ESWIN

Audio User Manual

EIC70X Series AI SoC

Engineering Draft / Rev0.8

2025-3-3

Notice

INFORMATION IN THIS DOCUMENT IS SUBJECT TO CHANGE WITHOUT NOTICE AND DOES NOT REPRESENT A COMMITMENT ON THE PART OF ESWIN. UNLESS OTHERWISE SPECIFIED, ALL STATEMENTS, INFORMATION, AND RECOMMENDATIONS IN THIS DOCUMENT ARE PROVIDED "AS IS" WITHOUTH WARRANTIES, GUARANTEES, OR REPRESENTATIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED.

If applicable, Open-Source Software and/or free software licensing notices are available in the product installation.

Revision History:

Document Version	Date	Note
Rev 0.1	2024/01/22	Initial version
Rev 0.3	2024/02/28	Updated content for audio input, audio output, audio encoding, and audio decoding sections.
Rev 0.6	2024/03/13	Updated content for audio input, audio output, audio encoding, and audio decoding sections.
Rev 0.7	2024/08/06	Updated content for audio input, audio output, audio encoding, and audio decoding sections.
Rev 0.8	2025/3/3	Adjust the Audio software and hardware sections and add an explanation of dual - DIE.

Table of contents

List of tables

List of pictures

1. Overview

The **EIC770X** is a versatile SoC (System on Chip) designed for a wide range of intelligent applications, including smart factories, smart education, smart living, and smart cities. In terms of audio functionality, it supports complex scenarios such as human-machine dialogue, conference calls, speech recognition, voice prompts/alarms, audio playback, and audio encoding. This document primarily introduces the audio software and hardware features related to the **EIC770X SoC**, including:

- EsSDK Audio SDK (Audio Software Development Kit)
- FFmpeg encoding and decoding
- Audio hardware features
- Dual-DIE SoC audio support

Note:
EsSDK is a comprehensive SDK package provided by ESWIN, specifically designed for user application development. It includes several modules, such as Audio, VEnc (Video Encoding), VDec (Video Decoding), Vps (Video Processing), and NPU (Neural Processing Unit), among others.

2. Audio SDK

Audio SDK Framework is shown in the following figure.

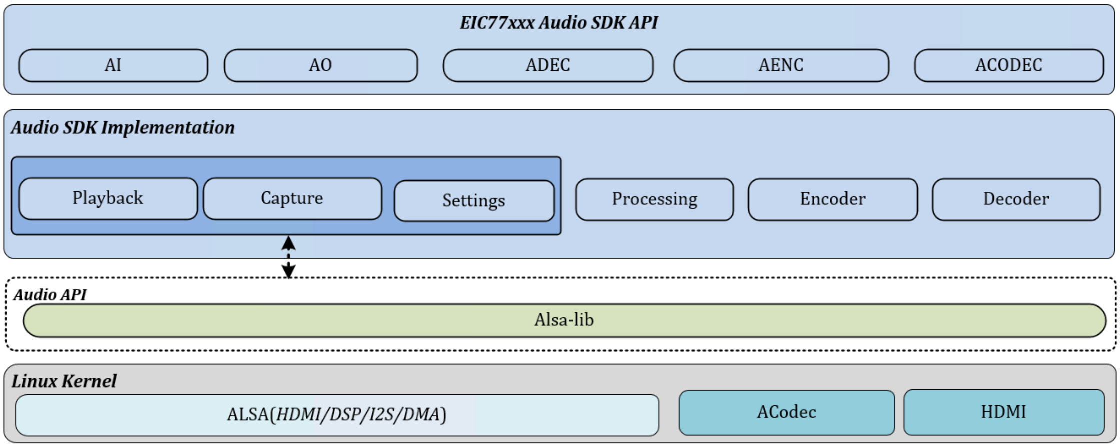


Figure 2-1 Audio SDK Framework

The Audio SDK primarily implements functionalities for audio playback (AO), audio recording (AI), audio encoding (AENC), and audio decoding (ADEC), corresponding to the AO module, AI module, AENC module, and ADEC module. Each of these four modules provides the necessary APIs for users to develop audio applications, supporting both synchronous and asynchronous interface calls. For detailed usage of the APIs, please refer to the Audio Development Manual.

