MICROCHIP

AN2582

使用MPLAB® Harmony在PIC32 MCU上创建USB音频设备

简介

通用串行总线（Universal Serial Bus，USB）是连接不同电子设备的最常用接口。除主要的PC操作系统之外，各种嵌入式系统平台也支持USB。USB协议为传输数字音频数据提供了原生支持。这种支持及其易用性使USB成为互连数字音频设备的普遍选择。

开发USB音频应用会带来一些设计挑战，其中包括USB协议复杂性、数字音频数据同步、编解码器配置和主机操作系统兼容性。因此，开发USB音频应用可能需要耗费巨大的开发成本和时间。

本应用笔记讨论了USB音频设备类规范v1.0，并为使用Microchip的MPLAB® Harmony集成软件框架在基于PIC32的单片机上实现USB音频设备解决方案提供了指导。本应用笔记重点讨论包含在MPLAB Harmony中的USB耳麦应用的开发。应用项目位于MPLAB Harmony安装目录的`apps\audio\usb_headset`文件夹中。

本应用笔记包含以下主题：

- 功能模型：讨论由音频子系统和USB子系统组成的USB耳麦应用的功能模型
- 音频概述：说明与音频子系统有关的数字音频通信的基本概念
- USB音频概述：说明与USB子系统有关的USB音频通信的基本概念
- MPLAB Harmony概述：概述Microchip的MPLAB Harmony集成软件框架
- 使用MPLAB Harmony构建USB音频设备：提供使用MPLAB Harmony配置器（MPLAB Harmony Configurator，MHC）构建USB耳麦应用的详细步骤，并讨论MPLAB Harmony音频驱动程序和USB音频库的架构和使用
- USB音频应用注意事项：有关时钟调节和缓冲方案等音频应用特定主题的注意事项

目录

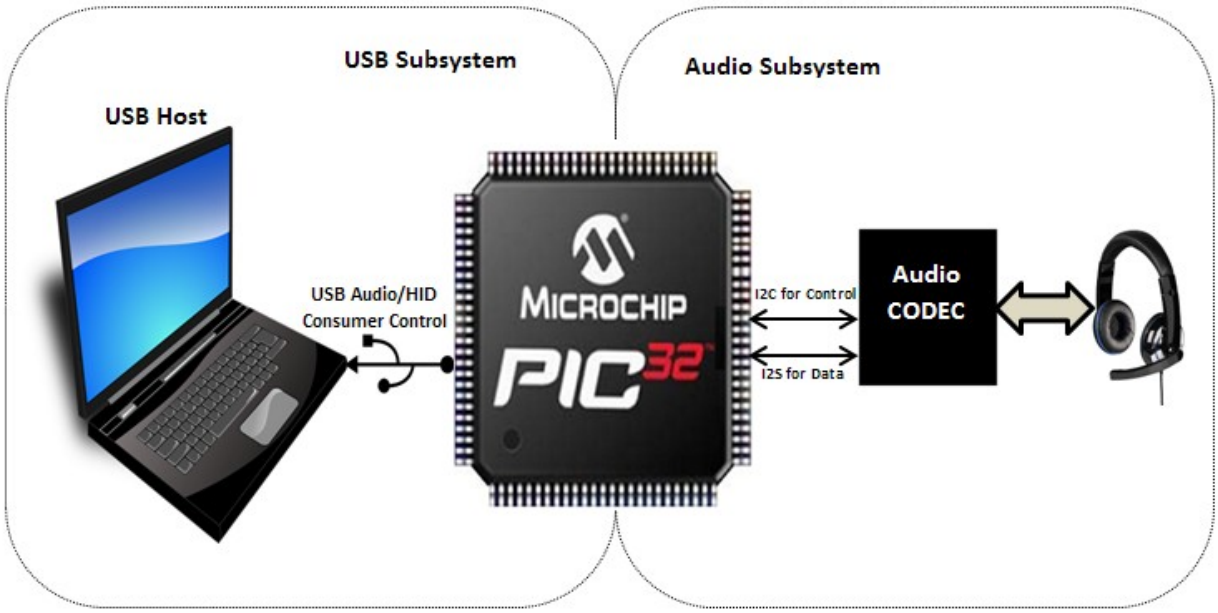
1. 功能模型

USB子系统包括主机和设备单元。设备单元通过USB电缆与主机相连。标准PC通常作为USB主机，而嵌入式器件（PIC32单片机）作为USB设备。USB主机运行USB主机软件库并安装了必需的USB音频设备驱动程序。USB设备运行USB设备软件协议栈并将自身标识为USB主机的USB音频设备。

在USB耳麦应用中，音频子系统由音频耳麦、音频编解码器和PIC32单片机组成。音频编解码器在模拟和数字信号域之间进行转换，允许PIC32单片机与耳麦之间以数字格式发送和接收信号。PIC32单片机通过串行通信与音频编解码器接口。这包括数据和控制接口。音频编解码器参数（例如音量、静音或均衡）通过控制接口进行访问。将音频控制（静音和音量等）从音频子系统转换到USB子系统的附加端口也属于USB音频系统的一部分。

下图显示了USB耳麦应用的功能模型。它包含USB子系统和音频子系统。

图1-1. USB耳麦应用的USB音频设备功能模型

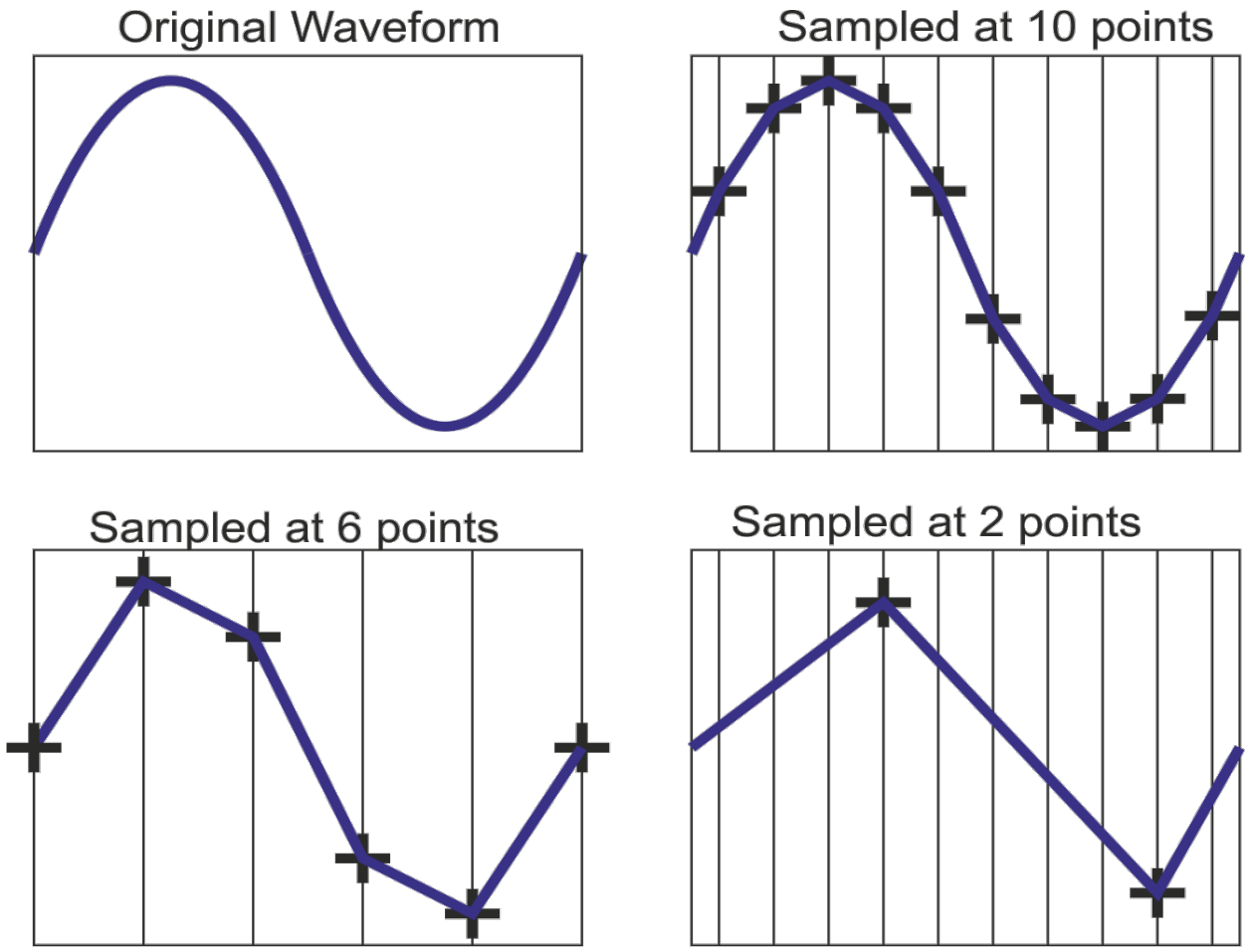


2. 音频概述

以数字形式表现声音是一种常见现象。通过转换器（模数转换器（Analog-to-Digital，ADC）/数模转换器（Digital-to-Analog，DAC））对音频信号进行数字化，并在数字媒体中进行表示和存储。下面介绍了一些与数字音频相关的特性和术语。

采样频率——采样和存储音频信号的次数（每秒）。以某一速率采样连续音频信号，并将这些采样值转换成数字形式。数字采样为离散值（数字），表示不同（采样）时间点上的音频信号的幅值。采样率的衡量单位为Hz或kHz。例如，44100 Hz采样率用于光盘（Compact Disc，CD）音频（见下图）。较高的采样频率能够提升系统的音频再现能力，但也会增加处理要求。

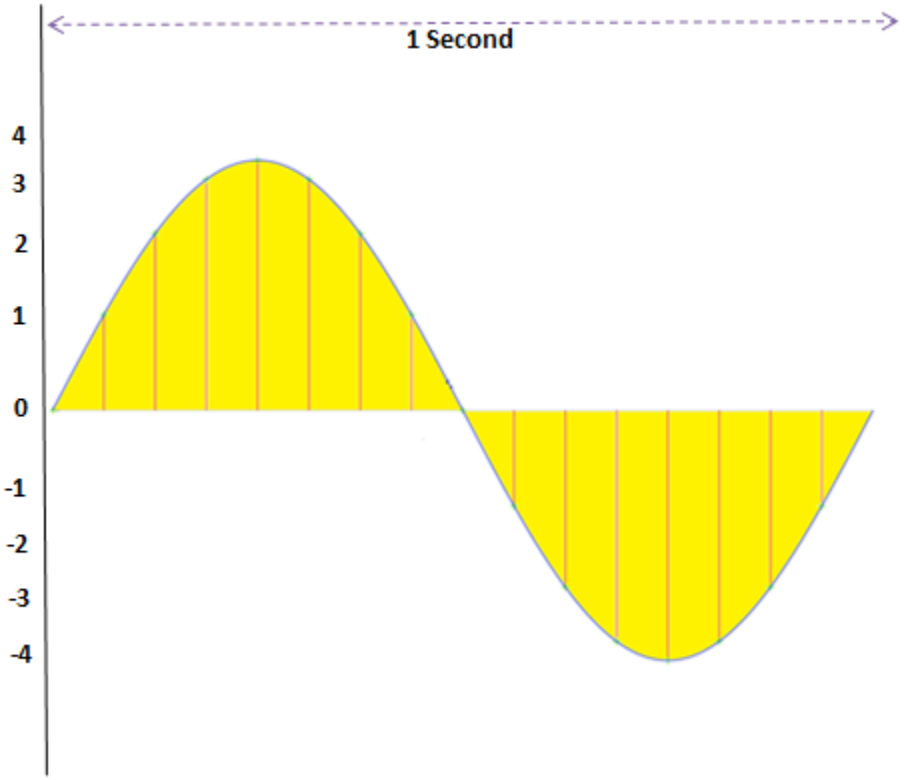
图2-1. 采样率



位深度——表示用于编码音频采样级别的位数。该参数定义了数字音频系统的动态范围。16位音频系统（位深度为16）的动态范围为2^{16}，这可获得65536个可能的级别。24位音频系统的动态范围为2^{24}，可提供超过1600万个级别。对于每一位，级别数都增加一倍。

在下图中，一秒钟有14次采样。最大采样级别为4，可以用4位表示（3位代表幅值，1位代表符号）。因此，音频信号的位深度是4。

图2-2. 位深度



比特率——回放时间内各单元处理的位数。未压缩数字音频文件的比特率为：采样率 x 位深度 x 通道数 = 比特率。

例如，CD音质的比特率：44100 x 16 x 2 = 1411200位/秒（1411 kbps或1.4 Mbps）。

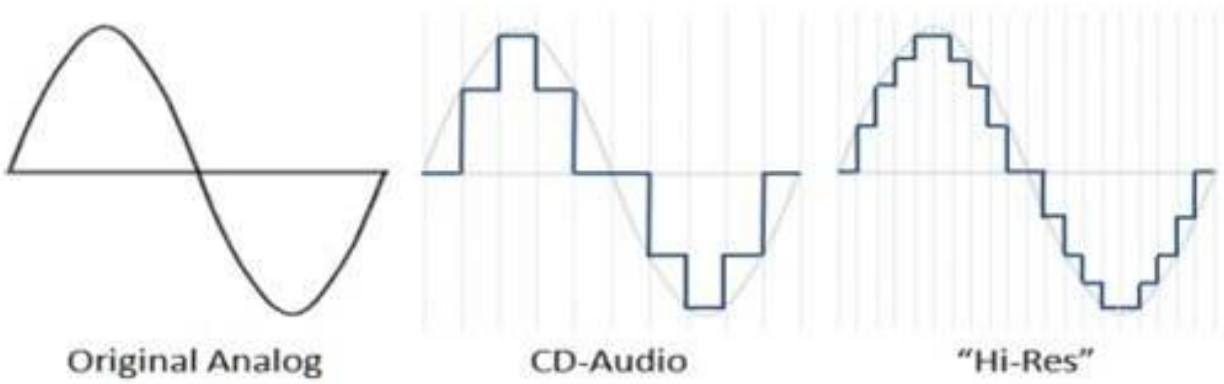
对于压缩数字音频文件，比特率是传输压缩数据以实现连续再现的最小速率。

例如，如果使用MP3算法“以128 kbps的比特率”对CD音频信号（16位、2通道、44100 Hz采样率和1411 kbps比特率）进行编码，则必须通过至少提供128 kbps比特率的链路进行连续传输。

分辨率——数字音频系统的分辨率基于三个因素：采样率、位深度和比特率。例如，CD音频信号具有44100 Hz采样率、16位深度和1411 kbps比特率的音频分辨率。

比特率大于CD数字音频信号的数字音频信号称为高分辨率音频信号。高分辨率或高保真音频信号以高于44100 Hz的频率和大于16位的位深度进行采样。典型的高分辨率音频信号以192 kHz频率进行采样并具有24位深度。下图显示了高分辨率信号的图形表示。

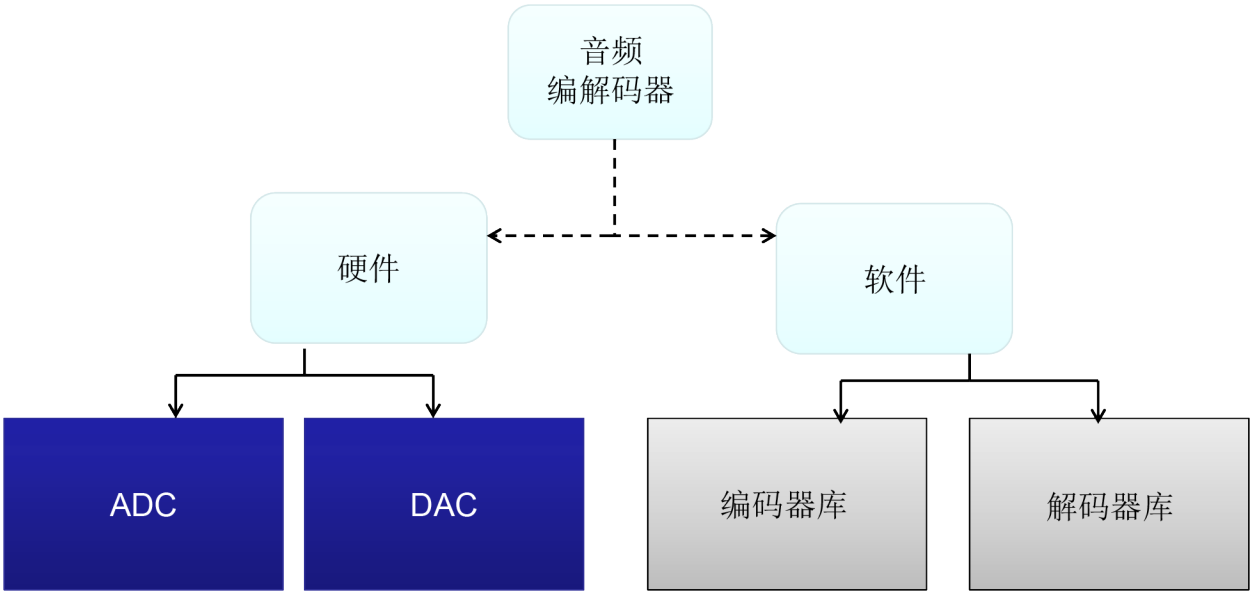
图2-3. 音频分辨率



音频文件格式——数字音频数据以未压缩或压缩形式存储在文件中。未压缩音频文件的大小往往大于压缩音频文件的大小。

音频编解码器——音频编解码器是具有编码或解码数字音频数据流功能的器件或程序。如下图所示，音频编解码器可分为软件和硬件两类。

图2-4. 音频编解码器



软件音频编解码器是一种计算机程序，实现了根据给定音频文件或流媒体音频编码格式来压缩和解压缩数字音频数据的算法。软件音频编解码器以库（例如MP3、SBC和AAC编解码器库）的形式实现。

硬件音频编解码器是指将模拟音频编码为数字信号（如ADC）和将数字数据解码回模拟信号（如DAC）的单个器件。支持音频输入和输出的声卡含有硬件音频编解码器。下图显示了PIC32音频编解码器AK4642EN子板的示例。

图2-5. 音频编解码器板

Audio Codec Daughter Board
(Part # AC320100)

典型的音频编解码器器件具有两个接口，一个控制接口和一个数据接口。控制接口是配置编解码器控制寄存器的媒介，通常使用I^2C协议。

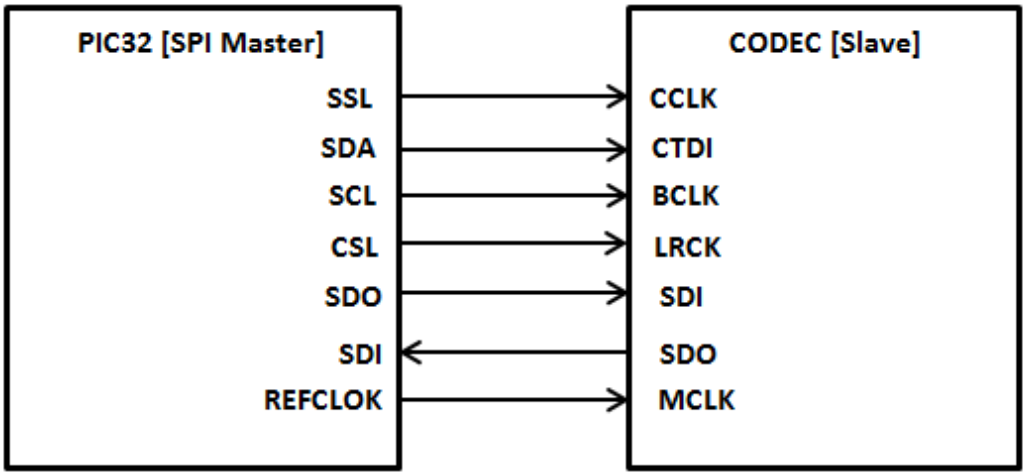
单片机与编解码器之间通过数据接口发送和接收数字音频信号。数据接口通常使用以下信号。

- 串行数据输出（Serial Data Output，SDO），用于向编解码器发送音频数据
- 串行数据输入（Serial Data Input，SDI），用于从编解码器接收音频数据
- 串行位时钟（Serial Bit Clock，SCK/BCLK）是必需的位时钟，由串行接口通信主器件提供
- 左/右时钟（Left/Right Clock，LRCK）是串行接口通信主器件为立体声数据提供的相位时钟

PIC32器件上的SPI模块支持用于数字音频通信的各种音频模式。这些模式支持各种接口格式、位分辨率和主/从配置。支持的数字音频模式包括I^2S格式、左对齐格式和右对齐格式。

PIC32器件上的SPI模块（在音频模式下）用作编解码器和PIC32单片机之间的数据接口。PIC32单片机用作SPI主器件时，PIC32器件的参考时钟（REFCLKO）输出是从编解码器器件的主时钟（MCLK），如下图所示。

图2-6. 音频编解码器接口



3. USB音频概述

3.1 USB操作

USB设备通过USB端口与USB主机系统相连。主机通过USB端点0的控制传输与设备通信，并检索与设备能力相关的信息。主机随后加载可以操作该设备的驱动程序。这种针对设备的检测、识别和加载驱动程序的过程称为*枚举*。

USB设备在枚举过程中使用描述符报告其属性和其他信息。USB描述符是具有指定格式的数据结构。每个描述符都以包含有描述符中总字节数的字段开头，后跟标识描述符类型的字段。

下面列出了枚举过程中USB主机将从设备请求的标准USB描述符：

- 设备描述符
- 配置描述符
- 接口描述符
- 端点描述符
- 字符串描述符

以下顺序步骤描述了典型的枚举过程。

1. **检测设备连接**
   USB接口由4根线组成：电源、地、数据加（D+）和数据减（D-）。当USB设备与USB主机相连时，USB数据线会发生变化。主机通过这种数据线上的变化来检测设备连接。数据线上的变化也会识别所连设备的速度。
2. **确定设备类型（设备描述符）**
   主机确定连接了USB设备并识别到通信速度后，会立即复位USB设备并尝试读取用于识别设备的信息。主机通过*设备描述符*获取该信息。主机加载基于设备描述符中包含的供应商ID（Vendor ID，VID）和产品ID（Product ID，PID）来管理设备的驱动程序。
3. **确定设备配置（配置描述符）**
   配置描述符提供设备特定的信息，例如设备支持的接口数和设备最大功耗。设备配置决定了USB设备在连接至USB时执行的功能。
4. **确定设备接口（接口描述符）**
   接口描述符提供关联的USB设备功能和执行该功能所需的端点数等信息。主机可以根据接口描述符中包含的类、子类和协议字段加载驱动程序。

与枚举相关的数据（例如设备描述符）使用控制传输进行传输。除控制传输外，USB还支持以下其他三种传输：批量传输、中断传输和同步传输。

- 批量传输通常用于处理大量非时间敏感型数据的传输。USB不会在调度批量传输时为批量传输保留带宽。批量传输可以在无其他待处理传输时进行调度。
- 中断传输不经常地或在具有有限延时的异步周期中发送和接收少量数据。USB音频设备可以使用中断传输向主机提供与音频控制相关的更新。
- 同步传输用于处理音频和视频数据等恒定速率的实时信息。该传输涉及连续数据流的流动。主机通过占用一部分可用USB带宽来确保满足数据的实时要求。分配给特定同步端点的带宽量由端点的接口描述符中包含的信息决定。在同步传输中，每一帧都传输新数据包。对于全速USB设备，每帧持续1毫秒。同步端点可配置为每帧发送1至1023个字节。同步传输不支持错误检测，也没有硬件控制的握手或错误校验机制。USB音频设备使用同步传输。

