用户手册

STM32F10xxx USB 开发人员工具包

介绍

STM32F10xxx USB 开发人员工具包是一个完整的固件和软件包，并且包括所有 USB 传输方式（控制，中断，块，同步的方式）的实例和范例，支持 STM32F10xxx 系列的所有微控制器。

STM32F10xxx USB 开发人员工具包的目标是在每种 USB 传输方式都提供一个使用 STM32F10xxx USB 库的固件范例。本文档是对 STM32F10xxx USB 开发人员工具包的所有组件的描述，包括以下内容：

- STM32F10xxx USB 库，关于默认端点和标准请求的过程
- 设备固件升级范例：控制传输方式
- 操纵杆鼠标范例：中断传输方式
- 大容量存储范例：批量传输方式
- 虚拟 COM 端口：批量传输方式
- USB 音频范例（USB 扬声器）：同步传输方式。

www.mxchip.com 021-52655026/025

目录

1 STM32F10xxx USB固件库

本章节介绍用于管理 STM32F10xxx USB 2.0 全速设备外设的固件接口（称之为 USB 库）。

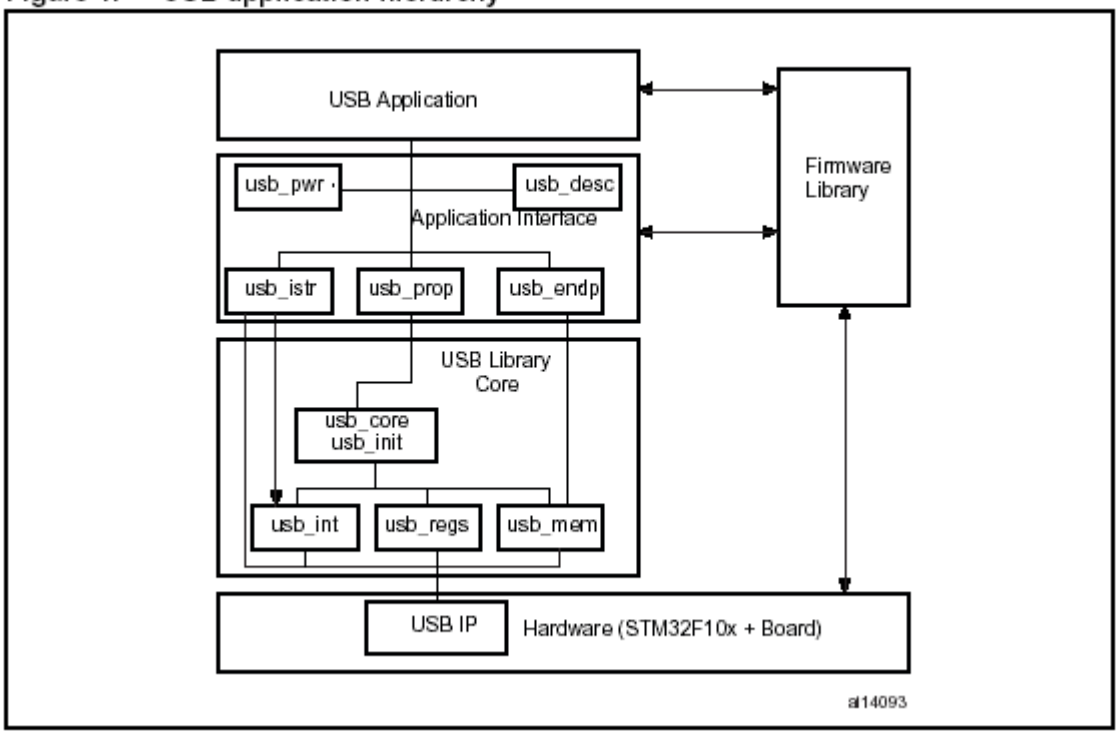
固件库的主要用途是利用 STM32F10xxx 系列微控制器家族中的 USB 宏单元来简化应用开发。

1.1 USB应用层次

图 1 显示了典型 USB 应用中不同组件和 USB 库之间的交互。

图 1 USB 应用层次

Figure 1. USB application hierarchy



USB 库分为两个层次：

- **USB 库内核层**：该层管理使用 USB IP 硬件和 USB 标准协议的直接传输。USB 库内核遵从 USB2.0 标准并和标准的 STM32F10xxx 固件库分离。
- **用户接口层**：本层为用户提供了库内核和最终应用之间的完整接口。

注意：用户接口层和最终应用可以与固件库通信来管理应用的硬件需求。

关于两个层的编码规则将在接下来的两个章节中介绍。

1.2 USB库内核

表 1 介绍了 USB 库的内核模块：

表 1 USB 库内核模块

文件	描述
Usb_type.h	库内核用到的数据类型，本文件用于保证 USB 库的独立性。
Usb_reg (.h,.c)	硬件抽象层
Usb_int.c	正确传输中断服务程序
Usb_init (.h,.c)	USB 初始化
Usb_core (.h,.c)	USB 协议管理(服从 USB2.0 规范的第九章)
Usb_mem (.h,.c)	数据传输管理（从包存储器区域发出的或者发往包存储器区域的）
Usb_def.h	USB 定义

1.2.1 usb_type.h

该文件提供了库中使用的主要数据类型，这些数据类型和使用的微控制器家族有关。

注意：USB 库中类型的定义和 STM32F10xxx 固件库中相同， 这保证了整个代码的一致性。

1.2.2 Usb_reg (.c,.h)

Usb_reg 模块实现了硬件抽象层，它提供了访问 USB 宏单元寄存器的一组基本函数。

注意：可用函数有两种调用方式：

– 作为宏：_调用方式是：函数名（参数 1，2...）

– 作为子程序：调用方式是：函数名（参数 1，2...）

通用寄存器函数

这些函数可以用于设置或获得不同寄存器的值。

表 2 通用寄存器函数

寄存器	函数
CNTR	Void SetCNTR (u16 wValue) U16 GetCNTR (void)
ISTR	Void SetISTR (u16 wValue) U16 GetISTR (void)
FNR	U16 GetIFNR (void)
DADDR	Void SetIDADDR(u16 wValue) U16 GetIDADDR(void)
BTABLE	Void SetBTABLE(u16 wValue) U16 GetBTABLEI(void)

端点寄存器函数

所有**端**点寄存器的操作可以通过 SetENDPOINT 和 GetENDPOINT 函数完成。但是，为了能够

直接对特定字段操作，从这两个函数派生了很多函数。

a) 设置/获取**端**点值

SetENDPOINT ：void SetENDPOINT(u8 bEpNum,u16 wRegValue)

bEpNum = 端点编号，wRegValue = 要写入的值

GetENDPOINT ：u16 GetENDPOINT(u8 bEpNum)

bEpNum = 端点编号

返回值：端点寄存器的值

b) 端点类型字段

端点类型方式字段可以取如下定义的值：

#define EP_BULK (0x0000) // 批量端点

#define EP_CONTROL (0x0200) // 控制端点

#define EP_ISOCHRNOUS (0x0400) // 同步端点

#define EP_INTERRUPT (0x0600) //中断端点

SetEPType : void SetEPType (u8 bEpNum, u16 wtype)

bEpNum = 端点编号, wtype = 端点类型

GetEPType : u16 GetEPType (u8 bEpNum)

bEpNum = 端点编号

返回值：如上定义的值。

c) 端点状态字段

端点状态寄存器的 STAT_TX/STAT_RX 字段可以取如下定义的值：

#define EP_TX_DIS (0x0000) //端点发送关闭

#define EP_TX_STALL (0x0010) // 端点发送延迟

#define EP_TX_NAK (0x0020) //端点发送不应答

#define EP_TX_VALID (0x0030) // 端点发送有效

#define EP_RX_DIS (0x0000) // 端点接收关闭

#define EP_RX_STALL (0x1000) //端点接收延迟

#define EP_RX_NAK (0x2000) //端点接收不应答

#define EP_RX_VALID (0x3000) //端点接收有效

SetEPTxStatus : void SetEPTxStatus(u8 bEpNum,u16 wState)

SetEPRxStatus : void SetEPRxStatus(u8 bEpNum,u16 wState)

bEpNum = 端点编号, wState = 如上定义的值

GetEPTxStatus : u16 GetEPTxStatus(u8 bEpNum)

GetEPRxStatus : u16 GetEPRxStatus(u8 bEpNum)

bEpNum = 端点编号

返回值：如上定义的值。

d) 端点种类字段

SetEP_KIND : void SetEP_KIND(u8 bEpNum)

ClearEP_KIND : void ClearEP_KIND(u8 bEpNum)

bEpNum = 端点编号

Set_Status_Out : void Set_Status_Out(u8 bEpNum)

Clear_Status_Out : void Clear_Status_Out(u8 bEpNum)

bEpNum = 端点编号

SetEPDoubleBuff ： void SetEPDoubleBuff(u8 bEpNum)

ClearEPDoubleBuff : void ClearEPDoubleBuff(u8 bEpNum)

bEpNum = 端点编号

e) 正确传输 Rx/Tx 字段

ClearEP_CTR_RX : void ClearEP_CTR_RX(u8 bEpNum)

ClearEP_CTR_TX : void ClearEP_CTR_TX(u8 bEpNum)

bEpNum = 端点编号

f) 数据翻转 Rx/Tx 字段

ToggleDTOG_RX : void ToggleDTOG_RX(u8 bEpNum)

ToggleDTOG_TX : void ToggleDTOG_TX(u8 bEpNum)

bEpNum = 端点编号

g) 地址字段

SetEPAdress : void SetEPAddress(u8 bEpNum,u8 bAddr)

bEpNum = 端点编号

bAddr =要设置的地址

GetEPAdress : BYTE GetEPAddress(u8 bEpNum)

bEpNum = 端点编号

缓冲描述符函数

这些函数用于设置或获取端点接收和发送缓冲地址和大小

a) Tx/Rx 缓冲地址字段

SetEPTxAddr : void SetEPTxAddr(u8 bEpNum,u16 wAddr);

SetEPRxAddr : void SetEPRxAddr(u8 bEpNum,u16 wAddr);

bEpNum = 端点编号

wAddr = 要设置的地址（PMA 缓存地址表示)

GetEPTxAddr : u16 GetEPTxAddr(u8 bEpNum);

GetEPRxAddr : u16 GetEPRxAddr(u8 bEpNum);

bEpNum = 端点编号

返回值：地址（PMA 缓冲地址表示）

b) Tx/Rx 缓冲计数器字段

SetEPTxCount : void SetEPTxCount(u8 bEpNum,u16 wCount);

SetEPRxCount : void SetEPRxCount(u8 bEpNum,u16 wCount);

bEpNum = 端点编号

wCount = 要设置的计数器值

GetEPTxCount : u16 GetEPTxCount(u8 bEpNum);

GetEPRxCount : u16 GetEPRxCount(u8 bEpNum);

bEpNum = 端点编号

返回值：计数器值

双缓存端点函数

双缓存模式是为了在批量和同步模式下得到高速的数据传输率而设计的。在这种模式下端点寄存器和缓冲描述符单元的某些字段会有不同的含义。

为了简化这一特性的使用，开发出一些函数。

SetEPDoubleBuff：工作在批量模式下的断点可以通过设置 EP-KIND 位进入双缓存模式。SetEPDoubleBuff() 函数完成此任务。

SetEPDoubleBuff : void SetEPDoubleBuff(u8 bEpNum);

bEpNum = 端点编号

FreeUserBuffer: 在双缓存模式下端点变为为单向的 ,不用方向的缓存描述符单元可以用于处理其他的缓存区。

地址和计数器必须使用不同的方法操作。Rx 和 Tx 地址和计数器单元**变为**缓存 0 和缓存 1。用于这个操作模式的函数在库中提供。

在批量传输时，传输线路填满其中一个缓存，另一个缓存保留给应用。用户应用必须在批量传输需要缓存之前处理数据。为应用保留的缓存必须及时地释放。

使用 **FreeUserBuffer** 函数释放使用中的缓存

FreeUserBuffer: void FreeUserBuffer(u8 bEpNum, u8 bDir);

bEpNum = 端点编号

a) 双缓存地址

这些函数再双缓存模式下从缓存描述符中设置或获取缓存的地址。

SetEPDblBuffAddr : void SetEPDblBuffAddr(u8 bEpNum,u16 wBuf0Addr,u16 wBuf1Addr);

SetEPDblbuf0Addr : void SetEPDblBuf0Addr(u8 bEpNum,u16 wBuf0Addr);

SetEPDblbuf1Addr : void SetEPDblBuf1Addr(u8 bEpNum,u16 wBuf1Addr);

bEpNum = 端点编号

wBuf0Addr, wBuf1Addr =缓存地址 (PMA 缓存地址表示)

GetEPDblBuf0Addr : u16 GetEPDblBuf0Addr(u8 bEpNum);

GetEPDblbuf1Addr : u16 GetEPDblBuf1Addr(u8 bEpNum);

bEpNum = 端点编号

返回值： 缓存地址

b) 双缓存计数器

这些函数再双缓存模式下从缓存描述符中设置或获取缓存计数器的值。

SetEPDblBuffCount: void SetEPDblBuffCount(u8 bEpNum, u8 bDir, u16 wCount);

SetEPDblBuf0Count:void SetEPDblBuf0Count(u8 bEpNum, u8 bDir, u16 wCount);

