

TOSHIBA

アプリケーションノート

USBD ループバック

TMPM365

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。

© 2017 Toshiba Electronic Devices & Storage Corporation

目次

1. 概要.....	1
2. CDC ループバックサンプルプログラム	2
3. 動作環境.....	3
4. 使用手順.....	4
5. 列挙(Enumeration)	6
6. ディスクリプタ	7
6.1 デバイスディスクリプタ.....	7
6.2 ディスクリプタ取得手順	8
6.3 CDC リクエスト	9
7. ソフトウェアファイル構造	11
8. キーコードとフローチャート	13
8.1 キーコード.....	13
8.2 フローチャート	21
9. コールバック関数.....	24

1. 概要

USB CDC ドライバは、CDC 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 CDC ドライバを使用し、簡易な USB CDC ループバック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。
- Universal Serial Bus Class Definitions for Communication Devices
Version 1.1 January 19.1999
→ 主に 5 章と 6 章をご覧ください。

2. CDC ループバックサンプルプログラム

CDC 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行に主に関連しています。

CDC クラスの代表的な例

- 電気通信デバイス(例: アナログモデム、ISDN 端子アダプタ、デジタル電話など)
- ネットワークデバイス(例: ADSL モデム、ケーブルモデムなど)

本 CDC サンプルプログラムは、TMPM365 評価ボード上の COM デバイスとしてシミュレーションされる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 v7.0
- 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

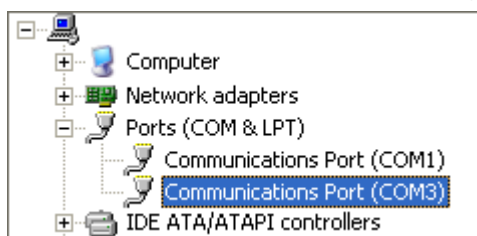
- M365 をサポート可能な IAR Embedded Workbench for ARM 6.401
- PC (OS: Windows XP)
- COM デバッグツール:アクセスポート 1.37 またはその他ポート

4. 使用手順

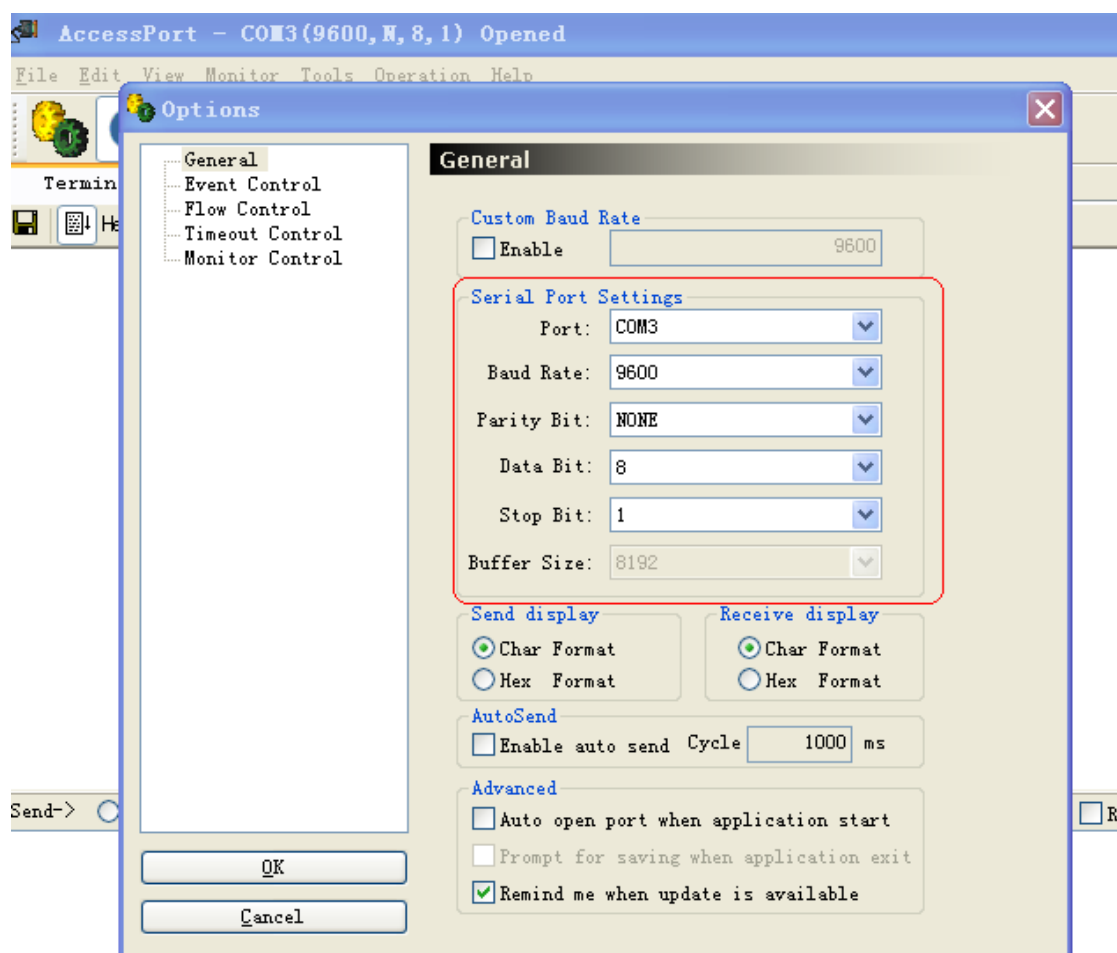
- 1 USB CDC ループバックサンプルプログラムを IAR J-Link-ARM を使用し、M365 ボードへダウンロードする。
- 2 USB で PC と EVB を接続する。
- 3 EVB をリセット
- 4 EVB が USB デバイスとして認識される。



