TOSHIBA

アプリケーションノート

USBD RAM Disk

TMPM366

第一版

2017 年 9 月

東芝デバイス＆ストレージ株式会社

CMDR-M366UDM-01J

本製品取り扱い上のお願い

- ソフトウエ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

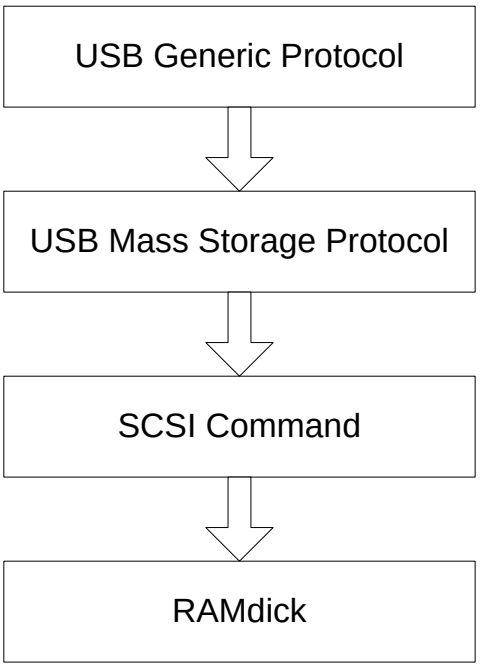
1. 概要

USB マスストレージクラス(MSC)ドライバは、MSC の再利用を容易にするドライバです。

本資料は、東芝 TMPM366FDFG 評価ボードと USBD ドライバを使用し、容易に USB MSC RAM Disk 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。

下図は USB マスストレージフレームワークの基本構造です。



2. MSC RAM Disk サンプルプログラム

USB マスストレージクラス(**USB MSC** あるいは **UMS**)は、ユニバーサルシリアルバス(USB)デバイスとホストコンピュータデバイスを接続し、その間のファイル転送を可能にするプロトコルです。

USB マスストレージデバイスクラスは、ユニバーサルシリアルバス上で実行される USB Implements Forum(USB-IF)により定義された通信プロトコルセットです。標準形は各種ストレージデバイスとのインタフェースを提供しています。

本標準のインタフェースを経由して接続するデバイスには以下のものがあります。

- 外付け磁気ハードドライブ
- 外付け光学ドライブ(CD および DVD R/W ドライブ)
- ポータブルフラッシュメモリデバイス
- …..

本標準によりサポートされるデバイスを **MSC** (マスストレージクラス)デバイスと呼びます。MSC が正式な略語である一方 **UMS** (ユニバーサルマスストレージ)がオンライン上での専門用語として一般的となっています。

EVB を USB インタフェース経由で PC に接続し、電源を投入すると、24k RAM Disk が認識され、PC によりフォーマット、ファイルの保存が可能となります。

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

- M366FDFG 評価ボード (非売品)
  (以降、評価ボードを“EVB”と省略)
- IAR J-Link-ARM v8.0
- USB の D+ピンを EVB の 64 番ピンに接続
- USB の D- ピンを EVB の 63 番ピンに接続
- USB の VBUS ピンを EVB の 20 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

