

TOSHIBA

アプリケーションノート

USBD マウス

TMPPM365

第一版

2017 年 9 月

東芝デバイス&ストレージ株式会社

本製品取り扱い上のお願い

- ソフトウェア使用権許諾契約書の同意無しに使用しないで下さい。

目次

1. 概要.....	1
2. HID マウスサンプルプログラム	2
3. 動作環境.....	3
4. 列挙(Enumeration)	4
5. ディスクリプタ	5
5.1 デバイスディスクリプタ.....	5
5.2 ディスクリプタ取得手順	6
4.3 HID リクエスト.....	7
6. ソフトウェアファイル構造	9
7. キーコードとフローチャート	11
7.1 キーコード.....	11
7.2 フローチャート	17
8. コールバック関数.....	20

1. 概要

USB D HID ドライバは、HID クラスの再利用を容易にするドライバです。

本資料は、東芝 TMPM365 評価ボードと USB D ドライバを使用し、簡易な USB HID マウス サンプル プログラムを作成する際のキーポイントを紹介しています。

ユーザは、USB および通信デバイスクラス用の「ディスクリプタ」、「エンドポイント」、「リクエスト」などの基本知識を有していることを想定しています。

上記内容の正式な資料は、下記ウェブサイトにて入手可能です。

<http://www.usb.org> :

- Universal Serial Bus Specification Revision 1.1, September 23, 1998,
➔ 主に 9 章をご覧ください。
- Device Class Definition for Human Interface Devices (HID), Firmware Specification—6/27/01 Version 1.11
➔ 主に HID クラスディスクリプタとレポートディスクリプタをご覧ください。

2. HID マウスサンプルプログラム

HID クラスは、通常コンピューターシステムのインタフェースと共に、ユーザにより使用されるデバイスの実装に主に関連しています。

HID クラスの代表的な例

- キーボード
- ポインティングデバイス(標準マウスデバイス、トラックボール、ジョイスティック)

本 HID サンプルプログラムは、TMPM365 評価ボード上にて、マウスデバイスのシミュレーションを行うように設計されています。

3. 動作環境

動作環境は以下のようになります。

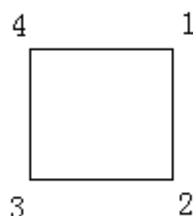
ハードウェア環境

- M365 評価ボード (非売品)
(以降、評価ボードを“EVB”と省略)
- IAR J-Link-ARM v6.0
- USB の D+ピンを EVB の G15 番ピンに接続
- USB の D- ピンを EVB の H15 番ピンに接続
- USB の VBUS ピンを EVB の P1 番ピンに接続
- USB の GND ピンを EVB の GND ピンに接続

ソフトウェア環境

- M365 をサポート可能な IAR Embedded Workbench for ARM 6.401
- PC (OS: Windows XP)

EVB とホスト PC を USB インタフェース経由で接続し、電源 ON すると、EVB が標準マウスとして認識されます。その後、PC モニタ上に次のような矢印アイコンを移動し、これを 3 回繰り返します。