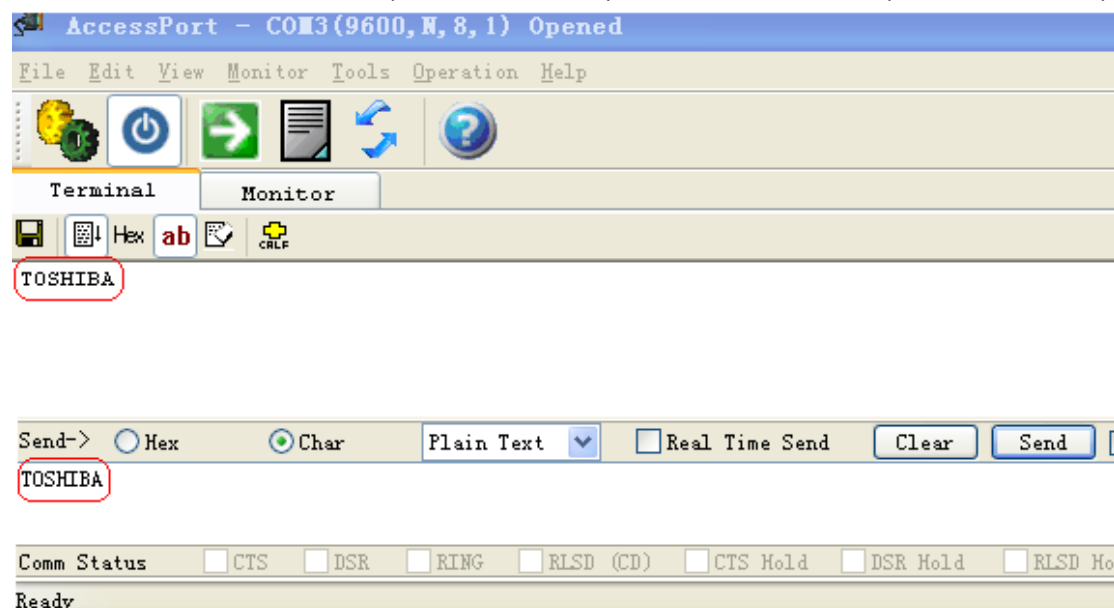
- 5 USB デバイス用ドライバをインストールする。
 - 5.1 “Found New Hardware Wizard”を開く。
 - 5.2 “Install from a list or specific location”を選択する。
 - 5.3 “grusbcdc2.inf”ファイル用のロケーションを指定する。
 - 5.4 “usbser.sys”ファイル用のロケーションを指定する。
Ex: C: ¥WINDOWS¥system32¥drivers
 - 5.5 USBドライバをインストールすると、標準 COM ポートとして認識される。
(例 COM3)
System Properties->Device Manager を開く。下記の例を示す。



- 6 COM debug tool (例: AccessPort.exe)を開き、9600bps、8-bit データ、パリティ無し、1 stop bit と COM3 ポートを下記のように指定する。



7 AccessPort.exe がデータ(例 "TOSHIBA")を COM3 へ送信すると、AccessPort.exe は、同一 (例 "TOSHIBA")を同時に受信する。(下記図を参照。)

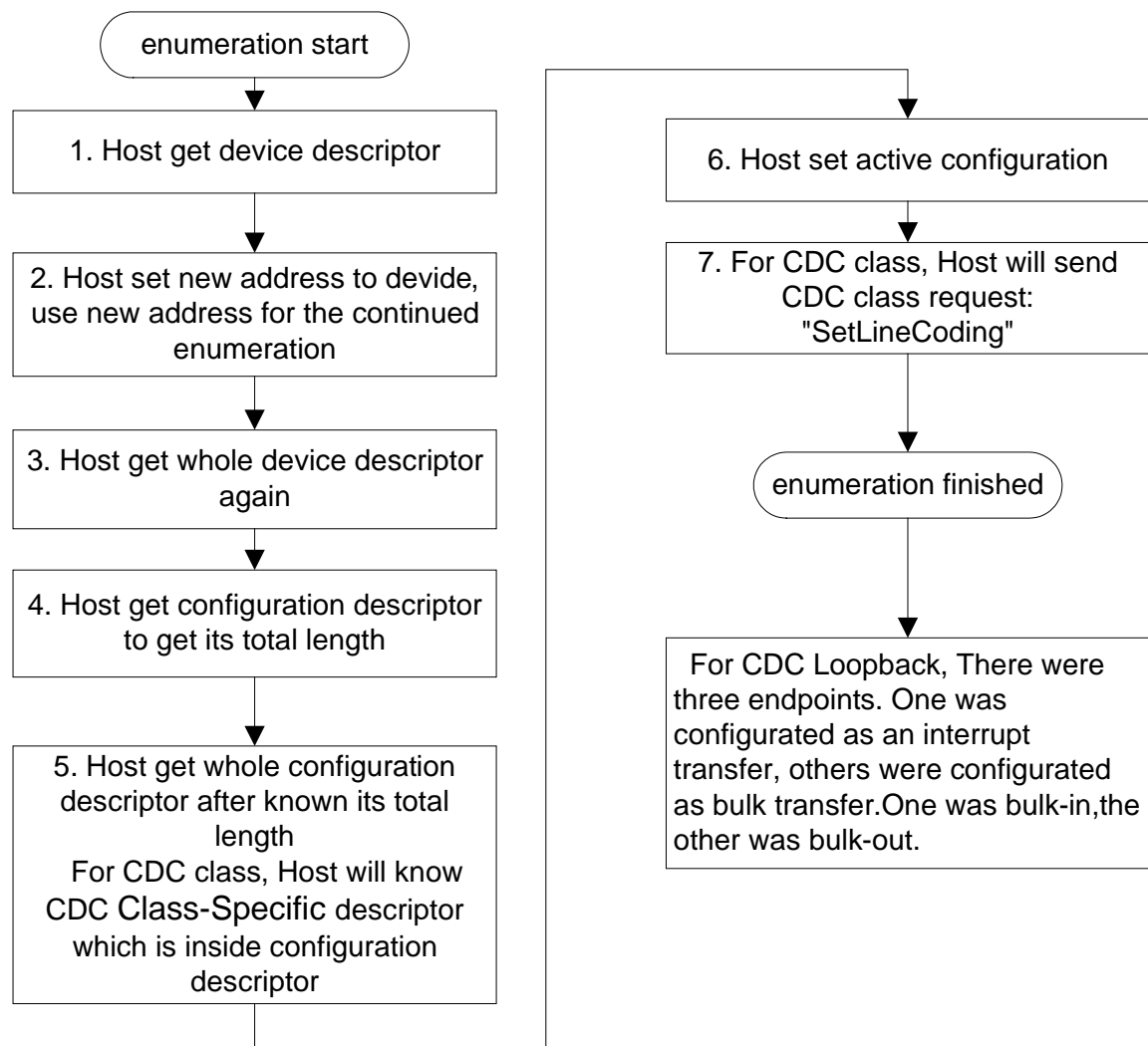


5. 列挙(Enumeration)

