AN13447

How To Implement Multiple USB Audio Formats On LPC5500

Rev. 0 — 08 November 2021 Application Note

Contents

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.

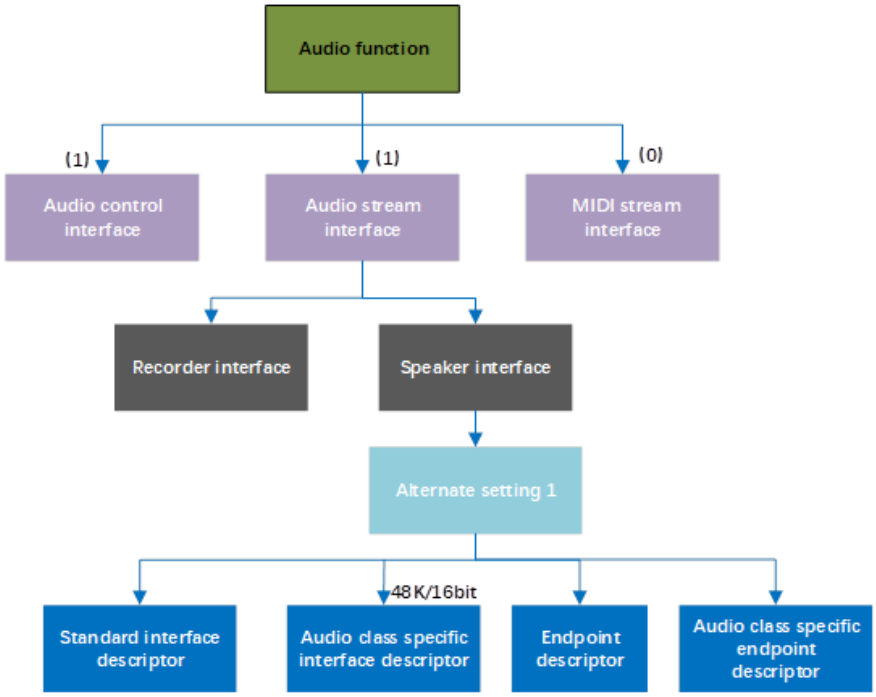
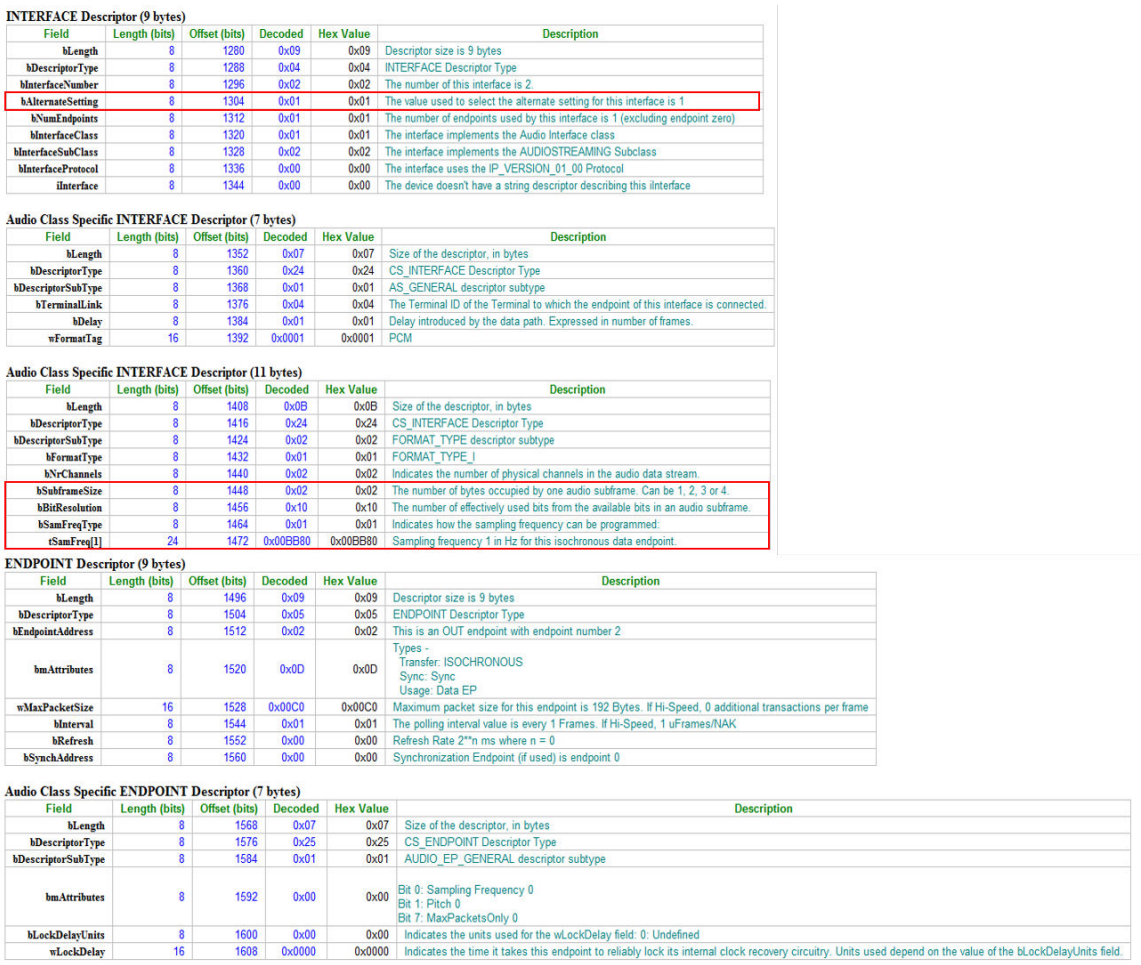


