TOSHIBA

Application Note

USBD Mouse

TMPM365

Ver 1

Sep, 2017

TOSHIBA ELECTRONIC DEVICES & STORAGE CORPORATION

CMDR-M365UDH-01E

RESTRICTIONS ON PRODUCT USE

- DO NOT USE THIS SOFTWARE WITHOUT THE SOFTWARE LISENCE AGREEMENT.

Index

1. Introduction

The USBD HID driver can be easily re-used by user to realize the HID class.

This document will introduce the key points in creating a simple USB HID mouse demonstration program, which is based on Toshiba TMPM365 Evaluation board and USBD driver.

It is assumed that the reader has had some basic knowledge, such as “descriptor”, “endpoint” and “report”, for the USB and Human Interface Devices.

The formal documents for the knowledge mentioned above are available at website: http://www.usb.org :

- Universal Serial Bus Specification Revision 1.1, September 23, 1998,
  ➔ Please mainly read chapter 9.

- Device Class Definition for Human Interface Devices (HID), Firmware Specification—6/27/01 Version 1.11
  ➔ Please mainly read for HID class descriptor and report descriptor.

2. HID Mouse Demo

The HID class is primarily relevant to the implementation of devices which are normally used by end users to interface with the computer systems.

Typical examples of HID class devices include:

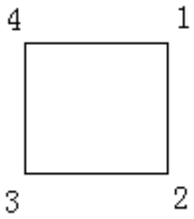
- Keyboard
- Pointing devices, i.e., standard mouse devices, trackballs, and joysticks.

On TMPM365 evaluation board, a simple HID demonstration program was designed to simulate a mouse device.

The prerequisite development environment must include:

- IAR Embedded Workbench for ARM 6.40.1 which should support M365
- M365 evaluation board (Not for sale)
  (Hereafter, we will call the evaluation board as “EVB” for short.)
- IAR J-Link-ARM v6.0
- PC (OS: Windows XP)

Once the EVB is connected via USB interface to PC and then powered on, it will be recognized to be a standard USB mouse and then it controls the arrow icon on PC screen as below, which will be repeated for 3 times.



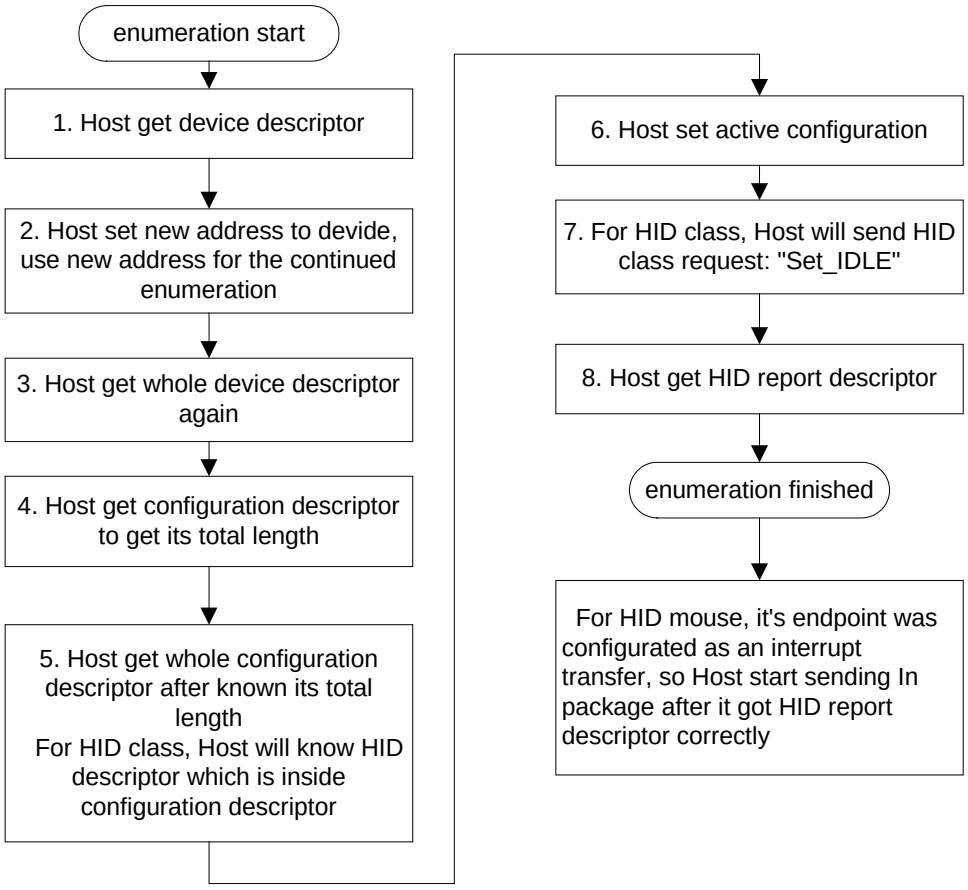
Step1: simulate a "right key click" at point 1
Step2: left move 10 pixel then slow move down to point 2, total 100 pixels
Step3: slow move left to point 3, total 100 pixels, then simulate a "left key click"
Step4: slow move up to point 4, total 100 pixels
Step5: slow move right to point 1, total 100 pixels
Step6: repeat from step 1 three times
(Note: the mouse arrow should be moved to central of screen at first)

3. Enumeration

Each time a device is plugged into the USB port of a host PC, it must be properly enumerated before entering its normal operating mode so as to transfer its data to the host PC.