SetEPDblBuf1Count:void SetEPDblBuf1Count(u8 bEpNum, u8 bDir, u16 wCount);

bEpNum = 端点编号

bDir = 端点方向

wCount = 缓存计数器值

GetEPDblBuf0Count : u16 GetEPDblBuf0Count(u8 bEpNum);

GetEPDblBuf1Count : u16 GetEPDblBuf1Count(u8 bEpNum);

bEpNum = 端点编号

返回值: 缓存计数器

c) 双缓存状态字段

在单一缓存和双缓存模式中使用一样的函数管理端点状态(除了 STALL 状态仅用于双缓存状态), 这个功能由下面这个函数管理。

SetDouBleBuffEPStall: void SetDouBleBuffEPStall(u8 bEpNum,u8 bDir)

bEpNum = 端点编号

bDir = 端点方向

1.2.3 usb_int (.c , .h)

usb_int 模块处理正确的传输中断服务程序，提供了 USB 协议事件和库内核之间的连接。

The STM32F10xxx USB 外设提供两种正确传输例程。

- 低优先级中断：由 CTR_LP()函数管理，在控制，中断，批量模式下使用（单缓存模式）。
- 高优先级中断：由 CTR_HP()函数管理，在快速传输方式（如同步，批量模式）(双缓存模式）

1.2.4 usb_core (.c , .h)

usb_core 模块是库的内核，实现了 USB 2.0 规范中第九章描述的所有功能。

可用的子例程覆盖了和控制端点（EP0）有关的 USB 标准请求处理以及为完成列举过程所需代码的提供。

为了处理设置事务的不同阶段，内核实现了一个状态机。

USB 内核模块还利用结构体 User_Standard_Requests 实现了标准请求和用户实现之间的动态接口。

只要需要，USB 内核可以将一些类专用的请求和总线事件分配给用户程序处理。用户处理过程在 Device_Property 结构中给出。

下面的段落中给出内核中使用的各种数据和函数结构。

1. 设备表结构

内核将设备级信息保存在设备表结构中，设备表结构是 **DEVICE** 类型，类型定义如下：

```
typedef struct _DEVICE {
  u8 Total_Endpoint;
  u8 Total_Configuration;
} DEVICE;
```

2. 设备信息结构

USB 内核将主机发送过来的用于实现 USB 设备的设置包保存在设备信息结构中，该结构类型是 **DEVICE_INFOR**，类型定义如下：

```
typedef struct _DEVICE_INFO {
  u8 USBbmRequestType;
  u8 USBbRequest;
  u16_u8 USBwValues;
  u16_u8 USBwIndexs;
  u16_u8 USBwLengths;
  u8 ControlState;
  u8 Current_Feature;
  u8 Current_Configuration;
  u8 Current_Interface;
u8 Current_AlternateSetting;
  ENDPOINT_INFO Ctrl_Info;
} DEVICE_INFO;
```

为了简化对 DEVICE_INFO 结构中的某些字段的访问（以 u16 或 u8 格式），定义了一个共用体 **u16_u8**。

```
typedef union {
  u16 w;
  struct BW {
  u8 bb1;
  u8 bb0;
  } bw;
} u16_u8;
```

结构体字段描述

- **USBbmRequestType 是**设置包中的 bmRequestType 的副本
- **USBbRequest 是**设置包中的 bRequest 的副本
- **USBwValues** 定义为 WORD_BYTE 类型并可以通过 3 个宏访问：

  #define USBwValue USBwValues.w

  #define USBwValue0 USBwValues.bw.bb0

  #define USBwValue1 USBwValues.bw.bb1

  USBwValue 是设置包中的 wValue 的副本

  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 是设置包中的 wIndex 的副本

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

– USBwLengths 定义为 WORD_BYTE 类型并可以通过 3 个宏访问：

#define USBwLength USBwLengths.w

#define USBwLength0 USBwLengths.bw.bb0

#define USBwLength1 USBwLengths.bw.bb1

USBwLength 设置包中的 wLength 的副本

USBwLength0 是 wLength,的低字节, USBwLength1 是 wLength 的高字节

– **ControlState** 是内核的状态，可用的值定义在 CONTROL_STATE 中。

– **Current_Feature** 反映了当前设备的特性。由 SET_FEATURE ，CLEAR_FEATURE 改写，GET_STATUS 来取值。用户代码不使用这个字段。

– **Current_Configuration** 指工作中设备的设置。由 SET_CONFIGURATION 和 GET_CONFIGURATION 分别设值和取值。

– **Current_Interface** 是选中的接口。

– **Current_Alternatesetting** 是为当前工作设置和接口的选中的备用设置。由 SET_ _INTERFACE 和 GET_ _INTERFACE 分别设值和取值。

– Ctrl_Info 是 ENDPOINT_INFO 类型

全局变量 pInformation 用于简化对设备信息表的访问 ,是一个指向 DEVICE_INFO 结构的指针。事实上， pInformation = &Device_Info.

3. 设备属性结构

USB 内核在必要时将控制权交给用户程序。用户处理过程以 Device_Property 数组给出，该结构属于 DEVICE_PROP 类型，类型定义如下：

typedef struct _DEVICE_PROP {

```
void (*Init)(void);

void (*Reset)(void);

void (*Process_Status_IN)(void);

void (*Process_Status_OUT)(void);

RESULT (*Class_Data_Setup)(u8 RequestNo);

RESULT (*Class_NoData_Setup)(u8 RequestNo);

RESULT    (*Class_Get_Interface_Setting)(u8 Interface,u8

AlternateSetting);

u8* (*GetDeviceDescriptor)(u16 Length);

u8* (*GetConfigDescriptor)(u16 Length);

u8* (*GetStringDescriptor)(u16 Length);

u8 MaxPacketSize;

} DEVICE_PROP;
```

4. 用户标准请求结构

用户标准需求结构是用户代码和标准请求管理之间的接口，该结构属于**USER_STARDARD_REQUESTS** 类型，类型定义如下：

```
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;

如果用户想要在收到标准 USB 请求后实现特定的代码 那么将不得不使用本结构中的相应函数。

应用开发人员必须实现 DEVICE_PROP，Device_Table ，USER_STANDARD 这三个树结构（用于管理类专用请求和应用相关的控制）。这些结构中的不同字段会在下个章节中介绍。

1.3 应用接口

应用接口模块是作为一个模版提供给用户的，开发人员可以根据实际的应用对模版进行裁剪，表 3 显示了应用接口中使用的各种模块。

表 3 应用接口模块

文件	**描述**
usb_istr (.c,.h)	USB 中断处理函数
usb_conf.h	USB 配置文件
usb_prop (.c, .h)	USB 应用相关属性
usb_endp.c	CTR 中断处理程序（非控制端点）
usb_pwr (.h, .c)	USB 电源管理模块
usb_desc (.c, .h)	USB 描述符

1.3.1 usb_istr(.c)

USB_istr 提供了一个名为 **USB_Istr**() 的函数，该函数用于处理所有的 USB 宏单元中断。每一个 USB 中断源有一个名为 XXX_Callback（如 RESET_Callback）的回调函数，这些回调函数用于实现用户中断处理器。为了启用回调程序，必须在 USB 配置文件 USB_conf.h 中定义名为 XXX_Callback 的与处理程序开关。

1.3.2 usb_conf(.h)

usb_conf.h 用于：

- 定义 BTABLE 和 PMA 中的所有端点地址
- 定义相应事件中的中断掩码

1.3.3 usb_endp (.c)

USB_endp 模块处理除端点 0（EP0）外所有的 CTR 的正确传输程序。

为了启用回调处理器进程，必须在 **USB_conf.h** 中定义一个名为 EPx_IN_Callback (IN 传输) 和 EPx_OUT_Callback (OUT 传输)的预处理器

1.3.4 usb_prop (.c , .h)

USB_prop 模块实现了 USB 内核使用的 Device_Property, Device_Table 和 USER_STANDARD_REQUEST 结构。

设备属性的实现

设备属性结构字段的描述如下：

- **void Init(void):** USB IP 的初始化过程. 在应用开始时调用一次，用于管理初始化进程。
- **void Reset(void):** USB IP 复位过程. 当宏单元收到 RESET 信号时调用，用户程序应该在此过程中设置端点（设置默认的控制端点并使能接收）。
- **void Process_Status_IN(void):** 回调函数,当一个进入某个阶段状态结束时调用。用户程序可以使用这个回调函数执行类或者应用相关的过程。
- **void Process_Status_OUT(void):** 回调函数，当一个退出某个阶段状态结束时调用，和 Process_Status_IN 一样，用户程序可以在退出某个阶段状态后执行一些动作。
- **RESULT**（看下面的注意事项）***(Class_Data_Setup)(BYTE RequestNo)**: 回调函数, 当一个需要 data stage 的类请求到来时调用，内核无法处理该请求,在这种情况下，用户代码使用用户编写的代码分析请求，准备数据，并把数据传给内核。再和主机交换信息。参数 RequestNo 显示了请求的数量，返回值为: RESULT 类型.。表示给内核的请求处理结果
- **RESULT (*Class_NoData_Setup)(BYTE RequestNo)** 回调函数, 当一个不需要 data stage 的非标准设备请求到来时调用，内核无法处理该请求，用户代码使用用户编写的代码分析请求并处理，返回值为: RESULT 类型.。表示给内核的请求处理结果
- **RESULT (*Class_GET_Interface_Setting)(u8 Interface, u8 AlternateSetting):**该回调用于测试收到的设置接口的标准请求。用户必须根据它们自身的实现检验接口和预备设置，如果这两个域出错则返回 USB_UNSUPPORT。
- **BYTE* GetDeviceDescriptor(WORD Length):** 内核得到设备描述符
- **BYTE* GetConfigDescriptor(WORD Length):** 内核得到配置描述符
- **BYTE* GetStringDescriptor(WORD Length):** 内核得到字符串描述符

– **WORD MaxPacketSize:** 默认控制端点的最大分组大小

注意: RESULT 类型如下

```
typedef enum _RESULT {
USB_SUCCESS = 0,/* 请求成功进行 */
USB_ERROR,          /* 错误
USB_UNSUPPORT,    /* 请求不支持
USB_NOT_READY/* 请求过程还没有完成,*/
/*端点将对此后的请求不应答*/
} RESULT;
```

设备表实现

结构字段描述：

– **Total_Endpoint** 代表 USB 应用使用的端点数量

– **Total_Configuration** 代表 USB 应用使用的配置数量

USER_STANDARD_REQUEST 实现

该结构用于管理接收到所有标准请求(获取描述符请求除外)之后用户处理。该结构的字段有：

– **void (*User_GetConfiguration)(void):** 在收到 Get Configuration（获得配置）标准请求后调用。

– **void (*User_SetConfiguration)(void):** 在收到 Set Configuration（设置配置）标准请求后调用。

– **void (*User_GetInterface)(void):** 在收到 Get interface（获取接口）标准请求后调用。

– **void (*User_SetInterface)(void):** 在收到 Set interface（设置接口）标准请求后调用。

– **void (*User_GetStatus)(void):** 在收到 Get interface（获得状态）标准请求后调用。

– **void (*User_ClearFeature)(void):** 在收到 Clear Feature（清除） 标准请求后调用。

– **void (*User_SetEndPointFeature)(void):** 在收到 Set Feature（设置） 标准请求后调用。

– **void (*User_SetDeviceFeature)(void):** 在收到 Set Feature（设置） 标准请求后调用。

– **void (*User_SetDeviceAddress)(void):** 在收到 set Address（设置地址） 标准请求后调用。

1.3.5 usb_pwr (.c , .h)

本模块管理 USB 设备的电源，提供了表 4 种列出的函数。

表 4 电源管理函数

函数名	描述
RESULT Power_on(void)	处理开关开情形
RESULT Power_off(void)	处理开关关情形
void Suspend(void)	设置挂起模式运行清醒
Void Resume(RESUME_STATE eResumeSetVal)	处理唤醒操作

1.4 用STM32F10xxx USB 库实现USB应用

1.4.1 实现无数据类专用请求

所有不带数据传输的类专用请求都实现了设备属性结构的 RESULT (*Class_NoData_Setup)(BYTE RequestNo) 字段。请求的 RequestNo 参数可以为 USBbRequest, 其他的请求字段保持在设备信息结构体中。用户需要测试请求的所有字段，如果请求可以用类实现，

函数返回 USB_SUCCESS，，否则，函数返回 UNSUPPORT，库以 STALL 握手的方式来应答。

1.4.2 如何实现数据类专用请求

当类请求事件需要数据传输时，用户实现向 USB 库报告所需数据的长度和内存中的位置（如果是从主机接收的数据则是 RAM，如果是发往主机的则是 RAM 或者 Flash）。这类的请求由 **RESULT (*Class_Data_Setup)(BYTE RequestNo)** 函数管理，

用户需要为每个类数据传输请求按下列的格式编写特定的函数：

u8* My_First_Data_Request (u16 Length)

如果 Length 参数为 0 ，则把 pInformation->Ctrl_Info.Usb_wLength 字段置为要传输的数据长度，并返回空指针，其它情况返回数据传输地址。

下面是一个用 C 实现的例子

```
  u8* My_First_Data_Request (u16 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)(BYTE RequestNo) 像下面 C 代码一样管理所有数据请