Figure 1. Structure diagram of the audio speaker interface

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 iInterface

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
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 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.

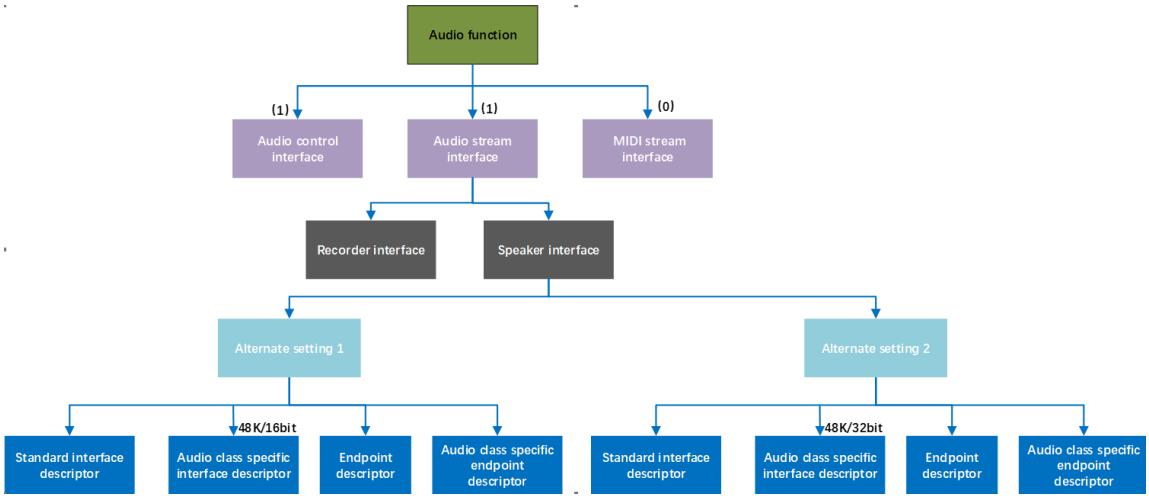


Figure 3. Structure diagram of the audio speaker interface that supports two audio formats

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
iInterface	8	1344	0x00	0x00	The device doesn't have a string descriptor describing this iInterface