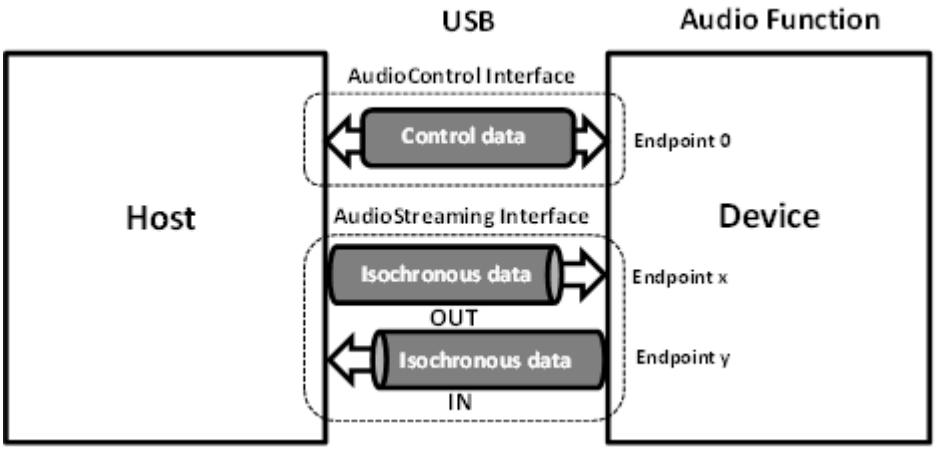
3.2 USB音频工作模型

USB音频设备实现了音频功能（耳机和麦克风等），并通过USB接口为主机提供对音频功能的访问。音频功能必须有一个音频控制（AudioControl，AC）接口，并且可以具有多个音频流（AudioStreaming，AS）接口，如下图所示。

AC接口用于控制音频功能的音频属性，例如音量控制和静音控制等。

AS接口是在主机与设备之间传送音频数据的传输媒介。

图3-1. USB音频工作模型



AC接口使用端点0进行控制数据通信。

AS接口使用专用的同步端点在主机和设备之间传输音频数据。

3.3 USB音频同步

USB音频设备包含以下时钟域：

- **采样时钟：**确定在USB主机和设备之间交换的音频样本的采样频率（速率）
- **USB总线时钟：**在全速USB设备上以1 kHz频率运行，通过总线上帧起始（Start-of-Frame，SOF）包的出现来指示
- **服务时钟：**客户端软件运行速率，用于处理执行操作之间可能累加的I/O请求包（I/O Request Packet，IRP）
- **编解码器时钟：**音频数据向/从编解码器模块传输的速率

为了可靠地传输同步数据，上述时钟必须同步。为避免因时钟漂移和抖动等原因产生不必要的多余声音，必须进行时钟同步。此类多余声音很容易被人耳听到并降低音质。

USB规范提供了三种同步类型来解决时钟不匹配问题，如下表所示。

表 3-1. USB音频同步类型

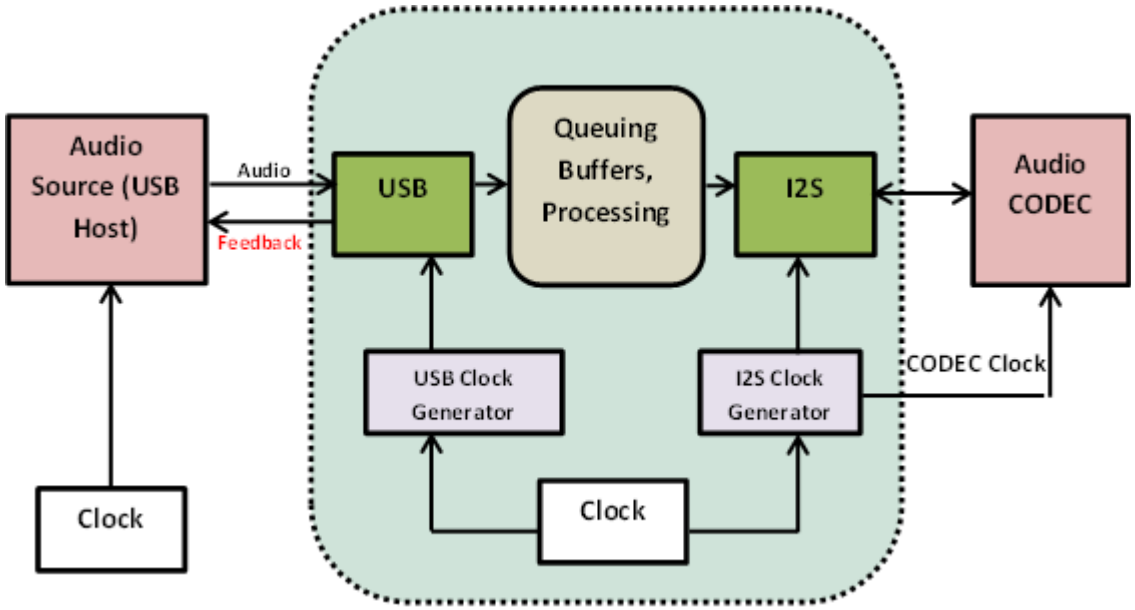
类型	发送方	接收方
异步	自由运行时钟。 向接收方提供隐式前馈	自由运行时钟。 向发送方提供显式反馈
同步	锁定到USB SOF的时钟 使用隐式反馈	锁定到USB SOF的时钟 使用隐式反馈
自适应	锁定到接收方的时钟 使用显式反馈	锁定到数据流的时钟 使用隐式反馈

3.3.1 异步

异步设备不通过公共时钟或USB SOF信号与主机同步。在异步设备中，自由运行的内部时钟决定了所需的数据速率。异步发送方端点将其数据速率信息隐含在每帧产生的采样数中。

异步接收方端点向发送方提供显式反馈信息，如下图所示。

图3-2. USB异步类型



在典型的异步接收方端点实现方案中，发送方（主机）和接收方（设备）通过协商使用同一工作音频采样率（例如48 kHz）。主机以每毫秒前都带有一个SOF的协定采样数（48）开始流数据。设备应用维持存储器缓冲区池来缓冲接收到的音频数据。它向编解码器转发数据以在音频设备上进行回放。设备管理从主机接收的数据的缓冲和处理。

由于主机和设备USB采用不同的时钟工作，因此可能出现时钟不匹配的情况，这可能导致主机发送的缓冲区超出设备可以管理的数量（缓冲区上溢）或发送的缓冲区较少而造成设备缺乏数据（缓冲区下溢）。

在异步端点实现方案中，设备通过维持缓冲区上溢（上限等级）和下溢（下限等级）情况的计数或水印等级来解决缓冲区上溢和下溢。设备检测到缓冲区计数超出水印等级时，将向主机发送报文以加速或减速。这种从设备发送到主机的报文称为*反馈报文*。反馈报文通过称为*反馈端点*的专用标准等时同步端点进行传输。来自设备的显式反馈包含加速量或减速量等信息。主机根据从设备接收的反馈值调整其数据速率来实现加速或减速。

除USB数据同步外，需要对编解码器的时钟进行选择以便其可以产生所需的音频采样频率。常用的音频采样频率是32 kHz、44.1 kHz、48 kHz、96 kHz和192 kHz。

例如，PIC32MX470F512L器件具有灵活的参考时钟输出模块。该模块可用于生成音频编解码器或DAC所需的小数时钟频率以适应各种采样率。

示例：

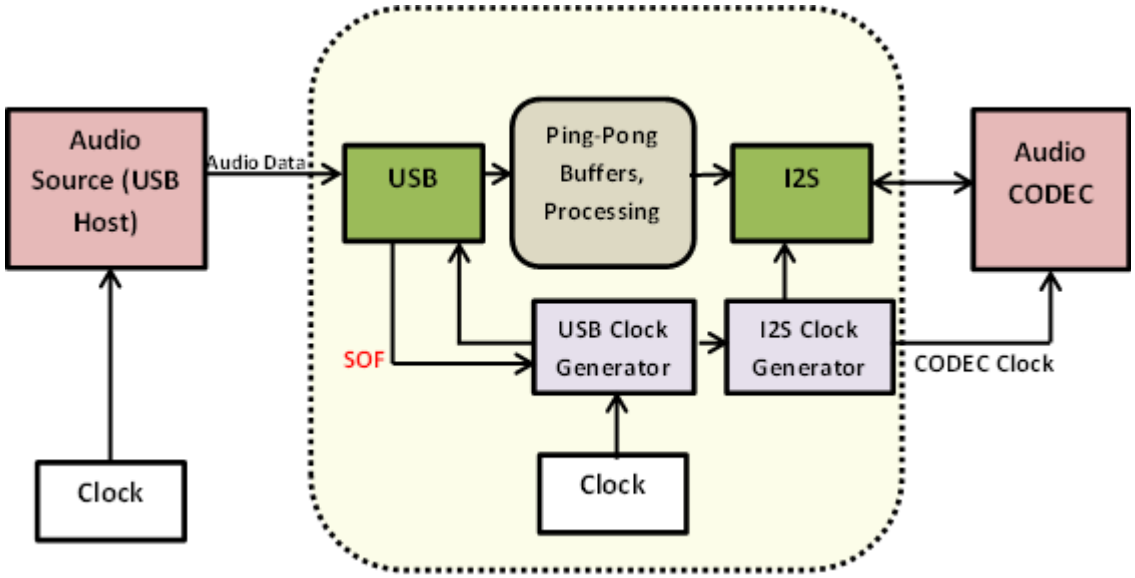
异步发送方——在基于PC的USB主机上运行的音频播放器，该主机根据内部时钟提供数据。

异步接收方——按其自身的内部采样时钟运行的耳机。

3.3.2 同步

对于同步类型，设备通过每个USB帧前的USB SOF信号与主机同步。SOF信号由USB主机生成，每间隔1毫秒出现一次。如下图所示，USB设备使用SOF信号校准产生设备采样时钟的内部振荡器。

图3-3. USB同步类型



在典型的同步接收方端点实现方案中，发送方（主机）和接收方（设备）通过协商使用同一工作音频采样率（即48 kHz）。USB主机每1毫秒发送一次后跟协定采样数（48）的SOF信号。SOF信号用作USB时钟发生器（通常是PLL）的输入，从而使USB设备时钟与USB同步。USB时钟通过生成I^2S时钟的PLL模块进一步用于生成编解码器的主时钟。

在该实现方案中，设备在乒乓缓冲区机制下使用两个缓冲区进行数据管理和处理。在乒乓缓冲区机制中，读取的数据存储在乒缓冲区中，而乓缓冲区中的数据将转发到编解码器中以进行回放。乒和乓缓冲区可以进行交换，从USB读取的数据可以连续转发到编解码器。

由于数据包的接收和发送都基于同一时钟，因此同步接收方端点实现方案极大地简化了缓冲区管理和处理。事实上，USB时钟和编解码器主时钟由同一个发送方生成，这有助于减轻常用音频采样频率的时钟不匹配问题。

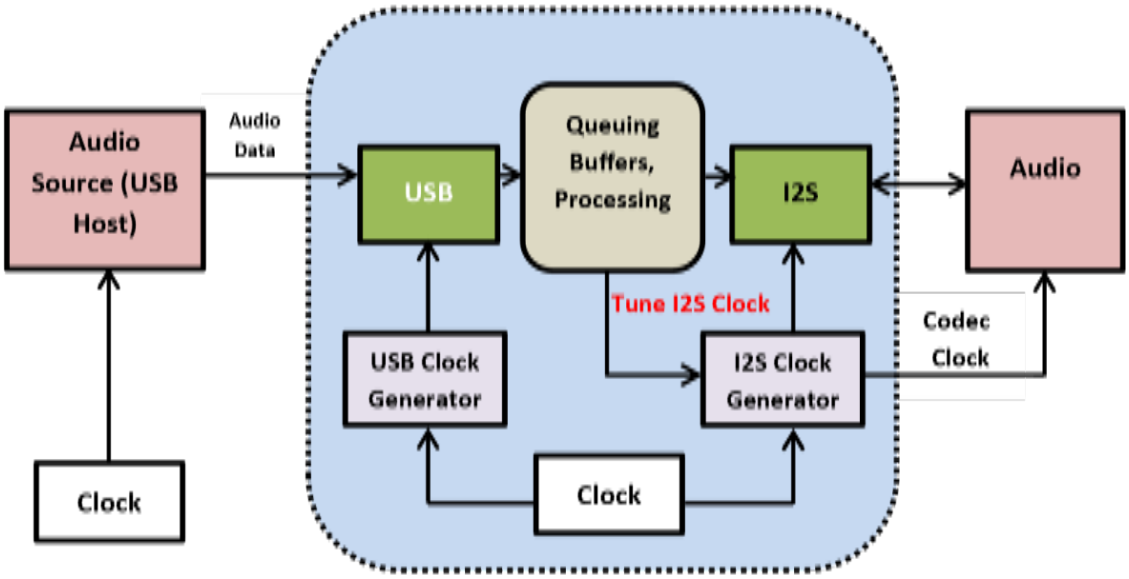
例如，PIC32MX470F512L器件上参考时钟输出PLL的输入可以来自USB PLL。生成的参考时钟作为编解码器的主时钟（MCLK）输入。

同步发送方的一个示例是麦克风，它从SOF合成其采样时钟，并且每一帧都会产生固定数量的音频采样。同样地，同步接收方通过USB SOF信号获得其采样时钟，而且每个USB帧都消耗固定数量的采样。

3.3.3 自适应

顾名思义，*自适应*表示发送方和接收方会根据彼此的状态调整其数据速率，如下图所示。

图3-4. USB自适应类型



自适应发送方端点生成数据的速率受数据接收方控制。接收方会向发送方提供反馈，使得发送方了解接收方所需的数据速率。

自适应接收方端点会将数据速率信息嵌入到数据流中。在平均时间窗口中接收到的样本平均数决定了瞬时数据速率。如果该数值在工作期间发生变化，则数据速率也相应进行调整。

在典型的自适应接收方端点实现方案中，发送方（主机）和接收方（设备）通过协商使用同一采样率（例如48 kHz）。USB主机每1毫秒发送一次后跟协定采样数（48）的SOF信号。

设备应用维持存储器缓冲区池以存储接收到的音频数据。它向编解码器转发数据以在音频设备上进行回放。应用也会维持在定义的周期内接收到的样本平均数。它会比较样本平均数与设置的下限或上限水印等级。当接收到的样本平均数超出水印范围时，应用通过略微提高或降低编解码器主时钟来调整时钟。调整编解码器主时钟可以防止缓冲区下溢或上溢情况，从而减少音频流中的多余声音。

例如，PIC32MX470F512L器件以USB PLL时钟为发送方提供可调参考时钟输出。参考时钟可以分步实时调整。调整范围通常在±0.2%之间，这样不会引入多余声音。

对于采样频率（fs）为48 kHz的数据流，参考时钟输出通常为256 fs，其中256是编解码器模块所需的采样频率乘数。

输出参考时钟频率 = 256 * 48 = 12288000 Hz。

该采样频率的摆幅为±0.2%，即要求参考时钟频率在12263424 Hz和12312576 Hz之间变化。

产生的采样频率范围是47.88 kHz至48.12 kHz。参考时钟的这种能力可防止缓冲区下溢或上溢情况，同时保持0.2%的可接受编解码器采样率摆幅范围并实现高质量音频。

自适应发送方的一个示例是含有完全自适应采样率转换器（Sample Rate Converter，SRC）的CD播放器，因此输出采样频率不再必须是44.1 kHz，而可以是SRC工作范围内的任意值。自适应接收方包括高端数字耳机和耳麦等设备。

3.4 音频功能拓扑

音频功能拓扑由表示音频功能的构件和音频信号流中各构件之间的互连组成。这些构件提供了操作音频功能属性（增益、音量和均衡）的方法。音频功能拓扑使用两种类型的构件：*单元*和*端子*。

在下面的讨论中，音频通道集群是共用相同特性（例如采样频率和位分辨率等）的逻辑音频通道的集合。

3.4.1 单元

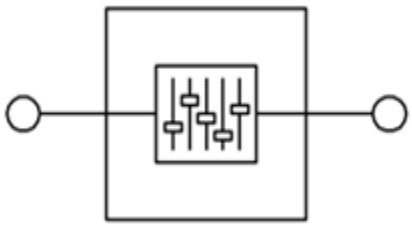
单元提供基本构件以充分说明大多数音频功能。音频功能通过连接多个单元来构建。一个单元有多个输入引脚和一个输出引脚，其中每个引脚都代表一个音频通道集群。根据所需拓扑连接单元的I/O引脚可以将单元连接在一起。单元的输入引脚从“1”开始编号，直到等于单元中输入引脚的总数。输出引脚编号始终为“1”。音频功能中的每个单元都由其相关的USB音频单元描述符（Unit Descriptor，UD）进行了充分说明。

USB音频规范v1.0介绍了五类单元。这五类单元分别是混合器单元、选择器单元、功能单元、处理单元和扩展单元。下面将对USB耳麦应用中使用的功能单元和混合器单元类型进行进一步说明。有关其他单元的详细说明，请参见USB音频规范v1.0。

功能单元：功能单元允许针对以下参数对输入的音频流进行操作：音量、静音、音调控制（低音、中音和高音）、图形均衡器、自动增益控制、延时、低音增强和响度。

功能单元描述符为功能单元中的各通道（包括*主*通道）报告存在的控制。功能单元支持多通道处理，允许单独操作每个逻辑通道。逻辑通道从“1”开始编号，直到等于通道总数。*主*通道（通道0）事实上始终存在。下图给出了功能单元图标。

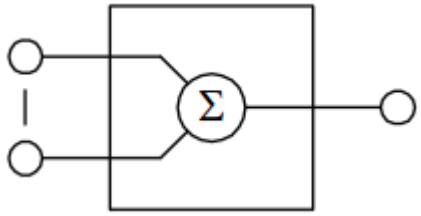
图3-5. 功能单元图标



混合器单元：混合器单元将多个逻辑输入通道转换成多个逻辑输出通道。输入通道分组到一个或多个音频通道集群。各集群通过输入引脚进入混合器单元。逻辑输出通道分组到一个音频通道集群并通过单个输出引脚离开混合器单元。

混合器单元描述符包含有关输入引脚数量以及输出引脚所含逻辑通道数的信息。下图给出了混合器单元图标。

图3-6. 混合器单元图标



3.4.2 端子

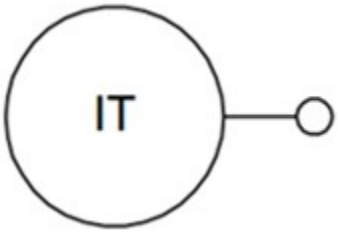
端子表示音频流的起点或终点。

USB端点是输入和输出端子的典型示例。它既可以向音频功能提供数据流（例如，基于PC的USB主机上的媒体播放器），也可以消耗来自音频功能的数据流（例如，连接到基于PC的USB主机的USB耳机）。

输入端子（Input Terminal，IT）：输入端子是代表音频功能内音频流起点的实体。它可用作流入音频功能的音频信息的插座。其功能是表示在将数据从原始音频流提取到嵌入在该流中的单独逻辑通道（解码过程）后的输入音频数据发送方。逻辑通道分组到一个音频通道集群并通过单个输出引脚离开输入端子。

通常，音频流通过USB OUT端点进入音频功能。端点与其相关的输入端子之间是一对一的关系。音频类特定端点描述符包含保持了对该输入端子的直接引用的字段。主机需要使用端点描述符和输入端子描述符来充分了解输入端子的特性和能力。与流相关的参数存储在端点描述符中。与控制相关的参数存储在端子描述符中。下图给出了输入端子图标。

图3-7. 输入端子图标

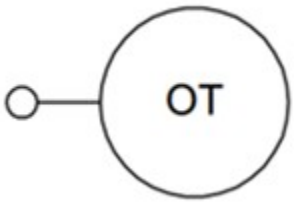


输出端子（Output Terminal，OT）：输出端子是代表音频功能内音频流终点的实体。它可用作流出音频功能的音频信息的插座。其功能是表示在将数据从单独的原始逻辑通道压缩到输出音频流（编码处理）之前的输出音频数据接收方。音频通道集群通过单个输入引脚进入输出端子。

通常，音频流通过USB输入端点退出音频功能。端点与其相关的输出端子之间是一对一的关系。

类特定端点描述符包含一个字段用于存放对该输出端子的直接引用。主机必须使用端点描述符和输出端子描述符来充分了解输出端子的特性和能力。与流相关的参数存储在音频类特定端点描述符中。下图给出了输出端子图标。

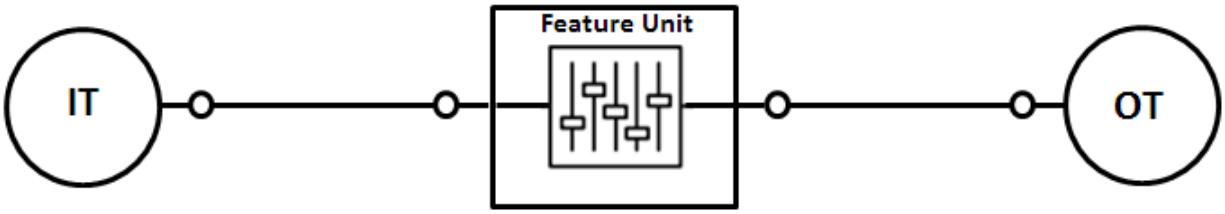
图3-8. 输出端子图标



示例：

下图显示了连接至基于PC的USB主机的USB耳机的基本拓扑。

图3-9. USB耳机拓扑



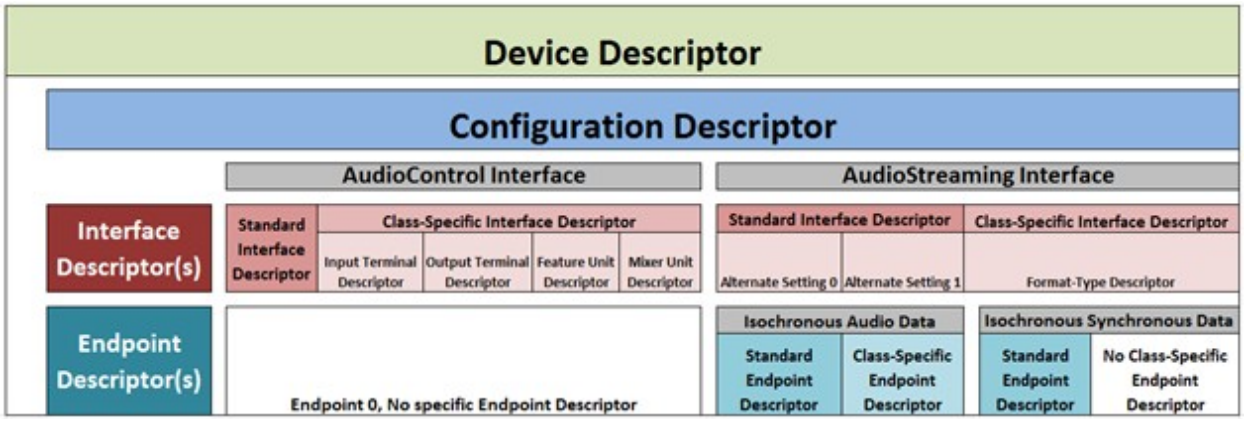
在上图中，输入端子（IT）表示音频流接口，该接口用于将音频从主机流入耳机设备。输入端子的输出引脚与功能单元的输入引脚相连。功能单元的输出引脚与代表物理耳机单元的输出端子（OT）的输入引脚相连。

