Application Note

Document No.: AN1095

APM32 USB Device Library Application Note

Version: V1.0

1 Introduction

This application note provides how to configure and use the APM32 USB device library, including directory file structure, function introduction, and application methods.

USB is the abbreviation for Universal Serial Bus. It is a serial bus standard and a technical specification for input and output interfaces. It is widely used in devices such as mice, keyboards, printers, scanners, and digital cameras.

Applicable product series
APM32F072xx, APM32L072xx, APM32F10x, APM32E10x, APM32S10x, APM32F4xx

Contents

2 Introduction to USB Device Library

The USB device library is applicable to all APM32 MCU with USB peripherals. The provided APM32 USB SDK includes the following content:

1. Underlying USB driver for each series of chips;
2. USB device standard library driver of each series of chips;
3. Public USB class driver;
4. Common applications for USB device, such as support for USB full-speed, high-speed control, interrupt, batch and synchronous transmission operations;
5. Routines of the following USB class:

   - Communication Device (CDC)

   Virtual COM port, used for USB-to-RS232 bridging.

   - Human Interface Device (HID)

   Key input suitable for development boards to simulate mouse wheel operation.

   - Custom Human Interface Device (CHID)

   Including HID output and input.

   - Mass storage

   MSC devices based on SRAM, SD Card or No Flash can also be used to test MSC transmission speed.

   - Winusb

   Batch transmission based on Microsoft winusb (Microsoft OS 1.0) driver-free mode.

3 USB Device Library Architecture

The architecture of the USB Device Library is mainly divided into five layers, and the application layer is at the top of the architecture. The second layer includes two drivers, i.e. core driver and class driver:

1. Core driver
   - The Core module provides internal device library management state machine and USB interrupt callback.
   - The Translation module provides various USB transmission handler functions.
   - The Request module provides requests related to Chapter 9.
2. Class driver

   The Class driver includes a series of drivers composed of predefined class drivers, which are linked to the USB core by USBD_Init() function.

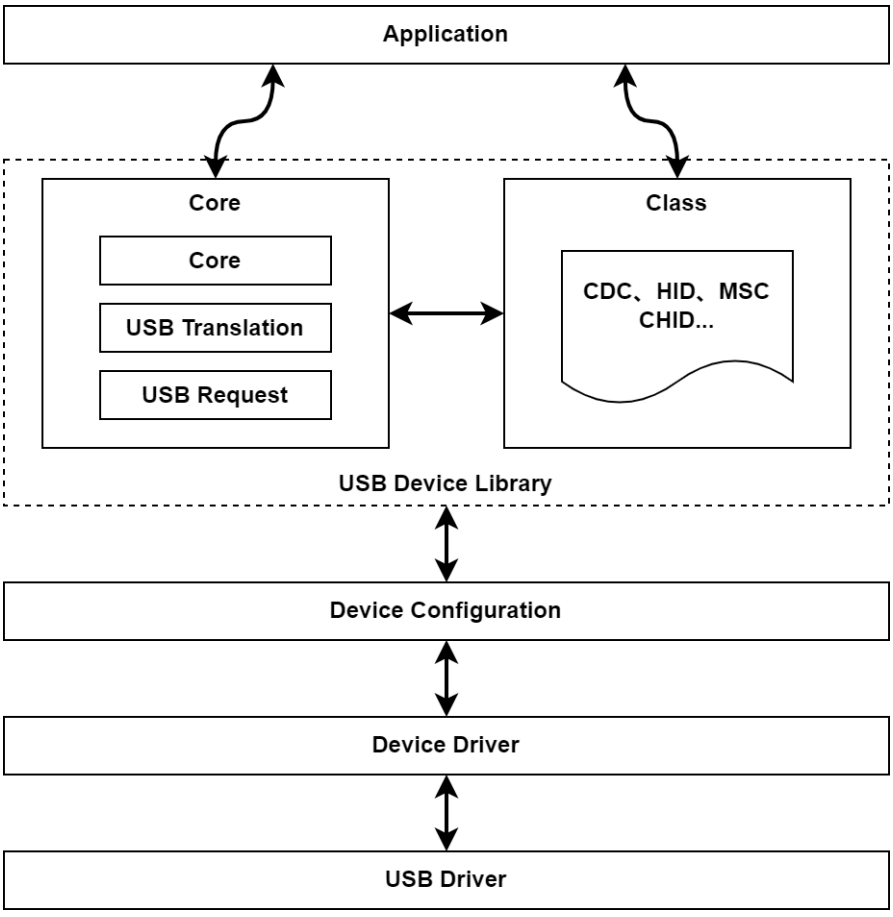

Figure 1 USB Device Library Architecture

4 USB Device Library File Directory Structure

4.1 Directory structure

The USB device library is based on the universal USB protocol stack underlying driver, and is applicable to both full-speed and high-speed modes. The files are included in the Device folder under the APM32_USB_Library directory.

