AN12541

KL27 USB Audio Application

Rev. 0 — 23 July 2019 Application Note

Contents

1 Introduction

Kinetis® KL2x-72/96 MHz, USB Ultra-Low-Power Microcontrollers (MCUs) based on Arm® Cortex®-M0+ Core, with support for one I2S interface and full-speed USB is ideal for USB audio applications.

This application highlights a low power 'plug and play' USB audio device based on KL27. Once set up, the device is plugged into a host system that supports the USB audio class. The audio is played back or recorded through the headphone of the phone and line-in connectors respectively.

NOTE

No special software or drivers is installed on the host systems for this application example.

1.1 System overview

The system contains a Host Controller (KL27) and a CODEC(ALC5658). It is responsible for transferring data from PC-to-Host Controller through USB interface and processing; transferring to CODEC through I2S interface to complete audio playback. Besides the main components above, the system has a speaker, a microphone, and some buttons to change volume or mute.

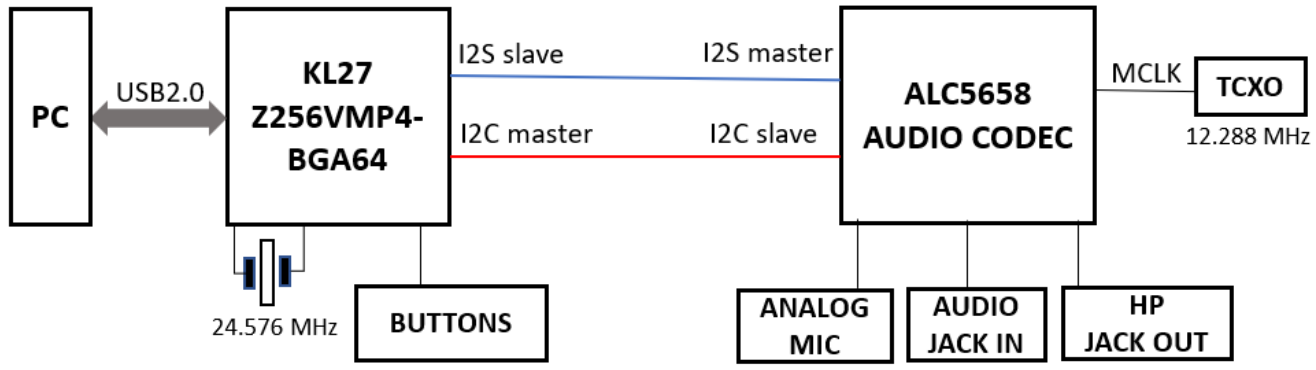


Figure 1. Simplified system overview

1.2 Highlights of the USB audio application

The USB audio application highlights the following KL27 MCU features:

- Core platform clock up to 48 MHz, bus clock up to 24 MHz
- Memory option is up to 256 KB flash and 32 KB RAM
- USB audio class support on the USB full-speed device using the in-ROM USB driver

- I2C interface for CODEC control: I2C1
- I2S transmits and receives interfaces for CODEC audio data: I2S0
- Very low-power usages during USB suspend
- Debug output support on a UART: UART0
- Audio clock synchronization to host clock

2 USB audio application overview

2.1 How the USB audio application works

When the application is started on the KL27 demo board and a USB cable is connected between the board and an audio host system, the USB enumerates as an audio class device. The audio class device has several audio endpoints (Isochronous IN and OUT endpoints) and the control endpoint.

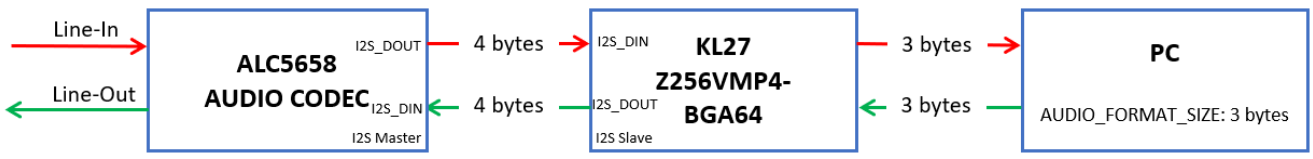


Figure 2. Transfer process of up/down stream

1. Record: The IN endpoint is used for streaming of audio data from the board (the red arrow),
   - The audio is entered through 'LINE IN', now CODEC is master of I2S and one output sample contains 32bits data(1 byte 0x00 added by CODEC in advance and 3 bytes audio data).

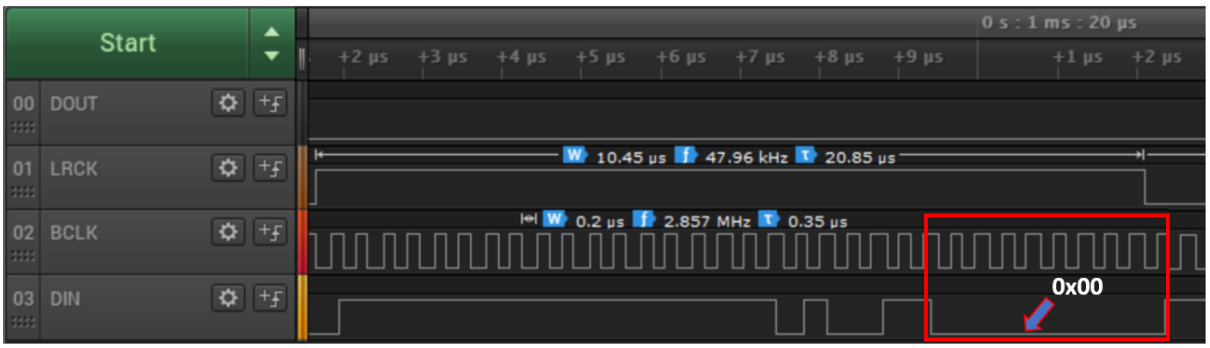


Figure 3. I2S_Master (CODEC) output 4 bytes data containing 1 byte 0x00

   - DMA channel transfers 32 bits data to SAI Receive Data Register (I2Sx_RDRn).
   - User need delete 1 byte 0x00, then can send 3 bytes to PC through USB interface.

NOTE

User can refer to API of SDK: USB_AudioRecorderGetBuffer' (audio_unified.c).

