

1 Introduction

For MCUs with USB device controllers, the NXP SDK (Software Development Kit) generally provides a USB audio example. This example only supports a single audio format. For example, LPC55S69-EVK, this board has a WM8904 codec. A USB audio example (`usb_device_composite_hid_audio_unified_bm`) is also provided in the SDK. Now, this example supports the 48 K/16 bit audio format. However, in the actual application of the customer, it is sometimes necessary to support multiple audio formats. This application note takes LPC55S69 as an example to describe how to make the USB audio speaker support 48 K/16 bit and 48 K/32 bit audio formats based on the USB audio example of LPC55S69 **SDK v2.10**. The core idea is to add an alternate interface setting that supports the 48 K/32 bit format to the audio speaker in the USB configuration descriptor. When a USB device receives a set interface request, it reconfigures I2S, DMA, and codec according to the value of `bAlternateSetting`. In this application note, the Full Speed (FS) USB interface is used, the USB Audio Class (UAC) version is 1.0, and USB audio works in synchronous mode.

2 USB audio stream interface descriptor

The USB audio function can be regarded as a closed system, opening a predefined interface to the outside world. Each audio function must include an Audio Control (AC) interface, zero or more Audio Stream (AS) interfaces, and zero or more MIDI Stream (MS) interfaces. The audio control interface is used to access all audio controls in the audio function. The audio stream interface is used to transfer the audio data stream into or out of the audio function and the MIDI stream interface is used to transfer MIDI data stream into or out of the audio function. This example uses two audio streaming interfaces: the recorder interface and the speaker interface.

Audio stream interface descriptors include standard interface descriptor, audio class-specific interface descriptor, audio stream endpoint descriptor, and so on. The structure diagram of the audio speaker interface in the USB audio example in the SDK package is shown in [Figure 1](#).

Contents

1	Introduction.....	1
2	USB audio stream interface descriptor.....	1
3	Add the 48 K/32 bit audio format.....	4
4	Test.....	12
5	Conclusion.....	13
6	References.....	13
7	Revision history	13



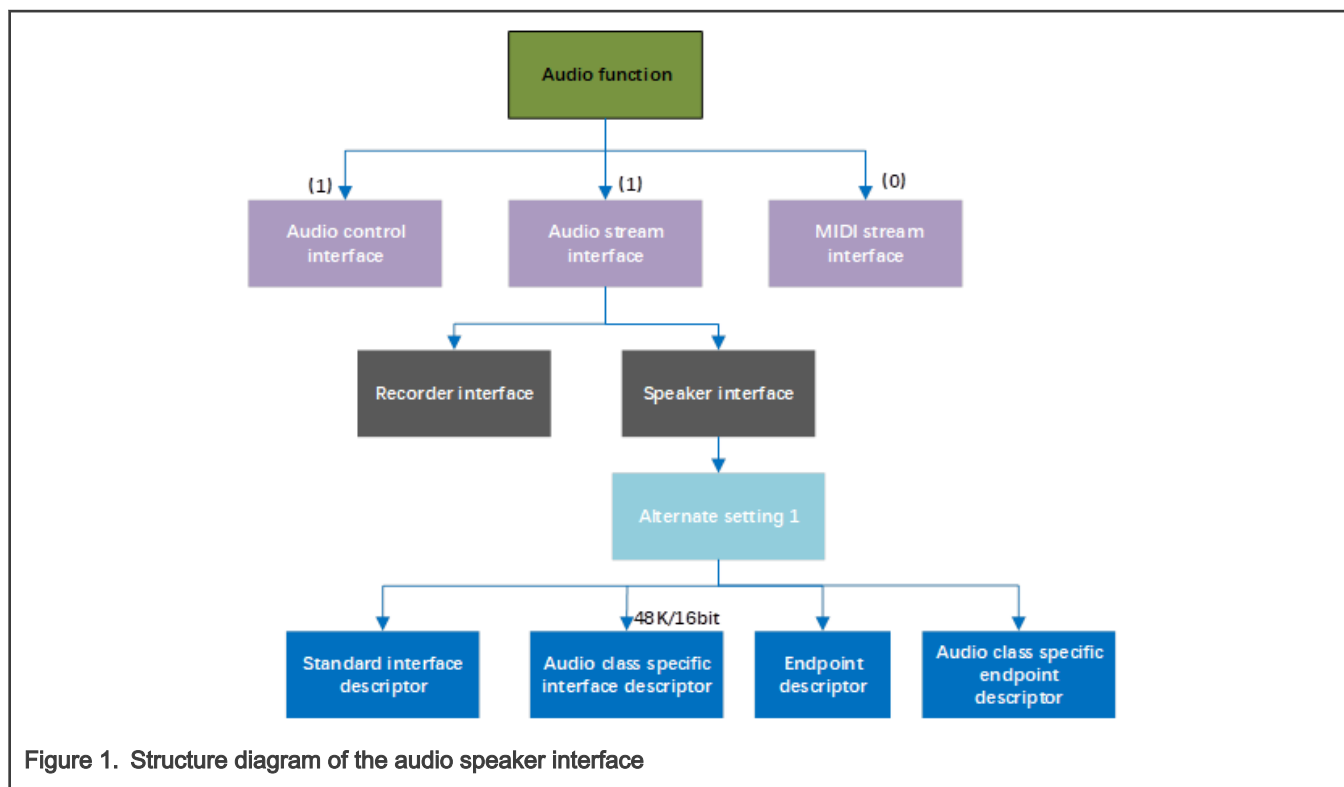


Figure 1. Structure diagram of the audio speaker interface

The audio speaker interface descriptor captured by the USB analyzer is shown in [Figure 2](#), the audio format supported by this configuration is 48 K/16 bit.

INTERFACE Descriptor (9 bytes)					
Field	Length (bits)	Offset (bits)	Decoded	Hex Value	Description
bLength	8	1280	0x09	0x09	Descriptor size is 9 bytes
bDescriptorType	8	1288	0x04	0x04	INTERFACE Descriptor Type
bInterfaceNumber	8	1296	0x02	0x02	The number of this interface is 2
bAlternateSetting	8	1304	0x01	0x01	The value used to select the alternate setting for this interface is 1
bNumEndpoints	8	1312	0x01	0x01	The number of endpoints used by this interface is 1 (excluding endpoint zero)
bInterfaceClass	8	1320	0x01	0x01	The interface implements the Audio Interface class
bInterfaceSubClass	8	1328	0x02	0x02	The interface implements the AUDIOSTREAMING Subclass
bInterfaceProtocol	8	1336	0x00	0x00	The interface uses the IP_VERSION_01_00 Protocol
iInterface	8	1344	0x00	0x00	The device doesn't have a string descriptor describing this interface

Audio Class Specific INTERFACE Descriptor (7 bytes)					
Field	Length (bits)	Offset (bits)	Decoded	Hex Value	Description
bLength	8	1352	0x07	0x07	Size of the descriptor, in bytes
bDescriptorType	8	1360	0x24	0x24	CS_INTERFACE Descriptor Type
bDescriptorSubType	8	1368	0x01	0x01	AS_GENERAL descriptor subtype
bTerminalLink	8	1376	0x04	0x04	The Terminal ID of the Terminal to which the endpoint of this interface is connected
bDelay	8	1384	0x01	0x01	Delay introduced by the data path. Expressed in number of frames.
wFormatTag	16	1392	0x0001	0x0001	PCM

