GigaDevice Semiconductor Inc.

ARM® Cortex™ 32-bit MCU

USBFS/HS 固件库用户指南

1.0 版本

(2019 年 4 月)

目录

图索引

表索引

1. 通用串行总线全速/高速接口（USBFS/USBHS）

1.1. 概述

本文基于 GD32 MCU 通用串行总线全速接口、高速接口的结构，介绍 USBFS/USBHS 模块的工作原理、固件库架构，简要描述了固件库函数的功能、主机状态机和 USB 中断的工作原理，以 MSC 主机和 HID 设备为例，概括地说明 USB 主机和设备的实现过程。

1.2. USBFS/USBHS 原理简介

目前，GD32F105/107/205/207/305/307/350/4xx/E103 系列 MCU 具有 USBFS 接口模块，且这些系列中 USBFS 接口模块的架构及原理相同，在本指南中将为读者以 GD32F4xx 系列为例介绍 USBFS 接口模块原理。另外，在 GD32F4xx 系列 MCU 中，也包括高速 USBHS 接口模块，本指南将一并介绍。

USBFS 接口模块可以作为仅支持全速（FS，12Mbps）传输的的设备模式运行，也可以作为支持全速（FS，12Mbps） 与低速（LS，1.5Mbps）传输的主机模式运行。USBFS 控制器实际是一个双角色设备（DRD），它在点到点通信中既可以作为主机，也可以作为设备。在 OTG 模式下，主机和从机的切换需遵从 OTG 标准的会话请求协议（SRP）和主机协商协议（HNP）。

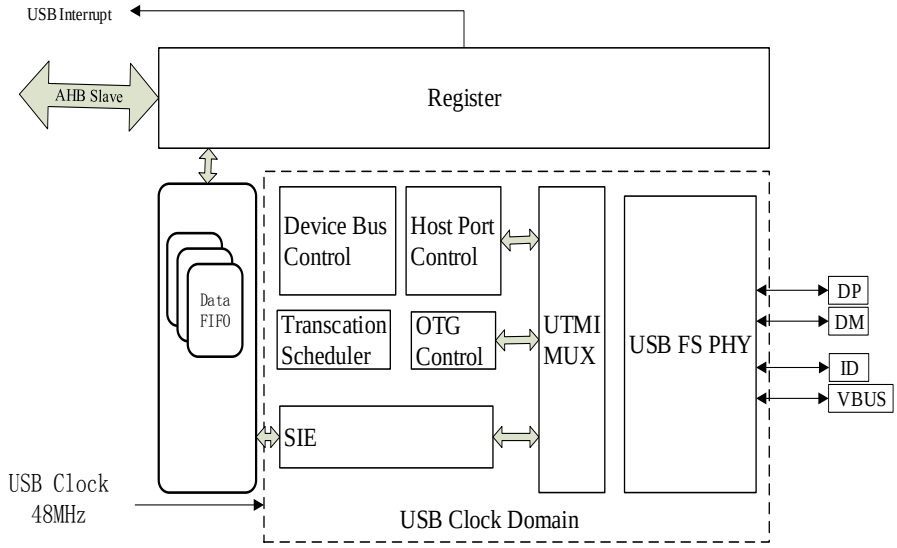
USBHS 接口模块具有 USBFS 接口模块所有的特性及功能，并且在此基础之上，USBHS 接口模块可支持 USB2.0 高速(480Mbps)主机或设备。

1.2.1. USBFS 原理

USBFS 结构

如*图 1-1. USBFS 结构框图*所示，其中，Cortex-M 内核通过 AHB 从机总线对 USBFS 接口模块寄存器进行读写操作；USBFS 寄存器可为 NVIC 产生相应 USB 中断；USBFS 包含一个 1.25KB Data FIFO，数据 FIFO 和串行接口引擎(SIE)相连，在设备模式下，数据 FIFO 可分为一个接收 FIFO 和多个发送 FIFO，其中所有的 OUT 端点共享接收 FIFO，每个 IN 端点可独立使用一个发送 FIFO，在主机模式下，数据 FIFO 分为三个部分，分别是：用于接收包数据的接收 FIFO、用于非周期性发送包数据的非周期性发送 FIFO 和用于周期性发送包数据的周期性发送 FIFO，所有的 IN 通道共享接收 FIFO，所有的周期性 OUT 通道共享周期性发送 FIFO，所有的非周期性 OUT 通道共享非周期性发送 FIFO；USB 时钟域时钟来自 RCU 所配置的 48Mhz 时钟；USBFS 控制器通过 UIMI 多路器与 USBFS PHY 物理层相连，USBFS PHY 物理层用于实现 USB 通信，其包括 USB 收发器和 USB 接口电路等。

图 1-1. USBFS 结构框图



USBFS 接口模块主要特性

USBFS 接口模块的主要特性如*表 1-1. USB_FS 接口模块主要特性列表*所示。

表 1-1. USB_FS 接口模块主要特性列表

主机特性	设备特性
8 个主机通道	4 个双向端点(包括端点 0)
周期性 TX_FIFO：存储需要传输的同步和中断传输数据；非周期性 TX_FIFO：存储需要传输的批量和控制传输请求。	4 个独立的 TX_FIFO 对应于 4 个 IN 端点
一个共享的 RX_FIFO 用以接收数据	一个共享的 RX_FIFO 用以接收数据
需要外接电源芯片为所连接的设备供电	支持软件断开
支持 USB 2.0 全速（ 12Mb/s） /低速（ 1.5Mb/s）主机模式	支持 USB 2.0 全速（ 12Mb/s） 设备模式
周期性队列：管理最多 8 个同步和中断传输请求；非周期性队列：管理最多 8 个批量和控制传输请求。	
支持所有的 4 种传输方式：控制传输、批量传输、中断传输和同步传输	
支持遵循 HNP（主机协商协议）和 SRP（会话请求协议）的 OTG 协议	

USBFS 主机模式

USBFS 在以下四种情况下，作为主机使用：

⑴ 插入 USB A 电缆时的 USBFS 接口模块的默认状态(OTG A 器件)；

⑵ 插入 USB B 电缆后的 USBFS 接口模块(OTG B 器件)被 HNP 切换为主机角色的状态；

⑶ OTG A 器件的全局 USB 配置寄存器中的 HNP 功能位清零(不可利用 HNP 切换为设备)；

⑷ 仅作主机：全局 USB 配置寄存器中的强制主机模式位置位，强制 USBFS 仅作为主机，这种情况下，将忽略 ID 线的状态，并且使能 DP 和 DM 上集成的下拉电阻。

USBFS 作为主机或设备的连接示意图如图 1-2 所示，若 USBFS 仅作为主机，则需要 5V 供电电源，在此种情况下，5V 供电电源为外部 USB 设备供电。

USBFS 设备模式

USBFS 在以下四种情况下作为设备使用：

⑴ 插入 USB B 端电缆时 USBFS 接口模块的默认状态(OTG B 器件)；

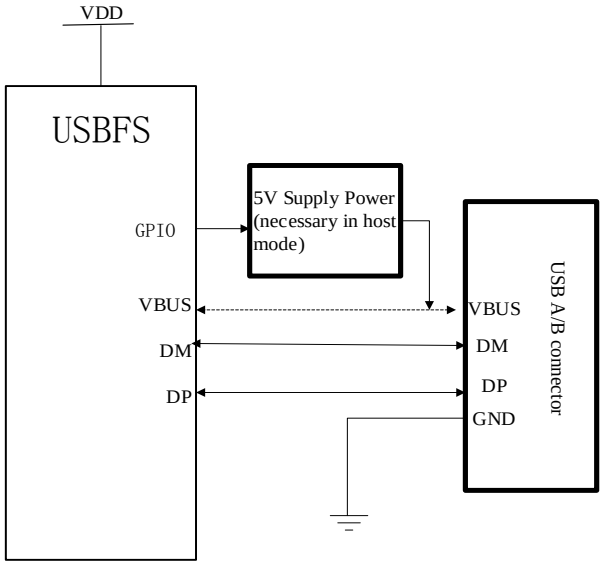
⑵ 插入 USB A 端电缆后 USBFS 接口模块(OTG A 器件)被 HNP 切换为设备角色后的状态；

⑶ OTG B 器件的全局 USB 配置寄存器中的 HNP 功能位清零(不可利用 HNP 切换为主机)；

⑷ 仅作设备：全局 USB 配置寄存器中的强制设备模式位置位，强制 USBFS 仅作为设备，这种情况下，将忽略 ID 线的状态。

USBFS 作为主机或设备的连接示意图如*图 1-2. USBFS 作为主机或设备连接示意图*所示。若 USBFS 仅作为主机，5V 供电电源不需要，VBUS 引脚用于检测 VBUS 电压，监测 OTG_FS 是否和主机断开连接；DP 和 DM 为 USB 通信的差分信号线。

