TOSHIBA

Application Note

USBD Loopback

TMPM366

Ver 1

Sep, 2017

TOSHIBA ELECTRONIC DEVICES & STORAGE CORPORATION

CMDR-M366UDC -01E

RESTRICTIONS ON PRODUCT USE

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

© 2017 Toshiba Electronic Devices & Storage Corporation

Index

1. Introduction

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

This document will introduce the key points in creating a simple USB CDC loopback demonstration program, which is based on Toshiba TMPM366FDFG Evaluation board and USBD driver.

It is assumed that the reader has had some basic knowledge, such as “descriptor”, “endpoint” and “requests”, for the USB and Communication Device Class.

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.

- Universal Serial Bus Class Definitions for Communication Devices Version 1.1 January 19.1999
  ➔ please mainly read chapter 5 and 6.

2. CDC Loopback Demo

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

Typical examples of CDC class devices include:

- Telecommunications devices i.e., analog modems, ISDN terminal adapters and digital telephones.
- Networking devices, i.e., ADSL modems, cable modems.

On TMPM366FDFG evaluation board, a simple CDC demonstration program was designed to simulate as a COM device.

3. Environment

The environment includes:

Hardware environment:

- TMPM366 evaluation board (Not for sale)
  (Hereafter, we will call the evaluation board as “EVB” for short.)
- IAR J-Link-ARM v7.0
- The pin D+ of USB is connected with pin64 of EVB
- The pin D- of USB is connected with pin63 of EVB
- The pin VBUS of USB is connected with pin20 of EVB
- The pin GND of USB is connected with GND of EVB

Software environment:

- IAR Embedded Workbench for ARM 6.401 which should support M366
- PC (OS: Windows XP)
- COM debug tool: AccessPort 1.37 or others

4. Usage Procedure

1 Download the USBD CDC loopback program to the M366 board by IAR J-Link-ARM v7.0.

2 Connect the PC to EVB by USB.

3 Reset the EVB.

4 EVB will be recognized as a USB device.

5 Install the Driver for USB device:

5.1 Open the “Found New Hardware Wizard”.

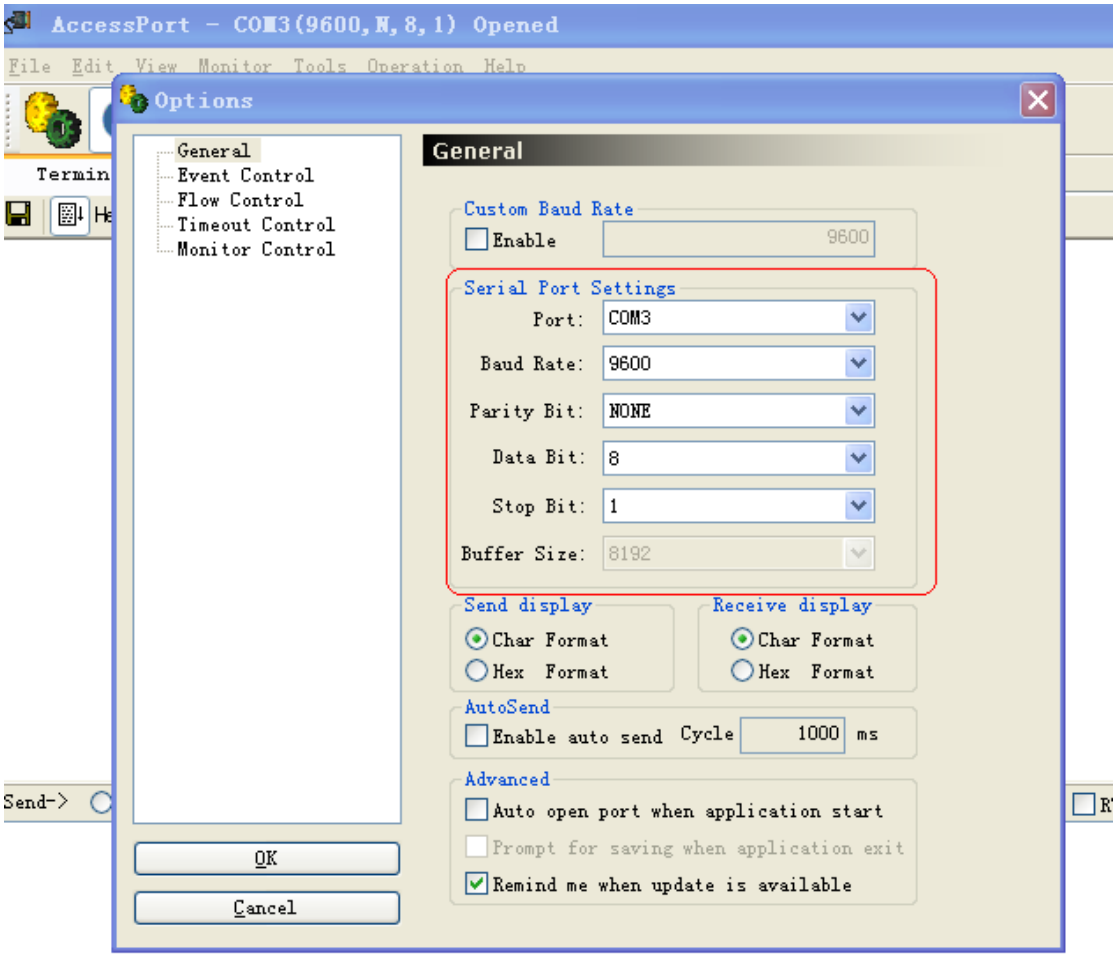
5.2 Select “Install from a list or specific location”.