Audio Class Specific INTERFACE Descriptor (11 bytes)					
Field	Length (bits)	Offset (bits)	Decoded	Hex Value	Description
bLength	8	1408	0x0B	0x0B	Size of the descriptor, in bytes
bDescriptorType	8	1416	0x24	0x24	CS_INTERFACE Descriptor Type
bDescriptorSubType	8	1424	0x02	0x02	FORMAT_TYPE descriptor subtype
bFormatType	8	1432	0x01	0x01	FORMAT_TYPE_I
bNrChannels	8	1440	0x02	0x02	Indicates the number of physical channels in the audio data stream
bSubframeSize	8	1448	0x02	0x02	The number of bytes occupied by one audio subframe. Can be 1, 2, 3 or 4.
bBitResolution	8	1456	0x10	0x10	The number of effectively used bits from the available bits in an audio subframe.
bSamFreqType	8	1464	0x01	0x01	Indicates how the sampling frequency can be programmed:
tSamFreq[1]	24	1472	0x00BB80	0x00BB80	Sampling frequency 1 in Hz for this isochronous data endpoint.

ENDPOINT Descriptor (9 bytes)					
Field	Length (bits)	Offset (bits)	Decoded	Hex Value	Description
bLength	8	1496	0x09	0x09	Descriptor size is 9 bytes
bDescriptorType	8	1504	0x05	0x05	ENDPOINT Descriptor Type
bEndpointAddress	8	1512	0x02	0x02	This is an OUT endpoint with endpoint number 2
bmAttributes	8	1520	0x0D	0x0D	Types - Transfer: ISOCHRONOUS Sync: Sync Usage: Data EP
wMaxPacketSize	16	1528	0x00C0	0x00C0	Maximum packet size for this endpoint is 192 Bytes. If Hi-Speed, 0 additional transactions per frame
bInterval	8	1544	0x01	0x01	The polling interval value is every 1 Frames. If Hi-Speed, 1 uFrames/NAK
bRefresh	8	1552	0x00	0x00	Refresh Rate 2^n ms where n = 0
bSynchAddress	8	1560	0x00	0x00	Synchronization Endpoint (if used) is endpoint 0

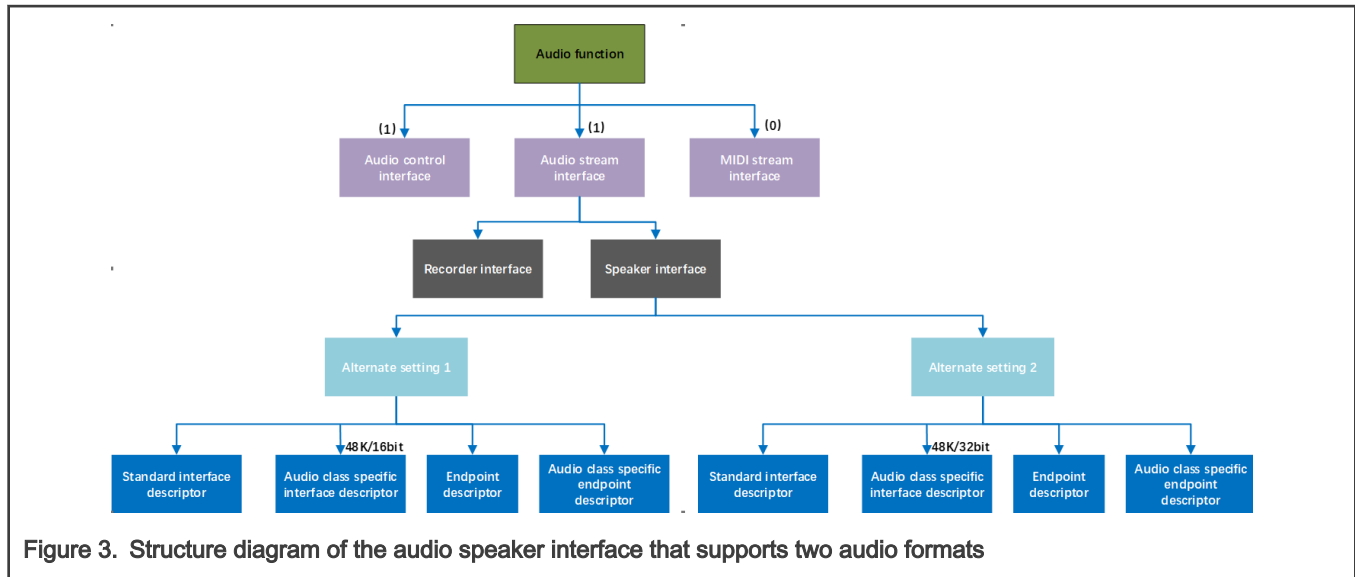
Audio Class Specific ENDPOINT Descriptor (7 bytes)					
Field	Length (bits)	Offset (bits)	Decoded	Hex Value	Description
bLength	8	1568	0x07	0x07	Size of the descriptor, in bytes
bDescriptorType	8	1576	0x25	0x25	CS_ENDPOINT Descriptor Type
bDescriptorSubType	8	1584	0x01	0x01	AUDIO_EP_GENERAL descriptor subtype
bAttributes	8	1592	0x00	0x00	Bit 0: Sampling Frequency 0 Bit 1: Pitch 0 Bit 7: MaxPacketsOnly 0
bLockDelayUnits	8	1600	0x00	0x00	Indicates the units used for the wLockDelay field: 0: Undefined
wLockDelay	16	1608	0x0000	0x0000	Indicates the time it takes this endpoint to reliably lock its internal clock recovery circuitry. Units used depend on the value of the bLockDelayUnits field.

Figure 2. USB audio speaker interface descriptor

NOTE

If USB audio asynchronous mode is used, an isochronous feedback endpoint is also required in the audio speaker interface descriptor.

If you want to make the audio speaker interface support multiple audio formats, add multiple corresponding format alternate settings for this speaker interface. Figure 3 shows the structure diagram of an audio speaker interface that supports two audio formats.



The next chapter introduces detailed steps of adding the 48 K/32 bit audio format.

3 Add the 48 K/32 bit audio format

This section introduces how to add a new 48 K/32 bit audio format based on the SDK example.

The complete code after the change is published on the NXP website as an attachment to the application note. In the attached code, search for the `USB_AUDIO_FORMAT_SWITCH_ENABLE` macro to find out what changes have been made based on the SDK code.

3.1 Key steps

This section describes the key steps of adding the 48 K/32 bit audio format.

3.1.1 Add a new audio stream interface descriptor for the audio speaker

Add an alternate audio stream interface descriptor for the speaker interface in the original configuration descriptor, and configure the audio format as 48 K/32 bit, as shown in [Figure 4](#).

