AN13364

USB Audio Synchronous Mode on KL27

Rev. 0 — 26 August 2021 Application Note

Contents

1 Introduction

When developing USB audio devices, one must consider audio synchronization. Three synchronization mechanisms for audio transmission are defined in the USB specification, listed in Table 1.

Table 1. Synchronization characteristics of USB audio device

	Source	Sink
Asynchronous	Free running F_s. Provides implicit feedforward (data stream).	Free running F_s. Provides explicit feedback (isochronous pipe).
Synchronous	F_s locked to SOF. Uses implicit feedback (SOF).	F_s locked to SOF. Uses implicit feedback (SOF).
Adaptive	F_s locked to sink. Uses explicit feedback (isochronous pipe).	F_s locked to data flow. Uses implicit feedforward (data stream).

Both Source and Sink in Table 1 are USB devices. Source is the device that generates audio data and sends it to the USB host, such as a USB microphone. Sink is the device that receives audio data from the USB host, such as a USB speaker. This application note describes synchronization in the USB speaker device. Therefore, all the USB devices below are Sink devices, unless otherwise specified.

The principles of the three synchronization mechanisms are:

- **Asynchronous mode**

  When the USB endpoint is working in an asynchronous mode, the USB device is not synchronized with the USB host's Start of Frame (SOF) signal. However, an additional feedback endpoint provides the true transmission rate of the device to the USB Host. The host then adjusts its transmission rate according to the feedback value.

- **Synchronous mode**

  When the USB endpoint is working in a synchronous mode, the transmission rate of the USB device must synchronize with the SOF signal of the USB host. The transmission rate of the device is synchronized with the SOF signal by adjusting the clock system of the device.

- **Adaptive mode**

  When the USB endpoint is working in an adaptive mode, it can work at any rate within the operating range and not just on the 32 KHz, 44.1 KHz, and 48 KHz frequencies. For sink devices, the device must adjust to its own transmission rate and match the data flow of the USB host.

For details about the synchronization mechanism, see chapter 5.12 of the USB 2.0 specification.

In the USB audio speaker example within the NXP software development kit (SDK), the asynchronous mode is used to realize the audio synchronization between the USB host and the device. The device provides the real transmission rate of the device to the host through the feedback endpoint, such that the host can adjust the actual transmission rate.

To meet the requirement of some customers, some devices in the NXP SDK already support the synchronous mode. For example, LPC54608 and RT600. The synchronous mode is implemented as follows:

- **LPC54608**

  In the example of LPC54608 SDK USB audio speaker, the program adjusts the trim value of the FRO(Free Running Oscillator) clock according to the margin of the ring buffer used to store audio data. Since the FRO clock is used as the input clock of the PLL(Phase locked loop) module, the I2S_BCLK and I2S_WS generated by the PLL division is also updated synchronously.

- **RT600**

  In the example of RT600 SDK USB audio speaker, the USB host SOF frame interval is measured in real time through SCTimer, and the fractional frequency division coefficient of PLL is adjusted according to the SOF frame interval, thereby adjusting the I2S_BCLK and I2S_WS.

Kinetis L (KL) series MCU is a Cortex®-M0+-based low-power and low-cost microcontroller with a USB device controller. Some customers may prefer to use KL series microcontrollers as the USB audio speaker controllers, such as KL27. However, if you want to make the USB audio speaker support PS4, which does not support asynchronous mode, the KL27 must work in synchronous mode.

Different from LPC54608 and RT600, KL27 is a low-cost microcontroller without the PLL module and the fractional divider, which makes KL27 unable to adjust I2S_BCLK and I2S_WS by adjusting PLL like LPC54608 and RT600. This application note introduces an implementation method of the audio synchronization mode based on the KL series MCU. To match the transmission rate of the USB host, you can adjust the integer divider of I2S_BCLK and the I2S word length to adjust the transmission rate of the device.

2 Implementation

This section lists the steps to implement audio synchronous mode in USB audio speaker. Here, KL27 is used as an example.

2.1 Required software and hardware

- **Hardware**

  The KL27 on FRDM-KL27 board is MKL27Z64VLH4 and this part does not have I2S peripherals. Additionally, there is no Codec on this board. Therefore, the USB audio speaker example cannot run on this board. The NxH3670 SDK board is an evaluation board based on the KL27 for evaluating wireless headset solution. There are MKL27Z256VMP4 and Codec WM8904 on this board, so this board is selected as the hardware platform for testing. The board configuration is shown in Figure 1. For more detailed about the board, please refer to document UM11150.

