ESP32-S3

ESP-SR User Guide

Release master
Espressif Systems
May 14, 2024

Table of contents

Table of contents i

1 Getting Started 3
- 1.1 Overview 3
- 1.2 What You Need 3
  - 1.2.1 Hardware 3
  - 1.2.2 Software 3
- 1.3 Compile an Example 3

2 AFE Audio Front-end 5
- 2.1 Audio Front-end Framework 5
  - 2.1.1 Overview 5
  - 2.1.2 Usage Scenarios 5
  - 2.1.3 Select AFE Handle 8
  - 2.1.4 Input Audio Data 8
  - 2.1.5 Output Audio 9
  - 2.1.6 Enable Wake Word Engine WakeNet 9
  - 2.1.7 Enable Acoustic Echo Cancellation (AEC) 9
  - 2.1.8 Resource Occupancy 10
- 2.2 Espressif Microphone Design Guidelines 10
  - 2.2.1 Microphone Electrical Performance Requirement 10
  - 2.2.2 Microphone Structure Design Suggestion 10
  - 2.2.3 Microphone Array Design Suggestion 10
  - 2.2.4 Microphone Leakproofness Suggestion 11
  - 2.2.5 Echo Reference Signal Design Suggestion 11
  - 2.2.6 Microphone Array Consistency Verification 11

3 Wake Word 13
- 3.1 WakeNet Wake Word Model 13
  - 3.1.1 Overview 13
  - 3.1.2 Use WakeNet 14
  - 3.1.3 Resource Occupancy 15
- 3.2 Espressif Speech Wake-up Solution Customization Process 15
  - 3.2.1 Wake Word Customization Process 15
  - 3.2.2 Requirements on Corpus 15
  - 3.2.3 Hardware Design and Test 16

4 Command Word 17
- 4.1 MultiNet Command Word Recognition Model 17
- 4.2 Commands Recognition Process 17
- 4.3 Speech Commands Customization Methods 17
  - 4.3.1 MultiNet7 customize speech commands 18
  - 4.3.2 MultiNet6 customize speech commands 18
  - 4.3.3 MultiNet5 customize speech commands 18
  - 4.3.4 Customize Speech Commands Via API calls 19
- 4.4 Use MultiNet 21
  - 4.4.1 Initialize MultiNet 21

4.4.2 Run MultiNet . . . 21
4.4.3 MultiNet Output . . . 21
4.5 Resource Occupancy . . . 22

5 TTS Speech Synthesis Model **23**
5.1 Overview . . . 23
5.2 Examples . . . 23
5.3 Programming Procedures . . . 24
5.4 Resource Occupancy . . . 24

6 Flashing Models **25**
6.1 Configuration . . . 25
6.1.1 Model Data Path . . . 25
6.1.2 Use AFE . . . 26
6.1.3 Use WakeNet . . . 26
6.1.4 Use Multinet . . . 27
6.2 How To Use . . . 27
6.2.1 Load Model Data from flash . . . 27
6.2.2 Load Model Data from SD Card . . . 28

7 Benchmark **29**
7.1 AFE . . . 29
7.1.1 Resource Consumption . . . 29
7.2 WakeNet . . . 29
7.2.1 Resource Consumption . . . 29
7.2.2 Performance Test . . . 29
7.3 MultiNet . . . 30
7.3.1 Resource Consumption . . . 30
7.3.2 Word Error Rate Performance Test . . . 30
7.3.3 Speech Commands Performance Test . . . 30
7.4 TTS . . . 30
7.4.1 Resource Consumption . . . 30
7.4.2 Performance Test . . . 30

8 Test Method and Test Report **31**
8.1 Test Room Requirement . . . 31
8.2 Test Case Design . . . 31
8.3 Espressif Test and Result . . . 32
8.3.1 Wake-up Rate Test . . . 32
8.3.2 Speech Recognition Rate Test . . . 33
8.3.3 False Wake-up Rate Test . . . 33
8.3.4 Response Accuracy Rate Under Playback . . . 33
8.3.5 Response Time Test . . . 34

9 Glossary **35**
9.1 General Terms . . . 35
9.2 Unique Terms . . . 35

This document contains ESP-SR usage for ESP32-S3 only.

Chapter 1

Getting Started

Espressif ESP-SR helps you build AI voice solution based on ESP32 or ESP32-S3 chips. This document introduces the algorithms and models in ESP-SR via some simple examples.

1.1 Overview

ESP-SR includes the following modules:

- *Audio Front-end AFE*
- *Wake Word Engine WakeNet*
- *Speech Command Word Recognition MultiNet*
- *Speech Synthesis (only supports Chinese language)*

1.2 What You Need

1.2.1 Hardware

- an audio development board. Recommendation: ESP32-S3-Korvo-1 or ESP32-S3-Korvo-2
- USB 2.0 cable (USB A / micro USB B)
- PC (Linux)

Note: Some development boards currently have the Type C interface. Make sure you use the proper cable to connect the board!

1.2.2 Software

- Download ESP-SKAINET, which also downloads ESP-SR as a component.
- Install the ESP-IDF version recommended in ESP-SKAINET. For detailed steps, please see Section Getting Started in ESP-IDF Programming Guide.

1.3 Compile an Example

- Navigate to ESP-SKAINET/examples/en_speech_commands_recognition .

- Compile and run an example following the instructions.
- The example only supports commands in English. Users can wake up the chip by using wake word "Hi ESP". Note that the chip stops listening to commands if the users wake up the chip and do not give any commands for some time. In this case, just wake up the chip again by saying the wake word.

Chapter 2

AFE Audio Front-end

2.1 Audio Front-end Framework

2.1.1 Overview

Any voice-enabled product needs to perform well in a noisy environment, and audio front-end (AFE) algorithms play an important role in building a sensitive voice-user interface (VUI). Espressif's AI Lab has created a set of audio front-end algorithms that can offer this functionality. Customers can use these algorithms with Espressif's powerful ESP32-S3 series of chips, in order to build high-performance, yet low-cost, products with a voice-user interface.

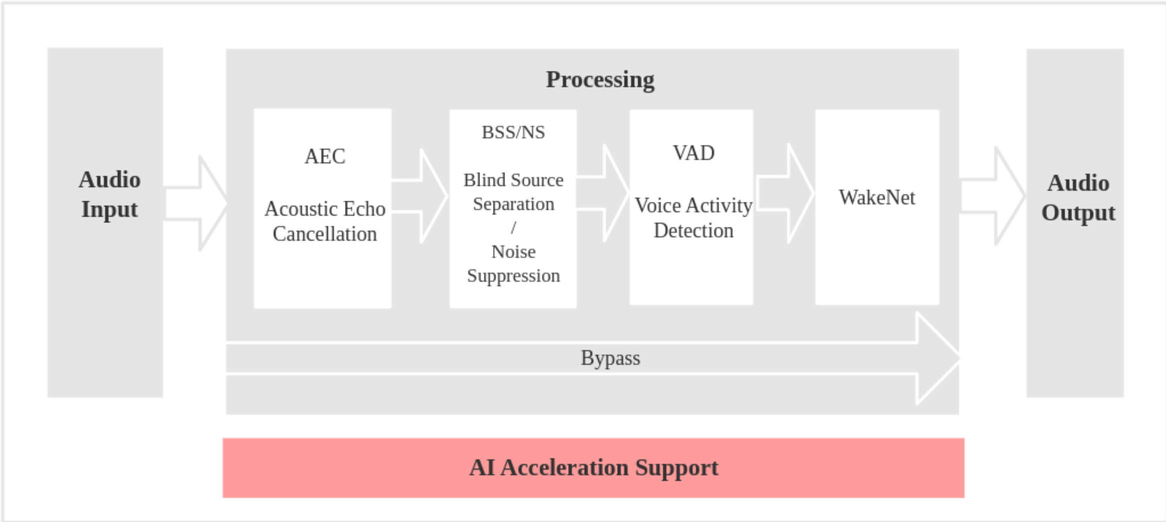
Name	Description
AEC (Acoustic Echo Cancellation)	Supports maximum two-mic processing, which can effectively remove the echo in the mic input signal, and help with further speech recognition.
NS (Noise Suppression)	Supports single-channel processing and can suppress the non-human noise in single-channel audio, especially for stationary noise.
BSS (Blind Source Separation)	Supports dual-channel processing, which can well separate the target sound source from the rest of the interference sound, so as to extract the useful audio signal and ensure the quality of the subsequent speech.
MISO (Multi Input Single Output)	Supports dual channel input and single channel output. It is used to select a channel of audio output with high signal-to-noise ratio when there is no WakeNet enable in the dual mic scene.
VAD (Voice Activity Detection)	Supports real-time output of the voice activity state of the current frame.
AGC (Automatic Gain Control)	Dynamically adjusts the amplitude of the output audio, and amplifies the output amplitude when a weak signal is input; When the input signal reaches a certain strength, the output amplitude will be compressed.
WakeNet	A wake word engine built upon neural network, and is specially designed for low-power embedded MCUs.

