# AN14531

## How to Implement USB Audio Mixer on LPC55S69

Rev. 1.0 — 13 January 2025 Application note

**Document information**

| Information | Content |
|---|---|
| Keywords | AN14531, LPC55S69, USB audio, audio mixer, audio speaker |
| Abstract | This application note describes how to implement a USB audio mixer on the NXP LPC55S69. It introduces the steps to implement a USB audio mixer based on the NXP SDK examples, the management of the audio ringbuffer, and the measurement of USB audio latency. |

# 1 Introduction

In the application of wired gaming headsets or wireless gaming headsets with USB dongle, a USB audio mixer is required. For example, when playing online games, we want to hear the sounds in the game and the sounds of teammates. It requires that a USB interface can be enumerated as two USB speakers, to hear the sounds of multiple USB audio speakers synchronously, and also to support the function of USB microphone. Taking the wired gaming headset as an example, it consists of two USB audio speaker: **Chat speaker** and **Game speaker**. The USB audio mixer implemented in this application note simply merges the USB audio data received by two USB audio speakers, and then transmits the merged audio data to the audio codec through the **I$^2$S** interface. Then, the customer can hear both sounds at the same time. This application note describes how to implement the USB audio mixer function on the LPC55S69. This USB audio mixer can support USB audio class 1.0 (UAC1.0) and USB audio class 2.0 (UAC2.0).

The solution based on KL27 and NXH3670 wireless gaming headset of NXP has implemented the function of USB audio mixer, but this solution only supports UAC1.0 and the KL27 SDK version used is SDK 2.7. This application note introduces how to implement the USB audio mixer function on the LPC55S69. This USB audio mixer can support the following functions:

- Supporting UAC1.0 and UAC2.0
- Supporting USB audio synchronous mode
- Supporting high-speed USB and full-speed USB interfaces of LPC55S69
- Supporting LPC55S69 SDK v2.15
- Supporting 48 K/16-bit stereo audio format

The USB audio mixer in this application note is implemented based on the LPC55S6-EVK board and `usb_composite_audio_unified_bm` example in LPC55S69 SDK 2.15. The relevant code of the USB dongle mixer in NXH3670 SDK G9.2 is ported to the USB audio example of LPC55S69 SDK 2.15, and the USB descriptor of UAC2.0 is added. The following chapters introduce how to implement the USB audio mixer on LPC55S69.

# 2 Implementation of USB audio mixer

## 2.1 LPC55S69 SDK example

The `usb_composite_audio_unified_bm` example in the LPC55S69 SDK 2.15 has implemented a USB composite device. It can support a USB audio speaker, a USB recorder, and a USB HID device. We can use the original USB audio speaker in the SDK as the Chat speaker of the USB audio mixer. Therefore, to realize the function of the USB audio mixer, we only need to add a Game speaker based on LPC55S69 SDK example.

## 2.2 Add a USB audio speaker

This section describes how to modify the USB description, related variables, and functions to add a USB audio speaker.

### 2.2.1 Add corresponding USB descriptor

To support a new USB audio speaker (Game speaker) interface, add USB descriptors as shown in Table 1.

