AB158x Series HA/PSAP/VividPT Developers Guide

Version: 1.0

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.

Version History

Version	Date	Description
0.1	3 June 2023	Initial draft
1.0	19 July 2023	Add Algorithm Customization

Table of Contents

List of Figures

List of Tables

1 Overview

This document describes the design and code structure of the Airoha Hearing aid (HA), Personal sound amplification product (PSAP) and Vivid passthrough (Vivid PT). It provides instructions and examples for customizing the application layer and the algorithm block in DSP.

Vivid PT is an Airoha proprietary technical name, meaning the most natural listening experience for the ambient sound. To achieve the goal of making the users feel like they are not wearing the earbuds, we shortened the latency of the audio path from the microphone to the speaker to 1.08ms to reduce the pipe sounds caused by the comb filtering effect that is a superposition of the ambient sound leaked through the earbud and the delayed speaker output. Compared to Vivid PT, the comb filtering effect is not that severe for HA and PSAP due to the output volume of the speaker is much higher than the leaked ambient sound.

The Airoha HA, PSAP and Vivid PT run on the DSP core (Cadence® HiFi5®) controlled by the MCU core (ARM® Cortex®-M33). We refer to HA, PSAP, and Vivid PT as Software Passthrough (SW PT) compared to our previous hardware passthrough solution.

1.1 Key feature

There are different feature sets and supported IC models for HA/PSAP/Vivid PT as shown in the table below:

Table 1-1. Key feature

	Feature	HA	PSAP	Vivid PT
HW	Supported IC models	AB1585H	AB1585 AB1588 AB1585H	AB1585 AB1588 AB1585H
	DAC Low power Class D	O	X (Not tested without HA firmware)	
	ADC LP_HA mode	O		
SW	Sampling rate (kHz)	25	25	50
	Full on gain (HFA-FOG) (dB)	> 20	≤ 20	0
	DRC for hearing compensation (channel)	50	12	X
	Adaptive Feedback Cancellation (AFC)	O	O	O
	Impulse Noise Reduction (INR)	O	X	X
	Noise Reduction (NR)	O	O	LDNR
	2-mic Wind Noise Reduction (WNR)	O	X	X
	Beamforming (Conversation boost)	O	O	X
	Latency (ms)	5.4	5.4	0.6
	Compensation EQ (band)	50	12	X
	Hearing Loss Compensation (HLC) (level)	16	9	X
	Hearing compensation for A2DP/LEA	DRC (25kHz)	DRC (50kHz)	-
	Hearing compensation for	DRC (25kHz)	DRC (50kHz)	-

The PSAP flavor is a scaled-down version of HA, with several features downgraded or removed. As the PSAP is facing the consumer market, the demand of the audio quality of multimedia (A2DP/LEA/SCO/VP) is higher than HA, so that the sampling rate with the multimedia compensation (DRC) enabled is also higher than that of HA.

Relatively, due to the more critical power consumption requirement, the sampling rate with the multimedia compensation (DRC) enabled for HA is lower than that of PSAP. For more information about the power consumption figures, refer to the test reports listed as below:

- AB1585H_TWS_with_Hearing_Aid_Power_Consumption_Test_Report
- AB1585_AB1585H_PSAP_Power_Consumption_Test_Report

1.2 Compile option & sla-addon

Table 1-2. Compile option & sla-addon

Compile option	Project	Addon package
AIR_PASSTHRU_ENABLE_TYPE = PASSTHRU_HEARING_AID	ab1585h_evk__earbuds_ref_design_hearing_aid	ha_addon_sla vivid_pt_addon_sla
AIR_PASSTHRU_ENABLE_TYPE = PASSTHRU_PSAP	ab1585_evk__earbuds_ref_design_psap	psap_addon_sla vivid_pt_addon_sla
AIR_PASSTHRU_ENABLE_TYPE = PASSTHRU_VIVID	ab1585_evk__earbuds_ref_design_vivid	vivid_pt_addon_sla

Note that the vivid_pt_addon_sla addon is also needed for the HA and PSAP projects.

1.3 Control flow

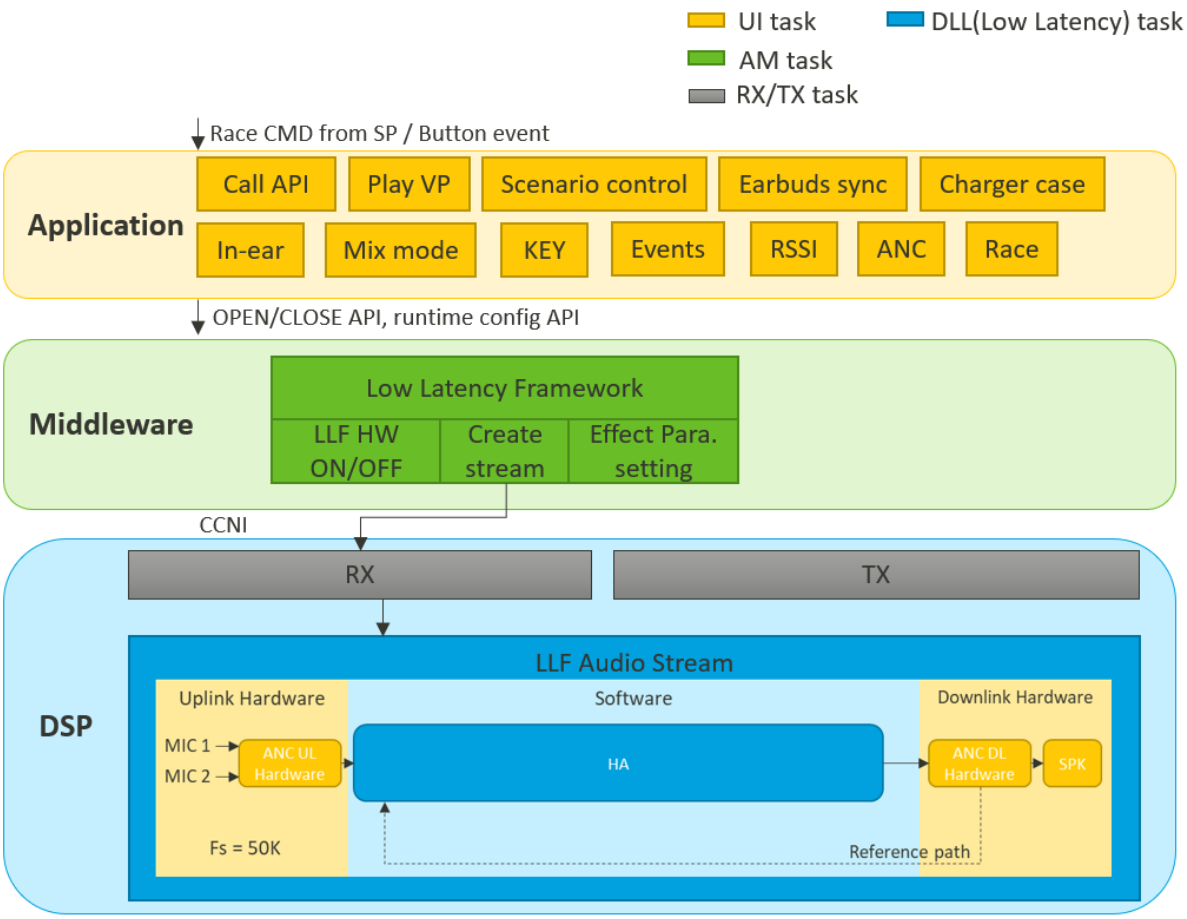


Figure 1-1. Control flow

2 Application

2.1 Application folder

There are 2 folders to process the Hearing Aid related features.

