GigaDevice Semiconductor Inc.

ARM® Cortex™ 32-bit MCU

USBFS/HS Firmware Library User Guide

Revision 1.0

(Apr. 2019)

Table of Contents

List of Figures

List of Table

1. USBFS/USBHS

1.1. Brief

Basing on universal serial bus full speed / high speed interface structure, this article introduce the interface principle and firmware structure of USBFS/USBHS, briefly describe the firmware function performance, host state machine and USB interrupt mechanism. Taking the MSC host and HID device for example, the article show the USB host and device demo implementation.

1.2. USBFS/USBHS principle brief

At present, GD32F105/107/205/207/305/307/350/4xx/E103 series have USBFS interface module, the framework of all series MCU USBFS is same. This article introduce principle of the most advanced MCU series of GD, GD32F4xx, which have both USBFS and USBHS.

As device, USBFS only support FS (12Mbps) device, meanwhile, as host, USBFS support FS (12Mbps) and LS (1.5Mbps) device. To be dual role device, USBFS could be configured as host or device. In OTG mode, the switch between host and device should conform to SRP protocol and HNP protocol.

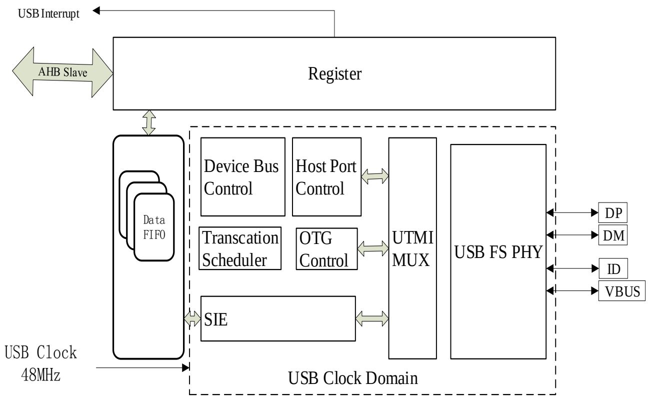
USBHS not only support all common characteristic and function, which USBFS support, but also support USB2.0 HS (480Mbps) host and device.

1.2.1. USBFS interface principle

USBFS structure

As the block diagram of ***Figure 1-1. USBFS structure diagram*** shown, Cortex-M core read and write USBFS module register through AHB slave bus. USBFS register generate USB interrupt for NVIC. There is 1.25KB data FIFO for USBFS, which is connected to SIE. In device mode, FIFO is compose of one receive FIFO and multiple FIFOs. Every IN endpoint own its transmit FIFO, and all OUT endpoints share receive FIFO. In host mode, data FIFO divide into three parts, one is receive FIFO which is used to receive data, one is periodical transmit FIFO which is used to transmit periodical data, and the other one is non-periodical transmit FIFO which is used to transmit non-periodical data. All IN channel share receive FIFO, all periodical OUT channel share periodical transmit FIFO and all non-periodical OUT channel share non-periodical transmit FIFO. Data FIFO is connected to SIE. USB clock domain comes from internal 48MHz clock. Through UIMI, USBFS controller is connected to USBFS PHY, which is used to realize USB communication and compose of USB receiver and USB interface circuit.

Figure 1-1. USBFS structure diagram



USBFS module character

Table 1-1. USB_FS characteristic

Host characteristic	Device characteristic
8 host channels	4 bidirectional endpoints (include endpoint 0)
dedicated TX_FIFO 1, periodical TX_FIFO：storage synchronization and interrupt transfer data 2, non-periodical TX_FIFO: storage bulk and control transfer data	Four independent TX_FIFO corresponding to its IN endpoint
One shared RX_FIFO to receive data	one shared RX_FIFO
Support power for connected device through external power chip	support software disconnect
Support USB 2.0 FS/LS host mode	Support USB 2.0 FS device mode
Two request queue: 1,periodical queue: manage utmost 8 synchronization and interrupt transfer request 2,non-periodical queue: manage utmost 8 bulk and control transfer request	
Support four kinds of transfer: control transfer, bulk transfer, interrupt transfer and synchronization transfer	
Support HNP protocol , SRP protocol and OTG protocol	

USBFS host mode

There is four condition when USBFS is identified as host

1, insert USB A port when USBFS interface is configured as default.

2, insert USB B port, then USBFS interface module is HNP switched.

3, HNP bit of USB configuration register is write to be 0.

4, force host mode bit of USB global configuration register is set, just only treat USB as host, in this condition, ignore ID line state and enable integrate pull-down resistor in DP and DM line

USBFS host or device connection diagram is shown in ***Figure 1-2. USBFS connect diagram as host or device.*** If USBFS is configured as host, it should supply 5V power, in this case, 5V power is external USB device power supply.

USBFS device mode

There is four conditions when USBFS is identified as device

1, insert USB B port when USBFS interface is configured as default.

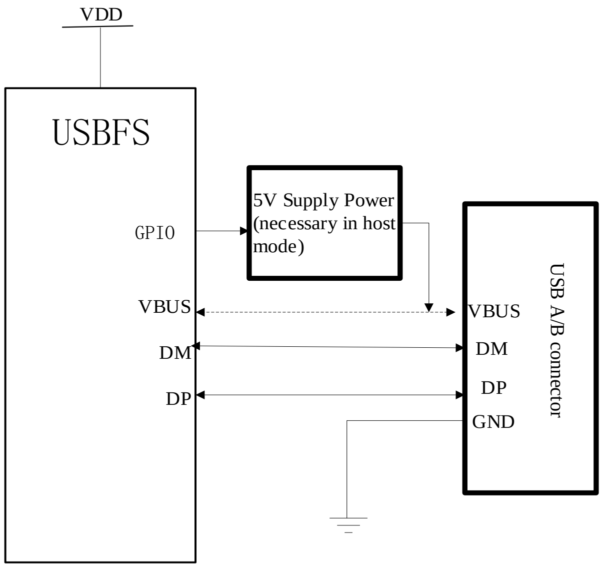
2, insert USB A port, then USBFS interface module is HNP switched.

3, HNP bit of USB configuration register is write to be 0.

4, force device mode bit of USB global configuration register is set, just only treat USB as device.

USBFS connection diagram is shown as below. If USBFS is host, 5V power supply is unnecessary, VBUS pin detect VBUS voltage, detect whether host is disconnect. DP and DM line represent differential signal line.

Figure 1-2. USBFS connect diagram as host or device

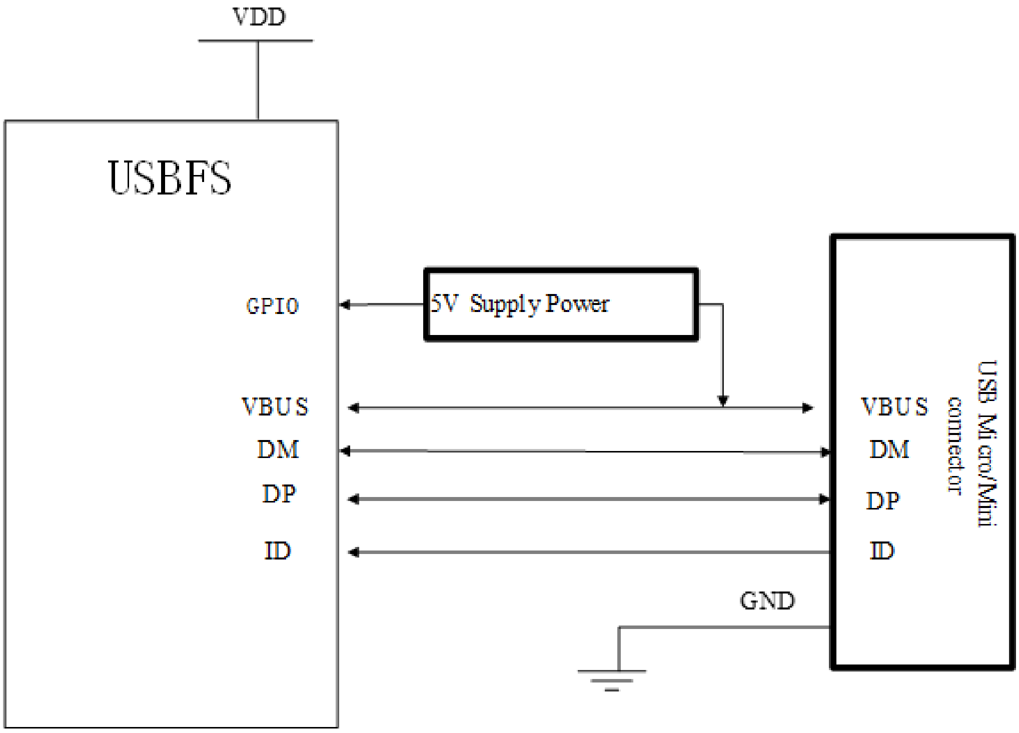


USBFS OTG mode

USBFS OTG connection diagram is shown in ***Figure 1-3. USBFS connection diagram as OTG device***. Except common four wires in USB, ID wire, which indicates role of USBFS, is necessary. If USB cable B port is connected, its ID wire is floating, high level of ID wire will be detected, and USBFS is configured as slave device default. If USB cable A port is connected, its ID wire is on the ground, USBFS will generate ID wire status change interrupt to initial host software, and switch to host automatically.