The Audio SDK also offers an audio binding feature, allowing data output from one module (e.g., ADEC) to be directly input into another module (e.g., AO) for improved testing and development efficiency. For instance, PCM data decoded by the ADEC module can be directly fed into the AO module for playback. Users can achieve this complex function simply by calling a few APIs.

The following sections describe the modules, functions, and use cases of the Audio SDK.

2.1 AI Module

The AI module is responsible for recording audio data from peripheral devices such as microphones connected to the system. The EIC770X SoC hardware supports up to three I2S devices, allowing for a maximum of three audio input channels. When using the AI module, it is necessary to configure the Card ID and Device ID according to the actual codec connections on the development board. You can view this information in the development environment by running the command ls /dev/snd.

The AI module supports both mono and stereo audio processing and can handle audio with bit depths of 16-bit, 24-bit, and 32-bit. Additionally, it is capable of performing sound quality enhancement on the recorded audio data to improve the overall audio quality. For more details on sound quality enhancement, you can refer to the Sound Quality Enhancement section of the documentation.

2.2 AO Module

The AO module is responsible for outputting PCM (Pulse Code Modulation) data to devices such as speakers, HDMI, and other peripheral devices connected to the system. The EIC770X SoC hardware supports up to three I2S devices, allowing for a maximum of three audio output channels. When using the AO module, it is necessary to configure the Card ID and Device ID based on the actual codec and HDMI connections on the development board.

The AO module supports both mono and stereo audio processing and can handle audio with bit depths of 16-bit, 24-bit, and 32-bit. Additionally, it is capable of performing sound quality enhancement on the audio data to improve the overall audio experience. For more details on sound quality enhancement, you can refer to the Sound Quality Enhancement section of the documentation.

2.3 ADEC Module

The ADEC module is primarily responsible for decoding the audio compressed data provided by the user, saving the decoded data to a file, and optionally binding to the AO module to output the decoded data to the AO device for playback.

The ADEC module supports various audio encoding types, including:

G711a,G711μ,G722,G726,AAC-LC,AAC-HEv1,AAC-HEv2,AMR.

The ADEC module can support up to 16 decoding channels simultaneously, allowing it to handle multiple audio streams in parallel.

2.4 AENC Module

The AENC module is primarily responsible for encoding the audio data provided by the user. Additionally, it can encode the audio data recorded by the AI module, providing the encoded audio stream for use by applications (APPs).

The AENC module supports the following audio encoding types:

G711a,G711μ,G722,G726,AAC-LC,AMR.

The AENC module can support up to 8 encoding channels simultaneously, allowing for the processing of multiple audio streams at once.

2.5 Resampling

Both the AI module and AO module support audio data resampling. If the AI resampling function is enabled, the internal system will perform resampling before returning the data, and the processed data will be returned. Similarly, if the AO resampling function is enabled, the audio data will be resampled before being sent to the AO channel for playback.

Audio resampling supports conversion between any two different sampling rates, except for 96kHz.

The supported input sampling rates are:
8kHz, 11.025kHz, 12kHz, 16kHz, 22.05kHz, 24kHz, 32kHz, 44.1kHz, 48kHz.

The supported output sampling rates are:
8kHz, 11.025kHz, 12kHz, 16kHz, 22.05kHz, 24kHz, 32kHz, 44.1kHz, 48kHz.

96kHz is not supported for sampling rate conversion.

2.6 Voice Quality Enhancement(VQE)

The **EIC770X** supports VQE (Voice Quality Enhancement) for both audio input and output, which can significantly improve the audio quality. The VQE function offers two distinct processing chains tailored to the characteristics of the AI module (audio input) and the AO module (audio output), with independent switches and parameter configurations for each.

The VQE chain of the AI module mainly includes the RES, VOLUME, HPF, EQ, DRC, ANR, AGC, and AEC modules, which are responsible for processing the AI data. The AI VQE processing chain is shown in the diagram below.