求。

```
RESULT Class_Data_Setup(u8 RequestNo)

{

u8*(*CopyRoutine)(u16);

    CopyRoutine = NULL;

if (My_First_Condition)//测试第一个请求的字段

CopyRoutine = My_First_Data_Request;

else if(My_Second_Condition) //测试二个请求的字段

 CopyRoutine = My_Second_Data_Request;

/*

   每个类数据请求的实现...

      ...

*/

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 */
```

1.4.2 如何管理非控制端点的数据传输

使用默认端点（端点 0）之外管道的数据传输管理在 usb_endp.c 文件中实现。

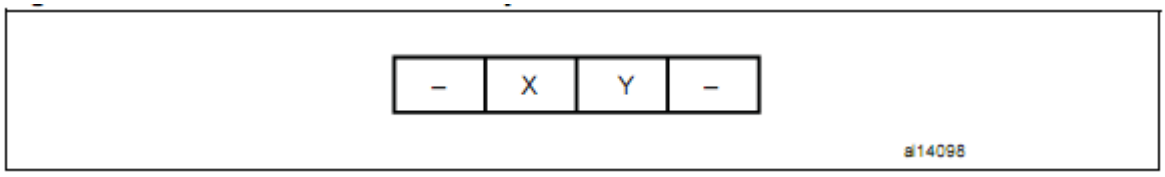
用户不能把文件 usb_conf.h 文件中这个端点(带方向)相关的行注释掉。

2 操纵杆鼠标范例

USB 鼠标（人机接口设备类）是一个简单但完整的 USB 应用的例子，操纵杆鼠标只使用一个中断端点（IN 方向的端点 1）。正常的枚举后，主机请求鼠标 HID 报告描述符，这个描述符（和标准描述符一期）出现在 usb_desc.c 文件中。

为了获取鼠标的位置，主机通过管道 1（端点 1）请求一个 4 个字节的数据，这个数据格式如图 2 所示 。

图 2 四字节的格式



鼠标范例演示的是根据用户的操作杆按钮动作设置 X Y 的值。JoyState()函数得到用户的动作，并返回鼠标的方向。Joystick_Send() 函数把要发给主机给主机的数据格式化并检查数据事务过程。

注意：函数的细节请看 hw_config.c 文件

3 设备固件升级

本部分介绍了 STM32F10xxx 微处理器的设备固件升级实现的能力，遵守了 USB 实现者论坛为

通过 USB 重编程应用定义的备固件升级规格说明。设备固件升级特别适合于需要重编程的 USB 应用。

同一个 USB 连接器可以同时用于标准操作模式和重编程进程。

大部分意法半导体 USB Flash 微控制器都具有支持 IAP 的特性，IAP 特性使得可以通过信道对 Flash 微控制器进行重编程，从而 DFU 操作就可以进行。

设备固件升级进程像其它 IAP 进程一样，是基于执行位于 Flash 中的固件，固件根据设备性能管理其他 Flash 模块的擦除和编写操作。可以是连接在微控制器的代码 Flash，数据 Flash，EEPROM 或者其他存储器，甚至可以使串行 Flash(通过 SPI 或 IIC 等.). STM32F10xxx 中的 DFU 范例是编程 STM32F10xxx-128K-EVAL 板上的内部存储器 和 SPI Flash 存储器。

注意：如果用户应用程序要写入的内部存储器有写或读保护，需要在使用 DFU 之前关保护。

3.1 DFU扩展协议

3.1.1 介绍

DFU 使用 USB 作为微控制器和编程工具之间的通信信道。通常是 PC。DFU 类规格说明中指出所有的命令，状态和数据交换需要通过端点 0 进行。命令集和基本协议都定义好了,但是上层协议（数据格式，错误信息等）是卖主相关的。也就是说 DFU 类没有定义数据传输格式（s19，16 进制，纯 2 进制等），

因为一个设备同时进行 DFU 操作和正常运行活动是不现实的，所以在 DFU 操作期间必须停止正常运行活动，这就意味着设备必须改变运行模式：也就是在进行固件更新的时候打印机不再是打印机，它是一个 Flash/存储器编程器，但是支持 DFU 的设备不能自主改变模式，这需要外部（人或者主机操作系统）的干预

3.1.2 阶段

完成实现固件升级可以分为 4 个不同的阶段。

1．枚举

设备把自身的一些特性告知主机，嵌入在设备正常运行描述符中的一个 DFU 类接口描述符和相关的函数符能够完成这个目的，并且能够为通过控制管道的类专用请求提供目标。

2. DFU 枚举

主机和设备同意开始固件升级，主机向设备发出 USB 复位，设备发出第二个描述符集合，并为传输阶段做准备，这会使相应设备的运行时设备驱动无效。并使得 DFU 驱动不受其他目标为该设备的通信的妨碍，重编程设备的固件。

3. 传输

主机将固件映像传输给设备，功能描述符中的参数用于确保非易失性存储器编程的块大小和时序的正确性。状态请求用于保持主机和设备之间的同步。

4. 显示,

一旦设备向主机报告重编程完成，主机向设备发出 USB 复位，设备重枚举并执行升级后的固件。

为保证只有 DFU 驱动加载，有必要在枚举 DFU 描述符集时改变 id- Product 字段。

这保证了 DFU 驱动将会在操作系统匹配到一特定驱动一致的 vendor ID 和 product ID 后加载。

3.1.3 请求

在升级操作中需要一些 DFU 类专用的请求。

表 5 DFU 类专用的请求总结

bmRequest	bReques	wValue	wIndex	wLength	Data

00100001b	DFU_DETACH (0)	wTimeout	Interface	0	无
00100001b	DFU_DNLOAD (1)	wBlockNum	Interface	长度	固件
10100001b	DFU_UPLOAD (2)	wBlockNum	Interface	长度	固件
10100001b	DFU_GETSTATUS(3)	0	Interface	6	状态
00100001b	DFU_CLRSTATUS (4)	0	Interface	0	无
10100001b	DFU_GETSTATE (5)	0	Interface	1	状态
00100001b	DFU_ABORT (6)	0	Interface	0	无

这些请求的更多信息，请参阅 DFU 类规格说明。

3.2 DFU 模式选择

主机应该可以以两种方式对支持 DFU 的设备进行枚举。

- 只支持 DFU 的单一设备 DFU 方式
- 合成设备：HID，大容量存储，功能类，DFU 性能。

在枚举阶段， 设备在合适的时间分别给出两个两个不同并且独立的描述符集：

- 运行描述符集：在设备正常操作显示。
- DFU 模式描述符集: 在主机和设备进行 DFU 操作时显示

3.2.1 运行时描述集

在设备正常操作时。设备有正常描述符集和两个附加的描述符集：

- 运行 DFU 接口描述符集
- 运行功能描述符集

注意：在每个设置描述符中支持 DFU 的接口数量，必须加 1 以和 DFU 接口描述符的增加一致。

3.2.2 DFU模式描述符集

在主机和设备进行 DFU 操作时，主机重枚举设备，这时设备发出如下的描述符集

- DFU 模式设备描述符
- DFU 模式配置描述符
- DFU 模式借口描述符
- DFU 模式功能描述符：和运行时 DFU 功能描述符相同

DFU 模式设备描述符

该描述符只出现在 DFU 模式描述符集中。

表 6 DFU 模式设备描述符

偏移	字段	大小	值	描述
0	bLength	1	12h	描述符大小（字节）
1	bDescriptorType	1	01h	描述符类型
2	bcdUSB	2	0100h	二进制 USB 规格说明发行号
4	bDeviceClass	1	00h	见接口
5	bDeviceSubClass	1	00h	见接口

6	bDeviceProtocol	1	00h	见接口
7	bMaxPacketSize0	1	8,16,32, 64	端点 0 的最大分组大小 8 ST7 低速 32： 16： 64 ST7 全速 STR7/9 &STM32F10xxx 设备
8	idVendor	1	0483h	Vendor ID
10	idProduct		DF11h	产品 ID
12	bcdDevice		011Ah	STMicroelectronics DFU 扩展规格说明 版本
14	iManufacturer		Index	字符串描述符索引
15	iProduct		Index	字符串描述符索引
16	iSerialNumber		Index	字符串描述符索引
17	bNumbConfigurations		01h	DFU.设置

DFU 模式配置描述符

除了 bNumInterfaces 必须为 01h.之外，本描述符和标准配置描述符（USB 规格说明 1.0）相同。

- DFU 模式接口描述符

这是 DFU 模式下唯一可用的接口的描述符，bInterfaceNumber 字段通常为 0。

表 7 DFU 模式接口描述符

偏移	字段	大小	值	描述
0	bLength	1	09h	描述符大小（字节）
1	bDescriptorType	1	04h	描述符类型
2	bInterfaceNumber	1	00h	接口数
3	bAlternateSetting	1	Number	备用设置
4	bNumEndpoints	1	00h	只使用控制管道
5	bInterfaceClass	1	FEh	应用相关类代码
6	bInterfaceSubClass	1	01h	设备固件升级代码
7	bInterfaceProtocol	1	00h	本接口中不使用类专用协议
8	iInterface	1	Index	接口字符串描述符索引

- 备用设置和字符串描述符定义

本章节描述了 ST 微电子实现的而未在标准 DFU 规格说明中的 ,预备的设置和字符串描述符定义。

备用设置用于访问附加的存储器段和其他存储器（Flash 存储器, RAM, EEPROM），这些存储器可能 CPU 存储器映射中实现也可能没有实现，如外部串行 SPI Flash 存储器或者外部 NOR/NAND Flash 存储器。

在这样的情况下，每个备用设置使用一个字符串描述符来说明目标存储器的段，如下所示：

@Target Memory Name/Start Address/Sector(1)_Count*Sector(1)_Size

Sector(1)_Type,Sector(2)_Count*Sector(2)_SizeSector(2)_Type,.....,

Sector(n)_Count*Sector(n)_SizeSector(n)_Type

STM32F10xxx Flash 微控制器的另外的一个例子：

@Internal Flash /0x08000000/12*001Ka,116*001Kg"

每个备用设置字符串描述符必须遵循如下的存储器映射，否则 PC 主机软件无法为选中的设备进行正确的映射解码：

- @: 表示这是一个特殊的映射描述符（避免按照标准描述符解码）
- /: 不同区之间的分隔符
- 最大 8 位的地址，以“0X”开头
- /: 不同区之间的分隔符
- 最大 2 位的扇区编号
- *: 扇区数量和扇区大小之间的分隔符
- 最大 3 位的扇区大小(0-999)
- 1 位的扇区大小单位.有效的输入是: B (字节), K (千), M (兆)
- 1 位的扇区类型:

  a (0x41): 可读

  b (0x42): 可擦除

  c (0x43): 可读可擦除

  d (0x44): 可写

e (0x45): 可读可写

f (0x46): 可写可擦除

g (0x47): 可读可写可擦除

注意 如果目标存储器不是连续的，用户可以在"/"之后加入要解码的新的扇区，如下所示：

"@Flash /0xF000/1*4Ka/0xE000/1*4Kg/0x8000/2*24Kg"

- DFU 功能描述符

  运行时和 DFU 模式描述符集中的描述符相同

表 8 DFU 功能描述符

偏移	字段	大小	值	描述
0	bLength	1	09H	描述符大小（字节）
1	bDescriptorType	1	21H	DFU 功能描述符类型
2	bmAttributes	1	00H	DFU 属性 – 位 7：如果该位被置位，设备将会将上传速度增加到 4096 字节/命令（bitCanAccelerate） 0: No 1:Yes – 位 6:4：保留 位 3： 收到 DFU_DETACH 命令时会进行总线 detach-attac 操作。 0 = No

				1 = Yes 注意：主机不能发布 USB 复位(bitWillDetach) – 位 2：设备在显示阶段之后可以通过 USB 通信(bitManifestation tolerant) 0 = No，必须出现总线复位 1 = Yes – 位 1：上传性能（bitCanUpload） 0 = No 1 = Yes – 位 0：下载性能（bitCanDnload） 0 = No 1 = Yes
3	wDetachTimeOut	2	Number	设备收到 DFU_DETACH 请求后的等待时间（毫秒），如果超时还没 USB 复位，设备中止重配置阶段转到正常操作，这是等待的最大时间（根据定时器）主机可能在 DFU_DETACH 命令中给出更短的超时时间。
5	wTransferSize	2	Number	设备每次控制写事务能接收的最大字节数。wTransferSize 取决于每个 MCU 固件的实现
7	bcdDFUVersion	2	011AH	STMicroelectronics DFU 扩展规格说明发行号

3.3 重配置阶段

一旦操作者确定设备并提供文件名后，主机和设备协商进行升级。

- 主机向端点 0 发出 DFU_DETACH 请求。
- 主机向设备发出 USB 复位请求，有些 PC 主机 OS 板本不能发出 USB 复位请求。为了避免出现此类问题，根据相关的实现 USB 复位请求可以由 MCU 进行。
- 向上面描述的一样，设备枚举 DFU 模式描述符集。

注意：

1. 一些设备的应用在运行模式下可能不使用 USB,如电动机控制应用或者安全系统，USB 可能只能用于存储器升级，这样的设备被叫做非 USB 应用（在本文档中），上述的一些操作也不能用于这些设备。
2. 非 USB 应用需要执行一些过程以进入 DFU 模式，这些可以再 USB 复位的时候简单的通过插入 USB 电缆或者跳转到 DFU 固件代码来实现，这样设备会枚举 DFU 描述符集。

