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

第四章 USB 库介绍

4.1 USB 库函数简介

Luminary Micro 公司提供 USB 处理器的 USB 库函数，应用在 Stellaris 处理器上，为 USB 设备、USB 主机、OTG 开发提供 USB 协议框架和 API 函数，适用于多种开发环境：Keil、CSS、IAR、CRT、CCS 等。本书中的所有例程都在 Keil uv4 中编译。

使用 USB 库开发时，要加入两个已经编译好的.lib。KEIL 中建立 USB 开发工程结构如图 1 所示：

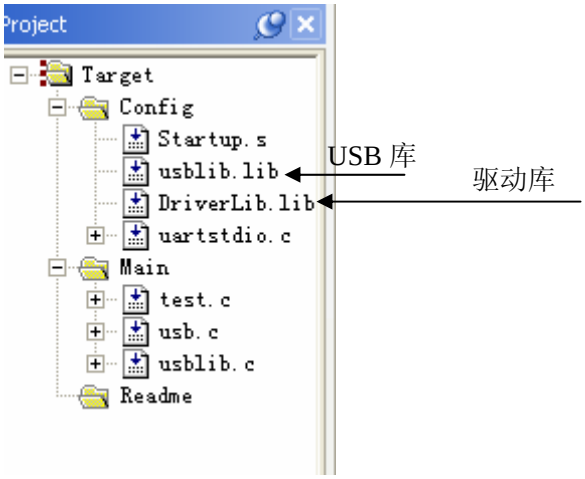

图 1 文件组织结构

在使用 USB 库之前必须了解 USB 库的结构，有助于开发者其理解与使用。USB 库分为三个层次：USB 设备 API、设备类驱动、设备类，如图 2 USB 库架构：

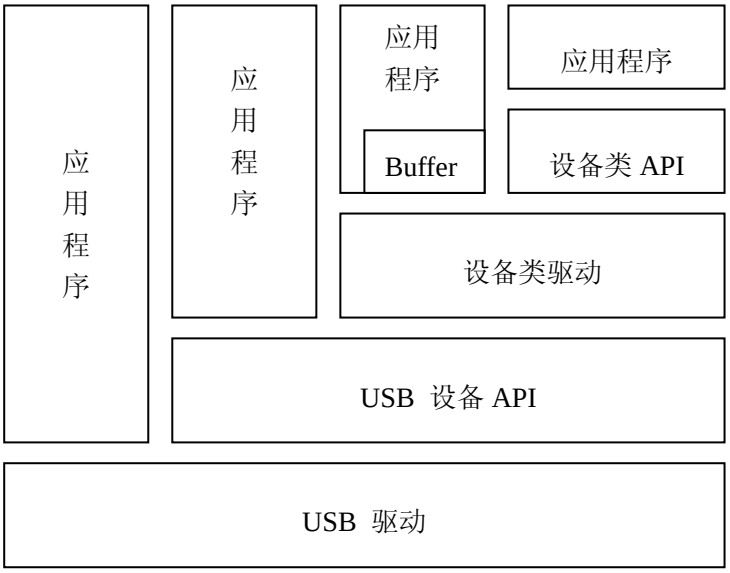

图 2 USB 库架构

从图 2 中可以看出，最底层驱动是第三章讲的 USB 驱动程序，只使用 USB 驱动程序可以进行简单的 USB 开发。对于更为复杂的 USB 工程，仅仅使用驱动程序开发是很困难的。在引入 USB 库后，可以很方便、简单进行复杂的 USB 工程设计。USB 库提供三层 API，底层为 USB 设备 API，提供最基础的 USB 协议和类定义；USB 设备驱动是在 USB 设备 API 基础上扩展的 USB 各种设备驱动，比如 HID 类、CDC 类等类驱动；为了更方便程序员使用，还提供设备类 API，扩展 USB 库的使用范围，进一步减轻开发人员的负担，在不用考虑更底层驱动情况下完成 USB 工程开发。

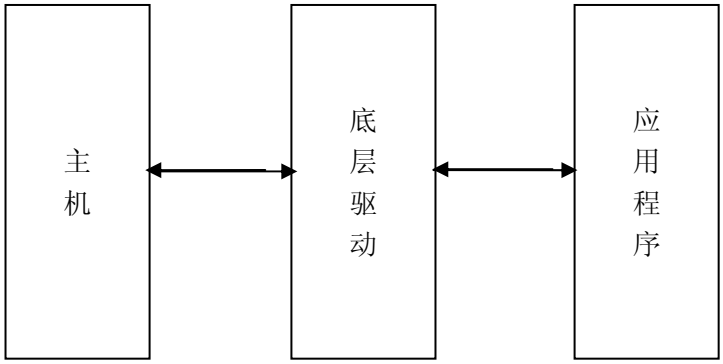

图 3

同时开发人员可有多种选择，开发 USB 设备。如图 3，开发人员可以使用最底层的 API 驱动函数进行开发，应用程序通过底层驱动与 USB 主机通信、控制。但要求开发人员对 USB 协议完全了解，并熟练于协议编写。

