STM32F10xxx 全速 USB 设备开发套件用户手册

孙旭朋　译

简介

STM32 全速 USB 设备开发套件是一个完整的固件和软件开发包，包括例子和演示所有 USB 传输类型（控制，中断，批量和同步）。它支持所有的 STM32 微控制器系列。

STM32 全速 USB 设备开发套件旨在让每个 USB 传输类型的设备库至少有一个固件演示程序。

该文件提出了一个描述的 STM32 全速 USB 设备的所有组件的开发工具包，包括：

■STM32 USB-FS 设备库：默认端点和标准请求有关的所有进程

■设备固件升级（DFU）演示：控制传输

■操纵杆鼠标演示：中断传输

■自定义 HID 演示：中断传输

■大容量存储演示：批量传输

■虚拟 COM 端口演示：中断和批量传输

■USB 音频扬声器演示（USB 扬声器）：同步传输

■USB 音频流演示：同步传输

表 1：使用产品

类型	产品子类
微控制器	STM32 F1 主流产品
	STM32 L1 超低功耗产品

目录

1 STM32 微控制器系列综述

在此文件中，STM32 是指以下设备：

■小容量设备：STM32F101xx，STM32F102xx 和 STM32F103xx 微控制器，flash 大小在 16-32KB 之间。

■中容量设备：STM32F101xx，STM32F102xx 和 STM32F103xx 微控制器，flash 大小在 64-128KB 之间。

■大容量设备：STM32F101xx 和 STM32F103xx 微控制器，flash 大小在 256-512 KB 之间。

■超大容量设备：STM32F101xx 和 STM32F103xx 微控制器，flash 大小在 512-1024KB 之间。

■互联型设备：STM32F105xx 和 STM32F107xx 微控制器。

■中容量低功耗设备：STM32L15xx 微控制器，flash 大小在 64-128KB 之间。

■增强型低功耗中容量设备：STM32L15xx 和 STM32L162xx 微控制器，flash 大小为 256 KB。

■低功耗大容量设备：STM32L15xx 和 STM32L162xx 微控制器，flash 大小为 384KB。

2 STM32 USB-FS-设备固件库

本节主要介绍了 STM32 USB 2.0 全速设备和 USB2.0 OTG 全速设备外设的固件接口（USB-FS-设备库）。在文章的其余部分，分别介绍了 USB_FS_Device 外设和 OTG_FS_Device。

USB-FS_Device 的设备库支持低，中，高容量器件，同时支持 USB2.0 全速设备开发。OTG-FS_Device 是 OTG-FS 外设库的一部分，可用于互联性设备，并支持主机，从机和双重角色。USB-FS-设备库支持 USB-FS_Device 的外设和 OTG-FS_Device 的外设，无论是在设备模式下，还是硬件抽象层。

这个固件库的主要目的是为基于 STM32 微控制器的 USB-FS_Device 和 OTG-FS_Device 的外围设备提供资源，使其应用程序的开发更加容易。

2.1 USB 应用层次

图 1 展示了一个典型的 USB 应用与 USB-FS-Devicelibrary 的关系图。

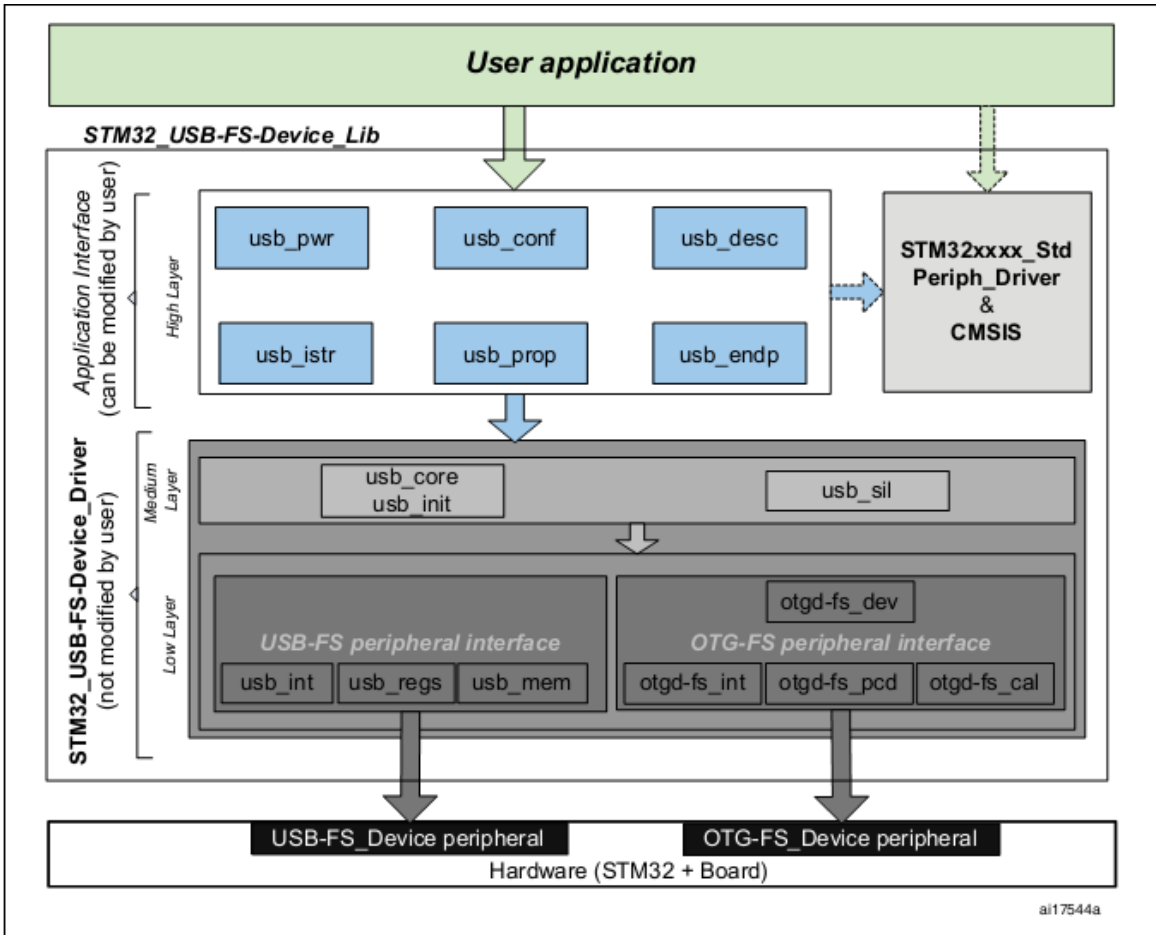

图 1：典型的 USB 应用与 USB-FS-Devicelibrary 的关系图

USB-FS-设备库分为两层：

●STM32_USB-FS_Device_Driver：该层管理 USB 外设（USB-FS_Device 外设及 OTG-FS_Device 外设）和 USB 标准协议的直接通信。STM32_USB FS_Device_Driver 是符合 USB 2.0 规范的，它与 STM32 的标准外设库是分开的。

●Application Interface layer（应用程序接口层）：该层为用户提供了一个完整的固件库内核和最终应用之间的接口。

当使用 STM32 互联性产品时，OTG-FS_外围设备接口层被加载（通过在编译时定义）和作为外设接口层。USB-FS 外围层的功能函数不会被加载。

当使用其它 STM32 设备，只有 USB-FS_外围设备接口层被加载（通过在编译时定义）和作为外设接口层。

然而，核心的库（usb_core（C，h），usb_istr（h）：C，...）是常见的，保持不变。

警告：在互联型设备中任何引用“_USB-FS_Device 外设”的声明是无效的，在所有其他设备中任何引用“OTG FS_Device 外设”的声明也是无效的。

注：应用程序接口层和最终应用可以与标准外设库通信来管理应用程序的硬件需求。

在接下来的章节中将对这些层和编码规则进行详细描述。

图 2 显示了 USB-FS-设备库范例和子文件夹的所有内容。

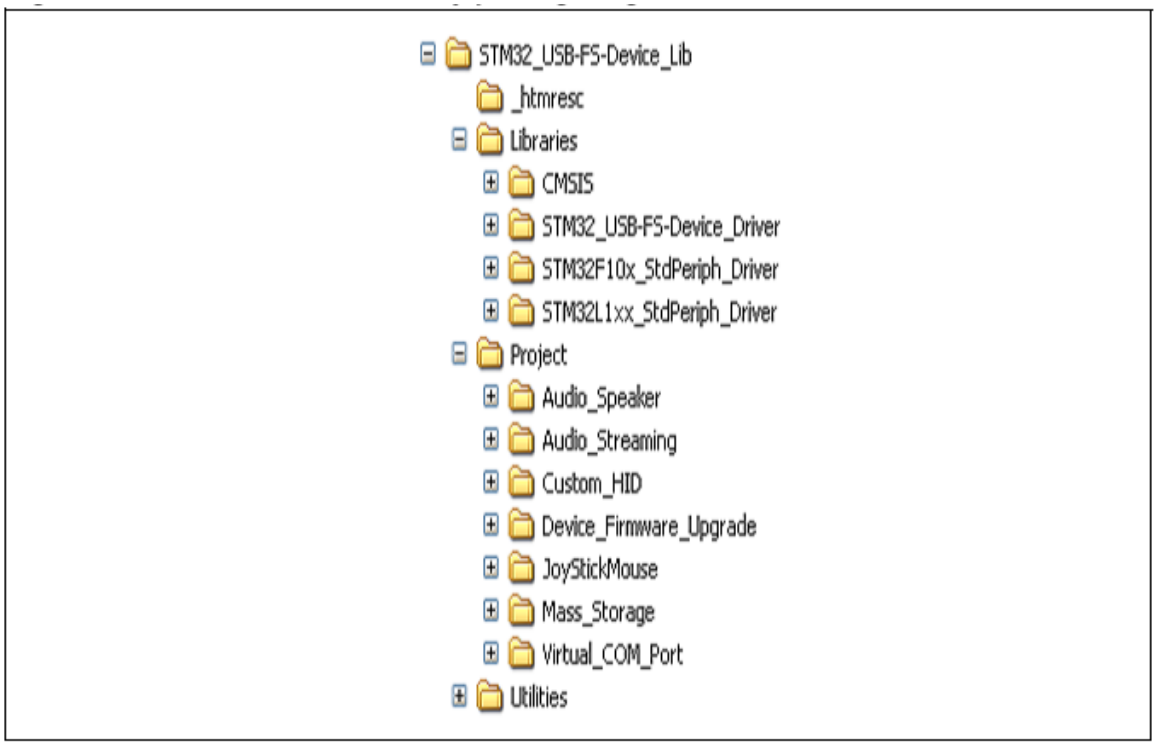

图 2：USB-FS-设备库范例和子文件夹

2.2 USB-FS 设备外围接口

表 2 列出了 USB-FS 设备外围接口模块。

表 2：USB-FS 设备外围接口模块

文件	描述
usb_reg (.h, .c)	硬件抽象层
usb_int.c	端点传输中断服务程序
usb_mem(.h,.c)	数据传输管理（内存区域）

2.2.1 usb_reg(.h, .c)

Usb_reg 模块实现了硬件抽象层的基本功能，它提供了一组用于访问 USB-FS 外设寄存器。

注：函数有两种调用方式：

- As a macro: the call is: _NameofFunction(parameter1,...)
- As a subroutine: the call is: NameofFunction(parameter1,...)

通用寄存器的函数

这些功能可以用来设置或得到不同的常见的 USB-FS 外设寄存器。

表 3：通用寄存器函数

寄存器	函数
CNTR	void SetCNTR (uint16_t wValue)
	uint16_t GetCNTR (void)
ISTR	void SetISTR (uint16_t wValue)
	uint16_t GetISTR (void)
FNR	uint16_t GetFNR (void)
DADDR	void SetDADDR (uint16_t wValue)
	uint16_t GetDADDR (void)
BTABLE	void SetBTABLE (uint16_t wValue)
	uint16_t GetBTABLE (void)

端点寄存器的功能

所有端点寄存器的操作，可以得到的 SetENDPOINT 和 GetENDPOINT 功能。然而，许多功能都源于这些直接作用于特定的字段可提供的优势。

（a） 设置/获取端点值

SetENDPOINT： void SetENDPOINT(uint8_t bEpNum,uint16_t wRegValue)

bEpNum = Endpoint number, wRegValue = Value to write

GetENDPOINT： uint16_t GetENDPOINT(uint8_t bEpNum)

bEpNum = Endpoint number

return value: the endpoint register value

(b) 端点类型方式字段

端点寄存器类型方式字段可以承担如下定义的值：

```
#define EP_BULK          (0x0000)   // Endpoint BULK
#define EP_CONTROL       (0x0200)   // Endpoint CONTROL
#define EP_ISOCHRNOUS    (0x0400)   // Endpoint ISOCHRONOUS
#define EP_INTERRUPT     (0x0600)   // Endpoint INTERRUPT
```

SetEPType : void SetEPType (uint8_t bEpNum, uint16_t wtype)

bEpNum = Endpoint number, wtype = Endpoint type (value from the above define's)

GetEPType : uint16_t GetEPType (uint8_t bEpNum)

bEpNum = Endpoint number

return value: a value from the above define's

(c) 端点状态字段

STAT_TX/ STAT RX 字段的端点寄存器可以承担如下定义的值：

```
#define EP_TX_DIS          (0x0000)   // Endpoint TX DISabled
#define EP_TX_STALL        (0x0010)   // Endpoint TX STALLed
#define EP_TX_NAK          (0x0020)   // Endpoint TX NAKed
#define EP_TX_VALID        (0x0030)   // Endpoint TX VALID
#define EP_RX_DIS          (0x0000)   // Endpoint RX DISabled
#define EP_RX_STALL        (0x1000)   // Endpoint RX STALLed
#define EP_RX_NAK          (0x2000)   // Endpoint RX NAKed
#define EP_RX_VALID        (0x3000)   // Endpoint RX VALID
```

SetEPTxStatus :void SetEPTxStatus(uint8_t bEpNum,uint16_t wState)

SetEPRxStatus: void SetEPRxStatus(uint8_t bEpNum,uint16_t wState)

bEpNum = Endpoint number, wState = a value from the above define's

GetEPTxStatus:uint16_t GetEPTxStatus(uint8_t bEpNum)

GetEPRxStatus:uint16_t GetEPRxStatus(uint8_t bEpNum)

bEpNum = endpoint number

return value:a value from the above define’s

缓冲描述符表功能

使用这些函数设置或获取端点接收和发送缓冲区的地址和大小。

a) Tx/Rx buffer address fields

SetEPTxAddr : void SetEPTxAddr(uint8_t bEpNum,uint16_t wAddr);

SetEPRxAddr : void SetEPRxAddr(uint8_t bEpNum,uint16_t wAddr);

bEpNum = endpoint number

wAddr = address to be set (expressed as PMA buffer address)

GetEPTxAddr : uint16_t GetEPTxAddr(uint8_t bEpNum);

GetEPRxAddr : uint16_t GetEPRxAddr(uint8_t bEpNum);

bEpNum = endpoint number

return value : address value (expressed as PMA buffer address)

b) Tx/Rx buffer counter fields

SetEPTxCount : void SetEPTxCount(uint8_t bEpNum,uint16_t Count);

SetEPRxCount : void SetEPRxCount(uint8_t bEpNum,uint16_t Count);

bEpNum = endpoint number

wCount = counter to be set

GetEPTxCount : uint16_t GetEPTxCount(uint8_t bEpNum);

GetEPRxCount : uint16_t GetEPRxCount(uint8_t bEpNum);

bEpNum = endpoint number

return value : counter value

双缓冲端点函数

在批量和同步模式中为了获得较高的数据传输率必须用双缓冲模式进行编程。在这种操作模式下，一些端点寄存器和缓冲描述符表有不同的含义。

为了简化此功能的使用，已经开发了几种功能。

SetEPDoubleBuff：在批量模式下编程时端点可以通过设置 EP-KIND 位设置为双缓冲模式。函数 SetEPDoubleBuff（）用来完成这个任务。