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 CDC クラスデバイスを列挙化する簡易化したステップです。



6. ディスクリプタ

6.1 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。

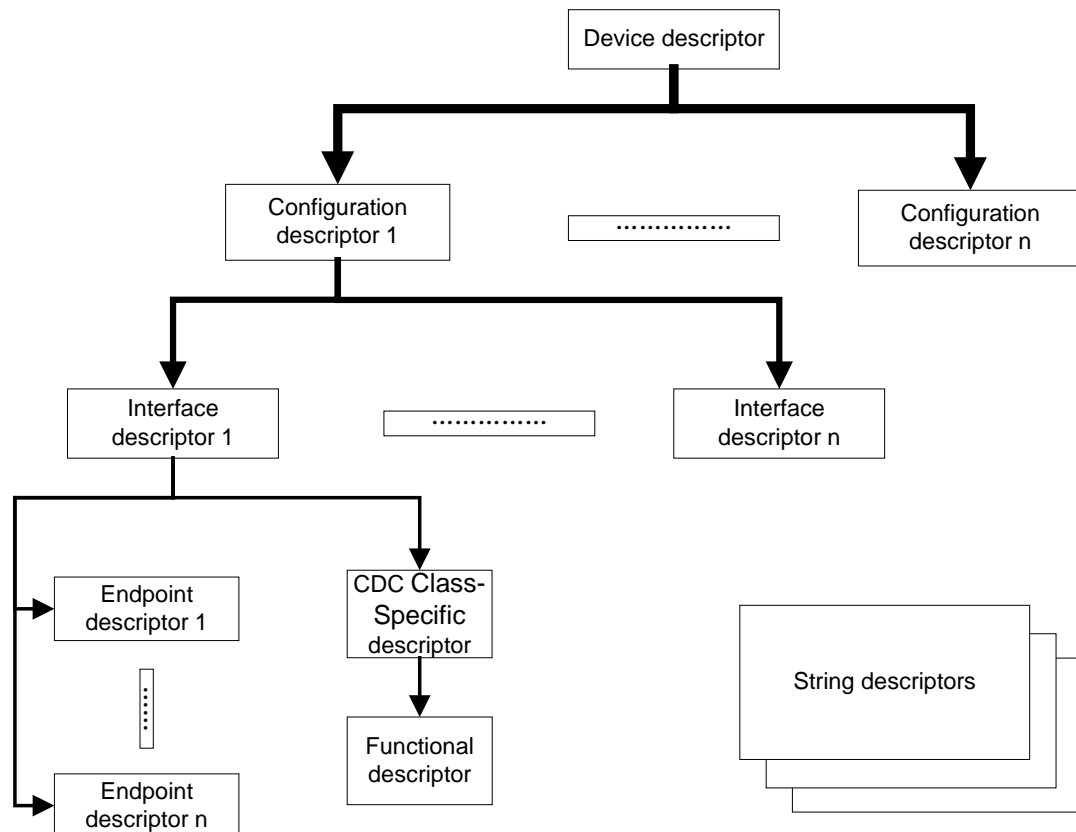
第 2 番目のバイトでは、ディスクリプタのタイプ(デバイス、構成、インタフェース、エンドポイント、クラスなど)が確認できます。

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_cdc.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。