- mcu/project/ab158x/apps/earbuds_ref_design/src/apps/app_hear_through
  - Control Hearing Aid/PSAP/Vivid according to the configuration or events.
    - Out of case to enable hear through, in case to disable hear through.
  - Process key event to switch ambient control, anc -> off -> hear through -> anc.
  - Sync configuration to partner side, then partner should execute related operation according to the new configurations.
  - Control ANC according to the status.
  - Advertise the data to support connection from smartphone.
    - Includes generate CRC16 value from the smartphone's name.
  - Read/write hear through configuration.
  - Race command handler to support remote control from smartphone.
    - Support to enable/disable hear through.
    - Support to switch hear through mode.
    - Support to control Vivid PT AFC/LDNR configuration.
    - Support to execute the race command at the same time in the agent and partner side.
- mcu/project/ab158x/apps/earbuds_ref_design/src/apps/app_hearing_aid
  - Hearing Aid middleware API calling flow.
  - Control Hearing Aid according to the status.
    - Including A2DP/eSCO/VP/RSSI/In-ear.
  - Support mix table to enable/disable Hearing Aid.
  - Support enable/disable Hearing Aid according to the DRC configuration.
  - Read, write, and restore Hearing Aid configurations.
    - Restore the Hearing Aid configurations to be default (only factory reset Hearing Aid)
  - Process the Hearing Aid Race CMD from smartphone.
    - Support to execute the race command at the same time in the agent and partner side.
  - Key to switch Hearing Aid status

- Adjust the level, volume, Hearing Aid mode, AEA, tuning mode, master MIC channel…
- Support to execute the related operation at the same time in the agent and partner side.

2.2 Application feature

The application supports the following features.
The following is the feature list of the Hear Through:

Table 2-1. The feature list of the Hear Through

Feature	Comment
Control From smartphone	● Support to receive race command to change the parameters, enable/disable Hear Through, open/close Vivid PT. ● Notify configuration to smartphone.
Key processing	● Ambient control (ANC -> off -> Hear Through). ● Enable/disable Hear Through.
Vivid PT control	Open/close Vivid Passthrough.
ANC control	● Control ANC according to the Hear Through type ● When Hear Through enabled, enable ANC with the configured type in the nvkey. ● When Hear Through disabled, restore ANC type before Hear Through enabled.
Advertisement support	● When A2DP connected and RACE disconnected, start to advertise the data for smartphone connection. ● When RACE connected or ADV timeout (can be configured in the nvkey), stop advertise.
Auto enable/disable	● When out of charger case, enable Hear Through (according to the configuration), if Hear Through is configured to be enable. ● When in case, disable Hear Through.
Configuration Sync	● Sync agent configuration to partner side, also support partner request the configuration. o When agent init done and AWS connected. o When agent and partner connected o When key processing

	• Execute the command/key event at the future same time on the agent and partner side.

The following is the feature list of the Hearing Aid:

Table 2-2. The feature list of the Hearing Aid

Feature	Comment
Control from smartphone	• Support to receive race command to change the parameters runtime. • Notify configuration/parameters to smartphone.
Key processing	• Level up/down/circular • Volume up/down/circular • Mode up/down/circular • Beamforming switch • Master MIC channel switch • Tuning mode switch
Voice prompt	• Hearing Aid enable/disable VP • Maximum level • Maximum/minimum volume • Beamforming • Mode index • Maximum/minimum mode
RSSI related	• Enable/disable Hearing Aid according to the RSSI value and the RSSI configuration (through mix table) • Auto power off o If RSSI is bigger than configured RSSI, auto power off the device.
In-ear related	Enable/disable Hearing Aid according to the in-ear state (through mix table).
Restore	Restore the Hearing Aid configuration.
MIX table	Mix table is a matrix table to determine to enable/disable Hearing Aid, according to the A2DP/eSCO/RSSI/in-Ear switch status and the corresponding states.
DRC Control	Support to control Hearing Aid according to the DRC switch and the streaming state of A2DP/eSCO.

Configuration/parameter sync	● Support to sync agent configuration/parameter to partner side. o Application configuration/information o Middleware parameters ● When to sync o When AWS connected ● Support to execute the configuration/parameter at the future same time on the agent and partner side.

2.3 Race command

Support Race command to control device's Hearing Aid and Hear Through configuration. Support to notify device status to smartphone.

Table 2-3. Request race command format

Header (1 byte)	Length (2 bytes)	Type (2 bytes)	Data (n bytes for Hearing Aid)		
0x5A	Length of the following data	0x2C87	Operation (1 byte)	Configure type (2 bytes)	Configure parameter (n bytes)
			0x01: set 0x02: get	0x0001 ~ 0x001D	

Table 2-4. Response race command format

Header (1 byte)	Length (2 bytes)	Type (2 bytes)	Data (n bytes for Hearing Aid)			
0x5B	Length of the following data	0x2C87	Status (1 byte)	Operation (1 byte)	Configure type (2 bytes)	Configure parameter (n bytes)
			0x00: success Others: failed	0x01: set 0x02: get	0x0001 ~ 0x001D	Get response data.

Table 2-5. Notify race command format

Header (1 byte)	Length (2 bytes)	Type (2 bytes)	Data (n bytes for Hearing Aid)	
0x5D	Length of the following data	0x2C87	**Configure type (2 bytes)**	**Notify data (n bytes)**
			0x0001 ~ 0x001F	

The race command support to control different types of Hearing Aid or Hear Through also.

- For Hearing Aid: 0x0001 ~ 0x001F.
- For Hear Through:

- Hear through switch: 0x1001.
- Hear through mode: 0x1002.

- For Vivid PT:
  - AFC switch: 0x2001.
  - LDNR switch: 0x2002.
  - By pass switch : 0x2003

2.4 Application configuration

Config tool support to configure the application UI behavior of the Hearing Aid, including VP switch, ANC mode of each Hearing Aid mode, RSSI parameter…, refer to the following screenshot.