INTERFACE Descriptor (9 bytes)

Field	Length (bits)	Offset (bits)	Decoded	Hex Value	Description
bLength	8	1280	0x09	0x09	Descriptor size is 9 bytes
bDescriptorType	8	1288	0x04	0x04	INTERFACE Descriptor Type
bInterfaceNumber	8	1296	0x02	0x02	The number of this interface is 2.
bAlternateSetting	8	1304	0x01	0x01	The value used to select the alternate setting for this interface is 1
bNumEndpoints	8	1312	0x01	0x01	The number of endpoints used by this interface is 1 (excluding endpoint zero)
bInterfaceClass	8	1320	0x01	0x01	The interface implements the Audio Interface class
bInterfaceSubClass	8	1328	0x02	0x02	The interface implements the AUDIOSTREAMING Subclass
bInterfaceProtocol	8	1336	0x00	0x00	The interface uses the IP_VERSION_01_00 Protocol
Interface	8	1344	0x00	0x00	The device doesn't have a string descriptor describing this interface

Audio Class Specific INTERFACE Descriptor (7 bytes)

Field	Length (bits)	Offset (bits)	Decoded	Hex Value	Description
bLength	8	1352	0x07	0x07	Size of the descriptor, in bytes
bDescriptorType	8	1360	0x24	0x24	CS_INTERFACE Descriptor Type
bDescriptorSubType	8	1368	0x01	0x01	AS_GENERAL descriptor subtype
bTerminalLink	8	1376	0x04	0x04	The Terminal ID of the Terminal to which the endpoint of this interface is connected.
bDelay	8	1384	0x01	0x01	Delay introduced by the data path. Expressed in number of frames.
wFormatTag	16	1392	0x0001	0x0001	PCM

Audio Class Specific INTERFACE Descriptor (11 bytes)

Field	Length (bits)	Offset (bits)	Decoded	Hex Value	Description
bLength	8	1408	0x0B	0x0B	Size of the descriptor, in bytes
bDescriptorType	8	1416	0x24	0x24	CS_INTERFACE Descriptor Type
bDescriptorSubType	8	1424	0x02	0x02	FORMAT_TYPE descriptor subtype
bFormatType	8	1432	0x01	0x01	FORMAT_TYPE_I
bChannels	8	1440	0x02	0x02	Indicates the number of physical channels in the audio data stream.
bSubframeSize	8	1448	0x02	0x02	The number of bytes occupied by one audio subframe. Can be 1, 2, 3 or 4.
bBitResolution	8	1456	0x10	0x10	The number of effectively used bits from the available bits in an audio subframe.
bSamFreqType	8	1464	0x01	0x01	Indicates how the sampling frequency can be programmed:
tsamFreq[1]	24	1472	0x00BB80	0x00BB80	Sampling frequency 1 in Hz for this isochronous data endpoint.

ENDPOINT Descriptor (9 bytes)

Field	Length (bits)	Offset (bits)	Decoded	Hex Value	Description
bLength	8	1496	0x09	0x09	Descriptor size is 9 bytes
bDescriptorType	8	1504	0x05	0x05	ENDPOINT Descriptor Type
bEndpointAddress	8	1512	0x02	0x02	This is an OUT endpoint with endpoint number 2
bmAttributes	8	1520	0x0D	0x0D	Types - Transfer: ISOCRONOUS Sync: Sync Usage: Data EP
wMaxPacketSize	16	1528	0x00C0	0x00C0	Maximum packet size for this endpoint is 192 Bytes. If Hi-Speed, 0 additional transactions per frame
Interval	8	1544	0x01	0x01	The polling interval value is every 1 Frames. If Hi-Speed, 1 uFrames/NAK
bRefresh	8	1552	0x00	0x00	Refresh Rate 2 ⁿ ms where n = 0
bSynchAddress	8	1560	0x00	0x00	Synchronization Endpoint (if used) is endpoint 0

Audio Class Specific ENDPOINT Descriptor (7 bytes)

Field	Length (bits)	Offset (bits)	Decoded	Hex Value	Description
bLength	8	1568	0x07	0x07	Size of the descriptor, in bytes
bDescriptorType	8	1576	0x25	0x25	CS_ENDPOINT Descriptor Type
bDescriptorSubType	8	1584	0x01	0x01	AUDIO_EP_GENERAL descriptor subtype
bmAttributes	8	1592	0x00	0x00	Bit 0: Sampling Frequency 0 Bit 1: Pitch 0 Bit 7: MaxPacketsOnly 0
bLockDelayUnits	8	1600	0x00	0x00	Indicates the units used for the wLockDelay field: 0: Undefined
wLockDelay	16	1608	0x0000	0x0000	Indicates the time it takes this endpoint to reliably lock its internal clock recovery circuitry. Units used depend on the value of the bLockDelayUnits field.

Figure 4. Newly added USB audio stream interface descriptor for speaker interface

3.1.2 Handle set interface request

After adding a new audio stream interface descriptor, modify the `USB_DeviceAudioSpeakerSetInterface()` function, as shown in Figure 5.

```

1288 usb_status_t USB_DeviceAudioSpeakerSetInterface(class_handle_t handle, uint8_t interface, uint8_t alternateSetting)
1289 {
1290     usb_status_t error = kStatus_USB_Success;
1291     if (USB_AUDIO_SPEAKER_STREAM_INTERFACE_ALTERNATE_1 == alternateSetting)
1292     {
1293         USB_DeviceAudioSpeakerStatusReset();
1294         memset((void *)(&audioPlayDataBuff[0]), 0, sizeof(audioPlayDataBuff));
1295         Init_Board_Audio();
1296     }
1297     error =
1298         USB_DeviceAudioRecv(g_deviceAudioComposite->audioUnified.audioSpeakerHandle,
1299                             USB_AUDIO_SPEAKER_STREAM_ENDPOINT, &audioPlayDataBuff[0], FS_ISO_OUT_ENDP_PACKET_SIZE);
1300     if (error != kStatus_USB_Success)
1301     {
1302         return error;
1303     }
1304     else
1305     {
1306         if defined(USB_DEVICE_AUDIO_USE_SYNC_MODE) && (USB_DEVICE_AUDIO_USE_SYNC_MODE > 0U)
1307         {
1308             if (!feedbackValueUpdating)
1309             {
1310                 *((uint32_t *)&usbAudioFeedBackBuffer[0]) = *((uint32_t *)&audioFeedBackBuffer[0]);
1311             }
1312             error = USB_DeviceAudioSend(g_deviceAudioComposite->audioUnified.audioSpeakerHandle,
1313                                         USB_AUDIO_SPEAKER_FEEDBACK_ENDPOINT, usbAudioFeedBackBuffer,
1314                                         (USB_SPEED_HIGH == g_composite.speed) ? HS_ISO_FEEDBACK_ENDP_PACKET_SIZE :
1315                                         FS_ISO_FEEDBACK_ENDP_PACKET_SIZE);
1316         }
1317     }
1318 }
1319
1320
1321 #if USB_AUDIO_FORMAT_SWITCH_ENABLE
1322 if (USB_AUDIO_SPEAKER_STREAM_INTERFACE_ALTERNATE_2 == alternateSetting)
1323 {
1324     USB_DeviceAudioSpeakerStatusReset();
1325     memset((void *)(&audioPlayDataBuff[0]), 0, sizeof(audioPlayDataBuff));
1326     Init_Board_Audio_32bit();
1327     error =
1328         USB_DeviceAudioRecv(g_deviceAudioComposite->audioUnified.audioSpeakerHandle,
1329                             USB_AUDIO_SPEAKER_STREAM_ENDPOINT, &audioPlayDataBuff[0], FS_ISO_OUT_ENDP_PACKET_SIZE_32BIT);
1330     if (error != kStatus_USB_Success)
1331     {
1332         return error;
1333     }
1334     else
1335     {
1336         if defined(USB_DEVICE_AUDIO_USE_SYNC_MODE) && (USB_DEVICE_AUDIO_USE_SYNC_MODE > 0U)
1337         {
1338             if (!feedbackValueUpdating)
1339             {
1340                 *((uint32_t *)&usbAudioFeedBackBuffer[0]) = *((uint32_t *)&audioFeedBackBuffer[0]);
1341             }
1342             error = USB_DeviceAudioSend(g_deviceAudioComposite->audioUnified.audioSpeakerHandle,
1343                                         USB_AUDIO_SPEAKER_FEEDBACK_ENDPOINT, usbAudioFeedBackBuffer,
1344                                         (USB_SPEED_HIGH == g_composite.speed) ? HS_ISO_FEEDBACK_ENDP_PACKET_SIZE :
1345                                         FS_ISO_FEEDBACK_ENDP_PACKET_SIZE);
1346         }
1347     }
1348 }