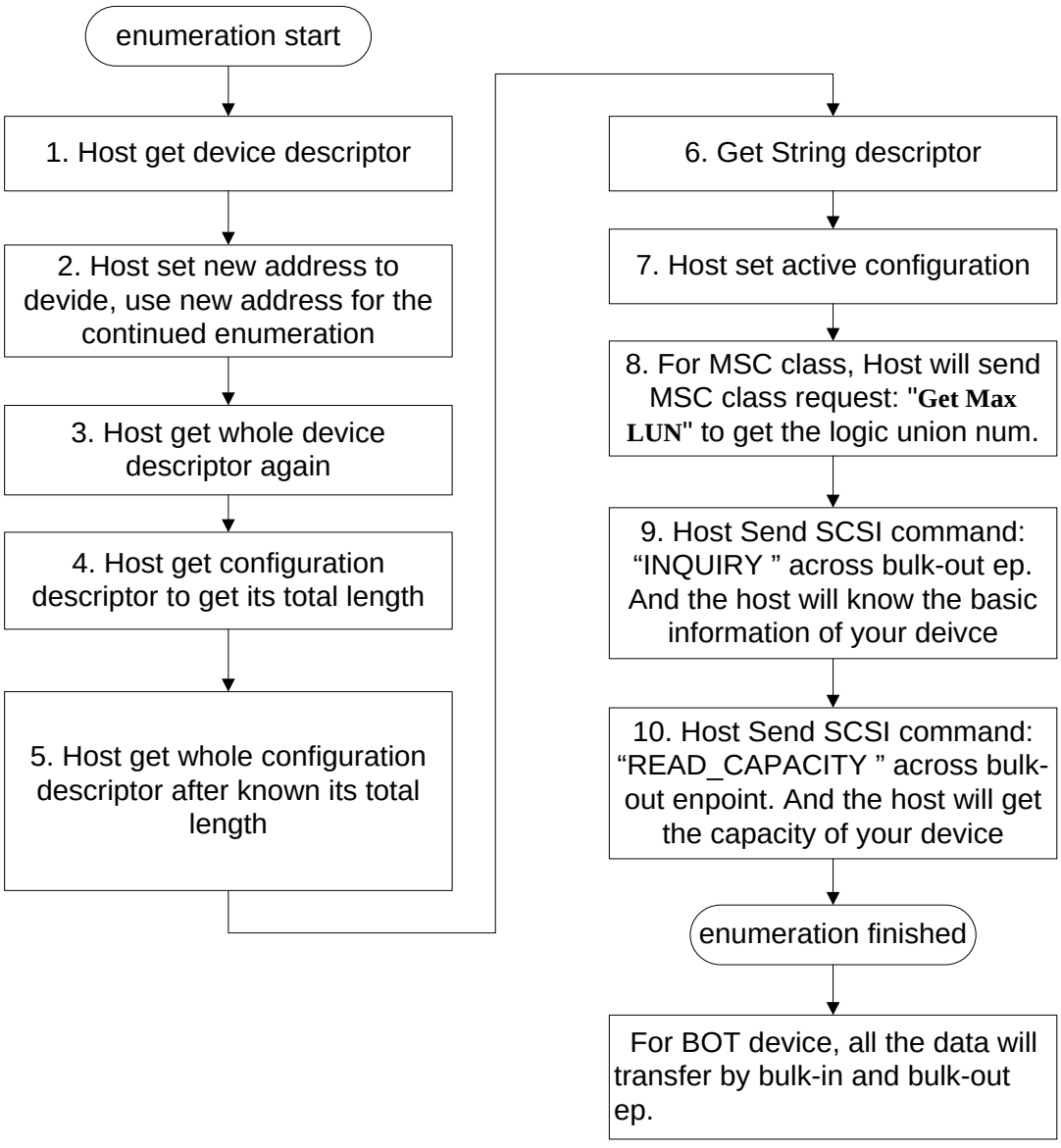
- M366FDFG をサポート可能な IAR Embedded Workbench for ARM 6.401
- PC (OS: Windows XP)

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 MSC 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

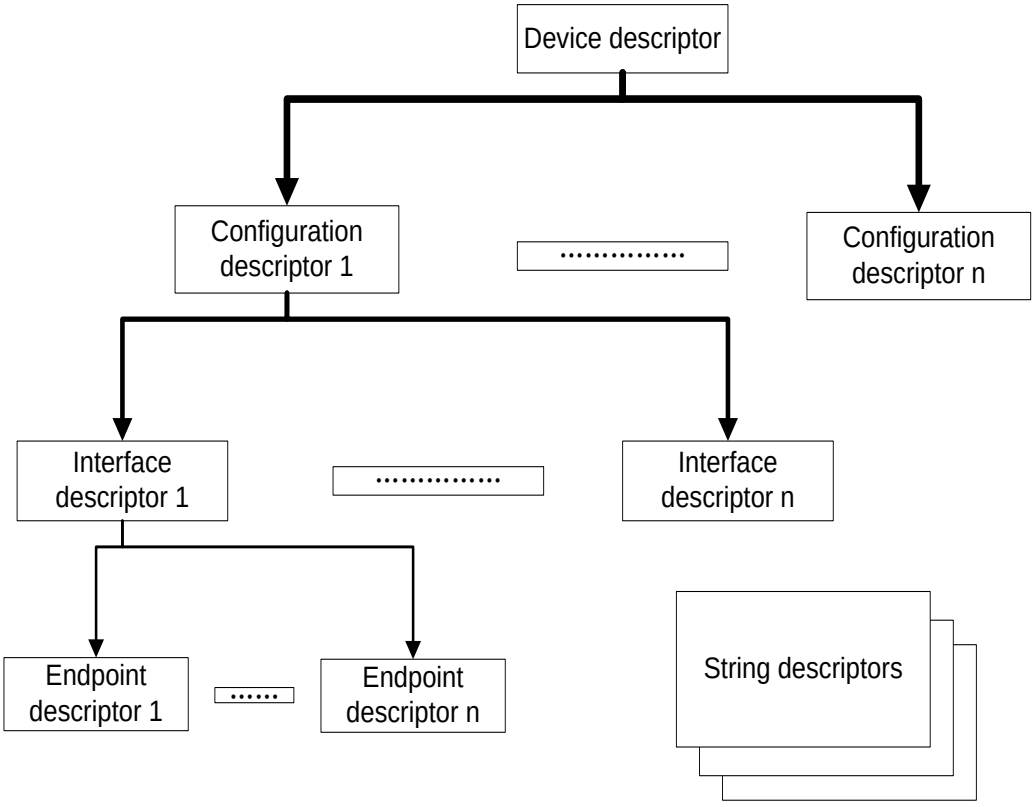
5.1 デバイスディスクリプタ

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_msc.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。