SetEPDoubleBuff : void SetEPDoubleBuff(uint8_t bEpNum);

bEpNum = endpoint number

FreeUserBuffer：在双缓冲模式下的端点成为单方向的，缓冲描述符单元未使用的方向可以用于处理第二个缓冲区。

地址和计数器必须以不同的方式处理。Rx 和 Tx 地址和计数器单元变为缓存 0 和缓存 1。函数在这种操作模式下致力于提供在库操作。

在批量传输时一个缓冲区被使用，另一个缓冲区保留给其他的应用程序。用户应用程序数据到来之前，需要一个缓冲区。预留给应用程序的缓冲区在一定时间后将被释放。

释放缓冲区从应用程序中使用的 FreeUserBuffer 提供缓冲功能：

FreeUserBuffer: void FreeUserBuffer(uint8_t bEpNum, uint8_t bDir);

bEpNum = endpoint number

2.2.2 usb_int (.h , .c)

usb_int 模块处理正确的传输中断服务程序，它提供了 USB 设备协议事件和库之间的联系。

STM32 USB-FS 设备外设提供两种正确传输例程：

●低优先级中断：由函数 CTR_LP（）管理，用于控制，中断，批量传输模式（单缓冲模式）。

●高优先级中断：由函数 CTR_HP（）管理，使用于更快的传输模式，如同步，批量（双缓冲模式）。

2.2.3 usb_mem (.h , .c)

该函数把用户存储区的缓冲区中的数据复制到数据包的内存区域（PMA），反之亦然。它提供了两种不同的功能：

void UserToPMABufferCopy(uint8_t *pbUsrBuf,uint16_t wPMABufAddr, uint16_t wNBytes);

void PMAToUserBufferCopy(uint8_t *pbUsrBuf,uint16_t wPMABufAddr, uint16_t wNBytes);

其中：

pbUsrBuf 是在用户存储区中，一般在产品的 SRAM 的指针。

wPMABufAddr 是 PMA 地址。（USB 内存区域中 512 字节的数据包）。

wNBytes 是要被复制的字节数。

2.3 OTG-FS 设备外围接口

表 4 给出了 OTG-FS_Device 外设接口模块。

表 4：OTG-FS_Device 外设接口模块

文件	描述
otgd_fs_dev (.h , .c)	OTG-FS_Device 高层管理设备模式
otgd_fs_int (.h , .c)	OTG-FS_Device 中断处理程序
otgd_fs_pcd (.h , .c)	OTG FS_Device 低层管理模式
otgd_fs_cal (.h , .c)	OTG-FS_Device 外设控制及状态管理（低层）。
otgd_fs_regs.h	OTG-FS_Device 外设寄存器定义

2.3.1 otgd_fs_dev(.h, .c)

该函数提供了 OTG FS_Device 外围设备模式主要的处理功能。它也包含了端点处理的主要功能。

主要功能函数在表 5 中列出。

表 5：otgd_fs_dev 函数

函数	描述
void OTGD_FS_Dev_Init(void)	复位和初始化所有的 OTG FS_Device 的外围设备，并初始化端点 0。
void OTG_DEV_EP_Init(uint8_t bEpAdd, uint8_t bEpType, uint16_t wEpMaxPackSize)	配置和使能所选端点（此功能不用于端点 0）。
void OTGD_FS_Dev_Connect(void)	使能 OTG-FS_Device 外设上拉电阻
Void OTGD_FS_Dev_Disconnect(void)	失能 OTG-FS_Device 外设上拉电阻
uint32_t OTGD_FS_GetEPTxStatus (uint8_t bEpNum)	返回所选择端点的发送状态(valid, stall, NaK, disable).
uint32_t OTGD_FS_GetEPRxStatus (uint8_t bEpNum)	返回所选择端点的接收状态(valid, stall, NaK, disable).
void OTGD_FS_SetEPTxStatus (uint8_t bEpNum, uint32_t Status)	配置所选端点的发送状态(valid, stall, NaK, disable).
void OTGD_FS_Set EPRxStatus (uint8_t bEpNum, uint32_t Status)	配置所选端点的接收状态(valid, stall, NaK, disable).

2.3.2 otgd_fs_int(.h, .c)

该函数处理 OTG-FS_Device 外设产生的不同中断服务程序。它提供了 USB 协议事件和库内核之间的联系。

STM32 的 OTG-FS_Device 外设提供如表 6 中列出的中断服务程序。

表 6：otgd_fs_int 函数

函数	描述
uint32_t OTGD_FS_HandleInEP_ISR (void);	IN 端点中断处理。
uint32_t OTGD_FS_HandleOutEP_ISR (void);	OUT 端点中断处理。
uint32_t OTGD_FS_HandleSof_ISR (void);	帧开始中断的处理
uint32_t OTGD_FS_HandleRxStatusQueueLevel_ISR (void);	RX 状态队列优先级中断处理（接收到的数据在 USB 内部 RAM）。
uint32_t OTGD_FS_HandleEnumDone_ISR(void);	枚举完成中断处理。
uint32_t OTGD_FS_HandleUsbReset_ISR(void);	USB 复位事件中断处理。
uint32_t OTGD_FS_HandleWakeup_ISR(void);	唤醒事件中断处理。
uint32_t OTGD_FS_HandleUSBSuspend_ISR(void);	挂起事件中断处理。
uint32_t OTGD_FS_Handle_EOPF_ISR(void);	预期的周期帧结束中断处理。
uint32_t OTGD_FS_Handle_PTXFEmpty_ISR(void);	周期的 Tx FIFO 空中断处理。
uint32_t OTGD_FS_Handle_EarlySuspend_ISR(void);	早期暂停事件中断处理。
uint32_t OTGD_FS_Handle_NPTxFE_ISR(void);	非定期发送 FIFO 空中断处理。

2.3.3 otgd_fs_pcd(.h, .c)

该函数是低层接口的 OTG-FS_Device 外围设备模式。它在必要的操作下可以处理硬件设备模式。

2.3.4 otgd_fs_cal(.h, .c)

该函数是低的 OTG FS_Device 外设核心层接口。它可以为控制和状态寄存器，以及外设配置和初始化处理所有必要的功能。

2.3.5 otgd_fs_regs.h

该函数包含的 OTG FS_Device 外设的寄存器定义。

2.4 USB-FS 设备驱动程序介质层

表 7 列出了 USB-FS 设备驱动程序的分层模块：

表 7：USB-FS 设备驱动程序的分层模块

文件	描述
usb_init (.h,.c)	USB 设备初始化的全局变量
usb_core (.h , .c)	USB 协议管理（符合 USB 2.0 规范的第 9 章）
usb_sil (.h,.c)	简化读取和写入访问的端点（抽象层两个 USB-FS_Device 的 OTG FS_Device 外围设备）
usb_def.h / usb_type.h	在库中使用 USB 的定义和类型

2.4.1 usb_init(.h,.c)

该函数设置将在库中使用的初始化程序和全局变量。

2.4.2 usb_core (.h , .c)

该函数是库的“核心”。它实现了所有 USB 2.0 规范第 9 章中描述的功能。

与 USB 标准请求的控制端点（ENDP0）相关的子例程提供了必要的代码来完成顺序和枚举阶段的任务。

一个状态位来设置事务处理的不同阶段。

USB 核心模块还使用结构体 User_Standard_Requests 实现了动态界面之间的标准请求和用户实现。

用户处理过程在 Device_Property 结构中，完成 USB 核心调度类的具体要求和一些总线事件到用户程序。

下面的段落描述了各种数据和功能所使用的内核结构。

1.设备表结构

内核将保存设备级信息在 Device_Table 结构中。此结构的具体类型：DEVICE。

```
typedef struct _DEVICE {
 uint8_t Total_Endpoint;
 uint8_t Total_Configuration;
```

} DEVICE;

2.设备信息结构

USB 内核将实现 USB 设备的主机与设备结构的安装包。此结构具有类型：DEVICE_INFO。

```
typedef struct _DEVICE_INFO {
 uint8_t USBbmRequestType;
 uint8_t USBbRequest;
 uint16_t_uint8_t USBwValues;
 uint16_t_uint8_t USBwIndexs;
 uint16_t_uint8_t USBwLengths;
 uint8_t ControlState;
 uint8_t Current_Feature;
 uint8_t Current_Configuration;
 uint8_t Current_Interface;
uint8_t Current_AlternateSetting;
 ENDPOINT_INFO Ctrl_Info;
} DEVICE_INFO;
```

在任一 uint16_t 或 uint8_t 格式的 DEVICE_INFO 中某些字段的访问被定义为一个 uint16_t 和_uint8_t 的共同体。

```
typedef union {
 uint16_t w;
  struct BW {
 uint8_t bb1;
 uint8_t bb0;
 } bw;
} uint16_t_uint8_t;
```

其中：

USBbmRequestType 是复制一个安装包中的 bmRequestType。

USBbRequest 是复制一个安装包中的 bRequest。

USBwValues 类型被定义为类型：uint16_t_uint8_t，可以通过 3 个宏来实现：

```
#define USBwValue USBwValues.w
#define USBwValue0 USBwValues.bw.bb0
#define USBwValue1 USBwValues.bw.bb1
```

USBwValue 是复制一个安装包中的 wValue。

USBwValue0 是 wValue 的低字节，USBwValue1 是 wValue 的高字节。

USBwIndexs 定义为 USBwValues 类型并可以通过 3 个宏访问：

```
#define USBwIndex USBwIndexs.w
#define USBwIndex0 USBwIndexs.bw.bb0
#define USBwIndex1 USBwIndexs.bw.bb1
```

USBwIndex 是复制一个安装包中的 wIndex。

USBwIndex0 是 wIndex 的低字节，USBwIndex1 是 wIndex 的高字节。

USBwLengths 被定义为类型 uint16_t_uint8_t，可以通过 3 个宏访问：

```
#define USBwLength USBwLengths.w
#define USBwLength0 USBwLengths.bw.bb0
#define USBwLength1 USBwLengths.bw.bb1
```

USBwLength i 是复制一个安装包中的 wLength。

USBwLength0 和 USBwLength1 分别是 wLength 低字节和高字节。

3. 设备属性结构

必要时，USBcore 将被调用来控制用户程序。用户处理程序在阵列中的设备属性。该结构具有类型：DEVICE_PROP：

```
typedef struct _DEVICE_PROP {
void (*Init)(void);
void (*Reset)(void);
void (*Process_Status_IN)(void);
void (*Process_Status_OUT)(void);
RESULT (*Class_Data_Setup)(uint8_t RequestNo);
RESULT (*Class_NoData_Setup)(uint8_t RequestNo);
```

```
RESULT  (*Class_Get_Interface_Setting)(uint8_t Interface,uint8_t
AlternateSetting);
uint8_t* (*GetDeviceDescriptor)(uint16_t Length);
uint8_t* (*GetConfigDescriptor)(uint16_t Length);
uint8_t* (*GetStringDescriptor)(uint16_t Length);
void* RxEP_buffer; /* This field is not used in current library
version.
It is kept only for compatibility with previous versions */
uint8_t MaxPacketSize;
} DEVICE_PROP;
```

4.用户标准请求结构

用户标准请求结构是用户代码和管理标准要求之间的接口。该结构具有类型：USER_STANDARD_REQUESTS：

```
typedef struct _USER_STANDARD_REQUESTS {
void(*User_GetConfiguration)(void);
void(*User_SetConfiguration)(void);
void(*User_GetInterface)(void);
void(*User_SetInterface)(void);
void(*User_GetStatus)(void);
void(*User_ClearFeature)(void);
void(*User_SetEndPointFeature)(void);
void(*User_SetDeviceFeature)(void);
void(*User_SetDeviceAddress)(void);
} USER_STANDARD_REQUESTS;
```

如果用户希望收到一个标准的 USB 设备请求，就应该在此结构中使用相应的代码实现特定的功能。

应用程序开发人员通过三层结构类型 DEVICE_PROP，Device_Table 和 USER_STANDARD_REQUEST 来管理和请求特定的应用程序功能。这些结构的不同字段的描述在第 2.4.4 节 usb_type.h/ usb_def.h 中。

2.4.3 usb_sil(.h, .c)

usb_sil 模块实现USB-FS_Device 和OTG-FS_Device 外围设备的一个抽象层。它提供了简单的功能，用于访问端点的读取和写入操作。

端点写函数

端点的写操作，可以通过下面的函数完成：

void USB_SIL_Write(uint32_t EPNum, uint8_t* pBufferPointer, uint32_t wBufferSize);

该函数的参数有：

●EPNum：IN 端点的数量相关的写操作

●pBufferPointer：要写入 IN 端点到用户缓冲区的指针。

●wBufferSize：要写入 IN 端点的数据字节数。

根据外围设备接口，此函数用于得到端点缓冲区的地址，并执行数据包的写操作。

端点读函数

端点的读操作，可以通过下面的函数完成：

uint32_t USB_SIL_Read(uint32_t EPNum, uint8_t* pBufferPointer);

该函数的参数有：

●EPNum：OUT 端点的数量相关的读操作

● pBufferPointer:要读取的 OUT 端点到用户缓冲区的指针。

根据不同的外设接口，这个函数执行两个连续的操作：

●获取从主机的相关 OUT 端点接收到的数据。

●复制所接收的数据从 USB 的专用存储器的缓冲区指针

地址。

该函数返回接收的数据字节给用户应用程序。

2.4.4usb_type.h / usb_def.h

该函数提供了在库中使用主要的类型和 USB 定义。

2.5 应用程序接口

该函数为应用程序接口提供一个模板，它们由每个应用程序开发者根据开发的应用程序调用。表 8 中显示了在不同的模块中使用的应用程序接口。

表 8：应用程序模块

函数	描述
usb_conf.h	USB-FS_Device 配置文件
usb_desc (.h, .c)	USB-FS_Device 描述文件
usb_prop (.h, .c)	USB-FS_Device 应用程序特定的性能
usb_endp.c	非控制端点的正确传输中断处理程序
usb_istr (.h,.c)	USB-FS_Device 中断处理程序
usb_pwr (.h, .c)	USB-FS_Device 电源和连接管理函数

2.5.1 usb_conf(.h)

usb_conf.h 用于自定义的 USB 的演示和配置设备如下：

●定义要使用的端点的数量（通过定义 EP_NUM）。

●允许使用端点和事件回调例程的评论相对回调定义（当正确的传输发生在端点 1 时，定义 EP1_IN_Callback 启用并使能此功能，当 SOF 中断发生时，为了使用和实现这个功能，定义了 INTR_SOFINTR_Callback）当一个回调在使用时，其相对 usb_conf.h 文件中的定义应加上注释。然后，它们在用户应用程序应具有相同的名称。（不需要声明回调函数的原型，因为它已经在 usb_istr.h 中声明）。

●对于 USB-FS_Device 设备:

配置 BTABLE 和 PMA 中的所有端点地址（通过修改和/或增加相对地址，定义为：BTABLE_ADDRESS，ENDP0_RXADDR，ENDP0_TXADDR...）。

定义中断并使能他们通过中断屏蔽函数 IMR_MSK。

● 对于 OTG-FS_Device 设备：

通过修改 RX_FIFO_SIZE，TX0_FIFO_SIZE，TX1_FIFO_SIZE... 配置 USB 设备 FIFO 的大小。

配置中断，使他们通过注释的关系定义（取消注释 INTR_SOFINTR 收率定义，使自由度中断，...）。

2.5.2 usb_desc (.h, .c)

usb_desc.c 文件包含所有相关的应用程序的 USB 描述符。用户可根据应用程序的类来设置这些描述符。

在“STM32 USB-FS_Device 开发的工具包”中所有可用的演示程序都在 STM32 设备的唯一 ID 寄存器（12 位）的基础上执行一个唯一的序列号字符串描述符。