3.5 USB音频描述符

与标准USB设备一样，USB音频设备也使用描述符来报告其属性。

对于音频设备，除标准描述符外，USB设备类为音频控制和流路径定义了类特定接口和端点描述符。下图显示了实现USB音频设备所需的描述符。

图3-10. USB音频描述符



注：针对本文档中讨论的USB耳麦应用，描述符部分对上图中提及的描述符进行了详细说明。

3.6 USB音频请求

3.6.1 标准请求

USB音频设备类支持USB规范中说明的标准请求。音频设备类对标准请求的值没有特定要求。

3.6.2 类特定请求

USB音频设备类支持附加的类特定请求以设置和获取与音频有关的控制。这些控制主要分为两组：

- 用于操作音量、音调和选择器位置等音频功能的控制
- 影响通过同步端点的数据传输（例如当前采样频率）的控制

音频控制请求：在音频控制请求中，通过控制嵌入在音频功能实体中的各控制的属性来执行对音频功能的控制。例如，功能单元。

音频流请求：在音频流请求中，通过控制接口控制或端点控制来执行对音频流接口的类特定行为的控制。

控制属性：对音频控制的典型设置/获取请求具有以下属性：

- 当前设置属性（SET_CUR）
- 当前设置属性（GET_CUR）
- 最低设置属性（SET_MIN）
- 最低设置属性（GET_MIN）
- 最高设置属性（SET_MAX）
- 最高设置属性（GET_MAX）
- 分辨率属性（SET_RES）
- 分辨率属性（GET_RES）

对实体（端子、单元和端点）的附加设置/获取请求具有以下属性：

- 存储空间属性（SET_MEM）
- 存储空间属性（GET_MEM）

3.7 USB音频1.0类功能

USB音频1.0类功能包括：

- 仅兼容USB全速模式，最大允许12 Mbps传输速率
- 最高允许24位、96 kHz音频
- 允许每毫秒传输一个音频数据帧：
  - 最大帧大小为1023字节。这支持96 kHz采样和24位宽立体声音频流。音频数据以576字节/毫秒的速率流动。
  - 较高的采样率（例如176 kHz）需要1056字节/毫秒，这超出了1023字节的最大帧大小。因此，该规范不支持24位176 kHz采样数字音频选项。

3.8 基于人机接口设备（HID）的音频控制

使用USB音频系统（即USB耳麦）时，音频控制按钮（静音、音量、下一曲目和上一曲目等）通常位于运行在主机PC上的音频播放器应用中。这些控制通过功能单元等构件与设备上的音频功能相关联并实施控制。在某些情况下，音频设备应用在其前面板上提供了附加的物理控制（例如音量旋钮、静音按钮或导航开关）。这些本地物理音频控制通过USB设备上的人机接口设备（Human Interface Device，HID）类接口表示。该HID类接口兼容音频类接口。

要在处理主物理音频控制的音频功能功能单元与本地物理音频控制之间建立联系，可以通过特殊的相关接口描述符对功能单元描述符进行补充。此相关接口描述符包含到HID类接口的链路。

HID类扩展了USB规范，为处理手动控制设备提供了一种标准方法。这包括常见的计算机设备，例如键盘、鼠标和游戏杆，以及电子设备控制器（例如VCR遥控器）和通用控制（例如旋钮和开关）。

HID类设备使用中断端点（Interrupt Endpoint，IN）与HID类驱动程序通信。

设备向主机传输（通过IN）异步数据，并从主机接收（通过OUT）低延时数据。

下表给出了USB HID使用名称、使用ID和使用页值，HID报告描述符必须使用这些使用页值来通告对常用物理音频控制的支持。有关下述和其他HID使用ID的详细信息，请参见www.usb.org/developers/hidpage上提供的*“HID使用表v1.12”*。

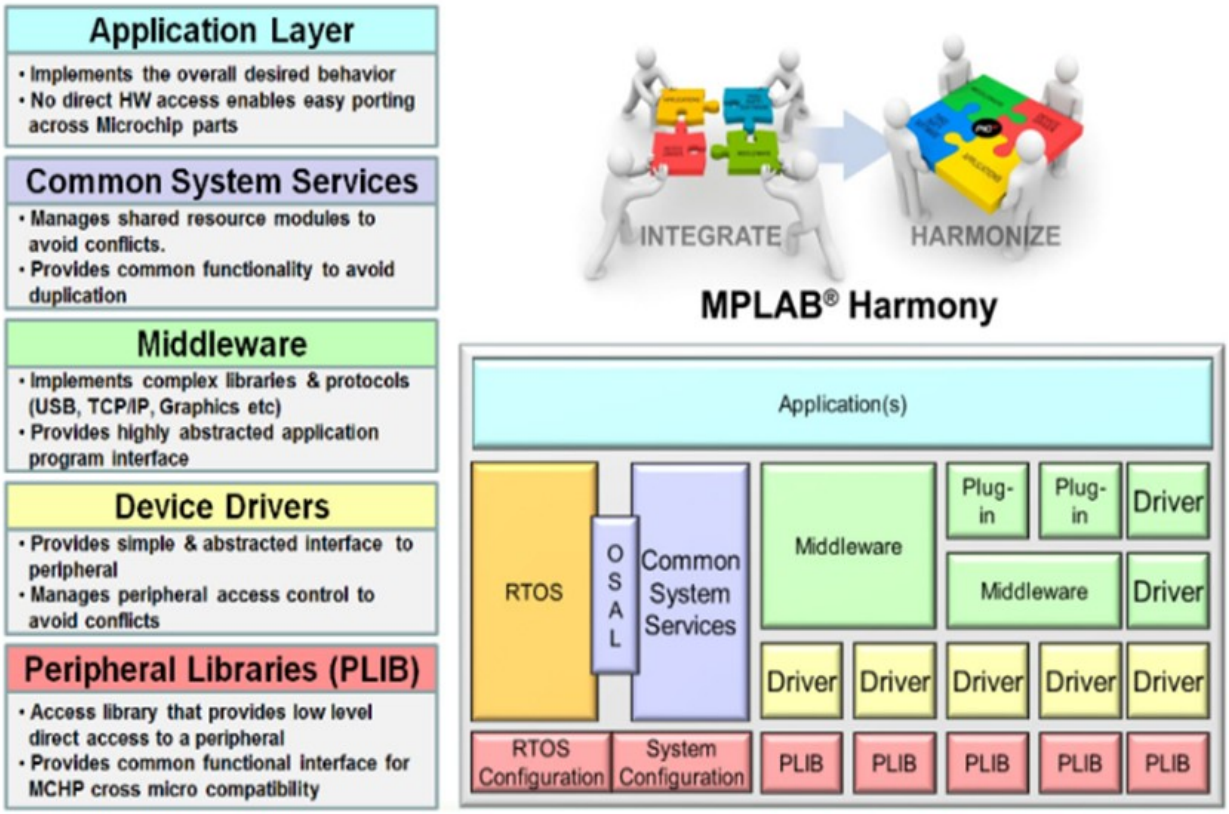
表3-2. 用于音频控制的HID使用ID和使用页

使用名称	使用页	使用ID
增大音量	消费电子（0x0C）	0xE9
减小音量	消费电子（0x0C）	0x0EA
电话静音	电话（0x0B）	0x2F
播放/暂停	消费电子（0x0C）	0xCD
扫描下一曲目	消费电子（0x0C）	0xB5
扫描上一曲目	消费电子（0x0C）	0xB6
停止	消费电子（0x0C）	0xB7
快进	消费电子（0x0C）	0xB3
快退	消费电子（0x0C）	0xB4

4. MPLAB Harmony概述

MPLAB Harmony是一款用于PIC32单片机的灵活、紧凑的全集成固件开发平台。该平台采用模块化和面向对象的设计要素，极大地增加了灵活性，既可搭配使用实时操作系统（Real-Time Operating System，RTOS），也可单独工作；同时，它还提供了易于使用的软件模块框架，这些软件模块可针对具体要求进行配置并且彼此之间可以完全协同工作。下图显示了MPLAB Harmony集成软件框架。

图4-1. MPLAB Harmony软件框架



MPLAB Harmony包含一组外设库、驱动程序、系统服务、中间件库和工具（MPLAB Harmony配置器（MHC）和MPLAB Harmony图形设计器（MPLAB Harmony Graphics Composer，MHGC）），根据设计，代码开发格式允许最大程度地重复使用和实现快速开发。

MHC是插入到MPLAB X IDE中的用户界面工具。MHC使MPLAB Harmony框架更易于使用。该工具负责处理基于MPLAB Harmony的项目的开销和整体结构，使您能够专注于为应用程序生成代码。基于MHC的UI选择。

- 添加项目所需的MPLAB Harmony源文件
- 生成一些MPLAB Harmony框架源文件（基于您的MHC选择）
- 生成入门MPLAB Harmony应用程序源文件：
  - 创建应用程序的入门状态机
  - 添加MPLAB Harmony项目所需的最少的函数
  - `#include`包含所需的所有源文件
- 生成入门MPLAB Harmony系统源文件：

  - 初始化PIC32（基于内核、时钟和外设的MHC选择）
  - 初始化MPLAB Harmony驱动程序（基于MHC选择）
  - 为中断服务程序创建桩函数
- 保存当前的MHC选择，以便导入到另一个项目

5. 使用MPLAB Harmony构建USB音频设备

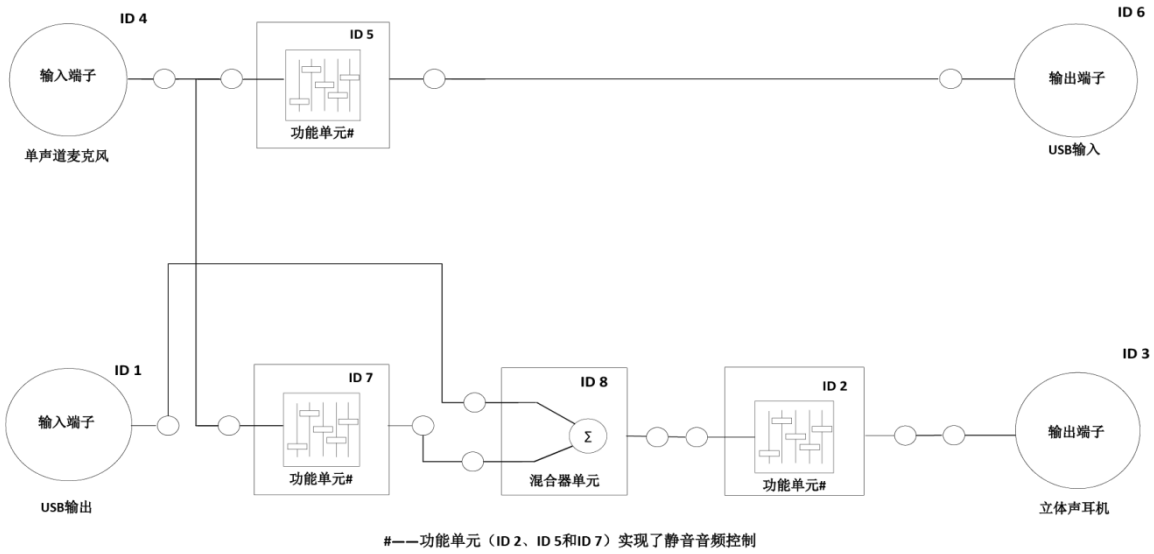
5.1 USB耳麦应用

USB耳麦设备是USB麦克风和USB耳机设备的结合。此外，USB耳麦还包含从麦克风输入到耳机输出的信号路径，这样便可通过耳麦的耳机听到麦克风采集的声音。以下几节介绍了USB耳麦应用的拓扑和描述符定义。

5.1.1 拓扑

下图给出了本应用笔记中讨论的USB耳麦应用的拓扑。

图5-1. USB耳麦应用拓扑



#——功能单元（ID 2、ID 5和ID 7）实现了静音音频控制

输入端子（ID 4）代表单声道麦克风。输入端子（ID 4）的输出引脚与功能单元（ID 5）的输入引脚相连。功能单元（ID 5）的输出引脚与输出端子（ID 6）的输入引脚相连，该输出端子代表用于将麦克风数据传输到主机的音频流接口。

输入端子（ID 1）代表从主机传输到耳麦设备的耳机数据。输入端子（ID 1）的输出引脚与混合器单元（ID 8）的输入引脚1相连。混合器单元（ID 8）的输出引脚与功能单元（ID 2）的输入引脚相连。功能单元（ID 2）的输出引脚与代表物理耳机的输出端子（ID 3）的输入引脚相连。

输入端子（ID 4）还与功能单元（ID 7）的输入引脚相连。功能单元（ID 7）的输出引脚与混合器单元（ID 8）的输入引脚2相连。该连接构成了麦克风与耳机之间的侧音混合路径。

本应用笔记中讨论的USB耳麦应用使用各描述符的*bTerminalID*和*bUnitID*字段，如上图所示。

5.1.2 描述符

设备描述符

下表提供了USB耳麦应用中设备描述符的详细信息。

bDeviceClass、*bDeviceSubClass*和*bDeviceProtocol*字段均设置为0x00，这些字段将在接口描述符中定义。

表5-1. 设备描述符

偏移量	字段	大小	值	说明
0	*bLength*	1	0x12	设备描述符的大小（字节）
1	*bDescriptorType*	1	0x01	设备描述符
2	*bcdUSB*	2	0x0200	支持USB 2.0规范
4	*bDeviceClass*	1	0x00	类在接口描述符中指定
5	*bDeviceSubClass*	1	0x00	子类在接口描述符中指定
6	*bDeviceProtocol*	1	0x00	协议在接口描述符中指定
7	*bMaxPacketSize0*	1	0x40	端口0的最大数据包大小为64字节
8	*idVendor*	2	0x04D8	供应商ID（例如，Microchip供应商ID = 04D8h）
10	*idProduct*	2	0x00FF	产品ID
12	*bcdDevice*	2	0x0100	设备版本号（例如01.00）
14	*iManufacturer*	1	0x01	字符串描述符中制造商字符串的索引
15	*iProduct*	1	0x02	字符串描述符中产品字符串的索引
16	*iSerialNumber*	1	0x00	字符串描述符中设备序列号的索引
17	*bNumConfigurations*	1	0x01	配置数设置为1

配置描述符

下表提供了USB耳麦应用中配置描述符的详细信息。

当设备收到获取配置描述符请求时，它会将配置描述符、接口描述符和端点描述符返回给USB主机。枚举期间，USB主机将向设备发出设置配置请求，从而要求设备将此配置设置为活动状态。

表5-2. 配置描述符

偏移量	字段	大小	值	说明
0	*bLength*	1	0x09	设备描述符的大小（字节）
1	*bDescriptorType*	1	0x02	配置描述符
2	*wTotalLength*	2	0x00EA	为此配置返回的数据的总长度为232字节
4	*bNumInterfaces*	1	0x03	此配置中的接口数为0x03

偏移量	字段	大小	值	说明
5	*bConfigurationValue*	1	0x01	用作选择此配置的SetConfiguration()函数请求的参数值
6	*iConfiguration*	1	0x00	描述此配置的字符串描述符的索引
7	*bmAttributes*	1	0xC0	自供电
8	*MaxPower*	2	0x32	此设备最多将消耗100 mA（2x MaxPower）

bNumInterfaces的值设置为3，表示1个控制接口和2个音频流接口（耳机和麦克风各一个）。

*wTotalLength*表示此配置的总数据长度；它包括下表中提及的标准和类特定接口以及端点描述符的长度。

表5-3. 全部描述符长度

描述符名称	长度
配置描述符	0x09
标准音频控制接口描述符	0x09
类特定音频控制接口描述符	0x0A
耳机——输入端子描述符	0x0C
麦克风——输入端子描述符	0x0C
功能单元描述符（ID 2）	0x0D
麦克风——功能单元描述符	0x0B
麦克风（侧音）——功能单元描述符	0x0B
混合器单元描述符	0x0D
耳机——输出端子描述符	0x09
麦克风——输出端子描述符	0x09
耳机——零带宽标准AS接口描述符（备用设置0）	0x09
耳机——工作标准AS接口描述符（备用设置1）	0x09
耳机——类特定的AS通用接口描述符	0x07
耳机——类型1格式类型描述符	0x11
耳机——标准AS音频数据端点描述符	0x09
耳机——类特定的AS音频数据端点描述符	0x07
麦克风——零带宽标准AS接口描述符（备用设置0）	0x09
麦克风——工作标准AS接口描述符（备用设置1）	0x09
麦克风——类特定的AS通用接口描述符	0x07
麦克风——类型1格式类型描述符	0x11
麦克风——标准AS音频数据端点描述符	0x09
麦克风——类特定的AS音频数据端点描述符	0x07

标准音频控制接口描述符

下表提供了USB耳麦应用中标准音频控制接口描述符的详细信息。

- *bNumEndpoints*设置为0x00，表示除控制端点0外无其他端点
- *bNumInterfaces*的值设置为3，表示1个控制接口和2个音频流接口（耳机和麦克风各一个）
- *bInterfaceClass*设置为0x01，对应于音频类
- *bInterfaceSubClass*设置为0x01，对应于音频控制子类
- *bInterfaceProtocol*设置为0x000，对应于音频接口协议代码的未定义协议。这些值由USB音频规范1.0指定，指示主机关联音频客户端驱动程序

表5-4. 标准音频控制接口描述符

偏移量	字段	大小	值	说明
0	*bLength*	1	0x09	此描述符的大小（字节）
1	*bDescriptorType*	1	0x04	接口描述符
2	*bInterfaceNumber*	1	0x00	此接口的编号
3	*bAlternateSetting*	1	0x00	用于标识此接口备用设置的值
4	*bNumEndpoints*	1	0x00	此接口使用的端点数量（除端点0外）
5	*bInterfaceClass*	1	0x01	音频——音频接口类代码
6	*bInterfaceSubClass*	1	0x01	音频控制——音频接口子类代码
7	*bInterfaceProtocol*	1	0x00	不使用，设置为0
8	*iInterface*	1	0x00	此接口无字符串描述符

类特定音频控制接口描述符

下表提供了USB耳麦应用中类特定音频控制接口描述符的详细信息。

表5-5. 类特定音频控制接口描述符

偏移量	字段	大小	值	说明
0	*bLength*	1	0x0A	此描述符的大小（字节）
1	*bDescriptorType*	1	0x24	类特定的接口描述符
2	*bDescriptorSubtype*	1	0x01	报头描述符子类型

偏移量	字段	大小	值	说明
3	*bcdADC*	2	0x0001	音频设备类规范版本号（二-十进制码）
5	*wTotalLength*	2	0x0064	为此描述符返回的字节的总数
7	*bInCollection*	1	0x02	此音频接口集合中音频流接口的数量
8	*baInterfaceNr(1)*	1	0x01	集合中第一个音频流接口的接口编号
9	*baInterfaceNr(2)*	1	0x02	集合中第二个音频流接口的接口编号

*wTotalLength*设置为0x0064。这表示此描述符报头和上表中提及的其所有单元与端子描述符的总长度。

表5-6. 类特定描述符长度

描述符名称	长度
类特定音频控制接口描述符	0x0A
耳机——输入端子描述符	0x0C
麦克风——输入端子描述符	0x0C
功能单元描述符（ID 2）	0x0D
麦克风——功能单元描述符	0x0B
麦克风（侧音）——功能单元描述符	0x0B
混合器单元描述符	0x0D
耳机——输出端子描述符	0x09
麦克风——输出端子描述符	0x09

*bInCollection*设置为0x02。这表示一个流接口用于耳机功能，一个流接口用于麦克风功能。

*baInterfaceNr(1)*设置为0x01，表示耳机功能的接口编号。

*baInterfaceNr(2)*设置为0x01，表示麦克风功能的接口编号。

耳机——输入端子描述符

下表提供了USB耳麦应用中对应于耳机功能的输入端子描述符的详细信息。

*bTerminalID*设置为0x01（见图5-1中的拓扑）。本应用中的耳机功能输入端子由此ID引用。

*wTerminalType*设置为0x0101。端子类型设置为USB流，因为该端子是从PC流输出的USB数据的输入点。

*bAssocTerminal*设置为0x00，因为此输入端子未与输出端子直接相连。

*bNrChannels*设置为0x02，因为端子的输出音频通道集群具有两个立体声数据输出逻辑通道（即左通道和右通道）。

*wChannelConfig*设置为0x0003，对应于表示端子输出音频通道集群空间位置的耳机通道字段的左前（L）和右前（R）位置。

表5-7. 耳机——输入端子描述符

偏移量	字段	大小	值	说明
0	*bLength*	1	0x0C	此描述符的大小（字节）
1	*bDescriptorType*	1	0x24	类特定的接口描述符
2	*bDescriptorSubtype*	1	0x02	INPUT_TERMINAL描述符子类型
3	*bTerminalID*	1	0x01	此端子的惟一ID设置为1
4	*wTerminalType*	2	0x0101	端子类型为USB流
6	*bAssocTerminal*	1	0x00	与此输入端子相关联的输出端子
7	*bNrChannels*	1	0x02	端子的输出音频通道集群中逻辑输出通道的数量
8	*wChannelConfig*	2	0x0003	逻辑通道的位图位置
10	*iChannelNames*	1	0x00	无非预定义逻辑通道，该索引设置为0
11	*iTerminal*	1	0x00	此接口无字符串描述符

注：下表给出了音频集群中的空间位置（位域是***wChannelConfig***字段的一部分）。

D0	左前（L）
D1	右前（R）
D2	前中（C）
D3	低频增强（LFE）
D4	左环绕（LS）
D5	右环绕（RS）
D6	中偏左（LC）
D7	中偏右（RC）
D8	环绕（S）
D9	左侧（SL）
D10	右侧（SR）
D11	顶部（T）
D15-D12	保留

麦克风——输入端子描述符

下表提供了USB耳麦应用中麦克风功能输入端子描述符的详细信息。

bTerminalID为0x04（见图5-1中的拓扑）。本应用中的麦克风功能输入端子由此ID引用。

wTerminalType设置为0x0201。端子类型设置为麦克风，因为该端子是麦克风数据的输入点。

bAssocTerminal为0x0，因为此输入端子未与输出端子直接相连。

bNrChannels为0x01，因为端子的输出音频通道集群具有一条单声道数据输出逻辑通道。

wChannelConfig为0x0004，对应于表示端子输出音频通道集群的空间位置的前中（C）字段。

表5-8. 麦克风——输入端子描述符

偏移量	字段	大小	值	说明
0	*bLength*	1	0x0C	此描述符的大小（字节）
1	*bDescriptorType*	1	0x24	类特定的接口描述符
2	*bDescriptorSubtype*	1	0x02	INPUT_TERMINAL描述符子类型
3	*bTerminalID*	1	0x04	此端子的惟一ID设置为4
4	*wTerminalType*	2	0x0201	端子类型为麦克风设备
6	*bAssocTerminal*	1	0x00	没有输出端子与此输入端子相关联
7	*bNrChannels*	1	0x01	端子的输出音频通道集群中逻辑输出通道的数量
8	*wChannelConfig*	2	0x0004	逻辑通道的位图位置
10	*iChannelNames*	1	0x00	无非预定义逻辑通道，该索引设置为0
11	*iTerminal*	1	0x00	此输入端子无字符串描述符

