AN0097
应用笔记

AT32 MCU USB Device Library Application Note

前言

本文档主要描述 AT32F4xx 系列 USB 设备库的架构和使用方法，方便开发者使用库快速开发 USB 相关的应用，同时也会介绍 AT32 BSP 里面对应的 USB 例程。

USB 在设备互联上使用非常多，常用的鼠标，键盘，耳机，打印机等都可以使用 USB 进行连接，AT32 的 USB 设备库将实现这些常规的设备类（HID,AUDIO,CDC,MSC 等），开发人员可根据应用需求去修改这些设备类，以实现具体应用，支持库版本 AT32F4xx_Firmware_Library_V2.0.0。

支持型号列表：

支持型号	AT32F4xx

目录

表目录

图目录

1 AT32 USB 设备协议库

这部分主要介绍AT32 USB设备库的结构和库的使用方法，AT32 USB是基于USB2.0全速设备，不支持USB2.0高速设备。这里库的作用是用来管理USB外设和实现USB的基本协议，使开发者能够更快的上手开发。

USB Deivce库包含以下几个模块 如图1：

- 用户应用程序

  此部分为开发者根据应用具体需求开发应用程序。

- USB Core Driver和USB设备类

  USB Core Driver：此部分实现USB设备标准协议栈，标准请求等接口。

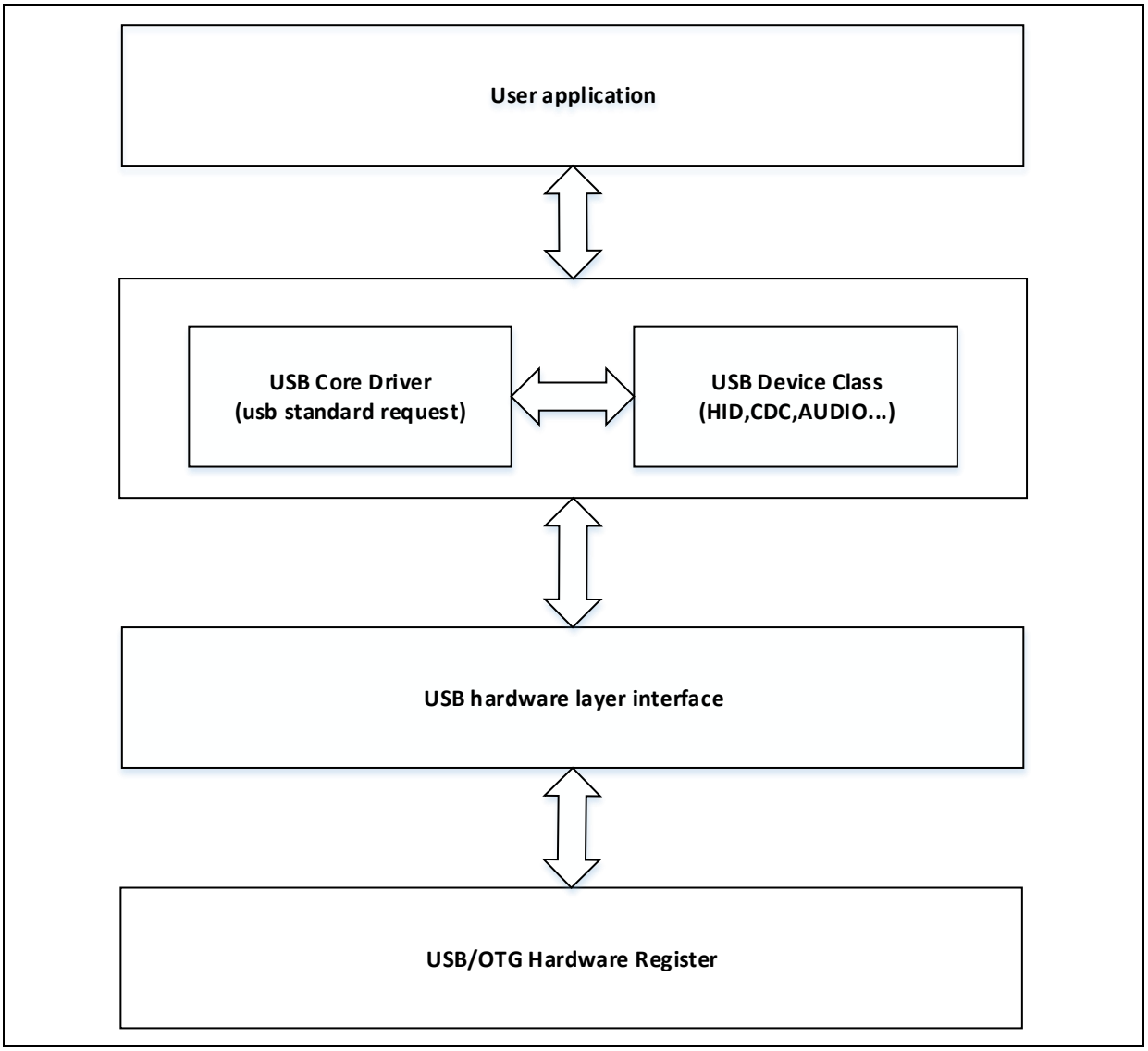
  USB设备类：此部分实现一个具体USB设备的描述和设备请求。

- USB硬件底层接口

  此部分实现硬件寄存器抽象接口

- USB/OTG外设

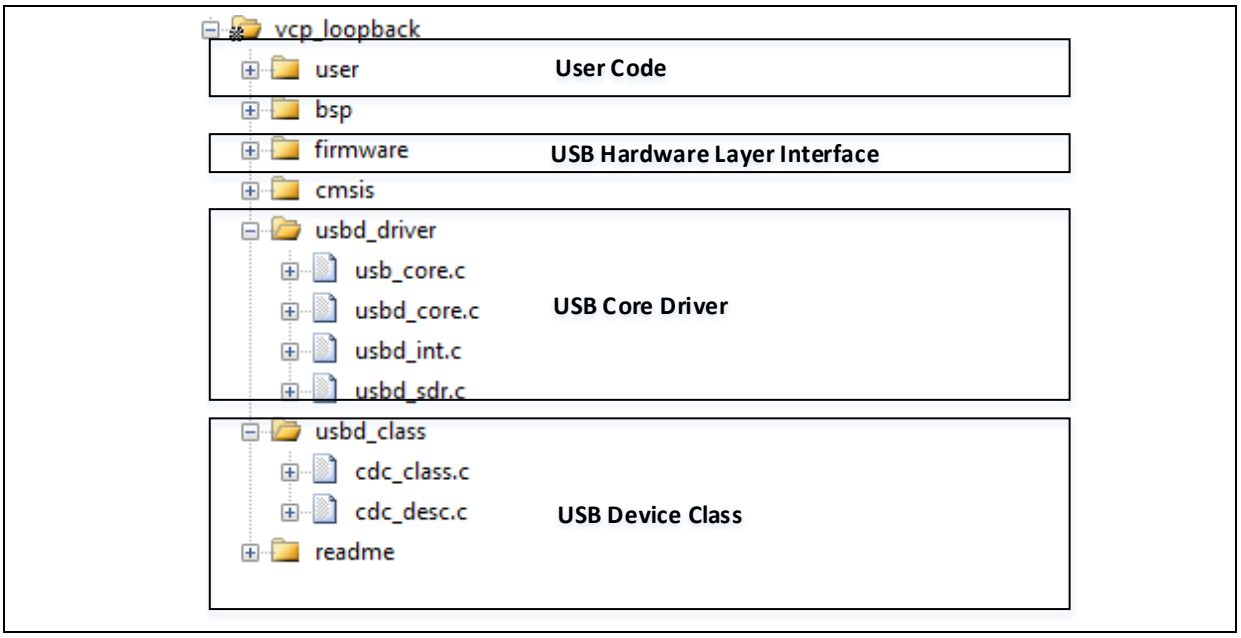
图 1. USB 库结构



1.1 AT32 USB 库文件

如下是 AT32 USB 应用工程结构图：

图 2. AT32 工程结构



Core Driver 库路径：OTGFS -->middlewares\usb_drivers

USBFS -->middlewares\usbd_drivers

Device Class 库路径：middlewares\usbd_class

如下是 USB 库文件列表：

表 1. USB 库文件列表

文件名	内容
usb_core.c/h	USB Core
usbd_core.c/h	USB Device Core Library
usbd_int.c/h	USB 中断处理
usbd_sdr.c/h	USB 标准请求
usbd_std.h	USB 标准头文件
at32f4xx_usb.c/h	USB 硬件寄存器接口

表 2. USB 设备类型文件列表

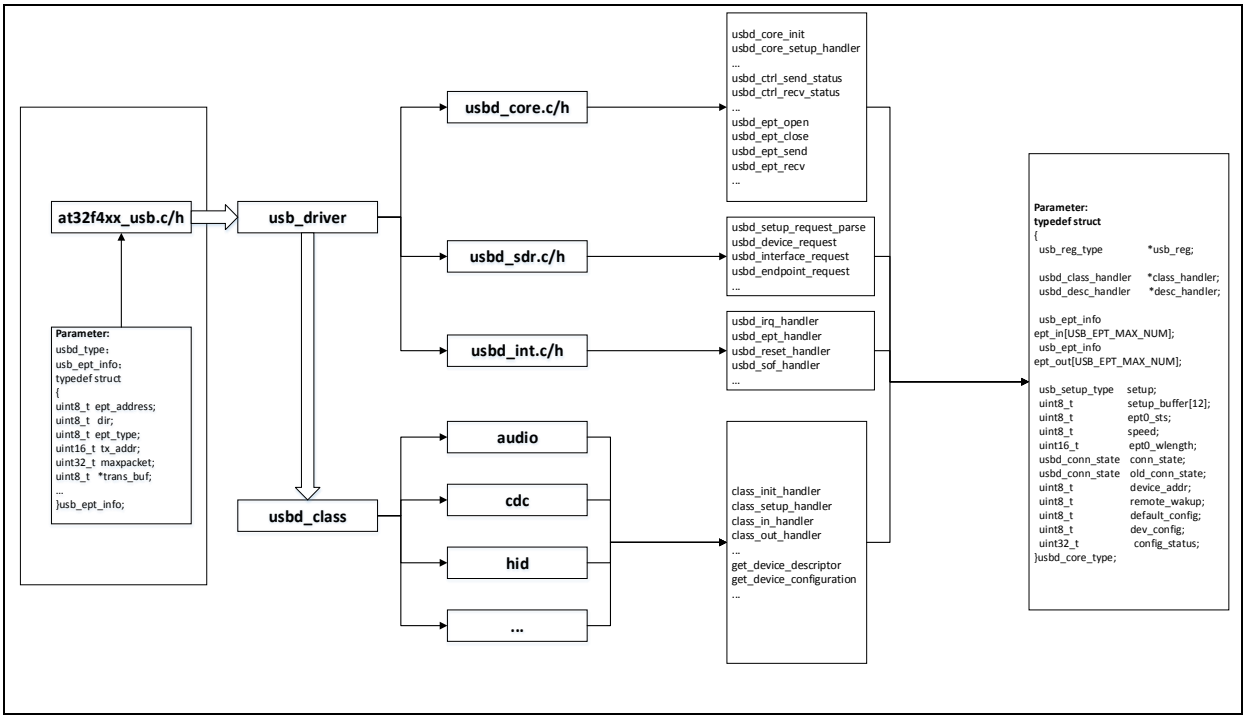
文件名	内容
audio_class.c/h	audio 设备数据处理接口
audio_desc.c/h	audio 设备描述
cdc_class.c/h	cdc 设备数据处理接口
cdc_desc.c/h	cdc 设备描述
cdc_keyboard_class.c/h	cdc 和 keyboard 复合设备数据处理接口
cdc_keyboard_desc.c/h	cdc 和 keyboard 复合设备描述
custom_hid_class.c/h	custom hid 数据处理接口
custom_hid_desc.c/h	custom hid 设备描述

文件名	内容
hid_iap_class.c/h	hid iap 数据处理接口
hid_iap_desc.c/h	hid iap 设备描述
keyboard_class.c/h	keyboard 数据处理接口
keyboard_desc.c/h	keyboard 设备描述
mouse_class.c/h	mouse 数据处理接口
mouse_desc.c/h	mouse 设备描述
msc_class.c/h	mass storage 数据处理接口
msc_bot_scsi.c/h	bulk-only transfer 和 scsi 命令处理接口
msc_desc.c/h	mass storage 设备描述
printer_class.c/h	printer 数据处理接口
printer_desc.c/h	printer 设备描述

2 USB 设备库文件说明

USB 库实现 USB 设备标准请求，同时实现 USB 规格里面的 4 种传输类型（control，interrupt，bulk，isochronous）的函数接口。

图 3 USB 库文件结构



2.1 USB 设备文件函数接口

2.1.1 usbd_int.c 函数接口

usbd_int.c 主要处理底层中断，不同 USB 外设，此部分会根据外设不同而有所改变。AT32 系列芯片 USB 外设存在 USBFS 和 OTGFS 两种外设，此部分函数接口基本相同。

表 3 usbd_int 函数接口

接口名称	描述
usbd_irq_handler	USB 中断入口函数
usbd_ept_handler	USB 端点中断函数
usbd_reset_handler	USB 复位中断函数
usbd_sof_handler	USB SOF 中断函数
usbd_suspend_handler	USB 挂起中断函数
usbd_wakeup_handler	USB 唤醒中断函数
…	…

2.1.2 usbd_core.c 函数接口

usbd_core.c 此文件封装不同的 usb 接口用于不同的地方调用，包括一些接收，发送函数等。

表 4 usbd_core 函数接口

接口名称	描述
usbd_core_init	usb device 初始化接口函数

接口名称	描述
usbd_core_in_handler	usb device IN 事务处理接口函数
usbd_core_out_handler	usb device OUT 事务处理接口函数
usbd_core_setup_handler	usb device setup 事务处理接口函数
usbd_ctrl_unsupport	不支持的传输接口函数
usbd_ctrl_send	控制传输发送接口函数
usbd_ctrl_recv	控制传输开始接收接口函数
usbd_ctrl_send_status	控制传输 IN 握手状态接口函数
usbd_ctrl_recv_status	控制传输 OUT 握手状态接口函数
usbd_set_stall	设置端点状态 STALL 接口函数
usbd_clear_stall	清除端点状态 STALL 接口函数
usbd_ept_open	打开端点接口函数
usbd_ept_close	关闭端点接口函数
usbd_ept_send	端点发送数据接口函数
usbd_ept_recv	端点开始接收接口函数
usbd_connect	USB 开始连接接口函数
usbd_disconnect	USB 断开连接接口函数
usbd_set_device_addr	设置 USB 设备地址
usbd_get_recv_len	获取当前接收数据长度
usbd_connect_state_get	获取当前 USB 状态
…	…

2.1.3 usbd_sdr.c 函数接口

usbd_sdr.c 此文件处理 USB 一些标准请求。

表 5 usbd_sdr 函数接口

接口名称	描述
usbd_device_request	设备请求接口函数
usbd_interface_request	接口请求接口函数
usbd_endpoint_request	端点请求接口函数
…	…

支持的标志设备请求如下表：

表 6 标准设备请求

请求	描述
GET_STATUS	获取状态
CLEAR_FEATURE	清除 feature
SET_FEATURE	设置 feature
SET_ADDRESS	设置设备地址
GET_DESCRIPTOR	获取设备描述
GET_CONFIGURATION	获取设备配置
SET_CONFIGURATION	设置设备配置
GET_INTERFACE	获取 alternate setting
SET_INTERFACE	设置 alternate setting

2.1.4 usbd_xx_class.c 函数接口

usbd_xx_class.c 此文件为具体设备类型的数据处理，通过结构体函数来实现不同设备类型数据的处理。开发者根据不同的设备类型，来实现下面函数结构体中的函数，达到不同应用结果。
函数结构体如下：

表 7 设备 class 函数结构体

```
typedef struct
{
  usb_sts_type (*init_handler)(void *udev);                              /*!< usb class init handler */
  usb_sts_type (*clear_handler)(void *udev);                             /*!< usb class clear handler */
  usb_sts_type (*setup_handler)(void *udev, usb_setup_type *setup);      /*!< usb class setup handler */
  usb_sts_type (*ept0_tx_handler)(void *udev);                           /*!< usb class endpoint 0 tx complete handler */
  usb_sts_type (*ept0_rx_handler)(void *udev);                           /*!< usb class endpoint 0 rx complete handler */
  usb_sts_type (*in_handler)(void *udev, uint8_t ept_num);               /*!< usb class in transfer complete handler */
  usb_sts_type (*out_handler)(void *udev, uint8_t ept_num);              /*!< usb class out transfer complete handler */
  usb_sts_type (*sof_handler)(void *udev);                               /*!< usb class sof handler */
  usb_sts_type (*event_handler)(void *udev, usbd_event_type event);      /*!< usb class event handler */
}usbd_class_handler;
```