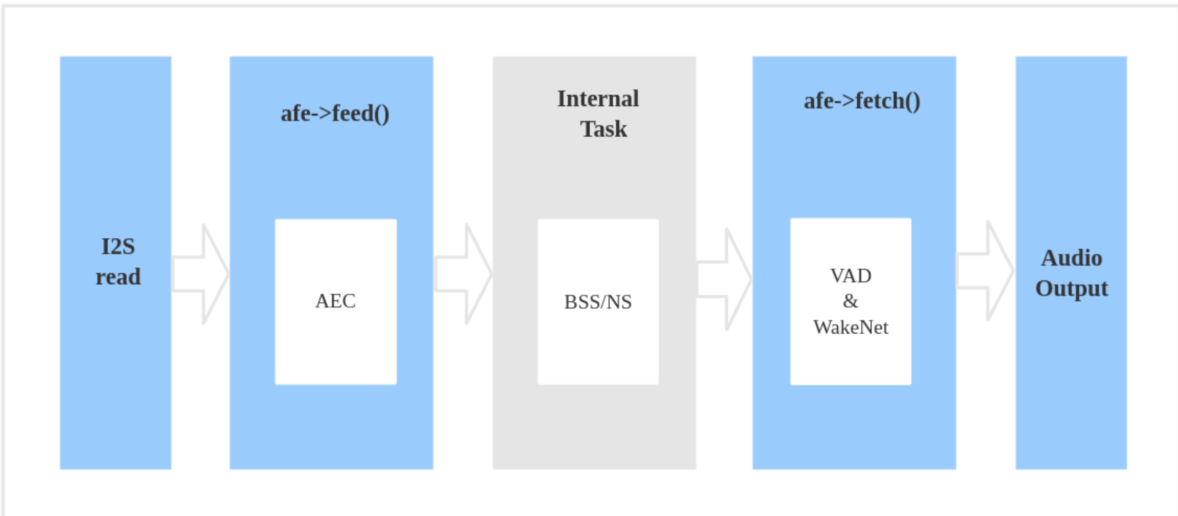
2.1.2 Usage Scenarios

This section introduces two typical usage scenarios of Espressif AFE framework.

Speech Recognition

Workflow

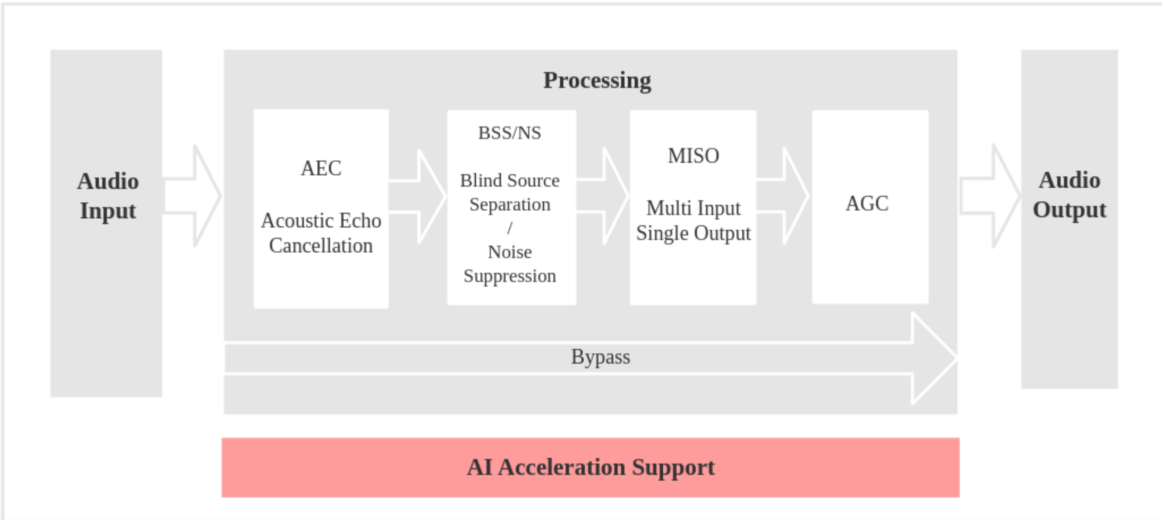

Audio Input
Processing
AEC
Acoustic Echo Cancellation
BSS/NS
Blind Source Separation
/
Noise Suppression
VAD
Voice Activity Detection
WakeNet
Audio Output
Bypass
AI Acceleration Support


I2S read
afe->feed()
AEC
Internal Task
BSS/NS
afe->fetch()
VAD
&
WakeNet
Audio Output

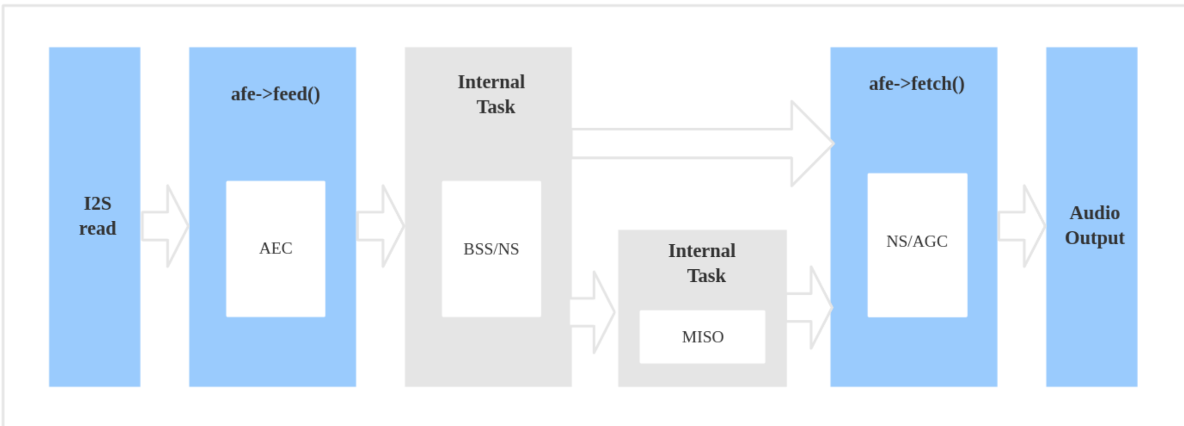
Data Flow

1. Use `ESP_AFE_SR_HANDLE()` to create and initialize AFE. Note, `voice_communication_init` must be configured as false.
2. Use `feed()` to input audio data, which will perform the AEC algorithm inside `feed()` first.
3. Perform the BSS/NS algorithms inside `feed()` first.
4. Use `fetch()` to obtain processed single channel audio data and related information. Note, VAD processing and wake word detection will be performed inside `fetch()`. The specific behavior depends on the configuration of `afe_config_t` structure.

Voice Communication



Workflow



Data Flow

1. Use `ESP_AFE_VC_HANDLE()` to create and initialize AFE. Note, `voice_communication_init` must be configured as true.
2. Use `feed()` to input audio data, which will perform the AEC algorithm inside `feed()` first.
3. Perform the BSS/NS algorithms inside `feed()` first. Additional MISO algorithm will be performed for dual mic setup.
4. Use `fetch()` to obtain processed single channel audio data and related information. The AGC algorithm processing will be carried out. And the specific gain depends on the config of `afe_config_t` structure. If it’ s dual mic, the NS algorithm processing will be carried out before AGC.

Note:

1. The `wakenet_init` and `voice_communication_init` in `afe_config_t` cannot be configured to true at the same time.
2. `feed()` and `fetch()` are visible to users, while other AFE interal tasks such as BSS/NS/MISO are not visible to users.
3. AEC algorithm is performed in `feed()`.
4. When `aec_init` is configured to false, BSS/NS algorithms are performed in `feed()`.

2.1.3 Select AFE Handle

Espressif AFE supports both single mic and dual mic setups, and allows flexible combinations of algorithms.

- **Single mic**
  - Internal task is performed inside the NS algorithm
- **Dual mic**
  - Internal task is performed inside the BSS algorithm
  - An additional internal task is performed inside the MISO algorithm for voice communication scenario (i.e., `wakenet_init` = false and `voice_communication_init` = true)

To obtain the AFE Handle, use the commands below:

- Speech recognition

```
esp_afe_sr_iface_t *afe_handle = &ESP_AFE_SR_HANDLE;
```

- Voice communication

```
esp_afe_sr_iface_t *afe_handle = &ESP_AFE_VC_HANDLE;
```

2.1.4 Input Audio Data

Currently, Espressif AFE framework supports both single mic and dual mic setups. Users can configure the number of channels based on the input audio (`esp_afe_sr_iface_op_feed_t()`).

