GigaDevice Semiconductor Inc.

GD32 USBFS&USBHS Firmware Library User Guide

Application Note
AN050

Table of Contents

List of Figures

List of Tables

1. Introduction

This application note is provided specifically for the GD32 MCU Universal Serial Bus full-speed interface USBFS module. USBHS modules are the same as USBFS in general, corresponding difference between USBFS and USBHS will be marked in the article. The purpose of this note is to make it easier for customers to use the USBFS and USBHS firmware library and to use this library for project development faster.

This application note is divided into four sections:

1. The architecture and files of the library;
2. Introduction to the bottom and middle layer driver functions of the library;
3. Introduction of the Device library;
4. Introduction to the Host library.

Table 1-1. Applicable products

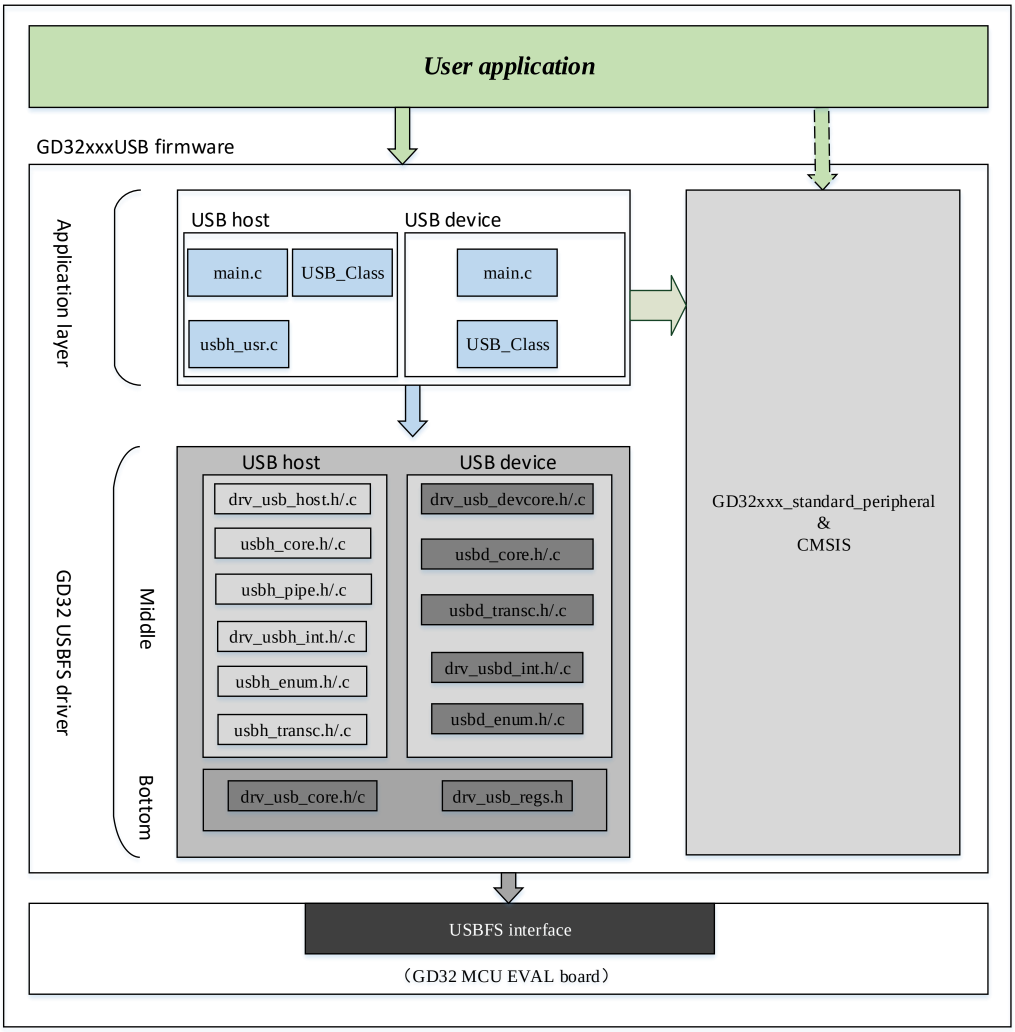
Type	Product Model	
MCU	FS	GD32F105xx series
		GD32F107xx series
		GD32F205xx series
		GD32F207xx series
		GD32F305xx series
		GD32F307xx series
		GD32F350xx series
		GD32F4xx series
		GD32E103xx series
		GD32C103xx series
		GD32E505xx series
		GD32E507xx series
		GD32E508xx series
		GD32VF103xx series
		GD32W515xx series
	HS	GD32F4xx series
		GD32E505xx series
		GD32E507xx series
		GD32E508xx series

2. Library architecture and file structure

2.1. Library Architecture

USBFS module firmware structure of GD32 series is shown in ***Figure 2-1. GD32 USBFS firmware library framework***. The figure show USBFS host and device structure, user application call the USBFS firmware library to realize the USB data commuction. The underlying structure is hardware, that is MCU evaluation board. USBFS firmware library consist of application layer and driver layer. User is able to modify application layer, while user should not modify driver layer, which consist of host driver, device driver and USB underlying layer. As a portion of application layer, USB application class file realize specific host or device application class request. The library structure of USBFS and USBHS is similar.

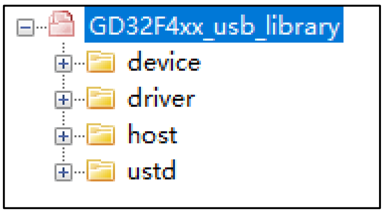
Figure 2-1. GD32 USBFS firmware library framework



2.2. File structure

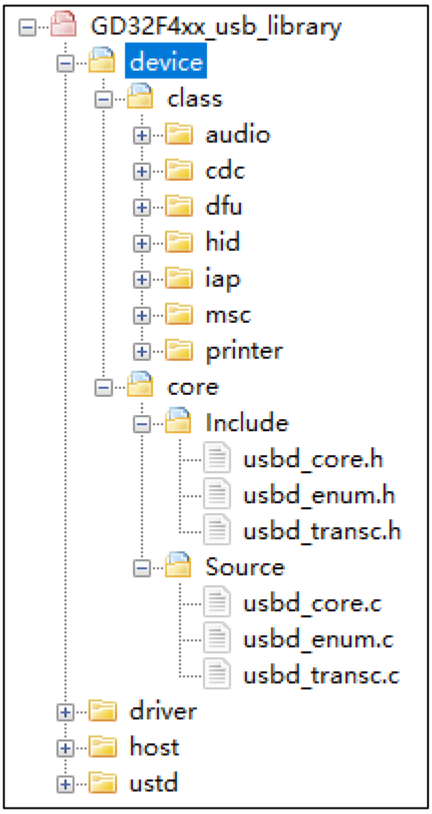
Take firmware library of GD32F4xx as an example, include the following four file folders.

Figure 2-2. USBFS Firmware Library Folder



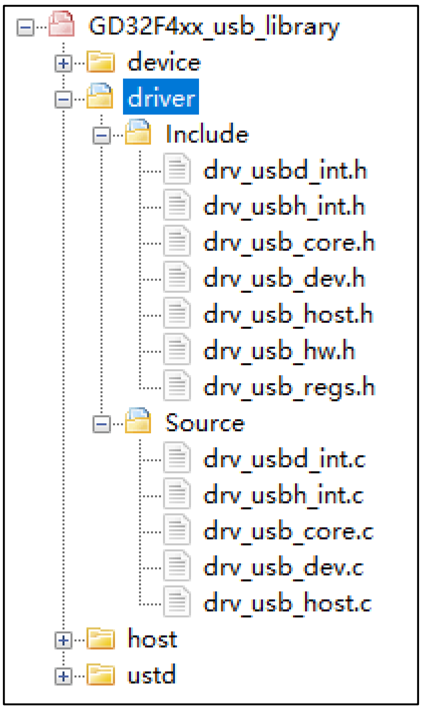
Device file folder include Protocol layer files and device class files, which is required by USB device.

Figure 2-3. Device folder



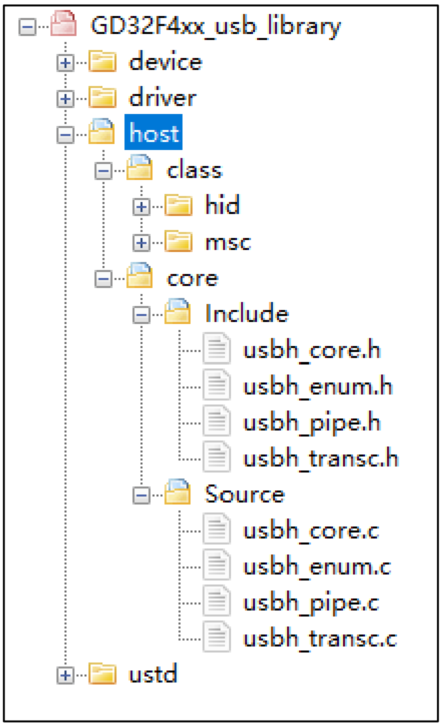
Driver folder include register definition, bottom layer driver and USB interrupt handler, which is used to build device and host application.

Figure 2-4. Driver folder



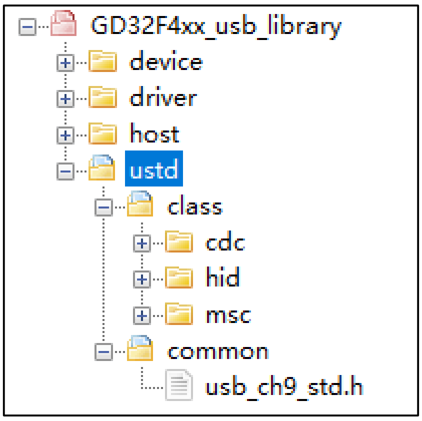
Host folder include register definition, bottom layer driver and USB interrupt handler, which is required by USB host.

Figure 2-5. Host folder



The folder ustd include common device class file and standard enumeration file, which is called by host and device.

Figure 2-6. Ustd folder



3. USBFS bottom driver

The file of USBFS bottom driver located in the driver folder. Bottom driver of whole firmware library is directly relate to USB hardware module, which include register read-write and FIFO operation, As shown in ***Table 3-1. USBFS underlying file***.

Table 3-1. USBFS underlying file

Used range	File name	Functional description
general used	drv_usb_core.h/.c	USB core driver
	drv_usb_regs.h	USB core bottom driver header file
	drv_usb_hw.h	USB hardware configuration header file

The functions in the C file are described in detail below. As shown in ***Table 3-2. usb_core.h/.c file function***.

Table 3-2. usb_core.h/.c file function

Function name	Functional description
usb_core_reset	config USB core to soft reset
usb_basic_init	config USB core basic
usb_core_init	initializes the USB controller registers and prepares the core device mode or host mode operation
usb_txfifo_write	write a packet into the Tx FIFO associated with the endpoint
usb_rxfifo_read	read a packet from the Rx FIFO associated with the endpoint
usb_txfifo_flush	flush a Tx FIFO or all Tx FIFOs
usb_rxfifo_flush	flush the entire Rx FIFO
usb_set_rxfifo	set Rx FIFO size
usb_set_txfifo	set Tx FIFO size

drv_usb_regs.h file define the whole USBFS module register content, all bottom layer operation should call the file. drv_usb_hw.h file declare the relevant function about USB clock, GPIO, delay, interrupt enable and CTC.

4. USBFS middle layer driver

The middle driver layer is divided into host middle layer and device middle layer. Device middle layer packages the transaction and basic function of USB device transfer. Host middle layer packages the transaction and basic function of USB host transfer. As shown in ***Table 4-1. USBFS middle layer driver file.***

Table 4-1. USBFS middle layer driver file

Range	folder	Function name	Functional description
Host	driver	drv_usb_host.h /.c	USB host mode low level driver
		drv_usbh_int.h /.c	USB host mode interrupt handler file
	host/core	usbh_core.h/.c	USB host core state machine driver
		usbh_enum.h/.c	USB host mode enumberation driver
		usbh_pipe.h/.c	USB host mode pipe operation driver
		usbh_transc.c	USB host mode transactions driver
device	driver	drv_usb_dev.h /.c	USB device low level driver
		drv_usbd_int.h /.c	USB device mode interrupt routines
	device/core	usbd_core.h /.c	USB device mode core functions
		usbd_enum.h /.c	USB enumeration function
		usbd_transc.h/.c	USB transaction core functions

4.1. Host middle layer driver function

Table 4-2. drv_usb_host.h/.c file function

Function name	Functional description
usb_host_init	initializes USB core for host mode
usb_portvbus_switch	control the VBUS to power
usb_port_reset	reset host port
usb_pipe_init	initialize host pipe
usb_pipe_xfer	prepare host channel for transferring packets
usb_pipe_halt	halt pipe
usb_pipe_ping	configure host pipe to do ping operation
usb_host_stop	stop the USB host and clean up FIFO

Table 4-3. drv_usbh_int.h/.c file function

Function name	Functional description
usbh_isr	handle global host interrupt
usbh_int_port	handle the host port interrupt
usbh_int_pipe	handle all host channels interrupt
usbh_int_pipe_in	handle the IN channel interrupt
usbh_int_pipe_out	handle the OUT channel interrupt
usbh_int_rxfifonoempty	handle the rx fifo non-empty interrupt

Function name	Functional description
usbh_int_txfifoempty	handle the TX FIFO empty interrupt

Table 4-4. usbh_core.h/.c file function

Function name	Functional description
usbh_init	USB host stack initializations
usbh_deinit	de-initialize USB host
usbh_class_register	USB host register device class
usbh_core_task	USB host core main state machine process
usbh_error_handler	handle the error on USB host side
usbh_sof	USB SOF callback function from the interrupt
usbh_connect	USB connect callback function from the interrupt
usbh_disconnect	USB disconnect callback function from the interrupt
usbh_port_enabled	USB port enable callback function from the interrupt
usbh_port_disabled	USB port disabled callback function from the interrupt
usbh_enum_task	handle the USB enumeration task

