GigaDevice Semiconductor Inc.

GD32 USBD Firmware Library User Guide

Application Note

AN049

Table of Contents

List of Figures

List of Tables

1. Introduction

Based on the structure of the GD32 MCU universal serial bus full-speed device interface, this article analyzes the firmware library architecture of the USBD module, and briefly describes the functions of the firmware library functions. Through analysising specific application examples，this article clarify the realization process of some USBD device classes and provide reference for customers' follow-up development.

The article contain two section, description of firmware library and description of protocol and routines. Section description of firmware library include main.c file and function description and usbd_driver bottom layer file and library function description. In section description of application protocol and routines, according to the USB protocol, the GD32 MCU USBD module support four types of data transfer: interrupt transfer, bulk transfer, control transfer and isochronous transfer, as shown in ***Table 1-1. The example of USBD***. The application protocol, descriptor, application class request, data transmission and DEMO result of USBD device is shown by successively introducing HID device, CDC device, DFU device and UAC device in the article.

Table 1-1. The example of USBD

DEMO name	USB transfer	Description
standard_hid_keyboard	Interrupt Transfer	Enumeration for keyboard, print characters
cdc_acm	Bulk Transfer	Enumeration for Virtual COM, Tx/Rx data
dev_firmware_update	Control Transfer	Enumerated as DFU device, upgrade firmware
audio_headphone	Isochronous Transfer	Enumeration for UAC device, play music

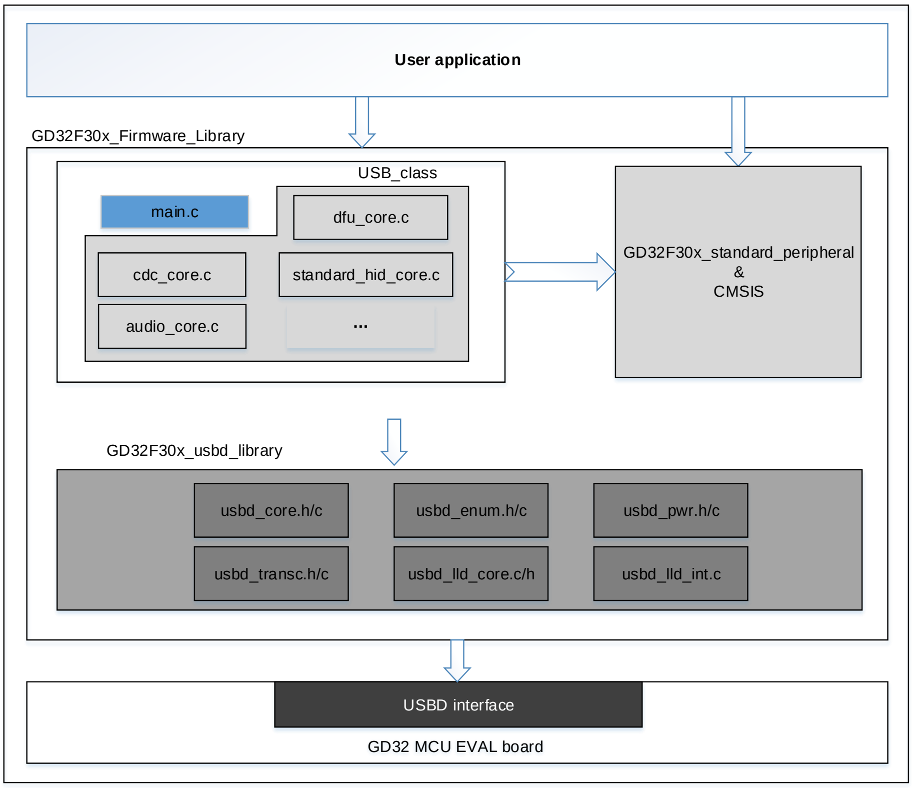
The article applicable product is shown in ***Table 1-2. Applicable product***, in this article, taking GD32F303xx as an example, GD firmware library and application examples of other product series are similar to GD32F303xx series.

Table 1-2. Applicable product

Product type	Product series
MCU	GD32F103xx series
	GD32F150xx series
	GD32F303xx series
	GD32E503xx series
	GD32EPRTxx series
	GD32L23x series

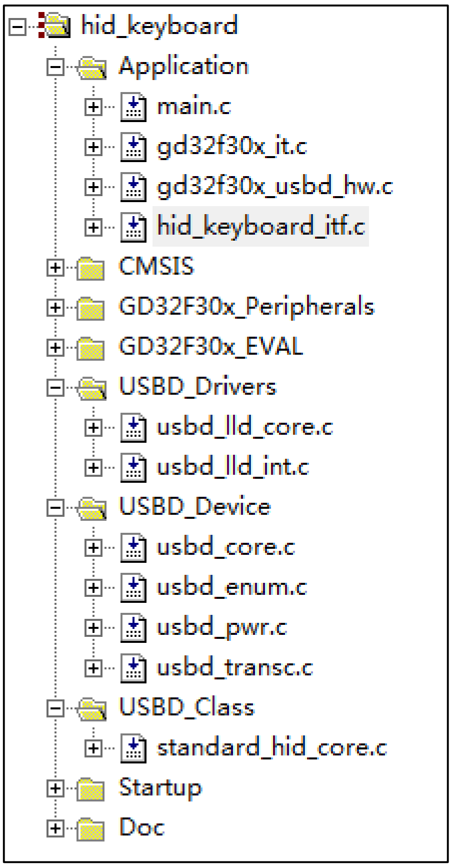
2. Description of firmware Library

Figure 2-1. USBD device firmware library schematic diagram



The GD32F30x firmware library architecture for the USBD device is shown in ***Figure 2-1. USBD device firmware library schematic diagram.*** The user application calls the interface in the firmware library of GD32 full speed USB device to realize the communication between the USB device and the host, and the lowest level of the architecture is the hardware layer of the GD32MCU development board. GD32 full speed USB device firmware library is divided into two layers. The top layer is the application interface layer, which users can modify, including main.c and USB related device class drivers. The bottom layer is the USBD device driver layer, which is not recommended to be modified, including the realization of USB communication protocol and USBD module operation.

Figure 2-2. USBD device firmware library folder tree structure diagram



Take the project structure of HID keyboard as example, which is shown in ***Figure 2-2. USBD device firmware library folder tree structure diagram***. Except common peripheral library、startup files and development board hardware library files, the USBD project need to call the underlying files of the USBD firmware library, such as usbd_lld_core.c and usbd_enum.c file， which are relatively fixed, and users are not recommended to modify them. For interface layer file, such as standard_hid_core.c and main.c file, users could modify the file according to the actual requirement of the application.