GPIO port is used to control generating 5V power supply, and this power is input from external, to be USBFS A device, supply power for other device and MCU. VBUS pin is used to detect VBUS voltage.

Figure 1-3. USBFS connection diagram as OTG device



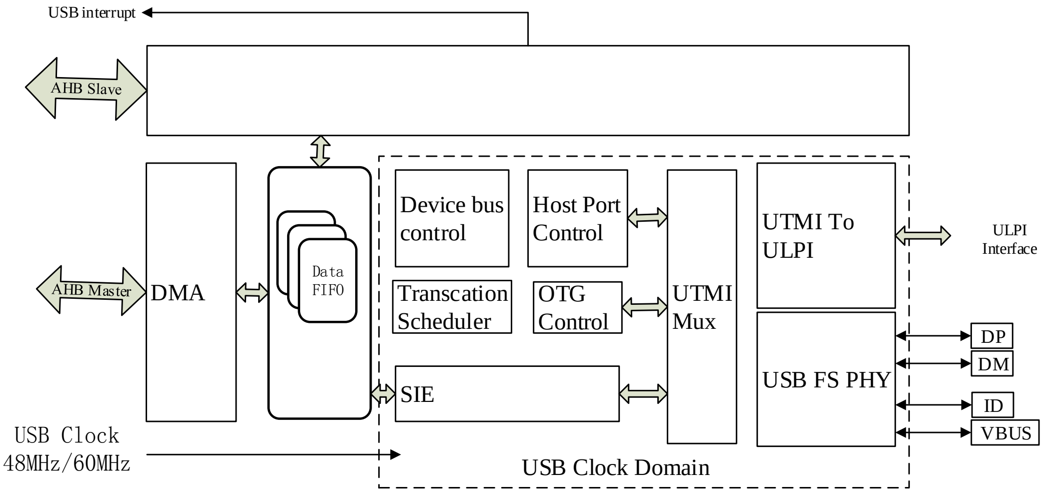
1.2.2. USBHS interface principle

USBHS interface module include an embedded USBFS PHY, the function of this PHY is stronger than the function of USBFS module, it could support 6 bidirectional device endpoints, 12 host channels and 4K bytes FIFO, thus, USBHS realize full speed and low speed by USBFS PHY. Moreover, if ULPI interface is connected to USBHS, it could realize USB high speed communication, its data transfer rate is up to 480Mbps. When USBHS is in high speed host mode, USBHS support hub connection. In addition, there is DMA engine in USBHS, which could accelerate data transfer rate between USBHS and system.

USBHS structure

USBHS structure diagram is shown in ***Figure 1-4. USBHS structure diagram***. Corresponding to USBFS, DMA transfer engine and ULPI interface is added in USBHS structure diagram.

Figure 1-4. USBHS structure diagram

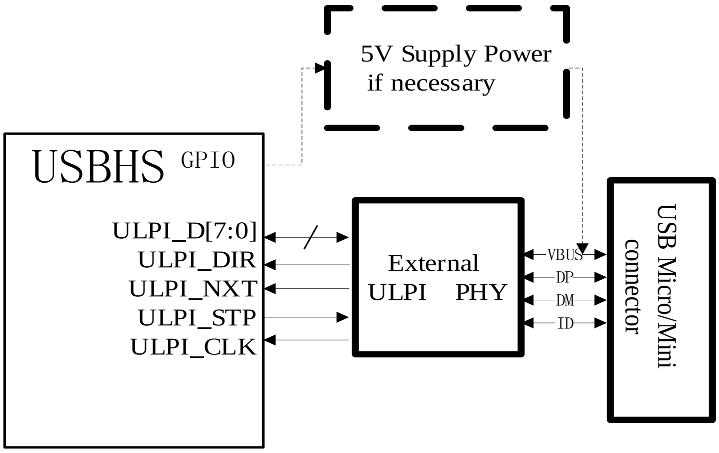


USBHS external ULPI PHY

USBHS provide an ULPI interface to external PHY, if USBHS module try to realize USBHS application, external HS ULPI PHY is necessary. Combining external ULPI PHY, USBHS support high speed host and device, meanwhile, support all modes which USBFS PHY support.

Software need clearing EMBPHY bit of USBHS_GUSBCS register to enable ULPI interface. When ULPI mode enable, 60MHz clock need be input from ULPI_CLK pin. Software could open or close 60MHz ULPI clock in RCU register.

Figure 1-5. External ULPI PHY connection diagram.

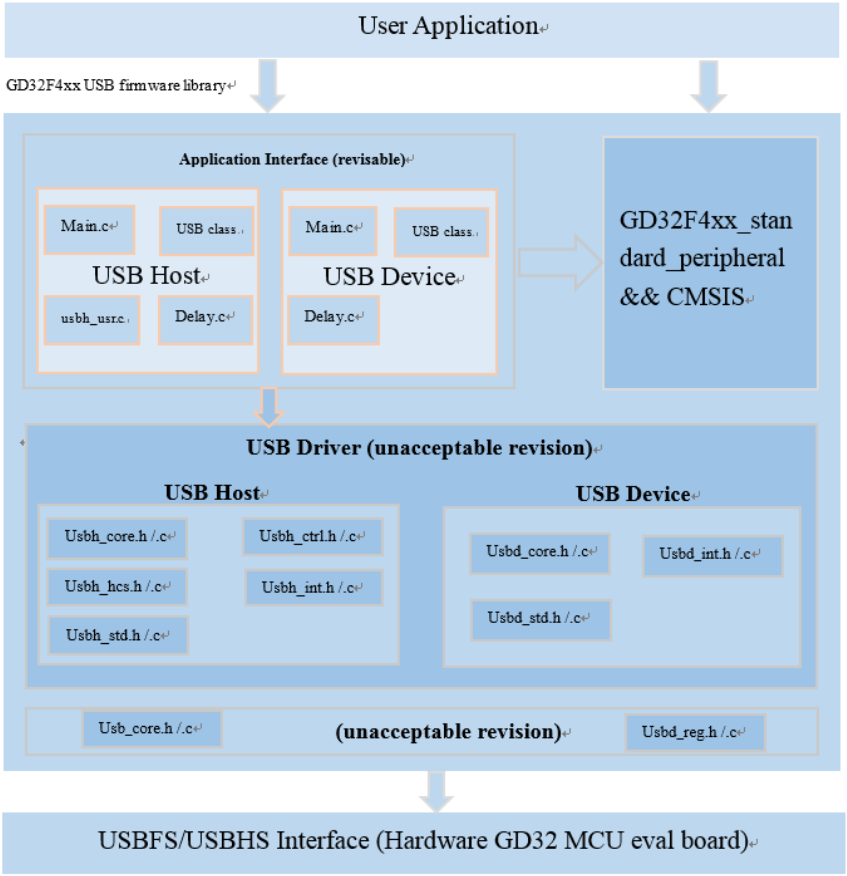


1.3. USBFS/USBHS module firmware

1.3.1. USBFS/USBHS module firmware structure

GD32F4xx series MCU interface module structure diagram is shown in ***Figure 1-6. GD32F4xx series MCU USBFS/USBHS structure***. The figure show USBFS/USBHS host and device structure. User application call GD32 USB firmware library to realize USB data communication. The bottom of structure is hardware of MCU evaluation board. There is three layers in GD32 USBFS/USBHS firmware library. The top is application interface layer, user could change. Middle layer is USB host or USB device, the bottom layer is USB drivers. Middle layer and bottom layer are called as “USB firmware library driver”, user could not modify it. USB class file, which implement USB host application class or USB application device class file, is part of application layer.

Figure 1-6. GD32F4xx series MCU USBFS/USBHS structure



1.3.2. Underlying file and function introduction

USB Driver underlying file include two files, which are shown in ***Table 1-2. USB bottom layer file table.***

Table 1-2. USB bottom layer file table

File name	Tips
usb_core.h/.c	USB core driver
usb_reg.h	USB register operation

Table 1-3. usb_core.h/.c file function table

Function name	Functional description
usb_commonint_enable	enable common interrupt
usb_core_reset	USB core reset
usb_fifo_write	write a packet into Tx FIFO of corresponding endpoint
usb_fifo_read	read a packet from Rx FIFO of corresponding endpoint
usb_core_select	select USB core
usb_core_init	Initial USB controller register and parameter
usb_txfifo_flush	Flush a Tx FIFO and all Tx FIFO
usb_rxfifo_flush	Flush all Rx FIFO
usb_mode_set	configure operation mode(host or device)
usb_hostcore_init	Initial USB core for host mode
usb_vbus_drive	configure USB interface power
usb_hostint_enable	enable host mode interrupt
usb_port_reset	reset host channel
usb_hostchannel_init	initial host channel
usb_hostchannel_startxfer	host channel start transmit
usb_hostchannel_halt	halt channel
usb_hostchannel_ping	transmit a PING token
usb_host_stop	stop host and clear FIFOS
usb_devcore_init	Initial USB core register for device mode
usb_devint_enable	enable device mode interrupt
usb_ep0_startout	Configure endpoint 0 start receive SETUP token
usb_remotewakeup_active	start remote wakeup
usb_clock_ungate	start USB core clock
usb_device_stop	halt USB device an d clear FIFOS

1.3.3. USB_Host middle layer file and library function introduction

USB_Host middle layer include 5 files, which are shown in ***Table 1-4. USB_Host middle layer file.***