Interface Descriptors
本デバイスは、MSC の Bulk-Only 方式です。

本サンプルプログラムでは、***usbd_descriptor_msc.c*** ファイル内の
“**gInterfaceDescriptor1_0[]**” を取得します。

```
const InterfaceDescriptor_t gInterfaceDescriptor1_0 = {
    CbLength_Interface,                 /* bLength                 */
    CbDescripterType_Interface,         /* bDescriptorType         */
    0x00U,                              /* bInterfaceNumber        */
    0x00U,                              /* bAlternateSetting       */
    0x02U,                              /* bNumEndpoints           */
    0x08U,                              /* bInterfaceClass         */
    0x06U,                              /* bInterfaceSubClass      */
    0x50U,                              /* bInterfaceProtocol      */
    0x00U                               /* iInterface              */
};
```

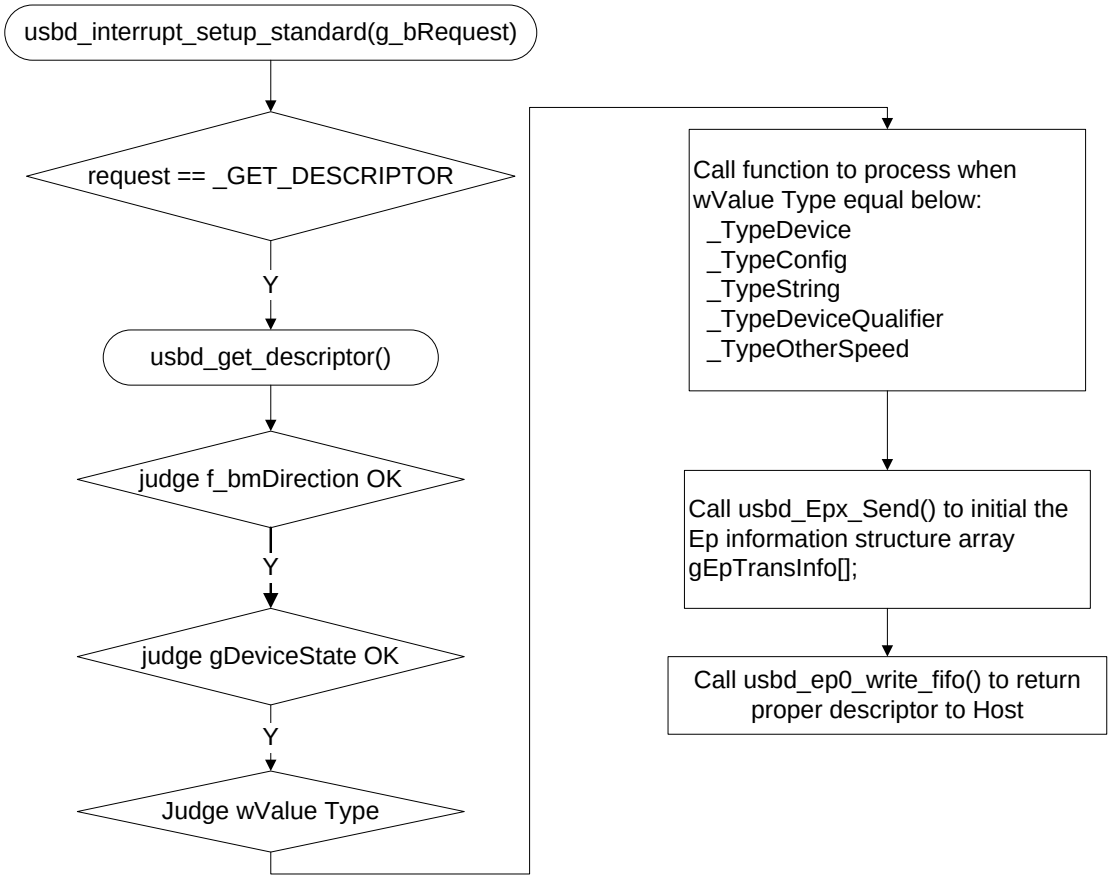
下記項目を対象にして説明します。これら 3 件の項目により、ホストはデバイスをマスストレージデバイスとして扱い、通信方法を知ることができます。

bInterfaceClass = 0x08 = Mass Storage
bInterfaceSubClass = 0x06 = SCSI Transparent
bInterfaceProtocol = 0x50 = Bulk Only Transport

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。



5.3 MSC リクエスト

Bulk-Only Mass Storage Reset *(特定クラスリクエスト)*
本リクエストはマスストレージデバイスをリセットするために使用され、インタフェースと関連付けされています。