5.3 Specify the location for “grusbcdc2.inf” file. (please refer to chapter 10)

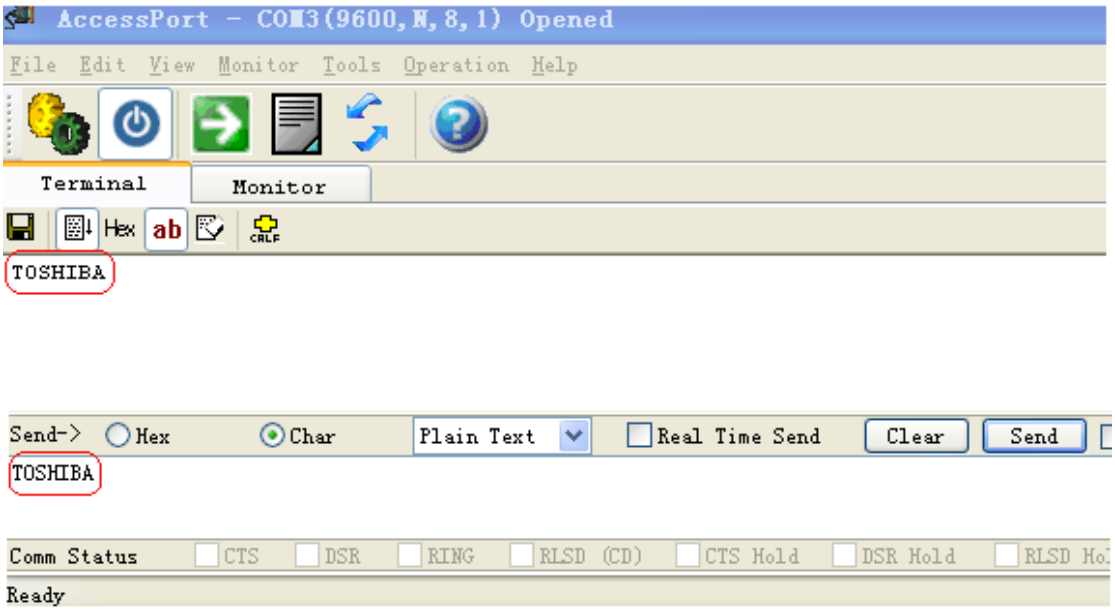
5.4 Specify the location for “usbser.sys” file which is windows file.
Ex: C: \WINDOWS\system32\drivers

5.5 After install the USB driver is finished,
It will be recognized as a standard COM port (Ex: COM3).
Open System Properties->Device Manager, display example as below:

6 Open the COM debug tool (Ex: AccessPort.exe), configure it as 9600bps,8 data bit ,none parity and 1 stop bit and COM3 port as below:



7 Once the AccessPort.exe send the data (Ex: “TOSHIBA”) to the COM3, the AccessPort.exe will receive the same data (Ex: “TOSHIBA”) at the same time. (see the diagram below)