```

Figure 5. Handle Set interface request

After you switch the audio format on the PC, the USB host sends a set interface request to the USB device and sends the corresponding `bAlternateSetting` value to the USB device. After the USB device receives this request, it calls the `USB_DeviceAudioSpeakerSetInterface()` function to reconfigure the I2S interface and codec according to the `bAlternateSetting` value so that the I2S interface and USB audio use the same audio format.

3.2 Implementation details

This section gives the details of the implementation process.

3.2.1 Modify the `usb_device_descriptor.h` file

Add the corresponding macro definition for the 48 K/32 bit audio format, the relevant code is shown below.

```

#define USB_AUDIO_SPEAKER_STREAM_INTERFACE_ALTERNATE_COUNT (3)
#define USB_AUDIO_SPEAKER_STREAM_INTERFACE_ALTERNATE_2 (2)

#define AUDIO_OUT_FORMAT_CHANNELS_32BIT (0x02U)
#define AUDIO_OUT_FORMAT_BITS_32BIT (32)
#define AUDIO_OUT_FORMAT_SIZE_32BIT (0x04)

#define AUDIO_OUT_TRANSFER_LENGTH_ONE_FRAME_32BIT \

```

```
(AUDIO_OUT_SAMPLING_RATE_KHZ * AUDIO_OUT_FORMAT_CHANNELS_32BIT * AUDIO_OUT_FORMAT_SIZE_32BIT)
#define FS_ISO_OUT_ENDP_PACKET_SIZE_32BIT (AUDIO_OUT_TRANSFER_LENGTH_ONE_FRAME_32BIT)
```

3.2.2 Modify usb_device_descriptor.c file

Modify the definition of the global variables of the interfaces and endpoints related to the audio speaker, such as `g_UsbDeviceAudioSpeakerEndpoints` and `g_UsbDeviceAudioSpeakerStreamInterface`.

3.2.2.1 Modify the `g_UsbDeviceAudioSpeakerEndpoints` array

```
75 usb_device_endpoint_struct_t g_UsbDeviceAudioSpeakerEndpoints[USB_AUDIO_SPEAKER_STREAM_ENDPOINT_COUNT] = {
76     /* Audio generator ISO OUT pipe */
77     {
78         USB_AUDIO_SPEAKER_STREAM_ENDPOINT | (USB_OUT << USB_DESCRIPTOR_ENDPOINT_ADDRESS_DIRECTION_SHIFT),
79         USB_ENDPOINT_ISOCHRONOUS,
80 #if USB_AUDIO_FORMAT_SWITCH_ENABLE
81         FS_ISO_OUT_ENDP_PACKET_SIZE_32BIT,
82 #else
83         FS_ISO_OUT_ENDP_PACKET_SIZE, /* The max
84                                     packet size should be increased otherwise if host send data larger
85                                     than max packet size will cause DMA error. */
86 #endif
87         FS_ISO_OUT_ENDP_INTERVAL,
88     },
89 };
```

Figure 6. Definition of `g_UsbDeviceAudioSpeakerEndpoints` array

The ISO out packet length is the largest packet length among multiple audio formats.

3.2.2.2 Modify the `g_UsbDeviceAudioSpeakerStreamInterface` array

Add an alternate setting for the audio speaker stream interface.

```
218 usb_device_interface_struct_t g_UsbDeviceAudioSpeakerStreamInterface[] = {
219     {
220         USB_AUDIO_SPEAKER_STREAM_INTERFACE_ALTERNATE_0,
221         {
222             OU,
223             NULL,
224             NULL,
225             NULL,
226         },
227     },
228     {
229         USB_AUDIO_SPEAKER_STREAM_INTERFACE_ALTERNATE_1,
230         {
231             USB_AUDIO_SPEAKER_STREAM_ENDPOINT_COUNT,
232             g_UsbDeviceAudioSpeakerEndpoints,
233             NULL,
234             NULL,
235         },
236     },
237     #if USB_AUDIO_FORMAT_SWITCH_ENABLE
238     {
239         USB_AUDIO_SPEAKER_STREAM_INTERFACE_ALTERNATE_2,
240         {
241             USB_AUDIO_SPEAKER_STREAM_ENDPOINT_COUNT,
242             g_UsbDeviceAudioSpeakerEndpoints,
243             NULL,
244             NULL,
245         },
246     },
247 #endif
248 };
```

