Read the Docs Template Documentation

Read the Docs

2024 年 07 月 25 日

Contents

1 Get Started **3**

1.1 Development Board Overview . . . 3
1.2 About ESP-ADF . . . 4
1.3 Quick Start . . . 4
1.4 Installation Step by Step . . . 6
1.5 Step 1. Set up ESP-IDF . . . 6
1.6 Step 2. Get ESP-ADF . . . 7
1.7 Step 3. Set up the environment . . . 7
1.8 Step 4. Start a Project . . . 8
1.9 Step 5. Connect Your Device . . . 9
1.10 Step 6. Configure . . . 9
1.11 Step 7. Build the Project . . . 10
1.12 Step 8. Flash onto the Device . . . 12
1.13 Step 9. Monitor . . . 13
1.14 VS Code Extension . . . 13
1.15 IDF Eclipse Plugin and Espressif IDE . . . 14
1.16 Update ESP-ADF . . . 17

2 API Reference **19**

2.1 Audio Framework . . . 20
2.2 音频流 . . . 82
2.3 播放列表 . . . 112
2.4 Codecs . . . 136
2.5 Audio Processing . . . 152
2.6 服务 . . . 165
2.7 Speech Recognition . . . 219
2.8 Peripherals . . . 229

2.9 Abstraction Layer . . . 264
2.10 Configuration Options . . . 290

3 Design Guide **297**
3.1 项目设计 . . . 297
3.2 Design Considerations . . . 300
3.3 Software Design . . . 303
3.4 Development Boards . . . 305
3.5 Audio Samples . . . 403

4 Resources **407**

5 Copyrights and Licenses **409**
5.1 Software Copyrights . . . 409

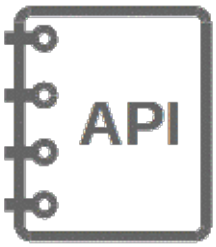
6 English-Chinese Glossary **411**

7 About **423**

索引 **425**

[English]

这里是乐鑫音频开发框架（ADF）的文档中心。

		
Get Started	API Reference	Design Guide

CHAPTER 1

Get Started

This document is intended to help users set up the software environment for the development of audio applications using hardware based on the ESP32 family of chips by Espressif. After that, a simple example will show you how to use ESP-ADF (Espressif Audio Development Framework).

1.1 Development Board Overview

For easier start with ESP-ADF, Espressif designed ESP32, ESP32-S2, and ESP32-S3 based development boards intended for audio applications. Click the links below to learn more about the available boards.

- *ESP32-LyraT*
- *ESP32-LyraT-Mini*
- *ESP32-LyraTD-MSC*
- ESP32-S2-Kaluga-1
- *ESP32-Korvo-DU1906*
- *ESP32-S3-Korvo-2*
- *ESP32-C3-Lyra*

If you do not have any of the above boards, you can still use ESP-ADF for the ESP32 and ESP32-S2 based audio applications. For this, your board needs to have a compatible audio codec or DSP chip; alternatively, you can develop a driver to support communication with your specific chip.

1.2 About ESP-ADF

The ESP-ADF is available as a set of components to extend the functionality already delivered by the ESP-IDF (Espressif IoT Development Framework).

To use ESP-ADF you need set up the ESP-IDF first, and this is described in the next section.

注解： ESP-ADF provides support for specific ESP-IDF versions. If your have already set up another version, please switch to a supported ESP-IDF version, or you may not be able to compile ESP-ADF applications. In addition, the python version needs to be between 3.7 and 3.11.

1.3 Quick Start

This section provides quick steps to run a simple ADF sample project on an ESP device for experienced users. For beginners, please go through the complete steps from *Step 1. Set up ESP-IDF* to *Step 9. Monitor* to build a project.

注解： If you encounter issues in the following steps, you could refer to the complete steps from *Step 1. Set up ESP-IDF* to *Step 9. Monitor* or describe them in GitHub Issues or ESP Forum.

1.3.1 Linux and macOS

The operating environment below is on Linux Ubuntu 18.04 and above.

1. Download the full ESP-ADF repository from GitHub by running:

   ```
   git clone --recursive https://github.com/espressif/esp-adf.git
   ```

   For users located in China, it is faster to download from Gitee:

   ```
   git clone --recursive https://gitee.com/EspressifSystems/esp-adf.git
   ```

2. Configure the `$ESP-IDF` and `$ESP-ADF` compilation environment by running:

   ```
   cd esp-adf
   ./install.sh
   . ./export.sh
   ```

3. After completing the above environment variable configuration, you can compile the ADF sample project `$ADF_PATH/examples/get-started/play_mp3_control`. Switch to the project's directory, com-

pile, and flash it onto your ESP device by running the following command. Then, you will see the serial port of the routine is printed.

```
cd $ADF_PATH/examples/get-started/play_mp3_control
idf.py build flash monitor
```

1.3.2 Windows

1. Download the full ESP-ADF repository from GitHub by running:

   ```
   git clone --recursive https://github.com/espressif/esp-adf.git
   ```

   For users located in China, it is faster to download from Gitee:

   ```
   git clone --recursive https://gitee.com/EspressifSystems/esp-adf.git
   ```

2. Install the `$ESP-IDF` compilation environment in the command prompt window:

   ```
   cd esp-adf
   .\install.bat
   ```

   Or first download the full ESP-IDF Windows Installer from ESP-IDF Windows Installer (Please download the ESP-IDF versions supported by ESP-ADF). And then turn off the antivirus software (Because it may prevent the installation as the software writes the Windows system regedit) and install the downloaded file. After the installation is complete, open the ESP-IDF CMD shortcut icon on the desktop, the script will automatically help you download submodules, and set environment variables such as `IDF_PATH`.

3. Set the `ADF_PATH` by running the following commands:

   ```
   .\export.bat
   echo %ADF_PATH%
   ```

4. If your `ADF_PATH` variable prints correctly, it's time to compile the ADF routines:

   ```
   cd %ADF_PATH%\examples\get-started\play_mp3_control
   idf.py build flash monitor
   ```

1.4 Installation Step by Step

This is a detailed roadmap to walk you through the installation process.

1.4.1 Setting up Development Environment

- *Step 1. Set up ESP-IDF* for Windows, Linux or Mac OS
- *Step 2. Get ESP-ADF*
- *Step 3. Set up the environment*

1.4.2 Creating Your First Project

- *Step 4. Start a Project*
- *Step 5. Connect Your Device*
- *Step 6. Configure*
- *Step 7. Build the Project*
- *Step 8. Flash onto the Device*
- *Step 9. Monitor*

1.5 Step 1. Set up ESP-IDF

Configure your PC according to **Getting Started** section of **ESP-IDF Programming Guide**. Windows, Linux and Mac OS operating systems are supported. Please select and follow the guide specific to ESP32 or ESP32-S2 chip. The chip name is provided in the board name.

注解： This guide uses the directory `~/esp` on Linux and macOS or `%userprofile%\esp` on Windows as an installation folder for ESP-ADF. You can use any directory, but you will need to adjust paths for the commands accordingly. Keep in mind that ESP-ADF does not support spaces in paths.

To make the installation easier and less prone to errors, use the `~/esp` default directory for the installation.

If this is your first exposure to the ESP-IDF, then it is recommended to get familiar with **hello_world** or **blink** example first.

After getting familiar with ESP-IDF, decide on which ESP-IDF version to use for your application depending on the Espressif chip that you have and your project type. For this, consult Versions section of ESP-IDF Programming Guide.

Once you successfully build, upload, and run examples for your version of ESP-IDF, you can proceed to the next step.

1.6 Step 2. Get ESP-ADF

Now you can start installing audio-specific API / libraries provided in ESP-ADF repository.

1.6.1 Windows

Open Command Prompt and run the following commands:

```
cd %userprofile%\esp
git clone --recursive https://github.com/espressif/esp-adf.git
```

1.6.2 Linux and macOS

Open Terminal, and run the following commands:

```
cd ~/esp
git clone --recursive https://github.com/espressif/esp-adf.git
```

1.7 Step 3. Set up the environment

Before being able to compile ESP-ADF projects, on each new session, ESP-IDF tools should be added to the PATH environment variable. To make the tools usable from the command line, some environment variables must be set. ESP-ADF provides a script which does that.

1.7.1 Windows

ESP-IDF Tools Installer for Windows creates an "ESP-IDF Command Prompt" shortcut in the Start Menu. This shortcut opens the Command Prompt and sets up all the required environment variables. You can open this shortcut and proceed to the next step.

Alternatively, if you want to use ESP-IDF in an existing Command Prompt window, you can run:

```
%userprofile%\esp\esp-adf\export.bat
```

1.7.2 Linux and macOS

In the terminal where you have installed ESP-IDF, run:

```
. $HOME/esp/esp-adf/export.sh
```

Note the space between the leading dot and the path!

You can also create an alias for the export script to your `.profile` or `.bash_profile` script. This way you can set up the environment in a new terminal window by typing `get_idf`:

```
alias get_idf='. $HOME/esp/esp-adf/export.sh'
```

Note that it is not recommended to source `export.sh` from the profile script directly. Doing so activates IDF virtual environment in every terminal session (even in those where IDF is not needed), defeating the purpose of the virtual environment and likely affecting other software.

1.8 Step 4. Start a Project

After initial preparation you are ready to build the first audio application. The process has already been described in ESP-IDF documentation. Now we would like to discuss remaining key steps and show how the toolchain is able to access the ESP-ADF components by using the `ADF_PATH` variable.

To demonstrate how to build an application, we will use get-started/play_mp3_control project from examples directory in the ADF.

1.8.1 Windows

```
cd %userprofile%\esp
xcopy /e /i %ADF_PATH%\examples\get-started\play_mp3_control play_mp3_control
```

1.8.2 Linux and macOS

```
cd ~/esp
cp -r $ADF_PATH/examples/get-started/play_mp3_control .
```

There is a range of example projects in the examples directory in ESP-ADF. You can copy any project in the same way as presented above and run it.

It is also possible to build examples in-place, without copying them first.

重要： The ESP-IDF build system does not support spaces in the paths to either ESP-IDF or to projects.

1.9 Step 5. Connect Your Device

Connect the audio board to the PC, check under what serial port the board is visible and verify, if serial communication works as described in ESP-IDF documentation.

注解： Keep the port name handy as you will need it in the next steps.

1.10 Step 6. Configure

Navigate to your `play_mp3_control` directory from *Step 4. Start a Project* and configure the project:

1.10.1 Windows

```
cd %userprofile%\esp\play_mp3_control
idf.py set-target esp32
idf.py menuconfig
```

1.10.2 Linux and macOS

```
cd ~/esp/play_mp3_control
idf.py set-target esp32
idf.py menuconfig
```

注解： If you are using an ESP32-S2 based board, then the second command above should be `idf.py set-target esp32s2`.

Setting the target with `idf.py set-target <target>` should be done once, after opening a new project. If the project contains some existing builds and configuration, they will be cleared and initialized. The target may be saved in environment variable to skip this step at all. See Selecting the Target in ESP-IDF Programming Guide for additional information.

If the previous steps have been done correctly, the following menu appears:

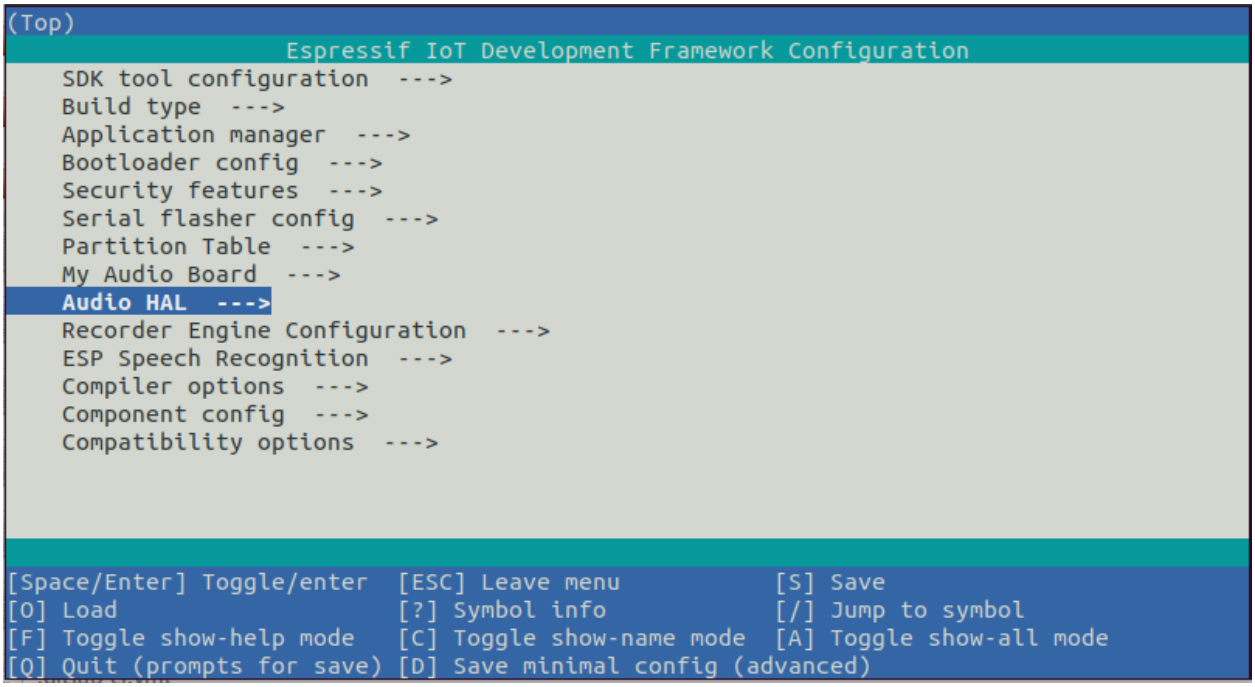


图 1: Project configuration - Home window

You are using this menu to set up your board type and other project specific variables, e.g. Wi-Fi network name and password, the processor speed, etc.

Select your board from the menu, press `S` to save configuration and then `Q` to exit.

注解: The colors of the menu could be different in your terminal. You can change the appearance with the option `--style`. Please run `idf.py menuconfig --help` for further information.

1.11 Step 7. Build the Project

Build the project by running:

```
idf.py build
```

This command will compile the application and all ESP-IDF and ESP-ADF components, then it will generate the bootloader, partition table, and application binaries.

```
$ idf.py build
 Executing action: all (aliases: build)
 Running ninja in directory /path/to/esp/play_mp3_control/build
 Executing "ninja all"...
```

(下页继续)

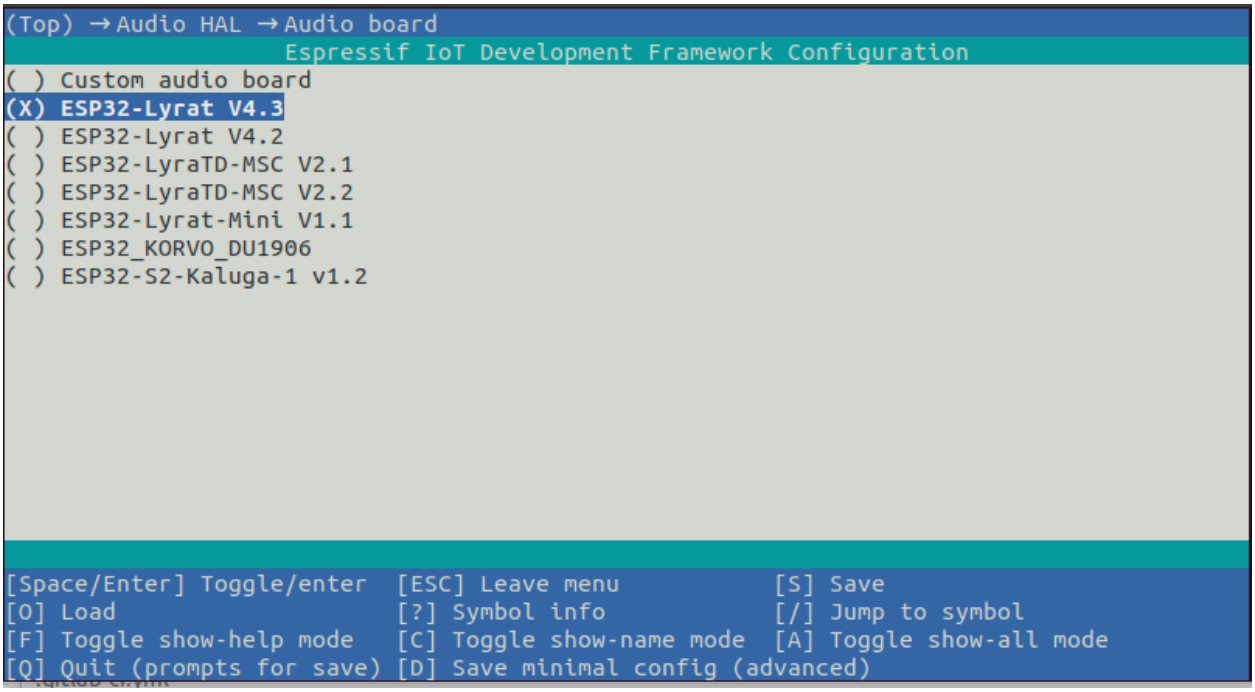


图 2: Project configuration - Board selection

(续上页)

```
 [0/1] Re-running CMake...

... (more lines of build system output)

 [1064/1064] Generating binary image from built executable
 esptool.py v3.0-dev
 Generated /path/to/esp/play_mp3_control/build/play_mp3_control.bin

 Project build complete. To flash it, run this command:
 /path/to/.espressif/python_env/idf4.2_py2.7_env/bin/python ../esp-idf/components/
↪esptool_py/esptool/esptool.py -p (PORT) -b 460800 --before default_reset --after␣
↪hard_reset --chip esp32  write_flash --flash_mode dio --flash_size detect --flash_
↪freq 40m 0x1000 build/bootloader/bootloader.bin 0x8000 build/partition_table/
↪partition-table.bin 0x10000 build/play_mp3_control.bin
 or run 'idf.py -p (PORT) flash'
```

If there are no errors, the build will finish by generating the firmware binary .bin file.

1.12 Step 8. Flash onto the Device

Flash the binaries that you just built onto your board by running:

```
idf.py -p PORT [-b BAUD] flash monitor
```

Replace PORT with your board's serial port name from *Step 5. Connect Your Device*.

You can also change the flasher baud rate by replacing BAUD with the baud rate you need. The default baud rate is `460800`.

For more information on idf.py arguments, see Using the Build System in ESP-IDF Programming Guide.

注解: The option `flash` automatically builds and flashes the project, so running `idf.py build` is not necessary.

To upload the binaries, the board should be put into upload mode. To do so, hold down **Boot** button, momentarily press **Reset** button and release the **Boot** button. The upload mode may be initiated anytime during the application build, but no later than "Connecting" message is being displayed:

```
...

esptool.py v3.0-dev
Serial port /dev/ttyUSB0
Connecting........_____....
```

Without the upload mode enabled, after showing several `.....`_____, the connection will eventually time out.

Once build and upload is complete, you should see the following:

```
...

Leaving...
Hard resetting via RTS pin...
Executing action: monitor
Running idf_monitor in directory /path/to/esp/play_mp3_control
Executing "/path/to/.espressif/python_env/idf4.2_py2.7_env/bin/python /path/to/esp/
↪esp-idf/tools/idf_monitor.py -p /dev/ttyUSB0 -b 115200 --toolchain-prefix xtensa-
↪esp32-elf- /path/to/esp/play_mp3_control/build/play_mp3_control.elf -m '/path/to/.
↪espressif/python_env/idf4.2_py2.7_env/bin/python' '/path/to/esp/esp-idf/tools/idf.py
↪'"...
--- idf_monitor on /dev/ttyUSB0 115200 ---
--- Quit: Ctrl+] | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H ---
```

If there are no issues by the end of the flash process, the board will reboot and start up the "play_mp3_control" application.

1.13 Step 9. Monitor

At this point press the **Reset** button to start the application. Following several lines of start up log, the `play_mp3_control` application specific messages should be displayed:

```
...

I (397) PLAY_FLASH_MP3_CONTROL: [ 1 ] Start audio codec chip
I (427) PLAY_FLASH_MP3_CONTROL: [ 2 ] Create audio pipeline, add all elements to␣
↪pipeline, and subscribe pipeline event
I (427) PLAY_FLASH_MP3_CONTROL: [2.1] Create mp3 decoder to decode mp3 file and set␣
↪custom read callback
I (437) PLAY_FLASH_MP3_CONTROL: [2.2] Create i2s stream to write data to codec chip
I (467) PLAY_FLASH_MP3_CONTROL: [2.3] Register all elements to audio pipeline
I (467) PLAY_FLASH_MP3_CONTROL: [2.4] Link it together [mp3_music_read_cb]-->mp3_
↪decoder-->i2s_stream-->[codec_chip]
I (477) PLAY_FLASH_MP3_CONTROL: [ 3 ] Set up  event listener
I (477) PLAY_FLASH_MP3_CONTROL: [3.1] Listening event from all elements of pipeline
I (487) PLAY_FLASH_MP3_CONTROL: [ 4 ] Start audio_pipeline
I (507) PLAY_FLASH_MP3_CONTROL: [ * ] Receive music info from mp3 decoder, sample_
↪rates=44100, bits=16, ch=2
I (7277) PLAY_FLASH_MP3_CONTROL: [ 5 ] Stop audio_pipeline
```

If there are no issues, besides the above log, you should hear a sound played for about 7 seconds by the speakers or headphones connected to your audio board. Reset the board to hear it again if required.

Now you are ready to try some other examples, or go right to developing your own applications. Check how the examples are made aware of location of the ESP-ADF. Open the get-started/play_mp3_control/Makefile and you should see

```
include($ENV{ADF_PATH}/CMakeLists.txt)
include($ENV{IDF_PATH}/tools/cmake/project.cmake)
```

The first line contains `ADF_PATH` to point the toolchain to another file in ESP-ADF directory that provides configuration variables and path to ESP-ADF components reacquired by the toolchain. You need similar `Makefile` in your own applications developed with the ESP-ADF.

1.14 VS Code Extension

1. Follow VS Code Extension Quick Installation Guide to install ESP-IDF Visual Studio Code Extension. If the previous steps have been done correctly, the following toolbar appears:
2. To install the ESP-ADF extension, open `Command Palette` and enter `install adf`. Then, a progress bar shows up in the lower right corner.

图 3: VS Code Extension Toolbar

If you have cloned the ESP-ADF repository before, please enter `open settings(ui)` in `Command Palette`. Go to `User > Extensions > ESP_IDF` and manually set the ESP-ADF path in `idf.espAdfPath` or `idf.espAdfPathWin` (for Windows). You can also set the ESP-ADF path in `.vscode/settings.json`.

3. In `Command Palette`, enter `show examples project`, and then a window will be opened with a list of example projects.
4. Select an example, click `Create project using example XX`, and select the directory to save the current example.
5. On the toolbar at the bottom of VS Code, click the gear symbol `menuconfig` to configure the example and click the column symbol `Build` to build the example. See available shortcut keys for VS code extensions.
6. On the toolbar at the bottom of VS Code, click the plug-in symbol `Select Port` to configure the serial port and click the lightning symbol `Flash Device` to flash firmware. After the firmware is flashed successfully, click `Monitor Device` to start the monitor function. Or, you can also use the flame symbol to build, flash, and monitor the example at the same time.

1.15 IDF Eclipse Plugin and Espressif IDE

1.15.1 Install and Set up Environment Variables

1. Follow IDF Eclipse Plugin Quick Installation Guide to install IDF Eclipse Plugin or download and install Espressif IDE from Espressif IDE Download Link. If the previous steps have been done correctly, you can create, build and flash IDF project in the Eclipse environment.
2. To install ESP-ADF, follow section *Step 2. Get ESP-ADF*.
3. To set `ADF_PATH` environment variable, open `Window > Preferences > C/C++ > Build > Environment` panel, click **Add** button and fill in `ADF_PATH`. After you complete the above steps, select `ADF_PATH` in `Environment variables` table and click **Edit** and **OK** button without changing any value (There is a bug in Eclipse CDT that is appending a null value before the path hence we need to click on edit and save it.).

   If this step does not work, you can delete `ADF_PATH` set in Eclipse and set `ADF_PATH` as system environment variable. For Windows, set environment variable in `Advanced System Setting` panel. For Linux and macOS, add `export ADF_PATH=your adf path` in file `/etc/profile`. However, it is not recommended. Doing so activates ADF virtual environment in every terminal session (including

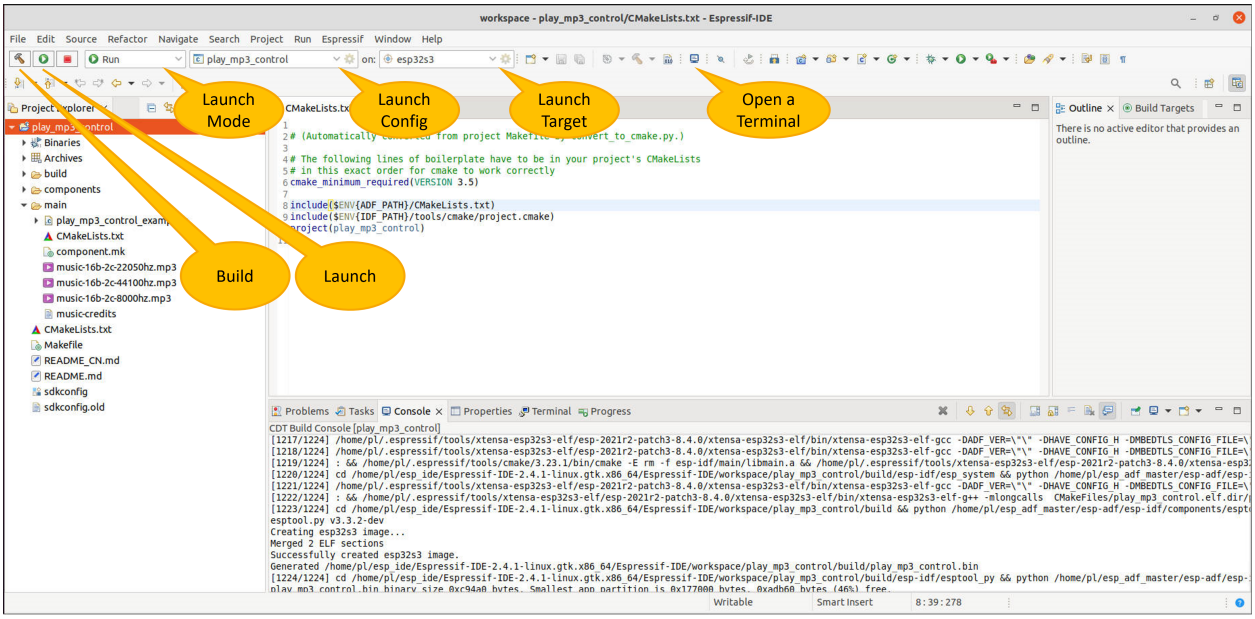


图 4: Espressif IDE (Reskinned Eclipse)

those where ADF is not needed), defeating the purpose of the virtual environment and likely affecting other software.

1.15.2 Create a New Project

1. To create new project, go to `File` > `New` > `Espressif IDF Project` and provide a project name.
2. Click **Finish** to create an empty project. Or click **Next** and check `Create a project using one of the templates` to create a project using ESP-IDF templates.

After creating a new project, you can use ESP-IDF and ESP-ADF to develop the project.

1.15.3 Import Existing Project

To import existing ESP-ADF examples, go to `File` > `Import` > `Espressif` > `Existing IDF Project` and select an ESP-ADF example (Opening an existing project directly may not be able to set the ESP target).

1.15.4 Quick Start

1. Select a project from `Project Explorer`.
2. In the **Launch Mode** drop-down menu, select `Run`.
3. In the **Launch Configuration** (auto-detected) drop-down menu, select your application.
4. Select ESP target from the third drop-down, which is called **Launch Target**. Click gear symbol **Edit** button of **Launch Target** to set `Serial Port`.
5. Double click `sdkconfig` file to launch the `SDK Configuration Editor`.
6. Click **Build** button to build the project.
7. Click **Launch** button to flash the project.
8. Click **Open a Terminal** button and select **ESP-IDF Serial Monitor** to view serial output.

For more information about IDF Eclipse Plugin and Espressif IDE, please refer to ESP-IDF Eclipse Plugin.

1.16 Update ESP-ADF

After some time of using ESP-ADF, you may want to update it to take advantage of new features or bug fixes. The simplest way to do so is by deleting existing `esp-adf` folder and cloning it again, which is same as when doing initial installation described in sections *Step 2. Get ESP-ADF*.

Another solution is to update only what has changed. This method is useful if you have a slow connection to the GitHub. To do the update run the following commands:

```
cd ~/esp/esp-adf
git pull
git submodule update --init --recursive
```

The `git pull` command is fetching and merging changes from ESP-ADF repository on GitHub. Then `git submodule update --init --recursive` is updating existing submodules or getting a fresh copy of new ones. On GitHub the submodules are represented as links to other repositories and require this additional command to get them onto your PC.

CHAPTER 2

API Reference

This API provides a way to develop audio applications using *Elements* like *Codecs* (Decoders and Encoders), *Streams* or *Audio Processing* functions.

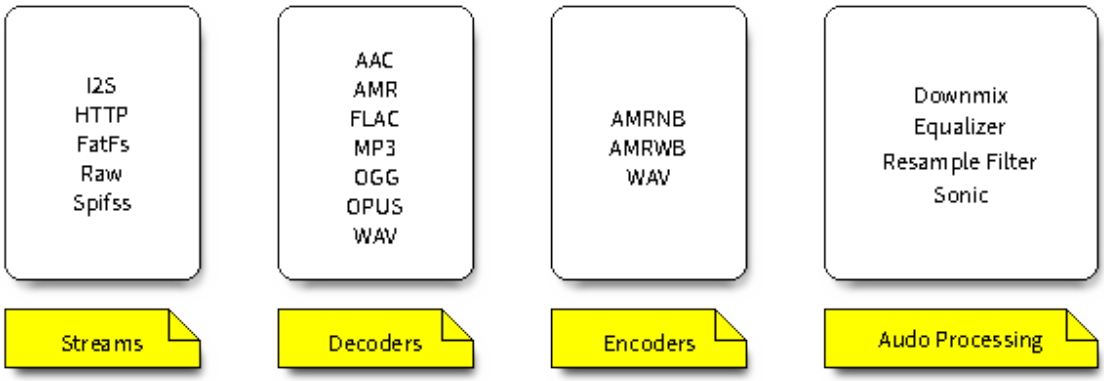


图 1: **Elements** of the Audio Development Framework

The application is developed by combining the *Elements* into a *Pipeline*. A diagram below presents organization of two elements, MP3 decoder and I2S stream, in the Audio Pipeline, that has been used in get-started/play_mp3_control example.

The audio data is typically acquired using an input *Stream*, processed with *Codecs* and in some cases with *Audio Processing* functions, and finally output with another *Stream*. There is an *Event Interface* to facilitate communication of the application events. Interfacing with specific hardware is done using *Peripherals*.

See a table of contents below with links to description of all the above components.

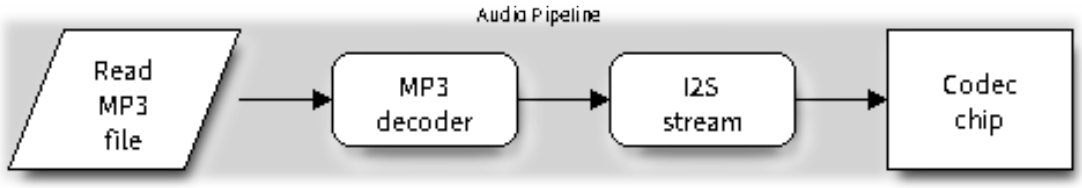


图 2: Sample Organization of Elements in Audio Pipeline

2.1 Audio Framework

2.1.1 Audio Element

The basic building block for the application programmer developing with ADF is the `audio_element` object. Every decoder, encoder, filter, input stream, or output stream is in fact an Audio Element.

This API has been designed and then used to implement Audio Elements provided by ADF.

The general functionality of an Element is to take some data on input, processes it, and output to the next. Each Element is run as a separate task. To enable control on particular stages of the data lifecycle from the input, during processing and up to the output, the `audio_element` object provides possibility to trigger callbacks per stage. There are seven types of available callback functions: open, seek, process, close, destroy, read and write, and they are defined in `audio_element_cfg_t`. Particular Elements typically use a subset of all available callbacks. For instance the *MP3 Decoder* is using open, process, close and destroy callback functions.

The available Audio Element types intended for development with this API are listed in description of *audio_common.h* header file under `audio_element_type_t` enumerator.

API Reference

Header File

- audio_pipeline/include/audio_element.h

Functions

audio_element_handle_t **audio_element_init** (*audio_element_cfg_t *config*)

Initialize audio element with config.

Return

- audio_elemenent handle object

- NULL

Parameters

- `config`: The configuration

esp_err_t **audio_element_deinit** (*audio_element_handle_t el*)

Destroy audio element handle object, stop, clear, deletel all.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] el`: The audio element handle

esp_err_t **audio_element_setdata** (*audio_element_handle_t el*, void **data*)

Set context data to element handle object. It can be retrieved by calling `audio_element_getdata`.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] el`: The audio element handle
- `data`: The data pointer

void ***audio_element_getdata** (*audio_element_handle_t el*)

Get context data from element handle object.

Return data pointer

Parameters

- `[in] el`: The audio element handle

esp_err_t **audio_element_set_tag** (*audio_element_handle_t el*, **const** char **tag*)

Set elemenet tag name, or clear if tag = NULL.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] el`: The audio element handle
- `[in] tag`: The tag name pointer

char *__audio_element_get_tag__ (*audio_element_handle_t el*)

Get element tag name.

Return Element tag name pointer

Parameters

- `[in] el`: The audio element handle

esp_err_t __audio_element_setinfo__ (*audio_element_handle_t el*, *audio_element_info_t *info*)

Set audio element infomation.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] el`: The audio element handle
- `info`: The information pointer

esp_err_t __audio_element_getinfo__ (*audio_element_handle_t el*, *audio_element_info_t *info*)

Get audio element infomation.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] el`: The audio element handle
- `info`: The information pointer

esp_err_t __audio_element_set_uri__ (*audio_element_handle_t el*, **const** char **uri*)

Set audio element URI.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] el`: The audio element handle
- `[in] uri`: The uri pointer

char ***audio_element_get_uri** (*audio_element_handle_t el*)

Get audio element URI.

Return URI pointer

Parameters

- `[in] el`: The audio element handle

esp_err_t **audio_element_run** (*audio_element_handle_t el*)

Start Audio Element. With this function, audio_element will start as freeRTOS task, and put the task into 'PAUSED' state. Note: Element does not actually start when this function returns.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] el`: The audio element handle

esp_err_t **audio_element_terminate** (*audio_element_handle_t el*)

Terminate Audio Element. With this function, audio_element will exit the task function. Note: this API only sends request. It does not actually terminate immediately when this function returns.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] el`: The audio element handle

esp_err_t **audio_element_terminate_with_ticks** (*audio_element_handle_t el*, TickType_t *ticks_to_wait*)

Terminate Audio Element with specific ticks for timeout. With this function, audio_element will exit the task function. Note: this API only sends request. It does not actually terminate immediately when this function returns.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] el`: The audio element handle
- `[in] ticks_to_wait`: The maximum amount of time to blocking

esp_err_t **audio_element_stop**(*audio_element_handle_t el*)

Request stop of the Audio Element. After receiving the stop request, the element will ignore the actions being performed (read/write, wait for the ringbuffer ···) and close the task, reset the state variables. Note: this API only sends requests, Element does not actually stop when this function returns.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] el`: The audio element handle

esp_err_t **audio_element_wait_for_stop**(*audio_element_handle_t el*)

After the `audio_element_stop` function is called, the Element task will perform some abort procedures. This function will be blocked (Time is DEFAULT_MAX_WAIT_TIME) until Element Task has done and exit.

Return

- ESP_OK, Success
- ESP_FAIL, The state is not AEL_STATE_RUNNING
- ESP_ERR_TIMEOUT, Timeout

Parameters

- `[in] el`: The audio element handle

esp_err_t **audio_element_wait_for_stop_ms**(*audio_element_handle_t el*, TickType_t *ticks_to_wait*)

After the `audio_element_stop` function is called, the Element task will perform some abort procedures. The maximum amount of time should block waiting for Element task has stopped.

Return

- ESP_OK, Success
- ESP_FAIL, The state is not AEL_STATE_RUNNING
- ESP_ERR_TIMEOUT, Timeout

Parameters

- `[in] el`: The audio element handle

- `[in] ticks_to_wait`: The maximum amount of time to wait for stop

esp_err_t **audio_element_pause**(*audio_element_handle_t el*)

Request audio Element enter 'PAUSE' state. In this state, the task will wait for any event.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] el`: The audio element handle

esp_err_t **audio_element_resume**(*audio_element_handle_t el*, float *wait_for_rb_threshold*, TickType_t *timeout*)

Request audio Element enter 'RUNNING' state. In this state, the task listens to events and invokes the callback functions. At the same time it will wait until the size/total_size of the output ringbuffer is greater than or equal to `wait_for_rb_threshold`. If the timeout period has been exceeded and ringbuffer output has not yet reached `wait_for_rb_threshold` then the function will return.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] el`: The audio element handle
- `[in] wait_for_rb_threshold`: The wait for rb threshold (0 .. 1)
- `[in] timeout`: The timeout

esp_err_t **audio_element_msg_set_listener**(*audio_element_handle_t el*, *audio_event_iface_handle_t listener*)

This function will add a `listener` to listen to all events from audio element `el`. Any event from el->external_event will be send to the `listener`.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `el`: The audio element handle
- `listener`: The event will be listen to

esp_err_t **audio_element_set_event_callback**(*audio_element_handle_t* *el*, *event_cb_func* *cb_func*, void **ctx*)

This function will add a `callback` to be called from audio element `el`. Any event to caller will cause to call callback function.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `el`: The audio element handle
- `cb_func`: The callback function
- `ctx`: Caller context

esp_err_t **audio_element_msg_remove_listener**(*audio_element_handle_t* *el*, *audio_event_iface_handle_t* *listener*)

Remove listener out of el. No new events will be sent to the listener.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] el`: The audio element handle
- `listener`: The listener

esp_err_t **audio_element_set_input_ringbuf**(*audio_element_handle_t* *el*, *ringbuf_handle_t* *rb*)

Set Element input ringbuffer.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] el`: The audio element handle
- `[in] rb`: The ringbuffer handle

ringbuf_handle_t **audio_element_get_input_ringbuf**(*audio_element_handle_t* *el*)

Get Element input ringbuffer.

Return ringbuf_handle_t

Parameters

- `[in] el`: The audio element handle

esp_err_t **audio_element_set_output_ringbuf** (*audio_element_handle_t el*, *ringbuf_handle_t rb*)

Set Element output ringbuffer.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] el`: The audio element handle
- `[in] rb`: The ringbuffer handle

ringbuf_handle_t **audio_element_get_output_ringbuf** (*audio_element_handle_t el*)

Get Element output ringbuffer.

Return ringbuf_handle_t

Parameters

- `[in] el`: The audio element handle

audio_element_state_t **audio_element_get_state** (*audio_element_handle_t el*)

Get current Element state.

Return audio_element_state_t

Parameters

- `[in] el`: The audio element handle

esp_err_t **audio_element_abort_input_ringbuf** (*audio_element_handle_t el*)

If the element is requesting data from the input ringbuffer, this function forces it to abort.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] el`: The audio element handle

esp_err_t **audio_element_abort_output_ringbuf** (*audio_element_handle_t el*)

If the element is waiting to write data to the ringbuffer output, this function forces it to abort.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] el`: The audio element handle

esp_err_t **audio_element_wait_for_buffer**(*audio_element_handle_t el*, int *size_expect*, TickType_t *timeout*)

This function will wait until the sizeof the output ringbuffer is greater than or equal to `size_expect`. If the timeout period has been exceeded and ringbuffer output has not yet reached `size_expect` then the function will return `ESP_FAIL`

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] el`: The audio element handle
- `[in] size_expect`: The size expect
- `[in] timeout`: The timeout

esp_err_t **audio_element_report_status**(*audio_element_handle_t el*, *audio_element_status_t status*)

Element will sendout event (status) to event by this function.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] el`: The audio element handle
- `[in] status`: The status

esp_err_t **audio_element_report_info**(*audio_element_handle_t el*)

Element will sendout event (information) to event by this function.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] el`: The audio element handle

esp_err_t **audio_element_report_codec_fmt** (*audio_element_handle_t el*)

Element will sendout event (codec format) to event by this function.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] el`: The audio element handle

esp_err_t **audio_element_report_pos** (*audio_element_handle_t el*)

Element will sendout event with a duplicate information by this function.

Return

- ESP_OK
- ESP_FAIL
- ESP_ERR_NO_MEM

Parameters

- `[in] el`: The audio element handle

esp_err_t **audio_element_set_input_timeout** (*audio_element_handle_t el*, TickType_t *timeout*)

Set input read timeout (default is `portMAX_DELAY`).

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] el`: The audio element handle
- `[in] timeout`: The timeout

esp_err_t **audio_element_set_output_timeout** (*audio_element_handle_t el*, TickType_t *timeout*)

Set output read timeout (default is `portMAX_DELAY`).

Return

- ESP_OK

- ESP_FAIL

Parameters

- `[in] el`: The audio element handle
- `[in] timeout`: The timeout

esp_err_t **audio_element_reset_input_ringbuf**(*audio_element_handle_t el*)

Reset inputbuffer.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] el`: The audio element handle

esp_err_t **audio_element_finish_state**(*audio_element_handle_t el*)

Set element finish state.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] el`: The audio element handle

esp_err_t **audio_element_change_cmd**(*audio_element_handle_t el*, *audio_element_msg_cmd_t cmd*)

Change element running state with specific command.

Return

- ESP_OK
- ESP_FAIL
- ESP_ERR_INVALID_ARG Element handle is null

Parameters

- `[in] el`: The audio element handle
- `[in] cmd`: Specific command from audio_element_msg_cmd_t

esp_err_t **audio_element_reset_output_ringbuf**(*audio_element_handle_t el*)

Reset outputbuffer.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] el`: The audio element handle

audio_element_err_t **audio_element_input** (*audio_element_handle_t el*, char **buffer*, int *wanted_size*)

Call this function to provide Element input data. Depending on setup using ringbuffer or function callback, Element invokes read ringbuffer, or calls read callback funtion.

Return

- > 0 number of bytes produced
- <=0 audio_element_err_t

Parameters

- `[in] el`: The audio element handle
- `buffer`: The buffer pointer
- `[in] wanted_size`: The wanted size

audio_element_err_t **audio_element_output** (*audio_element_handle_t el*, char **buffer*, int *write_size*)

Call this function to sendout Element output data. Depending on setup using ringbuffer or function callback, Element will invoke write to ringbuffer, or call write callback funtion.

Return

- > 0 number of bytes written
- <=0 audio_element_err_t

Parameters

- `[in] el`: The audio element handle
- `buffer`: The buffer pointer
- `[in] write_size`: The write size

esp_err_t **audio_element_set_read_cb** (*audio_element_handle_t el*, *stream_func fn*, void **context*)

This API allows the application to set a read callback for the first audio_element in the pipeline for allowing the pipeline to interface with other systems. The callback is invoked every time the audio element requires data to be processed.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] el`: The audio element handle
- `[in] fn`: Callback read function. The callback function should return number of bytes read or -1 in case of error in reading. Note that the callback function may decide to block and that may block the entire pipeline.
- `[in] context`: An optional context which will be passed to callback function on every invocation

esp_err_t **audio_element_set_write_cb**(*audio_element_handle_t el*, *stream_func fn*, void **context*)

This API allows the application to set a write callback for the last audio_element in the pipeline for allowing the pipeline to interface with other systems. The callback is invoked every time the audio element has a processed data that needs to be passed forward.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] el`: The audio element
- `[in] fn`: Callback write function The callback function should return number of bytes written or -1 in case of error in writing. Note that the callback function may decide to block and that may block the entire pipeline.
- `[in] context`: An optional context which will be passed to callback function on every invocation

stream_func **audio_element_get_write_cb**(*audio_element_handle_t el*)

Get callback write function that register to the element.

Return

- Callback write function pointer
- NULL Failed

Parameters

- `[in] el`: The audio element

stream_func **audio_element_get_read_cb**(*audio_element_handle_t el*)

Get callback read function that register to the element.

Return

- Callback read function pointer
- NULL Failed

Parameters

- `[in] el`: The audio element

QueueHandle_t **audio_element_get_event_queue**(*audio_element_handle_t el*)

Get External queue of Emitter. We can read any event that has been send out of Element from this `QueueHandle_t`.

Return QueueHandle_t

Parameters

- `[in] el`: The audio element handle

esp_err_t **audio_element_set_ringbuf_done**(*audio_element_handle_t el*)

Set inputbuffer and outputbuffer have finished.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] el`: The audio element handle

esp_err_t **audio_element_reset_state**(*audio_element_handle_t el*)

Enforce 'AEL_STATE_INIT' state.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] el`: The audio element handle

int **audio_element_get_output_ringbuf_size**(*audio_element_handle_t el*)

Get Element output ringbuffer size.

Return

- =0: Parameter NULL
- >0: Size of ringbuffer

Parameters

- `[in] el`: The audio element handle

esp_err_t **audio_element_set_output_ringbuf_size** (*audio_element_handle_t* *el*, int *rb_size*)

Set Element output ringbuffer size.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] el`: The audio element handle
- `[in] rb_size`: Size of the ringbuffer

int **audio_element_multi_input** (*audio_element_handle_t* *el*, char **buffer*, int *wanted_size*, int *index*, TickType_t *ticks_to_wait*)

Call this function to read data from multi input ringbuffer by given index.

Return

- ESP_ERR_INVALID_SIZE
- Others are same as the `rb_read`

Parameters

- `el`: The audio element handle
- `buffer`: The buffer pointer
- `wanted_size`: The wanted size
- `index`: The index of multi input ringbuffer, start from `0`, should be less than `NUMBER_OF_MULTI_RINGBUF`
- `ticks_to_wait`: Timeout of ringbuffer

int **audio_element_multi_output** (*audio_element_handle_t* *el*, char **buffer*, int *wanted_size*, TickType_t *ticks_to_wait*)

Call this function write data by multi output ringbuffer.

Return

- The return value is same as the `rb_write`

Parameters

- `[in] el`: The audio element handle
- `buffer`: The buffer pointer

- `[in] wanted_size`: The wanted size
- `ticks_to_wait`: Timeout of ringbuffer

esp_err_t **audio_element_set_multi_input_ringbuf**(*audio_element_handle_t* *el*, *ringbuf_handle_t* *rb*, int *index*)

Set multi input ringbuffer Element.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] el`: The audio element handle
- `[in] rb`: The ringbuffer handle
- `[in] index`: Index of multi ringbuffer, starts from `0`, should be less than `NUMBER_OF_MULTI_RINGBUF`

esp_err_t **audio_element_set_multi_output_ringbuf**(*audio_element_handle_t* *el*, *ringbuf_handle_t* *rb*, int *index*)

Set multi output ringbuffer Element.

Return

- ESP_OK
- ESP_ERR_INVALID_ARG

Parameters

- `[in] el`: The audio element handle
- `[in] rb`: The ringbuffer handle
- `[in] index`: Index of multi ringbuffer, starts from `0`, should be less than `NUMBER_OF_MULTI_RINGBUF`

ringbuf_handle_t **audio_element_get_multi_input_ringbuf**(*audio_element_handle_t* *el*, int *index*)

Get handle of multi input ringbuffer Element by index.

Return

- NULL Error
- Others ringbuf_handle_t

Parameters

- `[in] el`: The audio element handle

- `[in] index`: Index of multi ringbuffer, starts from `0`, should be less than `NUMBER_OF_MULTI_RINGBUF`

ringbuf_handle_t **audio_element_get_multi_output_ringbuf** (*audio_element_handle_t el*, int *index*)

Get handle of multi output ringbuffer Element by index.

Return

- NULL Error
- Others ringbuf_handle_t

Parameters

- `[in] el`: The audio element handle
- `[in] index`: Index of multi ringbuffer, starts from `0`, should be less than `NUMBER_OF_MULTI_RINGBUF`

esp_err_t **audio_element_process_init** (*audio_element_handle_t el*)

Provides a way to call element's `open`

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] el`: The audio element handle

esp_err_t **audio_element_process_deinit** (*audio_element_handle_t el*)

Provides a way to call element's `close`

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] el`: The audio element handle

esp_err_t **audio_element_seek** (*audio_element_handle_t el*, void **in_data*, int *in_size*, void **out_data*, int **out_size*)

Call element's `seek`

Return

- ESP_OK

- ESP_FAIL
- ESP_ERR_NOT_SUPPORTED

Parameters

- `[in] el`: The audio element handle
- `[in] in_data`: A pointer to in data
- `[in] in_size`: The size of the `in_data`
- `[out] out_data`: A pointer to the out data
- `[out] out_size`: The size of the `out_data`

bool **`audio_element_is_stopping`**(*audio_element_handle_t el*)

Get Element stopping flag.

Return element' s stopping flag

Parameters

- `[in] el`: The audio element handle

esp_err_t **`audio_element_update_byte_pos`**(*audio_element_handle_t el*, int *pos*)

Update the byte position of element information.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] el`: The audio element handle
- `[in] pos`: The byte_pos accumulated by this value

esp_err_t **`audio_element_set_byte_pos`**(*audio_element_handle_t el*, int *pos*)

Set the byte position of element information.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] el`: The audio element handle
- `[in] pos`: This value is assigned to byte_pos

esp_err_t **audio_element_update_total_bytes** (*audio_element_handle_t* *el*, int *total_bytes*)

Update the total bytes of element information.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] el`: The audio element handle
- `[in] total_bytes`: The total_bytes accumulated by this value

esp_err_t **audio_element_set_total_bytes** (*audio_element_handle_t* *el*, int *total_bytes*)

Set the total bytes of element information.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] el`: The audio element handle
- `[in] total_bytes`: This value is assigned to total_bytes

esp_err_t **audio_element_set_bps** (*audio_element_handle_t* *el*, int *bit_rate*)

Set the bps of element information.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] el`: The audio element handle
- `[in] bit_rate`: This value is assigned to bps

esp_err_t **audio_element_set_codec_fmt** (*audio_element_handle_t* *el*, int *format*)

Set the codec format of element information.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] el`: The audio element handle
- `[in] format`: This value is assigned to codec_fmt

esp_err_t **audio_element_set_music_info**(*audio_element_handle_t* *el*, int *sample_rates*, int *channels*, int *bits*)

Set the sample_rate, channels, bits of element information.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] el`: The audio element handle
- `[in] sample_rates`: Sample_rates of music information
- `[in] channels`: Channels of music information
- `[in] bits`: Bits of music information

esp_err_t **audio_element_set_duration**(*audio_element_handle_t* *el*, int *duration*)

Set the duration of element information.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] el`: The audio element handle
- `[in] duration`: This value is assigned to duration

esp_err_t **audio_element_set_reserve_user0**(*audio_element_handle_t* *el*, int *user_data0*)

Set the user_data_0 of element information.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] el`: The audio element handle
- `[in] user_data0`: This value is assigned to user_data_0

esp_err_t **audio_element_set_reserve_user1** (*audio_element_handle_t* *el*, int *user_data1*)

Set the user_data_1 of element information.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] el`: The audio element handle
- `[in] user_data1`: This value is assigned to user_data_1

esp_err_t **audio_element_set_reserve_user2** (*audio_element_handle_t* *el*, int *user_data2*)

Set the user_data_2 of element information.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] el`: The audio element handle
- `[in] user_data2`: This value is assigned to user_data_2

esp_err_t **audio_element_set_reserve_user3** (*audio_element_handle_t* *el*, int *user_data3*)

Set the user_data_3 of element information.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] el`: The audio element handle
- `[in] user_data3`: This value is assigned to user_data_3

esp_err_t **audio_element_set_reserve_user4** (*audio_element_handle_t* *el*, int *user_data4*)

Set the user_data_4 of element information.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] el`: The audio element handle
- `[in] user_data4`: This value is assigned to user_data_4

Structures

`struct audio_element_reserve_data_t`

Audio Element user reserved data.

Public Members

int **`user_data_0`**

user data 0

int **`user_data_1`**

user data 1

int **`user_data_2`**

user data 2

int **`user_data_3`**

user data 3

int **`user_data_4`**

user data 4

`struct audio_element_info_t`

Audio Element informations.

Public Members

int **`sample_rates`**

Sample rates in Hz

int **`channels`**

Number of audio channel, mono is 1, stereo is 2

int **`bits`**

Bit wide (8, 16, 24, 32 bits)

int **`bps`**

Bit per second

int64_t **`byte_pos`**

The current position (in bytes) being processed for an element

int64_t **total_bytes**

The total bytes for an element

int **duration**

The duration for an element (optional)

char ***uri**

URI (optional)

esp_codec_type_t **codec_fmt**

Music format (optional)

audio_element_reserve_data_t **reserve_data**

This value is reserved for user use (optional)

struct audio_element_cfg_t

Audio Element configurations. Each Element at startup will be a self-running task. These tasks will execute the callback open -> [loop: read -> process -> write] -> close. These callback functions are provided by the user corresponding to this configuration.

Public Members

el_io_func **open**

Open callback function

ctrl_func **seek**

Seek callback function

process_func **process**

Process callback function

el_io_func **close**

Close callback function

el_io_func **destroy**

Destroy callback function

stream_func **read**

Read callback function

stream_func **write**

Write callback function

int **buffer_len**

Buffer length use for an Element

int **task_stack**

Element task stack

int **task_prio**
: Element task priority (based on freeRTOS priority)

int **task_core**
: Element task running in core (0 or 1)

int **out_rb_size**
: Output ringbuffer size

void ***data**
: User context

const char ***tag**
: Element tag

bool **stack_in_ext**
: Try to allocate stack in external memory

int **multi_in_rb_num**
: The number of multiple input ringbuffer

int **multi_out_rb_num**
: The number of multiple output ringbuffer

Macros

AUDIO_ELEMENT_INFO_DEFAULT()

DEFAULT_ELEMENT_RINGBUF_SIZE

DEFAULT_ELEMENT_BUFFER_LENGTH

DEFAULT_ELEMENT_STACK_SIZE

DEFAULT_ELEMENT_TASK_PRIO

DEFAULT_ELEMENT_TASK_CORE

DEFAULT_AUDIO_ELEMENT_CONFIG()

Type Definitions

typedef struct audio_element ***audio_element_handle_t**

typedef esp_err_t (***el_io_func**)(*audio_element_handle_t* self)

typedef *audio_element_err_t* (***process_func**)(*audio_element_handle_t* self, char *el_buffer, int el_buf_len)

typedef *audio_element_err_t* (***stream_func**)(*audio_element_handle_t* self, char *buffer, int len, TickType_t ticks_to_wait, void *context)

typedef esp_err_t (***event_cb_func**) (*audio_element_handle_t* el, *audio_event_iface_msg_t* *event, void *ctx)

typedef esp_err_t (***ctrl_func**) (*audio_element_handle_t* self, void *in_data, int in_size, void *out_data, int *out_size)

Enumerations

enum audio_element_err_t

Values:

AEL_IO_OK = ESP_OK

AEL_IO_FAIL = ESP_FAIL

AEL_IO_DONE = -2

AEL_IO_ABORT = -3

AEL_IO_TIMEOUT = -4

AEL_PROCESS_FAIL = -5

enum audio_element_state_t

Audio element state.

Values:

AEL_STATE_NONE = 0

AEL_STATE_INIT = 1

AEL_STATE_INITIALIZING = 2

AEL_STATE_RUNNING = 3

AEL_STATE_PAUSED = 4

AEL_STATE_STOPPED = 5

AEL_STATE_FINISHED = 6

AEL_STATE_ERROR = 7

enum audio_element_msg_cmd_t

Audio element action command, process on dispatcher

Values:

AEL_MSG_CMD_NONE = 0

AEL_MSG_CMD_FINISH = 2

AEL_MSG_CMD_STOP = 3

AEL_MSG_CMD_PAUSE = 4

AEL_MSG_CMD_RESUME = 5

AEL_MSG_CMD_DESTROY = 6

AEL_MSG_CMD_REPORT_STATUS = 8

AEL_MSG_CMD_REPORT_MUSIC_INFO = 9

AEL_MSG_CMD_REPORT_CODEC_FMT = 10

AEL_MSG_CMD_REPORT_POSITION = 11

enum audio_element_status_t

Audio element status report

Values:

AEL_STATUS_NONE = 0

AEL_STATUS_ERROR_OPEN = 1

AEL_STATUS_ERROR_INPUT = 2

AEL_STATUS_ERROR_PROCESS = 3

AEL_STATUS_ERROR_OUTPUT = 4

AEL_STATUS_ERROR_CLOSE = 5

AEL_STATUS_ERROR_TIMEOUT = 6

AEL_STATUS_ERROR_UNKNOWN = 7

AEL_STATUS_INPUT_DONE = 8

AEL_STATUS_INPUT_BUFFERING = 9

AEL_STATUS_OUTPUT_DONE = 10

AEL_STATUS_OUTPUT_BUFFERING = 11

AEL_STATUS_STATE_RUNNING = 12

AEL_STATUS_STATE_PAUSED = 13

AEL_STATUS_STATE_STOPPED = 14

AEL_STATUS_STATE_FINISHED = 15

AEL_STATUS_MOUNTED = 16

AEL_STATUS_UNMOUNTED = 17

2.1.2 Audio Pipeline

Dynamic combination of a group of linked *Elements* is done using the Audio Pipeline. You do not deal with the individual elements but with just one audio pipeline. Every element is connected by a ringbuffer. The Audio Pipeline also takes care of forwarding messages from the element tasks to an application.

A diagram below presents organization of three elements, HTTP reader stream, MP3 decoder and I2S writer stream, in the Audio Pipeline, that has been used in player/pipeline_http_mp3 example.

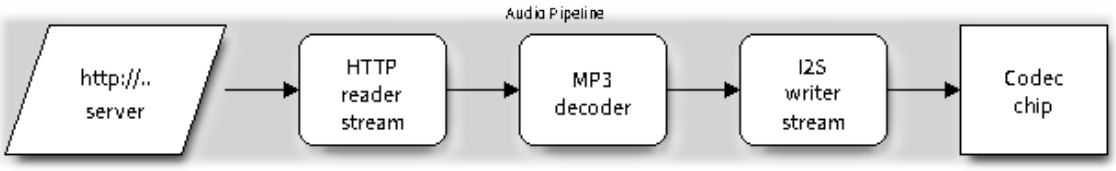


图 3: Sample Organization of Elements in Audio Pipeline

API Reference

Header File

- audio_pipeline/include/audio_pipeline.h

Functions

audio_pipeline_handle_t **audio_pipeline_init** (*audio_pipeline_cfg_t *config*)

Initialize audio_pipeline_handle_t object audio_pipeline is responsible for controlling the audio data stream and connecting the audio elements with the ringbuffer It will connect and start the audio element in order, responsible for retrieving the data from the previous element and passing it to the element after it. Also get events from each element, process events or pass it to a higher layer.

Return

- audio_pipeline_handle_t on success
- NULL when any errors

Parameters

- `config`: The configuration - audio_pipeline_cfg_t

esp_err_t **audio_pipeline_deinit** (*audio_pipeline_handle_t pipeline*)

This function removes all of the element’s links in audio_pipeline, cancels the registration of all events, invokes

the destroy functions of the registered elements, and frees the memory allocated by the init function. Briefly, frees all memory.

Return ESP_OK

Parameters

- `[in] pipeline`: The Audio Pipeline Handle

esp_err_t **audio_pipeline_register**(*audio_pipeline_handle_t pipeline*, *audio_element_handle_t el*, **const** char *name)

Registering an element for audio_pipeline, each element can be registered multiple times, but `name` (as String) must be unique in audio_pipeline, which is used to identify the element for link creation mentioned in the `audio_pipeline_link`

Note Because of stop pipeline or pause pipeline depend much on register order. Please register element strictly in the following order: input element first, process middle, output element last.

Return

- ESP_OK on success
- ESP_FAIL when any errors

Parameters

- `[in] pipeline`: The Audio Pipeline Handle
- `[in] el`: The Audio Element Handle
- `[in] name`: The name identifier of the audio_element in this audio_pipeline

esp_err_t **audio_pipeline_unregister**(*audio_pipeline_handle_t pipeline*, *audio_element_handle_t el*)

Unregister the audio_element in audio_pipeline, remove it from the list.

Return

- ESP_OK on success
- ESP_FAIL when any errors

Parameters

- `[in] pipeline`: The Audio Pipeline Handle
- `[in] el`: The Audio Element Handle

esp_err_t **audio_pipeline_run**(*audio_pipeline_handle_t pipeline*)

Start Audio Pipeline.

```
With this function audio_pipeline will create tasks for all elements,
that have been linked using the linking functions.
```

Return

- ESP_OK on success
- ESP_FAIL when any errors

Parameters

- `[in] pipeline`: The Audio Pipeline Handle

esp_err_t **audio_pipeline_terminate**(*audio_pipeline_handle_t pipeline*)

Stop Audio Pipeline.

```
With this function audio_pipeline will destroy tasks of all elements,
that have been linked using the linking functions.
```

Return

- ESP_OK on success
- ESP_FAIL when any errors

Parameters

- `[in] pipeline`: The Audio Pipeline Handle

esp_err_t **audio_pipeline_terminate_with_ticks**(*audio_pipeline_handle_t pipeline*, TickType_t *ticks_to_wait*)

Stop Audio Pipeline with specific ticks for timeout.

```
With this function audio_pipeline will destroy tasks of all elements,
that have been linked using the linking functions.
```

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] pipeline`: The Audio Pipeline Handle
- `[in] ticks_to_wait`: The maximum amount of time to block wait for element destroy

esp_err_t **audio_pipeline_resume**(*audio_pipeline_handle_t pipeline*)

This function will set all the elements to the `RUNNING` state and process the audio data as an inherent feature of audio_pipeline.

Return

- ESP_OK on success

- ESP_FAIL when any errors

Parameters

- `[in] pipeline`: The Audio Pipeline Handle

esp_err_t **audio_pipeline_pause**(*audio_pipeline_handle_t pipeline*)

This function will set all the elements to the `PAUSED` state. Everything remains the same except the data processing is stopped.

Return

- ESP_OK on success
- ESP_FAIL when any errors

Parameters

- `[in] pipeline`: The Audio Pipeline Handle

esp_err_t **audio_pipeline_stop**(*audio_pipeline_handle_t pipeline*)

Stop all of the linked elements. Used with `audio_pipeline_wait_for_stop` to keep in sync. The link state of the elements in the pipeline is kept, events are still registered. The stopped audio_pipeline restart by `audio_pipeline_resume`.

Return

- ESP_OK on success
- ESP_FAIL when any errors

Parameters

- `[in] pipeline`: The Audio Pipeline Handle

esp_err_t **audio_pipeline_wait_for_stop**(*audio_pipeline_handle_t pipeline*)

The `audio_pipeline_stop` function sends requests to the elements and exits. But they need time to get rid of time-blocking tasks. This function will wait `portMAX_DELAY` until all the Elements in the pipeline actually stop.

Return

- ESP_OK on success
- ESP_FAIL when any errors

Parameters

- `[in] pipeline`: The Audio Pipeline Handle

esp_err_t **audio_pipeline_wait_for_stop_with_ticks**(*audio_pipeline_handle_t pipeline*, TickType_t *ticks_to_wait*)

The `audio_pipeline_stop` function sends requests to the elements and exits. But they need time to get rid of time-blocking tasks. This function will wait `ticks_to_wait` until all the Elements in the pipeline actually stop.

Return

- ESP_OK on success
- ESP_FAIL when any errors

Parameters

- `[in] pipeline`: The Audio Pipeline Handle
- `[in] ticks_to_wait`: The maximum amount of time to block wait for stop

esp_err_t **audio_pipeline_link**(*audio_pipeline_handle_t pipeline*, **const** char **link_tag*[], int *link_num*)

The audio_element added to audio_pipeline will be unconnected before it is called by this function. Based on element's `name` already registered by `audio_pipeline_register`, the path of the data will be linked in the order of the link_tag. Element at index 0 is first, and index `link_num -1` is final. As well as audio_pipeline will subscribe all element's events.

Return

- ESP_OK on success
- ESP_FAIL when any errors

Parameters

- `[in] pipeline`: The Audio Pipeline Handle
- `link_tag`: Array of element `name` was registered by `audio_pipeline_register`
- `[in] link_num`: Total number of elements of the `link_tag` array

esp_err_t **audio_pipeline_unlink**(*audio_pipeline_handle_t pipeline*)

Removes the connection of the elements, as well as unsubscribe events.

Return

- ESP_OK on success
- ESP_FAIL when any errors

Parameters

- `[in] pipeline`: The Audio Pipeline Handle

audio_element_handle_t **audio_pipeline_get_el_by_tag**(*audio_pipeline_handle_t* *pipeline*, **const** char **tag*)

Find un-kept element from registered pipeline by tag.

Return

- NULL when any errors
- Others on success

Parameters

- `[in] pipeline`: The Audio Pipeline Handle
- `[in] tag`: A char pointer

audio_element_handle_t **audio_pipeline_get_el_once**(*audio_pipeline_handle_t* *pipeline*, **const** *audio_element_handle_t* *start_el*, **const** char **tag*)

Based on beginning element to find un-kept element from registered pipeline by tag.

Return

- NULL when any errors
- Others on success

Parameters

- `[in] pipeline`: The Audio Pipeline Handle
- `[in] start_el`: Specific beginning element
- `[in] tag`: A char pointer

esp_err_t **audio_pipeline_remove_listener**(*audio_pipeline_handle_t* *pipeline*)

Remove event listener from this audio_pipeline.

Return

- ESP_OK on success
- ESP_FAIL when any errors

Parameters

- `[in] pipeline`: The Audio Pipeline Handle

esp_err_t **audio_pipeline_set_listener**(*audio_pipeline_handle_t* *pipeline*, *audio_event_iface_handle_t* *evt*)

Set event listner for this audio_pipeline, any event from this pipeline can be listen to by `evt`

Return

- ESP_OK on success
- ESP_FAIL when any errors

Parameters

- `[in] pipeline`: The Audio Pipeline Handle
- `[in] evt`: The Event Handle

audio_event_iface_handle_t **audio_pipeline_get_event_iface** (*audio_pipeline_handle_t pipeline*)

Get the event iface using by this pipeline.

Return The Event Handle

Parameters

- `[in] pipeline`: The pipeline

esp_err_t **audio_pipeline_link_insert** (*audio_pipeline_handle_t pipeline*, bool *first*, *audio_element_handle_t prev*, *ringbuf_handle_t conect_rb*, *audio_element_handle_t next*)

Insert the specific audio_element to audio_pipeline, previous element connect to the next element by ring buffer.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] pipeline`: The audio pipeline handle
- `[in] first`: Previous element is first input element, need to set `true`
- `[in] prev`: Previous element
- `[in] conect_rb`: Connect ring buffer
- `[in] next`: Next element

esp_err_t **audio_pipeline_register_more** (*audio_pipeline_handle_t pipeline*, *audio_element_handle_t element_1*, ...)

Register a NULL-terminated list of elements to audio_pipeline.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] pipeline`: The audio pipeline handle

- `[in] element_1`: The element to add to the audio_pipeline.
- `[in] ...`: Additional elements to add to the audio_pipeline.

esp_err_t **audio_pipeline_unregister_more**(*audio_pipeline_handle_t pipeline*, *audio_element_handle_t element_1*, ...)

Unregister a NULL-terminated list of elements to audio_pipeline.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] pipeline`: The audio pipeline handle
- `[in] element_1`: The element to add to the audio_pipeline.
- `[in] ...`: Additional elements to add to the audio_pipeline.

esp_err_t **audio_pipeline_link_more**(*audio_pipeline_handle_t pipeline*, *audio_element_handle_t element_1*, ...)

Adds a NULL-terminated list of elements to audio_pipeline.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] pipeline`: The audio pipeline handle
- `[in] element_1`: The element to add to the audio_pipeline.
- `[in] ...`: Additional elements to add to the audio_pipeline.

esp_err_t **audio_pipeline_listen_more**(*audio_pipeline_handle_t pipeline*, *audio_element_handle_t element_1*, ...)

Subscribe a NULL-terminated list of element's events to audio_pipeline.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] pipeline`: The audio pipeline handle
- `[in] element_1`: The element event to subscribe to the audio_pipeline.

- `[in] ...`: Additional elements event to subscribe to the audio_pipeline.

esp_err_t **audio_pipeline_check_items_state**(*audio_pipeline_handle_t* *pipeline*, *audio_element_handle_t* *dest_el*, *audio_element_status_t* *status*)

Update the destination element state and check the all of linked elements state are same.

Return

- ESP_OK All linked elements state are same.
- ESP_FAIL All linked elements state are not same.

Parameters

- `[in] pipeline`: The audio pipeline handle
- `[in] dest_el`: Destination element
- `[in] status`: The new status

esp_err_t **audio_pipeline_reset_items_state**(*audio_pipeline_handle_t* *pipeline*)

Reset pipeline element items state to `AEL_STATUS_NONE`

Return

- ESP_OK on success
- ESP_FAIL when any errors

Parameters

- `[in] pipeline`: The Audio Pipeline Handle

esp_err_t **audio_pipeline_reset_ringbuffer**(*audio_pipeline_handle_t* *pipeline*)

Reset pipeline element ringbuffer.

Return

- ESP_OK on success
- ESP_FAIL when any errors

Parameters

- `[in] pipeline`: The Audio Pipeline Handle

esp_err_t **audio_pipeline_reset_elements**(*audio_pipeline_handle_t* *pipeline*)

Reset Pipeline linked elements state.

Return

- ESP_OK on success

- ESP_FAIL when any errors

Parameters

- `[in] pipeline`: The Audio Pipeline Handle

esp_err_t **audio_pipeline_reset_kept_state**(*audio_pipeline_handle_t pipeline*, *audio_element_handle_t el*)

Reset the specific element kept state.

Return

- ESP_OK on success
- ESP_FAIL when any errors

Parameters

- `[in] pipeline`: The Audio Pipeline Handle
- `[in] el`: The Audio element Handle

esp_err_t **audio_pipeline_breakup_elements**(*audio_pipeline_handle_t pipeline*, *audio_element_handle_t kept_ctx_el*)

Break up all the linked elements of specific `pipeline`. The include and before `kept_ctx_el` working (AEL_STATE_RUNNING or AEL_STATE_PAUSED) elements and connected ringbuffer will be reserved.

Note There is no element reserved when `kept_ctx_el` is NULL. This function will unsubscribe all element's events.

Return

- ESP_OK All linked elements state are same.
- ESP_ERR_INVALID_ARG Invalid parameters.

Parameters

- `[in] pipeline`: The audio pipeline handle
- `[in] kept_ctx_el`: Destination keep elements

esp_err_t **audio_pipeline_relink**(*audio_pipeline_handle_t pipeline*, **const** char **link_tag*[], int *link_num*)

Basing on element's `name` already registered by `audio_pipeline_register`, relink the pipeline following the order of `names` in the 'link_tag.

Note If the ringbuffer is not enough to connect the new pipeline will create new ringbuffer.

Return

- ESP_OK All linked elements state are same.
- ESP_FAIL Error.

- ESP_ERR_INVALID_ARG Invalid parameters.

Parameters

- `[in] pipeline`: The Audio Pipeline Handle
- `link_tag`: Array of elements `name` that was registered by `audio_pipeline_register`
- `[in] link_num`: Total number of elements of the `link_tag` array

esp_err_t **audio_pipeline_relink_more**(*audio_pipeline_handle_t pipeline*, *audio_element_handle_t element_1*, ...)

Adds a NULL-terminated list of elements to audio_pipeline.

Note If the ringbuffer is not enough to connect the new pipeline will create new ringbuffer.

Return

- ESP_OK All linked elements state are same.
- ESP_FAIL Error.
- ESP_ERR_INVALID_ARG Invalid parameters.

Parameters

- `[in] pipeline`: The Audio Pipeline Handle
- `[in] element_1`: The element to add to the audio_pipeline.
- `[in] ...`: Additional elements to add to the audio_pipeline.

esp_err_t **audio_pipeline_change_state**(*audio_pipeline_handle_t pipeline*, *audio_element_state_t new_state*)

Set the pipeline state.

Return

- ESP_OK All linked elements state are same.
- ESP_FAIL Error.

Parameters

- `[in] pipeline`: The Audio Pipeline Handle
- `[in] new_state`: The new state will be set

Structures

struct audio_pipeline_cfg

Audio Pipeline configurations.

Public Members

int **rb_size**

Audio Pipeline ringbuffer size

Macros

DEFAULT_PIPELINE_RINGBUF_SIZE

DEFAULT_AUDIO_PIPELINE_CONFIG()

Type Definitions

typedef struct audio_pipeline ***audio_pipeline_handle_t**

typedef struct *audio_pipeline_cfg* **audio_pipeline_cfg_t**

Audio Pipeline configurations.

2.1.3 Event Interface

The ADF provides the Event Interface API to establish communication between Audio Elements in a pipeline. The API is built around FreeRTOS queue. It implements 'listeners' to watch for incoming messages and inform about them with a callback function.

Application Examples

Implementation of this API is demonstrated in couple of examples including get-started/play_mp3_control.

API Reference

Header File

- audio_pipeline/include/audio_event_iface.h

Functions

audio_event_iface_handle_t **audio_event_iface_init** (*audio_event_iface_cfg_t *config*)

Initialize audio event.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `config`: The configurations

esp_err_t **audio_event_iface_destroy** (*audio_event_iface_handle_t evt*)

Cleanup event, it doesn't free evt pointer.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `evt`: The event

esp_err_t **audio_event_iface_set_listener** (*audio_event_iface_handle_t evt*, *audio_event_iface_handle_t listener*)

Add audio event `evt` to the listener, then we can listen `evt` event from `listen`

Return

- ESP_OK
- ESP_FAIL

Parameters

- `listener`: The event can listen another event
- `evt`: The event to be added to

esp_err_t **audio_event_iface_remove_listener** (*audio_event_iface_handle_t listener*, *audio_event_iface_handle_t evt*)

Remove audio event `evt` from the listener.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `listener`: The event listener
- `evt`: The event to be removed from

esp_err_t **audio_event_iface_set_cmd_waiting_timeout**(*audio_event_iface_handle_t evt*, TickType_t *wait_time*)

Set current queue wait time for the event.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `evt`: The event
- `[in] wait_time`: The wait time

esp_err_t **audio_event_iface_waiting_cmd_msg**(*audio_event_iface_handle_t evt*)

Waiting internal queue message.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `evt`: The event

esp_err_t **audio_event_iface_cmd**(*audio_event_iface_handle_t evt*, *audio_event_iface_msg_t *msg*)

Trigger an event for internal queue with a message.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `evt`: The event
- `msg`: The message

esp_err_t **audio_event_iface_cmd_from_isr**(*audio_event_iface_handle_t evt*, *audio_event_iface_msg_t *msg*)

It's same with `audio_event_iface_cmd`, but can send a message from ISR.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] evt`: The event
- `msg`: The message

esp_err_t **audio_event_iface_sendout**(*audio_event_iface_handle_t evt*, *audio_event_iface_msg_t *msg*)

Trigger and event out with a message.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `evt`: The event
- `msg`: The message

esp_err_t **audio_event_iface_discard**(*audio_event_iface_handle_t evt*)

Discard all ongoing event message.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `evt`: The event

esp_err_t **audio_event_iface_listen**(*audio_event_iface_handle_t evt*, *audio_event_iface_msg_t *msg*, TickType_t *wait_time*)

Listening and invoke callback function if there are any event are comming.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `evt`: The event

- `msg`: The message
- `wait_time`: The wait time

QueueHandle_t **audio_event_iface_get_queue_handle**(*audio_event_iface_handle_t evt*)

Get External queue handle of Emmitter.

Return External QueueHandle_t

Parameters

- `[in] evt`: The external queue

esp_err_t **audio_event_iface_read**(*audio_event_iface_handle_t evt*, *audio_event_iface_msg_t *msg*, TickType_t *wait_time*)

Read the event from all the registered event emitters in the queue set of the interface.

Return

- ESP_OK On successful receiving of event
- ESP_FAIL In case of a timeout or invalid parameter passed

Parameters

- `[in] evt`: The event interface
- `[out] msg`: The pointer to structure in which event is to be received
- `[in] wait_time`: Timeout for receiving event

QueueHandle_t **audio_event_iface_get_msg_queue_handle**(*audio_event_iface_handle_t evt*)

Get Internal queue handle of Emmitter.

Return Internal QueueHandle_t

Parameters

- `[in] evt`: The Internal queue

esp_err_t **audio_event_iface_set_msg_listener**(*audio_event_iface_handle_t evt*, *audio_event_iface_handle_t listener*)

Add audio internal event `evt` to the listener, then we can listen `evt` event from `listen`

Return

- ESP_OK
- ESP_FAIL

Parameters

- `listener`: The event can listen another event

- `evt`: The event to be added to

Structures

struct audio_event_iface_msg_t

Event message

Public Members

int **cmd**

Command id

void ***data**

Data pointer

int **data_len**

Data length

void ***source**

Source event

int **source_type**

Source type (To know where it came from)

bool **need_free_data**

Need to free data pointer after the event has been processed

struct audio_event_iface_cfg_t

Event interface configurations

Public Members

int **internal_queue_size**

It's optional, Queue size for event `internal_queue`

int **external_queue_size**

It's optional, Queue size for event `external_queue`

int **queue_set_size**

It's optional, QueueSet size for event `queue_set`

on_event_iface_func **on_cmd**

Function callback for listener when any event arrived

void ***context**

Context will pass to callback function

TickType_t **wait_time**

Timeout to check for event queue

int **type**

it will pass to *audio_event_iface_msg_t* source_type (To know where it came from)

Macros

DEFAULT_AUDIO_EVENT_IFACE_SIZE

AUDIO_EVENT_IFACE_DEFAULT_CFG()

Type Definitions

typedef esp_err_t (***on_event_iface_func**) (*audio_event_iface_msg_t* *, void *)

typedef struct audio_event_iface ***audio_event_iface_handle_t**

2.1.4 Audio Common

Enumerations that define type of *Audio Elements*, type and format of *Codecs* and type of *Streams*.

API Reference

Header File

- audio_pipeline/include/audio_common.h

Macros

ELEMENT_SUB_TYPE_OFFSET

mem_assert(x)

Enumerations

enum audio_element_type_t

Values:

AUDIO_ELEMENT_TYPE_UNKNOW = 0x01 « ELEMENT_SUB_TYPE_OFFSET

AUDIO_ELEMENT_TYPE_ELEMENT = 0x01 « (ELEMENT_SUB_TYPE_OFFSET + 1)

AUDIO_ELEMENT_TYPE_PLAYER = 0x01 « (ELEMENT_SUB_TYPE_OFFSET + 2)

AUDIO_ELEMENT_TYPE_SERVICE = 0x01 « (ELEMENT_SUB_TYPE_OFFSET + 3)

AUDIO_ELEMENT_TYPE_PERIPH = 0x01 « (ELEMENT_SUB_TYPE_OFFSET + 4)

enum audio_stream_type_t

Values:

AUDIO_STREAM_NONE = 0

AUDIO_STREAM_READER

AUDIO_STREAM_WRITER

enum audio_codec_type_t

Values:

AUDIO_CODEC_TYPE_NONE = 0

AUDIO_CODEC_TYPE_DECODER

AUDIO_CODEC_TYPE_ENCODER

2.1.5 ESP Audio

This component provides several simple high level APIs. It is intended for quick implementation of audio applications based on typical interconnections of standardized audio elements.

API Reference

Header File

- esp-adf-libs/esp_audio/include/audio_def.h

Structures

struct esp_audio_state_t

esp_audio status information parameters

Public Members

esp_audio_status_t **status**

Status of esp_audio

audio_err_t **err_msg**

Status is `AUDIO_STATUS_ERROR`, err_msg will be setup

media_source_type_t **media_src**

Media source type

Macros

ESP_ERR_AUDIO_BASE

Starting number of ESP audio error codes

Type Definitions

typedef void (***esp_audio_event_callback**) (*esp_audio_state_t* *audio, void *ctx)

typedef esp_err_t (***audio_volume_set**) (void *hd, int vol)

typedef esp_err_t (***audio_volume_get**) (void *hd, int *vol)

Enumerations

enum audio_err_t

Values:

ESP_ERR_AUDIO_NO_ERROR = ESP_OK

ESP_ERR_AUDIO_FAIL = ESP_FAIL

ESP_ERR_AUDIO_NO_INPUT_STREAM = ESP_ERR_AUDIO_BASE + 1

ESP_ERR_AUDIO_NO_OUTPUT_STREAM = ESP_ERR_AUDIO_BASE + 2

ESP_ERR_AUDIO_NO_CODEC = ESP_ERR_AUDIO_BASE + 3

ESP_ERR_AUDIO_HAL_FAIL = ESP_ERR_AUDIO_BASE + 4

ESP_ERR_AUDIO_MEMORY_LACK = ESP_ERR_AUDIO_BASE + 5

ESP_ERR_AUDIO_INVALID_URI = ESP_ERR_AUDIO_BASE + 6

ESP_ERR_AUDIO_INVALID_PATH = ESP_ERR_AUDIO_BASE + 7

ESP_ERR_AUDIO_INVALID_PARAMETER = ESP_ERR_AUDIO_BASE + 8

ESP_ERR_AUDIO_NOT_READY = ESP_ERR_AUDIO_BASE + 9

ESP_ERR_AUDIO_NOT_SUPPORT = ESP_ERR_AUDIO_BASE + 10

ESP_ERR_AUDIO_TIMEOUT = ESP_ERR_AUDIO_BASE + 11

ESP_ERR_AUDIO_ALREADY_EXISTS = ESP_ERR_AUDIO_BASE + 12

ESP_ERR_AUDIO_LINK_FAIL = ESP_ERR_AUDIO_BASE + 13

ESP_ERR_AUDIO_UNKNOWN = ESP_ERR_AUDIO_BASE + 14

ESP_ERR_AUDIO_OUT_OF_RANGE = ESP_ERR_AUDIO_BASE + 15

ESP_ERR_AUDIO_STOP_BY_USER = ESP_ERR_AUDIO_BASE + 16

ESP_ERR_AUDIO_OPEN = ESP_ERR_AUDIO_BASE + 0x100

ESP_ERR_AUDIO_INPUT = ESP_ERR_AUDIO_BASE + 0x101

ESP_ERR_AUDIO_PROCESS = ESP_ERR_AUDIO_BASE + 0x102

ESP_ERR_AUDIO_OUTPUT = ESP_ERR_AUDIO_BASE + 0x103

ESP_ERR_AUDIO_CLOSE = ESP_ERR_AUDIO_BASE + 0x104

enum esp_audio_status_t

Values:

AUDIO_STATUS_UNKNOWN = 0

AUDIO_STATUS_RUNNING = 1

AUDIO_STATUS_PAUSED = 2

AUDIO_STATUS_STOPPED = 3

AUDIO_STATUS_FINISHED = 4

AUDIO_STATUS_ERROR = 5

enum audio_termination_type_t

Values:

TERMINATION_TYPE_NOW = 0

Audio operation will be terminated immediately

TERMINATION_TYPE_DONE = 1

Audio operation will be stopped when finished

TERMINATION_TYPE_MAX

enum esp_audio_prefer_t

Values:

ESP_AUDIO_PREFER_MEM = 0

ESP_AUDIO_PREFER_SPEED = 1

enum media_source_type_t

Values:

MEDIA_SRC_TYPE_NULL = 0

MEDIA_SRC_TYPE_MUSIC_BASE = 0x100

MEDIA_SRC_TYPE_MUSIC_SD = *MEDIA_SRC_TYPE_MUSIC_BASE* + 1

MEDIA_SRC_TYPE_MUSIC_HTTP = *MEDIA_SRC_TYPE_MUSIC_BASE* + 2

MEDIA_SRC_TYPE_MUSIC_FLASH = *MEDIA_SRC_TYPE_MUSIC_BASE* + 3

MEDIA_SRC_TYPE_MUSIC_A2DP = *MEDIA_SRC_TYPE_MUSIC_BASE* + 4

MEDIA_SRC_TYPE_MUSIC_DLNA = *MEDIA_SRC_TYPE_MUSIC_BASE* + 5

MEDIA_SRC_TYPE_MUSIC_RAW = *MEDIA_SRC_TYPE_MUSIC_BASE* + 6

MEDIA_SRC_TYPE_MUSIC_MAX = 0x1FF

MEDIA_SRC_TYPE_TONE_BASE = 0x200

MEDIA_SRC_TYPE_TONE_SD = *MEDIA_SRC_TYPE_TONE_BASE* + 1

MEDIA_SRC_TYPE_TONE_HTTP = *MEDIA_SRC_TYPE_TONE_BASE* + 2

MEDIA_SRC_TYPE_TONE_FLASH = *MEDIA_SRC_TYPE_TONE_BASE* + 3

MEDIA_SRC_TYPE_TONE_MAX = 0x2FF

MEDIA_SRC_TYPE_RESERVE_BASE = 0x800

MEDIA_SRC_TYPE_RESERVE_MAX = 0xFFF

Header File

- esp-adf-libs/esp_audio/include/esp_audio.h

Functions

esp_audio_handle_t **esp_audio_create**(**const** *esp_audio_cfg_t* **cfg*)

Create esp_audio instance according to ‘cfg’ parameter.

This function create an esp_audio instance, at the specified configuration.

Return

- NULL: Error
- Others: esp_audio instance fully certifying

Parameters

- `[in] cfg`: Provide esp_audio initialization configuration

audio_err_t **esp_audio_destroy**(*esp_audio_handle_t handle*)

Specific esp_audio instance will be destroyed.

Return

- ESP_ERR_AUDIO_NO_ERROR: on success
- ESP_ERR_AUDIO_INVALID_PARAMETER: no instance to free, call esp_audio_init first

Parameters

- `[in] handle`: The esp_audio instance

audio_err_t **esp_audio_input_stream_add**(*esp_audio_handle_t handle*, *audio_element_handle_t in_stream*)

Add audio input stream to specific esp_audio instance.

Return

- ESP_ERR_AUDIO_NO_ERROR: on success
- ESP_ERR_AUDIO_INVALID_PARAMETER: invalid arguments
- ESP_ERR_AUDIO_MEMORY_LACK: allocate memory fail

Parameters

- `[in] handle`: The esp_audio instance
- `[in] in_stream`: Audio stream instance

audio_err_t **esp_audio_output_stream_add**(*esp_audio_handle_t handle*, *audio_element_handle_t out_stream*)

Add audio output stream to specific esp_audio instance.

Return

- ESP_ERR_AUDIO_NO_ERROR: on success
- ESP_ERR_AUDIO_INVALID_PARAMETER: invalid arguments
- ESP_ERR_AUDIO_MEMORY_LACK: allocate memory fail

Parameters

- `[in] handle`: The esp_audio instance
- `[in] out_stream`: The audio stream element instance

audio_err_t **esp_audio_codec_lib_add**(*esp_audio_handle_t handle*, *audio_codec_type_t type*, *audio_element_handle_t lib*)

Add a new codec lib that can decode or encode a music file.

Return

- ESP_ERR_AUDIO_NO_ERROR: on success
- ESP_ERR_AUDIO_INVALID_PARAMETER: invalid arguments
- ESP_ERR_AUDIO_MEMORY_LACK: allocate memory fail

Parameters

- `[in] handle`: The esp_audio instance
- `[in] type`: The audio codec type(encoder or decoder)
- `[in] lib`: To provide audio stream element

audio_err_t **esp_audio_codec_lib_query**(*esp_audio_handle_t handle*, *audio_codec_type_t type*, **const** char ***extension*)

Check if this kind of music extension is supported or not.

Note This function just query the codec which has already add by esp_audio_codec_lib_add. The max length of extension is 6.

Return

- ESP_ERR_AUDIO_NO_ERROR: supported
- ESP_ERR_AUDIO_NOT_SUPPORT: not support
- ESP_ERR_AUDIO_INVALID_PARAMETER: invalid arguments

Parameters

- `[in] handle`: The esp_audio instance
- `[in] type`: The CODEC_ENCODER or CODEC_DECODER
- `[in] extension`: Such as "mp3", "wav", "aac"

audio_err_t **esp_audio_play**(*esp_audio_handle_t handle*, *audio_codec_type_t type*, **const** char ***uri*, int *pos*)

Play the given uri.

The esp_audio_play have follow activity, setup inputstream, outputstream and codec by uri, start all of them. There is a rule that esp_audio will select input stream, codec and output stream by URI field.

Rule of URI field are as follow.

- `UF_SCHEMA` field of URI for choose input stream from existing streams. e.g:" http" ," file"
- `UF_PATH` field of URI for choose codec from existing codecs. e.g:" /audio/mp3_music.mp3"
- `UF_FRAGMENT` field of URI for choose output stream from existing streams, output stream is I2S by default.
- `UF_USERINFO` field of URI for specific sample rate and channels at encode mode.

The format "user:password" in the userinfo field, "user" is sample rate, "password" is channels.

Now esp_audio_play support follow URIs.