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
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.

```
usb_status_t USB_DeviceAudioSpeakerSetInterface(class_handle_t handle, uint8_t interface, uint8_t alternateSetting)
{
    usb_status_t error = kStatus_USB_Success;

    if (USB_AUDIO_SPEAKER_STREAM_INTERFACE_ALTERNATE_1 == alternateSetting)
    {
        USB_DeviceAudioSpeakerStatusReset();
        memset((void *)(&audioPlayDataBuff[0]),0,sizeof(audioPlayDataBuff));
        Init_Board_Audio();

        error =
            USB_DeviceAudioRecv(g_deviceAudioComposite->audioUnified.audioSpeakerHandle,
                                USB_AUDIO_SPEAKER_STREAM_ENDPOINT, &audioPlayDataBuff[0], FS_ISO_OUT_ENDP_PACKET_SIZE);
        if (error != kStatus_USB_Success)
        {
            return error;
        }
        else
        {
#if defined(USB_DEVICE_AUDIO_USE_SYNC_MODE) && (USB_DEVICE_AUDIO_USE_SYNC_MODE > 0U)
#else
            if (!feedbackValueUpdating)
            {
                *((uint32_t *)&usbAudioFeedBackBuffer[0]) = *((uint32_t *)&audioFeedBackBuffer[0]);
            }
            error = USB_DeviceAudioSend(g_deviceAudioComposite->audioUnified.audioSpeakerHandle,
                                        USB_AUDIO_SPEAKER_FEEDBACK_ENDPOINT, usbAudioFeedBackBuffer,
                                        (USB_SPEED_HIGH == g_composite.speed) ? HS_ISO_FEEDBACK_ENDP_PACKET_SIZE :
                                                                                FS_ISO_FEEDBACK_ENDP_PACKET_SIZE);
#endif
        }
    }

#if USB_AUDIO_FORMAT_SWITCH_ENABLE
    if (USB_AUDIO_SPEAKER_STREAM_INTERFACE_ALTERNATE_2 == alternateSetting)
    {
        USB_DeviceAudioSpeakerStatusReset();
        memset((void *)(&audioPlayDataBuff[0]),0,sizeof(audioPlayDataBuff));
        Init_Board_Audio_32bit();
        error =
            USB_DeviceAudioRecv(g_deviceAudioComposite->audioUnified.audioSpeakerHandle,
                                USB_AUDIO_SPEAKER_STREAM_ENDPOINT, &audioPlayDataBuff[0], FS_ISO_OUT_ENDP_PACKET_SIZE_32BIT);
        if (error != kStatus_USB_Success)
        {
            return error;
        }
        else
        {
#if defined(USB_DEVICE_AUDIO_USE_SYNC_MODE) && (USB_DEVICE_AUDIO_USE_SYNC_MODE > 0U)
#else
            if (!feedbackValueUpdating)
            {
                *((uint32_t *)&usbAudioFeedBackBuffer[0]) = *((uint32_t *)&audioFeedBackBuffer[0]);
            }
            error = USB_DeviceAudioSend(g_deviceAudioComposite->audioUnified.audioSpeakerHandle,
                                        USB_AUDIO_SPEAKER_FEEDBACK_ENDPOINT, usbAudioFeedBackBuffer,
                                        (USB_SPEED_HIGH == g_composite.speed) ? HS_ISO_FEEDBACK_ENDP_PACKET_SIZE :
                                                                                FS_ISO_FEEDBACK_ENDP_PACKET_SIZE);
#endif
        }
    }
```

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

```
usb_device_endpoint_struct_t g_UsbDeviceAudioSpeakerEndpoints[USB_AUDIO_SPEAKER_STREAM_ENDPOINT_COUNT] = {
    /* Audio generator ISO OUT pipe */
    {
        USB_AUDIO_SPEAKER_STREAM_ENDPOINT | (USB_OUT << USB_DESCRIPTOR_ENDPOINT_ADDRESS_DIRECTION_SHIFT),
        USB_ENDPOINT_ISOCHRONOUS,
#if USB_AUDIO_FORMAT_SWITCH_ENABLE
        FS_ISO_OUT_ENDP_PACKET_SIZE_32BIT,
#else
        FS_ISO_OUT_ENDP_PACKET_SIZE, /* The max
                        packet size should be increased otherwise if host send data larger
                        than max packet size will cause DMA error. */
#endif
        FS_ISO_OUT_ENDP_INTERVAL,
    },
};
```