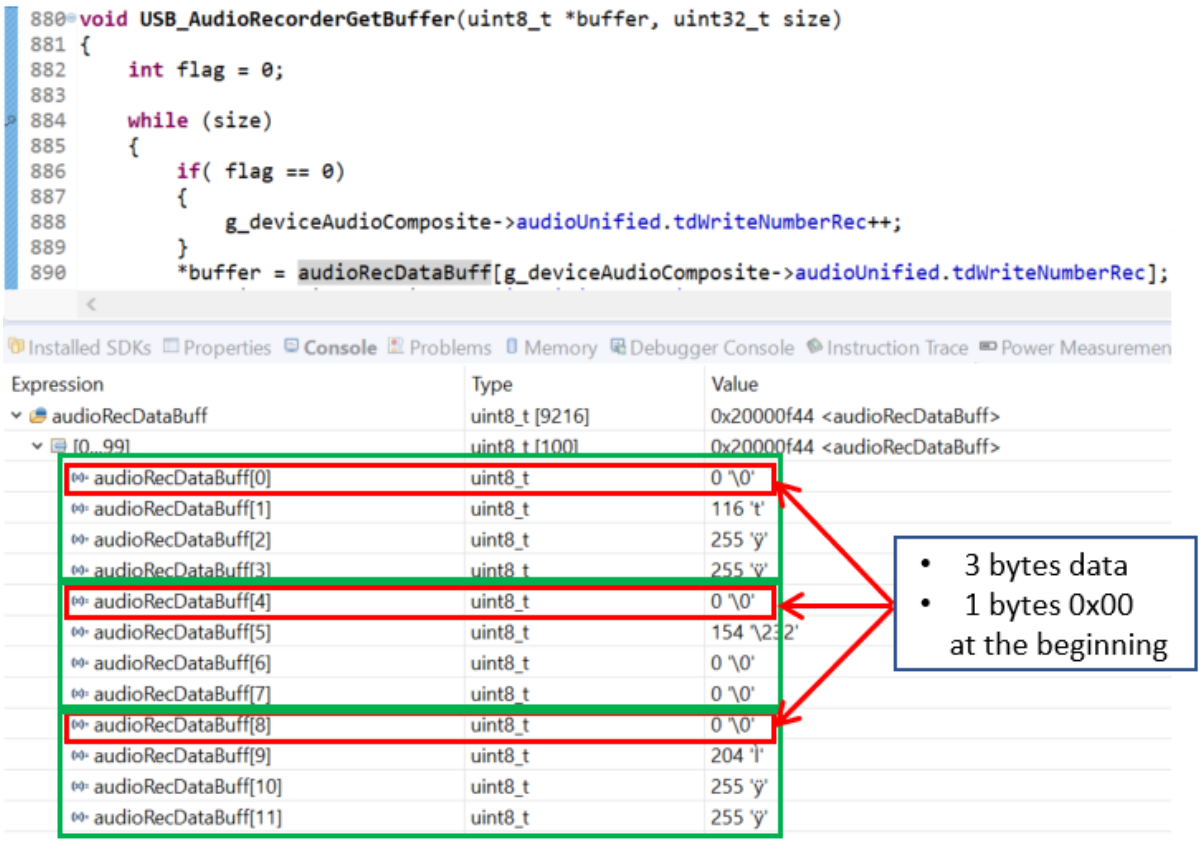


Figure 4. Stream with 0x00 at the beginning

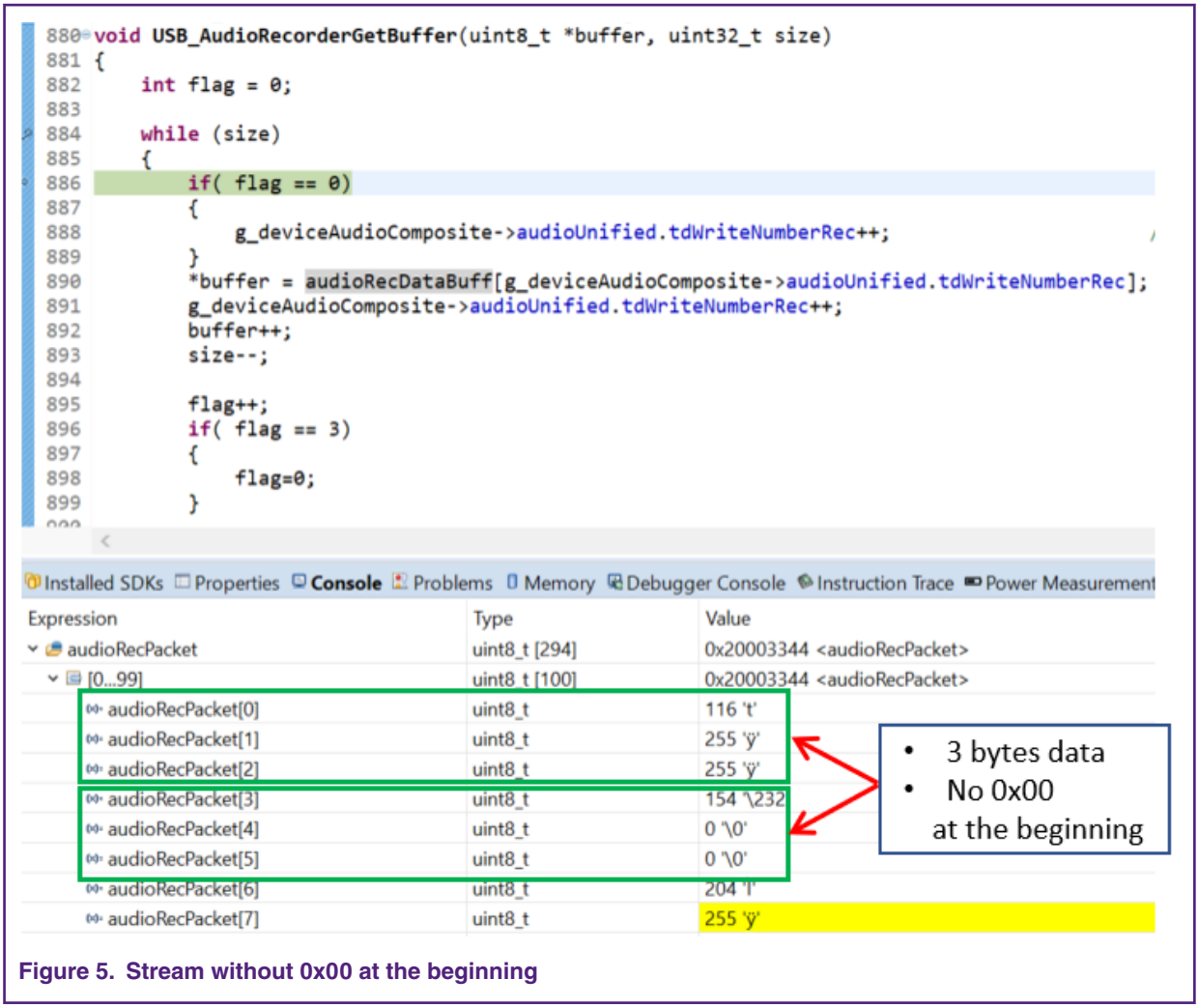


Figure 5. Stream without 0x00 at the beginning

2. Playback: the OUT endpoint is used for streaming of audio data to the board (the green arrow).
   - PC send 3 bytes to KL27 through USB interface, considering I2S was configured to transfer 4 bytes data per sample, so user need add 1 byte 0x00 in front of every 3 bytes data in USB_AudioSpeakerPutBuffer.