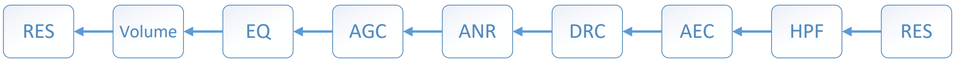


Figure 2-2 AI VQ Processing Chain

The VQE chain of the AO module mainly includes the RES, VOLUME, HPF, EQ, ANR, and AGC modules, which are responsible for processing AO data. The AO VQE processing chain is shown in the diagram below.

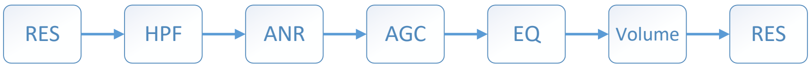


Figure 2-3 AO VQE Processing Chain

- The RES (Resample) module converts the input sampling rate to the required working sampling rate for various VQE algorithms (8kHz, 16kHz, 32kHz, 48kHz) and then converts the working sampling rate to the output sampling rate. The RES supports mono and stereo processing, does not support dynamic enabling or disabling.
- The VOLUME module is responsible for audio volume adjustment and includes mute/unmute functionality. The VOLUME module has no restrictions on the working sampling rate, supports mono and stereo processing, and does not have switch control.
- The HPF (High Pass Filter) module is primarily responsible for removing low-frequency noise, which often originates from hardware noise or power-line interference, resulting in an uncomfortable humming sound。 The HPF module has no restrictions on the working sampling rate,, supports mono and stereo processing, and supports switch control.
- The EQ(Equalize) module adjusts the gain of different frequency bands in the audio signal. The EQ supports 8kHz, 16kHz, 32kHz, and 48kHz working sampling rates, supports mono and stereo processing and supports switch control.
- The DRC(Dynamic Range Control) module controls the output level, keeping the gain within a certain range, ensuring the sound is neither too loud nor too quiet. The DRC has no restrictions on the working sampling rate, supports mono and stereo processing and supports switch

control.

- The ANR(Audio Noise Reduction) module removes background noise while preserving voice input, suitable for speech enhancement applications. The ANR supports 8kHz, 16kHz, 32kHz, and 48kHz working sampling rates, supports mono processing and supports switch control.
- The AGC (Auto Gain Control) module is primarily responsible for gain control of the output level. When the input sound volume fluctuates, it ensures the output volume remains within a consistent range, primarily functioning in scenarios where it is important to ensure the sound is neither too loud nor too quiet. The AGC supports 8kHz, 16kHz, 32kHz, and 48kHz working sampling rates, supports **mono** processing and supports switch control.
- The AEC (Acoustic Echo Cancellation Module) module is used in scenarios requiring echo cancellation, such as IPC intercom, where it removes the echo of the far-end voice that is played on the local device and captured by the microphone. The AEC supports 8kHz and 16kHz working sampling rates, supports mono processing and supports switch control.

2.7 Use Cases

After the AI module records audio data, it can be processed through VQE and saved to a file to implement the recording function. Additionally, the AI module's functionality can be extended based on the system binding method for various application scenarios, such as:

- After the AI module records audio data, the data is processed through VQE and then bound to the AO module through the system binding method. This allows the audio to be played through a speaker or HDMI peripheral, implementing the record and playback function.

- After the AI module records audio data, the data is processed through VQE and then bound to the AENC module through the system binding method, encoding the data into the corresponding audio stream for output to the APP, implementing the record and encode function.

The diagram below illustrates how the recorded audio by the AI module can be played through the AO module or encoded by the AENC module through the system binding method.