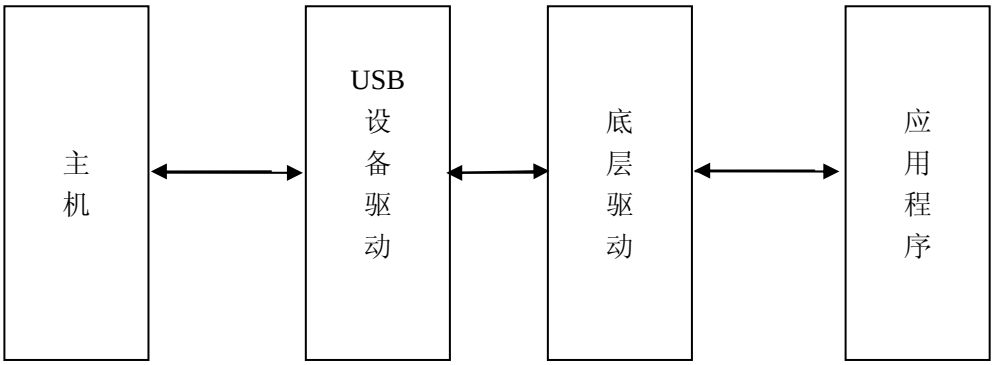

图 4

如图 4，开发人员可以在最底层的 API 驱动函数基础上使用 USB 库函数的 USB 设备驱动函数，进行 USB 控制，应用程序通过底层驱动和 USB 设备驱动与 USB 主机通信、控制。减轻了开发人员的负担。

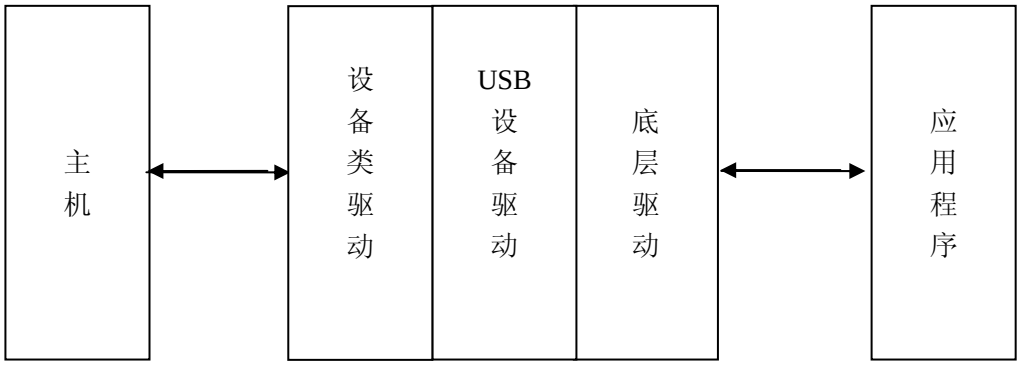

图 5

如图 5，开发人员可以利用最底层的 API 驱动函数、USB 设备驱动函数和设备类驱动函数，进行 USB 开发。设备类驱动主要提供各种 USB 设备类的驱动，比如 Audio 类驱动、HID 类驱动、Composite 类驱动、CDC 类驱动、Bulk 类驱动、Mass Storage 类驱动等 6 种基本类驱动，其它设备类驱动可以参考这 5 种驱动的源码，由开发者自己编写。

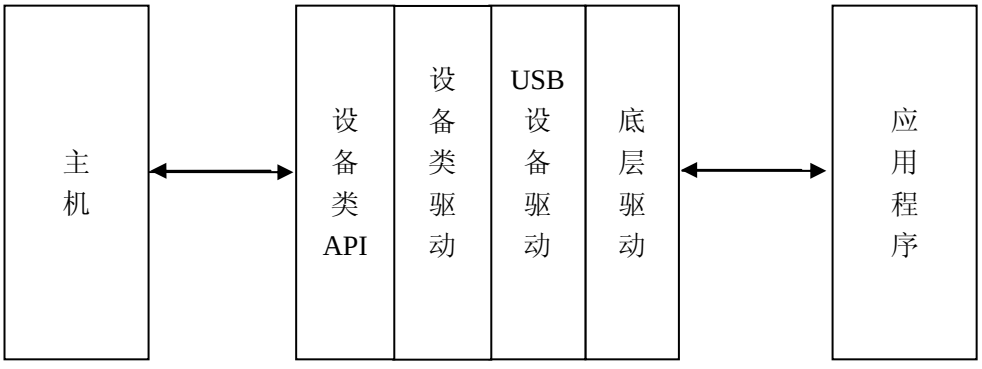

图 6

如图 6，开发人员可以利用最底层的 API 驱动函数、USB 设备驱动函数、设备类驱动函数和设备类 API，进行 USB 开发。设备类 API 主要提供各种 USB 设备类操作相关的函数，比

