USB HID 学习实例之如何枚举为键盘

1、基础知识

通过《USB HID 设备类协议入门》一节和上一节的实例我们知道决定 HID 设备“身份”的因素有

1）5 个标准描述符中与 HID 设备有关的部分有：

- 设备描述符中 bDeviceClass、bDeviceSubClass 和 bDeviceProtocol 三个字段的值必须为零。
- 接口描述符中 bInterfaceClass 的值必须为 0x03，bInterfaceSubClass 的值为 0 或 1，为 1 表示 HID 设备符是一个启动设备（Boot Device，一般对 PC 机而言才有意义，意思是 BIOS 启动时能识别并使用您的 HID 设备，且只有标准鼠标或键盘类设备才能成为 Boot Device。 bInterfaceProtocol 的取值含义如下表所示：

HID 接口描述符中 bInterfaceProtocol 的含义	
bInterfaceProtocol 的取值（十进制）	含义
0	NONE
1	鼠标
2	键盘
3~255	保留

2）HID 设备的描述符除了 5 个 USB 的标准描述符（设备描述符、配置描述符、接口描述符、端点描述符、字符串描述符，见百合电子工作室的另一篇文章：USB 开发基础——USB 命令（请求）和 USB 描述符）外，还包括 3 个 HID 设备类特定描述符：HID 描述符、报告描述符、实体描述符。

2、在上一节实例的基础上作一些修改来将 Easy USB 51 Programer 改造成键盘

1）下载上一节实例

2）修改接口描述符中 bInterfaceProtocol 的值为 0x02

在 Descriptor.c 中找到以下代码

```
1.    1,                        //bInterfaceProtocol 为 1 代表鼠标
```

将其修改为

```
2,                                      //bInterfaceProtocol 为 2 代表键盘
```

3）在上一节中也提到“虽然我们将接口描述符中的 bInterfaceProtocol 设为 1（代表鼠标），但这只是针对启动设备（Boot Device）而言有才有效果（即 PC 机的 BIOS 加载后能识别和使用），但真正对 HID 设备的数据流格式进行描述的是报告描述符，所以 bInterfaceProtocol 的取值实际意义不大”，所以现在我们来修改报告描述符

在 Descriptor.c 中找到以下代码：

```
code char MouseReportDescriptor[52] = {
    0x05, 0x01,                     // USAGE_PAGE (Generic Desktop)
    0x09, 0x02,                     // USAGE (Mouse)
    0xa1, 0x01,                     // COLLECTION (Application)
    0x09, 0x01,                     //   USAGE (Pointer)
    0xa1, 0x00,                     //   COLLECTION (Physical)
    0x05, 0x09,                     //     USAGE_PAGE (Button)
    0x19, 0x01,                     //     USAGE_MINIMUM (Button 1)
    0x29, 0x03,                     //     USAGE_MAXIMUM (Button 3)
    0x15, 0x00,                     //     LOGICAL_MINIMUM (0)
    0x25, 0x01,                     //     LOGICAL_MAXIMUM (1)
    0x95, 0x03,                     //     REPORT_COUNT (3)
    0x75, 0x01,                     //     REPORT_SIZE (1)
    0x81, 0x02,                     //     INPUT (Data,Var,Abs)
    0x95, 0x01,                     //     REPORT_COUNT (1)
    0x75, 0x05,                     //     REPORT_SIZE (5)
    0x81, 0x03,                     //     INPUT (Cnst,Var,Abs)
    0x05, 0x01,                     //     USAGE_PAGE (Generic Desktop)
    0x09, 0x30,                     //     USAGE (X)
    0x09, 0x31,                     //     USAGE (Y)
    0x09, 0x38,                     //     USAGE (Wheel)
    0x15, 0x81,                     //     LOGICAL_MINIMUM (-127)
    0x25, 0x7f,                     //     LOGICAL_MAXIMUM (127)
    0x75, 0x08,                     //     REPORT_SIZE (8)
    0x95, 0x03,                     //     REPORT_COUNT (3)
    0x81, 0x06,                     //     INPUT (Data,Var,Rel)
    0xc0,                           //   END_COLLECTION
    0xc0                            // END_COLLECTION
};
```