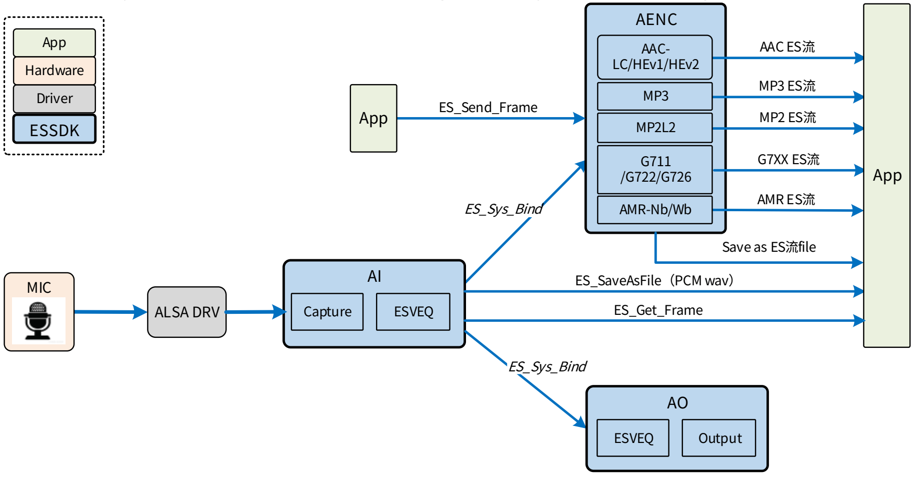


Figure 2-4 Binding AI with AENC and AO

In addition to directly playing audio files or audio from the AI module, the AO module can also bind the ADEC module through system binding. This allows the data decoded by ADEC to be played through the AO module, enabling the decode and playback functionality, as shown in the diagram below.

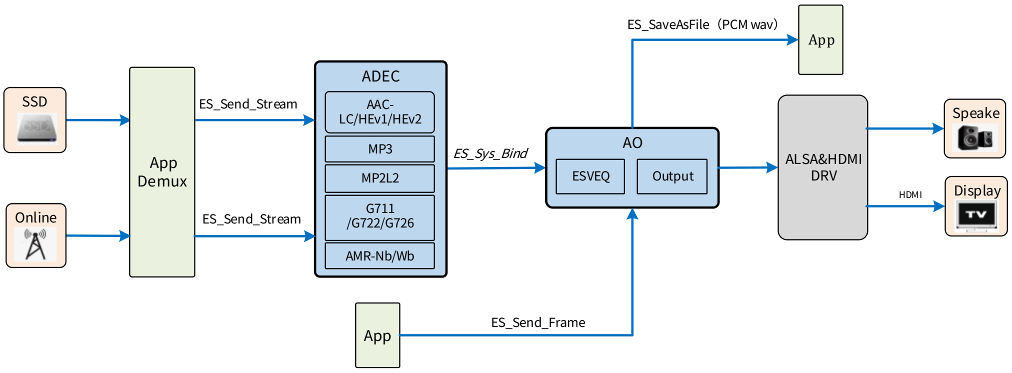


Figure 2-5 Binding AO and ADEC

3. Ffmpeg Plugin

The FFmpeg plugin is an efficient audio codec plugin developed based on the FFmpeg framework, providing users with convenient and flexible audio encoding and decoding capabilities. This plugin supports various mainstream audio formats and is suitable for a wide range of application scenarios, such as streaming services, voice communication, and embedded audio processing devices.

The FFmpeg codec plugin supports the following formats:

- AAC Series: Supports common high-efficiency audio decodings, such as AAC-LC and HE-AAC. The encoder currently supports the LC profile type encoding, suitable for high-quality audio scenarios.
- G Series: Covers formats like G711, G722, G726, widely used in voice communication, IP telephony, and other similar applications.
- AMR Series: Includes AMR-NB and AMR-WB, optimized for the voice compression needs of communication devices, providing efficient voice encoding solutions.
- MP3 Decoding: Supports mainstream MP3 format decoding, suitable for music playback, content consumption, and other related scenarios.

4. Gstreamer Plugin

The GStreamer plugin provides audio codec support for a variety of mainstream audio formats, including encoding and decoding functions for formats like AAC, G711, G722, G726, AMR, and decoding for MP3. This plugin is widely applicable in various scenarios such as streaming services, voice communication, and embedded audio processing devices.