To be specific, users can configure the `pcm_config` in `AFE_CONFIG_DEFAULT()`:

- `total_ch_num` : total number of channels
- `mic_num` : number of mic channels
- `ref_num` : number of REF channels

When configuring, note the following requirements:

1. `total_ch_num` = `mic_num` + `ref_num`
2. `ref_num` = 0 or `ref_num` = 1 (This is because AEC only supports up to one reference data now)

The supported configurations are:

```
total_ch_num=1, mic_num=1, ref_num=0
total_ch_num=2, mic_num=1, ref_num=1
total_ch_num=2, mic_num=2, ref_num=0
total_ch_num=3, mic_num=2, ref_num=1
```

AFE Single Mic

- Input audio data format: 16 KHz, 16 bit, two channels (one is mic data, another is REF data). Note that if AEC is not required, then there is no need for reference data. Therefore, users can only configure one channel of mic data, and the ref_num can be set to 0.

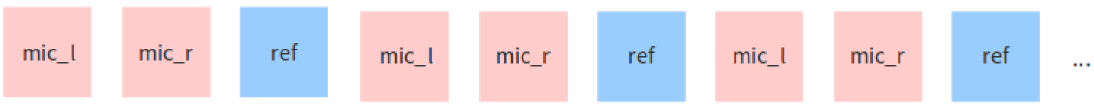
- The input data frame length varies to the algorithm modules configured by the user. Users can use `get_feed_chunksize()` to get the number of sampling points (the data type of sampling points is int16).

The input data is arranged as follows:

AFE Dual Mic

- Input audio data format: 16 KHz, 16 bit, three channels (two are mic data, another is REF data). Note that if AEC is not required, then there is no need for reference data. Therefore, users can only configure two channels of mic data, and the ref_num can be set to 0.
- The input data frame length varies to the algorithm modules configured by the user. Users can use `get_feed_chunksize()` to obtain the data size required (i.e., `get_feed_chunksize()` * `total_ch_num` * sizeof(short)).

The input data is arranged as follows:



2.1.5 Output Audio

The output audio of AFE is single-channel data. - In the speech recognition scenario, AFE outputs single-channel data with human voice when WakeNet is enabled. - In the voice communication scenario, AFE outputs single channel data with higher signal-to-noise ratio.

2.1.6 Enable Wake Word Engine WakeNet

When performing AFE audio front-end processing, the user can choose whether to enable wake word engine *WakeNet* to allow waking up the chip via wake words.

Users can disable WakeNet to reduce the CPU resource consumption and perform other operations after wake-up, such as offline or online speech recognition. To do so, users can configure `disable_wakenet()` to enter Bypass mode.

Users can also call `enable_wakenet()` to enable WakeNet later whenever needed.

ESP32-S3 allows users to switch among different wake words. After the initialization of AFE, ESP32-S3 allows users to change wake words by calling `set_wakenet()` . For example, use `set_wakenet(afe_data, "wn9_hilexin")` to use "Hi Lexin" as the wake word. For details on how to configure more than one wake words, see Section *flash_model*.

2.1.7 Enable Acoustic Echo Cancellation (AEC)

The usage of AEC is similar to that of WakeNet. Users can disable or enable AEC according to requirements.

- Disable AEC

```
afe->disable_aec(afe_data);
```

- Enable AEC

```
afe->enable_aec(afe_data);
```

2.1.8 Resource Occupancy

For the resource occupancy for this model, see *Resource Occupancy*.

2.2 Espressif Microphone Design Guidelines

This document provides microphone design guidelines and suggestions for the ESP32-S3 series of audio development boards.

2.2.1 Microphone Electrical Performance Requirement

- Type: omnidirectional MEMS microphone
- Sensitivity
  - Under 1 Pa sound pressure, the sensitivity should be no less than -38 dBV for analog microphones and -26 dB for digital microphones.
  - Tolerance should be within ±2 dB for microphones. And tolerance for microphone arrays should be within ±1 dB.
- Signal-to-noise ratio (SNR)
  - SNR: No less than 62 dB. Higher than 64 dB is recommended.
  - Frequency response should only fluctuate within ±3 dB from 50 to 16 kHz.
  - PSRR should be larger than 55 dB for microphones.

2.2.2 Microphone Structure Design Suggestion

- The aperture or width of the microphone hole is recommended to be greater than 1 mm, the pickup pipe should be as short as possible, and the cavity should be as small as possible. All to ensure that the resonance frequency of the microphone and structural components is above 9 kHz.
- The depth and diameter of the pickup hole are less than 2:1, and the thickness of the shell is recommended to be 1 mm. Increase the hole size of microphone if the shell is too thick.
- The microphone hole must be protected by an anti-dust mesh.
- Silicone sleeve or foam must be added between the microphone and the device shell for sealing and damping, and an interference fit design is required to ensure the leakproofness of the microphone.
- The microphone hole cannot be covered. The microphone in the bottom must keep some clearance from the surfaces such as desktop. Therefore, it' s suggested to design some legs for the product to provide such clearance.
- The microphone should be placed far away from the speaker and other objects that can produce noise or vibration, and be isolated and buffered by rubber pads from the speaker sound cavity.

2.2.3 Microphone Array Design Suggestion

Customers can design two or three microphones in an array:

- Two-microphone solution: the distance between the microphones should be 4 ~ 6.5 cm, the axis connecting them should be parallel to the horizontal line, and the center of the two microphones should be horizontally as close as possible to the center of the product.
- Three-microphone solution: the microphones are equally spaced and distributed in a perfect circle with the angle 120 °C from each other, and the spacing should be 4 ~ 6.5 cm.

There are some limitations when selecting microphones for the same array:

- Type: omnidirectional MEMS microphone. Use the same microphone models from the same manufacturer for the array. It' s not recommended to use different microphone models in the same array.
- The sensitivity difference of all the microphones in the same array should be within 3 dB.
- The phase difference of all the microphones in the same array should be within 10°.
- It is recommended to use the same structural design for all the microphones in the same array to ensure consistency.

2.2.4 Microphone Leakproofness Suggestion

Use plasticine or similar materials to seal the microphone pickup hole and compare how much the signals collected by the microphone decrease by before and after the seal. 25 dB is qualified, and 30 dB is recommended. Below are the test procedures:

1. Play white noise at 0.5 meters above the microphone, and keep the volume at the microphone 90 dB.
2. Use the microphone array to record for more than 10 s, and store the recording as recording file A.
3. Use plasticine or similar materials to block the microphone pickup hole, record for more than 10 s, and store it as recording file B.
4. Compare the frequency spectrum of the two files and make sure that the overall attenuation in the 100 ~ 8 kHz frequency band is more than 25 dB.

2.2.5 Echo Reference Signal Design Suggestion

- It is recommended that the echo reference signal be as close to the speaker side as possible, and recover from the DA post-stage and PA pre-stage.
- When the speaker volume is at its maximum, the echo reference signal input to the microphone should not have saturation distortion. At the maximum volume, the speaker amplifier output Total Harmonic Distortion (THD) is less than 10% at 100 Hz, less than 6% at 200 Hz, and less than 3% above 350 Hz.
- When the speaker volume is at its maximum, the sound pressure picked up by the microphone does not exceed 102 dB @ 1 kHz.
- The echo reference signal voltage does not exceed the maximum allowed input voltage of the ADC. If it is too high, an attenuation circuit should be added.
- A low-pass filter should be added to introduce the reference echo signal from the output of the Class D power amplifier. The cutoff frequency of the filter is recommended to be more than 22 kHz.
- When the volume is played at the maximum, the recovery signal peak value is -3 to -5 dB.

2.2.6 Microphone Array Consistency Verification

It is required that the difference between the sampled signals of each microphone in the same array is less than 3 dB. Below are the test procedures.

1. Play white noise at 0.5 meters above the microphone, and keep the volume at the microphone 90 dB.
2. Use the microphone array to record for more than 10 s, and check whether the recording amplitude and audio sampling rate of each microphone are consistent.

Chapter 3

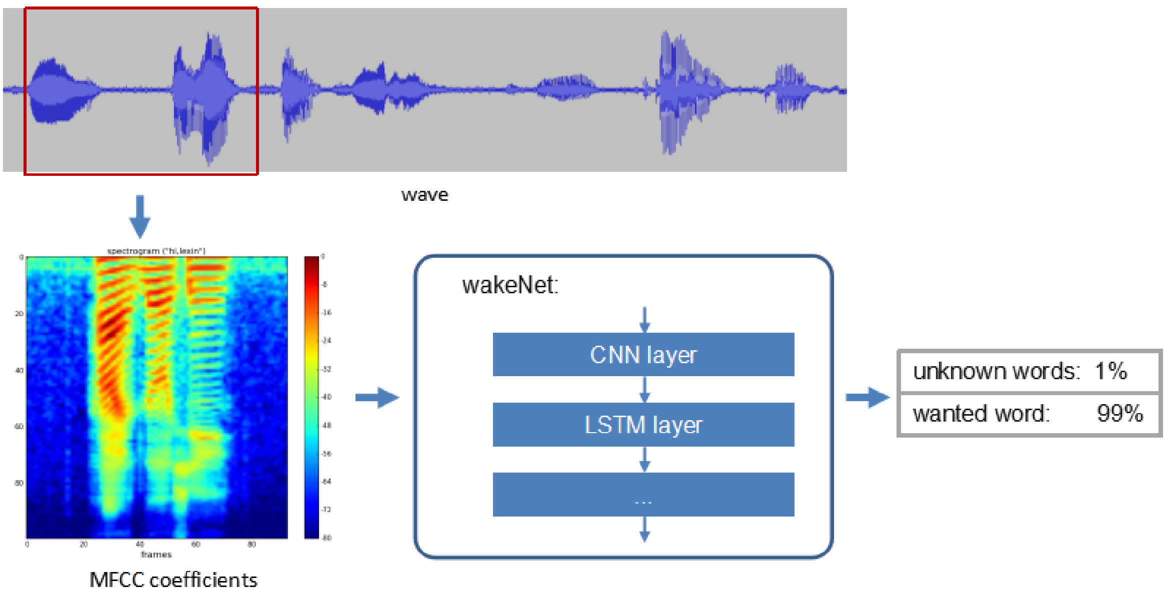
Wake Word

3.1 WakeNet Wake Word Model

WakeNet is a wake word engine built upon neural network for low-power embedded MCUs. Currently, WakeNet supports up to 5 wake words.