本クラスの特殊リクエストは、ホストから次の CBW 用デバイスを準備しなければなりません。

ホストは、本リクエストを標準のパイプを経由してデバイスに送信します。デバイスは、Bulk-Onlyマスストレージリセットの場合にも、バルクデータトグルビットの値とエンドポイント停止条件を保持しなければなりません。

本デバイスは、Bulk-Only マスストレージリセットが完了するまで、デバイスリクエストの状態を NAK にしておく必要があります。

Bulk-Onlyマスストレージリセットを発行するためには、ホストは下記標準パイプ上にデバイスリクエストを発行します。

- **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:** 255 (FFh)
- **wValue:** 0
- **wIndex:** インタフェース番号
- **wLength:** 0

Get Max LUN *(特定クラスリクエスト)*
Get Max LUN デバイスリクエストは、デバイスによりサポートされるロジックユニット数を指定するために使用されます。デバイス上のロジックユニット数は、LUN0 から最大 LUN15（Fh）まで連続的に番号付けされます。

Get Max LUN デバイスリクエストを発行するためには、ホストは下記標準パイプ上に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:** 254 (FEh)
- **wValue:** 0
- **wIndex:** インタフェース番号
- **wLength:** 1

6. Bulk-Only 転送

本サンプルプログラムは、bulk–only 転送プロトコルを使用します。 列挙化後、全データは bulk-in と bulk-out エンドポイントにより転送されます。

CBW のコマンドブロックには、デバイスにより実行される **SCSI** コマンドが含まれます。デバイスはコマンドブロックを解釈し、実行します。最後にデバイスは CSW をホストに返却します。

図 1 のコマンド/データ/ステータスフローは、コマンド転送、データイン、データアウト、ステータス転送のフローを表します。

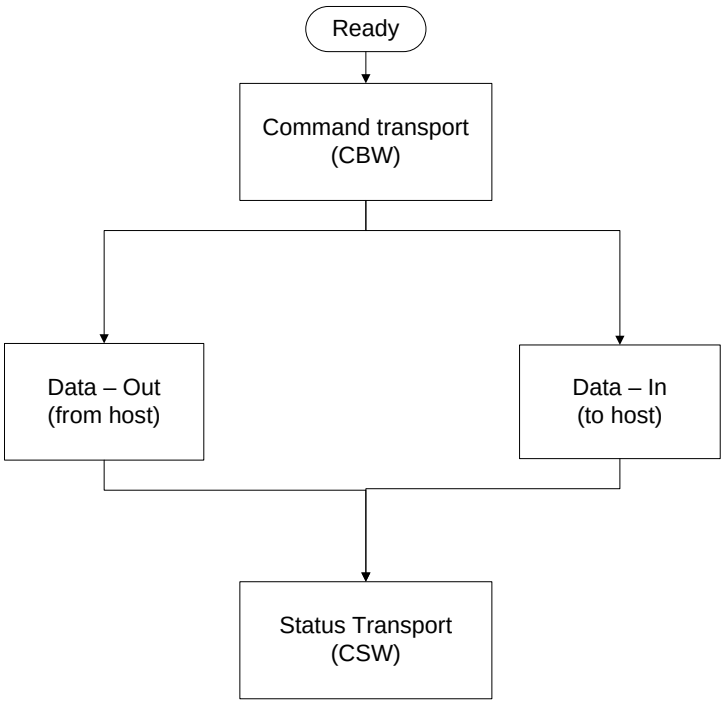


図 1 – コマンド/データ/ステータス

詳細は、usbd_scsi.c の**_USBD_MSC_SCSI_Command_Analysis(uint8_t *pucData)** 関数および8.2(4)章のフローチャートをご覧ください。

本関数は、CBW を解釈し、関数に対応する関連コマンドへ分岐させます。

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。

```
/---MSC_RAMDisk
|     +---APP
|     |     |     msc_ramdisk.c
|     |     |
|     |     +---msc_inc
|     |     |           usbd_descriptor_msc.h
|     |     |           usbd_msc.h
|     |     |           usbd_scsi.h
|     |     |
|     |     \---msc_src
|     |                 usbd_descriptor_msc.c
|     |                 usbd_msc.c
|     |                 usbd_scsi.c
|     +---IAR
|     |           msc_ramdisk.ewd
|     |           msc_ramdisk.ewp
|     |           msc_ramdisk.eww
|     |
|     \---KEIL
|                 MSC_RAMDisk.uvopt
|                 MSC_RAMDisk.uvproj
\---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
```

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. データ受信および送信用の内蔵変数と内蔵バッファを初期化します。

```
/* Initialization of an internal variable */
l_ucMediaStat          = false;
l_ulDataTransferLen = 0U;
l_ulLBA                = 0U;
l_usBlockNum           = 0U;
l_ulDataLenCount       = 0U;
l_usBlockCount         = 0U;
l_bRRState             = true;

/* Initialize buffers */
memset(l_aucSendBuff, 0x00U, BUFFER_SIZE);
memset(l_aucRecvBuff, 0x00U, BUFFER_SIZE);
```

