文档编号：AN2010

上海东软载波微电子有限公司

用户手册

ES32 USB 协议栈例程

修订历史

版本	修订日期	修改概要
V1.0	2019-3-11	初版
V1.1	2019-7-9	1. 修改 HID 自定义设备例程 2. 增加 HID Bootloader 例程 3. 修改 Audio 设备流程 4. 增加 MSC 设备例程 5. 增加 MSC 主机例程 6. 增加 HID Bootloader 自定义通信协议

地　址：中国上海市龙漕路 299 号天华信息科技园 2A 楼 5 层
邮　编：200235
E-mail：support@essemi.com
电　话：+86-21-60910333
传　真：+86-21-60914991
网　址：http://www.essemi.com/

目 录

内容目录

第1章 简介

1.1 概述

ES32 USB 协议栈提供了一套供上层应用使用的接口，包括 USB 主机以及设备功能接口。协议栈涵盖了目前市场上大部分常用的 USB 主机与设备的驱动，并同时提供给用户用于拓展协议栈驱动的方法，以及复合类设备开发的方法。

本套协议栈属于 ES32_SDK 的一部分，作为中间层驱动代码，相关驱动代码可以在 ES32_SDK\firmware\ES32_SDK\Middlewares\EastSoft\usblib 目录下找到，相关应用例程可以在 ES32_SDK\firmware\ES32_SDK\Projects\[相应芯片型号]\Applications\USB 目录下找到。USB 协议栈与 MD 以及 ALD 库对接，在 MD/ALD 的基础上拓展出主机/设备核心驱动层、主机/设备类层驱动、主机/设备层驱动。

USB 协议栈核心层驱动提供了包括枚举、中断管理、上层驱动管理、传输管理、描述符管理、标准请求管理等功能接口，是整个 USB 协议栈工作的核心。各主机/设备类层的驱动提供了 USB 各设备类的描述符解析与创建、设备类请求管理、设备类层数据传输管理、事件管理等功能，可以有效地处理 USB 核心层传来的数据。主机/设备层驱动提供了 USB 具体某个设备的驱动。图 1-1、1-2 展示了 USB 协议栈主机以及从机部分的结构：

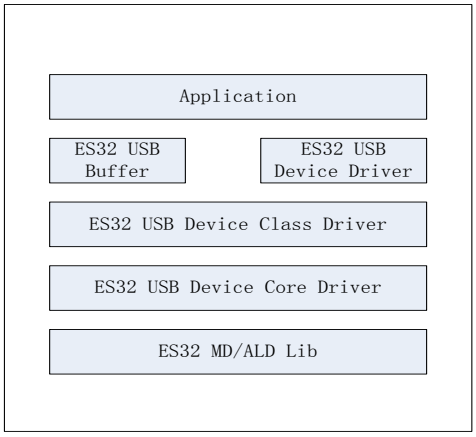


图 1-1 USB 协议栈设备驱动结构

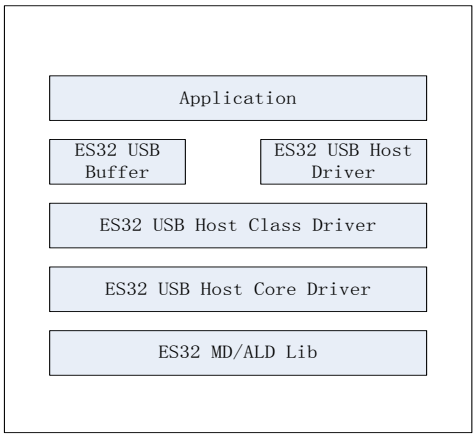


图 1-2 USB 协议栈主机驱动结构

1.2 操作模式

ES32 USB 协议栈中定义了以下用户操作模式，分别为：

eUSBModeDevice

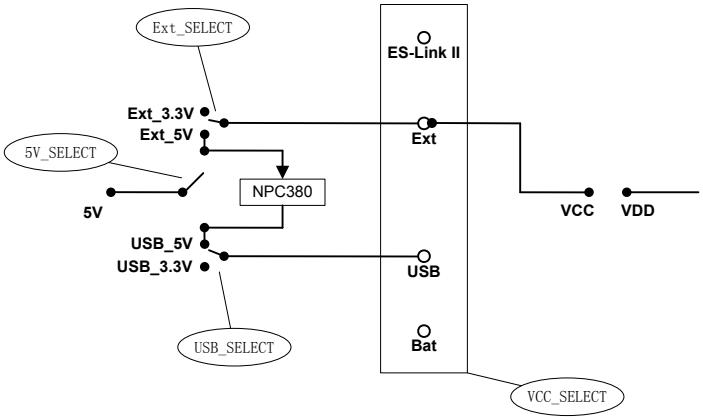
eUSBModeHost

eUSBModeForceHost

eUSBModeForceDevice

用户可以使用 USBStackModeSet()函数配置。用户配置为 eUSBModeDevice 模式时，USB 作用在设备模式，但是协议栈同时会检测 ID 以及 VBUS 引脚；配置为 eUSBModeHost 模式时，USB 作用在主机模式，但是协议栈同时会检测 ID 以及 VBUS 引脚；配置为 eUSBModeForceHost 模式时，USB 作用的主机模式，协议栈不会检测 ID 以及 VBUS 引脚；配置为 eUSBModeForceDevice 模式时，USB 作用在设备模式，协议栈不会检测 ID 以及 VBUS 引脚。如果应用明确在设备或者主机模式时，建议将协议栈工作模式设为强制主机/设备模式。

1.3 电源网络配置



电源网络配置由以下几个部分组成

- VCC_SELECT，通过单刀多掷开关来选择 VCC 的供电电源，分别为 ESLinkII、EXT、USB 和 BAT 备用电池。
- EXT_SELECT，当 VCC_SELECT 选择 EXT 供电时，EXT_SELECT 选择 VCC 是 EXT_3.3V 还是 EXT_5V。
- USB_SELECT, 当 VCC_SELECT 选择 USB 供电时，USB_SELECT 选择 VCC 是 USB_3.3V 还是 USB_5V。
- 5V_SELECT,选择 Arduino 的 5V 输入是 EXT_5V 还是 USB_5V。

当用做主机时，此时需要给 USB 接口供电，通常选择 EXT 供电的方式，配置方式如下：

1. 切换单刀多掷开关为 EXT，选择外部电源给系统供电。
2. EXT_SELECT 选择 EXT_3.3V，此时 VCC 就设置为 3.3V。
3. 5V_SELECT 选择 EXT_5V。

当用做从机时，通常选择 USB 接口给系统供电，配置方式如下：

1. 切换单刀多掷开关为 USB，选择 USB 给系统供电。

2. USB_SELECT 选择 USB_3.3V，此时 VCC 就设置为 3.3V。
3. 5V_SELECT 选择 USB_5V。

第2章 HID鼠标设备例程

2.1 简介

鼠标例程中，使用的是 HID 类驱动以及 HID 设备层驱动。驱动中已经提供好鼠标相关的发送数据的接口，如鼠标左右键、中键、滑轮、x/y 轴的移动数据，用户只需要调用鼠标的数据发送接口即可控制鼠标的动作。

2.1.1 使用方法

- 打开 ES32_SDK\Projects\ES32F0271\Applications\USB\usbdev\ usbdev_hid_mouse 路径下的鼠标代码工程，编译下载；
- 本例程可以直接使用 ES-LinkII 或者 J-Link 供电，也可以 USB 外部电源供电，当使用 USB 供电时，如使用 J-link 需要把 J-link 拔出，因为有的 J-link 的 RST 引脚上有下拉电阻；
- 按下开发板复位按钮，使用 USB 线将开发板连接到 PC 上；
- 连接到 PC 之后，HID 鼠标会枚举，之后在设备管理器中可以找到设备；
- 本例程使用通用的鼠标驱动，不需要安装驱动，用户可以使用开发板上的五向按键直接控制鼠标指针的移动。

2.1.2 程序接口

用户在参考本例程配置自己的设备时，主要修改的地方有三处，包括回调函数的完成、设备相关描述符重新配置、设备初始化结构体配置。

回调函数

回调函数用于 USB 协议栈通知上层应用一些协议栈的处理事件，例如当 USB 设备成功连接时，USB 协议栈会向上层应用发送“USB_EVENT_CONNECTED”事件。USB 协议栈设备的回调函数格式为：

```
typedef uint32_t (* tUSBCallback)(void *pvCBData, uint32_t ui32Event,
                                  uint32_t ui32MsgParam, void *pvMsgData);
```

其中参数 ui32Event 便是事件类型。

在鼠标的例程中回调函数放在 usblib_callback.c 文件中 MouseEventHandler()，用户可以参考此回调函数，选取合适的事件来重新定义设备的回调函数。

描述符

设备相关的字符串描述符在 hid_mouse_des.c 文件中，有语言支持描述符、制造商描述符、产品描述符、设备序列号描述符、接口字符串描述符、配置字符串描述符，用户可以修改这些字符串描述符来描述自己的设备，这里的描述的主要是在显示在设备列表中的一些信息，如果用户需要修改标准描述符的话就需要在各设备类层、设备层驱动的设备、配置、接口端点描述符中修改一些信息。

初始化结构体

初始化结构体主要用于初始化设备的相关接口以及相关的硬件参数，例程中统一和描述符放在同一文件中。用户可以使用此接口配置自己设备的 PID、VID、供电情况、以及

上层接口用到的回调函数指针。本例程使用的是鼠标的初始化结构体：

```
/**
  *  Mouse configuration.
  */
tUSBDHIDMouseDevice g_sMouseDevice =
{
  .ui16VID                 = USB_VID_EASTSOFT_30CC,
  .ui16PID                 = USB_PID_MOUSE,
  .ui16MaxPowermA          = 500,
  .ui8PwrAttributes        = USB_CONF_ATTR_SELF_PWR,
  .pfnCallback             = MouseEventHandler,
  .pvCBData                = (void *)&g_sMouseDevice,
  .ppui8StringDescriptors  = g_ppui8StringDescriptors,
  .ui32NumStringDescriptors = NUM_STRING_DESCRIPTORS
};
```

图 2-1 鼠标初始化结构体

以上三种接口用户可以用来重新开发设备，另外鼠标例程展示了 HID 鼠标的驱动的使用方法。鼠标例程的配置过程为：调用 usb_stack_mode_set ()函数配置 USB 协议栈的工作模式，这里配置为强制设备模式；调用 usbdhid_mouse_init ()初始化鼠标设备；在 USB_IRQHandler()中断中调用 USB 协议栈设备中断处理函数 usb_device_int_handler ()。至此一个鼠标设备的配置工作已经完成，用户如需发送鼠标的数据可用 usbdhid_mouse_state_change ()更新鼠标的状态，HID 鼠标的数据处理函数只有这一个，用户只需要使用此函数发送鼠标的状态信息到 PC 端即可。

2.2 例程演示

从图中可以看出 USB 鼠标枚举成功，在“鼠标和其它指针设备”中找到“HID-compliant mouse”，用户可以使用开发板上的五向按键直接控制 PC 端指针的移动。