如 HID 中的键盘、鼠标操作接口。设备类 API 也只提供了 5 种 USB 设备类 API，其它不常用的需要开发者自己编写。

Luminary Micro 公司提供 USB 函数库支持多种使用方法，完全能够满足 USB 产品开发，并且使用方便、快捷。

4.2 使用底层驱动开发

使用最底层 USB 驱动开发，要求开发人员对 USB 协议及相关事务彻底了解，开发 USB 产品有一定难度，但是这种开发模式占用内存少，运行效率高。但有容易出现 bug。

例如，使用底层 USB 驱动开发，开发一个音频设备。

第一：初始化 usb 处理器，包括内核电压、CPU 主频、USB 外设资源等。

```
SysCtlLDOSet(SYSCTL_LDO_2_75V);
// 主频 50MHz
SysCtlClockSet(SYSCTL_XTAL_8MHZ | SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL  | SYSCTL_OSC_MAIN );
//打开 USB 外设
SysCtlPeripheralEnable(SYSCTL_PERIPH_USB0);
//打开 USB 主时钟
SysCtlUSBPLLEnable();
//清除中断标志，并重新打开中断
USBIntStatusControl(USB0_BASE);
USBIntStatusEndpoint(USB0_BASE);
USBIntEnableControl(USB0_BASE, USB_INTCTRL_RESET |
                                    USB_INTCTRL_DISCONNECT |
                                    USB_INTCTRL_RESUME |
                                    USB_INTCTRL_SUSPEND |
                                    USB_INTCTRL_SOF);
USBIntEnableEndpoint(USB0_BASE, USB_INTEP_ALL);
//断开→连接
USBDevDisconnect(USB0_BASE);
SysCtlDelay(SysCtlClockGet() / 30);
USBDevConnect(USB0_BASE);
//使能总中断
IntEnable(INT_USB0);
//音频设备会传输大量数据，打开 DMA 最好。
uDMAChannelControlSet(psDevice->psPrivateData->ucOUTDMA,
                        (UDMA_SIZE_32 | UDMA_SRC_INC_NONE|
                         UDMA_DST_INC_32 | UDMA_ARB_16));
USBEndpointDMAChannel(USB0_BASE, psDevice->psPrivateData->ucOUTEndpoint,
                        psDevice->psPrivateData->ucOUTDMA);
```

