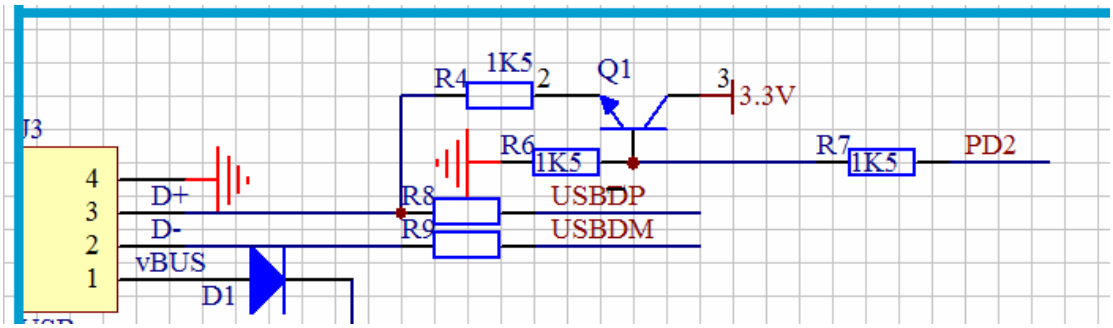
STM32 学习笔记 USB HID

———序

在 keilc 安装目录下 C:\Keil\ARM\Boards\Keil\MCBSTM32\下 USBHID 文件夹为 STM32 USB HID 操作例子。

将 USBHID 复制到用户目录下，进行改动就能运行 STM32 的 USB HID 程序。

在 USB 协议中，数据线的上下拉电阻为 USB 启动信号。试验板 USB 电路如下图：

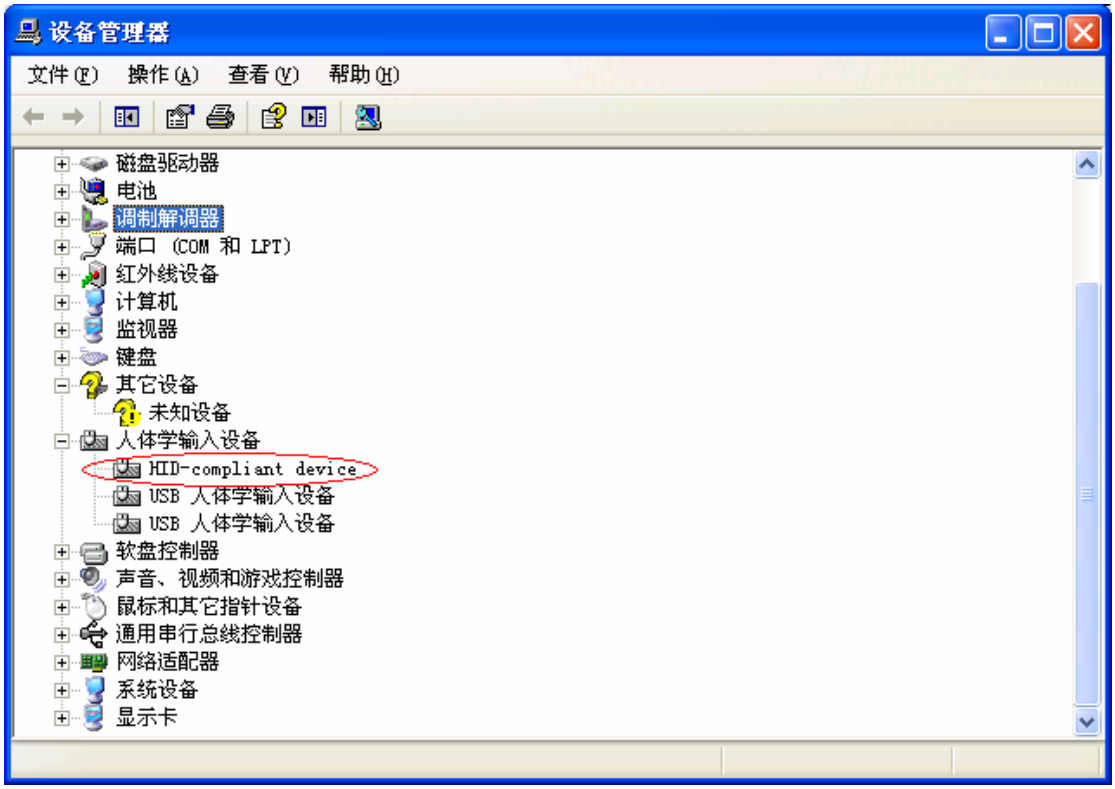


可以看出，上电复位后，由于对 PD2 无操作，则 USB 的数据线 D＋上拉电阻没有使能。在 USB 则处于空闲。如果要连接 USB 设备，只需要将 PD 使能三极管导通，则能让 USB 主机检测到本 USB 设备。所以在 USB 连接函数中改动:

```
void USB_Connect (BOOL con) {
 //=================================================================
  GPIO_InitTypeDef m_GPIO_InitTypeDef;
  m_GPIO_InitTypeDef.GPIO_Pin     = GPIO_Pin_2;
  m_GPIO_InitTypeDef.GPIO_Mode    = GPIO_Mode_Out_PP;//推挽输出
  m_GPIO_InitTypeDef.GPIO_Speed = GPIO_Speed_2MHz;//速度 2M
  GPIO_Init(GPIOD,&m_GPIO_InitTypeDef);
  //=================================================================
  //让 usb 处于复位状态，让 USB 模拟电路准备
  CNTR = CNTR_FRES;                                  /* Force USB Reset */
  ISTR = 0;                                         /* Clear Interrupt Status */
  if (con)  //:连接 USB
  {//清除 USB 复位信号，向 USB 主机发送"恢复请求",并且使能复位中断请求
   //则当 USB 主机向本机发送“恢复请求”后，相应中断，并调用 USB_Reset 函数.
