小川工作室编写，本书为 LM3S 的 USB 芯片编写，上传的均为草稿，还有没修改，可能还有很多地方不足，希望各位网友原谅！

QQ:2609828265

TEL:15882446438

E-mail:paulhyde@126.com

第五章 HID 设备

5.1 HID 介绍

为简化 USB 设备的开发过程,USB 提出了设备类的概念。所有设备类都必须支持标准 USB 描述符和标准 USB 设备请求。如果有必要，设备类还可以自行定义其专用的描述符和设备请求，这分别被称为设备类定义描述符和设备类定义请求。另外，一个完整的设备类还将指明其接口和端点的使用方法，如接口所包含端点的个数、端点的最大数据包长度等。

HID 设备类就是设备类的一类，HID 是 Human Interface Device 缩写，人机交互设备，例如键盘、鼠标与游戏杆等。不过 HID 设备并不一定要有人机接口，只要符合 HID 类别规范的设备都是 HID 设备。

HID 设备既可以是低速设备也可以是全速设备，其典型的数据传输类型为中断 IN 传输，即它适用于主机接收 USB 设备发来的小量到中等量的数据。HID 具有以下的功能特点：适用于传输少量或中量的数据；传输的数据具有突发性；传输的最大速率有限制；无固定的传输率。

HID 设备类除支持标准 USB 描述符外（设备描述符、配置描述符、接口描述符、端点描述符和字符串描述符），还自行定义了 3 种类描述符，分别为 HID 描述符（主要用于识别 HID 设备所包含的其他类描述符）、报告描述符（提供 HID 设备和主机间交换数据的格式）和物理描述符。一个 HID 设备只能支持一个 HID 描述符；可以支持一个或多个报告描述符；物理描述符是可选的，大多数 HID 设备不需要使用它。

5.2 HID 类描述符

除了标准 USB 描述符外，HID 设备自行定义了三种描述：HID 描述符、报告描述符、物理描述符，物理描述符不常用，不在此讲解。三种描述分别定义为：

```
#define USB_HID_DTYPE_HID         0x21
#define USB_HID_DTYPE_REPORT      0x22
#define USB_HID_DTYPE_PHYSICAL    0x23
```

HID 描述符，提供 HID 设备的相关信息。HID 描述符描述符如下表：

偏移量	域	大小	值	描述
0	bLength	1	数字	字节数。
1	bDescriptorType	1	常量	配置描述符类型
2	bcdHID	2	数字	版本号（BCD 码）

4	bCountryCode	1	数字	国家语言代码
5	bNumDescriptors	1	数字	描述符个数
6	bType	1	索引	下一个描述符类型
7	wLength	2	位图	下一个描述符长度

表 1. HID 描述符符

C 语言 HID 描述符结构体为：

```
typedef struct
{
    //HID 描述符长度
    unsigned char bLength;
    // USB_HID_DTYPE_HID (0x21).
    unsigned char bDescriptorType;

    //HID 协议版本号
    unsigned short bcdHID;
    //国家语言
    unsigned char bCountryCode;
    //描述符个数，至少为 1
    unsigned char bNumDescriptors;
    //下一个描述符类型
    unsigned char bType;
    //下一个描述符长度
    unsigned short wLength;
}
tHIDDescriptor;
```

例如：定义一个实际的 HID 设备描述符。

```
static const tHIDDescriptor g_sKeybHIDDescriptor =
{
    9,                                  // bLength
    USB_HID_DTYPE_HID,                  // bDescriptorType
    0x111,                              // bcdHID (version 1.11 compliant)
    USB_HID_COUNTRY_US,                 // bCountryCode (not localized)
    1,                                  // bNumDescriptors
    USB_HID_DTYPE_REPORT,               // Report descriptor
    sizeof(Report)                      // Size of report descriptor
};
```

国家语言定义：

```
#define USB_HID_COUNTRY_NONE                0x00
#define USB_HID_COUNTRY_ARABIC              0x01
#define USB_HID_COUNTRY_BELGIAN             0x02
#define USB_HID_COUNTRY_CANADA_BI           0x03
#define USB_HID_COUNTRY_CANADA_FR           0x04
#define USB_HID_COUNTRY_CZECH_REPUBLIC      0x05
#define USB_HID_COUNTRY_DANISH              0x06
```

```
#define USB_HID_COUNTRY_FINNISH              0x07
#define USB_HID_COUNTRY_FRENCH               0x08
#define USB_HID_COUNTRY_GERMAN               0x09
#define USB_HID_COUNTRY_GREEK                0x0A
#define USB_HID_COUNTRY_HEBREW               0x0B
#define USB_HID_COUNTRY_HUNGARY              0x0C
#define USB_HID_COUNTRY_INTERNATIONAL_ISO    0x0D
#define USB_HID_COUNTRY_ITALIAN              0x0E
#define USB_HID_COUNTRY_JAPAN_KATAKANA       0x0F
#define USB_HID_COUNTRY_KOREAN               0x10
#define USB_HID_COUNTRY_LATIN_AMERICAN       0x11
#define USB_HID_COUNTRY_NETHERLANDS          0x12
#define USB_HID_COUNTRY_NORWEGIAN            0x13
#define USB_HID_COUNTRY_PERSIAN              0x14
#define USB_HID_COUNTRY_POLAND               0x15
#define USB_HID_COUNTRY_PORTUGUESE           0x16
#define USB_HID_COUNTRY_RUSSIA               0x17
#define USB_HID_COUNTRY_SLOVAKIA             0x18
#define USB_HID_COUNTRY_SPANISH              0x19
#define USB_HID_COUNTRY_SWEDISH              0x1A
#define USB_HID_COUNTRY_SWISS_FRENCH         0x1B
#define USB_HID_COUNTRY_SWISS_GERMAN         0x1C
#define USB_HID_COUNTRY_SWITZERLAND          0x1D
#define USB_HID_COUNTRY_TAIWAN               0x1E
#define USB_HID_COUNTRY_TURKISH_Q            0x1F
#define USB_HID_COUNTRY_UK                   0x20
#define USB_HID_COUNTRY_US                   0x21
#define USB_HID_COUNTRY_YUGOSLAVIA           0x22
#define USB_HID_COUNTRY_TURKISH_F            0x23
```

在中国使用较多的是美国语言 USB_HID_COUNTRY_US。

USB HID 设备是通过报告来给传送数据的，报告有输入报告和输出报告。输入报告（Input）是 USB 设备发送给主机的，例如 USB 鼠标将鼠标移动和鼠标点击等信息返回给电脑，键盘将按键数据数据返回给电脑等；输出报告(Output)是主机发送给 USB 设备的，例如键盘上的数字键盘锁定灯和大写字母锁定灯等。报告是一个数据包，里面包含的是所要传送的数据。输入报告是通过中断输入端点输入的，而输出报告有点区别，当没有中断输出端点时，可以通过控制输出端点 0 发送，当有中断输出端点时，通过中断输出端点发出。

而报告描述符，是描述一个报告以及报告里面的数据是用来干什么用的。通过它，USB HOST 可以分析出报告里面的数据所表示的意思。它通过控制输入端点 0 返回，主机使用获取报告描述符命令来获取报告描述符，注意这个请求是发送到接口的，而不是到设备。一个报告描述符可以描述多个报告，不同的报告通过报告 ID 来识别，报告 ID 在报告最前面，即第一个字节。当报告描述符中没有规定报告 ID 时，报告中就没有 ID 字段，开始就是数据。更详细的说明请参看 USB HID 协议。

下面以一个实例介绍鼠标报告描述符：

```
static const unsigned char g_pucMouseReportDescriptor[]=
{
    UsagePage(USB_HID_GENERIC_DESKTOP),         //通用桌面
    Usage(USB_HID_MOUSE),                       //HID 鼠标
    Collection(USB_HID_APPLICATION),            //应用集合，以 EndCollection 结束
        Usage(USB_HID_POINTER),                 //指针设备
        Collection(USB_HID_PHYSICAL),           //集合
            UsagePage(USB_HID_BUTTONS),         //按键
            UsageMinimum(1),                    //最小值
            UsageMaximum(3),                    //最大值
            LogicalMinimum(0),                  //逻辑最小值
            LogicalMaximum(1),              //逻辑最大值
            ReportSize(1),                  //Report 大小为 1bit
            ReportCount(3),                 //Report 3 个位
            Input(USB_HID_INPUT_DATA | USB_HID_INPUT_VARIABLE |
                  USB_HID_INPUT_ABS),     //发送给主机的报告格式
            //剩余 5 位填满
            ReportSize(5),
            ReportCount(1),
            Input(USB_HID_INPUT_CONSTANT | USB_HID_INPUT_ARRAY |
                  USB_HID_INPUT_ABS),
            UsagePage(USB_HID_GENERIC_DESKTOP),  //通用桌面
            Usage(USB_HID_X),
            Usage(USB_HID_Y),
            LogicalMinimum(-127),
            LogicalMaximum(127),
            ReportSize(8),
            ReportCount(2),
            Input(USB_HID_INPUT_DATA | USB_HID_INPUT_VARIABLE |
                  USB_HID_INPUT_RELATIVE),
            ReportSize(8),
            ReportCount(MOUSE_REPORT_SIZE - 3),
            Input(USB_HID_INPUT_CONSTANT | USB_HID_INPUT_ARRAY |
                  USB_HID_INPUT_ABS),
        EndCollection,
    EndCollection,
};
```