2.1. main.c file and function description

Table 2-1. Code table main function

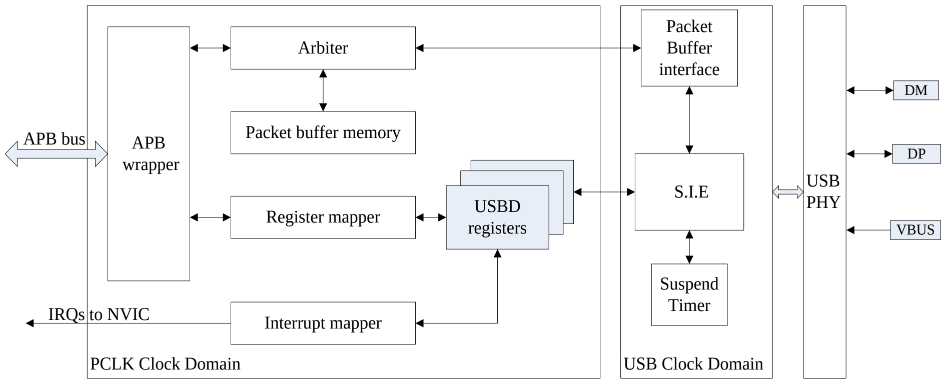
```
int main(void)
{
    /* system clocks configuration */
    rcu_config();

    /* GPIO configuration */
    gpio_config();

    hid_itfop_register (&usb_hid, &fop_handler);
```

```
        /* USB device configuration */
        usbd_init(&usb_hid, &hid_desc, &hid_class);

        /* NVIC configuration */
        nvic_config();

        usbd_connect(&usb_hid);

        while(USBD_CONFIGURED != usb_hid.cur_status){
        }

        while (1) {
            fop_handler.hid_itf_data_process(&usb_hid);
        }
}
```

As shown in the ***Table 2-1. Code table main function***, the project needs to configure clock, USBD-related pins, interrupt priority and USBD module initialization in the user's main function. After executing the usbd_connect function, the MCU needs to wait for the enumeration interaction between host and device to be completed, then the MCU executes the operation of the relevant application.

2.1.1. RCU configuration

Figure 2-3. USBD RCU domain



As shown in the ***Figure 2-3. USBD RCU domain***, USBD register configuration and other operations are completed under the PCLK clock domain, while data interaction between the USBD device and the host should be completed in the USBD clock domain (48MHz clock).

Table 2-2. USBD system clock

Product Series	System Clock	Frequency Factor
GD32F103xx	48/72/96 MHz	1 / 1.5 / 2 frequency division
GD32F150xx	48/72 MHz	1 / 1.5 frequency division
GD32F303xx	48/72/96/120 MHz	1 / 1.5 / 2 / 2.5 frequency division

GD32E503xx / GD32EPRTxx	48/72/96/120/144/168 MHz	1 / 1.5 / 2 / 2.5 / 3 / 3.5 frequency division
GD32L23xx	48MHz	null

In application, as shown in the ***Table 2-2. USBD system clock***, users usually configure the system clock as an integer multiple of 24MHz, according to frequency division coefficient, 48MHz clock is provided for USBD data transmission. GD32L23xx, GD32F303xx, GD32EPRTxx and GD32E503xx product series support IRC48M clock. Based on external high-accuracy reference signal source, CTC could calibrate the clock frequency of IRC48M. IRC48M clock is provided for USBD data transmission as well.

Table 2-3. Code table RCU configuration

```
void rcu_config(void)
{
    uint32_t system_clock = rcu_clock_freq_get(CK_SYS);

    /* enable USB pull-up pin clock */
    rcu_periph_clock_enable(RCC_AHBPeriph_GPIO_PULLUP);

    if (48000000U == system_clock) {
        rcu_usb_clock_config(RCU_CKUSB_CKPLL_DIV1);
    } else if (72000000U == system_clock) {
        rcu_usb_clock_config(RCU_CKUSB_CKPLL_DIV1_5);
    } else if (96000000U == system_clock) {
        rcu_usb_clock_config(RCU_CKUSB_CKPLL_DIV2);
    } else if (120000000U == system_clock) {
        rcu_usb_clock_config(RCU_CKUSB_CKPLL_DIV2_5);
    } else {
        /* reserved */
    }

    /* enable USB APB1 clock */
    rcu_periph_clock_enable(RCU_USBD);
}
```

As shown in the ***Table 2-3. Code table RCU configuration***, the 48M clock is derived from the system clock, which is divided by the corresponding frequency division coefficient, for most product series. In the following example, the system clock is configured as 120MHz, and it is divided by 2.5 frequency division coefficient, so as to support the 48MHz clock, which is required by the USBD clock domain.

2.1.2. GPIO configuration

Table 2-4. Code table GPIO configuration

```
void gpio_config(void)
{
    /* configure usb pull-up pin */
    gpio_init(USB_PULLUP,           GPIO_MODE_OUT_PP,           GPIO_OSPEED_50MHZ,
USB_PULLUP_PIN);

    /* USB wakeup EXTI line configuration */
    exti_interrupt_flag_clear(EXTI_18);
    exti_init(EXTI_18, EXTI_INTERRUPT, EXTI_TRIG_RISING);
}
```

As shown in the ***Table 2-4. Code table GPIO configuration***, the DP cable is connected to the pull-up resistor, which needs to be controlled by the USB_PULLUP_PIN, so as to ensure that MCU is recognized as a USB full speed device, according to the USB protocol. For L23xx series, DP pull up operation is controlled by configuring USBD_DPC regiseter. In addition, if the user needs to implement the USBD wake-up function, EXTI_18 should be configured as the trigger source.

2.1.3. NVIC configuration

Table 2-5. Code table NVIC configuration

```
void nvic_config(void)
{
    /* 2 bits for preemption priority, 2 bits for subpriority */
    nvic_priority_group_set(NVIC_PRIGROUP_PRE1_SUB3);

    /* enable the USB low priority interrupt */
    nvic_irq_enable((uint8_t)USBD_LP_CAN0_RX0_IRQn, 1U, 0U);

    /* enable the USB Wake-up interrupt */
    nvic_irq_enable((uint8_t)USBD_WKUP_IRQn, 0U, 0U);
}
```

As shown in the ***Table 2-5. Code table NVIC configuration***, considering that USBD interrupt is frequently called, it is necessary to ensure that USBD interrupt are not blocked for a long time in USBD related application. Otherwise, USBD data transmission exceptions possibly occur. Therefore, the interrupt priority of USBD needs to be as high as possible to ensure that the bus is not preempted by other interrupts for a long time.

2.1.4. Application configuration

Table 2-6. Code table application configuration

```
static void hid_key_data_send(usb_dev *udev)
{
    standard_hid_handler *hid =
(standard_hid_handler*)udev->class_data[USBD_HID_INTERFACE];
    if (hid->prev_transfer_complete) {
        switch (key_state()) {
        case CHAR_A:
            hid->data[2] = 0x04U;
            break;
        case CHAR_B:
            hid->data[2] = 0x05U;
            break;
        case CHAR_C:
            hid->data[2] = 0x06U;
            break;
        default:
            break;
        }
        if (0U != hid->data[2]) {
            hid_report_send(udev, hid->data, HID_IN_PACKET);
        }
    }
}
```

In the USBD application example, as shown in the ***Table 2-6. Code table application configuration***, after the USBD device enumeration completed, other modules of the MCU need to be called to update the application data, and then input or output data through the USBD endpoint. As shown in the code list above, press the key, hid->data[2] is assigned to corresponding value. Application call the usbd_ep_send function, load the preparing sent data into USBD buffer RAM and configure endpoint state as active, after USBD device have received IN token, and then sent the data to host. After completed data sent, MCU enter the interrupt handler USBD_LP_CAN0_TX_IRQHandler, then MCU execute IN transcation handler branch udev->ep_transc[ep_num][TRANSC_IN](udev, ep_num), that is data in transcation handler function, illustrate the IN transcation is completed. in this data in transcation handler function, user could add corresponding handler, for example, configure transmit completed flag.