**Table 1. USB descriptor for USB audio game speaker**

| USB descriptor type | USB descriptor name | Variable in USB descriptor structure | Descriptor length in UAC1.0 | Descriptor length in UAC2.0 | Comments |
|---|---|---|---|---|---|
| Interface Association Descriptor | Standard Interface Association Descriptor | `iadAudio` | 8 | 8 | |
| Audio Control (AC) Interface Descriptor | Standard AC Interface Descriptor | `control` | 9 | 9 | **UAC1.0**<br>byte 7: `bInterface Protocol`, not used, must be set to 0.<br>**UAC2.0**<br>byte 7: `IP_VERSION_02_00` |
| Audio Class-Specific AC Interface Descriptor | Class-Specific AC Interface Header Descriptor | `controlSub` | 9 | 9 | **UAC1.0**<br>byte 7: `bInCollection`<br>byte 8: `baInterfaceNr`<br>UAC2.0:<br>byte 5: `bCategory`<br>byte 8: `bmControls` |
| | Clock Source Descriptor | `controlSpkr.clock Source` | 0 | 8 | UAC1.0 has no CLOCK Source Descriptor. |
| | Input Terminal Descriptor | `controlSpkr.input Terminal` | 12 | 17 | **UAC2.0**<br>byte 7: `bCSourceID`<br>byte 14-15: `bmControls` |
| | Feature Unit Descriptor | | 0 | 18 | UAC1.0 has no Feature Unit. |
| | Output Terminal Descriptor | `controlSpkr.output Terminal` | 9 | 12 | **UAC2.0**<br>byte 8 `bCSourceID`<br>byte 9 `bmControls` |
| Endpoint Descriptor | Endpoint Descriptor | `controlInterrupt Endpoint` | 9 | 0 | UAC2.0 does not require an interrupt In endpoint descriptor. |
| Audio Streaming Interface Descriptor | Standard AS Interface Descriptor (alt 0) | `streamSpkr.altSet0` | 9 | 9 | **UAC1.0**<br>byte 7: `bInterface Protocol`, not used, must be set to 0.<br>**UAC2.0**<br>byte 7: `IP_VERSION_02_00` |
| | Standard AS Interface Descriptor (alt 1) | `streamSpkr.altSet1` | 9 | 9 | **UAC1.0**<br>byte 7: `bInterface Protocol`, not used, must be set to 0.<br>**UAC2.0**<br>byte 7: `IP_VERSION_02_00` |

**Table 1. USB descriptor for USB audio game speaker***...continued*

| USB descriptor type | USB descriptor name | Variable in USB descriptor structure | Descriptor length in UAC1.0 | Descriptor length in UAC2.0 | Comments |
|---|---|---|---|---|---|
| | Class-Specific AS Interface Descriptor | `streamSpkr.asInterface` | 7 | 16 | **UAC1.0**<br>byte 3: `bTerminalLink`<br>byte 4: `bDelay`<br>byte 5: `wFormatTag`<br>**UAC2.0**<br>byte 4: `bmControls`<br>byte 5: `bFormatType`<br>byte 6-9: `bmFormats`<br>byte 10: `bNrChannels`<br>byte 11-14: `bmChannelConfig`<br>byte 15: `iChannelNames` |
| | Class-Specific AS Format Type Descriptor | `streamSpkr.audioFormat` | 11 | 6 | **UAC1.0**<br>byte 4: `bNrChannels`<br>byte 7 `bSamFreqType`<br>byte 8-11 `tSamFreq` |
| Audio Streaming Endpoint Descriptors | Standard AS Isochronous Audio Data Endpoint Descriptor | `streamSpkr.isoEndpoint` | 9 | 7 | **UAC1.0**<br>byte 7: `bRefresh`<br>byte 8: `bSynchAddress` |
| | Class-Specific AS Isochronous Audio Data Endpoint Descriptor | `streamSpkr.specificIsoEndpoint` | 7 | 8 | **UAC1.0**<br>byte 4: `bLockDelayUnits`<br>byte 5-6: `wLockDelay`<br>**UAC2.0**<br>byte 3: `bEndpointAddress`<br>byte 5: `wMaxPacketSize` |

In this application note, the `usb_class_audio_headphones_device_descriptor_t` structure is used to represent the USB descriptor of the Game speaker. Figure 1 shows the member variables contained in this structure.

```
typedef struct _usb_class_audio_headphones_device_descriptor {
    usb_descriptor_interface_association_t iadAudio;
    usb_descriptor_interface_t control;
    usb_descriptor_class_specific_ac_interface_headphones_t controlSub;

    usb_class_audio_control_io_descriptor_group_t controlSpkr;
#if USB_DEVICE_CONFIG_AUDIO_CLASS_2_0
#else
    usb_descriptor_ac_interrupt_endpoint_t controlInterruptEndpoint;
#endif

    usb_class_audio_stream_descriptor_group_t streamSpkr;
#if USB_DEVICE_AUDIO_USE_SYNC_MODE
#else
    usb_descriptor_as_iso_sync_endpoint_t feedbackEndpointSpkr;
#endif
} usb_class_audio_headphones_device_descriptor_t;
```