手順

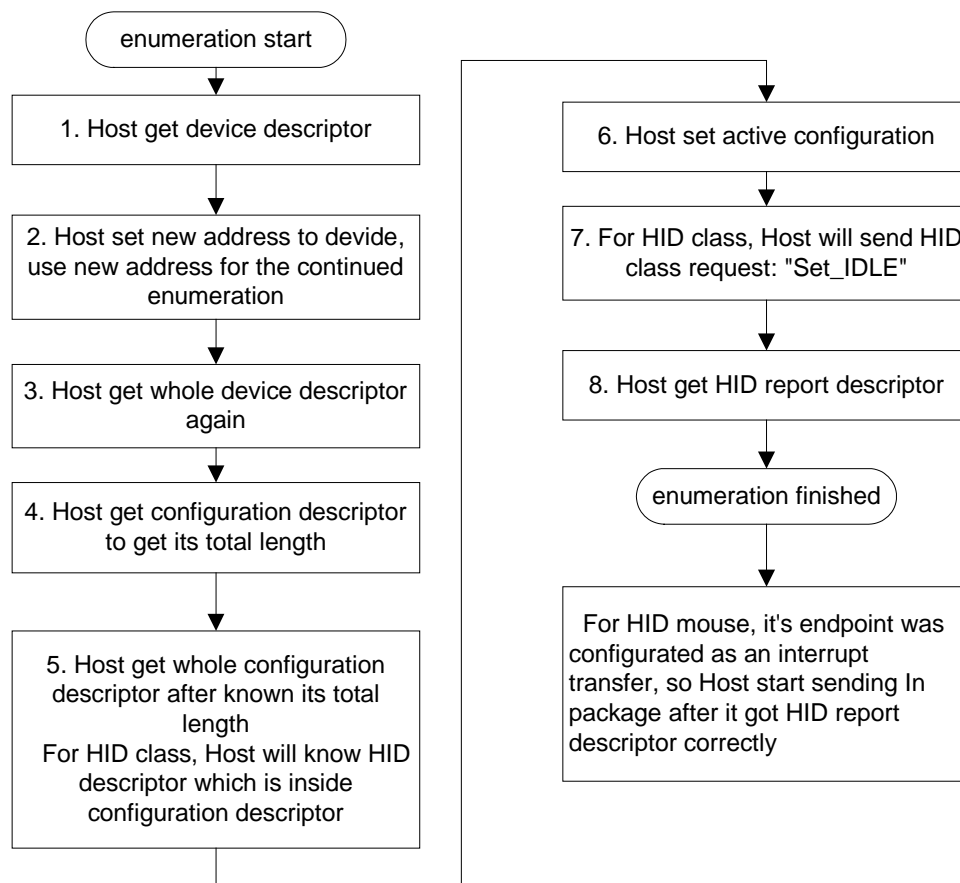
- 1: ポイント 1 にて右クリックします。
 - 2: ポイント 2 までゆっくり下へ 100 ピクセル分移動します。
 - 3: ポイント 3 までゆっくり左へ 100 ピクセル分移動します。その後左クリックします。
 - 4: ポイント 4 までゆっくり上へ 100 ピクセル分移動します。
 - 5: ポイント 1 までゆっくり右へ 100 ピクセル分移動します。
 - 6: 手順 1 に戻り、3 回繰り返します。
- (補足: 最初にマウスポインタを画面中央に移動してください。)

4. 列挙(Enumeration)

デバイスがホスト PC の USB ポートに接続される毎に、データをホスト PC へ転送するために、デバイスはノーマル動作モードに入る前に、適正に列挙される必要があります。