2. MSC(SCSI)ドライバを初期化します。**"gSample_MSC_CB"**構造体は、コールバック関数のアドレスを含みます。

```
/* Set MSC(SCSI) initialize parameters */
gSample_MSC_CB.pfnConnStat          = _Sample_ConnStat;
gSample_MSC_CB.pfnMassStorageReset = _Sample_MassStorageReset;
gSample_MSC_CB.pfnGetMaxLun         = _Sample_GetMaxLun;
gSample_MSC_CB.pfnMediaRead         = _Sample_MediaRead;
gSample_MSC_CB.pfnMediaWrite        = _Sample_MediaWrite;
gSample_MSC_CB.pfnSendData          = _Sample_SendData;
gSample_MSC_CB.pfnRecvData          = _Sample_RecvData;

/* Initialize MSC(SCSI) driver */
USBD_MSC_SCSI_Init(&gSample_MSC_CB);
```

3. RAM Disk を初期化します。

```
static void _Sample_RAM_Disk_Init(void)
{
    /* Clear sector 0 */
```

```
    memset(g_aucRamDisk[0], 0x00U, BLOCK_SIZE);
}
```

4. 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; }
```

5. USBD 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
   - USBD 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
   - USBD 割り込みの許可
   - エンドポイント 0 リセット