**Figure 1. `usb_class_audio_headphones_device_descriptor_t` structure**

As shown in Table 1, the descriptors of USB audio game speaker include the interface association descriptor, audio control interface descriptor, audio stream interface descriptor, and audio stream endpoint descriptor. In addition, when comparing UAC1.0 and UAC2.0, the content of the same descriptor may be different. The

Comments column in Table 1 also briefly lists the differences between the descriptors of UAC1.0 and UAC2.0. For a more specific comparison, see the USB spec (USB Audio Class 1.0 and USB Audio Class 2.0) and AN14531SW.

### 2.2.2 Modify other configurations

In addition to adding the USB descriptors in Table 1, it is also necessary to modify the number of interfaces and endpoints, as well as modify other variables and related callback functions.

#### 2.2.2.1 Modify the number of interfaces and endpoints

To implement a USB audio Game speaker, add two USB interfaces: the audio control interface and the audio stream interface. The new interface configuration is as below.

- #define USB_AUDIO_CHAT_CONTROL_INTERFACE_INDEX (0)
- #define USB_AUDIO_RECORDER_STREAM_INTERFACE_INDEX (1)
- #define USB_AUDIO_CHAT_SPEAKER_STREAM_INTERFACE_INDEX (2)
- **#define USB_AUDIO_GAME_CONTROL_INTERFACE_INDEX (3)**
- **#define USB_AUDIO_GAME_SPEAKER_STREAM_INTERFACE_INDEX (4)**
- #define USB_HID_CONSUMER_CONTROL_INTERFACE_INDEX (5)

It is also necessary to add two USB endpoints for the USB audio Game speaker, one is the audio control endpoint and the other is the audio stream endpoint. The new endpoint configuration is as below.

- #define USB_AUDIO_CHAT_CONTROL_ENDPOINT (6)
- #define USB_AUDIO_CHAT_SPEAKER_STREAM_ENDPOINT (1)
- #define USB_AUDIO_RECORDER_STREAM_ENDPOINT (3)
- **#define USB_AUDIO_GAME_CONTROL_ENDPOINT (7)**
- **#define USB_AUDIO_GAME_SPEAKER_STREAM_ENDPOINT (2)**
- #define USB_HID_CONSUMER_CONTROL_ENDPOINT (4)

#### 2.2.2.2 Modify related variables and functions

In addition to modifying the number of interfaces and endpoints, add some variables related to the USB audio Game speaker used in the USB enumeration process, as shown in Table 2.

**Table 2. Added variables for USB audio Game speaker**

| USB Game speaker related variables |
|---|
| `g_UsbDeviceAudioGameSpeakerEntity` |
| `g_UsbDeviceAudioGameSpeakerEntities` |
| `g_UsbDeviceAudioGameSpeakerControInterface` |
| `g_UsbDeviceAudioGameSpeakerInterfaces` |
| `g_UsbDeviceAudioInterfaceListGameSpeaker` |
| `g_UsbDeviceAudioClassGameSpeaker` |
| `g_CompositeClassConfig` |

For more information about the modification of related variables, see AN14531SW.

To handle the requests related to the USB audio game interface, modify functions shown in Table 3.

