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

第六章 Audio 设备

6.1 Audio 设备介绍

USB 协议制定时，为了方便不同设备的开发商基于 USB 进行设计，定义了不同的设备类来支持不同类型的设备。虽然在 USB 标准中定义了 USB_DEVICE_CLASS_AUDIO--AUDIO 设备。但是很少有此类设备问世。目前称为 USB 音箱的设备，大都使用 USB_DEVICE_CLASS_POWER，仅仅将 USB 接口作为电源使用。完全基于 USB 协议的 USB_DEVICE_CLASS_AUDIO 设备，采用一根 USB 连接线，在设备中不同的端点实现音频信号的输入，输出包括相关按键控制。

AUDIO 设备是专门针对 USB 音频设备定义的一种专用类别，它不仅定义了音频输入、输出端点的标准，还提供了音量控制、混音器配置、左右声道平衡，甚至包括对支持杜比音效解码设备的支持，功能相当强大。不同的开发者可以根据不同的需求对主机枚举自己的设备结构，主机则根据枚举的不同设备结构提供相应的服务。

AUDIO 设备采用 USB 传输模式中的 Isochronous transfers 模式，Isochronous transfers 传输模式是专门针对流媒体特点的传输方法。它依照设备在链接初始化时枚举的参数，保证提供稳定的带宽给采用该模式的设备或端点。由于对实时性的要求，它不提供相应的接收 / 应答和握手协议。这很好地适应了音频数据流量稳定、对差错相对不敏感的特点。

6.2 Audio 描述符

Audio 设备定义了三种接口描述符子类，子协议恒为 0x00。三种接口描述符子类分别定义为：

```
// Audio Interface Subclass Codes
#define USB_ASC_AUDIO_CONTROL    0x01
#define USB_ASC_AUDIO_STREAMING 0x02
#define USB_ASC_MIDI_STREAMING   0x03
```

USB_ASC_AUDIO_CONTROL，音频设备控制类；USB_ASC_AUDIO_STREAMING，音频流类；USB_ASC_MIDI_STREAMING，MIDI 流类。以上三种用于描述接口描述符的子类，确定被描述接口的功能。Audio 设备没有定义子协议，所以为定值 0x00。

Audio 设备接口描述符中要包含 Class-Specific AC Interface Header Descriptor，用于定义接口的其它功能端口。Class-Specific AC Interface Header Descriptor（AC 接口头）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 USB_DTYPE_CS_INTERFACE（36）
2	bDescriptorSubtype	1	数字	AC 接口头：USB_DSUBTYPE_HEADER
3	bcdADC	2	数字	Audio 设备发行号，BCD 码
5	wTotalLength	2	数字	接口描述符总长度，包括标准接口描述符
6	bInCollection	1	索引	AudioStreaming、 MIDIStreaming 使用代码
7	baInterfaceNr	2	位图	使用 AudioStreaming 或者 MIDIStreaming 的接口号

表 1. AC 接口头描述符

C 语言 AC 接口头描述符结构体为：

```
typedef struct
{
    //本描述符长度.
    unsigned char bLength;
    //描述类型 USB_DTYPE_CS_INTERFACE (36).
    unsigned char bDescriptorSubtype;
    //发行号（BCD 码）
    unsigned short bcdADC;
    //接口描述符总长度，包括标准接口描述符
    unsigned short wTotalLength;
    // AudioStreaming、 MIDIStreaming 使用代码
    unsigned char bInCollection;
    // 使用 AudioStreaming 或者 MIDIStreaming 的接口号
    unsigned char baInterfaceNr;
}
tACHeader;
```

例如：定义一个实际的 AC 接口头描述符。

```
const unsigned char g_pAudioInterfaceHeader[] =
{
    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 设备的输入端口描述符（Input Terminal Descriptor），对于输入端定义。输入端口描述符如下表：

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 USB_DTYPE_CS_INTERFACE（36）
2	bDescriptorSubtype	1	常量	USB_ACDSTYPE_IN_TERMINAL
3	bTerminalID	1	常量	本端口 ID 号
4	wTerminalType	2	常量	端口类型
6	bAssocTerminal	1	常量	对应输出端口 ID
7	bNrChannels	2	位图	通道数量
8	wChannelConfig	2	数字	通道配置
10	iChannelNames	1	常量	通道名字
11	iTerminal	1	数字	端口字符串描述符索引

表 1. HID 描述符符

C 语言输入端口描述符结构体为：