3.1.1 Overview

Please see the flow diagram of WakeNet below:



- **Speech Feature** We use MFCC method to extract the speech spectrum features. The input audio file has a sample rate of 16KHz, mono, and is encoded as signed 16-bit. Each frame has a window width and step size of 30ms.
- **Neural Network** Now, the neural network structure has been updated to the ninth edition, among which:
  - WakeNet1, WakeNet2, WakeNet3, WakeNet4, WakeNet6, and WakeNet7 had been out of use.
  - WakeNet5 only supports ESP32 chip.
  - WakeNet8 and WakeNet9 only support ESP32-S3 chip, which are built upon the Dilated Convolution structure.

The network structure of WakeNet5, WakeNet5X2 and WakeNet5X3 is the same, but WakeNetX2 and WakeNetX3 have more parameters than WakeNet5. Please refer to *Resource Consumption* for details.

- **Keyword Triggering Method:** For continuous audio stream, we calculate the average recognition results (M) for several frames and generate a smoothing prediction result, to improve the accuracy of keyword triggering. Only when the M value is larger than the set threshold, a triggering command is sent.

The wake words supported by Espressif chips are listed below:

Chip	ESP32			ESP32S3			
model	WakeNet 5			WakeNet 8		WakeNet 9	
	WakeNet 5	WakeNet 5X2	WakeNet 5X3	Q16	Q8	Q16	Q8
Hi,Lexin	√	√	√				√
nihaoxiaozhi	√		√				√
nihaoxiaoxin			√				
xiaoaitongxue							√
Alexa				√			√
Hi,ESP							√
Customized word							√

3.1.2 Use WakeNet

- Select WakeNet model

  To select WakeNet model, please refer to Section *Flashing Models* .

  To customize wake words, please refer to Section *Espressif Speech Wake-up Solution Customization Process*
- Run WakeNet

  WakeNet is currently included in the *AFE*, which is enabled by default, and returns the detection results through the AFE fetch interface.

  If users do not need WakeNet, please use:

```
afe_config.wakeNet_init = False.
```

If users want to enable/disable WakeNet temporarily, please use:

```
afe_handle->disable_wakenet(afe_data)
afe_handle->enable_wakenet(afe_data)
```

3.1.3 Resource Occupancy

For the resource occupancy for this model, see *Resource Occupancy*.

3.2 Espressif Speech Wake-up Solution Customization Process

3.2.1 Wake Word Customization Process

Espressif provides users with the **wake word customization** :

1. Espressif has already opened some wake words for customers' commercial use, such as "HI Leixi", or "Nihao Xiaoxin" .
   - For a complete list, see Table *Publicly Available Wake Words Provided by Espressif* .
   - Espressif also plans to provide more wake words that are free for commercial use soon.
2. Offline wake word customization can also be provided by Espressif:
   - Training corpus provided by customer
     – Customer must provide at least 20,000 qualified corpus entries. See detailed requirements in Section *Requirements on Corpus* .
     – It usually takes two to three weeks for Espressif to train and optimize the received corpus.
     – A fee will be charged for training and optimizing the corpus.
   - Training corpus provided by Espressif
     – Espressif provides all the corpus required for training.
     – The time required to collect corpus needs to be discussed separately. After the corpus is ready, it usually takes two to three weeks for Espressif to train and optimize the received corpus.
     – A fee will be charged for training and optimizing the corpus. A separate fee will be changed for collecting the corpus.
   - The actual fee and time for your customization depend on the **number of wake words you need** and the **scale of your mass production**. For details, please contact our sales person .
3. About Espressif wake word engine WakeNet:
   - Currently, up to 5 wake words are supported by each WakeNet model.
   - A wake word usually consists of 3 to 6 symbols, such as "Hi Leixin", "xiaoaitongxue", "nihaotianmao" .
   - More than one WakeNet models can be used together. However, more resource will be consumed when you use more models.
   - For more details, see Section *WakeNet Wake Word Model* .

3.2.2 Requirements on Corpus

As mentioned above, customers can provide Espressif with training corpus collected themselves or purchased from a third party. However, there are some limitations:

- Audio file format
  – Sample rate: 16 kHz
  – Encoding: 16-bit signed int
  – Channel: mono
  – Format: WAV

1. Sampling requirement
    - Number of samples: more than 500 people, including men and women of all ages and at least 100 children.
    - Sampling environment: a quiet room (< 40 dB). It is recommended to use a professional audio room.
    - Recording device: high-fidelity microphone.
    - **How to sample:**
        - At 1 meters away from the microphone: each person speaks the wake word out loud for 15 times (5 times in fast speed, 5 times in normal speed, 5 times in slow speed).
        - At 3 meters away from the microphone: each person speaks the wake word out loud for 15 times (5 times in fast speed, 5 times in normal speed, 5 times in slow speed).
    - File name: it is recommended to name the samples according to the age, gender, and quantity of the collected samples, such as `female_age_fast_id.wav`. Or you can use a separate file to present such information.

3.2.3 Hardware Design and Test

The voice wake-up performance heavily depends on the hardware design and cavity structure. Therefore, please pay special attention to the following requirements:

1. Hardware Design
    - Speaker designs: customers can make their own designs by modifying the reference designs (schematic/PCB) provided by Espressif. Also, Espressif can also review customers' speaker designs to avoid some common design issues.
    - Cavity structure: cavity should be designed by acoustic specialists. Espressif does not provide ID design reference. Customers can refer to other mainstream speaker cavity designs on the market, such as Tmall Genie, Xiaodu Smart Speaker, and Google Smart Speaker, etc.
2. Customers can perform the following tests to verify the hardware designs. Note that it's suggested to perform the following tests in a professional audio room. Customers can adjust the actual tests based on their actual testing environment.
    - Recording test to verify the gain and distortion of mic and codec
        - Play the sample (90 dB, 0.1 meter away from the mic), and adjust the gain to ensure that the recording is not saturated.
        - Use a sweep file of 0~20 KHz, and start recording using the sampling rate of 16 KHz. The recording should not have obvious frequency aliasing.
        - Record 100 samples, and feed these samples to open cloud voice recognition API. A certain recognition rate must be reached.
    - Playback test to verify the distortion of power amplifier (PA) and speaker
        - Test PA power @ 1% Total Harmonic Distortion (THD)
    - Speech algorithms test to verify the AEC, BFM and NS models
        - Adjust the delays of the reference signals based on the different requirements of different AEC algorithms.
        - Test the product based on the actual use scenario. For example, play `85DB-90DB Dreamer.wav` (a song) and record.
        - Analyze the processed signals to evaluate the performance of AEC, BFM, NS, etc.
    - DSP performance test to identify the correct DSP parameter and minimize the nonlinear distortion in the DSP algorithm
        - Noise Suppression
        - Acoustic Echo Cancellation
        - Speech Enhancement
3. Customers can also **send** 1 or 2 pieces of hardware to Espressif and ask us to optimize the product for better wakeup performance.

Chapter 4

Command Word

4.1 MultiNet Command Word Recognition Model

MultiNet is a lightweight model designed to recognize multiple speech command words offline based on ESP32-S3. Currently, up to 200 speech commands, including customized commands, are supported.

- Support Chinese and English speech commands recognition
- Support user-defined commands
- Support adding / deleting / modifying commands during operation
- Up to 200 commands are supported
- It supports single recognition and continuous recognition
- Lightweight and low resource consumption
- Low delay, within 500ms
- Support online Chinese and English model switching (esp32s3 only)
- The model is partitioned separately to support users to apply OTA

The MultiNet input is the audio processed by the audio-front-end algorithm (AFE), with the format of 16 KHz, 16 bit and mono. By recognizing the audio signals, speech commands can be recognized.

Please refer to *Models Benchmark* to check models supported by Espressif SoCs.

For details on flash models, see Section *Flashing Models* .

Note: Models ending with Q8 represents the 8 bit version of the model, which is more lightweight.

4.2 Commands Recognition Process

Please see the flow diagram for commands recognition below:

4.3 Speech Commands Customization Methods

Note: Mixed Chinese and English is not supported in command words.

