



Airoha IoT SDK DSP Audio Algorithm Developers Guide

Version: 1.1
Release date: 19 July 2023

Use of this document and any information contained therein is subject to the terms and conditions set forth in Exhibit 1. This document is subject to change without notice.

Document Revision History

Revision	Date	Description
1.0	31 May 2023	Initial release
1.1	19 July 2023	Add additional details and descriptions to enhance the information

Table of Contents

Document Revision History.....	2
Table of Contents.....	3
List of Figures.....	4
List of Tables.....	4
1 Introduction	5
1.1 BT Audio SDK.....	5
1.2 SW Architecture	5
1.3 SW tasks in DSP SDK.....	8
1.4 The communication from MCU to DSP	10
1.5 The communication from DSP to MCU	14
1.6 Debug methods	15
2 SW developer flow for algorithm porting	16
2.1 Life cycle of algorithm in DSP audio framework	16
2.2 Audio dump introduction.....	18
2.3 Frequently used APIs.....	20
2.4 Practice: Familiar with 1 st example project	21
3 BT Audio DSP SDK.....	23
3.1 Audio Chain introduction	23
3.2 Audio format in BT audio scenarios	26
3.3 Practice to add custom function	27
3.4 Available algorithms in DSP SDK	27
3.5 Available basic functions in DSP SDK.....	29
4 FAQ	31
4.1 How to handle frame size mismatch?	31
4.2 How to handle sampling rate mismatch?.....	31
4.3 How to pass parameter or control msg from mcu to dsp side?	32
5 Terms and definitions	34
6 Appendix	36
6.1 DSP algorithm's location in SDK	36
6.2 MCPS profiling method in DSP	36
6.3 Algorithm's memory location in DSP.....	36
Exhibit 1 Terms and Conditions.....	38

List of Figures

Figure 1. The SDK architecture at MCU side	7
Figure 2. The audio scenario control example	7
Figure 3. The SDK architecture at DSP side.....	8
Figure 4. The task priority definition in FreeRTOS	9
Figure 5. Audio message data structure stored in ccni message format	11
Figure 6. SW sequence in DSP's audio framework.....	17
Figure 7. Enable dump ID in the config tool	19
Figure 8. Audio dump in logging tool UI	19
Figure 9. Audio dump in the logging tool folder	20
Figure 10. Audio dump drop in the logging tool folder	20
Figure 11. Audio chain for music playback scenario.....	23
Figure 12. Audio chain for call scenario.....	24
Figure 13. The flow of sending paramters from MCU to DSP	32
Figure 14. The example flow of passing parameters in PEQ module.....	32

List of Tables

Table 1. Audio SW partitions between MCU and DSP	5
Table 2. SW tasks in DSP side.....	8
Table 3. Audio message structure.....	10
Table 4. The message control example code in voice prompt scenario.....	11
Table 5. The message control example code in voice prompt scenario.....	12
Table 6. The message control example code in A2DP reinit scenario.....	14
Table 7. Suggested steps about algorithm integration	17
Table 8. Not allowed operations inside algorithm	18
Table 9. Frequently used APIs for algorithm developers	20
Table 10. Audio format in BT audio scenario	26
Table 11. The description of available dsp algorithms in SDK.....	27
Table 12. Description of basic dsp functions available in SDK	29
Table 13. Control msg from mcu to dsp side example code in PEQ scenario.....	32
Table 14. Terms and definitions	34
Table 15. Mentioned algorithm modules in SDK path	36

1 Introduction

The purpose of this documentation is to provide developers who are working with the Airoha AB158X platform with a clear understanding of the Audio SDK. It also provides instructions and reference material for developers who want to port their own audio algorithms to the Audio SDK.

1.1 BT Audio SDK

Airoha platform provides comprehensive technology including hardware evaluation kit (EVK) and software development kit (SDK) to design and develop Bluetooth Audio applications. And it covers A2DP, HFP, LE Audio. The target platform will be AB1585 series.

This document guides you through:

- The audio software architecture.
- Supported modules/components (description of software modules).
- How to integrate customer's algorithm into Airoha's SDK.

1.2 SW Architecture

In Airoha IoT SDK, there are two processors, one is micro-controller unit (MCU) designed for application usage and the other is digital signal processor (DSP) for audio algorithm development. From Audio's point of view, here is the explanation about where the audio SW functions are handled.

Table 1. Audio SW partitions between MCU and DSP

Function name	Description	Function location
Retrieve audio parameters from flash	The audio parameters mean <ul style="list-style-type: none"> – Audio input/output device settings – Audio sample settings (Sampling rate, frame size, and bits per sample) – The volume table for different scenarios – Algorithm's parameters 	MCU
Scenario control	The scenario control is mainly handled by Audio Manager module and it maintains <ul style="list-style-type: none"> – The priority table between different audio scenarios <ul style="list-style-type: none"> ▪ Before a higher priority scenario is started, the lower priority scenario will be suspended. ▪ After a higher priority scenario is closed, the lower 	MCU

Airoha IoT SDK DSP Audio Algorithm Developers Guide

Function name	Description	Function location
	priority scenario will be resumed.	
Audio Algorithm	The audio algorithms typically include the following categories <ul style="list-style-type: none"> - Codec (Encoder/Decoder) - Post processing for audio - Acoustic processing for voice - Voice Assistant 	MCU/DSP
Audio Device Driver	The audio device driver controls <ul style="list-style-type: none"> - audio path for downlink Digital to Analog Conversion (DAC) and volume control - audio path for uplink Analog to Digital Conversion (ADC) and volume control - audio path for digital I/O (I2S, Digital Microphone) - other misc functions in Audio HW including sampling rate conversion, sidetone filter and DMAs for both uplink and downlink - SRAM memory management in Audio HW 	DSP

At MCU side, the related audio SW blocks (please refer to the audio section within the highlighted box in Figure 1) are mainly handling several things: scenario control, voice prompt with mp3 decoder, and voice assistant with audio record function.

It is important to note that as the SDK undergoes version upgrades, the information outside the highlighted box in Figure 1 below may change or undergo modifications.

- Located in middleware layer:
 - Audio Manager is the module to manage the priority between scenarios according to the definition by users.
 - Record control is to provide the interface for mic record function.
 - Prompt control is to provide the voice prompt function which supports mp3, wav and opus format.

Airoha IoT SDK DSP Audio Algorithm Developers Guide

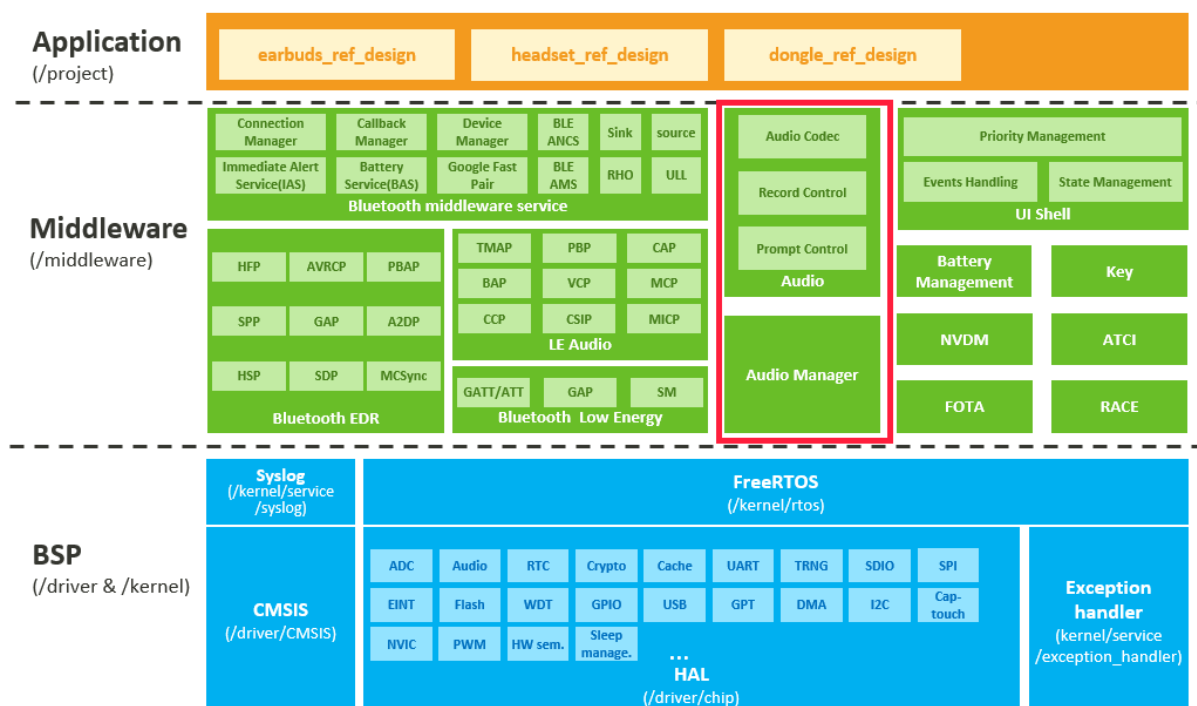


Figure 1. The SDK architecture at MCU side

Here is the example of audio scenario control:

- 1) A2DP is activated first.
- 2) When the HFP call is trying to make a connection, the application and audio manager will decide to suspend the A2DP scenario first. After that, the HFP call will be activated.
- 3) After the HFP call is ended, the application and audio manager will resume the A2DP scenario.