表 8 设备 class 函数接口

接口函数	描述
init_handler	Class 初始化接口
clear_handler	Class 清除接口
setup_handler	设备类型 setup 接口函数
ept0_tx_handler	端点 0 握手阶段发送完成接口函数
ept0_rx_handler	端点 0 握手阶段接收完成接口函数
in_handler	IN 端点传输完成接口函数
out_handler	OUT 端点接收完成接口函数
sof_handler	SOF 中断接口函数
event_handler	USB 事件接口函数

2.1.5 usbd_xx_desc.c 函数接口

usbd_xx_desc.c 此文件为设备描述文件，设备描述信息都通过此文件的函数接口返回给主机。

表 9 设备描述函数结构体

```
typedef struct
{
  usbd_desc_t *(*get_device_descriptor)(void);                   /*!< get device descriptor callback */
  usbd_desc_t *(*get_device_qualifier)(void);                    /*!< get device qualifier callback */
  usbd_desc_t *(*get_device_configuration)(void);                /*!< get device configuration callback */
  usbd_desc_t *(*get_device_other_speed)(void);                  /*!< get device other speed callback */
  usbd_desc_t *(*get_device_lang_id)(void);                      /*!< get device lang id callback */
  usbd_desc_t *(*get_device_manufacturer_string)(void);          /*!< get device manufacturer callback */
  usbd_desc_t *(*get_device_product_string)(void);               /*!< get device product callback */
```

```
  usbd_desc_t *(*get_device_serial_string)(void);          /*!< get device serial callback */
  usbd_desc_t *(*get_device_interface_string)(void);       /*!< get device interface string callback */
  usbd_desc_t *(*get_device_config_string)(void);          /*!< get device device config callback */
}usbd_desc_handler;
```

表 10 设备描述接口函数

接口函数	描述
get_device_descriptor	获取设备描述信息
get_device_qualifier	获取设备 qualifier 描述信息
get_device_configuration	获取设备配置信息
get_device_other_speed	获取设备 other speed 配置信息
get_device_lang_id	获取设备 lang id
get_device_manufacturer_string	获取设备厂商描述
get_device_product_string	获取设备产品信息描述
get_device_serial_string	获取设备序列号
get_device_interface_string	获取接口信息描述
get_device_config_string	获取配置信息描述

2.1.6 其它参数

函数的参数结构体如下，USB 设备库中参数传递使用结构体 usbd_core_type，如下图：

图 4 全局结构体

```
typedef struct
{
  usb_reg_type              *usb_reg;            /*!< usb register pointer */
  usbd_class_handler        *class_handler;      /*!< usb device class handler pointer */
  usbd_desc_handler         *desc_handler;       /*!< usb device descriptor handler pointer */
  usb_ept_info               ept_in[USB_EPT_MAX_NUM];    /*!< usb in endpoint infomation struct */
  usb_ept_info               ept_out[USB_EPT_MAX_NUM];   /*!< usb out endpoint infomation struct */
  usb_setup_type             setup;                      /*!< usb setup type struct */
  uint8_t                    setup_buffer[12];           /*!< usb setup request buffer */
  uint8_t                    ept0_sts;                   /*!< usb control endpoint 0 state */
  uint8_t                    speed;                      /*!< usb speed */
  uint16_t                   ept0_wlength;               /*!< usb endpoint 0 transfer length */
  usbd_conn_state            conn_state;                 /*!< usb current connect state */
  usbd_conn_state            old_conn_state;             /*!< usb save the previous connect state */
  uint8_t                    device_addr;                /*!< device address */
  uint8_t                    remote_wakup;               /*!< remote wakeup state */
  uint8_t                    default_config;             /*!< usb default config state */
  uint8_t                    dev_config;                 /*!< usb device config state */
  uint32_t                   config_status;              /*!< usb configure status */
}usbd_core_type;
```

USB 设备的连接状态如下图：

连接状态包含：

- 默认状态

- 地址状态
- 配置状态
- 挂起状态

可使用 usbd_connect_state_get 函数查询当前 USB 设备的连接状态。

图 5 USB 设备连接状态

```
typedef enum
{
  USB_CONN_STATE_DEFAULT          =1, /*!< usb device connect state default */
  USB_CONN_STATE_ADDRESSED,           /*!< usb device connect state address */
  USB_CONN_STATE_CONFIGURED,          /*!< usb device connect state configured */
  USB_CONN_STATE_SUSPENDED            /*!< usb device connect state suspend */
}usbd_conn_state;
```

USB 设备返回值，USB 函数接口使用如下函数返回值。

图 6 函数返回值

```
typedef enum
{
  USB_OK,              /*!< usb status ok */
  USB_FAIL,            /*!< usb status fail */
  USB_WAIT,            /*!< usb status wait */
  USB_NOT_SUPPORT,     /*!< usb status not support */
  USB_ERROR,           /*!< usb status error */
}usb_sts_type;
```

2.2 端点 FIFO 分配

USB 要正常收发数据，在初始化时需要为每个端点分配发送/接收的 FIFO，FIFO 的大小可以根据端点上传输的最大包长度确认。注意，分配给所有端点 FIFO 大小的和不能超过系统分配给 USB 缓冲区的最大长度，具体 USB 的缓冲区大小参考 RM 上的描述。
开发者可以参考例程 usb_conf.h 为每个端点自定义分配 FIFO。另外使用 USBFS 和 OTGFS 两种不同的外设时，usb_conf.h 中对端点 FIFO 分配稍有不同。

2.2.1 USBFS 外设端点 FIFO 分配

USBFS 外设端点分配实现了两种分配方式，一种自动分配，一种是用户自定义分配。

- 自动分配:
  1. 通过打开 usb_conf.h 中 USB_EPT_AUTO_MALLOC_BUFFER 宏开启自动分配功能
  2. 在调用打开端点函数（usbd_ept_open）时自动根据传入的最大包长度自动分配 FIFO
  3. 如果使用双缓冲模式（同步端点，双缓冲 Bulk）时，在打开端点前先调用（usbd_ept_dbuffer_enable）表示使用双缓冲模式，可参考 audio 例程。
  4. 自动分配 audio 例程配置如下：

     usb_conf.h

```
/**
* @brief auto malloc usb endpoint buffer
*/
```

```
#define USB_EPT_AUTO_MALLOC_BUFFER    /*!< usb auto malloc endpoint tx and rx buffer */
```

audio_class.c:端点打开

```
usb_sts_type class_init_handler(void *udev)
{
  usb_sts_type status = USB_OK;
  usbd_core_type *pudev = (usbd_core_type *)udev;

  /* enable microphone in endpoint double buffer mode */
  usbd_ept_dbuffer_enable(pudev, USBD_AUDIO_MIC_IN_EPT);

  /* open microphone in endpoint */
   usbd_ept_open(pudev, USBD_AUDIO_MIC_IN_EPT, EPT_ISO_TYPE,
   AUDIO_MIC_IN_MAXPACKET_SIZE);

  /* enable speaker out endpoint double buffer mode */
  usbd_ept_dbuffer_enable(pudev, USBD_AUDIO_SPK_OUT_EPT);

  /* open speaker out endpoint */
   usbd_ept_open(pudev, USBD_AUDIO_SPK_OUT_EPT, EPT_ISO_TYPE,
   AUDIO_SPK_OUT_MAXPACKET_SIZE);

  /* enable speaker feedback endpoint double buffer mode */
  usbd_ept_dbuffer_enable(pudev, USBD_AUDIO_FEEDBACK_EPT);

  /* open speaker feedback endpoint */
 usbd_ept_open(pudev, USBD_AUDIO_FEEDBACK_EPT, EPT_ISO_TYPE,
 AUDIO_FEEDBACK_MAXPACKET_SIZE);

    /* start receive speaker out data */
    usbd_ept_recv(pudev, USBD_AUDIO_SPK_OUT_EPT, audio_struct.audio_spk_data,
    AUDIO_SPK_OUT_MAXPACKET_SIZE);
    return status;
}
```

- 自定义分配:
  1. 关闭 usb_conf.h 中 USB_EPT_AUTO_MALLOC_BUFFER 宏开启自定义分配
  2. 在调用打开端点函数（usbd_ept_open）时调用 usbd_ept_buf_custom_define 函数自定义为端点分配 FIFO，参考 vcp_loopback 例程。
  3. vcp_loopback 例程自定义分配配置

     usb_conf.h:

```
/**
* @brief auto malloc usb endpoint buffer
*/
//#define USB_EPT_AUTO_MALLOC_BUFFER    /*!< usb auto malloc endpoint tx and rx buffer */
```

```
#ifndef USB_EPT_AUTO_MALLOC_BUFFER
      /**
* @brief user custom endpoint buffer
*            EPTn_TX_ADDR, EPTn_RX_ADDR must less than usb buffer size
*/
/* ept0 tx start address 0x40, size 0x40, so rx start address is 0x40 + 0x40 = 0x80 */
#define EPT0_TX_ADDR                    0x40      /*!< usb endpoint 0 tx buffer address offset */
#define EPT0_RX_ADDR                    0x80      /*!< usb endpoint 0 rx buffer address offset */

#define EPT1_TX_ADDR                    0xC0      /*!< usb endpoint 1 tx buffer address offset */
#define EPT1_RX_ADDR                    0x100     /*!< usb endpoint 1 rx buffer address offset */

#define EPT2_TX_ADDR                    0x140     /*!< usb endpoint 2 tx buffer address offset */
#define EPT2_RX_ADDR                    0x180     /*!< usb endpoint 2 rx buffer address offset */

#define EPT3_TX_ADDR                    0x00      /*!< usb endpoint 3 tx buffer address offset */
#define EPT3_RX_ADDR                    0x00      /*!< usb endpoint 3 rx buffer address offset */

#define EPT4_TX_ADDR                    0x00      /*!< usb endpoint 4 tx buffer address offset */
#define EPT4_RX_ADDR                    0x00      /*!< usb endpoint 4 rx buffer address offset */

#define EPT5_TX_ADDR                    0x00      /*!< usb endpoint 5 tx buffer address offset */
#define EPT5_RX_ADDR                    0x00      /*!< usb endpoint 5 rx buffer address offset */

#define EPT6_TX_ADDR                    0x00      /*!< usb endpoint 6 tx buffer address offset */
#define EPT6_RX_ADDR                    0x00      /*!< usb endpoint 6 rx buffer address offset */

#define EPT7_TX_ADDR                    0x00      /*!< usb endpoint 7 tx buffer address offset */
#define EPT7_RX_ADDR                    0x00      /*!< usb endpoint 7 rx buffer address offset */
```

cdc_class.c 端点打开：

```
usb_sts_type class_init_handler(void *udev)
{
  usb_sts_type status = USB_OK;
  usbd_core_type *pudev = (usbd_core_type *)udev;
#ifndef USB_EPT_AUTO_MALLOC_BUFFER
   /* use user define buffer address */
  usbd_ept_buf_custom_define(pudev, USBD_CDC_INT_EPT, EPT2_TX_ADDR);
  usbd_ept_buf_custom_define(pudev, USBD_CDC_BULK_IN_EPT, EPT1_TX_ADDR);
  usbd_ept_buf_custom_define(pudev, USBD_CDC_BULK_OUT_EPT, EPT1_RX_ADDR);
#endif
  /* open in endpoint */
  usbd_ept_open(pudev, USBD_CDC_INT_EPT, EPT_INT_TYPE, USBD_IN_MAXPACKET_SIZE);
  /* open in endpoint */
   usbd_ept_open(pudev, USBD_CDC_BULK_IN_EPT, EPT_BULK_TYPE,
```

```
    USBD_OUT_MAXPACKET_SIZE);

  /* open out endpoint */
   usbd_ept_open(pudev, USBD_CDC_BULK_OUT_EPT, EPT_BULK_TYPE,
   USBD_OUT_MAXPACKET_SIZE);

  /* set out endpoint to receive status */
  usbd_ept_recv(pudev, USBD_CDC_BULK_OUT_EPT, g_rx_buff, USBD_OUT_MAXPACKET_SIZE);
  g_tx_completed = 1;
  return status;
}
```

2.2.2 OTGFS 外设端点 FIFO 分配

OTGFS 对于端点的接收缓冲是共享的，因此对于所有的 OUT 端点，只需要分配一个接收 FIFO。对发送缓冲区则需要为每个发送端点分配一块自己专用的 FIFO。支持的端点个数请参考对应型号的 RM。OTGFS 的端点分配都需要开发者根据端点支持的最大包长度进行分配，注意 usb_conf.h 中对端点分配的 FIFO 大小单位为 word（Byte）。

以 vcp_loopback 例程为例：

```
#define USBD_RX_SIZE            128   /*共享的一个接收 FIFO，大小 128*4 Byte */
#define USBD_EP0_TX_SIZE         24   /*端点 0 发送 FIFO，大小 24*4 Byte */
#define USBD_EP1_TX_SIZE         20   /*端点 1 发送 FIFO，大小 20*4 Byte */
#define USBD_EP2_TX_SIZE         20   /*端点 2 发送 FIFO，大小 20*4 Byte */
#define USBD_EP3_TX_SIZE         20   /*端点 3 发送 FIFO，大小 20*4 Byte */
#define USBD_EP4_TX_SIZE         20   /*端点 4 发送 FIFO，大小 20*4 Byte */
#define USBD_EP5_TX_SIZE         20   /*端点 5 发送 FIFO，大小 20*4 Byte */
#define USBD_EP6_TX_SIZE         20   /*端点 6 发送 FIFO，大小 20*4 Byte */
#define USBD_EP7_TX_SIZE         20   /*端点 7 发送 FIFO，大小 20*4 Byte */
```

2.3 USB 设备初始化

在使用 USB 时，需要对 USB 的寄存器做一些基本的初始化，通过调用 USB 的初始化函数完成这部分的操作，对于外设 USBFS 和 OTGFS 在初始化时所调用的函数一定的区别。

2.3.1 USBFS 外设初始化

USBFS 初始化函数 usbd_core_init 包含 5 个参数：

```
void usbd_core_init(usbd_core_type *udev,
                    usb_reg_type *usb_reg,
                    usbd_class_handler *class_handler,
                    usbd_desc_handler *desc_handler,
                    uint8_t core_id)
```

例程 vcp_loopback 的初始化如下：

```
usbd_core_init(&usb_core_dev, USB, &class_handler, &desc_handler, 0);
```

2.3.2 OTGFS 外设初始化

OTGFS 初始化函数 usbd_init 包含 5 个参数：

```
usb_sts_type usbd_init(otg_core_type *otgdev,
                       uint8_t core_id, uint8_t usb_id,
                       usbd_class_handler *class_handler,
                       usbd_desc_handler *desc_handler)
```

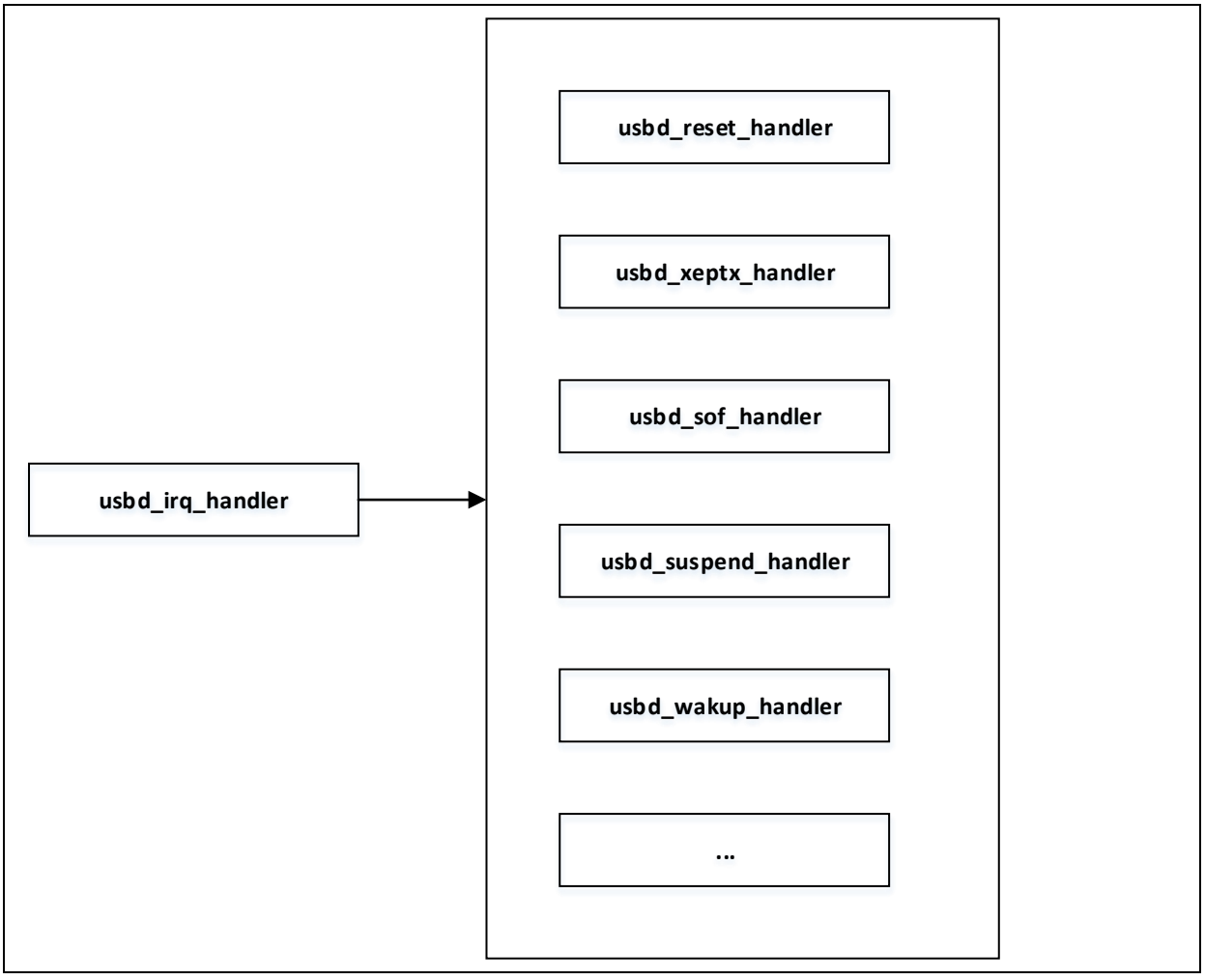
例程 vcp_loopback 的初始化如下：

```
usbd_init(&otg_core_struct,
          USB_FULL_SPEED_CORE_ID,
          USB_ID,
          &class_handler,
          &desc_handler);
```