第二：USB 音频设备描述符。

```
//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,
    'A', 0, 'u', 0, 'd', 0, 'i', 0, 'o', 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_pInterfaceString[] =
{
    (15 + 1) * 2,
    USB_DTYPE_STRING,
    'A', 0, 'u', 0, 'd', 0, 'i', 0, 'o', 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[] =
{
    (20 + 1) * 2,
    USB_DTYPE_STRING,
    'A', 0, 'u', 0, 'd', 0, 'i', 0, 'o', 0, ' ', 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_pInterfaceString,
    g_pConfigString
};
//音频设备描述符
static unsigned char g_pAudioDeviceDescriptor[] =
{
    18,                     // Size of this structure.
    USB_DTYPE_DEVICE,       // Type of this structure.
    USBShort(0x110),        // USB version 1.1 (if we say 2.0, hosts assume
                            // high-speed - see USB 2.0 spec 9.2.6.6)
    USB_CLASS_AUDIO,        // USB Device Class (spec 5.1.1)
    USB_SUBCLASS_UNDEFINED, // USB Device Sub-class (spec 5.1.1)
    USB_PROTOCOL_UNDEFINED, // USB Device protocol (spec 5.1.1)
    64,                     // Maximum packet size for default pipe.
    USBShort(0x1111),       // Vendor ID (filled in during USBDAudioInit).
    USBShort(0xffee),       // Product ID (filled in during USBDAudioInit).
    USBShort(0x100),        // Device Version BCD.
    1,                      // Manufacturer string identifier.
    2,                      // Product string identifier.
    3,                      // Product serial number.
    1                       // Number of configurations.
```

```
};
//音频配置描述符
static unsigned char g_pAudioDescriptor[] =
{
    9,                          // Size of the configuration descriptor.
    USB_DTYPE_CONFIGURATION,    // Type of this descriptor.
    USBShort(32),               // The total size of this full structure.
    2,                          // The number of interfaces in this
                                // configuration.
    1,                          // The unique value for this configuration.
    0,                          // The string identifier that describes this
                                // configuration.
    USB_CONF_ATTR_BUS_PWR,      // Bus Powered, Self Powered, remote wake up.
    250,                        // The maximum power in 2mA increments.
};
//音频接口描述符
unsigned char g_pIADAudioDescriptor[] =
{

    8,                          // Size of the interface descriptor.
    USB_DTYPE_INTERFACE_ASC,    // Interface Association Type.
    0x0,                        // Default starting interface is 0.
    0x2,                        // Number of interfaces in this association.
    USB_CLASS_AUDIO,            // The device class for this association.
    USB_SUBCLASS_UNDEFINED,     // The device subclass for this association.
    USB_PROTOCOL_UNDEFINED,     // The protocol for this association.
    0                           // The string index for this association.
};
//音频控制接口描述符
const unsigned char g_pAudioControlInterface[] =
{
    9,                          // Size of the interface descriptor.
    USB_DTYPE_INTERFACE,        // Type of this descriptor.
    AUDIO_INTERFACE_CONTROL,    // The index for this interface.
    0,                          // The alternate setting for this interface.
    0,                          // The number of endpoints used by this
                                // interface.
    USB_CLASS_AUDIO,            // The interface class
    USB_ASC_AUDIO_CONTROL,      // The interface sub-class.
    0,                          // The interface protocol for the sub-class
                                // specified above.
    0,                          // The string index for this interface.

    // Audio Header Descriptor.
    9,                          // The size of this descriptor.
    USB_DTYPE_CS_INTERFACE,     // Interface descriptor is class specific.
    USB_ACDSTYPE_HEADER,        // Descriptor sub-type is HEADER.
    USBShort(0x0100),           // Audio Device Class Specification Release
                                // Number in Binary-Coded Decimal.
                                // Total number of bytes in
                                // g_pAudioControlInterface
    USBShort((9 + 9 + 12 + 13 + 9)),
    1,                          // Number of streaming interfaces.
    1,                          // Index of the first and only streaming
                                // interface.

    // Audio Input Terminal Descriptor.
    12,                         // The size of this descriptor.
    USB_DTYPE_CS_INTERFACE,     // Interface descriptor is class specific.
```

```
    USB_ACDSTYPE_IN_TERMINAL,   // Descriptor sub-type is INPUT_TERMINAL.
    AUDIO_IN_TERMINAL_ID,       // Terminal ID for this interface.
                                // USB streaming interface.
    USBShort(USB_TTYPE_STREAMING),
    0,                          // ID of the Output Terminal to which this
                                // Input Terminal is associated.
    2,                          // Number of logical output channels in the
                                // Terminal担 output audio channel cluster.
    USBShort((USB_CHANNEL_L |   // Describes the spatial location of the
            USB_CHANNEL_R)),    // logical channels.
    0,                          // Channel Name string index.
    0,                          // Terminal Name string index.

    // Audio Feature Unit Descriptor
    13,                         // The size of this descriptor.
    USB_DTYPE_CS_INTERFACE,     // Interface descriptor is class specific.
    USB_ACDSTYPE_FEATURE_UNIT,  // Descriptor sub-type is FEATURE_UNIT.
    AUDIO_CONTROL_ID,           // Unit ID for this interface.
    AUDIO_IN_TERMINAL_ID,       // ID of the Unit or Terminal to which this
                                // Feature Unit is connected.
    2,                          // Size in bytes of an element of the
                                // bmaControls() array that follows.
                                // Master Mute control.
    USBShort(USB_ACONTROL_MUTE),
                                // Left channel volume control.
    USBShort(USB_ACONTROL_VOLUME),
                                // Right channel volume control.
    USBShort(USB_ACONTROL_VOLUME),
    0,                          // Feature unit string index.

    // Audio Output Terminal Descriptor.
    9,                          // The size of this descriptor.
    USB_DTYPE_CS_INTERFACE,     // Interface descriptor is class specific.
    USB_ACDSTYPE_OUT_TERMINAL,  // Descriptor sub-type is INPUT_TERMINAL.
    AUDIO_OUT_TERMINAL_ID,      // Terminal ID for this interface.
                                // Output type is a generic speaker.
    USBShort(USB_ATTYPE_SPEAKER),
    AUDIO_IN_TERMINAL_ID,       // ID of the input terminal to which this
                                // output terminal is connected.
    AUDIO_CONTROL_ID,           // ID of the feature unit that this output
                                // terminal is connected to.
    0,                          // Output terminal string index.
};
//音频流接口描述符
const unsigned char g_pAudioStreamInterface[] =
{
    9,                          // Size of the interface descriptor.
    USB_DTYPE_INTERFACE,        // Type of this descriptor.
    AUDIO_INTERFACE_OUTPUT,     // The index for this interface.
    0,                          // The alternate setting for this interface.
    0,                          // The number of endpoints used by this
                                // interface.
    USB_CLASS_AUDIO,            // The interface class
    USB_ASC_AUDIO_STREAMING,    // The interface sub-class.
    0,                          // Unused must be 0.
    0,                          // The string index for this interface.

    // Vendor-specific Interface Descriptor.
    9,                          // Size of the interface descriptor.
```

```
    USB_DTYPE_INTERFACE,         // Type of this descriptor.
    1,                           // The index for this interface.
    1,                           // The alternate setting for this interface.
    1,                           // The number of endpoints used by this
                                 // interface.
    USB_CLASS_AUDIO,             // The interface class
    USB_ASC_AUDIO_STREAMING,     // The interface sub-class.
    0,                           // Unused must be 0.
    0,                           // The string index for this interface.

    // Class specific Audio Streaming Interface descriptor.
    7,                           // Size of the interface descriptor.
    USB_DTYPE_CS_INTERFACE,      // Interface descriptor is class specific.
    USB_ASDSTYPE_GENERAL,        // General information.
    AUDIO_IN_TERMINAL_ID,        // ID of the terminal to which this streaming
                                 // interface is connected.
    1,                           // One frame delay.
    USBShort(USB_ADF_PCM),       //

    // Format type Audio Streaming descriptor.
    11,                          // Size of the interface descriptor.
    USB_DTYPE_CS_INTERFACE,      // Interface descriptor is class specific.
    USB_ASDSTYPE_FORMAT_TYPE,    // Audio Streaming format type.
    USB_AF_TYPE_TYPE_I,          // Type I audio format type.
    2,                           // Two audio channels.
    2,                           // Two bytes per audio sub-frame.
    16,                          // 16 bits per sample.
    1,                           // One sample rate provided.
    USB3Byte(48000),             // Only 48000 sample rate supported.

    // Endpoint Descriptor
    9,                               // The size of the endpoint descriptor.
    USB_DTYPE_ENDPOINT,              // Descriptor type is an endpoint.
                                     // OUT endpoint with address
                                     // ISOC_OUT_ENDPOINT.
    USB_EP_DESC_OUT | USB_EP_TO_INDEX(ISOC_OUT_ENDPOINT),
    USB_EP_ATTR_ISOC |               // Endpoint is an adaptive isochronous data
    USB_EP_ATTR_ISOC_ADAPT |         //  endpoint.
    USB_EP_ATTR_USAGE_DATA,
    USBShort(ISOC_OUT_EP_MAX_SIZE), // The maximum packet size.
    1,                               // The polling interval for this endpoint.
    0,                               // Refresh is unused.
    0,                               // Synch endpoint address.

    // Audio Streaming Isochronous Audio Data Endpoint Descriptor
    7,                               // The size of the descriptor.
    USB_ACSDT_ENDPOINT,              // Audio Class Specific Endpoint Descriptor.
    USB_ASDSTYPE_GENERAL,            // This is a general descriptor.
    USB_EP_ATTR_ACG_SAMPLING,        // Sampling frequency is supported.
    USB_EP_LOCKDELAY_UNDEF,          // Undefined lock delay units.
    USBShort(0),                     // No lock delay.
};
```

