e Semiconductor
应用笔记

Document Number: AN4665
Rev 0, 01/2013

如何使用飞思卡尔 USB 协议栈来设置音频类设备

作者: Wang Hao

内容

1 简介

USB 接口非常适合传输音频，从低保真语音连接到高质量多通道音频流。很多应用都可以充分利用 USB 的音频功能，从通信到娱乐，再到音乐录制和播放。

音频设备类规范实现了音频传输机制的标准化，让软件驱动程序尽可能具有通用性。该规范指定了在每个 USB 音频功能中必须存在的标准描述符和类特定描述符，并说明了实现完全音频功能控制的类特定请求的使用。它还定义了可寻址实体，例如单元和终端，它们用于描述音频功能拓扑，并提供操作音频功能的物理属性的接口。

从头开始开发音频类设备应用程序是一项艰巨的任务；但是，飞思卡尔提供了裸机 USB 协议栈，它支持很多常用 USB 设备类，例如个人医疗保健设备类（PHDC）、人机接口设备（HID）、大容量存储设备（MSD）、通信设备类（CDC）和音频类。源代码是免费的，可以移植且易于使用，可从 **freescale.com/medicalUSB** 下载。

2 音频类设备要求

很多情况下，音频功能不作为独立设备存在。它是一种功能与其他功能组合在一起，构成了“复合”设备。音频功能位于设备类分层结构的接口级别。

2.1 音频功能概述

每个音频功能必须具有单个 AudioControl 接口，可以具有零个或更多 AudioStreaming 接口，以及零个或更多 MIDIStreaming 接口。AudioControl 接口用于访问功能的音频控件，例如音量控制，而 AudioStreaming 接口则用于将音频流传入或传出音频功能。MIDIStreaming 接口可用于将 MIDI 数据流传入和传出音频功能。请参见下图，从 USB 总线接口的角度查看音频功能的全局概述。