```
typedef struct
{
    unsigned char bLength;
    unsigned char bDescriptorType;
    unsigned char bDescriptorSubtype;
    unsigned char bTerminalID;
    unsigned short wTerminalType;
    unsigned char bAssocTerminal;
    unsigned char bNrChannels;
    unsigned short wChannelConfig;
    unsigned char iChannelNames;
    unsigned char iTerminal;
}
tACInputTerminal;
```

Audio 设备的输出端口描述符（Output Terminal Descriptor），对于输出端定义。输出端口描述符如下表：

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 USB_DTYPE_CS_INTERFACE（36）
2	bDescriptorSubtype	1	常量	USB_ACDSTYPE_OUT_TERMINAL
3	bTerminalID	1	常量	本端口 ID 号
4	wTerminalType	2	常量	端口类型
6	bAssocTerminal	1	常量	对应输入端口 ID
7	bSourceID	1	常量	连接端口的 ID 号

8	iTerminal	1	数字	端口字符串描述符索引

表 1. HID 描述符符

C 语言输出端口描述符结构体为：

```
typedef struct
{
    unsigned char bLength;
    unsigned char bDescriptorType;
    unsigned char bDescriptorSubtype;
    unsigned char bTerminalID;
    unsigned short wTerminalType;
    unsigned char bAssocTerminal;
    unsigned char bSourceID;
    unsigned char iTerminal;
}
tACOutputTerminal;
```

其它与 Audio 设备相关的描述符，请读者参阅 USB_Audio_Class 手册。下面列出一个 Audio 设备的接口描述符：

```
const unsigned char g_pAudioControlInterface[] =
{
    //标准接口描述符
    9,                          // Size of the interface descriptor.
    USB_DTYPE_INTERFACE,        // Type of this descriptor.
    AUDIO_INTERFACE_CONTROL,    // 接口编号，从 0 开发编. AUDIO_INTERFACE_CONTROL = 0
    0,                          // The alternate setting for this interface.
    0,                          // The number of endpoints used by this interface.
    USB_CLASS_AUDIO,            // AUDIO 设备
    USB_ASC_AUDIO_CONTROL,      // 子类，USB_ASC_AUDIO_CONTROL 用于 Audio 控制.
    0,                          // 无协议，定值 0。
    0,                          // The string index for this interface.
    // Audio 接口头描述符.
    9,                          // The size of this descriptor.
    USB_DTYPE_CS_INTERFACE,     // 描述符类型.
    USB_ACDSTYPE_HEADER,        // 子类型，本描述为描述头.
    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 设备输入端口描述符.
    12,                         // The size of this descriptor.
    USB_DTYPE_CS_INTERFACE,     // Interface descriptor is class specific.
```

```
    USB_ACDSTYPE_IN_TERMINAL,    // 本描述符为输入端口.
    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 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 设备特征单元描述符
    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.
    //输出端口描述符.
    9,                           // The size of this descriptor.
    USB_DTYPE_CS_INTERFACE,      // Interface descriptor is class specific.
    USB_ACDSTYPE_OUT_TERMINAL,   // 输出端口描述符.
    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
```

```
const unsigned char g_pAudioStreamInterface[] =
{
    //标准接口描述符
    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.
    //端点描述符
    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.
};
```

6.3 Audio 数据类型

usbdaudio.h中已经定义好Audio设备类中使用的所有数据类型和函数，下面介绍Audio设备类使用的数据类型。

```
typedef struct
{
    unsigned long ulUSBBase;
     //设备信息指针
    tDeviceInfo *psDevInfo;
     //配置描述符
    tConfigDescriptor *psConfDescriptor;
    //最大音量值
    short sVolumeMax;
    //最小音量值
    short sVolumeMin;
    //音量控制阶梯值
    short sVolumeStep;
    struct
    {
        //callback 入口参数
        void *pvData;
        // pvData 大小
        unsigned long ulSize;
        // 可用 pvData 大小
        unsigned long ulNumBytes;
        // Callback
```

```
        tUSBAudioBufferCallback pfnCallback;
    } sBuffer;
    //请求类型.
    unsigned short usRequestType;
    //请求标志
    unsigned char ucRequest;
    // 更新值
     unsigned short usUpdate;
    // 当前音量设置.
    unsigned short usVolume;
    // 静音设置.
    unsigned char ucMute;
    //采样率
    unsigned long ulSampleRate;
    // 使用输出端点
    unsigned char ucOUTEndpoint;
    // 输出端点 DMA 通道
    unsigned char ucOUTDMA;
    //控制接口
    unsigned char ucInterfaceControl;
    //Audio 接口
    unsigned char ucInterfaceAudio;
}
tAudioInstance;
```

tAudioInstance，Audio 设备类实例。用于保存全部 Audio 设备类的配置信息，包括描述符、callback 函数、控制事件等。