序列号字符串描述符的默认值是“STM32”,在 USB 初始化函数 Get_SerialNum（）中读取设备的唯一 ID 寄存器并设置序列号的字符串描述符。

欲了解更多有关设备 ID 的唯一手册，请参阅 STM32 参考手册（RM0008）或 STM32L15xx 的参考手册（RM0038）。

2.5.3 usb_prop (.h , .c)

usb_prop 模块用于实现 Device_Property，Device_Table 和 SER_STANDARD_REQUEST 结构，是使用 USB 的核心。

设备属性的实现

设备属性结构字段的描述如下：

● void Init(void):USB-FSFS_Device 或 OTG FS_Device 的初始化程序。在应用程序管理的初始化过程开始时调用一次。

● void Reset(void) : 复位程序的 USB 外围设备。当宏单元从总线收到 RESET 信号时被调用。用户程序应在此过程中设置默认的控制端点（仅适用于的 USB-FS_Device 的外围设备），并使其成为能接收的端点。

● void Process_Status_IN(void): 回调过程中，它在一个阶段状态完成时被调用。用户程序可以使用这个回调函数执行类和应用程序相关的进程。

● void Process_Status_OUT(void): 回调过程中，它在退出某个阶段状态结束时被调用。于 Process_Status_IN 一样，用户程序可以执行的操作完成后退出某个阶段状态。

● RESULT (see note below) *(Class_Data_Setup)(uint8_tRequestNo) : 回调函数，当一类请求被认可并且这个请求需要一组数据时被调用。

● RESULT (*Class_NoData_S etup)(uint8_t RequestNo): 回调过程中，当一个非标准设备请求被确认并且这个请求不需要一组数据时被调用。

●RESULT(*Class_GET_Interface_Setting)(uint8_tInterface, uint8_tAlternateSetting): 这个程序是用来测试收到的设置接口的标准请求。

● uint8_t* GetDeviceDescriptor(uint16_t Length): 获取设备描述符。

● uint8_t* GetConfigDescriptor(uint16_t Length) : 获取配置描述符。

● uint8_t* GetStringDescriptor(uint16_t Length) :获取字符串描述符

● uint16_t MaxPacketSize: 设备默认控制端点的最大数据包大小。

注：

RESULT 类型如下：

```
typedef enum _RESULT {
USB_SUCCESS = 0,/* request process sucessfully */
USB_ERROR,       /* error
USB_UNSUPPORT,  /* request not supported
USB_NOT_READY/* The request process has not been finished,*/
/* endpoint will be NAK to further requests*/
} RESULT;
```

端点的实现

端点的结构字段的描述如下：

●Total_Endpoint:USB 应用使用的端点数量。

●Total_Configuration:配置的 USB 应用程序的数量。

用户标准要求的实施

这个结构是用来管理用户程序执行后接收到所有标准请求（获取描述符除外）。这种结构的字段是：

● void (*User_GetConfiguration)(void):获取配置的标准请求。

● void (*User_SetConfiguration)(void):设置配置的标准请求。

● void (*User_GetInterface)(void) :获取接口的标准请求。

● void (*User_SetInterface)(void) :设置接口的标准请求。

● void (*User_GetStatus)(void) :获取状态的标准请求。

● void (*User_ClearFeature)(void) :清除功能的标准请求。

● void (*User_SetEndPointFeature)(void):获取设置功能的标准请求。（仅适用于端点接收状态）。

● void (*User_SetDeviceFeature)(void):获取设置功能的标准请求。（仅适用于设备接收状态）。

● void (*User_SetDeviceAddress)(void) :获取设置地址的标准请求。

2.5.4 usb_endp (.c)

USB_endp 被用于：

●处理 USB-FS 设备的端点 0（EP 0）的终端以外的 CTR“正确的转移”程序。

●处理“传输完成”中断服务程序以外的其他端点 0（EP0）的 OTG FS_Device 周边的端点。它也允许接收 FIFO 水平同步端点的中断处理。

为了预处理这些回调函数,命名了如下几个函数：EPx_IN_Callback（IN 传输）或 EPx_OUT_Callback（OUT 传输）或 EPx_RX_ISOC_CALLBACK（同步输出）这些函数都定义在 USB_conf.h 文件中。

2.5.5 usb_istr(.c)

USB_istr 模块提供了一个函数名为 USB_Istr（）处理所有 USB 中断的函数。

对于每一个 USB 中断源，回调例程名为 XXX_Callback（例如，RESET_Callback)，是为了实现用户的中断处理程序。为了使每个回调例程处理，预处理开关命名为 XXX_Callback 必须定义在 USB 配置文件 USB_conf.h 中。

2.5.6 usb_pwr (.h , .c)

此函数为 USB 设备的电源管理函数。它提供的功能如表 9 所示。

表 9：电源管理函数

函数名	描述
RESULT Power_on(void)	电源打开
RESULT Power_off(void)	电源关闭
void Suspend(void)	挂起模式设置
void Resume(RESUME_STATE eResumeSetVal)	处理醒来操作

2.6 使用 STM32 USB-FS-设备库实现 USB-FS 设备应用程序。

2.6.1 无数据类专用请求

2.6.2 如何实现数据类专用请求

在一个事件类中需要数据传输时，用户程序向 USB-FS-移动设备库请求要传输的数据的长度和在内部存储器中的数据位置。这种类型的请求管理函数为：RESULT (*Class_Data_Setup)(uint8_t RequestNo)。

对于每一类数据传输请求的用户建立一个特定的功能与格式：

```
uint8_t* My_First_Data_Request (uint16_t Length)
```

如果这个函数被调用的参数长度等于零，那么设置 pInformation->

Ctrl_Info。Usb_wLength 字段的数据传输，返回一个 NULL 指针。在其他情况下，它返回要传输的数据的地址。下面的 C 代码显示了一个简单的例子：

```
uint8_t* My_First_Data_Request (uint16_t Length)
{
if (Length == 0)
{
  pInformation->Ctrl_Info.Usb_wLength  = My_Data_Length;
 return NULL;
 }
 else
 return (&My_Data_Buffer);
}
```

函数 RESULT (*Class_Data_Setup)(uint8_t RequestNo) 管理所有的数据请求，代码如下：

```
RESULT Class_Data_Setup(uint8_t RequestNo)
{
uint8_t*(*CopyRoutine)(uint16_t);
   CopyRoutine = NULL;
if (My_First_Condition)// test the filds of the first request
CopyRoutine = My_First_Data_Request;
else if(My_Second_Condition) // test the filds of the second request
  CopyRoutine = My_Second_Data_Request;
/*
...   same implementation for each class data requests
...
*/
```

```
if (CopyRoutine == NULL) return USB_UNSUPPORT;
pInformation->Ctrl_Info.CopyData = CopyRoutine;
pInformation->Ctrl_Info.Usb_wOffset = 0;
(*CopyRoutine)(0);
return USB_SUCCESS;
} /*End of Class_Data_Setup */
```

2.6.3 如何管理非控制端点的数据传输

数据传输的管理可以在 usb_end.c 文件中配置通道是不是缺省（端点 0）的。用户可在 usb_conf.h 中取消该行对应的端点（带方向）。

3 操纵杆鼠标例程

该例程应用于意法半导体（ST）STM3210B-EVAL, STM3210C-EVAL, STM3210E-EVAL, STM32L1 52-EVAL and STM32L152D-EVAL 评估板，也可以很容易地适应任何其他硬件。

选用意法半导体（ST）的评估板来运行例程，应在 platform_config.h 文件中取消相应的行。

3.1 描述

一个 USB 鼠标（人机接口设备 HID 类）是一个简单的例子，一个完整的 USB 应用。操纵杆鼠标只使用一个中断端点（端点 1 的 IN 方向）。正常的枚举后，主机请求鼠标的 HID 报告描述符。这个描述符（标准描述）在 usb_desc.c 文件中。

通过通道 1（端点 1）的四个字节获取鼠标指针的位置和主机请求，数据格式如图 3 中所示：

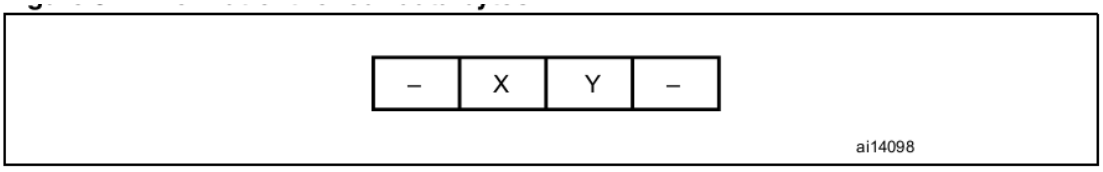

图 3：4 个字节的数据格式

鼠标演示的目的是要根据用户操作一个操纵杆的按钮设置的 X 和 Y 值。JoyState（）函数得到用户的动作，并返回鼠标指针的方向。Joystick_Send（）函数将数据发送到主机，并验证数据完整性。

注：详细信息，请参阅 hw_config.c 文件。

3.2 STM32 在挂起模式下的低功耗管理

这是一个在 USB 挂起/恢复事件下的电源管理的例子，操纵杆鼠标例程支持 STM32 的停止模式的进入和退出。

STM32 停止模式是基于 Cortex-M3 深睡眠模式与外设时钟选通相结合而来的。在停止模式下，所有的在 1.8 V 域时钟都停止了，PLL，HSI 的 RC 振荡器和 HSE 晶体振荡器也被禁止。从停止模式唤醒，可以只使用一个 EXTI 线中断或事件模式。

在本例程中，在停止模式下，电压调节器被配置在低功耗模式，以降低电力消耗，EXTI 线 18（USB-FS_Device/OTG-FS_Device 唤醒线）用于在中断模式中唤醒。

当挂起事件发生在总线上，USB-FS-设备库调用 Enter_LowPowerMode（）函数，（在文件 hw_config.c 中）。进入这个函数后，STM32 处于停止模式。

STM32 保持在停止模式下，直到它接收总线上的一个唤醒事件。在这种情况下，EXTI 线 18 被激活，并且唤醒的 STM32。唤醒后，USB-FS 设备库调用 Leave_LowPowerMode（）函数（在文件 hw_config.c 中），重新配置时钟（重新启用 HSE 和 PLL）。

为了测试、测量低功耗下 USB-FS_Device 挂起这个特性，连接一个电流表到 VDD 跳线（在 STM3210B-EVAL 板上的跳线 JP9， STM3210E-EVAL 板的跳线 JP12，STM3210C-EVAL 板的跳线 J23，STM32L152-EVAL 板的跳线 J4 或 STM32L152D-EVAL 板的跳线 JP10），在 PC 端，使用 USB HS 免费提供的电气测试工具包从 usb.org 中使 STM32 进入挂起/恢复状态。

3.3 远程唤醒

远程唤醒是一个 USB 设备的能力，把一个暂停使用的总线恢复到活动的状态。支持远程唤醒的设备上报告了这一功能的 PC 使用的配置描述符中 bmAttributes 字段（D5 位设置为 1）。

操纵杆例程的关键按钮是用来作为远程唤醒源的。该按钮连接到 STM3210B-EVAL 和 STM3210CEVAL 的 EXTI9 号线（GPIO PB.09），STM3210E-EVAL

的 EXTI 线 8（GPIO PG.08），STM32L152-EVAL 和 STM32L152D 的-EVAL 的 EXTI 线 0（GPIO PA.00）。

当按下该键时，相应的 EXTI 的 ISR 启动的 USB 设备的电源管理状态机使用 Resume（）函数。请注意，远程唤醒可能会被 PC 主机使用 SET_FEATURE 请求禁用，所以，只有当该功能被启用后 EXTI ISR 测试功能才能发送到 PC 远程唤醒信号。

4 自定义 HID 例程

该例程应用于意法半导体（ST）STM3210B-EVAL, STM3210C-EVAL, STM3210E-EVAL, STM32L1 52-EVAL and STM32L152D-EVAL 评估板，也可以很容易地适应任何其他硬件。

选用意法半导体（ST）的评估板来运行例程，应在 platform_config.h 文件中取消相应的行。

4.1 描述

HID（人机接口设备）类主要包括人类所使用的设备控制计算机系统的操作。HID 类设备的典型例子是标准的鼠标，键盘，蓝牙适配器等。

对于 HID 设备类的详细信息，请从 usb.org 网站参阅“设备类定义为 HID1.11”。

自定义 HID 例程是一个简单的演示提供了一个小型 PC 的小程序，举例如何基于原生的 Windows HID 驱动程序创建自定义的 HID。它由简单的 STM32 评估板和 PC 主机采用两个中断通道（IN 和 OUT）之间的数据交换。

交换的数据是相关的 LED 命令，按钮的状态和 ADC 转换值。

有关如何使用 PC 小程序的自定义 HID 的详细信息，请参阅 UM0551 用户手册“USB HID 演示”意法半导体微控制器网站下载：www.st.com。

4.2 描述符的拓扑结构

自定义 HID 拓扑是基于两个中断通道用于处理数据传输，为 7 个不同的报告。下面的图表显示自定义 HID 拓扑。

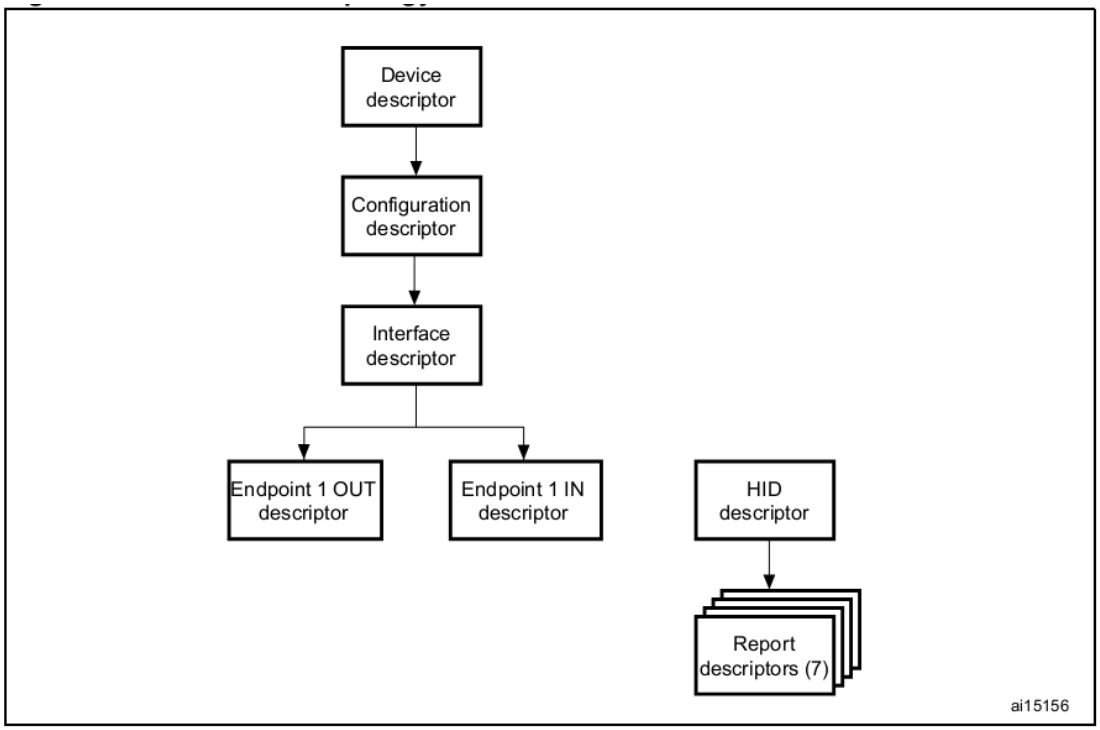

图 4：自定义 HID 拓扑

每一个报告描述符都与评估板上的一个特定的组件对应（指示灯，按钮或 ADC）。以下部分介绍了这些报告的功能。

4.3 自定义 HID 的实现

4.3.1 LED 控制

STM32 评估板有四个 LED。在自定义的 HID 演示，每个 LED 对应于一个特定的报告（报告 1～4）和 LED 状态（ON / OFF），由 PC 小程序设置。