The command word cannot contain Arabic numerals and special characters.

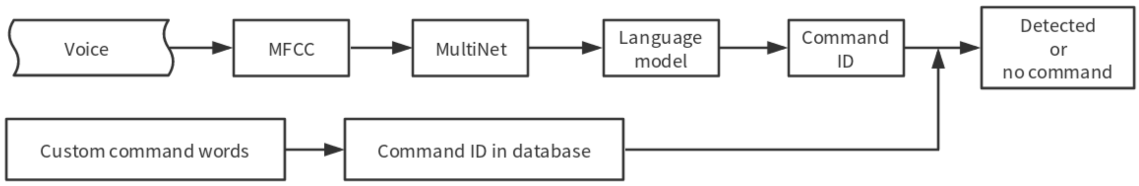


Fig. 1: speech_command-recognition-system

Please refer to Chinese version documentation for Chinese speech commands customization methods.

4.3.1 MultiNet7 customize speech commands

MultiNet7 use phonemes for English speech commands. Please modify a text file model/multinet_model/fst/commands_en.txt by the following format:

```
# command_id,command_grapheme,command_phoneme
1,tell me a joke,TfL Mm c qbK
2,sing a song,Sgl c Sel
```

- Column 1: command ID, it should start from 1 and cannot be set to 0.
- Column 2: command_grapheme, the command sentence. It is recommended to use lowercase letters unless it is an acronym that is meant to be pronounced differently.
- Column 3: command_phoneme, the phoneme sequence of the command which is optional. To fill this column, please use tool/multinet_g2p.py to do the Grapheme-to-Phoneme conversion and paste the results at the third column correspondingly (this is the recommended way).

If Column 3 is left empty, then an internal Grapheme-to-Phoneme tool will be called at runtime. But there might be a little accuracy drop in this way due the different Grapheme-to-Phoneme algorithms used.

4.3.2 MultiNet6 customize speech commands

MultiNet6 use grapheme for English speech commands, you can add/modify speech commands by words directly. Please modify a text file model/multinet_model/fst/commands_en.txt by the following format:

```
# command_id,command_grapheme
1,TELL ME A JOKE
2,MAKE A COFFEE
```

- Column 1: command ID, it should start from 1 and cannot be set to 0.
- Column 2: command_grapheme, the command sentence. It is recommended to use all capital letters.

The extra column in the default *commands_en.txt* is to keep it compatible with MultiNet7, there is no need to fill the third column when using MultiNet6.

4.3.3 MultiNet5 customize speech commands

MultiNet5 use phonemes for English speech commands. For simplicity, we use characters to denote different phonemes. Please use tool/multinet_g2p.py to do the convention.

- Via `menuconfig`
  1. Navigate to `idf.py menuconfig` > `ESP Speech Recognition` > `Add Chinese speech commands/Add English speech commands` to add speech commands. For details, please refer to the example in ESP-Skainet.

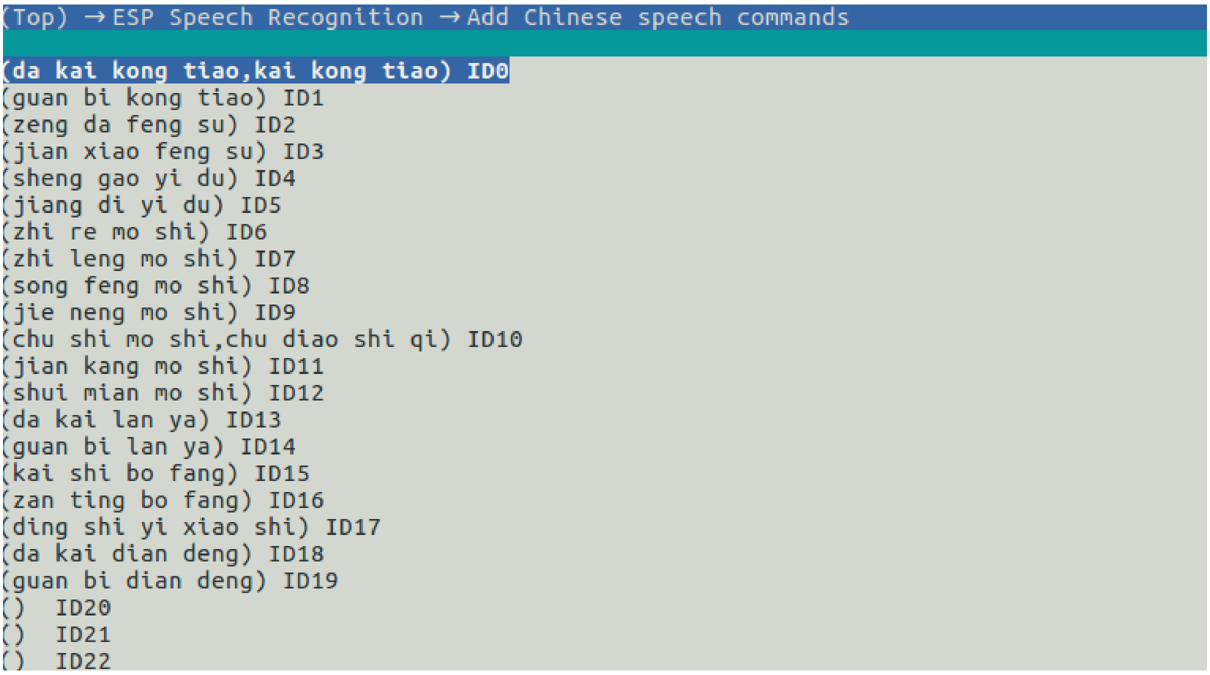


Fig. 2: menuconfig_add_speech_commands

Please note that a single `Command ID` can correspond to more than one commands. For example, "da kai kong tiao" and "kai kong tiao" have the same meaning. Therefore, users can assign the same command id to these two commands and separate them with "," (no space required before and after).

2. Call the following API:

```
/**
    * @brief Update the speech commands of MultiNet by menuconfig
    *
    * @param multinet              The multinet handle
    *
    * @param model_data            The model object to query
    *
    * @param langugae              The language of MultiNet
    *
    * @return
    *      - ESP_OK                    Success
    *      - ESP_ERR_INVALID_STATE    Fail
    */
    esp_err_t esp_mn_commands_update_from_sdkconfig(esp_mn_iface_t␣
↪*multinet, const model_iface_data_t *model_data);
```

4.3.4 Customize Speech Commands Via API calls

Alternatively, speech commands can be modified via API calls, this method works for MultiNet5, MultiNet6 and MultiNet7.

MutiNet5 requires the input command string to be phonemes, and MultiNet6 and MultiNet7 only accepts grapheme inputs to API calls.

- Apply new changes, the add/remove/modify/clear actions will not take effect util this function is called.

```
/**
* @brief Update the speech commands of MultiNet
```

(continues on next page)

(continued from previous page)

```
*
* @Warning: Must be used after [add/remove/modify/clear] function,
*           otherwise the language model of multinet can not be␣
↪updated.
*
* @return
*      - NULL                  Success
*      - others                The list of error phrase which can not be␣
↪parsed by multinet.
*/
esp_mn_error_t *esp_mn_commands_update();
```

Note: The modifications will not be applied, thus not printed out, until you call `esp_mn_commands_update()`.

- Add a new speech command, will return `ESP_ERR_INVALID_STATE` if the input string is not in the correct format.

```
/**
* @brief Add one speech commands with command string and command ID
*
* @param command_id      The command ID
* @param string  The command string of the speech commands
*
* @return
*      - ESP_OK                  Success
*      - ESP_ERR_INVALID_STATE   Fail
*/
esp_err_t esp_mn_commands_add(int command_id, char *string);
```

- Remove a speech command, will return `ESP_ERR_INVALID_STATE` if the command does not exist.

```
/**
* @brief Remove one speech commands by command string
*
* @param string  The command string of the speech commands
*
* @return
*      - ESP_OK                  Success
*      - ESP_ERR_INVALID_STATE   Fail
*/
esp_err_t esp_mn_commands_remove(char *string);
```

- Modify a speech command, will return `ESP_ERR_INVALID_STATE` if the command does not exist.

```
/**
* @brief Modify one speech commands with new command string
*
* @param old_string  The old command string of the speech commands
* @param new_string  The new command string of the speech commands
*
* @return
*      - ESP_OK                  Success
*      - ESP_ERR_INVALID_STATE   Fail
*/
esp_err_t esp_mn_commands_modify(char *old_string, char *new_string);
```

- Clear all speech commands.

```
/**
* @brief Clear all speech commands in linked list
*
```

(continues on next page)

(continued from previous page)

```
* @return
*      - ESP_OK                  Success
*      - ESP_ERR_INVALID_STATE   Fail
*/
esp_err_t esp_mn_commands_clear(void);
```

- Print cached speech commands, this function will print out all cached speech commands. Cached speech commands will be applied after `esp_mn_commands_update()` is called.

```
/**
* @brief Print all commands in linked list.
*/
void esp_mn_commands_print(void);
```

- Print active speech commands, this function will print out all active speech commands.

```
/**
* @brief Print all commands in linked list.
*/
void esp_mn_active_commands_print(void);
```

4.4 Use MultiNet

We suggest to use MultiNet together with audio front-end (AFE) in ESP-SR. For details, see Section *AFE Introduction and Use* .