The specific supported formats are as follows:

- AAC Series: Supports common high-efficiency audio decodings, such as AAC-LC and HE-AAC. The encoder currently supports the LC profile type encoding, suitable for high-quality audio scenarios.
- G Series: Covers formats like G711, G722, G726, widely used in voice communication, IP telephony, and similar applications.
- AMR Series: Includes AMR-NB and AMR-WB, optimized for the voice compression needs of communication devices, providing efficient voice encoding solutions.

- MP3 Decoding: Supports mainstream MP3 format decoding, suitable for music playback, content consumption, and related scenarios.

5. Audio Hardware Features

5.1 Audio Input and Output

The audio input process is as follows: the raw analog audio signal is converted into a digital signal by the Codec at a certain sampling rate and sampling precision. The Codec uses the standard I2S timing protocol to transmit the digital signal to the I2S device. The chip uses DMA to transfer the audio data from the I2S device to memory, completing the audio recording.

The audio output process is similar to the audio input process. DMA transfers the data from memory to the I2S device, which then sends the data to the Codec or HDMI via the I2S protocol. The Codec and HDMI subsequently handle the downstream data processing.

The audio input and output process is illustrated in the diagram below.

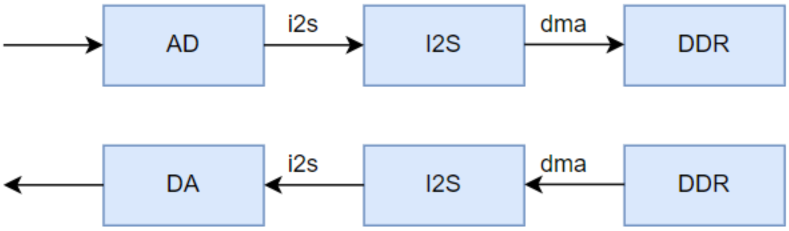


Figure 5-1 Hardware Data Flow

In the **EIC770X**, I2S is designated as the master device to provide the clock. The **EIC770X** has three I2S interfaces, all sharing the same master clock. Each I2S interface has an independent frequency divider, which can generate different clocks. If an I2S interface operates in full-duplex mode, the working clocks of the transmit and receive sides must be the same. The hardware clock is shown in the diagram below.

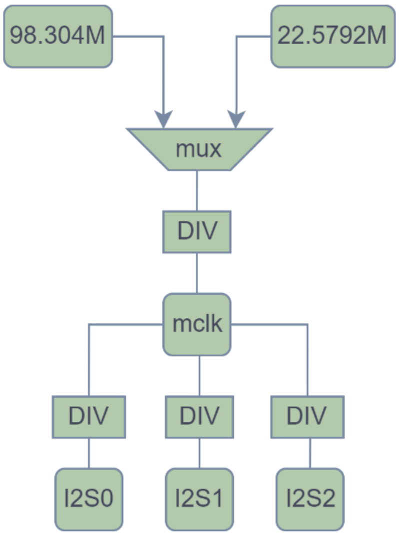


Figure 5-2 Hardware Clock

In the **EIC770X**, each of the three I2S interfaces supports the following sampling rates: 8kHz, 16kHz, 32kHz, 44.1kHz, 48kHz, and 96kHz. It supports data bit widths of 16bit, 24bit, and 32bit, as well as stereo channels and little-endian processing. The following points should be noted regarding the I2S:

- The three I2S interfaces share the same master clock. The clock corresponding to 44.1kHz cannot be a multiple of the clock corresponding to 48kHz, so 44.1kHz cannot work simultaneously with other sampling rates (96kHz, 48kHz, 32kHz, 16kHz, 8kHz). For example, if I2S0 is playing 44.1kHz audio, then I2S1 and I2S2 can only play 44.1kHz audio. If I2S0 is playing 48kHz audio, I2S1 and I2S2 cannot play 44.1kHz audio and can only play 96kHz, 48kHz, 32kHz, 16kHz, or 8kHz audio.
- The I2S0 interface, when connected to HDMI for PCM output, supports sampling rates of 96kHz, 48kHz, 44.1kHz, and 32kHz. The data bit widths supported are 16bit, 24bit, and 32bit, and it supports stereo channels and little-endian processing.

