AN14567

How to implement USB microphone on MCX Series MCUs

Rev. 1.0 — 18 February 2025 Application note

Document information

Information	Content
Keywords	AN14567, USB, Audio Class, PDM
Abstract	This application note describes how to implement the USB microphone function on MCX series microcontrollers. In this application note, we use USB Audio Class 1.0 and USB Audio Class 2.0 for USB audio transfer class and an external digital microphone or generated data as data source.

1 Introduction

This documentation describes how to implement a USB microphone on MCX Series MCUs. The data source could be an external digital microphone or generated data. A USB Audio Class 1.0 (UAC 1.0) and USB Audio Class 2.0 (UAC 2.0) microphone is used in this document.

The USB microphone function could not only be used as a normal microphone, but as a tool to debug multiple channels time series data. This documentation takes FRDM-MCXN947 as an example of how to implement the USB microphone function.

2 USB Audio Class introduction

Refer to the USB-IF documentation Universal Serial Bus Device Class Definition for Audio Devices that describes how a USB audio device is defined, works properly, and defines the USB descriptors. However, this documentation only defines the functions of the audio devices, the specific operation depends on how the USB Host is implemented. For example, the UAC 2.0 device supports multiple clock sources following the USB-IF's documentation. On Windows OS, the multiple clock source function does not support the following Microsoft's UAC2.0 driver documentation.

2.1 USB Audio Class 1.0

USB Audio Class 1.0 is introduced in 1998. It is the first specification for audio devices. Limited to USB 1.1 speed and platform drivers, the UAC 1.0 device supports only stereo or mono audio.

A UAC 1.0 device must implement exactly one Audio Control interface and one or more Audio Streaming interfaces. They are used to control the configuration of the audio device and perform the actual audio data transmission.

To be able to manipulate the physical properties of audio function, "Unit" and "Terminal" are introduced.

The seven types of standard Units and Terminals in the USB Audio Class 1.0 specification are the following: Input Terminal, Output Terminal, Mixer Unit, Selector Unit, Feature Unit, Processing Unit, Extension Unit. For more information, please visit the USB-IF website.

To build a basic UAC 1.0 microphone device, an Input Terminal and Output Terminal are needed. The Input Terminal (IT) is used to interface between the audio functions' "outside world" and other Units. The Output Terminal (OT) is used to interface between Units and the "outside world".

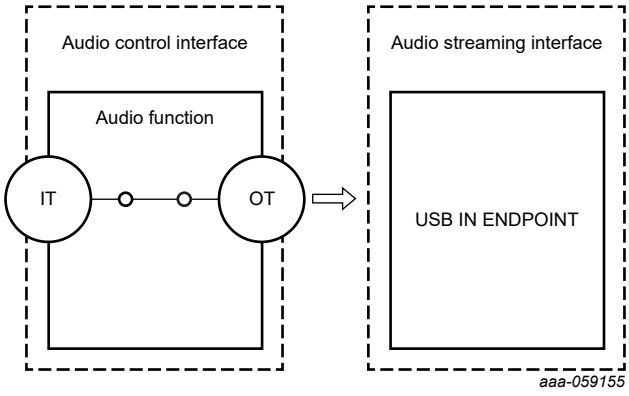

Figure 1. Basic UAC 1.0 microphone device

Audio Streaming interfaces are used to interchange digital audio data streams between the USB Host and the audio function. Each Audio Streaming interface can have at most one isochronous data endpoint. An Audio Streaming interface can have alternate settings that can be used to change certain characteristics of the

interface and underlying endpoint. A typical usage is providing an empty endpoint that does not change any data to save the USB bandwidth.

2.2 USB Audio Class 2.0

The USB Audio Class 2.0 is introduced in 2006, it mainly focuses on the limitations of UAC 1.0 in bandwidth, sampling rate, and audio quality. UAC 2.0 supports USB 2.0 HS 480 MHz rather than UAC 1.0 ‘s USB 1.1 FS 12 MHz.

Same as UAC 1.0, UAC 2.0 needs an Audio Control interface and one or more Audio Streaming interfaces. A basic UAC 2.0 microphone device topology is shown in Figure 2. The difference between UAC 1.0 and UAC 2.0 is that there is a Clock Source Unit.

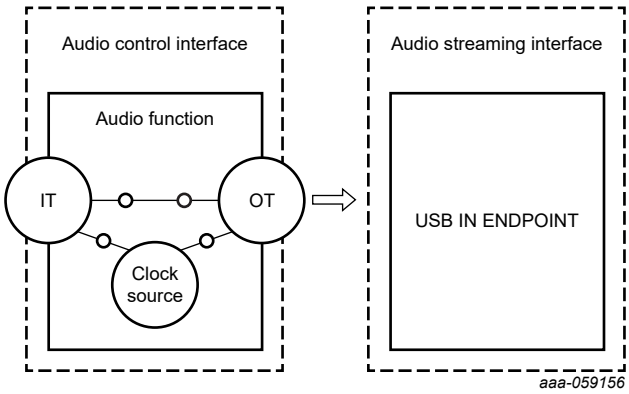

Figure 2. Basic UAC 2.0 microphone device

3 USB Audio Class implementation with MCUXpresso Config Tools

Handwriting USB code is difficult, NXP offers MCUXpresso Config Tools to generate USB-related code to simplify this process. MCUXpresso Config Tools can be download at the NXP website. MCUXpresso IDE provides the built-in version of MCUXpresso Config Tools.

3.1 PDM digital microphone

In this document, an external digital microphone is used to generate audio samples. For the digital microphone module from Adafruit used in this documentation, the following configuration works well.