将其修改为

```
1.    code char MouseReportDescriptor[63] = {
2.        //表示用途页为通用桌面设备
3.        0x05, 0x01,                    // USAGE_PAGE (Generic Desktop)
4.
5.        //表示用途为键盘
6.        0x09, 0x06,                    // USAGE (Keyboard)
7.
8.        //表示应用集合，必须要以 END_COLLECTION 来结束它，见最后的 END_COLLECTION
9.        0xa1, 0x01,                    // COLLECTION (Application)
10.
11.       //表示用途页为按键
12.       0x05, 0x07,                    //   USAGE_PAGE (Keyboard)
13.
14.       //用途最小值，这里为左 ctrl 键
15.       0x19, 0xe0,                    //   USAGE_MINIMUM (Keyboard LeftControl)
16.       //用途最大值，这里为右 GUI 键，即 window 键
17.       0x29, 0xe7,                    //   USAGE_MAXIMUM (Keyboard Right GUI)
18.       //逻辑最小值为 0
19.       0x15, 0x00,                    //   LOGICAL_MINIMUM (0)
20.       //逻辑最大值为 1
21.       0x25, 0x01,                    //   LOGICAL_MAXIMUM (1)
22.       //报告大小（即这个字段的宽度）为 1bit，所以前面的逻辑最小值为 0，逻辑最大值为 1
23.       0x75, 0x01,                    //   REPORT_SIZE (1)
24.       //报告的个数为 8，即总共有 8 个 bits
25.       0x95, 0x08,                    //   REPORT_COUNT (8)
26.       //输入用，变量，值，绝对值。像键盘这类一般报告绝对值，
27.       //而鼠标移动这样的则报告相对值，表示鼠标移动多少
28.       0x81, 0x02,                    //   INPUT (Data,Var,Abs)
29.       //上面这这几项描述了一个输入用的字段，总共为 8 个 bits，每个 bit 表示一个按键
30.       //分别从左 ctrl 键到右 GUI 键。这 8 个 bits 刚好构成一个字节，它位于报告的第一个字节。
31.       //它的最低位，即 bit-0 对应着左 ctrl 键，如果返回的数据该位为 1，则表示左 ctrl 键被按下，
32.       //否则，左 ctrl 键没有按下。最高位，即 bit-7 表示右 GUI 键的按下情况。中间的几个位，
33.       //需要根据 HID 协议中规定的用途页表（HID Usage Tables）来确定。这里通常用来表示
34.       //特殊键，例如 ctrl，shift，del 键等
35.
36.
37.       //这样的数据段个数为 1
38.       0x95, 0x01,                    //   REPORT_COUNT (1)
39.       //每个段长度为 8bits
40.       0x75, 0x08,                    //   REPORT_SIZE (8)
41.       //输入用，常量，值，绝对值
```

```
0x81, 0x03,                    //   INPUT (Cnst,Var,Abs)

//上面这 8 个 bit 是常量，设备必须返回 0


//这样的数据段个数为 5
0x95, 0x05,                    //   REPORT_COUNT (5)
//每个段大小为 1bit
0x75, 0x01,                    //   REPORT_SIZE (1)
//用途是 LED，即用来控制键盘上的 LED 用的，因此下面会说明它是输出用
0x05, 0x08,                    //   USAGE_PAGE (LEDs)
//用途最小值是 Num Lock，即数字键锁定灯
0x19, 0x01,                    //   USAGE_MINIMUM (Num Lock)
//用途最大值是 Kana，这个是什么灯我也不清楚^_^
0x29, 0x05,                    //   USAGE_MAXIMUM (Kana)
//如前面所说，这个字段是输出用的，用来控制 LED。变量，值，绝对值。
//1 表示灯亮，0 表示灯灭
0x91, 0x02,                    //   OUTPUT (Data,Var,Abs)

//这样的数据段个数为 1
0x95, 0x01,                    //   REPORT_COUNT (1)
//每个段大小为 3bits
0x75, 0x03,                    //   REPORT_SIZE (3)
//输出用，常量，值，绝对
0x91, 0x03,                    //   OUTPUT (Cnst,Var,Abs)
//由于要按字节对齐，而前面控制 LED 的只用了 5 个 bit，
//所以后面需要附加 3 个不用 bit，设置为常量。

//报告个数为 6
0x95, 0x06,                    //   REPORT_COUNT (6)
//每个段大小为 8bits
0x75, 0x08,                    //   REPORT_SIZE (8)
//逻辑最小值 0
0x15, 0x00,                    //   LOGICAL_MINIMUM (0)
//逻辑最大值 255
0x25, 0xFF,                    //   LOGICAL_MAXIMUM (255)
//用途页为按键
0x05, 0x07,                    //   USAGE_PAGE (Keyboard)
//使用最小值为 0
0x19, 0x00,                    //   USAGE_MINIMUM (Reserved (no event indicated))
//使用最大值为 0x65
0x29, 0x65,                    //   USAGE_MAXIMUM (Keyboard Application)
//输入用，变量，数组，绝对值
0x81, 0x00,                    //   INPUT (Data,Ary,Abs)
```

```
86.       //以上定义了 6 个 8bit 宽的数组，每个 8bit（即一个字节）用来表示一个按键，所以可以同时
87.       //有 6 个按键按下。没有按键按下时，全部返回 0。如果按下的键太多，导致键盘扫描系统
88.       //无法区分按键时，则全部返回 0x01，即 6 个 0x01。如果有一个键按下，则这 6 个字节中的第一
89.       //个字节为相应的键值（具体的值参看 HID Usage Tables），如果两个键按下，则第 1、2 两个
90.       //字节分别为相应的键值，以次类推。
91.
92.       //关集合，跟上面的对应
93.       0xc0                              // END_COLLECTION
94.   };
```