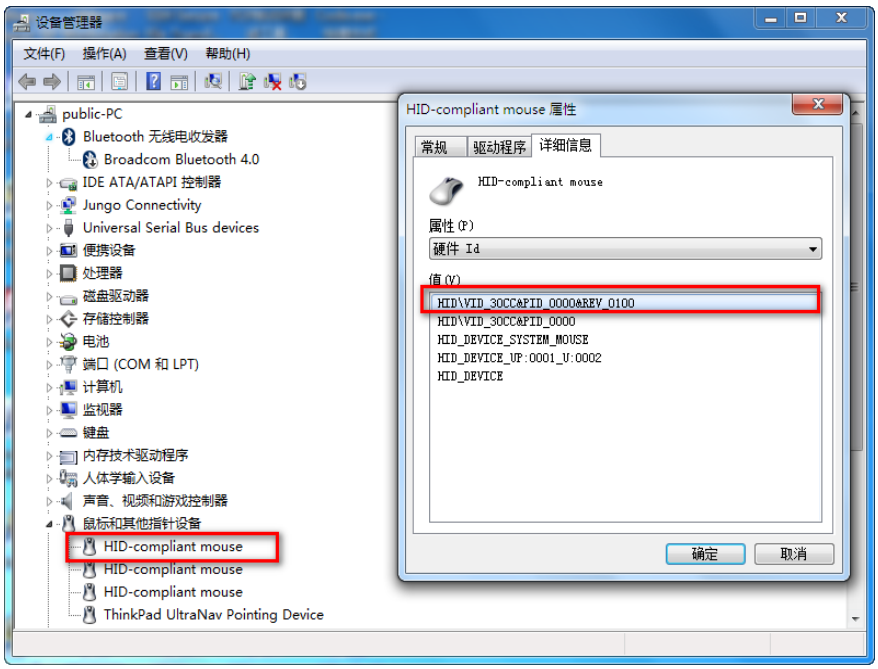


图 2-2 HID 鼠标设备

第3章 HID键盘设备例程

3.1 简介

键盘例程中，使用的是 HID 类驱动以及 HID 设备层驱动。驱动中已经提供好键盘相关的发送数据的接口，以及键盘的控制信息更新的事件。用户通过相关的数据发送接口发送数据到 PC 端，另外驱动也支持 PC 端的更新信息，如 Caps_Lock 按键按下信息更新。

3.1.1 使用方法

- 打开 ES32_SDK\Projects\ES32F0271\Applications\USB\usbdev\ usbdev_hid_keyboard 路径下的键盘代码工程，编译下载；
- 本例程可以直接使用 ES-LinkII 或者 J-Link 供电，也可以 USB 外部电源供电，当使用 USB 供电时，如使用 J-link 需要把 J-link 拔出，因为有的 J-link 的 RST 引脚上有下拉电阻；
- 打开文本编辑器；
- 按下开发板复位按钮，使用 USB 线将开发板连接到 PC 上；
- 连接到 PC 之后，HID 键盘会枚举，之后在设备管理器中可以找到设备；
- 本例程使用通用的键盘驱动，不需要安装驱动，设备枚举完成之后，用户可以使用开发板上的五向按键输入字符串到 PC 端文本编辑器，按下中间键输入"[MID]"，按下向上按键输入"[UP]"，按下向下按键输入"[DOWN]"，按下向左按键输入"[LEFT]"，按下向右按键输入"[RIGHT]"。

3.1.2 程序接口

用户在参考本例程配置自己的设备时，主要修改的地方有三处，包括回调函数的完成、设备相关描述符重新配置、设备初始化结构体配置。

回调函数

回调函数用于 USB 协议栈通知上层应用一些协议栈的处理事件，例如当 USB 设备成功连接时，USB 协议栈会向上层应用发送“USB_EVENT_CONNECTED”事件。USB 协议栈设备的回调函数格式为：

```
typedef uint32_t (* tUSBCallback)(void *pvCBData, uint32_t ui32Event,
                                  uint32_t ui32MsgParam, void *pvMsgData);
```

其中参数 ui32Event 便是事件类型。

在键盘的例程中回调函数放在 usblib_callback.c 文件中 KeyboardEventHandler()，用户可以参考此回调函数，选取合适的事件来重新定义设备的回调函数。

描述符

设备相关的字符串描述符在 hid_keyboard_des.c 文件中，有语言支持描述符、制造商描述符、产品描述符、设备序列号描述符、接口字符串描述符、配置字符串描述符，用户可以修改这些字符串描述符来描述自己的设备，这里的描述的主要是在显示在设备列表中的一些信息，如果用户需要修改标准描述符的话就需要在各设备类层、设备层驱动的设备、配置、接口端点描述符中修改一些信息。

初始化结构体

初始化结构体主要用于初始化设备的相关接口以及相关的硬件参数，例程中统一和描述符放在同一文件中。用户可以使用此接口配置自己设备的 PID、VID、供电情况、以及上层接口用到的回调函数指针。本例程使用的是键盘的初始化结构体：

```
/**
  *  Keyboard configuration.
  */
tUSBDHIDKeyboardDevice g_sKeyboardDevice =
{
  .ui16VID                 = USB_VID_EASTSOFT_30CC,
  .ui16PID                 = USB_PID_KEYBOARD,
  .ui16MaxPowermA          = 500,
  .ui8PwrAttributes        = USB_CONF_ATTR_SELF_PWR,
  .pfnCallback             = KeyboardEventHandler,
  .pvCBData                = (void *)&g_sKeyboardDevice,
  .ppui8StringDescriptors  = g_ppui8StringDescriptors,
  .ui32NumStringDescriptors = NUM_STRING_DESCRIPTORS
};
```

图 3-1 键盘初始化结构体

以上三种接口用户可以用来重新开发设备，另外键盘例程展示了 HID 键盘的驱动的使用方法。键盘例程的配置过程为：调用 usb_stack_mode_set ()函数配置 USB 协议栈的工作模式，这里配置为强制设备模式；调用 usbdhid_keyboard_init ()初始化键盘设备；在 USB_IRQHandler()中断中调用 USB 协议栈设备中断处理函数 usb_device_int_handler ()。至此一个键盘设备的配置工作已经完成，用户如需发送键盘的数据可用 usbdhid_keyboard_key_state_change ()更新键盘按键的状态，HID 键盘的数据处理函数只有这一个，用户只需要使用此函数发送键盘按键的状态信息到 PC 端即可，另外如果用户定义的键盘支持相关按键的 LED 指示功能的话，USB 协议栈给出了 USBD_HID_KEYB_EVENT_SET_LEDS 的事件（在例程中已经将事件引出，可作为用户参考），用户在此事件到来时可以设置相应的 LED 灯状态。

3.2 例程演示

设备枚举成功后，用户可以使用开发板上的五向按键输入字符串到 PC 端文本编辑器，如下图所示：

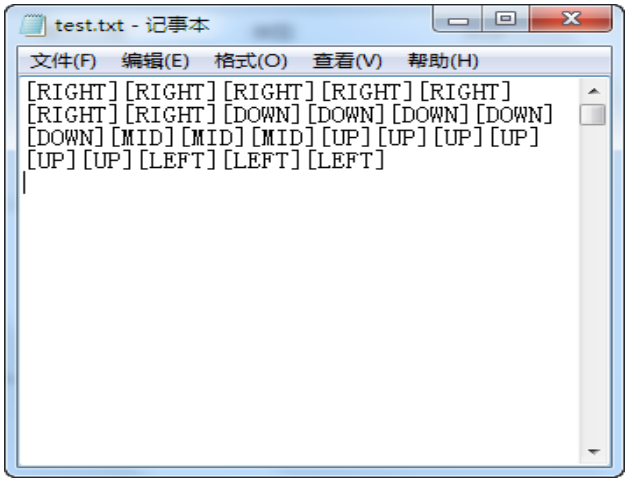


图 3-2 HID keyboard 演示图

第4章 HID自定义设备例程

4.1 简介

HID 自定义设备例程中，使用的是 HID 类驱动，为用户展示怎样在 HID 通用驱动上开发出自定设备。驱动中已经提供好 HID 相关的发送接收数据的接口，用户可以用来作为自定义设备数据收发的底层对接接口。

4.1.1 使用方法

- 打开 ES32_SDK\Projects\ES32F0271\Applications\USB\usbdev\usbdev_hid_custom_userspec 路径下的自定义设备代码工程，编译下载；
- 本例程可以直接使用 ES-LinkII 或者 J-Link 供电，也可以 USB 外部电源供电，当使用 USB 供电时，如使用 J-link 需要把 J-link 拔出，因为有的 J-link 的 RST 引脚上有下拉电阻；
- 按下开发板复位按钮，使用 USB 线将开发板连接到 PC 上；
- 连接到 PC 之后，HID 自定义设备会枚举，之后在设备管理器中可以找到设备，设备枚举完成用户可以通过 ES_USB_LAB 进行演示。

4.1.2 程序接口

用户在参考本例程配置自己的设备时，主要修改以下配置

HID 设备驱动

HID 自定义设备需要用户熟悉 HID 报告描述符，在 HID 自定义的例程中 HID 报告描述符定义在 customhid.c 中。

回调函数

回调函数用于 USB 协议栈通知上层应用一些协议栈的处理事件，例如当 USB 设备成功连接时，USB 协议栈会向上层应用发送“USB_EVENT_CONNECTED”事件。USB 协议栈设备的回调函数格式为：

```
typedef uint32_t (* tUSBCallback)(void *pvCBData, uint32_t ui32Event,
                                  uint32_t ui32MsgParam, void *pvMsgData);
```

其中参数 ui32Event 便是事件类型。

在 HID 自定义的例程中回调函数放在 usblib_callback.c 文件中 CustomHIDEventHandler()，用户可以参考此回调函数，选取合适的事件来重新定义设备的回调函数。

描述符

设备相关的字符串描述符在 customhid_des.c 文件中，有语言支持描述符、制造商描述符、产品描述符、设备序列号描述符、接口字符串描述符、配置字符串描述符，用户可以修改这些字符串描述符来描述自己的设备，这里的描述的主要是在显示在设备列表中的一些信息，如果用户需要修改标准描述符的话就需要在各设备类层、设备层驱动的设备、配置、接口端点描述符中修改一些信息。

初始化结构体

初始化结构体主要用于初始化设备的相关接口以及相关的硬件参数，例程中统一和描述符放在同一文件中。用户可以使用此接口配置自己设备的 PID、VID、供电情况、以及上层接口用到的回调函数指针。本例程使用的是 HID 自定义设备的初始化结构体：

```
/**
  *  CustomHID configuration.
  */
tUSBDHIDCustomHIDDevice g_sCustomHIDDevice =
{
  .ui16VID                  = USB_VID_EASTSOFT_30CC,
  .ui16PID                  = USB_PID_HID_CUSTOMHID,
  .ui16MaxPowermA           = 500,
  .ui8PwrAttributes         = USB_CONF_ATTR_SELF_PWR,
  .pfnCallback              = CustomHIDEventHandler,
  .pvCBData                 = (void *)&g_sCustomHIDDevice,
  .ppui8StringDescriptors   = g_ppui8StringDescriptors,
  .ui32NumStringDescriptors = NUM_STRING_DESCRIPTORS
};
```