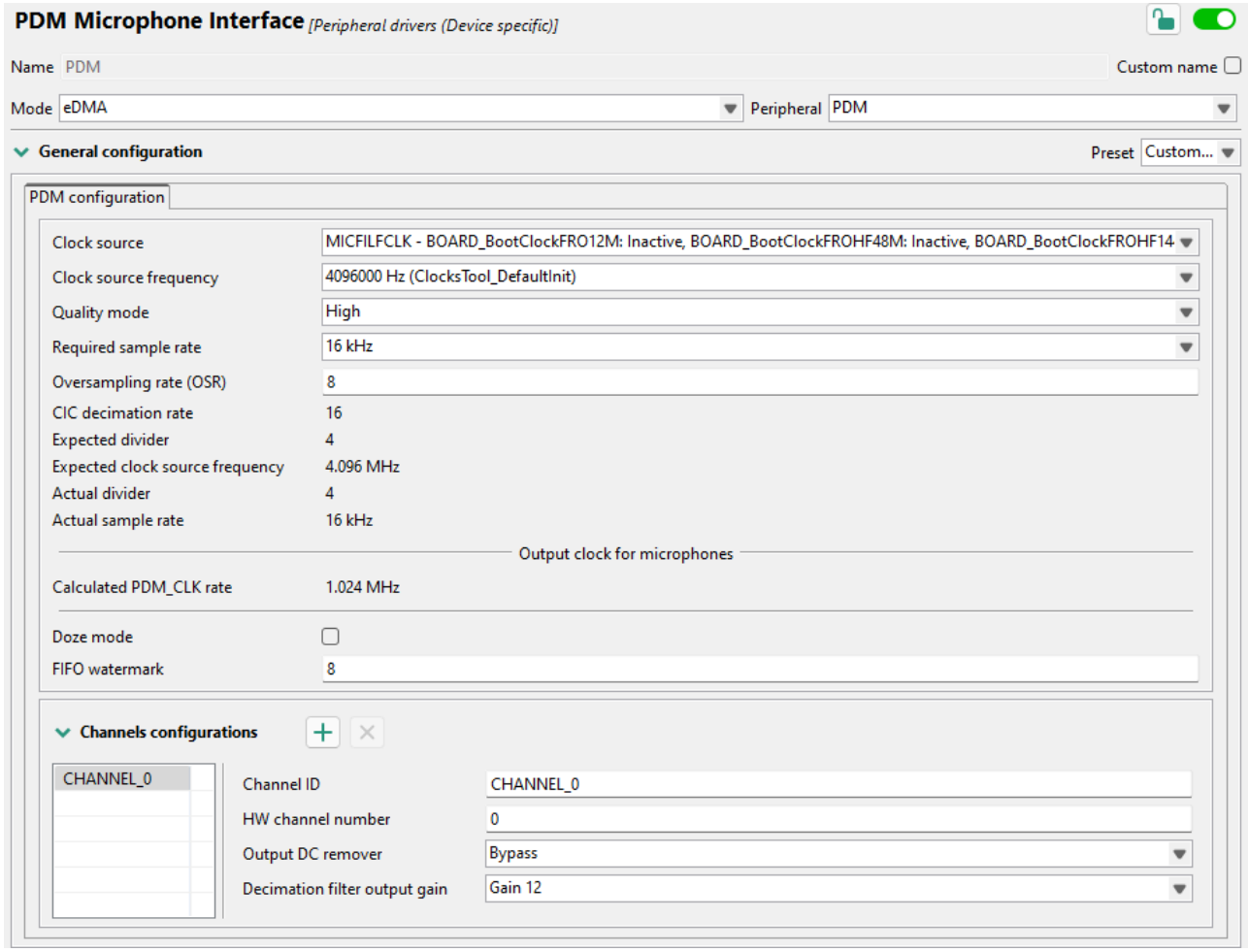

Figure 3. PDM configuration

A ping-pong transfer must be implemented to achieve continuous audio reception. A full code example can be found in SDK.

```
uint8_t pdmBuffer[PDM_BUFFER_SIZE_IN_BYTES * PDM_BUFFER_NUM] __ALIGNED(4) = {0};
uint32_t pdmBufferPosition = 0;
pdm_edma_transfer_t pdmTransfer[] = {
 {
  .data = &pdmBuffer[0],
  .dataSize = PDM_BUFFER_SIZE_IN_BYTES,
  .linkTransfer = &pdmTransfer[1],
 },
 {
  .data = &pdmBuffer[PDM_BUFFER_SIZE_IN_BYTES],
  .dataSize = PDM_BUFFER_SIZE_IN_BYTES,
  .linkTransfer = &pdmTransfer[0],
 },
};
```

3.2 Implement UAC 1.0

In this document, MCUXpresso IDE and FRDM-MCXN947 are taken as examples.

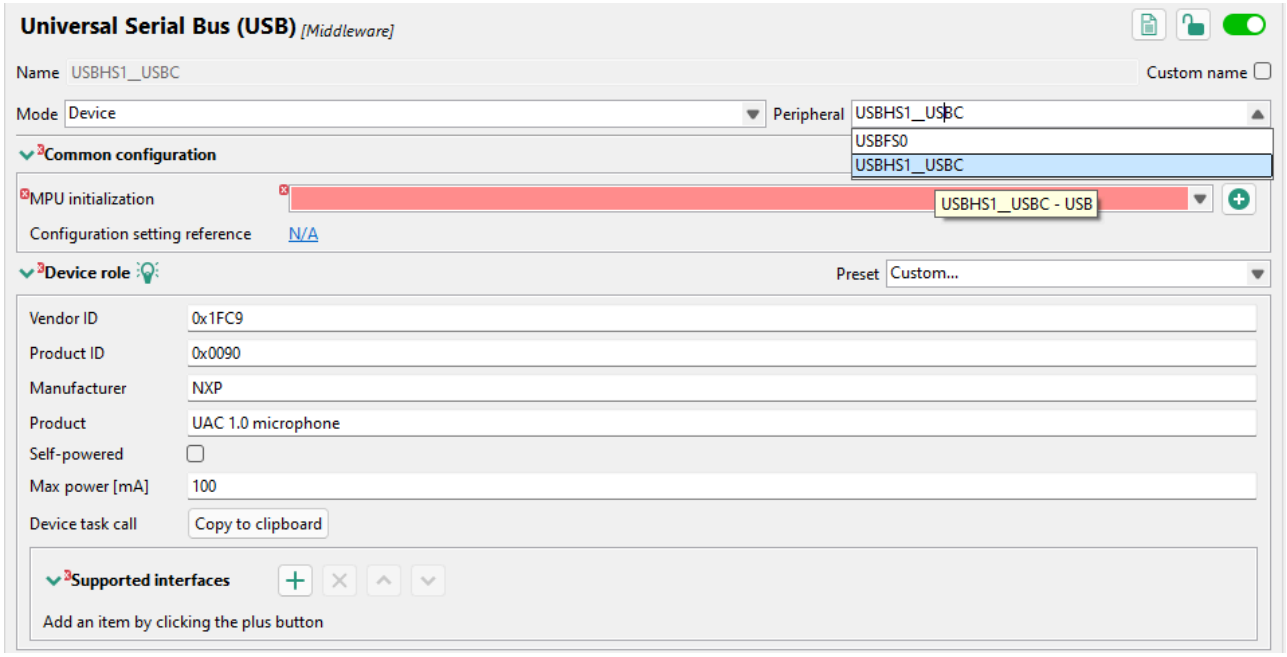

Figure 4. UAC1.0 device configuration

To implement UAC 1.0, follow the steps below:

1. Create a new project in MCUXpresso IDE and enter the MCUXpresso Config Tools page.
2. Create a new middleware component under the “Peripherals” subpage.
3. Choose “USBHS1_USBC” on the Peripheral select box, because the FRDM-MCXN947 board only exposes the USBHS port.
4. Create an MPU component following the tips.
5. Modify VID, PID, and names accordingly. In this example, set the Product name to “UAC 1.0 microphone”.
6. Add interfaces according to the previous introduction. A basic UAC 1.0 microphone device needs an Audio Control interface and an Audio Streaming interface.
7. Click the “+” button to create a new interface, then modify the “Class” option and choose “Audio 1.0”.
8. Select “Audio control” in the “Subclass” option, and get an Audio Control interface.

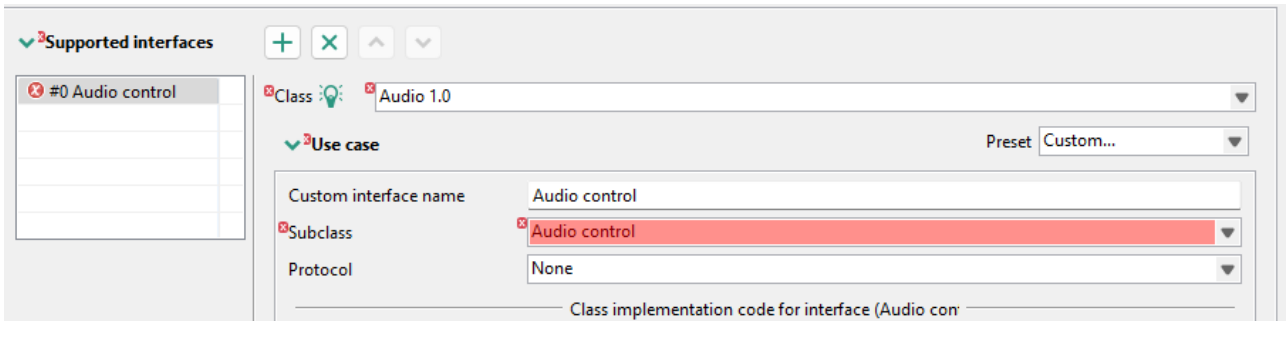

Figure 5. Add UAC1.0 Audio Control interface

9. An Input Terminal (IT) and Output Terminal (OT) are needed in an Audio Control interface. Create these terminals under the “Audio control interface configuration” block.
10. Modify the Terminal ID after creating the Units to prevent conflicts.
11. In the Output Terminal configuration block, set “Source ID” to the Input Terminal ID to implement the topology shown in Figure 6 and Figure 7. The Audio Control interface settings are now complete.

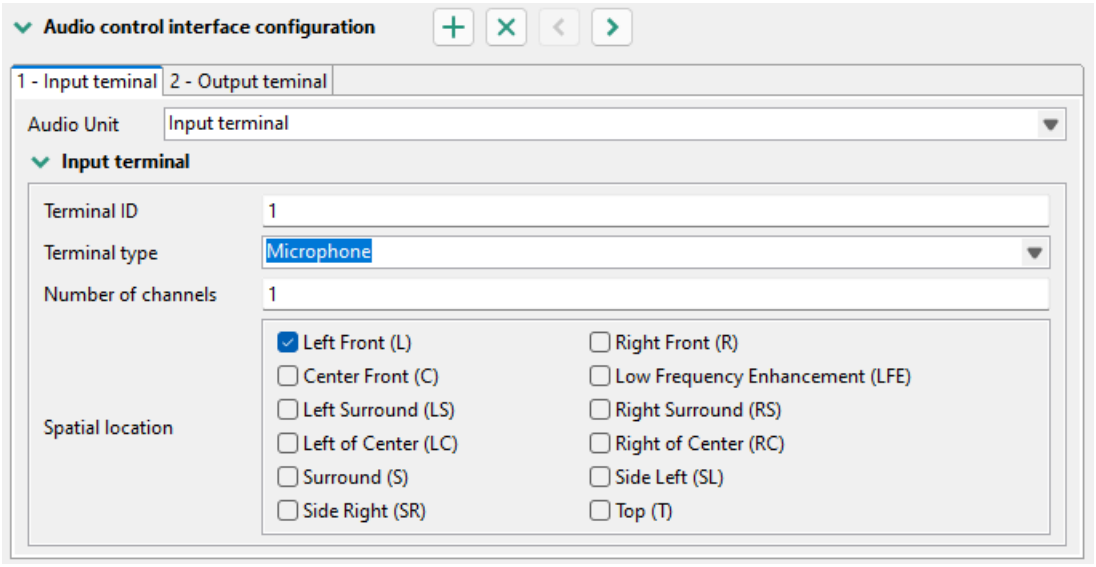

Figure 6. UAC1.0 Audio Control terminal/unit configuration

Figure 7. UAC1.0 Audio Control terminal/unit configuration

The Audio Streaming interface must include at least one endpoint, and a zero bandwidth endpoint is optional. In MCUXpresso Config Tools, the zero bandwidth endpoint is required. To create such an endpoint, create a new endpoint and place it in the first position.