   //说明：如果此操作在 1－15ms 有效,USB 主机将对 USB 模块发出"唤醒操作"
     CNTR = CNTR_RESETM;                              /* USB Reset Interrupt Mask */
      GPIO_SetBits(GPIOD,GPIO_Pin_2);//PD2＝1：连接 USB

  }
  else       //:断开 USB
  {//清除 USB 复位信号,关闭 USB 电源
     CNTR = CNTR_FRES | CNTR_PDWN;                  /* Switch Off USB Device */
      GPIO_ResetBits(GPIOD,GPIO_Pin_2);//PD2＝0，不连接 USB 设备
  }
}
```

至此，程序编译连接，烧写到 STM32，程序运行。
可以从我的电脑，设备管理器看到 HID－compliant device 为 USB HID 设备。



USB 连接测试等，因为 USB HID 协议，主机 32ms(HID 协议可以设置 ms) GetInReport。则在 GetInReport 函数中加了 USB 连接测试灯，当连接成功。USB 测试灯一闪一闪

```
void GetInReport (void)
{

   static    unsigned int i=0;
   InReport = 0x00;
   if ((GPIOA->IDR & S2) == 0) InReport |= 0x01;          // S2 pressed means 0
   if ((GPIOC->IDR & S3) == 0) InReport |= 0x02;          // S3 pressed means 0

   InReport = GPIO_ReadInputDataBit(GPIOC,GPIO_Pin_0);//读取按键值返回
   i++;
   if(i%10==0)
   {
       GPIOC->ODR = GPIOC->ODR ^ 0x0200;//每隔 640ms 亮灭一次。32ms×10 × 2
   }
}
```

STM32 学习笔记 USB HID

－－－－－STM32 USB HID 固件学习分析

1.系统复位和上电复位

发生系统复位或者上电复位时，应用程序首先需要做的是提供USB 模块所需要的时钟信号，然后清除复位信号，使程序可以访问USB 模块的寄存器。复位之后的初始化流程如下所述：