5.2 Device Node Correspondence

Based on the different hardware peripherals of the **EIC770X SOC** development boards, the hardware configuration varies. Taking the **EIC770X-EVB-1** development board as an example, it has a total of 7 PCM nodes, including 4 playback nodes and 3 recording nodes. The correspondence between the nodes and the peripherals is shown in the diagram below.

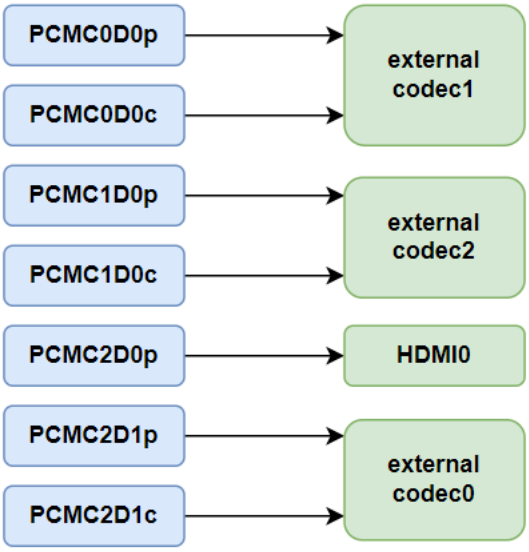


Figure 5-3 Audio node correspondence

Note: The **EIC770X-EVB-1** development board uses the same I2S0 for CODEC0 and HDMI, so only one device can be active at a time. Therefore, PCMC2D0p and PCMC2D1p are mutually exclusive, and only one node can be enabled at a time. If there is a requirement for more than three audio inputs and outputs, it is recommended to use a USB hub to expand the number of audio channels.

The correspondence between the audio device nodes and peripherals on different development boards based on the **EIC770X SOC** is shown in the table below.

Table 5-1 Correspondence Between Audio Nodes and Peripherals on Different Development Boards

Audio Node	EVB-1 Peripherals on Development Boards	EVB-2 Peripherals on Development Boards	DVB Peripherals on Development Boards
PCMC0D0p	CODEC0 Output	CODEC0 Output	CODEC0 Output
PCMC0D0c	CODEC0 Input	CODEC0 Input	CODEC0 Input
PCMC1D0p	CODEC1 Output	CODEC1 Output	HDMI Output
PCMC1D0c	CODEC1 Input	CODEC1 Input	X
PCMC2D0p	HDMI Output	HDMI Output	X
PCMC2D1p	CODEC2 Output	X	X
PCMC2D1c	CODEC2 Input	X	X

5.3 Audio Hardware Debugging

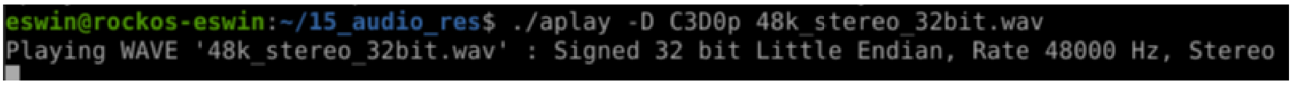
In the audio configuration file /etc/asound.conf, C0D0p to C5D0p are virtual PCM (Pulse Code Modulation) nodes defined based on the ALSA (Advanced Linux Sound Architecture) system. These virtual nodes are defined using the "plug" type in the configuration file, mapped to the corresponding physical audio hardware devices, and the dmix plugin is utilized to implement the audio stream mixing function. In this way, it supports multiple applications to share the audio device and efficiently manage audio resources. Each virtual node is mapped to a specific audio hardware device, as shown in the following table. The virtual nodes have the following main functions:

- Audio Device Abstraction
  Each node encapsulates the underlying hardware device, simplifying the access of upper-layer applications to the audio hardware. Through virtualization, it masks the differences between hardware devices and provides a unified interface for developers.
- Dynamic Audio Mixing
  The dmix plugin is used to provide a mixing function for each hardware device (such as multiple processes sharing one device). Through the virtual nodes, it supports multiple audio streams to be output to the same physical device simultaneously, without the need for the hardware to support multi-streaming.
- Format Conversion and Adaptation
  The plug plugin supports automatic format conversion, such as adjustments of the sampling rate, bit width, and the number of channels, to ensure that the input audio stream matches the device