图 4-1 HID 自定义设备初始化结构体

以上接口用户可以用来重新开发设备，在 HID 设备类层开发自定义设备的话需要了解 HID 设备类层的接口。HID 设备类层驱动接口函数主要分为三种，第一种为核心层事件驱动、设备类调度驱动、设备类数据读写相关驱动，与用户相关的驱动为事件驱动以及设备类数据读写驱动。

4. 2 例程演示

打开上位机选择 HID 一栏，查找到自定义 HID 设备“ES32 Customhid Device”。打开此设备后，就可以通过鼠标点击 LED 按钮来控制学习板上的 LED 情况；按键一栏显示学习板上的按键情况，当某个按键被按住时，对应的键就会变成绿色。

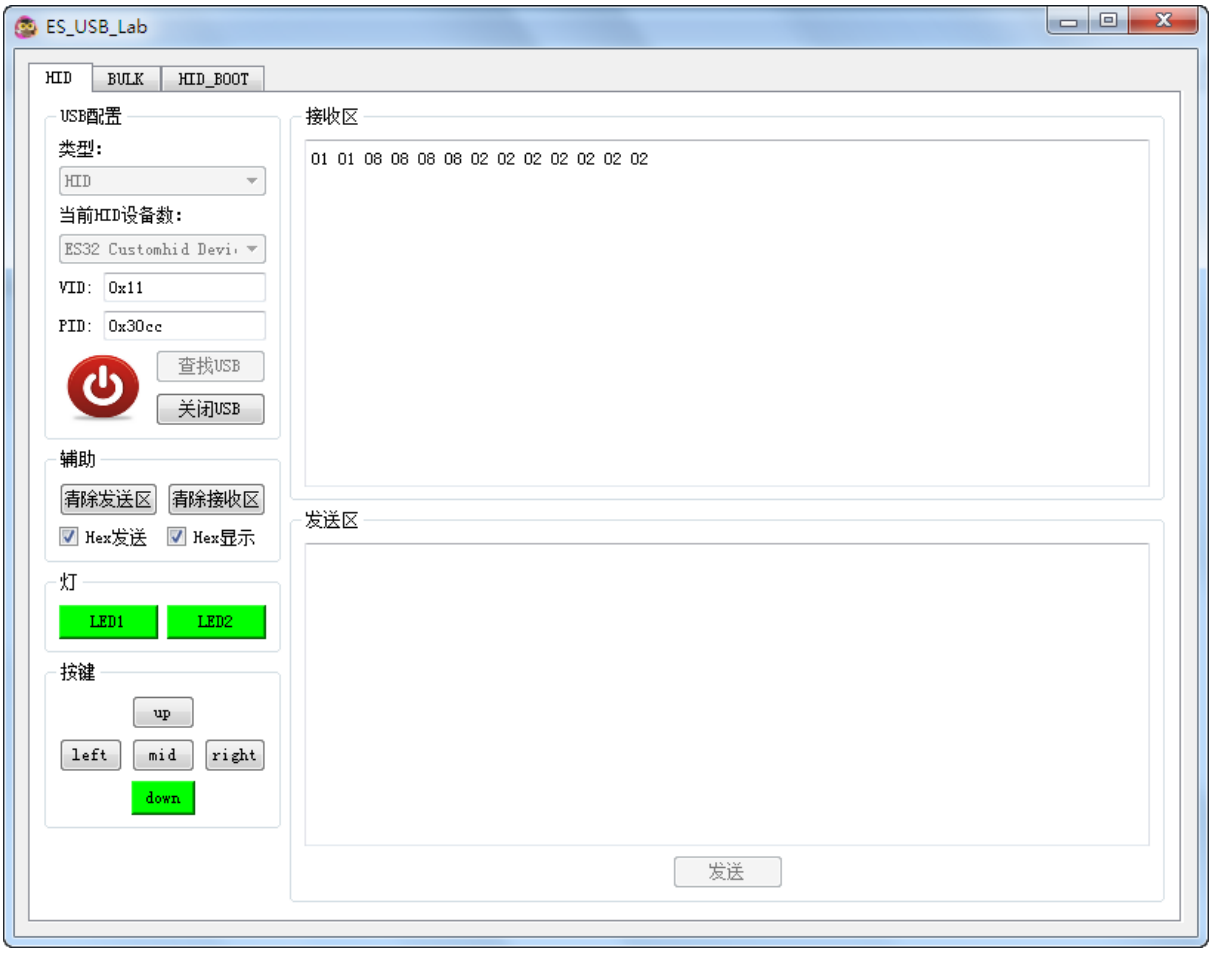


图 4-2 HID 自定义设备上位机演示

第5章 HID Boot例程

5. 1 简介

HID boot 设备例程中，通过上位机 ES_USB_LAB 使用 hid 接口对 MCU 的 flash 进行更新。

5. 1. 1 使用方法

- 打开 ES32_SDK\Projects\ES32F0271\Applications\USB\usbdev\usbdev_hid_custom_boot 路径下的自定义设备代码工程，编译下载；
- 本例程可以直接使用 ES-LinkII 或者 J-Link 供电，也可以 USB 外部电源供电，当使用 USB 供电时，如使用 J-link 需要把 J-link 拔出，因为有的 J-link 的 RST 引脚上有下拉电阻；
- 按下开发板复位按钮，使用 USB 线将开发板连接到 PC 上；
- 连接到 PC 之后，HID 自定义设备会枚举，之后在设备管理器中可以找到设备，设备枚举完成用户可以通过 ES_USB_LAB 进行演示。
- 通过 ES_USB_LAB 对 Flash 更新成功后，此设备就会进入 APP 程序,并保持在 APP 中运行。
- 按住开发板上五向按键的中间键的同时，拔下开发板并将其重新插入计算机中(或按下开发板上的复位按钮)，设备会重新进入 boot 程序中。
- ES32F0271 芯片只支持对二进制文件的更新。

5. 1. 2 程序设计

本例程实现了自定义 HID 设备，并接收上位机数据来实现 bootloader 功能。

回调函数

回调函数用于 USB 协议栈通知上层应用一些协议栈的处理事件，例如当 USB 设备成功连接时，USB 协议栈会向上层应用发送“USB_EVENT_CONNECTED”事件。USB 协议栈设备的回调函数格式为：

```
typedef uint32_t (* tUSBCallback)(void *pvCBData, uint32_t ui32Event,
                                  uint32_t ui32MsgParam, void *pvMsgData);
```

其中参数 ui32Event 便是事件类型。

在 HID 自定义的例程中回调函数放在 usblib_callback.c 文件中 RxHandler ()，在此函数中处理接收到的数据。

描述符

设备相关的字符串描述符在 hid_bootloader_des.c 文件中，有语言支持描述符、制造商描述符、产品描述符、设备序列号描述符、接口字符串描述符、配置字符串描述符，用户可以修改这些字符串描述符来描述自己的设备，这里的描述的主要是在显示在设备列表中的一些信息，如果用户需要修改标准描述符的话就需要在各设备类层、设备层驱动的设备、配置、接口端点描述符中修改一些信息。

HID Bootloader 是在 HID 自定义设备的基础上添加 USB 协议栈自带的 buffer 管理模块来实现数据的发送与接收。相关接口为：

usb_buffer_init()----buffer 初始化函数

usb_buffer_read ()----buffer 的读取函数

usb_buffer_write ()----buffer 的写函数

usb_buffer_data_available ()----buffer 可用数据长度判断函数

usb_buffer_flush ()----清空 buffer 函数

向以上四个接口写入或者读取数据即可实现串口数据的发送和读取。此外 buffer 对象在使用之前需要初始化，在串口例程中定义了发送和接收两个 buffer 对象。buffer 对象需要提供读写以及判断数据长度的接口，这三种接口在所有的类驱动中都会给出，用户只需要将相应的函数指针赋值给 buffer 对象即可。

5. 2 例程演示

打开上位机选择 HID 一栏，查找到自定义 HID 设备“ES32 Hid Bootloader Device”。打开此设备，在“bin 文件”一栏填入 APP 的起始地址，点击“bin”按钮选择要加载的二进制文件，点击“开始”按钮，开始对 MCU 的 flash 进行更新。

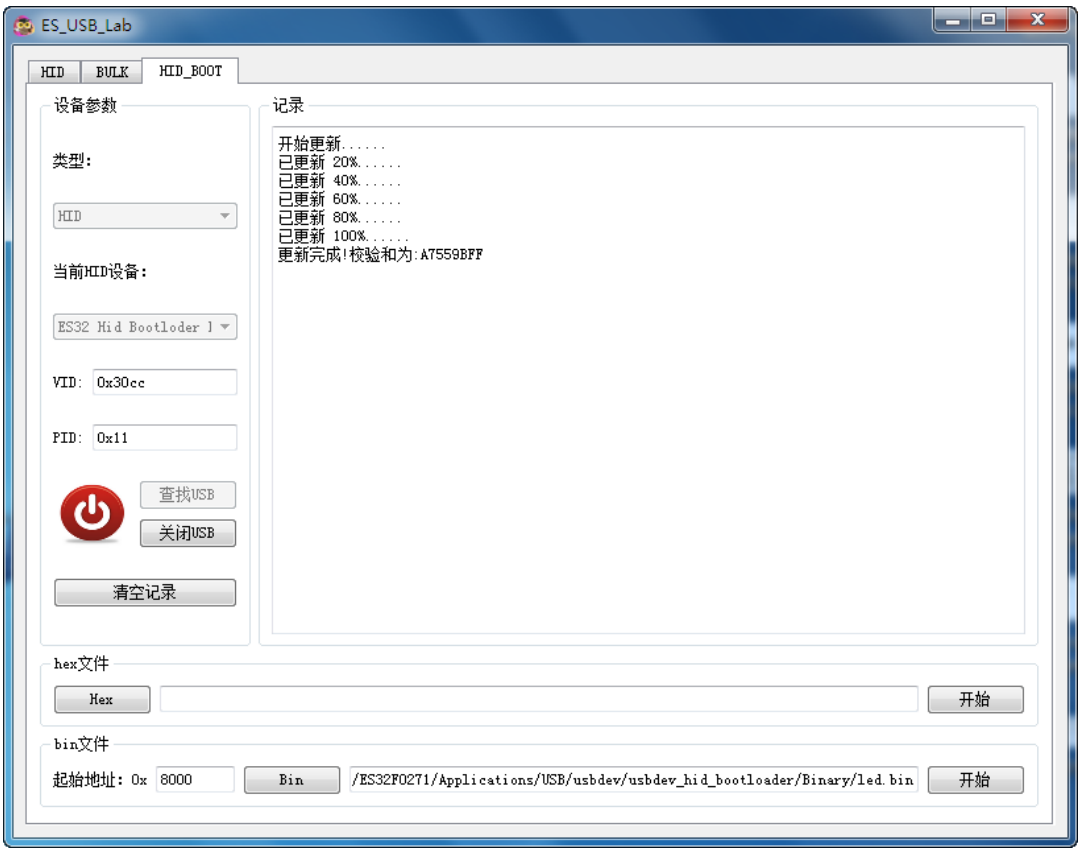


图 5-1 HID boot 设备上位机演示

上位机与 HID boot 设备的通信协议见附录。

第6章 CDC串口设备例程

6.1 简介

CDC 设备例程中，使用的是 CDC 类驱动，为用户展示 CDC 类驱动的使用方法。本例程配置了一个 CDC 虚拟串口，需要安装驱动（驱动放在 ES32_SDK\Utilities\usb_windows_drivers 文件夹中，第一次使用时需要引导安装驱动）。