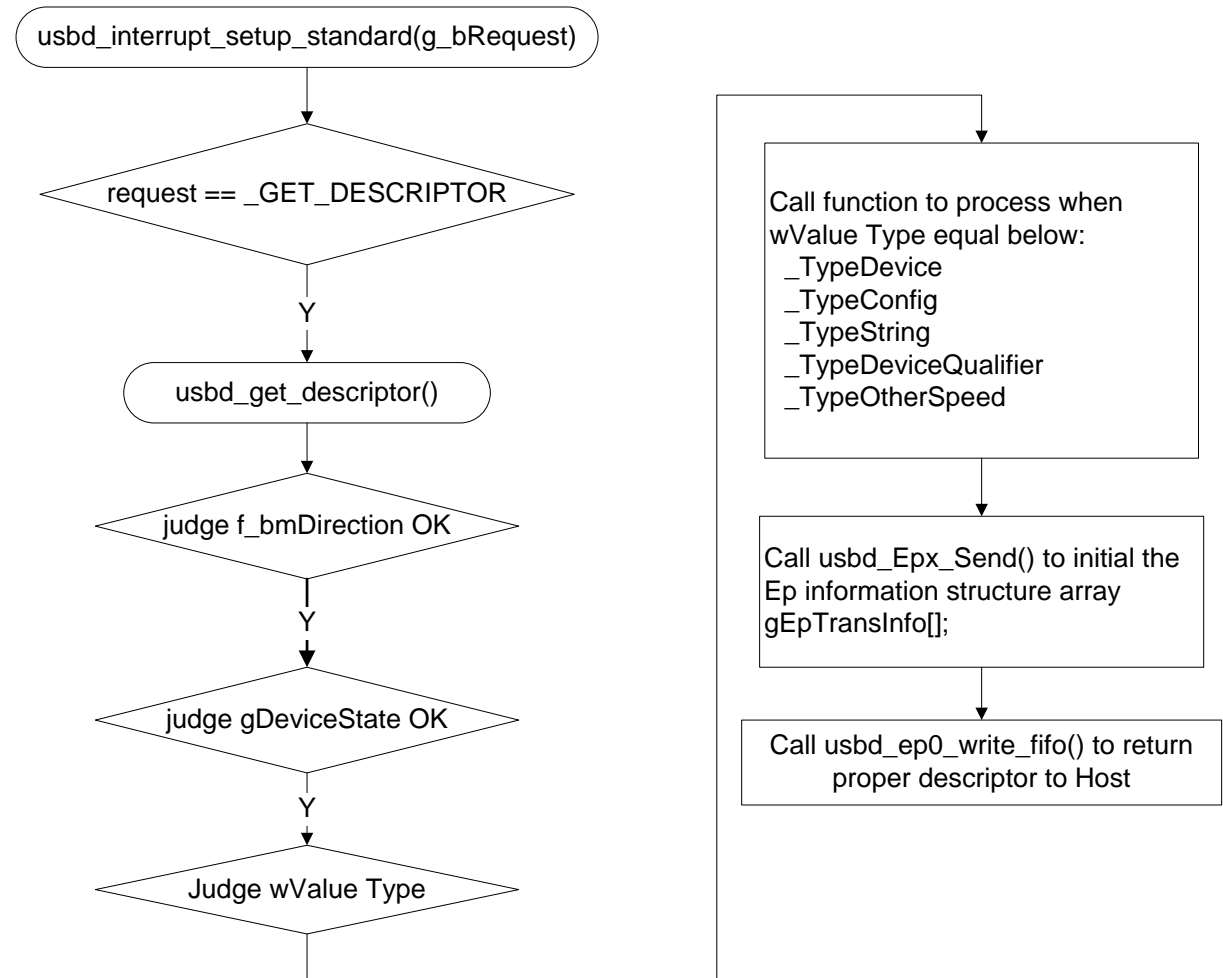
ディスクリプタの階層構造を下図に示します。



6.2 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。



6.3 CDC リクエスト

Send Encapsulated Command カプセル化コマンド送信 (特定クラスリクエスト)
本リクエストは、コミュニケーションクラスインタフェースのサポート制御プロトコルにおけるフォーマットにて発行されます。(本サンプルプログラムでは使用しません。)

Send Encapsulated Commandを発行するには、ホストは下記デフォルトパイプ上にて、デバイス要求を発行しなければなりません。

- **bmRequestType:** ホストからデバイスへ、クラス、インタフェースを要求
- **bRequest:** 0 (00h)
- **wValue:** 0
- **wIndex:** インタフェース番号
- **wLength:** 0

Get Encapslated Response カプセル化応答取得 (特定クラスリクエスト)
本リクエストは、コミュニケーションクラスインタフェースのサポート制御プロトコルにおけるフォーマットにて発行されます。(本デモでは使用しません。)

Get Encapslated Responseを発行するには、ホストは下記デフォルトパイプ上にて、デバイス要求を発行しなければなりません。

- **bmRequestType:** デバイスからホストへ、クラス、インタフェースを要求
- **bRequest:** 1 (01h)
- **wValue:** 0
- **wIndex:** インタフェース番号
- **wLength:** 0

Set Line Coding ラインコーディング設定 (特定クラスリクエスト)
本リクエストにより、アプリケーションにより要求される可能性のある標準的な非同期ラインーキャラクタフォーマットの資産をホストは指定することができます。本要求は、非同期のバイトストリームデータクラスインタフェースとエンドポイントに適用されます。また、ホストからデバイス、デバイスからホストへのデータ転送にも適用されます。

Set Line Coding を発行するには、ホストは下記デフォルトパイプ上にて、デバイス要求を発行しなければなりません。

- **bmRequestType:** ホストからデバイスへクラス、インタフェースを要求
- **bRequest:** 20 (14h)
- **wValue:** 0
- **wIndex:** インタフェース番号
- **wLength:** 構造サイズ(07h)

- **data:**ラインコーディング構造

Get Line Coding ラインコーディング取得(特定クラスリクエスト)

本リクエストにより、ホストは現在のラインコーディング構造を知ることが可能となります。

Get Line Codingを発行するには、ホストは下記デフォルトパイプ上にて、デバイスリクエストを発行しなければなりません。

- **bmRequestType:** デバイスからホストへクラス、インタフェースを要求
- **bRequest:** 21 (15h)
- **wValue:** 0
- **wIndex:** インタフェース番号
- **wLength:** 構造サイズ(07h)
- **data:**ラインコーディング構造

SetControlLineState (特定クラスリクエスト)

本リクエストは、RS-232/V.24型制御信号を生じさせます。

SetControlLineState を発行するには、ホストは下記デフォルトパイプ上にて、デバイス要求を発行しなければなりません。

- **bmRequestType:** ホストからデバイスへのクラス、インタフェースを要求
- **bRequest:** 22 (16h)
- **wValue:**制御信号ビットマップ
- **wIndex:** インタフェース番号
- **wLength:** 構造サイズ(07h)

7. 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。

```
/---CDC_Loopback
+---APP
|   |   cdc_loopback.c
|   |
|   +---cdc_inc
|   |       usbd_descriptor_cdc.h
|   |       usbd_cdc.h
|   |
|   \---cdc_src
|           usbd_descriptor_cdc.c
|           usbd_cdc.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
```

8. 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。

8.1 キーコード

1. ラインコーディング構造を初期化します。

```
I_aucSampleLineCoding[0] = 0x80U;  
I_aucSampleLineCoding[1] = 0x25U;  
I_aucSampleLineCoding[2] = 0x00U;  
I_aucSampleLineCoding[3] = 0x00U;  
I_aucSampleLineCoding[4] = 0x00U;  
I_aucSampleLineCoding[5] = 0x00U;  
I_aucSampleLineCoding[6] = 0x08U;
```