首先，由应用程序激活寄存器单元的时钟，再配置设备时钟管理逻辑单元的相关控制位，清除复位信号.

```
void USB_Init (void) {

  RCC->APB1ENR |= (1 << 23);                    /* enable clock for USB 时钟信号*/
  /* Enable USB Interrupts */
  NVIC->IPR [5] |= 0x00000010;                  /* set priority lower than SVC */
  NVIC->ISER[0] |= (1 << (USB_LP_CAN_RX0_IRQChannel & 0x1F));

  /* Control USB connecting via SW */
  RCC->APB2ENR |= (1 << 5);                     /* enable clock for GPIOD */
}
```

2. 开启模拟单元，使 USB 处于复位状态，直到模拟但愿准备完毕。

其次，必须配置CNTR 寄存器的PDWN 位用以开启USB 收发器相关的模拟部分，程序体现CNTR = CNTR_FRES;模拟单元准备完毕后，程序复位撤销，即一次复位操作。然后使能复位中断，那么主机将会检测到USB设备的复位，给出复位事件给USB设备。

```
void USB_Connect (BOOL con) {

 //=============================================================
  GPIO_InitTypeDef m_GPIO_InitTypeDef;
  m_GPIO_InitTypeDef.GPIO_Pin   = GPIO_Pin_2;
  m_GPIO_InitTypeDef.GPIO_Mode  = GPIO_Mode_Out_PP;//推挽输出
  m_GPIO_InitTypeDef.GPIO_Speed = GPIO_Speed_2MHz;//速度2M
  GPIO_Init(GPIOD,&m_GPIO_InitTypeDef);
  //=============================================================
  //让usb处于复位状态，让USB模拟电路准备
  CNTR = CNTR_FRES;                             /* Force USB Reset */

  ISTR = 0;                                     /* Clear Interrupt Status */
  if (con)   //:连接USB
  {//清除USB复位信号，向USB主机发送"恢复请求",并且使能复位中断请求
   //则当USB主机向本机发送“恢复请求”后，相应中断，并调用USB_Reset函数.
   //说明：如果此操作在1－15ms有效,USB主机将对USB模块发出"唤醒操作"
    CNTR = CNTR_RESETM;                         /* USB Reset Interrupt Mask */
  GPIO_SetBits(GPIOD,GPIO_Pin_2);
```

```
    }
    else        //:断开USB
    {//清除USB复位信号,关闭USB电源
      CNTR = CNTR_FRES | CNTR_PDWN;                 /* Switch Off USB Device */
    GPIO_ResetBits(GPIOD,GPIO_Pin_2);
    }
  }
```

3．USB 主机发送给 USB device 复位事件，STM32 的复位事件

```
 void USB_Reset (void) {
/* Double Buffering is not yet supported                    */

  ISTR = 0;                                             /* Clear Interrupt Status */
  //:CNTR_RESETM 在使能连接时被中断使能
  //:CNTR_CTRM 此处添加(正确传输中断使能==>某个端点成功完成一次传输.)
  //:下一次事件中断将检测 CNTR_CTRM 中断
  //   究竟是哪个端点成功传输(ISTR:EP_ID[3:0]),数据方向(ISTR:DIR)
  CNTR = CNTR_CTRM | CNTR_RESETM |
           (USB_SUSPEND_EVENT ? CNTR_SUSPM     : 0) |
           (USB_WAKEUP_EVENT   ? CNTR_WKUPM     : 0) |
           (USB_ERROR_EVENT    ? CNTR_ERRM      : 0) |
           (USB_ERROR_EVENT    ? CNTR_PMAOVRM : 0) |
           (USB_SOF_EVENT      ? CNTR_SOFM      : 0) |
           (USB_SOF_EVENT      ? CNTR_ESOFM     : 0);

  FreeBufAddr = EP_BUF_ADDR;                        //自由缓冲区地址
  BTABLE = 0x00;                                    /* set BTABLE Address */

  /* Setup Control Endpoint 0 */

  pBUF_DSCR->ADDR_TX = FreeBufAddr;         //:
  FreeBufAddr += USB_MAX_PACKET0;     //:
  pBUF_DSCR->ADDR_RX = FreeBufAddr;         //:
  FreeBufAddr += USB_MAX_PACKET0;     //:
  if (USB_MAX_PACKET0 > 62)
  {
    pBUF_DSCR->COUNT_RX = ((USB_MAX_PACKET0 << 5) - 1) | 0x8000;
  }
  else
  {
    pBUF_DSCR->COUNT_RX =     USB_MAX_PACKET0 << 9;
  }

  //配置端点 0：
  //1：     EP_CONTROL(控制端点) bit[10:9](EP_TYPE[1:0])
```

```
   //2:    EP_RX_VALID(数据接受状态位)bit[13:12](STAT_RX[1:0])      ==>端点可用于接受
   EPxREG(0) = EP_CONTROL | EP_RX_VALID;

   //设置 USB 地址为 0
   DADDR = DADDR_EF | 0;                                  /* Enable USB Default Address */
}
```