混合器——功能单元描述符

下表提供了USB耳麦应用中输入为混合器单元的功能单元描述符的详细信息。

*bUnitID*为0x02（见图5-1中的拓扑）。本应用中的功能单元（具有混合器输入端子）由此ID引用。

*bSourceID*设置为0x08。它是与此功能单元相连的混合器单元的ID。

*bControlSize*为0x02，表示功能单元针对每个通道（含主通道）最多支持16位（2字节——16位）控制。

*bmaControls(0)*设置为0x001，表示通道0（主通道）中使能了静音音频控制。

*bmaControls(1)*为0x000，表示通道1（左通道）中未使能音频控制。

*bmaControls(2)*为0x000，表示通道2（右通道）中未使能音频控制。

表5-9. 混合器——功能单元描述符（ID 2）

偏移量	字段	大小	值	说明
0	*bLength*	1	0x0B	此描述符的大小（字节）
1	*bDescriptorType*	1	0x24	类特定的接口描述符
2	*bDescriptorSubtype*	1	0x06	FEATURE_UNIT描述符子类型
3	*bUnitID*	1	0x02	此功能单元的惟一ID设置为2
4	*bSourceID*	1	0x08	此功能单元与输入端子（ID 8）相连
5	*bControlSize*	1	0x02	控制通道位图数组中的各元素都为2字节
6	*bmaControls(0)*	2	0x0001	通道0（主通道）中使能了静音音频控制
8	*bmaControls(1)*	2	0x0000	通道1中未使能音频控制
10	*bmaControls(2)*	2	0x0000	通道2中未使能音频控制
12	*iFeature*	1	0x00	此功能单元无字符串描述符

注：下表给出了音频控制通道字段*bmaControls(n)*的位图的位域。

表5-10. 音频控制通道位

位域	说明
D0	静音
D1	音量
D2	低音
D3	中音
D4	高音
D5	图形均衡器
D6	自动增益
D7	延时
D8	低音增强
D9	响度

麦克风——功能单元描述符

下表提供了音频耳麦应用中输入为麦克风的功能单元描述符的详细信息。

*bUnitID*为0x05（见图5-1中的拓扑）。本应用中的功能单元（具有麦克风输入端子）由此ID引用。

*bSourceID*设置为0x04。它是与此功能单元相连的麦克风输入端子的ID。

*bControlSize*为0x02，表示功能单元针对每个通道（含主通道）最多支持16位（2字节——16位）控制。

*bmaControls(0)*设置为0x001，表示通道0（主通道）中使能了静音音频控制。

*bmaControls(1)*为0x000，表示通道1中未使能音频控制。

表5-11. 麦克风——功能单元描述符

偏移量	字段	大小	值	说明
0	*bLength*	1	0x0B	此描述符的大小（字节）
1	*bDescriptorType*	1	0x24	类特定的接口描述符
2	*bDescriptorSubtype*	1	0x06	FEATURE_UNIT描述符子类型
3	*bUnitID*	1	0x05	此功能单元的惟一ID设置为5
4	*bSourceID*	1	0x04	此功能单元与输入端子（ID 4）相连
5	*bControlSize*	1	0x02	控制通道位图数组中的各元素都为2字节
6	*bmaControls(0)*	2	0x0001	主通道中使能了静音音频控制
8	*bmaControls(1)*	2	0x0000	通道1中未使能音频控制
10	*iFeature*	1	0x00	此功能单元无字符串描述符

麦克风（侧音）——功能单元描述符

下表提供了USB耳麦应用中输入为麦克风输入端子（侧音）的功能单元描述符的详细信息。

*bUnitID*为0x07（见图5-1中的拓扑）。本应用中的功能单元（具有用于侧音混合的麦克风输入端子）由此ID引用。

*bSourceID*设置为0x04。它是与此功能单元相连的麦克风输入端子（用于侧音混合）的ID。

*bControlSize*为0x02，表示功能单元针对每个通道（含主通道）最多支持16位（2字节——16位）控制。

*bmaControls(0)*设置为0x001，表示通道0（主通道）中使能了静音音频控制。

*bmaControls(1)*为0x000，表示通道1中未使能音频控制。

表5-12. 麦克风（侧音）——功能单元描述符

偏移量	字段	大小	值	说明
0	*bLength*	1	0x0B	此描述符的大小（字节）
1	*bDescriptorType*	1	0x24	类特定的接口描述符
2	*bDescriptorSubtype*	1	0x06	FEATURE_UNIT描述符子类型
3	*bUnitID*	1	0x07	此功能单元的惟一ID设置为7
4	*bSourceID*	1	0x04	此功能单元与输入端子（ID 4）相连
5	*bControlSize*	1	0x02	控制通道位图数组中的各元素都为2字节
6	*bmaControls(0)*	2	0x0001	主通道中使能了静音音频控制
8	*bmaControls(1)*	2	0x0000	通道1中未使能音频控制
10	*iFeature*	1	0x00	此功能单元无字符串描述符

混合器单元描述符

下表提供了USB耳麦应用中混合器单元描述符的详细信息。

*bUnitID*为0x08（见图5-1中的拓扑）。本应用中的混合器单元由此ID引用。

*bNrInPins*为0x02，表示此混合器单元具有两个输入引脚。

*bSourceID(1)*为0x01，表示具有ID 1的输入端子（对应于耳机功能）与此混合器单元的引脚1相连。

*bSourceID(2)*为0x07，表示具有ID 7的功能端子（对应于带侧音的麦克风）与此混合器单元的引脚2相连。

*bNrChannels*为0x02，因为端子的输出音频通道集群具有两个立体声数据输出逻辑通道（即左通道和右通道）。

*wChannelConfig*为0x0003，对应于表示端子输出音频通道集群的空间位置的左前（L）和右前（R）字段。

*bmControls*为0x00，因为[(n x m) MOD 8]是0x0。其中“n”为输入通道的数量（2），“m”为输出通道的数量（2）。

表5-13. 混合器单元描述符

偏移量	字段	大小	值	说明
0	*bLength*	1	0x0D	此描述符的大小（字节）
1	*bDescriptorType*	1	0x24	类特定的接口描述符
2	*bDescriptorSubtype*	1	0x04	MIXER_UNIT描述符子类型
3	*bUnitID*	1	0x08	此混合器单元的惟一ID设置为8
4	*bNrInPins*	1	0x02	此混合器单元的两个输入引脚
5	*baSourceID(1)*	1	0x01	ID 1与此单元的引脚1相连
6	*baSourceID(2)*	1	0x07	ID 7与此单元的引脚2相连
7	*bNrChannels*	1	0x02	端子的输出音频通道集群中逻辑输出通道的数量
8	*wChannelConfig*	2	0x0003	逻辑通道的位图位置
10	*iChannelNames*	1	0x00	无非预定义逻辑通道，该索引设置为0
11	*bmControls*	1	0x00	无可编程的混合控制
12	*iMixer*	1	0x00	此混合器单元无字符串描述符

耳机——输出端子描述符

下表提供了USB耳麦应用中对应于耳机功能的输出端子描述符的详细信息。

*bTerminalID*为0x03（见图5-1中的拓扑）。本应用中的耳机功能输出端子由此ID引用。

*wTerminalType*设置为0x0302。端子类型设置为耳机，因为该端子是从PC流输出的USB数据的输出点。

*bAssocTerminal*为0x00，因为此输出端子没有与输入端子直接相连。

*bSource*设置为0x02，表示用作此输出端子源的混合器单元的ID。

表5-14. 耳机——输出端子描述符

偏移量	字段	大小	值	说明
0	*bLength*	1	0x09	此描述符的大小（字节）
1	*bDescriptorType*	1	0x24	类特定的接口描述符
2	*bDescriptorSubtype*	1	0x03	OUTPUT_TERMINAL描述符子类型
3	*bTerminalID*	1	0x03	此端子的惟一ID设置为3
4	*wTerminalType*	2	0x0302	端子类型为耳机
6	*bAssocTerminal*	1	0x00	没有输入端子与此输出端子相关联
7	*bSource*	1	0x02	与此输出端子相连的混合器单元的ID
8	*iTerminal*	1	0x00	此输出端子无字符串描述符

麦克风——输出端子描述符

下表提供了USB耳麦应用中对应于耳机功能的输出端子描述符的详细信息。

*bTerminalID*为0x06（见图5-1中的拓扑）。本应用中的麦克风功能输出端子由此ID引用。

*wTerminalType*设置为0x0101。端子类型设置为耳机，因为该端子是从PC输出的USB数据流的输出点。

*wTerminalType*设置为0x0101。端子类型设置为USB流，因为该端子是输入到PC的USB数据流的输出点。

*bAssocTerminal*为0x0，因为此输出端子没有与输入端子直接相连。

*bSource*设置为0x05，表示用作此输出端子源的功能单元的ID。

表5-15. 麦克风——输出端子描述符

偏移量	字段	大小	值	说明
0	*bLength*	1	0x09	此描述符的大小（字节）
1	*bDescriptorType*	1	0x24	类特定的接口描述符

偏移量	字段	大小	值	说明
2	*bDescriptorSubtype*	1	0x03	OUTPUT_TERMINAL描述符子类型
3	*bTerminalID*	1	0x06	此端子的惟一ID设置为3
4	*wTerminalType*	2	0x0101	端子类型为USB流
6	*bAssocTerminal*	1	0x00	没有输入端子与此输出端子相关联
7	*bSource*	1	0x05	与此输出端子相连的功能单元的ID
8	*iTerminal*	1	0x00	此输出端子无字符串描述符

耳机——零带宽标准AS接口描述符（备用设置0）

下表提供了USB耳麦应用中耳机功能的零带宽标准AS接口描述符（备用设置0）的详细信息。

*bInterfaceNumber*设置为0x01。用于在此配置下的并发接口阵列中标识此接口的值。

*bAlternateSetting*为0x0，表示零带宽设置，用于在不使用音频设备时放弃总线上声明的带宽。

*bNumEndpoints*为0x0，由于它是零带宽接口，因此没有相关端点。

表5-16. 耳机——零带宽标准AS接口描述符（备用设置0）

偏移量	字段	大小	值	说明
0	*bLength*	1	0x09	此描述符的大小（字节）
1	*bDescriptorType*	1	0x04	接口描述符
2	*bInterfaceNumber*	1	0x01	从0开始的值，标识此配置支持的并发接口阵列中的索引
3	*bAlternateSetting*	1	0x00	此接口的备用设置数量为0
4	*bNumEndpoints*	1	0x00	零带宽接口没有相关端点
5	*bInterfaceClass*	1	0x01	音频接口类
6	*bInterfaceSubClass*	1	0x02	音频流接口子类
7	*bInterfaceProtocol*	1	0x00	未定义协议
8	*iInterface*	1	0x00	此接口无字符串描述符

耳机——工作标准AS接口描述符（备用设置1）

下表提供了USB耳麦应用中耳机功能的工作带宽标准AS接口描述符（备用设置1）的详细信息。

*bInterfaceNumber*设置为0x01。用于在此配置下的并发接口阵列中标识此接口的值。

*bAlternateSetting*为0x01，表示此接口的工作设置。它包含标准及类特定接口和端点描述符。

*bNumEndpoints*为0x01，表示端点与工作设置1标准接口相关。

表5-17. 耳机——工作标准AS接口描述符（备用设置1）

偏移量	字段	大小	值	说明
0	*bLength*	1	0x09	此描述符的大小（字节）
1	*bDescriptorType*	1	0x04	接口描述符
2	*bInterfaceNumber*	1	0x01	从0开始的值，标识此配置支持的并发接口阵列中的索引
3	*bAlternateSetting*	1	0x01	此接口的备用设置数量为1
4	*bNumEndpoints*	1	0x01	一个端点与此接口相关
5	*bInterfaceClass*	1	0x01	音频接口类
6	*bInterfaceSubClass*	1	0x02	音频流接口子类
7	*bInterfaceProtocol*	1	0x00	未定义协议
8	*iInterface*	1	0x00	此接口无字符串描述符

耳机——类特定AS通用接口描述符

下表提供了USB耳麦应用中耳机功能的类特定AS通用接口描述符的详细信息。

*bTerminalLink*设置为0x01。这是与耳机音频功能关联的输入端子的端子ID。

表5-18. 耳机——类特定的AS通用接口描述符

偏移量	字段	大小	值	说明
0	*bLength*	1	0x07	此描述符的大小（字节）
1	*bDescriptorType*	1	0x24	类特定的接口描述符
2	*bDescriptorSubtype*	1	0x01	AS_GENERAL描述符子类型
3	*bTerminalLink*	1	0x01	此接口连接的输入或输出端子的端子ID
4	*bDelay*	1	0x01	为通道间数据通信引入了1个帧传输的延时
5	*wFormatTag*	2	0x0001	与此接口通信时使用的PCM数据格式

耳机——类型1格式类型描述符

下表提供了USB耳麦应用中耳机功能的类型1格式类型的详细信息。

*bFormatType*设置为0x01。这表示实现的是TYPE_I音频格式，该格式的音频基于逐个采样转换机制（例如PCM）构建。

*bNrChannels*为0x02，表示2个音频通道（左通道和右通道）。

*bSubFrameSize*为0x02，表示2字节采样。（即，左通道有2字节，右通道有2字节。）

*bBitResolution*为0x10，表示2字节采样的全部16位均有效。

*bSamFreqType*为0x03，表示支持三种采样频率。

表5-19. 耳机——类型1格式类型描述符

偏移量	字段	大小	值	说明
0	*bLength*	1	0x11	此描述符的大小（字节）
1	*bDescriptorType*	1	0x24	类特定的接口描述符
2	*bDescriptorSubtype*	1	0x02	FORMAT_TYPE描述符子类型
3	*bFormatType*	1	0x01	FORMAT_TYPE_I音频流接口
4	*bNrChannels*	1	0x02	音频流中的通道数为2
5	*bSubFrameSize*	1	0x02	每个音频子帧（采样）为2字节
6	*bBitResolution*	1	0x10	每个采样16位
7	*bSamFreqType*	1	0x03	支持三种频率
8	*tSamFreq[1]*	3	0x003E80	16000 Hz采样率
11	*tSamFreq[2]*	3	0x007D00	32000 Hz采样率
14	*tSamFreq[3]*	3	0x00BB80	48000 Hz采样率

耳机——标准AS音频数据端点描述符

下表提供了音频耳麦应用中实现的耳机功能的标准AS音频数据端点描述符的详细信息。

*bEndpointAddress*设置为0x01。这表示选择端点1作为来自PC的耳机数据的输出端点。

*bmAttributes*为0x09，表示端点为同步自适应端点。

*wMaxPacketSize*为0x00C0，表示支持采样速率16000 Hz、32000 Hz和48000 Hz的最大数据包大小，计算方法如下：

- 采样数/ms = 采样率（Hz）/1000
- 每个采样的大小 = (通道数) * (每个采样的宽度（位深度）)
- 因此，每个帧（数据包）的字节数（wMaxPacketSize）= 采样数/ms * 每个采样的大小

采样率	位深度	通道数	wMaxPacket大小
16 kHz	16位（2字节）	2	64
32 kHz	16位（2字节）	2	128
48 kHz	16位（2字节）	2	192

表5-20. 耳机——标准AS音频数据端点描述符

偏移量	字段	大小	值	说明
0	*bLength*	1	0x09	此描述符的大小（字节）
1	*bDescriptorType*	1	0x05	端点描述符类型
2	*bEndpointAddress*	1	0x01	端点1配置为输出
3	*bmAttributes*	1	0x09	同步自适应端点
4	*wMaxPacketSize*	2	0x00C0	支持48000 Hz采样频率的最大数据包大小
6	*bInterval*	1	0x01	为进行数据传输而轮询端点时的时间间隔（ms）
7	*bRefresh*	1	0x00	无刷新
8	*bSyncAddress*	1	0x00	无同步端点

耳机——类特定AS音频数据端点描述符

下表提供了USB耳麦应用中耳机功能的类特定AS音频数据端点描述符的详细信息。

*bmAttributes*设置为0x01。开启采样频率控制。Bit D7为“0”，表示端点并非一定需要*wMaxPacketSize*的USB数据包，它可以处理短数据包。

表5-21. 耳机——类特定AS音频数据端点描述符

偏移量	字段	大小	值	说明
0	*bLength*	1	0x07	此描述符的大小（字节）
1	*bDescriptorType*	1	0x25	CS_ENDPOINT描述符类型
2	*bDescriptorSubType*	1	0x01	EP_GENERAL描述符子类型
3	*bmAttributes*	1	0x01	开启采样频率控制，无音高控制并且无数据包填充
4	*bLockDelayUnits*	1	0x00	表示wLockDelay字段使用的单位
5	*wLockDelay*	1	0x00	该端点可靠地锁定其内部时钟恢复电路所花费的时间

麦克风——零带宽标准AS接口描述符（备用设置0）

下表提供了USB耳麦应用中麦克风功能的零带宽标准AS接口描述符（备用设置0）的详细信息。

*bInterfaceNumber*设置为0x02。用于在此配置下的并发接口阵列中标识此接口的值。

*bAlternateSetting*为0x00，表示零带宽设置，用于在不使用音频设备时放弃总线上声明的带宽。

*bNumEndpoints*为0x00，表示由于是零带宽接口，因此没有相关端点。

表5-22. 麦克风——零带宽标准AS接口描述符（备用设置0）

偏移量	字段	大小	值	说明
0	*bLength*	1	0x09	此描述符的大小（字节）
1	*bDescriptorType*	1	0x04	接口描述符
2	*bInterfaceNumber*	1	0x02	接口编号。从0开始的值，标识此配置支持的并发接口阵列中的索引
3	*bAlternateSetting*	1	0x00	此接口的备用设置数量为0
4	*bNumEndpoints*	1	0x00	零带宽接口没有相关端点
5	*bInterfaceClass*	1	0x01	音频接口类
6	*bInterfaceSubClass*	1	0x02	音频流接口子类
7	*bInterfaceProtocol*	1	0x00	未定义协议
8	*iInterface*	1	0x00	此接口无字符串描述符

麦克风——工作标准AS接口描述符（备用设置1）

下表提供了USB耳麦应用中麦克风功能的工作带宽标准AS接口描述符（备用设置1）的详细信息。

*bInterfaceNumber*设置为0x02。用于在此配置下的并发接口阵列中标识此接口的值。

*bAlternateSetting*设置为0x01。这表示此接口的工作设置。它包含标准及类特定接口和端点描述符。

*bNumEndpoints*设置为0x01。这表示端点与工作设置1标准接口相关。

表5-23. 麦克风——工作标准AS接口描述符（备用设置1）

偏移量	字段	大小	值	说明
0	*bLength*	1	0x09	此描述符的大小（字节）
1	*bDescriptorType*	1	0x04	接口描述符
2	*bInterfaceNumber*	1	0x02	接口编号。从0开始的值，标识此配置支持的并发接口阵列中的索引
3	*bAlternateSetting*	1	0x01	此接口的备用设置数量为1
4	*bNumEndpoints*	1	0x01	一个端点与此接口相关
5	*bInterfaceClass*	1	0x01	音频接口类
6	*bInterfaceSubClass*	1	0x02	音频流接口子类
7	*bInterfaceProtocol*	1	0x00	未定义协议
8	*iInterface*	1	0x00	此接口无字符串描述符

麦克风——类特定AS通用接口描述符

下表提供了USB耳麦应用中麦克风功能的类特定AS通用接口描述符的详细信息。

*bTerminalLink*设置为0x06。这是与麦克风音频功能关联的输出端子的端子ID。

表5-24. 麦克风——类特定AS通用接口描述符

偏移量	字段	大小	值	说明
0	*bLength*	1	0x07	此描述符的大小（字节）
1	*bDescriptorType*	1	0x24	类特定的接口描述符
2	*bDescriptorSubtype*	1	0x01	AS_GENERAL描述符子类型
3	*bTerminalLink*	1	0x06	此接口连接的输入或输出端子的端子ID
4	*bDelay*	1	0x00	为通道间数据通信引入了0个帧传输的延时
5	*wFormatTag*	2	0x0001	与此接口通信时使用的PCM数据格式

麦克风——类型1格式类型描述符

下表提供了USB耳麦应用中麦克风功能的类型1格式类型的详细信息。

*bFormatType*设置为0x01。这表示实现的是TYPE_I音频格式，该格式的音频基于逐个采样转换机制（例如PCM）构建。

*bNrChannels*为0x01，表示1个音频通道。单声道音频。

*bSubFrameSize*为0x02，表示2字节采样。

*bBitResolution*为0x10，表示2字节采样的全部16位均有效。

*bSamFreqType*为0x03，表示支持三种采样频率。

表5-25. 麦克风——类型1格式类型描述符

偏移量	字段	大小	值	说明
0	*bLength*	1	0x11	此描述符的大小（字节）
1	*bDescriptorType*	1	0x24	类特定的接口描述符
2	*bDescriptorSubtype*	1	0x02	FORMAT_TYPE描述符子类型
3	*bFormatType*	1	0x01	FORMAT_TYPE_I音频流接口

偏移量	字段	大小	值	说明
4	*bNrChannels*	1	0x01	音频流中的通道数为2
5	*bSubFrameSize*	1	0x02	每个音频子帧（采样）为2字节
6	*bBitResolution*	1	0x10	每个采样16位
7	*bSamFreqType*	1	0x03	支持三种频率
8	*tSamFreq[1]*	3	0x003E80	16000 Hz采样率
11	*tSamFreq[2]*	3	0x007D00	32000 Hz采样率
14	*tSamFreq[3]*	3	0x00BB80	48000 Hz采样率

麦克风——标准AS音频数据端点描述符

下表提供了USB耳麦应用中麦克风功能的标准AS音频数据端点描述符的详细信息。

*bEndpointAddress*设置为0x81。这表示选择端点1作为发送至PC的麦克风数据的输入端点。

*bmAttributes*为0x0D，表示端点为等时同步发送方端点。

*wMaxPacketSize*为0x0060，表示支持单声道音频采样率16000 Hz、32000 Hz和48000 Hz的最大数据包大小，计算方法如下：

- 采样数/ms = 采样率（Hz）/1000
- 每个采样的大小 = (通道数) * (每个采样的宽度（位深度）)
- 因此，每个帧（数据包）的字节数（*wMaxPacketSize*）= 采样数/ms * 每个采样的大小

采样率	位深度	通道数	wMaxPacket大小
16 kHz	16位（2字节）	2	64
32 kHz	16位（2字节）	2	128
48 kHz	16位（2字节）	2	192

表5-26. 麦克风——标准AS音频数据端点描述符

偏移量	字段	大小	值	说明
0	*bLength*	1	0x09	此描述符的大小（字节）
1	*bDescriptorType*	1	0x05	端点描述符类型
2	*bEndpointAddress*	1	0x81	端点1配置为输出
3	*bmAttributes*	1	0x0D	等时同步端点
4	*wMaxPacketSize*	2	0x0060	支持单声道音频48000 Hz采样频率的最大数据包大小
6	*bInterval*	1	0x01	为进行数据传输而轮询端点时采用的时间间隔（ms）
7	*bRefresh*	1	0x00	无刷新
8	*bSyncAddress*	1	0x00	无同步端点