After configuring AFE, users can follow the steps below to configure and run MultiNet.

4.4.1 Initialize MultiNet

- Load and initialize MultiNet. For details, see Section *flash_model*
- Customize speech commands. For details, see Section *Speech Commands Customization Methods*

4.4.2 Run MultiNet

Users can start MultiNet after enabling AFE and WakeNet, but must pay attention to the following limitations:

- The frame length of MultiNet must be equal to the AFE fetch frame length
- The audio format supported is 16 KHz, 16 bit, mono. The data obtained by AFE fetch is also in this format

- Get the length of frame that needs to pass to MultiNet

```
int mu_chunksize = multinet->get_samp_chunksize(model_data);
```

  `mu_chunksize` describes the `short` of each frame passed to MultiNet. This size is exactly the same as the number of data points per frame obtained in AFE.
- Start the speech recognition
  We send the data from AFE `fetch` to the following API:

```
esp_mn_state_t mn_state = multinet->detect(model_data, buff);
```

  The length of `buff` is `mu_chunksize * sizeof(int16_t)`.

4.4.3 MultiNet Output

Speech command recognition must be used with WakeNet. After wake-up, MultiNet detection can start.

Afer running, MultiNet returns the recognition output of the current frame in real time `mn_state`, which is currently divided into the following identification states:

- ESP_MN_STATE_DETECTING
  Indicates that the MultiNet is detecting but the target speech command word has not been recognized.
- ESP_MN_STATE_DETECTED
  Indicates that the target speech command has been recognized. At this time, the user can call `get_results` interface to obtain the recognition results.

  ```
  esp_mn_results_t *mn_result = multinet->get_results(model_data);
  ```

  The recognition result is stored in the return value of the `get_result` API in the following format:

  ```
  typedef struct{
  esp_mn_state_t state;
   int num;                // The number of phrase in list, num<=5. When␣
  ↪num=0, no phrase is recognized.
   int phrase_id[ESP_MN_RESULT_MAX_NUM];        // The list of phrase id.
   float prob[ESP_MN_RESULT_MAX_NUM];           // The list of probability.
  } esp_mn_results_t;
  ```

  where,
  - `state` is the recognition status of the current frame
  - `num` means the number of recognized commands, `num` <= 5, up to 5 possible results are returned
  - `phrase_id` means the Phrase ID of speech commands
  - `prob` means the recognition probability of the recognized entries, which is arranged from large to small

  Users can use `phrase_id[0]` and `prob[0]` get the recognition result with the highest probability.
- ESP_MN_STATE_TIMEOUT
  Indicates the speech commands has not been detected for a long time and will exit automatically and wait to be waked up again.

Single recognition mode and Continuous recognition mode: * Single recognition mode: exit the speech recognition when the return status is `ESP_MN_STATE_DETECTED` * Continuous recognition mode: exit the speech recognition when the return status is `ESP_MN_STATE_TIMEOUT`

4.5 Resource Occupancy

For the resource occupancy for this model, see *Resource Occupancy*.

Chapter 5

TTS Speech Synthesis Model

Espressif TTS speech synthesis model is a lightweight speech synthesis system designed for embedded systems, with the following main features:

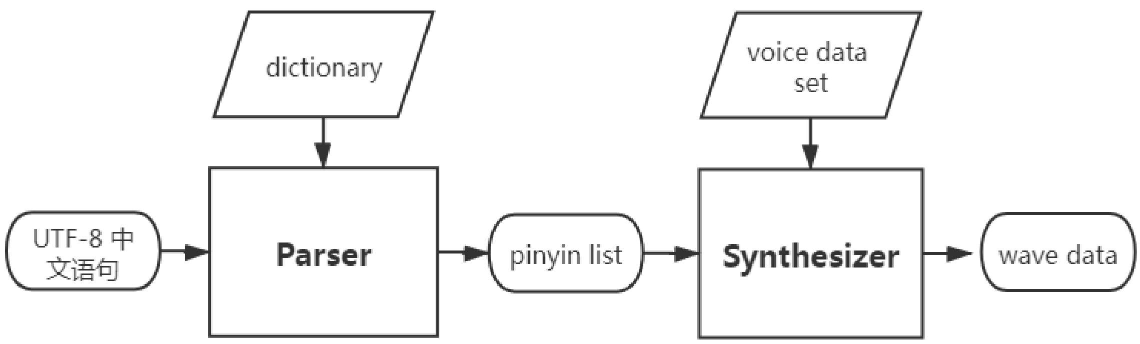
- Currently **Only supports Chinese language**
- Input text is encoded in UTF-8
- Streaming output, which reduces latency
- Polyphonic pronunciation
- Adjustable output speech rate
- Digital broadcasting optimization
- Customized sound set (coming soon)

5.1 Overview

Using a concatenative method, the current version of TTS includes the following components:

- Parser: converts Chinese text (encoded in UTF-8) to phonemes.
- Synthesizer: generates wave raw data from the phonemes provided by the parser and the sound set. Default output format: mono, 16 bit @ 16000 Hz.

Workflow:



5.2 Examples

- esp-tts/samples/xiaoxin_speed1.wav (voice=xiaoxin, speed=1): 欢迎使用乐鑫语音合成，支付宝收款 72.1 元，微信收款 643.12 元，扫码收款 5489.54 元

- esp-tts/samples/S2_xiaole_speed2.wav (voice=xiaole, speed=2): 支付宝收款 1111.11 元

5.3 Programming Procedures

```
#include "esp_tts.h"
#include "esp_tts_voice_female.h"
#include "esp_partition.h"

/*** 1. create esp tts handle  ***/


// initial voice set from separate voice data partition

const esp_partition_t* part=esp_partition_find_first(ESP_PARTITION_TYPE_DATA, ESP_
↪PARTITION_SUBTYPE_DATA_FAT, "voice_data");
if (part==0) printf("Couldn't find voice data partition!\n");
spi_flash_mmap_handle_t mmap;
uint16_t* voicedata;
esp_err_t err=esp_partition_mmap(part, 0, part->size, SPI_FLASH_MMAP_DATA, (const␣
↪void**)&voicedata, &mmap);
esp_tts_voice_t *voice=esp_tts_voice_set_init(&esp_tts_voice_template, voicedata);

// 2. parse text and synthesis wave data
char *text="欢迎使用乐鑫语音合成";
if (esp_tts_parse_chinese(tts_handle, text)) {  // parse text into pinyin list
    int len[1]={0};
    do {
        short *data=esp_tts_stream_play(tts_handle, len, 4); // streaming synthesis
        i2s_audio_play(data, len[0]*2, portMAX_DELAY);  // i2s output
    } while(len[0]>0);
    i2s_zero_dma_buffer(0);
}
```

See esp-tts/esp_tts_chinese/include/esp_tts.h for API reference and see the chinese_tts example in ESP-Skainet.

5.4 Resource Occupancy

For the resource occupancy for this model, see *Resource Occupancy*.

Chapter 6

Flashing Models

In the AI industry, a model refers to a mathematical representation of a system or process. It is used to make predictions or decisions based on input data. There are many types of models, such as decision trees, neural networks, and support vector machines, each with their own strengths and weaknesses. Esprssif also provides our trained models such as WakeNet and MultiNet (see the model data used in model)

To use our models in your project, you need to flash these models. Currently, ESP-SR supports the following methods to flash models:

ESP32-S3:

- Load directly from SIP Flash File System (flash)
- Load from external SD card

So that on ESP32-S3 you can:

- Greatly reduce the size of the user application APP BIN
- Supports the selection of up to two wake words
- Support online switching of Chinese and English Speech Command Recognition
- Convenient for users to perform OTA
- Supports reading and changing models from SD card, which is more convenient and can reduce the size of module Flash used in the project
- When the user is developing the code, when the modification does not involve the model, it can avoid flashing the model data every time, greatly reducing the flashing time and improving the development efficiency

6.1 Configuration

Run `idf.py menuconfig` to navigate to `ESP Speech Recognition`:

6.1.1 Model Data Path

This option indicates the storage location of the model data: `Read model data from flash` or `Read model data from SD card`.