```
#define USB_AUDIO_INSTANCE_SIZE sizeof(tAudioInstance);
#define COMPOSITE_DAUDIO_SIZE   (8 + 52 + 52)
```

USB_AUDIO_INSTANCE_SIZE ， 定 义 Audio 设 备 类 实 例 信 息 的 大 小 。 COMPOSITE_DAUDIO_SIZE 定义设备描述符与所有接口描述符总长度。

```
typedef struct
{
    // VID
    unsigned short usVID;
    // PID
    unsigned short usPID;
    //8 字节供应商字符串
    unsigned char pucVendor[8];
    //16 字节产品字符串
    unsigned char pucProduct[16];
    //4 字节版本号
    unsigned char pucVersion[4];
    //最大耗电量
    unsigned short usMaxPowermA;
```

```
    //电源属性  USB_CONF_ATTR_SELF_PWR   USB_CONF_ATTR_BUS_PWR
    //USB_CONF_ATTR_RWAKE.
    unsigned char ucPwrAttributes;
    // callback 函数
    tUSBCallback pfnCallback;
    //字符串描述符集合
    const unsigned char * const *ppStringDescriptors;
    //字符串描述符个数
    unsigned long ulNumStringDescriptors;
    //最大音量
    short sVolumeMax;
    //最小音量
    short sVolumeMin;
    //音量调节步进
    short sVolumeStep;
    //Audio 设备类实例
    tAudioInstance *psPrivateData;
}
tUSBDAudioDevice;
```

tUSBDAudioDevice，Audio 设备类。定义了 VID、PID、电源属性、字符串描述符等，还包括了一个 Audio 设备类实例。其它设备描述符、配置信息通过 API 函数储入 tAudioInstance 定义的 Audio 设备实例中。

6.4 API 函数

在 Audio 设备类 API 库中定义了 4 个函数，完成 USB Audio 设备初始化、配置及数据处理。下面为 usbdaudio.h 中定义的 API 函数：

```
void *USBDAudioInit(unsigned long ulIndex,
                            const tUSBDAudioDevice *psAudioDevice);
void *USBDAudioCompositeInit(unsigned long ulIndex,
                                    const tUSBDAudioDevice *psAudioDevice);
int USBAudioBufferOut(void *pvInstance, void *pvBuffer, unsigned long ulSize,
                              tUSBAudioBufferCallback pfnCallback);
void USBDAudioTerm(void *pvInstance);
```

```
void *USBDAudioInit(unsigned long ulIndex,
                                const tUSBDAudioDevice *psAudioDevice);
```

作用：初始化 Audio 设备硬件、协议，把其它配置参数填入 psAudioDevice 实例中。
参数：ulIndex，USB 模块代码，固定值：USB_BASE0。psAudioDevice，Audio 设备类。
返回：指向配置后的 tUSBDAudioDevice。

```
void *USBDAudioCompositeInit(unsigned long ulIndex,
                                        const tUSBDAudioDevice *psAudioDevice);
```

作用：初始化 Audio 设备协议，本函数在 USBDAudioInit 中已经调用。
参数：ulIndex，USB 模块代码，固定值：USB_BASE0。psAudioDevice，Audio 设备类。
返回：指向配置后的 tUSBDAudioDevice。