I_aucSampleLineCoding[0]~[3]	データデータ転送速度 例 0x00002580 は 9600bps です。
I_aucSampleLineCoding[4]	ストップビット。 例 0x00 は 1 ストップビットです。
I_aucSampleLineCoding[5]	パリティ。 例 0x00 はパリティ無しです。
I_aucSampleLineCoding[6]	データビット長。 例 0x08 は 8 ビットです。

2. ライン状態をクリアします。

```
I_usSampleControlLineState = 0x0000U;
```

3. CDC 初期パラメータを設定します。

```
gSample_CDC_CB.pfnConnStat          = _Sample_ConnStat;  
gSample_CDC_CB.pfnSendEncapsulatedCmd =  
_Sample_SendEncapsulatedCommad;  
gSample_CDC_CB.pfnGetEncapsulatedRes  =  
_Sample_GetEncapsulatedResponse;  
gSample_CDC_CB.pfnSetCommFeature      = NULL;  
gSample_CDC_CB.pfnGetCommFeature      = NULL;  
gSample_CDC_CB.pfnClearCommFeature    = NULL;  
gSample_CDC_CB.pfnSetLineCoding       = _Sample_SetLineCoding;  
gSample_CDC_CB.pfnGetLineCoding       = _Sample_GetLineCoding;  
gSample_CDC_CB.pfnSetControlLineState = _Sample_SetControlLineState;  
gSample_CDC_CB.pfnSendBreak           = NULL;
```



```
gSample_CDC_CB.pfnNetworkConnection    = NULL;
gSample_CDC_CB.pfnResponseAvailable    = _Sample_ResponseAvailable;
gSample_CDC_CB.pfnSerialState          = NULL;
gSample_CDC_CB.pfnSendData             = _Sample_SendData;
gSample_CDC_CB.pfnRecvData            = _Sample_RecvData;
```

_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 USBD_CDC_RecvData ()をコールします。

```
static void _Sample_ConnStat(bool bStat)
{
    if (bStat == ture) {
        /* Connect */
        /* Request for data receive */
        gbStatDC = USBD_CDC_RecvData(SAMPLE_BUF_SIZE, l_aucSampleBuf);
        /* No care status in sample */
    } else {
        /* Disconnect */
        /* Do nothing in sample */
    }

    return;
}
```

_Sample_GetLineCoding は、コールバック関数です。デバイスから Line Coding 構造体を得るために使用されます。ホストがデバイスに Get Line Coding 要求を送信した場合、本関数がコールされます。

```
static void _Sample_GetLineCoding(uint16_t usLength)
{
    /* GET_LINE_CODING received */

    /* Reply line coding */
    gbStatDC = USBD_CDC_Send_GetLineCoding(usLength,
l_aucSampleLineCoding);
    /* No care status in sample */

    return;
}
```

_Sample_SetLineCoding は、コールバック関数です。Line Coding 構造体デバイスに設定する場合に使用されます。ホストがデバイスに Set Line Coding 要求を送信した場合、本関数がコールされます。

```
static void _Sample_SetLineCoding(uint32_t ulSize, uint8_t *pucData)
```

```
{
    /* SET_LINE_CODING received */

    /* Keep line coding */
    memcpy(l_aucSampleLineCoding, pucData, ulSize);

    return;
}
```

`_Sample_SetControlLineState` は、コールバック関数です。
`l_usSample_SetControlLineState` を設定する場合に使用されます。ホストがデバイスに `SetControlLineState` 要求を送信した場合、本関数がコールされます。

```
static void _Sample_SetControlLineState(uint16_t usValue)
{
    /* SET_CONTROL_LINE_STATE received */

    /* Keep line state */
    l_usSampleControlLineState = usValue;

    return;
}
```

`_Sample_ResponseAvailable` は、コールバック関数です。本サンプルプログラム上では何もしません。

```
static void _Sample_ResponseAvailable(void)
{
    /* RESPONSE_AVAILABLE received */
    /* Do nothing in sample */

    return;
}
```

`_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。PC からのデータ受信に関数 `USBD_CDC_RecvData` がコールされます。

```
static void _Sample_SendData(uint32_t ulSize, uint8_t *pucData, uint16_t usStat)
{
    if (usStat == EPSTAT_COMPLETE) {
        /* Request for data receive */
        gbStatDC = USBD_CDC_RecvData(SAMPLE_BUF_SIZE, l_aucSampleBuf);
        /* No care status in sample */
    }
}
```

```
    return;  
}
```

_Sample_RecvData は、コールバック関数です。PC からのデータ受信に使用されます。データ転送が完了すると、本関数がコールされます。PC へのデータ送信に関数 USBDCDC_SendData がコールされます。

```
static void _Sample_RecvData(uint32_t ulSize, uint8_t *pucData, uint16_t usStat)  
{  
    if (usStat == EPSTAT_COMPLETE) {  
        /* Request for data send */  
        gbStatDC = USBDCDC_SendData(ulSize, l_aucSampleBuf);  
        /* No care status in sample */  
    }  
  
    return;  
}
```