- `Read model data from flash` means that the model data is stored in the flash, and the model data will be loaded from the flash partition
- `Read model data from SD card` means that the model data is stored in the SD card, and the model data will be loaded from the SD card

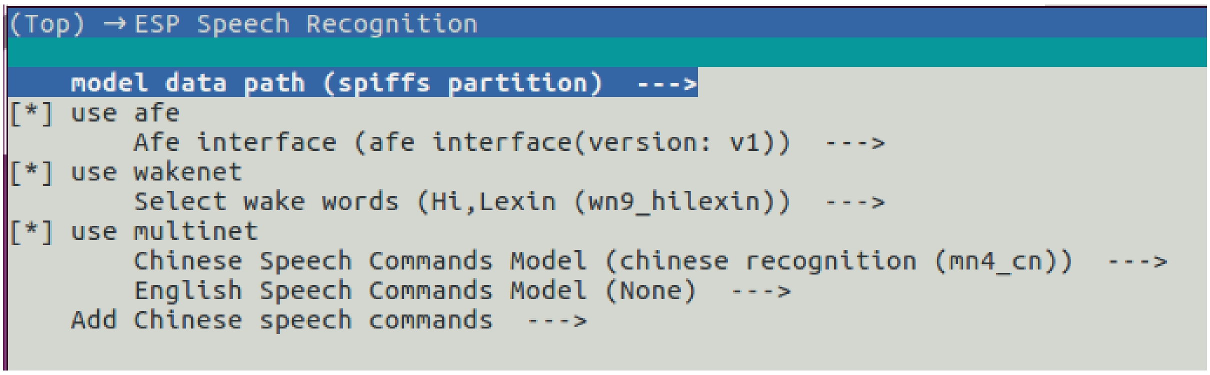


Fig. 1: overview

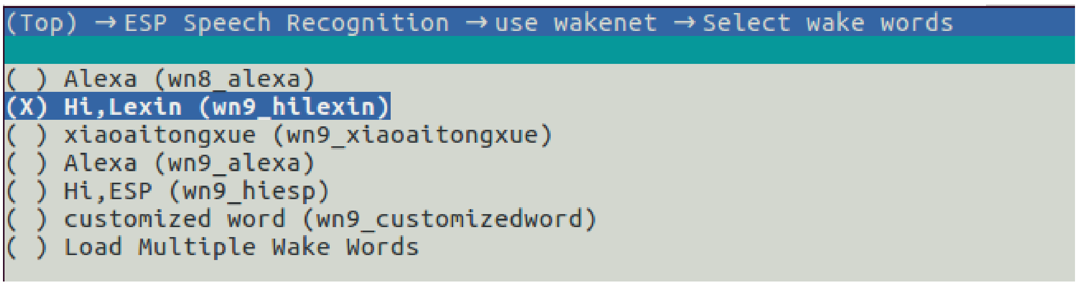
6.1.2 Use AFE

This option is enabled by default. Users do not need to modify it. Please keep the default configuration.

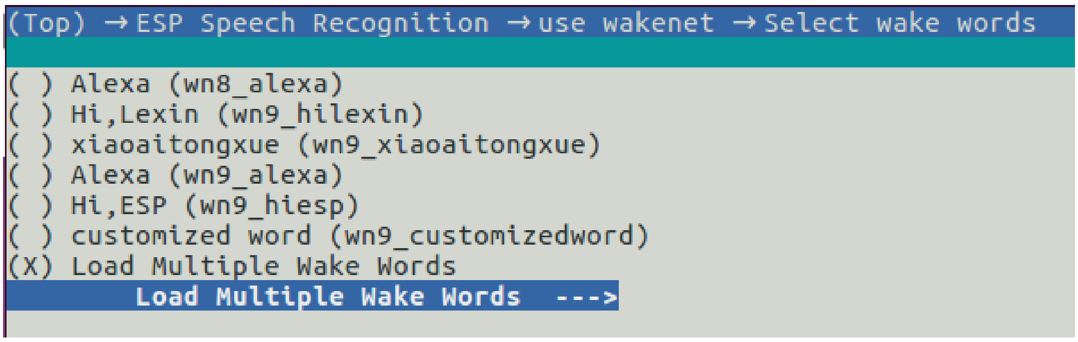
6.1.3 Use WakeNet

This option is enabled by default. When the user only uses `AEC` or `BSS`, etc., and does not need `WakeNet` or `MultiNet`, please disable this option, which reduces the size of the project firmware.

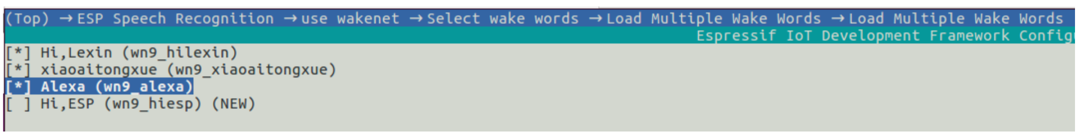
Select wake words by via `menuconfig` by navigating to `ESP Speech Recognition` > `Select wake words`. The model name of wake word in parentheses must be used to initialize WakeNet handle.



If you want to select multiple wake words, please select `Load Multiple Wake Words`



Then you can select multiple wake words at the same time:



Note: ESP32-S3 does support multiple wake words. Users can select more than one wake words according to the hardware flash size.

For more details, please refer to *WakeNet* .

6.1.4 Use Multinet

This option is enabled by default. When users only use WakeNet or other algorithm modules, please disable this option, which reduces the size of the project firmware in some cases.

Chinese Speech Commands Model

ESP32-S3 supports command words in both Chinese and English:

- None
- Chinese single recognition (MultiNet4.5)
- Chinese single recognition (MultiNet4.5 quantized with 8-bit)
- English Speech Commands Model

The user needs to add Chinese Speech Command words to this item when `Chinese Speech Commands Model` is not `None`.

For more details, please refer to Section *MultiNet* .

English Speech Commands Model

ESP32-S3 supports command words in both Chinese and English, and allows users to switch between these two languages.

- None
- English recognition (MultiNet5 quantized with 8-bit, depends on WakeNet8)
- Add Chinese speech commands

The user needs to add English Speech Command words to this item when `English Speech Commands Model` is not `None`.

6.2 How To Use

After the above-mentioned configuration, users can initialize and start using the models following the examples described in the ESP-Skainet repo.

Here, we only introduce the code implementation, which can also be found in model_path.c .

ESP32-S3 can load model data from flash or SD card.

6.2.1 Load Model Data from flash

1. Write a partition table:

   ```
   model,  data, spiffs,          , SIZE,
   ```

   Among them, `SIZE` can refer to the recommended size when the user uses `idf.py build` to compile, for example: `Recommended model partition size: 500K`
2. Initialize the flash partition: User can use `esp_srmodel_init(partition_label)` API to initialize flash and return all loaded models.
   - base_path: The model storage `base_path` is `srmodel` and cannot be changed
   - partition_label: The partition label of the model is `model`, which needs to be consistent with the `Name` in the above partition table

After completing the above configuration, the project will automatically generate `model.bin` after the project is compiled, and flash it to the flash partition.

6.2.2 Load Model Data from SD Card

When configured to load model data from `Read model data from SD card`, users need to:

- **Manually load model data from SD card** After the above-mentioned configuration, users can compile the code, and copy the files in `model/target` to the root directory of the SD card.
- **Initialize SD card** Users must initialize SD card so the chip can load SD card. Users of ESP-Skainet can call `esp_sdcard_init("/sdcard", num);` to initialize any board supported SD cards. Otherwise, users need to write the initialization code themselves. After the above-mentioned steps, users can flash the project.
- **Read models** User use `esp_srmodel_init(model_path)` to read models in `model_path` of SD card.

Chapter 7

Benchmark

7.1 AFE

7.1.1 Resource Consumption

Algorithm Type	RAM	Average cpu loading(compute with 2 cores)	Frame Length
AEC(LOW_COST)	152.3 KB	8%	32 ms
AEC(HIGH_PERF)	166 KB	11%	32 ms
BSS(LOW_COST)	198.7 KB	6%	64 ms
BSS(HIGH_PERF)	215.5 KB	7%	64 ms
NS	27 KB	5%	10 ms
MISO	56 KB	8%	16 ms
AFE Layer	227 KB		

7.2 WakeNet

7.2.1 Resource Consumption

Model Type	RAM	PSRAM	Average Running Time per Frame	Frame Length
Quantised WakeNet8 @ 2 channel	50 KB	1640 KB	10.0 ms	32 ms
Quantised WakeNet9 @ 2 channel	16 KB	324 KB	3.0 ms	32 ms
Quantised WakeNet9 @ 3 channel	20 KB	347 KB	4.3 ms	32 ms

7.2.2 Performance Test

Distance	Quiet	Stationary Noise (SNR = 4 dB)	Speech Noise (SNR = 4 dB)	AEC I nterruption (-10 dB)
1 m	98%	96%	94%	96%
3 m	98%	96%	94%	94%

False triggering rate: once in 12 hours

Note: In this test, we used ESP32-S3-Korvo V4.0 development board and WakeNet9(Alexa) model.