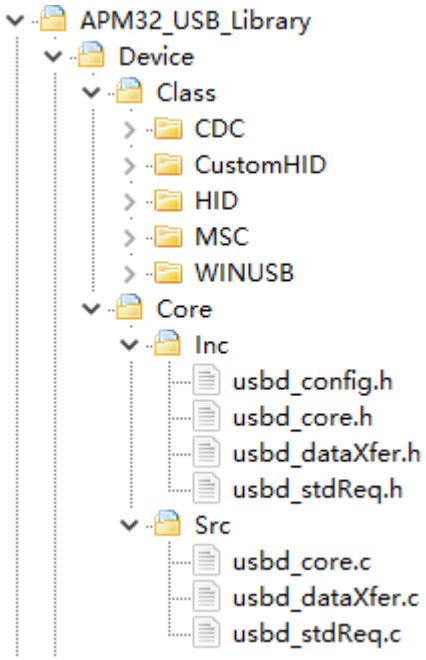

Figure 2 USB Device Library File Directory

4.2 File structure

The files of the USB device library include the underlying shared drivers, time base units, middleware core and class files, and user configuration files.

1. Shared drivers
    - apm32xxx.h
    - apm32xxx_usb.c/h
    - apm32xxx_usb_device.c/h
2. Timebase unit
    - bsp_delay.c/h
3. Middleware
    - Core files
    - Class files
4. User configuration files
    - usbd_board.c/h: USB devide configuration file
    - usbd_device_user.c/h: USB user initialization file
    - usb_descriptor.c/h: Descriptor configuration file
    - usb_interface.c/h: Data interface configuration file

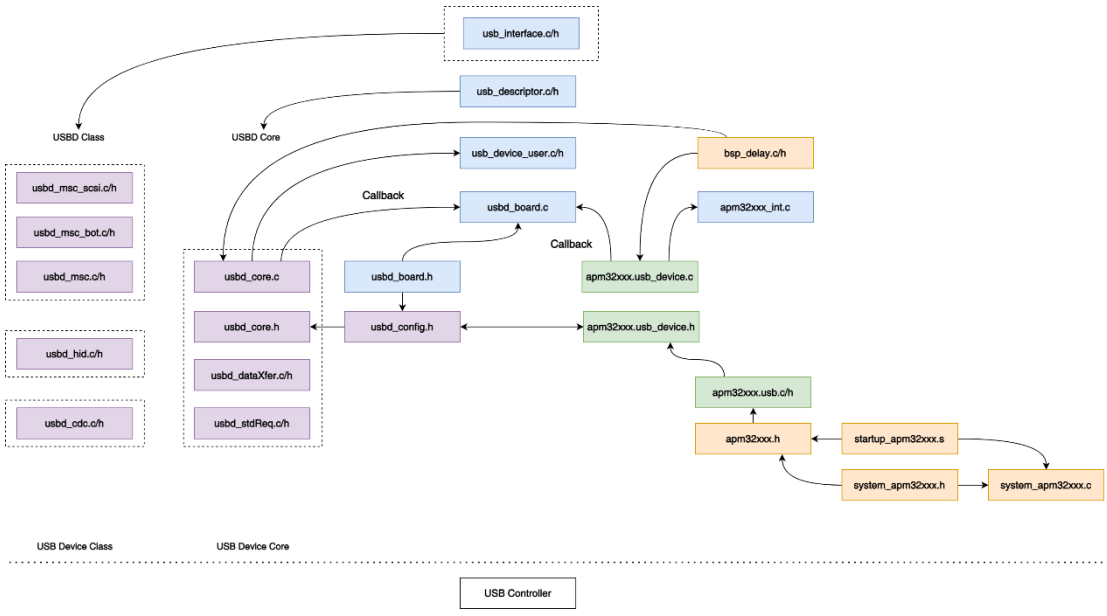

Figure 3 File Structure

4.3 USB Device Configuration

4.3.1 usbd_board(.c/h)

This file contains the configuration and callback functions of the USB device.