The enumeration process contains some steps that host requests a number of descriptors to describe all attributes of the device.

Below are the simplified enumeration steps for HID Class device:



4. Descriptor

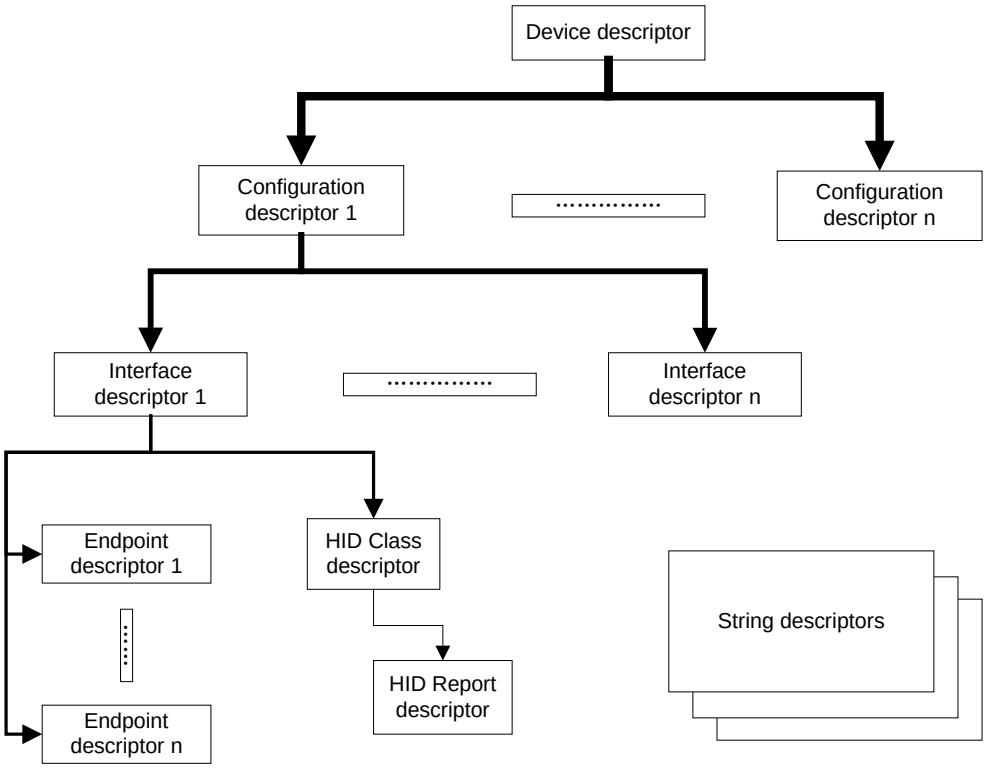
4.1 Device Descriptor

USB devices report their attributes to host by using descriptors.

A descriptor is a data structure with a specified format.
The first byte of descriptor indicates the total number of bytes in this descriptor.
The second byte identifies the descriptor type (Device, Configuration, Interface, Endpoint, Class, and so on…).
The remaining fields (bytes or words) are the content for this descriptor depending on its type.
(Refer to ***usbd_descriptor_hid.c*** for more details)

A USB device has only one device descriptor and one or more configuration descriptor. Each configuration has one or more interface descriptor. Each interface has one or more endpoint descriptor.