麦克风——类特定AS音频数据端点描述符

下表提供了USB耳麦应用中麦克风功能的类特定AS音频数据端点描述符的详细信息。

*bmAttributes*设置为0x01。这将开启采样频率控制。Bit D7为“0”，表示端点并非一定需要*wMaxPacketSize*的USB数据包，它可以处理短数据包。

注：本节未介绍以下描述符的值及其说明，因为本文档中讨论的MPLAB Harmony USB耳麦应用未实现这些描述符。

1. 用于实现异步同步类型的等时同步数据端点描述符。这是因为USB耳麦应用会将耳机数据端点配置为自适应，并将麦克风数据端点配置为同步。
2. 混合器功能单元的关联接口描述符、HID类接口描述符及其用于实现符合HID标准的消费类控制音频命令的报告描述符。

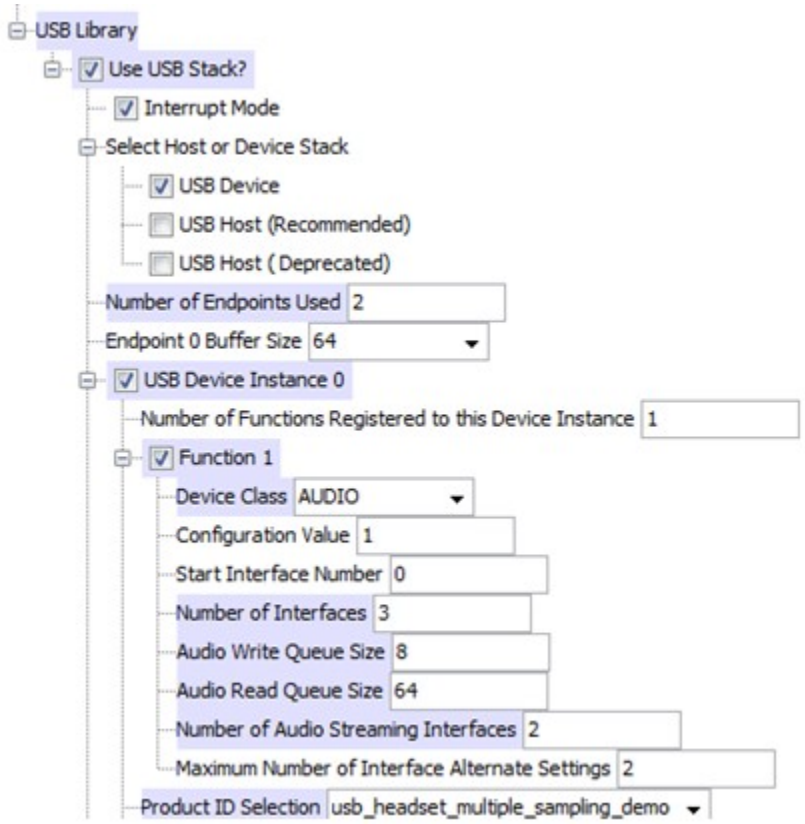
表5-27. 麦克风——类特定AS音频数据端点描述符

偏移量	字段	大小	值	说明
0	*bLength*	1	0x07	此描述符的大小（字节）
1	*bDescriptorType*	1	0x25	CS_ENDPOINT描述符类型
2	*bDescriptorSubType*	1	0x01	EP_GENERAL描述符子类型
3	*bmAttributes*	1	0x01	开启采样频率控制，无音高控制并且无数据包填充。
4	*bLockDelayUnits*	1	0x00	表示wLockDelay字段使用的单位
5	*wLockDelay*	1	0x00	该端点可靠地锁定其内部时钟恢复电路所花费的时间

5.2 使用MPLAB Harmony配置器（MHC）配置USB音频1.0库

1. 打开MPLAB X IDE，启动MPLAB Harmony配置器（MHC）。
2. 选择MHC **Options**（选项）选项卡，展开*Harmony Framework Configuration* > *USB Library*（Harmony框架配置 > USB库）。
3. 配置USB音频设备库，如下图所示。

图5-2. MHC——USB音频库



注：以下几点适用于创建USB耳麦应用时使用的USB音频库MHC配置。

- 将**Number of Endpoints Used**（使用的端点数）设置为2。其中一个端点为默认控制端点0，该端点将用于USB设备控制传输和USB音频控制传输。另一个端点将用于从主机向设备传输USB音频数据以及从设备向主机传输USB音频数据。
- 将**Endpoint 0 Buffer Size**（端点0缓冲区大小）保留为默认值64（最大缓冲区大小）。对于全速设备而言，USB规范允许端点0缓冲区的大小为8、16、32或64字节。
- 将**USB Device Instance 0**（USB设备实例0）选项选中并展开。由于设备上只有一个USB外设，因此只有一个USB实例（默认选中）。
- 将**Number of Functions Registered to this Device Instance**（此设备实例所注册函数的数量）保留为默认值1。这表示仅使用一个函数驱动程序（即，音频设备类）。
- 将**Device Class**（设备类）设置为AUDIO（音频）。这表示最终应用将是基于USB音频1.0规范的USB音频应用。
- 将**Configuration Value**（配置值）设置为1（默认值）。这表示函数驱动程序（音频）所连接的配置。将只有一个配置，因此保留默认配置值1。
- 将**Number of Interfaces**（接口数）设置为3。第一个接口将用于传输音频控制（音量和静音）。第二个接口用于从麦克风向PC传输音频数据，第三个接口用于从PC向耳机传输音频数据。

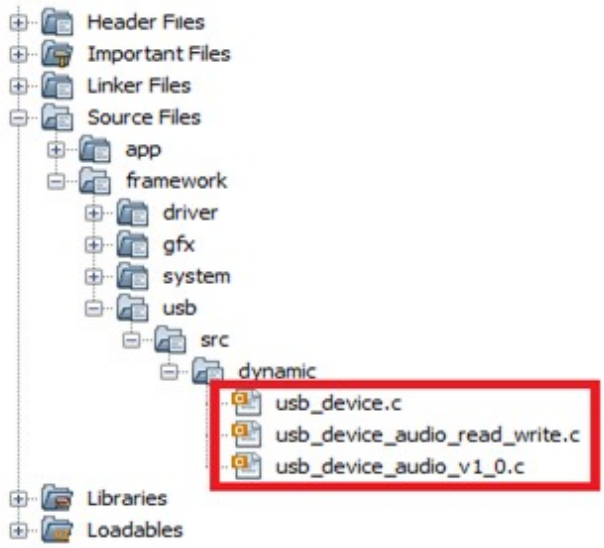
- 设置通过函数驱动程序实现的读写队列大小。读写队列供应用程序来缓冲音频数据。队列大小配置如下：
  - 将**Audio Write Queue Size**（音频写队列大小）设置为8。为了连续传输音频流并避免丢失音频数据，需将写队列大小设置为8，以支持麦克风操作的音频缓冲区队列。
  - 将**Audio Read Queue Size**（音频读队列大小）设置为64。为了连续传输音频流并避免丢失音频数据，需将读队列大小设置为64，以支持耳机操作的音频缓冲区队列。

注：上面分配的队列大小值是基于观察/实验结果得出的。

- 将**Number of Audio Streaming Interfaces**（音频流接口数）设置为2，一个用于麦克风接口，另一个用于耳机接口。
- 将**Maximum Number of Interface Alternate Settings**（接口备用设置的最大数量）保留为默认值2。使用两个备用设置：一个用于重新声明未使用音频功能时的带宽，另一个用于重新声明使用音频功能时的带宽。
- 按如下设置**Product ID Selection**（产品ID选择）：
  - 在产品ID选择中，选择**usb_headset_multiple_sampling_demo**。这表示应用归为USB耳麦类。当选择usb_headset_multiple_sampling_demo时，需填充以下字段，具体如下：
  - **Enter Vendor ID**（输入供应商ID）：0x04D8
  - **Enter Product ID**（输入产品ID）：0x00FF
  - **Manufacturer String**（制造商字符串）：“Microchip Technology Inc.”
  - **Product String**（产品字符串）：“Harmony USB Headset Multiple Sampling Rate Example”（Harmony USB耳麦多采样率示例）

USB音频库按上文所述进行配置后，MHC生成的项目会将必要的USB音频库文件添加到项目中，如下图所示。

图5-3. USB音频库文件



提示：
要配置并开始使用MPLAB Harmony（如使用MHC配置USB音频1.0库和使用MHC配置音频驱动程序章节所示），需打开目标器件的MPLAB X IDE MPLAB Harmony项目（例如，PIC32MX470F512L对应于带AK4642编解码器子板的PIC32蓝牙音频开发工具包）并启动MPLAB Harmony配置器（MHC）。

5.3 使用MPLAB Harmony配置器（MHC）配置音频驱动程序

MPLAB Harmony支持多种可供应用程序使用的编解码器驱动程序库。对于USB耳麦应用，在带PIC32音频编解码器子板AK4642EN的PIC32蓝牙音频开发工具包上，使用MHC配置AK4642编解码器驱动程序，具体如下。

注：AK4642编解码器器件使用I^2C作为控制接口，使用I^2S作为与PIC32单片机之间的数据接口介质。MPLAB Harmony AK4642编解码器驱动程序库以I^2C和I^2S驱动程序库为基础。AK4642编解码器驱动程序还使用DMA和时钟系统服务；因此以下部分除了介绍AK4642编解码器驱动程序的配置外，还介绍这些驱动程序和系统服务的配置。

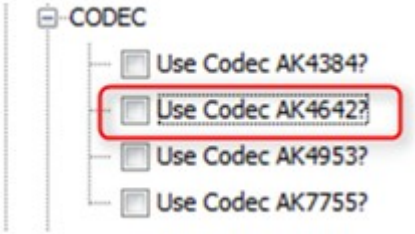
在MHC中，选择**Options**选项卡，展开*Harmony Framework Configuration*。

5.3.1 配置AK4642编解码器驱动程序

要配置AK4642编解码器驱动程序，请按以下步骤操作：

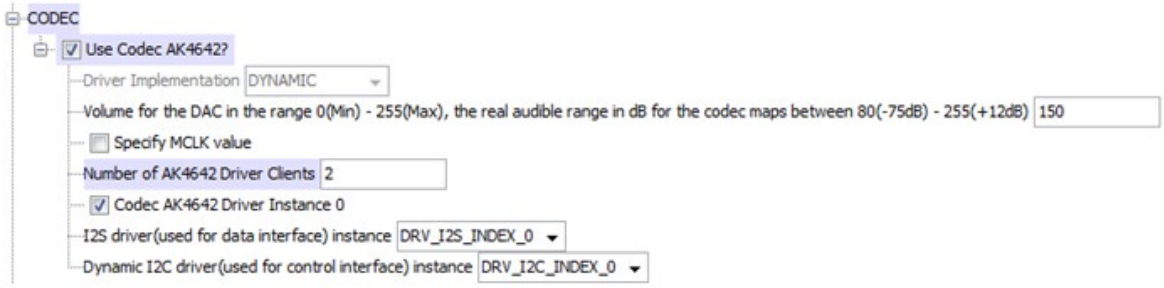
1. 展开*Drivers* > *CODEC*（驱动程序 > 编解码器），选择**Use Codec AK4642**（使用编解码器AK4642）。

图5-4. MHC——编解码器驱动程序



2. 配置AK4642编解码器驱动程序，如下图所示。

图5-5. MHC——编解码器AK4642驱动程序



注：以下几点适用于创建USB耳麦应用时使用的编解码器驱动程序配置。

- 将Number of AK4642 Driver Clients（AK4642驱动程序客户端的数量）设置为2。一个客户端用于麦克风，另一个客户端用于耳机。

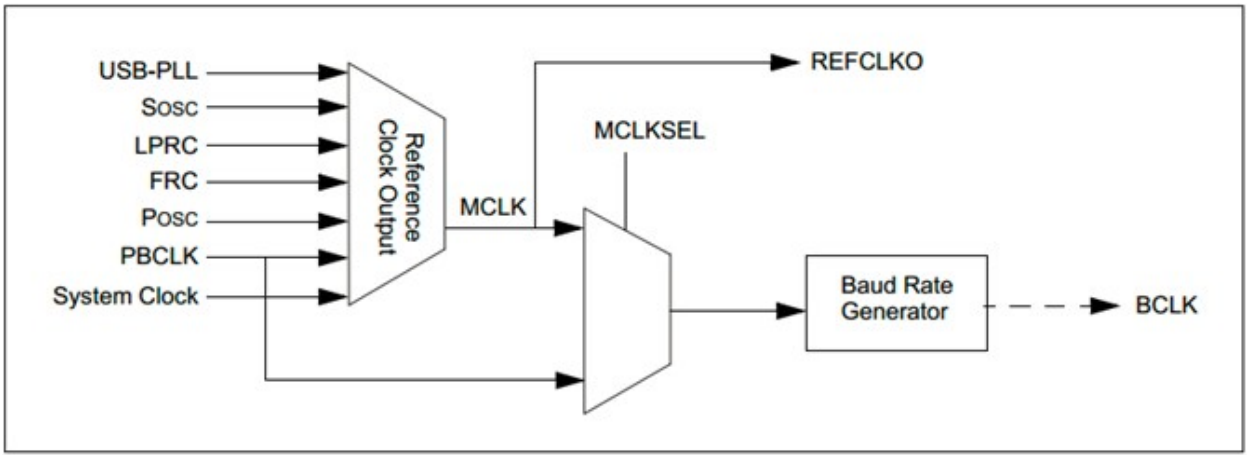
- 未选择Specify MCLK value（指定MCLK值），因为I²S驱动程序配置下为MCLK Sampling rate Multiplier（MCLK采样率倍数）选择了默认值256。

注：为编解码器AK4642驱动程序中的*Specify MCLK value*和I²S驱动程序中的MCLK Sampling rate Multiplier配置相同参数，具体如下所述。

AK4642编解码器器件需要一个输入时钟源来为其产生精确的音频采样率。PIC32MX470F512L器件可提供灵活的参考时钟输出（REFCLKO）。参考时钟输出用于生成供AK4642编解码器器件使用的小数时钟，以适应各种采样率。

下图所示为PIC32MX470F512L上用于生成AK4642编解码器小数时钟的REFCLKO电路。

图5-6. 编解码器主时钟（MCLK）和位时钟（BCLK）生成器



AK4642需要主时钟（MCLK），该时钟值等于工作采样频率乘以MCLK倍数（见编解码器AK4642驱动程序配置选项中的“指定MCLK值”和I²S驱动程序配置选项中的“MCLK采样速率倍数”）。

下图所示为编解码器支持的MCLK倍数值。

表5-28. 编解码器在不同采样率下所需的典型主时钟（MCLK）和位时钟（BCLK）

采样率（kHz）	不同过采样率下的编解码器主时钟/时基（MHz）							位时钟（MHz）	
fs	128 fs	192 fs	256 fs	384 fs	512 fs	768 fs	1152 fs	32 fs	64 fs
32.0	-	-	8.1920	12.2880	16.3840	24.5760	36.8640	1.024	2.0480
44.1	-	-	11.2896	16.9344	22.5792	33.8688	-	1.4112	2.8224
48.0	-	-	12.2880	18.4320	24.5760	36.8640	-	1.536	3.0720
88.2	11.2896	16.9344	22.5792	33.8688	-	-	-	2.8224	5.6448
96.0	12.2880	18.4320	24.5760	36.8640	-	-	-	3.072	6.1440
176.4	22.5792	33.8688	-	-	-	-	-	5.6448	11.2896
192.0	24.5760	36.8640	-	-	-	-	-	6.144	12.2880

例如，选择256 fs选项时，MCLK倍数值为256，fs为采样频率。

如果采样频率为48 kHz且MCLK倍数值为256，则主时钟（MCLK）= 256 * 48000 = 12288000 Hz。

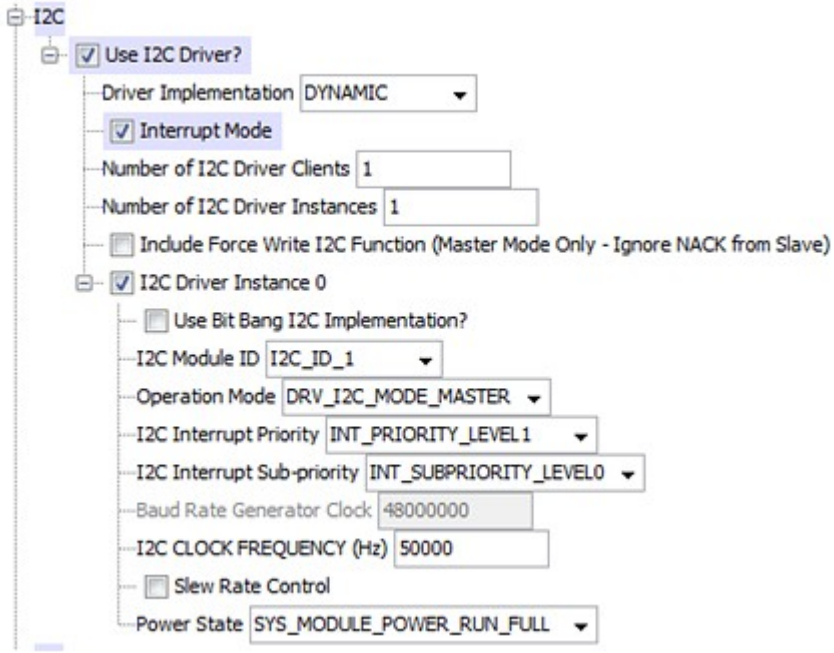
默认情况下，AK4642编解码器驱动程序将MCLK倍数值设置为256。因此，不选择该选项。如果需要其他MCLK倍数值，则必须选择Specify MCLK value选项。该选项有多种可供选择，分别为128、192、256、384、512、768和1152。

5.3.2 配置I²C驱动程序

要配置I²C驱动程序，请按以下步骤操作：

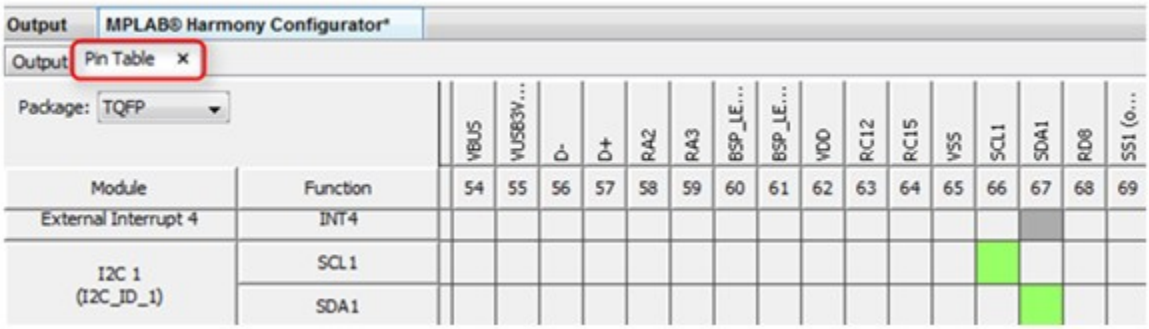
1. 展开*Drivers* > *I²C*（驱动程序 > I²C），选择**Use I²C Driver?**（使用I²C驱动程序？），然后配置I²C驱动程序，如下图所示。

图5-7. MHC——I²C驱动程序



I²C模块使用的I/O引脚通过图形式引脚管理器进行配置，如下图所示。

图5-8. MHC——引脚映射表——I²C驱动程序



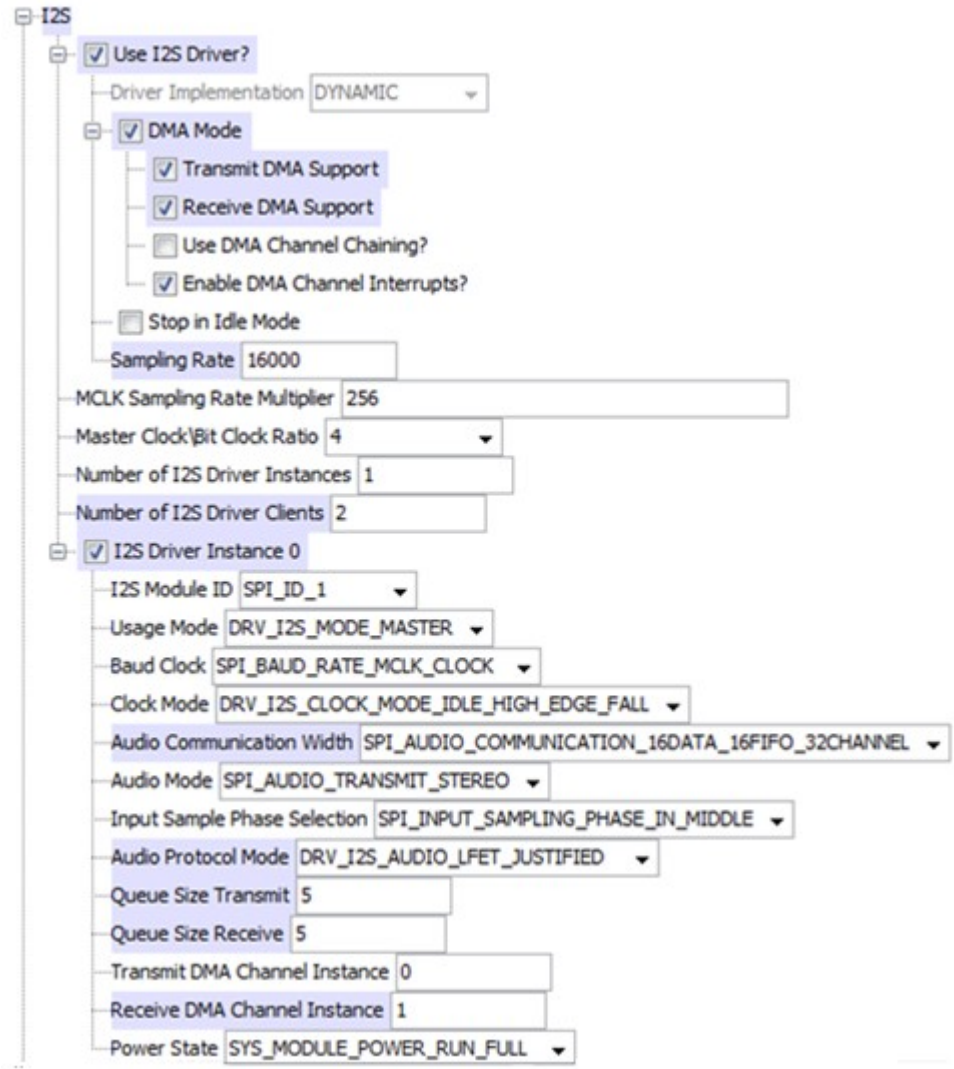
选择**Pin Diagram**（引脚图）子选项卡，然后在MHC输出窗格（IDE窗口底部）中选择**Pin Table**（引脚表）选项卡。