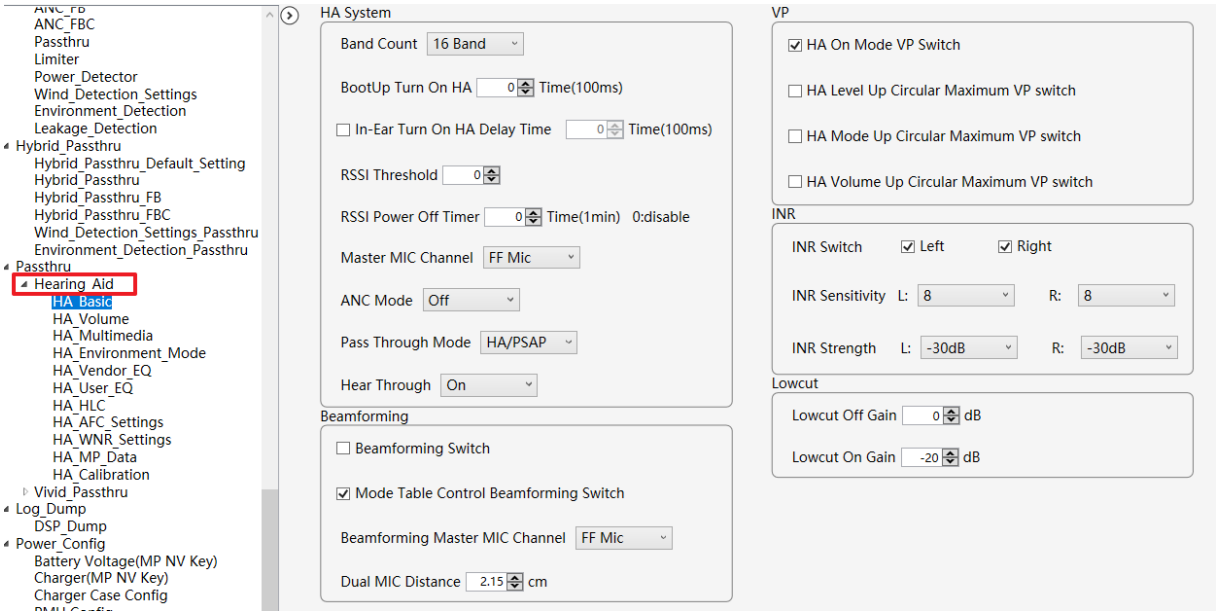


Figure 2-1. Config tool with Basic settings

Refer to the following screen shot for the mix table of Hearing Aid and DRC table.

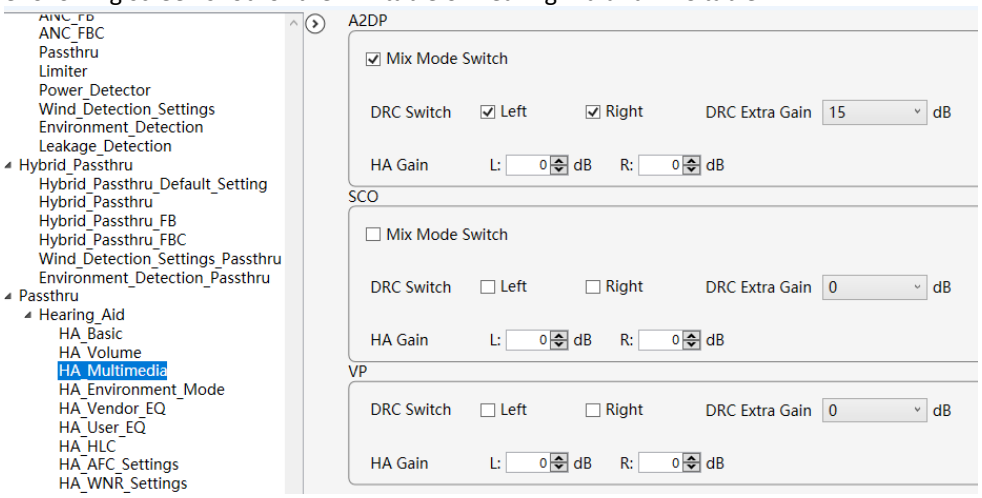


Figure 2-2. Config tool with Multimedia settings

2.5 Example for application

2.5.1 Environmental mode to adjust AFC & HLC level

When change the environmental mode index, if wish to change the AFC and HLC level index, you can add to the following function:

- File: app_hearing_aid_utils.c
- Function: app_hearing_aid_utils_adjust_mode

After called the audio_anc_psap_control_set_mode_index middleware API, call AFC and HLC level index API to configure the parameters.

```
bool app_hearing_aid_utils_adjust_mode(uint8_t target_mode_index)
{
    audio_psap_status_t mode_set_status =
audio_anc_psap_control_set_mode_index(target_mode_index);

    ha_afc_config_t config = {
        . afc_ctrl_switch_l = 1,
        . afc_ctrl_switch_r = 0,
        . reserved = 0,
    };
    audio_psap_device_role_t role = app_hearing_aid_utils_get_role();

    audio_psap_status_t afc_status = audio_anc_psap_control_set_afc_configuration(&config,
role);

    audio_psap_status_t set_level_status = audio_anc_psap_control_set_level_index (1, 2,
role);

    return true;
}
```

2.5.2 Mix table of Hearing Aids

The mix table is a matrix that is used to enable/disable hearing aids according to the A2D/SCO/in-ear/RSSI switch and state. For each switch/state that is changed, the state machine refers to the matrix to check whether it must enable or disable the hearing aid, and according to the output of the matrix, enable/disable hearing aid at a specific time.

The structure of the matrix is as follows:

```
typedef struct {
    bool a2dp_mix_switch;               // A2DP mix with HA or not
    bool sco_mix_switch;                // SCO mix with HA or not
    bool in_ear_switch;                 // In-ear to enable HA or not
    bool rssi_switch;                   // RSSI to enable HA or not
    ha_state a2dp_streaming;            // A2DP is streaming or not
    ha_state sco_streaming;             // SCO is streaming or not
    ha_state in_ear;                    // In-ear state
    ha_state less_than_threshold;       // RSSI over the threshold or not
} ha_mix_table;
```

You can configure the switch in the Config Tool or configure it dynamically via the smartphone. The state of each one is according to the status of the headphone. You can also modify the logic of the matrix in the ha_mix_table_list of the file app_hearing_aid_utils.c.

1) For A2DP case, to avoid impacting HA enable/disable in a short time if playing a ringtone on the smartphone side, add a delay to enable HA if the A2DP switch is on and in a2dp is streaming state. You can modify the timeout by modifying macro APP_HA_CONTROL_HEARING_AID_BY_A2DP_TIMEOUT (it is currently three (3) seconds) of the file app_hearing_aid_activity.c.
   You can check the logic in the function app_hearing_aid_activity_proc_bt_sink_event.
2) When the in-ear switch is enabled and not in-ear state, you must then suspend the ANC; When the user puts the earbud into the ear, it resumes ANC. When the in-ear switch changes to the disabled state, it must check whether ANC is suspended. If it is already suspended, it must first resume ANC, otherwise enabling HA/PSAP fails.
   You can check the function app_hearing_aid_activity_proc_app_interaction of the file app_hearing_aid_activity.c.
3) When AWS connected, agent side starts to read the RSSI of the partner if RSSI parameter is not configured to be 0x00.
   For reading event, you can check function app_hearing_aid_activity_handle_rssi_reading.
   For handling of the RSSI reading result, you can check the function app_hearing_aid_activity_handle_rssi_operation. It enters the matrix table to check whether it needs to enable or disable the hearing aid. This function is called on both agent and partner side to make sure the hearing aid behavior is the same.