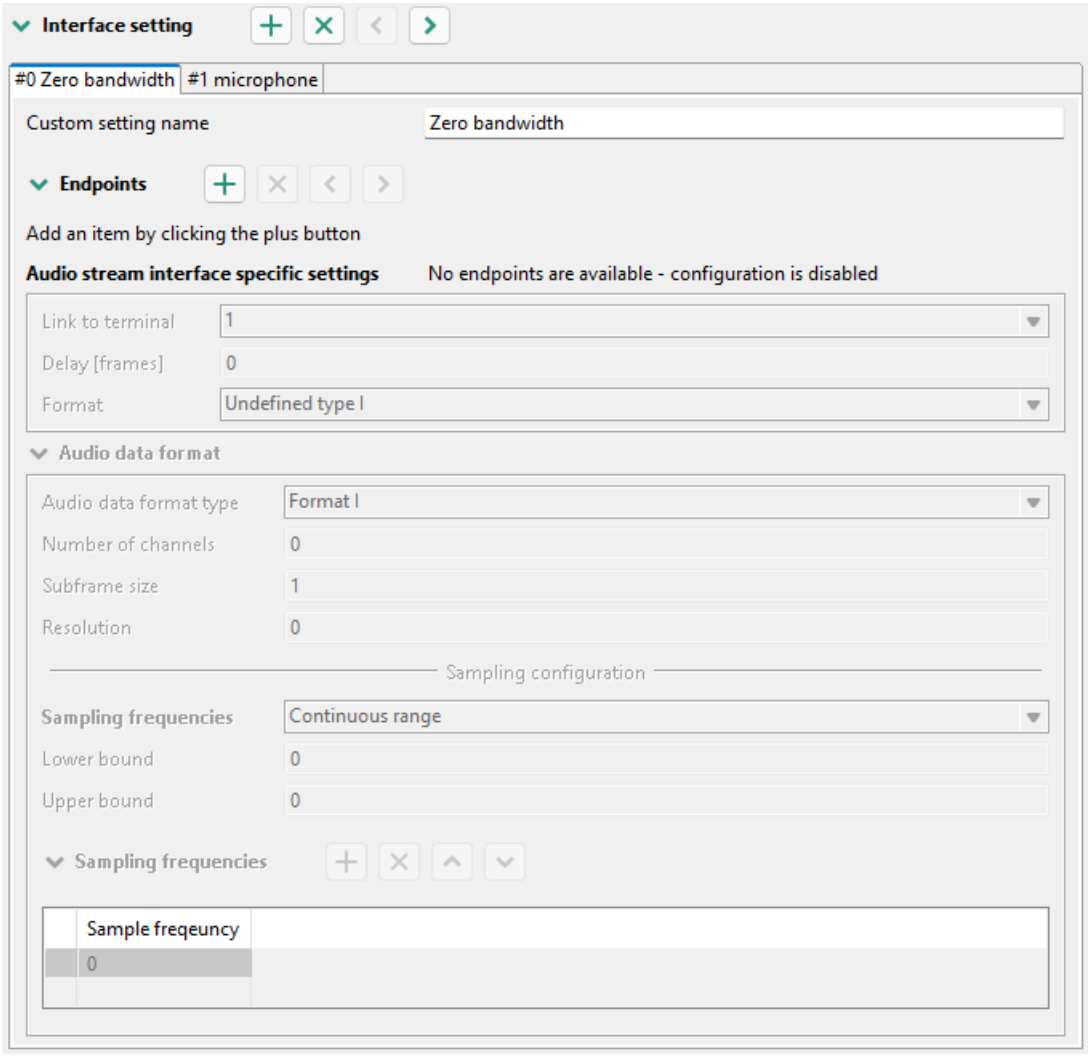

Figure 8. Zero bandwidth Audio Stream interface configuration

Create another endpoint called “microphone”, this endpoint can be used to send the microphone data. Configure this endpoint as shown in Figure 9. For packet size, make sure it is greater than (channels * sample bits / 4 * sample rate / 1000). An interval could be set to a reasonable value. For USB FS, the actual interval is 1 * 2 ^ (value – 1) microseconds, for USB HS the actual interval is 0.125 * 2 ^ (value – 1) microseconds. With the configuration in figure, the actual interval is 1 microsecond for both USB FS and HS. The configuration in Figure 9 is enough for a basic UAC 1.0 microphone device.

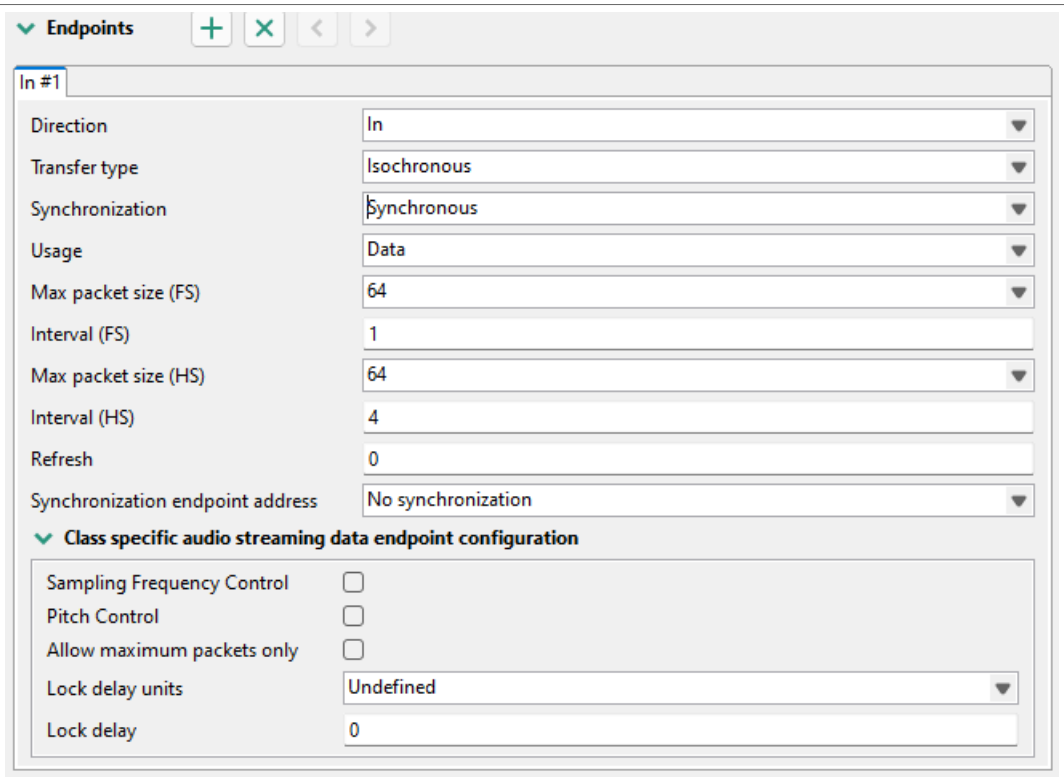

Figure 9. Data Audio Stream endpoint configuration

The configuration in Figure 10 is application-specific. For example, the audio format, PCM, is more common, and float can also be used. In this document, an external digital microphone is used. The PDM peripheral can generate a 24-bit sample. The “Subframe size” is set to 3 and “Resolution” to 24 and 16000 is added to Sample Frequency list.