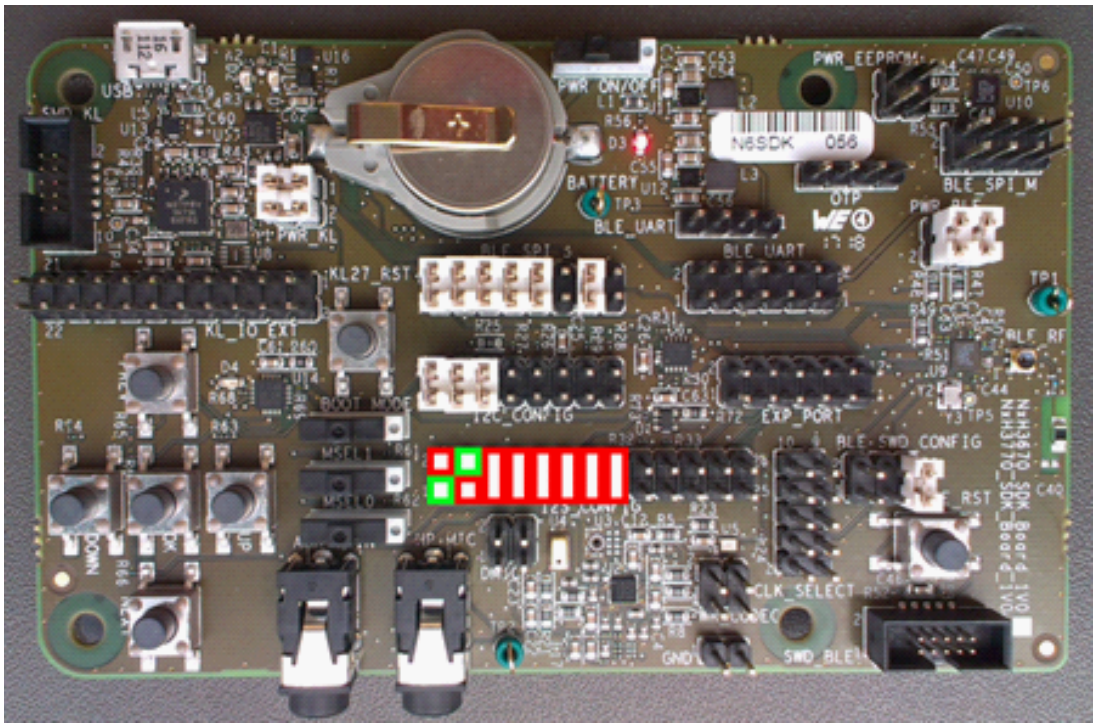

Figure 1. NxH3670 SDK board

- **Software**

  The FRDM-KL27 SDK is for the MKL27Z64VLH4 part, and there is no I2S driver. Therefore, the USB audio example in the FRDM-KL43 SDK is selected as the basic project.

2.2 System block diagram

The system block diagram of KL27 audio speaker is shown in Figure 2.

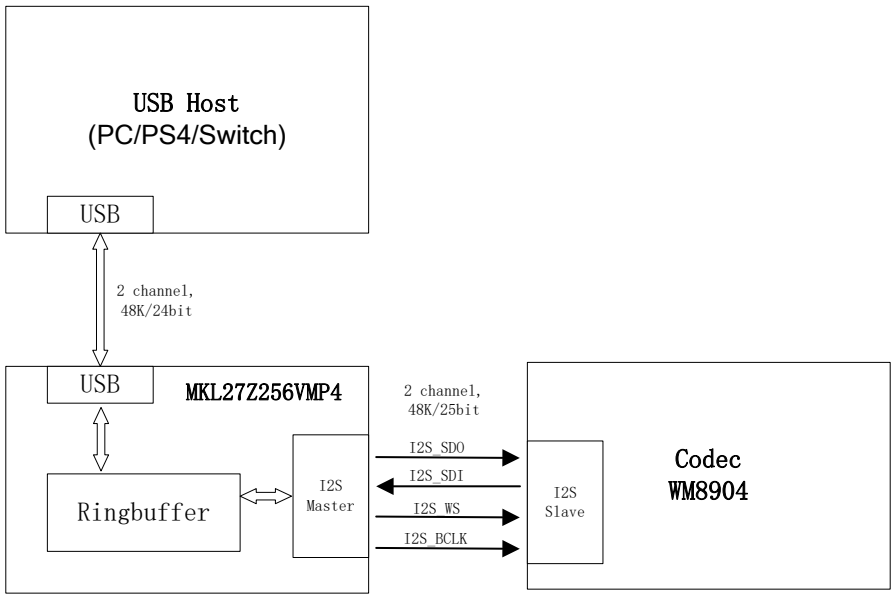

Figure 2. System block diagram of KL27 audio speaker

2.3 Enable USB audio synchronous mode

This example uses dev_composite_hid_audio_bm project in the KL43 SDK as the basic project. The codec driver and the audio ring buffer-related code are added based on the selected project.