本列挙プロセスには、ホストがデバイスの全ディスクリプタを記載するため、以下のステップでディスクリプタの要求が行われます。

下図は HID クラスデバイスを列挙化する簡易化したステップです。



5. ディスクリプタ

5.1 デバイスディスクリプタ

USB デバイスは、ディスクリプタを使用して、自身の属性をホストへ報告します。

ディスクリプタは、指定のフォーマットで構成されたデータです。

ディスクリプタの第 1 番目のバイトは、ディスクリプタの全バイト数を表わします。

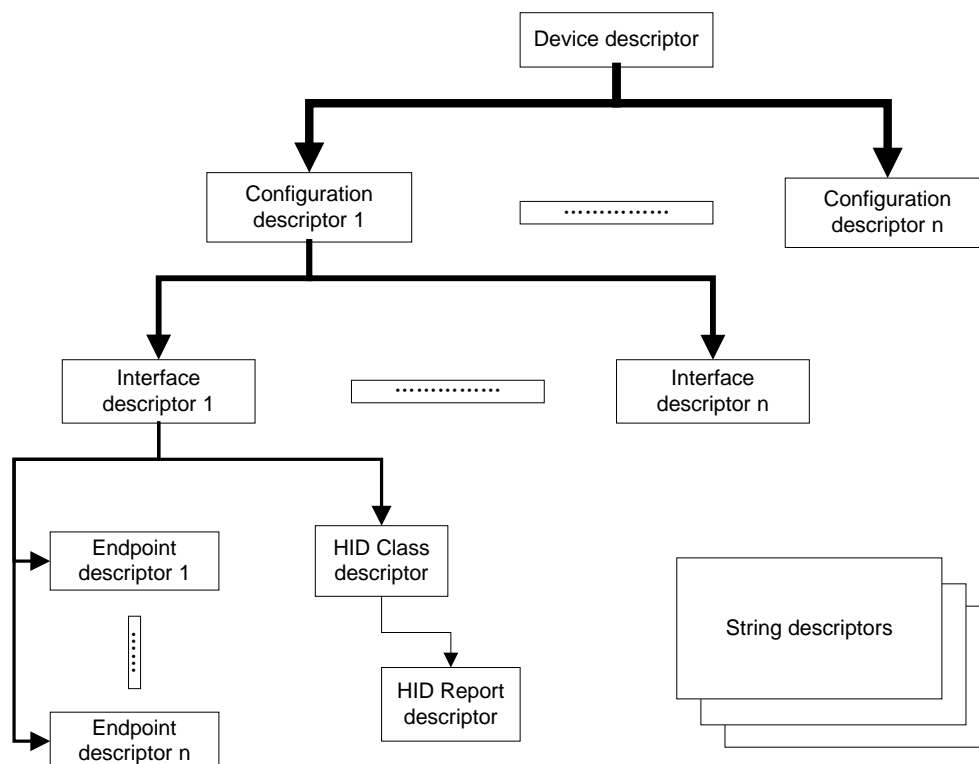
第 2 番目のバイトでは、ディスクリプタのタイプ(デバイス、構成、インタフェース、エンドポイント、クラスなど)が確認できます。

残りのフィールド(バイトまたはワード)は、本ディスクリプタの内容が記載され、ディスクリプタのタイプにより異なります。

(詳細は、`usbd_descriptor_hid.c` を参照してください。)

USB デバイスは、1 つのデバイスディスクリプタと 1 つ以上の構成ディスクリプタを備えています。各構成は、1 つ以上のインタフェースディスクリプタを備え、各インタフェースは 1 つ以上のエンドポイントディスクリプタを備えています。