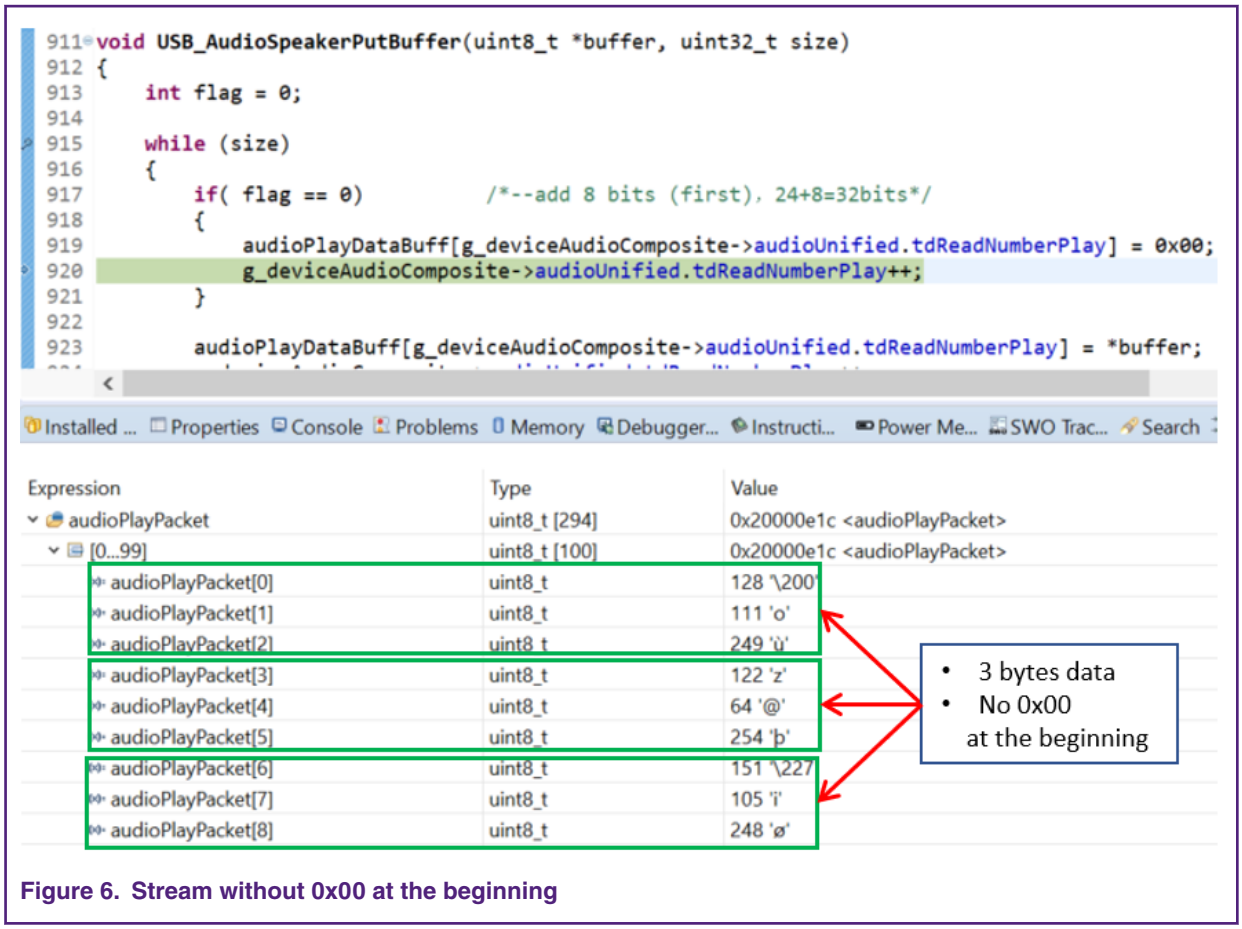


Figure 6. Stream without 0x00 at the beginning

```
911 void USB_AudioSpeakerPutBuffer(uint8_t *buffer, uint32_t size)
912 {
913     int flag = 0;
914
915     while (size)
916     {
917         if( flag == 0)          /*--add 8 bits (first), 24+8=32bits*/
918         {
919             audioPlayDataBuff[g_deviceAudioComposite->audioUnified.tdReadNumberPlay] = 0x00;
920             g_deviceAudioComposite->audioUnified.tdReadNumberPlay++;
921         }
922
923         audioPlayDataBuff[g_deviceAudioComposite->audioUnified.tdReadNumberPlay] = *buffer;
```

Installed ... | Properties | Console | Problems | Memory | Debugge... | Instructi... | Power M... | SWO Trac... | Search

Expression	Type	Value
audioPlayDataBuff[768]	uint8_t	0 '\0'
audioPlayDataBuff[769]	uint8_t	128 '\200'
audioPlayDataBuff[770]	uint8_t	111 'o'
audioPlayDataBuff[771]	uint8_t	249 'ù'
audioPlayDataBuff[772]	uint8_t	0 '\0'
audioPlayDataBuff[773]	uint8_t	122 'z'
audioPlayDataBuff[774]	uint8_t	64 '@'
audioPlayDataBuff[775]	uint8_t	254 'þ'
audioPlayDataBuff[776]	uint8_t	0 '\0'
audioPlayDataBuff[777]	uint8_t	151 '\227'
audioPlayDataBuff[778]	uint8_t	105 'i'
audioPlayDataBuff[779]	uint8_t	248 'ø'

- 3 bytes data
- 1 bytes 0x00 at the beginning

Figure 7. Stream with 0x00 at the beginning

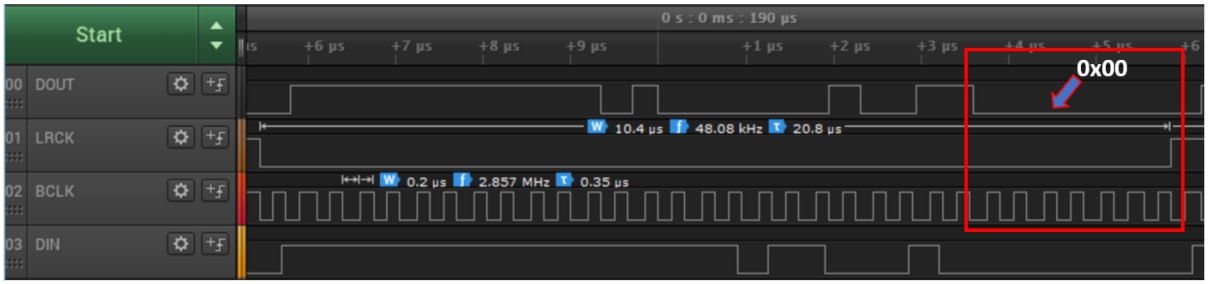


Figure 8. I2S_Slave (KL27) output 4 bytes data containing 1 byte 0x00

- Then transfer modified 4 bytes data to SAI Transmit Data Register (I2Sx_TDRn) using DMA channel.

2.1.1 SAI and CODEC

- In SAI configuration, data length of audio data is equal to 32, it means that one sample contains 32 bits data.
- In CODEC configuration, valid data length of audio data is equal to 24.

1. For audio playback, The I2S transmit interrupt is used to provide audio playback data to the I2S transmit block. The interrupt triggers when the I2S transmit FIFO is half empty. The interrupt moves data from the audio playback buffer to the I2S FIFO. The I2S block outputs data to the CODEC for playback based on the audio clock rate.
2. For audio record, the I2S block reads data from the CODEC based on the audio clock rate. The I2S receive interrupt triggers when the I2S receive FIFO is half full. The interrupt moves data from the I2S FIFO to the audio record data buffer.

2.1.2 Audio sample clocks

The main purpose of this document is to provide a solution for 48 KHz/24 bits up/down stream, this section lists all related clock information.

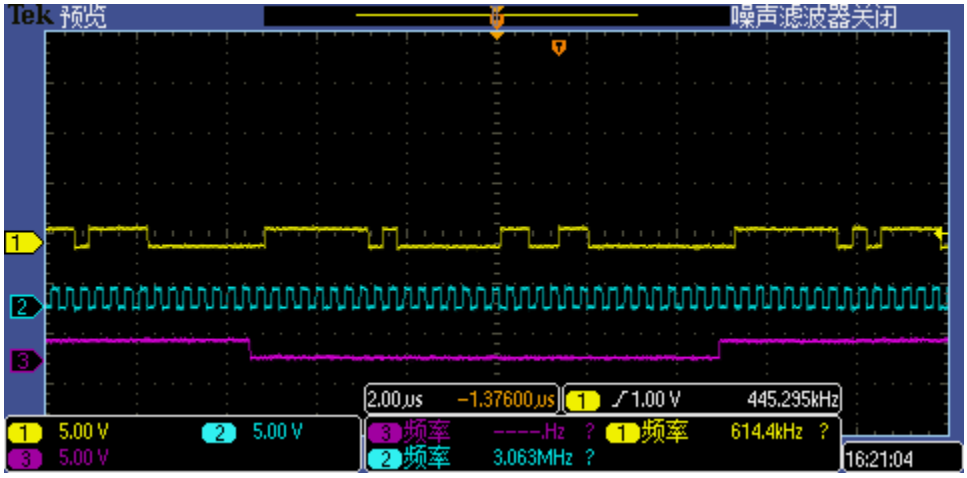


Figure 9. Sample clocks information

In #unique_9,

Channel 2: BCLK / SCLK

The I2S/SAI transmitter and receiver support asynchronous bit clocks (BCLKs) that can be generated internally from the audio master clock or supplied externally.

- if the clock division is 16, BLCK = MCLK / 16 = 24.567 MHz / 16 = 1.5035 MHz.

This value corresponds to a sample rate of 48 KHz, 2 channels (stereo) and 2 bytes (16 bits) per sample.

- if the clock division is 8, BLCK = MCLK / 8 = 24.567 MHz / 8 = 3.0709 MHz.

This value corresponds to a sample rate of 48 KHz, 2 channels (stereo) and 4 bytes (32 bits) per sample.

Channel 3: LRCK/WS

Frame clock WS(word select)/LRCK(left-right clock) , switch the data of left and right channels.

When LRCK is "1", it means that the data being transmitted is the data of right channel.

when LRCK is "0", it means that the data being transmitted is the data of left channel.

The frequency of LRCK is equal to the sampling frequency.

- The Master Clock (24.567 MHz) can be obtained by using PLL or come from an external crystal.