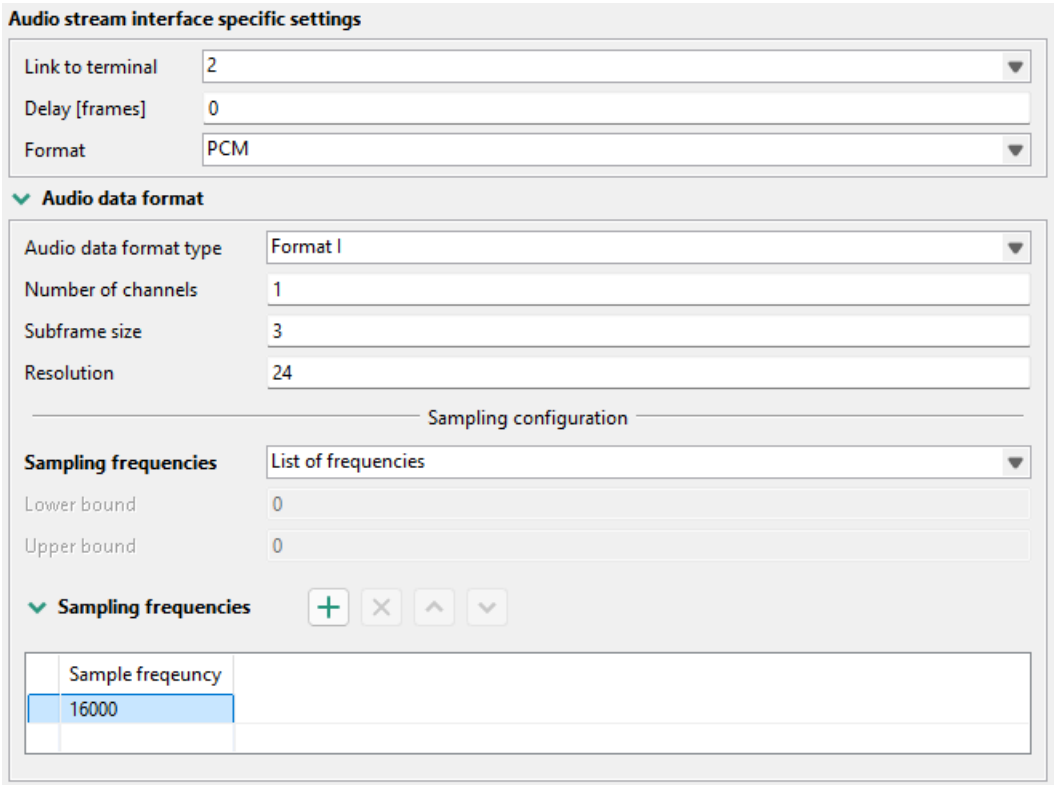

Figure 10. Data Audio Stream interface specific configuration

Now, generate code. To build this project successfully, implement these functions defined in `generated/usb_device_composite.c`.

```
extern usb_status_t USB_DeviceInterface0AudioControlInit(usb_device_composite_struct_t
 *deviceComposite);
extern usb_status_t USB_DeviceInterface0AudioControlCallback(class_handle_t handle, uint32_t
 event, void *param);
extern usb_status_t USB_DeviceInterface0AudioControlSetConfiguration(class_handle_t handle, uint8_t
 configuration_index);
extern usb_status_t USB_DeviceInterface0AudioControlSetInterface(class_handle_t handle, uint8_t
 alternateSetting);
extern usb_status_t USB_DeviceInterface0AudioControlBusReset(usb_device_composite_struct_t
 *deviceComposite);
extern usb_status_t USB_DeviceInterface1AudioStreamingSetInterface(class_handle_t handle, uint8_t
 alternateSetting);
```

An implementation could reference the following content.

```
static usb_status_t USB_DeviceAudioRequest(class_handle_t handle, uint32_t event, void *param)
{
 usb_device_control_request_struct_t *request = (usb_device_control_request_struct_t *)param;
 usb_status_t error = kStatus_USB_Success;

 switch (event)
 {

#if USB_DEVICE_CONFIG_AUDIO_CLASS_2_0
 case USB_DEVICE_AUDIO_CS_GET_RANGE_SAMPLING_FREQ_CONTROL:
  request->buffer = (uint8_t *)&usbAudiofreqRange;
  request->length = sizeof(usbAudiofreqRange);
  break;
 case USB_DEVICE_AUDIO_CS_GET_CUR_SAMPLING_FREQ_CONTROL:
  request->buffer = (uint8_t *)&usbAudioCurFreq;
  request->length = sizeof(usbAudioCurFreq);
  break;
```

```
#endif

 default:
  error = kStatus_USB_InvalidRequest;
  break;
 }

 return error;
}

usb_status_t USB_DeviceInterface0AudioControlInit(usb_device_composite_struct_t *deviceComposite)
{
 // reset buffer position and start PDM receiving
 pdmBufferPosition = 0;
 PDM_TransferReceiveEDMA(PDM_PERIPHERAL, &PDM_PDM_eDMA_Handle, pdmTransfer);

 return kStatus_USB_Success;
}

usb_status_t USB_DeviceInterface0AudioControlCallback(class_handle_t handle, uint32_t event, void
 *param)
{
 usb_status_t error = kStatus_USB_InvalidRequest;

 usb_device_endpoint_callback_message_struct_t *ep_cb_param;
 ep_cb_param = (usb_device_endpoint_callback_message_struct_t *)param;

 switch (event)
 {
 case kUSB_DeviceAudioEventStreamSendResponse:
  if (ep_cb_param->length == USB_ISO_IN_ENDP_PACKET_SIZE)
  {
   error = USB_SendAudioData(handle, USB_INTERFACE_1_AUDIO_STREAMING_INDEX);
  }
  break;

 default:
  if (param && (event > 0xFFU))
  {
   error = USB_DeviceAudioRequest(handle, event, param);
  }
  break;
 }

 return error;
}

usb_status_t USB_DeviceInterface0AudioControlSetConfiguration(class_handle_t handle, uint8_t
 configuration_index)
{
 return kStatus_USB_Success;
}

usb_status_t USB_DeviceInterface0AudioControlSetInterface(class_handle_t handle, uint8_t
 alternateSetting)
{
 return kStatus_USB_Success;
}

usb_status_t USB_DeviceInterface0AudioControlBusReset(usb_device_composite_struct_t
 *deviceComposite)
{
 return kStatus_USB_Success;
}

usb_status_t USB_DeviceInterface1AudioStreamingSetInterface(class_handle_t handle, uint8_t
 alternateSetting)
{
 usb_status_t error = kStatus_USB_Success;

 if (alternateSetting == USB_ALTERNATE_SETTING_1)
 {
  error = USB_SendAudioData(handle, USB_INTERFACE_1_AUDIO_STREAMING_INDEX);
 }
```

```
  return error;
}
```