3.4 传输阶段

传输阶段在 USB 复位和输出 DFU 模式描述符集之后开始，本阶段可以上传和下载固件。传输阶段有一连串的 DFU 请求组成，这将在下面的章节讨论。

3.4.1 请求

升级和上传操作需要 一些 DFU 类专用请求。

表 9 DFU 升级/上传请求总结

bmRequest	bRequest	wValuew	**wIndex**	wLength	数据
00100001b	DFU_DNLOAD(1)	wBlockNum	接口	长度	固件
10100001b	DFU_UPLOAD (2)	wBlockNum	接口	长度	固件
10100001b	DFU_GETSTATUS(3)	0	接口	6	状态
00100001b	DFU_CLRSTATUS (4)	0	接口	0	无
10100001b	DFU_GETSTATE (5)	0	接口	1	状态
00100001b	DFU_ABORT (6)	0	接口	0	无

这些请求的更多信息，请参阅 DFU 类规范。

3.4.2 特殊指令/协议描述

为了支持所有的特性（地址的解码和不、内存块擦除等功能），ST 微电子实现了 DFU 扩展，在 DFU_DNLOAD 请求中加入了一些格式规则，如表 10 中的定义：

表 10 特殊命令描述

命令	请求	wBlockNum	wLength	数据
Get 命令	DFU_DNLOAD	0	1	0x00
设置地制指针	DFU_DNLOAD	0	5	21h 地址（4 字节）

擦除含地址的扇区	DFU_DNLOAD	0	5	41h 地址（4 字节）

这些新的客户 DFU 实现只支持 3 条基本指令：

- Get 命令

  如果 Byte0 = 0x00 则无附加字节

  下一条 wBlockNum = 0 的 DFU_UPLOAD 请求需要给于支持的命令。

  支持指令的最大缓存为 256 字节，并且缓存必须支持如下命令：

  – 0x00 (Get 命令)

  – 0x21 (设置地址指针)

  – 0x41 (擦除带地址的扇区)

- 设置地址指针

  Byte0=0x21，那么接下来的四个字节包含了一个地址指针，这个地址是下一条 wBlockNum>1 的 DFU_DNLOAD 或者 DFU_UPLOAD 请求将下载或者上传的数据块的起始地址。

- 擦除带地址的扇区

  Byte0=0x41，那么接下来的四个字节包含了一个存储器扇区中的要擦除并且由备用设置的字符串描述符指出的有效地址。

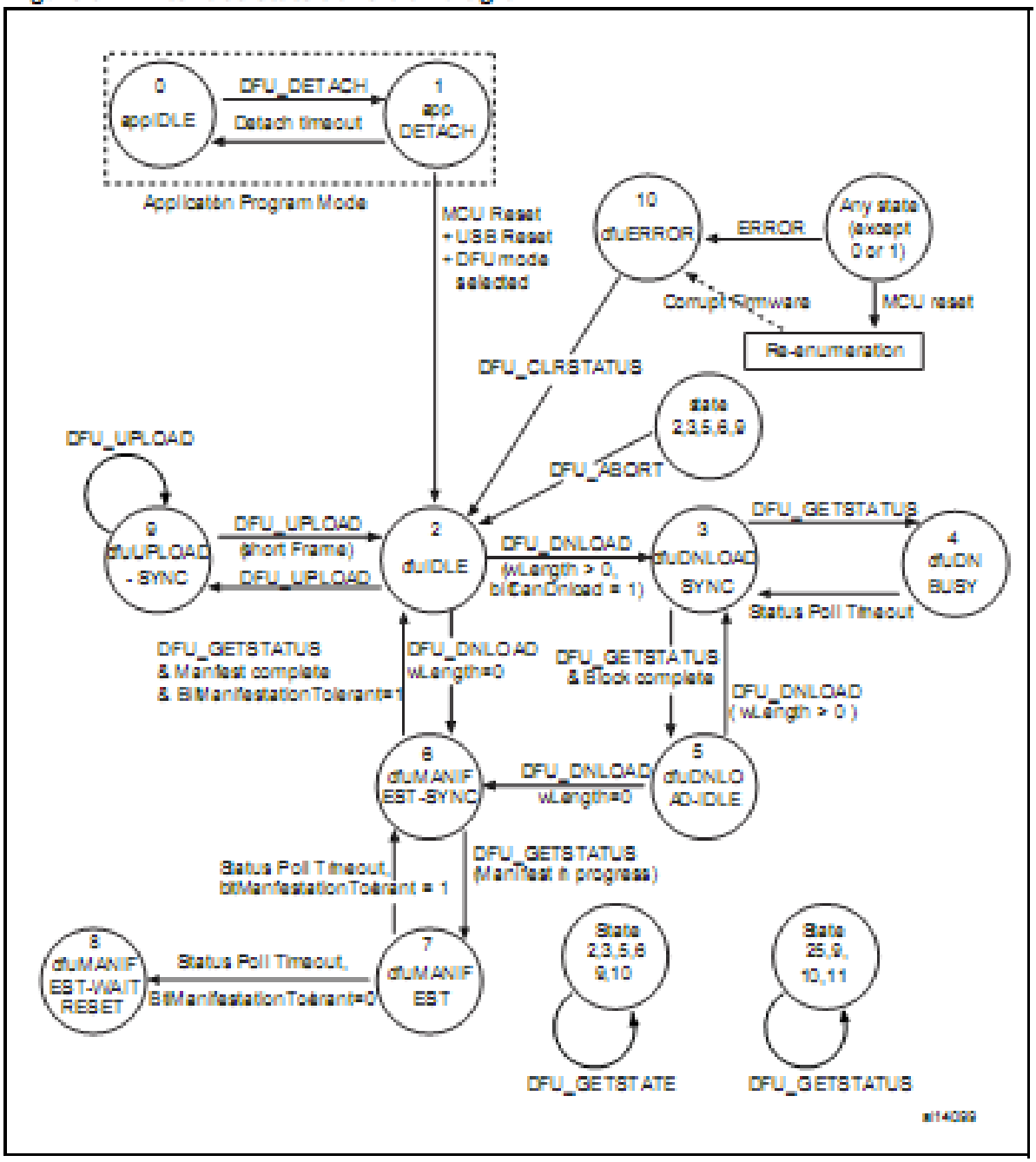
注意：wBlockNum = 1 的 DFU_DNLOAD 和 DFU_UPLOAD 请求都保留给将来 ST 微电子使用。

3.4.3 DFU 状态图

图 3 总结 DFU 接口状态和状态的转换。触发状态转换的事件可以被认为是图灵机概念中的多

重输入陷阱。

Figure 3. Interface state transition diagram



注意：图 3 中的状态转换图和 DFU 类规格说明中的定义基本相同，只是在状态 2 到状态 6 的转换，这个转换是新加的，可能在设备固件中实现，也可以不在设备固件中实现。

3.4.4 下载和上传

主机将固件映像文件分成 N 份，并以控制写操作的方式通过端点 0 发送给设备。

每次控制写操作传输的最大字节数在 DFU 功能描述符中的 wTransferSize field 字段指定。

根据 MCU 设备存储器的映射方式和存储器类型（可读，可擦除，可写或者复合）有几种可能的下载机制。

最常见的机制如下：(包含可读，可写，可擦除的存储器扇区)

- 除了枚举阶段后收集的整个存储器映射和设备性能的数据，主机开始发送 GetCommands 命令以获取更多的设备性能以及 DFU 支持哪些命令。
- 主机使用 DFU_DNLOAD 请求（wBlockNum = 0，wLength = 5）发送擦除带地址扇区命令，这阶段设备擦除主机发送的地址对应的存储器块，擦除操作后。DFU 固件可以将应用信息写入擦除后的块。
- 主机使用 DFU_DNLOAD 请求（wBlockNum = 0 and wLength = 5）发送设置地址指针命令。这个地址指针作为绝对偏移存储在设备的 RAM 中。
- 主机连续的通过 DFU_DNLOAD 请求发送 N 个命令给设备，(DFU 功能描述符中，.wBlockNum 从 2 开始 ，wTransferSize 为设备可接受的最大控制写方式传输的字节数)。

  最后写入存储器的数据将会在地址绝对偏移+ (wBlockNum -2) *wTransferSize + wLength 处（wBlockNum 和 wLength 是上一 DFU_DNLOAD 请求中的参数）。

如果主机想上传用于确认的存储器数据，或者取回和存档设备固件，定义了一个反向的下载：

- 主机使用 DFU_DNLOAD 请求（wBlockNum = 0 and wLength = 5）发送设置地址指针命令。这个地址指针作为绝对偏移存储在设备的 RAM 中。
- 主机连续的通过 DFU_DNLOAD 请求发送 N 个命令给设备，(DFU 功能描述符中，.wBlockNum

从 2 开始 ，如果 bitCanAccelerate = 0，wTransferSize 为设备可接受的最大控制写方式传输的字节数，如果 bitCanAccelerate = 1，wTransferSize 为 4096 字节)。

最后从存储器取回的数据将会在地址绝对偏移+ (wBlockNum -2) *wTransferSize + wLength 处（ wBlockNum 和 wLength 是上一 DFU_DNLOAD 请求中的参数)。

3.4.5 显示阶段

传输阶段完成后，设备已经可以执行新的固件，这在正常运行时操作下由 USB 复位和重枚举完成。

3.5 STM32F10xxx DFU 实现

3.5.1 支持的存储器

STM32F10xxx 的 DFU 实现支持如下存储器：

- **内部 Flash 存储器** :前 12 页为 DFU 保留(只读页)其他的 116 也可以由 DFU 编程(应用区)。
- **外部的串行 Flash 存储器 (M25P64)**: 包括 128 个 64 KB 的扇区。

注意：

1. 为内部存储器创建一个 DFU 映像，在 DFU 文件管理器中选择 Alternate Setting 00。
2. 为外部串行存储器创建一个 DFU 映像 ,在 DFU 文件管理器中选择 Alternate Setting 01 。

3.5.2 DFU 模式进入机制

如果出现下面的情况，STM32F10xxx 在 MCU 复位后进入 DFU 模式：

- DFU 模式由用户强制进入：用户在复位后按 Key 按钮。

在应用区域没有正确的代码可用：在跳转到应用的代码前，DFU 代码测试内部 Flash 存储器的应用区域的第一个地址是不是一个正确的栈顶地址(对 STM32F10xxx, ,第一个应用地址是 0x0800 3000)。这是通过读内存的应用区域开始地址并且验证高半字是否为 0x2000 。(STM32F10xxx RAM 基址)。

3.5.3 STM32F10xxx的可用DFU 映像

STM32F10xxx 开发人员工具包中的可用 DFU 映像有：

- 操纵杆鼠标范例
- 大容量存储范例
- 虚拟 COM 范例
- 扬声器范例

4 大容量存储范例

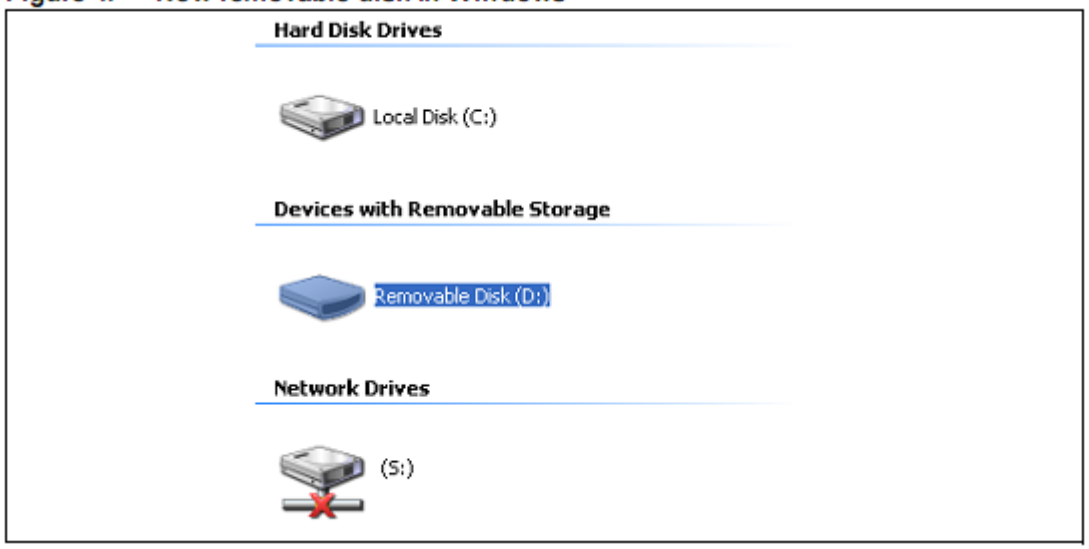
大容量存储范例 ,给出了如何使用 STM32F10xxx USB 外设和 PC 之间通过批量传输方式通信的的典型例子。

本范例支持 BOT (仅限批量传输)协议和所需的 SCSI (小型计算机接口)命令,并和 Windows XP (SP1/SP2) 和 Windows 2000 (SP4).兼容。

4.1 大容量存储范例总论

大容量存储范例遵守 USB2.0 和 USB 大容量存储类(仅限批量传输子类)规格说明。运行应用后，用户需要用 USB 电缆将设备和 PC 连接，之后设备会被检测（无需驱动）(Win 2000 and XP)，一个新的可移除的驱动器出现在系统窗口，写/读/格式化操作可以像其他移动设备一样进行(如图 4)。