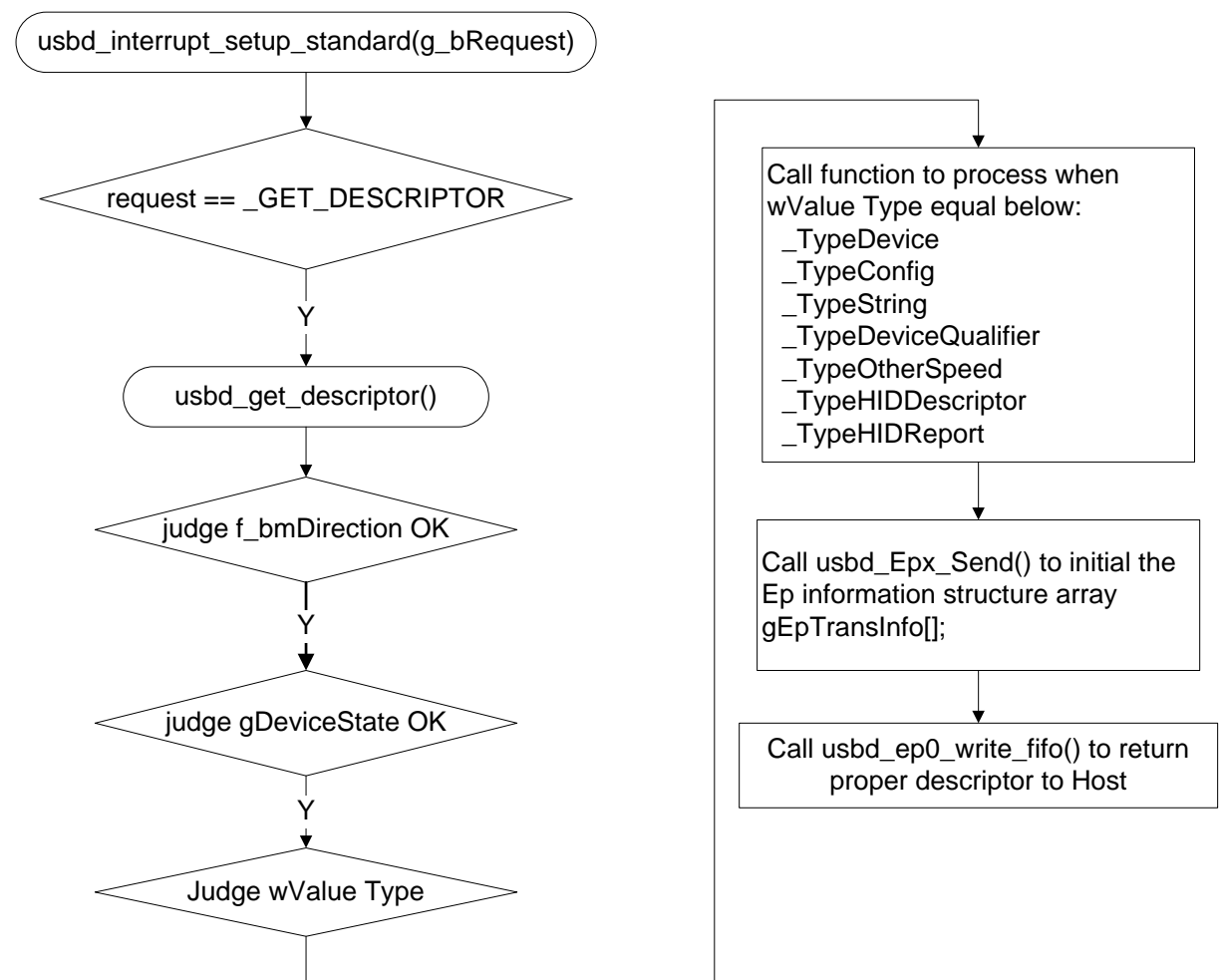
ディスクリプタの階層構造を下図に示します。



5.2 ディスクリプタ取得手順

上記列挙化で述べたように、列挙化で最も重要な作業は、必要なディスクリプタをホストへ返信することです。

下記に簡易化したディスクリプタ取得フローを示します。



4.3 HID リクエスト

HIDレポートディスクリプタは、いくつかの情報で構成されています。各情報は、**Item**と呼ばれます。本ディスクリプタにより、ホストはどのデバイスがデータを送信したのか、データの意味を解析します。

例えば、本サンプルプログラムにおいてテストされたマウスは、ホストが10msごとのデータを取得するためINパケットを送信している場合、マウスの動作とキーによるクリック情報をホストに返すために4バイトのデータ列を使用します。(本値は、エンドポイントディスクリプタ構造タイプ **EndpointDescriptor_t** アイテム **Interval**により定義されます。)

Byte	Bit	説明
0	0	レフトボタン
	1	ライトボタン
	2	ミドルボタン
	上記以外	未使用
1		X座標の移動量
2		Y座標の移動量
上記以外		未使用

バイト0は、ビット0をレフトボタン、ビット1をライトボタン、ビット2をミドルボタンとして使用し、それ以外のボタンは未使用です。ビットの値が“1”の場合は、ボタンDownとして、“0”の場合は、ボタンUpとして使用します。

バイト1は、X座標の(前回の位置座標からの)移動量を格納し、0x00~0x7Fが右方向、0xFF~0x80が左方向として使用します。(単位はピクセルです。)

バイト2は、Y座標の(前回の位置座標からの)移動量を格納し、0x00~0x7Fが下方向、0xFF~0x80が上方向として使用します。(単位はピクセルです。)

上記以外のバイトは、未使用です。

上記レポートは、下記のHIDレポートにより記述されます。

USB.org(<http://www.usb.org>)より、ディスクリプタ生成ツールが提供されています。

```
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)	*/

};

6. ソフトウェアファイル構造

本サンプルプログラムの主要ファイル構造を下記に示します。