图 1-2. USBFS 作为主机或设备连接示意图



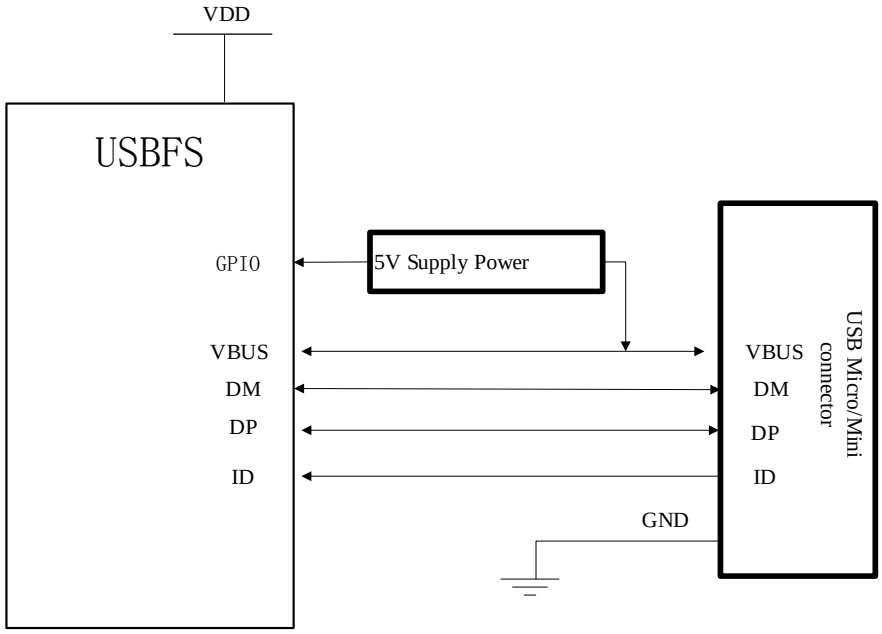
USBFS OTG 模式

USBFS OTG 设备的连接示意图如

图 1-3. **USBFS 作为 OTG 设备连接示意图**所示。除正常 USB 接口的四根线外，还多出了一根 ID 线，该线用于决定该 USBFS 接口模块的角色。若 USB 电缆 B 端连入，由于其 ID 线悬空，设备在 ID 线上集成上拉电阻，设备将检测到 ID 线高电平，USBFS 将采取默认的从机角色；若 USB 电缆 A 端连入，由于其 ID 线接地，则 OTG_FS 将发出 ID 线状态更改中断以初始化主机软件，并自动切换为主机角色。

GPIO 口用于控制产生 5V 电源，该电源为外部输入电源，当 USBFS 作为 A 设备时，该电源为另一设备供电和为 MCU 供电；VBUS 引脚用于监控 VBUS 电压。

图 1-3. USBFS 作为 OTG 设备连接示意图



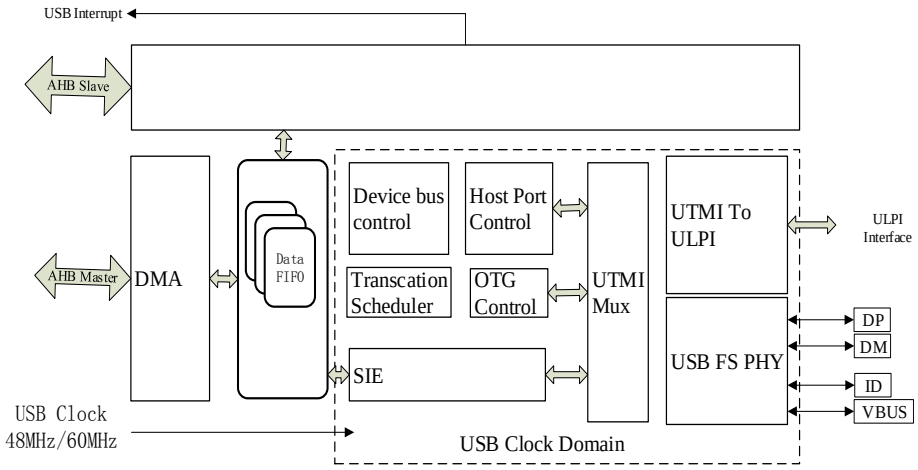
1.2.2. USBHS 原理

USBHS 接口模块内部包含一个嵌入式全速 USBFS PHY（物理层），该物理层比 USBFS 模块物理层功能更强，可支持多达 6 个双向设备端点和 12 个主机通道，4K FIFO 空间。因此 USBHS 可利用内部嵌入式 USBFS PHY 实现全速和低速操作。如果 USBHS 连接到外部 ULPI 物理层，则可实现 USB 高速通信，其数据传输速率可高达 480Mbps。当 USBHS 操作在高速主机模式下时，USBHS 可支持 HUB 连接。另外，USBHS 内部还有一个 DMA 引擎，利用该引擎可加速 USBHS 和系统之间的数据传输。

USBHS 结构

USBHS 结构框图如*图 1-4. USBHS 结构框图*所示，与 USBFS 相比，USBHS 结构框图中增加了 DMA 传输引擎和 ULPI 接口，可用于 USBHS 高速通信。

图 1-4. USBHS 结构框图

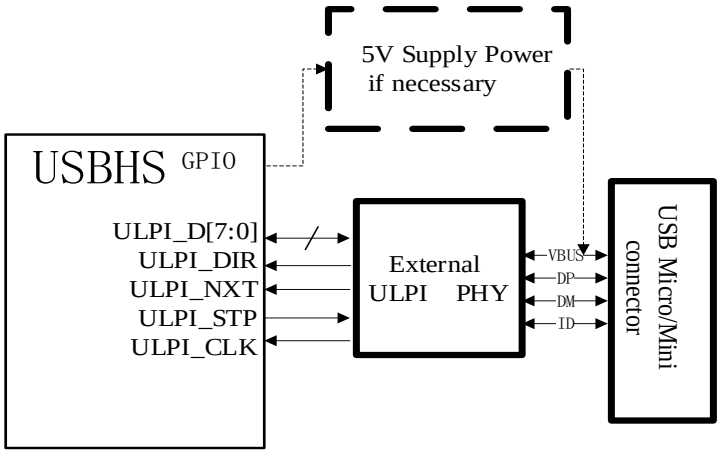


USBHS 使用外部 ULPI PHY

如*图 1-5. 使用外部 ULPI PHY 的连接示意图*所示， USBHS 为外部 PHY 提供了一个 ULPI 接口如果需要使用 USBHS 模块完成高速 USB 应用，那么则需要一个外部高速 ULPI PHY。结合外部 ULPI PHY，USBHS 支持高速主机和设备，也支持前文中 USBFS PHY 所描述的所有模式。

软件需要清除 USBHS_GUSBCS 寄存器中的 EMBPHY 控制位以使能 ULPI 接口。当 ULPI 模式能使，60MHz 时钟需要从 ULPI_CLK 引脚引入。软件可以在 RCU 模块中打开或关闭该 60MHz ULPI 时钟。

图 1-5. 使用外部 ULPI PHY 的连接示意图



此外，USBHS 模块原理与 USBFS 基本相同，在此不再详述。

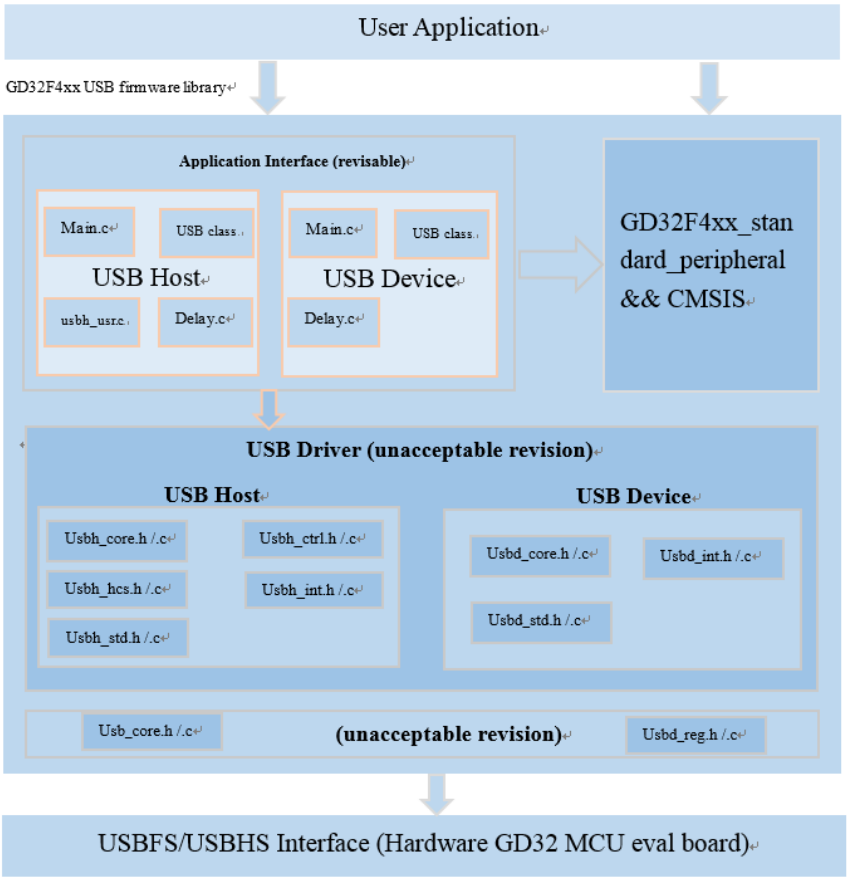
1.3. USBFS/USBHS 模块固件库

1.3.1. USBFS/USBHS 模块固件库架构

GD32F4xx 系列 MCU 的 USBFS/USBHS 接口模块固件库架构如*图 1-6. GD32F4xx 系列 MCU 的 USBFS/USBHS 接口模块固件库架构*所示。在本架构示意图中，仅介绍 USBFS/USBHS 作为主机和设备的架构，作为 OTG 设备的架构的设备并未在图中说明。其中，用户应用程序(User application)调用 GD32 USB 固件库中的接口实现 USB 的数据通信，架构的最底层为 GD32 MCU 开发板的硬件层。其中，GD32 USBFS/USBHS 固件库分为三层，顶层为应用接口层，用户可以修改，中间层为 USB_Host 或 USB_Device，底层为 USB_Drivers，中间层和底层统称为 USB 固件库驱动，该驱动层用户不可修改。