Figure 7. Definition of `g_UsbDeviceAudioSpeakerStreamInterface` array

3.2.2.3 Modify the USB configuration descriptor

In the USB configuration descriptor, an alternate audio stream interface descriptor must be added to the speaker interface and the length of the configuration descriptor must be modified, as shown in [Figure 8](#).

```

757 USB_DESCRIPTOR_LENGTH_CONFIGURE, /* Size of this descriptor in bytes */
758 USB_DESCRIPTOR_TYPE_CONFIGURE, /* CONFIGURATION Descriptor Type */
759 USB_SHORT_GET_LOW(
760     USB_DESCRIPTOR_LENGTH_CONFIGURE + USB_DESCRIPTOR_LENGTH_INTERFACE + USB_AUDIO_CONTROL_INTERFACE_HEADER_LENGTH +
761     USB_AUDIO_INPUT_TERMINAL_ONLY_DESC_SIZE + USB_AUDIO_FEATURE_UNIT_ONLY_DESC_SIZE(2, 1) +
762     USB_AUDIO_OUTPUT_TERMINAL_ONLY_DESC_SIZE + USB_AUDIO_INPUT_TERMINAL_ONLY_DESC_SIZE +
763     USB_AUDIO_FEATURE_UNIT_ONLY_DESC_SIZE(2, 1) + USB_AUDIO_OUTPUT_TERMINAL_ONLY_DESC_SIZE +
764     USB_DESCRIPTOR_LENGTH_AC_INTERRUPT_ENDPOINT + USB_DESCRIPTOR_LENGTH_INTERFACE +
765     USB_DESCRIPTOR_LENGTH_INTERFACE + USB_AUDIO_STREAMING_IFACE_DESC_SIZE + USB_AUDIO_STREAMING_TYPE_I_DESC_SIZE +
766     USB_ENDPOINT_AUDIO_DESCRIPTOR_LENGTH + USB_AUDIO_STREAMING_ENDP_DESC_SIZE + USB_DESCRIPTOR_LENGTH_INTERFACE +
767     USB_DESCRIPTOR_LENGTH_INTERFACE + USB_AUDIO_STREAMING_IFACE_DESC_SIZE + USB_AUDIO_STREAMING_TYPE_I_DESC_SIZE +
768     USB_ENDPOINT_AUDIO_DESCRIPTOR_LENGTH + USB_AUDIO_STREAMING_ENDP_DESC_SIZE
769 ) /* if defined(USB_DEVICE_AUDIO_USE_SYNC_MODE) && (USB_DEVICE_AUDIO_USE_SYNC_MODE > 0U)
770 #else
771     + USB_ENDPOINT_AUDIO_DESCRIPTOR_LENGTH
772 #endif
773 #endif
774 #if USB_AUDIO_FORMAT_SWITCH_ENABLE
775     + USB_DESCRIPTOR_LENGTH_INTERFACE + USB_AUDIO_STREAMING_IFACE_DESC_SIZE + USB_AUDIO_STREAMING_TYPE_I_DESC_SIZE +
776     USB_ENDPOINT_AUDIO_DESCRIPTOR_LENGTH + USB_AUDIO_STREAMING_ENDP_DESC_SIZE
777 #endif
778 #endif
779     + USB_DESCRIPTOR_LENGTH_INTERFACE + USB_DESCRIPTOR_LENGTH_HID + USB_DESCRIPTOR_LENGTH_ENDPOINT),

```



```

1132 #if USB_AUDIO_FORMAT_SWITCH_ENABLE
1133
1134 /* Audio Class Specific INTERFACE Descriptor, alternative interface 1 */
1135 USB_DESCRIPTOR_LENGTH_INTERFACE, /* Descriptor size is 9 bytes */
1136 USB_DESCRIPTOR_TYPE_INTERFACE, /* INTERFACE Descriptor Type */
1137 USB_AUDIO_SPEAKER_STREAM_INTERFACE_INDEX, /*The number of this interface is 1. */
1138 USB_AUDIO_SPEAKER_STREAM_INTERFACE_ALTERNATE_2, /* The value used to select the alternate setting for this interface
1139           is 1 */
1140 #if defined(USB_DEVICE_AUDIO_USE_SYNC_MODE) && (USB_DEVICE_AUDIO_USE_SYNC_MODE > 0U)
1141     0x01U, /* The number of endpoints used by this interface is 1 (excluding endpoint zero) */
1142 #else
1143     0x02U, /* The number of endpoints used by this interface is 2 (excluding endpoint zero) */
1144 #endif
1145 USB_AUDIO_CLASS, /* The interface implements the Audio Interface class */
1146 USB_SUBCLASS_AUDIOSTREAM, /* The interface implements the AUDIOSTREAMING Subclass */
1147 USB_AUDIO_PROTOCOL, /* The interface doesn't use any class-specific protocols */
1148 0x00U, /* The device doesn't have a string descriptor describing this interface */
1149
1150 /* Audio Class Specific CS INTERFACE Descriptor */
1151 USB_AUDIO_STREAMING_IFACE_DESC_SIZE, /* Size of the descriptor, in bytes */
1152 USB_DESCRIPTOR_TYPE_AUDIO_CS_INTERFACE, /* CS INTERFACE Descriptor Type */
1153 USB_DESCRIPTOR_SUBTYPE_AUDIO_STREAMING_AS_GENERAL, /* AS GENERAL descriptor subtype */
1154 USB_AUDIO_SPEAKER_CONTROL_INPUT_TERMINAL_ID, /* The Terminal ID of the Terminal to which the endpoint of this
1155           interface is connected. */
1156 0x01U, /* Delay introduced by the data path. Expressed in number of frames. */
1157 0x01U, 0x00U, /* PCM */
1158
1159 /* Audio Class Specific type I format INTERFACE Descriptor */
1160 USB_AUDIO_STREAMING_TYPE_I_DESC_SIZE, /* bLength (11) */
1161 USB_DESCRIPTOR_TYPE_AUDIO_CS_INTERFACE, /* bDescriptorType (CS INTERFACE) */
1162 USB_DESCRIPTOR_SUBTYPE_AUDIO_STREAMING_FORMAT_TYPE, /* DescriptorSubType: AUDIO STREAMING FORMAT TYPE */
1163 USB_AUDIO_FORMAT_TYPE_I, /* Format Type: Type I */
1164 AUDIO_OUT_FORMAT_CHANNELS_32BIT, /* Number of Channels */
1165 AUDIO_OUT_FORMAT_SIZE_32BIT, /* SubFrame Size: The number of bytes occupied by one audio subframe. */
1166 AUDIO_OUT_FORMAT_BITS_32BIT, /* Bit Resolution: 8 bits per sample */
1167 0x01U, /* One frequency supported */
1168 /* 0x40, 0x1F, 0x00U, 9 kHz */
1169 /* 0x80U, 0x3EU, 0x00U, 16 kHz */
1170 TSAMFREQ2BYTES(AUDIO_OUT_SAMPLING_RATE_KHZ * 1000),
1171 /* 0x80, 0xBB, 0x00U, 48 kHz */
1172 /* 0x00U, 0xFA, 0x00U, 72 kHz */
1173 #if defined(USB_DEVICE_AUDIO_USE_SYNC_MODE) && (USB_DEVICE_AUDIO_USE_SYNC_MODE > 0U)
1174 /* ENDPOINT Descriptor */
1175 USB_ENDPOINT_DESCRIPTOR_LENGTH, /* Descriptor size is 9 bytes */
1176 USB_DESCRIPTOR_TYPE_ENDPOINT, /* Descriptor type (endpoint descriptor) */
1177 USB_AUDIO_SPEAKER_STREAM_ENDPOINT |
1178     (USB_OUT << USB_DESCRIPTOR_ENDPOINT_ADDRESS_DIRECTION_SHIFT), /* OUT endpoint address 1 */
1179 USB_ENDPOINT_ISOCHRONOUS | 0x0CU, /* Isochronous endpoint and Synchronous */
1180 USB_SHORT_GET_LOW(FS_ISO_OUT_ENDP_PACKET_SIZE_32BIT), USB_SHORT_GET_HIGH(FS_ISO_OUT_ENDP_PACKET_SIZE_32BIT), /* 16 bytes */
1181 FS_ISO_OUT_ENDP_INTERVAL, /* bInterval(0x01U): x ms */
1182 0x00U, /* Unused */
1183 0x00U, /* Synchronization Endpoint (if used) is endpoint 0x83 */
1184 #else
1185 /* ENDPOINT Descriptor */
1186 USB_ENDPOINT_AUDIO_DESCRIPTOR_LENGTH, /* Descriptor size is 9 bytes */
1187 USB_DESCRIPTOR_TYPE_ENDPOINT, /* Descriptor type (endpoint descriptor) */
1188 USB_AUDIO_SPEAKER_STREAM_ENDPOINT |
1189     (USB_OUT << USB_DESCRIPTOR_ENDPOINT_ADDRESS_DIRECTION_SHIFT), /* OUT endpoint address 1 */
1190 USB_ENDPOINT_ISOCHRONOUS | 0x04, /* Isochronous endpoint */
1191 USB_SHORT_GET_LOW(FS_ISO_OUT_ENDP_PACKET_SIZE + AUDIO_OUT_FORMAT_CHANNELS * AUDIO_OUT_FORMAT_SIZE),
1192 USB_SHORT_GET_HIGH(FS_ISO_OUT_ENDP_PACKET_SIZE + AUDIO_OUT_FORMAT_CHANNELS * AUDIO_OUT_FORMAT_SIZE), /* 16 bytes */
1193 FS_ISO_OUT_ENDP_INTERVAL, /* bInterval(0x01U): x ms */
1194 0x00U, /* Unused */
1195 USB_AUDIO_SPEAKER_FEEDBACK_ENDPOINT |
1196     (USB_IN << USB_DESCRIPTOR_ENDPOINT_ADDRESS_DIRECTION_SHIFT), /* Synchronization Endpoint (if used) is endpoint 0x83 */
1197 #endif
1198 #endif
1199 /* Audio Class Specific ENDPOINT Descriptor */
1200 USB_AUDIO_STREAMING_ENDP_DESC_SIZE, /* Size of the descriptor, in bytes */
1201 USB_AUDIO_STREAM_ENDPOINT_DESCRIPTOR, /* CS_ENDPOINT Descriptor Type */
1202 USB_AUDIO_EP_GENERAL_DESCRIPTOR_SUBTYPE, /* AUDIO_EP_GENERAL descriptor subtype */
1203 0x00U, /* Bit 0: Sampling Frequency 0
1204           Bit 1: Pitch 0
1205           Bit 7: MaxPacketsOnly 0 */
1206 0x00U, /* Indicates the units used for the wLockDelay field: 0: Undefined */
1207 0x00U, 0x00U, /* Indicates the time it takes this endpoint to reliably lock its internal clock recovery circuitry */
1208
1209 #if defined(USB_DEVICE_AUDIO_USE_SYNC_MODE) && (USB_DEVICE_AUDIO_USE_SYNC_MODE > 0U)
1210 #else
1211 /* Endpoint 3 Feedback ENDPOINT */
1212 USB_ENDPOINT_AUDIO_DESCRIPTOR_LENGTH, /* bLength */
1213 USB_DESCRIPTOR_TYPE_ENDPOINT, /* bDescriptorType */
1214 USB_AUDIO_SPEAKER_FEEDBACK_ENDPOINT |
1215     (USB_IN << USB_DESCRIPTOR_ENDPOINT_ADDRESS_DIRECTION_SHIFT), /* This is an IN endpoint with endpoint number 3 */
1216 USB_ENDPOINT_ISOCHRONOUS | 0x10, /* Types -
1217           Transfer: ISOCHRONOUS
1218           Sync: Async
1219           Usage: Feedback EP */
1220 FS_ISO_FEEDBACK_ENDP_PACKET_SIZE, 0x00, /* wMaxPacketSize */
1221 FS_ISO_FEEDBACK_ENDP_INTERVAL, /* interval polling(2*x ms) */
1222 0x05, /* bRefresh(32ms) */
1223 0x00, /* unused */
1224 #endif
1225 #endif