Actually, considering that the SDK already provides some 48khz/16bits demos, this document mainly described how to configure KL27 for 48 KHz/24bits up/down stream. In the demo, setting 48 KHz sample rate on LRCK/WS with result in 3.07 MHz BCLK/SCLK (2 * 32 bits/word * 48000 Hz).

2.2 Hardware overview

- Pin setup
- Peripheral functions

2.2.1 Pin setup

All used pins are set up to their MUX state as needed by the system. This includes I2C Interface, I2S Interface, UART Interface, USB, and the audio clock output. Table 1 shows the pins that are used by this application.

Table 1. USB Audio application pin mapping

Interface	Pin name	Pin function
UART	PTE20/UART0_TX	UART receive
	PTE21/UART0_RX	UART transmit
I2C	PTC10/I2C1_SCL	I2C clock pin to CODEC
	PTC11/I2C1_SDA	I2C data pin to CODEC
I2S transmit	PTC0/ AUDIOUSB_SOF_OUT	I2S transmit data (to CODEC)
	PTC2/(WS/FS/LRCK)	I2S transmit frame (to CODEC)
	PTC3/(BCLK/SCLK)	I2S transmit clock (to CODEC)
I2S receive	PTC5/(AUDIOUSB_SOF_IN)	I2S receives data (from CODEC)
	PTC7/(WS/FS/LRCK)	I2S receives frame (to CODEC)
	-	I2S receives clock (to CODEC)
Audio clock	-	USB D+ Audio MCLK (to CODEC)
USB	USB_DP	USB D+
	USB_DN	USB D-

2.2.2 Peripheral functions

Table 2 lists the function of each peripheral or interface used by the board or USB Audio application.

Table 2. Peripherals and interfaces mapped to application function

Peripheral or interface	USB Audio application function
UART	Used for status messages with the DEBUGOUT() macro. Outputs text strings to a console connected to the debugger.
I2C	The I2C interface is connected to the audio CODEC. It is used for CODEC configuration and setup, volume control, and CODEC status.
I2S	The I2S interface handles the serial data to and from the audio CODEC. Two I2S blocks are used – one for playback and one for record.
Audio output	The 'Audio HP/Line-Out' plug on the MIC/Audio/OLED shield board is used as the analog audio output from the CODEC. This can be connected to a pair of headphones or the line-in of another system.
Audio input	The 'Audio Line-In' plug on the MIC/Audio/OLED shield board is used as the analog audio input from a Line-in source. It is connected to the audio CODEC.

3 Software based on SDK 48 KHz/16 bits

This section describes how to modify relevant API based on SDK to obtain a demo for 48 KHz/16 bits up/down stream.

3.1 Playback section

The following section assumes that the user has acquired a 48 KHz WS/FS/LRCK.

3.1.1 Usb_device_descriptor.h

```
#define AUDIO_FORMAT_BITS (24)
#define AUDIO_FORMAT_SIZE (3)
#define FS_ISO_OUT_ENDP_PACKET_SIZE_DMA (384)
```

3.1.2 USB_AudioSpeakerPutBuffer

The data sent from PC using USB cable is modified as below.

```
void USB_AudioSpeakerPutBuffer(uint8_t *buffer, uint32_t size)
{
    int flag = 0;

    while (size)
    {
        if( flag == 0)
        {
            audioPlayDataBuff[g_deviceAudioComposite->audioUnified.tdReadNumberPlay] = 0x00;
            g_deviceAudioComposite->audioUnified.tdReadNumberPlay++;
        }
        audioPlayDataBuff[g_deviceAudioComposite->audioUnified.tdReadNumberPlay] = *buffer;
        g_deviceAudioComposite->audioUnified.tdReadNumberPlay++;
        buffer++;
        size--;

        flag++;
        if( flag == 3)
        {
            flag=0;
        }

        if (g_deviceAudioComposite->audioUnified.tdReadNumberPlay >=
            AUDIO_SPEAKER_DATA_WHOLE_BUFFER_LENGTH * FS_ISO_OUT_ENDP_PACKET_SIZE)
        {
            g_deviceAudioComposite->audioUnified.tdReadNumberPlay = 0;
        }
```

Add 1 byte 0x00 in front of the next 3 bytes

Figure 10. Modification of USB_AudioSpeakerPutBuffer

Add 1 byte 0x00 at the beginning of each 3 bytes data, then can obtain 4 bytes data as sample. Table 3 provides introduction about 24 / 32 bits data as below,

Table 3. Audio data introduction

24bits	0x00, 0x00, 0x00, 0x15, 0xB5, 0x10,
32bits	***0x00***, 0x00, 0x00, 0x00, ***0x00***, 0x15, 0xB5, 0x10,

For example '0x15, 0xB5, 0x10',

- 0x00 was added at the beginning of each 3 bytes, the modified data buffer contains 0x00, 0x15, 0xB5, 0x10.
- Actual order of play is 0x10, 0xB5, 0x15, 0x00.

3.1.3 Codec

- LRCK / WS
- Configuration table
- BitWidth

3.1.3.1 LRCK / WS

Configure SAI interface based on SDK demo to make sure WS is equal to 48 KHz.

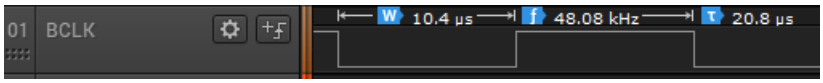


Figure 11. Frequency of LRCK / WS

3.1.3.2 Configuration table

This demo provides two tables that can be used as 16 bits and 24 bits repectively. For more configuration information, view the function in audio_unified.c as below:

```
ALC5658_PowerOnCMDtable[RT5658_command_word_length][5]
```

3.1.3.3 BitWidth

User can configure the data length of audio data according to their actual needs, the function is 'SGTL_Config_Audio_Formats'.

- audioFormat.bitWidth = 32;

The length of audio data is 32 bits, for example, user need to add 1 byte 0x00 to obtain 32 bits data if the length of original data sent from PC is 24 bits.

- audioFormat.bitWidth = 16;

The length of audio data is 16 bits, for example, user need to delete 1 byte 0x00 to obtain 16 bits data if the length of original data sent from PC is 24 bits.

3.1.4 DMA

- DMA Control Register (DMA_DCRn)
- DMA-related 'Transfer size'

3.1.4.1 DMA Control Register (DMA_DCRn)

In DMA_DCRn, SSIZE (bit 21-20) represents 'Source Size', it determines the data size of the source bus cycle for the DMA controller, including 32 bits (00), 8 bits (01), 16 bits (10), and reserved value.

For 48 KHz/24 bits up/down stream requirement, a sample contains 24 bits data, user could obtain 32 bits data by adding 1 byte 0x00 or acquire 16 bits data by deleting 1 bytes data.

For more information, see 'KL27 Sub-Family Reference Manual'.

3.1.4.2 DMA-related ‘Transfer size’

Change the DMA-related ‘Transfer size’ to 384.

```
(composite.c) void BOARD_DMA_EDMA_Start()
(composite.c) static void txCallback(I2S_Type *base, sai_dma_handle_t *handle, status_t status, void
*userData)
(composite.c) static void rxCallback(I2S_Type *base, sai_dma_handle_t *handle, status_t status, void
*userData)
```

3.1.5 MSB First

To configure bit 4 ‘MF’ of ‘SAI Transmit Configuration 4 Register’ (as the MSB is transmitted first):

```
MSB( base->TCR4 = I2S_TCR4_MF(1U) | I2S_TCR4_SYWD )
```

This document provides a brief introduction on how to configure bit 4 'MF'. For example, a sample contains 0x00, 0x01, 0x02 , 0x03.