- " https://dl.espressif.com/dl/audio/mp3_music.mp3"
- " http://media-ice.musicradio.com/ClassicFMMP3"
- " file://sdcard/test.mp3"
- " iis://16000:2@from.pcm/rec.wav#file"
- " iis://16000:1@record.pcm/record.wav#raw"
- " aadp://44100:2@bt/sink/stream.pcm"
- " hfp://8000:1@bt/hfp/stream.pcm"

Note

- The URI parse by `http_parser_parse_url`,any illegal string will be return `ESP_ERR_AUDIO_INVALID_URI`.
- If the esp_decoder codec is added to `handle`, then the `handle` of esp_decoder will be set as the default decoder, even if other decoders are added.
- Enabled `CONFIG_FATFS_API_ENCODING_UTF_8`, the URI can be support Chinese characters.
- Asynchronous interface
- The maximum of block time can be modify by `esp_audio_play_timeout_set`, default value is 25 seconds.

Return

- ESP_ERR_AUDIO_NO_ERROR: on success
- ESP_ERR_AUDIO_TIMEOUT: timeout the play activity
- ESP_ERR_AUDIO_NOT_SUPPORT: Currently status is AUDIO_STATUS_RUNNING
- ESP_ERR_AUDIO_INVALID_URI: URI is illegal
- ESP_ERR_AUDIO_INVALID_PARAMETER: invalid arguments
- ESP_ERR_AUDIO_STOP_BY_USER: Exit without play due to esp_audio_stop has been called.

Parameters

- `handle`: The esp_audio_handle_t instance
- `uri`: Such as "file://sdcard/test.wav" or "http://iot.espressif.com/file/example.mp3". If NULL to be set, the uri setup by`esp_audio_setup` will used.
- `type`: Specific handle type decoder or encoder
- `pos`: Specific starting position by bytes

audio_err_t **esp_audio_sync_play** (*esp_audio_handle_t handle*, **const** char **uri*, int *pos*)

Play the given uri until music finished or error occured.

Note

- All features are same with `esp_audio_play`
- Synchronous interface
- Support decoder mode only
- No any events post during playing

Return

- ESP_ERR_AUDIO_NO_ERROR: on success

- ESP_ERR_AUDIO_TIMEOUT: timeout the play activity
- ESP_ERR_AUDIO_NOT_SUPPORT: Currently status is AUDIO_STATUS_RUNNING
- ESP_ERR_AUDIO_INVALID_URI: URI is illegal
- ESP_ERR_AUDIO_INVALID_PARAMETER: invalid arguments

Parameters

- `handle`: The esp_audio_handle_t instance
- `uri`: Such as "file://sdcard/test.wav" or "http://iot.espressif.com/file/example.mp3",
- `pos`: Specific starting position by bytes

audio_err_t **esp_audio_stop**(*esp_audio_handle_t handle*, *audio_termination_type_t type*)

A synchronous interface for stop the esp_audio. The maximum of block time is 8000ms.

Note 1. If user queue has been registered by evt_que, AUDIO_STATUS_STOPPED event for success or AUDIO_STATUS_ERROR event for error will be received.

1. `TERMINATION_TYPE_DONE` only works with input stream which can't stopped by itself, e.g. `raw read/write stream`, others streams are no effect.
2. The synchronous interface is used to ensure that working pipeline is stopped.

Return

- ESP_ERR_AUDIO_NO_ERROR: on success
- ESP_ERR_AUDIO_INVALID_PARAMETER: invalid arguments
- ESP_ERR_AUDIO_NOT_READY: The status is not AUDIO_STATUS_RUNNING or AUDIO_STATUS_PAUSED or element has not created
- ESP_ERR_AUDIO_TIMEOUT: timeout(8000ms) the stop activity.

Parameters

- `[in] handle`: The esp_audio instance
- `[in] type`: Stop immediately or done

audio_err_t **esp_audio_pause**(*esp_audio_handle_t handle*)

Pause the esp_audio.

Note 1. Only support music and without live stream. If user queue has been registered by evt_que, AUDIO_STATUS_PAUSED event for success or AUDIO_STATUS_ERROR event for error will be received.

1. The Paused music must be stoped by `esp_audio_stop` before new playing, otherwise got block on new play.

Return

- ESP_ERR_AUDIO_NO_ERROR: on success
- ESP_ERR_AUDIO_INVALID_PARAMETER: invalid arguments
- ESP_ERR_AUDIO_NOT_READY: the status is not running
- ESP_ERR_AUDIO_TIMEOUT: timeout the pause activity.

Parameters

- `[in] handle`: The esp_audio instance

audio_err_t **esp_audio_resume** (*esp_audio_handle_t handle*)

Resume the music paused.

Note Only support music and without live stream. If user queue has been registered by evt_que, AUDIO_STATUS_RUNNING event for success or AUDIO_STATUS_ERROR event for error will be received.

Return

- ESP_ERR_AUDIO_NO_ERROR: on success
- ESP_ERR_AUDIO_INVALID_PARAMETER: invalid arguments
- ESP_ERR_AUDIO_TIMEOUT: timeout the resume activity.

Parameters

- `[in] handle`: The esp_audio instance

audio_err_t **esp_audio_eq_gain_set** (*esp_audio_handle_t handle*, int *band_index*, int *nch*, int *eq_gain*)

Set the audio gain to be processed by the equalizer.

Return

- ESP_ERR_AUDIO_NO_ERROR: on success
- ESP_ERR_AUDIO_INVALID_PARAMETER: invalid arguments

Parameters

- `[in] handle`: The esp_audio instance
- `[in] band_index`: The position of center frequencies of equalizer. The range of eq band index is [0 - 9].
- `[in] nch`: The number of channel. As for mono, the nch can only set to 1. As for dual, thc nch can set to 1 and 2.
- `[in] eq_gain`: The value of audio gain which in `band_index`.

audio_err_t **esp_audio_eq_gain_get** (*esp_audio_handle_t handle*, int *band_index*, int *nch*, int **eq_gain*)

Get the audio gain to be processed by the equalizer.

Return

- ESP_ERR_AUDIO_NO_ERROR: on success
- ESP_ERR_AUDIO_INVALID_PARAMETER: invalid arguments

Parameters

- `[in] handle`: Audio element handle
- `[in] band_index`: The position of center frequencies of equalizer. The range of eq band index is [0 - 9].
- `[in] nch`: The number of channel. As for mono, the nch can only set to 1. As for dual, thc nch can set to 1 and 2.
- `[out] eq_gain`: The pointer of the gain processed by equalizer

audio_err_t **esp_audio_speed_get**(*esp_audio_handle_t handle*, *esp_audio_play_speed_t *speed_index*)

Getting esp_audio play speed index, index value is from "esp_audio_speed_t" enum.

Return

- ESP_ERR_AUDIO_NO_ERROR: on success
- ESP_ERR_AUDIO_INVALID_PARAMETER: invalid arguments

Parameters

- `[in] handle`: The esp_audio instance
- `[out] speed_index`: Current audio play speed index.

audio_err_t **esp_audio_speed_set**(*esp_audio_handle_t handle*, *esp_audio_play_speed_t speed_index*)

Use speed_index which is from "esp_audio_speed_t" enum to set esp_audio play speed.

Return

- ESP_ERR_AUDIO_NO_ERROR: on success
- ESP_ERR_AUDIO_INVALID_PARAMETER: invalid arguments

Parameters

- `[in] handle`: The esp_audio instance
- `[in] speed_index`: Value from "esp_audio_speed_t" enum.

audio_err_t **esp_audio_speed_idx_to_float**(*esp_audio_handle_t handle*, *esp_audio_play_speed_t speed_index*, float *speed*)

Use speed_index which is from "esp_audio_speed_t" enum to get esp_audio play speed which is float type.

Return

- ESP_ERR_AUDIO_NO_ERROR: on success
- ESP_ERR_AUDIO_INVALID_PARAMETER: invalid arguments

Parameters

- `[in] handle`: The esp_audio instance
- `[in] speed_index`: Current audio play speed index.
- `[out] speed`: Current audio play speed.

audio_err_t **esp_audio_vol_set** (*esp_audio_handle_t handle*, int *vol*)

Setting esp_audio volume.

Return

- ESP_ERR_AUDIO_NO_ERROR: on success
- ESP_ERR_AUDIO_CTRL_HAL_FAIL: error with hardware.
- ESP_ERR_AUDIO_INVALID_PARAMETER: invalid arguments

Parameters

- `[in] handle`: The esp_audio instance
- `[in] vol`: Specific volume will be set. 0-100 is legal. 0 will be mute.

audio_err_t **esp_audio_vol_get** (*esp_audio_handle_t handle*, int **vol*)

Get esp_audio volume.

Return

- ESP_ERR_AUDIO_NO_ERROR: on success
- ESP_ERR_AUDIO_CTRL_HAL_FAIL: error with hardware.
- ESP_ERR_AUDIO_INVALID_PARAMETER: invalid arguments

Parameters

- `[in] handle`: The esp_audio instance
- `[out] vol`: A pointer to int that indicates esp_audio volume.

audio_err_t **esp_audio_state_get** (*esp_audio_handle_t handle*, *esp_audio_state_t* **state*)

Get esp_audio status.

Return

- ESP_ERR_AUDIO_NO_ERROR: on success
- ESP_ERR_AUDIO_INVALID_PARAMETER: no esp_audio instance or esp_audio does not playing

Parameters

- `[in] handle`: The esp_audio instance
- `[out] state`: A pointer to *esp_audio_state_t* that indicates esp_audio status.

audio_err_t **esp_audio_pos_get** (*esp_audio_handle_t handle*, int **pos*)

Get the position in bytes of currently played music.

Note This function works only with decoding music.

Return

- ESP_ERR_AUDIO_NO_ERROR: on success
- ESP_ERR_AUDIO_INVALID_PARAMETER: no esp_audio instance
- ESP_ERR_AUDIO_NOT_READY: no codec element

Parameters

- `[in] handle`: The esp_audio instance
- `[out] pos`: A pointer to int that indicates esp_audio decoding position.

audio_err_t **esp_audio_time_get** (*esp_audio_handle_t handle*, int **time*)

Get the position in microseconds of currently played music.

Note This function works only with decoding music.

Return

- ESP_ERR_AUDIO_NO_ERROR: on success
- ESP_ERR_AUDIO_INVALID_PARAMETER: no esp_audio instance
- ESP_ERR_AUDIO_NOT_READY: no out stream

Parameters

- `[in] handle`: The esp_audio instance
- `[out] time`: A pointer to int that indicates esp_audio decoding position.

audio_err_t **esp_audio_setup** (*esp_audio_handle_t handle*, *esp_audio_setup_t *sets*)

Choose the `in_stream`, `codec` and `out_stream` definitely, and set `uri`.

Note This function provide a manual way to select in/out stream and codec, should be called before the `esp_audio_play`, then ignore the `esp_audio_play` URI parameter only one time.

Return

- ESP_ERR_AUDIO_NO_ERROR: on success
- ESP_ERR_AUDIO_INVALID_PARAMETER: no esp_audio instance

- ESP_ERR_AUDIO_MEMORY_LACK: allocate memory fail

Parameters

- `[in] handle`: The esp_audio instance
- `[in] sets`: A pointer to *esp_audio_setup_t*.

audio_err_t **esp_audio_media_type_set**(*esp_audio_handle_t handle*, *media_source_type_t type*)

audio_err_t **esp_audio_music_info_get**(*esp_audio_handle_t handle*, *esp_audio_music_info_t *info*)

audio_err_t **esp_audio_info_get**(*esp_audio_handle_t handle*, *esp_audio_info_t *info*)

audio_err_t **esp_audio_info_set**(*esp_audio_handle_t handle*, *esp_audio_info_t *info*)

audio_err_t **esp_audio_callback_set**(*esp_audio_handle_t handle*, *esp_audio_event_callback cb*, void *cb_ctx*)

audio_err_t **esp_audio_seek**(*esp_audio_handle_t handle*, int *seek_time_sec*)

Seek the position in second of currently played music.

Note This function works only with decoding music.

Return

- ESP_ERR_AUDIO_NO_ERROR: on success
- ESP_ERR_AUDIO_FAIL: codec or allocation fail
- ESP_ERR_AUDIO_TIMEOUT: timeout for sync the element status
- ESP_ERR_AUDIO_INVALID_PARAMETER: no esp_audio instance
- ESP_ERR_AUDIO_NOT_SUPPORT: codec has finished
- ESP_ERR_AUDIO_OUT_OF_RANGE: the seek_time_ms is out of the range
- ESP_ERR_AUDIO_NOT_READY: the status is neither running nor paused

Parameters

- `[in] handle`: The esp_audio instance
- `[out] seek_time_sec`: A pointer to int that indicates esp_audio decoding position.

audio_err_t **esp_audio_duration_get**(*esp_audio_handle_t handle*, int *duration*)

Get the duration in microseconds of playing music.

Note This function works only with decoding music.

Return

- ESP_ERR_AUDIO_NO_ERROR: on success
- ESP_ERR_AUDIO_INVALID_PARAMETER: no esp_audio instance

- ESP_ERR_AUDIO_NOT_READY: no codec element or no in element

Parameters

- `[in] handle`: The esp_audio instance
- `[out] duration`: A pointer to int that indicates decoding total time.

audio_err_t **esp_audio_play_timeout_set** (*esp_audio_handle_t handle*, int *time_ms*)

Setting the maximum amount of time to waiting for `esp_audio_play` only.

Return

- ESP_ERR_AUDIO_NO_ERROR: on success
- ESP_ERR_AUDIO_INVALID_PARAMETER: invalid arguments

Parameters

- `[in] handle`: The esp_audio instance
- `[in] time_ms`: The maximum amount of time

audio_err_t **esp_audio_prefer_type_get** (*esp_audio_handle_t handle*, *esp_audio_prefer_t *type*)

Get the type of `esp_audio_prefer_t`

Return

- ESP_ERR_AUDIO_NO_ERROR: on success
- ESP_ERR_AUDIO_INVALID_PARAMETER: no esp_audio instance

Parameters

- `[in] handle`: The esp_audio instance
- `[out] type`: A pointer to esp_audio_prefer_t

audio_err_t **esp_audio_event_que_set** (*esp_audio_handle_t handle*, QueueHandle_t *que*)

Set event queue to notify the esp_audio status.

Return

- ESP_ERR_AUDIO_NO_ERROR: on success
- ESP_ERR_AUDIO_INVALID_PARAMETER: no esp_audio instance

Parameters

- `[in] handle`: The esp_audio instance
- `[out] que`: A pointer to QueueHandle_t

Structures

struct esp_audio_cfg_t

esp_audio configuration parameters

Public Members

int **in_stream_buf_size**

Input buffer size

int **out_stream_buf_size**

Output buffer size

int **resample_rate**

Destination sample rate, 0: disable resample; others: 44.1K, 48K, 32K, 16K, 8K has supported It should be make sure same with I2S stream `sample_rate`

int **component_select**

The select of audio forge component. eg. To choose equalizer and ALC together, please enter ESP_AUDIO_COMPONENT_SELECT_ALC | ESP_AUDIO_COMPONENT_SELECT_EQUALIZER.

QueueHandle_t **evt_que**

For received esp_audio events (optional)

esp_audio_event_callback **cb_func**

esp_audio events callback (optional)

void ***cb_ctx**

esp_audio callback context (optional)

esp_audio_prefer_t **prefer_type**

esp_audio works on sepcific type, default memory is preferred.

- `ESP_AUDIO_PREFER_MEM` mode stopped the previous linked elements before the new pipeline starting, except out stream element.
- `ESP_AUDIO_PREFER_SPEED` mode kept the previous linked elements before the new pipeline starting, except out stream element.

void ***vol_handle**

Volume change instance

audio_volume_set **vol_set**

Set volume callback

audio_volume_get **vol_get**

Get volume callback

int **task_prio**

esp_audio task priority

int **task_stack**

Size of esp_audio task stack

struct esp_audio_setup_t

esp_audio setup parameters by manual

Public Members

audio_codec_type_t **set_type**

Set codec type

int **set_sample_rate**

Set music sample rate

int **set_channel**

Set music channels

int **set_pos**

Set starting position

int **set_time**

Set starting position of the microseconds time (optional)

char ***set_uri**

Set URI

char ***set_in_stream**

Tag of in_stream

char ***set_codec**

Tag of the codec

char ***set_out_stream**

Tag of out_stream

struct esp_audio_info_t

esp_audio information

Public Members

audio_element_info_t **codec_info**

Codec information

audio_element_handle_t **in_el**

Handle of the in stream

audio_element_handle_t **out_el**

Handle of the out stream

audio_element_handle_t **codec_el**

Handle of the codec

audio_element_handle_t **filter_el**

Handle of the filter

esp_audio_state_t **st**

The state of esp_audio

int **time_pos**

Position of the microseconds time

float **audio_speed**

Play speed of audio

int64_t **in_stream_total_size**

Total size of in stream

struct esp_audio_music_info_t

The music informations.

Public Members

int **sample_rates**

Sample rates in Hz

int **channels**

Number of audio channel, mono is 1, stereo is 2

int **bits**

Bit wide (8, 16, 24, 32 bits)

int **bps**

Bit per second

esp_codec_type_t **codec_fmt**

Music format

Macros

ESP_AUDIO_COMPONENT_SELECT_DEFAULT

Default selected

ESP_AUDIO_COMPONENT_SELECT_ALC

ALC selected

ESP_AUDIO_COMPONENT_SELECT_EQUALIZER

Equalizer selected

DEFAULT_ESP_AUDIO_CONFIG()

Type Definitions

typedef void ***esp_audio_handle_t**

Enumerations

enum esp_audio_play_speed_t

esp_audio play speed

Values:

ESP_AUDIO_PLAY_SPEED_UNKNOW = -1

ESP_AUDIO_PLAY_SPEED_0_50 = 0

ESP_AUDIO_PLAY_SPEED_0_75 = 1

ESP_AUDIO_PLAY_SPEED_1_00 = 2

ESP_AUDIO_PLAY_SPEED_1_25 = 3

ESP_AUDIO_PLAY_SPEED_1_50 = 4

ESP_AUDIO_PLAY_SPEED_1_75 = 5

ESP_AUDIO_PLAY_SPEED_2_00 = 6

ESP_AUDIO_PLAY_SPEED_MAX = 7

2.2 音频流

[English]

音频流指的是负责获取和处理音频数据并将处理后的音频发送出去的音频元素。

以下为支持的音频流：

- 算法流
- *FatFs* 流
- *HTTP* 流
- *I2S* 流
- *PWM* 流
- 原始流
- *SPIFFS* 流
- *TCP* 客户端流
- 提示音流
- 嵌入 *Flash* 流
- 语音合成流

每个流 (stream) 有一个结构体作为输入参数进行初始化，其返回的 `audio_element_handle_t` 句柄用来调用 `audio_element.h` 中的函数。大多数流具有 `AUDIO_STREAM_READER` （读）和 `AUDIO_STREAM_WRITER` （写）两种类型。例如，可使用*`i2s_stream_init()`* 和*`i2s_stream_cfg_t`* 来设置 I2S 流的类型。

有关 API 的详细信息，请参阅下文。

2.2.1 算法流

算法流 (algorithm stream) 集成声学回声消除 (AEC)、自动增益控制 (AGC)、噪声抑制 (NS) 等前端算法，用于处理收取的音频，常用于 VoIP、语音识别、关键词唤醒等需要预处理的场景。算法流调用 esp-sr，故占用较大内存，且只支持 `AUDIO_STREAM_READER` 类型。

应用示例

- advanced_examples/algorithm/main/algorithm_examples.c
- protocols/voip/main/voip_app.c

Header File

- audio_stream/include/algorithm_stream.h

Functions

audio_element_handle_t **algo_stream_init**(*algorithm_stream_cfg_t* **config*)

Initialize algorithm stream.

Return The audio element handle

Parameters

- `config`: The algorithm Stream configuration

audio_element_err_t **algo_stream_set_delay**(*audio_element_handle_t* *el*, *ringbuf_handle_t* *ringbuf*, int *delay_ms*)

Set playback signal or recording signal delay when use type2.

Note The AEC internal buffering mechanism requires that the recording signal is delayed by around 0 - 10 ms compared to the corresponding reference (playback) signal.

Return

- ESP_OK
- ESP_FAIL
- ESP_ERR_INVALID_ARG

Parameters

- `el`: Handle of element
- `ringbuf`: Handle of ringbuf

- `delay_ms`: The delay between playback and recording in ms This delay_ms can be debugged by yourself, you can set the configuration debug_input to true, then get the original input data (left channel is the signal captured from the microphone, right channel is the signal played to the speaker), and check the delay with an audio analysis tool.

esp_err_t **algorithm_mono_fix**(uint8_t **sbuff*, uint32_t *len*)

Fix I2S mono noise issue.

Note This API only for ESP32 with I2S 16bits

Return ESP_OK

Parameters

- `sbuff`: I2S data buffer
- `len`: I2S data len

Structures

struct algorithm_stream_cfg_t

Algorithm stream configurations.

Public Members

algorithm_stream_input_type_t **input_type**

Input type of stream

int **task_stack**

Task stack size

int **task_prio**

Task peroid

int **task_core**

The core that task to be created

int **out_rb_size**

Size of output ringbuffer

bool **stack_in_ext**

Try to allocate stack in external memory

int **rec_linear_factor**

The linear amplication factor of record signal

int **ref_linear_factor**

The linear amplication factor of reference signal

bool **debug_input**

debug algorithm input data

bool **swap_ch**

Swap left and right channels

int8_t **algo_mask**

Choose algorithm to use

int **sample_rate**

The sampling rate of the input PCM (in Hz)

int **mic_ch**

MIC channel num

int **agc_gain**

AGC gain(dB) for voice communication

bool **aec_low_cost**

AEC uses less cpu and ram resources, but has poor suppression of nonlinear distortion

Macros

ALGORITHM_STREAM_PINNED_TO_CORE

ALGORITHM_STREAM_TASK_PERIOD

ALGORITHM_STREAM_RINGBUFFER_SIZE

ALGORITHM_STREAM_TASK_STACK_SIZE

ALGORITHM_STREAM_DEFAULT_SAMPLE_RATE_HZ

ALGORITHM_STREAM_DEFAULT_SAMPLE_BIT

ALGORITHM_STREAM_DEFAULT_MIC_CHANNELS

ALGORITHM_STREAM_DEFAULT_AGC_GAIN_DB

ALGORITHM_STREAM_DEFAULT_MASK

ALGORITHM_STREAM_CFG_DEFAULT( )

Enumerations

enum algorithm_stream_input_type_t

Two types of algorithm stream input method.

Values:

ALGORITHM_STREAM_INPUT_TYPE1 = 0

Type 1 is default used by mini-board, the reference signal and the recording signal are respectively read in from the left channel and the right channel of the same I2S

ALGORITHM_STREAM_INPUT_TYPE2 = 1

As the simple diagram above shows, when type2 is choosen, the recording signal and reference signal should be input by users.

enum algorithm_stream_mask_t

Choose the algorithm to be used.

Values:

ALGORITHM_STREAM_USE_AEC = (0x1 « 0)

Use AEC

ALGORITHM_STREAM_USE_AGC = (0x1 « 1)

Use AGC

ALGORITHM_STREAM_USE_NS = (0x1 « 2)

Use NS

ALGORITHM_STREAM_USE_VAD = (0x1 « 3)

Use VAD

2.2.2 FatFs 流

FatFs 流从 FatFs 文件系统中读取和写入数据，具有读和写两种类型，类型由*audio_stream_type_t* 定义。

应用示例

- 读类型示例：player/pipeline_play_sdcard_music
- 写类型示例：recorder/pipeline_recording_to_sdcard

Header File

- audio_stream/include/fatfs_stream.h

Functions

audio_element_handle_t **fatfs_stream_init** (*fatfs_stream_cfg_t *config*)

Create a handle to an Audio Element to stream data from FatFs to another Element or get data from other elements written to FatFs, depending on the configuration the stream type, either AUDIO_STREAM_READER or AUDIO_STREAM_WRITER.

Return The Audio Element handle

Parameters

- `config`: The configuration

Structures

struct fatfs_stream_cfg_t

FATFS Stream configurations, if any entry is zero then the configuration will be set to default values.

Public Members

audio_stream_type_t **type**

Stream type

int **buf_sz**

Audio Element Buffer size

int **out_rb_size**

Size of output ringbuffer

int **task_stack**

Task stack size

int **task_core**

Task running in core (0 or 1)

int **task_prio**

Task priority (based on freeRTOS priority)

bool **ext_stack**

Allocate stack on extern ram

bool **write_header**

Choose to write amrnb/amrwb header in fatfs whether or not (true or false, true means choose to write amrnb header)

Macros

FATFS_STREAM_BUF_SIZE

FATFS_STREAM_TASK_STACK

FATFS_STREAM_TASK_CORE

FATFS_STREAM_TASK_PRIO

FATFS_STREAM_RINGBUFFER_SIZE

FATFS_STREAM_CFG_DEFAULT()

2.2.3 HTTP 流

HTTP 流通过 esp_http_client() 获取和发送数据，具有读和写两种类型，类型由*audio_stream_type_t* 定义。AUDIO_STREAM_READER 支持 HTTP、HTTPS 和 HTTP 流直播流协议 (HTTP Live Stream) 等协议，使用前需要连接网络。

应用示例

- 读类型示例
  - player/pipeline_living_stream
  - player/pipeline_http_mp3
- 写类型示例
  - recorder/pipeline_raw_http

Header File

- audio_stream/include/http_stream.h

Functions

audio_element_handle_t **http_stream_init** (*http_stream_cfg_t *config*)

Create a handle to an Audio Element to stream data from HTTP to another Element or get data from other elements sent to HTTP, depending on the configuration the stream type, either AUDIO_STREAM_READER or AUDIO_STREAM_WRITER.

Return The Audio Element handle

Parameters

- `config`: The configuration

esp_err_t **http_stream_next_track** (*audio_element_handle_t el*)

Connect to next track in the playlist.

```
        This function can be used in event_handler of http_stream.
        User can call this function to connect to next track in playlist when he/
↪she gets `HTTP_STREAM_FINISH_TRACK` event
```

Return

- ESP_OK on success
- ESP_FAIL on errors

Parameters

- `el`: The http_stream element handle

esp_err_t **http_stream_restart** (*audio_element_handle_t el*)

esp_err_t **http_stream_fetch_again** (*audio_element_handle_t el*)

Try to fetch the tracks again.

```
        If this is live stream we will need to keep fetching URIs.
```

Return

- ESP_OK on success
- ESP_ERR_NOT_SUPPORTED if playlist is finished

Parameters

- `el`: The http_stream element handle

esp_err_t **http_stream_set_server_cert** (*audio_element_handle_t el*, **const** char *cert)

Set SSL server certification.

Note EM format as string, if the client requires to verify server

Return

- ESP_OK on success

Parameters

- `el`: The http_stream element handle
- `cert`: server certification

Structures

struct http_stream_event_msg_t

Stream event message.

Public Members

http_stream_event_id_t **event_id**

Event ID

void ***http_client**

Reference to HTTP Client using by this HTTP Stream

void ***buffer**

Reference to Buffer using by the Audio Element

int **buffer_len**

Length of buffer

void ***user_data**

User data context, from *http_stream_cfg_t*

audio_element_handle_t **el**

Audio element context

struct http_stream_cfg_t

HTTP Stream configurations Default value will be used if any entry is zero.

Public Members

audio_stream_type_t **type**

Type of stream

int **out_rb_size**

Size of output ringbuffer

int **task_stack**

Task stack size

int **task_core**

Task running in core (0 or 1)

int **task_prio**

Task priority (based on freeRTOS priority)

bool **stack_in_ext**

Try to allocate stack in external memory

http_stream_event_handle_t **event_handle**

The hook function for HTTP Stream

void ***user_data**

User data context

bool **auto_connect_next_track**

connect next track without open/close

bool **enable_playlist_parser**

Enable playlist parser

int **multi_out_num**

The number of multiple output

const char ***cert_pem**

SSL server certification, PEM format as string, if the client requires to verify server

esp_err_t (***crt_bundle_attach**) (void *conf)

Function pointer to esp_crt_bundle_attach. Enables the use of certification bundle for server verification, must be enabled in menuconfig

int **request_size**

Request data size each time from `http_client` Defaults use DEFAULT_ELEMENT_BUFFER_LENGTH if set to 0 Need care this setting if audio frame size is small and want low latency playback

int **request_range_size**

Range size setting for header `Range: bytes=start-end` Request full range of resource if set to 0 Range size bigger than request size is recommended

const char ***user_agent**

The User Agent string to send with HTTP requests

Macros

HTTP_STREAM_TASK_STACK

HTTP_STREAM_TASK_CORE

HTTP_STREAM_TASK_PRIO

HTTP_STREAM_RINGBUFFER_SIZE

HTTP_STREAM_CFG_DEFAULT()

Type Definitions

typedef int (***http_stream_event_handle_t**)(*http_stream_event_msg_t* *msg)

Enumerations

enum http_stream_event_id_t

HTTP Stream hook type.

Values:

HTTP_STREAM_PRE_REQUEST = 0x01

The event handler will be called before HTTP Client making the connection to the server

HTTP_STREAM_ON_REQUEST

The event handler will be called when HTTP Client is requesting data, If the fucntion return the value (-1: ESP_FAIL), HTTP Client will be stopped If the fucntion return the value > 0, HTTP Stream will ignore the post_field If the fucntion return the value = 0, HTTP Stream continue send data from post_field (if any)

HTTP_STREAM_ON_RESPONSE

The event handler will be called when HTTP Client is receiving data If the fucntion return the value (-1: ESP_FAIL), HTTP Client will be stopped If the fucntion return the value > 0, HTTP Stream will ignore the read function If the fucntion return the value = 0, HTTP Stream continue read data from HTTP Server

HTTP_STREAM_POST_REQUEST

The event handler will be called after HTTP Client send header and body to the server, before fetching the headers

HTTP_STREAM_FINISH_REQUEST

The event handler will be called after HTTP Client fetch the header and ready to read HTTP body

HTTP_STREAM_RESOLVE_ALL_TRACKS

HTTP_STREAM_FINISH_TRACK

HTTP_STREAM_FINISH_PLAYLIST

2.2.4 I2S 流

I2S 流通过芯片的 I2S、PDM、ADC、DAC 接口接收和发送音频数据，其中 ADC、DAC 功能需要芯片定义 `SOC_I2S_SUPPORTS_ADC_DAC`。I2S 流还集成自动电平控制 (ALC) 来调节音量，多通道输出和扩展发送音频数据位宽，相关控制位定义在*`i2s_stream_cfg_t`* 中。

应用示例

- 读类型示例：recorder/pipeline_wav_amr_sdcard
- 写类型示例：get-started/play_mp3_control

Header File

- audio_stream/include/i2s_stream.h

Functions

audio_element_handle_t **i2s_stream_init** (*i2s_stream_cfg_t *config*)

Create a handle to an Audio Element to stream data from I2S to another Element or get data from other elements sent to I2S, depending on the configuration of stream type is AUDIO_STREAM_READER or AUDIO_STREAM_WRITER.

Note If I2S stream is enabled with built-in DAC mode, please don ' t use I2S_NUM_1. The built-in DAC functions are only supported on I2S0 for the current ESP32 chip.

Return The Audio Element handle

Parameters

- `config`: The configuration

esp_err_t **i2s_stream_set_channel_type** (*i2s_stream_cfg_t *config*, *i2s_channel_type_t type*)

Set I2S stream channel format type.

Note ：This function only updates *i2s_stream_cfg_t*, so it needs to be called before i2s_stream_init.

Return

- ESP_OK
- ESP_ERR_INVALID_ARG

Parameters

- `[in] config`: The I2S stream configuration
- `[in] type`: I2S channel format type

esp_err_t **i2s_stream_set_clk** (*audio_element_handle_t i2s_stream*, int *rate*, int *bits*, int *ch*)

Setup clock for I2S Stream, this function is only used with handle created by `i2s_stream_init`

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] i2s_stream`: The i2s element handle
- `[in] rate`: Clock rate (in Hz)
- `[in] bits`: Audio bit width (8, 16, 24, 32)
- `[in] ch`: Number of Audio channels (1: Mono, 2: Stereo). But when set to tdm mode, ch is slot mask.(ex: I2S_TDM_SLOT0 | I2S_TDM_SLOT1 | I2S_TDM_SLOT2 | I2S_TDM_SLOT3)

esp_err_t **i2s_alc_volume_set** (*audio_element_handle_t i2s_stream*, int *volume*)

Setup volume of stream by using ALC.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] i2s_stream`: The i2s element handle
- `[in] volume`: The volume of stream will be set.

esp_err_t **i2s_alc_volume_get** (*audio_element_handle_t i2s_stream*, int **volume*)

Get volume of stream.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] i2s_stream`: The i2s element handle
- `[in] volume`: The volume of stream

esp_err_t **i2s_stream_sync_delay** (*audio_element_handle_t i2s_stream*, int *delay_ms*)

Set sync delay of stream.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] i2s_stream`: The i2s element handle
- `[in] delay_ms`: The delay of stream

Structures

`struct i2s_stream_cfg_t`

I2S Stream configurations Default value will be used if any entry is zero.

Public Members

audio_stream_type_t **`type`**

Type of stream

i2s_comm_mode_t **`transmit_mode`**

I2S transmit mode

i2s_chan_config_t **`chan_cfg`**

I2S controller channel configuration

i2s_std_config_t **`std_cfg`**

I2S standard mode major configuration that including clock/slot/gpio configuration

i2s_data_bit_width_t **`expand_src_bits`**

The source bits per sample when data expand

bool **`use_alc`**

It is a flag for ALC. If use ALC, the value is true. Or the value is false

int **`volume`**

The volume of audio input data will be set.

int **`out_rb_size`**

Size of output ringbuffer

int **`task_stack`**

Task stack size

int **`task_core`**

Task running in core (0 or 1)

int **`task_prio`**

Task priority (based on freeRTOS priority)

bool **stack_in_ext**

Try to allocate stack in external memory

int **multi_out_num**

The number of multiple output

bool **uninstall_drv**

whether uninstall the i2s driver when stream destroyed

bool **need_expand**

whether to expand i2s data

int **buffer_len**

Buffer length use for an Element. Note: when 'bits_per_sample' is 24 bit, the buffer length must be a multiple of 3. The recommended value is 3600

Macros

I2S_STREAM_TASK_STACK

I2S_STREAM_BUF_SIZE

I2S_STREAM_TASK_PRIO

I2S_STREAM_TASK_CORE

I2S_STREAM_RINGBUFFER_SIZE

I2S_STREAM_CFG_DEFAULT()

I2S_STREAM_CFG_DEFAULT_WITH_PARA(port, rate, bits, stream_type)

Enumerations

enum i2s_channel_type_t

Values:

I2S_CHANNEL_TYPE_RIGHT_LEFT

Separated left and right channel

I2S_CHANNEL_TYPE_ALL_RIGHT

Load right channel data in both two channels

I2S_CHANNEL_TYPE_ALL_LEFT

Load left channel data in both two channels

I2S_CHANNEL_TYPE_ONLY_RIGHT

Only load data in right channel (mono mode)

I2S_CHANNEL_TYPE_ONLY_LEFT

Only load data in left channel (mono mode)

2.2.5 PWM 流

一些成本敏感的场景中，音频信号不用 DAC 进行转换，而是使用 PWM 对信号进行调制并经过滤波电路实现。PWM 流实现了音频数据用芯片的 PWM 进行调制并发送的功能，只有 `AUDIO_STREAM_WRITER` 类型。注意，PWM 的数模转换信噪较低。

应用示例

- 写类型示例：player/pipeline_play_mp3_with_dac_or_pwm

Header File

- audio_stream/include/pwm_stream.h

Functions

audio_element_handle_t **pwm_stream_init** (*pwm_stream_cfg_t* **config*)

Initialize PWM stream Only support AUDIO_STREAM_READER type.

Return The audio element handle

Parameters

- `config`: The PWM Stream configuration

esp_err_t **pwm_stream_set_clk** (*audio_element_handle_t* *pwm_stream*, int *rate*, int *bits*, int *ch*)

Setup clock for PWM Stream, this function is only used with handle created by `pwm_stream_init`

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] pwm_stream`: The pwm element handle
- `[in] rate`: Clock rate (in Hz)
- `[in] bits`: Audio bit width (16, 32)
- `[in] ch`: Number of Audio channels (1: Mono, 2: Stereo)

Structures

struct audio_pwm_config_t

PWM audio configurations.

Public Members

timer_group_t **tg_num**

timer group number (0 - 1)

timer_idx_t **timer_num**

timer number (0 - 1)

int **gpio_num_left**

the LEDC output gpio_num, Left channel

int **gpio_num_right**

the LEDC output gpio_num, Right channel

ledc_channel_t **ledc_channel_left**

LEDC channel (0 - 7), Corresponding to left channel

ledc_channel_t **ledc_channel_right**

LEDC channel (0 - 7), Corresponding to right channel

ledc_timer_t **ledc_timer_sel**

Select the timer source of channel (0 - 3)

ledc_timer_bit_t **duty_resolution**

ledc pwm bits

uint32_t **data_len**

ringbuffer size

struct pwm_stream_cfg_t

PWM Stream configurations Default value will be used if any entry is zero.

Public Members

audio_stream_type_t **type**

Type of stream

audio_pwm_config_t **pwm_config**

driver configurations

int **out_rb_size**

Size of output ringbuffer

int **task_stack**
: Task stack size

int **task_core**
: Task running in core (0 or 1)

int **task_prio**
: Task priority (based on freeRTOS priority)

int **buffer_len**
: pwm_stream buffer length

bool **ext_stack**
: Allocate stack on extern ram

Macros

PWM_STREAM_GPIO_NUM_LEFT

PWM_STREAM_GPIO_NUM_RIGHT

PWM_STREAM_TASK_STACK

PWM_STREAM_BUF_SIZE

PWM_STREAM_TASK_PRIO

PWM_STREAM_TASK_CORE

PWM_STREAM_RINGBUFFER_SIZE

PWM_CONFIG_RINGBUFFER_SIZE

PWM_STREAM_CFG_DEFAULT()

2.2.6 原始流

原始流 (raw stream) 用于获取连接的前级元素输出数据，或者为后级连接的元素填充数据，本身不建立线程。AUDIO_STREAM_READER 的应用方式为 [i2s] -> [filter] -> [raw] 或 [i2s] -> [codec-amr] -> [raw]，AUDIO_STREAM_WRITER 的应用方式为 [raw]->[codec-mp3]->[i2s]。

应用示例

- 读类型示例：protocols/voip
- 写类型示例：advanced_examples/downmix_pipeline

Header File

- audio_stream/include/raw_stream.h

Functions

audio_element_handle_t **raw_stream_init** (*raw_stream_cfg_t* *cfg)

Initialize RAW stream.

Return The audio element handle

Parameters

- `cfg`: The RAW Stream configuration

int **raw_stream_read** (*audio_element_handle_t* *pipeline*, char *buffer, int *buf_size*)

Read data from Stream.

Return Number of bytes actually read.

Parameters

- `pipeline`: The audio pipeline handle
- `buffer`: The buffer
- `buf_size`: Maximum number of bytes to be read.

int **raw_stream_write** (*audio_element_handle_t* *pipeline*, char *buffer, int *buf_size*)

Write data to Stream.

Return Number of bytes written

Parameters

- `pipeline`: The audio pipeline handle
- `buffer`: The buffer
- `buf_size`: Number of bytes to write

Structures

struct raw_stream_cfg_t

Raw stream provides APIs to obtain the pipeline data without output stream or fill the pipeline data without input stream. The stream has two types / modes, reader and writer:

- AUDIO_STREAM_READER, e.g. [i2s]->[filter]->[raw],[i2s]->[codec-amr]->[raw]
- AUDIO_STREAM_WRITER, e.g. [raw]->[codec-mp3]->[i2s] Raw Stream configurations

Public Members

audio_stream_type_t **type**

Type of stream

int **out_rb_size**

Size of output ringbuffer

Macros

RAW_STREAM_RINGBUFFER_SIZE

RAW_STREAM_CFG_DEFAULT()

2.2.7 SPIFFS 流

SPIFFS 流从 SPIFFS 读取和写入音频数据。

应用示例

- player/pipeline_spiffs_mp3

Header File

- audio_stream/include/spiffs_stream.h

Functions

audio_element_handle_t **spiffs_stream_init** (*spiffs_stream_cfg_t *config*)

Create a handle to an Audio Element to stream data from SPIFFS to another Element or get data from other elements written to SPIFFS, depending on the configuration the stream type, either AUDIO_STREAM_READER or AUDIO_STREAM_WRITER.

Return The Audio Element handle

Parameters

- `config`: The configuration

Structures

struct spiffs_stream_cfg_t

SPIFFS Stream configuration, if any entry is zero then the configuration will be set to default values.

Public Members

audio_stream_type_t **type**

Stream type

int **buf_sz**

Audio Element Buffer size

int **out_rb_size**

Size of output ringbuffer

int **task_stack**

Task stack size

int **task_core**

Task running in core (0 or 1)

int **task_prio**

Task priority (based on freeRTOS priority)

bool **write_header**

Choose to write amrnb/armwb header in spiffs whether or not (true or false, true means choose to write amrnb header)

Macros

SPIFFS_STREAM_BUF_SIZE

SPIFFS_STREAM_TASK_STACK

SPIFFS_STREAM_TASK_CORE

SPIFFS_STREAM_TASK_PRIO

SPIFFS_STREAM_RINGBUFFER_SIZE

SPIFFS_STREAM_CFG_DEFAULT()

2.2.8 TCP 客户端流

TCP 客户端流 (TCP client stream) 通过 TCP 读取和写入服务器数据。

应用示例

- get-started/pipeline_tcp_client

Header File

- audio_stream/include/tcp_client_stream.h

Functions

audio_element_handle_t **tcp_stream_init**(*tcp_stream_cfg_t* **config*)

Initialize a TCP stream to/from an audio element This function creates a TCP stream to/from an audio element depending on the stream type configuration (e.g., AUDIO_STREAM_READER or AUDIO_STREAM_WRITER). The handle of the audio element is the returned.

Return The audio element handle

Parameters

- `config`: The configuration

Structures

struct tcp_stream_event_msg

TCP Stream massage configuration.

Public Members

void ***source**

Element handle

void ***data**

Data of input/output

int **data_len**

Data length of input/output

esp_transport_handle_t **sock_fd**

handle of socket

struct tcp_stream_cfg_t

TCP Stream configuration, if any entry is zero then the configuration will be set to default values.

Public Members

audio_stream_type_t **type**

Type of stream

int **timeout_ms**

time timeout for read/write

int **port**

TCP port>

char ***host**

TCP host>

int **task_stack**

Task stack size

int **task_core**

Task running in core (0 or 1)

int **task_prio**

Task priority (based on freeRTOS priority)

bool **ext_stack**

Allocate stack on extern ram

tcp_stream_event_handle_cb **event_handler**

TCP stream event callback

void ***event_ctx**

User context

Macros

TCP_STREAM_DEFAULT_PORT

TCP stream parameters.

TCP_STREAM_TASK_STACK

TCP_STREAM_BUF_SIZE

TCP_STREAM_TASK_PRIO

TCP_STREAM_TASK_CORE

TCP_SERVER_DEFAULT_RESPONSE_LENGTH

TCP_STREAM_CFG_DEFAULT()

Type Definitions

typedef struct *tcp_stream_event_msg* **tcp_stream_event_msg_t**

TCP Stream massage configuration.

typedef esp_err_t (***tcp_stream_event_handle_cb**) (*tcp_stream_event_msg_t* *msg, *tcp_stream_status_t* state, void *event_ctx)

Enumerations

enum tcp_stream_status_t

Values:

TCP_STREAM_STATE_NONE

TCP_STREAM_STATE_CONNECTED

2.2.9 提示音流

提示音流 (tone stream) 读取 tools/audio_tone/mk_audio_tone.py 生成的数据，只支持 AUDIO_STREAM_READER 类型。

应用示例

- player/pipeline_flash_tone

Header File

- audio_stream/include/tone_stream.h

Functions

audio_element_handle_t **tone_stream_init**(*tone_stream_cfg_t* **config*)

Create an Audio Element handle to stream data from flash to another Element, only support AUDIO_STREAM_READER type.

Return The Audio Element handle

Parameters

- config: The configuration

Structures

struct tone_stream_cfg_t

TONE Stream configurations, if any entry is zero then the configuration will be set to default values.

Public Members

audio_stream_type_t **type**

Stream type

int **buf_sz**

Audio Element Buffer size

int **out_rb_size**

Size of output ringbuffer

int **task_stack**

Task stack size

int **task_core**

Task running in core (0 or 1)

int **task_prio**

Task priority (based on freeRTOS priority)

const char ***label**

Label of tone stored in flash. The default value is `flash_tone`

bool **extern_stack**

Task stack allocate on the extern ram

bool **use_delegate**

Read tone partition with esp_delegate. If task stack is on extern ram, this MUST be TRUE

Macros

TONE_STREAM_BUF_SIZE

TONE_STREAM_TASK_STACK

TONE_STREAM_TASK_CORE

TONE_STREAM_TASK_PRIO

TONE_STREAM_RINGBUFFER_SIZE

TONE_STREAM_EXT_STACK

TONE_STREAM_USE_DELEGATE

TONE_STREAM_CFG_DEFAULT()

2.2.10 嵌入 Flash 流

嵌入 Flash 流 (flash-embedding stream) 读取 tools/audio_tone/mk_embed_flash.py 生成的数据，只支持 `AUDIO_STREAM_READER` 类型。

应用示例

- player/pipeline_embed_flash_tone

Header File

- audio_stream/include/embed_flash_stream.h

Functions

audio_element_handle_t **embed_flash_stream_init**(*embed_flash_stream_cfg_t* **config*)

Create an Audio Element handle to stream data from flash to another Element, only support AUDIO_STREAM_READER type.

Return The Audio Element handle

Parameters

- `config`: The configuration

esp_err_t **embed_flash_stream_set_context**(*audio_element_handle_t* *embed_stream*, **const** *embed_item_info_t* **context*, int *max_num*)

Set the embed flash context.

```
This function mainly provides information about embed flash data
```

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] embed_stream`: The embed flash element handle
- `[in] context`: The embed flash context
- `[in] max_num`: The number of embed flash context

Structures

struct embed_flash_stream_cfg_t

Flash-embedding stream configurations, if any entry is zero then the configuration will be set to default values.

Public Members

int **buf_sz**

Audio Element Buffer size

int **out_rb_size**

Size of output ringbuffer

int **task_stack**

Task stack size

int **task_core**

Task running in core (0 or 1)

int **task_prio**

Task priority (based on freeRTOS priority)

bool **extern_stack**

At present, task stack can only be placed on `SRAM`, so it should always be set to `false`

struct embed_item_info

Embed tone information in flash.

Public Members

const uint8_t ***address**

The corresponding address in flash

int **size**

Size of corresponding data

Macros

EMBED_FLASH_STREAM_BUF_SIZE

EMBED_FLASH_STREAM_TASK_STACK

EMBED_FLASH_STREAM_TASK_CORE

EMBED_FLASH_STREAM_TASK_PRIO

EMBED_FLASH_STREAM_RINGBUFFER_SIZE

EMBED_FLASH_STREAM_EXT_STACK

EMBED_FLASH_STREAM_CFG_DEFAULT()

Type Definitions

typedef struct *embed_item_info* **embed_item_info_t**

Embed tone information in flash.

2.2.11 语音合成流

语音合成流 (TTS stream) 获取 esp-sr 的 `esp_tts_voice` 数据，只支持 `AUDIO_STREAM_READER` 类型。

应用示例

- 读类型示例：player/pipeline_tts_stream

Header File

- audio_stream/include/tts_stream.h

Functions

audio_element_handle_t **tts_stream_init** (*tts_stream_cfg_t* **config*)

Create a handle to an Audio Element to stream data from TTS to another Element, the stream type only support AUDIO_STREAM_READER for now.

Return The Audio Element handle

Parameters

- `config`: The configuration

esp_err_t **tts_stream_set_strings** (*audio_element_handle_t* *el*, **const** char **strings*)

Set tts stream strings.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] el`: The audio element handle
- `[in] strings`: The string pointer

esp_err_t **tts_stream_set_speed** (*audio_element_handle_t* *el*, *tts_voice_speed_t* *speed*)

Setting tts stream voice speed.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] el`: The esp_audio instance
- `[in] speed`: Speed will be set. 0-5 is legal. 0 is the slowest speed.

esp_err_t **tts_stream_get_speed**(*audio_element_handle_t* *el*, *tts_voice_speed_t* **speed*)

Get tts stream voice speed.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] el`: The esp_audio instance
- `[in] speed`: Return tts stream Speed will be [0,5]

Structures

struct tts_stream_cfg_t

TTS Stream configurations, if any entry is zero then the configuration will be set to default values.

Public Members

audio_stream_type_t **type**

Stream type

int **buf_sz**

Audio Element Buffer size

int **out_rb_size**

Size of output ringbuffer

int **task_stack**

Task stack size

int **task_core**

Task running in core (0 or 1)

int **task_prio**

Task priority (based on freeRTOS priority)

bool **ext_stack**

Allocate stack on extern ram

Macros

TTS_STREAM_BUF_SIZE

TTS_STREAM_TASK_STACK

TTS_STREAM_TASK_CORE

TTS_STREAM_TASK_PRIO

TTS_STREAM_RINGBUFFER_SIZE

TTS_STREAM_CFG_DEFAULT()

Enumerations

enum tts_voice_speed_t

Values:

TTS_VOICE_SPEED_0

TTS_VOICE_SPEED_1

TTS_VOICE_SPEED_2

TTS_VOICE_SPEED_3

TTS_VOICE_SPEED_4

TTS_VOICE_SPEED_5

TTS_VOICE_SPEED_MAX

2.3 播放列表

[English]

播放列表是可以按顺序或按指定顺序播放的音频文件列表。

playlist/include/sdcard_scan.h 中的*sdcard_scan()* 函数可扫描 microSD 卡中的音频文件，并且生成播放列表。支持指定文件深度扫描和过滤文件类型，播放列表实例可以存放于多种存储介质，以下为支持的存储介质：

- *存储至 microSD 卡*
- *存储至 DRAM*
- *存储至 flash 的 NVS 分区*

- *存储至 flash 的 DATA_UNDEFINED 分区*

扫描音频文件后，可使用 `playlist_operator_handle_t` 句柄调用对应的函数来实现创建、保存、打印播放列表以及获取音频序号对应的路径等功能。目前，本文中提到的大多数存储介质支持上述功能。

有关 API 的详细信息，请参阅下文。

2.3.1 扫描 microSD 卡

`sdcard_scan()` 函数可扫描指定路径下的音频文件并生成播放列表，支持指定文件深度扫描和过滤文件类型。然后，可利用回调函数将播放列表保存到指定的存储介质。

应用示例

- player/pipeline_sdcard_mp3_control
- cli

Header File

- playlist/include/sdcard_scan.h

Functions

esp_err_t **sdcard_scan**(*sdcard_scan_cb_t cb*, **const** char *path, int *depth*, **const** char *file_extension[], int *filter_num*, void *user_data)

Scan files in SD card and use callback function to save files that meet filtering conditions.

Note example sdcard_scan(callback, "/sdcard", 5, const char *[]{"mp3", "aac"}, 2, user_data); Scan 5 levels folder in sdcard and save mp3 files and aac files.

Return

- ESP_OK success
- ESP_FAIL failed

Parameters

- `cb`: The callback function
- `path`: The path to be scanned
- `depth`: The depth of file scanning // .e.g. if you only want to save files in "/test", depth = 0. // if you want to save files in "/test/scan_test/", depth = 1
- `file_extension`: File extension of files that are supposed to be saved // .e.g. const char *[]{"mp3", "aac"}

- `filter_num`: Number of filters
- `user_data`: The data to be used by callback function

Type Definitions

typedef void (***sdcard_scan_cb_t**)(void *user_data, char *url)

2.3.2 存储播放列表

存储至 microSD 卡

播放列表可存储至 microSD 卡中，并通过 `playlist_operator_handle_t` 句柄调用相应函数来实现保存、显示播放列表等功能。

应用示例

- player/pipeline_sdcard_mp3_control
- cli

Header File

- playlist/include/sdcard_list.h

Functions

esp_err_t **sdcard_list_create**(*playlist_operator_handle_t *handle*)

Create a playlist in sdcard by list id.

Return

- ESP_OK success
- ESP_FAIL failed

Parameters

- `[out] handle`: The playlist handle from application layer

esp_err_t **sdcard_list_show**(*playlist_operator_handle_t handle*)

Show all the URLs in sdcard playlist.

Return

- ESP_OK success

- ESP_FAIL failed

Parameters

- `handle`: Playlist handle

esp_err_t **sdcard_list_next** (*playlist_operator_handle_t handle*, int *step*, char ***url_buff*)

The following URLs in sdcard playlist.

Return

- ESP_OK success
- ESP_FAIL failed

Parameters

- `handle`: Playlist handle
- `step`: The offset of URL from current URL
- `[out] url_buff`: A second rank pointer to get a address of URL

esp_err_t **sdcard_list_prev** (*playlist_operator_handle_t handle*, int *step*, char ***url_buff*)

The previous URLs in sdcard playlist.

Return

- ESP_OK success
- ESP_FAIL failed

Parameters

- `handle`: Playlist handle
- `step`: The offset of URL from current URL
- `[out] url_buff`: A second rank pointer to get a address of URL

bool **sdcard_list_exist** (*playlist_operator_handle_t handle*, **const** char **url*)

Judge whether the url exists in sdcard playlist.

Return

- true existence
- false Non-existent

Parameters

- `handle`: Playlist handle
- `url`: The url to be checked

esp_err_t **sdcard_list_reset**(*playlist_operator_handle_t handle*)

Reset sdcard playlist.

Return

- ESP_OK success
- ESP_FAIL failed

Parameters

- `handle`: Playlist handle

esp_err_t **sdcard_list_current**(*playlist_operator_handle_t handle*, char **url_buff*)

Get current URL in sdcard playlist.

Return

- ESP_OK success
- ESP_FAIL failed

Parameters

- `handle`: Playlist handle
- `[out] url_buff`: A second rank pointer to get a address of URL

esp_err_t **sdcard_list_choose**(*playlist_operator_handle_t handle*, int *url_id*, char **url_buff*)

Choose a url by url id.

Return

- ESP_OK success
- ESP_FAIL failed

Parameters

- `handle`: Playlist handle
- `url_id`: The id of url in sdcard list
- `[out] url_buff`: A second rank pointer to get a address of URL

int **sdcard_list_get_url_num**(*playlist_operator_handle_t handle*)

Get URLs number in sdcard playlist.

Return

- URLs number in sdcard playlist
- ESP_FAIL Fail to get number of urls

Parameters

- `handle`: Playlist handle

int **sdcard_list_get_url_id**(*playlist_operator_handle_t handle*)

Get current url id in the sdcard playlist.

Return

- Current url id in partition playlist
- ESP_FAIL Fail to get url id

Parameters

- `handle`: Playlist handle

esp_err_t **sdcard_list_destroy**(*playlist_operator_handle_t handle*)

Destroy sdcard playlist.

Return

- ESP_OK success
- ESP_FAIL failed

Parameters

- `handle`: Playlist handle

esp_err_t **sdcard_list_save**(*playlist_operator_handle_t handle*, **const** char ***url*)

Save URL to sdcard playlist.

Return

- ESP_OK success
- ESP_FAIL failed

Parameters

- `handle`: Playlist handle
- `url`: URL to be saved

存储至 DRAM

播放列表可存储至 DRAM 中，并通过 `playlist_operator_handle_t` 句柄调用相应函数来实现保存、显示播放列表等功能。

Header File

- playlist/include/dram_list.h

Functions

esp_err_t **`dram_list_create`**(*playlist_operator_handle_t* *_handle_)

Create a playlist in dram.

Return

- ESP_OK success
- ESP_FAIL failed

Parameters

- `[out] handle`: The playlist handle from application layer

esp_err_t **`dram_list_save`**(*playlist_operator_handle_t* *handle*, **`const`** char *_url_)

Save URL to dram playlist.

Return

- ESP_OK success
- ESP_FAIL failed

Parameters

- `handle`: Playlist handle
- `url`: URL to be saved

esp_err_t **`dram_list_next`**(*playlist_operator_handle_t* *handle*, int *step*, char **_url_buff_)

The following URLs in dram playlist.

Return

- ESP_OK success
- ESP_FAIL failed

Parameters

- `handle`: Playlist handle

- `step`: The offset of URL from current URL
- `[out] url_buff`: A second rank pointer to get a address of URL

esp_err_t **`dram_list_prev`** (*playlist_operator_handle_t handle*, int *step*, char ****url_buff*)

The previous URLs in dram playlist.

Return

- ESP_OK success
- ESP_FAIL failed

Parameters

- `handle`: Playlist handle
- `step`: The offset of URL from current URL
- `[out] url_buff`: A second rank pointer to get a address of URL

bool **`dram_list_exist`** (*playlist_operator_handle_t handle*, **`const`** char **url*)

Judge whether the url exists in dram playlist.

Return

- true existence
- false Non-existent

Parameters

- `handle`: Playlist handle
- `url`: The url to be checked

esp_err_t **`dram_list_reset`** (*playlist_operator_handle_t handle*)

Reset dram playlist.

Return

- ESP_OK success
- ESP_FAIL failed

Parameters

- `handle`: Playlist handle

esp_err_t **`dram_list_current`** (*playlist_operator_handle_t handle*, char ****url_buff*)

The current URL in current playlist.

Return

- ESP_OK success
- ESP_FAIL failed

Parameters

- `handle`: Playlist handle
- `[out] url_buff`: A second rank pointer to get a address of URL

esp_err_t **dram_list_choose** (*playlist_operator_handle_t handle*, int *url_id*, char **url_buff*)

Choose a url by url id.

Return

- ESP_OK success
- ESP_FAIL failed

Parameters

- `handle`: Playlist handle
- `url_id`: The id of url in dram list
- `[out] url_buff`: A second rank pointer to get a address of URL

int **dram_list_get_url_num** (*playlist_operator_handle_t handle*)

Get URLs number in the dram playlist.

Return

- URLs number in dram playlist
- ESP_FAIL Fail to get number of urls

Parameters

- `handle`: Playlist handle

int **dram_list_get_url_id** (*playlist_operator_handle_t handle*)

Get current url id in the dram playlist.

Return

- Current url id in dram playlist
- ESP_FAIL Fail to get url id

Parameters

- `handle`: Playlist handle

esp_err_t **dram_list_show**(*playlist_operator_handle_t handle*)

Show all the URLs in the dram playlist.

Return

- ESP_OK success
- ESP_FAIL failed

Parameters

- `handle`: Playlist handle

esp_err_t **dram_list_remove_by_url**(*playlist_operator_handle_t handle*, **const** char **url*)

Remove corrsponding url in dram list.

Return

- ESP_OK success
- ESP_FAIL failed

Parameters

- `handle`: Playlist handle
- `url`: The url to be removed

esp_err_t **dram_list_remove_by_url_id**(*playlist_operator_handle_t handle*, uint16_t *url_id*)

Remove url by id.

Return

- ESP_OK success
- ESP_FAIL failed

Parameters

- `handle`: Playlist handle
- `url_id`: The url id to be removed

esp_err_t **dram_list_destroy**(*playlist_operator_handle_t handle*)

Destroy the dram playlist.

Return

- ESP_OK success
- ESP_FAIL failed

Parameters

- `handle`: Playlist handle

存储至 flash 的 NVS 分区

播放列表可存储至 flash 的 NVS 分区 中，并通过 `playlist_operator_handle_t` 句柄调用相应函数来实现保存、显示播放列表等功能。

Header File

- playlist/include/flash_list.h

Functions

esp_err_t **`flash_list_create`** (*playlist_operator_handle_t* **handle*)

Create a playlist in nvs flash.

Return

- ESP_OK success
- ESP_FAIL failed

Parameters

- `[out] handle`: Playlist handle

esp_err_t **`flash_list_save`** (*playlist_operator_handle_t handle*, **`const`** char **url*)

Save URL to nvs flash list.

Return

- ESP_OK success
- ESP_FAIL failed

Parameters

- `handle`: Playlist handle
- `url`: URL to be saved

esp_err_t **`flash_list_show`** (*playlist_operator_handle_t handle*)

Show all the URLs in nvs flash list.

Return

- ESP_OK success
- ESP_FAIL failed

Parameters

- `handle`: Playlist handle

esp_err_t **`flash_list_next`** (*playlist_operator_handle_t handle*, int *step*, char *******url_buff*)

The following URLs in nvs flash playlist.

Return

- ESP_OK success
- ESP_FAIL failed

Parameters

- `handle`: Playlist handle
- `step`: The offset of URL from current URL
- `[out] url_buff`: A second rank pointer to get a address of URL

esp_err_t **`flash_list_prev`** (*playlist_operator_handle_t handle*, int *step*, char *******url_buff*)

The previous URLs in nvs flash playlist.

Return

- ESP_OK success
- ESP_FAIL failed

Parameters

- `handle`: Playlist handle
- `step`: The offset of URL from current URL
- `[out] url_buff`: A second rank pointer to get a address of URL

esp_err_t **`flash_list_current`** (*playlist_operator_handle_t handle*, char *******url_buff*)

The current URL in nvs flash playlist.

Return

- ESP_OK success
- ESP_FAIL failed

Parameters

- `handle`: Playlist handle
- `[out] url_buff`: A second rank pointer to get a address of URL

bool **flash_list_exist** (*playlist_operator_handle_t handle*, **const** char **url*)

Judge whether the url exists in flash playlist.

Return

- true existence
- false Non-existent

Parameters

- `handle`: Playlist handle
- `url`: The url to be checked

esp_err_t **flash_list_reset** (*playlist_operator_handle_t handle*)

Reset flash playlist.

Return

- ESP_OK success
- ESP_FAIL failed

Parameters

- `handle`: Playlist handle

esp_err_t **flash_list_choose** (*playlist_operator_handle_t handle*, int *url_id*, char ***url_buff*)

Choose a url by url id.

Return

- ESP_OK success
- ESP_FAIL failed

Parameters

- `handle`: Playlist handle
- `url_id`: The id of url in flash list
- `[out] url_buff`: A second rank pointer to get a address of URL

int **flash_list_get_url_num** (*playlist_operator_handle_t handle*)

Get URLs number in the flash playlist.

Return

- URLs number in flash playlist
- ESP_FAIL Fail to get number of urls

Parameters

- `handle`: Playlist handle

int **flash_list_get_url_id**(*playlist_operator_handle_t handle*)

Get current url id in the flash playlist.

Return

- Curernt url id in flash playlist
- ESP_FAIL Fail to get url id

Parameters

- `handle`: Playlist handle

esp_err_t **flash_list_destroy**(*playlist_operator_handle_t handle*)

Destroy the nvs flash playlist.

Return

- ESP_OK success
- ESP_FAIL failed

Parameters

- `handle`: Playlist handle

存储至 flash 的 DATA_UNDEFINED 分区

播放列表可存储至 flash 的 `DATA_UNDEFINED` 分区中（详情请参考 分区表），并通过 `playlist_operator_handle_t` 句柄调用相应函数来实现保存、显示播放列表等功能。需要先将子类型为 0x06 和 0x07 的 2 个分区添加到 flash 的分区表中。

Header File

- playlist/include/partition_list.h

Functions

esp_err_t **partition_list_create**(*playlist_operator_handle_t* *handle)

Create a playlist in flash partition by list id.

Note Please add 2 partitions to partition table whose subtype are 0x06 and 0x07 first

Return

- ESP_OK success
- ESP_FAIL failed

Parameters

- `[out] handle`: The playlist handle from application layer

esp_err_t **partition_list_save**(*playlist_operator_handle_t* *handle*, **const** char **url*)

Save URL to partition playlist.

Return

- ESP_OK success
- ESP_FAIL failed

Parameters

- `handle`: Playlist handle
- `url`: URL to be saved

esp_err_t **partition_list_next**(*playlist_operator_handle_t* *handle*, int *step*, char ***url_buff*)

The following URLs in partition playlist.

Return

- ESP_OK success
- ESP_FAIL failed

Parameters

- `handle`: Playlist handle
- `step`: The offset of URL from current URL
- `[out] url_buff`: A second rank pointer to get a address of URL

esp_err_t **partition_list_prev**(*playlist_operator_handle_t* *handle*, int *step*, char ***url_buff*)

The previous URLs in partition playlist.

Return

- ESP_OK success
- ESP_FAIL failed

Parameters

- `handle`: Playlist handle
- `step`: The offset of URL from current URL
- `[out] url_buff`: A second rank pointer to get a address of URL

bool **partition_list_exist** (*playlist_operator_handle_t handle*, **const** char **url*)

Judge whether the url exists in partition playlist.

Return

- true existence
- false Non-existent

Parameters

- `handle`: Playlist handle
- `url`: The url to be checked

esp_err_t **partition_list_reset** (*playlist_operator_handle_t handle*)

Reset partition playlist.

Return

- ESP_OK success
- ESP_FAIL failed

Parameters

- `handle`: Playlist handle

esp_err_t **partition_list_current** (*playlist_operator_handle_t handle*, char ***url_buff*)

Get current URL in the partition playlist.

Return

- ESP_OK success
- ESP_FAIL failed

Parameters

- `handle`: Playlist handle
- `[out] url_buff`: A second rank pointer to get a address of URL

esp_err_t **partition_list_choose** (*playlist_operator_handle_t handle*, int *url_id*, char ***url_buff*)

Choose a url by url id.

Return

- ESP_OK success
- ESP_FAIL failed

Parameters

- `handle`: Playlist handle
- `url_id`: The id of url in partition list
- `[out] url_buff`: A second rank pointer to get a address of URL

int **partition_list_get_url_num** (*playlist_operator_handle_t handle*)

Get URLs number in the partition playlist.

Return

- URLs number in partition playlist
- ESP_FAIL Fail to get number of urls

Parameters

- `handle`: Playlist handle

int **partition_list_get_url_id** (*playlist_operator_handle_t handle*)

Get curernt url id in the partition playlist.

Return

- Current url id in partition playlist
- ESP_FAIL Fail to get url id

Parameters

- `handle`: Playlist handle

esp_err_t **partition_list_show** (*playlist_operator_handle_t handle*)

Show all the URLs in the partition playlist.

Return

- ESP_OK success
- ESP_FAIL failed

Parameters

- `handle`: Playlist handle

esp_err_t **partition_list_destroy**(*playlist_operator_handle_t* *handle*)

Destroy the partition playlist.

Return

- ESP_OK success
- ESP_FAIL failed

Parameters

- `handle`: Playlist handle

2.3.3 播放列表管理器

播放列表管理器用于管理上述播放列表，可将多个播放列表实例添加到 `playlist_handle_t` 句柄。

Header File

- playlist/include/playlist.h

Functions

playlist_handle_t **playlist_create**(void)

Create a playlist manager handle.

Return

- playlist handle success
- NULL failed

esp_err_t **playlist_add**(*playlist_handle_t* *handle*, *playlist_operator_handle_t* *list_handle*, uint8_t *list_id*)

Create a playlist manager and add playlist handle to it.

Note The partition playlist can only be added once, or it will be overwrited by the newest partiiton playlist

Note Different lists must use different IDs, because even if they are in different handles, list_id is the only indicator that distinguishes them.

Return

- ESP_OK success
- ESP_FAIL failed

Parameters

- `handle`: Playlist manager handle
- `list_handle`: The playlist handle to be added
- `list_id`: The playlist id to be registered

esp_err_t **playlist_checkout_by_id** (*playlist_handle_t handle*, uint8_t *id*)

Playlist checkout by list id.

Return

- ESP_OK success
- ESP_FAIL failed

Parameters

- `handle`: Playlist handle
- `id`: Specified list id

int **playlist_get_list_num** (*playlist_handle_t handle*)

Get number of playlists in the handle.

Return

- success Number of playlists in handle
- failed -1

Parameters

- `handle`: Playlist handle

playlist_type_t **playlist_get_current_list_type** (*playlist_handle_t handle*)

Get current playlist type.

Return

- success Type of current playlist
- failed -1

Parameters

- `handle`: Playlist handle

int **playlist_get_current_list_id** (*playlist_handle_t handle*)

Get current playlist id.

Return

- success Current playlist id

- failed -1

Parameters

- `handle`: Playlist handle

esp_err_t **playlist_get_current_list_url** (*playlist_handle_t handle*, char **url_buff*)

Get current URL in current playlist.

Return

- ESP_OK success
- ESP_FAIL failed

Parameters

- `handle`: Playlist handle
- `[out] url_buff`: A second rank pointer to get a address of URL

int **playlist_get_current_list_url_num** (*playlist_handle_t handle*)

Get number of URLs in current playlist.

Return Number of URLS in current playlsit

Parameters

- `handle`: Playlist handle

int **playlist_get_current_list_url_id** (*playlist_handle_t handle*)

Get current url id in current playlist.

Return Current url’ s id in current playlsit

Parameters

- `handle`: Playlist handle

esp_err_t **playlist_save** (*playlist_handle_t handle*, **const** char **url*)

Save a URL to the current playlist.

Return

- ESP_OK success
- ESP_FAIL failed

Parameters

- `handle`: Playlist handle
- `url`: The URL to be saved ot sdcard

esp_err_t **playlist_next** (*playlist_handle_t handle*, int *step*, char ***url_buff*)

Next URl in current playlist.

Return

- ESP_OK success
- ESP_FAIL failed

Parameters

- `handle`: Playlist handle
- `step`: Next steps from current position
- `[out] url_buff`: A second rank pointer to get a address of URL

esp_err_t **playlist_prev** (*playlist_handle_t handle*, int *step*, char ***url_buff*)

Previous URL in current playlist.

Return

- ESP_OK success
- ESP_FAIL failed

Parameters

- `handle`: Playlist handle
- `step`: Previous steps from current position
- `[out] url_buff`: A second rank pointer to get a address of URL

esp_err_t **playlist_choose** (*playlist_handle_t handle*, int *url_id*, char ***url_buff*)

Choose a url by url id.

Return

- ESP_OK success
- ESP_FAIL failed

Parameters

- `handle`: Playlist handle
- `url_id`: The id of url in current list
- `[out] url_buff`: A second rank pointer to get a address of URL

esp_err_t **playlist_show** (*playlist_handle_t handle*)

Show URLs in current playlist.

Return

- ESP_OK success
- ESP_FAIL failed

Parameters

- `handle`: Playlist handle

esp_err_t **playlist_reset** (*playlist_handle_t handle*)

Reset current playlist.

Return

- ESP_OK success
- ESP_FAIL failed

Parameters

- `handle`: Playlist handle

esp_err_t **playlist_remove_by_url** (*playlist_handle_t handle*, **const** char **url*)

Remove corresponding url.

Return

- ESP_OK success
- ESP_FAIL failed

Parameters

- `handle`: Playlist handle
- `url`: The url to be removed

esp_err_t **playlist_remove_by_url_id** (*playlist_handle_t handle*, uint16_t *url_id*)

Remove url by url id.

Return

- ESP_OK success
- ESP_FAIL failed

Parameters

- `handle`: Playlist handle
- `url_id`: The id of url to be removed

bool **playlist_exist**(*playlist_handle_t handle*, **const** char *url)

Judge whether the url exists in current playlist.

Return

- true existence
- false Non-existent

Parameters

- `handle`: Playlist handle
- `url`: The url to be checked

esp_err_t **playlist_destroy**(*playlist_handle_t handle*)

Destroy all playlists in the handle.

Return

- ESP_OK success
- ESP_FAIL failed

Parameters

- `handle`: Playlist handle

Structures

struct playlist_operation_t

All types of Playlists' operation.

Public Members

esp_err_t (***show**)(void *playlist)

Show all the URLs in playlist

esp_err_t (***save**)(void *playlist, **const** char *url)

Save URLs to playlist

esp_err_t (***next**)(void *playlist, int step, char **url_buff)

Get next URL in playlist

esp_err_t (***prev**)(void *playlist, int step, char **url_buff)

Get previous URL in playlist

esp_err_t (***reset**)(void *playlist)

Reset the playlist

esp_err_t (*choose) (void *playlist, int url_id, char **url_buff)
Get url by url id

esp_err_t (*current) (void *playlist, char **url_buff)
Get current URL in playlist

esp_err_t (*destroy) (void *playlist)
Destroy playlist

bool (*exist) (void *playlist, const char *url)
Judge whether the url exists

int (*get_url_num) (void *playlist)
Get number of URLS in current playlist

int (*get_url_id) (void *playlist)
Get current url id in playlist

playlist_type_t type
Type of playlist

esp_err_t (*remove_by_url) (void *playlist, const char *url)
Remove the corresponding url

esp_err_t (*remove_by_id) (void *playlist, uint16_t url_id)
Remove url by id

struct playlist_operator_t
Information of playlist manager node.

Public Members

void *playlist
Specific playlist's pointer

esp_err_t (*get_operation) (*playlist_operation_t* *operation)
Function pointer to get playlists' handle

Type Definitions

typedef *playlist_operator_t* *playlist_operator_handle_t

typedef struct playlist_handle *playlist_handle_t

Enumerations

enum playlist_type_t

Type of playlist.

Values:

PLAYLIST_UNKNOWN = -1

Unknown type

PLAYLIST_SDCARD

Playlist in sdcard

PLAYLIST_FLASH

Playlist in nvs

PLAYLIST_DRAM

Playlist in ram

PLAYLIST_PARTITION

Playlist in partition

2.4 Codecs

2.4.1 AAC Decoder

Decode an audio data stream provided in AAC format.

API Reference

Header File

- esp-adf-libs/esp_codec/include/codec/aac_decoder.h

Functions

audio_element_handle_t **aac_decoder_init**(*aac_decoder_cfg_t* **config*)

Create an Audio Element handle to decode incoming AAC data.

Return The audio element handle

Parameters

- `config`: The configuration

Structures

struct aac_decoder_cfg_t

AAC Decoder configuration.

Public Members

int **out_rb_size**

Size of output ringbuffer

int **task_stack**

Task stack size

int **task_core**

CPU core number (0 or 1) where decoder task in running

int **task_prio**

Task priority (based on freeRTOS priority)

bool **stack_in_ext**

Try to allocate stack in external memory

bool **plus_enable**

Dynamically enable HE-AAC (v1 v2) decoding

Macros

AAC_DECODER_TASK_STACK_SIZE

AAC_DECODER_TASK_CORE

AAC_DECODER_TASK_PRIO

AAC_DECODER_RINGBUFFER_SIZE

DEFAULT_AAC_DECODER_CONFIG()

2.4.2 AMR Decoder and Encoder

Decode and encode an audio data stream from / to AMR format. Encoders cover both AMR-NB and AMR-WB formats.

Application Examples

Implementation of this API is demonstrated in the following examples:

- player/pipeline_play_sdcard_music
- recorder/pipeline_wav_amr_sdcard

API Reference - Decoder

Header File

- esp-adf-libs/esp_codec/include/codec/amr_decoder.h

Functions

audio_element_handle_t **amr_decoder_init**(*amr_decoder_cfg_t *config*)

Create an Audio Element handle to decode incoming AMR data.

Return The audio element handle

Parameters

- `config`: The configuration

Structures

struct amr_decoder_cfg_t

AMR Decoder configuration.

Public Members

int **out_rb_size**

Size of output ringbuffer

int **task_stack**

Task stack size

int **task_core**

CPU core number (0 or 1) where decoder task in running

int **task_prio**

Task priority (based on freeRTOS priority)

bool **stack_in_ext**

Try to allocate stack in external memory

Macros

AMR_DECODER_TASK_STACK_SIZE

AMR_DECODER_TASK_CORE

AMR_DECODER_TASK_PRIO

AMR_DECODER_RINGBUFFER_SIZE

DEFAULT_AMR_DECODER_CONFIG()

API Reference - AMR-NB Encoder

Header File

- esp-adf-libs/esp_codec/include/codec/amrnb_encoder.h

Functions

esp_err_t **amrnb_encoder_set_bitrate**(*audio_element_handle_t self*, *amrnb_encoder_bitrate_t bitrate_mode*)

Set AMRNB encoder bitrate.

Return

ESP_OK ESP_FAIL

Parameters

- `self`: Audio element handle
- `bitrate_mode`: Bitrate choose, value from amrnb_encoder_bitrate_t

audio_element_handle_t **amrnb_encoder_init**(*amrnb_encoder_cfg_t *config*)

Create an Audio Element handle to encode incoming AMRNB data.

Return The audio element handle

Parameters

- `config`: The configuration

Structures

struct amrnb_encoder_cfg_t

AMRNB Encoder configurations.

Public Members

int **out_rb_size**

Size of output ringbuffer

int **task_stack**

Task stack size

int **task_core**

Task running in core (0 or 1)

int **task_prio**

Task priority (based on freeRTOS priority)

amrnb_encoder_bitrate_t **bitrate_mode**

AMRNB Encoder bitrate choose

bool **contain_amrnb_header**

Choose to contain amrnb header in amrnb encoder whether or not (true or false, true means choose to contain amrnb header)

bool **stack_in_ext**

Try to allocate stack in external memory

Macros

AMRNB_ENCODER_TASK_STACK

AMRNB_ENCODER_TASK_CORE

AMRNB_ENCODER_TASK_PRIO

AMRNB_ENCODER_RINGBUFFER_SIZE

DEFAULT_AMRNB_ENCODER_CONFIG()

Enumerations

enum amrnb_encoder_bitrate_t

Enum of AMRNB Encoder bitrate choose.

Values:

AMRNB_ENC_BITRATE_UNKNOW = -1

Invalid mode

AMRNB_ENC_BITRATE_MR475 = 0

AMRNB_ENC_BITRATE_MR515 = 1

AMRNB_ENC_BITRATE_MR59 = 2

AMRNB_ENC_BITRATE_MR67 = 3

AMRNB_ENC_BITRATE_MR74 = 4

AMRNB_ENC_BITRATE_MR795 = 5

AMRNB_ENC_BITRATE_MR102 = 6

AMRNB_ENC_BITRATE_MR122 = 7

AMRNB_ENC_BITRATE_MRDTX = 8

AMRNB_ENC_BITRATE_N_MODES = 9

API Reference - AMR-WB Encoder

Header File

- esp-adf-libs/esp_codec/include/codec/amrwb_encoder.h

Functions

esp_err_t **amrwb_encoder_set_bitrate**(*audio_element_handle_t* *self*, *amrwb_encoder_bitrate_t* *bitrate_mode*)

Set AMRWB encoder bitrate.

Return

ESP_OK ESP_FAIL

Parameters

- `self`: Audio element handle
- `bitrate_mode`: Bitrate choose, value from amrwb_encoder_bitrate_t

audio_element_handle_t **amrwb_encoder_init** (*amrwb_encoder_cfg_t *config*)

Create an Audio Element handle to encode incoming amrwb data.

Return The audio element handle

Parameters

- `config`: The configuration

Structures

struct amrwb_encoder_cfg_t

AMRWB Encoder configurations.

Public Members

int **out_rb_size**

Size of output ringbuffer

int **task_stack**

Task stack size

int **task_core**

Task running in core (0 or 1)

int **task_prio**

Task priority (based on freeRTOS priority)

amrwb_encoder_bitrate_t **bitrate_mode**

AMRWB Encoder bitrate choose

bool **contain_amrwb_header**

Choose to contain amrwb header in amrwb encoder whether or not (true or false, true means choose to contain amrwb header)

bool **stack_in_ext**

Try to allocate stack in external memory

Macros

AMRWB_ENCODER_TASK_STACK

AMRWB_ENCODER_TASK_CORE

AMRWB_ENCODER_TASK_PRIO

AMRWB_ENCODER_RINGBUFFER_SIZE

DEFAULT_AMRWB_ENCODER_CONFIG()

Enumerations

enum amrwb_encoder_bitrate_t

Enum of AMRWB Encoder bitrate choose.

Values:

AMRWB_ENC_BITRATE_MDNONE = -1

Invalid mode

AMRWB_ENC_BITRATE_MD66 = 0

6.60kbps

AMRWB_ENC_BITRATE_MD885 = 1

8.85kbps

AMRWB_ENC_BITRATE_MD1265 = 2

12.65kbps

AMRWB_ENC_BITRATE_MD1425 = 3

14.25kbps

AMRWB_ENC_BITRATE_MD1585 = 4

15.85bps

AMRWB_ENC_BITRATE_MD1825 = 5

18.25bps

AMRWB_ENC_BITRATE_MD1985 = 6

19.85kbps

AMRWB_ENC_BITRATE_MD2305 = 7

23.05kbps

AMRWB_ENC_BITRATE_MD2385 = 8

23.85kbps>

AMRWB_ENC_BITRATE_N_MODES = 9

Invalid mode

2.4.3 FLAC Decoder

Decode an audio data stream provided in FLAC format.

API Reference

Header File

- esp-adf-libs/esp_codec/include/codec/flac_decoder.h

Functions

audio_element_handle_t **flac_decoder_init** (*flac_decoder_cfg_t* **config*)

Create an Audio Element handle to decode incoming FLAC data.

Return The audio element handle

Parameters

- `config`: The configuration

Structures

struct flac_decoder_cfg_t

FLAC Decoder configuration.

Public Members

int **out_rb_size**

Size of output ringbuffer

int **task_stack**

Task stack size

int **task_core**

CPU core number (0 or 1) where decoder task in running

int **task_prio**

Task priority (based on freeRTOS priority)

bool **stack_in_ext**

Try to allocate stack in external memory

Macros

FLAC_DECODER_TASK_STACK_SIZE

FLAC_DECODER_TASK_CORE

FLAC_DECODER_TASK_PRIO

FLAC_DECODER_RINGBUFFER_SIZE

DEFAULT_FLAC_DECODER_CONFIG()

2.4.4 MP3 Decoder

Decode an audio data stream provided in MP3 format.

Application Examples

Implementation of this API is demonstrated in the following examples:

- get-started/play_mp3_control
- player/pipeline_play_sdcard_music
- player/pipeline_play_mp3_with_dac_or_pwm

API Reference

Header File

- esp-adf-libs/esp_codec/include/codec/mp3_decoder.h

Functions

audio_element_handle_t **mp3_decoder_init**(*mp3_decoder_cfg_t* **config*)

Create an Audio Element handle to decode incoming MP3 data.

Return The audio element handle

Parameters

- `config`: The configuration

const esp_id3_info_t ***mp3_decoder_get_id3_info**(*audio_element_handle_t self*)

Get ID3 information.

Return esp_id3_info_t: success NULL: ID3 is not exist or memory aloocation failed.

Parameters

- `self`: The audio element handle

Structures

struct mp3_decoder_cfg_t

Mp3 Decoder configuration.

Public Members

int **out_rb_size**

Size of output ringbuffer

int **task_stack**

Task stack size

int **task_core**

CPU core number (0 or 1) where decoder task in running

int **task_prio**

Task priority (based on freeRTOS priority)

bool **stack_in_ext**

Try to allocate stack in external memory

bool **id3_parse_enable**

True: parse ID3. False: Don't parse ID3

Macros

MP3_DECODER_TASK_STACK_SIZE

MP3_DECODER_TASK_CORE

MP3_DECODER_TASK_PRIO

MP3_DECODER_RINGBUFFER_SIZE

DEFAULT_MP3_DECODER_CONFIG()

2.4.5 OGG Decoder

Decode an audio data stream provided in OGG format.

API Reference

Header File

- esp-adf-libs/esp_codec/include/codec/ogg_decoder.h

Functions

audio_element_handle_t **ogg_decoder_init** (*ogg_decoder_cfg_t* **config*)

Create an Audio Element handle to decode incoming OGG data.

Return The audio element handle

Parameters

- `config`: The configuration

Structures

struct ogg_decoder_cfg_t

OGG Decoder configuration.

Public Members

int **out_rb_size**

Size of output ringbuffer

int **task_stack**

Task stack size

int **task_core**

CPU core number (0 or 1) where decoder task in running

int **task_prio**

Task priority (based on freeRTOS priority)

bool **stack_in_ext**

Try to allocate stack in external memory

Macros

OGG_DECODER_TASK_STACK_SIZE

OGG_DECODER_TASK_CORE

OGG_DECODER_TASK_PRIO

OGG_DECODER_RINGBUFFER_SIZE

DEFAULT_OGG_DECODER_CONFIG()

2.4.6 OPUS Decoder

Decode an audio data stream provided in OPUS format.

API Reference

Header File

- esp-adf-libs/esp_codec/include/codec/opus_decoder.h

Functions

audio_element_handle_t **decoder_opus_init**(*opus_decoder_cfg_t *config*)

Create an Audio Element handle to decode incoming OPUS data.

Return The audio element handle

Parameters

- `config`: The configuration

Structures

struct opus_decoder_cfg_t

OPUS Decoder configuration.

Public Members

int **out_rb_size**
: Size of output ringbuffer

int **task_stack**
: Task stack size

int **task_core**
: CPU core number (0 or 1) where decoder task in running

int **task_prio**
: Task priority (based on freeRTOS priority)

bool **stack_in_ext**
: Try to allocate stack in external memory

Macros

OPUS_DECODER_TASK_STACK_SIZE

OPUS_DECODER_TASK_CORE

OPUS_DECODER_TASK_PRIO

OPUS_DECODER_RINGBUFFER_SIZE

DEFAULT_OPUS_DECODER_CONFIG()

2.4.7 WAV Decoder and Encoder

Decode and encode an audio data stream from / to WAV format.

Application Examples

Implementation of this API is demonstrated in the following examples:

- player/pipeline_play_sdcard_music
- recorder/pipeline_wav_amr_sdcard

API Reference - Decoder

Header File

- esp-adf-libs/esp_codec/include/codec/wav_decoder.h

Functions

audio_element_handle_t **wav_decoder_init**(*wav_decoder_cfg_t *config*)

Create an Audio Element handle to decode incoming WAV data.

Return The audio element handle

Parameters

- `config`: The configuration

Structures

struct wav_decoder_cfg_t

brief WAV Decoder configurations

Public Members

int **out_rb_size**

Size of output ringbuffer

int **task_stack**

Task stack size

int **task_core**

Task running in core (0 or 1)

int **task_prio**

Task priority (based on freeRTOS priority)

bool **stack_in_ext**

Try to allocate stack in external memory

Macros

WAV_DECODER_TASK_STACK

WAV_DECODER_TASK_CORE

WAV_DECODER_TASK_PRIO

WAV_DECODER_RINGBUFFER_SIZE

DEFAULT_WAV_DECODER_CONFIG()

API Reference - Encoder

Header File

- esp-adf-libs/esp_codec/include/codec/wav_encoder.h

Functions

audio_element_handle_t **wav_encoder_init**(*wav_encoder_cfg_t* **config*)

Create a handle to an Audio Element to encode incoming data using WAV format.

Return The audio element handle

Parameters

- `config`: The configuration

Structures

struct wav_encoder_cfg_t

WAV Encoder configurations.

Public Members

int **out_rb_size**

Size of output ringbuffer

int **task_stack**

Task stack size

int **task_core**

Task running in core (0 or 1)

int **task_prio**

Task priority (based on freeRTOS priority)

bool **stack_in_ext**

Try to allocate stack in external memory

Macros

WAV_ENCODER_TASK_STACK

WAV_ENCODER_TASK_CORE

WAV_ENCODER_TASK_PRIO

WAV_ENCODER_RINGBUFFER_SIZE

DEFAULT_WAV_ENCODER_CONFIG()

2.5 Audio Processing

There are couple of options implemented in the ESP-ADF to modify contents of an audio stream:

- Combine contents of two audio streams using *Downmix*
- Apply ten band *Equalizer*
- Change audio sampling frequency and convert between single and two channel with *Resample Filter*
- Modify pitch and speed of the stream using *Sonic*

Please refer to description of respective APIs below.

2.5.1 Downmix

This API is intended for mixing of two audio files (streams), defined as the base audio file and the newcome audio file, into one output audio file.

The newcome audio file will be downmixed into the base audio file with individual gains applied to each file.

The number of channel(s) of the output audio file will be the same with that of the base audio file. The number of channel(s) of the newcome audio file will also be changed to the same with the base audio file, if it is different from that of the base audio file.

The downmix process has 3 states:

- Bypass Downmixing –Only the base audio file will be processed;

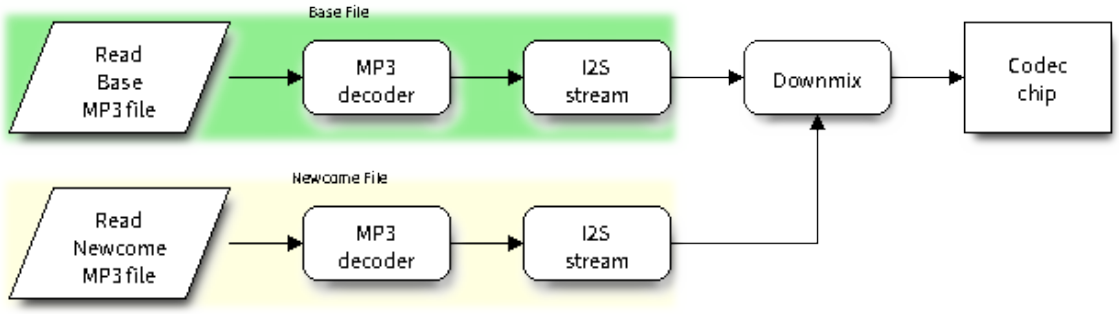


图 4: Illustration of Downmixing Process

- Switch on Downmixing –The base audio file and the target audio file will first enter the transition period, during which the gains of these two files will be changed from the original level to the target level; then enter the stable period, sharing a same target gain;
- Switch off Downmixing –The base audio file and the target audio file will first enter the transition period, during which the gains of these two files will be changed back to their original levels; then enter the stable period, with their original gains, respectively. After that, the downmix process enters the bypass state.

Note that, the sample rates of the base audio file and the newcome audio file must be the same, otherwise an error occurs.

Application Example

Implementation of this API is demonstrated in advanced_examples/downmix_pipeline example.

API Reference

Header File

- esp-adf-libs/esp_codec/include/codec/downmix.h

Functions

void **downmix_set_input_rb_timeout**(*audio_element_handle_t self*, int *ticks_to_wait*, int *index*)

Sets the downmix timeout.