Table 4-5. usbh_enum.h/.c file function

Function name	Functional description
usbh_ctlstate_config	configure USB control status parameters
usbh_devdesc_get	get device descriptor from the USB device
usbh_cfgdesc_get	get configuration descriptor from the USB device
usbh_strdesc_get	get string descriptor from the USB device
usbh_setaddress	set the address to the connected device
usbh_setcfg	set the configuration value to the connected device
usbh_setinterface	set the interface value to the connected device
usbh_setdevfeature	set or enable a specific device feature
usbh_clrdevfeature	clear or disable a specific device feature
usbh_clrfeature	clear or disable a specific feature
usbh_nextdesc_get	get the next descriptor header
usbh_interface_select	select a interface
usbh_interface_find	find the interface index for a specific class
usbh_interfaceindex_find	find the interface index for a specific class interface and alternate setting number
usbh_devdesc_parse	parse the device descriptor
usbh_cfgdesc_parse	parse the configuration descriptor
usbh_cfgset_parse	parse the configuration descriptor set
usbh_itfdesc_parse	parse the interface descriptor
usbh_epdesc_parse	parse the endpoint descriptor
usbh_strdesc_parse	parse the string descriptor

Table 4-6. usbh_pipe.h/.c file function

Function name	Functional description
usbh_pipe_create	create a pipe

Function name	Functional description
usbh_pipe_update	modify a pipe
usbh_pipe_allocate	allocate a new pipe
usbh_pipe_free	free a pipe
usbh_pipe_delete	delete all USB host pipe
usbh_freepipe_get	get a free pipe number for allocation

Table 4-7. usbh_transc.h/.c file function

Function name	Functional description
usbh_ctlsetup_send	send the setup packet to the USB device
usbh_data_send	send a data packet to the USB device
usbh_data_recev	receive a data packet from the USB device
usbh_ctl_handler	USB control transfer handler
usbh_urb_wait	wait for USB URB(USB request block) state
usbh_setup_transc	USB setup transaction
usbh_data_in_transc	USB data IN transaction
usbh_data_out_transc	USB data OUT transaction
usbh_status_in_transc	USB status IN transaction
usbh_status_out_transc	USB status OUT transaction
usbh_request_submit	prepare a pipe and start a transfer

4.2. Device middle layer driver function

Table 4-8. drv_usb_dev.h/.c file function

Function name	Functional description
usb_devcore_init	initialize USB core registers for device mode
usb_devint_enable	enable the USB device mode interrupts
usb_transc0_active	active the USB endpoint 0 transaction
usb_transc_active	active the USB transaction
usb_transc_deactivate	deactive the USB transaction
usb_transc_inxfer	configure USB transaction to start IN transfer
usb_transc_outxfer	configure USB transaction to start OUT transfer
usb_transc_stall	set the USB transaction STALL status
usb_transc_clrstall	clear the USB transaction STALL status
usb_iepintr_read	read device IN endpoint interrupt flag register
usb_ctlep_startout	configures OUT endpoint 0 to receive SETUP packets
usb_rwkup_active	active remote wakeup signalling
usb_clock_active	active USB core clock
usb_dev_suspend	USB device suspend
usb_dev_stop	stop the device and clean up fifos
usb_dev_disconnect	config the USB device to be disconnected
usb_dev_connect	config the USB device to be connected

Function name	Functional description
usb_devaddr_set	set the USB device address
usb_oepintnum_read	read device all OUT endpoint interrupt register
usb_oepintr_read	read device OUT endpoint interrupt flag register
usb_iepintnum_read	read device all IN endpoint interrupt register
usb_rwkup_set	set remote wakeup signalling
usb_rwkup_reset	reset remote wakeup signalling

Table 4-9. drv_usbd_int.h/.c file function

Function name	Functional description
usbd_int_dedicated_ep1out	USB dedicated OUT endpoint 1 interrupt service routine handler
usbd_int_dedicated_ep1in	USB dedicated IN endpoint 1 interrupt service routine handler
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_int_rxfifo	handle the RX status queue level interrupt
usbd_int_reset	handle USB reset interrupt
usbd_int_enumfinish	handle USB speed enumeration finish interrupt
usbd_int_suspend	USB suspend interrupt handler
usbd_emptytxfifo_write	check FIFO for the next packet to be loaded

Table 4-10. usbd_core.h/.c file function

Function name	Functional description
usbd_init	initailizes the USB device-mode stack and load the class driver
usbd_ep_setup	endpoint initialization
usbd_ep_clear	configure the endpoint when it is disabled
usbd_ep_recev	endpoint prepare to receive data
usbd_ep_send	endpoint prepare to transmit data
usbd_ep_stall	set an endpoint to STALL status
usbd_ep_stall_clear	clear endpoint STALLed status
usbd_fifo_flush	flush the endpoint FIFOs
usbd_connect	device connect
usbd_disconnect	device disconnect
usbd_addr_set	set USB device address
usbd_rxcount_get	get the received data length

Table 4-11. usbd_enum.h/.c file function

Function name	Functional description
usbd_standard_request	handle USB standard device request
usbd_class_request	handle USB device class request
usbd_vendor_request	handle USB vendor request
usbd_enum_error	handle USB enumeration error
int_to_unicode	convert hex 32bits value into unicode char
serial_string_get	get serial string

Function name	Functional description
_usb_std_reserved	no operation, just for reserved
_usb_dev_desc_get	get the device descriptor
_usb_config_desc_get	get the configuration descriptor
_usb_bos_desc_get	get the BOS descriptor
_usb_str_desc_get	get string descriptor
_usb_std_getstatus	handle Get_Status request
_usb_std_clearfeature	handle USB Clear_Feature request
_usb_std_setfeature	handle USB Set_Feature request
_usb_std_setaddress	handle USB Set_Address request
_usb_std_getdescriptor	handle USB Get_Descriptor request
_usb_std_setdescriptor	handle USB Set_Descriptor request
_usb_std_getconfiguration	handle USB Get_Configuration request
_usb_std_setconfiguration	handle USB Set_Configuration request
_usb_std_getinterface	handle USB Get_Interface request
_usb_std_setinterface	handle USB Set_Interface request
_usb_std_synchframe	handle USB SynchFrame request

Table 4-12. usbd_transc.h/.c file function

Function name	Functional description
usbd_ctl_send	USB send data in the control transaction
usbd_ctl_recev	USB receive data in control transaction
usbd_ctl_status_send	USB send control transaction status
usbd_ctl_status_recev	USB control receive status
usbd_setup_transc	USB setup stage processing
usbd_out_transc	data out stage processing
usbd_in_transc	data in stage processing

5. USBFS Device Library

USBFS device library is based on the above underlying layer and middle layer driver, which include device library configuration, descriptor, interrupt, user interface, device class interface and example introduction.

5.1. Device Library Configuration

Device configuration include two file, usbd_conf.h and usb_conf.h, which is located in project file folder, while device configuration file and other driver file are stored in same directory.

5.1.1. usbd_conf.h

The file configuration options are as follows:

```
#define USBD_CFG_MAX_NUM                    1
#define USBD_ITF_MAX_NUM                    1

#define USBD_HID_INTERFACE                  0

/* USB user string supported */
/* #define USB_SUPPORT_USER_STRING_DESC */

#define HID_IN_EP                           EP1_IN

#define HID_IN_PACKET                       8
```

Each configuration is defined as follows ***Table 5-1. usbd_conf.h configuration description***.

Table 5-1. usbd_conf.h configuration description

Configuration name	Functional description
USBD_CFG_MAX_NUM	configuration maximum number
USBD_ITF_MAX_NUM	interface maximum number
USBD_HID_INTERFACE	Device class index
HID_IN_EP	IN endpoint index
HID_IN_PACKET	endpoint data packet length

5.1.2. usb_conf.h

The usb_conf.h file mainly define FIFO allocation of USBFS module. USBFS has 1.25KB RAM buffer, that is 320 words FIFO. While, USBHS has 4KB RAM buffer, that is 1024 words FIFO.

The file configuration options are as follows:

```
#ifdef USB_FS_CORE
    #define RX_FIFO_FS_SIZE                          128
    #define TX0_FIFO_FS_SIZE                         64
    #define TX1_FIFO_FS_SIZE                         128
    #define TX2_FIFO_FS_SIZE                         0
    #define TX3_FIFO_FS_SIZE                         0
#endif /* USB_FS_CORE */

#define USB_SOF_OUTPUT              1
#define USB_LOW_POWER               1

//#define VBUS_SENSING_ENABLED

//#define USE_HOST_MODE
#define USE_DEVICE_MODE
//#define USE_OTG_MODE
```

Table 5-2. usb_conf.h configuration description

Configuration name	Functional description
RX_FIFO_FS_SIZE	RX FIFO size configuration
TX_FIFO_FS_SIZE	TX FIFO size configuration
USB_SOF_OUTPUT	enable SOF output
USB_LOW_POWER	enable low power mode
VBUS_SENSING_ENABLED	enable VBUS SENSING
USE_HOST_MODE	host mode
USE_DEVICE_MODE	device mode
USE_OTG_MODE	OTG mode

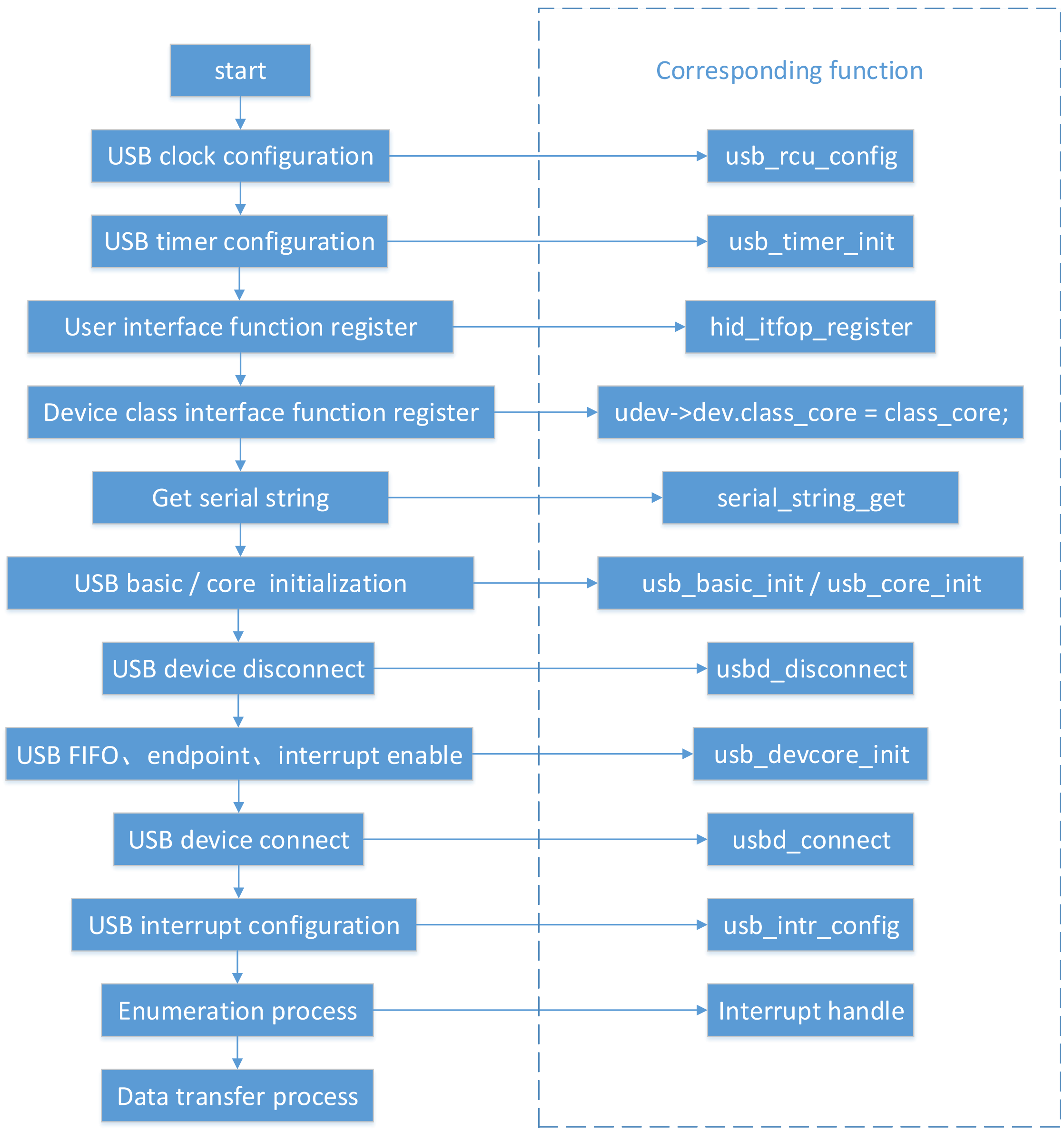
Note:

1. If user need to measure the SOF with an oscilloscope, it is necessary to configure the PA8 pin as the output function.
2. Only one of USE_HOST_MODE, USE_DEVICE_MODE and USE_OTG_MODE coule be selected.
3. GD32F105xx/107xx and GD32F205xx/F207xx series don`t has VBUS_SENSING_ENABLED function, if USB module is configured as device, user should connect VBUS pin.

5.2. Firmware library process

USBFS firmware library flow include clock configuration, interrupt enable, USB register configuration, endpoint configuration, connect, enumeration and data transfer.As shown in ***Figure 5-1. Firmware library flowchart***.