Table 5-2 Comparison of Virtual Nodes

Node Name	Physical Device	Describe	Notes
C0D0p	hw:0,0	corresponding to Physical Device 0, processes the audio output stream.	Valid on one die and dual die
C1D0p	hw:1,0	corresponding to Physical Device 1, processes the audio output stream.	Valid on one die and dual die
C2D0p	hw:2,0	corresponding to Physical Device 2, processes the audio output stream.	Valid on one die and dual die
C3D0p	hw:3,0	corresponding to Physical Device 3, processes the audio output stream.	Valid on dual die
C4D0p	hw:4,0	corresponding to Physical Device 4, processes the audio output stream.	Valid on dual die
C5D0p	hw:5,0	corresponding to Physical Device 5, processes the audio output stream.	Valid on dual die

Examples of virtual node playback：./aplay -D C3D0p 48k_stereo_32bit.wav



```
eswin@rockos-eswin:~/15_audio_res$ ./aplay -D C3D0p 48k_stereo_32bit.wav
Playing WAVE '48k_stereo_32bit.wav' : Signed 32 bit Little Endian, Rate 48000 Hz, Stereo
```

6. Audio Dual-DIE Input and Output

The framework of dual DIE audio is shown as follows.

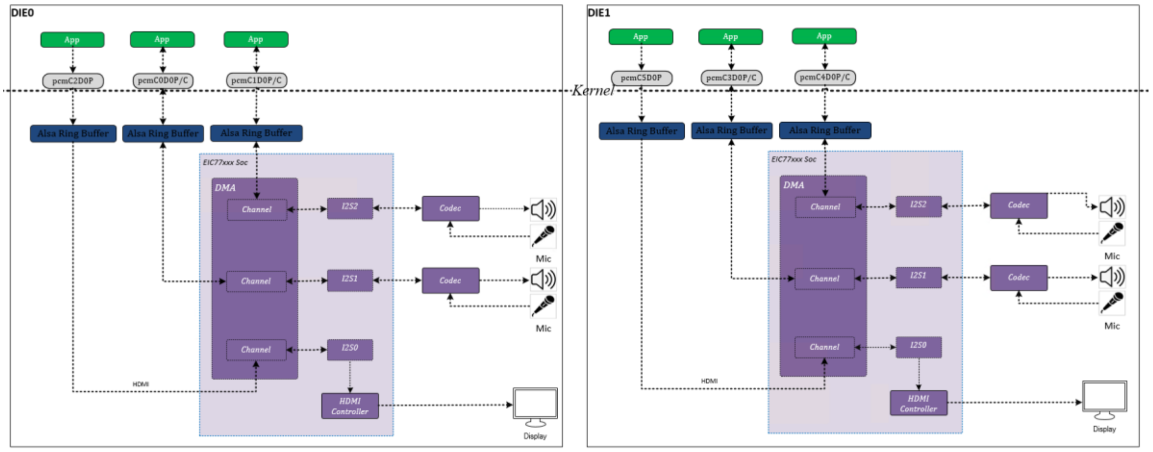


Figure 6-1 Dual DIE Audio Framework

The EIC77X is available in a dual-die chip version. The dual-die chip integrates two sets of identical hardware resources in the SoC architecture and operates on the same operating system. The hardware addresses are linearized in the system, enabling the CPU to directly and efficiently access these resources.

The available resources of the dual-die chip are doubled. For audio hardware, the number of I2S devices increases to six, and the corresponding audio nodes also increase accordingly. The correspondence between the audio nodes and peripherals on the dual - DIE development board is shown in the following table.