Table 1-4. USB_Host middle layer file

File name	Functional description
usbh_core.h/.c	USB host state machine handle function
usbh_ctrl.h/.c	USB host control transfer handle function
usbh_hcs.h/.c	USB open or close channel handle function

usbh_int.h/.c	USB host mode interrupt handle function
usbh_std.h/.c	USB communication handle function

Table 1-5. usbh_core.h/.c file function table

Function name	Functional description
host_state_polling_fun	host state machine polling function
host_idle_handle	HOST_IDLE state handle function
host_dev_attached_handle	HOST_DEV_ATTACHED state handle function
host_enum_handle	HOST_ENUMERATION state handle function
host_user_input_handle	HOST_USER_INPUT state handle function
host_class_request_handle	HOST_CLASS_REQUEST state handle function
host_class_handle	HOST_CLASS state handle function
host_suspended_handle	HOST_SUSPENDED state handle function
host_error_handle	HOST_ERROR state handle function
host_dev_detached_handle	HOST_DEV_DETACHED state handle function
host_detect_dev_speed_handle	HOST_DETECT_DEV_SPEED state handle function
usbh_connected	device connection interrupt callback function
usbh_disconnected	device disconnection interrupt callback function
usbh_sof	sof interrupt callback function
hcd_init	host core driver initialization
hcd_is_device_connected	check whether device is connected
hcd_urb_state_get	return latest URB state
hcd_xfer_count_get	return transfer data size
usbh_deinit	de-initialize host
scd_init	state machine core driver initialize
scd_table_regist	state machine core driver state table
scd_begin	state machine driver start
scd_state_move	state machine core driver state switch
scd_event_handle	state machine core driver event handle
scd_table_push	push current state machine into state stack
scd_table_pop	pop current state machine from state stack
class_req_state_polling_fun	device class request state machine polling function
class_state_polling_fun	device class state machine polling function
only_state_move	state switch function
goto_up_state_fun	return upper state machine

Table 1-6. usbh_ctrl.h/.c file function

Function name	Functional description
ctrl_state_polling_fun	Control transfer state machine polling function
ctrl_idle_handle	CTRL_IDLE state machine handle function
ctrl_setup_handle	CTRL_SETUP state machine handle function
ctrl_data_handle	CTRL_DATA state machine handle function
ctrl_status_handle	CTRL_STATUS state machine handle function
ctrl_error_handle	CTRL_ERROR state machine handle function
ctrl_stalled_handle	CTRL_STALLED state machine handle function
ctrl_complete_handle	CTRL_COMPLETE state machine handle function
usbh_xfer	transmit data form host port
usbh_ctltx_setup	Transmit SETUP token packet to device
hcd_submit_request	prepare channel, start once transfer

Table 1-7. usbh_hcs.h/.c file function table

Function name	Functional description
usbh_channel_open	open host channel
usbh_channel_modify	update channel
usbh_channel_alloc	configure new channel for pipe
usbh_channel_free	release USB host channel
usbh_allchannel_dealloc	release all USB host channel
usbh_freechannel_get	achieve a free host channel for configured device endpoint

Table 1-8. usbh_int.h/.c file function table

Function name	Functional description
usbh_isr	handle global host interrupt
usbh_intf_sof	Sof interrupt handle
usbh_intf_hc	handle all host channel interrupt
usbh_intf_disconnect	handle disconnect interrupt
usbh_intf_nptxfifo_empty	handle non-periodic transmit FIFO empty interrupt
usbh_intf_ptxfifo_empty	handle periodic transmit FIFO empty interrupt
usbh_intf_port	handle host port interrupt
usbh_intf_hc_out	handle out channel interrupt
usbh_intf_hc_in	handle in channel interrupt
usbh_intf_rxfifo_noempty	handle receive FIFO non empty interrupt
usbh_intf_iso_incomplete_xfer	handle Incomplete periodic transfer interrupt

Table 1-9. usbh_std.h/.c file function table

Function name	Functional description
enum_state_polling_fun	enumeration state machine polling function

enum_idle_handle	ENUM_IDLE state handle function
enum_get_full_dev_desc_handle	ENUM_GET_FULL_DEV_DESC state handle function
enum_set_addr_handle	ENUM_SET_ADDR state handle function
enum_get_cfg_desc_handle	ENUM_GET_CFG_DESC state handle function
enum_get_full_cfg_desc_handle	ENUM_GET_FULL_CFG_DESC state handle function
enum_get_mfc_string_desc_handle	ENUM_GET_MFC_STRING_DESC state handle function
enum_get_product_string_desc_handle	ENUM_GET_PRODUCT_STRING_DESC state handle function
enum_get_serialnum_string_desc_handle	ENUM_GET_SERIALNUM_STRING_DESC state handle function
enum_set_configuration_handle	ENUM_SET_CONFIGURATION state handle function
enum_dev_configured_handle	ENUM_DEV_CONFIGURED state handle function
usbh_enum_desc_get	get descriptor in host enumeration phase
usbh_enum_addr_set	configure address in host enumeration phase
usbh_enum_cfg_set	set configuration in host enumeration phase
usbh_device_desc_parse	analysis device descriptor
usbh_cfg_desc_parse	analysis device descriptor
usbh_interface_desc_parse	analysis interface descriptor
usbh_endpoint_desc_parse	analysis endpoint descriptor
usbh_string_desc_parse	analysis character string descriptor
usbh_next_desc_get	get next descriptor packet header

1.3.4. USB_Device middle layer file and library function introduction

Table 1-10. USB_Device middle layer file

Function name	Functional description
usbd_core.h/.c	USB device mode core driver
usbd_int.h/.c	USB device mode interrupt routines
usbd_std.h/.c	USB 2.0 standard handler driver

Table 1-11. usbd_core.h/.c file function table

Function name	Functional description
usbd_init	initailizes the USB device-mode handler stack
usbd_ep_init	endpoint initialization
usbd_ep_deinit	endpoint deinitialize
usbd_ep_rx	endpoint prepare to receive data
usbd_ep_tx	endpoint prepare to transmit data
usbd_status_enum usbd_ctltx	transmit data on the control endpoint

usbd_status_enum usbd_ctlrx	receive data on the control endpoint
usbd_status_enum usbd_ctlstatus_tx	transmit status on the control endpoint
usbd_status_enum usbd_ctlstatus_rx	receive status on the control endpoint
usbd_ep_stall	set an endpoint to STALL status
usbd_ep_clear_stall	clear endpoint stalled status
usbd_ep_fifo_flush	flushes the fifos
usbd_rxcount_get	get the received data length

Table 1-12. usbd_int.h/.c file function table

Function name	Functional description
usbd_isr	USB device-mode interrupts global service routine handler
usbd_intf_outep	indicates that an OUT endpoint has a pending interrupt
usbd_intf_inep	indicates that an in endpoint has a pending interrupt
usbd_intf_earlysuspend	indicates that early suspend state has been detected on the USB
usbd_intf_suspend	indicates that suspend state has been detected on the USB
usbd_intf_resume	indicates that the USB controller has detected a resume or remote Wake-up sequence
usbd_intf_sof	handle the SOF interrupts
usbd_intf_rxfifo	handle the rx FIFO non-empty interrupt
usbd_intf_reset	handle USB reset interrupt
usbd_intf_enumfinish	handle enumeration finish interrupt
usbd_intf_isoinincomplete	handle the ISO in incomplete interrupt
usbd_intf_isooutincomplete	handle the ISO OUT incomplete interrupt
usbd_emptytxfifo_write	check FIFO for the next packet to be loaded
usbd_intf_sessionrequest	indicates that the USB_OTG controller has detected a connection
usbd_intf_otg	indicates that the USB_OTG controller has detected an OTG event

Table 1-13. usbd_std.h/.c file function table

Function name	Functional description
usbd_setup_transaction	USB setup stage processing
usbd_out_transaction	data out stage processing
usbd_in_transaction	data in stage processing
usbd_standard_request	handle USB standard device request
usbd_device_class_request	handle USB device class request
usbd_vendor_request	handle USB vendor request
usbd_reserved	no operation, just for reserved

usbd_device_descriptor_get	get the device descriptor
usbd_configuration_descriptor_get	get the configuration descriptor
usbd_string_descriptor_get	get string descriptor
usbd_getstatus	handle Get_Status request
usbd_clrfeature	handle USB Clear_Feature request
usbd_setfeature	handle USB Set_Feature request
usbd_setaddress	handle USB Set_Address request
usbd_getdescriptor	handle USB Get_Descriptor request
usbd_setdescriptor	handle USB Set_Descriptor request
usbd_getconfig	handle USB Get_Configuration request
usbd_setconfig	handle USB Set_Configuration request
usbd_getinterface	handle USB Get_Interface request
usbd_setinterface	handle USB Set_Interface request
usbd_synchframe	handle USB SynchFrame request
usbd_setup_request_parse	decode setup data packet
usbd_enum_error	handle USB low level error event

1.3.5. Application interface layer file and library function introduction

Application layer include four files, which are shown in ***Table 1-14. Application interface layer file table***

Table 1-14. Application interface layer file table

File name	Introduction
main.c	**main application program interface**
usbh_usr.c	**user application program interface**
usb_delay.c	**delay function interface**
USB Class	**device class application program interface**

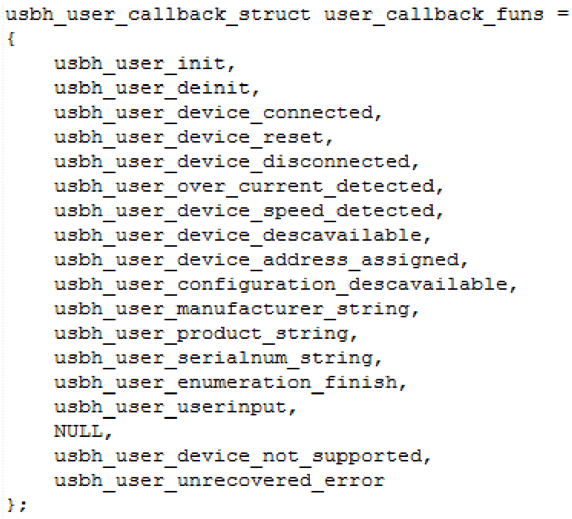
Main.c file mainly realize main application program interface function, it is shown in **Table 1-15. Main.c file function table**.