2.5.3 How to execute operation on the agent and partner side at the same time

Support to send the command to another side and execute the command on both the agent and partner side at the same time in future. Refer to the function apps_aws_sync_send_future_sync_event of the file app_aws_sync_event.h.

2.5.3.1 Sync-execute operation of Hear Through

Send the operation by apps_aws_sync_send_future_sync_event API and add the corresponding handler in the handler list.

1) Add the hear through event handler to the hear_through_event_aws_event_handler_list.

2.5.3.2 Sync-execute operation of Hearing Aid

Call app_hearing_aid_aws_send_operate_command API to send the operation to the partner side, and the parameter need_execute_locally to determine whether need execute on the sender side or not. If wish to execute on both sides, make sure it is true. The following is the example code.

1) Send the operation at the position where wish to execute on another side.

```
uint8_t extra_data[2] = {0x01, 0x02};
app_hearing_aid_aws_send_operate_command(0x10,
                                         extra_data,
```

```
                                    2,
                                    true, /* Execute locally */
                                    200 /* Delay 200ms execute */);
```

2) Add the operation handler to the app_hearing_aid_sync_execute_event_handler of the file app_hearing_aid_aws.c.

```
• static ha_sync_execute_event_handler
  app_hearing_aid_sync_execute_event_handler[] = {
    NULL,                                                    // 0x00
    app_hearing_aid_aws_sync_handle_control_ha,              // 0x01
    app_hearing_aid_aws_sync_handle_bf_switch,               // 0x02
    app_hearing_aid_aws_sync_handle_aea_switch,              // 0x03
    app_hearing_aid_aws_sync_handle_master_channel_switch,   // 0x04
    app_hearing_aid_aws_sync_handle_tunning_mode_switch,     // 0x05
    app_hearing_aid_aws_sync_handle_change_level,            // 0x06
    app_hearing_aid_aws_sync_handle_change_volume,           // 0x07
    app_hearing_aid_aws_sync_handle_change_mode,             // 0x08
    app_hearing_aid_aws_sync_handle_change_mode_index,       // 0x09
    app_hearing_aid_aws_sync_handle_set_user_switch,         // 0x0A
    app_hearing_aid_aws_sync_handle_sync_vp_play,            // 0x0B
    app_hearing_aid_aws_sync_handle_power_off_request,       // 0x0C
    app_hearing_aid_aws_sync_handle_race_cmd_request,        // 0x0D
    NULL,                                                    // 0x0E
    app_hearing_aid_aws_sync_handle_rssi_operation,          // 0x0F
    app_hearing_aid_aws_sync_handle_0x10_operation,          // 0x10
};
```

•

3) Implement the operation

```
static void app_hearing_aid_aws_sync_handle_0x10_operation (uint8_t
from_which_role, uint8_t current_role, void *data, size_t data_len)
{
    // TODO implement the operation.
    // from_which_role means the AWS sync command from agent or partner
    // current_role means the current role of the AWS
    // data means the data of the command
    // data_len means the data length
}
```

2.5.3.3 Adding custom race command for Hear Through

If want to add custom race command for Hear Through, refer to the following flow.

1) Construct the race command parameter format.
2) Race command handler
    a. For get command
        i. The get command is only executed on the agent side, so you must implement the get command handler in the app_hear_through_activity_handle_get_cmd.
    b. Set command for non-AWS project or AWS not connected
        i. Set command handler: app_hear_through_activity_handle_set_cmd.

c. Set command for AWS project and AWS connected

i. Add the corresponding handler into the list.

ii. If wish to add the vivid race command, you can add to the hear_through_event_vivid_cmd_handler_list

```
static hear_through_event_handler hear_through_event_vivid_cmd_handler_list[] = {
    NULL,                                                          // 0x2000
    app_hear_through_activity_handle_vivid_afc_set_cmd,            // 0x2001
    app_hear_through_activity_handle_vivid_ldnr_set_cmd,           // 0x2002
    app_hear_through_activity_handle_vivid_by_pass_set_cmd,        // 0x2003
    app_hear_through_activity_handle_vivid_2004_set_cmd,           // 0x2004
};
```

3) Implement the race command

```
static bool app_hear_through_activity_handle_vivid_2004_set_cmd(void
*extra_data, uint32_t data_len)
{
    // TODO Implement the 0x2004 race command.
    // extra_data means the race command structure of the user data.
    // data_len means the length of the structure.
}
```

4) Use app_hear_through_race_cmd_send_notification function to send the status update.

```
uint8_t notify_data[2] = {0x01, 0x02};
app_hear_through_race_cmd_send_notification(0x5000, notify_data, 2);
```

- Note:
  - If it is set command, the command should be sent to partner side and execute on both sides at the same time.
  - If it is the get command, it should be handled on the agent side, the agent responds with data directly.

2.5.3.4 Adding a custom race command for a Hearing Aid

If wish to add custom race command for Hearing Aid, refer the following flow.

1) Construct the race command parameter format.
2) Add the set and get handler in the app_ha_exe_handler_list which located in the file app_hearing_aid_config.c.

```
typedef struct {
    uint8_t                execute_get_where; /* Execute on agent or partner or both */
    uint8_t                execute_set_where; /* Execute on agent or partner or both */
    bool                   notify_need_sync; /* Need wait partner notify result or not */
    bool                   need_execute_set_sync; /* Execute on both at the same time or not */
    bool (*ha_cmd_get_handler)(app_hear_through_request_t *request, uint8_t *response, uint16_t
*response_len); /* Get handler */
    bool (*ha_cmd_set_handler)(uint8_t *parameter); /* Set handler */
    bool (*ha_cmd_get_combine_handler)(uint8_t *local, uint16_t local_length, uint8_t *remote,
uint16_t remote_len, uint8_t *response, uint16_t *response_len); /* Notify data combine together
handler */
    bool (*ha_notify)(uint8_t role, uint8_t *data, uint16_t data_len, uint8_t *notify_data,
uint16_t *notify_data_len); /* Notify handler */
```

```
} app_hearing_aid_execute_handler_t;
```

Note:

- The return value of the handler means the process result, true as succeed, false as failed.
- The get response buffer length is 102 bytes, if the response is larger than 102, modify the parameter APP_HEARING_AID_RESPONSE_MAX_LEN.