2.4 USB 设备中断处理

USB 中断入口函数 usbd_irq_handler 处理所有 USB 中断，包括 Reset，端点收发数据，SOF，挂起，唤醒等中断，下面介绍一些典型的中断处理。

图 7 USB 中断处理函数



2.4.1 Reset 中断处理

当 USB 设备在总线上检测到 Reset 信号时，将产生 Reset 中断。软件在收到 Reset 中断时，需要做基本的初始化，用于后面的枚举处理。

Reset 中断处理函数 usbd_reset_handler：

- 端点 FIFO 初始化
- 设备地址设置为 0
- 端点 0 初始化
- 调用设备类的事件函数
  udev->class_handler->event_handler(udev, USBD_RESET_EVENT);

2.4.2 端点中断处理

当 USB 端点收发数据完成时，将产生对应的端点完成中断，端点完成中断处理发送和接收的数据。

中断处理函数：**usbd_xeptx_handler**

2.4.3 SOF 中断处理

打开 SOF 中断之后，USB 设备在每收到一个主机发送的 SOF 就会产生 SOF 中断。

中断处理函数：usbd_sof_handler

- 中断处理函数会调用设备类的 SOF 处理函数
  udev->class_handler->sof_handler(udev);

2.4.4 Suspend 中断处理

当总线满足挂起条件时，USB 设备会产生一个挂起中断，开发者可根据此中断判断是否需要进入低功耗模式。

中断处理函数：usbd_suspend_handler

- 连接状态设置为挂起状态
- 设置设备进入挂起状态
- 调用设备类的事件处理函数
  udev->class_handler->event_handler(udev, USBD_SUSPEND_EVENT);

2.4.5 Wakeup 中断处理

当设备在挂起状态时，如果总线上有 wakeup 信号，USB 设备将产生 wakeup 中断。

中断处理函数：usbd_wakeup_handler

- 设备退出挂起状态
- 连接状态设置为进入挂起之前的状态
- 调用设备类事件处理函数
  udev->class_handler->event_handler(udev, USBD_WAKEUP_EVENT);

2.5 USB 设备端点数据处理流程

USB 设备在收到主机发送的数据包之后，对应端点 0 的数据（IN/OUT/SETUP）会做单独处理，其它端点的数据会调用设备类的 IN/OUT handler 进行数据处理。

如下图所示数据的处理过程：

图 8 端点数据处理流程

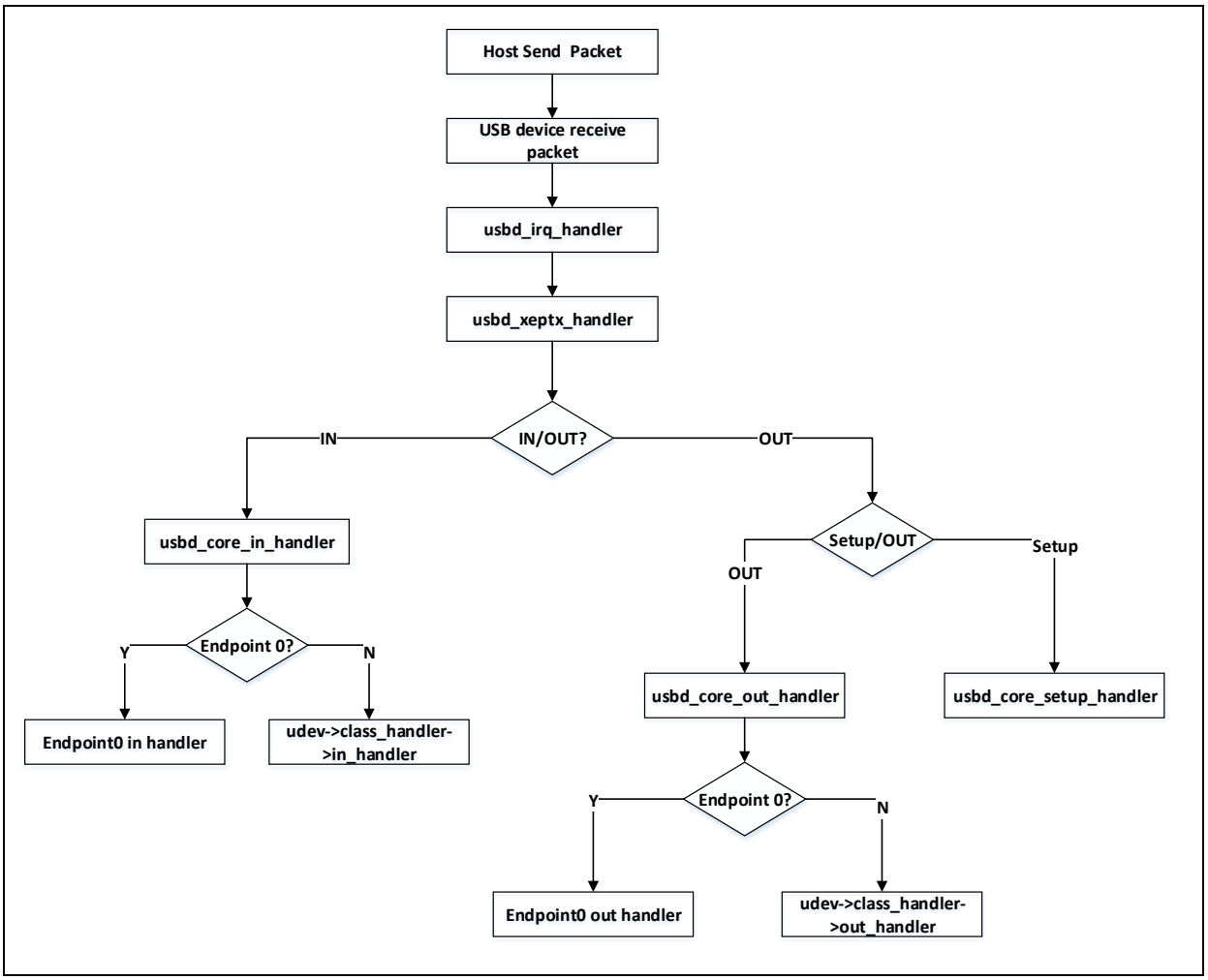
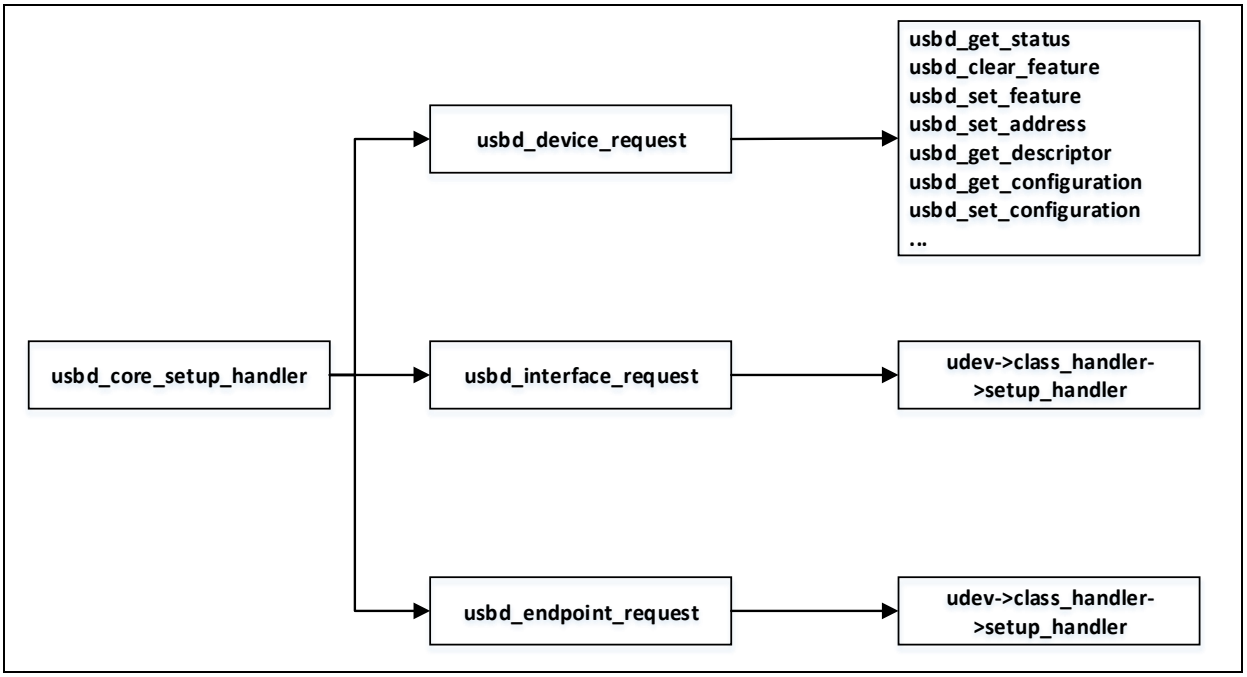


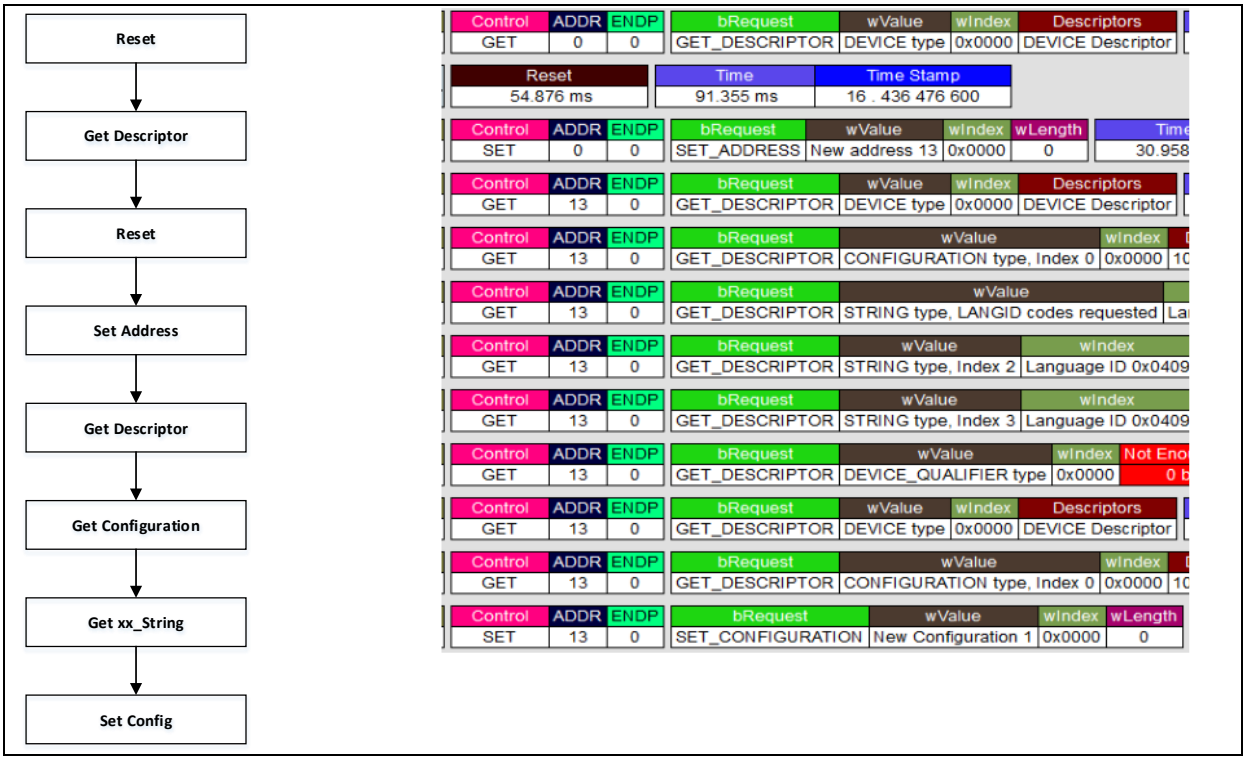
图 9 Setup 处理流程



2.5.1 USB 控制端点枚举流程

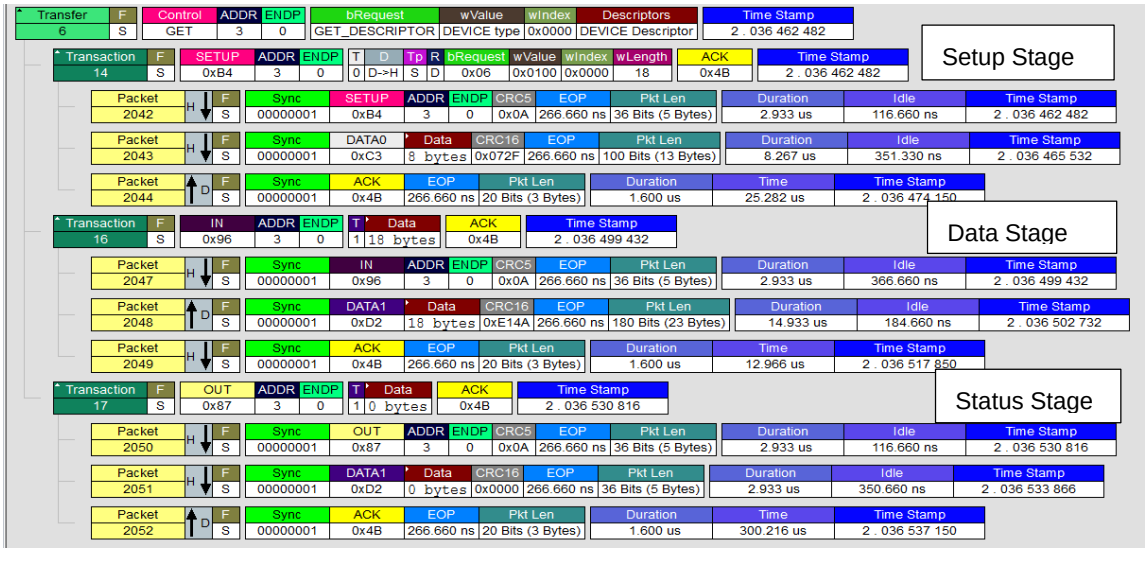
当设备接到主机之后，通过控制端点（端点 0）进行枚举动作，典型的枚举流程图如下：