In other application, there is possible OUT data transcation, call the usbd_ep_recev function, MCU enter the interrupt handler USBD_LP_CAN0_RX0_IRQHandler, then MCU execute OUT transcation handler branch, udev->ep_transc[ep_num][TRANSC_OUT](udev, ep_num), that is data out transcation handler function. in this data out transcation handler function, user could add corresponding handler, for example, configure receive completed flag.

2.2. usbd_driver bottom layer file and library function description

The usbd_driver device driver layer contains two folders, Include and Source. Include folder is the underlying header file, and Source folder is the underlying source file. The device driver layer file is illustrated as shown in ***Table 2-7. Device driver layer file description list***.

Table 2-7. Device driver layer file description list

File name	Description
usbd_core.h/.c	USBD device driver core interface layer driver
usbd_enum.h/c	USB enumeration function driver
usbd_pwr.h/.c	USBD device power management driver
usbd_transc.h/c	USBD transaction function driver
usb_ch9_std.h/	USBD 2.0 protocol chapter 9

The library function in the usbd_core.h/c file is illustrated as shown in ***Table 2-8. usbd_core.h/c library function description list***.

Table 2-8. usbd_core.h/c library function description list

Function name	Description
usbd_init	configure USB device initialization
usbd_ep_send	endpoints prepare to send data
usbd_ep_recev	endpoints prepare to receive data
usbd_connect	device connect
usbd_disconnect	device disconnect
usbd_core_deinit	deinitialize usbd core
usbd_ep_init	initialize endpoint
usbd_ep_deinit	deinitialize endpoint
usbd_ep_stall	set the endpoint to STALL
usbd_ep_clear_stall	clear the endpoint STALL state
usbd_ep_status_get	get the endpoint state

The library function in the usbd_transc.h/.c file is illustrated as shown in ***Table 2-9. usbd_transc.h/c library function description list***.

Table 2-9. usbd_transc.h/c library function description list

Function name	Description
_usb_setup_transc	USB setup stage processing
_usb_out0_transc	data out stage processing
_usb_in0_transc	data in stage processing
usb_stall_transc	USB stalled transaction
usb_ctl_status_in	USB control transaction status in stage
usb_ctl_data_in	USB control transaction data in & status out stage
usb_ctl_out	USB control transaction data out & status out stage
usb_0len_packet_send	USB send 0 length data packet

The library function in the usbd_pwr.h/.c file is illustrated as shown in ***Table 2-10. usbd_pwr.h/c library function description list***.

Table 2-10. usbd_pwr.h/c library function description list

Function name	Description
resume_mcu	MCU wake-up function
usbd_remote_wakeup_active	start remote wake-up
usbd_suspend	set up the USB device to the suspend mode

The library function in the usbd_enum.h/.c file is illustrated as shown in ***Table 2-11. usbd_enum.h/c Library Function Description List***.

Table 2-11. usbd_enum.h/c Library Function Description List

Function name	Description
usbd_standard_request	handle USB standard device request
usbd_class_request	handle USB device class request
usbd_vendor_request	handle USB vendor request
_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_str_desc_get	get string descriptor
_usb_bos_desc_get	get the BOS 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
int_to_unicode	convert hex 32bits value into unicode char
serial_string_get	get serial string

3. Description of application protocol and routines

3.1. HID

3.1.1. Protocol Overview

HID(Human Interface Device) is a common USB device class, including a lot of devices, such as USB mouse, USB keyboard, USB game joystick and so on. Except the control transfer used in the HID device enumeration phase, the interrupt transfer is used in the application data transfer phase, and the interrupt interval is configured by the bInterval field of the endpoint descriptor.

Except the standard descriptors, HID device descriptors also support three HID device class-specific descriptors: HID descriptor, report descriptor, and entity descriptor. The first two descriptors are described below. The HID descriptor is associated with the interface descriptor, including the version number of the HID specification, the length of the report descriptor, and so on. The report descriptor is complex, length is not fixed, and defines device input and output data formats. Entity descriptors are optional and used to describe the behavioral characteristics of the device.

3.1.2. Descriptor Analysis

This chapter shown the configuration descriptor, interface descriptor, HID descriptor, endpoint descriptor, and report descriptor of the HID keyboard.

Figure 3-1. HID congifuration descriptor

Configuration descriptor

Name	Value	Dec	Hex	Bin
bLength	Valid	9	0x09	00001001
bDescriptorType	CONFIGURATION	2	0x02	00000010
wTotalLength	34 bytes	34	0x0022	00000000 00100010
bNumInterface	1	1	0x01	00000001
bConfigurationValue	1	1	0x01	00000001
iConfiguration	0	0	0x00	00000000
bmAttributes. Reserved	Zero	0	0x00	00000
bmAttributes. RemoteWakeup	Supported	1	0x1	1
bmAttributes. SelfPowered	No, Bus Powered	0	0x0	0
bmAttributes. Reserved7	One	1	0x1	1
bMaxPower	100 mA	50	0x32	00110010

The configuration descriptor defines the length of the configuration descriptor set, the number of interfaces, and power supply characteristics.

Figure 3-2. HID interface descriptor

Name	Value	Dec	Hex	Bin
bLength	Valid	9	0x09	00001001
bDescriptorType	INTERFACE	4	0x04	00000100
bInterfaceNumber	0	0	0x00	00000000
bAlternateSetting	0	0	0x00	00000000
bNumEndpoints	1	1	0x01	00000001
bInterfaceClass	Human Interface Device (Find out more online)	3	0x03	00000011
bInterfaceSubClass	Boot Interface	1	0x01	00000001
bInterfaceProtocol	Keyboard	1	0x01	00000001
iInterface	0	0	0x00	00000000

The interface descriptor defines the interface class, interface protocol, and so on. As shown in the ***Figure 3-2. HID interface descriptor***, bInterfaceClass is defined as 0x03, that is HID device, and bInterfaceProtocal is defined as 0x01, that is keyboard device.

Figure 3-3. HID desctiptor

Name	Value	Dec	Hex	Bin
bLength	Valid	9	0x09	00001001
bDescriptorType	HID	33	0x21	00100001
bcdHID	1.1.1	273	0x0111	00000001 00010001
bCountryCode	Not Supported	0	0x00	00000000
bNumDescriptors	1	1	0x01	00000001
bDescriptorType[0]	REPORT	34	0x22	00100010
wDescriptorLength[0]	46 bytes	46	0x002E	00000000 00101110

The HID descriptor defines the version number of the HID specification and the length of the report descriptor. As shown in the ***Figure 3-3. HID desctiptor***, HID descriptor define the wDescriptorLength to 0x2E, which is shown that the length of the report descriptor is 46 bytes.