Figure 2. The audio scenario control example

At DSP side, it mainly handles the following tasks (refer to the audio section in the highlighted box in Figure 3):

- In driver layer: it mainly contains the low level operations to interact with the hw registers.
 - Control the audio i/o (Analog interface: DAC, ADC. Digital interface: I2S master/slave and digital mic.)
 - Control the hw functions in Audio part: HWSRC, hw gain, etc.
- In middleware layer: it mainly contains the dsp framework to implement the following features
 - Handle message control between mcu and dsp
 - Handle the legacy Bluetooth scenarios audio stream: A2DP, HFP, LE Audio.
 - Handle the audio playback and record scenario: voice prompt and voice assistant.

Airoha IoT SDK DSP Audio Algorithm Developers Guide

It is important to note that as the SDK undergoes version upgrades, the information outside the highlighted box in Figure 3 may change or undergo modifications.

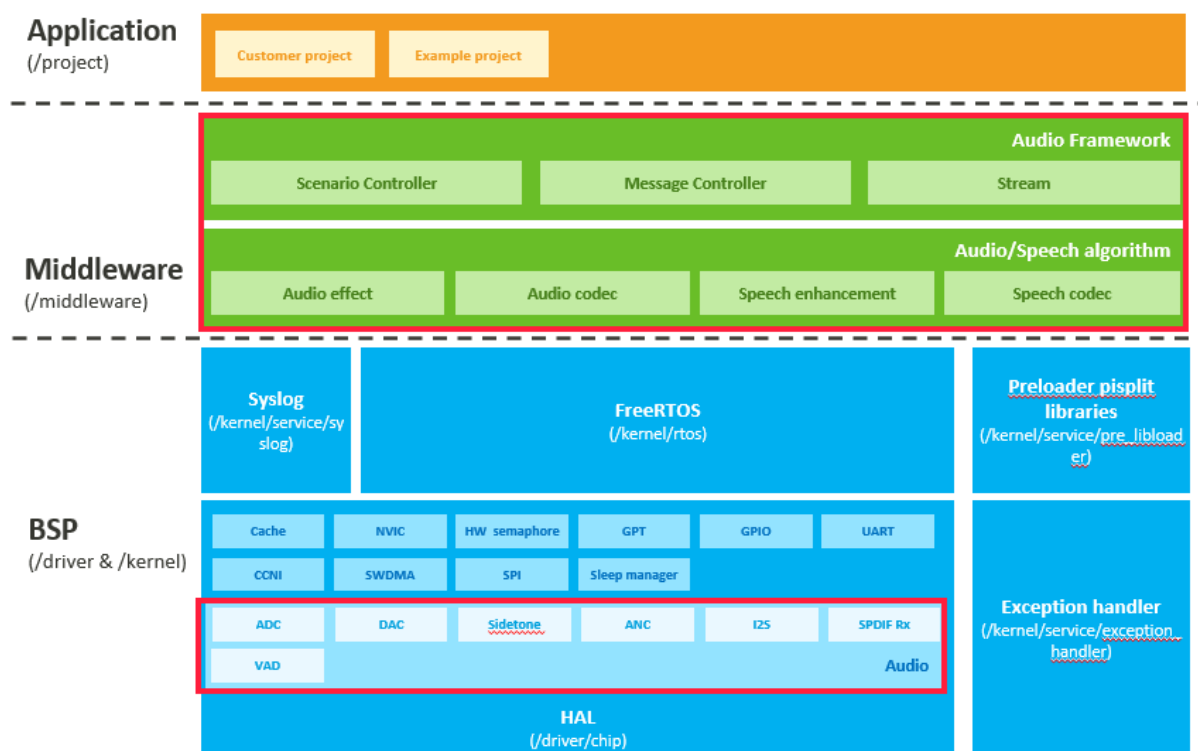


Figure 3. The SDK architecture at DSP side

1.3 SW tasks in DSP SDK

This section primarily focuses on SW tasks in DSP SDK. It introduces the scenarios (or functions) that each task runs and how to assign a specific stream to a particular task.

The list of relevant tasks and corresponding introduction on the DSP SDK is as follows (two of them are freeRTOS build-in task).

Table 2. SW tasks in DSP side

Task name	Description	Task Priority
(RX) Audio message handler	DSP's message handler task for the received MCU messages through the Cross Core Notification Interface (CCNI)	TASK_PRIORITY_SOFT_REALTIME
(TX) Audio message handler	DSP's message handler task for transmitting msg to MCU through CCNI	TASK_PRIORITY_SOFT_REALTIME
DSP High Priority (DHP) Task	DSP's high priority task for the following scenarios: <ul style="list-style-type: none"> Wired Audio playback from 3.5mm 	TASK_PRIORITY_ABOVE_NORMAL

Airoha IoT SDK DSP Audio Algorithm Developers Guide

Task name	Description	Task Priority
	port for headset formfactor project	
DSP Audio Voice (DAV) Task	DSP's audio/voice task for legacy scenarios: <ul style="list-style-type: none"> ○ A2DP, HFP ○ LE Audio 	TASK_PRIORITY_NORMAL
DSP Task Management (DTM) Task	Currently, reserved for new feature development	TASK_PRIORITY_NORMAL
DSP Priority (DPR) Task	DSP's low priority task to handle the following scenarios: <ul style="list-style-type: none"> ○ Voice prompt playback for notification ○ Voice record 	TASK_PRIORITY_BELOW_NORMAL
FreeRTOS IDLE Task	It is a default task in FreeRTOS that executes whenever the scheduler has no other tasks to execute, it has the lowest possible priority, so it can be pre-empted by any other task in the system	TASK_PRIORITY_IDLE
FreeRTOS TIMER Task	It is a built-in task in FreeRTOS that is responsible for handling software timers	TASK_PRIORITY_TIMER

The task priority configuration in FreeRTOS is defined in task_def.h in the following figure.

```
typedef enum {
    TASK_PRIORITY_IDLE = 0,                /* lowest, special for idle task */
    TASK_PRIORITY_SYSLOG,                  /* special for syslog task */

    /* User task priority begin, please define your task priority at this interval */
    TASK_PRIORITY_LOW,                     /* low */
    TASK_PRIORITY_BELOW_NORMAL,             /* below normal */
    TASK_PRIORITY_NORMAL,                  /* normal */
    TASK_PRIORITY_ABOVE_NORMAL,            /* above normal */
    TASK_PRIORITY_HIGH,                    /* high */
    TASK_PRIORITY_SOFT_REALTIME,           /* soft real time */
    TASK_PRIORITY_HARD_REALTIME,          /* hard real time */
    /* User task priority end */

    /*Be careful, the max-priority number can not be bigger than configMAX_PRIORITIES - 1, or kernel will crash!!! */
    TASK_PRIORITY_TIMER = configMAX_PRIORITIES - 1, /* highest, special for timer task to keep time accuracy */
} task_priority_type_t;
```

Figure 4. The task priority definition in FreeRTOS

The default setting of a new unspecified scenario will be dispatched to DAV Task. To modify scenario priority, User may need to assign scenario to different DSP task. Currently DHP/DAV/DPR Task is available to run audio scenario.

```
TaskHandle_t DSP_Callback_Config(SOURCE source, SINK sink, VOID
*feature_list_ptr, BOOL isEnabled)
```

Airoha IoT SDK DSP Audio Algorithm Developers Guide

If the default scenario priority is not feasible. User can reassign stream to different task by adding code into below API.

A straightforward example to modify voice prompt scenario priority is available in the API. After discovering current scenario is for voice prompt. Modify the task to run the callback.

Some sample code exist in DSP_Callback_Config is as below:

```
/* C Sample Program */
if (sink->type == SINK_TYPE_VP_AUDIO)
// discovering current scenario is voice prompt or not
{
    Source->taskId = DPR_TASK_ID; // Reassign task to run callback
    Sink->taskId = DPR_TASK_ID; // Reassign task to run callback
}
```

1.4 The communication from MCU to DSP

The communication from MCU to DSP is through message control which is implemented via the CCNI HW.

Step#1: Users need to define the message IDs and implement its message handlers.

- Defined in hal_audio_cm4_dsp_message.h
 - MCU to DSP: "MSG_MCU2DSP_xxx"
 - DSP to MCU: "MSG_DSP2MCU_xxx"

Step#2: Users put the message via the api: hal_audio_dsp_controller_send_message()

```
void hal_audio_dsp_controller_send_message (uint16_t message, uint16_t data16,
uint32_t data32, bool wait)
```

- Defined in mcu/driver/chip/ab1585/src/hal_audio_dsp_controller.c
- The audio message's data structure is as shown in the following table.