图 10 USB 枚举流程



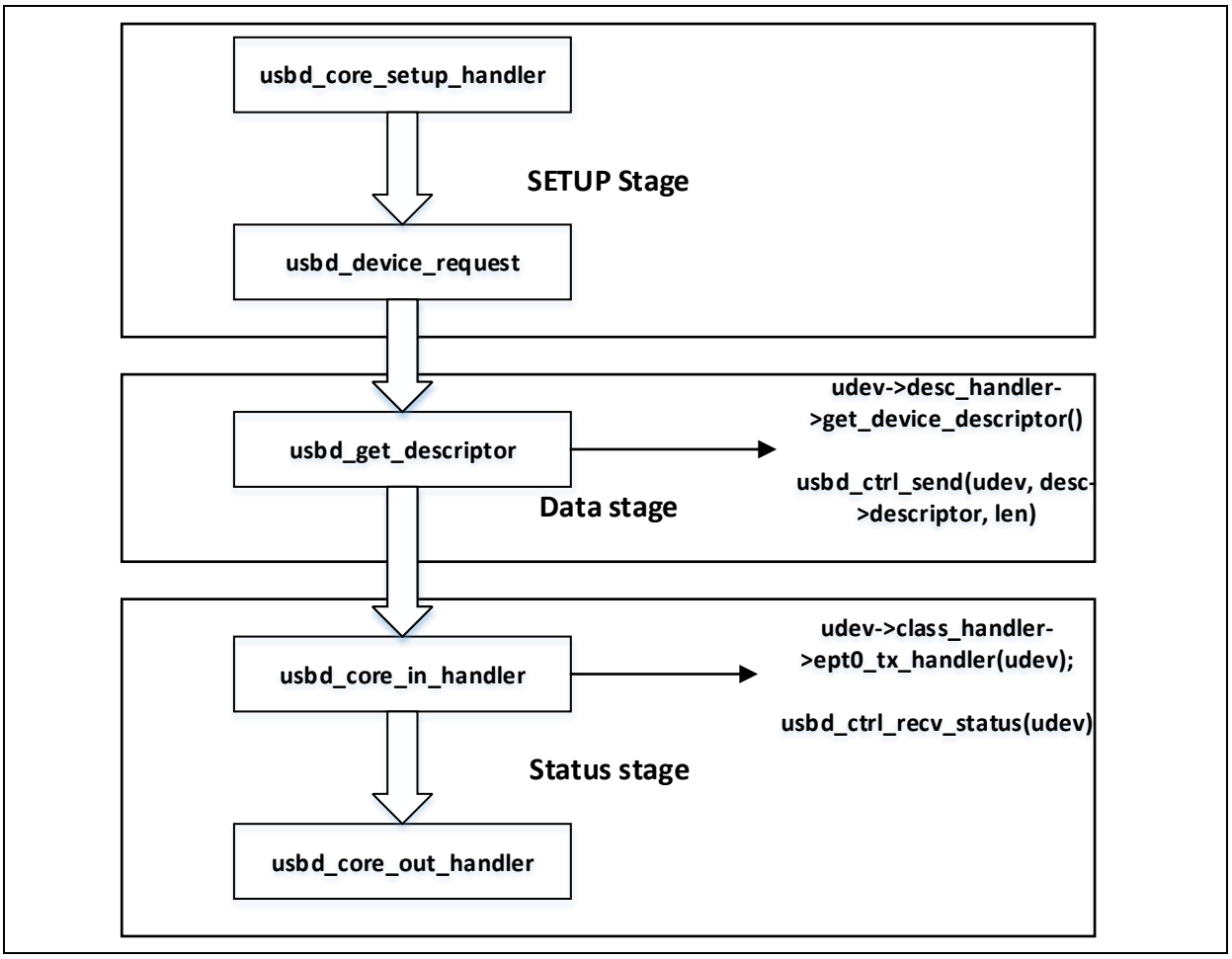
USB 控制传输过程包含 SETUP-DATA-STATUS 三个阶段， 如下是一个主机获取设备信息的传输过程 GET_DESCRIPTOR：

图 11 Get Descriptor



如下是 USB 库处理上图 Get Descriptor 的流程：

图 12 USB 库处理 Get Descriptor 调用流程



USB 设备请求格式（Setup 请求）

图 13 Setup 请求格式

Offset	Field	Size	Value	Description
0	*bmRequestType*	1	Bitmap	Characteristics of request: D7: Data transfer direction 0 = Host-to-device 1 = Device-to-host D6...5: Type 0 = Standard 1 = Class 2 = Vendor 3 = Reserved D4...0: Recipient 0 = Device 1 = Interface 2 = Endpoint 3 = Other 4...31 = Reserved
1	*bRequest*	1	Value	Specific request (refer to Table 9-3)
2	*wValue*	2	Value	Word-sized field that varies according to request
4	*wIndex*	2	Index or Offset	Word-sized field that varies according to request; typically used to pass an index or offset
6	*wLength*	2	Count	Number of bytes to transfer if there is a Data stage

2.5.2 USB 应用端点处理流程

应用端点指客户实际应用使用到的非 0 端点，包括 Bulk，interrupt，ISO 等端点类型，这些端点的数据通回调函数 in_handler 和 out_handler 进行处理。开发者只需要在 xxx_class.c 中 class_in_handler 和 class_out_handler 里面实现具体端点的数据处理即可。

IN 端点数据处理：

图 14 IN 端点数据处理

```
usb_sts_type class_in_handler(void *udev, uint8_t ept_num)
{
  usbd_core_type *pudev = (usbd_core_type *)udev;
  usb_sts_type status = USB_OK;
  if((ept_num & 0x7F) == 0x1)
  {
    /* IN endpoint 1 transfer complete*/
    /*trans next packet data
     usbd_ept_send(pudev, 0x1, send_data, len);
    */
  }
  if((ept_num & 0x7F) == 0x2)
  {
    /* IN endpoint 2 transfer complete*/
    /*trans next packet data
     usbd_ept_send(pudev, 0x2, send_data, len);
    */
  }
  ….
  return status;
}
```

OUT 端点数据处理:

图 15 OUT 端点数据处理

```
usb_sts_type class_out_handler(void *udev, uint8_t ept_num)
{
  usb_sts_type status = USB_OK;
  usbd_core_type *pudev = (usbd_core_type *)udev;
  if((ept_num & 0x7F) == 0x1)
  {
    /* get endpoint receive data length  */
    g_rxlen = usbd_get_recv_len(pudev, ept_num);
    /* start receive next data
    usbd_ept_recv(pudev, 0x1, buffer, max_packet_size);
    */
  }
  if((ept_num & 0x7F) == 0x2)
```

```
  {
    /* get endpoint receive data length    */
    g_rxlen = usbd_get_recv_len(pudev, ept_num);
    /* start receive next data
    usbd_ept_recv(pudev, 0x2, buffer, max_packet_size);
    */
  }
  …
  return status;
}
```

3 USB 设备类型例程

本章将说明使用 AT32 USB 设备库实现不同的设备类型的例程。目前实现的设备例程如下：

- Audio
- custom_hid
- keyboard
- mouse
- msc（mass strorage）
- printer
- vcp_loopback
- virtual_msc_iap
- composite_vcp_keyboard
- hid_iap

3.1 Audio 例程

Audio 例程使用 audio V1.0 的协议实现一个 Speaker 和 Microphone，传输 Audio 数据使用同步传输，Speaker 使用同步 OUT 传输，Microphone 使用同步 IN 传输。
Audio 的例程运行在 AT-START 评估板上，Audio Speaker 和 Mircophone 是基于 Audio Arduino Demo Board 进行实现，实验过程中需要连接 AT-START 和 Audio Arduino Board，更多开发板信息请参考《UM_Audio Arduino Daughter Board_V1.0/V2.0》,Audio 协议请参数《Universal Serial Bus Device Class Define for Audio Device V1.0》。

3.1.1 实现功能

实现一个 Speaker 和 Microphone 的 Audio 复合设备，可以同时进行音频播放和录音。
Speaker 功能：

- 支持 16K，48K 采样率
- 支持采样率切换
- 支持 16bit 采样
- 支持静音
- 支持音量调节
- 支持 feedback 功能
- 支持双通道

Microphone 功能：

- 支持 16K，48K 采样率
- 支持采样率切换
- 支持 16bit 采样
- 支持静音
- 支持音量调节
- 支持双通道

3.1.2 外设资源使用

USB 外设：

- 端点 0 IN/OUT：用于 USB 枚举以及 Audio 部分控制。
- 端点 1 IN：用于 Microphone 录音数据
- 端点 1 OUT：用于 Speaker 播放数据
- 端点 2 IN：用于 Feedback 数据

I2C：

- 使用 I2C 发送控制信息到音频 Board

I2S：

- 使用 I2S1 发送数据到音频 board（speaker）
- 使用 I2S2 从音频 board 接收数据（microphone）

DMA：

- 使用 DMA1 通道 3 传输 I2S1 的数据
- 使用 DMA1 通道 4 传输 I2S2 的数据

TIMER：

- 使用 TIMER 产生 Codec 所需要的时钟

3.1.3 Audio 设备实现

USB Audio 设备类实现源文件在 audio_class.c 和 audio_desc.c 中，外部 codec 的控制以及音频数据的处理都在 audio_codec.c 中实现，根据主机的请求设置会调用到 audio_codec.c 中具体的设置函数。另外需要特别注意关于 USB 设备端点 FIFO 大小的分配在 usb_conf.h 中配置，此部分根据具体端点传输的最大包长度进行分配。

3.1.3.1 设备描述：（audio_desc.c/h）

- Audio 设备的描述（g_usbd_descriptor）
- Audio 设备的配置描述信息（g_usbd_configuration）

  AC interface
  Mricrophone Streaming interface
  Microphone Terminal INPUT/OUTPUT
  Microphone Feature Unit
  Microphone Endpoint
  Speaker Streaming interface
  Speaker Terminal INPUT/OUTPUT
  Speaker Feature Unit
  Speaker Endpoint
  Feedback Endpoint
- Lang id （g_string_lang_id）
- 序列号（g_string_serial）
- 厂商产品 ID（audio_desc.h）

```
#define USBD_VENDOR_ID                0x2E3C
#define USBD_PRODUCT_ID               0x5730
```

- 制造商、产品名、配置描述、接口描述（audio_desc.h）

```
#define USBD_DESC_MANUFACTURER_STRING     "Artery"
#define USBD_DESC_PRODUCT_STRING          "AT32 Audio"
#define USBD_DESC_CONFIGURATION_STRING    "Audio Config"
#define USBD_DESC_INTERFACE_STRING        "Audio Interface"
```

3.1.3.2 数据处理部分（audio_class.c/h audio_codec.c/h ）

- 端点初始化（class_init_handler）

```
/* open microphone in endpoint */
usbd_ept_open(pudev, USBD_AUDIO_MIC_IN_EPT, EPT_ISO_TYPE, AUDIO_MIC_IN_MAXPACKET_SIZE);
/* open speaker out endpoint */
usbd_ept_open(pudev, USBD_AUDIO_SPK_OUT_EPT, EPT_ISO_TYPE, AUDIO_SPK_OUT_MAXPACKET_SIZE);
/* open speaker feedback endpoint */
usbd_ept_open(pudev, USBD_AUDIO_FEEDBACK_EPT, EPT_ISO_TYPE, AUDIO_FEEDBACK_MAXPACKET_SIZE);
/* start receive speaker out data */
usbd_ept_recv(pudev, USBD_AUDIO_SPK_OUT_EPT, audio_struct.audio_spk_data, AUDIO_SPK_OUT_MAXPACKET_SIZE);
```

- 端点清除（class_clear_handler）

```
/* close in endpoint */
usbd_ept_close(pudev, USBD_AUDIO_MIC_IN_EPT);
/* close in endpoint */
usbd_ept_close(pudev, USBD_AUDIO_FEEDBACK_EPT);
/* close out endpoint */
usbd_ept_close(pudev, USBD_AUDIO_SPK_OUT_EPT);
```

- Audio 控制请求（class_setup_handler）

实现如下 audio 控制请求：

请求	描述
GET_CUR	获取静音状态，音量状态
SET_CUR	设置静音和音量
GET_MIN	获取最小音量属性
GET_MAX	获取最大音量属性
GET_RES	获取音量分辨率属性

```
switch(setup->bRequest)
{
    case AUDIO_REQ_GET_CUR:
        audio_req_get_cur(pudev, setup);
        break;
    case AUDIO_REQ_SET_CUR:
        audio_req_set_cur(pudev, setup);
        break;
     case AUDIO_REQ_GET_MIN:
        audio_req_get_min(pudev, setup);
        break;
     case AUDIO_REQ_GET_MAX:
        audio_req_get_max(pudev, setup);
        break;
     case AUDIO_REQ_GET_RES:
        audio_req_get_res(pudev, setup);
```

```
            break;
        default:
            usbd_ctrl_unsupport(pudev);
            break;
}
```

- Audio 音量、静音、采样率设置（class_ept0_rx_handler）

  此函数用于接收完主机发送的设置数据之后进行处理，包括设置音量，静音，以及采样率的设置。

```
switch(audio_struct.request_no)
{
  case AUDIO_VOLUME_CONTROL:
  if(audio_struct.interface == AUDIO_SPK_FEATURE_UNIT_ID)
  {
    …
  }
  else
  {
    …
  }
  case AUDIO_MUTE_CONTROL:
  if(audio_struct.interface == AUDIO_SPK_FEATURE_UNIT_ID)
  {
    …
  }
  else
  {
    …
  }

  break;
  case AUDIO_FREQ_SET_CONTROL:
  if(audio_struct.enpd == USBD_AUDIO_MIC_IN_EPT)
  {
    …
  }
  else
  {
    …
  }
  break;
  default:
  break;
}
```

- Microphone 和 Feedback 数据传输（class_in_handler）

```
if((ept_num & 0x7F) == (USBD_AUDIO_MIC_IN_EPT & 0x7F)) //microphone
{
  len = audio_codec_mic_get_data(audio_struct.audio_mic_data);
  usbd_ept_send(pudev, USBD_AUDIO_MIC_IN_EPT, audio_struct.audio_mic_data, len);
}
else if((ept_num & 0x7F) == (USBD_AUDIO_FEEDBACK_EPT & 0x7F)) //feedback
{
  len = audio_codec_spk_feedback(audio_struct.audio_feed_back);
  usbd_ept_send(pudev, USBD_AUDIO_FEEDBACK_EPT, audio_struct.audio_feed_back, len);
}
```

- Speaker 数据接收（class_out_handler）

```
/* get endpoint receive data length  */
g_rxlen = usbd_get_recv_len(pudev, ept_num);

if((ept_num & 0x7F) == (USBD_AUDIO_SPK_OUT_EPT & 0x7F))
{
/* speaker data*/
audio_codec_spk_fifo_write(audio_struct.audio_spk_data, g_rxlen);

/* get next data */
usbd_ept_recv(pudev, USBD_AUDIO_SPK_OUT_EPT, audio_struct.audio_spk_data,
AUDIO_SPK_OUT_MAXPACKET_SIZE);
}
```

- audio_codec.c 中实现 codec 具体的控制以及数据处理，需要实现如下函数：

```
void audio_codec_spk_fifo_write(uint8_t *data, uint32_t len);
uint32_t audio_codec_mic_get_data(uint8_t *buffer);
uint8_t audio_codec_spk_feedback(uint8_t *feedback);
void audio_codec_spk_alt_setting(uint32_t alt_seting);
void audio_codec_mic_alt_setting(uint32_t alt_seting);
void audio_codec_set_mic_mute(uint8_t mute);
void audio_codec_set_spk_mute(uint8_t mute);
void audio_codec_set_mic_volume(uint16_t volume);
void audio_codec_set_spk_volume(uint16_t volume);
void audio_codec_set_mic_freq(uint32_t freq);
void audio_codec_set_spk_freq(uint32_t freq);
```

以上函数可根据例程里的方式进行实现，也可以根据开发者实际使用的 codec 来进行修改，对于 codec 的初始化部分，在这里将不再讲述。

- audio 例程功能配置

audio_conf.h 中可以对当前 audio 例程进行配置，例如：是否需要 speaker 功能，支持采样率等，有如下选项配置：

```
#define AUDIO_SUPPORT_SPK                1
#define AUDIO_SUPPORT_MIC                1
#define AUDIO_SUPPORT_FEEDBACK           0

#define AUDIO_SUPPORT_FREQ_16K           1
#define AUDIO_SUPPORT_FREQ_48K           1


#define AUDIO_SUPPORT_FREQ           (AUDIO_SUPPORT_FREQ_16K + \
                                      AUDIO_SUPPORT_FREQ_48K \
                                     )

#define AUDIO_FREQ_16K                16000
#define AUDIO_FREQ_48K                48000
#define AUDIO_BITW_16                 16

#define AUDIO_MIC_CHANEL_NUM          2
#define AUDIO_MIC_DEFAULT_BITW        AUDIO_BITW_16

#define AUDIO_SPK_CHANEL_NUM          2
#define AUDIO_SPK_DEFAULT_BITW        AUDIO_BITW_16


#define AUDIO_SUPPORT_MAX_FREQ        48
#define AUDIO_DEFAULT_FREQ            AUDIO_FREQ_48K
#define AUDIO_DEFAULT_BITW            AUDIO_BITW_16
```

3.1.4 如何根据 Audio 例程进行开发

本章将简单描述如何修改 audio 例程的代码来进行开发，根据应用具体的需求来修改代码。

- 根据功能需求修改 audio 配置（audio_conf.h）
- 根据功能需求修改设备描述信息（audio_desc.c, audio_desc.h）
  设备描述信息（g_usbd_descriptor）
  设备配置描述信息（g_usbd_configuration）
  其它描述
- 根据功能修改要使用端点(audio_class.c, audio_class.h)
  端点定义（audio_class.h）
  端点初始化（class_init_handler，class_clear_handler）
- 修改使用的 Audio 控制请求
  控制请求修改（class_setup_handler）
  控制请求设置处理（class_ept0_rx_handler）
- Audio 数据处理修改
  IN 数据处理（class_in_handler）
  OUT 数据处理（class_out_handler）
- 根据需求修改端点 FIFO 大小分配（usb_conf.h）