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.

```
usb_device_interface_struct_t g_UsbDeviceAudioSpeakerStreamInterface[] = {
    {
        USB_AUDIO_SPEAKER_STREAM_INTERFACE_ALTERNATE_0,
        {
            0U,
            NULL,
        },
        NULL,
    },
    {
        USB_AUDIO_SPEAKER_STREAM_INTERFACE_ALTERNATE_1,
        {
            USB_AUDIO_SPEAKER_STREAM_ENDPOINT_COUNT,
            g_UsbDeviceAudioSpeakerEndpoints,
        },
        NULL,
    },
#if USB_AUDIO_FORMAT_SWITCH_ENABLE
        USB_AUDIO_SPEAKER_STREAM_INTERFACE_ALTERNATE_2,
        {
            USB_AUDIO_SPEAKER_STREAM_ENDPOINT_COUNT,
            g_UsbDeviceAudioSpeakerEndpoints,
        },
        NULL,
#endif
};
```

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.

```
    USB_DESCRIPTOR_LENGTH_CONFIGURE, /* Size of this descriptor in bytes */
    USB_DESCRIPTOR_TYPE_CONFIGURE,   /* CONFIGURATION Descriptor Type */
    USB_SHORT_GET_LOW(
        USB_DESCRIPTOR_LENGTH_CONFIGURE + USB_DESCRIPTOR_LENGTH_INTERFACE + USB_AUDIO_CONTROL_INTERFACE_HEADER_LENGTH +
        USB_AUDIO_INPUT_TERMINAL_ONLY_DESC_SIZE + USB_AUDIO_FEATURE_UNIT_ONLY_DESC_SIZE(2, 1) +
        USB_AUDIO_OUTPUT_TERMINAL_ONLY_DESC_SIZE + USB_AUDIO_INPUT_TERMINAL_ONLY_DESC_SIZE +
        USB_AUDIO_FEATURE_UNIT_ONLY_DESC_SIZE(2, 1) + USB_AUDIO_OUTPUT_TERMINAL_ONLY_DESC_SIZE +
        USB_DESCRIPTOR_LENGTH_AC_INTERRUPT_ENDPOINT + USB_DESCRIPTOR_LENGTH_INTERFACE +
        USB_DESCRIPTOR_LENGTH_INTERFACE + USB_AUDIO_STREAMING_IFACE_DESC_SIZE + USB_AUDIO_STREAMING_TYPE_I_DESC_SIZE +
        USB_ENDPOINT_AUDIO_DESCRIPTOR_LENGTH + USB_AUDIO_STREAMING_ENDP_DESC_SIZE + USB_DESCRIPTOR_LENGTH_INTERFACE +
        USB_DESCRIPTOR_LENGTH_INTERFACE + USB_AUDIO_STREAMING_IFACE_DESC_SIZE + USB_AUDIO_STREAMING_TYPE_I_DESC_SIZE +
        USB_ENDPOINT_AUDIO_DESCRIPTOR_LENGTH + USB_AUDIO_STREAMING_ENDP_DESC_SIZE
#if defined(USB_DEVICE_AUDIO_USE_SYNC_MODE) && (USB_DEVICE_AUDIO_USE_SYNC_MODE > 0U)
#else
        + USB_ENDPOINT_AUDIO_DESCRIPTOR_LENGTH
#endif

#if USB_AUDIO_FORMAT_SWITCH_ENABLE
        + USB_DESCRIPTOR_LENGTH_INTERFACE +  USB_AUDIO_STREAMING_IFACE_DESC_SIZE + USB_AUDIO_STREAMING_TYPE_I_DESC_SIZE +
        USB_ENDPOINT_AUDIO_DESCRIPTOR_LENGTH +  USB_AUDIO_STREAMING_ENDP_DESC_SIZE

#endif
        + USB_DESCRIPTOR_LENGTH_INTERFACE + USB_DESCRIPTOR_LENGTH_HID + USB_DESCRIPTOR_LENGTH_ENDPOINT),
```

```
#if USB_AUDIO_FORMAT_SWITCH_ENABLE

    /* Audio Class Specific INTERFACE Descriptor, alternative interface 1 */
    USB_DESCRIPTOR_LENGTH_INTERFACE,               /* Descriptor size is 9 bytes  */
    USB_DESCRIPTOR_TYPE_INTERFACE,                 /* INTERFACE Descriptor Type  */
    USB_AUDIO_SPEAKER_STREAM_INTERFACE_INDEX,      /*The number of this interface is 1.  */
    USB_AUDIO_SPEAKER_STREAM_INTERFACE_ALTERNATE_2, /* The value used to select the alternate setting for this interface
                                                      is 1  */
#if defined(USB_DEVICE_AUDIO_USE_SYNC_MODE) && (USB_DEVICE_AUDIO_USE_SYNC_MODE > 0U)