```

Figure 8. USB configuration descriptor

3.2.3 Modify the audio_unified.h file

The macro below is used to calculate the length of the ring buffer that stores audio data, so the largest packet length among multiple audio formats must be used.

```
#define AUDIO_PLAY_BUFFER_SIZE_ONE_FRAME_32BIT AUDIO_OUT_TRANSFER_LENGTH_ONE_FRAME_32BIT
```

3.2.4 Modify the audio_unified.c file

This section describes the process of modification of the `audio_unified.c` file.

3.2.4.1 Modify the length of the data buffer of audio speaker

Modify the length of the data buffer related to audio speaker to make them support the 48 K/32 bit audio format.

```
#if USB_AUDIO_FORMAT_SWITCH_ENABLE
uint8_t audioPlayDataBuff[AUDIO_SPEAKER_DATA_WHOLE_BUFFER_COUNT_NORMAL * \
AUDIO_PLAY_BUFFER_SIZE_ONE_FRAME_32BIT];
#else
uint8_t audioPlayDataBuff[AUDIO_SPEAKER_DATA_WHOLE_BUFFER_COUNT_NORMAL * \
AUDIO_PLAY_BUFFER_SIZE_ONE_FRAME];
#endif

#if USB_AUDIO_FORMAT_SWITCH_ENABLE
uint8_t audioPlayPacket[FS_ISO_OUT_ENDP_PACKET_SIZE_32BIT];
#else
uint8_t audioPlayPacket[FS_ISO_OUT_ENDP_PACKET_SIZE];
#endif
```

The `audioPlayDataBuff` is used as a ring buffer to store data from the USB audio speaker interface. The `audioPlayPacket` array is used to store a complete frame of USB audio data, so its length must meet the maximum packet length in multiple audio formats.

3.2.4.2 Modify the USB_AudioSpeakerPutBuffer() function

The `USB_AudioSpeakerPutBuffer()` function is used to copy the latest USB audio data frame in `audioPlayPacket` to the ring buffer `audioPlayDataBuff`. In different audio formats, the length of an audio data frame sent by the USB host is different. It is necessary to calculate the the length of the audio data packet in the current audio format according to the value of `g_USBSpeakerAlternateSettingIndex`, then copy the data of the corresponding length to the ring buffer.

```
589 /* The USB_AudioSpeakerPutBuffer() function fills the audioRecDataBuff with audioPlayPacket in every callback*/
590 void USB_AudioSpeakerPutBuffer(uint8_t *buffer, uint32_t size)
591 {
592     uint32_t remainBufferSpace;
593     uint32_t buffer_length = 0;
594
595     uint32_t audio_out_format_size = 0;
596
597     if(g_USBSpeakerAlternateSettingIndex == 1)
598     {
599         audio_out_format_size = AUDIO_OUT_FORMAT_SIZE;
600     }
601     else if(g_USBSpeakerAlternateSettingIndex == 2)
602     {
603         audio_out_format_size = AUDIO_OUT_FORMAT_SIZE_32BIT;
604     }
605     remainBufferSpace = g_deviceAudioComposite->audioUnified.audioPlayBufferSize - USB_AudioSpeakerBufferSpaceUsed();
606     if (size >= remainBufferSpace) /* discard the overflow data */
607     {
608         if (remainBufferSpace > (AUDIO_OUT_FORMAT_CHANNELS * audio_out_format_size))
609         {
610             size = (remainBufferSpace - (AUDIO_OUT_FORMAT_CHANNELS * audio_out_format_size));
611         }
612         else
613         {
614             size = 0;
615         }
616     }
```

Figure 9. `USB_AudioSpeakerPutBuffer()` function

3.2.4.3 Modify the USB_DeviceAudioSpeakerSetInterface() function

The process `bAlternateSetting=2` in the `USB_DeviceAudioSpeakerSetInterface()` function is shown in [Figure 5](#). After the user switches the audio format on the PC, the USB host sends a set interface request. After receiving this request, the USB device reconfigures the I2S interface and codec. Every time the audio format is switched on the PC, the USB host sends a set interface (`bAlternateSetting=0`) request first, and then send the actual alternate setting. For specific processing details, refer to the code in the attachment.