Function `USB_DeviceAudioRequest` is used to receive a request from the Audio Control interface. In this basic UAC 1.0 device demo, there is no feature unit, so there will not be any requests.

Build the project and connect the USB port to the PC. Open System Settings and check connected devices, it must be similar to the one shown Figure 11.

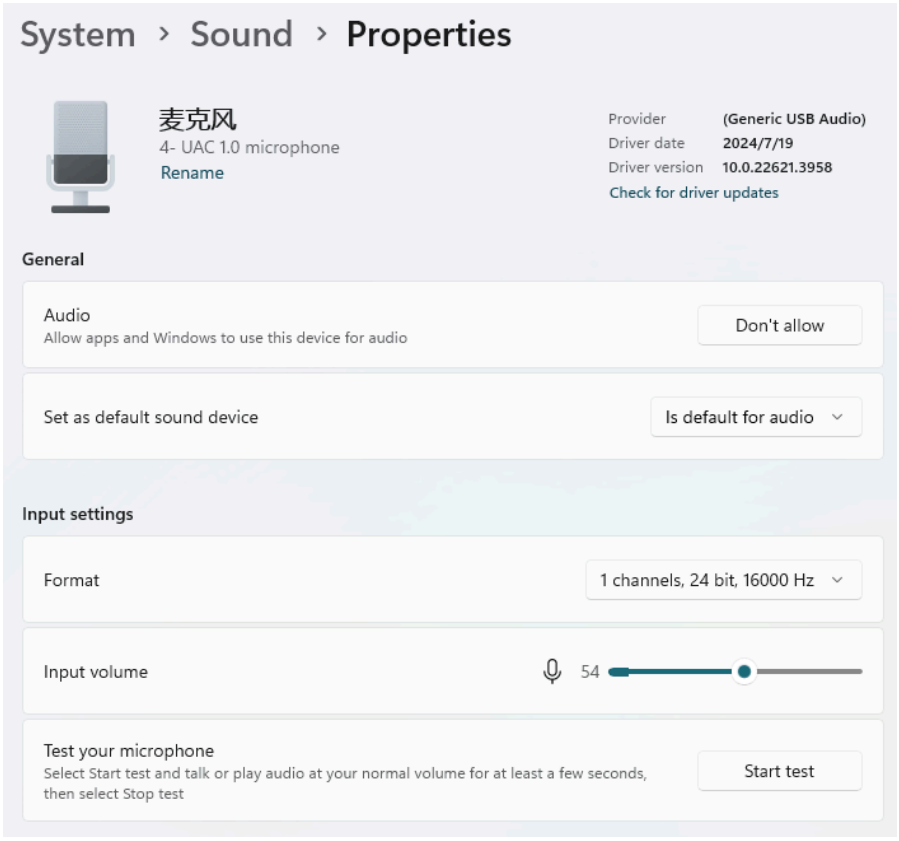

Figure 11. UAC1.0 microphone device in system settings

Use an audio tools like system sound recorder or Audacity to record and playback the sound to confirm it functions properly.

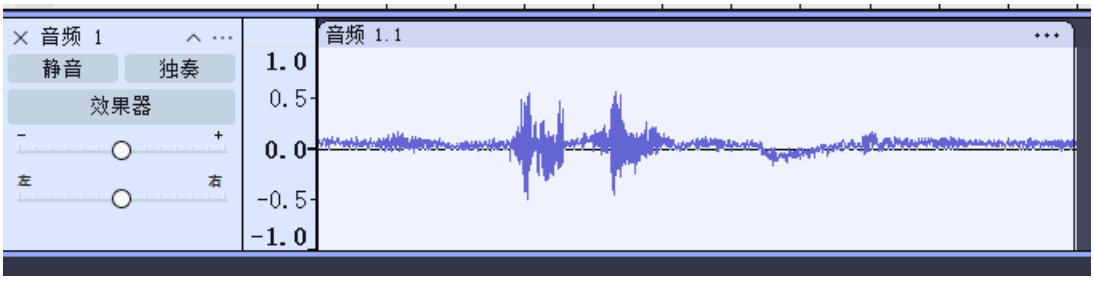

Figure 12. record with UAC1.0 microphone device in Audacity

3.3 Implement UAC 2.0

Same as UAC 1.0, create an Audio Control interface and an Audio Streaming interface with similar configuration. Add a Clock Source Unit in the Audio Control interface, to achieve the topology shown in the UAC 2.0 introduction. The final configurations should be similar to those below.