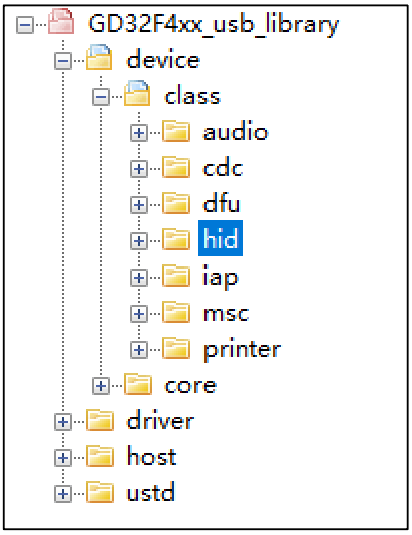
Figure 5-1. Firmware library flowchart



5.3. Descriptor

The descriptors of the USBFS device library are contained in the device class file, as shown in the ***Figure 5-2. Device class file path***.

Figure 5-2. Device class file path



Descriptor mainly include device descriptor, configuration descriptor and string descriptor, etc. There is some specific descriptor for some device class, such as HID device support report descriptor, high speed device support other speed configuration descriptor and qualifier descriptor.

Descriptor is sent to host in enumeration phase, corresponding device library code is introduced in device/core/usbd_enum.c file. Specifically, in the standard enumeration phase, the following two pointer arrays are implemented by callback function.

```
static usb_reqsta (*_std_dev_req[])(usb_core_driver *udev, usb_req *req) =
{
    [USB_GET_STATUS]          = _usb_std_getstatus,
    [USB_CLEAR_FEATURE]       = _usb_std_clearfeature,
    [USB_RESERVED2]           = _usb_std_reserved,
    [USB_SET_FEATURE]         = _usb_std_setfeature,
    [USB_RESERVED4]           = _usb_std_reserved,
    [USB_SET_ADDRESS]         = _usb_std_setaddress,
    [USB_GET_DESCRIPTOR]      = _usb_std_getdescriptor,     // get descriptor
    [USB_SET_DESCRIPTOR]      = _usb_std_setdescriptor,
    [USB_GET_CONFIGURATION] = _usb_std_getconfiguration,
    [USB_SET_CONFIGURATION] = _usb_std_setconfiguration,
    [USB_GET_INTERFACE]       = _usb_std_getinterface,
    [USB_SET_INTERFACE]       = _usb_std_setinterface,
    [USB_SYNCH_FRAME]         = _usb_std_synchframe,
};

/* get standard descriptor handler */
static uint8_t* (*std_desc_get[])(usb_core_driver *udev, uint8_t index, uint16_t *len) = {
    [(uint8_t)USB_DESCTYPE_DEV - 1U]             = _usb_dev_desc_get,
    [(uint8_t)USB_DESCTYPE_CONFIG - 1U]          = _usb_config_desc_get,
```

```
    [(uint8_t)USB_DESCTYPE_STR - 1U]                 = _usb_str_desc_get,
#ifdef USE_USB_HS
    [(uint8_t)USB_DESCTYPE_DEV_QUALIFIER - 3U]      = _usb_qualifier_desc_get,
    [(uint8_t)USB_DESCTYPE_OTHER_SPD_CONFIG - 3U] =
_usb_other_speed_config_desc_get,
#endif
};
```

5.4. Interrupt handling

The interrupt of USBFS device interface is shown in ***Table 5-3. USBFS device interruption***. Every interrupt flag corresponds to process handler of interrupt fuction, OEPIF, IEPIF and RXFNEIF flag is concerned about data transfer. GINTF_OEPIF of OUT endpoint is for OUT transaction, and GINTF_IEPIF of IN endpoint is for IN transaction.

Table 5-3. USBFS device interruption

Interrupt Flag	Description	Operation Mode
WKUPIF	Wakeup interrupt	Host or device mode
SESIF	Session interrupt	Host or device mode
IDPSC	ID pin status change	Host or device mode
LPMIF(1)	LPM interrupt flag	Host or device 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
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

Note:

(1) This bit is only in the E50X series.

The interrupt handler function of OUT endpoint is shown as below:

```
static uint32_t usbd_int_epout (usb_core_driver *udev)
{
    uint32_t epintnum = 0U;
    uint8_t ep_num = 0U;

    for (epintnum = usb_oepintnum_read (udev); epintnum; epintnum >>= 1, ep_num++) {
        if (epintnum & 0x01U) {
            __IO uint32_t oepintr = usb_oepintr_read (udev, ep_num);

            /* transfer complete interrupt */
            if (oepintr & DOEPINTF_TF) {
                /* clear the bit in DOEPINTF for this interrupt */
                udev->regs.er_out[ep_num]->DOEPINTF = DOEPINTF_TF;

                if ((uint8_t)USB_USE_DMA == udev->bp.transfer_mode) {
                    __IO uint32_t eplen = udev->regs.er_out[ep_num]->DOEPLEN;

                    udev->dev.transc_out[ep_num].xfer_count =
udev->dev.transc_out[ep_num].max_len - \
                                                                        (eplen & DEPLEN_TLEN);
                }

                /* inform upper layer: data ready */
                (void)usbd_out_transc (udev, ep_num);      // out transaction

                if ((uint8_t)USB_USE_DMA == udev->bp.transfer_mode) {
                    if ((0U == ep_num) && ((uint8_t)USB_CTL_STATUS_OUT ==
udev->dev.control.ctl_state)) {
                        usb_ctlep_startout (udev);
                    }
                }
            }

            /* setup phase finished interrupt (control endpoints) */
            if (oepintr & DOEPINTF_STPF) {
                /* inform the upper layer that a setup packet is available */
                (void)usbd_setup_transc (udev);      // setup transaction

                udev->regs.er_out[ep_num]->DOEPINTF = DOEPINTF_STPF;
            }
        }
    }
```

```
    return 1U;
}
```

In OUT endpoint interrupt handler function, depending on the interrupt flag register, out_endp_intr function judge the event of OUT endpoint interrupt, which include transfer finished interrupt and setup phase finished interrupt. After corresponding OUT endpoint interrupt event generated, MCU execute the corresponding interrupt handler function through polling interrupt flag.

The interrupt handler function of IN endpoint is shown as below:

```
static uint32_t usbd_int_epin (usb_core_driver *udev)
{
    uint32_t epintnum = 0U;
    uint8_t ep_num = 0U;

    for (epintnum = usb_iepintnum_read (udev); epintnum; epintnum >>= 1, ep_num++) {
        if (epintnum & 0x1U) {
            __IO uint32_t iepintr = usb_iepintr_read (udev, ep_num);

            if (iepintr & DIEPINTF_TF) {
                udev->regs.er_in[ep_num]->DIEPINTF = DIEPINTF_TF;

                /* data transmission is completed */
                (void)usbd_in_transc (udev, ep_num);    // IN transaction

                if ((uint8_t)USB_USE_DMA == udev->bp.transfer_mode) {
                    if ((0U == ep_num) && ((uint8_t)USB_CTL_STATUS_IN ==
udev->dev.control.ctl_state)) {
                        usb_ctlep_startout (udev);
                    }
                }
            }

            if (iepintr & DIEPINTF_TXFE) {
                usbd_emptytxfifo_write (udev, (uint32_t)ep_num);    // write FIFO

                udev->regs.er_in[ep_num]->DIEPINTF = DIEPINTF_TXFE;
            }
        }
    }

    return 1U;
}
```

In interrupt handler function of IN endpoint, process transfer finished interrupt and transmit FIFO empty interrupt. After corresponding IN endpoint interrupt event generated, MCU execute the corresponding interrupt handler function through polling interrupt flag.

The interrupt handler function of Rx FIFO non empty is shown as below:

```
static uint32_t usbd_int_rxfifo (usb_core_driver *udev)
{
    usb_transc *transc = NULL;

    uint8_t data_PID = 0U;
    uint32_t bcount = 0U;

    __IO uint32_t devrxstat = 0U;

    /* disable the Rx status queue non-empty interrupt */
    udev->regs.gr->GINTEN &= ~GINTEN_RXFNEIE;

    /* get the status from the top of the FIFO */
    devrxstat = udev->regs.gr->GRSTATP;

    uint8_t ep_num = (uint8_t)(devrxstat & GRSTATRP_EPNUM);

    transc = &udev->dev.transc_out[ep_num];

    bcount = (devrxstat & GRSTATRP_BCOUNT) >> 4U;
    data_PID = (uint8_t)((devrxstat & GRSTATRP_DPID) >> 15U);

    switch ((devrxstat & GRSTATRP_RPCKST) >> 17U) {
    case RSTAT_GOUT_NAK:
        break;

    case RSTAT_DATA_UPDT:
        if (bcount > 0U) {
            (void)usb_rxfifo_read (&udev->regs, transc->xfer_buf, (uint16_t)bcount);      // read
FIFO

            transc->xfer_buf += bcount;
            transc->xfer_count += bcount;
        }
        break;

    case RSTAT_XFER_COMP:
        /* trigger the OUT endpoint interrupt */
        break;
```

```
    case RSTAT_SETUP_COMP:
        /* trigger the OUT endpoint interrupt */
        break;

    case RSTAT_SETUP_UPDT:
        if ((0U == transc->ep_addr.num) && (8U == bcount) && (DPID_DATA0 == data_PID)) {
            /* copy the setup packet received in FIFO into the setup buffer in RAM */
            (void)usb_rxfifo_read (&udev->regs, (uint8_t *)&udev->dev.control.req,
(uint16_t)bcount);        // read FIFO

            transc->xfer_count += bcount;
        }
        break;

    default:
        break;
    }

    /* enable the Rx status queue level interrupt */
    udev->regs.gr->GINTEN |= GINTEN_RXFNEIE;

    return 1U;
}
```

In interrupt handler function of Rx FIFO non empty, mainly process FIFO data receiving, include SETUP transaction interrupt and OUT transaction interrupt.

5.5. USB Device Class Interface

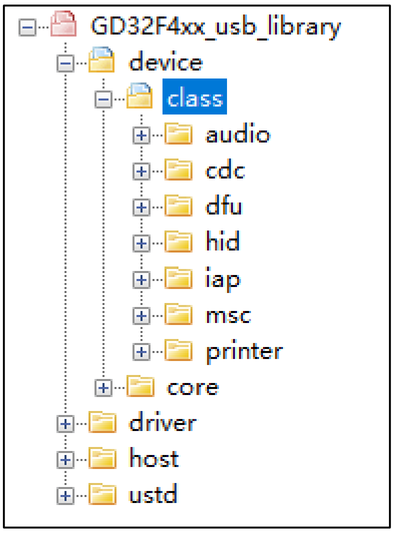
The USB device class interface is implemented by the following architecture:

```
typedef struct _usb_class_core
{
    uint8_t   command;                                                  /*!<
device class request command */
    uint8_t   alter_set;                                              /*!<
alternative set */

    uint8_t   (*init)              (usb_dev *udev, uint8_t config_index);    /*!< initialize
handler */
    uint8_t   (*deinit)            (usb_dev *udev, uint8_t config_index);    /*!< de-initialize
handler */

```

```
    uint8_t   (*req_proc)            (usb_dev *udev, usb_req *req);          /*!< device
request handler */

    uint8_t   (*set_intf)            (usb_dev *udev, usb_req *req);          /*!< device set
interface callback */

    uint8_t   (*ctlx_in)             (usb_dev *udev);                         /*!< device
contrl in callback */
    uint8_t   (*ctlx_out)            (usb_dev *udev);

    uint8_t   (*data_in)             (usb_dev *udev, uint8_t ep_num);         /*!< device
data in handler */
    uint8_t   (*data_out)            (usb_dev *udev, uint8_t ep_num);         /*!< device
data out handler */

    uint8_t   (*SOF)                 (usb_dev *udev);                         /*!< Start
of frame handler */

    uint8_t   (*incomplete_isoc_in)  (usb_dev *udev);                         /*!<
Incomplete synchronization IN transfer handler */
    uint8_t   (*incomplete_isoc_out) (usb_dev *udev);                         /*!<
Incomplete synchronization OUT transfer handler */
} usb_class_core;
```

The initialization of structure is realized by corresponding device class, each device class file is saved in corresponding folder of device/class folder path.As shown in ***Figure 5-3. Device class file***.

Figure 5-3. Device class file



For example, the initialization of a HID device is as follows:

```
usb_class_core usbd_hid_cb = {
    .command        = NO_CMD,
    .alter_set      = 0U,
    .init           = hid_init,
    .deinit         = hid_deinit,
    .req_proc       = hid_req,
    .data_in        = hid_data_in
};
```

The above initialization could implement initialization, deinitialization, device class request and data transfer of device class.

5.6. Data transmission process

After enumeration completed, USB could receive and send data, the process is controlled by host. Host receive data through sending IN packet to device, and send data through sending OUT packet. In the follow article, CDC routine briefly describe the data transfer process of USB data in GD FS library.

5.6.1. IN direction

In data transfer phase, non-zero endpoint data of IN direction is processed in usbd_in_transc function. In following figure, data_in callback function actually call the cdc_acm_in function. Once enter in cdc_acm_in function, it is indicated that some data are sent from device to host, and then call send function cdc_acm_data_send, so as to send next data packet.

```
uint8_t usbd_in_transc (usb_core_driver *udev, uint8_t ep_num)
{
    if (0U == ep_num) {
        usb_transc *transc = &udev->dev.transc_in[0];
        /* ...... */
    } else {
        if ((udev->dev.cur_status == (uint8_t)USBD_CONFIGURED) &&
(udev->dev.class_core->data_in != NULL)) {
            (void)udev->dev.class_core->data_in (udev, ep_num);
        }
    }

    return (uint8_t)USBD_OK;
}
```

5.6.2. OUT direction

In data transfer phase, non-zero endpoint data of OUT direction is processed in usbd_out_transc function. data_out callback function actually call the cdc_acm_out function. Once enter in cdc_acm_out function, it is indicated that some data are received by device, and then call function cdc_acm_data_receive, so as to prepare next data packet.