Table 1-15. Main.c file function table

Function name	Functional description
main	**application program main function**
usb_rcu_init	**USB RCU initialization**
usb_gpio_init	**USB GPIO initialization**
usb_hwp_interrupt_enable	**configure USB global interrupt**
usb_hwp_vbus_drive	**drive VBUS signal line via GPIO**
usb_hwp_vbus_config	**configure VBUS pin**

usbh_usr.c file contains user callback function structure, which is shown in ***Figure 1-7. User callback function structure***.

Figure 1-7. User callback function structure



```
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
};
```

Table 1-16. usr_cb user callback function structure function

Function name	Functional description
init	initialize user operation in host mode
deinit	configure user as default
device_connected	user operation of USB connection
device_reset	user operation of device resetting
device_disconnected	user operation of USB disconnection
over_current_detected	user operation of device overloading
device_speed_detected	user operation of detecting device speed
device_desc_available	user operation when device descriptor is available
device_address_set	user operation when device is successfully configured
configuration_desc_available	user operation when configuration descriptor is available
manufacturer_string	user operation when vendor string is available
product_string	user operation when product string is available
serial_num_string	user operation when serial number is exist
enumeration_finish	user operation when enumeration is accomplished
user_input	user operation when entering user state
user_application	user application program
device_not_supported	user operation when device is not supported
unrecovered_error	user operation when unrecoverable error happen

Table 1-17. usb_delay.c file function table

Function name	Functional description
usb_time_init	utilize timer 2 initialize delay unit
usb_udelay	delay function in microseconds
usb_mdelay	delay function in millisecond
hwp_delay	hardware delay function

hwp_time_set	**hardware timer initialization**

USB class include host class function file, USB protocol support serval class, as shown in *Table 1-18. HID/MSC host class library function*, HID and MSC host class function file would be introduced.

Table 1-18. HID/MSC host class library function

Device class	File name	Function name	Description
HID host class	usbh_hid_core.h/c	hid_req_state_polling_fun	HID request state machine polling function
		hid_state_polling_fun	HID state machine polling function
		hid_clear_feature	HID clear characteristic
		usbh_hid_desc_parse	analysis HID descriptor
		usbh_hid_interface_init	initialize HID class interface
		usbh_hid_interface_deinit	recover default configuration for host channel of HID class
		hid_req_set_idle	set idle state
		hid_req_set_protocol	set protocol state
	Usbh_hid_keybd.h/c	keybrd_decode	analysis keyboard keys
	usbh_hid_mouse.h/c	mouse_init	initialize mouse state
		mouse_decode	analysis mouse data
		hid_mouse_button_pressed	analysis mouse button press function
		hid_mouse_button_released	analysis mouse button release function
		hid_mouse_update_position	update mouse position function
MSC host class	usbh_msc_bot.h/c	usbh_msc_init	initialize MSC parameters
		usbh_msc_handle_botxfer	handle BOT transfer state
		usbh_msc_bot_abort	handle abort error for STALL state
		usbh_msc_decode_csw	decode CSW command block packet
	usbh_msc_core.h/c	usbh_msc_interface_init	initialize MSC interface
		usbh_msc_interface_deinit	recover interface default state
		msc_req_state_polling_fun	handle MSC request polling function
		msc_state_polling_fun	MSC state polling function
		usbh_msc_max_lun_get	get MSC max logic unit
		usbh_msc_error_handle	handle MSC error event
		usbh_clear_feature	clear or forbidden feature
	usbh_msc_sc	usbh_msc_test_unit_ready	transmit test unit ready

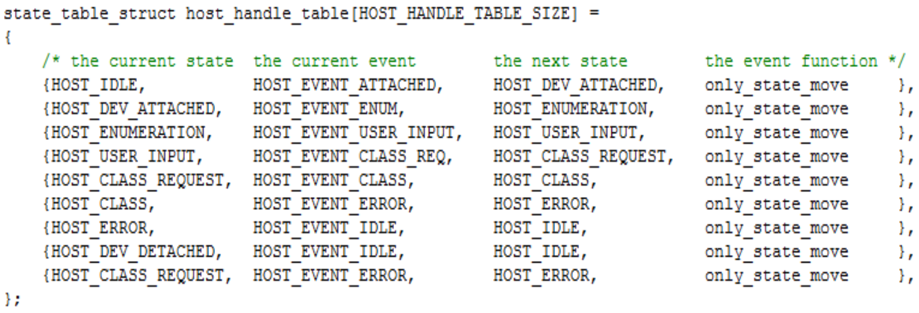
	si.c		commamd to device
		usbh_msc_read_capacity10	transmit read capacity 10 command to device
		usbh_msc_mode_sense6	transmit mode sense 6 command to device
		usbh_msc_request_sense	transmit request sense command to device
		usbh_msc_write10	transmit write 10 command to device
		usbh_msc_read10	transmit read 10 command to device
	usbh_msc_fatfs.h/c	disk_initialize	initialize disk driver
		disk_status	get disk status
		disk_read	read disk section
		disk_write	write disk section
		disk_ioctl	IO control function

1.4. USBFS host state machine

USBFS host state machine apply state machine nesting and state machine lookup table method. USBFS host machine include host machine handle (usbh_core.c), enumeration state machine handle (usbh_std.c) and control transfer state machine handle (usbh_ctrl.c).

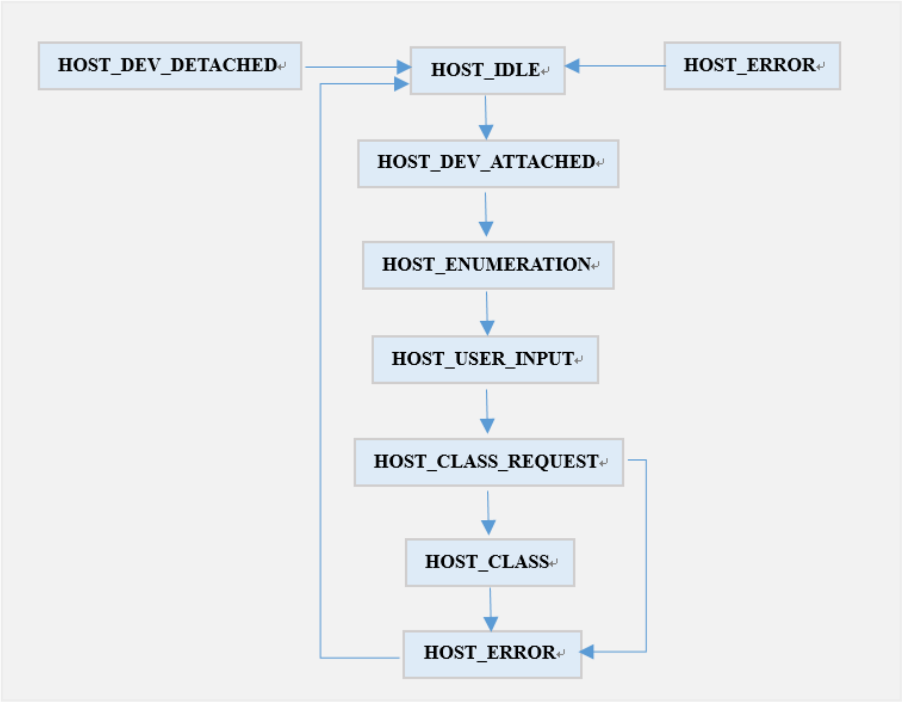
USBFS host state lookup table is shown as ***Figure 1-8. USBFS host state machine lookup table***, firstly, introduce the methods to lookup table, the table is consist of nine items. Every item, which contain four parts, is state switch. The first part is current status, the second part is current received event, the third part is the next status to which will switch, and the fourth part is event handle function which the status switch need. It means that, in current status, if receive current corresponding event, through lookup table and execute event function, switch to next status. It is easily to understand state machine lookup table.