Parameters

- `self`: audio element handle
- `ticks_to_wait`: input ringbuffer timeout
- `index`: The index of multi input ringbuffer.

void **downmix_set_input_rb**(*audio_element_handle_t self*, *ringbuf_handle_t rb*, int *index*)

Sets the downmix input ringbuffer. refer to `ringbuf.h`

Parameters

- `self`: audio element handle
- `rb`: handle of ringbuffer
- `index`: The index of multi input ringbuffer.

esp_err_t **downmix_set_output_type**(*audio_element_handle_t self*, esp_downmix_output_type_t *output_type*)

Passes number of channels for output stream. Only supported mono and dual.

Return ESP_OK ESP_FAIL

Parameters

- `self`: audio element handle
- `output_type`: down-mixer output type.

esp_err_t **downmix_set_work_mode**(*audio_element_handle_t self*, esp_downmix_work_mode_t *mode*)

Sets BYPASS, ON or OFF status of down-mixer.

Return ESP_OK ESP_FAIL

Parameters

- `self`: audio element handle
- `mode`: down-mixer work mode.

esp_err_t **downmix_set_out_ctx_info**(*audio_element_handle_t self*, esp_downmix_out_ctx_type_t *out_ctx*)

Passes content of per channel output stream by down-mixer.

Return ESP_OK ESP_FAIL

Parameters

- `self`: audio element handle
- `out_ctx`: content of output stream.

esp_err_t **downmix_set_source_stream_info**(*audio_element_handle_t self*, int *rate*, int *ch*, int *index*)

Sets the sample rate and the number of channels of input stream to be processed.

Return ESP_OK ESP_FAIL

Parameters

- `self`: audio element handle
- `rate`: sample rate of the input stream
- `ch`: number of channel(s) of the input stream
- `index`: The index of input stream. The index must be in [0, SOURCE_NUM_MAX - 1] range.

esp_err_t **downmix_set_gain_info** (*audio_element_handle_t self*, float **gain*, int *index*)

Sets the audio gain to be processed.

Return ESP_OK ESP_FAIL

Parameters

- `self`: audio element handle
- `gain`: the reset value of `gain`. The `gain` is an array of two elements.
- `index`: The index of input stream. The index must be in [0, SOURCE_NUM_MAX - 1] range.

esp_err_t **downmix_set_transit_time_info** (*audio_element_handle_t self*, int *transit_time*, int *index*)

Sets the audio `transit_time` to be processed.

Return ESP_OK ESP_FAIL

Parameters

- `self`: audio element handle
- `transit_time`: the reset value of `transit_time`
- `index`: The index of input stream. The index must be in [0, SOURCE_NUM_MAX - 1] range

esp_err_t **source_info_init** (*audio_element_handle_t self*, esp_downmix_input_info_t **source_num*)

Initializes information of the source streams for downmixing.

Return ESP_OK ESP_FAIL

Parameters

- `self`: audio element handle
- `source_num`: The information array of source streams

audio_element_handle_t **downmix_init** (*downmix_cfg_t *config*)

Initializes the Audio Element handle for downmixing.

Return The initialized Audio Element handle

Parameters

- `config`: the configuration

Structures

struct downmix_cfg_t

Downmix configuration.

Public Members

esp_downmix_info_t **downmix_info**

Downmix information

int **max_sample**

The number of samples per downmix processing

int **out_rb_size**

Size of ring buffer

int **task_stack**

Size of task stack

int **task_core**

Task running in core···

int **task_prio**

Task priority (based on the FreeRTOS priority)

bool **stack_in_ext**

Try to allocate stack in external memory

Macros

DOWNMIX_TASK_STACK

DOWNMIX_TASK_CORE

DOWNMIX_TASK_PRIO

DOWNMIX_RINGBUFFER_SIZE

DM_BUF_SIZE

DEFAULT_DOWNMIX_CONFIG()

2.5.2 Equalizer

Provided in this API equalizer supports:

- fixed number of ten (10) bands;
- four sample rates: 11025 Hz, 22050 Hz, 44100 Hz and 48000 Hz.

The center frequencies of bands are shown in table below.

Band Index	0	1	2	3	4	5	6	7	8	9
Frequency	31 Hz	62 Hz	125 Hz	250 Hz	500 Hz	1 kHz	2 kHz	4 kHz	8 kHz	16 kHz

Default gain of each band is -13 dB. To set the gains of all bands use structure `equalizer_cfg`. To set the gain of individual band use function `equalizer_set_gain_info()`.

Application Example

Implementation of this API is demonstrated in the audio_processing/pipeline_equalizer example.

API Reference

Header File

- esp-adf-libs/esp_codec/include/codec/equalizer.h

Functions

esp_err_t **equalizer_set_info**(*audio_element_handle_t self*, int *rate*, int *ch*)

Set the audio sample rate and the number of channels to be processed by the equalizer.

Return ESP_OK ESP_FAIL

Parameters

- `self`: Audio element handle
- `rate`: Audio sample rate
- `ch`: Audio channel

esp_err_t **equalizer_set_gain_info**(*audio_element_handle_t self*, int *index*, int *value_gain*, bool *is_channels_gain_equal*)

Set the audio gain to be processed by the equalizer.

Return ESP_OK ESP_FAIL

Parameters

- `self`: Audio element handle
- `index`: the position of center frequencies of equalizer
- `value_gain`: the value of audio gain which in `index`
- `is_channels_gain_equal`: if Number of audio channel is equal 2, the value of audio gains which two channels are equal by checking `is_channels_gain_equal`. if `is_channels_gain_equal` is `true`,it means equal,otherwise unequal.

audio_element_handle_t **equalizer_init** (*equalizer_cfg_t* **config*)

Create an Audio Element handle that equalizes incoming data.

Return The created audio element handle

Parameters

- `config`: The configuration

Structures

struct equalizer_cfg

Equalizer Configuration.

Public Members

int **samplerate**

Audio sample rate (in Hz)

int **channel**

Number of audio channels (Mono=1, Dual=2)

int ***set_gain**

Equalizer gain

int **out_rb_size**

Size of output ring buffer

int **task_stack**

Task stack size

int **task_core**

Task running in core···

int **task_prio**

Task priority

bool **stack_in_ext**

Try to allocate stack in external memory

Macros

EQUALIZER_TASK_STACK

EQUALIZER_TASK_CORE

EQUALIZER_TASK_PRIO

EQUALIZER_RINGBUFFER_SIZE

DEFAULT_EQUALIZER_CONFIG()

Type Definitions

typedef struct *equalizer_cfg* **equalizer_cfg_t**

Equalizer Configuration.

2.5.3 Resample Filter

The Resample Filter is an *Audio Element* designed to downsample or upsample the incoming data stream as well as to convert the data between stereo and mono.

Application Example

Implementation of this API is demonstrated in the following examples:

- audio_processing/pipeline_resample
- audio_processing/pipeline_spiffs_amr_resample
- get-started/play_mp3_control

API Reference

Header File

- esp-adf-libs/esp_codec/include/codec/filter_resample.h

Functions

esp_err_t **rsp_filter_set_src_info**(*audio_element_handle_t self*, int *src_rate*, int *src_ch*)

Set the source audio sample rate and the number of channels to be processed by the resample. If need change bits or don't consure source data infomation, please use rsp_filter_change_src_info ti instead this function. In future, the function will be forbidden.

Return ESP_OK ESP_FAIL

Parameters

- `self`: Audio element handle
- `src_rate`: The sample rate of stream data
- `src_ch`: The number channels of stream data

esp_err_t **rsp_filter_change_src_info**(*audio_element_handle_t self*, int *src_rate*, int *src_ch*, int *src_bit*)

Set the source audio sample rate and the number of channels, bit per sample to be processed by the resample.

Return ESP_OK ESP_FAIL

Parameters

- `self`: Audio element handle
- `src_rate`: The sample rate of stream data
- `src_ch`: The number channels of stream data
- `src_bit`: The bit per sample of stream data

audio_element_handle_t **rsp_filter_init**(*rsp_filter_cfg_t *config*)

Create an Audio Element handle to resample incoming data.

Depending on configuration, there are upsampling, downsampling, as well as converting data between mono and dual.

- If the esp_resample_mode_t is `RESAMPLE_DECODE_MODE`, `src_rate` and `src_ch` will be fetched from `audio_element_getinfo`.
- If the esp_resample_mode_t is `RESAMPLE_ENCODE_MODE`, `src_rate`, `src_ch`, `dest_rate` and `dest_ch` must be configured.

Return The audio element handler

Parameters

- `config`: The configuration

Structures

struct rsp_filter_cfg_t

Resample Filter Configuration.

Public Members

int **src_rate**

The sampling rate of the source PCM file (in Hz)

int **src_ch**

The number of channel(s) of the source PCM file (Mono=1, Dual=2)

int **dest_rate**

The sampling rate of the destination PCM file (in Hz)

int **dest_bits**

The bit for sample of the destination PCM data. Currently, supported bit width :16 bits.

int **dest_ch**

The number of channel(s) of the destination PCM file (Mono=1, Dual=2)

int **src_bits**

The bit for sample of the source PCM data. Currently, supported bit width :8bits 16 bits 24bits 32bits.

esp_resample_mode_t **mode**

The resampling mode (the encoding mode or the decoding mode). For decoding mode, input PCM length is constant; for encoding mode, output PCM length is constant.

int **max_indata_bytes**

The maximum buffer size of the input PCM (in bytes)

int **out_len_bytes**

The buffer length of the output stream data. This parameter must be configured in encoding mode.

esp_resample_type_t **type**

The resampling type (Automatic, Upsampling and Downsampling)

int **complexity**

Indicates the complexity of the resampling. This parameter is only valid when a FIR filter is used. Range:[1, 5]; 1 indicates the lowest complexity, which means the accuracy is the lowest and the speed is the fastest; Meanwhile, 5 indicates the highest complexity, which means the accuracy is the highest and the speed is the slowest.If user set `complexity` less than 1, `complexity` can be set 1. If user set `complexity` more than 5, `complexity` can be set 5.

int **down_ch_idx**

Indicates the channel that is selected (the right channel or the left channel). This parameter is only valid when

the complexity parameter is set to 0 and the number of channel(s) of the input file has changed from dual to mono.

esp_rsp_prefer_type_t **prefer_flag**
: The select flag about lesser CPU usage or lower INRAM usage, refer to esp_resample.h

int **out_rb_size**
: Output ringbuffer size

int **task_stack**
: Task stack size

int **task_core**
: Task running on core

int **task_prio**
: Task priority

bool **stack_in_ext**
: Try to allocate stack in external memory

Macros

RSP_FILTER_BUFFER_BYTE

RSP_FILTER_TASK_STACK

RSP_FILTER_TASK_CORE

RSP_FILTER_TASK_PRIO

RSP_FILTER_RINGBUFFER_SIZE

DEFAULT_RESAMPLE_FILTER_CONFIG()

2.5.4 Sonic

The Sonic component acts as a multidimensional filter that lets you adjust audio parameters of a WAV stream. This functionality may be useful to e.g. increase playback speed of an audio recording by a user selectable rate.

The following parameters can be adjusted:

- speed
- pitch
- interpolation type

The adjustments of the first two parameters are represented by *float* values that provide the rate of adjustment. For example, to increase the speed of an audio sample by 2 times, call `sonic_set_pitch_and_speed_info(el, 1.0, 2.0)`. To keep the speed as it is, call `sonic_set_pitch_and_speed_info(el, 1.0, 1.0)`.

For the *interpolation type* you may select either faster but less accurate linear interpolation, or slower but more accurate FIR interpolation.

Application Example

Implementation of this API is demonstrated in audio_processing/pipeline_sonic example.

API Reference

Header File

- esp-adf-libs/esp_codec/include/codec/audio_sonic.h

Functions

esp_err_t **sonic_set_info** (*audio_element_handle_t* *self*, int *rate*, int *ch*)

Sets the audio sample rate and the number of channels to be processed by the sonic.

Return ESP_OK ESP_FAIL

Parameters

- `self`: Audio element handle
- `rate`: The sample rate of stream data
- `ch`: The number channels of stream data

esp_err_t **sonic_set_pitch_and_speed_info** (*audio_element_handle_t* *self*, float *pitch*, float *speed*)

Sets the audio pitch and speed to be processed by the sonic.

Return ESP_OK ESP_FAIL

Parameters

- `self`: Audio element handle
- `pitch`: Scale factor of pitch of audio file. 0 means the original pitch. The range is [0.2 4.0].
- `speed`: Scale factor of speed of audio file. 0 means the original speed. The range is [0.1 8.0].

audio_element_handle_t **sonic_init** (*sonic_cfg_t* **config*)

Creates an Audio Element handle for sonic.

Return The sonic audio element handle

Parameters

- `config`: The sonic configuration

Structures

struct sonic_info_t

Information on audio file and configuration parameters required by sonic to process the file.

Public Members

int **samplerate**

Audio file sample rate (in Hz)

int **channel**

Number of audio file channels (Mono=1, Dual=2)

int **resample_linear_interpolate**

Flag of using simple linear interpolation. 1 indicates using simple linear interpolation. 0 indicates not using simple linear interpolation.

float **pitch**

Scale factor of pitch of audio file. If the value of ‘pitch’ is 0.3, the pitch of audio file processed by sonic islower than the original. If the value of ‘pitch’ is 1.3, the pitch of audio file processed by sonic is 30% higher than the original.

float **speed**

Scale factor of speed of audio file. If the value of ‘speed’ is 0.3, the speed of audio file processed by sonic is 70% slower than the original. If the value of ‘speed’ is 1.3, the speed of audio file processed by sonic is 30% faster than the original.

struct sonic_cfg_t

Sonic configuration.

Public Members

sonic_info_t **sonic_info**

Information of sonic

int **out_rb_size**

Size of output ring buffer

int **task_stack**

Task stack size

int **task_core**

Task running in core

int **task_prio**
: Task priority

bool **stack_in_ext**
: Try to allocate stack in external memory

Macros

SONIC_SET_VALUE_FOR_INITIALIZATION

SONIC_TASK_STACK

SONIC_TASK_CORE

SONIC_TASK_PRIO

SONIC_RINGBUFFER_SIZE

DEFAULT_SONIC_CONFIG()

2.6 服务

[English]

服务 (service) 框架是具体产品功能在软件层面上的实现，如输入按键、网络配置管理、电池检测等功能。每个服务也是对硬件的抽象，如输入按键服务支持 ADC 类型和 GPIO 类型的的按键。服务可以复用在不同产品上，高级 API 和事件可以让用户简单便捷的开发产品。

更多信息请参考以下文档。

2.6.1 蓝牙服务

[English]

蓝牙服务 (Bluetooth service) 专门用于与蓝牙设备进行交互，支持以下协议：

- 免提规范 (Hands-Free Profile, HFP)：通过免提设备远程控制手机以及二者间的语音连接。
- 高级音频分发框架 (Advanced Audio Distribution Profile, A2DP)：通过蓝牙连接播放多媒体音频。
- 音视频远程控制规范 (Audio Video Remote Control Profile, AVRCP)：与 A2DP 同时使用，用于远程控制耳机、汽车音响系统或扬声器等设备。

应用示例

以下示例展示了该 API 的实现方式。

- player/pipeline_bt_sink

Header File

- bluetooth_service/include/bluetooth_service.h

Functions

esp_err_t **bluetooth_service_start**(*bluetooth_service_cfg_t* *config)

Initialize and start the Bluetooth service. This function can only be called for one time, and `bluetooth_service_destroy` must be called after use.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `config`: The configuration

audio_element_handle_t **bluetooth_service_create_stream**()

Create Bluetooth stream, it is valid when Bluetooth service has started. The returned audio stream compatible with existing audio streams and can be used with the Audio Pipeline.

Return The Audio Element handle

esp_periph_handle_t **bluetooth_service_create_periph**()

Create Bluetooth peripheral, it is valid when Bluetooth service has started. The returned bluetooth peripheral compatible with existing peripherals and can be used with the ESP Peripherals.

Return The Peripheral handle

esp_err_t **periph_bluetooth_play**(*esp_periph_handle_t* periph)

Send the AVRC passthrough command (PLAY) to the Bluetooth device.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] periph`: The periph

esp_err_t **periph_bluetooth_pause**(*esp_periph_handle_t periph*)

Send the AVRC passthrough command (PAUSE) to the Bluetooth device.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] periph`: The periph

esp_err_t **periph_bluetooth_stop**(*esp_periph_handle_t periph*)

Send the AVRC passthrough command (STOP) to the Bluetooth device.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] periph`: The periph

esp_err_t **periph_bluetooth_next**(*esp_periph_handle_t periph*)

Send the AVRC passthrough command (NEXT) to the Bluetooth device.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] periph`: The periph

esp_err_t **periph_bluetooth_prev**(*esp_periph_handle_t periph*)

Send the AVRC passthrough command (PREV) to the Bluetooth device.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] periph`: The periph

esp_err_t **periph_bluetooth_rewind**(*esp_periph_handle_t periph*)

Send the AVRC passthrough command (REWIND) to the Bluetooth device.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] periph`: The periph

esp_err_t **periph_bluetooth_fast_forward**(*esp_periph_handle_t periph*)

Send the AVRC passthrough command (FAST FORWARD) to the Bluetooth device.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] periph`: The periph

esp_err_t **periph_bluetooth_discover**(*esp_periph_handle_t periph*)

Start device discovery.

Return

- ESP_OK : Succeed
- ESP_ERR_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_ERR_INVALID_ARG: if invalid parameters are provided
- ESP_FAIL: others

Parameters

- `[in] periph`: The periph

esp_err_t **periph_bluetooth_cancel_discover**(*esp_periph_handle_t periph*)

Cancel device discovery.

Return

- ESP_OK : Succeed
- ESP_ERR_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

Parameters

- `[in] periph`: The periph

esp_err_t **periph_bluetooth_connect**(*esp_periph_handle_t periph*, *bluetooth_addr_t remote_bda*)

Connect remote Device.

Return

- ESP_OK : Succeed
- ESP_ERR_INVALID_STATE: if bluetooth stack is not yet enabled
- ESP_FAIL: others

Parameters

- `[in] periph`: The periph
- `[in] remote_bda`: remote Bluetooth device address

esp_err_t **bluetooth_service_destroy**()

Destroy and cleanup bluetooth service, this function must be called after destroying the Bluetoth Stream and Bluetooth Peripheral created by `bluetooth_service_create_stream` and `bluetooth_service_create_periph`

Return

- ESP_OK
- ESP_FAIL

int **periph_bluetooth_get_a2dp_sample_rate**()

Get a2dp sample rate.

Return

- sample rate

Structures

struct bluetooth_service_user_cb_t

brief Bluetooth service user callback

Public Members

esp_a2d_cb_t **user_a2d_cb**

callback for a2dp

esp_a2d_sink_data_cb_t **user_a2d_sink_data_cb**

callback for a2dp sink data

esp_a2d_source_data_cb_t **user_a2d_source_data_cb**

callback for a2dp source data

esp_avrc_ct_cb_t **user_avrc_ct_cb**

callback for avrc ct

struct bluetooth_service_cfg_t

brief Bluetooth service configuration

Public Members

const char ***device_name**

Bluetooth local device name

const char ***remote_name**

Bluetooth remote device name

bluetooth_service_mode_t **mode**

Bluetooth working mode

bluetooth_service_user_cb_t **user_callback**

Bluetooth user callback

Macros

ESP_A2DP_SAMPLE_RATE

BLUETOOTH_ADDR_LEN

brief Bluetooth address length

Type Definitions

typedef uint8_t **bluetooth_addr_t**[**BLUETOOTH_ADDR_LEN**]

brief Bluetooth device address

Enumerations

enum bluetooth_service_mode_t

brief Bluetooth service working mode

Values:

BLUETOOTH_A2DP_SINK

A2DP Bluetooth sink audio, ESP32 will receive audio data from other bluetooth devices

BLUETOOTH_A2DP_SOURCE

A2DP Bluetooth source audio, ESP32 can send audio data to other bluetooth devices

Header File

- bluetooth_service/include/bt_keycontrol.h

Header File

- bluetooth_service/include/hfp_stream.h

Header File

- bluetooth_service/include/a2dp_stream.h

2.6.2 输入按键服务

[English]

输入按键服务 (input key service) 将 GPIO 输入中断和 ADC 按键功能以事件的形式提供给用户。事件定义的音频产品常用按键功能参见*input_key_user_id_t*。

应用示例

以下示例展示了该 API 的实现方式。

- checks/check_board_buttons
- player/pipeline_sdcard_mp3_control
- protocols/voip

Header File

- input_key_service/include/input_key_service.h

Functions

periph_service_handle_t **input_key_service_create** (*input_key_service_cfg_t* **input_key_config*)

Initialize and start the input key service.

Return NULL failed others input key service handle

Parameters

- `input_key_config`: Configuration of input key service

periph_service_state_t **get_input_key_service_state** (*periph_service_handle_t* *input_handle*)

Get the state of input key service.

Return state of input key service

Parameters

- `input_handle`: The handle of input key service

esp_err_t **input_key_service_add_key** (*periph_service_handle_t* *input_key_handle*, *input_key_service_info_t* **input_key_info*, int *add_key_num*)

Add input key's information to service list.

Return ESP_OK success ESP_FAIL failed

Parameters

- `input_key_handle`: handle of service
- `input_key_info`: input key's information
- `add_key_num`: number of keys

Structures

struct input_key_service_info_t

input key's infomation

Public Members

esp_periph_id_t **type**
ID of peripherals

int **user_id**
The key’s user id

int **act_id**
The key’s action id

struct input_key_service_cfg_t
input key’s configuration

Public Members

periph_service_config_t **based_cfg**
Peripheral service configuration

esp_periph_set_handle_t **handle**
Peripheral set handle

Macros

INPUT_KEY_SERVICE_TASK_STACK_SIZE

INPUT_KEY_SERVICE_TASK_PRIORITY

INPUT_KEY_SERVICE_TASK_ON_CORE

INPUT_KEY_SERVICE_DEFAULT_CONFIG()

Enumerations

enum input_key_service_action_id_t
input key action id

Values:

INPUT_KEY_SERVICE_ACTION_UNKNOWN = 0
unknown action id

INPUT_KEY_SERVICE_ACTION_CLICK
click action id

INPUT_KEY_SERVICE_ACTION_CLICK_RELEASE
click release action id

INPUT_KEY_SERVICE_ACTION_PRESS
long press action id

INPUT_KEY_SERVICE_ACTION_PRESS_RELEASE
long press release id

Header File

- input_key_service/include/input_key_com_user_id.h

Enumerations

enum input_key_user_id_t
input key user user-defined id

Values:

INPUT_KEY_USER_ID_UNKNOWN = -1
unknown user id

INPUT_KEY_USER_ID_REC = 0x01
user id for recording

INPUT_KEY_USER_ID_SET = 0x02
user id for settings

INPUT_KEY_USER_ID_PLAY = 0x03
user id for playing

INPUT_KEY_USER_ID_MODE = 0x04
user id for mode

INPUT_KEY_USER_ID_VOLDOWN = 0x05
user id for volume down

INPUT_KEY_USER_ID_VOLUP = 0x06
user id for volume up

INPUT_KEY_USER_ID_MUTE = 0x07
user id for mute

INPUT_KEY_USER_ID_CAPTURE = 0x08
user id for capture photo

INPUT_KEY_USER_ID_MSG = 0x09
user id for message

INPUT_KEY_USER_ID_BATTERY_CHARGING = 0x0A
user id for battery charging

INPUT_KEY_USER_ID_WAKEUP = 0x0B
user id for GPIO wakeup

INPUT_KEY_USER_ID_COLOR = 0x0C
user id for color

INPUT_KEY_USER_ID_MAX = 0x101

2.6.3 Wi-Fi Service

[English]

Wi-Fi 服务 (Wi-Fi service) 提供了网络配置和网络管理的功能。配置网络支持 SmartConfig、BluFi 和 AirKiss。

应用示例

以下示例展示了该 API 的实现方式。

- dueros

Header File

- wifi_service/include/wifi_service.h

Functions

periph_service_handle_t **wifi_service_create** (*wifi_service_config_t* **config*)

esp_err_t **wifi_service_destroy** (*periph_service_handle_t handle*)

esp_err_t **wifi_service_register_setting_handle** (*periph_service_handle_t handle*, *esp_wifi_setting_handle_t setting*, int **out_index*)

esp_err_t **wifi_service_setting_start** (*periph_service_handle_t handle*, int *index*)

esp_err_t **wifi_service_update_sta_info** (*periph_service_handle_t handle*, wifi_config_t **wifi_conf*)

esp_err_t **wifi_service_setting_stop** (*periph_service_handle_t handle*, int *index*)

esp_err_t **wifi_service_connect** (*periph_service_handle_t handle*)

esp_err_t **wifi_service_disconnect** (*periph_service_handle_t handle*)

esp_err_t **wifi_service_set_sta_info** (*periph_service_handle_t handle*, wifi_config_t **info*)

periph_service_state_t **wifi_service_state_get** (*periph_service_handle_t handle*)

wifi_service_disconnect_reason_t **wifi_service_disconnect_reason_get** (*periph_service_handle_t handle*)

esp_err_t **wifi_service_erase_ssid_manager_info**(*periph_service_handle_t handle*)

esp_err_t **wifi_service_get_last_ssid_cfg**(*periph_service_handle_t handle*, wifi_config_t **wifi_cfg*)

Structures

struct wifi_service_config_t

WiFi service configurations.

Public Members

int **task_stack**

>0 Service task stack; =0 with out task created

int **task_prio**

Service task priority (based on freeRTOS priority)

int **task_core**

Service task running in core (0 or 1)

bool **extern_stack**

Task stack allocate on the extern ram

periph_service_cb **evt_cb**

Service callback function

void ***cb_ctx**

Callback context

char ***user_data**

User data

int **setting_timeout_s**

Timeout of setting WiFi

int **max_retry_time**

Maximum times of reconnection

int **max_prov_retry_time**

Maximum times of reconnection after wifi provision

uint8_t **max_ssid_num**

Maximum ssid that can be stored

Macros

WIFI_SERVICE_DEFAULT_CONFIG()

Enumerations

enum wifi_service_event_t

WiFi STA service status.

Values:

WIFI_SERV_EVENT_UNKNOWN

WIFI_SERV_EVENT_CONNECTING

WIFI_SERV_EVENT_CONNECTED

WIFI_SERV_EVENT_DISCONNECTED

WIFI_SERV_EVENT_SETTING_TIMEOUT

WIFI_SERV_EVENT_SETTING_FAILED

WIFI_SERV_EVENT_SETTING_FINISHED

enum wifi_service_disconnect_reason_t

WiFi STA disconnection reasons.

Values:

WIFI_SERV_STA_UNKNOWN

WIFI_SERV_STA_COM_ERROR

WIFI_SERV_STA_AUTH_ERROR

WIFI_SERV_STA_AP_NOT_FOUND

WIFI_SERV_STA_BY_USER

WIFI_SERV_STA_SET_INFO

Header File

- wifi_service/include/esp_wifi_setting.h

Functions

esp_wifi_setting_handle_t **esp_wifi_setting_create**(**const** char **tag*)

brief Create wifi setting handle instance

Return

- NULL, Fail
- Others, Success

Parameters

- `tag`: Tag of the wifi setting handle

esp_err_t **esp_wifi_setting_destroy**(*esp_wifi_setting_handle_t handle*)

brief Destroy wifi setting handle instance

Return

- ESP_OK
- ESP_FAIL

Parameters

- `handle`: The wifi setting handle instance

esp_err_t **esp_wifi_setting_register_function**(*esp_wifi_setting_handle_t handle*, *wifi_setting_func start*, *wifi_setting_func stop*, *wifi_setting_teardown_func teardown*)

Register the wifi setting execute functions.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `handle`: The wifi setting handle instance
- `start`: The start wifi setting
- `stop`: The stop
- `teardown`: The destroy

esp_err_t **esp_wifi_setting_register_notify_handle**(*esp_wifi_setting_handle_t handle*, void **on_handle*)

Register the notify execute handle.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `handle`: The peripheral handle
- `on_handle`: The notify execute handle

esp_err_t **esp_wifi_setting_info_notify**(*esp_wifi_setting_handle_t handle*, wifi_config_t **info*)

Call this to notify the `wifi_config_t` to `on_handle`

Return

- ESP_OK
- ESP_FAIL

Parameters

- `handle`: The wifi setting handle instance
- `info`: The `wifi_config_t`

esp_err_t **esp_wifi_setting_set_data**(*esp_wifi_setting_handle_t handle*, void **data*)

Set the user data.

Note Make sure the `data` lifetime is sufficient, this function does not copy all data, it only stores the data pointer

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] handle`: The wifi setting handle instance
- `data`: The user data

void ***esp_wifi_setting_get_data**(*esp_wifi_setting_handle_t handle*)

Get the user data stored on `handle`

Return user data pointer

Parameters

- `[in] handle`: The wifi setting handle instance

esp_err_t **esp_wifi_setting_start**(*esp_wifi_setting_handle_t handle*)

Call this to execute `start` function of wifi setting instance.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `handle`: The wifi setting handle instance

esp_err_t **esp_wifi_setting_stop**(*esp_wifi_setting_handle_t handle*)

Call this to execute `stop` function of wifi setting instance.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `handle`: The wifi setting handle instance

esp_err_t **esp_wifi_setting_teardown**(*esp_wifi_setting_handle_t handle*, wifi_config_t *_info_)

Call this to execute `teardown` function of wifi setting instance.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `handle`: The wifi setting handle instance
- `info`: The `wifi_config_t`

Type Definitions

typedef struct esp_wifi_setting ***esp_wifi_setting_handle_t**

typedef esp_err_t (***wifi_setting_func**)(*esp_wifi_setting_handle_t* handle)

typedef esp_err_t (***wifi_setting_teardown_func**)(*esp_wifi_setting_handle_t* handle, wifi_config_t *info)

Header File

- wifi_service/include/smart_config.h

Functions

esp_wifi_setting_handle_t **smart_config_create**(*smart_config_info_t* **info*)

brief Create smartconfig setting handle instance

Return

- NULL, Fail
- Others, Success

Parameters

- `info`: Configuration of the smartconfig

Structures

struct smart_config_info_t

esp smartconfig configuration

Public Members

smartconfig_type_t **type**

Type of smartconfig

Macros

SMART_CONFIG_INFO_DEFAULT()

Header File

- wifi_service/include/blufi_config.h

Functions

esp_wifi_setting_handle_t **blufi_config_create** (void **info*)

Create blufi setting handle instance.

Return

- NULL, Fail
- Others, Success

Parameters

- `[in] info`: A pointer to void

esp_err_t **blufi_set_sta_connected_flag** (*esp_wifi_setting_handle_t handle*, bool *flag*)

Set flag to judge whether the station has connected to the AP.

Return

- NULL, Fail
- Others, Success

Parameters

- `[in] handle`: Wifi setting handle
- `[in] flag`: bool type of station connection state

esp_err_t **blufi_set_customized_data** (*esp_wifi_setting_handle_t handle*, char **data*, int *data_len*)

Set customized data to be sent after configurate wifi successfully.

Return

- ESP_FAIL, Fail
- ESP_OK, Success

Parameters

- `[in] handle`: Wifi setting handle
- `[in] data`: Customized data
- `[in] data_len`: Customized data length

esp_err_t **blufi_send_customized_data** (*esp_wifi_setting_handle_t handle*)

Send customized data that be set before.

Return

- ESP_FAIL, Fail

- ESP_OK, Success

Parameters

- `[in] handle`: Wifi setting handle

Header File

- wifi_service/include/airkiss_config.h

Functions

esp_wifi_setting_handle_t **airkiss_config_create**(*airkiss_config_info_t* **info*)

brief Create airkiss setting handle instance

Return

- NULL, Fail
- Others, Success

Parameters

- `info`: Configuration of the airkiss

Structures

struct airkiss_lan_pack_param_t

airkiss lan data pack

Public Members

void ***appid**

APP identifer data

void ***deviceid**

Device identifer data

struct airkiss_config_info_t

airkiss configurations

Public Members

airkiss_lan_pack_param_t **lan_pack**

User lan pack parameter

bool **ssdp_notify_enable**

Notify enable flag

char ***aes_key**

Airkiss aes key data

Macros

AIRKISS_CONFIG_INFO_DEFAULT()

Header File

- wifi_service/include/wifi_ssid_manager.h

Functions

wifi_ssid_manager_handle_t **wifi_ssid_manager_create**(uint8_t *max_ssid_num*)

esp_err_t **wifi_ssid_manager_get_latest_config**(*wifi_ssid_manager_handle_t handle*, wifi_config_t **config*)

esp_err_t **wifi_ssid_manager_save**(*wifi_ssid_manager_handle_t handle*, **const** char **ssid*, **const** char **pwd*)

esp_err_t **wifi_ssid_manager_get_best_config**(*wifi_ssid_manager_handle_t handle*, wifi_config_t **config*)

int **wifi_ssid_manager_get_ssid_num**(*wifi_ssid_manager_handle_t handle*)

esp_err_t **wifi_ssid_manager_list_show**(*wifi_ssid_manager_handle_t handle*)

esp_err_t **wifi_ssid_manager_erase_all**(*wifi_ssid_manager_handle_t handle*)

esp_err_t **wifi_ssid_manager_destroy**(*wifi_ssid_manager_handle_t handle*)

Type Definitions

typedef struct wifi_ssid_manager ***wifi_ssid_manager_handle_t**

2.6.4 OTA 服务

[English]

OTA 服务 (OTA service) 提供对固件进行 OTA 的功能，支持从本地文件、网络获取固件进行升级。

应用示例

以下示例展示了该 API 的实现方式。

- ota

Header File

- ota_service/include/ota_service.h

Functions

periph_service_handle_t **ota_service_create**(*ota_service_config_t* **config*)

Create the OTA service instance.

Return

- NULL: Failed
- Others: Success

Parameters

- `config`: configuration of the OTA service

esp_err_t **ota_service_set_upgrade_param**(*periph_service_handle_t* *handle*, *ota_upgrade_ops_t* **list*, int *list_len*)

Configure the upgrade parameter This function is not thread safe.

This function will set the parameter table to ota service, and the ota service will upgrade the partitions defined in the table one by one,

Return

- ESP_OK: Success
- Others: Failed

Parameters

- `[in] handle`: pointer to 'periph_service_handle_t' structure
- `[in] list`: pointer to '*ota_upgrade_ops_t*' structure
- `[in] list_len`: length of the 'list'

Structures

struct ota_service_config_t

The OTA service configuration.

Public Members

int **task_stack**

>0 Service task stack; =0 with out task created

int **task_prio**

Service task priority (based on freeRTOS priority)

int **task_core**

Service task running in core (0 or 1)

periph_service_cb **evt_cb**

Service callback function

void ***cb_ctx**

Callback context

struct ota_node_attr_t

The OTA node attributions.

Public Members

esp_partition_type_t **type**

Partition type

char ***label**

Partition label

char ***uri**

The upgrade URL

char ***cert_pem**

SSL server certification, PEM format as string, if the client requires to verify server

struct ota_upgrade_ops_t

The upgrade operation.

Public Members

ota_node_attr_t **node**

The OTA node

ota_service_err_reason_t (***prepare**) (void **handle, *ota_node_attr_t* *node)

Functions ready for upgrade

ota_service_err_reason_t (***need_upgrade**) (void *handle, *ota_node_attr_t* *node)

Detect whether an upgrade is required

ota_service_err_reason_t (***execute_upgrade**) (void *handle, *ota_node_attr_t* *node)

For execute upgrade

ota_service_err_reason_t (***finished_check**) (void *handle, *ota_node_attr_t* *node, *ota_service_err_reason_t* result)

Check result of upgrade

bool **reboot_flag**

Reboot or not after upgrade

bool **break_after_fail**

Abort upgrade when got failed

struct ota_result_t

The result of the OTA upgrade.

Public Members

uint8_t **id**

The result ID

ota_service_err_reason_t **result**

The error reason

Macros

OTA_SERVICE_ERR_REASON_BASE

OTA_SERVICE_DEFAULT_CONFIG()

Enumerations

enum ota_service_event_type_t

The OTA service event type.

Values:

OTA_SERV_EVENT_TYPE_RESULT

OTA_SERV_EVENT_TYPE_FINISH

enum ota_service_err_reason_t

The OTA service error reasons.

Values:

OTA_SERV_ERR_REASON_UNKNOWN = ESP_FAIL

OTA_SERV_ERR_REASON_SUCCESS = ESP_OK

OTA_SERV_ERR_REASON_NULL_POINTER = OTA_SERVICE_ERR_REASON_BASE + 1

OTA_SERV_ERR_REASON_URL_PARSE_FAIL = OTA_SERVICE_ERR_REASON_BASE + 2

OTA_SERV_ERR_REASON_ERROR_VERSION = OTA_SERVICE_ERR_REASON_BASE + 3

OTA_SERV_ERR_REASON_NO_HIGHER_VERSION = OTA_SERVICE_ERR_REASON_BASE + 4

OTA_SERV_ERR_REASON_ERROR_MAGIC_WORD = OTA_SERVICE_ERR_REASON_BASE + 5

OTA_SERV_ERR_REASON_ERROR_PROJECT_NAME = OTA_SERVICE_ERR_REASON_BASE + 6

OTA_SERV_ERR_REASON_FILE_NOT_FOUND = OTA_SERVICE_ERR_REASON_BASE + 7

OTA_SERV_ERR_REASON_PARTITION_NOT_FOUND = OTA_SERVICE_ERR_REASON_BASE + 8

OTA_SERV_ERR_REASON_PARTITION_WT_FAIL = OTA_SERVICE_ERR_REASON_BASE + 9

OTA_SERV_ERR_REASON_PARTITION_RD_FAIL = OTA_SERVICE_ERR_REASON_BASE + 10

OTA_SERV_ERR_REASON_STREAM_INIT_FAIL = OTA_SERVICE_ERR_REASON_BASE + 11

OTA_SERV_ERR_REASON_STREAM_RD_FAIL = OTA_SERVICE_ERR_REASON_BASE + 12

OTA_SERV_ERR_REASON_GET_NEW_APP_DESC_FAIL = OTA_SERVICE_ERR_REASON_BASE + 13

Header File

- ota_service/include/ota_proc_default.h

Functions

void **ota_app_get_default_proc**(*ota_upgrade_ops_t* *ops)

get the default process of `app partition` upgrade

Return

- void

Parameters

- `[in] ops`: pointer to *`ota_upgrade_ops_t`* structure

void **ota_data_get_default_proc**(*ota_upgrade_ops_t* *ops)

get the default process of `data partition` upgrade

Return

- void

Parameters

- `[in] ops`: pointer to *`ota_upgrade_ops_t`* structure

ota_service_err_reason_t **ota_data_image_stream_read**(void *handle, char *buf, int wanted_size)

read from the stream of upgrading

Return

- OTA_SERV_ERR_REASON_SUCCESS: Success
- Others: Failed

Parameters

- `[in] handle`: pointer to upgrade handle
- `[in] buf`: pointer to receive buffer
- `[in] wanted_size`: bytes to read

ota_service_err_reason_t **ota_data_partition_write**(void *handle, char *buf, int size)

write to the data partition under upgrading

Return

- OTA_SERV_ERR_REASON_SUCCESS: Success

- Others: Failed

Parameters

- `[in] handle`: pointer to upgrade handle
- `[in] buf`: pointer to data buffer
- `[in] size`: bytes to write

void **ota_data_partition_erase_mark**(void *handle)

Indicates that the ota partition has been erased By default, this part of flash will be erased during ota. If the behavior of erasing is called in applition, this API needs to be called.

Return

- void

Parameters

- `[in] handle`: pointer to upgrade handle

int **ota_get_version_number**(char *version)

Convert string of version to integer The version should be (V0.0.0 - V255.255.255)

Return

- -1: Failed
- Others: version number

Parameters

- `[in] version`: pointer to the string of version

Header File

- ota_service/include/esp_fs_ota.h

Functions

esp_err_t **esp_fs_ota**(*esp_fs_ota_config_t* *ota_config)

Upgrade the firmware from filesystem.

Note This API handles the entire OTA operation, so if this API is being used then no other APIs from `esp_fs_ota` component should be called. If more information and control is needed during the FS OTA process, then one can use `esp_fs_ota_begin` and subsequent APIs. If this API returns successfully, esp_restart() must be called to boot from the new firmware image.

Return

- ESP_OK: OTA data updated, next reboot will use specified partition.
- ESP_FAIL: For generic failure.
- ESP_ERR_INVALID_ARG: Invalid argument
- ESP_ERR_OTA_VALIDATE_FAILED: Invalid app image
- ESP_ERR_NO_MEM: Cannot allocate memory for OTA operation.
- ESP_ERR_FLASH_OP_TIMEOUT or ESP_ERR_FLASH_OP_FAIL: Flash write failed.
- For other return codes, refer OTA documentation in esp-idf ' s app_update component.

Parameters

- `[in] ota_config`: pointer to *esp_fs_ota_config_t* structure.

esp_err_t **esp_fs_ota_begin**(*esp_fs_ota_config_t* *ota_config, *esp_fs_ota_handle_t* *handle)

Start FS OTA Firmware upgrade.

This function initializes ESP FS OTA context and open the firmware file. This function must be invoked first. If this function returns successfully, then `esp_fs_ota_perform` should be called to continue with the OTA process and there should be a call to `esp_fs_ota_finish` on completion of OTA operation or on failure in subsequent operations.

Return

- ESP_OK: FS OTA Firmware upgrade context initialised and file opened successful
- ESP_FAIL: For generic failure.
- ESP_ERR_INVALID_ARG: Invalid argument (missing/incorrect config, etc.)
- For other return codes, refer documentation in app_update component and esp_http_client component in esp-idf.

Parameters

- `[in] ota_config`: pointer to *esp_fs_ota_config_t* structure
- `[out] handle`: pointer to an allocated data of type `esp_fs_ota_handle_t` which will be initialised in this function

esp_err_t **esp_fs_ota_perform**(*esp_fs_ota_handle_t* *fs_ota_handle*)

Read image data from file stream and write it to OTA partition.

This function reads image data from file stream and writes it to OTA partition. This function must be called only if esp_fs_ota_begin() returns successfully. This function must be called in a loop since it returns after every file read operation thus giving you the flexibility to stop OTA operation midway.

Return

- ESP_ERR_FS_OTA_IN_PROGRESS: OTA update is in progress, call this API again to continue.
- ESP_OK: OTA update was successful
- ESP_FAIL: OTA update failed
- ESP_ERR_INVALID_ARG: Invalid argument
- ESP_ERR_OTA_VALIDATE_FAILED: Invalid app image
- ESP_ERR_NO_MEM: Cannot allocate memory for OTA operation.
- ESP_ERR_FLASH_OP_TIMEOUT or ESP_ERR_FLASH_OP_FAIL: Flash write failed.
- For other return codes, refer OTA documentation in esp-idf ' s app_update component.

Parameters

- `[in] fs_ota_handle`: pointer to esp_fs_ota_handle_t structure

esp_err_t **esp_fs_ota_finish**(*esp_fs_ota_handle_t fs_ota_handle*)

Clean-up FS OTA Firmware upgrade and close the file stream.

This function closes the file stream and frees the ESP FS OTA context. This function switches the boot partition to the OTA partition containing the new firmware image.

Note If this API returns successfully, esp_restart() must be called to boot from the new firmware image

Return

- ESP_OK: Clean-up successful
- ESP_ERR_INVALID_STATE
- ESP_ERR_INVALID_ARG: Invalid argument
- ESP_ERR_OTA_VALIDATE_FAILED: Invalid app image

Parameters

- `[in] fs_ota_handle`: pointer to esp_fs_ota_handle_t structure

esp_err_t **esp_fs_ota_get_img_desc**(*esp_fs_ota_handle_t fs_ota_handle*, esp_app_desc_t **new_app_info*)

Reads app description from image header. The app description provides information like the "Firmware version" of the image.

Note This API can be called only after esp_fs_ota_begin() and before esp_fs_ota_perform(). Calling this API is not mandatory.

Return

- ESP_ERR_INVALID_ARG: Invalid arguments
- ESP_FAIL: Failed to read image descriptor

- ESP_OK: Successfully read image descriptor

Parameters

- `[in] fs_ota_handle`: pointer to esp_fs_ota_handle_t structure
- `[out] new_app_info`: pointer to an allocated esp_app_desc_t structure

int **esp_fs_ota_get_image_len_read**(*esp_fs_ota_handle_t fs_ota_handle*)

Structures

struct esp_fs_ota_config_t

ESP FS OTA configuration.

Public Members

const char ***path**

file path

const int **buffer_size**

Size of buffer

Macros

ESP_ERR_FS_OTA_BASE

ESP_ERR_FS_OTA_IN_PROGRESS

Type Definitions

typedef void ***esp_fs_ota_handle_t**

2.6.5 DuerOS 服务

[English]

DuerOS 服务 (DuerOS service) 实现了 DuerOS 的语音交互。

应用示例

以下示例展示了该 API 的实现方式。

- dueros

Header File

- dueros_service/include/dueros_service.h

Functions

audio_service_handle_t **dueros_service_create**()

Create the dueros service.

Return

- NULL, Fail
- Others, Success

service_state_t **dueros_service_state_get**()

Get dueros service state.

Return The state of service

esp_err_t **dueros_voice_upload**(*audio_service_handle_t handle*, void **buf*, int *len*)

Upload voice to backend server.

Return ESP_OK ESP_FAIL

Parameters

- `handle`: dueros service handle
- `buf`: Data buffer
- `len`: Size of buffer

esp_err_t **dueros_voice_cancel**(*audio_service_handle_t handle*)

Cancel the current session.

Return ESP_OK ESP_FAIL

Parameters

- `handle`: dueros service handle

esp_err_t **dueros_start_wifi_cfg** (*audio_service_handle_t handle*, duer_wifi_cfg_t *cfg)

Start the wifi configure process.

Return ESP_OK ESP_FAIL

Parameters

- `handle`: Dueros service handle
- `cfg`: Configuration

esp_err_t **dueros_stop_wifi_cfg** (*audio_service_handle_t handle*)

Stop the wifi configure process.

Return ESP_OK ESP_FAIL

Parameters

- `handle`: Dueros service handle

esp_err_t **dueros_wifi_status_report** (*audio_service_handle_t handle*, *dueros_wifi_st_t* *st)

Report the wifi status to dipb.

Return ESP_OK ESP_FAIL

Parameters

- `handle`: Dueros service handle
- `st`: WiFi status and error code

Structures

struct dueros_wifi_st_t

Status of WiFi connection.

Public Members

int **status**

Please refer to: `enum duer_dipb_client_status_e`

int **err**

Please refer to: `enum duer_wifi_connect_error_code_e`

2.6.6 显示服务

[English]

显示服务 (display service) 在`display_pattern_t` 中定义了一些常用的显示样式枚举值, 帮助用户设置 LED 或者 LED 灯条的对应样式。

已经支持的 LED 驱动芯片有 AW2013、WS2812、IS31x。

应用示例

以下示例展示了该 API 的实现方式。

- checks/check_display_led

Header File

- display_service/include/display_service.h

Functions

display_service_handle_t **display_service_create**(*display_service_config_t* *cfg)

esp_err_t **display_service_set_pattern**(void *handle, int disp_pattern, int value)

esp_err_t **display_destroy**(*display_service_handle_t* handle)

Structures

struct display_service_config_t

Display service configurations.

Public Members

periph_service_config_t **based_cfg**

Peripheral service configuration

void ***instance**

Sub-instance

Type Definitions

typedef struct display_service_impl ***display_service_handle_t**

Enumerations

enum display_pattern_t

Values:

DISPLAY_PATTERN_UNKNOWN = 0

DISPLAY_PATTERN_WIFI_SETTING = 1

DISPLAY_PATTERN_WIFI_CONNECTTING = 2

DISPLAY_PATTERN_WIFI_CONNECTED = 3

DISPLAY_PATTERN_WIFI_DISCONNECTED = 4

DISPLAY_PATTERN_WIFI_SETTING_FINISHED = 5

DISPLAY_PATTERN_BT_CONNECTTING = 6

DISPLAY_PATTERN_BT_CONNECTED = 7

DISPLAY_PATTERN_BT_DISCONNECTED = 8

DISPLAY_PATTERN_RECORDING_START = 9

DISPLAY_PATTERN_RECORDING_STOP = 10

DISPLAY_PATTERN_RECOGNITION_START = 11

DISPLAY_PATTERN_RECOGNITION_STOP = 12

DISPLAY_PATTERN_WAKEUP_ON = 13

DISPLAY_PATTERN_WAKEUP_FINISHED = 14

DISPLAY_PATTERN_MUSIC_ON = 15

DISPLAY_PATTERN_MUSIC_FINISHED = 16

DISPLAY_PATTERN_VOLUME = 17

DISPLAY_PATTERN_MUTE_ON = 18

DISPLAY_PATTERN_MUTE_OFF = 19

DISPLAY_PATTERN_TURN_ON = 20

DISPLAY_PATTERN_TURN_OFF = 21

DISPLAY_PATTERN_BATTERY_LOW = 22

DISPLAY_PATTERN_BATTERY_CHARGING = 23

DISPLAY_PATTERN_BATTERY_FULL = 24

DISPLAY_PATTERN_POWERON_INIT = 25

DISPLAY_PATTERN_WIFI_NO_CFG = 26

DISPLAY_PATTERN_SPEECH_BEGIN = 27

DISPLAY_PATTERN_SPEECH_OVER = 28

DISPLAY_PATTERN_MAX

Header File

- display_service/led_bar/include/led_bar_aw2013.h

Functions

void **aw2013_led_bar_task**(void **parameters*)

esp_periph_handle_t **led_bar_aw2013_init**(void)

Initialize led bar instance.

Return

- NULL Error
- others Success

esp_err_t **led_bar_aw2013_pattern**(void **handle*, int *pat*, int *value*)

Set led bar display pattern.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `handle`: led bar instance
- `pat`: display pattern
- `value`: value of pattern

esp_err_t **led_bar_aw2013_set_blink_time**(void **handle*, uint8_t *time*, int *period*)

Set blinking period and times.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `handle`: led bar instance
- `time`: times of blink
- `period`: period of blink

void **led_bar_aw2013_deinit** (*esp_periph_handle_t handle*)

Destroy esp_periph_handle_t instance.

Parameters

- `handle`: led bar instance

Header File

- display_service/led_bar/include/led_bar_is31x.h

Functions

esp_periph_handle_t **led_bar_is31x_init** ()

Initialize esp_periph_handle_t instance.

Return

- NULL, Fail
- Others, Success

esp_err_t **led_bar_is31x_pattern** (void **handle*, int *pat*, int *value*)

Set led bar display pattern.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `handle`: led bar instance
- `pat`: display pattern
- `value`: value of pattern

void **led_bar_is31x_deinit**(*esp_periph_handle_t handle*)

Destroy esp_periph_handle_t instance.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `handle`: led bar instance

Header File

- display_service/led_bar/include/led_bar_ws2812.h

Functions

led_bar_ws2812_handle_t **led_bar_ws2812_init**(gpio_num_t *gpio_num*, int *led_num*)

Initialize led_bar_ws2812_handle_t instance.

Return

- led_bar_ws2812_handle_t

Parameters

- `gpio_num`: The GPIO number of ws2812
- `led_num`: The number of all ws2812

esp_err_t **led_bar_ws2812_pattern**(void **handle*, int *pat*, int *value*)

Set ws2812 pattern.

Return

- ESP_OK, success
- Others, fail

Parameters

- `handle`: ws2812 indicator instance
- `pat`: display pattern
- `value`: value of pattern

esp_err_t **led_bar_ws2812_deinit**(*led_bar_ws2812_handle_t handle*)

Destroy led_bar_ws2812_handle_t instance.

Return

Return

- ESP_OK, success
- Others, fail

Parameters

- `handle`: ws2812 indicator instance

Type Definitions

typedef struct led_bar_ws2812_impl ***led_bar_ws2812_handle_t**

Header File

- display_service/led_indicator/include/led_indicator.h

Functions

led_indicator_handle_t **led_indicator_init** (gpio_num_t *num*)

Initialize led_indicator_handle_t instance.

Return

- NULL, Fail
- Others, Success

Parameters

- `num`: led gpio number

esp_err_t **led_indicator_pattern** (void **handle*, int *pat*, int *value*)

Set led indicator display pattern.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `handle`: led indicator instance
- `pat`: display pattern

- `value`: value of pattern

void **led_indicator_deinit**(*led_indicator_handle_t handle*)

Destroy led_indicator_handle_t instance.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `handle`: led indicator instance

Type Definitions

typedef struct led_indicator_impl ***led_indicator_handle_t**

2.6.7 电池服务

[English]

电池服务 (battery service) 提供了监测和管理电池电压的功能。电池电压和`battery_service_event_t` 定义的事件可以通过回调函数与用户进行交互。

应用示例

以下示例展示了该 API 的实现方式。

- system/battery

Header File

- battery_service/include/battery_service.h

Functions

periph_service_handle_t **battery_service_create**(*battery_service_config_t *config*)

Create the battery service instance.

Return

- NULL: Failed
- Others: Success

Parameters

- `config`: configuration of the battery service

esp_err_t **battery_service_vol_report_switch**(*periph_service_handle_t* *handle*, bool *on_off*)

Start or stop the battery voltage report.

Return

- ESP_OK: Success
- ESP_ERR_INVALID_ARG：handle is NULL

Parameters

- `[in] handle`: pointer to ‘periph_service_handle_t’ structure
- `[in] on_off`: ‘true’ to start, ‘false’ to stop

esp_err_t **battery_service_set_vol_report_freq**(*periph_service_handle_t* *handle*, int *freq*)

Set voltage report freqency.

Return

- ESP_OK: Success
- ESP_ERR_INVALID_ARG：handle is NULL

Parameters

- `[in] handle`: pointer to ‘periph_service_handle_t’ structure
- `[in] freq`: voltage report freqency

Structures

struct battery_service_config_t

Battery service configure.

Public Members

int **task_stack**

>0 Service task stack; =0 with out task created

int **task_prio**

Service task priority (based on freeRTOS priority)

int **task_core**

Service task running in core (0 or 1)

bool **extern_stack**

Task stack allocate on the extern ram

periph_service_cb **evt_cb**

Service callback function

void ***cb_ctx**

Callback context

vol_monitor_handle_t **vol_monitor**

Battery monitor

void ***charger_monitor**

Charger monitor. Not supported yet

Macros

BATTERY_SERVICE_DEFAULT_CONFIG( )

Enumerations

enum battery_service_event_t

Battery service event.

Values:

BAT_SERV_EVENT_UNKNOWN

BAT_SERV_EVENT_VOL_REPORT = 1

BAT_SERV_EVENT_BAT_FULL

BAT_SERV_EVENT_BAT_LOW

BAT_SERV_EVENT_CHARGING_BEGIN = 100

BAT_SERV_EVENT_CHARGING_STOP

Header File

- battery_service/monitors/include/voltage_monitor.h

Functions

vol_monitor_handle_t **vol_monitor_create**(*vol_monitor_param_t* **config*)

Create the voltage monitor instance.

Return

- NULL: Failed
- Others: Success

Parameters

- `[in] config`: pointer to '*vol_monitor_param_t*' structure

esp_err_t **vol_monitor_destroy**(*vol_monitor_handle_t handle*)

Destroy the voltage monitor.

Return

- ESP_OK: Success
- Others: Failed

Parameters

- `[in] handle`: pointer to 'vol_monitor_handle_t' structure

esp_err_t **vol_monitor_set_event_cb**(*vol_monitor_handle_t handle*, *vol_monitor_event_cb event_cb*, void **user_ctx*)

Set the event callback.

Return

- ESP_OK: Success
- Others: Failed

Parameters

- `[in] handle`: pointer to 'vol_monitor_handle_t' structure
- `[in] event_cb`: callback used to handle the events of voltage monitor
- `[in] user_ctx`: user's data

esp_err_t **vol_monitor_start_freq_report**(*vol_monitor_handle_t handle*)

Start the voltage report with the configured frequency.

Return

- ESP_OK: Success

- Others: Failed

Parameters

- `[in] handle`: pointer to 'vol_monitor_handle_t' structure

esp_err_t **vol_monitor_stop_freq_report**(*vol_monitor_handle_t handle*)

Stop the voltage frequency report.

Return

- ESP_OK: Success
- Others: Failed

Parameters

- `[in] handle`: pointer to 'vol_monitor_handle_t' structure

esp_err_t **vol_monitor_set_report_freq**(*vol_monitor_handle_t handle*, int *freq*)

Set the voltage report frequency.

Return

- ESP_OK: Success
- Others: Failed

Parameters

- `[in] handle`: pointer to 'vol_monitor_handle_t' structure
- `[in] freq`: voltage report freqency

Structures

struct vol_monitor_param_t

Battery adc configure.

Public Members

bool (***init**)(void *)

Voltage read init

bool (***deinit**)(void *)

Voltage read deinit

int (***vol_get**)(void *)

Voltage read interface

void ***user_data**

Parameters for callbacks

int **read_freq**

Voltage read frequency, unit: s

int **report_freq**

Voltage report frequency, voltage will be report with a interval calculate by (`read_freq` * `report_freq`)

int **vol_full_threshold**

Voltage threshold to report, unit: mV

int **vol_low_threshold**

Voltage threshold to report, unit: mV

Type Definitions

typedef void ***vol_monitor_handle_t**

voltage monitor handle

typedef void (***vol_monitor_event_cb**) (int msg_id, void *data, void *user_ctx)

callback define

Enumerations

enum vol_monitor_event_t

Battery adc configure.

Values:

VOL_MONITOR_EVENT_FREQ_REPORT

VOL_MONITOR_EVENT_BAT_FULL

VOL_MONITOR_EVENT_BAT_LOW

2.6.8 核心转储上传服务

[English]

如需调查已出售设备中的 crash 事件，则需要回收回溯信息进行分析。核心转储上传服务 (core dump upload service) 支持通过 HTTP 将存储在设备分区中的回溯信息传输回来。选择 `ESP_COREDUMP_ENABLE_TO_FLASH`，即可启用此功能。

应用示例

以下示例展示了该 API 的实现方式。

- system/coredump

Header File

- coredump_upload_service/include/coredump_upload_service.h

Functions

bool **coredump_need_upload**()

This function will check the reset code and determine whether to upload the coredump.

Return

- true: last reboot is a abnormal reset.
- false

esp_err_t **coredump_upload**(*periph_service_handle_t* *handle*, char **url*)

Upload the core dump image to the url. This function will block the current task until the upload process finished.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] handle`: the 'periph_service_handle_t'
- `[in] url`: server addr

periph_service_handle_t **coredump_upload_service_create**(*coredump_upload_service_config_t* **config*)

Create the core dump upload service instance.

Return

- NULL: Failed
- Others: Success

Parameters

- `config`: configuration of the OTA service

Structures

struct coredump_upload_service_config_t

coredump service configuration parameters

Public Members

int **task_stack**

>0 Service task stack; =0 with out task created

int **task_prio**

Service task priority (based on freeRTOS priority)

int **task_core**

Service task running in core (0 or 1)

periph_service_cb **evt_cb**

Service callback function

void ***cb_ctx**

Callback context

bool (***do_post**)(char *url, uint8_t *data, size_t len)

POST interface, users can override this to customize the http client. if left NULL, the service will use the default one

Macros

COREDUMP_UPLOAD_SERVICE_DEFAULT_CONFIG()

2.6.9 Header File

- esp_dispatcher/include/periph_service.h

2.6.10 Functions

periph_service_handle_t **periph_service_create**(*periph_service_config_t* **config*)

brief Create peripheral service instance

Return

- NULL, Fail
- Others, Success

Parameters

- `[in] config`: Configuration of the peripheral service instance

esp_err_t **periph_service_destroy**(*periph_service_handle_t handle*)

brief Destroy peripheral service instance

Return

- ESP_OK
- ESP_FAIL
- ESP_ERR_INVALID_ARG

Parameters

- `[in] handle`: The peripheral service instance

esp_err_t **periph_service_start**(*periph_service_handle_t handle*)

brief Start the specific peripheral service

Return

- ESP_OK
- ESP_FAIL
- ESP_ERR_INVALID_ARG

Parameters

- `[in] handle`: The peripheral service instance

esp_err_t **periph_service_stop**(*periph_service_handle_t handle*)

brief Stop the specific peripheral service

Return

- ESP_OK
- ESP_FAIL
- ESP_ERR_INVALID_ARG

Parameters

- `[in] handle`: The peripheral service instance

esp_err_t **periph_service_set_callback**(*periph_service_handle_t handle*, *periph_service_cb cb*, void **ctx*)

brief Set the specific peripheral service callback function

Return

- ESP_OK

- ESP_FAIL
- ESP_ERR_INVALID_ARG

Parameters

- `[in] handle`: The peripheral service instance
- `[in] cb`: A pointer to service_callback
- `[in] ctx`: A pointer to user context

esp_err_t **periph_service_callback** (*periph_service_handle_t handle*, *periph_service_event_t *evt*)

brief Called peripheral service by configurations

Return

- ESP_OK
- ESP_FAIL
- ESP_ERR_INVALID_ARG

Parameters

- `[in] handle`: The peripheral service instance
- `[in] evt`: A pointer to *periph_service_event_t*

esp_err_t **periph_service_set_data** (*periph_service_handle_t handle*, void *data)

brief Set user data to specific peripheral service instance

Return

- ESP_OK
- ESP_FAIL
- ESP_ERR_INVALID_ARG

Parameters

- `[in] handle`: The peripheral service instance
- `[in] data`: A pointer to user data

void *__periph_service_get_data__ (*periph_service_handle_t handle*)

brief Get user data by specific peripheral service instance

Return A pointer to user data

Parameters

- `[in] handle`: The peripheral service instance

esp_err_t **periph_service_ioctl**(*periph_service_handle_t* *handle*, void **ioctl_handle*, int *cmd*, int *value*)

brief In/out control by peripheral service instance

Return

- ESP_OK
- ESP_FAIL
- ESP_ERR_INVALID_ARG

Parameters

- `[in] handle`: The peripheral service instance
- `[in] ioctl_handle`: Sub-instance handle
- `[in] cmd`: Command of value
- `[in] value`: Data of the command

2.6.11 Structures

struct periph_service_event_t

Peripheral service event informations.

Public Members

int **type**

Type of event

void ***source**

Event source

void ***data**

Event data

int **len**

Length of data

struct periph_service_config_t

Peripheral service configurations.

Public Members

int **task_stack**

>0 Service task stack; =0 with out task created

int **task_prio**

Service task priority (based on freeRTOS priority)

int **task_core**

Service task running in core (0 or 1)

TaskFunction_t **task_func**

Service task function

bool **extern_stack**

Task stack allocate on the extern ram

periph_service_ctrl **service_start**

Start function

periph_service_ctrl **service_stop**

Stop function

periph_service_ctrl **service_destroy**

Destroy function

periph_service_io **service_ioctl**

In out control function

char ***service_name**

Name of peripheral service

void ***user_data**

User data

2.6.12 Type Definitions

typedef struct periph_service_impl ***periph_service_handle_t**

typedef esp_err_t (***periph_service_ctrl**) (*periph_service_handle_t* handle)

typedef esp_err_t (***periph_service_io**) (void *ioctl_handle, int cmd, int value)

typedef esp_err_t (***periph_service_cb**) (*periph_service_handle_t* handle, *periph_service_event_t* *evt, void *ctx)

2.6.13 Enumerations

enum periph_service_state_t

Peripheral service state.

Values:

PERIPH_SERVICE_STATE_UNKNOWN

PERIPH_SERVICE_STATE_IDLE

PERIPH_SERVICE_STATE_RUNNING

PERIPH_SERVICE_STATE_STOPPED

2.6.14 Header File

- esp_dispatcher/include/audio_service.h

2.6.15 Functions

audio_service_handle_t **audio_service_create**(*audio_service_config_t* **config*)

brief Create audio service instance

Return

- NULL, Fail
- Others, Success

Parameters

- `[in] config`: Configuration of the audio service instance

esp_err_t **audio_service_destroy**(*audio_service_handle_t handle*)

brief Destroy audio service instance

Return

- ESP_OK
- ESP_FAIL
- ESP_ERR_INVALID_ARG

Parameters

- `[in] handle`: The audio service instance

esp_err_t **audio_service_start**(*audio_service_handle_t handle*)

brief Start the specific audio service

Return

- ESP_OK
- ESP_FAIL
- ESP_ERR_INVALID_ARG

Parameters

- `[in] handle`: The audio service instance

esp_err_t **audio_service_stop**(*audio_service_handle_t handle*)

brief Stop the specific audio service

Return

- ESP_OK
- ESP_FAIL
- ESP_ERR_INVALID_ARG

Parameters

- `[in] handle`: The audio service instance

esp_err_t **audio_service_set_callback**(*audio_service_handle_t handle*, *service_callback cb*, void **ctx*)

brief Set the specific audio service callback function.

Return

- ESP_OK
- ESP_FAIL
- ESP_ERR_INVALID_ARG

Parameters

- `[in] handle`: The audio service instance
- `[in] cb`: A pointer to service_callback
- `[in] ctx`: A pointer to user context

esp_err_t **audio_service_callback**(*audio_service_handle_t handle*, *service_event_t *evt*)

brief Called audio service by configurations

Return

- ESP_OK
- ESP_FAIL

- ESP_ERR_INVALID_ARG

Parameters

- `[in] handle`: The audio service instance
- `[in] evt`: A pointer to *service_event_t*

esp_err_t **audio_service_connect** (*audio_service_handle_t handle*)

brief Connect the specific audio service

Return

- ESP_OK
- ESP_FAIL
- ESP_ERR_INVALID_ARG

Parameters

- `[in] handle`: The audio service instance

esp_err_t **audio_service_disconnect** (*audio_service_handle_t handle*)

brief Disconnect the specific audio service

Return

- ESP_OK
- ESP_FAIL
- ESP_ERR_INVALID_ARG

Parameters

- `[in] handle`: The audio service instance

esp_err_t **audio_service_set_data** (*audio_service_handle_t handle*, void **data*)

brief Set user data to specific audio service instance

Return

- ESP_OK
- ESP_FAIL
- ESP_ERR_INVALID_ARG

Parameters

- `[in] handle`: The audio service instance
- `[in] data`: A pointer to user data

void ***audio_service_get_data**(*audio_service_handle_t handle*)

brief Get user data by specific audio service instance

Return A pointer to user data

Parameters

- `[in] handle`: The audio service instance

2.6.16 Structures

struct service_event_t

Audio service event informations.

Public Members

int **type**

Type of event

void ***source**

Event source

void ***data**

Event data

int **len**

Length of data

struct audio_service_config_t

Audio service configurations.

Public Members

int **task_stack**

>0 Service task stack; =0 with out task created

int **task_prio**

Service task priority (based on freeRTOS priority)

int **task_core**

Service task running in core (0 or 1)

TaskFunction_t **task_func**

A pointer to TaskFunction_t for service task function

service_ctrl **service_start**

Start function

service_ctrl **service_stop**
: Stop function

service_ctrl **service_connect**
: Connect function

service_ctrl **service_disconnect**
: Disconnect function

service_ctrl **service_destroy**
: Destroy function

const char ***service_name**
: Name of audio service

void ***user_data**
: User context

2.6.17 Type Definitions

typedef struct audio_service_impl ***audio_service_handle_t**

typedef esp_err_t (***service_ctrl**) (*audio_service_handle_t* handle)

typedef esp_err_t (***service_callback**) (*audio_service_handle_t* handle, *service_event_t* *evt, void *ctx)

2.6.18 Enumerations

enum service_state_t
: Audio service state.

Values:

SERVICE_STATE_UNKNOWN

SERVICE_STATE_IDLE

SERVICE_STATE_CONNECTING

SERVICE_STATE_CONNECTED

SERVICE_STATE_RUNNING

SERVICE_STATE_STOPPED

2.7 Speech Recognition

ESP-ADF offers a comprehensive range of speech recognition processing functions, such as front-end speech processing, TTS, voice wake-up, and command word recognition. ESP-ADF's Element-based *audio recorder* integrates speech recognition and audio signal processing into an event-driven High-level API. This approach enables users to maintain flexibility in their configurations while making it easier to use.

2.7.1 Voice Activity Detection

Voice activity detection (VAD) is a technique used in speech processing to detect the presence (or absence) of human speech. Detection of somebody speaking may be used to activate some processes, e.g. automatically switch on voice recording. It may be also used to deactivate processes, e.g. stop coding and transmission of silence packets to save on computation and network bandwidth.

Provided in this section API implements VAD functionality together with couple of options to configure sensitivity of speech detection, set sample rate or duration of audio samples.

Application Example

Implementation of the voice activity detection API is demonstrated in speech_recognition/vad example.

API Reference

For the latest API reference please refer to Espressif Speech recognition repository.

2.7.2 WakeNet Interface

Setting up the speech recognition application to detect a wakeup word may be done using series of Audio Elements linked into a pipeline shown below.

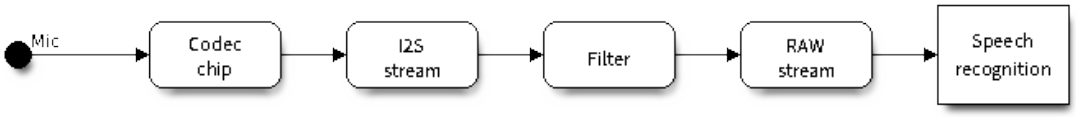


图 5: Sample Speech Recognition Pipeline

Configuration and use of particular elements is demonstrated in several examples linked to elsewhere in this documentation. What may need clarification is use of the **Filter** and the **RAW stream**. The filter is used to adjust the sample rate of the I2S stream to match the sample rate of the speech recognition model. The RAW stream is the way to feed the audio input to the model.

The above introduction is the primary guidance. ESP-ADF offers users a more flexible and convenient module, namely the *audio recorder*, which is strongly recommended for use.

API Reference

For the latest speech recognition API reference, please refer to ESP-SR Speech Recognition Framework.

2.7.3 Audio Recorder

The Audio Recorder API is a set of functions to facilitate voice recording. It combines two important functions, namely Audio Front End (AFE) and audio encoding. This allows users to customize AFE's Voice Activity Detection (VAD), Automatic Gain Control (AGC), and Acoustic Echo Cancellation (AEC) settings. The encoding function is used by users to establish the encoding audio element, which supports various formats such as AAC, AMR-NB, AMR-WB, ADPCM, WAV, OPUS, and G711. The *audio_rec_evt_t* event makes it easy for users to interact with the Audio Recorder software.

The data path of the Audio recorder is presented in the diagram below.

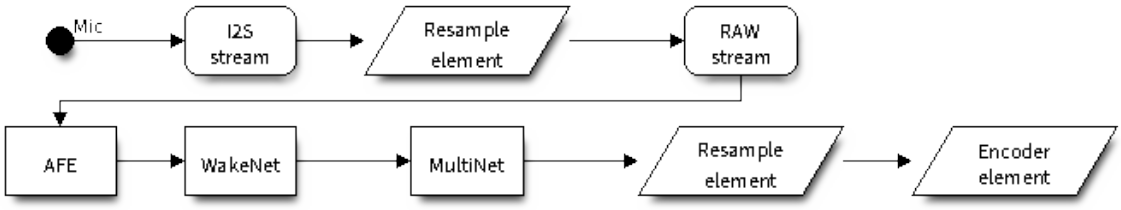


图 6: Audio recorder data path

The area represented by the parallelogram is configurable by the user according to their needs, such as sampling frequency, whether to encode, and encoding format.

Application Example

The speech_recognition/wwe/ example demonstrates how to initialize the speech recognition model, determine the number of samples and the sample rate of voice data to feed to the model, detect the wake-up word and command words, and encode voice to specific audio format.

API Reference

Header File

- audio_recorder/include/audio_recorder.h

Functions

audio_rec_handle_t **audio_recorder_create** (*audio_rec_cfg_t *cfg*)

Initialize and start up audio recorder.

Return NULL failed Others audio recorder handle

Parameters

- `cfg`: Configuration of audio recorder

esp_err_t **audio_recorder_trigger_start** (*audio_rec_handle_t handle*)

Start recording by force.

Note If there need to read from recorder without wake word detected or read from recorder while the wake word detection is disabled, this interface can be use to force start the recorder process.

Return ESP_OK ESP_FAIL

Parameters

- `handle`: Audio recorder handle

esp_err_t **audio_recorder_trigger_stop** (*audio_rec_handle_t handle*)

Stop recording by force.

Note No matter the recorder process is triggered by wake word detected or triggered by `audio_recorder_trigger_start`, this function can be used to force stop the recorder. And if the VAD detection is disabeled, this must be invoked to stop recording after `audio_recorder_trigger_start`.

Return ESP_OK ESP_FAIL

Parameters

- `handle`: Audio recorder handle

esp_err_t **audio_recorder_wakenet_enable** (*audio_rec_handle_t handle*, bool *enable*)

Enable or suspend wake word detection.

Return ESP_OK ESP_FAIL

Parameters

- `handle`: Audio recorder handle
- `enable`: true: enable wake word detection false: disable wake word detection

esp_err_t **audio_recorder_multinet_enable**(*audio_rec_handle_t handle*, bool *enable*)

Enable or suspend speech command recognition.

Return ESP_OK ESP_FAIL

Parameters

- `handle`: Audio recorder handle
- `enable`: true: enable speech command recognition false: disable speech command recognition

esp_err_t **audio_recorder_vad_check_enable**(*audio_rec_handle_t handle*, bool *enable*)

Enable or suspend voice duration check.

Return ESP_OK ESP_FAIL

Parameters

- `handle`: Audio recorder handle
- `enable`: true: enable voice duration check false: disable voice duration check

int **audio_recorder_data_read**(*audio_rec_handle_t handle*, void **buffer*, int *length*, TickType_t *ticks*)

Read data from audio recorder.

Return Length of data actually read ESP_ERR_INVALID_ARG

Parameters

- `handle`: Audio recorder handle
- `buffer`: Buffer to save data
- `length`: Size of buffer
- `ticks`: Timeout for reading

esp_err_t **audio_recorder_destroy**(*audio_rec_handle_t handle*)

Destroy audio recorder and recycle all resource.

Return ESP_OK ESP_FAIL

Parameters

- `handle`: Audio recorder handle

bool **audio_recorder_get_wakeup_state**(*audio_rec_handle_t handle*)

Get the wake up state of audio recorder.

Return true false

Parameters

- `handle`: Audio recorder handle

Structures

struct audio_rec_evt_t

Recorder event.

Public Types

enum [anonymous]

Audio recorder event type.

Values:

AUDIO_REC_WAKEUP_START = -100

Wakeup start

AUDIO_REC_WAKEUP_END

Wakeup stop

AUDIO_REC_VAD_START

Vad start

AUDIO_REC_VAD_END

Vad stop

AUDIO_REC_COMMAND_DECT = 0

Form 0 is the id of the voice commands detected by Multinet

Public Members

audio_rec_evt_t::**[anonymous] type**

Audio recorder event type.

Event type

void ***event_data**

Event data: For `AUDIO_REC_WAKEUP_START`, event data is `recorder_sr_wakeup_result_t` For `AUDIO_REC_COMMAND_DECT` or higher, event data is `recorder_sr_mn_result_t` For other events, event data is NULL

size_t **data_len**

Length of event data

struct audio_rec_cfg_t

Audio recorder configuration.

Public Members

int **pinned_core**

Audio recorder task pinned to core

int **task_prio**

Audio recorder task priority

int **task_size**

Audio recorder task stack size

rec_event_cb_t **event_cb**

Event callback function, event type as *audio_rec_evt_t* shown above

void ***user_data**

Pointer to user data (optional)

recorder_data_read_t **read**

Data callback function used feed data to audio recorder

void ***sr_handle**

SR handle

recorder_sr_iface_t ***sr_iface**

SR interface

int **wakeup_time**

Unit:ms. The duration that the wakeup state remains when VAD is not triggered

int **vad_start**

Unit:ms. Consecutive speech frame will be judged to vad start

int **vad_off**

Unit:ms. When the silence time exceeds this value, it is determined as AUDIO_REC_VAD_END state

int **wakeup_end**

Unit:ms. When the silence time after AUDIO_REC_VAD_END state exceeds this value, it is determined as AUDIO_REC_WAKEUP_END

void ***encoder_handle**

Encoder handle

recorder_encoder_iface_t ***encoder_iface**

Encoder interface

Macros

AUDIO_REC_DEF_TASK_SZ

Stack size of recorder task

AUDIO_REC_DEF_TASK_PRIO

Priority of recoder task

AUDIO_REC_DEF_TASK_CORE

Pinned to core

AUDIO_REC_DEF_WAKEUP_TM

Default wake up time (ms)

AUDIO_REC_DEF_WAKEEND_TM

Duration after vad off (ms)

AUDIO_REC_VAD_START_SPEECH_MS

Consecutive speech frame will be judged to vad start (ms)

AUDIO_REC_DEF_VAD_OFF_TM

Default vad off time (ms)

AUDIO_RECORDER_DEFAULT_CFG()

Type Definitions

typedef esp_err_t (***rec_event_cb_t**)(*audio_rec_evt_t* *event, void *user_data)

Event Notification.

typedef struct __audio_recorder ***audio_rec_handle_t**

Audio recorder handle.

Header File

- audio_recorder/include/recorder_sr.h

Functions

recorder_sr_handle_t **recorder_sr_create**(*recorder_sr_cfg_t* **cfg*, recorder_sr_iface_t ***iface*)

Initialize sr processor, and the sr is disabled as default.

Return NULL failed Others SR handle

Parameters

- `cfg`: Configuration of sr

- `iface`: User interface provide by recorder sr

esp_err_t **recorder_sr_destroy**(*recorder_sr_handle_t handle*)

Destroy SR processor and recycle all resource.

Return ESP_OK ESP_FAIL

Parameters

- `handle`: SR processor handle

esp_err_t **recorder_sr_reset_speech_cmd**(*recorder_sr_handle_t handle*, char **command_str*, char **err_phrase_id*)

Reset the speech commands.

Return ESP_OK ESP_FAIL

Parameters

- `handle`: SR processor handle
- `command_str`: String of the commands. more details on `#2reset-api-on-the-fly`
- `err_phrase_id`: error string output

Structures

struct recorder_sr_cfg_t

SR processor configuration.

Note Since the detection of command words requires a clear starting point, the moment the wake word is detected is taken as the default start of detection. Therefore, if the wake word detection is disabled, the detection will use the `vad_state` detected by `esp-sr` as the start of detection. However, due to the fluctuation of this `vad_state`, the effectiveness of command word detection will be limited.

Public Members

afe_config_t **afe_cfg**

Configuration of AFE

int8_t **input_order**[**DAT_CH_MAX**]

Channel order of the input data

bool **multinet_init**

Enable of speech command recognition

int **feed_task_core**

Core id of feed task

int **feed_task_prio**
: Priority of feed task

int **feed_task_stack**
: Stack size of feed task

int **fetch_task_core**
: Core id of fetch task

int **fetch_task_prio**
: Priority of fetch task

int **fetch_task_stack**
: Stack size of fetch task

int **rb_size**
: Ringbuffer size of recorder sr

char ***partition_label**
: Partition label which stored the model data

char ***mn_language**
: Command language for multinet to load

char ***wn_wakeword**
: Wake Word for WakeNet to load. This is useful when multiple Wake Words are selected in sdkconfig. Setting this to NULL will use the first found model.

Macros

FEED_TASK_STACK_SZ

FETCH_TASK_STACK_SZ

FEED_TASK_PRIO

FETCH_TASK_PRIO

FEED_TASK_PINNED_CORE

FETCH_TASK_PINNED_CORE

SR_OUTPUT_RB_SIZE

INPUT_ORDER_DEFAULT()

DEFAULT_RECORDER_SR_CFG()

Type Definitions

typedef void ***recorder_sr_handle_t**

SR processor handle.

Header File

- audio_recorder/include/recorder_encoder.h

Functions

recorder_encoder_handle_t **recorder_encoder_create**(*recorder_encoder_cfg_t* **cfg*, recorder_encoder_iface_t ***iface*)

Initialize encoder processor, and the encoder is disabled as default.

Return NULL failed Others encoder handle

Parameters

- `cfg`: Configuration of encoder
- `iface`: User interface provide by recorder encoder

esp_err_t **recorder_encoder_destroy**(*recorder_encoder_handle_t* *handle*)

Destroy encoder processor and recycle all resource.

Return ESP_OK ESP_FAIL

Parameters

- `handle`: Encoder processor handle

Structures

struct recorder_encoder_cfg_t

recorder encoder configuration parameters

Public Members

audio_element_handle_t **resample**
Handle of resample

audio_element_handle_t **encoder**
Handle of encoder

Type Definitions

typedef void ***recorder_encoder_handle_t**
encoder handle

2.8 Peripherals

There are several peripherals available in the ESP-ADF, ranging from buttons and LEDs to SD Card or Wi-Fi. The peripherals are implemented using common API that is then expanded with peripheral specific functionality. The following description covers common functionality.

2.8.1 ESP Peripherals

This library simplifies the management of peripherals, by pooling and monitoring in a single task, adding basic functions to send and receive events. And it also provides APIs to easily integrate new peripherals.

注解: Note that if you do not intend to integrate new peripherals into esp_peripherals, you are only interested in simple api `esp_periph_init`, `esp_periph_start`, `esp_periph_stop` and `esp_periph_destroy`. If you want to integrate new peripherals, please refer to *Periph Button* source code

---

Examples

Please refer to player/pipeline_http_mp3/main/play_http_mp3_example.c.

API Reference

Header File

- esp_peripherals/include/esp_peripherals.h

Functions

esp_periph_set_handle_t **esp_periph_set_init** (*esp_periph_config_t *config*)

Initialize esp_peripheral sets, create empty peripherals list. Call this function before starting any peripherals (with `esp_periph_start`). This call will initialize the data needed for esp_peripherals to work, but does not actually create the task. The `event_handle` is optional if you want to receive events from this callback function. The esp_peripherals task will send all events out to event_iface, can be listen by event_iface by `esp_periph_get_event_iface`. The `user_context` will sent `esp_periph_event_handle_t` as *context parameter.

Return The peripheral sets instance

Parameters

- `[in] config`: The configurations

esp_err_t **esp_periph_set_destroy** (*esp_periph_set_handle_t periph_set_handle*)

This function will stop and kill the monitor task, calling all destroy callback functions of the peripheral (so you do not need to destroy the peripheral object manually). It will also remove all memory allocated to the peripherals list, so you need to call the `esp_periph_set_init` function again if you want to use it.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `periph_set_handle`: The esp_periph_set_handle_t instance

esp_err_t **esp_periph_set_stop_all** (*esp_periph_set_handle_t periph_set_handle*)

Stop monitoring all peripherals, the peripheral state is still kept. This funciton only temporary disables the peripheral.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `periph_set_handle`: The esp_periph_set_handle_t instance

esp_periph_handle_t **esp_periph_set_get_by_id**(*esp_periph_set_handle_t periph_set_handle*, int *periph_id*)

Get the peripheral handle by Peripheral ID.

Return The esp_periph_handle_t

Parameters

- `periph_set_handle`: The esp_periph_set_handle_t instance
- `[in] periph_id`: as esp_periph_id_t, or any ID you use when calling `esp_periph_create`

audio_event_iface_handle_t **esp_periph_set_get_event_iface**(*esp_periph_set_handle_t periph_set_handle*)

Return the event_iface used by this esp_peripherals.

Return The audio event iface handle

Parameters

- `periph_set_handle`: The esp_periph_set_handle_t instance

esp_err_t **esp_periph_set_register_callback**(*esp_periph_set_handle_t periph_set_handle*, *esp_periph_event_handle_t cb*, void **user_context*)

Register peripheral sets event callback function.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `periph_set_handle`: The esp_periph_set_handle_t instance
- `cb`: The event handle callback function
- `user_context`: The user context pointer

QueueHandle_t **esp_periph_set_get_queue**(*esp_periph_set_handle_t periph_set_handle*)

Peripheral is using event_iface to control the event, all events are send out to event_iface queue. This function will be useful in case we want to read events directly from the event_iface queue.

Return The queue handle

Parameters

- `periph_set_handle`: The esp_periph_set_handle_t instance

esp_err_t **esp_periph_set_list_init**(*esp_periph_set_handle_t periph_set_handle*)

Call this function to initialize all the listed peripherals.

Note Work with no task peripheral set only

Return

- ESP_OK
- ESP_FAIL

Parameters

- `periph_set_handle`: The esp_periph_set_handle_t instance

esp_err_t **esp_periph_set_list_run**(*esp_periph_set_handle_t periph_set_handle*, *audio_event_iface_msg_t msg*)

Call this function to run all the listed peripherals.

Note Work with no task peripheral set only

Return

- ESP_OK
- ESP_FAIL

Parameters

- `periph_set_handle`: The esp_periph_set_handle_t instance
- `msg`: The *audio_event_iface_msg_t* handle message

esp_err_t **esp_periph_set_list_destroy**(*esp_periph_set_handle_t periph_set_handle*)

Call this function to destroy all the listed peripherals.

Note Work with no task peripheral set only

Return

- ESP_OK
- ESP_FAIL

Parameters

- `periph_set_handle`: The esp_periph_set_handle_t instance

esp_err_t **esp_periph_remove_from_set**(*esp_periph_set_handle_t periph_set_handle*, *esp_periph_handle_t periph*)

Call this function to remove periph from periph_set.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `periph_set_handle`: The esp_periph_set_handle_t instance
- `periph`: The esp_periph_handle_t instance

esp_err_t **esp_periph_set_change_waiting_time**(*esp_periph_set_handle_t periph_set_handle*, int *time_ms*)

Call this function to change periph_set waiting time.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `periph_set_handle`: The esp_periph_set_handle_t instance
- `time_ms`: The waiting time

esp_periph_handle_t **esp_periph_create**(int *periph_id*, **const** char **tag*)

Call this function to initialize a new peripheral.

Return The peripheral handle

Parameters

- `[in] periph_id`: The periph identifier
- `[in] tag`: The tag name, we named it easy to get in debug logs

esp_err_t **esp_periph_set_function**(*esp_periph_handle_t periph*, *esp_periph_func init*, *esp_periph_run_func run*, *esp_periph_func destroy*)

Each peripheral has a cycle of sequential operations from initialization, execution of commands to destroying the peripheral. These operations are represented by functions passed as call parameters to this function.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] periph`: The periph
- `[in] init`: The initialize
- `[in] run`: The run

- `[in] destroy`: The destroy

esp_err_t **esp_periph_start** (*esp_periph_set_handle_t periph_set_handle*, *esp_periph_handle_t periph*)

Add the peripheral to peripherals list, enable and start monitor task (if task stack size > 0)

Note This peripheral must be first created by calling `esp_periph_create`

Return

- ESP_OK on success
- ESP_FAIL when any errors

Parameters

- `[in] periph_set_handle`: The esp_periph_set_handle_t instance
- `[in] periph`: The peripheral instance

esp_err_t **esp_periph_stop** (*esp_periph_handle_t periph*)

Stop monitoring the peripheral, the peripheral state is still kept. This funciton only temporary disables the peripheral.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] periph`: The periph

esp_err_t **esp_periph_send_cmd** (*esp_periph_handle_t periph*, int *cmd*, void **data*, int *data_len*)

When this function is called, the command is passed to the event_iface command queue, and the `esp_periph_run_func` of this peripheral will be executed in the main peripheral task. This function can be called from any task, basically it only sends a queue to the main peripheral task.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] periph`: The periph
- `[in] cmd`: The command
- `data`: The data
- `[in] data_len`: The data length

esp_err_t **esp_periph_send_cmd_from_isr**(*esp_periph_handle_t periph*, int *cmd*, void **data*, int *data_len*)

Similar to `esp_periph_send_cmd`, but it can be called in the hardware interrupt handle.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] periph`: The periph
- `[in] cmd`: The command
- `data`: The data
- `[in] data_len`: The data length

esp_err_t **esp_periph_send_event**(*esp_periph_handle_t periph*, int *event_id*, void **data*, int *data_len*)

In addition to sending an event via event_iface, this function will dispatch the `event_handle` callback if the event_handle callback is provided at `esp_periph_init`

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] periph`: The peripheral
- `[in] event_id`: The event identifier
- `data`: The data
- `[in] data_len`: The data length

esp_err_t **esp_periph_start_timer**(*esp_periph_handle_t periph*, TickType_t *interval_tick*, *timer_callback callback*)

Each peripheral can initialize a timer, which is by default NULL. When this function is called, the timer for the peripheral is created and it invokes the callback function every interval tick.

Note

- You do not need to stop or destroy the timer, when the `esp_periph_destroy` function is called, it will stop and destroy all
- This timer using FreeRTOS Timer, with autoreload = true

Return

- ESP_OK

- ESP_FAIL

Parameters

- `[in] periph`: The peripheral
- `[in] interval_tick`: The interval tick
- `[in] callback`: The callback

esp_err_t **esp_periph_stop_timer**(*esp_periph_handle_t periph*)

Stop peripheral timer.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] periph`: The peripheral

esp_err_t **esp_periph_set_data**(*esp_periph_handle_t periph*, void **data*)

Set the user data.

Note Make sure the `data` lifetime is sufficient, this function does not copy all data, it only stores the data pointer

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] periph`: The peripheral
- `data`: The data

void ***esp_periph_get_data**(*esp_periph_handle_t periph*)

Get the user data stored in the peripheral.

Return Peripheral data pointer

Parameters

- `[in] periph`: The peripheral

esp_periph_state_t **esp_periph_get_state**(*esp_periph_handle_t periph*)

Get the current state of peripheral.

Return The peripharal working state

Parameters

- `[in] periph`: The handle of peripheral

esp_periph_id_t **esp_periph_get_id**(*esp_periph_handle_t periph*)

Get Peripheral identifier.