- 根据具体需求实现 codec 函数接口（audio_codec.c）

3.2 custom_hid 例程

custom_hid 实现一个 HID（human interface device）功能，与上位机（Artery_UsbHid_Demo）通信完成一些简单的交互操作，HID 使用中断传输与上位机通信，例程在 AT-START 开发板上运行，上位机可在官网下载，关于 HID 协议参考 《Human Interface Devices (HID) V1.11》。

3.2.1 实现功能

- 上位机显示按键状态
- 通过上位机控制开发板 LED 等开关状态
- HID 数据回环功能

3.2.2 外设资源使用

USB 外设：

- 端点 0 IN/OUT：用于 USB 枚举
- 端点 1 IN：用于数据发送
- 端点 1 OUT：用于数据接收

3.2.3 custom_hid 设备实现

custom_hid 设备类实现源代码主要在 custom_hid_class.c 和 custom_hid_desc.c 中，这两个源文件实现了对设备的描述和设备的处理。

3.2.3.1 设备描述：（custom_hid_desc.c/h）

- custom hid 设备描述（g_usbd_descriptor）
- custom hid 设备配置描述（g_usbd_configuration）
  HID interface
  HID Endpoint
- custom hid report 描述（g_usbd_hid_report）
  HID_REPORT_ID_2 (LED2)
  HID_REPORT_ID_3 (LED3)
  HID_REPORT_ID_4 (LED4)
  HID_REPORT_ID_5 (BUTTON)
  HID_REPORT_ID_6 (LOOPBACK DATA)
- Lang id （g_string_lang_id）
- 序列号（g_string_serial）
- 厂商产品 ID（custom_hid_desc.h）

```
#define USBD_VENDOR_ID                0x2E3C
#define USBD_PRODUCT_ID               0x5745
```

- 制造商、产品名、配置描述、接口描述（custom_hid_desc.h）

```
#define USBD_DESC_MANUFACTURER_STRING     "Artery"
#define USBD_DESC_PRODUCT_STRING          "Custom HID"
#define USBD_DESC_CONFIGURATION_STRING    "Custom HID Config"
#define USBD_DESC_INTERFACE_STRING        "Custom HID Interface"
```

3.3.3.2 数据处理部分（custom_hid_class.c/h）

- 端点初始化（class_init_handler）

```
/* open custom hid in endpoint */
usbd_ept_open(pudev, USBD_HID_IN_EPT, EPT_INT_TYPE, USBD_IN_MAXPACKET_SIZE);
/* open custom hid out endpoint */
usbd_ept_open(pudev, USBD_HID_OUT_EPT, EPT_INT_TYPE, USBD_OUT_MAXPACKET_SIZE);
/* set out endpoint to receive status */
usbd_ept_recv(pudev, USBD_HID_OUT_EPT, g_rxhid_buff, USBD_OUT_MAXPACKET_SIZE);
```

- 端点清除（class_clear_handler）

```
/* close custom hid in endpoint */
usbd_ept_close(pudev, USBD_HID_IN_EPT);
/* close custom hid out endpoint */
usbd_ept_close(pudev, USBD_HID_OUT_EPT);
```

- HID 设备类请求（class_setup_handler）

  实现如下请求：

  SET_PROTOCOL

  GET_PROTOCOL

  SET_IDLE

  GET_IDLE

  SET_REPORT

  代码如下：

```
switch(setup->bRequest)
{
  case HID_REQ_SET_PROTOCOL:
    hid_protocol = (uint8_t)setup->wValue;
    break;
  case HID_REQ_GET_PROTOCOL:
    usbd_ctrl_send(pudev, (uint8_t *)&hid_protocol, 1);
    break;
  case HID_REQ_SET_IDLE:
    hid_set_idle = (uint8_t)(setup->wValue >> 8);
    break;
  case HID_REQ_GET_IDLE:
    usbd_ctrl_send(pudev, (uint8_t *)&hid_set_idle, 1);
    break;
  case HID_REQ_SET_REPORT:
    hid_state = HID_REQ_SET_REPORT;
    usbd_ctrl_recv(pudev, hid_set_report, setup->wLength);
    break;
  default:
    usbd_ctrl_unsupport(pudev);
    break;
}
```

- Custom_HID 发送数据

```
usb_sts_type class_send_report(void *udev, uint8_t *report, uint16_t len)
{
    usb_sts_type status = USB_OK;
    usbd_core_type *pudev = (usbd_core_type *)udev;

    if(usbd_connect_state_get(pudev) == USB_CONN_STATE_CONFIGURED)
      usbd_ept_send(pudev, USBD_HID_IN_EPT, report, len);
    return status;
}
```

- Custom_HID 接收数据

```
/* get endpoint receive data length   */
uint32_t recv_len = usbd_get_recv_len(pudev, ept_num);
/* hid buffer process */
usb_hid_buf_process(udev, g_rxhid_buff, recv_len);
/* start receive next packet */
usbd_ept_recv(pudev, USBD_HID_OUT_EPT, g_rxhid_buff, recv_len);
```

- 数据处理

```
void usb_hid_buf_process(void *udev, uint8_t *report, uint16_t len)
{
  uint32_t i_index;
  usbd_core_type *pudev = (usbd_core_type *)udev;
  switch(report[0])
  {
    case HID_REPORT_ID_2:
      if(g_rxhid_buff[1] == 0)
      {
          at32_led_off(LED2);
       }
       else
       {
          at32_led_on(LED2);
       }
       break;
     case HID_REPORT_ID_3:
       if(g_rxhid_buff[1] == 0)
       {
          at32_led_off(LED3);
       }
       else
       {
          at32_led_on(LED3);
       }
```

```
            break;
        case HID_REPORT_ID_4:
            if(g_rxhid_buff[1] == 0)
            {
              at32_led_off(LED4);
            }
           else
           {
              at32_led_on(LED4);
            }
            break;
        case HID_REPORT_ID_6:
            for(i_index = 0; i_index < len; i_index ++)
            {
              g_txhid_buff[i_index] = report[i_index];
            }
            usbd_ept_send(pudev, USBD_HID_IN_EPT, g_txhid_buff, len);
        default:
            break;
    }
}
```

3.2.4 如何根据 custom hid 例程进行开发

本章将简单描述如何修改 custom_hid 例程的代码来进行开发，根据应用具体的需求来修改代码。

- 根据功能需求修改设备描述信息（custom_hid_desc.c, custom_hid_desc.h）
  设备描述信息（g_usbd_descriptor）
  设备配置描述信息（g_usbd_configuration）
  设备报告描述符（g_usbd_hid_report）
  其它描述
- 根据功能修改要使用端点(custom_hid_class.c, custom_hid_class.h)
  端点定义（custom_hid _class.h）
  端点初始化（class_init_handler，class_clear_handler）
- 修改使用的 custom_hid 控制请求
  控制请求修改（class_setup_handler）
  控制请求设置处理（class_ept0_rx_handler）
- custom_hid 发送接收数据处理修改
  IN 数据处理（class_in_handler）
  OUT 数据处理（class_out_handler）
- 根据需求修改端点 FIFO 大小分配（usb_conf.h）
- 修改数据处理部分

3.3 keyboard 例程

keyboard 实现一个键盘功能，使用中断传输与上位机通信，例程在 AT-START 开发板上运行，通过按键发送字符串到主机。

3.3.1 实现功能

- 通过按键发送字符串（” Keyboard Demo”）到主机

3.3.2 外设资源使用

USB 外设：

- 端点 0 IN/OUT：用于 USB 枚举
- 端点 1 IN：用于数据发送

3.3.3 keyboard 设备实现

keyboard 设备类实现源代码主要在 keyboard_class.c 和 keyboard_desc.c 中，这两个源文件实现了对设备的描述和设备的处理。

3.3.3.1 设备描述：（keyboard_desc.c/h）

- keyboard 设备描述（g_usbd_descriptor）
- keyboard 设备配置描述（g_usbd_configuration）
  keyboard interface
  keyboard endpoint
- keyboard report 描述（g_usbd_hid_report）
- Lang id （g_string_lang_id）
- 序列号（g_string_serial）
- 厂商产品 ID（keyboard_desc.h）

```
#define USBD_VENDOR_ID                0x2E3C
#define USBD_PRODUCT_ID               0x6040
```

- 制造商、产品名、配置描述、接口描述（keyboard_desc.h）

```
#define USBD_DESC_MANUFACTURER_STRING     "Artery"
#define USBD_DESC_PRODUCT_STRING          " Keyboard"
#define USBD_DESC_CONFIGURATION_STRING    " Keyboard Config"
#define USBD_DESC_INTERFACE_STRING        " Keyboard Interface"
```

3.3.3.2 数据处理部分（keyboard_class.c/h）

- 端点初始化（class_init_handler）

```
/* open hid in endpoint */
usbd_ept_open(pudev, USBD_HID_IN_EPT, EPT_INT_TYPE, USBD_IN_MAXPACKET_SIZE);
```

- 端点清除（class_clear_handler）

```
/* close hid in endpoint */
usbd_ept_close(pudev, USBD_HID_IN_EPT);
```

- HID 设备类请求（class_setup_handler）
  实现如下请求：
  SET_PROTOCOL
  GET_PROTOCOL

SET_IDLE
GET_IDLE
SET_REPORT

- keyboard 发送数据

```
usb_sts_type class_send_report(void *udev, uint8_t *report, uint16_t len)
{
    usb_sts_type status = USB_OK;
    usbd_core_type *pudev = (usbd_core_type *)udev;

    if(usbd_connect_state_get(pudev) == USB_CONN_STATE_CONFIGURED)
      usbd_ept_send(pudev, USBD_HID_IN_EPT, report, len);
    return status;
}
```

- keyboard 字符数据处理

```
void usb_hid_keyboard_send_char(void *udev, uint8_t ascii_code)
{
      uint8_t key_data = 0;
      static uint8_t temp = 0;
      static uint8_t keyboard_buf[8] = {0, 0, 6, 0, 0, 0, 0, 0};

      if(ascii_code > 128)
      {
        ascii_code = 0;
      }
      else
      {
        ascii_code = _asciimap[ascii_code];
        if(!ascii_code)
        {
          ascii_code = 0;
        }

        if(ascii_code & SHIFT)
        {
          key_data = 0x2;
          ascii_code &= 0x7F;
        }
      }
      if((temp == ascii_code) && (ascii_code != 0))
      {
        keyboard_buf[0] = 0;
        keyboard_buf[2] = 0;
        class_send_report(udev, keyboard_buf, 8);
      }
      else
```

```
        {
          keyboard_buf[0] = key_data;
          keyboard_buf[2] = ascii_code;
          class_send_report(udev, keyboard_buf, 8);
        }
}
```

3.3.4 如何根据 keyboard 例程进行开发

本章将简单描述如何修改 keyboard 例程的代码来进行开发，根据应用具体的需求来修改代码。

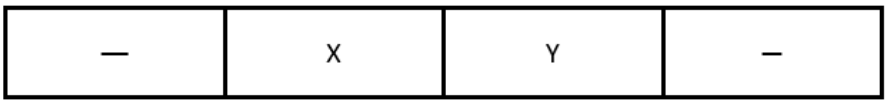
- 根据功能需求修改设备描述信息（keyboard_desc.c, keyboard_desc.h）
  设备描述信息（g_usbd_descriptor）
  设备配置描述信息（g_usbd_configuration）
  设备报告描述符（g_usbd_hid_report）
  其它描述
- 根据功能修改要使用端点(keyboard_class.c, keyboard_class.h)
  端点定义（keyboard _class.h）
  端点初始化（class_init_handler，class_clear_handler）
- 修改使用的 keyboard 控制请求
  控制请求修改（class_setup_handler）
  控制请求设置处理（class_ept0_rx_handler）
- keyboard 发送接收数据处理修改
  IN 数据处理（class_in_handler）
  OUT 数据处理（class_out_handler）
- 根据需求修改端点 FIFO 大小分配（usb_conf.h）
- 修改数据处理部分

3.4 mouse 例程

3.4.1 实现功能

mouse 实现一个简单的鼠标功能，使用中断传输与上位机通信，例程在 AT-START 开发板上运行，通过开发板上的按键发送鼠标右键功能。

图 16 鼠标传输格式



鼠标 d 通常是通过设置 X 和 Y 值来控制 PC 鼠标移动。

3.4.2 外设资源使用

USB 外设：

- 端点 0 IN/OUT：用于 USB 枚举
- 端点 1 IN：用于数据发送

3.4.3 mouse 设备实现

mouse 设备实现源代码主要在 mouse_class.c 和 mouse_desc.c 中，这两个源文件实现了对设备的描述和设备的处理。

3.4.3.1 设备描述：（mouse_desc.c/h）

- mouse 设备描述（g_usbd_descriptor）
- mouse 设备配置描述（g_usbd_configuration）
  mouseinterface
  mouseendpoint
- mouse report 描述（g_usbd_hid_report）
- Lang id （g_string_lang_id）
- 序列号（g_string_serial）
- 厂商产品 ID（mouse_desc.h）

```
#define USBD_VENDOR_ID                0x2E3C
#define USBD_PRODUCT_ID               0x5710
```

- 制造商、产品名、配置描述、接口描述（keyboard_desc.h）

```
#define USBD_DESC_MANUFACTURER_STRING     "Artery"
#define USBD_DESC_PRODUCT_STRING          " mouse"
#define USBD_DESC_CONFIGURATION_STRING    " mouse Config"
#define USBD_DESC_INTERFACE_STRING        " mouse Interface"
```

3.4.3.2 数据处理部分（mouse_class.c/h）

- 端点初始化（class_init_handler）

```
/* open hid in endpoint */
usbd_ept_open(pudev, USBD_HID_IN_EPT, EPT_INT_TYPE, USBD_IN_MAXPACKET_SIZE);
```

- 端点清除（class_clear_handler）

```
/* close hid in endpoint */
usbd_ept_close(pudev, USBD_HID_IN_EPT);
```

- HID 设备类请求（class_setup_handler）
  实现如下请求：
  SET_PROTOCOL
  GET_PROTOCOL
  SET_IDLE
  GET_IDLE
  SET_REPORT

- keyboard 发送数据

```
usb_sts_type class_send_report(void *udev, uint8_t *report, uint16_t len)
{
    usb_sts_type status = USB_OK;
    usbd_core_type *pudev = (usbd_core_type *)udev;

    if(usbd_connect_state_get(pudev) == USB_CONN_STATE_CONFIGURED)
      usbd_ept_send(pudev, USBD_HID_IN_EPT, report, len);
    return status;
}
```

- mouse 数据处理

```
void usb_hid_mouse_send(void *udev, uint8_t op)
{
    static uint8_t mouse_buffer[4] = {0, 0, 0, 0};
    int8_t posx = 0, posy = 0, button = 0;
    switch(op)
    {
      case LEFT_BUTTON:
        button = 0x01;
        break;

      case RIGHT_BUTTON:
        button = 0x2;
        break;

      case UP_MOVE:
        posy -= MOVE_STEP;
        break;

      case DOWN_MOVE:
        posy += MOVE_STEP;
        break;

      case LEFT_MOVE:
        posx -= MOVE_STEP;
        break;

      case RIGHT_MOVE:
        posx += MOVE_STEP;
        break;

      default:
        break;
    }
    mouse_buffer[0] = button;
```

```
        mouse_buffer[1] = posx;
        mouse_buffer[2] = posy;

        class_send_report(udev, mouse_buffer, 4);
}
```

3.4.4 如何根据 mouse 例程进行开发

本章将简单描述如何修改 mouse 例程的代码来进行开发，根据应用具体的需求来修改代码。

- 根据功能需求修改设备描述信息（mouse_desc.c, mouse_desc.h）
  设备描述信息（g_usbd_descriptor）
  设备配置描述信息（g_usbd_configuration）
  设备报告描述符（g_usbd_hid_report）
  其它描述
- 根据功能修改要使用端点(mouse_class.c, mouse_class.h)
  端点定义（mouse_class.h）
  端点初始化（class_init_handler，class_clear_handler）
- 修改使用的 mouse 控制请求
  控制请求修改（class_setup_handler）
  控制请求设置处理（class_ept0_rx_handler）
- mouse 发送接收数据处理修改
  IN 数据处理（class_in_handler）
  OUT 数据处理（class_out_handler）
- 根据需求修改端点 FIFO 大小分配（usb_conf.h）
- 修改数据处理部分

3.5 msc 例程

msc（mass storage）例程展示如何通过 USB BULK 传输，进行 PC 主机和 AT-START 通信，该例程支持 BOT(Bulk only transfer)协议和 SCSI（small computer system interface）指令。