Figure 1-8. USBFS host state machine lookup table



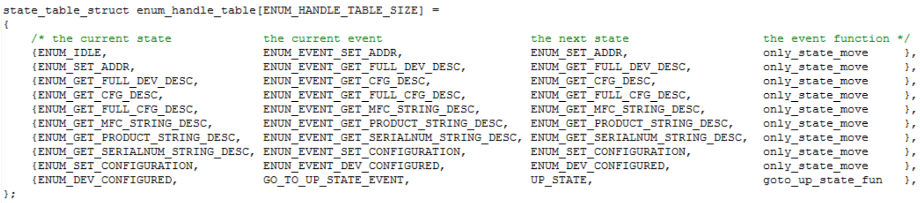
```
state_table_struct host_handle_table[HOST_HANDLE_TABLE_SIZE] =
{
    /* the current state  the current event        the next state       the event function */
    {HOST_IDLE,           HOST_EVENT_ATTACHED,     HOST_DEV_ATTACHED,   only_state_move     },
    {HOST_DEV_ATTACHED,   HOST_EVENT_ENUM,         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     },
};
```

Figure 1-9. USBFS host state machine flow diagram



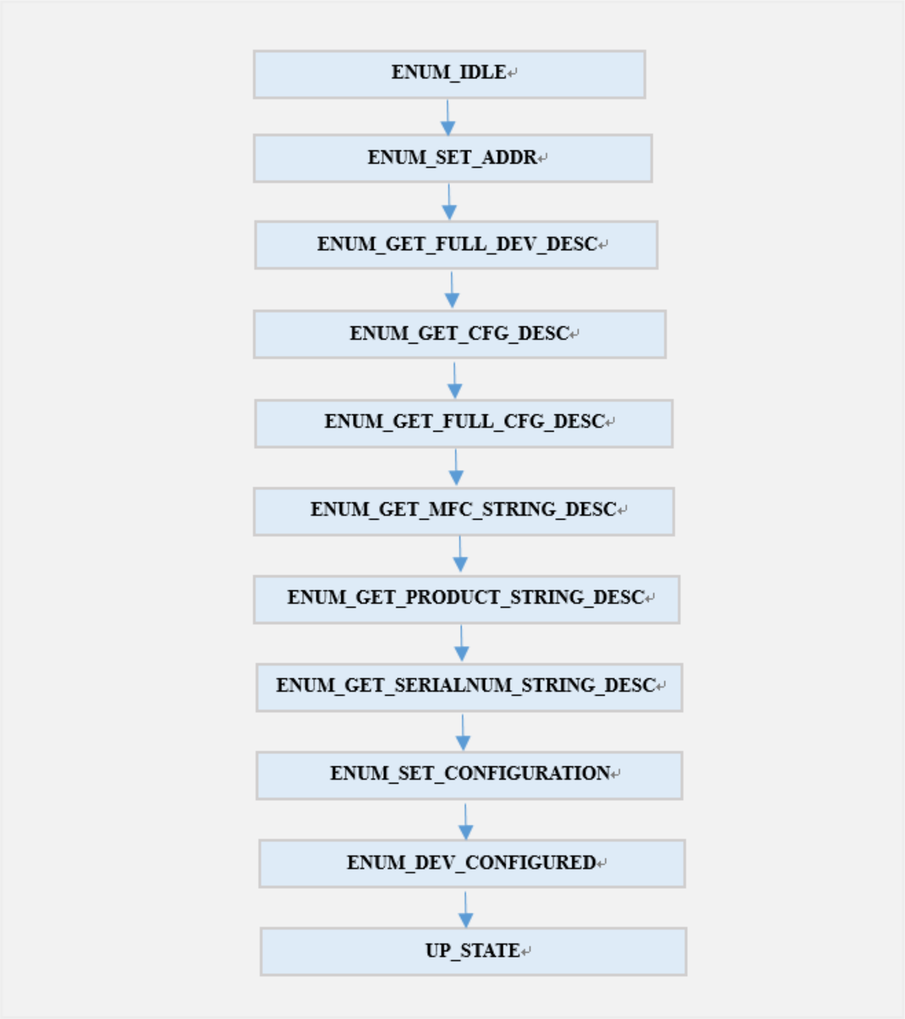
According to lookup state machine table, in HOST_DEV_ATTACHED status, if receive HOST_EVENT_ENUM event, then enter HOST_ENUMERATION status, so as to handle enumeration state machine, which is shown in ***Figure 1-10. Enumeration state machine handle lookup table a***nd ***Figure 1-11. Enumeration state machine flow diagram***. In processing of switch host state machine to enumeration state machine, function scd_table_push ( ) save last state machine to state machine stack, so as to return upper state machine and continue previous operation.

Figure 1-10. Enumeration state machine handle lookup table



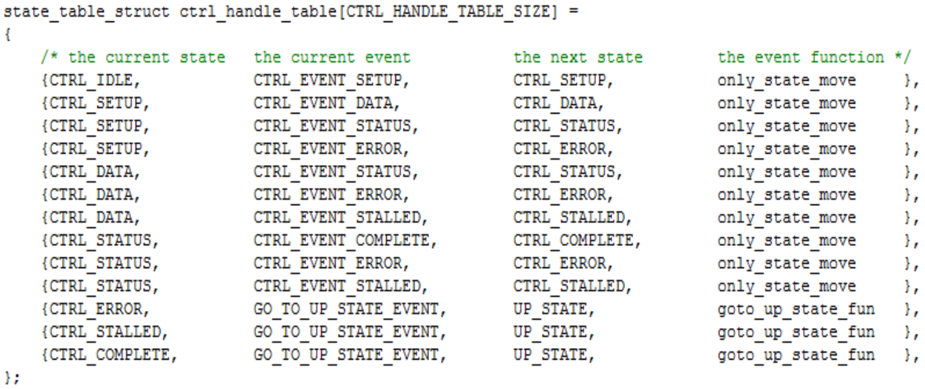
```
state_table_struct enum_handle_table[ENUM_HANDLE_TABLE_SIZE] =
{
    /* the current state             the current event                    the next state                   the event function */
    {ENUM_IDLE,                      ENUM_EVENT_SET_ADDR,                 ENUM_SET_ADDR,                   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   },
};
```

Figure 1-11. Enumeration state machine flow diagram



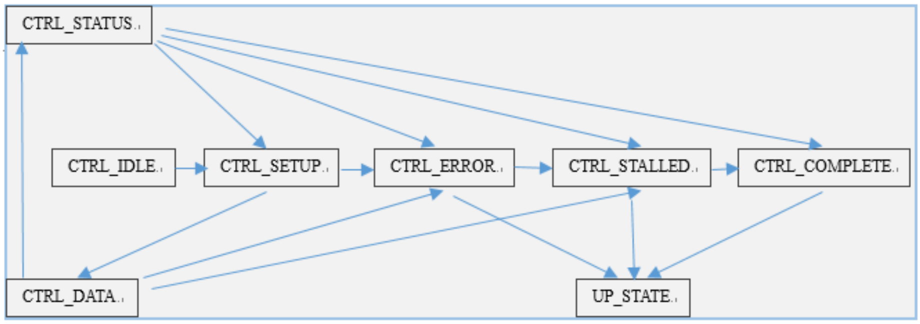
According to lookup enumeration state machine table, in ENUM_DEV_CONFIGURED status, if receive UP_STATE event, it indicate enumeration is completed, function scd_table_pop () return to upper state machine. Moreover, in the processing of enumeration, it is necessary to adopt control transfer, continue to enter control transfer state machine. Control transfer state machine lookup table is shown in ***Figure 1-12. Control transfer state machine handle lookup table***, control transfer state machine flow diagram is shown in ***Figure 1-13. Control transfer state machine flow diagram.***

Figure 1-12. Control transfer state machine handle lookup table



```
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   },
};
```

Figure 1-13. Control transfer state machine flow diagram



In the processing of control transfer, initial status is CTRL_IDLE, once control completed, the status switch to CTRL_COMPLETE, then return upper state machine.