Figure 4. New removable disk in Windows



本实现使用了 microSD 卡作为存储器支持，所有用于 microSD 初始化，读，写的相关的固件可以在“msc.c/.h”文件中找到。

4.2 大容量存储协议

4.2.1 仅块传输

仅批量传输协议只使用块管道传输命令，状态，数据（没有中断和控制管道），默认的管道（管道 0，也就是端点 0）只用于清除批量管道的状态（清 STALL 状态）和发送两个类专用请求：Mass Storage 复位 和 Get Max LUN。

命令传输

主机使用叫做命令块包裹(CBW)的特殊格式发送命令 ， CBW 是一个 31 字节的分组，

表 11 显示了 CBW 不同字段。

Table 11. CBW packet fields

	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 (小端)。
- **dCBWTag:** 主机为每一条命令中指定这个字段，设备需要在相应的状态中返回相同的 dCBWTag。
- **dCBWDataTransferLength:**要传输的字节数（主机期望值）。
- **bmCBWFlags:** 表示传输方向，该字段的位定义为：
  - 位 7: 方向位

    0: 主机到设备的传输

    1: 设备到主机的传输

    注意: 如果 dCBWDataTransferLength 字段为 0，则忽略这位。
  - 位 6:0: 保留
- **bCBWLUN:** 逻辑单元号.。
- **bCBWCBLength:** CBWCB 命令的长度（字节数）。
- **CBWCB:**设备要执行的命令块。

状态传输

为了通知主机每个受到的命令的状态，设备使用命令状态包(CSW)。表 12 显示了 CSW 的不同字段。

Table 12. CSW packet fields

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

- dCSWSignature: 53425355 USBS （小端)。
- dCSWTag: 设备把该域置该为相应 CBW 中接收到的 dCBWTag 的值。
- dCSWDataResidue:期望数据（相应 CBW 中的 dCBWDataTransferLength 的值）和接收到数据或者要发送的数据的实际值的不同。
- bCSWStatus: 相应命令的状态 ，该位可以取到表 13 中的三个非保留值。

值	描述
00h	命令通过
01h	命令失败
02h	阶段错误
03h=>FFh	保留

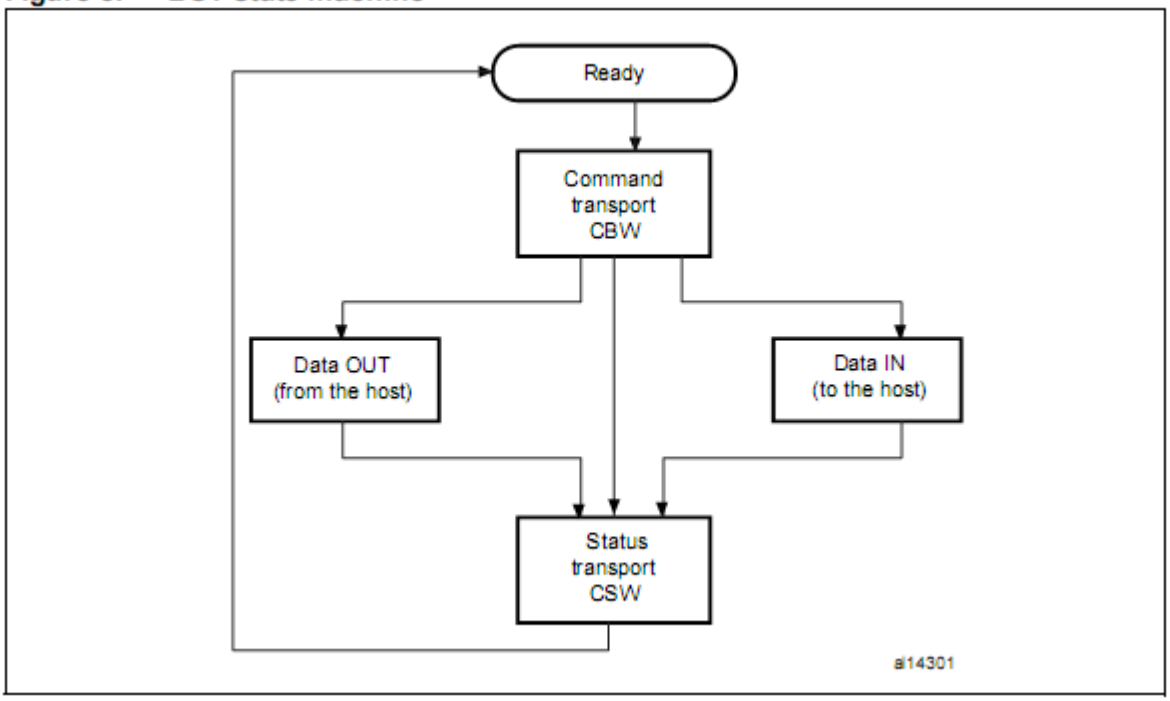
数据传输

数据传输阶段由通信者 CBW 的 dCBWDataTransferLength 和 bmCBWFlags 指定。主机尝试向设备发送确切数目的字节数，或者从设备接收确切数目的字节数。

图 5 显示 BOT 传输的状态机

注意：BOT 协议的更多信息请参考“通用串行总线大容量类仅批量传输”规格说明。

Figure 5. BOT state machine



4.2.2 小型计算机系统接口（SCSI）

SCSI 命令集为 SCSI 设备（例如硬盘，磁带和大容量存储设备）提供了高效的点对点操作。也就是说这保证了 SCSI 设备和 PC 操作系统的通信。

表 14 显示了可移动设备的 SCSI 命令，并没有全部列出，更多信息请参考 SPC 和 RBC 规格说明。

表 14 SCSI 命令集

命令名称	操作码	命令支持(1)	描述	参考
Inquiry	0x12	M	得到设备信息	SPC-2
Read Format Capacities	0x23	M	报告当前媒介支持的媒介能力和	SPC-2

			可格式化的能力	
Mode Sense (6)	0x1A	M	向主机报告参数	SPC-2
Mode Sense (10)		M	向主机报告参数	SPC-2
Prevent\ Allow Medium Removal	0x1E	M	阻止或者允许从可移动媒介设备的移除媒体	SPC-2
Read (10)	0x28	M	从媒介向主机传输二进制数据	RBC
Read Capacity (10)	0x25	M	报告当前媒介能力	RBC
Request Sense	0x03	O	向主机传输状态传感器数据	SPC-2
Start Stop Unit	0x1B	M	使能或者禁止媒体访问操作和控制特定电源情况的逻辑单元	RBC
Test Unit Ready	0x00	M	请求设备报告是否准备好	SPC-2
Verify (10)	0x2F	M	验证媒介上的数	RBC

			据	
Write (10)	0x2A	M	从主机向媒介传输二进制数据	RBC

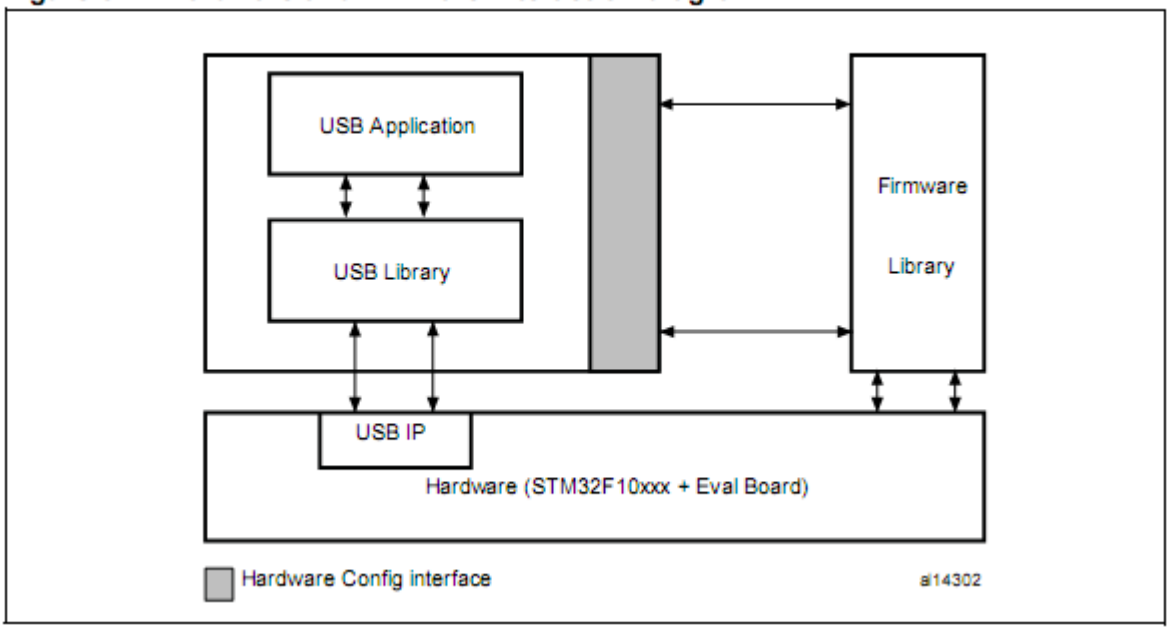
1. 命令支持关键字: M =强制的, O= 可选的

4.3 大容量范例的实现

4.3.1 硬件配置接口

硬件**配置**接口是在 USB 应用(这里指大容量存储器范例)和 STM32F10xxx 微控制器内部/外部硬件之间的一层。STM32F10xxx 微控制器内部/外部硬件由 STM32F10xxx 固件库管理，所以从固件的观点，硬件配置接口在固件库和 USB 应用之间的固件层。图 6 显示了不同固件组件和硬件环境之间的交互。

Figure 6. Hardware and firmware interaction diagram



硬件配置层由 HW_config.c 和 HW_config.h 实现，对大容量范例来说，硬件管理层管理下面的

硬件要求：

- 系统和 USB IP 时钟配置
- 读写 LED 设置
- LED 命令
- 初始化 microSD 卡
- 得到存储介质的特性（块大小和存储器容量）

4.3.2 端点配置和数据管理

本章节描述了传输模式对应的配置和数据流。

端点配置

每次 USB 复位事件之后，都应该进行端点配置，这部分代码在函数 MASS_Reset 函数中实现了(文件 usp_prop.c)。

要配置端点 0，必须作如下配置：

- 将端点 0 设定为默认的控制端点。
- 在 BTABLE 中设定端点 0 的 Rx 和 Tx 计数和缓冲地址(usb_conf.h 文件)。
- 设定端点的 Rx 状态为可用，Tx 状态为 NAK。

批量管道(端点 1 和 2)的配置如下：

- 设定端点 1 为批量传入。
- 在 BTABLE 中设定端点 1 的 Tx 计数和数据缓冲地址(usb_conf.h file)。
- 使端点 1 的 Rx 无效。
- 设定端点 1Tx 状态为 NAK。

- 设定端点 2 为批量输出。
- 设定端点 2 的 Rx 计数和数据缓冲地址(usb_conf.h file)。
- 使端点 2 的 Tx 无效。
- 设定端点 2 的 Rx 状态可用。

数据管理

数据管理由从 PMA 特定数据缓冲地址中取所需数据的传输组成，缓冲区地址由相关的端点决定(输入: ENDP1TXADDR ,输出: ENDP2RXADDR)。这些传输使用如下两个函数完成(usb_mem.c file)：

- **PMAToUserBufferCopy ():** 本函数从包存储区域中传输特定树木的字节到内部 PAM 中。这个函数用于把主机发送的数据拷贝到设备。
- **UserToPMABufferCopy ():** 本函数从内部 PAM 中传输特定字节的数据到包存储区域中。这个函数用于把来自设备的数据发送到主机。

4.3.3 类细节请求

大容量存储类规格说明描述了两种类专用请求：

仅批量传输的大容量存储复位

这个请求用于复位大容量存储设备和相应的接口。这个请求使设备为下一次 PC 发送的 CBW 做好准备。

为了发出仅批量传输大容量存储复位，主机在默认管道（端点 0）发出设备请求：

- bmRequestType: 类,主机设备间接口

- bRequest 设置为 0xFF
- wValue 设置为 0
- wIndex 设置为接口号 (本实现中为 0)
- wLength 设置为 0

这个请求作为非数据类专用请求在 MASS_NoData_Setup()函数中实现(usb_prop.c file)。

在接到这个请求后，设备清除两个批量端点的数据翻转，初始化 CBW 签名为默认值并设置 BOT 状态机为 BOT_IDLE 状态，为接收下一个 CBW 做好准备。

GET MAX LUN 请求

一个大容量存储设备可能实现多个逻辑单元，多个逻辑单元之间共享通用设备特性。主机使用 bCBWLUN 指派设备的哪个逻辑单元是 CBW 的目的地。

Get Max LUN 设备请求用于决定设备支持的逻辑单元数目。

为了发出 Get Max LUN 请求，主机必须在默认管道（端点 0）发出设备请求：

- bmRequestType:类 主机和设备之间的接口
- bRequest f 设置为 0xFE
- wValue 设置为 0
- wIndex 设置为接口号 (本实现中为 0)
- wLength 设置为 1

本请求作为数据类专用请求在 MASS_Data_Setup()函数中实现(usb_prop.c 文件)。

注意：在这两个实现中，只有一个 LUN，所以 Get Max LUN 直接返回 0x00，为了实现其它的 LUN，用户需要返回已实现的 LUN 的数目作为请求的回应。