UsagePage 常用参数：

```
#define USB_HID_GENERIC_DESKTOP 0x01
#define USB_HID_BUTTONS         0x09
#define USB_HID_USAGE_POINTER   0x0109
#define USB_HID_USAGE_BUTTONS   0x0509
#define USB_HID_USAGE_LEDS      0x0508
#define USB_HID_USAGE_KEYCODES  0x0507
```

```
#define USB_HID_X               0x30
#define USB_HID_Y               0x31
```

Usage 常用参数：

```
#define USB_HID_X               0x30
#define USB_HID_Y               0x31
#define USB_HID_POINTER         0x01
#define USB_HID_MOUSE           0x02
#define USB_HID_KEYBOARD        0x06
```

Collection 常用参数：

```
#define USB_HID_APPLICATION     0x00
#define USB_HID_PHYSICAL        0x01
```

Input 常用参数：

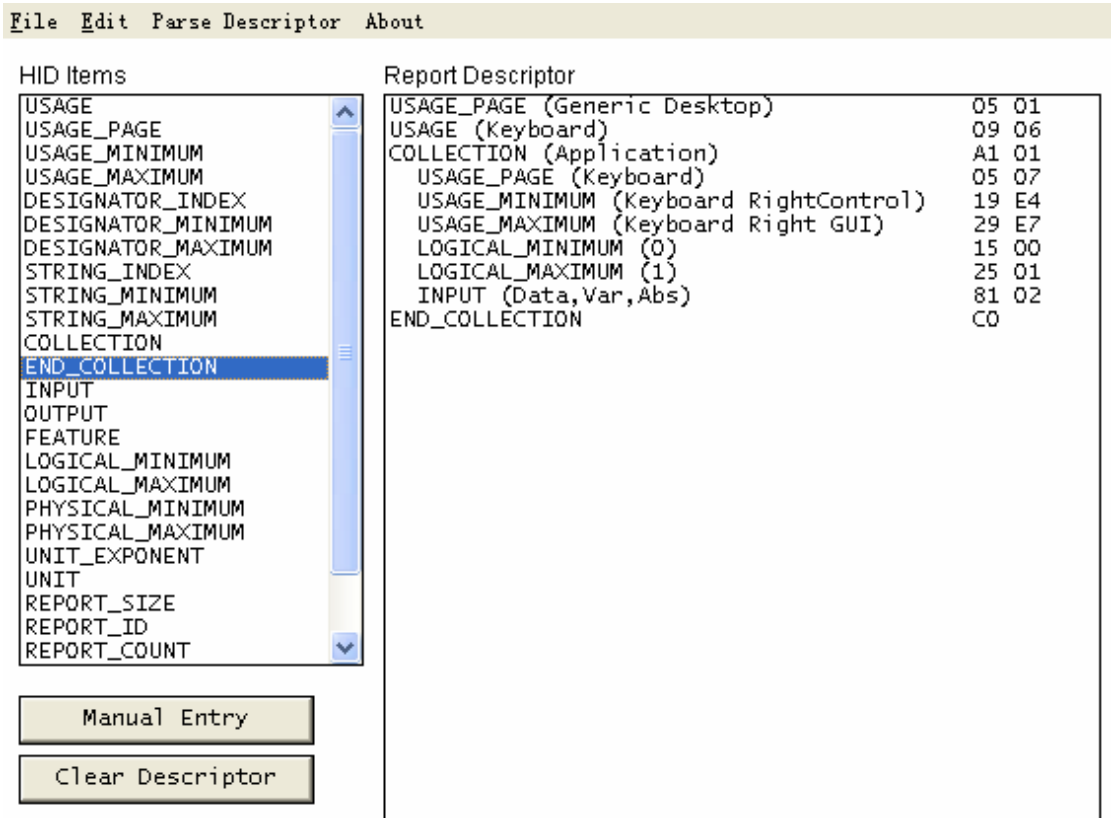
```
#define USB_HID_INPUT_DATA      0x0000
#define USB_HID_INPUT_CONSTANT  0x0001
#define USB_HID_INPUT_ARRAY     0x0000
#define USB_HID_INPUT_VARIABLE  0x0002
#define USB_HID_INPUT_ABS       0x0000
#define USB_HID_INPUT_RELATIVE  0x0004
#define USB_HID_INPUT_NOWRAP    0x0000
#define USB_HID_INPUT_WRAP      0x0008
#define USB_HID_INPUT_LINEAR    0x0000
#define USB_HID_INPUT_NONLINEAR 0x0010
#define USB_HID_INPUT_PREFER    0x0000
#define USB_HID_INPUT_NONPREFER 0x0020
#define USB_HID_INPUT_NONULL    0x0000
#define USB_HID_INPUT_NULL      0x0040
#define USB_HID_INPUT_BITF      0x0100
#define USB_HID_INPUT_BYTES     0x0000
```

Output 常用参数：

```
#define USB_HID_OUTPUT_DATA     0x0000
#define USB_HID_OUTPUT_CONSTANT 0x0001
#define USB_HID_OUTPUT_ARRAY    0x0000
#define USB_HID_OUTPUT_VARIABLE 0x0002
#define USB_HID_OUTPUT_ABS      0x0000
#define USB_HID_OUTPUT_RELATIVE 0x0004
#define USB_HID_OUTPUT_NOWRAP   0x0000
#define USB_HID_OUTPUT_WRAP     0x0008
#define USB_HID_OUTPUT_LINEAR   0x0000
#define USB_HID_OUTPUT_NONLINEAR   x0010
#define USB_HID_OUTPUT_PREFER   0x0000
#define USB_HID_OUTPUT_NONPREFER  0x0020
#define USB_HID_OUTPUT_NONULL   0x0000
#define USB_HID_OUTPUT_NULL     0x0040
#define USB_HID_OUTPUT_BITF     0x0100
```

```
#define USB_HID_OUTPUT_BYTES     0x0000
```

报告描述符使用复杂，可从 Http://www.usb.org 下载 HID Descriptor tool（如图 1 所示）来生成。HID Descriptor tool 是由 USB 官方编写的专用报告符生成工具。有关报告符具体内容请参阅 Http://www.usb.org 网上的 HID Usage Tables 文档。使用 USB 库函数开发 USBHID 设备可以不用考虑报告符，在库中已经定义。



如图 1

HID Descriptor tool 界面简洁，操作方便，集结了所有 HID 设备报告定义，如键盘、鼠标、操作杆、手写设备等，对于报告符不熟悉的开发者相当方便。

5.3 USB 键盘

在 USB 库中已经定义好 USB 键盘的数据类型、API，开发 USB 键盘非常快捷方便。相关定义和数据类型放在“usbdhidkeyb.h”中。

5.3.1 数据类型

usbdhidkeyb.h 中已经定义好 USB 键盘使用的所有数据类型和函数，下面介绍 USB 键盘使用的数据类型。

```
#define KEYB_MAX_CHARS_PER_REPORT       6
```

KEYB_MAX_CHARS_PER_REPORT 定义 USB 键盘一次发送 6 个数据给主机，如果每次发送的数据多于定义的 6 个，将会通过 USBDHIDKeyboardKeyStateChange（）函数将返回发送数据太多的错误：KEYB_ERR_TOO_MANY_KEYS。KEYB_MAX_CHARS_PER_REPORT 这个值的定义由键盘报告决定。

```
typedef enum
{
    //状态还没有定义
   HID_KEYBOARD_STATE_UNCONFIGURED,
```

```
    //空闲状态，没有按键按下或者没有等待数据。
    HID_KEYBOARD_STATE_IDLE,
    //等待主机发送数据
    HID_KEYBOARD_STATE_WAIT_DATA,
   //等待数据发送
HID_KEYBOARD_STATE_SEND
}
tKeyboardState;
```

tKeyboardState，定义键盘状态，用于在 USB 键盘工作时，保存键盘状态。

```
#define KEYB_IN_REPORT_SIZE 8
#define KEYB_OUT_REPORT_SIZE 1
```