Table 3. Audio message structure

Field name	Details	Example
message (or message id)	The MSG ID must be defined both in both size of the source/destination core	1. Pass "set volume" message and store vol value in data32 field
data16	to describe sub-id or format	hal_audio_dsp_controller_send_message(MSG_MCU2DSP_COMMON_SET_OUTPUT_DEVICE_VOLUME , hw_gain_index, vol, false)
data32	to describe the address of share memory	
wait	The bit decides this msg is an ack or not <ul style="list-style-type: none"> ▪ bit 1: it is an ack msg ▪ bit 0: it is not an ack msg 	2. Pass "open voice prompt" message and store the shared memory address in data32 file. You must copy the parameters to the share memory before sending audio message. hal_audio_dsp_controller_send_message(MSG_MCU2DSP_PLAYBACK_OPEN, AUDIO_DSP_CODEC_TYPE_PCM,

Airoha IoT SDK DSP Audio Algorithm Developers Guide

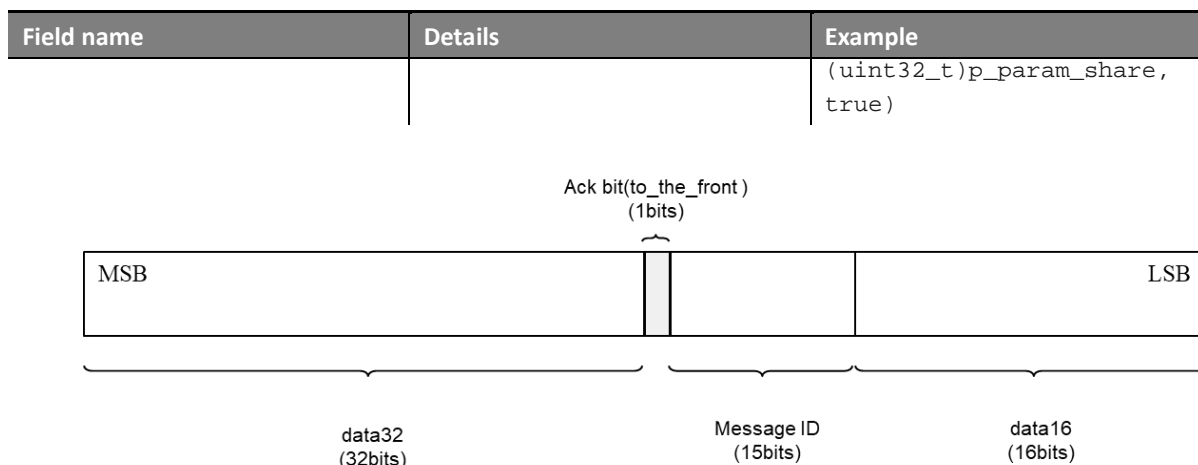


Figure 5. Audio message data structure stored in ccni message format

Step#3: Register the callback function on the dsp side.

```
aud_msg_status_t aud_msg_rx_register_callback(mcu2dsp_audio_msg_t ID, void
*rx_callback, uint8_t ack_opt)
```

- Defined in dsp/middleware/airoha/dspfw/port/chip/ab1585/dsp_temp.c
- The introduction for each parameter of the function `aud_msg_rx_register_callback()` is shown in the table below.

Table 4. The message control example code in voice prompt scenario

Field name	Details	Example
ID	The MSG ID must be defined in both size of the MCU and DSP	<ol style="list-style-type: none"> 1. Receive “set volume” message and store vol value in data32 field <code>aud_msg_rx_register_callb ack(MSG_MCU2DSP_COMMON_SE T_OUTPUT_DEVICE_VOLUME, DSP_GC_SetOutputVolume, 2)</code> 2. Receive “open voice prompt” message and store the shared memory address in data32 filed. You must get the parameters in the share memory before ack audio message.) <code>aud_msg_rx_register_callb ack(MSG_MCU2DSP_PLAYBACK_ OPEN, CB_CM4_PLAYBACK_OPEN, 2);</code>
rx_callback	The callback function for the corresponding MSG ID.	
ack_opt	The value decides how to ack this msg <ul style="list-style-type: none"> ▪ 0: execute callback function after response ACK. ▪ 1: not ack this msg ▪ 2: execute callback function before response ACK. 	

Airoha IoT SDK DSP Audio Algorithm Developers Guide

The transmission process of msg sending from MCU to DSP is as below, and each of the following steps contributes some latency to the CCNI msg communication from MCU to DSP:

- Msg enqueue and send to DSP by CCNI HW.
 - If the ack is enabled in MCU side by setting the parameter “wait” in the API `hal_audio_dsp_controller_send_message()`, a busy waiting flag should be set when the CCNI msg is send and be cleared after the corresponding ack msg is received, during this period the thread which `hal_audio_dsp_controller_send_message()` runs on should be stalled, If the MCU does not receive the ack within 2 seconds, MCU will trigger assert.
- The rx_callback function will be executed in the DSP RX task when the CCNI msg is received by DSP.
- If the ack is enabled in DSP side by setting the parameter “ack_opt” in the API `aud_msg_rx_register_callback()`, the ack msg will be send in the DSP TX task, then MCU can receive the ack msg.

Refer the example codes from the voice prompt scenario in the following table.

Table 5. The message control example code in voice prompt scenario

Modules	File path	Contents
[MCU] Defined messages	common/driver/chip/ab1585/inc/ hal_audio_cm4_dsp_message.h	MSG_MCU2DSP_PROMPT_OPEN
[MCU] copy the open parameters into share memory	mcu/middleware/airoha/audio/mp3_codec/src/mp3_codec.c	<code>hal_audio_dsp_cntrller_put_parameter(&open_param, size_of_open_parameters, msg_type)</code>
[MCU] send msg	mcu/middleware/airoha/audio/mp3_codec/src/mp3_codec.c	<code>hal_audio_dsp_controller_send_message(open_msg, AUDIO_DSP_CODEC_TYPE_PCM, (unit32_t)p_param_share, true)</code>
[DSP] register callback function	dsp/middleware/airoha/dspfw/port/chip/ab1585/dsp_temp.c	<code>aud_msg_rx_register_callback(MSG_MCU2DSP_PROMPT_OPEN, CB_CM4_VP_PLAYBACK_OPEN, 2)</code>
[DSP] message handler function	dsp/middleware/airoha/dspfw/port/chip/ab1585/src/dsp_scenario.c	<code>void CB_CM4_VP_PLAYBACK_OPEN(hal_ccni_message_t msg, hal_ccni_message_t *ack)</code>

Here is the example code of HFP to control the audio i/o from MCU.

in the SDK, change the Speaker and MIC path in the following file:

- mcu/middleware/airoha/audio/bt_codec/src/bt_hfp_codec.c

SDK API: bt_hfp_open()

Here is the open param structure and the basic explanation of each member which will be used for controlling the audio I/O

/ Open message parameter structure */*

typedef struct {

```
    mcu2dsp_param_t      param;
    audio_scenario_type_t audio_scenario_type;
    mcu2dsp_open_stream_in_param_t stream_in_param;
    mcu2dsp_open_stream_out_param_t stream_out_param;
} mcu2dsp_open_param_t, *mcu2dsp_open_param_p;
```

param: set stream in/stream out selection.

audio_scenario_type: audio scenario type, each audio scenario has its own type.

stream_in_param: param for source.

stream_out_param: param for sink.



For the full definition of the structure members, please reference to the header file: common/driver/chip/ab158x/inc/hal_audio_message_struct_common.h in the SDK.

Speaker: Configure the following two parameters in HFP Downlink config code flow:

- open_param->stream_out_param.afc.audio_device
- open_param->stream_out_param.afc.audio_interface

MIC: Config the following two parameters in the HFP Uplink config code flow:

- open_param->stream_in_param.afc.audio_device
- open_param->stream_in_param.afc.audio_interface

Here is the example of the speaker's configuration:

- open_param->stream_out_param.afc.audio_device
 - = HAL_AUDIO_DEVICE_DAC_L (speaker L)
 - = HAL_AUDIO_DEVICE_DAC_R (speaker R)
 - = HAL_AUDIO_DEVICE_DAC_DUAL (speaker L+R)
- open_param->stream_out_param.afc.audio_interface
 - = HAL_AUDIO_INTERFACE_1 (audio interface path 1)
 - = HAL_AUDIO_INTERFACE_2 (audio interface path 2)
 - = HAL_AUDIO_INTERFACE_3 (audio interface path 3)

Please be aware that the modification of audio I/O is usually achieved by using the config tool, not by modifying the code directly.

1.5 The communication from DSP to MCU

The communication from DSP to MCU is also through message control which is implemented via the CCNI HW.

Step#1: Users need to define the message IDs and implement its message handlers.

- Defined in hal_audio_cm4_dsp_message.h
 - MCU to DSP: "MSG_MCU2DSP_xxx"
 - DSP to MCU: "MSG_DSP2MCU_xxx"

Step#2: Users put the message via the api: aud_msg_tx_handler ()

```
aud_msg_status_t aud_msg_tx_handler(hal_ccni_message_t msg, uint8_t to_the_front,
uint8_t from_ISR)
```

- Defined in dsp/middleware/airoha/dspfw/port/chip/ab158x/src/dsp_audio_msg.c

Table 6. The message control example code in A2DP reinit scenario

Field name	Details	Example
message (message id, data 16, data 32)	The MSG ID must be defined both in both size of the source/destination core	1. Pass "A2DP reinit" message and send data in data32 field. hal_ccni_message_t msg; memset((void *)&msg, 0, sizeof(hal_ccni_message_t)); msg.ccni_message[0] = MSG_DSP2MCU_BT_AUDIO_DL_REINIT_REQUEST << 16;
to_the_front	Is this msg send first. <ul style="list-style-type: none"> ▪ bit 1: send this msg at first. ▪ bit 0: send this msg in order. 	msg.ccni_message[1] = reinit_msg;
from_ISR	Is the msg send in interrupt handler <ul style="list-style-type: none"> ▪ bit 1: send from ISR. ▪ bit 0: not send from ISR. 	aud_msg_tx_handler(msg, 0, FALSE);

Step#3: Add code to receive messages on the mcu side.

```
static void hal_audio_service_callback(audio_message_element_t *p_msg)
```

- Defined in mcu/driver/chip/ab158x/src/hal_audio_dsp_controller.c
- add your function to receive message by add switch case, in the function above.