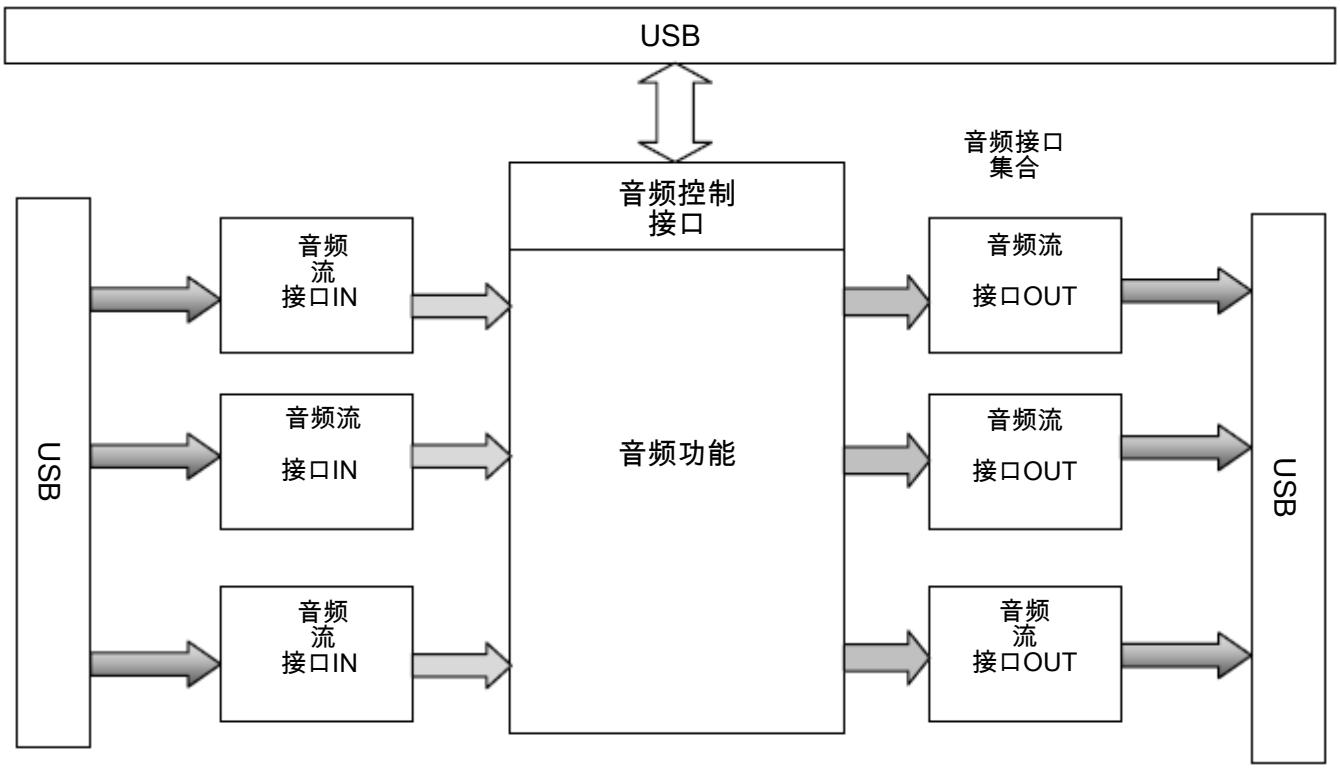

图 1. 音频功能全局概述

属于同一音频功能的单个 AudioControl 接口、AudioStreaming 接口和 MIDIStreaming 接口集合称为音频接口集合 (AIC)。一个设备可以同时具有多个激活的 AIC。这些集合用于控制同一复合设备中的多个独立音频功能。

2.2 音频功能拓扑

为了简单地表示拓扑，并操作音频功能的物理属性，我们定义了两种类型的通用实体，分别称为“单元”和“终端”。我们还引入了音频通道群集的概念，也就是将一组音频通道组合在一起。下面几小节将简要介绍所有这些实体。

2.2.1 单元

单元提供了用于完全描述大部分音频功能的基本构件块。音频功能是通过将多个此类单元连接在一起构建的。单元具有一个或多个输入引脚以及单个输出引脚，其中的每个引脚代表音频功能内部的一个逻辑音频通道群集。根据所需的拓扑，连接单元的 I/O 引脚，从而将单元连接在一起。

功能单元是常用的多通道处理单元，提供对输入逻辑通道上的多个单参数音频控件的基本操作，例如静音、音量控制等。

功能单元描述符报告有哪些控件可用于功能单元中的每个通道和“主”通道。

2.2.2 终端

介绍两种类型的终端。

- 输入终端：此实体代表音频功能内部的音频通道的起点。输入终端的功能是代表输入音频数据的源，此前这些数据已从原始音频流正确提取到嵌入在此音频流内的单独逻辑通道中。
- 输出终端：此实体代表音频通道的终点。USB 端点是输入或输出终端的典型示例。输出终端的功能是代表输出音频数据的接收端，此后会将这些数据从原始的单独逻辑通道打包为输出音频流。

2.2.3 音频通道群集

音频通道群集是传输密切关联的同步音频信息的音频通道分组。

音频通道群集的特征仅包括两个属性：

- 群集中的音频通道的数量
- 群集中的每个音频通道的空间位置，例如左通道和右通道

有两种类型的音频通道群集：

- 逻辑群集描述音频功能内的音频数据，其中的音频通道被视为逻辑概念。
- 物理群集描述 AudioStream 接口内的音频数据，处理音频流中的实际物理音频通道。

2.3 描述符

对于 USB 设备，描述符可向主机完全描述其功能。对于任何类型的设备而言，设备描述符和配置描述符都是相似的，因此本节中仅介绍其他一些描述符。

以下讨论基于与 USB 协议栈一同提供的 *audio_speaker* 演示；这些描述符在 *usb_descriptor.c* 中定义，并符合“USB 音频设备类规范 2.0”。

2.3.1 标准接口关联描述符（IAD）

标准 USB 接口关联机制用于描述音频接口集合，也就是将这些接口绑定在一起。以下代码行显示有两个接口，其中一个是 AudioControl 接口，另一个是 AudioStream 接口。

```
/* Standard Interface Association Descriptor */
0x08,                                   /* bLength(0x08) */
USB_INTERFACE_ASSOCIATION_DESCRIPTOR,   /* bDescriptorType(0x0B) */
0x00,                                   /* bFirstInterface(0x00) */
0x02,                                   /* bInterfaceCount(0x02) */
0x01,                                   /* bFunctionClass(0x01): AUDIO */
0x00,                                   /* bFunctionSubClass(0x00) */
0x20,                       /* bFunctionProtocol(0x2000): 2.0 AF_VERSION_02_00 */
0x00,                                   /* iFunction(0x00) */
```