在设备接收数据端点 1 OUT 中，EP1_OUT_Callback（）函数被调用，根据相应的 LED 报告编号派遣接收到的状态。所接收的数据格式如图 5 所示，其中：

●报告编号：报告从 1 到 4。

●LED 状态

0：关

1：开

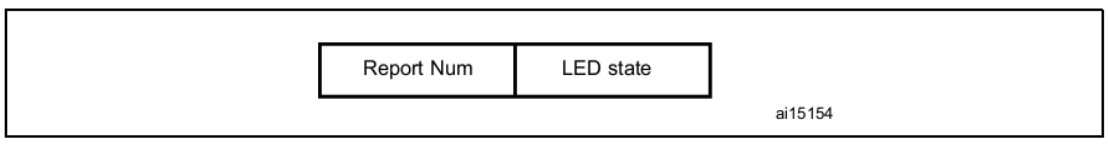

图 5：数据格式

4.3.2 按钮状态报告

STM32 评估板将按钮的状态改变使用的端点 1 报道给 PC 主机。（STM32L152-EVAL 板，左，右摇杆按钮除外）

关键按钮（或右按钮 STM32L152-EVAL 板），对应报告 5，防拆按钮（或左按钮的 STM32L152 板）对应报告 6。当两个按钮中的一个被按下时，该装置的有关报告数和按钮状态发送到主机。图 6 示出了所用的格式，其中：

●报告数：5 或 6

●按钮状态：1：按下

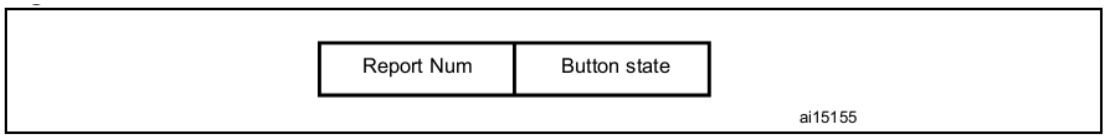

图 6：数据格式

4.3.3 ADC 转换的数据传输

演示这部分包含评估板将其电位计电压转换后传送到 PC 主机。ADC 的配置在连续模式下，DMA 数据传输到 RAM 的变量（ADC_ConvertedValueX）。在每次转换后，测试转换后的值与上一个值比较，如果两个值之间（电位计由用户改变的值）有差别，新的值使用端点 1 IN 发送到 PC。

5 大容量存储例程

该例程应用于意法半导体（ST）STM3210B-EVAL, STM3210C-EVAL, STM3210E-EVAL, STM32L1 52-EVAL and STM32L152D-EVAL 评估板，也可以很容易地适应任何其他硬件。

选用意法半导体（ST）的评估板来运行例程，应在 platform_config.h 文件中取消相应的行

5.1 描述

大容量存储例程为如何使用 STM32 USB-FS_Device 或 OTG FS_Device 外设与 PC 主机使用批量传输通讯提供了一个典型的例子。

本范例支持 BOT（仅限批量转移）协议和所有必要的 SCSI（小型计算机系统接口）命令，并且是兼容的 Windows XP（SP1，SP2，SPI3），Windows2000（SP4），Windows VISTA 和 Windows 7 的。

5.2 大容量存储例程概述

大容量存储范例符合 USB 2.0 和 USB 大容量存储类（仅块传输的子类）规范。运行该应用程序后，用户只需将 USB 电缆连到 PC 主机，不需要任何额外的驱动（与 Win2000，XP，VISTA 和 Windows 7），设备会自动被检测到。新的可移动磁盘显示在系统窗口，可以与任何其他可移动磁盘一样进行读/写/格式化操作（参见图 7）。

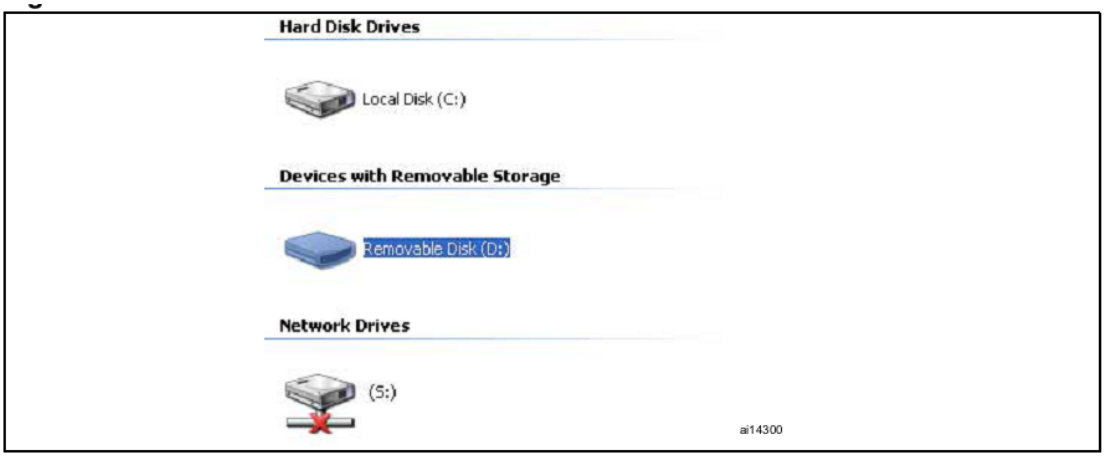

图 7：windows 中新的可移动磁盘

本文中，STM3210BEVAL，STM3210C-EVAL，STM32L152-EVAL 和 STM32L152D 的-EVAL 评估板支持使用 microSD 卡，STM3210E-EVAL 评估板支持一个 microSD 和 NANDflash。在 stm32_eval_sdio_sd.cc/.h，stm32_eval_spi_sd.c/.h 和 fsmc_nand.c/.H +文件中所有相关固件用于初始化，读取和写入。

注：

对于大容量存储类设备的固件和主机使用的文件系统并不需要知道和考虑。固件仅存储数据块并在当主机请求数据时发送数据块给主机。

5.3 大容量存储设备协议

5.3.1 仅块传输（BOT）

BOT 协议只使用批量传输通道传输命令,状态和数据(没有中断和控制通道)。默认的通道（通道 0，换句话说，端点 0）只用于清除批量通道的状态（清 STALL 状态），并发出两个类具体要求：大容量存储复位，并获得最大 LUN。

命令传输

要发送一个命令，主机采用被称为命令块包（CBW）的特定的格式，。 CBW 是一个 31 字节的数据包。表 10 显示了 CBW 的不同字段。

表 10：CBW 的不同字段

	7	6	5	4	3	2	1	0
0-3	dCBWSignature							
4-7	dCBWTag							
8-11	dCBWDataTransferLength							
12	bmCBWFlags							
13	Reserved (0)				bCBWLUN			
14	Reserved (0)				bCBWCBLength			
15-30	CBWCB							

- dCBWSignature : 43425355 : USBC (in little Endian)
- dCBWTag : 主机为每个命令指定这个字段。该设备应该返回相同的相关联的状态。
- dCBWDataTransferLength: 要传输的字节的总数（由主机统计）。
- bmCBWFlags: 此字段用于指定的数据传送的方向（如果有的话）。此输入的比特被定义为如下：

  Bit 7: 方向位

  0：数据 OUT 传输（主机到设备）。

  1：数据 IN 传输（设备到主机）。

  Bits 6:0: 保留（为零）。
- bCBWLUN: 相关逻辑单元号。
- bCBWCBLength : 此字段中设定 CBWCB 的长度（以字节为单位）。

● CBWCB ： 由该设备确定将要执行的命令块。

状态传输

每一个接收到的命令都会通知主机设备使用的命令状态包（CSW）的状态。表 11 示出了一个 CSW 的不同字段。

表 11：CSW 的不同字段

	7	6	5	4	3	2	1	0
0-3	dCSWSignature							
4-7	dCSWTag							
8-11	dCSWDataResidue							
12	bCSWStatus							

● dCSWSignature: 53425355 USBS (little Endian).

● dCSWTag ： 设备将此字段设置为 CBW 接收到的值 dCBWTag。

● dCSWDataResidue:的预期数据和由设备接收或发送的数据的实际价值之间的差异。（有关 CBW dCBWDataTransferLength 字段的值）

● bCSWStatus: 有关的命令的状态。该位可以取表 12 所示的三个非保留值之一。

表 12：命令块的状态值

值	描述
0x00	命令成功
0x01	命令失败
0x02	时钟错误
0x03=》0xff	保留

数据传输

数据传输阶段是指定相应的 CBW 的 dCBWDataCSW 与 bmCBWFlags 联系起来。主机尝试从设备获取传输的准确的字节数。图 8 中示出状态机的 BOT 传输。

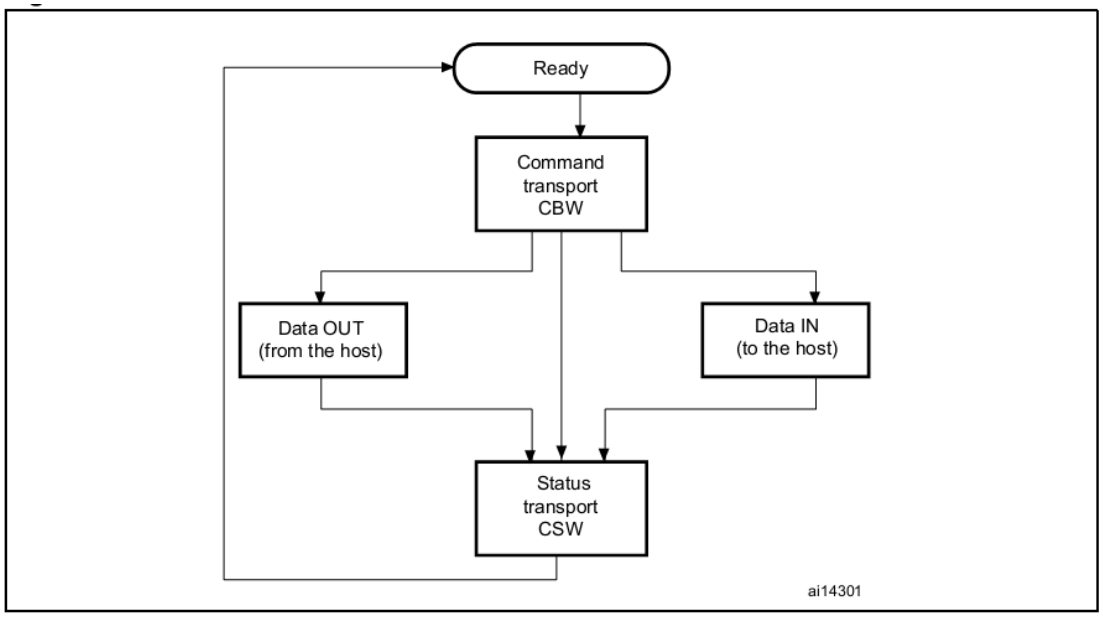

图 8：BOT 状态机

5.3.2 小型计算机系统接口（SCSI）

SCSI 命令集，旨在提供高效的对等操作，像 SCSI 设备一样，例如，硬盘，磁带和大容量存储设备。换句话说，这些是用来确保一个 SCSI 设备和 PC 主机操作系统之间的通信。

表 13 显示的可移动设备的 SCSI 命令。并非所有的命令都显示出来。欲了解更多信息，请参考 SPC 和 RBC 规范的。

表 13：SCSI 命令集

指令名	操作码	命令支持（1）	描述	参考
Inquiry	0x12	M	获取设备信息	SPC-2
Read Format Capacities	0x23	M	报告当前媒体支持的介质的的能力和了 Formattable 能力	SPC-2
Mode Sense (6)	0x1A	M	报告参数给主机	SPC-2
Mode Sense (10)		M	报告参数给主机	SPC-2
Prevent\ Allow Medium Removal	0x1E	M	禁止或允许从可移动媒体设备移除介质	SPC-2
Read (10)	0x28	M	从介质传输二进制数据到主机	RBC
Read Capacity(10)	0x25	M	报告当前媒介的能力	RBC
RequestSense	0x03	O	传输状态检测数据到主机	SPC-2
Start Stop Unit	0x1B	M	启用或禁用介质访问操作的逻辑单元，并控制一定的权力条件	RBC
Test Unit Ready	0x00	M	请求已准备好的设备报告。	SPC-2
Verify (10)	0x2F	M	验证介质上的数据	RBC
Write (10)	0x2A	M	从主机向介质传输二进制数据	RBC

1．命令支持键：M =支持是强制性的，O =支持是可选的。

5.4 大容量存储例程的实现

5.4.1 硬件配置接口

硬件配置接口是 USB 应用程序（在本例演示大容量存储）和 STM32 微控制器内部/外部硬件的之间的层。内部和外部硬件受 STM32 的标准外设库管理，所以从固件的角度来看，硬件配置接口是 USB 应用和标准外设库的固件层之间的接口。图 9 示出了不同的固件组件和硬件环境之间的相互作用。

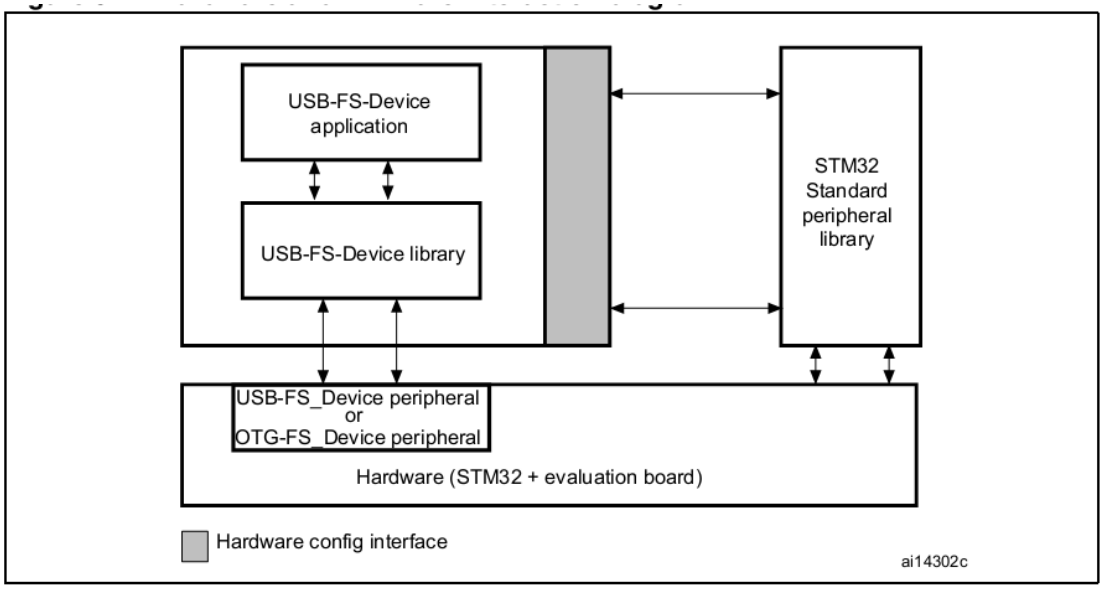

图 9：硬件和固件交互关系图

硬件配置层由 HW_config.c 和 hw_config.h 来实现。大容量存储例程中，硬件管理层管理以下硬件：

●系统和 USB-FS_Device 或 OTG FS_Device 外设的时钟配置

●读写 LED 配置

●LED 命令

●初始化的存储介质

●获取的存储介质的特性（块大小和存储器容量）

5.4.2 端点配置和数据管理

本部分提供了根据传输模式的数据流和结构的说明。

端点配置

每个 USB 复位事件后都应重新进行端点配置，这部分代码在 MASS_Reset 功能函数中实现（在文件 usp_prop.c 中）。

使用于除了互联型产品以外的所有 STM32：

要配置端点 0，这些是必要的：

●配置端点 0 默认为的控制端点

●配置端点 0 Rx 和 Tx 计数和缓冲地址在 BTABLE 中（usb_conf.h 文件，）

●配置端点接收状态为 VALID，发送状态为 NAK。

批量通道（端点 1 和 2）的结构如下所示：

●设定端点 1 为批量传入。

●配置端点 1 的 Tx 计数和数据缓冲地址在 BTABLE（usb_conf.h 文件）

●禁用端点 1 的 Rx

●配置端点 1 的 Tx 状态为 NAK。

●配置端点 2 为批量输出。

●配置端点 2 的 Rx 计数和数据缓冲地址在 BTABLE（usb_conf.h 文件）

●禁用端点 2 的 Tx

●设定端点 2 的 Rx 状态为 VALID。

对于 STM3210C -EVAL（互联型产品）：

端点 0 的配置工作在功能函数 USB_SIL_Init（）中。

批量通道（端点 1 和 2）的结构如下所示：

●使用的功能函数 OTGD_FS_EP_Init（）初始化端点方向，类型和最大数据包大小,，其中有这些参数：

- 端点地址（方向）：EPn_IN 为端点 Tx 和 EPn_OUT 的的端点 2 的 Rx（其中 n 是端点的数量）。

- 端点类型：在这种情况下，两个端点应使用批量类型。

- 端点的最大数据包的大小：要传输的最大数据量，从或向设备端点。

数据管理

数据管理是根据相关的端点直接从数据缓冲区地址转移所需要的数据到指定的 USB 存储器中。对于这些传输可用下面两个函数（在 usb_sil.c 文件中）：

●USB_SIL_Read（）：本函数作用是将接收的字节从 USB 的内存传输到内部 RAM。此功能用于复制由主机发送给设备的数据。接收到的数据字节的数目被确定到该函数（未作为参数传递的），并在操作结束时，返回此值。