3) Implement the set and get handler of the race command.
   a. If wish to process notify flow, you must also implement the ha_notify handler.

```
bool app_hearing_aid_utils_get_handler (app_hear_through_request_t *request, uint8_t *response,
uint16_t *response_len)
{

    // TODO implement the get handler.
}

bool app_hearing_aid_utils_set_handler (uint8_t *parameter)
{
    // TODO implement the set handler.
}
```

4) If wish to combine the agent and partner status in one notify data, you must also implement the handler as below:

```
bool app_hearing_aid_utils_get_level_index_combine_response(uint8_t *local, uint16_t local_length,
uint8_t *remote, uint16_t remote_len, uint8_t *response, uint16_t *response_len)
{
    // TODO implement the combine data notify.
}
```

2.6 Debug flow

For application debugging, you must filter “frame ~ “hearingaid”|| frame ~ “hearthrough”” in the logging tool.
After this filter, it lists all the processing flow of the application layer.
The log information should be “[**module**][**file**][**function**]” format.

```
]: [HearThrough][Activity][app_hear_through_proc_cm_event] AWS Connected, Role : 0x20
]: [HearingAid][ACTIVITY][app_hearing_aid_activity_proc_cm_event] AWS Connected, Role : 0x20
]: [HearThrough][Activity][app_hear_through_sync_event_handler_locally] target_gpt : 0, cur_gpt : 0
]: [HearingAid][ACTIVITY][app_hearing_aid_activity_set_user_switch] need_sync : 0, enable : 1
]: [HearingAid][AWS][app_hearing_aid_aws_handle_agent_user_configuration_sync] data : 0x422443c, data_len : 6
]: [HearingAid][Storage][app_hearing_aid_storage_sync_user_configuration][SYNC] level : 1, volume : 1, in_ear : 0, a2d
```

Figure 2-3. Debug log example

3 Algorithm block diagram

3.1 HA

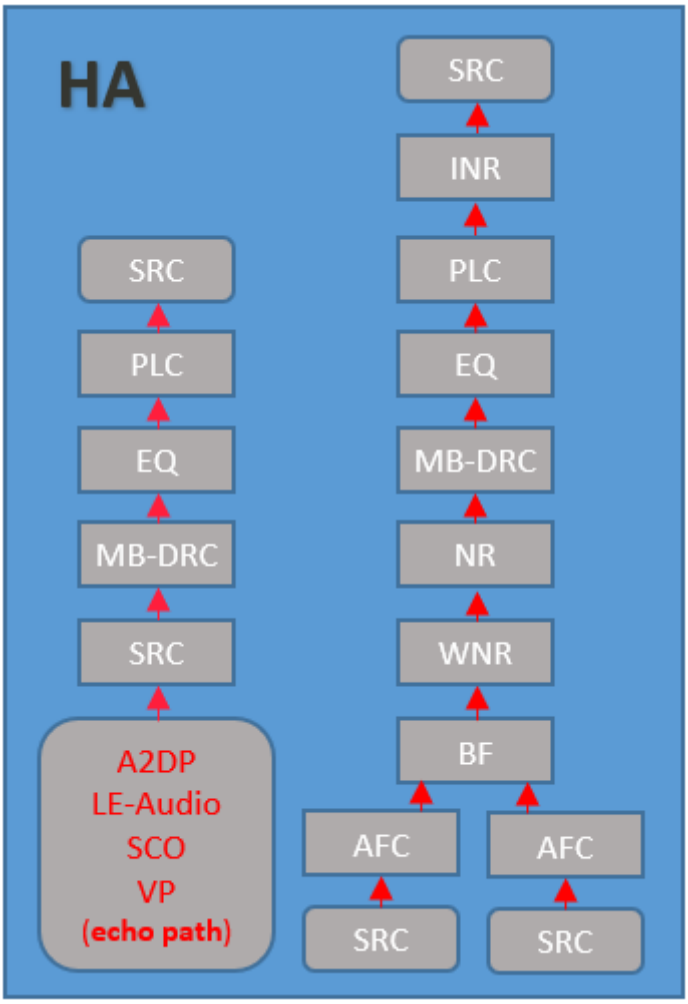
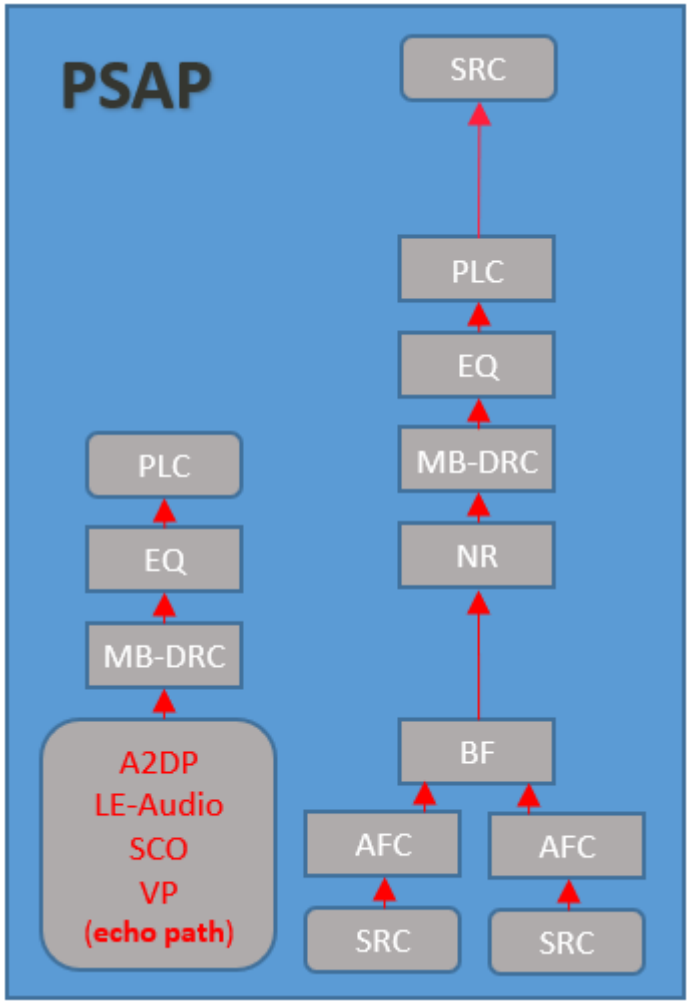