    0x01U,                      /* The number of endpoints used by this interface is 1 (excluding endpoint zero)    */
#else
    0x02U,                      /* The number of endpoints used by this interface is 2 (excluding endpoint zero)    */
#endif
    USB_AUDIO_CLASS,            /* The interface implements the Audio Interface class   */
    USB_SUBCLASS_AUDIOSTREAM, /* The interface implements the AUDIOSTREAMING Subclass   */
    USB_AUDIO_PROTOCOL,         /* The interface doesn't use any class-specific protocols  */
    0x00U,                      /* The device doesn't have a string descriptor describing this iInterface  */

    /* Audio Class Specific CS INTERFACE Descriptor*/
    USB_AUDIO_STREAMING_IFACE_DESC_SIZE,               /* Size of the descriptor, in bytes  */
    USB_DESCRIPTOR_TYPE_AUDIO_CS_INTERFACE,            /* CS_INTERFACE Descriptor Type  */
    USB_DESCRIPTOR_SUBTYPE_AUDIO_STREAMING_AS_GENERAL, /* AS_GENERAL descriptor subtype  */
    USB_AUDIO_SPEAKER_CONTROL_INPUT_TERMINAL_ID,       /* The Terminal ID of the Terminal to which the endpoint of this
                                                          interface is connected. */
    0x01U,        /* Delay introduced by the data path. Expressed in number of frames.  */
    0x01U, 0x00U, /* PCM  */

    /* Audio Class Specific type I format INTERFACE Descriptor */
    USB_AUDIO_STREAMING_TYPE_I_DESC_SIZE,              /* bLength (11) */
    USB_DESCRIPTOR_TYPE_AUDIO_CS_INTERFACE,            /* bDescriptorType (CS_INTERFACE) */
    USB_DESCRIPTOR_SUBTYPE_AUDIO_STREAMING_FORMAT_TYPE, /* DescriptorSubtype: AUDIO STREAMING FORMAT TYPE */
    USB_AUDIO_FORMAT_TYPE_I,                           /* Format Type: Type I */
    AUDIO_OUT_FORMAT_CHANNELS_32BIT,                        /* Number of Channels*/
    AUDIO_OUT_FORMAT_SIZE_32BIT, /* SubFrame Size: The number of bytes occupied by one audio subframe. */
    AUDIO_OUT_FORMAT_BITS_32BIT, /* Bit Resolution: 8 bits per sample */
    0x01U,                      /* One frequency supported */
                               /*   0x40, 0x1F,0x00U,                8 kHz */
    // 0x80U, 0x3EU, 0x00U,                          /* 16 kHz */
    TSAMFREQ2BYTES(AUDIO_OUT_SAMPLING_RATE_KHZ * 1000),
/*   0x80,0xBB,0x00U,                48 kHz */
/*   0x00U, 0xFA,0x00U,              72 kHz */

#if defined(USB_DEVICE_AUDIO_USE_SYNC_MODE) && (USB_DEVICE_AUDIO_USE_SYNC_MODE > 0U)
    /* ENDPOINT Descriptor */
    USB_ENDPOINT_AUDIO_DESCRIPTOR_LENGTH, /* Descriptor size is 9 bytes  */
    USB_DESCRIPTOR_TYPE_ENDPOINT,         /* Descriptor type (endpoint descriptor) */
    USB_AUDIO_SPEAKER_STREAM_ENDPOINT |
        (USB_OUT << USB_DESCRIPTOR_ENDPOINT_ADDRESS_DIRECTION_SHIFT), /* OUT endpoint address 1 */
    USB_ENDPOINT_ISOCHRONOUS | 0x0CU,                                 /* Isochronous endpoint and Synchronous*/
    USB_SHORT_GET_LOW(FS_ISO_OUT_ENDP_PACKET_SIZE_32BIT), USB_SHORT_GET_HIGH(FS_ISO_OUT_ENDP_PACKET_SIZE_32BIT), /* 16 bytes
    FS_ISO_OUT_ENDP_INTERVAL, /* bInterval(0x01U): x ms */
    0x00U,                    /* Unused */