2.3.2 标准 AudioControl（音频控制）接口描述符

标准音频控制接口描述符与在“USB 2.0 规范”的第 9 章中定义的标准接口描述符相同，该规范可从 **usb.org** 下载。在以下代码中，*bNumEndpoints* 字段为 0，这意味着仅为 AudioControl 接口提供默认控制通道，而没有可选的中断端点。

```
/* AUDIO CONTROL Interface */
/* Standard AC Interface Descriptor(4.7.1) */
0x09,       /* bLength(0x09) */
0x04,       /* bDescriptorType(0x04): INTERFACE */
0x00,       /* bInterfaceNumber(0x00) */
0x00,       /* bAlternateSetting(0x00) */
0x00,       /* bNumEndpoints(0x00) */
0x01,       /* bInterfaceClass(0x01): AUDIO */
0x01,       /* bInterfaceSubClass(0x01): AUDIOCONTROL */
0x20,       /* bInterfaceProtocol(0x20): IP 2.0 IP_VERSION_02_00 */
0x07,       /* iInterface(0x07): Not Requested */
```

2.3.3 类特定的音频控制描述符

类特定的 AudioControl 接口描述符是用于完全描述音频功能的所有描述符 (也就是所有“单元”和“终端”描述符) 的级联。

类特定的音频控制接口描述符的总长度取决于音频功能中的单元和终端的数量。因此，描述符的开头是一个标头，在 *wTotalLength* 字段中反映类特定的完整音频控制接口描述符的总长度，以字节为单位。请参见以下部分提供的代码。

标头描述符后面有一个或更多单元和/或终端描述符。音频功能中的每个单元和终端都分配了唯一标识编号，即单元 ID 或终端 ID。除了唯一标识音频功能中的所有可寻址单元之外，这些 ID 还可以描述音频功能的拓扑，也就是说，单元或终端描述符的 *bSourceID* 字段指示此单元或终端连接到其他哪个单元或终端。

2.3.3.1 类特定的 AS 接口标头描述符

```
/* Class-Specific AC Interface Header Descriptor(4.7.2) */
0x09,           /* bLength(0x09) */
0x24,           /* bDescriptorType(0x24): CS_INTERFACE */
0x01,           /* bDescriptorSubType(0x01): HEADER */
0x00, 0x02,     /* bcdADC(0x0200): 2.0 */
0x01,           /* bCategory(0x01): DESKTOP_SPEAKER */
//0x40, 0x00,     /* wTotalLength(64): 9 + 8 + 17 + 18 + 12 (2 channels)
0x3C, 0x00,     /* wTotalLength(60): 9 + 8 + 17 + 14 + 12 (1 channel)
                        Audio Control Interface size */
0x00,           /* bmControls(0b00000000) */
```

2.3.3.2 时钟源描述符

在以下代码中，*bmAttributes* 字段为 0x01，这意味着时钟类型为内部固定时钟。

```
/* Clock Source Descriptor(4.7.2.1) */
0x08,       /* bLength(0x08) */
0x24,       /* bDescriptorType(0x24): CS_INTERFACE */
0x0A,       /* bDescriptorSubType(0x0A): CLOCK_SOURCE */
0x10,       /* bClockID(0x10): CLOCK_SOURCE_ID */
0x01,       /* bmAttributes(0x01): internal fixed clock */
0x07,       /* bmControls(0x07):
                    clock frequency control: 0b11 - host programmable;
                    clock validity control: 0b01 - host read only */
0x00,       /* bAssocTerminal(0x00) */
0x01,       /* iClockSource(0x01): Not requested */
```

2.3.3.3 输入终端描述符

输入终端描述符（ITD）向主机提供有关输入终端的功能方面的信息。

在以下代码中，*wTerminalType* 字段为 0x0101，这意味着输入终端正在处理通过 AudioStreaming 接口中的端点传输的信号；*bCSourceID* 字段标识此输入终端的时钟源。

bNrChannels、*bmChannelConfig* 和 *iChannelNames* 字段共同构成了群集描述符，这里的音频通道群集是一组音频通道，传输紧密关联的同步音频信息。

```
/* Input Terminal Descriptor(4.7.2.4) */
0x11,           /* bLength(0x11): 17 */
0x24,           /* bDescriptorType(0x24): CS_INTERFACE */
0x02,           /* bDescriptorSubType(0x02): INPUT_TERMINAL */
0x20,           /* bTerminalID(0x20): INPUT_TERMINAL_ID */
0x01, 0x01,     /* wTerminalType(0x0101): USB streaming */
0x00,           /* bAssocTerminal(0x00) */
0x10,           /* bCSourceID(0x10): CLOCK_SOURCE_ID */
NB_CHANNELS,    /* bNrChannels(0x01) */
0x00, 0x00, 0x00, 0x00, /* bmChannelConfig(0x00): Mono, no spatial location */
0x00,           /* iChannelNames */
0x00, 0x00,     /* bmControls(0x0000) */
0x02,           /* iTerminal(0x02): not requested */
```