需要说明，在应用接口层中，USB_Class 项为一类文件，包含实现 USB 主机类或设备类的相关文件。

图 1-6. GD32F4xx 系列 MCU 的 USBFS/USBHS 接口模块固件库架构



USBFS/USBHS 接口模块固件库分层文件及库函数说明

1.3.2. 底层文件及库函数说明

USB_Driver 底层包含两个文件，具体说明如*表 1-2. USB 底层文件说明*所示。

表 1-2. USB 底层文件说明

文件名称	说明
usb_core.h/.c	USB 内核驱动
usb_reg.h	USB 寄存器操作

其中，usb_core.h/.c 文件中库函数说明如*表 1-3. usb_core.h/.c 文件库函数说明列表*所示。

表 1-3. usb_core.h/.c 文件库函数说明列表

函数名称	功能描述
usb_commonint_enable	使能通用中断
usb_core_reset	USB 内核软件复位
usb_fifo_write	写一包的数据到相应端点的 TX FIFO
usb_fifo_read	从相应端点的 RX FIFO 中读取一包数据
usb_core_select	选择 USB 内核
usb_core_init	初始化 USB 控制器寄存器和参数
usb_txfifo_flush	冲刷一个 TX FIFO 或所有的 TX FIFO
usb_rxfifo_flush	冲刷全部的 RX FIFO
usb_mode_set	设置操作模式(主机或设备)
usb_hostcore_init	为主机模式初始化 USB 内核
usb_vbus_drive	配置为 USB 端口供电
usb_hostint_enable	使能主机模式中断
usb_port_reset	复位主机端口
usb_hostchannel_init	初始化主机通道
usb_hostchannel_startxfer	主机通道启动发送
usb_hostchannel_halt	停止通道
usb_hostchannel_ping	发送一个 PING 令牌
usb_host_stop	停止主机并清除 FIFOs
usb_devcore_init	为设备模式初始化 USB 内核寄存器
usb_devint_enable	使能设备模式中断
usb_ep0_startout	配置端点 0 准备接收 SETUP 令牌包
usb_remotewakeup_active	启动远程唤醒
usb_clock_ungate	启动 USB 内核时钟
usb_device_stop	停止 USB 设备并清除 FIFOs

1.3.3. USBFS 主机中间层文件及库函数说明

USB_Host 中间层包含 5 个文件，具体文件说明如*表 1-4. USB_Host 中间层文件说明*所示。

表 1-4. USB_Host 中间层文件说明

文件名称	说明
usbh_core.h/.c	USB 主机状态机处理函数

usbh_ctrl.h/.c	USB 主机控制传输处理函数
usbh_hcs.h/.c	USB 主机通道处理函数
usbh_int.h/.c	USB 主机模式中断处理函数
usbh_std.h/.c	USB 主机枚举标准处理函数

其中，usbh_core.h/.c 文件中的库函数说明如*表 1-5. usbh_core.h/.c 文件库函数说明列表*所示。

表 1-5. usbh_core.h/.c 文件库函数说明列表

函数名称	功能描述
host_state_polling_fun	主机状态机轮询
host_idle_handle	HOST_IDLE 状态处理
host_dev_attached_handle	HOST_DEV_ATTACHED 状态处理
host_enum_handle	HOST_ENUMERATION 状态处理
host_user_input_handle	HOST_USER_INPUT 状态处理
host_class_request_handle	HOST_CLASS_REQUEST 状态处理
host_class_handle	HOST_CLASS 状态处理
host_suspended_handle	HOST_SUSPENDED 状态处理
host_error_handle	HOST_ERROR 状态处理
host_dev_detached_handle	HOST_DEV_DETACHED 状态处理
host_detect_dev_speed_handle	HOST_DETECT_DEV_SPEED 状态处理
usbh_connected	设备连接中断回调函数
usbh_disconnected	设备断开中断回调函数
usbh_sof	SOF 中断回调函数
hcd_init	主机核心驱动初始化
hcd_is_device_connected	检查设备是否连接
hcd_urb_state_get	返回最新 URB 状态
hcd_xfer_count_get	返回传输数据长度
usbh_deinit	去初始化主机
scd_init	状态机核心驱动初始化
scd_table_regist	状态机核心驱动状态表注册
scd_begin	启动状态机核心驱动
scd_state_move	状态机核心驱动状态转移
scd_event_handle	状态机核心驱动事件处理
scd_table_push	将当前状态表压入状态堆栈
scd_table_pop	将状态堆栈中的状态表
class_req_state_polling_fun	设备类请求状态机轮询函数
class_state_polling_fun	设备类状态机轮询函数
only_state_move	状态转移函数
goto_up_state_fun	返回上层状态机

usbh_ctrl.h/.c 文件中的库函数说明如*表 1-6. usbh_ctrl.h/.c 库函数说明列表*所示。

表 1-6. usbh_ctrl.h/.c 库函数说明列表

函数名称	功能描述

ctrl_state_polling_fun	控制传输状态机轮询
ctrl_idle_handle	CTRL_IDLE 状态处理
ctrl_setup_handle	CTRL_SETUP 状态处理
ctrl_data_handle	CTRL_DATA 状态处理
ctrl_status_handle	CTRL_STATUS 状态处理
ctrl_error_handle	CTRL_ERROR 状态处理
ctrl_stalled_handle	CTRL_STALLED 状态处理
ctrl_complete_handle	CTRL_COMPLETE 状态处理
usbh_xfer	从主机端发送数据
usbh_ctltx_setup	发送 SETUP 令牌包到设备
hcd_submit_request	准备通道，启动一次传输

usbh_hcs.h/.c 文件中的库函数说明如*表 1-7. usbh_hcs.h/.c 库函数说明列表*所示。

表 1-7. usbh_hcs.h/.c 库函数说明列表

函数名称	功能描述
usbh_channel_open	打开主机通道
usbh_channel_modify	更改一个通道
usbh_channel_alloc	为管道配置一个新的通道
usbh_channel_free	释放 USB 主机通道
usbh_allchannel_dealloc	释放所有的 USB 主机通道
usbh_freechannel_get	为配置的设备端点获取一个空闲的主机通道

usbh_int.h/.c 文件中的库函数说明如*表 1-8. usbh_int.h/.c 库函数说明列表*所示。

表 1-8. usbh_int.h/.c 库函数说明列表

函数名称	功能描述
usbh_isr	处理全局主机中断
usbh_intf_sof	SOF 中断处理
usbh_intf_hc	处理所有的主机通道中断
usbh_intf_disconnect	处理断开中断
usbh_intf_nptxfifo_empty	处理非周期性发送 FIFO 空中断
usbh_intf_ptxfifo_empty	处理周期性发送 FIFO 空中断
usbh_intf_port	处理主机端口中断
usbh_intf_hc_out	处理 OUT 通道中断
usbh_intf_hc_in	处理 IN 通道中断
usbh_intf_rxfifo_noempty	处理接收 FIFO 非空中断
usbh_intf_iso_incomplete_xfer	处理不完全周期性传输中断

Usbh_std.h/.c 文件库函数说明如*表 1-9. usbh_std.h/.c 文件库函数说明列表*所示。

表 1-9. usbh_std.h/.c 文件库函数说明列表

函数名称	功能描述
enum_state_polling_fun	枚举状态机轮询
enum_idle_handle	ENUM_IDLE 状态处理
enum_get_full_dev_desc_handle	ENUM_GET_FULL_DEV_DESC 状态处理

enum_set_addr_handle	ENUM_SET_ADDR 状态处理
enum_get_cfg_desc_handle	ENUM_GET_CFG_DESC 状态处理
enum_get_full_cfg_desc_handle	ENUM_GET_FULL_CFG_DESC 状态处理
enum_get_mfc_string_desc_handle	ENUM_GET_MFC_STRING_DESC 状态处理
enum_get_product_string_desc_handle	ENUM_GET_PRODUCT_STRING_DESC 状态处理
enum_get_serialnum_string_desc_handle	ENUM_GET_SERIALNUM_STRING_DESC 状态处理
enum_set_configuration_handle	ENUM_SET_CONFIGURATION 状态处理
enum_dev_configured_handle	ENUM_DEV_CONFIGURED 状态处理
usbh_enum_desc_get	在主机枚举阶段获取描述符
usbh_enum_addr_set	在主机枚举阶段设置地址
usbh_enum_cfg_set	在主机枚举阶段设置配置
usbh_device_desc_parse	解析设备描述符
usbh_cfg_desc_parse	解析配置描述符
usbh_interface_desc_parse	解析接口描述符
usbh_endpoint_desc_parse	解析端点描述符
usbh_string_desc_parse	解析字符串描述符
usbh_next_desc_get	获取下一个描述符包头