Return The peripheral identifier

Parameters

- `[in] periph`: The peripheral

esp_err_t **esp_periph_set_id**(*esp_periph_handle_t periph*, *esp_periph_id_t periph_id*)

Set Peripheral identifier.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] periph`: The peripheral
- `[in] periph_id`: The peripheral identifier

esp_err_t **esp_periph_init**(*esp_periph_handle_t periph*)

Call this to execute `init` function of peripheral instance.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `periph`: The peripheral handle

esp_err_t **esp_periph_run**(*esp_periph_handle_t periph*)

Call this to execute `run` function of peripheral instance.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `periph`: The peripheral handle

esp_err_t **esp_periph_destroy**(*esp_periph_handle_t periph*)

Call this to execute `destroy` function of peripheral instance.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `periph`: The peripheral handle

esp_err_t **esp_periph_register_on_events**(*esp_periph_handle_t periph*, *esp_periph_event_t *evts*)

Rigster peripheral on event handle.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `periph`: The peripheral handle
- `evts`: The esp_periph_event_t handle

Structures

struct esp_periph_config_t

Common peripherals configurations.

Public Members

int **task_stack**

>0 Service task stack size; =0 without task created

int **task_prio**

Service task priority (based on freeRTOS priority)

int **task_core**

Service task running in core (0 or 1)

bool **extern_stack**

Service task stack allocate on extern ram

struct esp_periph_event

peripheral events

Public Members

void ***user_ctx***
: peripheral context data

esp_periph_event_handle_t ***cb***
: peripheral callback function

audio_event_iface_handle_t ***iface***
: peripheral event

Macros

DEFAULT_ESP_PERIPH_STACK_SIZE

DEFAULT_ESP_PERIPH_TASK_PRIO

DEFAULT_ESP_PERIPH_TASK_CORE

DEFAULT_ESP_PERIPH_SET_CONFIG()

periph_tick_get

Type Definitions

typedef struct esp_periph_sets ***esp_periph_set_handle_t**

typedef struct esp_periph ***esp_periph_handle_t**

typedef esp_err_t (***esp_periph_func**)(*esp_periph_handle_t* periph)

typedef esp_err_t (***esp_periph_run_func**)(*esp_periph_handle_t* periph, *audio_event_iface_msg_t* *msg)

typedef esp_err_t (***esp_periph_event_handle_t**)(*audio_event_iface_msg_t* *event, void *context)

typedef void (***timer_callback**)(xTimerHandle tmr)

typedef struct *esp_periph_event* **esp_periph_event_t**
: peripheral events

Enumerations

enum esp_periph_id_t

Peripheral Identify, this must be unique for each peripheral added to the peripherals list.

Values:

PERIPH_ID_BUTTON = *AUDIO_ELEMENT_TYPE_PERIPH* + 1

PERIPH_ID_TOUCH = *AUDIO_ELEMENT_TYPE_PERIPH* + 2

PERIPH_ID_SDCARD = *AUDIO_ELEMENT_TYPE_PERIPH* + 3

PERIPH_ID_WIFI = *AUDIO_ELEMENT_TYPE_PERIPH* + 4

PERIPH_ID_FLASH = *AUDIO_ELEMENT_TYPE_PERIPH* + 5

PERIPH_ID_AUXIN = *AUDIO_ELEMENT_TYPE_PERIPH* + 6

PERIPH_ID_ADC = *AUDIO_ELEMENT_TYPE_PERIPH* + 7

PERIPH_ID_CONSOLE = *AUDIO_ELEMENT_TYPE_PERIPH* + 8

PERIPH_ID_BLUETOOTH = *AUDIO_ELEMENT_TYPE_PERIPH* + 9

PERIPH_ID_LED = *AUDIO_ELEMENT_TYPE_PERIPH* + 10

PERIPH_ID_SPIFFS = *AUDIO_ELEMENT_TYPE_PERIPH* + 11

PERIPH_ID_ADC_BTN = *AUDIO_ELEMENT_TYPE_PERIPH* + 12

PERIPH_ID_IS31FL3216 = *AUDIO_ELEMENT_TYPE_PERIPH* + 13

PERIPH_ID_GPIO_ISR = *AUDIO_ELEMENT_TYPE_PERIPH* + 14

PERIPH_ID_WS2812 = *AUDIO_ELEMENT_TYPE_PERIPH* + 15

PERIPH_ID_AW2013 = *AUDIO_ELEMENT_TYPE_PERIPH* + 16

PERIPH_ID_LCD = *AUDIO_ELEMENT_TYPE_PERIPH* + 17

enum esp_periph_state_t

Peripheral working state.

Values:

PERIPH_STATE_NULL

PERIPH_STATE_INIT

PERIPH_STATE_RUNNING

PERIPH_STATE_PAUSE

PERIPH_STATE_STOPPING

PERIPH_STATE_ERROR

PERIPH_STATE_STATUS_MAX

The peripheral specific functionality is available by calling dedicated functions described below. Some peripherals are available on both *ESP32-LyraT* and *ESP32-LyraTD-MSC* development boards, some on a specific board only. The following table provides all implemented peripherals broken down by development board.

2.8.2 Wi-Fi Peripheral

The Wi-Fi Peripheral is used to configure Wi-Fi connections, provide APIs to control Wi-Fi connection configuration, as well as monitor the status of Wi-Fi networks.

Application Example

Implementation of this API is demonstrated in player/pipeline_http_mp3 example.

API Reference

Header File

- esp_peripherals/include/periph_wifi.h

Functions

esp_periph_handle_t **periph_wifi_init** (*periph_wifi_cfg_t* *config)

Create the wifi peripheral handle for esp_peripherals.

Note The handle was created by this function automatically destroy when `esp_periph_destroy` is called

Return The esp peripheral handle

Parameters

- `config`: The configuration

esp_err_t **periph_wifi_wait_for_connected** (*esp_periph_handle_t* *periph*, TickType_t *tick_to_wait*)

This function will block current thread (in `tick_to_wait` tick) and wait until ESP32 connected to the Wi-Fi network, and got ip.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] periph`: The periph
- `[in] tick_to_wait`: The tick to wait

periph_wifi_state_t **periph_wifi_is_connected**(*esp_periph_handle_t periph*)

Check the Wi-Fi connection status.

Return Wi-Fi network status

Parameters

- `[in] periph`: The periph

esp_err_t **esp_wifi_set_listen_interval**(*esp_periph_handle_t periph*, int *interval*)

Set Wi-Fi listen interval for ESP32 station to receive beacon.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] periph`: The wifi periph
- `[in] interval`: listen interval. units: AP beacon intervals(see BcnInt, default: 100ms)

esp_err_t **periph_wifi_config_start**(*esp_periph_handle_t periph*, *periph_wifi_config_mode_t mode*)

Start Wi-Fi network setup in `mode`

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] periph`: The periph
- `[in] mode`: The mode

esp_err_t **periph_wifi_config_wait_done**(*esp_periph_handle_t periph*, TickType_t *tick_to_wait*)

Wait for Wi-Fi setup done.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] periph`: The periph
- `[in] tick_to_wait`: The tick to wait

Structures

struct periph_wpa2_enterprise_cfg_t

The WPA2 enterprise peripheral configuration.

Public Members

bool **diasble_wpa2_e**

Disable wpa2 enterprise

int **eap_method**

TLS: 0, PEAP: 1, TTLS: 2

char ***ca_pem_start**

binary wpa2 ca pem start

char ***ca_pem_end**

binary wpa2 ca pem end

char ***wpa2_e_cert_start**

binary wpa2 cert start

char ***wpa2_e_cert_end**

binary wpa2 cert end

char ***wpa2_e_key_start**

binary wpa2 key start

char ***wpa2_e_key_end**

binary wpa2 key end

const char ***eap_id**

Identity in phase 1 of EAP procedure

const char ***eap_username**

Username for EAP method (PEAP and TTLS)

const char ***eap_password**

Password for EAP method (PEAP and TTLS)

struct periph_wifi_cfg_t

The Wi-Fi peripheral configuration.

Public Members

bool **disable_auto_reconnect**

Disable Wi-Fi auto reconnect

int **reconnect_timeout_ms**

The reconnect timeout after disconnected from Wi-Fi network

wifi_config_t **wifi_config**

Wifi configure

periph_wpa2_enterprise_cfg_t **wpa2_e_cfg**

wpa2 enterprise config

Enumerations

enum periph_wifi_state_t

Peripheral Wi-Fi event id.

Values:

PERIPH_WIFI_UNCHANGE = 0

PERIPH_WIFI_CONNECTING

PERIPH_WIFI_CONNECTED

PERIPH_WIFI_DISCONNECTED

PERIPH_WIFI_SETTING

PERIPH_WIFI_CONFIG_DONE

PERIPH_WIFI_CONFIG_ERROR

PERIPH_WIFI_ERROR

enum periph_wifi_config_mode_t

Wi-Fi setup mode type.

Values:

WIFI_CONFIG_ESPTOUCH

Using smartconfig with ESPTOUCH protocol

WIFI_CONFIG_AIRKISS

Using smartconfig with AIRKISS protocol

WIFI_CONFIG_ESPTOUCH_AIRKISS

Using smartconfig with ESPTOUCH_AIRKISS protocol

WIFI_CONFIG_WPS
Using WPS (not support)

WIFI_CONFIG_BLUEFI
Using BLUEFI

WIFI_CONFIG_WEB
Using HTTP Server (not support)

2.8.3 SD Card Peripheral

If your board has a SD card connected, use this API to initialize, mount and unmount the card, see functions *periph_sdcard_init()*, periph_sdcard_mount() and periph_sdcard_unmount(). The data reading / writing is implemented in a separate API described in *FatFs 流*.

Application Examples

Implementation of this API is demonstrated in couple of examples:

- player/pipeline_play_sdcard_music
- player/pipeline_sdcard_mp3_control
- recorder/pipeline_recording_to_sdcard
- recorder/pipeline_wav_amr_sdcard

API Reference

Header File

- esp_peripherals/include/periph_sdcard.h

Functions

esp_periph_handle_t **periph_sdcard_init** (*periph_sdcard_cfg_t *sdcard_config*)
Create the sdcard peripheral handle for esp_peripherals.

Note The handle was created by this function automatically destroy when esp_periph_destroy is called

Return The esp peripheral handle

Parameters

- sdcard_config: The sdcard configuration

bool **periph_sdcard_is_mounted**(*esp_periph_handle_t periph*)

Check the sdcard is mounted or not.

Return SDCARD mounted state

Parameters

- `[in] periph`: The periph

Structures

struct periph_sdcard_cfg_t

The SD Card Peripheral configuration.

Public Members

int **card_detect_pin**

Card detect gpio number

const char ***root**

Base path for vfs

periph_sdcard_mode_t **mode**

card mode

Enumerations

enum periph_sdcard_event_id_t

Peripheral sdcard event id.

Values:

SDCARD_STATUS_UNKNOWN

No event

SDCARD_STATUS_CARD_DETECT_CHANGE

Detect changes in the card_detect pin

SDCARD_STATUS_MOUNTED

SDCARD mounted successfully

SDCARD_STATUS_UNMOUNTED

SDCARD unmounted successfully

SDCARD_STATUS_MOUNT_ERROR

SDCARD mount error

SDCARD_STATUS_UNMOUNT_ERROR
SDCARD unmount error

enum periph_sdcard_mode_t
SD card mode, SPI, 1-line SD mode, 4-line SD mode.

Values:

SD_MODE_SPI = 0x0
sd_card SPI

SD_MODE_1_LINE = 0x1
sd_card 1-line SD mode

SD_MODE_4_LINE = 0x4
sd_card 4-line SD mode

SD_MODE_8_LINE = 0x8
sd_card 8-line SD mode

SD_MODE_MAX

2.8.4 Spiffs Peripheral

Use this API to initialize, mount and unmount spiffs partition, see functions *periph_spiffs_init()*, `periph_spiffs_mount()` and `periph_spiffs_unmount()`. The data reading / writing is implemented in a separate API described in *SPIFFS 流*.

Application Example

Implementation of this API is demonstrated in audio_processing/pipeline_spiffs_amr_resample example.

API Reference

Header File

- esp_peripherals/include/periph_spiffs.h

Functions

esp_periph_handle_t **periph_spiffs_init** (*periph_spiffs_cfg_t *spiffs_config*)

Create the spiffs peripheral handle for esp_peripherals.

Note The handle created by this function will be automatically destroyed when `esp_periph_destroy` is called

Return The esp peripheral handle

Parameters

- `spiffs_config`: The spiffs configuration

bool **periph_spiffs_is_mounted** (*esp_periph_handle_t periph*)

Check if the SPIFFS is mounted or not.

Return SPIFFS mounted state

Parameters

- `[in] periph`: The periph

Structures

struct periph_spiffs_cfg_t

The SPIFFS Peripheral configuration.

Public Members

const char ***root**

Base path for vfs

const char ***partition_label**

Optional, label of SPIFFS partition to use. If set to NULL, first partition with subtype=spiffs will be used.

size_t **max_files**

Maximum number of files that could be open at the same time.

bool **format_if_mount_failed**

If true, it will format the file system if it fails to mount.

Enumerations

enum periph_spiffs_event_id_t

Peripheral spiffs event id.

Values:

SPIFFS_STATUS_UNKNOWN

No event

SPIFFS_STATUS_MOUNTED

SPIFFS mounted successfully

SPIFFS_STATUS_UNMOUNTED

SPIFFS unmounted successfully

SPIFFS_STATUS_MOUNT_ERROR

SPIFFS mount error

SPIFFS_STATUS_UNMOUNT_ERROR

SPIFFS unmount error

2.8.5 Console Peripheral

Console Peripheral is used to control the Audio application from the terminal screen. It provides 2 ways do execute command, one sends an event to esp_peripherals (for a command without parameters), another calls a callback function (need parameters). If there is a callback function, no event will be sent.

Please make sure that the lifetime of *periph_console_cmd_t* must be ensured during console operation, *periph_console_init()* only reference, does not make a copy.

Code example

Please refer to cli/main/console_example.c.

API Reference

Header File

- esp_peripherals/include/periph_console.h

Functions

esp_periph_handle_t **periph_console_init** (*periph_console_cfg_t* *config)

Initialize Console Peripheral.

Return The esp peripheral handle

Parameters

- `config`: The configuration

Structures

struct periph_console_cmd_t

Command structure.

Public Members

const char ***cmd**

Name of command, must be unique

int **id**

Command ID will be sent together when the command is matched

const char ***help**

Explanation of the command

console_cmd_callback_t **func**

Function callback for the command

struct periph_console_cfg_t

Console Peripheral configuration.

Public Members

int **command_num**

Total number of commands

const *periph_console_cmd_t* ***commands**

Pointer to array of commands

int **task_stack**

Console task stack, using default if the value is zero

int **task_prio**

Console task priority (based on freeRTOS priority), using default if the value is zero

int **buffer_size**

Size of console input buffer

const char ***prompt_string**

Console prompt string, using default CONSOLE_PROMPT_STRING if the pointer is NULL

Macros

CONSOLE_DEFAULT_TASK_PRIO

CONSOLE_DEFAULT_TASK_STACK

CONSOLE_DEFAULT_BUFFER_SIZE

CONSOLE_DEFAULT_PROMPT_STRING

Type Definitions

typedef esp_err_t (***console_cmd_callback_t**) (*esp_periph_handle_t* periph, int argc, char *argv[])

2.8.6 Touch Peripheral

Initialize ESP32 touchpad peripheral and retrieve information from the touch sensors.

Application Example

Implementation of this API is demonstrated in get-started/play_mp3_control example.

API Reference

Header File

- esp_peripherals/include/periph_touch.h

Functions

esp_periph_handle_t **periph_touch_init** (*periph_touch_cfg_t *config*)

Create the touch peripheral handle for esp_peripherals.

Note The handle was created by this function automatically destroy when `esp_periph_destroy` is called

Return The esp peripheral handle

Parameters

- `config`: The configuration

Structures

struct periph_touch_cfg_t

The Touch peripheral configuration.

Public Members

int **touch_mask**

Touch pad mask using for this Touch peripheral, ex: TOUCH_PAD_SEL0 | TOUCH_PAD_SEL1

int **tap_threshold_percent**

Tap threshold percent, Tap event will be determined if the percentage value is less than the non-touch value

int **long_tap_time_ms**

Long tap duration in milliseconds, default is 2000ms, PERIPH_TOUCH_LONG_TAP will be occurred if TAP and time hold longer than this value

Enumerations

enum esp_touch_pad_sel_t

Touch pad selection.

Values:

TOUCH_PAD_SEL0 = BIT(0)

TOUCH_PAD_SEL1 = BIT(1)

TOUCH_PAD_SEL2 = BIT(2)

TOUCH_PAD_SEL3 = BIT(3)

TOUCH_PAD_SEL4 = BIT(4)

TOUCH_PAD_SEL5 = BIT(5)

TOUCH_PAD_SEL6 = BIT(6)

TOUCH_PAD_SEL7 = BIT(7)

TOUCH_PAD_SEL8 = BIT(8)

TOUCH_PAD_SEL9 = BIT(9)

enum periph_touch_event_id_t

Peripheral touch event id.

Values:

PERIPH_TOUCH_UNCHANGE = 0

No event

PERIPH_TOUCH_TAP

When touch pad is tapped

PERIPH_TOUCH_RELEASE

When touch pad is released after tap

PERIPH_TOUCH_LONG_TAP

When touch pad is tapped and held after `long_tap_time_ms` time

PERIPH_TOUCH_LONG_RELEASE

When touch pad is released after long tap

2.8.7 Button Peripheral

To control application flow you may use buttons connected and read through the ESP32 GPIOs. This API provides functions to initialize specific GPIOs and obtain information on button events such as when it has been pressed, when released, when pressed for a long time and released after long press. To get information on a particular event, establish a callback function with `button_dev_add_tap_cb()` or `button_dev_add_press_cb()`.

Application Example

Implementation of this API is demonstrated in cloud_services/google_translate_device example.

API Reference

Header File

- esp_peripherals/include/periph_button.h

Functions

esp_periph_handle_t **periph_button_init** (*periph_button_cfg_t *but_cfg*)

Create the button peripheral handle for esp_peripherals.

Note The handle was created by this function automatically destroy when `esp_periph_destroy` is called

Return The esp peripheral handle

Parameters

- `but_cfg`: The but configuration

Structures

struct periph_button_cfg_t

The Button peripheral configuration.

Public Members

uint64_t **gpio_mask**

GPIO Mask using for this Button peripheral, it is BIT(GPIO_NUM), ex: GPIO_SEL_36 | GPIO_SEL_36

int **long_press_time_ms**

Long press duration in milliseconds, default is 2000ms

Enumerations

enum periph_button_event_id_t

Peripheral button event id.

Values:

PERIPH_BUTTON_UNCHANGE = 0

No event

PERIPH_BUTTON_PRESSED

When button is pressed

PERIPH_BUTTON_RELEASE

When button is released

PERIPH_BUTTON_LONG_PRESSED

When button is pressed and kept for more than `long_press_time_ms`

PERIPH_BUTTON_LONG_RELEASE

When button is released and event PERIPH_BUTTON_LONG_PRESSED happened

2.8.8 LED Peripheral

Blink or fade a LED connected to a GPIO with configurable On and Off times.

Application Examples

Implementation of this API is demonstrated in couple of examples:

- /cloud_services/google_translate_device
- /dueros

API Reference

Header File

- esp_peripherals/include/periph_led.h

Functions

esp_periph_handle_t **periph_led_init** (*periph_led_cfg_t* **config*)

Create the LED peripheral handle for esp_peripherals.

Note The handle was created by this function automatically destroy when `esp_periph_destroy` is called

Return The esp peripheral handle

Parameters

- `config`: The configuration

esp_err_t **periph_led_blink** (*esp_periph_handle_t* *periph*, int *gpio_num*, int *time_on_ms*, int *time_off_ms*, bool *fade*, int *loop*, *periph_led_idle_level_t* *level*)

Bink LED Peripheral, this function will automatically configure the gpio_num to control the LED, with `time_on_ms` as the time (in milliseconds) switch from OFF to ON (or ON if fade is disabled), and `time_off_ms` as the time (in milliseconds) switch from ON to OFF (or OFF if fade is disabled). When switching from ON -> OFF and vice versa, the loop decreases once, and will turn off the effect when the loop is 0. With a loop value less than 0, the LED effect will loop endlessly. PERIPH_LED_BLINK_FINISH events will be sent at each end of loop.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] periph`: The LED periph
- `[in] gpio_num`: The gpio number
- `[in] time_on_ms`: The time on milliseconds

- `[in] time_off_ms`: The time off milliseconds
- `[in] fade`: Fading enabled
- `[in] loop`: Loop
- `[in] level`: idle level

esp_err_t **periph_led_stop**(*esp_periph_handle_t* *periph*, int *gpio_num*)

Stop Blink the LED.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] periph`: The periph
- `[in] gpio_num`: The gpio number

Structures

struct periph_led_cfg_t

The LED peripheral configuration.

Public Members

ledc_mode_t **led_speed_mode**

LEDC speed speed_mode, high-speed mode or low-speed mode

ledc_timer_bit_t **led_duty_resolution**

LEDC channel duty resolution

ledc_timer_t **led_timer_num**

Select the timer source of channel (0 - 3)

uint32_t **led_freq_hz**

LEDC timer frequency (Hz)

int **gpio_num**

Optional, < 0 invalid gpio number

Enumerations

enum periph_led_event_id_t

Peripheral LED event id.

Values:

PERIPH_LED_UNCHANGE = 0

No event

PERIPH_LED_BLINK_FINISH

When LED blink is finished

enum periph_led_idle_level_t

Peripheral LED idle output level.

Values:

PERIPH_LED_IDLE_LEVEL_LOW

Low level output

PERIPH_LED_IDLE_LEVEL_HIGH

High level output

2.8.9 ADC Button Peripheral

Read status of buttons connected to an ADC input using a resistor ladder. Configuration provides for more than a singe ADC input to read several sets of buttons. For an example hardware implementation please refer to schematic of ESP32-LyraTD-MSC V2.2 Upper Board (PDF).

Application Examples

Implementation of this API is demonstrated in the following example:

- checks/check_board_buttons

API Reference

Header File

- esp_peripherals/include/periph_adc_button.h

Functions

esp_periph_handle_t **periph_adc_button_init** (*periph_adc_button_cfg_t *btn_cfg*)

Create the button peripheral handle for esp_peripherals.

Note The handle created by this function is automatically destroyed when esp_periph_destroy is called.

Return The esp peripheral handle.

Parameters

- `btn_cfg`: The button configuration.

Structures

struct periph_adc_button_cfg_t

The configuration of ADC Button.

Public Members

adc_arr_t ***arr**

An array with configuration of buttons

int **arr_size**

The array size

adc_btn_task_cfg_t **task_cfg**

Adc button task configuration

Macros

ADC_BUTTON_STACK_SIZE

ADC_BUTTON_TASK_PRIORITY

ADC_BUTTON_TASK_CORE_ID

PERIPH_ADC_BUTTON_DEFAULT_CONFIG()

ADC_DEFAULT_ARR()

ESP32 ADC1 channels and GPIO table ADC1_CHANNEL_0 - GPIO36 ADC1_CHANNEL_1 - GPIO37 ADC1_CHANNEL_2 - GPIO38 ADC1_CHANNEL_3 - GPIO39 ADC1_CHANNEL_4 - GPIO32 ADC1_CHANNEL_5 - GPIO33 ADC1_CHANNEL_6 - GPIO34 ADC1_CHANNEL_7 - GPIO35

Enumerations

enum periph_adc_button_event_id_t

Values:

PERIPH_ADC_BUTTON_IDLE = 0

PERIPH_ADC_BUTTON_PRESSED

PERIPH_ADC_BUTTON_RELEASE

PERIPH_ADC_BUTTON_LONG_PRESSED

PERIPH_ADC_BUTTON_LONG_RELEASE

2.8.10 LED Controller Peripheral

This peripheral is applicable to IS31Fl3216 chip that is a light LED controller with an audio modulation mode. It can store data of 8 Frames with internal RAM to play small animations automatically. You can also use it to control a number of LEDs connected to GPIOs. If you want to use the IS31Fl3216, see functions *periph_is31fl3216_init()*, *periph_is31fl3216_set_blink_pattern()*, *periph_is31fl3216_set_duty()*, *periph_is31fl3216_set_state()*.

Application Examples

Implementation of this API is demonstrated in checks/check_display_led example.

API Reference

Header File

- esp_peripherals/include/periph_is31fl3216.h

Functions

esp_periph_handle_t **periph_is31fl3216_init** (*periph_is31fl3216_cfg_t *is31fl3216_config*)

Initializate the is31fl3216.

Return

- ESP_OK Success
- ESP_FAIL Fail

Parameters

- `is31fl3216_config`:

esp_err_t **periph_is31fl3216_set_state**(*esp_periph_handle_t periph*, *periph_is31fl3216_state_t state*)

Set the state of all the channels.

Return

- ESP_OK Success
- ESP_FAIL Fail

Parameters

- `periph`: The is31fl3216 handle
- `state`: The state of all channels

esp_err_t **periph_is31fl3216_set_blink_pattern**(*esp_periph_handle_t periph*, uint16_t *blink_pattern*)

Set the current enable channels.

Return

- ESP_OK Success
- ESP_FAIL Fail

Parameters

- `periph`: The is31fl3216 handle
- `blink_pattern`: The bit pattern of enabled channels

esp_err_t **periph_is31fl3216_set_duty**(*esp_periph_handle_t periph*, uint8_t *index*, uint8_t *value*)

Set the duty of the channel.

Return

- ESP_OK Success
- ESP_FAIL Fail

Parameters

- `periph`: The is31fl3216 handle
- `index`: The channel number
- `value`: The value of the channel’s duty to be set

esp_err_t **periph_is31fl3216_set_duty_step**(*esp_periph_handle_t periph*, uint8_t *step*)

Set the duty step of flash.

Return

- ESP_OK Success
- ESP_FAIL Fail

Parameters

- `periph`: The is31fl3216 handle
- `step`: The step of flash

esp_err_t **periph_is31fl3216_set_interval**(*esp_periph_handle_t* *periph*, uint16_t *interval_ms*)

Set the internval time.

Return

- ESP_OK Success
- ESP_FAIL Fail

Parameters

- `periph`: The is31fl3216 handle
- `interval_ms`: Time of interval

esp_err_t **periph_is31fl3216_set_shift_mode**(*esp_periph_handle_t* *periph*, *periph_is31_shift_mode_t* *mode*)

Set the shift mode.

Return

- ESP_OK Success
- ESP_FAIL Fail

Parameters

- `periph`: The is31fl3216 handle
- `mode`: Mode of periph_is31_shift_mode_t

esp_err_t **periph_is31fl3216_set_light_on_num**(*esp_periph_handle_t* *periph*, uint16_t *light_on_num*, uint16_t *max_light_num*)

Set the light on numbers.

Return

- ESP_OK Success
- ESP_FAIL Fail

Parameters

- `periph`: The is31fl3216 handle
- `light_on_num`: Enabled led number
- `max_light_num`: Maximum led number

esp_err_t **periph_is31fl3216_set_act_time**(*esp_periph_handle_t* *periph*, uint16_t *act_ms*)

Set the action time.

Return

- ESP_OK Success
- ESP_FAIL Fail

Parameters

- `periph`: The is31fl3216 handle
- `act_ms`: Action time, unit is millisecond, 0 is infinite

Structures

struct periph_is31fl3216_cfg_t

The configuration of is31fl3216.

Public Members

uint32_t **duty**[**IS31FL3216_CH_NUM**]

An array of the is31fl3216's duty

uint16_t **is31fl3216_pattern**

Current enable channel

periph_is31fl3216_state_t **state**

The state of all the channels

Macros

IS31FL3216_CH_NUM

BLUE_LED_MAX_NUM

Enumerations

enum periph_is31fl3216_state_t

Values:

IS31FL3216_STATE_UNKNOWN

IS31FL3216_STATE_OFF

IS31FL3216_STATE_ON

IS31FL3216_STATE_FLASH

IS31FL3216_STATE_BY_AUDIO

IS31FL3216_STATE_SHIFT

enum periph_is31_shift_mode_t

Values:

PERIPH_IS31_SHIFT_MODE_UNKNOWN

PERIPH_IS31_SHIFT_MODE_ACC

accumulation mode

PERIPH_IS31_SHIFT_MODE_SINGLE

Name of Peripheral	ESP32-LyraT	ESP32-LyraTD-MSC
Wi-Fi	✓	✓
SD Card	✓	✓
Spiffs	✓	✓
Console	✓	✓
Touch	✓	
Button	✓	
LED	✓	
ADC Button		✓
LED Controller		✓

2.9 Abstraction Layer

2.9.1 Ring Buffer

Ringbuffer is designed in addition to use as a data buffer, also used to connect Audio Elements. Each Element that requests data from the Ringbuffer will block the task until the data is available. Or block the task when writing data and the Buffer is full. Of course, we can stop this block at any time.

图 7: Ring Buffer used in Audio Pipeline

Application Example

In most of ESP-ADF examples connecting of Elements with Ringbuffers is done "behind the scenes" by a function `audio_pipeline_link()`. To see this operation exposed check player/pipeline_sdcard_mp3_control example.

API Reference

Header File

- audio_pipeline/include/ringbuf.h

Functions

ringbuf_handle_t **rb_create** (int *block_size*, int *n_blocks*)

Create ringbuffer with total size = block_size * n_blocks.

Return ringbuf_handle_t

Parameters

- `[in] block_size`: Size of each block
- `[in] n_blocks`: Number of blocks

esp_err_t **rb_destroy** (*ringbuf_handle_t rb*)

Cleanup and free all memory created by ringbuf_handle_t.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] rb`: The Ringbuffer handle

esp_err_t **rb_abort** (*ringbuf_handle_t rb*)

Abort waiting until there is space for reading or writing of the ringbuffer.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] rb`: The Ringbuffer handle

esp_err_t **rb_reset** (*ringbuf_handle_t rb*)

Reset ringbuffer, clear all values as initial state.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] rb`: The Ringbuffer handle

esp_err_t **rb_reset_is_done_write** (*ringbuf_handle_t rb*)

Reset is_done_write flag.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] rb`: The Ringbuffer handle

int **rb_bytes_available** (*ringbuf_handle_t rb*)

Get total bytes available of Ringbuffer.

Return total bytes available

Parameters

- `[in] rb`: The Ringbuffer handle

int **rb_bytes_filled**(*ringbuf_handle_t rb*)

Get the number of bytes that have filled the ringbuffer.

Return The number of bytes that have filled the ringbuffer

Parameters

- `[in] rb`: The Ringbuffer handle

int **rb_get_size**(*ringbuf_handle_t rb*)

Get total size of Ringbuffer (in bytes)

Return total size of Ringbuffer

Parameters

- `[in] rb`: The Ringbuffer handle

int **rb_read**(*ringbuf_handle_t rb*, char **buf*, int *len*, TickType_t *ticks_to_wait*)

Read from Ringbuffer to `buf` with len and wait `tick_to_wait` ticks until enough bytes to read if the ringbuffer bytes available is less than `len`. If `buf` argument provided is `NULL`, then ringbuffer do pseudo reads by simply advancing pointers.

Return Number of bytes read

Parameters

- `[in] rb`: The Ringbuffer handle
- `buf`: The buffer pointer to read out data
- `[in] len`: The length request
- `[in] ticks_to_wait`: The ticks to wait

int **rb_write**(*ringbuf_handle_t rb*, char **buf*, int *len*, TickType_t *ticks_to_wait*)

Write to Ringbuffer from `buf` with `len` and wait `tick_to_wait` ticks until enough space to write if the ringbuffer space available is less than `len`

Return Number of bytes written

Parameters

- `[in] rb`: The Ringbuffer handle
- `buf`: The buffer
- `[in] len`: The length

- `[in] ticks_to_wait`: The ticks to wait

esp_err_t **rb_done_write**(*ringbuf_handle_t rb*)

Set status of writing to ringbuffer is done.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] rb`: The Ringbuffer handle

esp_err_t **rb_unblock_reader**(*ringbuf_handle_t rb*)

Unblock from rb_read.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] rb`: The Ringbuffer handle

esp_err_t **rb_set_reader_holder**(*ringbuf_handle_t rb*, void **holder*)

Set the owner of the ‘rb_read’.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] rb`: The Ringbuffer handle
- `[in] holder`: The owner of the ‘rb_read’

esp_err_t **rb_get_reader_holder**(*ringbuf_handle_t rb*, void ***holder*)

Get the owner of the ‘rb_read’.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] rb`: The Ringbuffer handle
- `[out] holder`: The owner of the ‘rb_read’

esp_err_t **rb_set_writer_holder**(*ringbuf_handle_t rb*, void **holder*)

Set the owner of the ‘rb_write’.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] rb`: The Ringbuffer handle
- `[in] holder`: The owner of the ‘rb_write’

esp_err_t **rb_get_writer_holder**(*ringbuf_handle_t rb*, void ***holder*)

Get the owner of the ‘rb_write’.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[in] rb`: The Ringbuffer handle
- `[out] holder`: The owner of the ‘rb_write’

Macros

RB_OK

RB_FAIL

RB_DONE

RB_ABORT

RB_TIMEOUT

Type Definitions

typedef struct ringbuf ***ringbuf_handle_t**

2.9.2 Audio HAL

Abstraction layer for audio board hardware, serves as an interface between the user application and the hardware driver for specific audio board like *ESP32 LyraT*.

The API provides data structures to configure sampling rates of ADC and DAC signal conversion, data bit widths, I2C stream parameters, and selection of signal channels connected to ADC and DAC. It also contains several specific functions to e.g. initialize the audio board, `audio_hal_init()`, control the volume, `audio_hal_get_volume()` and `audio_hal_set_volume()`.

API Reference

Header File

- audio_hal/include/audio_hal.h

Functions

audio_hal_handle_t **audio_hal_init**(*audio_hal_codec_config_t* **audio_hal_conf*, *audio_hal_func_t* **audio_hal_func*)

Initialize media codec driver.

Note If selected codec has already been installed, it' ll return the *audio_hal* handle.

Return int, 0success, othersfail

Parameters

- `audio_hal_conf`: Configure structure audio_hal_config_t
- `audio_hal_func`: Structure containing functions used to operate audio the codec chip

esp_err_t **audio_hal_deinit**(*audio_hal_handle_t audio_hal*)

Uninitialize media codec driver.

Return int, 0success, othersfail

Parameters

- `audio_hal`: reference function pointer for selected audio codec

esp_err_t **audio_hal_ctrl_codec**(*audio_hal_handle_t audio_hal*, *audio_hal_codec_mode_t mode*, *audio_hal_ctrl_t audio_hal_ctrl*)

Start/stop codec driver.

Return int, 0success, othersfail

Parameters

- `audio_hal`: reference function pointer for selected audio codec
- `mode`: select media hal codec mode either encode/decode/or both to start from audio_hal_codec_mode_t
- `audio_hal_ctrl`: select start stop state for specific mode

esp_err_t **audio_hal_codec_iface_config**(*audio_hal_handle_t audio_hal*, *audio_hal_codec_mode_t mode*, *audio_hal_codec_i2s_iface_t *iface*)

Set codec I2S interface samples rate & bit width and format either I2S or PCM/DSP.

Return

- 0 Success
- -1 Error

Parameters

- `audio_hal`: reference function pointer for selected audio codec
- `mode`: select media hal codec mode either encode/decode/or both to start from audio_hal_codec_mode_t
- `iface`: I2S sample rate (ex: 16000, 44100), I2S bit width (16, 24, 32),I2s format (I2S, PCM, DSP).

esp_err_t **audio_hal_set_mute**(*audio_hal_handle_t audio_hal*, bool *mute*)

Set voice mute. Enables or disables DAC mute of a codec.

Note `audio_hal_get_volume` will still give a non-zero number in mute state. It will be set to that number when speaker is unmuted.

Return int, 0success, othersfail

Parameters

- `audio_hal`: reference function pointer for selected audio codec
- `mute`: true/false. If true speaker will be muted and if false speaker will be unmuted.

esp_err_t **audio_hal_set_volume**(*audio_hal_handle_t audio_hal*, int *volume*)

Set voice volume.

Note if volume is 0, mute is enabled,range is 0-100.

Return int, 0success, othersfail

Parameters

- `audio_hal`: reference function pointer for selected audio codec
- `volume`: value of volume in percent(%)

esp_err_t **audio_hal_get_volume**(*audio_hal_handle_t audio_hal*, int **volume*)

get voice volume.

Note if volume is 0, mute is enabled, range is 0-100.

Return int, 0success, othersfail

Parameters

- `audio_hal`: reference function pointer for selected audio codec
- `volume`: value of volume in percent returned(%)

esp_err_t **audio_hal_enable_pa**(*audio_hal_handle_t audio_hal*, bool *enable*)

Enables or disables PA.

Return int, 0success, othersfail

Parameters

- `audio_hal`: reference function pointer for selected audio codec
- `enable`: true/false.

Structures

struct audio_hal_codec_i2s_iface_t

I2s interface configuration for audio codec chip.

Public Members

audio_hal_iface_mode_t **mode**

audio codec chip mode

audio_hal_iface_format_t **fmt**

I2S interface format

audio_hal_iface_samples_t **samples**

I2S interface samples per second

audio_hal_iface_bits_t **bits**

i2s interface number of bits per sample

struct audio_hal_codec_config_t

Configure media hal for initialization of audio codec chip.

Public Members

audio_hal_adc_input_t **adc_input**

set adc channel

audio_hal_dac_output_t **dac_output**

set dac channel

audio_hal_codec_mode_t **codec_mode**

select codec mode: adc, dac or both

audio_hal_codec_i2s_iface_t **i2s_iface**

set I2S interface configuration

struct audio_hal

Configuration of functions and variables used to operate audio codec chip.

Public Members

esp_err_t (***audio_codec_initialize**)(*audio_hal_codec_config_t* *codec_cfg)

initialize codec

esp_err_t (***audio_codec_deinitialize**)(void)

deinitialize codec

esp_err_t (***audio_codec_ctrl**)(*audio_hal_codec_mode_t* mode, *audio_hal_ctrl_t* ctrl_state)

control codec mode and state

esp_err_t (***audio_codec_config_iface**)(*audio_hal_codec_mode_t* mode, *audio_hal_codec_i2s_iface_t* *iface)

configure i2s interface

esp_err_t (***audio_codec_set_mute**)(bool mute)

set codec mute

esp_err_t (***audio_codec_set_volume**)(int volume)

set codec volume

esp_err_t (***audio_codec_get_volume**)(int *volume)

get codec volume

esp_err_t (***audio_codec_enable_pa**)(bool enable)

enable pa

xSemaphoreHandle **audio_hal_lock**

semaphore of codec

void ***handle**

handle of audio codec

Macros

AUDIO_HAL_VOL_DEFAULT

Type Definitions

typedef struct *audio_hal* ***audio_hal_handle_t**

typedef struct *audio_hal* **audio_hal_func_t**

Configuration of functions and variables used to operate audio codec chip.

Enumerations

enum audio_hal_codec_mode_t

Select media hal codec mode.

Values:

AUDIO_HAL_CODEC_MODE_ENCODE = 1

select adc

AUDIO_HAL_CODEC_MODE_DECODE

select dac

AUDIO_HAL_CODEC_MODE_BOTH

select both adc and dac

AUDIO_HAL_CODEC_MODE_LINE_IN

set adc channel

enum audio_hal_adc_input_t

Select adc channel for input mic signal.

Values:

AUDIO_HAL_ADC_INPUT_LINE1 = 0x00

mic input to adc channel 1

AUDIO_HAL_ADC_INPUT_LINE2

mic input to adc channel 2

AUDIO_HAL_ADC_INPUT_ALL

mic input to both channels of adc

AUDIO_HAL_ADC_INPUT_DIFFERENCE

mic input to adc difference channel

enum audio_hal_dac_output_t

Select channel for dac output.

Values:

AUDIO_HAL_DAC_OUTPUT_LINE1 = 0x00

dac output signal to channel 1

AUDIO_HAL_DAC_OUTPUT_LINE2

dac output signal to channel 2

AUDIO_HAL_DAC_OUTPUT_ALL

dac output signal to both channels

enum audio_hal_ctrl_t

Select operating mode i.e. start or stop for audio codec chip.

Values:

AUDIO_HAL_CTRL_STOP = 0x00

set stop mode

AUDIO_HAL_CTRL_START = 0x01

set start mode

enum audio_hal_iface_mode_t

Select I2S interface operating mode i.e. master or slave for audio codec chip.

Values:

AUDIO_HAL_MODE_SLAVE = 0x00

set slave mode

AUDIO_HAL_MODE_MASTER = 0x01

set master mode

enum audio_hal_iface_samples_t

Select I2S interface samples per second.

Values:

AUDIO_HAL_08K_SAMPLES

set to 8k samples per second

AUDIO_HAL_11K_SAMPLES

set to 11.025k samples per second

AUDIO_HAL_16K_SAMPLES

set to 16k samples in per second

AUDIO_HAL_22K_SAMPLES
set to 22.050k samples per second

AUDIO_HAL_24K_SAMPLES
set to 24k samples in per second

AUDIO_HAL_32K_SAMPLES
set to 32k samples in per second

AUDIO_HAL_44K_SAMPLES
set to 44.1k samples per second

AUDIO_HAL_48K_SAMPLES
set to 48k samples per second

enum audio_hal_iface_bits_t
Select I2S interface number of bits per sample.

Values:

AUDIO_HAL_BIT_LENGTH_16BITS = 1
set 16 bits per sample

AUDIO_HAL_BIT_LENGTH_24BITS
set 24 bits per sample

AUDIO_HAL_BIT_LENGTH_32BITS
set 32 bits per sample

enum audio_hal_iface_format_t
Select I2S interface format for audio codec chip.

Values:

AUDIO_HAL_I2S_NORMAL = 0
set normal I2S format

AUDIO_HAL_I2S_LEFT
set all left format

AUDIO_HAL_I2S_RIGHT
set all right format

AUDIO_HAL_I2S_DSP
set dsp/pcm format

2.9.3 ES8388 Driver

Driver for ES8388 codec chip used in *ESP32 LyraT* audio board.

API Reference

Header File

- audio_hal/driver/es8388/es8388.h

Functions

esp_err_t **es8388_init**(*audio_hal_codec_config_t* *cfg)

Initialize ES8388 codec chip.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `cfg`: configuration of ES8388

esp_err_t **es8388_deinit**(void)

Deinitialize ES8388 codec chip.

Return

- ESP_OK
- ESP_FAIL

esp_err_t **es8388_config_fmt**(es_module_t *mod*, es_i2s_fmt_t *cfg*)

Configure ES8388 I2S format.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `mod`: set ADC or DAC or both
- `cfg`: ES8388 I2S format

esp_err_t **es8388_i2s_config_clock**(es_i2s_clock_t *cfg*)

Configure I2s clock in MSATER mode.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `cfg`: set bits clock and WS clock

esp_err_t **es8388_set_bits_per_sample**(es_module_t *mode*, es_bits_length_t *bit_per_sample*)

Configure ES8388 data sample bits.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `mode`: set ADC or DAC or both
- `bit_per_sample`: bit number of per sample

esp_err_t **es8388_start**(es_module_t *mode*)

Start ES8388 codec chip.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `mode`: set ADC or DAC or both

esp_err_t **es8388_stop**(es_module_t *mode*)

Stop ES8388 codec chip.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `mode`: set ADC or DAC or both

esp_err_t **es8388_set_voice_volume**(int *volume*)

Set voice volume.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `volume`: voice volume (0~100)

esp_err_t **es8388_get_voice_volume**(int **volume*)

Get voice volume.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[out] *volume`: voice volume (0~100)

esp_err_t **es8388_set_voice_mute**(bool *enable*)

Configure ES8388 DAC mute or not. Basically you can use this function to mute the output or unmute.

Return

- ESP_FAIL Parameter error
- ESP_OK Success

Parameters

- `enable`: enable(1) or disable(0)

esp_err_t **es8388_get_voice_mute**(void)

Get ES8388 DAC mute status.

Return

- ESP_FAIL Parameter error
- ESP_OK Success

esp_err_t **es8388_set_mic_gain**(es_mic_gain_t *gain*)

Set ES8388 mic gain.

Return

- ESP_FAIL Parameter error
- ESP_OK Success

Parameters

- `gain`: db of mic gain

esp_err_t **es8388_config_adc_input**(es_adc_input_t *input*)

Set ES8388 adc input mode.

Return

- ESP_FAIL Parameter error
- ESP_OK Success

Parameters

- `input`: adc input mode

esp_err_t **es8388_config_dac_output**(es_dac_output_t *output*)

Set ES8388 dac output mode.

Return

- ESP_FAIL Parameter error
- ESP_OK Success

Parameters

- `output`: dac output mode

esp_err_t **es8388_write_reg**(uint8_t *reg_add*, uint8_t *data*)

Write ES8388 register.

Return

- ESP_FAIL Parameter error
- ESP_OK Success

Parameters

- `reg_add`: address of register
- `data`: data of register

void **es8388_read_all**()

Print all ES8388 registers.

Return

- void

esp_err_t **es8388_config_i2s** (*audio_hal_codec_mode_t* *mode*, *audio_hal_codec_i2s_iface_t* **iface*)

Configure ES8388 codec mode and I2S interface.

Return

- ESP_FAIL Parameter error
- ESP_OK Success

Parameters

- `mode`: codec mode
- `iface`: I2S config

esp_err_t **es8388_ctrl_state** (*audio_hal_codec_mode_t* *mode*, *audio_hal_ctrl_t* *ctrl_state*)

Control ES8388 codec chip.

Return

- ESP_FAIL Parameter error
- ESP_OK Success

Parameters

- `mode`: codec mode
- `ctrl_state`: start or stop decode or encode progress

esp_err_t **es8388_pa_power** (bool *enable*)

Set ES8388 PA power.

Return

- ESP_ERR_INVALID_ARG
- ESP_OK

Parameters

- `enable`: true for enable PA power, false for disable PA power

Macros

ES8388_ADDR
0x22:CE=1;0x20:CE=0

ES8388_CONTROL1

ES8388_CONTROL2

ES8388_CHIPPOWER

ES8388_ADCPOWER

ES8388_DACPOWER

ES8388_CHIPLOPOW1

ES8388_CHIPLOPOW2

ES8388_ANAVOLMANAG

ES8388_MASTERMODE

ES8388_ADCCONTROL1

ES8388_ADCCONTROL2

ES8388_ADCCONTROL3

ES8388_ADCCONTROL4

ES8388_ADCCONTROL5

ES8388_ADCCONTROL6

ES8388_ADCCONTROL7

ES8388_ADCCONTROL8

ES8388_ADCCONTROL9

ES8388_ADCCONTROL10

ES8388_ADCCONTROL11

ES8388_ADCCONTROL12

ES8388_ADCCONTROL13

ES8388_ADCCONTROL14

ES8388_DACCONTROL1

ES8388_DACCONTROL2

ES8388_DACCONTROL3

ES8388_DACCONTROL4

ES8388_DACCONTROL5

ES8388_DACCONTROL6

ES8388_DACCONTROL7

ES8388_DACCONTROL8

ES8388_DACCONTROL9

ES8388_DACCONTROL10

ES8388_DACCONTROL11

ES8388_DACCONTROL12

ES8388_DACCONTROL13

ES8388_DACCONTROL14

ES8388_DACCONTROL15

ES8388_DACCONTROL16

ES8388_DACCONTROL17

ES8388_DACCONTROL18

ES8388_DACCONTROL19

ES8388_DACCONTROL20

ES8388_DACCONTROL21

ES8388_DACCONTROL22

ES8388_DACCONTROL23

ES8388_DACCONTROL24

ES8388_DACCONTROL25

ES8388_DACCONTROL26

ES8388_DACCONTROL27

ES8388_DACCONTROL28

ES8388_DACCONTROL29

ES8388_DACCONTROL30

2.9.4 ES8374 Driver

Driver for ES8374 codec chip.

API Reference

Header File

- audio_hal/driver/es8374/es8374.h

Functions

esp_err_t **es8374_codec_init**(*audio_hal_codec_config_t* *cfg)

Initialize ES8374 codec chip.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `cfg`: configuration of ES8374

esp_err_t **es8374_codec_deinit**(void)

Deinitialize ES8374 codec chip.

Return

- ESP_OK
- ESP_FAIL

esp_err_t **es8374_config_fmt**(es_module_t *mode*, es_i2s_fmt_t *fmt*)

Configure ES8374 I2S format.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `mode`: set ADC or DAC or both
- `fmt`: ES8374 I2S format

esp_err_t **es8374_i2s_config_clock**(es_i2s_clock_t *cfg*)

Configure I2S clock in MSATER mode.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `cfg`: set bits clock and WS clock

esp_err_t **es8374_set_bits_per_sample**(es_module_t *mode*, es_bits_length_t *bit_per_sample*)

Configure ES8374 data sample bits.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `mode`: set ADC or DAC or both
- `bit_per_sample`: bit number of per sample

esp_err_t **es8374_start**(es_module_t *mode*)

Start ES8374 codec chip.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `mode`: set ADC or DAC or both

esp_err_t **es8374_stop**(es_module_t *mode*)

Stop ES8374 codec chip.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `mode`: set ADC or DAC or both

esp_err_t **es8374_codec_set_voice_volume**(int *volume*)

Set voice volume.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `volume`: voice volume (0~100)

esp_err_t **es8374_codec_get_voice_volume**(int **volume*)

Get voice volume.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[out] *volume`: voice volume (0~100)

esp_err_t **es8374_set_voice_mute**(bool *enable*)

Mute or unmute ES8374 DAC. Basically you can use this function to mute or unmute the output.

Return

- ESP_FAIL Parameter error
- ESP_OK Success

Parameters

- `enable`: mute(1) or unmute(0)

esp_err_t **es8374_get_voice_mute**(void)

Get ES8374 DAC mute status.

Return

- ESP_FAIL
- ESP_OK

esp_err_t **es8374_set_mic_gain**(es_mic_gain_t *gain*)

Set ES8374 mic gain.

Return

- ESP_FAIL Parameter error
- ESP_OK Success

Parameters

- `gain`: db of mic gain

esp_err_t **es8374_config_adc_input**(es_adc_input_t *input*)

Set ES8374 ADC input mode.

Return

- ESP_FAIL Parameter error
- ESP_OK Success

Parameters

- `input`: adc input mode

esp_err_t **es8374_config_dac_output**(es_dac_output_t *output*)

Set ES8374 DAC output mode.

Return

- ESP_FAIL Parameter error
- ESP_OK Success

Parameters

- `output`: dac output mode

esp_err_t **es8374_write_reg**(uint8_t *reg_add*, uint8_t *data*)

Write ES8374 register.

Return

- ESP_FAIL Parameter error
- ESP_OK Success

Parameters

- `reg_add`: address of register
- `data`: data of register

void **es8374_read_all**()

Print all ES8374 registers.

Return

- void

esp_err_t **es8374_codec_config_i2s**(*audio_hal_codec_mode_t mode*, *audio_hal_codec_i2s_iface_t *iface*)

Configure ES8374 codec mode and I2S interface.

Return

- ESP_FAIL Parameter error
- ESP_OK Success

Parameters

- `mode`: codec mode
- `iface`: I2S config

esp_err_t **es8374_codec_ctrl_state**(*audio_hal_codec_mode_t mode*, *audio_hal_ctrl_t ctrl_state*)

Control ES8374 codec chip.

Return

- ESP_FAIL Parameter error
- ESP_OK Success

Parameters

- `mode`: codec mode
- `ctrl_state`: start or stop decode or encode progress

esp_err_t **es8374_pa_power**(bool *enable*)

Set ES8374 PA power.

Return

- ESP_ERR_INVALID_ARG
- ESP_OK

Parameters

- `enable`: true for enable PA power, false for disable PA power

Macros

ES8374_ADDR

2.9.5 ZL38063 Driver

Driver for ZL38063 codec chip used in *ESP32-LyraTD-MSC* audio board.

API Reference

Header File

- audio_hal/driver/zl38063/zl38063.h

Functions

esp_err_t **zl38063_codec_init**(*audio_hal_codec_config_t* *cfg)

Initialize ZL38063 chip.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `cfg`: configuration of ZL38063

esp_err_t **zl38063_codec_deinit**(void)

Deinitialize ZL38063 chip.

Return

- ESP_OK
- ESP_FAIL

esp_err_t **zl38063_codec_ctrl_state**(*audio_hal_codec_mode_t* mode, *audio_hal_ctrl_t* ctrl_state)

Control ZL38063 chip.

The functions zl38063_ctrl_state and zl38063_config_i2s are not used by this driver. They are kept here to maintain the uniformity and convenience of the interface of the ADF project. These settings for zl38063 are burned in firmware and configuration files. Default i2s configuration: 48000Hz, 16bit, Left-Right channels. Use resampling to be compatible with different file types.

Return

- ESP_FAIL Parameter error
- ESP_OK Success

Parameters

- `mode`: codec mode
- `ctrl_state`: start or stop decode or encode progress

esp_err_t **zl38063_codec_config_i2s**(*audio_hal_codec_mode_t* *mode*, *audio_hal_codec_i2s_iface_t* **iface*)

Configure ZL38063 codec mode and I2S interface.

Return

- ESP_FAIL Parameter error
- ESP_OK Success

Parameters

- `mode`: codec mode
- `iface`: I2S config

esp_err_t **zl38063_codec_set_voice_mute**(bool *mute*)

mute or unmute the codec

Return

- ESP_OK
- ESP_FAIL

Parameters

- `mute`: true, false

esp_err_t **zl38063_codec_set_voice_volume**(int *volume*)

Set voice volume.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `volume`: voice volume (0~100)

esp_err_t **zl38063_codec_get_voice_volume**(int *volume)

Get voice volume.

Return

- ESP_OK
- ESP_FAIL

Parameters

- `[out] *volume`: voice volume (0~100)

2.10 Configuration Options

Compile-time configuration options specific to ESP-ADF.

2.10.1 Audio Codec Device Configuration

CODEC_ES8311_SUPPORT

Support ES8311 Codec Chip

Found in: Component config > Audio Codec Device Configuration

Enable this option to support codec ES8311.

CODEC_ES7210_SUPPORT

Support ES7210 Codec Chip

Found in: Component config > Audio Codec Device Configuration

Enable this option to support codec ES7210.

CODEC_ES7243_SUPPORT

Support ES7243 Codec Chip

Found in: Component config > Audio Codec Device Configuration

Enable this option to support codec ES7243.

CODEC_ES7243E_SUPPORT

Support ES7243E Codec Chip

Found in: Component config > Audio Codec Device Configuration

Enable this option to support codec ES7243E.

CODEC_ES8156_SUPPORT

Support ES8156 Codec Chip

Found in: Component config > Audio Codec Device Configuration

Enable this option to support codec ES8156.

CODEC_AW88298_SUPPORT

Support AW88298 Codec Chip

Found in: Component config > Audio Codec Device Configuration

Enable this option to support codec AW88298.

CODEC_ES8374_SUPPORT

Support ES8374 Codec Chip

Found in: Component config > Audio Codec Device Configuration

Enable this option to support codec ES8374.

CODEC_ES8388_SUPPORT

Support ES8388 Codec Chip

Found in: Component config > Audio Codec Device Configuration

Enable this option to support codec ES8388.

CODEC_TAS5805M_SUPPORT

Support TAS5805M Codec Chip

Found in: Component config > Audio Codec Device Configuration

Enable this option to support codec TAS5805M.

CODEC_ZL38063_SUPPORT

Support ZL38063 Codec Chip

Found in: Component config > Audio Codec Device Configuration

Enable this option to support codec ZL38063. ZL38063 firmware only support xtensa, don't enable for RISC-V IC.

2.10.2 Audio HAL

AUDIO_BOARD

Audio board

Found in: Audio HAL

Select an audio board to use with the ESP-ADF

Available options:

- AUDIO_BOARD_CUSTOM
- ESP_LYRAT_V4_3_BOARD
- ESP_LYRAT_V4_2_BOARD
- ESP_LYRATD_MSC_V2_1_BOARD
- ESP_LYRATD_MSC_V2_2_BOARD
- ESP_LYRAT_MINI_V1_1_BOARD
- ESP32_KORVO_DU1906_BOARD
- ESP32_S2_KALUGA_1_V1_2_BOARD
- ESP32_S3_KORVO2_V3_BOARD
- ESP32_S3_KORVO2L_V1_BOARD
- ESP32_S3_BOX_LITE_BOARD
- ESP32_S3_BOX_BOARD
- ESP32_S3_BOX_3_BOARD

- ESP32_C3_LYRA_V2_BOARD
- ESP32_C6_DEVKIT_BOARD

ESP32_KORVO_DU1906_DAC

ESP32 KORVO DU1906 Board DAC chip

Found in: Audio HAL

Select DAC chip to use on ESP32_KORVO_DU1906 board

Available options:

- ESP32_KORVO_DU1906_DAC_TAS5805M
- ESP32_KORVO_DU1906_DAC_ES7148

ESP32_KORVO_DU1906_ADC

ESP32 KORVO DU1906 Board ADC chip

Found in: Audio HAL

Select ADC chip to use on ESP32_KORVO_DU1906 board

Available options:

- ESP32_KORVO_DU1906_ADC_ES7243

2.10.3 ADF Features

ESP_DISPATCHER_DELEGATE_TASK_CORE

Delegation task core

Found in: ADF Features

Pinned delegate task to core 0 or core 1.

ESP_DISPATCHER_DELEGATE_TASK_PRIO

Delegate task’s prio

Found in: ADF Features

The delegate task’s prio.

ESP_DISPATCHER_DELEGATE_STACK_SIZE

Delegate task’s stack size

Found in: ADF Features

The delegate task’s stack is located in DRAM, modify this size to make sure all the needed operation can be run success in the it.

2.10.4 DuerOS Service

DUEROS_GEN_PROFILE

Generate the nvs partition which include the profile of dueros

Found in: Component config > DuerOS Service

DUEROS_FLASH_PROFILE

Flash the generated nvs partition

Found in: Component config > DuerOS Service

DUEROS_DEVICE_NAME

Duer device name

Found in: Component config > DuerOS Service

2.10.5 ADF Library Configuration

MEDIA_PROTOCOL_LIB_ENABLE

Enable Media Protocol Library

Found in: ADF Library Configuration

MEDIA_LIB_MEM_AUTO_TRACE

Support trace memory automatically after media_lib_sal init

Found in: ADF Library Configuration

MEDIA_LIB_MEM_TRACE_DEPTH

Memory trace stack depth

Found in: ADF Library Configuration

Set memory trace depth

MEDIA_LIB_MEM_TRACE_NUM

Memory trace number

Found in: ADF Library Configuration

Set memory trace number

MEDIA_LIB_MEM_TRACE_MODULE

Trace for module memory usage

Found in: ADF Library Configuration

MEDIA_LIB_MEM_TRACE_LEAKAGE

Trace for memory leakage

Found in: ADF Library Configuration

MEDIA_LIB_MEM_TRACE_SAVE_HISTORY

Trace to save memory history

Found in: ADF Library Configuration

MEDIA_LIB_MEM_SAVE_CACHE_SIZE

Cache buffer size to store save history

Found in: ADF Library Configuration

Set cache size for memory history

MEDIA_LIB_MEM_TRACE_SAVE_PATH

Memory trace save path

Found in: ADF Library Configuration

Set memory trace save path

CHAPTER 3

Design Guide

The ESP32 is a powerful chip well positioned as a MCU of the audio projects. This section is intended to provide guidance on process of designing an audio project with the ESP32 inside.

3.1 项目设计

[English]

在设计能够处理音频信号或音频数据的项目时，我们通常会考虑使用以下组件：

输入：

- **模拟信号输入**：连接麦克风等设备
- **存储媒体**：如 microSD 卡，可读取其中的音频文件
- **Wi-Fi 接口**：从互联网获取音频数据流
- **蓝牙接口**：从蓝牙耳机等设备获取音频数据流
- **I2S 接口**：从编解码芯片获取音频数据流
- **以太网接口**：从互联网获取音频数据流
- **芯片内部 flash**：播放音频样本
- **用户交互**：按钮或其他提供用户输入的方式

输出：

- **模拟信号输出**：连接耳机、放大器或扬声器
- **存储媒体**：如 microSD 卡，可通过录音写入音频文件
- **Wi-Fi 接口**：将音频数据流发送到互联网
- **蓝牙接口**：将音频数据流传输到蓝牙耳机等设备
- **I2S 接口**：将数据流传输到编解码器芯片
- **以太网接口**：将音频数据流传输到互联网
- **芯片内部 flash**：存储录音
- **用户交互**：如显示器、LED 或 振动反馈

主处理器：

微控制器或计算机，可从输入读取数据、处理数据（例如编码/编码）、输出数据。

3.1.1 选择项目类型

ESP 系列芯片（包括 ESP32、ESP32-S2、ESP32-S3 等）具有上述所有功能，或者说能够支持这些功能（如驱动以太网 PHY）。鉴于 ESP 芯片的成本较低，以及可免费使用的 ESP-ADF 软件开发平台，我们能够以极低的价格开发具有最少附加组件的音频项目。

基于应用程序、所需的功能和性能，我们可以考虑以下两种项目类型。

- **基础项目**：具有最少的附加组件，前提是使用板载的 I2S 、PDM 接口或 DAC，且不需要输出特别高质量的音频。
- **典型项目**：带有外部编解码器芯片和功率放大器，可输出高质量音频，提供多个输入/输出选项。

通过添加或删除上述项目的功能/组件，可以开发出其他不同的项目，示例请见下文。

3.1.2 基础项目

ESP 芯片拥有多个外设，可借助 I2S、PDM 或 DAC 接口可开发出一个基础项目。我们可以通过数字麦克风输入语音信号，构建一个可以与云服务通信的语音命令控制的基础项目。

如果可接受 8 位的数据输出，我们可使用两个板载 DAC 实现另一个基础项目，播放联网广播节目。

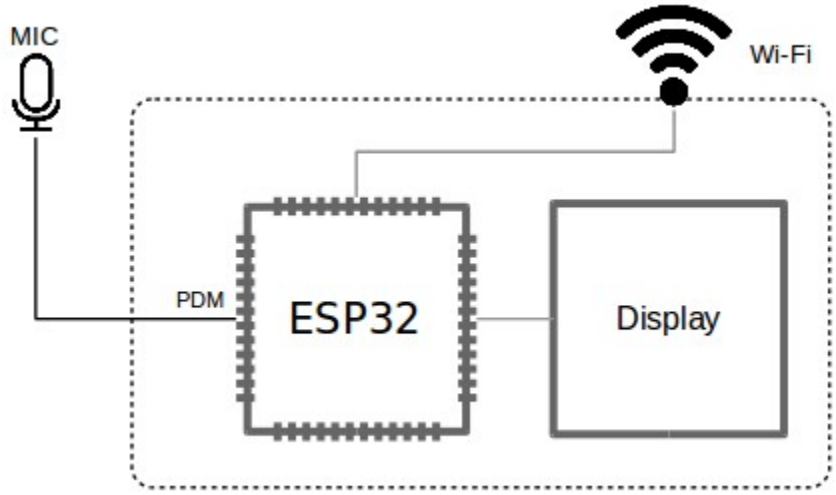


图 1: 音频项目示例 - 向云服务发送语音命令

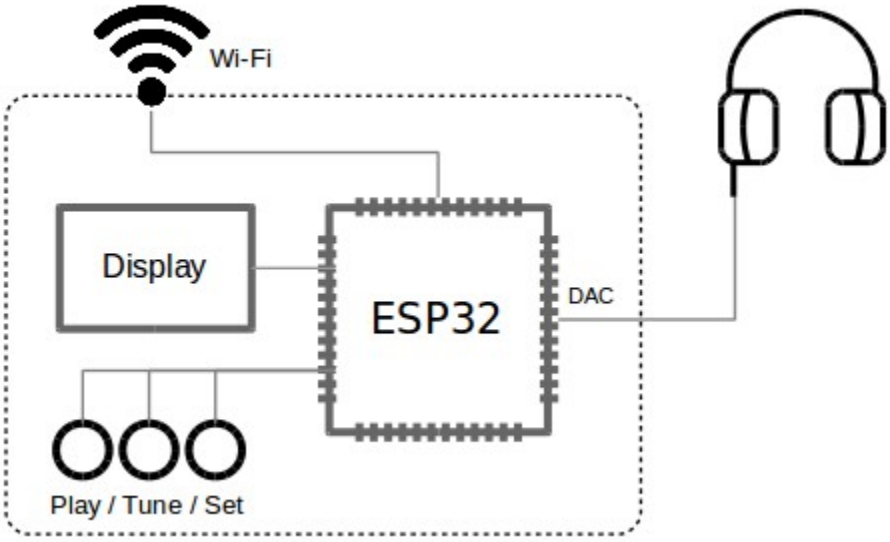


图 2: 音频项目示例 - 联网广播播放器

3.1.3 典型项目

如果需要更好的音频质量和更多的接口选项，可使用外部 I2S 编解码器来完成所有模拟输入和输出信号的处理。不同类型的编解码器芯片可提供不同的额外功能，如音频输入信号前置放大器、耳机输出放大器、多个模拟输入和输出、音效处理等。I2S 是音频编解码器芯片接口的行业标准，通常用于高速、连续传输音频数据。为了优化音频数据处理的性能，可能需要额外的内存。对于这种情况，请考虑使用集成 8 MB PSRAM 和 ESP32 芯片的 ESP32-WROVER-E 模组。

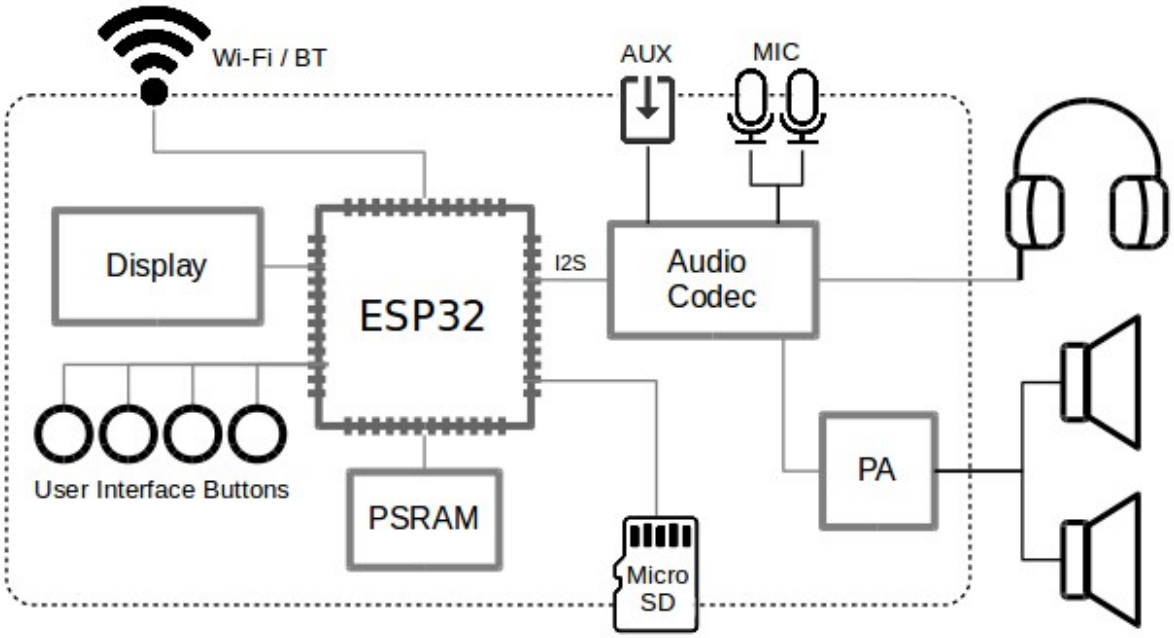


图 3: 典型音频项目示例

ESP-ADF 主要支持使用编解码器芯片的项目，如*ESP32 LyraT* 开发板与软件之间的交互由音频 HAL 和驱动程序完成，开发板使用的编解码芯片是 ES8388。可以通过提供不同的驱动程序来支持具有不同编解码器芯片的开发板。

3.2 Design Considerations

Depending on the audio data format, that may be lossless, lossy or compressed, e.g. WAV, MP3 or FLAC and the quality expressed in sampling rate and bitrate, the project will require different resources: memory, storage space, input / output throughput and the processing power. The resources will also depend on the project type and features discussed in 项目设计.

This section describes capacity and performance of ESP32 system resources that should be considered when designing an audio project to meet required data format, audio quality and functionality.

3.2.1 Memory

The spare internal Data-RAM is about 290kB with "hello_world" example. For audio system this may be insufficient, and therefore the ESP32 incorporates the ability to use up to 4MB of external SPI RAM (i.e. PSRAM) memory. The external memory is incorporated in the memory map and is, within certain restrictions, usable in the same way internal Data-RAM is.

Refer to External SPI-connected RAM section in IDF documenation for details, especially pay attention to its Restrictions section which is very important.

To be able to use the PSRAM, if installed on your board, it should be enabled in menucofig under *Component config* > *ESP32-specific* > *SPI RAM config*. The option *CONFIG_SPIRAM_CACHE_WORKAROUND*, set by default in the same menu, should be kept enabled.

注解: Bluetooth and Wi-Fi can not coexist without PSRAM because it will not leave enough memory for an audio application.

Optimization of Internal RAM and Use of PSRAM

Internal RAM is more valuable asset since there are some restrictions on PSRAM. Here are some tips for optimizing internal RAM.

- If PSRAM is in use, set all the static buffer to minimum value in *Component config* > *Wi-Fi*; if PSRAM is not used then dynamic buffer should be selected to save memory. Refer to Wi-Fi Buffer Usage section in IDF documentation for details.
- If PSRAM and BT are used, then *CONFIG_BT_ALLOCATION_FROM_SPIRAM_FIRST* and *CONFIG_BT_BLE_DYNAMIC_ENV_MEMORY* should be set as "yes" under *Component config* > *Bluetooth* > *Bluedroid Enable*, to allocate more of 40kB memory to PSRAM
- If PSRAM and Wi-Fi are used, then *CONFIG_WIFI_LWIP_ALLOCATION_FROM_SPIRAM_FIRST* should be set as "yes" under *Component config* > *ESP32-specific* > *SPI RAM config*, to allocate some memory to PSRAM
- Set *CONFIG_WL_SECTOR_SIZE* as 512 in *Component config* > *Wear Levelling*

注解: The smaller the size of sector be, the slower the Write / Read speed will be, and vice versa, but only 512 and 4096 are supported.

- Call `char *buf = heap_caps_malloc(1024 * 10, MALLOC_CAP_SPIRAM | MALLOC_CAP_8BIT)` instead of `malloc(1024 * 10)` to use PSRAM, and call `char *buf = heap_caps_malloc(512, MALLOC_CAP_INTERNAL | MALLOC_CAP_8BIT)` to use internal RAM.
- Not relying on `malloc()` to automatically allocate PSRAM allows to make a full control of the memory. By avoiding the use of the internal RAM by other `malloc()` calls, you can reserve more memory for high-efficiency

usage and task stack since PSRAM cannot be used as task stack memory.

- The task stack will always be allocated at internal RAM. On the other hand you can use of the `xTaskCreateStatic()` function that allows to create tasks with stack on PSRAM (see options in PSRAM and FreeRTOS menuconfig), but pay attention to its help information.

重要: Don't use ROM code in `xTaskCreateStatic` task: The ROM code itself is linked in `components/esp32/ld/esp32.rom.ld`. However, you also need to consider other pieces of code that *call* ROM functions, as well as the code that is not recompiled against the *CONFIG_SPIRAM_CACHE_WORKAROUND* patch, like the Wi-Fi and Bluetooth libraries. In general, we advise using this only in threads that do not call any IDF libraries (including libc), doing only calculations and using FreeRTOS primitives to talk to other threads.

Memory Usage by Component Overview

Below is a table that contains ESP-ADF components and their memory usage. Choose the components needed and find out how much internal RAM is left. The table is divided into two parts, when PSRAM is used or not. If PSRAM (external RAM) is in use, then some of the memory will be allocated at PSRAM automatically.

The initial spare internal RAM is 290kB.

Component	Internal RAM Required	
	PSRAM not used	With PSRAM
Wi-Fi [1]	50kB+	50kB+
Bluetooth	140kB (50kB if only BLE needed)	95kB (50kB if only BLE needed)
Flash Card [2]	12kB+	12kB+
I2S [3]	Configurable, 8kB for reference	Configurable, 8kB for reference
RingBuffer [4]	Configurable, 30kB for reference	0kB, all moved into PSRAM

Notes to the table above

1. According to the Wi-Fi menuconfig each Tx and Rx buffer occupies 1.6kB internal RAM. The value of 50kB RAM is assuming use of 5 Rx static buffers and 6 Tx static buffers. If PSRAM is not in use, then the "Type of WiFi Tx Buffer" option should be set as *DYNAMIC* in order to save RAM, in this case, the RAM usage will be far less than 50kB, but programmer should keep at least 50kB available for the Wi-Fi to be able to transmit the data. **[Internal RAM only]**
2. Taking ESP32-LyraT V4.3 as an example, depending on value of *SDCARD_OPEN_FILE_NUM_MAX* in audio_board/lyrat_v4_3/board_def.h, that is then used in `sdcard_mount()` function, the RAM needed will increase with a greater number of maximum open files. 12kB is the RAM needed with 5 max files and 512 bytes *CONFIG_WL_SECTOR_SIZE*. **[Internal RAM only]**
3. Depending on configuration settings of the I2S stream, refer to audio_stream/include/i2s_stream.h and audio_stream/i2s_stream.c. **[Internal RAM only]**

4. Depending on configuration setting of the Ringbuffer, refer to *DEFAULT_PIPELINE_RINGBUF_SIZE* in audio_pipeline/include/audio_pipeline.h or user setting, if the buffer is created with e.g. `rb_create()`.

3.2.2 System Settings

The following settings are recommended to achieve a high Wi-Fi performance in an audio project.

注解： Use ESP32 modules and boards from reputable vendors that put attention to product design, component selection and product testing. This is to have confidence of receiving well designed boards with calibrated RF.

- Set these following options in menuconfig.
  - Flash SPI mode as QIO
  - Flash SPI speed as 80MHz
  - CPU frequency as 240MHz
  - Set *Default receive window size* as 5 times greater than *Maximum Segment Size* in *Component config* > *LWIP* > *TCP*
- If external antenna is used, then set *PHY_RF_CAL_PARTIAL* as *PHY_RF_CAL_FULL* in ‘ ’ esp-idf/components/esp32/phy_init.c’ ’

3.3 Software Design

Espressif audio framework project.

3.3.1 Features

1. All of Streams and Codecs based on audio element.
2. All events based on queue.
3. Audio pipeline supports dynamic combination.
4. Audio pipeline supports multiple elements.
5. Pipeline Support functionality plug-in.
6. Audio common peripherals support work in the one task.
7. Support post-event mechanism in peripherals.
8. Support high level audio play API based on element and audio pipeline.
9. Audio high level interface supports dynamic adding of codec library.

10. Audio high level interface supports dynamic adding of input and output stream.
11. ESP audio supports multiple audio pipelines.

3.3.2 Design Components

Five basic components are - Audio Element, Audio Event, Audio Pipeline, ESP peripherals, ESP audio

Audio Element

Example

Please refer to audio_stream/fatfs_stream.c for an example of using an audio element.

Audio Event

Example

Please refer to player/pipeline_http_mp3/main/play_http_mp3_example.c for an example of using an audio event.

Audio Pipeline

Example

Please refer to player/pipeline_play_sdcard_music/main/play_sdcard_music_example.c for an example of linking elements into an audio pipeline.

Audio Peripheral

Example

```
ESP_LOGI(TAG, "[ 3 ] Start and wait for Wi-Fi network");
esp_periph_config_t periph_cfg = DEFAULT_ESP_PERIPH_SET_CONFIG();
esp_periph_set_handle_t set = esp_periph_set_init(&periph_cfg);
periph_wifi_cfg_t wifi_cfg = {
    .wifi_config.sta.ssid = CONFIG_WIFI_SSID,
    .wifi_config.sta.password = CONFIG_WIFI_PASSWORD,
};
esp_periph_handle_t wifi_handle = periph_wifi_init(&wifi_cfg);
esp_periph_start(set, wifi_handle);
periph_wifi_wait_for_connected(wifi_handle, portMAX_DELAY);
```

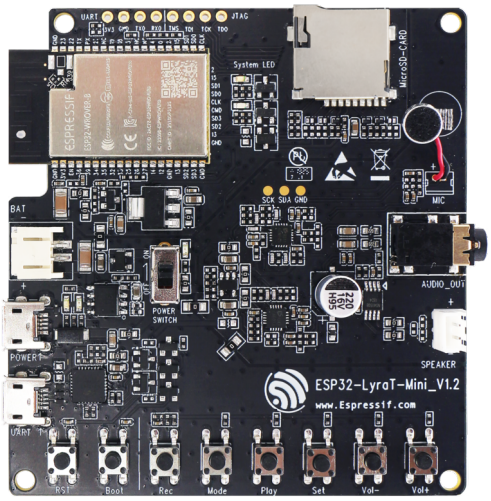
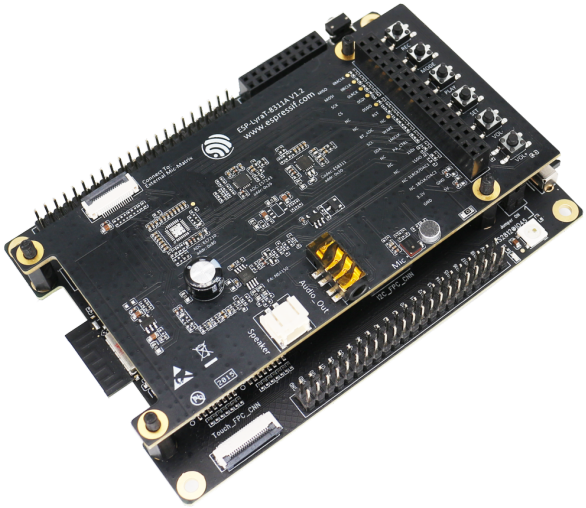
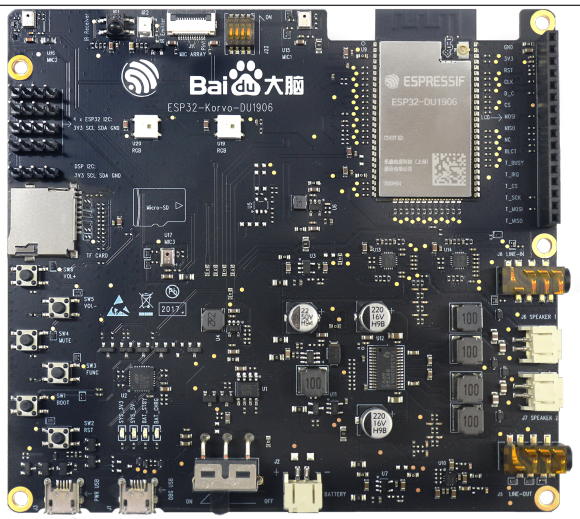
Audio Player

Example

Please refer to cli for an example of initializing `esp_audio` as an audio player.

3.4 Development Boards

Below are getting started guides and hardware details of audio development boards designed by Espressif.

	
Getting Started with ESP32-LyraT	Getting Started with ESP32-LyraT-Mini
Hardware Reference of ESP32-LyraT	Hardware Reference of ESP32-LyraT-Mini
	
Getting Started with ESP32-LyraTD-MSC	Getting Started with ESP32-S2-Kaluga-1-Kit
	
Getting Started with ESP32-Korvo-DU1906	Getting Started with ESP32-S3-Korvo-2

3.4.1 ESP32-LyraT-Mini V1.2 入门指南

[English]

本文档为用户提供 ESP32-LyraT-Mini v1.2 音频开发板的功能描述、配置选项以及如何快速入门使用 ESP32-LyraT 开发板。

ESP32-LyraT 是一个基于双核 ESP32 用于开发音频应用程序的硬件平台，例如 Wi-Fi 音箱、蓝牙音箱、语音遥控器以及有一个或多个音频功能的智能家居应用等。

ESP32-LyraT-Mini 是单声道音频开发板，如需立体声音频开发板，请前往*ESP32-LyraT V4.3 入门指南*。

准备工作

- *ESP32-LyraT-Mini V1.2 开发板*
- 扬声器或 3.5 mm 的耳机（若使用扬声器，建议功率不超过 3 瓦特，另外需要接口为 JST PH 2.0 毫米 2 针的插头，若没有此插头，开发过程中可替换为杜邦母跳线）
- 两个 Micro-USB 2.0 数据线（Type A 到 Micro B）
- PC（Windows、Linux 或 Mac OS）

可选组件

- Micro SD 卡
- 锂电池

如需立即使用此开发板，请直接前往应用程序开发。

概述

ESP32-LyraT-Mini 1.2 是基于 乐鑫 ESP32 专为音频应用市场打造的音频开发板。其主要包括了音频编解码芯片，外置扩展 RAM 和双核 ESP32 芯片。硬件具体包括：

- **ESP32-WROVER-E 模组**
- **音频编解码芯片**
- **ADC 芯片**
- **麦克风**
- **音频输出**
- **1 x 3 瓦特扬声器**输出
- **MicroSD 卡**槽（一线模式）
- **八个键**
- **两个系统指示灯**

- **JTAG** 和 **UART** 测试点
- 集成 **USB-UART 桥接芯片**
- 锂 **电池充电管理**

下图所示的是 ESP32-LyraT-Mini 的主要组件以及组件之间的连接方式。

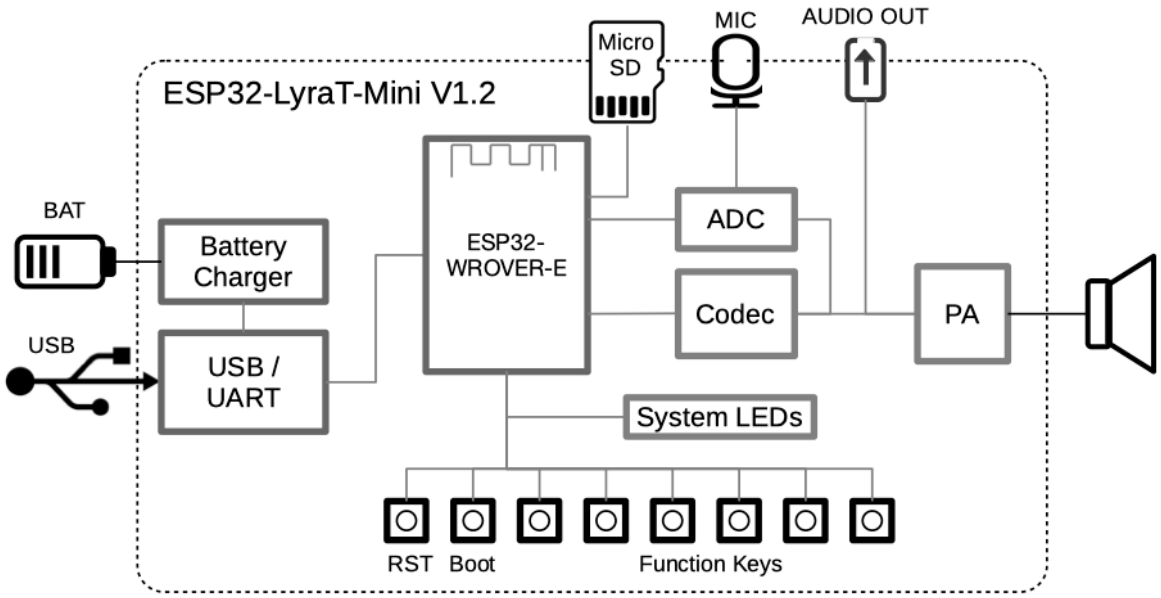


图 4: ESP32-LyraT-Mini 框图

组件

本指南涉及到的 ESP32-LyraT-Mini 开发板的主要组件、接口及控制方式见下。详细技术文档请参阅*ESP32-LyraT-Mini V1.2 Hardware Reference* 和 ESP32-LyraT-Mini V1.2 原理图 (PDF)。ESP32-LyraT-Mini 开发板各组件的详细描述见下表（从右上角起顺时针顺序）。

音频编解码芯片 音频编解码芯片 ES8311 是一款低功耗单声道音频编解码器。它由单通道 ADC、单通道 DAC、低噪声前置放大器、耳机驱动程序、数字音效处理器、模拟混音器和增益函数组成。该芯片通过 I2S 和 I2C 总线与 **ESP32-WROVER-E 模组**连接提供硬件音频处理功能。

音频输出 音频输出插槽，可连接 3.5 mm 立体声耳机。（请注意，此开发板输出单声道信号）

扬声器输出 音频输出插槽，采用 2.00 mm / 0.08” 排针间距，建议连接 4 欧姆 3 瓦特扬声器。

USB-UART 接口 作为 PC 与 ESP32 之间的通信接口。

USB 充电端口 可用作开发板的供电电源。

待机/充电指示灯 绿色 **待机**指示灯亮起表示电源已接入 **USB 充电端口**。红色 **充电**指示灯亮起表示 **电池座**上的电池正在充电。

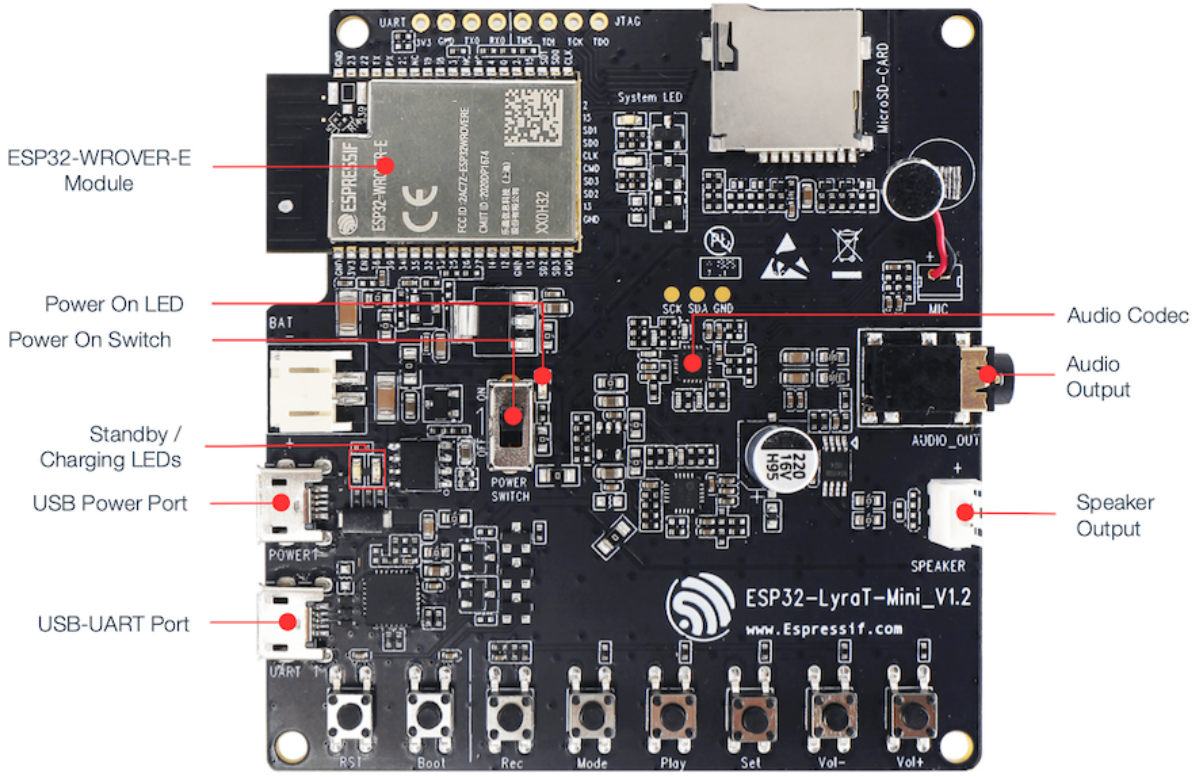


图 5: ESP32 LyraT-Mini V1.2 开发板布局概览

电源开关 电源开/关按钮：向上拨动按钮则开发板电源开启；向下拨动则电源关闭。

电源指示灯 红色指示灯亮起表示 **电源开关**已开启。

ESP32-WROVER-E 模组 ESP32-WROVER-E 模组采用 ESP32 芯片，可实现 Wi-Fi/蓝牙连接和数据处理，同时集成 4 MB 外部 SPI flash 和 8 MB SPI PSRAM，可实现灵活的数据存储。

应用程序开发

ESP32-LyraT-Mini 上电之前，请首先确认开发板完好无损。

初始设置

设置开发板，运行首个示例应用程序：

1. 连接扬声器到 **扬声器输出**。也可选择连接耳机到 **音频输出**。
2. 使用 Micro-USB 数据线将 ESP32-LyraT-Mini 开发板的 **两个 USB 端口**均与 PC 相连。
3. 此时，绿色 **待机指示灯**应亮起。假设电池未连接，那么红色 **充电指示灯**将每隔几秒闪烁一次。
4. 向上拨动 **电源开关**。
5. 此时，红色 **电源指示灯**应亮起。

如果指示灯如上述显示，则此开发板基本完好，可用于下载应用程序。现在，请按下文介绍运行并配置 PC 上的开发工具。

正式开始开发

如果 ESP32 LyraT 的初始设置已检查完成，请准备开发工具。前往*Development Boards* 查看以下步骤：

- *Step 1. Set up ESP-IDF* 提供了一套 ESP32 的 C 语言 PC 开发编译环境；
- *Step 2. Get ESP-ADF* 获取开发音频应用程序的 API；
- *Step 3. Set up the environment* 使编译器找到音频应用 API；
- *Step 4. Start a Project* 提供 ESP32-LyraT-Mini 开发板的音频应用程序示例；
- *Step 5. Connect Your Device* 准备加载应用程序；
- *Step 7. Build the Project* 最后运行应用程序并播放音乐。

修订历史

- 板上模组从 ESP32-WROVER-B 更新为 ESP32-WROVER-E。

其他 LyraT 系列开发板

- *ESP32-LyraT V4.3 入门指南*
- *ESP32-LyraTD-MSC V2.2 入门指南*

相关文档

- ESP32-LyraT-Mini V1.2 原理图 (PDF)
- ESP32-LyraT-Mini V1.2 尺寸图 (PDF)
- *ESP32-LyraT-Mini V1.2 Hardware Reference*
- ESP32 技术规格书 (PDF)
- ESP32-WROVER-E 技术规格书 (PDF)

3.4.2 ESP32-LyraT-Mini V1.2 Hardware Reference

This guide provides functional descriptions and configuration options for ESP32-LyraT-Mini V1.2 audio development board. As an introduction to functionality and using the LyraT, please see *ESP32-LyraT-Mini V1.2 入门指南*.