2.3.3.4 功能单元描述符

在以下代码中，*bSourceID* 字段为 0x20，这意味着功能单元将输入终端作为源； *bmaControls* 字段为 0x0000_000F，这表示用户能够通过 AudioControl 请求，更改此音频设备的音量或使其静音。

```
/* Feature Unit Descriptor(4.7.2.8) */
0x0E,           /* bLength(0x0E): 6 + (ch + 1) * 4, 1 channel */
0x24,           /* bDescriptorType(0x24): CS_INTERFACE */
0x06,           /* bDescriptorSubType(0x06): FEATURE_UNIT */
0x30,           /* bUnitID(0x30): FEATURE_UNIT_ID */
0x20,           /* bSourceID(0x20): INPUT_TERMINAL_ID */
0x0F, 0x00, 0x00, 0x00, /* bmaControls(0)(0x0000000F): Master Channel 0
                                0b11: Mute read/write
                                0b11: Volume read/write */
0x00, 0x00, 0x00, 0x00, /* bmaControls(1)(0x00000000): Logical Channel 1
*/
0x00,           /* iFeature(0x00) */
```

2.3.3.5 输出终端描述符

在以下代码中，*wTerminalType* 字段设置为 0x0101，表示 USB 流终端类型，*bSourceID* 设置为功能单元的 ID，意味着此输出终端在功能单元之后连接。

```
/* Output Terminal Descriptor(4.7.2.5) */
0x0C,           /* bLength(12) */
0x24,           /* bDescriptorType(0x24): CS_INTERFACE */
0x03,           /* bDescriptorSubType(0x03): OUTPUT_TERMINAL */
0x40,           /* bTerminalID(0x40) */
0x01, 0x01,     /* wTerminalType(0x0101): USB_STREAMING */
0x00,           /* bAssocTerminal(0x00): no association */
0x30,           /* bSourceID(0x30): FEATURE_UNIT_ID */
0x10,           /* bCSourceID(0x10): CLOCK_SOURCE_ID */
0x00, 0x00,     /* bmControls(0x0000) */
0x00,           /* iTerminal(0x00): Not Requested */
```