4.3.4 标准请求规需求

为了遵守 BOT 规范，设备必须在收到相同的标准请求后对两种规格说明作出回应：

- 当设备从未配置状态转向配置状态时，所有端点的数据翻转必须清除，需求由 Mass_Storage_SetConfiguration()函数实现(usb_prop.c 文件) 。
- 当主机发送 CBW 命令并带有有效的签名或者长度，设备必须保证端点 1 和 2 都在 STALL 状态，直到收到大容量存储设置类专用请求。这个功能由 Mass_Storage_ClearFeature()函数管理（usb_prop.c 文件）

4.3.5 BOT 状态机

为了提供 BOT 协议，实现了一个带有 5 个状态的状态机，状态描述如下：

- **BOT_IDLE:** USB 复位后， 仅批量传输大容量存储复位或者在 CSW 发出后的默认状态。在这个状态下，设备做好接收来自主机的下一个 CBW 的准备。
- **BOT_DATA_OUT:**在设备从主机收到带有数据流的 CBW 后进入这个状态。
- **BOT_DATA_IN:** 在设备向主机发送带有数据流的 CBW 后进入这个状。
- **BOT_DATA_IN_LAST:** 当发送最后一个主机所需数据时，进入这个状态。
- **BOT_CSW_SEND:** 当设备发送 CSW 时进入此状态，当设备在这个状态中并产生一个输入传输信号时，设备进入 BOT_IDLE 状态，并准备接受下一个 CBW。
- **BOT_ERROR:** 错误状态。

BOT 状态机由如下函数管理(usb_bot.c 和 usb_bot.h 固件文件):

- **Mass_Storage_In (); Mass_Storage_Out ():** 这两个函数在正确传输(IN 或者 OUT)发生时调用。这两个函数的目的是为了提供接收/发送 CBW，数据或者 CSW 的下一步操作。

  CBW_Decode (): 用于解码 CBW 和将固件分派为相应的 SCSI 命令。

- **DataInTransfer ():**用于传输特有的设备数据到主机。

- **Set_CSW ():** 用于根据命令的执行用所需的参数设置 CSW 字段。

- **Bot_Abort ():** 用于根据 BOT 流中发生的错误设置端点 1 或 2（或者都设置）为 STALL。

4.3.6 SCSI 协议实现

SCSI 协议的目的是为 PC 主机上的操作系统所需的所有的 SCSI 命令提供正确的反应，本章节详细的介绍了所有实现的 SCSI 命令的管理方法：

- **INQUIRY** 命令(操作码 = 0x12):

  跟据命令的 ALLOCATION LENGTH 字段发送所需的查询页数据（范例中只支持页 0 和标准页）和所需的数据长度。

- **SCSI READ FORMAT CAPACITIES** 命令(操作码 = 0x23):

  发送 Read Format Capacity 数据响应（SCSI_data.c file 文件中的 ReadFormatCapacity_Data[ ]）。

- **SCSI READ CAPACITY (10)** 命令(操作码= 0x25):

  发送 Read Capacity（10）数据响应（SCSI_data.c 文件中的 ReadCapacity10_Data[ ]）。

- **SCSI MODE SENSE (6)** 命令(操作码= 0x1A):

  发送 Mode Sense (6)数据响应（SCSI_data.c file 文件中的 Mode_Sense6_data[ ]）

- **SCSI MODE SENSE (10)** 命令(操作码= 0x5A):

发送 Mode Sense (10) 数据响应 (SCSI_data.c file 文件中的 Mode_Sense10_data[ ]).

- **SCSI REQUEST SENSE** 命令(操作码= 0x03):

  发送 Request Sense 数据响应，注意 Resquest_Sense_Data [ ]数组 (SCSI_data.c file) 使用 Set_Scsi_Sense_Data() 函数更新，这是为了能根据在传输中出现的任何错误来设置 Sense key 和 ASC 字段。

- **SCSI TEST UNIT READY** 命令(操作码= 0x00):

  通常返回 带有 COMMAND PASSED 状态的 CSW。

- **SCSI PREVENT\ALLOW MEDIUM REMOVAL** command (操作码= 0x1E):

  通常返回 带有 COMMAND PASSED 状态的 CSW。

- **SCSI START STOP UNIT** 命令(操作码= 0x1B):

  本命令有主机发送（当用户在窗口中右击）并选择 Eject 操作，这时，固件用 Stor_Data_In_Flash()函数编程内存中的数据。

- **SCSI READ 10** 命令(操作码= 0x28) 和 **SCSI WRITE** 10 命令(操作码= 0x2A):

  主机发出这两个命令来进行读写操作。这时设备需要检查地址是否合适以及 bmFlag 命令的方向位，如果通过验证，固件从 microSD 卡发出读写操作。

- **SCSI VERIFY 10** 命令 **(**操作码 =0x2F):

  The SCSI VERIFY 10 命令要求设备检查媒介中写入的数据，这时没有类 Flash 存储器支持，当接收到 SCSI VERIFY 命令 ,设备检查 BLKVFY 位 ,若果 BLKVFY 位为 1 ,CSW 返回命令通过。

4.3.7 存储器管理

所有的存储器管理函数都在 memory.c 和 memory.h.文件中。存储器管理由两个过程组成：

- SCSI READ (10) 和 SCSI WRITE (10) 命令地址范围管理和验证：这个过程由 Address_Management_Test()函数完成。该函数的作用是从 microSD 卡中抽取实际地址和存储

器偏移并测试当前的传输（读或写）在存储器的范围之内。如果超出范围，函数会 STALL 端点 0 或者两个端点（根据传输是读还是写）并且返回一个错误状态来中止传输。

- 读写过程的管理：由 Read_Memory() 和 Write_Memory()函数完成，这两个函数管理基于 thmsc.c 文件的 MSD_WriteBlock 和 MSD_ReadBlock 函数 microSD 卡访问。每次访问后，用先前传输长度更新当前存储器偏移和下次访问地址。

4.4 如何定制大容量存储范例

实现的固件是于演示 STM32F10xxxUSB 外设在批量传输性能的一个简单用例，也可以根据用户的需要定制。定制可以在实现大容量存储协议的三个层次上进行。

- BOT 层定制: 用户可以实现自己的 BOT 状态机或者修改已实现的（只修改 usb_BOT.c 和 usb_BOT.h 文件，保留数据传输方法）。
- SCSI 层定制: SCSI 协议的实现，比 Section4.3.6 中列举的所支持的命令多 ，SCSI 协议实现了一些未支持的命令。当主机发送这些命令时，CBW_Decode()函数像命令一样调用相应的函数，所有关于未支持的命令关联的函数定义在 SCSI_Invalid_Cmd() 函数 (参照 usb_scsi.c 文件)。SCSI_Invalid_Cmd() 函数将两个端点 (1 和 2)置为 STALL, 设置判断数据为无效命令关键值并且发送 CSW （带有命令失败状态.），为了支持无效命令，用户注释掉相应的行并且实现自己的过程。例如为了支持 SCSI_FormatUnit 命令，将下列行注释掉：

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

在 usb_scsi.c 文件中实现进程

```
void SCSI_Invalid_Cmd (void)
{
```

// 你的实现

}

通过这种方式定制的函数由 CBW_Decode()函数(usb_BOT.c 文件)自动调用，如果用户想实现一个先前没有列出的命令，需要修改 CBW_Decode()函数并实现加入新命令的协议。

大容量存储描述符

Table 15.设备描述**符**

字段	值	描述
bLength	0x12	描述符大小（字节）
bDescriptortype	0x01	描述符类型
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 的最大分组:64 字节
idVendor	0x0483	卖主标识符(ST 微电子)
idProduct	0x5720	产品标识符
bcdDevice	0x0100	设备发行号: 1.0
iManufacturer	4	生产商字符串描述符索引: 4
iProduct	42	产品字符串描述符索引: 42
iSerialNumber	96	线性编号字符串描述符索引

bNumConfigurations	0x01	可能的配置数目: 1

表 16 设置描述符

字段	值	描述
bLength	0x09	描述符大小
bDescriptortype	0x02	描述符类型
wTotalLength	32	描述符（包括接口端点描述符)返回的数据长度(字节数)
bNumInterfaces	0x0001	配置支持的接口数(只有一个接口)
bConfigurationValue	0x01	配置值
iConfiguration	0x00	配置字符串描述符索引
bmAttributes	0x80	配置特点：总线供电
Maxpower	0x32	USB 总线最大电能消耗:100mA

表 17 接口描述符

字段	值	描述
bLength	0x09	描述符大小
bDescriptortype	0x04	描述符类型
bInterfaceNumber	0x00	接口数
bAlternateSetting	0x00	备用设置编号
bNumEndpoints	0x02	使用的端点数:2

bInterfaceClass	0x08	接口类：大容量存储类
bInterfaceSubClass	0x06	接口子类：SCSI 透明
bInterfaceProtocl	0x50	接口协议: 0x50
iInterface	106	接口字符串描述符索引

表 18

字段	值	描述
输入端点		
bLength	0x07	描述符大小
bDescriptortype	0x05	描述符类型
bEndpointAddress	0x81	输入端点地址 1
bmAttributes	0x02	批量传输端点
wMaxPacketSize	0x40	64 字节
bInterval	0x00	批量传输端点不支持
输出端点		
bLength	0x07	描述符大小
bDescriptortype	0x05	描述符类型
bEndpointAddress	0x02	输出端点地址 2
bmAttributes	0x02	批量传输端点
wMaxPacketSize	0x40	64 字节
bInterval	0x00	批量传输端点不支持

5 虚拟COM端口范例

在现代 PC 中，USB 是很多外设的标准通信端口，然而很多工业软件应用仍用经典的 COM 端口（UART）。虚拟 COM 端口范例使用了一个很简单的方案解决了这个问题。通过改变为 COM 端口设计的 PC 应用程序来像使用 COM 端口一样使用 USB 端口。

本范例提供了 STM32F10xxx 固件用例和 PC 驱动用例。本章节对范例的实现和使用作简要的介绍。

5.1 虚拟COM端口范例建议

这个范例建议将 STM32F10xxx-128K-评估板作为 USB—USART 桥，并在 PC 和笔记本电脑（不通过 RS232 端口）之间通信（如图 7 所示）。通信中使用的 PC 应用是 Windows 超级终端，如图 8。

Figure 7. Virtual COM Port demo as USB-to-USART bridge.

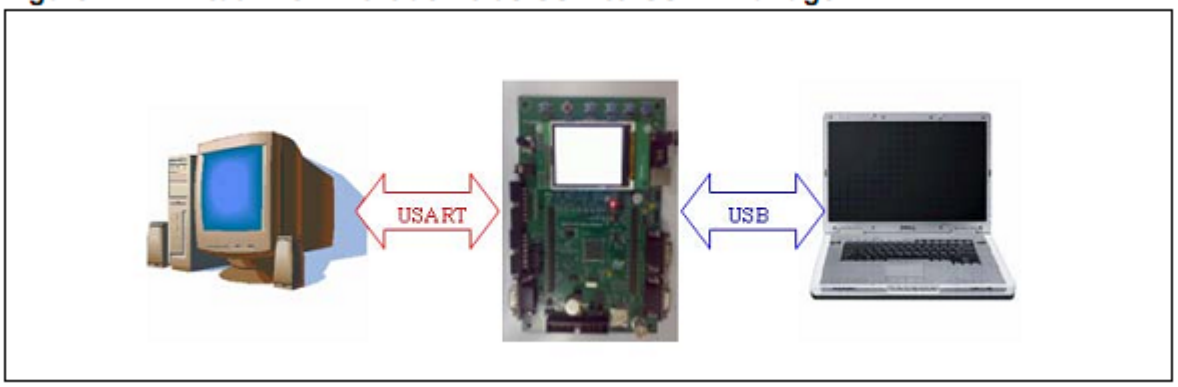
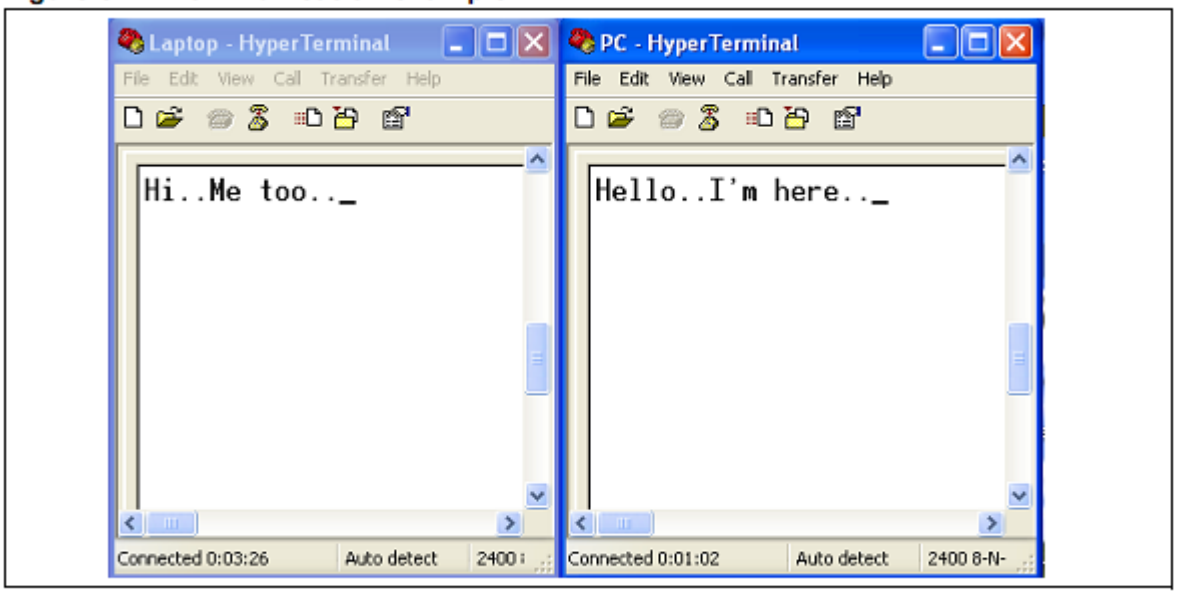


