_DeviceUnstallEndpoint(handle, ep);
}
}
}
return error;
}
```

实际上，有很多代码都可以用循环替代，这是应用程序自动合成类的关键一步。其他类似的代码未在此处列出，有关更多详细信息，请参阅AN14322SW。

6.3.2 删除非必需的设置

可以合并以下设置，通常，用户根本不需要调整这些参数。即使在某些特殊情况下一些用户需要调整参数，运行时初始化也支持修改参数。

```
/* Packet size. */
#define HS_CDC_VCOM_INTERRUPT_IN_PACKET_SIZE (16)
#define FS_CDC_VCOM_INTERRUPT_IN_PACKET_SIZE (16)
#define HS_CDC_VCOM_INTERRUPT_IN_INTERVAL (0x07) /* 2^(7-1) = 8ms */
#define FS_CDC_VCOM_INTERRUPT_IN_INTERVAL (0x08)
/* Packet size. */
#define HS_CDC_VCOM_INTERRUPT_IN_PACKET_SIZE_2 (16)
#define FS_CDC_VCOM_INTERRUPT_IN_PACKET_SIZE_2 (16)
#define HS_CDC_VCOM_INTERRUPT_IN_INTERVAL_2 (0x07) /* 2^(7-1) = 8ms */
#define FS_CDC_VCOM_INTERRUPT_IN_INTERVAL_2 (0x08)
#define HS_CDC_VCOM_INTERRUPT_IN_PACKET_SIZE_3 (16)
```

```
#define FS_CDC_VCOM_INTERRUPT_IN_PACKET_SIZE_3 (16)
#define HS_CDC_VCOM_INTERRUPT_IN_INTERVAL_3 (0x07) /* 2^(7-1) = 8ms */
#define FS_CDC_VCOM_INTERRUPT_IN_INTERVAL_3 (0x08)
#define HS_CDC_VCOM_INTERRUPT_IN_PACKET_SIZE_4 (16)
#define FS_CDC_VCOM_INTERRUPT_IN_PACKET_SIZE_4 (16)
#define HS_CDC_VCOM_INTERRUPT_IN_INTERVAL_4 (0x07) /* 2^(7-1) = 8ms */
#define FS_CDC_VCOM_INTERRUPT_IN_INTERVAL_4 (0x08)
#define HS_CDC_VCOM_INTERRUPT_IN_PACKET_SIZE_5 (16)
#define FS_CDC_VCOM_INTERRUPT_IN_PACKET_SIZE_5 (16)
#define HS_CDC_VCOM_INTERRUPT_IN_INTERVAL_5 (0x07) /* 2^(7-1) = 8ms */
#define FS_CDC_VCOM_INTERRUPT_IN_INTERVAL_5 (0x08)
```

合并后，以上代码变为：

```
/* Packet size. */
#define HS_CDC_VCOM_INTERRUPT_IN_PACKET_SIZE (16)
#define FS_CDC_VCOM_INTERRUPT_IN_PACKET_SIZE (16)
#define HS_CDC_VCOM_INTERRUPT_IN_INTERVAL (0x07) /* 2^(7-1) = 8ms */
#define FS_CDC_VCOM_INTERRUPT_IN_INTERVAL (0x08)
```

其他相似的代码未在此处列出。有关更多详细信息，请参考AN14322SW。

6.3.3 使用宏以避免手动修改代码

```
#define USB_INTERFACE_COUNT (30)

#define USB_INTERFACE_COUNT (2*USB_DEVICE_CONFIG_CDC_ACM)

#define USB_DEVICE_CONFIG_ENDPOINTS (16U)

#define USB_DEVICE_CONFIG_ENDPOINTS (1+2*USB_DEVICE_CONFIG_CDC_ACM)
```

其他类似的代码未在此处列出。有关更多详细信息，请参考AN14322SW。

6.3.4 使用运行时初始化代替静态初始化

6.3.4.1 动态分配接口号

```
void USB_CdcVcomInterfaceIndexInit(void)
{
    uint8_t i;
    for(i = 0; i < USB_DEVICE_CONFIG_CDC_ACM; i++)
    {
        g_CdcVcomCicInterfaceIndex[i] = 0 + i*2;
        g_CdcVcomDicInterfaceIndex[i] = 1 + i*2;
    }
}
```

6.3.4.2 动态分配端点号