4. STM32 的中断函数

```
   void USB_LP_CAN_RX0_IRQHandler (void)
{
   DWORD istr, num, val;
   istr = ISTR;
/*==========USB Reset Request=============*/
   if (istr & ISTR_RESET)
   {
      USB_Reset();
      #if USB_RESET_EVENT
          USB_Reset_Event();
      #endif
      ISTR = ~ISTR_RESET;
      #ifdef Debug_Uart
      printf("USB_RESET_EVENT\n");
      #endif
   }
 //……略过一些程序……
/*==========EndpointInterrupts==========================================*/
   while ((istr = ISTR) & ISTR_CTR)
   {//istr = ISTR：表示不是前面用到的中断
    //:CNTR_CTRM 表示某个端点成功完成一次传输.在复位中使能此中断
    //:第一次进入这个中断后，开始枚举，枚举一般都是端点 0 的
    //USB_EVT_IN、USB_EVT_SETUP 事件。USB_EVT_IN 是 USB_EVT_SETUP 事件需要
    //返回相应数据而使用的。枚举事务几乎都是 USB_EVT_SETUP 事务。
   //   究竟是哪个端点成功传输(ISTR:EP_ID[3:0]),数据方向(ISTR:DIR)
      ISTR = ~ISTR_CTR;
      num = istr & ISTR_EP_ID;   //num 端点号
//=========================
//#define EPxREG(x)            REG(USB_BASE_ADDR + 4*(x))     /* EndPoint Registers */
//#define REG(x)               (*((volatile unsigned int *)(x)))
//:通过前面两句知道 val = EPxREG(num);就是读取端点 num 寄存器的值。
      val = EPxREG(num);
      if (val & EP_CTR_RX)
      {//:端口 num    EP_CTR_RX 正确接受标志
      //bit[10:9]=EP_TYPE[1:0]:端点类型
         EPxREG(num) = val & ~EP_CTR_RX & EP_MASK;    //清除相关标志位
         if (USB_P_EP[num])
```

```
        {
          if (val & EP_SETUP)
          {//如果使 SETUP 包
            USB_P_EP[num](USB_EVT_SETUP);
          }
          else
          {//否则为 OUT 包
            USB_P_EP[num](USB_EVT_OUT);
          }
        }
      }
      if (val & EP_CTR_TX)
      {//::端口 num   EP_CTR_TX 正确发送标志
        EPxREG(num) = val & ~EP_CTR_TX & EP_MASK;
        if (USB_P_EP[num])
        {//为 IN 包
          USB_P_EP[num](USB_EVT_IN);
        }
      }
    }
}
```

5．枚举过程分析。

5．1 说明：枚举使用端点 0，在 void USB_EndPoint0 (DWORD event)函数处理。

STM32USB 枚举串口调试打印输出：

```
USB_RESET_EVENT
USB_RESET_EVENT
USB_Suspend
USB_RESET_EVENT
USB_WakeUp
USB_EVT_SETUP
...REQUEST_STANDARD
......USB_REQUEST_GET_DESCRIPTOR
USB_EVT_IN
USB_RESET_EVENT
USB_EVT_SETUP
...REQUEST_STANDARD
......USB_REQUEST_SET_ADDRESS
USB_EVT_SETUP
...REQUEST_STANDARD
......USB_REQUEST_GET_DESCRIPTOR
USB_EVT_IN
USB_EVT_IN
USB_EVT_IN
USB_EVT_SETUP
```