The transmission process of msg sending from DSP to MCU is as below, and each of the following steps contributes some latency to the CCNI msg communication from DSP to MCU:

- Msg enqueue and send to TX task by calling the API aud_msg_tx_handler().
- TX task get the msg, send it to MCU by CCNI HW.
- MCU get the msg by CCNI IRQ, then send the msg to AM task to process.

1.6 Debug methods

SDK provides several methods for debugging.

- Logging Tool for capture string log and raw audio dump (which is designed by SW)
- SW implement GPIO debug signal to use Logic Analyzer for timing analysis.
- JTAG for on-chip debugging.
- Raise exception in SW and analyze exception dumps from the logging tool part.



For Logging Tool, you can refer to “SDK_Toolkit(ATK)\doc\xxx_Sereies_Logging_Tool_User_Guide.pdf”

For GPIO, you can refer to the GPIO module document in the SDK’s api reference.

2 SW developer flow for algorithm porting

2.1 Life cycle of algorithm in DSP audio framework

Here is the definition of stream feature.

```
typedef struct stream_feature_s {
    uint32_t feature_type : 8;          /**< The feature Type. */
    uint32_t memory_size : 24;          /**< The instance memory(working
buffer) size reserved for feature usage. */
    uint32_t codec_output_size;          /**< The codec output size for
streaming buffer allocate, Set 0 for functions. */
    stream_feature_entry_t initialize_entry; /**< The Initializing entry for
procedure. */
    stream_feature_entry_t process_entry;  /**< The Processing entry for
procedure. */
}
```

- Defined in dsp/middleware/airoha/dspfw/port/chip/ab158x/src/dsp_sdk.c
- For example: stream_feature_sample_function / stream_feature_sample_codec, defined in dsp_sdk.c

If you want to add your feature in stream process flow, you need to add your feature id into the stream feature list.

- DSP FW will process features in the stream feature list one by one. For example: if you want to add sample function to A2DP stream, you need to add "FUNC_SAMPLE" into "stream_feature_list_a2dp", at the right place.

Here is the typical flow in audio framework of DSP.

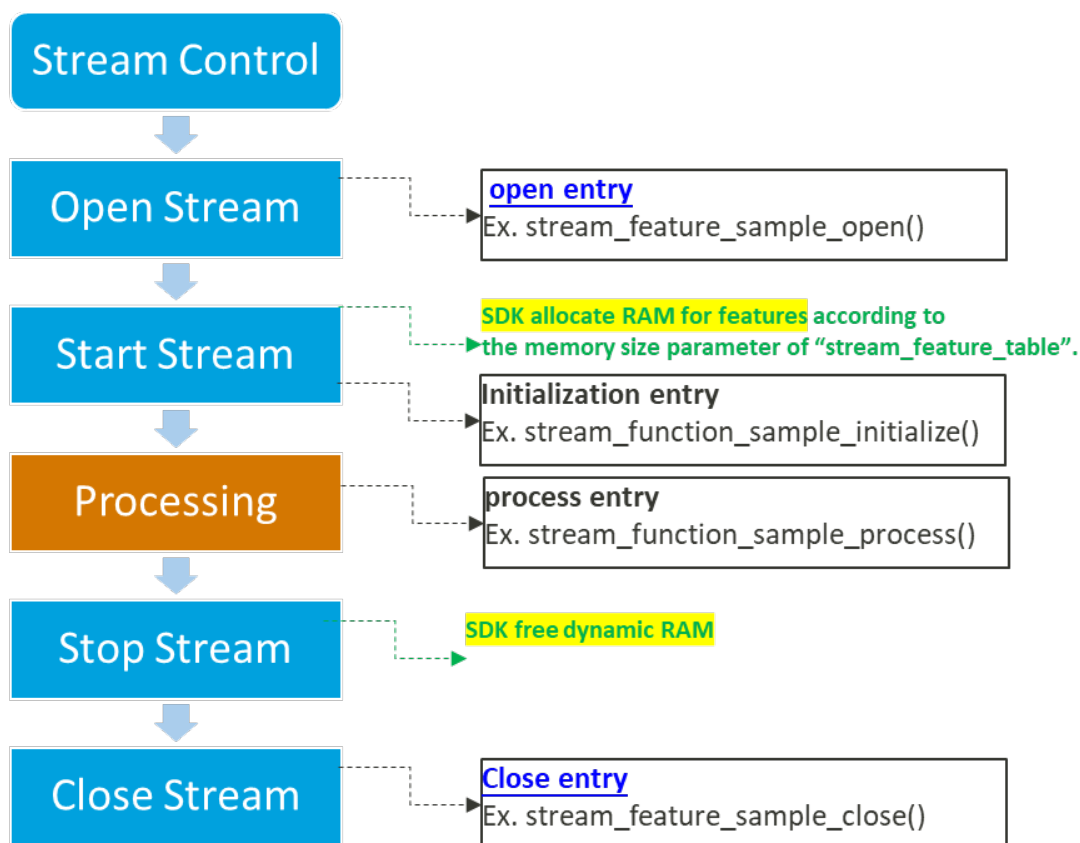


Figure 6. SW sequence in DSP's audio framework

- Stream Control: MCU sends msg to DSP to switch the stream on/off.
 - Open stream: set DSP HW param, load PIC lib, etc.
 - Start stream: framework will help user malloc memory, the size "memory_size" defined in stream feature. Then framework will execute initialize function, "initialize_entry" defined in stream feature.
 - Processing: framework will execute process function at each processing cycle, "process_entry" defined in stream feature.
 - Stop stream: framework will help user free memory which malloc when start stream.
 - Close stream: unload PIC lib, etc.

2.1.1 Steps about integration

Please refer to the suggested procedure for the algorithm integration into the SDK.

Table 7. Suggested steps about algorithm integration

Steps	Details	Note
1. Dev algorithm on simulator	Generate algorithm library from the DSP simulator	Memory allocation by SDK. The working buffer pointer get from <code>stream_function_get_working_buffer()</code> .
2. Prepare porting interface on SDK	Coding feature interface to integrate codec or feature	NA

Steps	Details	Note
	Link feature interface and apply feature to existing DSP scenario	NA
3. Run algorithm on EVK	Check on-chip result	NA

2.1.2 Not allowed operations inside algorithm

Please refer the list about the disallowed operations inside the algorithm.

Table 8. Not allowed operations inside algorithm

List	Reason	Note
Do memory allocation inside algorithms	malloc()/free() will use the os heap and its size is not enough for the algorithm	The DSP's audio framework provides the mechanism to declare the size requirement in other way.
print log with printf()	It will consume too much time than the proprietary logging api in SDK.	NA

2.2 Audio dump introduction

This section describes the raw audio debug mechanism what we called audio dump.

The audio dump is one debugging method for identifying the audio quality. It depends on the logging mechanism and uses the logging port to transmit the debug audio dump content. So the user can check the audio content in the PC side to identify the audio quality.

In order to using the audio dump in the FW, the user should:

- include below header file

```
#include "dsp_dump.h"
```

- Call below API to do the audio dump. The 1st parameter is the dump buffer address, the 2nd parameter is the size of dump buffer, and the 3rd parameter is the dump ID. The user should use different dump ID for different dump position. The dump ID is a reference for the definition of DSP_DATADUMP_MASK_BIT.

```
void LOG_AUDIO_DUMP(U8 *audio, U32 audio_size, U32 dumpID)
```

You should use the config tool to enable the specific dump ID. After that, only the allowed dump ID can be enabled in FW.

Note that you must run up the FW firstly. And after save the setting, you must reboot the FW to active the new setting.