4.3.2 usbd_descriptor(.c/h)

This file contains the basic descriptors of the USB device.

4.3.3 usb_device_user(.c/h)

This file is used to initialize USB device.

4.4 USB device core

4.4.1 usbd_core(.c/h)

This file contains the control of the USB core state machine and the handler and callback functions of different events. The following table gives a description of the main functions of this file.

Function	Description
USBD_Init	Initialize the USB device and state machine and link USB device structure to class processing unit
USBD_DeInit	Cancel USB device initialization
USBD_SetupStage	Handle the requests in the setup phase
USBD_DataOutStage	Process the data from Host on appropriate endpoints
USBD_DataInStage	Transmit data to Host on the appropriate end
USBD_SetSpeed	Set USB device speed
USBD_Resume	Restore the device state to the previous state before suspending
USBD_Suspend	Set the device status to suspending
USBD_Reset	Reinitialize the device
USBD_HandleSOF	Handle SOF events
USBD_IsoInInComplete	Handle ISO input incomplete events
USBD_IsoOutInComplete	Handle ISO output incomplete events
USBD_Connect	Handle device connection events
USBD_Disconnect	Handle device disconnection events

Table 1 usbd core function

4.4.2 usbd_dataXfer(.c/h)

This file contains the functions that implement data transmitting and receiving on endpoint 0. The following table gives a description of the main functions of this file.

Function	Description
USBD_CtrlSendData	Transmit data on endpoint 0
USBD_CtrlSendNextData	Transmit remaining data on endpoint 0
USBD_CtrlReceiveData	Receive data on endpoint 0
USBD_CtrlSendStatus	Transmit zero-length data packets on endpoint 0
USBD_CtrlReceiveStatus	Receive zero-length data packets on endpoint 0
USBH_SetupReqParse	Parse setup requests

Table 2 usbd data xfer function

4.4.3 usbd_stdReq(.c/h)

This file contains a standard request callback function array and is used to ensure that USB can complete the standard requests sent from the Host. The following table gives a description of the main functions of this file.

Function	Description
USBD_REQ_GetStatus	Handle "Get Status" requests
USBD_REQ_ClearFeature	Handle "Clear Feature" requests
USBD_REQ_SetFeature	Handle "Set Feature" requests
USBD_REQ_SetAddress	Handle "Set Address" requests
USBD_REQ_GetDesc	Handle "Get Descriptor" requests
USBD_REQ_GetCfg	Handle "Get Configuration" requests
USBD_REQ_SetCfg	Handle "Set Configuration" requests
USBD_REQ_CtrlError	Handle control transmission errors

Table 3 usbd stdReq function

4.4.4 usbd_config(.h)

This file contains definitions of commonly used USB device variables and structures.

4.5 USB Device Class

4.5.1 Human Interface Devices (HID) Class

This class is used for the devices for implementing human-machine interfaces and for interaction between humans and machines (e.g. game controllers, mice, and keyboards). The code for this class is implemented based on *Device Class Definition for Human Interface Devices (HID) Version 1.11*.

4.5.1.1 usbd_hid(.c/h)

This file contains all variables of HID class and specific API, and the default configuration descriptor is mouse. The following table gives a description of the main functions of this file.

Function	Description
USBD_HID_ClassInitHandler	Initialize the structure and endpoint of the HID class
USBD_HID_ClassDeInitHandler	Release the initialization of HID class structure and endpoint
USBD_HID_SetupHandler	Handle setup requests of HID class
USBD_HID_DataInHandler	Process the data sent by the HID class from the IN endpoint to the host
USBD_HID_SOFHandler	Handle SOF events
USBD_HID_ReadInterval	Read the polling interval
USBD_HID_TxReport	Transmit the report descriptor to host

Table 4 usbd hid function

4.5.1.2 usbd_hid_keyboard.c

This file contains all variables of HID class and specific API, and the default HID report descriptor is mouse. The function is consistent with **usbd_hid.c.**

4.5.2 Custom HID Class

The code for this class is implemented according to *Device Class Definition for Human Interface Devices (HID) Version 1.11*.

4.5.2.1 usbd_customhid(.c/h)

This file contains all variables of HID class and specific API, and the HID report descriptor can be customized. The following table gives a description of the main functions of this file.