1.5. Interrupt handle

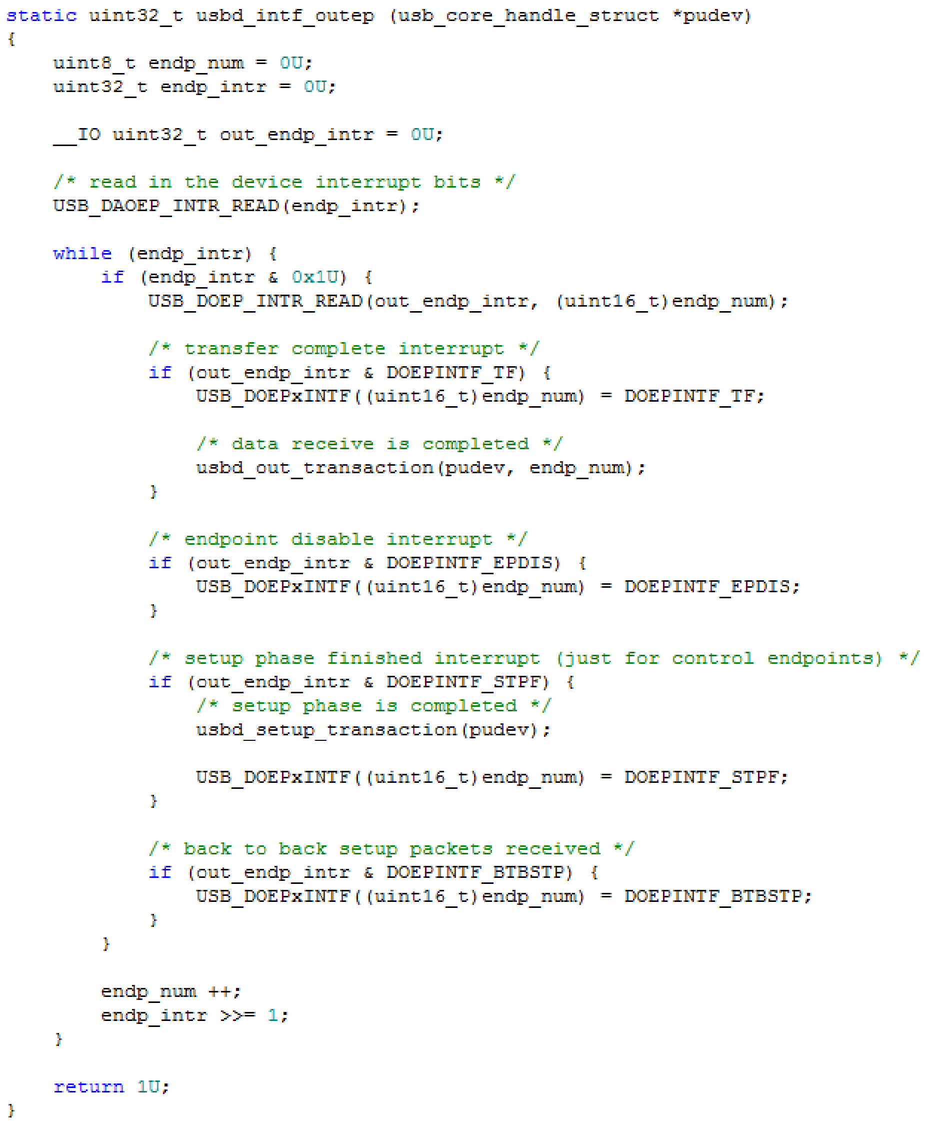
Interrupt of USBD module contain low priority interrupt and high priority interrupt. Commonly, IN and OUT transaction transfer, which are distinguished by IFR_DIR flag, would been handled in low priority interrupt. There are two different interrupts for IN and OUT transaction of USBFS, whose respective flag of interrupt is GINTF_IEPIF and GINTF_OEPIF, which are shown in ***Figure 1-13. Control transfer state machine flow diagram***. OUT endpoint interrupt handle function is shown in ***Figure 1-14. OUT endpoint interrupt handle function.***

Table 1-19. USB globle interrupt

Interrupt Flag	Description	Operation Mode
SEIF	Session interrupt	Host or device mode
DISCIF	Disconnect interrupt flag	Host Mode
IDPSC	ID pin status change	Host or device mode

Interrupt Flag	Description	Operation Mode
PTXFEIF	Periodic Tx FIFO empty interrupt flag	Host Mode
HCIF	Host channels interrupt flag	Host Mode
HPIF	Host port interrupt flag	Host Mode
ISOONCIF/PXNCIF	Periodic transfer Not Complete Interrupt flag /Isochronous OUT transfer Not Complete Interrupt Flag	Host or device mode
ISOINCIF	Isochronous IN transfer Not Complete Interrupt Flag	Device mode
OEPIF	OUT endpoint interrupt flag	Device mode
IEPIF	IN endpoint interrupt flag	Device mode
EOPFIF	End of periodic frame interrupt flag	Device mode
ISOOPDIF	Isochronous OUT packet dropped interrupt flag	Device mode
ENUMF	Enumeration finished	Device mode
RST	USB reset	Device mode
SP	USB suspend	Device mode
ESP	Early suspend	Device mode
GONAK	Global OUT NAK effective	Device mode
GNPINAK	Global IN Non-Periodic NAK effective	Device mode
NPTXFEIF	Non-Periodic Tx FIFO empty interrupt flag	Host Mode
RXFNEIF	Rx FIFO non-empty interrupt flag	Host or device mode
SOF	Start of frame	Host or device mode
OTGIF	OTG interrupt flag	Host or device mode
MFIF	Mode fault interrupt flag	Host or device mode

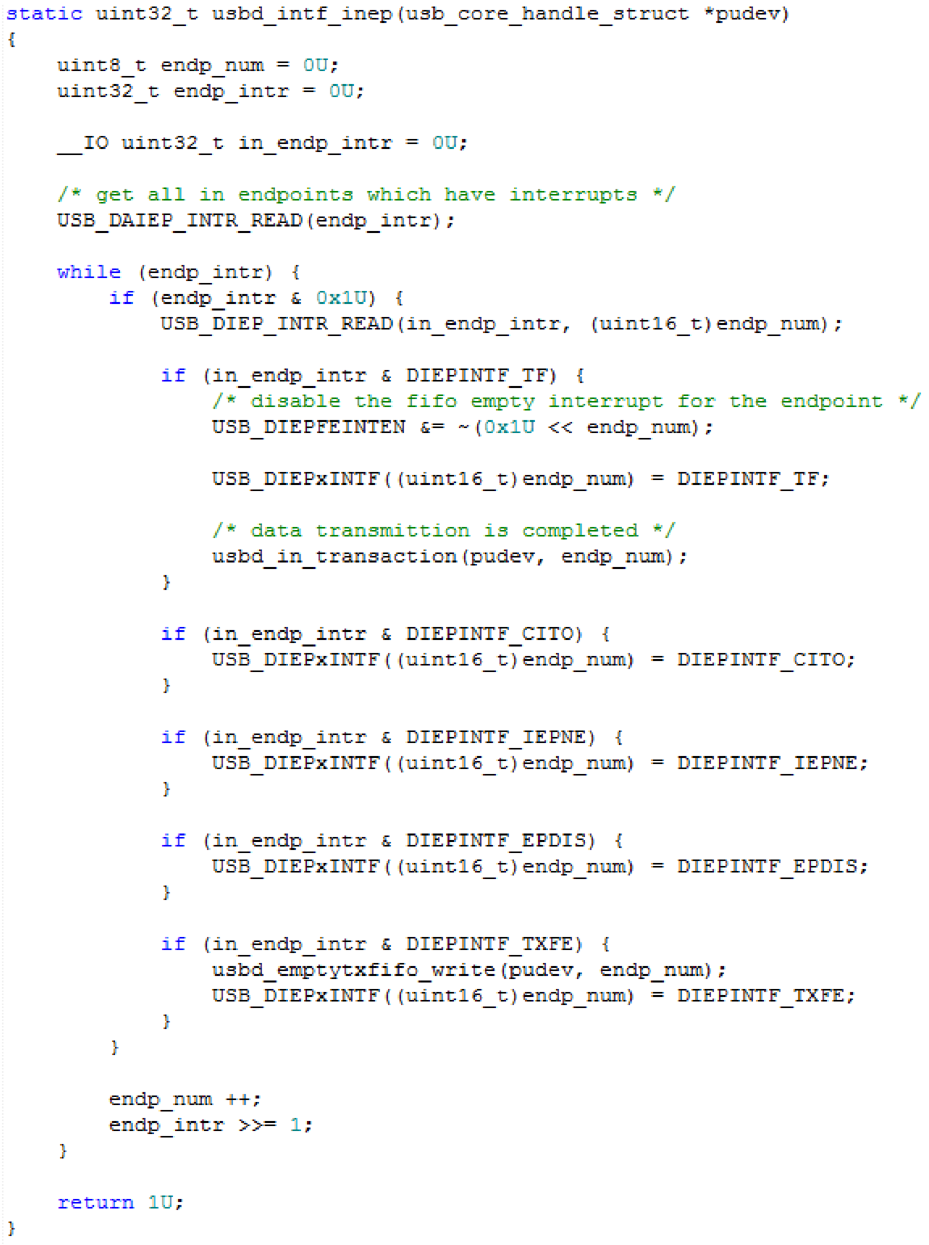
Figure 1-14. OUT endpoint interrupt handle function



```
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;
}
```

In OUT endpoint interrupt handler function, depending on endpoint interrupt flag register, interrupt event of OUT endpoint could be distinguished. The events include such events below: transfer finished event, endpoint disabled event, SETUP phase finished event, endpoint Rx FIFO overrun event and back-to-back SETUP packets event. When OUT endpoint event is generated, polling interrupt flag is easy to enter corresponding interrupt handler function. IN endpoint interrupt handle function is shown in ***Figure 1-15. IN endpoint interrupt handle function.***