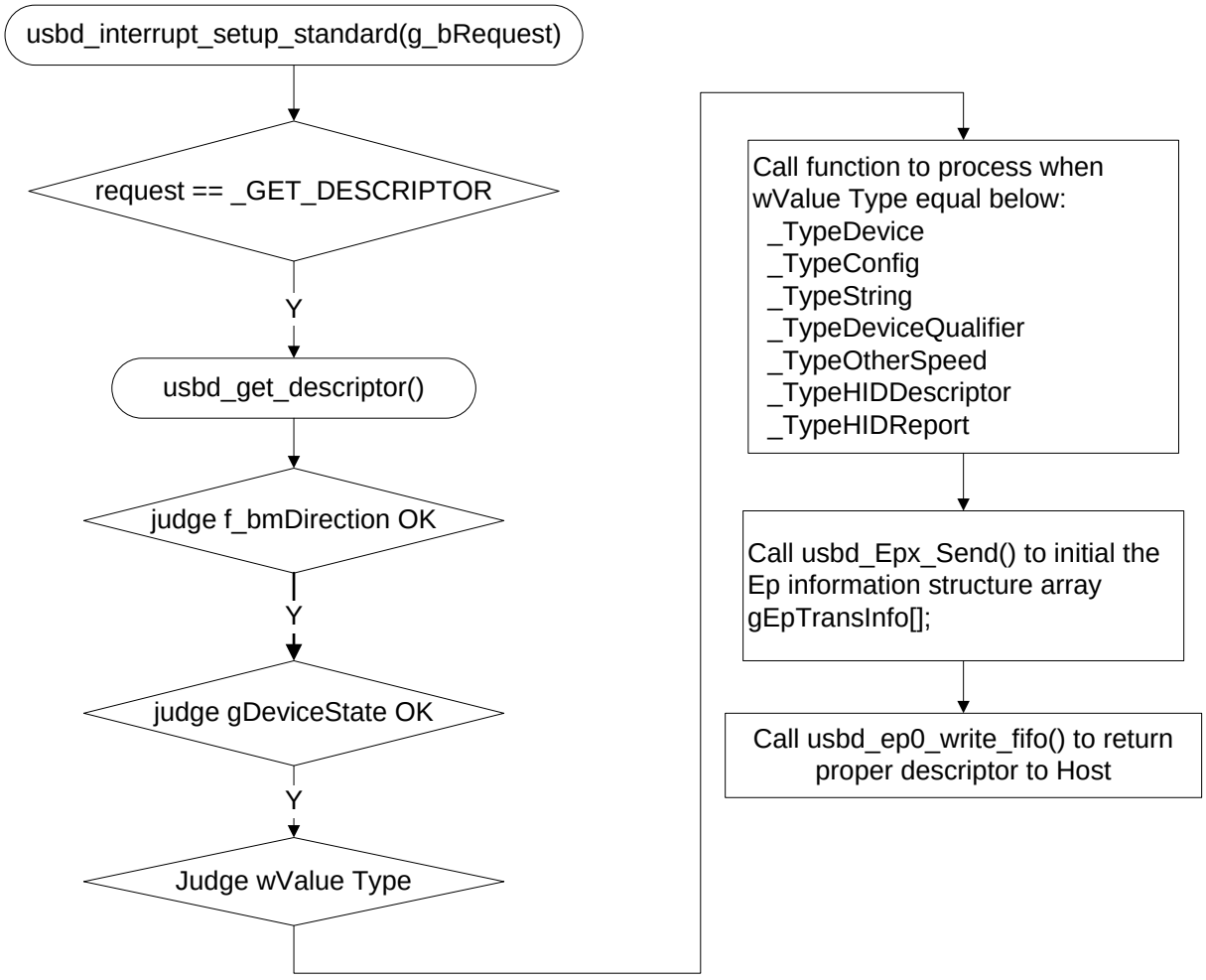
The layer structure of descriptor is as shown in the figure below:



4.2 Steps to Get Descriptor

As mentioned in enumeration part above, the most important work during enumeration is to return the required descriptor to host.

Below is the simplified flow for getting descriptors:



4.3 HID Report Descriptor

The HID report descriptor is composed of pieces of information. Each piece of information is called an ***Item***. The host parses the meaning of the data which the device sends to it, depending on this descriptor.

For example, the simulated mouse in this demo uses a 4-byte array to return the movement and key click information to host when the host sends the IN packet to ask for the data every 10mS (this value is defined by the item ***bInterval*** in endpoint descriptor structure type ***EndpointDescriptor_t***)

Byte0: Key	Byte1: X offset	Byte2: Y offset	Byte3: Z offset

For Byte0, bit0 is for left key, bit1 is for right key and bit2 is for middle key, bit value '1' means key is pressed and value '0' means key is up.
For X offset, 0x00 to 0x7F is Right and 0xFF to 0x80 is Left, unit is pixel.
For Y offset, 0x00 to 0x7F is Down, 0xFF to 0x80 is Up, unit is pixel.
Z offset isn't used in this demo.

The above report is described by the HID report descriptor listed below, It is not easy to understand this descriptor, the USB organization (http://www.usb.org) provides a tool to generate it.

```
const uint8_t HIDReportDescriptor[CbLength_HIDReportDescriptor] = {
    0x05U,  0x01U,          /* Usage Page (Generic Desktop)          */
    0x09U,  0x02U,          /* Usage (Mouse)                         */
    0xA1U,  0x01U,          /* Collection (Application)              */
    0x09U,  0x01U,          /*     Usage (Pointer)                   */
    0xA1U,  0x00U,          /*     Collection     (Physical)         */
    0x05U,  0x09U,          /*       Usage Page (Buttons)            */
    0x19U,  0x01U,          /*       Usage Minimum (01)              */
    0x29U,  0x03U,          /*       Usage Maximum (03)              */
    0x15U,  0x00U,          /*       Logical Minimum (0)             */
    0x25U,  0x01U,          /*       Logical Maximum (1)             */
    0x95U,  0x03U,          /*       Report Count      (1)           */
    0x75U,  0x01U,          /*       Report Size (1)                 */
    0x81U,  0x02U,          /*       Input (Data, Variable, Absolute)*/
    0x95U,  0x01U,          /*       Report Count      (1)           */
    0x75U,  0x05U,          /*       Report Size (5)                 */
    0x81U,  0x01U,          /*       Input (Constant) for padding    */
    0x05U,  0x01U,          /*       Usage Page (Generic Desktop)    */
    0x09U,  0x30U,          /*       Usage (X)                       */
```

```
        0x09U,  0x31U,          /*      Usage (Y)                       */
        0x09U,  0x38U,          /*      Usage (Z)                       */
        0x15U,  0x81U,          /*      Logical Minimum (-127)          */
        0x25U,  0x7FU,          /*      Logical Maximum (127)           */
        0x75U,  0x08U,          /*      Report Size (8)                 */
        0x95U,  0x03U,          /*      Report Count    (2)             */
        0x81U,  0x06U,          /*      Input (Data, Variable, Relative)*/
        0xC0U,                  /*    End Collection(Physical)      */
        0xC0U                   /* End Collection (Application)       */
};
```