Function	Description
USBD_CUSTOM_HID_ClassInitHandler	Initialize the structure and endpoint of the CHID class
USBD_CUSTOM_HID_ClassDeInitHandler	Release the initialization of CHID class structure and endpoint
USBD_CUSTOM_HID_SOFHandler	Handle SOF events
USBD_CUSTOM_HID_SetupHandler	Handle setup requests of CHID class
USBD_CUSTOM_HID_RxEP0Handler	Handle receive ready event on endpoint 0
USBD_CUSTOM_HID_DataInHandler	Process the data sent by the CHID class from the IN endpoint to the host
USBD_CUSTOM_HID_DataOutHandler	Process the host data received by the CHID class from the OUT endpoint
USBD_CUSTOM_HID_ReportDescHandler	Transmit the report descriptor to host
USBD_CUSTOM_HID_ReadInterval	Read the polling interval
USBD_CUSTOM_HID_RegisterItf	Register the CHID class interface handlers

Table 5 usbd custom hid function

4.5.3 Mass Storage Class (MSC)

This class is used to implement mass storage devices for data storage and exchange through USB. The code for this class is implemented based on *Universal Serial Bus Mass Storage Class (MSC) Bulk-Only Transport (BOT) Version 1.0*.

4.5.3.1 usbd_msc(.c/h)

This file contains all variables of MSC class and specific API. The following table gives a description of the main functions of this file.

Function	Description
USBD_MSC_ClassInitHandler	Initialize the structure and endpoint of the MSC class
USBD_MSC_ClassDeInitHandler	Release the initialization of MSC class structure and endpoint
USBD_MSC_SOFHandler	Handle SOF events
USBD_MSC_SetupHandler	Handle setup requests of MSC class
USBD_MSC_DataInHandler	Process the data sent by the MSC class from the IN endpoint to the host
USBD_MSC_DataOutHandler	Process the host data received by the MSC class from the OUT endpoint
USBD_MSC_RegisterMemory	Register the MSC class interface handlers

Table 6 usbd msc function

4.5.3.2 usbd_msc_bot(.c/h)

Bulk-Only-Transport (BOT) handler function contained in this file.

4.5.3.3 usbd_msc_scsi(.c/h)

This file contains the necessary SCSI command handler functions and Inquiry and Sense information of MSC.

4.5.4 Communications Devices Class (CDC)

This class is used to implement virtual COM ports and modulator-demodulators. The code for this class is implemented based on *Universal Serial Bus Class Definitions for Communications Devices Revision 1.2*.

4.5.4.1 usbd_cdc(.c/h)

This file contains all variables of CDC class and specific API. The following table gives a description of the main functions of this file.

Function	Description
USBD_CDC_ClassInitHandler	Initialize the structure and endpoint of the CDC class
USBD_CDC_ClassDeInitHandler	Release the initialization of CDC class structure and endpoint
USBD_CDC_SOFHandler	Handle SOF events
USBD_CDC_SetupHandler	Handle setup requests of CDC class
USBD_CDC_RxEP0Handler	Handle receive ready event on endpoint 0
USBD_CDC_DataInHandler	Process the data sent by the CDC class from the IN endpoint to the host
USBD_CDC_DataOutHandler	Process the host data received by the CDC class from the OUT endpoint
USBD_CDC_ReadInterval	Read the polling interval
USBD_CDC_RegisterItf	Register the CDC class interface handlers

Table 7 usbd cdc function

4.5.5 WinUSB

This class is used to implement driver-free devices in the windows environment. The code for this class is implemented according to "WinUSB 1.0".

4.5.5.1 usbd_winusb(.c/h)

This file contains all variables of winusb 1.0 supplier specific class and specific API. The following table gives a description of the main functions of this file.

Function	Description
USBD_WINUSB_ClassInitHandler	Initialize the structure and endpoint of the WINUSB class
USBD_WINUSB_ClassDeInitHandler	Release the initialization of WINUSB class structure and endpoint
USBD_WINUSB_SOFHandler	Handle SOF events
USBD_WINUSB_SetupHandler	Handle setup requests of WINUSB class
USBD_WINUSB_DataInHandler	Process the data sent by the WINUSB class from the IN endpoint to the host
USBD_WINUSB_DataOutHandler	Process the host data received by the WINUSB class from the OUT endpoint
USBD_WINUSB_ReadInterval	Read the polling interval
USBD_WINUSB_RegisterItf	Register the WINUSB class interface handlers