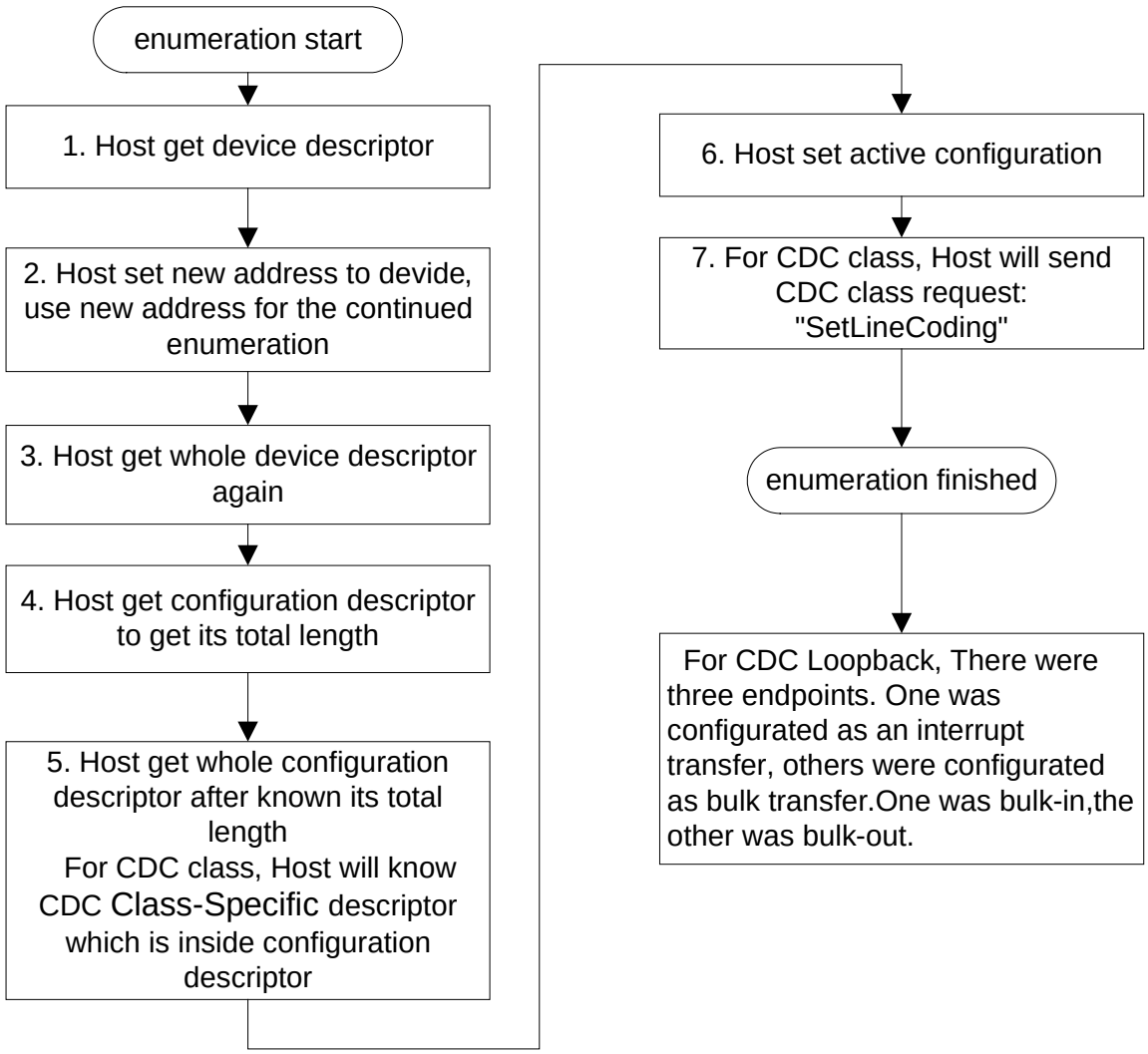


5. Enumeration

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 CDC Class device:



6. Descriptor

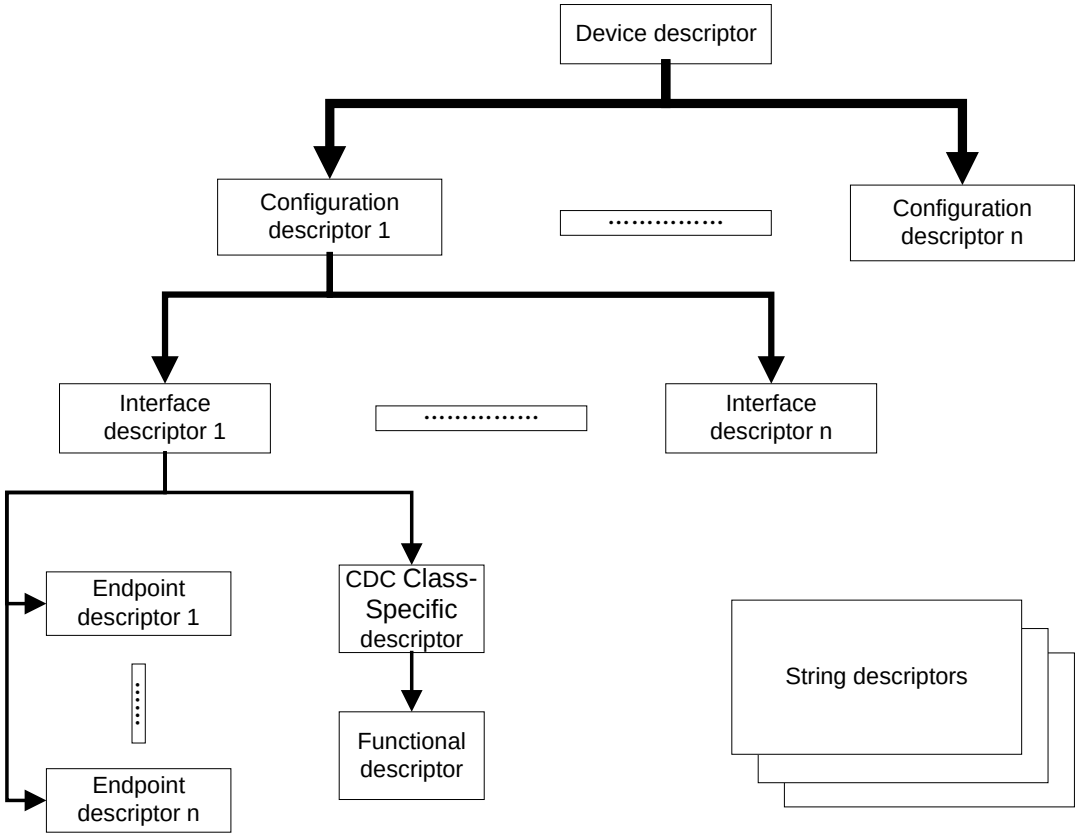
6.1 Device Descriptor

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_cdc.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.