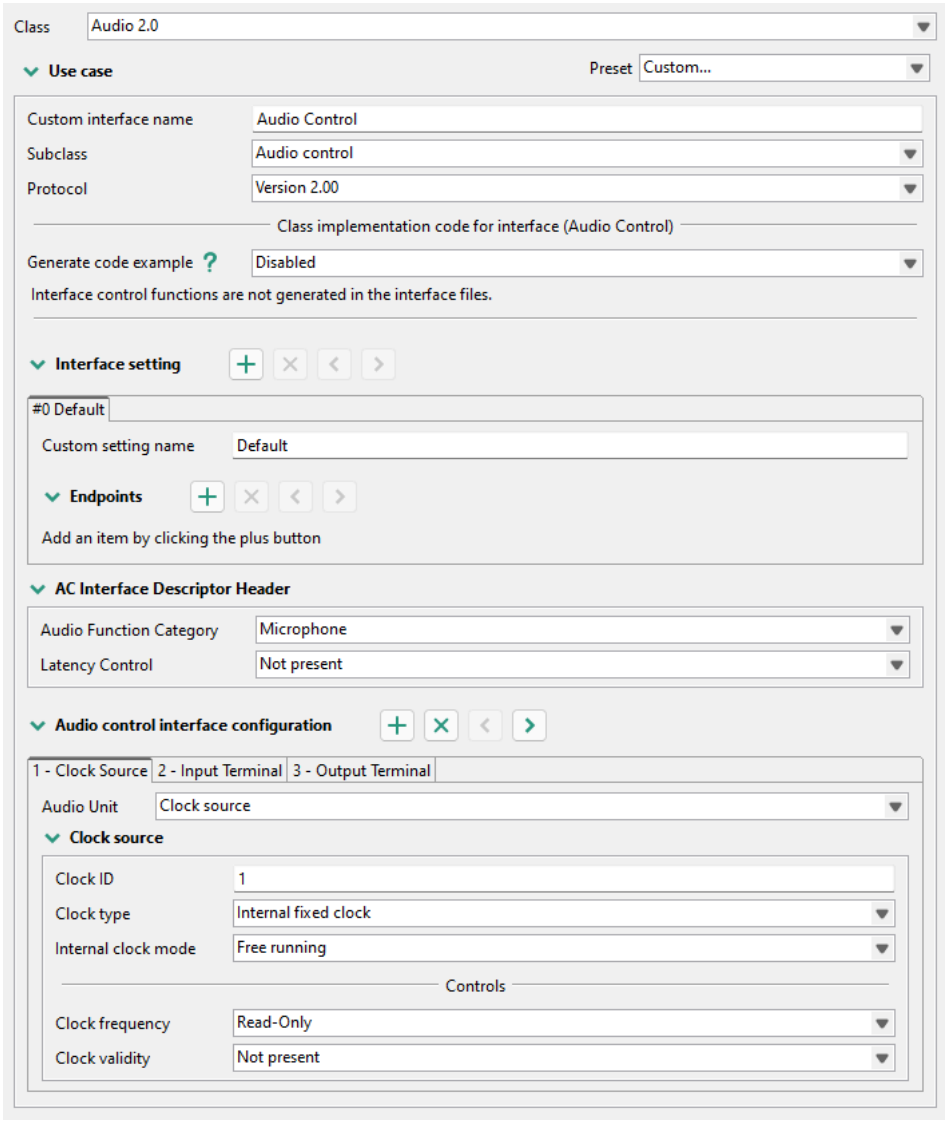

Figure 13. Audio Control interface configuration

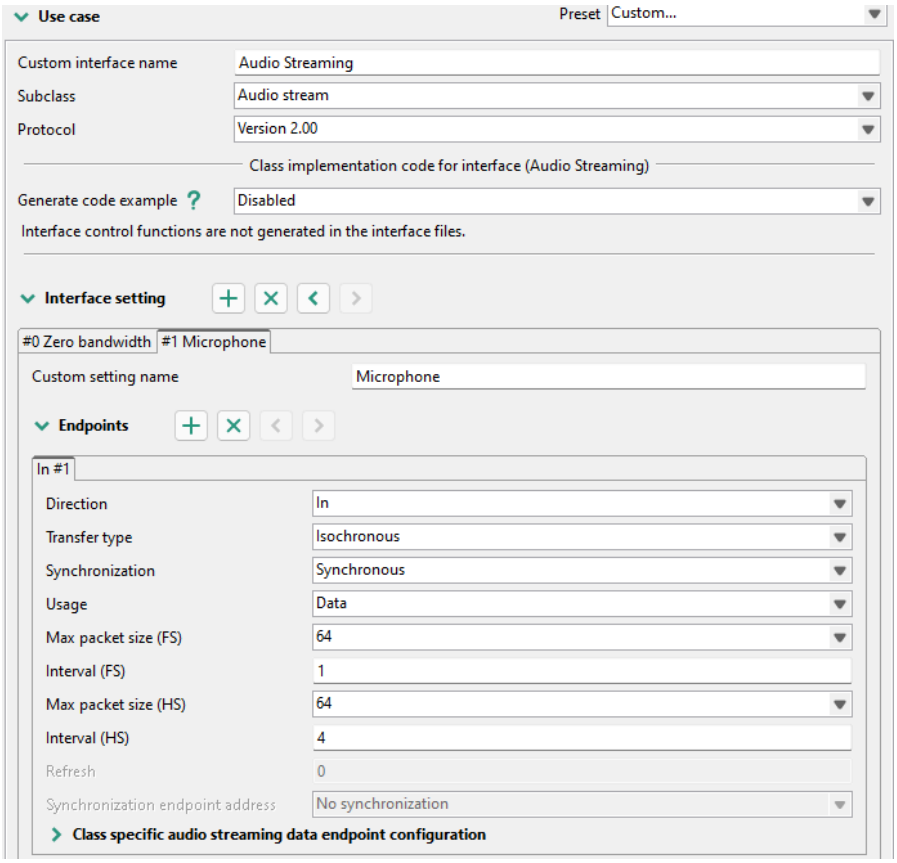

Figure 14. Data Audio Stream interface configuration

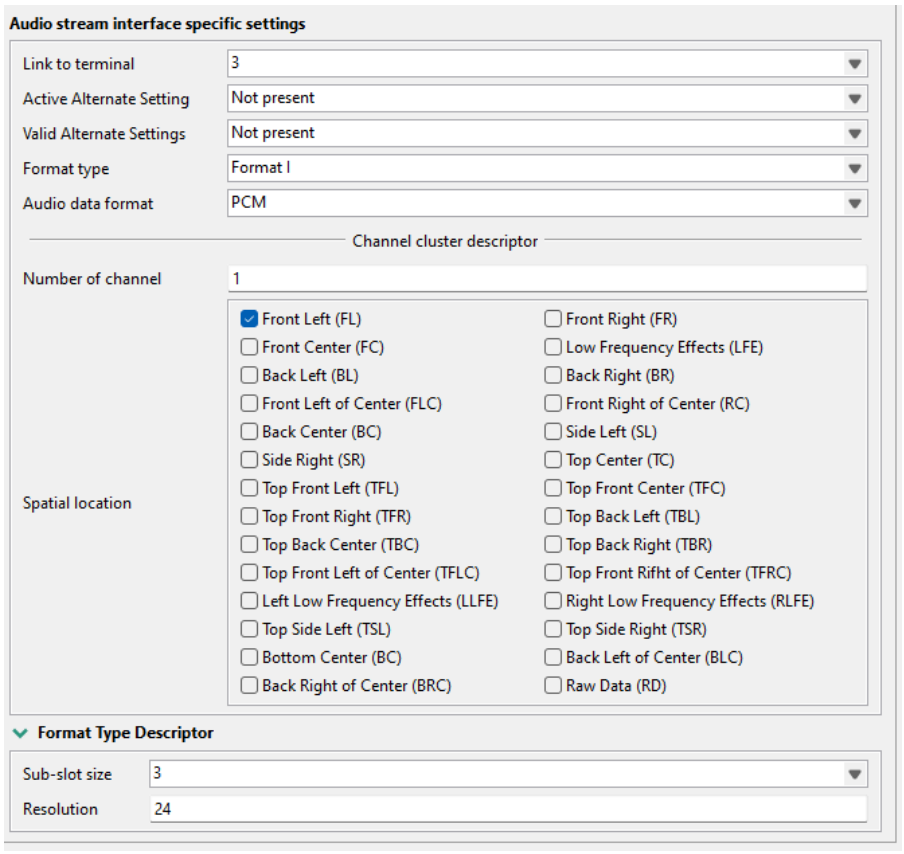

Figure 15. Data Audio Stream interface configuration

After generating the code, implement the interface defined in `generated/usb_device_composite.c`. They have the same definition as UAC 1.0.

```
extern usb_status_t USB_DeviceInterface0AudioControlInit(usb_device_composite_struct_t
 *deviceComposite);
extern usb_status_t USB_DeviceInterface0AudioControlCallback(class_handle_t handle, uint32_t
 event, void *param);
extern usb_status_t USB_DeviceInterface0AudioControlSetConfiguration(class_handle_t handle, uint8_t
 configuration_index);
extern usb_status_t USB_DeviceInterface0AudioControlSetInterface(class_handle_t handle, uint8_t
 alternateSetting);
extern usb_status_t USB_DeviceInterface0AudioControlBusReset(usb_device_composite_struct_t
 *deviceComposite);
extern usb_status_t USB_DeviceInterface1AudioStreamingSetInterface(class_handle_t handle, uint8_t
 alternateSetting);
```

The implementation is the same as UAC 1.0, except the `USB_DeviceAudioRequest` function. In this function, implement the behavior when a specific request is sent.

```