Table 8 usbd winusb function

5 USB Device Library Data Structure

5.1 Setup request processing structure

When a request is received from the host, it will be decoded and placed in the following structure to simplify the processing.

```
typedef struct
{
    union
    {
        uint8_t REQ_DATA[8];

        struct
        {
            USBD_REQ_TYPE_T     bmRequest;
            uint8_t             bRequest;
            uint8_t             wValue[2];
            uint8_t             wIndex[2];
            uint8_t             wLength[2];
        } DATA_FIELD;
    };
} USBD_REQ_SETUP_T;
```

5.2 Descriptor processing structure

This structure contains configuration functions of all common descriptors.

```
typedef struct
{
    const char*             descName;
    USBD_DescCallback_T     deviceDescHandler;
    USBD_DescCallback_T     configDescHandler;
    USBD_DescCallback_T     configStrDescHandler;
    USBD_DescCallback_T     interfaceStrDescHandler;
    USBD_DescCallback_T     langIdStrDescHandler;
    USBD_DescCallback_T     manufacturerStrDescHandler;
    USBD_DescCallback_T     productStrDescHandler;
    USBD_DescCallback_T     serialStrDescHandler;
#if USBD_SUP_LPM
    USBD_DescCallback_T     bosDescHandler;
#endif
    USBD_DescCallback_T     winUsbOsStrDescHandler;
    USBD_DescCallback_T     otherSpeedConfigDescHandler;
    USBD_DescCallback_T     devQualifierDescHandler;
} USBD_DESC_T;
```

5.3 USB device class structure

This structure contains the functions required for common classes.

```
typedef struct
{
    /* Class handler */
    const char*          className;
    void*                classData;
    USBD_STA_T(*ClassInitHandler)(struct _USBD_INFO_T* usbInfo, uint8_t
cfgIndex);
    USBD_STA_T(*ClassDeInitHandler)(struct _USBD_INFO_T* usbInfo, uint8_t
cfgIndex);
    USBD_STA_T(*ClassSofHandler)(struct _USBD_INFO_T* usbInfo);

    /* Control endpoint */
    USBD_STA_T(*ClassSetup)(struct _USBD_INFO_T* usbInfo, USBD_REQ_SETUP_T*
req);
    USBD_STA_T(*ClassTxEP0)(struct _USBD_INFO_T* usbInfo);
    USBD_STA_T(*ClassRxEP0)(struct _USBD_INFO_T* usbInfo);
    /* Specific endpoint */
    USBD_STA_T(*ClassDataIn)(struct _USBD_INFO_T* usbInfo, uint8_t epNum);
    USBD_STA_T(*ClassDataOut)(struct _USBD_INFO_T* usbInfo, uint8_t epNum);
    USBD_STA_T(*ClassIsoOutIncomplete)(struct _USBD_INFO_T* usbInfo,
uint8_t epNum);
    USBD_STA_T(*ClassIsoInIncomplete)(struct _USBD_INFO_T* usbInfo, uint8_t
epNum);
} USBD_CLASS_T;
```

5.4 USB device processing structure

This structure contains USB device data and information.

```
typedef struct _USBD_INFO_T
{
    __IO uint8_t            devState;
    __IO uint8_t            preDevState;
    __IO uint8_t            devEp0State;
    uint32_t                devEp0DataLen;

    uint8_t                 devSpeed;
    uint8_t                 devAddr;

    USBD_DESC_T*            devDesc;
    USBD_CLASS_T*           devClass[USBD_SUP_CLASS_MAX_NUM];

    void*                   devClassUserData[USBD_SUP_CLASS_MAX_NUM];
    uint32_t                classID;
    uint32_t                classNum;

    void*                   cfgDesc;
    USBD_REQ_SETUP_T        reqSetup;

    uint32_t                devCfg;
    uint32_t                devCfgStatus;
    uint32_t                devCfgDefault;
    uint8_t                 devTestModeStatus;
    uint32_t                devRemoteWakeUpStatus;

    USBD_EP_INFO_T          devEpIn[16];
    USBD_EP_INFO_T          devEpOut[16];
    void (*userCallback)(struct _USBD_INFO_T* usbInfo, uint8_t userStatus);
    void*                   dataPoint;
} USBD_INFO_T;
```