3.2.5 Modify the composite.c file

This section describes the details of modifying the `composite.c` file.

3.2.5.1 Modify the length of the related array

```
#if USB_AUDIO_FORMAT_SWITCH_ENABLE
static uint8_t audioPlayDMATempBuff[AUDIO_PLAY_BUFFER_SIZE_ONE_FRAME_32BIT];
#else
static uint8_t audioPlayDMATempBuff[AUDIO_PLAY_BUFFER_SIZE_ONE_FRAME];
#endif
```

When the USB host stops sending audio data, the DMA data source must be configured as `audioPlayDMATempBuff`, the data in `audioPlayDMATempBuff` are all 0x00. The data length of `audioPlayDMATempBuff` must meet the maximum packet length in different audio formats.

3.2.5.2 Add the g_USBSpeakerAlternateSettingIndex variable

To record the current interface configuration of the USB audio speaker, add a variable (`g_USBSpeakerAlternateSettingIndex`). Different interface configurations correspond to different audio formats, as shown in [Table 1](#).

```
uint8_t g_USBSpeakerAlternateSettingIndex = 0;
```

Table 1. Correspondence between interface configuration and audio format

g_USBSpeakerAlternateSettingIndex	Audio format
0	-
1	48 K/16 bit
2	48 K/32 bit

3.2.5.3 Modify the I2S interface configuration

Different USB audio formats correspond to different I2S configurations, so it is also necessary to add corresponding I2S configurations for the 48 K/32 bit format. Such as adding `Init_Board_Audio_32bit()`, `BOARD_DMA_EDMA_Start_32bit()`, `I2S_USB_Audio_TxInit_32bit()`, and other functions.

3.2.5.3.1 Add the I2S_USB_Audio_TxInit_32bit() function

```
293 void I2S_USB_Audio_TxInit_32bit(I2S_Type *SAIBase)
294 {
295     I2S_TxGetDefaultConfig(&s_TxConfig);
296     s_TxConfig.dataLength = 32;
297     s_TxConfig.frameLength = 64;
298     s_TxConfig.divider = (CLOCK_GetPllOutFreq() / AUDIO_SAMPLING_RATE / AUDIO_OUT_FORMAT_BITS_32BIT / AUDIO_OUT_FORMAT_CHANNELS_32BIT);
299     I2S_TxInit(DEMO_I2S_TX, &s_TxConfig);
300 }
```

Figure 10. I2S_USB_Audio_TxInit_32bit() function

Use the `I2S_USB_Audio_TxInit_32bit()` function to configure the I2S interface to the 48 K/32 bit format.

3.2.5.3.2 Add the Init_Board_Audio_32bit() function

```

452 void Init_Board_Audio_32bit(void)
453 {
454     usb_echo("Init Audio I2S and CODEC\r\n");
455
456     BOARD_USB_AUDIO_KEYBOARD_Init();
457
458     I2S_USB_Audio_TxInit_32bit(DEMO_I2S_TX);
459     I2S_USB_Audio_RxInit(DEMO_I2S_RX);
460     wm8904Config.format.bitWidth = kWM8904_BitWidth32;
461     BOARD_Codec_Init();
462     BOARD_DMA_EDMA_Config();
463     BOARD_DMA_EDMA_Start_32bit();
464 }

```

Figure 11. Init_Board_Audio_32bit() function

Use Init_Board_Audio_32bit() function to configure the I2S interface and codec to the 48 K/32 bit format.

3.2.5.3.3 Add the BOARD_DMA_EDMA_Start_32bit() function

```

411 void BOARD_DMA_EDMA_Start_32bit()
412 {
413     s_TxTransfer.dataSize = FS_ISO_OUT_ENDP_PACKET_SIZE_32BIT;
414     s_TxTransfer.data = audioPlayDMATempBuff;
415     s_RxTransfer.dataSize = FS_ISO_IN_ENDP_PACKET_SIZE;
416     s_RxTransfer.data = audioRecDMATempBuff;
417
418     I2S_TxTransferCreateHandleDMA(DEMO_I2S_TX, &s_TxHandle, &s_DmaTxHandle, TxCallback, (void *)&s_TxTransfer);
419     I2S_RxTransferCreateHandleDMA(DEMO_I2S_RX, &s_RxHandle, &s_DmaRxHandle, RxCallback, (void *)&s_RxTransfer);
420
421     /* need to queue two receive buffers so when the first one is filled,
422      * the other is immediately starts to be filled */
423     I2S_RxTransferReceiveDMA(DEMO_I2S_RX, &s_RxHandle, s_RxTransfer);
424     I2S_RxTransferReceiveDMA(DEMO_I2S_RX, &s_RxHandle, s_RxTransfer);
425
426     I2S_TxTransferSendDMA(DEMO_I2S_TX, &s_TxHandle, s_TxTransfer);
427 }

```

Figure 12. BOARD_DMA_EDMA_Start_32bit() function

Use BOARD_DMA_EDMA_Start_32bit() to modify the data length of each DMA transfer to the USB data packet length corresponding to the 48 K/32 bit format.

3.2.5.4 Modify the USB_DeviceCallback() function

When the USB host initiates a set interface request, the value of the g_USBSpeakerAlternateSettingIndex variable is updated to indicate different interface configurations, and the length of each frame of USB audio packet is updated according to the value of g_USBSpeakerAlternateSettingIndex.

```

731 case USB_AUDIO_SPEAKER_STREAM_INTERFACE_INDEX:
732     if (alternateSetting < USB_AUDIO_SPEAKER_STREAM_INTERFACE_ALTERNATE_COUNT)
733     {
734         #if USB_AUDIO_FORMAT_SWITCH_ENABLE
735             g_USBSpeakerAlternateSettingIndex = alternateSetting;
736
737             if(g_USBSpeakerAlternateSettingIndex == 1)
738             {
739                 g_composite.audioUnified.audioPlayTransferSize = AUDIO_PLAY_BUFFER_SIZE_ONE_FRAME;
740                 g_composite.audioUnified.currentStreamOutMaxPacketSize = (FS_ISO_OUT_ENDP_PACKET_SIZE);
741             }
742             else if(g_USBSpeakerAlternateSettingIndex == 2)
743             {
744                 g_composite.audioUnified.audioPlayTransferSize = AUDIO_PLAY_BUFFER_SIZE_ONE_FRAME_32BIT;
745                 g_composite.audioUnified.currentStreamOutMaxPacketSize = (FS_ISO_OUT_ENDP_PACKET_SIZE_32BIT);
746             }
747         #endif
748     }
749     error = USB_DeviceAudioSpeakerSetInterface(g_composite.audioUnified.audioSpeakerHandle,
750         interface, alternateSetting);

```

Figure 13. USB_DeviceCallback() function

4 Test

Connect the FS USB interface of LPC55S69-EVK to a PC through a micro USB cable, download the modified program to the board, and reset the board. After the enumeration is successful, the USB host recognizes a USB audio device. The characteristics of the device are shown in [Figure 15](#).