static usb_status_t USB_DeviceAudioRequest(class_handle_t handle, uint32_t event, void *param)
{
 usb_device_control_request_struct_t *request = (usb_device_control_request_struct_t *)param;
 usb_status_t error = kStatus_USB_Success;

 switch (event)
 {

#if USB_DEVICE_CONFIG_AUDIO_CLASS_2_0
 case USB_DEVICE_AUDIO_CS_GET_RANGE_SAMPLING_FREQ_CONTROL:
  request->buffer = (uint8_t *)&usbAudiofreqRange;
  request->length = sizeof(usbAudiofreqRange);
```

```
   break;
  case USB_DEVICE_AUDIO_CS_GET_CUR_SAMPLING_FREQ_CONTROL:
   request->buffer = (uint8_t *)&usbAudioCurFreq;
   request->length = sizeof(usbAudioCurFreq);
   break;
#endif

  default:
   error = kStatus_USB_InvalidRequest;
   break;
  }

  return error;
```

Build and flash, the USB device must work properly.

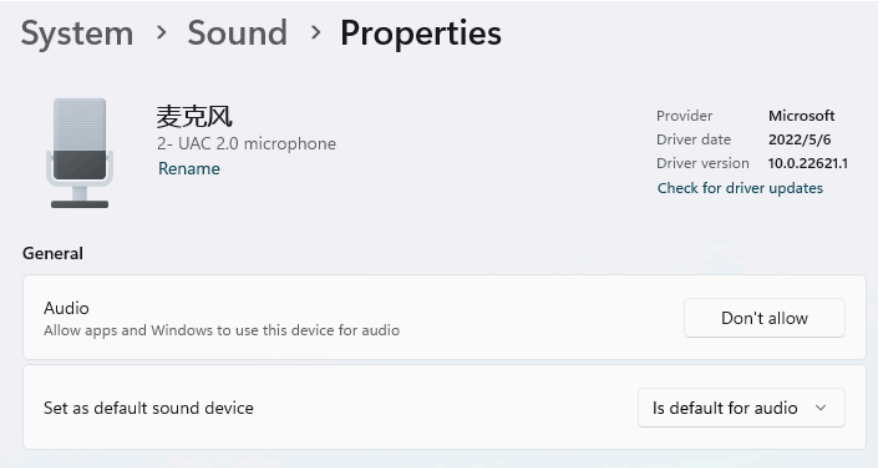

Figure 16. UAC2.0 microphone device in system settings

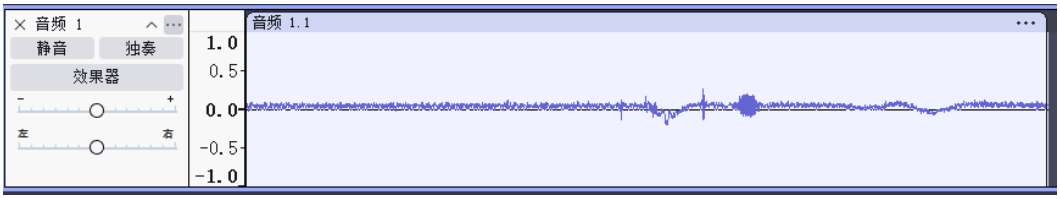

Figure 17. Record with UAC2.0 microphone device in Audacity

4 Revision history

Table 1. Revision history

Document ID	Release date	Description
AN14567 v.1.0	18 February 2025	Initial version

5 Note about the source code in the document

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.

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.

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.

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