●USB_SIL_Write（）：本函数作用是将发送的字节从内部 RAM 传输到 USB 的内存。此功能用于复制由设备发送给主机的数据。

5.4.3 类的具体请求

大容量存储类规范描述了两个类的具体要求：

仅块大容量存储复位

这个请求用于复位大容量存储设备和相关的接口。

Get Max LUN 请求

大容量存储设备可以实现多个逻辑单元共享共同的设备特征。

5.4.4 标准请求

要符合 BOT 规范，设备必须收到相同的标准请求后回应以下两个要求：

●当设备从未配置状态变为配置状态时，所有端点的数据触发都必须被清除。这由 Mass_Storage_SetConfiguration（）函数完成，该函数在 usb_prop.c 文件中。

●当主机发送 CBW 命令并带有一个无效的签名或者长度时，设备必须保证端点 1 和 2 都在 STALL，直到它收到大容量存储复位类的具体要求。此功能由 Mass_Storage_ClearFeature（）函数完成，该函数在 usb_prop.c 文件中。

5.4.5 BOT 状态机

为了提供 BOT 协议，一个特定的带有 5 个状态的状态机被提出。状态描述如下：

● BOT_IDLE：这是一个 USB 复位的默认状态，仅批量传输的大容量存储复位或在 CSW 发出后触发。在这种状态下，设备已准备就绪从主机接收一个新的 CBW。

● BOT_DATA_OUT：当设备接收到一个从主机到设备的 CBW 数据流后进入这个状态。

● BOT_DATA_IN ：当设备接收一个从设备到主机的 CBW 数据流后进入这个状态。

● BOT_DATA_IN_LAST ：当发送最后的数据由主机要求提供时进入这个状态。

● BOT_CSW_SEND: 当设备发送 CSW 时进入此状态。在该状态下，一个正确的 IN 传输发生，设备变换成 BOT_IDLE 状态从而使能接收下一个 CBW。

● BOT_ERROR: 错误状态。

5.4.6 SCSI 协议的实现

SCSI 协议的目的是对 PC 主机上操作系统所需要的所有 SCSI 命令提供正确的反应。本节将详细介绍所有 SCSI 命令的管理方法。

●INQUIRY 命令（操作码= 0x12）：根据分配长度字段的命令，发送所需的查询页数据所需要的数据长度。

●SCSI READ FORMAT CAPACITIES 命令（操作码=0X23）：在检查介质后发送 Read Format Capacity 命令数据。如果没有介质被检测到，MEDIUM_NOT_PRESENT 错误将返回到主机强制其更新内部参数。

● SCSI READ CAPACITY (10)命令（操作码=0X25）：在检查介质后发送 READ CAPACITY (10)命令数据。如果没有介质被检测到，MEDIUM_NOT_PRESENT 错误将返回到主机强制其更新内部参数。

●SCSI MODE SENSE（6）命令（操作码=0x1A）

●SCSI MODE SENSE（10）命令（操作码=0x5A）

●SCSI REQUEST SENSE 命令（操作码=0x03）

●SCSI TEST UNIT READY 命令（操作码=0x00）

●SCSI PREVENT\ALLOW MEDIUM REMOVAL 命令（操作码=0x1E）

●SCSI START STOP UNIT 命令（操作码=0x1B）

●SCSI READ 10 命令（操作码=0x28）和 SCSI WRITE 10 命令（操作码=0x2A）

●SCSI VERIFY10 命令（操作码=0x2F）

5.4.7 内存管理

所有的内存管理功能都集中在 memory.c 和 memory.h 两个文件中。内存管理包括两个基本过程：

●SCSI READ（10）和 SCSI WRITE（10）命令用来管理和验证地址的范围：这个过程是由 Address_Management_Test（）函数完成的。这个函数的作用是提取真正的地址和存储介质的地址并检验是否当前的传输(读或写)在内存范围内。如果不在内存范围内，函数 STALLS 端点 1 或 2 或两个端点（根据传输是读还是写）返回一个错误状态来中止传输。

●Read_Memory（）和 Write_Memory（）两个命令用来管理读取和写入的过程。这两个命令管理介质访问通过两个函数 MAL_WriteBlock 和 MAL_ReadBlock，在 mass_mal.c 文件中。每次访问后，使用上一传输的长度更新当前存储器偏移量和下次访问的地址。

5.5 如何自定义大容量存储范例

该范例是一个简单的例子，用来证明 STM32 USB 外设在批量传输性能。但是它可以根据用户要求定制。定制可以实现大容量存储协议的三个层次：

●BOT 层定制：用户可以定制自己的 BOT 状态机或者通过修改这两个文件 usb_bot.c 和 usb_bot.h 修改已实现的 BOT 状态机，只是需要保持相同的数据传输方法。

●SCSI 层定制：SCSI 协议的实现，除了第 5.4.6 节中列出的所支持的命令外，其他命令都不支持。当主机发送这些命令时，相应的函数将被函数 BW_Decode（）调用。然而，所有相关的功能不支持的命令被定义为在 SCSI_Invalid_Cmd（）函数中，（见 usb_scsi.c 文件）。在 SCSI_Invalid_Cmd（）函数 STALLS 的两个端点（1 和 2），设置检测数据无效的命令键，并发送一个命令失败状态的 CSW。为了支持无效的命令，用户可以注释掉相关的行来实现自己的功能。例如，对于需要支持 SCSI_FormatUnit 的命令，注释行：

```
// #define SCSI_FormatUnit_Cmd SCSI_Invalid_Cmd
```

实现过程的函数应具有相同的名称在 usb_scsi.c 文件中：

```
void SCSI_Invalid_Cmd (void)
{
// your implementation
}
```

通过这种方式，自定义函数被调用时自动调用 CBW_Decode（）函数（在 usb_BOT.c 文件中）。

但是，如果你需要执行一个命令没有列在前面的列表，您必须修改 CBW_Decode（）函数和实现新的命令的协议。

大容量存储描述

表 14：设备描述符

文件	值	描述
bLength	0x12	描述符长度（字节）
bDescriptortype	0x01	描述符类型（设备描述符）
bcdUSB	0x0200	规范版本号：2.00
bDeviceClass	0x00	设备类
bDeviceSubClass	0x00	设备子类
bDeviceProtocol	0x00	设备协议
bMaxPacketSize0	0x40	端点 0 的最大数据包大小：64 字节
idVendor	0x0483	产商 ID（ST）
dProduct	0x5720	产品 ID
bcdDevice	0x0100	移动设备版本号：1.00
iManufacturer	4	制造商的字符串描述符索引：4
iProduct	42	产品字符串描述符索引：42
iSerialNumber	96	序列号的字符串描述符的索引
bNumConfigurations	0x01	可能配置的数量：1

表 15：配置描述符

文件	值	描述
bLength	0x09	描述符长度（字节）
bDescriptortype	0x02	描述符类型（配置描述符）
wTotalLength	32	返回的数据总长度（包括接口端点描述符描述），（以字节为单位）
bNumInterfaces	0x0001	这个配置所支持的接口数量（1）
bConfigurationValue	0x01	配置值
iConfiguration	0x00	配置描述符的索引
bmAttributes	0x80	配置特点：总线供电
Maxpower	0x32	通过 USB 总线的最大功耗：100 毫安

表 16：接口描述符

文件	值	描述
bLength	0x09	描述符长度（字节）
bDescriptortype	0x04	描述符类型（接口描述符）
bInterfaceNumber	0x00	接口数量
bAlternateSetting	0x00	备用设置编号
bNumEndpoints	0x02	使用端点数：2
bInterfaceClass	0x08	接口类型：大容量存储设备
bInterfaceSubClass	0x06	接口子类：SCSI 传输
bInterfaceProtocl	0x50	接口协议：0x50
iInterface	106	接口字符串描述符索引

表 17：端点描述符

文件	值	描述
IN 端点		
bLength	0x07	描述符长度（字节）
bDescriptortype	0x05	描述符类型（端点描述符）
bEndpointAddress	0x81	IN 端点地址：1
bmAttributes	0x02	批量端点
wMaxPacketSize	0x40	64 字节
bInterval	0x00	不适用于批量端点
OUT 端点		
bLength	0x07	描述符长度（字节）
bDescriptortype	0x05	描述符类型（端点描述符）
bEndpointAddress	0x02	OUT 端点地址：2
bmAttributes	0x02	批量端点
wMaxPacketSize	0x40	64 字节
bInterval	0x00	不适用于批量端点

6 虚拟 COM 端口例程

该例程应用于意法半导体（ST）STM3210B-EVAL, STM3210C-EVAL, STM3210E-EVAL, STM32L1 52-EVAL and STM32L152D-EVAL 评估板，也可以很容易地适应任何其他硬件。

选用意法半导体（ST）的评估板来运行例程，应在 platform_config.h 文件中取消相应的行

6.1 描述

在现代 PC 中，USB 的标准通信端口几乎支持所有的外设。然而，许多工业软件应用程序仍然使用传统的 COM 口（UART）。虚拟 COM 端口范例提供了一个简单的解决办法绕过这个问题。它使用 USB 设备作为一个 COM 端口与 PC 的应用程序通信。

虚拟 COM 端口范例提供了 STM32 系列产品和 PC 驱动器固件的例子。本节提供了一个简短的描述来介绍如何运行演示。

6.2 虚拟 COM 端口范例方案

演示的方案是，使用 STM32 EVAL 板作为 USB-USART 桥，并提供一台笔记本电脑（不包括 RS-232 端口）和一个标准的 PC 工作站之间的通信，如图 10 所示。在通信中使用的 PC 应用是 Windows 超级终端。请参阅图 11。

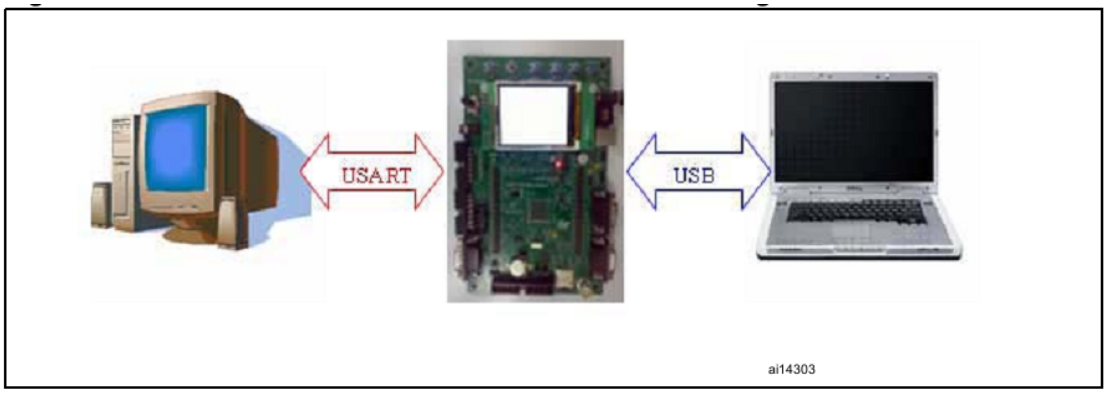

图 10：虚拟 COM 端口演示 USB-USART 桥

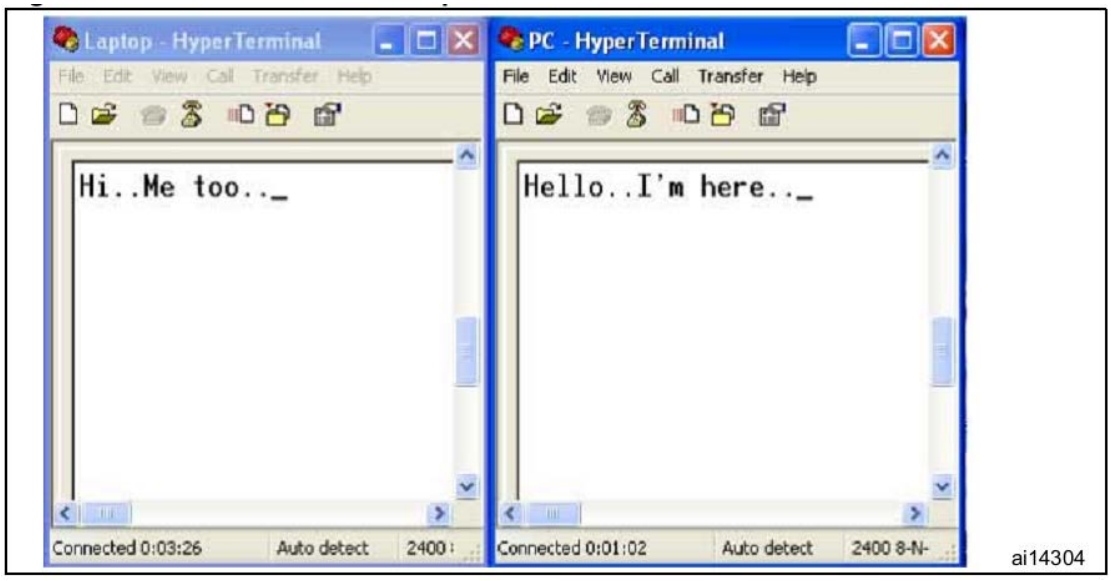

图 11：通信例子

6.3 软件驱动程序的安装

要安装该软件的虚拟 COM 端口驱动程序，的意法半导体网站：www.st.com 下载并执行“虚拟 COM 端口驱动程序安装”。在安装结束时，一个新的 COM 端口出现在“设备管理器”窗口，如图 12 所示。