这们还需要将 Descriptor.h 中的以下代码

```
1.    extern code char MouseReportDescriptor[52];
```

修改为

```
1.    extern code char MouseReportDescriptor[63];
```

上面的报告描述符中只有一个报告，所以没有报告 ID，
因此返回的都是实际使用的数据。总共有 8 字节输入，1 字节输出。其中输入的
第一字节用来表示特殊按键，第二字节保留，后面的六字节为普通按键。如果
只有左 ctrl 键按下，则返回 01 00 00 00 00 00 00 00（十六进制），如果
只有数字键 1 按下，则返回 00 00 59 00 00 00 00 00，如果数字
键 1 和 2 同时按下，则返回 00 00 59 5A 00 00 00 00，如果
再按下左 shift 键，则返回 02 00 59 5A 00 00 00 00，
然后再释放 1 键，则返回 02 00 5A 00 00 00 00 00，
然后全部按键释放，则返回 00 00 00 00 00 00 00 00。
这些数据（即报告）都是通过中断端点返回的。当按下 Num Lock 键时，PC 会发送
输出报告，从报告描述符中我们知道，Num Lock 的 LED 对应着输出报告的最低位，
当数字小键盘打开时，输出 xxxxxxx1（二进制，打 x 的由其它的 LED 状态决定）；
当数字小键盘关闭时，输出 xxxxxxx0（同前）。取出最低位就可以控制数字键锁定 LED 了。

（注：以上说明摘自 computer00 的《USB HID 报告及报告描述符简介》一文）

5）将 Descriptor 中如下语句

```
1.    0x66,0x02,                        //设备制造商定的产品 ID
```

修改为

```
0x66,0x03,                    //设备制造商定的产品 ID
```

4）模拟 NumLock 键盘和 Windows 键

我们定义扩展板 EXT-BOARD-A 上的 K1 键对应 Windows 键（即 USB HID Usage Table 中定义的 GUI 键，Left GUI 或 Right GUI，我们这里就选 Left GUI 吧），而 K2 键盘对应 NumLock 键，当 NumLock 使能时应点亮 NumLock 指示灯，我定义 D0 为 NumLock 指示灯。