Airoha IoT SDK DSP Audio Algorithm Developers Guide

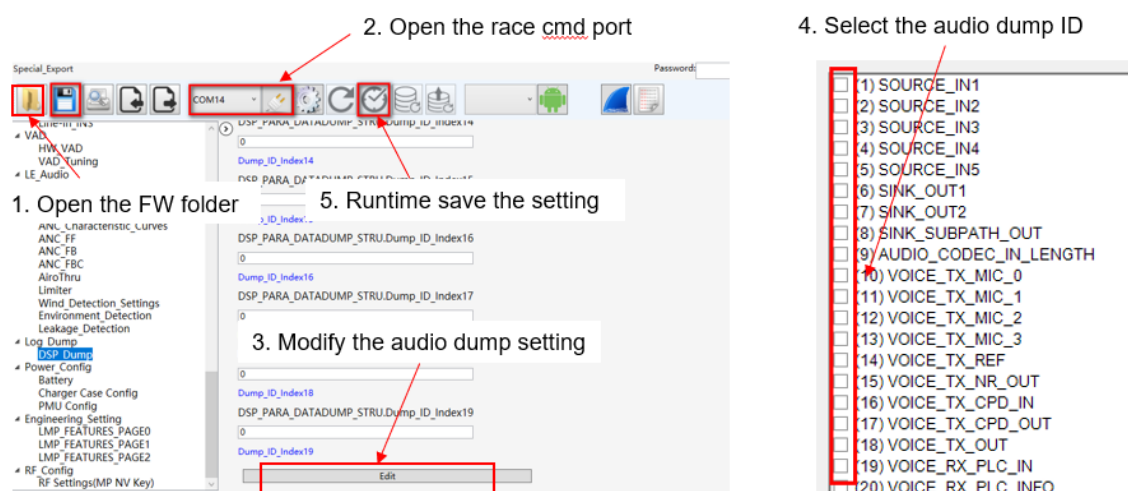


Figure 7. Enable dump ID in the config tool

Then the user should use the logging tool to capture the audio dump. When the audio dump content is sent, the user can see below information in the logging tool UI.

```

8.757241 28555 MCU 251 0 11 RACE_LOG_MSG 24 [M:PKA_LOG_DM2L C:info F: L:]: [ChA]select best LE channel =21 (Rssi=-88
8.757245 RACE_AUDIO... 328 [RACE_AUDIO_V2_DATA]
8.757248 28555 DSP 213 255 3 RACE_LOG_MSG 92 [M:u11_v2_log C:info F: L:]: [ULL Audio V2][UL][handle 0x454eeb8][source
8.757256 28556 MCU 252 0 11 RACE_LOG_MSG 24 [M:PKA_LOG_DM2L C:info F: L:]: [ChA]min_idx = 10, rssi=-88
8.757259 28556 MCU 253 0 11 RACE_LOG_MSG 24 [M:PKA_LOG_DM2L C:info F: L:]: [ChA]select best LE channel =10 (Rssi=-88
8.757261 28556 MCU 254 0 11 RACE_LOG_MSG 24 [M:PKA_LOG_DM2L C:info F: L:]: [ChA]min_idx = 21, rssi=-88
8.757264 28556 MCU 255 0 11 RACE_LOG_MSG 24 [M:PKA_LOG_DM2L C:info F: L:]: [ChA]select best LE channel =21 (Rssi=-88
8.757269 RACE_AUDIO... 328 [RACE_AUDIO_V2_DATA]
8.757271 28560 DSP 214 255 3 RACE_LOG_MSG 92 [M:u11_v2_log C:info F: L:]: [ULL Audio V2][UL][handle 0x454eeb8][source
8.757281 RACE_AUDIO... 328 [RACE_AUDIO_V2_DATA]
8.757283 28565 DSP 215 255 3 RACE_LOG_MSG 92 [M:u11_v2_log C:info F: L:]: [ULL Audio V2][UL][handle 0x454eeb8][source
8.757291 HCI_EVT Rcvd LE Meta (LE Extended Advertising Report)

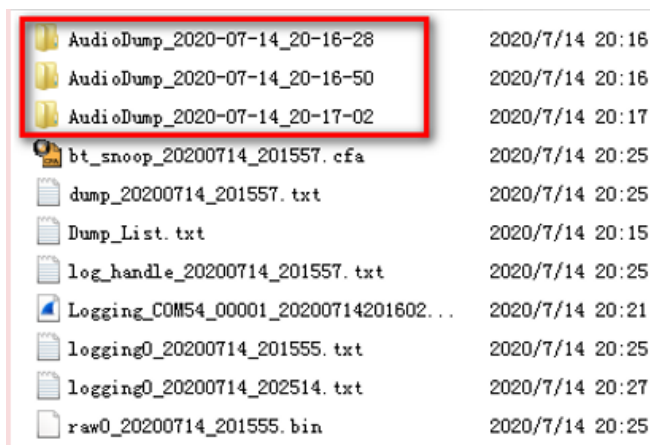
```

Figure 8. Audio dump in logging tool UI

The audio dump data will be saved in the dump folder like below, when the logging tool is keeping to capture the syslog.

- Ex: log\Logging_20200709_170304_COM54\AudioDump_2020-07-09_17-03-08

The dump data is saved to the file with the name same as audio dump type. Such as "SINK_OUT1.pcm". The dump folder is named with the timestamp, which indicates the start time of the audio dump.





 SINK_OUT1.pcm	2020/7/14 20:16	Raw PCM file	36 KB
 SINK_OUT2.pcm	2020/7/14 20:16	Raw PCM file	35 KB

Figure 9. Audio dump in the logging tool folder

If too many audio dump content is enabled at the same time, the dump may be dropped due to the shortage of logging bandwidth. The user can check it in the dump folder.



 audio_dump_drop_2023-04-14_10-29...	2023/4/15 1:30
 unknown_id_101.pcm	2023/4/15 1:30

Figure 10. Audio dump drop in the logging tool folder

2.3 Frequently used APIs

This section describes frequently used APIs.

Table 9. Frequently used APIs for algorithm developers

Function	Brief	Return value
void *stream_function_get_working_buffer(void *para)	Get the pointer of the function's memory instance [in] para, a pointer to corresponding function	memory pointer
void *stream_function_get_1st_inout_buffer(void *para)	Get 1st stream pointer for this function. [in] para, a pointer to corresponding function	Stream's buffer pointer
void *stream_function_get_2nd_inout_buffer(void *para)	Get 2nd stream pointer for this function. [in] para, a pointer to corresponding function	Stream's buffer pointer
uint8_t stream_function_get_channel_number(void *para)	Get the channel number of the stream in this function [in] para, a pointer to corresponding function	Channel numbers
uint8_t stream_function_get_output_resolution(void *para)	Get the resolution of function process [in] para, a pointer to corresponding function	Resolutions (defined in @stream_resolution_t)
uint16_t stream_function_get_output_size(void *para)	Get the size of the stream in this function [in] para, a pointer to corresponding function	The size of the frame (unit: byte)
bool stream_function_modify_output_size(void *para, uint16_t size)	Modify the resolution of function process	TRUE/FLASE

Airoha IoT SDK DSP Audio Algorithm Developers Guide

Function	Brief	Return value
	[in] para, a pointer to corresponding function [in] resolution, the value of new resolution. (defined in @stream_resolution_t)	
uint8_t stream_function_get_samplingrate(void *para)	Get the sampling rate of the stream [in] para, a pointer to corresponding function	Sampling rate (defined in @stream_samplerate_t)

Here is the example code for users.

```

/* C Sample Program for algorithm */

//get the memory address which is allocated by framework
void * working_buffer = stream_function_get_working_buffer(para);
// get the frame size (unit : bytes)
U32 FrameSize_Byte = stream_function_get_output_size(para);
//get bytes per sample
U32 BytesPerSample = (stream_function_get_output_resolution(para) ==
RESOLUTION_32BIT) ? 4 : 2;
//calculate the frame size (unit : sample)
U32 FrameSize_Sample = FrameSize_Byte / BytesPerSample;
//get the buffer address of streaming data (L);
S8* Buf_L = stream_function_get_1st_inout_buffer(para);
//get the buffer address of streaming data (R)
S8* Buf_R = (S8 *)stream_function_get_2nd_inout_buffer(para);
//get the channel number (mono:1, stereo:2)
U32 Channels = stream_function_get_channel_number(para);

```

Note:

- The audio raw data's format in DSP is fixed point stored in 16bits or 32bits. Normally, we treat it in Q1.15 or Q1.31 respectively.
- In **stream_function_get_output_size()**, the frame size unit is "Byte" and users can get the resolution via **stream_function_get_output_resolution()**. With the information, it is possible to calculate how many samples each frame contains (as in the example above).

2.4 Practice: Familiar with 1st example project

For an example of adding the 1st function to the HFP scenario, please refer the DSP API reference > API reference > Middleware > Stream > Interface > Function under <sdk_root>/dsp/doc folder.

Search for "AB155x_Customized_Audio_Processing_Porting_Guide.rar" on Mediatek-on-line (MOL)

- <https://online.mediatek.com/>



3 BT Audio DSP SDK

3.1 Audio Chain introduction

3.1.1 Music playback scenario

The feature list of A2DP is “**stream_feature_list_a2dp**”. Developers can add customized algorithms to it and there are some default algorithms in the default feature list. The explanations for them is as follows.

1. **Codec**: It is the decoder for A2DP and there are two codes (SBC, AAC) in default SDK.
2. **CH SEL**: It is used to classify channels. In earbuds project, it helps to put the streaming data corresponding to that ear into 1st buffer.
3. **PEQ**: Post equalizer is used to provide customers to adjust the frequency response of the output sound. In default SDK, we provide two equalizers, one pre-equalizer and another post-equalizer.
4. **DRC**: The feature dynamic range compression is used to reduce the volume of loud sounds to reduce or compress the dynamic range of audio signal.
5. **CLK SKEW**: This feature is used to compensate the CLK drift between the source and sink device.

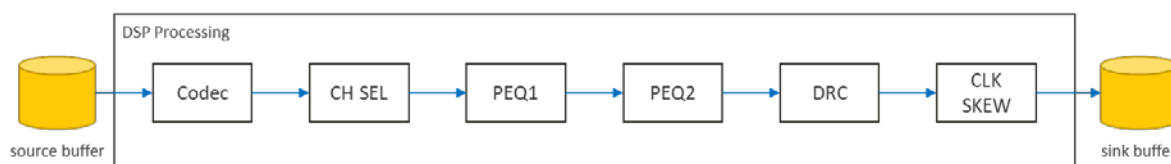


Figure 11. Audio chain for music playback scenario

3.1.2 Call scenario

There are two DSP data streams in HFP which called Downlink (Rx) and Uplink (Tx). The data format of Downlink/Uplink streams is 16k/16bit/mono/240 samples (15ms period). And Downlink has two decode types: mSBC/CVSD, Uplink has mSBC encode only. Therefore, there are two feature lists for Downlink:

stream_feature_list_hfp_downlink_msbc and **stream_feature_list_hfp_downlink_cvdsd**. The feature list of Uplink is **stream_feature_list_hfp_uplink**.

Figure 12 shows all features in the three feature lists above. The reason that the CVSD feature list has no CVSD decode is because HW decoded the CVSD data before DSP processing.

Developers can add customized feature functions into them and there are some default feature functions in the default feature lists.