In this Section

- *Overview*
- *Functional Description*
- *Allocation of ESP32 Pins to Test Points*
  - *JTAG Test Point*
  - *UART Test Point*
- *MicroSD Card*
- *GPIO Allocation Summary*
- *Notes on Power Distribution*
  - *Power Supply over USB and from Battery*
  - *Independent Audio and Digital Power Supply*
- *Selecting of the Audio Output*

- *Related Documents*

Overview

ESP32-LyraT is a hardware platform designed for the dual-core ESP32 audio applications, e.g., Wi-Fi or BT audio speakers, speech-based remote controllers, connected smart-home appliances with one or more audio functionality, etc.

The block diagram below presents main components of ESP32-LyraT-Mini.

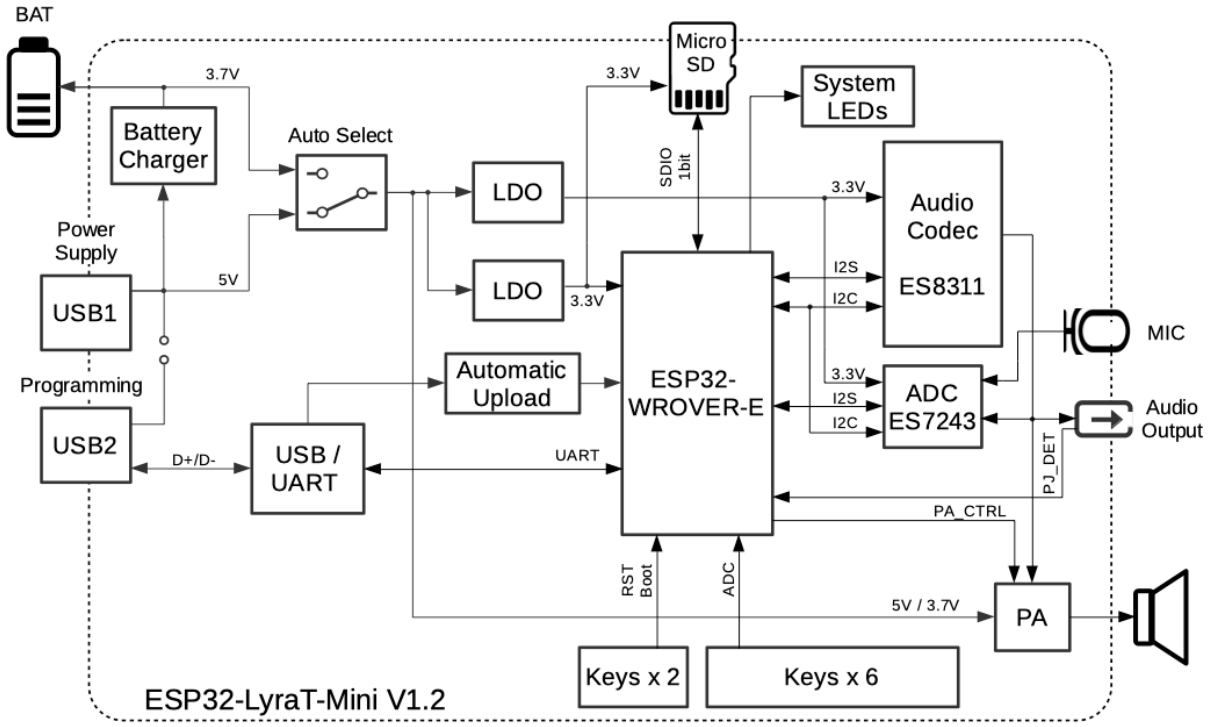


图 6: ESP32-LyraT-Mini V1.2 Electrical Block Diagram

Functional Description

The following list and figure describe key components, interfaces, and controls of the ESP32-LyraT-Mini board. The list provides description starting from the picture’ s top right corner and going clockwise.

MicroSD Slot The development board supports a MicroSD card in SPI/1-bit modes, and can store or play audio files in the MicroSD card. See *MicroSD Card* for pinout details.

Microphone On-board microphone connected to AINRP/AINRP of the **Audio ADC Chip**.

System LEDs Two general purpose LEDs (green and red) controlled by **ESP32-WROVER-E Module** to indicate certain operation states of the audio application using dedicated API.

Audio Codec The audio codec chip, ES8311, is a low power mono audio codec. It consists of 1-channel ADC, 1-channel DAC, low noise pre-amplifier, headphone driver, digital sound effects, analog mixing, and gain functions. It is interfaced with **ESP32-WROVER-E Module** over I2S and I2C buses to provide audio processing in hardware independently from the audio application.

Audio Output Output socket to connect headphones with a 3.5 mm stereo jack. One of the socket's terminals is wired to ESP32 to provide jack insertion detection.

ADC The audio codec chip, ES7243, is a low power multi-bit delta-sigma audio ADC and DAC. In this board this chip is used as the microphone interface.

PA A power amplifier used to amplify the audio signal from the **Audio Codec Chip** for driving the speaker.

Speaker Output Output socket to connect a speaker. The 4-ohm and 3-watt speaker is recommended. The pins have a 2.00 mm / 0.08" pitch.

Function Press Keys Six press keys labeled **Rec**, **Mode**, **Play**, **Set**, **Vol-**, and **Vol+**. They are routed to **ESP32-WROVER-E Module** and intended for development and testing of a UI for audio applications using dedicated API.

Boot/Reset Press Keys Boot: holding down the **Boot** button and momentarily pressing the **Reset** button initiates the firmware upload mode. Then user can upload firmware through the serial port. Reset: pressing this button alone resets the system.

Automatic Upload A simple two transistor circuit to put ESP32 into firmware upload mode depending on the status of UART DTR and RTS signals. The signals are controlled by an external application to upload the firmware over the USB-UART interface.

USB-UART Port Functions as the communication interface between a PC and the ESP32 module.

USB-UART Bridge A single chip USB-UART bridge CP2102N provides up to 3 Mbps transfers rates.

USB Power Port Provides the power supply for the board.

Standby/Charging LEDs The **Standby** green LED indicates that power has been applied to the **USB Power Port**. The **Charging** red LED indicates that a battery connected to the **Battery Socket** is being charged.

Battery Socket Two-pin socket to connect a single cell Li-ion battery. The pins have a 2.00 mm / 0.08" pitch. The battery serves as an alternative power supply to the **USB Power Port** for charging the board. Make sure to use a Li-ion battery that has protection circuit and fuse. The recommended specifications of the battery: capacity > 1000 mAh, output voltage 3.7 V, input voltage 4.2 V –5 V. Please verify if polarity on the battery plug matches polarity of the socket as marked on the board's soldermask besides the socket.

Battery Charger Constant current and constant voltage linear charger for single cell lithium-ion batteries AP5056. Used for charging of a battery connected to the **Battery Socket** over the **USB Power Port**.

Power Supervisor Provides EN signal to enable ESP32 once power supply voltage stabilizes.

Power On Switch Power on/off knob: toggling it to the top powers the board on; toggling it to the down powers the board off.

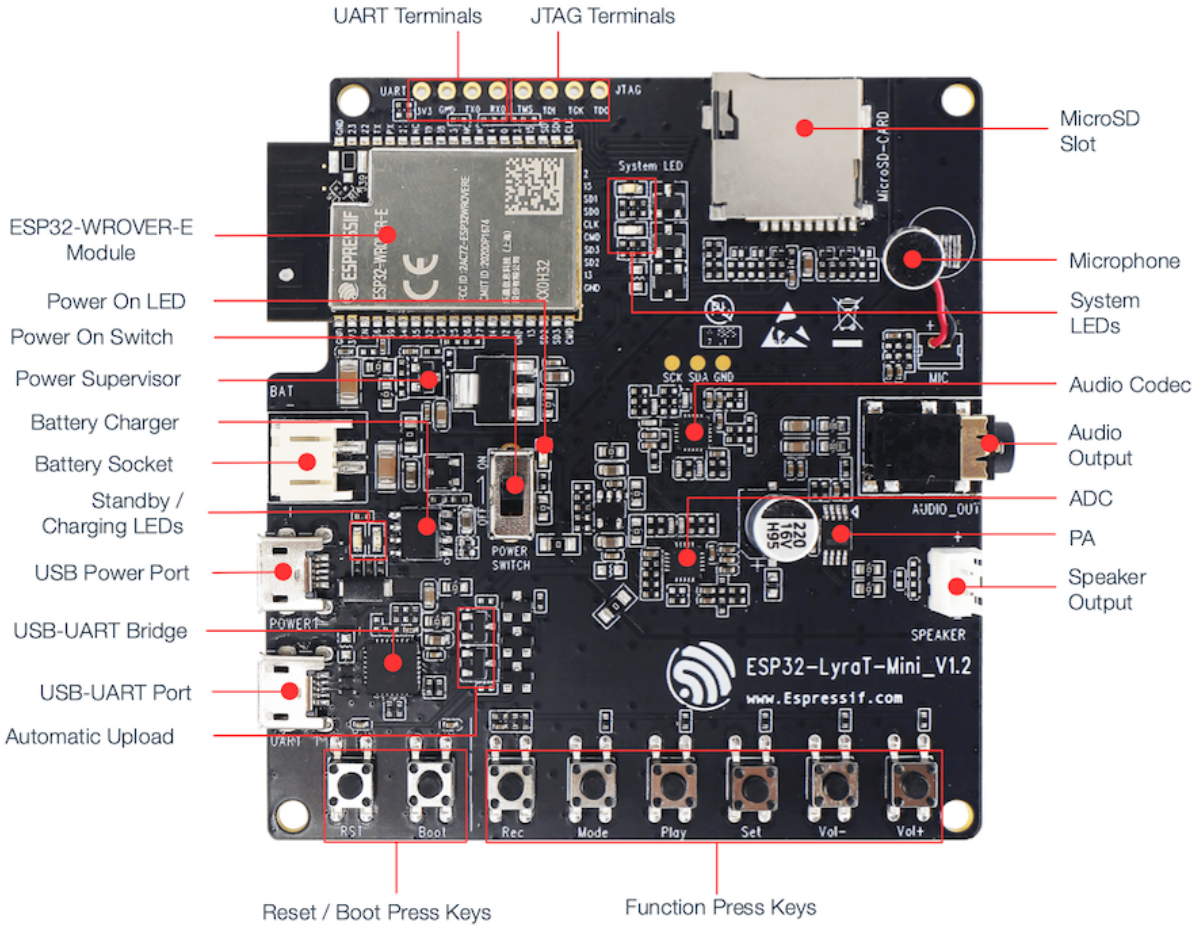


图 7: ESP32 LyraT-Mini V1.2 Board Layout

注解： The **Power On Switch** does not affect / disconnect the Li-ion battery charging. More information, you can refer to ESP32-LyraT-Mini V1.2 schematic (PDF).

Power On LED Red LED indicating that **Power On Switch** is turned on.

ESP32-WROVER-E Module The ESP32-WROVER-E module contains ESP32 chip to provide Wi-Fi / Bluetooth connectivity and data processing power as well as integrates 4 MB external SPI flash and an additional 8 MB PSRAM for flexible data storage.

UART Termininals Serial port: provides access to the serial TX/RX signals between **ESP32-WROVER-E Module** and **USB-UART Bridge Chip**. See *UART Test Point* for pinout details.

JTAG Termininals Provides access to the **JTAG** interface of **ESP32-WROVER-E Module**. It may be used for debugging, application upload, as well as implementing several other functions, e.g., Application Level Tracing. See *JTAG Test Point* for pinout details.

Allocation of ESP32 Pins to Test Points

This section describes allocation of test points available on the ESP32-LyraT-Mini board.

The test points are bare through hole solder pads and have standard 2.54 mm / 0.1 inch pitch. User may need to populate them with pin headers or sockets for easy connection of external hardware.

JTAG Test Point

.	ESP32 Pin	JTAG Signal
1	MTDO / GPIO15	TDO
2	MTCK / GPIO13	TCK
3	MTDI / GPIO12	TDI
4	MTMS / GPIO14	TMS

UART Test Point

.	ESP32 Pin	Pin Description
1	RXD0	RX
2	TXD0	TX
3	GND	GND
4	n/a	3.3 V

MicroSD Card

Implemented on this board MicoSD card interface operates in SPI/1-bit mode. The board is able to support SPI/4-b it mode after populating couple of additional components on locations reserved on the PCB. See ESP32-LyraT-Mini V1.2 schematic (PDF) for additional information. Not populated components are marked *(NC)* on the schematic.

.	ESP32 Pin	MicroSD Signal
1	MTDI / GPIO12	–
2	MTCK / GPIO13	–
3	MTDO / GPIO15	CMD
4	MTMS / GPIO14	CLK
5	GPIO2	DATA0
6	GPIO4	–
7	GPIO34	CD

GPIO Allocation Summary

The table below provides allocation of GPIOs exposed on terminals of **ESP32-WROVER-E Module** to control specific components or functions of the board.

Pin [1]	Pin Name	ES8311	ES7243	Keys	MicroSD	Other
3	EN			EN_KEY		
4	S_VP		I2S_DATA			
5	S_VN			REC, MODE, PLAY, SET, VOL-, VOL+		
6	IO34				CD	
7	IO35	I2S0_ASDOUT				
8	IO32		I2S1_SCLK			
9	IO33		I2S1_LRCK			
10	IO25	I2S0_LRCK				
11	IO26	I2S0_DSDIN				
12	IO27					Blue_LED
13	IO14				CLK	
14	IO12				NC (DATA2)	
16	IO13				NC (DATA3)	
17	SD2					
18	SD3					
19	CMD					
20	CLK					
21	SD0					

下页继续

表 1 – 续上页

Pin [1]	Pin Name	ES8311	ES7243	Keys	MicroSD	Other
22	SD1					
23	IO15				CMD	
24	IO2			IO2_KEY	DATA0	
25	IO0	I2S0_MCLK	I2S1_MCLK	IO0_KEY		
26	IO4				NC (DATA1)	
27	NC (IO16)					
28	NC (IO17)					
29	IO5	I2S0_SCLK				
30	IO18	I2C_SDA	I2C_SDA			
31	IO19					PJ_DET [2]
33	IO21					PA_CTRL [3]
34	RXD0					RXD0 [4]
35	TXD0					TXD0 [4]
36	IO22					Green_LED
37	IO23	I2C_SCK	I2C_SCL			

1. **Pin** - ESP32-WROVER-E module pin number, GND and power supply pins are not listed
2. **PJ_DET** - phone jack insertion detect signal
3. **PA_CTRL** - NS4150 power amplifier chip control signal
4. **RXD0**, **TXD0** - serial communication signals connected to TXD and RXD pins of CP2102N USB-UART bridge
5. **NC** - not connected

Notes on Power Distribution

The ESP32-LyraT-Mini board provides some basic features to isolate noise from digital components by providing separate power distribution for audio and digital subsystems.

Power Supply over USB and from Battery

There are two ways to power the development board: 5 V USB Power Port or 3.7 V optional battery. The optional battery is preferable for applications where a cleaner power supply is required.

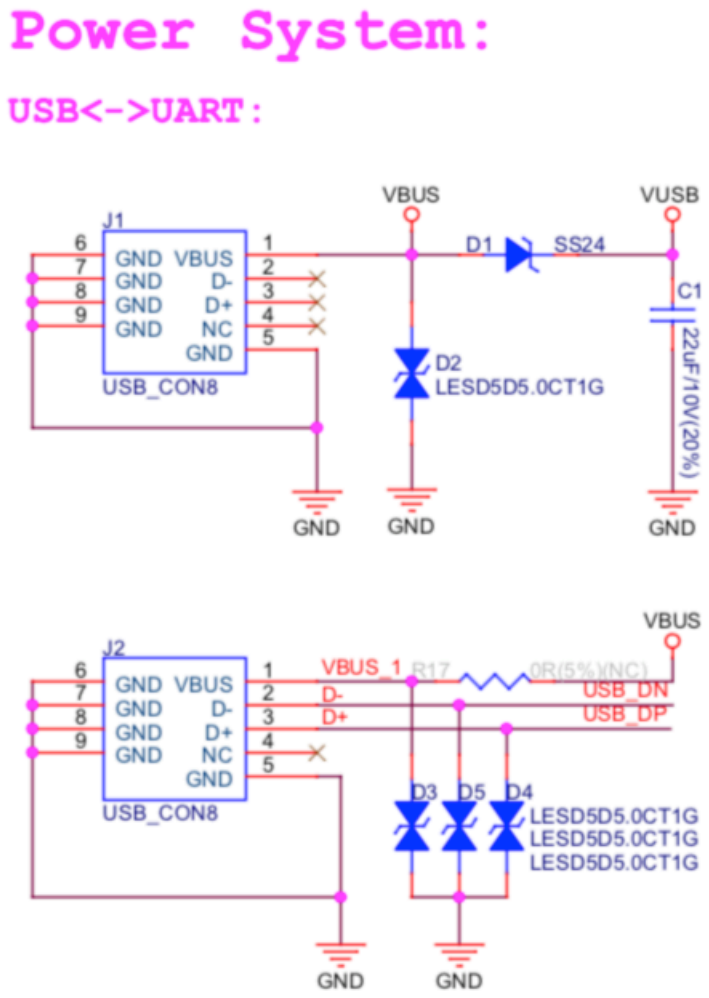


图 8: ESP32-LyraT-Mini V1.2 - Dedicated USB Power Supply Socket

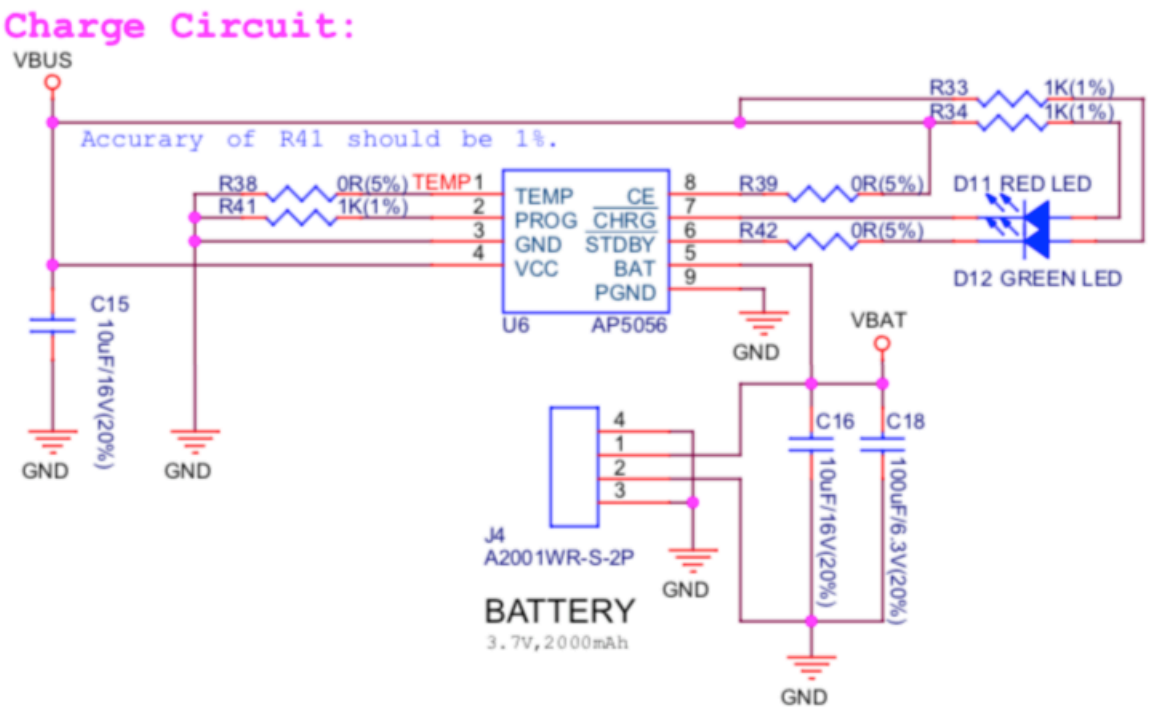


图 9: ESP32-LyraT-Mini V1.2 - Power Supply from a Battery

Independent Audio and Digital Power Supply

The board features independent power supplies to the audio components and the ESP32 module. This should reduce noise in the audio signal from digital components and improve overall performance of the components.

Selecting of the Audio Output

The board provides a mono audio output signal on pins OUTN and OUTP of the ES8311 codec chip. The signal is routed to two outputs:

- Power amplifier (PA) to drive an external speaker
- Phone jack socket to drive external headphones

The board design assumes that selection between one of these outputs is implemented in software, as opposed to using traditional mechanical contacts in a phone jack socket, that would disconnect the speaker once a headphone jack is inserted.

Two digital IO signals are provided to implement selection between the speaker and the headphones:

- **PJ_DET** - digital input signal to o detect when a headphone jack is inserted,
- **PA_CTRL** - digital output signal to enable or disable the amplifier IC.

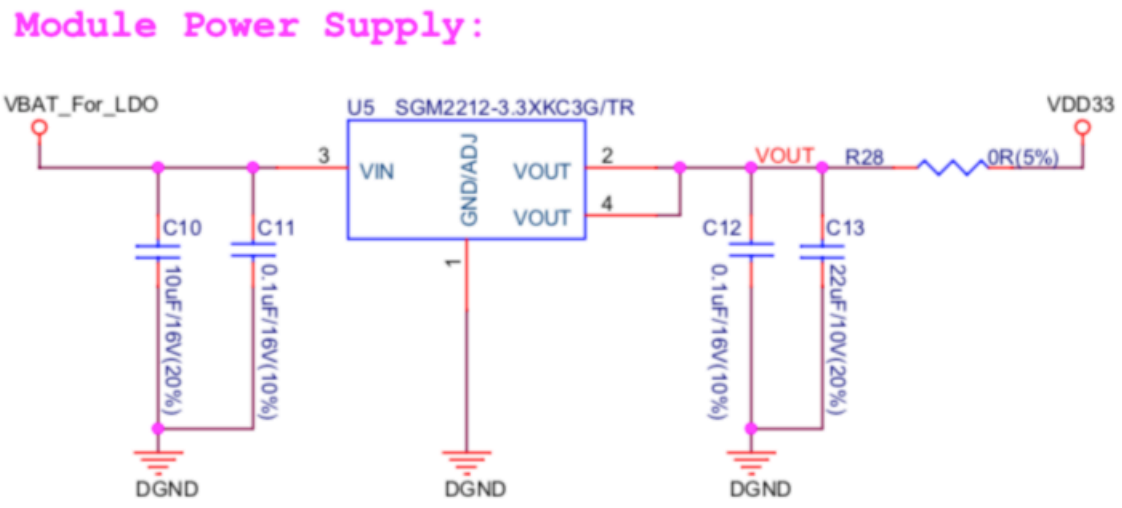


图 10: ESP32-LyraT-Mini V1.2 - Digital Power Supply

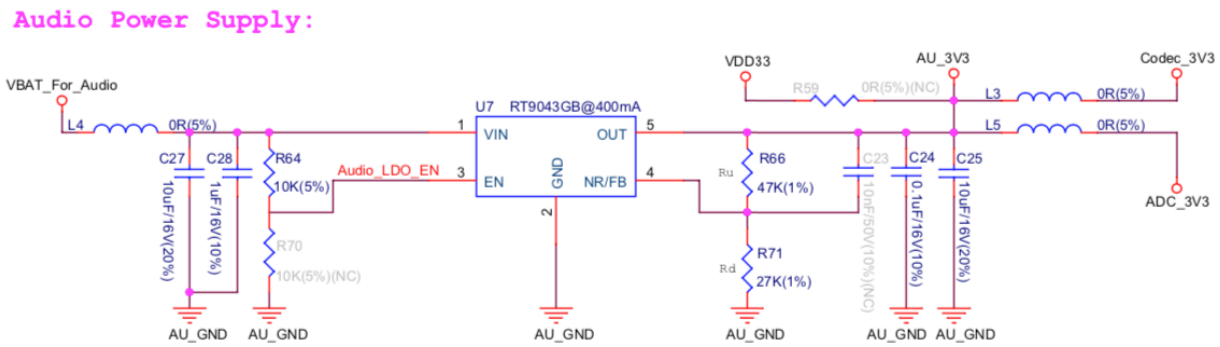


图 11: ESP32-LyraT-Mini V1.2 - Audio Power Supply

The application running on ESP32 may then enable or disable the PA with **PA_CTRL** basing on status of **PJ_DET**. Please see *GPIO Allocation Summary* for specific GPIO numbers allocated to these signals.

Related Documents

- ESP32-LyraT-Mini V1.2 Schematic (PDF)
- ESP32-LyraT-Mini V1.2 Board Dimensions (PDF)
- *ESP32-LyraT-Mini V1.2 入门指南*
- ESP32 Datasheet (PDF)
- ESP32-WROVER-E Datasheet (PDF)

3.4.3 ESP32-LyraT V4.3 入门指南

[English]

本指南旨在向用户介绍 ESP32-LyraT V4.3 音频开发版的功能、配置选项以及如何快速入门。如果您的开发板不是 4.3 版本，请前往其他 *LyraT 开发板版本*。

ESP32-LyraT 开发板面向搭载 ESP32 双核芯片的音频应用领域，如 Wi-Fi 音箱、蓝牙音箱、语音遥控器以及所有需要音频功能的智能家居应用。

ESP32-LyraT 是一款立体声音频开发板，如需单声道音频开发板，请前往*ESP32-LyraT-Mini V1.2 入门指南*。

准备工作

- 1 × *ESP32 LyraT V4.3 开发板*
- 2 x 扬声器或 3.5 mm 的耳机（若使用扬声器，建议功率不超过 3 瓦特，另外需要接口为 JST PH 2.0 毫米 2 针的插头，若没有此插头，开发过程中可替换为杜邦母跳线）
- 2 x Micro-USB 2.0 数据线（Type A 转 Micro B）
- 1 × PC（Windows、Linux 或 Mac OS）

如需立即使用此开发板，请直接前往正式开始开发。

概述

ESP32-LyraT V4.3 是一款基于 乐鑫 ESP32 的开发板，专为音频应用市场打造，除 ESP32 芯片原有的硬件外，还提供了可用于音频处理硬件和扩展 RAM。具体包括以下硬件：

- ESP32-WROVER-E 模组
- 音频编解码芯片

- 板载双麦克风
- 耳机输出
- 2 个 3-watt 扬声器输出
- 双辅助输入
- MicroSD 卡槽（一线模式或四线模式）
- 6 个按键（2 个物理按键和 4 个触摸按键）
- JTAG 排针
- 集成 USB-UART 桥接芯片
- 锂电池充电管理

下图展示的是 ESP32-LyraT 的主要组件以及组件之间的连接方式。

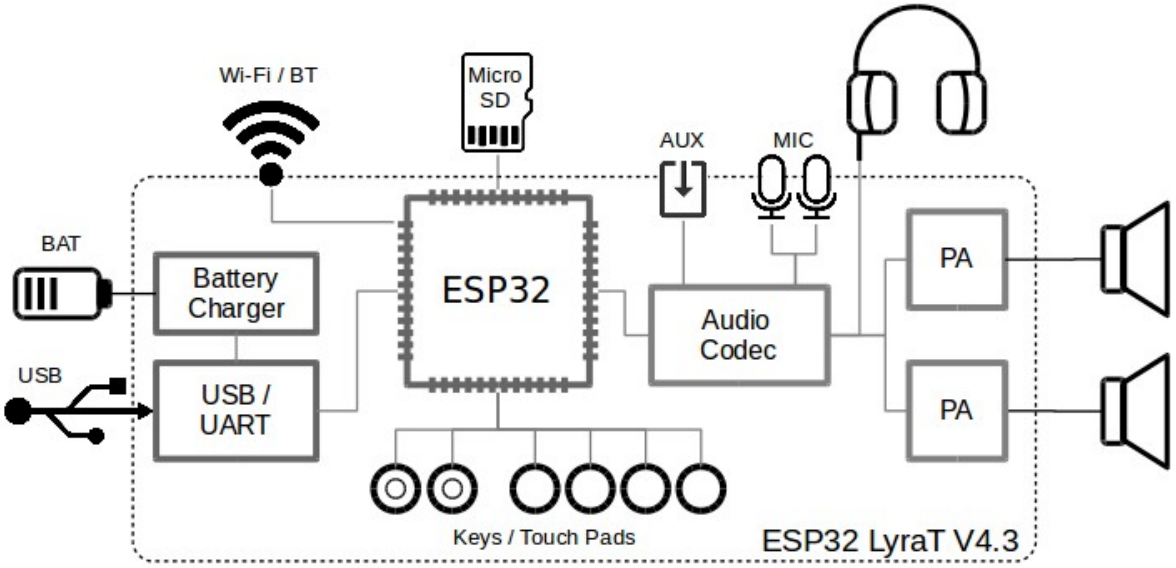


图 12: ESP32-LyraT 开发板框图

组件

以下列表和图片仅涉及 ESP32-LyraT 的主要组件、接口和控制方式，仅展示目前所需的信息。欲访问详细的技术文档，请前往*ESP32-LyraT V4.3 Hardware Reference* 和 ESP32 LyraT V4.3 schematic (PDF)。

ESP32-WROVER-E 模组 ESP32-WROVER-E 模组采用 ESP32 芯片，可实现 Wi-Fi/蓝牙连接和数据处理，同时集成 4 MB 外部 SPI flash 和 8 MB SPI PSRAM，可实现灵活的数据存储。

耳机输出 输出插槽可连接 3.5 mm 立体声耳机。

> **注意：** 该插槽可接入移动电话耳机，只与 OMPT 标准耳机兼容，与 CTIA 耳机不兼容。更多耳机标准信息请访问维基百科 Phone connector (audio) 词条。

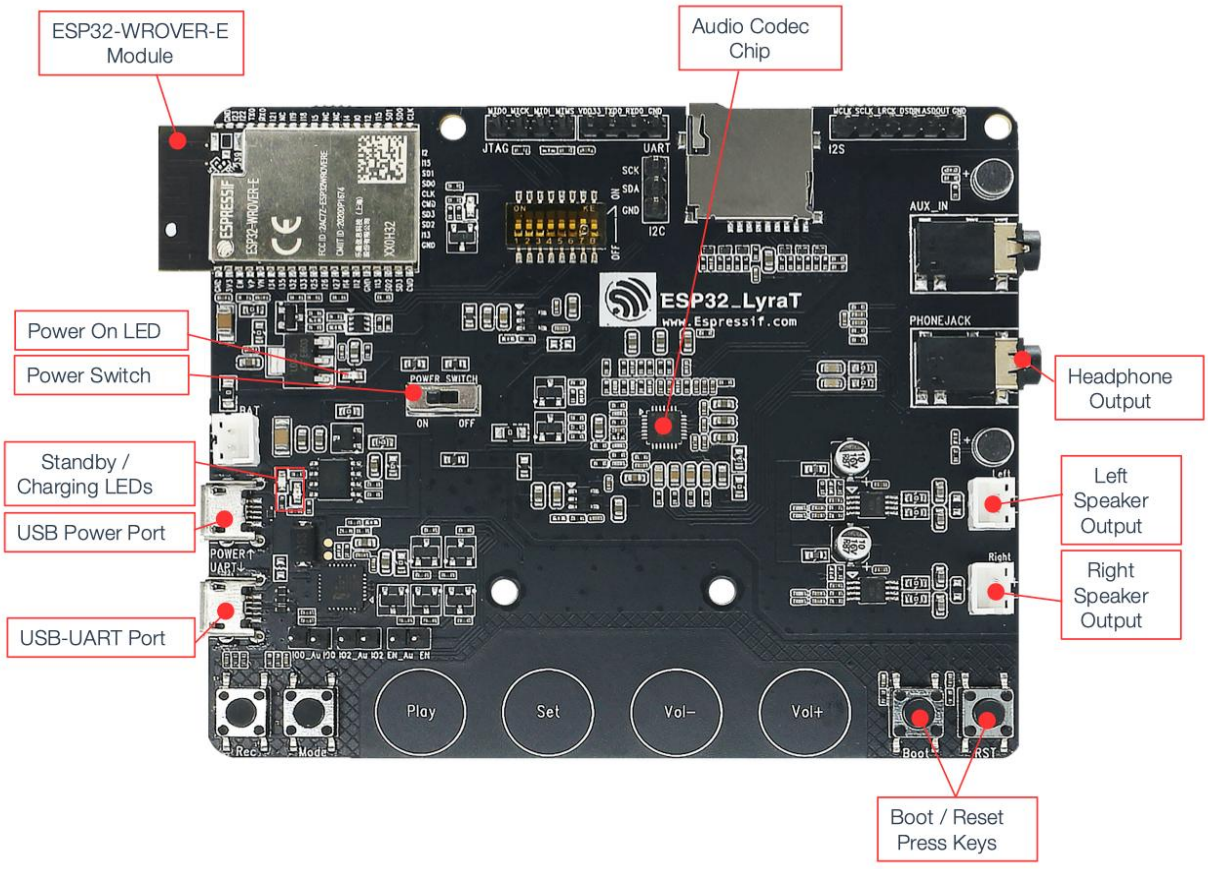


图 13: ESP32-LyraT V4.3 开发板布局示意图

左侧扬声器输出 音频输出插槽，采用 2.00 mm / 0.08” 排针间距，建议连接 4 欧姆 3 瓦特扬声器。

右侧扬声器输出 音频输出插槽，采用 2.00 mm / 0.08” 排针间距，建议连接 4 欧姆 3 瓦特扬声器。

启动/复位按键 启动: 长按 **Boot** 键，然后按下 **Reset** 键进入烧写模式，此时可通过串行端口上传固件。复位: 仅按下 **Reset** 键只能重置系统。

音频编解码芯片 ES8388 音频编解码芯片是一款低功耗立体声编解码器，它由双通道 ADC、双通道 DAC、麦克风放大器、耳机放大器、数字音效处理器、模拟混音和增益控制功能组成。该芯片通过 I2S 和 I2C 总线与 **ESP32-WROVER-E 模组**连接，可在芯片内独立完成音频处理，无需依赖音频应用软件。

USB-UART 接口 作为 PC 和 **ESP32-WROVER-E** 模组之间的通信接口。

USB 供电接口 为开发板供电。

待机/充电指示灯 绿色 **待机**指示灯亮起时，表示电源已接入 **Micro USB 接口**；红色 **充电**指示灯亮起时，表示

连接至 **电池接口**上的电池正在充电。

电源开关 电源开/关按钮：向左拨动按钮则开发板电源开启，向右拨动则电源关闭。

电源指示灯 红色指示灯亮起表示 **电源开关**已开启。

应用程序开发

ESP32-LyraT 上电之前，请首先确认开发板完好无损。

初始设置

设置开发板，运行首个示例应用程序：

1. 连接扬声器至 **两个扬声器输出**，或将耳机连接至 **耳机输出**。
2. 插入 Micro-USB 数据线，连接 PC 与 ESP32-LyraT 开发板的 **两个 USB 端口**。
3. 此时，绿色 **待机指示灯**应亮起。若电池未连接，红色 **充电指示灯**每隔几秒闪烁一次。
4. 向左拨动 **电源开关**。
5. 此时，红色 **电源指示灯**应亮起。

如果指示灯如上述显示，则初始设置已经完成，开发板可用于下载应用程序。现在，请按下文介绍运行并配置 PC 上的开发工具。

正式开始开发

若已完成初始设置，请准备开发工具。请前往*Installation Step by Step* 查看以下步骤的：

- **Set up ESP-IDF** 提供一套 ESP32 和 ESP32-S2 芯片的 C 语言 PC 开发编译环境；
- **Get ESP-ADF** 获取开发音频应用程序的 API；
- **Setup Path to ESP-ADF** 使开发框架获取到音频应用 API；
- **Start a Project** 为开发板提供音频应用程序示例；
- **Connect Your Device** 准备加载应用程序；
- **Build the Project** 运行应用程序，播放音乐。

与 LyraT V4.2 相比的主要变化

- 板上模组从 ESP32-WROVER 更新为 ESP32-WROVER-E；
- 移除红色 LED 指示灯；
- 增添耳机插孔插入检测功能；
- 使用两枚独立芯片代替单个功率放大器；
- 更新一些电路的功率管理设计：电池充电、ESP32、MicroSD、编解码芯片以及功率放大器；
- 更新一些电路的电器实施设计：UART、编解码芯片、左右两侧麦克风、AUX 输入、耳机输出、MicroSD、按键以及自动上传。

其他 LyraT 开发板版本

- *ESP32-LyraT V4.2 Getting Started Guide*
- *ESP32-LyraT V4 Getting Started Guide*

其他 LyraT 系列开发板

- *ESP32-LyraT-Mini V1.2 入门指南*
- *ESP32-LyraTD-MSC V2.2 入门指南*

相关文档

- *ESP32-LyraT V4.3 Hardware Reference*
- ESP32 LyraT V4.3 schematic (PDF)
- ESP32-LyraT V4.3 Component Layout (PDF)
- ESP32 技术规格书 (PDF)
- ESP32-WROVER-E 技术规格书 (PDF)

3.4.4 ESP32-LyraT V4.3 Hardware Reference

This guide provides functional descriptions, configuration options for ESP32-LyraT V4.3 audio development board. As an introduction to functionality and using the LyraT, please see *ESP32-LyraT V4.3 入门指南*. Check section *Other Versions of LyraT* if you have different version of the board.

In this Section

- *Overview*
- *Functional Description*
  - *Hardware Setup Options*
    - *Enable MicroSD Card in 1-wire Mode*
    - *Enable MicroSD Card in 4-wire Mode*
    - *Enable JTAG*
    - *Using Automatic Upload*
  - *Allocation of ESP32 Pins*
  - *Pinout of Extension Headers*
    - *UART Header / JP2*
    - *I2S Header / JP4*
    - *I2C Header / JP5*
    - *JTAG Header / JP7*
  - *Notes of Power Distribution*
    - *Power Supply Separation*
    - *Three Dedicated LDOs*
    - *Separate Power Feed for the PAs*
  - *Selecting of the Audio Output*
- *Other Versions of LyraT*
- *Related Documents*

Overview

The ESP32-LyraT development board is a hardware platform designed for the dual-core ESP32 audio applications, e.g., Wi-Fi or BT audio speakers, speech-based remote controllers, smart-home appliances with audio functionality(ies), etc.

The block diagram below presents main components of the ESP32-LyraT.

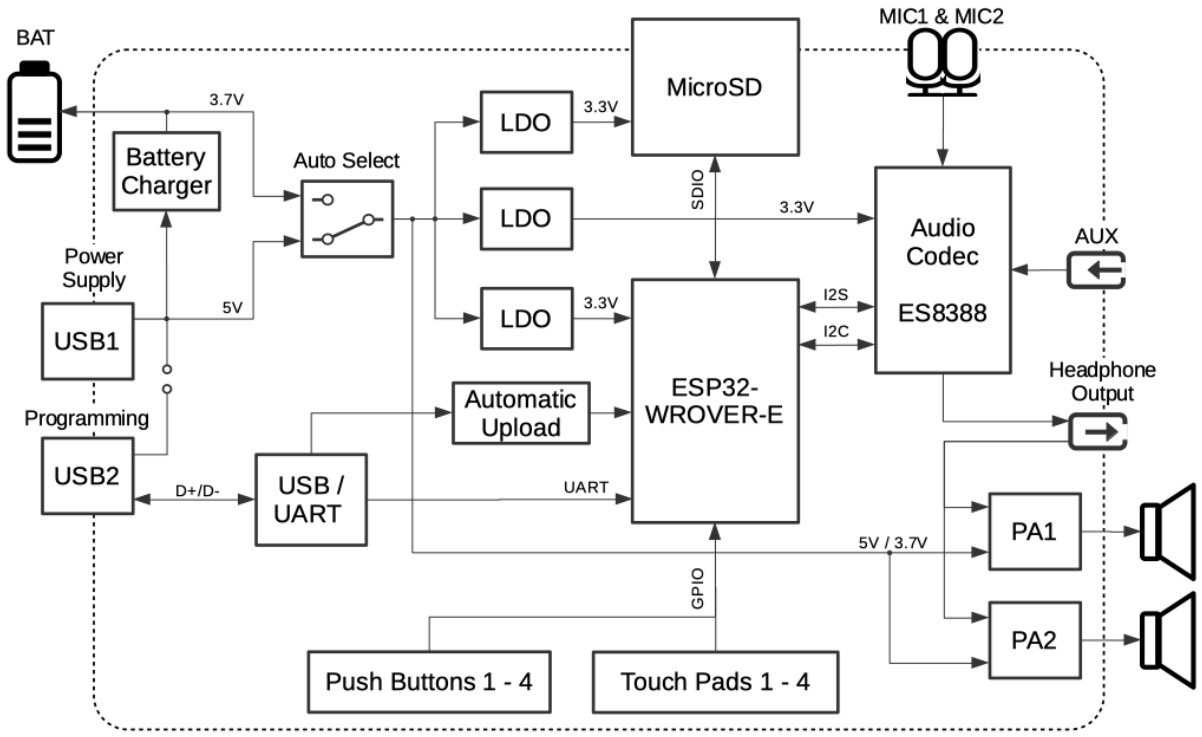


图 14: ESP32-LyraT V4.3 Electrical Block Diagram

Functional Description

The following list and figure describe key components, interfaces and controls of the ESP32-LyraT board.

ESP32-WROVER-E Module The ESP32-WROVER-E module contains ESP32 chip to provide Wi-Fi / Bluetooth connectivity and data processing power as well as integrates 4 MB external SPI flash and an additional 8 MB PSRAM for flexible data storage.

Green LED A general purpose LED controlled by the **ESP32-WROVER-E Module** to indicate certain operation states of the audio application using dedicated API.

Function DIP Switch Used to configure function of GPIO12 to GPIO15 pins that are shared between devices, primarily between **JTAG Header** and **MicroSD Card**. By default, the **MicroSD Card** is enabled with all switches in *OFF* position. To enable the **JTAG Header** instead, switches in positions 3, 4, 5 and 6 should be put *ON*. If **JTAG** is not used and **MicroSD Card** is operated in the one-line mode, then GPIO12 and GPIO13 may be assigned to other functions. Please refer to ESP32 LyraT V4.3 schematic for more details.

JTAG Header Provides access to the **JTAG** interface of **ESP32-WROVER-E Module**. It may be used for debugging, application upload, as well as implementing several other functions, e.g., Application Level Tracing. See *JTAG Header / JP7* for pinout details. Before using **JTAG** signals to the header, **Function DIP Switch** should be enabled. Please note that when **JTAG** is in operation, **MicroSD Card** cannot be used and should be disconnected because some of JTAG signals are shared by both devices.

UART Header Serial port: provides access to the serial TX/RX signals between **ESP32-WROVER-E Module** and **USB-UART Bridge Chip**.

I2C Header Provides access to the I2C interface. Both **ESP32-WROVER-E Module** and **Audio Codec Chip** are connected to this interface. See *I2C Header / JP5* for pinout details.

MicroSD Slot The development board supports a MicroSD card in SPI/1-bit/4-bit modes, and can store or play audio files in the MicroSD card. Note that **JTAG** cannot be used and should be disconnected by setting **Function DIP Switch** when **MicroSD Card** is in operation, because some of signals are shared by both devices.

I2S Header Provides access to the I2S interface. Both **ESP32-WROVER-E Module** and **Audio Codec Chip** are connected to this interface. See *I2S Header / JP4* for pinout details.

Left Microphone Onboard microphone connected to IN1 of the **Audio Codec Chip**.

AUX Input Auxiliary input socket connected to IN2 (left and right channel) of the **Audio Codec Chip**. Use a 3.5 mm stereo jack to connect to this socket.

Headphone Output Output socket to connect headphones with a 3.5 mm stereo jack.

注解： The socket may be used with mobile phone headsets and is compatible with OMPT standard headsets only. It does work with CTIA headsets. Please refer to Phone connector (audio) on Wikipedia.

Right Microphone Onboard microphone connected to IN1 of the **Audio Codec Chip**.

Left Speaker Output Output socket to connect a speaker. The 4-ohm and 3-watt speaker is recommended. The pins have a 2.00 mm / 0.08” pitch.

Right Speaker Output Output socket to connect a speaker. The 4-ohm and 3-watt speaker is recommended. The pins have a 2.00 mm / 0.08” pitch.

PA Chip A power amplifier used to amplify stereo audio signal from the **Audio Codec Chip** for driving two speakers.

Boot/Reset Press Keys Boot button: holding down the **Boot** button and momentarily pressing the **Reset** button to initiate the firmware download mode. Then you can download firmware through the serial port. Reset button: pressing this button alone resets the system.

Touch Pad Buttons Four touch pads labeled *Play*, *Sel*, *Vol+* and *Vol-*. They are routed to **ESP32-WROVER-E Module** and intended for development and testing of a UI for audio applications using dedicated API.

Audio Codec Chip The Audio Codec Chip, ES8388, is a low power stereo audio codec with a headphone amplifier. It consists of 2-channel ADC, 2-channel DAC, microphone amplifier, headphone amplifier, digital sound effects, analog mixing and gain functions. It is interfaced with **ESP32-WROVER-E Module** over I2S and I2S buses to provide audio processing in hardware independently from the audio application.

Automatic Upload Install three jumpers on this header to enable automatic loading of application to the ESP32. Install all jumpers together on all three headers. Remove all jumpers after upload is complete.

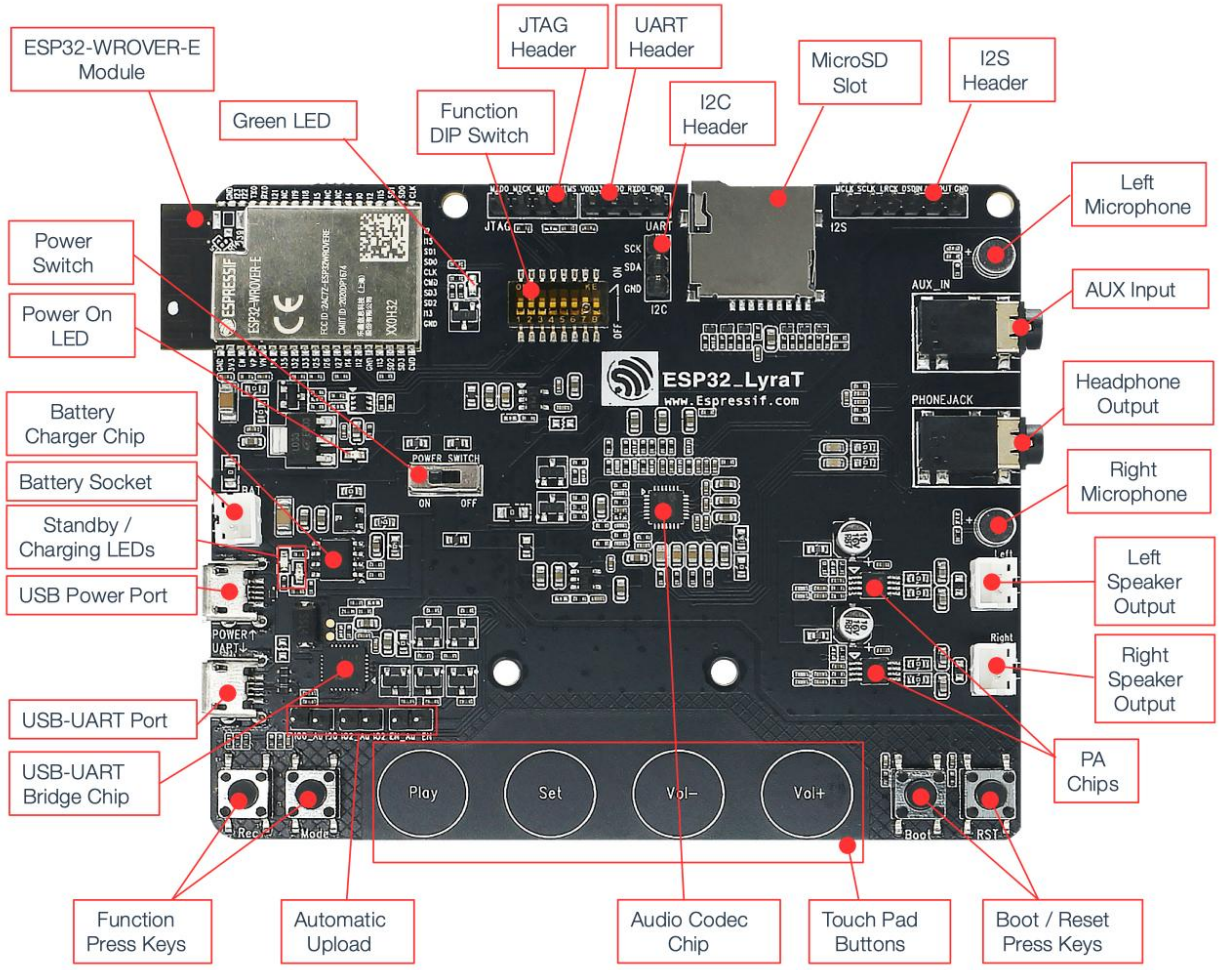


图 15: ESP32-LyraT V4.3 Board Layout

Function Press Keys Two key labeled *Rec* and *Mode*. They are routed to **ESP32-WROVER-E Module** and intended for developing and testing a UI for audio applications using dedicated API.

USB-UART Bridge Chip A single chip USB-UART bridge provides up to 1 Mbps transfers rate.

USB-UART Port Functions as the communication interface between a PC and the ESP32 module.

USB Power Port Provides the power supply for the board.

Standby / Charging LEDs The **Standby** green LED indicates that power has been applied to the **Micro USB Port**. The **Charging** red LED indicates that a battery connected to the **Battery Socket** is being charged.

Battery Socket Two pins socket to connect a single cell Li-ion battery.

注解: Please verify if polarity on the battery plug matches polarity of the socket as marked on the board's soldermask besides the socket.

Battery Charger Chip Constant current & constant voltage linear charger for single cell lithium-ion batteries AP5056. Used for charging of a battery connected to the **Battery Socket** over the **Micro USB Port**.

Power On LED Red LED indicating that **Power On Switch** is turned on.

注解: The **Power On Switch** does not affect / disconnect the Li-ion battery charging.

Power Switch Power on/off knob: toggling it to the left powers the board on; toggling it to the right powers the board off.

Hardware Setup Options

There are a couple of options to change the hardware configuration of the ESP32-LyraT board. The options are selectable with the **Function DIP Switch**.

Enable MicroSD Card in 1-wire Mode

DIP SW	Position
1	OFF
2	OFF
3	OFF
4	OFF
5	OFF
6	OFF
7	OFF [1]
8	n/a

1. **AUX Input** detection may be enabled by toggling the DIP SW 7 *ON*. Note that the **AUX Input** signal pin should not be be plugged in when the system powers up. Otherwise the ESP32 may not be able to boot correctly.

In this mode:

- **JTAG** functionality is not available
- *Vol-* touch button is available for use with the API

Enable MicroSD Card in 4-wire Mode

DIP SW	Position
1	ON
2	ON
3	OFF
4	OFF
5	OFF
6	OFF
7	OFF
8	n/a

In this mode:

- **JTAG** functionality is not available
- *Vol-* touch button is not available for use with the API
- **AUX Input** detection from the API is not available

Enable JTAG

DIP SW	Position
1	OFF
2	OFF
3	ON
4	ON
5	ON
6	ON
7	ON
8	n/a

In this mode:

- **MicroSD Card** functionality is not available, remove the card from the slot
- *Vol-* touch button is not available for use with the API
- **AUX Input** detection from the API is not available

Using Automatic Upload

Entering of the ESP32 into upload mode may be done in two ways:

- Manually by pressing both **Boot** and **RST** keys and then releasing first **RST** and then **Boot** key.
- Automatically by software performing the upload. The software is using **DTR** and **RTS** signals of the serial interface to control states of **EN**, **IO0** and **IO2** pins of the ESP32. This functionality is enabled by installing jumpers in three headers **JP23**, **JP24** and **JP25**. For details see ESP32 LyraT V4.3 schematic. Remove all jumpers after upload is complete.

Allocation of ESP32 Pins

Several pins ESP32 module are allocated to the on board hardware. Some of them, like GPIO0 or GPIO2, have multiple functions. Please refer to the table below or ESP32 LyraT V4.3 schematic for specific details.

GPIO Pin	Type	Function Definition
SENSOR_VP	I	Audio **Rec** (PB)
SENSOR_VN	I	Audio **Mode** (PB)
IO32	I/O	Audio **Set** (TP)
IO33	I/O	Audio **Play** (TP)
IO27	I/O	Audio **Vol+** (TP)
IO13	I/O	JTAG **MTCK**, MicroSD **D3**, Audio **Vol-** (TP)
IO14	I/O	JTAG **MTMS**, MicroSD **CLK**
IO12	I/O	JTAG **MTDI**, MicroSD **D2**, Aux signal **detect**
IO15	I/O	JTAG **MTDO**, MicroSD **CMD**
IO2	I/O	Automatic Upload, MicroSD **D0**
IO4	I/O	MicroSD **D1**
IO34	I	MicroSD insert **detect**
IO0	I/O	Automatic Upload, I2S **MCLK**
IO5	I/O	I2S **SCLK**
IO25	I/O	I2S **LRCK**
IO26	I/O	I2S **DSDIN**
IO35	I	I2S **ASDOUT**
IO19	I/O	Headphone jack insert **detect**
IO22	I/O	Green LED indicator
IO21	I/O	PA Enable output
IO18	I/O	I2C **SDA**
IO23	I/O	I2C **SCL**

- (TP) - touch pad
- (PB) - push button

Pinout of Extension Headers

There are several pin headers available to connect external components, check the state of particular signal bus or debug operation of ESP32. Note that some signals are shared, see section *Allocation of ESP32 Pins* for details.

UART Header / JP2

	Header Pin
1	3.3V
2	TX
3	RX
4	GND

I2S Header / JP4

	I2C Header Pin	ESP32 Pin
1	MCLK	GPI0
2	SCLK	GPIO5
1	LRCK	GPIO25
2	DSDIN	GPIO26
3	ASDOUT	GPIO35
3	GND	GND

I2C Header / JP5

	I2C Header Pin	ESP32 Pin
1	SCL	GPIO23
2	SDA	GPIO18
3	GND	GND

JTAG Header / JP7

	ESP32 Pin	JTAG Signal
1	MTDO / GPIO15	TDO
2	MTCK / GPIO13	TCK
3	MTDI / GPIO12	TDI
4	MTMS / GPIO14	TMS

Notes of Power Distribution

The board features quite extensive power distribution system. It provides independent power supplies to all critical components. This should reduce noise in the audio signal from digital components and improve overall performance of the components.

Power Supply Separation

The main power supply is 5V and provided by a USB. The secondary power supply is 3.7V and provided by an optional battery. The USB power itself is fed with a dedicated cable, separate from a USB cable used for an application upload. To further reduce noise from the USB, the battery may be used instead of the USB.

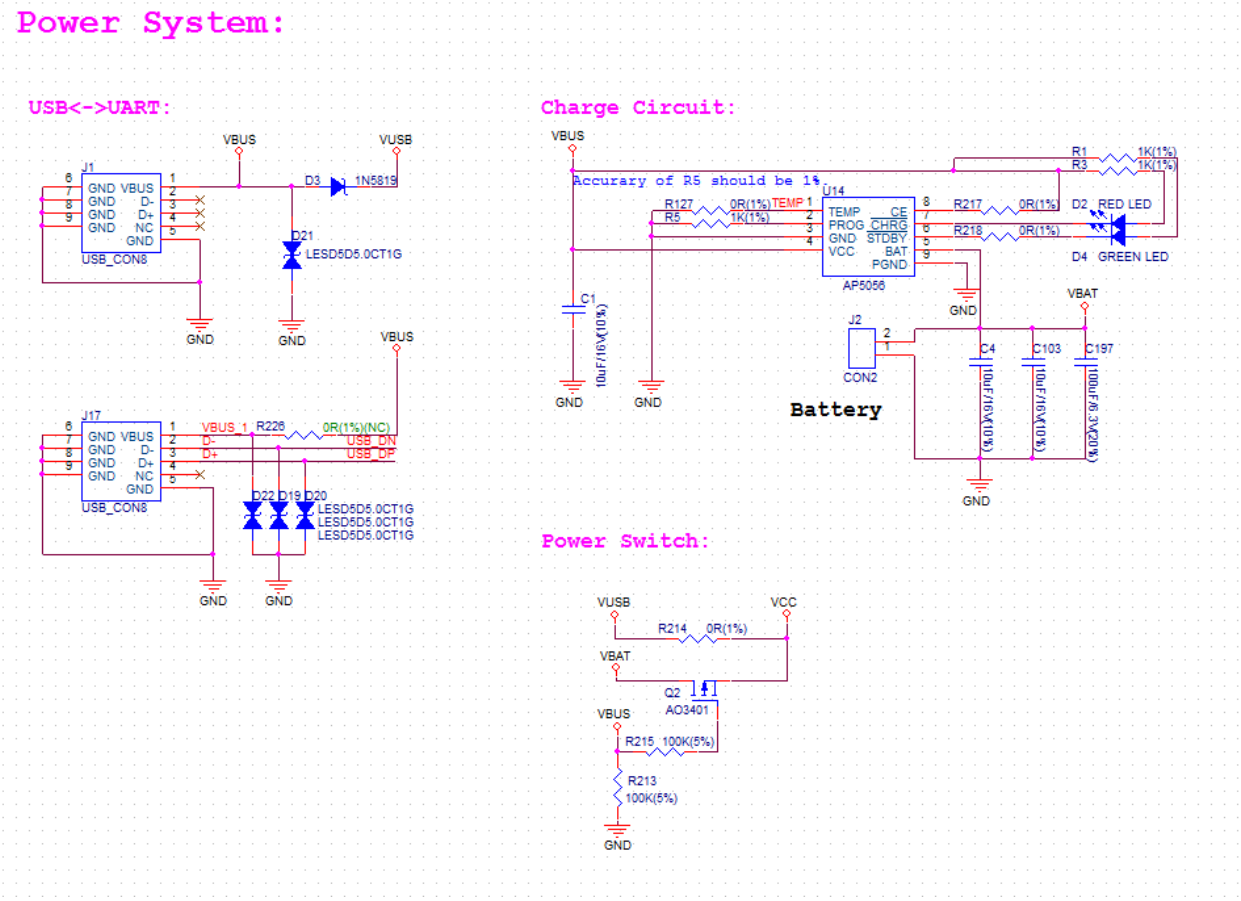


图 16: ESP32 LyraT V4.3 - Power Supply Separation

Three Dedicated LDOs

ESP32 Module

To provide enough current the ESP32, the development board adopts LD1117S33CTR LDO capable to supply the maximum output current of 800mA.

MicroSD Card and **Audio Codec**

Two separate LDOs are provided for the MicorSD Card and the Audio Codec. Both circuits have similar design that includes an inductor and double decoupling capacitors on both the input and output of the LDO.

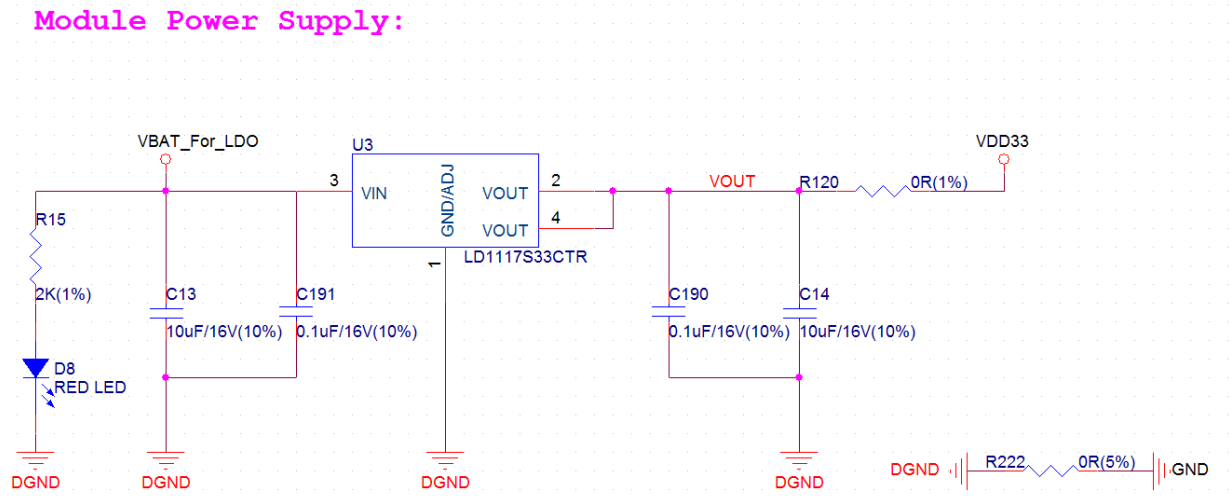


图 17: ESP32 LyraT V4.3 - Dedicated LDO for the ESP32 Module

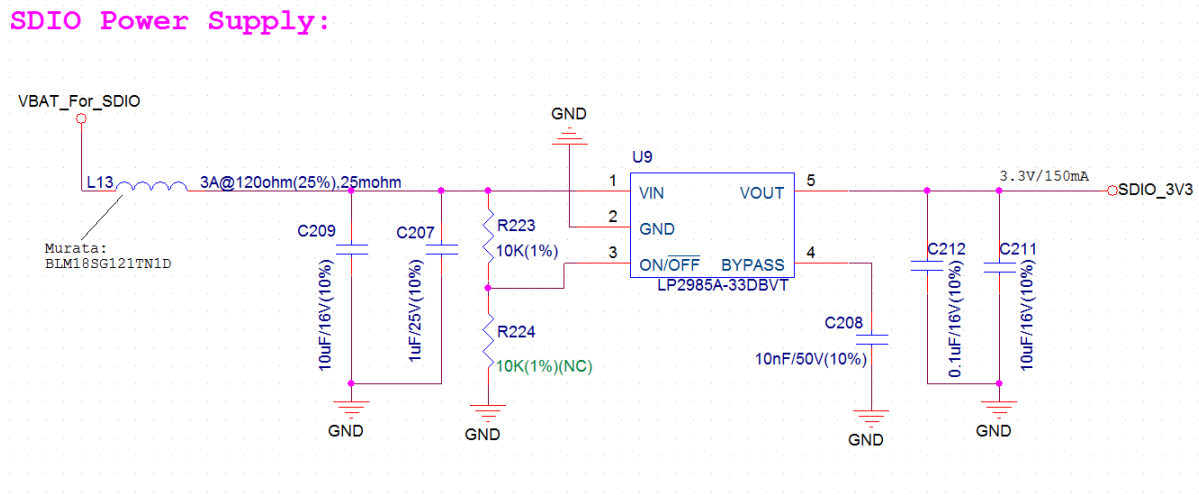


图 18: ESP32 LyraT V4.3 - Dedicated LDO for the MicroSD Card

Separate Power Feed for the PAs

The audio amplifier unit features two NS4150 that require a large power supply for driving external speakers with the maximum output power of 3W. The power is supplied directly to both PAs from the battery or the USB. The development board adds a set of LC circuits at the front of the PA power supply, where L uses 1.5A magnetic beads and C uses 10uF aluminum electrolytic capacitors, to effectively filter out power crosstalk.

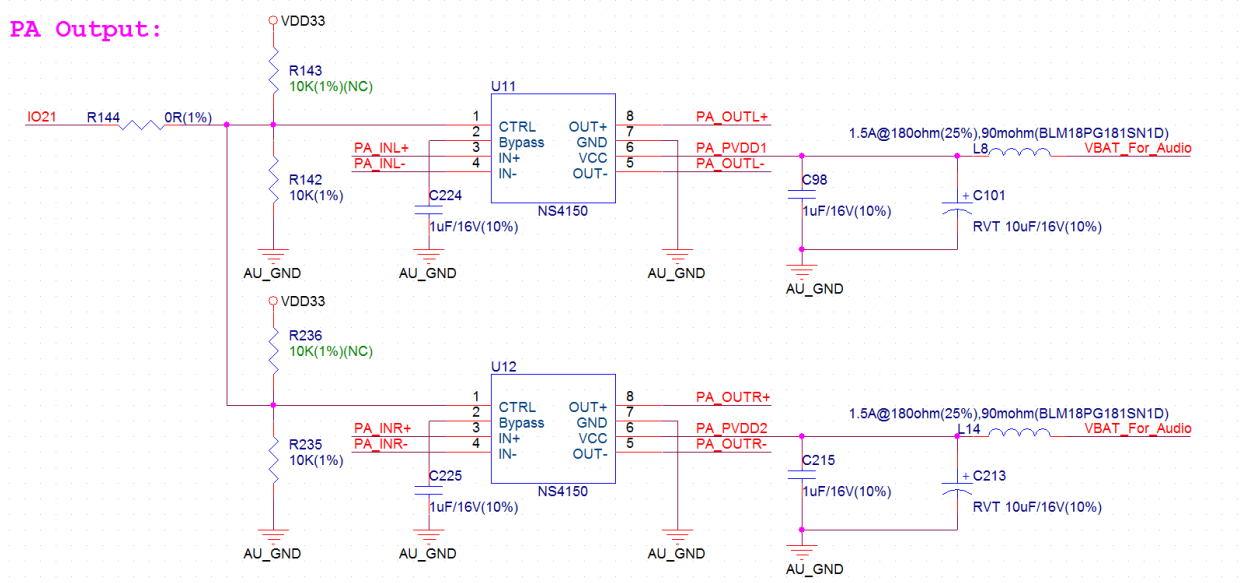


图 19: ESP32 LyraT V4.3 - Power Supply for the PAs

Selecting of the Audio Output

The development board uses two mono Class D amplifier ICs, model number NS4150 with maximum output power of 3W and operating voltage from 3.0V to 5.25V.

The audio input source is the digital-to-analog converter (DAC) output of the ES8388. Audio output supports two external speakers.

An optional audio output is a pair of headphones feed from the same DACs as the amplifier ICs.

To switch between using headphones and speakers, the board provides a digital input signal to detect when a headphone jack is inserted and a digital output signal to enable or disable the amplifier ICs. In other words selection between speakers and headphones is under software control instead of using mechanical contacts that would disconnect speakers once a headphone jack is inserted.

Other Versions of LyraT

- *ESP32-LyraT V4.2 Getting Started Guide*
- *ESP32-LyraT V4 Getting Started Guide*

Related Documents

- ESP32 LyraT V4.3 schematic (PDF)
- *ESP32-LyraT V4.3 入门指南*
- ESP32 Datasheet (PDF)
- ESP32-WROVER-E Datasheet (PDF)
- JTAG Debugging

3.4.5 ESP32-LyraT V4.2 Getting Started Guide

This guide provides users with functional descriptions, configuration options for ESP32-LyraT V4.2 audio development board, as well as how to get started with the ESP32-LyraT board.

The ESP32-LyraT development board is a hardware platform designed for the dual-core ESP32 audio applications, e.g., Wi-Fi or BT audio speakers, speech-based remote controllers, smart-home appliances with audio functionality(ies), etc.

If you like to start using this board right now, go directly to section *Start Application Development*.

What You Need

- 1 × *ESP32 LyraT V4.2 board*
- 2 × Speaker or headphones with a 3.5 mm jack. If you use a speaker, it is recommended to choose one no more than 3 watts, and JST PH 2.0 2-Pin plugs are needed. In case you do not have this type of plug it is also fine to use Dupont female jumper wires during development.
- 2 x Micro-USB 2.0 cables, Type A to Micro B
- 1 × PC loaded with Windows, Linux or Mac OS

Overview

The ESP32-LyraT V4.2 is an audio development board produced by Espressif built around ESP32. It is intended for audio applications, by providing hardware for audio processing and additional RAM on top of what is already onboard of the ESP32 chip. The specific hardware includes:

- **ESP32-WROVER Module**
- **Audio Codec Chip**
- Dual **Microphones** on board
- **Headphone** input
- **2 x 3-watt Speaker** output
- Dual **Auxiliary Input**
- **MicroSD Card** slot (1 line or 4 lines)
- **Six buttons** (2 physical buttons and 4 touch buttons)
- **JTAG** header
- Integrated **USB-UART Bridge Chip**
- Li-ion **Battery-Charge Management**

The block diagram below presents main components of the ESP32-LyraT and interconnections between components.

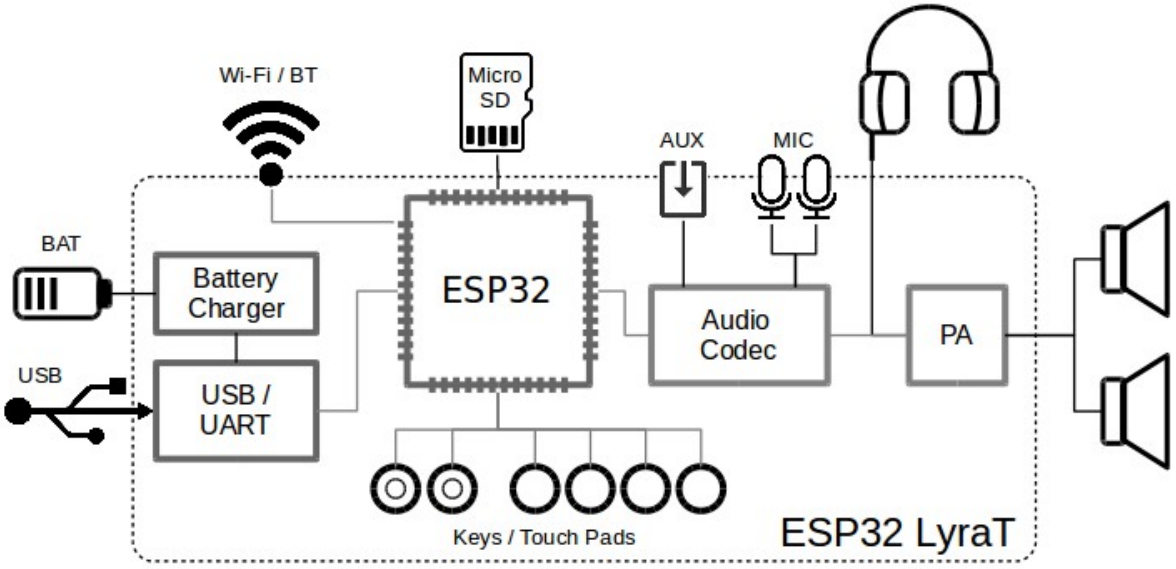


图 20: ESP32-LyraT Block Diagram

Functional Description

The following list and figure describe key components, interfaces and controls of the ESP32-LyraT board.

ESP32-WROVER Module The ESP32-WROVER module contains ESP32 chip to provide Wi-Fi / BT connectivity and data processing power as well as integrates 32 Mbit SPI flash and 32 Mbit PSRAM for flexible data storage.

Green and Red LEDs Two general purpose LEDs controlled by **ESP32-WROVER Module** to indicate certain operation states of the audio application using dedicated API.

Function DIP Switch Used to configure function of GPIO12 to GPIO15 pins that are shared between devices, primarily between **JTAG Header** and **MicroSD Card**. By default, the **MicroSD Card** is enabled with all switches in *OFF* position. To enable the **JTAG Header** instead, switches in positions 3, 4, 5 and 6 should be put *ON*. If **JTAG** is not used and **MicroSD Card** is operated in the one-line mode, then GPIO12 and GPIO13 may be assigned to other functions. Please refer to ESP32 LyraT V4.2 schematic for more details.

JTAG Header Provides access to the **JTAG** interface of **ESP32-WROVER Module**. It may be used for debugging, application upload, as well as implementing several other functions, e.g., Application Level Tracing. See *JTAG Header / JP7* for pinout details. Before using **JTAG** signals to the header, **Function DIP Switch** should be enabled. Please note that when **JTAG** is in operation, **MicroSD Card** cannot be used and should be disconnected because some of JTAG signals are shared by both devices.

UART Header Serial port: provides access to the serial TX/RX signals between **ESP32-WROVER Module** and **USB-UART Bridge Chip**.

I2C Header Provides access to the I2C interface. Both **ESP32-WROVER Module** and **Audio Codec Chip** are connected to this interface. See *I2C Header / JP5* for pinout details.

MicroSD Card The development board supports a MicroSD card in SPI/1-bit/4-bit modes, and can store or play audio files in the MicroSD card. See *MicroSD Card / J5* for pinout details. Note that **JTAG** cannot be used and should be disconnected by setting **Function DIP Switch** when **MicroSD Card** is in operation, because some of signals are shared by both devices.

I2S Header Provides access to the I2S interface. Both **ESP32-WROVER Module** and **Audio Codec Chip** are connected to this interface. See *I2S Header / JP4* for pinout details.

Left Microphone Onboard microphone connected to IN1 of the **Audio Codec Chip**.

AUX Input Auxiliary input socket connected to IN2 (left and right channel) of the **Audio Codec Chip**. Use a 3.5 mm stereo jack to connect to this socket.

Headphone Output Output socket to connect headphones with a 3.5 mm stereo jack.

Right Microphone Onboard microphone connected to IN1 of the **Audio Codec Chip**.

Left Speaker Output Output socket to connect a speaker. The 4-ohm and 3-watt speaker is recommended. The pins have a 2.00 mm / 0.08” pitch.

Right Speaker Output Output socket to connect a speaker. The 4-ohm and 3-watt speaker is recommended. The pins have a 2.00 mm / 0.08” pitch.

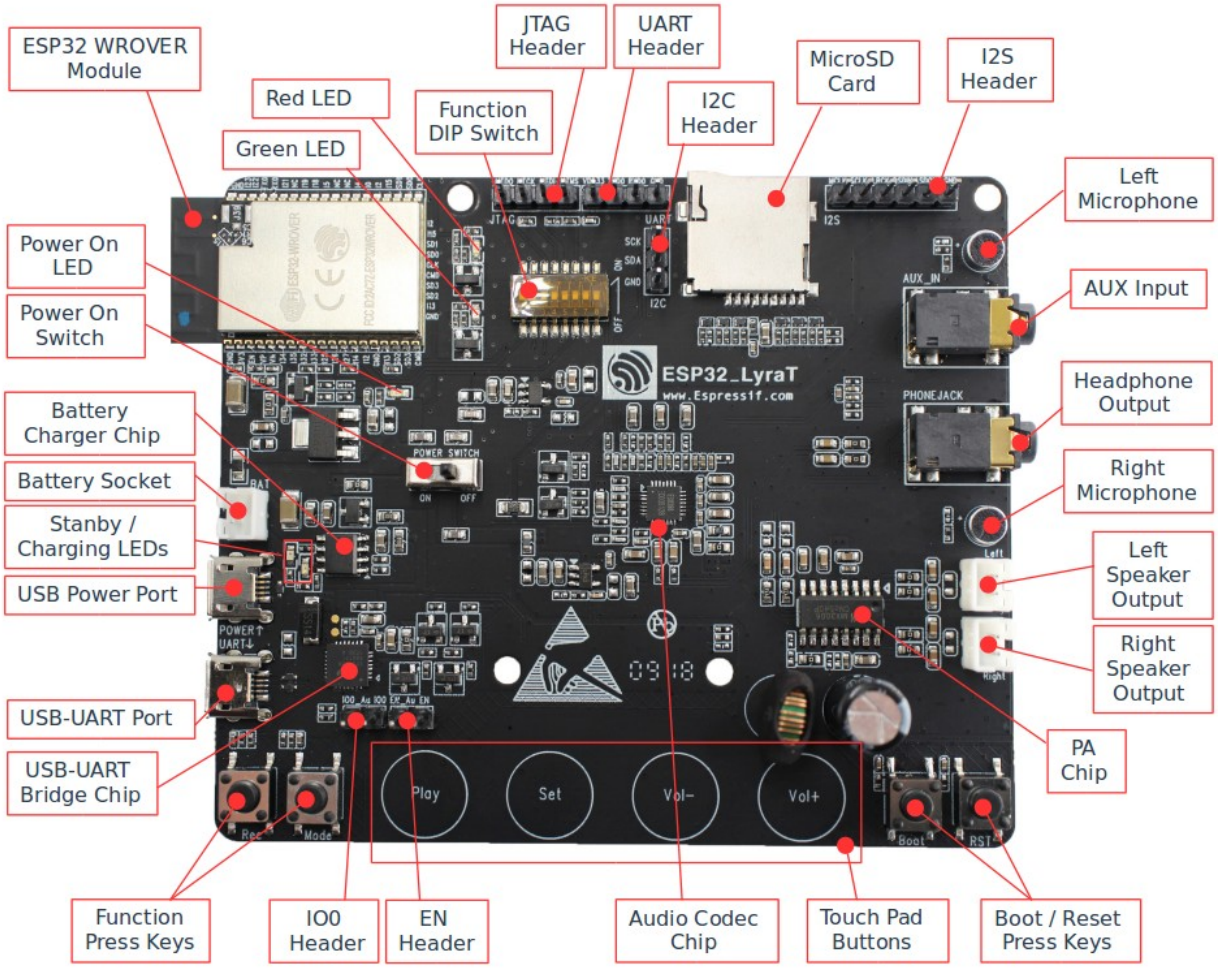


图 21: ESP32-LyraT V4.2 Board Layout

PA Chip A power amplifier used to amplify stereo audio signal from the **Audio Codec Chip** for driving two speakers.

Boot/Reset Press Keys Boot: holding down the **Boot** button and momentarily pressing the **Reset** button initiates the firmware upload mode. Then user can upload firmware through the serial port. Reset: pressing this button alone resets the system.

Touch Pad Buttons Four touch pads labeled *Play*, *Sel*, *Vol+* and *Vol-*. They are routed to **ESP32-WROVER Module** and intended for development and testing of a UI for audio applications using dedicated API.

Audio Codec Chip The Audio Codec Chip, ES8388, is a low power stereo audio codec with a headphone amplifier. It consists of 2-channel ADC, 2-channel DAC, microphone amplifier, headphone amplifier, digital sound effects, analog mixing and gain functions. It is interfaced with **ESP32-WROVER Module** over I2S and I2S buses to provide audio processing in hardware independently from the audio application.

EN Header Install a jumper on this header to enable automatic loading of application to the ESP32. Install or remove jumpers together on both IO0 and EN headers.

IO0 Header Install a jumper on this header to enable automatic loading of application to the ESP32. Install or remove jumpers together on both IO0 and EN headers.

Function Press Keys Two key labeled *Rec* and *Mode*. They are routed to **ESP32-WROVER Module** and intended for developing and testing a UI for audio applications using dedicated API.

USB-UART Bridge Chip A single chip USB-UART bridge provides up to 1 Mbps transfers rate.

USB-UART Port Functions as the communication interface between a PC and the ESP32 module.

USB Power Port Provides the power supply for the board.

Standby / Charging LEDs The **Standby** green LED indicates that power has been applied to the **Micro USB Port**. The **Charging** red LED indicates that a battery connected to the **Battery Socket** is being charged.

Battery Charger Chip Constant current & constant voltage linear charger for single cell lithium-ion batteries AP5056. Used for charging of a battery connected to the **Battery Socket** over the **Micro USB Port**.

Power On Switch Power on/off knob: toggling it to the left powers the board on; toggling it to the right powers the board off.

Battery Socket Two pins socket to connect a single cell Li-ion battery.

Power On LED Red LED indicating that **Power On Switch** is turned on.

注解： The **Power On Switch** does not affect / disconnect the Li-ion battery charging.

Hardware Setup Options

There are a couple of options to change the hardware configuration of the ESP32-LyraT board. The options are selectable with the **Function DIP Switch**.

Enable MicroSD Card in 1-wire Mode

DIP SW	Position
1	OFF
2	OFF
3	OFF
4	OFF
5	OFF
6	OFF
7	OFF [1]
8	n/a

1. **AUX Input** detection may be enabled by toggling the DIP SW 7 *ON*

In this mode:

- **JTAG** functionality is not available
- *Vol-* touch button is available for use with the API

Enable MicroSD Card in 4-wire Mode

DIP SW	Position
1	ON
2	ON
3	OFF
4	OFF
5	OFF
6	OFF
7	OFF
8	n/a

In this mode:

- **JTAG** functionality is not available
- *Vol-* touch button is not available for use with the API

- **AUX Input** detection from the API is not available

Enable JTAG

DIP SW	Position
1	OFF
2	OFF
3	ON
4	ON
5	ON
6	ON
7	ON
8	n/a

In this mode:

- **MicroSD Card** functionality is not available, remove the card from the slot
- *Vol-* touch button is not available for use with the API
- **AUX Input** detection from the API is not available

Allocation of ESP32 Pins

Several pins / terminals of ESP32 modules are allocated to the on board hardware. Some of them, like GPIO0 or GPIO2, have multiple functions. Please refer to the tables below or ESP32 LyraT V4.2 schematic for specific details.

Red / Green LEDs

	ESP32 Pin	LED Color
1	GPIO19	Red LED
2	GPIO22	Green LED

Touch Pads

	ESP32 Pin	Touch Pad Function
1	GPIO33	Play
2	GPIO32	Set
3	GPIO13	Vol- [1]
4	GPIO27	Vol+

1. *Vol-* function is not available if **JTAG** is used. It is also not available for the **MicroSD Card** configured to operate in 4-wire mode.

MicroSD Card / J5

	ESP32 Pin	MicroSD Signal
1	MTDI / GPIO12	DATA2
2	MTCK / GPIO13	CD / DATA3
3	MTDO / GPIO15	CMD
4	MTMS / GPIO14	CLK
5	GPIO2	DATA0
6	GPIO4	DATA1
7	GPIO21	CD

UART Header / JP2

	Header Pin
1	3.3V
2	TX
3	RX
4	GND

EN and IO0 Headers / JP23 and J24

	ESP32 Pin	Header Pin
1	n/a	EN_Auto
2	EN	EN

	ESP32 Pin	Header Pin
1	n/a	IO0_Auto
2	GPIO0	IO0

I2S Header / JP4

	I2C Header Pin	ESP32 Pin
1	MCLK	GPI0
2	SCLK	GPIO5
1	LRCK	GPIO25
2	DSDIN	GPIO26
3	ASDOUT	GPIO35
3	GND	GND

I2C Header / JP5

	I2C Header Pin	ESP32 Pin
1	SCL	GPIO23
2	SDA	GPIO18
3	GND	GND

JTAG Header / JP7

	ESP32 Pin	JTAG Signal
1	MTDO / GPIO15	TDO
2	MTCK / GPIO13	TCK
3	MTDI / GPIO12	TDI
4	MTMS / GPIO14	TMS

Function DIP Switch / JP8

	Switch OFF	Switch ON
1	GPIO12 not allocated	MicroSD Card 4-wire
2	Touch *Vol-* enabled	MicroSD Card 4-wire
3	MicroSD Card	JTAG
4	MicroSD Card	JTAG
5	MicroSD Card	JTAG
6	MicroSD Card	JTAG
7	MicroSD Card 4-wire	AUX IN detect [1]
8	not used	not used

1. The **AUX Input** signal pin should not be be plugged in when the system powers up. Otherwise the ESP32 may not be able to boot correctly.

Start Application Development

Before powering up the ESP32-LyraT, please make sure that the board has been received in good condition with no obvious signs of damage.

Initial Setup

Prepare the board for loading of the first sample application:

1. Install jumpers on **IO0** and **EN** headers to enable automatic application upload. If there are no jumpers then upload may be triggered using **Boot / RST** buttons.
2. Connect speakers to the **Right** and **Left Speaker Output**.Connecting headphones to the **Headphone Output** is an option.
3. Plug in the Micro-USB cables to the PC and to **both USB ports** of the ESP32 LyraT.
4. The **Standby LED** (green) should turn on. Assuming that a battery is not connected, the **Charging LED** will blink every couple of seconds.
5. Toggle left the **Power On Switch**.
6. The red **Power On LED** should turn on.

If this is what you see on the LEDs, the board should be ready for application upload. Now prepare the PC by loading and configuring development tools what is discussed in the next section.

Develop Applications

Once the board is initially set up and checked, you can start preparing the development tools. The Section *Installation Step by Step* will walk you through the following steps:

- **Set up ESP-IDF** to get a common development framework for the ESP32 (and ESP32-S2) chips in C language;
- **Get ESP-ADF** to install the API specific to audio applications;
- **Set up env** to make the framework aware of the audio specific API;
- **Start a Project** that will provide a sample audio application for the board;
- **Connect Your Device** to prepare the application for loading;
- **Build the Project** to finally run the application and play some music.

Related Documents

- ESP32 LyraT V4.2 schematic (PDF)
- ESP32 Datasheet (PDF)
- ESP32-WROVER Datasheet (PDF)
- JTAG Debugging
- *ESP32-LyraT V4 Getting Started Guide*

3.4.6 ESP32-LyraT V4 Getting Started Guide

This guide provide users with functional descriptions, configuration options for ESP32-LyraT V4 audio development board, as well as how to get started with ESP32-LyraT board.

The ESP32-LyraT development board is a hardware platform specifically designed for the dual-core ESP32 audio applications, e.g., Wi-Fi or BT audio speakers, speech-based remote controllers, smart-home appliances with audio functionality(ies), etc.

If you like to start using this board right now, go directly to section *Start Application Development*.

What You Need

- 1 × *ESP32-LyraT V4 board*
- 2 × Speaker or headphones with a 3.5 mm jack. If you use a speaker, it is recommended to choose one no more than 3 watts, and JST PH 2.0 2-Pin plugs are needed. In case you do not have this type of plug it is also fine to use Dupont female jumper wires during development.
- 1 × Micro USB 2.0 Cable, Type A to Micro B

- 1 × PC loaded with Windows, Linux or Mac OS

Overview

The ESP32-LyraT V4 is an audio development board produced by Espressif built around ESP32. It is intended for audio applications, by providing hardware for audio processing and additional RAM on top of what is already onboard of the ESP32 chip. The specific hardware includes:

- **ESP32-WROVER Module**
- **Audio Codec Chip**
- Dual **Microphones** on board
- **Headphone** input
- **2 x 3 Watt Speaker** output
- Dual **Auxiliary Input**
- **MicroSD Card** slot (1 line or 4 lines)
- **6 buttons** (2 physical buttons and 4 touch buttons)
- **JTAG** header
- Integrated **USB-UART Bridge Chip**
- Li-ion **Battery-Charge Management**

Block diagram below presents main components of the ESP32-LyraT and interconnections between components.

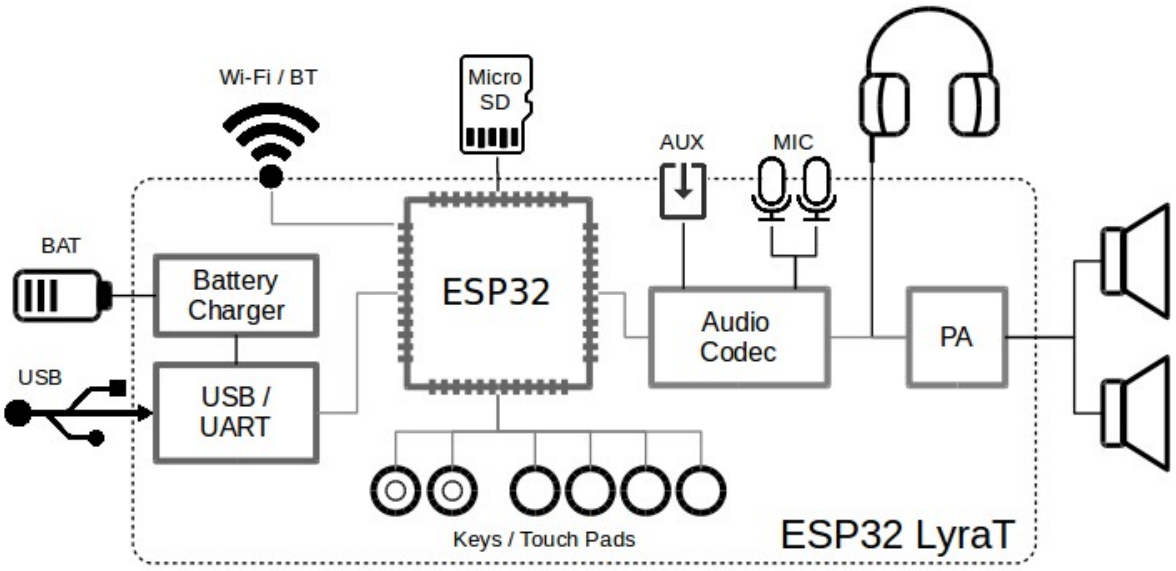


图 22: ESP32-LyraT block diagram

Functional Description

The following list and figure below describe key components, interfaces and controls of the ESP32-LyraT board.

ESP32-WROVER Module The ESP32-WROVER module contains ESP32 chip to provide Wi-Fi / BT connectivity and data processing power as well as integrates 32 Mbit SPI flash and 32 Mbit PSRAM for flexible data storage.