根据报告描述符的定义，再参考 USB HID Usage Table，要模拟 Windows 键盘，应将送给主机的 8 个字节的第一个字节置为 0x04，要模拟 NumLock 键，应将第三个字节置为 0x53。如果要模拟 NumLock 指示灯，不应该在主控芯片里判断当到 K2 按下后就打开或熄灭 LED，其正确的方法是：当主机接收到 NumLock 按键信息后，会根据系统当前 NumLock 的状态决定打开还是关闭 LED 指示灯，然后将这一信息通过传给设备（发送一个字节的数据给设置，最低位表示 NumLock 的状态）。

更改 Main.c 文件中的 main 函数为：

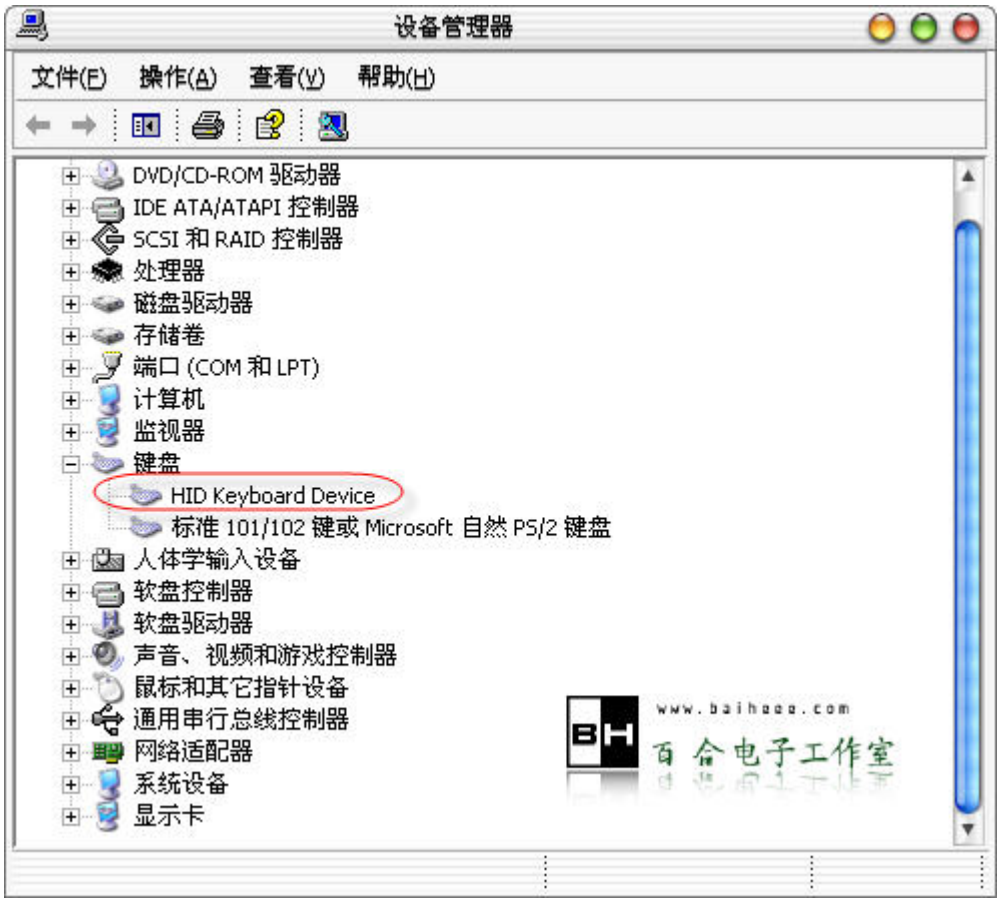
```
void main()
{
    unsigned char i = 0;
    signed char cKeyIn[8];
    static bit bKeyPressed  = 0;        //键按下标志，防止重入

    if (Init_D12()!=0)                  //初始化 D12
        return;                         //如果初始化不成功,返回

    IT0 = 0;                            //外部中断 0 为电平触发方式

    EX0 = 1;                            //开外部中断 0
    PX0 = 0;                            //设置外部中断 0 中断优先级
    EA  = 1;                                //开 80C51 总中断

    P0  = 0;

    while(1)
    {
```

```
20.          usbserve();                        //处理 USB 事件
21.          if(bEPPflags.bits.configuration)
22.          {
23.              //在这里添加端点操作代码
24.              if(bEPPflags.bits.ep2_rxdone )  //主端点接收到数据(从主机发往设备的数据)
25.              {
26.                  bEPPflags.bits.ep2_rxdone       = 0;
27.
28.                  //判断 NumLock 状态
29.                  if(EpBuf[0] & 0x01) //EpBuf 为接收缓冲
30.                  {
31.                      P0  = 0x01;
32.                  }
33.                  else
34.                  {
35.                      P0  = 0x00;
36.                  }
37.
38.              }
39.
40.              K1  = 1;        //P3.5
41.              K2  = 1;        //P3.6
42.
43.              for(i=0;i<100;i++); //延时
44.
45.              if(~K1 & K2)    //K1 按下（模拟左 Windows 键）
46.              {
47.                  if(!bKeyPressed)
48.                  {
49.                      bKeyPressed = 1;
50.
51.                      cKeyIn[0]=0x08;
52.                      cKeyIn[1]=0;            //保留
53.                      cKeyIn[2]=0;
54.                      cKeyIn[3]=0;
55.                      cKeyIn[4]=0;
56.                      cKeyIn[5]=0;
57.                      cKeyIn[6]=0;
58.                      cKeyIn[7]=0;
59.
60.                      D12_WriteEndpoint(5,8,cKeyIn);          //发 8 个字节到 PC 机