**Table 3. Functions that need to be modified**

| Function name | Description |
|---|---|
| `USB_DeviceCallback()` | Add the processing of Set Interface request of game interface |
| `APPInit()` | Add `audioGameSpeakerHandle` related configuration |
| `USB_DeviceAudioRequest()` | Update audio specific request handling |
| `USB_DeviceAudioSpeakerSetInterface()` | Add processing for set Game interface request |

***Note:*** *Table 2 and Table 3 only show some of the variables and functions to be modified or added. For more code modifications, see AN14531SW.*

## 2.3 ringbuffer management

This application note uses the `ringbuffer` management mechanism from the NXH3670 SDK (USB dongle mixer project). Each audio speaker uses a `ringbuffer` to manage audio data. Figure 2 shows the flow of audio data.

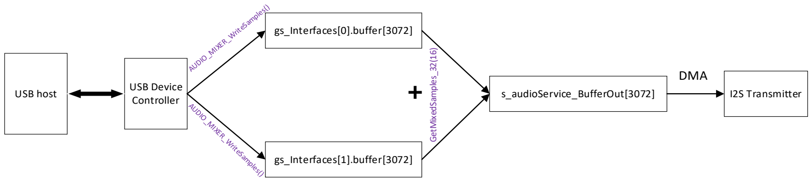



**Figure 2. Audio data flow**

The following source files are used for the management of audio `ringbuffer`.

- *audio_ringbuffer.h*
- *audio_ringbuffer.c*
- *audio_mixer.h*
- *audio_mixer.c*
- *audio_tx.h*
- *audio_tx.c*

The *audio_mixer.c* file defines the `gs_Interfaces[2]` structure array, which includes two structures corresponding to the two audio speakers: Chat and Game. Each structure has a buffer variable. Both buffers are `ringbuffer` with a length of 3072 bytes, which are used to store the USB audio data received by the Chat and Game speakers. In the USB interrupt service routine, the `AUDIO_MIXER_WriteSamples()` function is called to copy the received USB audio packet to `gs_Interfaces[x].buffer[3072]`. When the filling value of the ringbuffer reaches the set threshold, DMA transmission starts. Before each new DMA transfer, the `audio_GetAndTransmitSamples` function is called (the `audio_GetAndTransmitSamples` function calls the `GetMixedSamples_32(16)` function) to merge the audio data in `gs_Interfaces[0].buffer` and `gs_Interfaces[1].buffer`, and copy the merged data to the `s_audioService_BufferOut[3072]` array. Then, DMA transfer is started to move the mixed audio data to the I2S TX FIFO for playback.

## 2.4 USB audio synchronization

The USB audio device in this application note works in synchronous mode, which means that the USB device must follow the Start Of Frame (SOF) signal of the USB host to match the playback speed of the audio data on the USB device side with the USB SOF signal. In this application note, the Ctimer timer is used to capture the USB SOF signal, and the fractional division coefficient of the audio clock (Audio PLL) is adjusted in the

Ctimer capture interrupt service routine to match the audio clock with the USB SOF signal. For the specific implementation method, see the `CTIMER_SOF_TOGGLE_HANDLER_PLL()` function in AN14531SW.

## 2.5 Audio latency measurement

The audio latency in this application note is related to the filling level of the ringbuffer when the DMA transfer starts. The length of each DMA transfer is the length of a USB audio packet. For the UAC1.0 device with two channels 48K/16bit audio format, the length of each USB audio packet is 192 bytes, so the length of each DMA transfer is 192 bytes. As shown in Figure 3, when the filling level of the ringbuffer is greater than or equal to four DMA transfer lengths. DMA transfer starts, that is, at least four USB audio packets must be received before DMA transfers can be started. For UAC1.0 devices, the USB host sends a USB audio packet every 1ms, that is, the audio latency from the USB host starting to send audio data to the USB device starting to transmit audio data to the Codec is about 3-4 ms. For high-speed UAC2.0 devices, the minimum interval of USB audio packets is 125 µs, that is, a USB audio packet is sent in each microframe, and the size of this audio packet is 24 bytes. In the LPC55S69 SDK, the USB audio packet interval of the high-speed UAC2.0 device is set to four microframes by default. That is, an audio packet is sent every 0.5 ms, and each audio packet is 96 bytes, so the audio latency should be 1.5 - 2 ms. For high-speed UAC2.0 devices, if the audio packet interval is adjusted to 125 µs and the initial fill level is adjusted to 2-3 USB audio packet lengths, the audio latency can theoretically be less than 1 ms.