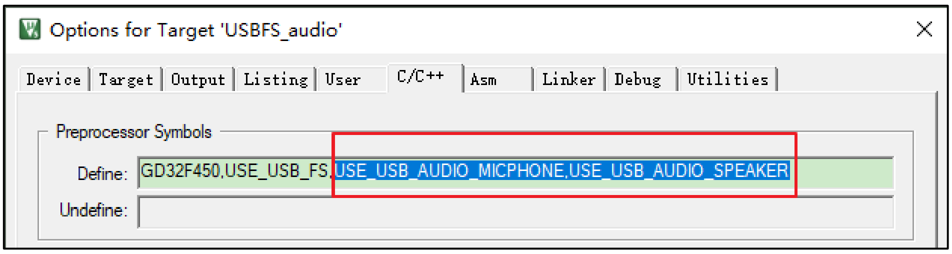
```
uint8_t usbd_out_transc (usb_core_driver *udev, uint8_t ep_num)
{
    if (0U == ep_num) {
        usb_transc *transc = &udev->dev.transc_out[0];
        /*   ......   */
    } else if ((udev->dev.class_core->data_out != NULL) && (udev->dev.cur_status ==
(uint8_t)USBD_CONFIGURED)) {
        (void)udev->dev.class_core->data_out (udev, ep_num);
    } else {
        /* no operation */
    }

    return (uint8_t)USBD_OK;
}
```

5.7. USB device class routine

5.7.1. AUDIO

AUDIO device include speaker and micphone interface, which could be selected in project configuration and shown as ***Figure 5-4. AUDIO macro configuration***.

Figure 5-4. AUDIO macro configuration



Audio descriptor introduction

Device descriptor include VID(0x28e9)/PID(0x9574) of AUDIO device. In configuration descriptor set, include speaker and micphone corresponding descriptor item. Speaker and micphone of AUDIO corresponds to one interface, interface descriptor is shown as ***Table 5-4. AUDIO relevant descriptors***.

Table 5-4. AUDIO relevant descriptors

Descriptor name	Functional description
usb_desc_AC_itf	AC interface descriptor

Descriptor name	Functional description
usb_desc_input_terminal	input terminal descriptor
usb_desc_mono_feature_unit	mono feature unit descriptor
usb_desc_output_terminal	output terminal descriptor
usb_desc_AS_itf	AS interface descriptor
usb_desc_format_type	format type descriptor
usb_desc_std_ep	standard endpoint descriptor
usb_desc_AS_ep	AS endpoint descriptor

AUDIO device class interface

AUDIO device class interface is shown as below struct, the function of struct is referred to *Table 5-5. AUDIO device class interface function*.

```
usb_class_core usbd_audio_cb = {
    .init        = audio_init,
    .deinit      = audio_deinit,
    .req_proc    = audio_req_handler,
    .ctlx_out    = audio_ctlx_out,
    .data_in     = audio_data_in,
    .data_out    = audio_data_out,
    .SOF         = usbd_audio_sof
};
```

Table 5-5. AUDIO device class interface function

Descriptor name	Functional description
audio_init	Initialize AUDIO device
audio_deinit	deinitialize AUDIO device
audio_req_handler	AUDIO device class request function
audio_ctlx_out	OUT control transfer callback
audio_data_in	IN data transfer callback
audio_data_out	OUT data transfer callback
usbd_audio_sof	SOF event callback

AUDIO device class request

AUDIO contains individual device class requests as shown *Table 5-6. AUDIO device class request*.

Table 5-6. AUDIO device class request

Request name	Functional description
AUDIO_REQ_SET_CUR	set current value request
AUDIO_REQ_GET_CUR	get current value request
AUDIO_REQ_SET_MIN	set minimum value request
AUDIO_REQ_GET_MIN	get minimum value request

Request name	Functional description
AUDIO_REQ_SET_MAX	set maximum value request
AUDIO_REQ_GET_MAX	get maximum value request
AUDIO_REQ_SET_RES	set resolution request
AUDIO_REQ_GET_RES	get resolution request

AUDIO user interface

AUDIO user interface definition is shown as below struct.

```
audio_fops_struct audio_out_fops =
{
      init,
      deinit,
      audio_cmd,
      volume_ctl,
      mute_ctl,
      periodic_tc,
      get_state
};
```

Corresponding function is shown as ***Table 5-7. AUDIO user interface functions***.

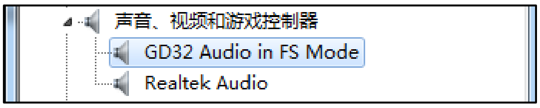
Table 5-7. AUDIO user interface functions

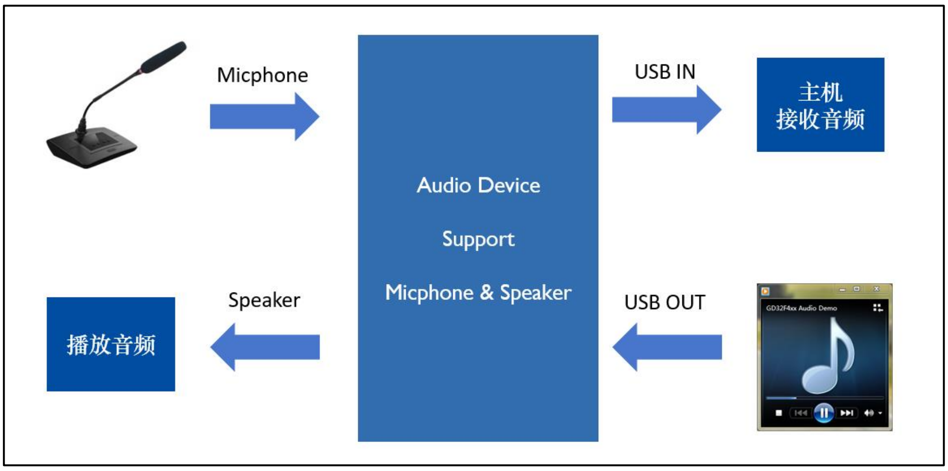
Function name	Functional description
init	initialize AUDIO required hardware resources
deinit	halt AUDIO function, release hardware resources
audio_cmd	play, stop, pause and restart command
volume_ctl	volume control
mute_ctl	mute control
periodic_tc	periodic transfer control
get_state	get AUDIO current state

AUDIO Routine operation guide

Download the audio routine to EVAL board, the newly added device is visible in device manager.

Figure 5-5. AUDIO device class





1） data OUT phase

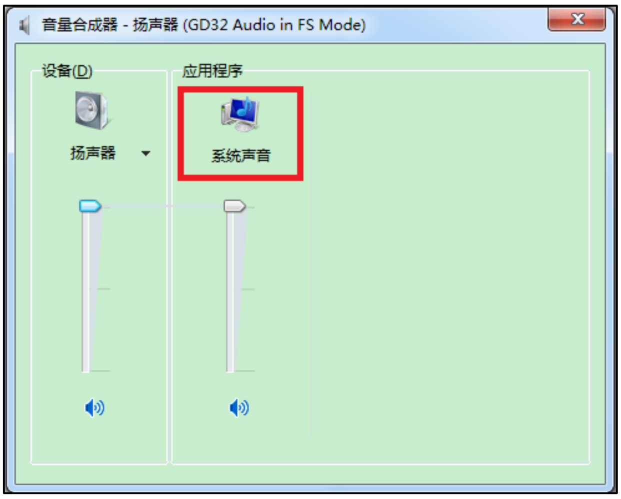
Opening audio file in host is shown in ***Figure 5-6. Audio playback file***, the playing audio file in host is heared through headphone which is connected to EVAL board.

Figure 5-6. Audio playback file

2） data IN phase

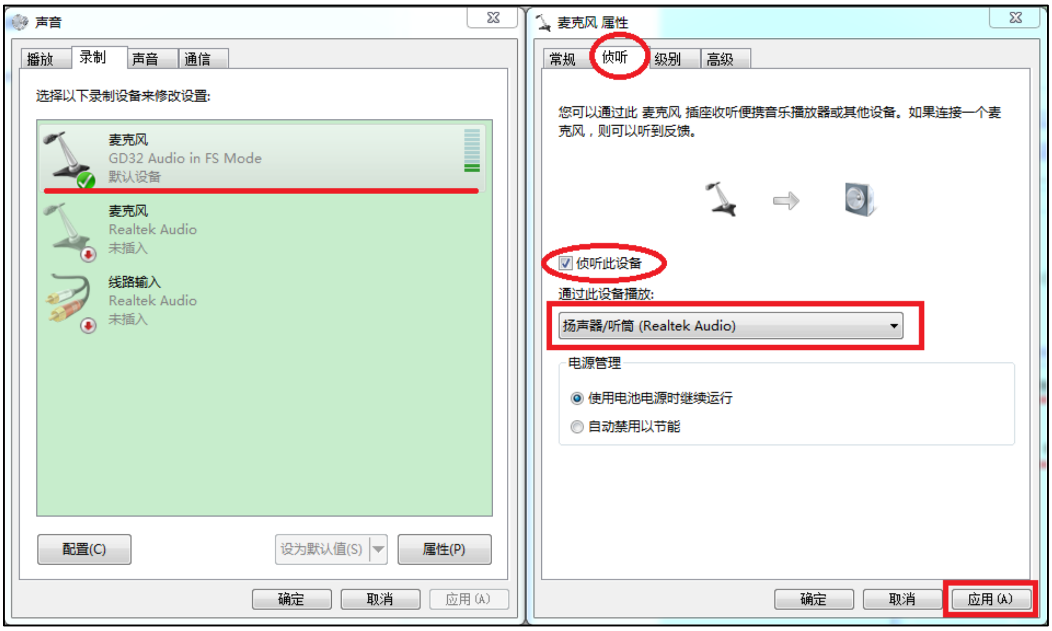
Click the loudspeaker in bottom right corner of desktop, and click volume synthesizer of loudspeaker. Then, click system sound and pop the operation interface of sound. As shown in ***Figure 5-7. Audio system sound configuration***.

Figure 5-7. Audio system sound configuration



Enter in "record" item, and double click on the microphone. In microphone attribute, select "monitor" interface, check "monitor this device" and select default playing device in "monitor" interface. After the above configuration, the sound which is transmitted from MCU device to host, could be heared by default playing device.As shown in ***Figure 5-8. Audio recording listening configuration***.

Figure 5-8. Audio recording listening configuration



5.7.2. CDC

Virtual serial port CDC rountine complies with USB communication sub class protocol, configure USB as virtual COM, which is operated as same as common COM. It is necessary to intall corresponding driver in Win7, Win8 and XP operation system, except for Win10 operation system, which has own driver.

CDC descriptor introduction

Device descriptor include CDC device VID(0x28e9) and PID(0x018a). In configuration descriptor set, include CDC corresponding descriptor item, two interface, one command interface and one data interface, command interface corresponding descriptor is shown as ***Table 5-8. CDC relevant descriptors***.

Table 5-8. CDC relevant descriptors

Descriptor name	Functional description
usb_desc_header_func	header descriptor
usb_desc_call_managment_func	communication management descriptor
usb_desc_acm_func	abstract control management descriptor
usb_desc_union_func	union function descriptor

CDC device class interface

The CDC device class interface is shown in the following structure, whose function implementation is shown in the following ***Table 5-9. CDC device class interface functions***:

```
usb_class_core cdc_class =
{
    .command     = NO_CMD,
    .alter_set   = 0U,

    .init        = cdc_acm_init,
    .deinit      = cdc_acm_deinit,

    .req_proc    = cdc_acm_req,
    .ctlx_out    = cdc_ctlx_out,

    .data_in     = cdc_acm_in,
    .data_out    = cdc_acm_out
};
```

Table 5-9. CDC device class interface functions

Function name	Functional description
cdc_acm_init	Initialize AUDIO device
cdc_acm_deinit	Deinitialize AUDIO device
cdc_acm_req	AUDIO device class request function
cdc_ctlx_out	OUT control transfer callback
cdc_acm_in	IN data transfer callback
cdc_acm_out	OUT data transfer callback

CDC device class request

The CDC contains individual device class requests as shown in the following ***Table 5-10.***

CDC device class request.

Table 5-10. CDC device class request

Request name	Functional description
SEND_ENCAPSULATED_COMMAND	Not used
GET_ENCAPSULATED_RESPONSE	Not used
SET_COMM_FEATURE	Not used
GET_COMM_FEATURE	Not used
CLEAR_COMM_FEATURE	Not used
SET_LINE_CODING	set serial port parameters
GET_LINE_CODING	get serial port parameters
SET_CONTROL_LINE_STATE	Not used
SEND_BREAK	Not used

CDC user interface

The user interface for CDC contains the following functions:

```
/* function declarations */
/* check CDC ACM is ready for data transfer */
uint8_t cdc_acm_check_ready(usb_dev *udev);
/* send CDC ACM data */
void cdc_acm_data_send(usb_dev *udev);
/* receive CDC ACM data */
void cdc_acm_data_receive(usb_dev *udev);
```

The functions are shown in the following *Table 5-11. CDC user interface functions*:

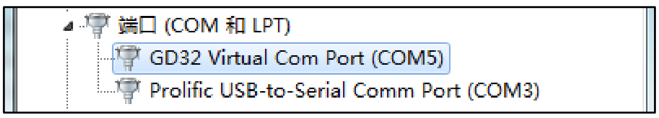
Table 5-11. CDC user interface functions

Function name	Functional description
cdc_acm_check_ready	CDC check ready for data transfer
cdc_acm_data_send	CDC data send
cdc_acm_data_receive	CDC data receive

CDC routine operation guide

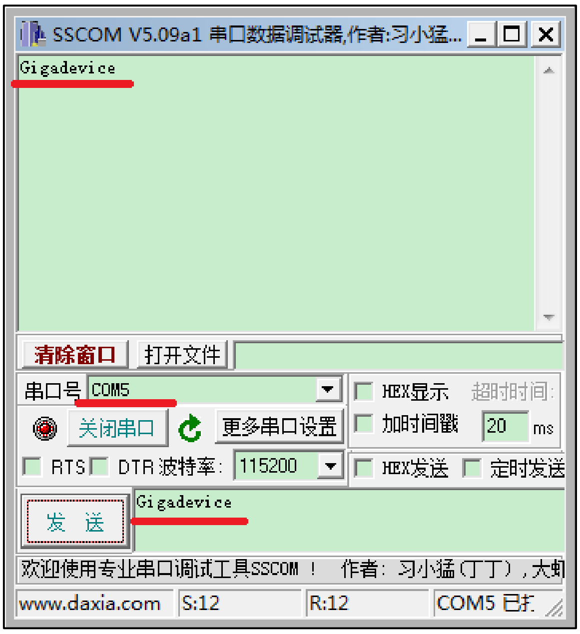
Download the CDC_ACM rountine to develop board, a newly added COM is visible in device manager.