5. Software File Structure

The main file structure of this demo is shown as below:

```
/---HID_Mouse
+---APP
|     |     hid_mouse.c
|     |
|     +---hid_inc
|     |           usbd_descriptor_hid.h
|     |           usbd_hid.h
|     |
|     \---hid_src
|                 usbd_descriptor_hid.c
|                 usbd_hid.c
|
+---TX03_CMSIS
|     |     system_TMPM366.c
|     |     system_TMPM366.h
|     |     TMPM366.h
|     |
|     \---startup
|                 startup_TMPM366.s
|
+---TX03_Periph_Driver
|     +---inc
|     |           tmpm366_cg.h
|     |           tmpm366_gpio.h
|     |           tx03_common.h
|     |           usbd_hw.h
|     |
|     \---src
|                 tmpm366_cg.c
|                 tmpm366_gpio.c
|                 usbd_hw.c
\---USBD_Common
       +---inc
       |           usbd_descriptor.h
       |           usbd_device_request.h
       |           usbd_hw_com.h
       |           usbd_hw_interrupt.h
```

```
|           usbd_trans.h
|           usbd_typedefs.h
|           usbd_var.h
|
\---src
            usbd_device_request.c
            usbd_hw_com.c
            usbd_hw_interrupt.c
            usbd_trans.c
            usbd_var.c
```

6. Key Code and Flowchart

The key codes and flowchart for this demo software are listed below:

6.1 Key code

1. Set HID initialize parameters.

```
gSample_HID_CB.pfnConnStat        = _Sample_ConnStat;
gSample_HID_CB.pfnSuspend         = NULL;
gSample_HID_CB.pfnResume          = NULL;
gSample_HID_CB.pfnGetReport       = NULL;
gSample_HID_CB.pfnSetReport       = NULL;
gSample_HID_CB.pfnGetIdle         = NULL;
gSample_HID_CB.pfnSetIdle         = NULL;
gSample_HID_CB.pfnGetProtocol     = NULL;
gSample_HID_CB.pfnSetProtocol     = NULL;
gSample_HID_CB.pfnSendData        = _Sample_SendData;
gSample_HID_CB.pfnCtrlSendData = NULL;
gSample_HID_CB.pfnCtrlRecvData = NULL;
```

_Sample_ConnStat() is a call back function, which will be called if the usb device is connect to the pc.it call function USBD_HID_SendData() to initial the transfer EP information.

```
static void _Sample_ConnStat(bool bStat)
{
    if (bStat == true) {
        /* Connect */
        /* Send input report */
        gbStatDC = USBD_HID_SendData(sizeof(SampleInputReport_t), (uint8_t
*)&l_tSampleInputReport[l_ucSampleIndexNum]);
        /* No care status in sample */
    } else {
        /* Disconnect */
        /* Do nothing in sample */
    }
}
```

_Sample_SendData() is a also a call back function. It's used to send the data the pc, which will be called when data transfer is complete. The function

USBD_HID_SendData() called in it use to initial the transfer EP information for next time transfer. USBD_HID_SendData() will be call for three time, and the the mouse pointer will cycle for three times in the pc desktop.

```
static void _Sample_SendData(uint32_t ulSize, uint8_t *pucData, uint16_t usStat)
{
    uint16_t    i;

    if (usStat == EPSTAT_COMPLETE) {
        /* Update repeat count */
        if (++l_ucSampleRepeatCnt ==
l_aucSampleInputReportCount[l_ucSampleIndexNum]) {
            l_ucSampleRepeatCnt = 0U;

            /* Update table index number */
            if (++l_ucSampleIndexNum == SAMPLEINPUTREPORT_NUM) {
                l_ucSampleIndexNum = 0U;
                CyclyCnt++;
            }
        }

        /* Wait for a while */
        for (i=0; i<5U; i++) {
            usbd_wait_1ms();
        }

       if(CyclyCnt < 3U){
        /* Send input report */
        gbStatDC = USBD_HID_SendData(sizeof(SampleInputReport_t), (uint8_t
*)&l_tSampleInputReport[l_ucSampleIndexNum]);
        }
        /* No care status in sample */
    }

    return;
}
```