    0x00U,                    /* Synchronization Endpoint (if used) is endpoint 0x83  */
#else
    /* ENDPOINT Descriptor */
    USB_ENDPOINT_AUDIO_DESCRIPTOR_LENGTH, /* Descriptor size is 9 bytes  */
    USB_DESCRIPTOR_TYPE_ENDPOINT,         /* Descriptor type (endpoint descriptor) */
    USB_AUDIO_SPEAKER_STREAM_ENDPOINT |
        (USB_OUT << USB_DESCRIPTOR_ENDPOINT_ADDRESS_DIRECTION_SHIFT), /* OUT endpoint address 1 */
    USB_ENDPOINT_ISOCHRONOUS | 0x04,                                  /* Isochronous endpoint */
    USB_SHORT_GET_LOW(FS_ISO_OUT_ENDP_PACKET_SIZE + AUDIO_OUT_FORMAT_CHANNELS * AUDIO_OUT_FORMAT_SIZE),
    USB_SHORT_GET_HIGH(FS_ISO_OUT_ENDP_PACKET_SIZE + AUDIO_OUT_FORMAT_CHANNELS * AUDIO_OUT_FORMAT_SIZE), /* 16 bytes  */
    FS_ISO_OUT_ENDP_INTERVAL, /* bInterval(0x01U): x ms */
    0x00U,                    /* Unused */
    USB_AUDIO_SPEAKER_FEEDBACK_ENDPOINT |
        (USB_IN
         << USB_DESCRIPTOR_ENDPOINT_ADDRESS_DIRECTION_SHIFT), /* Synchronization Endpoint (if used) is endpoint 0x83  */
#endif
    /* Audio Class Specific ENDPOINT Descriptor  */
    USB_AUDIO_STREAMING_ENDP_DESC_SIZE,      /*  Size of the descriptor, in bytes  */
    USB_AUDIO_STREAM_ENDPOINT_DESCRIPTOR,    /* CS_ENDPOINT Descriptor Type  */
    USB_AUDIO_EP_GENERAL_DESCRIPTOR_SUBTYPE, /* AUDIO_EP_GENERAL descriptor subtype  */
    0x00U,                                   /* Bit 0: Sampling Frequency 0
                                                Bit 1: Pitch 0
                                                Bit 7: MaxPacketsOnly 0   */
    0x00U,                                   /* Indicates the units used for the wLockDelay field: 0: Undefined  */
    0x00U, 0x00U, /* Indicates the time it takes this endpoint to reliably lock its internal clock recovery circuitry */

#if defined(USB_DEVICE_AUDIO_USE_SYNC_MODE) && (USB_DEVICE_AUDIO_USE_SYNC_MODE > 0U)
#else
    /* Endpoint 3 Feedback ENDPOINT */
    USB_ENDPOINT_AUDIO_DESCRIPTOR_LENGTH, /* bLength */
    USB_DESCRIPTOR_TYPE_ENDPOINT,         /* bDescriptorType */
    USB_AUDIO_SPEAKER_FEEDBACK_ENDPOINT |
        (USB_IN << USB_DESCRIPTOR_ENDPOINT_ADDRESS_DIRECTION_SHIFT), /* This is an IN endpoint with endpoint number 3 *
    USB_ENDPOINT_ISOCHRONOUS | 0x10,                                 /*  Types -
                                                                         Transfer: ISOCHRONOUS
                                                                         Sync: Async
                                                                         Usage: Feedback EP   */
    FS_ISO_FEEDBACK_ENDP_PACKET_SIZE, 0x00,                          /* wMaxPacketSize */
    FS_ISO_FEEDBACK_ENDP_INTERVAL,                                   /* interval polling(2^x ms) */
    0x05,                                                            /* bRefresh(32ms)  */
    0x00,                                                            /* unused */
#endif

#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_GetPll0OutFreq() / 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