Figure 5-9. CDC device class



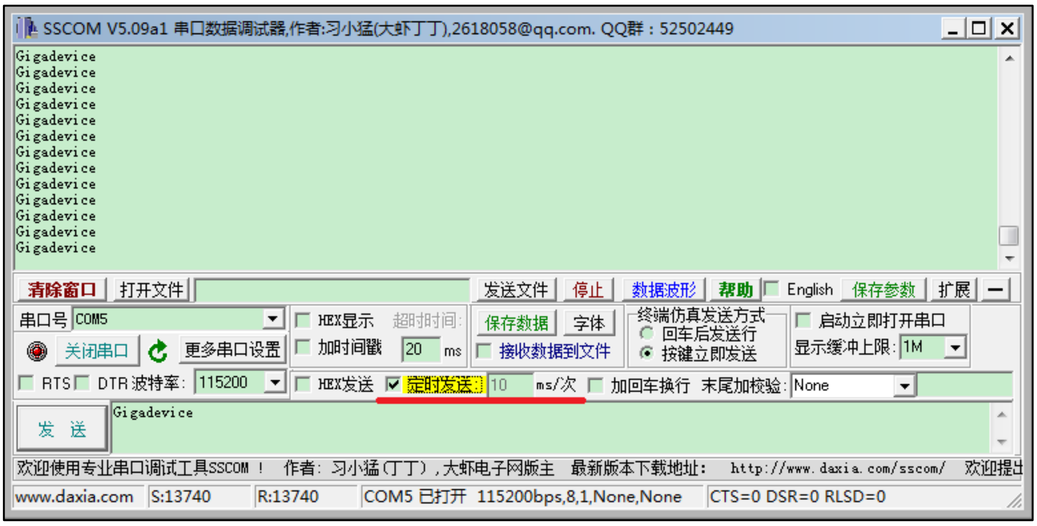
Select the newly added COM ID in serial debugging assistant, open the COM, considering the loopback function, send a charater string, and then receive the same charater string.

Figure 5-10. Virtual serial data transmitting and receiving



For mass data test, it is necessary to add send byte number, and configure Timing sending function in serial debugging assistant, which is shown in ***Figure 5-11. Virtual serial port large data transmitting and receiving***.

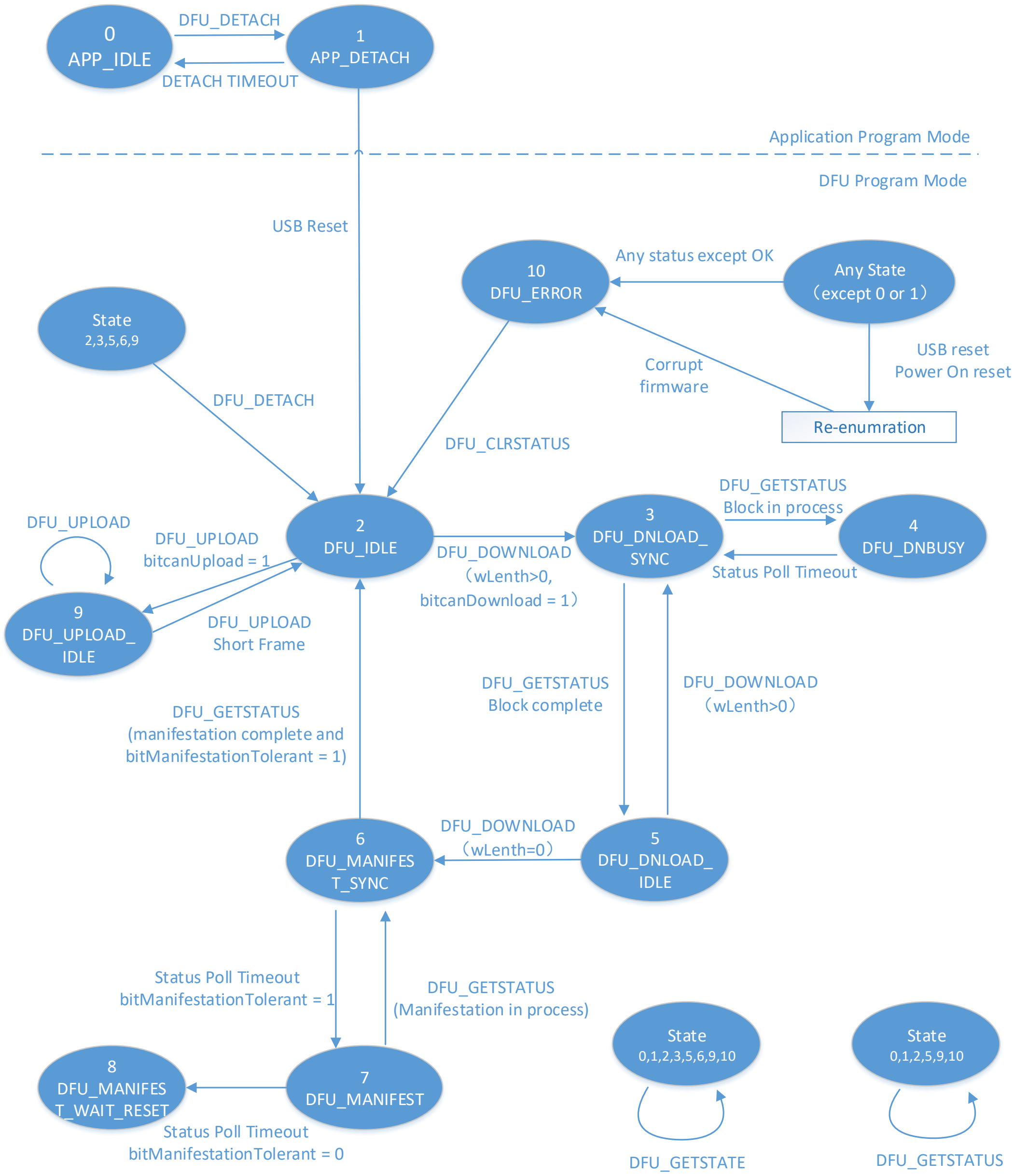
Figure 5-11. Virtual serial port large data transmitting and receiving



5.7.3. DFU

DFU is specific protocol for upgrading firmware, without non-zero endpoint, only endpoint 0 is used in data transfer course. DFU protocol flow is drived by state machine, which is shown as ***Figure 5-12. DFU state machine flow chart***.

Figure 5-12. DFU state machine flow chart



DFU descriptor introduction

Device descriptor include DFU device VID(0x28e9) and PID(0x0189). In configuration descriptor set, include DFU corresponding descriptor item, which is shown as ***Table 5-12. DFUrelevant descriptors***.

Table 5-12. DFUrelevant descriptors

Descriptor name	Functional description
usb_desc_dfu_func	DFU function descriptor

DFU device class interface

DFU device class interface is shown in below structure, and structure function is shown in

Table 5-13. DFU device class interface functions.

```
usb_class_core dfu_class = {
    .init           = dfu_init,
    .deinit         = dfu_deinit,
    .req_proc       = dfu_req_handler,
    .ctlx_in        = dfu_ctlx_in
};
```

Table 5-13. DFU device class interface functions

Function name	Functional description
dfu_init	initialize DFU device
dfu_deinit	deinitialize DFU device
dfu_req_handler	DFU device class request function
dfu_ctlx_in	IN control transfer callback

DFU device class request

DFU include device class request is shown in *Table 5-14. DFU device class request*.

Table 5-14. DFU device class request

Request name	value	Functional description
DFU_DETACH	0	DFU detach
DFU_DNLOAD	1	Download
DFU_UPLOAD	2	Upload
DFU_GETSTATUS	3	Get status
DFU_CLRSTATUS	4	Clear status
DFU_GETSTATE	5	Get state
DFU_ABORT	6	abort

DFU user interface

DFU user interface is flash operation corresponding function, which is shown in below structure.

```
dfu_mal_prop DFU_Flash_cb =
{
    (const uint8_t *)FLASH_IF_STRING,

    flash_if_init,
    flash_if_deinit,
    flash_if_erase,
    flash_if_write,
    flash_if_read,
    flash_if_checkaddr,
    60, /* flash erase timeout in ms */
    80   /* flash programming timeout in ms (80us * RAM Buffer size (1024 Bytes) */
```

```
};
```

The functions of each function are shown in the following ***Table 5-15. DFU user interface functions***:

Table 5-15. DFU user interface functions

Function name	Functional description
flash_if_init	Memory medium interface initialization
flash_if_deinit	Memory medium interface deinitialization
flash_if_erase	Memory medium erase operation
flash_if_write	Memory medium write operation
flash_if_read	Memory medium read operation
flash_if_checkaddr	Memory medium address legal check
FLASH_IF_STRING	Interface string

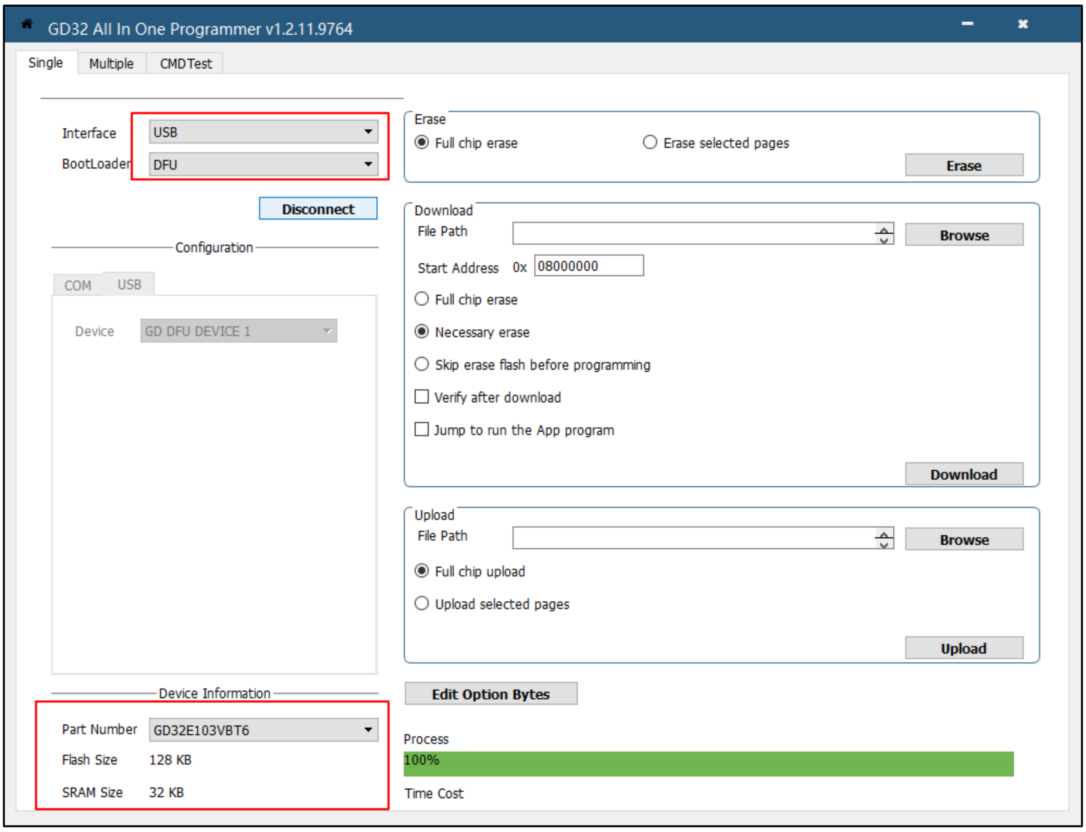
DFU Routine operation guide

Download the DFU rountine to EVAL board, a newly added device is visible in device manager.

Figure 5-13. DFU device class

Open GD32-All-In-One-Programmer software, if device is successfully connected to host, DFU interface would display connected state and display connected device type. The software function is mainly consist of download, upload and option byte operation.

Figure 5-14. All in one connection



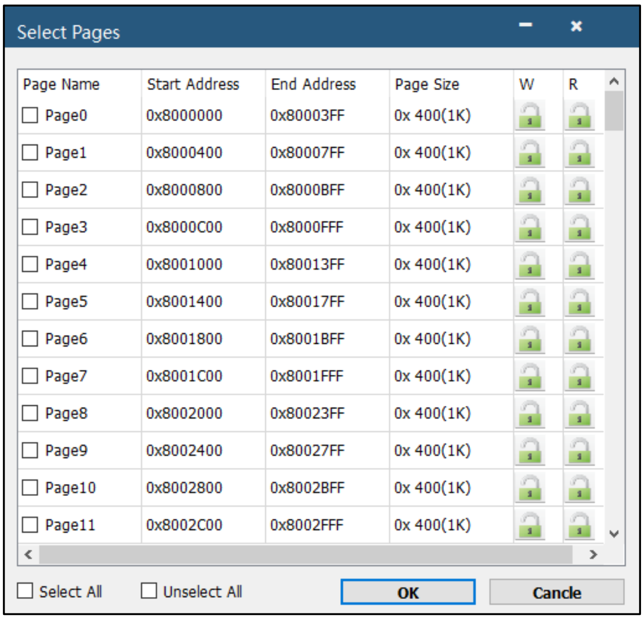
1） Download

Select the target file and configure the corresponding download address, after downloading, reset the chip, and then execute application.