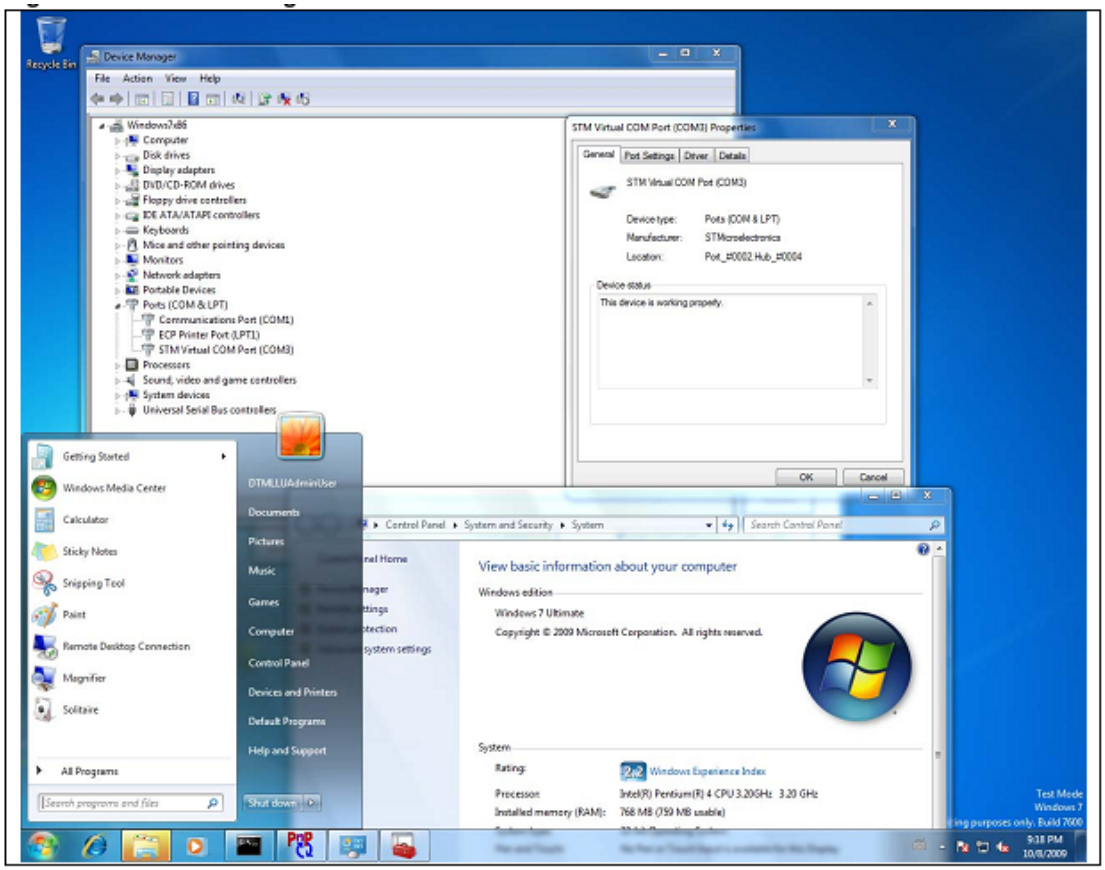

图 12：“设备管理器”窗口

6.4 实现

6.4.1 硬件实现

虚拟 COM 端口范例使用的 USART1 在 STM3210B-EVAL 和 STM3210E-EVAL 评估板，或 USART2 在 STM3210C-EVAL，STM32L152-EVAL STM32L152D-EVAL 评估板。不需要添加外部的硬件辅助演示。

6.4.2 软件实现

为了被视为一个 COM 端口，USB 设备必须实现两个接口的通信设备类（CDC）规范：

●抽象控制模型通信，中断 IN 端点：在我们的实现这个接口描述符中声明但不使用相关的端点（端点 2）

●抽象控制模型数据，一个批量输入和一个批量输出端点：该接口在演示中表示端点 1 的（IN）和端点 2 的（OUT）。端点 1 用于从 UART 到 PC 发送数据，通过 PC 的 USB 接收。端点 3 用于从 PC 到 USART 发送数据，通过 USART 接收。

CDC 类的更多信息，请参阅 www.usb.orgwebsite 提供的通信设备规格的通用串行总线类定义。

硬件配置接口

硬件配置接口（hw_config.c， H）在虚拟的 COM 端口管理下面的例程：

●配置系统和外围设备（USB&USART）的时钟和中断

●初始化 USART 为默认值。

●配置 USART 接收到的参数的 SET_LINE_CODING 请求

●发送接收到的数据通过 USART 的 PC 通过 USB

●发送从 USB 接收到的数据通过 USART

注：

对于 STM32，支持的数据格式是 7 位或 8 位（在超级终端）和带宽范围是从 1200 到 115200。

7 USB 音频扬声器例程

该例程应用于意法半导体（ST）STM3210B-EVAL, STM3210C-EVAL, STM3210E-EVAL, STM32L1 52-EVAL and STM32L152D-EVAL 评估板，也可以很容易地适应任何其他硬件。

选用意法半导体（ST）的评估板来运行例程，应在 platform_config.h 文件中取消相应的行。

7.1 描述

USB 音频扬声器演示举例说明了如何使用 STM32 USB 外设在同步传输模式与 PC 主机进行通信。他们提供了一个配置的同步端点，接收或示范的正确方法向主机发送数据。他们还展示了如何使用数据的实时应用程序。

本用户指南中描述的可用语音演示，USB 扬声器。

7.2 同步传输综述

当应用程序需要同步传输，以保证访问 USB 界延迟，稳定的数据传输速率和带宽，不尝试新的数据数据传输操作失败的情况下。

事实上，同步传输方式并没有握手阶段，没有 ACK 数据包预期或之后发送的数据包。图 13 示出一个例子的等时 OUT 具有 64 个字节的数据包中的转移。

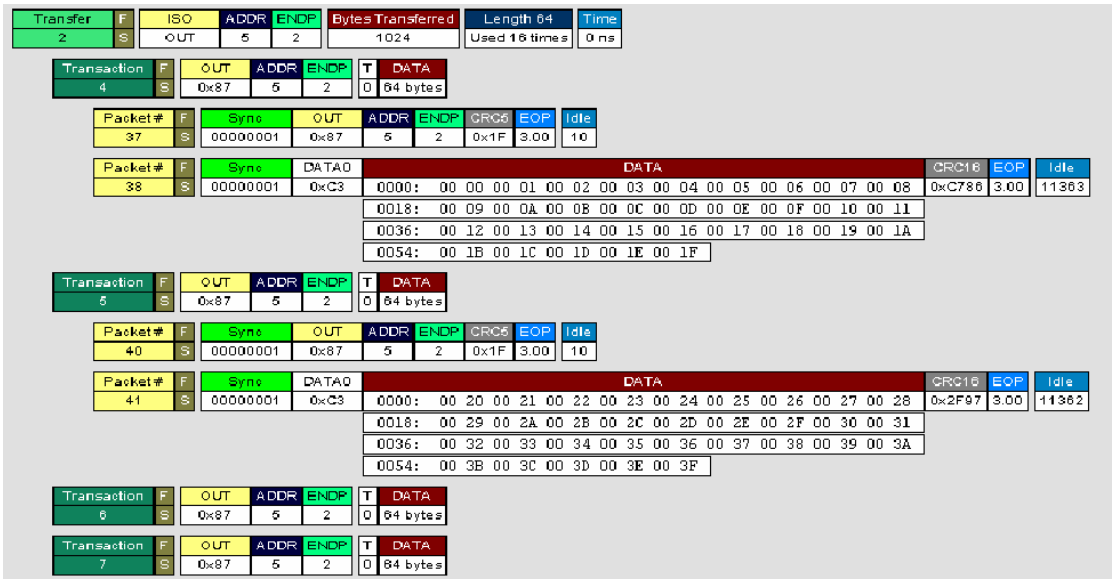

图 13：同步输出转移

同步传送模式下的应用程序使用的典型的例子是音频采样，压缩的视频数据流，并在一般，严格的采样数据的任何形式的交付的频率精度要求。

USB 同步传输的详细信息，请参阅 USB 2.0 规范模式的特点。

7.3 音频设备类综述

音频设备，音频设备的通用串行总线类定义中定义说明书中，是一个设备或一个函数，用于嵌入在复合设备操作音频，语音和声音相关的功能。这既包括音频数据（模拟和数字）的功能，是用来直接控制音频环境，如音量和音调控制。

所有的音频设备进行分组，从 USB 的观点看，在音频接口类。这类，分为几个子类。通用串行总线类定义的音频设备规范详细说明了以下三个子类：

音频控制接口子类（AC）：每个音频功能有一个唯一的音频接口。 AC 接口用于控制一个特定的功能行为音频功能。为了实现这一功能，这个接口可以使用下面的端点：

---操作单元和终端设置一个控制端点（端点 0）和检索音频功能，用类专用请求的状态。

---中断端点的状态回报。此端点是可选的。

音频控制接口是通过单一的入口点来访问内部的音频功能。所有请求所关心的某些音频控制操作在音频功能的单位或终端必须的音频控制接口的音频功能。同样，所有的描述符相关的内部音频功能类专用音频控制接口描述符的一部分。

音频功能的音频控制接口，可以支持多个备用设置。替代的音频控制接口的设置可以例如被用于实现音频功能，支持多种拓扑的不同的类专用音频控制接口描述符的为每个替代设置。

音频流接口子类（AS）：音频流接口用于交换数字音频数据流之间的主机和音频功能。他们是可选的。音频功能可以有零个或多个音频流接口相关的，每一个可能具有不同的性质和格式的数据。每音频流接口可以最多有一个同步数据端点。

MIDI 流接口的小类（MIDIS）：MIDI 流接口用于运输 MIDI 数据流的流入和流出的音频功能。

为了能够操纵的音频功能，其功能的物理性质必须被分成寻址实体。两种类型的例如通用的实体是识别和被叫做单元和终端。通用串行总线类定义音频设备规范定义了 7 种标准的单元和终端被认为足以代表大多数的音频功能。

这些是：

- 输入端
- 输出终
- 混合机组
- 选择单位
- 功能单元
- 处理单元
- 扩展单元

音频类的特点和要求的更多信息，请参阅通用串行总线设备类定义的音频设备规范网站的 usb.org。

7.4 STM32 USB 扬声器范例

从 USB 扬声器范例的目的是为了接收音频数据流（数据）使用 USB 的 PC 主机，并发挥它通过 STM32 MCU。图 14 为 STM32 USBFS_设备扬声器范例数据流体现了 PC 主机之间的数据流和扬声器。

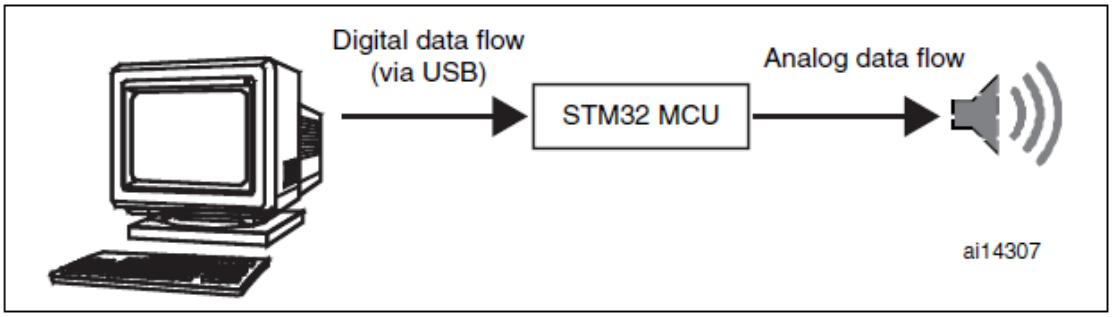

图 14：STM32 USB-FS_Device 的扬声器范例数据流

7.4.1 一般特性

●USB 设备的特点：

- 端点 0：用于枚举设备类专用的要求。这个端点的最大数据包大小为 64 字节。
- 端点 1（OUT）：用于接收音频数据流从 PC 主机最大数据包大小为 22 字节。

●音频特性：

- 音频数据格式：Type I / PCM8 格式/ Mono（单声道）。
- 音频数据分辨率：8 位。
- 采样频率：22 千赫。

●硬件要求：

在 STM3210B-EVAL 板的情况下，因为 STM32 MCU 不具有一个片上 DAC 来产生模拟

数据流的另一种方法是用来实现 1 通道 DAC。此方法包括在使用生成的脉冲宽度调制（PWM）模块以产生一个信号，其脉冲宽度成比例的振幅的样本数据。 PWM 的输出信号通过一个低通滤波器，然后集成移除高频分量，只留下低频含量的输出的低通滤波器提供了一种合理的原始模拟信号的再现。

图 15 显示了使用内置的 PWM 的音频回放数据流图。在 STM3210E-EVAL 的情况下，I2S 独立的音频外围设备被用来产生音频数据。

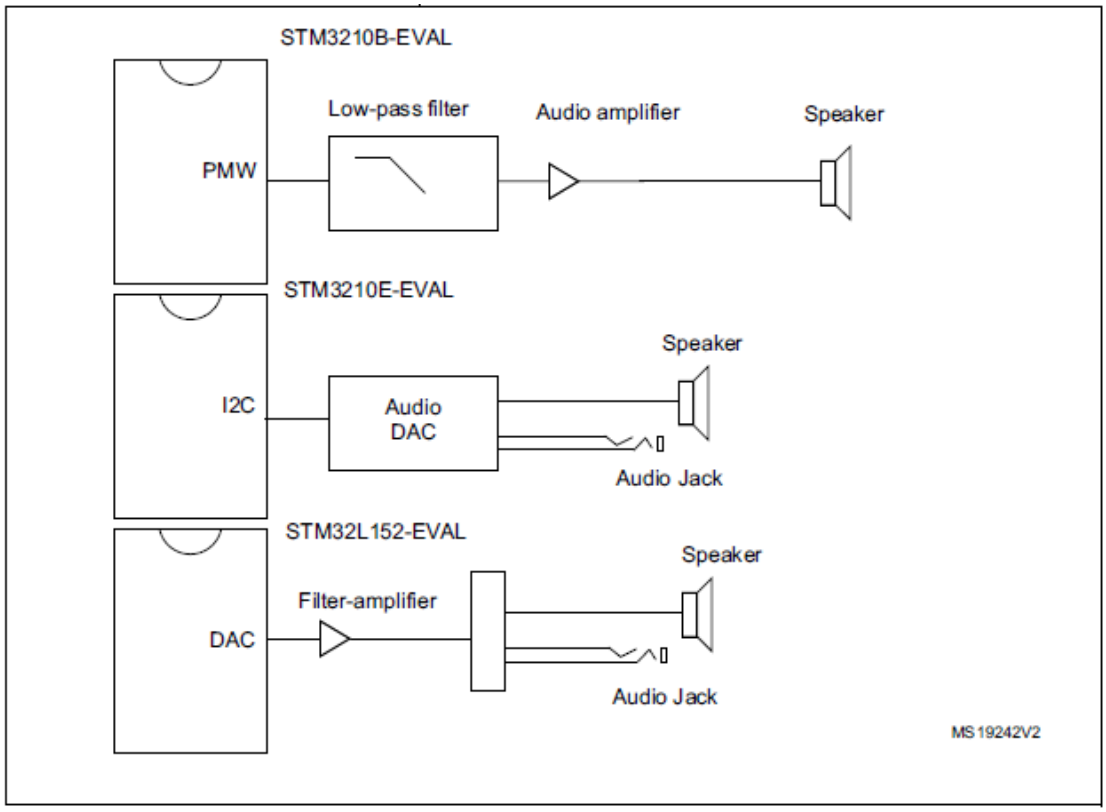

图 15：音频播放流

上图文字注释：

Low-pass filter——低通滤波器

Audio amplifier——音频放大器

Speaker——扬声器

AudioDAC——音频 DAC

Audio Jack——音频插孔

Filter-amplifier——Filter-amplifier

7.4.2 实施

本节介绍了硬件和软件解决方案，用于实现一个 USB 音频扬声器使用 STM32 微控制器。

硬件实现在 STM3210B-EVAL 板，实现 PWM 功能，以下的情况下，STM32 的内置的定时器被使用：

●TIM2 输出比较时序模式作为系统定时器。

●在 PWM 模式下 TIM4

在 STM3210E-EVAL 板的情况下，独立的 I2S 音频外围设备直接生成音频数据。

在 STM32L152-EVAL 板，外设直接嵌入 DAC 的情况下，产生的音频数据（帧同步的控制使用 TIM6 定时器）。

固件实现

STM32 的扬声器演示的目的是存储接收到的数据（音频流）在一个特定的缓存中并且使用 PWM 播放一段音频流（8 位主机格式）45.45 微秒（～22 千赫）。

硬件配置界面：

硬件配置接口是 USB 应用层之间的（在我们的 USB 设备音频扬声器）和内部/外部硬件的

STM32 微控制器。此内部和外部硬件管理 theSTM32 标准外设库，所以从固件的角度来看，在硬件配置接口的固件层之间的 USBFS_设备应用程序和标准外设库。图 16 显示了之间的相互作用的不同的固件组件和硬件环境。硬件配置层的 HW_config.c 和 hw_config.h。对于 USB 扬声器范例，硬件管理层管理以下硬件要求：