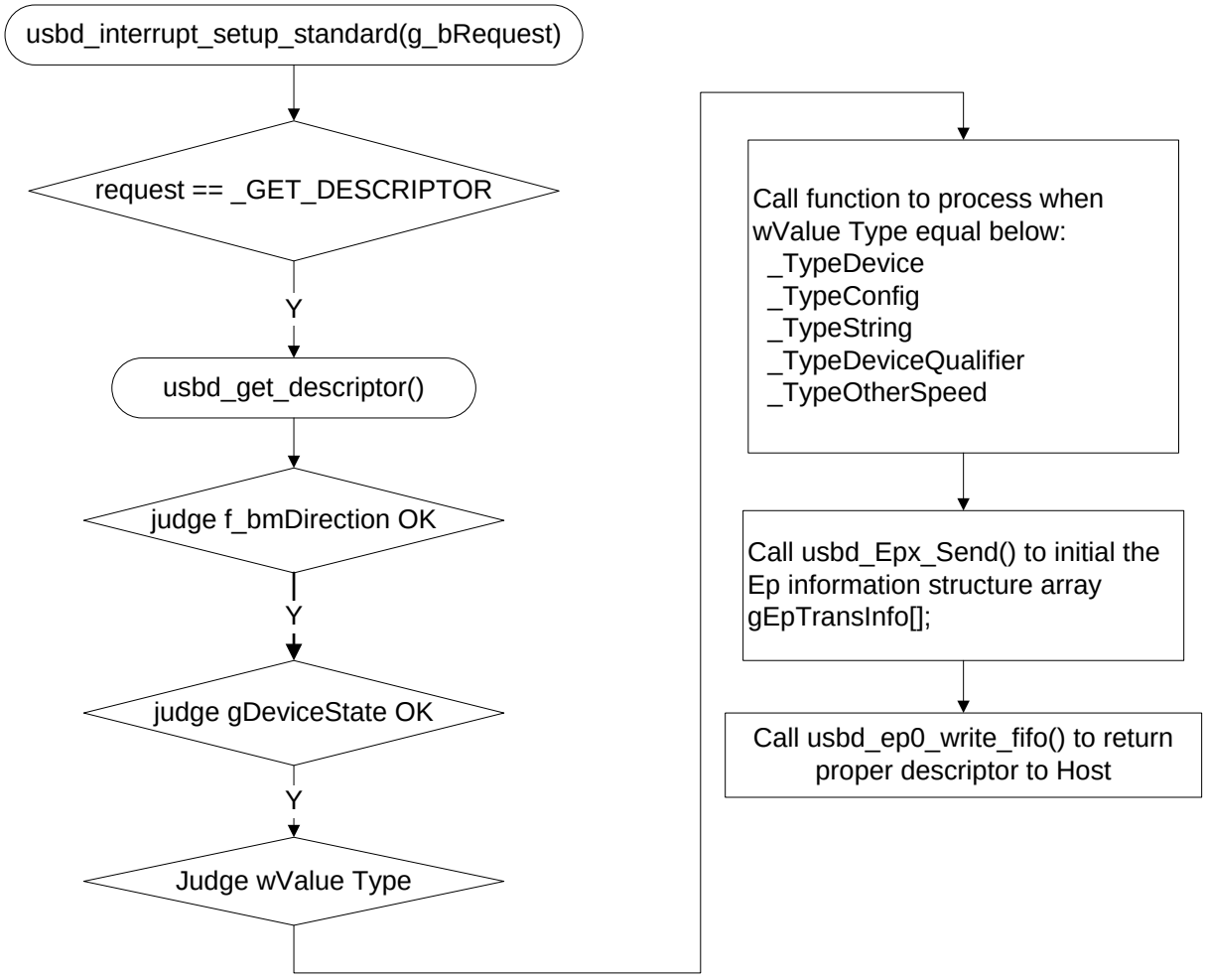
The layer structure of descriptor is as shown in the figure below:



6.2 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:



6.3 CDC Request

Send Encapsulated Command *(class-specific request)*
This request is used to issue a command in the format of the supported control protocol of the Communication Class interface (It does nothing in this demo).

To issue the Send Encapsnlated Command the host shall issue a device request on the default pipe of:

- **bmRequestType:** Class, Interface, host to device
- **bRequest:** 0 (00h)
- **wValue:** 0
- **wIndex:** the interface number
- **wLength:** 0

Get Encapsnlated Response *(class-specific request)*
This request is used to issue a command in the format of the supported control protocol of the Communication Class interface. (It does nothing in this demo).

To issue the Get Encapsnlated Response the host shall issue a device request on the default pipe of:

- **bmRequestType:** Class, Interface, device to host
- **bRequest:** 1 (01h)
- **wValue:** 0
- **wIndex:** the interface number
- **wLength:** 0

Set Line Coding *(class-specific request)*
This request allows the host to specify typical asynchronous line-character formatting properties, which may be required by some applications. This request applies to asynchronous byte stream data class interfaces and endpoints; it also applies to data transfers both from the host to the device and from the device to the host.