基于上述描述符，音频扬声器的内部拓扑显示在下图中。

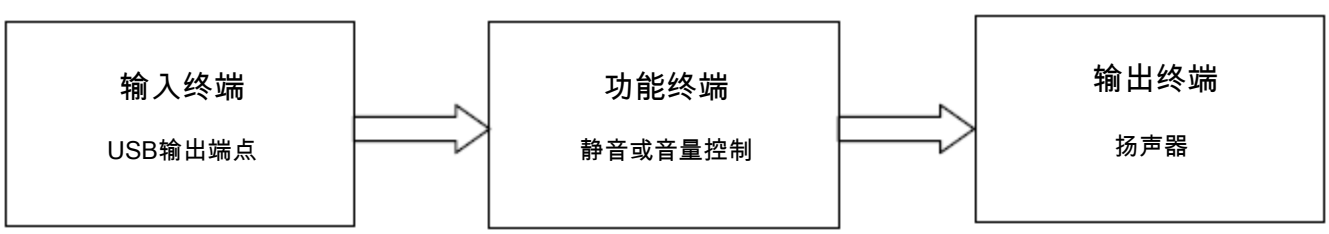

图 2. 音频扬声器内部拓扑

2.3.4 标准 AudioStream（音频流）接口描述符

在以下代码中，我们为此接口定义了两种备选设置。

- 零带宽设置，*bNumEndpoints* 字段设置为 0：在没有使用音频功能时，使用此设置可以放弃总线上的请求带宽。
- *bNumEndpoints* 字段设置为 2：它表示此接口具有数据端点，以及用于同步的显式反馈端点。

```
/* AUDIO STREAMING Interface */
/* Standard AS Interface Descriptor(4.9.1) */
/* Interface 1, Alternate 0 */
/* default alternate setting with 0 bandwidth */
0x09,       /* bLength(9) */
0x04,       /* bDescriptorType(0x04): INTERFACE */
0x01,       /* bInterfaceNumber(0x01) */
0x00,       /* bAlternateSetting(0x00) */
0x00,       /* bNumEndpoints(0x00) */
0x01,       /* bInterfaceClass(0x01): AUDIO */
0x02,       /* bInterfaceSubClass(0x02): AUDIOSTREAMING */
0x20,       /* bInterfaceProtocol(0x20): IP 2.0 */
0x08,       /* iInterface */

/* Standard AS Interface Descriptor(4.9.1) */
/* INterface 1, Alternate 1 */
/* alternate interface for data streaming */
0x09,       /* bLength(9) */
0x04,       /* bDescriptorType(0x04): INTERFACE */
0x01,       /* bInterfaceNumber(0x01) */
0x01,       /* bAlternateSetting(0x01) */
0x02,       /* bNumEndpoints(0x02) */
0x01,       /* bInterfaceClass(0x01): AUDIO */
0x02,       /* bInterfaceSubClass(0x02): AUDIO_STREAMING */
0x20,       /* bInterfaceProtocol(0x20): IP 2.0 */
0x09,       /* iInterface */
```

2.3.5 类特定的音频流接口描述符

在以下代码中，*bTerminalLink* 字段设置为输入终端的 ID，这意味着 AudioStreaming 接口连接到输入终端。*bFormatType* 和 *bmFormats* 字段共同定义了通过 AudioStreaming 接口的音频数据格式为常用 PCM 数据。*bNrChannels* 和 *bmChannelConfig* 字段定义了音频流接口中的物理音频通道群集，包括左通道和右通道。

```
/* Class-Specific AS Interface Descriptor(4.9.2) */
0x10,           /* bLength(16) */
0x24,           /* bDescriptorType(0x024): CS_INTERFACE */
0x01,           /* bDescriptorSubType(0x01): AS_GENERAL */
0x20,           /* bTerminalLink(0x20): INPUT_TERMINAL_ID */
0x00,           /* bmControls(0x00) */
0x01,           /* bFormatType(0x01): FORMAT_TYPE_I */
0x01, 0x00, 0x00, 0x00, /* bmFormats(0x00000001): PCM */
0x02,           /* bNrChannels(0x02): NB_CHANNELS */
0x03, 0x00, 0x00, 0x00, /* bmChannelConfig(0x00000003) */
0x00,           /* iChannelNames(0x00): None */
```

2.3.6 类型 I 格式类型描述符

在以下代码中，我们可以看到音频采样为 24 位，采样率为 8 kHz。

```
/* Type I Format Type Descriptor(2.3.1.6 - Audio Formats) */
0x06,           /* bLength(6) */
0x24,           /* bDescriptorType(0x24): CS_INTERFACE */
0x02,           /* bDescriptorSubtype(0x02): FORMAT_TYPE */
```

```
0x01,           /* bFormatType(0x01): FORMAT_TYPE_I */
0x04,           /* bSubSlotSize(0x01) */
0x18,           /* bBitResolution(0x18): 24 bits per sample */
0x01,                           /* One frequency supported */
0x40,0x1F,0x00,                 /* 8 kHz */
```

2.3.7 标准音频流同步数据端点描述符

在音频扬声器演示中，需要一个 OUT 端点，用于接收来自主机的音频流数据。在代码中，*bmAttributes* 为 0x05，它表示这是一个同步端点，同步类型为异步。

```
/* Standard AS Isochronous Audio Data Endpoint Descriptor(4.10.1.1) */
0x07,       /* bLength(7) */
0x05,       /* bDescriptorType(0x05): ENDPOINT_DESCRIPTOR */
EP01_OUT,   /* bEndpointAddress(0x01) */
0x05,       /* bmAttributes(0x05): iso+asynch+data */
0x08, 0x00, /* wMaxPacketSize(0x0008): 8(8 samples * 1 bytes * 1 channel) */
#if HIGH_SPEED_DEVICE
0x04,       /* bInterval(0x04): 2^x ms */
#else
0x01,       /* bInterval(0x01): 2^x ms */
#endif
```