KEYB_IN_REPORT_SIZE、KEYB_OUT_REPORT_SIZE 定义键盘 IN 报告符、OUT 报告符长度。

```
typedef struct
{
    // 指示当前 USB 设备是否配置成功。
    unsigned char ucUSBConfigured;
    // USB 键盘使用的子协议：USB_HID_PROTOCOL_BOOT 或者 USB_HID_PROTOCOL_REPORT
    // 将会传给设备描述符和端点描述符
    unsigned char ucProtocol;
    // 键盘 LED 灯当前状态
    volatile unsigned char ucLEDStates;
    // 记录有几个键按下
    unsigned char ucKeyCount;
    // 中断 IN 端点状态.
    volatile tKeyboardState eKeyboardState;
    // 指示当前是否有键状态改变
    volatile tBoolean bChangeMade;
    // 用于接收 OUT 报告
    unsigned char pucDataBuffer[KEYB_OUT_REPORT_SIZE];
    // 用于收送 IN 报告，保存最新一次报告
    unsigned char pucReport[KEYB_IN_REPORT_SIZE];
    // 按下键的 usage 代码
    unsigned char pucKeysPressed[KEYB_MAX_CHARS_PER_REPORT];
    // 为 IN 报告定义时间。
    tHIDReportIdle sReportIdle;
    // HID 设备实例，保存键盘 HID 信息。
    tHIDInstance sHIDInstance;
    // HID 设备驱动
    tUSBDHIDDevice sHIDDevice;
}
tHIDKeyboardInstance;
```

tHIDKeyboardInstance，键盘实例。在 tHIDInstance 和 tUSBDHIDDevice 的基础上，用于保存全部 USB 键盘的配置信息，包括描述符、callback 函数、按键事件等。

```
typedef struct
```

```
{
    //VID
    unsigned short usVID;
   //PID
    unsigned short usPID;
    //设备最大耗电量
    unsigned short usMaxPowermA;
    //电源属性
    unsigned char ucPwrAttributes;
    //函数指针，处理返回事务
    tUSBCallback pfnCallback;
    //Callback 第一个入口参数
    void *pvCBData;
    //指向字符串描述符集合
    const unsigned char * const *ppStringDescriptors;
    //字符串描述符个数 (1 + (5 * (num languages)))
    unsigned long ulNumStringDescriptors;
    //键盘实例，保存 USB 键盘的相关信息。
    tHIDKeyboardInstance *psPrivateHIDKbdData;
}
tUSBDHIDKeyboardDevice;
```

tUSBDHIDKeyboardDevice，USB 键盘类。定义了 VID、PID、电源属性、字符串描述符等，还包括了一个 USB 键盘实例。其它 HID 设备描述符、配置信息通过 API 函数储入 tHIDKeyboardInstance 定义的 USB 键盘实例中。

5.3.2 API 函数

在 USB 键盘 API 库中定义了 7 个函数，完成 USB 键盘初始化、配置及数据处理。下面为 usbdhidkeyb.h 中定义的 API 函数：

```
void *USBDHIDKeyboardInit(unsigned long ulIndex, const tUSBDHIDKeyboardDevice *psDevice);
void *USBDHIDKeyboardCompositeInit(unsigned long ulIndex,
                                   const tUSBDHIDKeyboardDevice *psDevice);
unsigned long USBDHIDKeyboardKeyStateChange(void *pvInstance, unsigned char ucModifiers,
                                            unsigned char ucUsageCode, tBoolean bPressed);
void USBDHIDKeyboardTerm(void *pvInstance);
void *USBDHIDKeyboardSetCBData(void *pvInstance, void *pvCBData);
void USBDHIDKeyboardPowerStatusSet(void *pvInstance, unsigned char ucPower);
tBoolean USBDHIDKeyboardRemoteWakeupRequest(void *pvInstance);
```

```
void *USBDHIDKeyboardInit(unsigned long ulIndex,
                          const tUSBDHIDKeyboardDevice *psDevice);
```

作用：初始化键盘硬件、协议，把其它配置参数填入 psDevice 的键盘实例中。

参数：ulIndex，USB 模块代码，固定值：USB_BASE0。psDevice，USB 键盘类。

返回：指向配置后的 tUSBDHIDKeyboardDevice。

```
void *USBDHIDKeyboardCompositeInit(unsigned long ulIndex,
                                   const tUSBDHIDKeyboardDevice *psDevice);
```

作用：初始化键盘协议，本函数在 USBDHIDKeyboardInit 中已经调用。

参数：ulIndex，USB 模块代码，固定值：USB_BASE0。psDevice，USB 键盘类。

返回：指向配置后的 tUSBDHIDKeyboardDevice。

```
unsigned long USBDHIDKeyboardKeyStateChange(void *pvInstance,
                                            unsigned char ucModifiers,
                                            unsigned char ucUsageCode,
                                            tBoolean bPressed);
```

作用：键盘状态改变，并发送报告给主机。

参数：pvInstance，指向 tUSBDHIDKeyboardDevice，本函数将修改其按键状态等。ucModifiers，功能按键代码。ucUsageCode，普通按键代码。bPressed，是否加入到报告中并发送给主机。

返回：程序错误代码。

```
void USBDHIDKeyboardTerm(void *pvInstance);
```

作用：结束 usb 键盘。

参数：pvInstance，指向 tUSBDHIDKeyboardDevice。

返回：无。

```
void *USBDHIDKeyboardSetCBData(void *pvInstance, void *pvCBData);
```

作用：修改 tUSBDHIDKeyboardDevice 中的 pvCBData 指针。

参数：pvInstance，指向 tUSBDHIDKeyboardDevic。pvCBData，数据指针，用于替换 tUSBDHIDKeyboardDevice 中的 pvCBData 指针。

返回：以前 tUSBDHIDKeyboardDevice 的 pvCBData 的指针。

```
void USBDHIDKeyboardPowerStatusSet(void *pvInstance, unsigned char ucPower);
```

作用：设置键盘电源模式。

参数：pvInstance，指向 tUSBDHIDKeyboardDevic。ucPower，电源工作模式，USB_STATUS_SELF_PWR 或者 USB_STATUS_BUS_PWR。

返回：无。

```
tBoolean USBDHIDKeyboardRemoteWakeupRequest(void *pvInstance);
```

作用：唤醒请求。

参数：pvInstance，指向 tUSBDHIDKeyboardDevic。

返回：是否成功唤醒。

这些 API 中使用最多是 USBDHIDKeyboardInit 和 USBDHIDKeyboardPowerStatusSet 两个函数，在首次使用 USB 键盘时，要初始化设备，使用 USBDHIDKeyboardInit 完成 USB 键盘初始化、打开 USB 中断、枚举设备、描述符补全等；USBDHIDKeyboardPowerStatusSet 设置按键状态，并通过报告发送给主机，这是键盘与主机进行数据通信最主要的接口函数，使用频率最高。