第三：USB 音频设备枚举。

```
/枚举用到函数
static void USBDGetStatus(tUSBRequest *pUSBRequest);
static void USBDClearFeature(tUSBRequest *pUSBRequest);
static void USBDSetFeature(tUSBRequest *pUSBRequest);
static void USBDSetAddress(tUSBRequest *pUSBRequest);
static void USBDGetDescriptor(tUSBRequest *pUSBRequest);
```

```
static void USBDSetDescriptor(tUSBRequest *pUSBRequest);
static void USBDGetConfiguration(tUSBRequest *pUSBRequest);
static void USBDSetConfiguration(tUSBRequest *pUSBRequest);
static void USBDGetInterface(tUSBRequest *pUSBRequest);
static void USBDSetInterface(tUSBRequest *pUSBRequest);
static void USBDEP0StateTx(void);
static long USBDStringIndexFromRequest(unsigned short usLang,
                                       unsigned short usIndex);

//该结构能够完整表述 USB 设备枚举过程。

typedef struct
{
    //当前 USB 设备地址，也可以能过 DEV_ADDR_PENDING 最高位改变.
    volatile unsigned long ulDevAddress;
    //保存设备当前生效的配置.
    unsigned long ulConfiguration;
    //当前设备的接口设置
    unsigned char ucAltSetting;
    //指向端点 0 正发接收或者发送的数据组。
    unsigned char *pEP0Data;
    //指示端点 0 正发接收或者发送数据的剩下数据量。
    volatile unsigned long ulEP0DataRemain;
    //端点 0 正发接收或者发送数据的数据总量
    unsigned long ulOUTDataSize;
    //当前设备状态
    unsigned char ucStatus;
     //在处理过程中是否使用 wakeup 信号。
    tBoolean bRemoteWakeup;
    //bRemoteWakeup 信号计数。
    unsigned char ucRemoteWakeupCount;
}
tDeviceState;

//定义端点输出/输入。

#define HALT_EP_IN              0
#define HALT_EP_OUT             1
//端点 0 的状态，在枚举过程中使用。
typedef enum
{
    //等待主机请求。
    USB_STATE_IDLE,
    //通过 IN 端口 0 给主机发送数块。
    USB_STATE_TX,
    //通过 OUT 端口 0 从主机接收数据块。
    USB_STATE_RX,
    //端点 0 发送/接收完成，等待主机应答。
    USB_STATE_STATUS,
    //端点 0STALL，等待主机响应 STALL。
    USB_STATE_STALL
}
tEP0State;
//端点 0 最大传输包大小。
#define EP0_MAX_PACKET_SIZE     64
//用于指示设备地址改变
#define DEV_ADDR_PENDING        0x80000000
//总线复位后，默认的配置编号。
#define DEFAULT_CONFIG_ID       1
```

```
//REMOTE_WAKEUP 的信号毫秒数，在协议中定义为 1ms-15ms.
#define REMOTE_WAKEUP_PULSE_MS 10
//REMOTE_WAKEUP 保持 20ms.
#define REMOTE_WAKEUP_READY_MS 20
//端点 0 的读数据缓存。
static unsigned char g_pucDataBufferIn[EP0_MAX_PACKET_SIZE];
//定义当前设备状态信息实例。
static volatile tDeviceState g_sUSBDeviceState;
//定义当前端点 0 的状态
static volatile tEP0State g_eUSBDEP0State = USB_STATE_IDLE;
//请求函数表。
static const tStdRequest g_psUSBDStdRequests[] =
{
    USBDGetStatus,
    USBDClearFeature,
    0,
    USBDSetFeature,
    0,
    USBDSetAddress,
    USBDGetDescriptor,
    USBDSetDescriptor,
    USBDGetConfiguration,
    USBDSetConfiguration,
    USBDGetInterface,
    USBDSetInterface,
};
//在读取 usb 中断时合并使用。
#define USB_INT_RX_SHIFT        8
#define USB_INT_STATUS_SHIFT    24
#define USB_RX_EPSTATUS_SHIFT   16
//端点控制状态寄存器转换
#define EP_OFFSET(Endpoint)     (Endpoint - 0x10)

//从端点 0 的 FIFO 中获取数据。
long USBEndpoint0DataGet(unsigned char *pucData, unsigned long *pulSize)
{
    unsigned long ulByteCount;
     //判断端点 0 的数据是否接收完成。
    if((HWREGH(USB0_BASE + USB_O_CSRL0) & USB_CSRL0_RXRDY) == 0)
    {
        *pulSize = 0;
        return(-1);
    }
     //USB_O_COUNT0 指示端点 0 收到的数据量。
    ulByteCount = HWREGH(USB0_BASE + USB_O_COUNT0 + USB_EP_0);
    //确定读回的数据量。
    ulByteCount = (ulByteCount < *pulSize) ? ulByteCount : *pulSize;
    *pulSize = ulByteCount;
     //从 FIFO 中读取数据。
    for(; ulByteCount > 0; ulByteCount--)
    {
         *pucData++ = HWREGB(USB0_BASE + USB_O_FIFO0 + (USB_EP_0 >> 2));
    }
    return(0);
}
//端点 0 应答。
void USBDevEndpoint0DataAck(tBoolean bIsLastPacket)
{
    HWREGB(USB0_BASE + USB_O_CSRL0) =
```

```
        USB_CSRL0_RXRDYC | (bIsLastPacket ? USB_CSRL0_DATAEND : 0);
}
// 向端点 0 中放入数据。
long USBEndpoint0DataPut(unsigned char *pucData, unsigned long ulSize)
{
    if(HWREGB(USB0_BASE + USB_O_CSRL0 + USB_EP_0) & USB_CSRL0_TXRDY)
    {
        return(-1);
    }
    for(; ulSize > 0; ulSize--)
    {
        HWREGB(USB0_BASE + USB_O_FIFO0 + (USB_EP_0 >> 2)) = *pucData++;
    }
    return(0);
}
//向端点 0 中写入数据。
long USBEndpoint0DataSend(unsigned long ulTransType)
{
    //判断是否已经有数据准备好。
    if(HWREGB(USB0_BASE + USB_O_CSRL0 + USB_EP_0) & USB_CSRL0_TXRDY)
    {
        return(-1);
    }
    HWREGB(USB0_BASE + USB_O_CSRL0 + USB_EP_0) = ulTransType & 0xff;
    return(0);
}
//端点 0 从主机上获取数据。
void USBRequestDataEP0(unsigned char *pucData, unsigned long ulSize)
{
    g_eUSBDEP0State = USB_STATE_RX;
    g_sUSBDeviceState.pEP0Data = pucData;
    g_sUSBDeviceState.ulOUTDataSize = ulSize;