...REQUEST_STANDARD
.....USB_REQUEST_GET_DESCRIPTOR
USB_EVT_IN
USB_EVT_IN
USB_EVT_SETUP
...REQUEST_STANDARD
.....USB_REQUEST_GET_DESCRIPTOR
USB_EVT_IN
USB_EVT_RUUSBEUUUSB_EVT
UUSBUUSUSBCUUSB.USUSB_EVT_SETUP
...REQUEST_STUUUSB.SUSB_PUNUSB__USB_EVT_SETUP
...REQUEST_CLASS
.....REQUEST_TO_INTERFACE
USB_EVT_SETUP
...REQUEST_STANDARD
.....USB_REQUEST_GET_DESCRIPTOR
USB_EVUE

5．2 端点 0 USB_EVT_SETUP 事件函数分析说明

5.2.1 void USB_EndPoint0 (DWORD event)程序结构：

```
void USB_EndPoint0 (DWORD event)
{
  switch (event)
  {
    //*****************是SETUP包事件*************************************************
    case USB_EVT_SETUP://是SETUP包事件
        #ifdef Debug_Uart
        USB_SetupStage();
        USB_DirCtrlEP(SetupPacket.bmRequestType.BM.Dir);
        EP0Data.Count = SetupPacket.wLength;//数据长度, USB主设备
        /* wLength域这个域表明第二阶段的数据传输长度。
        switch (SetupPacket.bmRequestType.BM.Type)
        {//SETUP包请求 bmRequestType(D6..5: 种类)
        break;
    //***************是OUT包事件*************************************************
    case USB_EVT_OUT:
        if (SetupPacket.bmRequestType.BM.Dir == 0)
        {//0=主机至设备
        else
        {//没有数据需要读取
        break;
    //***************是IN包事件*************************************************
    case USB_EVT_IN:
        if (SetupPacket.bmRequestType.BM.Dir == 1)
        {//1=设备至主机
        else
        {
        break;
    //***************USB_EVT_IN_STALL*************************************************
    case USB_EVT_IN_STALL:
        #ifdef Debug_Uart
        USB_ClrStallEP(0x80);
        break;
    //***************USB_EVT_IN_STALL*************************************************
    case USB_EVT_OUT_STALL:
        #ifdef Debug_Uart
```

在枚举时，USB_EVT_OUT 包没有使用，只使用了

USB_EVT_IN、USB_EVT_SETUP

5.2.2 USB_EVT_IN 分析

```
        break;
    //***************是IN包事件*****************************************************
    case USB_EVT_IN:
        if (SetupPacket.bmRequestType.BM.Dir == 1)
        {//1=设备至主机
            //=====================================================
            //测试代码:
            #ifdef Debug_Uart
            printf("USB_EVT_IN\n");
            #endif
            //=====================================================
            USB_DataInStage();  ////继续传送数据到USB主机
        }
        else
        {
            if (USB_DeviceAddress & 0x80)
            {
                USB_DeviceAddress &= 0x7F;
                USB_SetAddress(USB_DeviceAddress);
            }
        }
        break;
    //***************USB_EVT_IN_STALL*********************************************
    case USB_EVT_IN_STALL:
```

可查看函数 USB_DataInStage()功能为传送通道 EndPoint0 的数据。在 USB_EVT_IN 事件中只需要传输在枚举时需要的数据到 USB 主机，需要传输的数据都时在枚举事务中与 USB 主机商量好的长度。