1.3.4. USBFS 设备中间层文件及库函数说明

USB_Device 中间层包含 3 个文件，具体文件说明如*表 1-10. USB_Device 中间层文件说明*所示。

表 1-10. USB_Device 中间层文件说明

文件名称	说明
usbd_core.h/.c	USB 设备模式内核驱动函数
usbd_int.h/.c	USB 设备模式中断处理函数
usbd_std.h/.c	USB 设备枚举标准处理函数

其中，usbd_core.h/.c 文件中的库函数说明如*表 1-11. usbd_core.h/.c 文件库函数说明列表*所示。

表 1-11. usbd_core.h/.c 文件库函数说明列表

函数名称	功能描述
usbd_init	初始化设备模式
usbd_ep_init	端点初始化
usbd_ep_deinit	端点去初始化
usbd_ep_rx	端点准备接收
usbd_ep_tx	端点准备发送
usbd_status_enum usbd_ctltx	控制端点发送数据
usbd_status_enum usbd_ctlrx	控制端点接收数据

usbd_status_enum usbd_ctlstatus_tx	控制端点发送状态
usbd_status_enum usbd_ctlstatus_rx	控制端点接收状态
usbd_ep_stall	设置端点 stall 状态
usbd_ep_clear_stall	清除端点 stall 状态
usbd_ep_fifo_flush	清除 FIFO
usbd_rxcount_get	获取接收数据长度

usbd_int.h/.c 文件中的库函数说明如*表 1-12. usbd_int.h/.c 库函数说明列表*所示。

表 1-12. usbd_int.h/.c 库函数说明列表

函数名称	功能描述
usbd_isr	处理设备中断
usbd_intf_outep	处理 OUT 端点中断
usbd_intf_inep	处理 IN 端点中断
usbd_intf_earlysuspend	处理早挂起中断
usbd_intf_suspend	处理挂起中断
usbd_intf_resume	处理唤醒中断
usbd_intf_sof	处理 SOF 中断
usbd_intf_rxfifo	处理接收 FIFO 非空中断
usbd_intf_reset	处理复位中断
usbd_intf_enumfinish	处理枚举完成中断
usbd_intf_isoinincomplete	处理未完成周期性同步 IN 传输中断
usbd_intf_isooutincomplete	处理未完成周期性同步 OUT 传输中断
usbd_emptytxfifo_write	处理发送 FIFO 空中断
usbd_intf_sessionrequest	处理会话请求中断
usbd_intf_otg	处理 OTG 中断

Usbd_std.h/.c 文件库函数说明如*表 1-13. usbd_std.h/.c 文件库函数说明列表*所示。

表 1-13. usbd_std.h/.c 文件库函数说明列表

函数名称	功能描述
usbd_setup_transaction	处理 SETUP 事务
usbd_out_transaction	处理 OUT 事务
usbd_in_transaction	处理 IN 事务
usbd_standard_request	处理标准设备请求
usbd_device_class_request	处理设备类请求
usbd_vendor_request	处理厂商自定义请求
usbd_reserved	处理保留请求
usbd_device_descriptor_get	获取设备描述符
usbd_configuration_descriptor_get	获取配置描述符
usbd_string_descriptor_get	获取字符串描述符
usbd_getstatus	处理获取状态请求
usbd_clrfeature	处理清除特性请求
usbd_setfeature	处理设置特性请求
usbd_setaddress	处理设置地址请求

usbd_getdescriptor	处理获取描述符请求
usbd_setdescriptor	处理设置描述符请求
usbd_getconfig	处理获取配置请求
usbd_setconfig	处理设置配置请求
usbd_getinterface	处理获取接口请求
usbd_setinterface	处理设置接口请求
usbd_synchframe	处理同步帧请求
usbd_setup_request_parse	解析 SETUP 数据包
usbd_enum_error	处理枚举错误

1.3.5. 应用接口顶层文件及库函数说明

应用接口顶层包含四个文件，具体说明如*表 1-14. 应用接口顶层文件说明列表*所示。

表 1-14. 应用接口顶层文件说明列表

文件名称	说明
main.c	主应用程序接口
usbh_usr.c	用户应用程序接口
usb_delay.c	延迟函数实现接口
application class	设备类应用程序接口

其中 main.c 文件主要实现主应用程序接口函数，其函数说明如*表 1-15. main.c 文件函数说明列表*所示。

表 1-15. main.c 文件函数说明列表

函数名称	功能描述
main	应用程序主函数
usb_rcu_init	USB RCU 初始化
usb_gpio_init	USB GPIO 初始化
usb_hwp_interrupt_enable	配置 USB 全局中断
usb_hwp_vbus_drive	通过 GPIO 驱动 VBUS 信号线
usb_hwp_vbus_config	配置 VBUS 引脚

usbh_usr.c 文件主要实现 usr_cb 用户回调函数结构体，该结构体定义如*图 1-7. 用户回调函数结构体*所示。

图 1-7. 用户回调函数结构体

```
usbh_user_callback_struct user_callback_funs =
{
    usbh_user_init,
    usbh_user_deinit,
    usbh_user_device_connected,
    usbh_user_device_reset,
    usbh_user_device_disconnected,
    usbh_user_over_current_detected,
    usbh_user_device_speed_detected,
    usbh_user_device_descavailable,
    usbh_user_device_address_assigned,
    usbh_user_configuration_descavailable,
    usbh_user_manufacturer_string,
    usbh_user_product_string,
    usbh_user_serialnum_string,
    usbh_user_enumeration_finish,
    usbh_user_userinput,
    NULL,
    usbh_user_device_not_supported,
    usbh_user_unrecovered_error
};
```

为了介绍 usbh_usr.h/c 文件的普适性，在此我们仅介绍用户回调函数结构体中所需实现的函数，具体说明如*表 1-16. usr_cb 用户回调函数结构体函数说明列表*所示。

表 1-16. usr_cb 用户回调函数结构体函数说明列表

函数名称	功能描述
init	主机模式初始化时的用户操作
deinit	将用户状态恢复为默认
device_connected	设备连接时的用户操作
device_reset	复位设备时的用户操作
device_disconnected	设备断开时的用户操作
over_current_detected	设备过载时的用户操作
device_speed_detected	检测设备速度时的用户操作
device_desc_available	当设备描述符可用时的用户操作
device_address_set	设备成功设置地址时的用户操作
configuration_desc_available	当配置描述符可用时的用户操作
manufacturer_string	当厂商字符串存在时的用户操作
product_string	当产品字符串存在时的用户操作
serial_num_string	当产品序列号存在时的用户操作
enumeration_finish	枚举完成时的用户操作
user_input	进入用户状态时的用户操作
user_application	用户应用程序
device_not_supported	设备不支持时的用户操作
unrecovered_error	当不可恢复的错误发生时的用户操作

Usb_delay.c 文件函数说明如*表 1-17. usb_delay.c 文件函数说明列表*所示。

表 1-17. usb_delay.c 文件函数说明列表

函数名称	功能描述
usb_time_init	利用定时器 2 初始化延时单元
usb_udelay	以微秒为单位的延迟函数

usb_mdelay	以毫秒为单位的延迟函数
hwp_delay	硬件延时函数
hwp_time_set	硬件定时器初始化

USB_Class 包含主机类实现函数文件，USB 协议支持的主机类很多，在此我们仅介绍 HID 和 MSC 主机类实现的函数文件，具体说明如*表 1-18. HID 和 MSC 主机类库函数*所示。

表 1-18. HID 和 MSC 主机类库函数

设备类	文件名称	函数名称	说明
HID 主机类	usbh_hid_core.h/c	hid_req_state_polling_fun	HID 请求状态机轮询函数
		hid_state_polling_fun	HID 状态机轮询函数
		hid_clear_feature	HID 清除特性
		usbh_hid_desc_parse	解析 HID 描述符
		usbh_hid_interface_init	初始化 HID 类接口
		usbh_hid_interface_deinit	为 HID 类的主机通道恢复默认设置
		hid_req_set_idle	设置空闲状态
		hid_req_set_protocol	设置协议状态
	Usbh_hid_keybd.h/c	keybrd_decode	解析键盘按键
	usbh_hid_mouse.h/c	mouse_init	初始化鼠标状态
		mouse_decode	解析鼠标数据
		hid_mouse_button_pressed	解析鼠标按键按下函数
		hid_mouse_button_released	解析鼠标按键松开函数
		hid_mouse_update_position	更新鼠标位置函数
MSC 主机类	usbh_msc_bot.h/c	usbh_msc_init	初始化大容量存储器参数
		usbh_msc_handle_botxfer	管理 BOT 传输的不同状态
		usbh_msc_bot_abort	为 STALL 状态管理不同的错误处理
		usbh_msc_decode_csw	解析 CSW 命令块封包
	usbh_msc_core.h/c	usbh_msc_interface_init	接口初始化
		usbh_msc_interface_deinit	接口恢复默认状态
		msc_req_state_polling_fun	MSC 请求处理轮询函数
		msc_state_polling_fun	MSC 状态轮询函数
		usbh_msc_max_lun_get	获取大容量存储设备的最大逻辑单元
		usbh_msc_error_handle	处理错误事件
		usbh_clear_feature	清除或禁用一个特性
	usbh_msc_scsi.c	usbh_msc_test_unit_ready	向设备发送 test uint ready 命令
		usbh_msc_read_capacity10	向设备发送 read capacity 10 命令
		usbh_msc_mode_sense6	向设备发送 sense6 命令

		usbh_msc_request_sense	向设备发送 requset sense 命令
		usbh_msc_write10	向设备发送 write10 命令
		usbh_msc_read10	向设备发送 read10 命令
	usbh_msc_fatfs.h/c	disk_initialize	初始化磁盘驱动
		disk_status	获取磁盘状态
		disk_read	读取磁盘扇区
		disk_write	写磁盘扇区
		disk_ioctl	IO 控制功能函数