2.3.8 类特定的音频流同步数据端点描述符

在以下代码中，*bLockDelayUnits* 和 *wLockDelay* 字段用于向主机指示此端点的时钟恢复电路需要花费多长时间来锁定和可靠地生成或使用音频数据流。

```
/* Class-Specific AS IsochronousAudio Data Endpoint Descriptor(4.10.1.2) */
0x08,           /* bLength(8) */
0x25,           /* bDescriptorType(0x25): CS_ENDPOINT */
0x01,           /* bDescriptorSubtype(0x01): EP_GENERAL */
0x00,           /* bmAttributes(0x00): MaxPacketsOnly = FALSE */
0x00,           /* bmControls(0x00) */
0x00,           /* bLockDelayUnits(0x00) */
0x00, 0x00,     /* wLockDelay(0x0000) */
```

2.3.9 标准音频流同步反馈端点描述符

在以下代码中，反馈端点为 IN 端点，*bmAttributes* 字段设置为 0x11，这意味着传输类型为同步，使用类型为反馈端点。

```
/* Standard AS Isochronous Audio Data Endpoint Descriptor(4.10.1.1) */
 0x07,           /* bLength(7) */
 0x05,           /* bDescriptorType(0x05): ENDPOINT_DESCRIPTOR */
 EP02_IN,        /* bEndpointAddress(0x82) */
 0x11,           /* bmAttributes(0x11): iso+feedback */
 0x04, 0x00,     /* wMaxPacketSize(0x0004) */
#if HIGH_SPEED_DEVICE
 0x04,           /* bInterval(0x04): 2^x ms */
#else
 0x01,           /* bInterval(0x01): 2^x ms */
#endif
```

2.4 类特定的请求

类特定的请求用于设置和获取与音频相关的控件。这些控件分为两组：用于操作音频功能的控件（例如音量、音调、选择器位置等）；用于影响在同步端点上的数据传输（例如当前采样频率）。

- AudioControl 请求：对音频功能的控制是通过对各个控件的属性的操作来执行的，这些控件嵌入在功能单元之类的音频功能实体中。
- AudioStreaming 请求：对 AudioStreaming 接口的类特定行为的控制是通过对接口控件或端点控件的操作来执行的。

2.4.1 控件属性

以下是实体的当前定义控件属性。

- 当前设置属性，用于操作控件的当前实际设置
- 范围属性，实际包含一系列属性，包括最小值、最大值和分辨率。

2.4.2 控制请求布局

请求布局遵循在 USB 2.0 规范中定义的标准请求布局，该规范可从 **usb.org** 下载。参见下表。

表 1. 控制请求布局

bmRequestType	bRequest	wValue	wIndex	wLength
0010_0001B 1010_0001B	当前范围	控件选择器和通道编号	实体 ID 和接口	参数块长度
0010_0010B 1010_0010B			端点	

从表 1 可以看到，请求能够定向到音频功能的接口（AudioControl 或 AudioStreaming），或 AudioStreaming 接口的同步端点。

wValue 字段在高位字节中指定控件选择器（CS），在低位字节中指定通道编号（CN）。控件选择器指示此请求正在操作哪种类型的控制。通道编号（CN）指示群集的哪一个逻辑通道将会受到影响。

每个不同类型的实体或单元都会发出不同类型的控制请求，例如，对于功能单元，用户可能要对基础音频流进行静音控制和音量控制。

3 飞思卡尔 USB 协议栈

飞思卡尔 USB 协议栈分成几个层，帮助应用程序开发人员专注于开发应用程序，而无需担心与底层 USB 控制器和在“USB 2.0 规范”中定义的通用框架相关的通信，该规范可从 **usb.org** 下载。

从音频类设备的角度来看，图 3 是飞思卡尔 USB 协议栈的分层架构。

- 应用程序层：除了提供代码来设置特定 USB 功能（例如音频扬声器）之外，应用程序层还需要提供 *usb_descriptor.c* 文件，以描述符方式向主机通知其功能。
- 类层包括两个子层，一个子层针对每个特定 USB 设备类（例如音频类、CDC 类），并且包括类特定的请求的设置；另一个子层则包括不同类之间的相同操作，例如为 USB 事件（例如 USB 复位、挂起和恢复事件）注册回调函数。

- 框架层为默认控制通道设置服务，这些服务符合“USB 2.0 规范”第 9 章的要求，该规范可从 usb.org 下载。
- 设备层包括为用于 USB 通信的基础 USB 控制器编程的代码。