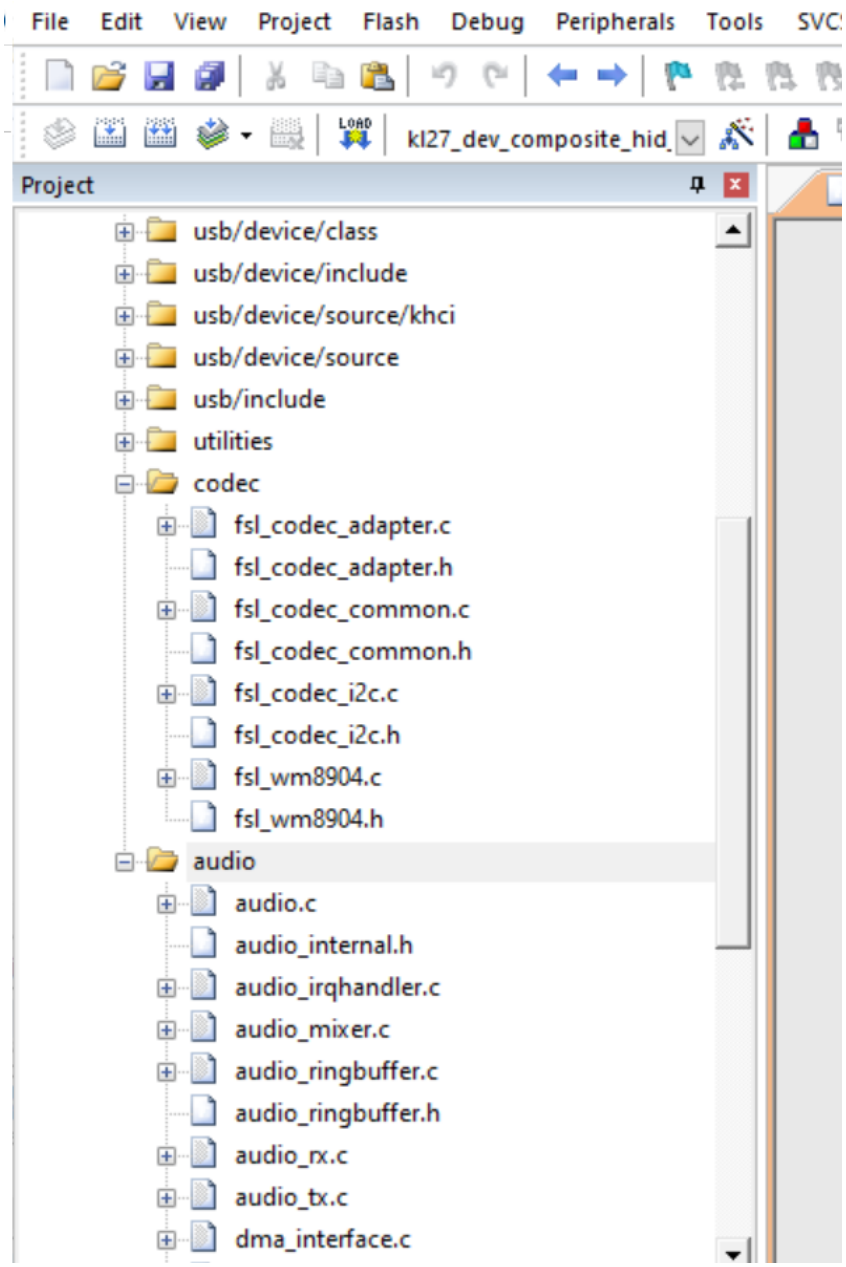

Figure 3. Add codec driver and ring buffer management code

In the original code, the USB audio device works in asynchronous mode. Therefore, the related USB descriptor is modified to make it work in the synchronous mode. In this example, the USB_DEVICE_AUDIO_USE_SYNC_MODE macro is used to configure the USB audio device to work in an asynchronous mode or a synchronous mode. Set the USB_DEVICE_AUDIO_USE_SYNC_MODE macro to 1 to make KL27 work in synchronous mode. For specific configuration details, see the attached code.

2.4 KL27 clock distribution

Configure the I2S master clock (I2S_MCLK) as the 48 M system clock and then divide the I2S_MCLK to generate I2S_BCLK. The format of USB audio device used in this example is a two-channel 48 K/24 bit data stream. If the I2S interface is configured as 48

K/24 bit, the corresponding I2S_BCLK = 48 K * 24 * 2 = 2.034 M. However, you cannot get 2.034 M from 48 M I2S_MCLK integer frequency division. Therefore, the I2S word length can be adjusted to 25 bit. Now, I2S_BCLK = 48 K * 25 * 2 = 2.4 M, I2S_MCLK can be divided by 20 to get the required I2S_BCLK.