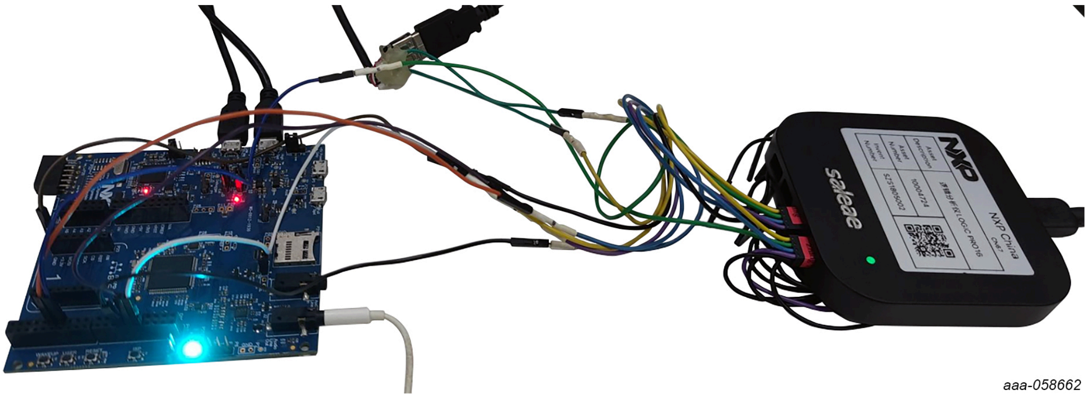
```
    if (!inInterface->isActive && (inInterface->fillLevel >= (AUDIO_GetTxDMATransferSize() * 4) )) {
        inInterface->isActive = true;
        gs_ActiveCount++;

        /* if this was the first interface to become active, then the mixer is ready to start outputting samples */
        if ((gs_ActiveCount == 1) && (gs_ReadyStateChangedCb != NULL)) {
            gs_ReadyStateChangedCb(true);
        }
    }
}
```

Start I2S TX DMA transfer

**Figure 3. Timing to start DMA transfer**

For full-speed USB, you can use a logic analyzer to measure the USB_DP/DM signal and the $I^2S$ signal to calculate the USB to $I^2S$ audio latency, as shown in Figure 4.




aaa-058662

**Figure 4. Measure USB to $I^2S$ audio latency using a logic analyzer**

When the threshold of the initial fill value of the ringbuffer is *AUDIO_GetTxDMATransferSize()* * 4, that is, DMA transfer starts after receiving four USB audio packets. Figure 5 shows the audio delay measurement results of UAC1.0. The audio latency from USB to $I^2S$ is 3.4 ms.