Figure 3-1. HA algorithm block diagram

3.2 PSAP



S

Figure 3-2. PSAP algorithm block diagram

3.3 Vivid PT

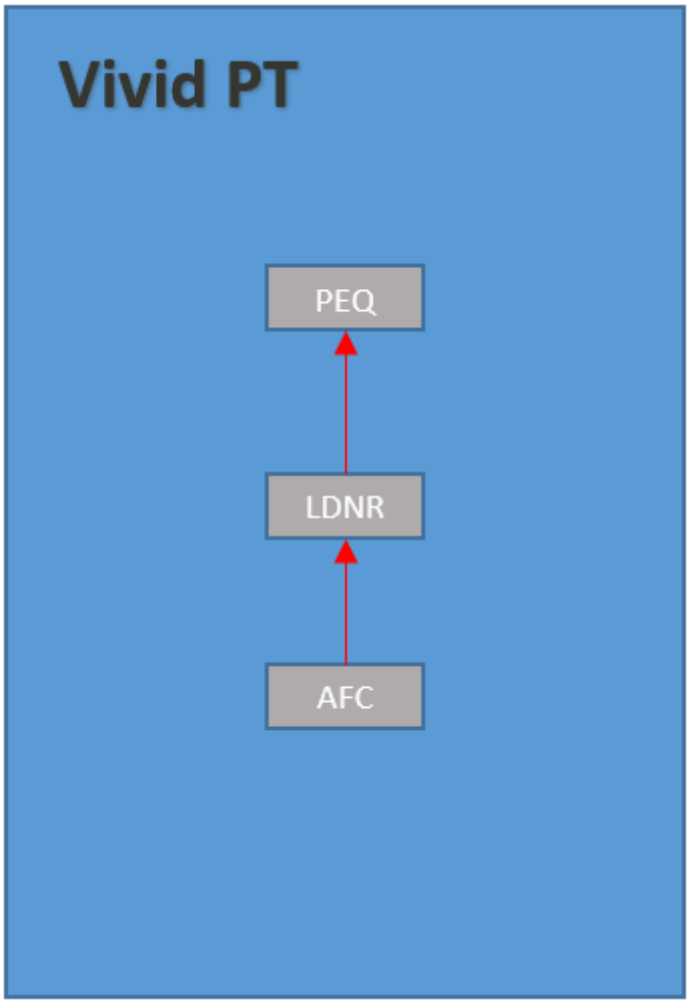


Figure 3-3. Vivid PT algorithm block diagram

4 Algorithm customization

We can provide you with information and examples to assist you with replacing the algorithm.

4.1 Customizable block diagram

The red boxes in the diagram as shown in Figure 4-1 represent the customizable blocks for customized algorithm.

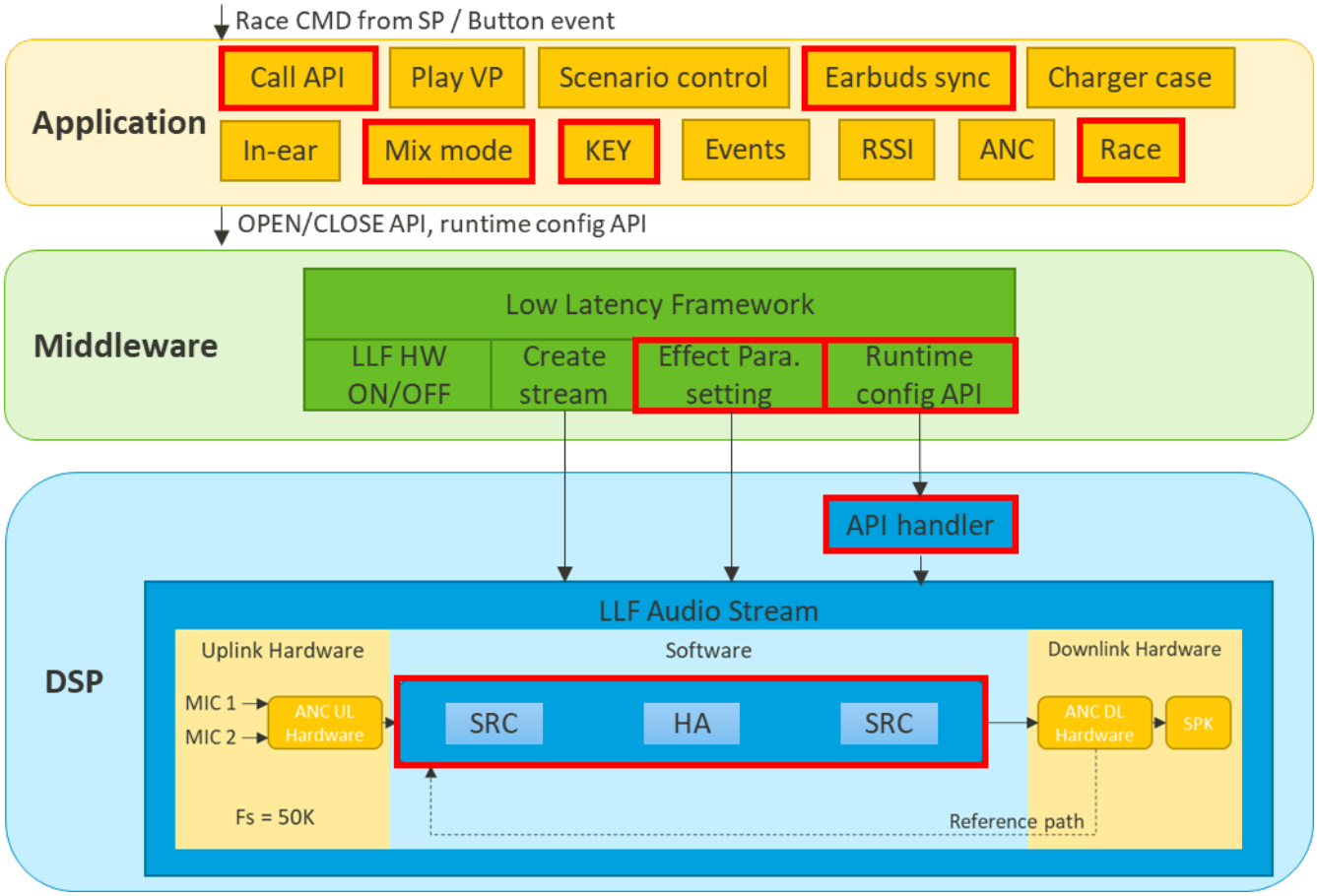


Figure 4-1. Customizable blocks.

4.2 Example for customization

Below are some customization examples. The files used in the examples can be referenced in Table 4-1. Reference files for algorithm customization.

Table 4-1. Reference files for algorithm customization.

mcu/middleware/airoha/audio/anc/psap/inc/audio_anc_psap_control.h
mcu/middleware/airoha/audio/anc/psap**/src/**audio_anc_psap_control**.c**
dsp/middleware/airoha/llf/psap/src/sub_scenario/llf_psap.c
dsp/middleware/airoha/llf/stream/inc/stream_llf.h
dsp/middleware/airoha/dspfw/port/chip/ab158x/src/dsp_sdk.c
dsp/middleware/airoha/dspfw/port/chip/ab158x/inc/dsp/dsp_sdk.h
dsp/prebuilt/ middleware/airoha/dspalg/hearing_aid/ab158x/inc/hearing_aid_interface.h

4.2.1 How to replace the HA algorithm

The control flow of opening LLF stream:

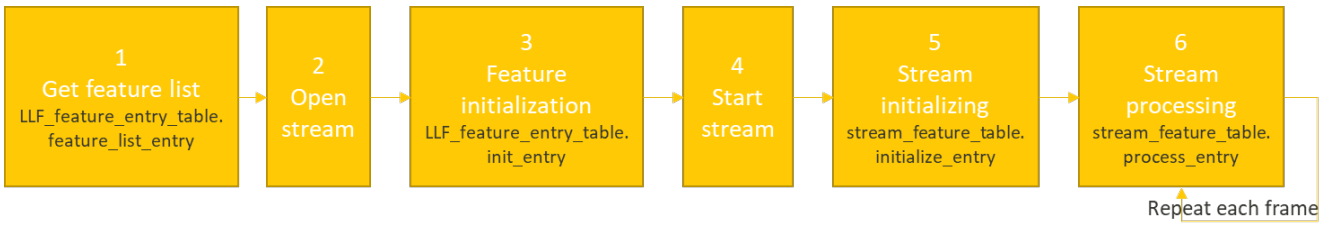


Figure 4-2. The control flow of opening LLF stream

We provide several feature entries for configuring the feature. These entries are defined in stream_llf.h. Replace the entries in *LLF_feature_entry_table* according to your requirement.

```
typedef struct {
    dsp_llf_feature_get_list_entry    feature_list_entry; //Declare feature table for the stream.
    dsp_llf_feature_ctrl_entry        init_entry; // The portable operation before starting the stream
    dsp_llf_feature_common_entry      deinit_entry; // The portable operation after stopping the stream
    dsp_llf_feature_ctrl_entry        suspend_entry; // The portable operation for re-loading the parameters
    dsp_llf_feature_ctrl_entry        resume_entry; // The portable operation for re-loading the parameters
    dsp_llf_runtime_config_entry      runtime_config_entry; // The portable operation for runtime configuration
    dsp_llf_set_dl_state              set_dl_state_entry; // The portable operation for music/call/VP
} dsp_llf_feature_entry_t;
```

- **The entry of getting feature list:**

The HA feature list is defined in scenario_hearing_aid.c.

```
Stream_feature_list_t stream_feature_list_HA[] = {
   CODEC_PCM_COPY,
   FUNC_SRC_FIXED_RATIO,
   FUNC_HA,
   FUNC_SRC_FIXED_RATIO,
   FUND_END,
}
```

Replace *FUNC_HA* with the newly created feature type in dsp_sdk.h. The example below is the newly created feature type(*FUNC_PSAP_SAMPLE*).

```
stream_feature_table_t stream_feature_table[DSP_FEATURE_MAX_NUM] = {
/*feature_type,                memory_size,      (UNUSED),   *initialize_entry,                    *process_entry*/
{FUNC_PSAP_SAMPLE,   PSAP_SAMPLE_MEM_SIZE,      0,      stream_function_psap_sample_initialize,   stream_function_psap_sample_process},}
```

- **The entry of feature initialization:**
  The portable operation before starting the stream. The HA feature initialization is in *HA_hearing_aid_feature_init()* of llf_psap.c.

- **The entry of Stream initializing and processing:**
  The customized algorithm should be in the initialize entry and the process entry. You can refer to *stream_function_sample_initialize()* and *stream_function_sample_process()*. The following image is an example code to get LLF data.

```
ATTR_TEXT_IN_IRAM bool stream_function_hearing_aid_process(void *para)
{
    uint32_t in_frame_size = stream_function_get_output_size(para);

    S32 *InBuf_rear       = (S32 *)stream_function_get_inout_buffer(para, hal_llf_get_data_buf_idx(LLF_DATA_TYPE_REAR_L));
    S32 *InBuf_in_ear     = (S32 *)stream_function_get_inout_buffer(para, hal_llf_get_data_buf_idx(LLF_DATA_TYPE_INEAR_L));
    S32 *InBuf_front      = (S32 *)stream_function_get_inout_buffer(para, hal_llf_get_data_buf_idx(LLF_DATA_TYPE_TALK));
    S32 *InBuf_music_voice = (S32 *)stream_function_get_inout_buffer(para, hal_llf_get_data_buf_idx(LLF_DATA_TYPE_MUSIC_VOICE));

    S32* OutBuf        = (S32 *)stream_function_get_inout_buffer(para, 1);
```

Figure 4-3. Example code for getting data buffer.

If you want more features in HA, add more features in the HA feature list. The data flow is shown as Figure 4-4. Data flow of customized function.Figure 4-4. Data flow of customized function.

```
stream_feature_list_t stream_feature_list_HA[] = {
   CODEC_PCM_COPY,
   FUNC_CUSTOM_A,
   FUNC_CUSTOM_B,
   FUND_END,
}
```

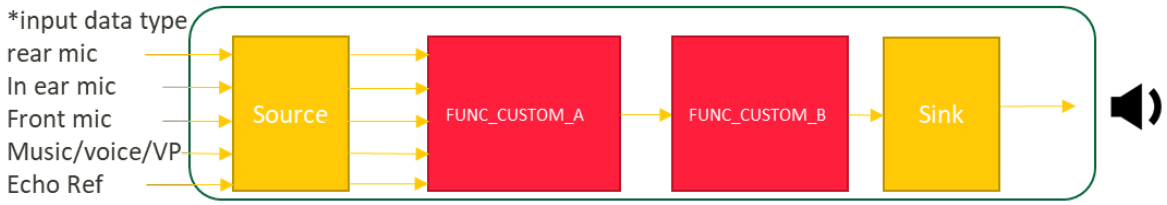


Figure 4-4. Data flow of customized function.

- The entry of suspend and resume:
  The portable operation for re-loading the algorithm parameters from flash when config tool update NVKey in runtime. The HA suspend/resume function is in *HA_hearing_aid_suspend()* and *HA_hearing_aid_resume()*. After the stream is suspended, the parameters are reloaded from flash. Refer to section 4.2.3 for instructions on reloading the parameter. Finally, resume HA. The stream is re-initialized. Remember to remove or replace the *stream_function_hearing_aid_deinitialized()* function with your stream function.

- The entry of runtime configuration:
  The API handler on DSP to receive the notification and set the algorithm parameters in runtime. Refer to section 4.2.2 for more details.

- The entry of setting DL state:
  This is a notification for when the music/call/VP starts or stops, and you can add the action here. Refer to *dsp_llf_set_audio_dl_status() in* stream_llf.c.