5.3.3 USB 键盘开发

USB 键盘开发只需要 4 步就能完成。如图 2 所示，键盘设备配置（主要是字符串描述符）、callback 函数编写、USB 处理器初始化、按键处理。

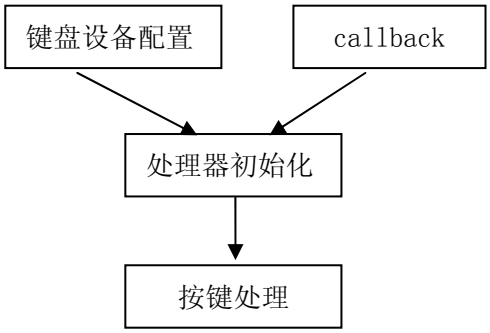


图 2

第一步：键盘设备配置（主要是字符串描述符），按字符串描述符标准完成串描述符配置，进而完成键盘设备配置。

```
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_gpio.h"
#include "inc/hw_types.h"
#include "driverlib/debug.h"
#include "driverlib/gpio.h"
#include "driverlib/interrupt.h"
#include "driverlib/sysctl.h"
#include "driverlib/systick.h"
#include "driverlib/usb.h"
#include "usblib/usblib.h"
#include "usblib/usbhid.h"
#include "usblib/usb-ids.h"
#include "usblib/device/usbdevice.h"
#include "usblib/device/usbdhid.h"
#include "usblib/device/usbdhidkeyb.h"
//声明函数原型
unsigned long KeyboardHandler(void *pvCBData,
                              unsigned long ulEvent,
                              unsigned long ulMsgData,
                              void *pvMsgData);
//*****************************************************************************
// 语言描述符
//*****************************************************************************
const unsigned char g_pLangDescriptor[] =
{
    4,
    USB_DTYPE_STRING,
    USBShort(USB_LANG_EN_US)
};
//*****************************************************************************
```

```
// 制造商 字符串 描述符
//*****************************************************************************
const unsigned char g_pManufacturerString[] =
{
    (17 + 1) * 2,
    USB_DTYPE_STRING,
    'T', 0, 'e', 0, 'x', 0, 'a', 0, 's', 0, ' ', 0, 'I', 0, 'n', 0, 's', 0,
    't', 0, 'r', 0, 'u', 0, 'm', 0, 'e', 0, 'n', 0, 't', 0, 's', 0,
};
//*****************************************************************************
//产品 字符串 描述符
//*****************************************************************************
const unsigned char g_pProductString[] =
{
    (16 + 1) * 2,
    USB_DTYPE_STRING,
    'K', 0, 'e', 0, 'y', 0, 'b', 0, 'o', 0, 'a', 0, 'r', 0, 'd', 0, ' ', 0,
    'E', 0, 'x', 0, 'a', 0, 'm', 0, 'p', 0, 'l', 0, 'e', 0
};
//*****************************************************************************
//   产品 序列号 描述符
//*****************************************************************************
const unsigned char g_pSerialNumberString[] =
{
    (8 + 1) * 2,
    USB_DTYPE_STRING,
    '1', 0, '2', 0, '3', 0, '4', 0, '5', 0, '6', 0, '7', 0, '8', 0
};
//*****************************************************************************
// 设备接口字符串描述符
//*****************************************************************************
const unsigned char g_pHIDInterfaceString[] =
{
    (22 + 1) * 2,
    USB_DTYPE_STRING,
    'H', 0, 'I', 0, 'D', 0, ' ', 0, 'K', 0, 'e', 0, 'y', 0, 'b', 0,
    'o', 0, 'a', 0, 'r', 0, 'd', 0, ' ', 0, 'I', 0, 'n', 0, 't', 0,
    'e', 0, 'r', 0, 'f', 0, 'a', 0, 'c', 0, 'e', 0
};
//*****************************************************************************
//   设备配置字符串描述符
//*****************************************************************************
const unsigned char g_pConfigString[] =
{
```

```
    (26 + 1) * 2,
    USB_DTYPE_STRING,
    'H', 0, 'I', 0, 'D', 0, ' ', 0, 'K', 0, 'e', 0, 'y', 0, 'b', 0,
    'o', 0, 'a', 0, 'r', 0, 'd', 0, ' ', 0, 'C', 0, 'o', 0, 'n', 0,
    'f', 0, 'i', 0, 'g', 0, 'u', 0, 'r', 0, 'a', 0, 't', 0, 'i', 0,
    'o', 0, 'n', 0
};
//*****************************************************************************
// 字符串描述符集合，一定要按这个顺序排列，因为在描述符中已经定义好描述索引。
//*****************************************************************************
const unsigned char * const g_pStringDescriptors[] =
{
    g_pLangDescriptor,
    g_pManufacturerString,
    g_pProductString,
    g_pSerialNumberString,
    g_pHIDInterfaceString,
    g_pConfigString
};
#define NUM_STRING_DESCRIPTORS (sizeof(g_pStringDescriptors) /              \
                                sizeof(unsigned char *))
//*****************************************************************************
//键盘实例，键盘配置并为键盘设备信息提供空间
//*****************************************************************************
tHIDKeyboardInstance g_KeyboardInstance;
//*****************************************************************************
//键盘设备配置
//*****************************************************************************
const tUSBDHIDKeyboardDevice g_sKeyboardDevice =
{
    USB_VID_STELLARIS,          //自行定义VIP.
    USB_PID_KEYBOARD,           //自行定义PID.
    500,
    USB_CONF_ATTR_SELF_PWR | USB_CONF_ATTR_RWAKE,
    KeyboardHandler,
    (void *)&g_sKeyboardDevice,
    g_pStringDescriptors,
    NUM_STRING_DESCRIPTORS,
    &g_KeyboardInstance
};
```

第二步：完成Callback函数。Callback函数用于处理按键事务。可能是主机发出，也可能是状态信息。USB键盘设备中包含了以下事务：USB_EVENT_CONNECTED、USB_EVENT_DISCONNECTED、USBD_HID_KEYB_EVENT_SET_LEDS、USB_EVENT_SUSPEND、USB_EVENT_RESUME、USB_EVENT_TX_COMPLETE。如下表：

名称	说明
USB_EVENT_CONNECTED	USB 设备已经连接到主机
USB_EVENT_DISCONNECTED	USB 设备已经与主机断开
USBD_HID_KEYB_EVENT_SET_LEDS	USB 键盘有 LED 灯设置，查询功能按键再确定 LED
USB_EVENT_SUSPEND	挂起
USB_EVENT_RESUME	唤醒
USB_EVENT_TX_COMPLETE	发送完成

表 2. USB 键盘事务

根据以上事务编写 Callback 函数：

```
unsigned long KeyboardHandler(void *pvCBData, unsigned long ulEvent,
                              unsigned long ulMsgData, void *pvMsgData)
{
    switch (ulEvent)
    {
        case USB_EVENT_CONNECTED:
        {
              //连接成功时，点亮 LED1
              GPIOPinWrite(GPIO_PORTF_BASE,0x10,0x10);
            break;
        }
        case USB_EVENT_DISCONNECTED:
        {
              //断开连接时，LED1 灭
              GPIOPinWrite(GPIO_PORTF_BASE,0x10,0x00);
            break;
        }
        case USB_EVENT_TX_COMPLETE:
        {
              //发送完成时，LED2 亮
              GPIOPinWrite(GPIO_PORTF_BASE,0x20,0x20);
            break;
        }
        case USB_EVENT_SUSPEND:
        {
              //发送完成时，LED2 亮
              GPIOPinWrite(GPIO_PORTF_BASE,0x20,0x0);
            break;
        }
        case USB_EVENT_RESUME:
        {
            break;
        }
        case USBD_HID_KEYB_EVENT_SET_LEDS:
        {
```

```
            //HID_KEYB_CAPS_LOCK 灯
            GPIOPinWrite(GPIO_PORTF_BASE,0x80,((char)ulMsgData &
                                    HID_KEYB_CAPS_LOCK)?0 : 0x80);
        break;
      }
      default:
      {
        break;
      }
    }
    return (0);
}
```

第三步：系统初始化，配置内核电压、系统主频、使能端口、配置按键端口、LED 控制等，本例中使用 4 个按键模拟普通键盘，使用 4 个 LED 进行指示。原理图如图 3 所示：

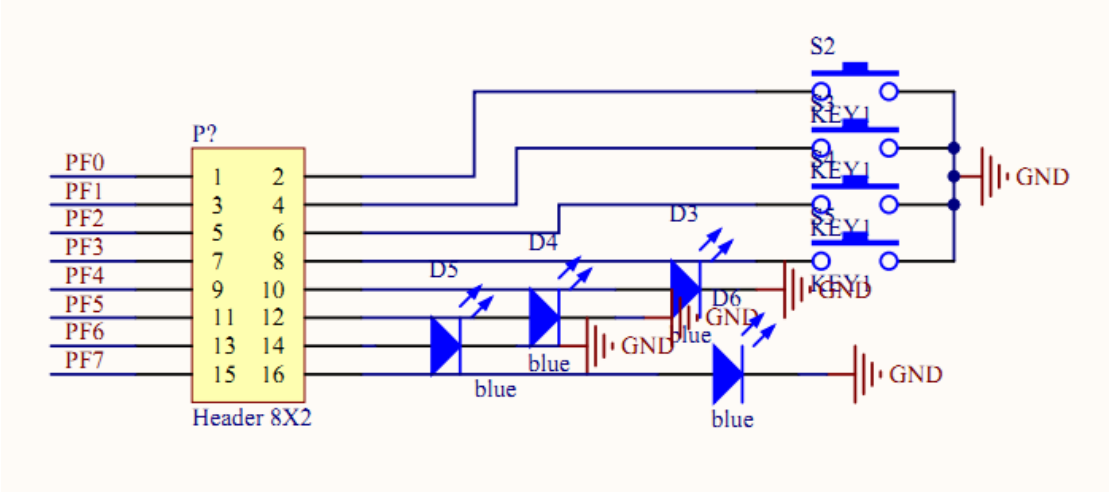


图 3

系统初始化：

```
//设置系统内核电压 与 主频
SysCtlLDOSet(SYSCTL_LDO_2_75V);
SysCtlClockSet(SYSCTL_XTAL_8MHZ | SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL  | SYSCTL_OSC_MAIN );
//使能端口
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE,0xf0);
GPIOPinTypeGPIOInput(GPIO_PORTF_BASE,0x0f);
HWREG(GPIO_PORTF_BASE+GPIO_O_PUR) |= 0x0f;
//初始化键盘设备
USBDHIDKeyboardInit(0, &g_sKeyboardDevice);
```

第四步：按键处理。主要使用 USBDHIDKeyboardPowerStatusSet 设置按键状态，并通过报告发送给主机。

```
while(1)
{

    USBDHIDKeyboardKeyStateChange((void *)&g_sKeyboardDevice, HID_KEYB_CAPS_LOCK,
```

```
                                HID_KEYB_USAGE_A,
                                (GPIOPinRead(GPIO_PORTF_BASE, 0x0f) & GPIO_PIN_0)
                                ? false : true);
        USBDHIDKeyboardKeyStateChange((void *)&g_sKeyboardDevice, 0,
                                HID_KEYB_USAGE_DOWN_ARROW,
                                (GPIOPinRead(GPIO_PORTF_BASE, 0x0f) & GPIO_PIN_1)
                                ? false : true);
        USBDHIDKeyboardKeyStateChange((void *)&g_sKeyboardDevice, 0,
                                HID_KEYB_USAGE_UP_ARROW,
                                (GPIOPinRead(GPIO_PORTF_BASE, 0x0f) & GPIO_PIN_2)
                                ? false : true);
        USBDHIDKeyboardKeyStateChange((void *)&g_sKeyboardDevice, 0,
                                HID_KEYB_USAGE_ESCAPE,
                                (GPIOPinRead(GPIO_PORTF_BASE, 0x0f) & GPIO_PIN_3)
                                ? false : true);
        SysCtlDelay(SysCtlClockGet()/3000);
    }
```