Green and Red LEDs Two general purpose LEDs controlled by **ESP32-WROVER Module** to indicate certain operation states of the audio application using dedicated API.

Function DIP Switch Used to configure function of GPIO12 to GPIO15 pins that are shared between devices, primarily between **JTAG Header** and **MicroSD Card**. By default **MicroSD Card** is enabled with all switches in *OFF* position. To enable **JTAG Header** instead, switches in positions 3, 4, 5 and 6 should be put *ON*. If **JTAG** is not used and **MicroSD Card** is operated in one-line mode, then GPIO12 and GPIO13 may be assigned to other functions. Please refer to ESP32 LyraT V4 schematic for more details.

JTAG Header Provides access to the **JTAG** interface of **ESP32-WROVER Module**. May be used for debugging, application upload, as well as implementing several other functions, e.g., Application Level Tracing. See *JTAG Header / JP7* for pinout details. Before using **JTAG** signals to the header, **Function DIP Switch** should be enabled. Please note that when **JTAG** is in operation, **MicroSD Card** cannot be used and should be disconnected because some of JTAG signals are shared by both devices.

UART Header Serial port provides access to the serial TX/RX signals between **ESP32-WROVER Module** and **USB-UART Bridge Chip**.

I2C Header Provides access to the I2C interface. Both **ESP32-WROVER Module** and **Audio Codec Chip** are connected to this interface. See *I2C Header / JP5* for pinout details.

MicroSD Card The development board supports a MicroSD card in SPI/1-bit/4-bit modes, and can store or play audio files in the MicroSD card. See *MicroSD Card / J5* for pinout details. Note that **JTAG** cannot be used and should be disconnected by setting **Function DIP Switch** when **MicroSD Card** is in operation, because some of the signals are shared by both devices.

I2S Header Provides access to the I2S interface. Both **ESP32-WROVER Module** and **Audio Codec Chip** are connected to this interface. See *I2S Header / JP4* for pinout details.

Left Microphone Onboard microphone connected to IN1 of the **Audio Codec Chip**.

AUX Input Auxiliary input socket connected to IN2 (left and right channels) of the **Audio Codec Chip**. Use a 3.5 mm stereo jack to connect to this socket.

Headphone Output Output socket to connect headphones with a 3.5 mm stereo jack.

Right Microphone Onboard microphone connected to IN1 of the **Audio Codec Chip**.

Left Speaker Output Output socket to connect a speaker. The 4-ohm and 3-watt speaker is recommended. The pins have a 2.00 mm / 0.08” pitch.

Right Speaker Output Output socket to connect a speaker. The 4-ohm and 3-watt speaker is recommended. The pins have a 2.00 mm / 0.08” pitch.

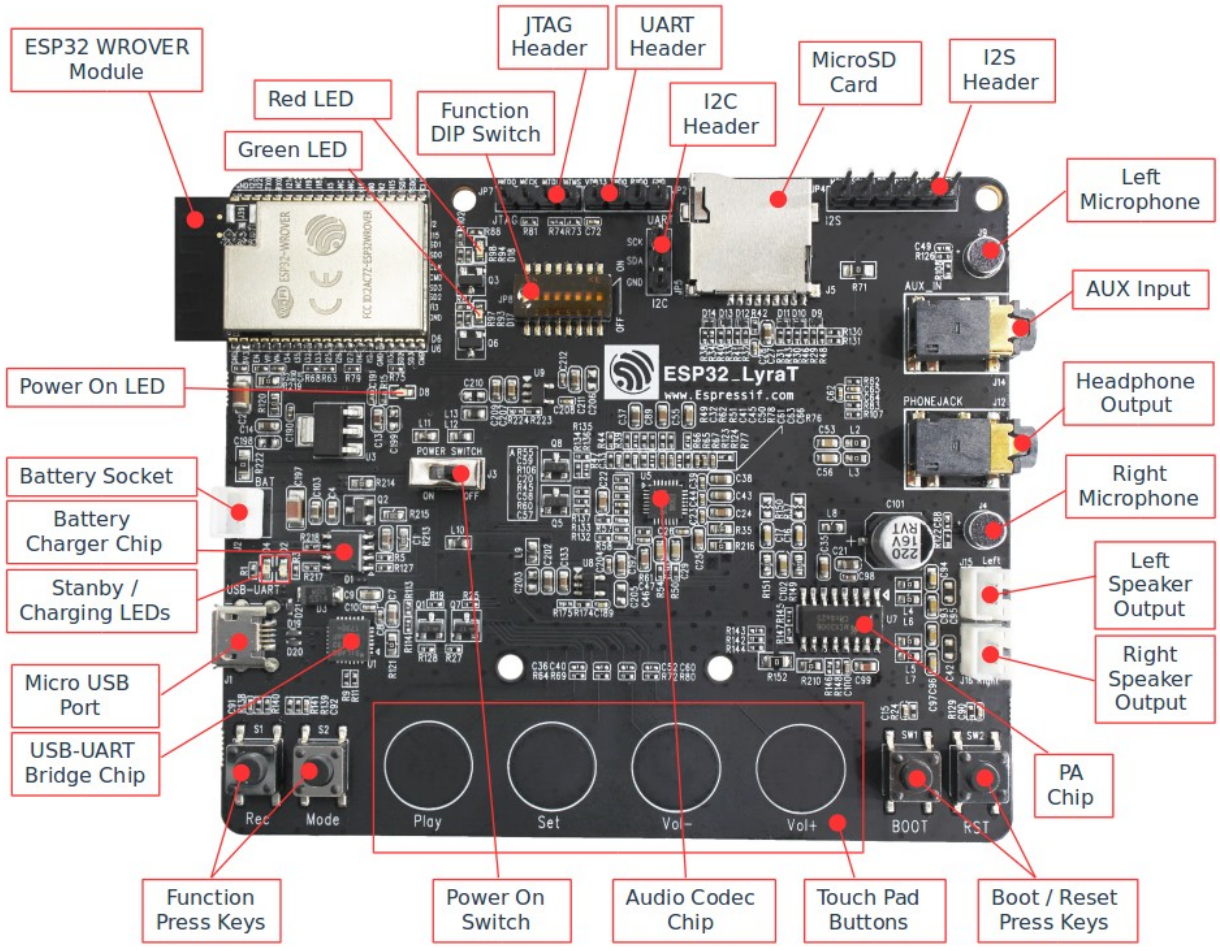


图 23: ESP32 LyraT V4 board layout

PA Chip A power amplifier used to amplify stereo audio signal from the **Audio Codec Chip** for driving two speakers.

Boot/Reset Press Keys Boot: holding down the **Boot** button and momentarily pressing the **Reset** button initiates the firmware upload mode. Then user can upload firmware through the serial port. Reset: pressing this button alone resets the system.

Touch Pad Buttons Four touch pads labeled *Play*, *Sel*, *Vol+* and *Vol-*. They are routed to **ESP32-WROVER Module** and intended for development and testing of a UI for audio applications using dedicated API.

Audio Codec Chip The Audio Codec Chip, ES8388, is a low-power stereo audio codec with headphone amplifier. It consists of 2-channel ADC, 2-channel DAC, microphone amplifier, headphone amplifier, digital sound effects, analog mixing and gain functions. It is interfaced with **ESP32-WROVER Module** over I2S and I2S buses to provide audio processing in hardware independently from the audio application.

Function Press Keys Two key labeled *Rec* and *Mode*. They are routed to **ESP32-WROVER Module** and intended for developing and testing a UI for audio applications using dedicated API.

USB-UART Bridge Chip A single chip USB-UART bridge provides up to 1 Mbps transfer rate.

Micro USB Port USB interface. It functions as the power supply for the board and the communication interface between a PC and the ESP32 module.

Standby / Charging LEDs The **Standby** green LED indicates that power has been applied to the **Micro USB Port**. The **Charging** red LED indicates that a battery connected to the **Battery Socket** is being charged.

Battery Charger Chip Constant current & constant voltage linear charger for single cell lithium-ion batteries AP5056. Used for charging of a battery connected to the **Battery Socket** over the **Micro USB Port**.

Power On Switch Power on/off knob: toggling it to the left powers the board on; toggling it to the right powers the board off.

Battery Socket Two pins socket to connect a single cell Li-ion battery.

Power On LED Red LED indicating that **Power On Switch** is turned on.

注解: The **Power On Switch** does not affect / disconnect the Li-ion battery charging.

Hardware Setup Options

There are couple of options to change the hardware configuration of the ESP32-LyraT board. The options are selectable with the **Function DIP Switch**.

Enable MicroSD Card in 1-wire Mode

DIP SW	Position
1	OFF
2	OFF
3	OFF
4	OFF
5	OFF
6	OFF
7	OFF [1]
8	n/a

1. **AUX Input** detection may be enabled by toggling the DIP SW 7 *ON*

In this mode:

- **JTAG** functionality is not available
- *Vol-* touch button is available for use with the API

Enable MicroSD Card in 4-wire Mode

DIP SW	Position
1	ON
2	ON
3	OFF
4	OFF
5	OFF
6	OFF
7	OFF
8	n/a

In this mode:

- **JTAG** functionality is not available
- *Vol-* touch button is not available for use with the API
- **AUX Input** detection from the API is not available

Enable JTAG

DIP SW	Position
1	OFF
2	OFF
3	ON
4	ON
5	ON
6	ON
7	ON
8	n/a

In this mode:

- **MicroSD Card** functionality is not available, remove the card from the slot
- *Vol-* touch button is not available for use with the API
- **AUX Input** detection from the API is not available

Allocation of ESP32 Pins

Several pins / terminals of ESP32 modules are allocated to the onboard hardware. Some of them, like GPIO0 or GPIO2, have multiple functions. Please refer to tables below or ESP32 LyraT V4 schematic for specific details.

Red / Green LEDs

	ESP32 Pin	LED Color
1	GPIO19	Red LED
2	GPIO22	Green LED

Touch Pads

	ESP32 Pin	Touch Pad Function
1	GPIO33	Play
2	GPIO32	Set
3	GPIO13	Vol- [1]
4	GPIO27	Vol+

1. *Vol-* function is not available if **JTAG** is used. It is also not available for the **MicroSD Card** configured to operate in 4-wire mode.

MicroSD Card / J5

	ESP32 Pin	MicroSD Signal
1	MTDI / GPIO12	DATA2
2	MTCK / GPIO13	CD / DATA3
3	MTDO / GPIO15	CMD
4	MTMS / GPIO14	CLK
5	GPIO2	DATA0
6	GPIO4	DATA1
7	GPIO21	CD

UART Header / JP2

	Header Pin
1	3.3V
2	TX
3	RX
4	GND

I2S Header / JP4

	I2C Header Pin	ESP32 Pin
1	MCLK	GPI0
2	SCLK	GPIO5
1	LRCK	GPIO25
2	DSDIN	GPIO26
3	ASDOUT	GPIO35
3	GND	GND

I2C Header / JP5

	I2C Header Pin	ESP32 Pin
1	SCL	GPIO23
2	SDA	GPIO18
3	GND	GND

JTAG Header / JP7

	ESP32 Pin	JTAG Signal
1	MTDO / GPIO15	TDO
2	MTCK / GPIO13	TCK
3	MTDI / GPIO12	TDI
4	MTMS / GPIO14	TMS

Function DIP Switch / JP8

	Switch OFF	Switch ON
1	GPIO12 not allocated	MicroSD Card 4-wire
2	Touch *Vol-* enabled	MicroSD Card 4-wire
3	MicroSD Card	JTAG
4	MicroSD Card	JTAG
5	MicroSD Card	JTAG
6	MicroSD Card	JTAG
7	MicroSD Card 4-wire	AUX IN detect [1]
8	not used	not used

1. The **AUX Input** signal pin should not be plugged in when the system powers up. Otherwise the ESP32 may not be able to boot correctly.

Start Application Development

Before powering up the ESP32-LyraT, please make sure that the board has been received in good condition with no obvious signs of damage.

Initial Setup

Prepare the board for loading of the first sample application:

1. Connect speakers to the **Right** and **Left Speaker Output**. Optionally connect headphones to the **Headphone Output**.
2. Plug in the Micro-USB cable to the PC and to the **Micro USB Port** of the ESP32-LyraT.
3. The **Standby LED** (green) should turn on. Assuming that a battery is not connected, the **Charging LED** will momentarily blink every couple of seconds.
4. Toggle left the **Power On Switch**.
5. The red **Power On LED** should turn on.

If this is what you see on the LEDs, the board should be ready for application upload. Now prepare the PC by loading and configuring development tools what is discussed in the next section.

Develop Applications

Once the board is initially set up and checked, you can start preparing the development tools. The Section *Installation Step by Step* will walk you through the following steps:

- **Set up ESP-IDF** to get a common development framework for the ESP32 (and ESP32-S2) chips in C language;
- **Get ESP-ADF** to install the API specific to audio applications;
- **Set up env** to make the framework aware of the audio specific API;
- **Start a Project** that will provide a sample audio application for the board;
- **Connect Your Device** to prepare the application for loading;
- **Build the Project** to finally run the application and play some music.

Related Documents

- ESP32 LyraT V4 schematic (PDF)
- ESP32 Datasheet (PDF)
- ESP32-WROVER Datasheet (PDF)
- JTAG Debugging

3.4.7 ESP32-LyraTD-MSC V2.2 入门指南

[English]

本指南旨在向用户介绍 ESP32-LyraTD-MSC V2.2 音频开发板的功能、配置选项以及如何快速入门。

ESP32-LyraTD-MSC 是一款面向智能音箱和 AI 应用程序的硬件平台，支持声学回声消除 (Acoustic Echo Cancelation, AEC)、自动语音识别 (Automatic Speech Recognition, ASR)、打断唤醒 (Wake-up Interrupt) 以及语音互动 (Voice Interaction) 功能。

准备工作

- 1 × *ESP32-LyraTD-MSC V2.2 board*
- 2 x 扬声器或 3.5 mm 的耳机（若使用扬声器，建议功率不超过 3 瓦特，另外需要接口为 JST PH 2.0 毫米 2 针的插头，若没有此插头，开发过程中可替换为杜邦母跳线）
- 2 x Micro-USB 2.0 数据线（Type A 转 Micro B）
- 1 × PC（Windows、Linux 或 Mac OS）

如需立即使用此开发板，请直接前往正式开始开发。

概述

ESP32-LyraTD-MSC V2.2 是一款基于 乐鑫 ESP32 芯片的音频开发板，专为智能音箱和 AI 应用程序打造，除 ESP32 芯片原有的硬件外，还增添了数字信号处理、麦克风阵列以及扩展 RAM。

ESP32-LyraTD-MSC V2.2 由上板和下板两部分组成，上板 (B) 集成三麦克风阵列、功能按键和 LED 灯，下板 (A) 集成 ESP32-WROVER-E 模组、MicroSemi Digital Signal Processing (DSP) 芯片以及电源管理模块。

具体包括以下硬件：

- ESP32-WROVER-E 模组
- DSP 芯片
- 3 颗支持远场语音唤醒功能的麦克风
- 2 个 3-watt 扬声器输出
- 耳机输出
- MicroSD 卡槽（一线模式或四线模式）
- 围绕开发板边缘分布的 12 个 LED 灯，可独立控制
- 6 个功能按键，可配置不同功能
- 多个端口：I2S、I2C、SPI、JTAG
- 集成 USB-UART 桥接芯片

图 24: ESP32-LyraTD-MSC 侧面图

- 锂电池充电管理

下图展示的是 ESP32-LyraTD-MSC 的主要组件以及组件之间的连接方式。

组件

以下列表和图片仅涉及 ESP32-LyraTD-MSC 的主要组件、接口和控制方式，只展示目前所需的信息，更多细节请参阅相关文档 中的原理图。

ESP32-WROVER-E 模组 ESP32-WROVER-E 模组采用 ESP32 芯片，可实现 Wi-Fi/蓝牙连接和数据处理，同时集成 4 MB 外部 SPI flash 和 8 MB SPI PSRAM，可实现灵活的数据存储。

DSP 芯片 ZL38063 DSP 芯片用于自动语音识别应用程序，可从外部麦克风阵列获取音频数据，并通过自身 DAC 端口输出音频信号。

耳机输出 输出插槽可连接 3.5 mm 立体声耳机。

> **注意：** 该插槽可接入移动电话耳机，只与 OMPT 标准耳机兼容，与 CTIA 耳机不兼容。更多耳机标准信息请访问维基百科 Phone connector (audio) 词条。

左侧扬声器输出 音频输出插槽，采用 2.00 mm / 0.08” 排针间距，建议连接 4 欧姆 3 瓦特扬声器。

右侧扬声器输出 音频输出插槽，采用 2.00 mm / 0.08” 排针间距，建议连接 4 欧姆 3 瓦特扬声器。

USB-UART 接口 作为 PC 和 ESP32-WROVER-E 模组之间的通信接口。

USB 供电接口 为开发板供电。

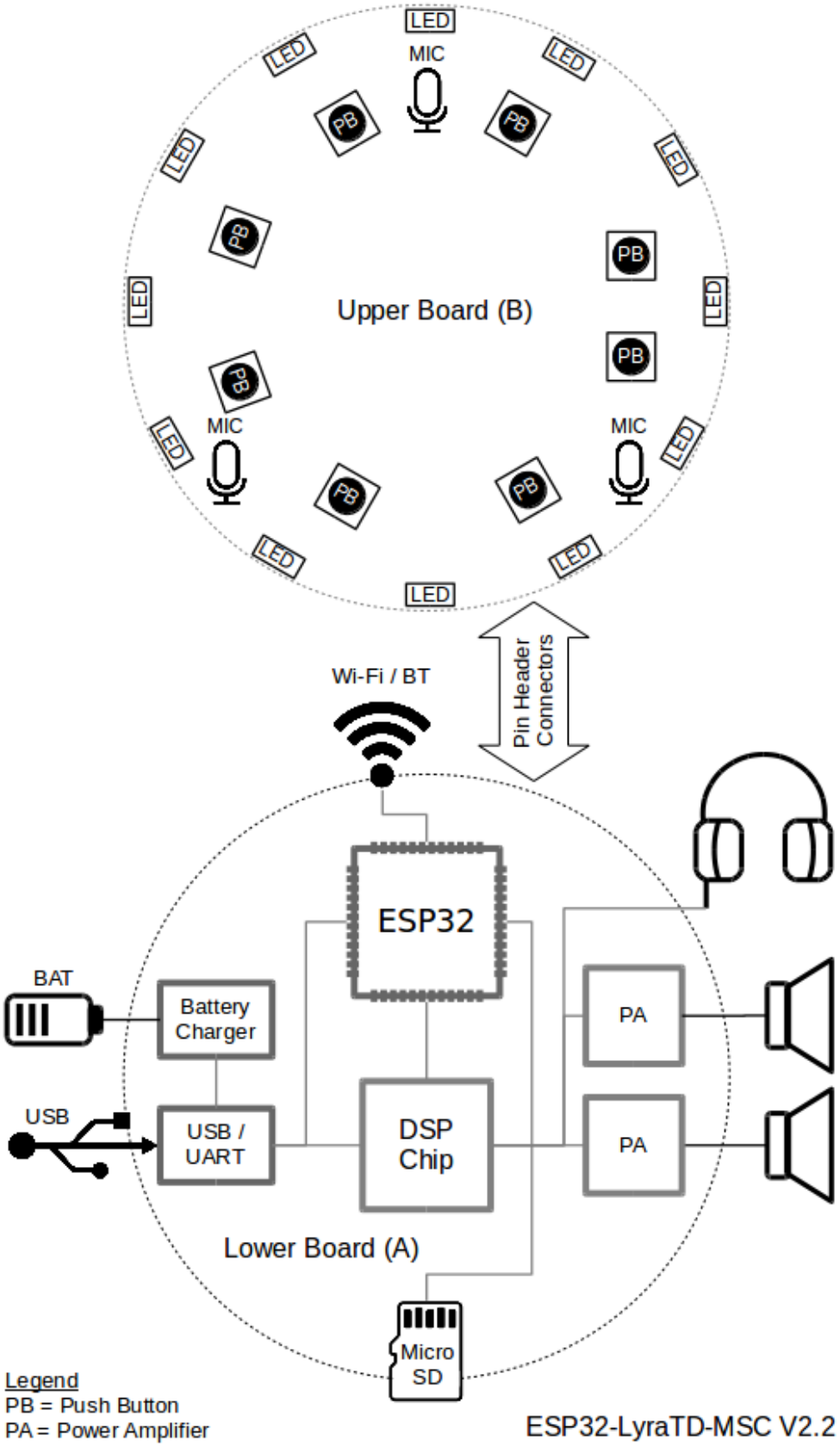


图 25: ESP32-LyraTD-MSC 开发板框图

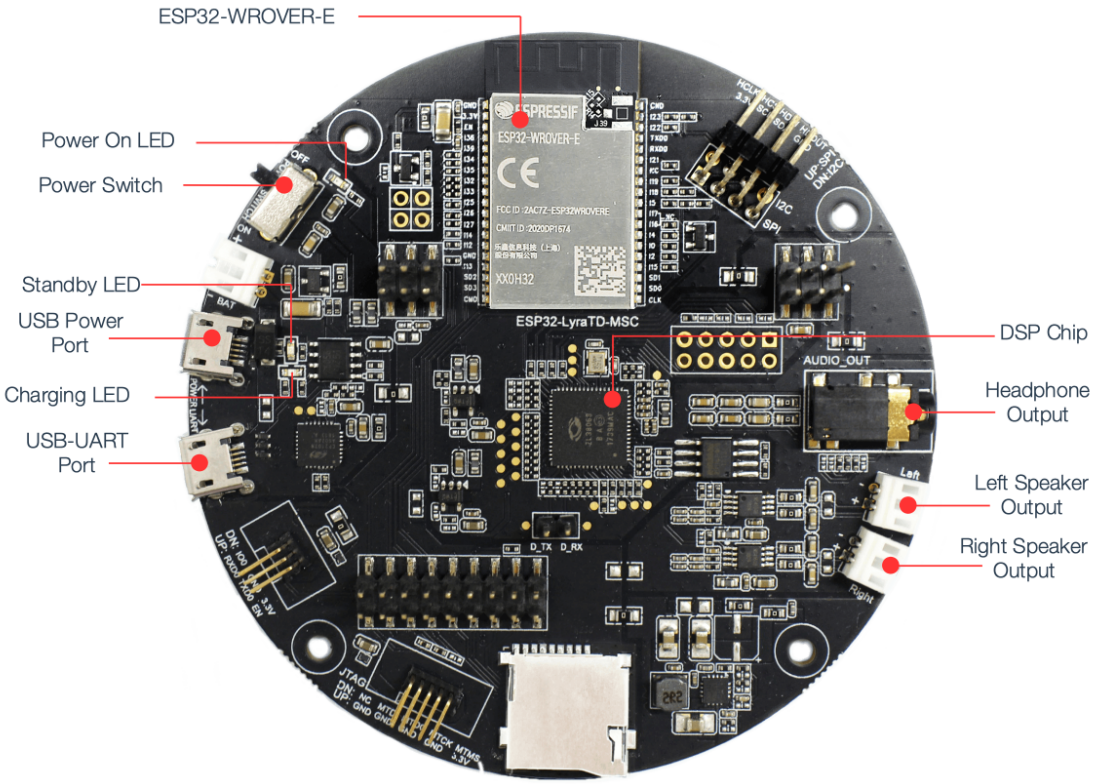


图 26: ESP32-LyraTD-MSC V2.2 下板 (A) 组件图

待机/充电指示灯 绿色 **待机**指示灯亮起时，表示电源已接入 **Micro USB 接口**；红色 **充电**指示灯亮起时，表示连接至 **电池接口**上的电池正在充电。

电源开关 电源开/关按钮：向右拨动按钮则开发板电源开启，向左拨动则电源关闭。

电源指示灯 红色指示灯亮起表示 **电源开关**已开启。

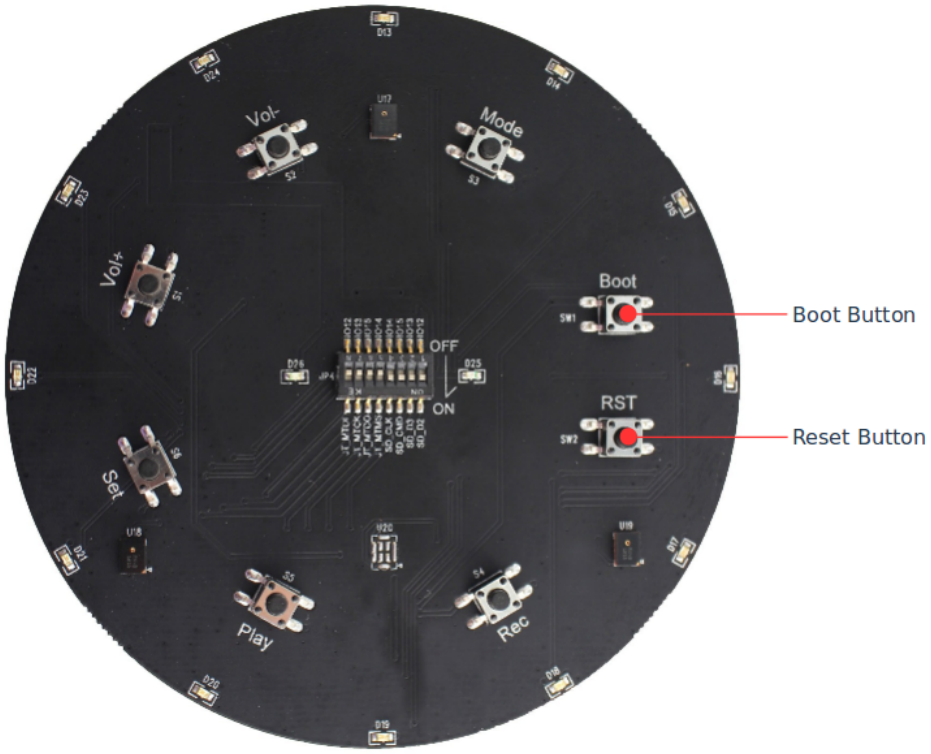


图 27: ESP32-LyraTD-MSC V2.2 上板 (B) 组件图

启动/复位按键 启动：长按 **Boot** 键，然后点按 **Reset** 键进入烧写模式，此时可通过串行端口上传固件。

复位：仅按下 **Reset** 键只能重置系统。

应用程序开发

ESP32-LyraTD-MSC 上电之前，请首先确认开发板完好无损，且上板 (B) 和下板 (A) 紧紧固定在一起。

初始设置

设置开发板，以运行首个示例应用程序：

1. 将扬声器连接至 **两个扬声器输出**，或将耳机连接至 **耳机输出**。
2. 插入 Micro-USB 数据线，连接 PC 与 ESP32-LyraTD-MSC 开发板的 **两个 USB 端口**。
3. 此时，绿色 **待机指示灯**应亮起。若电池未连接，红色 **充电指示灯**每隔几秒闪烁一次。
4. 向右拨动 **电源开关**。
5. 此时，红色 **电源指示灯**应亮起。

如果指示灯如上述显示，则初始设置已经完成，开发板可用于下载应用程序。现在，请按下文介绍运行并配置 PC 上的开发工具。

正式开始开发

若已完成初始设置和检查工作，请准备开发工具，请前往*Installation Step by Step* 查看以下步骤：

- **Set up ESP-IDF** 提供一套 ESP32 和 ESP32-S2 芯片的 C 语言 PC 开发编译环境；
- **Get ESP-ADF** 获取开发音频应用程序的 API；
- **Setup Path to ESP-ADF** 使开发框架获取到音频应用 API；
- **Start a Project** 为开发板提供音频应用程序示例；
- **Connect Your Device** 准备加载应用程序；
- **Build the Project** 运行应用程序，播放音乐。

修订历史

- 板上模组从 ESP32-WROVER-B 更新为 ESP32-WROVER-E。

其他 LyraT 系列开发板

- *ESP32-LyraT V4.3 入门指南*
- *ESP32-LyraT-Mini V1.2 入门指南*

相关文档

- ESP32-LyraTD-MSC V2.2 Schematic Lower Board (A) (PDF)
- ESP32-LyraTD-MSC V2.2 Schematic Upper Board (B) (PDF)
- ESP32 技术规格书 (PDF)
- ESP32-WROVER-E 技术规格书 (PDF)

3.4.8 ESP32-Korvo-DU1906

[English]

本文档介绍了如何使用 ESP32-Korvo-DU1906 开发板。

- 入门指南：简要介绍了 ESP32-Korvo-DU1906 和硬件、软件设置指南。
- 开始开发应用：详细介绍了 ESP32-Korvo-DU1906 的开发过程。
- 相关文档：列出了相关文档的链接。

入门指南

ESP32-Korvo-DU1906 的核心组件包括一个 ESP32-DU1906 蓝牙/Wi-Fi 音频模组，可降低噪声、消除回声，并实现波束形成与检测。ESP32-Korvo-DU1906 集成了电源管理、蓝牙/Wi-Fi 音频模组、编解码器、功率放大器等，包括以下功能：

- ADC
- 麦克风阵列
- SD 卡
- 功能按键
- 指示灯
- 电池恒流恒压线性电源管理芯片
- USB 转 UART
- LCD 接口

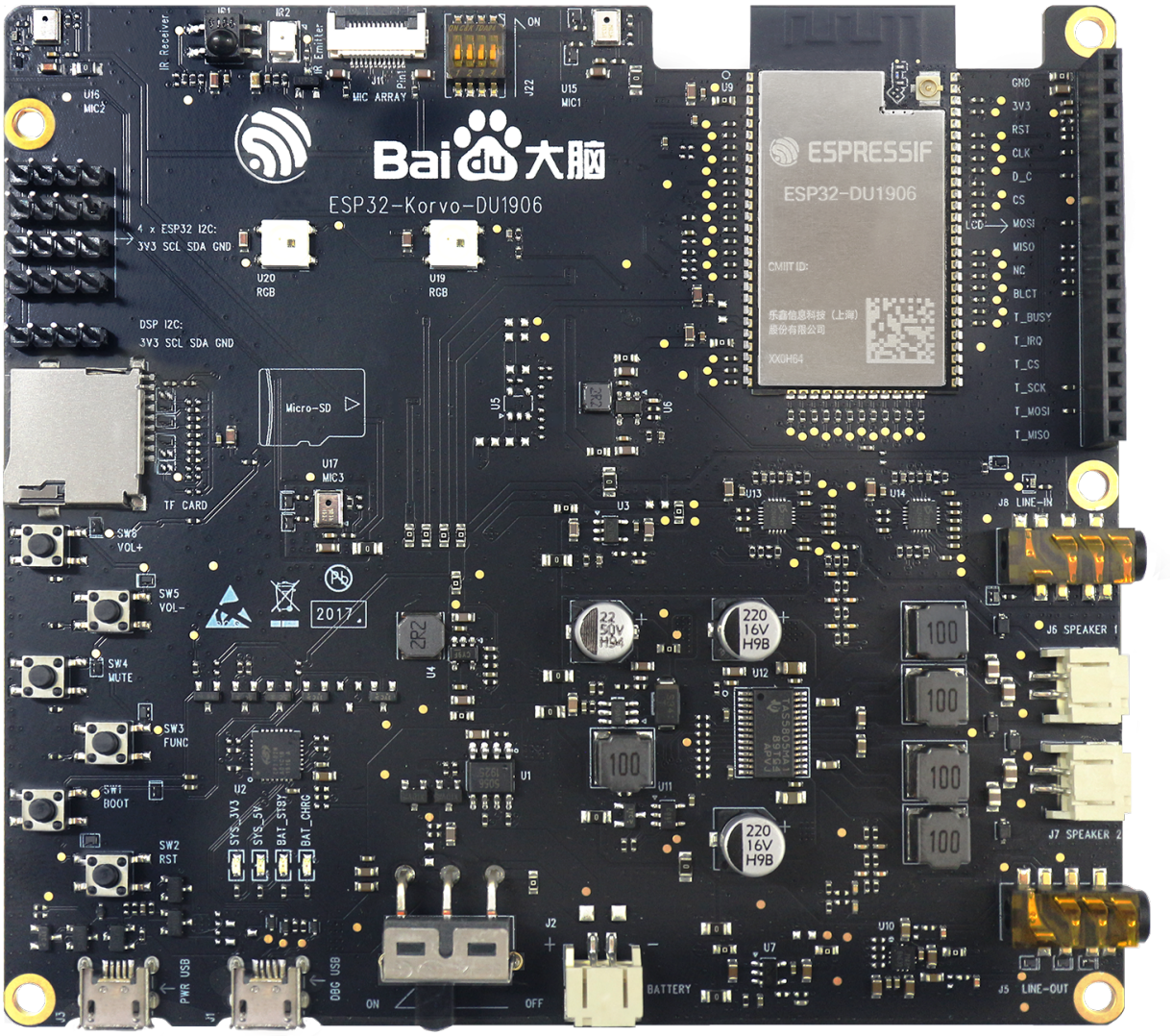


图 28: ESP32-Korvo-DU1906 实物图（点击图片放大）

硬件准备

- 1 x 装有 Windows、Mac OS 或 Linux 系统的 PC（推荐使用 Linux 操作系统）
- 1 x ESP32-Korvo-DU1906
- 2 x Micro USB 数据线
- 2 x 麦克风 (4 Ohm, 2.5 W)

概述

该开发板最大的不同在于添加了语音芯片，核心处理模组 ESP32-DU1906 是一款集 Wi-Fi+Bluetooth+Bluetooth Low Energy 以及音频语音处理的 AI 模组，功能强大、用途广泛，提供业内先进的端到端语音解决方案，以及高效率的一体化 AI 服务能力，同时提供端云一体的设备级 AIOT 平台，大大降低物联网 AI 接入门槛。

DU1906 是百度推出的语音处理旗舰芯片，算法集成度高，一颗芯片同时解决了远场阵列信号实时处理和极低误报率的高精度语音唤醒的产业需求。

ESP32-Korvo-DU1906 的主要组件和连接方式如下图所示。

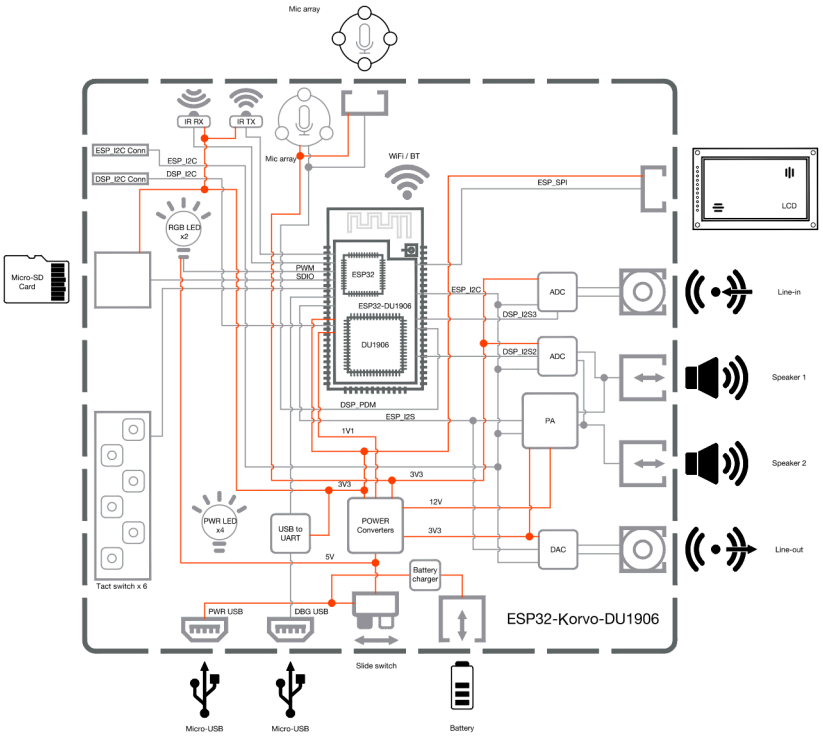


图 29: ESP32-Korvo-DU1906 框图（点击图片放大）

组件介绍

本节介绍了目前所需 ESP32-Korvo-DU1906 的主要组件、接口和控件等。更多详细信息，请参考相关文档中提供的原理图。

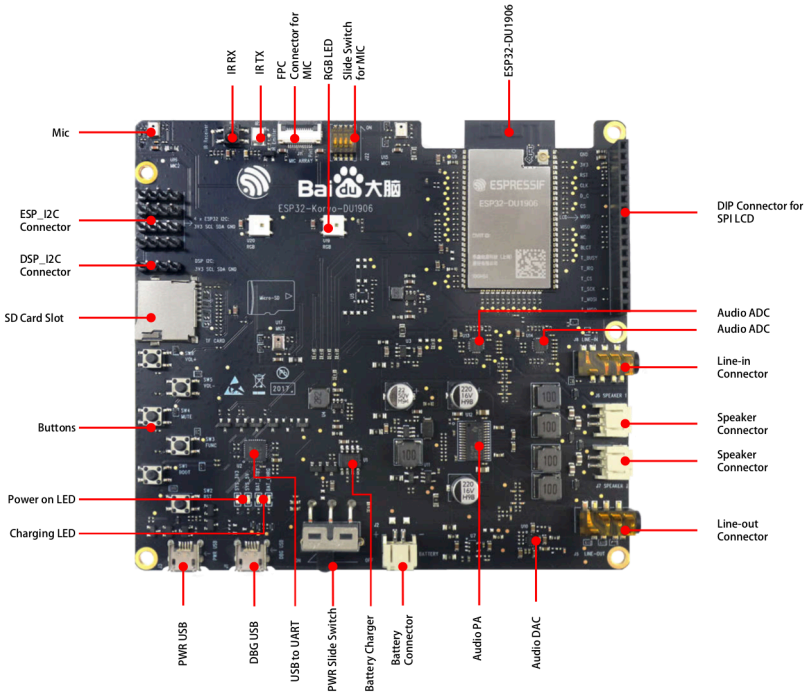


图 30: ESP32-Korvo-DU1906 组件（点击图片放大）

Key Componenet	Description
ESP32-DU1906	ESP32-DU1906 模组是一款功能强大的通用型 Wi-Fi/蓝牙音频通信模组，适用范围广泛，主要面向低功耗传感器网络和语音编码/解码、音乐流及智能语音助手客户端等一系列要求较高的应用场景。
SPI LCD 连接器	ESP32-Korvo-DU1906 上有一个 2.54 mm 间距的连接器用来连接 SPI LCD。
音频模数转换器 (Audio ADC)	ESP32-Korvo-DU1906 上有两颗 ES7243 高性能模数转换器。一颗用来采集数字音频功率放大器（Audio PA）的输出，一个用来采集（Line-in 的输入）。两个模数转换器都可以用作声学回声消除（AEC）。
耳机插孔 (Line-in/out connector)	两个耳机插孔分别用来连接音频数模转换器的 Line-out 输出和音频模数转换器的 Line-in 的输入。当音频数模转换器的 Line-out 耳机插孔插入设备后，ESP32-Korvo-DU1906 上的数字音频功率放大器 (Audio PA) 会被关掉。
扬声器输出端口 (Speaker Connector)	可通过数字音频功率放大器（Audio PA）支持两个外部扬声器实现立体声功能。
音频数模转换器 (Audio DAC)	ES7148 立体声数模转换器可以将音频数字信号转换为模拟音频输出。
数字音频功率放大器 (Audio PA)	TAS5805M 是一款具有低功率耗散和丰富声音的高效立体声闭环 D 类放大器，可以将音频数字信号转换为高功率模拟音频输出发送至外部扬声器进行播放。当音频数模转换器的 Line-out 耳机插孔插入设备后，ESP32-Korvo-DU1906 上的数字音频功率放大器（Audio PA）会被关掉。
电池连接端口 (Battery Connector)	可以连接电池。
电池充电管理芯片 (Battery Charger)	恒流/恒压线性电源管理芯片 AP5056 可以用于单节锂离子电池的充电管理。
电源开关 (PWR Slide Switch)	开发板电源开关，turn on 为开启供电，turn off 为关闭供电。
USB-UART 转换器 (USB to UART)	CP2102N 支持 USB 到 UART 的转换，方便软件下载与调试。
Micro-USB 调试接口 (DBG USB)	通用 USB 通信端口，用于 PC 端与 ESP32-DU1906 模组的通信。
电源输入 (PWR USB)	为整个系统提供电源。建议使用至少 5 V / 2 A 电源适配器供电，保证供电稳定。
充电指示 LED	指示电池的状态。电池连接后，BAT_CHRG 指示灯亮红灯（表示正在充电），BAT_STBY 指示灯亮绿灯（表示电量已充满）。若未连接电池，默认 BAT_CHRG（红色），BAT_STBY（绿色）。
电源指示 LED	指示供电状态。上电后，两个指示灯 (SYS_3V3, SYS_5) 都亮红灯。
按键 (Buttons)	ESP32-Korvo-DU1906 上有 4 个功能按键、1 个重启按键和 1 个 Boot 选择按键。
TF 卡连接器 (SD Card Slot)	用于连接标准 TF 卡。
调试接口 (ESP_I2C Connector/DSP_I2C Connnector)	开发板预留了两组 I2C 调试接口，分别为 ESP_I2C Connector and DSP_I2C Connnector，以供用户调试代码。
麦克阵列 (Mic)	ESP32-Korvo-DU1906 上有三个板载数字麦克风。三个麦克风的拾音孔呈正三角分布并且相互之间的距离 60 mm。麦克风阵列配合 DSP 可以实现降低噪声、回声消除，并且实现波束形成与检测功能。
红外发射/接收器 (IR TX/RX)	ESP32-Korvo-DU1906 上有红外发射和接收器各一个，可以配合 ESP32 的红外遥控器使用。
FPC 连接器 (FPC Connector	ESP32-Korvo-DU1906 上有两个 FPC 连接器分别用来连接 SPI LCD 显示

开始开发应用

通电前，请确保 ESP32-Korvo-DU1906 开发板完好无损。

初始设置

设置开发板，运行首个示例应用程序：

1. 将 4 欧姆扬声器接至两个 **扬声器输出端口**，或将耳机接至 **Line-out 输出口**。
2. 使用两根 Micro-USB 数据线连接 PC 与 ESP32-Korvo-DU1906 的两个 **USB 接口**。
3. 如果电池已连接，则 **充电指示 LED** 将亮红灯。
4. 将 **电源开关**拨至左侧。
5. **电源指示 LED** 应亮红灯。

如果指示灯如上述显示，则该开发板基本完好，可以开始上传应用。现在，请按照下文介绍，运行并配置 PC 上的开发工具。

开发应用

如果已检查确认完成初始设置，请准备开发工具。前往*Development Boards* 查看以下步骤：

- **Set up ESP-IDF** 提供了一套 ESP32（以及 ESP32-S2）系列芯片的 C 语言开发框架；
- **Get ESP-ADF** 安装音频应用程序的 API；
- **Setup Path to ESP-ADF** 使开发框架获取到音频应用的 API；
- **Start a Project** 提供 ESP32-Korvo-DU1906 开发板的音频应用程序示例；
- **Connect Your Device** 准备加载应用程序；
- **Build the Project** 最后运行应用程序并播放音乐。

相关开发板

- *ESP32-LyraT V4.3 入门指南*
- *ESP32-LyraT-Mini V1.2 入门指南*
- *ESP32-LyraTD-MSC V2.2 入门指南*

内含组件和包装

零售订单

如购买样品，每个 ESP32-Korvo-DU1906 底板将以塑料包装盒或零售商选择的其他方式包装。

零售订单请前往 https://www.espressif.com/zh-hans/products/devkits/esp32-korvo-du1906。

相关文档

- ESP32-Korvo-DU1906 原理图 (PDF)
- ESP32 技术规格书 (PDF)
- 百度 IOT 技能后台
- ESP32-DU1906 & ESP32-DU1906-U 技术规格书 (PDF)

3.4.9 ESP32-S3-Korvo-2 V3.0

[English]

本指南将帮助您快速上手 ESP32-S3-Korvo-2 V3.0，并提供该款开发板的详细信息。

ESP32-S3-Korvo-2 是一款基于 ESP32-S3 芯片的多媒体开发板，搭载双麦克风阵列，支持语音识别和近/远场语音唤醒。同时它还搭载 LCD、摄像头、microSD 卡等外设，可支持基于 JPEG 的视频流处理，满足用户对低成本、低功耗、联网的音视频产品开发需求。

ESP32-S3-Korvo-2 开发板主要由以下几个部分组成：

- 主板：ESP32-S3-Korvo-2
- LCD 扩展板：*ESP32-S3-Korvo-2-LCD*
- 摄像头

本文档主要介绍 ESP32-S3-Korvo-2 主板，有关其他部分的信息请点击相应的链接。

本指南包括如下内容：

- 入门指南：简要介绍了开发板和硬件、软件设置指南。
- 硬件参考：详细介绍了开发板的硬件。
- 硬件版本：介绍硬件历史版本和已知问题，并提供链接至历史版本开发板的入门指南（如有）。
- 相关文档：列出了相关文档的链接。

图 31: ESP32-S3-Korvo-2 V3.0（板载 ESP32-S3-WROOM-1 模组）

入门指南

本小节将简要介绍 ESP32-S3-Korvo-2 V3.0，说明如何在 ESP32-S3-Korvo-2 V3.0 上烧录固件及相关准备工作。

组件介绍

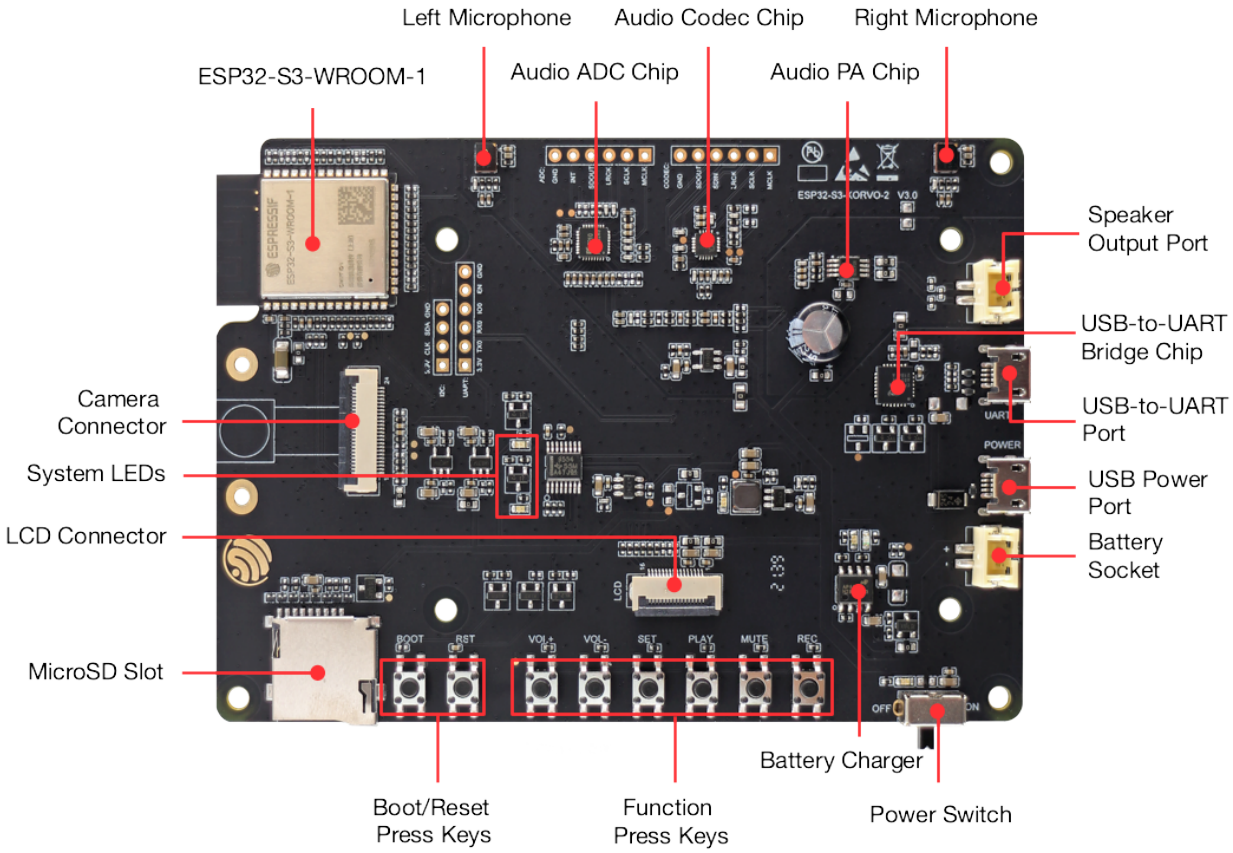


图 32: ESP32-S3-Korvo-2 V3.0（点击放大）

以下按照顺时针的顺序依次介绍开发板上的主要组件。

主要组件	介绍
ESP32-S3-WROOM-1 模组	ESP32-S3-WROOM-1 模组是两款通用型 Wi-Fi + 低功耗蓝牙 MCU 模组，搭载 ESP32-S3 系列芯片。除具有丰富的外设接口外，模组还拥有强大的神经网络运算能力和信号处理能力，适用于 AIoT 领域的多种应用场景，例如唤醒词检测和语音命令识别、人脸检测和识别、智能家居、智能家电、智能控制面板、智能扬声器等。
左侧麦克风 (Left Microphone)	板载麦克风，连接至 ADC。
音频模数转换器 (Audio ADC Chip)	ES7210 是一款用于麦克风阵列应用的高性能、低功耗 4 通道音频模数转换器，非常适合音乐和语音应用。此外，ES7210 也可以用作声学回声消除 (AEC)。
音频编解码芯片 (Audio Codec Chip)	音频编解码器芯片 ES8311 是一种低功耗单声道音频编解码器，包含单通道 ADC、单通道 DAC、低噪声前置放大器、耳机驱动器、数字音效、模拟混音和增益功能。它通过 I2S 和 I2C 总线与 ESP32-S3-WROOM-1 模组连接，以提供独立于音频应用程序的硬件音频处理。
音频功率放大器 (Audio PA Chip)	NS4150 是一款低 EMI、3 W 单声道 D 类音频功率放大器，用于放大来自音频编解码芯片的音频信号，以驱动扬声器。
右侧麦克风 (Right Microphone)	板载麦克风，连接至 ADC。
扬声器输出端口 (Speaker Output Port)	可通过音频功率放大器的支持，实现外部扬声器播放功能。
USB-to-UART 桥接器 (USB-to-UART Bridge Chip)	单芯片 USB-UART 桥接器 CP2102N 为软件下载和调试提供高达 3 Mbps 的传输速率。
USB-to-UART 端口 (USB-to-UART Port)	用于 PC 端与 ESP32-S3-WROOM-1 模组的通信。
USB 电源端口 (USB Power Port)	为整个系统提供电源。建议使用至少 5V/2A 电源适配器供电，保证供电稳定。
电池接口 (Battery Socket)	用于连接单节锂离子电池的两针接口。
电源开关 (Power Switch)	电源拨动开/关：向下拨动开启开发板电源，向上拨动关闭开发板电源。
电池充电管理芯片 (Battery Charger)	恒流/恒压线性电源管理芯片 AP5056 可以用于单节锂离子电池的充电管理，为通过 Micro USB 端口连接到电池接口的电池充电。
功能按键 (Function Press Keys)	六个按键，分别为 REC、MUTE、PLAY、SET、VOL- 和 VOL+，与 ESP32-S3-WROOM-1 模组连接，借助该 UI 和专用的 API 可以开发和测试音频应用程序。
Boot/Reset 按键 (Boot/Reset Press Keys)	Boot：长按 Boot 键时，再按 Reset 键可启动固件上传模式，然后便可通过串口上传固件。 Reset：单独按下此按键会重置系统。
microSD 插槽 (MicroSD Slot)	本开发板支持一线模式的 microSD 卡，可以存储或播放 microSD 卡中的音频文件。
LCD 连接器 (LCD Connector)	一款 0.5 mm 间距的 FPC 连接器，用以连接 LCD 扩展板。
系统 LED (System LEDs)	两个通用 LED（绿色和红色），由 ESP32-S3-WROOM-1 模组控制，可借助专用的 API 为音频应用程序做状态行为指示。
摄像头连接器 (Camera Connector)	通过连接器外接摄像头模组至开发板，实现图像传输。

开始开发应用

通电前，请确保开发板完好无损。

必备硬件

- 1 x ESP32-S3-Korvo-2 V3.0
- 1 x 扬声器
- 2 x USB 2.0 数据线（标准 A 型转 Micro-B 型）
- 1 x 电脑（Windows、Linux 或 macOS）

注解： 请确保使用适当的 USB 数据线。部分数据线仅可用于充电，无法用于数据传输和编程。

可选硬件

- 1 x microSD 卡
- 1 x 锂离子电池

注解： 请务必使用内置保护电路的锂离子电池。

硬件设置

1. 连接扬声器至 **扬声器输出**端口。
2. 插入 USB 数据线，分别连接 PC 与开发板的两个 USB 端口。
3. 此时，绿色待机指示灯应亮起。若电池未连接，红色充电指示灯每隔几秒闪烁一次。
4. 打开 **电源开关**。
5. 此时，红色电源指示灯应亮起。

软件设置

请前往*Get Started*，在*Installation Step by Step* 小节查看如何快速设置开发环境，并将 应用程序示例 烧录至您的开发板。

内含组件和包装

可分开购买主板或主板配件，其中配件包含：

- LCD 扩展板：ESP32-S3-Korvo-2-LCD
- 摄像头
- 连接器
  - 20 针 FPC 线
- 紧固件：
  - 安装螺栓 (x8)
  - 螺丝 (x4)

零售订单

如购买样品，每个开发板将以防静电袋或零售商选择的其他方式包装。

零售订单请前往 https://www.espressif.com/zh-hans/company/contact/buy-a-sample。

批量订单

如批量购买，开发板将以大纸板箱包装。

批量订单请前往 https://www.espressif.com/zh-hans/contact-us/sales-questions。

硬件参考

功能框图

ESP32-S3-Korvo-2 V3.0 的主要组件和连接方式如下图所示。

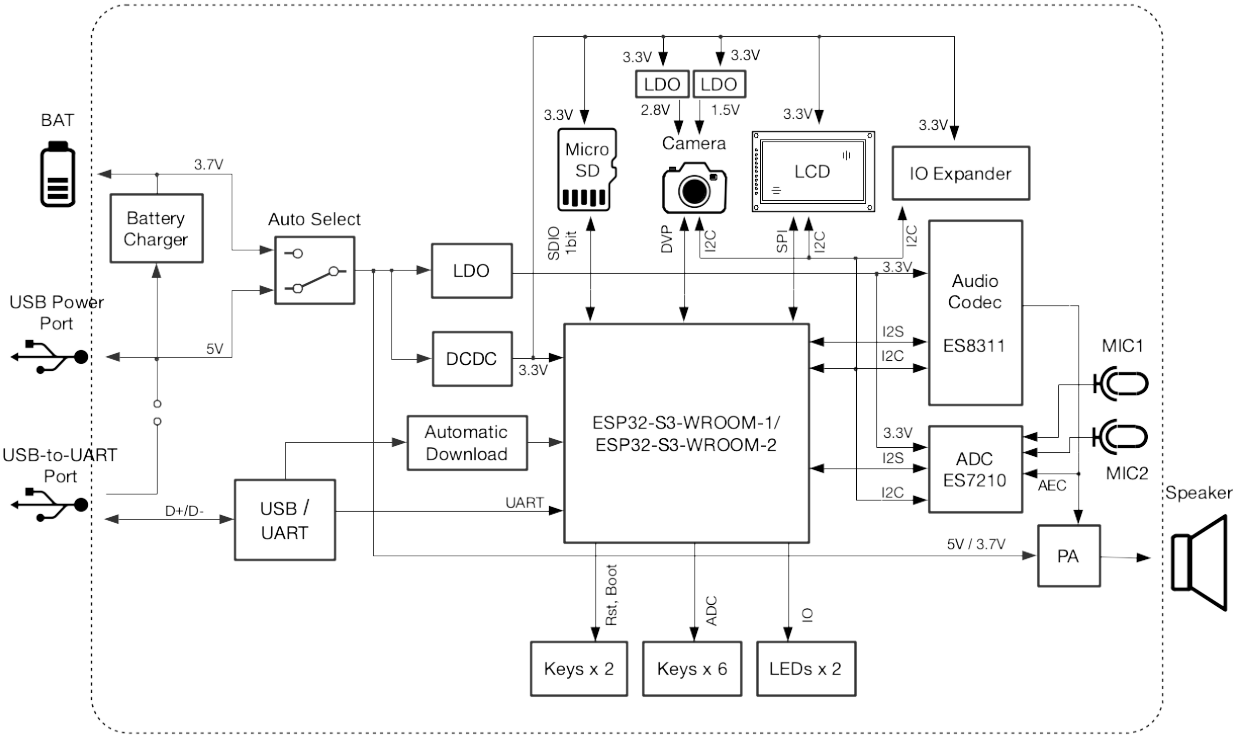


图 33: ESP32-S3-Korvo-2 V3.0 电气功能框图

供电说明

USB 与电池供电

主电源为 5 V，由 USB 提供。辅助电源为 3.7 V，由电池提供，为可选项。USB 供电使用专用的数据线，与用于上传应用程序的 USB 数据线分开。为了近一步减少来自 USB 的噪音，可使用电池代替 USB。

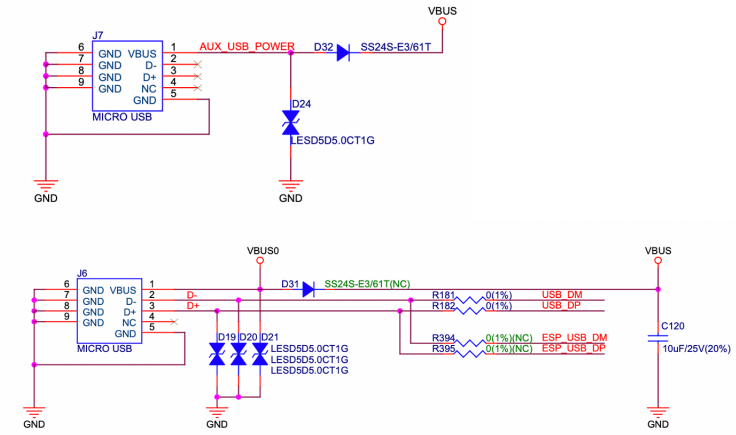


图 34: ESP32-S3-Korvo-2 V3.0 - USB 电源供电

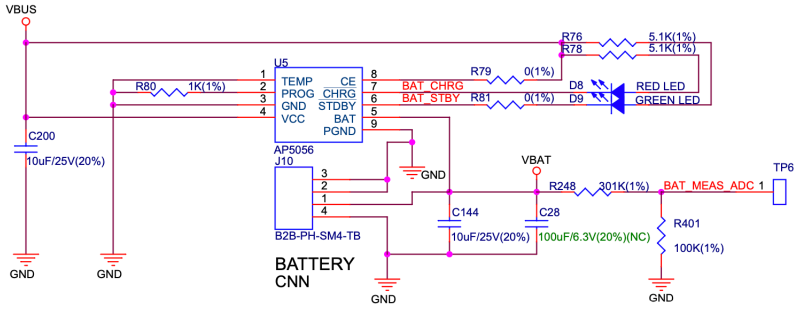


图 35: ESP32-S3-Korvo-2 V3.0 - 电池供电

如下图所示，当 USB 供电和电池供电同时存在时，VBUS 为高电平，Q14 处于截止状态，VBAT 自动与系统电源切断。此时，USB 为系统供电。

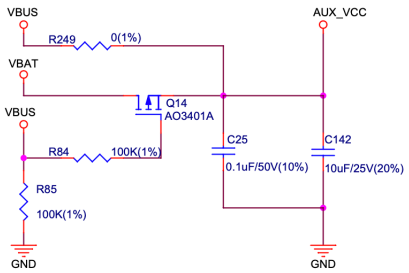


图 36: ESP32-S3-Korvo-2 V3.0 - 供电选项

音频和数字独立供电

ESP32-S3-Korvo-2 V3.0 可为音频组件和 ESP 模组提供相互独立的电源，可降低数字组件给音频信号带来的噪声并提高组件的整体性能。

Power for Digital:

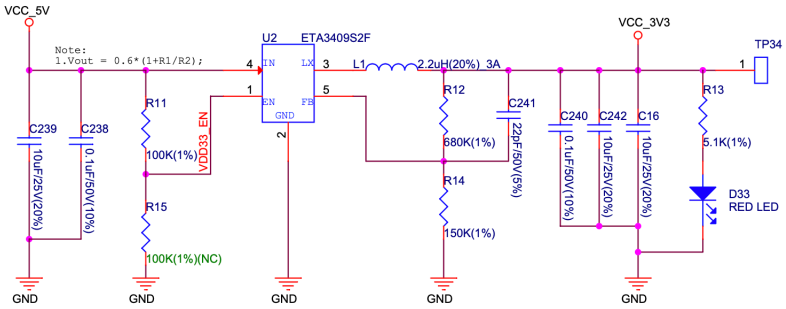


图 37: ESP32-S3-Korvo-2 V3.0 - 数字供电

GPIO 分配列表

下表为 ESP32-S3-WROOM-1 模组管脚的 GPIO 分配列表，用于控制开发板的特定组件或功能。

表 2: ESP32-S3-WROOM-1 GPIO 分配

管脚[1]	管脚名称	ES8311	ES7210	摄像头	LCD	按键	microSD 卡	IO 扩展	其他
3	EN					EN_KEY			
4	IO4						DATA0		

下页继续

表 2 – 续上页

管脚[1]	管脚名称	ES8311	ES7210	摄像头	LCD	按键	microSD 卡	IO 扩展	其他
5	IO5					REC, MUTE, PLAY, SET, VOL-, VOL+			
6	IO6								BAT_MEAS_ADC
7	IO7						CMD		
8	IO15						CLK		
9	IO16	I2S0_MCLK	MCLK						
10	IO17	I2C_SDA	I2C_SDA	SIOD	TP_I2C_SDA			I2C_SDA	
11	IO18	I2C_CLK	I2C_CLK	SIOC	TP_I2C_CLK			I2C_CLK	
12	IO8	I2S0_DSDIN							
13	IO19								ESP_USB_DM (Reserve)
14	IO20								ESP_USB_DP (Reserve)
15	IO3			D5					
16	IO46								NC
17	IO9	I2S0_SCLK	SCLK						
18	IO10		SDOUT						
19	IO11			PCLK					
20	IO12			D6					
21	IO13			D2					
22	IO14			D4					
23	IO21			VSYNC					
24	IO47			D3					
25	IO48								PA_CTRL
26	IO45	I2S0_LRCK	LRCK						
27	IO0				LCD_SPI_SDO	BOOT_KEY			
28	IO35								NC
29	IO36								NC
30	IO37								NC
31	IO38			HREF					
32	IO39			D9					

下页继续

表 2 – 续上页

管脚[1]	管脚名称	ES8311	ES7210	摄像头	LCD	按键	microSD 卡	IO 扩展	其他
33	IO40			XCLK					
34	IO41			D8					
35	IO42			D7					
36	RXD0								ESP0_UART0_RX
37	TXD0								ESP0_UART0_TX
38	IO2				LCD_SPI_DC				
39	IO1				LCD_SPI_CLK				
41	EPAD								

分配给 IO 扩展器的 GPIO 被近一步分配为多个 GPIO。

表 3: IO 扩展器 GPIO 分配

IO 扩展器管脚	管脚名称	LCD	其他
4	P0		PA_CTRL
5	P1	LCD_CTRL	
6	P2	LCD_RST	
7	P3	LCD_CS	
9	P4	TP_INT	
10	P5		PERI_PWR_ON
11	P6		LED1
12	P7		LED2

连接器

[1] 管脚 - ESP32-S3-WROOM-1 模组管脚号，不含 GND 和供电管脚。

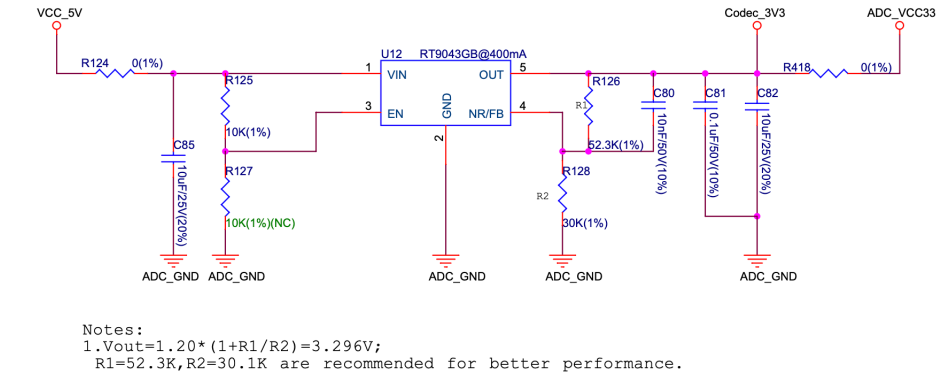


图 38: ESP32-S3-Korvo-2 V3.0 - 音频供电

摄像头连接器

No.	摄像头信号	ESP32-S3 管脚
1	SIOD	GPIO17
2	SIOC	GPIO18
3	D5	GPIO3
4	PCLK	GPIO11
5	D6	GPIO12
6	D2	GPIO13
7	D4	GPIO14
8	VSYNC	GPIO21
9	D3	GPIO47
10	HREF	GPIO38
11	D9	GPIO39
12	XCLK	GPIO40
13	D8	GPIO41
14	D7	GPIO42

LCD 连接器

No.	LCD 信号	ESP32-S3 管脚
1	TP_I2C_SDA	GPIO17
2	TP_I2C_CLK	GPIO18
3	LCD_SPI_SDA	GPIO0
4	LCD_SPI_DC	GPIO2
5	LCD_SPI_CLK	GPIO1

No.	LCD 信号	扩展器管脚
1	ESP_LCD_CTRL	P1
2	ESP_LCD_RST	P2
3	ESP_LCD_CS	P3
4	ESP_TP_INT	P4

AEC 电路

AEC 电路为 AEC 算法提供参考信号。

ESP32-S3-Korvo-2 回声参考信号源有两路兼容设计，一路是 Codec (ES8311) DAC 输出 (DAC_AOUTLN/DAC_AOUTLP)，一路是 PA (NS4150) 输出 (PA_OUTL+/PA_OUTL-)。默认推荐将 Codec (ES8311) DAC 输出 (DAC_AOUTLN/DAC_AOUTLP) 作为回声参考信号，同时将下图中电阻 R132、R140 NC。

回声参考信号通过 ADC (ES7210) 的 ADC_MIC3P/ADC_MIC3N 采集后送回给 ESP32-S3 用于 AEC 算法。

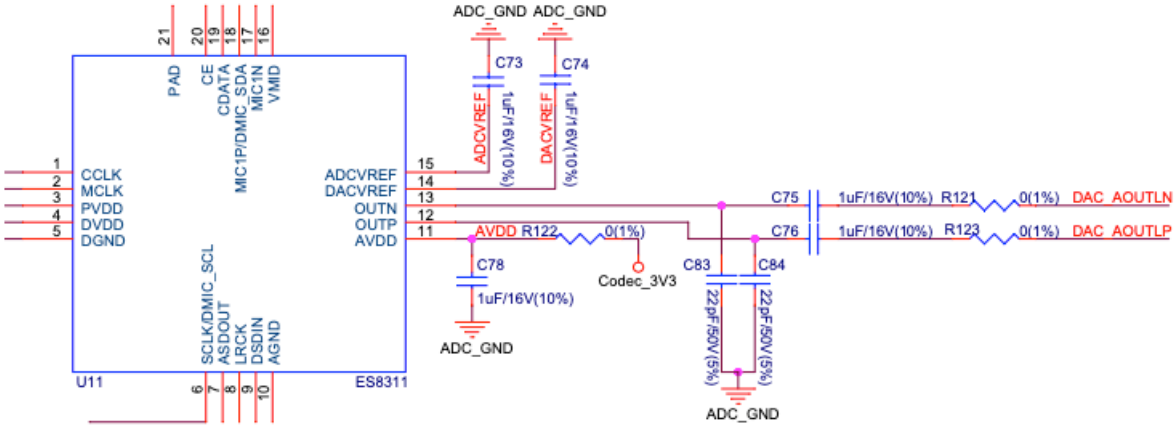


图 39: ESP32-S3-Korvo-2 V3.0 - AEC Codec DAC 输出（点击放大）

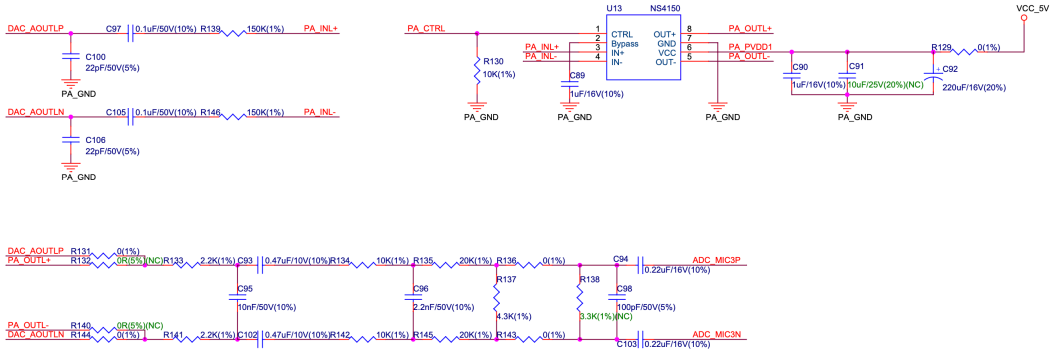


图 40: ESP32-S3-Korvo-2 V3.0 - AEC PA 输出（点击放大）

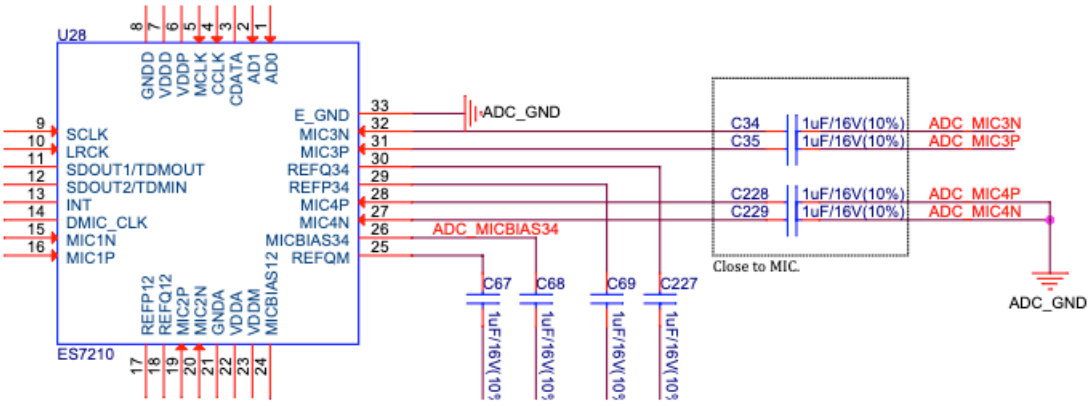


图 41: ESP32-S3-Korvo-2 V3.0 - 参考信号采集（点击放大）

硬件设置选项

自动下载

可以通过两种方式使 ESP 开发板进入下载模式：

- 手动按下 Boot 和 RST 键，然后先松开 RST，再松开 Boot 键。
- 由软件自动执行下载。软件利用串口的 DTR 和 RTS 信号来控制 ESP 开发板的 EN、IO0 管脚的状态。详情请参见 ESP32-S3-Korvo-2 V3.0 原理图 (PDF)。

ESP 管脚测试点分配

本节介绍了 ESP32-S3-Korvo-2 V3.0 板上可分配的测试点。

测试点是裸通孔，具有标准的 2.54 毫米/0.1 英寸间距。您可能需要接入排针或排针插孔，从而连接外部硬件。

编解码器测试点/J15

No.	编解码器管脚	ESP32-S3 管脚
1	MCLK	GPIO16
2	SCLK	GPIO9
3	LRCK	GPIO45
4	DSDIN	GPIO8
5	ASDOUT	–
6	GND	–

ADC 测试点/J16

No.	ADC 管脚	ESP32-S3 管脚
1	MCLK	GPIO16
2	SCLK	GPIO9
3	LRCK	GPIO45
4	SDOUT	GPIO10
5	INT	–
6	GND	–

UART 测试点/J17

No.	UART 管脚
1	3.3V
2	TXD
3	RXD
4	IO0
5	EN
6	GND

I2C 测试点/J18

No.	I2C 管脚	ESP32-S3 管脚
1	3.3V	–
2	CLK	GPIO18
3	SDA	GPIO17
4	GND	–

硬件版本

无历史版本。

相关文档

- ESP32-S3 技术规格书 (PDF)
- ESP32-S3-WROOM-1/1U 技术规格书 (PDF)
- ESP32-S3-Korvo-2 V3.0 原理图 (PDF)
- ESP32-S3-Korvo-2 V3.0 PCB 布局图 (PDF)

有关本开发板的更多设计文档，请联系我们的商务部门 sales@espressif.com。

3.4.10 ESP32-S3-Korvo-2-LCD V1.0

[English]

本指南将帮助您快速上手 ESP32-S3-Korvo-2-LCD 扩展板，并提供该款扩展板的详细信息。

本扩展板需与其他的*ESP32-S3-Korvo-2 V3.0 配件* 一同购买，无法单独购买。

ESP32-S3-Korvo-2-LCD 增配 LCD 图形显示器和电容式触摸屏，扩展了 ESP32-S3-Korvo-2 V3.0（以下简称“主板”）的功能。

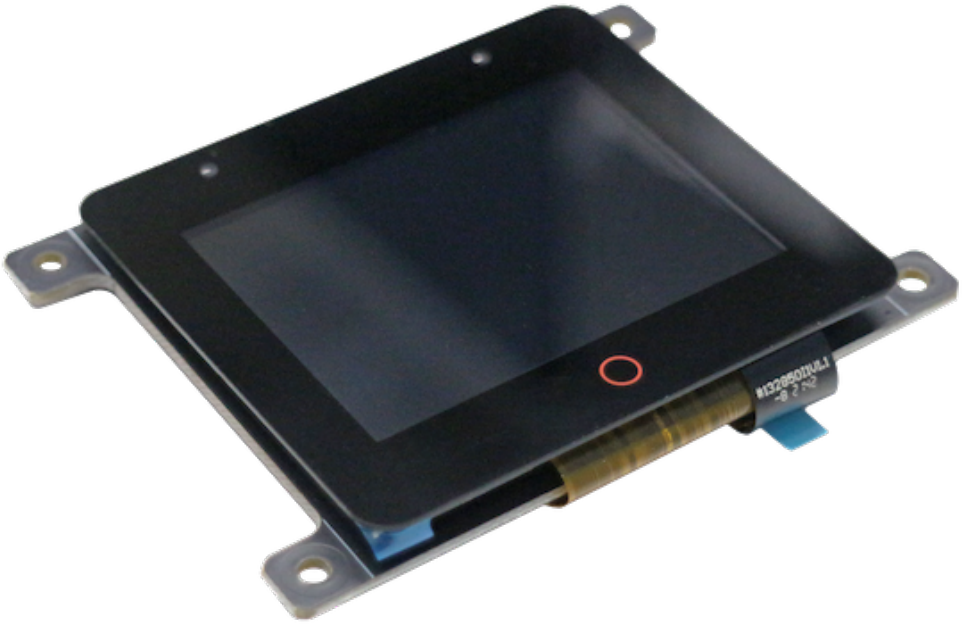


图 42: ESP32-S3-Korvo-2-LCD V1.0

本指南包括如下内容：

- 入门指南：简要介绍了开发板和硬件、软件设置指南。
- 硬件概况：详细介绍了开发板的硬件。
- 相关文档：列出了相关文档的链接。

入门指南

ESP32-S3-Korvo-2-LCD 搭载 320x240 分辨率的 2.4 英寸 LCD 图形显示器和 10 点电容式触摸屏。该显示器通过 SPI 总线连接到 ESP32-S3。

组件介绍

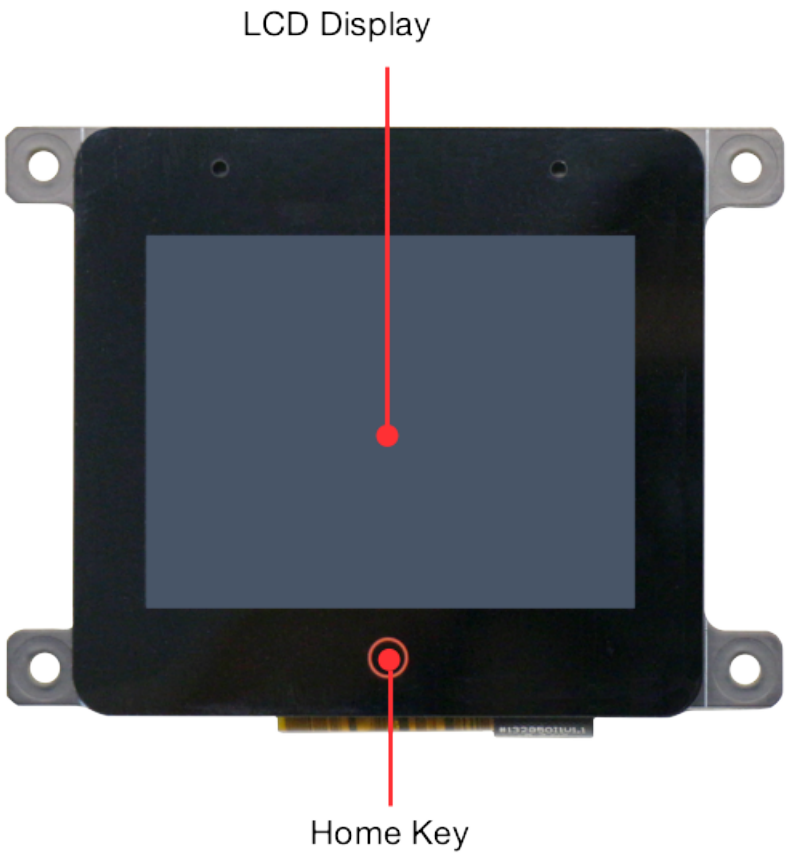


图 43: ESP32-S3-Korvo-2-LCD V1.0 - 正面

以下按照顺时针的顺序依次介绍开发板上的主要组件。**保留**表示该功能可用，但当前版本并未启用该功能。

主要组件	介绍
LCD 显示器 (LCD Display)	2.4 英寸 320x240 SPI LCD 显示模块，显示驱动器/控制器是 Ilitek ILI934。
Home 键 (Home Key)	（保留）返回首页或上一页。
信号连接器 (Signal Connector)	借助 FPC 排线连接扩展板和主板之间的电源线、地线和信号线。
LCD 连接器 (LCD Connector)	将 LCD 显示器连接到扩展板的驱动电路。
TP 连接器 (TP Connector)	将 LCD 显示器连接到扩展板的触摸电路。

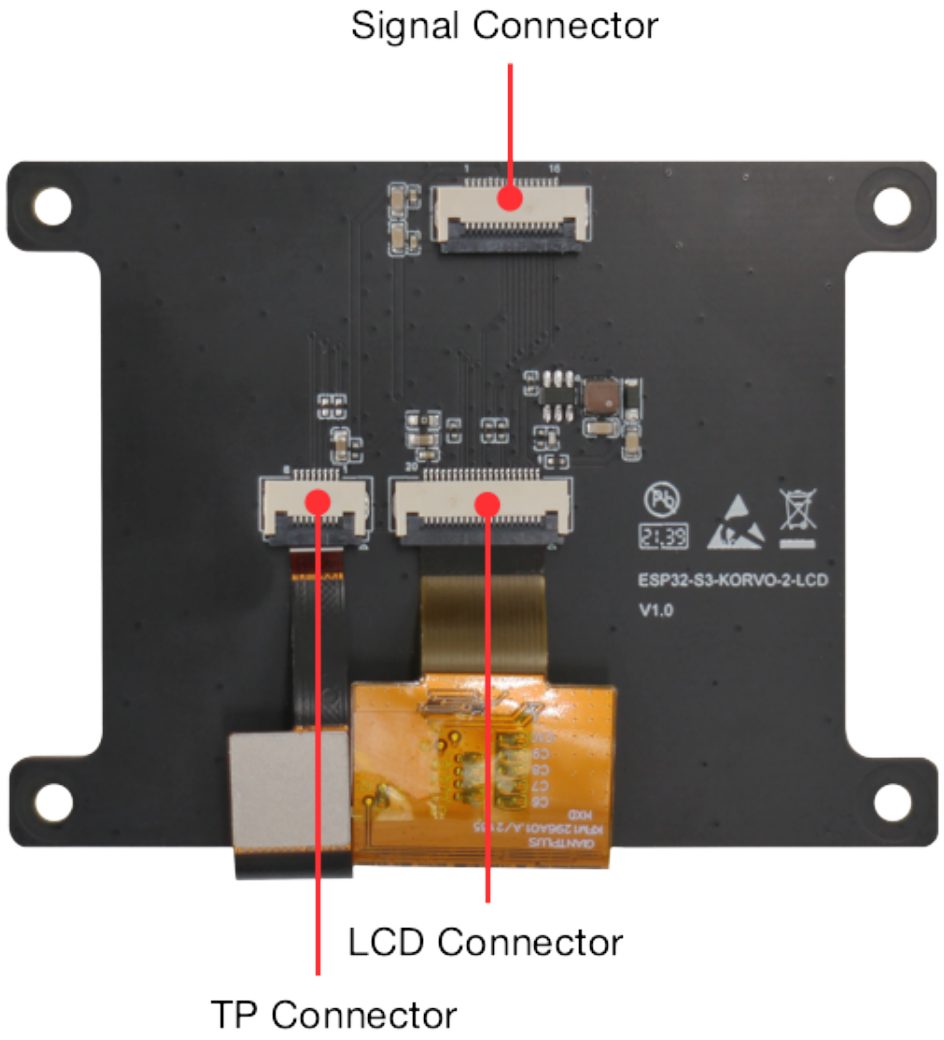


图 44: ESP32-S3-Korvo-2-LCD V1.0 - 背面

开始开发应用

通电前，请确保开发板完好无损。

必备硬件

- 主板：ESP32-S3-Korvo-2 V3.0
- 扩展板：ESP32-S3-Korvo-2-LCD V1.0
- 两根 USB 2.0 数据线（标准 A 型转 Micro-B 型）
- 安装螺栓和螺丝（用于稳定安装）
- FPC 排线（用于连接主板和扩展板）
- 电脑（Windows、Linux 或 macOS）

硬件设置

按照以下步骤将 ESP32-S3-Korvo-2-LCD 安装到 ESP32-S3-Korvo-2 上：

1. 使用 FPC 线连接两块开发板。
2. 安装螺栓和螺丝，保证安装稳定。

软件设置

请参考主板的用户指南的软件设置 章节。

硬件概况

功能框图

ESP32-S3-Korvo-2-LCD 的主要组件和连接方式如下图所示。

硬件版本

无历史版本。

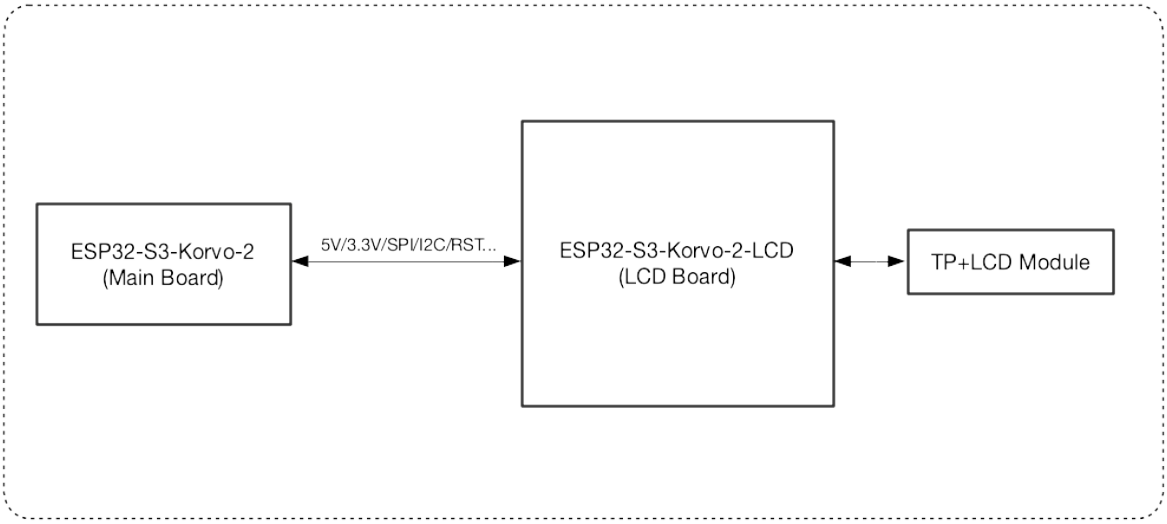


图 45: ESP32-S3-Korvo-2-LCD

相关文档

- *ESP32-S3-Korvo-2 V3.0 用户指南*
- ESP32-S3-Korvo-2-LCD 原理图 (PDF)
- ESP32-S3-Korvo-2-LCD PCB 布局图 (PDF)

有关本开发板的更多设计文档，请联系我们的商务部门 sales@espressif.com。

3.4.11 ESP32-C3-Lyra V2.0

[English]

本指南将帮助您快速上手 ESP32-C3-Lyra V2.0，并提供该款开发板的详细信息。

本指南包括如下内容：

- 开发板概述：简要介绍了开发板的软件和硬件。
- 开始开发应用：介绍了应用程序开发过程中的软硬件设置。
- 硬件参考：详细介绍了开发板的硬件。
- 硬件版本：列出了硬件历史版本和已知问题，并链接至历史版本开发板的入门指南。
- 订购信息：提供了购买开发板的途径。
- 相关文档：列出了相关文档的链接。

开发板概述

ESP32-C3-Lyra 是乐鑫基于 ESP32-C3 推出的音频灯控开发板，该开发板对麦克风、扬声器以及 LED 灯带的控制，可满足客户对超高性价比的音频播报机以及律动灯带的产品开发需求。

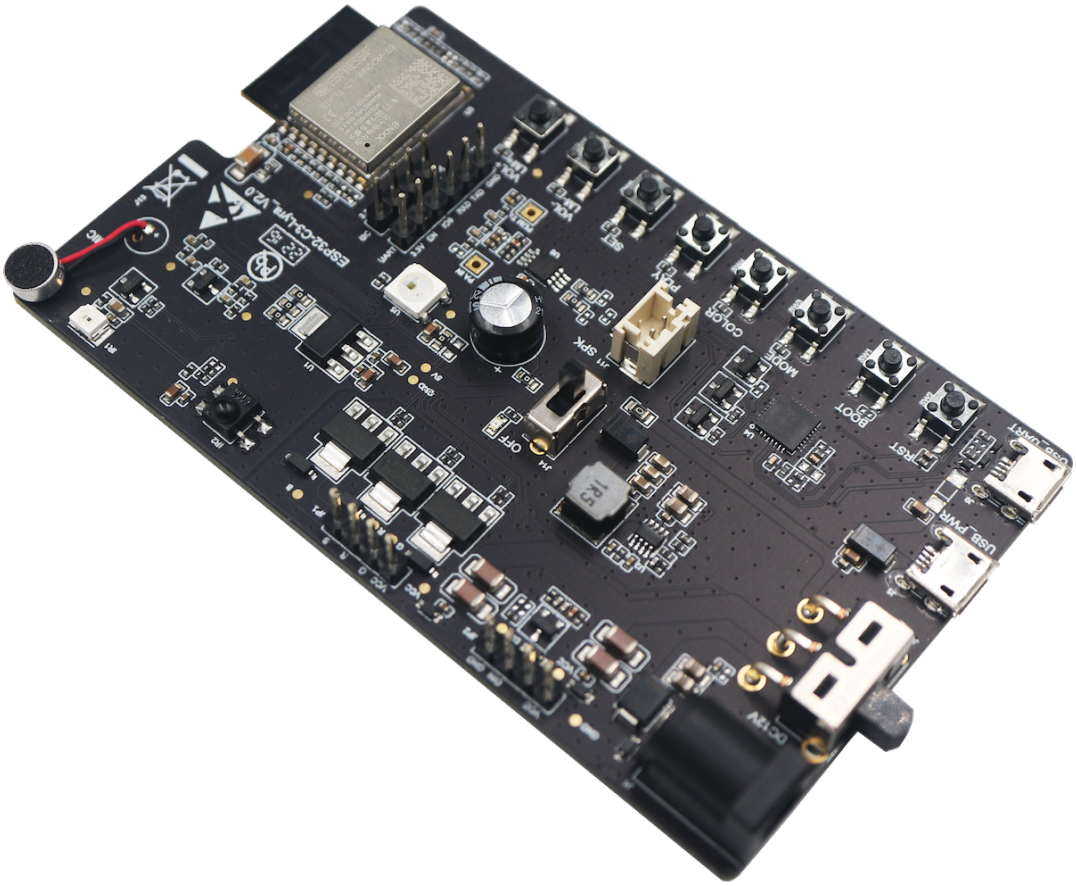


图 46: 搭载 ESP32-C3-WROOM-02 模组的 ESP32-C3-Lyra

特性列表

该开发板包括以下主要特性：

- **嵌入式模组：** 板载 ESP32-C3-WROOM-02 模组，外接 4 MB SPI flash
- **音频：** 内置 ECM 麦克风、扬声器功率放大器、扬声器连接器
- **LED 灯带连接器：** 支持连接两种类型的灯带，一种是可寻址的 LED 灯带，一种是 RGB LED 灯带