    g_sUSBDeviceState.ulEP0DataRemain = ulSize;
}
//端点 0 请求发送数据。
void USBSendDataEP0(unsigned char *pucData, unsigned long ulSize)
{
    g_sUSBDeviceState.pEP0Data = pucData;
    g_sUSBDeviceState.ulEP0DataRemain = ulSize;
    g_sUSBDeviceState.ulOUTDataSize = ulSize;
    USBDEP0StateTx();
}
//端点 0 处于停止状态。
void USBStallEP0(void)
{
    HWREGB(USB0_BASE + USB_O_CSRL0) |= (USB_CSRL0_STALL | USB_CSRL0_RXRDYC);
    g_eUSBDEP0State = USB_STATE_STALL;
}
//从端点 0 中获取一个请求。
static void USBDReadAndDispatchRequest(void)
{
    unsigned long ulSize;
    tUSBRequest *pRequest;
    pRequest = (tUSBRequest *)g_pucDataBufferIn;
    ulSize = EP0_MAX_PACKET_SIZE;
    USBEndpoint0DataGet(g_pucDataBufferIn, &ulSize);
    if(!ulSize)
    {
        return;
```

```
    }
     //判断是否是标准请求。
    if((pRequest->bmRequestType & USB_RTYPE_TYPE_M) != USB_RTYPE_STANDARD)
    {
         UARTprintf("非标准请求.....\r\n");
    }
    else
    {
         //调到标准请求处理函数中。
        if((pRequest->bRequest <
           (sizeof(g_psUSBDStdRequests) / sizeof(tStdRequest))) &&
           (g_psUSBDStdRequests[pRequest->bRequest] != 0))
        {
            g_psUSBDStdRequests[pRequest->bRequest](pRequest);
        }
        else
        {
            USBStallEP0();
        }
    }
}
//枚举过程。
// USB_STATE_IDLE -*--> USB_STATE_TX -*-> USB_STATE_STATUS -*->USB_STATE_IDLE
//                 |                   |                    |
//                 |--> USB_STATE_RX -                      |
//                 |                                        |
//                 |--> USB_STATE_STALL ---------->---------
//
//  ------------------------------------------------------------
// | Current State       | State 0           | State 1          |
// | --------------------|-------------------|-------------------
// | USB_STATE_IDLE      | USB_STATE_TX/RX   | USB_STATE_STALL  |
// | USB_STATE_TX        | USB_STATE_STATUS  |                  |
// | USB_STATE_RX        | USB_STATE_STATUS  |                  |
// | USB_STATE_STATUS    | USB_STATE_IDLE    |                  |
// | USB_STATE_STALL     | USB_STATE_IDLE    |                  |
//  ------------------------------------------------------------
void USBDeviceEnumHandler(void)
{
    unsigned long ulEPStatus;
     //获取中断状态。
    ulEPStatus = HWREGH(USB0_BASE + EP_OFFSET(USB_EP_0) + USB_O_TXCSRL1);
    ulEPStatus |= ((HWREGH(USB0_BASE + EP_OFFSET(USB_EP_0) + USB_O_RXCSRL1)) <<
                  USB_RX_EPSTATUS_SHIFT);

   //端点 0 的状态。
    switch(g_eUSBDEP0State)
    {
        case USB_STATE_STATUS:
        {
              UARTprintf("USB_STATE_STATUS.....\r\n");
            g_eUSBDEP0State = USB_STATE_IDLE;
            //判断地址改变。
            if(g_sUSBDeviceState.ulDevAddress & DEV_ADDR_PENDING)
            {
                  //设置地址。
                g_sUSBDeviceState.ulDevAddress &= ~DEV_ADDR_PENDING;
                HWREGB(USB0_BASE + USB_O_FADDR) =
                    (unsigned char)g_sUSBDeviceState.ulDevAddress;
```

```
    }
    //端点 0 接收包准备好。
    if(ulEPStatus & USB_DEV_EP0_OUT_PKTRDY)
    {
        USBDReadAndDispatchRequest();
    }
    break;
}
//等待从主机接收数据。
case USB_STATE_IDLE:
{
    if(ulEPStatus & USB_DEV_EP0_OUT_PKTRDY)
    {
        USBDReadAndDispatchRequest();
    }
    break;
}
//数据处理好，准备发送。
case USB_STATE_TX:
{
    USBDEP0StateTx();
    break;
}
 //接收数据。
case USB_STATE_RX:
{
    unsigned long ulDataSize;
    if(g_sUSBDeviceState.ulEP0DataRemain > EP0_MAX_PACKET_SIZE)
    {
        ulDataSize = EP0_MAX_PACKET_SIZE;
    }
    else
    {
        ulDataSize = g_sUSBDeviceState.ulEP0DataRemain;
    }
    USBEndpoint0DataGet(g_sUSBDeviceState.pEP0Data, &ulDataSize);
    if(g_sUSBDeviceState.ulEP0DataRemain < EP0_MAX_PACKET_SIZE)
    {
        USBDevEndpoint0DataAck(true);
        g_eUSBDEP0State =  USB_STATE_IDLE;
        if(g_sUSBDeviceState.ulOUTDataSize != 0)
        {
        }
    }
    else
    {
        USBDevEndpoint0DataAck(false);
    }
    g_sUSBDeviceState.pEP0Data += ulDataSize;
    g_sUSBDeviceState.ulEP0DataRemain -= ulDataSize;
    break;
}
 //停止状态
case USB_STATE_STALL:
{
    if(ulEPStatus & USB_DEV_EP0_SENT_STALL)
    {
        HWREGB(USB0_BASE + USB_O_CSRL0) &= ~(USB_DEV_EP0_SENT_STALL);
        g_eUSBDEP0State = USB_STATE_IDLE;
```

```
            }
            break;
        }

        default:
        {
            break;
        }
    }
}
```