Figure 3-4. HID endpoint descriptor

Name	Value	Dec	Hex	Bin
bLength	Valid	7	0x07	00000111
bDescriptorType	ENDPOINT	5	0x05	00000101
bEndpointAddress	1 IN	129	0x81	10000001
bmAttributes. TransferType	Interrupt	3	0x3	11
bmAttributes. Reserved	Zero	0	0x00	000000
wMaxPacketSize	8 bytes	8	0x0008	00000000 00001000
bInterval	64 frames (64 ms)	64	0x40	01000000

The endpoint descriptor defines the endpoint transfer type, time interval, etc. As shown in the ***Figure 3-4. HID endpoint descriptor***, define bmAttributes.TransferType as 03, and define bInternal as 0x40, which is shown that expressed as the interrupt time interval is 64 milliseconds.

Figure 3-5. Report descriptor

HID Report Descriptor	
Item	**Data**
Usage Page *(Generic Desktop)*	05 01
Usage *(Keyboard)*	09 06
Collection *(Application)*	A1 01
Usage Page *(Keyboard)*	05 07
Usage Minimum *(Keyboard Left Control)*	19 E0
Usage Maximum *(Keyboard Right GUI)*	29 E7
Logical minimum *(0)*	15 00
Logical maximum *(1)*	25 01
Report Count *(8)*	95 08
Report Size *(1)*	75 01
Input *(Data,Value,Absolute,Bit Field)*	81 02
Report Count *(1)*	95 01
Report Size *(8)*	75 08
Input *(Constant,Value,Absolute,Bit Field)*	81 03
Report Count *(6)*	95 06
Report Size *(8)*	75 08
Logical minimum *(0)*	15 00
Logical maximum *(255)*	26 FF 00
Usage Page *(Keyboard)*	05 07
Usage Minimum *(No event indicated)*	19 00
Usage Maximum *(Keyboard Application)*	29 65
Input *(Data,Array,Absolute,Bit Field)*	81 00
End Collection	C0

The report descriptor defines the format of input and output data. In the data transmission stage of the application, the data, which is sent to or received from the host, must conform to the report descriptor.

3.1.3. Application Class Request Brief Introduction

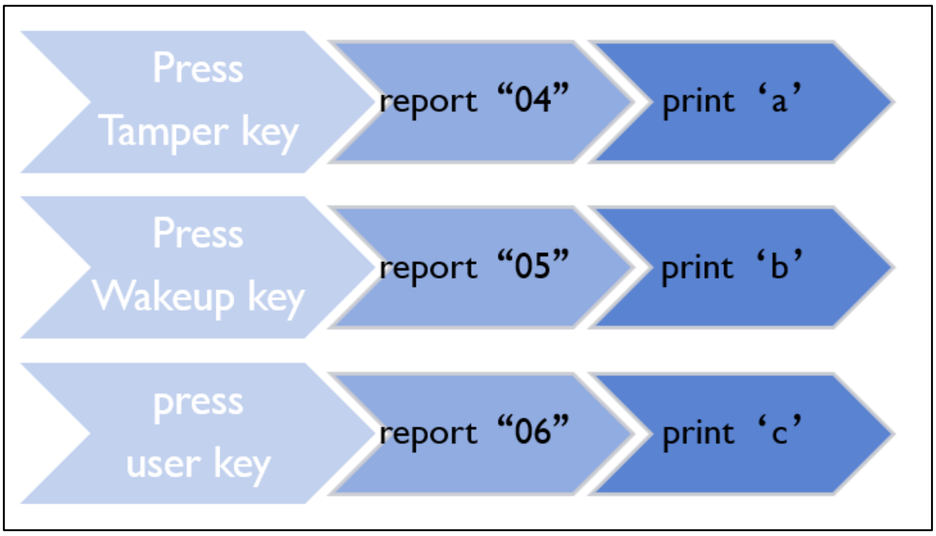
Table 3-1. HID partial class resquests introduction

Class request	Description
USB_GET_DESCRIPTOR	get report descriptor
GET_REPORT	get report control information
SET_REPORT	set report control information
GET_IDLE	get idle state
SET_IDLE	set idle state

Considering that there is lots of application class request, only several requests are listed in the article, in the course of application development, user should handle the class request according to the requirement of project. The class request of CDC class and UAC class are in a similar way

3.1.4. Data Transmission

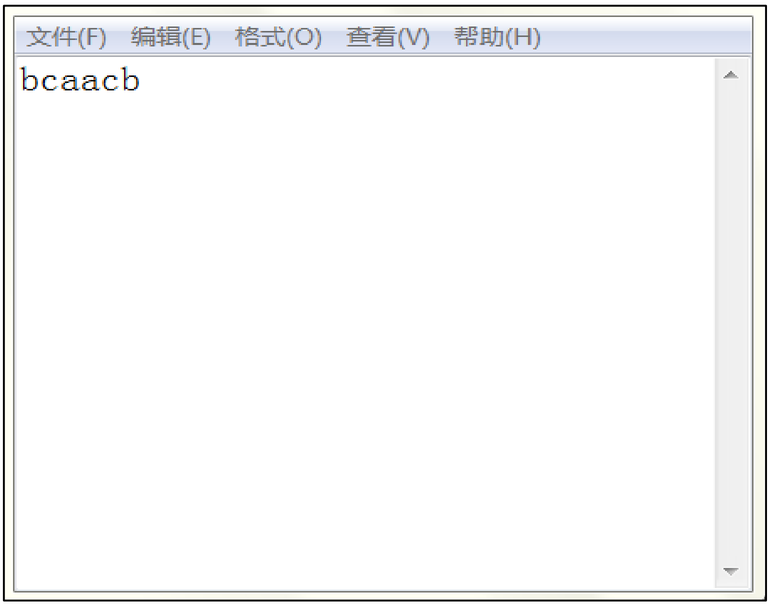
Figure 3-6. HID keyboard applicaton example



After the enumeration of HID keyboard devices completed, a keyboard device is added to the device manager of the host. As shown in the ***Figure 3-6. HID keyboard applicaton example***, Press the Tamper key of the development board, set the value of hid->data [2] to "04", send the message to the host through the device IN endpoint, and the host displays "a"; press the Wakeup key, and the host displays "b"; press the User key, and the host displays "c".

3.1.5. DEMO Result

Figure 3-7. HID keyboard output



As shown in the ***Figure 3-7. HID keyboard output***, after open the text editor, press the corresponding key on the development board, the text editor will print the corresponding character.

3.2. CDC

3.2.1. Protocol Overview

CDC (Communication Device Class) is USB subclass which is defined for various communication device by USB organization, in this article, the virtual serial port is the abstract control model of the telephone service model.

In CDC device descriptor, bDeviceClass field is 2, that is CDC class, define two interface, CDC functions are defined in subclass in its interface descriptor. When bInterfaceClass is set to 02, that is communication control class, then bInterfaceSubClass is set to 02, that is Abstract line control model. While bInterfaceClass is set to 10, that is communication data class.

For operating system before Win10, CDC device needs to install the corresponding device driver, gigadevice has developed CDC device driver adapted to GD32 MCU (USB Virtual COM Port Driver). The download address is ***http://www.gd32mcu.com/cn/download/7***. The computer software can use serial debugging assistant.

3.2.2. Descriptor Analysis

This chapter shows the configuration descriptor, interface descriptor and endpoint descriptor of the CDC device.The configuration descriptor defines the length of the configuration descriptor set, the number of interfaces, and power supply characteristics. Interface descriptors define interface classes, interface subclasses, and so on.