- **红外控制：** 支持红外 (IR) 发射和接收
- **按键：** 开机键、复位键、六个功能键 (MODE、COLOR、PLAY/PAUSE、SET、VOL+/LM+、VOL-/LM-)
- **USB：** 1 个 USB 电源端口，1 个 USB 转 UART 端口
- **电源：** 5 V USB 供电或 12 V 直流供电

功能框图

ESP32-C3-Lyra 的主要组件和连接方式如下图所示。

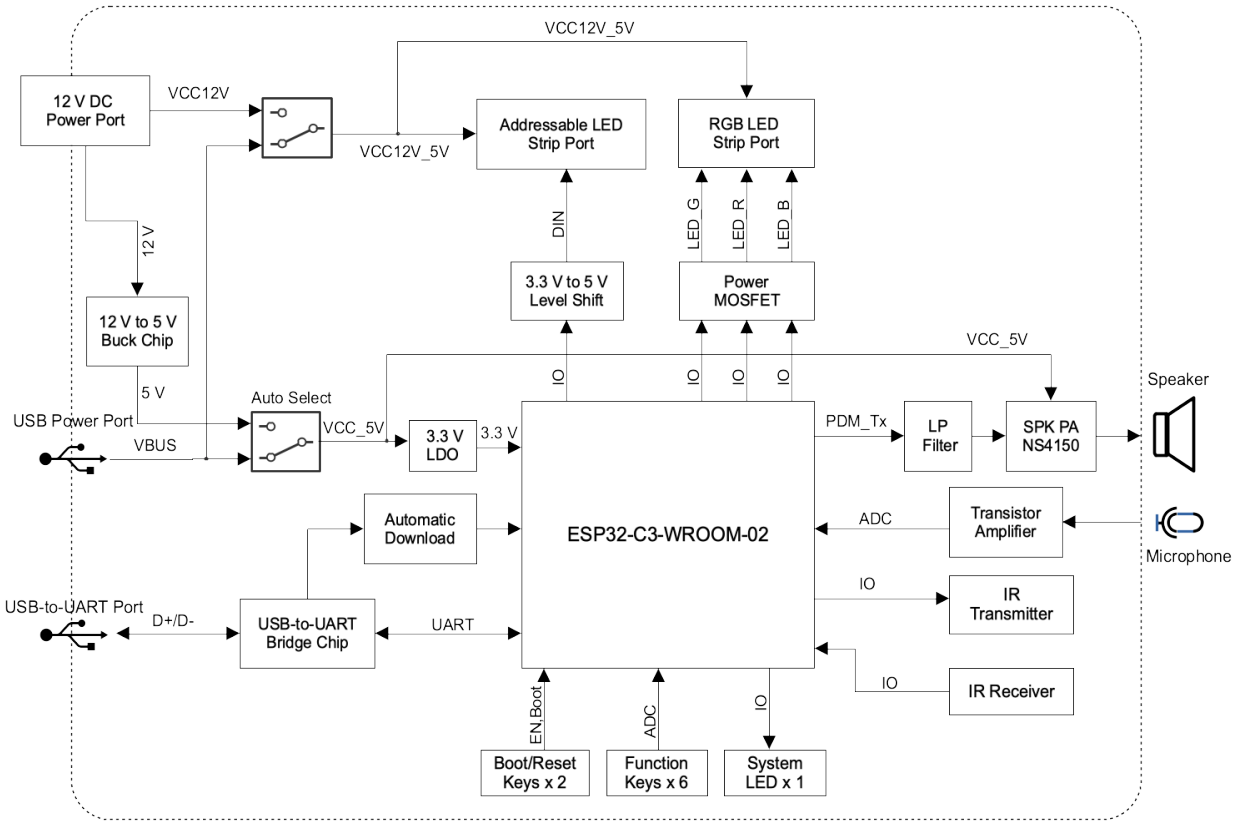


图 47: ESP32-C3-Lyra 功能框图（点击放大）

组件介绍

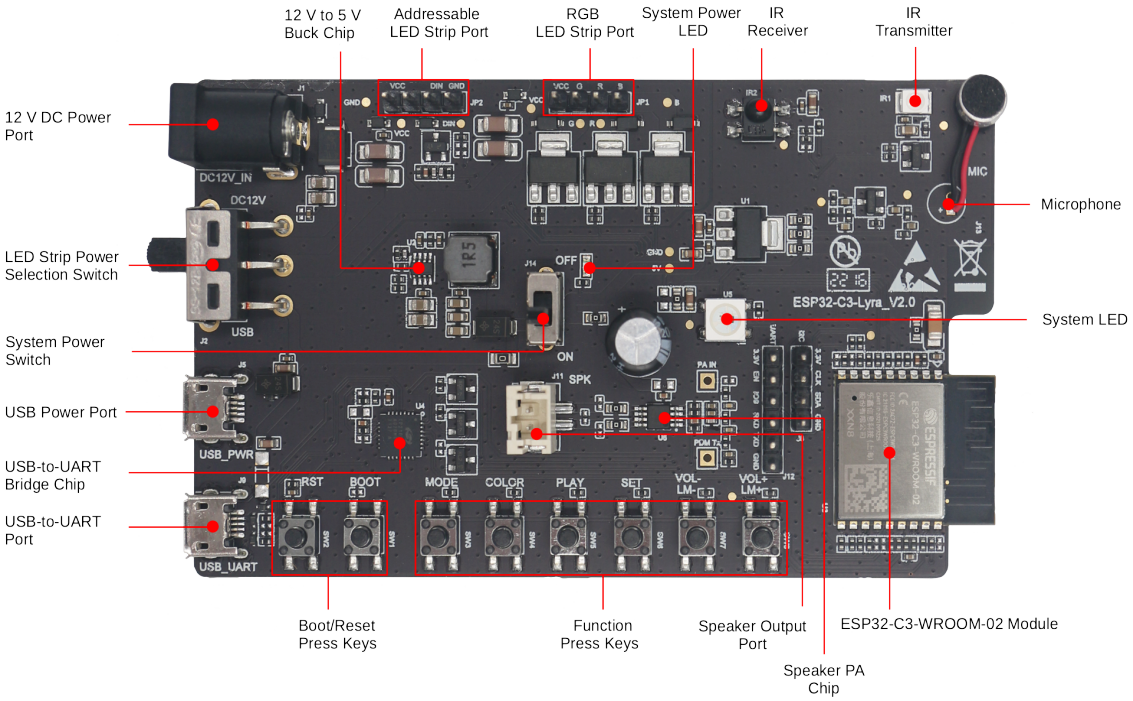


图 48: ESP32-C3-Lyra - 正面（点击放大）

以下按照顺时针的顺序依次介绍开发板上的主要组件。

主要组件	介绍
ESP32-C3-WROOM-02 模组	ESP32-C3-WROOM-02 模组是一款基于 ESP32-C3 开发的通用 Wi-Fi 和蓝牙 LE 模组。ESP32-C3 是一款运行频率高达 160 MHz 的 32 位 RISC-V 单核处理器。该高性能模组集成众多外设，适用于智能家居、工业自动化、医疗保健、消费电子等应用场景。模组包含一个 4 MB 的外部 SPI flash 和一个板载 PCB 天线。ESP32-C3-WROOM-02U 模组也兼容此开发板，但需要外接天线。
扬声器功率放大器 (Speaker PA Chip)	NS4150 是一款 EMI、3 W 单声道 D 类音频功率放大器，用于放大来自 ESP32-C3 PDM_TX 的音频信号，以驱动扬声器。
扬声器输出端口 (Speaker Output Port)	用于连接扬声器的端口。推荐使用 4 欧姆和 3 瓦的扬声器。管脚间距为 2.00 毫米/0.08 英寸。
功能按键 (Function Press Keys)	六个功能按键，包括 MODE、COLOR、PLAY/PAUSE、SET、VOL+/LM+、VOL-/LM-。它们与 **ESP32-C3-WROOM-02 Module** 模组连接，可借助专门的 API 来开发和测试音频应用程序的用户界面或 LED 灯带。
Boot/Reset 按键 (Boot/Reset Press Keys)	Boot：按住 Boot 键并短按 Reset 按钮，启动固件上传模式，通过串口上传固件。Reset：单独按下此键，重置系统。
USB-to-UART 端口 (USB-to-UART Port)	用于 PC 端与 ESP32-C3-WROOM-02 模组的通信。
USB-to-UART 桥接芯片 (USB-to-UART Bridge Chip)	单芯片 USB-UART 桥接器 CP2102N 为软件下载和调试提供高达 3 Mbps 的传输速率。
USB 电源端口 (USB Power Port)	为整个系统提供电源。建议使用至少 5 V/2 A 电源适配器供电，以保证供电稳定。
系统电源开关 (System Power Switch)	系统电源开/关旋钮。将其切换到 **ON** 会打开 5 V 系统电源，切换到 **OFF** 则关闭 5 V 系统电源。
LED 灯带电源选择开关 (LED Strip Power Selection Switch)	拨动此开关，根据 LED 灯带的工作电压和实际使用的电源适配器类型，选择 USB 5 V 供电或 LED 灯带 12 V 直流供电。
12 V 直流供电端口 (12 V DC Power Port)	支持最大电流为 2 A 的 12 V 直流电源适配器。直流电源插孔触点的外径为 5.5 mm，内径为 2.5 mm。
12 V 至 5 V 降压芯片 (12 V to 5 V Buck Chip)	12 V 至 5 V 降压芯片 MP2313 是一款在 1 A 和 2 MHz 情况下工作的高效同步降压转换器。
可寻址 LED 灯带端口 (Addressable LED Strip Port)	可寻址 LED 灯带端口是一个 4 x 1 P，2.54 mm 间距的排针接口，可以连接到通过单线控制的可寻址 LED 灯带。它支持 5 V 和 12 V LED 灯带，例如 WS2811 和 WS2812 LED。ESP32-C3 可以通过 RMT 或 SPI 发送命令来控制 LED 灯带。
RGB LED 灯带端口 (RGB LED Strip Port)	RGB LED 灯带端口是一个 4 x 1 P，2.54 mm 间距的排针接口，可以连接到在 5 V 或 12 V 下运行的常规 RGB LED 灯带（不可寻址，各颜色独立线路控制）。ESP32-C3 可以通过该端口输出 PWM 波形来控制 LED 灯带。
系统电源 LED (System Power LED)	当 **系统电源开关**切换到 ON 时，LED 变为红色。
红外接收器 (IR Receiver)	IRM-H638T/TR2 是一款微型贴片型红外遥控系统接收器。解调后的输出信号可以直接由 ESP32-C3 解码。
红外线发射器 (IR Transmitter)	IR67-21C/TR8 是红外线发光二极管，与硅光电二极管和光电晶体管进行光谱匹配。
麦克风 (Microphone)	板载 ECM 麦克风。它采集的信号通过晶体管放大后送到 ESP32-C3-WROOM-02 的 ADC。

默认固件和功能测试

每个 ESP32-C3-Lyra 都配有预构建的默认固件，支持功能测试，包括 LED 控制 (LEDC)、远程控制收发器 (RMT)、ADC 和脉冲密度调制 (PDM_TX)。本节将介绍如何使用预建固件测试外围设备的功能。

注解： 《ESP32-C3 物联网工程开发实战》套餐二中 ESP32-C3-Lyra 的默认固件为音乐律动灯环效果。

硬件准备

相关详细信息，请参阅*必备硬件* 和*可选硬件* 章节。

- 1 x ESP32-C3-Lyra
- 2 x USB 2.0 数据线（标准 A 型转 Micro-B 型）
- 1 x 电脑（Windows、Linux 或 macOS）
- 1 x 5 V RGB LED 灯带 WS2812（可选）
- 1 x 手机或音乐播放器
- 1 x 扬声器（可选）

硬件连接

- 通电前，请确保开发板完好无损。
- 插入 USB 数据线，通过 **USB 电源端口**将开发板连接到 5 V 电源。开发板通电后，**系统电源 LED** 亮起，代表开发板已通电。如果 LED 不亮，请切换 **系统电源开关**。
- 将 **LED 灯带电源选择开关**切换至 USB 电源侧。
- 插入 USB 数据线，通过 **USB-to-UART 端口**将开发板连接到电脑。

默认固件测试

注解： 若您使用《ESP32-C3 物联网工程开发实战》套餐二中的 ESP32-C3-Lyra 进行开发，请跳过此环节。

---

1. 按下开发板的 **Reset** 键。
2. 开发板自动开始 flash 测试。连接到 USB-to-UART 端口的 PC 上显示日志如下：

```
Step1 Flash Test Start
Step1 Flash Test OK
```

3. 开发板测试 **功能按键**。请按照日志提示按键，例如，当显示以下日志时，按下 **VOL+**：

```
Step2 Keys Test Start
Please press The Key: VOL+
```

4. 开发板测试 **系统 LED**。此时，LED 将在红色、蓝色和绿色之间不断切换。按下 `VOL+/LM+` 键进入下一步。
5. 开发板测试 LEDC (PWM)。将 RGB LED 灯带连接到 **RGB LED 灯带端口**，您将看到 LED 呼吸灯效果。按下 `VOL+/LM+` 键进入下一步。
6. 开发板测试 ADC。用手机或音乐播放器靠近 **麦克风**，播放 1 kHz 正弦音频信号。开发板检测到音频信号后，显示如下：

```
Step5 Adc Test Start
Please play 1khz audio
Step5 Adc Test OK
```

7. 开发板测试 PDM_TX 功能。扬声器连接到 **扬声器输出端口**后，播放 flash 中的音乐。

软件支持

ESP32-C3-Lyra 的开发框架是 ESP-ADF。如需查看本开发板支持的 ESP-ADF 版本，请前往 硬件。

您也可以前往乐鑫开发的其他软件仓库，更加全面地探索 ESP32-C3-Lyra 的相关功能。

- ESP-IDF：基于 FreeRTOS 的乐鑫 SoC 开发框架，具有众多组件，包括 LED 控制 (LEDC)、ADC、RMT、SPI 等。

开发板的应用示例存放在 application example 中。

开始开发应用

本节介绍硬件和软件的设置方法，以及烧录固件至开发板以开发应用程序的说明。

必备硬件

硬件	数量	说明
ESP32-C3-Lyra	1	–
USB 2.0 数据线（标准 A 型转 Micro-B 型）	2	一个用于 USB 电源，另一个用于将固件烧录至开发板。请确保使用适当的 USB 数据线。部分数据线仅可用于充电，无法用于数据传输和编程。
电脑（Windows、Linux 或 macOS）	1	–
扬声器	1	建议使用 4 欧姆 3 瓦的扬声器，并配备 PH 2 mm 间距，1 x 2 P 的插孔端子。如果没有这种类型的插头，也可以使用杜邦母跳线进行开发。

可选硬件

硬件	数量	说明
12 V 直流适配器	1	适配器为 12 V LED 灯带提供电源，最大工作电流为 2 A。
5 V 或 12 V 可寻址 LED 灯带/灯环	1	建议使用 WS2812 或 WS2811 LED 灯带（4 x 1 P，2.54 mm 间距的排母连接器）/16 颗灯珠的 WS2812 LED 灯环（3 x 1 P，2.54 mm 间距的排母连接器）。此 LED 灯带/灯环应连接到 **可寻址 LED 灯带端口** (*JP2*)。
5 V 或 12 V RGB LED 灯带	1	应为带有 4 x 1 P，2.54 mm 间距的排母连接器。此 LED 灯带应连接到 **RGB LED 灯带端口** (*JP1*)。

电源选项

有两种方式为开发板供电：

- **USB 供电端口** (5 V)
- **12 V 直流供电端口**

硬件设置

准备开发板，加载第一个示例应用程序：

1. 连接扬声器至 **扬声器输出端口**。
2. （可选）根据 LED 灯带的类型，将 LED 灯带连接到开发板的可寻址 LED 灯带端口或 RGB LED 灯带端口。

3. 根据负载的工作电压和电流，将电源连接到开发板的 **USB 供电端口** (5 V) 或 **12 V 直流供电端口**。
4. （可选）根据 LED 灯带的工作电压和电流，切换 **LED 灯带电源选择开关**，为 LED 灯带供电。

注解： 如果开关未切换至正确的一侧，灯带将无法正常工作。**请勿**使用 12 V 直流适配器为 5 V LED 灯带供电，否则灯带将损坏。

---

5. 切换 **系统电源开关**至 **ON**。红色 **系统电源 LED** 亮起。
6. 插入 USB 数据线，通过 **USB-to-UART 端口**将开发板连接到电脑。

此时，硬件设置完成，可以进行软件设置。

软件设置

硬件设置完成后，请前往*Get Started*，准备开发工具。

有关开发应用程序的更多软件信息，请访问软件支持。

硬件参考

本节提供有关开发板硬件的更多详细信息。

GPIO 分配列表

表 为 ESP32-C3-WROOM-02 模组管脚的 GPIO 分配列表，用于控制开发板的特定组件或功能。

供电说明

通过 USB 或 12 V/2 A 直流输入供电

开发板有两种供电方式：5 V USB 供电端口或 12 V/2 A 直流输入供电。

LED 灯带电源选择开关

根据灯带的工作电压和电流，选择合适的电源适配器和端口，切换 **LED 灯带电源选择开关**至相应侧，即可为灯带通电。

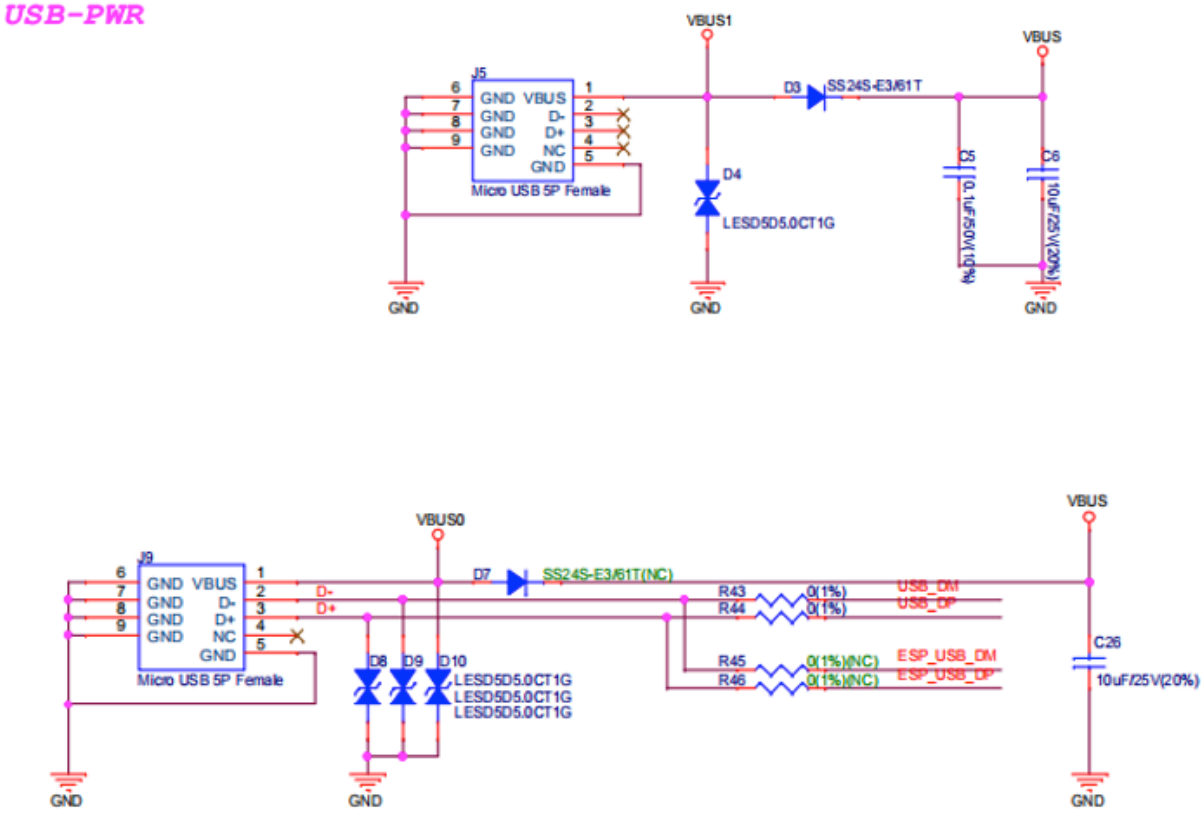


图 49: ESP32-C3-Lyra - 专用 USB 电源端口

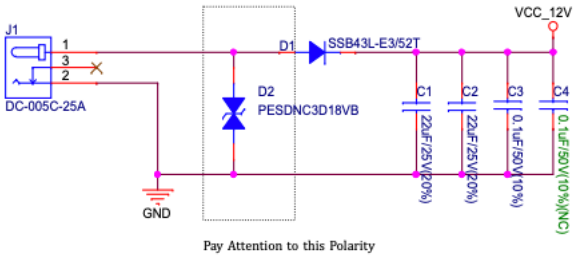


图 50: ESP32-C3-Lyra - 12 V 直流输入供电

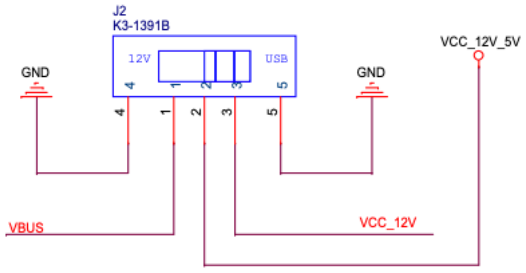


图 51: LED 灯带电源选择开关

12 V 至 5 V 降压电源

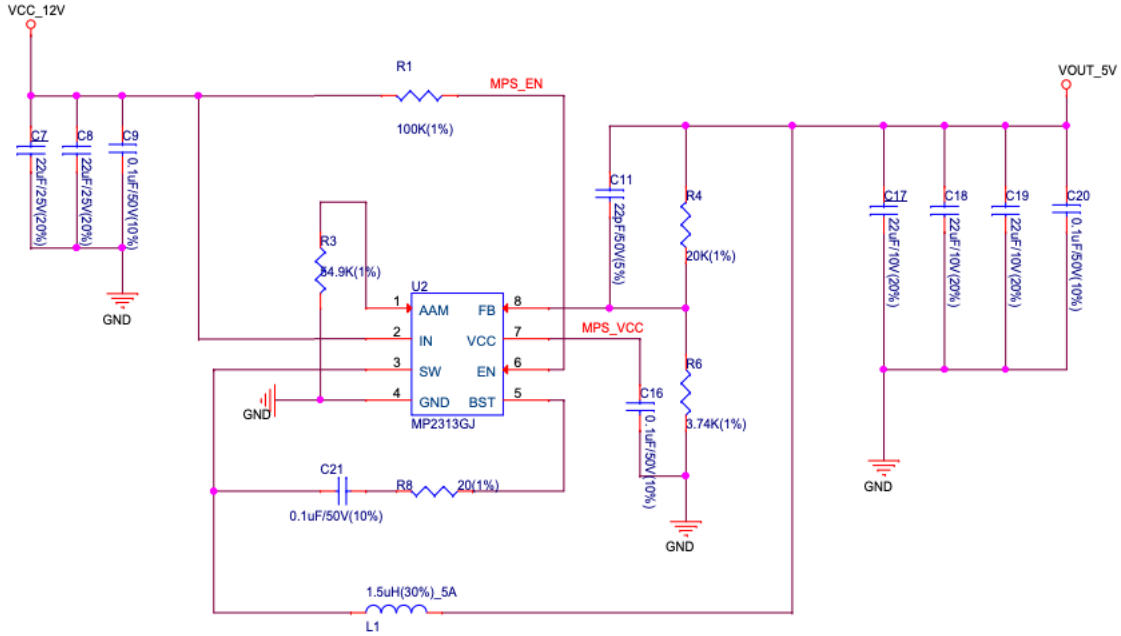


图 52: 12 V 至 5 V 降压电源

系统 3.3 V 电源

连接器

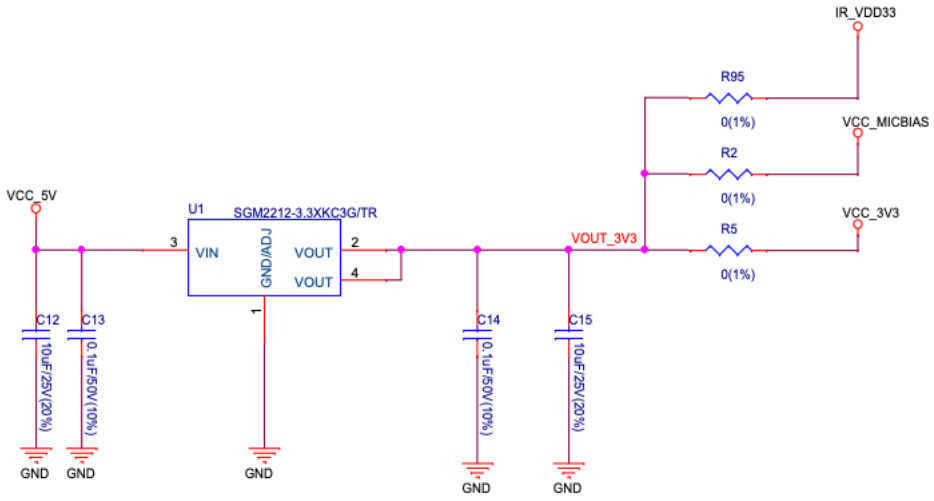


图 53: 系统 3.3 V 电源

RGB LED 灯带连接器 (JP1)

No.	信号	ESP32-C3 管脚
1	VCC_12V_5V	–
2	LED_G	GPIO6
3	LED_R	GPIO5
4	LED_B	GPIO4

可寻址 LED 灯带连接器 (JP2)

No.	信号	ESP32-C3 管脚
1	VCC_12V_5V	–
2	DIN	GPIO7
3	DIN	GPIO7
4	GND	–

扩展排针管脚分布

有多个可用于连接外部组件、检查特定信号总线的状态和调试 ESP32-C3 操作的管脚。请注意，部分管脚共享同个信号。详细信息请参阅*GPIO 分配列表* 章节。

UART 接口 (JP12)

No.	信号	ESP32-C3 管脚
1	VCC_3V3	–
2	ESP_EN	EN
3	ESP_BOOT	GPIO9
4	ESP_UART_RXD	U0RXD
5	ESP_UART_TXD	U0TXD
6	GND	–

I2C 接口 (JP8)

No.	信号	ESP32-C3 管脚
1	VCC_3V3	–
2	I2C_CLK	GPIO8
3	I2C_DATA	GPIO9
4	GND	–

硬件版本

无历史版本。

订购信息

如购买样品，每个开发板将独立包装。

零售订单请前往官方网站 https://www.espressif.com/zh-hans/contact-us/get-samples，或是前往淘宝下单 https://world.taobao.com/item/677273363812.htm?spm=a21wu.12321156-tw.recommend-tpp.4.19a61924ZMaqpf。

批量订单请前往官方网站 https://www.espressif.com/zh-hans/contact-us/sales-questions。

相关文档

- 技术规格书
  - ESP32-C3 系列芯片技术规格书 (PDF)
  - ESP32-C3-WROOM-02 & ESP32-C3-WROOM-02U 规格书 (PDF)
- 原理图
  - ESP32-C3-Lyra 原理图 (PDF)
- PCB 布局图
  - ESP32-C3-Lyra PCB 布局图 (PDF)

有关本开发板的更多设计文档，请联系我们的商务部门 sales@espressif.com。

3.4.12 ESP32-C3-Lyra

3.5 Audio Samples

Music files in this section are intended for testing of audio applications. The files are organized into different *Formats* and *Sample Rates*.

3.5.1 Formats

The tables below provides an audio file converted from ‘wav’ format into several other audio formats.

Long Samples

The audio track duration in this section is 3 minutes and 7 seconds.

Two Channel Audio

No	Format	Audio File	Size [kB]
1	aac	ff-16b-2c-44100hz.aac	2,995
2	ac3	ff-16b-2c-44100hz.ac3	2,994
3	aiff	ff-16b-2c-44100hz.aiff	33,002
4	flac	ff-16b-2c-44100hz.flac	22,406
5	m4a	ff-16b-2c-44100hz.m4a	3,028
6	mp3	ff-16b-2c-44100hz.mp3	2,994
7	mp4	ff-16b-2c-44100hz.mp4	3,079
8	ogg	ff-16b-2c-44100hz.ogg	2,612
9	opus	ff-16b-2c-44100hz.opus	2,598
10	ts	ff-16b-2c-44100hz.ts	5,510
11	wav	ff-16b-2c-44100hz.wav	32,229
12	wma	ff-16b-2c-44100hz.wma	3,227

Playlist containing all above files: ff-16b-2c-playlist.m3u

Single Channel Audio

No	Format	Audio File	Size [kB]
1	aac	ff-16b-1c-44100hz.aac	1,650
2	ac3	ff-16b-1c-44100hz.ac3	2,193
3	aiff	ff-16b-1c-44100hz.aiff	16,115
4	amr	ff-16b-1c-8000hz.amr	299
5	flac	ff-16b-1c-44100hz.flac	10,655
6	m4a	ff-16b-1c-44100hz.m4a	1,628
7	mp3	ff-16b-1c-44100hz.mp3	1,463
8	ogg	ff-16b-1c-44100hz.ogg	1,558
9	opus	ff-16b-1c-44100hz.opus	1,641
10	wav	ff-16b-1c-44100hz.wav	16,115
11	wma	ff-16b-1c-44100hz.wma	3,151

Playlist containing all above files: ff-16b-1c-playlist.m3u

Short Samples

If you need shorter audio files for testing, this section provides 16 seconds audio tracks.

Two Channel Audio

No	Format	Audio File	Size [kB]
1	aac	gs-16b-2c-44100hz.aac	241
2	ac3	gs-16b-2c-44100hz.ac3	380
3	aiff	gs-16b-2c-44100hz.aiff	2,792
4	flac	gs-16b-2c-44100hz.flac	1,336
5	m4a	gs-16b-2c-44100hz.m4a	258
6	mp3	gs-16b-2c-44100hz.mp3	254
7	mp4	gs-16b-2c-44100hz.mp4	259
8	ogg	gs-16b-2c-44100hz.ogg	229
9	opus	gs-16b-2c-44100hz.opus	219
10	ts	gs-16b-2c-44100hz.ts	286
11	wav	gs-16b-2c-44100hz.wav	2,792
12	wma	gs-16b-2c-44100hz.wma	276

Playlist containing all above files: gs-16b-2c-playlist.m3u

Single Channel Audio

No	Format	Audio File	Size [kB]
1	amr	gs-16b-1c-8000hz.amr	25
2	aac	gs-16b-1c-44100hz.aac	137
3	ac3	gs-16b-1c-44100hz.ac3	190
4	aiff	gs-16b-1c-44100hz.aiff	1,397
5	flac	gs-16b-1c-44100hz.flac	645
6	m4a	gs-16b-1c-44100hz.m4a	258
7	mp3	gs-16b-1c-44100hz.mp3	127
8	ogg	gs-16b-1c-44100hz.ogg	144
9	opus	gs-16b-1c-44100hz.opus	132
10	wav	gs-16b-1c-44100hz.wav	1,497
11	wma	gs-16b-1c-44100hz.wma	276

Playlist containing all above files: gs-16b-1c-playlist.m3u

3.5.2 Sample Rates

The files in this section have been prepared by converting a single audio file into different sampling rates defined in MPEG Layer III specification. Both mono and stereo versions of files are provided. The bit depth of files is 16 bits.

	Sample Rate	MPEG III	Channels	Bit Rate	Size
Audio File	[Hz]	ver		[kbit/s]	[kB]
ff-16b-1c-8000hz.mp3	8000	2.5	mono	8	183
ff-16b-1c-11025hz.mp3	11025	2.5	mono	16	366
ff-16b-1c-12000hz.mp3	12000	2.5	mono	16	366
ff-16b-1c-16000hz.mp3	16000	2	mono	24	548
ff-16b-1c-22050hz.mp3	22050	2	mono	32	731
ff-16b-1c-24000hz.mp3	24000	2	mono	32	731
ff-16b-1c-32000hz.mp3	32000	1	mono	48	1,097
ff-16b-1c-44100hz.mp3	44100	1	mono	64	1,462
ff-16b-2c-8000hz.mp3	8000	2.5	joint stereo	24	549
ff-16b-2c-11025hz.mp3	11025	2.5	joint stereo	32	731
ff-16b-2c-12000hz.mp3	12000	2.5	joint stereo	32	731
ff-16b-2c-16000hz.mp3	16000	2	joint stereo	48	1,097
ff-16b-2c-22050hz.mp3	22050	2	joint stereo	64	1,462
ff-16b-2c-24000hz.mp3	24000	2	joint stereo	64	1,462
ff-16b-2c-32000hz.mp3	32000	1	joint stereo	96	2,194
ff-16b-2c-44100hz.mp3	44100	1	joint stereo	128	2,924

Playlist containing all above files: ff-16b-mp3-playlist.m3u

Original music files: "Furious Freak" and "Galway", Kevin MacLeod (incompetech.com), Licensed under Creative Commons: By Attribution 3.0, http://creativecommons.org/licenses/by/3.0/

CHAPTER 4

Resources

- Third party frameworks and libraries to develop audio applications with Espressif chips:
  - The JOSH operating system supports the ESP32 and can be used in scenarios such as intelligent voice interaction, smart home appliances, and smart gateways.
- Third party audio development modules and boards that work with ESP-ADF:
  - ESP32-A1S Audio Module equipped CodeC audio decoding chip that supports music playback and recording, and 4MB PSRAM. The module application schematic is available in datasheet.
- The esp32.com forum is a place to ask questions and find community resources. The forum has a section dedicated to ESP-ADF.
- This ESP Audio Development Framework inherits from ESP IoT Development Framework and you can learn about it in ESP-IDF Programming Guide.
- Check the Issues section on GitHub if you find a bug or have a feature request. Please check existing Issues before opening a new one.
- If you ' re interested in contributing to ESP Audio Development Framework, please check the Contributions Guide.
- Several books have been written about ESP32 and they are listed on Espressif web site.
- For additional ESP32 product related information, please refer to documentation Section of Espressif site.
- To buy audio development boards, check list of distributors under Get Samples on Espressif web site.

CHAPTER 5

Copyrights and Licenses

5.1 Software Copyrights

All original source code in this repository is Copyright (C) 2015-2018 Espressif Systems. This source code is licensed under the ESPRESSIF MIT License as described in the file LICENSE.

Additional third party copyrighted code is included under the following licenses:

- mp3 library is Copyright (c) 2005-2008, The Android Open Source Project, and is licensed under the Apache License Version 2.0.
- aac library is Copyright (c) 2005-2008, The Android Open Source Project, and is licensed under the Apache License Version 2.0.
- amr library is Copyright (C) 2009 Martin Storsjo and is licensed under the Apache License Version 2.0.
- opus library, Copyright 2001-2011 Xiph.Org, Skype Limited, Octasic,Jean-Marc Valin, Timothy B. Terriberry, CSIRO, Gregory Maxwell, Mark Borgerding, Erik de Castro Lopo, is licensed under 3-clause BSD license.
- flac library, Copyright (C) 2011-2016 Xiph.Org Foundation, is licensed under Xiph.Org’ s BSD-like license.
- vorbis library, Copyright (c) 2002-2020 Xiph.org Foundation, is licensed under the 3-Clause BSD license.
- aac-enc library is Copyright 2003-2010, VisualOn, and is licensed under the Apache License Version 2.0.
- adpcm library is Copyright (C) 2020 aikiriao <samuraiaiki@gmail.com>.
- jpeg-dec library is Copyright (C) 2019, ChaN.
- alac library is Copyright (C) 2004, developed by Apple, and is licensed under the Apache License Version 2.0.

Please refer to the COPYRIGHT in ESP-IDF Programming Guide

Where source code headers specify Copyright & License information, this information takes precedence over the summaries made here.

CHAPTER 6

English-Chinese Glossary

This document lists terms that are used in Espressif Audio Development Framework Guide and other audio related documentation. Each term is followed by its Chinese equivalents and some have definitions.

AAC Chinese equivalent: AAC

Abbreviation for Advanced Audio Coding, an industry-standard audio compression format.

acoustic Chinese equivalent: 声学

acoustic calibrator Chinese equivalent: 声校准器

Also known as sound level calibrator.

acoustic echo cancellation Chinese equivalent: 声学回声消除

Spelled-out form of AEC.

AEC Chinese equivalent: AEC

Abbreviation for acoustic echo cancellation.

Advanced Audio Distribution Profile Chinese equivalent: 高级音频分发框架

Spelled-out form of A2DP.

A2DP Chinese equivalent: A2DP.

Abbreviation for Advanced Audio Distribution Profile.

A2DP sink Chinese equivalent: A2DP sink

A2DP source Chinese equivalent: A2DP source

AFE Chinese equivalent: AFE

Abbreviation for audio front end. Espressif audio front-end algorithm framework is developed by Espressif AI Lab to provide high-quality and stable audio data to the host.

AirKiss Chinese equivalent: AirKiss

AirKiss is a quick-connection technique provided by Weixin device platform for Wi-Fi devices to configure network connection.

AMR Chinese equivalent: AMR

Abbreviation for Adaptive Multi-Rate, an audio compression format optimized for speech coding.

AMR-NB Chinese equivalent: AMR-NB

Abbreviation for Adaptive Multi-Rate Narrowband, a narrowband speech audio coding standard developed based on Adaptive Multi-Rate encoding.

AMR-WB Chinese equivalent: AMR-WB

Abbreviation for Adaptive Multi-Rate Wideband, a wideband speech audio coding standard developed based on Adaptive Multi-Rate encoding.

analog-to-digital converter Chinese equivalent: 模拟数字转换器

Spelled-out form of ADC.

ADC Chinese equivalent: ADC

Abbreviation for analog-to-digital converter.

audio codec Chinese equivalent: 音频编码解码器

audio forge Chinese equivalent: 音频塑造

A combination of several audio backend processing techniques, including resample, downmix, automatic level control, equalizer and sonic. Users can enable or disable certain techniques as needed.

audio gate Chinese equivalent: 音频网关

Spelled-out form of AG.

AG Chinese equivalent: AG

Abbreviation for audio gate.

audio front end Chinese equivalent: 声学前端

Spelled-out form of AFE.

audio passthru Chinese equivalent: 音频透传

Also known as pipeline passthru. It is an audio technique that allows audio files to pass through a pipeline unaltered.

audio pipeline Chinese equivalent: 音频管道

Often used as "pipeline". It is a chain of audio processing elements arranged in a particular order so that the output of each element is the input of the next.

Audio Video Remote Control Profile Chinese equivalent: 音视频远程控制规范

Spelled-out form of AVRCP.

AVRCP Chinese equivalent: AVRCP

Abbreviation for Audio Video Remote Control Profile.

automatic gain control Chinese equivalent: 自动增益控制

Spelled-out form of AGC.

AGC Chinese equivalent: AGC

Abbreviation for automatic gain control.

automatic level control Chinese equivalent: 自动电平控制

Spelled-out form of ALC.

ALC Chinese equivalent: ALC

Abbreviation for automatic level control.

automatic speech recognition Chinese equivalent: 自动语音识别

Spelled-out form of ASR.

aux cable Chinese equivalent: aux 音频线

Also known as auxiliary cable.

ASR Chinese equivalent: ASR

Abbreviation for automatic speech recognition.

bandwidth Chinese equivalent: 带宽

Bass Frequency Chinese equivalent: 低频

BCLK Chinese equivalent: BCLK

Abbreviation for base clock.

BluFi Chinese equivalent: BluFi

A Wi-Fi network configuration function via Bluetooth channel. See ESP-IDF Programming Guide for more information.

cavity Chinese equivalent: 腔体

command word Chinese equivalent: 命令词

core dump Chinese equivalent: 核心转储

cutoff frequency Chinese equivalent: 截止频率

decoder Chinese equivalent: 解码器

digital media renderer Chinese equivalent: 数字媒体渲染器

Spelled-out form of DMR.

digital signal processor Chinese equivalent: 数字信号处理器

Spelled-out form of DSP.

DSP Chinese equivalent: DSP

Abbreviation for digital signal processor or digital signal processing.

digital-to-analog converter Chinese equivalent: 数字模拟转换器

Spelled-out form of DAC.

dispatcher Chinese equivalent: 调度器

distortion Chinese equivalent: 失真

DAC Chinese equivalent: DAC

Abbreviation for digital-to-analog converter.

Digital Living Network Alliance Chinese equivalent: 数字生活网络联盟

Spelled-out form of DLNA.

DLNA Chinese equivalent: DLNA

Abbreviation for Digital Living Network Alliance.

DMR Chinese equivalent: DMR

Abbreviation for digital media renderer.

downmix Chinese equivalent: 向下混叠

An audio processing technique that mixes more audio streams to less output audio streams.

DuerOS Chinese equivalent: DuerOS

DuerOS is a conversational AI system developed by Baidu.

echo Chinese equivalent: 回声

A reflection of sound that arrives at the listener with a delay after the direct sound.

echo reference signal Chinese equivalent: 回声参考信号

electret condenser microphone Chinese equivalent: 驻极体麦克风

Spelled-out form of ECM.

ECM Chinese equivalent: ECM

Abbreviation for electret condenser microphone.

element Chinese equivalent: 元素

Also known as audio element. It is the basic building block for the application programmer developing with ADF. Every decoder, encoder, filter, input stream, or output stream is in fact an audio element.

encoder Chinese equivalent: 编码器

equalizer Chinese equivalent: 均衡器

ESP VoIP Chinese equivalent: ESP VoIP

ESP VoIP is a telephone client based on the standard SIP protocol, which can be used in some P2P or audio conference scenarios.

fast Fourier transform Chinese equivalent: 快速傅里叶变换

Spelled-out form of FFT.

FFT Chinese equivalent: FFT

Abbreviation for fast Fourier transform.

FatFs Chinese equivalent: FatFs

FatFs stream Chinese equivalent: FatFs 流

FLAC Chinese equivalent: FLAC

Abbreviation for Free Lossless Audio Codec, an audio coding format for lossless compression of digital audio.

flexible pipeline Chinese equivalent: 灵活管道

FPS Chinese equivalent: FPS

Abbreviation for frames per second.

frames per second Chinese equivalent: 每秒传输帧数

Spelled-out form of FPS.

frequency response Chinese equivalent: 频率响应

full band Chinese equivalent: 全频带

Spelled-out form of FB.

FB Chinese equivalent: FB

Abbreviation for full band.

Hands-Free Chinese equivalent: 免提

Spelled-out form of HF.

HF Chinese equivalent: HF

Abbreviation for Hands-Free.

Hands-Free Audio Gateway Chinese equivalent: 免提音频网关

Spelled-out form of HFP-AG.

Hands-Free Profile Chinese equivalent: 免提规范

Hands-Free Unit Chinese equivalent: 免提组件

HFP Chinese equivalent: HFP

Abbreviation for Hands-Free Profile.

hardware abstraction layer Chinese equivalent: 硬件抽象层

Spelled-out form of HAL.

HAL Chinese equivalent: HAL

Abbreviation for hardware abstraction layer.

headset Chinese equivalent: 耳机

HFP-AG Chinese equivalent: HFP-AG

Abbreviation for Hands-Free Audio Gateway.

Hi-Fi speaker Chinese equivalent: 高保真音箱

Also known as high-fidelity speaker.

High Frequency Chinese equivalent: 高频

high-fidelity microphone Chinese equivalent: 高保真麦克风

HLS Chinese equivalent: HLS

Abbreviation for HTTP Live Streaming.

HTTP stream Chinese equivalent: HTTP 流

HTTP Live Streaming Chinese equivalent: HTTP 直播流

Spelled-out form of HLS.

I2S stream Chinese equivalent: I2S 流

insertion loss Chinese equivalent: 插入损失

Internet radio Chinese equivalent: 网络电台

Internet of Things Chinese equivalent: 物联网

IoT Chinese equivalent: IoT

Abbreviation for Internet of Things.

JPEG Chinese equivalent: JPEG

A commonly used method of lossy compression for digital images. Same as JPG.

JPG Chinese equivalent: JPG

A commonly used method of lossy compression for digital images. Same as JPEG.

Light and Versatile Graphics Library Chinese equivalent: 轻量级多功能图形库

Spelled-out form of LVGL

low-pass filter Chinese equivalent: 低通滤波器

LVGL Chinese equivalent: LVGL

Abbreviation for Light and Versatile Graphics Library.

M3U8 Chinese equivalent: M3U8

The Unicode version of M3U is M3U8, which uses UTF-8-encoded characters.

M4A Chinese equivalent: M4A

An audio encoding format for lossless compression of digital audio.

mass production Chinese equivalent: 量产

maximum output power Chinese equivalent: 最大输出功率

MCLK Chinese equivalent: MCLK

Abbreviation for master clock.

mel-frequency cepstral coefficients Chinese equivalent: 梅尔频率倒谱系数

Spelled-out form of MFCC.

MFCC Chinese equivalent: MFCC

Abbreviation for mel-frequency cepstral coefficients.

microphone Chinese equivalent: 麦克风

mic Chinese equivalent: 麦克风

Informal form for microphone.

micro-electro-mechanical systems microphone Chinese equivalent: 微型机型系统麦克风

Spelled-out form of MEMS mic.

MEMS mic Chinese equivalent: MEMS 麦克风

Abbreviation for micro-electro-mechanical systems microphone.

microphone gain Chinese equivalent: 麦克风增益

microphone hole Chinese equivalent: 麦克孔

microSD card Chinese equivalent: microSD 卡

MP3 Chinese equivalent: MP3

MP4 Chinese equivalent: MP4

MultiNet Chinese equivalent: MultiNet

MultiNet is a lightweight model specially designed based on CRNN and CTC for the implementation of multi-command recognition.

multi-room Chinese equivalent: 多房间

Multi-Room Music Chinese equivalent: Multi-Room Music

ESP Multi-Room Music is a Wi-Fi-based communication protocol to share music across multiple interconnected speakers. Under this protocol, those connected speakers form a Group. They can play music synchronously and are controlled together, which can easily achieve a theater-grade stereo surround sound system.

narrowband Chinese equivalent: 窄带

Spelled-out form of NB.

NB Chinese equivalent: NB

Abbreviation for narrowband.

NimBLE Chinese equivalent: NimBLE

An open-source Bluetooth Low Energy or Bluetooth Smart stack.

noise criteria curve Chinese equivalent: 噪声标准曲线

Also known as NC curve.

noise rating curve Chinese equivalent: 噪声评价曲线

Also known as NR curve.

noise floor Chinese equivalent: 本底噪声

noise suppression Chinese equivalent: 噪声抑制

Spelled-out form of NS.

NS Chinese equivalent: NS

Abbreviation for noise suppression.

non-volatile storage Chinese equivalent: 非易失性存储

Spelled-out form of NVS.

NVS Chinese equivalent: NVS

Abbreviation for non-volatile storage.

OGG Chinese equivalent: OGG

An audio compression format.

OPUS Chinese equivalent: OPUS

A lossy audio coding format.

PCM Chinese equivalent: PCM

Abbreviation for pulse-code modulation.

pixel Chinese equivalent: 像素

playback Chinese equivalent: 回放

It is a noun. The verb is “play back”.

programmable gain amplifier Chinese equivalent: 可编程增益放大器

Spelled-out form of PGA.

PGA Chinese equivalent: PGA

Abbreviation for programmable gain amplifier.

protractor Chinese equivalent: 量角尺

pulse-code modulation Chinese equivalent: 脉冲编码调制

Spelled-out form of PCM.

raw stream Chinese equivalent: 原始流

resample Chinese equivalent: 重采样

resample filter Chinese equivalent: 重采样过滤器

resonant frequency Chinese equivalent: 谐振频率

reverberation Chinese equivalent: 混响

RGB Chinese equivalent: RGB

The RGB color model is an additive color model in which the red, green, and blue primary colors of light are added together in various ways to reproduce a broad array of colors.

ring buffer Chinese equivalent: 环形缓冲区

SBC Chinese equivalent: SBC

Abbreviation for subband codec.

SD card Chinese equivalent: SD 卡

Session Initiation Protocol Chinese equivalent: 会话发起协议

Spelled-out form of SIP.

signal-to-echo ratio Chinese equivalent: 信回比

signal-to-noise ratio Chinese equivalent: 信噪比

Spelled-out form of SNR.

SIP Chinese equivalent: SIP

Abbreviation for Session Initiation Protocol.

SNR Chinese equivalent: SNR

Abbreviation for signal-to-noise ratio.

SmartConfig Chinese equivalent: SmartConfig

The SmartConfig ™ is a provisioning technology developed by TI to connect a new Wi-Fi device to a Wi-Fi network. It uses a mobile app to broadcast the network credentials from a smartphone, or a tablet, to an unprovisioned Wi-Fi device.

sonic Chinese equivalent: 变声

An audio processing technique that modifies sound frequency and speed.

sound card Chinese equivalent: 声卡

Also known as audio card.

sound level meter Chinese equivalent: 分贝仪

Also known as sound pressure level meter.

sound pickup hole Chinese equivalent: 拾音孔

sound pickup tube Chinese equivalent: 拾音管道

sound transmission loss Chinese equivalent: 传声损失

Spelled-out form of STL.

speech Chinese equivalent: 语音

speech recognition Chinese equivalent: 语音识别

Spelled-out form of SR.

SR Chinese equivalent: SR

Abbreviation for speech recognition.

SPI Flash File System Chinese equivalent: SPI 闪存文件系统

Spelled-out form of SPIFFS.

SPIFFS Chinese equivalent: SPIFFS

Abbreviation for SPI Flash File System.

SPIFFS stream Chinese equivalent: SPIFFS 流

subband codec Chinese equivalent: 次频带编码

Spelled-out form of SBC.

STL Chinese equivalent: STL

Abbreviation for sound transmission loss.

super wide band Chinese equivalent: 超宽频带

Spelled-out form of SWB.

SWB Chinese equivalent: SWB

Abbreviation for super wide band.

tape measure Chinese equivalent: 卷尺

text-to-speech Chinese equivalent: 语音合成

Spelled-out form of TTS.

TTS Chinese equivalent: TTS

Abbreviation for text-to-speech.

tolerance Chinese equivalent: 公差

tone Chinese equivalent: 提示音

total harmonic distortion Chinese equivalent: 总谐波失真

Spelled-out form of THD.

THD Chinese equivalent: THD

Abbreviation for total harmonic distortion.

voice activity detection Chinese equivalent: 语音活动检测

Spelled-out form of VAD.

VAD Chinese equivalent: VAD

Abbreviation for voice activity detection.

VoIP Chinese equivalent: VoIP

Abbreviation for Voice over Internet Protocol.

wake word Chinese equivalent: 唤醒词

wake word engine Chinese equivalent: 唤醒词引擎

Spelled-out form of WWE.

WakeNet Chinese equivalent: WakeNet

WakeNet is a wake word engine built upon neural network for low-power embedded MCUs.

wake-up Chinese equivalent: 唤醒

It is a noun.

wideband Chinese equivalent: 宽带

Spelled-out form of WB.

WB Chinese equivalent: WB

Abbreviation for wideband.

WWE Chinese equivalent: WWE

Abbreviation for wake word engine.

YUV Chinese equivalent: YUV

A color model typically used as part of a color image pipeline.

CHAPTER 7

About

This is documentation of ESP-ADF, the framework to develop audio applications for ESP32 chip by Espressif.

The **ESP32** is 2.4 GHz Wi-Fi and Bluetooth combo, 32 bit dual core chip running up to 240 MHz, designed for mobile, wearable electronics, and Internet-of-Things (IoT) applications. It has several peripherals on board including I2S interfaces to easy integrate with dedicated audio chips. These hardware features together with the ESP-ADF software provide a powerful platform to implement audio applications including native wireless networking and powerful user interface.

The **ESP-ADF** provides a range of API components including **Audio Streams**, **Codecs** and **Services** organized in **Audio Pipeline**, all integrated with audio hardware through **Media HAL** and with **Peripherals** onboard of **ESP32**.

The ESP-ADF also provides integration with **Baidu DauerOS** cloud services. A range of components is coming to provide integration with DeepBrain, Amazon, Google, Alibaba and Turing cloud services.

The **ESP-ADF** builds on well established, FreeRTOS based, Espressif IOT Development Framework ESP-IDF.

- genindex

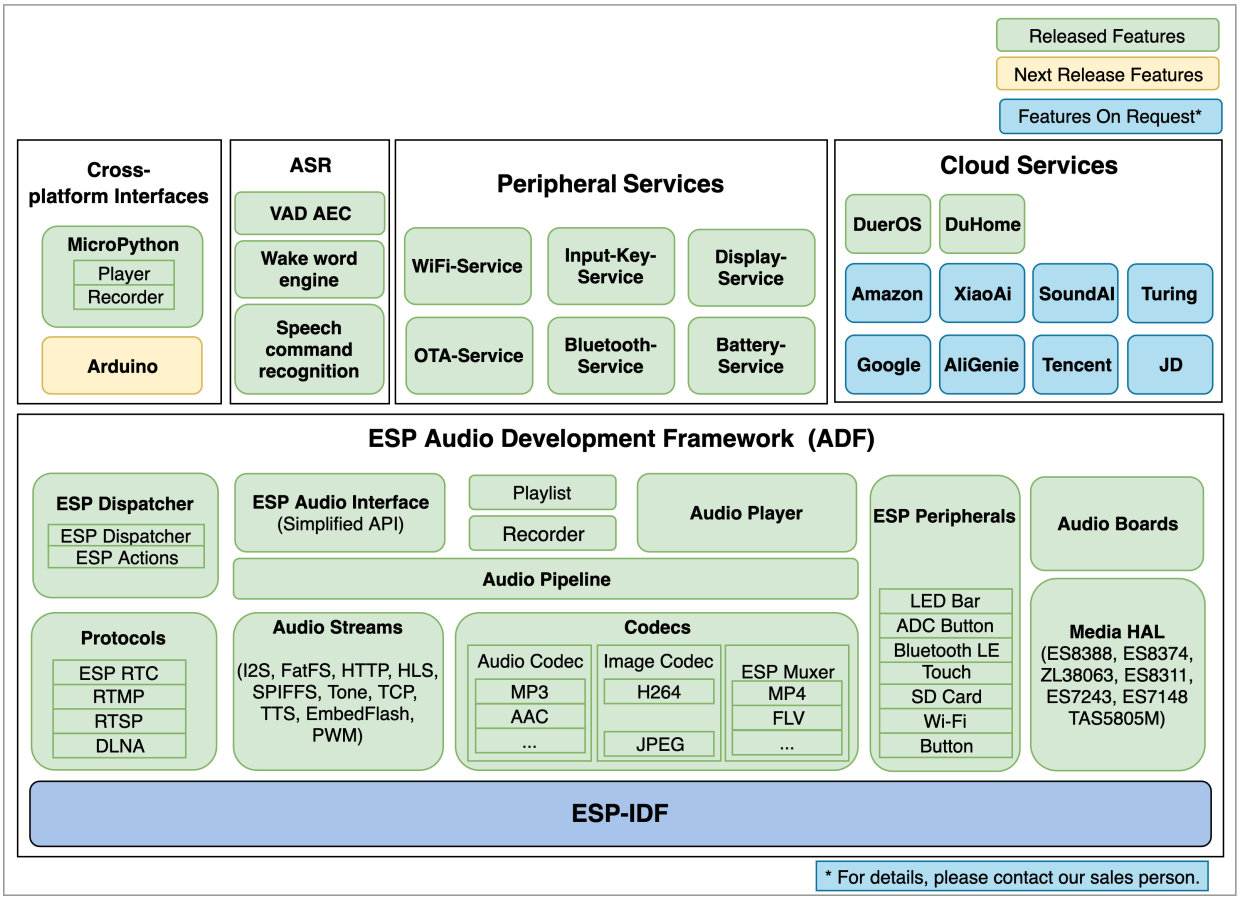


图 1: Espressif Audio Development Framework

索引

A

A2DP, **411**
A2DP sink, **411**
A2DP source, **411**
AAC, **411**
aac_decoder_cfg_t (*C++ class*), 137
aac_decoder_cfg_t::out_rb_size (*C++ member*), 137
aac_decoder_cfg_t::plus_enable (*C++ member*), 137
aac_decoder_cfg_t::stack_in_ext (*C++ member*), 137
aac_decoder_cfg_t::task_core (*C++ member*), 137
aac_decoder_cfg_t::task_prio (*C++ member*), 137
aac_decoder_cfg_t::task_stack (*C++ member*), 137
aac_decoder_init (*C++ function*), 136
AAC_DECODER_RINGBUFFER_SIZE (*C 宏*), 137
AAC_DECODER_TASK_CORE (*C 宏*), 137
AAC_DECODER_TASK_PRIO (*C 宏*), 137
AAC_DECODER_TASK_STACK_SIZE (*C 宏*), 137
acoustic, **411**
acoustic calibrator, **411**
acoustic echo cancellation, **411**
ADC, **412**
ADC_BUTTON_STACK_SIZE (*C 宏*), 258
ADC_BUTTON_TASK_CORE_ID (*C 宏*), 258
ADC_BUTTON_TASK_PRIORITY (*C 宏*), 258
ADC_DEFAULT_ARR (*C 宏*), 258
Advanced Audio Distribution Profile, **411**
AEC, **411**
AEL_IO_ABORT (*C++ enumerator*), 44
AEL_IO_DONE (*C++ enumerator*), 44
AEL_IO_FAIL (*C++ enumerator*), 44
AEL_IO_OK (*C++ enumerator*), 44
AEL_IO_TIMEOUT (*C++ enumerator*), 44
AEL_MSG_CMD_DESTROY (*C++ enumerator*), 45
AEL_MSG_CMD_FINISH (*C++ enumerator*), 44
AEL_MSG_CMD_NONE (*C++ enumerator*), 44
AEL_MSG_CMD_PAUSE (*C++ enumerator*), 44
AEL_MSG_CMD_REPORT_CODEC_FMT (*C++ enumerator*), 45
AEL_MSG_CMD_REPORT_MUSIC_INFO (*C++ enumerator*), 45
AEL_MSG_CMD_REPORT_POSITION (*C++ enumerator*), 45
AEL_MSG_CMD_REPORT_STATUS (*C++ enumerator*), 45
AEL_MSG_CMD_RESUME (*C++ enumerator*), 44
AEL_MSG_CMD_STOP (*C++ enumerator*), 44
AEL_PROCESS_FAIL (*C++ enumerator*), 44
AEL_STATE_ERROR (*C++ enumerator*), 44
AEL_STATE_FINISHED (*C++ enumerator*), 44
AEL_STATE_INIT (*C++ enumerator*), 44
AEL_STATE_INITIALIZING (*C++ enumerator*), 44
AEL_STATE_NONE (*C++ enumerator*), 44
AEL_STATE_PAUSED (*C++ enumerator*), 44
AEL_STATE_RUNNING (*C++ enumerator*), 44

AEL_STATE_STOPPED (C++ enumerator), 44
AEL_STATUS_ERROR_CLOSE (C++ enumerator), 45
AEL_STATUS_ERROR_INPUT (C++ enumerator), 45
AEL_STATUS_ERROR_OPEN (C++ enumerator), 45
AEL_STATUS_ERROR_OUTPUT (C++ enumerator), 45
AEL_STATUS_ERROR_PROCESS (C++ enumerator), 45
AEL_STATUS_ERROR_TIMEOUT (C++ enumerator), 45
AEL_STATUS_ERROR_UNKNOWN (C++ enumerator), 45
AEL_STATUS_INPUT_BUFFERING (C++ enumerator), 45
AEL_STATUS_INPUT_DONE (C++ enumerator), 45
AEL_STATUS_MOUNTED (C++ enumerator), 45
AEL_STATUS_NONE (C++ enumerator), 45
AEL_STATUS_OUTPUT_BUFFERING (C++ enumerator), 45
AEL_STATUS_OUTPUT_DONE (C++ enumerator), 45
AEL_STATUS_STATE_FINISHED (C++ enumerator), 45
AEL_STATUS_STATE_PAUSED (C++ enumerator), 45
AEL_STATUS_STATE_RUNNING (C++ enumerator), 45
AEL_STATUS_STATE_STOPPED (C++ enumerator), 45
AEL_STATUS_UNMOUNTED (C++ enumerator), 45
AFE, **412**
AG, **412**
AGC, **413**
AirKiss, **412**
airkiss_config_create (C++ function), 183
AIRKISS_CONFIG_INFO_DEFAULT (C 宏), 184
airkiss_config_info_t (C++ class), 183
airkiss_config_info_t::aes_key (C++ member), 184
airkiss_config_info_t::lan_pack (C++ member), 184
airkiss_config_info_t::ssdp_notify_enable (C++ member), 184
airkiss_lan_pack_param_t (C++ class), 183
airkiss_lan_pack_param_t::appid (C++ member), 183
airkiss_lan_pack_param_t::deviceid (C++ member), 183
ALC, **413**
algo_stream_init (C++ function), 83
algo_stream_set_delay (C++ function), 83
algorithm_mono_fix (C++ function), 84
ALGORITHM_STREAM_CFG_DEFAULT (C 宏), 85
algorithm_stream_cfg_t (C++ class), 84
algorithm_stream_cfg_t::aec_low_cost (C++ member), 85
algorithm_stream_cfg_t::agc_gain (C++ member), 85
algorithm_stream_cfg_t::algo_mask (C++ member), 85
algorithm_stream_cfg_t::debug_input (C++ member), 84
algorithm_stream_cfg_t::input_type (C++ member), 84
algorithm_stream_cfg_t::mic_ch (C++ member), 85
algorithm_stream_cfg_t::out_rb_size (C++ member), 84
algorithm_stream_cfg_t::rec_linear_factor (C++ member), 84
algorithm_stream_cfg_t::ref_linear_factor (C++ member), 84
algorithm_stream_cfg_t::sample_rate (C++ member), 85
algorithm_stream_cfg_t::stack_in_ext (C++ member), 84
algorithm_stream_cfg_t::swap_ch (C++ member), 85
algorithm_stream_cfg_t::task_core (C++ member), 84
algorithm_stream_cfg_t::task_prio (C++ member), 84
algorithm_stream_cfg_t::task_stack (C++ member), 84
ALGORITHM_STREAM_DEFAULT_AGC_GAIN_DB (C 宏), 85
ALGORITHM_STREAM_DEFAULT_MASK (C 宏), 85

ALGORITHM_STREAM_DEFAULT_MIC_CHANNELS (*C 宏*), 85
ALGORITHM_STREAM_DEFAULT_SAMPLE_BIT (*C 宏*), 85
ALGORITHM_STREAM_DEFAULT_SAMPLE_RATE_HZ (*C 宏*), 85
ALGORITHM_STREAM_INPUT_TYPE1 (*C++ enumerator*), 86
ALGORITHM_STREAM_INPUT_TYPE2 (*C++ enumerator*), 86
algorithm_stream_input_type_t (*C++ enum*), 86
algorithm_stream_mask_t (*C++ enum*), 86
ALGORITHM_STREAM_PINNED_TO_CORE (*C 宏*), 85
ALGORITHM_STREAM_RINGBUFFER_SIZE (*C 宏*), 85
ALGORITHM_STREAM_TASK_PERIOD (*C 宏*), 85
ALGORITHM_STREAM_TASK_STACK_SIZE (*C 宏*), 85
ALGORITHM_STREAM_USE_AEC (*C++ enumerator*), 86
ALGORITHM_STREAM_USE_AGC (*C++ enumerator*), 86
ALGORITHM_STREAM_USE_NS (*C++ enumerator*), 86
ALGORITHM_STREAM_USE_VAD (*C++ enumerator*), 86
AMR, **412**
amr_decoder_cfg_t (*C++ class*), 138
amr_decoder_cfg_t::out_rb_size (*C++ member*), 138
amr_decoder_cfg_t::stack_in_ext (*C++ member*), 138
amr_decoder_cfg_t::task_core (*C++ member*), 138
amr_decoder_cfg_t::task_prio (*C++ member*), 138
amr_decoder_cfg_t::task_stack (*C++ member*), 138
amr_decoder_init (*C++ function*), 138
AMR_DECODER_RINGBUFFER_SIZE (*C 宏*), 139
AMR_DECODER_TASK_CORE (*C 宏*), 139
AMR_DECODER_TASK_PRIO (*C 宏*), 139
AMR_DECODER_TASK_STACK_SIZE (*C 宏*), 139
AMR-NB, **412**
AMR-WB, **412**
AMRNB_ENC_BITRATE_MR102 (*C++ enumerator*), 141
AMRNB_ENC_BITRATE_MR122 (*C++ enumerator*), 141
AMRNB_ENC_BITRATE_MR475 (*C++ enumerator*), 141
AMRNB_ENC_BITRATE_MR515 (*C++ enumerator*), 141
AMRNB_ENC_BITRATE_MR59 (*C++ enumerator*), 141
AMRNB_ENC_BITRATE_MR67 (*C++ enumerator*), 141
AMRNB_ENC_BITRATE_MR74 (*C++ enumerator*), 141
AMRNB_ENC_BITRATE_MR795 (*C++ enumerator*), 141
AMRNB_ENC_BITRATE_MRDTX (*C++ enumerator*), 141
AMRNB_ENC_BITRATE_N_MODES (*C++ enumerator*), 141
AMRNB_ENC_BITRATE_UNKNOW (*C++ enumerator*), 141
amrnb_encoder_bitrate_t (*C++ enum*), 141
amrnb_encoder_cfg_t (*C++ class*), 140
amrnb_encoder_cfg_t::bitrate_mode (*C++ member*), 140
amrnb_encoder_cfg_t::contain_amrnb_header (*C++ member*), 140
amrnb_encoder_cfg_t::out_rb_size (*C++ member*), 140
amrnb_encoder_cfg_t::stack_in_ext (*C++ member*), 140
amrnb_encoder_cfg_t::task_core (*C++ member*), 140
amrnb_encoder_cfg_t::task_prio (*C++ member*), 140
amrnb_encoder_cfg_t::task_stack (*C++ member*), 140
amrnb_encoder_init (*C++ function*), 139
AMRNB_ENCODER_RINGBUFFER_SIZE (*C 宏*), 140
amrnb_encoder_set_bitrate (*C++ function*), 139

AMRNB_ENCODER_TASK_CORE (*C 宏*), 140
AMRNB_ENCODER_TASK_PRIO (*C 宏*), 140
AMRNB_ENCODER_TASK_STACK (*C 宏*), 140
AMRWB_ENC_BITRATE_MD1265 (*C++ enumerator*), 143
AMRWB_ENC_BITRATE_MD1425 (*C++ enumerator*), 143
AMRWB_ENC_BITRATE_MD1585 (*C++ enumerator*), 143
AMRWB_ENC_BITRATE_MD1825 (*C++ enumerator*), 143
AMRWB_ENC_BITRATE_MD1985 (*C++ enumerator*), 143
AMRWB_ENC_BITRATE_MD2305 (*C++ enumerator*), 143
AMRWB_ENC_BITRATE_MD2385 (*C++ enumerator*), 143
AMRWB_ENC_BITRATE_MD66 (*C++ enumerator*), 143
AMRWB_ENC_BITRATE_MD885 (*C++ enumerator*), 143
AMRWB_ENC_BITRATE_MDNONE (*C++ enumerator*), 143
AMRWB_ENC_BITRATE_N_MODES (*C++ enumerator*), 143
amrwb_encoder_bitrate_t (*C++ enum*), 143
amrwb_encoder_cfg_t (*C++ class*), 142
amrwb_encoder_cfg_t::bitrate_mode (*C++ member*), 142
amrwb_encoder_cfg_t::contain_amrwb_header (*C++ member*), 142
amrwb_encoder_cfg_t::out_rb_size (*C++ member*), 142
amrwb_encoder_cfg_t::stack_in_ext (*C++ member*), 142
amrwb_encoder_cfg_t::task_core (*C++ member*), 142
amrwb_encoder_cfg_t::task_prio (*C++ member*), 142
amrwb_encoder_cfg_t::task_stack (*C++ member*), 142
amrwb_encoder_init (*C++ function*), 141
AMRWB_ENCODER_RINGBUFFER_SIZE (*C 宏*), 142
amrwb_encoder_set_bitrate (*C++ function*), 141
AMRWB_ENCODER_TASK_CORE (*C 宏*), 142
AMRWB_ENCODER_TASK_PRIO (*C 宏*), 142
AMRWB_ENCODER_TASK_STACK (*C 宏*), 142
analog-to-digital converter, **412**
ASR, **413**
audio codec, **412**
audio forge, **412**
audio front end, **412**
audio gate, **412**
audio passthru, **412**
audio pipeline, **412**
Audio Video Remote Control Profile, **413**
AUDIO_CODEC_TYPE_DECODER (*C++ enumerator*), 64
AUDIO_CODEC_TYPE_ENCODER (*C++ enumerator*), 64
AUDIO_CODEC_TYPE_NONE (*C++ enumerator*), 64
audio_codec_type_t (*C++ enum*), 64
audio_element_abort_input_ringbuf (*C++ function*), 27
audio_element_abort_output_ringbuf (*C++ function*), 27
audio_element_cfg_t (*C++ class*), 42
audio_element_cfg_t::buffer_len (*C++ member*), 42
audio_element_cfg_t::close (*C++ member*), 42
audio_element_cfg_t::data (*C++ member*), 43
audio_element_cfg_t::destroy (*C++ member*), 42
audio_element_cfg_t::multi_in_rb_num (*C++ member*), 43
audio_element_cfg_t::multi_out_rb_num (*C++ member*), 43
audio_element_cfg_t::open (*C++ member*), 42
audio_element_cfg_t::out_rb_size (*C++ member*), 43
audio_element_cfg_t::process (*C++ member*), 42
audio_element_cfg_t::read (*C++ member*), 42

audio_element_cfg_t::seek (C++ member), 42
audio_element_cfg_t::stack_in_ext (C++ member), 43
audio_element_cfg_t::tag (C++ member), 43
audio_element_cfg_t::task_core (C++ member), 43
audio_element_cfg_t::task_prio (C++ member), 42
audio_element_cfg_t::task_stack (C++ member), 42
audio_element_cfg_t::write (C++ member), 42
audio_element_change_cmd (C++ function), 30
audio_element_deinit (C++ function), 21
audio_element_err_t (C++ enum), 44
audio_element_finish_state (C++ function), 30
audio_element_get_event_queue (C++ function), 33
audio_element_get_input_ringbuf (C++ function), 26
audio_element_get_multi_input_ringbuf (C++ function), 35
audio_element_get_multi_output_ringbuf (C++ function), 36
audio_element_get_output_ringbuf (C++ function), 27
audio_element_get_output_ringbuf_size (C++ function), 33
audio_element_get_read_cb (C++ function), 32
audio_element_get_state (C++ function), 27
audio_element_get_tag (C++ function), 22
audio_element_get_uri (C++ function), 23
audio_element_get_write_cb (C++ function), 32
audio_element_getdata (C++ function), 21
audio_element_getinfo (C++ function), 22
audio_element_handle_t (C++ type), 43
AUDIO_ELEMENT_INFO_DEFAULT (C 宏), 43
audio_element_info_t (C++ class), 41
audio_element_info_t::bits (C++ member), 41
audio_element_info_t::bps (C++ member), 41
audio_element_info_t::byte_pos (C++ member), 41
audio_element_info_t::channels (C++ member), 41
audio_element_info_t::codec_fmt (C++ member), 42
audio_element_info_t::duration (C++ member), 42
audio_element_info_t::reserve_data (C++ member), 42
audio_element_info_t::sample_rates (C++ member), 41
audio_element_info_t::total_bytes (C++ member), 41
audio_element_info_t::uri (C++ member), 42
audio_element_init (C++ function), 20
audio_element_input (C++ function), 31
audio_element_is_stopping (C++ function), 37
audio_element_msg_cmd_t (C++ enum), 44
audio_element_msg_remove_listener (C++ function), 26
audio_element_msg_set_listener (C++ function), 25
audio_element_multi_input (C++ function), 34
audio_element_multi_output (C++ function), 34
audio_element_output (C++ function), 31
audio_element_pause (C++ function), 25
audio_element_process_deinit (C++ function), 36
audio_element_process_init (C++ function), 36
audio_element_report_codec_fmt (C++ function), 29
audio_element_report_info (C++ function), 28
audio_element_report_pos (C++ function), 29
audio_element_report_status (C++ function), 28
audio_element_reserve_data_t (C++ class), 41
audio_element_reserve_data_t::user_data_0

(C++ member), 41
audio_element_reserve_data_t::user_data_1 (C++ member), 41
audio_element_reserve_data_t::user_data_2 (C++ member), 41
audio_element_reserve_data_t::user_data_3 (C++ member), 41
audio_element_reserve_data_t::user_data_4 (C++ member), 41
audio_element_reset_input_ringbuf (C++ function), 30
audio_element_reset_output_ringbuf (C++ function), 30
audio_element_reset_state (C++ function), 33
audio_element_resume (C++ function), 25
audio_element_run (C++ function), 23
audio_element_seek (C++ function), 36
audio_element_set_bps (C++ function), 38
audio_element_set_byte_pos (C++ function), 37
audio_element_set_codec_fmt (C++ function), 38
audio_element_set_duration (C++ function), 39
audio_element_set_event_callback (C++ function), 25
audio_element_set_input_ringbuf (C++ function), 26
audio_element_set_input_timeout (C++ function), 29
audio_element_set_multi_input_ringbuf (C++ function), 35
audio_element_set_multi_output_ringbuf (C++ function), 35
audio_element_set_music_info (C++ function), 39
audio_element_set_output_ringbuf (C++ function), 27
audio_element_set_output_ringbuf_size (C++ function), 34
audio_element_set_output_timeout (C++ function), 29
audio_element_set_read_cb (C++ function), 31
audio_element_set_reserve_user0 (C++ function), 39
audio_element_set_reserve_user1 (C++ function), 39
audio_element_set_reserve_user2 (C++ function), 40
audio_element_set_reserve_user3 (C++ function), 40
audio_element_set_reserve_user4 (C++ function), 40
audio_element_set_ringbuf_done (C++ function), 33
audio_element_set_tag (C++ function), 21
audio_element_set_total_bytes (C++ function), 38
audio_element_set_uri (C++ function), 22
audio_element_set_write_cb (C++ function), 32
audio_element_setdata (C++ function), 21
audio_element_setinfo (C++ function), 22
audio_element_state_t (C++ enum), 44
audio_element_status_t (C++ enum), 45
audio_element_stop (C++ function), 24
audio_element_terminate (C++ function), 23
audio_element_terminate_with_ticks (C++ function), 23
AUDIO_ELEMENT_TYPE_ELEMENT (C++ enumerator), 63
AUDIO_ELEMENT_TYPE_PERIPH (C++ enumerator), 64
AUDIO_ELEMENT_TYPE_PLAYER (C++ enumerator), 63
AUDIO_ELEMENT_TYPE_SERVICE (C++ enumerator), 63
audio_element_type_t (C++ enum), 63
AUDIO_ELEMENT_TYPE_UNKNOW (C++ enumerator), 63
audio_element_update_byte_pos (C++ function), 37
audio_element_update_total_bytes (C++ function), 37

audio_element_wait_for_buffer (C++ function), 28
audio_element_wait_for_stop (C++ function), 24
audio_element_wait_for_stop_ms (C++ function), 24
audio_err_t (C++ enum), 65
audio_event_iface_cfg_t (C++ class), 62
audio_event_iface_cfg_t::context (C++ member), 62
audio_event_iface_cfg_t::external_queue_size (C++ member), 62
audio_event_iface_cfg_t::internal_queue_size (C++ member), 62
audio_event_iface_cfg_t::on_cmd (C++ member), 62
audio_event_iface_cfg_t::queue_set_size (C++ member), 62
audio_event_iface_cfg_t::type (C++ member), 63
audio_event_iface_cfg_t::wait_time (C++ member), 62
audio_event_iface_cmd (C++ function), 59
audio_event_iface_cmd_from_isr (C++ function), 59
AUDIO_EVENT_IFACE_DEFAULT_CFG (C 宏), 63
audio_event_iface_destroy (C++ function), 58
audio_event_iface_discard (C++ function), 60
audio_event_iface_get_msg_queue_handle (C++ function), 61
audio_event_iface_get_queue_handle (C++ function), 61
audio_event_iface_handle_t (C++ type), 63
audio_event_iface_init (C++ function), 58
audio_event_iface_listen (C++ function), 60
audio_event_iface_msg_t (C++ class), 62
audio_event_iface_msg_t::cmd (C++ member), 62
audio_event_iface_msg_t::data (C++ member), 62
audio_event_iface_msg_t::data_len (C++ member), 62
audio_event_iface_msg_t::need_free_data (C++ member), 62
audio_event_iface_msg_t::source (C++ member), 62
audio_event_iface_msg_t::source_type (C++ member), 62
audio_event_iface_read (C++ function), 61
audio_event_iface_remove_listener (C++ function), 58
audio_event_iface_sendout (C++ function), 60
audio_event_iface_set_cmd_waiting_timeout (C++ function), 59
audio_event_iface_set_listener (C++ function), 58
audio_event_iface_set_msg_listener (C++ function), 61
audio_event_iface_waiting_cmd_msg (C++ function), 59
audio_hal (C++ class), 272
audio_hal::audio_codec_config_iface (C++ member), 272
audio_hal::audio_codec_ctrl (C++ member), 272
audio_hal::audio_codec_deinitialize (C++ member), 272
audio_hal::audio_codec_enable_pa (C++ member), 272
audio_hal::audio_codec_get_volume (C++ member), 272
audio_hal::audio_codec_initialize (C++ member), 272
audio_hal::audio_codec_set_mute (C++ member), 272
audio_hal::audio_codec_set_volume (C++ member), 272
audio_hal::audio_hal_lock (C++ member), 272
audio_hal::handle (C++ member), 272
AUDIO_HAL_08K_SAMPLES (C++ enumerator), 274
AUDIO_HAL_11K_SAMPLES (C++ enumerator), 274
AUDIO_HAL_16K_SAMPLES (C++ enumerator), 274
AUDIO_HAL_22K_SAMPLES (C++ enumerator), 274

AUDIO_HAL_24K_SAMPLES (C++ enumerator), 275
AUDIO_HAL_32K_SAMPLES (C++ enumerator), 275
AUDIO_HAL_44K_SAMPLES (C++ enumerator), 275
AUDIO_HAL_48K_SAMPLES (C++ enumerator), 275
AUDIO_HAL_ADC_INPUT_ALL (C++ enumerator), 273
AUDIO_HAL_ADC_INPUT_DIFFERENCE (C++ enumerator), 273
AUDIO_HAL_ADC_INPUT_LINE1 (C++ enumerator), 273
AUDIO_HAL_ADC_INPUT_LINE2 (C++ enumerator), 273
audio_hal_adc_input_t (C++ enum), 273
AUDIO_HAL_BIT_LENGTH_16BITS (C++ enumerator), 275
AUDIO_HAL_BIT_LENGTH_24BITS (C++ enumerator), 275
AUDIO_HAL_BIT_LENGTH_32BITS (C++ enumerator), 275
audio_hal_codec_config_t (C++ class), 271
audio_hal_codec_config_t::adc_input (C++ member), 272
audio_hal_codec_config_t::codec_mode (C++ member), 272
audio_hal_codec_config_t::dac_output (C++ member), 272
audio_hal_codec_config_t::i2s_iface (C++ member), 272
audio_hal_codec_i2s_iface_t (C++ class), 271
audio_hal_codec_i2s_iface_t::bits (C++ member), 271
audio_hal_codec_i2s_iface_t::fmt (C++ member), 271
audio_hal_codec_i2s_iface_t::mode (C++ member), 271
audio_hal_codec_i2s_iface_t::samples (C++ member), 271
audio_hal_codec_iface_config (C++ function), 270
AUDIO_HAL_CODEC_MODE_BOTH (C++ enumerator), 273
AUDIO_HAL_CODEC_MODE_DECODE (C++ enumerator), 273
AUDIO_HAL_CODEC_MODE_ENCODE (C++ enumerator), 273
AUDIO_HAL_CODEC_MODE_LINE_IN (C++ enumerator), 273
audio_hal_codec_mode_t (C++ enum), 273
audio_hal_ctrl_codec (C++ function), 269
AUDIO_HAL_CTRL_START (C++ enumerator), 274
AUDIO_HAL_CTRL_STOP (C++ enumerator), 274
audio_hal_ctrl_t (C++ enum), 274
AUDIO_HAL_DAC_OUTPUT_ALL (C++ enumerator), 274
AUDIO_HAL_DAC_OUTPUT_LINE1 (C++ enumerator), 274
AUDIO_HAL_DAC_OUTPUT_LINE2 (C++ enumerator), 274
audio_hal_dac_output_t (C++ enum), 274
audio_hal_deinit (C++ function), 269
audio_hal_enable_pa (C++ function), 271
audio_hal_func_t (C++ type), 273
audio_hal_get_volume (C++ function), 271
audio_hal_handle_t (C++ type), 273
AUDIO_HAL_I2S_DSP (C++ enumerator), 275
AUDIO_HAL_I2S_LEFT (C++ enumerator), 275
AUDIO_HAL_I2S_NORMAL (C++ enumerator), 275
AUDIO_HAL_I2S_RIGHT (C++ enumerator), 275
audio_hal_iface_bits_t (C++ enum), 275
audio_hal_iface_format_t (C++ enum), 275
audio_hal_iface_mode_t (C++ enum), 274
audio_hal_iface_samples_t (C++ enum), 274
audio_hal_init (C++ function), 269
AUDIO_HAL_MODE_MASTER (C++ enumerator), 274
AUDIO_HAL_MODE_SLAVE (C++ enumerator), 274
audio_hal_set_mute (C++ function), 270
audio_hal_set_volume (C++ function), 270
AUDIO_HAL_VOL_DEFAULT (C 宏), 273
audio_pipeline_breakup_elements (C++ function), 55
audio_pipeline_cfg (C++ class), 57
audio_pipeline_cfg::rb_size (C++ member), 57
audio_pipeline_cfg_t (C++ type), 57

audio_pipeline_change_state (C++ function), 56
audio_pipeline_check_items_state (C++ function), 54
audio_pipeline_deinit (C++ function), 46
audio_pipeline_get_el_by_tag (C++ function), 50
audio_pipeline_get_el_once (C++ function), 51
audio_pipeline_get_event_iface (C++ function), 52
audio_pipeline_handle_t (C++ type), 57
audio_pipeline_init (C++ function), 46
audio_pipeline_link (C++ function), 50
audio_pipeline_link_insert (C++ function), 52
audio_pipeline_link_more (C++ function), 53
audio_pipeline_listen_more (C++ function), 53
audio_pipeline_pause (C++ function), 49
audio_pipeline_register (C++ function), 47
audio_pipeline_register_more (C++ function), 52
audio_pipeline_relink (C++ function), 55
audio_pipeline_relink_more (C++ function), 56
audio_pipeline_remove_listener (C++ function), 51
audio_pipeline_reset_elements (C++ function), 54
audio_pipeline_reset_items_state (C++ function), 54
audio_pipeline_reset_kept_state (C++ function), 55
audio_pipeline_reset_ringbuffer (C++ function), 54
audio_pipeline_resume (C++ function), 48
audio_pipeline_run (C++ function), 47
audio_pipeline_set_listener (C++ function), 51
audio_pipeline_stop (C++ function), 49
audio_pipeline_terminate (C++ function), 48
audio_pipeline_terminate_with_ticks (C++ function), 48
audio_pipeline_unlink (C++ function), 50
audio_pipeline_unregister (C++ function), 47
audio_pipeline_unregister_more (C++ function), 53
audio_pipeline_wait_for_stop (C++ function), 49
audio_pipeline_wait_for_stop_with_ticks (C++ function), 49
audio_pwm_config_t (C++ class), 98
audio_pwm_config_t::data_len (C++ member), 98
audio_pwm_config_t::duty_resolution (C++ member), 98
audio_pwm_config_t::gpio_num_left (C++ member), 98
audio_pwm_config_t::gpio_num_right (C++ member), 98
audio_pwm_config_t::ledc_channel_left (C++ member), 98
audio_pwm_config_t::ledc_channel_right (C++ member), 98
audio_pwm_config_t::ledc_timer_sel (C++ member), 98
audio_pwm_config_t::tg_num (C++ member), 98
audio_pwm_config_t::timer_num (C++ member), 98
audio_rec_cfg_t (C++ class), 224
audio_rec_cfg_t::encoder_handle (C++ member), 224
audio_rec_cfg_t::encoder_iface (C++ member), 224
audio_rec_cfg_t::event_cb (C++ member), 224
audio_rec_cfg_t::pinned_core (C++ member), 224
audio_rec_cfg_t::read (C++ member), 224
audio_rec_cfg_t::sr_handle (C++ member), 224
audio_rec_cfg_t::sr_iface (C++ member),

224
audio_rec_cfg_t::task_prio (C++ member), 224
audio_rec_cfg_t::task_size (C++ member), 224
audio_rec_cfg_t::user_data (C++ member), 224
audio_rec_cfg_t::vad_off (C++ member), 224
audio_rec_cfg_t::vad_start (C++ member), 224
audio_rec_cfg_t::wakeup_end (C++ member), 224
audio_rec_cfg_t::wakeup_time (C++ member), 224
AUDIO_REC_DEF_TASK_CORE (C 宏), 225
AUDIO_REC_DEF_TASK_PRIO (C 宏), 225
AUDIO_REC_DEF_TASK_SZ (C 宏), 225
AUDIO_REC_DEF_VAD_OFF_TM (C 宏), 225
AUDIO_REC_DEF_WAKEEND_TM (C 宏), 225
AUDIO_REC_DEF_WAKEUP_TM (C 宏), 225
audio_rec_evt_t (C++ class), 223
audio_rec_evt_t::AUDIO_REC_COMMAND_DECT (C++ enumerator), 223
audio_rec_evt_t::AUDIO_REC_VAD_END (C++ enumerator), 223
audio_rec_evt_t::AUDIO_REC_VAD_START (C++ enumerator), 223
audio_rec_evt_t::AUDIO_REC_WAKEUP_END (C++ enumerator), 223
audio_rec_evt_t::AUDIO_REC_WAKEUP_START (C++ enumerator), 223
audio_rec_evt_t::data_len (C++ member), 223
audio_rec_evt_t::event_data (C++ member), 223
audio_rec_evt_t::type (C++ member), 223
audio_rec_evt_t::[anonymous] (C++ enum), 223
audio_rec_handle_t (C++ type), 225
AUDIO_REC_VAD_START_SPEECH_MS (C 宏), 225
audio_recorder_create (C++ function), 221
audio_recorder_data_read (C++ function), 222
AUDIO_RECORDER_DEFAULT_CFG (C 宏), 225
audio_recorder_destroy (C++ function), 222
audio_recorder_get_wakeup_state (C++ function), 222
audio_recorder_multinet_enable (C++ function), 222
audio_recorder_trigger_start (C++ function), 221
audio_recorder_trigger_stop (C++ function), 221
audio_recorder_vad_check_enable (C++ function), 222
audio_recorder_wakenet_enable (C++ function), 221
audio_service_callback (C++ function), 215
audio_service_config_t (C++ class), 217
audio_service_config_t::service_connect (C++ member), 218
audio_service_config_t::service_destroy (C++ member), 218
audio_service_config_t::service_disconnect (C++ member), 218
audio_service_config_t::service_name (C++ member), 218
audio_service_config_t::service_start (C++ member), 217
audio_service_config_t::service_stop (C++ member), 217
audio_service_config_t::task_core (C++ member), 217
audio_service_config_t::task_func (C++ member), 217
audio_service_config_t::task_prio (C++ member), 217
audio_service_config_t::task_stack (C++ member), 217
audio_service_config_t::user_data (C++ member), 218
audio_service_connect (C++ function), 216
audio_service_create (C++ function), 214
audio_service_destroy (C++ function), 214
audio_service_disconnect (C++ function), 216

audio_service_get_data (C++ function), 216
audio_service_handle_t (C++ type), 218
audio_service_set_callback (C++ function), 215
audio_service_set_data (C++ function), 216
audio_service_start (C++ function), 214
audio_service_stop (C++ function), 215
AUDIO_STATUS_ERROR (C++ enumerator), 66
AUDIO_STATUS_FINISHED (C++ enumerator), 66
AUDIO_STATUS_PAUSED (C++ enumerator), 66
AUDIO_STATUS_RUNNING (C++ enumerator), 66
AUDIO_STATUS_STOPPED (C++ enumerator), 66
AUDIO_STATUS_UNKNOWN (C++ enumerator), 66
AUDIO_STREAM_NONE (C++ enumerator), 64
AUDIO_STREAM_READER (C++ enumerator), 64
audio_stream_type_t (C++ enum), 64
AUDIO_STREAM_WRITER (C++ enumerator), 64
audio_termination_type_t (C++ enum), 66
audio_volume_get (C++ type), 65
audio_volume_set (C++ type), 65
automatic gain control, **413**
automatic level control, **413**
automatic speech recognition, **413**
aux cable, **413**
AVRCP, **413**
aw2013_led_bar_task (C++ function), 198