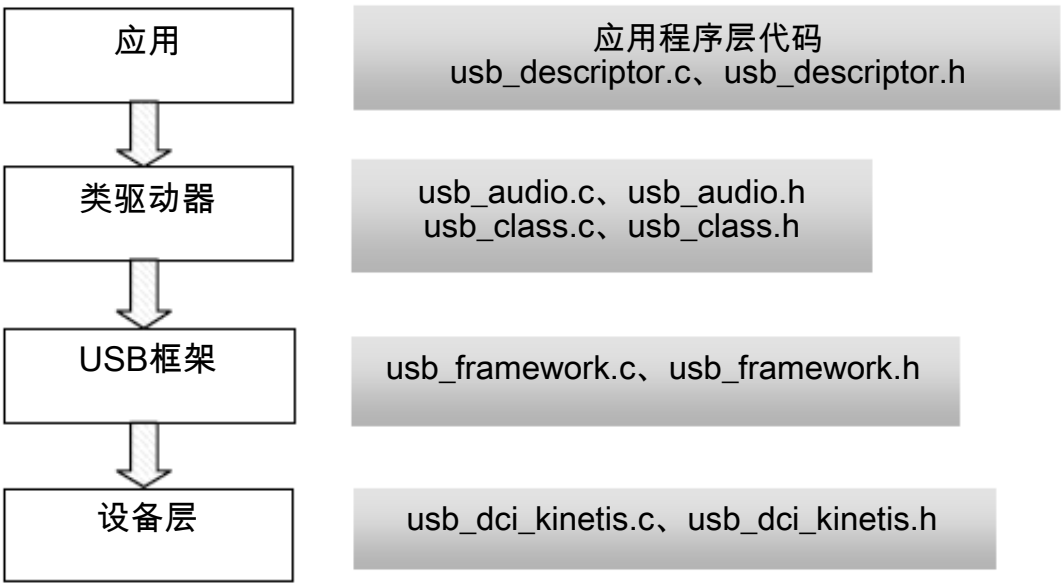

图 3. 飞思卡尔 USB 协议栈架构

下表列出了飞思卡尔 USB 协议栈中的针对类层和 USB 框架层的 API。

表 2. 针对类层和框架层的 API

文件名	API	说明
usb_audio.c	USB_Class_Audio_Init	• 使用*_usb_device_init* 初始化设备层 • 使用 *USB_Class_Init* 初始化通用类函数，并且注册类回调和其他请求回调函数
	USB_Class_Audio_Send_Data	内部调用 *USB_Class_Send_Data*
	USB_Class_Audio_Recv_Data	内部调用*_usb_device_recv_data*
	USB_Class_Audio_Event	这是一个类回调，当接收已完成枚举事件时，它会初始化音频端点，然后为中断通道和同步通道注册服务。
	USB_Other_Requests	这是另外一个请求回调，可处理音频类特定的请求，例如接口和端点的获取和设置请求。
	USB_Get_Request_Interface USB_Set_Request_Interface	这些 API 基于实体 ID 来处理接口级别的请求，并将请求路由到相应的实体。然后，它从设置数据包提取控制选择器，并调用相应的 set 或 get 控制函数。
	USB_Get_Request_Endpoint USB_Set_Request_Endpoint	这些 API 处理端点级别的请求，从设置数据包提取控制选择器，并调用相应的 set 或 get 控制函数。
usb_class.c	USB_Class_Init	内部调用 *USB_Framework_Init*，为 USB 事件（例如总线复位、SOF、挂起、恢复和停滞）注册服务。
	USB_Class_Send_Data	内部调用*_usb_device_send_data*

下一页继续介绍此表...

表 2. 针对类层和框架层的 API (继续)

文件名	API	说明
usb_framework.c	USB_Framework_Init	为默认控制通道注册服务。
	USB_Control_Service	处理标准 USB 请求或类特定请求。

4 音频类演示

我们提供飞思卡尔 USB 协议栈版本 4.0.3 中的音频设备类的现成演示，可从 **freescale.com** 下载。安装软件包之后，这些演示可在 Freescale USB Stack v4.0.3\Source\Device\app 下找到。提供两个演示：音频生成器和音频扬声器。

4.1 音频扬声器

音频扬声器的 USB 描述符已在音频类设备要求 中进行了讨论。音频扬声器演示的主要流程显示在以下代码中，其中 FTM0_CH0 设置为输出音频数据。用户需要外部连接低通滤波器和话筒，才能听到从 PC 发送的音频。要设置演示，请按照“飞思卡尔 USB 设备协议栈用户指南”中的“附录 G: USB 音频演示”中的说明操作，该文档在 **freescale.com** 上提供。