4. CDCドライバを初期化します。

```
void usbd_cdc_init(CDC_CB_t *p_cdc_cb)  
{  
    /* Set of callback functions */  
    gCDC_CB.pfnConnStat = p_cdc_cb->pfnConnStat;  
    gCDC_CB.pfnSendEncapsulatedCmd = p_cdc_cb->pfnSendEncapsulatedCmd;  
    gCDC_CB.pfnGetEncapsulatedRes = p_cdc_cb->pfnGetEncapsulatedRes;  
    gCDC_CB.pfnSetCommFeature = p_cdc_cb->pfnSetCommFeature;  
    gCDC_CB.pfnGetCommFeature = p_cdc_cb->pfnGetCommFeature;  
    gCDC_CB.pfnClearCommFeature = p_cdc_cb->pfnClearCommFeature;  
    gCDC_CB.pfnSetLineCoding = p_cdc_cb->pfnSetLineCoding;  
    gCDC_CB.pfnGetLineCoding = p_cdc_cb->pfnGetLineCoding;  
    gCDC_CB.pfnSetControlLineState = p_cdc_cb->pfnSetControlLineState;  
    gCDC_CB.pfnSendBreak = p_cdc_cb->pfnSendBreak;  
    gCDC_CB.pfnNetworkConnection = p_cdc_cb->pfnNetworkConnection;  
    gCDC_CB.pfnResponseAvailable = p_cdc_cb->pfnResponseAvailable;  
    gCDC_CB.pfnSerialState = p_cdc_cb->pfnSerialState;  
    gCDC_CB.pfnSendData = p_cdc_cb->pfnSendData;  
    gCDC_CB.pfnRecvData = p_cdc_cb->pfnRecvData;  
  
    /* Initialize buffers */  
    memset(l_aucReqDataBuf, 0x00U, sizeof(l_aucReqDataBuf));  
    memset(&l_tNetworkConnection, 0x00U, sizeof(l_tNetworkConnection));  
    memset(&l_tResponseAvailable, 0x00U, sizeof(l_tResponseAvailable));  
    memset(&l_tSerialState, 0x00U, sizeof(l_tSerialState));  
    l_wEncapsulatedResponseSize = 0U;
```

```
/* Initialization of an internal variable */
l_UsbStat = false;

return;
}
```

5. 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, CDC */

    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;
}
```

6. USB D 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
- USB D 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
- USB D 割り込みの許可
- エンドポイント 0 リセット

```
void Config_USB(void)
{
    USB D_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 PA0 control the pull up of D+, it must be set output '1' */
    GPIO_SetOutput(GPIO_PA, GPIO_BIT_0);
    GPIO_WriteDataBit(GPIO_PA, GPIO_BIT_0, GPIO_BIT_VALUE_1);

    USB D_SetINTMask(USB D_INT_USB_RESET_END, ENABLE);
    USB D_SetINTMask(USB D_INT_USB_RESET, ENABLE);

    /*  UDPWCTL Power Reset and */
    /*  PHY Reset & Suspend: 1ms */
    pwr = USB D_GetPowerCtrlStatus();
    pwr.Bit.PW_Resetb = 0U;
    pwr.Bit.PHY_Resetb = 0U;
    pwr.Bit.PHY_Suspend = 1U;
    USB D_SetPowerCtrl(pwr);
    usbd_wait_1ms();
}
```

```
/* PHY Reset off : 1ms */
pwr = USBD_GetPowerCtrlStatus();
pwr.Bit.PHY_Resetb = 1U;
pwr.Bit.PHY_Suspend = 1U;
USB_D_SetPowerCtrl(pwr);
usb_wait_1ms();

/* PHY Suspend off : 1ms */
pwr = USBD_GetPowerCtrlStatus();
pwr.Bit.PHY_Suspend = 0U;
USB_D_SetPowerCtrl(pwr);
usb_wait_1ms();
usb_wait_1ms();

/* UDPWCTL Power Reset Off: 1ms */
pwr = USBD_GetPowerCtrlStatus();
pwr.Bit.PW_Resetb = 1U;
USB_D_SetPowerCtrl(pwr);
usb_wait_1ms();
usb_wait_1ms();
pwr = USBD_GetPowerCtrlStatus();

/* clear pending UDC2 interrupt and disable INT for SOF and NAK */
USB_WriteUDC2Reg(UDC2_INT, 0x90FFU);

USB_SetEPCMD(USB_EP0, USB_CMD_ALL_EP_INVALID);
USB_SetEPCMD(USB_EP0, USB_CMD_USB_READY);

NVIC_EnableIRQ(INTUSB_IRQn);

//VBUS
NVIC_EnableIRQ(INTUSBPON_IRQn);