```
void USB_CdcVcomEndpointInit(void)
{
    uint8_t i;
    for(i = 0; i < USB_DEVICE_CONFIG_CDC_ACM; i++)
    {
```

```
        #if USB_CDC_CIC_INTERRUPT_IN_ENDPOINT_ENABLE

            g_CdcVcomCicInterruptInEndpoint[i] = 1 + i*2;
            g_CdcVcomDicBulkInEndpoint[i] = 2 + i*2;
            g_CdcVcomDicBulkOutEndpoint[i] = 2 + i*2;
        #else
           g_CdcVcomDicBulkInEndpoint[i] = 1 + i*1;
             g_CdcVcomDicBulkOutEndpoint[i] = 1 + i*1;

        #endif
    }
}
```

6.3.4.3 配置描述符的运行时初始化

```
void USB_DescriptorInit(void)
{
    uint8_t *p = NULL;
    uint8_t i;
    /* copy configuration descriptor */
    memcpy(g_UsbDeviceConfigurationDescriptor + 0,
 g_CdcConfigurationDescriptorTemplate, USB_DESCRIPTOR_LENGTH_CONFIGURE);

    /* copy cdc iap, interface, endpoint descriptor */
    for(i = 0; i < USB_DEVICE_CONFIG_CDC_ACM ; i++)
    {
        memcpy(g_UsbDeviceConfigurationDescriptor +
 USB_DESCRIPTOR_LENGTH_CONFIGURE + USB_CDC_DESCRIPTOR_INSTANCE_LENGTH * i,
        g_CdcDescriptorTemplate,
        USB_CDC_DESCRIPTOR_INSTANCE_LENGTH
        );
    }

    /* update interface and endpoint descirptor */
    for(i = 0; i < USB_DEVICE_CONFIG_CDC_ACM ; i++)
    {
        p = g_UsbDeviceConfigurationDescriptor + USB_DESCRIPTOR_LENGTH_CONFIGURE
 + USB_CDC_DESCRIPTOR_INSTANCE_LENGTH * i;
        /* The first interface number associated with this function */
        p[2] = g_CdcVcomCicInterfaceIndex[i];
        p += USB_IAD_DESC_SIZE;
        /* CIC interface index */
        p[2] = g_CdcVcomCicInterfaceIndex[i];
        p += USB_DESCRIPTOR_LENGTH_INTERFACE;
        p += USB_DESCRIPTOR_LENGTH_CDC_HEADER_FUNC;
        p += USB_DESCRIPTOR_LENGTH_CDC_CALL_MANAG;
        p += USB_DESCRIPTOR_LENGTH_CDC_ABSTRACT;
        p[3] = g_CdcVcomCicInterfaceIndex[i];
        p[4] = g_CdcVcomDicInterfaceIndex[i];
        p += USB_DESCRIPTOR_LENGTH_CDC_UNION_FUNC;

#if USB_CDC_CIC_INTERRUPT_IN_ENDPOINT_ENABLE
        p[2] = g_CdcVcomCicInterruptInEndpoint[i] | (USB_IN <<7);
        p += USB_DESCRIPTOR_LENGTH_CDC_CIC_INTERRUPT_ENDPOINT;
#endif
        /* data interface descriptor */
        p[2] = g_CdcVcomDicInterfaceIndex[i];

        /* bulk in endpoint descriptor */
```

```
            p += USB_DESCRIPTOR_LENGTH_INTERFACE;
            p[2] = g_CdcVcomDicBulkInEndpoint[i] | (USB_IN <<7);