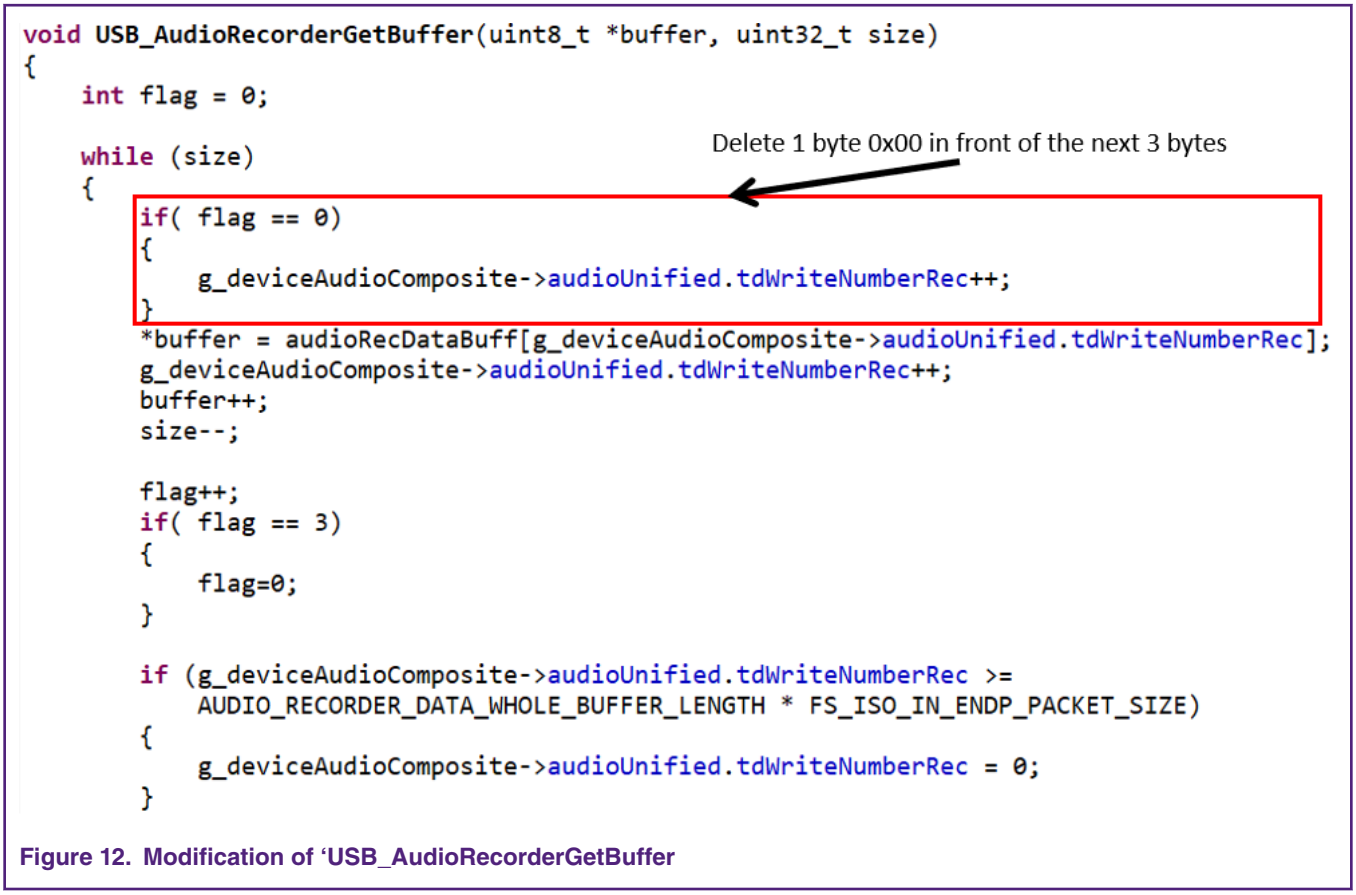
- Store process : software saves 0x00 first, so the sample buffer[4]={ 0x00, 0x01,0x02,0x03 };
- Transmit process: software trnsfers 32 bits to I2Sx_TDRn, actually CODEC view 24 bit (0x03, 0x02, 0x01) as valid data.

3.2 Record section

- USB_AudioRecorderGetBuffer

3.2.1 USB_AudioRecorderGetBuffer

Figure 12 shows how to modify the data sent from CODEC.




```
void USB_AudioRecorderGetBuffer(uint8_t *buffer, uint32_t size)
{
    int flag = 0;

    while (size)
    {
        if( flag == 0)
        {
            g_deviceAudioComposite->audioUnified.tdWriteNumberRec++;
        }
        *buffer = audioRecDataBuff[g_deviceAudioComposite->audioUnified.tdWriteNumberRec];
        g_deviceAudioComposite->audioUnified.tdWriteNumberRec++;
        buffer++;
        size--;

        flag++;
        if( flag == 3)
        {
            flag=0;
        }

        if (g_deviceAudioComposite->audioUnified.tdWriteNumberRec >=
            AUDIO_RECORDER_DATA_WHOLE_BUFFER_LENGTH * FS_ISO_IN_ENDP_PACKET_SIZE)
        {
            g_deviceAudioComposite->audioUnified.tdWriteNumberRec = 0;
        }
```

Figure 12. Modification of ‘USB_AudioRecorderGetBuffer

From the code above see, delete 1 byte 0x00 in front of next 3 bytes data, then can obtain 3 bytes data as new sample and send it to PC through USB interface. For details on setting information, see Playback section.

4 Firmware development

This section provides an overview on the different steps to set up the MCUXpresso environment. Table 4 describes the materials required to perform the steps described in this chapter.

Table 4. Materials list

List	Description
PC	Host device connected to the development board
Debugger	J-Link
IDE	The MCUXpresso IDE
SDK	SDK_2.5.0_MKL27Z256xxx4.zip
Demo	USB_Audio_96KHz_24bit_Playback_Record

MCUXpresso is available for download on the NXP website:

https://www.nxp.com/support/developer-resources/software-development-tools/mcuxpresso-software-and-tools/mcuxpresso-integrated-development-environment-ide:MCUXpresso-IDE

The following steps assume that the user has installed the IDE.

Initiating MCUXpresso prompts you to create a workspace.

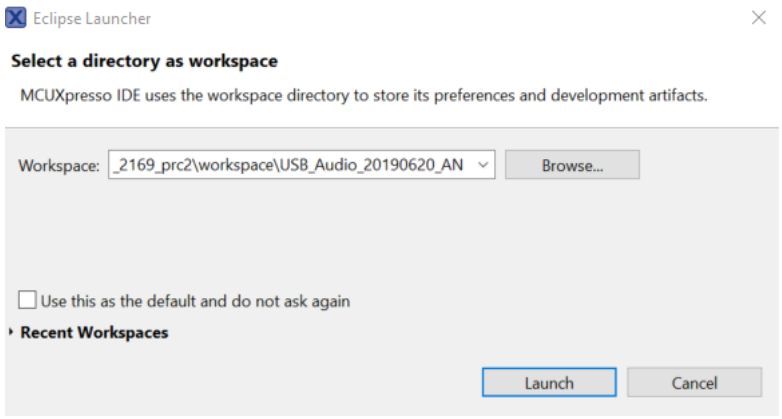


Figure 13. MCUXpresso workspace prompt

The location of this workspace can be chosen as the default value or as preferred. If you do not intend to switch workspaces, you can check the box to use this workspace as default. After choosing a workspace, you should see an empty workspace.