1.4. USBFS 主机状态机

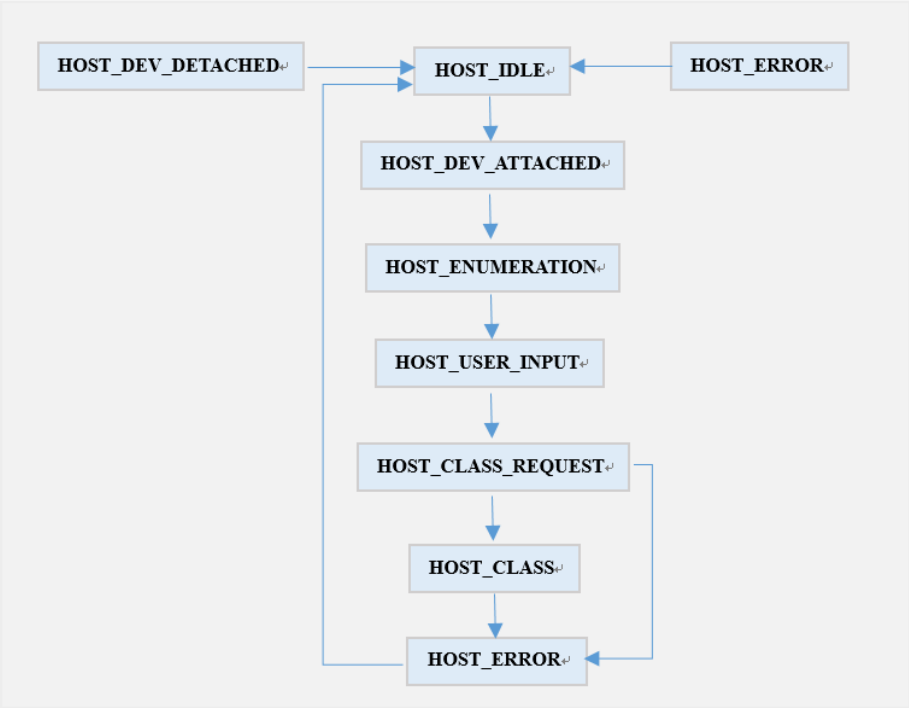
USBFS 主机状态机采用状态机嵌套和状态机查表的方式进行处理。USBFS 主机状态机包含主机状态机处理(usbh_core.c 中)、枚举状态机处理(usbh_std.c 中)和控制传输状态机处理(usbh_ctrl.c 中)。

USBFS 主机状态机查询表如*图 1-8. USBFS 主机状态机表*所示。首先为大家介绍如何阅读该表，该表分为 9 项，每项为一种状态转移，分为四个部分，第一部分为当前状态，第二部分为接收到的当前事件、第三部分为转移到的下一个状态、第四个部分为转移过程中所需要执行的事件函数。具体可表示为：在当前状态下，如果接收到当前所对应的事件，经过查表执行事件函数，进而转移到下一个状态。采用状态机查表的方式结构清晰，易于阅读。

图 1-8. USBFS 主机状态机表

```
state_table_struct host_handle_table[HOST_HANDLE_TABLE_SIZE] =
{
    /* the current state  the current event        the next state        the event function */
    {HOST_IDLE,           HOST_EVENT_ATTACHED,     HOST_DEV_ATTACHED,    only_state_move    },
    {HOST_DEV_ATTACHED,   HOST_EVENT_ENUM,         HOST_ENUMERATION,     only_state_move    },
    {HOST_ENUMERATION,    HOST_EVENT_USER_INPUT,   HOST_USER_INPUT,      only_state_move    },
    {HOST_USER_INPUT,     HOST_EVENT_CLASS_REQ,    HOST_CLASS_REQUEST,   only_state_move    },
    {HOST_CLASS_REQUEST,  HOST_EVENT_CLASS,        HOST_CLASS,           only_state_move    },
    {HOST_CLASS,          HOST_EVENT_ERROR,        HOST_ERROR,           only_state_move    },
    {HOST_ERROR,          HOST_EVENT_IDLE,         HOST_IDLE,            only_state_move    },
    {HOST_DEV_DETACHED,   HOST_EVENT_IDLE,         HOST_IDLE,            only_state_move    },
    {HOST_CLASS_REQUEST,  HOST_EVENT_ERROR,        HOST_ERROR,           only_state_move    },
};
```

图 1-9. USBFS 主机状态机流程图

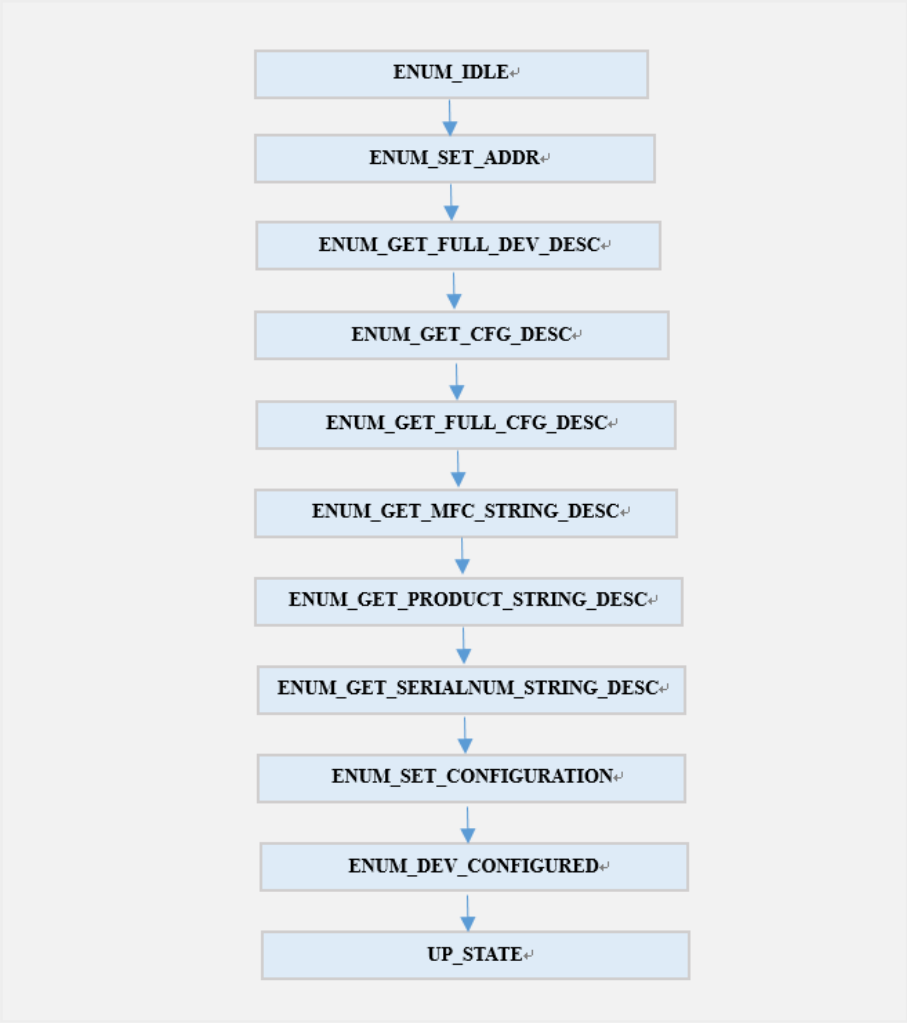


由主机状态机查询表可知，在 HOST_DEV_ATTACHED 状态下，如果接收到 HOST_EVENT_ENUM 事件，将进入到 HOST_ENUMERATION 状态。在 HOST_ENUMERATION 状态下，将进入到枚举状态机中处理，枚举状态机查询表如*图 1-10. 枚举状态机处理查询表*所示，枚举状态机流程图如*图 1-11. USBFS 枚举状态机流程图*所示。在由主机状态机(上层状态机)转移到枚举状态机(下层状态机)的处理过程中，将采用 scd_table_push()函数将上次状态机保持到状态机堆栈中，以利于返回上层状态机时继续处理。