```
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;
}
```

6. 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 RAM Disk(**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。ユーザは、PC を使用してファームウェアをフォーマットすることが可能です。フォーマット後、ファイルの保存、読み出しができます。

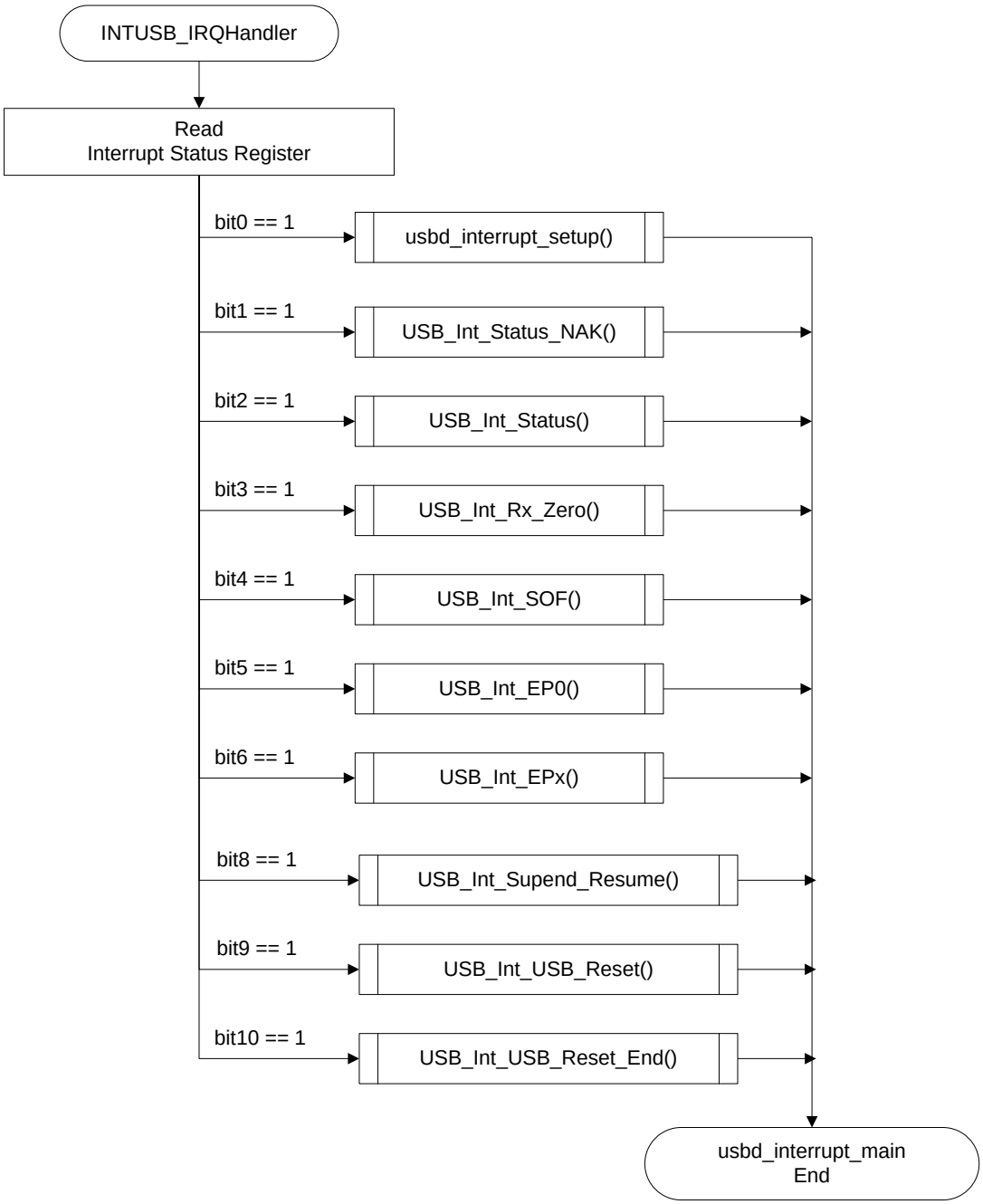
   詳細は以下の関数を参照してください。

   - **usbd_hw_interrupt.c** 内の INTUSB_IRQHandler()関数と INTUSBPON_IRQHandle()関数。
   - **usbd_device_request.c** 内の usbd_interrupt_setup_standard()関数。
   - **usbd_msc.c** 内の usbd_interrupt_MSC_req()関数。
   - **usbd_scsi.c** 内の