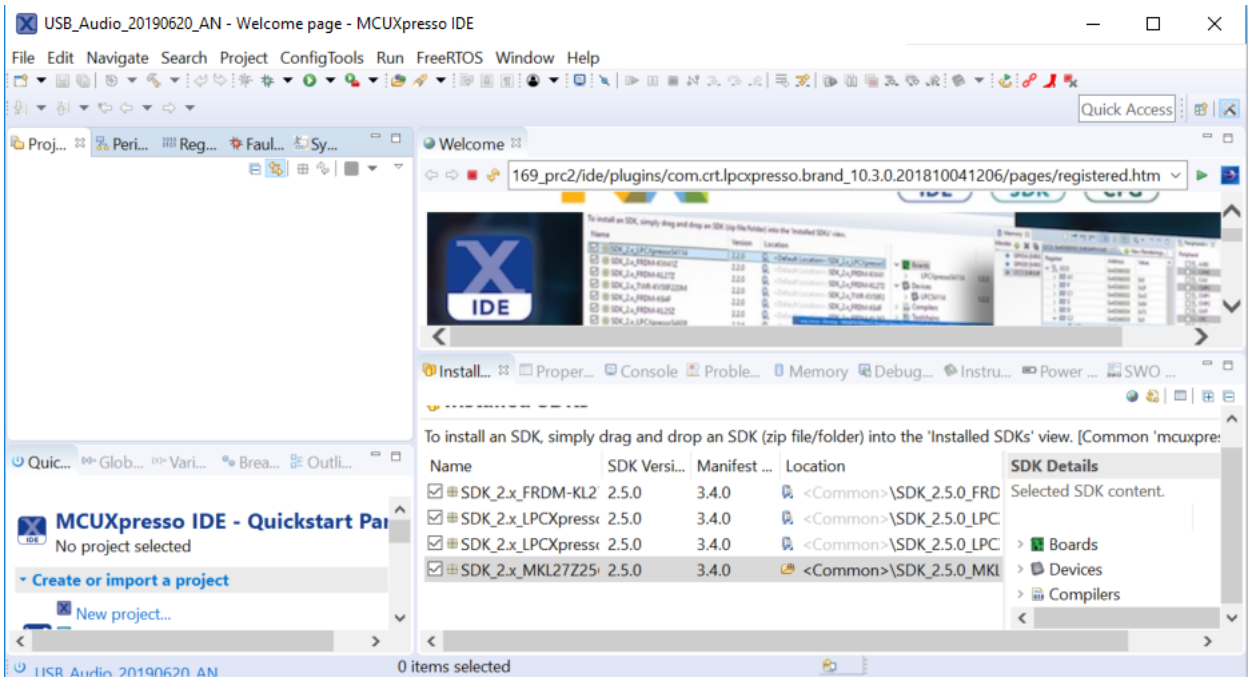


Figure 14. MCUXpresso empty workspace

To import a project, follow these steps:

1. From the **Quickstart** panel, select **Import project(s) from file system**.

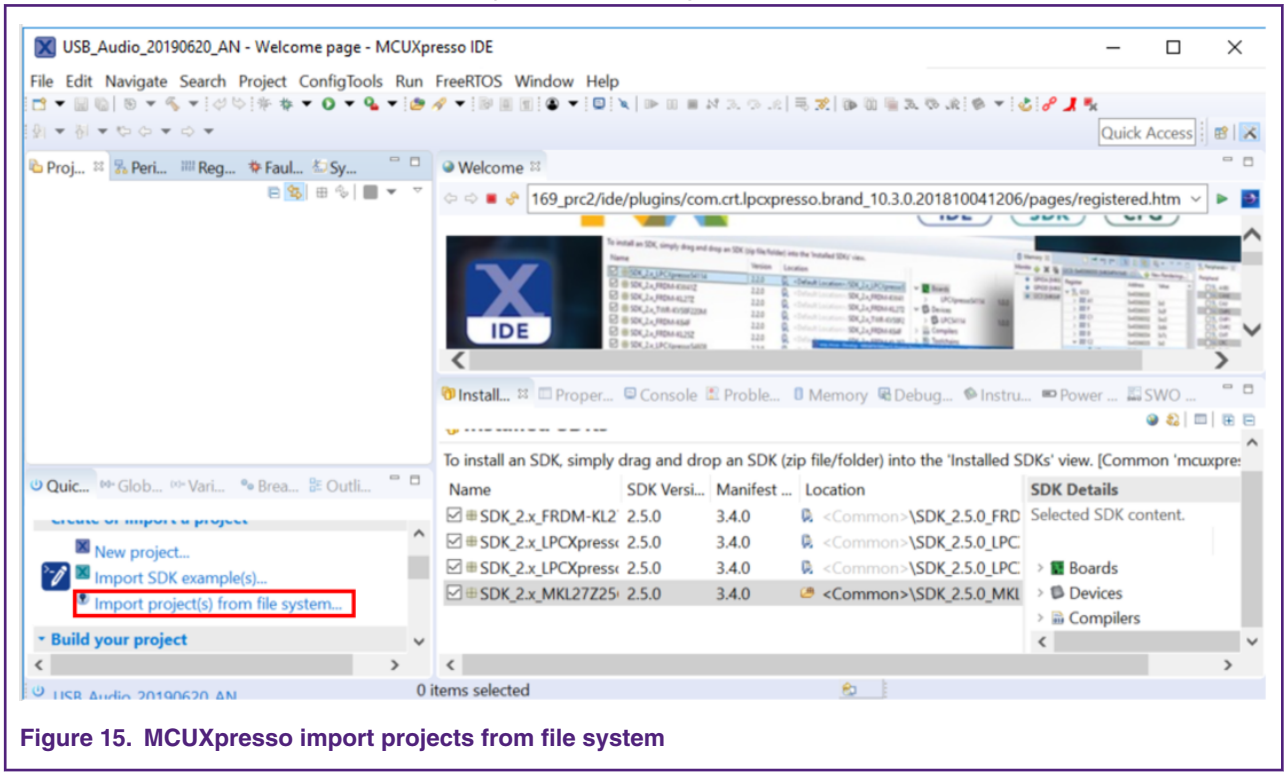


Figure 15. MCUXpresso import projects from file system

2. Fill in the path to the NxH3670 Kinetis apps folder under **Project directory (unpacked)**.

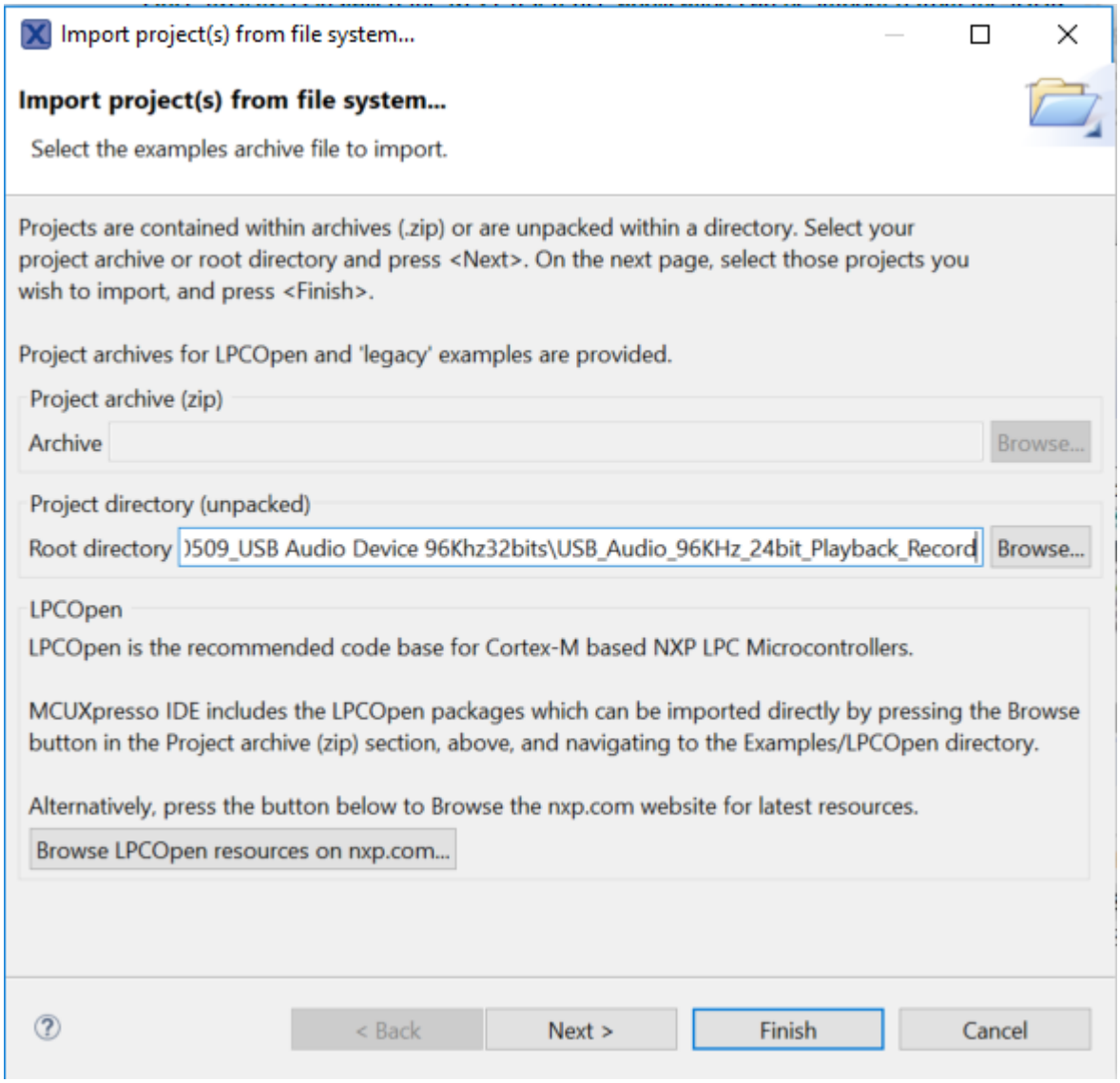


Figure 16. MCUXpresso import project directory

The projects should now appear in a list.