图 1-10. 枚举状态机处理查询表

```
state_table_struct enum_handle_table[ENUM_HANDLE_TABLE_SIZE] =
{
    /* the current state            the current event                   the next state                  the event function */
    {ENUM_IDLE,                     ENUM_EVENT_SET_ADDR,                ENUM_SET_ADDR,                  only_state_move     },
    {ENUM_SET_ADDR,                 ENUN_EVENT_GET_FULL_DEV_DESC,       ENUM_GET_FULL_DEV_DESC,         only_state_move     },
    {ENUM_GET_FULL_DEV_DESC,        ENUN_EVENT_GET_CFG_DESC,            ENUM_GET_CFG_DESC,              only_state_move     },
    {ENUM_GET_CFG_DESC,             ENUN_EVENT_GET_FULL_CFG_DESC,       ENUM_GET_FULL_CFG_DESC,         only_state_move     },
    {ENUM_GET_FULL_CFG_DESC,        ENUN_EVENT_GET_MFC_STRING_DESC,     ENUM_GET_MFC_STRING_DESC,       only_state_move     },
    {ENUM_GET_MFC_STRING_DESC,      ENUN_EVENT_GET_PRODUCT_STRING_DESC, ENUM_GET_PRODUCT_STRING_DESC,   only_state_move     },
    {ENUM_GET_PRODUCT_STRING_DESC,  ENUN_EVENT_GET_SERIALNUM_STRING_DESC, ENUM_GET_SERIALNUM_STRING_DESC, only_state_move     },
    {ENUM_GET_SERIALNUM_STRING_DESC, ENUN_EVENT_SET_CONFIGURATION,      ENUM_SET_CONFIGURATION,         only_state_move     },
    {ENUM_SET_CONFIGURATION,        ENUN_EVENT_DEV_CONFIGURED,          ENUM_DEV_CONFIGURED,            only_state_move     },
    {ENUM_DEV_CONFIGURED,           GO_TO_UP_STATE_EVENT,               UP_STATE,                       goto_up_state_fun   },
};
```

图 1-11. USBFS 枚举状态机流程图

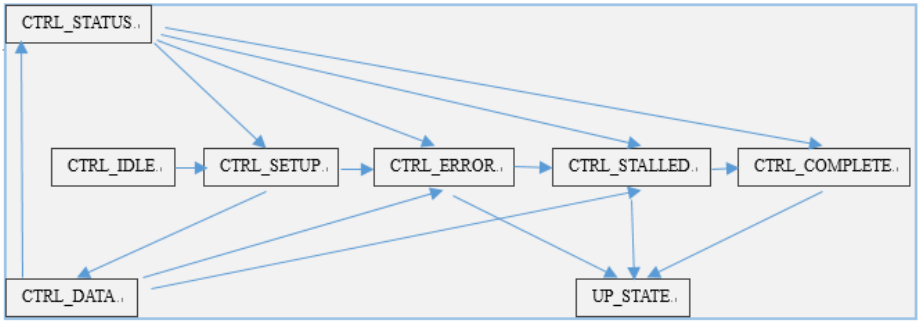


在枚举状态机查询表中，如果在 ENUM_DEV_CONFIGURED 状态下，接收到 UP_STATE 事件，表明枚举完成，将会采用 scd_table_pop()返回到上层主机状态机中继续处理。另外，在枚举的过程中，需要控制传输，将会继续进入到控制传输状态机(更下一层状态机)中处理。控制传输状态机查询表如*图 1-12. 控制传输状态机处理查询表*所示，控制传输状态机流程图如*图 1-13. 控制传输状态机流程图*所示。

图 1-12. 控制传输状态机处理查询表

```
state_table_struct ctrl_handle_table[CTRL_HANDLE_TABLE_SIZE] =
{
    /* the current state   the current event          the next state       the event function */
    {CTRL_IDLE,            CTRL_EVENT_SETUP,         CTRL_SETUP,          only_state_move     },
    {CTRL_SETUP,           CTRL_EVENT_DATA,          CTRL_DATA,           only_state_move     },
    {CTRL_SETUP,           CTRL_EVENT_STATUS,        CTRL_STATUS,         only_state_move     },
    {CTRL_SETUP,           CTRL_EVENT_ERROR,         CTRL_ERROR,          only_state_move     },
    {CTRL_DATA,            CTRL_EVENT_STATUS,        CTRL_STATUS,         only_state_move     },
    {CTRL_DATA,            CTRL_EVENT_ERROR,         CTRL_ERROR,          only_state_move     },
    {CTRL_DATA,            CTRL_EVENT_STALLED,       CTRL_STALLED,        only_state_move     },
    {CTRL_STATUS,          CTRL_EVENT_COMPLETE,      CTRL_COMPLETE,       only_state_move     },
    {CTRL_STATUS,          CTRL_EVENT_ERROR,         CTRL_ERROR,          only_state_move     },
    {CTRL_STATUS,          CTRL_EVENT_STALLED,       CTRL_STALLED,        only_state_move     },
    {CTRL_ERROR,           GO_TO_UP_STATE_EVENT,     UP_STATE,            goto_up_state_fun   },
    {CTRL_STALLED,         GO_TO_UP_STATE_EVENT,     UP_STATE,            goto_up_state_fun   },
    {CTRL_COMPLETE,        GO_TO_UP_STATE_EVENT,     UP_STATE,            goto_up_state_fun   },
};
```

图 1-13. 控制传输状态机流程图



在一次控制传输中，初始状态为 CTRL_IDLE，当一次控制传输完成，状态将转移到 CTRL_COMPLETE，然后将会返回上层状态机。

1.5. 中断处理

USBFS 接口模块与 USBD 接口模块在中断处理方面有些不同，USBD 接口模块的正确传输中断分为低优先级和高优先级中断，一般情况下，IN 和 OUT 事务的正确传输在低优先级中断中进行处理，根据传输方向的标志位 IFR_DIR 进行区分。如*表 1-19. USB 全局中断*所示，在 USBFS 接口模块中 IN 和 OUT 事务的正确传输分为两个不同的中断，分别为 OUT 端点中断标志 GINTF_OEPIF 和 IN 端点中断标志 GINTF_IEPIF。OUT 端点中断处理函数如*图 1-14. OUT 端点中断处理函数*所示。

表 1-19. USB 全局中断

中断标志	描述	运行模式
SEIF	会话中断	主机或设备模式
DISCIF	断开连接中断标志	主机模式
IDPSC	ID 引脚状态变化	主机或设备模式
PTXFEIF	周期性 Tx FIFO 空中断标志	主机模式
HCIF	主机通道中断标志	主机模式
HPIF	主机端口中断	主机模式
ISOONCIF/PXNCIF	周期性传输未完成中断标志 / 同步OUT传输未完成中断标志	主机或设备模式
ISOINCIF	同步 IN 传输未完成中断标志	设备模式
OEPIF	OUT 端点中断标志	设备模式
IEPIF	IN 端点中断标志	设备模式

中断标志	描述	运行模式
EOPFIF	周期性帧尾中断标志	设备模式
ISOOPDIF	同步 OUT 丢包中断标志	设备模式
ENUMF	枚举完成	设备模式
RST	USB 复位	设备模式
SP	USB挂起	设备模式
ESP	早挂起	设备模式
GONAK	全局OUT NAK有效	设备模式
GNPINAK	全局非周期IN NAK有效	设备模式
NPTXFEIF	非周期Tx FIFO空中断标志	主机模式
RXFNEIF	Rx FIFO非空中断标志	主机或设备模式
SOF	帧首	主机或设备模式
OTGIF	OTG 中断标志	主机或设备模式
MFIF	模式错误中断标志	主机或设备模式

图 1-14. OUT 端点中断处理函数

```
static uint32_t usbd_intf_outep (usb_core_handle_struct *pudev)
{
    uint8_t endp_num = 0U;
    uint32_t endp_intr = 0U;

    __IO uint32_t out_endp_intr = 0U;

    /* read in the device interrupt bits */
    USB_DAOEP_INTR_READ(endp_intr);

    while (endp_intr) {
        if (endp_intr & 0x1U) {
            USB_DOEP_INTR_READ(out_endp_intr, (uint16_t)endp_num);

            /* transfer complete interrupt */
            if (out_endp_intr & DOEPINTF_TF) {
                USB_DOEPxINTF((uint16_t)endp_num) = DOEPINTF_TF;

                /* data receive is completed */
                usbd_out_transaction(pudev, endp_num);
            }

            /* endpoint disable interrupt */
            if (out_endp_intr & DOEPINTF_EPDIS) {
                USB_DOEPxINTF((uint16_t)endp_num) = DOEPINTF_EPDIS;
            }

            /* setup phase finished interrupt (just for control endpoints) */
            if (out_endp_intr & DOEPINTF_STPF) {
                /* setup phase is completed */
                usbd_setup_transaction(pudev);

                USB_DOEPxINTF((uint16_t)endp_num) = DOEPINTF_STPF;
            }

            /* back to back setup packets received */
            if (out_endp_intr & DOEPINTF_BTBSTP) {
                USB_DOEPxINTF((uint16_t)endp_num) = DOEPINTF_BTBSTP;
            }
        }

        endp_num ++;
        endp_intr >>= 1;
    }

    return 1U;
}
```

在 OUT 端点中断处理函数中，又可根据端点中断标志寄存器 out_endp_intr 进行判断产生 OUT 端点中断的事件，具体可分为：传输完成中断处理、端点禁用中断处理、SETUP 令牌包完成中断处理、收到连续的 SETUP 令牌包处理。产生相应的 OUT 端点中断事件后，根据轮询中断标志位进入不同的中断处理函数，相关中断处理函数与 USBD 的相关中断处理函数相同，读者可参考阅读。

