



## TXW80X USB 方案开发指南




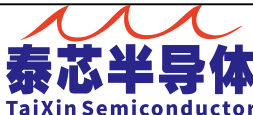
### 注意

由于产品版本升级或者其他原因，本文档会不定期更新。除非另行约定，本文档仅作为使用指导，不做任何担保。

珠海泰芯半导体有限公司  
Zhuhai Taixin Semiconductor Co., Ltd

珠海市高新区港湾一号科创园港 11 栋 3 楼

保密等级	A	TXW80X USB 方案开发指南	文件编号	TX-0000																								
发行日期	2022-05-25		文件版本	V1.0																								
<div>修订记录</div> <table><tr><td>日期</td><td>版本</td><td>描 述</td><td>修订人</td></tr><tr><td>2022-05-25</td><td>V1.0</td><td>初始版本</td><td>TX</td></tr><tr><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td><td></td></tr></table>					日期	版本	描 述	修订人	2022-05-25	V1.0	初始版本	TX																
日期	版本	描 述	修订人																									
2022-05-25	V1.0	初始版本	TX																									
		珠海泰芯半导体有限公司 Zhuhai Taixin Semiconductor Co.,Ltd	珠海市高新区港湾一号科创园港 11 栋 3 楼																									
版权所有 侵权必究 Copyright © 2022 by Tai Xin All rights reserved																												

保密等级	A	TXW80X USB 方案开发指南	文件编号	TX-0000
发行日期	2022-05-25		文件版本	V1.0
<div>目录</div> <div>TXW80X USB 方案开发指南..... 1</div> <div>1. 概述..... 1</div> <div>2. USB Common..... 1</div> <div>3. USB HOST..... 3</div> <div>3.1. USB Host 相关配置..... 3</div> <div>3.2. USB Host 工作流程..... 3</div> <div>4. USB Device..... 4</div> <div>4.1. USB Device 相关配置..... 4</div> <div>4.2. USB Device 工作流程..... 7</div>				
<div><div>泰芯半导体</div><div>TaiXin Semiconductor</div></div>		<div>珠海泰芯半导体有限公司</div> <div>Zhuhai Taixin Semiconductor Co.,Ltd</div>	<div>珠海市高新区港湾一号科技园港 11 栋 3 楼</div>	
<div>版权所有 侵权必究</div> <div>Copyright © 2022 by Tai Xin All rights reserved</div>				

# 1. 概述

本文为使用方案软件设计开发人员而写，目的帮助您快速入门 USB 方案开发。

本文档主要适用于以下工程师：

- 技术支持工程师
- 方案软件开发工程师

本文档适用的产品范围：

型号	封装	包装
TXW806		
TXW803		
TXW802		
TXW801		

# 2. USB Common

TXW80X USB 支持 DEVICE 和 HOST 模式，支持 USB2.0 HighSpeed 传输，共有 EP0/1/2 三个收发端点，除去 EP0 枚举外还有 EP1/2 收发四个端点。

USB device 硬件抽象层请查看 usb\_device.h，此抽象包含了 USB HOST 和 DEVICE。

```
69: struct usb_device {
70:     ... struct dev_obj dev;
71:     ... uint32 obj;
72:     ... int32 (*open)(struct usb_device *p_usb_d, struct usb_device_cfg *p_usbdev_cfg);
73:     ... int32 (*close)(struct usb_device *p_usb_d);
74:     ... int32 (*read)(struct usb_device *p_usb_d, int8 ep, int8 *buff, uint32 len, int8 sync);
75:     ... int32 (*write)(struct usb_device *p_usb_d, int8 ep, int8 *buff, uint32 len, int8 sync);
76:     ... int32 (*ioctl)(struct usb_device *p_usb_d, uint32 cmd, uint32 param1, uint32 param2);
77:     ... int32 (*request_irq)(struct usb_device *p_usb_d, usbdev_irq_hdl irqhdl, uint32 data);
78: };
79:
80: int32 usb_device_open(struct usb_device *p_usb_d, struct usb_device_cfg *p_usbdev_cfg);
81:
82: int32 usb_device_close(struct usb_device *p_usb_d);
83:
84: int32 usb_device_write(struct usb_device *p_usb_d, int8 ep, int8 *buff, uint32 len, int8 sync);
85:
86: int32 usb_device_read(struct usb_device *p_usb_d, int8 ep, int8 *buff, uint32 len, int8 sync);
87:
88: int32 usb_device_ioctl(struct usb_device *p_usb_d, uint32 cmd, uint32 param1, uint32 param2);
89:
90: int32 usb_device_request_irq(struct usb_device *p_usb_d, usbdev_irq_hdl handle, uint32 data);
91:
```