6.1.1 使用方法

- 打开 ES32_SDK\Projects\ES32F0271\Applications\USB\usbdev\usbdev_cdc_serial 路径下的自定义设备代码工程，编译下载；
- 本例程可以直接使用 ES-LinkII 或者 J-Link 供电，也可以 USB 外部电源供电，当使用 USB 供电时，如使用 J-link 需要把 J-link 拔出，因为有的 J-link 的 RST 引脚上有下拉电阻；
- 按下开发板复位按钮，使用 USB 线将开发板连接到 PC 上；
- 连接到 PC 之后，CDC 设备会枚举，之后在设备管理器中可以找到设备；
- 第一次使用时需要安装驱动，在设备管理器中更新设备驱动，找到 ES32_SDK\Utilities\usb_windows_drivers 文件夹，然后安装即可，安装成功后，在串口设备列表中可以找到“ES32 Serial Port”设备。
- 打开串口助手，选择本 CDC 串口，在串口助手上向开发板发送数据，开发板会将数据回显到串口助手上。

6.1.2 程序接口

用户在参考本例程配置自己的设备时，主要修改的地方有三处，包括回调函数的完成、设备相关描述符重新配置、设备初始化结构体配置。

回调函数

回调函数用于 USB 协议栈通知上层应用一些协议栈的处理事件，例如当 USB 设备成功连接时，USB 协议栈会向上层应用发送“USB_EVENT_CONNECTED”事件。USB 协议栈设备的回调函数格式为：

```
typedef uint32_t (* tUSBCallback)(void *pvCBData, uint32_t ui32Event,
                                  uint32_t ui32MsgParam, void *pvMsgData);
```

其中参数 ui32Event 便是事件类型。

在 CDC 设备的例程中回调函数放在 usblib_callback.c 文件中，由于 CDC 有控制类接口以及数据类接口，所以 ControlHandler()函数用于控制类接口，TxHandler()和 RxHandler()回调函数用于数据接收和发送，用户可以参考这些回调函数，选取合适的事件来重新定义设备的回调函数。

描述符

设备相关的字符串描述符在 cdc_des.c 文件中，有语言支持描述符、制造商描述符、产品描述符、设备序列号描述符、接口字符串描述符、配置字符串描述符，用户可以修改这些字符串描述符来描述自己的设备，这里的描述的主要是在显示在设备列表中的一些信

息，如果用户需要修改标准描述符的话就需要在各设备类层、设备层驱动的设备、配置、接口端点描述符中修改相应的信息。

初始化结构体

初始化结构体主要用于初始化设备的相关接口以及相关的硬件参数，例程中统一和描述符放在同一文件中。用户可以使用此接口配置自己设备的 PID、VID、供电情况、以及上层接口用到的回调函数指针。本例程使用的是 HID 自定义设备的初始化结构体：

```
tUSBDCDCDevice g_psCDCDevice =
{
    .ui16VID                = USB_VID_EASTSOFT_30CC,
    .ui16PID                = USB_PID_SERIAL,
    .ui16MaxPowermA         = 250,
    .ui8PwrAttributes       = USB_CONF_ATTR_BUS_PWR,
    .pfnControlCallback     = ControlHandler,
    .pvControlCBData        = (void *)&g_psCDCDevice,
    .pfnRxCallback          = usb_buffer_event_callback,
    .pvRxCBData             = (void *)&g_sRxBuffer,
    .pfnTxCallback          = usb_buffer_event_callback,
    .pvTxCBData             = (void *)&g_sTxBuffer,
    .ppui8StringDescriptors = g_ppui8StringDescriptors,
    .ui32NumStringDescriptors = NUM_STRING_DESCRIPTORS
};
```

图 6-1 CDC 设备初始化结构体

以上三种接口用户可以用来重新开发设备，另外 CDC 串口例程展示了 CDC 类驱动的使用方法。CDC 设备配置过程为：初始化 buffer 对象，包括发送和接收 buffer，调用 usb_stack_mode_set ()函数配置 USB 协议栈的工作模式，这里配置为强制设备模式；调用 usbdcdc_init ()初始化 CDC 串口设备；在 USB_IRQHandler()中断中调用 USB 协议栈设备中断处理函数 usb_device_int_handler ()。至此一个 CDC 串口设备的配置工作已经完成，本例程为了方便用户管理 CDC 串口的数据发送与接收，在 CDC 串口类上使用 USB 协议栈自带的 buffer 管理模块来实现数据的发送与接收。相关接口为：

usb_buffer_init()----buffer 初始化函数

usb_buffer_read ()----buffer 的读取函数

usb_buffer_write ()----buffer 的写函数

usb_buffer_data_available ()----buffer 可用数据长度判断函数

usb_buffer_flush ()----清空 buffer 函数

向以上四个接口写入或者读取数据即可实现串口数据的发送和读取。此外 buffer 对象在使用之前需要初始化，在串口例程中定义了发送和接收两个 buffer 对象。buffer 对象需要提供读写以及判断数据长度的接口，这三种接口在所有的类驱动中都会给出，用户只需要将相应的函数指针赋值给 buffer 对象即可。

```
uint8_t g_pui8USBRxBuffer[BULK_BUFFER_SIZE];
tUSBBuffer g_sRxBuffer =
{
    .bTransmitBuffer    = false,                    \
                          // This is a receive buffer.
    .pfnCallback        = RxHandler,                \
                          // pfnCallback
    .pvCBData           = (void *)&g_psCDCDevice,   \
                          // Callback data is our device pointer.
    .pfnTransfer        = usbdcdc_packet_read,      \
                          // pfnTransfer
    .pfnAvailable       = usbdcdc_rx_packet_available, \
                          // pfnAvailable
    .pvHandle           = (void *)&g_psCDCDevice,   \
                          // pvHandle
    .pui8Buffer         = g_pui8USBRxBuffer,        \
                          // pi8Buffer
    .ui32BufferSize     = BULK_BUFFER_SIZE,         \
                          // ui32BufferSize
};
```

图 6-2 接收 buffer 初始化结构体

```
uint8_t g_pui8USBTxBuffer[BULK_BUFFER_SIZE];
tUSBBuffer g_sTxBuffer =
{
    .bTransmitBuffer    = true,                     \
                          // This is a transmit buffer.
    .pfnCallback        = TxHandler,                \
                          // pfnCallback
    .pvCBData           = (void *)&g_psCDCDevice,   \
                          // Callback data is our device pointer.
    .pfnTransfer        = usbdcdc_packet_write,     \
                          // pfnTransfer
    .pfnAvailable       = usbdcdc_tx_packet_available, \
                          // pfnAvailable
    .pvHandle           = (void *)&g_psCDCDevice,   \
                          // pvHandle
    .pui8Buffer         = g_pui8USBTxBuffer,        \
                          // pi8Buffer
    .ui32BufferSize     = BULK_BUFFER_SIZE,         \
                          // ui32BufferSize
};
```

图 6-3 发送 buffer 初始化结构体

设备类数据读写相关驱动

前面有介绍到 buffer 的数据传输接口，实际上 buffer 的数据传输接口最终会调用使用 buffer 传输的设备类的数据传输接口，CDC 的数据传输接口为：

usbdcdc_packet_write ()-CDC 数据包发送函数

usbdcdc_packet_read ()-CDC 数据包读取函数

usbdcdc_tx_packet_available ()-CDC 判断发送数据是否有效函数

usbdcdc_rx_packet_available ()-CDC 判断是否有可读数据函数

在所有的设备类层的驱动中都会提供以上四个函数，用户如果有需要直接在设备类层开发设备的话可以直接使用以上四类函数来操作数据的读写。

6.2 例程演示

使用串口助手查找并打开“ES32 Serial Port”，通过串口助手向开发板发送数据，开发板会将数据返回到串口助手上。

图 6-4 usb cdc 例程演示图

第7章 复合设备例程

7.1 简介

复合设备例程创建了一个串口和鼠标的复功能设备，其中串口部分需要安装驱动（驱动放在 ES32_SDK\Utilities\usb_windows_drivers 文件夹中，第一次使用时需要引导安装驱动）。

7.1.1 使用方法

- 打开

  ES32_SDK\Projects\ES32F0271\Applications\USB\usbdev\ usbdev_composite_device 路径下的复合设备代码工程，编译下载；

- 本例程可以直接使用 ES-LinkII 或者 J-Link 供电，也可以 USB 外部电源供电，当使用 USB 供电时，如使用 J-link 需要把 J-link 拔出，因为有的 J-link 的 RST 引脚上有下拉电阻；
- 按下开发板复位按钮，使用 USB 线将开发板连接到 PC 上；
- 连接到 PC 之后，复合设备会枚举，之后在设备管理器中可以找到相应的串口以及鼠标设备，设备管理器会显示三种设备（复合设备、鼠标、串口）；