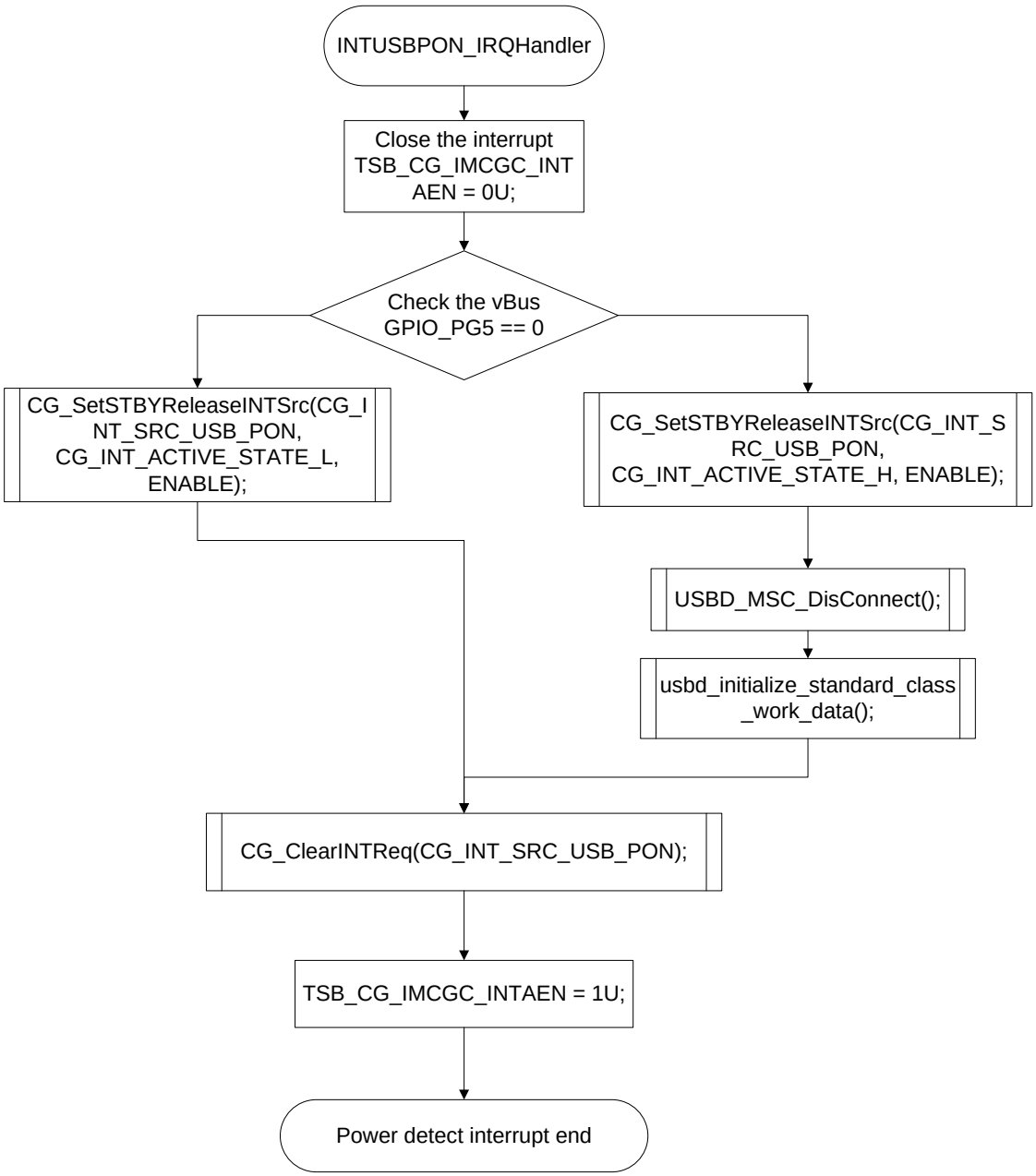
_USBD_MSC_SCSI_Command_Analysis(uint8_t *pucData)関数。

7.2 フローチャート

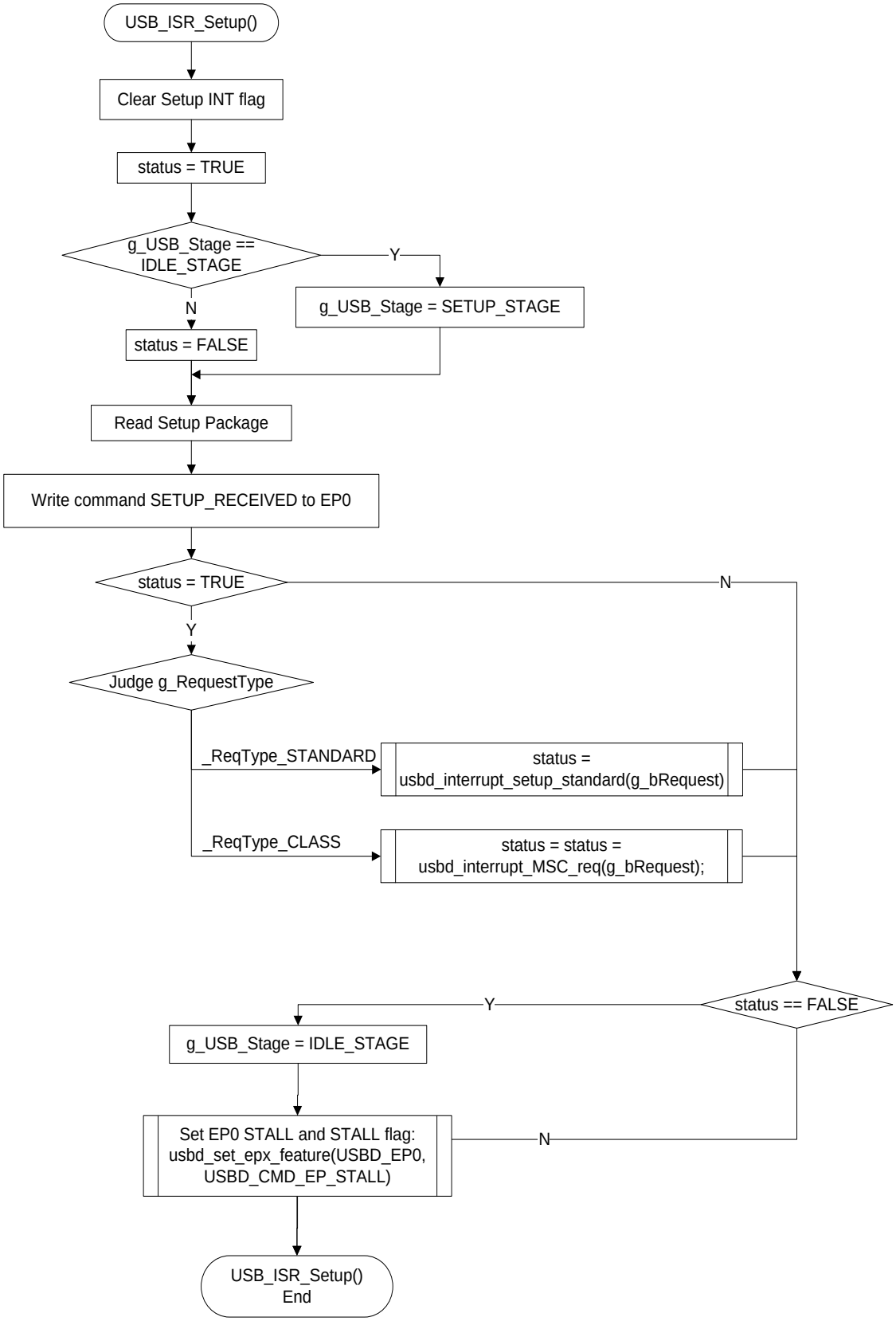
1) 主要USB割り込みルーチンのフローチャートを下記に示します。



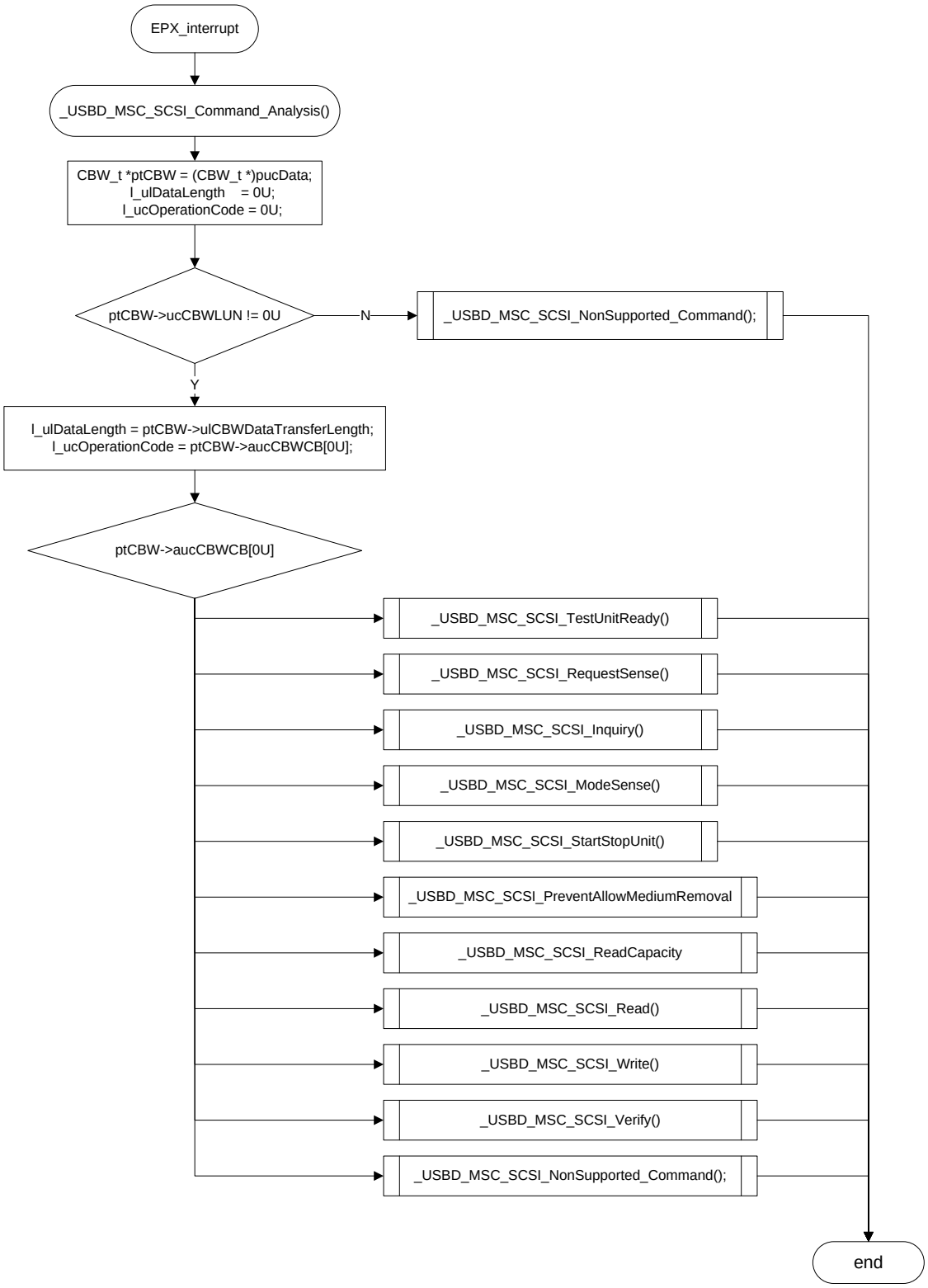
2) VBUS検出割り込みルーチンのフローチャートを下記に示します。



3) 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。



4) 下記フローチャートでは、CBWにより実行されるSCSIコマンドを解析します。CBWを受信すると、epx割り込みが発生し、システムはCBWを解析するために**_USBD_MSC_SCSI_Command_Analysis(uint8_t *pucData)**関数を呼び出します。



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。