2. Initialize HID driver

```
void usbd_hid_init(HID_CB_t *p_hid_cb)
{
    uint8_t ucCnt;

    /* Set of callback functions */
```

```
    gHID_CB.pfnConnStat        = p_hid_cb->pfnConnStat;
    gHID_CB.pfnResume          = p_hid_cb->pfnResume;
    gHID_CB.pfnSuspend         = p_hid_cb->pfnSuspend;
    gHID_CB.pfnGetReport       = p_hid_cb->pfnGetReport;
    gHID_CB.pfnSetReport       = p_hid_cb->pfnSetReport;
    gHID_CB.pfnGetIdle         = p_hid_cb->pfnGetIdle;
    gHID_CB.pfnSetIdle         = p_hid_cb->pfnSetIdle;
    gHID_CB.pfnGetProtocol     = p_hid_cb->pfnGetProtocol;
    gHID_CB.pfnSetProtocol     = p_hid_cb->pfnSetProtocol;
    gHID_CB.pfnSendData        = p_hid_cb->pfnSendData;
    gHID_CB.pfnCtrlSendData = p_hid_cb->pfnCtrlSendData;
    gHID_CB.pfnCtrlRecvData = p_hid_cb->pfnCtrlRecvData;

    /* Initialization of an internal variable */
    l_bUsbStat        = false;
    *l_pucProtocol = (uint8_t)USBD_HID_REPORT_PROTOCOL;

    for(ucCnt=0U; ucCnt<USBD_HID_REPORT_ID_NUM; ucCnt++)
    {
        l_aucIdleRate[ucCnt] = 0U;
    }

    return;
}
```

3. Initialize work data

```
void usbd_initialize_standard_class_work_data(void)
{
    const ConfigDescriptor_t *config;
    ConfigDescriptor_bmAttributes_t *attribute;

    gUDC2Addr.All = CgUD2ADDR_INIT;
    gUDC2AddrBuf.All = CgUD2ADDR_INIT;

    fEP0StallFeature = CEPxStallFeature_OFF;
    fEP1StallFeature = CEPxStallFeature_OFF;
    fEP2StallFeature = CEPxStallFeature_OFF;
    fEP3StallFeature = CEPxStallFeature_OFF;

    gEP0Payload_Size = CwMaxPacketSize_EP0;
    gEP1Payload_Size = CwMaxPacketSize_EP1_INIT;
    gEP2Payload_Size = CwMaxPacketSize_EP2_INIT;
    gEP3Payload_Size = CwMaxPacketSize_EP3_INIT;
```

```
    gEPxConfigArray[USBD_EP1] = 0x88U;     /* EP1 as BULK IN for CDC,MSC */
    gEPxConfigArray[USBD_EP2] = 0x08U;     /* EP2 as BULK OUT for CDC,MSC */
    gEPxConfigArray[USBD_EP3] = 0x8CU;     /* EP3 as INTERRUPT IN for CDC,HID
*/

    s_Buf_Current_Config = CbConfigurationValue_Init;
    s_Buf_Current_Interface = CbInterfaceNumber_Init;
    s_Buf_Current_Alternate = CbAlternateSetting_Init;

    s_Current_Config = s_Buf_Current_Config;
    s_Current_Interface = s_Buf_Current_Interface;
    s_Current_Alternate = s_Buf_Current_Alternate;

    g_USB_Stage = IDLE_STAGE;

    memset(gEpTransInfo, 0x00U, sizeof(gEpTransInfo));

    /* make status device result       */
    sGetStatusDevice = CsGetStatusDevice_INIT;
    config = gStructConfigDesc[CbConfigurationValue1 - 1].p_config;
    attribute = (ConfigDescriptor_bmAttributes_t *) config->bmAttributes;
    fSelfPowered = attribute->SelfPowered;
    fRemoteWakeup = attribute->RemoteWakeup;

    return;
}
```

4. Configure USBD module:
    - Enable USB clock supply
    - Enable the pull up resistor in D+ pin
    - Power on reset for USBD module
    - Set USB interrupt masks
    - Enable USBD interrupt
    - Reset endpoint 0

```
void Config_USB(void)
{
    USBD_PowerCtrl pwr = { 0U };

    TSB_CG->USBCTL = 0x100U;       /* enable USB Clock */

    /*USBON pin config*/
    GPIO_SetInput(GPIO_PG, GPIO_BIT_5);
```

```
GPIO_SetInputEnableReg(GPIO_PG, GPIO_BIT_5, ENABLE);

/* pin PE4 control the pull up of D+, must be set output '1' */
GPIO_SetOutput(GPIO_PE, GPIO_BIT_4);
GPIO_WriteDataBit(GPIO_PE, GPIO_BIT_4, GPIO_BIT_VALUE_1);

USBD_SetINTMask(USBD_INT_USB_RESET_END, ENABLE);
USBD_SetINTMask(USBD_INT_USB_RESET, ENABLE);

/*   UDPWCTL Power Reset and                */
/*         PHY Reset & Suspend: 1ms         */
pwr = USBD_GetPowerCtrlStatus();
pwr.Bit.PW_Resetb = 0U;
pwr.Bit.PHY_Resetb = 0U;
pwr.Bit.PHY_Suspend = 1U;
USBD_SetPowerCtrl(pwr);
usbd_wait_1ms();

/* PHY Reset off : 1ms                     */
pwr = USBD_GetPowerCtrlStatus();
pwr.Bit.PHY_Resetb = 1U;
pwr.Bit.PHY_Suspend = 1U;
USBD_SetPowerCtrl(pwr);
usbd_wait_1ms();

/*   PHY Suspend off : 1ms                  */
pwr = USBD_GetPowerCtrlStatus();
pwr.Bit.PHY_Suspend = 0U;
USBD_SetPowerCtrl(pwr);
usbd_wait_1ms();

usbd_wait_1ms();

/*   UDPWCTL Power Reset Off: 1ms           */
pwr = USBD_GetPowerCtrlStatus();
pwr.Bit.PW_Resetb = 1U;
USBD_SetPowerCtrl(pwr);
usbd_wait_1ms();
usbd_wait_1ms();
pwr = USBD_GetPowerCtrlStatus();

/* clear pending UDC2 interrupt and disable INT for SOF and NAK */
USBD_WriteUDC2Reg(UDC2_INT, 0x90FFU);
```

```
    USBD_SetEPCMD(USBD_EP0, USBD_CMD_ALL_EP_INVALID);
    USBD_SetEPCMD(USBD_EP0, USBD_CMD_USB_READY);

    NVIC_EnableIRQ(INTUSB_IRQn);

// VBUS
    NVIC_EnableIRQ(INTUSBPON_IRQn);

    return;
}
```