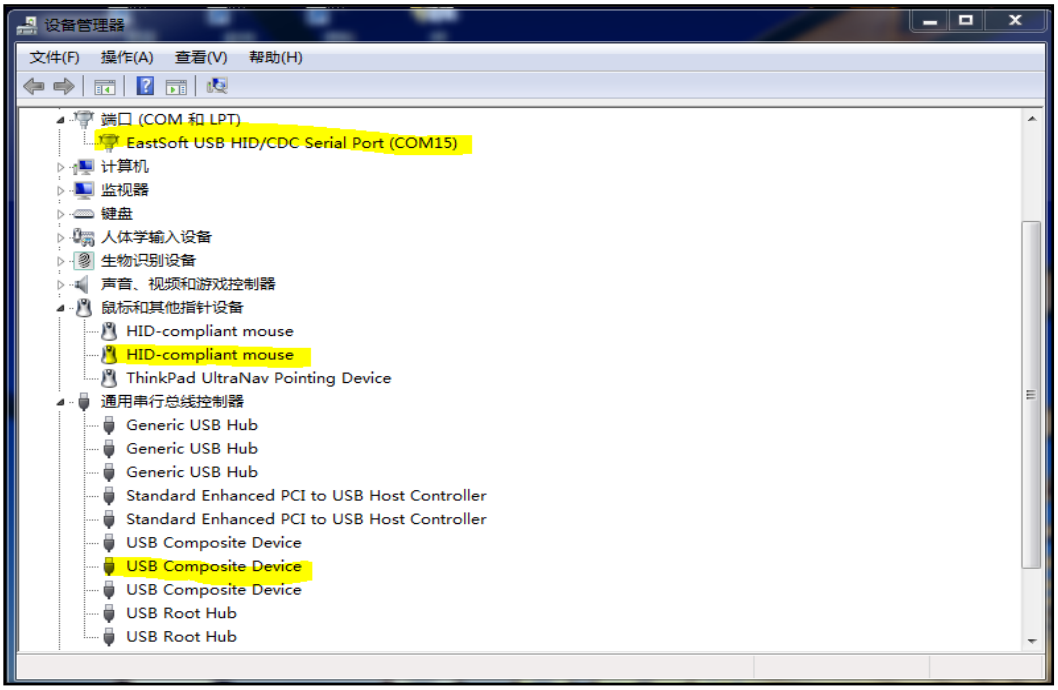


图 7-1 复合设备枚举显示

- 第一次使用时需要安装驱动，在设备管理器中更新设备驱动，找到 ES32_SDK\Utilities\usb_windows_drivers 文件夹，然后安装即可，安装成功后，在串口设备列表中可以找到“ES32 USB HID/CDC Serial Port”设备。
- 打开串口助手，选择本 CDC 串口（安装驱动后在串口助手上会显示“EastSoft USB HID/CDC Serial Port”），在串口助手上向开发板发送数据，开发板会将数据回显到串口助手上，另外鼠标设备功能是一直控制鼠标指针向右平移。

7. 1. 2 程序接口

用户在参考本例程配置自己的设备时，主要修改的地方有三处，包括回调函数的完成、设备相关描述符重新配置、设备初始化结构体配置。

回调函数

回调函数用于 USB 协议栈通知上层应用一些协议栈的处理事件，例如当 USB 设备成功连接时，USB 协议栈会向上层应用发送“USB_EVENT_CONNECTED”事件。USB 协议栈设备的回调函数格式为：

```
typedef uint32_t (* tUSBCallback)(void *pvCBData, uint32_t ui32Event,
                                  uint32_t ui32MsgParam, void *pvMsgData);
```

其中参数 ui32Event 便是事件类型。

在复合设备的例程中回调函数放在 usblib_callback.c 文件中，这里需要注意的是，复合类设备的回调函数还是会按照每个设备类的回调来完成，例如本例程中的回调函数使用的是鼠标以及 CDC 串口设备的回调函数，回调函数可以参考鼠标以及 CDC 串口设备例程。复合类设备的驱动会将所需创建的每个子设备的驱动管理起来，包括回调以及描述符的管理，所以当有 CDC 类数据传输的话，使用的依然是 CDC 类的驱动，当有 HID 类设备的数据传输的话，使用的是 HID 类各种设备驱动。

描述符

设备相关的字符串描述符在 cdc_des.c 文件中，有语言支持描述符、制造商描述符、产品描述符、设备序列号描述符、接口字符串描述符、配置字符串描述符，用户可以修改这些字符串描述符来描述自己的设备，这里的描述的主要是在显示在设备列表中的一些信息，如果用户需要修改标准描述符的话就需要在各设备类层、设备层驱动的设备、配置、接口端点描述符中修改一些信息。

初始化结构体

初始化结构体主要用于初始化设备的相关接口以及相关的硬件参数，例程中统一和描述符放在同一文件中。用户可以使用此接口配置自己设备的 PID、VID、供电情况、以及上层接口用到的回调函数指针。本例程使用三个设备类的初始化结构体，CDC、鼠标、复合设备的初始化结构体。

```
tUSBDCompositeDevice g_sCompDevice =
{
  .ui16VID                  = USB_VID_EASTSOFT_30CC,
  .ui16PID                  = USB_PID_COMP_HID_SER,
  .ui16MaxPowermA           = 250,
  .ui8PwrAttributes         = USB_CONF_ATTR_BUS_PWR,
  .pfnCallback              = 0,
  .ppui8StringDescriptors   = g_ppui8StringDescriptors,
  .ui32NumStringDescriptors = NUM_STRING_DESCRIPTORS,
  .ui32NumDevices           = NUM_COMP_DEVICES,
  .psDevices                = g_psCompEntries
};
```

图 7-2 复合设备初始化结构体

```
tUSBDHIDMouseDevice g_sMouseDevice =
{
  .ui16VID                  = USB_VID_EASTSOFT_30CC,
  .ui16PID                  = USB_PID_MOUSE,
  .ui16MaxPowermA           = 250,
  .ui8PwrAttributes         = USB_CONF_ATTR_SELF_PWR,
  .pfnCallback              = MouseEventHandler,
  .pvCBData                 = (void *)&g_sMouseDevice,
  .ppui8StringDescriptors   = g_ppui8StringDescriptors,
  .ui32NumStringDescriptors = NUM_STRING_DESCRIPTORS
};
```

图 7-3 鼠标设备初始化结构体

```
tUSBDCDCDevice g_psCDCDevice =
{
  .ui16VID                  = USB_VID_EASTSOFT_30CC,
  .ui16PID                  = USB_PID_SERIAL,
  .ui16MaxPowermA           = 250,
  .ui8PwrAttributes         = USB_CONF_ATTR_SELF_PWR,
  .pfnControlCallback       = ControlHandler,
  .pvControlCBData          = (void *)&g_psCDCDevice,
  .pfnRxCallback            = USBBufferEventCallback,
  .pvRxCBData               = (void *)&g_sRxBuffer,
  .pfnTxCallback            = USBBufferEventCallback,
  .pvTxCBData               = (void *)&g_sTxBuffer,
  .ppui8StringDescriptors   = g_ppui8StringDescriptors,
  .ui32NumStringDescriptors = NUM_STRING_DESCRIPTORS
};
```

图 7-4 CDC 串口设备初始化结构体

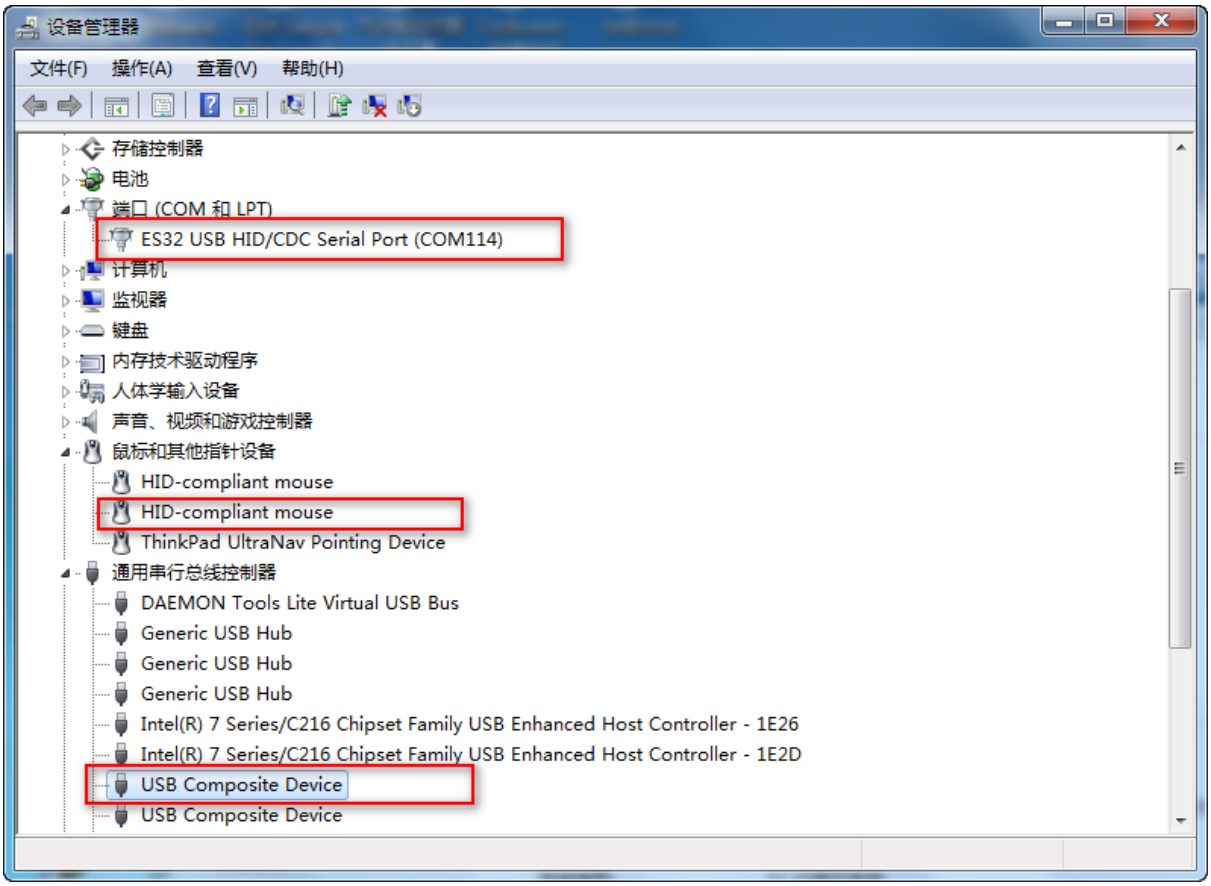
虽然使用的是三个初始化结构体，但是最终复合类设备会将三个初始化结构体结合，PID、VID、以及电流特性最终以复合设备初始化结构体所定义的内容为准。

设备类数据读写相关驱动

对于 CDC 类的数据传输接口可以参考 CDC 串口设备例程中讲到的 CDC 类读写相关的接口，对于鼠标设备来说，只有 usbdhid_mouse_state_change()一个函数接口来发送

鼠标动作数据。

7.2 例程演示



第8章 Audio设备例程

8.1 简介

audio 设备采用 USB 传输模式中的 Isochronous transfers 模式，Isochronous transfers 传输模式是专门针对流媒体特点的传输方法。此例程实现了 usb 声卡的功能，用户需要外接 ES-PSD_AUDIO 子板，此子板板载了一颗高性能的 CODEC 芯片：WM8978。

WM8978 是一个低功耗、高质量的立体声对媒体数字信号编译码器，集成 DAC 和 ADC，可以实现声音信号量化成数字量输出，也可以实现数字量音频数据转化为模拟量声音驱动扬声器。WM8978 芯片本身没有保存音频数据的功能，在此例程中 ES32F0271 利用 USB 接口接收 PC 端的数据，并通过 I2S 协议与 WM8978 芯片进行数据传输。用户耳机插入 ES-PSD_AUDIO 子板的 PHONE 端口就能收听来自电脑的音频信号。