Figure 8. Communication example



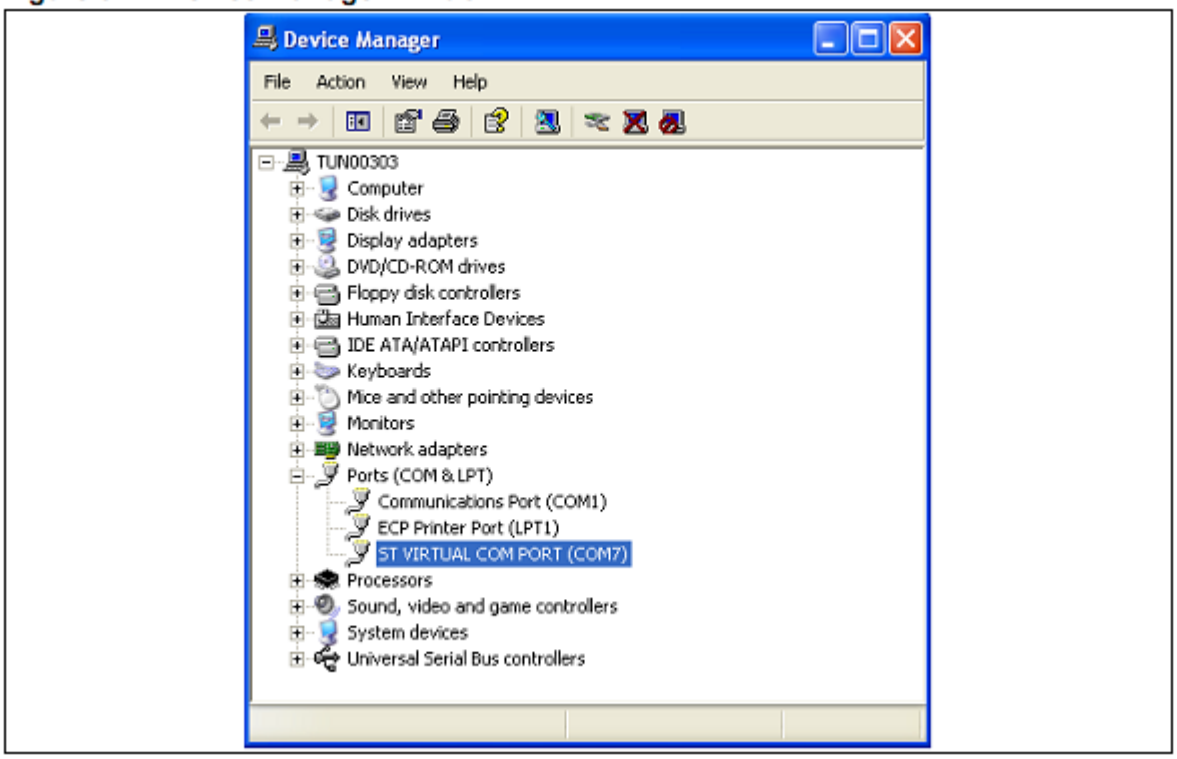
5.2 软件驱动安装

按照下面的步骤安装虚拟 COM 端口驱动：

- 在评估板上加载并运行应用
- 用 USB 电缆连接 PC
- 安装的时候在 PC 上选择好 stmcdc.inf 文件的位置(工具包里面提供)

安装完成后，在设备管理器窗口中出现一个新的 COM 端口，如图 9 所示。

Figure 9. Device Manager window



5.3 实现

5.3.1 硬件实现

虚拟 COM 端口范例使用 STM32F10xxx-128K-评估板上的 UART1 端口。运行范例不需要外部的硬件。

5.3.2 固件实现

为了被当作 COM 端口，根据通信设备类(CDC)规范，USB 设备需要实现两个接口：

- 抽象控制模型通信：一个输入中断端点。在描述符中申明了但是相关的端点未使用(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，在范例中用端点 1 和端点 3 代表，端点 1 用于向 PC 的真实 USB 发送从 USART 收到的数据。端点 3 用于接收来自 PC 的数据并通过 UART 发送。

CDC类的更多信息,请参阅通信设备的通用串行总线类定义规范,可以从 www.usb.org获取。

类专用请求

为了实现虚拟 COM 端口，设备需要支持如下的类专用请求：

- **SET_CONTROL_LINE_STATE :** RS-232 信号用于告诉设备出现了数据终端设备，这个请求通常在 Virtual_Com_Port_NoData_Setup()函数(usb_porp.c 文件)中返回 USB_SUCCESS 状态。
- **SET_COMM_FEATURE** ：控制某个特定的通信特性的设置，请求通常在 Virtual_Com_Port_NoData_Setup()函数(usb_porp.c 文件)中返回 USB_SUCCESS 状态。
- **SET_LINE_CODING :** 发送设备配置。包括波特率，停止位，字符位数。收到的数据保存在特殊的数据结构（叫做 linecoding）中并用于更新 UART 0 参数。
- **GET_LINE_CODING:** 本命令请求设备当前的波特率，停止位，奇偶和字符位数。设备返回保存在线性编码结构中的数据。

硬件配置接口

虚拟 COM 端口中的硬件**配置**接口(hw_config.c and .h 文件)在管理如下流程：

- **配**置系统和 IPs(USB&USART1)的时钟和中断。
- 将 USART1 初始化为默认值。
- 用 SET_LINE_CODING 请求收到的参数**配**置 USART1。
- 通过 PC 向 PC 发送 USART 1 接收到的数据。
- 通过 USART 1 发送从 USB 接收到的数据。

注意：STM32F10xxx 支持 7 位或 8 位数据格式（在高速终端端），并且带宽为 1200 到 115200。

6 USB音频范例

USB 音频范例为在同步模式下使用 STM32F10xxx USB 外设和 PC 主机进行通信提供了实例。提供了正确的同步端点设置方法，从主机接收或者向主机发送数据的实例，以及如何在实时应用中使用数据。

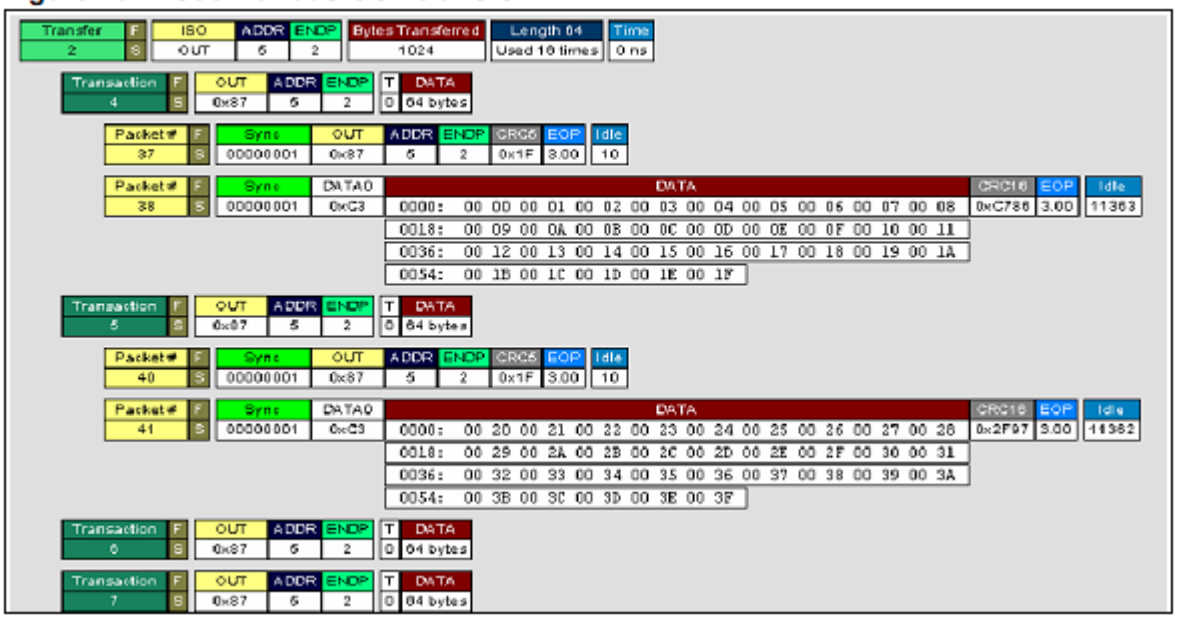
这份用户指南中描述的可用的音频范例是 USB 扬声器。

6.1 同步传输综述

同步传输用于需要保证 USB 带宽的访问只能由有限的延迟，稳定的传输率以及出错的时候不重新开始数据传输操作的时候。

实际上同步传输方式并没有使用握手并且数据包之后不需要等待 ACK 也不需要发送 ACK。图 10 给出了一个 64 字节的同步输出的例子。

Figure 10. Isochronous OUT transfer



典型的使用同步方式的例子是音频采样，压缩的视频流以及所有对传输率有严格的精确要求采样

数据。

USB2.0 同步传输模式的更多详细信息请参阅 USB2.0 规范。

6.2 音频设备类综述

如通用串行总线类定义的音频设备规范一样，音频设备是用于操作音频，声音和声音相关功能的单个设备或者嵌入在合成设备的中的功能部件，包括两种音频数据（数字或者模拟）和直接控制音频环境（例如音量和音调控制）的功能。

从 USB 的观点看，所有的音频设备在音频接口类中都是成组的，这个类分为几个子类，音频设备的通用串行总线类定义规范详细列出了下面 3 种子类：

- **音频控制接口子类（AC):**每个音频功能有一个唯一的音频控制端口。AC 接口用于控制特定功能的功能行为。为了实现这些功能，接口可以使用如下的端点：
  - 一个控制端点(端点 0),用于使用单元和终端设置 ,以及用类专用请求得到音频功能的状态。
  - 一个中断端点，用于状态返回（可选）。

  音频控制端口是访问音频功能内部的唯一入口点。所有的关于音频功能单元或者终端的特定音频控件的操作必须指向饮品功能的音频控制接口。同样，所有的关于音频功能内部的描述符都是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能支持多重备用设置。音频控制接口的备用设置可以通过为每个备用设置设定不同的类专用音频控制接口描述符来实现支持多重拓扑的音频功能。为每个可选设置使用不同的类细节音频控制接口描述符。
- **音频流接口子类（AS):** 音频流接口用于在主机和音频功能之间内部交换数字音频数据流，这是可选的。一个音频功能可以有 0 个或者多个相关的音频流接口，每个接口可以使用不用的格式传

输数据或者传输自然数据。每个音频流接口最多可以有一个同步数据端点。

- **MIDI 流接口子类(MIDIS)**：MIDI 流接口用于向音频功能输入 MIDI 数据流或者从饮品功能输出数据流。

为了操纵音频函数的物理属性，在功能上分为两个可寻址实体，两种类型的实体可以识别，分别叫做单元和终端。音频设备的通用串行总线类定义规范定义了 7 种标准的单元和终端，这些足够实现大部分音频功能。他们分别是：

– 输入终端

– 输出终端

– 混合单元

– 选择器单元

– 特性单元

– 处理单元

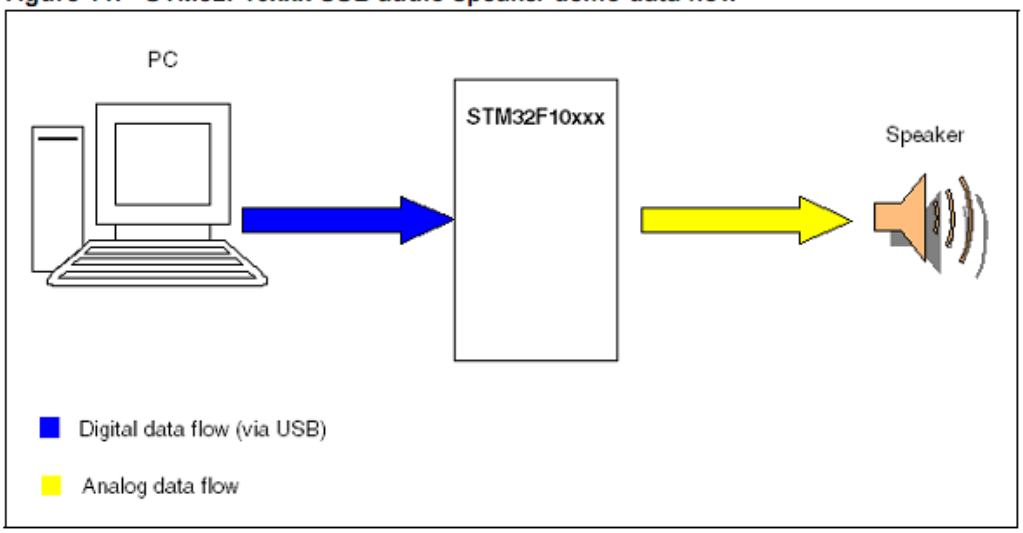
– 扩展单元

音频类特性和需求的更多信息请参阅音频设备的通用串行总线类定义规范（usb.org 网站可以获取）。

6.3 STM32FF10xxx USB扬声器范例

USB 扬声器范例是使用 USB 接口从 PC 主机接收音频数据流并可以通过 STM32F10xxx MCU 回放。图 11：STM32F10xxx USB 扬声器范例数据流体现了 PC 主机和扬声器之间的数据流向。