6 USB Device Library Application Method

6.1 Configure USB Device Library

6.1.1 Library configuration item

The library is configured in usbd_board.h file. The USB device library supports the following configuration items. Relevant function items need to be configured when related functions are used.

```
#define USBD_SUP_CLASS_MAX_NUM              1      //Maximum number supported
by the class
#define USBD_SUP_INTERFACE_MAX_NUM          1      //Maximum number supported
by the interface
#define USBD_SUP_CONFIGURATION_MAX_NUM      1       //Maximum number supported
by configuration
#define USBD_SUP_STR_DESC_MAX_NUM           512     //Maximum character length
of character string descriptor
#define USBD_SUP_LPM                        0       //LPM switch macro
#define USBD_SUP_SELF_PWR                   1       //Self-powered switch
macro
```

6.1.2 Class configuration items

In addition to the configuration items in the class header file, some classes also need to configure relevant descriptors in the source file. For example, the configuration items of HID class are as follows:

```
#define USBD_HID_MOUSE_REPORT_DESC_SIZE         74      //Mouse HID report
descriptor length
#define USBD_HID_KEYBOARD_REPORT_DESC_SIZE      63      //Keyboard HID report
descriptor length
#define USBD_HID_DESC_SIZE                       9       //HID descriptor
length
#define USBD_HID_FS_INTERVAL                     10      //HID FS polling
interval
#define USBD_HID_HS_INTERVAL                     7       //HID HS  polling
interval
#define USBD_HID_IN_EP_ADDR                      0x81    //HID IN endpoint
address
#define USBD_HID_IN_EP_SIZE                      0x04    //HID IN endpoint
size
#define USBD_HID_FS_MP_SIZE                      0x40    //HID FS maximum
packet length
```

6.2 Configure USB Bottom Layer

The configuration items for delay, IO, clock, interrupt, and endpoint size of USB devices are USBD_HardwareInit() function in usbd_board.c file.

```
void USBD_HardwareInit(USBD_INFO_T* usbInfo)
{
    ...
    /* Configure FIFO */
    /* Configure RX FIFO */
    USB_OTG_ConfigRxFifoSize(usbDeviceHandler.usbGlobal, USBD_FS_RX_FIFO_SIZE);
    /* Configure TX FIFO */
    USBD_OTG_ConfigDeviceTxFifo(&usbDeviceHandler, 0, USBD_FS_TX_FIFO_0_SIZE);
    USBD_OTG_ConfigDeviceTxFifo(&usbDeviceHandler, 1, USBD_FS_TX_FIFO_1_SIZE);

    ...
}
```

It is important to notice that the endpoint number and size also need to be modified in the corresponding referenced class file, such as the following configuration items and initialization functions in the CDC Class.

```
#define USBD_CDC_FS_MP_SIZE                     0x40
#define USBD_CDC_HS_MP_SIZE                     0x200
#define USBD_CDC_CMD_MP_SIZE                    0x08
#define USBD_CDC_DATA_MP_SIZE                   0x07
#define USBD_CDC_CMD_EP_ADDR                    0x82
#define USBD_CDC_DATA_IN_EP_ADDR                0x81
#define USBD_CDC_DATA_OUT_EP_ADDR               0x01
#define USBD_CDC_FS_INTERVAL                    16
#define USBD_CDC_HS_INTERVAL                    16


USBD_STA_T USBD_CDC_ClassInitHandler(USBD_INFO_T* usbInfo, uint8_t cfgIndex)
```

6.3 Configure interrupt service function

The USB interrupt request function needs to be put in the relevant interrupt service functions.

6.4 Configure USB descriptors

The commonly used descriptors of USB devices are in usbd_descriptor.c file, including:

1. Device descriptor
2. Configuration descriptor
3. Configuration character string descriptor
4. Interface character string descriptor
5. Language ID character string descriptor
6. Manufacturer character string descriptor
7. Product character string descriptor
8. Serial number character string descriptor
9. Other speed descriptor
10. Device modification descriptor

6.5 Configuration USB data interfaces

Some classes require a data processing interface so they need to be registered in the USB core during USB device initialization.