- 系统和 USB 外设的时钟配置
- 定时器配置（当使用 STM3210B-EVAL）
- I2S 配置（当使用 STM3210E-EVAL）
- DAC 和定时器配置（当使用 STM32L152-EVAL）

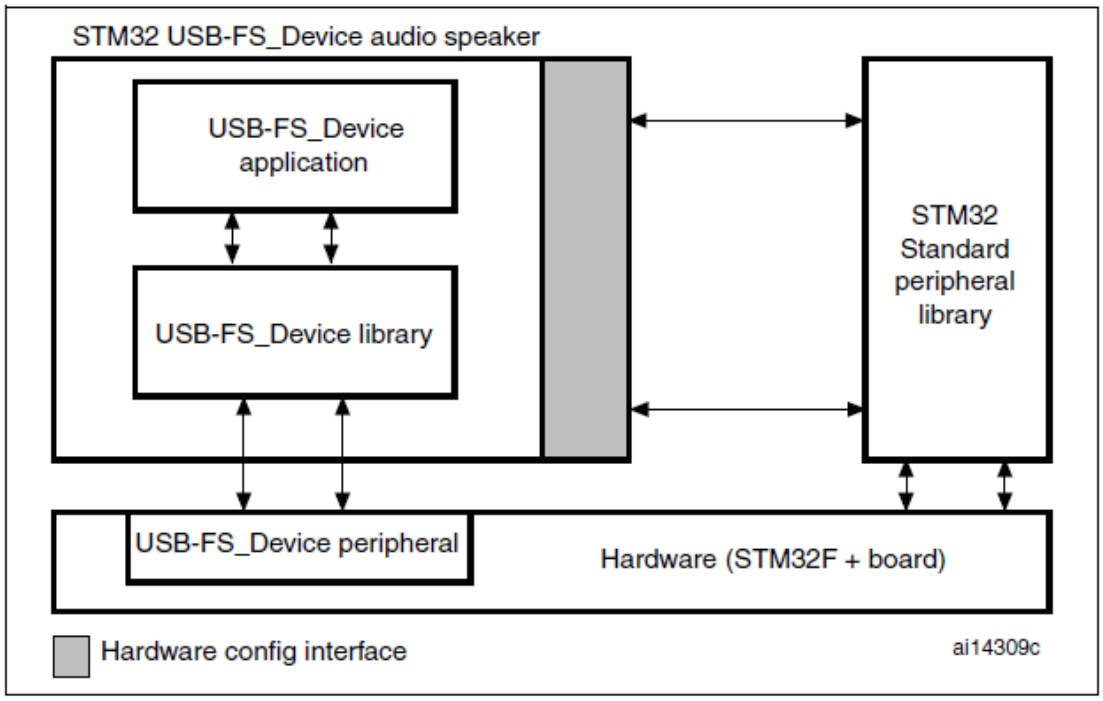

图 16：硬件和固件交互图

上图文字注释

USB-FS_Device application ——USB-FS 设备的应用

USB-FS_Device library——USB-FS 设备库

USB-FS_Device peripheral——USB-FS 设备周边

STM32 Standard peripheral library——STM32 的标准外设库

Hardware (STM32F + board)——硬件（STM32F+板）

Hardware config interface——硬件配置界面

端点配置：

在 STM32 USB 设备的扬声器演示，两个端点被用来沟通与 PC 主机：端点 0 和端点 1。需要注意的是端点 1 是一个 OUT 端点的端点，而且这种同步管理的 STM32USB 周边设备使用双缓冲模式，这样的固件提供两个数据包存储区的缓冲区,这个端点。下面的 C 代码描述了用于配置同步输出的方法端点(见 usb_prop.c 的文件，Speaker_Reset（）函数)。

```
/* Initialize Endpoint 1 */
SetEPType(ENDP1, EP_ISOCHRONOUS);
SetEPDblBuffAddr(ENDP1,ENDP1_BUF0Addr,ENDP1_BUF1Addr);
SetEPDblBuffCount(ENDP1, EP_DBUF_OUT, 22);
```

ClearDTOG_RX(ENDP1);

ClearDTOG_TX(ENDP1);

ToggleDTOG_TX(ENDP1);

SetEPRxStatus(ENDP1, EP_RX_VALID);

SetEPTxStatus(ENDP1, EP_TX_DIS);

c）类的具体要求

这个实现仅支持静音控制。此功能所管理的 Mute_command 函数（usb_prop.c 文件）。

d）同步数据传输管理

正如前面的描述，的 STM32 管理的同步数据传输使用双缓冲模式。所以，要复制所接收的数据从 PMAStream_Buffer 中，交换之间的两个 PMA 缓存（ENDP1_BUF0Addr

和 ENDP1_BUF1Addr）来进行管理。数据交换的 PMA 根据之间的 USB 外围设备和缓冲区使用管理固件。此操作提供的的 EP1_OUT_Callback（）函数（usb_endp.c 文件文件）。复制过程结束后，一个全局变量 IN_Data_Offset 更新所接收的字节数，并在复制 Stream_Buffer 中。

e）音频播放实现：

使用时要播放的音频采样从主机接收 STM3210B-EVAL 板，定时器 TIM4 将被编程来产生一个 125.5kHzPWM 信号和 TIM2 被编程以产生一个中断的频率等于 22 千赫。在每个 TIM2 中断上，一个音频流被用来更新的 PWM 脉冲。一个全局变量（Out_Data_Offset）被用于指向下一个流要播放的流缓冲。

当 I2S 音频外设的 STM3210E-EVAL 板，Out_Data_Offset 变量控制的流的流量要同步的数据从 USB 流缓冲使用 I2S 周边的。

当 DAC 外围设备采用的是 STM32L152-EVAL 板，Out_Data_Offset 变量控制的流的流量要同步的数据从 USB 流缓冲的 DAC 外围设备使用。

注意：　　请注意，两个“IN_Data_Offset”和“Out_Data_Offset”被初始化为 0，在每个开始帧中断（见 usb_istr.c 的文件，SOF_Callback（）函数），以避免溢出的“Stream_Buffer 中”。

音频扬声器描述符

表 18：设备描述符

文件	值	描述
bLength	0x12	该描述符的大小（以字节为单位）
bDescriptortype	0x01	描述符类型（设备描述符）
bcdUSB	0x0200	USB 规范版本号：2.0
bDeviceClass	0x00	设备类
bDeviceSubClass	0x00	设备子类
bDeviceProtocol	0x00	设备协议
bMaxPacketSize0	0x40	端点 0 的最大数据包大小：64 字节;
idVendor	0x0483	销售商标识符（意法半导体）
idProduct	0x5730	产品标识
bcdDevice	0x0100	移动设备版本号：1.00
iManufacturer	0x01	制造商的字符串描述符的索引：1
iProduct	0x02	指数产品描述符：2
iSerialNumber	0x03	指数的序列号字符串描述符：3
bNumConfigurations	0x01	可能的配置数：1

表 19：配置描述符

领域	值	描述
bLength	0x09	该描述符的大小（以字节为单位）
bDescriptortype	0x02	描述符类型（配置描述符）
wTotalLength	0x006D	返回的数据的总长度（以字节为单位）描述符（包括接口端点描述符）
bNumInterfaces	0x02	这个配置所支持的接口的数目（两个接口）
bConfigurationValue	0x01	配置值
iConfiguration	0x00	配置描述符的索引
bmAttributes	0x80	配置特点：总线供电
Maxpower	0x32	通过 USB 总线的最大功耗：100 毫安

表 20：接口描述符

领域	值	描述
USB 扬声器标准接口 AC 描述符（接口为 0，替代设置为 0）		
bLength	0x09	该描述符的大小（以字节为单位）
bDescriptortype	0x04	描述符类型：接口描述符
bInterfaceNumber	0x00	接口数量
bAlternateSetting	0x00	备用设置号
bNumEndpoints	0x00	使用端点号：0（仅用于端点 0 该接口）
bInterfaceClass	0x01	接口：USB 类设备类 AUDIO
bInterfaceSubClass	0x01	接口子类：音频的子类音频控制
bInterfaceProtocol	0x00	接口协议：音频协议 UNDEFINED
iInterface	0x00	接口字符串描述符索引
USB 扬声器类专用 AC 接口描述符		
bLength	0x09	该描述符的大小（以字节为单位）
bDescriptortype	0x24	描述符类型：AUDIO 接口描述符 TYPE
bDescriptorSubtype	0x01	描述符子类型：音频控制头
bcdADC	0x0100	bcdADC：1.00
wTotalLength	0x0027	总长度：39
bInCollection	0x01	流接口数：1
baInterfaceNr	0x01	baInterfaceNr：1
USB 音箱输入终端描述符		
bLength	0x0C	这个描述符的大小（以字节为单位）：12
bDescriptortype	0x24	描述符类型：AUDIO 接口描述符 TYPE
bDescriptorSubtype	0x02	描述符子类型：音频控制输入 TERMINAL
bTerminalID	0x01	终端 ID：1
wTerminalType	0x0101	音频端子 USB 流媒体终端类型：
bAssocTerminal	0x00	无关联
bNrChannels	0x01	一个通道
wChannelConfig	0x0000	信道配置：单声道

iChannelNames	0x00	未使用
iTerminal	0x00	未使用
USB 扬声器音频功能单元描述符		
bLength	0x09	该描述符的大小（以字节为单位）
bDescriptortype	0x24	描述符类型：AUDIO 接口描述符 TYPE
bDescriptorSubtype	0x06	描述符子类型：音频控制功能单元
bUnitID	0x02	单位 ID：2
bSourceID	0x01	来源 ID：1
bControlSize	0x01	控制尺寸：1
bmaControls	0x0001	支持静音控制
iTerminal	0x00	未使用
USB 音箱输出终端描述符		
bLength	0x09	该描述符的大小（以字节为单位）
bDescriptortype	0x24	描述符类型：AUDIO 接口描述符 TYPE
bDescriptorSubtype	0x03	描述符子类型：声音控制输出 TERMINAL
bTerminalID	0x03	终端 ID：3
wTerminalType	0x0301	终端类型：扬声器音频端子
bAssocTerminal	0x00	无关联
bSourceID	0x02	来源 ID：2
Terminal	0x00	未使用
USB 扬声器标准 AS 接口描述符 - 音频流零带宽（接口 1 个，备用设置 0）		
bLength	0x09	该描述符的大小（以字节为单位）
bDescriptortype	0x24	描述符类型：AUDIO 接口描述符 TYPE
bInterfaceNumber	0x01	接口数：1
bAlternateSetting	0x00	备用设定值：0
bNumEndpoints	0x00	不使用（零带宽）
bInterfaceClass	0x01	接口：USB 类设备类 AUDIO
bInterfaceSubClass	0x02	接口子类：音频 SUBCLASS 音频流
bInterfaceProtocol	0x00	接口协议：音频协议 UNDEFINED

iInterface	0x00	未使用
USB 扬声器标准 AS 接口描述符 - 音频流操作（接口 1 个，备用设置 1）		
bLength	0x09	该描述符的大小（以字节为单位）
bDescriptortype	0x24	描述符类型：AUDIO 接口描述符 TYPE
bInterfaceNumber	0x01	接口数量：1
bAlternateSetting	0x01	替代设置：1
bNumEndpoints	0x01	一个端点
bInterfaceClass	0x01	接口：USB 类级的音频
bInterfaceSubClass	0x02	接口子类：音频 SUBCLASS 音频流
bInterfaceProtocol	0x00	接口协议：音频协议 UNDEFINED
iInterface	0x00	未使用
USB 的扬声器音频流接口描述符		
bLength	0x07	该描述符的大小（以字节为单位）
bDescriptortype	0x24	描述符类型：AUDIO 接口描述符 TYPE
bInterfaceNumber	0x01	接口数量：1
bAlternateSetting	0x01	替代设置：1
bNumEndpoints	0x01	一个端点
wFormatTag	0x0002	PCM 8 格式
USB 扬声器音频 I 型格式的接口描述符		
bLength	0x0B	该描述符的大小（以字节为单位）
bDescriptortype	0x24	描述符类型：AUDIO 接口描述符 TYPE
bDescriptorSubtype	0x03	描述符亚型：音频流的格式类型
bFormatType	0x01	格式类型：I 型
bNrChannels	0x01	通道数：1 通道
bSubFrameSize	0x01	副车架大小：1 个字节音频子
bBitResolution	0x08	每个样品位分辨率：8 位
bSamFreqType	0x01	一个频率支持
tSamFreq	0x0055F0	22 kHz

表 21：端点描述符

领域	值	描述
端点 1 - 标准描述符		
bLength	0x07	该描述符的大小（以字节为单位）
bDescriptortype	0x05	描述符类型（端点描述符）
bEndpointAddress	0x01	OUT 端点地址 1
bmAttributes	0x01	同步端点
wMaxPacketSize	0x0016	22 字节
bInterval	0x00	未使用
端点 1 - 音频流描述符		
bLength	0x07	该描述符的大小（以字节为单位）
bDescriptortype	0x05	描述符类型：AUDIO 接口描述符 TYPE
bDescriptor	0x01	音频终端一般
bmAttributes	0x80	bmAttributes: 0x80
bLockDelayUnits	0x00	未使用
wLockDelay	0x0000	未使用

8 USB 音频流例程

该例程应用于意法半导体（ST）STM3210B-EVAL, STM3210C-EVAL, STM3210E-EVAL, STM32L1 52-EVAL and STM32L152D-EVAL 评估板，也可以很容易地适应任何其他硬件。

选用意法半导体（ST）的评估板来运行例程，应在 platform_config.h 文件中取消相应的行。

8.1 概述

USB 音频流演示提供了一个例子，如何使用 STM32 OTGFS_，处理外围设备与外围设备的音频级 I2S 通信与合并在同步传送模式和输出高品质的音频流中的 PC 主机。它是基于演示的 USB 语音增强功能和使用的音频级 I2S 周边性能。

本用户指南中描述的可用语音演示，USB 扬声器。

8.2 STM32 USB 音频流例程

USB 音频流演示的目的，是为了从接收音频数据流（数据）PC 使用 OTG FS_Device 外设和主机播放该通过 STM32 音频类 I2S 外设。数据流和使用的硬件是基本相似的声音演示第 7 节所述。

8.2.1 一般特性

●OTG FS_Device特点：

- 端点 0：用于枚举设备类专用的要求。这个端点的最大数据包大小为 64 字节。
- 端点 1（OUT）：用于接收音频数据流从 PC 主机可配置的最大数据包大小（根据所需的音频速率，固定在 usb_conf.h 文件通过定义 AUDIO_FREQ_xxK。取消对相应的定义使用选定的声频。