使用上面四步就完成 USB 键盘开发，与普通 USB 键盘没有什么差别。由于在这个例子中使用的是 Demo 开发板，只模拟了四个按键，但功能与普通 USB 键盘一样。USB 键盘开发时要加入两个 lib: usblib.lib 和 DriverLib.lib，在启动代码中加入 USB0DeviceIntHandler 中断服务函数。USB 键盘源码如下：

```
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_gpio.h"
#include "inc/hw_types.h"
#include "driverlib/debug.h"
#include "driverlib/gpio.h"
#include "driverlib/interrupt.h"
#include "driverlib/sysctl.h"
#include "driverlib/systick.h"
#include "driverlib/usb.h"
#include "usblib/usblib.h"
#include "usblib/usbhid.h"
#include "usblib/usb-ids.h"
#include "usblib/device/usbdevice.h"
#include "usblib/device/usbdhid.h"
#include "usblib/device/usbdhidkeyb.h"
//声明函数原型
unsigned long KeyboardHandler(void *pvCBData,
                                unsigned long ulEvent,
                                unsigned long ulMsgData,
                                void *pvMsgData);
//*****************************************************************************
// 语言描述符
```

```
//*****************************************************************************
const unsigned char g_pLangDescriptor[] =
{
    4,
    USB_DTYPE_STRING,
    USBShort(USB_LANG_EN_US)
};
//*****************************************************************************
// 制造商 字符串 描述符
//*****************************************************************************
const unsigned char g_pManufacturerString[] =
{
    (17 + 1) * 2,
    USB_DTYPE_STRING,
    'T', 0, 'e', 0, 'x', 0, 'a', 0, 's', 0, ' ', 0, 'I', 0, 'n', 0, 's', 0,
    't', 0, 'r', 0, 'u', 0, 'm', 0, 'e', 0, 'n', 0, 't', 0, 's', 0,
};
//*****************************************************************************
//产品 字符串 描述符
//*****************************************************************************
const unsigned char g_pProductString[] =
{
    (16 + 1) * 2,
    USB_DTYPE_STRING,
    'K', 0, 'e', 0, 'y', 0, 'b', 0, 'o', 0, 'a', 0, 'r', 0, 'd', 0, ' ', 0,
    'E', 0, 'x', 0, 'a', 0, 'm', 0, 'p', 0, 'l', 0, 'e', 0
};
//*****************************************************************************
//   产品 序列号 描述符
//*****************************************************************************
const unsigned char g_pSerialNumberString[] =
{
    (8 + 1) * 2,
    USB_DTYPE_STRING,
    '1', 0, '2', 0, '3', 0, '4', 0, '5', 0, '6', 0, '7', 0, '8', 0
};
//*****************************************************************************
// 设备接口字符串描述符
//*****************************************************************************
const unsigned char g_pHIDInterfaceString[] =
{
    (22 + 1) * 2,
    USB_DTYPE_STRING,
    'H', 0, 'I', 0, 'D', 0, ' ', 0, 'K', 0, 'e', 0, 'y', 0, 'b', 0,
```

```
    'o', 0, 'a', 0, 'r', 0, 'd', 0, ' ', 0, 'I', 0, 'n', 0, 't', 0,
    'e', 0, 'r', 0, 'f', 0, 'a', 0, 'c', 0, 'e', 0
};
//*****************************************************************************
//   设备配置字符串描述符
//*****************************************************************************
const unsigned char g_pConfigString[] =
{
    (26 + 1) * 2,
    USB_DTYPE_STRING,
    'H', 0, 'I', 0, 'D', 0, ' ', 0, 'K', 0, 'e', 0, 'y', 0, 'b', 0,
    'o', 0, 'a', 0, 'r', 0, 'd', 0, ' ', 0, 'C', 0, 'o', 0, 'n', 0,
    'f', 0, 'i', 0, 'g', 0, 'u', 0, 'r', 0, 'a', 0, 't', 0, 'i', 0,
    'o', 0, 'n', 0
};
//*****************************************************************************
// 字符串描述符集合
//*****************************************************************************
const unsigned char * const g_pStringDescriptors[] =
{
    g_pLangDescriptor,
    g_pManufacturerString,
    g_pProductString,
    g_pSerialNumberString,
    g_pHIDInterfaceString,
    g_pConfigString
};

#define NUM_STRING_DESCRIPTORS (sizeof(g_pStringDescriptors) /              \
                                sizeof(unsigned char *))
//*****************************************************************************
//键盘实例，键盘配置并为键盘设备信息提供空间
//*****************************************************************************
tHIDKeyboardInstance g_KeyboardInstance;
//*****************************************************************************
//键盘设备配置
//*****************************************************************************
const tUSBDHIDKeyboardDevice g_sKeyboardDevice =
{
    USB_VID_STELLARIS,
    USB_PID_KEYBOARD,
    500,
    USB_CONF_ATTR_SELF_PWR | USB_CONF_ATTR_RWAKE,
    KeyboardHandler,
```

```
    (void *)&g_sKeyboardDevice,
    g_pStringDescriptors,
    NUM_STRING_DESCRIPTORS,
    &g_KeyboardInstance
};
//*****************************************************************************
//键盘 callback 函数
//*****************************************************************************
unsigned long KeyboardHandler(void *pvCBData, unsigned long ulEvent,
                              unsigned long ulMsgData, void *pvMsgData)
{
    switch (ulEvent)
    {
        case USB_EVENT_CONNECTED:
        {
              //连接成功时，点亮 LED1
              GPIOPinWrite(GPIO_PORTF_BASE,0x10,0x10);
            break;
        }
        case USB_EVENT_DISCONNECTED:
        {
              //断开连接时，LED1 灭
              GPIOPinWrite(GPIO_PORTF_BASE,0x10,0x00);
            break;
        }
        case USB_EVENT_TX_COMPLETE:
        {
              //发送完成时，LED2 亮
              GPIOPinWrite(GPIO_PORTF_BASE,0x20,0x20);
            break;
        }
        case USB_EVENT_SUSPEND:
        {
              //发送完成时，LED2 亮
              GPIOPinWrite(GPIO_PORTF_BASE,0x20,0x0);
            break;
        }
        case USB_EVENT_RESUME:
        {
            break;
        }
        case USBD_HID_KEYB_EVENT_SET_LEDS:
        {
              //HID_KEYB_CAPS_LOCK 灯
```

```
                GPIOPinWrite(GPIO_PORTF_BASE,0x80,((char)ulMsgData  &  HID_KEYB_CAPS_LOCK)?0  :
0x80);
            break;
        }
        default:
        {
            break;
        }
    }
    return (0);
}
//*****************************************************************************
//主函数
//*****************************************************************************
int main(void)
{
    //系统初始化。
    SysCtlLDOSet(SYSCTL_LDO_2_75V);
    SysCtlClockSet(SYSCTL_XTAL_8MHZ | SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL  | SYSCTL_OSC_MAIN );

    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE,0xf0);
    GPIOPinTypeGPIOInput(GPIO_PORTF_BASE,0x0f);
    HWREG(GPIO_PORTF_BASE+GPIO_O_PUR) |= 0x0f;
    USBDHIDKeyboardInit(0, &g_sKeyboardDevice);
    while(1)
    {

        USBDHIDKeyboardKeyStateChange((void *)&g_sKeyboardDevice, HID_KEYB_CAPS_LOCK,
                                      HID_KEYB_USAGE_A,
                                      (GPIOPinRead(GPIO_PORTF_BASE, 0x0f) & GPIO_PIN_0)
                                      ? false : true);
        USBDHIDKeyboardKeyStateChange((void *)&g_sKeyboardDevice, 0,
                                      HID_KEYB_USAGE_DOWN_ARROW,
                                      (GPIOPinRead(GPIO_PORTF_BASE, 0x0f) & GPIO_PIN_1)
                                      ? false : true);
        USBDHIDKeyboardKeyStateChange((void *)&g_sKeyboardDevice, 0,
                                      HID_KEYB_USAGE_UP_ARROW,
                                      (GPIOPinRead(GPIO_PORTF_BASE, 0x0f) & GPIO_PIN_2)
                                      ? false : true);
        USBDHIDKeyboardKeyStateChange((void *)&g_sKeyboardDevice, 0,
                                      HID_KEYB_USAGE_ESCAPE,
                                      (GPIOPinRead(GPIO_PORTF_BASE, 0x0f) & GPIO_PIN_3)
                                      ? false : true);
```

```
        SysCtlDelay(SysCtlClockGet()/3000);
    }
}
```

5.4 USB 鼠标

在 USB 库中已经定义好 USB 鼠标的数据类型、API，开发 USB 鼠标非常快捷方便。相关定义和数据类型放在“usbdhidmouse.h”中。

5.4.1 数据类型

usbdhidmouse.h 中已经定义好 USB 鼠标使用的所有数据类型和函数，下面介绍 USB 鼠标使用的数据类型。

```
#define MOUSE_REPORT_SIZE       3
```