5.3.3 配置I²S驱动程序

要配置I²S驱动程序，请按以下步骤操作：

1. 展开*Drivers* > *I²S*。然后选择**Use I²S Driver?**并配置I²S驱动程序，如下图所示。

MHC——I²S驱动程序



注：

以下几点适用于创建USB耳麦应用时使用的I²S驱动程序配置：

- 选择DMA工作模式。在DMA Mode（DMA模式）下，选择Transmit DMA Support（发送DMA支持）和Receive DMA Support（接收DMA支持）。默认情况下选择的是Enable DMA Channel Interrupts?（允许DMA通道中断？）选项。使能发送和接收DMA支持的目的是使用DMA通道在存储器和I²S外设缓冲区之间传输音频数据。允许DMA通道中断的目的是允许DMA传输完成通知事件。
- 为MCLK Sampling rate Multiplier保留默认值256。请参见上文“编解码器AK4642驱动程序配置”部分中的MCLK倍数值说明。

- 为Master Clock/Bit Clock ratio（主时钟/位时钟比）保留默认值4。由PIC32器件提供给编解码器的位时钟（BCLK）来自主时钟（MCLK）。给定BCLK倍数和采样率组合（例如，32 fs和64 fs）能够生成的常见位时钟（BCLK）将为MCLK/1、MCLK/2、MCLK/4 或 MCLK/8。
  - 例如，BCLK值为64 fs，采样速率为48000 Hz。

    BCLK将为64 * 48000 = 3072000 Hz。

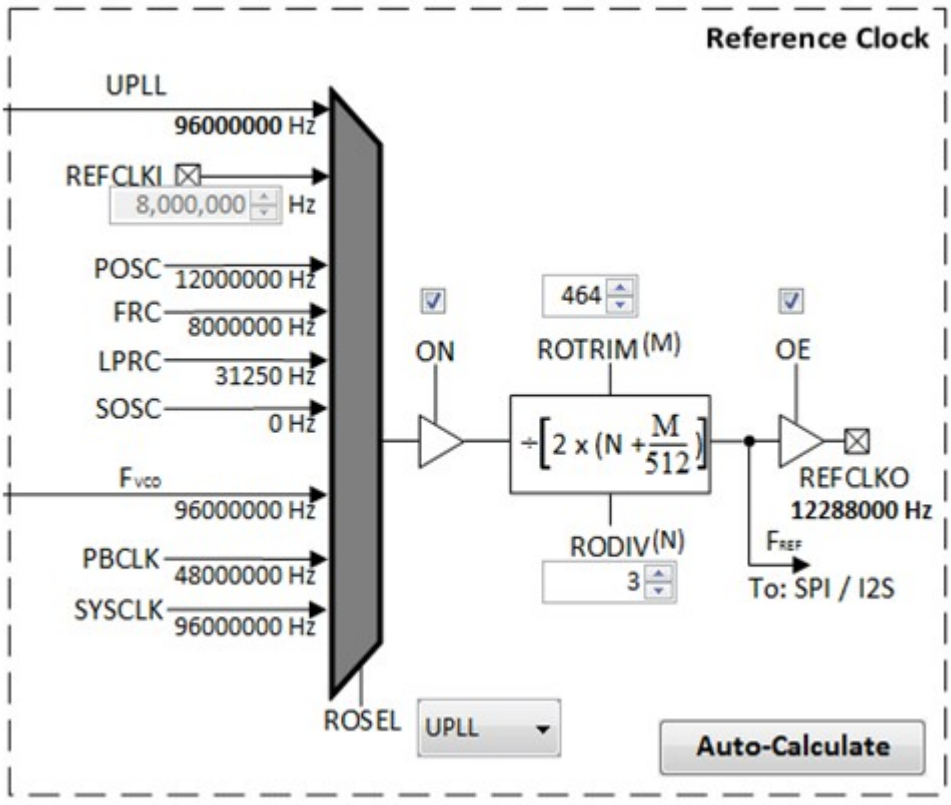
    因此，MCLK/BCLK比为12288000/3072000 = 4。
- 将Usage Mode（使用模式）设置为DRV_I2S_MODE_MASTER。这将指示I^2S实例用作主器件还是从器件。在主器件模式下，PIC32将为从器件生成BCLK。在从器件模式下，PIC32将接收来自I^2S主器件的BCLK。在AK4642编解码器接口中，PIC32 I^2S用作主器件，生成BCLK。
- 将Audio Communication Width（音频通信宽度）设置为SPI_AUDIO_COMMUNICATION_16DATA_16FIFO_32CHANNEL，表示支持CD音质的音频播放以及设置每通道16位音频数据和32位通道宽度。
- 将Audio Mode（音频模式）设置为SPI_AUDIO_TRANSMIT_STEREO，该设置对应于立体声音频播放。
- 将Input Sample Phase Selection（输入采样阶段选择）设置为SPI_INPUT_SAMPLING_PHASE_IN_MIDDLE，该设置对应于I^2S协议的默认阶段。
- 将Audio Protocol Mode（音频协议模式）设置为DRV_I2S_AUDIO_LEFT_JUSTIFIED，因为这是编解码器AK4642支持的协议模式。
- 将Queue Size Transmit（发送队列大小）和Queue Size Receive（接收队列大小）均设置为值5，这是基于观察/实验结果设置的。

5.3.4 配置编解码器时钟

采样率、MCLK倍数和MCLK/BCLK比为MHC中的时钟配置器提供输入，以通过参考时钟电路生成所需的编解码器输入主时钟。

1. 展开*System Services* > *Clock* > *Use Clock System Service?*（系统服务 > 时钟 > 使用时钟系统服务？）。
2. 配置参考时钟，如下图所示。

   MHC——参考时钟生成器



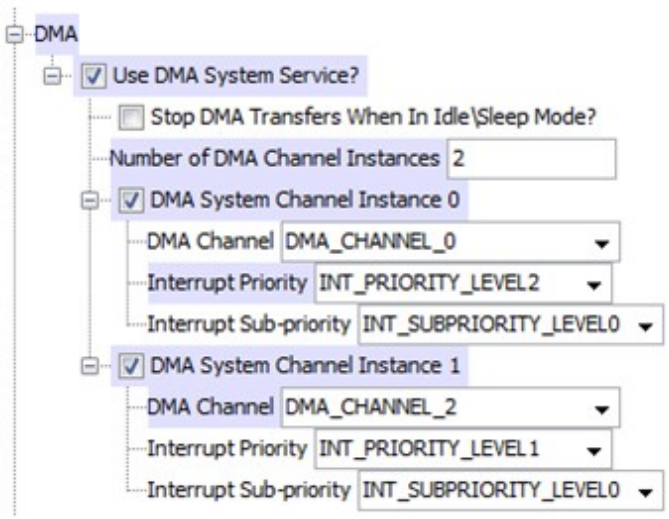
3. 单击**Auto-Calculate**（自动计算），检查Reference Clock Divisor（参考时钟分频比）和Trim Auto Calculator（微调自动计算器）窗口是否显示所需参考时钟和采样率值。

5.3.5 配置DMA系统服务

将I²S驱动程序配置为使用DMA通道与编解码器交换数据。要配置和验证DMA系统服务，请按以下步骤操作：

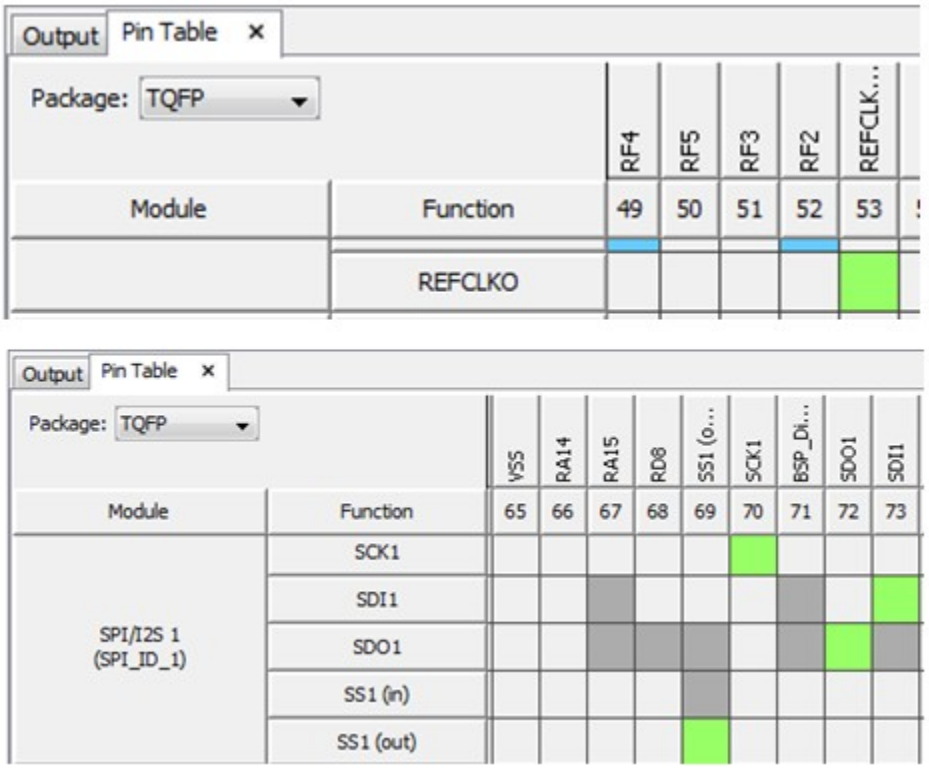
1. 展开*System Services* > *DMA*（系统服务 > DMA），然后选择**Use Clock System Service?**（使用时钟系统服务？）并配置DMA系统服务，如下图所示。

图5-9. MHC——DMA系统服务



注：I²S模块使用的I/O引脚通过图形式引脚管理器进行配置，如下图所示。

图5-10. 参考时钟和I²S驱动程序的引脚映射

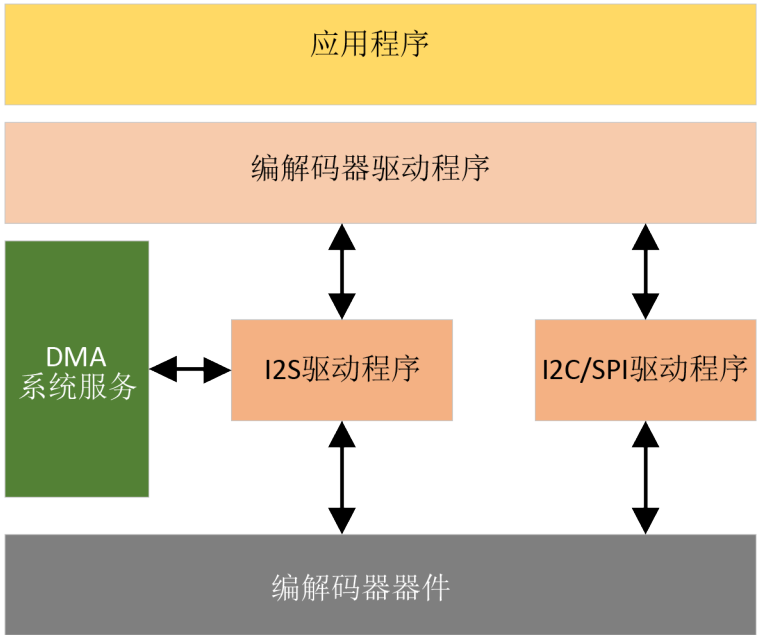


5.4 音频驱动程序的架构与使用

5.4.1 架构概述

MPLAB Harmony通过为PIC32开发板上安装的编解码器器件提供驱动程序来支持音频开发。下图显示了编解码器驱动程序在MPLAB Harmony软件框架中的位置。编解码器驱动程序使用SPI/I²C和I²S驱动程序来进行控制以及向编解码器模块传输音频数据。I²S驱动程序使用DMA通道（通过DMA系统服务）在用户缓冲区和I²S外设数据寄存器之间发送和接收音频数据。

图5-11. 音频驱动程序架构



5.4.2 编解码器驱动程序库API概述

编解码器驱动程序库提供API来向串行连接的编解码器模块传输控制命令和数字音频数据。MPLAB Harmony中的编解码器驱动程序实现了一个常用接口，如下表所示。

库接口部分		说明
系统函数	DRV_CODEC_Initialize DRV_CODEC_Deinitialize DRV_CODEC_Status DRV_CODEC_Tasks	提供系统模块接口、器件初始化、取消初始化、重新初始化、任务和状态函数。
客户端设置函数	DRV_CODEC_Open DRV_CODEC_Close	提供打开和关闭函数。
编解码器特定的函数	示例：对于DAC AK4384 DRV_AK4384_ZeroDetectEnable	提供编解码器模块特定的函数。
数据传输函数	DRV_CODEC_BufferAddWrite DRV_CODEC_BufferAddRead DRV_CODEC_BufferAddWriteRead DRV_CODEC_BufferEventHandlerSet	提供数据传输函数。
其他函数	DRV_CODEC_SamplingRateSet DRV_CODEC_SamplingRateGet DRV_CODEC_VolumeSet	提供驱动程序特定的其他函数，例如采样率设置和控制命令函数等

库接口部分		说明
	DRV_CODEC_VolumeGet DRV_CODEC_MuteOn DRV_CODEC_MuteOff	

注：

之前的API将映射到system_config.h文件中编解码器特定的API。在USB耳麦应用中，通用编解码器API将映射到AK4642编解码器特定的API，如下面示例所示。

编解码器API映射

```
#define DRV_CODEC_Initialize
DRV_AK4642_Initialize
#define DRV_CODEC_Deinitialize
DRV_AK4642_Deinitialize
#define DRV_CODEC_Status                                    DRV_AK4642_Status
#define DRV_CODEC_Tasks                                     DRV_AK4642_Tasks
#define DRV_CODEC_Open                                      DRV_AK4642_Open
#define DRV_CODEC_Close                                     DRV_AK4642_Close
#define DRV_CODEC_BufferEventHandlerSet
DRV_AK4642_BufferEventHandlerSet
#define DRV_CODEC_BufferAddWrite
DRV_AK4642_BufferAddWrite
#define DRV_CODEC_BufferAddRead
DRV_AK4642_BufferAddRead
#define DRV_CODEC_BufferAddWriteRead
DRV_AK4642_BufferAddWriteRead
#define DRV_CODEC_SamplingRateSet
DRV_AK4642_SamplingRateSet
#define DRV_CODEC_SamplingRateGet
DRV_AK4642_SamplingRateGet
#define DRV_CODEC_VolumeSet
DRV_AK4642_VolumeSet
#define DRV_CODEC_VolumeGet
DRV_AK4642_VolumeGet
#define DRV_CODEC_MuteOn                                    DRV_AK4642_MuteOn
#define DRV_CODEC_MuteOff                                   DRV_AK4642_MuteOff
#define DRV_CODEC_MicrophoneTypeSet
DRV_AK4642_IntExtMicSet
#define DRV_CODEC_MicrophoneSoundSet
DRV_AK4642_MonoStereoMicSet
#define DRV_CODEC_CommandEventHandlerSet
DRV_AK4642_CommandEventHandlerSet
```

5.4.3 使用编解码器驱动程序库

对于USB耳麦应用，以下示例给出了编解码器、I^2S和I^2C驱动程序以及DMA系统服务的数据结构。这些数据结构位于system_init.c文件中，通过MHC基于所选配置生成。

编解码器驱动程序数据初始化

```
/*** 编解码器驱动程序初始化数据 ***/
const DRV_AK4642_INIT drvak4642Codec0InitData =
{
    .moduleInit.value = SYS_MODULE_POWER_RUN_FULL,
    .i2sDriverModuleIndex = DRV_AK4642_I2S_DRIVER_MODULE_INDEX_IDX0,
    .i2cDriverModuleIndex = DRV_AK4642_I2C_DRIVER_MODULE_INDEX_IDX0,
    .volume = DRV_AK4642_VOLUME,
};
```

I²S驱动程序数据初始化

```
/*** I2S驱动程序初始化数据 ***/ const
DRV_I2S_INIT drvI2S0InitData =
{
    .moduleInit.value = DRV_I2S_POWER_STATE_IDX0,
    .spiID = DRV_I2S_PERIPHERAL_ID_IDX0,
    .usageMode = DRV_I2S_USAGE_MODE_IDX0,
    .baudClock = SPI_BAUD_RATE_CLK_IDX0,
    .baud = DRV_I2S_BAUD_RATE,
    .clockMode = DRV_I2S_CLK_MODE_IDX0,
    .audioCommWidth = SPI_AUDIO_COMM_WIDTH_IDX0,
    .audioTransmitMode = SPI_AUDIO_TRANSMIT_MODE_IDX0,
    .inputSamplePhase = SPI_INPUT_SAMPLING_PHASE_IDX0,
    .protocolMode = DRV_I2S_AUDIO_PROTOCOL_MODE_IDX0,
    .queueSizeTransmit = QUEUE_SIZE_TX_IDX0,
    .queueSizeReceive = QUEUE_SIZE_RX_IDX0,
    .dmaChannelTransmit = DRV_I2S_TX_DMA_CHANNEL_IDX0,
    .txInterruptSource = DRV_I2S_TX_INT_SRC_IDX0,
    .dmaInterruptTransmitSource = DRV_I2S_TX_DMA_SOURCE_IDX0,
    .dmaChannelReceive = DRV_I2S_RX_DMA_CHANNEL_IDX0,
    .rxInterruptSource = DRV_I2S_RX_INT_SRC_IDX0,
    .dmaInterruptReceiveSource = DRV_I2S_RX_DMA_SOURCE_IDX0,
};
```

I²C驱动程序数据初始化

```
/* I2C驱动程序初始化数据
*/

const DRV_I2C_INIT drvI2C0InitData =
{
    .i2cId = DRV_I2C_PERIPHERAL_ID_IDX0,
    .i2cMode = DRV_I2C_OPERATION_MODE_IDX0,
    .baudRate = DRV_I2C_BAUD_RATE_IDX0,
    .busspeed = DRV_I2C_SLEW_RATE_CONTROL_IDX0,
    .buslevel = DRV_I2C_SMBus_SPECIFICATION_IDX0,
    .mstrInterruptSource = DRV_I2C_MASTER_INT_SRC_IDX0,
    .errInterruptSource = DRV_I2C_ERR_MX_INT_SRC_IDX0,
};
```

DMA系统服务数据初始化

```
/*** 系统DMA初始化数据 ***/

const SYS_DMA_INIT sysDmaInit =
{
    .sidl = SYS_DMA_SIDL_DISABLE,

};
```

在系统初始化期间，AK4642编解码器驱动程序库通过system_init.c文件中的SYS_Initialize函数进行初始化，方法是调用DRV_AK4642_Initialize函数，如下面示例所示。

AK4642编解码器驱动程序初始化

```
sysObj.drvak4642Codec0 = DRV_AK4642_Initialize(DRV_AK4642_INDEX_0,
(SYS_MODULE_INIT *)&drvak4642Codec0InitData);
```

system_interrupt.c包含已配置I²C和DMA系统服务的中断处理程序，如下面示例所示。

DMA和I²C中断服务程序

```
//
*************************************************************************
//
*************************************************************************
// 部分：系统中断向量函数
//
*************************************************************************
//
*************************************************************************

void ___ISR(_DMA0_VECTOR, ipl2AUTO) _IntHandlerSysDmaCh0(void)
{
    SYS_DMA_TasksISR(sysObj.sysDma, DMA_CHANNEL_0); (1)
}

void ___ISR(_DMA2_VECTOR, ipl1AUTO) _IntHandlerSysDmaCh1(void)
{
    SYS_DMA_TasksISR(sysObj.sysDma, DMA_CHANNEL_2); (2)
}

void ___ISR(_I2C_1_VECTOR, ipl1AUTO) _IntHandlerDrvI2CInstance0(void)
{
    DRV_I2C_Tasks(sysObj.drvI2C0); (3)

}
```

DMA系统服务提供I²S驱动程序缓冲区完成事件的中断触发信号。使用两个单独的DMA通道中断（一个用于发送，另一个用于接收）。处理音频数据发送的I²S驱动程序任务（DRV_I2S_WriteTasks）和处理音频数据接收的驱动程序任务（DRV_I2S_ReadTasks）通过DMA通道ISR中的DMA系统服务任务（SYS_DMA_TasksISR）调用，这两个驱动程序任务在上一个示例中标记为“1”和“2”。

I²C驱动程序任务处理函数通过I2C ISR调用，该函数在上图中标记为“3”。

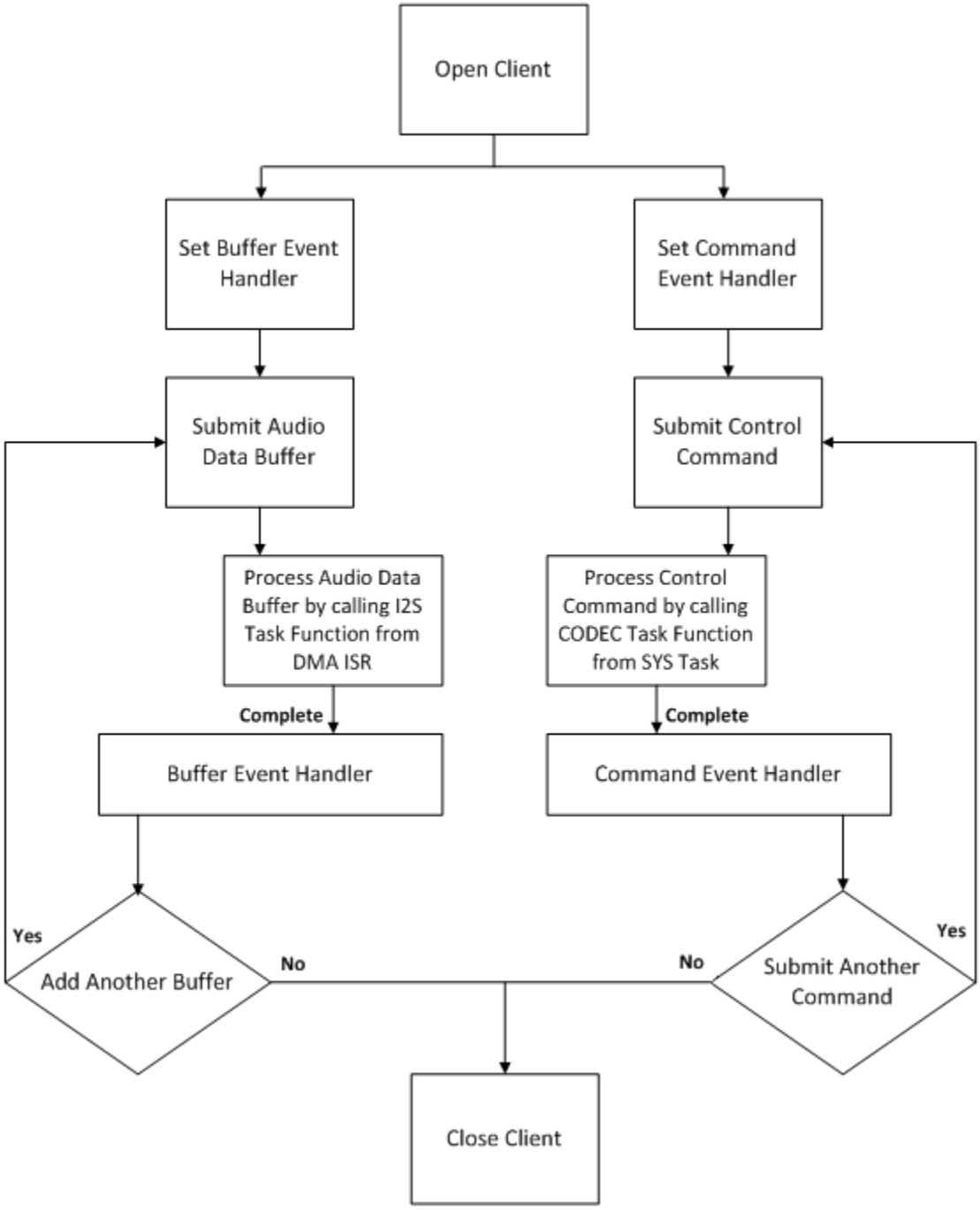
system_tasks.c文件包含SYS_Tasks程序，该程序为编解码器驱动程序运行状态机，如下面示例所示。