Table 6-1Correspondence Between Audio nodes and Peripherals on the Dual-DIE development board

Audio Node	Peripherals on Dual DIE Development Boards	Notes
PCMC0D0p、PCMC0D0c	CODEC0 Output、CODEC0 Input	Valid on one die and dual die
PCMC1D0p、PCMC1D0c	CODEC1 Output、CODEC1 Input	Valid on one die and dual die
PCMC2D0p	HDMI0 Output	Valid on one die and dual die
PCMC3D0p、PCMC3D0c	CODEC2 Output、CODEC2 Input	Valid on dual die
PCMC4D0p、PCMC4D0c	CODEC3 Output、CODEC3 Input	Valid on dual die
PCMC5D0p	HDMI1 Output	Valid on dual die

In a dual-die environment, the performance of cross-DIE data access can impact the overall system performance. Therefore, during system operation, data transmission across DIEs should be minimized to improve the overall operational efficiency. Consequently, in the software, applications from CARD0 to CARD2 are configured to be processed on DIE 0, while those from CARD3 to CARD5 are configured to be processed on DIE 1.

7. Development Process

The audio SDK provided by the **EIC770X** includes the necessary header files and libraries for development. Additionally, the SDK offers a reference example for audio development named sample_audio. Based on sample_audio, developers can carry out the development of the AI, AO, ADEC, and AENC module SDKs. The relevant files of the audio SDK development package are as follows.

```
├── bin
│   └── sample_audio
├── include
│   └── audio
│       ├── es_audio.h
│       ├── es_comm_adec.h
│       ├── es_comm_aenc.h
│       ├── es_comm_aio.h
│       └── es_debug.h
├── lib
│   └── audio
│       ├── libes_audio_control.so
│       ├── libes_audio_hal.so
│       ├── libes_audio_processing.so
│       ├── libes_audio_sdk.so
│       ├── libes_audio.so
│       ├── libes_codec_processor.so
│       ├── libes_codec.so
│       ├── libes_params_convert.so
│       ├── libes_vqe.so
│       ├── libes_aec.so
│       ├── libes_agc.so
│       ├── libes_hpf.so
│       ├── libes_drc.so
│       ├── libes_eq-iir.so
│       ├── libes_mux.so
│       ├── libes_ns.so
│       ├── libes_src.so
│       ├── libes_volume.so
```

```
└── samples
    └── audio_sample
        ├── adp
        │   ├── audio_adpcm_adp.c
        │   └── audio_adpcm_adp.h
        ├── utils
        │   ├── utils.c
        │   └── utils.h
        ├── build.sh
        ├── CMakeLists.txt
        ├── README.md
        ├── data
        │   └── audio
        │       ├── 48000_16bit_setreo_4000_10000ms.wav
        │       ├── 48000_32bit_setreo_4000_10000ms.wav
        │       └── music.wav
        ├── sample_audio.c
        ├── sample_comm_audio.c
        └── sample_comm.h
```

The following is an explanation of the SDK development package directory:

- bin： sample_audio is the executable file of the reference example, which can be tested on the development board to verify the functions of AI, AO, AENC, and ADEC.
- include/audio：It contains the header files on which the SDK development depends. These header files need to be included when writing code. Among them, es_audio.h is the interface header file for the AI, AO, AENC, and ADEC modules.
- lib/audio：It contains the dependent libraries, including the algorithm libraries related to VQE. These algorithm libraries are required at runtime.
- samples/audio_sample：The source files and test files corresponding to the reference example sample_audio. By compiling based on the provided source code, the executable file sample_audio can be generated, which is used to test the relevant AUDIO functions on the development board.

The following are the development examples corresponding to sample_audio:

- For AI-related examples, please refer to SAMPLE_AUDIO_AI. This example demonstrates the overall process and usage of audio recording.
- For AO development-related examples, please refer to SAMPLE_AUDIO_AO. This example demonstrates the overall process and usage of audio playback.
- For AENC development-related examples, please refer to SAMPLE_AUDIO_AENC. This example demonstrates the overall process and usage of audio encoding.
- For ADEC development-related examples, please refer to SAMPLE_AUDIO_ADEC. This example demonstrates the overall process and usage of audio decoding
- For system binding-related examples, please refer to SAMPLE_AUDIO_AiAo, SAMPLE_AUDIO_AiAenc, SAMPLE_AUDIO_AdecAo, etc. These examples illustrate the calling process of system binding and its usage.