MOUSE_REPORT_SIZE 定义鼠标 IN 报告的大小，可用于判断 IN 报告是否正确，如果超出 3 个字节会返回错误代码。

```
typedef enum
{
     //状态还没有定义
    HID_MOUSE_STATE_UNCONFIGURED,
    //空闲状态，没有按键按下或者没有等待数据。
    HID_MOUSE_STATE_IDLE,
    //等待主机发送数据
    HID_MOUSE_STATE_WAIT_DATA,
    //等待数据发送
    HID_MOUSE_STATE_SEND
}
tMouseState;
```

tMouseState，定义 USB 鼠标状态，USB 鼠标正常工作时，保存鼠标工作状态。

```
typedef struct
{
    // 指示当前 USB 设备是否配置成功。
    unsigned char ucUSBConfigured;
    // USB 键盘使用的子协议：USB_HID_PROTOCOL_BOOT 或者 USB_HID_PROTOCOL_REPORT
    // 将会传给设备描述符和端点描述符
    unsigned char ucProtocol;
    // 用于收送 IN 报告，保存最新一次报告
    unsigned char pucReport[MOUSE_REPORT_SIZE];
    // 中断 IN 端点状态.
    volatile tMouseState eMouseState;
    // 为 IN 报告定义时间。
    tHIDReportIdle sReportIdle;
     // HID 设备实例，保存键盘 HID 信息。
    tHIDInstance sHIDInstance;
    // HID 设备驱动
    tUSBDHIDDevice sHIDDevice;
}
```

```
tHIDMouseInstance;
```

tHIDMouseInstance，USB 鼠标实例。在 tHIDInstance 和 tUSBDHIDDevice 的基础上，用于保存全部 USB 鼠标的配置信息，包括描述符、callback 函数、按键事件等。

```
typedef struct
{
    //VID
    unsigned short usVID;
   //PID
    unsigned short usPID;
    //设备最大耗电量
    unsigned short usMaxPowermA;
    //电源属性
    unsigned char ucPwrAttributes;
    //函数指针，处理返回事务
    tUSBCallback pfnCallback;
    //Callback 第一个入口参数
    void *pvCBData;
    //指向字符串描述符集合
    const unsigned char * const *ppStringDescriptors;
    //字符串描述符个数 (1 + (5 * (num languages)))
    unsigned long ulNumStringDescriptors;
    //鼠标实例，保存 USB 鼠标的相关信息。
    tHIDMouseInstance *psPrivateHIDMouseData;
}
tUSBDHIDMouseDevice;
```

tUSBDHIDMouseDevice，USB 鼠标类。定义了 VID、PID、电源属性、字符串描述符等，还包括了一个 USB 鼠标实例。其它 HID 设备描述符、配置信息通过 API 函数储入 tHIDMouseInstance 定义的 USB 鼠标实例中。

```
#define MOUSE_ERR_TX_ERROR      2
```

MOUSE_ERR_TX_ERROR，USB 鼠标 API 函数 USBDHIDMouseStateChange 返回的错误代码。

5.4.2 API 函数

在 USB 鼠标 API 库中定义了 7 个函数，完成 USB 键盘初始化、配置及数据处理。下面为 usbdhidkeyb.h 中定义的 API 函数：

```
void *USBDHIDMouseInit(unsigned long ulIndex,
                       const tUSBDHIDMouseDevice *psDevice);
void *USBDHIDMouseCompositeInit(unsigned long ulIndex,
                                const tUSBDHIDMouseDevice *psDevice);
unsigned long USBDHIDMouseStateChange(void *pvInstance, char cDeltaX,
                                      char cDeltaY,
                                      unsigned char ucButtons);
void USBDHIDMouseTerm(void *pvInstance);
void *USBDHIDMouseSetCBData(void *pvInstance, void *pvCBData);
void USBDHIDMousePowerStatusSet(void *pvInstance,
```

```
                                        unsigned char ucPower);
tBoolean USBDHIDMouseRemoteWakeupRequest(void *pvInstance);
```

```
void *USBDHIDMouseInit(unsigned long ulIndex,
                       const tUSBDHIDMouseDevice *psDevice);
```

作用：初始化鼠标硬件、协议，把其它配置参数填入 psDevice 的鼠标实例中。

参数：ulIndex，USB 模块代码，固定值：USB_BASE0。psDevice，USB 鼠标类。

返回：指向配置后的 tUSBDHIDMouseDevice。

```
void *USBDHIDMouseCompositeInit(unsigned long ulIndex,
                                const tUSBDHIDMouseDevice *psDevice);
```

作用：初始化鼠标协议，本函数在 USBDHIDMouseInit 中已经调用。

参数：ulIndex，USB 模块代码，固定值：USB_BASE0。psDevice，USB 鼠标类。

返回：指向配置后的 tUSBDHIDMouseDevice。

```
unsigned long USBDHIDMouseStateChange(void *pvInstance, char cDeltaX,
                                      char cDeltaY,
                                      unsigned char ucButtons);
```

作用：鼠标状态改变，并发送报告给主机。

参数：pvInstance，指向 tUSBDHIDMouseDevice，本函数将修改其 X、Y、按键状态等。cDeltaX，X 值。cDeltaY，Y 值。ucButtons，鼠标按键。

返回：程序错误代码。

```
void USBDHIDMouseTerm(void *pvInstance);
```

作用：结束 usb 鼠标。

参数：pvInstance，指向 tUSBDHIDMouseDevice。

返回：无。

```
void *USBDHIDMouseSetCBData(void *pvInstance, void *pvCBData);
```

作用：修改 tUSBDHIDMouseDevice 中的 pvCBData 指针。

参数：pvInstance，指向 tUSBDHIDMouseDevice。pvCBData，数据指针，用于替换 tUSBDHIDMouseDevice 中的 pvCBData 指针。

返回：以前 tUSBDHIDMouseDevice 的 pvCBData 的指针。

```
void USBDHIDMousePowerStatusSet(void *pvInstance,
                                unsigned char ucPower);
```

作用：设置鼠标电源模式。

参数：pvInstance，指向 tUSBDHIDMouseDevice。ucPower，电源工作模式，USB_STATUS_SELF_PWR 或者 USB_STATUS_BUS_PWR。

返回：无。

```
tBoolean USBDHIDMouseRemoteWakeupRequest(void *pvInstance);
```

作用：唤醒请求。

参数：pvInstance，指向 tUSBDHIDMouseDevice。

返回：是否成功唤醒。

这些 API 中使用最多是 USBDHIDMouseInit 和 USBDHIDMouseStateChange 两个函数，在首次使用 USB 鼠标时，要初始化 USB 设备，使用 USBDHIDMouseInit 完成 USB 鼠标初始化、打开 USB 中断、枚举设备、描述符补全等；USBDHIDMouseStateChange 设置鼠标 X、Y、按键状态，并通过 IN 报告发送给主机，这是 USB 鼠标与主机进行数据通信最主要的接口函数，使用频率最高。