8.1.1 使用方法

- 打开 ES32_SDK\Projects\ES32F0271\Applications\USB\usbdev\usbdev_audio 路径下的代码工程，编译下载；
- 本例程可以直接使用 ES-LinkII 或者 J-Link 供电，也可以 USB 外部电源供电，当使用 USB 供电时，如使用 J-link 需要把 J-link 拔出，因为有的 J-link 的 RST 引脚上有下拉电阻；
- 按下开发板复位按钮，使用 USB 线将开发板连接到 PC 上；
- 连接到 PC 之后，audio 设备会枚举，之后在设备管理器中可以找到相应的声卡设备，显示为“ES32 Audio Device”；
- 本例程不需要安装驱动，直接使用通用声卡驱动即可；
- 连接耳机后可以听到 PC 端的音频信号（本例程是 48KHz 采样率，双声道 16-bits 数据）。

8.1.2 程序接口

用户在参考本例程配置自己的设备时，主要修改的地方有三处，包括回调函数的完成、设备相关描述符重新配置、设备初始化结构体配置。

回调函数

回调函数用于 USB 协议栈通知上层应用一些协议栈的处理事件，例如当 USB 设备成功连接时，USB 协议栈会向上层应用发送“USB_EVENT_CONNECTED”事件。USB 协议栈设备的回调函数格式为：

```
typedef uint32_t (* tUSBCallback)(void *pvCBData, uint32_t ui32Event,
                                  uint32_t ui32MsgParam, void *pvMsgData);
```

其中参数 ui32Event 便是事件类型。

在 audio 设备的例程中回调函数放在 usblib_callback.c 文件中，audio 设备中使用的回调为 AudioHandler()函数，但是此回调仅仅用于实现控制 audio 设备，audio 设备的数据传输在 audio 类驱动中给出了单独设置 audio 数据传输的接口，以下将会介绍。

描述符

设备相关的字符串描述符在 audio_des.c 文件中，有语言支持描述符、制造商描述符、

产品描述符、设备序列号描述符、接口字符串描述符、配置字符串描述符，用户可以修改这些字符串描述符来描述自己的设备，这里的描述的主要是在显示在设备列表中的一些信息，如果用户需要修改标准描述符的话就需要在各设备类层、设备层驱动的设备、配置、接口端点描述符中修改一些信息。

初始化结构体

初始化结构体主要用于初始化设备的相关接口以及相关的硬件参数，例程中统一和描述符放在同一文件中。用户可以使用此接口配置自己设备的 PID、VID、供电情况、以及上层接口用到的回调函数指针。

```
tUSBDAudioDevice g_sAudioDevice =
{
    .ui16VID                = USB_VID_EASTSOFT_30CC,
    .ui16PID                = USB_PID_AUDIO,
    .ui16MaxPowermA         = 250,
    .ui8PwrAttributes       = USB_CONF_ATTR_BUS_PWR,
    .pfnCallback            = AudioHandler,
    .ppui8StringDescriptors = g_ppui8StringDescriptors,
    .ui32NumStringDescriptors = NUM_STRING_DESCRIPTORS,
    .i16VolumeMax           = 60,
    .i16VolumeMin           = 0,
    .i16VolumeStep          = 1
};
```

图 8-1 audio 设备初始化结构体

设备类数据读写相关驱动

audio 设备并没有像前面提到的 HID 或者 CDC 设备类驱动都提供了数据读写的接口，由于 audio 设备是同步传输，所以数据需要及时地处理。audio 驱动的做法是直接提供一个设置存储 audio 数据的接口，即 usb_audio_buffer_out()，该接口设置了 audio 数据的存储位置、存储块大小以及回调函数的传入接口。在 audio 例程中创建了 AudioBufferHandler() 的回调函数，用来处理 audio 数据读取事件，即 USBD_AUDIO_EVENT_DATAOUT 事件。

8. 2 例程演示

接上“ES-PDS_AUDIO”音频子板后，接上电脑，在“音量合成器-扬声器”中选择“扬声器（ES32 Audio Device）”，接上耳机中就能听到 PC 端的声音。

图 8-2 USB Audio 演示

开发板使用的外部 8M 时钟倍频后分频用作 IIS 时钟，时钟输出（MCLK）实际为 46.875KHz，存在误差。导致 PC 主机传输的数据比 IIS 传输的数据快，会产生噪音，如果想产生准确的 48K 采样率，可以为系统外接 24.576MHz 的晶振。

第9章 MSC设备例程

9.1 简介

MSC 设备例程创建了大容量存储设备（mass storage device），将固件复制到此存储设备后，设备会将此固件烧录到 MCU 的 flash 中。

9.1.1 使用方法

- 打开 ES32_SDK\Projects\ES32F0271\Applications\USB\usbdev\usbdev_msc_bootloader 路径下的 msc 代码工程，编译下载；
- 本例程可以直接使用 ES-LinkII 或者 J-Link 供电，也可以 USB 外部电源供电，当使用 USB 供电时，如使用 J-link 需要把 J-link 拔出，因为有的 J-link 的 RST 引脚上有下拉电阻；
- 按下开发板复位按钮，使用 USB 线将开发板连接到 PC 上；
- 连接到 PC 之后，会在 Windows 系统弹出名称为“BOOTLOADER”的盘符；
- 打开“BOOTLOADER”U 盘，里面有 READY.TXT 表示引导程序已就绪，可以接收固件。
- 复制新固件到 U 盘内，USB 设备会自动重启，当在此显示“BOOTLOADER”U 盘时，再次打开 U 盘，里面有 SUCCESS.TXT 表示升级成功，有 FAILED.TXT 表示升级失败。
- 如果升级成功，将此 USB 设备重新上电后就会进入 APP 程序,并保持在 APP 中运行。
- 按住开发板上五向按键的中间键，拔下开发板并将其重新插入计算机中(或按下开发板上的复位按钮)，设备会重新进入 bootloader 程序中。
- ES32F0271 芯片只支持对二进制文件的更新

9.1.2 程序接口

用户在参考本例程配置自己的设备时，主要修改的地方有三处，包括回调函数的完成、设备相关描述符重新配置、设备初始化结构体配置。

描述符

设备相关的字符串描述符在 msc_des.c 文件中，有语言支持描述符、制造商描述符、产品描述符、设备序列号描述符、接口字符串描述符、配置字符串描述符，用户可以修改这些字符串描述符来描述自己的设备，这里的描述的主要是在显示在设备列表中的一些信息，如果用户需要修改标准描述符的话就需要在各设备类层、设备层驱动的设备、配置、接口端点描述符中修改一些信息。

初始化结构体

初始化结构体主要用于初始化设备的相关接口以及相关的硬件参数，例程中统一和描述符放在同一文件中。用户可以使用此接口配置自己设备的 PID、VID、供电情况、以及上层接口用到的回调函数指针。

```
tUSBDMSCDevice g_sMSCDevice =
{
    .ui16VID            = USB_VID_EASTSOFT_30CC,
    .ui16PID            = USB_PID_MSC,
    .pui8Vendor         = "EASTSOFT",
    .pui8Product        = "Mass Storage    ",
    .pui8Version        = "1.00",
    .ui16MaxPowermA     = 500,
    .ui8PwrAttributes   = USB_CONF_ATTR_SELF_PWR,
    .ppui8StringDescriptors = g_ppui8StringDescriptors,
    .ui32NumStringDescriptors = NUM_STRING_DESCRIPTORS,
    .sMediaFunctions    =
    {
        usbd_msc_storage_open,
        usbd_msc_storage_close,
        usbd_msc_storage_read,
        usbd_msc_storage_write,
        usbd_msc_storage_block_number,
    },
    .pfnEventCallback   = USBDMSCEventCallback,
};
```

图 9-1 msc dev bootloader 设备初始化结构体

存储介质访问功能

sMediaFunctions 结构保存了大容量存储类设备使用的存储介质的访问功能，用户可已根据实际的存储设备来填写，并设定存储介质的固定块大小为 512.

目标芯片的 IAP 程序

用户需要更具 APP 的实际地址修改程序的起始地址和大小，相关描述在 app_update_if.c（APP_START_ADDRESS）中。

9.2 例程演示

将开发板连接到 PC 之后，会在 Windows 系统弹出名称为“BOOTLOADER”的盘符。如图所示

图 9-2 windows 系统盘符

打开此 U 盘，READY.TXT 表示引导程序已就绪，可以接收固件。

图 9-3 打开 U 盘

复制待升级固件到 U 盘内，windows 会重新弹出此盘符，U 盘中文件更新为 SUCCESS.TXT 表示升级成功，更新为 FAILED.TXT 表示升级失败。

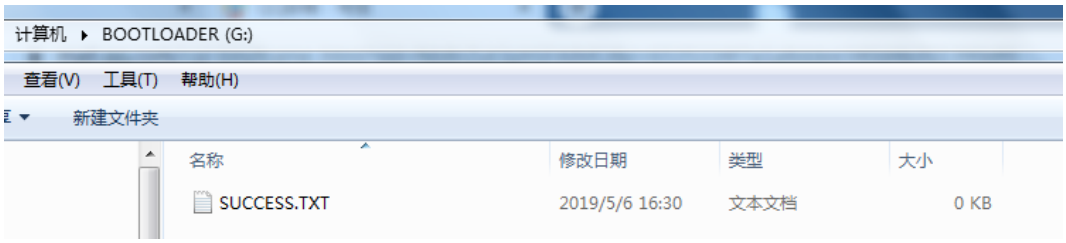


图 9-4 固件升级成功

第10章 BULK设备例程

10. 1 简介

BULK 设备例程中，为用户展示 BULK 类自定义设备的使用方法。本例程需要安装驱动（驱动放在 ES32_SDK\Utilities\usb_windows_drivers 文件夹中，第一次使用时需要引导安装驱动）。

10. 1. 1 使用方法

- 打开 ES32_SDK\Projects\ES32F0271\Applications\USB\usbhos\usbdev_bulk 路径下的代码工程，编译下载；
- 本例程可以直接使用 ES-LinkII 或者 J-Link 供电，也可以 USB 外部电源供电，当使用 USB 供电时，如使用 J-link 需要把 J-link 拔出，因为有的版本 J-link 的 RST 引脚上有下拉电阻；
- 按下开发板复位按钮，使用 USB 线将开发板连接到 PC 上；
- 连接到 PC 之后，会在设备管理器中可以找到设备，设备枚举完成用户可以通过 ES_USB_LAB 进行演示。

10. 1. 2 程序接口

用户在参考本例程配置自己的设备时，主要修改回调函数的完成、设备相关描述符重新配置。

回调函数

回调函数用于 USB 协议栈通知上层应用一些协议栈的处理事件，例如当 USB 设备成功连接时，USB 协议栈会向上层应用发送“USB_EVENT_CONNECTED”事件。USB 协议栈设备的回调函数格式为：

```
typedef uint32_t (* tUSBCallback)(void *pvCBData, uint32_t ui32Event,
                                  uint32_t ui32MsgParam, void *pvMsgData);
```