Figure 3-8. CDC configuration descriptor

Configuration descriptor				
Name	**Value**	**Dec**	**Hex**	**Bin**
bLength	Valid	9	0x09	00001001
bDescriptorType	CONFIGURATION	2	0x02	00000010
wTotalLength	67 bytes	67	0x0043	00000000 01000011
bNumInterface	2	2	0x02	00000010
bConfigurationValue	1	1	0x01	00000001
iConfiguration	0	0	0x00	00000000
bmAttributes. Reserved	Zero	0	0x00	00000
bmAttributes. RemoteWakeup	Not supported	0	0x0	0
bmAttributes. SelfPowered	No, Bus Powered	0	0x0	0
bmAttributes. Reserved7	One	1	0x1	1
bMaxPower	100 mA	50	0x32	00110010

As shown in ***Figure 3-8. CDC configuration descriptor***, the value of bNumInterface field is 2, there is two interface descriptor for CDC device. One is CDC control interface descriptor, the other one is CDC data interface descriptor.

Figure 3-9. CDC control interface descriptor

Interface descriptor				
Name	**Value**	**Dec**	**Hex**	**Bin**
bInterfaceNumber	0	0	0x00	00000000
bAlternateSetting	0	0	0x00	00000000
bNumEndpoints	1	1	0x01	00000001
bInterfaceClass	Communication Control (Find out more online)	2	0x02	00000010
bInterfaceSubClass	Abstract Line Control Model	2	0x02	00000010
bInterfaceProtocol	Common AT commands	1	0x01	00000001

As shown in ***Figure 3-9. CDC control interface descriptor***, the value of bInterfaceNumber field is 0, that is interface index is 0, the value of bInterfaceClass field is 2, that is communication control interface.

Figure 3-10. CDC header descriptor

Functional Descriptor				
Name	**Value**	**Dec**	**Hex**	**Bin**
bFunctionLength	Valid	5	0x05	00000101
bDescriptorType	CS_INTERFACE	36	0x24	00100100
bDescriptorSubtype	Header	0	0x00	00000000
bcdCDC	1.1	272	0x0110	00000001 00010000

Figure 3-11. CDC call management descriptor

Functional Descriptor				
Name	**Value**	**Dec**	**Hex**	**Bin**
bFunctionLength	Valid	5	0x05	00000101
bDescriptorType	CS_INTERFACE	36	0x24	00100100
bDescriptorSubtype	Call Management	1	0x01	00000001
bmCapabilities. HandleManagement	No	0	0x0	0
bmCapabilities. DataClass	No	0	0x0	0
bmCapabilities. Reserved	0x00	0	0x00	000000
bDataInterface	1	1	0x01	00000001

As shown in ***Figure 3-11. CDC call management descriptor***, the value of bDataInterface field is 1, that is corresponding data interface index is 1, in multiple COM device, the bDataInterface field of CDC call management is equal to the index of corresponding interface descriptor.

Figure 3-12. CDC abstract control management descriptor

Functional Descriptor				
Name	**Value**	**Dec**	**Hex**	**Bin**
bFunctionLength	Valid	4	0x04	00000100
bDescriptorType	CS_INTERFACE	36	0x24	00100100
bDescriptorSubtype	Abstract Control Management	2	0x02	00000010
bmCapabilities. CommFeature	No	0	0x0	0
bmCapabilities. LineCoding	Yes	1	0x1	1
bmCapabilities. SendBreak	No	0	0x0	0
bmCapabilities. NetworkConnection	No	0	0x0	0
bmCapabilities. Reserved	0x0	0	0x0	0000

Figure 3-13. CDC union functional descriptor

Functional Descriptor				
Name	**Value**	**Dec**	**Hex**	**Bin**
bFunctionLength	Valid	5	0x05	00000101
bDescriptorType	CS_INTERFACE	36	0x24	00100100
bDescriptorSubtype	Union Functional descriptor	6	0x06	00000110
bMasterInterface	0	0	0x00	00000000
bSlaveInterface0	1	1	0x01	00000001

Figure 3-14. CDC command endpoint descriptor

Endpoint descriptor				
Name	**Value**	**Dec**	**Hex**	**Bin**
bLength	Valid	7	0x07	00000111
bDescriptorType	ENDPOINT	5	0x05	00000101
bEndpointAddress	2 IN	130	0x82	10000010
bmAttributes. TransferType	Interrupt	3	0x3	11
bmAttributes. Reserved	Zero	0	0x00	000000
wMaxPacketSize	8 bytes	8	0x0008	00000000 00001000
bInterval	10 frames (10 ms)	10	0x0A	00001010

As shown in *Figure 3-14. CDC command endpoint descriptor*, the value of bmAttributes.TransferType field is 3, that is interrupt transfer endpoint, even though there is no data input from command endpoint in CDC device, otherwise, it is undeletable, deleting could lead exception.

Figure 3-15. CDC data interface descriptor

Interface descriptor				
Name	**Value**	**Dec**	**Hex**	**Bin**
bLength	Valid	9	0x09	00001001
bDescriptorType	INTERFACE	4	0x04	00000100
bInterfaceNumber	1	1	0x01	00000001
bAlternateSetting	0	0	0x00	00000000
bNumEndpoints	2	2	0x02	00000010
bInterfaceClass	Communication Data (Find out more online)	10	0x0A	00001010
bInterfaceSubClass	Unknown (0x00)	0	0x00	00000000
bInterfaceProtocol	None	0	0x00	00000000
iInterface	0	0	0x00	00000000

As shown in ***Figure 3-15. CDC data interface descriptor***, the value of bInterfaceNumber field is 1, that is interface index is 1, the value of bInterfaceClass field is 10, that is communication data interface descriptor, the value of bNumEndpoints field is 2, which is shown that there is two endpoint for the data interface.

Figure 3-16. CDC OUT descriptor

Endpoint descriptor				
Name	**Value**	**Dec**	**Hex**	**Bin**
bLength	Valid	7	0x07	00000111
bDescriptorType	ENDPOINT	5	0x05	00000101
bEndpointAddress	3 OUT	3	0x03	00000011
bmAttributes. TransferType	Bulk	2	0x2	10
bmAttributes. Reserved	Zero	0	0x00	000000
wMaxPacketSize	64 bytes	64	0x0040	00000000 01000000
bInterval	Ignored for full speed Bulk endpoints	0	0x00	00000000

As shown in ***Figure 3-16. CDC OUT descriptor***, the value of bmAttributes.TransferType field is 2, that is bulk transfer endpoint, OUT endpoint is used to receive data for CDC device.

Figure 3-17. CDC IN descriptor

Endpoint descriptor				
Name	**Value**	**Dec**	**Hex**	**Bin**
bLength	Valid	7	0x07	00000111
bDescriptorType	ENDPOINT	5	0x05	00000101
bEndpointAddress	1 IN	129	0x81	10000001
bmAttributes. TransferType	Bulk	2	0x2	10
bmAttributes. Reserved	Zero	0	0x00	000000
wMaxPacketSize	64 bytes	64	0x0040	00000000 01000000
bInterval	Ignored for full speed Bulk endpoints	0	0x00	00000000

As shown in ***Figure 3-17. CDC IN descriptor***, the value of bmAttributes.TransferType field is 2, that is bulk transfer endpoint, IN endpoint is used to send data for CDC device.