设备枚举过程中 USBDSetAddress 和 USBDGetDescriptor 很重要，下面只列出这两个函数的具体内容。

```
// SET_ADDRESS 标准请求。
static void USBDSetAddress(tUSBRequest *pUSBRequest)
{
    USBDevEndpoint0DataAck(true);
    g_sUSBDeviceState.ulDevAddress = pUSBRequest->wValue | DEV_ADDR_PENDING;
    g_eUSBDEP0State = USB_STATE_STATUS;
    //HandleSetAddress();
}
//GET_DESCRIPTOR 标准请求。
static void USBDGetDescriptor(tUSBRequest *pUSBRequest)
{
    USBDevEndpoint0DataAck(false);
    switch(pUSBRequest->wValue >> 8)
    {
        case USB_DTYPE_DEVICE:
        {
            g_sUSBDeviceState.pEP0Data =
              (unsigned char *)g_pDFUDeviceDescriptor;
            g_sUSBDeviceState.ulEP0DataRemain = g_pDFUDeviceDescriptor[0];
            break;
        }
        case USB_DTYPE_CONFIGURATION:
        {
            unsigned char ucIndex;
            ucIndex = (unsigned char)(pUSBRequest->wValue & 0xFF);
            if(ucIndex != 0)
            {
                USBStallEP0();
                g_sUSBDeviceState.pEP0Data = 0;
                g_sUSBDeviceState.ulEP0DataRemain = 0;
            }
            else
            {
           g_sUSBDeviceState.pEP0Data =
                  (unsigned char *)g_pDFUConfigDescriptor;
                g_sUSBDeviceState.ulEP0DataRemain =
                    *(unsigned short *)&(g_pDFUConfigDescriptor[2]);
            }
            break;
        }
        case USB_DTYPE_STRING:
        {
            long lIndex;
            lIndex = USBDStringIndexFromRequest(pUSBRequest->wIndex,
                                                pUSBRequest->wValue & 0xFF);
            if(lIndex == -1)
            {
```

```
                USBStallEP0();
                break;
            }
            g_sUSBDeviceState.pEP0Data =
              (unsigned char *)g_pStringDescriptors[lIndex];
            g_sUSBDeviceState.ulEP0DataRemain = g_pStringDescriptors[lIndex][0];
            break;
        }
         case 0x22:
         {
              //USBDevEndpoint0DataAck(false);
              g_sUSBDeviceState.pEP0Data = (unsigned char *)ReportDescriptor;
              g_sUSBDeviceState.ulEP0DataRemain = sizeof(&ReportDescriptor[0]);
              //USBDEP0StateTx();
         }
        default:
        {
            USBStallEP0();
            break;
        }
    }
    if(g_sUSBDeviceState.pEP0Data)
    {

        if(g_sUSBDeviceState.ulEP0DataRemain > pUSBRequest->wLength)
        {
            g_sUSBDeviceState.ulEP0DataRemain = pUSBRequest->wLength;
        }
        USBDEP0StateTx();
    }
}
```