其中参数 ui32Event 便是事件类型。

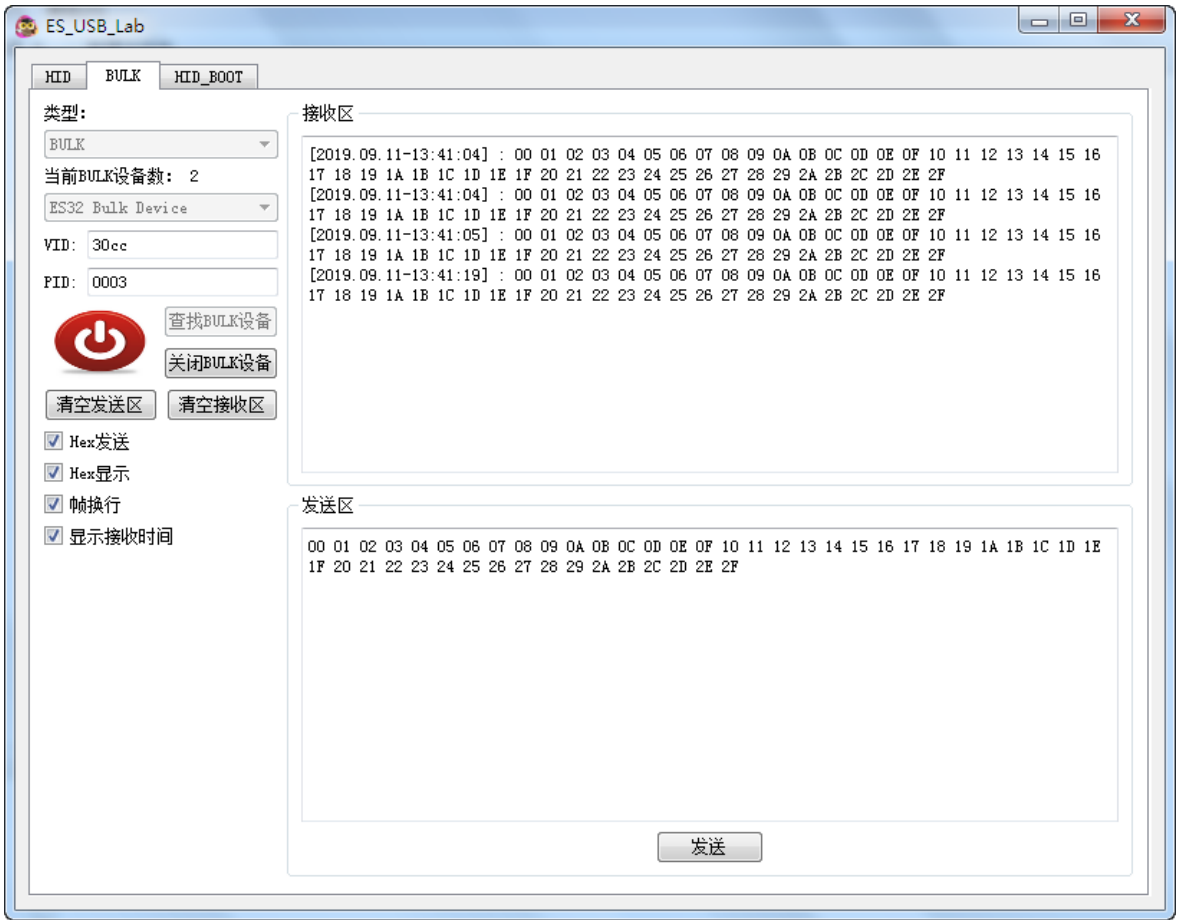
在 BULK 自定义的例程中回调函数放在 usblib_callback.c，回调为 RxHandler ()函数，用户可以参考此回调函数，选取合适的事件来重新定义设备的回调函数。

描述符

设备相关的字符串描述符在 bulk_des.c 文件中，有语言支持描述符、制造商描述符、产品描述符、设备序列号描述符、接口字符串描述符、配置字符串描述符，用户可以修改这些字符串描述符来描述自己的设备，这里的描述的主要是在显示在设备列表中的一些信息，如果用户需要修改标准描述符的话就需要在各设备类层、设备层驱动的设备、配置、接口端点描述符中修改一些信息。

10. 2 例程演示

打开上位机选择 BULK 一栏，查找到自定义 HID 设备“ES32 Bulk Device”。打开此设备，在发送区写入数据，点击“发送”按钮，在接收区会收到发送区的数据。



10-1 BULK 设备上位机演示

第11章 HID鼠标主机例程

11. 1 简介

本例程创建了一个 HID 鼠标主机例程，可以使用 HID 鼠标设备例程与之配合使用。

11. 1. 1 使用方法

- 打开 ES32_SDK\Projects\ES32F0271\Applications\USB\usbhos\ hid_mouse 路径下的代码工程，编译下载；
- 本例程可以直接使用 ES-LinkII 或者 J-Link 供电，也可以 USB 外部电源供电，当使用 USB 供电时，如使用 J-link 需要把 J-link 拔出，因为有的版本 J-link 的 RST 引脚上有下拉电阻；
- 打开串口助手，将开发板串口连接到 PC 端。
- 按下开发板复位按钮，使用另一块烧录 HID 鼠标设备程序的开发板连接到此开发板；
- HID 鼠标设备接入 HID 鼠标主机后会枚举，枚举成功 USB 主机会将接下来设备传入的鼠标数据通过串口（PB6、PB7）打印出来。

11. 1. 2 程序接口

USB 主机的驱动程序和设备驱动结构以及使用方法存在一定的差别与联系。在 USB 主机和设备驱动中同样都存在 USB 协议栈的事件回调，USB 设备类驱动提供相关接口将回调函数作为参数传入到 USB 协议栈底层，在 USB 主机驱动中同样也是需要将回调函数传入到 USB 协议栈底层。与 USB 设备类驱动不同的是，USB 主机驱动包含了事件驱动以及设备的相关驱动，设备相关的驱动主要处理 USB 设备传输的数据，事件驱动主要处理 USB 协议栈的一些事件，如前面设备驱动中提到过的连接事件。

回调函数

回调函数用于 USB 协议栈通知上层应用一些协议栈的处理事件，例如当 USB 设备成功连接时，USB 协议栈会向上层应用发送“USB_EVENT_CONNECTED”事件。USB 协议栈设备的回调函数格式为：

```
typedef uint32_t (* tUSBCallback)(void *pvCBData, uint32_t ui32Event,
                                  uint32_t ui32MsgParam, void *pvMsgData);
```

其中参数 ui32Event 便是事件类型。

在 HID 鼠标主机的例程中回调函数放在 usblib_callback.c 文件中，主机中使用的回调为 MouseCallback ()函数。

设备驱动

USB 主机设备类驱动架构如图 8-1 所示，ui32InterfaceClass 为接口类型，pfnOpen 以及 pfnClose 在每个主机设备的驱动中都会给出，例如鼠标驱动中相应的函数为 hid_driver_open 以及 hid_driver_close，pfnIntHandler 是事件驱动的参数，当驱动类型为设备类驱动时就不需要为此项赋值。

```
typedef struct
{
    //
    //! The interface class that this device class driver supports.
    //
    uint32_t ui32InterfaceClass;
    //
    //! The function is called when this class of device has been detected.
    //
    void *(*pfnOpen)(tUSBHostDevice *psDevice);
    //
    //! The function is called when the device, originally opened with a call
    //! to the pfnOpen function, is disconnected.
    //
    void (*pfnClose)(void *pvInstance);
    //
    //! This is the optional interrupt handler that will be called when an
    //! endpoint associated with this device instance generates an interrupt.
    //
    void (*pfnIntHandler)(void *pvInstance);
}
tUSBHostClassDriver;
```

图 11-1 主机驱动架构

事件驱动

当驱动为事件驱动时，只需要为 pfnIntHandler 项赋值，在本例程中事件驱动为 USBHCDEvents()函数。

主机创建步骤

主机协议栈工作也需要设定工作模式，这里使用 usb_stack_mode_set()函数配置协议栈工作的模式；usbhcd_register_drivers()函数用于向协议栈注册已经准备好的事件以及设备类驱动；在注册完驱动后使用 usbh_mouse_open()函数初始化鼠标主机，这里注意在函数参数中例程传入了一个 MouseCallback()的回调函数入口，主要用于鼠标相关的事件处理，例如鼠标接收到数据的处理；以上工作准备好之后调用 usbhcd_init()函数初始化主机协议栈。USB 协议栈主机部分在使用时需要间隔一段时间调用 usbhcd_main()函数，主要作用为调度主机程序驱动的运作。

11. 2 例程演示

将鼠标设备接入开发板并移动鼠标，开发板会将鼠标的数据通过串口打印出来。

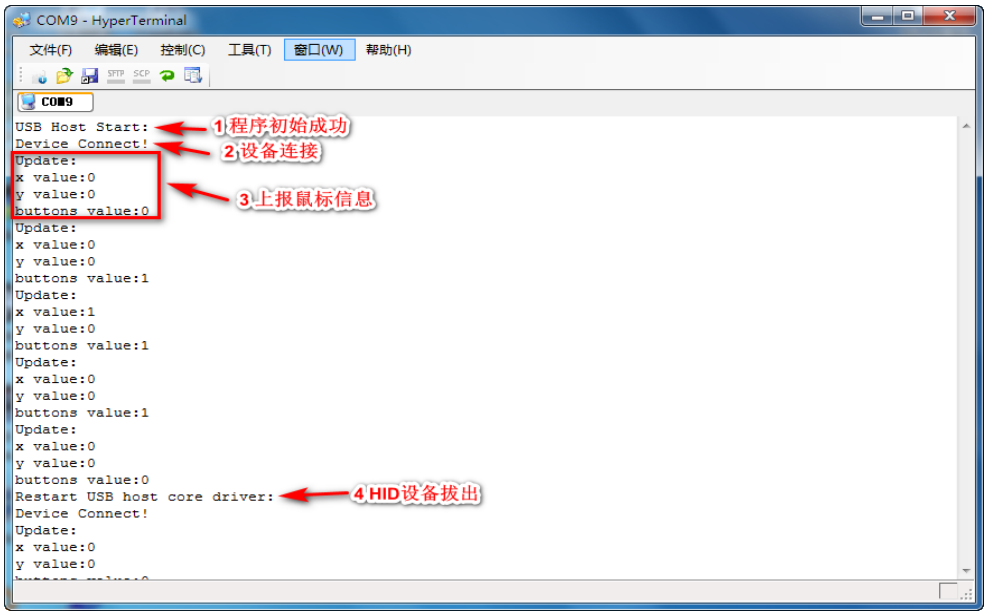


图 11-2 HID 鼠标主机演示

第12章 MSC主机例程

12. 1 简介

MSC 设备例程创建了大容量存储设备（mass storage device）的主机例程，此程序有 bootloader 的功能，从 U 盘中读取二进制文件并将其编程到 MCU FLASH 的另一个位置。将用户应用程序编程到 FLASH 后，该程序将始终运行用户加载的应用程序，直到加载新应用程序。