B

bandwidth, **413**
Bass Frequency, **413**
BAT_SERV_EVENT_BAT_FULL (C++ enumerator), 204
BAT_SERV_EVENT_BAT_LOW (C++ enumerator), 204
BAT_SERV_EVENT_CHARGING_BEGIN (C++ enumerator), 204
BAT_SERV_EVENT_CHARGING_STOP (C++ enumerator), 204
BAT_SERV_EVENT_UNKNOWN (C++ enumerator), 204
BAT_SERV_EVENT_VOL_REPORT (C++ enumerator), 204
battery_service_config_t (C++ class), 203
battery_service_config_t::cb_ctx (C++ member), 204
battery_service_config_t::charger_monitor (C++ member), 204
battery_service_config_t::evt_cb (C++ member), 204
battery_service_config_t::extern_stack (C++ member), 203
battery_service_config_t::task_core (C++ member), 203
battery_service_config_t::task_prio (C++ member), 203
battery_service_config_t::task_stack (C++ member), 203
battery_service_config_t::vol_monitor (C++ member), 204
battery_service_create (C++ function), 202
BATTERY_SERVICE_DEFAULT_CONFIG (C 宏), 204
battery_service_event_t (C++ enum), 204
battery_service_set_vol_report_freq (C++ function), 203
battery_service_vol_report_switch (C++ function), 203
BCLK, **413**
BLUE_LED_MAX_NUM (C 宏), 262
BLUETOOTH_A2DP_SINK (C++ enumerator), 171
BLUETOOTH_A2DP_SOURCE (C++ enumerator), 171
BLUETOOTH_ADDR_LEN (C 宏), 170
bluetooth_addr_t (C++ type), 170
bluetooth_service_cfg_t (C++ class), 170
bluetooth_service_cfg_t::device_name (C++ member), 170
bluetooth_service_cfg_t::mode (C++ member), 170
bluetooth_service_cfg_t::remote_name (C++ member), 170
bluetooth_service_cfg_t::user_callback (C++ member), 170
bluetooth_service_create_periph (C++ function), 166
bluetooth_service_create_stream (C++ function), 166

bluetooth_service_destroy (*C++ function*), 169
bluetooth_service_mode_t (*C++ enum*), 171
bluetooth_service_start (*C++ function*), 166
bluetooth_service_user_cb_t (*C++ class*), 169
bluetooth_service_user_cb_t::user_a2d_cb (*C++ member*), 170
bluetooth_service_user_cb_t::user_a2d_sink_data_cb (*C++ member*), 170
bluetooth_service_user_cb_t::user_a2d_source_data_cb (*C++ member*), 170
bluetooth_service_user_cb_t::user_avrc_ct_cb (*C++ member*), 170
BluFi, **413**
blufi_config_create (*C++ function*), 182
blufi_send_customized_data (*C++ function*), 182
blufi_set_customized_data (*C++ function*), 182
blufi_set_sta_connected_flag (*C++ function*), 182

C

cavity, **413**
command word, **413**
console_cmd_callback_t (*C++ type*), 251
CONSOLE_DEFAULT_BUFFER_SIZE (*C 宏*), 251
CONSOLE_DEFAULT_PROMPT_STRING (*C 宏*), 251
CONSOLE_DEFAULT_TASK_PRIO (*C 宏*), 251
CONSOLE_DEFAULT_TASK_STACK (*C 宏*), 251
core dump, **413**
coredump_need_upload (*C++ function*), 208
coredump_upload (*C++ function*), 208
coredump_upload_service_config_t (*C++ class*), 209
coredump_upload_service_config_t::cb_ctx (*C++ member*), 209
coredump_upload_service_config_t::do_post (*C++ member*), 209
coredump_upload_service_config_t::evt_cb (*C++ member*), 209
coredump_upload_service_config_t::task_core (*C++ member*), 209
coredump_upload_service_config_t::task_prio (*C++ member*), 209
coredump_upload_service_config_t::task_stack (*C++ member*), 209
coredump_upload_service_create (*C++ function*), 208
COREDUMP_UPLOAD_SERVICE_DEFAULT_CONFIG (*C 宏*), 209
current data, **413**
cutoff frequency, **413**

D

DAC, **414**
decoder, **414**
decoder_opus_init (*C++ function*), 148
DEFAULT_AAC_DECODER_CONFIG (*C 宏*), 137
DEFAULT_AMR_DECODER_CONFIG (*C 宏*), 139
DEFAULT_AMRNB_ENCODER_CONFIG (*C 宏*), 140
DEFAULT_AMRWB_ENCODER_CONFIG (*C 宏*), 142
DEFAULT_AUDIO_ELEMENT_CONFIG (*C 宏*), 43
DEFAULT_AUDIO_EVENT_IFACE_SIZE (*C 宏*), 63
DEFAULT_AUDIO_PIPELINE_CONFIG (*C 宏*), 57
DEFAULT_DOWNMIX_CONFIG (*C 宏*), 156
DEFAULT_ELEMENT_BUFFER_LENGTH (*C 宏*), 43
DEFAULT_ELEMENT_RINGBUF_SIZE (*C 宏*), 43
DEFAULT_ELEMENT_STACK_SIZE (*C 宏*), 43
DEFAULT_ELEMENT_TASK_CORE (*C 宏*), 43
DEFAULT_ELEMENT_TASK_PRIO (*C 宏*), 43
DEFAULT_EQUALIZER_CONFIG (*C 宏*), 159
DEFAULT_ESP_AUDIO_CONFIG (*C 宏*), 81
DEFAULT_ESP_PERIPH_SET_CONFIG (*C 宏*), 239
DEFAULT_ESP_PERIPH_STACK_SIZE (*C 宏*), 239
DEFAULT_ESP_PERIPH_TASK_CORE (*C 宏*), 239
DEFAULT_ESP_PERIPH_TASK_PRIO (*C 宏*), 239
DEFAULT_FLAC_DECODER_CONFIG (*C 宏*), 145
DEFAULT_MP3_DECODER_CONFIG (*C 宏*), 146
DEFAULT_OGG_DECODER_CONFIG (*C 宏*), 148
DEFAULT_OPUS_DECODER_CONFIG (*C 宏*), 149
DEFAULT_PIPELINE_RINGBUF_SIZE (*C 宏*), 57
DEFAULT_RECORDER_SR_CFG (*C 宏*), 227
DEFAULT_RESAMPLE_FILTER_CONFIG (*C 宏*), 162

DEFAULT_SONIC_CONFIG (*C 宏*), 165
DEFAULT_WAV_DECODER_CONFIG (*C 宏*), 151
DEFAULT_WAV_ENCODER_CONFIG (*C 宏*), 152
Digital Living Network Alliance, **414**
digital media renderer, **414**
digital signal processor, **414**
digital-to-analog converter, **414**
dispatcher, **414**
display_destroy (*C++ function*), 196
DISPLAY_PATTERN_BATTERY_CHARGING (*C++ enumerator*), 197
DISPLAY_PATTERN_BATTERY_FULL (*C++ enumerator*), 197
DISPLAY_PATTERN_BATTERY_LOW (*C++ enumerator*), 197
DISPLAY_PATTERN_BT_CONNECTED (*C++ enumerator*), 197
DISPLAY_PATTERN_BT_CONNECTTING (*C++ enumerator*), 197
DISPLAY_PATTERN_BT_DISCONNECTED (*C++ enumerator*), 197
DISPLAY_PATTERN_MAX (*C++ enumerator*), 198
DISPLAY_PATTERN_MUSIC_FINISHED (*C++ enumerator*), 197
DISPLAY_PATTERN_MUSIC_ON (*C++ enumerator*), 197
DISPLAY_PATTERN_MUTE_OFF (*C++ enumerator*), 197
DISPLAY_PATTERN_MUTE_ON (*C++ enumerator*), 197
DISPLAY_PATTERN_POWERON_INIT (*C++ enumerator*), 198
DISPLAY_PATTERN_RECOGNITION_START (*C++ enumerator*), 197
DISPLAY_PATTERN_RECOGNITION_STOP (*C++ enumerator*), 197
DISPLAY_PATTERN_RECORDING_START (*C++ enumerator*), 197
DISPLAY_PATTERN_RECORDING_STOP (*C++ enumerator*), 197
DISPLAY_PATTERN_SPEECH_BEGIN (*C++ enumerator*), 198
DISPLAY_PATTERN_SPEECH_OVER (*C++ enumerator*), 198
display_pattern_t (*C++ enum*), 197
DISPLAY_PATTERN_TURN_OFF (*C++ enumerator*), 197
DISPLAY_PATTERN_TURN_ON (*C++ enumerator*), 197
DISPLAY_PATTERN_UNKNOWN (*C++ enumerator*), 197
DISPLAY_PATTERN_VOLUME (*C++ enumerator*), 197
DISPLAY_PATTERN_WAKEUP_FINISHED (*C++ enumerator*), 197
DISPLAY_PATTERN_WAKEUP_ON (*C++ enumerator*), 197
DISPLAY_PATTERN_WIFI_CONNECTED (*C++ enumerator*), 197
DISPLAY_PATTERN_WIFI_CONNECTTING (*C++ enumerator*), 197
DISPLAY_PATTERN_WIFI_DISCONNECTED (*C++ enumerator*), 197
DISPLAY_PATTERN_WIFI_NO_CFG (*C++ enumerator*), 198
DISPLAY_PATTERN_WIFI_SETTING (*C++ enumerator*), 197
DISPLAY_PATTERN_WIFI_SETTING_FINISHED (*C++ enumerator*), 197
display_service_config_t (*C++ class*), 196
display_service_config_t::based_cfg (*C++ member*), 196
display_service_config_t::instance (*C++ member*), 196
display_service_create (*C++ function*), 196
display_service_handle_t (*C++ type*), 197
display_service_set_pattern (*C++ function*), 196
distortion, **414**
DLNA, **414**
DM_BUF_SIZE (*C 宏*), 156
DMR, **414**
downmix, **414**
downmix_cfg_t (*C++ class*), 156
downmix_cfg_t::downmix_info (*C++ member*),

156
downmix_cfg_t::max_sample (C++ member), 156
downmix_cfg_t::out_rb_size (C++ member), 156
downmix_cfg_t::stack_in_ext (C++ member), 156
downmix_cfg_t::task_core (C++ member), 156
downmix_cfg_t::task_prio (C++ member), 156
downmix_cfg_t::task_stack (C++ member), 156
downmix_init (C++ function), 155
DOWNMIX_RINGBUFFER_SIZE (C 宏), 156
downmix_set_gain_info (C++ function), 155
downmix_set_input_rb (C++ function), 154
downmix_set_input_rb_timeout (C++ function), 153
downmix_set_out_ctx_info (C++ function), 154
downmix_set_output_type (C++ function), 154
downmix_set_source_stream_info (C++ function), 154
downmix_set_transit_time_info (C++ function), 155
downmix_set_work_mode (C++ function), 154
DOWNMIX_TASK_CORE (C 宏), 156
DOWNMIX_TASK_PRIO (C 宏), 156
DOWNMIX_TASK_STACK (C 宏), 156
dram_list_choose (C++ function), 120
dram_list_create (C++ function), 118
dram_list_current (C++ function), 119
dram_list_destroy (C++ function), 121
dram_list_exist (C++ function), 119
dram_list_get_url_id (C++ function), 120
dram_list_get_url_num (C++ function), 120
dram_list_next (C++ function), 118
dram_list_prev (C++ function), 119
dram_list_remove_by_url (C++ function), 121
dram_list_remove_by_url_id (C++ function), 121
dram_list_reset (C++ function), 119
dram_list_save (C++ function), 118
dram_list_show (C++ function), 120
DSP, **414**
DuerOS, **414**
dueros_service_create (C++ function), 194
dueros_service_state_get (C++ function), 194
dueros_start_wifi_cfg (C++ function), 194
dueros_stop_wifi_cfg (C++ function), 195
dueros_voice_cancel (C++ function), 194
dueros_voice_upload (C++ function), 194
dueros_wifi_st_t (C++ class), 195
dueros_wifi_st_t::err (C++ member), 195
dueros_wifi_st_t::status (C++ member), 195
dueros_wifi_status_report (C++ function), 195

E

echo, **414**
echo reference signal, **414**
ECM, **414**
el_io_func (C++ type), 43
electret condenser microphone, **414**
element, **415**
ELEMENT_SUB_TYPE_OFFSET (C 宏), 63
EMBED_FLASH_STREAM_BUF_SIZE (C 宏), 109
EMBED_FLASH_STREAM_CFG_DEFAULT (C 宏), 109
embed_flash_stream_cfg_t (C++ class), 108
embed_flash_stream_cfg_t::buf_sz (C++ member), 109
embed_flash_stream_cfg_t::extern_stack (C++ member), 109
embed_flash_stream_cfg_t::out_rb_size (C++ member), 109
embed_flash_stream_cfg_t::task_core (C++ member), 109
embed_flash_stream_cfg_t::task_prio (C++ member), 109
embed_flash_stream_cfg_t::task_stack (C++ member), 109
EMBED_FLASH_STREAM_EXT_STACK (C 宏), 109
embed_flash_stream_init (C++ function), 108
EMBED_FLASH_STREAM_RINGBUFFER_SIZE (C 宏), 109

embed_flash_stream_set_context (*C++ function*), 108
EMBED_FLASH_STREAM_TASK_CORE (*C 宏*), 109
EMBED_FLASH_STREAM_TASK_PRIO (*C 宏*), 109
EMBED_FLASH_STREAM_TASK_STACK (*C 宏*), 109
embed_item_info (*C++ class*), 109
embed_item_info::address (*C++ member*), 109
embed_item_info::size (*C++ member*), 109
embed_item_info_t (*C++ type*), 110
encoder, **415**
equalizer, **415**
equalizer_cfg (*C++ class*), 158
equalizer_cfg::channel (*C++ member*), 158
equalizer_cfg::out_rb_size (*C++ member*), 158
equalizer_cfg::samplerate (*C++ member*), 158
equalizer_cfg::set_gain (*C++ member*), 158
equalizer_cfg::stack_in_ext (*C++ member*), 158
equalizer_cfg::task_core (*C++ member*), 158
equalizer_cfg::task_prio (*C++ member*), 158
equalizer_cfg::task_stack (*C++ member*), 158
equalizer_cfg_t (*C++ type*), 159
equalizer_init (*C++ function*), 158
EQUALIZER_RINGBUFFER_SIZE (*C 宏*), 159
equalizer_set_gain_info (*C++ function*), 157
equalizer_set_info (*C++ function*), 157
EQUALIZER_TASK_CORE (*C 宏*), 159
EQUALIZER_TASK_PRIO (*C 宏*), 159
EQUALIZER_TASK_STACK (*C 宏*), 159
ES8374_ADDR (*C 宏*), 288
es8374_codec_config_i2s (*C++ function*), 287
es8374_codec_ctrl_state (*C++ function*), 287
es8374_codec_deinit (*C++ function*), 283
es8374_codec_get_voice_volume (*C++ function*), 285
es8374_codec_init (*C++ function*), 283
es8374_codec_set_voice_volume (*C++ function*), 284
es8374_config_adc_input (*C++ function*), 286
es8374_config_dac_output (*C++ function*), 286
es8374_config_fmt (*C++ function*), 283
es8374_get_voice_mute (*C++ function*), 285
es8374_i2s_config_clock (*C++ function*), 283
es8374_pa_power (*C++ function*), 287
es8374_read_all (*C++ function*), 286
es8374_set_bits_per_sample (*C++ function*), 284
es8374_set_mic_gain (*C++ function*), 285
es8374_set_voice_mute (*C++ function*), 285
es8374_start (*C++ function*), 284
es8374_stop (*C++ function*), 284
es8374_write_reg (*C++ function*), 286
ES8388_ADCCONTROL1 (*C 宏*), 281
ES8388_ADCCONTROL10 (*C 宏*), 281
ES8388_ADCCONTROL11 (*C 宏*), 281
ES8388_ADCCONTROL12 (*C 宏*), 281
ES8388_ADCCONTROL13 (*C 宏*), 281
ES8388_ADCCONTROL14 (*C 宏*), 281
ES8388_ADCCONTROL2 (*C 宏*), 281
ES8388_ADCCONTROL3 (*C 宏*), 281
ES8388_ADCCONTROL4 (*C 宏*), 281
ES8388_ADCCONTROL5 (*C 宏*), 281
ES8388_ADCCONTROL6 (*C 宏*), 281
ES8388_ADCCONTROL7 (*C 宏*), 281
ES8388_ADCCONTROL8 (*C 宏*), 281
ES8388_ADCCONTROL9 (*C 宏*), 281
ES8388_ADCPOWER (*C 宏*), 281
ES8388_ADDR (*C 宏*), 281
ES8388_ANAVOLMANAG (*C 宏*), 281
ES8388_CHIPLOPOW1 (*C 宏*), 281
ES8388_CHIPLOPOW2 (*C 宏*), 281
ES8388_CHIPPOWER (*C 宏*), 281
es8388_config_adc_input (*C++ function*), 279
es8388_config_dac_output (*C++ function*), 279
es8388_config_fmt (*C++ function*), 276
es8388_config_i2s (*C++ function*), 280
ES8388_CONTROL1 (*C 宏*), 281
ES8388_CONTROL2 (*C 宏*), 281
es8388_ctrl_state (*C++ function*), 280
ES8388_DACCONTROL1 (*C 宏*), 281
ES8388_DACCONTROL10 (*C 宏*), 282

ES8388_DACCONTROL11 (*C 宏*), 282
ES8388_DACCONTROL12 (*C 宏*), 282
ES8388_DACCONTROL13 (*C 宏*), 282
ES8388_DACCONTROL14 (*C 宏*), 282
ES8388_DACCONTROL15 (*C 宏*), 282
ES8388_DACCONTROL16 (*C 宏*), 282
ES8388_DACCONTROL17 (*C 宏*), 282
ES8388_DACCONTROL18 (*C 宏*), 282
ES8388_DACCONTROL19 (*C 宏*), 282
ES8388_DACCONTROL2 (*C 宏*), 281
ES8388_DACCONTROL20 (*C 宏*), 282
ES8388_DACCONTROL21 (*C 宏*), 282
ES8388_DACCONTROL22 (*C 宏*), 282
ES8388_DACCONTROL23 (*C 宏*), 282
ES8388_DACCONTROL24 (*C 宏*), 282
ES8388_DACCONTROL25 (*C 宏*), 282
ES8388_DACCONTROL26 (*C 宏*), 282
ES8388_DACCONTROL27 (*C 宏*), 282
ES8388_DACCONTROL28 (*C 宏*), 282
ES8388_DACCONTROL29 (*C 宏*), 282
ES8388_DACCONTROL3 (*C 宏*), 281
ES8388_DACCONTROL30 (*C 宏*), 282
ES8388_DACCONTROL4 (*C 宏*), 281
ES8388_DACCONTROL5 (*C 宏*), 281
ES8388_DACCONTROL6 (*C 宏*), 282
ES8388_DACCONTROL7 (*C 宏*), 282
ES8388_DACCONTROL8 (*C 宏*), 282
ES8388_DACCONTROL9 (*C 宏*), 282
ES8388_DACPOWER (*C 宏*), 281
es8388_deinit (*C++ function*), 276
es8388_get_voice_mute (*C++ function*), 278
es8388_get_voice_volume (*C++ function*), 278
es8388_i2s_config_clock (*C++ function*), 276
es8388_init (*C++ function*), 276
ES8388_MASTERMODE (*C 宏*), 281
es8388_pa_power (*C++ function*), 280
es8388_read_all (*C++ function*), 279
es8388_set_bits_per_sample (*C++ function*), 277
es8388_set_mic_gain (*C++ function*), 278
es8388_set_voice_mute (*C++ function*), 278
es8388_set_voice_volume (*C++ function*), 277
es8388_start (*C++ function*), 277
es8388_stop (*C++ function*), 277
es8388_write_reg (*C++ function*), 279
ESP VoIP, **415**
ESP_A2DP_SAMPLE_RATE (*C 宏*), 170
esp_audio_callback_set (*C++ function*), 76
esp_audio_cfg_t (*C++ class*), 78
esp_audio_cfg_t::cb_ctx (*C++ member*), 78
esp_audio_cfg_t::cb_func (*C++ member*), 78
esp_audio_cfg_t::component_select (*C++ member*), 78
esp_audio_cfg_t::evt_que (*C++ member*), 78
esp_audio_cfg_t::in_stream_buf_size (*C++ member*), 78
esp_audio_cfg_t::out_stream_buf_size (*C++ member*), 78
esp_audio_cfg_t::prefer_type (*C++ member*), 78
esp_audio_cfg_t::resample_rate (*C++ member*), 78
esp_audio_cfg_t::task_prio (*C++ member*), 78
esp_audio_cfg_t::task_stack (*C++ member*), 79
esp_audio_cfg_t::vol_get (*C++ member*), 78
esp_audio_cfg_t::vol_handle (*C++ member*), 78
esp_audio_cfg_t::vol_set (*C++ member*), 78
esp_audio_codec_lib_add (*C++ function*), 68
esp_audio_codec_lib_query (*C++ function*), 68
ESP_AUDIO_COMPONENT_SELECT_ALC (*C 宏*), 81
ESP_AUDIO_COMPONENT_SELECT_DEFAULT (*C 宏*), 81
ESP_AUDIO_COMPONENT_SELECT_EQUALIZER (*C 宏*), 81
esp_audio_create (*C++ function*), 67
esp_audio_destroy (*C++ function*), 67
esp_audio_duration_get (*C++ function*), 76
esp_audio_eq_gain_get (*C++ function*), 72
esp_audio_eq_gain_set (*C++ function*), 72
esp_audio_event_callback (*C++ type*), 65
esp_audio_event_que_set (*C++ function*), 77

esp_audio_handle_t (C++ type), 81
esp_audio_info_get (C++ function), 76
esp_audio_info_set (C++ function), 76
esp_audio_info_t (C++ class), 79
esp_audio_info_t::audio_speed (C++ member), 80
esp_audio_info_t::codec_el (C++ member), 80
esp_audio_info_t::codec_info (C++ member), 80
esp_audio_info_t::filter_el (C++ member), 80
esp_audio_info_t::in_el (C++ member), 80
esp_audio_info_t::in_stream_total_size (C++ member), 80
esp_audio_info_t::out_el (C++ member), 80
esp_audio_info_t::st (C++ member), 80
esp_audio_info_t::time_pos (C++ member), 80
esp_audio_input_stream_add (C++ function), 68
esp_audio_media_type_set (C++ function), 76
esp_audio_music_info_get (C++ function), 76
esp_audio_music_info_t (C++ class), 80
esp_audio_music_info_t::bits (C++ member), 80
esp_audio_music_info_t::bps (C++ member), 80
esp_audio_music_info_t::channels (C++ member), 80
esp_audio_music_info_t::codec_fmt (C++ member), 80
esp_audio_music_info_t::sample_rates (C++ member), 80
esp_audio_output_stream_add (C++ function), 68
esp_audio_pause (C++ function), 71
esp_audio_play (C++ function), 69
ESP_AUDIO_PLAY_SPEED_0_50 (C++ enumerator), 81
ESP_AUDIO_PLAY_SPEED_0_75 (C++ enumerator), 81
ESP_AUDIO_PLAY_SPEED_1_00 (C++ enumerator), 81
ESP_AUDIO_PLAY_SPEED_1_25 (C++ enumerator), 81
ESP_AUDIO_PLAY_SPEED_1_50 (C++ enumerator), 81
ESP_AUDIO_PLAY_SPEED_1_75 (C++ enumerator), 81
ESP_AUDIO_PLAY_SPEED_2_00 (C++ enumerator), 81
ESP_AUDIO_PLAY_SPEED_MAX (C++ enumerator), 81
esp_audio_play_speed_t (C++ enum), 81
ESP_AUDIO_PLAY_SPEED_UNKNOW (C++ enumerator), 81
esp_audio_play_timeout_set (C++ function), 77
esp_audio_pos_get (C++ function), 75
ESP_AUDIO_PREFER_MEM (C++ enumerator), 66
ESP_AUDIO_PREFER_SPEED (C++ enumerator), 66
esp_audio_prefer_t (C++ enum), 66
esp_audio_prefer_type_get (C++ function), 77
esp_audio_resume (C++ function), 72
esp_audio_seek (C++ function), 76
esp_audio_setup (C++ function), 75
esp_audio_setup_t (C++ class), 79
esp_audio_setup_t::set_channel (C++ member), 79
esp_audio_setup_t::set_codec (C++ member), 79
esp_audio_setup_t::set_in_stream (C++ member), 79
esp_audio_setup_t::set_out_stream (C++ member), 79
esp_audio_setup_t::set_pos (C++ member), 79
esp_audio_setup_t::set_sample_rate (C++ member), 79
esp_audio_setup_t::set_time (C++ member), 79
esp_audio_setup_t::set_type (C++ member), 79

esp_audio_setup_t::set_uri (C++ member), 79
esp_audio_speed_get (C++ function), 73
esp_audio_speed_idx_to_float (C++ function), 73
esp_audio_speed_set (C++ function), 73
esp_audio_state_get (C++ function), 74
esp_audio_state_t (C++ class), 64
esp_audio_state_t::err_msg (C++ member), 64
esp_audio_state_t::media_src (C++ member), 64
esp_audio_state_t::status (C++ member), 64
esp_audio_status_t (C++ enum), 66
esp_audio_stop (C++ function), 71
esp_audio_sync_play (C++ function), 70
esp_audio_time_get (C++ function), 75
esp_audio_vol_get (C++ function), 74
esp_audio_vol_set (C++ function), 74
ESP_ERR_AUDIO_ALREADY_EXISTS (C++ enumerator), 65
ESP_ERR_AUDIO_BASE (C 宏), 65
ESP_ERR_AUDIO_CLOSE (C++ enumerator), 66
ESP_ERR_AUDIO_FAIL (C++ enumerator), 65
ESP_ERR_AUDIO_HAL_FAIL (C++ enumerator), 65
ESP_ERR_AUDIO_INPUT (C++ enumerator), 66
ESP_ERR_AUDIO_INVALID_PARAMETER (C++ enumerator), 65
ESP_ERR_AUDIO_INVALID_PATH (C++ enumerator), 65
ESP_ERR_AUDIO_INVALID_URI (C++ enumerator), 65
ESP_ERR_AUDIO_LINK_FAIL (C++ enumerator), 65
ESP_ERR_AUDIO_MEMORY_LACK (C++ enumerator), 65
ESP_ERR_AUDIO_NO_CODEC (C++ enumerator), 65
ESP_ERR_AUDIO_NO_ERROR (C++ enumerator), 65
ESP_ERR_AUDIO_NO_INPUT_STREAM (C++ enumerator), 65
ESP_ERR_AUDIO_NO_OUTPUT_STREAM (C++ enumerator), 65
ESP_ERR_AUDIO_NOT_READY (C++ enumerator), 65
ESP_ERR_AUDIO_NOT_SUPPORT (C++ enumerator), 65
ESP_ERR_AUDIO_OPEN (C++ enumerator), 65
ESP_ERR_AUDIO_OUT_OF_RANGE (C++ enumerator), 65
ESP_ERR_AUDIO_OUTPUT (C++ enumerator), 66
ESP_ERR_AUDIO_PROCESS (C++ enumerator), 66
ESP_ERR_AUDIO_STOP_BY_USER (C++ enumerator), 65
ESP_ERR_AUDIO_TIMEOUT (C++ enumerator), 65
ESP_ERR_AUDIO_UNKNOWN (C++ enumerator), 65
ESP_ERR_FS_OTA_BASE (C 宏), 193
ESP_ERR_FS_OTA_IN_PROGRESS (C 宏), 193
esp_fs_ota (C++ function), 190
esp_fs_ota_begin (C++ function), 191
esp_fs_ota_config_t (C++ class), 193
esp_fs_ota_config_t::buffer_size (C++ member), 193
esp_fs_ota_config_t::path (C++ member), 193
esp_fs_ota_finish (C++ function), 192
esp_fs_ota_get_image_len_read (C++ function), 193
esp_fs_ota_get_img_desc (C++ function), 192
esp_fs_ota_handle_t (C++ type), 193
esp_fs_ota_perform (C++ function), 191
esp_periph_config_t (C++ class), 238
esp_periph_config_t::extern_stack (C++ member), 238
esp_periph_config_t::task_core (C++ member), 238
esp_periph_config_t::task_prio (C++ member), 238
esp_periph_config_t::task_stack (C++ member), 238
esp_periph_create (C++ function), 233
esp_periph_destroy (C++ function), 237
esp_periph_event (C++ class), 238
esp_periph_event::cb (C++ member), 239
esp_periph_event::iface (C++ member), 239
esp_periph_event::user_ctx (C++ member), 239

esp_periph_event_handle_t (C++ type), 239
esp_periph_event_t (C++ type), 239
esp_periph_func (C++ type), 239
esp_periph_get_data (C++ function), 236
esp_periph_get_id (C++ function), 237
esp_periph_get_state (C++ function), 236
esp_periph_handle_t (C++ type), 239
esp_periph_id_t (C++ enum), 240
esp_periph_init (C++ function), 237
esp_periph_register_on_events (C++ function), 238
esp_periph_remove_from_set (C++ function), 232
esp_periph_run (C++ function), 237
esp_periph_run_func (C++ type), 239
esp_periph_send_cmd (C++ function), 234
esp_periph_send_cmd_from_isr (C++ function), 234
esp_periph_send_event (C++ function), 235
esp_periph_set_change_waiting_time (C++ function), 233
esp_periph_set_data (C++ function), 236
esp_periph_set_destroy (C++ function), 230
esp_periph_set_function (C++ function), 233
esp_periph_set_get_by_id (C++ function), 231
esp_periph_set_get_event_iface (C++ function), 231
esp_periph_set_get_queue (C++ function), 231
esp_periph_set_handle_t (C++ type), 239
esp_periph_set_id (C++ function), 237
esp_periph_set_init (C++ function), 230
esp_periph_set_list_destroy (C++ function), 232
esp_periph_set_list_init (C++ function), 231
esp_periph_set_list_run (C++ function), 232
esp_periph_set_register_callback (C++ function), 231
esp_periph_set_stop_all (C++ function), 230
esp_periph_start (C++ function), 234
esp_periph_start_timer (C++ function), 235
esp_periph_state_t (C++ enum), 240
esp_periph_stop (C++ function), 234
esp_periph_stop_timer (C++ function), 236
esp_touch_pad_sel_t (C++ enum), 252
esp_wifi_set_listen_interval (C++ function), 242
esp_wifi_setting_create (C++ function), 178
esp_wifi_setting_destroy (C++ function), 178
esp_wifi_setting_get_data (C++ function), 179
esp_wifi_setting_handle_t (C++ type), 180
esp_wifi_setting_info_notify (C++ function), 179
esp_wifi_setting_register_function (C++ function), 178
esp_wifi_setting_register_notify_handle (C++ function), 178
esp_wifi_setting_set_data (C++ function), 179
esp_wifi_setting_start (C++ function), 179
esp_wifi_setting_stop (C++ function), 180
esp_wifi_setting_teardown (C++ function), 180
event_cb_func (C++ type), 43

F

fast Fourier transform, **415**
FatFs, **415**
FatFs stream, **415**
FATFS_STREAM_BUF_SIZE (C 宏), 88
FATFS_STREAM_CFG_DEFAULT (C 宏), 88
fatfs_stream_cfg_t (C++ class), 87
fatfs_stream_cfg_t::buf_sz (C++ member), 87
fatfs_stream_cfg_t::ext_stack (C++ member), 87
fatfs_stream_cfg_t::out_rb_size (C++ member), 87
fatfs_stream_cfg_t::task_core (C++ member), 87
fatfs_stream_cfg_t::task_prio (C++ member), 87
fatfs_stream_cfg_t::task_stack (C++ member), 87

fatfs_stream_cfg_t::type (C++ member), 87
fatfs_stream_cfg_t::write_header (C++ member), 87
fatfs_stream_init (C++ function), 87
FATFS_STREAM_RINGBUFFER_SIZE (C 宏), 88
FATFS_STREAM_TASK_CORE (C 宏), 88
FATFS_STREAM_TASK_PRIO (C 宏), 88
FATFS_STREAM_TASK_STACK (C 宏), 88
FB, **415**
FEED_TASK_PINNED_CORE (C 宏), 227
FEED_TASK_PRIO (C 宏), 227
FEED_TASK_STACK_SZ (C 宏), 227
FETCH_TASK_PINNED_CORE (C 宏), 227
FETCH_TASK_PRIO (C 宏), 227
FETCH_TASK_STACK_SZ (C 宏), 227
FFT, **415**
FLAC, **415**
flac_decoder_cfg_t (C++ class), 144
flac_decoder_cfg_t::out_rb_size (C++ member), 144
flac_decoder_cfg_t::stack_in_ext (C++ member), 144
flac_decoder_cfg_t::task_core (C++ member), 144
flac_decoder_cfg_t::task_prio (C++ member), 144
flac_decoder_cfg_t::task_stack (C++ member), 144
flac_decoder_init (C++ function), 144
FLAC_DECODER_RINGBUFFER_SIZE (C 宏), 145
FLAC_DECODER_TASK_CORE (C 宏), 145
FLAC_DECODER_TASK_PRIO (C 宏), 145
FLAC_DECODER_TASK_STACK_SIZE (C 宏), 145
flash_list_choose (C++ function), 124
flash_list_create (C++ function), 122
flash_list_current (C++ function), 123
flash_list_destroy (C++ function), 125
flash_list_exist (C++ function), 123
flash_list_get_url_id (C++ function), 125
flash_list_get_url_num (C++ function), 124
flash_list_next (C++ function), 123
flash_list_prev (C++ function), 123
flash_list_reset (C++ function), 124
flash_list_save (C++ function), 122
flash_list_show (C++ function), 122
flexible pipeline, **415**
FPS, **415**
frames per second, **415**
frequency response, **415**
full band, **415**

G

get_input_key_service_state (C++ function), 172

H

HAL, **416**
Hands-Free, **415**
Hands-Free Audio Gateway, **416**
Hands-Free Profile, **416**
Hands-Free Unit, **416**
hardware abstraction layer, **416**
headset, **416**
HF, **415**
HFP, **416**
HFP-AG, **416**
Hi-Fi speaker, **416**
High Frequency, **416**
high-fidelity microphone, **416**
HLS, **416**
HTTP Live Streaming, **416**
HTTP stream, **416**
HTTP_STREAM_CFG_DEFAULT (C 宏), 92
http_stream_cfg_t (C++ class), 90
http_stream_cfg_t::auto_connect_next_track (C++ member), 91
http_stream_cfg_t::cert_pem (C++ member), 91
http_stream_cfg_t::crt_bundle_attach (C++ member), 91
http_stream_cfg_t::enable_playlist_parser (C++ member), 91
http_stream_cfg_t::event_handle (C++ member), 91

http_stream_cfg_t::multi_out_num (C++ member), 91
http_stream_cfg_t::out_rb_size (C++ member), 90
http_stream_cfg_t::request_range_size (C++ member), 91
http_stream_cfg_t::request_size (C++ member), 91
http_stream_cfg_t::stack_in_ext (C++ member), 91
http_stream_cfg_t::task_core (C++ member), 91
http_stream_cfg_t::task_prio (C++ member), 91
http_stream_cfg_t::task_stack (C++ member), 90
http_stream_cfg_t::type (C++ member), 90
http_stream_cfg_t::user_agent (C++ member), 91
http_stream_cfg_t::user_data (C++ member), 91
http_stream_event_handle_t (C++ type), 92
http_stream_event_id_t (C++ enum), 92
http_stream_event_msg_t (C++ class), 90
http_stream_event_msg_t::buffer (C++ member), 90
http_stream_event_msg_t::buffer_len (C++ member), 90
http_stream_event_msg_t::el (C++ member), 90
http_stream_event_msg_t::event_id (C++ member), 90
http_stream_event_msg_t::http_client (C++ member), 90
http_stream_event_msg_t::user_data (C++ member), 90
http_stream_fetch_again (C++ function), 89
HTTP_STREAM_FINISH_PLAYLIST (C++ enumerator), 92
HTTP_STREAM_FINISH_REQUEST (C++ enumerator), 92
HTTP_STREAM_FINISH_TRACK (C++ enumerator), 92
http_stream_init (C++ function), 89
http_stream_next_track (C++ function), 89
HTTP_STREAM_ON_REQUEST (C++ enumerator), 92
HTTP_STREAM_ON_RESPONSE (C++ enumerator), 92
HTTP_STREAM_POST_REQUEST (C++ enumerator), 92
HTTP_STREAM_PRE_REQUEST (C++ enumerator), 92
HTTP_STREAM_RESOLVE_ALL_TRACKS (C++ enumerator), 92
http_stream_restart (C++ function), 89
HTTP_STREAM_RINGBUFFER_SIZE (C 宏), 92
http_stream_set_server_cert (C++ function), 89
HTTP_STREAM_TASK_CORE (C 宏), 92
HTTP_STREAM_TASK_PRIO (C 宏), 92
HTTP_STREAM_TASK_STACK (C 宏), 92

I

I2S stream, **416**
i2s_alc_volume_get (C++ function), 94
i2s_alc_volume_set (C++ function), 94
I2S_CHANNEL_TYPE_ALL_LEFT (C++ enumerator), 96
I2S_CHANNEL_TYPE_ALL_RIGHT (C++ enumerator), 96
I2S_CHANNEL_TYPE_ONLY_LEFT (C++ enumerator), 96
I2S_CHANNEL_TYPE_ONLY_RIGHT (C++ enumerator), 96
I2S_CHANNEL_TYPE_RIGHT_LEFT (C++ enumerator), 96
i2s_channel_type_t (C++ enum), 96
I2S_STREAM_BUF_SIZE (C 宏), 96
I2S_STREAM_CFG_DEFAULT (C 宏), 96
I2S_STREAM_CFG_DEFAULT_WITH_PARA (C 宏), 96
i2s_stream_cfg_t (C++ class), 95
i2s_stream_cfg_t::buffer_len (C++ member), 96
i2s_stream_cfg_t::chan_cfg (C++ member), 95

i2s_stream_cfg_t::expand_src_bits (C++ member), 95
i2s_stream_cfg_t::multi_out_num (C++ member), 96
i2s_stream_cfg_t::need_expand (C++ member), 96
i2s_stream_cfg_t::out_rb_size (C++ member), 95
i2s_stream_cfg_t::stack_in_ext (C++ member), 95
i2s_stream_cfg_t::std_cfg (C++ member), 95
i2s_stream_cfg_t::task_core (C++ member), 95
i2s_stream_cfg_t::task_prio (C++ member), 95
i2s_stream_cfg_t::task_stack (C++ member), 95
i2s_stream_cfg_t::transmit_mode (C++ member), 95
i2s_stream_cfg_t::type (C++ member), 95
i2s_stream_cfg_t::uninstall_drv (C++ member), 96
i2s_stream_cfg_t::use_alc (C++ member), 95
i2s_stream_cfg_t::volume (C++ member), 95
i2s_stream_init (C++ function), 93
I2S_STREAM_RINGBUFFER_SIZE (C 宏), 96
i2s_stream_set_channel_type (C++ function), 93
i2s_stream_set_clk (C++ function), 94
i2s_stream_sync_delay (C++ function), 94
I2S_STREAM_TASK_CORE (C 宏), 96
I2S_STREAM_TASK_PRIO (C 宏), 96
I2S_STREAM_TASK_STACK (C 宏), 96
INPUT_KEY_SERVICE_ACTION_CLICK (C++ enumerator), 173
INPUT_KEY_SERVICE_ACTION_CLICK_RELEASE (C++ enumerator), 173
input_key_service_action_id_t (C++ enum), 173
INPUT_KEY_SERVICE_ACTION_PRESS (C++ enumerator), 173
INPUT_KEY_SERVICE_ACTION_PRESS_RELEASE (C++ enumerator), 174
INPUT_KEY_SERVICE_ACTION_UNKNOWN (C++ enumerator), 173
input_key_service_add_key (C++ function), 172
input_key_service_cfg_t (C++ class), 173
input_key_service_cfg_t::based_cfg (C++ member), 173
input_key_service_cfg_t::handle (C++ member), 173
input_key_service_create (C++ function), 172
INPUT_KEY_SERVICE_DEFAULT_CONFIG (C 宏), 173
input_key_service_info_t (C++ class), 172
input_key_service_info_t::act_id (C++ member), 173
input_key_service_info_t::type (C++ member), 173
input_key_service_info_t::user_id (C++ member), 173
INPUT_KEY_SERVICE_TASK_ON_CORE (C 宏), 173
INPUT_KEY_SERVICE_TASK_PRIORITY (C 宏), 173
INPUT_KEY_SERVICE_TASK_STACK_SIZE (C 宏), 173
INPUT_KEY_USER_ID_BATTERY_CHARGING (C++ enumerator), 174
INPUT_KEY_USER_ID_CAPTURE (C++ enumerator), 174
INPUT_KEY_USER_ID_COLOR (C++ enumerator), 175
INPUT_KEY_USER_ID_MAX (C++ enumerator), 175
INPUT_KEY_USER_ID_MODE (C++ enumerator), 174
INPUT_KEY_USER_ID_MSG (C++ enumerator), 174
INPUT_KEY_USER_ID_MUTE (C++ enumerator), 174
INPUT_KEY_USER_ID_PLAY (C++ enumerator), 174
INPUT_KEY_USER_ID_REC (C++ enumerator), 174
INPUT_KEY_USER_ID_SET (C++ enumerator), 174
input_key_user_id_t (C++ enum), 174
INPUT_KEY_USER_ID_UNKNOWN (C++ enumerator), 174
INPUT_KEY_USER_ID_VOLDOWN (C++ enumerator),

174
INPUT_KEY_USER_ID_VOLUP (*C++ enumerator*), 174
INPUT_KEY_USER_ID_WAKEUP (*C++ enumerator*), 174
INPUT_ORDER_DEFAULT (*C 宏*), 227
insertion loss, **416**
Internet of Things, **416**
Internet radio, **416**
IoT, **416**
IS31FL3216_CH_NUM (*C 宏*), 262
IS31FL3216_STATE_BY_AUDIO (*C++ enumerator*), 263
IS31FL3216_STATE_FLASH (*C++ enumerator*), 263
IS31FL3216_STATE_OFF (*C++ enumerator*), 263
IS31FL3216_STATE_ON (*C++ enumerator*), 263
IS31FL3216_STATE_SHIFT (*C++ enumerator*), 263
IS31FL3216_STATE_UNKNOWN (*C++ enumerator*), 263

J

JPEG, **416**
JPG, **417**

L

led_bar_aw2013_deinit (*C++ function*), 199
led_bar_aw2013_init (*C++ function*), 198
led_bar_aw2013_pattern (*C++ function*), 198
led_bar_aw2013_set_blink_time (*C++ function*), 198
led_bar_is31x_deinit (*C++ function*), 199
led_bar_is31x_init (*C++ function*), 199
led_bar_is31x_pattern (*C++ function*), 199
led_bar_ws2812_deinit (*C++ function*), 200
led_bar_ws2812_handle_t (*C++ type*), 201
led_bar_ws2812_init (*C++ function*), 200
led_bar_ws2812_pattern (*C++ function*), 200
led_indicator_deinit (*C++ function*), 202
led_indicator_handle_t (*C++ type*), 202
led_indicator_init (*C++ function*), 201
led_indicator_pattern (*C++ function*), 201
Light and Versatile Graphics Library, **417**
low-pass filter, **417**
LVGL, **417**

M

M3U8, **417**
M4A, **417**
mass production, **417**
maximum output power, **417**
MCLK, **417**
media_source_type_t (*C++ enum*), 66
MEDIA_SRC_TYPE_MUSIC_A2DP (*C++ enumerator*), 66
MEDIA_SRC_TYPE_MUSIC_BASE (*C++ enumerator*), 66
MEDIA_SRC_TYPE_MUSIC_DLNA (*C++ enumerator*), 66
MEDIA_SRC_TYPE_MUSIC_FLASH (*C++ enumerator*), 66
MEDIA_SRC_TYPE_MUSIC_HTTP (*C++ enumerator*), 66
MEDIA_SRC_TYPE_MUSIC_MAX (*C++ enumerator*), 67
MEDIA_SRC_TYPE_MUSIC_RAW (*C++ enumerator*), 67
MEDIA_SRC_TYPE_MUSIC_SD (*C++ enumerator*), 66
MEDIA_SRC_TYPE_NULL (*C++ enumerator*), 66
MEDIA_SRC_TYPE_RESERVE_BASE (*C++ enumerator*), 67
MEDIA_SRC_TYPE_RESERVE_MAX (*C++ enumerator*), 67
MEDIA_SRC_TYPE_TONE_BASE (*C++ enumerator*), 67
MEDIA_SRC_TYPE_TONE_FLASH (*C++ enumerator*), 67
MEDIA_SRC_TYPE_TONE_HTTP (*C++ enumerator*), 67
MEDIA_SRC_TYPE_TONE_MAX (*C++ enumerator*), 67
MEDIA_SRC_TYPE_TONE_SD (*C++ enumerator*), 67
mel-frequency cepstral coefficients, **417**
mem_assert (*C 宏*), 63

MEMS mic, 417
MFCC, 417
mic, 417
micro-electro-mechanical systems microphone, 417
microphone, 417
microphone gain, 417
microphone hole, 417
microSD card, 417
MP3, 417
mp3_decoder_cfg_t (C++ class), 146
mp3_decoder_cfg_t::id3_parse_enable (C++ member), 146
mp3_decoder_cfg_t::out_rb_size (C++ member), 146
mp3_decoder_cfg_t::stack_in_ext (C++ member), 146
mp3_decoder_cfg_t::task_core (C++ member), 146
mp3_decoder_cfg_t::task_prio (C++ member), 146
mp3_decoder_cfg_t::task_stack (C++ member), 146
mp3_decoder_get_id3_info (C++ function), 145
mp3_decoder_init (C++ function), 145
MP3_DECODER_RINGBUFFER_SIZE (C 宏), 146
MP3_DECODER_TASK_CORE (C 宏), 146
MP3_DECODER_TASK_PRIO (C 宏), 146
MP3_DECODER_TASK_STACK_SIZE (C 宏), 146
MP4, 418
multi-room, 418
Multi-Room Music, 418
MultiNet, 418

N

narrowband, 418
NB, 418
NimBLE, 418
noise criteria curve, 418
noise floor, 418
noise rating curve, 418
noise suppression, 418
non-volatile storage, 418
NS, 418
NVS, 418

O

OGG, 418
ogg_decoder_cfg_t (C++ class), 147
ogg_decoder_cfg_t::out_rb_size (C++ member), 147
ogg_decoder_cfg_t::stack_in_ext (C++ member), 147
ogg_decoder_cfg_t::task_core (C++ member), 147
ogg_decoder_cfg_t::task_prio (C++ member), 147
ogg_decoder_cfg_t::task_stack (C++ member), 147
ogg_decoder_init (C++ function), 147
OGG_DECODER_RINGBUFFER_SIZE (C 宏), 148
OGG_DECODER_TASK_CORE (C 宏), 148
OGG_DECODER_TASK_PRIO (C 宏), 148
OGG_DECODER_TASK_STACK_SIZE (C 宏), 148
on_event_iface_func (C++ type), 63
OPUS, 419
opus_decoder_cfg_t (C++ class), 148
opus_decoder_cfg_t::out_rb_size (C++ member), 149
opus_decoder_cfg_t::stack_in_ext (C++ member), 149
opus_decoder_cfg_t::task_core (C++ member), 149
opus_decoder_cfg_t::task_prio (C++ member), 149
opus_decoder_cfg_t::task_stack (C++ member), 149
OPUS_DECODER_RINGBUFFER_SIZE (C 宏), 149
OPUS_DECODER_TASK_CORE (C 宏), 149
OPUS_DECODER_TASK_PRIO (C 宏), 149
OPUS_DECODER_TASK_STACK_SIZE (C 宏), 149
ota_app_get_default_proc (C++ function), 189
ota_data_get_default_proc (C++ function), 189

ota_data_image_stream_read (C++ function), 189
ota_data_partition_erase_mark (C++ function), 190
ota_data_partition_write (C++ function), 189
ota_get_version_number (C++ function), 190
ota_node_attr_t (C++ class), 186
ota_node_attr_t::cert_pem (C++ member), 186
ota_node_attr_t::label (C++ member), 186
ota_node_attr_t::type (C++ member), 186
ota_node_attr_t::uri (C++ member), 186
ota_result_t (C++ class), 187
ota_result_t::id (C++ member), 187
ota_result_t::result (C++ member), 187
OTA_SERV_ERR_REASON_ERROR_MAGIC_WORD (C++ enumerator), 188
OTA_SERV_ERR_REASON_ERROR_PROJECT_NAME (C++ enumerator), 188
OTA_SERV_ERR_REASON_ERROR_VERSION (C++ enumerator), 188
OTA_SERV_ERR_REASON_FILE_NOT_FOUND (C++ enumerator), 188
OTA_SERV_ERR_REASON_GET_NEW_APP_DESC_FAIL (C++ enumerator), 188
OTA_SERV_ERR_REASON_NO_HIGHER_VERSION (C++ enumerator), 188
OTA_SERV_ERR_REASON_NULL_POINTER (C++ enumerator), 188
OTA_SERV_ERR_REASON_PARTITION_NOT_FOUND (C++ enumerator), 188
OTA_SERV_ERR_REASON_PARTITION_RD_FAIL (C++ enumerator), 188
OTA_SERV_ERR_REASON_PARTITION_WT_FAIL (C++ enumerator), 188
OTA_SERV_ERR_REASON_STREAM_INIT_FAIL (C++ enumerator), 188
OTA_SERV_ERR_REASON_STREAM_RD_FAIL (C++ enumerator), 188
OTA_SERV_ERR_REASON_SUCCESS (C++ enumerator), 188
OTA_SERV_ERR_REASON_UNKNOWN (C++ enumerator), 188
OTA_SERV_ERR_REASON_URL_PARSE_FAIL (C++ enumerator), 188
OTA_SERV_EVENT_TYPE_FINISH (C++ enumerator), 188
OTA_SERV_EVENT_TYPE_RESULT (C++ enumerator), 188
ota_service_config_t (C++ class), 186
ota_service_config_t::cb_ctx (C++ member), 186
ota_service_config_t::evt_cb (C++ member), 186
ota_service_config_t::task_core (C++ member), 186
ota_service_config_t::task_prio (C++ member), 186
ota_service_config_t::task_stack (C++ member), 186
ota_service_create (C++ function), 185
OTA_SERVICE_DEFAULT_CONFIG (C 宏), 187
OTA_SERVICE_ERR_REASON_BASE (C 宏), 187
ota_service_err_reason_t (C++ enum), 188
ota_service_event_type_t (C++ enum), 188
ota_service_set_upgrade_param (C++ function), 185
ota_upgrade_ops_t (C++ class), 186
ota_upgrade_ops_t::break_after_fail (C++ member), 187
ota_upgrade_ops_t::execute_upgrade (C++ member), 187
ota_upgrade_ops_t::finished_check (C++ member), 187
ota_upgrade_ops_t::need_upgrade (C++ member), 187
ota_upgrade_ops_t::node (C++ member), 187
ota_upgrade_ops_t::prepare (C++ member), 187
ota_upgrade_ops_t::reboot_flag (C++ member), 187

P

partition_list_choose (C++ function), 127

partition_list_create (C++ function), 126
partition_list_current (C++ function), 127
partition_list_destroy (C++ function), 129
partition_list_exist (C++ function), 127
partition_list_get_url_id (C++ function), 128
partition_list_get_url_num (C++ function), 128
partition_list_next (C++ function), 126
partition_list_prev (C++ function), 126
partition_list_reset (C++ function), 127
partition_list_save (C++ function), 126
partition_list_show (C++ function), 128
PCM, **419**
periph_adc_button_cfg_t (C++ class), 258
periph_adc_button_cfg_t::arr (C++ member), 258
periph_adc_button_cfg_t::arr_size (C++ member), 258
periph_adc_button_cfg_t::task_cfg (C++ member), 258
PERIPH_ADC_BUTTON_DEFAULT_CONFIG (C 宏), 258
periph_adc_button_event_id_t (C++ enum), 259
PERIPH_ADC_BUTTON_IDLE (C++ enumerator), 259
periph_adc_button_init (C++ function), 258
PERIPH_ADC_BUTTON_LONG_PRESSED (C++ enumerator), 259
PERIPH_ADC_BUTTON_LONG_RELEASE (C++ enumerator), 259
PERIPH_ADC_BUTTON_PRESSED (C++ enumerator), 259
PERIPH_ADC_BUTTON_RELEASE (C++ enumerator), 259
periph_bluetooth_cancel_discover (C++ function), 168
periph_bluetooth_connect (C++ function), 169
periph_bluetooth_discover (C++ function), 168
periph_bluetooth_fast_forward (C++ function), 168
periph_bluetooth_get_a2dp_sample_rate (C++ function), 169
periph_bluetooth_next (C++ function), 167
periph_bluetooth_pause (C++ function), 167
periph_bluetooth_play (C++ function), 166
periph_bluetooth_prev (C++ function), 167
periph_bluetooth_rewind (C++ function), 167
periph_bluetooth_stop (C++ function), 167
periph_button_cfg_t (C++ class), 254
periph_button_cfg_t::gpio_mask (C++ member), 254
periph_button_cfg_t::long_press_time_ms (C++ member), 254
periph_button_event_id_t (C++ enum), 254
periph_button_init (C++ function), 253
PERIPH_BUTTON_LONG_PRESSED (C++ enumerator), 254
PERIPH_BUTTON_LONG_RELEASE (C++ enumerator), 254
PERIPH_BUTTON_PRESSED (C++ enumerator), 254
PERIPH_BUTTON_RELEASE (C++ enumerator), 254
PERIPH_BUTTON_UNCHANGE (C++ enumerator), 254
periph_console_cfg_t (C++ class), 250
periph_console_cfg_t::buffer_size (C++ member), 250
periph_console_cfg_t::command_num (C++ member), 250
periph_console_cfg_t::commands (C++ member), 250
periph_console_cfg_t::prompt_string (C++ member), 251
periph_console_cfg_t::task_prio (C++ member), 250
periph_console_cfg_t::task_stack (C++ member), 250
periph_console_cmd_t (C++ class), 250
periph_console_cmd_t::cmd (C++ member), 250
periph_console_cmd_t::func (C++ member), 250
periph_console_cmd_t::help (C++ member), 250

periph_console_cmd_t::id (C++ member), 250
periph_console_init (C++ function), 250
PERIPH_ID_ADC (C++ enumerator), 240
PERIPH_ID_ADC_BTN (C++ enumerator), 240
PERIPH_ID_AUXIN (C++ enumerator), 240
PERIPH_ID_AW2013 (C++ enumerator), 240
PERIPH_ID_BLUETOOTH (C++ enumerator), 240
PERIPH_ID_BUTTON (C++ enumerator), 240
PERIPH_ID_CONSOLE (C++ enumerator), 240
PERIPH_ID_FLASH (C++ enumerator), 240
PERIPH_ID_GPIO_ISR (C++ enumerator), 240
PERIPH_ID_IS31FL3216 (C++ enumerator), 240
PERIPH_ID_LCD (C++ enumerator), 240
PERIPH_ID_LED (C++ enumerator), 240
PERIPH_ID_SDCARD (C++ enumerator), 240
PERIPH_ID_SPIFFS (C++ enumerator), 240
PERIPH_ID_TOUCH (C++ enumerator), 240
PERIPH_ID_WIFI (C++ enumerator), 240
PERIPH_ID_WS2812 (C++ enumerator), 240
PERIPH_IS31_SHIFT_MODE_ACC (C++ enumerator), 263
PERIPH_IS31_SHIFT_MODE_SINGLE (C++ enumerator), 263
periph_is31_shift_mode_t (C++ enum), 263
PERIPH_IS31_SHIFT_MODE_UNKNOWN (C++ enumerator), 263
periph_is31fl3216_cfg_t (C++ class), 262
periph_is31fl3216_cfg_t::duty (C++ member), 262
periph_is31fl3216_cfg_t::is31fl3216_pattern (C++ member), 262
periph_is31fl3216_cfg_t::state (C++ member), 262
periph_is31fl3216_init (C++ function), 259
periph_is31fl3216_set_act_time (C++ function), 262
periph_is31fl3216_set_blink_pattern (C++ function), 260
periph_is31fl3216_set_duty (C++ function), 260
periph_is31fl3216_set_duty_step (C++ function), 260
periph_is31fl3216_set_interval (C++ function), 261
periph_is31fl3216_set_light_on_num (C++ function), 261
periph_is31fl3216_set_shift_mode (C++ function), 261
periph_is31fl3216_set_state (C++ function), 260
periph_is31fl3216_state_t (C++ enum), 263
periph_led_blink (C++ function), 255
PERIPH_LED_BLINK_FINISH (C++ enumerator), 257
periph_led_cfg_t (C++ class), 256
periph_led_cfg_t::gpio_num (C++ member), 256
periph_led_cfg_t::led_duty_resolution (C++ member), 256
periph_led_cfg_t::led_freq_hz (C++ member), 256
periph_led_cfg_t::led_speed_mode (C++ member), 256
periph_led_cfg_t::led_timer_num (C++ member), 256
periph_led_event_id_t (C++ enum), 257
PERIPH_LED_IDLE_LEVEL_HIGH (C++ enumerator), 257
PERIPH_LED_IDLE_LEVEL_LOW (C++ enumerator), 257
periph_led_idle_level_t (C++ enum), 257
periph_led_init (C++ function), 255
periph_led_stop (C++ function), 256
PERIPH_LED_UNCHANGE (C++ enumerator), 257
periph_sdcard_cfg_t (C++ class), 246
periph_sdcard_cfg_t::card_detect_pin (C++ member), 246
periph_sdcard_cfg_t::mode (C++ member), 246
periph_sdcard_cfg_t::root (C++ member), 246
periph_sdcard_event_id_t (C++ enum), 246
periph_sdcard_init (C++ function), 245
periph_sdcard_is_mounted (C++ function), 245

periph_sdcard_mode_t (C++ enum), 247
periph_service_callback (C++ function), 211
periph_service_cb (C++ type), 213
periph_service_config_t (C++ class), 212
periph_service_config_t::extern_stack (C++ member), 213
periph_service_config_t::service_destroy (C++ member), 213
periph_service_config_t::service_ioctl (C++ member), 213
periph_service_config_t::service_name (C++ member), 213
periph_service_config_t::service_start (C++ member), 213
periph_service_config_t::service_stop (C++ member), 213
periph_service_config_t::task_core (C++ member), 213
periph_service_config_t::task_func (C++ member), 213
periph_service_config_t::task_prio (C++ member), 213
periph_service_config_t::task_stack (C++ member), 213
periph_service_config_t::user_data (C++ member), 213
periph_service_create (C++ function), 209
periph_service_ctrl (C++ type), 213
periph_service_destroy (C++ function), 210
periph_service_event_t (C++ class), 212
periph_service_event_t::data (C++ member), 212
periph_service_event_t::len (C++ member), 212
periph_service_event_t::source (C++ member), 212
periph_service_event_t::type (C++ member), 212
periph_service_get_data (C++ function), 211
periph_service_handle_t (C++ type), 213
periph_service_io (C++ type), 213
periph_service_ioctl (C++ function), 211
periph_service_set_callback (C++ function), 210
periph_service_set_data (C++ function), 211
periph_service_start (C++ function), 210
PERIPH_SERVICE_STATE_IDLE (C++ enumerator), 214
PERIPH_SERVICE_STATE_RUNNING (C++ enumerator), 214
PERIPH_SERVICE_STATE_STOPPED (C++ enumerator), 214
periph_service_state_t (C++ enum), 214
PERIPH_SERVICE_STATE_UNKNOWN (C++ enumerator), 214
periph_service_stop (C++ function), 210
periph_spiffs_cfg_t (C++ class), 248
periph_spiffs_cfg_t::format_if_mount_failed (C++ member), 248
periph_spiffs_cfg_t::max_files (C++ member), 248
periph_spiffs_cfg_t::partition_label (C++ member), 248
periph_spiffs_cfg_t::root (C++ member), 248
periph_spiffs_event_id_t (C++ enum), 249
periph_spiffs_init (C++ function), 248
periph_spiffs_is_mounted (C++ function), 248
PERIPH_STATE_ERROR (C++ enumerator), 240
PERIPH_STATE_INIT (C++ enumerator), 240
PERIPH_STATE_NULL (C++ enumerator), 240
PERIPH_STATE_PAUSE (C++ enumerator), 240
PERIPH_STATE_RUNNING (C++ enumerator), 240
PERIPH_STATE_STATUS_MAX (C++ enumerator), 240
PERIPH_STATE_STOPPING (C++ enumerator), 240
periph_tick_get (C 宏), 239
periph_touch_cfg_t (C++ class), 252
periph_touch_cfg_t::long_tap_time_ms (C++ member), 252
periph_touch_cfg_t::tap_threshold_percent (C++ member), 252
periph_touch_cfg_t::touch_mask (C++ member), 252

periph_touch_event_id_t (C++ enum), 252
periph_touch_init (C++ function), 251
PERIPH_TOUCH_LONG_RELEASE (C++ enumerator), 253
PERIPH_TOUCH_LONG_TAP (C++ enumerator), 253
PERIPH_TOUCH_RELEASE (C++ enumerator), 253
PERIPH_TOUCH_TAP (C++ enumerator), 253
PERIPH_TOUCH_UNCHANGE (C++ enumerator), 252
periph_wifi_cfg_t (C++ class), 243
periph_wifi_cfg_t::disable_auto_reconnect (C++ member), 244
periph_wifi_cfg_t::reconnect_timeout_ms (C++ member), 244
periph_wifi_cfg_t::wifi_config (C++ member), 244
periph_wifi_cfg_t::wpa2_e_cfg (C++ member), 244
PERIPH_WIFI_CONFIG_DONE (C++ enumerator), 244
PERIPH_WIFI_CONFIG_ERROR (C++ enumerator), 244
periph_wifi_config_mode_t (C++ enum), 244
periph_wifi_config_start (C++ function), 242
periph_wifi_config_wait_done (C++ function), 242
PERIPH_WIFI_CONNECTED (C++ enumerator), 244
PERIPH_WIFI_CONNECTING (C++ enumerator), 244
PERIPH_WIFI_DISCONNECTED (C++ enumerator), 244
PERIPH_WIFI_ERROR (C++ enumerator), 244
periph_wifi_init (C++ function), 241
periph_wifi_is_connected (C++ function), 242
PERIPH_WIFI_SETTING (C++ enumerator), 244
periph_wifi_state_t (C++ enum), 244
PERIPH_WIFI_UNCHANGE (C++ enumerator), 244
periph_wifi_wait_for_connected (C++ function), 241
periph_wpa2_enterprise_cfg_t (C++ class), 243
periph_wpa2_enterprise_cfg_t::ca_pem_end (C++ member), 243
periph_wpa2_enterprise_cfg_t::ca_pem_start (C++ member), 243
periph_wpa2_enterprise_cfg_t::diasble_wpa2_e (C++ member), 243
periph_wpa2_enterprise_cfg_t::eap_id (C++ member), 243
periph_wpa2_enterprise_cfg_t::eap_method (C++ member), 243
periph_wpa2_enterprise_cfg_t::eap_password (C++ member), 243
periph_wpa2_enterprise_cfg_t::eap_username (C++ member), 243
periph_wpa2_enterprise_cfg_t::wpa2_e_cert_end (C++ member), 243
periph_wpa2_enterprise_cfg_t::wpa2_e_cert_start (C++ member), 243
periph_wpa2_enterprise_cfg_t::wpa2_e_key_end (C++ member), 243
periph_wpa2_enterprise_cfg_t::wpa2_e_key_start (C++ member), 243
PGA, **419**
pixel, **419**
playback, **419**
playlist_add (C++ function), 129
playlist_checkout_by_id (C++ function), 130
playlist_choose (C++ function), 132
playlist_create (C++ function), 129
playlist_destroy (C++ function), 134
PLAYLIST_DRAM (C++ enumerator), 136
playlist_exist (C++ function), 133
PLAYLIST_FLASH (C++ enumerator), 136
playlist_get_current_list_id (C++ function), 130
playlist_get_current_list_type (C++ function), 130
playlist_get_current_list_url (C++ function), 131
playlist_get_current_list_url_id (C++ function), 131
playlist_get_current_list_url_num (C++ function), 131
playlist_get_list_num (C++ function), 130
playlist_handle_t (C++ type), 135

playlist_next (C++ function), 131
playlist_operation_t (C++ class), 134
playlist_operation_t::choose (C++ member), 134
playlist_operation_t::current (C++ member), 135
playlist_operation_t::destroy (C++ member), 135
playlist_operation_t::exist (C++ member), 135
playlist_operation_t::get_url_id (C++ member), 135
playlist_operation_t::get_url_num (C++ member), 135
playlist_operation_t::next (C++ member), 134
playlist_operation_t::prev (C++ member), 134
playlist_operation_t::remove_by_id (C++ member), 135
playlist_operation_t::remove_by_url (C++ member), 135
playlist_operation_t::reset (C++ member), 134
playlist_operation_t::save (C++ member), 134
playlist_operation_t::show (C++ member), 134
playlist_operation_t::type (C++ member), 135
playlist_operator_handle_t (C++ type), 135
playlist_operator_t (C++ class), 135
playlist_operator_t::get_operation (C++ member), 135
playlist_operator_t::playlist (C++ member), 135
PLAYLIST_PARTITION (C++ enumerator), 136
playlist_prev (C++ function), 132
playlist_remove_by_url (C++ function), 133
playlist_remove_by_url_id (C++ function), 133
playlist_reset (C++ function), 133
playlist_save (C++ function), 131
PLAYLIST_SDCARD (C++ enumerator), 136
playlist_show (C++ function), 132
playlist_type_t (C++ enum), 136
PLAYLIST_UNKNOWN (C++ enumerator), 136
process_func (C++ type), 43
programmable gain amplifier, **419**
protractor, **419**
pulse-code modulation, **419**
PWM_CONFIG_RINGBUFFER_SIZE (C 宏), 99
PWM_STREAM_BUF_SIZE (C 宏), 99
PWM_STREAM_CFG_DEFAULT (C 宏), 99
pwm_stream_cfg_t (C++ class), 98
pwm_stream_cfg_t::buffer_len (C++ member), 99
pwm_stream_cfg_t::ext_stack (C++ member), 99
pwm_stream_cfg_t::out_rb_size (C++ member), 98
pwm_stream_cfg_t::pwm_config (C++ member), 98
pwm_stream_cfg_t::task_core (C++ member), 99
pwm_stream_cfg_t::task_prio (C++ member), 99
pwm_stream_cfg_t::task_stack (C++ member), 98
pwm_stream_cfg_t::type (C++ member), 98
PWM_STREAM_GPIO_NUM_LEFT (C 宏), 99
PWM_STREAM_GPIO_NUM_RIGHT (C 宏), 99
pwm_stream_init (C++ function), 97
PWM_STREAM_RINGBUFFER_SIZE (C 宏), 99
pwm_stream_set_clk (C++ function), 97
PWM_STREAM_TASK_CORE (C 宏), 99
PWM_STREAM_TASK_PRIO (C 宏), 99
PWM_STREAM_TASK_STACK (C 宏), 99

R

raw stream, **419**
RAW_STREAM_CFG_DEFAULT (C 宏), 101
raw_stream_cfg_t (C++ class), 101

raw_stream_cfg_t::out_rb_size (C++ member), 101
raw_stream_cfg_t::type (C++ member), 101
raw_stream_init (C++ function), 100
raw_stream_read (C++ function), 100
RAW_STREAM_RINGBUFFER_SIZE (C 宏), 101
raw_stream_write (C++ function), 100
RB_ABORT (C 宏), 268
rb_abort (C++ function), 265
rb_bytes_available (C++ function), 265
rb_bytes_filled (C++ function), 266
rb_create (C++ function), 264
rb_destroy (C++ function), 264
RB_DONE (C 宏), 268
rb_done_write (C++ function), 267
RB_FAIL (C 宏), 268
rb_get_reader_holder (C++ function), 267
rb_get_size (C++ function), 266
rb_get_writer_holder (C++ function), 268
RB_OK (C 宏), 268
rb_read (C++ function), 266
rb_reset (C++ function), 265
rb_reset_is_done_write (C++ function), 265
rb_set_reader_holder (C++ function), 267
rb_set_writer_holder (C++ function), 268
RB_TIMEOUT (C 宏), 268
rb_unblock_reader (C++ function), 267
rb_write (C++ function), 266
rec_event_cb_t (C++ type), 225
recorder_encoder_cfg_t (C++ class), 228
recorder_encoder_cfg_t::encoder (C++ member), 229
recorder_encoder_cfg_t::resample (C++ member), 229
recorder_encoder_create (C++ function), 228
recorder_encoder_destroy (C++ function), 228
recorder_encoder_handle_t (C++ type), 229
recorder_sr_cfg_t (C++ class), 226
recorder_sr_cfg_t::afe_cfg (C++ member), 226
recorder_sr_cfg_t::feed_task_core (C++ member), 226
recorder_sr_cfg_t::feed_task_prio (C++ member), 226
recorder_sr_cfg_t::feed_task_stack (C++ member), 227
recorder_sr_cfg_t::fetch_task_core (C++ member), 227
recorder_sr_cfg_t::fetch_task_prio (C++ member), 227
recorder_sr_cfg_t::fetch_task_stack (C++ member), 227
recorder_sr_cfg_t::input_order (C++ member), 226
recorder_sr_cfg_t::mn_language (C++ member), 227
recorder_sr_cfg_t::multinet_init (C++ member), 226
recorder_sr_cfg_t::partition_label (C++ member), 227
recorder_sr_cfg_t::rb_size (C++ member), 227
recorder_sr_cfg_t::wn_wakeword (C++ member), 227
recorder_sr_create (C++ function), 225
recorder_sr_destroy (C++ function), 226
recorder_sr_handle_t (C++ type), 228
recorder_sr_reset_speech_cmd (C++ function), 226
resample, **419**
resample filter, **419**
resonant frequency, **419**
reverberation, **419**
RGB, **419**
ring buffer, **419**
ringbuf_handle_t (C++ type), 269
RSP_FILTER_BUFFER_BYTE (C 宏), 162
rsp_filter_cfg_t (C++ class), 161
rsp_filter_cfg_t::complexity (C++ member), 161
rsp_filter_cfg_t::dest_bits (C++ member), 161
rsp_filter_cfg_t::dest_ch (C++ member), 161

rsp_filter_cfg_t::dest_rate (C++ member), 161
rsp_filter_cfg_t::down_ch_idx (C++ member), 161
rsp_filter_cfg_t::max_indata_bytes (C++ member), 161
rsp_filter_cfg_t::mode (C++ member), 161
rsp_filter_cfg_t::out_len_bytes (C++ member), 161
rsp_filter_cfg_t::out_rb_size (C++ member), 162
rsp_filter_cfg_t::prefer_flag (C++ member), 162
rsp_filter_cfg_t::src_bits (C++ member), 161
rsp_filter_cfg_t::src_ch (C++ member), 161
rsp_filter_cfg_t::src_rate (C++ member), 161
rsp_filter_cfg_t::stack_in_ext (C++ member), 162
rsp_filter_cfg_t::task_core (C++ member), 162
rsp_filter_cfg_t::task_prio (C++ member), 162
rsp_filter_cfg_t::task_stack (C++ member), 162
rsp_filter_cfg_t::type (C++ member), 161
rsp_filter_change_src_info (C++ function), 160
rsp_filter_init (C++ function), 160
RSP_FILTER_RINGBUFFER_SIZE (C 宏), 162
rsp_filter_set_src_info (C++ function), 160
RSP_FILTER_TASK_CORE (C 宏), 162
RSP_FILTER_TASK_PRIO (C 宏), 162
RSP_FILTER_TASK_STACK (C 宏), 162

S

SBC, **419**
SD card, **419**
SD_MODE_1_LINE (C++ enumerator), 247
SD_MODE_4_LINE (C++ enumerator), 247
SD_MODE_8_LINE (C++ enumerator), 247
SD_MODE_MAX (C++ enumerator), 247
SD_MODE_SPI (C++ enumerator), 247
sdcard_list_choose (C++ function), 116
sdcard_list_create (C++ function), 114
sdcard_list_current (C++ function), 116
sdcard_list_destroy (C++ function), 117
sdcard_list_exist (C++ function), 115
sdcard_list_get_url_id (C++ function), 117
sdcard_list_get_url_num (C++ function), 116
sdcard_list_next (C++ function), 115
sdcard_list_prev (C++ function), 115
sdcard_list_reset (C++ function), 115
sdcard_list_save (C++ function), 117
sdcard_list_show (C++ function), 114
sdcard_scan (C++ function), 113
sdcard_scan_cb_t (C++ type), 114
SDCARD_STATUS_CARD_DETECT_CHANGE (C++ enumerator), 246
SDCARD_STATUS_MOUNT_ERROR (C++ enumerator), 246
SDCARD_STATUS_MOUNTED (C++ enumerator), 246
SDCARD_STATUS_UNKNOWN (C++ enumerator), 246
SDCARD_STATUS_UNMOUNT_ERROR (C++ enumerator), 246
SDCARD_STATUS_UNMOUNTED (C++ enumerator), 246
service_callback (C++ type), 218
service_ctrl (C++ type), 218
service_event_t (C++ class), 217
service_event_t::data (C++ member), 217
service_event_t::len (C++ member), 217
service_event_t::source (C++ member), 217
service_event_t::type (C++ member), 217
SERVICE_STATE_CONNECTED (C++ enumerator), 218
SERVICE_STATE_CONNECTING (C++ enumerator), 218
SERVICE_STATE_IDLE (C++ enumerator), 218
SERVICE_STATE_RUNNING (C++ enumerator), 218
SERVICE_STATE_STOPPED (C++ enumerator), 218
service_state_t (C++ enum), 218
SERVICE_STATE_UNKNOWN (C++ enumerator), 218

Session Initiation Protocol, **419**
signal-to-echo ratio, **419**
signal-to-noise ratio, **420**
SIP, **420**
smart_config_create (*C++ function*), 181
SMART_CONFIG_INFO_DEFAULT (*C 宏*), 181
smart_config_info_t (*C++ class*), 181
smart_config_info_t::type (*C++ member*), 181
SmartConfig, **420**
SNR, **420**
sonic, **420**
sonic_cfg_t (*C++ class*), 164
sonic_cfg_t::out_rb_size (*C++ member*), 164
sonic_cfg_t::sonic_info (*C++ member*), 164
sonic_cfg_t::stack_in_ext (*C++ member*), 165
sonic_cfg_t::task_core (*C++ member*), 164
sonic_cfg_t::task_prio (*C++ member*), 164
sonic_cfg_t::task_stack (*C++ member*), 164
sonic_info_t (*C++ class*), 164
sonic_info_t::channel (*C++ member*), 164
sonic_info_t::pitch (*C++ member*), 164
sonic_info_t::resample_linear_interpolate (*C++ member*), 164
sonic_info_t::samplerate (*C++ member*), 164
sonic_info_t::speed (*C++ member*), 164
sonic_init (*C++ function*), 163
SONIC_RINGBUFFER_SIZE (*C 宏*), 165
sonic_set_info (*C++ function*), 163
sonic_set_pitch_and_speed_info (*C++ function*), 163
SONIC_SET_VALUE_FOR_INITIALIZATION (*C 宏*), 165
SONIC_TASK_CORE (*C 宏*), 165
SONIC_TASK_PRIO (*C 宏*), 165
SONIC_TASK_STACK (*C 宏*), 165
sound card, **420**
sound level meter, **420**
sound pickup hole, **420**
sound pickup tube, **420**
sound transmission loss, **420**
source_info_init (*C++ function*), 155
speech, **420**
speech recognition, **420**
SPI Flash File System, **420**
SPIFFS, **420**
SPIFFS stream, **420**
SPIFFS_STATUS_MOUNT_ERROR (*C++ enumerator*), 249
SPIFFS_STATUS_MOUNTED (*C++ enumerator*), 249
SPIFFS_STATUS_UNKNOWN (*C++ enumerator*), 249
SPIFFS_STATUS_UNMOUNT_ERROR (*C++ enumerator*), 249
SPIFFS_STATUS_UNMOUNTED (*C++ enumerator*), 249
SPIFFS_STREAM_BUF_SIZE (*C 宏*), 103
SPIFFS_STREAM_CFG_DEFAULT (*C 宏*), 103
spiffs_stream_cfg_t (*C++ class*), 102
spiffs_stream_cfg_t::buf_sz (*C++ member*), 102
spiffs_stream_cfg_t::out_rb_size (*C++ member*), 102
spiffs_stream_cfg_t::task_core (*C++ member*), 102
spiffs_stream_cfg_t::task_prio (*C++ member*), 102
spiffs_stream_cfg_t::task_stack (*C++ member*), 102
spiffs_stream_cfg_t::type (*C++ member*), 102
spiffs_stream_cfg_t::write_header (*C++ member*), 102
spiffs_stream_init (*C++ function*), 102
SPIFFS_STREAM_RINGBUFFER_SIZE (*C 宏*), 103
SPIFFS_STREAM_TASK_CORE (*C 宏*), 103
SPIFFS_STREAM_TASK_PRIO (*C 宏*), 103
SPIFFS_STREAM_TASK_STACK (*C 宏*), 103
SR, **420**
SR_OUTPUT_RB_SIZE (*C 宏*), 227
STL, **421**
stream_func (*C++ type*), 43
subband codec, **421**
super wide band, **421**

SWB, **421**

T

tape measure, **421**
TCP_SERVER_DEFAULT_RESPONSE_LENGTH (*C 宏*), 105
TCP_STREAM_BUF_SIZE (*C 宏*), 105
TCP_STREAM_CFG_DEFAULT (*C 宏*), 105
tcp_stream_cfg_t (*C++ class*), 104
tcp_stream_cfg_t::event_ctx (*C++ member*), 105
tcp_stream_cfg_t::event_handler (*C++ member*), 104
tcp_stream_cfg_t::ext_stack (*C++ member*), 104
tcp_stream_cfg_t::host (*C++ member*), 104
tcp_stream_cfg_t::port (*C++ member*), 104
tcp_stream_cfg_t::task_core (*C++ member*), 104
tcp_stream_cfg_t::task_prio (*C++ member*), 104
tcp_stream_cfg_t::task_stack (*C++ member*), 104
tcp_stream_cfg_t::timeout_ms (*C++ member*), 104
tcp_stream_cfg_t::type (*C++ member*), 104
TCP_STREAM_DEFAULT_PORT (*C 宏*), 105
tcp_stream_event_handle_cb (*C++ type*), 105
tcp_stream_event_msg (*C++ class*), 104
tcp_stream_event_msg::data (*C++ member*), 104
tcp_stream_event_msg::data_len (*C++ member*), 104
tcp_stream_event_msg::sock_fd (*C++ member*), 104
tcp_stream_event_msg::source (*C++ member*), 104
tcp_stream_event_msg_t (*C++ type*), 105
tcp_stream_init (*C++ function*), 103
TCP_STREAM_STATE_CONNECTED (*C++ enumerator*), 105
TCP_STREAM_STATE_NONE (*C++ enumerator*), 105
tcp_stream_status_t (*C++ enum*), 105
TCP_STREAM_TASK_CORE (*C 宏*), 105
TCP_STREAM_TASK_PRIO (*C 宏*), 105
TCP_STREAM_TASK_STACK (*C 宏*), 105
TERMINATION_TYPE_DONE (*C++ enumerator*), 66
TERMINATION_TYPE_MAX (*C++ enumerator*), 66
TERMINATION_TYPE_NOW (*C++ enumerator*), 66
text-to-speech, **421**
THD, **421**
timer_callback (*C++ type*), 239
tolerance, **421**
tone, **421**
TONE_STREAM_BUF_SIZE (*C 宏*), 107
TONE_STREAM_CFG_DEFAULT (*C 宏*), 107
tone_stream_cfg_t (*C++ class*), 106
tone_stream_cfg_t::buf_sz (*C++ member*), 106
tone_stream_cfg_t::extern_stack (*C++ member*), 107
tone_stream_cfg_t::label (*C++ member*), 107
tone_stream_cfg_t::out_rb_size (*C++ member*), 106
tone_stream_cfg_t::task_core (*C++ member*), 106
tone_stream_cfg_t::task_prio (*C++ member*), 107
tone_stream_cfg_t::task_stack (*C++ member*), 106
tone_stream_cfg_t::type (*C++ member*), 106
tone_stream_cfg_t::use_delegate (*C++ member*), 107
TONE_STREAM_EXT_STACK (*C 宏*), 107
tone_stream_init (*C++ function*), 106
TONE_STREAM_RINGBUFFER_SIZE (*C 宏*), 107
TONE_STREAM_TASK_CORE (*C 宏*), 107
TONE_STREAM_TASK_PRIO (*C 宏*), 107
TONE_STREAM_TASK_STACK (*C 宏*), 107
TONE_STREAM_USE_DELEGATE (*C 宏*), 107
total harmonic distortion, **421**
TOUCH_PAD_SEL0 (*C++ enumerator*), 252
TOUCH_PAD_SEL1 (*C++ enumerator*), 252
TOUCH_PAD_SEL2 (*C++ enumerator*), 252

TOUCH_PAD_SEL3 (*C++ enumerator*), 252
TOUCH_PAD_SEL4 (*C++ enumerator*), 252
TOUCH_PAD_SEL5 (*C++ enumerator*), 252
TOUCH_PAD_SEL6 (*C++ enumerator*), 252
TOUCH_PAD_SEL7 (*C++ enumerator*), 252
TOUCH_PAD_SEL8 (*C++ enumerator*), 252
TOUCH_PAD_SEL9 (*C++ enumerator*), 252
TTS, **421**
TTS_STREAM_BUF_SIZE (*C 宏*), 112
TTS_STREAM_CFG_DEFAULT (*C 宏*), 112
tts_stream_cfg_t (*C++ class*), 111
tts_stream_cfg_t::buf_sz (*C++ member*), 111
tts_stream_cfg_t::ext_stack (*C++ member*), 111
tts_stream_cfg_t::out_rb_size (*C++ member*), 111
tts_stream_cfg_t::task_core (*C++ member*), 111
tts_stream_cfg_t::task_prio (*C++ member*), 111
tts_stream_cfg_t::task_stack (*C++ member*), 111
tts_stream_cfg_t::type (*C++ member*), 111
tts_stream_get_speed (*C++ function*), 111
tts_stream_init (*C++ function*), 110
TTS_STREAM_RINGBUFFER_SIZE (*C 宏*), 112
tts_stream_set_speed (*C++ function*), 110
tts_stream_set_strings (*C++ function*), 110
TTS_STREAM_TASK_CORE (*C 宏*), 112
TTS_STREAM_TASK_PRIO (*C 宏*), 112
TTS_STREAM_TASK_STACK (*C 宏*), 112
TTS_VOICE_SPEED_0 (*C++ enumerator*), 112
TTS_VOICE_SPEED_1 (*C++ enumerator*), 112
TTS_VOICE_SPEED_2 (*C++ enumerator*), 112
TTS_VOICE_SPEED_3 (*C++ enumerator*), 112
TTS_VOICE_SPEED_4 (*C++ enumerator*), 112
TTS_VOICE_SPEED_5 (*C++ enumerator*), 112
TTS_VOICE_SPEED_MAX (*C++ enumerator*), 112
tts_voice_speed_t (*C++ enum*), 112

V

VAD, **421**
voice activity detection, **421**
VoIP, **421**
vol_monitor_create (*C++ function*), 205
vol_monitor_destroy (*C++ function*), 205
VOL_MONITOR_EVENT_BAT_FULL (*C++ enumerator*), 207
VOL_MONITOR_EVENT_BAT_LOW (*C++ enumerator*), 207
vol_monitor_event_cb (*C++ type*), 207
VOL_MONITOR_EVENT_FREQ_REPORT (*C++ enumerator*), 207
vol_monitor_event_t (*C++ enum*), 207
vol_monitor_handle_t (*C++ type*), 207
vol_monitor_param_t (*C++ class*), 206
vol_monitor_param_t::deinit (*C++ member*), 206
vol_monitor_param_t::init (*C++ member*), 206
vol_monitor_param_t::read_freq (*C++ member*), 207
vol_monitor_param_t::report_freq (*C++ member*), 207
vol_monitor_param_t::user_data (*C++ member*), 206
vol_monitor_param_t::vol_full_threshold (*C++ member*), 207
vol_monitor_param_t::vol_get (*C++ member*), 206
vol_monitor_param_t::vol_low_threshold (*C++ member*), 207
vol_monitor_set_event_cb (*C++ function*), 205
vol_monitor_set_report_freq (*C++ function*), 206
vol_monitor_start_freq_report (*C++ function*), 205
vol_monitor_stop_freq_report (*C++ function*), 206

W

wake word, **421**
wake word engine, **421**
wake-up, **422**

WakeNet, **421**
wav_decoder_cfg_t (*C++ class*), 150
wav_decoder_cfg_t::out_rb_size (*C++ member*), 150
wav_decoder_cfg_t::stack_in_ext (*C++ member*), 150
wav_decoder_cfg_t::task_core (*C++ member*), 150
wav_decoder_cfg_t::task_prio (*C++ member*), 150
wav_decoder_cfg_t::task_stack (*C++ member*), 150
wav_decoder_init (*C++ function*), 150
WAV_DECODER_RINGBUFFER_SIZE (*C 宏*), 151
WAV_DECODER_TASK_CORE (*C 宏*), 151
WAV_DECODER_TASK_PRIO (*C 宏*), 151
WAV_DECODER_TASK_STACK (*C 宏*), 151
wav_encoder_cfg_t (*C++ class*), 151
wav_encoder_cfg_t::out_rb_size (*C++ member*), 151
wav_encoder_cfg_t::stack_in_ext (*C++ member*), 152
wav_encoder_cfg_t::task_core (*C++ member*), 151
wav_encoder_cfg_t::task_prio (*C++ member*), 151
wav_encoder_cfg_t::task_stack (*C++ member*), 151
wav_encoder_init (*C++ function*), 151
WAV_ENCODER_RINGBUFFER_SIZE (*C 宏*), 152
WAV_ENCODER_TASK_CORE (*C 宏*), 152
WAV_ENCODER_TASK_PRIO (*C 宏*), 152
WAV_ENCODER_TASK_STACK (*C 宏*), 152
WB, **422**
wideband, **422**
WIFI_CONFIG_AIRKISS (*C++ enumerator*), 244
WIFI_CONFIG_BLUEFI (*C++ enumerator*), 245
WIFI_CONFIG_ESPTOUCH (*C++ enumerator*), 244
WIFI_CONFIG_ESPTOUCH_AIRKISS (*C++ enumerator*), 244
WIFI_CONFIG_WEB (*C++ enumerator*), 245
WIFI_CONFIG_WPS (*C++ enumerator*), 244
WIFI_SERV_EVENT_CONNECTED (*C++ enumerator*), 177
WIFI_SERV_EVENT_CONNECTING (*C++ enumerator*), 177
WIFI_SERV_EVENT_DISCONNECTED (*C++ enumerator*), 177
WIFI_SERV_EVENT_SETTING_FAILED (*C++ enumerator*), 177
WIFI_SERV_EVENT_SETTING_FINISHED (*C++ enumerator*), 177
WIFI_SERV_EVENT_SETTING_TIMEOUT (*C++ enumerator*), 177
WIFI_SERV_EVENT_UNKNOWN (*C++ enumerator*), 177
WIFI_SERV_STA_AP_NOT_FOUND (*C++ enumerator*), 177
WIFI_SERV_STA_AUTH_ERROR (*C++ enumerator*), 177
WIFI_SERV_STA_BY_USER (*C++ enumerator*), 177
WIFI_SERV_STA_COM_ERROR (*C++ enumerator*), 177
WIFI_SERV_STA_SET_INFO (*C++ enumerator*), 177
WIFI_SERV_STA_UNKNOWN (*C++ enumerator*), 177
wifi_service_config_t (*C++ class*), 176
wifi_service_config_t::cb_ctx (*C++ member*), 176
wifi_service_config_t::evt_cb (*C++ member*), 176
wifi_service_config_t::extern_stack (*C++ member*), 176
wifi_service_config_t::max_prov_retry_time (*C++ member*), 176
wifi_service_config_t::max_retry_time (*C++ member*), 176
wifi_service_config_t::max_ssid_num (*C++ member*), 176
wifi_service_config_t::setting_timeout_s (*C++ member*), 176
wifi_service_config_t::task_core (*C++ member*), 176
wifi_service_config_t::task_prio (*C++ member*), 176

wifi_service_config_t::task_stack (C++ member), 176
wifi_service_config_t::user_data (C++ member), 176
wifi_service_connect (C++ function), 175
wifi_service_create (C++ function), 175
WIFI_SERVICE_DEFAULT_CONFIG (C 宏), 177
wifi_service_destroy (C++ function), 175
wifi_service_disconnect (C++ function), 175
wifi_service_disconnect_reason_get (C++ function), 175
wifi_service_disconnect_reason_t (C++ enum), 177
wifi_service_erase_ssid_manager_info (C++ function), 176
wifi_service_event_t (C++ enum), 177
wifi_service_get_last_ssid_cfg (C++ function), 176
wifi_service_register_setting_handle (C++ function), 175
wifi_service_set_sta_info (C++ function), 175
wifi_service_setting_start (C++ function), 175
wifi_service_setting_stop (C++ function), 175
wifi_service_state_get (C++ function), 175
wifi_service_update_sta_info (C++ function), 175
wifi_setting_func (C++ type), 180
wifi_setting_teardown_func (C++ type), 180
wifi_ssid_manager_create (C++ function), 184
wifi_ssid_manager_destroy (C++ function), 184
wifi_ssid_manager_erase_all (C++ function), 184
wifi_ssid_manager_get_best_config (C++ function), 184
wifi_ssid_manager_get_latest_config (C++ function), 184
wifi_ssid_manager_get_ssid_num (C++ function), 184
wifi_ssid_manager_handle_t (C++ type), 185
wifi_ssid_manager_list_show (C++ function), 184
wifi_ssid_manager_save (C++ function), 184
WWE, **422**

Y

YUV, **422**

Z

zl38063_codec_config_i2s (C++ function), 289
zl38063_codec_ctrl_state (C++ function), 288
zl38063_codec_deinit (C++ function), 288
zl38063_codec_get_voice_volume (C++ function), 289
zl38063_codec_init (C++ function), 288
zl38063_codec_set_voice_mute (C++ function), 289
zl38063_codec_set_voice_volume (C++ function), 289