4.2.2 How to replace runtime setting API

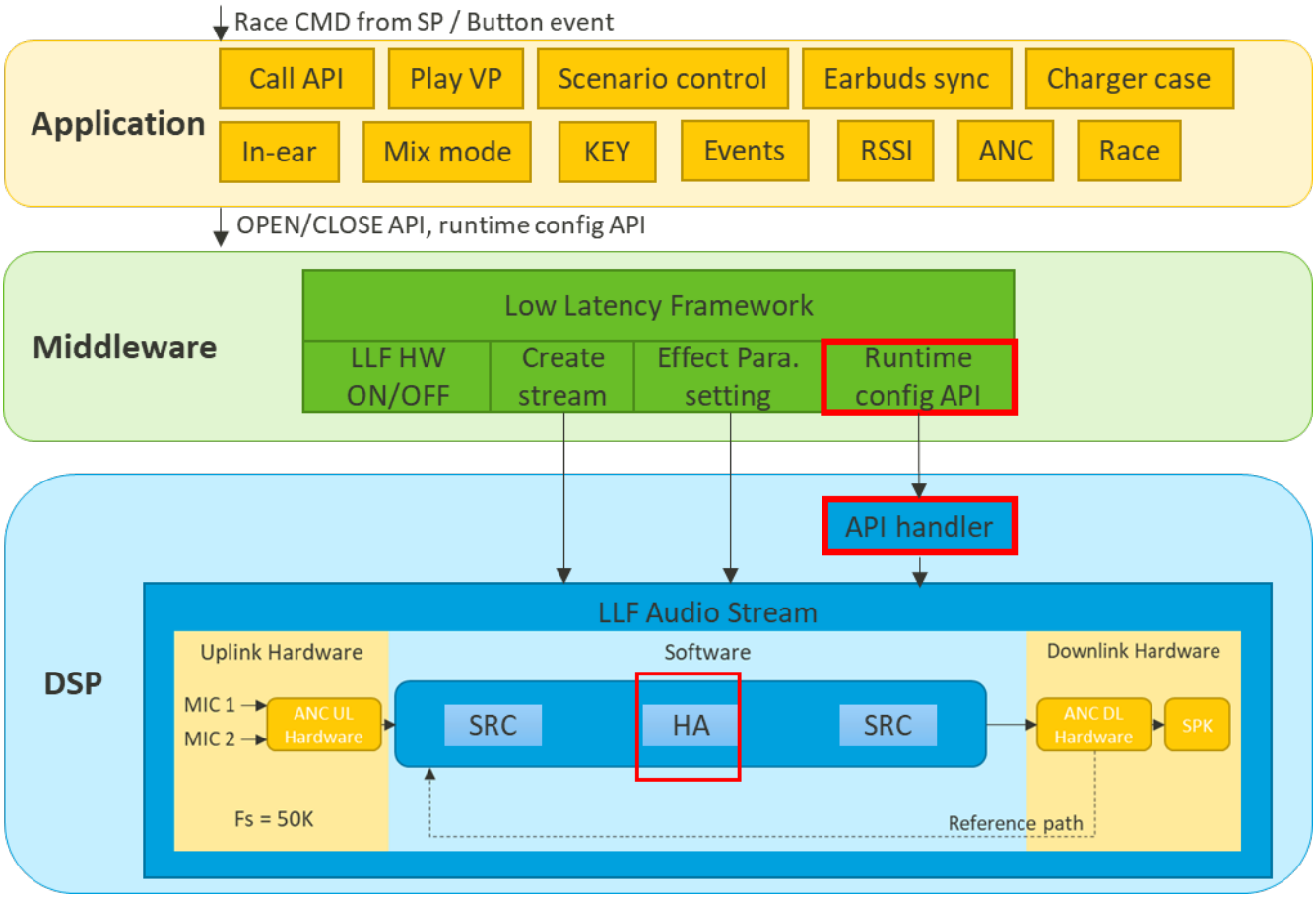


Figure 4-5. Customized block of runtime setting API

The algorithm runtime APIs are defined in hearing_aid_interface.h, these APIs need to be replaced after replacing the algorithm. The setup process for Runtime Configuration API is depicted in the above diagram. It starts by preparing parameters from middleware, then notifying DSP with the parameters. Finally, the algorithm parameters are set. Taking *audio_psap_control_set_eq_gain()* as an example:

- Starting by preparing parameters from middleware, then notifying DSP with the parameters. (Defined in audio_anc_psap_control.c)

```
audio_psap_status_t audio_psap_control_set_user_eq_gain(ha_usr_eq_para_t *usr_eq_gain,
audio_psap_device_role_t role)
{
   memcpy(&g_ha_ctrl.user_setting.ha_usr_eq_para_t, usr_eq_gain, sizeof(ha_usr_eq_para_t));

  U32 config_len = sizeof(llf_control_runtime_config_t);
  U32 usr_eq_len = sizeof(ha_usr_eq_para_t);
  U32 para_len = config_len + usr_eq_len;

  void *malloc_ptr = pvPortMalloc(para_len);
  llf_control_cap_t psap_cap = {LLF_TYPE_HEARING_AID,
                                HA_SUB_MODE,
                                0, 0, 0, 0,
                                (U32)malloc_ptr, para_len};
                               //fill the address of parameters and length.
  llf_control_runtime_config_t config = {LLF_TYPE_HEARING_AID,
                                         HA_SUB_MODE,
                                         HA_RUNTIME_CONFIG_EVENT_USR_EQ_GAIN,
```

```
                                        //fill the runtime config event ID
                                       role};

  memcpy(malloc_ptr, &config, config_len);
  malloc_ptr += config_len;
  memcpy(malloc_ptr, &usr_eq_gain, usr_eq_len);

  llf_control(LLF_CONTROL_EVENT_RUNTIME_CONFIG, &psap_cap);
   //send the runtime config request to LLF control.
}
```

- DSP API handler receives the notification and sets the algorithm parameters. (Defined in llf_psap.c)

```
void HA_hearing_aid_runtime_config(audio_llf_runtime_config_t *param)
{
  switch(param->config_event)
  {
    case HA_RUNTIME_CONFIG_EVENT_USR_EQ_GAIN:
      ha_usr_eq_para_t *usr_eq_gain = (ha_usr_eq_para_t*)((U8*)param
                                    + sizeof(llf_control_runtime_config_t));
      //Here to replace the Airoha algorithm API with customized algorithm API.
  }
}
```

You can refer to the Audio_PSAP_API_Reference_Manual to check if any modifications, additions, or removals of APIs are required.

- Modification of API：

  If the parameter structure you are using remains unchanged, you can directly set the algorithm parameters. Conversely, if you need to modify the parameter structure, the entire configuration process must be modified.

- Additions, or removals of API：

  Referring to the above process, you can add or remove an HA_RUNTIME _CONFIG_EVENT number along with its corresponding API function.

4.2.3 How to create a NVKey and pass the content to DSP

To read parameters from flash and send them to DSP during the activation process of HA, you can refer to *hearing_aid_set_parameter().*

- We provide three methods for sending parameters to DSP. You can choose any one of them according to your needs.
  1. *llf_control_set_parameter(U16 nvkey_id)*
     Read a NVKey from flash and send it to DSP.
  2. *llf_control_set_multi_parameters(U16 nvkey_id_list[], U8 key_num)*
     Read *key_num* specified NVKeys from flash and send all the content to DSP.
  3. *llf_control_realtime_set_parameter(U16 nvkey_id, U32 nvkey_length, U8* data)*
     Read a NVKEY from memory address and send it to DSP.
- How to create a NVKey:
  1. Create an NVKey ID in mcu/tools/nvkey/NVkey_ID_list.csv, then build the project, NVKey ID header is generated in mcu/prebuilt/middleware/airoha/nvdm.
  2. Add the new NVKey content in mcu/project/CHIP_NAME/apps/earbuds_ref_design/config_bin/PROJ_NAME/nvkey.xml.

3. (Optional) Read the NVKey and send it to DSP.
4. Clean and build the project.

- The entry of receiving parameters in DSP is *dsp_set_algorithm_param()*. You can keep these parameters and set them during initialization of the stream. If you use the same NVKey ID of HA, replace the function *stream_function_hearing_aid_load_nvkey()* with your stream function.

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.