使用步骤如下：

1、和其他外设一样，使用前在 device.c 中先将设备加入系统设备链表

```
351:
352: #if USB_EN
353: #if USB_HOST_EN
354: .... husb20_host_attach(HG_USBDEV_DEVID, &usb20_dev);
355: #else
356: .... husb20_dev_attach(HG_USBDEV_DEVID, &usb20_dev);
357: #endif
358: #endif
359:
```

2、获取设备句柄（dev\_get）、打开设备（usb\_device\_open）、配置设备（usb\_device\_ioctl）、中断请求（usb\_device\_request\_irq）

```
196: .... usb_test.dev = (struct usb_device *) dev_get(HG_USBDEV_DEVID);
197: .... if (usb_test.dev) {
198: ....     usb_host_task_init(usb_test.dev);
199: ....     if (!usb_device_open(usb_test.dev, NULL)) {
200: ....         usb_host_uvc_ioctl(usb_test.dev, USB_HOST_IO_CMD_INSERT_HUB, &msgbuf, 0);
201: ....         printf("%s:%d\n", __FUNCTION__, __LINE__);
202: ....         uint32 usb_host_bus_irq(uint32 irq, uint32 param1, uint32 param2, uint32 param3);
203: ....         usb_device_request_irq(usb_test.dev, usb_host_bus_irq, (int32)usb_test.dev);
204: ....     }
205: }
```

详细说明：

1、usb\_device\_ioctl 使用说明

```
24: enum usb_dev_io_cmd {
25: .... USB_DEV_IO_CMD_AUTO_TX_NULL_PKT_ENABLE,
26: .... USB_DEV_IO_CMD_AUTO_TX_NULL_PKT_DISABLE,
27: .... USB_DEV_IO_CMD_REMOTE_WAKEUP,
28:
29: .... /* this function need call after attach & before open
30: .... * msg[1:0]::vid
31: .... * msg[3:2]::pid
32: .... */
33: .... USB_DEV_IO_CMD_SET_ID,
34: };
35:
```

【USB\_DEV\_IO\_CMD\_AUTO\_TX\_NULL\_PKT\_ENABLE】：

【USB\_DEV\_IO\_CMD\_AUTO\_TX\_NULL\_PKT\_DISABLE】：USB 自动发送空包：当发送包长和 MaxPacketSize 对齐时，自动补发空包作为短包结束开关。

【USB\_DEV\_IO\_CMD\_REMOTE\_WAKEUP】：USB Device 远程唤醒 Host

【USB\_DEV\_IO\_CMD\_SET\_ID】：USB Device VID/PID 设置，此动作需要在 usb\_device\_open 后马上执行。

2、usb\_device\_request\_irq 使用说明

目前支持的 usb irq 如下：

```

9:
10: enum usb_dev_irqs {
11:     ... USB_DEV_RESET_IRQ,
12:     ... USB_DEV_SUSPEND_IRQ,
13:     ... USB_DEV_RESUME_IRQ,
14:     ... USB_DEV_SOF_IRQ,
15:     ... USB_DEV_CTL_IRQ,
16:     ... USB_EP_RX_IRQ,
17:     ... USB_EP_TX_IRQ,
18:     ... USB_CONNECT,
19:     ... USB_DISCONNECT,
20:     ... USB_BABBLE,
21:     ... USB_XACT_ERR,
22: };
23:

```

【USB\_DEV\_RESET\_IRQ】：usb device 收到总线 reset 信号

【USB\_DEV\_SUSPEND\_IRQ】：usb device 收到总线 suspend 信号

【USB\_DEV\_RESUME\_IRQ】：usb device 收到总线 resume 信号

【USB\_DEV\_SOF\_IRQ】：usb device 收到总线 sof 信号

【USB\_DEV\_CTL\_IRQ】：usb 端点 0 收发中断

【USB\_EP\_RX\_IRQ】：usb 端点收到数据中断

【USB\_EP\_TX\_IRQ】：usb 端点发送数据完成中断

【USB\_CONNECT】：usb host 发现设备连接上来中断

【USB\_DISCONNECT】：usb host 发现设备连接断开中断

【USB\_BABBLE】：usb host 遇到 babble

【USB\_XACT\_ERR】：usb host 遇到 xact error

## 3. USB HOST

### 3.1. USB Host 相关配置

方案配置看 project\_config.h 配置：USB\_HOST\_EN 和 USB\_EN 均使能。

### 3.2. USB Host 工作流程

USB Host 软件配置和工作流程如下。

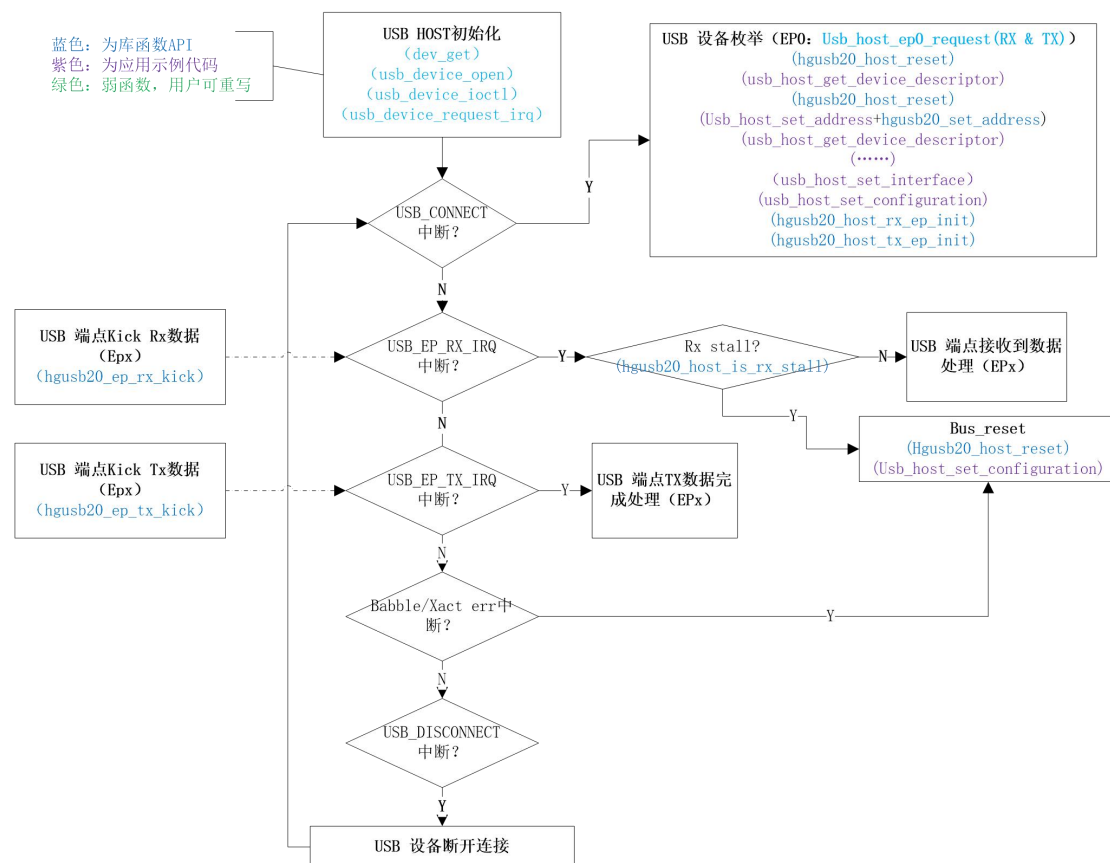


图3. 2. 1 USB Host工作流程图

## 4. USB Device

### 4.1. USB Device 相关配置

方案配置看 project\_config.h 配置：USB\_EN 使能。

Device 配置已经抽象到了一个结构体：struct usb\_device\_cfg，此结构体在 usb\_device\_open（）的时候传入，USB 后台枚举时会提取相应信息做枚举，结构体定义如下：

```

44: struct usb_device_cfg {
45:     ....uint16.....vid;
46:     ....uint16.....pid;
47:     ....uint8.....speed;
48:     ....uint8.....*p_device_descriptor;
49:     ....uint8.....*p_config_descriptor_head;
50:     ....void.....*p_config_desc;
51:     ....uint16.....config_desc_len;
52:     ....uint8.....interface_num;
53:
54:     ....uint8.....*p_language_id;
55:     ....uint16.....language_id_len;
56:     ....uint8.....*p_str_manufacturer;
57:     ....uint16.....str_manufacturer_len;
58:     ....uint8.....*p_str_product;
59:     ....uint16.....str_product_len;
60:     ....uint8.....*p_str_serial_number;
61:     ....uint16.....str_serial_number_len;
62:
63:     ....uint8.....ep_nums;
64:     ....struct usb_device_ep_cfg.....ep_cfg[8];...../*must same with config_desc*/
65: }«end usb_device_cfg»;
66:

```

【vid】：厂商 ID

【pid】：产品 ID

【speed】：暂未使用

【p\_device\_descriptor】：设备描述符

【p\_config\_descriptor\_head】：配置描述符通用头部信息

【p\_config\_desc】：配置描述符

【config\_desc\_len】：配置描述符长度