5. After proper configuration, the firmware will wait for USB interrupt and then finish the standard enumeration steps so as to be treated as an HID mouse in interrupt service routine.

   For more details, please refer to the function **INTUSB_IRQHandler()** and **INTUSBPON_IRQHandler()** in ***usbd_hw_interrupt.c*** and function **usbd_interrupt_setup_standard()** in ***usbd_device_request.c*** and **usbd_interrupt_HID_req()** in ***usbd_hid.c.***

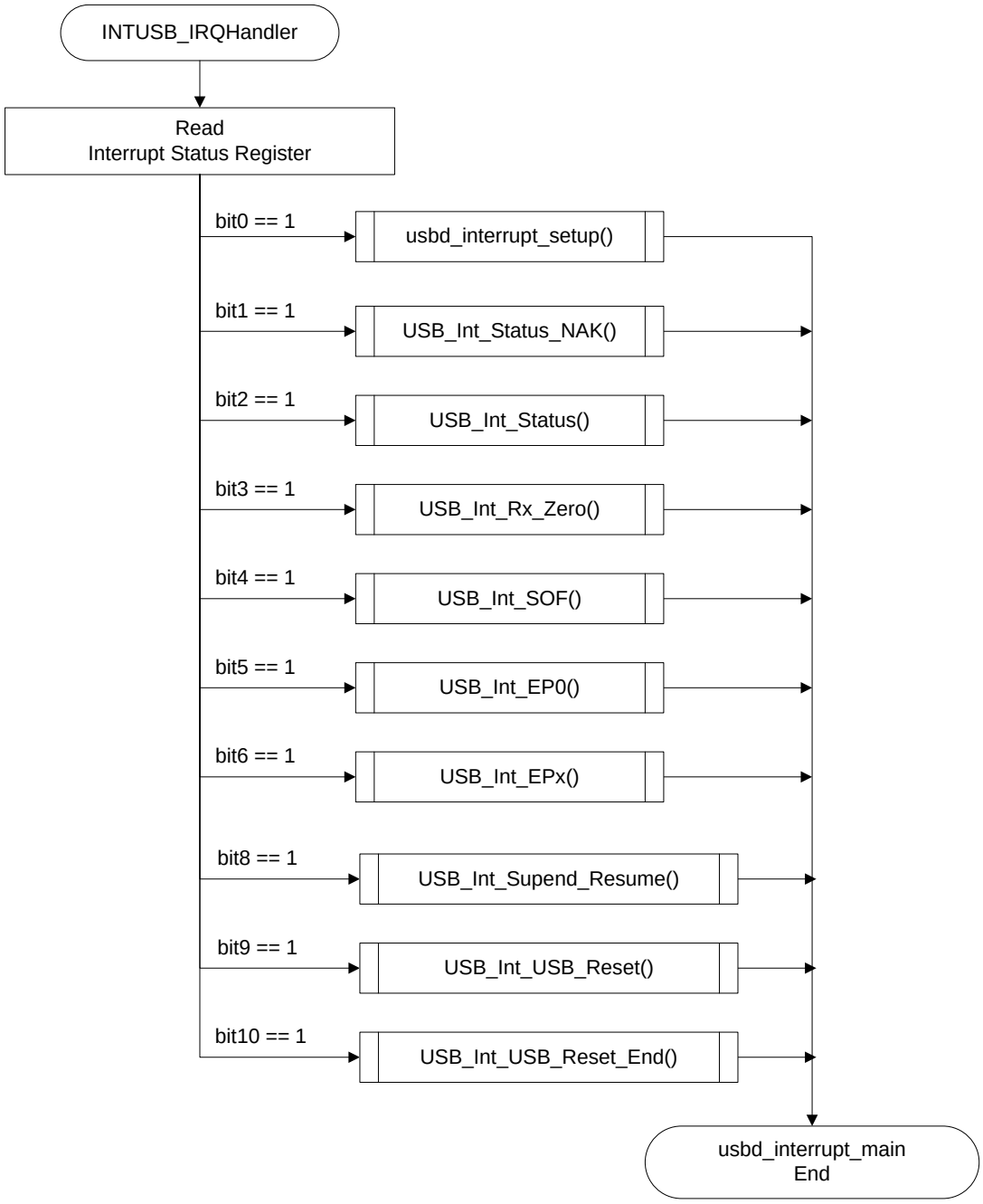
6. After enumeration finished, the firmware starts to send mouse data to FIFO of endpoint 3 (interrupt transfers) when it is empty to simulate the mouse movement and key click.