AK4642编解码器任务程序系统调用

```
void SYS_Tasks ( void )
{
    /* 维护系统服务*/

    /* 维护设备驱动程序 */
    DRV_AK4642_Tasks(sysObj.drvak4642Codec0);

    /* 维护中间件和其他库 */

     /* USB FS驱动程序任务程序 */
     DRV_USBFS_Tasks(sysObj.drvUSBObject);

    /* USB设备层任务程序 */
    USB_DEVICE_Tasks(sysObj.usbDevObject0);

    /* 维护应用程序的状态机。*/
    APP_Tasks();
}
```

下图所示为应用程序使用编解码器驱动程序的流程。

图5-12. 编解码器驱动程序使用流程



应用程序首先通过调用具有相同模块索引的DRV_CODEC_Open函数来将AK4642编解码器驱动程序打开两次。该操作将获得两个驱动程序句柄。在打开的两个客户端中，一个用于写音频数据（耳机功能），另一个用于读数据（麦克风功能）。

编解码器打开

```
appData.codecClientWrite.handle = DRV_CODEC_Open (DRV_CODEC_INDEX_0,
DRV_IO_INTENT_WRITE);

appData.codecClientRead.handle = DRV_CODEC_Open (DRV_CODEC_INDEX_0,
DRV_IO_INTENT_READ);
```

AK4642编解码器驱动程序将提供一个回调通知函数，用于向应用程序通知数据传输请求何时完成。回调函数通过驱动程序调用DRV_CODEC_BufferEventHandlerSet function的方式来注册。

编解码器缓冲区处理程序设置

```
.codecClientWrite.bufferHandler = (DRV_CODEC_BUFFER_EVENT_HANDLER)
APP_CODECBufferEventHandler,
.
.
.
.codecClientRead.bufferHandler = (DRV_CODEC_BUFFER_EVENT_HANDLER)
APP_CODECBufferEventHandlerRead,
```

应用程序通过调用DRV_CODEC_BufferAddWrite和DRV_CODEC_BufferAddRead函数来提交音频缓冲区写/读请求，如下面示例所示。

编解码器缓冲区写/读

```
DRV_CODEC_BufferAddWrite(appData.codecClientWrite.handle,
                                  &current->writeHandle,
                                  current->buffer,
                                  appData.codecClientWrite.buffersize);

DRV_CODEC_BufferAddRead(appData.codecClientRead.handle,
                        &appData.codecClientRead.writeBufHandle1,
                        appData.codecClientRead.txbufferObject1,
                        appData.codecClientRead.bufferSize);
```

当编解码器驱动程序完成音频数据写/读请求时，它将回调注册的事件处理程序函数APP_CODECBufferEventHandler和APP_CODECBufferEventHandlerRead。应用程序通过实现相应逻辑来处理音频数据缓冲区发送和接收请求，最终实现耳机和麦克风功能。

5.5 USB音频库的架构与使用

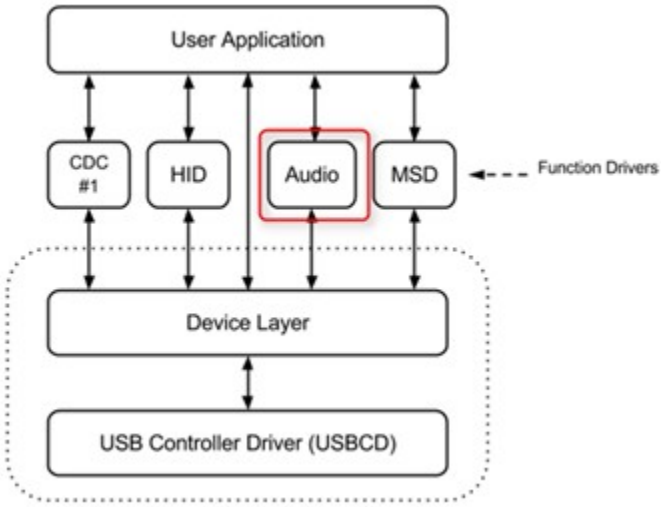
5.5.1 USB音频库架构概述

USB设备库具有模块化分层框架，使开发人员能够设计和开发各种USB设备。USB设备库通过实现标准USB设备类规范的函数驱动程序来简化标准USB设备的开发。该库由以下三个主要部分组成：

- USB控制器驱动程序（USB Controller Driver，USBCD）
- 设备层
- 函数驱动程序

音频函数驱动程序为USB音频设备提供各种服务，通过抽象USB规范详细信息与主机进行通信。音频函数驱动程序与USB设备层和USB控制器配合工作以与USB主机进行通信。下图所示为MPLAB Harmony USB设备协议栈架构的框图。

图5-13. USB库架构



USB控制器驱动程序负责管理设备上的USB外设。USB设备层处理设备枚举等，USB设备层将所有音频特定控制传输转发到音频函数驱动程序。

5.5.2 USB音频库API概述

MPLAB Harmony USB音频设备库具有可实现USB音频设备的程序。下表列出了USB音频设备库API及其描述符。

表5-29. USB音频设备库API

库接口部分	说明
USB_DEVICE_AUDIO_EventHandlerSet	该函数为指定音频函数驱动程序实例注册事件处理程序。
USB_DEVICE_AUDIO_Read	该函数请求从USB设备音频函数驱动程序层读取数据。
USB_DEVICE_AUDIO_Write	该函数请求向USB设备音频函数驱动程序层写入数据。

5.5.3 使用USB音频库

基于在MHC中选择的配置选项，可通过MHC在`system_init.c`文件中为USB耳麦应用生成USB设备层说明以及以下两个数据结构：

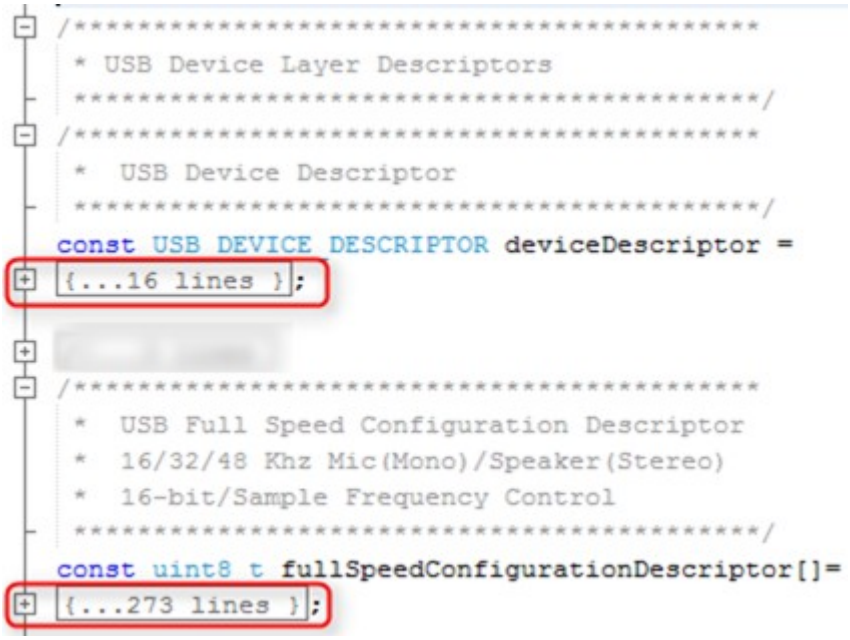
- DRV_USBFS_INIT（USB驱动程序的初始化数据结构）
- USB_DEVICE_AUDIO_INIT（音频函数驱动程序的初始化数据结构）

下图所示为`system_init.c`中的DRV_USBFS_INIT数据结构。

图5-14. USB驱动程序数据初始化

下图所示为system_init.c中的USB设备层描述符的初始化数据结构。

图5-15. USB设备层数据初始化



当主机对配置进行设置时，音频函数驱动程序通过设备层进行初始化。

音频函数驱动程序初始化数据结构USB_DEVICE_AUDIO_INIT通过读写队列大小进行初始化。音频函数驱动程序实例的函数驱动程序注册表项的funcDriverInit成员指向此初始化数据结构。USB_DEVICE_AUDIO_FUNCTION_DRIVER数据结构为设备层提供音频函数驱动函数的入口点。以下示例说明了如何通过设备层注册音频函数驱动程序。

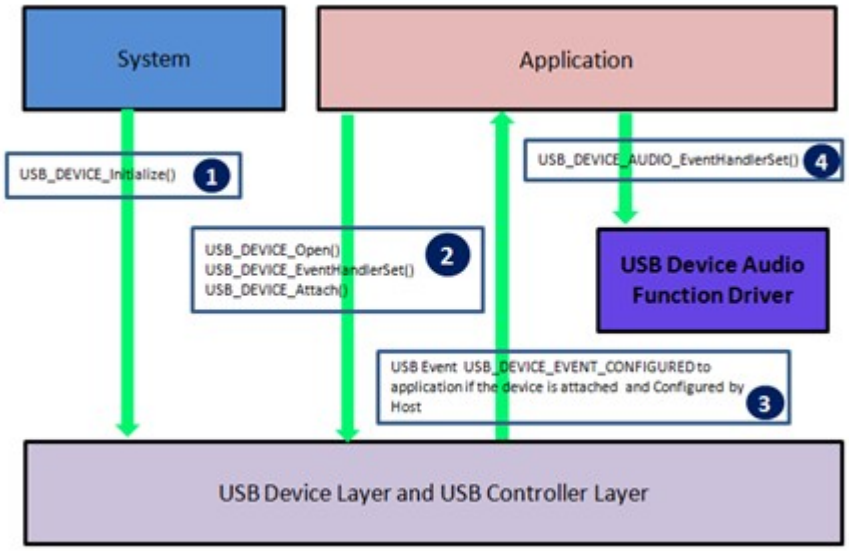
USB音频函数驱动程序数据初始化

```
/*************************************************
 * USB设备函数驱动程序初始化数据
 *************************************************/
    const USB_DEVICE_AUDIO_INIT audioInit0 =
    {
        .queueSizeRead = 64,
        .queueSizeWrite = 8
    };
```

```
/*************************************************
 * USB设备层函数驱动程序注册
 * 表
 *************************************************/
const USB_DEVICE_FUNCTION_REGISTRATION_TABLE funcRegistrationTable[1] =
{
    /* 函数1 */
    {
        .configurationValue = 1,     /* 配置值*/
        .interfaceNumber = 0,        /* 此函数的第一个接口编号
*/
        .speed = USB_SPEED_FULL,     /* 速度函数 */
        .numberOfInterfaces = 3,     /* 接口数量 */
        .funcDriverIndex = 0,        /* 音频函数驱动程序的索引 */
        .driver = (void*)USB_DEVICE_AUDIO_FUNCTION_DRIVER,   /* 暴露给设备
层的USB音频函数数据 */
        .funcDriverInit = (void*)&audioInit0,    /* 函数驱动程序初始化数据*/
    },
};
```

下图所示为使用音频函数驱动程序时应用程序遵循的典型序列。

图5-16. USB音频设备执行序列



- 系统通过调用USB_DEVICE_Initialize函数来初始化USB设备层。
- 设备层允许应用程序通过注册回调函数来接收连接、供电、配置等USB设备事件。应用程序使用USB_DEVICE_EVENT_CONFIGURED事件来继续操作。
- 设备层配置完成后，应用程序将通过音频函数驱动程序注册一个回调函数，以接收音频控制传输和其他音频函数驱动程序事件。此时，应用程序可使用音频函数驱动程序API来与USB主机进行通信。

system_interrupt.c文件包含USB设备协议栈的中断处理程序。USB驱动程序任务处理函数通过USB ISR进行调用，如下图所示。

图5-17. USB中断处理程序

```
// *****************************************************************
// *****************************************************************
// Section: System Interrupt Vector Functions
// *****************************************************************
// *****************************************************************

void __ISR(_USB_1_VECTOR, ipl2AUTO) _IntHandlerUSBInstance0(void)
{
    DRV_USBFS_Tasks_ISR(sysObj.drvUSBObject);
}
```

system_tasks.c文件包含SYS_Tasks程序，该程序为USB函数驱动程序和设备层运行状态机任务程序，如下图所示。

图5-18. 系统调用USB驱动程序和设备任务

```
void SYS_Tasks ( void )
{
    /* Maintain system services */

    /* Maintain Device Drivers */

    /* Maintain Middleware & Other Libraries */

    /* USB FS Driver Task Routine */
    DRV_USBFS_Tasks(sysObj.drvUSBObject);

    /* USB Device layer tasks routine */
    USB_DEVICE_Tasks(sysObj.usbDevObject0);

    /* Maintain the application's state machine. */
    APP_Tasks();
}
```

5.5.4 注册音频函数驱动程序回调函数

要创建一个USB耳麦应用，应用程序需通过音频函数驱动程序注册事件处理程序：

- 从主机接收音频控制请求，例如音量控制和静音控制等。
- 通过USB音频设备驱动程序处理其他事件（即数据写入完成或数据读取完成）

事件处理程序将在USB设备层应答来自USB主机的SET CONFIGURATION请求之前进行注册。为此，需在设备层生成的USB_DEVICE_EVENT_CONFIGURED事件中设置回调函数。

图5-19. 用于注册音频事件处理程序的USB设备事件处理程序

```
void APP_USBDeviceEventHandler(USB_DEVICE_EVENT event, void * pEventData, uintptr_t context )
{
    volatile USB_DEVICE_EVENT_DATA_CONFIGURED* configuredEventData;
    switch( event )
    {
        case USB_DEVICE_EVENT_RESET:
            /* ..TBD Application Code.. */
            break;
        case USB_DEVICE_EVENT_DECONFIGURED:
            /* ..TBD Application Code.. */
            break;
        case USB_DEVICE_EVENT_CONFIGURED:
            /* check the configuration */
            configuredEventData =
                    (USB_DEVICE_EVENT_DATA_CONFIGURED *)pEventData;
            if(configuredEventData->configurationValue == 1)
            {
                /* ..TBD Application Code.. */
                USB_DEVICE_AUDIO_EventHandlerSet(0,
                        APP_USBDeviceAudioEventHandler ,
                        (uintptr_t)NULL);
                /* Mark that set configuration is complete */
                appData.isConfigured = true;
            }
            break;
        case USB_DEVICE_EVENT_SUSPENDED:
            /* ..TBD Application Code.. */
            break;

        case USB_DEVICE_EVENT_RESUMED:
        case USB_DEVICE_EVENT_POWER_DETECTED:
            USB_DEVICE_Attach (appData.usbDevHandle);
            break;

        case USB_DEVICE_EVENT_POWER_REMOVED:
            USB_DEVICE_Detach (appData.usbDevHandle);
        case USB_DEVICE_EVENT_ERROR:
        default:
            break;
    }
}
```

注：

上图简要说明了事件处理程序代码，并不代表USB耳麦应用下实现的完整USB设备事件处理程序。请参见USB耳麦应用下的app.c文件来查看源代码。下面的*“USB耳麦应用解决方案”*部分给出了对源代码的引用。

- USB音频设备驱动程序提供相应函数来发送和接收数据。
- USB_DEVICE_AUDIO_Read函数用于调度数据读取操作。当主机向设备传输数据时，音频函数驱动程序将接收数据并调用USB_DEVICE_AUDIO_EVENT_READ_COMPLETE事件。该事件表示音频数据已位于指定的应用程序缓冲区中。音频数据将传输到编解码器驱动程序中以供播放，USB读取缓冲区队列随后可接收下一个音频数据。
- USB_DEVICE_AUDIO_Write函数用于调度数据写入操作。当主机发送数据请求时，音频函数驱动程序将传输数据并调用USB_DEVICE_AUDIO_EVENT_WRITE_COMPLETE事件。事件处理程序将通知主机写入操作已完成，并指示可安排向主机写入下一个数据。

以下示例给出了USB_DEVICE_AUDIO_EVENT_READ_COMPLETE和USB_DEVICE_AUDIO_EVENT_WRITE_COMPLETE事件的事件处理程序代码。

USB设备音频事件处理程序代码——写/读

```
case USB_DEVICE_AUDIO_EVENT_READ_COMPLETE:
{
    readEventData = (USB_DEVICE_AUDIO_EVENT_DATA_READ_COMPLETE *)pData;

    _APP_SetUSBReadBufferReady(readEventData->handle);
    appPlaybackBuffer.usbReadCompleteBufferLevel++;
    queueEmpty = false;

    if(appData.state == APP_SUBMIT_INITIAL_CODEC_WRITE_REQUEST)
    {
    if(_APP_USBReadAllBufferReady())
    {
        usbReadCompleteFlag = true;
    }
    }
}
    break;

case USB_DEVICE_AUDIO_EVENT_WRITE_COMPLETE:
{
    writeEventData=(USB_DEVICE_AUDIO_EVENT_DATA_WRITE_COMPLETE *)pData;

    if (writeEventData->handle == appData.writeTransferHandle1)
    {
    appData.isUSBWriteComplete1 = true;
    }
    else if(writeEventData->handle == appData.writeTransferHandle2)
    {
    appData.isUSBWriteComplete2 = true;
    }
    else
    {
    }
}
break;
```

当音频函数驱动程序接收到音频类特定控制传输请求时，它会将该控制传输作为音频函数驱动程序事件传送到应用程序。USB音频设备驱动程序通过实现SET和GET事件来完成音频控制的当前设置。当主机发送设置数据包来设置当前音频控制时，音频函数驱动程序将调用USB_DEVICE_AUDIO_EVENT_CONTROL_SET_CUR事件。事件处理程序标识设置数据包指向接口还是端点。

如果设置数据包的接收方为接口，它将标识控制命令指向的实体。如果标识的实体是三个功能单元之一，则音频控制命令将设置静音控制，驱动程序将调用USB_DEVICE_ControlReceive函数并且可随时接收数据包。当实体ID为混合器单元时，驱动程序将以指示混合器单元未实现任何音频控制的错误状态来进行响应，从而停止请求。

如果设置数据包的接收方为端点，并且音频控制命令将设置采样频率控制，则驱动程序将调用SB_DEVICE_ControlReceive函数并且可随时接收数据包。

以下示例给出了当前音频控制设置事件的音频事件处理程序代码的源代码。

USB设备音频事件处理程序代码——设置当前控制

```
case USB_DEVICE_AUDIO_EVENT_CONTROL_SET_CUR:
{
    if(((USB_SETUP_PACKET*)pData)->Recipient ==
         USB_SETUP_REQUEST_RECIPIENT_INTERFACE)
    {
        entityID = ((USB_AUDIO_CONTROL_INTERFACE_REQUEST*)pData)->entityID;
        if ((entityID == APP_ID_FEATURE_UNIT) ||
        (entityID == APP_ID_FEATURE_UNIT_MICROPHONE) ||
        (entityID == APP_ID_FEATURE_UNIT_SIDE_TONING))
        {
        controlSelector = ((USB_AUDIO_FEATURE_UNIT_CONTROL_REQUEST*)
                    pData)->controlSelector;
           if (controlSelector == USB_AUDIO_MUTE_CONTROL)
           {
        USB_DEVICE_ControlReceive(appData.usbDevHandle,
                        (void *) &(appData.dacMute),
                        1);
        appData.currentAudioControl = APP_USB_AUDIO_MUTE_CONTROL;
           }

        }
        else if (entityID == APP_ID_MIXER_UNIT)
        {
        USB_DEVICE_ControlStatus(appData.usbDevHandle,
                     USB_DEVICE_CONTROL_STATUS_ERROR);
        }
    }
    else if (((USB_SETUP_PACKET*)pData)->Recipient ==
           USB_SETUP_REQUEST_RECIPIENT_ENDPOINT)
    {
        controlSelector =
        ((USB_AUDIO_ENDPOINT_CONTROL_REQUEST*)
        pData)->controlSelector;

        if (controlSelector == USB_AUDIO_SAMPLING_FREQ_CONTROL)
        {
        if (((USB_AUDIO_ENDPOINT_CONTROL_REQUEST*)
        pData)->endpointNumber == MICROPHONE_EP)
        {
           USB_DEVICE_ControlReceive(appData.usbDevHandle,
                 (void *) &(appData.sampleFreqMic),
                 3);
           appData.currentAudioControl =
               APP_USB_AUDIO_SAMPLING_FREQ_CONTROL_MP;
        }

        else if (((USB_AUDIO_ENDPOINT_CONTROL_REQUEST*)
        pData)->endpointNumber == HEADPHONE_EP)
        {
           USB_DEVICE_ControlReceive(appData.usbDevHandle,
                 (void *) &(appData.sampleFreq),
                 3);
           appData.currentAudioControl =
               APP_USB_AUDIO_SAMPLING_FREQ_CONTROL_HP;
        }
        }
    }
}
break;
```

当主机发送数据包时，音频函数驱动程序将调用USB_DEVICE_AUDIO_EVENT_CONTROL_TRANSFER_DATA_RECEIVED事件。驱动程序将应答接收到的数据并以OK状态进行响应，表示实现了音频控制并将处理请求。事件处理程序将识别并处理控制请求，以更改静音控制或者耳机和麦克风的采样频率。

以下示例给出了当前音频控制设置事件数据阶段的音频事件处理程序代码的源代码。

USB设备音频事件处理程序代码——数据阶段——设置当前控制

```
case USB_DEVICE_AUDIO_EVENT_CONTROL_TRANSFER_DATA_RECEIVED:
{

USB_DEVICE_ControlStatus(appData.usbDevHandle,USB_DEVICE_CONTROL_STATUS_OK);

    if (appData.currentAudioControl == APP_USB_AUDIO_MUTE_CONTROL)
    {
        appData.state = APP_MUTE_AUDIO_PLAYBACK;
        appData.currentAudioControl = APP_USB_CONTROL_NONE;
    }
    if (appData.currentAudioControl == APP_USB_AUDIO_SAMPLING_FREQ_CONTROL_HP)
    {
        if (appData.sampleFreq == SAMPLING_RATE_48000)
        {
            appData.codecClientWrite.bufferSize = 192;
            appData.USBReadBufSize = 192;
        }
        else if (appData.sampleFreq == SAMPLING_RATE_32000)
        {
            appData.codecClientWrite.bufferSize = 128;
            appData.USBReadBufSize = 192;
        }
        else if (appData.sampleFreq == SAMPLING_RATE_24000)
        {
            appData.codecClientWrite.bufferSize = 96;
            appData.USBReadBufSize = 192;
        }
        else if (appData.sampleFreq == SAMPLING_RATE_16000)
        {
            appData.codecClientWrite.bufferSize = 64;
            appData.USBReadBufSize = 192;
        }
        appData.state = APP_SAMPLING_FREQUENCY_CHANGE;
        appData.currentAudioControl = APP_USB_CONTROL_NONE;
    }
    else if (appData.currentAudioControl ==
         APP_USB_AUDIO_SAMPLING_FREQ_CONTROL_MP)
    {
        if (appData.sampleFreqMic == SAMPLING_RATE_48000)
        {
            appData.codecClientRead.bufferSize = 192;
        }
        else if (appData.sampleFreqMic == SAMPLING_RATE_32000)
        {
            appData.codecClientRead.bufferSize = 128;
        }
        else if (appData.sampleFreqMic == SAMPLING_RATE_24000)
        {
            appData.codecClientRead.bufferSize = 96;
        }
        else if (appData.sampleFreqMic == SAMPLING_RATE_16000)
        {
             appData.codecClientRead.bufferSize = 64;
        }
        appData.state = APP_MUTE_AUDIO_PLAYBACK;
        appData.state = APP_SAMPLING_FREQUENCY_CHANGE;
        appData.currentAudioControl = APP_USB_CONTROL_NONE;
    }
}
break;
```