Table 2. KL27 clock distribution

Clock	Clock source	Frequency
System clock	HIRC	48 M
Bus clock	HIRC/2	24 M
USB functional clock	HIRC	48 M
I2S_MCLK	System clock	48 M
I2S_BCLK	I2S_MCLK/20	2.4 M

2.5 Codec configuration

The codec on the NxH3670 SDK board is WM8904. The format of USB audio device is two-channel 48 K/24-bit data. Therefore, the codec must be configured to the same data format in the I2S slave mode. The I2S_BCLK and I2S_WS signals are provided by KL27. The master clock of codec is provided by an external 12.288 M crystal oscillator on the board.

Table 3. Digital audio interface data control

Register Address	Bit	Label	Default	Description
R25 (19 h) Audio Interface 1	3:2	AIF_WL [1:0]	10	Digital audio interface word length 00 = 16 bits 01 = 20 bits 10 = 24 bits 11 = 32 bits
	1:0	AIF_FMY [1:0]	10	Digital audio interface format 00 = Right justified 01 = Left justified 10 = I2S 11 = DSP

NOTE

WM8904 is a 24-bit device. If the I2S word length of KL27 is configured to exceed 24-bit length, then WM8904 automatically ignores data that exceeds 24-bit length. Even if the word length of I2S master is configured to 25 bit, it does not affect the actual playback effect of the codec.

2.6 Ring buffer management

This example uses ring buffer to store the audio data sent by the USB host and uses DMA to transfer the data in the ring buffer to the I2S TX data register. The two interrupt service functions implement the management of ring buffer. The two interrupt service functions are:

- USB0_IRQHandler

- Audio_DMATxCallback

The ring buffer mechanism used in this example comes from NxH3670 SDK, using three ring buffers, namely `gs_Interfaces[0].buffer[4096]`, `gs_Interfaces[1].buffer[4096]`, and `s_audioService_BufferOut[4096]`. The ring buffer mechanism in the NxH3670 SDK supports the function of audio mixer. It can simultaneously output the audio data in the two audio speaker interfaces of the game channel and the chat channel. The principle is to store the audio data in the two audio interfaces of the game and the chat separately in the two ring buffers: `gs_Interfaces[0].buffer` and `gs_Interfaces[1].buffer`. Copy the data in the two ring buffers to the `s_audioService_BufferOut` array in chronological order and then use DMA to transfer the data in the `s_audioService_BufferOut` array to I2S TX data register. This enables human ear to hear the mixed sound of the game and chat channels at the same time.

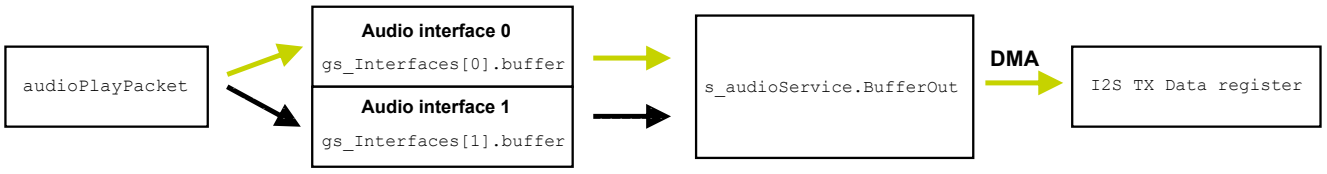

Figure 4. Audio data stream

In this example, only one audio interface is implemented and the function of audio mixer is not used, that is, the audio data stream only moves according to the direction of the green arrow of Figure 4.

2.6.1 Threshold setting of ring buffer

The length of the `gs_Interfaces[0].buffer` ring buffer used in this example is 4096, the threshold setting is shown in Figure 5.

```
36 /* Desired buffer upper limit (from 0 to 100)*/
37 #define BUFFER_UPPER_LIMIT_PERCENTAGE  (60)
38 /* Desired buffer lower limit (from 0 to 100)*/
39 #define BUFFER_LOWER_LIMIT_PERCENTAGE  (40)
```

Figure 5. Threshold setting of ring buffer

The normal range of the margin of the ring buffer is between 40 % (1638) and 60 % (2457) of the length. If the range exceeds, the I2S transfer rate must be adjusted. If the margin exceeds 2457, the I2S transfer rate must be increased. If the margin is less than 1638, the I2S transfer rate must be reduced.

2.6.2 USB interrupt service function

The processing of the audio stream in the USB interrupt service function is shown in Figure 6.