Figure 11. STM32F10xxx USB audio speaker demo data flow

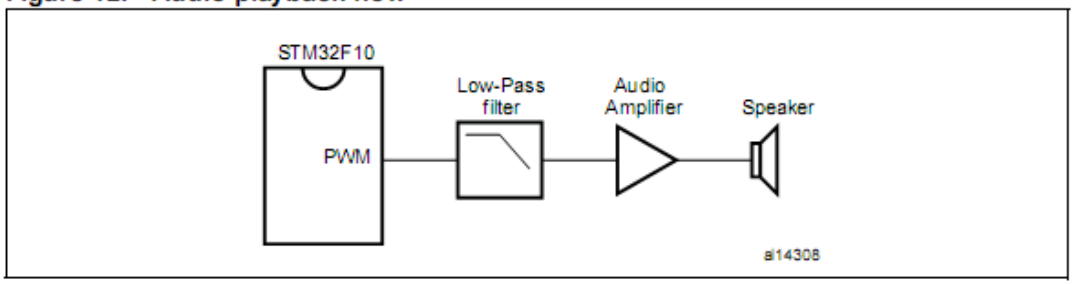


6.3.1 通用功能

- USB 特点：
  - 端点 0(IN):用来列举设备以及应答类专用的请求。这个端点的包最大值是 64 字节。
  - 端点 1（OUT）: 用来从一个 PC 主机接收音频数据流，每个包最大为 22 字节。
- 音频特点：
  - 音频数据格式：Type I / PCM8 格式 / Mono
  - 音频数据解析度：8 位
  - 采样频率：22kHz
- 硬件要求：

由于 STM32F10xxx MCU 没有片上 DAC 来产生模拟数据流，那么将使用一个替换方案来实现 1 通道 DAC，这个方案包含了使用内建脉宽调制（PWM）模块来产生一个信号，该信号的脉宽和采样数据的振幅成比例。PWM 的输出信号靠一个低通滤波器被合成，滤波器可以去除高频部分，只留下低频部分。这个低通滤波器的输出为原始模拟信号提供了一个合理的重建。图 12 显示了使用内建 PWM 的音频回放数据流图。

Figure 12. Audio playback flow



6.3.2 实现

这一节描述了使用 STM32F10xxx 微控制器实现一个 USB 扬声器的硬件和软件实现方案。

硬件实现

为了实现 PWM，需要使用下面两个 STM32F10xxx 内建定时器：

- 设定为输出比较时序模式的 TIM2，用做系统定时器。
- 设定为 PWM 模式的 TIM4。

固件实现

STM32F10xxx 扬声器范例的目的把从主机接收到的数据（音频流）存储到称为 Stream_Buffer 的特定缓存中并且使用 PWM 在每 45.45us（大约 22kHz）播放一段音频流（8 位格式）。

a） 硬件配置接口：

硬件配置接口是在 USB 应用（在这里就是 USB 音频扬声器）和 STM32F10xxx 微控制器的内部/外部硬件之间的一个层。这个内部和外部的硬件是通过 STM32F10xxx 固件库来进行管理的，所以从固件角度来看，该硬件配置接口是 USB 应用和固件库之间的一个固件层。图 13 显示了不同固件和硬件环境的交互。

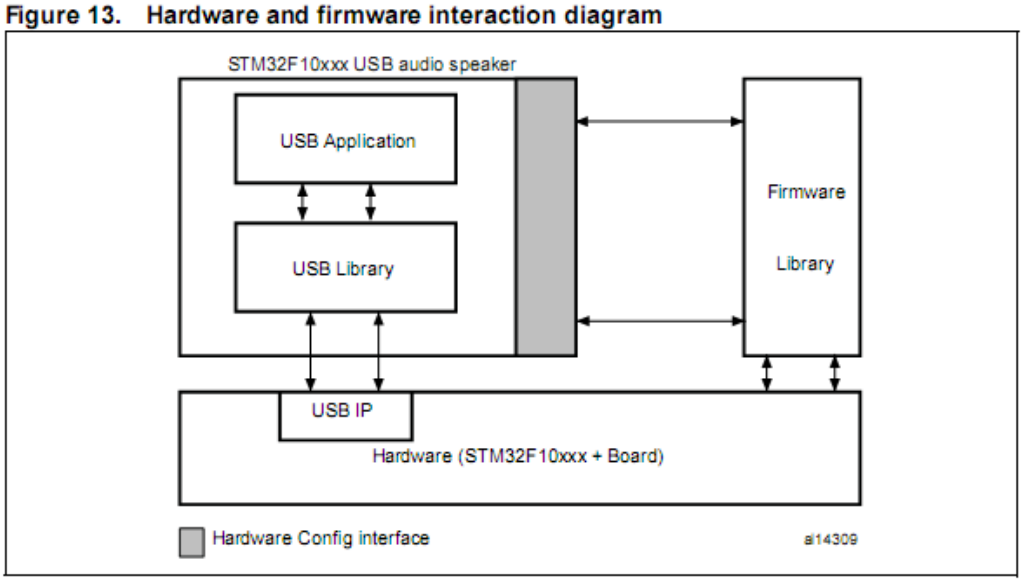
硬件配置层通过两个文件：hw_config.c 和 hw_config.h 来实现。对于 USB 扬声器范例来说，硬

件管理层管理以下硬件需求：

– 系统的和 USB IP 时钟配置

– 定时器配置

Figure 13. Hardware and firmware interaction diagram



b ）端点配置：

在STM32F10xxxUSB扬声器范例中，两个端点被用来与PC主机进行通信：端点0和端点1。注意端点1是一个同步输出端点，而且这种端点是通过STM32F10xxx USB IP来进行管理，使用的是双缓存模式，所以这个固件必须在包存储区域（Packet Memory Area）内为这个端点提供两个的数据缓存。下面的C代码描述了配置一个同步输出端点的方法(见usb_prop.c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 ） 类专用请求

它的实现只是为了支持静音控制。这个特性通过 Mute_command 函数（usb_prop.c 文件）来管理。

d） 同步数据传输管理

按照前面的描述，STM32F10xxx 使用双缓存模式管理同步数据传输，所以为了从 PMA 中把接收到的数据复制到Stream_Buffer ,需要管理两个PMA缓存(ENDP1_BUF0Addr和ENDP1_BUF1Addr）之间的数据交换。PMA 的数据交换是根据 USB IP 和固件库之间的缓存使用情况来管理的。这个操作通过 EP1_OUT_Callback()函数（ usb_endp.c 文件）来实现。复制过程结束后，一个叫做 IN_Data_Offset 的全局变量将更新为被接收并且被复制到 Stream_Buffer 中的字节数。

e） 音频播放实现：

为了回放从主设备接收的音频采样 ,定时器 TIM4 将被编程来产生一个 125.5kHz 的 PWM 信号，TIM2 被编程来产生一个频率等于 22kHz 的中断。在每个 TIM2 中断上 ,一个音频流被用来更新 PWM 的脉冲。一个全局变量（Out_Data_Offset）被用来指出流缓存中的下一个将被播放的流。

注意： 注意“IN_Data_Offset” 和 “Out_Data_Offset” 在每个桢中断（见 usb_istr.c 文件，SOF_Callback()函数）起始时都被初始化为 0，这样来避免“Stram_Buffer” 溢出。

音频扬声器描述符

表 19 设备描述符

域	值	描述
bLength	0x12	描述符大小（字节）
bDescriptortype	0x01	描述符类型（设备描述符）
bcdUSB	0x0200	USB 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 的最大包大小：64 字节
idVendor	0x0483	卖主标识符（STMicroelectronics）
idProduct	0x5730	产品标识符
bcdDevice	0x0100	设备版本号：1.00
iManufacturer	0x01	制造商描述符索引：1
iProduct	0x02	产品描述符索引：2
iSerialNumber	0x03	系列号描述符索引：3
bNumConfigurations	0x01	可能的配置数：1

表 20　配置描述符

域	值	描述
bLength	0x09	描述符大小（字节）
bDescriptortype	0x02	描述符类型（配置描述符）
wTotalLength	0x6D	标识符（包括接口端点描述福）返回的数据总长度（字节）
bNumInterfaces	0x0002	该配置支持的接口数（两个接口）

bConfigurationValue	0x01	配置值
iConfiguration	0x00	配置描述符索引
bmAtrributes	0x80	配置特点：总线供电
Maxpower	0x32	通过 USB 总线的最大电源消耗量：100mA

表 21　　接口描述符

域	值	描述
USB 扬声器标准接口 AC 描述符（接口 0，备用设置 0）		
bLength	0x09	描述符大小（字节）
bDescriptortype	0x04	描述符类型：接口描述符
bInterfaceNumber	0x00	接口号
bAlternateSetting	0x00	备用设置号
bNumEndpoints	0x00	已使用端点号：0（只有端点 0 在该接口中使用）
bInterfaceClass	0x01	接口类：USB DEVICE CLASS AUDIO
bInterfaceSubClass	0x01	接口子类：AUDIO SUBCLASS AUDIOCONTROL
bInterfaceProtocol	0x00	借口协议：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	描述符大小（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	流接口号：1
baInterfaceNr	0x01	baInterfaceNr：1
USB 扬声器输入终端描述符		
bLength	0x0C	描述符大小（12 字节）
bDescriptortype	0x24	描述符类型：AUDIO INTERFACE DESCRIPTOR TYPE
bDescriptorSubtype	0x02	描述符子类型：AUDIO CONTROL INPUT TERMINAL
bTerminalID	0x01	终端 ID：1
wTerminalType	0x0101	终端类型：AUDIO TERMINAL USB STREAMING
bAssocTerminal	0x00	无关联
bNrrChannels	0x01	一个通道
wChannelConfig	0x0000	通道配置：单向
iChannelNames	0x00	未使用
iTerminal	0x00	未使用

USB 扬声器音频特性单元描述符		
bLength	0x09	描述符大小（字节）
bDescriptortype	0x24	描述符类型：AUDIO INTERFACE DESCRIPTOR TYPE
bDescriptorSubtype	0x06	描述符子类型：AUDIO CONTROL FEATURE UNIT
bUnitID	0x02	单元 ID：2
bSourceID	0x01	源 ID：1
bControlSize	0x01	控制大小：1
bmaControls	0x0001	只支持静音控制
iTerminal	0x00	未使用
USB 扬声器输出终端描述符		
bLength	0x09	描述符大小（字节）
bDescriptortype	0x24	描述符类型：AUDIO INTERFACE DESCRIPTOR TYPE
bDescriptorSubtype	0x03	描述符子类型：AUDIO 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	源 ID：2
iTerminal	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，备用设置 0）		

bLength	0x09	描述符大小（字节）
bDescriptortype	0x24	描述符类型：AUDIO INTERFACE DESCRIPTOR TYPE
bInterfaceNumber	0x01	接口号：1
bAlternateSetting	0x00	备用设置：0
bNumEndpoints	0x00	未使用（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	未使用
USB 扬声器标准 AS 接口描述符 – 音频流操作 （接口 1，备用设置 1）		
bLength	0x09	描述符大小（字节）
bDescriptortype	0x24	描述符类型：AUDIO INTERFACE DESCRIPTOR TYPE
bInterfaceNumber	0x01	接口号：1
bAlternateSetting	0x01	备用设置：1

bNumEndpoints	0x01	一个端点
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	未使用
USB 扬声器音频类型 1 格式接口描述符		
bLength	0x0B	描述符大小（字节）
bDescriptortype	0x24	描述符类型：AUDIO INTERFACE DESCRIPTOR TYPE
bDescriptorSubtype	0x03	描述符子类型：AUDIO STREAMING FORMAT TYPE
bFormatType	0x01	格式类型：类型 1
bNrChannels	0x01	通道数：一个通道
bSubFrameSize	0x01	子帧大小：每个音频子帧一字节
bBitResolution	0x08	位精度：每个采样 8 位
bSamFreqType	0x01	只支持一个采样频率
tSamFreq	0x0055F0	22kHz

表 22　端点描述符

域	值	描述

bLength	0x07	描述符大小（字节）
bDescriptortype	0x05	描述符类型：(端点描述符)
bEndpointAddress	0x01	输出端点地址 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	描述符大小（字节）
bDescriptortype	0x25	描述符类型：AUDIO ENDPOINT DESCRIPTOR TYPE
bDescriptor	0x01	AUDIO ENDPOINT GENERAL
bmAttributes	0x80	bmAttributes:0x80
bLockDelayUnits	0x00	未使用
wLockDelay	0x0000	未使用

7 修改历史

表 23 文档修改历史

日期	**版本**	**变化**
2007-05-28	1	第一次发布

8 版权声明：