第四：USB 音频数据处理与控制。此过程包括数据处理，音量控制，静音控制等，控制过程较为复杂，在此不在一一讲解，可以参考相关 USB 音频设备书籍。在第 6 章有讲其它方法进行开发。

4.3 使用 USB 库开发

使用 USB 库函数进行开发，开发人员可以不深入研究 USB 协议，包括枚举过程、中断处理、数据处理等。使用库函数提供的 API 接口函数就可以完成开发工作。使用 USB 库函数方便、快捷、缩短开发周期、不易出现 bug，但占用存储空间、内存较大，由于 Stellaris USB 处理器的存储空间达 128K，远远超过程序需要的存储空间，所以使用 USB 库函数开发是比较好的方法。

例如：使用库函数开发一个 USB 鼠标。

1.完成字符串描述符。

```
//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
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
};
// 配置字符串描述
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 *))
```

2.了解鼠标设备 tUSBDHIDMouseDevice 在库函数中的定义，并完成鼠标设备实例。

```
//定义 USB 鼠标实例
tHIDMouseInstance g_sMouseInstance;
//定义 USB 鼠标相关信息
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
};
```

3.初始化 USB 鼠标设备，并进行数据处理。

```
//初始化 USB 鼠标设备，只需用这个函数就完成配置，包括枚举配置。
USBDHIDMouseInit(0, (tUSBDHIDMouseDevice *)&g_sMouseDevice);
//数据处理，改变鼠标位置和按键状态。
USBDHIDMouseStateChange((void *)&g_sMouseDevice,
                                    (char)lDeltaX, (char)lDeltaY,
                                     ucButtons);
```

从库函数开发 USB 鼠标设备过程中可以看出，使用 USB 库函数开发非常简单、方便、快捷，不用考虑底层的驱动、类协议。在 HID 类中，报告符本身就很复杂，但是使用 USB 库函数开发完全屏蔽报告符配置过程。从第五章开始介绍使用 USB 库函数开发 USB 设备与主机。