   For more detailed contents, please refer to the condition to call the function **_Sample_SendData()** and data array **l_tSampleInputReport[]** in ***hid_mouse.c***

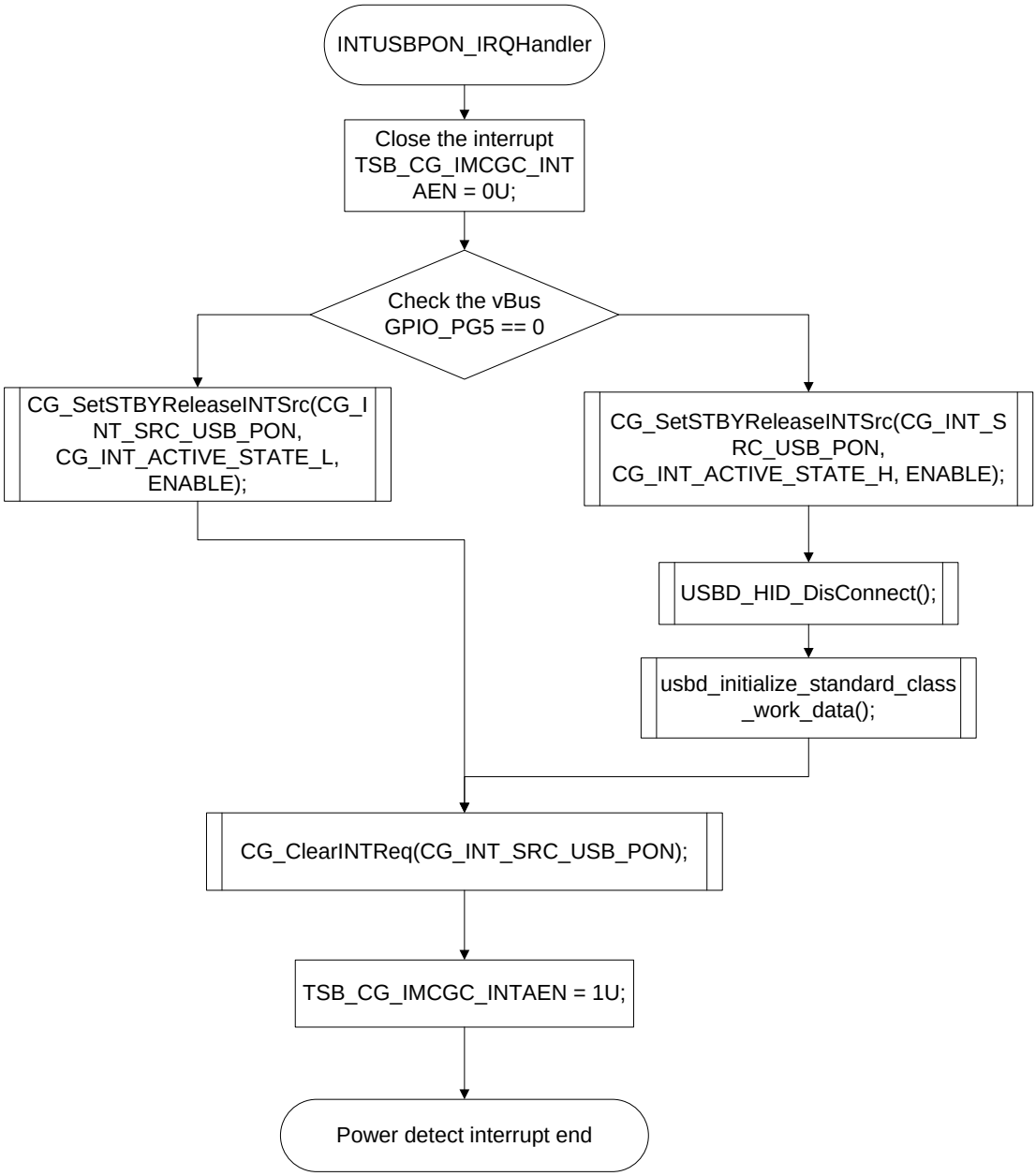
7. To avoid affecting the normal mouse usage, the data sending is stopped after three times demonstration.