【interface\_num】：接口数量

【p\_language\_id】：语言 ID 描述符

【language\_id\_len】：语言 ID 描述符长度

【p\_str\_manufacturer】：厂商描述符

【str\_manufacturer\_len】：厂商描述符长度

【p\_str\_product】：产品描述符

【str\_product\_len】：产品描述符长度

【p\_str\_serial\_number】：序列号描述符（如果没有可以为空）

【str\_serial\_number\_len】：序列号描述符长度

【ep\_nums】：端点数量（EP0 不计入）

【ep\_cfg】：端点配置：端点号、传输方向、传输类型、最大包大小

下面是自定义 USB 设备类和 USB Audio 类的配置示例：



```

153:
154: static const struct usb_device_cfg usb_dev_wifi_cfg = {
155:     .vid = USB_VID,
156:     .pid = USB_PID,
157:     .speed = USB_SPEED_HIGH,
158:     .p_device_descriptor = (uint8_t*)tbl_usb_wifi_device_descriptor,
159:     .p_config_descriptor_head = (uint8_t*)tbl_usb_wifi_config_all_descriptor,
160:
161:     .p_config_desc = (uint8_t*)tbl_usb_wifi_config_all_descriptor_wifi,
162:     .config_desc_len = sizeof(tbl_usb_wifi_config_all_descriptor_wifi),
163:     .interface_num = 1,
164:
165:     .p_language_id = (uint8_t*)tbl_usb_wifi_language_id,
166:     .language_id_len = sizeof(tbl_usb_wifi_language_id),
167:     .p_str_manufacturer = (uint8_t*)tbl_usb_wifi_str_manufacturer,
168:     .str_manufacturer_len = sizeof(tbl_usb_wifi_str_manufacturer),
169:     .p_str_product = (uint8_t*)tbl_usb_wifi_str_product,
170:     .str_product_len = sizeof(tbl_usb_wifi_str_product),
171:     .p_str_serial_number = (uint8_t*)tbl_usb_wifi_str_serial_number,
172:     .str_serial_number_len = sizeof(tbl_usb_wifi_str_serial_number),
173:
174:     .ep_nums = 2,
175:     .ep_cfg[0].ep_id = USB_WIFI_RX_EP,
176:     .ep_cfg[0].ep_type = USB_ENDPOINT_XFER_BULK,
177:     .ep_cfg[0].ep_dir_tx = 0,
178:     .ep_cfg[0].max_packet_size = USB_WIFI_EP_MAX_PKT_SIZE,
179:     .ep_cfg[1].ep_id = USB_WIFI_TX_EP,
180:     .ep_cfg[1].ep_type = USB_ENDPOINT_XFER_BULK,
181:     .ep_cfg[1].ep_dir_tx = 1,
182:     .ep_cfg[1].max_packet_size = USB_WIFI_EP_MAX_PKT_SIZE,
183: };
184:

```

图4. 1. 1 自定义USB设备类配置示例

```

389: const struct usb_device_cfg usb_dev_audio_cfg = {
390:     .vid = USB_VID,
391:     .pid = USB_AUDIO_PID,
392:     .speed = USB_SPEED_HIGH,
393:     .p_device_descriptor = (uint8_t*)tbl_usb_audio_device_descriptor,
394:     .p_config_descriptor_head = (uint8_t*)tbl_usb_audio_config_all_descriptor_gen,
395:     .p_config_desc = (uint8_t*)tbl_usb_audio_config_all_descriptor_audio,
396:     .config_desc_len = sizeof(tbl_usb_audio_config_all_descriptor_audio),
397:     .interface_num = USB_AUDIO_INTERFACE_NUMS+1,
398:     .p_language_id = (uint8_t*)tbl_usb_audio_language_id,
399:     .language_id_len = sizeof(tbl_usb_audio_language_id),
400:     .p_str_manufacturer = (uint8_t*)tbl_usb_audio_str_manufacturer,
401:     .str_manufacturer_len = sizeof(tbl_usb_audio_str_manufacturer),
402:     .p_str_product = (uint8_t*)tbl_usb_audio_str_product,
403:     .str_product_len = sizeof(tbl_usb_audio_str_product),
404:     .p_str_serial_number = (uint8_t*)tbl_usb_audio_str_serial_number,
405:     .str_serial_number_len = sizeof(tbl_usb_audio_str_serial_number),
406:     .ep_nums = USB_AUDIO_INTERFACE_NUMS,
407:     .if (USB_AUDIO_TPYE == USB_AUDIO_SPEAKER | USB_AUDIO_MIC)
408:     {
409:         .ep_cfg[0].ep_id = USB_AUDIO_SPK_EP,
410:         .ep_cfg[0].ep_type = USB_ENDPOINT_XFER_ISOC,
411:         .ep_cfg[0].ep_dir_tx = 0,
412:         .ep_cfg[0].max_packet_size = USB_AUDIO_SPK_MAX_PKT_SIZE,
413:         .ep_cfg[1].ep_id = USB_AUDIO_MIC_EP,
414:         .ep_cfg[1].ep_type = USB_ENDPOINT_XFER_ISOC,
415:         .ep_cfg[1].ep_dir_tx = 1,
416:         .ep_cfg[1].max_packet_size = USB_AUDIO_MIC_MAX_PKT_SIZE,
417:     }
418:     #elif (USB_AUDIO_TPYE & USB_AUDIO_SPEAKER)
419:     {
420:         .ep_cfg[0].ep_id = USB_AUDIO_SPK_EP,
421:         .ep_cfg[0].ep_type = USB_ENDPOINT_XFER_ISOC,
422:         .ep_cfg[0].ep_dir_tx = 0,
423:         .ep_cfg[0].max_packet_size = USB_AUDIO_SPK_MAX_PKT_SIZE,
424:     }
425:     #elif (USB_AUDIO_TPYE & USB_AUDIO_MIC)
426:     {
427:         .ep_cfg[0].ep_id = USB_AUDIO_MIC_EP,
428:         .ep_cfg[0].ep_type = USB_ENDPOINT_XFER_ISOC,
429:         .ep_cfg[0].ep_dir_tx = 1,
430:         .ep_cfg[0].max_packet_size = USB_AUDIO_MIC_MAX_PKT_SIZE,
431:     }
432:     #else
433:     {
434:         .ep_cfg[0].ep_id = USB_AUDIO_MIC_EP,
435:         .ep_cfg[0].ep_type = USB_ENDPOINT_XFER_ISOC,
436:         .ep_cfg[0].ep_dir_tx = 1,
437:         .ep_cfg[0].max_packet_size = USB_AUDIO_MIC_MAX_PKT_SIZE,
438:     }
439:     #endif
440: };