            /* bulk out endpoint descriptor */
            p += USB_DESCRIPTOR_LENGTH_ENDPOINT;
            p[2] = g_CdcVcomDicBulkOutEndpoint[i] | (USB_OUT <<7);
        }
    }
```

手动添加CDC类描述符也是一个容易出错的步骤。当使用15个VCOM描述符时，数组长度非常长。这种动态方法使配置描述符变得很容易。完成上述四项优化后，用户可以通过宏USB_DEVICE_CONFIG_CDC_ACM设置VCOM的数量，还可以使用宏USB_CDC_CIC_INTERRUPT_IN_ENDPOINT_ENABLE来决定是否在CIC接口中使用中断端点。

7 功能测试

本应用笔记的附件提供了两个适用于MCX C444的MCUXpresso版本的工程。第一个工程是未通过第6.3节中的步骤进行优化的工程，第二个工程是优化后的工程。同样，还为MCX C444提供了两个类似的工程。使用MCX C444的优化工程进行功能测试：

将宏USB_DEVICE_CONFIG_CDC_ACM和宏USB_CDC_CIC_INTERRUPT_IN_ENDPOINT_ENDABLE设置为不同的值，然后编译工程并将程序下载到开发板上，结果如图10所示。

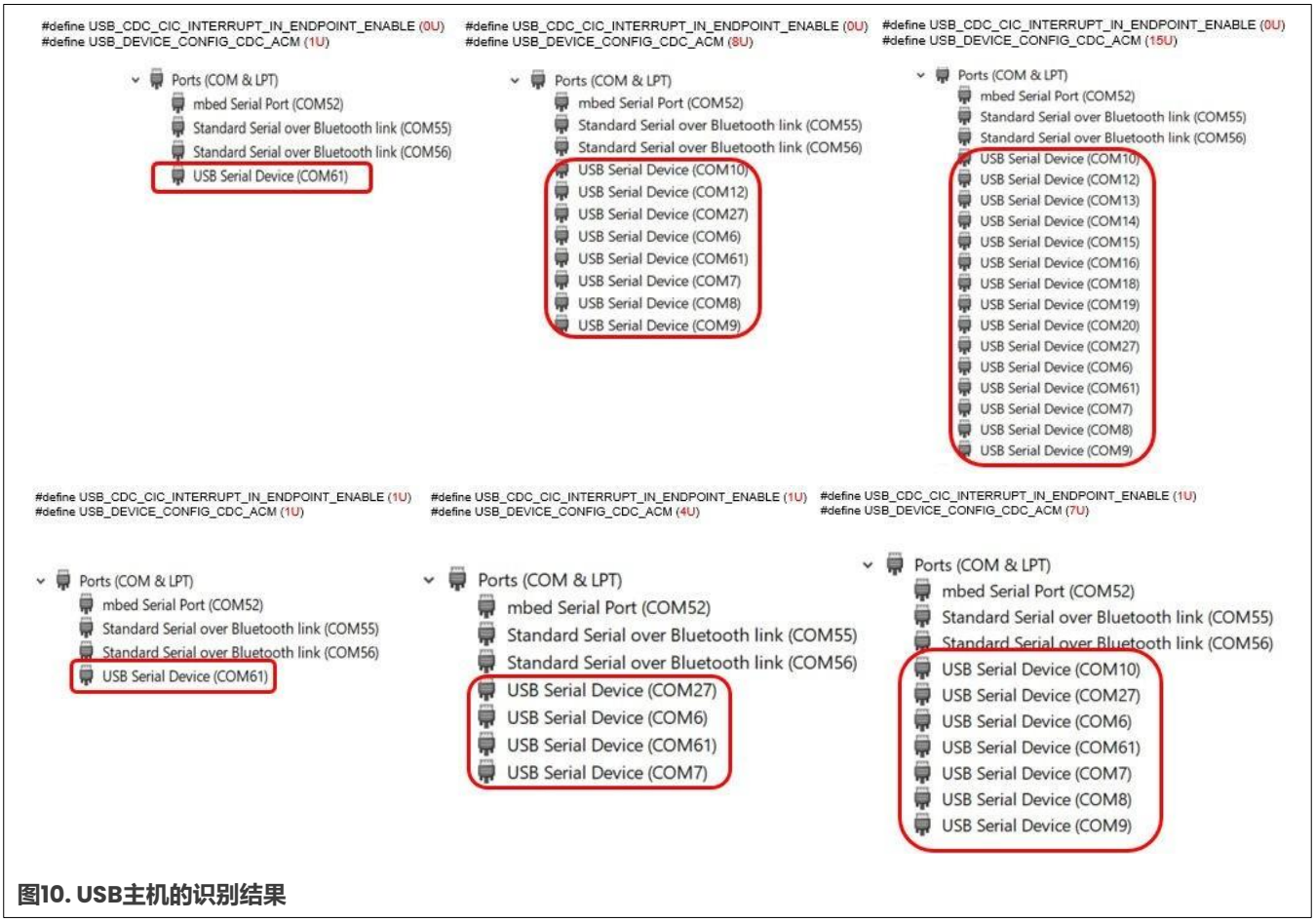

图10. USB主机的识别结果

当不使用CIC接口中的中断IN端点时，USB设备最多可以支持15个VCOM。当使用中断IN端点时，USB设备最多可支持7个VCOM。

注：*如果Windows系统自带CDC驱动程序，则用户无需手动安装驱动程序；如果PC上未安装CDC驱动程序，则用户需要手动安装该驱动程序。有关驱动程序的安装，请参阅随附的readme.pdf文档。*

8 结论

本应用笔记基于SDK代码，在MCX C444系列MCU上实现了一个FS USB设备到多个虚拟串口的功能。最多可以支持15个虚拟串口，且每个虚拟串口都实现了将接收到的数据返回给主机的功能。

9 参考资料

1. 《LPC54018和LPC5500上的USB转虚拟串口》（文档AN12458）
2. 《MCXC444参考手册》
3. 《USB 2.0规范》
4. 《MCU信号和协议中的USB》
5. 《微控制器USB的技术及应用入门》

10 关于本文中源代码的说明

本文中所示的示例代码具有以下版权和BSD-3-Clause许可：

2024年恩智浦版权所有；在满足以下条件的情况下，可以源代码和二进制文件的形式重新分发和使用本源代码（无论是否经过修改）：

1. 重新分发源代码必须保留上述版权声明、这些条件和以下免责声明。
2. 以二进制文件形式重新分发时，必须在文档和/或随分发提供的其他材料中复制上述版权声明、这些条件和以下免责声明。
3. 未经事先书面许可，不得使用版权所有者的姓名或参与者的姓名为本软件的衍生产品进行背书或推广。

本软件由版权所有者和参与者“按原样”提供，不承担任何明示或暗示的担保责任，包括但不限于对适销性和特定用途适用性的暗示保证。在任何情况下，无论因何种原因或根据何种法律条例，版权所有者或参与者均不对因使用本软件而导致的任何直接、间接、偶然、特殊、惩戒性或后果性损害（包括但不限于采购替代商品或服务；使用损失、数据损失或利润损失或业务中断）承担责任，无论是因合同、严格责任还是侵权行为（包括疏忽或其他原因）造成的，即使事先被告知有此类损害的可能性也不例外。

11 修订历史

表11汇总了本文档的修订情况。

表11. 修订历史

文档ID	发布日期	说明
AN14322 v.1	2024年7月15日	首次公开发布

Legal information

Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

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.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Limiting values — Stress above one or more limiting values (as defined in the Absolute Maximum Ratings System of IEC 60134) will cause permanent damage to the device. Limiting values are stress ratings only and (proper) operation of the device at these or any other conditions above those given in the Recommended operating conditions section (if present) or the Characteristics sections of this document is not warranted. Constant or repeated exposure to limiting values will permanently and irreversibly affect the quality and reliability of the device.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at https://www.nxp.com.cn/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

No offer to sell or license — Nothing in this document may be interpreted or construed as an offer to sell products that is open for acceptance or the grant, conveyance or implication of any license under any copyrights, patents or other industrial or intellectual property rights.

Quick reference data — The Quick reference data is an extract of the product data given in the Limiting values and Characteristics sections of this document, and as such is not complete, exhaustive or legally binding.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this document expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. 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.

NXP B.V. — NXP B.V. is not an operating company and it does not distribute or sell products.

Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

Kinetis — is a trademark of NXP B.V.

MCX — is a trademark of NXP B.V.

Microsoft, Azure, and ThreadX — are trademarks of the Microsoft group of companies.

目录