To issue the Set Line Coding the host shall issue a device request on the default pipe of:

- **bmRequestType:** Class, Interface, host to device
- **bRequest:** 20 (14h)
- **wValue:** 0
- **wIndex:** the interface number
- **wLength:** the size of structure(07h)
- **data:**Line Coding Structure

Get Line Coding *(class-specific request)*
This request allows the host to find out the currently configured line coding.
To issue the Get Line Coding the host shall issue a device request on the default pipe of:

- **bmRequestType:** Class, Interface, device to host
- **bRequest:** 21 (15h)
- **wValue:** 0
- **wIndex:** the interface number
- **wLength:** the size of structure(07h)
- **data:**Line Coding Structure

SetControlLineState *(class-specific request)*
This request generates RS-232/V.24 style control signals.
To issue the SetControlLineState the host shall issue a device request on the default pipe of:

- **bmRequestType:** Class, Interface, host to device
- **bRequest:** 22 (16h)
- **wValue:** Control Signal Bitmap
- **wIndex:** the interface number
- **wLength:** the size of structure(07h)

7. Software File Structure

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

```
/---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_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
```

8. Key Code and Flowchart

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

8.1 Key code

1. Initialize Line Coding Structure

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

l_aucSampleLineCoding[0] ~[3] is data terminal rate,
Ex: 0x00002580 is 9600bps.
l_aucSampleLineCoding[4] is stop bits. Ex: 0x00 is 1 stop bit.
l_aucSampleLineCoding[5] is parity. Ex: 0x00 is none parity.
l_aucSampleLineCoding[6] is data bits length. Ex: 0x08 is 8bits.

2. Clear line state

```
l_usSampleControlLineState = 0x0000U;
```

3. Set CDC initialize parameters.

```
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.pfnReciveData              = _Sample_RecvData;
```

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

```
static void _Sample_ConnStat(bool bStat)
{
    if (bStat == true) {
        /* 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 is also a call back function. It's used to get the Line Coding Structure from the device, which will be called when host sent out the Get Line Coding request to the device.

```
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 is also a call back function. It's used to set the Line Coding Structure to the device, which will be called when host sent out the Set Line Coding request to the device.

```
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 is also a call back function. It's used to set the l_usSampleControlLineState, which will be called when host sent out the SetControlLineState request to the device.

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

    /* Keep line state */
    l_usSampleControlLineState = usValue;

    return;
}
```

_Sample_ResponseAvailable is also a call back function. It does nothing in this demo.

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

    return;
}
```

_Sample_SendData is also a call back function. It's used to send the data to pc, which will be called when data transfer is complete. The function USBD_CDC_RecvData called in it used to receive the data from pc.

```
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 is also a call back function. It's used to receive the data from pc, which will be called when data transfer is complete. The function USBD_CDC_SendData called in it used to send the data to pc.

```
static void _Sample_RecvData(uint32_t ulSize, uint8_t *pucData, uint16_t usStat)
{
    if (usStat == EPSTAT_COMPLETE) {
```

```
            /* Request for data send */
            gbStatDC = USBD_CDC_SendData(ulSize, l_aucSampleBuf);
            /* No care status in sample */
        }

        return;
}
```

4. Initialize CDC driver

```
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.pfnReciveData           = p_cdc_cb->pfnReciveData;

    /* 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_bUsbStat = false;

    return;
}
```