return;
}
```

7. 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 COM(下記参照)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、*usb_hw_interrupt.c* 内の **INTUSB_IRQHandler()** 関数と **INTUSBPON_IRQHandle()** 関数、*usb_device_request.c* 内の **usb_interrupt_setup_standard()** 関数と *usb_cdc.c* 内の

`usb_d_interrupt_CDC_req()`関数を参照してください。

8. 列挙化終了後、ユーザは COM として扱う USB ヘデータ送信する COM デバッグツールを使用することができます。COM デバッグツールは COM から同様のデータを受信します。これでデータのループバックは完了します。

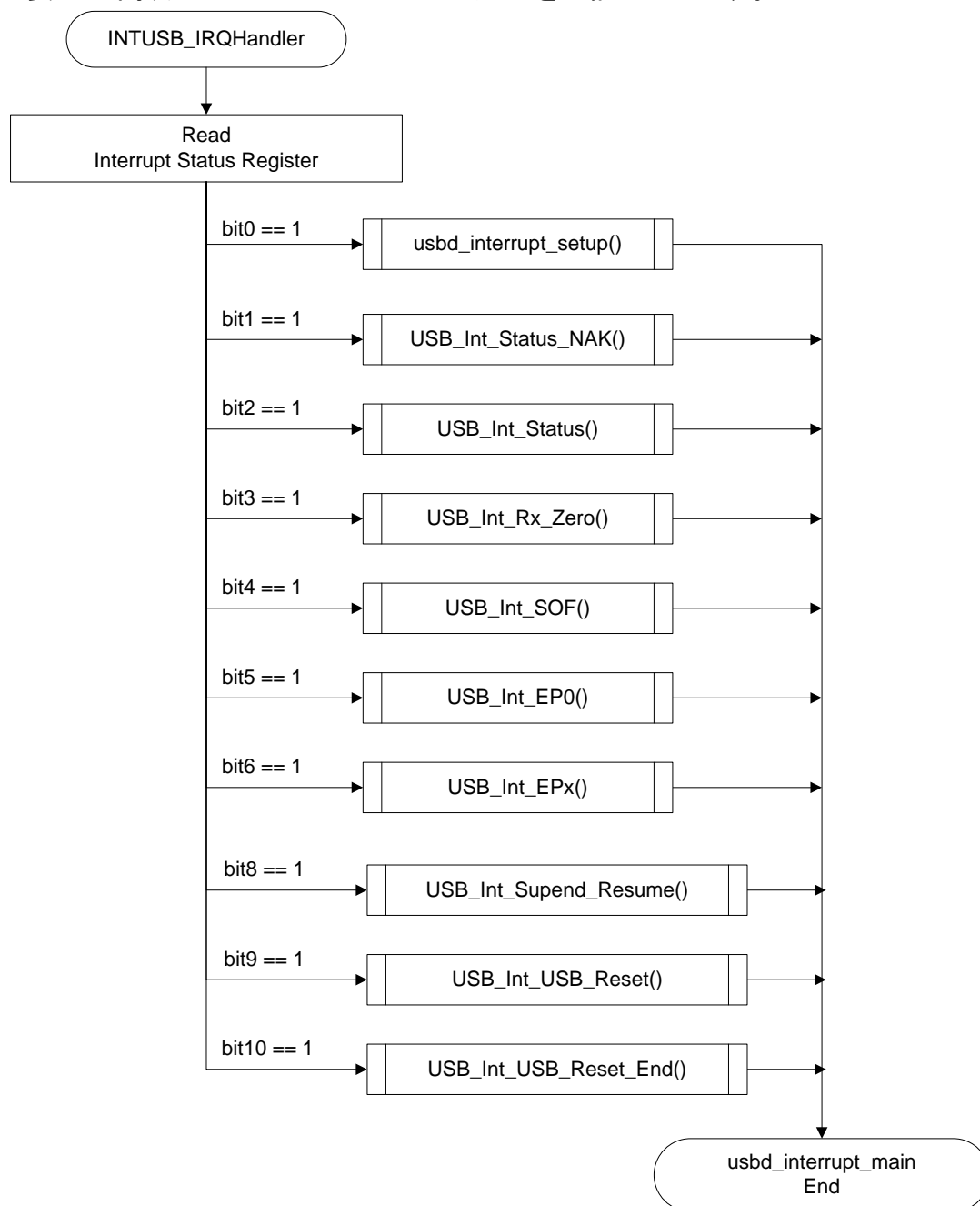
内容の詳細は、コールバック関数 `_Sample_SendData()` と `_Sample_RecvData()` のコール条件を参照してください。

補足:

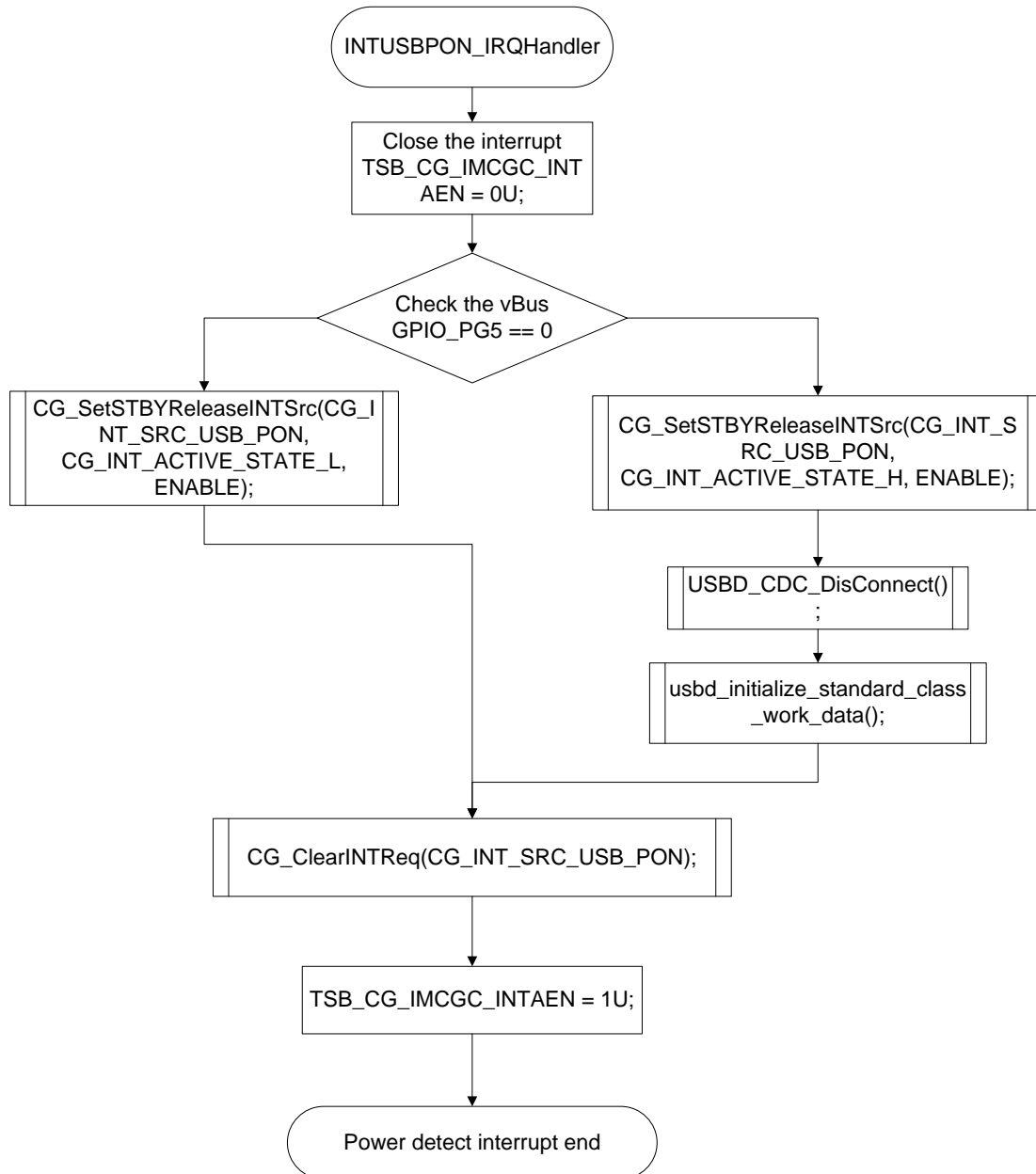
ファイル `grusbcdc2.inf` は、PC が USB ドライバを COM ポートとして認識する前に設定してください。

8.2 フローチャート

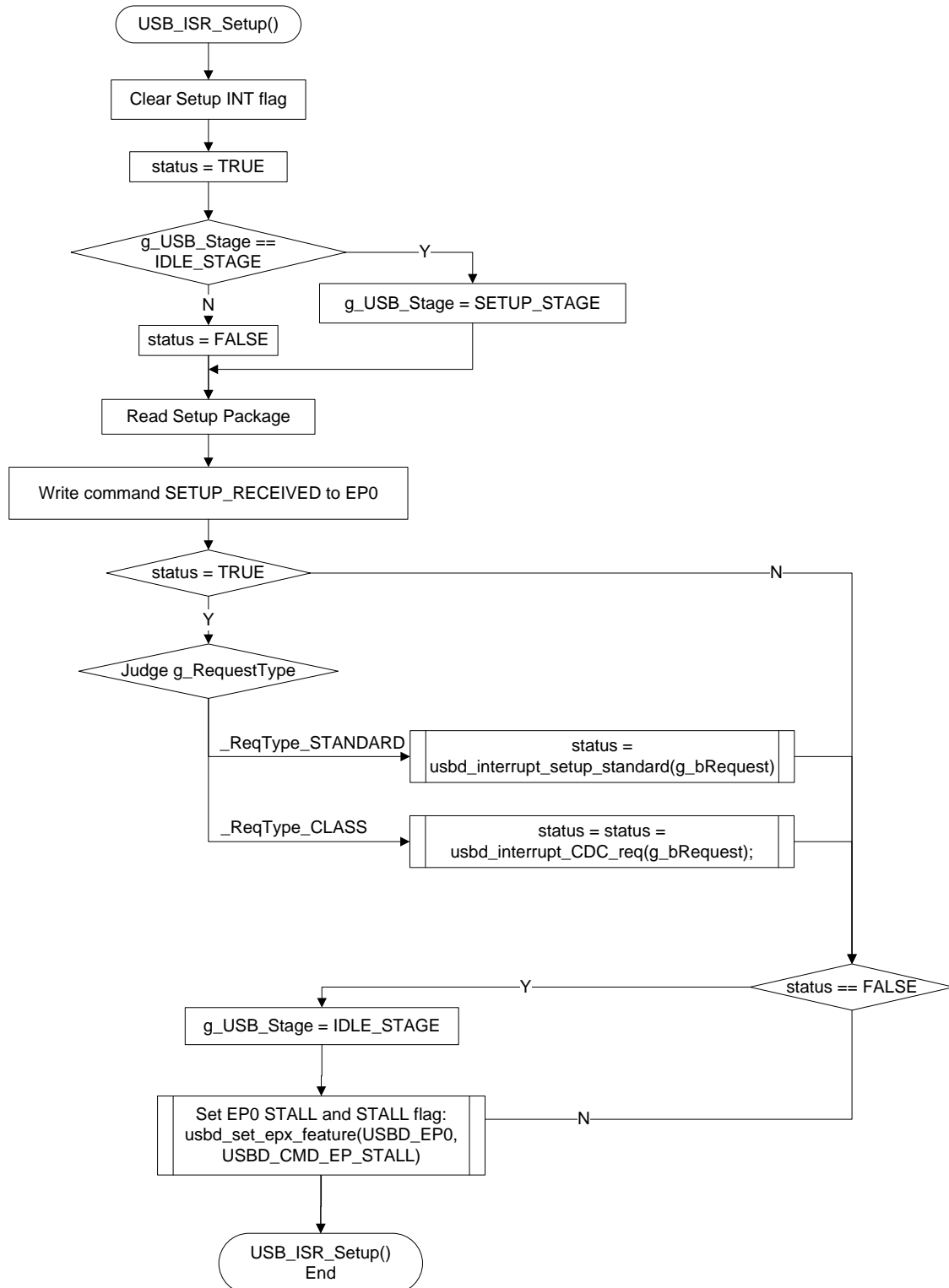
主要USB割り込みルーチンのフローチャートを下記に示します。



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。



9. 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 {  
    CDC_CB_ConnStat                pfnConnStat;  
    CDC_CB_SendEncapsulatedCmd     pfnSendEncapsulatedCmd;  
    CDC_CB_Get                     pfnGetEncapsulatedRes;  
    CDC_CB_Set                     pfnSetCommFeature;  
    CDC_CB_Get                     pfnGetCommFeature;  
    CDC_CB_ClearCommFeature        pfnClearCommFeature;  