IN 端点的中断处理函数如*图 1-15. IN 端点中断处理函数*所示。

图 1-15. IN 端点中断处理函数

```
static uint32_t usbd_intf_inep(usb_core_handle_struct *pudev)
{
    uint8_t endp_num = 0U;
    uint32_t endp_intr = 0U;

    __IO uint32_t in_endp_intr = 0U;

    /* get all in endpoints which have interrupts */
    USB_DAIEP_INTR_READ(endp_intr);

    while (endp_intr) {
        if (endp_intr & 0x1U) {
            USB_DIEP_INTR_READ(in_endp_intr, (uint16_t)endp_num);

            if (in_endp_intr & DIEPINTF_TF) {
                /* disable the fifo empty interrupt for the endpoint */
                USB_DIEPFEINTEN &= ~(0x1U << endp_num);

                USB_DIEPxINTF((uint16_t)endp_num) = DIEPINTF_TF;

                /* data transmittion is completed */
                usbd_in_transaction(pudev, endp_num);
            }

            if (in_endp_intr & DIEPINTF_CITO) {
                USB_DIEPxINTF((uint16_t)endp_num) = DIEPINTF_CITO;
            }

            if (in_endp_intr & DIEPINTF_IEPNE) {
                USB_DIEPxINTF((uint16_t)endp_num) = DIEPINTF_IEPNE;
            }

            if (in_endp_intr & DIEPINTF_EPDIS) {
                USB_DIEPxINTF((uint16_t)endp_num) = DIEPINTF_EPDIS;
            }

            if (in_endp_intr & DIEPINTF_TXFE) {
                usbd_emptytxfifo_write(pudev, endp_num);
                USB_DIEPxINTF((uint16_t)endp_num) = DIEPINTF_TXFE;
            }
        }

        endp_num ++;
        endp_intr >>= 1;
    }

    return 1U;
}
```

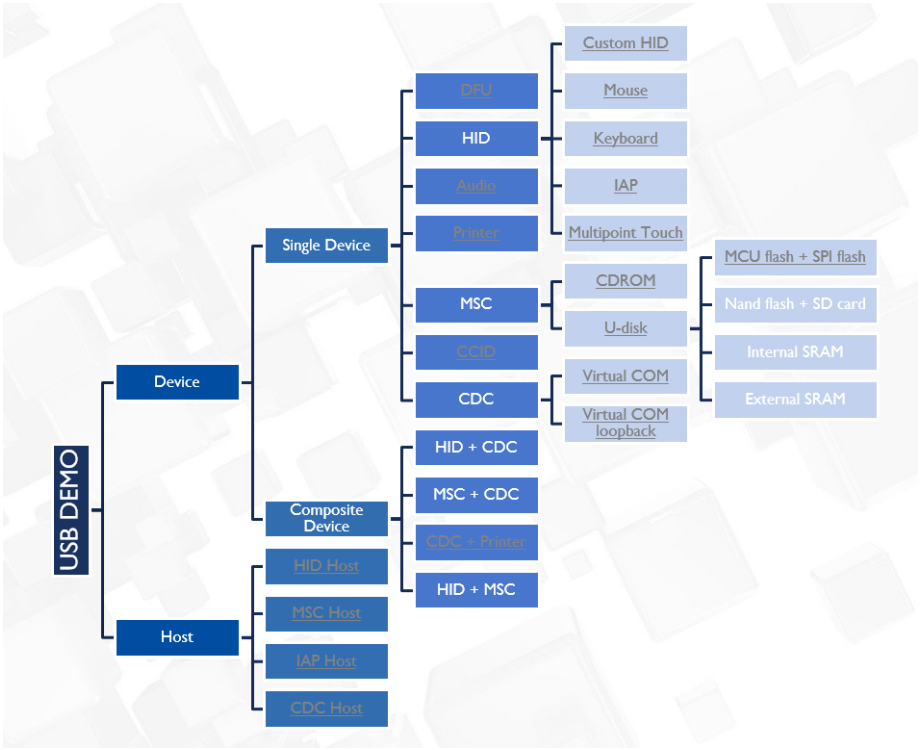
在 IN 端点中断处理函数中，又可根据端点中断标志寄存器 in_endp_intr 进行判断产生 IN 端点中断的事件，具体可分为：传输完成中断处理、超时中断处理、IN 端点 NAK 有效中断处理、端点禁用中断处理、发送 FIFO 空中断处理。产生相应的 IN 端点中断事件后，根据轮询中断标志位进入不同的中断处理函数。

相关中断处理函数与 USBD 的相关中断处理函数相同，读者可参考阅读。

1.6. USB 例程

1.6.1. USB 例程概述

图 1-16. USBFS 例程示意图



如*图 1-16. USBFS 例程示意图*所示，USBFS 的 DEMO 分为主机和设备两种，其中主机包含 HID 主机、MSC 主机、IAP 主机和 CDC 主机。设备分为复合设备和单设备，其中复合设备包含 HID+CDC 复合设备、MSC+CDC 复合设备、CDC+Printer 复合设备和 HID+MSC 复合设备。单设备按照应用协议依次分为 DFU 设备、HID 设备、Audio 设备、Printer 设备、MSC 设备、CCID 设备和 CDC 设备。HID 设备根据具体应用情景分为自定义 HID 设备、鼠标、键盘、IAP 设备和多点触摸设备。MSC 设备分为只读的 CDROM 设备和可读可写的 U 盘设备，U 盘的存储介质分为 MCU Flash、SPI Flash、Nand Flash、SD Card、MCU SRAM 和外部 SRAM。在上述例程中，下文将选择 MSC 主机例程和 HID 键盘设备例程简要介绍主机和设备的工作原理和运行结果。

1.6.2. MSC 主机

应用概述

在本应用中，简要介绍 USBFS 作为主机对 U 盘进行读写操作的功能。当 U-Disk 经 OTG 线连接到 GD32F450i 开发板上，USBFS 将对其进行枚举，枚举完成后，进入用户应用程序，本应用可以读取 U 盘文件目录列表和写文件操作。

主函数介绍

本应用的主函数如*图 1-17. USBFS 主机接 U 盘例程主函数*所示，在 while(1)之前的代码是 USBFS 主机初始化部分，初始化完成之后，程序将进入主循环中，并启动 USBFS 主机状态机。主机状态机将在下节中说明。

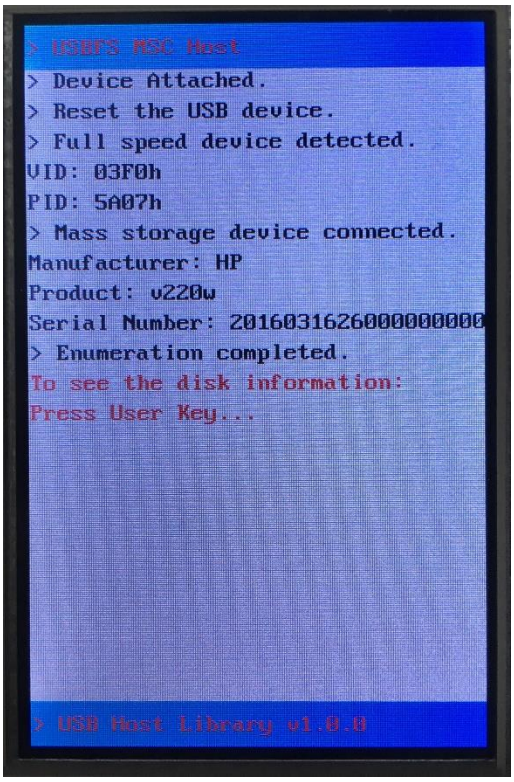
图 1-17. USBFS 主机接 U 盘例程主函数

```
int main(void)
{
    /* config system clock */
    system_clock_config();

    /* usb gpio init */
    usb_gpio_init();

    /* usb rcu init */
    usb_rcu_init();

    /* usb_timer init */
    usb_time_init();

    /* configure GPIO pin used for switching VBUS power */
    usb_hwp_vbus_config(&usb_core_dev);

    /* host de-initializations */
    usbh_deinit(&usb_core_dev, &usb_host, &usbh_state_core);

    /* start the USB core */
    hcd_init(&usb_core_dev,

#ifdef USE_USBFS
    USB_FS_CORE_ID
#elif defined(USE_USBHS)
    USB_HS_CORE_ID
#endif /* USE_USBFS */
             );

    /* init usr call back */
    usb_host.usr_cb->init();

    /* enable interrupts */
    usb_hwp_interrupt_enable(&usb_core_dev);

    while (1) {
        host_state_polling_fun(&usb_core_dev, &usb_host, &usbh_state_core);
    }
}
```