```
/* Callback Information */
typedef struct {
    MSC_CB_ConnStat                 pfnConnStat;
    MSC_CB_MassStorageReset         pfnMassStorageReset;
    MSC_CB_GetMaxLun                pfnGetMaxLun;
    SCSI_CB_Media                   pfnMediaRead;
    SCSI_CB_Media                   pfnMediaWrite;
    SCSI_CB_TrnsData                pfnSendData;
    SCSI_CB_TrnsData                pfnRecvData;
} SCSI_CB_t;
```

コールバック関数は、3種類に分類することができます。
1. ハードウェア応答:
pfnConnStat; 接続/切断用コールバック関数

2. MSCクラスリクエスト: これらはホストがデバイスにリクエストを送信する際に呼び出されます。
pfnMassStorageReset; pfnGetMaxLun;

3. SCSIデータ転送:
pfnMediaRead: メディアリード用コールバック関数

pfnMediaWrite: メディアライト用コールバック関数

pfnSendData: データ送信(読み出し)用コールバック関数。ホストへのデータ転送が完了するごとに、本コールバック関数が、データ転送が完了したかどうかを判定するために呼び出されます。

pfnRecvData: 受信用(書き込み)用コールバック関数。ホストからデバイスへのデータ転送が完了するごとに、デバイスが全データを受信したか判定するため、本コールバック関数が呼び出されます。