```
int USBAudioBufferOut(void *pvInstance, void *pvBuffer, unsigned long ulSize,
```

tUSBAudioBufferCallback pfnCallback);

作用：从输出端点获取数据，并放入 pvBuffer 中。

参数:pvInstance,指向 tUSBDAudioDevice。pvBuffer,用于存放输出端点数据。ulSize，设置数据大小。pfnCallback，输出端点返回，只有一个事件，USBD_AUDIO_EVENT_DATAOUT。

返回：无。

void USBDAudioTerm(void *pvInstance);

作用：结束 Audio 设备。

参数：pvInstance，指向 tUSBDAudioDevice。

返回：无。

在这些函数中 USBDAudioInit 和 USBAudioBufferOut 函数最重要并且使用最多，USBDAudioInit 第一次使用 Audio 设备时，用于初始化 Audio 设备的配置与控制。USBAudioBufferOut 从输出端点中获取数据，并放入数据缓存区内。

6.5 Audio 设备开发

Audio 设备开发只需要 5 步就能完成。如图 2 所示，Audio 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、DMA 控制、数据处理。

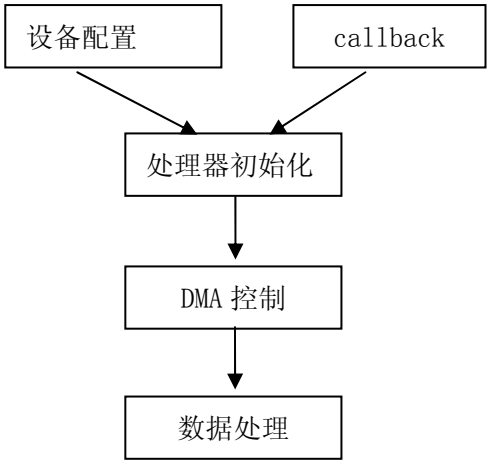

图 2

第一步：Audio 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 Audio 设备配置。

```
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_sysctl.h"
#include "inc/hw_udma.h"
#include "inc/hw_gpio.h"
#include "driverlib/gpio.h"
#include "driverlib/interrupt.h"
#include "driverlib/sysctl.h"
#include "driverlib/udma.h"
#include "usblib/usblib.h"
#include "usblib/usb-ids.h"
```

```
#include "usblib/device/usbdevice.h"
#include "usblib/device/usbdaudio.h"

//根据具体 Audio 芯片修改
#define VOLUME_MAX              ((short)0x0C00)  // +12db
#define VOLUME_MIN              ((short)0xDC80)  // -34.5db
#define VOLUME_STEP             ((short)0x0180)  // 1.5db
//Audio 设备
const tUSBDAudioDevice g_sAudioDevice;
//DMA 控制
tDMAControlTable sDMAControlTable[64] __attribute__ ((aligned(1024)));
//*****************************************************************************
// 缓存与标志.
//*****************************************************************************
#define AUDIO_PACKET_SIZE       ((48000*4)/1000)
#define AUDIO_BUFFER_SIZE       (AUDIO_PACKET_SIZE*20)
#define SBUFFER_FLAGS_PLAYING   0x00000001
#define SBUFFER_FLAGS_FILLING   0x00000002
struct
{
    //主要 buffer，USB audio class 和 sound driver 使用.
    volatile unsigned char pucBuffer[AUDIO_BUFFER_SIZE];
    // play pointer.
    volatile unsigned char *pucPlay;
    // USB fill pointer.
    volatile unsigned char *pucFill;
    // 采样率 调整.
    volatile int iAdjust;
    // 播放状态
    volatile unsigned long ulFlags;
} g_sBuffer;
//*****************************************************************************
// 当前音量
//*****************************************************************************
short g_sVolume;
//*****************************************************************************
// 通过 USBDAudioInit() 函数，完善 Audio 设备配置信息
//*****************************************************************************
void *g_pvAudioDevice;
// 音量更新
#define FLAG_VOLUME_UPDATE      0x00000001
// 更新静音状态
#define FLAG_MUTE_UPDATE        0x00000002
// 静音状态
```

```
#define FLAG_MUTED              0x00000004
// 连接成功
#define FLAG_CONNECTED          0x00000008
volatile unsigned long g_ulFlags;
extern unsigned long
AudioMessageHandler(void *pvCBData, unsigned long ulEvent,
                    unsigned long ulMsgParam, void *pvMsgData);
//*****************************************************************************
// 设备语言描述符.
//*****************************************************************************
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
};
#define NUM_STRING_DESCRIPTORS (sizeof(g_pStringDescriptors) /               \
                                sizeof(unsigned char *))
//*****************************************************************************
// 定义 Audio 设备实例
//*****************************************************************************
static tAudioInstance g_sAudioInstance;
//*****************************************************************************
// 定义 Audio 设备类
//*****************************************************************************
const tUSBDAudioDevice g_sAudioDevice =
```