3.2.3. Application Class Request Brief Introduction

Table 3-2. CDC partial class resquests introduction

Class request	Description
SET_LINE_CODING	set serial port attributes, such as baud rate
GET_LINE_CODING	get serial port attributes, such as baud rate
SET_CONTROL_LINE_STATE	configure serial port state, such as open or close

3.2.4. Data Transmission

Before connecting the device to the host, firstly install the CDC device driver. As shown in ***Figure 3-18. CDC device enumeration***, after the enumeration of CDC devices completed, “GD32 Virtual Com Port (COMx)” is displayed in “Universal Serial Bus Controller of the Device Manager”. The number of COM is depended on local serial port installation.

Figure 3-18. CDC device enumeration

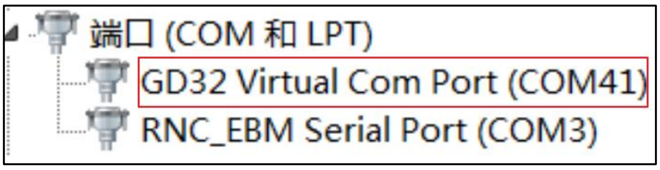
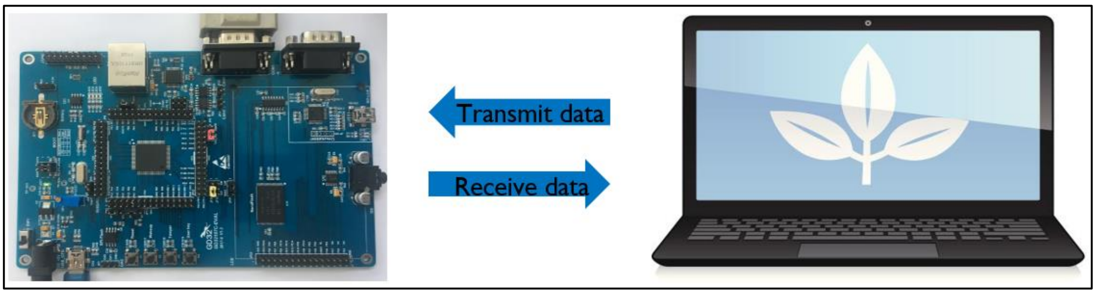


Figure 3-19. Virtual serial port data interaction



As shown in ***Figure 3-19. Virtual serial port data interaction***, CDC device implemented data callback function. When host sent the data to device, computer software transmit data to OUT endpoint of CDC device through USB bus, OUT endpoint load the received data into application buffer. When device sent the data to host, the data in application buffer is loaded into Tx FIFO, CDC device send the received data to host through IN endpoint, the received data is displayed in HyperTerminal.

3.2.5. DEMO Result

Download the program to the board and run. When you input message through computer keyboard, the HyperTerminal will receive and shown the message. As is shown in ***Figure***

3-20. Virtual serial port print, for example, when you input “GigaDevice MCU”, the HyperTerminal will get and show it as below.

Figure 3-20. Virtual serial port print

3.3. DFU

3.3.1. Protocol Overview

DFU(Device Firmware Upgrade) is mainly used to upload and download firmware through USB ports. DFU device could be regraded as a data channel between MCU and programming tool (host computer). Considering that DFU device needs to install the corresponding device driver and arrange upper computer to ensure execute its normal function, gigadevice has developed a multi-interface programming environment (GD32 All-In-One Programmer) and has made DFU device driver adapted to GD32 MCU (GD32 DFU Drivers). The download address is ***http://www.gd32mcu.com/cn/download/7***.

3.3.2. Descriptor Analysis

This chapter shows the configuration descriptor, interface descriptor and function descriptor of the DFU device .The configuration descriptor defines the length of the configuration descriptor set, the number of interfaces, and power supply characteristics. Interface descriptor define interface classes, interface subclasses, and so on.

Figure 3-21. DFU configuration descriptor

Configuration descriptor

Name	Value	Dec	Hex	Bin
bLength	Valid	9	0x09	00001001
bDescriptorType	CONFIGURATION	2	0x02	00000010
wTotalLength	27 bytes	27	0x001B	00000000 00011011
bNumInterface	1	1	0x01	00000001
bConfigurationValue	1	1	0x01	00000001
iConfiguration	0	0	0x00	00000000
bmAttributes. Reserved	Zero	0	0x00	00000
bmAttributes. RemoteWakeup	Not supported	0	0x0	0
bmAttributes. SelfPowered	No, Bus Powered	0	0x0	0
bmAttributes. Reserved7	One	1	0x1	1
bMaxPower	100 mA	50	0x32	00110010

Figure 3-22. DFU interface descriptor

Interface descriptor				
Name	**Value**	**Dec**	**Hex**	**Bin**
bLength	Valid	9	0x09	00001001
bDescriptorType	INTERFACE	4	0x04	00000100
bInterfaceNumber	0	0	0x00	00000000
bAlternateSetting	0	0	0x00	00000000
bNumEndpoints	0	0	0x00	00000000
bInterfaceClass	Application-specific (Find out more online)	254	0xFE	11111110
bInterfaceSubClass	Device Firmware Upgrade	1	0x01	00000001
bInterfaceProtocol	DFU Mode v1.1	2	0x02	00000010
iInterface	0	0	0x00	00000000

As shown in ***Figure 3-22. DFU interface descriptor***, bInterfaceClass is defined as 0xFE, which is shown as specific application class device, and bInterfaceSubClass is defined as 0x01, which is shown as DFU device.

Figure 3-23. DFU function descriptor

DFU functional descriptor				
Name	**Value**	**Dec**	**Hex**	**Bin**
bLength	Valid	9	0x09	00001001
bDescriptorType	DFU_FUNCTIONAL	33	0x21	00100001
bmAttributes.bitCanDnload	Yes	1	0x1	1
bmAttributes.bitCanUpload	Yes	1	0x1	1
bmAttributes. bitManifestationTolerant	No, must see bus reset	0	0x0	0
bmAttributes.bitWillDetach	Yes	1	0x1	1
bmAttributes.reserved	Zero	0	0x0	0000
wDetachTimeOut	255 ms	255	0x00FF	00000000 11111111
wTransferSize	2,048 bytes	2,048	0x0800	00001000 00000000
bcdDfuVersion	1.1	272	0x0110	00000001 00010000

3.3.3. Application Class Request Brief Introduction

Table 3-3. DFU class resquests introduction

request code	class resquests	description
0	DFU_DETACH	Request the device to leave DFU mode and enter the application.
1	DFU_DNLOAD	Data from the host is sent to the device, and the data is loaded to the device storage media, including erasing

request code	class resquests	description
		operations.
2	DFU_UPLOAD	Transfer data from the device to the host, and load data from the storage media to the target file on the host.
3	DFU_GETSTATUS	Request the device to send a status report to the host.
4	DFU_CLRSTATUS	Ask the device to clear the error status and move to the next step.
5	DFU_GETSTATE	The requesting device only sends the state it is currently entering.
6	DFU_ABORT	Request the device to leave the current state/operation and enter the idle state.