2） Upload

Select the target file, click OK and then pop out “selected pages” interface, select the upload page and then get the corresponding data.

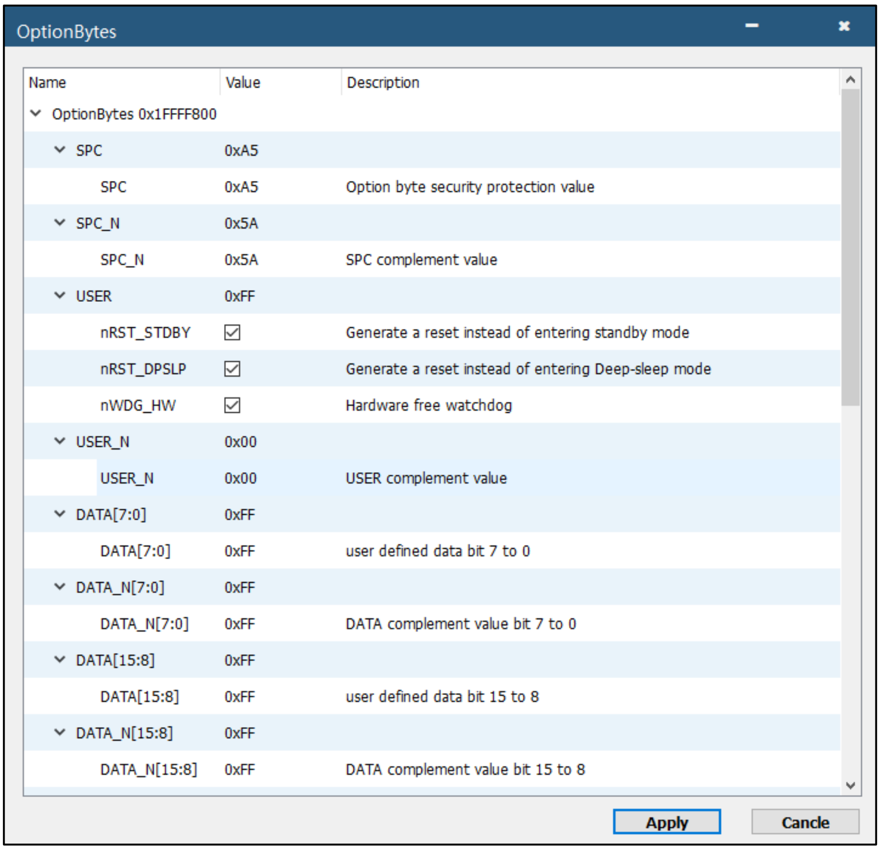
Figure 5-15. All in one uploading



3） Option Byte operation

Double click “Edit Option Bytes”, and then pop out the option byte corresponding information, which is shown as ***Figure 5-16. All in one Option Byte operation***.

Figure 5-16. All in one Option Byte operation



5.7.4. MSC

MSC device is mass storage device, include U-disk and CDROM.

MSC descriptor introduction

Device descriptor include MSC device VID(0x28e9) and PID(0x028f). In configuration descriptor set, include configure descriptor, interface descriptor and endpoint descriptor, which is shown as below.

MSC device class interface

MSC device class interface is shown in below structure, and structure function is shown in ***Table 5-16. MSC device class interface functions***.

```
usb_class_core msc_class =
{
    .init       = msc_core_init,
    .deinit     = msc_core_deinit,

    .req_proc = msc_core_req,

    .data_in    = msc_core_in,
    .data_out = msc_core_out
};
```

Table 5-16. MSC device class interface functions

Function name	Functional description
msc_core_init	Initialize MSC device
msc_core_deinit	Deinitialize MSC device
msc_core_req	MSC device class request function
msc_core_in	IN data transfer callback
msc_core_out	OUT data transfer callback

MSC device class request

MSC include device class request is shown in ***Table 5-17. MSC device class request***.

Table 5-17. MSC device class request

Request name	value	Functional description
BBB_GET_MAX_LUN	0xFE	Gets the maximum logical unit number
BBB_RESET	0xFF	Reset

MSC user interface

MSC user interface is the initialization, read, write and get information operation of memory medium, which is shown in below structure.

```
usbd_mem_cb USBD_Internal_Storage_fops =
{
    .mem_init          = STORAGE_Init,
    .mem_ready         = STORAGE_IsReady,
    .mem_protected     = STORAGE_IsWriteProtected,
    .mem_read          = STORAGE_Read,
    .mem_write         = STORAGE_Write,
    .mem_maxlun        = STORAGE_GetMaxLun,

    .mem_inquiry_data = {(uint8_t *)STORAGE_InquiryData},

    .mem_block_size    = {ISRAM_BLOCK_SIZE},
    .mem_block_len     = {ISRAM_BLOCK_NUM}
};
```

Individual functions and variable functions is shown as ***Table 5-18. MSC user interface functions.***

Table 5-18. MSC user interface functions

Function/variable name	Functional description
STORAGE_Init	Memory medium interface initialization
STORAGE_IsReady	Check whether memory medium is ready
STORAGE_IsWriteProtected	Check whether memory medium is write protected
STORAGE_Read	Memory medium read operation
STORAGE_Write	Memory medium write operation
STORAGE_GetMaxLun	Get supported logic unit number
STORAGE_InquiryData	Memory medium standard inquiry data
ISRAM_BLOCK_SIZE	Memory medium block size
ISRAM_BLOCK_NUM	Memory medium block number

The storage capacity is determined by the value of ISRAM_BLOCK_SIZE and ISRAM_BLOCK_NUM.

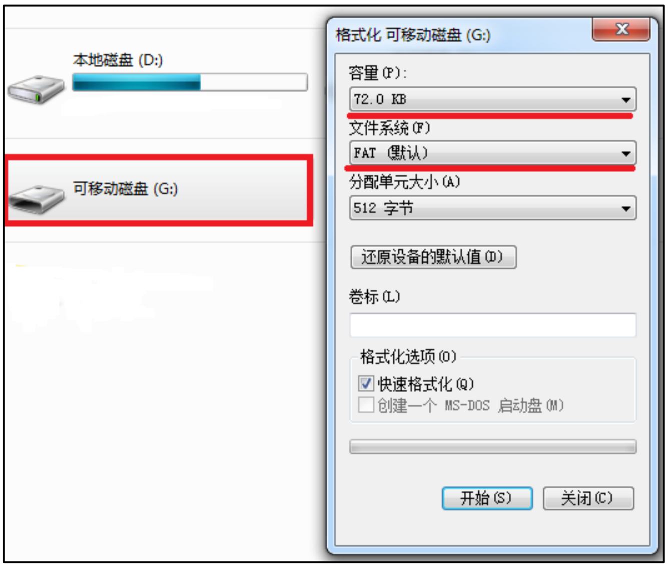
MSC routine operation guide

Download the MSC rountine to EVAL board, a newly added mass storage device is visible in device manager.

Figure 5-17. MSC device class

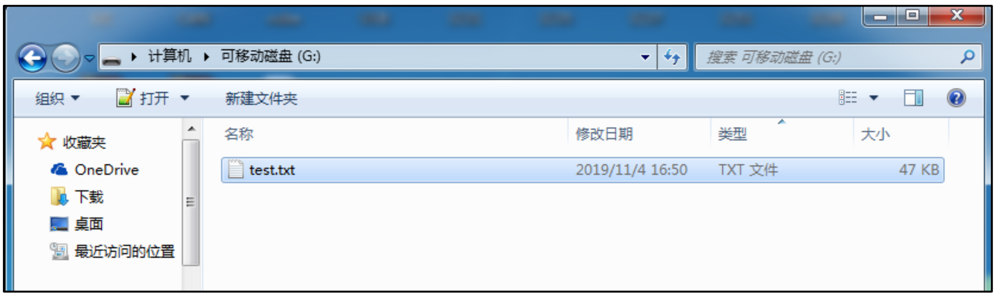
In my computer, the newly added disk is visible, because disk is lack of file system, so it is necessary to format the disk firstly, which is shown in *Figure 5-18. MSC device formatting*.

Figure 5-18. MSC device formatting



After formatting is finished, write test is operated through copy the test file to disk, after copy the file in the disk to other disk, and execute read test. Finally, comparing the write in file and read out file is to proving correctness of read and write operation.

Figure 5-19. MSC device read-write test



5.7.5. HID

HID device class is implement human-machine interaction interface, HID device has a wide usage range, not only include mouse, keyboard and touch device, but also include customed HID device.

HID descriptor introduction

Device descriptor include HID device VID(0x28e9) and PID(0x0380). In HID configuration descriptor set, include HID descriptor item and report descriptor, corresponding descriptor is shown as ***Table 5-19. HID relevant descriptors***.

Table 5-19. HID relevant descriptors

Descriptor name	Functional description
hid_vendor	HID descriptor
hid_report_desc	report descriptor

HID device class interface

HID device class interface is shown in below structure, and structure function is shown in below ***Table 5-20. HID device class interface functions***.

```
usb_class_core usbd_hid_cb = {
    .command        = NO_CMD,
    .alter_set      = 0U,
    .init           = hid_init,
    .deinit         = hid_deinit,
    .req_proc       = hid_req,
    .data_in        = hid_data_in
};
```

Table 5-20. HID device class interface functions

Function name	Functional description
hid_init	Initialize HID device
hid_deinit	Deinitialize HID device
hid_req	HID device class request function
hid_data_in	IN data transfer callback

HID device class request

HID include device class request is shown in ***Table 5-21. HID device class request***.

Table 5-21. HID device class request

Request name	value	Functional description
GET_REPORT	0x01	Get report
GET_IDLE	0x02	Get idle

Request name	value	Functional description
GET_PROTOCOL	0x03	Get protocol
SET_REPORT	0x09	Set report
SET_IDLE	0x0A	Set idle
SET_PROTOCOL	0x0B	Set protocol

HID user interface

HID user interface is enumerated as keyboard, which is shown in below structure.

```
hid_fop_handler fop_handler = {
    .hid_itf_config = key_config,
    .hid_itf_data_process = hid_key_data_send
};
```

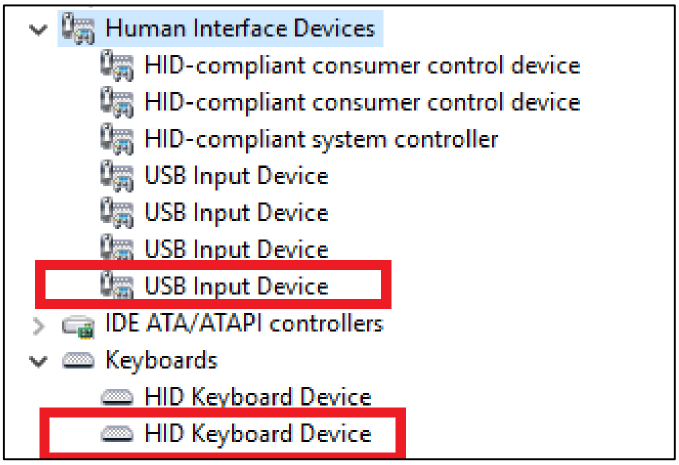
Table 5-22. HID user interface functions

Function/variable name	Functional description
key_config	Key configuration
hid_key_data_send	Send key value

HID routine operation guide

Download the HID rountine to EVAL board, a newly added HID device is visible in device manager.As shown in ***Figure 5-20. HID device class***.

Figure 5-20. HID device class



Press Wakeup key, output “b”; Press Tamper key, output “a”; Press User key, output “c”. The below step show how to verify USB remote wakeup function.

1. configure PC to be sleep mode;

2. wait PC host to be sleep mode completely;

3. press wakeup key;

4. if the host is waked up, indicate that remote wakeup function is successful, otherwise it is

fail.

5.7.6. USB printer

Printer descriptor introduction

Device descriptor include Printer device VID(0x28e9) and PID(0x028d). In printer configuration descriptor set, include configuration, interface and endpoint descriptor, corresponding descriptor is shown as below.

Printer device class interface

Printer device class interface is shown in below structure, and structure function is shown in below ***Table 5-23. printer device class interface function***.

```
usb_class_core usbd_printer_cb = {
    .init           = printer_init,
    .deinit         = printer_deinit,
    .req_proc       = printer_req,
    .data_in        = printer_in,
    .data_out       = printer_out
};
```

Table 5-23. printer device class interface function

Function name	Functional description
printer_init	Initialize printer device
printer_deinit	Deinitialize printer device
printer_req	Printer device class request function
printer_in	IN data transfer callback
printer_out	OUT data transfer callback

Printer device class request

Priner device include device class request is shown in ***Table 5-24. printer device class request***.

Table 5-24. printer device class request

Request name	value	Functional description
GET_DEVICE_ID	0x00	Get device ID
GET_PORT_STATUS	0x01	Get port status
SOFT_RESET	0x02	Software reset

Printer user interface

Currently, printer rountine merely implement enumeration, data transfer is subject to printer hardware, and without corresponding user interface, which is used to implement data transfer function.

Printer routine operation guide

Download the Printer rountine to EVAL board, a newly added printer device is visible in device manager.As shown in ***Figure 5-21. printer device class***.

Figure 5-21. printer device class

6. USBFS Host Library

6.1. Host Library Configuration

6.1.1. usbh_conf.h

File configuration item is shown as below.

```
#define USBH_MAX_EP_NUM                      2
#define USBH_MAX_INTERFACES_NUM              2
#define USBH_MAX_ALT_SETTING                 2
#define USBH_MAX_SUPPORTED_CLASS             2

#define USBH_DATA_BUF_MAX_LEN                0x200
#define USBH_CFGSET_MAX_LEN                  0x200
```

Each configuration is defined as ***Table 6-1. usbh_conf.h Configuration description***.