6.6 Initialize USB device

The initialization function USB_DeviceInit() of USB device is in usb_device_user.c file, including the following functions:

1. Registration of Class data processing interface
2. Core speed selection
3. Descriptor registration
4. Class handler function registration
5. USB device user callback function registration

```
USBD_STA_T USBD_Init(USBD_INFO_T* usbInfo, USBD_SPEED_T usbDevSpeed, \
                     USBD_DESC_T* usbDevDesc, \
                     USBD_CLASS_T* usbDevClass, \
                     void (*userCallbackFunc)(struct _USBD_INFO_T*,
uint8_t))
{
    USBD_STA_T usbStatus = USBD_OK;

    /* Register descriptor function */
    if (usbDevDesc != NULL)
    {
        usbInfo->devDesc = usbDevDesc;
    }

    /* Register class function */
    if (usbDevClass == NULL)
    {
        usbStatus = USBD_FAIL;
        return usbStatus;
    }
    else
    {
        usbInfo->devClass[usbInfo->classNum++] = usbDevClass;
    }

    /* Register user application */
    usbInfo->userCallback = userCallbackFunc;

    usbInfo->devState = USBD_DEV_DEFAULT;
    /* Set USB device speed */
    usbInfo->devSpeed = usbDevSpeed;

    /* Init USB hardware */
    USBD_HardwareInit(usbInfo);

    return usbStatus;
}
```

7 Revision History

Table 9 Document Revision History

Date	Revision	Changes
July 20, 2023	1.0	First edition

Statement

This manual is formulated and published by Zhuhai Geehy Semiconductor Co., Ltd. (hereinafter referred to as "Geehy"). The contents in this manual are protected by laws and regulations of trademark, copyright and software copyright. Geehy reserves the right to correct and modify this manual at any time. Please read this manual carefully before using the product. Once you use the product, it means that you (hereinafter referred to as the "users") have known and accepted all the contents of this manual. Users shall use the product in accordance with relevant laws and regulations and the requirements of this manual.

1. Ownership of rights

This manual can only be used in combination with chip products and software products of corresponding models provided by Geehy. Without the prior permission of Geehy, no unit or individual may copy, transcribe, modify, edit or disseminate all or part of the contents of this manual for any reason or in any form.

The "Geehy" or "Geehy" words or graphics with "®" or "TM" in this manual are trademarks of Geehy. Other product or service names displayed on Geehy products are the property of their respective owners.

2. No intellectual property license

Geehy owns all rights, ownership and intellectual property rights involved in this manual.

Geehy shall not be deemed to grant the license or right of any intellectual property to users explicitly or implicitly due to the sale and distribution of Geehy products and this manual.

If any third party's products, services or intellectual property are involved in this manual, it shall not be deemed that Geehy authorizes users to use the aforesaid third party's products, services or intellectual property, unless otherwise agreed in sales order or sales contract of Geehy.

3. Version update

Users can obtain the latest manual of the corresponding products when ordering Geehy products.

If the contents in this manual are inconsistent with Geehy products, the agreement in Geehy sales order or sales contract shall prevail.

4. Information reliability

The relevant data in this manual are obtained from batch test by Geehy Laboratory or cooperative third-party testing organization. However, clerical errors in correction or errors caused by differences in testing environment may occur inevitably. Therefore, users should understand that Geehy does not bear any responsibility for such errors that may occur in this manual. The relevant data in this manual are only used to guide users as performance parameter reference and do not constitute Geehy's guarantee for any product performance.

Users shall select appropriate Geehy products according to their own needs, and effectively verify and test the applicability of Geehy products to confirm that Geehy products meet their own needs, corresponding standards, safety or other reliability requirements. If loses are caused to users due to the user's failure to fully verify and test Geehy products, Geehy will not bear any responsibility.

5. Compliance requirements

Users shall abide by all applicable local laws and regulations when using this manual and the matching Geehy products. Users shall understand that the products may be restricted by the export, re-export or other laws of the countries of the product suppliers, Geehy, Geehy distributors and users. Users (on behalf of itself, subsidiaries and affiliated enterprises) shall agree and promise to abide by all applicable laws and regulations on the export and re-export of Geehy products and/or technologies and direct products.

6. Disclaimer

This manual is provided by Geehy "as is". To the extent permitted by applicable laws, Geehy does not provide any form of express or implied warranty, including without limitation the warranty of product merchantability and applicability of specific purposes.

Geehy will bear no responsibility for any disputes arising from the subsequent design and use of Geehy products by users.

7. Limitation of liability

In any case, unless required by applicable laws or agreed in writing, Geehy and/or any third party providing this manual "as is" shall not be liable for damages, including any general damages, special direct, indirect or collateral damages arising from the use or no use of the information in this manual (including without limitation data loss or inaccuracy, or losses suffered by users or third parties).

8. Scope of application

The information in this manual replaces the information provided in all previous versions of the manual.