    CDC_CB_Set                     pfnSetLineCoding;  
    CDC_CB_Get                     pfnGetLineCoding;  
    CDC_CB_SetControlLineState     pfnSetControlLineState;  
    CDC_CB_SendBreak               pfnSendBreak;  
    CDC_CB_Notification             pfnNetworkConnection;  
    CDC_CB_Notification             pfnResponseAvailable;  
    CDC_CB_Notification             pfnSerialState;  
    CDC_CB_TrnsData                pfnSendData;  
    CDC_CB_TrnsData                pfnReciveData;  
} CDC_CB_t;
```

本構成は、4種類の関数に大別されます。

1. ハードウェア応答:

pfnConnStat;接続/接続解除用コールバック

2. CDC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。

pfnSendEncapsulatedCmd; ***pfnGetEncapsulatedRes***;
pfnSetCommFeature; ***pfnGetCommFeature***; ***pfnClearCommFeature***;
pfnSetLineCoding; ***pfnGetLineCoding***; ***pfnSetControlLineState***;
pfnSendBreak;

3. 通信インタフェースクラス通知:

pfnNetworkConnection; ***pfnResponseAvailable***; ***pfnSerialState***;

4. データ転送

pfnSendData: バルク転送における EP2 へのデータ転送。関数“***bool USB_D_CDC_RecvData (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。

pfnReciveData: バルク転送における EP1 へのデータ転送。関数“***bool***

*USBD_CDC_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実行されます。