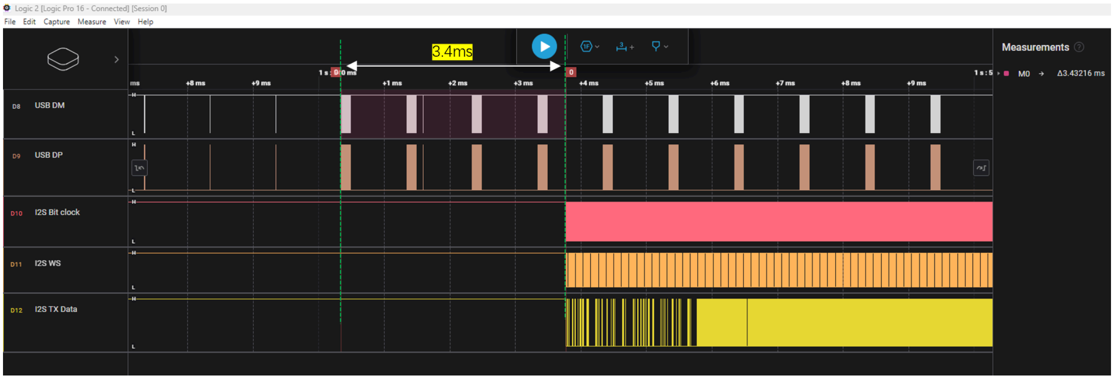



**Figure 5. Measure audio latency of UAC1.0 device**

When using the high-speed USB interface and enabling UAC2.0 mode, the high-speed USB clock is 480 M. Since the maximum sample rate of the logic analyzer used is 500 M, it cannot accurately sample USB DP/ DM signals. Currently, we can use the GPIO toggle method to measure the USB to I$^2$S latency. Toggle a GPIO (P1_7) every time a USB audio packet is received.

```
case kUSB_DeviceAudioEventStreamRecvResponse:
    /* endpoint callback length is USB_CANCELLED_TRANSFER_LENGTH (0xFFFFFFFFU) when transfer is canceled */
    if ((g_deviceAudioComposite->audioUnified.attach) &&
        (ep_cb_param->length != (USB_CANCELLED_TRANSFER_LENGTH)))
    {
        USB_LatencyDebug_ReceivingAudioFromHost(true);   toggle P1_7
```

**Figure 6. Toggle GPIO P1_7 after receiving USB audio packet**

Then toggle another GPIO (P1_6) when starting a DMA transfer and when the DMA transfer is completed.

```
s_audio_GetSamplesCb = getMixedSamplesCb;
/* get and write samples into the output ring buffer */
audio_GetAndTransmitSamples(nbSamples);


HAL_AudioTransferSendNonBlocking((hal_audio_handle_t)&audioTxHandle[0], &s_TxTransfer);   Start TX DMA transfer
AUDIO_LatencyDebug_SendingAudioToI2s(true);
AUDIO_LatencyDebug_SendingAudioToI2s(false);   toggle P1_6
```

**Figure 7. Toggle GPIO P1_6 when starting a DMA transfer and when a DMA transfer is completed**

The time from receiving the first USB audio packet to starting DMA transfer is the audio latency from USB to I$^2$S. As shown in Figure 8, the interval between USB audio packets is 500 µs, that is, a USB audio packet is sent every four microframes. The length of each USB audio packet is 96 bytes. After receiving four USB audio packets, DMA transfer starts. The audio latency from USB to I$^2$S is 1.65 ms.

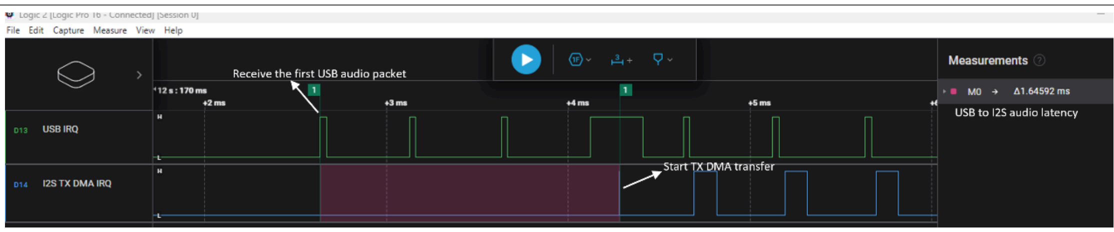



**Figure 8. Measure audio latency on high-speed UAC2.0 device**

To get a shorter audio latency, set the interval of the USB audio packet to 125 µs, that is, send a USB audio packet in each microframe, and reduce the initial fill threshold of the ringbuffer. However, the minimum threshold is the length of two USB audio packets, because we must prepare at least two DMA transfer data before starting DMA transfer to form a link DMA transfer. After the first DMA transfer is completed, the second DMA transfer is started immediately to ensure continuous playback of audio data.