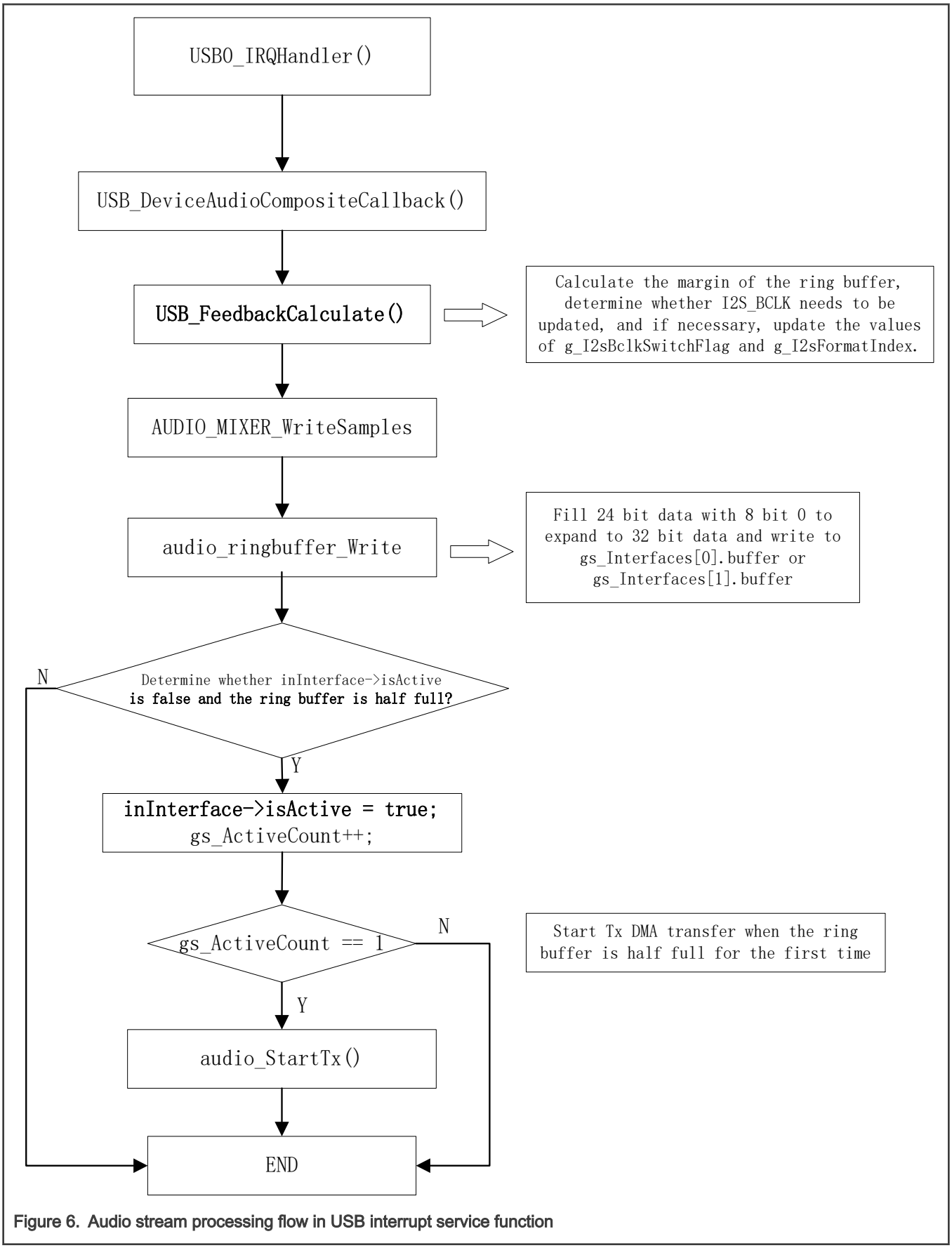

Figure 6. Audio stream processing flow in USB interrupt service function

For a full-speed USB audio device, the USB host sends a packet of audio stream data every millisecond, and the packet size is 288 bytes (48 K*2*3). It can be seen from Figure 6 that in the `USB interrupt service` function, the `USB_FeedbackCalculate` function is first called to calculate the margin of the ring buffer to determine whether I2S_BCLK should be updated. If I2S_BCLK should be updated, update the values of the `g_I2sBclkSwitchFlag` and `g_I2sFormatIndex` variables.

Note that the value of I2S_WS is not updated immediately. It will be updated in the DMA interrupt service function after the current DMA transfer completes. After executing the `USB_FeedbackCalculate` function, call the `audio_ringbuffer_write` function to write the received audio data to `gs_Interfaces[0].buffer` ring buffer.

Before writing, the program firsts expand the 24-bit data to 32-bit data because the length of DMA transfer is 32 bit. Observe whether the ring buffer is half full. If it is half full, call the `audio_StartTx` function to start DMA transfer. Under normal circumstances, the margin of the ring buffer should be kept at about 50 % of the length. If you have strict requirements on the delay of audio data, the length of ring buffer can be reduced to shorten the audio delay.