5.4.3 USB 鼠标开发

USB 鼠标开发只需要 4 步就能完成。如图 2 所示,鼠标设备配置(主要是字符串描述符)、callback 函数编写、USB 处理器初始化、X\Y\按键处理。

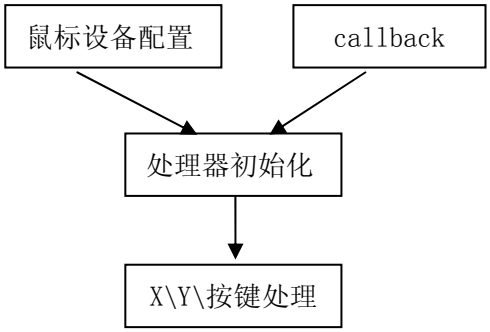


图 2

第一步：鼠标设备配置（主要是字符串描述符），按字符串描述符标准完成串描述符配置，进而完成鼠标设备配置。

```
#include "inc/hw_types.h"
#include "driverlib/usb.h"
#include "usblib/usblib.h"
#include "usblib/usbhid.h"
#include "usblib/usb-ids.h"
#include "usblib/device/usbdevice.h"
#include "usblib/device/usbdhid.h"
#include "usblib/device/usbdhidmouse.h"
#include "usb_mouse_structs.h"
//声明函数原型
unsigned long MouseHandler(void *pvCBData,
                           unsigned long ulEvent,
                           unsigned long ulMsgData,
                           void *pvMsgData);
//*****************************************************************************
// 语言描述符
//*****************************************************************************
const unsigned char g_pLangDescriptor[] =
{
    4,
    USB_DTYPE_STRING,
    USBShort(USB_LANG_EN_US)
};
//*****************************************************************************
// 制造商 字符串 描述符
//*****************************************************************************
const unsigned char g_pManufacturerString[] =
```

```
{
    (17 + 1) * 2,
    USB_DTYPE_STRING,
    'T', 0, 'e', 0, 'x', 0, 'a', 0, 's', 0, ' ', 0, 'I', 0, 'n', 0, 's', 0,
    't', 0, 'r', 0, 'u', 0, 'm', 0, 'e', 0, 'n', 0, 't', 0, 's', 0,
};
//*****************************************************************************
//产品 字符串 描述符
//*****************************************************************************
const unsigned char g_pProductString[] =
{
    (13 + 1) * 2,
    USB_DTYPE_STRING,
    'M', 0, 'o', 0, 'u', 0, 's', 0, 'e', 0, ' ', 0, 'E', 0, 'x', 0, 'a', 0,
    'm', 0, 'p', 0, 'l', 0, 'e', 0
};
//*****************************************************************************
//  产品 序列号 描述符
//*****************************************************************************
const unsigned char g_pSerialNumberString[] =
{
    (8 + 1) * 2,
    USB_DTYPE_STRING,
    '1', 0, '2', 0, '3', 0, '4', 0, '5', 0, '6', 0, '7', 0, '8', 0
};
//*****************************************************************************
// 设备接口字符串描述符
//*****************************************************************************
const unsigned char g_pHIDInterfaceString[] =
{
    (19 + 1) * 2,
    USB_DTYPE_STRING,
    'H', 0, 'I', 0, 'D', 0, ' ', 0, 'M', 0, 'o', 0, 'u', 0, 's', 0,
    'e', 0, ' ', 0, 'I', 0, 'n', 0, 't', 0, 'e', 0, 'r', 0, 'f', 0,
    'a', 0, 'c', 0, 'e', 0
};
//*****************************************************************************
//  设备配置字符串描述符
//*****************************************************************************
const unsigned char g_pConfigString[] =
{
    (23 + 1) * 2,
    USB_DTYPE_STRING,
    'H', 0, 'I', 0, 'D', 0, ' ', 0, 'M', 0, 'o', 0, 'u', 0, 's', 0,
```

```
    'e', 0, ' ', 0, 'C', 0, 'o', 0, 'n', 0, 'f', 0, 'i', 0, 'g', 0,
    'u', 0, 'r', 0, 'a', 0, 't', 0, 'i', 0, 'o', 0, 'n', 0
};
//*****************************************************************************
// 字符串描述符集合
//*****************************************************************************
const unsigned char * const g_pStringDescriptors[] =
{
    g_pLangDescriptor,
    g_pManufacturerString,
    g_pProductString,
    g_pSerialNumberString,
    g_pHIDInterfaceString,
    g_pConfigString
};

#define NUM_STRING_DESCRIPTORS (sizeof(g_pStringDescriptors) /              \
                                sizeof(unsigned char *))
//*****************************************************************************
//鼠标实例，鼠标配置并为鼠标设备信息提供空间
//*****************************************************************************
tHIDMouseInstance g_sMouseInstance;
//*****************************************************************************
//鼠标设备配置
//*****************************************************************************
const tUSBDHIDMouseDevice g_sMouseDevice =
{
    USB_VID_STELLARIS,      //开发者自己定义
    USB_PID_MOUSE,          //开发者自己定义
    500,
    USB_CONF_ATTR_SELF_PWR,
    MouseHandler,
    (void *)&g_sMouseDevice,
    g_pStringDescriptors,
    NUM_STRING_DESCRIPTORS,
    &g_sMouseInstance
};
```

第二步：完成 Callback 函数。Callback 函数用于处理 X、Y、按键事务。可能是主机发出，也可能是状态信息。USB 鼠标设备中包含了以下事务：USB_EVENT_CONNECTED、USB_EVENT_DISCONNECTED 、 USB_EVENT_SUSPEND 、 USB_EVENT_RESUME 、 USB_EVENT_TX_COMPLETE。如下表：

名称	说明
USB_EVENT_CONNECTED	USB 设备已经连接到主机
USB_EVENT_DISCONNECTED	USB 设备已经与主机断开

USB_EVENT_SUSPEND	挂起
USB_EVENT_RESUME	唤醒
USB_EVENT_TX_COMPLETE	发送完成

表 2. USB 鼠标事务

根据以上事务编写 Callback 函数：

```
unsigned long MouseHandler(void *pvCBData, unsigned long ulEvent,
                           unsigned long ulMsgData, void *pvMsgData)
{
    switch (ulEvent)
    {
        case USB_EVENT_CONNECTED:
        {
              //连接成功时，点亮 LED1
              GPIOPinWrite(GPIO_PORTF_BASE,0x10,0x10);
            break;
        }
        case USB_EVENT_DISCONNECTED:
        {
              //断开连接时，LED1 灭
              GPIOPinWrite(GPIO_PORTF_BASE,0x10,0x00);
            break;
        }
        case USB_EVENT_TX_COMPLETE:
        {
              //发送完成时，LED2 亮
              GPIOPinWrite(GPIO_PORTF_BASE,0x20,0x20);
            break;
        }
        case USB_EVENT_SUSPEND:
        {
              //发送完成时，LED2 亮
              GPIOPinWrite(GPIO_PORTF_BASE,0x20,0x0);
            break;
        }
        case USB_EVENT_RESUME:
        {
            break;
        }
        default:
        {
            break;
        }
    }
    return (0);
```

```
}
```

第三步：系统初始化，配置内核电压、系统主频、使能端口、配置按键端口、LED 控制等，本例中使用 4 个按键控制鼠标移动，使用 4 个 LED 进行指示动作。原理图如图 3 所示：

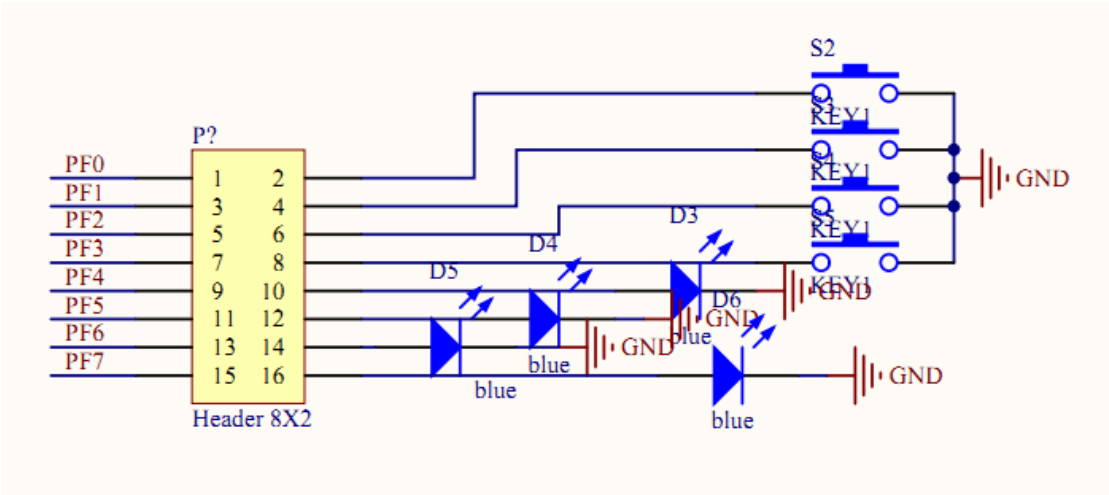


图 3

系统初始化：

```
//设置系统内核电压 与 主频
SysCtlLDOSet(SYSCTL_LDO_2_75V);
SysCtlClockSet(SYSCTL_XTAL_8MHZ | SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL  | SYSCTL_OSC_MAIN );
//使能端口
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE,0xf0);
GPIOPinTypeGPIOInput(GPIO_PORTF_BASE,0x0f);
HWREG(GPIO_PORTF_BASE+GPIO_O_PUR) |= 0x0f;
//初始化鼠标设备
USBDHIDMouseInit (0, &g_sMouseDevice);
```

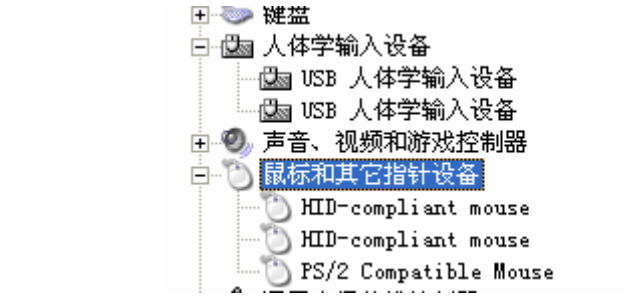
第四步：X、Y、按键处理。主要使用 USBDHIDMouseStateChange 设置 X、Y、按键状态，并通过报告发送给主机。

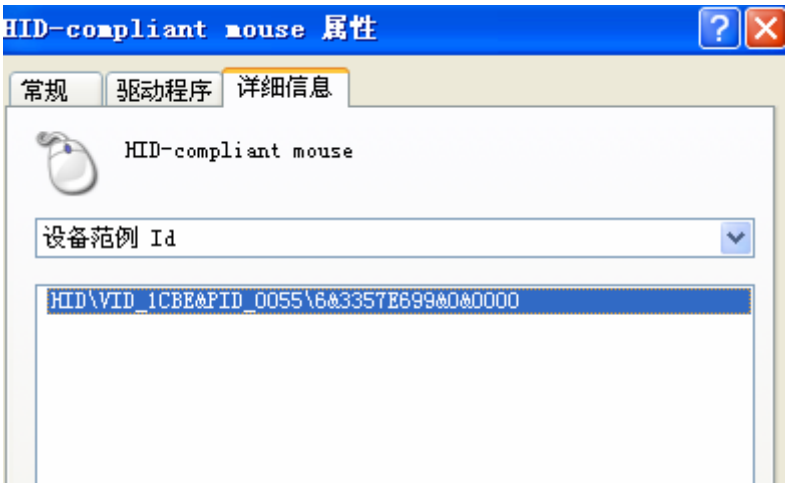
```
while(1)
{
    ulTemp = (~GPIOPinRead(GPIO_PORTF_BASE, 0x0f)) & 0x0f;
    switch(ulTemp)
    {
        case 0x01:
        x = x + 1;
        key = 0;
        break;
        case 0x02:
        x = x - 1 ;
        key = 0;
        break;
        case 0x04:
        y = y + 1;
```

```
                key = 0;
                break;
                case 0x08:
                y = y - 1;
                key = 0;
                break;
                case 0x03:
                key = 1;
                break;
                case 0x0c:
                key = 2;
                break;
                case 0x09:
                key = 4;
                break;
                default:
                key = 0;
                break;
            }
            if(ulTemp)
            USBDHIDMouseStateChange((void *)&g_sMouseDevice,x,y,key);
          SysCtlDelay(SysCtlClockGet()/30);
    }
```