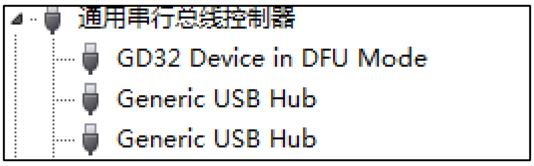
Table 3-4. Summary of parameters for DFU specific class requests

bmRequest	bRequest	wValue	wIndex	wLength	Data
00100001b	DETACH	wTimeout	Interface	Zero	None
00100001b	DNLOAD	wBloackNum	Interface	length	firmware
10100001b	UPLOAD	Zero	Interface	length	firmware
00100001b	GETSTATUS	Zero	Interface	6	Status
00100001b	CLRSTATUS	Zero	Interface	Zero	None
00100001b	GETSTATE	Zero	Interface	1	State
00100001b	ABORT	Zero	Interface	Zero	None

3.3.4. Data Transmisson

Before connecting the device to the host, firstly install the DFU device driver. As shown in ***Figure 3-24. DFU device enumeration***, after the enumeration of DFU devices completed, "GD32 Device in DFU Mode" is displayed in "Universal Serial Bus Controller of the Device Manager".

Figure 3-24. DFU device enumeration

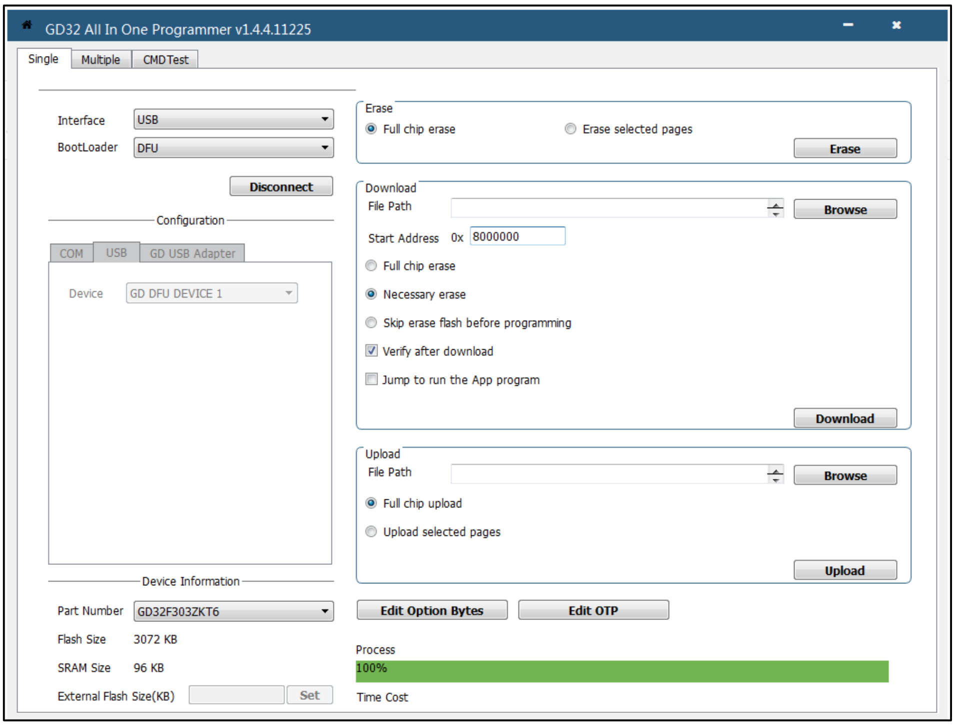


During the period of excuting download function, the host sends data to the DFU device through the USB bus, and then loads the data to the storage media. During the period of excuting upload function, the host receives the data from the DFU device through the USB bus and generates the bin file.

3.3.5. DEMO Result

As shown in ***Figure 3-25. DFU host computer***, Open the "GD32 All In One Programmer" host computer, select the interface as USB from ComboBox, user can see "GD DFU DEVICE 1" , and then click "Connect". User can perform various functions of the DFU device, such as full chip erase, page erase, file download, file upload and option byte operations, etc.

Figure 3-25. DFU host computer



3.4. UAC

3.4.1. Protocol Overview

UAC (USB Audio Class) can transmit digital audio data. The USB audio class is defined in the interface layer, and the USB audio class is divided into different subclasses for further detailed enumeration and configuration. All USB audio functions are included in the subclassed of the USB audio class. When bInterfaceSubClass is set to 01, that is AudioControl Interface Subclass; when bInterfaceSubClass is set to 02, that is AudioStreaming Interface Subclass.

In the subclass of the AudioControl interface, the wTerminalType of the Output Terminal descriptor is defined as 0301 (Speaker), which plays the audio source data, which is sent to the device through the OUT endpoint. If wTerminalType is defined as 0101 (Micphone), audio source data is collected through the IN endpoint and sent to the host.

UAC device data transmission adopts isochronous transfer, and the transmission time interval is determined by the bInterval field of the endpoint descriptor below.

3.4.2. Descriptor Analysis

This chapter shows the configuration descriptor, interface descriptor and endpoint descriptor of the UAC device. The configuration descriptor defines the length of the configuration descriptor set, the number of interfaces, and power supply characteristics. Interface descriptors define interface classes, interface subclasses, and so on.

Figure 3-26. UAC configuration descriptor

Configuration descriptor				
Name	Value	Dec	Hex	Bin
bNumInterface	2	2	0x02	00000010
bConfigurationValue	1	1	0x01	00000001
bmAttributes. RemoteWakeup	Not supported	0	0x0	0
bmAttributes. SelfPowered	Yes	1	0x1	1
bMaxPower	100 mA	50	0x32	00110010

Figure 3-27. UAC interface descriptor

Interface descriptor				
Name	Value	Dec	Hex	Bin
bInterfaceNumber	0	0	0x00	00000000
bAlternateSetting	0	0	0x00	00000000
bNumEndpoints	0	0	0x00	00000000
bInterfaceClass	Audio (Find out more online)	1	0x01	00000001
bInterfaceSubClass	Audio Control	1	0x01	00000001

As show in *Figure 3-27. UAC interface descriptor*, Define bInterfaceClass to 0x01, which is the Audio device, and bInterfaceSubClass to 0x01, which is the Audio Control subclass.

Figure 3-28. UAC header descriptor

Audio Descriptor				
Name	**Value**	**Dec**	**Hex**	**Bin**
bDescriptorSubtype	HEADER	1	0x01	00000001
bcdADC	1.0	256	0x0100	00000001 00000000
wTotalLength	39 bytes	39	0x0027	00000000 00100111
baInterfaceNr(1)	1	1	0x01	00000001

As shown in *Figure 3-28. UAC header descriptor*, The length defined by the field wTotalLength of the HEADER descriptor is the length of itself + length of the input Terminal descriptor + length of the Feature unit descriptor + length of the Output Terminal descriptor.