●音频特性：

- 音频数据格式：Type III / PCM 格式（16 位）/立体声。
- 音频数据分辨率：16 位。
- 采样频率：固定频率：8，11.025，16，22.05，32，44.1 或 48 kHz。

●硬件要求：

对于 STM3210C-EVAL，I2S 独立的音频外围设备被用来产生音频数据，如图 17 所示。

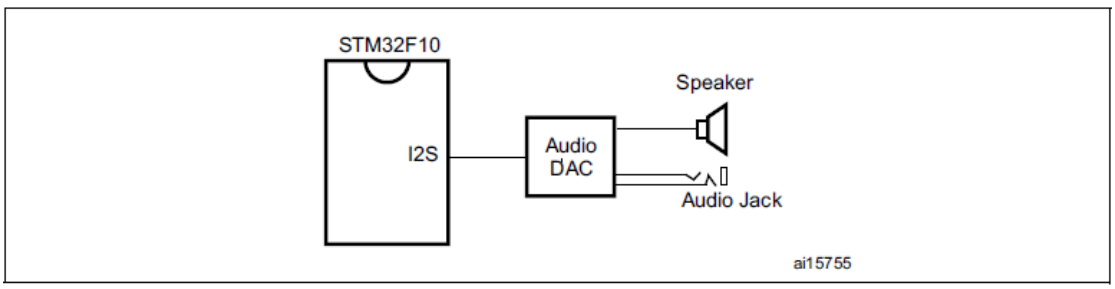

图 17：音频播放流

8.2.2 实施

本节介绍使用 STM32 微控制器实现一个 USB 音频流的硬件和软件解决方案。

硬件实现

对于 STM3210C-EVAL 板，I2S 独立的音频外设直接生成的音频数据使用 DMA 传输。

软件实现

STM32 的音频流演示的目的是存储在一个从主机接收的多缓冲 Isoc_Buffer 数据（音频流）和使用的 I2S 结合 DMA 播放数据流（16 位）。

A）硬件配置接口：

硬件配置接口是 USB 应用程序（在这里是 USB 音频流）和 STM32F105/107xx 互联型微控制器内部/外部硬件之间的层。STM32F105/107xx 互联型微控制器内部/外部硬件受标准外设库管理，所以从固件的角度来看，硬件配置接口是 USB 应用程序和标准外设库的固件层之间的层。图 18 示出了不同固件组件和硬件环境之间的相互作用。

硬件配置层由两个文件：hw_config.c 和 hw_config.h 来实现。USB 音频流演示，硬件管理层管理以下硬件要求：

- 系统和 OTG FS_Device 外设时钟配置
- 编解码器的配置（通过 I2C 控制接口和 IO 扩展器复位引脚）
- 音频属性和 I2S 配置。

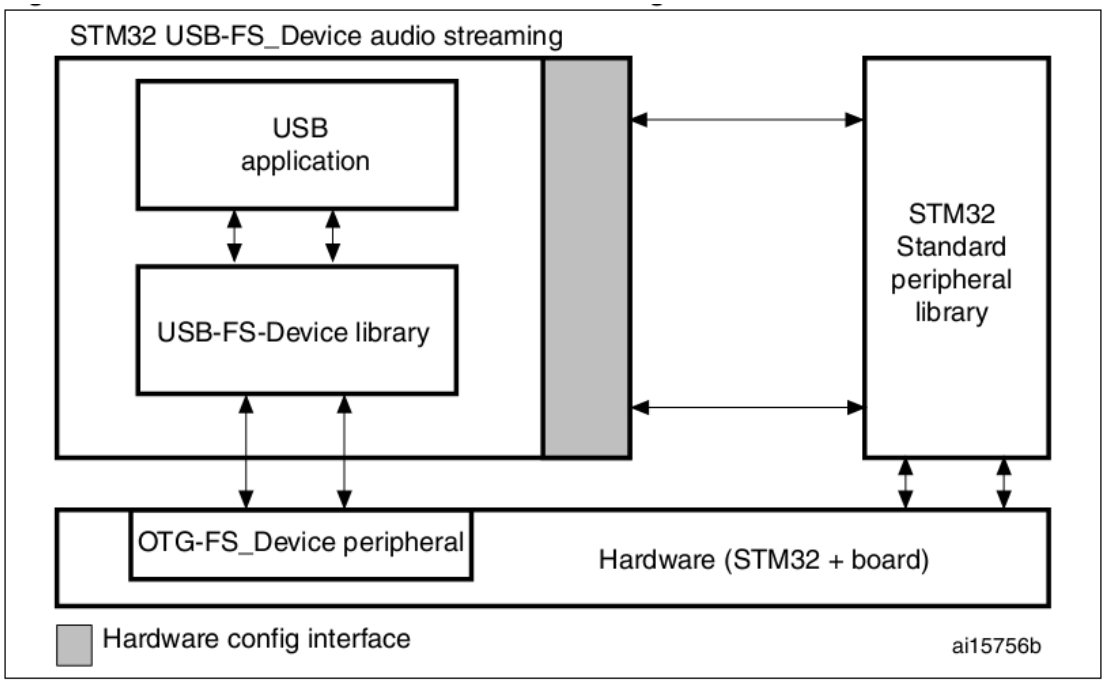

图 18：硬件和固件交互图

B）端点配置

在 STM32 USB 音频流演示中，两个端点用来和 PC 主机通信：端点 0 和端点 1。下面的 C 代码描述了配置一个同步输出端点所使用的方法（见 usb_prop.c 文件，Speaker_Reset（）函数）。

/ * Initialize Endpoint 1 */

OTGD_FS_EP_Init(ENDP1_OUT, EP_ISOCHRONOUS, ISOC_BUFFER_SZE);

ISOC_BUFFER_SZE 被定义在 usb_conf.h 文件中，计算方法如下：

ISOC_BUFFER_SZE = (2 * 2 * (音频)/1000)

例如：如果音频频率为 48 kHz，则 ISOC_BUFFER_SZE=（2 * 2* 48）= 192

C）特殊类要求

这个功能仅支持静音控制。此功能管理通过 Mute_command 函数（在 usb_prop.c 文件中）。

D）同步数据传输管理

通过可配置的自由的多缓冲区，同步传输被 DMA 管理。嵌入式 SRAM 分配的大尺寸的缓冲区分为多个子缓冲区，每一个子缓冲区的大小等于先前计算的 ISOC_BUFFER_SZE 的大小。子缓冲器的数量是一个整数。

例如，如果音频的频率为 48 kHz，则 ISOC_BUFFER_SZE 是 192。如果子缓冲器的数目是 30，那么的缓冲区总大小 Isoc_Buffer192* 30 =5.625 KB。子缓冲区的数量可以被配置通过在 usb_conf.h 文件定义 NUM_SUB_BUFFERS。

循环连续模式下，通过 DMA 将数据从这个缓冲区传输到 I2S 外设。同时，OTG-FS_Device 的外设拷贝接收到的数据从主机到该缓冲区。为了保证正确的传输，全局缓冲器被分为两部分。当 OTG-FS_Device 的传输在第一部分中执行，则 DMA 传输应该被执行在第二部分中，当 OTG-FS_Device 传输达到第二部分，DMA 应该开始读取第一部分。图 19 示出了这种机制。

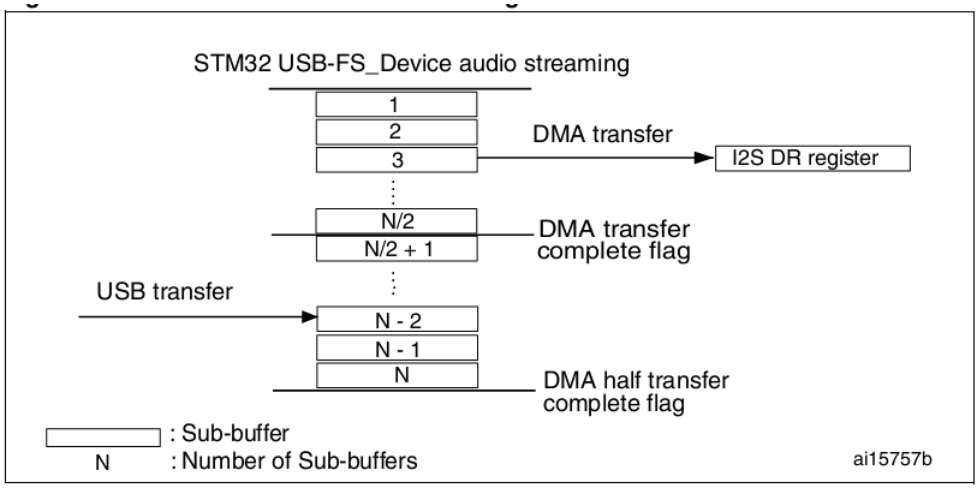

图 19：同步数据传输管理

如果主机停止传输数据到设备（暂停或停止流），这是通过 SOF 的 ISR 位检测是否一个新的包已经被接收。在这种情况下，DMA 传输被停止，并在音频输出静音。

音频扬声器描述符

表 22：设备描述符

文件	值	描述
bLength	0x12	描述符长度（字节）
bDescriptortype	0x01	描述符类型（设备描述符）
bcdUSB	0x0200	规范版本号：2.00
bDeviceClass	0x00	设备类
bDeviceSubClass	0x00	设备子类
bDeviceProtocol	0x00	设备协议
bMaxPacketSize0	0x40	端点 0 的最大数据包大小：64 字节
idVendor	0x0483	产商 ID（ST）
dProduct	0x5730	产品 ID
bcdDevice	0x0100	移动设备版本号：1.00
iManufacturer	0x01	制造商的字符串描述符的索引：1
iProduct	0x02	产品描述符索引：2
iSerialNumber	0x03	序列号字符串描述符索引：3
bNumConfigurations	0x01	可能的配置数：1

表 23：配置描述符

文件	值	描述
bLength	0x09	描述符长度（字节）
bDescriptortype	0x02	描述符类型（配置描述符）
wTotalLength	0x006D	描述符的总长度返回的数据（包括接口端点描述符）（以字节为单位）
bNumInterfaces	0x02	配置所支持的接口（2）
bConfigurationValue	0x01	配置值
Configuration	0x00	配置描述符的索引
bmAttributes	0x80	配置特点：总线供电
Maxpower	0x32	通过 USB 总线的最大功耗：100 毫安

表 24：接口描述符

文件	值	描述
USB 扬声器标准接口 AC 描述符（接口为 0，设置为 0）		
bLength	0x09	描述符长度（字节）
bDescriptortype	0x04	描述符类型（接口描述符）
bInterfaceNumber	0x00	接口数
bAlternateSetting	0x00	备用设置号
bNumEndpoints	0x00	使用端点 0（用于此接口只有端点 0）
bInterfaceClass	0x01	接口类型：USB DEVICE CLASS AUDIO
bInterfaceSubClass	0x01	接口子类：AUDIO SUBCLASS AUDIOCONTROL
bInterfaceProtocol	0x00	接口协议：AUDIO PROTOCOL UNDEFINED
iInterface	0x00	接口字符串描述符索引
USB 扬声器类专用 AC 接口描述符		
bLength	0x09	描述符长度（字节）
bDescriptortype	0x24	描述符类型：AUDIO INTERFACE DESCRIPTOR TYPE
bDescriptorSubtype	0x01	描述符子类型：AUDIO CONTROL HEADER
bcdADC	0x0100	bcdADC:1.00

wTotalLength	0x0027	总长度：39
bInCollection	0x01	流接口数：1
baInterfaceNr	0x01	baInterfaceNr：1
USB 音箱输入终端描述符		
bLength	0x0C	描述符长度（字节）
bDescriptortype	0x24	描述符类型：AUDIO INTERFACE DESCRIPTOR TYPE
bDescriptorSubtype	0x02	描述符子类：A UDIO CONTROL INPUT TERMINAL
bTerminalID	0x01	终端 ID：1
wTerminalType	0x0101	终端类型：AUDI O TERMINAL USB STREAMING
bAssocTerminal	0x00	无关联
bNrChannels	0x01	单通道
wChannelConfig	0x0000	信道配置：单声道
iChannelNames	0x00	保留
iTerminal	0x00	保留
USB 扬声器音频功能单元描述符		
bLength	0x09	描述符长度（字节）
bDescriptortype	0x24	描述符类型：AUDIO INTERFACE DESCRIPTOR TYPE
bDescriptorSubtype	0x06	描述符子类：AUDIO CONTROL FEATURE UNIT
bUnitID	0x02	单位 ID：2
bSourceID	0x01	来源 ID：1
bControlSize	0x01	控制尺寸：1
bmaControls	0x0001	支持静音控制
iTerminal	0x00	保留
USB 音箱输出终端描述符		
bLength	0x09	描述符长度（字节）
bDescriptortype	0x24	描述符类型：AUDIO INTERFACE DESCRIPTOR TYPE

bDescriptorSubtype	0x03	描述符子类：A UDIO CONTROL OUTPUT TERMINAL
bTerminalID	0x03	终端 ID：3
wTerminalType	0x0301	终端类型：AUDIO TERMINAL SPEAKER
bAssocTerminal	0x00	无关联
bSourceID	0x02	来源 ID：2
iTerminal	0x00	保留
USB 扬声器标准 AS 接口描述符 - 音频流零带宽（接口 1 个，备用设置 0）		
bLength	0x09	描述符长度（字节）
bDescriptortype	0x24	描述符类型：AUDIO INTERFACE DESCRIPTOR TYPE
bInterfaceNumber	0x01	接口数：1
bAlternateSetting	0x00	备用设定值：0
bNumEndpoints	0x00	不使用（零带宽）
bInterfaceClass	0x01	接口类：USB DEVICE CLASS AUDIO
bInterfaceSubClass	0x02	接口子类：AUDIO SUBCLASS AUDIOSTREAMING
bInterfaceProtocol	0x00	接口协议：AUDIO PROTOCOL UNDEFINED
iInterface	0x00	保留
USB 扬声器标准 AS 接口描述符 - 音频流操作（接口 1 个，备用设置 1）		
bLength	0x09	描述符长度（字节）
bDescriptortype	0x24	描述符类型：AUDIO INTERFACE DESCRIPTOR TYPE
bInterfaceNumber	0x01	接口数：1
bAlternateSetting	0x00	备用设定值：1
bNumEndpoints	0x01	1 个端点
bInterfaceClass	0x01	接口类：USB DEVICE CLASS AUDIO
bInterfaceSubClass	0x02	接口子类：AUDIO SUBCLASS AUDIOSTREAMING
bInterfaceProtocol	0x00	接口协议：AUDIO PROTOCOL UNDEFINED
iInterface	0x00	保留

USB 的扬声器音频流接口描述符		
bLength	0x07	描述符长度（字节）
bDescriptortype	0x24	描述符类型：AUDIO INTERFACE DESCRIPTOR TYPE
bInterfaceNumber	0x01	接口数：1
bAlternateSetting	0x00	备用设定值：1
bNumEndpoints	0x01	1 个端点
wFormatTag	0x0001	PCM 格式（16 位）
III 型 USB 扬声器音频格式接口描述符		
bLength	0x0b	描述符长度（字节）
bDescriptortype	0x24	描述符类型：AUDIO INTERFACE DESCRIPTOR TYPE
bDescriptorSubtype	0x03	描述符子类：A UDIO STREAMING FORMAT TYPE
bFormatType	0x03	格式类型：3
bNrChannels	0x02	通道数：2
bSubFrameSize	0x02	子帧大小：2 个字节音频子
bBitResolution	0x10	每个样本位分辨率：16 位
bSamFreqType	0x01	一个频率支持
tSamFreq	AUDIO_F REQ	根据 usb_conf.h 文件，该值将被自动设定