```
/---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_TMPM365.c
|   |   system_TMPM365.h
|   |   TMPM365.h
|   |
|   \---startup
|           startup_TMPM365.s
|
+---TX03_Periph_Driver
|   +---inc
|   |       tmpm365_cg.h
|   |       tmpm365_gpio.h
|   |       tx03_common.h
|   |       usbd_hw.h
|   |
|   \---src
|           tmpm365_cg.c
|           tmpm365_gpio.c
|           usbd_hw.c
\---USB_D_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
```

7. キーコードとフローチャート

本サンプルプログラム用のキーコードおよびフローチャートを下記に示します。

7.1 キーコード

1. HID 初期パラメータを設定します。

```
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 () は、コールバック関数です。USB デバイスが PC に接続されるとコールされます。本関数は、EP 転送情報を受信するために関数 USB_D_HID_SendData ()をコールします。

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

_Sample_SendData は、コールバック関数です。PC へのデータ送信に使用されます。データ転送が完了すると、本関数がコールされます。本関数の中でコールさ

れる関数 USBD_HID_SendData は、次回転送用 EP 転送情報を初期化するために使用します。USBD_HID_SendData()は3回コールされ、マウスポインタはPCモニタ上で3週します。

```
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;
            }
        }

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

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

    return;
}
```