MSC 主机接 U 盘实验结果

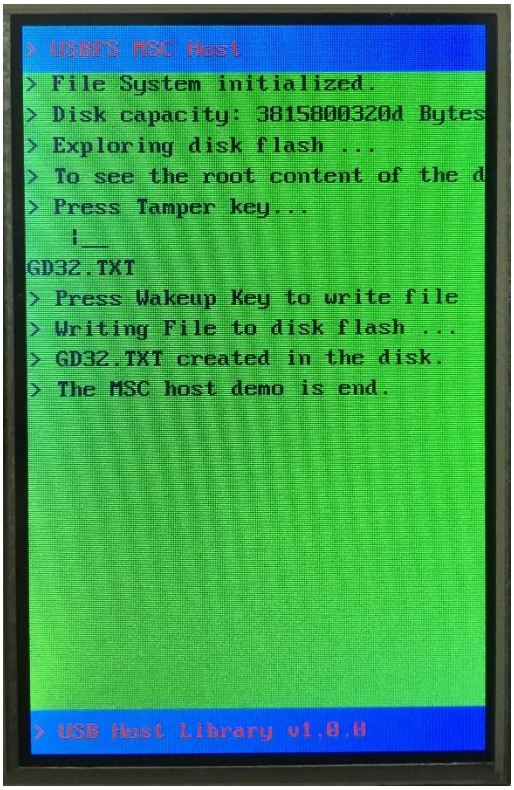
将 MSC_Host/GD32450I_EVAL_Fullspeed 工程代码下载到 GD32F450i 开发板上，并利用 OTG 电缆线将 U 盘连接到 USB_FS 接口，运行代码后，首先用户将观察到 U 盘枚举信息，如*图 1-18. USBFS 主机接 U 盘枚举信息*所示。

图 1-18. USBFS 主机接 U 盘枚举信息



首先按下 User 按键将会看到 U 盘信息；之后按下 Tamper 按键将会看到 U 盘根目录内容；然后按下 Wakeup 按键将会向 U 盘写入文件；最后用户将会看到 MSC 主机示例结束的信息，具体现象如*图 1-19. USBFS 主机接 U 盘实例现象*所示。

图 1-19. USBFS 主机接 U 盘实例现象



1.6.3. HID 键盘设备

应用概述

在本应用中，简要介绍 USBFS 设备，通过 USB 连接线与主机相连，在设备完成枚举后，主机将 USBFS 设备识别为键盘。通过按压开发板的按键，可以在主机打印出相应的字符，如按 Tamper 键输出‘a’，按 Wakeup 键输出‘b’，按 User 键输出‘c’。

主函数及初始化

USBFS 接口模块工作于设备模式下，其实现代码与 USBD 设备接口模块类似，可参考阅读。本应用举例将 USBFS 设备枚举为一个键盘，其主函数如*图 1-20. USBFS 设备主函数*所示。

图 1-20. USBFS 设备主函数

```
int main(void)
{
    /* configure USB GPIO */
    usb_gpio_config();

    /* configure USB clock */
    usb_clock_config();

    /* configure key */
    key_config();

    /* USB device stack configure */
    usbd_init(&usbhs_core_dev,
#ifdef USE_USBFS
    USB_FS_CORE_ID
#elif defined(USE_USBHS)
    USB_HS_CORE_ID
#endif
    );

    /* USB interrupt configure */
    usb_interrupt_config();

#ifdef USE_IRC48M
    /* CTC peripheral clock enable */
    rcu_periph_clock_enable(RCU_CTC);

    /* CTC config */
    ctc_config();

    while(ctc_flag_get(CTC_FLAG_CKOK) == RESET) {
    }
#endif
```

```
    /* check if USB device is enumerated successfully */
    while (usbhs_core_dev.dev.status != USB_STATUS_CONFIGURED) {
    }

    while (1) {
        if (prev_transfer_complete) {
            switch (key_state()) {
            case CHAR_A:
                key_buffer[2] = 0x04U;
                break;
            case CHAR_B:
                key_buffer[2] = 0x05U;
                break;
            case CHAR_C:
                key_buffer[2] = 0x06U;
                break;
            default:
                break;
            }

            if (key_buffer[2] != 0U) {
                usbd_hid_report_send (&usbhs_core_dev, key_buffer, 8U);
            }
        }
    }
}
```

在初始化过程中，并未进行 DP 引脚的上拉，这是 USBFS 设备与 USBD 设备的一个区别，将在下节中展开介绍。

DP 引脚的上拉和断开

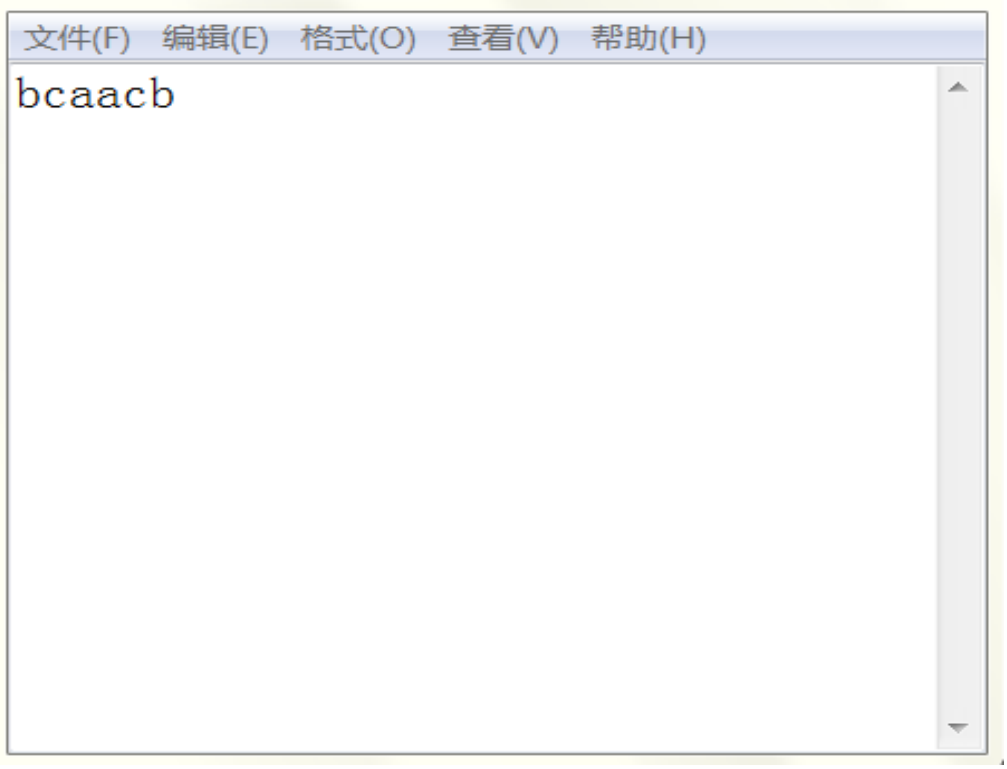
USBFS 设备 DP 引脚的上拉通过硬件监测自动实现，软件无需操作。当 VBUS 引脚检测到 B 类会话有效电平时，USBFS 接口模块会自动连接 DP 引脚的上拉电阻来向主机发送全速设备连接的信号，并触发设备中断(SESIF@ USBFS_GINTF)。

当 MCU 检测到 VBUS 引脚电平低于 B 类会话有效电平，USBFS 接口模块将会自动断开 USB 连接，并触发设备中断(SESEND@USBFS_GOTGINTF)。另外，也可通过软件断开设备连接，设置设备控制寄存器的软件断开位(SD@USBFS_DCTL)，将使能软件断开特征，之后 USBFS 接口模块将移除 DP 引脚的上拉电阻，将引起设备断开检测中断，从而使主机即使在 USB 电缆依然连接时，仍然可以识别到一个设备断开事件。

HID 键盘设备实验结果

将 HID_Keyboard 工程代码下载到 GD32F450i 开发板上，并采用 USB 电缆线将 USBFS 接口连接 PC 主机，运行代码后，如*图 1-21. HID 键盘设备实验结果*所示，按下 Wakeup 键，输出‘b’；按下 Tamper 键，输出‘a’；按下 User 键，输出‘c’。

图 1-21. HID 键盘设备实验结果



2. 版本历史

表 2-1. 版本历史

版本号	描述	日期
1.0	初稿发布	2019 年 4 月 1 日

Important Notice

This document is the property of GigaDevice Semiconductor Inc. and its subsidiaries (the "Company"). This document, including any product of the Company described in this document (the “Product”), is owned by the Company under the intellectual property laws and treaties of the People’s Republic of China and other jurisdictions worldwide. The Company reserves all rights under such laws and treaties and does not grant any license under its patents, copyrights, trademarks, or other intellectual property rights. The names and brands of third party referred thereto (if any) are the property of their respective owner and referred to for identification purposes only.

The Company makes no warranty of any kind, express or implied, with regard to this document or any Product, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The Company does not assume any liability arising out of the application or use of any Product described in this document. Any information provided in this document is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Except for customized products which has been expressly identified in the applicable agreement, the Products are designed, developed, and/or manufactured for ordinary business, industrial, personal, and/or household applications only. The Products are not designed, intended, or authorized for use as components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, atomic energy control instruments, combustion control instruments, airplane or spaceship instruments, transportation instruments, traffic signal instruments, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or Product could cause personal injury, death, property or environmental damage ("Unintended Uses"). Customers shall take any and all actions to ensure using and selling the Products in accordance with the applicable laws and regulations. The Company is not liable, in whole or in part, and customers shall and hereby do release the Company as well as it’s suppliers and/or distributors from any claim, damage, or other liability arising from or related to all Unintended Uses of the Products. Customers shall indemnify and hold the Company as well as it’s suppliers and/or distributors harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of the Products.

Information in this document is provided solely in connection with the Products. The Company reserves the right to make changes, corrections, modifications or improvements to this document and Products and services described herein at any time, without notice.