The explanation for them is shown as below:

1. **mSBC Dec**: It is used to decode mSBC data received from BT to PCM data.
2. **PLC**: Packet Loss Concealment is a function to overcome packet loss problems.
3. **EC/NR**: This algorithm function Echo Canceling (EC) and Noise Reduction (NR) is used to improve the quality of voice.
4. **DRC**: The feature dynamic range compression is used to reduce the volume of loud sounds to reduce or compress the dynamic range of audio signal.
5. **CLK SKEW**: This feature is used to compensate the CLK drift between the source and sink device.

6. **Fixed SRC:** The data sample rate is 8k in CVSD codec flow, but HFP DSP stream sample rate is 16k. So this feature is used to convert 8k to 16k.
7. **CH SEL:** It is used to classify channels. In earbuds project, it would help to put the streaming data corresponding to that ear into 1st buffer.
8. **SW gain:** Adjusting the voice volume before EC/NR processing by Software.
9. **mSBC Enc:** It is used to encode PCM data to mSBC data for BT.

➤ Downlink: MSBC



➤ Downlink: CVSD



➤ Uplink



Figure 12. Audio chain for call scenario

3.1.3 Microphone sensing scenario

Here is the scenario that we call multi-mic streaming which allows users to get the input data from multiple microphones and do further processing on the DSP side. For example, it allows you to control the volume of the call due to louder environment noise. You can use this scenario to customize the MIC setting and the process arithmetic.

3.1.3.1 Feature option

Users must add multi-MIC relative option in MCU/DSP project's feature.mk.

- For MCU side project
 - AIR_AUDIO_TRANSMITTER_ENABLE = y
 - AIR_MULTI_MIC_STREAM_ENABLE = y
- For DSP side project
 - AIR_AUDIO_TRANSMITTER_ENABLE = y
 - AIR_MULTI_MIC_STREAM_ENABLE = y

3.1.3.2 The method to trigger this feature

The AT command to trigger the multi-mic streaming.

Airoha IoT SDK DSP Audio Algorithm Developers Guide

- Step 1: Input the following AT command to configure the MIC combination. This command will select the specific MICs for the multi-mic streaming scenario based on the actual HW circuit.
 - AT+EAUDIO=MULTIMIC_SEL
 - Identify the device/interface for each MIC channel.
- Step 2: Input the following AT command to initialize the multi-MIC streaming. This command will do the initialization for the audio stream and audio AFE hardware.
 - AT+EAUDIO=AUDIO_TRANSMITTER,init,2,<scenario_id>,<sample rate>,<resolution>,<sample counter per IRQ>
 - Ex: AT+EAUDIO=AUDIO_TRANSMITTER,init,2,0,16000,16,160\0d\0a
 - 16KHz sample rate, 16 bit, fetch 160 samples each time.
- Step 3: Input the following AT command to start the multi-MIC streaming. This command will trigger the audio AFE HW and active the audio stream to process regularly.
 - AT+EAUDIO=AUDIO_TRANSMITTER,start,2,<scenario_id>
- Step 4: Input the following AT command to stop multi-MIC streaming. This command will stop the audio AFE HW and audio stream.
 - AT+EAUDIO=AUDIO_TRANSMITTER,stop,2,<scenario_id>
- Step 5: Input below AT command to de-initialize multi-MIC streaming. This command will do de-initialization of audio stream and audio AFE HW.
 - AT+EAUDIO=AUDIO_TRANSMITTER,deinit,2,<scenario_id>

Note: refer to the following enum for the value of <scenario_id >

- File: mcu\middleware\MTK\audio\audio_transmitter\inc\audio_transmitter_control_port.h
- Enum: audio_transmitter_scenario_sub_id_multimic_t

3.1.3.3 The API interface

The API interface to use the multi-mic streaming.

- Step 1: The MCU side call below the API does the initialization of multi-MIC streaming scenario. This API will do the initialization for the audio stream and audio AFE hardware.
 - audio_transmitter_id_t audio_transmitter_init(audio_transmitter_config_t *config)
 - When calling audio_transmitter_init() to do initialization of multi-MIC streaming scenario, the user can also configure the MIC parameter by one of those method.
 - By user's setting
 - ◆ Configure hal_audio_mic_config_t and attach it to mic_para
 - By the setting from NVDM
 - ◆ if set the mic_para to NULL
 - The user also needs to register the callback, so that the DSP side can notify the user process result (Only 1 word) by event of AUDIO_TRANSMITTER_EVENT_DATA_DIRECT.
- Step 2: The MCU side call below API to start multi-MIC streaming scenario. This API will trigger the audio AFE HW and active the audio stream to process regularly.
 - audio_transmitter_status_t audio_transmitter_start(audio_transmitter_id_t id)
- Step 3: The DSP side can call below API to send the message to MCU with single word. The parameter of "message" is the word for user to transmit. This API can be used to share the status of audio process from DSP side to MCU side.

- void audio_transmitter_send_message
(audio_transmitter_scenario_t scenario_type, audio_transmitter_scenario_sub_id_t scenario_id, uint32_t message)
 - Step 4: The MCU side should receive the word from the second parameter in the callback function. The prototype of the user's callback defined as below in MCU side. This API can be used to share the status of audio process from DSP side to MCU side.
 - typedef void (*audio_transmitter_msg_handler_t)(audio_transmitter_event_t event, void *data, void *user_data)
 - Step 5: The MCU side call below API to stop multi-MIC streaming scenario. This API will stop the audio AFE HW and audio stream.
 - audio_transmitter_status_t audio_transmitter_stop(audio_transmitter_id_t id)
 - Step 6: The MCU side call below API to do de-initialization of multi-MIC streaming scenario. This API will do de-initialization of audio stream and audio AFE HW.
 - audio_transmitter_status_t audio_transmitter_deinit(audio_transmitter_id_t id)
- Note that the Start and stop operation can be called alternately without call initial API again.

3.1.3.4 How about the customization

Customized feature list in the DSP side.

The user can add customized initialization and process feature functions in stream_feature_table[DSP_FEATURE_MAX_NUM]. The feature type is referred as below.

- dsp/middleware/airoha/dspfw/port/chip/{platform_name}/src/dsp_sdk.c
 - stream_featuremulti_mic_function_a[]
 - FUNC_FUNCTION_A
 - stream_featuremulti_mic_function_b[]
 - FUNC_FUNCTION_B
 - stream_featuremulti_mic_function_c[]
 - FUNC_FUNCTION_C
 - stream_featuremulti_mic_function_f[]
 - FUNC_FUNCTION_F

3.2 Audio format in BT audio scenarios

The audio format in BT audio scenario is listed as below.

Table 10. Audio format in BT audio scenario

BT Audio scenario	Codec	Frame size	Resolution of processing
A2DP	SBC	128 / 64 / 32 @ 44.1kHz / 48kHz	32bits
	AAC-LC	1024 @ 44.1kHz / 48kHz	
LE Audio music	LC3	360 @ 48kHz 480 @ 48kHz	16bits
LE Audio call	LC3(Uplink/downlink)	240 @ 32kHz	16bits

Airoha IoT SDK DSP Audio Algorithm Developers Guide

BT Audio scenario	Codec	Frame size	Resolution of processing
		320 @ 32kHz	
HFP	CVSD(Uplink/downlink)	240 @ 16kHz	16bits
	mSBC(Uplink/downlink)	240 @ 16kHz	16bits

For both HFP and LE audio call, the SDK's default acoustic algorithms process 15 ms voice data with accumulate input data to 480@32kHz or 240@16kHz.

If the customized algorithm from users could not support all the frame size in Table 10, it is required to handle frame size transformation inside the algorithm.

3.3 Practice to add custom function

3.3.1 A2DP

User can add a new function in the feature list of A2DP “**stream_feature_list_a2dp**” in “**dsp_sdk.h**”. It contains many defaults feature in the list. The following are the precautions when porting function.

1. In the earbuds project, if you want to get stereo data (like spatial audio algorithms), add the feature before **FUNC_CH_SEL**.

If you want the frame size to be fixed in the feature, do not port the feature after “**FUNC_CLK_SKEW_A2DP_DL**”. Because “**FUNC_CLK_SKEW_A2DP_DL**” would increase or decrease the amount of data to achieve balance in order to compensate for the difference between CLK.

3.3.2 HFP

If you want to add new feature to HFP, the following rules must be followed and you can refer to the default existing to extend:

1. Add new feature type to **stream_feature_type_t** enum in **dsp_sdk.h** file.
2. Register feature (defined step1) into feature list: **stream_feature_list_hfp_downlink_msbc**, **stream_feature_list_hfp_downlink_cvdsd** or **stream_feature_list_hfp_uplink** according to requirements in **dsp_sdk.c** file.
3. To run any new feature in the SDK, it is necessary to register two APIs (feature Init API and feature Process API) in advance to SDK. So you need to define new feature Init API and Process API first.
4. Then, it is needed to register the Init API and Process API into **stream_feature_table** in **dsp_sdk.c** file.

3.4 Available algorithms in DSP SDK

In DSP SDK, the available algorithms could be founded under the following path.

- dsp/prebuilt/middleware/airoha/dspalg
- dsp/prebuilt/middleware/third_party/dspalg
-

Here is the description of the available algorithms in the DSP SDK package.