Figure 14. LPC55S69 EVK Board

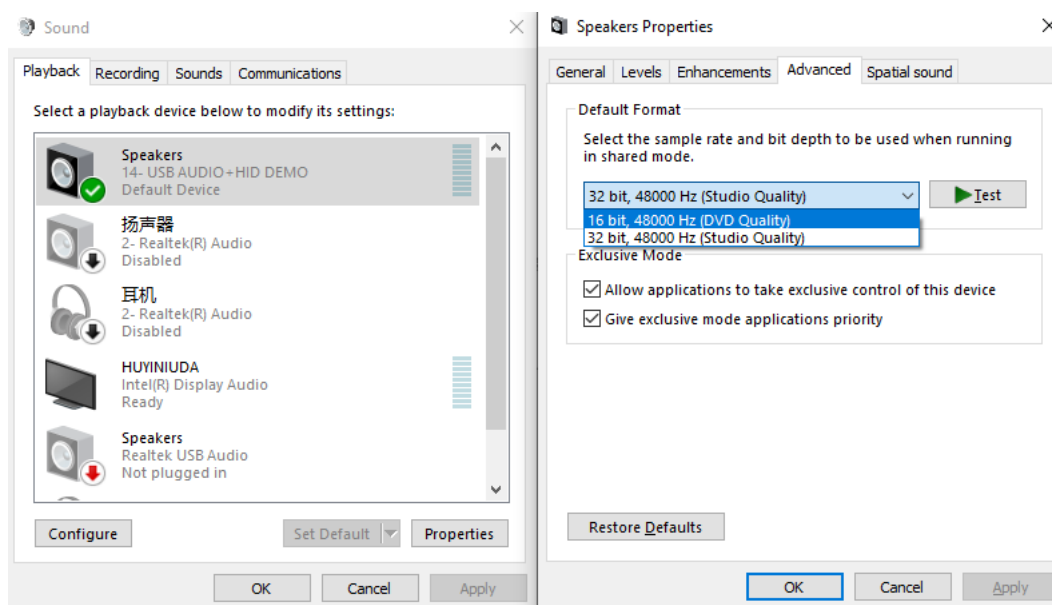


Figure 15. PC recognizes devices that support multiple audio formats

When the audio format is switched on the PC, the PC sends the set interface request to the USB audio device and send audio data according to the new audio format, as shown in [Figure 16](#).

Transfer	F	Control	ADDR	ENDP	bRequest	wIndex	Alt. Setting	wLength	Time	Time Stamp	
43	S	SET	18	0	SET_INTERFACE	2	2	0	217.877 ms	12 . 581 501 116	
Transfer	F	Isoch	ADDR	ENDP	Bytes Transferred	Isochronous Packet Info				Time	Time Stamp
44	S	OUT	18	2	4316160	11240 packets ranging from 384 bytes to 384 bytes				11.242 sec	12 . 799 378 466
Transfer	F	Control	ADDR	ENDP	bRequest	wIndex	Alt. Setting	wLength	Time	Time Stamp	
45	S	SET	18	0	SET_INTERFACE	2	0	0	2.588 sec	24 . 041 222 750	
Transfer	F	Control	ADDR	ENDP	bRequest	wIndex	Alt. Setting	wLength	Time	Time Stamp	
46	S	SET	18	0	SET_INTERFACE	2	1	0	206.752 ms	26 . 629 646 900	
Transfer	F	Isoch	ADDR	ENDP	Bytes Transferred	Isochronous Packet Info				Time	Time Stamp
47	S	OUT	18	2	1985280	10340 packets ranging from 192 bytes to 192 bytes				10.342 sec	26 . 836 398 550
Transfer	F	Control	ADDR	ENDP	bRequest	wIndex	Alt. Setting	wLength	Time	Time Stamp	
48	S	SET	18	0	SET_INTERFACE	2	0	0	4.075 sec	37 . 178 471 400	
Transfer	F	Control	ADDR	ENDP	bRequest	wIndex	Alt. Setting	wLength	Time	Time Stamp	
49	S	SET	18	0	SET_INTERFACE	2	2	0	219.840 ms	41 . 253 577 832	
Transfer	F	Isoch	ADDR	ENDP	Bytes Transferred	Isochronous Packet Info				Time	Time Stamp
50	S	OUT	18	2	1091328	2842 packets ranging from 384 bytes to 384 bytes				41 . 473 417 682	

Figure 16. PC sends set interface request

Set the audio format to 48 K/16 bit on the PC, the USB host sends a set interface request with Alt. Setting = 1 to the USB device, and then send a packet of audio data every 1 ms, the packet length is 192 bytes. Set the audio format to 48 K/32 bit on the PC, and the USB host sends a set interface request with Alt. Setting = 2 to the USB device, and then send a packet of audio data every 1 ms, the packet length is 384 bytes.

5 Conclusion

This application note introduces how to support multiple audio formats based on the USB audio speaker example in the LPC55S69 SDK, and provides detailed implementation steps. You can refer to this article to implement more audio format support.

6 References

1. Access USB Technology and Application Based on Microcontrollers.
2. [USB 2.0 specification](#).

7 Revision history

Table 2. Revision history

Revision number	Date	Substantive changes
0	08 November 2021	Initial release

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

Limited warranty and liability — Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including "typicals," must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

Right to make changes - NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Security — Customer understands that all NXP products may be subject to unidentified or documented vulnerabilities. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately. Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP. NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX, EMBRACE, GREENCHIP, HITAG, ICODE, JCOP, LIFE, VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFARE4MOBILE, MIGLO, NTAG, ROADLINK, SMARTLX, SMARTMX, STARPLUG, TOPFET, TRENCHMOS, UCODE, Freescale, the Freescale logo, Altivec, CodeWarrior, ColdFire, ColdFire+, the Energy Efficient Solutions logo, Kinetis, Layerscape, MagniV, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, QorIQ Qonverge, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Vybrid, Airfast, BeeKit, BeeStack, CoreNet, Flexis, MXC, Platform in a Package, QUICC Engine, Tower, TurboLink, EdgeScale, EdgeLock, eIQ, and Immersive3D are trademarks of NXP B.V. All other product or service names are the property of their respective owners. AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, uVision, Versatile are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org. M, M Mobileye and other Mobileye trademarks or logos appearing herein are trademarks of Mobileye Vision Technologies Ltd. in the United States, the EU and/or other jurisdictions.

© NXP B.V. 2021.

All rights reserved.

For more information, please visit: <http://www.nxp.com>

For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 08 November 2021

Document identifier: AN13447