Figure 1-15. IN endpoint interrupt handle function



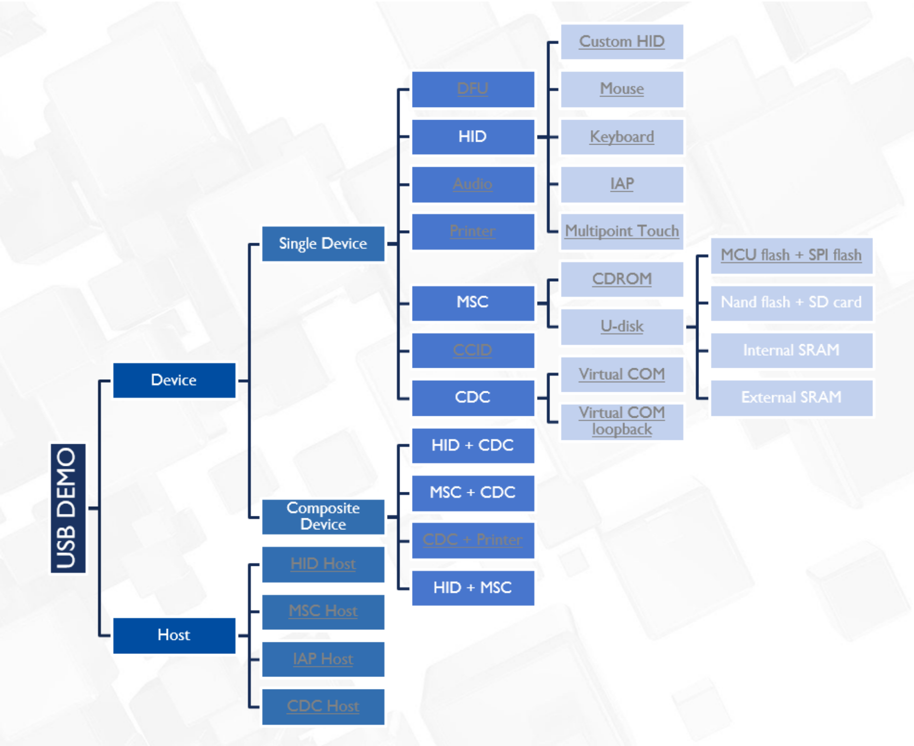
```
static uint32_t usbd_intf_inep(usb_core_handle_struct *pudev)
{
    uint8_t endp_num = 0U;
    uint32_t endp_intr = 0U;

    __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;
}
```

In IN endpoint interrupt handler function, depending on IN endpoint interrupt flag register, interrupt event of IN endpoint could be distinguished. The events include such events below: transfer finished event, endpoint disabled event, control In Timeout interrupt event, endpoint Tx FIFO underrun event, IN endpoint NAK effective event and Transmit FIFO empty event. When IN endpoint event is generated, polling interrupt flag is easy to enter corresponding interrupt handler function.

1.6. USB DEMO

1.6.1. USB demo brief

Figure 1-16. USBFS demo diagram



As shown in ***Figure 1-16. USBFS demo diagram***, USB demo compose of host demo and device demo. Host contain HID host, MSC host, CDC host and IAP host. Device compose of compsite device and single device. Compsite device could be composed of different device, like HID and CDC compsite device, MSC and CDC compsite device, CDC and Printer compsite device and HID and MSC device. According to application protocol, single device include DFU, HID, Audio, Printer, MSC, CCID and CDC device. There are severval sub HID class, such as mouse, keyboard, IAP and multi-touch device. MSC device have severval sub MSC class, like CD-ROM and U-disk device, whose storage medium is MCU flash, SPI flash, Nand flash, SD card, MCU SRAM and external SRAM. In above those demo, MSC host demo and HID keyboard demo are selected to introduce Working principle and operating results of host and device.

1.6.2. MSC host

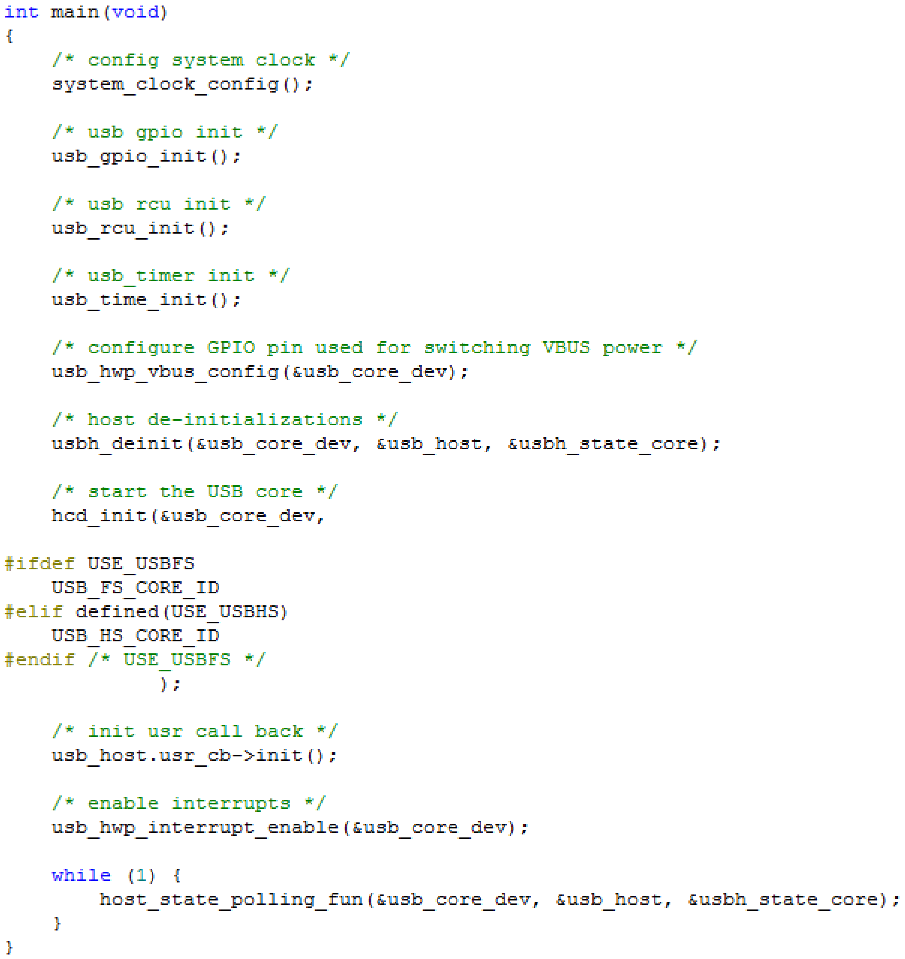
Overview

The demo briefly introduce USBFS host as U disk, once U disk is connected to GD32F450i board, USBFS enumerate the U disk. After U disk completed enumeration, user application start, the demo could read U disk file index and write file to U disk.

Main function introduction

In the demo, there is USBFS host initialization section before while (1), after initialization, program enter main loop and start USBFS host machine, which would be introduced as below.

Figure 1-17. USBFS host U disk main function

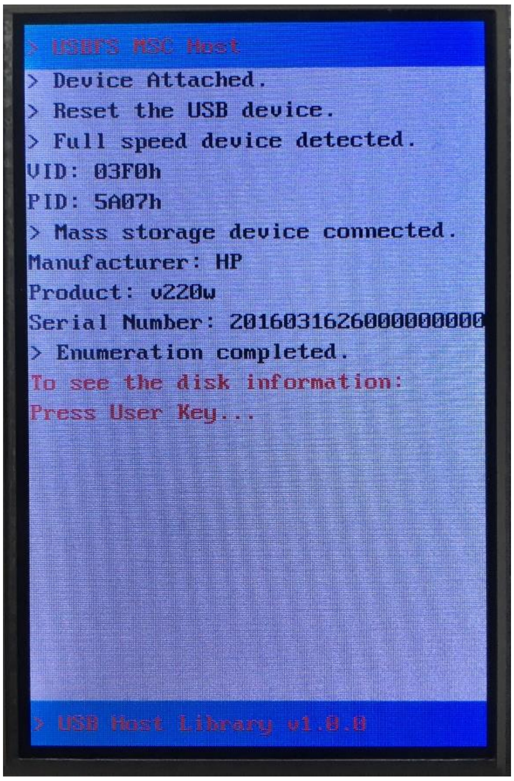


```
int main(void)
{
    /* config system clock */
    system_clock_config();

    /* 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 host experiment result

Download MSC_Host/GD32450I_EVAL_Fullspeed project code to GD32F450i board, and connect U disk to USBFS via OTG cable, run the program, user could observe U disk enumeration information, which is shown in ***Figure 1-18. USBFS host U disk enumeration information.***

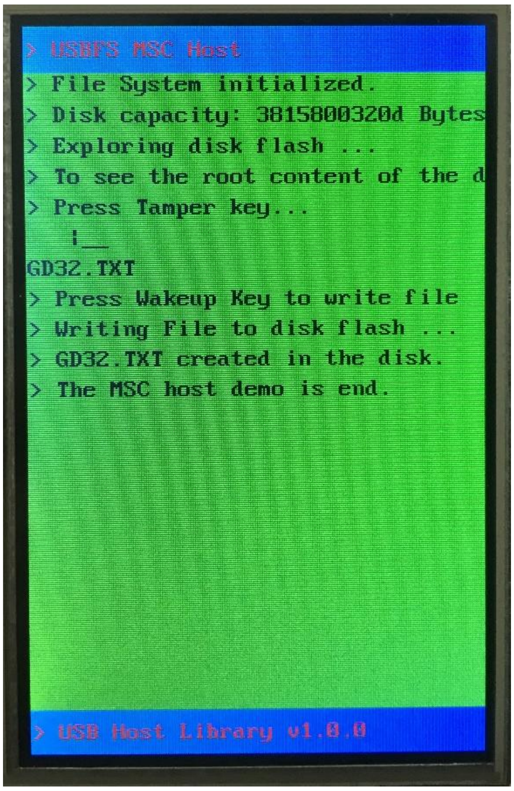
Figure 1-18. USBFS host U disk enumeration information



```
> USBFS MSC Host
> Device Attached.
> Reset the USB device.
> Full speed device detected.
VID: 03F0h
PID: 5A07h
> Mass storage device connected.
Manufacturer: HP
Product: v220w
Serial Number: 2016031626000000000
> Enumeration completed.
To see the disk information:
Press User Key...