使用上面四步就完成 USB 鼠标开发，与普通 USB 鼠标一样操作。由于在这个例子中使用的是 Demo 开发板，只能用四个按键，模拟鼠标移动。USB 鼠标开发时也要加入两个 lib: usblib.lib 和 DriverLib.lib，在启动代码中加入 USB0DeviceIntHandler 中断服务函数。程序运行进如下图：

从图中可以看出 USB 鼠标枚举成功，在“设备管理器”中可以看到“USB 人体学输入设备”，而且可以“鼠标和其它指针设备”中找到“HID-compliant mouse”，如下图。其中有一个就是上面开发的 USB 鼠标，查看“属性”可以看到下图：其中 VID 和 PID 都是之前配置的。





USB 鼠标源码如下：

```
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_gpio.h"
#include "inc/hw_types.h"
#include "driverlib/debug.h"
#include "driverlib/gpio.h"
#include "inc/hw_types.h"
#include "driverlib/usb.h"
#include "inc/hw_sysctl.h"
#include "driverlib/sysctl.h"
#include "usblib/usblib.h"
#include "usblib/usbhid.h"
#include "usblib/usb-ids.h"
#include "usblib/device/usbdevice.h"
#include "usblib/device/usbdhid.h"
#include "usblib/device/usbdhidmouse.h"
//声明函数原型
unsigned long MouseHandler(void *pvCBData,
                           unsigned long ulEvent,
                           unsigned long ulMsgData,
                           void *pvMsgData);
//*****************************************************************************
// 语言描述符
//*****************************************************************************
```

```
const unsigned char g_pLangDescriptor[] =
{
    4,
    USB_DTYPE_STRING,
    USBShort(USB_LANG_EN_US)
};
//*****************************************************************************
// 制造商 字符串 描述符
//*****************************************************************************
const unsigned char g_pManufacturerString[] =
{
    (17 + 1) * 2,
    USB_DTYPE_STRING,
    'T', 0, 'e', 0, 'x', 0, 'a', 0, 's', 0, ' ', 0, 'I', 0, 'n', 0, 's', 0,
    't', 0, 'r', 0, 'u', 0, 'm', 0, 'e', 0, 'n', 0, 't', 0, 's', 0,
};
//*****************************************************************************
//产品 字符串 描述符
//*****************************************************************************
const unsigned char g_pProductString[] =
{
    (13 + 1) * 2,
    USB_DTYPE_STRING,
    'M', 0, 'o', 0, 'u', 0, 's', 0, 'e', 0, ' ', 0, 'E', 0, 'x', 0, 'a', 0,
    'm', 0, 'p', 0, 'l', 0, 'e', 0
};
//*****************************************************************************
//   产品 序列号 描述符
//*****************************************************************************
const unsigned char g_pSerialNumberString[] =
{
    (8 + 1) * 2,
    USB_DTYPE_STRING,
    '1', 0, '2', 0, '3', 0, '4', 0, '5', 0, '6', 0, '7', 0, '8', 0
};
//*****************************************************************************
// 设备接口字符串描述符
//*****************************************************************************
const unsigned char g_pHIDInterfaceString[] =
{
    (19 + 1) * 2,
    USB_DTYPE_STRING,
    'H', 0, 'I', 0, 'D', 0, ' ', 0, 'M', 0, 'o', 0, 'u', 0, 's', 0,
    'e', 0, ' ', 0, 'I', 0, 'n', 0, 't', 0, 'e', 0, 'r', 0, 'f', 0,
```

```
    'a', 0, 'c', 0, 'e', 0
};
//*****************************************************************************
//   设备配置字符串描述符
//*****************************************************************************
const unsigned char g_pConfigString[] =
{
    (23 + 1) * 2,
    USB_DTYPE_STRING,
    'H', 0, 'I', 0, 'D', 0, ' ', 0, 'M', 0, 'o', 0, 'u', 0, 's', 0,
    'e', 0, ' ', 0, 'C', 0, 'o', 0, 'n', 0, 'f', 0, 'i', 0, 'g', 0,
    'u', 0, 'r', 0, 'a', 0, 't', 0, 'i', 0, 'o', 0, 'n', 0
};
//*****************************************************************************
// 字符串描述符集合
//*****************************************************************************
const unsigned char * const g_pStringDescriptors[] =
{
    g_pLangDescriptor,
    g_pManufacturerString,
    g_pProductString,
    g_pSerialNumberString,
    g_pHIDInterfaceString,
    g_pConfigString
};
#define NUM_STRING_DESCRIPTORS (sizeof(g_pStringDescriptors) /                \
                                sizeof(unsigned char *))
//*****************************************************************************
//键盘实例，键盘配置并为键盘设备信息提供空间
//*****************************************************************************
tHIDMouseInstance g_sMouseInstance;
//*****************************************************************************
//键盘设备配置
//*****************************************************************************
const tUSBDHIDMouseDevice g_sMouseDevice =
{
    USB_VID_STELLARIS,
    USB_PID_MOUSE,
    500,
    USB_CONF_ATTR_SELF_PWR,
    MouseHandler,
    (void *)&g_sMouseDevice,
    g_pStringDescriptors,
    NUM_STRING_DESCRIPTORS,
```

```
    &g_sMouseInstance
};
//*****************************************************************************
//键盘 callback 函数
//*****************************************************************************
unsigned long MouseHandler(void *pvCBData, unsigned long ulEvent,
                           unsigned long ulMsgData, void *pvMsgData)
{
    switch (ulEvent)
    {
        case USB_EVENT_CONNECTED:
        {
              //连接成功时，点亮 LED1
              GPIOPinWrite(GPIO_PORTF_BASE,0x10,0x10);
            break;
        }
        case USB_EVENT_DISCONNECTED:
        {
              //断开连接时，LED1 灭
              GPIOPinWrite(GPIO_PORTF_BASE,0x10,0x00);
            break;
        }
        case USB_EVENT_TX_COMPLETE:
        {
              //发送完成时，LED2 亮
              GPIOPinWrite(GPIO_PORTF_BASE,0x20,0x20);
            break;
        }
        case USB_EVENT_SUSPEND:
        {
              //发送完成时，LED2 亮
              GPIOPinWrite(GPIO_PORTF_BASE,0x20,0x0);
            break;
        }
        case USB_EVENT_RESUME:
        {
            break;
        }
        default:
        {
            break;
        }
    }
    return (0);
```

```
}
//*****************************************************************************
//主函数
//*****************************************************************************
int main(void)
{
    //系统初始化。
    unsigned long x=0,y=0,key=0,ulTemp=0;
    SysCtlLDOSet(SYSCTL_LDO_2_75V);
    SysCtlClockSet(SYSCTL_XTAL_8MHZ  |  SYSCTL_SYSDIV_4  |  SYSCTL_USE_PLL  |
SYSCTL_OSC_MAIN );
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE,0xf0);
    GPIOPinTypeGPIOInput(GPIO_PORTF_BASE,0x0f);
    HWREG(GPIO_PORTF_BASE+GPIO_O_PUR) |= 0x0f;
    USBDHIDMouseInit(0, &g_sMouseDevice);
    while(1)
    {
        ulTemp = (~GPIOPinRead(GPIO_PORTF_BASE, 0x0f)) & 0x0f;
        switch(ulTemp)
        {
            case 0x01:
            x = x + 1;
            key = 0;
            break;
            case 0x02:
            x = x - 1 ;
            key = 0;
            break;
            case 0x04:
            y = y + 1;
            key = 0;
            break;
            case 0x08:
            y = y - 1;
            key = 0;
            break;
            case 0x03:
            key = 1;
            break;
            case 0x0c:
            key = 2;
            break;
            case 0x09:
```

```
                key = 4;
                break;
                default:
                key = 0;
                break;
            }
            if(ulTemp)
            USBDHIDMouseStateChange((void *)&g_sMouseDevice,x,y,key);
          SysCtlDelay(SysCtlClockGet()/30);
        }
    }
```

5.5 小结

经过本章节介绍，读者对 HID 设备有初步了解，如果想了解更深层次的 HID 设备类，可能参考官方数据手册。本章主要介绍了 HID 设备类的结构与编程、USB 库函数编程、USB 键盘开发与 USB 鼠标开发。当然这些代码都是简单的实现功能，真正的产口还需要在这基础之上进一步完善与优化。