```

图4.1.2 USB Audio类的配置示例

## 4.2. USB Device 工作流程

USB Device 配置和工作流程如：

图 4.2.1 USB Device 软件配置和工作主流程

图 4.2.2 USB Device EP0 后台枚举流程

常规情况下，简单的 USB Device 方案，用户只需要配置好 struct usb\_device\_cfg 即可实现；

如果配置 usb\_device\_cfg 无法满足用户需求，则可以重写绿色部分的函数实现用户想要实现的功能，此动作需要要求熟悉 USB 协议，非必要不建议使用。

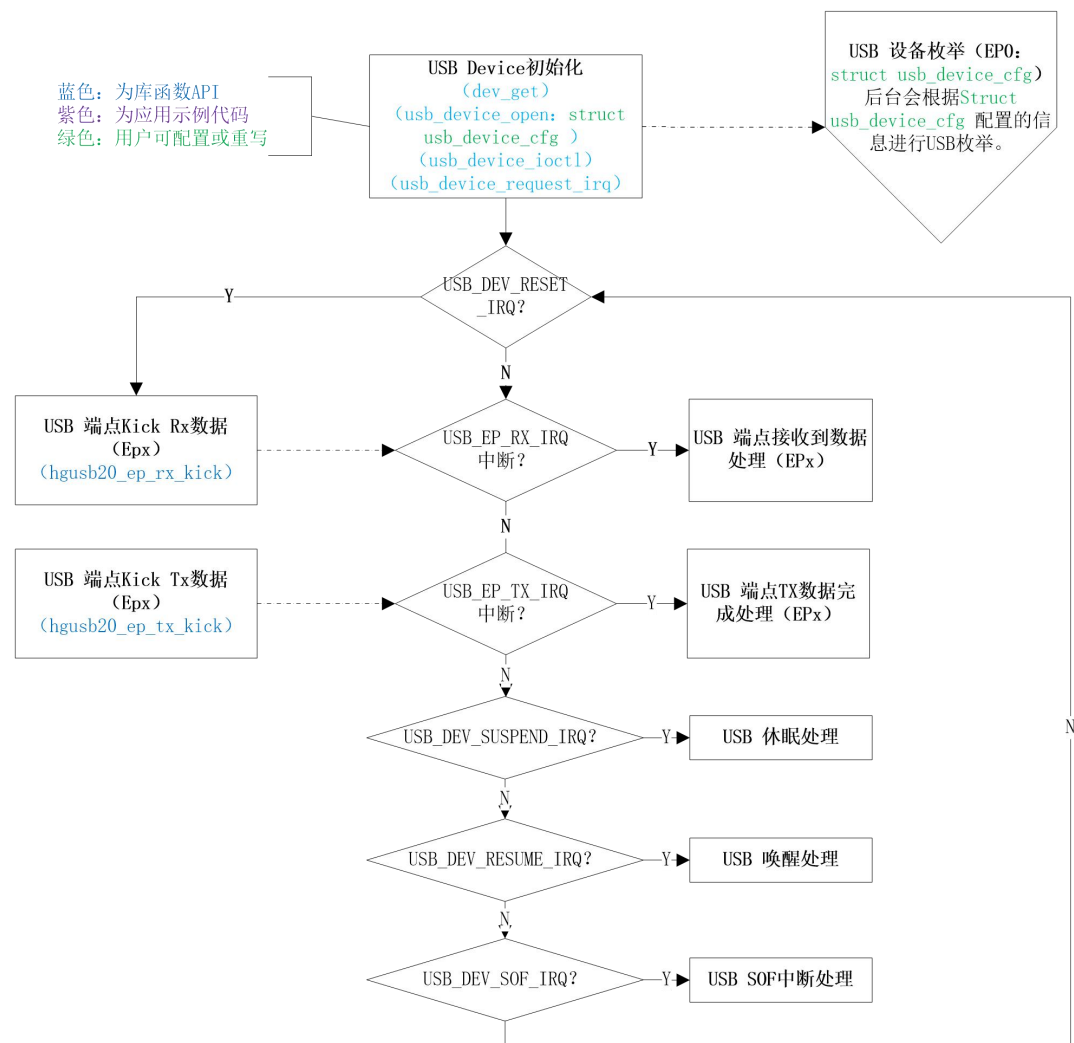


图4.2.1 USB Device软件配置和工作主流程

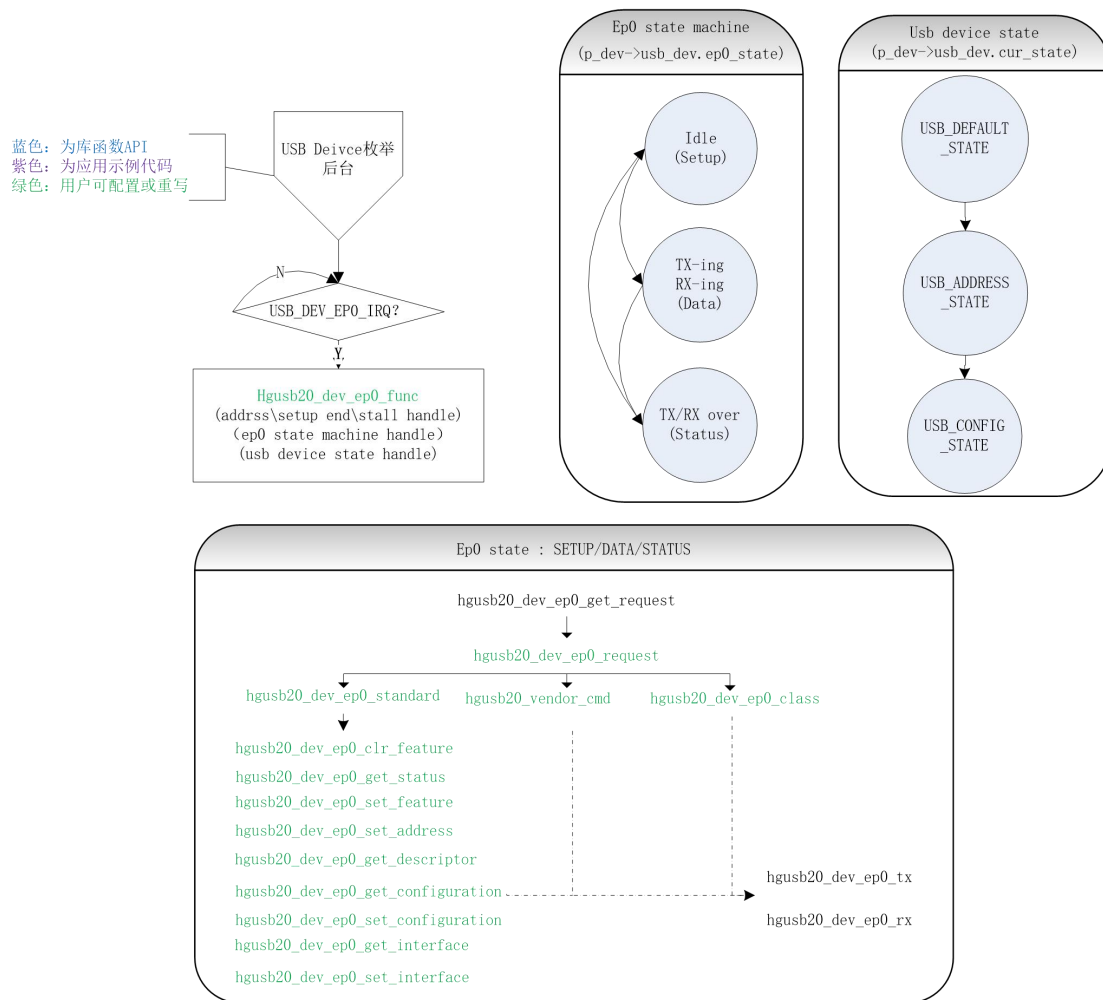


图4. 2. 2 USB Device EP0后台枚举流程