61.                  }
62.              }
63.              else if(K1 & ~K2)   //K2 按下（模拟 NumLock 键）
```

```
64.             {
65.                 if(!bKeyPressed)
66.                 {
67.                     bKeyPressed = 1;
68.
69.                     cKeyIn[0]=0;
70.                     cKeyIn[1]=0;            //保留
71.                     cKeyIn[2]=0x53;
72.                     cKeyIn[3]=0;
73.                     cKeyIn[4]=0;
74.                     cKeyIn[5]=0;
75.                     cKeyIn[6]=0;
76.                     cKeyIn[7]=0;
77.
78.                     D12_WriteEndpoint(5,8,cKeyIn);          //发 8 个字节到 PC 机
79.                 }
80.             }
81.             else if(~K1 & ~K2)  //K1 和 K2 同时按下（Window 和 NumLock 同时按下)
82.             {
83.
84.                 if(!bKeyPressed)
85.                 {
86.                     bKeyPressed = 1;
87.
88.                     cKeyIn[0]=0x08;
89.                     cKeyIn[1]=0;            //保留
90.                     cKeyIn[2]=0x53;
91.                     cKeyIn[3]=0;
92.                     cKeyIn[4]=0;
93.                     cKeyIn[5]=0;
94.                     cKeyIn[6]=0;
95.                     cKeyIn[7]=0;
96.
97.                     D12_WriteEndpoint(5,8,cKeyIn);          //发 8 个字节到 PC 机
98.                 }
99.             }
100.            else if(K1 & K2)
101.            {
102.                if(bKeyPressed)
103.                {
104.                    bKeyPressed = 0;
105.
106.                    cKeyIn[0]=0;
107.                    cKeyIn[1]=0;            //保留
```

```
108.                    cKeyIn[2]=0;
109.                    cKeyIn[3]=0;
110.                    cKeyIn[4]=0;
111.                    cKeyIn[5]=0;
112.                    cKeyIn[6]=0;
113.                    cKeyIn[7]=0;
114.
115.                    D12_WriteEndpoint(5,8,cKeyIn);          //发 8 个字节到 PC 机
116.                }
117.            }
118.
119.        }
120.    }
121. }
```

在测试这个例子时，按下 EXT-BOARD-A 上的 K1 键，会弹出开始菜单，按 K2 键，D0 的状态会改变，同时原有键盘上的 NumLock 指示灯也会同 EXT-BOARD-A 上的 D0 状态同步，相反，按原有键盘上的 NumLock 键，D0 的状态也会跟着改变。



下载源代码