2.6.3 DMA interrupt service function

Configure the DMA transmission length to be 1 ms audio data stream size. DMA completes a transmission every millisecond and triggers a DMA interrupt. The `audio_DMATxCallback` function is called in the DMA interrupt service function. The audio processing flow in `audio_DMATxCallback` is shown in Figure 7.

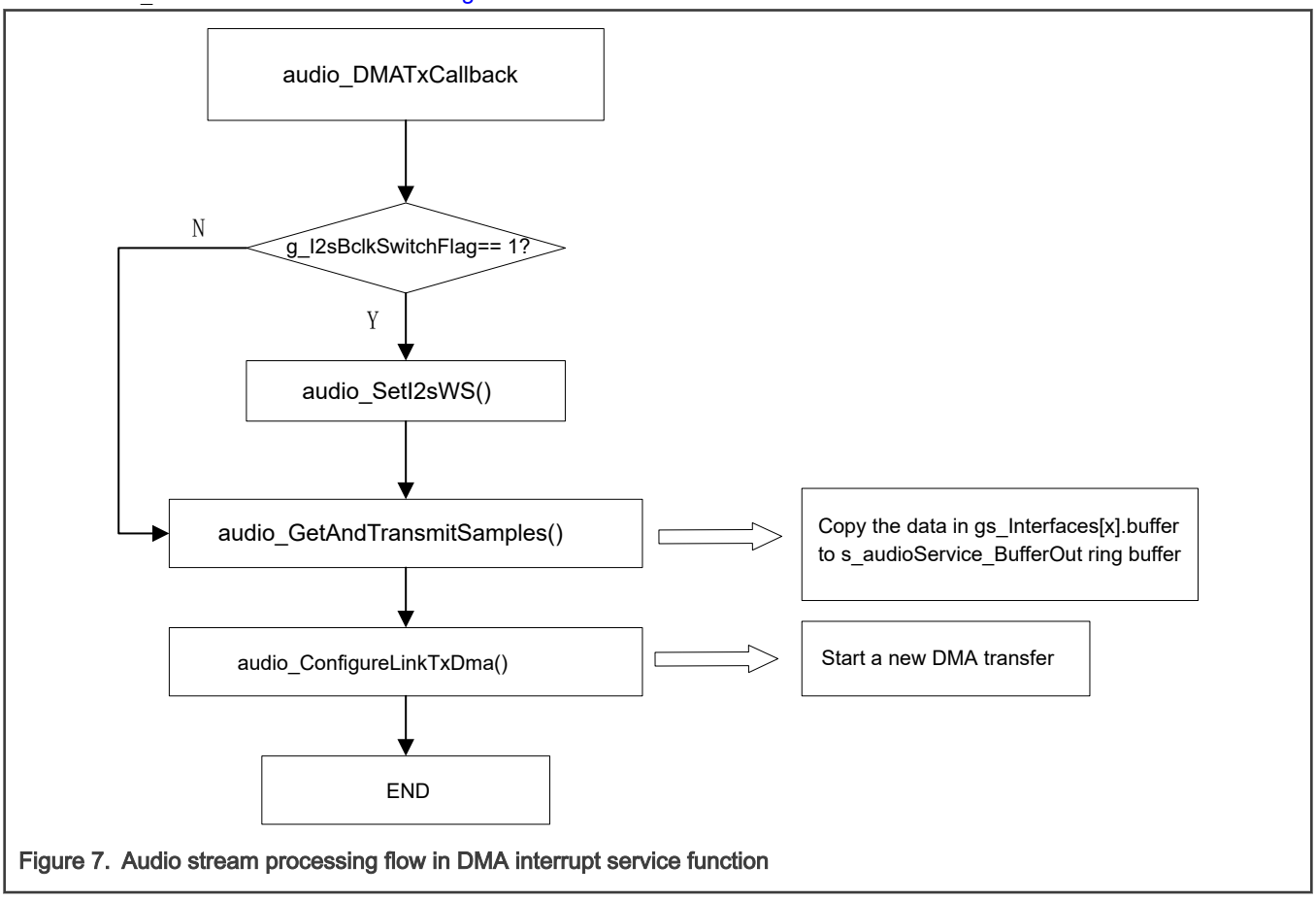

Figure 7. Audio stream processing flow in DMA interrupt service function

In the `audio_DMATxCallback` function, it first determines whether the `g_I2sBclkSwitchFlag` variable is 1. If it is 1, it indicates that the margin of the ring buffer exceeds the set threshold. The program calls the `audio_SetI2sWS` function to adjust the frequency division coefficient of I2S_BCLK and the word length of I2S to adjust the I2S transmission rate. The margin of the ring buffer returns to the threshold range as soon as possible to avoid overflow or underflow of the ring buffer. For details on adjusting I2S_WS, see Update the I2S transfer rate. If there is no need to adjust I2S_WS, directly call the `audio_GetAndTransmitSamples` function to copy 384 bytes audio data in `gs_Interfaces[0].buffer` to the s_audioService_BufferOut array. Call the `audio_ConfigureLinkTxDma` function to start a new DMA transfer, and transfer 384 bytes from the `s_audioService_BufferOut` array to the I2S TX data register.