图 17 BOT 命令/数据/状态 流程

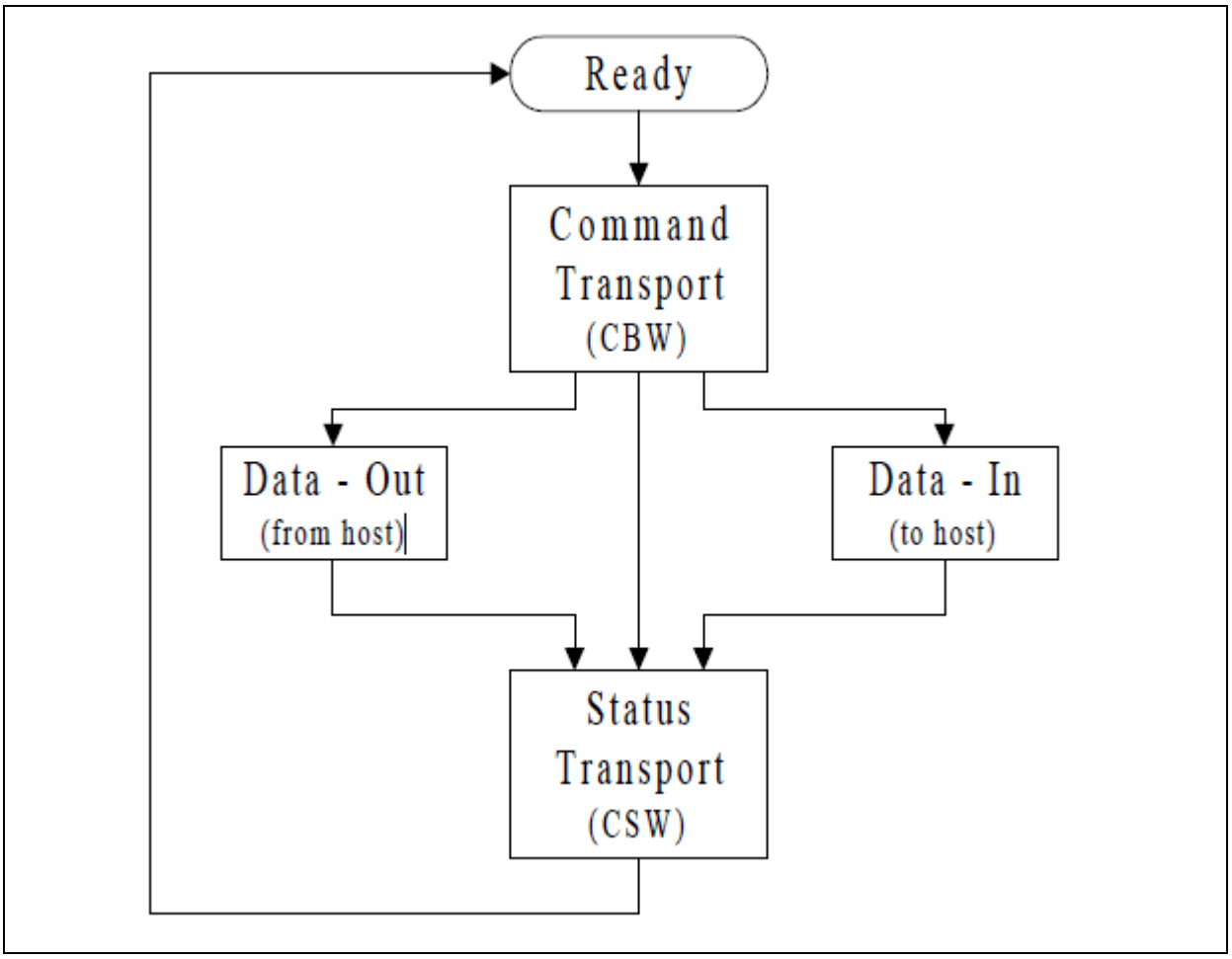


图 18 BOT 命令格式

bit / Byte	7	6	5	4	3	2	1	0
0-3	*dCBWSignature*							
4-7	*dCBWTag*							
8-11 (08h-0Bh)	*dCBWDataTransferLength*							
12 (0Ch)	*bmCBWFlags*							
13 (0Dh)	Reserved (0)					*bCBWLUN*		
14 (0Eh)	Reserved (0)					*bCBWCBLength*		
15-30 (0Fh-1Eh)	*CBWCB*							

图 19 BOT 状态格式

bit / Byte	7	6	5	4	3	2	1	0
0-3	*dCSWSignature*							
4-7	*dCSWTag*							
8-11 (8-Bh)	*dCSWDataResidue*							
12 (Ch)	*bCSWStatus*							

3.5.1 实现功能

- 将内部 FLASH 虚拟成一个磁盘
- 实现 bulk-only 传输协议
- 实现 subclass SCSI 传输命令

  MSC_CMD_INQUIRY
  MSC_CMD_START_STOP
  MSC_CMD_MODE_SENSE6
  MSC_CMD_MODE_SENSE10
  MSC_CMD_ALLOW_MEDIUM_REMOVAL
  MSC_CMD_READ_10
  MSC_CMD_READ_CAPACITY
  MSC_CMD_REQUEST_SENSE
  MSC_CMD_TEST_UNIT
  MSC_CMD_VERIFY
  MSC_CMD_WRITE_10
  MSC_CMD_READ_FORMAT_CAPACITY

3.5.2 外设资源使用

USB 外设：

- 端点 0 IN/OUT：用于 USB 枚举
- 端点 1 IN：用于数据发送
- 端点 1 OUT：用于数据接收

3.5.3 msc 设备实现

3.5.3.1 设备描述：（msc_desc.c/h）

- msc 设备描述（g_usbd_descriptor）
- msc 设备配置描述（g_usbd_configuration）

  msc interface

msc endpoint

- Lang id （g_string_lang_id）
- 序列号（g_string_serial）
- 厂商产品 ID（custom_hid_desc.h）

```
#define USBD_VENDOR_ID                  0x2E3C
#define USBD_PRODUCT_ID                 0x5745
```

- 制造商、产品名、配置描述、接口描述（msc_desc.h）

```
#define USBD_DESC_MANUFACTURER_STRING      "Artery"
#define USBD_DESC_PRODUCT_STRING           "AT32 Mass Storage"
#define USBD_DESC_CONFIGURATION_STRING     "Mass Storage Config"
#define USBD_DESC_INTERFACE_STRING         "Mass Storage Interface"
```

3.5.3.2 数据处理部分（msc_class.c/h）

- 端点初始化（class_init_handler）

```
/* open in endpoint */
usbd_ept_open(pudev, USBD_MSC_BULK_IN_EPT, EPT_BULK_TYPE, USBD_OUT_MAXPACKET_SIZE);
/* open out endpoint */
usbd_ept_open(pudev, USBD_MSC_BULK_OUT_EPT, EPT_BULK_TYPE,
USBD_OUT_MAXPACKET_SIZE);
/* set out endpoint to receive status */
usbd_ept_recv(pudev, USBD_MSC_BULK_OUT_EPT, (uint8_t *)&cbw_struct, CBW_CMD_LENGTH);
```

- 端点清除（class_clear_handler）

```
/* close in endpoint */
usbd_ept_close(pudev, USBD_MSC_BULK_IN_EPT);
/* close out endpoint */
usbd_ept_close(pudev, USBD_MSC_BULK_OUT_EPT);
```

- MSC 设备请求（class_setup_handler）

  GET_MAX_LUN

  BO_RESET

  代码如下：

```
switch(setup->bRequest)
 {
     case MSC_REQ_GET_MAX_LUN:
       usbd_ctrl_send(pudev, &msc_struct.max_lun, 1);
       break;
     case MSC_REQ_BO_RESET:
       usbd_ctrl_send_status(pudev);
       break;
     default:
       usbd_ctrl_unsupport(pudev);
       break;
 }
```

- IN 传输处理

```
usb_sts_type class_in_handler(void *udev, uint8_t ept_num)
{
  usb_sts_type status = USB_OK;

  bot_scsi_datain_handler(udev, ept_num);
  return status;
}
```

- OUT 传输处理（接收数据）

```
usb_sts_type class_out_handler(void *udev, uint8_t ept_num)
{
  usb_sts_type status = USB_OK;
  bot_scsi_dataout_handler(udev, ept_num);
  return status;
}
```

3.5.3.3 BOT 和 SCSI 命令处理

Bulk-only transfer 和 SCSI 的命令处理在库文件 msc_bot_scsi.c/h 中。

表 11 msc_bot_scsi 函数列表

函数名称	描述
bot_scsi_init	初始化 bulk-only 传输
bot_scsi_datain_handler	Bulk-only data in
bot_scsi_dataout_handler	Bulk-only data out
bot_scsi_cmd_process	SCSI 命令处理
bot_scsi_cmd_state	SCSI 命令状态
bot_scsi_test_unit	Test unit 命令
bot_scsi_inquiry	Inquiry 命令
bot_scsi_start_stop	Start stop 命令
bot_scsi_allow_medium_removal	Allow medium removal 命令
bot_scsi_mode_sense6	Mode sense6 命令
bot_scsi_mode_sense10	Mode sense10 命令
bot_scsi_read10	Read10 命令
bot_scsi_capacity	Capacity 命令
bot_scsi_format_capacity	Format capacity 命令
bot_scsi_request_sense	Request sense 命令
bot_scsi_verify	Verify 命令
bot_scsi_write10	Write10 命令
bot_scsi_unsupport	不支持的命令

3.5.3.4 diskio 处理

此部分主要处理与存储设备间接口，例程里面以内部 flash 的存储控制作为说明，msc_diskio.c/h 根据开发者使用存储不同，只需要实现对应存储的读写函数即可。

表 12 inquiry 描述

```
uint8_t scsi_inquiry[MSC_SUPPORT_MAX_LUN][SCSI_INQUIRY_DATA_LENGTH] =
{
  /* lun = 0 */
  {
    0x00,              /* peripheral device type (direct-access device) */
    0x80,              /* removable media bit */
    0x00,              /* ansi version, ecma version, iso version */
    0x01,              /* respond data format */
    SCSI_INQUIRY_DATA_LENGTH - 5, /* additional length */
    0x00, 0x00, 0x00, /* reserved */
    'A', 'T', '3', '2', ' ', ' ', ' ', ' ', /* vendor information "AT32" */
    'D', 'i', 's', 'k', '0', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', /* Product identification "Disk" */
    '2', '.', '0', '0'   /* product revision level */
  }
};
```

表 13 diskio 操作函数

函数名称	描述
get_inquiry	获取 inquiry 描述
msc_disk_read	从存储空间读数据
msc_disk_write	写函数到存储空间
msc_disk_capacity	获取存储容量

3.5.4 如何根据 msc 例程进行开发

本章将简单描述如何修改 msc 例程的代码来进行开发，根据应用具体的需求来修改代码。

- 根据功能需求修改设备描述信息（msc_desc.c, msc_desc.h）
  设备描述信息（g_usbd_descriptor）
  设备配置描述信息（g_usbd_configuration）
  其它描述
- 根据功能修改要使用端点(msc_class.c, msc_class.h)
  端点定义（msc_class.h）
  端点初始化（class_init_handler，class_clear_handler）
- 修改使用的 msc 控制请求
  控制请求修改（class_setup_handler）
  控制请求设置处理（class_ept0_rx_handler）
- msc 发送接收数据处理修改
  IN 数据处理（class_in_handler）
  OUT 数据处理（class_out_handler）
- 根据需求修改端点 FIFO 大小分配（usb_conf.h）
- 修改 diskio 部分,实现表 13 里面的函数（msc_diskio.c/h）

3.6 printer 例程

Printer 例程展示了使用 USB Device 作为打印机设备，此 demo 可在 PC 端识别到一个打印机设备并且可应答 PC 端发送的关于 printer 类的状态请求命令（例如打印机的有纸/无纸状态）

3.6.1 实现功能

- 实现一个打印机设备

3.6.2 外设资源使用

USB 外设：

- 端点 0 IN/OUT：用于 USB 枚举
- 端点 1 IN：用于数据发送
- 端点 1 OUT：用于数据接收

3.6.3 printer 设备实现

3.6.3.1 设备描述（printer_desc.c/h）

- printer 设备描述（g_usbd_descriptor）
- printer 设备配置描述（g_usbd_configuration）
  printer interface
  printer endpoint
- Lang id （g_string_lang_id）
- 序列号（g_string_serial）
- 厂商产品 ID（custom_hid_desc.h）

```
#define USBD_VENDOR_ID                0x2E3C
#define USBD_PRODUCT_ID               0x57FF
```

- 制造商、产品名、配置描述、接口描述（msc_desc.h）

```
#define USBD_DESC_MANUFACTURER_STRING     "Artery"
#define USBD_DESC_PRODUCT_STRING          "AT32 Printer"
#define USBD_DESC_CONFIGURATION_STRING    "Printer Config"
#define USBD_DESC_INTERFACE_STRING        "Printer Interface"
```

3.6.3.2 数据处理（printer_class.c/h）

- 端点初始化（class_init_handler）

```
/* open in endpoint */
usbd_ept_open(pudev, USBD_PRINTER_BULK_IN_EPT, EPT_BULK_TYPE,
USBD_OUT_MAXPACKET_SIZE);
/* open out endpoint */
usbd_ept_open(pudev, USBD_PRINTER_BULK_OUT_EPT, EPT_BULK_TYPE,
USBD_OUT_MAXPACKET_SIZE);
```

```
/* set out endpoint to receive status */
usbd_ept_recv(pudev, USBD_PRINTER_BULK_OUT_EPT, g_rx_buff, USBD_OUT_MAXPACKET_SIZE);
```

- 端点清除（class_clear_handler）

```
/* close in endpoint */
usbd_ept_close(pudev, USBD_PRINTER_BULK_IN_EPT);
/* close out endpoint */
usbd_ept_close(pudev, USBD_PRINTER_BULK_OUT_EPT);
```

- printer 设备请求（class_setup_handler）

  GET_DEVICE_ID

  PORT_STATUS

  SOFT_RESET

  代码如下：

```
switch(setup->bRequest)
{
  case PRINTER_REQ_GET_DEVICE_ID:
    usbd_ctrl_send(pudev, PRINTER_DEVICE_ID, PRINTER_DEVICE_ID_LEN);
    break;
  case PRINTER_REQ_GET_PORT_STATUS:
    usbd_ctrl_send(pudev, (uint8_t *)&g_printer_port_status, 1);
    break;
  case PRINTER_REQ_GET_SOFT_RESET:
    usbd_ctrl_recv(pudev, g_printer_data, USBD_OUT_MAXPACKET_SIZE);
    break;
  default:
    usbd_ctrl_unsupport(pudev);
    break;
}
```

- IN 传输处理

```
usb_sts_type class_in_handler(void *udev, uint8_t ept_num)
{
  usb_sts_type status = USB_OK;

  return status;
}
```

- OUT 传输处理（接收数据）

```
usb_sts_type class_out_handler(void *udev, uint8_t ept_num)
{
  usb_sts_type status = USB_OK;
  return status;
}
```

3.6.4 如何根据 printer 例程进行开发

本章将简单描述如何修改 printer 例程的代码来进行开发，根据应用具体的需求来修改代码。

- 根据功能需求修改设备描述信息（printer_desc.c, printer_desc.h）
  设备描述信息（g_usbd_descriptor）
  设备配置描述信息（g_usbd_configuration）
  其它描述
- 根据功能修改要使用端点(printer_class.c, printer_class.h)
  端点定义（printer_class.h）
  端点初始化（class_init_handler，class_clear_handler）
- 修改使用的 printer 控制请求
  控制请求修改（class_setup_handler）
  控制请求设置处理（class_ept0_rx_handler）
- printer 发送接收数据处理修改
  IN 数据处理（class_in_handler）
  OUT 数据处理（class_out_handler）
- 根据需求修改端点 FIFO 大小分配（usb_conf.h）

3.7 vcp_loopback 例程

在现代 PC 中，USB 是和绝大部分外设通信的标准接口。尽管如此，大部分的工业软件依旧使用 COM 接口（UART）通信。vcp_loopback 例程提供使用 USB 设备模拟 COM 接口方法，解决了该问题, vcp_loopback 例程展示了如何通过 CDC 协议进行 USB 数据收发。改例程需要使用虚拟串口驱动，可在官网下载。

3.7.1 实现功能

- 实现一个虚拟串口

3.7.2 外设资源使用

USB 外设：

- 端点 0 IN/OUT：用于 USB 枚举
- 端点 1 IN：用于数据发送
- 端点 1 OUT：用于数据接收
- 端点 2 IN：监控中断传输

3.7.3 vcp_loopback 设备实现

3.7.3.1 设备描述（cdc_desc.c/h）

- cdc 设备描述（g_usbd_descriptor）
- cdc 设备配置描述（g_usbd_configuration）
  cdc interface

cdc endpoint

- Lang id （g_string_lang_id）
- 序列号（g_string_serial）
- 厂商产品 ID（custom_hid_desc.h）

```
#define USBD_VENDOR_ID                  0x2E3C
#define USBD_PRODUCT_ID                 0x5740
```