5. 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,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. 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;
}
```

7. After proper configuration, the firmware will wait for USB interrupt and then finish the standard enumeration steps so as to be treated as a COM (**See note below**) in interrupt service routine.

   For more details, please refer to the function **INTUSB_IRQHandler()** and **INTUSBPON_IRQHandle()** in ***usbd_hw_interrupt.c*** and function **usbd_interrupt_setup_standard()** in ***usbd_device_request.c*** and **usbd_interrupt_CDC_req()** in ***usbd_cdc.c.***

8. After enumeration finished, you can use com debug tool to send some data to the USB which is treated as COM, and then the com debug tool will reveive the same data sent from the COM. So the loopback of data is done.

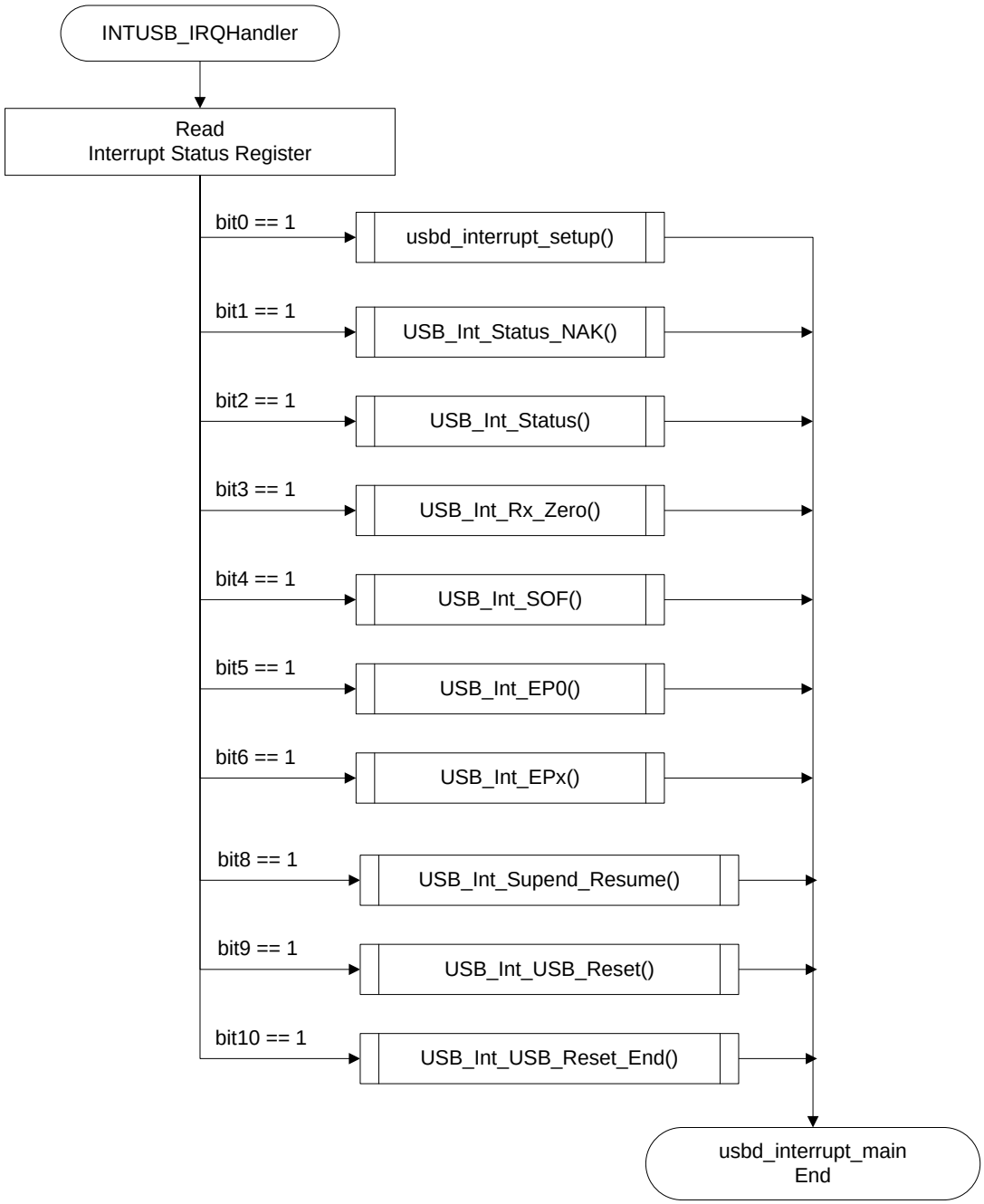
   For more detailed contents, please refer to the condition to call the function **_Sample_SendData()** and **_Sample_RecvData().**

   Note:

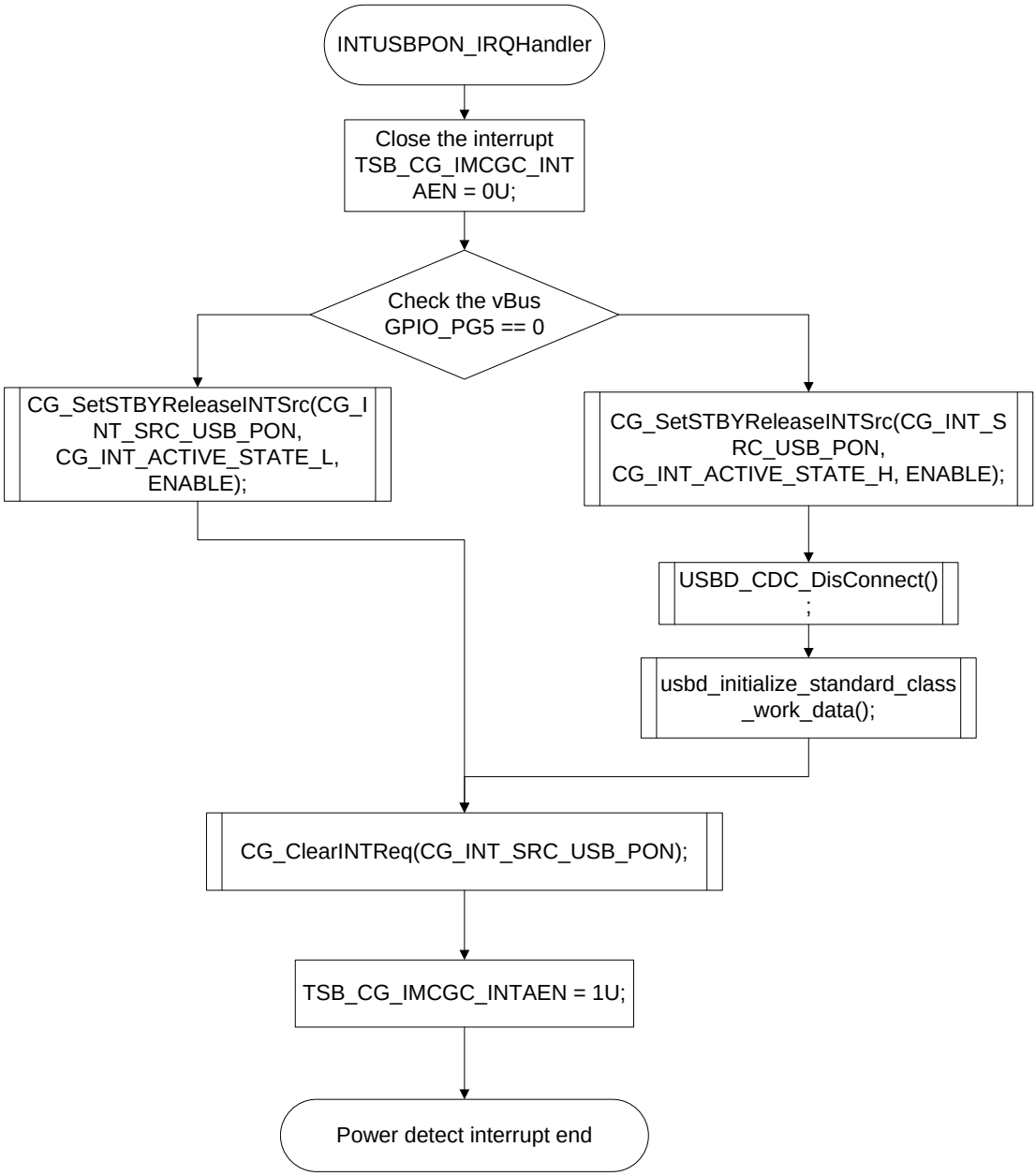
The file grusbcdc2.inf must be setup before the pc recognizes the USB device as COM port.

8.2 Flowchart

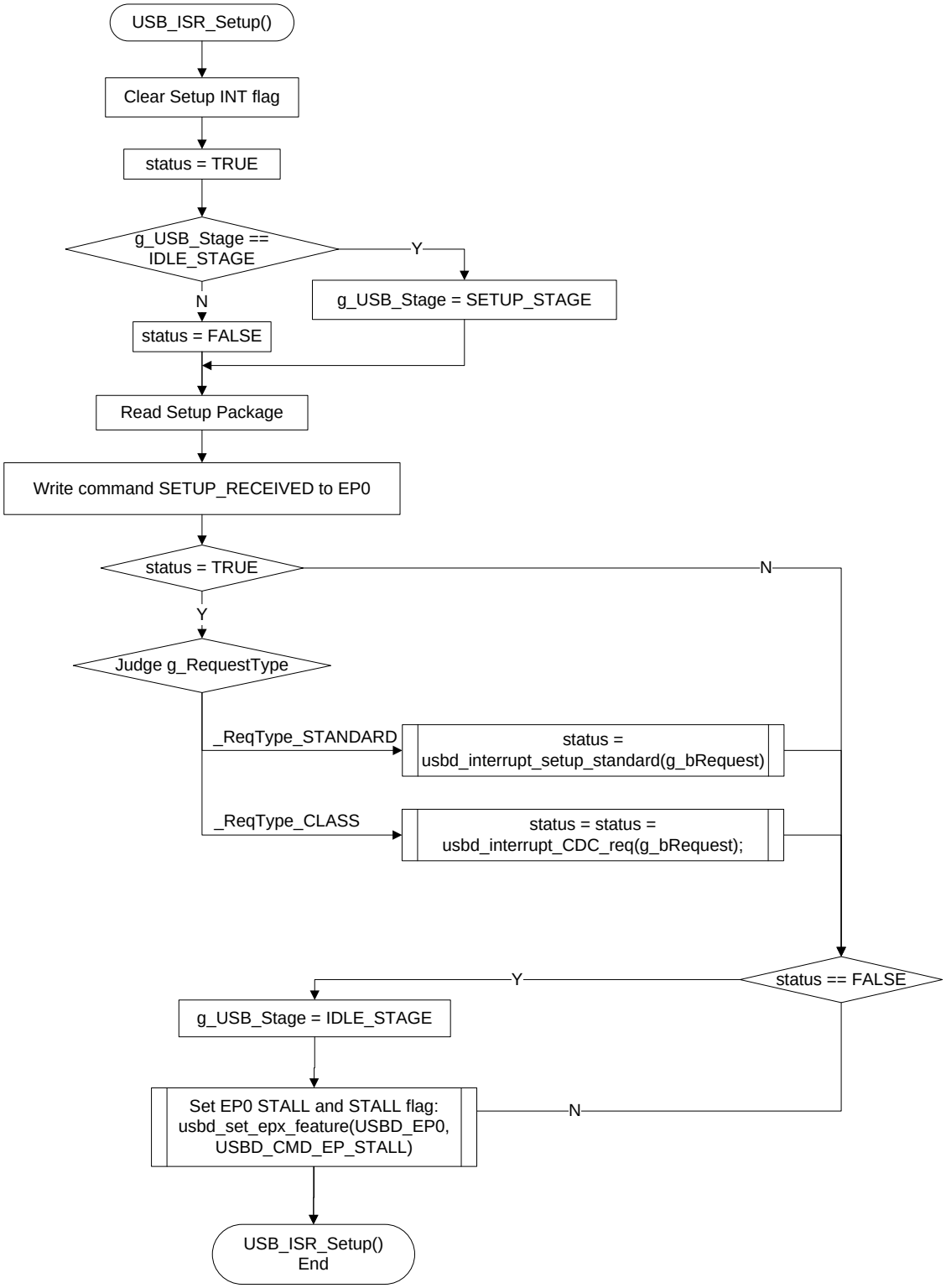
The following flowchart is for the main USB interrupts routine



The following flowchart is for VBUS detect Interrupt Service Routine.



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



9. Call back function

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