Table 11. The description of available dsp algorithms in SDK

Airoha IoT SDK DSP Audio Algorithm Developers Guide

Module name	Description	Used scenario	Note
AAC decoder	AAC-LC decoder for A2DP	A2DP	NA
clk_skew	Clock drift compensation for BT audio	A2DP/HFP	NA
compander	DRC (Dynamic Range Control)	A2DP/HFP	NA
ec_nr	Echo cancellation, Noise reduction, and EQ for voice process	HFP	<ul style="list-style-type: none"> Noise reduction and EQ is for both uplink and downlink. Echo cancellation is only for uplink.
env_detect	Detect environment noise level and adjust the ANC strength	ANC	NA
ins	Suppress the idle noise	Wired audio – input via 3.5mm audio jack	NA
leakage_compensation	ear tip's fit detection algorithm (one time and trigger by user)	Ear tip's fit detection	NA
msbc_decoder	mSBC codec for HFP (decoder and encoder)	HFP	NA
peq	parametric EQ for music	A2DP	NA
sbc_decoder	sbc decoder for A2DP	A2DP	NA
src_fixed_ratio	fixed ratio sampling rate conversion algorithm (up or down conversion in 2x/3x)	Not been used in default SDK.	NA
sw_gain	SW gain: to apply sw gain	HFP	NA
sw_src	SW SRC	Not been used in default SDK.	(Support 16K/32K/44.1K/48K

Airoha IoT SDK DSP Audio Algorithm Developers Guide

Module name	Description	Used scenario	Note
			in 16bits per sample)
voice_plc	Voice's packet loss concealment	HFP	(Support 8K/16K Hz in 16bits per sample)
volume_estimator	Volume estimator which used for silence detection feature	Not been used in default SDK.	NA
wind_detection	wind noise detection and adjust the ANC strength	ANC	NA
wwe	The beamforming algorithm is designed to enhance the acoustic performance for wake-up word scenario.	Voice Assistant	(The wakeup word engine algorithm is Not included in default SDK.)
audio_plc	Audio's packet loss concealment. (handles for the 48kHz sampling rate)	Not been used in default SDK.	(Support 44.1K/48K/88.2K/96 KHz)

Note:

1. Do not include the available algorithms in add-on package.
2. Although the algorithms are initiated with independent scratch memory, the dsp algorithms are not designed for multi-instance in the SDK. If you have such a requirement, we suggest that you speak to your contact window for more information.

3.5 Available basic functions in DSP SDK

In DSP SDK, some basic functions are available in the following path:

- dsp/middleware/airoha/dspalg/src
- dsp/middleware/airoha/dspalg/inc

Here is the description of the basic dsp functions that are available in the DSP SDK package.

Table 12. Description of basic dsp functions available in SDK

Airoha IoT SDK DSP Audio Algorithm Developers Guide

Module name	Description	Used scenario	Note
Ch_select	Allow user to select certain channel in stereo data streaming or mix Stereo data stream into one Mono stream.	A2DP	Mode : L_ch, R_ch, Mono, Stereo
Framesize_adjustor	Allow users to adjust process frame size with given audio processing when the process size is limited, and default audio frame size cannot match the size.	Has not been used in the default SDK.	Additional DRAM required and caused additional latency
Mute_smooth	The feature can mute A2DP streaming when severe jumpiness happened and recover after jumpiness does not occur for a while.	Has not been used in the default SDK.	NA
Queue_buffer	Help to queue frames when audio processing handles multiple frames in once.	Has not been used in the default SDK.	Additional DRAM required.



Note: Refer to the readme.txt in the SDK folder.

4 FAQ

4.1 How to handle frame size mismatch?

In the default SDK, we provide some APIs for users to get the parameter of the stream. For example, users can get the frame size and sampling rate via following APIs in “`dsp_feature_interface.h`”.

- **frame size(unit : bytes):** `uint16_t stream_function_get_output_size(void *para)`
- **sampling rate:** `uint8_t stream_function_get_samplingrate(void *para)`

When doing post-processing, there are three cases of frame size that may occur. The following are the suggested methods for the three situations:

- Case 1: The frame size of the algorithm can be compatible with the frame size output by codec.
 - A. Stream data can be directly sent into the algorithm.
- Case 2: The frame size of the algorithm cannot be compatible with the frame size output by codec. (frame size of the algorithm < frame size of codec)
 - A. Before sending the stream data into the algorithm, the stream must be segmented.
- Case 3: The frame size of the algorithm cannot be compatible with the frame size output by codec. (frame size of the algorithm > frame size of codec)
 - A. Developers need to create an additional buffer for storing stream data
 - B. Before the amount of data is sufficient and the algorithm is executed, the size of the output must be changed to 0.
(`bool stream_function_modify_output_size(void *para, uint16_t size)`)
 - C. When the amount of data in the temporary buffer is sufficient, move the data back to the original buffer and send it to the algorithm
 - D. When the algorithm has an output, the size of the output must be changed to the amount of data output by the algorithm.
(`bool stream_function_modify_output_size(void *para, uint16_t size)`)

4.2 How to handle sampling rate mismatch?

When a you algorithm does not support all the listed sampling rates in Table 10, we suggest you consult your contact window for further support.

Suggested flow:

1. Provide the supported format in your algorithms.
2. Look into whether the SDK's solution for handling the sampling rate conversion satisfies your needs.



4.3 How to pass parameter or control msg from mcu to dsp side?

Normally, we will pass parameters from MCU to DSP via the share memory pre-defined in SDK.

As shown in the following figure, here are several steps.

1. Read the parameters from flash.
2. Put the parameters into share memory and get the first address.
3. Send share memory's address to DSP via ccni message.
4. DSP get the parameters from share memory.

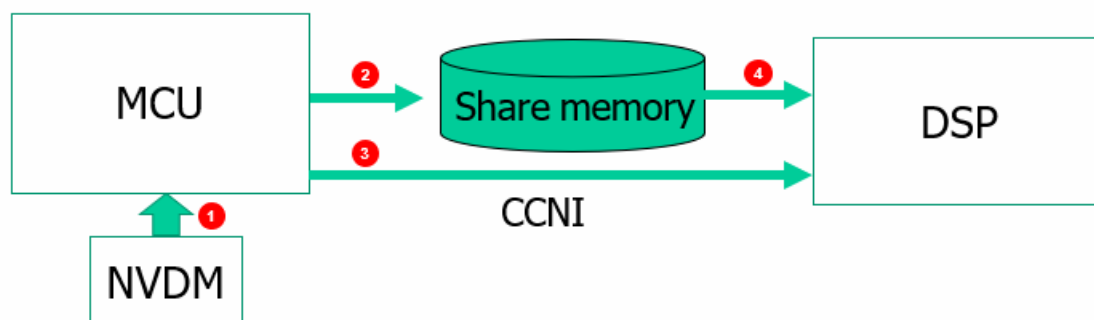


Figure 13. The flow of sending parameters from MCU to DSP

Here we use programmable equalizer (PEQ) in A2DP as an example.

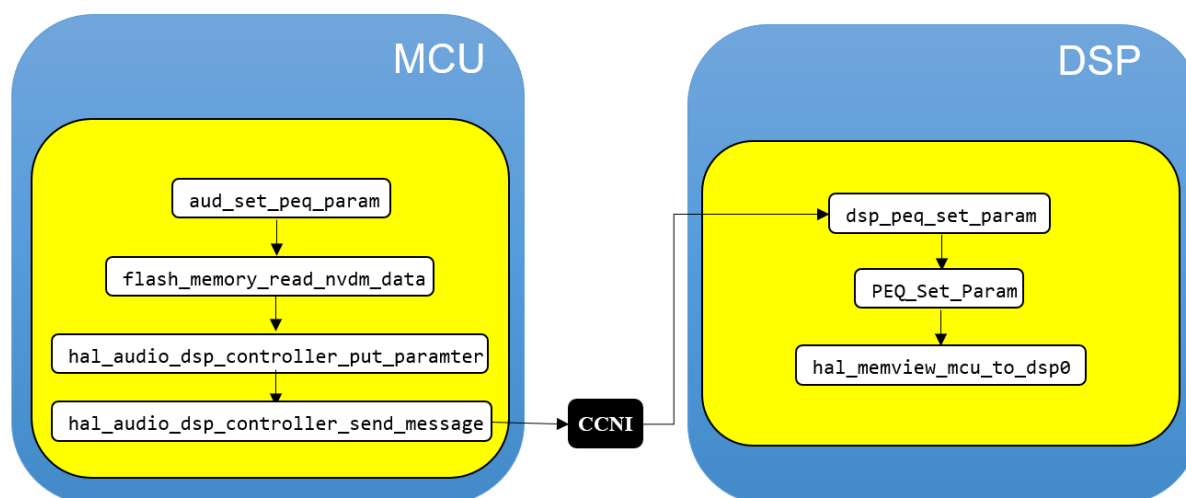


Figure 14. The example flow of passing parameters in PEQ module

Please refer the example code for more details.