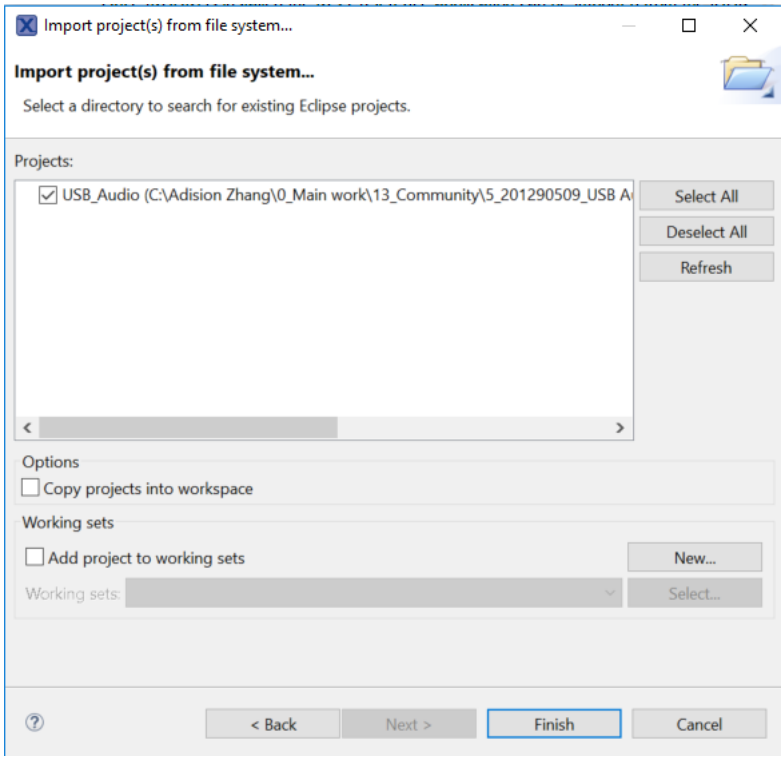


Figure 17. MCUXpresso projects list

3. Select the projects you want to build.
4. Ensure to clear the **Copy projects into workspace** checkbox.
5. Click the **Finish** button.

   You can now build the copied projects.

To build the projects:

1. Select the project you want to build in the “Project Explorer” tab.
2. Select the configuration you want to build from the dropdown menu (next to the hammer-symbol).

   The project is built for the selected configuration.

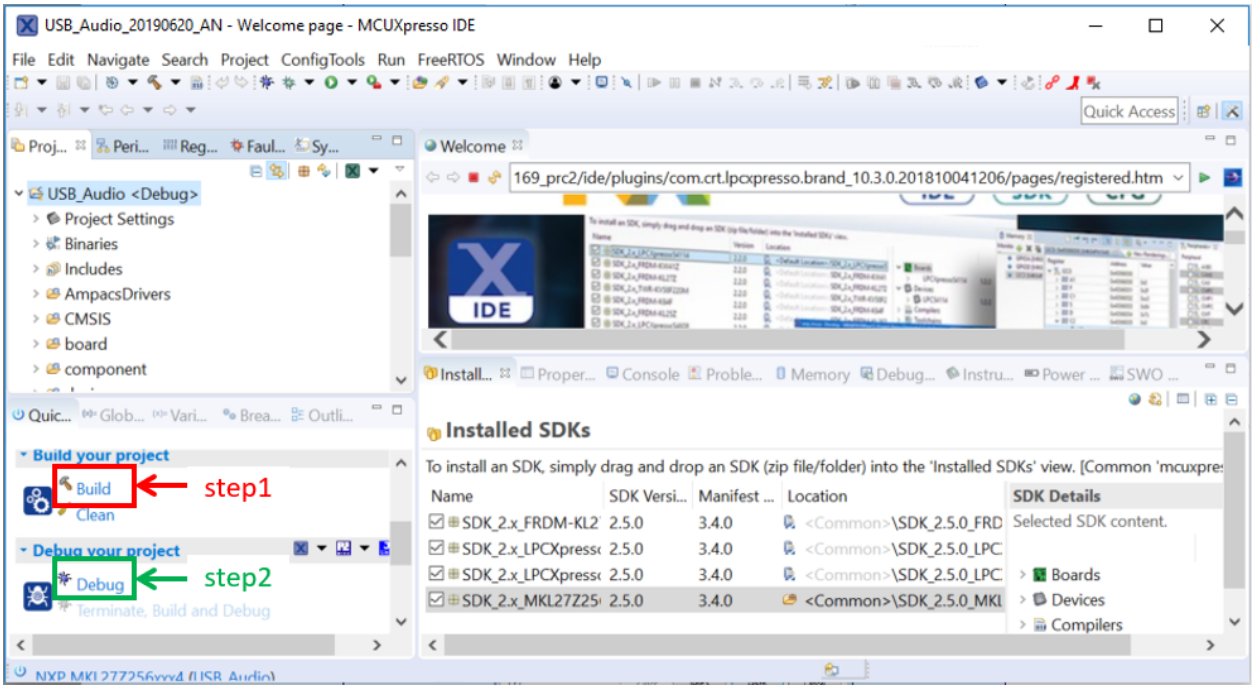


Figure 18. MCUXpresso building and debugging a reference project

5 Playing and recording audio

After downloading the demo code, user can press the ‘RESET’ button of the board. The USB simultaneously enumerates the board as the audio class device, including playback device, and the record device.