## 3 USB audio mixer testing

The USB audio mixer implemented in this application note supports UAC1.0 and UAC2.0 modes, and supports full-speed USB and high-speed USB interfaces. Customers can set the following macro definitions to select different working modes. These macros are defined in the *usb_device_config.h* file.

*#define USB_DEVICE_CONFIG_LPCIP3511FS (1U)*

*#define USB_DEVICE_CONFIG_LPCIP3511HS (0U)*

*#define USB_DEVICE_CONFIG_AUDIO_CLASS_2_0 (0U)*

After setting the desired mode, you can start compiling the project and download the compiled program to the LPC55S69-EVK through the onboard debugger interface (P6) or the external debugger interface (P7). Press the RESET button (S4) on the board to start running the program. Pay attention to connect the correct USB port to the PC or other USB host. The full-speed USB interface is P10, and the high-speed USB interface is P9. The USB host recognizes a composite USB audio device, as shown in Figure 9.


**Figure 9. USB audio mixer counted by the USB host**

Open two audio players on the USB host and select NXP Dongle mix (Chat) speaker and NXP Dongle mix (Game) speaker to play the audio. Then you can connect a 3.5 mm headphone to the headphone jack (J2) of the LPC55S69-EVK board to hear a mixed audio.

## 4 Conclusion

This application note introduces how to implement a USB audio mixer on the LPC55S69-EVK. Based on the `usb_composite_audio_unified_bm` example in the LPC55S69 SDK v2.15, port the descriptors of the USB dongle mixer in the NXH3670 SDK to this basic project, and add the UAC2.0 function. In addition, the ringbuffer management mechanism, the implementation of the USB audio synchronization mode, and the measurement of audio latency are introduced. Customers can implement the USB audio mixer function on the LPC55S69 and other NXP MCU platforms based on this application note and AN14531SW.

## 5 Reference

1. NXH3670 SDK Gaming Package
2. *Getting started with NxH3670 gaming use case* (document AN12360)
3. *LPC55S6x/LPC55S2x/LPC552x User manual* (document UM11126)
4. USB spec, Audio Device Document 1.0
5. USB spec, Audio Device Rev. 2.0

## 6 Note about the source code in the document

The example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2025 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials must be provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

## 7 Revision history

Table 4 summarizes the revisions to this document.

**Table 4. Revision history**

| Document ID | Release date | Description |
|---|---|---|
| AN14531 v1.0 | 13 January 2025 | Initial public release |

# Legal information

## Definitions

**Draft** — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

## Disclaimers

**Limited warranty and liability** — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

**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.

**Suitability for use** — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

**Applications** — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

**Limiting values** — Stress above one or more limiting values (as defined in the Absolute Maximum Ratings System of IEC 60134) will cause permanent damage to the device. Limiting values are stress ratings only and (proper) operation of the device at these or any other conditions above those given in the Recommended operating conditions section (if present) or the Characteristics sections of this document is not warranted. Constant or repeated exposure to limiting values will permanently and irreversibly affect the quality and reliability of the device.

**Terms and conditions of commercial sale** — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at https://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

**No offer to sell or license** — Nothing in this document may be interpreted or construed as an offer to sell products that is open for acceptance or the grant, conveyance or implication of any license under any copyrights, patents or other industrial or intellectual property rights.

**Quick reference data** — The Quick reference data is an extract of the product data given in the Limiting values and Characteristics sections of this document, and as such is not complete, exhaustive or legally binding.

**Export control** — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

**Suitability for use in non-automotive qualified products** — Unless this document expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

**HTML publications** — An HTML version, if available, of this document is provided as a courtesy. Definitive information is contained in the applicable document in PDF format. If there is a discrepancy between the HTML document and the PDF document, the PDF document has priority.

**Translations** — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

**Security** — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. 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 B.V.** — NXP B.V. is not an operating company and it does not distribute or sell products.

## Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

**NXP** — wordmark and logo are trademarks of NXP B.V.

**Microsoft, Azure, and ThreadX** — are trademarks of the Microsoft group of companies.

## Contents