2.7 Update the I2S transfer rate

From the content of the previous section, if the margin of the ring buffer exceeds the threshold range, the `audio_SetI2sWS` function must be called in the interrupt service function triggered by the current DMA transfer to update the I2S_WS. According to the description in KL27 clock distribution, I2S_BCLK is obtained by dividing the frequency of I2S_MCLK, and I2S_BCLK = I2S_MCLK/(DIV+1)/2 = I2S_WS * 2 * wordLength. You can adjust I2S_WS by modifying the integer division factor of I2S_BCLK and I2S word length.

Table 4. Frequency division factor of I2S_BCLK

DIV	Bit Clock Divide Divides down the audio master clock to generate the bit clock when configured for an internal bit clock. The division value in (DIV + 1) * 2.

Table 4 shows that the frequency division factor of I2S_BCLK can only be an even number. The initial DIV value is 9, and the frequency division factor is 20. I2S_BCLK = 48 M /20 = 2.4 M. To increase I2S_BCLK, set the value of DIV to 8 and the frequency division factor is 18, I2S_BCLK = 48 M / 18 = 2.67 M, I2S_WS = I2S_BCLK/2/25 bit = 53.3 K. To reduce I2S_BCLK, set the value of DIV to 10, the frequency division factor is 22, I2S_BCLK = 48 M / 22 = 2.18 M, I2S_WS = I2S_BCLK/2/25 bit = 43.6 K.

If only the frequency division factor of I2S_BCLK is modified, the I2S_WS and 48 K sample rate are quite different. It is also possible to adjust the I2S word length to obtain a sample rate closer to 48 K. For example, when the frequency division factor is 22, modify the word length to 24 bit, I2S_WS = I2S_BCLK/2/24 = 45.4 K. The three I2S_WS frequencies in Table 5 are used in this example.

Table 5. Three I2S_WS frequency configurations

I2S_WS	Word Length	DIV	g_I2sFormatIndex
47619	28	8	0
48000	25	9	1
48387	31	7	2

```
54  typedef struct _i2s_tx_format_config_t
55  {
56      uint32_t sampleRate;
57      uint32_t  wordLength;
58  }i2s_tx_format_config_t;
59
60  i2s_tx_format_config_t g_I2sTxFormat[3] =
61  {
62      {47619, 28},   // DIV 8
63      {48000, 25},   // DIV 9
64      {48387, 31},   // DIV 7
65  };
```

Figure 8. I2S TX format definition

Three I2S_WS frequencies of 47619, 48000, and 48387 are used in this example to match the transfer rate of the USB host. The USB interrupt service function, periodically calculates the margin of the ring buffer and adjusts I2S_BCLK and I2S_WS in time to avoid overflow and underflow of audio data in the ring buffer. For details on updating I2S_WS, see the code attached with the application note.

3 Test

Download the modified SDK code to the NxH3670 SDK board and run the program. Figure 9 shows the I2S signal captured by the logic analyzer. The USB host used for the test is Windows 10.