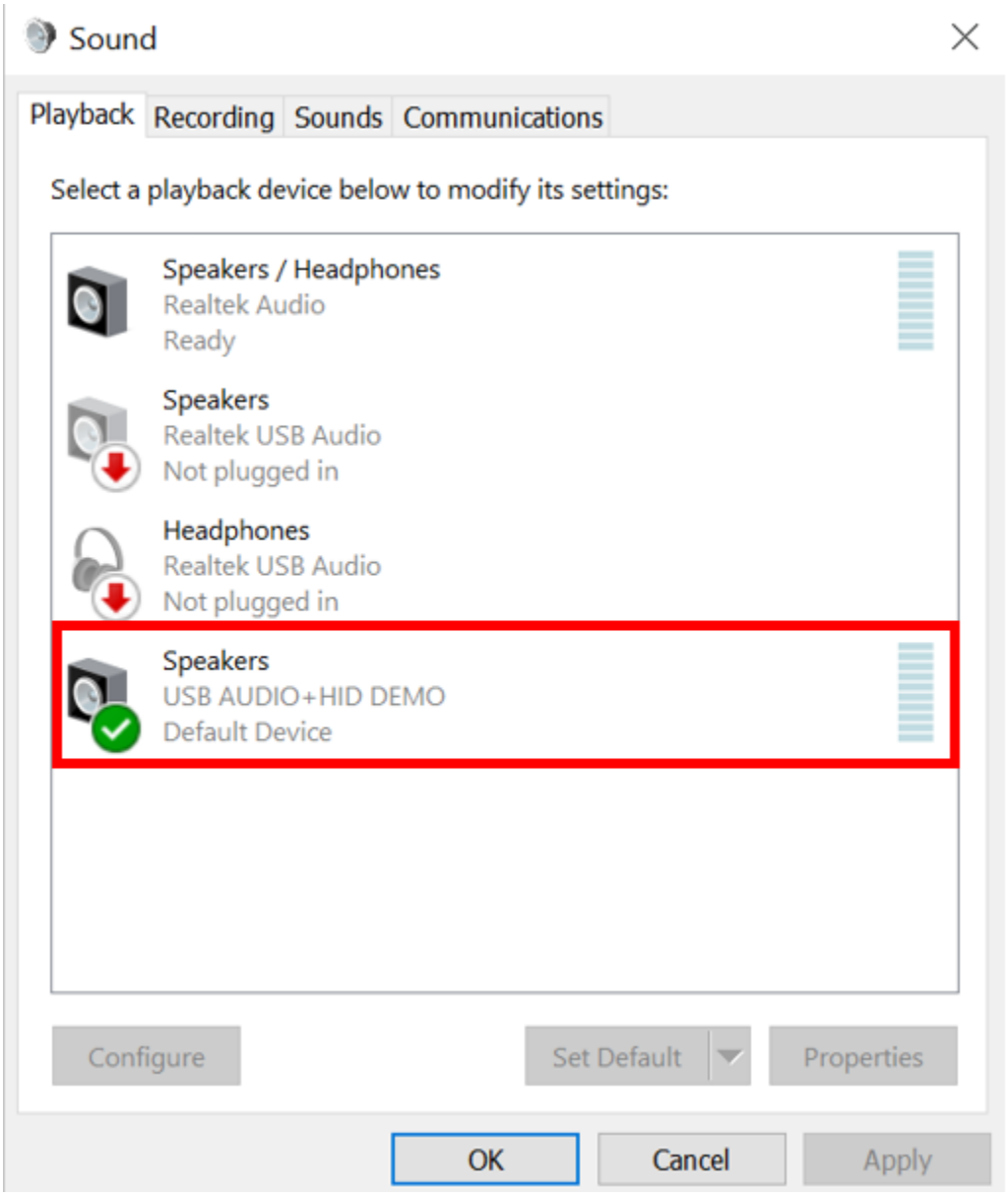


Figure 19. Playback device

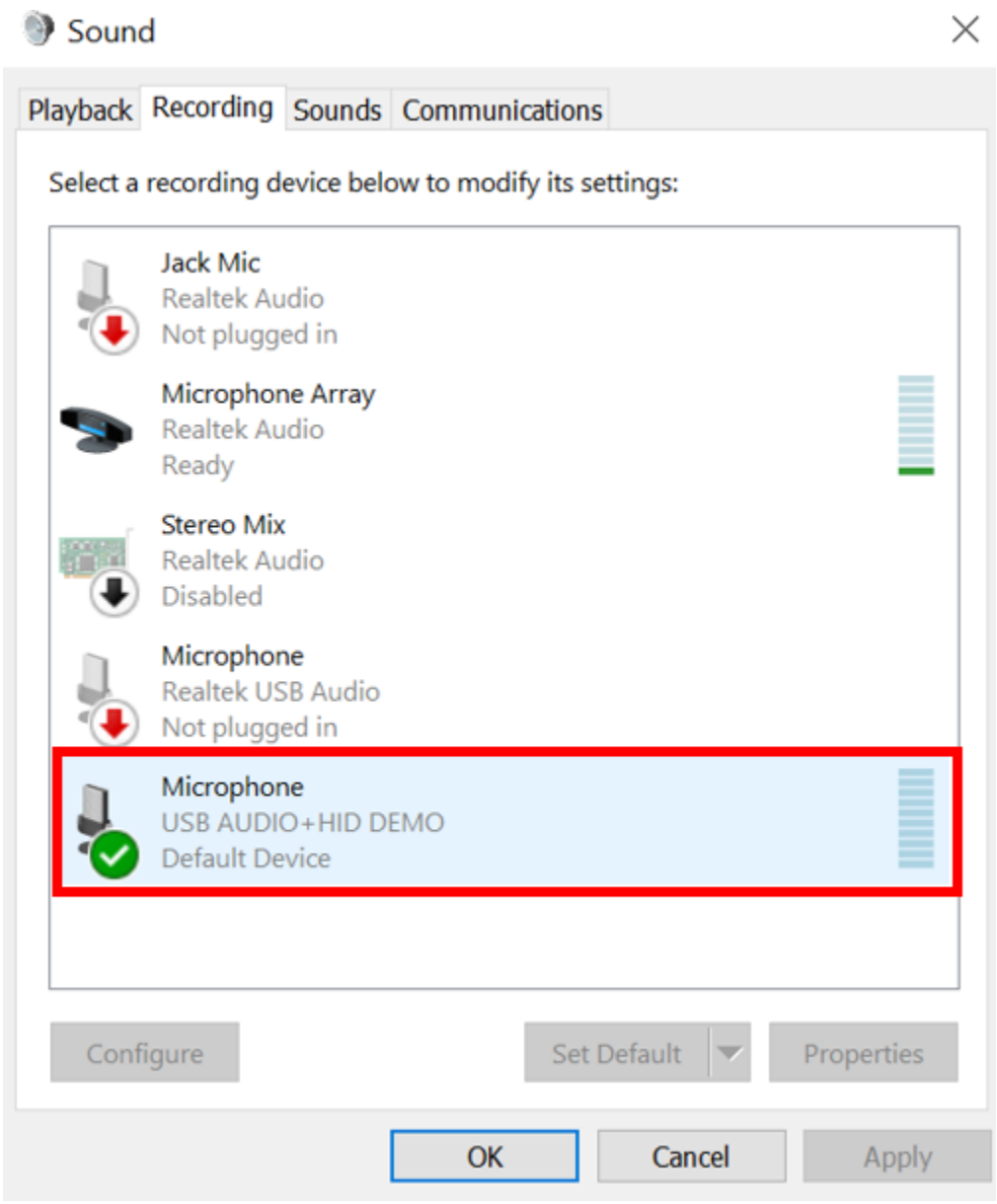


Figure 20. Record device

In this document, the measurement of the main signal using oscilloscope are provided. You can use it as a reference to debug your demo project.

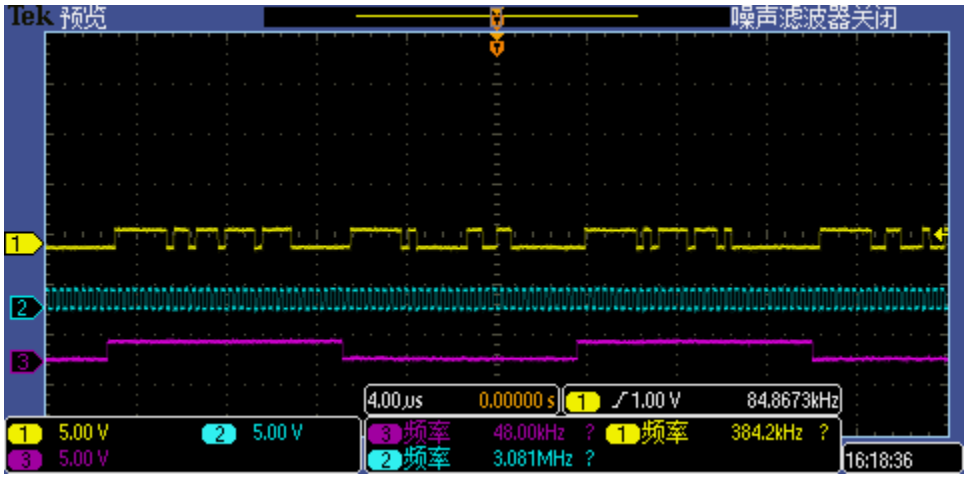


Figure 21. Measurement of the main signal using oscilloscope

6 Conclusion

This document described how to configure KL27 for 48 KHz/24 bits up/down stream. For more information, download and run the demo.

7 Revision history

This table summarizes changes to this document.

Table 5. Revision history

Rev	Date	Description
0	23 July 2019	Initial version

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

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

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

Date of release: 23 July 2019

Document identifier: AN12541

arm