Table 6-1. usbh_conf.h Configuration description

Configuration name	Functional description
USBH_MAX_EP_NUM	Maximum number of endpoints
USBH_MAX_INTERFACES_NUM	Maximum number of interfaces
USBH_MAX_ALT_SETTING	Maximum number of alternate interfaces
USBH_MAX_SUPPORTED_CLASS	Maximum number of supported device classes
USBH_DATA_BUF_MAX_LEN	Maximum length of data buffer
USBH_CFGSET_MAX_LEN	Maximum length of configuration descriptor set

6.1.2. usb_conf.h

```
#ifdef USB_FS_CORE
    #define USB_RX_FIFO_FS_SIZE                  128
    #define USB_HTX_NPFIFO_FS_SIZE               96
    #define USB_HTX_PFIFO_FS_SIZE                96
#endif

#ifdef USB_HS_CORE
    #define USB_RX_FIFO_HS_SIZE                  512
    #define USB_HTX_NPFIFO_HS_SIZE               256
    #define USB_HTX_PFIFO_HS_SIZE                256

    #ifdef USE_ULPI_PHY
        #define USB_ULPI_PHY_ENABLED
    #endif
```

```
    #ifdef USE_EMBEDDED_PHY
        #define USB_EMBEDDED_PHY_ENABLED
    #endif

//      #define USB_HS_INTERNAL_DMA_ENABLED
#endif

#define USB_SOF_OUTPUT                                          0
#define USB_LOW_POWER                                            0

//#define USB_LOW_PWR_ENABLE

/****************** USB OTG MODE CONFIGURATION ********************************/
#define USE_HOST_MODE
//#define USE_DEVICE_MODE
//#define USE_OTG_MODE
```

Table 6-2. usb_conf.h Configuration description

Configuration name	Functional description
USB_RX_FIFO_FS_SIZE	Received FIFO size
USB_HTX_NPFIFO_FS_SIZE	Non periodic transmit FIFO size
USB_HTX_PFIFO_FS_SIZE	periodic transmit FIFO size
USB_ULPI_PHY_ENABLED	Enable ULPI PHY
USB_EMBEDDED_PHY_ENABLED	Enable embedded PHY
USB_SOF_OUTPUT	Enable SOF output (PA8 pin)
USB_LOW_POWER	Enable low power mode
USB_LOW_PWR_ENABLE	Enable VBUS SENSING
USE_HOST_MODE	Host mode
USE_DEVICE_MODE	Device mode
USE_OTG_MODE	OTG mode

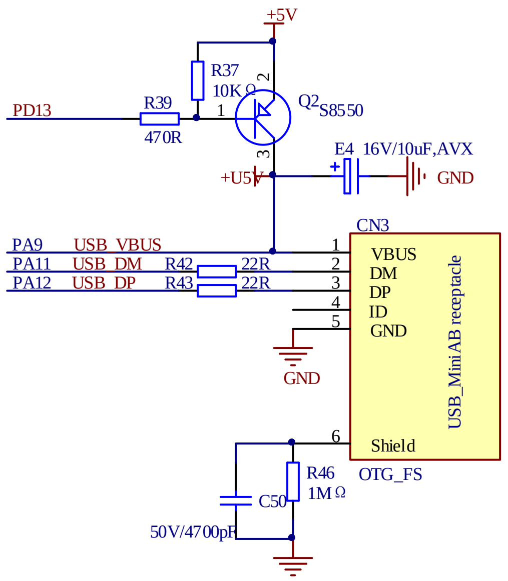
Note: only merely one of three modes could be selected.

6.2. Host VBUS Configuration

There is two type circuits for USB host in GD32 EVAL board.

1, control VBUS through building triode circuit (include F10X/F20X/F30X/E103/F450Z-EVAL)

Figure 6-1. Construct circuit through triode to control VBUS



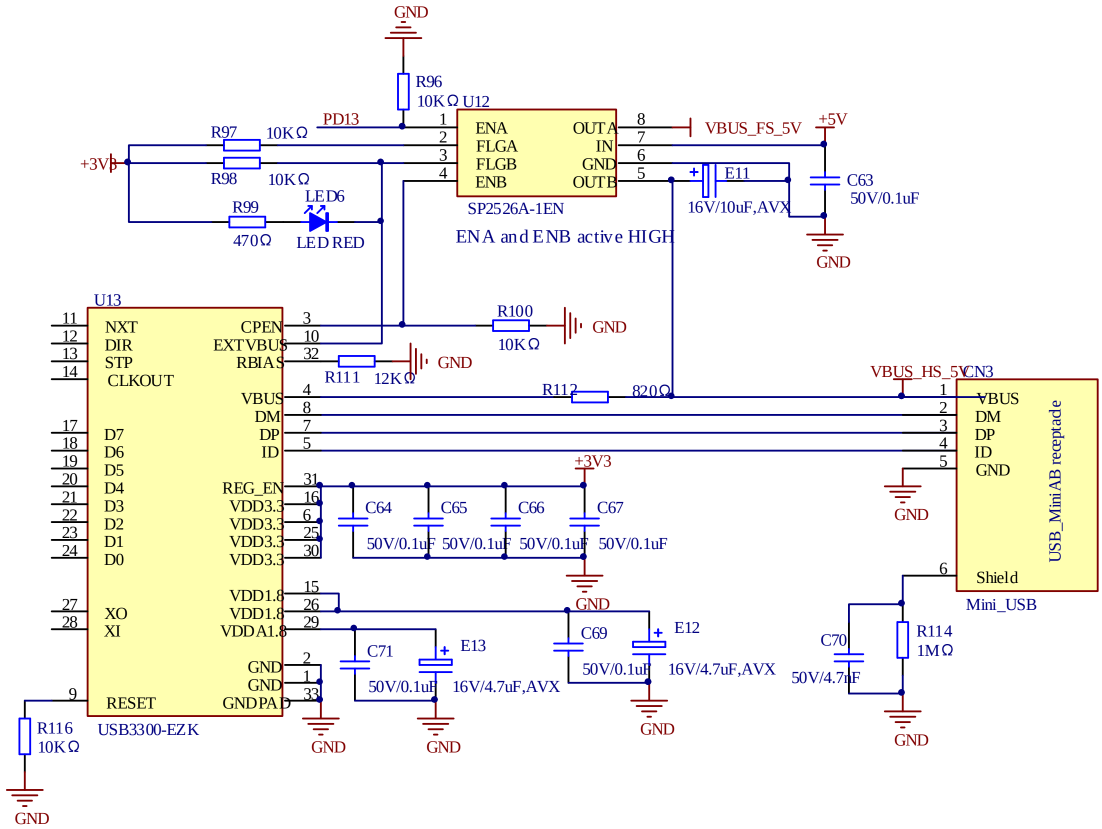
As shown in ***Figure 6-1. Construct circuit through triode to control VBUS***, PD13 is configured to be GPIO open drain mode(OD).

Enable USB VBUS output 5V: PD13 output low voltage(0)

Disable USB VBUS output 5V: PD13 output high voltage(1)

2, control VBUS through logic chip circuit(F450I-EVAL)

Figure 6-2. Control VBUS by Logic Chip Circuit



As shown in *Figure 6-2. Control VBUS by Logic Chip Circuit*, PD13 is configured to be GPIO push pull output mode(PP).

Enable USB VBUS output 5V: PD13 output high voltage(1)

Disable USB VBUS output 5V: PD13 output low voltage(0)

6.3. Interrupt handling

USBFS host interface global interrupt is shown in *Table 6-3. USBFS host interrupt*, every interrupt flag corresponds to one interrupt handler item, such as RXFNEIF, NPTXFEIF and PTXFEIF flag. In USBFS host interface, receiving data is based on RXFNEIF interrupt flag, sending data is based on NPTXFEIF and PTXFEIF interrupt flag.

Table 6-3. USBFS host interrupt

Interrupt Flag	Description	Operation Mode
WKUPIF	Wakeup interrupt	Host or device mode
SEIF	Session interrupt	Host or device mode
DISCIF	Disconnect interrupt flag	Host Mode
IDPSC	ID pin status change	Host or device mode
LPMIF	LPM interrupt flag	Host or device mode
PTXFEIF	Periodic Tx FIFO empty interrupt flag	Host Mode

Interrupt Flag	Description	Operation 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
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

Data receiving processing is mainly implemented in below function

```
static uint32_t usbh_int_rxfifonoempty (usb_core_driver *udev)
{
    uint32_t count = 0U;

    __IO uint8_t pp_num = 0U;
    __IO uint32_t rx_stat = 0U;

    /* disable the RX status queue level interrupt */
    udev->regs.gr->GINTEN &= ~GINTEN_RXFNEIE;

    rx_stat = udev->regs.gr->GRSTATP;
    pp_num = (uint8_t)(rx_stat & GRSTATRP_CNUM);

    switch ((rx_stat & GRSTATRP_RPCKST) >> 17U) {
    case GRXSTS_PKTSTS_IN:
        count = (rx_stat & GRSTATRP_BCOUNT) >> 4U;

        /* read the data into the host buffer. */
        if ((count > 0U) && (NULL != udev->host.pipe[pp_num].xfer_buf)) {
            (void)usb_rxfifo_read (&udev->regs, udev->host.pipe[pp_num].xfer_buf,
(uint16_t)count);       // read FIFO

            /* manage multiple transfer packet */
            udev->host.pipe[pp_num].xfer_buf += count;
            udev->host.pipe[pp_num].xfer_count += count;

            udev->host.backup_xfercount[pp_num] = udev->host.pipe[pp_num].xfer_count;

            if (udev->regs.pr[pp_num]->HCHLEN & HCHLEN_PCNT) {
                /* re-activate the channel when more packets are expected */
```

```
                __IO uint32_t pp_ctl = udev->regs.pr[pp_num]->HCHCTL;

                pp_ctl |= HCHCTL_CEN;
                pp_ctl &= ~HCHCTL_CDIS;

                udev->regs.pr[pp_num]->HCHCTL = pp_ctl;
            }
        }
        break;

    case GRXSTS_PKTSTS_IN_XFER_COMP:
        break;

    case GRXSTS_PKTSTS_DATA_TOGGLE_ERR:
        count = (rx_stat & GRSTATRP_BCOUNT) >> 4U;

        while (count > 0U) {
            rx_stat = udev->regs.gr->GRSTATP;
            count--;
        }
        break;

    case GRXSTS_PKTSTS_CH_HALTED:
        break;

    default:
        break;
    }

    /* enable the RX status queue level interrupt */
    udev->regs.gr->GINTEN |= GINTEN_RXFNEIE;

    return 1U;
}
```

Data transmitting processing is mainly implemented in below function

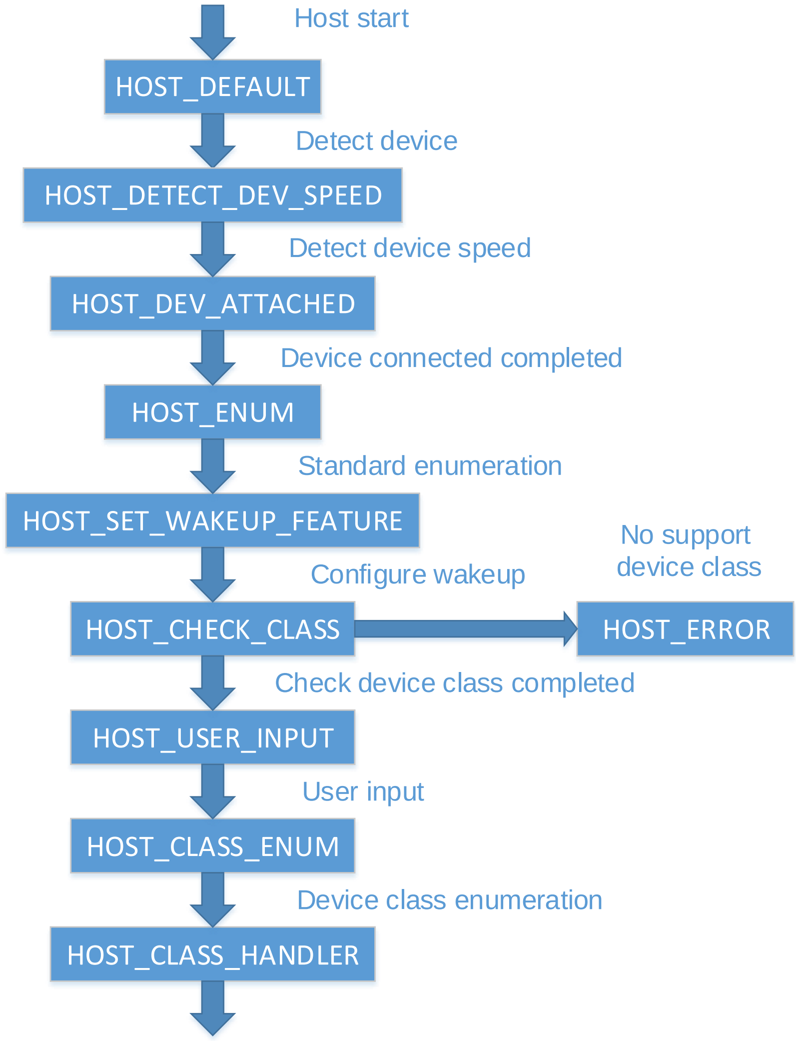
```
static uint32_t usbh_int_txfifoempty (usb_core_driver *udev, usb_pipe_mode pp_mode)
{
    uint8_t pp_num = 0U;
    uint16_t word_count = 0U, len = 0U;
    __IO uint32_t *txfiforeg = 0U, txfifostate = 0U;

    if (PIPE_NON_PERIOD == pp_mode) {
        txfiforeg = &udev->regs.gr->HNPTFQSTAT;
```

```
    } else if (PIPE_PERIOD == pp_mode) {
        txfiforeg = &udev->regs.hr->HPTFQSTAT;
    } else {
        return 0U;
    }

    txfifostate = *txfiforeg;

    pp_num = (uint8_t)((txfifostate & TFQSTAT_CNUM) >> 27U);

    word_count = (uint16_t)(udev->host.pipe[pp_num].xfer_len + 3U) / 4U;

    while (((txfifostate & TFQSTAT_TXFS) >= word_count) && (0U !=
udev->host.pipe[pp_num].xfer_len)) {
        len = (uint16_t)(txfifostate & TFQSTAT_TXFS) * 4U;

        if (len > udev->host.pipe[pp_num].xfer_len) {
            /* last packet */
            len = (uint16_t)udev->host.pipe[pp_num].xfer_len;

            if (PIPE_NON_PERIOD == pp_mode) {
                udev->regs.gr->GINTEN &= ~GINTEN_NPTXFEIE;
            } else {
                udev->regs.gr->GINTEN &= ~GINTEN_PTXFEIE;
            }
        }

        word_count = (uint16_t)((udev->host.pipe[pp_num].xfer_len + 3U) / 4U);
        usb_txfifo_write (&udev->regs, udev->host.pipe[pp_num].xfer_buf, pp_num, len); // write
FIFO

        udev->host.pipe[pp_num].xfer_buf += len;
        udev->host.pipe[pp_num].xfer_len -= len;
        udev->host.pipe[pp_num].xfer_count += len;

        txfifostate = *txfiforeg;
    }

    return 1U;
}
```

6.4. State Machine Process

Based on the below state machine, USB implement device operation, such as connecting, detecting and enumeration. state machine is loop executing in main function.