```
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;
```

They can be divided into four kind of function:
1. Hardware response:
pfnConnStat;Callback for connect/disconnect

2. CDC Class requests: they will be called when the host sent the request to the device.
pfnSendEncapsulatedCmd; pfnGetEncapsulatedRes; pfnSetCommFeature; pfnGetCommFeature; pfnClearCommFeature; pfnSetLineCoding; pfnGetLineCoding; pfnSetControlLineState; pfnSendBreak;

3. Communication Interface Class notifications:
pfnNetworkConnection; pfnResponseAvailable; pfnSerialState;

4. Data transfer:
pfnSendData: Transfer data to EP2 in the bulk-out transfer. When call function "***bool USBD_CDC_RecvData (uint32_t ulSize, uint8_t *pucData)***", this call back function will be called.

pfnReciveData: Transfer data to EP1 in the bulk-in transfer. When call function "***bool USBD_CDC_CtrlSendData(uint32_t ulSize, uint8_t *pucData)***", this call back will be execute.

10. grusbcdc2.inf file

- **Code and Explanation for the grusbcdc2.inf file.**
  INF means Device ***INF***ormation File,

```
[Version]                    ; required section
Signature="$Windows NT$"
; Specify the suitable OS for this INF file by Signature.
; Symbol "$" is delimiter.

Class=Ports                  ; device's class name
ClassGuid={4D36E978-E325-11CE-BFC1-08002BE10318}
; Device class Registry's GUID which is 128 bits identifier.

Provider=%MSFT%              ; "MSFT" will be define in strings section.
LayoutFile=layout.inf
DriverVer=01/01/2005,1.00    ; version information, date format is "mm/dd/yyyy"


[Manufacturer]
; describe the device manufacture which can be recognized by this INF file.
%MFNAME%=DeviceList,NTx86,NTia64,NTamd64
; "MFNAME" will be defined in strings section.

[DestinationDirs]
DefaultDestDir=12            ; specify the installed location

[SourceDisksFiles]

[SourceDisksNames]


[DeviceList]
%DESCRIPTION%=DriverInstall, USB\VID_0930&PID_6505
; The vendor identifier (VID) is 0930 which is created by the USB committee.
; The product identifier (PID) is 6505 which is created by the manufacture.
; When change the product, the VID and PID may be changed.

[DeviceList.NTx86]
%DESCRIPTION%=DriverInstall.NTx86, USB\VID_0930&PID_6505

[DeviceList.NTia64]
%DESCRIPTION%=DriverInstall.NTia64, USB\VID_0930&PID_6505

[DeviceList.NTamd64]
%DESCRIPTION%=DriverInstall.NTamd64, USB\VID_0930&PID_6505


[DriverInstall.nt]
```

```
include=mdmcpq.inf
CopyFiles=FakeDriverCopyFiles          ; copy files
AddReg=DriverInstall.nt.AddReg         ; Add register item section name

[DriverInstall.NTx86]
include=mdmcpq.inf
CopyFiles=FakeDriverCopyFiles
AddReg=DriverInstall.nt.AddReg

[DriverInstall.NTia64]
include=mdmcpq.inf
CopyFiles=FakeDriverCopyFiles
AddReg=DriverInstall.nt.AddReg

[DriverInstall.NTamd64]
include=mdmcpq.inf
CopyFiles=FakeDriverCopyFiles
AddReg=DriverInstall.nt.AddReg


[DriverInstall.nt.Services]
include=mdmcpq.inf
AddService=usbser, 0x00000002, DriverService
; “usbser” is the name of service which need be installed.
; “0x00000002” is the system flag,
; “DriverService” is the special section which as defined in the INF file.

[DriverInstall.NTx86.Services]
include=mdmcpq.inf
AddService=usbser, 0x00000002, DriverService

[DriverInstall.NTia64.Services]
include=mdmcpq.inf
AddService=usbser, 0x00000002, DriverService

[DriverInstall.NTamd64.Services]
include=mdmcpq.inf
AddService=usbser, 0x00000002, DriverService


[FakeDriverCopyFiles]
usbser.sys,,,0x20

[DriverInstall.nt.AddReg]
HKR,,DevLoader,,*ntkern
HKR,,NTMPDriver,,usbser.sys
HKR,,EnumPropPages32,,"MsPorts.dll,SerialPortPropPageProvider"


[DriverService]
DisplayName=%SERVICE%
ServiceType=1                          ; SERVICE_KERNEL_DRIVER
StartType=3                            ; SERVICE_DEMAND_START
ErrorControl=1                         ; SERVICE_ERROR_NORMAL
ServiceBinary=%12%\usbser.sys

;--------------------------------------------------------------------------
; String Definitions
;--------------------------------------------------------------------------
```

```
[Strings]                                   ; strings section
MSFT="Microsoft Corporation."
MFNAME="Grape Systems Inc."
DESCRIPTION="Communications Port"
SERVICE="USB CDC Comport Emulation Driver"
```

The more information about the INF file please refer to http://msdn.microsoft.com web