然后，它调用 *USB_Class_Audio_Init* API，通过设备层 API 初始化 USB 控制器，并为应用程序层注册回调函数；准确的回调为 *USB_App_Callback*。用户在此编写代码以响应从下层接收的不同事件，例如 *USB_APP_ENUM_COMPLETE*、*USB_APP_DATA_RECEIVED* 或 *USB_APP_SEND_COMPLETE*。

```
void TestApp_Init(void)
{
     sci_init(); //initialize default console for output
     pit1_init(); //initialize PIT timer for 0.1ms timeout
     pwm_init(); //initialize FTM0_CH0 (PTC1) to output the audio signal
     error = USB_Class_Audio_Init(CONTROLLER_ID,USB_App_Callback,
                                        NULL,NULL);
}
```

USB_App_Callback 使用 event_type 参数来确定发生了何种事件，并且相应地进行响应。当它获取 *USB_APP_DATA_RECEIVED* 事件时，它会将音频数据复制到本地缓冲区 *audio_data_recv*，这些数据最终将在该缓冲区中用于更新 PIT 定时器 ISR 中的 FTM0_CH0 的 PWM 占空因数。参见以下代码。

```
static void USB_App_Callback (
    uint_8 controller_ID, /* [IN] Controller ID */
    uint_8 event_type, /* [IN] value of the event */
    void* val /* [IN] gives the configuration value */
)
{
        if(event_type == USB_APP_BUS_RESET)
         {
             start_app=FALSE;
         }
         else if(event_type == USB_APP_ENUM_COMPLETE)
         {
             start_app=TRUE;
     #ifdef USE_FEEDBACK_ENDPOINT
            // Send initial rate control feedback (48Khz)
            USB_Class_Audio_Send_Data(controller_ID, AUDIO_FEEDBACK_ENDPOINT,
                                (uint_8_ptr)&feedback_data,
AUDIO_FEEDBACK_ENDPOINT_PACKET_SIZE);
     #endif // USE_FEEDBACK_ENDPOINT
            …
         }
```

```
        else if ((event_type == USB_APP_DATA_RECEIVED) && (TRUE == start_app))
        {
            (void)USB_Class_Audio_Recv_Data(controller_ID, AUDIO_ENDPOINT,
                                (uint_8_ptr)g_curr_recv_buf,
AUDIO_ENDPOINT_PACKET_SIZE);
             audio_event = USB_APP_DATA_RECEIVED;
             data_receive = (APP_DATA_STRUCT*)val;
              (void)memcpy(audio_data_recv, data_receive->data_ptr, data_receive->data_size);
        }
    #ifdef USE_FEEDBACK_ENDPOINT
        else if((event_type == USB_APP_SEND_COMPLETE) && (TRUE == start_app))
        {
           feedback_data <<= 14; // 10.14 format
           (void)USB_Class_Audio_Send_Data(controller_ID,
                               AUDIO_FEEDBACK_ENDPOINT,
                                       (uint_8_ptr)&feedback_data,
                                      AUDIO_FEEDBACK_ENDPOINT_PACKET_SIZE);
         }
    #endif
}
```

5 结论

飞思卡尔 USB 协议栈为开发 USB 应用程序提供了良好框架。它提供多个层，以便隐藏 USB 数据通信的底层详细信息，还提供适用于常见 USB 设备类的很多现有演示，用户可以根据自己的用途进行调整。

要开发音频类设备，用户需要知道此设备的内部拓扑、设备具有的终端和单元数量、它们的互连、设备具有的接口数量，以及它是独立设备还是复合设备。然后，用户可以按照“飞思卡尔 USB 协议栈”中的示例，写下设备的 USB 描述符。

另外，用户还必须编写应用程序回调，以便响应底层发送的事件，并且设置音频设备的独特功能。

6 参考

可从 **usb.org** 获取以下参考文档。

- USB 2.0 规范
- 音频设备的 USB 设备类定义
- 终端类型的 USB 设备类定义
- 音频数据格式的 USB 设备类定义

可从 **freescale.com** 获取以下参考文档。

- USBUG：USB 协议栈用户指南
- USBAPIRM：飞思卡尔 USB 协议栈和 PHDC 设备 API—参考手册

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number AN4665
Revision 0, 01/2013