> USB Host Library v1.0.0
```

Firstly, press User key, screen shown U disk information, then press Tamper key, screen shown root index content. Pressing Wakeup key is writing file to U disk, then user could observe MSC host demo complete information, which is shown in ***Figure 1-19. USBFS host connect U disk routine result.***

Figure 1-19. USBFS host connect U disk routine result



```
> USBFS MSC Host
> File System initialized.
> Disk capacity: 3815800320d Bytes
> Exploring disk flash ...
> To see the root content of the d
> Press Tamper key...
   |__
GD32.TXT
> Press Wakeup Key to write file
> Writing File to disk flash ...
> GD32.TXT created in the disk.
> The MSC host demo is end.

> USB Host Library v1.0.0
```

1.6.3. HID keyboard device

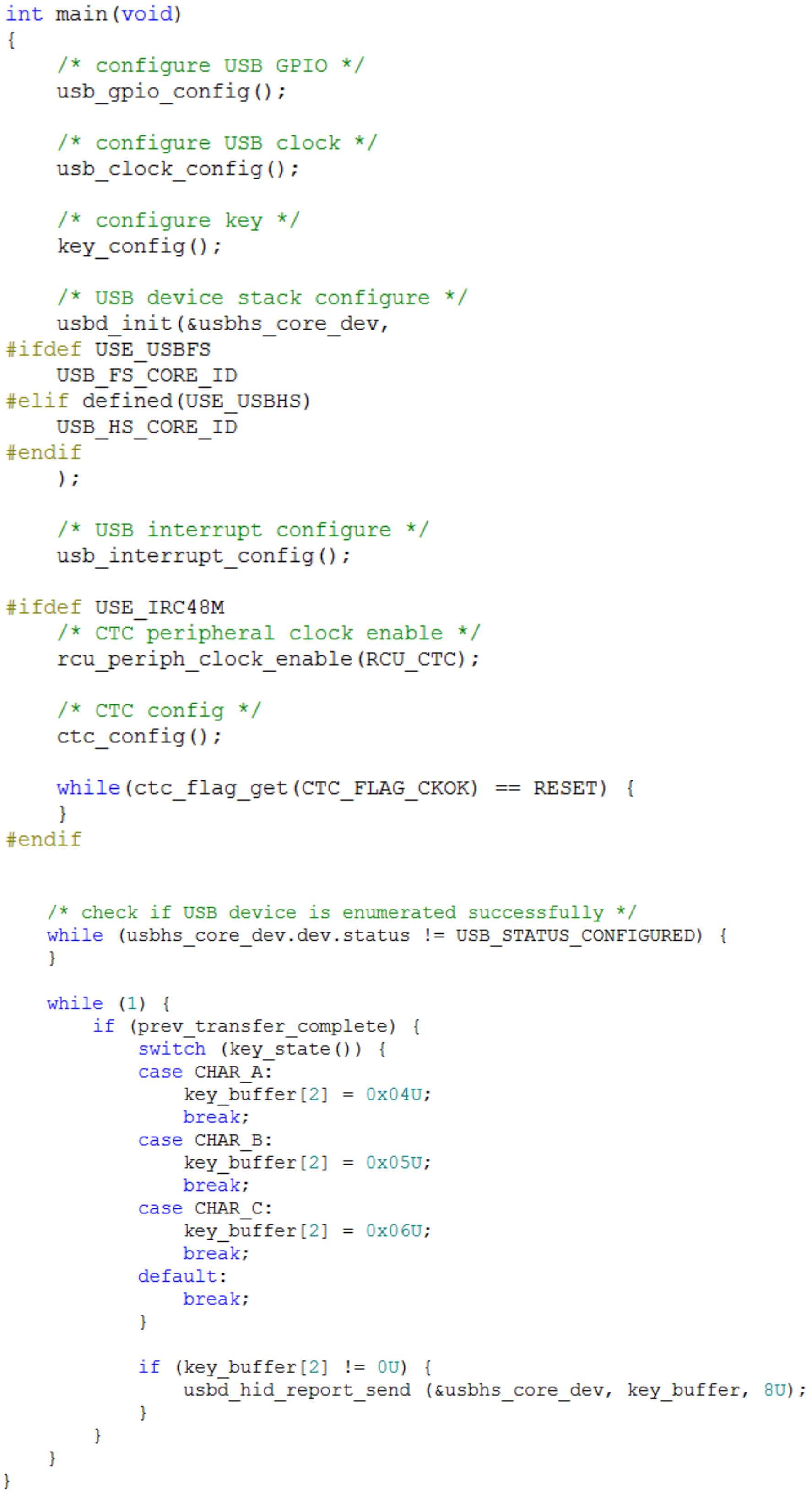
Overview

The demo briefly introduce USBFS device, as keyboard, once GD32F450i board is connected to host, host enumerate the board as keyboard. Press the board key is print the corresponding character, pressing the tamper key would output ‘a’, pressing the wakeup key would output ‘b’, pressing the user key would output ‘c’.

Main function and initialization

USBFS interface module work in device mode, the demo enumerate USBFS device as keyboard, and its main function is shown in ***Figure 1-20. USBFS main function***

Figure 1-20. USBFS main function



```
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);
            }
        }
    }
}
```

In the processing of initialization, DP pin doesn`t pull-up, it is difference between USBFS device and USBD device.

DP pin pull up and disconnect

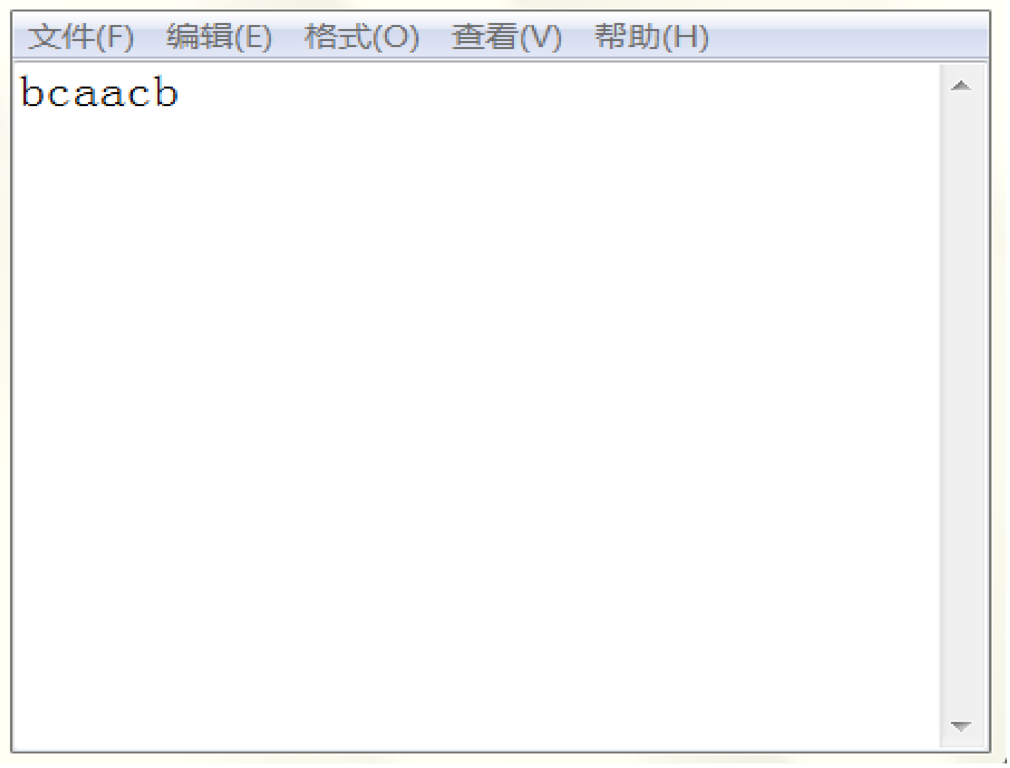
When DP pin pull-up of USBFS device is automatically realized through hardware detected, software do not need operation. When VBUS pin detect B class convention is active level, USBFS interface module could automatically connect pull-up resistance on DP pin, so as to send full speed device connected signal to host, and trigger device interrupt (SESIF@ USBFS_GINTF).

When MCU detected that VBUS pin voltage level is lower than B class convention valid voltage level, USBFS module will automatically disconnect, and trigger device interrupt (SESEND@USBFS_GOTGINTF). Moreover, software is also used to disconnect device. Configuring software disconnected bit (SD@USBFS_DCTL) of device control register, so as to enable software disconnect. Then, USBFS interface module remove pull-up resistance on DP pin, brings out detecting interrupt of device disconnect, thus, even if USB cable is remain connected, device disconnect interrupt could be identified still.

HID keyboard device experiment result

Download HID_Keyboard/GD32450I_EVAL_Fullspeed project code to GD32F450i EVAL board, and connect U disk to USBFS via OTG cable, run the program, as shown in ***Figure 1-21. HID device experiment result***, if user press Wakeup key, Tamper key or User key, correspondingly output ‘b’, ’a’ or ’c’.

Figure 1-21. HID device experiment result



```
bcaacb
```

2. Revision history

Table 2-1. Revision history

Revision No.	Description	Date
1.0	Initial Release	Apr.1st, 2019

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.