```
{
    // VID
    USB_VID_STELLARIS,
    // PID
    USB_PID_AUDIO,
    // 8 字节供应商字符串.
    "TI      ",
    //16 字节产品字符串.
    "Audio Device    ",
    //4 字节版本字符串.
    "1.00",
    500,
    USB_CONF_ATTR_SELF_PWR,
    AudioMessageHandler,
    g_pStringDescriptors,
    NUM_STRING_DESCRIPTORS,
    VOLUME_MAX,
    VOLUME_MIN,
    VOLUME_STEP,
    &g_sAudioInstance
};
```

第二步：完成 Callback 函数。Callback 函数用于处理输出端点数据事务。主机发出的音频流数据，也可能是状态信息。Audio 设备中包含了以下事务：USBD_AUDIO_EVENT_IDLE、USBD_AUDIO_EVENT_ACTIVE 、 USBD_AUDIO_EVENT_MUTE 、 USBD_AUDIO_EVENT_VOLUME 、USB_EVENT_DISCONNECTED、USB_EVENT_CONNECTED。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_AUDIO_EVENT_VOLUME	更新音量
USBD_AUDIO_EVENT_MUTE	静音
USBD_AUDIO_EVENT_ACTIVE	Audio 处于活动状态
USBD_AUDIO_EVENT_IDLE	Audio 处于空闲状态

表 2. Audio 事务

根据以上事务编写 Callback 函数：

```
//*****************************************************************************
//USB Audio 设备类返回事件处理函数（callback）.
//*****************************************************************************
unsigned long AudioMessageHandler(void *pvCBData, unsigned long ulEvent,
                                  unsigned long ulMsgParam, void *pvMsgData)
{
    switch(ulEvent)
    {
        //Audio 正处于空闲或者工作状态。
        case USBD_AUDIO_EVENT_IDLE:
```

```
case USBD_AUDIO_EVENT_ACTIVE:
{
      GPIOPinWrite(GPIO_PORTF_BASE,0x10,0x10);
    g_ulFlags |= FLAG_CONNECTED;
    break;
}
// 静音控制.
case USBD_AUDIO_EVENT_MUTE:
{
    // 检查是否静音.
    if(ulMsgParam == 1)
    {
        //静音.
        g_ulFlags |= FLAG_MUTE_UPDATE | FLAG_MUTED;
    }
    else
    {
        // 取消静音.
        g_ulFlags &= ~(FLAG_MUTE_UPDATE | FLAG_MUTED);
        g_ulFlags |= FLAG_MUTE_UPDATE;
    }
    break;
}
//音量控制.
case USBD_AUDIO_EVENT_VOLUME:
{
    g_ulFlags |= FLAG_VOLUME_UPDATE;
    //最大音量.
    if(ulMsgParam == 0x8000)
    {
        //设置为最小
        g_sVolume = 0;
    }
    else
    {
        //声音控制器，设置音量.
        g_sVolume = (short)ulMsgParam - (short)VOLUME_MIN;
    }
    break;
}
// 断开连接.
case USB_EVENT_DISCONNECTED:
{
      GPIOPinWrite(GPIO_PORTF_BASE,0x10,0x00);
```

```
                GPIOPinWrite(GPIO_PORTF_BASE,0x80,0x00);
                g_ulFlags &= ~FLAG_CONNECTED;
                break;
            }
             //连接
             case USB_EVENT_CONNECTED:
             {
                   GPIOPinWrite(GPIO_PORTF_BASE,0x80,0x80);
                   g_ulFlags |= FLAG_CONNECTED;
                   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 LED 进行指示。原理图如图 3 所示：