5.2.3 USB_EVT_SETUP 分析

A.

```
//*****************是SETUP包事件*************************************************
case USB_EVT_SETUP://是SETUP包事件
    #ifdef Debug_Uart
    printf("USB_EVT_SETUP\n");
    #endif
    USB_SetupStage();
    USB_DirCtrlEP(SetupPacket.bmRequestType.BM.Dir);
    EP0Data.Count = SetupPacket.wLength;//数据长度, USB主设备
    /* wLength域这个域表明第二阶段的数据传输长度。
    //传输方向由bmRequstType域的Direction位指出。wLength域为0则表明无数据传输。
    //在输入请求下，设备返回的数据长度不应多于wLength，但可以少于。在输出请求下，
    //wLength指出主机发出的确切数据量。如果主机发送多于wLength的数据，设备做出
    //的响应是无定义的      */
    switch (SetupPacket.bmRequestType.BM.Type)
    {//SETUP包请求 bmRequestType(D6..5: 种类)
    break;
//****************是OUT包事件*****************************************************
```

分析：USB_SetupStage()函数为将端点 0 的数据读到 SetupPacket 变量。SetupPacket 定义格式如下：

```
typedef __packed struct _USB_SETUP_PACKET {
  REQUEST_TYPE     bmRequestType;
  BYTE             bRequest;
  WORD_BYTE        wValue;
  WORD_BYTE        wIndex;
  WORD             wLength;
} USB_SETUP_PACKET;
```

可以看到这为一个 USB 总线 Setup 包格式，具体可参照 USB 协议第 9 章。SetupPacket.wLength 这个域表明第二阶段的数据传输长度。传输方向由

bmRequstType 域的 Direction 位指出。wLength 域为 0 则表明无数据传输。在输入请求下，设备返回的数据长度不应多于 wLength，但可以少于。在输出请求下，wLength 指出主机发出的确切数据量。如果主机发送多于 wLength 的数据，设备做出的响应是无定义的。

比如响应一个 GET_DESCRIPTOR 的 setup 包，则下一阶段需要传回 GET_DESCRIPTOR 的内容，而这个内容时固定的，这时长度就为 SetupPacket.wLengt，说明下一个阶段为 GET_DESCRIPTOR 数据传入 USB 主机（一个 USB_EVT_IN 事务实现），那在 GET_DESCRIPTOR 的内容的地址给 EP0Data 的地址(后面说明会指明在哪儿)，而长度在这儿指定。则下一个 IN 事务直接传输这个 EP0Data 的内容，直到将 EP0Data.Count 数据传输完毕。

B.下面内容均可参考 USB 协议第 9 章内容(将已附件格式打包)。

```
//******************是SETUP包事件*****************************************************
case USB_EVT_SETUP://是SETUP包事件
    #ifdef Debug_Uart
    USB_SetupStage();
    USB_DirCtrlEP(SetupPacket.bmRequestType.BM.Dir);
    EP0Data.Count = SetupPacket.wLength;//数据长度，USB主设备
    /* wLength域这个域表明第二阶段的数据传输长度。
    switch (SetupPacket.bmRequestType.BM.Type)
    {//SETUP包请求 bmRequestType(D6..5: 种类)
        //=============================================================================
        case REQUEST_STANDARD://bmRequestType(D6..5: 种类)00=标准请求:用于枚举
            #ifdef Debug_Uart
            switch (SetupPacket.bRequest)
            {//bRequest具体请求
            break;
        //=============================================================================
        case REQUEST_CLASS://bmRequestType(D6..5: 种类)01=类  :指定类 的OUT/IN事务
            #ifdef Debug_Uart
            #if USB_CLASS
               switch (SetupPacket.bmRequestType.BM.Recipient)
               {//SetupPacket.bmRequestType(D4..0: 接受者)=1:接口
                class_ok: break;
            #else
                goto stall_i;
            #endif  /* USB_CLASS */
        //=============================================================================
        case REQUEST_VENDOR://:bmRequestType(D6..5: 种类)02=厂商
            #ifdef Debug_Uart
            goto stall_i;
        default:
    stall_i:
        USB_SetStallEP(0x80);
        EP0Data.Count = 0;
        break;
    }//switch (SetupPacket.bmRequestType.BM.Type)
    break;
//*****************是OUT包事件*****************************************************
```