7.3 MultiNet

7.3.1 Resource Consumption

Model Type	Internal RAM	PSRAM	Average Running Time per Frame	Frame Length
MultiNet 4	16.8KB	1866 KB	18 ms	32 ms
MultiNet 4 Q8	10.5 KB	1009 KB	11 ms	32 ms
MultiNet 5 Q8	16 KB	2310 KB	12 ms	32 ms
MultiNet 6	32 KB	4100 KB	12 ms	32 ms
MultiNet 7	18 KB	2920 KB	11 ms	32 ms

7.3.2 Word Error Rate Performance Test

Model Type	librispeech test-clean	librispeech test-other
MultiNet5-en	16.5%	41.4%
MultiNet6-en	9.0%	21.3%
MultiNet7-en	8.5%	21.3%

7.3.3 Speech Commands Performance Test

Model Type	Distance	Quiet	Stationary Noise (SNR=5~10dB dB)	Speech Noise (SNR=5~10dB dB)
MultiNet 5_en	3 m	95.4%	85.9%	82.7%
MultiNet 6_en	3 m	96.8%	87.9%	85.5%
MultiNet 7_en	3 m	97.2%	92.3%	90.6%

7.4 TTS

7.4.1 Resource Consumption

Flash image size: 2.2 MB

RAM runtime: 20 KB

7.4.2 Performance Test

CPU loading test (ESP32 @240 MHz):

Speech Rate	0	1	2	3	4	5
Times faster than real time	4.5	3.2	2.9	2.5	2.2	1.8

Chapter 8

Test Method and Test Report

To ensure the DUT performance, some tests can be performed to verify the following parameters:

- Wake-up rate
- Speech recognition rate
- False wake-up rate
- Response Accuracy Rate Under Playback
- Response time

8.1 Test Room Requirement

These tests must be performed in a proper test room. The requirements for this test room include:

- **Size**
  - Area: no smaller than 4 m * 3.2 m
  - Height: no lower than 2.3 m
- **Setup**
  - The floor should be equipped with carpet, the ceiling should be equipped with common acoustic damping materials, and the wall should have 1 to 2 walls with curtains to prevent strong reflection.
  - Room reverberation time (RT60) within the range of [125, 8k] shall be within 0.2 - 0.7 seconds.
  - Do not use anechoic chamber.
- **Background noise**: must < 35 dBA, best < 30 dBA
- **Temperature and humidity**: 20±10°C, 50%±20%
- **Placement of DUT, external noise and voice**:
  - Place the DUT, external noise and voice according the actual use scenario of your DUT.

Note: The RT60, background noise, and the placement of DUT, external noise and voice should be kept the same in all tests.

8.2 Test Case Design

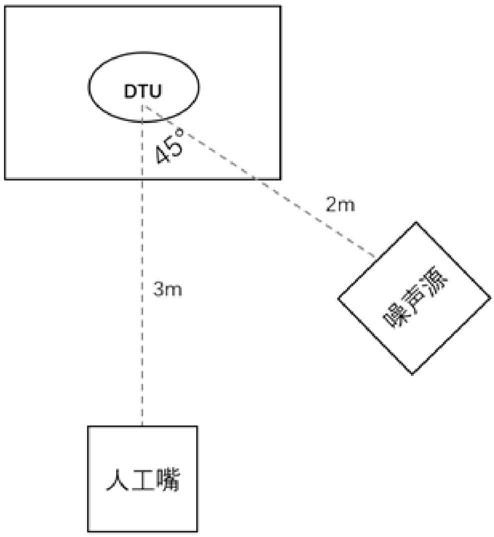
When designing test cases, it' s suggested to factor in **some or all of the following parameters** based on the actual use scenarios of the product. For example,

- **Different types of noises**
  - White noise
  - Human noise
  - Music

    - News
    - … …
    - Test cases with multiple noise sources can also be added when necessary
- **Different noise levels**
    - < 35 dBA
    - 45 dBA
    - 55 dBA
    - 65 dBA
- **Different voice levels**
    - 54 dBA
    - 59 dBA
    - 64 dBA
- **Different SNR**
    - 9 dBA
    - 4 dBA
    - -1 dBA

8.3 Espressif Test and Result

In all the tests described in this section, the placement of DUT, external noise and voice can be seen in the diagrams below.



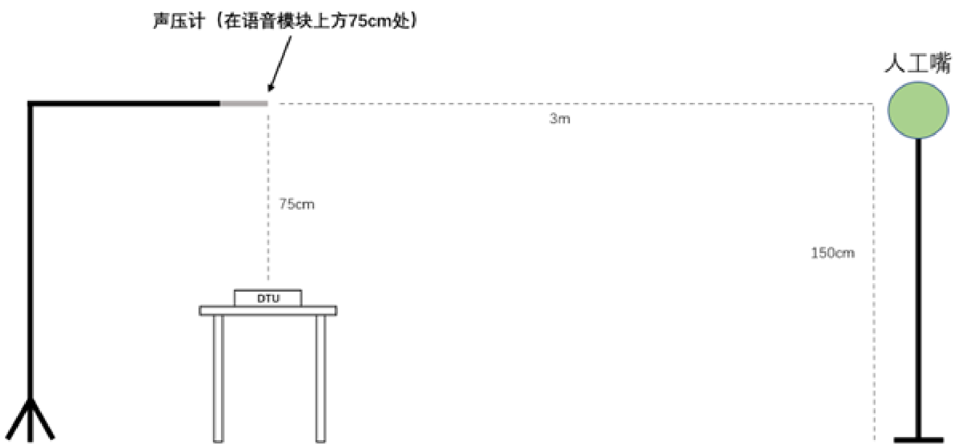
As seen in the diagrams above, place

- The DUT 0.75 meters above the ground.
- The voice 3 meters away from the DUT and 1.5 meters above the ground.
- The external noise 45°C apart from the voice, 2 meters away from the DUT and 1.2 meters above the ground.
- The sound pressure meter right above the DUT by 0.75 meters.

8.3.1 Wake-up Rate Test

Wake-up rate: the probability of the DUT correctly wakes up to a wake word.

Espressif' s Wake-up Rate Test and Result



Test Case	Noise Type	Noise Decibel	Voice Decibel	SNR	Wake-up Rate
1	/	/	59 dBA	/	99%
2	White noise	55 dBA	59 dBA	>= 4 dBA	99%
3	Human noise	55 dBA	59 dBA	>= 4 dBA	99%

8.3.2 Speech Recognition Rate Test

Speech recognition rate: the probability of the DUT correctly recognizes the established command words when the DUT is in the speech recognition state.

Espressif's Speech Recognition Rate Test and Result

Test Case	Noise Type	Noise Decibel	Voice Decibel	SNR	Speech Recognition Rate
1	/	/	59 dBA	/	91.5%
2	White noise	55 dBA	59 dBA	>= 4 dBA	78.25%
3	Human noise	55 dBA	59 dBA	>= 4 dBA	82.77%

8.3.3 False Wake-up Rate Test

False wake-up rate: the probability of the DUT incorrectly wakes up to a random word (that is not a wake word).

Espressif's False Wake-up Rate Test and Result

Test Case	Noise Type	Noise Decibel	Test Duration	Number of False Wake-up
1	Music	55 dBA	12 hours	1 time
2	News	55 dBA	12 hours	1 time

8.3.4 Response Accuracy Rate Under Playback

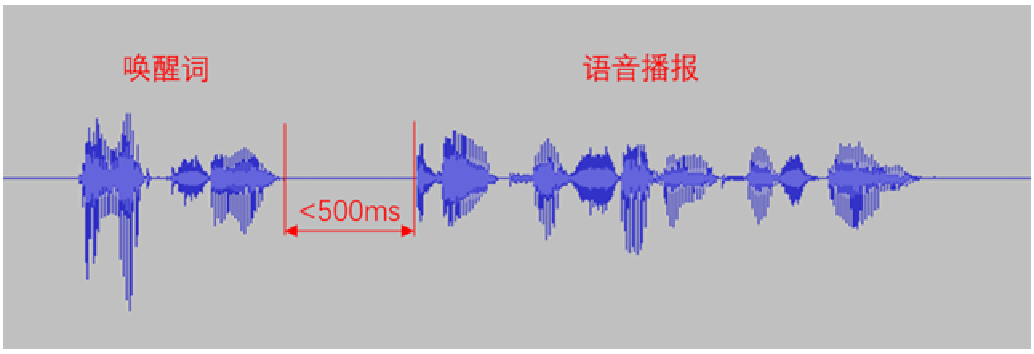
Interrupting wake-up rate: the probability of the DUT correctly responds to a wake word or a command word while playing sounds, such as music or TTS. This test is required for products with AEC feature.

Espressif's Interrupting Wake-up Rate Test and Result

Test Case	Noise Type	Noise / Voice Decibel	SNR	Wake-up Rate	Speech Recognition Rate
1	Music	69 dBA / 59 dBA	>= 10 dBA	100%	96%
2	TTS	69 dBA / 59 dBA	>= 10 dBA	100%	96%

8.3.5 Response Time Test

Response time: the time required for the DUT to respond to a command word. It’s measured as the time duration after a command word and before the DUT starts playing sound (see the diagram below).



Espressif’s Response Time Test and Result

Test Case	Noise / Voice Decibel	SNR	Response Time
1	NA / 59 dBA	/	< 500 ms

Chapter 9

Glossary

9.1 General Terms

ESP-SR reuses most of its terms in Espressif Audio Development Framework. See details in ADF English-Chinese Glossary .

9.2 Unique Terms

ESP-SR's unique terms are listed below.

Voice-User Interface (VUI) 语音用户界面