Figure 6-3. USB host state machine



6.5. USB Host Library User Interface

USB host user interface define the below structure.

```
/* points to the DEVICE_PROP structure of current device */
usbh_user_cb usr_cb =
{
       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_desc_available,
    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,
    usbh_usr_msc_application,
    usbh_user_device_not_supported,
    usbh_user_unrecovered_error
};
```

The functions of each function are described in ***Table 6-4. USB host library user interface function***.

Table 6-4. USB host library user interface function

Function name	Functional description
usbh_user_init	initialize user operation in host mode
usbh_user_deinit	configure user as default
usbh_user_device_connected	user operation of USB connection
usbh_user_device_reset	user operation of device resetting
usbh_user_device_disconnected	user operation of USB disconnection
usbh_user_over_current_detected	user operation of device overloading
usbh_user_device_speed_detected	user operation of detecting device speed
usbh_user_device_desc_available	user operation when device descriptor is available
usbh_user_device_address_assigned	user operation when device is successfully configured
usbh_user_configuration_descavailable	user operation when configuration descriptor is available
usbh_user_manufacturer_string	user operation when vendor string is available
usbh_user_product_string	user operation when product string is available
usbh_user_serialnum_string	user operation when serial number is exist
usbh_user_enumeration_finish	user operation when enumeration is accomplished
usbh_user_userinput	user operation when entering user state
usbh_usr_xxx_application	user application code callback function
usbh_user_device_not_supported	user operation when device is not supported
usbh_user_unrecovered_error	user operation when unrecoverable error happen

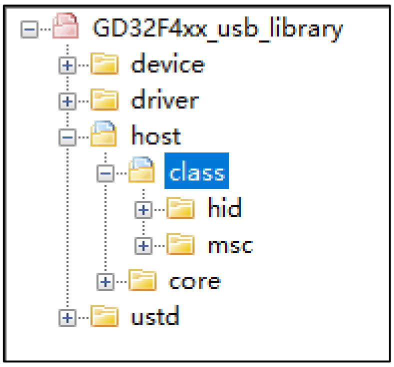
6.6. USB Host Library Device Class Interface

USB device class interface is implemented through the below structure.

```
/* device class callbacks */
typedef struct
{
    uint8_t          class_code;          /*!< USB class type */

    usbh_status (*class_init)         (struct _usbh_host *phost);
    void           (*class_deinit)     (struct _usbh_host *phost);
    usbh_status (*class_requests)  (struct _usbh_host *phost);
    usbh_status (*class_machine)   (struct _usbh_host *phost);
    usbh_status (*class_sof)           (struct _usbh_host *uhost);

    void              *class_data;
} usbh_class;
```

The structure initialization is implemented separately by each device class. Interface file of each device class is reserved in host/class path device class folder.

Figure 6-4. Host device class interface file path



The structure is implemented to operate initialization, deinitialization, device class request and data transfer.

6.6.1. HID device class

The HID device is initialized as follows:

```
usbh_class usbh_hid =
{
    USB_HID_CLASS,
    usbh_hid_itf_init,
    usbh_hid_itf_deinit,
    usbh_hid_class_req,
    usbh_hid_handle,
```

```
    usbh_hid_sof
};
```

The initialization function of structure is shown in usbh_hid_core.c file, except for include structure initialization, and include other HID device class function and corresponding function, shown in ***Table 6-5. HID host class library function***.

Table 6-5. HID host class library function

Device class	File name	Function name	Description
HID host class	usbh_hid_core.h/c	usbh_get_report	get HID report
		usbh_set_report	set HID report
		usbh_hid_device_type_get	get HID device function
		usbh_hid_poll_interval_get	get HID device poll time
		usbh_hid_fifo_read	read data from FIFO
		usbh_hid_fifo_write	write data to FIFO
		usbh_hid_fifo_init	initialize FIFO
		usbh_hiddesc_parse	parse the HID descriptor
		usbh_hid_itf_deinit	de-initialize the host pipes used for the HID class
		usbh_hid_itf_init	initialize the hid class
		usbh_hid_class_req	handle HID class requests for HID class
		usbh_hid_handle	manage state machine for HID data transfers
		usbh_hid_reportdesc_get	send get report descriptor command to the device
		usbh_hid_sof	manage the SOF process
		usbh_hid_desc_get	send get HID descriptor command to the device
		usbh_set_idle	set idle state
		usbh_set_protocol	set protocol state
	usbh_hid_keybd.h/c	usbh_hid_keybd_init	initialize the keyboard function
		usbh_hid_keybd_info_get	get keyboard information
		usbh_hid_ascii_code_get	get the ascii code of hid
		usbh_hid_keybrd_machine	keyboard machine
		usbh_hid_keybrd_decode	decode keyboard information
	usbh_hid_mouse.h/c	usbh_hid_mouse_init	initialize mouse function
		usbh_hid_mouse_info_get	get mouse information
		usbh_hid_mouse_machine	mouse machine
		usbh_hid_mouse_decode	decode mouse data
	usbh_hid_parser.h/c	hid_item_read	read a hid report item
		hid_item_write	write a hid report item

6.6.2. MSC device class

The MSC device is initialized as follows:

```
usbh_class usbh_msc =
{
    USB_CLASS_MSC,
    usbh_msc_itf_init,
    usbh_msc_itf_deinit,
    usbh_msc_req,
    usbh_msc_handle,
};
```

The initialization function of structure is shown in usbh_msc_core.c file, except for include structure initialization, and include other MSC device class function and corresponding function, shown in ***Table 6-6. MSC host class library function***.

Table 6-6. MSC host class library function

Device class	File name	Function name	Description
MSC host class	usbh_msc_bbb.h/c	usbh_msc_init	initialize the mass storage parameters
		usbh_msc_bot_process	manage the different states of BOT transfer and updates the status to upper layer
		usbh_msc_bot_abort	manages the different error handling for stall
		usbh_msc_bot_reset	reset msc bot request
		usbh_msc_csw_decode	decode the CSW received by the device and updates the same to upper layer
	usbh_msc_core.h/c	usbh_msc_lun_info_get	get msc logic unit information
		usbh_msc_read	msc read interface
		usbh_msc_write	msc write interface
		usbh_msc_itf_deinit	de-initialize interface by freeing host channels allocated to interface
		usbh_msc_itf_init	interface initialization for MSC class
		usbh_msc_req	initialize the MSC state machine
		usbh_msc_handle	MSC state machine handler
		usbh_msc_maxlun_get	get max lun of the mass storage device
		usbh_msc_rdwr_process	mass storage device read and write process
	usbh_msc_sc	usbh_msc_scsi_inquiry	send 'Inquiry' command to the

Device class	File name	Function name	Description
	si.c		device
		usbh_msc_test_unitready	send 'Test unit ready' command to the device
		usbh_msc_read_capacity10	send the read capacity command to the device
		usbh_msc_mode_sense6	send the mode sense6 command to the device
		usbh_msc_request_sense	send the Request Sense command to the device
		usbh_msc_write10	send the write10 command to the device
		usbh_msc_read10	send the read10 command to the device
	usbh_msc_fatfs.c	disk_initialize	initialize disk driver
		disk_status	get disk status
		disk_read	read sectors
		disk_write	write sectors
		disk_ioctl	I/O control function
		get_fattime	get fat time

6.7. USB Host Library Routine

6.7.1. HID HOST

HID host rountine could be used to identify the keyboard and mouse, enumeration course and data transfer phase is displayed in display screen.

Firstly, insert cable into USB connector, and then download HID_Host program file to EVAL board and run the application.

If one mouse is connected, mouse enumeration information could be displayed on the display screen. Depending on the tips of display screen, firstly, the inserted device could be viewed as mouse, and then move mouse, the mouse position and key pressing state could be displayed on the display screen.

If one keyboard is connected, keyboard enumeration information could be displayed on the display screen. Depending on the tips of display screen, firstly, the inserted device could be viewed as keyboard, and then press keyboard, the corresponding character could be displayed on the display screen.

Figure 6-5. Hid host routine operation diagram

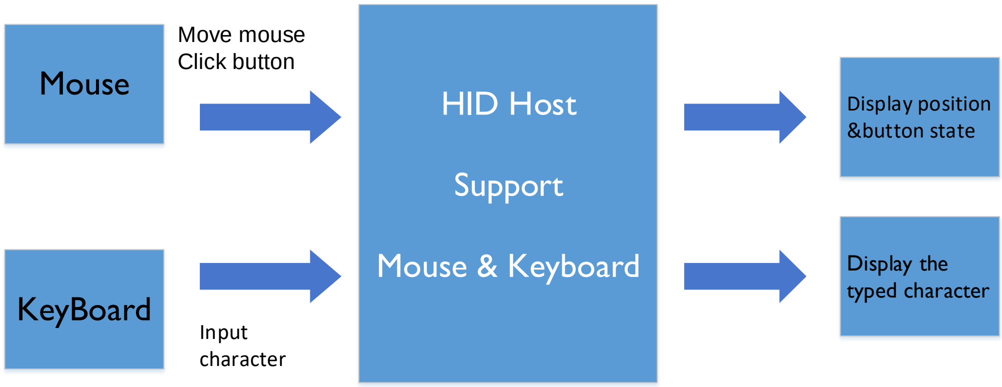
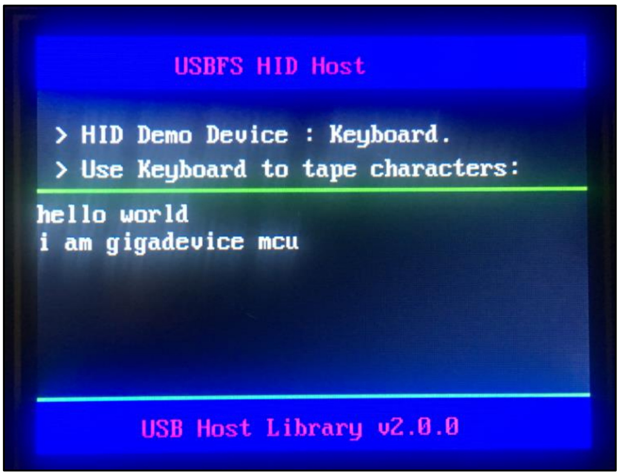


Figure 6-6. Routine for mouse-over display of HID host

Figure 6-7. Routine for HID host keyboard display



6.7.2. MSC HOST

MSC host rountine could be used to identify the U-disk, enumeration course and data transfer phase is displayed in display screen.

The operation steps of MSC host rountine is shown as below figure, firstly, insert OTG cable into USB connector, and then download MSC_Host program file to EVAL board and run the application.

If one U-disk is connected, U-disk enumeration information could be displayed on the display screen. Firstly, press USER key, U-disk information could be displayed on the display screen. sceondly, press TAMPER key, U-disk root contents could be displayed; Then, press WAKEUP key, write the file to U-disk; finally, the information, which indicate MSC host routine is done, is displayed in display screen.

Figure 6-8. MSC host routine operation steps

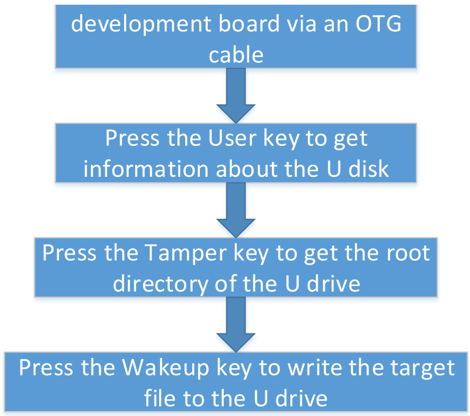
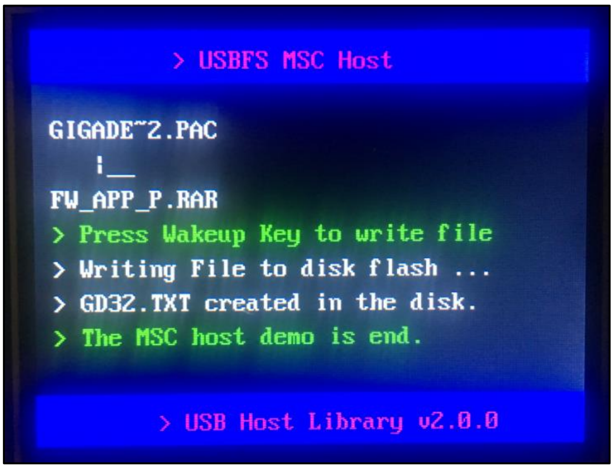


Figure 6-9. MSC host routine display



7. Revision history

Table 7-1. Revision history

Revision No.	Descriptor	Date
1.0	Initial Release	Mar.28, 2022

Important Notice