2. HIDドライバを初期化します。

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

    /* Set of callback functions */
    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. 作業データを初期化します。

```
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. USBBD モジュールを構成します。

- USB クロック供給の許可
- D+端子上プルアップレジスタの許可
- USBBD モジュール用パワーオンリセット
- USB 割り込みマスク設定
- USBBD 割り込みの許可
- エンドポイント 0 リセット

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

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

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

    /* pin PA0 control the pull up of D+, must be set output '1' */
    GPIO_SetOutput(GPIO_PA, GPIO_BIT_0);
    GPIO_WriteDataBit(GPIO_PA, GPIO_BIT_0, GPIO_BIT_VALUE_1);

    USBBD_SetINTMask(USBBD_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. 正常構成後、ファームウェアは USB 割り込みを待ち、その後割り込みルーチンで HID マウスとして扱うための標準列挙化のステップを終了させます。

詳細は、*usbd_hw_interrupt.c* 内の *INTUSB_IRQHandler()*関数と *INTUSBPON_IRQHandle()*関数を、*usbd_device_request.c* 内の *usbd_interrupt_setup_standard()*関数を、*usbd_hid.c* 内の *usbd_interrupt_HID_req()*関数をそれぞれ参照してください。

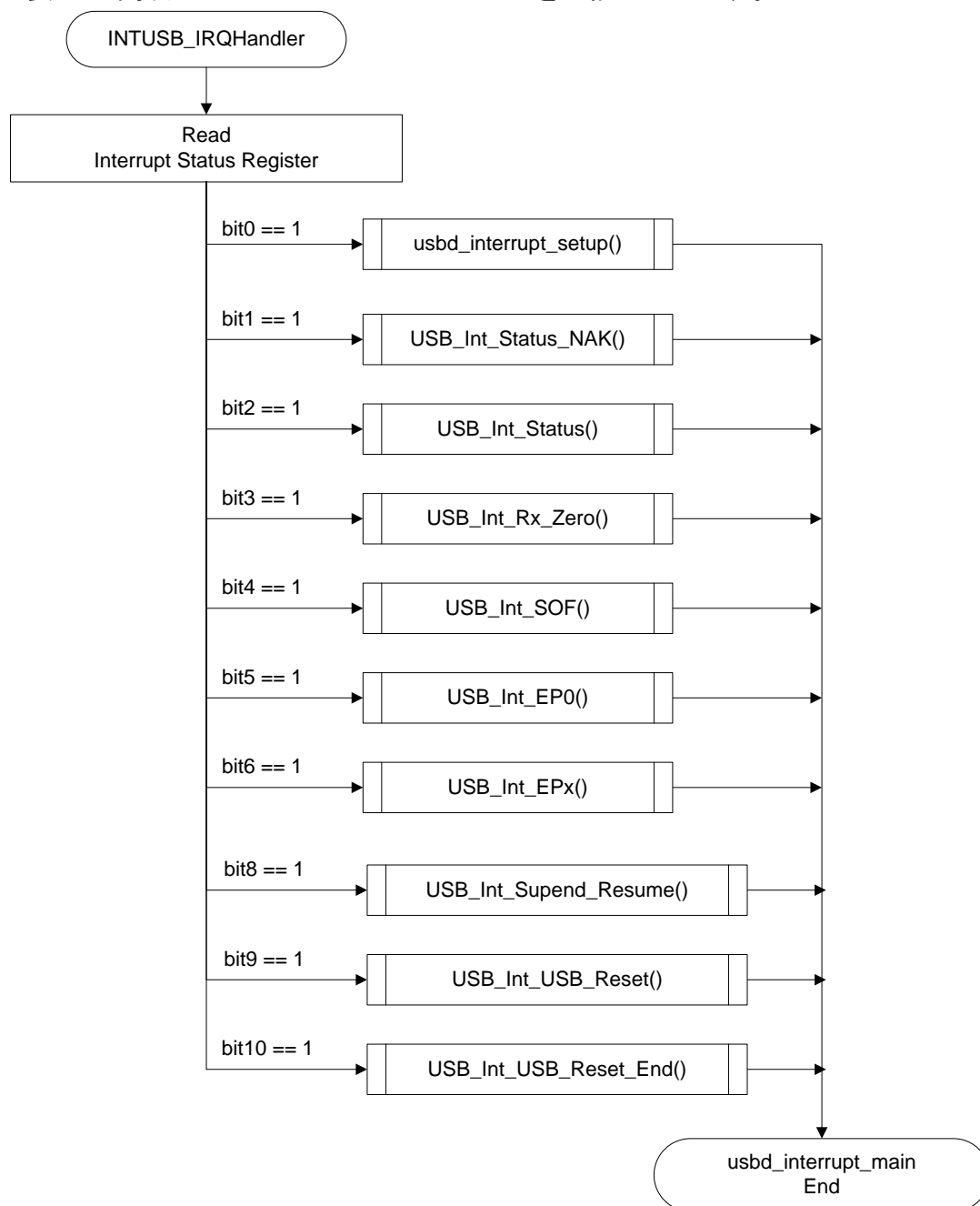
6. 列挙化終了後、ファームウェアはマウス動作とボタン操作がなければエンドポイント 3(割り込み転送)の FIFO へマウスデータを送信します。

詳細は、*hid_mouse.c* 内の *_Sample_SendData()*関数と *I_tSampleInputReport[]*データ配列を参照してください。

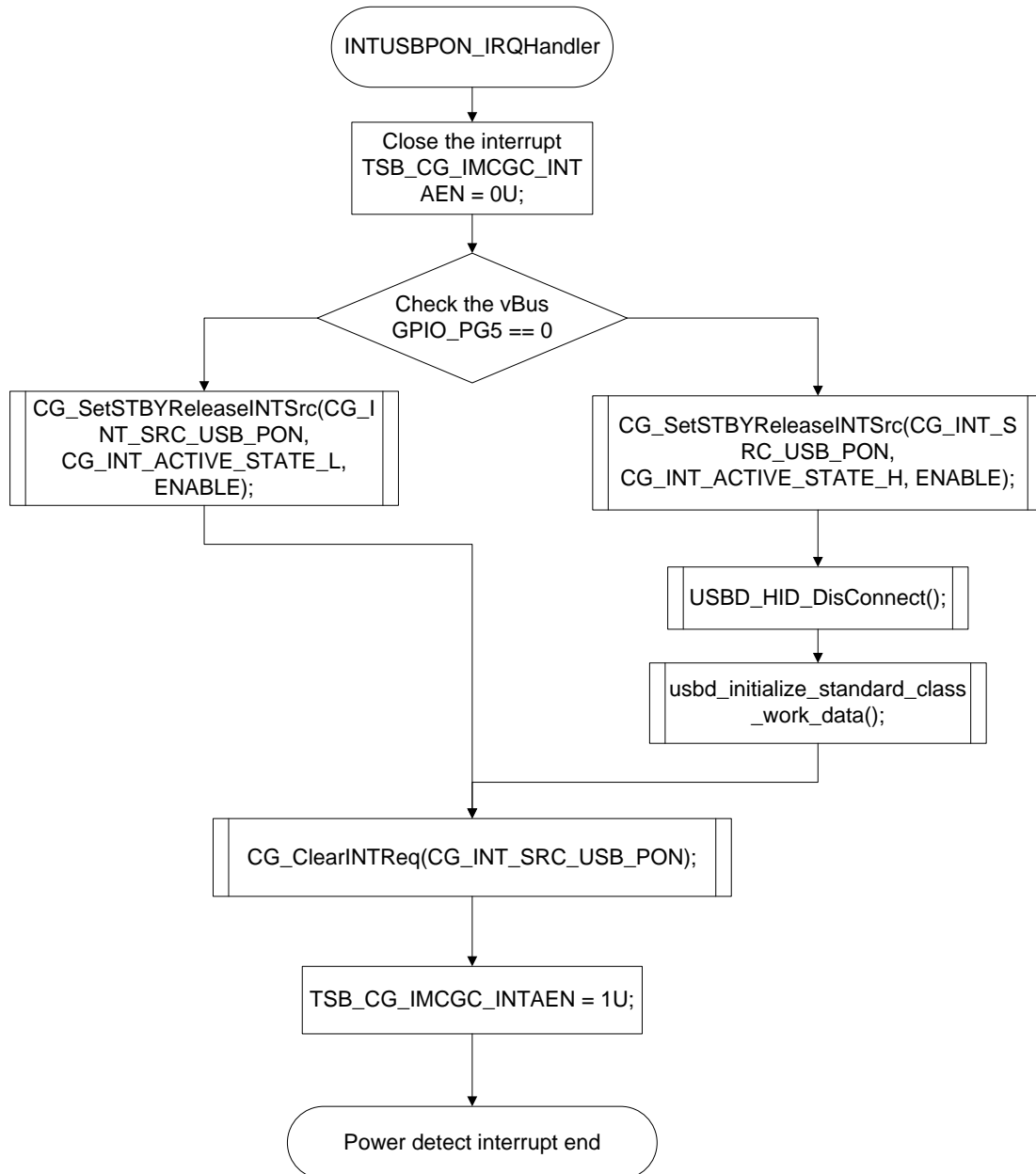
7. 通常のマウス使用への影響を避けるため、サンプルプログラムの動作終了後にデータ送信を停止してください。

7.2 フローチャート

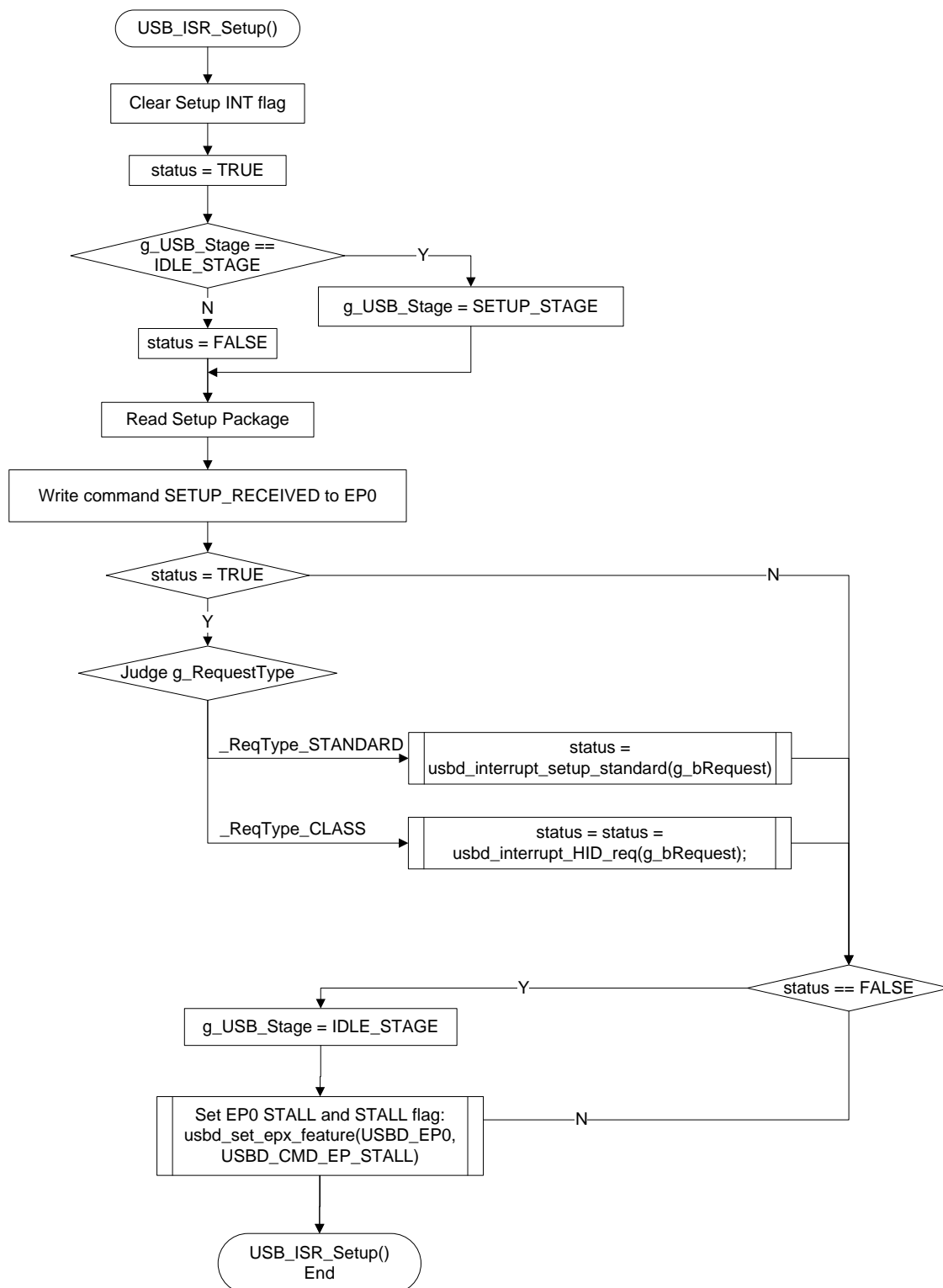
主要USB割り込みルーチンのフローチャートを下記に示します。



VBUS検出割り込みルーチンのフローチャートを下記に示します。



パッケージ設定とその処理工程のフローチャートを下記に示します。



8. コールバック関数

下記構成は、コールバック関数の記録に使用され、システムの初期化に使用します。

```
typedef struct {  
    USBD_HID_ConnStat      pfnConnStat;  
    USBD_HID_Notice        pfnSuspend;  
    USBD_HID_Notice        pfnResume;  
    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;
```

本構成は、3種類の関数に大別されます。

1. ハードウェア応答:

pfnConnStat; pfnSuspend; pfnResume;

2. HIDクラス要求: ホストがデバイスに要求を送信すると本関数がコールされます。

pfnGetReport; pfnSetReport; pfnGetIdle; pfnSetIdle; pfnGetProtocol; pfnSetProtocol;

3. データ転送:

pfnSendData: 割り込み転送における EP3 へのデータ転送。関数“***bool USBD_HID_SendData(uint32_t ulSize, uint8_t *pucData)***”がコールされると、本コールバック関数がコールされます。

pfnCtrlSendData: コントロール転送における EP0 へのデータ転送。関数“***bool USBD_HID_CtrlSendData(uint32_t ulSize, uint8_t *pucData)***”がコールされると、本コールバック関数がコールされます。

pfnCtrlRecvData: コントロール転送におけるEP0からのデータ受信。関数“***bool USBD_HID_CtrlRecvData(uint32_t ulSize, uint8_t *pucData)***”がコールされると、本コールバック関数がコールされます。