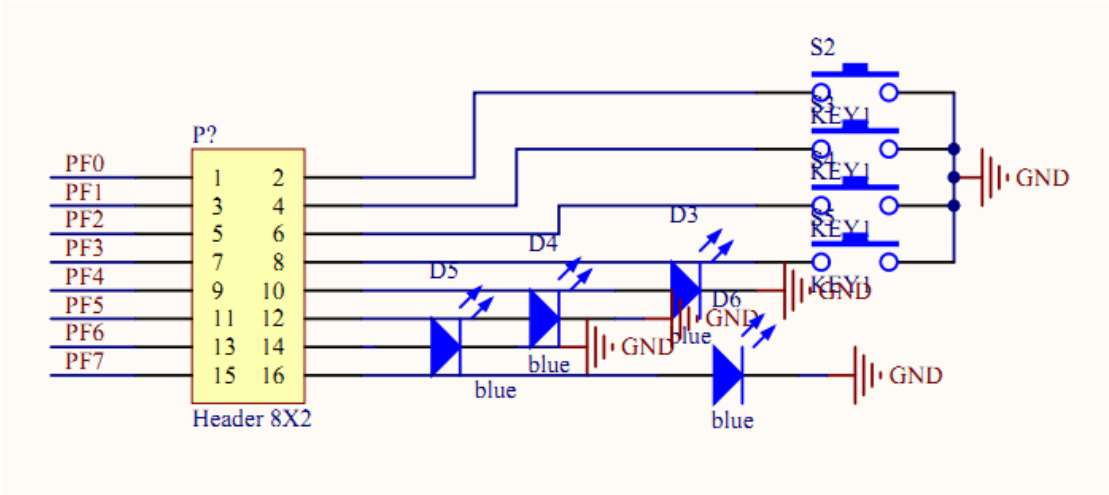

图 3

系统初始化：

```
//设置内核电压、主频 50Mhz
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;
// 全局状态标志.
g_ulFlags = 0;
// 初始化 Audio 设备.
```

```
g_pvAudioDevice = USBDAudioInit(0, (tUSBDAudioDevice *)&g_sAudioDevice);
```

第四步：使能、配置 DMA，Audio 设备要传输大量数据，所以 USB 库函数内部已经使用了 DMA，在使用前必须使能、配置 DMA。

```
//配置使能 DMA
SysCtlPeripheralEnable(SYSCTL_PERIPH_UDMA);
SysCtlDelay(10);
uDMAControlBaseSet(&sDMAControlTable[0]);
uDMAEnable();
```

第五步：数据处理。主要使用 USBAudioBufferOut 从输出端点中获取数据并处理，并且进行 Audio 设备控制。

```
while(1)
{
    //等待连接结束.
    while((g_ulFlags & FLAG_CONNECTED) == 0)
    {
    }
     //初始化 Buffer
    g_sBuffer.pucFill = g_sBuffer.pucBuffer;
    g_sBuffer.pucPlay = g_sBuffer.pucBuffer;
    g_sBuffer.ulFlags = 0;
     //从 Audio 设备类中获取数据
    if(USBAudioBufferOut(g_pvAudioDevice,
                         (unsigned char *)g_sBuffer.pucFill,
                         AUDIO_PACKET_SIZE, USBBufferCallback) == 0)
    {
        //标记数据放入 buffer 中.
        g_sBuffer.ulFlags |= SBUFFER_FLAGS_FILLING;
    }
    //设备连接到主机.
    while(g_ulFlags & FLAG_CONNECTED)
    {
        // 检查音量是否有改变.
        if(g_ulFlags & FLAG_VOLUME_UPDATE)
        {
            // 清除更新音量标志.
            g_ulFlags &= ~FLAG_VOLUME_UPDATE;
            // 修改音量，自行添加代码.在此以 LED 灯做指示。
            //UpdateVolume();
               GPIOPinWrite(GPIO_PORTF_BASE,0x40,~GPIOPinRead(GPIO_PORTF_BASE,0x40));
        }
        //是否静音
        if(g_ulFlags & FLAG_MUTE_UPDATE)
        {
```

```
                //修改静音状态，自行添加函数.在此以 LED 灯做指示。
                //UpdateMute();
                   if(g_ulFlags & FLAG_MUTED)
                        GPIOPinWrite(GPIO_PORTF_BASE,0x20,0x20);
                   else
                        GPIOPinWrite(GPIO_PORTF_BASE,0x20,0x00);
                // 清除静音标志
                g_ulFlags &= ~FLAG_MUTE_UPDATE;
            }
        }
    }
    //*****************************************************************************
    //USBAudioBufferOut 的 Callback 入口参数
    //*****************************************************************************
    void USBBufferCallback(void *pvBuffer, unsigned long ulParam, unsigned long ulEvent)
    {
     //数据处理，自行加入代码。
     // Your Codes .........
        //再一次获取数据.
        USBAudioBufferOut(g_pvAudioDevice, (unsigned char *)g_sBuffer.pucFill,
                          AUDIO_PACKET_SIZE, USBBufferCallback);
    }
```

使用上面五步就完成 Audio 设备开发。Audio 设备开发时要加入两个 lib 库函数：usblib.lib 和 DriverLib.lib，在启动代码中加入 USB0DeviceIntHandler 中断服务函数。以上 Audio 设备开发完成，在 Win xp 下运行效果如下图所示：