```
void Init_Board_Audio_32bit(void)
{
    usb_echo("Init Audio I2S and CODEC\r\n");

    BOARD_USB_AUDIO_KEYBOARD_Init();

    I2S_USB_Audio_TxInit_32bit(DEMO_I2S_TX);
    I2S_USB_Audio_RxInit(DEMO_I2S_RX);
    wm8904Config.format.bitWidth = kWM8904_BitWidth32;
    BOARD_Codec_Init();
    BOARD_DMA_EDMA_Config();
    BOARD_DMA_EDMA_Start_32bit();
}
```

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

```
void BOARD_DMA_EDMA_Start_32bit()
{
    s_TxTransfer.dataSize = FS_ISO_OUT_ENDP_PACKET_SIZE_32BIT;
    s_TxTransfer.data     = audioPlayDMATempBuff;
    s_RxTransfer.dataSize = FS_ISO_IN_ENDP_PACKET_SIZE;
    s_RxTransfer.data     = audioRecDMATempBuff;

    I2S_TxTransferCreateHandleDMA(DEMO_I2S_TX, &s_TxHandle, &s_DmaTxHandle, TxCallback, (void *)&s_TxTransfer);
    I2S_RxTransferCreateHandleDMA(DEMO_I2S_RX, &s_RxHandle, &s_DmaRxHandle, RxCallback, (void *)&s_RxTransfer);

    /* need to queue two receive buffers so when the first one is filled,
     * the other is immediatelly starts to be filled */
    I2S_RxTransferReceiveDMA(DEMO_I2S_RX, &s_RxHandle, s_RxTransfer);
    I2S_RxTransferReceiveDMA(DEMO_I2S_RX, &s_RxHandle, s_RxTransfer);

    I2S_TxTransferSendDMA(DEMO_I2S_TX, &s_TxHandle, s_TxTransfer);
}
```

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`.

```
                case USB_AUDIO_SPEAKER_STREAM_INTERFACE_INDEX:
                    if (alternateSetting < USB_AUDIO_SPEAKER_STREAM_INTERFACE_ALTERNATE_COUNT)
                    {
#if USB_AUDIO_FORMAT_SWITCH_ENABLE
                        g_USBSpeakerAlternateSettingIndex = alternateSetting;

                        if(g_USBSpeakerAlternateSettingIndex == 1)
                        {
                            g_composite.audioUnified.audioPlayTransferSize = AUDIO_PLAY_BUFFER_SIZE_ONE_FRAME;
                            g_composite.audioUnified.currentStreamOutMaxPacketSize = (FS_ISO_OUT_ENDP_PACKET_SIZE);

                        }
                        else if(g_USBSpeakerAlternateSettingIndex == 2)
                        {
                            g_composite.audioUnified.audioPlayTransferSize = AUDIO_PLAY_BUFFER_SIZE_ONE_FRAME_32BIT;
                            g_composite.audioUnified.currentStreamOutMaxPacketSize = (FS_ISO_OUT_ENDP_PACKET_SIZE_32BIT);
                        }
#endif
                        error = USB_DeviceAudioSpeakerSetInterface(g_composite.audioUnified.audioSpeakerHandle,
                                                                   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.