- 制造商、产品名、配置描述、接口描述（msc_desc.h）

```
#define USBD_DESC_MANUFACTURER_STRING      "Artery"
#define USBD_DESC_PRODUCT_STRING           "AT32 Virtual Com Port"
#define USBD_DESC_CONFIGURATION_STRING     "Virtual ComPortConfig"
#define USBD_DESC_INTERFACE_STRING         "Virtual ComPort Interface"
```

3.7.3.2 数据处理（cdc_class.c/h）

- 端点初始化（class_init_handler）

```
/* open in endpoint */
usbd_ept_open(pudev, USBD_CDC_INT_EPT, EPT_INT_TYPE, USBD_IN_MAXPACKET_SIZE);
/* open in endpoint */
usbd_ept_open(pudev, USBD_CDC_BULK_IN_EPT, EPT_BULK_TYPE, USBD_OUT_MAXPACKET_SIZE);
/* open out endpoint */
usbd_ept_open(pudev, USBD_CDC_BULK_OUT_EPT, EPT_BULK_TYPE,
USBD_OUT_MAXPACKET_SIZE);
/* set out endpoint to receive status */
usbd_ept_recv(pudev, USBD_CDC_BULK_OUT_EPT, g_rx_buff, USBD_OUT_MAXPACKET_SIZE);
```

- 端点清除（class_clear_handler）

```
/* close in endpoint */
usbd_ept_close(pudev, USBD_CDC_INT_EPT);
/* close in endpoint */
usbd_ept_close(pudev, USBD_CDC_BULK_IN_EPT);
/* close out endpoint */
usbd_ept_close(pudev, USBD_CDC_BULK_OUT_EPT);
```

- cdc 设备请求（class_setup_handler）

  SET_LINE_CODING

  GET_LINE_CODING

  代码如下：

```
case USB_REQ_TYPE_CLASS:
      if(setup->wLength)
      {
        if(setup->bmRequestType & USB_REQ_DIR_DTH)
        {
          usb_vcp_cmd_process(udev, setup->bRequest, g_cmd, setup->wLength);
          usbd_ctrl_send(pudev, g_cmd, setup->wLength);
        }
        else
```

```
      {
        g_req = setup->bRequest;
        g_len = setup->wLength;
        usbd_ctrl_recv(pudev, g_cmd, g_len);

      }
    }

void usb_vcp_cmd_process(void *udev, uint8_t cmd, uint8_t *buff, uint16_t len)
{
  switch(cmd)
  {
    case SET_LINE_CODING:
      linecoding.bitrate = (uint32_t)(buff[0] | (buff[1] << 8) | (buff[2] << 16) | (buff[3] <<24));
      linecoding.format = buff[4];
      linecoding.parity = buff[5];
      linecoding.data = buff[6];
      break;

    case GET_LINE_CODING:
      buff[0] = (uint8_t)linecoding.bitrate;
      buff[1] = (uint8_t)(linecoding.bitrate >> 8);
      buff[2] = (uint8_t)(linecoding.bitrate >> 16);
      buff[3] = (uint8_t)(linecoding.bitrate >> 24);
      buff[4] = (uint8_t)(linecoding.format);
      buff[5] = (uint8_t)(linecoding.parity);
      buff[6] = (uint8_t)(linecoding.data);
      break;

    default:
      break;
  }
}
```

- IN 传输处理

```
usb_sts_type class_in_handler(void *udev, uint8_t ept_num)
{
  usbd_core_type *pudev = (usbd_core_type *)udev;
  usb_sts_type status = USB_OK;

  /* ...user code...
    trans next packet data
  */
  usbd_flush_tx_fifo(pudev, ept_num);
  g_tx_completed = 1;

  return status;
```

```
}

发送数据：
error_status usb_vcp_send_data(void *udev, uint8_t *send_data, uint16_t len)
{
  error_status status = SUCCESS;
  usbd_core_type *pudev = (usbd_core_type *)udev;
  if(g_tx_completed)
  {
    g_tx_completed = 0;
    usbd_ept_send(pudev, USBD_CDC_BULK_IN_EPT, send_data, len);
  }
  else
  {
    status = ERROR;
  }
  return status;
}
```

- OUT 传输处理（接收数据）

```
usb_sts_type class_out_handler(void *udev, uint8_t ept_num)
{
  usb_sts_type status = USB_OK;
  /* get endpoint receive data length  */
  g_rxlen = usbd_get_recv_len(pudev, ept_num);
  /*set recv flag*/
  g_rx_completed = 1;
  return status;
}

uint16_t usb_vcp_get_rxdata(void *udev, uint8_t *recv_data)
{
  uint16_t i_index = 0;
  uint16_t tmp_len = g_rxlen;
  usbd_core_type *pudev = (usbd_core_type *)udev;

  if(g_rx_completed == 0)
  {
    return 0;
  }
  g_rx_completed = 0;
  tmp_len = g_rxlen;
  for(i_index = 0; i_index < g_rxlen; i_index ++)
  {
    recv_data[i_index] = g_rx_buff[i_index];
  }
```

```
    usbd_ept_recv(pudev, USBD_CDC_BULK_OUT_EPT, g_rx_buff, USBD_OUT_MAXPACKET_SIZE);

    return tmp_len;
}
```

3.7.4 如何根据 vcp_loopback 例程进行开发

本章将简单描述如何修改 cdc 例程的代码来进行开发，根据应用具体的需求来修改代码。

- 根据功能需求修改设备描述信息（cdc_desc.c, cdc_desc.h）
  设备描述信息（g_usbd_descriptor）
  设备配置描述信息（g_usbd_configuration）
  其它描述
- 根据功能修改要使用端点(cdc_class.c, cdc_class.h)
  端点定义（cdc_class.h）
  端点初始化（class_init_handler，class_clear_handler）
- 修改使用的 cdc 控制请求
  控制请求修改（class_setup_handler）
  控制请求设置处理（class_ept0_rx_handler）
- cdc 发送接收数据处理修改
  IN 数据处理（class_in_handler）
  OUT 数据处理（class_out_handler）
- 根据需求修改端点 FIFO 大小分配（usb_conf.h）

3.8 virtual_msc_iap 例程

virtual msc iap 实现一个升级功能的设备，不依赖上位机，当接入 PC 之后，通过将固件拷贝到磁盘以达到升级目的。

3.8.1 实现功能

- 将 flash 虚拟成磁盘进行升级
- Iap 保留使用 20K byte 空间
- 升级完成之后 reset usb 设备返回升级状态
- 支持下载地址设置
- 支持升级完成之后跳转到 APP 运行
- 支持 bin 文件升级

3.8.2 外设资源使用

USB 外设：

- 端点 0 IN/OUT：用于 USB 枚举
- 端点 1 IN：用于数据发送
- 端点 1 OUT：用于数据接收

3.8.3 virtual_msc_iap 设备实现

3.8.3.1 设备描述（msc_desc.c/h）

参考 3.5.3.1

3.8.3.2 数据处理部分（msc_class.c/h）

参考 3.5.3.2

3.8.3.3 BOT 和 SCSI 命令处理

参考 3.5.3.3

3.8.3.4 diskio 处理

参考 3.5.3.4

3.8.3.5 flash 升级部分（flash_fat16.c/h）

- 升级状态

```
typedef enum
{
  UPGRADE_READY                                = 0,
  UPGRAGE_ONGOING,
  UPGRADE_SUCCESS,
  UPGRADE_ERROR,
  UPGRADE_LARGE,
  UPGRADE_UNKNOWN,
  UPGRADE_DONE,
  UPGRADE_JUMP
}upgrade_status_type;
```

当连接 Host 之后，在响应磁盘上通过 TXT 文档显示当前状态
准备升级状态（Ready.TXT）
升级成功（Success.TXT）
升级失败（Failed.TXT）
未知文件或错误（Unkonwn.TXT）
升级文件大于 FLASH 大小（Large.TXT）

- FAT16 分区表描述

```
static uint8_t fat16_sector[FAT16_SECTOR_SIZE] =
{
        0xEB,                                   /*00 - BS_jmpBoot */
```

```
        0x3C,                                   /*01 - BS_jmpBoot */
        0x90,                                   /*02 - BS_jmpBoot */
        'M','S','D','O','S','5','.','0',        /* 03-10 - BS_OEMName */
        0x00,                                   /*11 - BPB_BytesPerSec = 2048 */
        0x08,                                   /*11 - BPB_BytesPerSec = 2048 */
        0x04,                                   /*13 - BPB_Sec_PerClus = 2K*4 = 8K*/
        2,                                      /*14 - BPB_RsvdSecCnt = 2 */
        0,                                      /*15 - BPB_RsvdSecCnt = 2 */
        2,                                      /*16 - BPB_NumFATs = 2 */
        0x0,                                    /*17 - BPB_RootEntCnt = 512 */
        0x2,                                    /*18 - BPB_RootEntCnt = 512 */
        0,                                      /*19 - BPB_TotSec16 = 0 */
        0,                                      /*20 - BPB_TotSec16 = 0 */
        0xF8,                                   /*21 - BPB_Media = 0xF8 */
        0x0D,                                   /*22 - BPBFATSz16 = 0x000D */
        0,                                      /*23 - BPBFATSz16 = 0x000D */
        0x3F,                                   /*24 - BPB_SecPerTrk = 0x003F */
        0,                                      /*25 - BPB_SecPerTrk = 0x003F */
        0xFF,                                   /*26 - BPB_NumHeads = 255 */
        0,                                      /*27 - BPB_NumHeads = 255 */
        0,                                      /*28 - BPB_HiddSec = 0 */
        0,                                      /*29 - BPB_HiddSec = 0 */
        0,                                      /*30 - BPB_HiddSec = 0 */
        0,                                      /*31 - BPB_HiddSec = 0 */
        0x00,                                   /*32 - BPB_TotSec32 = */
        0xC8,                                   /*33 - BPB_TotSec32 = 0x0000C800 100Mb*/
        0x00,                                   /*34 - BPB_TotSec32 = */
        0x00,                                   /*35 - BPB_TotSec32 = */
        0x80,                                   /*36 - BS_DrvNum = 0x80 */
        0,                                      /*37 - BS_Reserved1 = 0 , dirty bit = 0*/ /* Updated from
FSL*/
        0x29,                                   /*38 - BS_BootSig = 0x29 */
        0xBD,                                   /*39 - BS_VolID = 0x02DDA5BD */
        0xA5,                                   /*40 - BS_VolID = 0x02DDA5BD */
        0xDD,                                   /*41 - BS_VolID = 0x02DDA5BD */
        0x02,                                   /*42 - BS_VolID = 0x02DDA5BD */
        'N','O',' ','N','A','M','E',' ',' ',' ',' ',/*43-53 - BS_VolLab */
        'F','A','T','1','6',' ',' ',' '         /*54-61 - BS_FilSysType */
};


const uint8_t fat16_root_dir_sector[FAT16_DIR_SIZE]=
{
        0x20,                                   /*11 - Archive Attribute set */
        0x00,                                   /*12 - Reserved */
        0x4B,                                   /*13 - Create Time Tenth */
```

```
        0x9C,                                   /*14 - Create Time */
        0x42,                                   /*15 - Create Time */
        0x92,                                   /*16 - Create Date */
        0x38,                                   /*17 - Create Date */
        0x92,                                   /*18 - Last Access Date */
        0x38,                                   /*19 - Last Access Date */
        0x00,                                   /*20 - Not used in FAT16 */
        0x00,                                   /*21 - Not used in FAT16 */
        0x9D,                                   /*22 - Write Time */
        0x42,                                   /*23 - Write Time */
        0x92,                                   /*24 - Write Date */
        0x38,                                   /*25 - Write Date */
        0x00,                                   /*26 - First Cluster (none, because file is empty) */
        0x00,                                   /*27 - First Cluster (none, because file is empty) */
        0x00,                                   /*28 - File Size */
        0x00,                                   /*29 - File Size */
        0x00,                                   /*30 - File Size */
        0x00,                                   /*31 - File Size */
        'A','T','3','2',' ','I','A','P',' ',' ',' ',    /*32-42 - Volume label */
        0x08,                                   /*43 - File attribute = Volume label */
        0x00,                                   /*44 - Reserved */
        0x00,                                   /*45 - Create Time Tenth */
        0x00,                                   /*46 - Create Time */
        0x00,                                   /*47 - Create Time */
        0x00,                                   /*48 - Create Date */
        0x00,                                   /*49 - Create Date */
        0x00,                                   /*50 - Last Access Date */
        0x00,                                   /*51 - Last Access Date */
        0x00,                                   /*52 - Not used in FAT16 */
        0x00,                                   /*53 - Not used in FAT16 */
        0x9D,                                   /*54 - Write Time */
        0x42,                                   /*55 - Write Time */
        0x92,                                   /*56 - Write Date */
        0x38,                                   /*57 - Write Date */
};
```

- 升级接口函数

函数名称	描述
flash_fat16_init	初始化
flash_fat16_loop_status	升级状态监测
flash_fat16_set_name	修改状态对应的 TXT 文档名称
flash_fat16_get_upgrade_flag	获取升级状态
flash_fat16_write	写 flash 数据
flash_fat16_read	读 flash 数据

3.8.4 如何根据 virtual_msc_iap 例程进行开发

本章将简单描述如何修改 virtual_msc_iap 例程的代码来进行开发，根据应用具体的需求来修改代码。

- 根据功能需求修改设备描述信息（msc_desc.c, msc_desc.h）
  设备描述信息（g_usbd_descriptor）
  设备配置描述信息（g_usbd_configuration）
  其它描述
- 根据功能修改要使用端点(msc_class.c, msc_class.h)
  端点定义（msc_class.h）
  端点初始化（class_init_handler，class_clear_handler）
- 修改使用的 msc 控制请求
  控制请求修改（class_setup_handler）
  控制请求设置处理（class_ept0_rx_handler）
- msc 发送接收数据处理修改
  IN 数据处理（class_in_handler）
  OUT 数据处理（class_out_handler）
- 根据需求修改端点 FIFO 大小分配（usb_conf.h）
- 修改 diskio 部分,实现表 13 里面的函数（msc_diskio.c/h）
- 修改 flash_fat16.c/h 升级参数，包括 APP 起始地址，IAP 占用空间等，要保证 IAP 和 APP 地址不重合。

3.9 composite_vcp_keyboard 例程

复合设备定义如下：拥有多个相互独立接口的设备被称为复合设备。
当使用该设备时，该设备上拥有多个组合的功能。例如，Composite vcp keyboard demo 提供的复合设备包含 HID 和 CDC 功能（键盘和串口通信）

3.9.1 实现功能

- 实现一个 USB 虚拟串口（参考 3.7）
- 实现一个 USB 键盘设备（参考 3.3）

3.9.2 外设资源使用

USB 外设：

- 端点 0 IN/OUT：用于 USB 枚举
- 端点 1 IN：用于数据发送
- 端点 1 OUT：用于数据接收
- 端点 2 IN：CDC 命令中断传输
- 端点 3 IN：Keyboard 发送数据

3.9.3 composite_vcp_keyboard 设备实现

3.9.3.1 设备描述（cdc_keyboard_desc.c/h）

- cdc_keyboard 设备描述（g_usbd_descriptor）
- cdc_keyboard 设备配置描述（g_usbd_configuration）
  cdc interface
  cdc endpoint
  keyboard interface
  keyboard endpoint
- Lang id （g_string_lang_id）
- 序列号（g_string_serial）
- 厂商产品 ID（custom_hid_desc.h）

```
#define USBD_VENDOR_ID                0x2E3C
#define USBD_PRODUCT_ID               0x5750
```

- 制造商、产品名、配置描述、接口描述（msc_desc.h）

```
#define USBD_DESC_MANUFACTURER_STRING      "Artery"
#define USBD_DESC_PRODUCT_STRING           "AT32 Composite VCP and Keyboard"
#define USBD_DESC_CONFIGURATION_STRING     "Composite VCP and Keyboard Config"
#define USBD_DESC_INTERFACE_STRING         "Composite VCP and Keyboard Interface"
```

3.9.3.2 数据处理（cdc_keyboard_class.c/h）

- 端点初始化（class_init_handler）

```
/* open in endpoint */
usbd_ept_open(pudev, USBD_CDC_INT_EPT, EPT_INT_TYPE, USBD_IN_MAXPACKET_SIZE);
/* open in endpoint */
usbd_ept_open(pudev, USBD_CDC_BULK_IN_EPT, EPT_BULK_TYPE, USBD_OUT_MAXPACKET_SIZE);
/* open out endpoint */
usbd_ept_open(pudev, USBD_CDC_BULK_OUT_EPT, EPT_BULK_TYPE,
USBD_OUT_MAXPACKET_SIZE);
/* open hid in endpoint */
usbd_ept_open(pudev, USBD_HID_IN_EPT, EPT_INT_TYPE, USBD_OUT_MAXPACKET_SIZE);
/* set out endpoint to receive status */
usbd_ept_recv(pudev, USBD_CDC_BULK_OUT_EPT, g_rx_buff, USBD_OUT_MAXPACKET_SIZE);
```