在枚举过程中可以看出，在电脑右下脚可以看到“Audio Example”字样，标示正在进行枚举。枚举成功后，在“设备管理器”的“声音、视屏和游戏控制器”中看到“USB Audio Device”设备，如下图。现在 Audio 设备可以正式使用。

Audio 设备开发源码如下：

```
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_sysctl.h"
#include "inc/hw_udma.h"
#include "inc/hw_gpio.h"
#include "driverlib/gpio.h"
#include "driverlib/interrupt.h"
#include "driverlib/sysctl.h"
#include "driverlib/udma.h"
#include "usblib/usblib.h"
#include "usblib/usb-ids.h"
#include "usblib/device/usbdevice.h"
#include "usblib/device/usbdaudio.h"

//根据具体 Audio 芯片修改
#define VOLUME_MAX              ((short)0x0C00)  // +12db
#define VOLUME_MIN              ((short)0xDC80)  // -34.5db
#define VOLUME_STEP             ((short)0x0180)  // 1.5db
//Audio 设备
const tUSBDAudioDevice g_sAudioDevice;
//DMA 控制
tDMAControlTable sDMAControlTable[64] __attribute__ ((aligned(1024)));
//*****************************************************************************
// 缓存与标志.
//*****************************************************************************
#define AUDIO_PACKET_SIZE       ((48000*4)/1000)
#define AUDIO_BUFFER_SIZE       (AUDIO_PACKET_SIZE*20)
#define SBUFFER_FLAGS_PLAYING   0x00000001
#define SBUFFER_FLAGS_FILLING   0x00000002
struct
{
    //主要 buffer，USB audio class 和 sound driver 使用.
    volatile unsigned char pucBuffer[AUDIO_BUFFER_SIZE];
    // play pointer.
    volatile unsigned char *pucPlay;
    // USB fill pointer.
    volatile unsigned char *pucFill;
    // 采样率 调整.
    volatile int iAdjust;
    // 播放状态
    volatile unsigned long ulFlags;
} g_sBuffer;
//*****************************************************************************
```

```
// 当前音量
//*****************************************************************************
short g_sVolume;
//*****************************************************************************
// 通过 USBDAudioInit() 函数，完善 Audio 设备配置信息
//*****************************************************************************
void *g_pvAudioDevice;
// 音量更新
#define FLAG_VOLUME_UPDATE      0x00000001
// 更新静音状态
#define FLAG_MUTE_UPDATE        0x00000002
// 静音状态
#define FLAG_MUTED              0x00000004
// 连接成功
#define FLAG_CONNECTED          0x00000008
volatile unsigned long g_ulFlags;
extern unsigned long
AudioMessageHandler(void *pvCBData, unsigned long ulEvent,
                    unsigned long ulMsgParam, void *pvMsgData);
//*****************************************************************************
// 设备语言描述符.
//*****************************************************************************
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
};
#define NUM_STRING_DESCRIPTORS (sizeof(g_pStringDescriptors) /                \
                                sizeof(unsigned char *))
//*****************************************************************************
// 定义 Audio 设备实例
//*****************************************************************************
static tAudioInstance g_sAudioInstance;
//*****************************************************************************
// 定义 Audio 设备类
//*****************************************************************************
const tUSBDAudioDevice g_sAudioDevice =
{
    // VID
    USB_VID_STELLARIS,
    // PID
    USB_PID_AUDIO,
    // 8 字节供应商字符串.
    "TI      ",
    //16 字节产品字符串.
    "Audio Device    ",
    //4 字节版本字符串.
    "1.00",
    500,
    USB_CONF_ATTR_SELF_PWR,
    AudioMessageHandler,
    g_pStringDescriptors,
    NUM_STRING_DESCRIPTORS,
    VOLUME_MAX,
    VOLUME_MIN,
    VOLUME_STEP,
    &g_sAudioInstance
};
//*****************************************************************************
//USB Audio 设备类返回事件处理函数（callback）.
//*****************************************************************************
unsigned long AudioMessageHandler(void *pvCBData, unsigned long ulEvent,