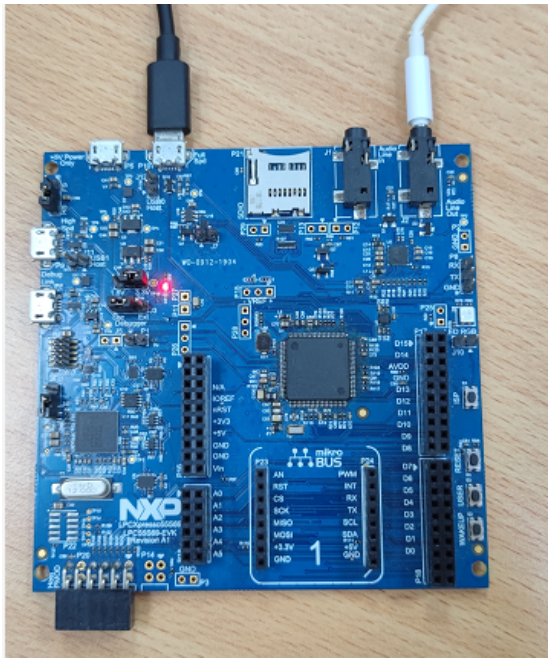


Figure 14. LPC55S69 EVK Board

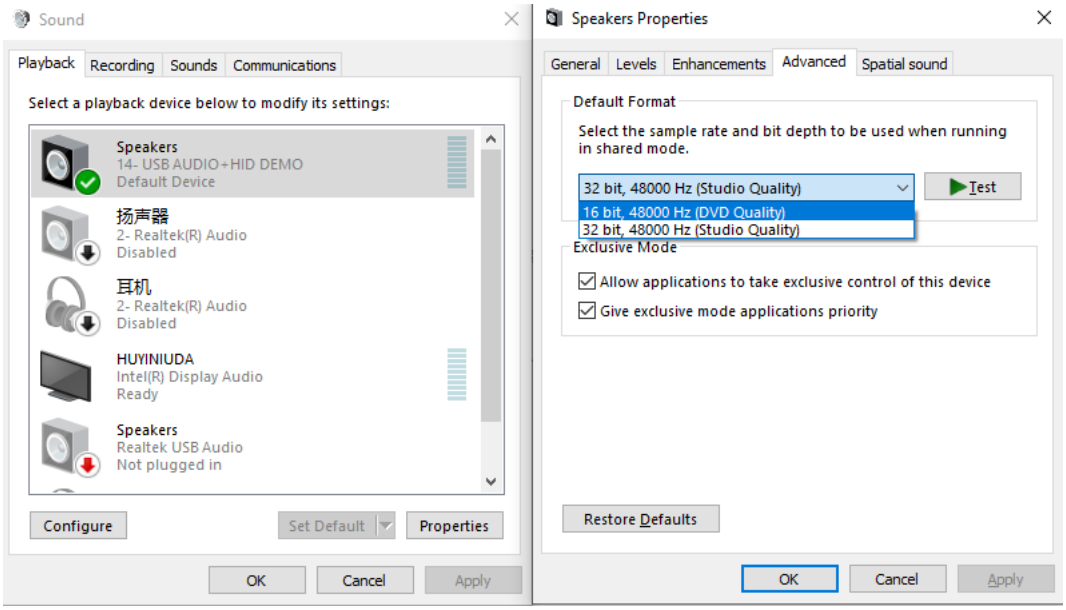
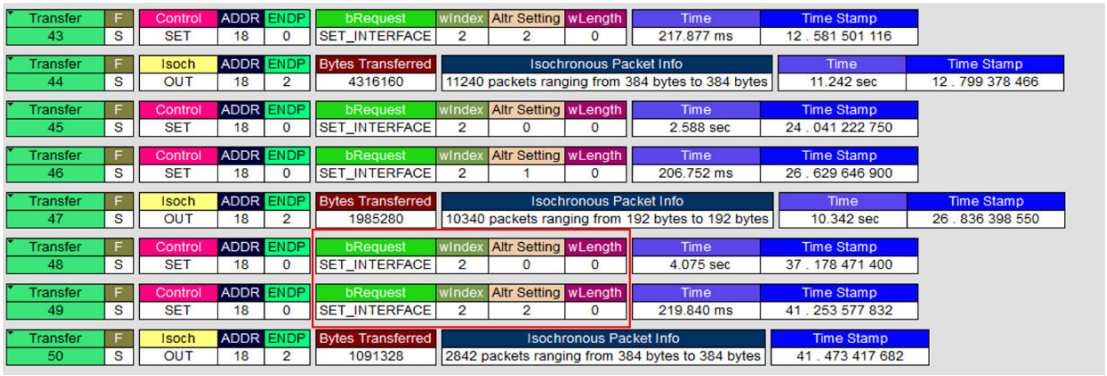


Figure 15. PC recognizes devices that support multiple audio formats

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	Altr 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	Altr 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	Altr 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	Altr 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	Altr 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 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 Altr 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 Altr 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.

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

arm