- 端点清除（class_clear_handler）

```
/* close in endpoint */
usbd_ept_close(pudev, USBD_CDC_INT_EPT);
/* close in endpoint */
usbd_ept_close(pudev, USBD_CDC_BULK_IN_EPT);
/* close out endpoint */
usbd_ept_close(pudev, USBD_CDC_BULK_OUT_EPT);
/* close in endpoint */
```

```
usbd_ept_close(pudev, USBD_HID_IN_EPT);
```

- 设备请求（class_setup_handler）

cdc 设备类请求:

SET_LINE_CODING

GET_LINE_CODING

Keyboard hid 设备类请求:

SET_PROTOCOL

GET_PROTOCOL

SET_IDLE

GET_IDLE

SET_REPORT

代码如下：

```
switch(setup->bmRequestType & USB_REQ_RECIPIENT_MASK)
{
    case USB_REQ_RECIPIENT_INTERFACE:
      if(setup->wIndex == HID_KEYBOARD_INTERFACE)
      {
        keyboard_class_setup_handler(udev, setup);
      }
      else
      {
        cdc_class_setup_handler(pudev, setup);
      }
      break;
    case USB_REQ_RECIPIENT_ENDPOINT:
      break;

}
```

- IN 传输处理

```
usb_sts_type class_in_handler(void *udev, uint8_t ept_num)
{
  usb_sts_type status = USB_OK;

  /* ...user code...
    trans next packet data
  */
  if((ept_num & 0x7F) == (USBD_CDC_BULK_IN_EPT & 0x7F))
  {
    g_tx_completed = 1;
  }
  if((ept_num & 0x7F) == (USBD_HID_IN_EPT & 0x7F))
  {
    g_keyboard_tx_completed = 1;
```

```
  }
  return status;
}
```

CDC 发送数据：

```
error_status usb_vcp_send_data(void *udev, uint8_t *send_data, uint16_t len)
{
  error_status status = SUCCESS;
  usbd_core_type *pudev = (usbd_core_type *)udev;
  if(g_tx_completed)
  {
    g_tx_completed = 0;
    usbd_ept_send(pudev, USBD_CDC_BULK_IN_EPT, send_data, len);
  }
  else
  {
    status = ERROR;
  }
  return status;
}
```

Keyboard 发送数据：

```
usb_sts_type class_send_report(void *udev, uint8_t *report, uint16_t len)
{
  usb_sts_type status = USB_OK;
  usbd_core_type *pudev = (usbd_core_type *)udev;

  if(usbd_connect_state_get(pudev) == USB_CONN_STATE_CONFIGURED)
    usbd_ept_send(pudev, USBD_HID_IN_EPT, report, len);
  return status;
}
```

- OUT 传输处理（接收数据）

```
usb_sts_type class_out_handler(void *udev, uint8_t ept_num)
{
  usb_sts_type status = USB_OK;
  /* get endpoint receive data length  */
  g_rxlen = usbd_get_recv_len(pudev, ept_num);
  /*set recv flag*/
  g_rx_completed = 1;
  return status;
}

uint16_t usb_vcp_get_rxdata(void *udev, uint8_t *recv_data)
{
```

```
    uint16_t i_index = 0;
    uint16_t tmp_len = g_rxlen;
    usbd_core_type *pudev = (usbd_core_type *)udev;

    if(g_rx_completed == 0)
    {
      return 0;
    }
    g_rx_completed = 0;
    tmp_len = g_rxlen;
    for(i_index = 0; i_index < g_rxlen; i_index ++)
    {
      recv_data[i_index] = g_rx_buff[i_index];
    }

     usbd_ept_recv(pudev, USBD_CDC_BULK_OUT_EPT, g_rx_buff, USBD_OUT_MAXPACKET_SIZE);

    return tmp_len;
}
```

3.9.4 如何根据 composite_vcp_keyboard 例程进行开发

本章将简单描述如何修改 composite_vcp_keyboard 例程的代码来进行复合设备开发，根据应用具体的需求来修改代码。

- 根据功能需求修改设备描述信息（cdc_keyboard_desc.c, cdc_keyboard _desc.h）
  设备描述信息（g_usbd_descriptor）
  设备配置描述信息（g_usbd_configuration）
  其它描述
- 根据功能修改要使用端点(cdc_keyboard _class.c, cdc_keyboard _class.h)
  端点定义（cdc_class.h）
  端点初始化（class_init_handler，class_clear_handler）
- 修改使用的 cdc 控制请求
  控制请求修改（class_setup_handler）
  控制请求设置处理（class_ept0_rx_handler）
- cdc_keyboard 发送接收数据处理修改
  IN 数据处理（class_in_handler）
  OUT 数据处理（class_out_handler）
- 根据需求修改端点 FIFO 大小分配（usb_conf.h）

3.10 hid_iap 例程

hid_iap 例程使用 usb hid 实现 IAP 升级功能，需要搭配上位机使用，上位机可在官网下载 IAP_Programmer。hid iap 的例程代码位于 BSP 固件库 utilities\at32f435_437_usb_iap_demo 中，使用方法可参考《AN0007_AT32_IAP_using_the_USB_HID_ZH_V2.x.x.pdf》

3.10.1 实现功能

- 实现使用 HID 进行设备升级

3.10.2 外设资源使用

USB 外设：

- 端点 0 IN/OUT：用于 USB 枚举
- 端点 1 IN：用于数据发送
- 端点 1 OUT：用于数据接收

3.10.3 hid_iap 设备实现

3.10.3.1 设备描述（hid_iap_desc.c/h）

- hid iap 设备描述（g_usbd_descriptor）
- hid iap 设备配置描述（g_usbd_configuration）
  HID interface
  HID Endpoint
- hid iap report 描述（g_usbd_hid_report）
- Lang id （g_string_lang_id）
- 序列号（g_string_serial）
- 厂商产品 ID（hid_iap_desc.h）

```
#define USBD_VENDOR_ID                0x2E3C
#define USBD_PRODUCT_ID               0xAF01
```

- 制造商、产品名、配置描述、接口描述（hid_iap_desc.h）

```
#define USBD_DESC_MANUFACTURER_STRING     "Artery"
#define USBD_DESC_PRODUCT_STRING          "Custom HID"
#define USBD_DESC_CONFIGURATION_STRING    "Custom HID Config"
#define USBD_DESC_INTERFACE_STRING        "Custom HID Interface"
```

3.10.3.2 数据处理（hid_iap_class.c/h）

- 端点初始化（class_init_handler）

```
/* open custom hid in endpoint */
usbd_ept_open(pudev, USBD_HID_IN_EPT, EPT_INT_TYPE, USBD_IN_MAXPACKET_SIZE);
/* open custom hid out endpoint */
usbd_ept_open(pudev, USBD_HID_OUT_EPT, EPT_INT_TYPE, USBD_OUT_MAXPACKET_SIZE);
/* set out endpoint to receive status */
usbd_ept_recv(pudev, USBD_HID_OUT_EPT, g_rxhid_buff, USBD_OUT_MAXPACKET_SIZE);
```

- 端点清除（class_clear_handler）

```
/* close custom hid in endpoint */
usbd_ept_close(pudev, USBD_HID_IN_EPT);
/* close custom hid out endpoint */
```

```
usbd_ept_close(pudev, USBD_HID_OUT_EPT);
```

- HID 设备类请求（class_setup_handler）

实现如下请求：

SET_PROTOCOL

GET_PROTOCOL

SET_IDLE

GET_IDLE

SET_REPORT

代码如下：

```
switch(setup->bRequest)
{
  case HID_REQ_SET_PROTOCOL:
    hid_protocol = (uint8_t)setup->wValue;
    break;
  case HID_REQ_GET_PROTOCOL:
    usbd_ctrl_send(pudev, (uint8_t *)&hid_protocol, 1);
    break;
  case HID_REQ_SET_IDLE:
    hid_set_idle = (uint8_t)(setup->wValue >> 8);
    break;
  case HID_REQ_GET_IDLE:
    usbd_ctrl_send(pudev, (uint8_t *)&hid_set_idle, 1);
    break;
  case HID_REQ_SET_REPORT:
    hid_state = HID_REQ_SET_REPORT;
    usbd_ctrl_recv(pudev, hid_set_report, setup->wLength);
    break;
  default:
    usbd_ctrl_unsupport(pudev);
    break;
}
```

- hid iap 发送数据

发送完成处理：

```
usb_sts_type class_in_handler(void *udev, uint8_t ept_num)
{
  usb_sts_type status = USB_OK;

  /* ...user code...
    trans next packet data
  */
  usbd_hid_iap_in_complete(udev);

  return status;
}
```

发送数据：

```
usb_sts_type class_send_report(void *udev, uint8_t *report, uint16_t len)
{
    usb_sts_type status = USB_OK;
    usbd_core_type *pudev = (usbd_core_type *)udev;

    if(usbd_connect_state_get(pudev) == USB_CONN_STATE_CONFIGURED)
      usbd_ept_send(pudev, USBD_HID_IN_EPT, report, len);
    return status;
}
```

- hid iap 接收数据

```
usb_sts_type class_out_handler(void *udev, uint8_t ept_num)
{
  usb_sts_type status = USB_OK;
  usbd_core_type *pudev = (usbd_core_type *)udev;

  /* get endpoint receive data length  */
  uint32_t recv_len = usbd_get_recv_len(pudev, ept_num);

  /* hid iap process */
  usbd_hid_iap_process(udev, g_rxhid_buff, recv_len);

  /* start receive next packet */
  usbd_ept_recv(pudev, USBD_HID_OUT_EPT, g_rxhid_buff, USBD_OUT_MAXPACKET_SIZE);

  return status;
}
```

- 升级命令数据处理

```
iap_result_type usbd_hid_iap_process(void *udev, uint8_t *pdata, uint16_t len)
{
  iap_result_type status = IAP_SUCCESS;
  uint16_t iap_cmd;

  if(len < 2)
  {
    return IAP_FAILED;
  }

  iap_info.respond_flag = 0;

  iap_cmd = (pdata[0] << 8) | pdata[1];

  switch(iap_cmd)
  {
```

```
    case IAP_CMD_IDLE:
      iap_idle();
      break;
    case IAP_CMD_START:
      iap_start();
      break;
    case IAP_CMD_ADDR:
      iap_address(pdata, len);
      break;
    case IAP_CMD_DATA:
      iap_data_write(pdata, len);
      break;
    case IAP_CMD_FINISH:
      iap_finish();
      break;
    case IAP_CMD_CRC:
      iap_crc(pdata, len);
      break;
    case IAP_CMD_JMP:
      iap_jump();
      break;
    case IAP_CMD_GET:
      iap_get();
      break;
    default:
      status = IAP_FAILED;
      break;
  }

  if(iap_info.respond_flag)
  {
    class_send_report(udev, iap_info.iap_tx, 64);
  }
  return status;
}
```

3.10.3.3 hid iap 升级协议

表 14 hid iap 升级命令

命令值	描述
0x5AA0	进入 IAP 模式
0x5AA1	开始下载
0x5AA2	设置地址(按 1K 对齐)
0x5AA3	下载数据命令

命令值	描述
0x5AA4	下载结束
0x5AA5	CRC 校验
0x5AA6	跳转命令（跳转到用户 code）
0x5AA7	获取 IAP 设置的 User Code 地址

1. 0x5AA0进入IAP模式

   作为一个特定的命令，当用户APP收到这个命令之后将进入IAP模式。实现方式为收到这个命令之后擦除flag然后reset

   上位机：[0x5A, 0xA0]

   IAP设备响应：[0x5A, 0XA0, ACK/NACK]

2. 0x5AA1 开始下载

   上位机：[0x5A,0xA1]

   IAP设备响应: [0x5A,0xA1,ACK/NACK]

3. 0x5AA2设置下载地址

   设置下载地址需按照1KB对齐，每下载1Kbyte数据之后，都需要重新设置下载地址。

   上位机(命令+地址)：[0x5A, 0xA2, 0x08, 0x00, 0x40, 0x00]

   IAP设备响应：[0x5A,0xA2, ACK/NACK]

4. 0x5AA3 下载数据命令(1KB 对齐多个包发送)

   下载数据命令采用 命令+长度+数据的格式进行发送，每包最大数据量为60Byte （64 - 命令 - 长度），当发送数据达到1KB时，上位机需要等待设备的ACK响应。此时设备需将1KB的数据写到FLASH。

   上位机（命令(2Byte)+长度(2 Byte)+数据(n byte)）:[0x5A,0xA3,LEN1, LEN0,DATA0….DATAn]

   收完1KB数据后IAP设备响应：[0x5A, 0XA3, ACK/NACK]

5. 0x5AA4 下载结束

   上位机：[0x5A, 0xA4]

   IAP设备响应：[0x5A, 0xA4, ACK/NACK]

6. 0x5AA5 固件CRC校验

   上位机传输固件起始地址和固件大小/1KB（固件大小按1KB对齐，不足补0xFF），由IAP计算CRC之后返回给上位机。

   上位机：[0x5A,0xA5, 0x08, 0x00, 0x40, 0x00, LEN1, LEN0]

   IAP设备响应: [0x5A, 0xA5, ACK/NACK, CRC3, CRC2, CRC1, CRC0]

7. 0x5AA6 跳转命令

   跳转命令将跳转到用户代码进行运行

   上位机：[0x5A,0xA6, 0x08, 0x00, 0x40, 0x00]

   IAP设备响应: [0x5A,0xA6,ACK/NACK]

8. 0x5AA7 获取IAP设置的app地址

   返回IAP设置的app地址

上位机：[0x5A, 0xA7]

IAP 设备响应：[0x5A, 0xA7, ACK/NACK, 0x08, 0x00, 0x40, 0x00]

3.10.4 如何根据 hid_iap 例程进行开发

本章将简单描述如何修改 hid_iap 例程的代码来进行开发，根据应用具体的需求来修改代码。

- 根据功能需求修改设备描述信息（hid_iap_desc.c, hid_iap_desc.h）
  设备描述信息（g_usbd_descriptor）
  设备配置描述信息（g_usbd_configuration）
  其它描述
- 根据功能修改要使用端点(hid_iap_class.c, hid_iap_class.h)
  端点定义（hid_iap_class.h）
  端点初始化（class_init_handler，class_clear_handler）
- 修改使用的 hid 控制请求
  控制请求修改（class_setup_handler）
  控制请求设置处理（class_ept0_rx_handler）
- hid_iap 发送接收数据处理修改
  IN 数据处理（class_in_handler）
  OUT 数据处理（class_out_handler）
- 根据需求修改端点 FIFO 大小分配（usb_conf.h）
- 修改 hid_iap_user.h 中的升级参数，包括 APP 起始地址等，IAP 占用空间等，保证 APP 地址和 IAP 的地址不要重合。

4 版本历史

表 15. 文档版本历史

日期	版本	变更
2021.11.05	2.0.0	最初版本

重要通知 - 请仔细阅读

买方自行负责对本文所述雅特力产品和服务的选择和使用，雅特力概不承担与选择或使用本文所述雅特力产品和服务相关的任何责任。

无论之前是否有过任何形式的表示，本文档不以任何方式对任何知识产权进行任何明示或默示的授权或许可。如果本文档任何部分涉及任何第三方产品或服务，不应被视为雅特力授权使用此类第三方产品或服务，或许可其中的任何知识产权，或者被视为涉及以任何方式使用任何此类第三方产品或服务或其中任何知识产权的保证。

除非在雅特力的销售条款中另有说明，否则，雅特力对雅特力产品的使用和/或销售不做任何明示或默示的保证，包括但不限于有关适销性、适合特定用途(及其依据任何司法管辖区的法律的对应情况)，或侵犯任何专利、版权或其他知识产权的默示保证。

雅特力的产品不得应用于武器。此外，雅特力产品也不是为下列用途而设计并不得应用于下列用途：(A) 对安全性有特别要求的应用，例如：生命支持、主动植入设备或对产品功能安全有要求的系统；(B) 航空应用；(C) 汽车应用或汽车环境，且/或(D) 航天应用或航天环境。如果雅特力产品不是为前述应用设计的，而采购商擅自将其用于前述应用，即使采购商向雅特力发出了书面通知，采购商仍将独自承担因此而导致的任何风险，雅特力的产品设计规格明确指定的汽车、汽车安全或医疗工业领域专用产品除外。根据相关政府主管部门的规定，ESCC、QML 或 JAN 正式认证产品适用于航天应用。

经销的雅特力产品如有不同于本文档中提出的声明和/或技术特点的规定，将立即导致雅特力针对本文所述雅特力产品或服务授予的任何保证失效，并且不应以任何形式造成或扩大雅特力的任何责任。