Table 13. Control msg from mcu to dsp side example code in PEQ scenario

Steps of PEQ	File path	Key api list
1	mcu/middleware/airoha/audio_manager/src/peq_setting.c	<pre> int32_t aud_set_peq_param(peq_audio_path_id_t audio_path, bt_sink_srv_am_peq_param_t *ami_peq_param) </pre>

Airoha IoT SDK DSP Audio Algorithm Developers Guide

Steps of PEQ	File path	Key api list
		<pre>sysram_status_t flash_memory_read_nvdm_data(uint16_t id, uint8_t *data, uint32_t *length)</pre>
2	mcu/middleware/airoha/audio_manager/src/peq_setting.c	<pre>void *hal_audio_dsp_controller_put_paramter(const void *p_param_addr, uint32_t param_size, audio_message_type_t msg_type)</pre>
3	mcu/middleware/airoha/audio_manager/src/peq_setting.c	<pre>void hal_audio_dsp_controller_send_message(uint16_t message, uint16_t data16, uint32_t data32, bool wait)</pre>
4	dsp/middleware/airoha/dspfw/port/chip/ab158x/src/dsp_scenario.c	<p>DSP side will handle the PEQ's parameter setting in the following api.</p> <pre>void dsp_peq_set_param(hal_ccni_message_t msg, hal_ccni_message_t *ack)</pre>

Note:

1. The shared memory is allocated by MCU.

In SDK, audio_share_buffer is declared as share memory in the following path:

- mcu/driver/chip/\$chip_name/src/hal_audio_dsp_controller.c

Its data structure is defined as "audio_share_buffer_t" in the following path:

- mcu/driver/chip/\$chip_name/inc/hal_audio_message_struct.h

2. DSP side must do address mapping to dsp's view before doing the memory copy via the following api:

```
uint32_t hal_memview_mcu_to_dsp0(uint32_t mcu_address)
```

- The parameter 'mcu_address' is a non-cacheable address in the MCU view, and the return value is a non-cacheable address in the DSP view.



5 Terms and definitions

Table 14. Terms and definitions

Terms	Definitions
AAC	Advanced Audio Coding (AAC) is an audio coding standard for lossy digital audio compression. For more information, please refer to Wikipedia.
ANC	Active Noise Cancellation
A2DP	Advanced Audio Distribution Profile
BTA	Bluetooth Audio
CM4	Cortex-M4 is one of the cores in ARM Cortex-M series. For more information, please refer to Wikipedia.
CCNI	Cross Core Notification Interface, a HW interface for communication between cores. For more information, please refer the API document of this module.
CVSD	Continuously Variable Slope Delta modulation, a codec of audio. For more information, please refer to Continuously variable slope delta modulation in Wikipedia.
DHP	SW Task in DSP SDK
DAV	SW Task in DSP SDK
DTM	SW Task in DSP SDK
DPR	SW Task in DSP SDK
DRC	Dynamic Range Compression is adjust the loudness of sound. For more information, please refer to Wikipedia.
EC	Echo Cancellation is a technique to prevent echo during talks. For more information, please refer to Wikipedia.
eSCO	Enhanced synchronous connection oriented links. For more information, please refer to "List_of_Bluetooth_protocols" in Wikipedia.
EVK	Evaluation Kit
GPIO	General purpose inputs-outputs
HFP	Hands-Free Profile. For more information, please refer to Hands-free profile in Wikipedia.
HWSRC	The sampling rate conversion done by hardware.
I2S	I2S is a type of digital audio interface. For more information, please refer to Wikipedia.
IRQ	Interrupt Request
LE Audio	LE Audio operates on the Bluetooth Low Energy radio. For more information, please refer to Wikipedia.

Airoha IoT SDK DSP Audio Algorithm Developers Guide

Terms	Definitions
LC3	Low Complexity Communication Codec is an audio codec specified by the Bluetooth Special Interest Group (SIG) for the LE Audio audio protocol introduced in Bluetooth 5.2. For more information, please refer to Wikipedia.
MCPS	Million Cycle Per Second
MCU	Micro Controller Unit
MSBC	Modified Sub Band Codec, a codec of audio.
NVIC	Nested Vectored Interrupt Controller. NVIC is the interrupt controller of ARM Cortex-M4. For more details, please refer to NVIC introduction in ARM Cortex-M4 Processor Technical Reference Manual.
NVDM	Non-volatile Data Management
NR	Noise Reduction
PEQ	Programmable EQ refers to each filter boosting a range of frequencies in a parametric equalizer.
PIC	Position independent code. Please refer DSP's API document at the session – "preloader_pisplit".
PLC	Packet Loss Concealment
SBC	The Low Complexity Subband Coding (SBC) is an audio subband codec specified by the Bluetooth Special Interest Group (SIG) for the Advanced Audio Distribution Profile (A2DP). SBC is a digital audio encoder and decoder used to transfer data to Bluetooth audio output devices like headphones or loudspeakers. For more information, please refer to Wikipedia.
S/PDIF	Sony/Philips Digital Interface is a type of digital audio interface. For more information, please refer to Wikipedia.
VAD	Voice Activity Detection

6 Appendix

6.1 DSP algorithm's location in SDK

Here is SDK path of the mentioned algorithm modules.

Table 15. Mentioned algorithm modules in SDK path

Modules	SDK path
Codec - AAC	dsp/prebuilt/middleware/airoha/dspalg/aac_decoder
Codec – SBC	dsp/prebuilt/middleware/airoha/dspalg/sbc_decoder
Codec – mSBC	dsp/prebuilt/middleware/airoha/dspalg/msbc_decoder
CH SEL	dsp/middleware/airoha/dspalg/src/ch_select_interface.c
DRC	dsp/prebuilt/middleware/airoha/dspalg/compander
EC/NR	dsp/prebuilt/middleware/airoha/dspalg/ec_nr
Fixed SRC	dsp/prebuilt/middleware/airoha/dspalg/src_fixed_ratio
PEQ	dsp/prebuilt/middleware/airoha/dspalg/peq
SW gain	dsp/middleware/airoha/dspalg/sw_gain
CLK SKEW	dsp/middleware/airoha/dspalg/clk_skew

6.2 MCPS profiling method in DSP

You can get the DSP clock (dsp_clock), sampling rate (sample_rate), and process frame size from the DSP log. With the information. A simple calculation as shown below can get the MCPS for a specific audio process:

- Process MCPS = duration(us) / 1000000 * dsp_clock * sample_rate/sample_per_frame
 - The keyword to get dsp_clock from log is “DVFS [mcu_clk] \r\n” where dsp_clock equals to mcu_clk*2
 - User can get sampling rate and frame size during MCPS profiling for certain algorithm with SDK API `stream_function_get_samplingrate()` and `stream_function_get_output_size()` which are revealed in *section 2.3 Frequently used APIs*.

6.3 Algorithm's memory location in DSP

DSP supports 4 types of physical memory:

- Serial Flash
- Instruction RAM(IRAM)
- Data RAM(DRAM)
- system Random Access Memory(SYSRAM)

Airoha IoT SDK DSP Audio Algorithm Developers Guide

The IRAM and DRAM are only used by DSP. Serial Flash and SYSRAM are shared between MCU and DSP.

The algorithm's libraries will be downloaded to Serial Flash in download stage, and default dynamically loaded into IRAM and DRAM for execution after system power on, similarly SYSRAM can also be configured to execute algorithm's libraries. For algorithm's libraries dynamically loaded, you can refer to the section API

Reference->Kernel_service->preloader_pisplit in document

Airoha_IoT_SDK_for_XXXX_DSP_API_Reference_Manual.html that under <SDK_root>/dsp/doc.

Airoha IoT SDK DSP Audio Algorithm Developers Guide

Exhibit 1 Terms and Conditions

Your access to and use of this document and the information contained herein (collectively this “Document”) is subject to your (including the corporation or other legal entity you represent, collectively “You”) acceptance of the terms and conditions set forth below (“T&C”). By using, accessing or downloading this Document, You are accepting the T&C and agree to be bound by the T&C. If You don’t agree to the T&C, You may not use this Document and shall immediately destroy any copy thereof.

This Document contains information that is confidential and proprietary to Airoha Technology Corp. and/or its affiliates (collectively “Airoha”) or its licensors and is provided solely for Your internal use with Airoha’s chipset(s) described in this Document and shall not be used for any other purposes (including but not limited to identifying or providing evidence to support any potential patent infringement claim against Airoha or any of Airoha’s suppliers and/or direct or indirect customers). Unauthorized use or disclosure of the information contained herein is prohibited. You agree to indemnify Airoha for any loss or damages suffered by Airoha for Your unauthorized use or disclosure of this Document, in whole or in part.

Airoha and its licensors retain titles and all ownership rights in and to this Document and no license (express or implied, by estoppels or otherwise) to any intellectual propriety rights is granted hereunder. This Document is subject to change without further notification. Airoha does not assume any responsibility arising out of or in connection with any use of, or reliance on, this Document, and specifically disclaims any and all liability, including, without limitation, consequential or incidental damages.

THIS DOCUMENT AND ANY OTHER MATERIALS OR TECHNICAL SUPPORT PROVIDED BY AIROHA IN CONNECTION WITH THIS DOCUMENT, IF ANY, ARE PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE. AIROHA SPECIFICALLY DISCLAIMS ALL WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR A PARTICULAR PURPOSE, COMPLETENESS OR ACCURACY AND ALL WARRANTIES ARISING OUT OF TRADE USAGE OR OUT OF A COURSE OF DEALING OR COURSE OF PERFORMANCE. AIROHA SHALL NOT BE RESPONSIBLE FOR ANY AIROHA DELIVERABLES MADE TO MEET YOUR SPECIFICATIONS OR TO CONFORM TO A PARTICULAR STANDARD OR OPEN FORUM.

Without limiting the generality of the foregoing, Airoha makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Airoha assume any liability arising out of the application or use of any product, circuit or software. You agree that You are solely responsible for the designing, validating and testing Your product incorporating Airoha’s product and ensure such product meets applicable standards and any safety, security or other requirements.

The above T&C and all acts in connection with the T&C or this Document shall be governed, construed and interpreted in accordance with the laws of Taiwan, without giving effect to the principles of conflicts of law.