Figure 3-29. UAC input terminal descriptor

Audio Descriptor				
Name	**Value**	**Dec**	**Hex**	**Bin**
bDescriptorSubtype	INPUT_TERMINAL	2	0x02	00000010
bTerminalID	1	1	0x01	00000001
wTerminalType	USB streaming	257	0x0101	00000001 00000001
bNrChannels	1	1	0x01	00000001
wChannelConfig. Left Front (L)	Not present	0	0x0	0
wChannelConfig. Right Front (R)	Not present	0	0x0	0
wChannelConfig. Center Front (C)	Not present	0	0x0	0
wChannelConfig. Low Frequency Enhancement (LFE)	Not present	0	0x0	0
wChannelConfig. Left Surround (LS)	Not present	0	0x0	0
wChannelConfig. Right Surround (RS)	Not present	0	0x0	0
wChannelConfig. Left of Center (LC)	Not present	0	0x0	0
wChannelConfig. Right of Center (RC)	Not present	0	0x0	0
wChannelConfig. Surround (S)	Not present	0	0x0	0
wChannelConfig. Side Left (SL)	Not present	0	0x0	0
wChannelConfig. Side Right (SR)	Not present	0	0x0	0
wChannelConfig. Top (T)	Not present	0	0x0	0

Figure 3-30. UAC feature unit descriptor

Audio Descriptor				
Name	**Value**	**Dec**	**Hex**	**Bin**
bDescriptorSubtype	FEATURE_UNIT	6	0x06	00000110
bUnitID	2	2	0x02	00000010
bSourceID	1	1	0x01	00000001
bmaControls(0). Mute	Supported	1	0x1	1
bmaControls(0). Volume	Not supported	0	0x0	0
bmaControls(0). Bass	Not supported	0	0x0	0
bmaControls(0). Mid	Not supported	0	0x0	0
bmaControls(0). Treble	Not supported	0	0x0	0
bmaControls(0). Graphic Equalizer	Not supported	0	0x0	0
bmaControls(0). Automatic Gain	Not supported	0	0x0	0
bmaControls(0). Delay	Not supported	0	0x0	0
bmaControls(1). Mute	Not supported	0	0x0	0
bmaControls(1). Volume	Not supported	0	0x0	0
bmaControls(1). Bass	Not supported	0	0x0	0
bmaControls(1). Mid	Not supported	0	0x0	0
bmaControls(1). Treble	Not supported	0	0x0	0
bmaControls(1). Graphic Equalizer	Not supported	0	0x0	0
bmaControls(1). Automatic Gain	Not supported	0	0x0	0
bmaControls(1). Delay	Not supported	0	0x0	0

Figure 3-31. UAC output terminal descriptor

Audio Descriptor				
Name	**Value**	**Dec**	**Hex**	**Bin**
bDescriptorSubtype	OUTPUT_TERMINAL	3	0x03	00000011
bTerminalID	3	3	0x03	00000011
wTerminalType	Speaker	769	0x0301	00000011 00000001
bSourceID	2	2	0x02	00000010

Figure 3-32. UAC standard data stream zero bandwidth interface descriptor

Interface descriptor				
Name	**Value**	**Dec**	**Hex**	**Bin**
bInterfaceNumber	1	1	0x01	00000001
bAlternateSetting	0	0	0x00	00000000
bNumEndpoints	0	0	0x00	00000000
bInterfaceClass	Audio (Find out more online)	1	0x01	00000001
bInterfaceSubClass	Audio Streaming	2	0x02	00000010

As shown in ***Figure 3-32. UAC standard data stream zero bandwidth interface descriptor***, bInterfaceClass is defined as 0x01, that is audio class, and bInterfaceSubClass is defined as 0x02, that is audio streaming subclass.

Figure 3-33. UAC standard AC interface descriptor

Interface descriptor				
Name	Value	Dec	Hex	Bin
bInterfaceNumber	1	1	0x01	00000001
bAlternateSetting	1	1	0x01	00000001
bNumEndpoints	1	1	0x01	00000001
bInterfaceClass	Audio (Find out more online)	1	0x01	00000001
bInterfaceSubClass	Audio Streaming	2	0x02	00000010

Figure 3-34. UAC generic data flow descriptor

Audio Descriptor				
Name	Value	Dec	Hex	Bin
bDescriptorSubtype	AS_GENERAL	1	0x01	00000001
bTerminalLink	1	1	0x01	00000001
bDelay	1 frame	1	0x01	00000001
wFormatTag	PCM	1	0x0001	00000000 00000001

Figure 3-35. UAC format type descriptor

Audio Descriptor				
Name	Value	Dec	Hex	Bin
bDescriptorSubtype	FORMAT_TYPE	2	0x02	00000010
bFormatType	FORMAT_TYPE_III	3	0x03	00000011
bNrChannels	2	2	0x02	00000010
bSubframeSize	2 bytes	2	0x02	00000010
bBitResolution	16 bits	16	0x10	00010000
bSamFreqType	1 discrete sampling frequencies	1	0x01	00000001
tSamFreq(1)	16.0 kHz	16,000	0x003E80	00000000 00111110 10000000

As shown in ***Figure 3-35. UAC format type descriptor***, bFormatType defines the audio source format as FORMAT_TYPE_III, bBitResolution is positioned as 0x10, which means that the audio source is played in a 16-bit format, and tSamFreq defines the audio source collection frequency.

Figure 3-36. UAC endpoint descriptor

Endpoint descriptor				
Name	Value	Dec	Hex	Bin
bEndpointAddress	1 OUT	1	0x01	00000001
bmAttributes. TransferType	Isochronous	1	0x1	01
wMaxPacketSize	64 bytes	64	0x0040	00000000 01000000
bInterval	1 frame (1000 us)	1	0x01	00000001
bRefresh	Not used	0	0x00	00000000
bSynchAddress	Not used	0	0x00	00000000

Figure 3-37. UAC endpoint general descriptor

Audio Descriptor				
Name	Value	Dec	Hex	Bin
bDescriptorSubtype	EP_GENERAL	1	0x01	00000001
bmAttributes. Sampling Frequency	Not supported	0	0x0	0
bmAttributes. Pitch	Not supported	0	0x0	0
bmAttributes. MaxPacketsOnly	Not supported	0	0x0	0
bLockDelayUnits	Undefined	0	0x00	00000000
wLockDelay	0	0	0x0000	00000000 00000000

3.4.3. Application Class Request Brief Introduction

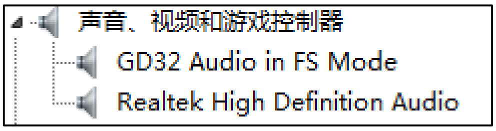
Table 3-5. UAC partial class resquests introduction

Class request	Description
AUDIO_REQ_GET_CUR	get current audio parameter
AUDIO_REQ_SET_CUR	set current audio parameter

3.4.4. Data Transmission

As shown in ***Figure 3-38. UAC device enumeration***, the device "GD32 Audio in FS Mode" will be appeared in the sub-item "Sound, Video and Game Controller" of the Device Manager after the UAC device enumeration completed. Host select an audio file and play the audio file. The audio data is transmitted to the UAC through the USB bus. The UAC transfer the obtained data to the headphone interface through the I2S bus, then audio file is played through headphone.

Figure 3-38. UAC device enumeration



3.4.5. DEMO Result

As shown in ***Figure 3-39. UAC channel configuration***, in the sub-item "Play" of the host sound configuration, select "GD32 Audio in FS Mode" as the default speaker, insert the earphone into the earphone jack. As shown in ***Figure 3-40. Audio source playback***, double-click the audio file and hear what the host play through the earphone jack of the development board.

Figure 3-39. UAC channel configuration

Figure 3-40. Audio source playback

4. Revision history

Table 4-1. Revision history

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

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.