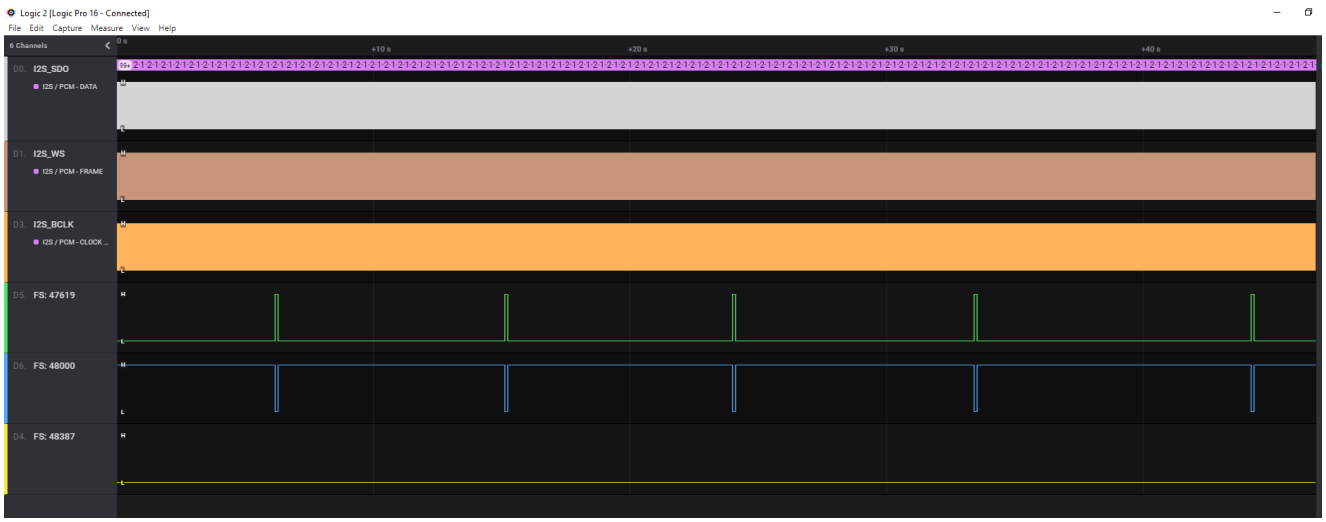

Figure 9. I2S signal captured by logic analyzer

Figure 9 shows that when testing on Window 10, the I2S_WS is adjusted every 9 seconds. In practice, the frequency of adjusting I2S_WS is related to the clock of the USB host and the I2S clock of the USB device.

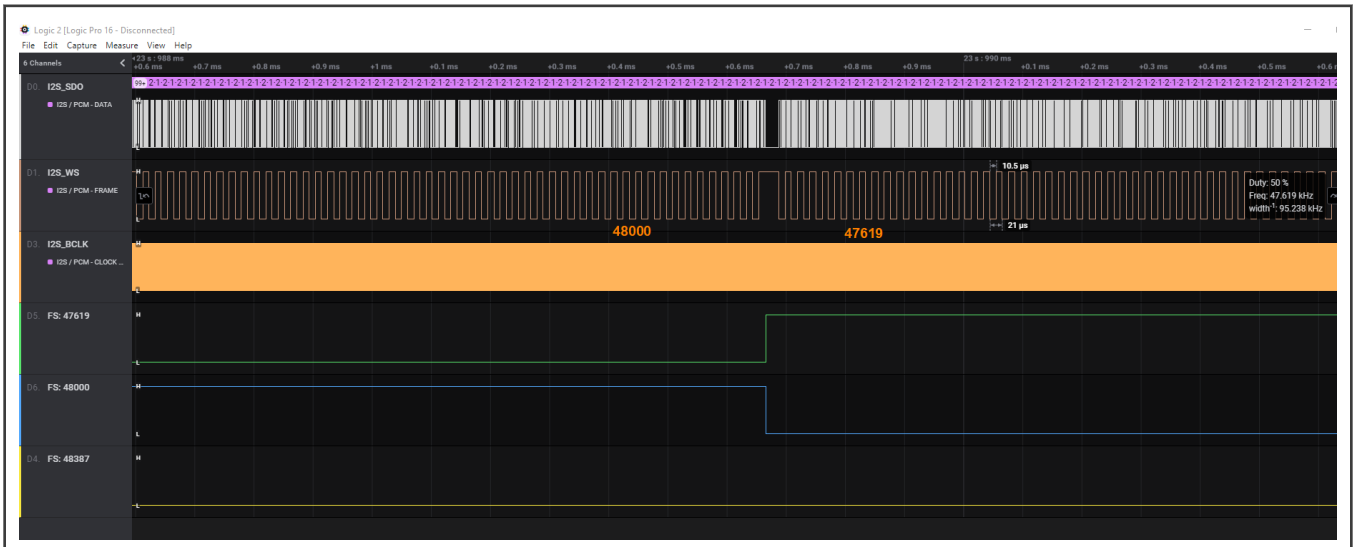

Figure 10. Update I2S_WS frequency

Figure 10 shows the time when I2S_WS is updated. The frequency of I2S_WS is switched from 48000 to 47619, and the word length is switched from 25 bit to 28 bit. Codec automatically ignores the excess 4-bit data.

4 Conclusion

This application note introduces an audio synchronization method for USB audio speaker of Kinetis L series MCUs. The integer division factor of I2S_BCLK and I2S word length is dynamically adjusted to adjust the sample rate of I2S such that the working frequency of I2S synchronizes with the transfer rate of the USB host.

5 Reference

1. UM11150.
2. USB 2.0 Specification
3. Universal Serial Bus Device Class Definition for Audio Devices

6 Revision history

Revision history summarizes the changes done to this document since the initial release.

Table 6. Revision history

Revision number	Date	Substantive changes
0	26 August 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: 26 August 2021

Document identifier: AN13364

arm