6.2 Flowchar

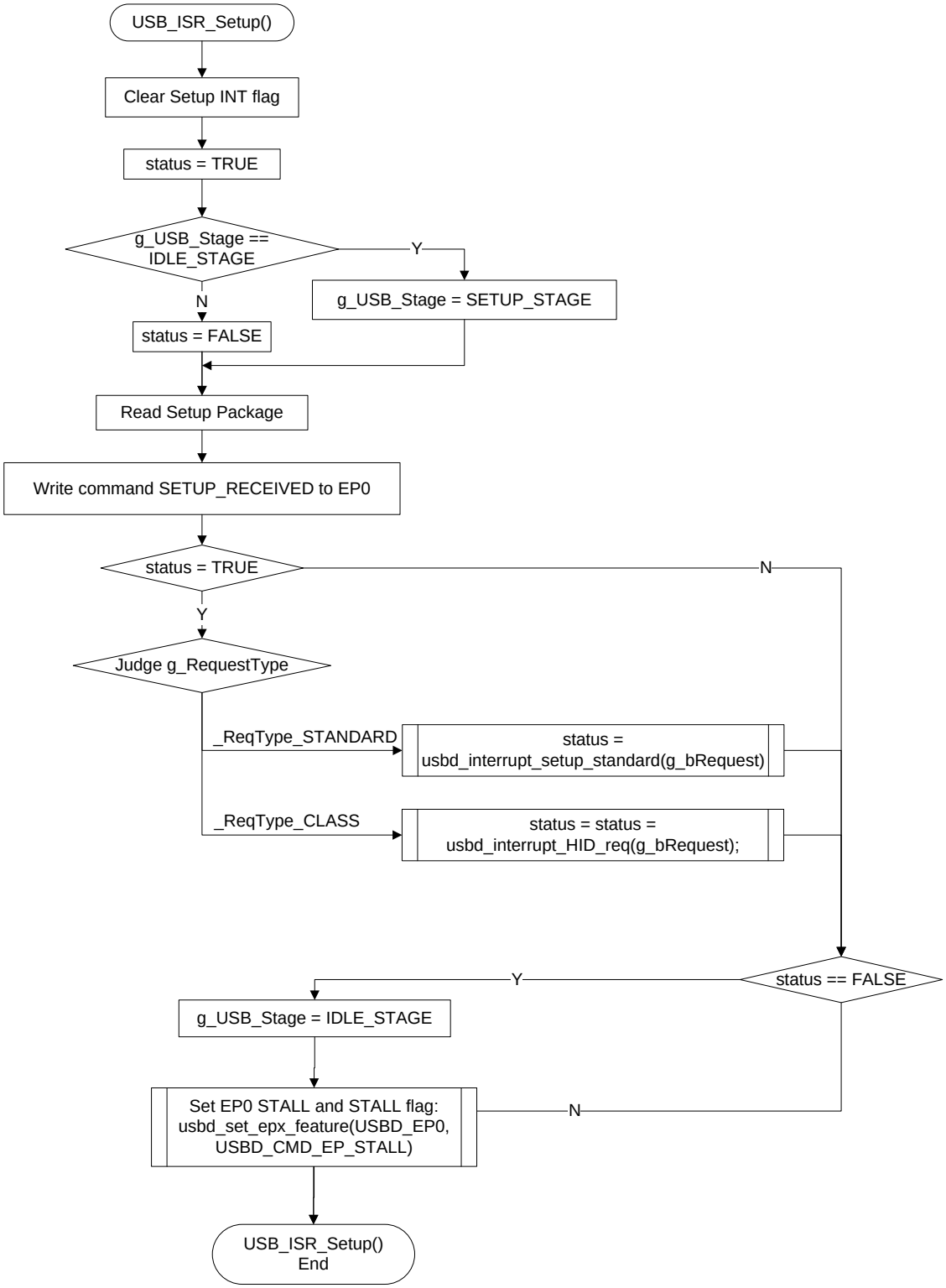
The following flowchart is for the main USB interrupts routine



The following flowchart is for VBUS detect Interrupt Service Routine.



The following flowchart is to get setup package and process it.



7. Call back function

The following structure is used to record the callback function and help to initial the system.

```
typedef struct {
    USBD_HID_ConnStat          pfnConnStat;
    USBD_HID_Notice            pfnSuspend;
    USBD_HID_Notice            pfnSuspend;
    USBD_HID_Report_Req        pfnGetReport;
    USBD_HID_Report_Req        pfnSetReport;
    USBD_HID_GetIdle_Req       pfnGetIdle;
    USBD_HID_SetIdle_Req       pfnSetIdle;
    USBD_HID_GetProtocol_Req   pfnGetProtocol;
    USBD_HID_SetProtocol_Req   pfnSetProtocol;
    USBD_HID_TrnsData          pfnSendData;
    USBD_HID_TrnsData          pfnCtrlSendData;
    USBD_HID_TrnsData          pfnCtrlRecvData;
} HID_CB_t;
```

They can be divided into three kind of function:
1. Hardware response:
pfnConnStat; pfnSuspend; pfnSuspend;

2. HID Class requests: they will be call when the host sent out the request to the device.
pfnGetReport; pfnSetReport; pfnGetIdle; pfnSetIdle; pfnGetProtocol; pfnSetProtocol;

3. Data transfer:
pfnSendData: Transfer data to EP3 in the interrupt transfer. When call function "***bool USBD_HID_SendData(uint32_t ulSize, uint8_t *pucData)***",this call back function will be called.

pfnCtrlSendData: Transfer data to EP0 in the control transfer. When call function "***bool USBD_HID_CtrlSendData(uint32_t ulSize, uint8_t *pucData)***", this call back will be execute.

pfnCtrlRecvData: Receive data from EP0 in the control transfer. When call function "***bool USBD_HID_CtrlRecvData(uint32_t ulSize, uint8_t *pucData)***", this call back function will be execute.