12. 1. 1 使用方法

- 打开 ES32_SDK\Projects\ES32F0271\Applications\USB\usbhos\usbhos_msc_bootloader 路径下的代码工程，编译下载；
- 本例程可以直接使用 ES-LinkII 或者 J-Link 供电，也可以 USB 外部电源供电，当使用 USB 供电时，如使用 J-link 需要把 J-link 拔出，因为有的版本 J-link 的 RST 引脚上有下拉电阻；
- 主机设备会一直等待 USB 接口插入 U 盘，一旦检测到 U 盘后，将在它的根目录中搜索特定的文件名，默认情况下为 FIRMWARE.BIN。此文件必须是要加载的程序的二进制文件。

  此 U 盘必须格式化为 FAT16 或 FAT32 文件系统，U 盘最好使用金士顿等品牌，且二进制文件必须位于根目录中。
- 如果升级成功，此设备就会进入 APP 程序,并保持在 APP 中运行。
- 按住开发板上五向按键的中间键的同时，拔下开发板并将其重新插入计算机中(或按下开发板上的复位按钮)，设备会重新进入 bootloader 程序中。

12. 1. 2 程序接口

USB 主机的驱动程序和设备驱动结构以及使用方法存在一定的差别与联系。在 USB 主机和设备驱动中同样都存在 USB 协议栈的事件回调，USB 设备类驱动提供相关接口将回调函数作为参数传入到 USB 协议栈底层，在 USB 主机驱动中同样也是需要将回调函数传入到 USB 协议栈底层。与 USB 设备类驱动不同的是，USB 主机驱动包含了事件驱动以及设备的相关驱动，设备相关的驱动主要处理 USB 设备传输的数据，事件驱动主要处理 USB 协议栈的一些事件，如前面设备驱动中提到过的连接事件。

回调函数

回调函数用于 USB 协议栈通知上层应用一些协议栈的处理事件，例如当 USB 设备成功连接时，USB 协议栈会向上层应用发送“USB_EVENT_CONNECTED”事件。USB 协议栈设备的回调函数格式为：

```
typedef uint32_t (* tUSBCallback)(void *pvCBData, uint32_t ui32Event,
                                  uint32_t ui32MsgParam, void *pvMsgData);
```

其中参数 ui32Event 便是事件类型。

在 MSC 主机的例程中回调函数放在 usblib_callback.c 文件中，主机中使用的回调为 MscCallback ()函数。

设备驱动

USB 主机设备类驱动架构如图 8-1 所示，ui32InterfaceClass 为接口类型，pfnOpen 以及 pfnClose 在每个主机设备的驱动中都会给出，例如 MSC 驱动中相应的函数为 usbhmsc_open 以及 usbhmsc_close，pfnIntHandler 是事件驱动的参数，当驱动类型为设备类驱动时就不需要为此项赋值。

```
typedef struct
{
    //
    //! The interface class that this device class driver supports.
    //
    uint32_t ui32InterfaceClass;
    //
    //! The function is called when this class of device has been detected.
    //
    void *(*pfnOpen)(tUSBHostDevice *psDevice);
    //
    //! The function is called when the device, originally opened with a call
    //! to the pfnOpen function, is disconnected.
    //
    void (*pfnClose)(void *pvInstance);
    //
    //! This is the optional interrupt handler that will be called when an
    //! endpoint associated with this device instance generates an interrupt.
    //
    void (*pfnIntHandler)(void *pvInstance);
}
tUSBHostClassDriver;
```

图 12-1 主机驱动架构

事件驱动

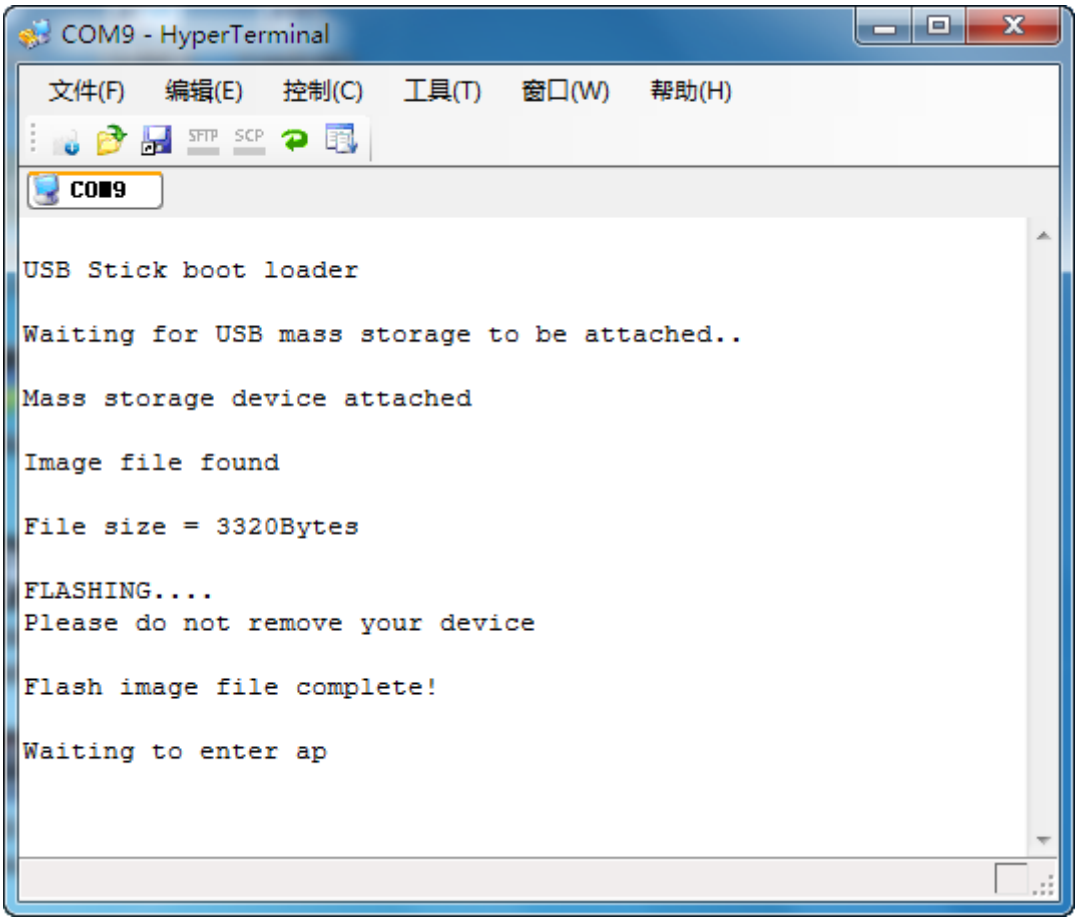
当驱动为事件驱动时，只需要为 pfnIntHandler 项赋值，在本例程中事件驱动为 USBHCDEvents()函数。

主机创建步骤

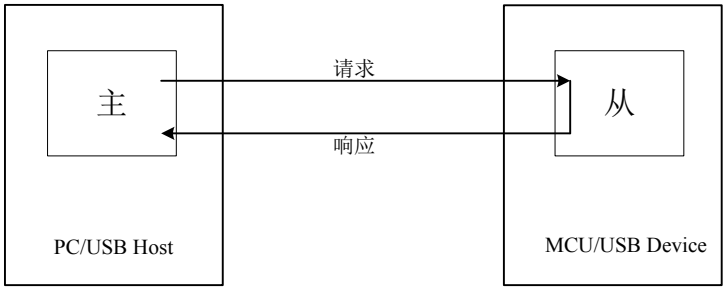
主机协议栈工作也需要设定工作模式，这里使用 usb_stack_mode_set()函数配置协议栈工作的模式；usbhcd_register_drivers()函数用于向协议栈注册已经准备好的事件以及设备类驱动；在注册完驱动后使用 usbhmsc_drive_open()函数初始化 MSC 主机，这里注意在函数参数中例程传入了一个 MSCCallback()的回调函数入口，主要用于 MSC 相关的事件处理，例如 MSC 接收到数据的处理；以上工作准备好之后调用 usbhcd_init()函数初始化主机协议栈。USB 协议栈主机部分在使用时需要间隔一段时间调用 usbhcd_main()函数，主要作用为调度主机程序驱动的运作。

12. 2 例程演示

将 U 盘接入开发板，如果在 U 盘的根目录中找到“FIRMWARE.BIN”文件，开发板会更新并运行此文件。



附录1　HID Bootloader自定义通信协议



附录1. 1　基本格式

所有命令都以单个 ASCII 字符串形式发送。命令以“？”或“！”开始，“？”表示请求命令，“！”表示响应命令。“？”或“！”紧接着的一个 ASCII 码为命令码。一条命令中的的多个参数码或返回码用空格间隔。参数码和返回码均用 ASCII 字符串表示，数据是以十六进制编码格式表示。所有命令响应都是以<CR><LF>结束的，多余的 <CR> 和 <LF> 将被忽略。

附录1. 2　请求命令格式

“？命令码□参数码 0□参数码 1□...□参数码 n<CR><LF>”“HEX 数据+累加和”（数据只适用于写命令）。

附录1. 3　响应命令格式

“！命令码 0□返回码 0□返回码 1□...□返回码 n<CR><LF>”“HEX 数据+累加和”（数据只适用于读命令）。

附录1. 4　数据格式

数据流采用 HEX 编码格式。每次传送的长度都不应超过 64 个字节，接收器应当将该校验和与接收字节的校验和进行比较。如果校验和匹配，接收器响应“OK<CR><LF>”来继续下一次发送。如果校验和不匹配，接收器响应“RESEND<CR><LF>”。作为响应，发送器应当重新发送那些字节。

附录1. 5　命令集

下面的命令是主机发送的命令。每个命令都支持具体的返回代码。当从机接收到未定义命令时，从机会发送返回代码 INVALID_COMMAND。命令和返回代码为 ASCII 格式。只有当接收到的命令执行完毕时，从机才会发送 OK，这时主机才能发送新的命令。但“”、“”、“”和“”命令除外。

附录1. 5. 1　写数据命令

◆　命令格式

命令	W
输入参数	参数 0（起始写入地址）□参数 1（数据总长度）
返回代码	OK|PARAM_ERROR

描述	固件更新时使用，写命令发送地址和长度后再通过发送数据命令进行数据传送

附录1. 5. 2 发送数据命令

◆ 命令格式

命令	D
输入参数	参数 0: 序号/地址(ASCII) □参数_1: 长度(ASCII)<CR><LF> 数据(nbyte HEX) 累加和(1byte HEX)
返回代码	参数 0: 序号/地址(ASCII) □参数_1: OK/ERR/OVER
描述	发送数据命令 参数 0: 目前保持为 0，

附录1. 6 命令返回代码

◆ 返回代码总览

返回字符串	描述
OK	执行成功
CMD_INVALID	无效的命令
PARAM_ERROR	参数错误
COUNT_ERROR	字节数错误
ADDR_ERROR	地址错误
BUSY	硬件忙
*	有后续帧
CONFIG_FAIL	配置失败
CHECK_ERROR	数据校验错误
SLAVE_NACK	从机无应答
SLAVE_BUSY	从机忙