当主机发送设置数据包来获取当前音频控制时，音频函数驱动程序将调用USB_DEVICE_AUDIO_EVENT_CONTROL_GET_CUR事件。事件处理程序标识设置数据包指向接口还是端点。

如果设置数据包的接收方为接口，它将标识控制命令指向的实体。如果标识的实体是三个功能单元之一，则音频控制命令将获取静音控制，驱动程序将调用USB_DEVICE_ControlSend函数并发送静音控制状态。当实体ID为混合器单元时，驱动程序将以指示混合器单元未实现任何音频控制的错误状态来进行响应，从而停止请求。

如果设置数据包的接收方为端点，并且音频控制命令将获取采样频率控制，则驱动程序将调用USB_DEVICE_ControlSend函数并且会将当前采样频率值发送给主机。

以下示例给出了当前音频控制获取事件的音频事件处理程序代码的源代码。

设备音频事件处理程序代码——获取当前控制

```
case USB_DEVICE_AUDIO_EVENT_CONTROL_GET_CUR:
{
    if (((USB_SETUP_PACKET*)pData)->Recipient ==
         USB_SETUP_REQUEST_RECIPIENT_INTERFACE)
    {
        entityID = ((USB_AUDIO_CONTROL_INTERFACE_REQUEST*) pData)->entityID;
        if ((entityID == APP_ID_FEATURE_UNIT) ||
         (entityID == APP_ID_FEATURE_UNIT_MICROPHONE) ||
         (entityID == APP_ID_FEATURE_UNIT_SIDE_TONING))
        {
           controlSelector = ((USB_AUDIO_FEATURE_UNIT_CONTROL_REQUEST*)
                    pData)->controlSelector;
           if (controlSelector == USB_AUDIO_MUTE_CONTROL)
           {
                USB_DEVICE_ControlSend(appData.usbDevHandle,
                     (void *)&(appData.dacMute),
                     1);
           }
        }

        else if (entityID == APP_ID_MIXER_UNIT)
        {
            USB_DEVICE_ControlStatus (appData.usbDevHandle,
                    USB_DEVICE_CONTROL_STATUS_ERROR);
        }
    }
    else if (((USB_SETUP_PACKET*)pData)->Recipient ==
           USB_SETUP_REQUEST_RECIPIENT_ENDPOINT)
    {
        controlSelector = ((USB_AUDIO_ENDPOINT_CONTROL_REQUEST*)
                           pData)->controlSelector;
        if (controlSelector == USB_AUDIO_SAMPLING_FREQ_CONTROL)
        {
            if (((USB_AUDIO_ENDPOINT_CONTROL_REQUEST*)
                pData)->endpointNumber == MICROPHONE_EP)
            {
               USB_DEVICE_ControlSend(appData.usbDevHandle,
                          (void *)&(appData.sampleFreqMic),
                          3);
            }

            else if (((USB_AUDIO_ENDPOINT_CONTROL_REQUEST*)
                      pData)->endpointNumber == HEADPHONE_EP)
            {
               USB_DEVICE_ControlSend(appData.usbDevHandle,
                        (void *)&(appData.sampleFreq), 3 );
            }
        }
    }
}
break;
```

注：
前几页上的图片对音频事件处理程序代码进行了说明，并不代表USB耳麦应用下实现的完整USB设备音频事件处理程序。请参见USB耳麦应用下的app.c文件来查看完整源代码。下面的*“USB耳麦应用解决方案”*部分给出了对源代码的引用。

5.6 USB耳麦应用解决方案

MPLAB Harmony随附预先开发的*usb_headset*应用程序，用于演示USB耳麦功能。MPLAB Harmony中关于此应用程序演示的简要说明和位置如下：

名称：usb_headset

说明：在此应用演示中，基于PIC32单片机的系统连接到USB主机（例如个人计算机），该主机可满足USB音频设备类耳麦的要求。系统枚举USB音频设备端点，这使得系统能够通过USB端口同时发送播放音频和接收录音音频。

项目名称：usb_headset.X

位置：<install-dir>/apps/audio/usb_headset/firmware

注：
USB耳麦应用演示未实现符合HID标准的消费类音频控制。

5.7 应用注意事项

5.7.1 USB和编解码器时钟域不匹配问题

在USB音频系统中，由于存在多个时钟（USB域时钟、采样率、服务和编解码器域时钟），可能会出现音质问题。无论何时出现，都需要解决时钟不匹配的问题，以防止出现多余噪声。要解决时钟不匹配问题，USB音频应用程序应遵循以下准则。

需要选择和实施适当的同步机制。机制的选择取决于应用需求、软件能力、硬件能力和可用性。例如，当硬件允许编解码器时钟源与USB SOF信号同步时，同步型实现（在USB音频同步中讨论）是理想选择。

发送方和接收方的同步方法需要兼容。下表列出了已知可行的音频发送方和接收方的同步组合。

表5-30. USB音频同步组合

发送方	接收方	USB时钟来自
同步	同步	SOF
自适应	异步	采样率
异步	异步	自由运行的内部时钟

以下各节是实现USB音频设备应用的重要注意事项。

5.7.2 音频数据缓冲

USB音频系统引入了与实时系统相关的典型生成方-消耗方问题。
适用于USB音频系统的时序约束如下。

- 延时——在提交音频采样之前处理和输入音频采样所需的时间
- 采样周期——相邻两次采样之间的时间
- 处理时间——下一个输入采样到达前处理输入采样和生成输出所需的时间

USB音频设备需要通过实现缓冲机制来应对面临的时序挑战。以下是实现USB音频数据缓冲的两种可行方法。

乒乓缓冲区方法：乒乓方法使用两个缓冲区。一个缓冲区从USB端点中读取数据，而另一个缓冲区将数据提交给编解码器以供播放。当向编解码器提交的操作完成后，两个缓冲区的角色将进行切换。角色切换是通过修改指向缓冲区的指针来完成的。

乒乓缓冲区实现方案在大多数情况下都能正常工作，但有些情况下无法按预期正常工作：

- 交换的数据量非常大。大数据量需要更多处理时间来准备下一组数据。这可能会导致传输完成以及下一个要调度的数据未就绪的情况发生
- 数据生成方和数据消耗方的时钟源不同
- 数据速率有所不同
- 数据包大小有所不同

缓冲区排队方法：缓冲区排队方法克服了乒乓缓冲的缺点。在这种方法中，应用程序将维护存储器缓冲区池。这些缓冲区存储接收的音频数据。一旦达到设定的缓冲级别，应用程序就会以先入先出的方式开始向音频设备上的编解码器提交读取的数据。读取USB端点与写入编解码器通过数据缓冲区的循环排队实现同时进行。

USB音频函数驱动程序和编解码器驱动程序为应用程序提供支持以实现缓冲区排队机制。
USB_DEVICE_AUDIO_Read/USB_DEVICE_AUDIO_Write和
DRV_CODEC_BufferAddWrite/DRV_CODEC_BufferAddRead函数被应用程序用来实施缓冲机制。

这些API实现的是异步缓冲区排队数据传输模型。这些API本质上是非阻塞的，并且可以通过多次调用来对音频数据进行排队。因此，客户不需要等待正在进行的传输完成即可提交新的数字音频缓冲区。

为了支持缓冲区排队方法，驱动程序（USB和编解码器）需以数组形式维护一个公共缓冲区对象池，其大小由配置宏确定。当需要缓冲区对象时，驱动程序逻辑搜索数组，查找未分配的缓冲区对象。配置宏的值会影响该驱动程序分配的静态RAM空间。

对于USB，配置常量为USB_DEVICE_AUDIO_QUEUE_DEPTH_COMBINED，而对于编解码器，它是关联的I^2S驱动程序实例DRV_I2S_QUEUE_DEPTH_COMBINED的最大队列大小。当在USB和I^2S的MHC驱动程序配置下配置队列大小时，将自动创建这些宏。

注：
乒乓实现中的这些宏的大小为2。对于缓冲区排队实现，该值将根据需要进行适当配置。

5.7.3 水印等级和编解码器时钟调整

在USB音频应用中，由于生成方和消耗方以不同的时钟速率运行，因此数据同步问题会加剧。在这种情况下，应用程序需要与生成方/消耗方交互，并指示另一方加快或减慢速度。

其中一种实现同步的方法是通过维护和管理排队采样的等级为相应同步点提供反馈。例如，在采用异步端点实现方法的USB音频设备的实现过程中，应用程序将维护要发送到编解码器器件的数据缓冲区对象的数量。然后，应用程序将通过反馈端点向USB主机发送反馈报文以实现同步（见上文中USB音频同步中讨论的“异步同步”）。

编解码器驱动程序通过提供API DRV_CODEC_BufferCombinedQueueSizeGet（可扩展到I^2S API DRV_I2S_BufferCombinedQueueSizeGet）协助应用程序维护其缓冲的采样。该API将返回队列中当前的数据缓冲区对象的数量。

USB音频应用中观察到的时钟域不匹配问题也可以通过调整（减慢或加快）编解码器主时钟（MCLK）来解决。通常，应用程序会维持在定义的周期内接收到的样本平均数。它会比较样本平均数与上限和下限水印等级。当接收到的样本平均数超出可接受的范围时，应用程序通过略微提高或降低编解码器MCLK来对其进行调整。调节编解码器时钟可防止缓冲区下溢和上溢，从而减少音频流中的多余声音（见上文中USB音频同步部分讨论的“自适应同步”）。

DRV_CODEC_SamplingRateSet编解码器驱动程序API用于设置所需采样率以加快或减慢相关编解码器时钟。

6. 结论

开发USB音频设备应用程序需要了解各种标准、协议和中间件。用户可以凭借MPLAB Harmony USB设备协议栈固件框架、附带的音频固件和编解码器驱动程序来设计解决方案，而无需担心基础标准或协议。MPLAB Harmony为用户提供了一款用于PIC32单片机的灵活、紧凑的全集成固件开发平台。本应用笔记介绍了由音频子系统和USB子系统组成的USB耳麦应用的功能模型，概述了与音频和USB子系统有关的音频通信的基本概念，还介绍了用户如何使用MPLAB Harmony配置器（MHC）实用程序来创建和配置USB耳麦应用。此外，它还涵盖了与实施USB音频应用程序相关的特定主题需注意的事项。

MPLAB Harmony集成软件框架包含了各种音频解决方案的更多示例和演示，可从Microchip网站下载（见资源部分）。

7. 参考资料

通用串行总线规范2.0

http://www.usb.org/developers/docs/usb20_docs/usb_20_033017.zip

音频设备的USB设备类定义

http://www.usb.org/developers/docs/devclass_docs/audio10.pdf

基本音频设备的USB音频设备类规范

http://www.usb.org/developers/docs/devclass_docs/BasicAudioDevice-10.zip

人机接口设备（HID）的USB设备类定义

http://www.usb.org/developers/hidpage/HID1_11.pdf

AN1422——采用PIC32的高质量音频应用

http://ww1.microchip.com/downloads/en/AppNotes/01422A.pdf

Atmel AVR32716：AVR UC3 USB音频类

http://www.atmel.com/Images/doc32139.pdf

AT91 USB HID驱动程序实现

http://www.atmel.com/Images/doc6273.pdf

MPLAB Harmony软件框架安装程序随附的“help_harmony.pdf”

http://www.microchip.com/mplab/mplab-harmony

http://www.microchip.com/mplab/mplab-harmony

http://microchip.wikidot.com/harmony:overview

MPLAB Harmony配置器（MHC）概述

http://microchip.wikidot.com/harmony:mhc-overview

MPLAB® Harmony图形设计器（MHGC）概述

http://microchip.wikidot.com/harmony:mhc-mhgc-overview

8. 版本历史

版本A（2017年11月）

本文档的初始版本。

Microchip网站

Microchip网站（http://www.microchip.com/）为客户提供在线支持。客户可通过该网站方便地获取文件和信息。只要使用常用的互联网浏览器即可访问，网站提供以下信息：

- **产品支持**——数据手册和勘误表、应用笔记和示例程序、设计资源、用户指南以及硬件支持文档、最新的软件版本以及归档软件
- **一般技术支持**——常见问题（FAQ）、技术支持请求、在线讨论组以及Microchip顾问计划成员名单
- **Microchip业务**——产品选型和订购指南、最新Microchip新闻稿、研讨会和活动安排表、Microchip销售办事处、代理商以及工厂代表列表

变更通知客户服务

Microchip的变更通知客户服务有助于客户了解Microchip产品的最新信息。注册客户可在他们感兴趣的某个产品系列或开发工具发生变更、更新、发布新版本或勘误表时，收到电子邮件通知。

欲注册，请登录Microchip网站http://www.microchip.com/。在“支持”（Support）下，点击“变更通知客户”（Customer Change Notification）服务后按照注册说明完成注册。

客户支持

Microchip产品的用户可通过以下渠道获得帮助：

- 代理商或代表
- 当地销售办事处
- 应用工程师（FAE）
- 技术支持

客户应联系其代理商、代表或应用工程师（FAE）寻求支持。当地销售办事处也可为客户提供帮助。本文档后附有销售办事处的联系方式。

也可通过以下网站获得技术支持：http://www.microchip.com/support

Microchip器件代码保护功能

请注意以下有关Microchip器件代码保护功能的要点：

- Microchip的产品均达到Microchip数据手册中所述的技术指标。
- Microchip确信：在正常使用的情况下，Microchip系列产品是当今市场上同类产品中最安全的产品之一。
- 目前，仍存在着恶意、甚至是非法破坏代码保护功能的行为。就我们所知，所有这些行为都不是以Microchip数据手册中规定的操作规范来使用Microchip产品的。这样做的人极可能侵犯了知识产权。
- Microchip愿意与关心代码完整性的客户合作。

- Microchip或任何其他半导体厂商均无法保证其代码的安全性。代码保护并不意味着我们保证产品是“牢不可破”的。

代码保护功能处于持续发展中。Microchip承诺将不断改进产品的代码保护功能。任何试图破坏Microchip代码保护功能的行为均可视为违反了《数字器件千年版权法案（Digital Millennium Copyright Act）》。如果这种行为导致他人在未经授权的情况下，能访问您的软件或其他受版权保护的成果，您有权依据该法案提起诉讼，从而制止这种行为。

法律声明

本出版物中所述的器件应用信息及其他类似内容仅为您提供便利，它们可能由更新之信息所替代。确保应用符合技术规范，是您自身应负的责任。Microchip对这些信息不作任何明示或暗示、书面或口头、法定或其他形式的声明或担保，包括但不限于针对其使用情况、质量、性能、适销性或特定用途的适用性的声明或担保。

Microchip对因这些信息及使用这些信息而引起的后果不承担任何责任。如果将Microchip器件用于生命维持和/或生命安全应用，一切风险由买方自负。买方同意在由此引发任何一切伤害、索赔、诉讼或费用时，会维护和保障Microchip免于承担法律责任，并加以赔偿。除非另外声明，否则在Microchip知识产权保护下，不得暗中或以其他方式转让任何许可证。

商标

AVR、AVR徽标、AVR Freaks、BitCloud、chipKIT、chipKIT徽标、CryptoMemory、CryptoRF、dsPIC、FlashFlex、flexPWR、Heldo、JukeBlox、KeeLoq、Kleer、LANCheck、LINK MD、maXStylus、maXTouch、MediaLB、megaAVR、MOST、MOST徽标、MPLAB、OptoLyzer、PIC、picoPower、PICSTART、PIC32徽标、Prochip Designer、QTouch、SAM-BA、SpyNIC、SST、SST徽标、SuperFlash、tinyAVR、UNI/O及XMEGA均为Microchip Technology Inc.在美国和其他国家或地区的注册商标。

ClockWorks、The Embedded Control Solutions Company、EtherSynch、Hyper Speed Control、HyperLight Load、IntelliMOS、mTouch、Precision Edge和Quiet-Wire均为Microchip Technology Inc.在美国的注册商标。

Adjacent Key Suppression、AKS、Analog-for-the-Digital Age、Any Capacitor、AnyIn、AnyOut、BodyCom、CodeGuard、CryptoAuthentication、CryptoAutomotive、CryptoCompanion、CryptoController、dsPICDEM、dsPICDEM.net、Dynamic Average Matching、DAM、ECAN、EtherGREEN、In-Circuit Serial Programming、ICSP、INICnet、Inter-Chip Connectivity、JitterBlocker、KleerNet、KleerNet徽标、memBrain、Mindi、MiWi、motorBench、MPASM、MPF、MPLAB Certified 徽标、MPLIB、MPLINK、MultiTRAK、NetDetach、Omniscient Code Generation、PICDEM、PICDEM.net、PICkit、PICtail、PowerSmart、PureSilicon、QMatrix、REAL ICE、Ripple Blocker、SAM-ICE、Serial Quad I/O、SMART-I.S.、SQI、SuperSwitcher、SuperSwitcher II、Total Endurance、TSHARC、USBCheck、VariSense、ViewSpan、WiperLock、Wireless DNA和ZENA均为Microchip Technology Inc.在美国和其他国家或地区的商标。

SQTP为Microchip Technology Inc.在美国的服务标记。

Silicon Storage Technology为Microchip Technology Inc.在除美国外的国家或地区的注册商标。

GestIC是Microchip Technology Inc.的子公司Microchip Technology Germany II GmbH & Co. KG在除美国外的国家或地区的注册商标。

在此提及的所有其他商标均为各持有公司所有。

ISBN：978-1-5224-3145-9

DNV认证的质量管理体系

ISO/TS 16949

Microchip位于美国亚利桑那州Chandler和Tempe与位于俄勒冈州Gresham的全球总部、设计和晶圆生产厂及位于美国加利福尼亚州和印度的设计中心均通过了ISO/TS-16949:2009认证。Microchip的PIC® MCU和dsPIC® DSC、KEELOQ®跳码器件、串行EEPROM、单片机外设、非易失性存储器和模拟产品严格遵守公司的质量体系流程。此外，Microchip在开发系统的设计和生产方面的质量体系也已通过了ISO 9001:2000认证。

10/25/17

全球销售及服务网点

美洲

公司总部 Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 1-480-792-7200
Fax: 1-480-792-7277
技术支持：
http://www.microchip.com/support
网址：www.microchip.com

亚特兰大 Atlanta
Duluth, GA
Tel: 1-678-957-9614
Fax: 1-678-957-1455

奥斯汀 Austin, TX
Tel: 1-512-257-3370

波士顿 Boston
Westborough, MA
Tel: 1-774-760-0087
Fax: 1-774-760-0088

芝加哥 Chicago
Itasca, IL
Tel: 1-630-285-0071
Fax: 1-630-285-0075

达拉斯 Dallas
Addison, TX
Tel: 1-972-818-7423
Fax: 1-972-818-2924

底特律 Detroit
Novi, MI
Tel: 1-248-848-4000

休斯敦 Houston, TX
Tel: 1-281-894-5983

印第安纳波利斯 Indianapolis
Noblesville, IN
Tel: 1-317-773-8323
Fax: 1-317-773-5453
Tel: 1-317-536-2380

洛杉矶 Los Angeles
Mission Viejo, CA
Tel: 1-949-462-9523
Fax: 1-949-462-9608
Tel: 1-951-273-7800

罗利 Raleigh, NC
Tel: 1-919-844-7510

纽约 New York, NY
Tel: 1-631-435-6000

圣何塞 San Jose, CA
Tel: 1-408-735-9110
Tel: 1-408-436-4270

加拿大多伦多 Toronto
Tel: 1-905-695-1980
Fax: 1-905-695-2078

亚太地区

中国 - 北京
Tel: 86-10-8569-7000

中国 - 成都
Tel: 86-28-8665-5511

中国 - 重庆
Tel: 86-23-8980-9588

中国 - 东莞
Tel: 86-769-8702-9880

中国 - 广州
Tel: 86-20-8755-8029

中国 - 杭州
Tel: 86-571-8792-8115

中国 - 南京
Tel: 86-25-8473-2460

中国 - 青岛
Tel: 86-532-8502-7355

中国 - 上海
Tel: 86-21-3326-8000

中国 - 沈阳
Tel: 86-24-2334-2829

中国 - 深圳
Tel: 86-755-8864-2200

中国 - 苏州
Tel: 86-186-6233-1526

中国 - 武汉
Tel: 86-27-5980-5300

中国 - 西安
Tel: 86-29-8833-7252

中国 - 厦门
Tel: 86-592-238-8138

中国 - 香港特别行政区
Tel: 852-2943-5100

中国 - 珠海
Tel: 86-756-321-0040

台湾地区 - 高雄
Tel: 886-7-213-7830

台湾地区 - 台北
Tel: 886-2-2508-8600

台湾地区 - 新竹
Tel: 886-3-577-8366

亚太地区

澳大利亚 Australia - Sydney
Tel: 61-2-9868-6733

印度 India - Bangalore
Tel: 91-80-3090-4444

印度 India - New Delhi
Tel: 91-11-4160-8631

印度 India - Pune
Tel: 91-20-4121-0141

日本 Japan - Osaka
Tel: 81-6-6152-7160

日本 Japan - Tokyo
Tel: 81-3-6880-3770

韩国 Korea - Daegu
Tel: 82-53-744-4301

韩国 Korea - Seoul
Tel: 82-2-554-7200

马来西亚 Malaysia - Kuala Lumpur
Tel: 60-3-7651-7906

马来西亚 Malaysia - Penang
Tel: 60-4-227-8870

菲律宾 Philippines - Manila
Tel: 63-2-634-9065

新加坡 Singapore
Tel: 65-6334-8870

泰国 Thailand - Bangkok
Tel: 66-2-694-1351

越南 Vietnam - Ho Chi Minh
Tel: 84-28-5448-2100

欧洲

奥地利 Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

丹麦 Denmark - Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

芬兰 Finland - Espoo
Tel: 358-9-4520-820

法国 France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

德国 Germany - Garching
Tel: 49-8931-9700

德国 Germany - Haan
Tel: 49-2129-3766400

德国 Germany - Heilbronn
Tel: 49-7131-67-3636

德国 Germany - Karlsruhe
Tel: 49-721-625370

德国 Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

德国 Germany - Rosenheim
Tel: 49-8031-354-560

以色列 Israel - Ra'anana
Tel: 972-9-744-7705

意大利 Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

意大利 Italy - Padova
Tel: 39-049-7625286

荷兰 Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

挪威 Norway - Trondheim
Tel: 47-7289-7561

波兰 Poland - Warsaw
Tel: 48-22-3325737

罗马尼亚 Romania - Bucharest
Tel: 40-21-407-87-50

西班牙 Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

瑞典 Sweden - Gothenberg
Tel: 46-31-704-60-40

瑞典 Sweden - Stockholm
Tel: 46-8-5090-4654

英国 UK - Wokingham
Tel: 44-118-921-5800
Fax: 44-118-921-5820