                    unsigned long ulMsgParam, void *pvMsgData)
{
    switch(ulEvent)
    {
         //Audio 正处于空闲或者工作状态。
        case USBD_AUDIO_EVENT_IDLE:
        case USBD_AUDIO_EVENT_ACTIVE:
```

```
{
      GPIOPinWrite(GPIO_PORTF_BASE,0x10,0x10);
    g_ulFlags |= FLAG_CONNECTED;
    break;
}
// 静音控制.
case USBD_AUDIO_EVENT_MUTE:
{
    // 检查是否静音.
    if(ulMsgParam == 1)
    {
        //静音.
        g_ulFlags |= FLAG_MUTE_UPDATE | FLAG_MUTED;
    }
    else
    {
        // 取消静音.
        g_ulFlags &= ~(FLAG_MUTE_UPDATE | FLAG_MUTED);
        g_ulFlags |= FLAG_MUTE_UPDATE;
    }
    break;
}
//音量控制.
case USBD_AUDIO_EVENT_VOLUME:
{
    g_ulFlags |= FLAG_VOLUME_UPDATE;
    //最大音量.
    if(ulMsgParam == 0x8000)
    {
        //设置为最小
        g_sVolume = 0;
    }
    else
    {
        //声音控制器，设置音量.
        g_sVolume = (short)ulMsgParam - (short)VOLUME_MIN;
    }
    break;
}
// 断开连接.
case USB_EVENT_DISCONNECTED:
{
      GPIOPinWrite(GPIO_PORTF_BASE,0x10,0x00);
      GPIOPinWrite(GPIO_PORTF_BASE,0x80,0x00);
```

```
            g_ulFlags &= ~FLAG_CONNECTED;
              break;
        }
         case USB_EVENT_CONNECTED:
         {
               GPIOPinWrite(GPIO_PORTF_BASE,0x80,0x80);
               g_ulFlags |= FLAG_CONNECTED;
               break;
         }
        default:
        {
            break;
        }
    }
    return(0);
}
//*****************************************************************************
//USBAudioBufferOut 的 Callback 入口参数
//*****************************************************************************
void USBBufferCallback(void *pvBuffer, unsigned long ulParam, unsigned long ulEvent)
{
     //数据处理，自行加入代码。
     // Your Codes .........
    //再一次获取数据.
    USBAudioBufferOut(g_pvAudioDevice, (unsigned char *)g_sBuffer.pucFill,
                      AUDIO_PACKET_SIZE, USBBufferCallback);
}

//*****************************************************************************
// 应用主函数.
//*****************************************************************************
int   main(void)
{
    //设置内核电压、主频 50Mhz
     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;

    //配置使能 DMA
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UDMA);
```

```
SysCtlDelay(10);
uDMAControlBaseSet(&sDMAControlTable[0]);
uDMAEnable();

// 全局状态标志.
g_ulFlags = 0;
// 初始化 Audio 设备.
g_pvAudioDevice = USBDAudioInit(0, (tUSBDAudioDevice *)&g_sAudioDevice);
while(1)
{
    //等待连接结束.
    while((g_ulFlags & FLAG_CONNECTED) == 0)
    {
    }
     //初始化 Buffer
    g_sBuffer.pucFill = g_sBuffer.pucBuffer;
    g_sBuffer.pucPlay = g_sBuffer.pucBuffer;
    g_sBuffer.ulFlags = 0;
     //从 Audio 设备类中获取数据
    if(USBAudioBufferOut(g_pvAudioDevice,
                         (unsigned char *)g_sBuffer.pucFill,
                         AUDIO_PACKET_SIZE, USBBufferCallback) == 0)
    {
        //标记数据放入 buffer 中.
        g_sBuffer.ulFlags |= SBUFFER_FLAGS_FILLING;
    }
    //设备连接到主机.
    while(g_ulFlags & FLAG_CONNECTED)
    {
        // 检查音量是否有改变.
        if(g_ulFlags & FLAG_VOLUME_UPDATE)
        {
            // 清除更新音量标志.
            g_ulFlags &= ~FLAG_VOLUME_UPDATE;
            // 修改音量，自行添加代码.在此以 LED 灯做指示。
            //UpdateVolume();
            GPIOPinWrite(GPIO_PORTF_BASE,0x40,~GPIOPinRead(GPIO_PORTF_BASE,0x40));
        }
        //是否静音
        if(g_ulFlags & FLAG_MUTE_UPDATE)
        {
            //修改静音状态，自行添加函数.在此以 LED 灯做指示。
            //UpdateMute();
               if(g_ulFlags & FLAG_MUTED)
```

```
                    GPIOPinWrite(GPIO_PORTF_BASE,0x20,0x20);
                else
                    GPIOPinWrite(GPIO_PORTF_BASE,0x20,0x00);
                // 清除静音标志
                g_ulFlags &= ~FLAG_MUTE_UPDATE;
            }
        }
    }
}
```