分析：bmRequestType(D6..5: 种类)00=标准请求:用于枚举

01=类 :指定类 的 OUT/IN 事务

02=厂商

C.由于在枚举时几乎都是用标准请求的。所以只是分析 REQUEST_STANDARD 请求

```
{//SETUP包请求 bmRequestType(D6..5: 种类)
    //=========================================================================
    case REQUEST_STANDARD://bmRequestType(D6..5: 种类)00=标准请求:用于枚举
        #ifdef Debug_Uart
        switch (SetupPacket.bRequest)
        {//bRequest具体请求
            case USB_REQUEST_GET_STATUS://获取状态
                #ifdef Debug_Uart
                if (!USB_GetStatus())
                {
                break;
            case USB_REQUEST_CLEAR_FEATURE://清除特性
                #ifdef Debug_Uart
                if (!USB_SetClrFeature(0))
                {
                USB_StatusInStage();
                #if USB_FEATURE_EVENT
                break;
            case USB_REQUEST_SET_FEATURE://设置特性
                #ifdef Debug_Uart
                if (!USB_SetClrFeature(1))
                {
                USB_StatusInStage();
                #if USB_FEATURE_EVENT
                break;
            case USB_REQUEST_SET_ADDRESS://设置地址
                #ifdef Debug_Uart
                switch (SetupPacket.bmRequestType.BM.Recipient)
                {//SetupPacket.bmRequestType(D4..0: 接受者)=0:设备
                break;
            case USB_REQUEST_GET_DESCRIPTOR://获取描述符
                #ifdef Debug_Uart
                if (!USB_GetDescriptor())
                {
                break;
            case USB_REQUEST_SET_DESCRIPTOR://设置描述符
                #ifdef Debug_Uart

o:*/            USB_SetStallEP(0x00);
                EP0Data.Count = 0;
                break;
            case USB_REQUEST_GET_CONFIGURATION://获取配置
                #ifdef Debug_Uart
                switch (SetupPacket.bmRequestType.BM.Recipient)
                {//SetupPacket.bmRequestType(D4..0: 接受者)=0:设备
                break;
            case USB_REQUEST_SET_CONFIGURATION://设置配置
                #ifdef Debug_Uart
                switch (SetupPacket.bmRequestType.BM.Recipient)
                {//SetupPacket.bmRequestType(D4..0: 接受者)=0:设备
                break;
            case USB_REQUEST_GET_INTERFACE://获取接口设置
                #ifdef Debug_Uart
                switch (SetupPacket.bmRequestType.BM.Recipient)
                {//SetupPacket.bmRequestType(D4..0: 接受者)=1:接口
                break;
            case USB_REQUEST_SET_INTERFACE://设置接口
                #ifdef Debug_Uart
                switch (SetupPacket.bmRequestType.BM.Recipient)
                {//SetupPacket.bmRequestType(D4..0: 接受者)=1:接口
                break;
            default:
                goto stall_i;
        }//switch (SetupPacket.bRequest)
        break;
    //=========================================================================
    case REQUEST_CLASS://bmRequestType(D6..5: 种类)01=类  :指定类 的OUT/IN事务
```

下面为第 9 章的标准请求列表，和上面吻合。

表 8-3 标准设备请求

Brequest	Value	备注	
GET_STATUS	0	获取状态	1
CLEAR_FEATURE	1	清除特性	2
为将来保留	2		
SET_FEATURE	3	设置特性	3
为将来保留	4		
SET_ADDRESS	5	设置地址	4
GET_DESCRIPTOR	6	获取描述符	5
SET_DESCRIPTOR	7	设置描述符	6
GET_CONFIGURATION	8	获取配置	7
SET_CONFIGURATION	9	设置配置	8
GET_INTERFACE	10	获取接口设置	9
SET_INTERFACE	11	设置接口置	10
SYNCH_FRAME	12	同步帧	